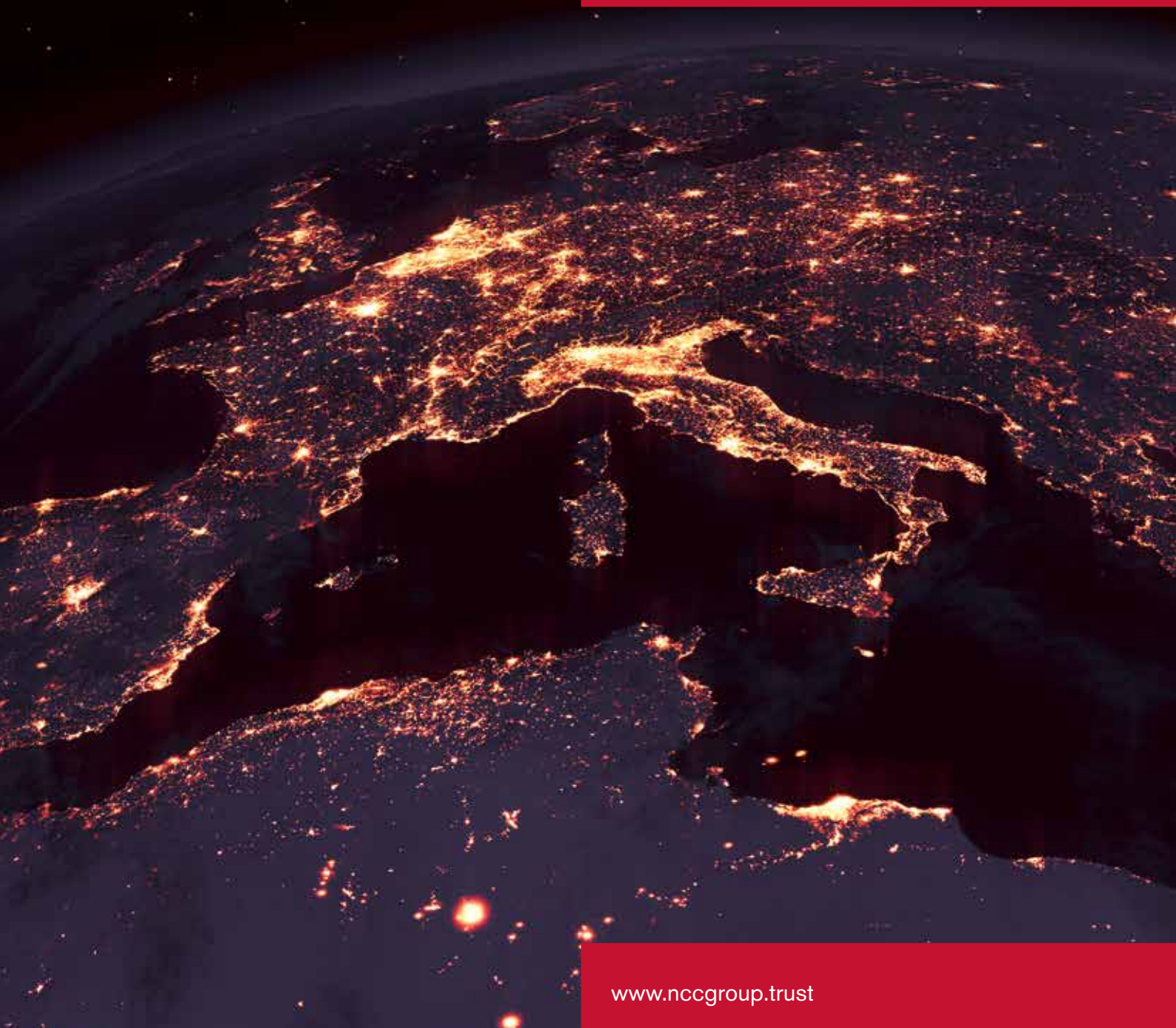
nccgroup
freedom from doubt

RESEARCH INSIGHTS

Hardware Design: FPGA Security Risks



www.nccgroup.trust

CONTENTS

All Rights Reserved. © NCC Group 2015

AUTHOR

DUNCAN HURWOOD

Duncan is a senior consultant at NCC Group, specialising in telecom, embedded systems and application review. He has over 18 years' experience within the telecom and security industry performing almost every role within the software development cycle from design and development to integration and product release testing.

