<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Secure Intellectual Property Management in Reconfigurable Computing Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Kepa, Krzysztof</td>
</tr>
<tr>
<td><strong>Publication Date</strong></td>
<td>2010</td>
</tr>
<tr>
<td><strong>Item record</strong></td>
<td><a href="http://hdl.handle.net/10379/4886">http://hdl.handle.net/10379/4886</a></td>
</tr>
</tbody>
</table>
Secure Intellectual Property Management in Reconfigurable Computing Systems

A Dissertation presented

by

Krzysztof Michał Kępa, M.Sc., Eng.

Electrical & Electronic Engineering
College of Engineering and Informatics
National University of Ireland, Galway

Submitted in fulfilment of the requirements for the
Degree of
Doctor of Philosophy in Electrical & Electronic Engineering

Supervisor
Dr. Fearghal Morgan

April 2010
I hereby declare that this thesis is my original work except where stated.

Signature:__________________________
Krzysztof Kęp
–April 2010
Acknowledgements

I dedicate this dissertation to Maga, for reminding me that there is much more to life than study, and to my parents, Rena and Leszek, for their unconditional love and support throughout my life. Thank you!

The work in this thesis was carried out under the superb supervision of Dr. Fearghal Morgan whom I thank particularly for his astute guidance and reassurance, as well as his endless energy, contagious enthusiasm and common sense. Ai.

I thank to Krzysztof Kościuszkiewicz for our stimulating conversations about informatics, life and everything. I also thank research and staff members of Electrical & Electronic Engineering in NUI Galway, research and staff members of Applied Optics Group at School of Physics in NUI Galway and staff members of Computer Architecture Group at the Institute of Computer Engineering, Control and Robotics in Wroclaw University of Technology.

I thank research and staff members of the Embedded Electronics Systems Group in the Institute for Information Processing Technology who hosted me at Karlsruhe Institute of Technology for two months. I also thank research and staff members of the Configurable Computing Laboratory in the Bradley Department of Electrical and Computer Engineering who hosted me at Virginia Tech in Blacksburg for two months.

I thank Dr. Michael Hübner, Prof. Jürgen Becker, Prof. Stanislaw Piestrak, Prof. Janusz Biernat, Dr. Jacek Majewski, Prof. Chris Dainty and recently passed Dr. Marian Bogdan for their advices and encouragement. I also thank Prof. Christophe Bobda who hosted me at Dagstuhl Workshops.

This project has been funded by the Irish Research Council for Science, Engineering and Technology (IRCSET) under the EMBARK Initiative. The equipment was provided by Xilinx under the Xilinx University Programme and Intel Communications Europe from Shannon, Ireland.

Numerous friends supported me in the life ‘outside the lab’: Anka, Bart, Charlie, Eugenie, Fabien, Grześ, Indiana, Ksenia, Karolina, Maro, Migle, Mindaugas, Oscar, Rafik, Sabine, Sedao, Stefan, Sylwek, Szakul, Tom, members of PolishSoc and Mountaineering Club at NUI Galway, and all my unconventional Galway flatmates. Thank you all.

Krzysztof Kępa, Wroclaw, Poland, April 2010
Abstract

This thesis contributes to Intellectual Property (IP) security and IP usage accounting in Partially Reconfigurable (PR) Xilinx Field-Programmable Gate Array (FPGA)-based Reconfigurable Computing (RC) systems.

The outsourcing of RC system design to external entities results in an extended, multi-player design environment and implicit chains of trust between various parties. A consequence is an increased risk to system integrity and to design IP protection, e.g. design IP theft, cloning, counterfeiting and tampering. Reported research on IP infringement countermeasures within RC devices has not considered the security risks caused by including erroneous or malicious IP cores in the PR-enabled RC system. Also, current security measures do not support system usage accounting and IP license enforcement in a multi-party design flow and in deployed PR-enabled RC systems. This hinders massive-scale adoption of third-party IP cores in high assurance RC systems.

This thesis proposes a new IP-aware method for the development of trustworthy PR systems and reports the implementation of a trusted Secure Reconfiguration Controller (SeReCon) IP core which is a Root of Trust (RoT) for RC systems. SeReCon provides design IP protection and maintains the integrity of the RC system by analysing the IP core structure prior to RC system reconfiguration and by mediating access to the internal Xilinx FPGA reconfiguration port. SeReCon protects the RC system from structural issues resulting from the inclusion of malicious IP cores. SeReCon supports the use of untrusted third-party IP cores in high-assurance RC systems. SeReCon also protects IP of third party designs and provides design IP license enforcement within the deployed system. The thesis proposes and describes a modification to the FPGA fabric to enable SeReCon security credentials to be generated and stored internally (within the FPGA) during the RoT certification process. In the SeReCon-based IP management scheme the Trusted Authority party participates only during certification of the RC system RoT. This policy reduces the risk of security credentials leakage and reduces the chain-of-trust requirements in a multi-player design flow.

The thesis also describes the development and application of the FPGA Design Analysis Tool (FDAT), which supports rapid prototyping of FPGA CAD applications for FPGA system-level design, design verification and application porting to SeReCon. FDAT provides a set of high-level FDAT APIs which abstract the Xilinx FPGA fabric, the implemented design (placed and routed netlist) and the associated FPGA configuration bitstream. The operation of FDAT is governed by “recipe” scripts and a lightweight graphic front-end.

A SeReCon-enabled RC system prototype has been implemented in a Xilinx Virtex-5 FPGA and targets a Software-Defined-Radio (SDR) application incorporating dynamically loadable IP cores. The RC system prototype includes a number of PR IP cores, e.g. AES cipher and decipher, in order to demonstrate the feasibility of the SeReCon-based IP management scheme. The SeReCon demonstrator application provides detailed and interactive insight into the operation of the RC system during SeReCon initialisation and operation, and illustrates that even genuine IP cores, when developed in multi-party environment, could include implicit communication channels and could therefore introduce security risks.
# Table Of Contents

Acknowledgements ........................................................................................................... v
Abstract ........................................................................................................................... vii
Table Of Contents ............................................................................................................. ix
List Of Figures ................................................................................................................ xii
List Of Tables ................................................................................................................ xiv
Glossary .......................................................................................................................... xvi

Chapter 1. Introduction ..................................................................................................... 1
  1.1. Thesis Summary ...................................................................................................... 1
  1.2. FPGA-based Reconfigurable Computing (RC) ...................................................... 4
  1.3. Partial Reconfiguration (PR) Design Flow ........................................................... 4
  1.4. Security Requirements In PR-Enabled RC Systems ............................................. 5
  1.5. Multi-Party Design Environment & RC System Life-Cycle ................................. 6
      1.5.1. Introduction ................................................................................................ 6
      1.5.2. Players In RC System Life Cycle ............................................................... 7
      1.5.3. Four Phase RC System Life Cycle ............................................................. 9
      1.5.4. RC System Stack Model ............................................................................ 9
  1.6. Illustration Of SeReCon Within A Software Defined Radio (SDR) Application ...... 12
  1.7. Security Issues In FPGA-Based PR RC Systems ............................................... 18
  1.8. FDAT: FPGA Design Analysis Tool ..................................................................... 19
  1.9. Structure Of The Thesis ....................................................................................... 21
  1.10. Thesis Contributions And Novelty Claims ......................................................... 23

Chapter 2. Partially Reconfigurable Computing Systems Background .............................. 26
  2.1. Introduction .......................................................................................................... 26
  2.2. RC Systems Vs ASICs And GPPs ....................................................................... 26
  2.3. FPGA Architecture And RC System Design Flow ............................................. 28
      2.3.1. Introduction .............................................................................................. 28
      2.3.2. FPGA Architecture Overview ................................................................. 28
      2.3.3. Xilinx FPGA Configuration Bitstream Structure ...................................... 29
      2.3.4. FPGA-Based RC System Design Flow ..................................................... 32
  2.4. Xilinx Partial Reconfiguration (PR) Design Flow .............................................. 34
      2.4.1. Introduction .............................................................................................. 34
      2.4.2. Xilinx Early Access PR Design Flow ....................................................... 34
      2.4.3. Xilinx PR Bitstream Structure .................................................................. 36

Chapter 3. Reconfigurable Computing Security: Background .......................................... 38
  3.1. Introduction .......................................................................................................... 38
  3.2. Security Risks In PR FPGA Systems: A Motivating Example .......................... 38
      3.2.1. Introduction .............................................................................................. 38
      3.2.2. CB4CLE Design Representation ............................................................... 39
      3.2.3. The Security Risk Of Implicit Communication Channel Between IP Cores . 40
      3.2.4. IP Core Implicit Communication Channel Over The Clock Line ............... 41
  3.3. State-Of-The Art In FPGA Design Security ....................................................... 44
      3.3.1. Introduction .............................................................................................. 44
      3.3.2. Side-Channel Attacks And Countermeasures .......................................... 46
      3.3.3. EDA Tools As The Security Threat ........................................................... 47
      3.3.4. Malicious FPGA Designs ........................................................................ 47
      3.3.5. Xilinx FPGA Fabric Protection ................................................................. 48
      3.3.6. Trusted Computing (TC) .......................................................................... 49
      3.3.7. RC System Integrity Maintenance ............................................................ 52
      3.3.8. Design IP Protection ............................................................................... 54
  3.4. FPGA IP Core Licensing Models ......................................................................... 57
Chapter 4. SeReCon Proposal: RoT and Usage Accounting In PR FPGAs ........................................ 64
 4.1. Introduction .................................................................................................................. 64
 4.2. Requirements Of Credentials Storage And Usage Accounting In A RoT .................. 64
 4.3. Review Of Techniques For Storage Of RoT Data ....................................................... 65
 4.4. FPGA Fabric Extension For Tamper-Proof Storage Of RoT Data ............................... 67
 4.4.1. Introduction .............................................................................................................. 67
 4.4.2. IDR Support For Tamper-Proof Storage Of RoT Credentials During Powerup .......... 69
 4.4.3. Extending IDR For System And IP Core Usage Accounting ................................. 70
 4.4.4. EIDR Prototype Implementation ............................................................................. 72
 4.5. Multi-Party RoT Certification Process ........................................................................ 76
 4.6. Chapter Summary ......................................................................................................... 77

Chapter 5. FDAT Framework For Low-Level FPGA Design Analysis ................................. 78
 5.1. Introduction .................................................................................................................. 78
 5.2. Review Of Low Level Design Analysis Tools ............................................................... 80
 5.2.1. Introduction .............................................................................................................. 80
 5.2.2. Low-Level Design Analysis Tools ......................................................................... 80
 5.2.3. Proposed Functionality For Low-Level FPGA Design Analysis Toolset ............... 81
 5.3. Proposed FPGA Design Analysis Tool (FDAT) ............................................................ 82
 5.3.1. Introduction .............................................................................................................. 82
 5.3.2. FPGA Fabric API (FFAPI) Module ........................................................................ 83
 5.3.3. Design API (DAPI) Module ................................................................................... 85
 5.3.4. Bitstream API (BAPI) Module ................................................................................ 87
 5.3.5. FDAT Function Recipes Exploiting A Script Programming Paradigm ................... 88
 5.4. FDAT Implementation And Analysis Of Virtex-II Pro Routing ................................... 91
 5.4.1. Introduction .............................................................................................................. 91
 5.4.2. Results Of FDAT Application In The Analysis Of Virtex-II Pro Routing ................. 92
 5.5. Porting Of FDAT Functionality To The Embedded SeReCon ...................................... 96
 5.5.1. Introduction .............................................................................................................. 96
 5.5.2. Requirements For ERDB Implementation .............................................................. 96
 5.5.3. Embedded Routing DataBase (ERDB) ................................................................. 97
 5.5.4. ERDB-Based IP Core Routing Analysis ................................................................. 102
 5.5.5. Verification Of The ERDB Correctness ................................................................. 106
 5.6. Chapter Summary ......................................................................................................... 108

Chapter 6. SeReCon Initialisation And Operation For Secure FPGA Reconfiguration .......... 110
 6.1. Introduction .................................................................................................................. 110
 6.2. SeReCon Internal State Diagram ................................................................................ 111
 6.3. SeReCon RoT Initialisation .......................................................................................... 112
 6.3.1. Introduction .............................................................................................................. 112
 6.3.2. Initialisation Algorithm ........................................................................................... 113
 6.3.3. Safelock - EIDR Support For IP Core Privacy & Integrity Protection ...................... 114
 6.4. IP Core Installation .................................................................................................... 116
 6.4.1. Introduction .............................................................................................................. 116
 6.4.2. Shared Key Agreement Between IPV And SeReCon .............................................. 116
 6.4.3. IP Core Production And Transfer To The RC System ............................................. 119
 6.4.4. IP Core Installation In The RC System ................................................................. 121
Appendix A – List of Reconfigurable Computing architectures ........................................... 195
Appendix B – EIDR source code (eidl.vhdl) ...................................................................... 196
Appendix C – ERDB C Data Structures (erd.h) ................................................................. 199
Appendix D – “BitPipsVerifier” recipe report ................................................................... 201
D.1. List of XDL pips which are not found in the Bitstream ............................................. 201
D.2. List of additional bitstream pips which are not found in the reference XDL file ........ 204
Appendix E – SeReCon debug console output during RC system demonstration ............ 206
E.1. Default view of the SeReCon debug console ........................................................... 206
E.2. EIDR initialisation ................................................................................................... 208
E.3. IP cores installation ............................................................................................... 209
E.4. IP cores activation ............................................................................................... 224
List Of Figures

1-1 Typical players, the four phase life cycle and the 3-layer stack model of a typical RC system.................................................................7
1-2 a. stack model of a self-reconfigurable system. b. risk of unprotected software access to the reconfiguration interface. c. the proposed SeReCon-enabled system stack model. including Root-of-Trust. d. SeReCon-based controlled reconfiguration........................................10
1-3 a. SDR environment and application implemented within a Partially-Reconfigurable (PR) FPGA-based Reconfigurable Computing (RC) device. b. potential attack vectors and the security assurances carried out by the proposed SeReCon element. .........................................................13
1-4 a. Block diagram of the FPGA-based SDR device (PR-enabled RC system) including SeReCon. b. SDR application software stack. c. SeReCon IP core. d. the Extended ID Register element. e. the SeReCon firmware installed in local memory. ........................................16
1-5 a. SeReCon-based Root-of-Trust (RoT) usage scenarios. b. RoT initialisation. c. IP core installation. d. IP core activation. ..17
1-6 FPGA Design Analysis Tool (FDAT) block diagram and context .................................................................20
2-1 Computational efficiency gap between silicon and microprocessors, expressed in Million Operations Per Second (MOPS) per Watt, widening as feature size reduces [39]. .........................26
2-2 Flexibility vs performance of processor classes [11]. .....................................................................................27
2-3 Block diagram of an FPGA device. .............................................................................................................29
2-4 Tile-based frame organisation of the configuration memory in the Xilinx Virtex-5 (V5LX50T) FPGA device. ........................................................................................................30
2-5 FPGA-based RC system Design Flow [49]. ..................................................................................................33
2-6 Xilinx EAPR design flow [10]. ..................................................................................................................35
3-1 Various (equivalent) representations of the CB4CLE counter design [55]. a. VHDL source. b. mapped netlist. c. configuration bitstream. d. FPGA Editor view of the P&R design netlist. e. FDAT view of the CLB tiles (red) and the P&R design netlist (blue) with separate routing tiles (yellow). ........................................................................................................39
3-2 Tile view of the Xilinx Virtex-5 FPGA with highlighted routing resources (turquoise) and CLBs (red). ........................................................................................................................................41
3-3 Communication channel existing outside two IP cores. This introduces the risk of setup of an implicit communication channel, which may introduce errors, system failure, or expose the IP to security risks...............................................................42
3-4 FDAT-generated visualisation of the abundant connectivity of the CB4CLE counter design. ab. within a Virtex 5 LXT FPGA. c.d. within a Virtex-2 Pro FPGA. The CB4CLE (blue and yellow) utilises a single CLB. CB4CLE signals use routing resources (green) which are shared by at least one electrical connection (a & c –without clock tiles, b & d – including clock tiles). ........................................................................................................43
3-5 Block diagram of the malicious IP core in a steganographic application................................................44
4-1 Block diagram of the FPGA fabric including the ID Register (IDR) element .................................................68
4-2 Block diagram of the ID Register (IDR) used for controlled generation of unique, random and partially-anonymous security credentials......................................................................................70
4-3 Extended ID Register block diagram. Monotonic-counters support enforcement of IP core license, time-limited and counted usage in a multi-party PR design environment.........................................72
4-4 VHDL description of the EIDR element. Implementation details are removed for clarity........75
4-5 Certification of the SeReCon Root-of-Trust (including SeReCon firmware binary). ..........................77
5-1 The available data structures and most significant FFAPi methods (the method parameters and less important methods are omitted to aid clarity). ......................................................... 84
5-2 FDAT screenshots. a. the Tile view of the Xilinx Virtex-5 device (colour highlights tiles of the same type). b. the user-design (blue — FPGA tiles used as logic, yellow — FPGA tiles used for routing) and unused resources (grey) within the Virtex-5 device. ........................................ 85
5-3 The available data structures and most significant DAPI methods (the method parameters and less important methods are omitted to aid clarity). ......................................................... 87
5-4 The available data structures and most significant BAPI methods (the method parameters and less important methods are omitted to aid clarity). ......................................................... 88
5-5 FDAT recipe structural organisation (left side) and interaction with FDAT modules, other recipes and user interface. ................................................................................................................. 89
5-6 Layered model of FDAT recipes. .............................................................................................. 91
5-7 "Pip2BitMapping" recipe flowchart. The recipe is used in the FDAT framework verification test. .................................................................................................................................................. 94
5-8 a. the bit mapping table generated for CLB column PIPs within the Virtex-II Pro. b. bit patterns for various PIP classes (used by FDAT to analyse partial-bitstreams and verify design spatial isolation). .................................................................................. 95
5-9 The FDAT "ErdbcGenerator" recipe flowchart. ........................................................................... 97
5-10 The FDAT GetRoutingShapes() function flowchart. .................................................................. 98
5-11 The CreateRoutingDB() function flowchart. The algorithm generates the ERDB PIPs database (ERDB_PD), the ERDB Routing database (ERDB_RD) and the ERDB Tilegroup database (ERDB_TG). ........................................................................................................... 100
5-12 The GeneratePipBitData() function flowchart. This algorithm exploits the "Pip2BitMapping" recipe in order to generate ERDB PIPs Bit database (ERDB_PB) which describes relative locations of PIP configuration bits in the bitstream. ................................................................ 101
5-13 The GenerateErdbc() function flowchart .............................................................................. 102
5-14 The FDAT "GetBitPips" recipe flowchart. This recipe reports all PIPs (including fake arcs) in the IP core. The recipe uses ERDB and FPGA configuration frames (obtained using the BAPI module)............................................................................................................. 104
5-15 The "GetExtWires" recipe flowchart. The recipe exploits the "GetBitPips" recipe and uses ERDB in order to detect and report IP core external routing which could be used to set implicit communication channels. ............................................................................. 105
5-16 Relation between the IP core region, the tile envelope and wire shape taps. .............................. 105
5-17 The "BitPipsVerifier" recipe flowchart which verifies accuracy of the PIP list generated using the ERDB-based "GetBitPips" recipe. .............................................................................................................. 107
6-1 The SeReCon internal state diagram. SeReCon RoT Initialisation ................................................. 112
6-2 The SeReCon RoT initialisation flowchart. .................................................................................... 114
6-3 The structure of the SafeLock backup file. .................................................................................... 116
6-4 The IP core installation flowchart and players. a. Shared Key (SK) negotiation between the IPV and SeReCon RoT. b. IP core preparation and transfer. c. IP core installation in the RC system. ................................................................. 117
6-5 The IPV algorithm flowchart for Diffie-Hellman (DH) Shared Key (SK) agreement. ................... 118
6-6 The SeReCon algorithm flowchart for DH SK agreement. ........................................................... 120
6-7 a. IP core preparation flowchart. b. the IP package...................................................................... 121
6-8 The flowchart of the IP core installation in the RC system. ........................................................ 122
6-9 The IP core isolation region and its relation with the IP core region. ........................................... 123
6-10 a. Structure of the IP core analysis report. The report defines IP core configuration and isolation regions. b. List of external PIPs in both regions. ................................................................. 123
6-11 Main steps in the IP core activation process. ................................................................. 124
6-12 The data structure which describes the current state of the RC system PR region. ........ 125
6-13 The IP core compatibility verification flowchart ............................................................. 126
6-14 The IP core license checking flowchart. a. SeReCon verifies the number of remaining IP core activations and usage lifetime prior to IP core activation. b. SeReCon updates IP license during unsuccessful IP core activation. c. SeReCon updates IP license during IP core deactivation. ................................................................. 127
6-15 The IP core activation flowchart ..................................................................................... 128
6-16 The SeReCon-based IP core deactivation process. ........................................................ 129
7-1 a. the block diagram of the SeReCon demonstrator. b. SeReCon-enabled RC system........ 132
7-2 Block diagram of the SeReCon-enabled RC system which is implemented in the Xilinx Virtex-5 FPGA (Xilinx Virtex-5 LXT FPGA ML505 Evaluation Platform). ............................................ 133
7-3 The Xilinx PlanAhead view of the SeReCon-enabled RC system top-level netlists. .......... 134
7-4 Xilinx ISE Schematic view of the SeReCon-enabled RC system ...................................... 135
7-5 The VHDL description of the modified PCIe reference design which includes the additional memory interface which is used for communication with the RC system. PCIe Endpoint-related signals are removed for clarity. ................................................................. 136
7-6 The VHDL description of the PCIe BAR Splitter IP core interface. .................................... 137
7-7 The VHDL description of the SeReCon-enabled RC system Configuration Controller IP core interface..................................................................................................................... 139
7-8 The VHDL description of the PR region interface. ............................................................ 141
7-9 The FPGA fabric with a size-constrained PR region and location-constrained Bus Macros (a view from the Xilinx PlanAhead tool). ................................................................. 142
7-10 The Xilinx EDK view of the SeReCon internal organisation ............................................. 146
7-11 VHDL description of the SeReCon IP core interface. ......................................................... 147
7-12 View of the SeReCon serial console which uses the UART interface in order to support RC system debugging. ........................................................................................................... 148
7-13 The block diagram of the SeReCon PCIe interface IP core ............................................... 150
7-14 The FPGA Editor view of the SeReCon-enabled RC system implementation in the Xilinx Virtex-5 FPGA and location of the PR region. a. FPGA logic occupation. b. complexity of RC system routing. .................................................................................................................. 156
7-15 The relative (percentage of FPGA resources) cost of the main SeReCon-enabled RC system prototype elements. a. FPGA LUT usage. b. FPGA FF usage. ......................................................... 157
7-16 The relative (percentage of FPGA resources) cost of the SeReCon elements. a. FPGA LUT usage. b. FPGA FF usage. ........................................................................................................... 159
7-17 a. the SDR device hardware prototype. b. SDR System-in-Package internals. (Source: VT CCM Lab) ........................................................................................................................................ 175
7-18 SDR block diagram and the suggested location of the SeReCon element ......................... 176
List Of Tables

2-1 Approximated resource density of current and future FPGA technologies [42] .................. 28
4-1 Resource occupation of the EIDR prototype implemented in Virtex-5 LXT. ...................... 74
4-2 The EIDR API which is provided by the SeReCon EIDR driver ........................................ 74
5-1 ERDB C source and header files produced by the “ErdbCGenerator” FDAT recipe which are used by the SeReCon firmware ................................................................. 102
5-2 Fragment of the "BitPipsVerifier" report which shows the difference between the list of XDL PIPs and a list of bitstream PIPs (for the ‘INT’ FPGA tile type) ........................................ 108
5-3 Correlation between the ERDB-based bitstream analysis and the reference XDL file, which is reported by the FDAT “BitPipsVerifier”recipe .................................................. 108
6-1 The SafeLock API which is provided by the SeReCon EIDR driver ................................... 115
6-2 IP core license modes and license restrictions which are supported by SeReCon .............. 121
7-1 Description of the PCIe interface BARs ................................................................. 137
7-2 Description of Configuration Controller registers which are mapped into the PCIe BAR1 .... 140
7-3 Description of the Adder interface registers .................................................................. 143
7-4 Description of the Multiplier register interface .......................................................... 143
7-5 Description of the AES Cipher register interface ........................................................ 144
7-6 Description of the AES Decipher register interface ..................................................... 145
7-7 Description of the SeReCon RC system interface signals which connect SeReCon to the Configuration Controller IP core and RC system BMs ........................................ 149
7-8 Description of the TRNG IP core registers which are mapped in the MicroBlaze memory and are available to the SeReCon firmware .................................................. 151
7-9 Description of the SeReCon AES IP core registers (continued in Table 7-10) which are mapped into the MicroBlaze memory and are available to the SeReCon firmware .................................. 152
7-10 Description of the SeReCon AES IP core registers (continuation of Table 7-9) ................ 153
7-11 FPGA logic resources used by the prototype SeReCon-enabled RC system .................. 156
7-12 FPGA logic resources used by the SeReCon IP core ................................................. 158
7-13 FPGA memory resources used by the SeReCon firmware ........................................... 160
7-14 The percentage resource cost of the SeReCon implementation in modern Xilinx FPGAs ........ 163
7-15 Description of the RC system communication library which is used by the host-side SeReCon API .......................................................... 165
7-16 Description of the host-side SeReCon API which is used by the RC system demonstrator application ........................................................................................................ 165
7-17 Description of the SeReCon demonstrator API which is tested using the interactive Python command line. The API description is continued in Table 7-18 and Table 7-19 ... 166
7-18 Continued description of the SeReCon demonstrator API which includes support for RC system interactive debugging ................................................................. 167
7-19 Continued description of the SeReCon demonstrator API which includes functional tests for activated IP cores and LEDs on the RC system board ................................... 168
**Glossary**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES</td>
<td>Triple DES</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific IC</td>
</tr>
<tr>
<td>BAPI</td>
<td>Bitstream API</td>
</tr>
<tr>
<td>BaseSFC</td>
<td>Base SeReCon FPGA Configuration</td>
</tr>
<tr>
<td>BAR</td>
<td>Block Address Range</td>
</tr>
<tr>
<td>BB</td>
<td>Base Band</td>
</tr>
<tr>
<td>BEL</td>
<td>Basic Element of Logic</td>
</tr>
<tr>
<td>BIOS</td>
<td>Basic Input Output System</td>
</tr>
<tr>
<td>BIT</td>
<td>Bitstream file</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CADBUS</td>
<td>Command, Address &amp; Data Bus</td>
</tr>
<tr>
<td>CB4CLE</td>
<td>4-bit binary counter</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher Block Chaining</td>
</tr>
<tr>
<td>CBT</td>
<td>Configuration Bit Template</td>
</tr>
<tr>
<td>CCL</td>
<td>Configuration Control Logic</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
</tr>
<tr>
<td>CM</td>
<td>Configuration Memory</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal–Oxide–Semiconductor</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial Off-The-Shelf</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>CSR</td>
<td>Control Status Register</td>
</tr>
<tr>
<td>DAPI</td>
<td>Design API</td>
</tr>
<tr>
<td>DCE</td>
<td>Data Communications Equipment</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DH</td>
<td>Diffie-Hellman</td>
</tr>
<tr>
<td>DPA</td>
<td>Differential Power Analysis</td>
</tr>
<tr>
<td>DRC</td>
<td>Design Rules Checker</td>
</tr>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
</tr>
<tr>
<td>DTE</td>
<td>Data terminal equipment</td>
</tr>
<tr>
<td>DWDDL</td>
<td>Double WDDL</td>
</tr>
<tr>
<td>EAPR</td>
<td>Early Access PR</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronic Design Automation</td>
</tr>
<tr>
<td>EDK</td>
<td>Embedded Development Kit</td>
</tr>
<tr>
<td>EIDR</td>
<td>Extended Identity Register</td>
</tr>
<tr>
<td>ERDB</td>
<td>Embedded Routing Database</td>
</tr>
<tr>
<td>FAE</td>
<td>Field Application Engineer</td>
</tr>
<tr>
<td>FAR</td>
<td>Frame Address Register</td>
</tr>
<tr>
<td>FASE</td>
<td>FPGA Architecture for Secure Embedded systems</td>
</tr>
<tr>
<td>FDAT</td>
<td>FPGA Design Analysis Tool</td>
</tr>
<tr>
<td>FDRI</td>
<td>Frame Data Input Register</td>
</tr>
<tr>
<td>FFAPI</td>
<td>FPGA Fabric API</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standard</td>
</tr>
<tr>
<td>Flash</td>
<td>non-volatile computer memory</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FRC</td>
<td>Free-Running Counter</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>FV</td>
<td>FPGA Fabric Vendor</td>
</tr>
<tr>
<td>GPIO</td>
<td>General Purpose Input Output</td>
</tr>
<tr>
<td>GPP</td>
<td>General-Purpose Processor</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HAIPES</td>
<td>High Assurance Internet Protocol Encryption</td>
</tr>
<tr>
<td>HAIPE</td>
<td>High Assurance Internet Protocol Encryption</td>
</tr>
<tr>
<td>HAIPIES</td>
<td>HAIPES Interoperability Specification</td>
</tr>
<tr>
<td>HMAC</td>
<td>Hash-based MAC</td>
</tr>
<tr>
<td>HPC</td>
<td>High-Performance Computing</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>ICAP</td>
<td>Internal Configuration Access Port</td>
</tr>
<tr>
<td>IDR</td>
<td>Identity Register</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IO (I/O)</td>
<td>Input Output</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IOB</td>
<td>IO Block</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>IPV</td>
<td>IP Vendor</td>
</tr>
<tr>
<td>ISE</td>
<td>Integrated Synthesis Environment</td>
</tr>
<tr>
<td>IVT</td>
<td>Isolation Verification Tool</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
</tr>
<tr>
<td>KR</td>
<td>Key Register</td>
</tr>
<tr>
<td>Linux</td>
<td>Unix-like free and open source OS</td>
</tr>
<tr>
<td>LIPS</td>
<td>Local IP Storage</td>
</tr>
<tr>
<td>LTC</td>
<td>Life Time Counter</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Table</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>MB</td>
<td>MicroBlaze</td>
</tr>
<tr>
<td>MFW</td>
<td>Multi-Frame Write</td>
</tr>
<tr>
<td>MIM</td>
<td>Man-in-the-Middle</td>
</tr>
<tr>
<td>MNA</td>
<td>Minor Address</td>
</tr>
<tr>
<td>MNC</td>
<td>MsgNo Counter</td>
</tr>
<tr>
<td>MOPS</td>
<td>Million Operations Per Second</td>
</tr>
<tr>
<td>NCD</td>
<td>Native Circuit Description</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>OO</td>
<td>Object-Oriented</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>P&amp;R</td>
<td>Place &amp; Route</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PCIe</td>
<td>PCI express</td>
</tr>
<tr>
<td>PHY</td>
<td>physical layer device</td>
</tr>
<tr>
<td>PIP</td>
<td>Programmable Interconnection Point</td>
</tr>
<tr>
<td>PMV</td>
<td>Process Monitor Vehicle</td>
</tr>
<tr>
<td>PR</td>
<td>Partial Reconfiguration</td>
</tr>
<tr>
<td>PRM</td>
<td>PR Module</td>
</tr>
<tr>
<td>PRNG</td>
<td>Pseudo RNG</td>
</tr>
<tr>
<td>PUF</td>
<td>Physically Unclonable Function</td>
</tr>
<tr>
<td>R&amp;D</td>
<td>Research &amp; Development</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-Access Memory</td>
</tr>
<tr>
<td>RC</td>
<td>Reconfigurable Computing</td>
</tr>
<tr>
<td>RE</td>
<td>Regular Expression</td>
</tr>
<tr>
<td>RoT</td>
<td>Root of Trust</td>
</tr>
<tr>
<td>RNS</td>
<td>Residue Number System</td>
</tr>
<tr>
<td>RNG</td>
<td>Random Number Generator</td>
</tr>
<tr>
<td>RO</td>
<td>Ring Oscillator</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest, Shamir and Adleman</td>
</tr>
<tr>
<td>RTR</td>
<td>Run-Time Reconfiguration</td>
</tr>
<tr>
<td>SAFES</td>
<td>Security Architecture For Embedded Systems</td>
</tr>
<tr>
<td>SCC</td>
<td>Single Chip Crypto</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SDR</td>
<td>Software Defined Radio</td>
</tr>
<tr>
<td>SDRAM</td>
<td>Synchronous Dynamic RAM</td>
</tr>
<tr>
<td>SECD-ED</td>
<td>Single-Error Correction Double-Error Detection</td>
</tr>
<tr>
<td>SeReCon</td>
<td>Secure Reconfiguration Controller</td>
</tr>
<tr>
<td>SEU</td>
<td>Single-Event Upset</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure Hash Algorithm</td>
</tr>
<tr>
<td>SI</td>
<td>System Integrator</td>
</tr>
<tr>
<td>SIP</td>
<td>System-in-Package</td>
</tr>
<tr>
<td>SoC</td>
<td>Systems-on-Chip</td>
</tr>
<tr>
<td>SODIMM</td>
<td>Small Outline Dual In-line Memory Module</td>
</tr>
<tr>
<td>SoPC</td>
<td>Systems-on-Programmable-Chip</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>TA</td>
<td>Trusted Authority</td>
</tr>
<tr>
<td>TC</td>
<td>Trusted Computing</td>
</tr>
<tr>
<td>TCG</td>
<td>Trusted Computing Group</td>
</tr>
<tr>
<td>TPM</td>
<td>Trusted Platform Module</td>
</tr>
<tr>
<td>TV</td>
<td>EDA Tool Vendor</td>
</tr>
<tr>
<td>TRNG</td>
<td>True RNG</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>Linux distribution</td>
</tr>
<tr>
<td>WDDL</td>
<td>Wave Dynamic Differential Logic</td>
</tr>
<tr>
<td>XDL</td>
<td>Xilinx Design Language</td>
</tr>
<tr>
<td>XDLRC</td>
<td>FPGA Fabric netlist file</td>
</tr>
<tr>
<td>VASSP</td>
<td>Virtual Application Specific Standard Product</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very-Large-Scale Integration</td>
</tr>
</tbody>
</table>
Chapter 1. Introduction

This introductory chapter first summarises the focus and elements of this thesis and outlines the motivations for performing this research. The chapter also includes an introduction to FPGA-based Reconfigurable Computing (RC) systems and the Partial Reconfiguration (PR) design flow. The various parties participating in a multi-party PR RC system design, implementation and application are described. Various security risks associated with a multi-party design flow are highlighted. A software Defined Radio (SDR) application implementation incorporating a PR RC element is considered in various usage scenarios to illustrate potential security risks. Finally, the chapter summarises the main contributions of the work.

1.1. Thesis Summary

This thesis contributes to Intellectual Property (IP) security and IP usage accounting in Partially Reconfigurable (PR) Xilinx Field-Programmable Gate Array (FPGA)-based Reconfigurable Computing (RC) systems.

RC systems typically include a number of IP cores, commonly designed by external parties. The high design complexity of RC systems, and pressure for reduced design time, have stimulated strong adoption of a design reuse methodology. This adoption is also supported by a growing development of third-party IP cores.

IP design outsourcing results in an extended, multi-player design environment, and therefore implicit chains of trust between various parties. Players include IP vendor, system integrator, user, trusted authority etc, typical in a software engineering domain. A consequence of the multi-player environment is an increased risk to system integrity and to design IP protection, e.g. design IP theft, cloning, counterfeiting and tampering [1]–[5]. Reported research on security measures within RC devices has not considered the security risks caused by including erroneous or malicious IP cores in the PR-enabled RC system, i.e. compromising system integrity through undiscovered IP design errors or malicious design overbuilds [2]. Current design IP protection methods focus on the confidentiality of the IP core implementation, mainly by using authentication and encryption protocols [6][7]. Adoption of an IP core protection and configuration model in a multi-party environment requires a modification of the configuration scheme for FPGA-based RC devices [8]. IP privacy protection and in-system license enforcement must be ensured to commercially available third-party IP core vendors. This may not be so where the system integrator has unrestricted access to, and is in full control of, all design modules including third-party IP cores. A new secure FPGA configuration model must be immune to IP core design errors and should allow design privacy protection, i.e. secure incorporation of third-party IP cores, without the need for disclosure of the IP implementation details.
Current IP infringement countermeasures and multi-player PR design flow do not support IP core usage accounting and license enforcement (e.g. time-limited, functionality/performance-limited or pay-per-use) in a multi-party design flow and in active (deployed) PR-enabled RC systems. An IP protection model should provide a reliable mechanism supporting IP core management according to a range of license restrictions. This approach hinders massive-scale adoption of third-party IP cores in high-assurance RC systems.

This thesis reviews security risks in RC, while focusing on FPGA-based RC systems using PR. State-of-the-art FPGA security measures are critically evaluated. The thesis proposes a new IP-aware method for the development of trustworthy PR systems and describes the implementation of a trusted Secure Reconfiguration Controller (SeReCon) IP core. SeReCon is a fixed-footprint, trusted reconfiguration base Root of Trust (RoT) for RC systems. SeReCon provides a trusted design environment, generates RC system security credentials and performs prior-to-reconfiguration IP core security verification and RC system self-reconfiguration.

A SeReCon prototype has been implemented in a Xilinx Virtex-5 FPGA and targets a Software-Defined-Radio (SDR) application incorporating dynamically loadable hardware radio modules (IP cores). SeReCon provides design IP protection and maintains the integrity of the RC system by analysing incoming FPGA reconfiguration requests during run-time, and by mediating all access to the internal Xilinx FPGA Internal Configuration Access Port (ICAP) [9]. Autonomous analysis of the structure of a new IP core prior to RC system reconfiguration verifies IP core spatial isolation and run-time protection of an already-configured PR system SeReCon interrupts the reconfiguration process if the IP core configuration violates the integrity of the RC system. This protects the RC system from structural issues resulting from erroneously placed (or malicious) IP cores. SeReCon enables the use of unverified (untrusted) third-party IP cores in high-assurance systems so long as they do not interfere with the active system configuration.

The thesis also proposes the use of SeReCon for IP core licensing and usage accounting, e.g., total runtime, no. of activations etc, in a PR system. SeReCon facilitates new IP core licensing models, e.g. transaction-based and metered access, during the PR system life-cycle. The SeReCon-based RoT supports license enforcement within the FPGA design flow, including the FPGA configuration bitstream in the target system. The SeReCon-based RoT and IP management scheme requires the participation of the Trusted Authority party only during certification of the RC system RoT. This reduces the chain-of-trust requirements in a multi-player design flow.

The thesis proposes and describes a modification to the FPGA fabric, the Extended ID Register (EIDR), which has been implemented in a Virtex-5 LXT device (ML505 Board) using the Xilinx ISE toolset. EIDR enables SeReCon security credentials (used for RC system identification) to be generated and stored internally (within the FPGA) during the system certification process. SeReCon employs authentication, public-key and symmetric-key cryptographic algorithms in order to protect the confidentiality and integrity of third party IP designs installed in the RC
system. Security credentials are generated using a Ring-Oscillator (RO)-based True Random Number Generator (TRNG), and remain within the SeReCon RoT security perimeter. This policy protects against un-authenticated access to the RC system credential storage and hence reduces the risk of leakage of security credentials.

The thesis also describes the development and application of the FPGA Design Analysis Tool (FDAT), which is a Python-based, versatile, modular and open tools framework for low-level analysis and verification of FPGA design bitstreams. FDAT supports rapid prototyping of EDA tools for FPGA system-level design, design verification and application porting to SeReCon. FDAT can be used as a trusted and verifiable reference design in the development, analysis and verification of PR designs targeting Xilinx FPGAs. FDAT extends the Xilinx design flow\(^1\) by providing a set of high-level FDAT APIs which abstract the Xilinx FPGA fabric, the implemented design (placed and routed netlist) and the related FPGA configuration bitstream. The operation of FDAT is governed by “recipe” Python-based scripts. A lightweight graphic front-end allows custom visualisation of the design within the FPGA fabric. To the best knowledge of the authors, FDAT is the first available toolset to provide high-level and unrestricted access to the low-level description of the Xilinx FPGA fabric and the user design at the netlist- and bitstream-level.

The prototype of the SeReCon-enabled RC system has been implemented in the Xilinx Virtex-5 LXT FPGA using the Xilinx EAPR design flow [10] and Xilinx EDA software, e.g ISE, EDK and PlanAhead tools. The RC system design files are included in the thesis DVD. The RC system prototype uses four IP cores, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher in order to demonstrate the SeReCon-based system reconfiguration. The SeReCon element is a CPU-based system uses an embedded 32-bit MicroBlaze processor operating at 125 MHz. Analysis of the SeReCon implementation resource usage confirms the feasibility of the SeReCon-based IP core management model in the largest FPGA-based RC systems.

The thesis reports the SeReCon demonstrator application which is implemented in Python and executed on the Intel server. The demonstrator application communicates with the RC system prototype (and SeReCon) using the PCIe interface and provides detailed insight into the operation of the RC system during the SeReCon (and EIDR) initialisation, IP core installation and activation. The demonstrator application shows that even genuine IP cores, when developed in multi-party environment, could include implicit communication channels and could introduce security risks.

---

\(^1\) The FPGA Design Analysis Tool (FDAT) is an extension for Xilinx EDA tools, developed only for educational purposes. The FDAT development process did not involve reverse-engineering of Xilinx proprietary tools. FDAT targets only Xilinx FPGA devices. FDAT does not support and does not enable any interoperability of Xilinx tools and FPGA devices from other FPGA vendors.
1.2. FPGA-based Reconfigurable Computing (RC)

Reconfigurable Computing (RC) is defined as: “the study of computation using reconfigurable devices” [11]. Thus, RC systems blur the boundary between hardware and software. RC systems offer the programmable flexibility of General-Purpose Processors (GPPs) at a fraction of the power consumption. RC systems provide high-performance computational acceleration in hardware, similar to that provided by Application Specific Integrated Circuits (ASICs). RC systems offer faster time-to-solution, cost reduction for small- to mid-volume applications, and improved fault-tolerance with respect to production defects. Also, RC systems leverage Intellectual Property (IP) R&D costs, while providing benefits usually associated with expensive High-Performance Computing (HPC) systems. Various technologies (see Appendix A) are used in RC systems, depending on the required reconfiguration granularity.

This thesis focuses on RC systems implemented using FPGA technology. RC hardware such as FPGAs provides a cost-attractive alternative to ASIC implementation for small- to mid-volume applications. The number of designs which use reconfigurable hardware is exploding, with applications ranging from embedded systems [12] to super-computers [13]. FPGA-based RC systems are extensively used for rapid prototyping, in-system and in-field customisation, multi-modal computation, and adaptive computing systems. Bobda has provided a survey of application domains which significantly benefit from the use of RC systems [11]. The list includes pattern matching, video streaming, Digital Signal Processing (DSP) using distributed arithmetic, adaptive controllers, adaptive cryptographic systems, Software Defined Radio (SDR) and HPC\(^2\).

1.3. Partial Reconfiguration (PR) Design Flow

The Run-Time Reconfiguration (RTR) paradigm enables RC systems to perform PR [14]. Active PR, available in some FPGAs, provides the flexibility of system reconfiguration during runtime. PR technology leads to an unprecedented flexibility and freedom in adapting to temporal changes within an RC system [15] or environmental adaptivity [16], [17]. PR in RC systems can occur not only at the software-level, but also at the configware-level [18]. Configware defines a virtualised hardware platform on which the software is executed.

Figure 1- presents the block diagram of a PR-enabled RC (FPGA-based) SDR device which includes the proposed SeReCon element. Typically, the SDR device contains a number of application specific radio-modules in the transmitter cores (TX region) and receiver cores (RX region). The SDR device also includes a number of additional interfaces, e.g. communication interface (COMM IF), external memory

controller (*MEM CTRL IF*), device-specific IO controller (*PERIPH IF*) and the FPGA Internal Configuration Access Port (ICAP) for self-reconfiguration. SeReCon is an additional IP core, and an element in the static FPGA base system configuration.

PR is offered by Atmel and Xilinx FPGA vendors[^3]. PR is facilitated in Xilinx FPGAs using the SelectMAP interface or the Internal Configuration Access Port (ICAP) [9]. PR provides full access to the FPGA configuration memory during system runtime. Self-reconfiguration using PR enables the RC system software layer to modify the underlying hardware configuration during system runtime, e.g. insertion of new computing modules (IP cores), without restarting the system [9].

### 1.4. Security Requirements In PR-Enabled RC Systems

A recent Department of Defense report [19] has identified several trends contributing to the threat of covert insertion of circuitry into computing hardware. Modifying hardware provides attackers with a fundamental advantage over software-based attacks [20], [2]. Attacks at the hardware level are more difficult to detect and to prevent than attacks on software. Also, defending against hardware intrusion is more difficult, as the offender has control over all system layers, including the software stack. This thesis reviews security risks in RC systems, while focusing on FPGA-based RC systems using PR. State-of-the-art FPGA security measures are critically evaluated.

PR in RC systems introduces risks to hardware system security (design integrity) on a scale associated to date only with the software domain. Risks exist such as covert insertion of circuitry into PR computing hardware (system tampering), or creation of implicit communication channels between IP cores. Modifying hardware provides attackers with a fundamental advantage over software-based attacks; attacks at the hardware level are more difficult to detect and prevent than attacks on software. Extending hardware support for security verification (intrusion detection) and handling is therefore required.

In FPGA-based RC systems, no protection layer exists below the system hardware layer. Without protective measures, PR FPGAs could be exposed to a range of attacks, some requiring the addition of only a small amount of covert-inserted hardware [20]. King [2] illustrates that an attacker can design hardware to support multiple attacks and demonstrates this concept using a system implemented in an FPGA.

The most strict adversary model in embedded system design assumes that a security risk exists where a device is held by one entity (system user) and where secrets (design IP) within the device are controlled by another entity (IP

[^3]: Also, Altera announced it will support PR in future 28-nm FPGA devices (http://www.eetimes.com/news/latest/showArticle.jhtml?articleID=222600544)
vendor/system integrator). A goal in secure systems engineering is to design a system which an attacker (e.g. user) cannot subvert, either by malice, accident, or trickery. Figure 1- and Figure 1-b illustrate possible attack vectors on a PR FPGA-based SDR device, and the security countermeasures provided by the embedded SeReCon.

While the methodology of IP core reuse reduces design time and associated cost, the intensive growth of the market for pre-designed IP modules introduces concerns about the protection of design IP rights and the integrity of designs incorporating third party IP cores. Ideally, each of the design components should be formally specified, tested, and verified, followed by certification by an external Trusted Authority (TA). In reality, IP components are typically created through in-house design reuse, obtained from third-party IP Vendors\(^4\), or generated using automated core generation tools, e.g. Xilinx CoreGen\(^5\). Current design IP protection methods focus on the confidentiality of the IP core implementation, mainly by using authentication and encryption protocols, though without considering the risks caused by including erroneous or malicious IP cores in the PR-enabled RC system [20], [21]. This approach hinders massive-scale adoption of third-party IP cores in high assurance RC systems. Also, design reuse in the RC system design flow results in IP cores of increasing complexity. As a consequence, attack methods can be generalised and are becoming obscure by the complexity of the RC system.

1.5. Multi-Party Design Environment & RC System Life-Cycle

1.5.1. Introduction

This section describes the typical players, the four phase life cycle and the 3-layer stack model of a typical RC system (Figure 1-). Trust between players is limited. The interactions between the various parties introduce multi-level risks associated with the design flow and the RC system itself. IP and EDA tool vendors seek appropriate IP protection against unauthorised design cloning, overbuilding (manipulation) and reverse engineering. System integrators (design houses and manufacturers) seek automated methods to provide effective system security to protect design integrity in the field.


\(^5\) CoreGen is included in Xilinx ISE.
1.5.2. Players In RC System Life Cycle

This section describes each player in a multi-party design environment (Figure 1-).

**Trusted Authority (TA)** is an authorisation and/or certification centre. The TA mediates communication between players in order to provide the required element of trust. The TA is assumed to be trustworthy by all other entities and is usually not involved in the system development process. The centralised role of the TA makes the TA an attractive target to attackers. A risk exists that the TA operation, trustworthy during RC system deployment, can be compromised in the future, e.g. by a successful attack on the TA facilities, leakage of sensitive data, bankruptcy etc. For RC systems using the proposed SeReCon element, the TA confirms the

---

authenticity of the RC system key generation process (during the initial system start-up), and certifies the resulting key material.

**EDA Tool Vendor (TV)** provides software tools to the other parties and strives to ensure software quality. The TV can develop a strong reputation, based on long-term trusted activity, but cannot be trusted entirely [21].

**IP Vendor (IPV)** is an external entity which provides reusable components (IP cores) for the RC system. The IPV wishes to protect its own design secrets. The IPV is not directly involved in the RC system design process, and is only aware of the system requirements which the IP core must meet. The IPV guarantees compliance of the IP design to the functionality specification, but not the optimal design. Competitive market (e.g. constrained deadlines, cost etc) can drive IPV to provide poorly tested or otherwise sub-optimal design. Similar to EDA software an IP core can contain hidden malicious functionality [4].

**The End-System User (User)** is an end-customer who operates the RC system, possibly in hostile environments. The User requires the system to be secure, but could also try to gain personal profit by attempting to circumvent the implemented security countermeasures since the User has physical access to the RC system. Also, while software can be erased at the end of the system life-cycle, the hardware platform often remains intact. A hardware reverse-engineering (de-packaging and fabric analysis) or recycling process can reveal some sensitive data, e.g. permanently embedded encryption keys. Under certain circumstances, even volatile memory can retain data [22], [23]. Therefore, recovery of sensitive data (e.g. keys) or algorithms may be possible.

**System Integrator (SI)** designs, manufactures and provides the RC system to the User. The SI can also issue a product upgrade in the field. A typical RC system consists of custom elements and multiple third-party IP cores. IP cores can be distributed in various formats: HDL source, netlist or FPGA device-specific (partial-) bitstream, depending on the level of trust between the SI and IPV. The SI is interested in optimising costs associated with external IP cores used in the RC system, e.g. royalties, license fees etc. Thus, the SI is motivated to circumvent legal license restrictions by e.g. IP core cloning, reverse-engineering or tampering.

**FPGA Fabric Vendor (FV)** provides the FPGA fabric. Modern FVs are fabless. Risks involved with outsourcing Integrated Circuit (IC) fabrication are detailed in [19]. Usually the FV maintains the implementation details of the fabric confidential, and guarantees quality of service and compliance to the FPGA specification.

The FV or its outsourced (typically off-shore) fabrication sub-contractor are interested in extensive fabric testing and debugging to aid silicon yield. Thus, the FPGA fabric includes undocumented vendor-specific debugging and test facilities, e.g. Xilinx Process Monitor Vehicles (PMVs) [24]. Typically, vendor-specific control circuits are not available to the User. This can lead to a device security breach when user access is left unprotected in production devices [25]. Also, the fabrication sub-contractor can weaken the fabric or inject additional malicious
functionality into the FPGA device\textsuperscript{7} [19], [20]. Verification by the TA is required to ensure that undocumented access to the FPGA internals (e.g. configuration memory) is not possible. This guards against Trojan ICs [20]. In the proposed SeReCon model, the FPGA fabric is assumed to be trusted.

1.5.3. Four Phase RC System Life Cycle

This section describes each phase of the RC system life cycle in a multi-party design environment (Figure 1-).

IC fabrication: delivers the FPGA fabric. The FV uses tools from the TV to design the chip. The major FVs (e.g. Altera, Lattice, Xilinx) are fables; fabrication of the FPGA devices is typically outsourced to the external foundry, e.g. TMSC\textsuperscript{8}. Thus, the TA can be required in order to establish trust in the produced FPGA devices (device certification) [26].

System design: delivers the RC device. The SI develops the RC system prototype, hardware platform and software. The SI uses EDA tools and reuses third party IP cores provided by the IPV. The TA can be used as a trusted third party during the process, when the SI and IPV do not trust each other.

System maturity: the main phase in the system life cycle. The device is installed in the field and is operated by the User. The User can extend the functionality of the device by ordering additional IP cores from the IPV or SI. The TA can also participate.

Hardware disposal: terminates the RC system life cycle. The device is deactivated and its hardware is ready for reuse. This phase can be intentional, e.g. scheduled service termination, or can result from a tampering attack. Thus, it is vital that the FPGA provides a facility for instant and irreversible system sanitisation, e.g. one-shot encryption key erasing. Some Xilinx FPGAs already support this feature as a ‘KEY_CLEAR’ primitive [27]. The KEY_CLEAR element provides the facility supporting immediate erase of the configuration encryption circuit key register from the internal logic.

1.5.4. RC System Stack Model

Figure 1-a illustrates the stack model of a typical self-reconfigurable system and highlights the risks of unprotected software access to the reconfiguration interface (Figure 1-b). Figure 1-c illustrates the proposed SeReCon-enabled self-reconfigurable system stack model. Figure 1-d illustrates SeReCon-based controlled (secure) reconfiguration.

\footnotesize

\textsuperscript{7} In 2008 US and Canadian agencies seized counterfeit Cisco gear \url{http://www.thestandard.com/news/2008/02/29/us-canadian-agencies-seize-counterfeit-cisco-gear}

\textsuperscript{8} Taiwan Semiconductor Manufacturing Company Limited (TSMC, \url{http://www.tsmc.com})
Figure 1 - a. stack model of a self-reconfigurable system. b. risk of unprotected software access to the reconfiguration interface. c. the proposed SeReCon-enabled system stack model. including Root-of-Trust. d. SeReCon-based controlled reconfiguration.

