Lauri Lötjönen

Field-Programmable Gate Arrays in Nuclear Power Plant Safety Automation

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Thesis supervisor:

Prof. Kai Zenger

Thesis advisor:

M.Sc. (Tech.) Janne Valkonen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Lauri Lötjönen

Title: Field-Programmable Gate Arrays in Nuclear Power Plant Safety Automation

Date: 27.5.2013 Language: English Number of pages: 99

Department of Automation and Systems Technology

Professorship: Control engineering Code: AS-74

Supervisor: Prof. Kai Zenger

Advisor: M.Sc. (Tech.) Janne Valkonen

The field-programmable gate array (FPGA) technology is becoming increasingly common in the instrumentation and control (I&C) systems of nuclear power plants (NPPs). The technology is now being adopted even in the most safety-critical systems. An FPGA is a digital semiconductor device that can be used as a replacement for the current microprocessor-based software systems with which there are problems e.g. in safety justification. An FPGA application has practically the same capabilities as a software application but is less complex and thus arguably easier to qualify. However, the FPGA is still rather new in the nuclear power industry and thus there are no consistent and harmonised international guidance and standards regarding how to design and license FPGAs for NPP safety automation applications.

In this thesis, a general overview of the FPGA technology is first given. The activities in the different phases of the FPGA design and verification and validation (V&V) processes are defined based on processes found in various different sources with the intent to identify best practices for said processes. After the general overview, the current applications, advantages and disadvantages, and other aspects related to FPGAs in NPP safety automation are looked into. A comparison with microprocessor-based systems is also done in order to recognise the perceived benefits of using FPGAs instead of microprocessors. Finally, a case study is presented. In the case study, a fictional but realistic safety automation application is implemented using two FPGA devices. The objectives of the case study are to try out the different design and V&V methods found in the literature, and to get hands-on experience on the FPGA application development and V&V.

Keywords: FPGA, nuclear, automation, safety, design, verification, validation

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Lauri Lötjönen

Työn nimi: Kenttäohjelmoitavat porttimatriisit ydinvoimalaitosten turva-automaatiossa

Päivämäärä: 27.5.2013 Kieli: englanti Sivumäärä: 99

Automaatio- ja systeemitekniikan laitos

Professuuri: Systeemitekniikka Koodi: AS-74

Valvoja: Prof. Kai Zenger

Ohjaaja: DI Janne Valkonen

Kenttäohjelmoitava porttimatriisi (FPGA) -tekniikasta on tulossa yhä yleisempi ydinvoimaloiden instrumentointi- ja säätöjärjestelmissä. Nykyään FPGA-tekniikkaa on alettu käyttää jo kaikkein kriittisimmissäkin turvajärjestelmissä. FPGA on digitaalinen puolijohdetekniikkaan perustuva mikropiiri, jota voi käyttää esimerkiksi korvaamaan mikroprosessoripohjaisia ohjelmistoteknisiä järjestelmiä, joiden kanssa on tällä hetkellä ongelmia muun muassa turvallisuuden ja toimivuuden osoittamisessa. FPGA-sovelluksella pystytään lähes kaikkeen samaan kuin ohjelmistosovelluksella, mutta se on yksinkertaisempi ja täten väitetysti helpompi kelpoistaa eli osoittaa sen toimivuus. FPGA-tekniikalle ei kuitenkaan ole vielä ydinvoima-alalla kansainvälisesti hyväksyttyjä ja yhdenmukaisia ohjeistuksia ja standardeja, jotka ottaisivat kantaa siihen, mitä kaikkea FPGA-pohjaisten sovellusten kelpoistaminen ja lisensiointi vaatii.

Tässä diplomityössä esitellään ensin FPGA-tekniikka yleisesti. Esittelyssä käydään muun muassa läpi eri lähteistä yhdistetyt FPGA-sovellusten suunnittelu- ja V&V prosessit. Eräs työn tavoitteista on löytää näihin prosesseihin liittyvät parhaat toimintatavat. Yleisesittelyn jälkeen käydään läpi nykyiset sovellukset, hyödyt ja haitat, sekä muut FPGA-tekniikan käyttöön ydinvoimaloiden turva-automaatiossa liittyvät asiat. Samassa yhteydessä tehdään vertailu FPGA- ja mikroprosessoritekniikan välillä, jotta voitaisiin tunnustaa FPGA:n edut mikroprosessoriin verrattuna. Lopuksi diplomityössä esitellään case-tutkimus, jossa toteutetaan yksi kuvitteellinen mutta realistinen turva-automaatiojärjestelmä käyttäen kahta FPGA-laitetta. Case-tutkimuksen tarkoituksena on kokeilla eri suunnittelu- ja V&V-menetelmiä, jotka löydettiin kirjallisuudesta, ja saada käytännön kokemusta FPGA-sovellusten suunnittelusta ja V&V:stä.

Avainsanat: FPGA, ydinvoima, automaatio, turvallisuus, suunnittelu, verifiointi, validointi

Preface

I would like to thank Janne Valkonen for his active role as the advisor, and supervisor Prof. Kai Zenger for his counselling and advice regarding the technical aspects and the scope of the thesis. I would also like to thank Jukka Ranta, Jussi Lahtinen, Jan-Erik Holmberg, and Jari Hämäläinen at VTT for their valuable comments during the writing process. Additional thanks goes to Jussi Lahtinen for performing the model checking in the case study.

Otaniemi, 27.5.2013

Lauri Lötjönen

Contents

Abbreviations

2oo4	two-out-of-four
A/D	analogue to digital
AS1	automatic system 1
AS2	automatic system 2
ASIC	application specific integrated circuit
CEC	complex electronic component
CLB	configurable logic block
CMOS	complementary metal oxide semiconductor
CORSICA	Coverage and Rationality of the Software I&C Safety Assurance
CPLD	complex programmable logic device
CPU	central processing unit
D/A	digital to analogue
EARS	easy approach to requirements syntax
ECO	engineering change order
EDF	Électricité de France
EDIF	electronic design interchange format
EEPROM	electronically erasable programmable read-only memory
EPRI	Electric Power Research Institute
ESFAS	engineered safety features and actuation system
FPGA	field-programmable gate array
FSWS	fast stepwise shutdown system
HDL	hardware description language
I&C	instrumentation and control
I/O	input/output
IAEA	International Atomic Energy Agency
IC	integrated circuit
IDE	integrated design environment
IEC	International Electrotechnical Commission
IP	intellectual property
JTAG	joint test action group
LE	logic element
LEC	logic equivalence checking
LED	light emitting diode
LTL	linear temporal logic
MS	manual system
NRC	(United States) Nuclear Regulatory Commission
PAR	place and route
PC	personal computer
PL	priority logic
PLD	programmable logic device
PLS	power limitation system
PS	priority system
PSL	property specification language
RTL	register transfer level
SEE	single event effect
SRAM	static random access memory
SSWS	slow stepwise shutdown system

STA	static timing analysis
SWS	stepwise shutdown system
TMR	triple modular redundancy
USB	universal serial bus
V&V	verification and validation
VHDL	very high speed integrated circuit hardware description language

1 Introduction

This introductory chapter is organised in four brief sections. First, the background for the work done in for thesis is explained in Section 1.1. Then, the current situation of the field-programmable gate array (FPGA) technology in nuclear power plant (NPP) safety automation is shortly described in Section 1.2. After that, the objectives of this work are recited in Section 1.3, and finally the organisation of the rest of the chapters of the thesis is elaborated in Section 1.4.

1.1 Background

This work has been done at VTT Technical Research Centre of Finland as part of the "Coverage and Rationality of the Software I&C Safety Assurance" (CORSICA) -project of the Finnish Research Programme on Nuclear Power Plant Safety 2011-2014 (SAFIR2014, 2013). One of the tasks in the CORSICA project is looking into new technologies that could be used as the implementation technology of the digital NPP instrumentation and control (I&C) systems such as safety automation. Among these technologies is the FPGA.

The NPP-related research of FPGAs was started at VTT in 2011. In early 2012, the first publication regarding the subject was published (Ranta, 2012). Later that year, a case study was conducted where a fictional but realistic safety function called the Stepwise Shutdown System (SWS) was implemented using FPGA technology (Lötjönen, 2012). Now as part of this work, the SWS case study is expanded by designing and implementing a more complex system using two different FPGA devices. In the new case study, more emphasis is put on the verification and validation (V&V) of the design.

The FPGA technology is rather new in the nuclear power industry, and therefore the FPGA-specific guidance and regulations have not yet matured. There are inconsistencies between the guidance and regulatory views of different organisations and regulators on how to design and qualify FPGA-based systems for NPPs. The interest toward the use of the FPGA technology in NPP safety automation applications is rising world-wide, and thus there is a need to harmonise the international regulations and guidance so that they can be correctly understood by the power utilities and system vendors. The work done for this thesis can be seen as part of the international harmonisation.

1.2 State-of-the-art

International research regarding the use of the FPGA technology in NPP safety I&C systems is on-going by many organisations. There are a lot of actual projects where the FPGA is used as e.g. a replacement for old systems in NPP modernisation projects, and also as an integral part of new I&C platforms in new NPPs. Because there are no internationally harmonised guidance and regulations regarding the design and V&V methodologies of FPGAs in NPPs, different standards and guidance are used in many projects where NPP safety automation is implemented using the FPGA technology. It is not yet known, what kinds of evidence should be required regarding the correctness of the applications.

The well-defined software-related standards most commonly used in the design and V&V of software-based applications are not by themselves adequate for use with

FPGAs because the design of FPGA applications also requires the hardware aspects to be taken into account. The need to address the hardware-related issues makes the design process more complex as the designer needs to have knowledge of both the software and the hardware aspects of the FPGA design.

In 2012, the International Electrotechnical Commission (IEC) published the standard 62566 (IEC, 2012) specifically regarding the development of FPGA-type devices for category A functions in NPPs. However, the guidance in the standard is seen as insufficient because its contents are not as comprehensive as was initially expected. In many places, there are merely references to other e.g. software-related standards.

The International Atomic Energy Agency (IAEA) has also shown interest in the technology by joining the sponsorship of a dedicated "Topical Group on FPGA Applications in NPPs" that has held annual meetings five times by 2013 (IAEA, 2013). The meetings are organised as workshops where representatives from regulators, power utilities, system developers and vendors, and research organizations gather to discuss the current tides regarding the subject. As part of this work, the fifth workshop held in 2012 in Beijing, China was attended. Related to these workshops, the IAEA is currently working on a publication regarding the applications of FPGA technology in NPPs and it is due to be published in 2014 (IAEA, 2013).

Even though the FPGA is rather new in the nuclear power industry, it is as such a mature technology and has been extensively used in other safety- and mission-critical applications such as aviation and space flight. Therefore, the standards and guidance describing the methods and activities in the design and V&V processes from the other fields may prove to be useful when developing applications for FPGA-based NPP safety automation applications as well.

1.3 Objectives

One of the main objectives of this work is to try to identify the best practices for the design and V&V methods used in the development of FPGA-based systems for NPP safety automation. This is important because the technology is new in the field and the international standards and guidance are not yet consistent and mature. In this work, the perceived best practices are first collected from various sources in literature and then combined together to form consistent design and V&V processes. As it is still not clear what V&V methods are sufficient to provide enough evidence of the correctness of the applications, it is also interesting to assess whether using these methods provides sufficient evidence or are additional methods still required.

Another objective is to assess the advantages and disadvantages of using FPGA technology in NPP safety automation and whether it actually provides added value compared to using e.g. software-based systems. In many sources in literature, experience of actual NPP safety automation applications implemented using the FPGA technology is reported which may be helpful when assessing the suitability of the technology for these applications.

With the help of a case study, the identified design and V&V methods are tested to see whether they are easily applicable to actual systems, and also to get hands-on experience of the challenges and limitations related to them and to the overall design and V&V processes. To get a realistic impression it is not enough to merely read about the methods, and therefore actual FPGA devices are used. Experience gained from studies such as this one, where the methods are applied to actual designs, may be valuable for the on-going harmonisation of the international standards and guidance.

The scope of the case study is reduced to merely trying out the different design and V&V methods instead of aiming to produce comprehensive evidence of the correctness

of the system in the case study. Another objective of the development of the case study is to generally build up VTT's competence for assessing FPGA-based NPP safety automation applications. It is possible that VTT will be asked to assess such applications as they may be considered for the Finnish NPPs in the future.

1.4 Structure of the thesis

After this introductory chapter, the FPGA technology is described in Chapter 2. The description includes a general overview of the technology, a look at the hardware aspects, and detailed explanations of the identified design and V&V processes for developing and testing FPGA-based applications.

Different aspects related to the use of the FPGA technology in NPP safety automation are then looked into in Chapter 3. The chapter includes a short description of the main characteristics and general principles of NPP safety automation applications after which some of the difficulties regarding conventional software-based systems are listed to justify the search for new technologies such as the FPGA.

Following this in Chapter 3, there is an explanation of how the FPGA technology can be used in NPP safety automation applications and what are the main advantages and disadvantages. The FPGA is compared to microprocessors and then some examples of actual FPGA-based systems are given at the end of the chapter.

Chapter 4 is dedicated to the case study where a fictional but realistic safety automation application was implemented on two FPGA devices. In the chapter, the design and V&V processes that were followed through in the case study are described with a lot of illustrations of the design products along the way.

2 Field-Programmable Gate Array Technology

This chapter describes the field-programmable gate array (FPGA) technology in general. The purpose of this chapter is to provide general knowledge of the technology without any special emphasis on the nuclear power industry. The NPP applications of the FPGA technology are looked into in Chapter 3.

This chapter is organised in four sections. An overview of the FPGA technology is given in Section 2.1. Section 2.2 describes the hardware aspects of the technology. In Section 2.3, the different phases of application development process for FPGAs are described. Section 2.4 elaborates methods to verify and validate FPGA applications and design phase products.

2.1 Overview

Field-programmable gate array technology (FPGA) was invented in 1984 by a company called Xilinx (Maxfield, 2004). An FPGA is an integrated circuit (IC) device and is sometimes classified as a programmable logic device (PLD), a complex programmable logic device (CPLD), or a complex electronic component (CEC). The classification varies between different organisations, and this sometimes causes confusion especially in the nuclear power industry, where the technology itself is still rather new (Arndt *et al.*, 2012). Essentially, an FPGA is a digital programmable IC that contains thousands or millions of logic gates and interconnections that can be configured to implement desired functionality. The two biggest FPGA manufacturers are Altera and Xilinx, but there are some smaller manufacturers as well, such as Microsemi (formerly Actel).

FPGA evolved from a technology called Application Specific Integrated Circuit (ASIC). ASICs are functionally quite similar to FPGAs: they contain logic gates that are configured to implement functionality. However, the term “field-programmable” in FPGA refers to the feature of FPGAs that they are programmed by the customer rather than by the FPGA manufacturer. The programming of ASICs is done by the manufacturer during the manufacturing process. This makes ASIC technology ideal for mass produced products, where identical devices are used for example in millions of similar applications. However, the development of ASICs is costly because a special cast needs to be made in order to manufacture them. One of the early applications of FPGAs was prototyping ASIC designs (Maxfield, 2004).

In the beginning of 1990s, FPGAs were mostly used in telecommunication and networking applications. They were ideal for high speed processing of big chunks of data that is typical of those applications. By the end of the 1990s, the rapidly evolving FPGA technology had spread to multiple fields such as automotive industry, consumer electronics, aviation, some industrial applications, and even space applications. (Maxfield, 2004)

FPGAs are manufactured blank i.e. the transistors are not configured in any particular way. This enables a low price and a great flexibility in their use. FPGA manufacturers provide software tools that are used to create applications for the devices. Many FPGAs are also re-programmable, which enables an iterative design process, where a designer can test designs on actual hardware and then make modifications to them later on. This also makes an FPGA application updatable, which is beneficial for preventing obsolescence after years of operation (Arndt *et al.*, 2012).

FPGA is said to be “hardware designed like software”, meaning that the design process resembles software development in many ways but the end product is more like a hardware application (Ranta, 2012). FPGA devices can implement very complex functionality like software, but can still offer the reliability of a hardware application.

FPGAs do not require an operating system or task switching, and are therefore considered less complex than software applications (EPRI, 2009). However, as the FPGA design process is similar to software development, it is also susceptible to similar errors.

Although much of the FPGA application development is similar to software development, it requires additional steps where the hardware aspects are taken into consideration. A designer must be familiar with hardware related issues such as input/output pin voltages and assignments, signal propagation delays and timings, clock frequencies, and device resources. When developing applications for traditional microcontroller-based systems, the designer typically does not need to take into account what type of a central processing unit (CPU) is used or to which pins of the CPU various devices are connected. A high level programming language, an automatic compiler, and the operating system can take care of these aspects.

2.2 Hardware

This section covers the hardware-related aspects of FPGAs. First, the internal structure and the main components of an FPGA device are elaborated. After that, the different memory technologies that are used to store and maintain the configuration of the FPGAs are described. The operational principle of an FPGA is then described after which some types of failures that may occur with FPGAs are described.

An FPGA device is typically a black rectangular device a couple of centimetres in diameter whose bottom is covered with pins that connect it to a circuit board. Inside the device, the actual FPGA core chip is connected to a pin package. On the chip there are four kinds of major components: input/output (I/O) blocks, configurable logic blocks (CLBs), wirings, and switch boxes. A simplified illustration of the internal structure of an FPGA device is in Fig. 1.

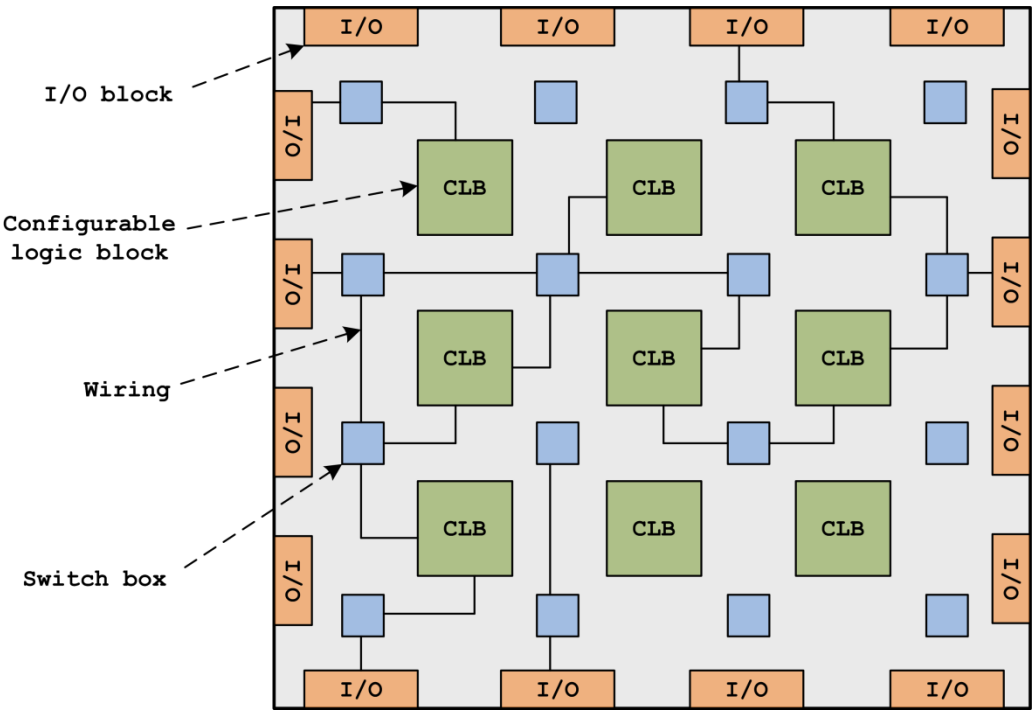

Figure 1: Simplified illustration of the internal structure of an FPGA device

The CLBs comprise of the basic semiconductor components called ‘system gates’ on the chip. The overall amount of system gates varies from thousands to millions.

Different manufacturers name and implement the I/O blocks and CLBs differently but the functions are essentially the same between manufacturers (Smith, 2010). For example, Actel calls them VersaTiles (Microsemi, 2012a), and Altera calls them logic elements (LEs) (Altera 2006).

The I/O blocks handle the communication between the FPGA and other devices through the pins of the FPGA device. The I/O blocks typically contain registers, buffers, pull-up resistors, voltage translators, and other components that are assisting communication (Smith, 2010). The I/O blocks also transform data into suitable formats to make it be usable inside or outside the FPGA (EPRI, 2009).

The CLBs are the lowest abstraction level components that the designer can observe and manually affect during the design process. They consist of smaller components such as lookup tables, flip-flops, registers, adders, multiplexers, etc., which in turn are constructed using the basic system gates. Using these basic features, the CLBs can be configured to perform various tasks. Additionally, An FPGA may contain predefined non-configurable blocks e.g. for implementing memory.

The capacity of an FPGA is measured in the amount of CLBs rather than system gates. This means that even though one FPGA has more system gates than another FPGA, it can still have a smaller capacity if it has fewer CLBs than the other FPGA. In this case, the other manufacturer has used fewer system gates to implement the CLBs.

The switchboxes can be implemented using different memory technologies. As the wirings in Fig. 1 enable the communication between the CLBs and I/O blocks, the switch boxes determine how the wirings are configured. In the figure, only some wires are drawn but actually there are wires between all the elements with only some being used. The switch boxes are memory elements that can be implemented using three different technologies: static random access memory (SRAM), flash, and antifuse (Ranta, 2012). Each of these has advantages and disadvantages over each other.

SRAM is the most common memory technology found in FPGAs (Maxfield, 2004). It consists of six complementary metal oxide semiconductor (CMOS) transistors and implements a functionality of a flip-flop that can be used as a register element (Ranta, 2012). Although six transistors are needed, the SRAM-based memory elements are physically smaller due to the more advanced CMOS technology. This makes it possible to pack vast amounts of transistors in a smaller space than for example with flash-based memory elements.

Being at the forefront of the IC industry, CMOS is more advanced and further developed than the other technologies. One big advantage of SRAM-based FPGAs is that all the other components on the FPGA are also CMOS-based. Therefore SRAM-based FPGAs do not require special manufacturing processes that are needed when combining different technologies (Maxfield, 2004).

SRAM is a re-programmable memory technology that is also volatile, which means that it retains its configuration only as long as the power is enabled i.e. a certain voltage is present (Ranta, 2012). SRAM-based FPGAs are used for example in high-performance applications such as network routers, and since they are easily re-programmed, they are ideal for developing and testing new designs on actual devices. Due to their volatility, SRAM-based FPGAs need to be configured each time they are powered up, which requires them to have an external non-volatile memory where the configuration is stored and from which it can be read. SRAM is more susceptible to disturbances caused by for example radiation than other memory technologies (Ranta, 2012).

Flash is based on a technology called electronically erasable programmable read-only memory (EEPROM) and, like SRAM, it is re-programmable. Unlike SRAM, flash is

non-volatile, which means that a voltage is not needed to maintain the configuration. Although only one flash transistor is needed to implement a memory element, the technology is less mature than in SRAM-based FPGAs. Due to the less advanced technology, flash-based FPGAs have typically smaller capacity than SRAM-based FPGAs. Flash-based FPGAs are used in low-power applications such as mobile devices. As they have a better resistance to disturbances than SRAM-based FPGAs, they can be used in more harsh environments such as aerospace (Ranta, 2012).

Antifuse is the simplest and most reliable of the three technologies. Its name derives from "fuse" that refers to a device that is used to cut power if too large a current flows through it. Antifuse acts the opposite way: initially it does not let current through but when a large enough current is introduced, a path for the current is "burned" through the antifuse element. Antifuse is immune to most disturbances because its configuration is mechanical and irreversible (NRC, 2010a). For the same reason, the information security of an antifuse-based FPGA is the best. Antifuse-based FPGAs are suited for high-reliability applications such as safety-critical systems and spaceflight (Ranta, 2012).

When an FPGA operates, signals propagate via the wirings and switchboxes through the configurable logic blocks (CLBs) that operate according to their configuration. An FPGA does not execute instructions or lines of code like a microprocessor. Instead, the static configuration that does not change during operation determines the functionality. FPGAs operate truly parallel, since each CLB performs its functions independently of others and using its own resources.

For example, consider a two-input logical AND-block. Its inputs could be assigned to some pins that are connected to for example two hardware switches. Similarly, its output could be connected to a pin that is connected to a light emitting diode (LED) that indicates the result of the AND operation. If the switches are turned on indicating "TRUE", signals would propagate via the wirings and switchboxes to the CLB or CLBs that implement the AND-block. The CLBs would then execute the AND-functionality and, as a result, would send a "TRUE" signal via the wirings and switchboxes to the pin that the LED is connected to.