A dedicated security assessor since 2010, his consultancy experience includes multiple technologies, languages and platforms from web and mobile applications, to consumer devices and high-end telecom hardware.

All Rights Reserved. © NCC Group 2015

GLOSSARY

AES	Advanced encryption standard, a cryptography cipher
ASIC	Application-specific integrated circuit, non-programmable hardware logic chip
Bitfile	Binary instruction file used to program FPGAs
CLB	Configurable logic block, an internal part of an FPGA
CPLD	Complex programmable logic device
EEPROM	Electronically erasable programmable read-only memory
eFuse	A system to allow one-time writing of data
Flash	Non-volatile memory
FPGA	Field programmable gate array
HDL	Hardware description language, such as Verilog or VHDL
JTAG	Standard test access port on a device (IEEE 1149.1 – 1990)
NVRAM	Non-volatile RAM, retaining value after power loss
OTP	One time programmable, allowing write once only
PCB	Printed circuit board
PLA	Programmable logic array, forerunner of FPGA technology
PUF	Physically unclonable function
POWF	Physical one-way function
PSoC	Programmable system on chip, an FPGA and other hardware on a single chip
SoC	System on chip, a non-programmable logic chip with additional hardware
SRAM	Static RAM, volatile memory storage
SystemVerilog	HDL programming language, extension of Verilog from 2002 onwards
Verilog	HDL programming language, developed in the 1980s and 1990s, with C-like syntax
VHDL	HDL programming language, developed from the 1980s, based on ADA

All Rights Reserved. © NCC Group 2015

INTRODUCTION

This document provides an introduction to the use of FPGAs. FPGA stands for field-programmable gate array. An FPGA is a logic device whose function can be changed while the device is in place within its working environment, allowing the hardware processing of a system to be altered by an external configuration loading process. Configuration bitstreams for FPGAs can be used to change the logical processing of input data, and so alter the functionality of a device.

The most common uses of FPGAs are areas within automotive, medical, factory equipment and defence equipment that required rapid concurrent processing.

The paper examines the process of developing configuration binaries for FPGA devices and the potential security problems that could be encountered. It assumes no prior knowledge of FPGA technology.

The information in this paper is useful for anyone who works with embedded devices, software and hardware developers or producers who may want to understand the potential security risks of using FPGAs within their devices.

All Rights Reserved. © NCC Group 2015

FPGA History

The direct antecedents of FGPAs were the programmable logic array (PLA) chips of the late 1970s which allowed the programming, though usually not the reprogramming, of a set of AND and OR gates into the equivalent of a state machine. The development of the technology encompassed several strands, including the concurrent, but initially quite distinct, development of programmable array logic (PAL) chips, which themselves diversified into complex programmable logic devices (CPLDs).

Top-of-the range modern FPGA chips have kept pace with processor evolution and can be far more effective compared to microprocessors.

FPGA devices initially differed from CPLD through the use of on-board static random-access memory (SRAM). This volatile storage system is used as a short-cut, providing the result of logic operations as data in a look-up table. Multiple data operations are combined in a series of complex logic blocks (CLB). The results of the CLB operations are routed via switch blocks to chip outputs or as inputs to additional CLB operations. Therefore the complexity of the processing within FPGAs is limited by the number of logical operations on a chip, usually determined by the semiconductor technology, measured since the late 1980s in nanometre size.

Chip complexity has expanded both by increased density of logical operation, approximately following Moore's Law since 1965, and physically, as additional layers can be added horizontally to chip design. The result is that top-of-the range modern FPGA chips have kept pace with processor evolution and can be far more effective compared to microprocessors when reliable concurrent or low latency operations are required.

The key step in the development of modern FPGAs was the invention of reliable non-volatile electronically erasable programmable read-only memory (EEPROM), and its derivative flash memory, in the early 1980s. The use of NAND flash memory provided a reliable and quick storage system for FPGA configuration files and made it feasible to load behavioural logic onto FPGA chips during system start-up.

With the separation of logical behaviour storage from the chip carrying out the operations, the ability to alter the programming of the device in the field became feasible. Consequently the appearance of non-volatile RAM (NVRAM) in proximity to FPGA chips in order to store the FPGA configuration file is expected.