The FPGA-based RC system comprises three layers, namely hardware, configware and software.

The hardware layer is the physical device which contains the FPGA fabric with the Internal Configuration Access Port (ICAP), peripheral ICs (memory and interface PHYs), power supply etc. The hardware platform remains unchanged during the RC system life cycle. An attacker can disassemble the device, gaining access to system interfaces in order to eavesdrop and manipulate RAM, communication or configuration data. Drimer et al. have demonstrated successful tampering with secure PIN entry devices [28] and protocols [29]. Thus, it is vital to limit the RC system security boundary only to the tamper-proof FPGA chip and to facilitate RC system integrity monitoring [30].

Configware provides virtualisation of the RC hardware platform to the system application software. The capacity of modern FPGAs supports the implementation of complex Systems-on-Chip (SoCs) within a single FPGA [31], [32]. Complex SoCs incorporate a number of (third-party) application-specific IP cores (Figure 1-a and Figure 1-a). In some SoCs, configware also provides access to the ICAP. This allows system self-reconfiguration (IP cores exchange) using PR. When the ICAP is used, the configware (and software) has unlimited access to the FPGA configuration memory. When bitstream encryption is used, the ICAP remains enabled [9], [33]. Unprotected access to ICAP can lead to covert (undetected by software) change of
the SoC configuration or the installation of a Trojan IP core [2], [5]. The proposed SeReCon protects access to the ICAP and analyses the IP core content prior to reconfiguration, interrupting the reconfiguration process if the IP core configuration violates the integrity of the RC system.

The software layer contains RC system firmware and possibly the OS or application specific code. Firmware provides hardware and configware drivers to the RC application and system maintenance procedures, e.g. IP core load, unload and configuration routines. If the software layer has some exploitable vulnerability this allows the attacker to gain full control over the configware and thus the entire RC system. The proposed SeReCon performs PR on behalf of the software. This ensures that the requested PR does not overwrite the active part of the system, and that it limits the RC system security perimeter to the SeReCon IP core, ICAP and the FPGA configuration logic.

Design outsourcing of digital circuits and systems to third-party vendors and open source communities (e.g. OpenCores\(^9\)) gives rise to a network of trust relationships between parties and toolsets. This multi-party arrangement leads to increased security risks [21]. Individual IP cores are typically verified, and then certified as secure. Controlling the consistency of a complete design flow, from the system specification through IP core integration and deployment of the final product, becomes more challenging when reconfigurable systems and a PR flow are considered. PR introduces a risk of replacing a part of the design with malicious (Trojan-horse) functionality and thus a risk of compromising the entire system [2]. During self-reconfiguration, illegal connections (called covert channels) can potentially be set up, allowing for unwanted ad-hoc interactions between IP cores. This introduces the risk to IP ‘secrets’, e.g. data interception, algorithm theft, encryption key extraction etc.

\(^9\) OpenCores (http://opencores.org)
1.6. Illustration Of SeReCon Within A Software Defined Radio (SDR) Application

A SeReCon prototype demonstrator has been implemented in a Xilinx Virtex-5 FPGA and targets a Software-Defined-Radio (SDR) application incorporating dynamically loadable hardware radio modules (IP cores). SeReCon is a trusted element within the SDR device, which performs secure system reconfiguration. In this section, a SDR application\textsuperscript{10} implementation incorporating a PR RC system is considered in various usage scenarios to illustrate potential security risks. Farrel et al. overview the SDR demonstrators and future trends [34].

Figure 1-a illustrates a typical SDR environment and SDR application. The SDR Data Communications Equipment (DCE) is implemented in the RC system which includes radio interface (\textit{radio PHY}), the PR FPGA device and communication IOs (\textit{peripherals}). The RC device also incorporates the proposed embedded Secure Reconfiguration Controller (SeReCon) element. SDR DCE provides a radio communication link between the user Data Terminal Equipment (DTE), which might be also part of the SDR DCE device, and the host DCE/DTE system. The FPGA device is configured with the proposed embedded SeReCon element and a number of third-party digital IP cores, e.g. transceiver processing pipelines (TX, RX) which communicate with radio PHY and IO peripherals. IP cores are used e.g. for the Base Band (BB) signal processing, signal modulation/demodulation, data control and error correction etc.

The SDR User operates the SDR RC device in the field. The SDR User can request application-specific IP cores from the third party IP vendor. Third party IP cores are used, i.e. for proprietary BB signal processing etc. Third party IP cores are delivered to the FPGA through the unsecured channel, e.g. internet, radio link etc. This introduces a number of security risks for the SDR system\textsuperscript{11} (Figure 1-b), e.g. malicious IP core delivery, host or SDR device impersonation/tampering, radio communication interception and IP core theft, tampering, reverse engineering and cloning. Malicious IP cores could compromise the trustworthiness of the SDR system. Host device impersonation or communication tampering could mislead the SDR device to accept and install malicious IP cores. The SDR device impersonation, the communication link eavesdropping and the SDR device tampering could facilitate theft, tampering, reverse engineering and cloning of the genuine IP cores. SeReCon counteracts these risks through its execution of IP core analysis and verification prior to FPGA reconfiguration, end-to-end data encryption, DCE authentication and IP core license enforcement.


\textsuperscript{11} Presentation describing the JTRS systems security context can be found hereSoftware-Based Communications Workshop (http://www.omg.org/news/meetings/workshops/SBC_2005/SBC_2005_Propceedings/03-1_Monahan_etal_V2.pdf).

– 12 –
Figure 1- a. SDR environment and application implemented within a Partially-Reconfigurable (PR) FPGA-based Reconfigurable Computing (RC) device. b. potential attack vectors and the security assurances carried out by the proposed SeReCon element.
Figure 1-a illustrates a block diagram of the PR FPGA-based RC system incorporating the proposed SeReCon IP core, PR region for dynamically loadable SDR IP cores, radio and IO communication interface (Comm. IF/PHY), external non-volatile memory (Local IP Storage) and additional peripherals (DDR/FLASH, EXT DEVICE). SeReCon is a fixed-footprint, trusted reconfiguration base Root of Trust (RoT) for RC systems, which controls all access to the FPGA configuration area, via the ICAP. The Base SeReCon FPGA Configuration (BaseSFC) is loaded into the FPGA after system power up and contains:

- the SeReCon IP core
- a communication interface to control RC system configuration (using the SeReCon element)
- the memory interface to provide non-volatile Local IP Storage (LIPS) implemented using external Compact Flash which stores the RC system IP cores.

Figure 1-b illustrates the SDR application software stack which contains:

- the SDR application
- a high-level SeReCon communication API
- the Operating System (OS), e.g. Linux
- SDR DCE device drivers which uses PR technology and the SeReCon RoT to exchange cryptographic IP cores in response to the type and amount of application data traffic.

Figure 1-c illustrates the SeReCon IP core which contains:

- 32-bit CPU (Microblaze) with local firmware memory (BRAM)
- Communication interface (GPIO IP core from EDK) which allows FIFO-buffered communication with the SDR application and SDR User
- LIPS interface (SysAce IP core from EDK) which enables persistent IP core storage
- True Random Number Generator (TRNG) using LUT-based Ring Oscillators (RO) [35], providing random data used in SeReCon encryption
- Hardware AES block cipher based on the open source design [36]
- Extended ID Register (EIDR) which is an extension to FPGA fabric and provides secure storage of SeReCon security credentials
- ICAP, the FPGA internal configuration port used to activate SDR IP cores.

Figure 1-d illustrates the Extended ID Register (EIDR) which is the FPGA hard-macro primitive embedded within a tamper-proof FPGA fabric. The EIDR provides state-keeping FPGA configuration and tamper-proof user secret credentials storage.

Figure 1-e illustrates the SeReCon firmware stack stored in local memory. The firmware stack contains:

- Board Support Package (BSP) which provides low-level drivers to CPU peripherals
- Various libraries providing generic support for ICAP, TRNG, AES/ECC, Comm and Memory IP cores
• Analyser API which uses Embedded Routing DataBase\(^\text{12}\) (ERDB) for analysing the IP core internals during IP core installation
• Verifier API which uses ERDB for verifying RC system state and IP core requirements prior to IP core activation
• Configuration Manager which provides SeReCon API to drivers used by the SDR application.

SeReCon employs authentication, public-key and symmetric-key cryptographic algorithms in order to protect third party IP designs and facilitate license enforcement of IP cores installed in the system (Figure 1-a). SeReCon maintains the integrity of the PR RC system by mediating all access to the internal Xilinx FPGA reconfiguration port. SeReCon enables the use of untrusted third-party IP cores in high-assurance systems and provides design IP protection and IP core license restrictions enforcement in the active (deployed) system.

The proposed SeReCon system requires a modification to the FPGA fabric to enable SeReCon security credentials (Figure 1-d), used for RC system identification, to be generated internally (within the FPGA) during the system certification process. Security credentials are generated using a Ring-Oscillator (RO)-based True Random Number Generator (TRNG) \([37]\) and remain forever within the SeReCon security perimeter. This policy protects against un-authenticated access to the system credential storage and hence reduces the risk of leakage of security credentials.

The thesis proposes the use of SeReCon for IP core usage accounting, e.g. total runtime, number of activations etc, in a PR system. This facilitates new IP core licensing models, e.g. time-limited and pay-per-use, and provides confidentiality of the IP core implementation during the PR system life-cycle. The SeReCon-based RoT supports license enforcement within the FPGA design flow down to the FPGA configuration bitstream and also within the deployed system.

Figure 1-a illustrates usage scenarios of the SeReCon-based RoT. The RoT incorporates novel algorithms for the generation of system security credentials and trusted design verification. The SI implements the RC system BaseSFC (Figure 1-a), loaded after power up. The BaseSFC, after RC system power up contains only the SeReCon IP core, communication interface and memory interface to provide non-volatile Local IP Storage (LIPS) in external Compact Flash. During RoT initialisation the RC device (with its installed SeReCon RoT) is verified by the TA. The TA certifies the device, the BaseSFC and internally generates the RoT public-key (Figure 1-b). The SeReCon-based RoT and IP management scheme requires the TA participation only during certification of the RC system. This reduces the chain-of-trust in a multi-player design flow. The SI uses the RoT and its public-key to install (in LIPS) the encrypted IP cores (Figure 1-c), obtained from the third-party IPV upon a request from the SI (Figure 1-a). SeReCon enables the use of untrusted

\(^{12}\) The Embedded Routing DataBase (ERDB) contains FPGA routing data which is not documented by Xilinx, but available within genuine Xilinx ‘debug’ bitstreams generated by legitimate Xilinx bitgen tool (with ‘debug’ option turned on). ERDB is compilation of data obtained from multiple ‘debug’ bitstreams.
third-party IP cores in high-assurance systems so long as they do not interfere with the active system configuration. SeReCon provides design IP protection and IP core license restriction enforcement in the active (deployed) system. IP cores installed in LIPS (Figure 1-d) are activated on receipt of a request from the SDR User or SDR application (Figure 1-e). SeReCon maintains the integrity of the RC system by analysing incoming FPGA reconfiguration requests during run-time and by mediating all access to the internal Xilinx FPGA reconfiguration port.

Figure 1 - a. block diagram of the FPGA-based SDR device (PR-enabled RC system) including SeReCon. b. SDR application software stack. c. SeReCon IP core. d. the Extended ID Register element. e. the SeReCon firmware installed in local memory.
Figure 1- a. SeReCon-based Root-of-Trust (RoT) usage scenarios. b. RoT initialisation. c. IP core installation. d. IP core activation.
1.7. Security Issues In FPGA-Based PR RC Systems

Figure 1-a illustrates the example SDR application of the PR RC system. The SDR DCE is an FPGA-based RC system. The SDR DCE exploits PR in order to update radio signal processing modules (IP cores) in the transmitter and receiver pipeline. The SDR DCE is connected to the SDR Data DTE using, e.g. a PCIe interface. The SDR DTE communicates with the host DTE, e.g. the GSM Base Station, using a DCE-to-DCE radio link. When the SDR DCE requires new functionality, the IPV issues updated IP cores which are delivered to the SDR DCE using a radio link or local DCE-DTE interface.

This section describes various attack vectors on the SDR DCE device which are illustrated in Figure 1-b.

**Tampering and impersonation of communication peers.** The attacker mimics or controls the legitimate SDR DTE, or the host DTE/DCE. The attacker manipulates or replays the reconfiguration protocol in order to control the SDR DCE IP cores and RC configuration. The attacker could try to roll-back the SDR DCE configuration by replaying previously intercepted legitimate communication with the SDR DCE. The attacker can also reconfigure the SDR DCE using malicious or erroneous (e.g. obsolete) IP cores. The countermeasure to this attack could be to provide SDR DCE support for authentication of communication peers, assessment of their trustworthiness and certainty that replayed communication will be detected as such.

**Tampering and impersonation of the SDR DCE device.** The attacker physically tampers with the SDR DCE in order to gain access to the configware data and thus control the device. Also, the attacker can mimic (impersonate) the legitimate device in order to receive upgraded IP cores from the host DTE/DCE. This may expose the IP implementation, e.g. algorithm details, used encryption keys etc. This could lead to IPV revenue loss or compromise the system of which the SDR DCE is a part. Tamper proof design of the SDR DCE RoT and its authentication to the host device could be protected against such attacks.

**Indirect malicious IP core delivery.** The attacker has no direct access to the device. However, the attacker could act as a third party IPV or compromise production of a legitimate IP core in order to install the Trojan IP core in the remote SDR DCE. This leads to system compromise without the need to interfere with the reconfiguration process and/or players. A trusted IP core validation within the SDR DCE, which decides whether the IP core violates the SDR DCE integrity, could protect against this risk.

**Communication interception and tampering.** The attacker intercepts and tampers with a legitimate communication channel, e.g. radio link. The attacker could attempt to interfere with an active reconfiguration process, e.g. during the man-in-the-middle attack [38] by, e.g. actively altering the content of the transmitted data (e.g. bitstream, commands etc). Successful interception exposes the reconfiguration protocol and IP core implementation which can then be reverse-engineered or cloned. This typically leads to IPV revenue loss. Successful tampering could install a
Trojan IP core in the SDR DCE during a legitimate reconfiguration request originating from a legitimate host DTE/DCE. This attack can be protected against by use of authenticated encryption.

This thesis proposes SeReCon (Figure 1-c), which provides countermeasures against non-invasive attacks. The thesis also proposes tamper-resistant implementations which provide a countermeasure to semi-invasive and invasive attacks. SeReCon acts as a trusted reconfiguration agent, incorporated within the SDR RC system.

The SeReCon IP core provides the following security measures within the SDR system:

1. Secure communication channel between communicating devices (e.g. SDR DCE and host DTE/DCE or SDR DTE). SeReCon protects the confidentiality and integrity of reconfiguration commands and IP cores while transmitted over the radio link or local interface
2. IP core license restriction enforcement within the SDR DCE
3. Authentication of the communication devices
4. Detection of erroneous configuration data (corrupted or malicious IP cores)
5. Provision of reconfiguration access only to authorised devices (e.g. host or SDR DTE)

1.8. FDAT: FPGA Design Analysis Tool

The thesis also describes the development and application of the FPGA Design Analysis Tool (FDAT) (Figure 1-), a versatile Python framework for low-level analysis and verification of FPGA design bitstreams, which supports rapid prototyping of algorithms for system-level design verification before porting to SeReCon. FDAT provides a set of high-level Application Programming Interfaces (APIs) which abstract the Xilinx FPGA fabric, the implemented design (placed and routed netlist) and the related FPGA configuration bitstream. A lightweight graphic front-end allows custom visualisation of the design within the FPGA fabric.
Figure 1- FPGA Design Analysis Tool (FDAT) block diagram and context.
1.9. Structure Of The Thesis

This thesis is organised as follows:

Chapter 2 reviews and describes the RC system application domain and advantages offered by RC systems over general purpose processors and ASICs. FPGA technology and architectures are introduced and the FPGA-based RC system design flow and PR are described.

Chapter 3 begins with a motivating example on security risks within PR FPGA systems. The example illustrates the risk of implicit communication channels between IP cores in the PR RC system. The risk of side-channel attacks, the threat of rogue EDA software and the issue of malicious FPGA designs are also highlighted. The chapter reviews the state of the art in RC security. Security countermeasures supported by Xilinx FPGA fabric are described prior to critical examination of the reported work on the RC system integrity protection and countermeasures for design IP theft. The principle of IP licensing models is also described. This chapter proposes use of new IP core licensing models, e.g. the time-limited license and metered-access license and highlights the need for the trusted IP-aware RC system security countermeasures. The chapter concludes with the proposal of a Secure Reconfiguration Controller (SeReCon) and a summary of the SeReCon requirements.

Chapter 4 considers the requirements of credentials storage in a secure RoT and the implementation of usage accounting for RC systems. The chapter proposes and describes an extension to the Xilinx FPGA fabric to provide a tamper-proof hardware element which protects the SeReCon-based RoT credentials and usage data during power-up cycles. Techniques for storage of RoT security credentials and usage accounting data in modern FPGAs are reviewed. The suitability and limitations of using SRAM configuration memory are discussed. Other non-volatile memory schemes for credentials storage are also reported. The EIDR element prototype implementation in a Virtex-5 LXT device (ML505 Board) is reported. The register-based EIDR control/status interface, which is implemented in the FPGA user-logic, is highlighted. This chapter also describes EIDR API functions, which are provided by the SeReCon EIDR driver. The associated multi-party RoT credentials generation process is proposed. The activities of SeReCon and various parties (e.g. SI, TA, IPV) during RoT initialisation are highlighted. The RoT credentials generation process supports public security audit of the RC device and guarantees exclusive and authenticated access to the sensitive part of the RC system security credentials for the legitimate system, e.g. SeReCon RoT. The SeReCon-based RoT is immune to credentials leakage as a result of a future successful attack on the TA.

Chapter 5 describes and demonstrates the FPGA Design Analysis Tool (FDAT), a host-based (off-line) bitstream analysis and low-level design verification tool which supports a Xilinx FPGA design assurance strategy and automated extraction and analysis of bitstream-level designs, within the PR design flow. FDAT is an extendable, Python-based system which exploits the functionality of dynamic languages and uses modular libraries of custom-defined analysis scripts. The chapter
Chapter 1 - Introduction

reviews a number of existing tools which facilitate access to low-level design descriptions, and proposes the desired functionality of FDAT. The FDAT architecture and the script-based functionality which exploits the advantages of the Python dynamic language is described prior to presentation of the detailed implementation and evaluation of FDAT, a selection of FDAT recipes, and the FDAT algorithm execution time for analysis of Xilinx Virtex-II Pro inter-tile routing. The chapter proposes porting FDAT functionality to the embedded Secure Reconfiguration Controller (SeReCon) for on-line Xilinx Virtex-5 bitstream analysis. Considerations in creating an embedded routing database and IP core routing analysis are also highlighted.

Chapter 6 describes the internals (state diagram, the block diagram and firmware) of the SeReCon IP core. The chapter reports SeReCon RoT operation within the PR RC device during initialisation, IP core installation, IP activation and IP deactivation. A SafeLock scheme for IP core security credentials protection is highlighted. The process of establishing the shared encryption key between the IPV and SeReCon, using the Diffie-Hellman (DH) shared key agreement protocol is also described. The main steps in the IP core activation process are illustrated. The verification of IP core compliance with the current RC system state is highlighted. The IP core license validation and RC system reconfiguration are also described.

Chapter 7 reports on the implementation and application of the prototype SeReCon-enabled RC system using Xilinx Virtex-5 FPGA technology. The implementation of SeReCon internal elements, the main RC system elements and example PR IP cores is described. Analysis of the SeReCon FPGA resource usage and RC system prototype implementation issues is included. This chapter reports and describes the SeReCon-enabled RC system demonstrator application (including the PCIe communication library and host-side SeReCon API). Demonstrator application results are also reported. The chapter provides detailed insight into the operation of the prototype RC system during the SeReCon (and EIDR) initialisation, IP core installation and activation. The implemented RC system uses four IP cores in order to demonstrate the SeReCon-based PR, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher. The VHDL model for each of these IP cores is included in the thesis DVD. This chapter also describes the SDR device prototype and illustrates how SeReCon element can be included within the SDR RC system. Modifications to the SeReCon implementation required to integrate SeReCon within the prototype SDR device are also highlighted.

Chapter 8 concludes the thesis and proposes future work.
1.10. Thesis Contributions And Novelty Claims

This thesis makes four novel contributions.

1. **SeReCon (Secure Reconfiguration Controller) IP core.**

   The thesis proposes, implements and demonstrates a trusted Secure Reconfiguration Controller (SeReCon) IP core. SeReCon is a fixed-footprint trusted reconfiguration base for RC systems, which performs hardware-level self-reconfiguration and prior-to-reconfiguration IP core analysis (security verification). SeReCon maintains the integrity of the PR-enabled RC system by analysing incoming FPGA reconfiguration requests during run-time and by mediating all access to the internal Xilinx FPGA reconfiguration port. Autonomous analysis of the structure of a new IP core prior to RC system reconfiguration verifies IP core spatial isolation and run-time protection of the already-configured PR system. This protects the RC system from structural issues resulting from the inclusion of erroneously placed (or malicious) IP cores and enables the use of untrusted third-party IP cores in high-assurance systems. SeReCon employs authentication, public-key and symmetric-key cryptographic algorithms in order to protect third party IP designs and facilitate license enforcement of IP cores installed in the deployed PR-enabled RC system. Use of SeReCon IP core reduces the chain-of-trust in a multi-player design flow.

2. **Algorithm for secure generation of RC system security credentials.**

   The thesis proposes, implements and demonstrates a new method for generation and storage of RC system security credentials. Unique security credentials are generated internally (within the FPGA) during the system certification process, using random data obtained from the TRNG. Only the non-sensitive part of the credentials (e.g. SeReCon public-key) is revealed to the TA during the system certification process. This policy protects against un-authenticated access to the system credentials storage (by third-parties) and hence reduces the risk of leakage of security credentials. The private-part of the generated credentials remains within the SeReCon security perimeter, stored in a dedicated Identity Register (IDR). The IDR is part of the SeReCon RoT, embedded in the FPGA fabric, and facilitates authenticated access to credentials material. This makes the RC system (and installed IP cores) immune to tampering by the system integrator and the post-deployment compromise of the TA (which has certified the system). The proposed scheme requires a modification to the FPGA fabric to enable authenticated FPGA configuration and storage of SeReCon security credentials, used for RC system identification.

3. **Algorithm for IP core license enforcement in deployed RC system**

   The thesis proposes, implements and demonstrates a new IP core license enforcement scheme which provides system usage accounting and supports counted- and time-limited IP core licensing. The proposed scheme incorporates a number of monotonic counters which facilitate system usage accounting (uptime, activations etc). Prior to reconfiguration, SeReCon verifies whether IP core usage does not
Chapter 1 - Introduction

exceed license limits. The proposed scheme supports new IP core licensing models, e.g. time-limited and pay-per-use, and enforces license restrictions down to the bitstream-level and within the deployed device. This requires a modification to the FPGA fabric to include tamper-proof monotonic counters within the IDR.

4. FPGA Design Analysis Tool.

The thesis proposes, implements and demonstrates the FPGA Design Analysis Tool (FDAT), an off-line design verification tool. FDAT is a versatile, modular and open tools framework for low-level analysis and verification of FPGA design bitstreams. FDAT enables the development and verification of the SeReCon bitstream analysis algorithms, used in the described implementation of SeReCon. FDAT provides a set of high-level APIs abstracting the Xilinx FPGA fabric, the implemented design (e.g. placed and routed netlist) and the related FPGA configuration bitstream. A lightweight graphic front-end allows custom visualisation of the design within the FPGA fabric. The operation of FDAT is governed by “recipe” Python-based scripts which support rapid prototyping of the algorithms for system-level design verification before porting to SeReCon. To the best knowledge of the authors, FDAT is the first available toolset to provide high-level and unrestricted access to the low-level description of the Xilinx FPGA fabric and the user design at the netlist- and bitstream-level.
Chapter 1 - Introduction

Related publications

Journal Papers


Peer Reviewed Conference Papers


Chapter 2. Partially Reconfigurable Computing
Systems Background

2.1. Introduction

This chapter reviews and describes the RC system application domain and advantages offered by RC systems over general purpose processors and ASICs. FPGA technology and architectures are introduced and the FPGA-based RC system design flow and PR are described.

2.2. RC Systems Vs ASICs And GPPs

Figure 2- illustrates the computational efficiency gap between silicon and microprocessors (General Purpose Processors, GPPs) [39]. The gap widens as feature size reduces. The computational gap between ASICs and GPPs is the target domain of RC systems.

![Figure 2: Computational efficiency gap between silicon and microprocessors, expressed in Million Operations Per Second (MOPS) per Watt, widening as feature size reduces [39].]
A general classification of computing systems in terms of performance and flexibility [11] is presented in Figure 2-. Microprocessor (Von Neumann) architectures (e.g. GPP architecture) provide the ultimate flexibility, e.g. the application (software) is always adapted to the hardware. GPPs are capable of performing any type of computation, at the cost of performance which is limited by sequential computation, i.e. Instruction Read, Instruction Decode, Input Data Read, Instruction Execute and Output Data Write. In contrast, an ASIC architecture is optimised for a particular application. ASICs offer very high performance at the cost of flexibility, since the instruction set is typically hardwired. The design space between ASICs and GPPs is occupied by a number of architectures (refer to Appendix A – List of Reconfigurable Computing architectures). Digital Signal Processors (DSPs) offer the flexibility of GPPs, with increased performance through the addition of domain-specific blocks which support signal processing kernels, e.g. multiply-and-accumulate.

Figure 2- Flexibility vs performance of processor classes [11].

RC systems aim to deliver both the flexibility of GPPs and the performance of ASICs within the same device. For example, Xilinx FPGA devices (Figure 2-) offer a vast amount of Configurable Logic Blocks (CLBs), RAM memory blocks (BRAM), and dedicated function-optimised hardware blocks (e.g. DSP, Multi-Gigabit Transceivers, embedded PowerPC processors etc). Predicted developments in silicon technology indicate a continuous increase in silicon device capacity [40]. Table 2- illustrates approximate (node-scaled) resource density of current and future FPGA technologies, along with the expected advancements in RC technology. FPGA blocks
are connected using interconnect and switch matrix routing resources (Figure 2-), a significant amount of inter-tile wiring of variable length and shape, terminating at Programmable Interconnection Points (PIPs) [41]. PIPs, SRAM technology, and Xilinx FPGA block software configuration (configware) allow modification of the hardware functionality and interconnection even during run time PR. The RC system hardware structure can be modified by downloading the RC device configuration at compile time (static configuration) or at run-time PR. Spatial modification of the RC system hardware architecture can maximise application performance [11]. Advances in RC technology have been significant during the past two decades, and FPGA technology has been widely applied.

SRAM FPGA technology advancements follow Moore’s Law. Modern SRAM FPGA devices from Altera and Xilinx offer capacities which outperform the requirements of all but the most demanding applications such as High Performance Computing (HPC). Several FPGA architectural trends (increasing FPGA component density, raw computational throughput and system functionality) suggest that FPGAs will become increasingly important in the future [42], [43].

<table>
<thead>
<tr>
<th>Technology</th>
<th>Year</th>
<th>LUTs</th>
<th>DSPs</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>65 nm</td>
<td>2007</td>
<td>340 k</td>
<td>500</td>
<td>10 Mbit</td>
</tr>
<tr>
<td>45 nm</td>
<td>2010</td>
<td>700 k</td>
<td>1000</td>
<td>21 Mbit</td>
</tr>
<tr>
<td>32 nm</td>
<td>2013</td>
<td>1400 k</td>
<td>2000</td>
<td>42 Mbit</td>
</tr>
<tr>
<td>22 nm</td>
<td>2016</td>
<td>2900 k</td>
<td>4300</td>
<td>89 Mbit</td>
</tr>
</tbody>
</table>

Table 2- Approximated resource density of current and future FPGA technologies [42].

2.3. FPGA Architecture And RC System Design Flow

2.3.1. Introduction

This section provides an overview of general FPGA architectures and design flow, the FPGA configuration bitstream structure and partial reconfiguration technology.

2.3.2. FPGA Architecture Overview

Figure 2- illustrates an FPGA block diagram. An FPGA is a programmable device consisting of a set of configurable resources (e.g. logic blocks, programmable interconnect network and I/O blocks) [44]–[46], configuration memory, configuration control logic and debug interface (JTAG). The functionality of an RC system design is partitioned into modules and implemented using configurable logic
and I/O blocks within the FPGA. Logic and I/O blocks are connected using an interconnect network. All configurable resources (logic blocks, interconnect network and IO blocks) can be programmed by the user in the field. FPGAs can be one-time or many-times programmable, depending of the memory technology used (Antifuse, SRAM, EEPROM, FLASH, etc). Some Xilinx FPGA devices (e.g. Virtex-II/4/5/6, Spartan-6) support bitstream encryption [44], [47], [9], [33]. Virtex-6 also supports authenticated configuration mode [33].

![Block diagram of an FPGA device.](image)

### 2.3.3. Xilinx FPGA Configuration Bitstream Structure

Modern Xilinx FPGA devices (e.g. Virtex-4/5/6, Spartan-6) have configuration memory arranged in frames within a tile structure (Figure 2-). These frames are the smallest addressable segments of the Xilinx FPGA configuration memory space, and all FPGA reconfiguration operations must therefore act upon whole configuration frames [9]. Figure 2- illustrates the tile-based frame organisation of the configuration memory in the Xilinx Virtex-5 (V5LX50T) FPGA device.
Figure 2- Tile-based frame organisation of the configuration memory in the Xilinx Virtex-5 (V5LX50T) FPGA device.

The Xilinx FPGA configuration file (bitstream) contains commands to the FPGA device configuration logic (Figure 2-) as well as configuration data (Figure 2-). All Xilinx FPGA bitstream commands, such as data or control/status read and write, are executed by reading or writing data packets from/to the configuration registers respectively, in the FPGA Configuration Control Logic (CCL, Figure 2-). Listing 2- illustrates the Xilinx Virtex-5 bitstream internal structure decoded by the FPGA Design Analysis Tool (FDAT) reported in this thesis. Lines 10-54 illustrate bitstream packets (Type_1 and Type_2) which include commands for FPGA Configuration Control Logic (CCL). The CCL command sequence is mostly consistent with the bitstream composition described in [9]. Listing 2- contains additional commands in lines 18 and 43.
FDAT Bit parser demo. Parsing the '../adder_v7/download.bit' bitstream...

@ 0x00000050 | 0xFFFFFFFF : DUMMY word
...

@ 0x00000050 | 0xFFFFFFFF : DUMMY word
@ 0x00000070 | 0x000000BB : BUS WIDTH word
@ 0x00000074 | 0x11220044 : BUS WIDTH (x2)
@ 0x00000078 | 0xFFFFFFFF : DUMMY word (x2)
@ 0x00000080 | 0xAA995566 : SYNC word
@ 0x00000084 | 0x20000000 Type_1 : NOP
@ 0x00000088 | 0x30020001 Type_1 : Write 1 word to 'WBSTAR' 0x00000000 -> WBSTAR
@ 0x00000090 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000000 -> CMD (NULL)
@ 0x00000098 | 0x20000000 Type_1 : NOP
@ 0x0000009C | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000007 -> CMD (RCRC)
@ 0x000000A4 | 0x20000000 Type_1 : NOP (x2)
@ 0x000000AC | 0x30022001 Type_1 : Write 1 word to 'TIMER' 0x00000000 -> TIMER
@ 0x000000B4 | 0x30026001 Type_1 : Write 1 word to 'R0x13' 0x00000000 -> R0x13
@ 0x000000BC | 0x30012001 Type_1 : Write 1 word to 'COR0' 0x000031E5 -> COR0
@ 0x000000C4 | 0x3001C001 Type_1 : Write 1 word to 'COR1' 0x00000000 -> COR1
@ 0x000000CC | 0x30018001 Type_1 : Write 1 word to 'IDCODE' 0x02A96093 -> IDCODE
@ 0x000000D4 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000009 -> CMD (SWITCH)
@ 0x000000E0 | 0x3000C001 Type_1 : Write 1 word to 'MASK' 0x00400000 -> MASK
@ 0x000000E8 | 0x3000A001 Type_1 : Write 1 word to 'CTL0' 0x00400000 -> CTL0
@ 0x00000100 | 0x3000C001 Type_1 : Write 1 word to 'MASK' 0x00000000 -> MASK
@ 0x00000108 | 0x30030001 Type_1 : Write 1 word to 'CTL1' 0x00000000 -> CTL1
@ 0x00000110 | 0x20000000 Type_1 : NOP
...
@ 0x00000120 | 0x30020001 Type_1 : Write 1 word to 'FAR' 0x00000000 -> FAR (t:0 h:0 r:0 c:0 m:0)
@ 0x00000128 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000001 -> CMD (WCFG)
@ 0x00000130 | 0x20000000 Type_1 : NOP
@ 0x00000134 | 0x30004000 Type_1 : Write 0 words to 'FDRI' (0 frames)
@ 0x00000138 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000009 -> CMD (SWITCH)
@ 0x00000140 | 0x20000000 Type_1 : NOP
...
@ 0x00000160 | 0x30020001 Type_1 : Write 1 word to 'FAR' 0x00000000 -> FAR (t:0 h:0 r:0 c:0 m:0)
@ 0x00000168 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000001 -> CMD (WCFG)
@ 0x00000170 | 0x20000000 Type_1 : NOP
...
@ 0x000001A0 | 0x30000001 Type_1 : Write 1 word to 'CRC' 0xFF03D69A -> CRC
@ 0x000001A8 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x0000000A -> CMD (GRESTORE)
@ 0x000001B0 | 0x20000000 Type_1 : NOP
@ 0x000001B4 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x00000003 -> CMD (DHIGH/LFPM)
@ 0x000001C0 | 0x20000000 Type_1 : NOP
...
@ 0x000001E0 | 0x30000001 Type_1 : Write 1 word to 'CRC' 0xFF03D69A -> CRC
@ 0x000001E8 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x0000000A -> CMD (GRESTORE)
@ 0x000001F0 | 0x20000000 Type_1 : NOP
...
@ 0x00000220 | 0x30008001 Type_1 : Write 1 word to 'CMD' 0x0000000A -> CMD (GRESTORE)

Listing 2.- Example of Xilinx Virtex-5 bitstream internal structure decoded by FPGA Design Analysis Tool (FDAT).
Configuration bitstream data can be delivered to the FPGA device using either a serial or parallel interface (8, 16 or 32-bit wide). Listing 2-1 command descriptions are as follows:

Lines 2-7 synchronise word-aligned FPGA configuration control logic with the bitstream data source.

Lines 9 to 28 setup various FPGA control registers, e.g., line 27 sets the Frame Address Register (FAR) to the starting address, while line 28 resets FAR auto incremenrer and resets the FPGA internal frame buffers.

Line 30 sets CCL Frame Data Input Register (FDRI) as a destination register for the following Type_2 data packet write in line 31. The command loads 438864 words into the FDRI register. This configures the entire FPGA configuration memory using the FAR auto-incrementer facility [9].

Line 33 delivers the CRC value. CCL compares the CRC with the internally calculated value.

Lines 34-42 toggle the global set/reset lines, write the final configuration frame to the configuration memory and activate the new configuration.

Line 44 sets FAR at the dummy (nonexistent) location.

Line 48 disconnects the configuration logic.

The command in lines 22 and 46 is not documented in publicly available documentation. The relevant bit in the CCL CTL0 register is marked as ‘reserved’ in [9]. Also, the command writing 0 to register 0x13, in line 15, is not documented.

2.3.4. FPGA-Based RC System Design Flow

Figure 2- illustrates the FPGA-based RC system design flow. The main stages are described below.

**RC System architecture design.** This stage includes the analysis of the RC system requirements, problem decomposition and functional simulation (at gate and system level). Functional simulation verifies the RC system behavioural correctness. The output of this stage is an RC system specification which describes, in a formal hardware description language (HDL), the device architecture, structural blocks and their functions and interfaces.

**Design capture and generation of FPGA configuration bitstream.** This stage includes synthesis of a captured HDL description to a design netlist. Synthesis can reveal some problems and potential errors that cannot be found using behavioural simulation. The main phase of the implementation stage is Place and Route (P&R). During P&R a synthesis-generated netlist is mapped onto the internal structure of the FPGA device. P&R allocates FPGA resources such as logic cells and connection wires. The timing analysis stage checks whether the implemented design satisfies the system timing constraints (such as clock frequency) specified by the architectural constraints. The output of this stage is the FPGA configuration data, in the form of an
FPGA configuration file (bitstream). Details of the Xilinx FPGA design flow are described in [48].

**Device hardware implementation.** This stage includes the configuration of the RC device hardware, existing as a component on a Printed Circuit Board (PCB).

**Device software implementation and integration.** This stage includes the development of FPGA device drivers and user/communication interfaces, running on a host processor system. APIs are developed for use with the RC system. During the integration phase the software is installed in the RC device non-volatile memory (e.g. Flash). The RC system functionality is verified by applying test vectors to the pins of the device and comparing the output obtained with results obtained from behavioural simulation of the design.

Figure 2- FPGA-based RC system Design Flow [49].
2.4. Xilinx Partial Reconfiguration (PR) Design Flow

2.4.1. Introduction

This section describes Xilinx Early Access PR (EAPR) technology\(^\text{13}\), its advantages and disadvantages, and highlights PR-related FPGA design flow steps. Also, Rousseau et al. report an alternative PR design flow which enables design certification in safety-critical applications (e.g. avionics) [50]. The alternative PR flow exploits standard FPGA design flow and tools, and does not require access to proprietary EAPR tools.

User-programmable features of Xilinx FPGAs are controlled by configuration memory cells. FPGA configuration memory is programmed using a configuration bitstream. PR enables a programmed Xilinx Virtex device to be partially reconfigured using a partial bitstream, which can be loaded without interrupting the operation of the remaining part of the device. PR is an analogy of CPU context switching in software processes. PR increases the functionality of a single FPGA, allowing a part of the system functionality to be time-multiplexed. Thus, with PR, smaller FPGA devices with PR-based time-multiplexed functionality can be used in the RC system, or the system can incorporate more functionality within the same sized device. Figure 1- illustrates a PR RC system with functions such as a PCIe communication link, a configuration manager (SeReCon), and an interface to external, non-volatile memory. The base static configuration region in FPGA is the portion of the design that does not change during PR and includes logic controlling the PR process. With PR, all of these base static configuration modules can maintain their real-time links while other modules within the FPGA PR region are exchanged.

2.4.2. Xilinx Early Access PR Design Flow

The first release of Xilinx PR [51] introduced two PR flows, namely difference-based and module-based. The difference-based PR targets applications which require only minor changes to the design. The applications are typically limited to cases which update the FPGA block configuration (BRAM/LUT content, I/O standard etc). Configuration changes are performed manually on the placed and routed design netlist using, e.g. the Xilinx FPGA Editor. The module-based PR uses a Xilinx Modular Design methodology [52] and supports Xilinx FPGA devices up to Virtex-II Pro. The EAPR design flow supports Virtex-4 and provides limited support for Virtex-5.

---

\(^{13}\) As supported by Partial Reconfiguration Early Access Software Tools overlay for ISE 9.2i SP4 (available from \[http://www.origin.xilinx.com:80/support/prealounge/protected/index.htm\]). This tool overlay provides limited support for Virtex-5 architecture referenced through this thesis. A new version of PR tools, with full support for Virtex-5/6 is announced as an integral part of the ISE Design Suite 12.1 release (\[http://www.xilinx.com/tools/partial-reconfiguration.htm\]).
Module-based PR flow assumes:

- column-based PR (the PR region spans the full FPGA column)
- all logic resources within the PR region are part of the reconfigurable module (this includes slices, TBUFS, block RAMs, DSP blocks, IOBs and all routing resources)
- Bus Macro (BM) based communication with PR region. BMs are used as fixed (directional) data paths for signal connections between a reconfigurable region and another region or static design.

The current release of the EAPR tools\textsuperscript{14} relaxes restrictions on PR designs. EAPR includes full support for Xilinx Virtex-4 FPGA devices and experimental support for Virtex-5 (including single-slice BMs). The PR region can span a group of rectangular sub-regions and can contain routing of the static part (this increases timing performance and simplifies the implementation). Xilinx PlanAhead [53] software supports a single project which manages the reconfigurable modules and runs the necessary implementation tools to generate the static and partial bitstreams. Full support for Virtex-5 and Virtex-6 devices is expected with the next release of Xilinx tools\textsuperscript{15}.

Figure 2- illustrates the Xilinx EAPR design flow [10]. The EAPR design flow requires additional steps not found in the generic FPGA design flow (Figure 2-). The generic FPGA design flow involves a single pass through the implementation tools (NGDBuild, MAP and PAR) while EAPR requires the base design and each PRM to be implemented separately, with a final merge step which generates the full and partial configuration bitstreams. EAPR design flow steps 1-4 are similar to the non-PR design flow while EAPR design flow steps 5-7 are unique to the PR design flow. The full bitstream contains the base static configuration merged with one of the PR bitstreams (default configuration). Details of the EAPR design flow steps are described in [10].

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{eapr_flow_diagram.png}
\caption{Xilinx EAPR design flow [10].}
\end{figure}

\textsuperscript{14} Available from \url{http://www.origin.xilinx.com/support/prealounge/protected/index.htm}.

\textsuperscript{15} Xilinx online announcement (\url{http://www.xilinx.com/tools/partial-reconfiguration.htm}).
Chapter 2 - Partially Reconfigurable Computing Systems Background

The Xilinx EAPR design flow supports static routing in the PR region [10]. This introduces a risk of implicit communication setup, e.g. the attacker could prepare a malicious IP core which could connect to unprotected static routing resources. This thesis assumes that static routing does not exist within the PR region. Thus, all detected external PIPs are presumed to be the interface between the BaseSFC and the PR region.

2.4.3. Xilinx PR Bitstream Structure

Listing 2- illustrates the structure of a Xilinx Virtex-5 PR bitstream, decoded using FDAT. The main difference between the full bitstream (Listing 2-) and the PR bitstream is the use of data compression (which reduces the PR bitstream size). The full bitstream has constant size, and contains the complete set of FPGA configuration frames (except special frame types which are not documented and not included in default bitstreams). This guarantees constant configuration time after power up, which is independent of the bitstream payload. Many configuration frames are redundant, e.g. empty since the FPGA routing resources are never fully utilised. With PR, only a region within the FPGA is configured. The region size is not constant, changes depending of the application and PR module implementation. In PR applications, e.g. real-time video processing [54] or SDR, it is beneficial to reduce the reconfiguration time, which is linearly related to the amount of configuration frames. Thus, during PR bitstream creation, frame addresses with similar data are grouped and configured in a batch, using the frame buffer within the FPGA [9], the Multiple Frame Write (MFW) command and MFW register (lines 77-83 in Listing 2-). The exact frame grouping algorithm for MFW writes is not published and some experiments suggest that the order of configuration frames is not arbitrary and, when violated, could lead to design congestions16. This raises concerns about the reliability of third-party tools used to create PR bitstreams.

16 "Very interesting finding about V4 CLB configuration bits “ from the Comp.Arch.FPGA Usenet (http://www.fpgarelated.com/usenet/fpga/show/88244-1.php)
FDAT Bit parser demo

Parsing the './adder_v7/blank.bit' bitstream...

@ 0x00000071 | 0xFFFFFFFF : DUMMY word
... 
@ 0x00000071 | 0xFFFFFFFF : DUMMY word
@ 0x000000B | 0x000000BB : BUS WIDTH word
@ 0x1220004 | 0x000000BB : BUS WIDTH
@ 0x000000D | 0xFFFFFFFF : DUMMY word
@ 0x000000E | 0x000000BB : BUS WIDTH word
@ 0x0000008 | 0x000000BB : BUS WIDTH
@ 0x000000B | 0x0A995566 : SYNC word
@ 0x0000008 | 0x0A995566 : SYNC word

Listing 2.- Example of Xilinx Virtex-5 (V5LX50T) PR bitstream structure including Multi Frame Write (MFW) command packets decoded by FPGA Design Analysis Tool (FDAT).
Chapter 3. Reconfigurable Computing Security: Background

3.1. Introduction

This chapter begins with a motivating example on security risks within PR FPGA systems. The example illustrates the risk of implicit communication channels between IP cores in the PR RC system. The risk of side-channel attacks, the threat of rogue EDA software and the issue of malicious FPGA designs are also highlighted. The chapter reviews the state of the art in RC security. Security countermeasures supported by Xilinx FPGA fabric are described prior to critical examination of the reported work on the RC system integrity protection and countermeasures for design IP theft. The principle of IP licensing models is also described. This chapter highlights the need for the trusted IP-aware RC system security countermeasures. The chapter concludes with the proposal of a Secure Reconfiguration Controller (SeReCon) and a summary of the SeReCon requirements.

3.2. Security Risks In PR FPGA Systems: A Motivating Example

3.2.1. Introduction

This section illustrates an example bitstream-level security problem, namely the risk of occurrence (and consequence) of implicit communication channel setup between IP blocks. The HDL to FPGA configuration bitstream translation process is detailed along with the various databases (design netlists) which result.

The Xilinx CB4CLE (4-bit loadable binary counter with enable and asynchronous clear) component example [55] indicates that even a simple IP core, occupying one CLB, can potentially share routing resources (and thus potentially directly communicate) with a second IP core. This situation would occur if the P&R database of the second IP core uses a connection also used by the CB4CLE IP core. A risk of implicit communication between IP cores, e.g. through inadvertent inter-IP block connection, challenges the assurance of an FPGA-based design. Malicious interference using standard routing line connections or clock line connections is highlighted. There is therefore a need for extended tool support for low-level (bitstream, P&R design netlist) analysis and verification. FDAT-based low-level
design analysis of the CB4CLE IP core within a Virtex-5 fabric is illustrated, with extraction of inter-tile routing to verify design spatial isolation.

3.2.2. CB4CLE Design Representation

Figure 3- illustrates, for a 4-bit binary counter (CB4CLE), a range of design representations occurring during the FPGA design flow.

Figure 3 - Various (equivalent) representations of the CB4CLE counter design [55]. a. VHDL source. b. mapped netlist. c. configuration bitstream. d. FPGA Editor view of the P&R design netlist. e. FDAT view of the CLB tiles (red) and the P&R design netlist (blue) with separate routing tiles (yellow).
Chapter 3 - Reconfigurable Computing Security: Background

The VHDL model (Figure 3-a) is synthesised to produce an RTL/technology netlist (Figure 3-b), mapped to the FPGA logic resources. The Place & Route (P&R) process annotates the netlist components with physical FPGA location data, connected using FPGA configurable routing resources. The P&R netlist is translated to the FPGA configuration bitstream file, used to configure the FPGA device (Figure 3-c).

The implemented design can be viewed using vendor-specific (e.g, Xilinx FPGA Editor, Figure 3-d) or third-party tools such as the proposed FDAT tool (Figure 3-e). Tools such as FPGA Editor are useful in providing a view of the FPGA fabric and can be used to perform a modification to the Xilinx Native Circuit Description (NCD) design netlist file, but do not support the analysis of the FPGA configuration bitstream file (BIT). Such tools may omit details of the underlying architecture, e.g. certain tile types, test wires etc, and do not support a broad script-driven enquiry of the bitstream database.

3.2.3. The Security Risk Of Implicit Communication Channel Between IP Cores

Figure 3- illustrates routing tiles within a Xilinx Virtex-5 LX50T which contain a significant amount of inter-tile wiring of variable length and shape, terminating at PIPs. The PIP configuration defines the wire connections required to enable the distribution of signals to distant locations [41], e.g. using multi-hop connections [56]. If all PIPs associated with a particular wire are located within an IP core boundary, implementation tools have full control over IP core routing. Therefore, all unused PIP connections are not accessible externally, e.g. by other IP blocks, and automatically remain unconnected externally. If a wire is connected to a PIP outside the IP core boundary, an implicit communication channel may result.

Figure 3- illustrates the potential implicit communication channels which can occur due to distributed connections across the FPGA fabric, available for use during IP design P&R. This channel connection may not be detectable during standard system testing (design debugging, functional verification of design implementation) and thus may introduce errors, system failure, or expose the IP to security risks. A guarantee is required that the unused routing from IP core $k$ remains unused in IP core $n$. IP cores could be implemented with consideration of additional constraints which globally enforce isolation of the IP core internal wiring [26], or use dedicated isolation primitives [57]. In a multi-party design flow, it is challenging for the System Integrator (SI) to ensure, without the use of a low-level analysis tools such as FDAT, that isolation constraints have been implemented within third-party IP cores. The automated extraction of IP core connectivity is not readily available in existing tools for low level analysis of placed and routed netlists, bitstreams and FPGA fabric netlists.
3.2.4. IP Core Implicit Communication Channel Over The Clock Line

Figure 3- illustrates an FDAT-generated visualisation of the abundant connectivity of the CB4CLE design within a Virtex-5 and Virtex-II Pro FPGAs. While the CB4CLE function utilises a single CLB tile, its IO signals use routing resources (30 Virtex-5 FPGA tiles, 24 Virtex-II Pro tiles for a full design) which are shared by at least one electrical connection within 5% of the FPGA tiles (689 out of 13832 tiles, Figure 3-a). When global clock routing resources (four clock-tree tiles) are included, the counter IO signals use routing resources which are shared by at least one electrical connection within 21% of FPGA tiles (2915 out of 13832 tiles, Figure 3-b). This suggests that taking control of the FPGA “clock-spine” could potentially be an attractive target method for embedding malicious design elements (in order to set up an implicit communication channel). Two possible examples of attack using FPGA clock resources are as follows:

a) malicious interference using a clock signal, e.g. dynamic control of the clock line load (number of active FPGA resources using FPGA clock spine, called “sinks”) changing clock jitter/skew, clock signal keying, clock frequency manipulation etc. This dynamic control of the clock source, when performed by an attacker in a systematic way (e.g. undisturbed, default clock operation...
would define a logical ‘0’, while detected interference would define a logical ‘1’) could be used for steganographic information hiding within the genuine clock signal. Hidden information could be detected (recovered from the signal) by the malicious design at the clock sink side thus facilitating a unidirectional subliminal communication channel. Figure 3- illustrates the block diagram of the malicious IP core in a steganographic application. The Attacker manipulates genuine clock signal in order to hide control messages send to the malicious IP core.

b) the setting up of an additional clock line between two malicious designs solely for malicious communication purposes.

Figure 3- Communication channel existing outside two IP cores. This introduces the risk of setup of an implicit communication channel, which may introduce errors, system failure, or expose the IP to security risks.

The disadvantage of the “clock-spine” based communication is that both of the above scenarios require additional logic on both ends of the clock-based communication link, one used for injecting the interference (data) at the clock source, the second one for detecting it at the clock sink. Thus, eavesdropping on the design (IP core #k, Figure 3-) requires design modification. Also, the hierarchical structure of the clock spine (root lines, branches, buffers etc) and the FPGA fabric technology (which supports low-skew and large number of clock line sinks) limits the robustness of this type of malicious communication. Detailed risk analysis of clock line attacks feasibility is beyond the scope of this thesis and further research is encouraged.

17 Steganography (http://en.wikipedia.org/wiki/Steganography)
Figure 3- FDAT-generated visualisation of the abundant connectivity of the CB4CLE counter design. a.b. within a Virtex 5 LXT FPGA. c.d. within a Virtex-2 Pro FPGA. The CB4CLE (blue and yellow) utilises a single CLB. CB4CLE signals use routing resources (green) which are shared by at least one electrical connection (a & c –without clock tiles, b & d – including clock tiles).
Chapter 3 - Reconfigurable Computing Security: Background

3.3. State-Of-The Art In FPGA Design Security

3.3.1. Introduction

The main objectives of the FPGA-based RC system attacker are system integrity compromise and/or design IP theft. System integrity compromise gives the attacker unauthorised control over the system elements, e.g. security countermeasures and configuration [58]. The attacker can covertly falsify or leak data processed by the system, alter system elements with Trojan substitutes or implant additional (malicious) functionality, e.g. for bypassing authentication (back door function) [2]. The risk scales to external systems when the compromised system is part of a trusted infrastructure and its misbehaviour cannot be detected.

RC system integrity protection deals with issues of malicious bitstream eavesdropping, device tamper-resistance etc. RC security countermeasures are mainly applied to high-assurance systems. System integrity compromise could also lead to IP theft. The motivation behind design IP theft is the high design cost and time and market share advantages. In RC systems the design IP can be infringed in the following ways:

- **Reverse Engineering (RE):** a low-level and laborious decoding of the configware, e.g. FPGA bitstream, back to the HDL description. RE reveals proprietary algorithm implementation, or a key embedded (‘hidden’) in the design. McLoughlin examines the RE problem scope and classification [59].
- **Cloning:** copying of the design (configware, netlist, HDL sources etc) with little or no modification.
- **Counterfeiting:** producing a fake or lower-quality design imitating the original product.\(^{18}\)

---

\(^{18}\)Counterfeit / Fake Cisco WIC-1DSU-T1 V2: Guide to tell Genuine from Counterfeit (http://www.andovercg.com/services/cisco-counterfeit-wic-1dsu-t1-v2.shtml), Departments of Justice and
Chapter 3 - Reconfigurable Computing Security: Background

- **Overbuilding (license abuse):** violation or bypassing of the restrictions limiting design IP use (and the number of legal copies). Also, blocking of license verification algorithms.