As another example, consider the AND-block from the previous example. This time, five instances of the gate are implemented in an FPGA. The I/O-ports of each instance would be connected to different pins connected to different switches and LEDs. Now, additional resources, i.e. CLBs, I/O blocks, and wirings are required to implement the additional gates. The function is similar to the case where there was only one AND-block: when a switch is turned, the corresponding wire conveys the signal to the corresponding CLB(s) where the signal is processed and the corresponding result is assigned to the wire that is connected to the corresponding LED. Each instance is implemented in separate hardware and separate wirings which enables a truly parallel execution. Each AND-block is oblivious of the others.

There are a few kinds of faults that can occur when using an FPGA. Detailed causes and effects of different faults are described in (Ranta, 2012). Some cause permanent damage and some may only be temporary but can still affect the operation in a possibly disastrous way. Some permanent faults are the result of aging and degrading due to long operation times and/or a harsh operating environment. Faults introduced in manufacturing and during application design are also considered permanent faults, although design faults in re-programmable FPGAs can be corrected through re-configuration.

Transient faults are faults that do not cause permanent damage but may cause failures in the operation. Common sources of transient failures are single event effects (SEEs),

which can be caused by power fluctuation or radiation. SEEs may alter the configuration or the states of registers of the CLBs of the FPGA. SEEs may also cause permanent damage to the chip if severe enough.

The faults due to aging and the operating environment introduced over time are mostly related to the physical properties of the FPGA device. More details about the physical aspects of different faults are given in (Ranta, 2012).

2.3 Design

Developing applications for FPGAs is similar to developing software. Typically, the development involves designing and writing code that is similar to normal software source code. However, an FPGA designer must also consider hardware-related issues which in software development are taken care of by the compiler and the operating system. Toward the end of the design process, the phases resemble those found in integrated circuit (IC) design rather than software development (Wikipedia, 2013)

The FPGA design process is described in various sources, and most of them contain the same basic phases. Some go into more detail and divide some phases into multiple parts. In this section, a general overview of the FPGA design process, inspired by various models from several sources (NRC, 2010a; Smith, 2010; Gaisler, 2002; EPRI, 2009), is elaborated phase by phase. In Fig. 2, the design process is illustrated along with the corresponding V&V phases and output documents.

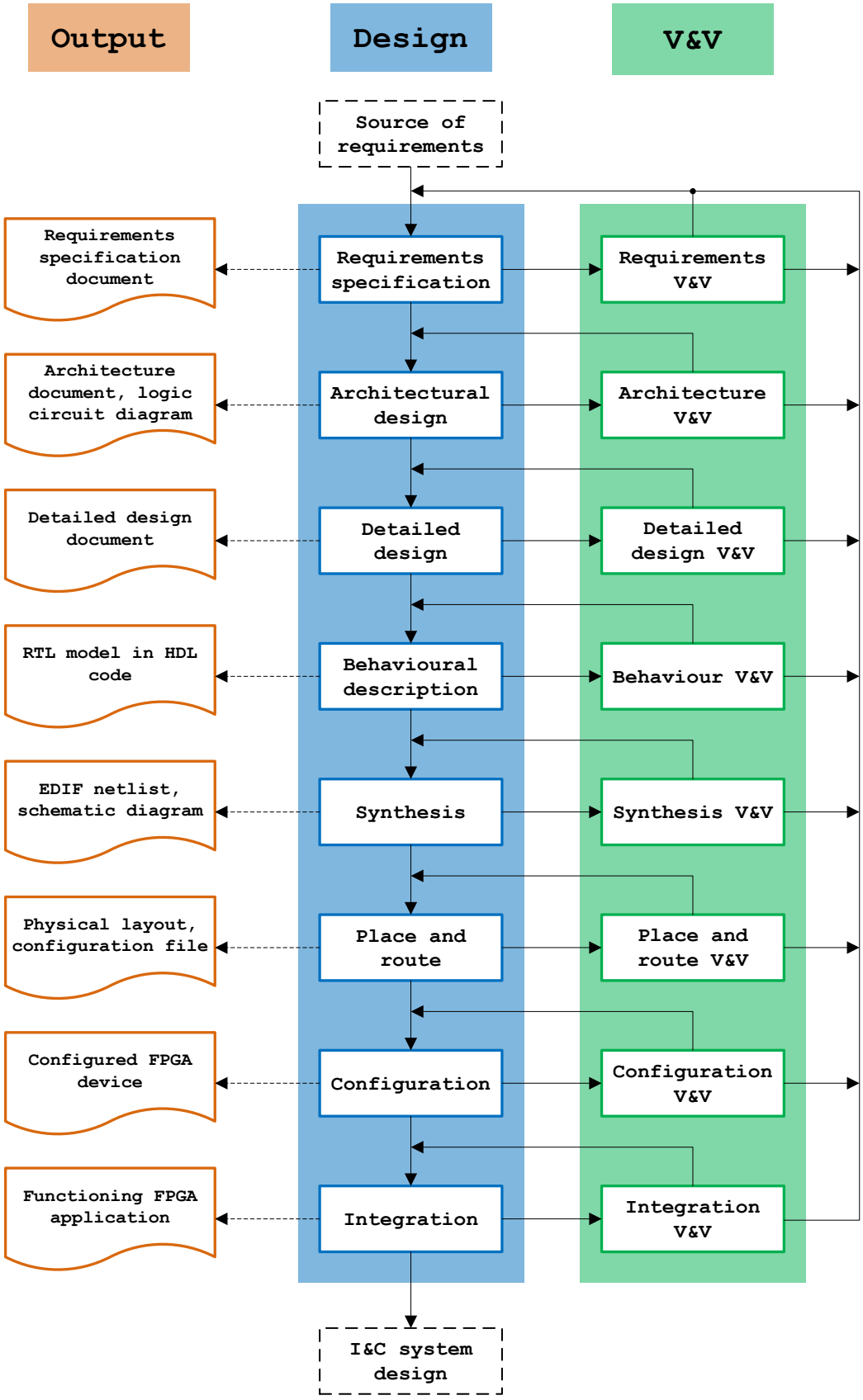

Figure 2: Combined design and V&V processes with the types of output products of each phase

The design and V&V processes in Fig. 2 represent the author's subjective view of the most suitable flow of the phases although they bear many similarities to those presented in the NUREG/CR-7006 of the United States Nuclear Regulatory Commission (NRC, 2010a). However, the phases are named differently from the NUREG/CR-7006, and each phase has a corresponding V&V phase. The names of the design phases in the 'Design' column in Fig. 2 seem to be more common than those in the NUREG/CR-7006 in other sources.

The V&V phases in the NUREG/CR-7006 only contain individual activities such as reviews and simulation. In the 'V&V' column of Fig. 2, the phases are more ambiguously named because each phase may contain several activities and not merely simulations or reviews. The V&V activities related to different phases are elaborated in Section 2.4.

The 'Output' column in Fig. 2 contains the types of products that are the outputs of each design phase. The purpose of the output boxes is to give an idea what type of outputs are generated in each design phase, so there may be additional documents to those listed in the figure.

Requirements specification

The requirements specification starts the design process. The purpose of the requirements specification is to completely and correctly specify all the features that a system is required to have so it fulfils its intended purpose and function. Sometimes requirements specification is not considered to be part of the actual design but instead an independent document that the following design documents are based on and compared to.

The requirements may be based on a concept of the system, or they can be imposed by regulations and legislation. This is especially characteristic of the nuclear power industry. The requirements may derive from higher level instrumentation and control (I&C) system architecture requirements. The requirements are usually specified in a requirements specification document which is a textual document written in natural language. Also graphical requirements specifications exist. They may impose requirements regarding a certain type of architecture for example.

The requirements can be roughly divided into functional and non-functional requirements. Functional requirements describe the required behaviour of the system, and non-functional requirements describe requirements related to e.g. the response times, performance, operating environment, interface, hardware characteristics, radiation tolerance, etc. The requirements are typically described as individual clauses. Some characteristics that a good requirement should have are (SAREMAN, 2012, Tommila, 2012):

- Correctness: requirements corresponds to the needs of the stakeholders and the purpose of the system
- Feasibility: requirement can be satisfied within the constraints of the available technology
- Unambiguity: requirement only has one interpretation
- Traceability: requirement can be traced back to a higher level source and to the lower lever solution
- Consistency: requirement does not conflict with other requirements
- Verifiability: system can be tested against the requirement
- Completeness: requirement does not lack relevant information

The requirements specification document can be fairly similar to a software requirements specification document. In an FPGA requirements specification document, there is additional information regarding FPGA-specific hardware issues such as capacity and performance requirements, circuit board size, radiation and electromagnetic disturbance tolerance, input/output response times, and clock frequencies (NRC, 2010a).

One way of forming requirements is to use the Easy Approach to Requirements Syntax (SAREMAN, 2012) which is a requirement template that utilises common natural language expressions and keywords assigning them special meanings. The EARS is now presented as an example of how the requirements could actually look like. In the EARS, there are five sentence types:

1. Ubiquitous: The <system name> shall <system response>
2. Event driven: When <optional preconditions> <trigger>, the <system> shall <system response>
3. State-driven: While <in a state>, the <system> shall <system response>
4. Unwanted behaviour: If <optional preconditions> <trigger>, the <system> shall <system response>
5. Optional: Where <feature>, the <system> shall <system response>

A ubiquitous requirement describes a general attribute of the system e.g. *"The System 1 shall have five inputs"* or *"The indicator light shall blink once a second"*. This type of a sentence can be used to describe both non-functional and functional requirements as demonstrated above respectively.

An event-driven requirement sentence specifies a required response of the system to some triggering event. The keyword indicating the event-driven functionality is 'when'. For example, the following requirement describes a simple functionality: *"When input is activated, output shall be activated"*. Now, the function is only initiated when the activation of the input signal is detected regardless of the subsequent state of the signal.

A state-driven requirement sentence describes behaviour when a system or a signal is in a specified state. The keyword indicating a state-driven functionality is 'while'. Now, the specified system response is required to take place when a system or a signal is in a specified state. For example, the following requirements describes a simple function: "*While input 1 and input 2 are active, output shall be active"* meaning quite axiomatically that while the input signals are in an active state at the same time, the output signal shall be made active.

When there is a need to specify unwanted behaviour, the keywords used in the EARS are 'if' and 'then'. The sentence structure is similar to the event-driven sentence. For example, describing a simple functionality: *"If the buttons are pressed in the wrong order, then the system shall disregard them"*.

The final sentence type is 'Optional'. This means that when a system has a certain feature, it shall cause a certain response. For example, *"Where the automation system has support for model-based control, the system shall use a linear quadratic controller for process control"*. This sentence type, along with the ubiquitous sentence type, is well suited for specifying non-functional requirements as well as functional requirements.

The EARS sentences can be combined to achieve a requirement where e.g. a specified event along with a certain state causes a response. For example, such a

requirement could be: "*When the input is activated, while the level is below the high limit, the pump shall be activated"*. Now, a pump system can be activated by an 'input' when a certain level is below a specified limit.

The required design properties can be formalised by using a formal language such as the Property Specification Language (PSL) (IEEE, 2010). PSL is an extension of Linear Temporal Logic (LTL) that can be augmented into HDL code used in FPGA design. Formalised requirements are necessary for the employment of certain formal verification methods that are described in Section 2.4.

Architectural design

Architectural design consists of dividing the whole system into functional modules. Decisions about the module division, redundancy, and other architecture-related features are made in this phase. The division can be made so that it is possible to use pre-developed modules that are called intellectual property (IP) cores. By using an IP core, it is possible to implement the functionality of many basic logic functions and even an entire microprocessor. (Ranta, 2012)

The architecture should be designed to be device independent. However, if an FPGA device has already been selected for the design, the designer should take into account the possible limitations of the particular device. If the architecture is sufficiently generic, the resulting document could be even used as a basis for developing an equivalent software application of the system as well.

The results of architectural design should be a textual document that describes the architecture in natural language, and an architecture diagram illustrating the architecture graphically. The textual document should determine why the architecture has been defined as it is, and how the choices that were made are traceable to the requirements. The architecture diagram should illustrate the division to modules along with the other features mentioned above. A simple example of an architectural diagram is in Fig. 3.

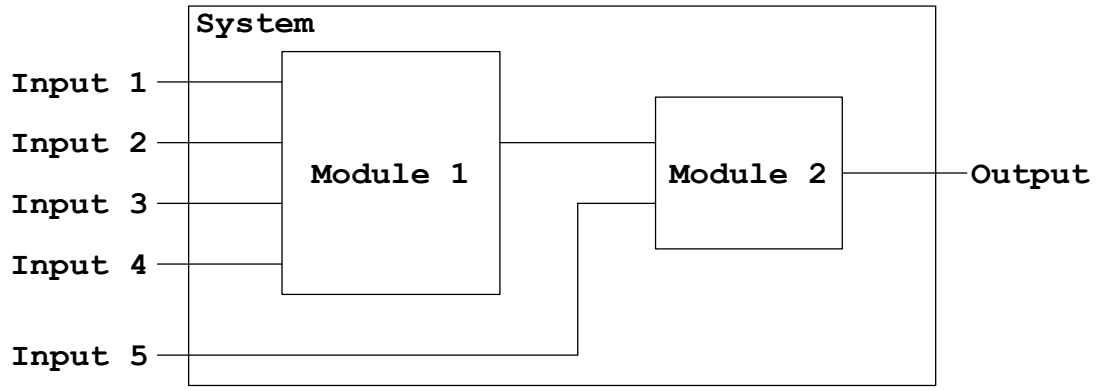

Figure 3: Illustration of the architecture of a very simple example system

An illustration of the architecture of a simple system whose only output is active only when all the five inputs are active is in Fig. 3. In this very simple case, the modules (Module 1 and Module 2) in the figure would only consist of logical AND-blocks.

Architectural design does not need to consider how the modules are implemented. The example in Fig. 3 shall be used in the upcoming design phases as well to illustrate the activities and the output products of each phase. The implementation of the modules using more elementary blocks is done in the detailed design phase which is described next.

Detailed design

In detailed design, the modules that resulted from architectural design are defined in more detail i.e. the contents of the modules are designed. The desired functions of the modules are implemented using for example basic combinatorial logic gates, timers, lookup tables, memory elements, or other low-level function blocks. Also the internal redundancies, diagnostic features, built-in self-tests, and other non-functional features are designed into the modules in this phase.

The architectural document should be refined into a more detailed logic circuit diagram. For creating the logic circuit diagram, symbols defined e.g. in the IEEE standard 91 can be used (IEEE, 1996). Together with architectural design, detailed design constitutes what is sometimes called the preliminary design phase (EPRI, 2009).

The detailed design phase is already FPGA-dependent since the characteristics such as resources, and timing properties of the target FPGA need to be taken into account (Ranta, 2012). In addition to the detailed circuit diagram, there should be a detailed design document that contains information such as the I/O pin assignments, clock frequencies, and timing requirements. This information is vital for later design phases when the design is starting to morph into a hardware application.

'Module 2' of the example system introduced in Fig. 3 would only consist of one two-input AND-block. A logic circuit diagram depicting Module 2 is in Fig. 4. In this case, the box that previously represented the module is now completed with a standardised shape of a two-input AND-block.

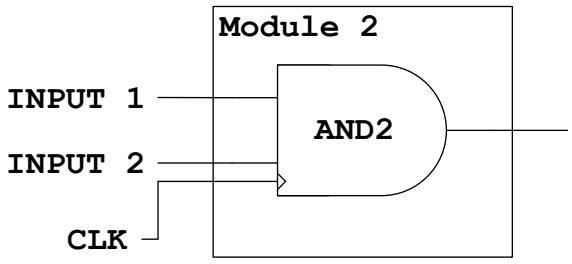

Figure 4: Detailed design logic circuit diagram of Module 2 of the example system containing a two-input AND-block

In Fig. 4, there is now an additional input signal 'CLK' compared to the inputs in the architecture design. CLK represents the clock signal that drives all the blocks in a synchronous design. This is one of the hardware-related features that need to be implemented in the detailed design phase. A similar logic circuit diagram is created of all of the modules and of eventually of the entire system.

Behavioural description

Behavioural description corresponds to writing or generating code in software development. Instead of traditional programming languages such as C or Java, lower abstraction level hardware description languages (HDLs) are used. HDLs were originally used for making models of existing electronic devices for simulation and analysis purposes. However, it turned out that HDLs work the other way as well; they can be used for developing new hardware applications.

In some cases, like in software development, the HDL code can be automatically generated from a higher level description using a software tool (Ranta, 2012). These

methods are described later on in this section. Most commonly used HDLs are Very high speed integrated circuit HDL (VHDL) (IEEE, 2009) and Verilog (IEEE, 2006). Based on various conference papers, reports, and discussions, VHDL seems to be more common in the nuclear domain (Druilhe *et al.*, 2010; EPRI, 2009).

VHDL resembles Ada which is a language that has been used for many safety- and mission critical applications (ISO/IEC, 2012). It is natural to use VHDL for implementing these kinds of systems in an FPGA, since it is based on a language that was designed for and has already been used in safety- and mission-critical applications. Ada and VHDL also have a good support for concurrent programming, which makes designing applications for the naturally concurrent FPGA devices easy.

As an example, the two-input logical AND-block of the simple example system previously introduced, now written in VHDL, is in Program 1. The AND-block was used in the power limitation system (PLS) case study system described in Chapter 4. The PLS case study was implemented in VHDL, and more examples and explanations of VHDL are presented in Chapter 4 along with the rest of the case study.

```
library ieee;
use ieee.std_logic_1164.all;

entity AND2GATE is
   port ( CLK    : in  bit ;
          INPUT1 : in  bit ;
          INPUT2 : in  bit ;
          OUTPUT : out bit );
end entity AND2GATE;


architecture BEHAVIORAL of AND2GATE is
   signal RESULT : bit; -- result is stored here before assigning to
the output
begin
   process ( CLK, INPUT1, INPUT2 ) is
   begin
      -- the AND-operation is done only when there is a rising edge on
the clk signal
      if CLK'event and CLK = '1' then
      RESULT <= INPUT1 and INPUT2;
      end if;
   end process;

   -- after each clk cycle, the result is assigned to the output port
   OUTPUT <= RESULT;

end architecture;
```

Program 1: Two-input logical AND-block written in VHDL

The result of the behavioural description is a so called register transfer level (RTL) model of the system. RTL model is a parallel model of an electronic circuit that describes the behaviour caused by signals being processed according to the configured logic and then transferred between memory registers synchronously (IEC, 2012).
Behavioural description can be seen as a melting point of the software and hardware aspects of the FPGA design process. The RTL model is formed using a software-like method, e.g. writing the HDL code, but the concept of the RTL model or RTL level is commonly used in IC design (Wikipedia, 2013). The RTL design should be done so that

it is FPGA-independent. This enables better portability between devices, which in turn offers better obsolescence resistance, because for example an FPGA that is no longer available on the market could then be easily replaced by a newer FPGA.

The higher-level methods enable describing the behaviour using another language such as C, Matlab, or LabVIEW (Ranta, 2012). This enables easier implementation of complex functions and can speed up the design process. This may however result in unnecessarily complex constructs and HDL code that is very difficult to read. That would make the verification of the HDL code more difficult. The need for the higher-level design entries to be compiled into HDL code adds one more software tool into the already heavily software tool-dependent design process. This may impose an additional need to verify the tool to ensure that it does not generate errors during the compilation.

Some higher-level languages are intended for hardware design. Examples of these are IEEE 1800 SystemVerilog (IEEE, 2007), IEEE 1666 SystemC (IEEE, 2012), and Esterel (Boussinot, 1991). They offer various useful features like built-in support for verification and validation methods such as formal verification. However, these languages are intended to make designing large and complex systems easier. Considering that NPP safety system applications are usually very simple, the higher-level languages may not offer any added value to the design process.

If the requirements were formalised using PSL, they could be directly augmented into the HDL code. PSL can be augmented into either VHDL or Verilog and it uses the boolean expressions of the underlying HDL to determine the states of the system. Additionally, it uses temporal logic to determine the behaviour of states in time domain. PSL generates assertions if certain properties are not met, i.e. if at some point the system is in a state that it should not be (Doulos, 2013). These assertions can then be used in simulation and formal verification to catch errors as will be seen in the V&V Section 2.4.

The HDL code should be comprehensively commented and all the decisions that were made should be explicit in a way that tracing the properties of the code back to the previous phases is possible. There are some guides such as (ESA, 1994; NRC, 2010a) for writing good VHDL code. Therefore, producing additional documents in addition to the document that contains the HDL code is not necessary in this phase.

Synthesis

In synthesis, the HDL code is compiled and turned into a mid-level netlist. This netlist is usually generated in the textual electronic design interchange format (EDIF). Also a graphical schematic document that describes the gate-level design architecture is generated. EDIF netlist describes the blocks and interconnections in such a way that it is understood by software tools. EDIF is an FPGA-independent format whereas the schematic takes into account some internal resources and attributes of the specific FPGA device. An example of a schematic diagram is in Fig. 5. It is the two-input logical AND-block from the simple example whose VHDL code was presented in Program 1. The VHDL code was synthesised using the synthesis tool of Quartus II Suite by Altera (Altera, 2011).

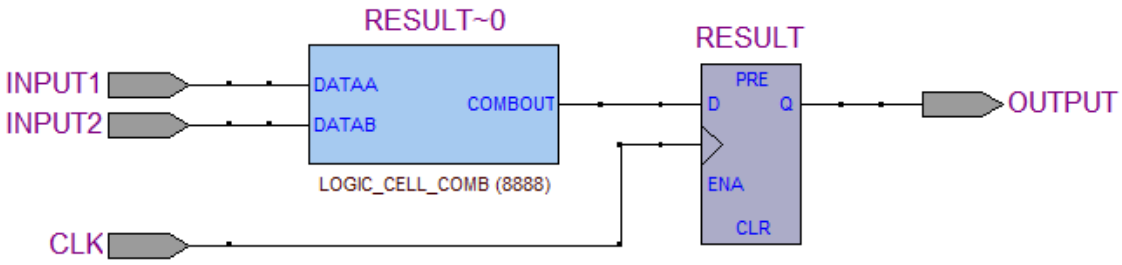

Figure 5: Example of a schematic diagram (two-input logical AND-block)

During synthesis, the logic block design is optimised and the 'synthesisability' of the written HDL code is also verified. Due to their original purpose of modelling existing systems, HDLs have many of the high-level features that are in many other programming languages as well. In Fig. 5, the synthesis tool has divided the AND-block into two parts: the combinatorial part and the register element. The box labelled "RESULT~0" is the combinatorial part where the AND operation is done, and the box labelled "RESULT" represents the signal in which the result of the AND operation is stored before assigning it to the output port.

However, when creating hardware designs using an HDL, only a subset of the features of the language can be automatically turned into hardware. For example, only about ten per cent of VHDL is in the synthesisable subset that is defined in an extension of the IEEE 1076 standard named IEEE 1076-6 (IEEE, 2004). The reason that only a small subset of the language is synthesisable is that most features of the language are simply too complex for the software tools to automatically transform into hardware equivalents.

The designer may skip the HDL coding of the behavioural description phase by manually creating the schematic diagram that results from synthesis. This method eliminates the possibility of errors introduced during HDL synthesis through the omission of the entire phase. It also gives the designer more control of the design. However, creating a schematic manually is very laborious and while for example the U.S. NRC still recommends it for safety-critical systems, it is expected to be replaced by the HDL method completely (NRC, 2010a).

After the automatic synthesis, the designer may alter the output products in order to add features that might have been omitted during synthesis or to otherwise modify the design manually. This is called an engineering change order (ECO). ECOs are a potential source for introducing errors into the design, but proper verification should catch these errors as will be seen in the V&V Section 2.4. ECOs should be documented in order to maintain traceability, if the design structure changes dramatically because of them.

Place and route

The place and route (PAR) phase takes the design one step closer to becoming hardware. The EDIF netlist generated during synthesis is now used to create a model of the physical implementation of the design on the specific FPGA. Now the layout and configurations of the configurable logic blocks (CLBs), I/O blocks, and switchboxes are defined.

The designer must specify for example to which pins the I/O ports of the design are connected, and what voltage should be used in the I/O pins. The designer can also affect how the CLBs are utilised. This means that the designer can affect which functions are

assigned to which particular CLBs. This can be done using a ‘floor plan’ editor usually provided in the software tool that is used for PAR. An image of a floor plan editor that is used to edit the physical layout of the two-input logical AND-block presented previously is in Fig. 6. The tool used for PAR was integrated in the Quartus II.