As FPGAs have evolved, the simplicity of the design has been complicated by the development of programmable system-on-chip (PSoC) families, where processors, non-volatile memory, timing sources, and connecting buses are all present alongside the FPGA. The configuration file for the FPGA can be loaded directly onto these chips, without the need for an external NVRAM device. Many PSoC devices have pre-configured libraries of available functions and allow quicker initial development, but using such devices incurs the cost of reduced flexibility.

As FPGAs have evolved, the simplicity of the design has been complicated by the development of programmable system-on-chip (PSoC) families.

ASICs are related to FPGAs and often mentioned alongside them. An ASIC (application-specific integrated circuit) can perform the same function as an FPGA, but it is not reprogrammable. Instead, the circuit design is fixed during manufacture.

All Rights Reserved. © NCC Group 2015

Because of this, ASICs may share logical security flaws with FPGA devices, but not issues related to reprogramming. ASICs often require a great deal of investment to design correctly (using FPGAs as prototypes within that process) but will run at a much lower power and are therefore suitable for mass-production use within smaller devices.

An ASIC (application-specific integrated circuit) can perform the same function as an FPGA, but it is not reprogrammable.

Note that the use of ASICs requires that a design has been exhaustively proven to function correctly, and therefore the resources needed to develop them are significantly greater. The cost of each iteration of ASIC design can be very large, depending on the number of layers within the chip, with each layer requiring a separate mask to specify the connections on the chip precisely. This means that ASICs are only economical when used within devices with a large volume of sales.

Use of FPGAs

As FPGAs are logic processing units there is no set application for which they are used. Unlike a CPU or microprocessor, which processes instructions through a sequential list, an FPGA will process instructions in parallel and therefore will be capable of handling multiple concurrent inputs simultaneously. Uses for this property include signal processing, particularly image processing, industrial automation, and aerospace or defence systems. Within aerospace applications the problem of radiation causing the slow degradation of devices through ionisation or sudden changes of behaviour through single particle collision events has long been recognised.

Initial solutions were to use one-time programmable (OTP) FPGAs without on board SDRAM, but subsequently several companies, such as Microsemi [1] and Xilinx [2], developed radiation-tolerant FPGAs, specifically for use in non-terrestrial or vulnerable locations.

In each case FPGAs will be used where multiple simultaneous inputs must be dealt with inside a limited time or where a strictly defined latency must be adhered to. While this need can be addressed with a microcontroller running a real-time operating system, the advantages gained by parallel processing of signals can produce timing and precision advantages for FPGAs.

FPGAs have also found increasing use within four recent growth areas:

- **Automotive**, where increasing numbers of monitor points require simultaneous processing under extremely time-limited conditions. As vehicles continue to increase the areas in which autonomous controls can back up or supplant human decisions, vehicle data processing must take account of all inputs at the same time. A sequential review of data may not produce the correct result in the short window of opportunity.
- **Medical**, where the electronic devices within the armamentarium are often not produced in the numbers required to make ASIC development feasible. Medical applications using FPGAs are particularly focused on image processing.
- **Data communication**, both in data centres and where wired communication requires rapid processing of simultaneous signals.
- **Cryptography**, especially during the Bitcoin mania, which made extensive use of FPGA parallelism, though several solutions migrated to the use of ASICs to speed up the processing once the design was fixed.

[1] See http://www.microsemi.com/document-portal/doc_view/131374-radiation-tolerant-proasic3-fpgas-radiation-effects-report
[2] See http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf

All Rights Reserved. © NCC Group 2015

The use of FPGAs and ASICs tends to follow a cyclical pattern, with the lower individual cost of ASICs resulting in their increasing use as products mature and reach a viable sales volume. However, as products develop, ASICs may again be replaced by FPGAs to allow new technologies to be introduced, restarting the cycle.

FPGAs tend to be used within larger embedded systems rather than within the current Internet of Things proliferation of domestic products, though they may be found in prosumer items such as higher-level digital cameras.

However, as products develop, ASICs may again be replaced by FPGAs to allow new technologies to be introduced.

As FPGAs are found within a system rather than forming the main part, the security around their use can often be overlooked.

Recent FPGA Developments

Currently two companies, Altera [3] and Xilinx [4], are believed to account for 85% of all FGPA sales.

Xilinx was considered to be the leading supplier of the two, although Intel [5] has recently acquired Altera, with effects which are yet to be fully seen.