A survey of security challenges facing embedded systems has been conducted by Ravi et al. [60]. The survey discusses the challenges involved in secure embedded system design, the ‘security processing gap’ in battery powered devices and the “assurance gap,” which relates to the gap between functional security measures (e.g. security services, protocols, and their constituent cryptographic algorithms) and actual secure implementations. Wollinger et al. [61] review the advantages of FPGAs for security applications, list FPGA shortcomings with proposed countermeasures and state a number of open questions (from a systems point of view). Valette et al. [62] list FPGA security features and emphasise configuration memory programming as the point of least security. Jurjens [63] provides an overview of the challenges in developing secure embedded systems and demonstrates model-based security engineering, while Irvine and Levitt [64] explore challenges to trusted hardware. Kastner and Huffmire [65] address threats and challenges in reconfigurable hardware security by describing attacks, solutions and areas for future research. Their work discusses the life cycle of reconfigurable hardware and focuses on the topics of trusted hardware, physical attacks, design tool subversion, design theft and system security. Drimer [66] reviews FPGA design security, with a focus on volatile (SRAM) technology, examining a wide range of attacks and defences, along with the current state of industry offerings. Drimer also outlines on-going research and latest developments in the field.

Attacks on FPGA-based designs are classified depending of their invasiveness level.

**Non-invasive** attacks involve monitoring and interaction using FPGA I/Os (glitching, power analysis etc) or configware implementation (data residue, API attacks), e.g. by brute-force (exhaustive) testing, side-channel analysis, probing and copying external data, manipulating inputs and observing outputs etc. This is the cheapest type of attack which typically does not require significant resources (advanced tools, funds, time) and does not include physical tampering of the FPGA chip.

**Semi-invasive** attacks involve chip depackaging and direct (microscopic) observation of the silicon die, e.g. thermal, electromagnetic emanations, UV attacks, focused IOB etc. The silicon die is not tampered with.

**Invasive** attacks involve chip depackaging, micro-probing and altering points inside the chip using, e.g. Focused Ion Beam (FIB) and Scanning Electron Microscopy (SEM). This type of attack allows fault injection inside the active design, e.g. by changing the dynamic state of CLB Flip Flops or routing PIPs. This is the most
Chapter 3 - Reconfigurable Computing Security: Background

intrusive and advanced type of attack which typically requires a significant amount of resources and advanced tools.

This thesis assumes a tamper-proof FPGA device. Thus only non-invasive attacks are investigated, e.g. side-channel eavesdropping, communication protocol tampering etc.

3.3.2. Side-Channel Attacks And Countermeasures

When active, digital circuits expose dynamic power consumption which is related to the processing algorithm, the fabric technology and the system internal state. Also the processed data is taken into account. The technology and the processing algorithm are constant. Thus current fluctuations result from state transitions within the design logic (dynamic current leakage). When sampled over time, the current fluctuations provide a power footprint which is specific to invariant algorithm steps and input data. A similar footprint can be observed for Electro-Magnetic (EM) emanations (with additional spatial properties).

Side-Channel Analysis (SCA) exploits power or the EM footprint of the device in order to predict its internal state. For a known processing algorithm SCA could reveal input data, e.g. encryption keys or plaintext in a cryptographic application. Coron and Naccache propose the definition of information leakage immunity and a relevant testing methodology [67]. Aigner and Oswald provide a tutorial on Differential Power Analysis (DPA) attacks [68][69].

Kocher, Jaffe and Jun examine DPA of tamper-resistant devices and propose countermeasures [70]. Tiri and Verbauwhede propose simulation models for side-channel information leaks in DPA [71]. Standaert et al. provide an overview of power analysis attacks targeting FPGAs [72]. Peeters, Standaert and Quisquater also provide an improved ‘switching-distance’ leakage model of CMOS devices [73]. Tiri examines pitfalls of side-channel analysis and discusses possible countermeasures [74]. Agrawal, Archambeault and Rao examine Electro-magnetic (EM) side-channel attacks on CMOS devices [75] and develop a practical assessment methodology for such devices.

Poon, Wilton and Yan examine a FPGA power model used by the Versatile Place and Route (VPR) tool [76]. Shang, Kaviani and Bathala examine dynamic power consumption in Xilinx Virtex-II devices [77]. Curd [78] describes the impact of architectural innovations on power consumption in Xilinx Virtex-5 devices.

Chaudhuri et al. examine the viability of multi-rail routing in FPGAs and its robustness against side-channel attacks [79]. Tiri, Schaumot and Verbauwhede examine design of DPA side-channel leakage tolerant architectures [80]. Yu and Schaumont propose an extended Wave Dynamic Differential Logic (WDDL) technology, called Double WDDL (DWDDL) [81]. DWDDL implements a side-channel-resistant logic with invariant signal pair load, regardless of signal transitions. Reported DWDDL implementation withstands reference DPA attack, but at a significant resource cost and delay.
3.3.3. EDA Tools As The Security Threat

Design functionality expressed by the HDL source code is generally preserved by the EDA tools through to design implementation, in order to preserve the functional equivalence of the design description. While synthesis and P&R tools operate successfully over a wide range of design translations, certain corner cases exist where post-P&R manual editing is required. An example is the ring-oscillator (RO) which is a core element in Random Number Generator designs. Due to cycles in combinatorial logic, the RO is typically optimised out during P&R and needs to be manually added to design netlist (NCD) afterwards, using tools such as FPGA editor. Thompson [21] highlights how development tools in the software domain could implicitly produce overbuilt (additional functionality) and malicious binary output, not functionally-equivalent to the input source code. Also, Roy et al. [4] challenge the security of EDA tools, which could covertly inject Trojan circuits into user designs. Roy et al. also examine methods for Trojan circuit planting and masking inside the legitimate design. Thus, the source-code-only design verification (e.g. HDL verification) may not be sufficient. Roy et al. propose dynamic verification as a countermeasure. Bitstream analysis tools could be applied within the PR implementation flow to detect malicious elements following third-party tool application. The assurance that two CB4CLE IP cores (each one sharing at least one wire with 21% of the FPGA tiles) do not share any routing resources could be provided by precise allocation of 42% of a mid-sized FPGA device. This approach is clearly impractical. Since post P&R design simulation is not performed directly on the FPGA bitstream it cannot trap potential implicit communication channels between IP cores.

The proposed FDAT tool can determine if an implicit communication channel has occurred. In this work, this FDAT functionality has been ported to the SeReCon element within the RC system in order to enable detection of implicit communication channels in installed third party IP cores.

3.3.4. Malicious FPGA Designs

Research on malicious configware and IP cores has been reported by Hadzic et al., King et al., Roy et al. and Kucera and Vetter. Hadzic et al. describe the threat of malicious FPGA bitstreams [1]. The paper outlines FPGA properties supporting the threat and demonstrates the implementation of the FPGA virus which physically destroys the device elements. Hadzic et al. also propose changes to the FPGA architecture, bitstream verification and high current detection as a countermeasure. King et al. report implementation of “the Illinois Malicious Processors (IMPs)” [2]. IMP is a general purpose processor with an architecture which supports a number of general security attacks. King also investigates design space, trade-offs, and challenges of malicious circuits. Alkabani and Koushanfar propose disguising hardware Trojan circuits as additional states within the IP core Finite Sate Machine
Chapter 3 - Reconfigurable Computing Security: Background

(FSM) [3]. Kucera and Vetter discuss the viability of the FPGA rootkit\textsuperscript{19} deployment [5]. Kucera and Vetter outline rootkit applications in embedded systems and list the shortcomings of existing FPGA security features and its implications in Trusted Computing.

3.3.5. Xilinx FPGA Fabric Protection

Xilinx FPGAs and EDA software employ a number of technologies in order to increase security of SRAM FPGAs. Lesea [6] and Trimberger [7] provide IP-centric and TC-oriented context for some of the following features.

- Design sources and netlist encryption: implemented in Xilinx design tools since early versions of Xilinx ISE, used to protect IP core netlists generated by Xilinx Coregen and IP core source files delivered with the Xilinx EDK software. ISE uses the DES encryption algorithm (up to version 9 of ISE\textsuperscript{20}).
- eFUSE primitive: eFUSE has been officially available since Virtex-6/Spartan-6 series release, and is an undocumented feature of Virtex-5. EFUSE technology uses electromigration [82] to provide one-time programmable storage for the encryption key (Virtex-6 and Spartan-6 devices only) and user data (Spartan-3AN/6, Virtex-6).
- Bitstream encryption: available in Virtex-II/4/5/6 and Spartan-6 devices, enables FPGA configuration with a user-encrypted bitstream. Modern Xilinx devices use AES 256 encryption in Cipher Block Chaining (CBC) mode. Virtex-II Pro (now-obsolete) uses the 3DES encryption algorithm.
- Volatile storage: implemented in Virtex devices since Virtex-II, and also available in Spartan-6. This enables volatile storage of encryption keys and instant zeroisation upon tamper detection (using the KeyClear primitive) which is compliant with Federal Information Processing Standard (FIPS) [83] published by the National Institute of Standards and Technology (NIST)\textsuperscript{21}.
- Authentication: introduced in Virtex-6 devices, using SHA-256 and HMAC algorithms. Authentication enables prior-to-configuration bitstream authentication with a user-defined key. Its description in [33] suggests that the key is part of the encrypted bitstream and is not stored in the FPGA.
- DeviceDNA: available in Spartan-3AN/6 and Virtex-6 devices. Device DNA is a vendor programmed design primitive which provides the unique serial number of the FPGA fabric to the user design.

\textsuperscript{19} Rootkit is a software program or coordinated set of programs designed to gain control over a computer system or network of computing systems without being detected (http://en.wikipedia.org/wiki/Rootkit).

\textsuperscript{20} “How to decrypt Xilinx IPCORE source code” (http://newsgroups.derkeiler.com/Archive/Comp/comp.arch.fpga/2009-01/msg00110.html) describes required decryption steps and provides relevant source codes. Warning, the thread available through the Google Groups is censored.

\textsuperscript{21} National Institute of Standards and Technology (NIST, http://www.nist.gov)
3.3.6. Trusted Computing (TC)

3.3.6.1. Introduction

This section reviews the Trusted Computing (TC) paradigm and its objectives. Various FPGA implementations of TC components are also highlighted. TC is an emerging technology which supports building trustworthy computing platforms. Thus TC concepts and objectives could contribute to the development of secure PR RC systems. Security of TC cryptographic protocols depends on the ability of the TC system to generate quality random data. Thus, various implementations of the Random Number Generators are also discussed.

The TC paradigm addresses information security issues in general purpose computing. TC standards are driven by the computing and communication industries through the Trusted Computing Group\(^{22}\) (TCG). TC aims to enhance the security of the target system by using the Trusted Platform Module (TPM)\(^{[84]}\). The TPM is an integral part of the target system which provides a set of cryptographic and security functions. TPM typically implements tamper-resistance techniques to prevent a range of physical and hardware-based attacks. TPM is capable of attesting, in a trustworthy way, certain system properties, thus establishing a system RoT. The RoT in a secure system is defined as a component that must always behave in a defined manner, since its misbehaviour cannot be detected. The RoT contains at least the functions to enable a description of the system characteristics (i.e. system state) that affect the trustworthiness of the system, e.g. loaded OS modules, device drivers, hardware state etc. In TC and RC, the RoT may be based on a tamper-proof hardware element within the FPGA fabric. The RoT in RC systems can be partially implemented within user logic (configware), to provide a flexible security mechanism.

A security policy is a description of implemented system protection policies. A trusted system, for example a RoT, is one where failure can break a security policy\(^{[25]}\). In the trustworthy system the security policy and RoT cannot fail. A recent compromise of the TPM\(^{23}\) shows that secure implementation of the trustworthy RoT is challenging. Skorobogatov examines data retention characteristics of modern SRAM chips as a function of temperature\(^{[22]}\). The study shows that some memory devices could retain 80% of data, up to one minute after power down. Skorobogatov also reports that retention time varies significantly between devices within the same family. Tuan, Strander and Trimberger report data residue analysis for 90nm SRAM FPGAs\(^{[23]}\). The report indicates the correlation between data retention time,

\(^{22}\) Trusted Computing Group (TCG, http://www.trustedcomputinggroup.org)

\(^{23}\) During the “Hacking the Smartcard Chip” talk at the BlackHat 2010 conference (Feb 2-3, Arlington, VA, USA) Christopher Tarnovsky from Flylogic Engineering demonstrated a successful attack on the TPM hardware. The attack requires physical access to the TPM module. An estimated attack cost is ~200k\$ and one man/day labour time of the reverse-engineering expert. Video from the talk can be found on the YouTube (http://www.youtube.com/results?search_query=Black+Hat+DC+2010%3A+Hacking+the+Smartcard+Chip&search_type=&aq=f)
memory cell structure and memory cell content. Logic memory cells also exhibit lower data remanence time than routing memory cells. Data remanence could be exploited in order to recover sensitive data, e.g. plaintext data or encryption keys which are buffered in RAM prior to encryption. Thus, since Virtex-4 FPGAs (officially available since Virtex-5) Xilinx have introduced the KEY_CLEAR primitive which supports instantaneous erase of the encryption key register from the internal configuration control logic, e.g. when device tampering is detected.

Trusted systems should be reliable and provide predictable behaviour, e.g. efficiently protect secret encryption keys, or generate truly random data etc. Secure implementation of the FPGA RoT requires resistance to various types of invasive and non-invasive attacks.

3.3.6.2. TC objectives

The main TC objectives are system authentication and data protection.

System and user authentication allows peers (users and system) who are using TPM-enabled systems to securely authenticate themselves when communicating over the untrusted channel, e.g. Internet. TPM supports trusted attestation of the system state, by producing a unique TPM-signed checksum of the system initialisation process, e.g. hardware state after reset and load of BIOS, bootloader, OS, device drivers and applications. During authentication, checksums received from the remote system are validated using a known public-key of the remote TPM. Checksums are also compared with a database of sequences known to be secure. If a match is found then the remote system is assumed to be trusted, e.g. its authenticity is confirmed and its state is secure (tampering is not detected). The TPM-based (“something you have”) authentication can be extended to include PIN/password (“something you know”) verification.

Data protection enables encryption of the user data, e.g. single files or full hard drive. TPM provides symmetric-key and public-key (RSA 2048) encryption primitives, hash generator (SHA-1) and True Random Number Generator (TRNG) which support generation of unique keys and one-time-used (‘nonce’) values. Cryptographic keys are generated and stored inside TPM, after the user authenticates himself to the TPM. Thus, the original TPM is required in order to decrypt the ‘sealed’ data and therefore TPM-based data protection provides secure ‘sealed’ storage for user data within the TC system.

The TPM specification is open [84]. Security of the TPM-based cryptographic algorithms is based on the tamper resistant TPM implementation and confidentiality of the internal Endorsement Key (EK). The EK is the main TPM key used to recognise a genuine TPM and protect TPM internal data. The EK is a key-pair composed of a public and private 2048-bit RSA key which is randomly created at manufacture time. The EK is read only and never leaves the TPM perimeter [85].
3.3.6.3. **TC in FPGA-based RC systems**

Chaves proposed Secure Computing Module (SCM) for RC systems [85]. Chaves examines TPM functionality which is beneficial to RC systems. The TPM-equivalent SCM implementation provides significant performance improvement when compared to genuine (ASIC) TPM devices. Chaves proposes the use of Residue Number System (RNS) in order to improve the SCM performance.

Eisenbarth et al. examine TC viability in RC systems and propose the inclusion of the TPM functionality within RC systems [86]. This would enable a scalable trusted computing base in the RC hardware and allow a standardised TC-based software binding with the hardware. The proposed solution would be also TPM vendor independent. A TC scheme for embedded RC systems has been also reported by Glas et al. [87]. Glas proposes a protection model built upon trusted configuration attestation of the RC system state. The model assumes the use of certified and thus trusted modules. The growing number and complexity of available Third Party IP cores increases the risk of undetected malicious interaction even between certified cores. This TC approach does not protect the privacy of IP cores installed in the system, since only the system credential memory is tamper-proof. Verbauwhede and Schaudt review methodologies for the design of secure electronic systems and propose the ‘tree-of-trust’ methodology [88]. The ‘tree-of-trust’ methodology supports recursive secure partitioning and integration on all levels of design abstraction, e.g. protocols, software, hardware and circuits. Schumant and Hwang also propose the use of parasitic effects in Deep Sub-Micron (DSM) technology in order to aid the design of secure circuits [89]. The sub-threshold power leakage, process variability, noise, signal integrity and power density could be exploited to increase DSM technology resistance to side-channel attack, support PUF design, provide random data, or mask power consumption.

Chaudhuri et al. propose the ‘FASE’ architecture [90]. FASE supports dynamic (coarse-grain) resource management using PR. FASE provides implementation evolution prior to PR in order to countermeasure DPA and fault-injection attacks. The authors do not provide any implementation results. Huffmire et al. propose the trusted overlay to COTS devices using 3D stacking technology [91]. Potentially this would increase the use of COTS in high assurance applications without the need to redesign the IC die.

3.3.6.4. **True Random Number Generators (TRNGs)**

Quality random bitstreams are required for Initialization Vectors (IVs), data block padding, challenges, nonces, and keys. This data is typically transmitted unencrypted and can thus be intercepted and analysed by the attacker. Thus, the trustworthy Random Number Generator (RNG) is a typical element in cryptographic applications, i.e. in TC. The Diehard [92] and NIST RNG [93] are industry-standard tests for assessing the quality of RNG bitstreams. Depending on the implementation, RNGs are categorised as Pseudo RNGs (PRNGs) or True RNGs (TRNGs).
Pseudo RNGs are software algorithms initialised with an externally generated sequence (‘seed’). PRNGs produce long sequences that appear to be random\(^{24}\). The initial ‘seed’ value and the internal state of the generator determine the next bit. Thus, PRNG always produces the same sequence when given the same ‘seed’ value.

True RNGs (TRNGs) base their output entirely on an underlying random physical process. TRNG does not have internal state and the output is based only on the physical process and not on any previously produced bits. Often the raw bits generated by the physical source are biased (the probability of a '1' is not 0.5). Thus bias reduction could be necessary. Sunar, Martin and Stinson provide and examine a theoretical model of a provably secure TRNG\(^{37}\). Other implementations of TRNGs have been also reported by Fischer and Drutarovsky\(^{94}\), Kohlbrenner and Gaj\(^{95}\), Schellekens et al.\(^{96}\), Simka et al.\(^{97}\), Dichtl and Golic\(^{98}\), Yoo et al.\(^{99}\), Vasyltsov et al.\(^{100}\), Wold and Tan\(^{101}\), Maiti et al.\(^{35}\), and Drimer\(^{102}\).

3.3.7. RC System Integrity Maintenance

In order to protect design integrity and design tampering of non-PR designs, configuration bitstream encryption is a viable option for non-PR designs which has been adopted by EDA tools and FPGA vendors, e.g. Altera, Xilinx etc. Use of bitstream encryption disables configuration readback through external configuration interfaces, e.g. SelectMap and JTAG\(^{9}\). Protection measures in Virtex-II Pro include blocking the ICAP function when bitstream encryption is used\(^{44}\). Thus, PR and bitstream encryption are mutually exclusive in Virtex-II devices. In modern FPGAs PR and bitstream encryption can coexist, but Xilinx advises against the use of ICAP, e.g. when design IP protection is required or in high-assurance systems design\(^{47}\),\(^{9}\),\(^{33}\). For PR designs, the existing ICAP does not fully protect against unrestricted FPGA configuration memory read back, e.g. due to an exploitable error in the user application implementation.

McLean and Moore report the FPGA-based Single Chip Crypto (SCC) methodology which is developed by Xilinx and NSA\(^{26}\). The SCC lounge is available only to authorised users and provides design and isolation verification tools, security IP cores (‘monitor’) and application notes\(^{25}\). McLean and Moore report advantages of PR in Information Assurance applications and highlight the need of post-implementation design verification, e.g. validation of used FPGA routing and module isolation. They also propose secure BMs and support for a new design constraint (‘NOBOUNDARYCROSS’) in EDA tools which provide an isolation ‘fence’ between FPGA (PR) regions. An isolation enforcement scheme which uses ‘fences’ and a CRC-based tampering monitoring circuit has been patented by Lesea and Drimer\(^{103}\). SCC targets High Assurance Internet Protocol Encryption (HAIPE) and High Assurance Internet Protocol Encryption

\(^{24}\) PRNGs were commented by J. von Neumann: „Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.”

Interoperability Specification (HAIPIS) which is a NSA secure internet protocol specification [104]. HAIPIS permits enclaves equipped with compliant gateways to communicate securely over untrusted networks [105]. An example of SCC-compliant design and further references to FPGA-based secure designs are described in [106].

Pittman and Forin examine eMIPS slot-based implementation of the security model for reconfigurable hardware [107]. The eMIPS is a dynamically extensible processor which contains reconfigurable extension slots. Pittman and Forin use two-level verification of the extension IP core trustworthiness. In the first stage eMIPS checks IP core digital signature. During the second-stage eMIPS use a software Design Rules Checker (DRC) to verify whether the extension IP core is compatible with eMIPS layout, e.g. matches extension slot region and slot interfaces, and does not contain malicious configuration, e.g. short-circuits etc. DRC uses IP core bitstream to build a circuit graph which is equivalent to the IP core netlist which checks for the occurrence of malicious sub-graphs from the DRC database. The eMIPS allows the use of multiple IP cores since the DRC verifies their structure prior to reconfiguration. Pittman and Forin do not discuss the DRC trustworthiness. Also eMIPS DRC implementation is not detailed.

Drimer proposes FPGA bitstream authentication which provides cryptographic-level assurance of configuration data integrity and authentication of the bitstream source [108]. Authentication supports tamper-evident operation of FPGA systems. This allows the bitstream to remain open and subject to public audit and scrutiny, e.g. in voting machines. Bitstream authentication has been already adopted in Virtex-6. Hori et al. reported similar work which targets authenticated-encryption for PR [109]. Badrignans et al. propose a monotonic-counter extension to the authenticated encryption scheme in order to mitigate the risk of a replay-attack during remote update of the FPGA system [110]. The counter is implemented in a tamper-resistant FPGA area and guarantees the uniqueness of the reconfiguration transaction. This countermeasures replay attacks.

Gogniat et al. propose the Security Architecture For Embedded Systems (SAFES) which exploits the Intrusion Detection System (IDS) [111]. IDS is a set of clock/circuit/bus/channel monitors used to detect and deter system tampering. When IDS detects abnormal behaviour (e.g. tampering) the system is reconfigured with a hardened version of the security primitive. Gogniat et al. do not discuss security countermeasures regarding tampering threads to stored (not-active) PR modules, which might compromise the proposed security model and affect the implementation results (power consumption, size, etc).

Jones implements a portable Single Event Upset (SEU) detection and correction IP core targeting Xilinx Virtex-4 FPGAs [112]. The IP core supports Single-Error Correction Double-Error Detection (SECDED) using continuous configuration scrubbing, and can be used as a deterrent against malicious configuration memory tampering. Heiner et al. demonstrate ‘self scrubber’ which enables simultaneous PR and configuration scrubbing [113]. Since Virtex-5 FPGAs Xilinx have provided the FRAME_ECC primitive which supports SECDED. Also, Dutt and Li propose the use
of randomisation and 2D Error-Correcting Code (ECC) structures for FPGA design tampering detection [114].

Design integrity protection schemes do not address IP-related issues, e.g. IP license restrictions enforcement and design IP protection in the deployed device etc. Also, integrity protection schemes assume unrestricted availability of the IP core configuration bitstream and source files. This limits the scope to unrestricted (fully-licensed) designs, thus hindering adoption of cost-effective third party IP cores which are distributed at a lower cost but include license restrictions.

3.3.8. Design IP Protection

Challenges in modern VLSI design IP protection are examined by Yuan et al. [115]. The authors assume that increasing design IP reuse forces engineers to cooperate with others and share their expertise, data, documentation and tools which support design IP infringement. This forces IPVs to mark their designs using, e.g. IP watermarking or design fingerprinting. During watermarking, the IPV modifies the IP core (sources or production files) in order to include a unique signature (watermark) which proves IPV copyright ownership during any legal action. Ideally, the watermark should be easy to detect and impossible to remove. The main challenges listed by Youan et al. are: restrain the design overhead (e.g. watermarking cost) and reliable soft IP, CAD tools and algorithms protection. Also, Kean [116] and Bossuet et al. [117] highlight the vulnerability of volatile FPGAs to IP piracy and reverse engineering, and propose bitstream encryption as a countermeasure. Nakanishi proposes another bitstream encryption scheme [118]. Lesea provides IP-centric context for some of Xilinx FPGA security features [6].

IP protection methods in FPGA designs can be classified into two main groups: low-cost security and high-end security.

Low-cost methods target mainly massive scale (industrial) IP theft. This ‘time-stopper’ approach may not prevent the efforts of motivated peers to obtain FPGA configuration details and to publish results [119]. “Security-by-obscurity” is considered an adequate hindrance only for resource- and time-limited attackers. The competitive price (or time to market advantage) dictates a minimal-volume (maximum individual cost) for which the attack pays off (i.e. is successful). Thus, low-cost protection schemes target high-volume markets, e.g. consumer electronics (infotainment devices, digital TV etc) where the device development cost (including security measures) is bound to remain minimal, and revenue is achieved by the economies of scale.

A separate class of low-cost design protection is design watermarking. Various watermarking techniques are reported, e.g. by Kahng et al. [120], [121], Qu [122], Jain et al. [123], Ziener and Teich [124] and Ziener et al. [125]. Christiansen et al. [126], [127] propose decoy circuits as viable countermeasure to reverse engineering.

26 uLogic’s “Debit” project (http://www.ulogic.org)
High-end protection includes methods aimed at providing security of the design IP against all but sophisticated attackers who have unlimited resources. IP protection provided by FPGA design software [128], [129] supports IP core licensing and Digital Rights Management (DRM), down to the design netlist-level. The software-level protection effort is augmented by configuration bitstream encryption, being a de-facto industry standard provided by FPGA-specific tools. Kean also patented an extension to the FPGA fabric which facilitates token-based design rights management in the FPGA [130]. The proposed scheme supports Virtual Application Specific Standard Product (VASSP) model where IPV controls the number of activated copies through the use of device-specific (unique) cryptographic tokens. VASSP does not support multiple IP cores within a single device. This approach is limited and does not support PR. When PR is used the attacker could, under certain conditions, intercept the plaintext content of the IP core by accessing it, using the ICAP for example.

Siripokarirom describes a scheme for IP core evaluation before purchase [131]. Security of the proposed scheme relies only on closed bitstream format. Also, Castillo et al. propose hiding encryption keys in unencrypted bitstreams [132].

Kuhn proposes the ‘TrustNo1’ cryptoprocessor [133] which protects applications against unauthorised execution and reverse engineering. Kuhn examines necessary hardware/software mechanisms which apply in multi-tasking systems. The proposed scheme includes bus data encryption and on-chip encryption key storage. A license revocation scheme and key management protocol are also outlined. Graf and Athanas report a multi-layer key management architecture which supports secure off-chip data transfers [134], [135]. The proposed scheme uses a tamper-proof cryptographic iButton device to authenticate the application user. The FPGA and iButton authenticate themselves in order to enable application access to the data in the external memory. Adi et al. propose a concept for design IP locking using a SmartCard [136]. Also Suh, O'Donnell and Devadas examine the architecture of a secure single-chip processor (AEGIS) and describe the techniques used to execute private and authenticated software from untrusted off-chip memory [137]. A proposed security model for trusted systems design is to trust the on-chip environment while assuming that the off-chip environment is untrustworthy. Similar work has been reported by Lee et al. [138], Edmison [139] and Mahar et al. [140]. Lee et al. propose an extension of generic processor architecture. The work of Edmison and Mahar et al. does not require such a modification and could be applied within existing devices. Simpson and Schaumont examine an off-line hardware/software authentication scheme [141], [142]. The proposed PUF-based challenge-response binding scheme enables the locking of a third party design IP to a particular FPGA fabric. The architecture of PUFs has also been examined by Lee et al. [143], Suh and Devadas [144], Guajarado et al. [145] and Maiti et al. [35]. A work similar to Simpson and Schaumont has been reported by Alkabani et al. [146]. Alkabani’s scheme exploits additional states in the IP core control FSM. Also, it requires the TA to be involved in device fabrication.

Zeineddini and Gaj propose a secure PR scheme [147], [148]. The scheme employs bitstream encryption and authentication in order to ensure its security.
PR bitstream content is not verified and is assumed to be harmless to the target platform. Zeineddini and Gaj suggest the use of dedicated volatile (battery-backed) memory to store the key used during PR bitstream decryption.

Guneysu et al. propose a scheme which exploits PR and the use of a Trusted Authority (TA) to provide secure transfer of the encrypted IP to the FPGA system [149]. The proposed scheme does not support systems which include more than one IP core. Also, the security of the installed IP core is not verified.

Drimer proposes a security protocol for the remote update of FPGA configurations [150]. The update protocol can be implemented on existing FPGAs. The proposed security model requires a tamper-proof package for both the FPGA fabric and the external non-volatile memory in order to ensure that a genuine configuration is loaded after power-up.

The reported research summarised in this section mainly focuses on cryptographic algorithms and secure protocols, thus neglecting the problem of post-PR IP core interaction with the system (or other IP cores). None of the above protection methods considers third-party IP cores as a security risk. Secure IP core implementation and IPV trustworthiness is taken for granted. Protection against design errors can only be achieved by trustworthy design verification. Even if the IP core source code is available, it is vital to assure that the EDA tools used to produce the FPGA configuration bitstream are secure. Testing can be used to show the presence of errors, but never to show the absence of errors [151]. Thompson [21] discusses this issue and concludes that “You can’t trust code that you did not totally create yourself. ... No amount of source-level verification or scrutiny will protect you from using untrusted code”.

The proposed SeReCon element facilitates trusted (in-system) analysis of the structure of a new IP core prior to RC system reconfiguration and verifies IP core spatial isolation. This provides run-time protection of the already-configured PR system from structural issues resulting from erroneously placed (or malicious) IP cores. The SeReCon-based IP core validation scheme also protects confidentiality of the IP core bitstream during RC system implementation and in the deployed device. The plaintext IP core bitstream never leaves the SeReCon security boundary, the SI and the User access only the encrypted IP core and. Also, the SeReCon design has an open architecture which can be audited, e.g. through public scrutiny.

---

27 Public audits are already adopted, e.g. in the Secure Hash Standard public competition (http://csrc.nist.gov/groups/ST/hash/sha-3/)
3.4. FPGA IP Core Licensing Models

3.4.1. Introduction

The IP business model describes methods used in order to provide revenue from the IP product or IP-related service. The IP business model is associated with an IP license model. An IP license model is the terms and conditions (or rights and restrictions) that are granted to an IP user and is defined by the IP business model [152]. A key objective in the IP market is to ensure that the IPV is profitably rewarded for the provided IP. The IPV typically targets a number of applications in order to increase the market share. This requires a number of IP licensing models in order to match user expectations. Xilinx Targeted Design Platform is an example of such EDA software diversity [153].

This section describes common software business models and the Xilinx LogiCORE IP licensing scheme adopted by the Xilinx Alliance Program which provides IP cores and EDA tools for Xilinx FPGA devices. Shortcomings of the time-based LogiCORE-based IP core business model are analysed, and support for transaction based and metered access business models is proposed.

3.4.2. Common Software Business Models

In the FPGA-based design, the IP core is a hardware implementation (application) of the data processing algorithm. In the SDR application illustrated in Figure 1- the IP core could be one of the modules in the PR region, e.g. error correction coder/decoder, data cipher/decipher etc. Thus the IP core is the hardware counterpart of the software application. Hohmann [152] describes the common software business models:

**Time-based access** allows the licensee to use the licensed software for a defined period of time. The user pays for the license in order to use the software. In a rental time-based model, the period of time is set when the license is generated (software purchase). Rental time-based models are becoming increasingly popular in certain industries, e.g. the software test automation and the EDA industries. Hohmann predicts that rentals will reach all market segments. Thus all available software could be rented. The business motivations for rentals are compelling and rental enables reaching a new market.

**Transaction-based** access associates a fee with each transaction. A transaction is a defined and measurable unit of work, e.g. activating the software. The software can be activated a pre-defined number of times before it becomes inoperable. This requires a trusted transaction counter which is typically included within the application. The transaction counter enforces the maximum number of licensed activations and disables the application when this number is reached. Transaction-based business models can be found within enterprise software.
**Metering access** business models constrain consumption of a defined resource, e.g. application run-time. A consumptive model uses a pool of resources that are consumed, e.g. time period. The software becomes inoperable when all of the pool resources are consumed, e.g. the time has expired. This business model requires a trusted resource meter, e.g. monotonic timer included in the application. The monotonic timer counts the total application run time and disables the application when the licensed amount of time has expired. This business model could be used in consumer products where the user is charged for the multimedia decoder life-time, e.g. audio or video stream length.

**Hardware based** business models associate the cost of the software with the hardware. The software can be provided without charge when the hardware is otherwise unusable, e.g. firmware. This model is based on the hardware property, e.g. number of CPUs, which affects the performance of the application and can be enforced as required to meet business needs.

**Service-based** business models focus on service provided to the user, not the software enabling serviced access. Service-based business models are often used with open source licensing initiatives, e.g. RedHat provides assistance in the installation, configuration, and operation of RedHat Linux distribution which is open source software. Other services could include education programs, custom development, integration services etc.

**Revenue based** business models charge a percentage of revenue obtained from using the application. The revenue based model requires potential users to identify the potential revenues or savings fee. This could make other business models more viable.

### 3.4.3. Xilinx LogiCORE IP Licensing Scheme

Xilinx LogiCORE IP license model provides free\(^{28}\) IP core evaluation and supports fee-based licensing. Evaluation licenses are provided with Xilinx EDA tools. Xilinx LogiCORE evaluation licenses support simulation-only evaluation and hardware evaluation\(^{29}\).

**Simulation-only** evaluation license: typically included with most ISE (CORE Generator) IP cores. The simulation-only license allows user interaction with the customisation GUI and generation of IP core simulation models. IP core evaluation in the hardware is not supported.

**Hardware evaluation** license: allows generation of the IP core implementation netlists and bitstreams which support time-limited IP core evaluation in the FPGA hardware. Evaluation bitstreams include a timer element which disables the IP core after a few hours of operation. The exact duration of the hardware evaluation time

---


\(^{29}\) Xilinx LogiCORE site ([http://www.xilinx.com/ipcenter/ip_license/ip_licensing_eval.htm](http://www.xilinx.com/ipcenter/ip_license/ip_licensing_eval.htm))
depends on the core. The timer element is automatically inserted by the FPGA vendor EDA tools, e.g. Xilinx ISE. IP cores provided with the Xilinx EDK software include a 14 month full hardware evaluation license. The hardware evaluation license for Xilinx ISE IP cores can be generated using the Xilinx website or can be requested through a local Xilinx Field Application Engineer (FAE).

Fee-based licenses can be obtained through the Xilinx website or requested through a local Xilinx FAE. A fee-based license is available as a project-locked license, site-locked license or as a full electronic license.

**Project-locked license**: restricts the use of the licensed IP core within a single project. This could be a single bitstream including multiple instances of the licensed IP core and targeting multiple FPGA devices, or multiple bitstreams including multiple instances of the licensed IP core targeting single FPGA device.

**Site-locked license**: restricts the use of the licensed IP core in an unlimited number of projects originating within a 5-mile radius of an address designated as the "Licensed Site".

**Full electronic license**: enables full access to a core. This includes the simulation model, implementation netlist generation, place and route, and generation of a bitstream. Full electronic license is generated Xilinx core-specific product lounge.

### 3.4.4. Mixed Business Models For FPGA IP Core Licenses

The licensing scheme used by the Xilinx EDA tools, e.g Xilinx Design Studio supports the time-limited and hardware-based business models. Xilinx EDA tools use the FlexLM license manager in order to enforce time-limited operation of Xilinx Design Studio and included IP core license restrictions, e.g. simulation only, site- and project-locking, time-limited hardware evaluation etc. The time-limited IP core hardware evaluation license uses a timer element embedded in the IP core bitstream. Thus the FPGA device support in the license enforcement is not required. The IP core specific firmware, e.g IP core drivers, example applications etc, are distributed using a hardware base business model. Thus their use is limited only to Xilinx FPGAs.

The FPGA IP core, like a software application, is a collection of ‘features’. Each of the features can be offered using different business and licensing models, e.g. the time-limited license could be used for the base set of IP core features (e.g. data encryption), while a transaction-fee or metered-access license would apply for additional, application-specific IP features, e.g. IP core resistance to side-channel attacks. The transaction-based and metered access business models could increase the use of IP cores in reconfigurable consumer devices. Both models require support to provide trusted measurement of the IP core activations and usage (life time).

---

30 Xilinx Product Download and Licensing site (www.xilinx.com/getproduct)
Couture and Kent describe current software and hardware licensing techniques [154]. The licensing architecture for IP core metered-usage is proposed. This requires IPV to possess the encryption key which is used to secure communication between the FPGA and the external trusted time-keeping module. The encryption keys are permanent and thus vulnerable to replay attack.

3.5. Thesis Proposition: Trusted Design Verification And Reconfiguration

3.5.1. Introduction

This chapter reviews the state of the art in RC security. A motivating example on security risks in PR FPGA systems is provided. Risks of the side-channel attacks and threat of rogue EDA software also are highlighted. Security countermeasures supported by Xilinx FPGA fabric are described prior to critical examination of the reported work on the RC system integrity protection. The IP theft countermeasures and the principle of IP licensing models are also described. This section concludes with the need for a Secure Reconfiguration Controller (SeReCon) and summarises the requirements of SeReCon.

The CB4CLE example indicates that even a simple IP core, occupying one CLB, can potentially share resources (and thus setup implicit communication channel) with a second IP core occupying any of almost 3000 FPGA tiles which is 21% of the mid-size Xilinx Virtex-5 FPGA.

In the trusted computing paradigm the Trusted Platform Module (TPM) acts as a RoT for the target system and provides capabilities for secure data storage, secure reporting of platform configuration measurements (e.g. trusted boot-up configuration assessment) and cryptographic key generation. Applications of TPM in RC provide RC system with capabilities of authenticated power-up configuration. This requires SI to be in possession of TPM EK (system security credentials), exposing third-party design IP to the SI. Eisenbarth et al. do not consider the loaded reconfigurable modules to be malicious, while Glas et al. checks only the destination address of the reconfiguration data and disable TPM element when error occurs. This leaves FPGA unprotected, with the IP core already loaded into configuration memory. Thus, a new model of TPM operation is required to eliminate system integrator from the TC chain-of-trust and to protect design IP upon system tampering.

Directions of research activity in the field of RC systems security are two-fold, focusing on system integrity protection and design IP protection. System integrity protection measures aim towards seamless integration of multiple externally-developed IP cores into a stable and trustworthy system. Design IP privacy protection must be ensured to commercially available third-party IP core vendors. This is not ensured where the system integrator is in full control of, and has
unrestricted access to, all design modules, including third-party IP cores. Current design IP protection methods focus on the confidentiality of the IP core implementation, mainly by using authentication and encryption protocols, though without considering the risks caused by including erroneous or malicious IP cores. This exhibits contradictory goals of system protection (integrity) and design IP protection (privacy). Also, no current design IP protection methods assume a system model which includes IP protection of untrusted third-party cores while guaranteeing system integrity, e.g. protecting against design errors in third-party IP. Thus, new IP-aware methods for development of trustworthy systems are required.

The provision of IP core transaction-based and metered access licensing models in addition to a protection model for PR systems could increase the use of IP cores in reconfigurable consumer devices. A trusted license enforcement scheme requires methods for reliable control of IP core utilisation in the RC system, e.g. enforcement of counted IP core activation and IP core run time metering.

The SDR application, its attack vectors and IP core business models constitute two security problems, namely:

**The secure reconfiguration problem:** “How to maintain integrity of the PR RC system?”

**The IP licensing problem:** “How to ensure IP license enforcement in PR RC system such that the IPV can: a) tie the IP core to a particular RC device, b) reliably meter and limit IP core usage (total lifetime and activations number) in the deployed system”. The IPV wishes to enforce license-restricted usage of the IP core in the deployed PR RC system.

This thesis proposes a novel Secure Reconfigurable Controller (SeReCon) element which facilitates trustworthy and IP-aware PR for FPGA-based RC systems. SeReCon performs trusted design verification in order to provide system integrity protection during PR. SeReCon also protects the IP cores against IP theft and enforces IP core license restriction in the deployed RC system. This supports IPV’s in adoption of the transaction-based and metered access business models.

SeReCon extends the TPM architecture in order to provide support for FPGA-specific secure reconfiguration problem (e.g. IP core placement and isolation) and to facilitate new IP business models in the FPGA design community. The architecture of the SeReCon element is described. Figure 1-a illustrates the block diagram of the SeReCon-enabled PR RC system. SeReCon (Figure 1-cde) incorporates novel algorithms for building a system Root-of-Trust (Figure 1-) and a two-phase integrity-maintaining self-reconfiguration process.
3.5.2. Secure Reconfiguration Controller (SeReCon)

This section describes the proposed Secure Reconfiguration Controller (SeReCon) architecture.

The proposed SeReCon IP core is an integral part of a PR system and guarantees the physical isolation of non-trusted PR modules and thus the integrity of the system configuration between reconfiguration cycles. SeReCon provides the required infrastructure for secure PR at a remote site, e.g. at the SI or User site. FPGA system integrity protection is maintained by controlled PR. SeReCon analyses IP core structure and detects if the intended PR might affect the system integrity and expose the FPGA to a malicious (unsecure) configuration. Attacks such as this could activate configuration memory read-back and further compromise FPGA system security through modification of for example the design routing or logic resources. If the IP core resource requirements comply with the current state of the FPGA system, SeReCon allows reconfiguration to take place though controlled access to the ICAP.

In order to guarantee the integrity of IP cores, SeReCon can perform on-line context analysis of PR modules (physical placement, usage, confirmation that the active modules interact only through legitimate interfaces). This mechanism protects system integrity during PR at a remote site and allows non-trusted (un-verified) modules to be used so long as they do not interfere with those currently active on the FPGA.

A novel algorithm is proposed for generating credentials in order to establish the secure RoT. SeReCon performs the requested system reconfiguration on behalf of the RC system software. SeReCon aims to protect the integrity of the RC system by mediating access to the ICAP and by analysing incoming reconfiguration requests during run-time. A two-phase self-reconfiguration (IP core installation and activation) process is implemented in SeReCon in order to improve performance of IP core activation.

A novel algorithm is proposed within SeReCon for IP core licensing and usage accounting, e.g. total runtime, number of activations etc, in a PR system. SeReCon facilitates new IP core licensing models, e.g. transaction-based and metered access, during the PR system life-cycle. SeReCon ensures license restrictions enforcement within the target FPGA-based RC system. This supports trusted license management which requires participation of the Trusted Authority party only during device certification. This reduces the chain-of-trust requirements in a multi-player design flow.
3.5.3. Methods And Assumptions

The following assumptions have been made in defining the SeReCon system model:

a) The FPGA device is trusted (i.e. Trojan-free) and provides hardware support for RoT (details are described in Chapter 4).
b) The RC system comprises a number of integrated IP cores, configured using PR.
c) The IP Vendor explicitly declares some of IP core resources to be used as its communication interfaces.
d) IP cores are not trusted and their placement on the FPGA cannot overlap each other.
e) Subliminal channel and side-channel attacks are not considered.

3.6. Chapter Summary

This chapter provides a motivating example on security risks within PR FPGA systems. The example illustrates the risk of implicit communication channels between IP cores in the PR RC system. The risk of side-channel attacks, the threat of rogue EDA software and the issue of malicious FPGA designs are also highlighted. The chapter reviews the state of the art in RC security. Security countermeasures supported by Xilinx FPGA fabric are described prior to critical examination of the reported work on the RC system integrity protection and countermeasures for design IP theft. The chapter describes the principle of IP licensing models and proposes use of new IP core licensing models, e.g. the time-limited license and metered-access license, which could increase the use of IP cores in reconfigurable consumer devices. The need for the trusted IP-aware RC system security countermeasures is also highlighted. The chapter concludes with the proposal of a Secure Reconfiguration Controller (SeReCon) and a summary of the SeReCon requirements.
Chapter 4. SeReCon Proposal: RoT and Usage Accounting In PR FPGAs

4.1. Introduction

This chapter considers the requirements of credentials storage in a secure RoT and the implementation of usage accounting for RC systems. Techniques for storage of RoT security credentials and usage accounting data in modern FPGAs are reviewed. The suitability and limitations of using SRAM configuration memory are discussed. Other non-volatile memory schemes for credentials storage are also reported. The chapter proposes and describes an extension to the Xilinx FPGA fabric to provide a tamper-proof hardware element which protects the SeReCon-based RoT credentials and usage data during power-up cycles. The EIDR element prototype implementation in a Virtex-5 LXT device (ML505 Board) is reported. The register-based EIDR control/status interface, which is implemented in the FPGA user-logic, is highlighted. This chapter also describes EIDR API functions, which are provided by the SeReCon EIDR driver. The associated multi-party RoT credentials generation process is proposed. The activities of SeReCon and various parties (e.g. SI, TA, IPV) during RoT initialisation are highlighted. The RoT credentials generation process supports public security audit of the RC device and guarantees exclusive and authenticated access to the sensitive part of the RC system security credentials for the legitimate system, e.g. SeReCon RoT. The SeReCon-based RoT is immune to credentials leakage as a result of a future successful attack on the TA.

4.2. Requirements Of Credentials Storage And Usage Accounting In A RoT

The TC defines a system’s RoT as a component that must always behave in the expected manner, because its misbehaviour cannot be detected. The RoT contains at least the minimum set of functions to enable a description of the platform characteristics that affect the trustworthiness of the platform\(^3\). Trustworthy operation of an RC system requires the RoT security credentials to remain confidential during the system lifecycle and to be available only to authorised system elements (e.g. RoT elements). Also, the integrity of the system usage accounting data, e.g. number of system activations, its lifetime and uptime since last restart, must be preserved (unchanged) during RC system power-down cycles.

\(^3\) http://www.trustedcomputinggroup.org/developers/glossary
Figure 1 illustrates the Base SeReCon FPGA Configuration (BaseSFC), loaded after system power up. The BaseSFC contains only the SeReCon IP core, the communication interface to control RC system configuration (using the SeReCon element) and the memory interface to provide non-volatile Local IP Storage (LIPS) (in external Flash memory). LIPS is a repository used to store IP cores after processing by SeReCon, for use during future RC system reconfigurations. The BaseSFC forms the RC system RoT and is assumed to be secure. The BaseSFC should not contain any proprietary (closed-source) IP cores. This allows the BaseSFC to be freely audited, either by the independent TA or under open public scrutiny. During an audit the independent TA confirms the correct implementation of the BaseSFC design provided by the SI.

The BaseSFC can be likened to a trusted OS boot-loader in the TC domain; the proposed SeReCon IP core serves reconfiguration requests (Figure 1- and Figure 1-), received from the RC system software layer through the communication interface (Comm IF), and interrupts the reconfiguration process when a potentially malicious configuration bitstream is detected.

Security of the RoT is built upon secure credentials which are confidential, random and device-unique RC system encryption keys and Message Authentication Codes (MACs) [25]. Generation of Credentials are used to:

- protect the integrity of the RoT and the installed IP cores (in both the LIPS and the configured FPGA)
- allow the User or IPV parties to authenticate the BaseSFC (and the RC system environment)
- guarantee secure communication between the BaseSFC and IPV, during installation of new IP cores.

The BaseSFC uses symmetric-key encryption in order to protect the LIPS (Figure 1-). Public-key encryption is used to provide a private and secure (authenticated and encrypted) end-to-end communication channel between the IPV and the BaseSFC.

The TA generates and installs the credentials within the SeReCon firmware, e.g. as the pre-initialised BRAM content in the FPGA configuration bitstream, available for a RoT CPU. The SI and TA agree on the credentials structure and the functionality of the API which provides access to the credentials data structures. It is assumed that the SeReCon-based RoT is not likely to change during a product lifetime. If such a change is required, e.g. during a BaseSFC system upgrade, new unique credentials must be generated and installed in the RC system by the TA to establish trust in the new BaseSFC.

4.3. Review Of Techniques For Storage Of RoT Data

The BaseSFC requires security credentials to remain protected during the system lifecycle. Also, the RC system usage data (system state) must be preserved during RC system power-down cycles.
Security credentials and system usage data cannot be stored in volatile memory such as Xilinx FPGA configuration SRAM. Commercially available SRAM-based FPGAs, e.g. Xilinx Virtex family, exhibit the property of a stateless power up configuration; the internal configuration memory is cleared during the power up sequence and the device enters the default ‘blank’ (unconfigured) state, waiting to be configured with a bitstream provided from the external source [44], [47], [9], [33]. The stateless power up configuration strategy ensures that a newly configured device configuration is not affected by the previous one. This renders any system usage accounting infeasible as all data on system usage is lost after power down. Stateless configuration exposes the FPGA to a Man-in-the-Middle (MIM) attack [110] where the attacker alters the active configuration bitstream with an earlier version (possibly erroneous and exploitable) during power-up configuration. A genuine FPGA configuration is undistinguishable from a bogus configuration, which is introduced by an attacker using a malicious environment (typically requires physical access to the device). The non-stateless configuration is offered only in Virtex-5 and Virtex-6, supported by a ‘warm-boot’ facility [9], [33]. Warm-boot allows ‘rolling-back’ to a default fail-safe FPGA configuration in the event of an unsuccessful updated bitstream configuration. The warm-boot configuration state register does not offer any data privacy protection and does not support authenticated access control to the configuration state. Thus, a warm-boot cannot support system usage accounting and cannot be used to store and preserve confidential system state (e.g. its security credentials) between FPGA power-up cycles.

A battery-backed memory configuration could be implemented to store SeReCon security credentials. This technique is used in Xilinx Virtex-II/4/5/6 and Spartan-6 devices to store bitstream decryption keys [44], [47], [9], [33], [155], [116]. Read/write access to this memory by the user design is not allowed in commercially available Virtex devices. This renders it unsuitable for use in SeReCon for credentials and system usage data storage, since SeReCon is a user design element.

Modern FPGA devices require a TA to generate and install security credentials directly within the SeReCon firmware and to encrypt the BaseSFC bitstream. Afterwards, the TA securely programs the decryption key into the RC system. This is required in order to protect the BaseSFC against reverse engineering [119], e.g. extraction of the sensitive part of the credentials from the plaintext bitstream. Also, the TA should use one-time, device-unique encryption keys to avoid compromising multiple devices, e.g. in the event of a single encryption key being leaked. When the TA is in possession of encryption keys used during BaseSFC encryption, the RoT security could be compromised through a successful attack on the TA. The feasibility of such an attack is based on the fact that the TA is aware of sensitive credential material or the BaseSFC bitstream encryption key of many devices.

In summary, it is vital to provide the RoT with the ability to distinguish between arbitrary FPGA power-up cycles, e.g. in order to protect the RC device against MIM attacks and provide FPGA fabric support for RC system usage accounting, required for counted (pay-per-use) and time-limited IP core license enforcing.
4.4. FPGA Fabric Extension For Tamper-Proof Storage Of RoT Data

4.4.1. Introduction

This section proposes and describes a novel extension to the Xilinx FPGA fabric to provide a tamper-proof hardware element which protects the SeReCon-based RoT credentials and usage data during power-up cycles. This section describes the ID Register (IDR)-based facility for TA-controlled generation of unique, random and partially-anonymous security credentials, entirely within RoT (BaseSFC) security perimeter. FPGA fabric elements (e.g. configuration control logic and configuration memory) and user-design elements (random number generator and control/status register wrapper) which support the IDR facility also are highlighted. Figure 4- illustrates the FPGA fabric block diagram, including the IDR element. IDR is an FPGA fabric extension for tamper-proof storage of RoT credentials and RC system usage accounting.

Figure 4- also illustrates the SeReCon-based RoT certification process flowchart, which provides a trusted BaseSFC and IP core protection for RC systems. This section also proposes an extension to the SeReCon IDR to provide support for usage accounting and for enforcement of IP core license restriction in a multi-party design and user environment for PR systems. The introduction of flexible IP core licensing schemes (e.g. time-limited or counted) supports cost-effective reuse of the third party IP cores. However, this requires reliable license enforcement mechanisms to be implemented in the RC devices. The location of the IDR within the RC system is illustrated in Figure 1- and Figure 1-.

A description of each FPGA security credentials and usage accounting elements is included below:

**ID Register (IDR)** is the proposed non-volatile FPGA fabric element, accessible as a hard-macro and implemented using SRAM technology with battery backup. The IDR supports instant memory scrubbing (“one-shot zeroisation”) upon receipt of a key-clear request (assertion of RST), e.g. the SysMon FPGA primitive [27] could trigger IDR ‘zeroisation’ when FPGA input voltage or temperature exceeds the threshold level (this could suggest device tampering).