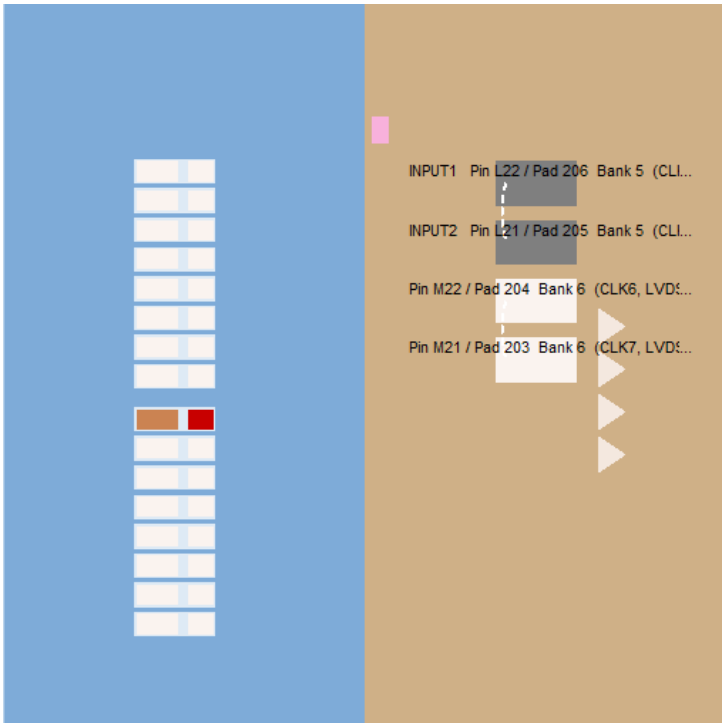

Figure 6: Part of the physical layout model of the two-input logical AND-block

In Fig. 6, one CLB on the blue background and four I/O pins on the brown background are shown. No other CLBs are utilised by the AND-block so the view has been zoomed in from the view where all CLBs and other resources are visible. The white rectangles on the blue background are the resources of one CLB. One of the rectangles is currently utilised which is indicated by the brown-red colouring. The combinatorial operation is done in the brown part and the result is stored in the red part.

The boxes on the beige background represent I/O pins. Some of the ports such as INPUT1, and INPUT2, are assigned to the pins in the figure. The designer can assign the design components to different CLBs or I/O pins. The changes manually made in this phase are called ECOs like in synthesis.

In addition to the model of the physical layout, a low-level netlist and a configuration file are generated. The netlist can be used for technology-level simulation. The configuration file is a bit stream that can be used to configure the specific FPGA device. For SRAM- and flash-based FPGAs, the configuration file is simply called a configuration file or a programming file, and for antifuse-based FPGAs, the configuration file is called a ‘burn list’ due to the nature of the configuration process.

Configuration

Configuration is the phase where the design is finally transferred to the FPGA by configuring it with the programming file. This phase is most often called ‘programming’ and sometimes ‘implementation’ or ‘prototyping’. In this text, the term

‘configuration’ is used to avoid confusion with writing HDL code, and because it describes quite well what actually happens in the phase.

The configuration process is physically somewhat different depending on the FPGA. Some FPGAs only need a personal computer (PC) and a universal serial bus (USB) cable whereas other FPGAs may require special equipment in order to access the configuration.

SRAM-based FPGAs can either be directly configured using the joint test action group (JTAG) pins of the FPGA, or the configuration file can be transferred to an on-board non-volatile memory. The memory is then used to configure the FPGA every time it is powered up. The JTAG configuration method takes only seconds while writing the configuration file into a non-volatile EEPROM or flash memory may take several minutes.

A flash-based FPGA has the non-volatile memory built-in, so the configuration file is simply used to configure the internal flash transistors, i.e. the switchboxes, the CLBs, and the I/O blocks according to the design. Therefore, the configuration of a flash-based FPGA also takes minutes like the configuration of the non-volatile memory used with SRAM-based FPGAs. The flash FPGA needs to be erased before it can be re-configured, and the erasing may take even longer than the actual configuration.

The switchboxes of antifuse-based FPGAs are “burned” according to the burn list. Separating each interconnection, there are two conductors originally kept apart by a thin layer of amorphous silicon between them. When a high voltage is applied to it, the silicon turns into a conducting poly-crystalline silicon metal alloy thus letting the current flow through (Ranta, 2012).

After the configuration is completed, the CLBs, I/O blocks, and switchboxes should be configured according to the design. After this is completed, the FPGA should automatically begin to execute the specified functionality when it is powered up.

Integration

Integration is the final phase in which the FPGA is connected to other devices. If there is no circuit board to which the FPGA is already connected, this phase also includes the circuit board design and fabrication, and the mounting of the FPGA on the board. The FPGA circuit board is placed for example in a computer cabinet or a rack among other instrumentation and control (I&C) equipment. Or, depending on the application, it can be installed for example inside a car or a mobile phone. From this point on, the FPGA design can be seen as blending into the overall I&C system design.

2.4 Verification and Validation

Since the FPGA application is susceptible to errors in all design phases, the design needs to go through a V&V process. The purpose of V&V is to show that the products of each design phase meet their requirements and fulfil their intended use and purpose. Verification is a process where it is determined if the product of a design phase fulfils the requirements imposed by a previous design phase. Validation is a process where it is determined if the product of the whole design process is correct and fulfils the original intended purpose. (IAEA, 1999)

However, in the industry, the use and definitions of the terms ‘verification’ and ‘validation’ vary, and in this text the term ‘V&V’ is used as a single entity although some of the phases could contain only verification or validation. The semantic meaning of ‘V&V’ in this document is the process of acquiring evidence of the correctness or incorrectness of a given application or design through various methods.

Since there is no one definitive guide to FPGA V&V, especially for NPP safety systems, the activities described in this section are collected from various sources. Many activities especially in the early stages of development are very close to or the same as those used in software development. Toward the end, the V&V activities are more hardware oriented and FPGA specific. It has been argued that the lesser complexity of FPGAs makes V&V easier than software V&V (NRC, 2009a; EPRI 2009). However, the additional design phases specific to FPGAs impose new V&V requirements that require knowledge of the hardware aspects similarly to the design process.

Overall, the V&V methods can be roughly divided into three categories: *reviews, testing,* and *formal verification*. Many of the V&V phases employ methods from more than one of these categories. In this section, the three categories are first briefly described after which the V&V activities related to each design phase are described in more detail.

Review

A review is a semi-formal gathering where the designer presents the document to an audience that consists of e.g. other designers and/or independent auditors. The document is walked through step by step, and the audience, already familiar with the document, should question all aspects of it and try to find errors. In the end, corrections and additions to the design document are made. (IAEA, 1999)

Traceability analysis may be conducted during a review. It aims to verify that the properties of a product of a given design phase are traceable back to the requirements imposed by the preceding design phases. Also the requirements should be traceable to the products of the design phases in order to verify that all requirements have been met.

For example, a duplicated logic block in the RTL model may first be traced back to detailed design, where a decision to duplicate the block was made. Architectural design document mentions that some form of redundancy is required in the module. The redundancy requirement may then be traced back to the system requirements. Thus, the duplicated block in the RTL model has been traced back to the requirements.

Testing

Testing refers to the verification of the functionality of the design through different methods that are based on running the system in some way. Testing can be divided into static and dynamic testing. The difference between these is that static testing is challenging the design without executing its functions, e.g. program code, and dynamic testing refers to challenging the design through execution of its functions (EPRI, 1995).

Static testing includes different kinds of analyses of the source code. Review-like activities may sometimes be regarded as part of static testing (EPRI, 1995). For software, there are methods that test such properties as logical correctness, performance and timing, different failure modes, and interface and data transfer properties. Many of the methods are most likely applicable to FPGA designs especially in the behavioural description phase, since the product of that phase resembles a low complexity software source code.

Dynamic testing of FPGAs can be divided into two parts: simulation and hardware verification. Simulation enables the testing of arbitrary input combinations that might be non-testable on actual hardware. These combinations are necessary to test because during failures, some inputs may get values that are extremely improbable during

normal operation. Simulation can be performed against the requirements or statistically by using random-generated input vectors.

Dynamic timing analysis is also performed during simulation in the later design phases. Timing analysis refers to determining the propagation and execution delays of the signals and functions of the design (Ranta, 2012).

Simulation also provides almost unlimited observability of the design. Observability determines how much of the FPGA's internal functionality can be observed during operation. During the operation of FPGAs, only the I/O ports assigned to pins that are accessible can be monitored, which means that observability is very limited. (EPRI, 2009)

During simulation, any of the internal signals can be observed in the simulation tool. For small designs and when appropriate I/O devices are present, it is possible in the hardware implementation to assign some internal signals to observable I/O ports. This may be useful at least when testing parts of the design on the hardware. The limiting factor is most likely the amount of observable I/O pins and monitoring equipment.

Hardware testing is done after the FPGA is configured. The methods for hardware testing depend on the platform on which the FPGA is mounted. For example, some development kits offer built-in switches, LEDs, and screens that can be used to give inputs and observe the outputs of the FPGA (Altera, 2006; Microsemi, 2012a). Typically such a development kit also provides an interface for serial communication, which enables the communication between a PC and the FPGA. This enables the use of the same inputs in hardware testing that were used in simulation.

Formal verification

Formal verification is based on mathematically verifying different aspects of a design in the different design phases. Formal verification requires a model of the system, special tools and skills, and properly formalised requirements against which the properties of the design can be compared. Formal methods for FPGAs can be used to both validate the model of a system against the requirements, and to verify that no unintentional changes to the design have occurred after certain design phases.

Model checking is a formal verification method where all possible behaviours caused by all possible inputs of a model of the system are checked (Clarke *et al.*, 1999). Model checking is not a brute force method such as simulation, but instead based on exhaustively looking for certain properties in a tree-type data structure. The model is checked against the formalised requirements, and if the model's behaviour violates the specification, the model checker gives a counter-example that shows how and why the specified property was violated. Model checking originates from hardware verification where it first proved to be effective for verifying large and complex ICs such as microprocessors (Burch *et al.*, 1992; Fix, 2008).

Logic equivalence checking (LEC) is another type of a formal verification method related to FPGAs. LEC can be done after synthesis and place and route using special software tools to verify that the logic has not changed as a result of the optimisations and modifications during these phases. Where model checking verifies the correct functioning, equivalence checking only verifies the equivalence of the design in different phases i.e. it does not reveal design errors. (Simpson, 2010)

The activities related to different V&V phases in Fig. 2 are listed in Table 1. In the table, each row represents activities related to a particular V&V phase, and each column represents one of the three approaches described above. Explanations of each V&V phase and the related activities are then given in the following text.

Table 1: The V&V activities and methods related to different V&V phases organised in three categories

	Review	**Formal Verification**	**Testing**
Requirements V&V	Requirements review, Formalised requirements review		
Architecture V&V	Architecture review		Simulation
Detailed Design V&V	Logic circuit review, Hardware specification review	Model checking	Logic Circuit Simulation
Behaviour V&V	HDL code review	Model checking	RTL simulation, static analyses
Synthesis V&V	EDIF netlist review, schematic diagram review	Model checking, logic equivalence checking	Gate-level simulation
Place and Route V&V	Physical layout model review	Model checking, logic equivalence checking	Technology-level simulation, static timing analysis
Configuration V&V			FPGA testing, verification of configuration
Integration V&V			Operational tests, circuit board tests

The activities in Table 1 are merely the ones identified so far in various sources. Additional activities and methods most likely exist and are extensively in use. However, using the methods in the table already produces a good amount of evidence of the correctness or incorrectness of the design under V&V.

Requirements V&V

Requirements V&V aims to determine if the requirements specification completely and correctly describes the system and that the specified system satisfies its purpose (EPRI, 1995). Many errors in finished applications can be traced back to requirements, i.e. the application may meet the requirements but the requirements are erroneous and thus the application is too. The requirements specification is usually a textual document which means that the main method for requirements V&V is requirements review.

In requirements review, the requirements are also verified against the characteristics of a good requirement that were presented in Section 2.3. The verification of traceability is also known as ‘traceability analysis’, and since it is an important part of the

verification, it is often regarded as a separate V&V activity from the review. If the requirements were formalised using for example PSL or LTL, they should be reviewed as well. Their compliance with the natural language requirements should be verified.

Architecture V&V

Architecture V&V deals with the results of the architectural design phase. The results of the design phase are the architecture document and some form of diagram illustrating the designed system architecture. The architecture document should be reviewed, and the design choices that were made should be traced back to the requirements. The review is important, since errors found in these early phases are again much cheaper to correct than errors found later on due to the amount of work done. The architecture diagram is reviewed and its contents are traced back to the architecture document. Functional simulation may also be possible. However, the architectural design may be of too high abstraction level, which would make simulation using a software tool quite difficult.

Detailed design V&V

Detailed design V&V is related to the detailed design phase. A detailed design document defines the modules in the architectural design document. In addition to defining the logic, some hardware-related issues need to be considered. Therefore, two kinds of things need to be verified in this phase: the correctness of the designed logic and the accuracy of the hardware-related characteristics.

The verification of the logic may include a review, functional simulation, and model checking. Typically the detailed design logic circuit diagram consists of simpler elementary blocks than the architectural design diagram which should make simulation in detailed design V&V more viable than in architectural V&V. There exist third party software tools such as the Simulink extension of Matlab that can be used for simulating basic logic circuit diagrams (Mathworks, 2012).

Model checking can be used for function block-based designs such as the detailed design diagram. Models of such designs are easy to build as the higher-level functions of the designs are implicitly defined by combining generic low-level blocks in a certain way. The functions of the low-level blocks are very easy to implement.

The hardware-related characteristics can be verified against the datasheet or any other document that contains the necessary information about the FPGA device. It should contain information about the FPGA resources such as amount of CLBs, I/O pins, clock frequencies, voltage levels, and timing properties. It should be verified that it is possible to implement the design in the specific FPGA in terms of the hardware capabilities.

Behavioural description V&V

In behavioural description V&V, the results of the behavioural description design phase are gone through. The HDL code resulting from behavioural description is very similar to regular software source code. The difference is that HDL code is typically less complex than software code, since only subsets of HDLs are synthesisable. Similar reviews as in software development are therefore viable for HDL code as well. It may even be easier due to the lesser complexity.

Many of the same testing methods that are used in software code can be used for HDL code as well. Static testing methods such as flow graph analyses and timing analyses are suitable for HDL code (Balboni *et al.*, 1994). Simulation is an important method for verifying the correctness of the RTL model. For simulation, an additional test bench needs to be created for example by using an HDL. An instance of the design HDL module of the system is created inside the test bench module. The test bench can then assign any signals to the I/O ports of the system, and thus emulate an operating environment.

One challenge is to create a test bench with enough coverage of the functionality of the system. For simple designs, it may be possible to cover all the states through all possible input combinations, but usually the amount of states is so vast that simulating all of them would take too much time. The test bench input vectors should derive from the requirements. If the formal design property requirements were augmented into the HDL code in PSL format, the simulator could make use of the assertions that are generated if the system's behaviour violates a property.

These same assertions are used in model checking. The tool used to do model checking does not execute the code but instead mathematically examines all possible behaviours of the system model and verifies them against formalised requirements. In recent years, the use of model checking has been studied in relation to safety-critical software verification (Lahtinen *et al.*, 2012). One commercial model checking tool that has been used in NPP related projects is IBM's RuleBase (Daumas *et al.*, 2012) but there are also good experiences of open source tools, such as NuSMV (Cavada *et al.*, 2010; Björkman *et al.*, 2009).

Synthesis V&V

Synthesis V&V deals with the results of the synthesis design phase. Both products of synthesis, the EDIF netlist and the schematic diagram, can be subject to reviews. However, an informal inspection by the designer or an EDIF or schematic expert is most likely enough, because for example structural anomalies introduced during synthesis should be revealed by other synthesis verification methods. In addition to conducting the reviews, two things need to be verified after synthesis: the logical equivalence with the RTL model, and the correct behaviour.

In synthesis, the logic is optimised and implemented to function as effectively as possible and to use as little resources as possible. If the designer has for example made certain blocks redundant to increase reliability and mitigate the effect of SEEs (Ranta, 2012), the synthesis tool might falsely interpret the redundant blocks as unnecessary and remove them during synthesis. To verify the equivalence, LEC should be conducted. There are such LEC tools as Conformal LEC by Cadence (Yarom *et al.*, 2006) and FormalPro by Mentor Graphics (Mentor Graphics, 2008).

The second thing to verify in synthesis V&V is that the behaviour has not changed during the automated procedure. It can be done in gate-level simulation and utilising model checking. The gate-level simulation is sometimes referred to as 'post-synthesis simulation' or 'logic-level simulation'. The simulation needs to be done using the exact same inputs that were used when simulating the RTL model in order to correctly verify the behavioural equality of the two products.

To utilise model checking, a model is created based on the synthesised gate-level schematic diagram. The same formalised requirements can then be checked using the new model. The EDIF netlist might also be used to create a model for model checking because it is a formal representation of the design.

In synthesis, some FPGA resources are taken into account, which may cause some additional delays or other hardware imposed issues to show up in the simulation results. At this phase, the delays are not yet necessarily realistic but they may cause the RTL simulation and gate-level simulation results to differ. The possible differences should be analysed and their effect on the design should be determined and documented (NRC, 2010a).

Place and route V&V

The activities related to place and route V&V are similar to the ones in synthesis V&V. The netlist generated in PAR is very low-level which might make reviewing it difficult. The configuration file is also out of the scope of reviews, since its format is not intended to be graphically presented. The physical layout model however can be reviewed at least by the designer especially if there are some special requirements regarding the placement of the CLBs.

After PAR, the hardware-related aspects are realistic, which means that they can be verified against the detailed design document that specifies for example pin assignments, voltages, etc. The timing properties can be verified through technology-level simulation of the low-level netlist. Technology-level simulation is sometimes referred to as 'post-layout simulation'.

The simulation needs to be done using the same inputs as in the gate-level simulation in synthesis V&V. These inputs were the same that were used in the RTL simulation. Doing this helps to ensure that the behaviour of the design has not changed during the several conversions of the design. It might be possible to create a model based on the physical layout model. The model could then be used in model checking.

In addition to the simulation, the LEC performed in synthesis V&V can similarly be performed after PAR. Its purpose is also to verify that the design has not changed as a result of the PAR procedure. If there have been ECOs, their effects on the design need to be determined and documented.

One activity specific to PAR verification is static timing analysis (STA). STA utilises a tool to cover all the signal paths in the design to verify their timing properties. Generally, the propagation delays through wirings and CLBs configured in a certain way are known, and with the physical layout available, the delays can be analytically calculated. STA can be performed to the EDIF netlist resulting from synthesis but, as mentioned before, the delay values in the EDIF netlist are often non-realistic. (Ranta, 2012)

Configuration V&V

In configuration V&V, after the FPGA has been configured according to the configuration file, its correct behaviour needs to be verified. Errors may be introduced during the configuration process. The FPGA testing is conducted using hardware-generated inputs whose values need to be the same as in all the simulations in the previous phases (NRC, 2010a). Differences between the results of the technology-level simulation and FPGA testing indicate errors in the configuration process.

There are methods to verify the correctness of these processes. In some cases, the configuration of an FPGA can be read back from the device (Altera, 2006). These features may not be included in all types of FPGAs. The read back can also be disabled for security reasons (Microsemi, 2012b).

Integration V&V

In integration V&V, when the FPGA has been mounted on the circuit board and placed into its operating environment, the correct functioning is validated against the requirements. The input vectors do not need to be exactly the same as in the previous tests but it needs to cover the expected operation inputs as well as unexpected operation inputs in order to verify that there are no unexpected responses (NRC, 2010a). If the inputs are generated by the actual operating environment, e.g. sensors or buttons, arbitrary input values, which may be included in the previous input vectors, may not even be possible to produce.

The aim of the integration V&V is also to verify that all the peripheral devices on the circuit board function correctly with the FPGA. These include external memory elements, analogue to digital (A/D) and digital to analogue (D/A) converters, power sources, I/O buffers, etc. The detailed testing of the peripheral devices is out of the scope of this document. After this phase, the V&V activities are more related to the overall instrumentation and control system V&V.

3 Nuclear Power Plant Safety Automation Applications

This chapter concentrates on the FPGA-based NPP safety automation applications. The chapter is organised in six sections. A short overview is first provided in Section 3.1. Then, the essential NPP safety functions and design principles relevant to the FPGA technology are briefly described in Section 3.2. Challenges related to the use of conventional microprocessor-based software technology in NPP safety automation are elaborated in Section 3.3. In Section 3.4, the perceived advantages and disadvantages related to the use of the FPGA technology in NPP safety automation are described. A comparison of the features of the FPGA and its main rival, the microprocessor-based software technology, is given in Section 3.5. Finally, some examples of actual NPP safety automation applications incorporating FPGAs are given in Section 3.6.

3.1 Overview

The interest toward the use of the FPGA technology in NPP safety automation is rising internationally as the various advantages of the FPGAs over the currently used microprocessor-based software technology are being recognised. The typically high complexity and the difficulties in demonstrating the safety of traditional software-based applications are among the reasons that have led the nuclear power utilities and I&C system developers to look for new technologies such as the FPGA for implementing the safety automation applications.

Although rather new in the nuclear power industry, FPGAs are already in use in the safety automation systems of several NPPs around the world. One way to use an FPGA is in a modernising project to replace obsolete technology such as a microprocessor no longer available in the market (Druilhe, 2010). FPGA-based systems can also be used as diverse backups for software-based primary systems, but there are also examples of the entire I&C platforms implemented using FPGAs (EPRI, 2009).

The rigid licensing process and demanding requirements related to NPP safety automation and the lack of internationally consistent regulatory guidance and standards regarding FPGAs in NPPs have hindered the widespread adoption of FPGA-based in NPP safety automation. As there is still rather little experience of operational FPGA-based systems in NPPs, the basis for writing consistent regulatory guidance regarding FPGAs in NPP safety automation is not yet strong. However, work is being done to come up with internationally consistent standards and guidance that address the design and V&V issues specific to the FPGA technology.

To promote the technology, the International Atomic Energy Agency (IAEA) holds annual workshops where regulators, system vendors, power utilities, and researchers from different countries come together to discuss different aspects of the development and use of the FPGA technology in NPPs. The IAEA is currently working toward publishing a document addressing different aspects of the FPGA design and V&V (IAEA, 2013). One goal of the workshops has been to determine what the publication should contain.

The FPGA technology itself is quite mature, and has been used in other fields in several safety- and mission-critical applications for many years. It is only a matter of refining the design and V&V methods to fulfil the demanding requirements of the nuclear power industry and getting more operational experience through more implemented systems before the technology can make its final breakthrough.

3.2 Essentials of NPP Safety Automation

This section briefly describes what the main tasks of the NPP safety automation applications relevant to the use of FPGAs are, and what the basic design principles are. In a nuclear power plant, the three most important functions to ensure safety are the shutdown of the reactor, the removal of the residual heat from the core, and the confinement of radioactivity (IAEA, 2002). During normal operation, the process control systems are able to perform these functions. If the process control systems fail to perform the functions, certain dedicated safety systems take control of the situation.

The safety systems consist of the actuating devices that physically perform the functions, and the safety automation system, commonly referred to as the reactor protection system, which detects deviations from normal operating states and forms the signals necessary for the actuators to initiate the execution of the safety functions (IAEA, 2002).

The safety automation functions are typically simple. For example, one of the most important safety systems is the reactor shutdown system that receives measurement signals of the reactor variables such as reactor power, coolant water temperature and reactor pressure. The system compares the values of the signals to the threshold values, and then either remains inactive, or forms and sends initiating signals to the control rod actuators and other systems to initiate the shutdown.

The safety requirements in the nuclear power industry are among the most demanding of all industries. To improve the safety and reliability of the systems, it is often required that they incorporate features such as redundancy and diversity. An important requirement often met by using these features is that no failure of a single system should lead to accident conditions (IAEA, 2002). A failure of multiple systems due to the same cause is called a common cause failure. The mitigation of common cause failures is a key design principle in NPP safety automation, and systems that are independent, redundant, and diverse are typically quite resistant to common cause failures.

Redundancy refers to implementing a function using multiple identical or diverse systems that can each perform the same function independent of the others (IAEA, 2007). Common types of redundancy are e.g. two-out-of-four (2oo4) redundancy and triple-modular redundancy (TMR). For example, a protection system can use four sensors and a 2oo4 voting to determine if a certain pressure is too high or too low. An illustration of a 2oo4-redundant system is in Fig. 7.

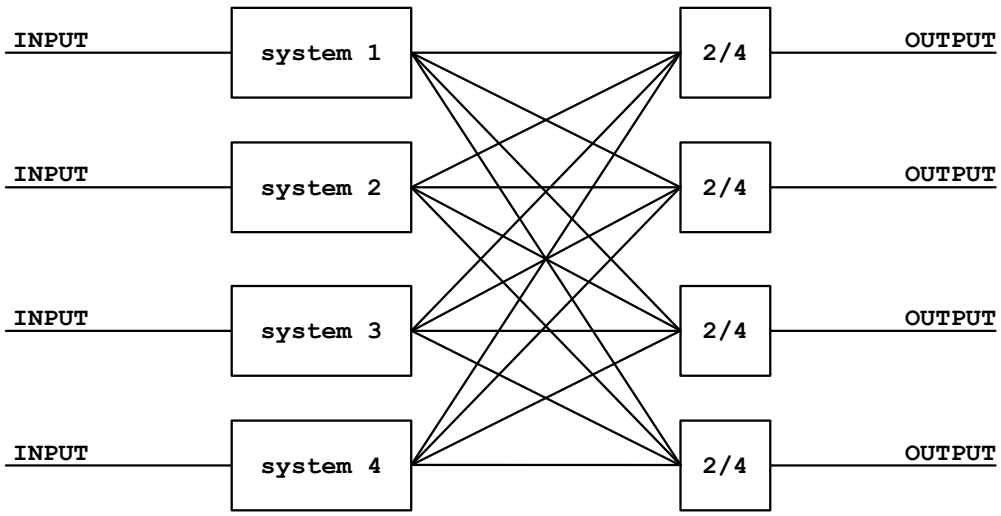

Figure 7: An illustration of a two-out-of-four redundant system

The systems 1, 2, 3, and 4 in Fig. 7 perform the same functions based on the redundant inputs. The output of each system is conveyed to a 2oo4-voter. Each voter thus determines the correct output and the four redundant OUTPUT-signals should be identical if everything functions correctly. The OUTPUT-signals can then be used e.g. as inputs for other 2oo4 redundant systems.