Xilinx produce several families of FPGA chips, each of which have been designed in multiple versions of increasing cost and complexity. These range from the SPARTAN chips, the low-end of which is often encountered in training boards, to the cost-focused ARTIX, the performance-focused VIRTEX, designed for fast traffic processing, and the balanced KINTEX chips recommended for digital signal processing.

All of the chips come in a variety of capabilities, from 45nm to ultra-high performance 16nm-based chips. Xilinx separate out their SoC offerings into the ZYNQ family, which combine ARM processors with the FPGA; the most recent of these is the ZYNQ UltraScale+ 16nm SoC device.

Altera produce four families of FPGAs, each of which contains a set of FPGAs developed between 2002 and 2015. The Cyclone chips are focused on cost, with the Stratix family the equivalent of Xilinx VIRTEX in its performance focus. The Arria family balance between the two, while the recently introduced Max chip contains an ARM processor, dual flash banks, timing, a power regulator and RAM.

Other companies that produce FPGAs include Lattice Semiconductor [6], whose FPGAs are often used in telecommunications devices, Microsemi [7], and Quicklogic [8].

[3] See www.altera.com
[4] See www.xilinx.com
[5] Agreement was reached in June 2015, see http://intelacquiresaltera.transactionannouncement.com/
[6] See http://www.latticesemi.com/
[7] See http://www.microsemi.com/products/fpga-soc/fpga-and-soc
[8] See http://www.quicklogic.com/platforms/connectivity/pp3e/

All Rights Reserved. © NCC Group 2015

FPGA Development

FPGA Languages

An FPGA is a set of inputs and outputs with configurable gates between them. The position of the inputs and outputs depends upon the chip in use, but the logical behaviour to be programmed into the device should be FPGA-independent, as the logical design does not reflect physical characteristics. Because of this, the use of higher-level logical descriptions to define the behaviour is possible; this is the role of the hardware description language (HDL).

The two most common HDLs in use are Verilog and VHDL. Verilog has most recently been standardised as IEEE1364-2005 and tends to be in use more in use in the US than VHDL. VHDL stands for very high speed integrated circuit (VHSIC) hardware description language, and was most recently standardised within IEEE 1076-2008. Although first developed in the US, it tends to be used less in the US but is thought to be the preferred language within the UK and Europe.

This document will concentrate on VHDL, but note differences with Verilog where appropriate.

VHDL

VHDL is not procedural like C or Java, but a description of data flow over logical elements. The feel of the language draws on Ada, due to the development of both languages within the Department of Defense in the US.

Like other languages, VHDL code files contain libraries and packages to allow users to make use of pre-existing logic. Entities are added to these, which describe logic inputs and outputs; entities do not do anything in themselves, but describe a logical map over which architectural code can operate. The third basic element of VHDL is the architectural code, which describes the logical operations that will take place.

It is important to note that none of the above links the VHDL code to any specific hardware. For this, a constraints file is required, which describes the relationship between the physical inputs and outputs of the FPGA and the logical hardware description code. In Xilinx a constraints file can be recognised by the .ucf extension (standing for "user constraint file"), but other manufacturers use different terms, such as SDC or XDC.

Due to its ADA roots, VHDL is strongly typed. This is generally thought to make VHDL code easier to read than Verilog and may help prevent obvious coding errors. The 'self-documenting' aspect of the language, through its verbose syntax, has also led to its popularity in teaching and academic circles.

Due to its ADA roots, VHDL is strongly typed. This is generally thought to make VHDL code easier to read than Verilog and may help prevent obvious coding errors.

Verilog

Verilog was named after a concatenation of the words "verification" and "logic", and was developed initially as a simulation language in the 1980s. Its syntactical roots are in C, a language with which it shares the characteristics of weak type enforcement and sparse verbosity. These features, however, allow prototype logic to be constructed quickly and have increased the popularity of the language among the engineering community.

Some tasks are easier to perform in Verilog than in VHDL. For example, Verilog has an interface to the C language through the Verilog procedural interface (VPI), which allows users to write C code to interface directly with the simulation, a feature more directly coupled than is the case with VHDL foreign language interfaces.

All Rights Reserved. © NCC Group 2015

However, there are aspects of VHDL that are simpler than the Verilog equivalent, such as the re-use of code packages, which is simpler than the use of library modules within Verilog. In either case both languages have the capability, but the task is easier to perform in one compared to the other.