**Configuration Control Logic (CCL)** configures and controls the FPGA fabric using a configuration bitstream delivered through a dedicated (CFG_IO) interface (e.g. JTAG, SelectMAP etc) [156]. CCL is embedded in the FPGA fabric and contains a keyed Hash Message Authentication Code (HMAC) algorithm [33] implemented in hardware, and used for bitstream authentication. HMAC generates the MAC [25] of the active configuration bitstream, e.g. the BaseSFC. CCL
generates the configuration clock\textsuperscript{32} (CFG_CLK) and indicates successful FPGA configuration (CFG_DONE_OK).

\footnotesize{Figure 4- Block diagram of the FPGA fabric including the ID Register (IDR) element.}

\textbf{Configuration Memory} (CM) stores the active FPGA configuration (user-design) during FPGA runtime. The simplest CM design is the BaseSFC. CM content is configured using the configuration frames delivered by the CCL packet processor \cite{9} during power up configuration process (Figure 1-).

\textbf{True Random Number Generator} (TRNG) provides a unique random bitstream (DATA) to the IDR. The TRNG is a SeReCon element. The TRNG contains a set of Ring-Oscillators configured to harvest randomness based on physical phenomena \cite{37}, \cite{35}.

\textbf{Control/Status Registers} (CSRs) is a BaseSFC register-file used to control the IDR hard-macro (through register read/write access). When the genuine BaseSFC is

\textsuperscript{32} Available description of the Xilinx FPGA Configuration Control Logic (CCL) is not complete. Xilinx FPGA User Guides and Xilinx patents cover only general overview and partial implementation.
active, the CSRs provide access to the stored RoT credentials (CREDENTIALS) and system usage data, e.g. number of system activations (AAC), system lifetime (LTC), system uptime (FRC). The CSR is a SeReCon element implemented in CM. In the SeReCon-based RoT, the sensitive part of the RoT credentials (STORAGE_KEY, PRIVATE_KEY) is always kept within the RoT security perimeter. The public part (PUBLIC_KEY) of the RoT credentials is not restricted and can therefore be transferred outside the RoT for device certification (e.g. the TA signs and publishes the device public-key) or authentication (e.g. the IPV uses the device public-key to encrypt the installed IP core).

4.4.2. IDR Support For Tamper-Proof Storage Of RoT Credentials During Powerup

Figure 4- illustrates the IDR block diagram, including elements which support credentials storage. This section describes each IDR element.

Key Register (KR) stores the MAC describing the active FPGA design when WRITE is asserted, e.g. the MAC of the BaseSFC during RoT activation. KR is hardwired to the HMAC module (Figure 4-). The state of KR is preserved between power cycles. EQUAL_EN is asserted when the MAC value of the active design matches the stored KR value. Assertion of EQUAL_EN indicates that the currently active design is authorised to access the Credentials Register. Assertion of the RST signal clears KR content and disables IDR.

Credentials Register (CR) stores DATA (e.g. sensitive BaseSFC credentials) when WRITE is asserted. CR content (CREDENTIALS) is preserved between power cycles. CR content is confidential and available only to the authorised design (e.g. BaseSFC), when EQUAL_EN is asserted. This enables continuous operation of the SeReCon-based RoT, without the need to re-establish trust after a power cycle or in the event of non-authorised system reconfiguration. Assertion of the RST signal clears CR content.

The proposed IDR method extends the authenticated configuration recently made available in Virtex-6 devices [33] (proposed by Drimer [108]). During the first time activation of SeReCon within a controlled and trusted environment (i.e. at the TA facility), a system-unique, random and partially-anonymous credentials value (DATA) is generated using a RO-based TRNG [35] technique. Since credentials are generated internally within the SeReCon-based RoT (Figure 4-), and locked with the BaseSFC MAC, the sensitive elements, i.e. private-key, storage key etc, never leave the boundary of the SeReCon RoT (or FPGA fabric).

Generation of a malicious (‘Trojan’) bitstream, having an identical MAC to the genuine bitstream could be used in a birthday attack [157]. Thus, this work assumes that the MAC values generated by HMAC are ‘collision-resistant’ [157], [38], e.g. the generation of a Trojan bitstream with a MAC value identical to the genuine (not modified) design is hard (not feasible in a reasonable amount of time). The IDR provides authenticated access to the CR content. During an IDR write access, input
data is stored in the CR and the MAC is stored in the KR. When IDR data is requested, the current KEY input (the MAC value of the current bitstream, e.g. the BaseSFC) is compared with the KR content. The CR content is available to the FPGA design only if the KR and KEY values match, e.g. when the MAC of the current design (e.g. BaseRFC configuration) is equal to the genuine MAC which was used to store credentials. The IDR content (security credentials) is generated locally, within the RoT (Figure 4-), and is non-permanent. This preserves the confidential state of the RC system when the power is off and ensures that the private part of the credentials is known only to the authenticated BaseRFC (which contains the SeReCon RoT), even after a power cycle. Non-permanent (battery-backed) storage supports prompt revocation of the credentials upon detected tampering (within a single clock-cycle). Thus, secure and trustworthy operation of SeReCon can be certified by a TA and also by public audit.

Figure 4- Block diagram of the ID Register (IDR) used for controlled generation of unique, random and partially-anonymous security credentials.

4.4.3. Extending IDR For System And IP Core Usage Accounting

This section describes the extension to the SeReCon IDR to provide support for enforcement of IP core license restriction in a multi-party design and user environment for PR systems and system usage accounting. To support time-limited or counted IP usage in a secure FPGA system, it is necessary to distinguish between particular FPGA power cycles, and further to conclude whether the IP core license life-time (located in the installed IP core file stored in LIPS) has expired. Otherwise the FPGA system is vulnerable to exploitation by an attacker using a replay-attack scenario [110]. This could lead to possible time-unlimited IP usage or FPGA configuration downgrading, e.g. use of the earlier (possibly erroneous) FPGA configuration. Figure 4- illustrates the block diagram of the Extended SeReCon IDR (EIDR). Three monotonic counters provide a mechanism for accounting device runtime for licensing purposes. A description of each element is included below.

The Authenticated-Configuration Counter (ACC) counts the number of
authenticated FPGA configuration events, e.g. the number of times that the SeReCon-enabled RC system has been powered up and authenticated. For each power-up, the ACC provides a value which is always increasing (monotonic) and guarantees that every restart of the device is unique (Nonce value). Therefore, a replay-attack can be detected. The ACC value can be used to enforce a pay-per-use business model [152].

The **Life Time Counter (LTC)** counts the total number of clock cycles (referred to as lifetime count) which have elapsed from the moment of SeReCon activation. The LTC content is preserved between system power cycles. This enables measurement of the IP core lifetime and enforcement of a time-limited usage policy, e.g. if the LTC value exceeds a predefined threshold value (embedded within the IP core license) SeReCon can automatically disable the IP core activity (see Chapter 6). The LTC can also provide a real-time clock for the RC system.

The **MsgNo Counter (MNC)** is a monotonic counter which contains the transaction message number (MsgNo). SeReCon asserts the UPDATE signal in order to increase the MNC content. MsgNo is a unique (Nonce) value included in IPV messages to SeReCon which are used, e.g. to establish the shared encryption key$^{33}$. SeReCon accepts IPV message only when the MsgNo included in the message matches the value stored in EIDR MNC. Always-increasing MNC updates is used to ensure the ‘freshness’ of the IPV messages sent to SeReCon, e.g. SeReCon rejects (old) messages with MsgNo lower than the EIDR MNC.

The **Free-Running Counter (FRC)** counts the number of clock cycles elapsed since the previous system power-up cycle. FRC is used to calculate the period of uninterrupted system activity and thus provides a mechanism for enforcing IP core license restrictions for time-limited hardware evaluation. Use of the FRC along with the LTC provides a distinction between system uptime and system lifetime and thus supports a flexible IP core licensing model.

ACC, LTC, MNC and FRC monotonic counters are cleared when the contents of the CR (Figure 4-) changes, e.g. when a new SeReCon RoT identity is generated by the TA or reset (RST) is asserted. Counter outputs are available only to an authenticated SeReCon design. If the FPGA is configured with a bitstream whose signature (MAC) does not match the current KR value (the MAC of the authenticated BaseSFC), the system remains inactive and the counter contents are not made available to the unrecognised (and inappropriate) FPGA design. This allows FPGA device reuse when the EIDR functionality is not required.

Assuming a 5-year lifetime for a device running at 100 MHz, the additional EIDR resources (ACC, LTC, MNC and FRC) required to support IP protection are as follows:

- Two 28-bit binary counters for ACC and MNC (assuming SeReCon power-up configuration and application requests occur at a maximum frequency of once per second)

$^{33}$ Chapter 7 describes implementation and usage of the RC system demonstrator.
Chapter 4 - SeReCon Proposal: RoT and Usage Accounting In PR

- Two 56-bit binary counters for LTC and FRC

The bit-width of these monotonic counters guarantees against counter overflow during the system lifetime. Counter values are available through the SeReCon CSRs (Figure 4-).

The state of the counters provides reliable device usage statistics. Licensing restrictions such as the maximum allowed number of IP core activations, the system usage-time limit for hardware verification purposes etc, can therefore be implemented.

![Figure 4- Extended ID Register block diagram. Monotonic-counters support enforcement of IP core license, time-limited and counted usage in a multi-party PR design environment.](image)

4.4.4. EIDR Prototype Implementation

This section reports the EIDR prototype implementation. The register-based EIDR control/status SeReCon interface implemented in the FPGA user-logic is
highlighted. The section also describes EIDR API functions, which are provided by the SeReCon EIDR driver.

An EIDR element prototype has been implemented in a Virtex-5 LXT device (ML505 Board) using the Xilinx ISE toolset\(^{34}\). Appendix B provides the reference EIDR source code in VHDL. Table 4- illustrates the EIDR resource utilisation and performance. Figure 4- illustrates the VHDL description of the EIDR element. Some details of the EIDR implementation are removed for clarity\(^{35}\).

The EIDR element connects to the SeReCon system bus through the Control/Status Register (CSR) wrapper (Figure 4-) which provides a bidirectional register-based interface between the EIDR and the SeReCon bus. Table 4- illustrates The EIDR API which is provided by the SeReCon EIDR driver. The SeReCon firmware (Figure 1-e) uses the EIDR API in order to access the RC system security credentials and system usage data. Details of EIDR API usage in SeReCon are provided in Chapter 6.

The \texttt{idr\_reset()} API function resets the EIDR. This clears all data stored in the EIDR registers, e.g. stored MAC, SeReCon credentials and SeReCon usage counters. The EIDR reset can be also executed when SeReCon detects abnormal system activity, e.g. when the FPGA SYMON primitive [27] signals the unusual FPGA devices temperature or input voltage level which could suggest potential attack.

The \texttt{idr\_init()} function resets and initialises the EIDR element (Figure 4-). After resetting the EIDR, SeReCon writes a new TRNG data value into the CR by asserting the WRITE signal. This also writes the MAC of the active bitstream, e.g. the BaseSFC, into the KR ACC, MNC, LTC and FRC registers are also cleared to their initial values. Typically SeReCon firmware calls \texttt{idr\_init()} only once, during RC system initialisation at the TA site.

The \texttt{idr\_status()} function returns current status of the EIDR element, e.g. uninitialised, active, tampered etc.

The \texttt{get\_idr\_credentials()} function checks the EIDR status and provides access to security credentials stored in the EIDR. The CR content is available through a pointer which is provided as a function parameter (\texttt{credentials\_ptr}). SeReCon uses security credentials to protect system integrity and IP core confidentiality during IP core installation, activation and deactivation.

The \texttt{get\_idr\_counters()} function checks the EIDR status and provides access to the EIDR counters. The ACC, LTC and FRC contents is available through a pointer which is provided as a function parameter (\texttt{counters\_ptr}). EIDR counter values are used during IP core activation and deactivation.

The \texttt{get\_idr\_msg\_no()} function checks the EIDR status and provides access to the MNC counter. The MNC content is available through a pointer which is provided as function parameter (\texttt{msg\_no\_ptr}). SeReCon uses the monotonic (always-increasing)

\footnotesize

34 Xilinx ISE Version 11.2

35 Appendix B provides a complete source code (VHDL) for the EIDR element.
Chapter 4 - SeReCon Proposal: RoT and Usage Accounting In PR

MNC value to enable unique identification of received commands. SeReCon accepts only commands with msg_no matching its current MNC value (subsequently incremented). This counters replay-attacks, e.g. where an attacker attempts to reuse (replay) old sequence of commands in order to subvert system configuration.

The `idr_update_msg_no()` function asserts the UPDATE signal and increments the value of the MNC counter.

When the EIDR is tampered (e.g. when attacker depackages the FPGA fabric and attempts physical access to the EIDR structure) or the design is not authorised to access its content, e.g. if the BaseSFC MAC does not match the EIDR KR value, the `get_idr_credentials()`, `get_idr_counters()`, `idr_update_msg_no()` and `get_idr_msg_no()` and functions return an error.

<table>
<thead>
<tr>
<th>FF’s</th>
<th>LUTs</th>
<th>IOs</th>
<th>Estimated Fmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIDR</td>
<td>427</td>
<td>549</td>
<td>685</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API call</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idr_reset()</td>
<td>None</td>
<td>Resets EIDR</td>
</tr>
<tr>
<td>idr_init()</td>
<td>None</td>
<td>Initialises EIDR</td>
</tr>
<tr>
<td>idr_status()</td>
<td>None</td>
<td>Returns the IDR status.</td>
</tr>
<tr>
<td>get_idr_credentials()</td>
<td>credentials_ptr</td>
<td>Updates the credential data structure using EIDR credentials.</td>
</tr>
<tr>
<td>get_idr_counters()</td>
<td>counters_ptr</td>
<td>Updates the counter data structure using EIDR counter values.</td>
</tr>
<tr>
<td>get_idr_msg_no()</td>
<td>msg_no_ptr</td>
<td>Updates the msg_no content using the EIDR value.</td>
</tr>
<tr>
<td>idr_update_msg_no()</td>
<td>None</td>
<td>Increases the content of EIDR MNC.</td>
</tr>
</tbody>
</table>

Table 4- The EIDR API which is provided by the SeReCon EIDR driver.
entity id_reg_top is
  Generic(
    MAC_WIDTH : integer := 256;--taken from V6CG
    CREDENTIALS_WIDTH : integer := 128;--useAES128
    ACC_WIDTH : integer := 28;
    MNC_WIDTH : integer := 28;
    LTC_WIDTH : integer := 56;
    UTC_WIDTH : integer := 56);
  Port ( 
    clk : in STD_LOGIC;
    rst : in STD_LOGIC;
    wr : in STD_LOGIC;
    done : in STD_LOGIC;
    update : in STD_LOGIC;
    mac : in STD_LOGIC_VECTOR (MAC_WIDTH-1 downto 0);
    data : in STD_LOGIC_VECTOR (CREDENTIALS_WIDTH-1 downto 0);
    credentials : out STD_LOGIC_VECTOR (CREDENTIALS_WIDTH-1 downto 0);
    device_restarts : out STD_LOGIC_VECTOR (ACC_WIDTH-1 downto 0);
    lifetime : out STD_LOGIC_VECTOR (LTC_WIDTH-1 downto 0);
    msg_no : out STD_LOGIC_VECTOR (MNC_WIDTH-1 downto 0);
    uptime : out STD_LOGIC_VECTOR (UTC_WIDTH-1 downto 0);
  );
end id_reg_top;

architecture Behavioral of id_reg_top is
  type state_type is (FSM_IDLE, FSM_INIT, FSM_ACTIVATED);
  signal state, next_state : state_type;
  signal kr_data : STD_LOGIC_VECTOR (MAC_WIDTH-1 downto 0);
  signal cr_data : STD_LOGIC_VECTOR (CREDENTIALS_WIDTH-1 downto 0);
  signal device_restarts_cnt : STD_LOGIC_VECTOR (ACC_WIDTH-1 downto 0);
  signal lifetime_cnt : STD_LOGIC_VECTOR (LTC_WIDTH-1 downto 0);
  signal msg_no_cnt : STD_LOGIC_VECTOR (MNC_WIDTH-1 downto 0);
  signal uptime_cnt : STD_LOGIC_VECTOR (UTC_WIDTH-1 downto 0);
  signal equal_en, auth_cfg_ok, auth_update, device_restarts_cnt_en : STD_LOGIC;
  signal auth_cfg_ok_en : std_logic;
  function is_zero(input_vector : in STD_LOGIC_VECTOR) return std_logic is...
begin
  EQUAL_EN_SIGNAL: ...
  AUTH_CFG_OK_SIGNAL: ...
  CREDENTIAL_ENABLE: ...
  LTC_ENABLE: ...
  AAC_ENABLE: ...
  FRC_ENABLE: ...
  AUTH_UPDATE_SIGNAL: ...
  MNC_ENABLE: ...
  KR : process (clk, rst) ... --KEY_REGISTER
  CR : process (clk, rst) ... --CREDENTIAL_REGISTER
  AAC: process (clk, rst) ... --AUTHENTICATED_CONFIGURATIONS_COUNTER
  LTC: process (clk, rst) ... --LIFETIME_COUNTER
  MNC: process (clk, rst) ... --MSG_NO_COUNTER
  FRC: process (clk, rst, done) ... --FREE_RUNNING_COUNTER
  ONE_SHOT_SYNC_PRO: process (clk) ...
  ONE_SHOT_NEXT_STATE: process (state, auth_cfg_ok) ...
  ONE_SHOT_OUTPUT: process (state) ...
end Behavioral;

Figure 4- VHDL description of the EIDR element. Implementation details are removed for clarity.
4.5. Multi-Party RoT Certification Process

The RoT certification process securely establishes a trustworthy RoT within the RC system. The TA only monitors and audits correct RoT implementation and the process of credentials generation. The TA does not generate or install security credentials itself. Credentials are used to authenticate the RC system, to provide a secure communication channel (authenticated and encrypted) to the IPV, and to protect IP cores installed in LIPS. The SeReCon-based RoT certification process flowchart, which provides trusted BaseSFC and IP core protection for RC systems, is illustrated in Figure 4-. This process generates the keys used for secure IP core transfer between IP Vendor and SeReCon.

The SI develops the hardware platform, installs the SeReCon IP core and sends the certification request and the RC device (along with BaseSFC design source files and binaries) to the TA facility for trustworthy initialisation and certification.

The TA performs the following tasks:

- audits the RC system hardware platform (FPGA fabric and peripherals)
- verifies the SeReCon firmware source code, the BaseSFC VHDL source code and the FPGA bitstream. The FPGA bitstream includes the SeReCon firmware binary (Figure 1-).
- ensures regulated environmental conditions (i.e. ambient temperature, stable FPGA voltage and clock signal etc) during the EIDR initialisation process.
- monitors the RC system during the first start-up in order to prevent the malicious initialisation process which could result in biased (not random) security credentials generated by the SeReCon.
- Certifies (signs) and publishes the SeReCon public-key to involved parties, e.g. IVPS, SI or User.

SeReCon generates credentials (e.g. master symmetric-key, public-private key pair, etc) using a TRNG and stores the credentials in the IDR. SeReCon also reports the public-key to the TA. Credentials authenticity is confirmed by the TA through signing and registering of the RoT public-key. The TA certifies and publishes the SeReCon public-key to all involved parties (IPVs, Users etc). IPV and other parties use the certified credentials to authenticate the SeReCon during its life-time. The SI and IPV use AES encryption in order to communicate with SeReCon using, e.g. one-time shared keys generated using Diffie-Hellman key agreement protocol [38].

The SeReCon-based RoT initialisation algorithm has three important properties:

- Initial assumptions guarantee exclusive access to the sensitive part of the credentials (private crypto keys, etc) only for the legitimate system, e.g. SeReCon RoT.
- The base configuration bitstream does not contain any credentials and closed-source IP cores. Thus it can be publicly analysed prior to audit by the TA in order to avoid vulnerabilities that might be introduced by the SI or third-party IP cores.
The private part of the RC system security credentials is revealed only to the authenticated base configuration (BaseSFC). Thus the SeReCon-based RoT is immune to credentials leakage as a result of a future successful attack on TA.

Figure 4- Certification of the SeReCon Root-of-Trust (including SeReCon firmware binary).

4.6. Chapter Summary

This chapter considers the requirements of credentials storage in a secure RoT and the implementation of usage accounting for RC systems. The chapter proposes and describes EIDR, a novel extension to the FPGA fabric, which provides non-volatile storage of RoT credentials and RC system usage data. Techniques for storage of the RoT security credentials and RC system usage accounting data in modern FPGAs are reviewed. The suitability of SRAM-based configuration memory is discussed. Other non-volatile memory schemes are also reported. The EIDR element prototype implementation in a Virtex-5 LXT device (ML505 Board) is reported. Appendix B provides the reference EIDR source code in VHDL. The register-based EIDR control/status interface (which is implemented in the FPGA user-logic) is highlighted. This chapter also describes EIDR API functions, which are provided by the SeReCon EIDR driver.

The activities of SeReCon and various parties (e.g. SI, TA, IPV) during RoT initialisation are highlighted. The RoT credentials generation process supports public security audit of the RC device and guarantees exclusive and authenticated access to the sensitive part of the RC system security credentials for the legitimate system, e.g. SeReCon RoT. The SeReCon-based RoT is immune to credentials leakage as a result of a future successful attack on the TA.
Chapter 5. FDAT Framework For Low-Level FPGA Design Analysis

5.1. Introduction

This chapter describes and demonstrates the FPGA Design Analysis Tool (FDAT), a host-based (off-line) bitstream analysis and low-level design verification tool which supports a Xilinx FPGA design assurance strategy and automated extraction and analysis of bitstream-level designs, within the PR design flow. Figure 1- illustrates the FDAT context block diagram. FDAT provides a number of generic APIs which enable automated generic access to the standardised XDL (Xilinx Design Language) Xilinx FPGA platform description format (common to all Xilinx FPGA device families). The FDAT GUI provides visualisation of design analysis results. FDAT is an extendable, Python-based system which exploits the functionality of dynamic languages and uses modular libraries of custom-defined analysis scripts.

The PR design flow offers opportunities for new applications [11], e.g. in the automotive industry [158], video processing [54], [159], etc. PR can expose FPGAs to security risks, e.g. malicious (Trojan/backdoor) designs [26], [65], [103], [57], design tools subversion [21], [160], etc. Reliability and security of reconfigurable systems must be ensured within a multi-party environment (Figure 1-), and in critical applications, e.g. aerospace and defence [160], telecoms [161], [162], surveillance [163]–[165].

Unlike software where computing resources (hardware) are managed by an operating system and software has no control over the hardware, IP cores necessarily have very fine grain control over the underlying hardware. If considering a PR methodology, companies must be confident of the quality and security of third party IP cores and of secure access to the FPGA reconfiguration area. The problem of implicit communication channels between PR RC IP design sub-modules has been discussed in Section 3.2. Any design error or imperfection in the design of an IP core could result in reduced system performance, application failure, or even a compromised system [2].

The development of an FPGA design assurance strategy at the level of the FPGA configuration bitstream, and related EDA tool support, offers a solution to the problem of implicit communication channels. Low-level design tools are increasingly required for RC bitstream debugging [119] (e.g detection of implicit communication channels) and IP core design assurance [20], [26], particularly in multi-party PR designs. While tools for low-level analysis of design netlists do exist [166] (e.g. FPGA Editor in Xilinx ISE toolset [48]), extended tool support which provides unrestricted, script-driven, design analysis and verification at the bitstream-level, supporting publicly available device data, is not available. Such a toolset
would find application in design assurance, e.g. design security and IP protection [150]. There is therefore an increasing demand for automated and customisable bitstream analysis tools [1] [5], [26].

The proposed FDAT framework enables analysis of interface compatibility between dynamic IP modules and an already configured FPGA design. FDAT also enables verification of user-defined design constraints, e.g. the required spatial isolation between IP cores. This chapter illustrates sample applications of the FDAT application, including bitstream-level design analysis of the Virtex-II Pro and Virtex-5 inter-tile routing and the verification of design spatial isolation. Results illustrate the FDAT tool capability for automated detection of potential external routing points within a design configuration bitstream. Such analysis of IP core bitstreams can be used to ensure that the IP core interfaces (external routing points) which are defined in the IP core PR bitstream are not compromised when combining with other IP units.

Analysis algorithms (Python recipes) developed for FDAT can be ported into embedded systems, such as SeReCon, to support secure IP core run-time management in self-reconfigurable systems. The SeReCon IP core, which is embedded within the FPGA design, enforces an IP core spatial isolation policy imposed by the requirement to protect FPGA system integrity during PR. This protects a PR system from structural issues resulting from erroneously placed (or malicious) IP cores. This chapter presents the results of porting of the FDAT-verified spatial verification algorithm to SeReCon the implementation of the SeReCon Embedded Routing Database (ERDB) and ERDB-based IP core analysis. ERDB is implemented in the SeReCon firmware and describes routing resources which are available in the FPGA device. SeReCon uses ERDB in order to detect implicit communication channels in IP cores which are provided by third party IPVs.

In the software domain, vendors publish and popularise their hardware architecture and instruction set. In the RC domain, FPGA vendors provide considerable but limited information on the internal device architecture and the format of the FPGA configuration bitstream (e.g. bit-wise description of the bitstream configuration frames). This limited publication limits the ability of the research community and third party EDA software vendors to perform low-level design verification, particularly within a PR design flow, to ensure both IP protection and IP security, and to support the increasing demand for secure IP development. Tools used in the FPGA design flow are typically vendor-specific, closed-source applications using proprietary file formats. This limits general user access to low level design information and requires development of in-house analysis tools, e.g. PyXDL [167] or ADB [168]. Arguably, the provision of open-source tools and detailed specification of proprietary file formats could benefit both system designers and design assurance researchers [21].

FDAT uses the Xilinx XDL file data, which provides a text-format (ASCII) data description of the design configuration for Xilinx FPGAs [17], [167], [169]–[173]. FDAT implementation is based on Xilinx published datasheets [9], [44] [46] and documentation [168], [174]–[176]. The quality of analysis provided by a bitstream
analysis tool such as FDAT is dependent upon the accuracy and completeness of information provided by the FPGA vendor.

FPGA architectural information which may not be included in the XDL report file (generated by the Xilinx xdl tool) introduces some uncertainty as to the correctness of the FDAT analysis. Steiner [174] discusses inconsistencies between the FPGA fabric description, FPGA Editor, XDL file content and Xilinx bxtest (unreleased) tools. Provision of complete information on PR-accessible FPGA device resources and routing data would support the development of a viable PR design assurance strategy and toolset such as the proposed FDAT system.

In applications requiring design assurance, e.g. security and design IP protection, design analysis is required at the lowest possible level, i.e. on the P&R netlist or the configuration bitstream [26]. Also, in applications exploiting a PR flow, a detailed knowledge of FPGA resources configured by dynamically loadable modules is important since the module source may be unknown (untrusted).

The structure of this chapter is as follows. Section 5.2 reviews a number of existing tools which facilitate access to low-level design descriptions, and proposes the desired functionality of a low-level FPGA Design Analysis Tool (FDAT). Section 5.3 describes the FDAT architecture and the script-based functionality which exploits advantages of the Python dynamic language. Section 5.4 presents the detailed implementation and evaluation of FDAT, a selection of FDAT recipes, and the FDAT algorithm execution time for analysis of Xilinx Virtex-II Pro inter-tile routing. Section 5.5 proposes porting FDAT functionality to the embedded SeReCon for on-line Xilinx Virtex-5 bitstream analysis (which is demonstrated in Chapter 7). Considerations in creating an embedded routing database and IP core routing analysis are highlighted in Section 5.5.3. Section 5.6 concludes the chapter.

5.2. Review Of Low Level Design Analysis Tools

5.2.1. Introduction

This section reviews existing tools which facilitate access to low-level design descriptions, e.g. placed and routed FPGA netlist or configuration bitstream, and highlights the need for low-level extended tool support. The desired functionality of a low-level FPGA design analysis toolset (such as FDAT) is proposed.

5.2.2. Low-Level Design Analysis Tools

Current FPGA analysis tools provide only a limited facility for FPGA visualisation (focused on user-logic or IO).

The Xilinx FPGA Editor [48] is a proprietary graphical application for Xilinx FPGA design visualisation and configuration. FPGA Editor analyses the Xilinx
Native Circuit Description (NCD) netlist file. The NCD file contains the design logic information mapped to components, such as Configurable Logic Blocks (CLBs) and Input-Output Blocks (IOBs). FPGA Editor does not support bitstream analysis. FPGA Editor offers limited, script-based, design-oriented, task automation through scripts and provides a visual representation of design placement and routing. The tool does not support analysis of the design bitstream (BIT) file and presents an abstract view of the FPGA fabric though does not display all of the information required to perform design assurance, e.g. IP core isolation verification analysis [57].

Xilinx JBits 3.0 SDK [166] contains class files for the creation of Run Time Reconfigurable applications and tools for use with the Xilinx Virtex-II architecture. JBits is an API to the Xilinx configuration bitstream, which allows Java applications to dynamically read, create and modify Xilinx Virtex-II bitstreams. JBits may be used as a stand-alone tool or as a foundation to produce other tools. The current JBits release supports Xilinx technology up to Virtex-II only. The proposed FDAT framework provides a subset of JBits functionality (design P&R is not supported in the current version), though with support for modern Xilinx FPGA devices.

Debit36 [119] is an open-source toolset, aimed at netlist recovery from an FPGA proprietary bitstream format. Debit supports Virtex II/4/5 and Spartan3 FPGAs. Debit decomposes the bitstream and extracts information about site configuration and PIPs. Debit does not support partial bitstreams or parse the FPGA fabric netlist and thus does not support structural analysis of the FPGA fabric architecture.

In order to address the needs of design assurance, Xilinx has developed the Isolation Verification Tool (IVT) [26]. IVT operates on placed and routed designs and performs design analysis, including verification of design spatial isolation, in order to track all hypothetical interconnects that can be created [30]. IVT targets design assurance, though is not publicly available (provided under the Xilinx Single Chip Crypto program [30]).

In summary, currently available tools either do not support automated low-level analysis of FPGA fabric resources or are not publicly available. Extended tools are required for the PR design flow to ensure both IP protection and IP security, and to support increasing IP development.

5.2.3. Proposed Functionality For Low-Level FPGA Design Analysis Toolset

Design assurance requires low level analysis of design files produced during the FPGA design flow. These files describe placed and routed designs and the FPGA configuration bitstream. For effective design assurance, it is vital to have access to information about the internal architecture of the FPGA fabric. In a general case, it is not feasible to obtain the complete documentation on the FPGA fabric. The XDL file

36 Available on the uLogic site (http://www.ulogic.org)
containing the netlist of all user-configurable resources and routing can be produced using the Xilinx ‘xdl’ tool.

In summary, a tool for low level FPGA analysis would:

a) be aware of low-level architectural details (netlist) of the target FPGA device
b) include an open and modular architecture in order to facilitate code inspection and customisation
c) provide read and modify access to the implemented (placed and routed) design netlist and output (partial) bitstream
d) support modern FPGA devices
e) provide a uniform API across device families and design input files (design and FPGA fabric netlists, bitstreams etc)
f) provide an interface for user-defined tool customisation and task automation (script-based functionality)
g) provide graphical visualisation of the design and FPGA fabric resources

5.3. Proposed FPGA Design Analysis Tool (FDAT)

5.3.1. Introduction

This section describes the detailed FDAT architecture and its script-based functionality which exploits advantages of Python, the Object-Oriented (OO) dynamic language. Figure 1 illustrates the FDAT block diagram and context. FDAT contains API modules FFAPI, DAPI and BAPI which provide access to information about the FPGA fabric, design, and bitstream, and access to a number of standalone Python scripts which define tool functionality. The functionality of FFAPI, DAPI and BAPI are described in turn.

The FDAT system aims to address the design assurance requirements listed in section 3.2. The main goal of FDAT is the support of low level analysis of the FPGA fabric architecture and design IP cores. This goal is addressed by the use of a set of APIs which abstract the user design, the FPGA programming bitstream (configuration frames and control commands), and the FPGA fabric.

The main advantages of using dynamic (scripting) languages in application development are: type-less programmability, rapid prototyping, code simplicity and ease of understanding. FDAT functionality is separated from the underlying implementation (API modules). High-level analysis algorithms define FDAT functionality which uses data sets and API methods exposed by framework components. The FDAT framework functionality is defined using a number of Python scripts, called “recipes”. Recipes implement high-level algorithms by gluing together the functionality provided by the FDAT modules. The use of a scripting
language and a loosely-coupled architecture increases the code portability and improves its application-oriented customisation capabilities.

The following subsections describe the FPGA fabric API (FFAPI), the design API (DAPI) and the bitstream API (BAPI).

5.3.2. FPGA Fabric API (FFAPI) Module

The FDAT FPGA Fabric API (FFAPI class) module provides an interface class to access information about resources available within the Xilinx FPGA fabric. The available data structures and most significant FFAPI class methods are depicted in Figure 5-; the method parameters and less important methods are omitted to aid clarity.

The relevant information is retrieved by parsing the EDIF-like FPGA fabric netlist file (methods grouped as XDLRC file parser), generated by the Xilinx xd1 tool in the device resource report mode (xd1 -report <device_name>). To the best of the authors’ knowledge, the generated XDL netlist is the only detailed description of the Xilinx FPGA fabric structure provided by Xilinx in a non-proprietary ASCII format. Xilinx Design Language (XDL) is a fully featured physical design language which provides read and write access to NCD files [17], [167]–[173]. XDL format is the default netlist format used by Xilinx tools. XDL enables users to create tools to address their individual FPGA design needs. The XDL file describes designs using human-readable ASCII syntax and provides detailed information about, e.g. physical layout of the placed and routed design, used instance names, primitive site configuration, routing details, hardware macro modules etc. The netlist file contains detailed information on available logic primitives, FPGA fabric routing, etc. While the netlist file is in human-readable text format, it requires automated processing and filtering to extract useful information.

The FDAT FFAPI module acts as a filter, processing an arbitrary Xilinx FPGA fabric netlist file. The information obtained from the FPGA fabric netlist is a superset of the data available in the Xilinx FPGA Editor tool. FFAPI extracts relevant data structures along with their context, i.e. logical location within the device hierarchy and physical location of the tile within the netlist file, and generates a small index file (generateFpgaTOC()) for the particular FPGA fabric. Logical location of the FPGA elements, e.g. Basic Elements of Logic (BELs), PIPs or routing wires, in the device hierarchy provides information such as

a) location within the FPGA row/column tile grid
b) parent tile for primitive site/wire/PIP
c) site type for pin/BEL

Maintaining records (within the index file) of the FPGA element offset within the file aids detection of undocumented netlist syntax. This also speeds up random access to the netlist information when caching of the pre-processed netlist file is not
be possible, e.g. when a new FPGA device technology is added or in embedded applications such as SeReCon.

![Diagram of data structures and FFAPI methods](image)

**Figure 5-** The available data structures and most significant FFAPI methods (the method parameters and less important methods are omitted to aid clarity).

The GUI module provides graphical visualisation of data representing FPGA internal resources. Resources are depicted in an array-like structure of tiles (Figure 5-a). All tiles are tagged according to their type and position. User tags can also be assigned to individual tiles. This allows group selection, i.e. tiles containing user logic resources, BRAMs, clock tree etc. Moreover, when combined with the data available from the Design API class (DAPI), visualisation of a user design is possible (Figure 5-b). The FPGA fabric view is different from that presented in the FPGA Editor and Debit tools due to the tile-centric organisation of the FPGA fabric data.

The recipe writer can directly use FFAPI class methods such as `getTileTypes()` and `getSiteTypes()` (Figure 5-) in order to extract information related to tile and primitive types respectively, available within a particular FPGA device. Methods `getTileDetails()` and `getSiteDetails()` provide detailed description of the internal structure of an arbitrary tile or site, using a mix of abstract data types such as dictionaries, lists, tuples etc. Methods `getGroupedPips()` and `getExternalWires()` are reference implementations used to categorise resource descriptions by property, e.g. the class property of the PIP connection or wire connectivity.

Methods available in the FFAPI class are organised into three functional groups based on the type of information provided, as follows:

**FPGA geometry:** Methods in this group provide information about the type of FPGA package, the number of rows and columns and primitive types.

**FPGA tile:** Methods in this group return information about the internal structure of the tile, namely available primitive sites, routing resources and PIPs.
FPGA primitive: Methods in this group provide information on the particular type of the site primitive, names, number and direction of the site’s I/O pins, site internal routing and components, e.g. BELs. This information is compliant with the view available in the FPGA Editor and documented by the FPGA vendor (Xilinx) [44], [27], [46], [9].

FPGA element: Methods in this group return information about the internal structure of the primitive element (BEL), namely the set of valid BEL configurations, and the number and direction of the element’s I/O pins and connections.

Figure 5- FDAT screenshots. a. the Tile view of the Xilinx Virtex-5 device (colour highlights tiles of the same type). b. the user-design (blue – FPGA tiles used as logic, yellow – FPGA tiles used for routing) and unused resources (grey) within the Virtex-5 device.

5.3.3. Design API (DAPI) Module

The FDAT Design API module provides access to the user-design netlist. Just before the FPGA bitstream is generated by the Xilinx bitgen application (during the final stage of the FPGA design implementation flow), the netlist describes the design as a set of FPGA family-specific primitives and interconnections. All used primitives are annotated with information about their placement at the dedicated physical primitive
site within the FPGA fabric. Also, design nets (inter-site connections) are tied to physical routing resources, namely pin-wires, wires and PIPs.

The DAPI class module has been proposed to facilitate unrestricted user access to already placed and routed designs. In order to gain access to the ASCII version of the design netlist file (.xdl) the Xilinx ISE xdl tool is used in the ncd-to-xdl conversion mode (xdl -ncd2xdl <design.ncd>).

The DAPI class implements an XDL syntax parser along with a number of interfacing and conversion methods. Figure 5- illustrates the available data structures and most significant DAPI methods (the method parameters and less important methods are omitted to aid clarity). The XDL description of the design is parsed and the FPGA resource usage is reported. The parser recognises design structures such as named instances of FPGA primitives, routing resources establishing a connection network and design-module definitions used in non-standard design flows, e.g. modular design and partial reconfiguration. Design structures are in the form of mixed abstract data types, e.g. dictionaries, lists and tuples. Design structures retrieved by DAPI (depicted in Figure 5-) are available as DAPI methods which operate on the following design dictionary:

**Design dictionary**: provides information about design name, target FPGA device and logic organisation of the design IO pins and buses. Information is available through the method getDesignInfo().

**Instance dictionary**: contains all FPGA primitives (sites) used by the design (indexed by instance name). Data available through getDesignInstances() describes the location of design sites along with configuration.

**Nets dictionary**: contains all design nets (indexed by net name). Data available through getDesignNets() describes the design signals along with sources (inpins), sinks (outpins), routing PIPs usage, and element configuration.

**Module dictionary**: contains all design modules used within the modular design flow and partial reconfiguration (indexed by module name). Data available through getDesingModules() describes the design modules along with internals such as IO ports, resources used, and routing.

**Tile dictionary**: provided by getDesignTiles(), getInstanceTiles() or getNetTiles() contains information (indexed by tile name) on all used FPGA tiles, design instances or nets, respectively. The returned dictionary is cross-referenced with Instances and Routing dictionaries.

**Site dictionary**: provided by getDesignSites() contains all FPGA sites used by design (indexed by site name), cross-referenced with the Instances dictionary.

The Tiles and Sites dictionaries contain redundant design information grouped by physical location. This grouping is introduced in order to speed up design analysis, targeting location constraints (rather than logical organisation).

To ease the process of design analysis, the DAPI class module also provides methods for spatial grouping of occupied resources (getDesignTiles(), getDesignSites() methods). This provides a different view on design resources used
within a relevant FPGA location, e.g. tile or site. This approach supports tile-by-tile analysis of the design.

![Diagram](image)

**Figure 5**: The available data structures and most significant DAPI methods (the method parameters and less important methods are omitted to aid clarity).

### 5.3.4. Bitstream API (BAPI) Module

The Bitstream API (BAPI) module of FDAT parses the Xilinx bitstream in order to decode FPGA configuration commands (see the section 2.3.3) and extract content of FPGA configuration frames (Figure 2-). Xilinx FPGA devices are typically configured by loading FPGA configuration data into the device configuration memory at power-up. To program configuration memory, instructions for the configuration control logic and data are included as packets in the bitstream (Figure 5-). Configuration packets extended with a bitstream header form the complete bitstream which is delivered to the FPGA device through one of its configuration interfaces [9], e.g. serial configuration interface, SelectMAP, SPI, JTAG etc.

The structural description of the configuration bitstream for a particular device is usually provided by the FPGA vendor. Documentation for all but obsolete devices (i.e. first generation of Virtex/Virtex-E devices) does not describe the internal structure of the FPGA configuration frames [44]. Results reported in [119], [174], [175], [177] provide only partial information. The use of FDAT and the BAPI class module allows discovery of the relation between the configuration of an arbitrary resource (logic, routing, IO etc) and the content of the configuration frames.

The BAPI class, depicted in Figure 5-, implements a bitstream parser (`bitstreamParser()`) and packet parser (`packetParser()`). The packet parser reads the bitstream file and decomposes it into packets on-the-fly. Partial bitstreams are also supported. Data packets are unpacked in order to extract the content of configuration frames and the address of their intended location within the FPGA device (`getConfigurationFrames()`). The packet parser emulates behavioural functionality and part of the internal structure of the FPGA configuration control logic [44]. All
configuration commands (and their parameters are available through the method `getRegisterTransactions()`). Methods `decodeFarAutoincSeq()` and `getFPGAGeometry()` are used for debug purposes, e.g. to reconstruct the frame addressing sequence used by FPGA control logic (in FAR auto-increment mode) and the geometry of the FPGA device, respectively.

The BAPI class also provides methods for decoding the bitstream header (`headerParser()`) and analysing the content of the frame by checking which bits are asserted and their location within the frame (`reportSetBits()`). This functionality is verified in the FDAT-based FPGA routing analysis test scenario which is described in section 5.4.

![Diagram of data structures and BAPI methods](image)

**Figure 5-** The available data structures and most significant BAPI methods (the method parameters and less important methods are omitted to aid clarity).

### 5.3.5. FDAT Function Recipes Exploiting A Script Programming Paradigm

This section describes the script-based functionality of FDAT which exploits advantages of the Python dynamic language. The framework functionality is supported by a number of recipe scripts. FDAT recipes are regular Python scripts operating on class objects provided by underlying components (FDAT modules) or other recipes. Recipe interaction with FDAT modules is depicted in Figure 5-. FDAT recipes can be considered as glue for module functionality in order to provide an automated design analysis flow. For example, the FDAT recipe, “Pip2BitMapping” described in the next section, uses FFAPI and BAPI modules to discover information about the configuration bits for all PIPs within an arbitrary FPGA tile.

Figure 5- illustrates FDAT recipe structural organisation and interaction with FDAT modules, other recipes and user interface. FDAT recipes are divided into two logical parts, namely the FDAT initialisation part and the FDAT functionality part. The FDAT initialisation part provides recipe dependency checking, similar to related functionality available within software packet managers from major Linux...
distributions (e.g., Aptitude in Debian\textsuperscript{37}). Execution of the initialisation recipe element is optional. Dependency checks performed prior to recipe execution verify that all conditions required for correct operation are satisfied, e.g., that the required versions of FDAT modules, external tools, and design netlist files (.NCD or .XDL) are available and that active sub-recipes are initialised etc.

![FDAT recipe structural organisation](image)

**Figure 5 - FDAT recipe structural organisation (left side) and interaction with FDAT modules, other recipes and user interface.**

The FDAT recipe structure is flexible in order to exploit advantages offered by the dynamic scripting language, and to aid the recipe writer. Depending on the application domain, recipes can be implemented as single functions or groups of functions. Functions can be further grouped into classes and modules, spanning one

\textsuperscript{37} Debian Aptitude Wiki (http://wiki.debian.org/Aptitude)
or more script files. This flexibility of implementation allows the recipe writer to apply implementation standards (coding patterns) most suitable for the target application domain and programming proficiency level. Moreover, recipes can be developed in other languages (e.g. C/C++, Java etc) and executed within the FDAT Python environment by using wrapper code generated by the interface compiler [178]. Integration of recipes (native Python scripts) within different languages (e.g. C/C++) is also possible.

FDAT also provides an interactive command interface to the Python interpreter. This allows rapid prototyping of the verification libraries by interactive combination and reuse of available recipes to capture the new algorithm function. This facility also enables fast, customised processing of design analysis results.

Figure 5- illustrates the layered model of FDAT recipes. FDAT recipes can be categorised into four groups, depending of recipe main role (e.g. low-level design access, design modelling, model transformation etc):

**Module-specific (low-level) recipes:** extend the functionality of the basic FDAT modules (FFAPI, DAPI, BAPI etc) by providing an additional set of low-level functions to support design analysis and data conversion.

**Transformation (mid-level) recipes:** used to define new abstract models of the FDAT system, e.g. component model for platform-based design [179], dataflow/actor model for communication systems etc. Transformation recipes also provide bidirectional conversion functions between the new and the base model of the system.

**Model-specific (high-level) recipes:** provide an API which is specific to the abstract model defined by the Transformation recipe, e.g. methods for manipulating reconfigurable modules (modification, relocation, error checking etc) within a PR design flow.

**Auxiliary recipes:** provide support for other recipes, e.g. the “zipselhe” recipe provides archiving (snapshot) and reuse of partial results of the FDAT analysis for debugging purposes. This type of recipe extends the general functionality of the FDAT framework. The auxiliary recipes do not contribute directly to the design analysis.

Since Python is an interpreted language, recipes can be dynamically generated and applied (executed) without the need for code recompilation. Also, lower-level recipes can be easily imported into advanced recipes (bottom-up algorithm composition) in order to build powerful analysis tools. This approach exploits the advantages of scripting languages and is naturally convergent with their programming paradigms. Recipes allow the FDAT user to focus on the behavioural part of the analysis algorithm rather than on its implementation details. Use of hierarchical recipe structures supports the definition of FDAT functionality at different abstraction-levels in order to maintain clarity of the analysis algorithms, e.g. for auditing or formal verification purposes. Recipes also allow the application of a straightforward trial-and-error methodology for obtaining new knowledge about the FPGA fabric, the design and its bitstream.
5.4. FDAT Implementation And Analysis Of Virtex-II Pro Routing

5.4.1. Introduction

This section reports FDAT implementation in Python. The FDAT GUI and visualisation front-end has been developed using the Python TkInter graphic framework. This section also provides results of FDAT application in the analysis of Virtex-II Pro configurable routing.

The FDAT framework has been implemented using Python. The GUI and visualisation front-end has been developed using the TkInter\textsuperscript{38} graphic framework. Evaluation of FDAT FFAPI and DAPI recipes has been performed on a FPGA netlist. The design XDL description and FPGA configuration bitstreams have been generated using Xilinx ISE tools operating in the standard ISE flow and targeting Xilinx XC2VP30 and XC5VLX50T devices. FDAT components are implemented and evaluated according to publicly available information about the Xilinx FPGA structure\cite{cite1,cite2},\cite{cite3},\cite{cite4},\cite{cite5}–\cite{cite11} and bitstream\cite{cite12},\cite{cite13},\cite{cite14},\cite{cite15}.

All API modules are implemented as separate classes with data structures based on abstract data types, e.g. sets, lists, dictionaries and tuples. Each of the API classes (FFAPI, DAPI, BAPI) contains a dedicated one-pass, top-down parser used for contextual processing of the input data file (FPGA fabric netlist, design netlist and bitstream, respectively). The use of automatic parser generators would simplify the FDAT implementation. However, this approach would require a complete formal grammar of the input data to be available a priori, e.g. syntax described using

\textsuperscript{38}Tkinter is Python’s de-facto standard GUI package (http://wiki.python.org/moin/TkInter).
Regular Expressions\(^{39}\) (regex) or Backus–Naur Form\(^{40}\) (BNF). While the detailed syntax of the configuration bitstream, e.g. the internal structure of configuration frames, is not published and the XDL file format described in Xilinx ISE 6.1 documentation is incomplete, all parsers have been implemented manually. Manual implementation provides full control over data processing and thus enables detection of undocumented data structures (e.g. “_DESIGN_PROP” and “_INST_PROP” XDL file properties) and commands (access to reserved registers and non-default reserved bits in the configuration bitstream. When the parser detects an unrecognised data structure, an exception is raised and the file position of the unrecognised statement is reported to the user, along with the processing context. Manual implementation also enables fine control over the parser memory footprint, which enables its use in embedded applications such as SeReCon.

The FDAT framework uses a two-level caching mechanism for both permanent and temporary data storage in order to speed up the retrieval of information from the FPGA fabric netlist (the netlist size can be in the range of several gigabytes). During the initial execution of the FFAPI module the FFAPI parser optionally produces persistent cache files containing parsed and compressed information about every tile within the FPGA fabric, along with the location index of its definition within the original netlist file. During design analysis, the required FPGA fabric information is retrieved from the persistent cache file and stored in a limited-size local memory cache. The advantage of this two-level cache is that the typical memory footprint of 20-60MB, regardless of the netlist size which can vary from a few megabytes to tens of gigabytes. Moreover, the relevant FPGA fabric information is available immediately since parsing is performed only once.

5.4.2. Results Of FDAT Application In The Analysis Of Virtex-II Pro Routing

This section reports the application of FDAT in the analysis of the Xilinx (Virtex-II Pro) FPGA routing configuration, e.g. identification of PIP configuration bits in the bitstream. A selection of FDAT recipes have been implemented in order to verify the correct implementation of the FDAT framework, e.g. the “Pip2BitMapping” reference recipe identifies PIP configuration bits, the “ShowDesign” recipe reports and visualises user design (logic and routing) within the FPGA fabric. The FDAT has been also tested using a set of XDL design files. To evaluate the correctness of the FFAPI and BAPI modules, a “Pip2BitMapping” reference recipe has been developed. The recipe flowchart (Figure 5-) algorithm determines the bit patterns within bitstream configuration frames (using the BAPI

---

\(^{39}\) Regular Expressions (regex) provide a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters (http://en.wikipedia.org/wiki/Regular_expression).

\(^{40}\) The Backus–Naur Form (BNF) is a grammar for expressing context-free grammars (http://en.wikipedia.org/wiki/Backus-Naur_Form).
module), corresponding to configuration of the FPGA element (e.g. PIP, BEL etc) provided as an input. In this section, inter-tile routing analysis is considered, e.g. configuration bit patterns of PIPs which connect routing resources in more than one tile, in order to detect possible implicit communication channels in IP core bitstreams. The algorithm generates a list of relevant bit locations (packet, frame and internal offset) for all tileName PIPs that can be used in inter-tile routing.

The list of inter-tile PIP’s (getExternalTilePips()) (Figure 5-) is extracted from the detailed structure of the tile (tileName), retrieved using the FFAPI method (getTileDetails()). An exhaustive search approach is applied in an iterative loop. A single PIP is configured (included in the design) during each iteration of the algorithm. The xdlFile is generated (genXdlFile()) from the template and converted (convXDL2NCD()) to the Xilinx NCD using the xdl tool (xdl xdl2ncd <pip_name.xdl>). In order to create a partial bitstream the Xilinx bitgen tool is executed (genPBitFile()) with the NCD netlist file (ncdFile) provided as a parameter. The cfgFrames data within the partial bitstream file (pBitFile) is analysed using the getCfgFrames() BAPI method in order to extract relevant configuration frame content and the positions of the bits corresponding to the configured PIP (getPipBits()). The recipe iterates over the list of PIPs used in the inter-tile routing. On completion of each iteration, the PIP entry (pip) in the externalPips list is annotated with the position of the active bits. Partial results from the single iteration are accumulated, providing a list of possible switchbox configurations which use inter-tile routing. When all PIPs in the list are processed, a final text report is produced (report) which contains PIPs mapping (indexed by PIP name). Figure 5- illustrates a HTML version of the bit mapping table which is produced in order to enable visualisation of addressing patterns. Accumulated results for Virtex-II Pro are consistent with data reported by Hubner et al. [177] which are obtained using the JBits tool. The “Pip2BitMapping” recipe has also been used to obtain configuration bit patterns for Virtex-5 external PIPs. The reference dataset of PIP configuration bits for the Virtex-5 family is not provided. The “Pip2BitMapping” recipe is generic for all Xilinx FPGA families and could be used with other Xilinx devices, e.g. Virtex-4/6 and Spartan-3/6 due to the use of the technology independent fabric abstraction produced by the FFAPI module (derived from the data provided by the xdl tool).

Data obtained from the “Pip2BitMapping” recipe is used to detect active inter-tile routing. This type of analysis is important in design assurance and within the PR design flow. Active inter-tile routing can lead to the setup of covert communication channels, e.g. violation of spatial isolation between designs [26], [57]. This is detailed in Section 3.2).

It is assumed, in a similar way to that proposed by [119], that the FPGA configuration defined by the design bitstream is a superposition of configurations of all FPGA resources and that all of these configurations are independent. Thus, identification of configured (asserted) bits for a single configuration of the particular resource (PIP, BEL etc) reveals the corresponding configuration mapping.
Figure 5- “Pip2BitMapping” recipe flowchart. The recipe is used in the FDAT framework verification test.