Diversity is usually required in redundant systems. Employing diversity is a good way to increase the independence between systems through e.g. monitoring different process parameters, using different technology, logic, algorithms or means of actuation in order to provide several different methods to get information from the process and to control it (IAEA, 2002).

Diversity can also be applied to humans by e.g. having different teams design the redundant systems. Diversity is an effective means against common cause failures. Sufficiently diverse systems are less likely to have the same errors or to experience a failure due to the malfunction of e.g. a certain power source or a sensor of a particular manufacturer than systems using the same components.

3.3 Software-based digital technology

Digital technology has been used in NPP I&C systems since the 1980s (EPRI, 2009). Old analogue systems were replaced by microcontrollers, programmable logic controllers and distributed control systems, which all are based on microprocessors running some form of software (EPRI, 2009). Digital technology was initially used only in systems such as data collection systems and information displays that are not important to safety. Since then, they have also been widely adopted in systems such as reactor control and monitoring systems, turbine automation, and emergency diesel generators and they are becoming increasingly common in the most critical safety systems as well (IAEA, 2000).

Digital technology offers numerous advantages compared to traditional analogue technology. These include easier implementation of complex functionality such as improved process monitoring, interface, diagnostics, and testing. Digital systems are considered more accurate, stable, and flexible than traditional analogue hardware-based systems. They are also said to offer economic advantages over traditional analogue systems since less hardware is needed to implement a complex function using digital programmable technology. (IAEA, 2000)

One difficulty of using software-based digital systems to implement NPP safety systems is related to demonstrating their safety for licensing. Where traditional hardware failures are usually considered random and occur due to aging, wear, stress, etc., the nature of software failures is considered deterministic instead. Software does not age or become worn but it can still fail due to errors introduced during the design. Also, the hardware running the software is subject to the same failures as any other piece of hardware. To verify that the design does not contain errors, proper V&V processes are necessary during the design of software applications.

The software V&V is difficult due to the complex nature of the applications. Using software, there is a risk that the applications easily become unnecessarily complex. Traditional simulation and testing methods can cover all possible states of only very simple systems. Although safety applications are simple, usually the state space of the software implementations becomes so vast that it would take years to test all possible behaviours through simulation. There are formal methods such as model checking that can be used to mathematically check all the possible behaviours of a system (Clarke *et al.*, 1992). However, the formal methods typically require expertise and special knowledge about the methods. Also, the requirements that are checked need to be formalised and a suitable model of the system needs to be created.

On top of the actual software application algorithms, there is some overhead complexity that results from the need for an operating system to switch between different tasks in the application. Different applications typically run on the same microprocessor that can only execute one task at a time. This makes truly separating the actual safety functions and the supporting functions from each other difficult or even impossible in software-based systems. (EPRI, 2009)

One solution could be to use multicore microprocessors. The important functions could be assigned to a dedicated CPU and would therefore be independent from the other functions. However, multicore microprocessor applications are considered to be even more complex than traditional single core applications which may lessen the advantages of having independent execution of certain functions. Still, the technology is constantly evolving and in the near future, there may be modern multicore processors that are tailored for use in safety-critical systems. (Ranta, 2013)

As the development of the microprocessor technology is not driven by the safety critical industry but instead the consumer market, the characteristics of modern microprocessors are not optimal for safety-critical applications. They are typically very complex, contain many functions not needed in safety-critical applications, and their lifetime is too short for the decades of operation required in an NPP. The safety functions need to share the same resources with the support functions on the microprocessor, which makes deterministic demonstration of the correctness of the safety application difficult (Arndt *et al.*, 2012).

The difficulties with designing and implementing the digital automation applications have led to delays in some NPP construction projects. For example, in the Finnish Olkiluoto 3 NPP currently under construction, there have been problems with the independence between the process control and the safety automation systems (STUK, 2009).

3.4 Use of the FPGA technology – advantages and disadvantages

FPGAs have been on the market for nearly thirty years, and in the past ten years, they have become increasingly advanced and capable of implementing complex functionality. Thus, they are now seen as a viable alternative for implementing safety-critical NPP safety automation applications. The use of the FPGA technology has also some disadvantages that are mainly related to the complex design process and the novelty of the technology in the nuclear power industry. This section describes how the FPGA technology can be used in NPP safety automation and what are the related advantages and disadvantages.

FPGAs are capable of performing practically any logic functionality with the limiting factor being only the capacity of the device i.e. how many functions can be implemented using the CLBs of the device. Therefore, FPGAs can be used in any NPP system that involves logic functions, e.g. in protection systems, actuation decision systems, and communication systems (NRC, 2010a).

In the current regulations and standards, there are some inconsistencies with issues such as what even constitutes an FPGA. In some documents, the FPGA is considered to be e.g. a programmable logic device (PLD), or a complex programmable logic device (CPLD), both of which are similar to but nonetheless different from the FPGA. Issues such as this have prevented the FPGA-based system vendors and developers from fully understanding the regulatory requirements in a way that would enable them to produce applications that meet the regulations (Arndt *et al.*, 2012). This probably has in its part hindered the adoption of the technology in the nuclear power industry.

In 2012, the IEC published the standard 62566 entitled "Nuclear power plants – Instrumentation and control important to safety – Development of HDL-programmed integrated circuits for systems performing category A functions" (IEC, 2012). The title suggests that the standard provides comprehensive development guidance for FPGA applications (as they are HDL-programmed integrated circuits).

However, according to the discussions at the Fifth International Workshop on the Application of FPGAs in Nuclear Power Plants held in Beijing in 2012, the IEC standard 62566 should only be used as an informative reference. The contents of the standard are not as comprehensive as was initially expected, and in many parts, there are just references to other e.g. software-related standards.

Typically, an NPP safety automation application deals with concurrent measurement and data processing tasks because there are many subsystems such as sensors transmitting information simultaneously. There are many functions that need to be independent of each other and executed concurrently. As FPGAs operate naturally concurrently, and all of the functions can be executed simultaneously and independently, they are suitable for implementing these kinds of NPP safety automation applications.

The performance of FPGAs far exceeds the requirements for NPP safety automation applications. Typically, the functions need to be executed within tens of milliseconds or even more (EPRI, 2009). FPGAs typically operate at clock frequencies of tens or hundreds of megahertz which, accompanied with the naturally concurrent processing of FPGAs, means that the response time requirements are easily met.

FPGA technology can be made resistant to obsolescence by designing the applications in an appropriate way. If the design has been implemented using e.g. VHDL, it should be device independent and thus easily portable to other FPGAs. This is an effective measure against obsolescence because when the time comes to replace the

aged and worn FPGA, it can most likely be easily replaced with a currently available device (EPRI, 2009).

An FPGA can be used to implement even software-based applications if necessary. For example, if a microprocessor that is no longer available on the market is needed, it can be emulated on an FPGA by using an IP core that is a predefined block implementing the functionality of e.g. an entire microprocessor (Daumas, 2012). Then, using the emulated processor, software applications can be run on the FPGA.

The fact that so many functionalities can be implemented inside a single FPGA chip means that the use FPGAs can significantly reduce the amount of different hardware needed to implement functions (EPRI, 2009). There are many ready-made IP cores that can be used for implementing e.g. digital signal processing, finite impulse response filters, or Fast Fourier Transform calculation that may be needed in NPP safety automation applications (NRC, 2009a).

FPGAs are suitable for implementing features that conform to the basic design principles of NPP safety automation applications. The principles include redundancy, diversity, and independence as described in Section 3.2. As the FPGA is naturally modular, redundancy can be easily implemented on various levels. Individual logic blocks, larger modules, or even the complete logic can be made redundant within the same chip. The fact that the functions of an FPGA are independent means that applying redundancy inside the chip is meaningful since the disturbances caused by e.g. radiation may only affect certain CLBs instead of the entire chip.

In addition to the actual safety functions, there are also supporting features such as communication and data processing in NPP safety automation applications. In the nuclear industry, a common requirement is that the safety features should be separated from the supporting features to achieve independence of the functions. Each CLB in an FPGA is implemented in physically separate transistors and is thus independent from the others (EPRI, 2009). This means that separation of the functions from each other is not only possible but also quite effortless when using FPGAs.

The use of the FPGA technology as a diverse backup system alongside a microprocessor-based primary system is one of the common applications of the FPGA technology in NPP safety automation. The different structure and operating principle of an FPGA bring physical and functional diversity into the design.

However, since the current regulations regarding diversity do not offer specific guidance on this subject, it is unclear what degree of diversity the FPGA offers for an NPP safety automation application (Arndt *et al.*, 2012). Additionally, the software-like design process is susceptible to similar errors as software design which may affect the degree of diversity compared to software. Furthermore, the limited amount of available software tools may also be harmful to diversity because even if the application was created using different language and different designers, if the same software tool is used in the process, similar errors might be introduced into the design (Arndt *et al.*, 2012).

The FPGA manufacturers all have proprietary tools for developing applications for their FPGAs. As the development of the tools is not driven by the safety-critical requirements and the requirements for NPP safety automation applications are demanding, each product of each design phase where a software tool is used needs to be verified to detect errors possibly introduced during the use of the tool.

The verification of the tools is difficult because the outputs of a tool typically cannot be compared to the outputs of another tool because of the tool-dependence of the device (Arndt *et al.*, 2012). Based on the discussions at the Fifth International Workshop on the Application of FPGAs in NPPs, the need to verify the software tools

is one of the issues related to the use of the FPGA technology in NPP safety automation applications.

Even though the verification of the software tools used in the FPGA design process is difficult, there are many good methods for the V&V of the resulting FPGA application. Where the safety justification has traditionally been a challenge with digital software-based technology, the lower complexity of the FPGA applications is said to make the justification easier by enabling more comprehensive and simpler V&V methods for providing evidence of the correctness of the application. Even though the design process is rather complex, the design products can be comprehensively verified after each phase of the process.

It is argued in (EPRI, 2009) that the observability of the FPGA designs may be limited i.e. all the states and signals in the system may not be accessible for observation. However, provided there are suitable I/O devices, practically any internal signal can be assigned to an output port that can be observed. The size of the design and the number of output ports that can be observed seem to be the only limiting factors in this practice.

Because many NPP safety automation functions involving timed sequences operate in the second-scale and clocks of FPGAs operate in the nanosecond-scale, the realistic simulation of the designs may be difficult. In the simulation, the clock signal needs to be generated by implementing a binary variable that changes its value e.g. every few nanoseconds. When simulating a multiple-second operation cycle of the application, it takes a very long time to even get through the first second of operation. The simulation may thus need to be done using a much slower clock frequency which may affect the integrity of the simulation as a verification method.

An easy solution for this problem could be to use a significantly slower hardware clock with the FPGA. The problems regarding the large clock frequency would be avoided. However, as the NPP safety automation applications should be synchronous, in which case the signals propagate between the blocks only when there is a rising edge on the clock signal, a significantly slower clock could cause problems with meeting the response time requirements.

The security of an FPGA during the operation is good because it has a simple structure and unlike in software, there are no operating systems or general purpose functions that would be easy to hijack by a malicious application or an attack. Also, because the re-configuration of an FPGA can be made difficult or even permanently impossible (with antifuse-based FPGAs and some flash-based FPGAs), the security grows even better (EPRI, 2009). However, when emulating a microprocessor on an FPGA, the similar security issues apply as with software.

The main challenges regarding security seem to be related to the software tool-dependent design process (Arndt *et al.*, 2012). As the outputs of the tools are difficult to verify, unauthorised alterations to the design are possible if good informational security practices are not employed during the design process.

The physical lifetime of an FPGA device is also something that needs to be considered in NPP applications. Typically in mobile phones or other consumer electronics applications where FPGAs are used, the expected lifetime of a product may only be a few years whereas NPPs may be intended to be operated for as long as fifty years. Although modernisations are inevitable at some point in the plant's lifetime, the I&C system could still expected to be operational for tens of years. It has been reported that e.g. the contemporary trend of using lead-free soldering may cause problems with the longevity of the connectors and configuration in prolonged use because a phenomenon called whisker growth can affect the solders and cause failures (NRC, 2010a).

In some applications in NPPs, it may be necessary for the FPGAs to be able to withstand radiation. FPGAs can be made radiation resistant, a subject that has been studied in relation to aviation and space flight (Gaisler, 2002), but in many cases it has not been necessary because the I&C cabinets typically reside in areas where the radiation is not intense (EPRI, 2009).

FPGAs using all the three memory technologies, SRAM, flash, and antifuse, have been used in NPP safety automation applications (EPRI, 2009). However, since flash and antifuse are non-volatile and generally more resistant to disturbances, they may be better for safety-critical systems than SRAM that requires special measures to be resistant to disturbances (EPRI, 2009).

The fact that SRAM-based FPGAs are volatile means that they need to be re-configured each time when powering up. This may cause problems if there is a situation where the supply power is cut for a short time and the subsequent re-configuration fails for some reason. It may be difficult to verify that the configuration has been successful if the application is a passive safety function.

The flash and antifuse are technologically one or more generations behind the CMOS technology that the SRAM-based FPGAs are based on. However, their attributes are usually more than adequate for the simple NPP safety automation functions. If additional features such as diagnostics and self-testing are implemented, antifuse-based FPGAs may not have sufficient capacity (EPRI, 2009). The capacity of flash-based FPGAs is constantly improving and is already considered sufficient for these extensive functions. Additionally, the larger physical size of the flash-based FPGA device is generally not an issue in NPPs unlike e.g. in the mobile phone industry.

3.5 Comparison with microprocessors

Because the FPGA technology is mainly seen as a challenger to the microprocessor-based software technology, this section compares the characteristics of FPGAs to microprocessors. An FPGA application is generally less complex than a software application since there is no operating system necessary in an FPGA. With software, an operating system is needed to switch between different tasks that are executed on the CPU. Even the simplest operating system designed specifically for the safety-critical features typically contains tens of thousands of lines of code which, laid on top of the actual application code, increases the complexity overhead of the system. (EPRI, 2009)

Although the product of the FPGA design process and the FPGA application is less complex than its software counterpart, the design process itself has more phases than software design process and is more complex. The FPGA design process is dependent on several software tools whereas the software design process typically requires only the compiler and linker.

With an FPGA it is very easy to separate the safety and the support features in an application. The CLBs are physically separated and independent of each other, meaning that the functions implemented on different CLBs are actually also separated and truly independent.

With software, the execution of the features can also be logically separated within the source code. However, in software all features are executed using the same CPU meaning that the different features need to share the same resources. This means that the features are not completely separated or independent of each other. A multi-core microprocessor might help to resolve this since different CPUs could be dedicated for specific functions. However, the use of multi-core processors is considered to make the software application more complex than when using a single-core processor (Ranta, 2013).

A CPU typically executes one instruction per each clock cycle. Even many simple functions consist of tens or hundreds of instructions, and an operating system is needed to switch between different functions in an application. When using a single CPU, only one task can be executed at a time which may cause concurrency problems such as deadlocks or starvation (Bacon, 1998). The CLBs of an FPGA can be independently driven by the same clock signal to execute all desired functionality simultaneously parallel to each other.

FPGAs have better performance than microprocessors. It may take several tens or hundreds of clock cycles for a microprocessor to complete even a simple function whereas functions of almost any size can be executed during a few clock cycles when using an FPGA. For a microprocessor it takes time to process each piece of the function one by one by running the code line by line whereas an FPGA can execute them in a fraction of the time when they are assigned to different CLBs. (EPRI, 2009)

The lifetime of microprocessors is generally not designed to be decades because the development of microprocessors is mainly driven by the consumer market where the microprocessors are only used for a few years before they are replaced. Even the microprocessors used in industry-oriented programmable logic controllers do not have as long lifetimes as is desired. (EPRI, 2009)

The security issues are considered to be lesser for FPGAs than for software. By using a non-re-programmable FPGA or manually preventing the re-programming, the configuration can be made permanent. Because the FPGA does not execute code, it is considered immune to traditional malicious attacks such as viruses. However, if an FPGA is used to emulate a microprocessor, the security issues need to be addressed in the same way as when using a normal microprocessor. Although the operational security is better with FPGAs, the heavily software tool dependent design process is considered more vulnerable to malicious attacks than software development due to the need of multiple different tools.

The need for an FPGA designer to be familiar with the hardware related aspects of the technology can be seen as a disadvantage when compared to software. A software engineer typically does not need to worry about voltages, pin assignments, and signal timings, or to be familiar with electronics netlists or schematic diagrams. Software experts are normally easy to find whereas experts in both software and hardware are scarcer. However, this is not a big disadvantage in NPP design projects since the apparent benefits of using FPGAs instead of microprocessors probably weigh more than the convenience of the design process.

3.6 Examples of FPGA-based systems

This section contains examples of FPGA-based systems in NPP safety automation. The examples are collected from various sources such as conference proceedings and technical research reports.

The first installation of an FPGA-based safety system called the power range neutron monitor by Toshiba was done in a Japanese NPP in 2007. A power range neutron monitor system receives signals from the neutron flux sensors inside the reactor and calculates the overall neutron flux. The system then processes the information and provides the necessary signals for the protection system and other systems that use the information. (EPRI, 2009)

The Ukrainian Research and Production Company Radiy is one of the most significant developers and manufacturers of FPGA-based I&C platforms for NPPs. One of the features of the platform is the engineered safety features and actuation system (ESFAS). The ESFAS contains various safety-related functions such as automatic

actuation of safeguard equipment and automatic control of actuators according to process control algorithms, and manual actuation based on the signals coming from the control room. In addition, ESFAS performs such support functions as transmission of signals to other systems, and diagnostic, information, and service functions. (EPRI, 2009)

In the Temelin NPP in the Czech Republic, the emergency diesel sequencers have been implemented using FPGA technology. When the normal power is lost, all the devices such as safety systems, valves, and pumps that use power are disconnected from the power bus. The diesel generators are then started and the systems that use power are sequentially connected to the generators output. The devices cannot be all connected to the generator simultaneously because the load spike would be too large. The antifuse-based FPGAs in the sequencers have now been in operation over ten years without problems. (Waage, 2012)

The Finnish Olkiluoto 3 NPP under construction is planned to have an FPGA-based hardwired backup system that is capable of performing some of the key functions of the main safety automation system (NRC, 2009a). The FPGA-based system would act as a diverse backup to the microprocessor-based primary systems in order to lessen the probability of common cause failures related to the microprocessor-based systems (NRC, 2010b).

The French electric utility Électricité de France (EDF) is doing modernisations for its NPPs. The aged Motorola MC6800 microprocessors that are currently used in the I&C systems need to be replaced but the MC6800 is no longer available on the market. Rolls Royce has developed an IP core of the MC6800 for FPGAs. The core is to be implemented on FPGAs and used to replace the obsolete microprocessors. (Druilhe, 2010)

4 Case study: Power Limitation System

A fictional safety automation application called the Power Limitation System (PLS) was implemented using actual FPGA devices in this case study. The PLS is a system that contains multiple subsystems located on different hardware. Inputs are of different priorities and require the outputs of different subsystems. This means that a prioritisation is needed between the subsystems to determine the correct output of the overall system. In the case study, the whole design process was run through starting from writing the requirements specification and ending with a functional two-FPGA system.

The PLS case study is an expansion of a previously conducted case study (Lötjönen, 2012) where a preventive safety function called the Stepwise Shutdown System (SWS) was implemented on an FPGA device. The SWS has previously been used in other studies such as a model checking study (Björkman *et al.*, 2009).

The purpose of the previous FPGA case study was to get hands-on experience on designing FPGA applications. Also, it was interesting to see if a known design error in the SWS would cause a failure in the FPGA design as it did in the model checking study. The FPGA case study proved to be successful, and it was decided that it would be expanded as part of this work.

The main objective of the PLS case study was to get hands-on experience on the different V&V methods related to FPGA designs. Two FPGA devices from different manufacturers were in order to get experience of the design tools of two different manufacturers, and to have a more realistic system through having different functions physically separated on different hardware. Communication between the two FPGAs was established using a cable to find out what kinds of challenges are related to it. A case study with actual FPGAs also further helps illustrate the design and V&V methods described in Chapter 2.

This chapter is organised in five sections. The concept of the PLS is first described in Section 4.1. The hardware used in the case study is then described in Section 4.2. The design process that was followed through is elaborated in Section 4.3 and the V&V process is elaborated in Section 4.4. The case study is then summarised in Section 4.5.

4.1 System description

The PLS is a fictional NPP safety automation function that has three operating modes and a priority logic to switch between the modes. The system has four binary inputs and three binary outputs that are combined as one signal. The concept of the PLS is illustrated in Fig. 8.

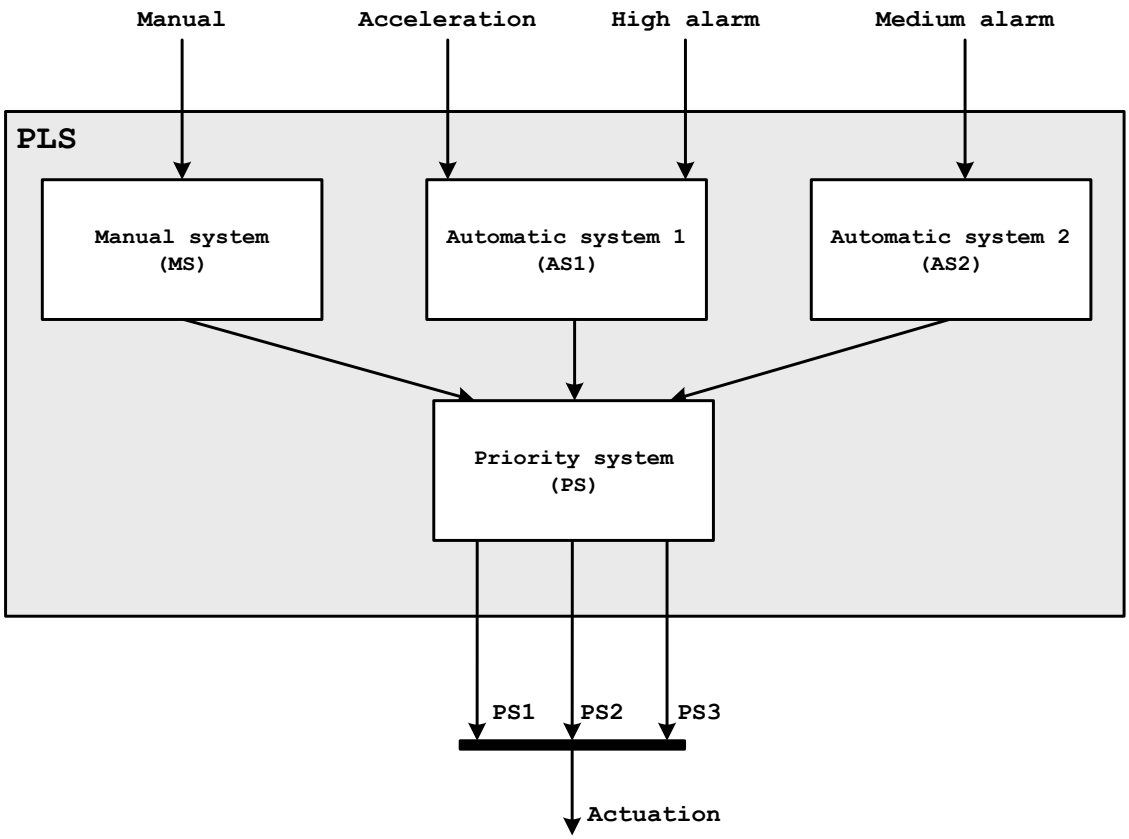

Figure 8: Illustration of the Power Limitation System

The PLS is confined inside the grey box in Fig. 8, and the interface consists of the signal that go in on the top and that come out in bottom of the box. On the top are the inputs: *Manual*, *Acceleration*, *High alarm*, and *Medium alarm*, and on the bottom are the outputs *PS1*, *PS2*, and *PS3* that are combined as one signal, *Actuation*, after the PLS. The signals are combined because they all initiate the actuation in the same way. The three outputs are viewed as separate signals only to make the system more observable making the testing of the system easier.