Since the mid-1990s Verilog has influenced the development of SystemVerilog, an extension of Verilog which is now generally considered a separate language. SystemVerilog has extended the language to include a greater number of C and object-oriented concepts such as records (equivalent to structs), classes, enums, and interfaces. Many of these additions give SystemVerilog access to features present in VHDL and led to its use as language for PSoC devices.

Since the mid-1990s Verilog has influenced the development of SystemVerilog, an extension of Verilog.

Process of 'Compilation'

The process of turning a conceptualised hardware design into an FPGA configuration bitstream requires a series of stages to take place. Several of these are similar to the stages used when compiling and linking a C program into processor-dependent binary; however, it is important to keep in mind that for FPGA development the end product is a logic description, not a series of steps that will be carried out atomically by a processor.

When developing VHDL the use of a development environment such as the Xilinx Vivado [9], IspLever [10] from Lattice Semiconductor or Synplify [11] will force the selection of a target FPGA and encourage the user to design with that target in mind.

The three necessary steps are:

1. Creation of a hardware description of the logic processes required using the chosen language. This is the VHDL and associated files.
2. Design synthesis, a process that converts the HDL into a general circuit schematic, not necessarily FPGA dependent, though the use of a development environment may include some aspects of this. In synthesis stage, the code syntax is checked, and it is ensured that the design is logically coherent.
3. Implementation of the design onto the target FPGA; this requires the design to be translated, placed and routed logically through the specifics of the chip it will run on. The final bitstream configuration file will be generated at this point.

The first stage above requires the author to write the HDL code, while the second tests the internal logic of the code in order to produce a coherent whole. It is only in the third stage that the capabilities of the target FPGA device are fully considered.

The output from the third stage of process would be a bitstream file, often with a suffix of .bit, that can be loaded onto the FPGA during the start-up of the board. During development the bitstream may be loaded onto the board via the JTAG interface of the FPGA, though the actual mechanism used will depend on the board and manufacturer. Once in production the bitstream is present in an adjacent or on-chip memory device and will be automatically loaded onto the FPGA, usually through a serial configuration port.

[9] See http://www.xilinx.com/products/design-tools/vivado.html
[10] See http://www.latticesemi.com/ispleverclassic
[11] See http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/ SynplifyPro.aspx

All Rights Reserved. © NCC Group 2015

FGPA Security Assessment

FPGA Assets and Threats

The assets associated with an FPGA can include:

- **The intellectual property of the developer**, such as the FPGA bitstream and configuration files. If this information were to be copied, cloned, or emulated, it could allow the device to be reproduced by a developer who had not invested resources into the development process. The bitstream key will be programmed onto non-volatile memory and should not be readable by any external process.
- **The reputation of the developer**, especially if the product behaviour can be subverted. If attackers are able to alter the behaviour of the FPGA, they may be able to make the machine behave in an unexpected manner, or allow access to functionality that requires a more expensive license.
- **The data being processed within the FPGA**. If the system processes confidential or personally identifiable information this may be obtainable from within the system. This could include the use of sensitive keys used for video and audio stream decryption, such as those used in DRM. These may be stored in volatile memory on the chip during runtime.

A review of FPGA use within a product would include the relevant assets and threats to the system.

FPGA Review

An FPGA review would not consider the FPGA in isolation from the device in which it is placed. Instead, the whole device would be examined, but particular attention would be paid to the FPGA configuration file within the device and the data outputs from the FPGA chip.

Areas considered in the review would include:

- **The assets present on the device**, in line with the risks identified above.
- **The input and outputs from the FPGA on the PCB:** How many tracks exist that are not used? Would it be possible to extract data from these connections? Are there inputs that are not normally used, and could these be activated to subvert the behaviour of the device? Do any outputs contain log files or other potentially confidential information from the FPGA? Are the tracks used to load the software onto the FPGA on the surface layer of the PCB? Is any of the output information stored on the device, or externally?
- **The hardware layout of the equipment:** This would include a review of the physical ports on the system, both accessible from the outside and present on the PCB as either a connector or adaptable connector-location. If parts of the device can be removed, or if it accepts memory cards or USB input, the effects of this will be included in the review. Is it possible to connect to the JTAG interface? How much does the device rely on physical security to prevent tampering?
- **The normal operating location of the device:** If the device is present in a public location, the opportunity for attack may be greater than if the device is kept within a secure environment. The review would also consider the possibility of insider or accidental alteration.
- **Bitfile security at rest on the device:** Is it possible for an attacker to obtain the bitfile from the NVRAM present on the device? Are bitfiles encrypted or signed, and if so how is the key to the encrypted data conveyed to the process in charge of loading the FPGA? Is a second copy of the bitstream file present elsewhere on the system? Can software be downloaded from the device, or the version of the FPGA software obtained from a console login to the system? Are files in the flash location used without checking their veracity or origin? Is more than one version of the FPGA configuration file retained on the device?
- **The upgrade process:** If the device can be upgraded, is the upgrade of the FPGA configuration also a possibility? Is the version of FPGA in use fixed, or can the chip be changed