Figure 5-a illustrates a bit mapping table which is used by FDAT to verify design spatial isolation. Figure 5-b illustrates bit patterns for various PIP classes. Frame content is analysed using the bit mapping table in order to obtain a list of configured PIPs. The list is then compared with the current system state in order to detect illegal routing, e.g. the configured connections which are not exported by the system as a communication interface. Upon successful analysis of the IP core, reconfiguration can occur. If an illegal connection is detected, then the reconfiguration process terminates.

The FDAT “Pip2BitMapping” recipe execution time is in the order of 11 hours for all inter-tile PIPs within the Virtex-II Pro CLB tile (around 3200 algorithm iterations), measured on a mid-class PC (1GB RAM, Intel Pentium Dual-Core 2.16GHz). Over 99% of this time is spent on conversion of XDL files to NCD format, and partial-bitstream generation (using Xilinx bitgen tool). The
“Pip2BitMapping” execution time depends only of the number of PIPs within the tile (here less than 3200). Thus the time will vary for different tile types.

The time devoted to bitstream parsing and XDL template generation is between 90-120 seconds. In order to speed-up the execution time, the recipe could be modified to accommodate multiple PIPs within a single bitstream, e.g. one PIP per CLB tile.

![Figure 5- a. the bit mapping table generated for CLB column PIPs within the Virtex-II Pro. b. bit patterns for various PIP classes (used by FDAT to analyse partial-bitstreams and verify design spatial isolation).](image)

The DAPI module has been tested using a set of XDL design files, with designs containing forced invalid XDL syntax. In all cases, the XDL parser correctly detects the introduced issue (XDL syntax error) either by returning details on the specific design structure or by raising an exception when the syntax error is detected.

The “ShowDesign” recipe reports and visualises areas (user-logic and routing) occupied by the design within the FPGA fabric (Figure 5-b). This can be used to assess logic utilisation ratio within the IP core.

The “ResourceSelect” recipe allows selection and grouping of FPGA resources based on arbitrary logic or regular expressions. Groups can be reported and highlighted within the FDAT GUI. Resources can be selected by their coordinates or properties. Also, user defined tags can be attached to groups. This recipe supports logical resource partitioning. The results of various recipe executions are demonstrated in Figure 3-, Figure 3-, Figure 3-, Figure 5-, Figure 5- and Figure 5-. For example, Figure 5-a illustrates the tile view of the Xilinx Virtex-5 device (colour highlights tiles of the same type). Figure 5-b illustrates the tile view of the User-
Chapter 5 - FDAT Framework For Low-Level FPGA Design Analysis

design (blue indicates FPGA tiles used as logic, yellow indicates FPGA tiles used for routing) and unused resources (coloured grey) within the Virtex-5 device.

The device-independent syntax of the FPGA fabric netlists and XDL files allows generic support for designs targeting other Xilinx FPGA architectures (e.g. Virtex-4/5/6, Spartan-3/6). Although FDAT provides device-independent access to the configuration bitstream structures, every device family (e.g. Virtex-II Pro, Virtex-5) requires a customised implementation of the BAPI module due to the differences in configuration logic.

5.5. Porting Of FDAT Functionality To The Embedded SeReCon

5.5.1. Introduction

This section proposes the porting of FDAT functionality to the embedded SeReCon for on-line bitstream analysis. FDAT enables the generation of the Embedded Routing Database (ERDB) which includes description of the FPGA fabric. Considerations in creating the ERDB are highlighted. The ERDB is a compact and size-optimised description of the FPGA routing which contains the description of the inter-tile routing shapes available in various FPGA tiles and relative locations of PIP configuration bits in the bitstream. The feasibility and accuracy of the ERDB-based IP core routing verification are demonstrated. Also FDAT recipes, describing the design verification algorithm using an abstract FPGA model, can be ported and reused within the SeReCon firmware.

5.5.2. Requirements For ERDB Implementation

Figure 1- illustrates a block diagram of an example PCIe-connected FPGA reconfigurable accelerator system which incorporates the SeReCon module along with a communication gateway and a number of IP cores. SeReCon controls all access to the PR region of the FPGA and loads requested IP cores from the local repository via the FPGA reconfiguration port (ICAP). System level drivers use PR to exchange IP cores during run-time, e.g. to modify system operation in response to application data traffic trends. The proposed FDAT tool can generate the FPGA routing database containing a minimal FPGA description which is embedded in SeReCon firmware and used by SeReCon for on-line verification of IP core routing. The maximum generated database size is limited since available memory within the FPGA package boundary also includes the SeReCon root-of-trust firmware. Analysis of the Virtex-II Pro bit-mapping table, produced by the “Pip2BitMapping” recipe, reveals regular patterns in the location of PIP configuration bits within bitstream configuration frames (Figure 5-). These patterns
can be exploited for online calculation of PIP configuration bit positions within the bitstream configuration frames. Online calculation allows balancing of the database memory footprint, at the cost of SeReCon performance. In a similar way, the database implementation could exploit the regularity of the FPGA fabric (2D array of tiles) and configuration bitstream structure (frame addressing, block types etc).

5.5.3. Embedded Routing DataBase (ERDB)

The C ERDB is a set of C source and header file which can be included in embedded RC applications, e.g. in the SeReCon firmware. SeReCon uses the ERDB to detect implicit communication channels in the IP core design (Figure 3- and Figure 3-). The Python ERDB is the FDAT zipshelf archive which is used to validate SeReCon-based IP core analysis. Figure 5- illustrates the FDAT “ErdbCGenerator” recipe flowchart which generates the ERDB C source and header files. The recipe uses the FFAPI module, temporary wire shape database (WS_DB) and PIP database (PIP_DB) in order to generate the ERDB (in C and Python). This section describes the algorithm used in the FDAT “ErdbCGenerator” recipe.

Figure 5- The FDAT “ErdbCGenerator” recipe flowchart.
5.5.3.1. GetRoutingShapes()

The \textit{GetRoutingShapes} algorithm is illustrated in Figure 5-. The function processes all tiles described in the FPGA fabric description file (\textit{FpgaFabricFile}), provided by the FDAT’s FAPI module. The algorithm produces wires shape database (WS\_DB) and PIPs database (PIP\_DB) which support ERDB generation.

For every tile a list of external PIPs is extracted (\textit{getExternalPips}) and added (\textit{addPip}) to the PIPs database (PIP\_DB). The external PIP is a PIP interconnecting at least one shared routing resource (e.g. the wire connected multiple FPGA tiles). The function also generates a list of shared wires (\textit{getPipWires}) and their \textit{taps} (\textit{getWireTaps}). These are converted to wire \textit{shapes} (\textit{normaliseWireShape}) and are added (\textit{addWireShape}) to the output wire shape database (WS\_DB). Wire taps are Cartesian coordinates of FPGA tiles connected by the shared wire. A wire shape is a set of wire taps normalised using the Manhattan distance metrics\textsuperscript{41} with the current tile (an ‘anchor tap’) set in the origin.

\textsuperscript{41} Manhattan distance definition (http://www.itl.nist.gov/div897/sqg/dads/HTML/manhattanDistance.html)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{The FDAT \textit{GetRoutingShapes()} function flowchart.}
\end{figure}
The FPGA routing, e.g., in Xilinx Virtex-5 devices, is non-homogeneous [41]. The wire shape (number and relative distribution of taps) depends on the wire group, e.g., ‘double-wires’, ‘pent-wires’ etc. The wire shape within the wire group is also irregular [173, 174], and depends on the wire location within the FPGA fabric, e.g., some wires have additional taps. Shape irregularity can be observed using e.g., the Xilinx FPGA Editor. Public FPGA documentation provided by Xilinx does not describe routing irregularities or its correlation with the wire location in the FPGA fabric.

5.5.3.2. CreateRoutingDB()

The FDAT createRoutingDB algorithm flowchart is illustrated in Figure 5-. The algorithm generates the ERDB PIPs database (ERDB_PD), the ERDB Routing database (ERDB_RD) and the ERDB Tile Groups database (ERDB_TG). The getTileTypePips function reads PIPs data (e.g., FPGA tile type, associated wire names and interconnection type) from the PIP_DB. The list of PIP tiles is compressed (function groupPipTiles) into the group and saved (addGroup) into ERDB_TG. The PIP and the list of its group indices are saved in the ERDB_PD. The algorithm (getWireTileTypes) reads the WS_DB file and produces (getShape) a list of normalised wire shapes and shape-associated tiles. Tiles which are associated with the wire are grouped (groupShapeTiles) and are added (addGroup) to the ERDB_TG using the grID index which is returned by the addGroup function. The function transposeShape converts the wire shape tap coordinates (relative manhattan distances to the anchor tap) to non-negative values. This ‘moves’ the shape origin point (0, 0) from the anchor tap (shape tap in the current tile) to the bottom-left shape tap. This reduces the number of wire shape entries in the ERDB, e.g., all wire entries which have different sets of tap coordinates (relative to the parent tiles) share the same (single) shape description (set of transposed taps). The function addShape stores the shape in the ERDB Wire Shape database (ERDB_WS) using the shID index which is returned by the addShape function. The function addRoute stores the wire shape entry in the ERDB (ERDB_RD). The wire shape entry includes grID, shID and the wire shape anchor tap.

5.5.3.3. GeneratePipBitData()

The GeneratePipBitData algorithm flowchart is illustrated in Figure 5-. The function exploits the “Pip2BitMapping” recipe in order to generate the relative locations of PIP configuration bits in the bitstream. The “Pip2BitMapping” recipe is executed (pip2BitMapping) to generate partial bitstreams (partial.bit) for all PIP tile groups (getGroupPips) and FPGA tile types (getTileTypeGroups) which are included in the ERDB_PD. For every PIP the configuration word offsets are normalised (normalisePipCfgWords) prior to storing PIP configuration (cfg) data in the ERDB.

---

42 This type of wires is introduced in Xilinx Virtex-5 FPGAs and connects FPGA tiles which are located 5 tiles away.
Pip Bits database (ERDB_PB). Word offset normalisation is required to reflect the relative vertical position of the PIP tile within the bitstream configuration frames, e.g. some PIPS in the CLB column appear only in certain rows or span multiple rows.

Figure 5 - The CreateRoutingDB() function flowchart. The algorithm generates the ERDB PIPs database (ERDB_PD), the ERDB Routing database (ERDB_RD) and the ERDB Tilegroup database (ERDB_TG).
Figure 5- The GeneratePipBitData() function flowchart. This algorithm exploits the “Pip2BitMapping” recipe in order to generate ERDB PIPs Bit database (ERDB_PB) which describes relative locations of PIP configuration bits in the bitstream.

5.5.3.4. GenerateErdbC()

The GenerateErdbC algorithm flowchart is illustrated in Figure 5-. The function uses Python ERDB to generate the ERDB headers and C source files (ERDB_C) which are included in the SeReCon firmware (Figure 1-e). The genErdb_H function generates the C header (erdb.h) which includes the ERDB data structures\(^{43}\) used by SeReCon. Table 5- illustrates the ERDB C source and header files (produced by the “ErdbCGenerator” FDAT recipe) which are used by the SeReCon firmware.

\(^{43}\) Appendix C illustrates the ERDB C header file (erdh.h) which includes declarations of ERDB C data structures.
Chapter 5 - FDAT Framework For Low-Level FPGA Design Analysis

Figure 5 - The $\text{GenerateErdbC()}$ function flowchart.

<table>
<thead>
<tr>
<th>File name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>erdb.h</td>
<td>ERDB header file which defines ERDB data structures (see Appendix C).</td>
</tr>
<tr>
<td>erdb_tg.c</td>
<td>Provides tile group database (ERDB_TG).</td>
</tr>
<tr>
<td>erdb_ws.c</td>
<td>Provides wire shape database (ERDB_WS).</td>
</tr>
<tr>
<td>erdb_pd.c</td>
<td>Provides PIP database (ERDB_PD) which includes PIP bits locations. The</td>
</tr>
<tr>
<td></td>
<td>wire name database, and PIP mode database are also included.</td>
</tr>
<tr>
<td>erdb_routing.c</td>
<td>Provides FPGA routing database which includes shape database of FPGA</td>
</tr>
<tr>
<td></td>
<td>shared wires.</td>
</tr>
<tr>
<td>erdb_layout.c</td>
<td>Provides Cartesian tile type layout of the FPGA fabric.</td>
</tr>
</tbody>
</table>

Table 5- ERDB C source and header files produced by the “ErdbCGenerator” FDAT recipe which are used by the SeReCon firmware.

5.5.4. ERDB-Based IP Core Routing Analysis

The ERDB provides a compact description of the routing resources in the FPGA device. This facilitates IP core bitstream verification within the embedded RC system, e.g. SeReCon. The ERDB-based IP core analysis examines the IP core bitstream in order to detect active external routing, e.g. wires which cross the boundary of the FPGA region configured by the IP core. The active external routing is typically documented by the IPV and describes the legitimate IP core I/O interface.
Undocumented external routing which could result from inaccurate or malicious use of EDA software, creates implicit communication channels. Risk can result from the use of a multi-party PR design flow and IP core implementation secrecy. The FDAT "GetBitPips" and “getExtWires” recipes demonstrate ERDB-based IP core external routing detection.

Figure 5- illustrates the "GetBitPips" recipe flowchart. This recipe reports all PIPs (including ‘fake’ always-on PIPs) in the IP core. The recipe uses ERDB and FPGA configuration frames (the cfgFrames parameter). The CfgFrames parameter is obtained from the IP core bitstream (bitFile) using the BAPI module (getCfgFrames). The tile configuration data is extracted (getTileCfg) for all tiles within the PR region configured by the bitstream. Non-PR bitstreams (i.e. bitstreams which do not participate in PR) are also supported. The getTileGroupPips uses the ERDB_PB in order to provide a list of external PIPs which is specific for a tile group, e.g. the number of external PIPs in a tile varies with tile type and also depends on physical location of the tile within the FPGA fabric. Tiles in the same tile group include the same set of PIPs. A tile can be a member of a number of tile groups. Thus the complete list of tile PIPs is a concatenation of PIP lists from all tile groups. For every PIP in the list of tile PIPs, the getPipCfg() function reads the PIP configuration bit templates from the ERDB_PB. The PIP Configuration Bit Template (CBT) includes the list of frame Minor Addresses (MNAs) [9], configuration word offsets (within the FPGA configuration frame) configuration word bit masks which must be set in order to ‘activate’ the PIP. For the ‘fake arc’ (always-on) PIPs the CBT is empty. If the PIP is a ‘fake arc’ (always-on PIP) or the bit pattern in the tile configuration frames (tileCfgFrames) match the CBT (checkPipBits), then the PIP is assumed active and is added (addBitPip) to the list (bitPips). The bitPips list is returned to the user (report).

Figure 5- illustrates the “getExtWires” recipe flowchart. The recipe exploits the “GetBitPips” recipe and uses the ERDB to detect and report IP core (bitFile) external routing which could be used to setup implicit communication channels. The “GetBitPips” recipe provides a list of IP core PIPs (including ‘fake arcs’). The calcCfgRegion function uses IP core configuration frames (cfgFrames) to calculate the IP core perimeter (region) and its location within the FPGA device. For every PIP tile, an envelope is calculated (calcShapeEnvelope). Figure 5- illustrates the relation between the IP core region, the tile envelope and wire shape taps. The envelope is the manhattan distance between the tile and region corners. Also, the wire shape, e.g. the number and distribution of taps, depends on the physical tile location within the FPGA fabric. Tiles which use the same wire shape form a tile group (similar to the PIPs tile group). The getTileShapeGroup function provides the wire shape index to the checkWireShape function in order to obtain wire shape taps from the ERDB_WS. The checkWireShape function uses the tile envelope in order to verify whether any of PIP wire shape extends outside the IP core region. If any of the wire shape taps (manhattan distances calculated from the current tile (anchor tap) exceeds the tile envelope the wire is added to the list of IP core external wires.

---

44 For clarity, this thesis assumes rectangular region shape.
(addExtWire). The process is repeated for all PIPs. The complete extWires list is returned to the user (report). The user compares the extWires list with IP core interfaces documented by the IPV. The existence of additional external wires which are not documented in the IP core interface specification provided by the IPV could suggest the presence of an implicit communication channel and therefore a possible security threat.

Figure 5- The FDAT "GetBitPips" recipe flowchart. This recipe reports all PIPs (including fake arcs) in the IP core. The recipe uses ERDB and FPGA configuration frames (obtained using the BAPI module).
Chapter 5 - FDAT Framework For Low-Level FPGA Design Analysis

Figure 5- The "GetExtWires" recipe flowchart. The recipe exploits the “GetBitPips” recipe and uses ERDB in order to detect and report IP core external routing which could be used to set implicit communication channels.

Figure 5- Relation between the IP core region, the tile envelope and wire shape taps.
5.5.5. Verification Of The ERDB Correctness

The correctness of the ERDB-based IP core analysis depends on the FPGA fabric model and the accuracy of the Configuration Bit Template (CBT)-to-PIP mapping. The FPGA fabric model is provided by the FDAT FFAPI module which uses the FPGA description generated by the Xilinx XDL tool. The CBT mapping is obtained using the FDAT Pip2BitMapping recipe. The recipe implementation assumes:

- configuration superposition, e.g. the CBT of a single PIP does not depend on other PIPs
- location invariant CBT; the PIP CBT is the same for corresponding PIPs in all tiles of the same type

Figure 5- illustrates the "BitPipsVerificator" recipe flowchart which verifies the accuracy of the ERDB-based “GetBitPips” recipe. The list of bitstream PIPs (bitPips) is compared with the reference PIP list (xdlPips) which is obtained (getXdlPips) from the reference XDL file (xdlFile). Both lists are split (getXdlTilePips, getBitTilePips) into sub lists (xdlTPips vs. bitTPips) in order to support tile-based comparison. If xdlTPips and bitTPips are not equal, e.g. when some reference PIPs are not detected or the bitstream contains ‘extra’ PIPs which do not exist in the reference set, then lists of missing PIPs (missingTPips) and extra PIPs (extraTPips) are included (updateMissingPips, updateExtraPips) in the global lists (missingPips, extraPips) and are reported to the user (reportDiffs).

Appendix D provides the detailed report of results obtained after "BitPipsVerificator" recipe execution. Table 5- illustrates fragment of the "BitPipsVerificator" report which shows the difference between the list of XDL PIPs and a list of bitstream PIPs (for the ‘INT’ FPGA tile type). The report includes a list of XDL PIPs which are not found in the bitstream and a list of bitstream PIPs which do not appear in the reference design. The difference between the XDL reference PIP set and the bitstream PIP set is limited to certain routing resources only, e.g. carry-chain routing (*_COUT* wires) in CLB tiles, clock-related routing (*CLK* wires) in various, mostly clock-related, tiles and global routing (LV* and LH* wires) in the routing (INT) tiles. Also, the visible correlation between the set of missing and extra PIPs in the bitstream, e.g. PIPs missing in some tiles appear as extra PIPs in other ones, suggests that the EDA tools have modified the design netlist prior to bitstream generation.

Table 5- illustrates the correlation between the ERDB-based bitstream analysis and the reference XDL file, which is reported by the FDAT "BitPipsVerificator" recipe. Results highlight >99% accuracy of the ERDB-based IP core analysis and minor (<0.22%) difference in the number of detected PIPs. The bitstream file

45 The report provides analysis results for the Virtex-5 LX50T FPGA design (the prototype of a SeReCon-enabled RC system).
contains 2621 (0.65%) extra PIPs which do not appear in the reference XDL design. Also, 837 (0.21%) of XDL PIPs are missing in the bitstream file. Further analysis of this problem is required and suggested as a future work.

Figure 5- The "BitPips Verificator" recipe flowchart which verifies accuracy of the PIP list generated using the ERDB-based “GetBitPips” recipe.
Chapter 5 - FDAT Framework For Low-Level FPGA Design Analysis

<table>
<thead>
<tr>
<th>XDL PIPs missing in the bitstream (BIT) file</th>
<th>Extra bitstream PIPs not included in the XDL file</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT tile type (6 pips):</td>
<td>INT tile type (6 pips):</td>
</tr>
<tr>
<td>LH0 = LH18 (131 tiles):</td>
<td>LH0 = LV0 (57 tiles):</td>
</tr>
<tr>
<td>LV0 = LH0 (23 tiles):</td>
<td>LH0 = LV18 (55 tiles):</td>
</tr>
<tr>
<td>LV0 = LH18 (44 tiles):</td>
<td>LH18 = LH0 (151 tiles):</td>
</tr>
<tr>
<td>LV0 = LV18 (242 tiles):</td>
<td>LH18 = LV0 (117 tiles):</td>
</tr>
<tr>
<td>LV18 = LH0 (66 tiles):</td>
<td>LH18 = LV18 (78 tiles):</td>
</tr>
<tr>
<td>LV18 = LH18 (38 tiles):</td>
<td>LV18 = LV0 (240 tiles):</td>
</tr>
</tbody>
</table>

Table 5- Fragment of the "BitPipsVerificator" report which shows the difference between the list of XDL PIPs and a list of bitstream PIPs (for the ‘INT’ FPGA tile type).

<table>
<thead>
<tr>
<th>Source</th>
<th>Total PIPs</th>
<th>Extra PIPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>.XDL</td>
<td>405869</td>
<td>837</td>
</tr>
<tr>
<td>.BIT</td>
<td>404990</td>
<td>2621</td>
</tr>
</tbody>
</table>

Table 5- Correlation between the ERDB-based bitstream analysis and the reference XDL file, which is reported by the FDAT "BitPipsVerificator" recipe.

5.6. Chapter Summary

This chapter discusses the risks in a multi-party PR design flow, and investigates the issue of implicit communication channels which could be created during PR using third party IP cores. The need for low (bitstream) level design analysis is highlighted and the FPGA Design Analysis Tool (FDAT) is proposed and applied as a solution.

To the best knowledge of the authors, FDAT is the first available toolset to provide high-level and unrestricted access to the low-level description of the Xilinx FPGA fabric and the user design at the netlist- and bitstream-level. The GUI front-end extends FDAT functionality by providing customised visualisations of the design and FPGA resources. Use of a Python programming language provides clean and self-documenting code (algorithm syntax), unrestricted tool customisation and defines higher-level abstractions for design analysis.

FDAT enables the generation of the ERDB embedded database containing a minimal description of the FPGA fabric and bitstream for use by the SeReCon IP core to perform on-line verification of IP core routing. The FDAT framework offers a generic and unified support for analysis of designs targeting all Xilinx architectures. The FDAT framework has been tested using Virtex-II Pro and Virtex-5 LXT designs and device descriptions.
FDAT has been developed around the concept of component and recipe separation. Components provide the necessary data and data (design/device/bitstream) abstract models, while recipes describe policies (algorithms) defining data model usage. This separation enables the reuse of high-level (model-specific) recipes which can be ported to other systems, e.g. the SeReCon IP core. Also, the hierarchical recipe structure supports a range of high-level analysis flows and offers virtually unlimited functionality extensions, thus supporting domain-specific design analysis.

The chapter proposes porting FDAT functionality to SeReCon for on-line bitstream analysis. Considerations in creating an ERDB (including FPGA fabric description) are highlighted. The feasibility and accuracy of the ERDB-based IP core routing verification is demonstrated.

The proposed FDAT framework offers increased productivity in low-level design analysis by seamlessly extending the FPGA design flow. Similar tools could be developed for other FPGA fabrics, i.e. Altera, Actel etc.
Chapter 6. SeReCon Initialisation And Operation For Secure FPGA Reconfiguration

6.1. Introduction

This thesis proposes the IP-aware SeReCon element which provides RC system integrity protection during PR. SeReCon is included in the RC system design and comprises both hardware element (IP core) and firmware. This chapter describes the internals (state diagram, the block diagram and firmware) of the SeReCon IP core. The SeReCon firmware stack is highlighted. This chapter also describes the operation of SeReCon RoT within the PR RC device during RoT initialisation, IP core installation, IP activation and IP deactivation.

The SeReCon RoT initialisation process occurs at the trusted TA site and includes EIDR credentials initialisation, RC system security credentials generation and publication. SeReCon exploits the EIDR element to provide design IP protection and executes in-system design analysis of new IP cores to maintain the integrity of the RC system.

SeReCon implements a two-phase RC system reconfiguration process which includes the IP core installation and IP core activation (RC system reconfiguration).

IP core installation is performed online, once for every new IP core. A SafeLock scheme for IP core security credentials protection is highlighted. The process of establishing the shared encryption key between the IPV and SeReCon, using the Diffie-Hellman (DH) shared key agreement protocol is also described. During the IP core activation process, SeReCon performs verification of the IP core compliance with the current RC system state in order to protect the integrity of the BaseSFC and to countermeasure the risk of implicit communication channel setup.

The IP core activation process is initiated by the RC system software. The main steps in the IP core activation process are illustrated. The verification of IP core compliance with the current RC system state is highlighted. The IP core license validation and RC system reconfiguration are also described. License validation prior to RC system PR enforces both transaction-based and metered-usage IP business models. The IP core deactivation process removes the remains of previously activated IP cores which could interact with the current system configuration, thus leading to RC system integrity issues. IP core deactivation ensures that the unused IP core configuration is removed from the FPGA configuration memory.
6.2. SeReCon Internal State Diagram

Figure 1-c illustrates the block diagram of the SeReCon IP core. SeReCon is a CPU-based embedded system which is almost entirely (except for the EIDR element) implemented in FPGA user logic to reduce the cost of modification to the FPGA fabric which is required in order to support the proposed RoT. The SeReCon element has an open source architecture which supports public code auditing. SeReCon hardware (Figure 1-c) includes a MicroBlaze CPU\(^{46}\) with local memory containing SeReCon firmware, general purpose IO registers which are used for communication (CommIF), a non-volatile (LIPS) memory interface (MemoryCtrlIF), hardware RNG (TRNG), symmetric-key encryption module (AES) \([36]\), internal configuration port (ConfigPort) and trusted security credentials register (EIDR).

Figure 1-e illustrates the SeReCon firmware stack which is installed in the CPU local memory (Figure 1-c). The firmware stack provides low-level drivers for SeReCon hardware elements, e.g. Board Support Package, ICAP, TRNG, EIDR, AES/ECC, Communication and Memory IP cores (MFS, SysAce, FATFS). Low-level drivers are used by the ERDB-based routing analyser (ERDB Analyser) and the ERDB Verifier for IP core verification prior to RC system reconfiguration. The Configuration Manager provides an abstract SeReCon API to the SDR application.

Figure 1- illustrates the SeReCon-based RoT and its usage scenario. The BaseSFC after power up contains only the SeReCon IP core, communication interface and LIPS interface. During RoT initialisation (Figure 1-b), the TA verifies the RoT implementation, certifies the device and the BaseSFC, and internally generates the RoT public-key. The SI uses the RoT and its public-key to install encrypted IP cores (Figure 1-c), obtained from the third-party IPV upon receipt of a request from the SI (Figure 1-a). The SDR User activates IP cores installed in the LIPS (Figure 1-d) by sending activation requests to the RoT (Figure 1-a) which validate the IP core resource requirements with the RC system current configuration. The integrity of the system is maintained by SeReCon through spatial isolation between components, constraining the IP core configuration data to predefined areas of the FPGA. IP core analysis is employed prior to reconfiguration in order to enforce this policy.

A two-phase self-reconfiguration process is implemented in SeReCon in order to improve the performance of the IP core activation (Figure 1-). During phase 1 the IP core is \textit{installed} in the system. SeReCon performs analysis of its structure and generates a resource report which becomes an integral part of the installed IP core. This approach speeds up the subsequent reconfiguration process. In phase 2, when IP core \textit{activation} is requested, SeReCon performs a controlled reconfiguration.

\(^{46}\) Open source CPU is an ultimate goal. The prototype implementation of SeReCon uses Xilinx MicroBlaze CPU as it is well integrated with the Xilinx EDA software. Since June 2008 MicroBlaze source code is not available (\url{http://www.xilinx.com/support/documentation/customer_notices/xcn08003.pdf}). Number of open source CPU designs is available from OpenCores (\url{http://opencores.org}), e.g. OpenRisc (or1k, \url{http://opencores.org/project/or1k}).
following verification of the available resources and interfaces for the required IP core and the current system configuration. Only IP cores verified by SeReCon can be downloaded and configured in the RC system. The two-phase reconfiguration algorithm supports IP core spatial isolation [26] and allows dynamic instantiation of physical isolation primitives [57]. The interface between SeReCon, IP cores and the rest of the system must be well defined so that activation of an IP core which eavesdrops on current IP cores can be prevented.

Figure 6- illustrates the SeReCon internal state diagram. After power up (POWER UP), SeReCon checks EIDR status (SERECON INITIALISATION); when EIDR is not activated (e.g. during the first power up), SeReCon waits for the TA initialisation command. If the EIDR is already activated, e.g. during a subsequent power up, SeReCon waits for commands from the SDR application (IDLE). Upon receipt of an IP core installation request, SeReCon installs the new IP core in the RC system (IP CORE INSTALLATION). The IP core activation request reconfigures the RC system using a previously installed IP core (IP CORE ACTIVATION), while a deactivation request removes the currently active IP core (IP CORE DEACTIVATION).

**Figure 6-** The SeReCon internal state diagram. SeReCon RoT Initialisation

### 6.3. SeReCon-based RoT Initialisation

#### 6.3.1. Introduction

This section details the SeReCon RoT initialisation process which occurs at the trusted TA site (see Figure 1-b). This minimises the risk of malicious device tampering (during initialisation) and enables independent scrutiny, e.g. through public audit.
6.3.2. Initialisation Algorithm

Figure 6 illustrates the SeReCon RoT initialisation flowchart. The initialisation process includes EIDR credentials initialisation, RC system security credentials generation and publication. The initialisation is successful if all steps complete without errors.

**EIDR credentials initialisation.** During EIDR credentials initialisation, SeReCon clears \( \text{resetEIDR} \) the Credential and Key registers (Figure 4-), and resets the EIDR counters (AAC, LTC, FRC, MNC). The TRNG element \( \text{genTrngData} \) generates a configuration bitstream containing random data which is used as the new EIDR credentials \( \text{rndCredentials} \) and is stored in the EIDR the Credentials Register \( \text{writeEidr} \). The BaseSFC signature (MAC) is also stored in the EIDR Key Register (Figure 4-). This provides access to the EIDR content (credentials and counters) only to authenticated design, e.g. BaseSFC which includes the SeReCon RoT.

EIDR credentials protect the unique RC system security credentials which are generated within the SeReCon RoT and are stored in the unprotected LIPS\(^{47}\). RC system security credentials include the private key and public key pair which is used in the SeReCon messages signing (using public-key cryptography) in order to facilitate RC system authentication to the IPV and to secure IP core transfer over the untrusted network (Figure 1-). SeReCon supports Elliptic Curve Cryptography\(^{48}\) (ECC) which uses the algebraic structure of elliptic curves over finite fields. Blake et al. [180] provide a thorough review of ECC mathematical foundations.

**RC system security credentials generation.** SeReCon uses the random TRNG bitstream \( \text{rndKey} \) in order to initialise \( \text{initPrivKey} \) the secret ECC private key \( \text{PrivKey} \) prior to its encryption \( \text{encPrivKey} \). SeReCon uses the secret \text{privKey} and standard (NIST-recommended), ECC parameters \([181]\) in order to calculate \( \text{calcPubKey} \) the ECC public key \( \text{PubKey} \).

**RC system security credentials publication.** During credentials publication, \text{pubKey} and \text{privKey} are stored in the LIPS \( \text{storeLipsFile} \). The content of the \text{privKey} file is encrypted using a symmetric-key cipher, (e.g. AES in CBC mode\(^{49}\)), with EIDR credentials serving as the symmetric-key and the initialisation vector (IV). Thus, RC system credentials are available only to the SeReCon RoT. The authenticity of an unencrypted \text{pubKey} file is certified by the TA (Figure 1-b) which makes it available to parties involved in RC system development (Figure 1-), e.g. SI, IPV’s, user etc.

---

\(^{47}\) This approach reduces the size of the EIDR Credentials Register.

\(^{48}\) The U.S. National Security Agency has endorsed ECC technology by including it in its Suite B set of recommended algorithms, and allows their use for protection of information classified up to top secret. (http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml)

\(^{49}\) Cipher-Block-Chaining mode (http://en.wikipedia.org/wiki/CBC_mode_of_operation)
6.3.3. SafeLock - EIDR Support For IP Core Privacy & Integrity Protection

IP cores installed in the RC system are encrypted. SeReCon uses random, core-unique security credentials (encryption key and IV) for every installed IP core. IP core security credentials are organised in a dynamic list structure which is referred to as the SafeLock. The SafeLock data is kept in the SeReCon local memory (Figure 1-a), entirely inside the RoT. The backup copy of the SafeLock data is encrypted using EIDR credentials and stored in the LIPS. During RC system power up, the SafeLock data is restored from the backup copy located in LIPS. SeReCon rewrites the SafeLock backup copy during SafeLock update, e.g. when a new IP core is installed and its credentials are added to SafeLock.

Table 6- illustrates the SafeLock API which is provided by the SeReCon EIDR driver. The SeReCon firmware uses SafeLock in order to access IP cores installed in LIPS.

The safelock_reset() call resets the SafeLock data. The SafeLock backup copy in LIPS is also rewritten. This call is used during EIDR initialisation.

Figure 6- The SeReCon RoT initialisation flowchart.
Chapter 6 - SeReCon Initialisation And Operation For Secure FPGA Reconfiguration

The `safelock_atomic_backup()` call creates an encrypted backup copy of SafeLock data using EIDR credentials. The backup copy is stored in LIPS. This call uses EIDR MNC and random data from TRNG in order to prevent file tampering. SeReCon includes the updated (`idr_update_msg_no`) EIDR MNC value (`msgNo`) and random TRNG data (`nonce`) in the SafeLock file prior to file encryption and storing in the LIPS. During SafeLock data recovery the `msgNo` and `nonce` from the backup file are compared with the original values. Any mismatch indicates tampering.

The `safelock_get_new_credentials()` call generates new SafeLock credentials for the IP core file and the license file, e.g. during IP core installation. This call updates the `credentials_ptr` contents using the SafeLock entry specified by the `msg_no` value. This is an atomic operation.

The `safelock_get_credentials()` call updates the `credentials_ptr` contents using the SafeLock entry for `msg_no` value.

The `safelock_update_license_credentials()` call generates new license credentials for the SafeLock entry specified by the `msg_no` value. This call uses the `safelock_atomic_backup()` call.

<table>
<thead>
<tr>
<th>API call</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>safelock_reset()</td>
<td>None</td>
<td>Resets SafeLock data. This also rewrites the SafeLock backup file.</td>
</tr>
<tr>
<td>safelock_get_new_credentials()</td>
<td>credentials_ptr</td>
<td>Generates new SafeLock credentials for the IP core file and license file. Updates the <code>credentials_ptr</code> content using the SafeLock entry specified by the <code>msg_no</code> value. This is an atomic operation (increases EIDR MNC).</td>
</tr>
<tr>
<td>safelock_get_credentials()</td>
<td>msg_no</td>
<td>Updates the <code>credentials_ptr</code> contents using the SafeLock entry for <code>msg_no</code> value.</td>
</tr>
<tr>
<td>safelock_update_license_credentials()</td>
<td>msgNo</td>
<td>Generates new license credentials for the SafeLock entry specified by the <code>msg_no</code> value.</td>
</tr>
<tr>
<td>safelock_atomic_backup()</td>
<td>atomic_ptr</td>
<td>Creates encrypted backup copy of SafeLock data. This is an atomic operation (increases EIDR MNC).</td>
</tr>
</tbody>
</table>

Table 6- The SafeLock API which is provided by the SeReCon EIDR driver.

Figure 6- illustrates the structure of the SafeLock backup file. The file includes the list of security credentials (encryption key and IV) which are used to access installed IP cores. The `ipCredentialsCnt` field provides the size of the list. Other fields (Magicfield, `msgNo` and `nonce`) are used in order to detect file tampering.
6.4. IP Core Installation

6.4.1. Introduction

This section describes the process of IP core installation within the SeReCon-enabled RC system. IP core installation is performed online, once for every new IP core.

Figure 1-a and c illustrate the SeReCon-based RoT activity during the IP core installation. The SI uses SeReCon RoT and RC system security credentials in order to install IP cores which are obtained from the third-party IPV upon receipt of a request from SI. SeReCon facilitates an ECC-based Diffie-Hellman (DH) key exchange protocol [157] and Elliptic Curve Digital Signature Algorithm (ECDSA) [182] in order to establish the authenticated secret Shared Key (SK). SeReCon encrypts subsequent communication with IPV, e.g. IP core transfer which is encrypted using the AES symmetric-key cipher [38].

Figure 6- illustrates The IP core installation flowchart and players. This includes the DH Shared Key (SK) agreement between the IPV and SeReCon, the IP core preparation, and the transfer and installation of the IP into the RC system. The DH SK agreement [157] requires a single message exchange between the IPV and SeReCon. This exchange results in a secret SK which is used to encrypt the IP core and its license. The IPV message and SeReCon reply do not require encryption. Thus, communication can occur over an unprotected network, e.g. Internet.

6.4.2. Shared Key Agreement Between IPV And SeReCon

This section describes the process of establishing the shared encryption key between the IPV and SeReCon, using the DH shared key agreement protocol (Figure 6-a). The SK is used in order to protect the IP core PR bitstream and the IP core license during transfer of IP over the unsecured network and non-trusted parties (Figure 1-). The SI initiates the SK agreement by sending the specification of the
requested IP core to the IPV. The specification describes the requested core functionality, target region for IP placement within the FPGA fabric and the location of required interfaces, e.g. BM location within the IP core region.

6.4.2.1. IPV-side DH Shared Key Generation

Figure 6- illustrates the IPV algorithm flowchart for the DH SK agreement which starts when the IPV receives a request from the SI (receiveIpRequest). On receipt of an acknowledge from the IPV (getSeReConCreds&msgNo), the SI
provides the RC system security credentials (SeReCon pubKey) and msgNo which is the current value of the EIDR MNC. The IPV uses the TA certificate in order to validate the pubKey authenticity. This can also be performed on-line (see Figure 6-). If the SeReCon pubKey is genuine, then the IPV generates a random bitstream (getRndData) which is required in order to generate (sign) the IPV message (the ECDSA signature of the msgNo\(^{50}\)). The use of the rndData ensures the IPV message uniqueness. The IPV message is bounded-to-msgNo and verifiable using the IPV credentials (public key). The IPV uses the SI in order to deliver the message to SeReCon (sendMessage) and waits for a reply from SeReCon (receiveSeReConReply). The SI also delivers the IPV credentials (public key).

The SeReCon reply is the ECDSA signature of the current EIDR state (msgNo2) signed using the RC system credentials (SeReCon privKey). The IPV uses the SeReCon pubKey in order to verify the signature authenticity. If the signature is valid and the msgNo2 is equal to msgNo + 1, then the DH SK is calculated (calcSharedkey) using the signature and rndData. The msgNo2 authentication ensures the reply message has been generated by the genuine RC system since only the SeReCon firmware has access to the privKey which is required during the sign operation.

![Figure 6- The IPV algorithm flowchart for Diffie-Hellman (DH) Shared Key (SK) agreement.](image)

\(^{50}\) Details of the ECDSA and DH key agreement are omitted for clarity. Wikipedia provides good examples of the ECDSA (http://en.wikipedia.org/wiki/ECDSA) and DH key agreement (http://en.wikipedia.org/wiki/Diffie–Hellman_key_exchange).
Message authentication protects the communication between the IPV and SeReCon against (communication) message tampering. Unique message marking (using msgNo) is a countermeasure to replay attacks, e.g. this makes the IP core readable only for an RC system with valid security credentials and matching EIDR state (‘synchronised’ MNC value). If the SI would reuse an old SeReCon reply with the previous msgNo2, the IPV SK would differ from the one calculated by SeReCon (due to the use of different rndData).

6.4.2.2. SeReCon-side DH Shared Key Generation

Figure 6- illustrates the SeReCon algorithm flowchart for DH SK agreement. SeReCon reads and verifies the IPV message prior to calculating the SK. The SK is encrypted using the EIDR credentials and stored in the LIPS. SeReCon generates and signs the reply message prior to saving it in the LIPS. The SI returns the reply message to the IPV.

SeReCon calculates the SK on receipt of a request from the SI (receiveSkRequest). SeReCon reads the IPV message file (readIgvMsgFile), verifies the IPV signature and compares the current value of the EIDR MNC (msgNo) with the value included in the message (IgvMsgNo). Signature verification is not required but could be used to enforce the RC system to accept IP cores only from a closed IPV list. This could facilitate vendor-locking in commercial applications.

If msgNo and IgvMsgNo match, then SeReCon calculates the SK (calcSharedKey) using the IPV signature and a random data bitstream (rndData) which is generated by the TRNG (getRndData). The calculated SK is encrypted using the EIDR credentials (encSK) and stored in the LIPS (saveSkFile). SeReCon updates the EIDR MNC (updateMsgNo) and generates a reply message which is the ECDSA signature of the msgNo, signed (sign) using rndData and the RC system security credentials (privKey) prior to storage in the LIPS (saveReply).

The message unique marking (using msgNo) protects against a replay attack, e.g. if the SI would reuse an old IPV message then the msgNo will not match the current SeReCon state (EIDR MNC value) and a reply message is not generated.

6.4.3. IP Core Production And Transfer To The RC System

Figure 6-a illustrates the IP core preparation flowchart. The IPV implements the IP core (implementIpCore) according to the SI specification (spec) and generates the PR bitstream with IP core placement constrained to the SI-defined FPGA region. IP core interfaces (IO), e.g. Bus Macros, are also defined in spec. The IP core license (genLicense) is merged (merge) with the IP core PR bitstream into IP package (IP core license + PR bitstream), and is encrypted (encIpPackage) using the calculated SK, prior to sending the IP to the SI (sendIpPackage). The SI receives the IP package, e.g. over the Internet, and forwards it the RC system, using an RC system communication interface (Figure 1-a), or stores the IP package directly in the LIPS (Figure 6-a, Figure 1-a).
Table 6- illustrates the IP core license modes and license restrictions which are supported by SeReCon. The IPV limits IP core usage, e.g. the number of activations and/or total lifetime depending on the agreement with the SI, as follows:

**Unlimited IP usage license**: unconstrained IP core usage. The IP core can be activated an unlimited number of times and will operate for an unlimited amount of time.

**Limited life-time license**: allows a predefined number of IP core activations. IP remaining lifetime decreases with time when the IP core is active in the RC system. The number of IP core activations is not limited. This supports a metered IP usage business model.

**Activation-limited license**: allows only a predefined number of IP core activations to be performed. The remaining number of activations allowed decreases each time a PR using the IP core takes place. The IP core becomes inoperable when the remaining number of activations (activation counter) reaches zero. This supports a transaction-based IP usage business model.
Combined license: enforces both a time-limited and an activation counted licensing schemes. The IP core is disabled when the remaining lifetime is zero or when the remaining number of activations reach zero.

Expired license: indicates an already disabled IP core. The IP core can be installed in the RC system but cannot be activated. This ‘dummy’ IP core installation can be used during RC system integration in order to verify RC system implementation and ensure correct communication with SeReCon.

<table>
<thead>
<tr>
<th>Activations limit</th>
<th>Lifetime limit</th>
<th>License description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xffffffff</td>
<td>0xffffffff</td>
<td>Unlimited usage – has no restrictions</td>
</tr>
<tr>
<td>0xffffffff</td>
<td>0x1-0xffffffff</td>
<td>Limited life-time license supports metered-based business model</td>
</tr>
<tr>
<td>0x1-0xffffffff</td>
<td>0xffffffff</td>
<td>Activation-limited license supports transaction-based business model</td>
</tr>
<tr>
<td>0x1-0xffffffff</td>
<td>0x1-0xffffffff</td>
<td>Combined license supports metered-based and transaction-based business models</td>
</tr>
<tr>
<td>0x1</td>
<td>0</td>
<td>Expired license. The IP core can be installed but not activated.</td>
</tr>
</tbody>
</table>

Table 6- IP core license modes and license restrictions which are supported by SeReCon.

6.4.4. IP Core Installation In The RC System

Figure 6- illustrates the IP core installation flowchart in the RC system. SeReCon uses EIDR credentials in order to access (getSharedKey) the SK (used to
decode the IP package), and to extract the IP core license and the PR bitstream (decIpPackage). The PR bitstream is analysed using the ERDB Analyser (callErdbAnalyser) which returns the IP core configuration region and a list of IP core external PIPs (extPips). SeReCon uses region and extPips in order to generate the IP core analysis report (genCoreReport). Also, new SafeLock credentials (safeLockGetNewCredentials) are generated in order to encrypt the IP core analysis report, license and PR bitstream prior to saving in the LIPS (saveReport, saveLicense and saveIpCore).

Figure 6- The flowchart of the IP core installation in the RC system.

The main goal of ERDB-based IP core analysis is to check if the external routing (external PIPs) match the system interface and the current system state. The ERDB Analyser also calculates the IP core isolation region which is an expanded configuration region that covers the external wires used by ‘fake’ PIPs.

Figure 6- illustrates the IP core isolation region and its relation with the IP core configuration region. The isolation region contains additional unused tiles in order to include all external always-on PIPs (‘fake arcs’). Use of the isolation region instead of the configuration region simplifies IP core analysis, e.g. the ERDB Analyser is not required to determine which always-on PIPs are used by the IP core.

Figure 6- illustrates the structure of the IP core analysis report. The report defines the IP core configuration and isolation regions and provides lists of external PIPs in both regions. The PIP entry includes PIP tile location (x, y) and indexes in the ERDB.
Figure 6- The IP core isolation region and its relation with the IP core region.

Figure 6- a. Structure of the IP core analysis report. The report defines IP core configuration and isolation regions. b. List of external PIPs in both regions.
6.5. IP Core Activation And Deactivation

6.5.1. Introduction

The IP core activation process is initiated by the RC system software, e.g. an SDR application which requests new functionality to be loaded in the RC system. Figure 6- illustrates the main steps in the IP core activation process. During the IP core activation, SeReCon checks the IP core license, the IP core usage restrictions (e.g. limited lifetime or limited activations) and verifies the IP core compatibility with the current state of the RC system (e.g. loaded modules) prior to reconfiguration. If any of these checks fail, then SeReCon aborts the activation process.

This section describes the IP core activation and deactivation in the RC system. The verification of IP core compliance with the current RC system state is highlighted. The IP core license validation and RC system reconfiguration are also described.

![SeReCon Diagram](Image)

Figure 6- Main steps in the IP core activation process.

6.5.2. Verification of IP Core Compatibility with Current System State

Figure 6- illustrates the data structure which describes the current state of the RC system PR region. The data structure includes the details of the PR configuration and isolation regions, e.g. region location within the FPGA fabric (Xmin, Ymin to Xmax, Ymax) and a list of external PIPs which provide the IP core interface (PIP ENTRY#1..N). SeReCon uses the Xilinx FPGA ICAP and ERDB Analyser in order to
detect external PIPs in the empty PR region within the BaseSFC (Figure 1-a). The perimeter of the PR region is predefined by the SI during system design and is hardcoded into the SeReCon data structure which describes the RC system state. IP core usage statistics (e.g. activation time and deactivation time) of the currently active IP cores are also included in order to enforce IP core deactivation when the remaining lifetime is zero.

During IP core activation, SeReCon updates the data structure with active IP core details\(^{51}\), e.g IP core name, msgNo and IP core activation/deactivation time (\(LTC\) start, \(LTC\) max). SeReCon uses the data structure which describes active IP core details in order to support IP core license restrictions enforcement, e.g. IP core deactivation when all of its lifetime ‘credit’ is consumed.

Figure 6- The data structure which describes the current state of the RC system PR region.

Figure 6- illustrates the IP core compatibility verification flowchart. SeReCon compares the current PR region state (see Figure 6-) with the IP core resource requirements which are obtained from the IP core analysis report (see Figure 6-). The comparison includes verification of the IP core configuration and isolation region perimeters (\(cmpC_{fgRegion}\), \(cmpIsolationRegion\)). This enables detection of BaseSFC configuration overwrites. Also, lists of IP core external PIPs (in both configuration and isolation regions) are compared with a list of allowed (interface) PIPs. This enables detection of possible implicit communication channels between the activated IP core and the current RC system configuration, e.g. BaseSFC and active cores. If any of IP core regions extends outside the PR region perimeter or additional external PIPs are detected, then the verification process fails and SeReCon interrupts the IP core activation.

\(^{51}\) This thesis assumes only one IP core within the RC system PR region. Research on dynamic isolation of multiple IP cores is encouraged and recommended as a future work.
6.5.3. IP Core License Validation

Figure 6- illustrates the IP core license checking flowchart. SeReCon verifies the number of remaining IP core activations (checkActivationsLimit) and the remaining lifetime (checkLifetime) prior to IP core activation (Figure 6-a). If the IP core license includes limits on the number of IP core activations and/or IP core lifetime (see Table 6-) then SeReCon decreases the number of the remaining activations (updRemainingActivations) and calculates the IP core deactivation time (calcDeactivationTime). The IP core name, the activation and deactivation times are written to the RC system state (updateSysState, see Figure 6-).

SeReCon also updates the IP core license if the IP core activation has been unsuccessful (updRemainingActivations), e.g. the number of activations is restored when the IP core footprint is not compatible with the current RC system state (Figure 6-b).

During the IP core deactivation (Figure 6-c) SeReCon decreases the remaining IP core lifetime by the time period for which the IP core was activated (updRemainingLifetime).

The attacker could circumvent the license restrictions by replacing the license file with its old version in order to ‘freeze’ the pool of available activations or
lifetime. Thus, SeReCon updates the IP core license key ($\text{licenseKey}$) ($\text{safelockUpdCreds}$) and re-encrypts the license file ($\text{saveLicenseFile}$) with a new license key.

Figure 6 - The IP core license checking flowchart. a. SeReCon verifies the number of remaining IP core activations and usage lifetime prior to IP core activation. b. SeReCon updates IP license during unsuccessful IP core activation. c. SeReCon updates IP license during IP core deactivation.
6.5.4. RC System Reconfiguration

Figure 6- illustrates the IP core activation flowchart. IP core activation requires msgNo which has been used during the IP core installation. SeReCon uses msgNo SafeLock credentials (getSafeLockCreds) and licenseKey (see Figure 6-) in order to decrypt the IP core license file (decIpLicense) prior to license verification (verifyLicense(ACTIVATE)). The IP core analysis report is also decoded (decIpReport) and validated with the current RC system state (verifyCoreCompatibility). IP core compatibility verification ensures that the requested PR does not overwrite the BaseSFC or active IP cores. SeReCon also checks if the IP core communication interface matches the RC system interface (IO) in order to eliminate the risk of implicit communication channels. The IP core activation process is interrupted and the IP core license update is ‘rolled-back’ (Figure 6-b) when the IP core contains additional external PIPs which are not listed in the RC system resources (see Figure 6-).

If the IP core does not contain additional external PIPs SeReCon decodes the IP core bitstream (decIpCore) and disables the IP core interface (disableBM) prior to reconfiguring the RC system (loadIcap) using the FPGA reconfiguration port and the IP core PR bitstream. SeReCon enables the IP core interface (enableBM) after successful reconfiguration.

Figure 6- The IP core activation flowchart.
6.5.5. IP Core Deactivation

In the RC system, different IP cores can occupy the same PR region over the lifetime of a system. Multiple IP cores can also share a single PR region. Thus, the IP core PR bitstream typically configures only a fraction of the PR region. The remains of previously activated IP cores, which are scattered around the PR region, could interact with the current system configuration, leading to RC system integrity issues. SeReCon avoids this by using IP core deactivation which ensures that the IP core configuration is removed, when unused, from the FPGA configuration memory. This also supports metered IP core usage, e.g. during the IP core deactivation SeReCon updates the remaining IP core lifetime.

Figure 6- illustrates the SeReCon-based IP core deactivation process. The RC application, e.g. SDR system, sends an IP core deactivation request to SeReCon. SeReCon uses the IP core SafeLock credentials in order to access the IP core license file prior to updating the remaining IP core lifetime (Figure 6-c). SeReCon also uses the PR region description (Figure 6-) in order to obtain the location of the deactivated IP core (configuration region) which is then reconfigured using default (empty) FPGA configuration frames.

Figure 6- The SeReCon-based IP core deactivation process.

52 SeReCon prototype supports only single IP core in the PR region. Extending SeReCon support for multiple cores is proposed as future work.
6.6. Chapter Summary

This chapter describes the operation of SeReCon RoT within the PR RC device during initialisation, IP core installation, IP activation and IP deactivation. SeReCon is included in the RC system design and comprises both hardware element (IP core) and firmware. The internal state diagram and the block diagram of the SeReCon IP core are also described and the SeReCon firmware stack is highlighted.

This chapter describes the SeReCon RoT operations during the RoT initialisation process which occurs at the trusted TA site in order to minimise the risk of malicious device tampering and support independent scrutiny, e.g. through public audit. The SeReCon initialisation process includes EIDR credentials initialisation, RC system security credentials generation and publication. SeReCon exploits the EIDR element to provide design IP protection and executes in-system design analysis of new IP cores to maintain the integrity of the RC system.

The processes of IP core installation is performed online, once for every new IP core. A SafeLock scheme for IP core security credentials protection is highlighted. The process of establishing the shared encryption key between the IPV and SeReCon, using the Diffie-Hellman (DH) shared key agreement protocol is also described.