The three modes are implemented as three separate systems: *Manual system* (MS), *Automatic system 1* (AS1), and *Automatic system 2* (AS2). The three modes are of descending priority with the MS being the highest and the AS2 being the lowest. Therefore, the Priority system (PS) is needed to determine which of the systems gets to drive the Actuation signal based on which inputs are active.

The only task of the system is to drive an imaginary actuator in three different ways according to the current state of the process. The state is indicated by the binary input signals High alarm and Medium alarm that are alarms of different severities. The alarm signals would be coming from another system (not implemented in this case study) that monitors the measurement signals and initiates the appropriate alarms if the measurement signals cross certain thresholds. The Manual input is a signal coming from the pressing of a button in the control room. Similarly, the Acceleration is also activated by the pressing of a button in case the operation of the AS1 needs to be accelerated.

The three different modes MS, AS1, and AS2 do not communicate with each other or affect each other's functions in any way. They are thus independent of each other and are viewed as separate subsystems. The MS simply activates the output whenever the input Manual is active. When the input High alarm is activated, the AS1 starts a six-second operation sequence where the output is first active for two seconds and then

inactive for four seconds. This sequence is repeated if the input High alarm is active at the end of the cycle. Additional two-second output pulses can be initiated by the input Acceleration. The AS2 functions similarly to the AS1 but instead of six seconds, the operation sequence is fifteen seconds with first a three-second output pulse and then a twelve-second idle period. Additional three-second pulses cannot be initiated in the AS2.

The PS takes in the outputs of the three subsystems and lets the one with the highest priority through. For example, if the AS1 and AS2 are both active, only the output of the AS1 will get through because it has a higher priority. Similarly, if the MS is active, its output always gets through because it has the highest priority.

4.2 Hardware

In the case study, two different FPGA devices were used. One was an Altera FPGA and the other was the Actel FPGA that was used previously in the SWS case study (Lötjönen, 2012). FPGAs by both Actel and Altera have been used in actual NPP safety automation applications (NRC, 2009b; EPRI, 2009). Both FPGAs were mounted on development kits that are circuit boards with many I/O and other devices that can be used with the FPGA. A forty-pin parallel ATA cable, commonly used in optical disk drives in regular PCs, was chosen as the interconnecting cable because both development kits had an appropriate connector for it.

Additionally, a serial connection was established between a PC and the Cyclone. Through this connection, the outputs of the systems could be properly processed and recorded since they would be transmitted to the PC rather than just be indicated by blinking light emitting diodes (LEDs) or displays on the development kits. The actual hardware setup of the case study is presented in Section 4.3 when the design process is elaborated.

In Fig. 9, the Cyclone FPGA mounted on the development kit is pictured. Those devices on the development kit that were used are numbered and explained in the following text with a reference to their number.

1. Cyclone II FPGA

2. USB port

3. Configuration memory

4. Switches

5. Buttons

6. 7-segment displays

7. Serial communication port

8. 40-pin parallel port

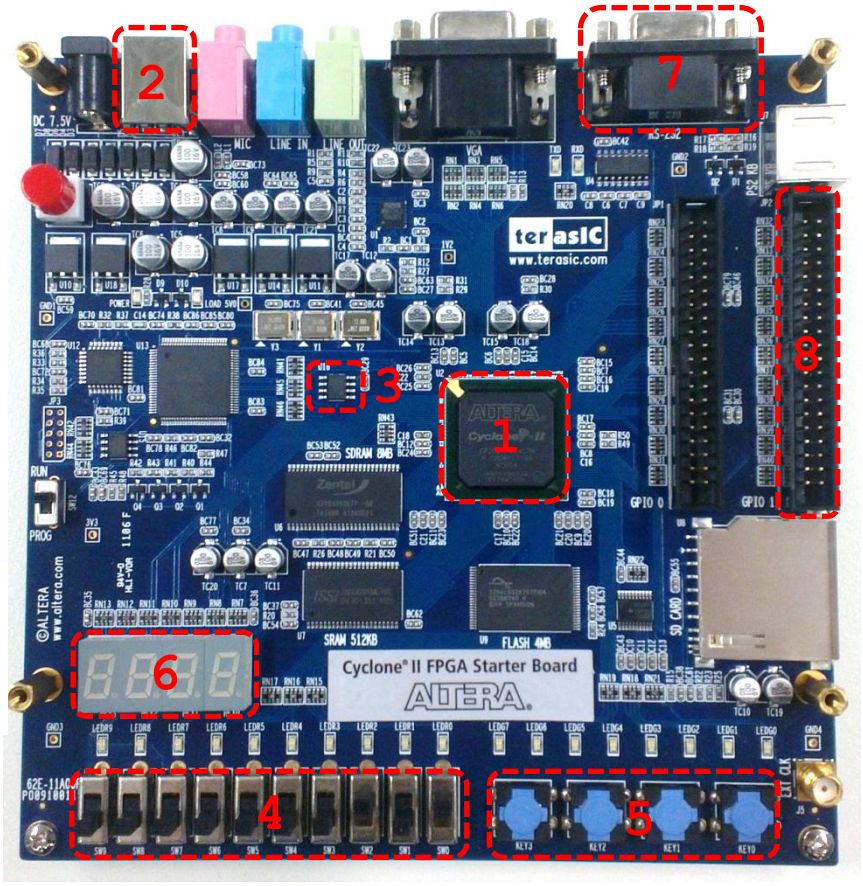

Figure 9: Altera Cyclone II FPGA starter development kit

In Fig. 9, the Altera Cyclone II SRAM-based FPGA (1) is mounted on the development kit with several other devices that could be used with the FPGA. In the development kit, there are a lot of devices such as displays, LEDs, switches, audio and video coder-decoders, several memory chips, a connector for a VGA monitor, and a connector for a mouse or a keyboard that can be used for testing many different kinds of applications (Altera, 2006).

The size of the circuit board of the kit is about 160 mm by 160 mm. The full name of the FPGA on the kit is Cyclone II EP2C20F484C7N FPGA and its size is about 15 mm by 15 mm. The FPGA has 18,752 logic elements (LEs) that are Altera's version of CLBs. There are 484 pins in the bottom of the package that the FPGA is in, and 315 of these are usable by the developer. (Altera, 2006)

The configuration of the Cyclone FPGA is done via a USB cable that connects to a USB port (2) in the development kit. As the FPGA is SRAM-based, the configuration can be done in two ways. The faster way is to directly configure the FPGA through the JTAG pins. As the memory elements of the FPGA are volatile, the configuration done like this is lost when the power is cut. To cope with this, there is also a dedicated non-volatile on-board configuration memory (3) for storing the configuration. Using the same USB cable, the configuration can be transferred to the configuration memory and subsequently each time the FPGA is powered up, it is automatically configured according to the contents of the configuration memory.

The switches (4) and buttons (5) are used to generate the inputs in the PLS case study. The outputs can be read from the 7-segment displays (6) and are also transmitted to a PC using the serial communication port (7). In the end of the PLS case study, the Cyclone FPGA is connected to the other FPGA via a 40-pin parallel cable because both

development kits have appropriate connectors (8) for them. The development kit is powered by wall-socket power supply that provides 7.5 volts and 0.8 amperes of direct current.

The second FPGA is a flash-based Actel IGLOO FPGA. It is mounted on a development kit that, like the Cyclone kit, also has many devices and features of which only a small portion is used in the case study. The IGLOO development kit is in Fig. 10.

1. IGLOO FPGA
2. USB port
3. Switches
4. LEDs
5. 40-pin parallel port

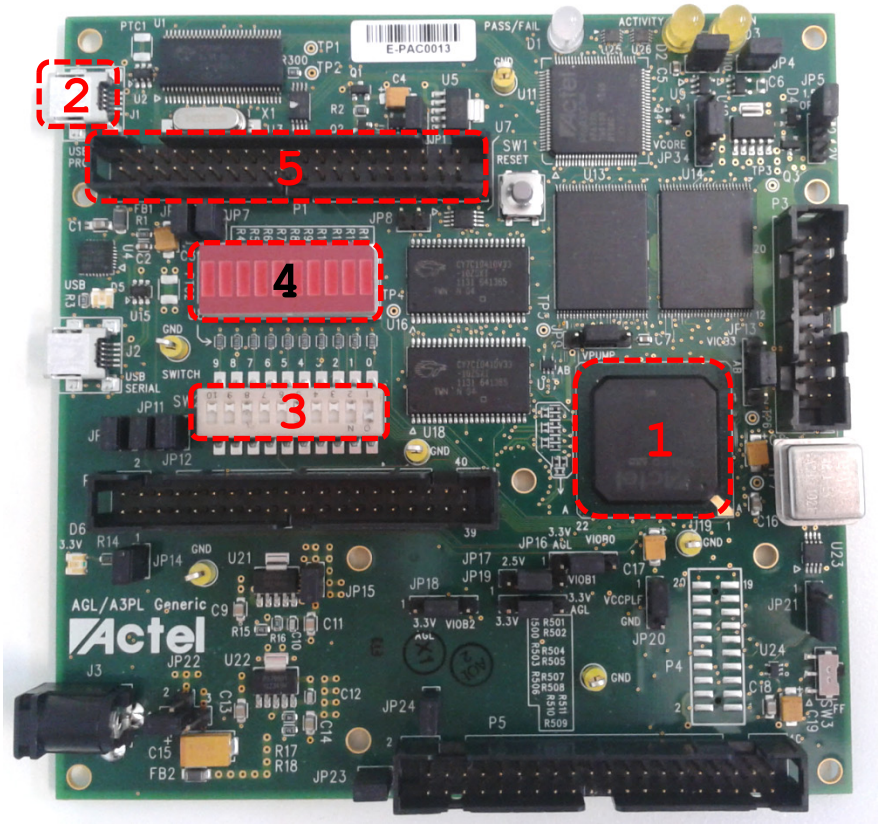

Figure 10: Actel M1AGL1000 IGLOO FPGA development kit

The size of the IGLOO development kit circuit board in Fig. 10 is about 100 mm by 100 mm making it a little smaller than the Cyclone development kit. Similarly to the Cyclone development kit, the devices of the IGLOO development kit used in the case study are numbered in Fig. 10 and explained in the following text.

The whole make and model of the FPGA device is Actel M1AGL1000V2-FGG484 FPGA. It has one million system gates that are organised in 24,576 VersaTiles that are Actel's version of CLBs (Microsemi, 2012a). The capacities of the devices are thus in the same region with the IGLOO having a few thousand CLBs more than the Cyclone. As the IGLOO is a non-volatile flash-based FPGA, no external configuration memory is required. The configuration is done using a USB cable for which there is a connector on the development kit (2). The CLBs and wirings of the FPGA are directly configured according to the configuration file.

The I/O devices on the IGLOO development kit are not as abundant as on the Cyclone development kit. There are ten small switches (3) and ten red LEDs (4) that can be used to generate inputs to and observe the outputs of the system. The IGLOO development kit has several forty-pin parallel cable slots of which one (5) was used for the communication with the Cyclone. The IGLOO development kit is powered by a wall socket power supply that provides five volts and three amperes of direct current.

4.3 Design

This section describes the design process of the PLS. The design process was divided into phases and carried out according to Section 2.3 where the design process was described. The activities and products that were generated during each phase are presented and elaborated in the following text.

As the case study includes FPGAs of different manufacturers, different software tools were used in the development of the applications. For the Cyclone, Altera's Quartus II (Altera, 2011) was used and for the IGLOO, Actel's Libero integrated design environment (IDE) (Actel, 2010) was used. The Quartus II has all of the functions needed in different design phases built-in whereas the Libero is only an environment for managing the design process and starting different software tools needed in different phases.

Requirements specification

The requirements specification was created containing all required features of the PLS. The document is divided in roughly two parts: one that contains the non-functional requirements and one that contains the functional requirements.

The non-functional requirements include e.g. a description the operating environment, system structure, interface, and other features that do not deal with the behaviour of the system directly. In this case, as the system was only intended to be an example for trying out the design and V&V methods, the operating environment and the interface requirements do not mimic a real-life NPP situation.

However, regarding the system structure, it was stated in the requirements specification that the priority system (PS) shall reside on different hardware than the automatic systems. This requirement originates from one of the objectives of the case study that was to get two FPGAs communicating with each other. Also, it is quite common in real systems that the priority logic is implemented separate and independent from the applications such as differently prioritised subsystems that it is communicating with.

Regarding the performance, it was required that an appropriate reaction to an input should happen within one millisecond. For the interface, it was required that the outputs can be observed via a display and through serial communication, and that all I/O signals shall be made fail-safe to achieve some realism.

In the formulation of the functional requirements, the EARS was used. It was easy to formulate the requirements in a way that corresponds to the five EARS requirement sentence types presented in Section 2.3. In the following list, some of the requirements related to the PLS are laid out:

- When *High alarm* is activated, AS1 shall perform the six second operation sequence.
- While *Medium alarm* is inactive, while AS2 is not performing the fifteen-second operation sequence, *AS2 output* shall be inactive.
- While one of the outputs of PS is active, all other outputs of PS shall be inactive.

The first requirement is related to the AS1 and specifies that an activation event in the input High alarm shall initiate the basic operation sequence. The second requirement is

related to the AS2 and specifies that when the initiating criterion Medium alarm is not active and the system is not performing an operation cycle, the output of the system shall be inactive.

Architectural design

The architectural design was quite effortless because the required modules were already specified in the requirements specification. What was left for the architectural design was naming the I/O signals and the modules explicitly in a way that they would eventually be named in the VHDL implementation. The modules of the architectural design are illustrated in Fig. 11.

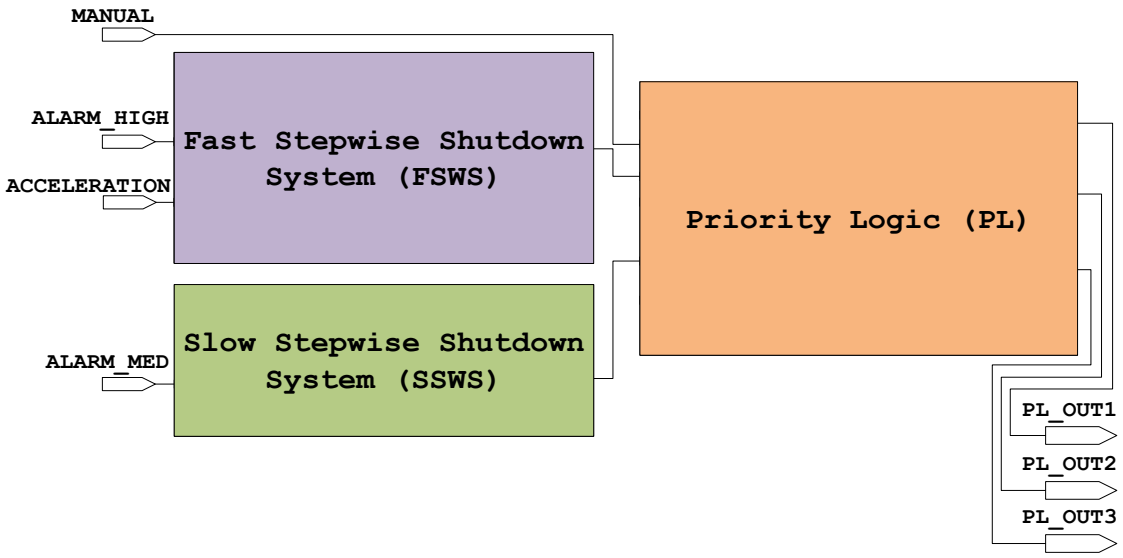

Figure 11: Architecture diagram of the Power Limitation System

The I/O signals in Fig. 11 are named as they will be in the following stages of development. Although it was specified that all drive the same actuator, the system outputs PL_OUT1, PL_OUT2, and PL_OUT3 are combined. This is to make testing easier as the different outputs could now be easily observed.

In the architecture diagram, there are three subsystems instead of the four in the original system description in Fig. 8. This is because the manual system (MS) is only a direct signal from the input MANUAL to the Priority Logic (PL) due to its trivial functionality i.e. because the corresponding output merely follows the value of the MANUAL.

The automatic systems AS1 and AS2 in the system description in Fig. 8 are now named Fast Stepwise Shutdown System (FSWS) and Slow Stepwise Shutdown System (SSWS), respectively. The names come from the previous case study where a system called Stepwise Shutdown System (SWS) was implemented (Lötjönen, 2012). Since the functionalities of the FSWS and the SSWS are similar to functionality of the SWS, they were named accordingly.

The PL is named similarly to the original PS and it needed to be designed from scratch as the previous case study did not contain a prioritisation logic that could be reused in the PLS case study. The next phase was the design of the internal structures of the modules in the detailed design phase.

Detailed design

In the detailed design phase, the internal structures of the modules in the architectural diagram in Fig. 11 were designed. As the FSWS and the SSWS were mere modification of the original SWS, designing their logic gate structure was easy. The PL needed to be designed from the beginning. The logic circuit diagram of the entire PLS is in Fig. 12. The individual modules FSWS, SSWS, and PL in the logic circuit diagram in the figure are examined more closely with better pictures later on in this section. Figure 12 is only supposed to offer a glance at the complete logic circuit diagram of the PLS.

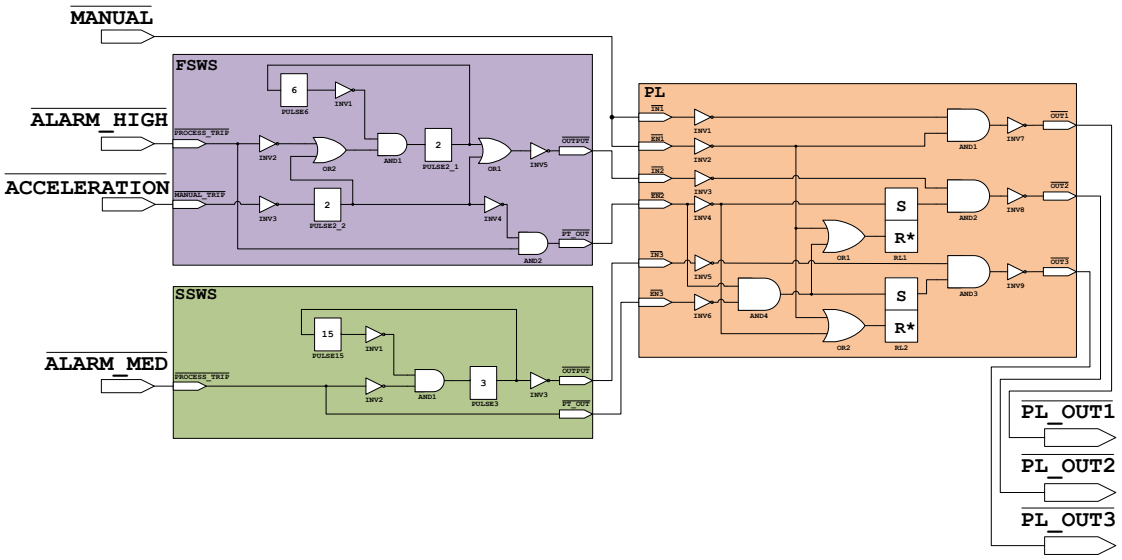

Figure 12: Detailed design diagram of the Power Limitation System

Apart from the modules, some additional changes have been made into Fig. 12 compared to the architecture diagram in Fig. 11. The lines above the I/O signals indicate that they are ‘active low’ signals. Active low signals are related to making the I/O signals fail-safe so that if a line is cut, the system would be activated instead of being rendered non-functional.

Also, there are now six signals going into the PL whereas there are only three signals in the architecture diagram. The three additional signals are the ‘enable’ signals that are used to convey the inputs MANUAL, ALARM_HIGH, etc. to the PL to help determine which system gets to drive the actuator based on which criteria are active. The logic circuits of the systems were designed using the standard logic component symbols defined in the IEEE Std. 91 (IEEE, 1996). The internal structure of the SSWS is in Fig. 13.

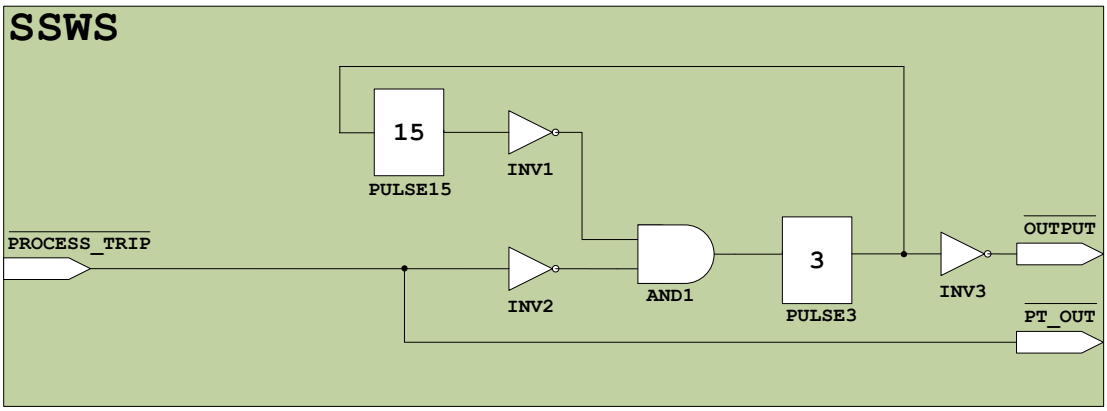

Figure 13: Detailed design diagram of the Slow Stepwise Shutdown System (SSWS)

The SSWS has one input, PROCESS_TRIP, whose name originates from the SWS case study (Lötjönen, 2012). The PROCESS_TRIP represents the alarm caused by the measurement signals crossing their threshold values. The two outputs are OUTPUT and PT_OUT. The other output PT_OUT (abbreviated from 'process trip out') simply follows the value of the PROCESS_TRIP and is thus only used to convey the activating criteria to the prioritising system. All I/O signals are active low which is indicated by the lines over the signal names. The internal logic however is active high which is why there are the inverters INV2 and INV3 near two of the I/O ports.

The functionality of the SSWS was mostly implemented using standard logic circuit symbols for the basic logic blocks such as inverters and AND-blocks. The PULSE blocks are not standard logic blocks but instead blocks that implement the functionality of a monostable multivibrator circuit that initiates a pulse of a specified length on their output when a specified event on their input signal is detected. The pulse-blocks are only activated by a rising edge on the input signal and do not react to the input during an initiated pulse. In the SSWS, there are three- and fifteen-second pulse blocks.

From the internal structure of the SSWS the specified behaviour of the SSWS emerges in the following way: when the PROCESS_TRIP is activated i.e. its value goes from '1' to '0', that signal is first converted into a '1' by the inverter (INV2) and then fed to the second input of the first AND-block (AND1). As the system was inactive before the activation of the PROCESS_TRIP, the other input of the first AND-block (AND1) is '1' for now.

The output of the AND1 then becomes '1' since both its inputs are '1'. As the output of the AND1 is also the input of the three-second pulse block (PULSE3), its activation then initiates a three-second pulse on the output of the PULSE3. The activated output of the PULSE3 is then inverted in INV3 and then assigned to the output port OUTPUT. The output of the SSWS is now active and will remain active for three seconds since the three-second pulse-block (PULSE3) maintains the pulse for three seconds.

At the same time, the output of the PULSE3 is conveyed to the fifteen-second pulse-block (PULSE15). Its output is activated for fifteen seconds, inverted by the inverter (INV1) and conveyed to the first input of the AND1. Now, the output of the AND1 will be '0' for fifteen seconds regardless of its second output PROCESS_TRIP. This means that after the three seconds that the output of the SSWS has been active, a twelve-second idle period follows.

Then, after the fifteen-second sequence, the system becomes responsive to the PROCESS_TRIP that can once again have an effect on the output of the AND1. The

functionality of the FSWS is very similar to the SSWS with only the added acceleration feature and a shorter operation sequence. The logic circuit of the FSWS is in Fig. 14.