All Rights Reserved. © NCC Group 2015

to expose exploitable behaviour? Does the process and security of a software upgrade apply also to FPGA files, or is there a separate system for FPGA files? What is the delivery mechanism for upgraded or altered files? Can local delivery of a new file supersede the normal process? Can upgrades be initiated by a user with access to the device?

- **The encryption of software during transit to the device:** Where is the decryption key stored on the device? What mechanism of encryption is used? What type of memory stores the key at different times on the system? Can the key be overwritten? Is the transport of the file encrypted at all times on its journey? Will bitfiles be stored in transit in intermediate servers? Is there a single key in use across the ecosystem, or is there a separate key per device?
- **The signing of configuration files for the device:** Is signing in use? Does the device check the signing of FPGA files before running them or upon delivery? What mechanism is used to verify the signature? Can this be subverted? What anti-cloning or watermarking techniques are in use?
- **Security within the development process:** What is the security situation at the manufacturing and repair facilities? Could the FPGA bitfiles or key leak during testing? At what point in the process is the key burned into the FPGA? Are all the mechanisms to prevent key retrieval set correctly from this point? How much general code could leak from the development process? Are schematics, datasheets, and configuration guides publicly accessible? Can casual users gain access to a relevant development environment? What is the security within the development environment location? Can information aid any reverse-engineering attacks?
- **Developmental debugging facilities and production equivalents:** How is debugging carried out during development? Is any of this accessible in released versions? Are log files created and stored during the running of the system and will these contain information useful to an attacker? Is the same key in use or accepted during development, integration testing and release? Is test bench data, recording formal test results under particular criteria, available for the development?
- **FPGA design review:** examination of the datasheets for the FPGA, configuration guide for the chip, board schematics for the PCB in use, software controlling the board, and design specifications referenced by the developers when coding the HDL.
- **For more in-depth reviews,** can any side channel attacks produce meaningful output? Side channel attacks include differential power analysis, which may be able to recover the key information, optical emissions (at a very low level) from the FPGA once the outer covering has been careful removed, optical fault injection, and timing attacks.

The above represents a starting point for the review of an FPGA implementation, but the process would depend upon the individual status of the device in questions; in particular assets on the device as described in Section 2.3.1 above and would inform the weighting of the subsequent review.

HDL Code Review

Another aspect of the FPGA review process would be an HDL code review.

An HDL code review will not limit itself to purely examining the HDL code, but will include the aspects of the external processor code that deals with FPGA configuration file maintenance, storage, and upgrade.

A code review would be conducted with reference to the external hardware rather than in isolation. Normal code and business logic review techniques would be used, but with reference to the following code considerations:

- What version of the development software is in use?
- Does the code make use of well-known libraries? Are any of these publicly available or known to be in use by the company?

All Rights Reserved. © NCC Group 2015

- Does the design implement any security-critical functions or features that need to be reviewed in the scope of the wider system?
- Have unnecessary connections been activated within the constraints file?
- What manufacturer protection mechanisms have been used, such as bitfile encryption or JTAG protection? (See section 2.2.4 below for more details.)
- How are encryption keys managed?
- Are the security mechanisms embedded within the code, or a later add-on?
- Does the code interface with other languages or verification test files? Do these files leak any restricted data?
- Does the code deal with external clock sources or work asynchronously? What mechanisms to defend against metastability have been considered? Does the device support multiple power or clock domains, and how are these managed?
- Are test bench simulations available? Are they written by the developer?
- What coding guidelines are in use?
- Is design reuse occurring? What is the implication of this for security with reference to the previous and current use of the design?
- Is there a naming convention for processes, entities, architectures, functions, and signals? Are they easy to find in the code?
- Are comments used in the code?
- What interactions are there with other coded or controlled components?
- Are test code or feedback mechanisms used? Are these removed from the finished code?