The IP core activation process is initiated by the RC system software. The main steps in the IP core activation process are illustrated. During the IP core activation process, SeReCon performs verification of the IP core compliance with the current RC system state in order to protect the integrity of the BaseSFC and to countermeasure the risk of implicit communication channel setup. IP core license validation and RC system reconfiguration are also described. License validation prior to RC system PR enforces both transaction based and metered usage IP business models. The IP core deactivation process removes the remains of previously activated IP cores which could interact with the current system configuration, thus leading to RC system integrity issues. IP core deactivation ensures that the unused IP core configuration is removed from the FPGA configuration memory.
Chapter 7. Case Study: SeReCon Architecture
Implementation And Application in SDR device

7.1. Introduction

This chapter reports on the implementation and application of the prototype SeReCon-enabled RC system using Xilinx Virtex-5 FPGA technology. The implementation of SeReCon internal elements, the main RC system elements and example PR IP cores is described. Analysis of the SeReCon FPGA resource usage and RC system prototype implementation issues is included. The SeReCon IP core is a CPU-based system which is implemented using the Xilinx EDK software. SeReCon uses an embedded 32-bit MicroBlaze processor which is operating at 125 MHz.

Figure 7-a illustrates the block diagram of the SeReCon demonstrator. The demonstrator application is implemented in Python and executed within the Python interpreter which is a standard part of Ubuntu Linux distribution, installed on the Intel host server. The demonstrator application uses the RC system communication library and SeReCon API. The Intel server includes a Xilinx ML505 FPGA evaluation board which contains the prototype of SeReCon-enabled RC system (Figure 7-b). The RC system is connected to the host using the PCIe interface and standard Linux PCIe Device Driver. This chapter reports and describes the SeReCon-enabled RC system prototype, including implementation results, and SeReCon demonstrator application (Figure 7-b). The communication library of the RC system demonstrator and host-side SeReCon API are highlighted. Demonstrator application results are also reported.

The chapter provides detailed insight into the operation of the prototype RC system during the SeReCon (and EIDR) initialisation, IP core installation and activation (see Chapter 6). The implemented RC system uses four IP cores in order to demonstrate the SeReCon-based PR, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher. The VHDL model for each of these IP cores is included in the thesis DVD.

This chapter describes the SDR device prototype and illustrates how the SeReCon element can be included within the SDR RC system. Modifications to the SeReCon implementation, required to integrate SeReCon within the prototype SDR device, are also highlighted.
7.2. RC System Implementation

7.2.1. Introduction

The prototype of the SeReCon-enabled RC system has been implemented in the Xilinx Virtex-5 LXT FPGA \(^53\) using the Xilinx EAPR design flow [10] and Xilinx EDA software \(^54\), e.g. ISE, EDK and PlanAhead tools.

This section illustrates the block diagram of the implemented SeReCon-enabled RC system. The functionality of the RC system elements, e.g. PCIe interface, PCIe BAR Splitter and Configuration Controller are highlighted. SeReCon internals and example PR region IP cores are also described.

7.2.2. SeReCon-Enabled RC System Block Diagram

Figure 7- illustrates the block diagram of the SeReCon-enabled RC system (Figure 1-) which is implemented in the Xilinx Virtex-5 FPGA. For clarity, the block diagram includes only simplified IP core interfaces. The RC system includes SeReCon, PCIe interface and a number of interfacing modules, e.g. PCIe BAR Splitter, Config Controller and Bus Macros. The RC system also includes the PR region which is reconfigured with a number of IP cores \(^55\) (e.g. AES cipher/decipher

---

\(^{53}\) Xilinx ML505 board includes Virtex XC5VLX50TFFG1136 FPGA device.

\(^{54}\) Xilinx ISE v9.2.04i_PR11, Xilinx EDK v9.2.02i and Xilinx PlanAhead v10.1.8.

\(^{55}\) The RC system prototype is implemented using the Xilinx EAPR design flow and tools. EAPR tools require that all PR IP cores must be included during the RC system development.
and simple math adder/multiplier IP cores which are described in Section 7.2.6) in order to demonstrate SeReCon functionality.

External devices, e.g. LEDs, the PCIe edge-connector, the DDR2 SDRAM memory, the SysAce chip and the RS232 voltage converter, are included on the ML505 board. Xilinx IP cores are available from the Xilinx EDA software\(^56\), e.g. Xilinx ISE (PCIe reference design), Xilinx EDK (SeReCon elements) or the Xilinx EAPR lounge website (PR Bus Macros).

Figure 7: Block diagram of the SeReCon-enabled RC system which is implemented in the Xilinx Virtex-5 FPGA (Xilinx Virtex-5 LXT FPGA ML505 Evaluation Platform).

\(^{56}\) Xilinx IP core design files, e.g. HDL sources and netlists, are proprietary and are not included in this thesis. Also, valid license or access to the EAPR Lounge is typically required.
Figure 7- illustrates the Xilinx PlanAhead view of the SeReCon-enabled RC system top-level netlists.

Figure 7- illustrates the Xilinx ISE Schematic top-level view for the SeReCon-enable the RC system which includes all interfaces between the RC system elements.
7.2.3. PCIe Interface IP Core

The demonstrator application (Figure 7-) communicates with the PR region and the SeReCon system using the PCIe endpoint (Figure 7-). The PCIe interface is implemented using the modified PCIe reference design which is generated using the Xilinx CORE Generator\(^{57}\).

The PCIe reference design used in the implementation example includes a small block of (BRAM) memory. This BRAM element is substituted with the IP core external interface which is connected to the RC system (PCIe BAR Splitter). This enables support for (and demonstration of) data transfer (reading and writing) over the PCIe link. Communication to/from the SeReCon-enabled RC system is performed through writing/reading registers which are mapped into the PCIe Block Address Ranges (BARs\(^{58}\)). Figure 7- illustrates the VHDL description of the

\[\text{Figure 7- Xilinx ISE Schematic view of the SeReCon-enabled RC system.}\]

\(^{57}\) The CORE Generator tool is part of the Xilinx ISE software.

\(^{58}\) PCIe BARs are host memory regions reserved for a PCIe device (here RC systems).
Chapter 7 - Case Study: SeReCon Architecture Implementation

modified PCIe reference design which includes the additional memory interface which is used for communication with the RC system. The PCIe endpoint-related signals are removed for clarity. The PCIe reference design signal data dictionary is extended with following signals:

- **mem_rd_data[31:0]**: 32-bit PCIe input port which is read from memory upon receipt of a read data request from the PCIe root complex (PCIe master device).
- **mem_rd_addr[10:0]**: bits [10:9] select the active BAR while bits [8:0] provide the 9-bit address of the memory location which is to be read.
- **mem_rd_be[3:0]**: byte enable flag. Asserted bits indicate the valid bytes in the mem_rd_data word.
- **mem_wr_data[31:0]**: 32-bit PCIe output port which is written to memory upon receipt of the write data request from the PCIe root complex (PCIe master device).
- **mem_wr_addr[10:0]**: bits [10:9] select the active BAR while bits [8:0] provide the 9-bit address of the memory location which is to be written.
- **mem_wr_be[3:0]**: byte enable flag. Asserted bits indicate the valid bytes in the mem_wr_data word.
- **mem_wr_en**: enables write to the memory cell (when asserted).
- **mem_wr_busy**: assertion indicates that the target memory is busy. The PCIe root complex waits until the signal is deasserted by the memory.

Table 7- describes the PCIe interface BARs which are used by the SeReCon-enabled RC system. The PCIe reference design supports four BARs, but only two are used. Requests addressing BAR1 reach the RC system Configuration Controller IP core (Config Ctrl) which connects SeReCon with the RC system, while BAR2 supports communication with the PR region.

![VHDL description of the modified PCIe reference design](image-url)

Figure 7- The VHDL description of the modified PCIe reference design which includes the additional memory interface which is used for communication with the RC system. PCIe Endpoint-related signals are removed for clarity.

59 Experiments with the PCIe reference design shown (reboot-requiring) stalls of the server when the `mem_wr_busy` is left asserted.
Table 7 - Description of the PCIe interface BARs.

<table>
<thead>
<tr>
<th>Block Address Range (BAR)</th>
<th>Used addresses</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAR0</td>
<td>N/A</td>
<td>Used internally by PCIe reference design.</td>
</tr>
<tr>
<td>BAR1</td>
<td>TBD</td>
<td>Provides R/W access to the SeReCon configuration controller.</td>
</tr>
<tr>
<td>BAR2</td>
<td>TBD</td>
<td>Provides R/W access to the PR region in the RC system.</td>
</tr>
<tr>
<td>BAR3</td>
<td>-</td>
<td>Not used.</td>
</tr>
</tbody>
</table>

7.2.4. PCIe BAR Splitter IP Core

The PCIe BAR Splitter IP core separates PCIe requests depending on the destination BAR (2 MSBs in the `mem_rd_addr` or `mem_wr_addr`). Figure 7 illustrates the VHDL description of the PCIe BAR Splitter IP core interface. The PCIe BAR Splitter connects the PCIe interface with the `Config Ctrl` IP core and PR region (Figure 7- and Figure 7-). The description of `Config Ctrl` interface signals (`bar_1_*`) and PR region interface signals (`bar_2_*`) is the same as in the PCIe reference design.

```vhdl
COMPONENT bar_splitter
PORT(
  -- PCIe interface
  rd_addr_i : IN std_logic_vector(10 downto 0);
  rd_be_i : IN std_logic_vector(3 downto 0);
  wr_addr_i : IN std_logic_vector(10 downto 0);
  wr_be_i : IN std_logic_vector(7 downto 0);
  wr_data_i : IN std_logic_vector(31 downto 0);
  wr_en_i : IN std_logic;
  rd_data_o : OUT std_logic_vector(31 downto 0);
  wr_busy_o : OUT std_logic;

  -- Ctrl interface
  bar_1_rd_data_i : IN std_logic_vector(31 downto 0);
  bar_1_wr_busy_i : IN std_logic;
  bar_1_rd_addr_o : OUT std_logic_vector(8 downto 0);
  bar_1_rd_be_o : OUT std_logic_vector(3 downto 0);
  bar_1_wr_addr_o : OUT std_logic_vector(8 downto 0);
  bar_1_wr_be_o : OUT std_logic_vector(7 downto 0);
  bar_1_wr_data_o : OUT std_logic_vector(31 downto 0);
  bar_1_wr_en_o : OUT std_logic;

  -- PR region interface
  bar_2_rd_data_i : IN std_logic_vector(31 downto 0);
  bar_2_wr_busy_i : IN std_logic;
  bar_2_rd_addr_o : OUT std_logic_vector(8 downto 0);
  bar_2_rd_be_o : OUT std_logic_vector(3 downto 0);
  bar_2_wr_addr_o : OUT std_logic_vector(8 downto 0);
  bar_2_wr_be_o : OUT std_logic_vector(8 downto 0);
  bar_2_wr_data_o : OUT std_logic_vector(31 downto 0);
  bar_2_wr_en_o : OUT std_logic;
);
END COMPONENT;
```

Figure 7 - The VHDL description of the PCIe BAR Splitter IP core interface.

---

60 The VHDL source code of the PCIe BAR Splitter IP core (`bar_splitter.vhd`) is included in the attached DVD.
Chapter 7 - Case Study: SeReCon Architecture Implementation

7.2.5. Configuration Controller IP core

The Configuration Controller (Config Ctrl) IP core provides the SeReCon wrapper interface which is compatible with the PCIe interface. This IP core controls SeReCon, the FPGA PR region and the FPGA development system LEDs which are located on the ML505 board. The IP core contains two asynchronous FIFO buffers\(^{61}\) (DIN, DOUT) which are used for data transfers between SeReCon and the demonstrator application.

Figure 7- illustrates the VHDL description of the SeReCon-enabled RC system Config Ctrl IP core interface.

The SeReCon interface provides a communication link between the Config Ctrl IP core and SeReCon IP core (system_i.xmp). Communication with SeReCon occurs through a number of Config Ctrl IP core registers which are mapped into the PCIe BAR memory space (all signals are active high unless stated otherwise), as follows:

- \textit{mb_bm_en}: SeReCon asserts this signal in order to enable the bus macros between the PR region and the PCIe interface (Figure 7-, Figure 7-)
- \textit{mb_dcm_locked}: assertion indicates that the FPGA Digital Clock Managers (DCMs) used by SeReCon have been successfully initialised (locked). This signal is used during FPGA power up
- \textit{mb_rst_n}: active low SeReCon reset signal which supports SeReCon reset using PCIe
- \textit{mb_clk}: 125MHz clock signal generated by SeReCon. This signal is required by the DDR2 IP core
- \textit{mb_dout_pin}[31:0]: FIFO-buffered SeReCon data register (DOUT) output port
- \textit{mb_dout_wr_pin}: asserted by SeReCon to write data from the SeReCon data output register into the Configuration Controller FIFO
- \textit{mb_din_pin}[31:0]: FIFO-buffered SeReCon data register input port
- \textit{mb_din_rd_pin}: asserted by SeReCon to read data from the Configuration Controller FIFO into the SeReCon data input register
- \textit{mb_stat}[31:0]: unbuffered SeReCon status register
- \textit{mb_ctrl}[31:0]: unbuffered SeReCon control register

The Ctrl interface provides a communication link between the Config Ctrl IP core and the PCIe interface. The Ctrl interface signals are connected to the PCIe BAR Splitter IP core (Figure 7-).

LED interface signals are connected to 8 LEDs mounted on the ML505 board. LEDs are controlled through the register which is mapped into the PCIe BAR memory space.

\(^{61}\) The VHDL source code of the Configuration Controller IP core (config_manager.vhd) is included in the attached DVD. The code does not include DIN and DOUT FIFO netlists which are generated using the Xilinx ISE CORE generator.
PR region interface signals control the FPGA PR region, as follows:

- $\text{prm\_bm\_en}$: enables and disables PR region Bus Macros
- $\text{prm\_id}$: 4-bit ID of the IP core loaded into the PR region

The $\text{rst\_btn}$ signal is connected to the push button on the ML505 board. This facilitates manual SeReCon reset.

Table 7- describes the contents of the Config Ctrl IP core registers which are mapped into the PCIe (BAR1) address space.

```vhdl
COMPONENT config_manager
PORT(
  -- Cntrl interface
  pcie_clk          : in std_logic;
  pcie_rst_n        : in std_logic;
  pcie_rd_addr      : in std_logic_vector(8 downto 0);
  pcie_rd_be        : in std_logic;
  pcie_rd_data      : out std_logic_vector(31 downto 0);
  pcie_wr_addr      : in std_logic_vector(8 downto 0);
  pcie_wr_be        : in std_logic;
  pcie_wr_data      : in std_logic_vector(31 downto 0);
  pcie_wr_en        : in std_logic;
  pcie_wr_busy      : out std_logic;
  -- SeReCon interface
  mb_bm_en          : in std_logic;
  mb_dcm_locked     : in std_logic;
  mb_rst_n          : out std_logic;
  mb_clk            : in std_logic;
  mb_dout_wr_pin    : in std_logic;
  mb_dout_pin       : in std_logic_vector(31 downto 0);
  mb_din_pin        : out std_logic_vector(31 downto 0);
  mb_din_rd_pin     : in std_logic;
  mb_ctrl           : out std_logic_vector(31 downto 0);
  mb_stat           : in std_logic_vector(31 downto 0);
  -- LED interface
  user_led          : out std_logic_vector(7 downto 0);
  -- PR region interface
  prm_bm_en         : OUT std_logic;
  prm_id            : in std_logic_vector(3 downto 0);
  -- Other interfaces
  rst_btn           : in std_logic
);
END COMPONENT;
```

Figure 7- The VHDL description of the SeReCon-enabled RC system Configuration Controller IP core interface.
### Chapter 7 - Case Study: SeReCon Architecture Implementation

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LED</td>
<td>R/W</td>
<td>Bits [8:0] control LEDs on the ML505 board.</td>
</tr>
<tr>
<td>1</td>
<td>STATUS</td>
<td>R</td>
<td>Bits [31:0] indicate current SeReCon status.</td>
</tr>
<tr>
<td>2</td>
<td>CONTROL</td>
<td>R/W</td>
<td>This register controls SeReCon, internal FIFOs and PR region BMs. Bits description: 0 – resets SeReCon IP core, 1 – resets internal DIN FIFO and write counter, 2 – resets internal DOUT FIFO and read counter, 3 – indicates DIN FIFO is full, 4 – indicates DIN FIFO is empty, 5 – indicates DOUT FIFO is full, 6 – indicates DOUT FIFO is empty, 7 – indicates SeReCon BM enable status, 11:8 – Provide the 4-bit ID of the IP core in the PR region, 12 – enables RC system BM’s.</td>
</tr>
<tr>
<td>3</td>
<td>DIN</td>
<td>R</td>
<td>Contains 32-bit word which is send to SeReCon (written to DIN FIFO).</td>
</tr>
<tr>
<td>4</td>
<td>DOUT</td>
<td>R</td>
<td>Contains 32-bit word which is received from SeReCon (read from DOUT FIFO).</td>
</tr>
<tr>
<td>5</td>
<td>RD_CNT</td>
<td>R/W</td>
<td>Dummy write to this register reads SeReCon data word (from DOUT FIFO) to DOUT register. Register read returns number of 32-bit words read from DOUT FIFO.</td>
</tr>
<tr>
<td>6</td>
<td>WR_CNT</td>
<td>R/W</td>
<td>Dummy write to this register sends DIN register content to SeReCon (writes data word to DIN FIFO). Register read returns number of 32-bit words written to DIN FIFO.</td>
</tr>
</tbody>
</table>

Table 7- Description of Configuration Controller registers which are mapped into the PCIe BAR1.

---

**Page 140**
7.2.6. PR Region IP Cores

7.2.6.1. Introduction

The PR region is the part of the RC system which is reconfigured in run time using the IP cores (Figure 1-, Figure 1-, Figure 1-). The implemented RC system uses four IP cores in order to demonstrate the SeReCon-based PR, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher. The VHDL model for each of these IP cores is included in the thesis DVD. Each core is described briefly in this section.

Figure 7- illustrates the VHDL description of the PR region interface which is available through the PCIe interface (BAR2). Most signals are already described in the PCIe interface section. Additional signals are:

- **Clk**: the input clock signal (100MHz) provided by the PCIe interface
- **rst_n**: active low reset signal provided by the PCIe interface
- **id[3:0]**: 4-bit IP core ID. The IP core ID is used by the demonstrator in order to identify the activated IP core

```
COMPONENT prm_wrapper
PORT(
  clk : IN std_logic;
  rst_n : IN std_logic;
  rd_addr_i : IN std_logic_vector(8 downto 0);
  rd_be_i : IN std_logic_vector(3 downto 0);
  rd_data_o : OUT std_logic_vector(31 downto 0);
  wr_addr_i : IN std_logic_vector(8 downto 0);
  wr_be_i : IN std_logic_vector(7 downto 0);
  wr_data_i : IN std_logic_vector(31 downto 0);
  wr_en_i : IN std_logic;
  wr_busy_o : OUT std_logic;
  id : OUT std_logic_vector(3 downto 0)
);
END COMPONENT;
```

Figure 7- The VHDL description of the PR region interface.

Figure 7- illustrates the FPGA fabric with a size-constrained PR region and location-constrained BMs (a view from the Xilinx PlanAhead tool). The implemented RC system includes cryptographic and math IP cores, e.g. AES cipher/decipher, simple adder and multiplier.

The Xilinx EAPR design flow does not support multiple IP cores within a single PR region62. EAPR also requires BMs between the PR IP cores and the static part of the RC system, e.g. BaseSFC. Thus, all IP cores must be implemented using the

62 Thus, this thesis does not address the issue of implicit communication channels between multiple IP cores in the PR region.
default ‘wrapper’ IP core which provides the interface compatible with the SeReCon-enabled RC system.

Figure 7- The FPGA fabric with a size-constrained PR region and location-constrained Bus Macros (a view from the Xilinx PlanAhead tool).
Chapter 7 - Case Study: SeReCon Architecture Implementation

7.2.6.2. Adder PR IP Core (adder.vhd)

This IP core adds content of two 32-bit registers (A and B) and stores the result in the third register (C). Table 7- describes IP core registers which are available through the PR region memory-mapped interface. The Adder IP core uses four memory-mapped registers in the PCIe BAR2.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>REG_A</td>
<td>R/W</td>
<td>Contains 32-bit argument A.</td>
</tr>
<tr>
<td>1</td>
<td>REG_B</td>
<td>R/W</td>
<td>Contains 32-bit argument B.</td>
</tr>
<tr>
<td>2</td>
<td>REG_C</td>
<td>R</td>
<td>Contains 32-bit sum of A and B.</td>
</tr>
<tr>
<td>3</td>
<td>REG_CTRL</td>
<td>R</td>
<td>Bits [3:0] contain IP core ID = “0001”.</td>
</tr>
</tbody>
</table>

Table 7- Description of the Adder interface registers.

7.2.6.3. Multiplier PR IP Core (multiplier.vhd)

The multiplier IP core multiplies the contents of two 32-bit registers (A and B) and stores the lower-half of the 64-bit result in register (C). Table 7- describes the IP core registers which are available through the PR region memory-mapped interface. The Multiplier IP core uses four memory-mapped registers in the PCIe BAR2.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>REG_A</td>
<td>R/W</td>
<td>Contains 32-bit multiplier A.</td>
</tr>
<tr>
<td>1</td>
<td>REG_B</td>
<td>R/W</td>
<td>Contains 32-bit multiplicand B.</td>
</tr>
<tr>
<td>2</td>
<td>REG_C</td>
<td>R</td>
<td>Contains lower 32-bits of product of A and B.</td>
</tr>
<tr>
<td>3</td>
<td>REG_CTRL</td>
<td>R</td>
<td>Bits [3:0] contain IP core ID = “1101”.</td>
</tr>
</tbody>
</table>

Table 7- Description of the Multiplier register interface.
7.2.6.4. **AES Cipher PR IP Core (aes_enc_wrapper.vhd)**

The AES Cipher IP core uses the open-source design of the 128-bit AES block cipher [36] which is available through OpenCores. The IP core encrypts the contents of a 128-bit TEXT_IN register using the key which is stored in the 128-bit KEY register. The ciphertext is stored in the 128-bit TEXT_OUT register.

Table 7 describes IP core registers which are available through the PR region memory-mapped interface. The AES Cipher IP core uses 13 memory-mapped registers in the PCIe BAR2.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>KEY_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES KEY register.</td>
</tr>
<tr>
<td>1</td>
<td>KEY_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES KEY register.</td>
</tr>
<tr>
<td>2</td>
<td>KEY_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES KEY register.</td>
</tr>
<tr>
<td>3</td>
<td>KEY_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES KEY register.</td>
</tr>
<tr>
<td>4</td>
<td>TXT_IN_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>5</td>
<td>TXT_IN_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>6</td>
<td>TXT_IN_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>7</td>
<td>TXT_IN_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>9</td>
<td>TXT_OUT_2</td>
<td>R</td>
<td>Contains [95:64] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>10</td>
<td>TXT_OUT_1</td>
<td>R</td>
<td>Contains [63:32] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>11</td>
<td>TXT_OUT_0</td>
<td>R</td>
<td>Contains [31:0] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>12-15</td>
<td>-</td>
<td>-</td>
<td>Unused.</td>
</tr>
<tr>
<td>16</td>
<td>STAT/CTL</td>
<td>R/W</td>
<td>Bit 0 provides the IP core status (when asserted the IP core is busy). Write to this register initiates encryption process.</td>
</tr>
</tbody>
</table>

Table 7- Description of the AES Cipher register interface.

---

63 OpenCores (http://www.opencores.org)
The AES decipher IP core uses the open-source design of the AES block decipher [36] which is available through the OpenCores. The IP core decrypts the contents of a 128-bit TEXT_IN register using the key which is stored in the 128-bit KEY register. The plaintext is stored in the 128-bit TEXT_OUT register.

Table 7 describes the AES Decipher IP core registers which are available through the PR region memory-mapped interface. The AES Decipher IP core uses 13 memory-mapped registers in the PCIe BAR2.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>KEY_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES KEY register.</td>
</tr>
<tr>
<td>1</td>
<td>KEY_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES KEY register.</td>
</tr>
<tr>
<td>2</td>
<td>KEY_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES KEY register.</td>
</tr>
<tr>
<td>3</td>
<td>KEY_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES KEY register.</td>
</tr>
<tr>
<td>4</td>
<td>TXT_IN_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>5</td>
<td>TXT_IN_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>6</td>
<td>TXT_IN_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>7</td>
<td>TXT_IN_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES TEXT_IN register.</td>
</tr>
<tr>
<td>9</td>
<td>TXT_OUT_2</td>
<td>R</td>
<td>Contains [95:64] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>10</td>
<td>TXT_OUT_1</td>
<td>R</td>
<td>Contains [63:32] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>11</td>
<td>TXT_OUT_0</td>
<td>R</td>
<td>Contains [31:0] bits of the AES TEXT_OUT register.</td>
</tr>
<tr>
<td>12-15</td>
<td>-</td>
<td>-</td>
<td>Unused.</td>
</tr>
<tr>
<td>16</td>
<td>STAT/CTL</td>
<td>R/W</td>
<td>Bits [1:0] provides the IP core status (assertion of either bit indicates that the IP core is busy). A write to this register initiates the decryption process.</td>
</tr>
<tr>
<td>17</td>
<td>KEY_LD</td>
<td>W</td>
<td>A write to this register initiates the key calculation process.</td>
</tr>
</tbody>
</table>

Table 7- Description of the AES Decipher register interface.
7.2.7. SeReCon IP Core

The SeReCon IP core is a CPU-based system which is implemented using the Xilinx EDK software. SeReCon uses an embedded 32-bit MicroBlaze processor which is operating at 125 MHz.

Figure 7- illustrates the Xilinx EDK view of the SeReCon internal organisation. SeReCon includes the MicroBlaze CPU (MICROBLAZE_0) along with a number of IP cores which are connected using the Processor Local Bus (PLB) interface and the Local Memory Block (LMB) interface. Only the AES (AES_0), TRNG (VT_TRNG_PUF_0) and the PCIe interface IP cores (PCIE_INTERFACE_0) have been implemented and described in this thesis. The remaining IP cores are generated using the Xilinx EDK software.

<table>
<thead>
<tr>
<th>Bus Interfaces</th>
<th>Parts</th>
<th>Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Bus Connection</td>
<td>IP Type</td>
</tr>
<tr>
<td>microblaze_0</td>
<td>microblaze</td>
<td>7.00.b</td>
</tr>
<tr>
<td>mm</td>
<td>imb_v10</td>
<td>1.00.a</td>
</tr>
<tr>
<td>mm</td>
<td>imb_v10</td>
<td>1.00.a</td>
</tr>
<tr>
<td>mm</td>
<td>plbl_v46</td>
<td>1.00.a</td>
</tr>
<tr>
<td>mm</td>
<td>imb_bram_if_cntlr2.10.a</td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>imb_bram_if_cntlr2.10.a</td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>bram_block</td>
<td>1.00.a</td>
</tr>
<tr>
<td>SRAM</td>
<td>xps_mch_emc</td>
<td>1.00.a</td>
</tr>
<tr>
<td>aes_0</td>
<td>aes</td>
<td>1.00.a</td>
</tr>
<tr>
<td>xps_gpio_0</td>
<td>xps_gpio</td>
<td>1.00.a</td>
</tr>
<tr>
<td>xps_init_0</td>
<td>xps_initc</td>
<td>1.00.a</td>
</tr>
<tr>
<td>DDR2_SDRAM</td>
<td>mpmc</td>
<td>3.00.b</td>
</tr>
<tr>
<td>xps_timer_1</td>
<td>xps_timer</td>
<td>1.00.a</td>
</tr>
<tr>
<td>HS2_S2_Uart_1</td>
<td>xps_uartlite</td>
<td>1.00.a</td>
</tr>
<tr>
<td>xps_hwncap_0</td>
<td>xps_hwncap</td>
<td>1.00.a</td>
</tr>
<tr>
<td>debug_module</td>
<td>mdm</td>
<td>1.00.a</td>
</tr>
<tr>
<td>vt_rng_put_0</td>
<td>vt_rng_put</td>
<td>1.00.a</td>
</tr>
<tr>
<td>pcie_interface_0</td>
<td>pcie_interface</td>
<td>1.00.a</td>
</tr>
<tr>
<td>proc_sys_reset_0</td>
<td>proc_sys_reset</td>
<td>2.00.a</td>
</tr>
<tr>
<td>xps_system_rdesk_1</td>
<td>xps_system_rdesk</td>
<td>1.00.a</td>
</tr>
<tr>
<td>clock_generator_0</td>
<td>clock_generator</td>
<td>1.00.a</td>
</tr>
<tr>
<td>SysACE_CompactFlash</td>
<td>SysACE_CompactFlash</td>
<td>1.00.a</td>
</tr>
<tr>
<td>SRAM_util_bus_split_0</td>
<td>util_bus_split</td>
<td>1.00.a</td>
</tr>
</tbody>
</table>

Figure 7- The Xilinx EDK view of the SeReCon internal organisation.

Figure 7- illustrates the VHDL description of the SeReCon IP core interface. SeReCon includes the RC system interface, SysAce interface, UART interface and DDR2_SDRAM interface as follows:

**The RC system interface** connects SeReCon to the Configuration Controller IP core and to the RC system BMs. Table 7- describes the SeReCon RC system interface signals.
Chapter 7 - Case Study: SeReCon Architecture Implementation

**SysAce interface** signals drive the ML505 SysAce CompactFlash controller which is used as the SeReCon LIPS. Xilinx EDK documentation provides the SysAce signals description.

The **UART interface** provides a serial console interface\(^{64}\) used for SeReCon debugging. Figure 7- illustrates the view of the SeReCon serial console which uses the UART interface in order to support RC system debugging.

The **DDR2_SDRAM interface** drives the external 256MB DDR2 memory card mounted in the ML505 SODIMM slot. The RC system prototype uses this memory in order to store the SeReCon firmware\(^{65}\).

```vhdl
COMPONENT system
PORT(
  -- RC system interface
  pcie_interface_0_mb_din_pin : IN std_logic_vector(31 downto 0);
  pcie_interface_0_mb_ctrl_pin : IN std_logic_vector(31 downto 0);
  pcie_interface_0_mb_dout_wr_pin : OUT std_logic;
  pcie_interface_0_mb_din_rd_pin : OUT std_logic;
  pcie_interface_0_mb_dout_pin : OUT std_logic_vector(31 downto 0);
  pcie_interface_0_mb_stat_pin : OUT std_logic_vector(31 downto 0);
  BM_enable_pin : OUT std_logic_vector(0 to 0);
  sys_clk : OUT std_logic;
  sys_clk_pin : IN std_logic;
  sys_rst_pin : IN std_logic;
  Dcm_all_locked_pin : OUT std_logic;
  -- SysAce interface
  SysACE_ComactFlash_SysACE_CLK_pin : IN std_logic;
  SysACE_ComactFlash_SysACE_MPIRQ_pin : IN std_logic;
  SysACE_ComactFlash_SysACE_MPD_pin : INOUT std_logic_vector(15 downto 0);
  SysACE_ComactFlash_SysACE_MPA_pin : OUT std_logic_vector(6 downto 0);
  SysACE_ComactFlash_SysACE_CEN_pin : OUT std_logic;
  SysACE_ComactFlash_SysACE_OEN_pin : OUT std_logic;
  SysACE_ComactFlash_SysACE_WEN_pin : OUT std_logic;
  -- UART interface
  RS232_Uart_1_TX_pin : OUT std_logic;
  RS232_Uart_1_RX_pin : IN std_logic;
  -- DDR2_SDRAM interface
  DDR2_SDRAM_DDR2_DQS : INOUT std_logic_vector(7 downto 0);
  DDR2_SDRAM_DDR2_DQS_n : INOUT std_logic_vector(7 downto 0);
  DDR2_SDRAM_DDR2_DQ : INOUT std_logic_vector(63 downto 0);
  DDR2_SDRAM_DDR2_DQH : INOUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_DQH : INOUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_BankAddr_pin : OUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_BAS_n_pin : OUT std_logic;
  DDR2_SDRAM_DDR2_CE_n_pin : OUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_CS_n_pin : OUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_RAS_n_pin : OUT std_logic;
  DDR2_SDRAM_DDR2_WE_n_pin : OUT std_logic;
  DDR2_SDRAM_DDR2_CLK_pin : OUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_CLK_n_pin : OUT std_logic_vector(1 downto 0);
  DDR2_SDRAM_DDR2_DM_pin : OUT std_logic_vector(7 downto 0);
); END COMPONENT;
```

Figure 7- VHDL description of the SeReCon IP core interface.

\(^{64}\) 115200 kbps, 8-bit, no parity.

\(^{65}\) The DDR2 SDRAM memory is used to simplify the prototyping process. The thesis assumes that the SeReCon firmware is located in the FPGA internal (BRAM) memory resources.
Figure 7- View of the SeReCon serial console which uses the UART interface in order to support RC system debugging.
### Table 7 - Description of the SeReCon RC system interface signals which connect SeReCon to the Configuration Controller IP core and RC system BMs.

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pcie_interface_0_mb_din_pin[31:0]</code></td>
<td>SeReCon data input port which is connected to DIN FIFO inside the Configuration Controller IP core.</td>
</tr>
<tr>
<td><code>pcie_interface_0_mb_dout_pin[31:0]</code></td>
<td>SeReCon data output port which is connected to DOUT FIFO inside the Configuration Controller IP core.</td>
</tr>
<tr>
<td><code>pcie_interface_0_mb_ctrlr_pin[31:0]</code></td>
<td>SeReCon control port which is connected to the CTRL register inside the Configuration Controller IP core.</td>
</tr>
<tr>
<td><code>pcie_interface_0_mb_stat_pin[31:0]</code></td>
<td>SeReCon status port which is connected to STAT register inside the Configuration Controller IP core.</td>
</tr>
<tr>
<td><code>pcie_interface_0_mb_din_rd_pin</code></td>
<td>DIN FIFO read signal. SeReCon asserts this signal prior to reading the data word from <code>pcie_interface_0_mb_din_pin</code>.</td>
</tr>
<tr>
<td><code>pcie_interface_0_mb_dout_wr_pin</code></td>
<td>DOUT FIFO write signal. SeReCon asserts this signal after writing the data word to <code>pcie_interface_0_mb_dout_pin</code>.</td>
</tr>
<tr>
<td><code>sys_clk_pin</code></td>
<td>The main RC system clock signal (100MHz). SeReCon uses this signal as a reference input clock signal.</td>
</tr>
<tr>
<td><code>sys_clk</code></td>
<td>SeReCon internal clock signal (125MHz) which is used to drive SeReCon interface of DI and DOUT FIFOs inside the Configuration Controller IP core.</td>
</tr>
<tr>
<td><code>sys_rst_pin</code></td>
<td>Assertion of this signal disables SeReCon. The RC system toggles this signal in order to restart SeReCon firmware (the EIDR content is not affected)</td>
</tr>
<tr>
<td><code>dcm_all_locked_pin</code></td>
<td>Assertion of this signal indicates that SeReCon internal Digital Clock Managers (DCMs) are operating correctly.</td>
</tr>
<tr>
<td><code>bm_enable_pin</code></td>
<td>Assertion of this signal indicates that SeReCon enabled RC system bus macros, e.g. after successful IP core activation. In the RC system prototype this signal is used only for information, e.g. RC system bus macros are enabled using separate bit in the Configuration Ctrl IP core.</td>
</tr>
</tbody>
</table>
7.2.7.1. **SeReCon PCIe Interface IP Core**

Figure 7- illustrates the block diagram of the SeReCon PCIe interface IP core. The SeReCon IP core is connected to the RC system through the modified Interface (IPIF) core wrapper which is generated using Xilinx EDK. The modification includes the extension of the IP core IO interface in order to connect the IPIF registers to the Configuration Controller IP core. This supports SeReCon access to the RC system through the IPIF registers which are mapped into the Microblaze memory.

Figure 7: The block diagram of the SeReCon PCIe interface IP core.

---

VHDL model source code for the SeReCon PCIe interface IP core is included in the thesis DVD.
7.2.7.2. *SeReCon TRNG IP Core*

The TRNG IP core implements a compact (TRNG + PUF) Ring-Oscillator (RO) based design which was first proposed by Maiti et al. [35]. The design provides scalability and portability. The SeReCon TRNG IP core uses 8 RO ‘macrocels’\(^{67}\). The macrocell contains single RO which is implemented using FPGA device specific primitives. Macrocells are implemented in the Virtex-5 FPGA as hard macros (similar to BMs). This prevents the Xilinx EDA software from automatic RO removal during the TRNG netlist implementation and supports macrocell location constraining in the PlanAhead during PR design flow.

The TRNG IP core is connected to the Microblaze using the Xilinx IPIF core wrapper. Table 7- describes TRNG IP core registers which are mapped in the MicroBlaze memory and are available to the SeReCon firmware. Current version of the SeReCon firmware does not use the PUF functionality.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TRNG_DATA</td>
<td>R</td>
<td>Contains a 32-bit random word generated by the IP core in the TRNG mode.</td>
</tr>
<tr>
<td>1</td>
<td>PUF_CNT</td>
<td>R</td>
<td>Not used. Contains 32-bit PUF counter value. The registered value is a number of RO oscillations during a period of time which is defined by the TIMER register value.</td>
</tr>
<tr>
<td>2</td>
<td>RO_ENABLE</td>
<td>R/W</td>
<td>Bits[7:0] enable TRNG ROs. In the TRNG mode all RO’s are enabled. In the PUF mode a single RO is enabled only. RO’s are activated only during the SeReCon requests.</td>
</tr>
<tr>
<td>3</td>
<td>TIMER</td>
<td>R/W</td>
<td>Not used. Contains 32-bit PUF timer value. This is ten number of system clock cycles for which the IP core counts RO oscillations. The counted value is read from the PUF_CNT register. Write to this register starts counting.</td>
</tr>
<tr>
<td>4</td>
<td>CTRL</td>
<td>R/W</td>
<td>Bit 0 – IP core mode (‘1’– TRNG, ‘0’–PUF), bit 1 – is a TRNG busy flag, bit 2 – is a PUF busy flag, bit 3 – is asserted when the content of the TRNG_DATA is updated with new data, bits [31:4] are unused.</td>
</tr>
</tbody>
</table>

\(^{67}\) The increase in the number of included Ring-Oscillators (ROs) up to 32 requires only simple modification of the TRNG VDHL code and does not change TRNG API. If more than 32 RO’s are required, e.g. for generating high-quality random data, an additional RO_ENABLE register must be added to the core.
7.2.7.3. SeReCon AES IP Core

The SeReCon AES IP core uses the CBC operation mode in order to support the encryption of arbitrary-length data, e.g. IP core bitstreams. In the CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. The first block of the plaintext is XORed with the initialisation vector (IV). The SeReCon AES IP core uses the same AES block cipher (and decipher) as the RC system PR IP cores (the AES cipher/decipher PR IP core). The SeReCon AES IP core is connected to the Microblaze using the Xilinx IPIF core wrapper. Table 7- and Table 7- describe the SeReCon AES IP core registers which are mapped into the MicroBlaze memory and are available to the SeReCon firmware.

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ENC_KEY_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES cipher KEY register.</td>
</tr>
<tr>
<td>1</td>
<td>ENC_KEY_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES cipher KEY register.</td>
</tr>
<tr>
<td>2</td>
<td>ENC_KEY_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES cipher KEY register.</td>
</tr>
<tr>
<td>3</td>
<td>ENC_KEY_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES cipher KEY register.</td>
</tr>
<tr>
<td>4</td>
<td>ENC_TXT_IN_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES cipher TEXT_IN register.</td>
</tr>
<tr>
<td>5</td>
<td>ENC_TXT_IN_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES cipher TEXT_IN register.</td>
</tr>
<tr>
<td>6</td>
<td>ENC_TXT_IN_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES cipher TEXT_IN register.</td>
</tr>
<tr>
<td>7</td>
<td>ENC_TXT_IN_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES cipher TEXT_IN register.</td>
</tr>
<tr>
<td>9</td>
<td>ENC_TXT_OUT_2</td>
<td>R</td>
<td>Contains [95:64] bits of the AES cipher TEXT_OUT register.</td>
</tr>
<tr>
<td>10</td>
<td>ENC_TXT_OUT_1</td>
<td>R</td>
<td>Contains [63:32] bits of the AES cipher TEXT_OUT register.</td>
</tr>
<tr>
<td>11</td>
<td>ENC_TXT_OUT_0</td>
<td>R</td>
<td>Contains [31:0] bits of the AES cipher TEXT_OUT register.</td>
</tr>
</tbody>
</table>

Table 7- Description of the SeReCon AES IP core registers (continued in Table 7-) which are mapped into the MicroBlaze memory and are available to the SeReCon firmware.

---

# Chapter 7 - Case Study: SeReCon Architecture Implementation

<table>
<thead>
<tr>
<th>Register number</th>
<th>NAME</th>
<th>Access mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>DEC_KEY_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES decipher KEY register.</td>
</tr>
<tr>
<td>14</td>
<td>DEC_KEY_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES decipher KEY register.</td>
</tr>
<tr>
<td>15</td>
<td>DEC_KEY_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES decipher KEY register.</td>
</tr>
<tr>
<td>16</td>
<td>DEC_TXT_IN_3</td>
<td>R/W</td>
<td>Contains [127:96] bits of the AES decipher TEXT_IN register.</td>
</tr>
<tr>
<td>17</td>
<td>DEC_TXT_IN_2</td>
<td>R/W</td>
<td>Contains [95:64] bits of the AES decipher TEXT_IN register.</td>
</tr>
<tr>
<td>18</td>
<td>DEC_TXT_IN_1</td>
<td>R/W</td>
<td>Contains [63:32] bits of the AES decipher TEXT_IN register.</td>
</tr>
<tr>
<td>19</td>
<td>DEC_TXT_IN_0</td>
<td>R/W</td>
<td>Contains [31:0] bits of the AES decipher TEXT_IN register.</td>
</tr>
<tr>
<td>21</td>
<td>DEC_TXT_OUT_2</td>
<td>R</td>
<td>Contains [95:64] bits of the AES decipher TEXT_OUT register.</td>
</tr>
<tr>
<td>23</td>
<td>DEC_TXT_OUT_0</td>
<td>R</td>
<td>Contains [31:0] bits of the AES decipher TEXT_OUT register.</td>
</tr>
<tr>
<td>24</td>
<td>STAT/CTL</td>
<td>R/W</td>
<td>Bit 0 – provides the AES cipher busy flag, [127:96] of the AES decipher</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>key register. [95:64] – starts AES encryption, [63:32] – provide the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>AES decipher busy flags, [31:6] – starts AES decryption, [31:6] are</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unused.</td>
</tr>
</tbody>
</table>

Table 7- Description of the SeReCon AES IP core registers (continuation of Table 7-).
7.2.7.4. **SeReCon Firmware**

Figure 1-e illustrates the SeReCon firmware installed in the FPGA local memory (BRAM). The SeReCon-enabled RC system prototype uses the DDR2 SDRAM memory in order to simplify SeReCon debugging. The SeReCon firmware is divided into modules\(^{69}\), as follows:

**The Board Support Package (BSP) module** provides the standard drivers for Xilinx EDK IP cores and software libraries which are required by the MicroBlaze CPU. The BSP module is automatically generated by Xilinx EDK.

**The AES/ECC module** is a SeReCon crypto library which provides 128-bit AES encryption in the CBC mode (using the SeReCon AES IP core) and the generic implementation of the Elliptic Curve Digital Signature Algorithm (ECDSA) and ECC-based Diffie-Hellman SK agreement routines which are using (NIST standard) P192 elliptic curve.

**The ERDB module** implements the ERDB-based IP core analyser and verifier, used prior to IP core installation and activation.

**The ICAP module** uses modified MicroBlaze drivers in order to support FPGA self-reconfiguration using PR bitstreams stored in the LIPS. The ICAP module also provides user (read/write) access to FPGA configuration registers and individual FPGA configuration frames.

**The EIDR module** emulates the EIDR in software. The emulation is mostly transparent to the SeReCon software, e.g. SeReCon has additional debug routines which are used to update the EIDR content, e.g. increment the EIDR counters. The EIDR module also implements the SafeLock functionality (using the EIDR element and the AES module).

**The MFS module** supports the volatile Memory File System (MFS), implemented in the DDR2 memory in order to support SeReCon debugging. The MFS module extends the MFS library which is available from Xilinx EDK.

**The Comm module** provides high-level drivers and a communication library for the SeReCon PCIe interface IP core. The communication library supports file-based data transfer between the demonstrator application and SeReCon LIPS.

**The Config Manager module** includes the SeReCon main program routine which services the RC system requests in an infinite loop. This module also includes a number of debug routines which are available through the SeReCon UART interface.

**The SysAce module** and the **FATFS module** provide file-based access to the SeReCon LIPS which is implemented in the ML505 CF card.

\(^{69}\) The C source code of the SeReCon firmware is included in the attached DVD. The Xilinx proprietary code is removed.
The TRNG module provides drivers for the SeReCon TRNG IP core. This module is mainly used by the AES/ECC module.

The Memory Manager module uses the AES/ECC module, MFS module and FATFS module in order to support transparent\(^70\) data (e.g. IP core file) encryption prior to storage in the SeReCon LIPS. Transparent decryption during file reading is also supported.

## 7.3. RC System and SeReCon Implementation Results

### 7.3.1. Introduction

Figure 7- illustrates the FPGA Editor view of the SeReCon-enabled RC system implementation on the Xilinx Virtex-5 FPGA. The figure illustrates the location of the PR region, FPGA logic occupation (a) and complexity of RC system routing (b). The location of RC system modules (e.g. PCIe interface, SeReCon etc) is not constrained in order to speed-up implementation time.

This section highlights quantitative results of the prototype SeReCon-enabled RC system implementation and the SeReCon resource cost analysis (including firmware). Implementation issues and proposed solutions are also highlighted.

### 7.3.2. Hardware Implementation Results

Table 7- illustrates the FPGA logic resources (LUTs, FFs and BRAMs) which are used by the prototype SeReCon-enabled RC system. Figure 7- illustrates the relative (percentage of the FPGA) resource costs of each of the main SeReCon-enabled RC system prototype elements, e.g. FPGA LUTs (a) and FPGA FF’s (b). Results show that the SeReCon resource cost is 36% of the size of the mid-size Xilinx Virtex-5 device (used in the ML505 board).

Table 7- illustrates the FPGA logic resources used by the SeReCon IP core. Figure 7- illustrates the relative (percentage of the FPGA) resource costs of each of the SeReCon elements, e.g. FPGA LUTs (Figure 7-a) and FPGA FF’s (Figure 7-b). Results show a significant amount of resources consumed by the DDR2 SDRAM memory controller (up to 38% of SeReCon size). The RC system prototype uses this memory in order to store the SeReCon firmware. The SeReCon security model and SeReCon-enabled RC system RoT assume that the SeReCon firmware is stored within the FPGA local memory, e.g. BRAM.

\(^70\) The current SeReCon implementation uses intermediate, dynamically allocated buffers? in the DDR2 SDRAM memory.
Figure 7- The FPGA Editor view of the SeReCon-enabled RC system implementation in the Xilinx Virtex-5 FPGA and location of the PR region. a. FPGA logic occupation. b. complexity of RC system routing.

<table>
<thead>
<tr>
<th>RC system module name</th>
<th>FFs (max 28800)</th>
<th>LUTs (max 28800)</th>
<th>BRAMs (36kbits per block)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeReCon</td>
<td>10480</td>
<td>10492</td>
<td>32&lt;sup&gt;71&lt;/sup&gt;</td>
</tr>
<tr>
<td>PR region</td>
<td>3840</td>
<td>3840</td>
<td>8</td>
</tr>
<tr>
<td>PCIe endpoint</td>
<td>3059</td>
<td>2486</td>
<td>6</td>
</tr>
<tr>
<td>PCIe endpoint wrapper</td>
<td>199</td>
<td>240</td>
<td>0</td>
</tr>
<tr>
<td>Config controller</td>
<td>162</td>
<td>247</td>
<td>2</td>
</tr>
<tr>
<td>PCIe BAR splitter</td>
<td>0</td>
<td>59</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7- FPGA logic resources used by the prototype SeReCon-enabled RC system.

<sup>71</sup> The RC system prototype uses the DDR2 SDRAM memory in order to store the SeReCon firmware. Thus, the required number of SeReCon RAM blocks does not include SeReCon firmware cost.
Figure 7 - The relative (percentage of FPGA resources) cost of the main SeReCon-enabled RC system prototype elements. a. FPGA LUT usage. b. FPGA FF usage.
### Chapter 7 - Case Study: SeReCon Architecture Implementation

<table>
<thead>
<tr>
<th>SeReCon module name</th>
<th>FFs (max 28800)</th>
<th>LUTs (max 28800)</th>
<th>36k BRAMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>1789</td>
<td>2294</td>
<td>0</td>
</tr>
<tr>
<td>BM CTRL (GPIO)</td>
<td>73</td>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>DDR2 IF</td>
<td>3708</td>
<td>2244</td>
<td>0</td>
</tr>
<tr>
<td>ICAP</td>
<td>418</td>
<td>1152</td>
<td>0</td>
</tr>
<tr>
<td>INT CTRL</td>
<td>157</td>
<td>113</td>
<td>0</td>
</tr>
<tr>
<td>LMB</td>
<td>6</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>MB_PLB</td>
<td>170</td>
<td>594</td>
<td>0</td>
</tr>
<tr>
<td>MDM_DEBUG</td>
<td>123</td>
<td>108</td>
<td>0</td>
</tr>
<tr>
<td>MICROBLAZE</td>
<td>1848</td>
<td>1966</td>
<td>3</td>
</tr>
<tr>
<td>PCIE IF</td>
<td>204</td>
<td>170</td>
<td>0</td>
</tr>
<tr>
<td>RS232</td>
<td>144</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>SYS_RESET</td>
<td>67</td>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>SYSACE IF</td>
<td>265</td>
<td>95</td>
<td>0</td>
</tr>
<tr>
<td>SYSMON</td>
<td>178</td>
<td>127</td>
<td>0</td>
</tr>
<tr>
<td>TIMER</td>
<td>361</td>
<td>308</td>
<td>0</td>
</tr>
<tr>
<td>TRNG</td>
<td>360</td>
<td>273</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7- FPGA logic resources used by the SeReCon IP core.
Figure 7- The relative (percentage of FPGA resources) cost of the SeReCon elements. a. FPGA LUT usage. b. FPGA FF usage.
7.3.3. SeReCon Firmware Implementation Results

Table 7- illustrates the FPGA memory resources used by the SeReCon firmware modules. Some modules, e.g. the ECC module and the ERDB module, include multiple files which represent the internal module hierarchy.

The total size of the SeReCon firmware is dominated by the ERDB module (95%). The ERDB module mainly contains the description of the FPGA routing (erdbo_routing.o, 42.3%) and PIP configuration bits (erdbo_pd.o, 47.6%). Further investigation of compact FPGA fabric representations, e.g. erdbo_routing and erdbo_pd organisation could optimise ERDB footprint.

<table>
<thead>
<tr>
<th>SeReCon module</th>
<th>Module files</th>
<th>Module size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes</td>
<td>aes_soft.o</td>
<td>25980</td>
</tr>
<tr>
<td>ecc</td>
<td>bignum.o</td>
<td>15424</td>
</tr>
<tr>
<td></td>
<td>ecdsa.o</td>
<td>6068</td>
</tr>
<tr>
<td></td>
<td>ecdsa_demo.o</td>
<td>20104</td>
</tr>
<tr>
<td></td>
<td>ellipticcurve.o</td>
<td>8188</td>
</tr>
<tr>
<td>erdb</td>
<td>bit_data_parser.o</td>
<td>7972 (0.2%)</td>
</tr>
<tr>
<td></td>
<td>bit_header-parser.o</td>
<td>1832 (&lt;0.1%)</td>
</tr>
<tr>
<td></td>
<td>cfg_analyser.o</td>
<td>16060 (0.4%)</td>
</tr>
<tr>
<td></td>
<td>erdbo Demo.o</td>
<td>9452 (0.3%)</td>
</tr>
<tr>
<td></td>
<td>erdbo_layout.o</td>
<td>19824 (0.5%)</td>
</tr>
<tr>
<td></td>
<td>erdbo_pd.o</td>
<td>1764372 (47.6%)</td>
</tr>
<tr>
<td></td>
<td>erdbo_routing.o</td>
<td>1565036 (42.2%)</td>
</tr>
<tr>
<td></td>
<td>erdbo_tg.o</td>
<td>235764 (6.4%)</td>
</tr>
<tr>
<td></td>
<td>erdbo_ws.o</td>
<td>32392 (0.9%)</td>
</tr>
<tr>
<td></td>
<td>far_seq_v5ls50t.o</td>
<td>54097 (1.46%)</td>
</tr>
<tr>
<td>debug</td>
<td>debug.o</td>
<td>4124 (0.1%)</td>
</tr>
<tr>
<td>icap</td>
<td>icap_demo</td>
<td>9092 (0.2%)</td>
</tr>
<tr>
<td>debug</td>
<td>menu.o</td>
<td>2748 (0.1%)</td>
</tr>
<tr>
<td>eidr</td>
<td>idr.o</td>
<td>12708 (0.3%)</td>
</tr>
<tr>
<td>mfs</td>
<td>mfs_demo.o</td>
<td>7296 (0.2%)</td>
</tr>
<tr>
<td>comm. (pcie)</td>
<td>pcie_demo.o</td>
<td>43484 (1.1%)</td>
</tr>
<tr>
<td>erdb</td>
<td>analyzer/verifier</td>
<td>serecon.o</td>
</tr>
<tr>
<td>main</td>
<td><a href="mailto:SeReCon@Intel.o">SeReCon@Intel.o</a></td>
<td>3984 (0.1%)</td>
</tr>
<tr>
<td>sysace/fats</td>
<td>sysace_demo.o</td>
<td>7752 (0.2%)</td>
</tr>
<tr>
<td>trng</td>
<td>trng.o</td>
<td>1032 (&lt;0.1%)</td>
</tr>
<tr>
<td>SeReCon firmware</td>
<td>serecon.elf</td>
<td>Summarised size = 3900417 (100%)</td>
</tr>
</tbody>
</table>

Table 7- FPGA memory resources used by the SeReCon firmware.
7.3.4. ScReCon-enabled RC System Prototype Implementation

Issues

The following implementation issue have been observed during prototyping of the ScReCon-enable RC system, and require address during a future implementation:

- **The unreliable Xilinx EAPR design flow support for Xilinx Virtex-5 FPGAs** results in a random implementation error. Listing 7- illustrates the EAPR critical exceptions which have occurred during the ScReCon-enabled RC system implementation. Exceptions appear to be related to the always-on ‘fake’ PIPs and appear randomly, e.g. re-implementation of an unchanged design results in errors occurring in a different phase (e.g. Xilinx ISE P&R Phase 10) or even successful completion of implementation with no errors. The Xilinx EAPR design flow support for Xilinx Virtex-5 devices is unreliable, e.g a successful implementation currently requires a number of time consuming iterations, and is not advised.