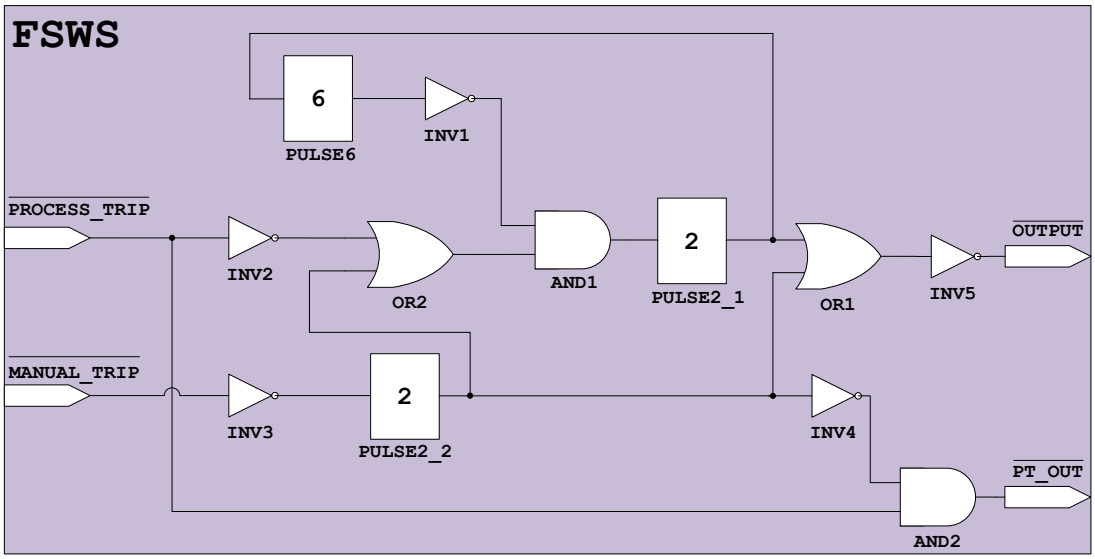

Figure 14: Detailed design diagram of the Fast Stepwise Shutdown System (FSWS)

The FSWS has two inputs: PROCESS_TRIP and MANUAL_TRIP. Their names, like in the SSWS, originate from the SWS case study. The PROCESS_TRIP is the main initiating criterion that works just like in the SSWS. The MANUAL_TRIP represents a button in the control room that initiates additional actuation pulses. The outputs of the FSWS are OUTPUT and PT_OUT. OUTPUT is the actual output of the system and PT_OUT follows the values of the inputs PROCESS_TRIP and MANUAL_TRIP as they are both initiating criteria. The PT_OUT is simply used to convey the initiating criteria to the prioritisation. The I/O signals of the FSWS are all active low, hence the lines over their names. Like in the SSWS, the internal logic is active high which is why there are inverters near the I/O ports.

If the MANUAL_TRIP is untouched, the functionality of the FSWS is very similar to the SSWS with only the pulse being two seconds instead of three seconds and the idle time being four seconds instead of twelve seconds. The complete sequence of the FSWS without the acceleration is thus six seconds instead of the fifteen seconds of the SSWS.

The MANUAL_TRIP initiates the following functionality: when it is activated i.e. its value goes from ‘1’ to ‘0’, it is first inverted by the inverter (INV3) so that the two-second pulse block (PULSE2_2) gets a rising edge on its input. A two-second pulse is initiated and conveyed to both OR-blocks of the system. The OR1 enables the MANUAL_TRIP to directly affect the output signal whereas the OR2 enables the MANUAL_TRIP to drive the overall logic like the PROCESS_TRIP. The two OR-gates enable some built-in redundancy of the MANUAL_TRIP functionality.

The MANUAL_TRIP has a direct effect on the output of the FSWS since the output of the PULSE2_2 is conveyed to the OR-block in the end (OR1) whose output then is inverted and assigned to the output port of the FSWS. The other input of the OR1 is the output of the basic six-second cycle that, if running, is not affected in any way when the MANUAL_TRIP is activated. The MANUAL_TRIP thus initiates additional two-second pulses on top of the basic sequence if the sequence is currently running. Both outputs of SSWS and FSWS are connected to the inputs of the PL. The logic circuit diagram of the PL is in Fig. 15.

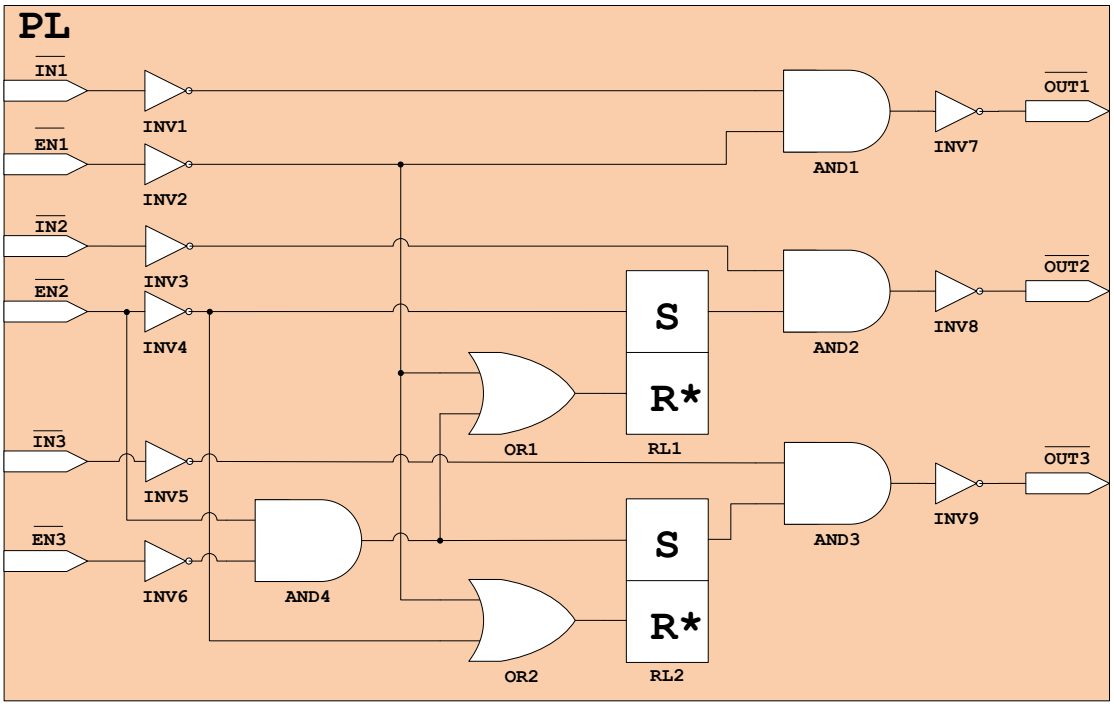

Figure 15: Detailed design diagram of the Priority Logic (PL)

The PL in Fig. 15 has six inputs and three outputs and all of them are active low. The inputs IN1, IN2, and IN3 are the 'payload' signals i.e. the output signals of the subsystems to be prioritised. The inputs EN1, EN2, and EN3 are the 'enable' signals that are used to determine which of the payload signals gets through to be the output of the entire PLS.

The outputs correspond to the inputs so that the OUT1 is active when the IN1 and EN1 are active, the OUT2 is active when IN2 and EN2 are active, and the OUT3 is active when the IN3 and EN3 are active. Only one of the outputs OUT1, OUT2, or OUT3 can be active at a time. The prioritisation operates so that the highest level signal always gets through. For example, when the IN3 and EN3 and thus OUT3 are active, if the EN2 becomes active, the IN3 and EN3 no longer have any effect on the OUT3 because now the IN2 is followed and only OUT2 can be activated.

The 'R-latch' blocks (RL1 and RL2) are memory elements that enable the persistence of the previous state of the PL when all signals are inactive. They are needed because there may be situations where a subsystem is active but the initiating criterion has already become inactive. The requirements state that the operating sequence of a system must be completed regardless of the state of the initiating criterion. The internal structure of an R-latch is illustrated in Fig. 16.

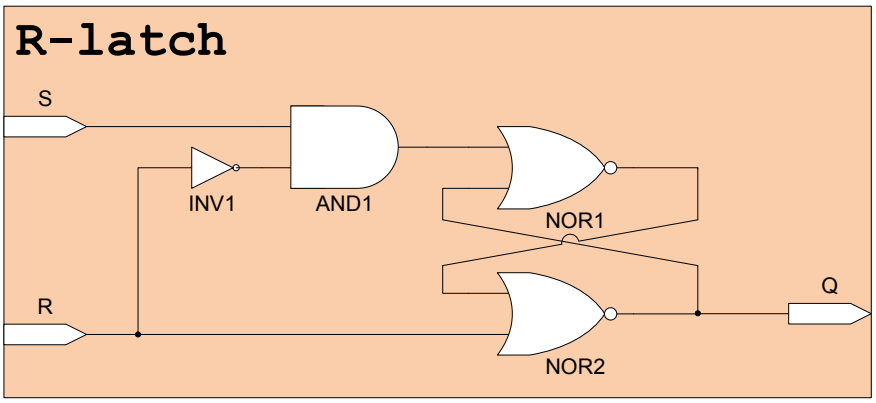

Figure 16: R-latch – An SR-latch whose reset (R) signal has a higher priority than the set (S) signal

The R-latch in Fig. 16 is a modified version of an SR-latch that is a common component in digital logic design. The traditional SR-latch lacks the INV1 and AND1 blocks that are in the R-latch i.e. it only consists of the NOR1 and NOR2 blocks. It is said that the SR-latch should not be used in digital designs due to some shortcomings that cause problems (Poiksalo, 2007). Problems arise because in a normal SR-latch, if both inputs become active, the logic cannot be resolved deterministically because of the feedback between the NOR-gates. In digital logic design, this kind of an issue is called a race condition or a metastable condition. Additionally, when multiple SR-latches are used to set or reset each other, the timings can cause problems because of the apparent concurrent events.

In the self-made R-latch, these problems have been avoided by adding the additional components (INV1 and AND1). The main function of the additional components is to implement a higher priority for the reset (R) signal. Additionally, when both inputs are active, the logic can be resolved because INV1 and AND1 make the set (S) signal to have no effect and the reset gets through.

Despite the possible shortcomings, the R-latch was used in the design because it is simple and there are no components with an equivalent functionality. As there are no feedbacks between the R-latches in the PL design, and the problem with both of the inputs being active at the same time has been mitigated, the R-latch can be deemed a good enough component for the design.

Behavioural description

The behavioural description was done using VHDL. The code was written by hand using a regular text editor that supports VHDL syntax. The basic logic blocks such as AND-, OR-, and inverter blocks were the same that were used in the Stepwise Shutdown System case study (Lötjönen, 2012). The subsystems were put together using the basic blocks.

To enable implementing the systems in different hardware, some modifications needed to be made to the detailed design logic circuit diagram. The somewhat simplified logic circuit diagram that was eventually implemented in VHDL is in Fig. 17.

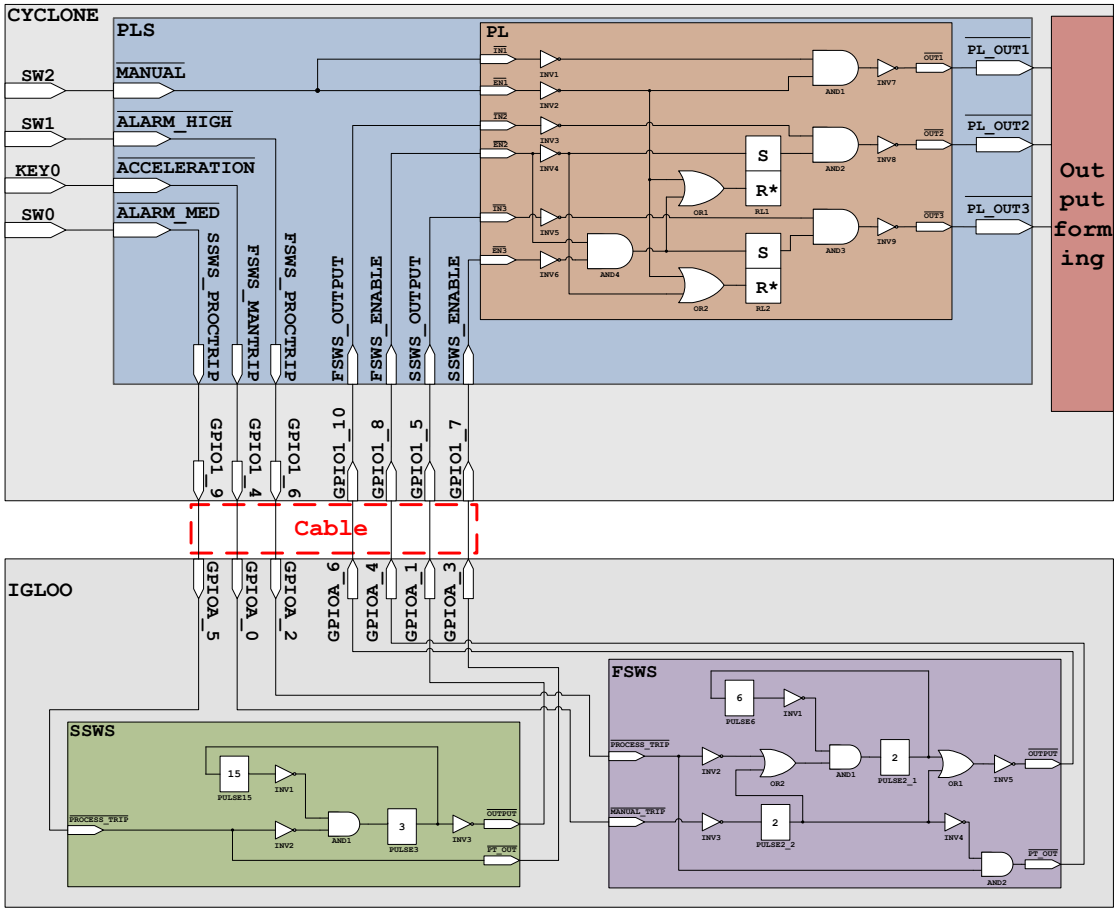

Figure 17: Final VHDL logic diagram - placement of the systems on different hardware

The priority logic and the user I/O interface were implemented in the Cyclone FPGA and the FSWS and SSWS were implemented in the IGLOO FPGA. Thus, the highest level VHDL entities CYCLONE and IGLOO in Fig. 17 are named accordingly and reside in the respective FPGAs. Every block in the design gets a clock signal which is omitted in the figures to make them clearer. The blocks inside the CYCLONE entity get a different clock signal than the ones in the IGLOO because they are from physically different sources. A 50 MHz clock is used in the Cyclone development kit and a 48 MHz clock is used in the IGLOO development kit.

The design is synchronous which means that on an FPGA, every block receives and is activated by the same clock signal. Synchronous design, as opposed to an asynchronous design, introduces additional delays that are however minor and do not cause violations of the one millisecond response time stated in the requirements. Additionally, based on the discussions at the fifth International Workshop on the Applications of FPGAs, the FPGA-based NPP safety automation applications should always be synchronous. Synchronous design mitigates some of problems found in asynchronous design such as glitches and timing issues (Bobrek *et al.*, 2007).

In Fig. 17, the signals representing the individual wires in the interconnecting cable are highlighted. The introduction of the cable imposed some major modifications to the system architecture as the PLS needed to be split in two parts due to its separation on to different hardware.

Inside the blue PLS entity of in Fig. 17, the additional I/O ports are in the bottom named SSWS_PROCTRIP, FSWS_MANTRIP, FSWS_PROCTRIP, FSWS_OUTPUT, FSWS_ENABLE, SSWS_OUTPUT, and SSWS_ENABLE. These are then connected

to the I/O ports (GPIO1_N) of the CYCLONE entity. The ports represent the pins of a 40-pin connector used in the interconnection. Corresponding ports were implemented in the IGLOO as well (GPIOA_N). The ports representing the pins in both the CYCLONE and the IGLOO are named according to the reference manuals of the devices (Altera, 2006; Microsemi, 2012c).

The GPIOA_N-ports are the only ones in the IGLOO entity but there are additional ports in the CYCLONE entity. The SW2, SW1 and SW0 correspond to three of the switches in Fig. 9, and KEY0 is one of the buttons in the figure. The KEY0 is the ACCELERATION, and the switches are MANUAL, ALARM_HIGH, and ALARM_MED, respectively.

The other outputs of the CYCLONE entity have not been elaborated in Fig. 17 and are simply regarded as the "Output forming" box in the figure. The output forming contains an entity and a few functions for driving the seven-segment screens, blinking the green LEDs, and transmitting data through the serial port in Fig. 9. As these are simply supporting the observation of the outputs, the details of their implementation are not elaborated in Fig.17 to conserve space.

The design of the system is based on the principle of modularity. Every VHDL entity (a block in the design) resides within a separate source code file and they are instantiated inside other entities. For example, the CYCLONE VHDL entity contains an instance of the PLS entity as well as a few processes and an entity that deal with the output forming. Inside the PLS there is an instance of the PL entity that in turn contains instances of the basic blocks defined in separate source code files. The highest level entities are CYCLONE and IGLOO, and the lowest level entities are the elementary logic blocks such as inverters.

As examples of VHDL, the source code for the IGLOO, SSWS, and a pulse block is elaborated next. Similarly to Section 2.3, these will be used as examples in the rest of this section when the products of the other design phases are presented. It would not be meaningful to elaborate the source code of the entire system as there are thousands of lines of code.

The elaboration of the source code will be conducted from a top-down perspective. First, the IGLOO entity is described, and then one of the components, SSWS in particular, is described after which one of the blocks, the PULSE block, is described. The source code for the IGLOO is presented piecewise below with elaboration between different parts.

```
entity IGLOO is
   port( CLK     : in  bit;
         GPIOA_0 : in  bit; -- fsws mantrip
         GPIOA_1 : out bit; -- ssws output
         GPIOA_2 : in  bit; -- fsws proctrip
         GPIOA_3 : out bit; -- ssws enable
         GPIOA_4 : out bit; -- fsws enable
         GPIOA_5 : in  bit; -- ssws proctrip
         GPIOA_6 : out bit); -- fsws output
end entity;
```

The first part of the source code of the entity is the *entity declaration* (Ashenden, 1996). In the entity declaration, the name of the entity, the I/O ports, and *generic parameters* are defined. Generic parameters can be used to convey information such as constant values when making component instances of an entity. The `IGLOO` does not have generic parameters. The entity declaration is the part that is visible outside of the entity.

As the IGLOO VHDL entity represents the IGLOO development kit, the ports GPIOA_N are named according to the actual names of the specific pins in the reference manual. The green text after the port declarations are VHDL comments showing which signals the ports correspond to. A port declaration defines the name and the type of the port and whether it is an input or an output port. For example, the CLK is the clock signal that is an input of type bit (**in** bit).

```
architecture MAIN of IGLOO is
```

An architecture is where internal implementation of the entity is defined. An entity can have multiple architectures but only one can be synthesised at a time. IGLOO has only one architecture MAIN that contains everything that is related to the IGLOO. After this line, the declarative part of the architecture begins. In it, the constants, variables, components, etc. are declared.

```
   constant SECOND : integer := 48000000;
```

The only constant of the IGLOO is the SECOND that is used to convey to the FSWS and the SSWS instances the amount of clock cycles needed to spend one second of time. As the clock frequency of the IGLOO development kit is 48 MHz, it takes 48,000,000 clock cycles to spend one second.

```
   component FSWS is
      generic( SECOND : integer);
      port( CLK          : in bit; -- clock signal
            PROCESS_TRIP : in bit; -- active low
            MANUAL_TRIP  : in bit; -- active low
            OUTPUT       : out bit; -- actuation
            PT_OUT       : out bit); -- conveys the pt
   end component;

   component SSWS is
      generic( SECOND : integer);
      port( CLK          : in bit;
            PROCESS_TRIP : in bit;
            OUTPUT       : out bit;
            PT_OUT       : out bit);
   end component;
```

The FSWS and the SSWS are used in the IGLOO as *components*. First they are declared and then later on instantiated. A *component declaration* looks very similar to an entity declaration as the same information is given in it. The implementation of the components need not be defined if they have already been defined in a separate source code file. The compilers can automatically find them if they are named the same way in the component declaration and in the original entity declaration.

```
begin

         FSWS_C : component FSWS
      generic map( SECOND => SECOND )
      port map( CLK => CLK,
                PROCESS_TRIP => GPIOA_2,
                MANUAL_TRIP => GPIOA_0,
                OUTPUT => GPIOA_6,
                PT_OUT => GPIOA_4);
```

```
   SSWS_C : component SSWS
      generic map( SECOND => SECOND )
      port map( CLK => CLK,
                PROCESS_TRIP => GPIOA_5,
                OUTPUT => GPIOA_1,
                PT_OUT => GPIOA_3);

end architecture;
```

After the "begin", the declarative part of the architecture ends and the actual behaviour or structure of the architecture is defined. The architecture of the IGLOO is a structural architecture since it does not define any sequential behaviour but instead just the structure of the entity IGLOO. In a structural architecture, the behaviour becomes implicitly defined through the components and their interconnections.

The architecture consists only of the instantiations of the two components FSWS_C and SSWS_C. The 'C'-suffix stands for "component". The I/O ports of the IGLOO entity are connected to the corresponding I/O ports of the component instances by using the "port map" statement. The constant SECOND is also conveyed to the instances through the "generic map" statement. The left side of a mapping is the name of the port of the instantiated entity and the right side, after the "=>", is the signal or port that is assigned to the port of the instantiated entity. The source code for the entity ends in the "end architecture" statement.

As the IGLOO entity is only used to wrap together the FSWS and SSWS and to provide an interface between the actual devices on the development kit and the VHDL constructs, it is very simple. The complexity lies within the instantiated components. The source code for the SSWS is piecewise presented below with elaboration between the parts.

```
entity SSWS is
   generic( SECOND : integer );
   port( CLK          : in bit;
         PROCESS_TRIP : in bit;
         OUTPUT       : out bit;
         PT_OUT       : out bit);
end entity;
```

First, there is the entity declaration of the SSWS. In the SSWS, there are four I/O ports: CLK, PROCESS_TRIP, OUTPUT, and PT_OUT. The type of the ports is specified and whether a port is an input or an output port. For example, PROCESS_TRIP is defined as an input that is of type bit (in bit).

The SSWS also has one generic parameter SECOND. It is used when the SSWS is instantiated to specify the amount of clock cycles it takes to spend one second of time. Since the SSWS is implemented in the IGLOO development kit that has a 48 MHz clock, the value of the SECOND is 48,000,000. The generic parameter enables the use of the SSWS in other devices with a different clock frequency as well.

```
architecture MAIN of SSWS is

   signal AND1_OUT    : bit;
   signal INV1_OUT    : bit;
   signal INV2_OUT    : bit;
   signal INV3_OUT    : bit;
```

```
signal PULSE15_OUT : bit;
signal PULSE3_OUT  : bit;
```

Signals in VHDL can be used to transmit information between instances of different components or to convey information to and from the I/O ports. The signals correspond to the lines between the blocks in the entities e.g. in Fig. 17. Signals are essential when building a system out of interconnected blocks as the interconnections are implemented using the signals. The signals of the SSWS entity correspond to the "wires" between the blocks in the SSWS logic circuit diagram in Fig. 13.

```
component PULSE is
   generic( PULSE_LENGTH : integer;
            SECOND       : integer);
   port( CLK    : in  bit;
         INPUT  : in  bit;
         OUTPUT : out bit);
end component;


component AND2GATE is
   port( CLK    : in  bit;
         INPUT1 : in  bit;
         INPUT2 : in  bit;
         OUTPUT : out bit);
end component;


component INVERTER is
   port( CLK    : in bit;
         INPUT  : in bit;
         OUTPUT : out bit);
end component;
```

The SSWS architecture, like the IGLOO architecture, is also a structural architecture. There are three components: PULSE, AND2GATE, and INVERTER. The PULSE block acts as a monostable multivibrator: it outputs a pulse of a fixed length in case of a specific event. In this case, the event is a rising edge in the input signal. Components that only respond to rising or falling edges of an input signal are called edge-sensitive and they are common in digital design (Poiksalo, 2007).

The desired length of the pulse is conveyed to the PULSE block using a generic parameter. The amount of clock cycles needed to spend one second is also given as a generic parameter. The I/O ports are simply the CLK, INPUT, and OUTPUT. A rising edge on the INPUT initiates the fixed-length pulse on the OUTPUT. The component declarations of the AND2GATE and the INVERTER are also quite straightforward: clock signal, and inputs and outputs.

```
begin

   PT_OUT <= PROCESS_TRIP;
```

The second output PT_OUT is only used to convey the input PROCESS_TRIP to the PL, so the value of the PROCESS_TRIP is assigned to the PT_OUT before the component instantiations.

```
   PULSE15 : component PULSE
```

```
      generic map( PULSE_LENGTH => 15,
                   SECOND => SECOND)
      port map( CLK    => CLK,
                INPUT  => PULSE3_OUT,
                OUTPUT => PULSE15_OUT);

   PULSE3 : component PULSE
      generic map( PULSE_LENGTH => 3,
                   SECOND => SECOND)
      port map( CLK    => CLK,
                INPUT  => AND1_OUT,
                OUTPUT => PULSE3_OUT);

   INV1 : component INVERTER
      port map( CLK    => CLK,
                INPUT  => PULSE15_OUT,
                OUTPUT => INV1_OUT);

   INV2 : component INVERTER
      port map( CLK    => CLK,
                INPUT  => PROCESS_TRIP,
                OUTPUT => INV2_OUT);

   INV3 : component INVERTER
      port map( CLK    => CLK,
                INPUT  => PULSE3_OUT,
                OUTPUT => OUTPUT); -- the system output assigned here

   AND1 : component AND2GATE
      port map( CLK    => CLK,
                INPUT1 => INV1_OUT,
                INPUT2 => INV2_OUT,
                OUTPUT => AND1_OUT);

end architecture;
```

The instantiation of the components is similar to the `IGLOO` entity, there are just more components in the SSWS. The `PULSE` blocks of two different lengths are instances of the same `PULSE` component but with a different value in the generic parameter that is used to define the length of the pulse. The source code for the entity ends in the "`end architecture`" statement.