FPGA Protections

FPGA manufacturers are aware of the potential security risks when designing hardware devices, and have created a number of security mechanisms to aid developers.

Bitfile Encryption

All manufacturers now include encryption within their FPGA chips, generally using AES anti-tamper mechanisms, although these may have to be activated and used correctly by developers. The purpose of the protection is to allow bitstream files to be encrypted during transit and delivery to the FPGA itself.

Decryption will take place on the device, using a key stored within the FPGA. Clearly, if the key can be read from the device the purpose of the encryption will be defeated, so one-time write key fuses can be used to set a key within the FPGA which subsequently cannot be read or altered. Doing so is intended to limit the programmability of the device to a single manufacturer. Storage and security of the relevant keys by the manufacturer then becomes the focus of FPGA security.

All manufacturers now include encryption within their FPGA chips.

Xilinx Virtex-6 and Virtex-7 FPGAs [12] have included both encryption and HMAC authentication since at least 2010, though encryption itself was available for users many years previously. The stage at which the anti-tamper mechanisms can be introduced to the design is user-controllable. The use of anti-tamper mechanisms will consume FPGA resources and therefore may not be feasible to add to designs at a later stage.

When encryption is used the key must be stored in a secure location, which on embedded systems must be on the device itself. The key must also be programmed onto the device using a

[12] See http://www.xilinx.com/support/documentation/application_notes/xapp1239-fpga-bitstream-encryption.pdf

All Rights Reserved. © NCC Group 2015

different channel to the normal FPGA loading (to ensure the key is not removable by simple programming). This can be through JTAG programming, and the key stored on the chip in a non-volatile location, such as a one-time programmable eFuse [13].

The key must be stored in plain text (or there would need to be another key stored somewhere to decrypt it) and can be read again via JTAG from the device if the eFuse control register bits are set to allow it.

Therefore the setting of these registers is crucial to the security of the device. The control register bits can be set to disallow reading and to prevent further changes to the control bits themselves.

FPGA developers need to determine the correct balance of security and usability, especially during development. Choosing the correct time to make use of the production mechanisms will require careful planning.

FPGA developers need to determine the correct balance of security and usability.

Altera FPGAs can be configured to only allow encrypted bitfiles if programmed over the Configuration via Protocol [14] method. As with Xilinx, this key is used with AES256 and can be stored either in volatile (requiring on-board battery) or non-volatile memory.

The non-volatile key is programmed using the JTAG interface, and bit switches can be set while programming to ensure only encrypted bitstreams are accepted and to place the device in secure JTAG mode, where most JTAG instructions are ignored. Note that secure JTAG mode can be removed and is not a one-way change.

The recent Max family of FPGAs use a 128-bit, rather than 256-bit, key, and offer a unique Chip ID for each device, though this feature has to be manually activated.

Modern FPGAs from Lattice with on-chip non-volatile memory allow the use of encrypted bitfiles as the key can be permanently stored on the device [15]. The system uses 128-bit AES for encryption, and a one-time password fuse mechanism, similar to those used by other FPGA manufacturers.

If encryption is not enabled, a key code can be written into a file, which could present a risk as the file is likely to be in text format on the development systems. The iCE40 series [16] of FPGAs have dedicated non-volatile memory on chip to prevent the need for storage of the bitstreams externally for startup.

Some manufacturers include mechanisms for clearing the key from the one-time write area if tampering is suspected. The configuration and use of this mechanism must be carefully controlled, as a false positive would be liable to render the FPGA incapable of loading a configuration file.

While these features represent an advance over the security position adopted by some FPGA manufacturers during the previous fifteen years, many of the protections are voluntary and require configuration to be correctly set to assure the maximum level of protection. In addition, lower-end products will use FPGAs that do not contain the full range of currently advertised security measures.

Some manufacturers include mechanisms for clearing the key from the one-time write area if tampering is suspected.