- **ScReCon resource cost**: Table 7- illustrates the relative (percentage of FPGA resources) cost of the ScReCon implementation in a range of modern Xilinx FPGAs, e.g. the largest Virtex-6 FPGA device, and also the mid-sized and the largest Virtex-5 FPGA. The ScReCon size introduces a reasonable (2%-5%) overhead only in the largest FPGAs. The ScReCon firmware size exceeds the amount of memory which is available in most of the currently available Xilinx devices, except for the Virtex-6 device. The ScReCon-enabled RC system prototype uses the DDR2 SDRAM memory for ScReCon firmware storage. This indicates the need for firmware optimisation, e.g. investigation of the optimal ERDB structure is suggested as future work. Generalisation of FPGA routing types and the use of device-specific data field sizes (e.g. bit fields, instead of generic ‘int’) could also be considered in order to reduce ScReCon firmware footprint. Further work could also investigate the use of external RAM encryption, which was proposed by Edmison [139], or software authentication as a complementary approach to use of scarce FPGA internal memory (BRAM) resources for the ScReCon firmware storage.

- **Incomplete ScReCon AES implementation**. Prototype ScReCon implementation does not support authentication-encryption (e.g. AES in EAX\(^{72}\) mode); only the CBC mode is supported. A solution could be to include an additional authentication field in the IP core entry in SafeLock. Also, the AES IP core which is included in the ScReCon element does not pass test routines provided with the IP core source code\(^{73}\). Thus, the IP core

---

\(^{72}\) EAX mode is a mode of operation for cryptographic block ciphers (http://en.wikipedia.org/wiki/EAX_mode)

\(^{73}\) This could be caused by experimental-only EAPR support for Virtex-5 FPGAs. The exact reason requires further investigation. Interestingly, the same AES IP core operates correctly when used as the PR IP core.
remains unused and the software-only implementation of the AES cipher and decipher is included in the SeReCon firmware (as a workaround). Additionally, the software implementation of AES slows down SeReCon-based PR.

- **The software ECC implementation is slow.** The SeReCon ECC uses the P192 elliptic curve [181] which offers only 96 secure bits (half of the key width). Thus the hardware accelerator for large number ECC operations would improve SeReCon performance during the DH shared key agreement with the IPV and would support longer ECC keys.

- **The TRNG IP core uses only 8 ROs.** The small RO number affects the TRNG quality, e.g. the output data stream is biased and the TRNG fails the standard tests for data randomness. The TRNG extension up to 32 ROs is a straightforward addition of RO macro cells and does not require a change to the TRNG API. The TRNG macrocell architecture could also be optimised towards the architecture of Virtex-5 (or Virtex-6) CLB.

- **The ERDB Analyser does not support the detection of IP core internal design errors,** e.g. short circuits between signals. Future work could improve the ERDB Analyser robustness by including this functionality. Also, the ERDB Analyser reports only the shape of external wires (without its unique name in the FPGA tile). The distinction between two FPGA tile wires, both having the same shape, is not possible. Thus the ERDB Verifier (Figure 1-e) could raise a false alarm in situation when two IP cores are isolated and but use routing (wires) of the same shape. This affects only the ERDB implemented in SeReCon. The FDAT (offline) version of ERDB contains an additional namespace database which eliminates this risk. The ERDB Verifier also uses a single static and a predefined reconfigurable region in order to support the EAPR design flow. In future work, an extension of the IP core verification is suggested in order to incorporate a model of multiple cores within the reconfigurable region.

- **The Microblaze architecture is closed-source.** Thus, public audit of the SeReCon element is not possible. Also, use of formal verification methods could significantly improve the robustness of SeReCon. Thus, open-source CPUs, e.g. OpenRISC\(^74\) or Leon\(^75\), is suggested for future SeReCon implementations. Also, the SeReCon architecture and firmware is not hardened against power analysis attacks which could expose the SeReCon security credentials, e.g. the EIDR contents or the SafeLock encryption keys. Thus, future work could also support SeReCon firmware and architecture modification in order to include DPA-preventing hardware primitives (e.g. WDDL logic) and algorithms (branch balancing etc).

- **The EIDR is emulated in SeReCon firmware.** Thus, the EIDR is assumed to be active after power-up and no check of BaseSFC configuration bitstream

---

\(^74\) OpenCores, OpenRISC processor ([http://opencores.org/project.or1k](http://opencores.org/project.or1k))

can be made. Also, an activity of the internal EIDR counters (AAC, LTC, FRC and MSG_NO) is emulated through the ‘backup-modify-restore’ operation on the EIDR contents. Future work could implement the EIDR using an external FPGA board which is connected to the RC system configuration interface in order to detect BaseSFC tampering.

- **The PCIe interface uses a generic (slow) PCIe reference design** which supports only single-word PCIe transactions. Future work could focus on more robust PCIe interface implementation which would support multi-word (burst) read/write PCIe transactions.

- **SeReCon does not currently deactivate expired IP cores**, e.g. when the IP core is active and its lifetime has expired. SeReCon includes the Timer IP core which could be used to generate periodic interrupts to MicroBlaze which are used to request a poll and check of the IP core license.

| Starting Router                                                                 |
|---|---|---|---|---|
| 1 | Phase 1: 16593 unrouted; REAL time: 52 secs |
| 2 | Phase 2: 15126 unrouted; REAL time: 54 secs |
| 3 | Phase 3: 5927 unrouted; REAL time: 1 mins 4 secs |
| 4 | Phase 4: 5927 unrouted; (140864) REAL time: 1 mins 5 secs |
| 5 | Phase 5: 6069 unrouted; (111829) REAL time: 1 mins 30 secs |
| 6 | Phase 6: 6072 unrouted; (111817) REAL time: 1 mins 31 secs |
| 7 | Phase 7: 0 unrouted; (117091) REAL time: 3 mins 5 secs |
| 8 | Updating file: top_routed.ncd with current fully routed design. |
| 9 | Pin<BRID:102377> to be detached from RUGNODE:FAKE_RFNODE not found |
| 10| Exception:Rf:Rf_DeviceMgr.c:221:1.7 - GetDevice called with bad index |
| 11| Pin<BRID:111380> to be detached from RUGNODE:FAKE_RFNODE not found |
| 12| Exception:Rf:Rf_DeviceMgr.c:221:1.7 - GetDevice called with bad index |
| 13| Pin<BRID:114121> to be detached from RUGNODE:FAKE_RFNODE not found |
| 14| Pin<BRID:114172> to be detached from RUGNODE:FAKE_RFNODE not found |
| 15| Exception:Rf:Rf_DeviceMgr.c:221:1.7 - GetDevice called with bad index |

Listing 7- EAPR critical exceptions which occur during the SeReCon-enabled RC system implementation.

<table>
<thead>
<tr>
<th>FPGA device</th>
<th>SeReCon size</th>
<th>10480 FFs</th>
<th>10492 LUTs</th>
<th>24712 BRAM Kbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mid-size FPGA (ML505)</td>
<td></td>
<td>36%</td>
<td>36%</td>
<td>1144 % (2160)</td>
</tr>
<tr>
<td>(XC5VLX50T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest Virtex-5 FPGA</td>
<td></td>
<td>5%</td>
<td>5%</td>
<td>212% (11664)</td>
</tr>
<tr>
<td>(XC5VLX330T)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Largest Virtex-6 FPGA</td>
<td></td>
<td>1%</td>
<td>2%</td>
<td>95% (25920)</td>
</tr>
<tr>
<td>(XC6VLX760)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7- The percentage resource cost of the SeReCon implementation in modern Xilinx FPGAs.
7.4. **RC System Demonstrator**

7.4.1. **Introduction**

This section reports and describes the SeReCon-enabled RC system demonstrator application (Figure 7-). The communication library of the RC system demonstrator and host-side SeReCon API are highlighted. Demonstrator application results are also reported.

The demonstrator application uses the interactive Python command line in order to provide a live experience of the RC system behaviour. This supports non-standard ad-hoc system tests and interactive RC system application development.

The RC system communication library uses the internally developed PCIe device driver which provides Python API for communication with the RC system and SeReCon. The RC system communication library supports read and write of the RC system registers, BM control and identification of the active PR module.

The SeReCon API supports SeReCon control, status monitoring and bidirectional data transfer between SeReCon and demonstrator application.

The demonstrator application shows that SeReCon detects additional external PIPs in IP cores which are created using the genuine Xilinx EAPR design flow. This confirms that even genuine IP cores, when developed in a multi-party environment, could include implicit communication channels and could introduce security risks.

7.4.2. **RC System Communications Library**

The demonstrator application communicates with the SeReCon-enabled RC system through the single-lane (x1) PCIe interface. Table 7- describes the RC system communication library which is used by the host-side SeReCon API. The RC system communication library is implemented in Python and uses the unpublished driver (C++/Python) for the Xilinx PCIe reference design. The RC system communication library supports read and write of RC system registers which are mapped to PCIe BARs. RC system BM control and identification (ID read) of the active PR module is also supported.

---

76 The Python/C++ driver for the PCIe reference design was implemented by Krzysztof Kościuszkwiewicz.
Table 7- Description of the RC system communication library which is used by the host-side SeReCon API.

### 7.4.3. Host-side SeReCon API

Table 7- describes the host-side SeReCon API which is implemented in Python and used by the SeReCon-enabled RC system demonstrator application. The API supports SeReCon control (CTRL register), status monitoring (STAT register) and bidirectional data transfer (DIN and DOUT FIFOs).

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg</td>
<td>number, value, BAR</td>
<td>Sets the number register in BAR to value (if value is not none). Returns current content of the register.</td>
</tr>
<tr>
<td>bm</td>
<td>mode</td>
<td>Enables and disables RC system BMs. Returns the current state of the BM enable register.</td>
</tr>
<tr>
<td>moduleid</td>
<td>-</td>
<td>Returns 4-bit PR IP core ID.</td>
</tr>
<tr>
<td>leds</td>
<td>value</td>
<td>Uses value [7:0] in order to control the ML505 board LEDs. Returns current status of the Configuration Controller LED register.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbRdCnt</td>
<td>-</td>
<td>Returns MB_READ counter.</td>
</tr>
<tr>
<td>mbWrCnt</td>
<td>-</td>
<td>Returns MB_WRITE counter.</td>
</tr>
<tr>
<td>mbDout</td>
<td>-</td>
<td>Returns MB_DOUT value.</td>
</tr>
<tr>
<td>mbDin</td>
<td>value</td>
<td>Sets MB_DIN to value. Returns current content.</td>
</tr>
<tr>
<td>mbWrite</td>
<td>-</td>
<td>Writes MB_DIN data to MB FIFO (asserts pcie_interface_0_mb_dout_wr_pin signal).</td>
</tr>
<tr>
<td>mbRead</td>
<td>-</td>
<td>Reads MB FIFO data to MB_DOUT (asserts pcie_interface_0_mb_din_rd_pin signal).</td>
</tr>
<tr>
<td>mbSend</td>
<td>word</td>
<td>Writes the 32-bit word to the Configuration Controller DIN FIFO. Waits if the FIFO is full and timeouts after 100 failed retries.</td>
</tr>
<tr>
<td>mbReceive</td>
<td>-</td>
<td>Returns the 32-bit word which is read from the Configuration Controller DOUT FIFO. Raises an exception if the FIFO is empty.</td>
</tr>
<tr>
<td>mbStat</td>
<td>-</td>
<td>Returns the SeReCon status word.</td>
</tr>
<tr>
<td>mbRst</td>
<td>mode</td>
<td>Asserts/deasserts the SeReCon reset line and returns the current state of the reset line.</td>
</tr>
<tr>
<td>mbCmd</td>
<td>cmd</td>
<td>Sets SeReCon command word. Returns current command.</td>
</tr>
<tr>
<td>mbIsFull</td>
<td>fifo</td>
<td>Returns true if fifo (DIN/DOUT) FIFO is full.</td>
</tr>
<tr>
<td>mbIsEmpty</td>
<td>fifo</td>
<td>Returns true if fifo (DIN/DOUT) FIFO is empty.</td>
</tr>
</tbody>
</table>

Table 7- Description of the host-side SeReCon API which is used by the RC system demonstrator application.
7.4.4. Demonstrator Application Results

The demonstrator application uses the interactive Python command line in order to provide a live experience of the RC system behaviour. This supports non-standard ad-hoc system tests and interactive RC system application development.

Table 7- describes the SeReCon demonstrator API which is tested using the interactive Python command line. Table 7- describes SeReCon demonstrator API support for RC system interactive debugging. Table 7- describes SeReCon demonstrator API support for testing activated IP cores and LEDs on the RC system board.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initEidr</td>
<td>-</td>
<td>SeReCon initialises the EIDR register.</td>
</tr>
<tr>
<td>getMsgNo</td>
<td>-</td>
<td>Return the current EIDR MNC value (msgNo).</td>
</tr>
<tr>
<td>instIpCore</td>
<td>name, vendor, msgNo, dest</td>
<td>Demonstrate SeReCon-based IP core (name) installation in the RC system dest (LIPS or MFS).</td>
</tr>
<tr>
<td>actIpCore</td>
<td>name, msgNo, dest</td>
<td>Demonstrate SeReCon-based IP core (name) activation in the RC system dest (LIPS or MFS).</td>
</tr>
<tr>
<td>deactIpCore</td>
<td>name, msgNo, dest</td>
<td>Demonstrate SeReCon-based IP core (name) deactivation.</td>
</tr>
<tr>
<td>genIpvSKey</td>
<td>vendor, msgNo, source</td>
<td>Demonstrate ECDSA-based SeReCon DH shared-key calculation.</td>
</tr>
<tr>
<td>removeFile</td>
<td>file, fs</td>
<td>Remove file from fs file system (CF or MFS).</td>
</tr>
<tr>
<td>getFileNames</td>
<td>fs</td>
<td>Return dictionary of files and their sizes in the fs file system (CF or MFS).</td>
</tr>
<tr>
<td>sendFile</td>
<td>file, dest</td>
<td>Send file to SeReCon which stores it in dest (LIPS or MFS).</td>
</tr>
<tr>
<td>receiveFile</td>
<td>file, source</td>
<td>Receive file from SeReCon which reads it from source (LIPS or MFS).</td>
</tr>
</tbody>
</table>

Table 7- Description of the SeReCon demonstrator API which is tested using the interactive Python command line. The API description is continued in Table 7- and Table 7-.
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>restoreFs</td>
<td>source, target</td>
<td>Restore the SeReCon target files, e.g. CF (LIPS) or MFS, using the host source directory content.</td>
</tr>
<tr>
<td>resetMfs</td>
<td>-</td>
<td>Reset MFS.</td>
</tr>
<tr>
<td>getMfsStats</td>
<td>-</td>
<td>Return the number of MFS blocks (free and used).</td>
</tr>
<tr>
<td>checkFile</td>
<td>file, source</td>
<td>Check file in dest (LIPS or MFS). Returns file status</td>
</tr>
<tr>
<td>getFpgaFrame</td>
<td>block, bottom, row, major, minor</td>
<td>Return FPGA configuration frame (41 32-bit words) which is addressed by the block type, FPGA half (bottom), clock row, major and minor frame number.</td>
</tr>
<tr>
<td>setFpgaReg</td>
<td>reg, values</td>
<td>Write a list of 32-bit words (values) into the FPGA configuration register (reg).</td>
</tr>
<tr>
<td>getFpgaReg</td>
<td>reg</td>
<td>Return the current content of the FPGA configuration register (reg).</td>
</tr>
<tr>
<td>tamperEidr</td>
<td>-</td>
<td>Mimic malicious EIDR tampering. This resets the EIDR content.</td>
</tr>
<tr>
<td>backupEidr</td>
<td>target</td>
<td>This is the prototype-only debug support function which uses target (LIPS or MFS) in order to store a backup copy of the EDIR content.</td>
</tr>
<tr>
<td>restoreEidr</td>
<td>source</td>
<td>This is the prototype-only debug support function which restores the EDIR content from the source backup copy (in the LIPS or MFS).</td>
</tr>
<tr>
<td>initPr</td>
<td>-</td>
<td>Initialise RC system PR region using default (blank) configuration.</td>
</tr>
<tr>
<td>allowRiskyIpCore</td>
<td>-</td>
<td>SeReCon overrides the results of the ERDB-based verification prior to PR. This allows activation of insecure IP cores.</td>
</tr>
<tr>
<td>getEidrCnts</td>
<td>-</td>
<td>Return current values of EIDR counters (AAC, LTC and FRC).</td>
</tr>
<tr>
<td>getSysState</td>
<td>-</td>
<td>Return RC system PR region data, e.g. region coordinates, total number and list of region external PIPs. Configuration and isolation regions are supported.</td>
</tr>
<tr>
<td>resetSysAce</td>
<td>-</td>
<td>Enforce SysAce-based FPGA reconfiguration using the default (.ACE) configuration file.</td>
</tr>
<tr>
<td>resetIcap</td>
<td>-</td>
<td>Reset ICAP.</td>
</tr>
<tr>
<td>loadPrFile</td>
<td>file, source</td>
<td>This is the prototype-only debug support function which uses ICAP in order to reconfigure FPGA using the PR bitstream loaded from source.</td>
</tr>
</tbody>
</table>

Table 7- Continued description of the SeReCon demonstrator API which includes support for RC system interactive debugging.
### Function Parameters Description

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>leds</code></td>
<td><code>val</code></td>
<td>Use <code>val</code> bits [7:0] to control (on/off) 8 LEDs on the ML505 board.</td>
</tr>
<tr>
<td><code>test</code></td>
<td><code>arg1, arg2</code></td>
<td>Test and demonstrate simple math IP cores, e.g. adder and multiplier which are loaded in PR region. <code>Arg1</code> and <code>arg2</code> are parameters used by the core.</td>
</tr>
<tr>
<td><code>testAesEnc</code></td>
<td><code>key, txtIn</code></td>
<td>Test and demonstrate the AES Cipher IP core which is loaded in PR region. <code>Key</code> and <code>txtIn</code> are parameters used by the core.</td>
</tr>
<tr>
<td><code>testAesDec</code></td>
<td><code>key, txtIn</code></td>
<td>Test and demonstrate the AES decipher IP core which is loaded in PR region. <code>Key</code> and <code>txtIn</code> are parameters used by the core.</td>
</tr>
</tbody>
</table>

*Table 7- Continued description of the SeReCon demonstrator API which includes functional tests for activated IP cores and LEDs on the RC system board.*

Listing 7- illustrates the SeReCon-enabled RC system demonstrator application algorithm which includes SeReCon initialisation and credentials exchange (lines 1-10), installation of IP cores in the SeReCon-enabled RC system (lines 13-36) and their activation (lines 38-40). Appendix E illustrates the complete output of the SeReCon debug console during SeReCon-enabled RC system demonstration.

**SeReCon initialisation and credentials exchange.** The demonstrator application initialises SeReCon EIDR (line 5) and receives the SeReCon public key (line 7). The IPV security credentials are generated (line 8) prior to sending the IPV public key to SeReCon (line 10).

**IP cores installation in the RC system.** For every IP core in the list (line 2) the demonstrator requests the current EIDR MNC value (line 15) and creates the IPV message (line 17) which is sent to SeReCon (line19) in order to generate a shared key using the DH key agreement protocol (line 21). The IPV reads the SeReCon reply (line 25) and locally generates the DH shared key (line 27) using the updated value of the EIDR MNC value (line 23). The IPV uses the shared key in order to prepare the IP package (line 29), e.g. the encrypted IP core PR bitstream and its license, which is sent to SeReCon (lines 31-33). The demonstrator also commands SeReCon to install the IP core (line 35).

Listing 7-, Listing 7-, Listing 7-, Listing 7- and Listing 7- illustrate fragments of the SeReCon debug console output which are related to IP core analysis (during RC system demonstration). SeReCon analyses adder (Listing 7-), blank (Listing 7-), multiplier (Listing 7-), AES encoder (Listing 7-) and AES decoder IP cores (Listing 7-).

**IP cores activation in the RC system.** The demonstrator application uses the IP core name and the EIDR MNC value (from the IP core installation) in order to activate the IP cores (line 40).
Chapter 7 - Case Study: SeReCon Architecture Implementation

Listing 7-

Listing 7- Listing 7-, Listing 7- and Listing 7- illustrate fragments of the SeReCon debug console output which are related to verification of IP core compatibility with current SeReCon-enabled RC system state (during IP core activation). SeReCon attempts to activate adder (Listing 7-), blank (Listing 7-), multiplier (Listing 7-), AES encoder (Listing 7-), and AES decoder IP core (Listing 7-). Lines 8 and 10 in Listing 7-, Listing 7-, Listing 7- and Listing 7- show that SeReCon detects additional external PIPs during verification of IP core compatibility with the RC system current state. Those PIPs could be used to setup implicit communication channel so SeReCon writes an activation report file (line 15), cancels the IP core activation (line 19) and rolls back the IP core license (line 20).

77 The blank IP core in Listing 7-b passed the test because it is used as the default (reference) layout of the 'empty' PR region.

78 The report includes list of additional PIPs in IP core configuration region and isolation region. Its structure is compatible with ERDB Analiser report file and can be read using FDAT tool.
All affected IP cores (*adder, multiplier AES cipher* and *AES decipher*) are created using the genuine Xilinx EAPR design flow which ensures that the static part of the design (e.g. BaseSFC) does not share routing with PR IP cores, except explicitly defined BMs. This non-sharing policy cannot be ensured in a multi-party PR design flow where IP cores are delivered through third-party IPVs with no knowledge of the static design, e.g. third party IP core which is delivered as an update to the already-deployed SeReCon-enabled RC system. This example shows that even genuine IP cores, when developed in a multi-party environment, could include implicit communication channels and could introduce security risks.

Listing 7- Fragment of the SeReCon debug console output which is related to Adder IP core analysis.

```plaintext
1 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
2 - read header
3   Header data:
4      - HeaderLength : 86
5      - BitstreamLength : 145732
6      - DesignName : prm_adder_partial.ncd
7      - PartName : 5vlx50tff1136
8      - Date : 2009/10/27
9      - Time : 19: 7:46
10   - parse bitstream...OK
11      Region (x,y): (10,0) to (42,43)
12   - detect external pips...OK
13      Found 2699 external pips (2254 real, 445 fake).
14   - analyse isolation boundary...OK
15      Region (x,y): (10,0) to (42,45)
16   - detect io pips...OK
17      Found 2468 io pips (2154 real, 314 fake).
18 Analysis finished successfully.
```

Listing 7- Fragment of the SeReCon debug console output which is related to blank design analysis.

```plaintext
1 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
2 - read header
3   Header data:
4      - HeaderLength : 85
5      - BitstreamLength : 126420
6      - DesignName : pblock_prm_blank.ncd
7      - PartName : 5vlx50tff1136
8      - Date : 2009/10/27
9      - Time : 19:36:42
10   - parse bitstream...OK
11      Region (x,y): (10,0) to (42,43)
12   - detect external pips...OK
13      Found 2684 external pips (2239 real, 445 fake).
14   - analyse isolation boundary...OK
15      Region (x,y): (10,0) to (42,45)
16   - detect io pips...OK
17      Found 2453 io pips (2139 real, 314 fake).
18 Analysis finished successfully.
```
Chapter 7 - Case Study: SeReCon Architecture Implementation

Listing 7- Fragment of the SeReCon debug console output which is related to Multiplier IP core analysis.

<table>
<thead>
<tr>
<th>ERDB analyser v.1.0 (release Jan 29 2010, 18:40:42)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- read header</td>
</tr>
<tr>
<td>Header data:</td>
</tr>
<tr>
<td>- HeaderLength : 91</td>
</tr>
<tr>
<td>- BitstreamLength : 150968</td>
</tr>
<tr>
<td>- DesignName : prm_multiplier_partial.ncd</td>
</tr>
<tr>
<td>- PartName : 5vlx50tff1136</td>
</tr>
<tr>
<td>- Date : 2009/10/27</td>
</tr>
<tr>
<td>- Time : 19:14: 0</td>
</tr>
<tr>
<td>- parse bitstream...OK</td>
</tr>
<tr>
<td>Region (x,y): (10,0) to (42,43)</td>
</tr>
<tr>
<td>Found 2709 external pips (2264 real, 445 fake).</td>
</tr>
<tr>
<td>- analyse isolation boundary...OK</td>
</tr>
<tr>
<td>Region (x,y): (10,0) to (42,45)</td>
</tr>
<tr>
<td>Found 2478 io pips (2164 real, 314 fake).</td>
</tr>
<tr>
<td>Analysis finished successfully.</td>
</tr>
</tbody>
</table>

Listing 7- Fragment of the SeReCon debug console output which is related to AES Encoder IP core analysis.

<table>
<thead>
<tr>
<th>ERDB analyser v.1.0 (release Jan 29 2010, 18:40:42)</th>
</tr>
</thead>
<tbody>
<tr>
<td>- read header</td>
</tr>
<tr>
<td>Header data:</td>
</tr>
<tr>
<td>- HeaderLength : 88</td>
</tr>
<tr>
<td>- BitstreamLength : 165696</td>
</tr>
<tr>
<td>- DesignName : prm_encoder_partial.ncd</td>
</tr>
<tr>
<td>- PartName : 5vlx50tff1136</td>
</tr>
<tr>
<td>- Date : 2009/10/27</td>
</tr>
<tr>
<td>- Time : 19:20:24</td>
</tr>
<tr>
<td>- parse bitstream...OK</td>
</tr>
<tr>
<td>Region (x,y): (10,0) to (42,43)</td>
</tr>
<tr>
<td>Found 3330 external pips (2885 real, 445 fake).</td>
</tr>
<tr>
<td>- analyse isolation boundary...OK</td>
</tr>
<tr>
<td>Region (x,y): (10,0) to (42,45)</td>
</tr>
<tr>
<td>Found 3064 io pips (2750 real, 314 fake).</td>
</tr>
<tr>
<td>Analysis finished successfully.</td>
</tr>
</tbody>
</table>
Chapter 7 - Case Study: SeReCon Architecture Implementation

Listing 7- Fragment of the SeReCon debug console output which is related to AES Decoder IP core analysis.

```plaintext
ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
- read header
  Header data:
  - HeaderLength : 88
  - BitstreamLength : 168576
  - DesignName : pmn_decoder_partial.ncd
  - PartName : 5vlx50tff1136
  - Date : 2009/10/27
  - Time : 19:27:4
- parse bitstream...OK
Region (x,y): (10,0) to (42,43)
detect external pips...OK
  Found 3869 external pips (3424 real, 445 fake).
analyse isolation boundary...OK
Region (x,y): (10,0) to (42,45)
detect io pips...OK
  Found 3594 io pips (3280 real, 314 fake).
Analysis finished successfully.
```

Listing 7- Fragments of the SeReCon debug console output which is related to verification of Adder IP core compatibility with the current RC system state (during IP core activation).

```plaintext
ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
- get IP core report 'add.d01'...OK
- make diff with system state...OK
  Verification report:
  - IP core configuration region: (10,0 - 42,43) ...OK
  - IP core isolation region : (10,0 - 42,45) ...OK
  - external pips found : 2699 (2254 real, 445 fake)
  - unmatched ext pips : 15 (15 real, 0 fake)
  - io pips found : 2468 (2154 real, 314 fake)
  - unmatched io pips : 15 (15 real, 0 fake)

  !!!WARNING!!!
  Potentially dangerous IP core!!!

- write report file 'add.t01'...OK
Verification finished without errors.
IP core 'add' violates security requirements for reconfigurable region.
- check security bypass flag...OFF
IP core activation cancelled due to security risk.
- roll back license update...OK
Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.
```

---

- 172 -
Chapter 7 - Case Study: SeReCon Architecture Implementation

Listing 7- Fragments of the SeReCon debug console output which is related to verification of blank design compatibility with the current RC system state (during IP core activation).

```
1 ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
2 - get IP core report 'blank.d03'...OK
3 - make diff with system state...OK
4 Verification report:
5   - IP core configuration region: (10,0 - 42,43)...OK
6   - IP core isolation region  : (10,0 - 42,45) ...OK
7   - external pips found      : 2684 (2239 real, 445 fake)
8   - unmatched ext pips       : 0 (0 real, 0 fake)
9   - io pips found            : 2453 (2139 real, 314 fake)
10  - unmatched io pips        : 0 (0 real, 0 fake)
11 Verification finished without errors.
12 IP core 'blank' is safe.
13 - get IP core from "blank.i03" file...OK
14 - check bitstream header...OK
15 - disable BM interface...OK
16 - load IP core to ICAF...OK
17 - update system state...OK
18 IP core 'blank.i03' (msg 03) activated.
19 - enable BM interface...OK
20 Waiting for ACK...OK
21 Waiting for NULL...OK
22 PCIe command finished.
```

Listing 7- Fragments of the SeReCon debug console output which is related to verification of Multiplier IP core compatibility with the current RC system state (during IP core activation).

```
1 ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
2 - get IP core report 'mul.d05'...OK
3 - make diff with system state...OK
4 Verification report:
5   - IP core configuration region: (10,0 - 42,43)...OK
6   - IP core isolation region  : (10,0 - 42,45) ...OK
7   - external pips found      : 2709 (2264 real, 445 fake)
8   - unmatched ext pips       : 25 (25 real, 0 fake)
9   - io pips found            : 2478 (2164 real, 314 fake)
10  - unmatched io pips        : 25 (25 real, 0 fake)
11
12 !!!WARNING!!!
13 Potentially dangerous IP core!!!
14 15 - write report file 'mul.t05'...OK
16 Verification finished without errors.
17 IP core 'mul' violates security requirements for reconfigurable region.
18 - check security bypass flag...OFF
19 IP core activation cancelled due to security risk.
20 - roll back license update...OK
21 Waiting for ACK...OK
22 Waiting for NULL...OK
23 PCIe command finished.
```
Chapter 7 - Case Study: SeReCon Architecture Implementation

Listing 7- Fragments of the SeReCon debug console output which is related to verification of AES encoder IP core compatibility with the current RC system state (during IP core activation).

Listing 7- Fragments of the SeReCon debug console output which is related to verification of AES decoder IP core compatibility with the current RC system state (during IP core activation).
7.5. Proposal: SeReCon Application Within The SDR Device

7.5.1. Introduction

This section describes the SDR device prototype and proposes how SeReCon element can be included within the SDR RC system. Modifications to the SeReCon implementation required to integrate SeReCon within the prototype SDR device are also described.

7.5.2. SDR Device Block Diagram

Figure 7-a illustrates the SDR device prototype\textsuperscript{79}. The SDR device includes a radio interface, communication interface and the RC System-in-Package (SiP). Figure 7-b highlights System-in-Package (SiP) internals, e.g. embedded FPGA fabrics. Figure 7- illustrates the SiP block diagram. The SIP includes a number of FPGAs which are used for digital RF signal processing and the SDR application which is implemented in the Linux-based embedded control system running on an ARM CPU. The SIP block diagram also includes the suggested location of the SeReCon element which could manage secure FPGA reconfiguration within the SDR system and could provide SDR IP core protection. Proposed integration with the CADBUS interface and FPGA ICAPs is also highlighted.

\textsuperscript{79} SDR device prototype and block diagram included with the permission of Configurable Computing Lab, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.
7.5.3. Proposed SeReCon Modifications For Embedding Within The Prototype SDR System

Integrating SeReCon within the prototype SDR device would require only minor changes in the SeReCon hardware and SDR prototype. Modifications include an updated SeReCon communication/control interface and extension of the SeReCon interface in order to enable and control multiple ICAPs in remote FPGAs (see Figure 7-). Also, a SeReCon firmware update and small configuration update of SIP FPGAs are also required in order to support ICAP-based reconfiguration using SeReCon which is implemented in the remote FPGA (see Figure 7-). The required modifications are outlined below.

**SeReCon communication/control interface.** The SDR application uses a simple Control-Address-Data (CADBUS) bus-based communication interface which provides register-based access to the SDR IP cores implemented in the SIP FPGAs (Figure 7-). The SeReCon element also uses a register-based control interface (SeReCon PCIe interface IP core) in order to communicate with the RC system Configuration Ctrl IP core (Figure 7-). Thus, an additional wrapper IP core is required in order to provide an interface bridge between the CADBUS and SeReCon.

**Extension of the SeReCon interface.** The SeReCon prototype uses a single ICAP element which is connected internally. In the SDR application, SeReCon is
required to control multiple ICAPs in external FPGAs. Thus, SeReCon should provide an external ICAP interface with multiple enable signals which could select the active (FPGA) ICAP, e.g. using the GPIO IP core which is available from Xilinx EDK. Multiple ICAP support also requires changes to the SeReCon firmware which must be updated in order to facilitate multiple FPGA configurations, e.g. through the implementation of the RC system state context switching. Static configuration update of the SIP FPGAs is also required in order to provide exclusive SeReCon access to ICAP.

7.6. Chapter Summary

This chapter reports on the implementation and application of the prototype SeReCon-enabled RC system using Xilinx Virtex-5 FPGA technology. The implementation of SeReCon internal elements, the main RC system elements and example PR IP cores is described. Analysis of the SeReCon FPGA resource usage and RC system prototype implementation issues is included.

The implemented RC system uses four IP cores in order to demonstrate the SeReCon-based PR, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher. The VHDL model for each of these IP cores is included in the thesis DVD.

The SeReCon IP core is a CPU-based system which is implemented using the Xilinx EDK software. SeReCon uses an embedded 32-bit MicroBlaze processor which is operating at 125 MHz.

The chapter also reports the SeReCon demonstrator which includes the demonstrator application, implemented in Python and executed on the Intel server. The RC system communication library of the SeReCon demonstrator and the host-side SeReCon API are highlighted. The Intel server includes a Xilinx Virtex-5 FPGA evaluation board which contains the prototype of SeReCon-enabled RC system (and SeReCon IP core), connected to the host server using the PCIe interface. This chapter reports and describes the SeReCon-enabled RC system prototype, including implementation results. The demonstrator application results confirm the feasibility of the SeReCon-based secure reconfiguration in the largest Xilinx Virtex-6 FPGAs (e.g. XC6VLX760). The chapter provides detailed insight into the operation of the prototype RC system during the SeReCon (and EIDR) initialisation, IP core installation and activation.

The demonstrator application shows that even genuine IP cores, when developed in multi-party environment, could include implicit communication channels and could introduce security risks.

This chapter also describes the SDR device prototype and proposes how the SeReCon element can be included within the SDR RC system. Modifications to the SeReCon implementation required to integrate SeReCon within the prototype SDR device are also highlighted.
Chapter 8. Conclusions And Future Work

8.1. Introduction

This thesis contributes to IP security and IP usage accounting in PR Xilinx FPGA-based RC systems. This chapter concludes the thesis and proposes future work.

8.2. RC Systems And Security Risks In The Multi-Party Design Environment

This thesis reviews and describes the RC system application domain and advantages offered by RC systems over general purpose processors and ASICs. FPGA technology and architectures are introduced and the FPGA-based RC system design flow and PR are described.

This thesis investigates secure IP management in RC systems and describes FPGA-based RC systems in context of SDR application. The SDR application exploits the PR technology in order to update RC system configuration using hardware components (IP cores) which are obtained from third party IP vendors within the multi-player design environment.

A consequence of the multi-player environment is an increased risk to system integrity and to design IP protection, e.g. design IP theft, cloning, counterfeiting and tampering. Also, current IP infringement countermeasures do not support IP core usage accounting and license enforcement in a multi-party design flow and in active (deployed) PR-enabled RC systems. This approach hinders massive-scale adoption of third-party IP cores in high assurance RC systems.

The provision of IP core transaction-based and metered access licensing models in addition to a protection model for PR systems could increase the use of IP cores in reconfigurable consumer devices. A trusted license enforcement scheme requires methods for reliable control of IP core utilisation in the RC system, e.g. enforcement of counted IP core activation and IP core run time metering.

This thesis reviews the state of the art in RC security. A motivating example on security risks in PR FPGA systems is provided. Risks of the malicious IP core designs and threat of rogue EDA software also are highlighted. Security countermeasures supported by Xilinx FPGA fabric are described prior to critical examination of the reported work on the RC system integrity protection. The IP theft countermeasures and the principle of IP licensing models are also described. Directions of research activity in the field of RC systems security are two-fold,
focusing on system integrity protection and design IP protection. System integrity protection measures aim towards seamless integration of multiple externally-developed IP cores into a stable and trustworthy system. Design IP privacy protection must be ensured to commercially available third-party IP core vendors. This is not ensured where the system integrator is in full control of, and has unrestricted access to, all design modules, including third-party IP cores. Current design IP protection methods focus on the confidentiality of the IP core implementation, mainly by using authentication and encryption protocols, though without considering the risks caused by including erroneous or malicious IP cores. This exhibits contradictory goals of system protection (integrity) and design IP protection (privacy). Also, none assume a system model which includes IP protection of untrusted third-party cores while guaranteeing system integrity, e.g. protecting against design errors in third-party IP. Thus, new IP-aware methods for development of trustworthy systems are required.

8.3. SeReCon, A RoT For PR FPGAs

The thesis proposes SeReCon, a RoT for PR FPGAs with RC system usage accounting. The requirements of credentials storage in a secure RoT and the implementation of usage accounting for RC systems are reviewed. The thesis proposes and describes the EIDR element, which is a novel extension to the FPGA fabric. EIDR provides non-volatile storage of RoT credentials and RC system usage data. Techniques for storage of the RoT security credentials and RC system usage accounting data in modern FPGAs are also reviewed. The suitability of SRAM-based configuration memory is discussed. The EIDR element prototype implementation in a Virtex-5 LXT device (ML505 Board) and the register-based EIDR control/status interface (which is implemented in the FPGA user-logic) are reported prior to description of EIDR API functions, which are provided by the SeReCon EIDR driver.

The thesis proposes secure multi-party RoT credentials generation process and highlights activity of SeReCon and various parties (e.g. SI, TA, IPV) during RoT initialisation. The proposed RoT initialisation process supports public audit of the RC device security (device, source and implementation files review) and guarantees exclusive and authenticated access to the sensitive part of the RC system security credentials only for the legitimate system, e.g. SeReCon RoT. The SeReCon-based RoT is immune to credentials leakage as a result of a future successful attack on the TA.
8.4. FDAT Framework For Low-Level FPGA Design Analysis

The thesis proposes FPGA Design Analysis Tool (FDAT) framework for low-level FPGA design analysis. Risks in a multi-party PR design flow are discussed. The thesis investigates the issue of implicit communication channels which could be created during PR using third party IP cores. The need for low (bitstream) level design analysis is highlighted and FDAT is proposed and applied as a solution.

FDAT is the first available toolset to provide high-level and unrestricted access to the low-level description of the Xilinx FPGA fabric and the user design at the netlist- and bitstream-level. The GUI front-end extends FDAT functionality by providing customised visualisations of the design and FPGA resources. Use of a Python programming language provides clean and self-documenting code (algorithm syntax), unrestricted tool customisation and defines higher-level abstractions for design analysis.

FDAT has been developed around the concept of component and recipe separation. Components provide the necessary data and abstract models (design, device or bitstream), while recipes describe policies (algorithms) defining data model usage. This separation enables the reuse of high-level (model-specific) recipes which can be ported to other systems, e.g. SeReCon. Also, the hierarchical recipe structure supports a range of high-level analysis flows and offers virtually unlimited functionality extensions, thus supporting domain-specific design analysis.

FDAT enables the generation of the ERDB embedded database containing a minimal description of the FPGA fabric and bitstream for use by the SeReCon IP core to perform on-line verification of IP core routing. The FDAT framework offers a generic and unified support for analysis of designs targeting all Xilinx architectures. The FDAT framework has been tested using Virtex-II Pro and Virtex-5 LXT designs and device descriptions.

The thesis reports porting FDAT functionality to SeReCon for on-line Virtex-5 bitstream analysis. Considerations in creating an ERDB (including FPGA fabric description) are highlighted. The feasibility and accuracy of the ERDB-based IP core routing verification is demonstrated.
8.5. SeReCon Initialisation And Operation For Secure FPGA Reconfiguration

The thesis proposes the SeReCon initialisation which provides secure FPGA reconfiguration and usage accounting, e.g. SeReCon-based IP core installation, activation and deactivation. The internal state diagram and the block diagram of the SeReCon IP core are also described and the SeReCon firmware stack is highlighted.

The thesis describes the SeReCon RoT operations during the RoT initialisation process which occurs at the trusted TA site in order to minimise the risk of malicious device tampering and support independent scrutiny, e.g. through public audit. The SeReCon initialisation process includes EIDR credentials initialisation, RC system security credentials generation and publication. SeReCon exploits the EIDR element in order to provide design IP protection and executes in-system design analysis of new IP cores to maintain the integrity of the RC system.

IP core installation is performed online, once for every new IP core. A SafeLock scheme for IP core security credentials protection is highlighted. The process of establishing the shared encryption key between the IPV and SeReCon, using the Diffie-Hellmann (DH) shared key agreement protocol is also described.

During the IP core activation process, SeReCon performs verification of the IP core compliance with the current RC system state in order to protect the integrity of the BaseSFC and to countermeasure the risk of implicit communication channel setup. IP core license validation and RC system reconfiguration are also described. License validation prior to RC system PR enforces both transaction based and metered usage IP business models. The IP core deactivation process removes the remains of previously activated IP cores which could interact with the current system configuration, thus leading to RC system integrity issues. IP core deactivation ensures that the unused IP core configuration is removed from the FPGA configuration memory.

8.6. Case Study: SeReCon Architecture Implementation And Proposed Application In SDR Device

The thesis reports the SeReCon case study which demonstrates SeReCon architecture implementation, and proposes the potential SeReCon application in the SDR device. The prototype of the SeReCon-enabled RC system is implemented using the Xilinx Virtex-5 FPGA. The implementation of SeReCon internal elements, the main RC system elements and example PR IP cores is described. The implemented RC system uses four IP cores in order to demonstrate the SeReCon-based PR, e.g. 32-bit Adder, 32-bit Multiplier, 128-AES Cipher and 128-bit AES Decipher. The SeReCon IP core is a CPU-based system which is implemented using
the Xilinx EDK software. SeReCon uses an embedded 32-bit MicroBlaze processor which is operating at 125 MHz. Analysis of the SeReCon FPGA resource usage is included. The RC system prototype implementation issues and suggested future work also are highlighted. The VHDL sources for each of implemented (non-proprietary) IP cores are included in the thesis DVD.

The thesis also reports the SeReCon demonstrator which includes demonstrator application, the RC system communication library and SeReCon API. The SeReCon demonstrator is implemented in Python and executed on the Intel server. Demonstrator application results confirm feasibility of the SeReCon-based secure reconfiguration in largest Xilinx Virtex-6 FPGAs (e.g. XC6VLX760). The thesis provides detailed insight into the operation of the prototype RC system during the SeReCon (and EIDR) initialisation, IP core installation and activation.

The demonstrator application shows that even genuine IP cores, when developed in multi-party environment, could include implicit communication channels and could introduce security risks.

This thesis also describes the SDR device prototype and proposes how the SeReCon element can be included within the SDR RC system. Modifications to the SeReCon implementation required to integrate SeReCon within the prototype SDR device are also highlighted.

8.7. Future Research Directions

Future work could investigate risks of implicit communication channel exploiting design clock signals as the communication medium. The motivating example included in this thesis suggests that further research is required in this area, and that further research on RC systems security could focus on SeReCon optimisation and FDAT functional extension.

Future work could optimise the performance and robustness of the SeReCon IP core. The following work packages are suggested:

- ERDB structure optimisation, e.g. generalisation of FPGA routing types and the use of device-specific data field sizes (e.g. bit fields) could be considered in order to reduce SeReCon firmware footprint.
  The ERDB Analyser does not support the detection of IP core internal design errors, e.g. short circuits between signals. Future work could improve the ERDB Analyser robustness by including this functionality.
  The ERDB Verifier uses a single static and a predefined reconfigurable region in order to support the EAPR design flow. In future work an extension of the IP core verification is suggested in order to incorporate a model of multiple cores within the reconfigurable region.
Chapter 8 – Conclusions

- External memory encryption. Further work could investigate the use of external RAM encryption as a complementary approach to the use of scarce FPGA internal memory (BRAM) resources for the SeReCon firmware storage.
- The prototype SeReCon implementation does not support data authentication, e.g. authenticated-encryption. Only the Cipher Block Chaining encryption-mode is supported. A solution could be to include an additional authentication field in the IP core entry in the SafeLock data structures. Practical analysis of trade-offs between the encryption methods used and RC system reconfiguration time could also be investigated.
- The TRNG macrocell architecture could be optimised towards the architecture of Virtex-5 (or Virtex-6) CLB.
- The software ECC implementation could be updated in order to include the hardware accelerator for large number ECC operations. This would improve SeReCon performance during the DH shared key calculation. SeReCon could also support longer ECC keys.
- The EIDR could be implemented using an external FPGA board which is connected to the RC system configuration interface (e.g. JTAG) in order to detect BaseSFC tampering.
- A more robust PCIe interface could be implemented in order to support multi-word (burst) read/write PCIe transactions.
- A new, open-source CPU architecture is suggested for future SeReCon implementations. Also, the SeReCon architecture and firmware is not hardened against power analysis attacks which could expose the SeReCon security credentials, e.g. the EIDR contents or the SafeLock encryption keys. Thus, future work could also support SeReCon firmware and architecture modification in order to include DPA-preventing hardware primitives (e.g. WDDL logic) and algorithms (branch balancing etc).

Future work could also focus on the FDAT framework extension. The following work packages are suggested:

- The proposed FDAT framework offers increased productivity in low-level design analysis by seamlessly extending the Xilinx FPGA design flow. Similar tools could be developed for other FPGA fabrics, i.e. Altera, Actel etc.
- Future FDAT releases could provide support for industry standard open design netlist formats (e.g. EDIF) and standard XML-based IP core descriptions (e.g. IP-XACT).
- Additional FDAT modules could also be developed in order to support FDAT communication and debug of RC systems, e.g. FDAT JTAG module etc. Also, research on FDAT modules supporting RC system PR and debugging, e.g. by providing write access to design XDL netlist or implementing various phases of the FPGA design flow (design placers & routers, JBITS-like functionality etc) could increase RC systems design productivity.
References


References


References


References


References


References


References


References


References


References


# Appendix A – List of Reconfigurable Computing architectures

This appendix lists various technologies which are used in RC systems.

- 3P plus 1 Technology
- Achronix Semiconductor Corp
- Ambric
- AsceniumCorp
- Aspex
- ChipWrights
- Clearspeed
- Coherent Logix
- Connex
- Context Corporation
- Cradle
- Element CXI
- IP Flex
- IceraSemiconductor Ltd
- IkoaCorporation
- IntellasysCorporation
- M2000
- MathStar
- Mesh Semiconductor
- Morphotech
- PACT
- PicochipDesigns
- Plurarity
- Rapport
- Raytheon Monarch
- ReCore
- Sandbridge
- Silicon Hive
- Spiral Gateway
- Stream Processors
- Stretch
- Systemonic
- Tabula
- Tilera
- Videantis
- VivaceSemiconductor
- XMOS Semiconductor
- Xelerated

This appendix provides the reference EIDR source code in VHDL.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

description: This file contains the reference implementation of the Extended ID Register (EIDR).

-- Company: BIRC Group, NUI Galway, Ireland
-- Author: Krzysztof Kepa
-- Copyright: Krzysztof Kepa (2009). All rights reserved.
-- Copying, using and modification without author's written permission PROHIBITED.
-- Create Date: 14:53:05 11/18/2009
-- Design Name: IDR
-- Module Name: EIDR reference design - Behavioral
-- Project Name: Secure Reconfiguration Controller (SeReCon)
-- Target Devices: Virtex-5 LXT
-- Tool versions: 9.2.04i_PR11
-- Description: This file contains the reference implementation of the Extended ID Register (EIDR).
-- Revision: Revision 1.00 - Reference implementation
----------------------------------------------------------------------------------
variable result : STD_LOGIC := '1';
begin
for i in input_vector'low to input_vector'high loop
    if '0' /= input_vector(i) then
        result := '0';
    end if;
end loop;
return result;
end is_zero;

begin
    EQUAL_EN_SIGNAL:
    equal_en <= '1' when (mac = kr_data and '0'=is_zero(kr_data)) else '0';

    AUTH_CFG_OK_SIGNAL:
    auth_cfg_ok <= '1' when done='1' and equal_en = '1' else '0';

    CREDENTIAL_ENABLE:
    credentials <= cr_data when auth_cfg_ok='1' else (others=>'0');

    LTC_ENABLE:
    lifetime <= lifetime_cnt when auth_cfg_ok='1' else (others=>'0');

    AAC_ENABLE:
    device_restarts <= device_restarts_cnt when auth_cfg_ok='1' else (others=>'0');

    FRC_ENABLE:
    uptime <= uptime_cnt;

    AUTH_UPDATE_SIGNAL:
    auth_update <= auth_cfg_ok and update;

    MNC_ENABLE:
    msg_no <= msg_no_cnt when auth_cfg_ok='1' else (others=>'0');

KR: process (clk, rst) --KEY_REGISTER
begin
    if rst='1' then
        kr_data <= (others => '0');
    elsif (clk'event and clk='1') then
        if wr = '1' then
            kr_data <= mac;
        end if;
    end if;
end process;

CR: process (clk, rst) --CREDENTIAL_REGISTER
begin
    if rst='1' then
        cr_data <= (others => '0');
    elsif (clk'event and clk='1') then
        if wr = '1' then
            cr_data <= data;
        end if;
    end if;
end process;

AAC: process (clk, rst) --AUTHENTICATED_CONFIGURATIONS_COUNTER
begin
    if rst='1' then
        device_restarts_cnt <= (others => '0');
    elsif clk='1' and clk'event then
        if device_restarts_cnt_en = '1' then
            device_restarts_cnt <= device_restarts_cnt + 1;
        end if;
    end if;
end process;

LTC: process (clk, rst) --LIFETIME_COUNTER
begin
if rst='1' then
  lifetime_cnt <= (others => '0');
elsif clk='1' and clk'event then
  if auth_cfg_ok='1' then
    lifetime_cnt <= lifetime_cnt + 1;
  end if;
end if;
end process;

MNC: process (clk, rst) --MSG_NO_COUNTER
begin
if rst='1' then
  msg_no_cnt <= (others => '0');
elsif clk='1' and clk'event then
  if auth_update='1' then
    msg_no_cnt <= msg_no_cnt + 1;
  end if;
end if;
end process;

FRC: process (clk, rst, done) --FREE_RUNNING_COUNTER
begin
if (rst='1' or done='0') then
  uptime_cnt <= (others => '0');
elsif clk='1' and clk'event then
  uptime_cnt <= uptime_cnt + 1;
end if;
end process;

ONE_SHOT_SYNC_PROC: process (clk)
begin
if (clk'event and clk = '1') then
  if (rst = '1') then
    state <= FSM_IDLE;
    device_restarts_cnt_en <= '0';
  else
    state <= next_state;
    device_restarts_cnt_en <= auth_cfg_ok_en_i;
  end if;
end if;
end process;

ONE_SHOT_NEXT_STATE_DECODE: process (state, auth_cfg_ok)
begin
next_state <= state;
case (state) is
  when FSM_IDLE =>
    if auth_cfg_ok = '1' then
      next_state <= FSM_INIT;
    end if;
  when others =>
    next_state <= FSM_ACTIVATED;
end case;
end process;

ONE_SHOT_OUTPUT_DECODE: process (state)
begin
if state = FSM_INIT then
  auth_cfg_ok_en_i <= '1';
else
  auth_cfg_ok_en_i <= '0';
end if;
end process;
end Behavioral;
Appendix C – ERDB C Data Structures (erdb.h)

This appendix provides the ERDB header file (erdb.h) which defines ERDB data structures.

```c
/*
** Author: Krzysztof Kepa
* Company: BIRC Group, NUI Galway, Ireland
* Copyright: Krzysztof Kepa (2010). All rights reserved. Copying, using and
* implementation without author's written permission PROHIBITED.
**/

#ifndef __ERDB_H__
#define __ERDB_H__

typedef struct {
    unsigned char x;
    unsigned char y;
    unsigned char x_next;
    unsigned char y_next;
    unsigned char x_offset;
    unsigned char y_offset;
    unsigned char x_times;
    unsigned char y_times;
} tilegroup_entry_t;

typedef struct {
    unsigned int len;
    tilegroup_entry_t* tilegroup_entry;
} tilegroup_t;

typedef struct {
    int x;
    int y;
} shape.tap_t;

typedef struct {
    unsigned int len;
    shape.tap_t* tap;
} shape_t;

typedef struct {
    unsigned char mna; //cfg mna
    unsigned char word;//cfg word offset (or number in some cases!)
    unsigned int word_mask;//actual word mask
} cfg_data_t;

typedef struct {
    unsigned int wire1_index; //offset in WIRE_NAME_DB
    unsigned char dir_index;  //offset in PIP_DIR_DB
    unsigned int wire2_index; //offset in WIRE_NAME_DB
    shape_t* tap;
    unsigned char tap_cnt;
} cfg_data_t;
```

---

- 199 -
typedef struct {
    unsigned int pips_cnt; // number of pips in the tilegroup
    pip_entry_t* pips; // pip entries
    tilegroup_t* tilegroup; // related tilegroup
} tile_type_entry_t;

typedef struct {
    unsigned int tilegroups_cnt; // number of tiletype grups
    unsigned char tiletype_index; // offset in TILE_TYPE_DB
    tile_type_entry_t* tilegroups; // related tiletype group entry
} tile_type_t;

typedef struct {
    shape_tap_t tap;
    shape_t* shape;
    tilegroup_t* tilegroup;
} wire_entry_t;

typedef struct {
    unsigned int wire_index; // wire name
    unsigned int tilegroup_cnt;
    wire_entry_t* tilegroups;
} wire_tg_entry_t;

typedef struct {
    unsigned int wire_cnt; // number of wires in tile type
    unsigned char tiletype_index; // offset in TILE_TYPE_DB
    wire_tg_entry_t* wires; // related wire entry
} wire_tile_type_t;

extern const unsigned int WS_DB_LEN;  // shape_t array
extern const unsigned int TG_DB_LEN;  // tilegroup_t array
extern const unsigned int PIP_DIR_DB_LEN; // char* array
extern const unsigned int WIRE_NAME_DB_LEN; // char* array
extern const unsigned int WIRE_MODE_DB_LEN; // char* array
extern const unsigned int TILE_TYPE_DB_LEN; // tile_type_t array
extern const unsigned int PD_DB_LEN; // tile_type_t array
extern const unsigned int ROUTING_DB_LEN; // wire_tile_type_t array
extern const unsigned int LAYOUT_DB_LEN; // unsigned char* array

extern shape_t WS_DB[];
extern tilegroup_t TG_DB[];
extern unsigned char* PIP_DIR_DB[];
extern unsigned char* WIRE_NAME_DB[];
extern unsigned char* WIRE_MODE_DB[];
extern unsigned char* TILE_TYPE_DB[];
extern tile_type_t PD_DB[];
extern wire_tile_type_t ROUTING_DB[];
extern unsigned char* LAYOUT_DB[];

#include "erdb.h"
Appendix D – “BitPipsVerificator“ recipe report

This appendix provides the detailed report of results obtained after "BitPipsVerificator” recipe execution, e.g. the list of XDL pips which are not found in the bitstream and the list of additional bitstream pips which are not found in the reference XDL file.