The source code for the `PULSE` block is piecewise presented below with added elaboration between the parts. The `PULSE` block is the most complex block used in the design because it involves dealing with time for which there are no ready-made timer applications that are present e.g. in most microcontrollers. The time needs to be spent by incrementing an integer counter each clock cycle and comparing its value to some constant representing e.g. the amount of clock cycles needed to spend one second.

```
entity PULSE is
   generic( PULSE_LENGTH : integer ;
            SECOND       : integer );
   port(CLK    : in  bit ;
        INPUT  : in  bit ;
        OUTPUT : out bit );
end entity;
```

The entity declaration is similar to the previous ones presented. There are two generic parameters and three I/O ports. The PULSE_LENGTH is used to specify the length of the pulse in seconds, and the SECOND is the amount of clock cycles needed to spend one second. The OUTPUT is activated for PULSE_LENGTH seconds when there is a rising edge on the INPUT.

```
architecture BEHAVIORAL of PULSE is
   signal   RESULT      : bit;
```

The PULSE has one architecture and, unlike the previous ones, it is a behavioural architecture. This means that the architecture describes behaviour e.g. through sequential or concurrent operations rather than defining a structure through instantiating components. The signal RESULT is used to temporarily store the value of the output before assigning it to the output port. The reason for this will be elaborated later on.

```
begin

   process ( CLK, INPUT ) is
      variable COUNTER    : integer := 0;
      variable PULSE_ON   : bit;
      variable PREV_INPUT : bit;
```

The architecture contains one *process* that is used in VHDL when sequential operations are needed. The statements and operations inside a process are executed in a sequential order like in normal software programmes. To spend time, sequential operations are necessary. After the word "process", there are brackets inside which are the inputs CLK and INPUT. This is called a *sensitivity list* and it is used to define which signals cause the process to be executed. Now, whenever there is activity in either the CLK or the INPUT, the process is executed.

Like an architecture, a process also has a declarative part. The variables defined in the declarative part of this process are supporting the functionality of the block. The COUNTER is incremented each clock cycle to keep track of how many clock cycles have elapsed i.e. how much time has passed. The PULSE_ON is a flag used as an indicator when the pulse is on-going. The PREV_INPUT is used to store the value of the INPUT at the end of each process cycle. The value is then used on the next process cycle to determine if there is a rising edge on the INPUT. This will be elaborated later on.

```
   begin

      if CLK'event and CLK = '1' then
```

The very first thing that is checked each time the process is executed is whether there is a rising edge on the clock signal CLK. This is also done in every other block such as the AND2GATE or OR2GATE that has a process. The check makes the design synchronous as the rest of the statements in the processes are only executed when there is a rising edge on the clock signal.

```
         -- if current input is greater than the previous one,
         -- there's a rising edge which initializes the pulse
         if INPUT > PREV_INPUT then
            PULSE_ON := '1';
```

Now, the previously declared PREV_INPUT variable is used to check if there is a rising edge on the input signal INPUT. This is done by simply checking if the current value is greater than the previous value. Also, if there is a rising edge, the flag PULSE_ON is set to 'true' to indicate that the pulse is considered to be on from this point on.

```
-- pulse is kept on for the duration of the pulse
if COUNTER < PULSE_LENGTH*SECOND then
   RESULT <= '1';
   COUNTER := COUNTER + 1;

-- when time runs out, the pulse ends
else
   RESULT <= '0';
   PULSE_ON := '0';
   COUNTER := 0;
end if;
```

If a rising edge was detected on the input, the current value of the counter is compared to the fixed value that indicates the time. As an example of a value that the counter is compared to, if the desired length of the pulse is two seconds and the clock frequency is 48 MHz, the value of the PULSE_LENGTH*SECOND expression would be 2*48,000,000 that is evaluated as 96,000,000.

If the counter has not yet reached the limit i.e. the if-statement is evaluated as true, a '1' is assigned to the temporary value of the output and the counter is incremented by one. The temporary value is needed because the ports cannot be directly manipulated inside a process.

If the counter has reached the limit i.e. the if-statement evaluates false, it means that the pulse has been on for the desired time. Subsequently, a '0' is assigned to the temporary value RESULT of the output, the PULSE_ON is turned to '0' to indicate that the pulse is not supposed to be on anymore, and the counter is reset.

```
-- if there is no rising edge but there is still time,
-- pulse is continued
elsif COUNTER < PULSE_LENGTH*SECOND and PULSE_ON = '1' then
   RESULT <= '1';
   COUNTER := COUNTER + 1;
else
   RESULT <= '0';
   PULSE_ON := '0';
   COUNTER := 0;
end if;
```

The elsif-statement corresponds to the if-statement where the rising edge of the input was checked. As the rising edge only initiates the pulse, most of the time there is no rising edge detected but the pulse still needs to be output.

The condition inside the elsif-statement is similar to the one in the preceding if-statement with the added PULSE_ON = '1' check. If the pulse has previously been initiated by a rising edge, the flag has also been turned to '1'. Thus, if the counter has not reached the limit, and the flag has been turned, a '1' is assigned to the temporary value of the output and the counter is incremented.

If either the counter reaches the limit or the pulse has not been initialised i.e. the PULSE_ON is '0', the pulse has been completed or does not need to be output due to it

not having even been initialised. Then, a '0' is assigned to the temporary value RESULT of the output, the flag PULSE_ON is turned to '0', and the counter is reset.

```
        PREV_INPUT := INPUT; -- previous input saved for the next clk
    cycle

       end if; -- clk if
    end process;

    -- after each clk cycle, the result is assigned to the output port
    OUTPUT <= RESULT;

end architecture;
```

The last statement of the process is the storing of the value of the current input to the variable PREV_INPUT. The value is then available in the next process cycle. After the process, the value of RESULT is assigned to the output port OUTPUT. As this happens outside of the process, it is done concurrently.

This is a fundamental difference between traditional software run on a CPU and a VHDL application run on an FPGA. When using an FPGA, there can be hundreds or thousands of concurrent commands that are actually executed concurrently whereas when using a microprocessor, the commands are executed sequentially one after the other. In the VHDL code above, the value of the OUTPUT is updated concurrently but the value of the RESULT is updated only at specified times inside the sequential process.

The serial transmission was implemented by creating a VHDL entity that transmits the seven I/O signals of the PLS inside a byte. The speed of the transmission is 9600 bauds i.e. the entity sends 9600 bits per second. The individual bits inside each byte contain the current values of the I/O signals.

The Simulink extension of Matlab is used to read and process the bytes sent by the Cyclone development kit. In order to get the individual bits from the bytes, the bytes first needs to be converted into an eight character binary numbers of 1s and 0s. After that, the binary numbers are converted into strings and exported into an external file. The files can then be opened and inspected e.g. using Microsoft Excel (Microsoft, 2013). This is demonstrated in Section 4.4 when the integration V&V phase is described.

As there are only seven signals, the eighth bit in the byte is redundant. It is constantly kept as '1' to indicate that data is actually being transmitted. If the bit at any time goes to '0', it is an indication that something is wrong with the transmission or reception. The rest of the blocks were implemented in the same way as the ones described above. After the VHDL code of the design blocks was complete, the next step was synthesis where the code was be compiled into a gate-level model and a netlist.

Synthesis

As the FPGAs used in the case study were of different manufacturers, different software tools were used for the FPGAs in synthesis. Altera's Quartus II was used when designing the systems for the Cyclone and it has the synthesis tool built-in whereas Actel's Libero IDE is a collection of third party software tools. Thus, for the IGLOO synthesis, a software tool called Synplify Pro by Synopsys included in the Libero IDE was used.

When synthesising the systems, manual corrections were not necessary. The systems have no redundancy that could be interpreted obsolete by the synthesis tools and thus omitted. The gate-level schematic diagram of the CYCLONE VHDL entity generated by the Quartus II synthesis tool is in Fig. 18.

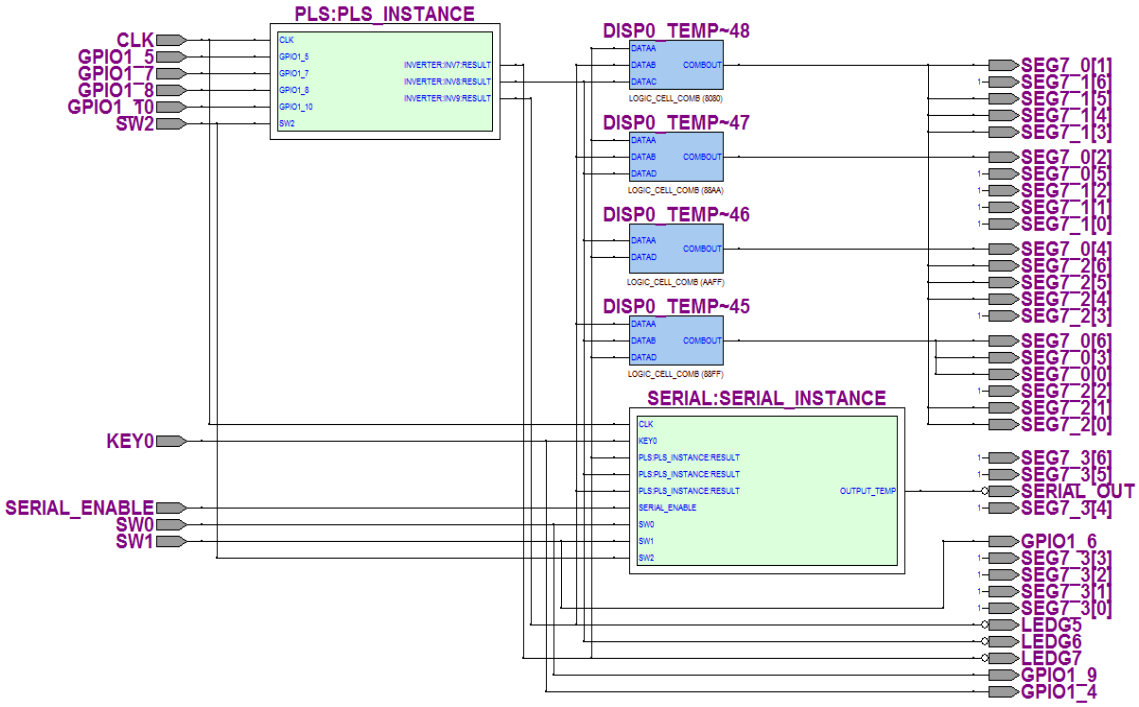

Figure 18: Gate-level schematic diagram of the CYCLONE VHDL entity generated by the Altera Quartus II software tool

The input ports are on the left and the output ports are on the right in Fig. 18. The outputs were regarded as the “Output forming” block in Fig. 17 and they include ports for driving the seven-segment screens on the Cyclone development kit (SEG7_N[N]), and the signal going to the serial port (SERIAL_OUT). There is also an input port called SERIAL_ENABLE that was not in Fig. 17. That is merely used for enabling or disabling the serial transmission when needed.

In Fig. 18, there are the familiar I/O ports SW2, SW1, SW0, KEY0, and the cable outputs GPIO1_N. The ‘PLS:PLS_INSTANCE’ and ‘SERIAL:SERIAL_INSTANCE’ boxes are components and the ‘DISP0_TEMP~4N’ boxes are related to a process dealing with driving the seven-segment displays. The schematic diagram of the IGLOO VHDL entity generated by the Synplify Pro software tool is in Fig. 19.

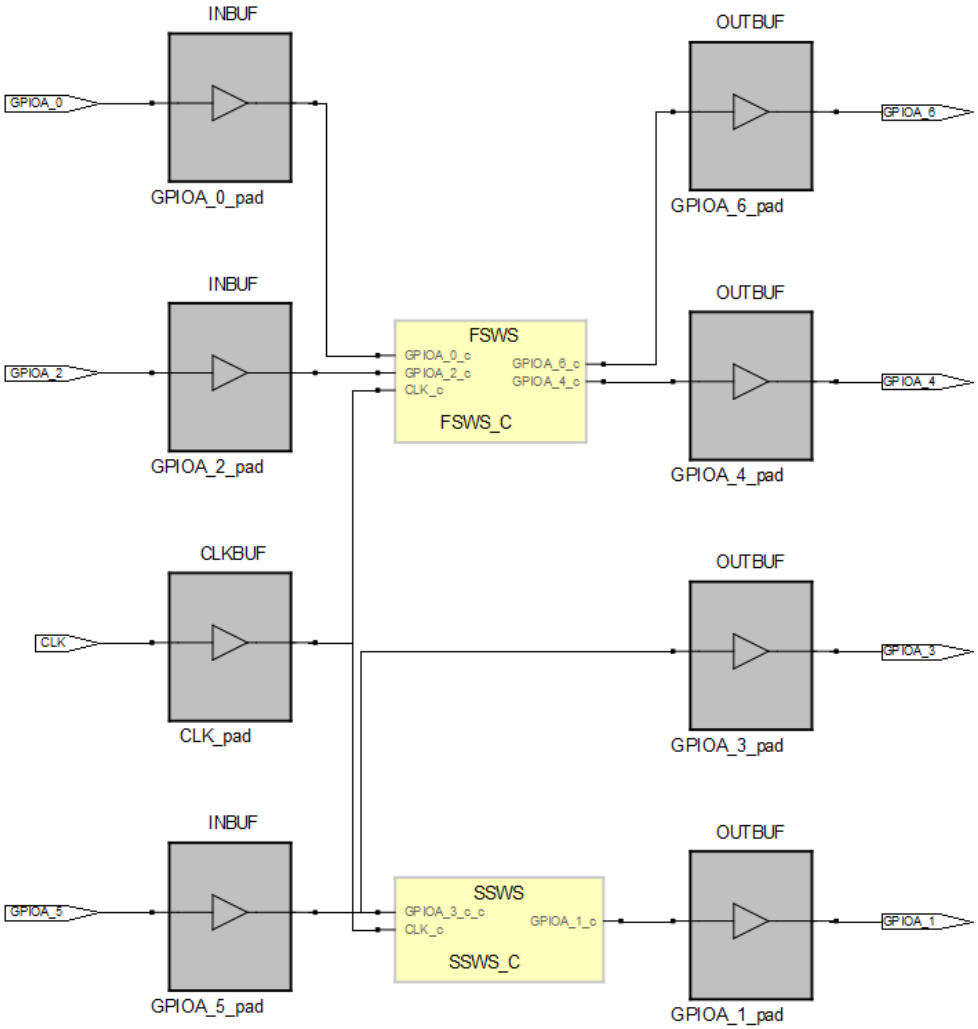

Figure 19: Gate-level schematic diagram of the IGLOO VHDL entity generated by the Synplify Pro software tool

The IGLOO contains the two component instances of the entities FSWS and SSWS that are visible as the yellow boxes in Fig. 19. The I/O ports are also visible and there are no additional ports to the ones in Fig. 17. The grey boxes labelled 'INBUF', 'OUTBUF', and 'CLKBUF' are buffers related to the I/O ports.

Any of the blocks in Fig. 19 can be opened up in the tool to show the contents of the block. The SSWS and one of its components (the PULSE block) are used as an example of the output products of different phases in this section. Therefore, the contents of the yellow SSWS box in Fig. 19 are presented in more detail in Fig. 20.

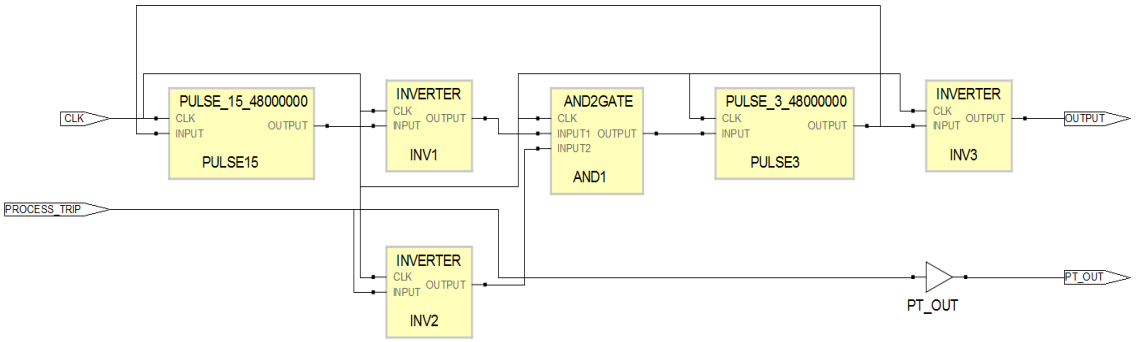

Figure 20: Gate-level schematic diagram of the SSWS VHDL entity generated by the Synplify software tool

The schematic diagram in Fig. 20 looks quite familiar when compared to the detailed design logic circuit diagram in Fig. 13. The shape of the blocks in the schematic diagram is not as distinctive as in the detailed design diagram but the functions of the blocks can be inferred from their names.

It was expected that the diagrams would be similar because the VHDL description was based on a ready-made logic circuit. Had the VHDL code been written based on a functional description instead of a ready-made structure, the synthesis tool would have created a logic circuit design that would have probably looked quite different.

Again, each of the yellow component instances of the entities in Fig. 20 could be opened in the tool and their contents viewed. The schematic diagram of the PULSE block is inspected next. It is not based on a ready-made logic circuit but instead on a functional description of a monostable multivibrator. A small part of the schematic diagram of the three-second pulse block labelled ‘PULSE_3_48000000’ in Fig. 20 is opened up in Fig. 21.

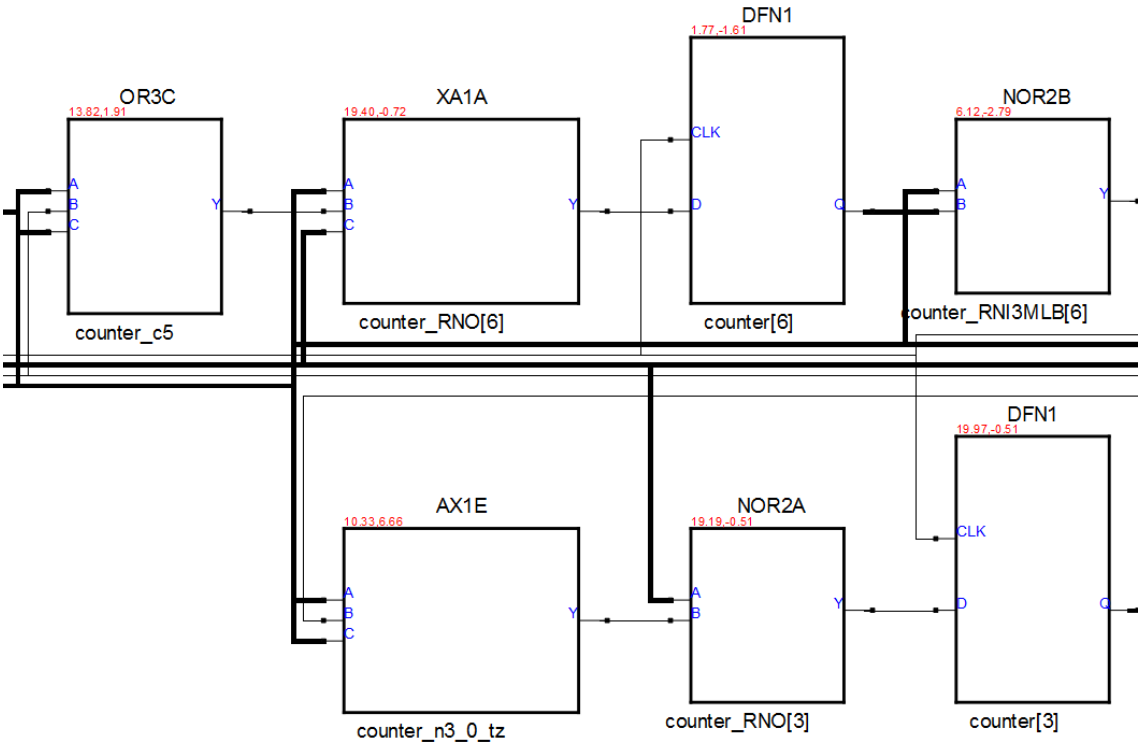

Figure 21: Small part of the gate-level schematic diagram of the three-second pulse VHDL entity in the SSWS generated by the Synplify software tool

The schematic diagram in Fig. 21 looks even more confusing than the previous ones. This is mainly because the actual implementation is now at actual the gate-level. There are a lot of d-flip-flops (DFN1) to implement the integer counter that is used in the pulse. There are also other basic components such as combinatorial logic gates (OR3C, NOR2A). Only seven of the fifty blocks overall in the gate-level schematic diagram are visible in Fig. 21.

Place and route (PAR)

Like synthesis, the PAR was done using different software tools. The PAR for the Cyclone was performed using the built-in tool in the Quartus II whereas the PAR for the IGLOO was done using Actel's Designer software tool included in the Libero IDE. In the PAR, the I/O ports of the design blocks were assigned to specific pins of the development kits. The reference manuals of the development kits contain information about which devices are connected to which pins. The software tools have an interface for easily assigning the pins.

One of the products of the PAR is the configuration file. Two configuration files were generated for the Cyclone FPGA: one for direct JTAG configuration and one for configuring the non-volatile configuration memory on the Cyclone development kit. For the flash-based IGLOO, only one configuration file was generated.

The information in the configuration files is binary i.e. at the lowest possible abstraction level. Such information is difficult to interpret manually without special expertise. A small part of the JTAG configuration file containing the configuration for the CYCLONE entity for the Cyclone FPGA is in Fig. 22. The configuration file was examined using the Frhed free hex editor (Frhed, 2009).

00000	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	6a	d7	ff	40
00014	00	00	65	3b	00	6a	d7	ff	40	00	00	65	3b	00	6a	d7	ff	40	00	00
00028	65	3b	00	00	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff
0003c	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	00	04	05	04
00050	04	04	05	05	04	07	07	07	07	07	07	07	07	07	07	06	06	07	06	07
00064	07	07	06	06	06	06	06	06	06	03	03	02	02	03	03	02	03	77	f3	01
00078	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
0008c	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
000a0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
000b4	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
000c8	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
000dc	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
000f0	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00104	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
00118	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
0012c	01	01	01	01	01	01	01	01	01	01	01	00	00	00	00	00	00	00	00	00
00140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00154	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00168	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0017c	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001a4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001b8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001cc	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001f4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00208	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0021c	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 22: Part of the JTAG configuration file of the CYCLONE entity for the Cyclone FPGA

There are 486,568 bytes in total in the JTAG configuration file, and only the first 580 bytes are shown in Fig. 22. The configuration files used to configure the IGLOO FPGA and the non-volatile configuration memory on the Cyclone development kit are larger and seem more complex than the JTAG configuration file.

The other product of the PAR is the model of the physical layout of the design on the FPGA. All of the CLBs and I/O pins of the FPGA chip are visible in the software tools. A general view of the physical layout model of the CYCLONE design on the Cyclone FPGA is in Fig. 23.

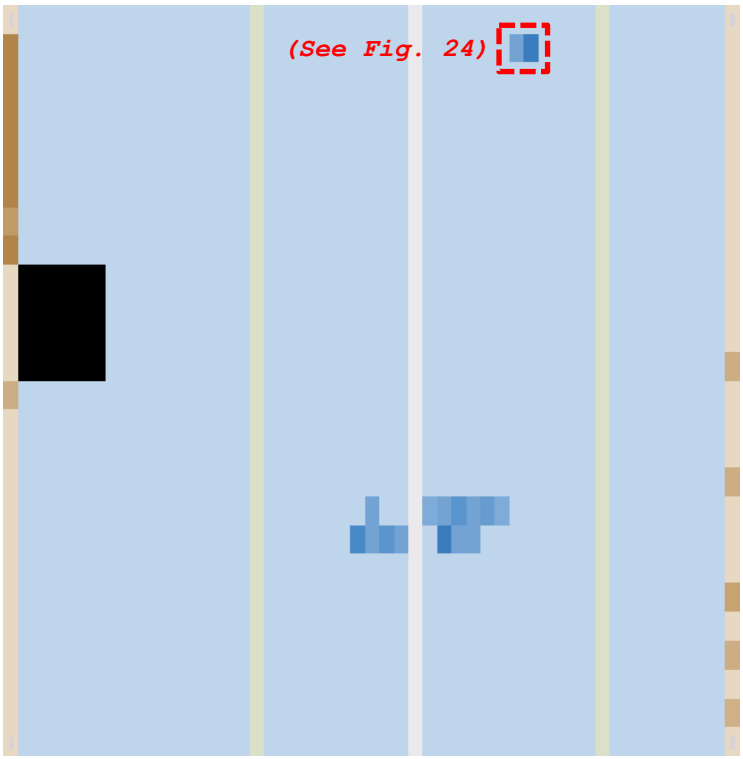

Figure 23: Model of the CYCLONE design mapped into the Cyclone FPGA with the added red rectangle whose contents are elaborated in Fig. 24

Many details cannot be seen in Fig. 23. However, the utilisation of the FPGA resources can be seen. The small blue rectangles represent the CLBs and the brown blocks on the edges represent the I/O blocks. The utilisation of a block is indicated by how dark its colour is: the darker the block, the more of the resources of the CLB are utilised. The black section on the left does not contain CLBs.