[14] See https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_cvp.pdf
[15] See http://www.latticesemi.com/~/media/LatticeSemi/Documents/ApplicationNotes/AD/AdvancedSecurityEncryptionKeyProgrammingGuide.pdf
[16] See http://www.latticesemi.com/~/media/LatticeSemi/Documents/ApplicationNotes/IK/iCE40ProgrammingandConfiguration.pdf?document_id=46502
[13] See IBM https://www-304.ibm.com/jct03002c/press/us/en/pressrelease/7246.wss

All Rights Reserved. © NCC Group 2015

Anti-Cloning Systems

Manufacturers are concerned with the possibility that FPGA designs will be copied or cloned and used in rival systems.

Physically unclonable functions (PUF), first introduced as "physical one-way functions" in 2002, gained traction through use in smartcards due to their low cost.

The effectiveness of the function derives from the difficulty in reproducing output that was added in a semi-randomised manner in the production process and which produces a repeatable but unique output.

To reproduce the function, the exact responses of the target function must be known and then a method of producing the same responses during manufacturing must be found, which using current technology is not thought to be feasible.

Recent developments within FPGAs have involved work from Intrinsic-ID [17], and have been employed within FPGAs developed by Microsemi, while Altera has partnered with Intrinsic-ID in the development of the new Stratix chips. The Stratix 10 advertises itself as the "Best-in-Class" [18] FPGA for security, with side-channel attack protection through the use of an on-chip secure device manager.

This requires the use of a separate configuration processor within the FPGAs, with the result that while security is enhanced, the solution is not available on medium and lower end products or FPGAs without an additional on-board processor. Additional protections against cloning include embedding device numbers within chips such as Device DNA in Xilinx.

Other protection measures include JTAG disabling systems, monitoring to prevent side-channel attacks, boot code protection, and CRC checks to prevent tampering. As with the above protections, the use of correct configuration of the device at the appropriate point in the development cycle is key to its effective deployment.

Note that the vast majority of these measures are on-chip defences, and it is vital that developers consider the threat to the FPGA as part of the system as a whole, as well as on the FPGA itself.

Manufacturers are concerned with the possibility that FPGA designs will be copied or cloned and used in rival systems.

[17] See https://www.intrinsic-id.com/technology/physically-unclonable-functions-puf/
[18] See https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01252-secure-device-manager-for-fpga-soc-security.pdf

All Rights Reserved. © NCC Group 2015

Conclusion

This document presents an overview of FPGAs from a historical and programming perspective and provides information about the questions that will be asked during a review of FPGA security. The security requirements of an FPGA device begin with the design stage of a product and should be established before implementation begins, particularly if the devices are to be sent to a third party for manufacture and programming.

FPGA manufacturers now provide multiple on-chip mechanisms for FPGA security, but these must be configured correctly to provide the advertised level of security. Older or lower-end products may be using FPGAs that do not have the higher level of security built in, and ongoing use of these chips should be accompanied by a review to ensure that no method of bypassing the protections has been left open.

The security requirements of an FPGA device begin with the design stage of a product and should be established before implementation begins.

All Rights Reserved. © NCC Group 2015

References & Further Reading

FPGA Security Books:

From Features to Capabilities to Trusted Systems, Trimberger & Moore, 2014

Fault Tolerant Design Implementation on Radiation Hardened By Design SRAM-Based FPGAs, Schmidt, 2013

Website: http://www.fpgarelated.com/

Education FPGA boards: http://valentfx.com/

Tutorial: http://valentfx.com/wiki/index.php?title=LOGI_Guide_-_Your_First_Project_using_Xilinx_ISE

Blog: https://blog.digilentinc.com/index.php/history-of-the-fpga/

Spartan-6 User Guide: http://www.xilinx.com/support/documentation/user_guides/ug384.pdf

Understanding metastability: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01082-quartus-ii-metastability.pdf

All Rights Reserved. © NCC Group 2015

CONTACT US

0161 209 5200
response@nccgroup.trust
@nccgroupplc
www.nccgroup.trust

United Kingdom

Manchester - Head office

Basingstoke

Cambridge

Cheltenham

Edinburgh

Glasgow

Leatherhead

Leeds

London

Milton Keynes

Wetherby

Europe

Amsterdam

Copenhagen

Luxembourg

Munich

Zurich

North America

Atlanta

Austin

Chicago

New York

San Francisco

Seattle

Sunnyvale

Asia Pacific

Sydney

www.nccgroup.trust
@nccgroupplc

All Rights Reserved. © NCC Group 2016