D.1. List of XDL pips which are not found in the bitstream

1. Found 15 tile types with missing pips:
   2. CLBL tile type (2 pips):
      3. L_COUT -> M_COUT_N (24 tiles):
      4. L_COUT -> L_COUT_N (18 tiles):

2. CLBML tile type (2 pips):
   3. M_COUT -> M_COUT_N (27 tiles):

3. CLK_BUFGMUX tile type (11 pips):
   4. CLK_BUFGMUX_CLKINT1_12 -> CLK_BUFGMUX_PREMUX0_CLK12 (1 tile):
   5. CLK_BUFGMUX_MIXED_IN_CLKB_P10 -> CLK_BUFGMUX_PREMUX0_CLK5 (1 tile):
   6. CLK_BUFGMUX_MIXED_IN_CLKB_P12 -> CLK_BUFGMUX_PREMUX0_CLK6 (1 tile):
   7. CLK_BUFGMUX_MIXED_IN_CLKB_P14 -> CLK_BUFGMUX_PREMUX0_CLK7 (1 tile):
   8. CLK_BUFGMUX_MIXED_IN_CLKB_P16 -> CLK_BUFGMUX_PREMUX0_CLK8 (1 tile):
   9. CLK_BUFGMUX_MIXED_IN_CLKB_P18 -> CLK_BUFGMUX_PREMUX0_CLK9 (1 tile):
   10. CLK_BUFGMUX_MIXED_IN_CLKB_P2 -> CLK_BUFGMUX_PREMUX0_CLK1 (1 tile):
   11. CLK_BUFGMUX_MIXED_IN_CLKB_P10 -> CLK_BUFGMUX_PREMUX0_CLK10 (1 tile):
   12. CLK_BUFGMUX_MIXED_IN_CLKB_P30 -> CLK_BUFGMUX_PREMUX0_CLK15 (1 tile):
   13. CLK_BUFGMUX_MIXED_IN_CLKB_P6 -> CLK_BUFGMUX_PREMUX0_CLK3 (1 tile):
   14. CLK_BUFGMUX_MIXED_IN_CLKB_P8 -> CLK_BUFGMUX_PREMUX0_CLK4 (1 tile):

4. CLK_CMT_BOT tile type (12 pips):
   5. CLK_CMT_CMT_CLK_00 -> CLK_IOB_MUXED_CLKOUT18 (1 tile):
   6. CLK_CMT_CMT_CLK_01 -> CLK_IOB_MUXED_CLKOUT14 (1 tile):
   7. CLK_CMT_CMT_CLK_12 -> CLK_IOB_MUXED_CLKOUT6 (1 tile):
   8. CLK_CMT_CMT_CLK_13 -> CLK_IOB_MUXED_CLKOUT8 (1 tile):
   9. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
   10. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT12 (1 tile):
   11. CLK_CMT_CMT_CLK_22 -> CLK_IOB_MUXED_CLKOUT16 (1 tile):
   12. CLK_CMT_CMT_CLK_25 -> CLK_IOB_MUXED_CLKOUT20 (1 tile):
   13. CLK_CMT_CMT_CLK_00 -> CLK_IOB_MUXED_CLKOUT18 (1 tile):
   14. CLK_CMT_CMT_CLK_01 -> CLK_IOB_MUXED_CLKOUT14 (1 tile):
   15. CLK_CMT_CMT_CLK_12 -> CLK_IOB_MUXED_CLKOUT6 (1 tile):
   16. CLK_CMT_CMT_CLK_13 -> CLK_IOB_MUXED_CLKOUT8 (1 tile):
   17. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
   18. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT12 (1 tile):
   19. CLK_CMT_CMT_CLK_22 -> CLK_IOB_MUXED_CLKOUT16 (1 tile):
   20. CLK_CMT_CMT_CLK_25 -> CLK_IOB_MUXED_CLKOUT20 (1 tile):
   21. CLK_CMT_CMT_CLK_00 -> CLK_IOB_MUXED_CLKOUT18 (1 tile):
   22. CLK_CMT_CMT_CLK_01 -> CLK_IOB_MUXED_CLKOUT14 (1 tile):
   23. CLK_CMT_CMT_CLK_12 -> CLK_IOB_MUXED_CLKOUT6 (1 tile):
   24. CLK_CMT_CMT_CLK_13 -> CLK_IOB_MUXED_CLKOUT8 (1 tile):
   25. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
   26. CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT12 (1 tile):
   27. CLK_CMT_CMT_CLK_22 -> CLK_IOB_MUXED_CLKOUT16 (1 tile):
   28. CLK_CMT_CMT_CLK_25 -> CLK_IOB_MUXED_CLKOUT20 (1 tile):
   29. CLK_IOB_MUXED_CLKIN10 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
   30. CLK_IOB_MUXED_CLKIN20 -> CLK_IOB_MUXED_CLKOUT20 (1 tile):
   31. CLK_IOB_MUXED_CLKIN6 -> CLK_IOB_MUXED_CLKOUT6 (1 tile):
   32. CLK_IOB_MUXED_CLKIN8 -> CLK_IOB_MUXED_CLKOUT8 (2 tiles):

37. CLK_HROW tile type (19 pips):
   38. CLK_HROW_CLK_MTA9_4 -> CLK_HROW_CLK_H_MTA9_4 (1 tile):
   39. CLK_HROW_CLK_MTA9_6 -> CLK_HROW_CLK_H_MTA9_6 (1 tile):
   40. CLK_HROW_CLK_BUFI -> CLK_HROW_HOLD_PIN0 (1 tile):
   41. CLK_HROW_CLK_BUFO10 -> CLK_HROW_HOLD_PIN0 (1 tile):
   42. CLK_HROW_CLK_BUFI10 -> CLK_HROW_HOLD_PIN0 (1 tile):
Appendix D – “BitPipsVerificator “ recipe report

CLK_HROW_GCLK_BUF12 -> CLK_HROW_HCLKL_P3 (1 tile):
CLK_HROW_GCLK_BUF12 -> CLK_HROW_HCLKL_P4 (1 tile):
CLK_HROW_GCLK_BUF12 -> CLK_HROW_HCLKR_P2 (2 tiles):
CLK_HROW_GCLK_BUF3 -> CLK_HROW_HCLKR_P3 (2 tiles):
CLK_HROW_GCLK_BUF4 -> CLK_HROW_HCLKL_P1 (1 tile):
CLK_HROW_GCLK_BUF4 -> CLK_HROW_HCLKL_P2 (2 tiles):
CLK_HROW_GCLK_BUF5 -> CLK_HROW_HCLKL_P6 (1 tile):
CLK_HROW_GCLK_BUF7 -> CLK_HROW_HCLKL_P2 (1 tile):
CLK_HROW_GCLK_BUF7 -> CLK_HROW_HCLKR_P3 (1 tile):
CLK_HROW_GCLK_BUF8 -> CLK_HROW_HCLKL_P5 (2 tiles):
CLK_HROW_GCLK_BUF9 -> CLK_HROW_HCLKL_P1 (1 tile):
CLK_HROW_GCLK_BUF9 -> CLK_HROW_HCLKR_P0 (2 tiles):
CLK_HROW_MGT_CLK_P2 -> CLK_HROW_MGT_CLKV2 (1 tile):
CLK_IOB_B tile type (6 pips):
  CLK_IOB_B_CLK_BUF12 -> CLK_IOB_MUXED_CLKOUT2 (1 tile):
CLK_IOB_CLK_BUF2 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
CLK_IOB_CLK_BUF4 -> CLK_IOB_MUXED_CLKOUT4 (1 tile):
CLK_IOB_CLK_BUF5 -> CLK_IOB_MUXED_CLKOUT5 (1 tile):
CLK_IOB_MUXED_CLKIN18 -> CLK_IOB_MUXED_CLKOUT18 (1 tile):
CLK_IOB_MUXED_CLKIN20 -> CLK_IOB_MUXED_CLKOUT20 (1 tile):
CMT_BOT tile type (5 pips):
  CMT_BUFG0 -> CMT_DCM_0_CLKFB (1 tile):
  CMT_BUFG0 -> CMT_PLL_CLKIN1 (1 tile):
  CMT_BUFG4 -> CMT_DCM_0_CLKIN (1 tile):
  CMT_BUFG4 -> CMT_DCM_1_CLKIN (1 tile):
  CMT_GIOB5 -> CMT_DCM_1_CLKFB (1 tile):
CMT_TOP tile type (2 pips):
  CMT_BUFG6 TOP -> CMT_DCM_1_CLKFB (1 tile):
  CMT_GIOB4 -> CMT_DCM_1_CLKIN (1 tile):
GT3 tile type (4 pips):
  GT3_CLK_B_0_10 -> GT3_RXUSRCLK0 (1 tile):
  GT3_CLK_B_0_11 -> GT3_TXUSRCLK0 (1 tile):
  GT3_CLK_B_1_8 -> GT3_TXUSRCLK1 (1 tile):
  GT3_CLK_B_1_9 -> GT3_RXUSRCLK1 (1 tile):
HCLK_IOI tile type (11 pips):
  HCLK_IOI_BUFIO_OUT0 -> HCLK_IOI_IDCLKP0 (2 tiles):
  HCLK_IOI_BUFIO_OUT1 -> HCLK_IOI_IDCLKP1 (3 tiles):
  HCLK_IOI_BUFIO_OUT2 -> HCLK_IOI_IDCLKP2 (2 tiles):
  HCLK_IOI_BUFIO_OUT3 -> HCLK_IOI_IDCLKP3 (1 tile):
  HCLK_IOI_G_HCLK_P0 -> HCLK_IOI_LEAF_GCLK_P0 (4 tiles):
  HCLK_IOI_G_HCLK_P1 -> HCLK_IOI_LEAF_GCLK_P1 (2 tiles):
  HCLK_IOI_G_HCLK_P2 -> HCLK_IOI_LEAF_GCLK_P2 (2 tiles):
  HCLK_IOI_G_HCLK_P3 -> HCLK_IOI_LEAF_GCLK_P3 (2 tiles):
  HCLK_IOI_G_HCLK_P5 -> HCLK_IOI_LEAF_GCLK_P5 (2 tiles):
  HCLK_IOI_LEAF_GCLK_P2 -> HCLK_IOI_REFLCLK (1 tile):
  HCLK_IOI_LEAF_GCLK_P5 -> HCLK_IOI_REFLCLK (2 tiles):
HCLK_IOI_BOTCEN tile type (1 pips):
  HCLK_IOI_G_HCLK_P0 -> HCLK_IOI_LEAF_GCLK_P0 (1 tile):

– 202 –
Appendix D – “BitPipsVerificator “ recipe report

99  HCLK_IOI_CMT tile type (1 pips):
100  HCLK_IOI_G_HCLK_P0 -> HCLK_IOI_LEAF_GCLK_P0 (1 tile):
101
102  HCLK_IOI_TOPCEN tile type (1 pips):
103  HCLK_IOI_G_HCLK_P0 -> HCLK_IOI_LEAF_GCLK_P0 (1 tile):
104
105  INT tile type (6 pips):
106    LH0 = LH18 (131 tiles):
107    LV0 = LH0 (23 tiles):
108    LV0 = LH18 (44 tiles):
109    LV0 = LV18 (242 tiles):
110    LV18 = LH0 (66 tiles):
111    LV18 = LH18 (38 tiles):
112
113  IOI tile type (12 pips):
114    IOI_IOCLKP0 -> IOI_ICLK_0 (9 tiles):
115    IOI_IOCLKP0 -> IOI_ICLK_1 (7 tiles):
116    IOI_IOCLKP0 -> IOI_ICLK_1 (7 tiles):
117    IOI_IOCLKP0 -> IOI_ICLK_1 (7 tiles):
118    IOI_LEAF_GCLK_P2 -> IOI_OCLKP_0 (15 tiles):
119    IOI_LEAF_GCLK_P2 -> IOI_OCLKP_1 (12 tiles):
120    IOI_LEAF_GCLK_P3 -> IOI_ICLK_0 (7 tiles):
121    IOI_LEAF_GCLK_P3 -> IOI_ICLK_1 (9 tiles):
122
123
124
125
126 Found 837 missing pips (95 tiles of 15 types)
127
D.2. List of additional bitstream pips which are not found in the reference XDL file

1 Found 11 tile types with extra pips:

2

3 CLBLL tile type (2 pips):
4   LL_COUT -> M_COUT_N (21 tiles):
5   L_COUT -> L_COUT_N (18 tiles):

6

7 CLBLM tile type (2 pips):
8   L_COUT -> L_COUT_N (25 tiles):
9   M_COUT -> M_COUT_N (27 tiles):

10

11 CLK_CMT_BOT tile type (8 pips):
12   CLK_CMT_CMT_CLK_00 -> CLK_IOB_MUXED_CLKOUT14 (1 tile):
13   CLK_CMT_CMT_CLK_01 -> CLK_IOB_MUXED_CLKOUT10 (1 tile):
14   CLK_CMT_CMT_CLK_12 -> CLK_IOB_MUXED_CLKOUT2 (2 tiles):
15   CLK_CMT_CMT_CLK_13 -> CLK_IOB_MUXED_CLKOUT4 (2 tiles):
16   CLK_CMT_CMT_CLK_18 -> CLK_IOB_MUXED_CLKOUT8 (1 tile):
17   CLK_IOB_MUXED_CLKIN12 -> CLK_IOB_MUXED_CLKOUT12 (1 tile):
18   CLK_IOB_MUXED_CLKIN2 -> CLK_IOB_MUXED_CLKOUT2 (1 tile):
19   CLK_IOB_MUXED_CLKIN4 -> CLK_IOB_MUXED_CLKOUT4 (1 tile):

20

21 CLK_HROW tile type (4 pips):
22   CLK_HROW_GCLK_BUF28 -> CLK_HROW_HCLKL_P3 (1 tile):
23   CLK_HROW_GCLK_BUF29 -> CLK_HROW_HCLKL_P3 (1 tile):
24   CLK_HROW_GCLK_BUF29 -> CLK_HROW_HCLKL_P6 (1 tile):
25   CLK_HROW_GCLK_BUF29 -> CLK_HROW_HCLKR_P6 (1 tile):

26

27 CLK_IOB_B tile type (3 pips):
28   CLK_IOB_CLK_BUF2 -> CLK_IOB_MUXED_CLKOUT26 (1 tile):
29   CLK_IOB_MUXED_CLKIN2 -> CLK_IOB_MUXED_CLKOUT2 (1 tile):
30   CLK_IOB_MUXED_CLKIN4 -> CLK_IOB_MUXED_CLKOUT4 (1 tile):

31

32 GT3 tile type (1 pips):
33   GT3_CLKPN -> GT3_CLKOUT_NORTH_N (1 tile):

34

35 HCLK_I0I tile type (6 pips):
36   HCLK_I0I_G_HCLK_P3 -> HCLK_I0I_LEAF_GCLK_P3 (1 tile):
37   HCLK_I0I_G_HCLK_P4 -> HCLK_I0I_LEAF_GCLK_P4 (1 tile):
38   HCLK_I0I_I2ICLK_P9 -> HCLK_I0I_BUFI0 (1 tile):
39   HCLK_I0I_LEAF_GCLK_P5 -> HCLK_I0I_REFCLK (1 tile):
40   HCLK_I0I_LEAF_GCLK_P9 -> HCLK_I0I_REFCLK (1 tile):
41   HCLK_I0I_MGT_CLK_P2 -> HCLK_I0I_BUFI0 (2 tiles):

42

43 HCLK_I0I_BOTCEN tile type (1 pips):
44   HCLK_I0I_LEAF_GCLK_P0 -> HCLK_I0I_REFCLK (1 tile):

45

46 HCLK_I0I_CMT tile type (1 pips):
47   HCLK_I0I_G_HCLK_P4 -> HCLK_I0I_LEAF_GCLK_P4 (1 tile):
Appendix D – “BitPipsVerificator “ recipe report

INT tile type (6 pips):

LH0 = LV0 (57 tiles):
LH0 = LV18 (55 tiles):
LH18 = LH0 (151 tile):
LH18 = LV0 (117 tiles):
LH18 = LV18 (78 tiles):
LV18 = LV0 (240 tiles):

IOI tile type (44 pips):

IOI_IMUX_B11 = IOI_ICLKP_0 (67 tiles):
IOI_IMUX_B11 = IOI_ICLKP_1 (67 tiles):
IOI_IMUX_B5 = IOI_ICLKP_0 (67 tiles):
IOI_ICLKP0 = IOI_OCLKP_1 (67 tiles):
IOI_ICLKP1 = IOI_OCLKP_0 (3 tiles):
IOI_ICLKP1 = IOI_OCLKP_1 (5 tiles):
IOI_IOCLKP2 = IOI_ICLKP_0 (16 tiles):
IOI_IOCLKP2 = IOI_ICLKP_1 (17 tiles):
IOI_IOCLKP3 = IOI_ICLKP_0 (67 tiles):
IOI_IOCLKP3 = IOI_ICLKP_1 (67 tiles):
IOI_LEAF_GCLK_P1 = IOI_ICLKP_0 (16 tiles):
IOI_LEAF_GCLK_P1 = IOI_ICLKP_1 (17 tiles):
IOI_LEAF_GCLK_P2 = IOI_ICLKP_0 (16 tiles):
IOI_LEAF_GCLK_P2 = IOI_ICLKP_1 (17 tiles):
IOI_LEAF_GCLK_P4 = IOI_ICLKP_0 (16 tiles):
IOI_LEAF_GCLK_P4 = IOI_ICLKP_1 (17 tiles):
IOI_I}$,

Found 2621 extra pips (78 tiles of 11 types)
Appendix E – SeReCon debug console output during RC system demonstration

This appendix illustrates the complete output of the SeReCon debug console during SeReCon-enabled RC system demonstration.

The SeReCon debug output report is divided into four sections. Section E.1 illustrates the default view of the SeReCon debug console. SeReCon debug messages which are generated during EIDR initialisation are reported in Section E.2. Section E.3 illustrates installation of demonstrator IP cores, e.g. Adder (‘add’), blank design (‘blank’), Multiplier (‘mul’), AES encoder (‘enc’) and AES decoder (‘dec’). Activation of demonstrator IP cores is highlighted in Section 0.

E.1. Default view of the SeReCon debug console

This section illustrates the default view of the SeReCon debug console.

```
1 Initializing SysAce...OK
2 MFS: Initializing MFS ... DONE
3 MFS: 131072 KB MFS occupied @ 0x98000000
4 MFS: Used 1 block(s) out of 252288
5 MFS: Current dirname is "/"
6 Initializing ICAP...OK
7 Restore IDR
8 Reading 68 bytes from "serecon.idr" (CF) to RAM (@ 0x901B8FB4)...OK
9 68 bytes read.
10 Restore system state.
11 - load system state data (serecon.sys)...ERROR (get file size)
12
13 ===============================
14 ==     NUI SeReCon Demo     ==
15 ==   Jan 28 2010 22:41:34   ==
16 ===============================
17
18 [1]  Receive commands from PCIe
19 [2]  MFS reset
20 [3]  MFS file remove
21 [4]  MFS stats
22 [5]  MFS file copy
23 [6]  MFS file rename
24 [7]  MFS list dir
25 [8]  MFS cat file
26 [9]  MFS cat file HEX
27 [10] CF list directory
29 [12] CF remove dir
30 [13] CF create dir
31 [14] CF change dir
32 [15] CF cat file content
33 [16] Read FPGA registers
```
Appendix E – SeReCon debug console output during RC system demonstration

34  [17] Initialise IDR
35  [18] Display IDR content
36  [19] IDR content backup to CF
37  [20] IDR content restore from CF
38  [21] IDR content backup to MFS
39  [22] IDR content restore from MFS
40  [23] Read IPV demo.ipv
41  [24] Read IPV demo.m00
42  [25] Generate shared key for IPV demo
43  [26] Restore pubkey in CF
44  [27] Demo install IP core
45  [28] Demo activate IP core
46  [29] Generate default system state
47  [30] Allow non-safe IP cores
48  [31] Display system state
49  [32] Reset safelock
50
51 Choice? (1-32)1
52 Listening to PCIe interface...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

E.2. EIDR initialisation

This section illustrates SeReCon debug messages which are generated during EIDR initialisation.

1  Init IDR.
2  Generating new data...OK
3
4  Generating ECC-P192 key-pair...
5     - initialise curve...OK
6     - initialise generating point...OK
7     - Collect random data...OK
8     - calculate public key...OK
9     - encrypt private key...OK
10    - store encrypted private key to "serecon.prv"...OK
11    - store public key to "serecon.pub"...Writing 276 bytes (from 0x901BEAC8) to K
12    OK
13    Done.
14    Waiting for ACK...OK
15    Waiting for NULL...OK
16    PCIe command finished.
17
18    Waiting for PCIe command...(press ESC key to break)
19
20    PCIe: file download.
21    Reading 'serecon.pub' file from CF card to RAM...OK
22    Uploading file from RAM...OK
23    Send file 'serecon.pub' (276 bytes).
24    File transfer completed.
25    Waiting for ACK...OK
26    Waiting for NULL...OK
27    PCIe command finished.
28
29    Waiting for PCIe command...(press ESC key to break)
30
31    PCIe: file upload.
32    Downloading file 'ipv_demo.ipv' to RAM...OK
33    Received file 'ipv_demo.ipv' (252 bytes).
34    Writing 252 bytes (from 0x901BE578) to MFS("ipv_demo.ipv")...OK
35    OK
36    File transfer completed.
37    Waiting for ACK...OK
38    Waiting for NULL...OK
39    PCIe command finished.
40
41    Waiting for PCIe command...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

E.3. IP cores installation

This section provides SeReCon debug messages illustrating installation of demonstrator IP cores, e.g. Adder (‘add’), blank design (‘blank’), Multiplier (‘mul’), AES encoder (‘enc’) and AES decoder (‘dec’).

E.3.1. The “add” IP core installation

This section illustrates installation of the Adder IP core (‘add’).

1. PCIe: get idr msg_no.
2. Sending msg_no...DONE
3. Waiting for ACK...OK
4. Waiting for NULL...OK
5. PCIe command finished.
6. Waiting for PCIe command...(press ESC key to break)
7. PCIe: file upload.
8. Downloading file 'ipv_demo.m00' to RAM...OK
9. Received file 'ipv_demo.m00' (108 bytes).
10. Writing 108 bytes (from 0x901BE578) to MFS("ipv_demo.m00")...OK
11. OK
12. File transfer completed.
13. Waiting for ACK...OK
14. Waiting for NULL...OK
15. PCIe command finished.
16. Waiting for PCIe command...(press ESC key to break)
17. Generate session key for ipv_demo IPV
18. - get IDR msg_no...OK (0x00)
19. - read IPV public key file "ipv_demo.ipv"...OK
20. - read IPV message file "ipv_demo.m00"...OK
21. - compare msg_no...OK
22. - get random data...OK
23. - get IDR credentials...OK
24. - get access to private key "serecon.prv"...OK
25. - calculate shared key...OK
26. Shared_key: 0x5F4FCB40C11DDEEA8351230556DF7D4F298E1A6C2B11369
27. - sign reply msg...OK
28. - save reply msg to "ipv_demo.r01"...OK
29. - save shared key to "ipv_demo.s01"...OK
30. Done.
31. Waiting for ACK...OK
32. Waiting for NULL...OK
33. PCIe command finished.
34. Waiting for PCIe command...(press ESC key to break)
35. PCIe: get idr msg_no.
Appendix E – SeReCon debug console output during RC system demonstration

42 Sending msg_no...DONE
43 Waiting for ACK...OK
44 Waiting for NULL...OK
45 PCIe command finished.
46
47 Waiting for PCIe command...(press ESC key to break)
48 PCIe: file download.
50 Reading file 'ipv_demo.r01' from MFS to RAM...OK
51 Uploading file from RAM...OK
52 Send file 'ipv_demo.r01' (80 bytes).
53 File transfer completed.
54 Waiting for ACK...OK
55 Waiting for NULL...OK
56 PCIe command finished.
57
58 Waiting for PCIe command...(press ESC key to break)
59 PCIe: file upload.
61 Downloading file 'add.e01'to RAM...OK
62 Received file 'add.e01' (145824 bytes).
63 Writing 145824 bytes (from 0x901BFB08) to MFS("add.e01")...OK
64 OK
65 File transfer completed.
66 Waiting for ACK...OK
67 Waiting for NULL...OK
68 PCIe command finished.
69
70 Waiting for PCIe command...(press ESC key to break)
71 PCIe: file upload.
73 Downloading file 'add.c01'to RAM...OK
74 Received file 'add.c01' (32 bytes).
75 Writing 32 bytes (from 0x901BE578) to MFS("add.c01")...OK
76 OK
77 File transfer completed.
78 Waiting for ACK...OK
79 Waiting for NULL...OK
80 PCIe command finished.
81
82 Waiting for PCIe command...(press ESC key to break)
83 PCIe: Install IP core.
85
86 Install 'add' core (provided by 'ipv_demo'):
87 - get IDR credentials...OK
88 - get IPV shared key from "ipv_demo.s01" file...OK
89 - get IP core license from "add.c01"...OK
90 - get safelock entry for msg 01...OK
91 - install IP core license in "add.l01"...OK
92 - get IP core from "add.e01"...OK
93 - call ERDB analyser
94
95 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
96
97 - read header
Appendix E – SeReCon debug console output during RC system demonstration

98
99  Header data:
100  - HeaderLength : 86
101  - BitstreamLength : 145732
102  - DesignName : prm_adder_partial.ncd
103  - PartName : 5vlx50tff1136
104  - Date : 2009/10/27
105  - Time : 19: 7:46
106
107  - parse bitstream...OK
108    Region (x,y): (10,0) to (42,43)
109  - detect external pips...OK
110    Found 2699 external pips (2254 real, 445 fake).
111  - analyse isolation boundary...OK
112    Region (x,y): (10,0) to (42,45)
113  - detect io pips...OK
114    Found 2468 io pips (2154 real, 314 fake).
115  Analysis finished successfully.
116  - update safelock credentials...OK
117  - create installed IP file "add.i01"...OK
118  - create IP analysis report file "add.d01"...OK
119  Installed IP core 'add'.
120
121  Waiting for ACK...OK
122  Waiting for NULL...OK
123  PCIe command finished.
124
125  Waiting for PCIe command...(press ESC key to break)
E.3.2. The “blank” IP core installation

This section illustrates installation of the blank design IP core (‘blank’).

1. PCIe: get idr msg_no.
2. Sending msg_no...DONE
3. Waiting for ACK...OK
4. Waiting for NULL...OK
5. PCIe command finished.
6. Waiting for PCIe command...(press ESC key to break)
7. PCIe: file upload.
8. Downloading file 'ipv_demo.m02'to RAM...OK
9. Received file 'ipv_demo.m02' (108 bytes).
10. Writing 108 bytes (from 0x901BE578) to MFS("ipv_demo.m02")...OK
11. OK
12. File transfer completed.
13. Waiting for ACK...OK
14. Waiting for NULL...OK
15. PCIe command finished.
16. Waiting for PCIe command...(press ESC key to break)
17. Generate session key for ipv_demo IPV
18. - get IDR msg_no...OK (0x02)
19. - read IPV public key file "ipv_demo.ipv"...OK
20. - read IPV message file "ipv_demo.m02"...OK
21. - compare msg_no...OK
22. - get random data...OK
23. - get IDR credentials...OK
24. - get access to private key "serecon.prv"...OK
25. - calculate shared key...OK
26. Shared_key: 0x35C3495E63F3C4AFC11DAE993600D96B31A668C950F5C28E
27. - sign reply msg...OK
28. - save reply msg to "ipv_demo.r03"...OK
29. - save shared key to "ipv_demo.s03"...OK
30. Done.
31. Waiting for ACK...OK
32. Waiting for NULL...OK
33. PCIe command finished.
34. Waiting for PCIe command...(press ESC key to break)
35. PCIe: get idr msg_no.
36. Sending msg_no...DONE
37. Waiting for ACK...OK
38. Waiting for NULL...OK
39. PCIe command finished.
40. Waiting for PCIe command...(press ESC key to break)
PCIe: file download.
Reading file 'ipv_demo.r03' from MFS to RAM...OK
Uploading file from RAM...OK
Send file 'ipv_demo.r03' (80 bytes).
File transfer completed.
Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

PCIe: file upload.
Downloading file 'blank.e03' to RAM...OK
Received file 'blank.e03' (126512 bytes).
Writing 126512 bytes (from 0x901BFB08) to MFS("blank.e03")...OK
OK
File transfer completed.
Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

PCIe: file upload.
Downloading file 'blank.e03' to RAM...OK
Received file 'blank.e03' (32 bytes).
Writing 32 bytes (from 0x9023F3D0) to MFS("blank.e03")...OK
OK
File transfer completed.
Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

PCIe: Install IP core.
Install 'blank' core (provided by 'ipv_demo'):
- get IDR credentials...OK
- get IPV shared key from "ipv_demo.s03" file...OK
- get IP core license from "blank.c03"...OK
- get safelock entry for msg 03...OK
- install IP core license in "blank.l03"...OK
- get IP core from "blank.e03"...OK
- call ERDB analyser

ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
- read header
Header data:
- HeaderLength : 85
- BitstreamLength : 126420
- DesignName : pblock_prm_blank.ncd
- PartName : 5vlx50tff1136
Appendix E – SeReCon debug console output during RC system demonstration

- Date : 2009/10/27
- Time : 19:36:42

- parse bitstream...OK
- Region (x,y): (10,0) to (42,43)
- detect external pips...OK
- Found 2684 external pips (2239 real, 445 fake).
- analyse isolation boundary...OK
- Region (x,y): (10,0) to (42,45)
- detect io pips...OK
- Found 2453 io pips (2139 real, 314 fake).

Analysis finished successfully.
- update safelock credentials...OK
- create installed IP file "blank.i03"...OK
- create IP analysis report file "blank.d03"...OK
- Installed IP core 'blank'.

Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)
E.3.3. The “mul” IP core installation

This section illustrates installation of the Multiplier IP core (‘mul’).

1. PCIe: get idr msg_no.
2. Sending msg_no...DONE
3. Waiting for ACK...OK
4. Waiting for NULL...OK
5. PCIe command finished.
6. Waiting for PCIe command...(press ESC key to break)
7. PCIe: file upload.
8. Downloading file 'ipv_demo.m04' to RAM...OK
9. Received file 'ipv_demo.m04' [108 bytes].
10. Writing 108 bytes (from 0x902152E0) to MFS("ipv_demo.m04")...OK
11. OK
12. File transfer completed.
13. Waiting for ACK...OK
14. Waiting for NULL...OK
15. PCIe command finished.
16. Waiting for PCIe command...(press ESC key to break)
17. Generate session key for ipv_demo IPV
18. - get IDR msg_no...OK [0x04]
19. - read IPV public key file "ipv_demo.ipv"...OK
20. - read IPV message file "ipv_demo.m04"...OK
21. - compare msg_no...OK
22. - get random data...OK
23. - get IDR credentials...OK
24. - get access to private key "serecon.prv"...OK
25. - calculate shared key...OK
26. Shared_key: 0xDB5DBA0AED57E124B5D9730676471F4DD8D0115FA7E3E542
27. - sign reply msg...OK
28. - save reply msg to "ipv_demo.r05"...OK
29. - save shared key to "ipv_demo.s05"...OK
30. Done.
31. Waiting for ACK...OK
32. Waiting for NULL...OK
33. PCIe command finished.
34. Waiting for PCIe command...(press ESC key to break)
35. PCIe: get idr msg_no.
36. Sending msg_no...DONE
37. Waiting for ACK...OK
38. Waiting for NULL...OK
39. PCIe command finished.
40. Waiting for PCIe command...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

PCIe: file download.
52 Reading file 'ipv_demo.r05' from MFS to RAM...OK
53 Uploading file from RAM...OK
54 Send file 'ipv_demo.r05' (80 bytes).
55 File transfer completed.
56 Waiting for ACK...OK
57 Waiting for NULL...OK
58 PCIe command finished.

59 PCIe: file upload.
60 Reading file 'ipv_demo.r05' from MFS to RAM...OK
61 Uploading file from RAM...OK
62 Send file 'ipv_demo.r05' (80 bytes).
63 File transfer completed.
64 PCIe command finished.

65 PCIe: file upload.
66 Uploading file 'ipv_demo.r05' from RAM...OK
67 Reading file 'ipv_demo.r05' from MFS...OK
68 File transfer completed.
69 PCIe command finished.
70 PCIe: file upload.
71 Uploading file 'ipv_demo.r05' from RAM...OK
72 Reading file 'ipv_demo.r05' from MFS...OK
73 File transfer completed.
74 PCIe command finished.

75 PCIe: file upload.
76 Uploading file 'ipv_demo.r05' from RAM...OK
77 Reading file 'ipv_demo.r05' from MFS...OK
78 File transfer completed.
79 PCIe command finished.

79 PCIe: file upload.
80 Uploading file 'ipv_demo.r05' from RAM...OK
81 Reading file 'ipv_demo.r05' from MFS...OK
82 File transfer completed.
83 PCIe command finished.

84 PCIe: file upload.
85 Uploading file 'ipv_demo.r05' from RAM...OK
86 Reading file 'ipv_demo.r05' from MFS...OK
87 File transfer completed.
88 PCIe command finished.

89 PCIe: Install IP core.
90 Install 'mul' core (provided by 'ipv_demo'):
91 - get IDR credentials...OK
92 - get IPV shared key from 'ipv_demo.s05' file...OK
93 - get IP core license from 'mul.c05'...OK
94 - get safelock entry for msg 05...OK
95 - install IP core license in 'mul.c05'...OK
96 - call ERDB analyser
97 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
98 Header data:
99 - HeaderLength : 91
100 - BitstreamLength : 150968
101 - DesignName : prm_multiplier_partial.ncd
102 - PartName : 5vlx50tff1136
Appendix E – SeReCon debug console output during RC system demonstration

- Date : 2009/10/27
- Time : 19:14:0

- parse bitstream...OK
  Region (x,y): (10,0) to (42,43)
- detect external pips...OK
  Found 2709 external pips (2264 real, 445 fake).
- analyse isolation boundary...OK
  Region (x,y): (10,0) to (42,45)
- detect io pips...OK
  Found 2478 io pips (2164 real, 314 fake).
- Analysis finished successfully.
- update safelock credentials...OK
- create installed IP file "mul.i05"...OK
- create IP analysis report file "mul.d05"...OK
- Installed IP core 'mul'.

- Waiting for ACK...OK
- Waiting for NULL...OK
- PCIe command finished.

- Waiting for PCIe command...(press ESC key to break)
E.3.4. The “enc” IP core installation

This section illustrates installation of the AES Encoder IP core (‘enc’).

1. PCIe: get idr msg_no.
2. Sending msg_no...DONE
3. Waiting for ACK...OK
4. Waiting for NULL...OK
5. PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

PCle: file upload.

10. Downloading file 'ipv_demo.m06'to RAM...OK
11. Received file 'ipv_demo.m06' (108 bytes).
12. Writing 108 bytes (from 0x901E40E0) to MFS("ipv_demo.m06")...OK
13. OK
14. File transfer completed.
15. Waiting for ACK...OK
16. Waiting for NULL...OK
17. PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

Generate session key for ipv_demo IPV

- get IDR msg_no...OK (0x06)
- read IPV public key file "ipv_demo.ipv"...OK
- read IPV message file "ipv_demo.m06"...OK
- compare msg_no...OK
- get random data...OK
- get IDR credentials...OK
- get access to private key "serecon.prv"...OK
- calculate shared key...OK
  - Shared_key: 0x4FC8376B29BBAE05E1C1695725B6348B4A83B7019FC90
- sign reply msg...OK
- save reply msg to "ipv_demo.r07"...OK
- save shared key to "ipv_demo.s07"...OK

Done.

Waiting for ACK...OK
38. Waiting for NULL...OK
39. PCIe command finished.

Waiting for PCIe command...(press ESC key to break)

PCle: get idr msg_no.

44. Sending msg_no...DONE
45. Waiting for ACK...OK
46. Waiting for NULL...OK
47. PCIe command finished.

Waiting for PCIe command...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

51 PCIe: file download.
52 Reading file 'ipv_demo.r07' from MFS to RAM...OK
53 Uploading file from RAM...OK
54 Send file 'ipv_demo.r07' (80 bytes).
55 File transfer completed.
56 Waiting for ACK...OK
57 Waiting for NULL...OK
58 PCIe command finished.
59
60 Waiting for PCIe command...(press ESC key to break)
61 PCIe: file upload.
62 Downloading file 'enc.e07'to RAM...OK
63 Received file 'enc.e07' (165792 bytes).
64 Writing 165792 bytes (from 0x902D6708) to MFS("enc.e07")...OK
65 OK
66 File transfer completed.
67 Waiting for ACK...OK
68 Waiting for NULL...OK
69 PCIe command finished.
70
71 Waiting for PCIe command...(press ESC key to break)
72 PCIe: file upload.
73 Downloading file 'enc.c07'to RAM...OK
74 Received file 'enc.c07' (32 bytes).
75 Writing 32 bytes (from 0x90276E70) to MFS("enc.c07")...OK
76 OK
77 File transfer completed.
78 Waiting for ACK...OK
79 Waiting for NULL...OK
80 PCIe command finished.
81
82 Waiting for PCIe command...(press ESC key to break)
83 PCIe: Install IP core.
84
85 Install 'enc' core (provided by 'ipv_demo'):
86 - get IDR credentials...OK
87 - get IPV shared key from "ipv_demo.s07" file...OK
88 - get IP core license from "enc.c07"...OK
89 - get safelock entry for msg 07...OK
90 - install IP core license in "enc.l07"...OK
91 - get IP core from "enc.e07"...OK
92 - call ERDB analyser
93 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
94 - read header
95
96 Header data:
97 - HeaderLength : 88
98 - BitstreamLength : 165696
99 - DesignName : prm_encoder_partial.ncd
100 - PartName : 5vlix30tff1136
Appendix E – SeReCon debug console output during RC system demonstration

- Date : 2009/10/27
- Time : 19:20:24

parse bitstream...OK
Region (x,y): (10,0) to (42,43)
detect external pips...OK
Found 3330 external pips (2885 real, 445 fake).
analyse isolation boundary...OK
Region (x,y): (10,0) to (42,45)
detect io pips...OK
Found 3064 io pips (2750 real, 314 fake).
Analysis finished successfully.
update safelock credentials...OK
create installed IP file "enc.i07"...OK
create IP analysis report file "enc.d07"...OK
Installed IP core 'enc'.

Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)
E.3.5. The “dec” IP core installation

This section illustrates installation of the AES Decoder IP core (‘dec’).

1. PCIe: get idr msg_no.
2. Sending msg_no...DONE
3. Waiting for ACK...OK
4. Waiting for NULL...OK
5. PCIe command finished.
6. Waiting for PCIe command...(press ESC key to break)
7. PCIe: file upload.
8. Downloading file 'ipv_demo.m08' to RAM...OK
9. Received file 'ipv_demo.m08' (108 bytes).
10. Writing 108 bytes (from 0x90203108) to MFS("ipv_demo.m08")...OK
11. OK
12. File transfer completed.
13. Waiting for ACK...OK
14. Waiting for NULL...OK
15. PCIe command finished.
16. Waiting for PCIe command...(press ESC key to break)
17. Generate session key for ipv_demo IPV
18. - get IDR msg_no...OK (0x08)
19. - read IPV public key file "ipv_demo.ipv"...OK
20. - read IPV message file "ipv_demo.m08"...OK
21. - compare msg_no...OK
22. - get random data...OK
23. - get IDR credentials...OK
24. - get access to private key "serecon.prv"...OK
25. - calculate shared key...OK
26. Shared_key: 0x70337D681E6EAA099C7D0E5015935178103DE12F6CDE3BE0
27. - sign reply msg...OK
28. - save reply msg to "ipv_demo.r09"...OK
29. - save shared key to "ipv_demo.s09"...OK
30. Done.
31. Waiting for ACK...OK
32. Waiting for NULL...OK
33. PCIe command finished.
34. Waiting for PCIe command...(press ESC key to break)
35. PCIe: get idr msg_no.
36. Sending msg_no...DONE
37. Waiting for ACK...OK
38. Waiting for NULL...OK
39. PCIe command finished.
40. Waiting for PCIe command...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

51 PCIe: file download.
52 Reading file 'ipv_demo.r09' from MFS to RAM...OK
53 Uploading file from RAM...OK
54 Send file 'ipv_demo.r09' (80 bytes).
55 File transfer completed.
56 Waiting for ACK...OK
57 Waiting for NULL...OK
58 PCIe command finished.
59 Waiting for PCIe command...(press ESC key to break)
60 PCIe: file upload.
61 Downloading file 'dec.e09'to RAM...OK
62 Received file 'dec.e09' (168672 bytes).
63 Writing 168672 bytes (from 0x90325010) to MFS("dec.e09")...OK
64 OK
65 File transfer completed.
66 Waiting for ACK...OK
67 Waiting for NULL...OK
68 PCIe command finished.
69 Waiting for PCIe command...(press ESC key to break)
70 PCIe: file upload.
71 Downloading file 'dec.c09'to RAM...OK
72 Received file 'dec.c09' (32 bytes).
73 Writing 32 bytes (from 0x901E6358) to MFS("dec.c09")...OK
74 OK
75 File transfer completed.
76 Waiting for ACK...OK
77 Waiting for NULL...OK
78 PCIe command finished.
79 Waiting for PCIe command...(press ESC key to break)
80 PCIe: Install IP core.
81 Install 'dec' core (provided by 'ipv_demo'):
82 - get IDR credentials...OK
83 - get IPV shared key from "ipv_demo.s09" file...OK
84 - get IP core license from "dec.c09"...OK
85 - get safelock entry for msg 09...OK
86 - install IP core license in "dec.l09"...OK
87 - get IP core from "dec.e09"...OK
88 - call ERDB analyser
89 ERDB_analyser v.1.0 (release Jan 29 2010, 18:40:42)
90 - read header
91 Header data:
92 - HeaderLength : 88
93 - BitstreamLength : 168576
94 - DesignName : prm_decoder_partial.ncd
95 - PartName : 5vlx30tff1136
Appendix E – SeReCon debug console output during RC system demonstration

- Date : 2009/10/27
- Time : 19:27:4

- parse bitstream...OK
- Region (x,y): (10,0) to (42,43)
- detect external pips...OK
- Found 3869 external pips (3424 real, 445 fake).
- analyse isolation boundary...OK
- Region (x,y): (10,0) to (42,45)
- detect io pips...OK
- Found 3594 io pips (3280 real, 314 fake).

Analysis finished successfully.

- update safelock credentials...OK
- create installed IP file "dec.i09"...OK
- create IP analysis report file "dec.d09"...OK
- Installed IP core 'dec'.

Waiting for ACK...OK
Waiting for NULL...OK
PCIe command finished.

Waiting for PCIe command...(press ESC key to break)
E.4. IP cores activation

This section provides SeReCon debug messages illustrating activation of demonstrator IP cores, e.g. Adder (`add`), blank design (`blank`), Multiplier (`mul`), AES encoder (`enc`) and AES decoder (`dec`).

E.4.1. The “add” IP core activation

This section illustrates installation of the Adder IP core (`add`).

1. PCIe: Activate IP core.

4. Activate 'add' core (msg 01):
   - get safelock credentials...OK
   - check IP core license file...OK
   - update safelock credentials...OK
   - call ERDB verifier...

9. ERDB verifier v.1.0 (release Jan 29 2010, 18:40:42)

12. - get IP core report 'add.d01'...OK
13. - make diff with system state...OK

19. Verification report:
20. - IP core configuration region: (10,0 - 42,43)...OK
21. - IP core isolation region : (10,0 - 42,45) ...OK
22. - external pips found : 2699 (2254 real, 445 fake)
23. - unmatched ext pips : 15 (15 real, 0 fake)
24. - io pips found : 2468 (2154 real, 314 fake)
25. - unmatched io pips : 15 (15 real, 0 fake)

27. !!!WARNING!!!
29. Potentially dangerous IP core!!!
30. - write report file 'add.t01'...OK

Verification finished without errors.
31. IP core 'add' violates security requirements for reconfigurable region.
32. - check security bypass flag...OFF
33. IP core activation cancelled due to security risk.
34. - rollback license update...OK
36. Waiting for ACK...OK
37. Waiting for NULL...OK
38. PCIe command finished.
39. Waiting for PCIe command...(press ESC key to break)
E.4.2. The “blank” IP core activation

This section illustrates activation of the blank design IP core (‘blank’).

1. PCIe: Activate IP core.

2. Activate 'blank' core (msg 03):
   - get safelock credentials...OK
   - check IP core license file...OK
   - update safelock credentials...OK
   - call ERDB verifier...

3. ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)

4. - get IP core report 'blank.d03'...OK
5. - make diff with system state...OK

6. Verification report:
   - IP core configuration region: (10,0 - 42,43)...OK
   - IP core isolation region    : (10,0 - 42,45) ...OK
   - external pips found        : 2684 (2239 real, 445 fake)
   - unmatched ext pips         : 0 (0 real, 0 fake)
   - io pips found              : 2453 (2139 real, 314 fake)
   - unmatched io pips          : 0 (0 real, 0 fake)

7. Verification finished without errors.

8. IP core 'blank' is safe.

9. - get IP core from "blank.i03" file...OK
10. - check bitstream header...OK
11. - disable BM interface...OK
12. - load IP core to ICAP...OK
13. - update system state...OK

14. IP core 'blank.i03' (msg 03) activated.

15. - enable BM interface...OK

16. Waiting for ACK...OK
17. Waiting for NULL...OK
18. PCIe command finished.
19. Waiting for PCIe command...(press ESC key to break)
E.4.3. The “mul” IP core activation

This section illustrates activation of the Multiplier IP core (‘mul’).

1 PCIe: Activate IP core.
2
3 Activate ‘mul’ core (msg 05):
4 - get safelock credentials...OK
5 - check IP core license file...OK
6 - update safelock credentials...OK
7 - call ERDB verifier...
8
9 ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
10
11 - get IP core report 'mul.d05'...OK
12 - make diff with system state...OK
13
14 Verification report:
15
16 - IP core configuration region: (10,0 - 42,43)...OK
17 - IP core isolation region: (10,0 - 42,45) ...OK
18 - external pips found: 2709 (2264 real, 445 fake)
19 - unmatched ext pips: 25 (25 real, 0 fake)
20 - io pips found: 2478 (2164 real, 314 fake)
21 - unmatched io pips: 25 (25 real, 0 fake)
22
23 !!!WARNING!!!
24
25 Potentially dangerous IP core!!!
26
27 - write report file 'mul.t05'...OK
28 Verification finished without errors.
29 - IP core 'mul' violates security requirements for reconfigurable region.
30 - check security bypass flag...OFF
31 - roll back license update...OK
32 Waiting for ACK...OK
33 Waiting for NULL...OK
34 PCIe command finished.
35
36 Waiting for PCIe command...(press ESC key to break)
E.4.4. The “enc” IP core activation

This section illustrates activation of the AES Encoder IP core (‘enc’).

1 PCIe: Activate IP core.
2
3 Activate ’enc’ core (msg 07):
4 - get safelock credentials...OK
5 - check IP core license file...OK
6 - update safelock credentials...OK
7 - call ERDB verifier...
8
9 ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
10
11 - get IP core report 'enc.d07'...OK
12 - make diff with system state...OK
13
14 Verification report:
15
16 - IP core configuration region: (10,0 - 42,43)...OK
17 - IP core isolation region : (10,0 - 42,45) ...OK
18 - external pips found : 3330 (2885 real, 445 fake)
19 - unmatched ext pips : 646 (646 real, 0 fake)
20 - io pips found : 3064 (2750 real, 314 fake)
21 - unmatched io pips : 611 (611 real, 0 fake)
22
23
24 !!!WARNING!!!
25
26 Potentially dangerous IP core!!!
27
28 - write report file 'enc.t07'...OK
29 Verification finished without errors.
30 IP core 'enc' violates security requirements for reconfigurable region.
31 - check security bypass flag...OFF
32 IP core activation cancelled due to security risk.
33 - roll back license update...OK
34 Waiting for ACK...OK
35 Waiting for NULL...OK
36 PCIe command finished.
37
38 Waiting for PCIe command...(press ESC key to break)
Appendix E – SeReCon debug console output during RC system demonstration

E.4.5. The “dec” IP core activation

This section illustrates activation of the AES Decoder IP core (‘dec’).

1. PCIe: Activate IP core.
2. Activate 'dec' core (msg 09):
3. - get safelock credentials...OK
4. - check IP core license file...OK
5. - update safelock credentials...OK
6. - call ERDB verifier...
7. ERDB_verifier v.1.0 (release Jan 29 2010, 18:40:42)
8. - get IP core report 'dec.d09'...OK
9. - make diff with system state...OK
10. Verification report:
11. - IP core configuration region: (10,0 - 42,43)...OK
12. - IP core isolation region : (10,0 - 42,45) ...OK
13. - external pips found : 3869 (3424 real, 445 fake)
14. - unmatched ext pips : 1185 (1185 real, 0 fake)
15. - io pips found : 3594 (3280 real, 314 fake)
16. - unmatched io pips : 1141 (1141 real, 0 fake)
17. !!!WARNING!!!
18. Potentially dangerous IP core!!!
19. - write report file 'dec.t09'...OK
20. Verification finished without errors.
21. IP core 'dec' violates security requirements for reconfigurable region.
22. - check security bypass flag...OFF
23. IP core activation cancelled due to security risk.
24. - roll back license update...OK
25. Waiting for ACK...OK
26. Waiting for NULL...OK
27. PCIe command finished.
28. Waiting for PCIe command...(press ESC key to break)
29. Received break signal.
30. Exiting.