Now, only a very small amount of the blocks are used since the design is relatively simple. The individual CLBs can be inspected by zooming in on the model. In Fig. 23, two of the CLBs are surrounded by a red circle. When zoomed in, these CLBs become more detailed as can be seen in Fig. 24.

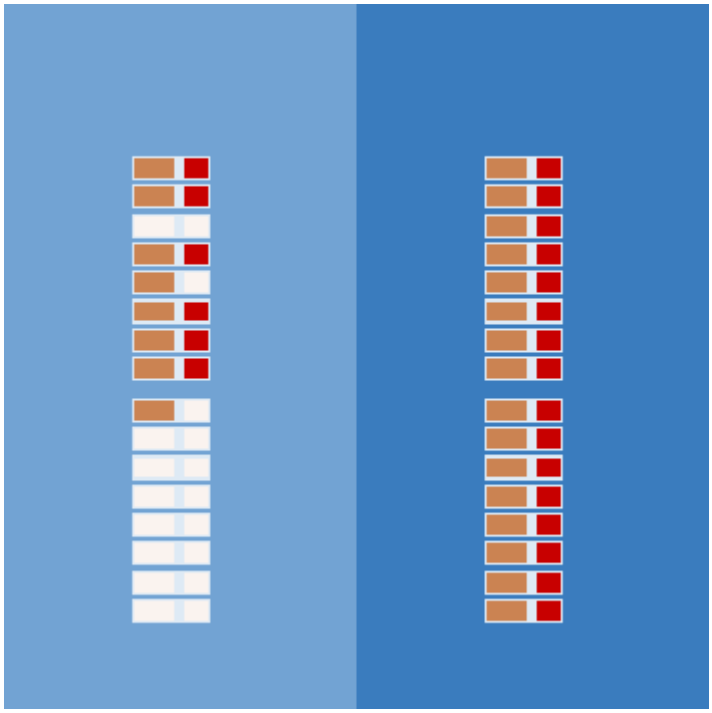

Figure 24: The utilisation of the resources of two CLBs in the physical layout model of the Cyclone

The orange-red-coloured blocks in Fig. 24 are the utilised resources of the CLB and the white blocks are unutilised. In the CLB on the left, in the resources where the orange and red part are both utilised, information such as temporary results of inverters and other combinatorial logic blocks are stored. In the two resources with only the orange part utilised, temporary results related to one of the seven-segment displays are stored. All of the resources of the CLB on the right are utilised.

The model of the physical model of the IGLOO design on the IGLOO FPGA is similar to the Cyclone. The CLBs are portrayed a little differently since the PAR is done using a different software tool. Half of the model of the physical layout of the IGLOO is in Fig. 25.

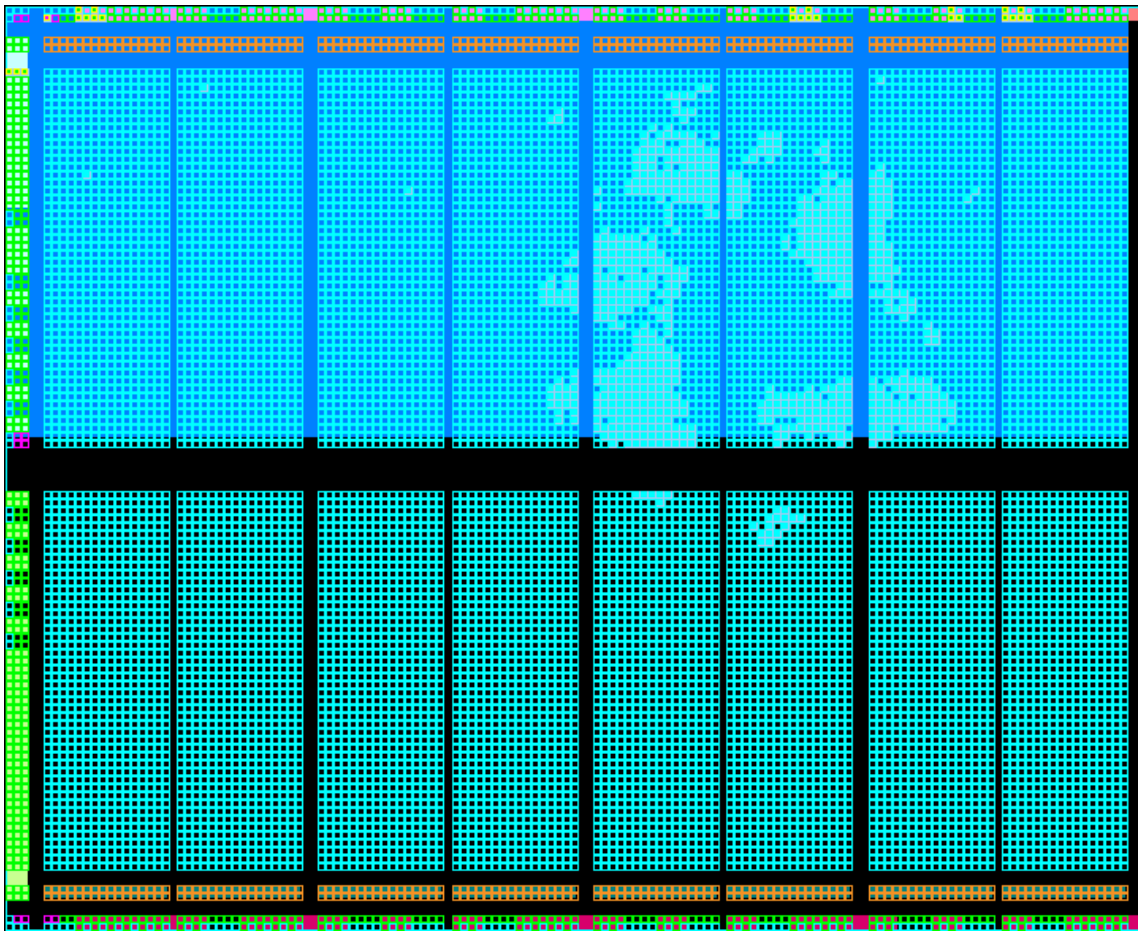

Figure 25: Half of the model of the physical layout of the IGLOO design on the IGLOO FPGA.

Only half of the physical layout model is pictured in Fig. 25. Actually, the model continues to the right but as the other half is empty (CLBs unutilised), and the whole picture would be too wide to fit on a page, only the left half is included here.

The small black squares filling the sixteen visible centre regions in Fig. 25 represent the CLBs and the squares on the edges represent the I/O pins. Utilised blocks are again coloured differently. The utilised CLBs are lighter than the unutilised meaning that most of the CLBs are unutilised in the IGLOO FPGA like in the Cyclone.

The pulse blocks in the FSWS and SSWS are responsible for utilising most of the CLBs in the design. In most blocks, operations are only done using binary valued data but in the pulse blocks, there are the large integer counters that are used to spend time. Operating with integers requires a lot more memory capacity than operating with binary data.

Configuration

In configuration, the configuration files generated in the PAR were used to configure the FPGAs. The Cyclone has two configuration methods: JTAG and Active Serial (Altera, 2006). Both of these are done using a USB-cable between a PC and the development kit. JTAG is simply the direct configuration of the FPGA through the JTAG pins. Active Serial configuration is the transfer of the configuration file to the external non-volatile configuration memory on the Cyclone development kit in Fig. 9.

Whenever the development kit is powered up, the FPGA is automatically configured through the JTAG pins using the stored configuration bit stream file.

The IGLOO is also configured through the JTAG pins (Microsemi, 2012a). The configuration was done using Actel's FlashPro software tool included in the Libero IDE. As the flash-based switchboxes themselves are non-volatile, no external memory is needed to store the configuration when the power is cut. The configuration of the IGLOO is also done using a USB-cable between the PC and the IGLOO development kit.

Integration

In integration, the FPGAs were connected together after being individually configured. This was done by connecting a forty-pin parallel cable to the appropriate connectors on each development kit. The assignment of the signals to the wires of the cable is illustrated in Fig. 26.

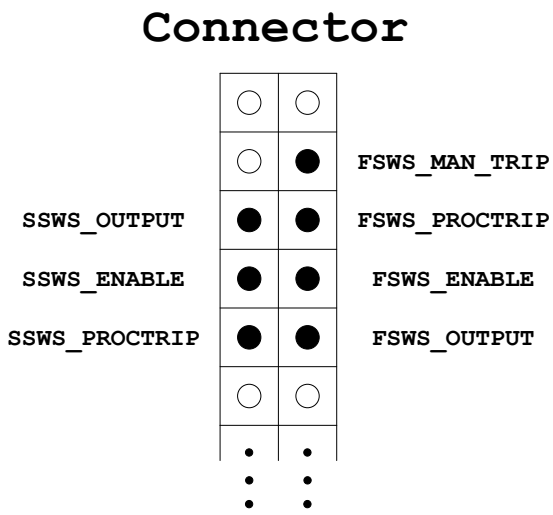

Figure 26: Signals in the interconnection cable

The top part of the forty-pin cable/connector is illustrated in Fig. 26. The signals corresponding to those in Fig. 17 are assigned to specific pins and thus separate wires in the parallel cable. A standard serial cable was also connected to the Cyclone development kit in order to enable the transmission of the I/O signals to a PC. The complete FPGA installation is in Fig. 27.

Figure 27: The Cyclone (left) and IGLOO FPGA development kits connected together, powered up, and running the PLS

As the system was not intended to be installed in a realistic environment, the connection of the cable constituted the whole integration design phase which then completed the development process. Thus, the installation in Fig. 27 is the final product of the whole design process.

4.4 Verification and Validation

The V&V process was followed through similarly to the design process. As mentioned earlier, the aim was not to comprehensively show that the PLS is error-free but instead to try out the different methods described in Section 2.4 and to see what kinds of evidence they could provide of the correctness of the system. The purpose was to get an impression of what are the challenges and limitations related to the utilisation of the methods and are they easily applicable to actual systems. In the following text, the activities performed and challenges faced in different phases of the V&V process are described.

Requirements V&V

The requirements V&V consisted of a requirements review. The requirements specification document was reviewed by several people and, based on the comments of the reviewers, several changes were made and some errors found. Some of the changes are described in the following text.

A few of the requirements were in conflict with each other. One of the original requirements was “While *Acceleration* is inactive, *AS1 enable* shall be deactivated”. The AS1 enable is supposed to be active when either of the inputs High alarm or Acceleration is active. However, the requirement stated that whenever the Acceleration is deactivated, the AS1 enable should also be deactivated even though the High alarm could be active and thus the AS1 enable should not be deactivated.

The requirement was changed to "While *High alarm* and *Acceleration* are inactive, *AS1 enable* shall be inactive”. Now it is required that both inputs are inactive in order

for the AS1 enable to also be made inactive. The independent reviewers had an objective viewpoint on the requirements, and thus other changes that were made were mainly clarifications and making the requirements less ambiguous to an independent reader.

Architecture V&V

The architecture V&V consisted only of an architecture review. Simulation was not done because it would have required additional effort to develop the modules in a way that would have enabled simulation. Because the key logic circuit diagrams were already partially known (modifications of the SWS), the additional effort in coming up with an architecture that could be simulated would have been in vain. It was much more feasible to simulate the subsequent detailed design logic circuits.

The architecture review was iterative. There were six different versions of the architecture before the final one emerged. Each of the versions had the same three subsystems that were in the final version but they were organised differently and the enable-signals were also implemented a little differently. The major changes originated from the need to make the system more realistic.

For example, originally the priority system was located before the other subsystems so that it would activate the systems instead of the inputs activating them directly. The priority logic was then moved to the other side, after the other systems and closer to the actuator, because one of the reviewers said that the priority logic is typically implemented that way in real systems.

Detailed design V&V

A review was also conducted during detailed design V&V. In the review, the logic circuit diagrams were inspected. The review did not reveal any errors in the structure. However, errors are difficult to find in logic circuit diagrams by simply looking at them. Other V&V was still required to deem the circuits correct.

The logic circuit diagrams were also simulated because they were implemented using standard logic blocks. The software tool used for the simulation was the Simulink extension of Matlab by Mathworks (Mathworks, 2012). Simulink supports the simulation of the standard logic blocks and even has a wide variety of ready-made blocks that could be used. A model of the PLS was constructed using the blocks found in Simulink and is illustrated in Fig. 28.

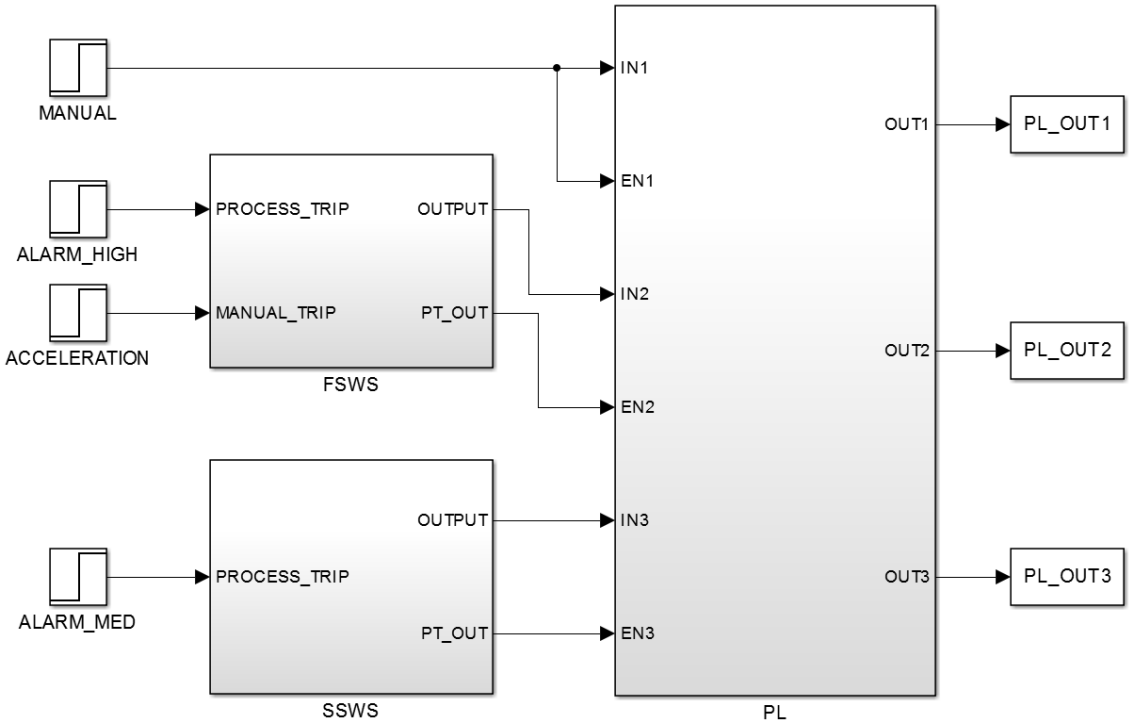

Figure 28: Simulink-model of the Power Limitation System (PLS)

The design in Fig. 28 looks similar to the architecture diagram in Fig. 11. The subsystems were implemented using the 'Subsystem'-blocks in Simulink. The input signals were generated using the 'Step'-block. The model could be easily modified to accept external inputs but as the test cases were scarce and simple, it was easier to use the Step-block. The output signals were exported to the Matlab workspace by using the 'To workspace'-blocks. In Fig. 29, the contents of the FSWS subsystem are illustrated.

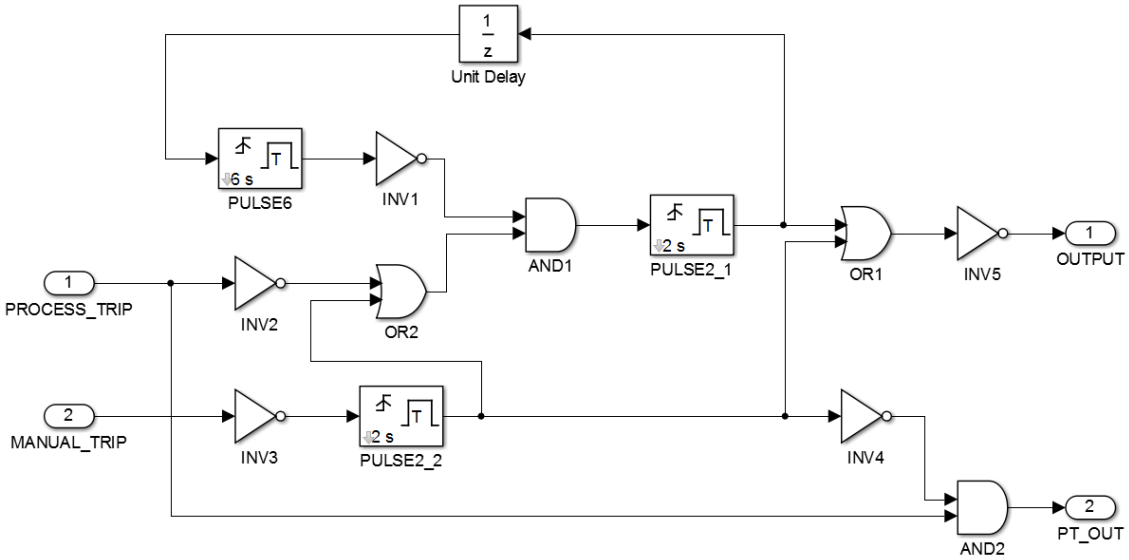

Figure 29: Simulink model of the Fast Stepwise Shutdown System (FSWS)

The diagram in Fig. 29 is similar to the detailed design logic circuit diagram of the FSWS illustrated in Fig. 14. The basic blocks such as the inverters, the AND-, and the OR-blocks are the same as in Fig. 14 but two other changes were made.

First, the pulse blocks were replaced by the ‘Monostable’-blocks found in Simulink. The Monostable-block implements the functionality of a monostable multivibrator which is exactly how the pulse blocks are supposed to work. Additionally, because the feedback from the output of the PULSE2_1 to the input of the PULSE6 caused an algebraic loop that Simulink could not solve as such, some delay was added to the feedback loop by using the ‘Unit Delay’-blocks of Simulink.

The other subsystems were implemented in the same way. The SSWS was very similar to the FSWS and the PL was constructed using the basic blocks. The R-latches that were used as the memory elements in the PL were also implemented using the ‘Subsystem’-block.

The amount of test cases was small because the purpose was only to try out different methods. The same test cases were used in the subsequent V&V phases and they were such that they could be used during the hardware tests as well i.e. they could be easily tested using the switches and the buttons on the Cyclone development kit.

The simulations were done ‘bottom-up’ i.e. first the subsystems were simulated individually and then the whole PLS was simulated. The individual blocks were not tested because they were provided by Matlab and thus needed no testing. Test cases were devised for each of the subsystems as well as the whole system. The test cases for the SSWS were:

1. PROCESS_TRIP is activated after 1 second
2. PROCESS_TRIP is activated after 1 second and deactivated after 2 seconds
3. PROCESS_TRIP is activated after 1 second and deactivated after 2 seconds, PROCESS_TRIP is activated after 8 seconds and deactivated after 9 seconds

The test cases verify the key features of the SSWS specified in the requirements specification. The temporal properties in the test cases refer to the absolute time after the start of the simulation. The results of test case 1 of the SSWS are illustrated in Fig. 30. The simulation was run for thirty seconds.

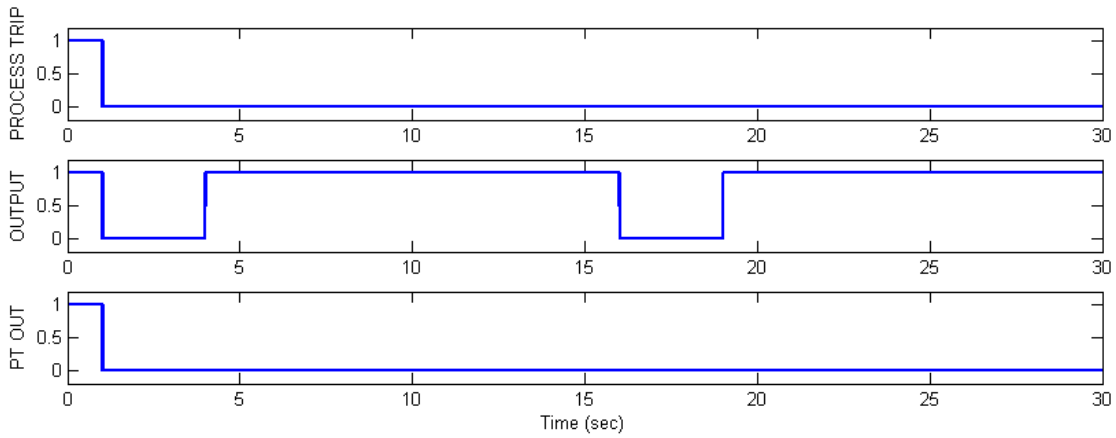

Figure 30: Results of the detailed design simulation of the test case 1 of the SSWS in Simulink

In Fig. 30, the three I/O signals of the SSWS are illustrated. The thirty-second run time is indicated by the x-axis in the figure. After one second, the active-low process trip is activated which initiates the fifteen-second sequence. Based on the simulations, the behaviour of the SSWS was deemed correct. After the simulations of the SSWS, the FSWS was simulated. The test cases were quite similar to the SSWS test cases.

However, the additional input MANUAL_TRIP increases the number of test cases. The test cases for the FSWS were:

1. PROCESS_TRIP is activated after 1 second
2. MANUAL_TRIP is activated after 1 second
3. PROCESS_TRIP is activated after 1 second and deactivated after 2 seconds
4. PROCESS_TRIP is activated after 1 second and deactivated after 2 seconds, PROCESS_TRIP is activated after 4 seconds and deactivated after 5 seconds
5. PROCESS_TRIP is activated after 1 second, MANUAL_TRIP is activated after 10 seconds and deactivated after 11 seconds

The FSWS test cases cover the key features that were required in the requirements specification. The design was simulated in Simulink and the results of test case 5 are illustrated in Fig. 31.

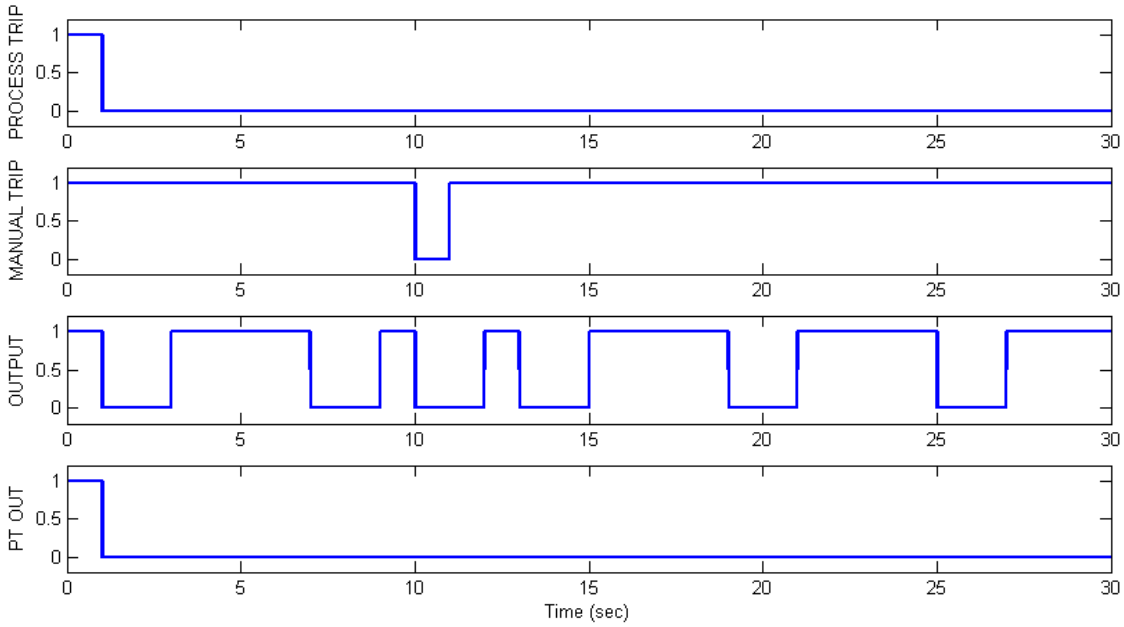

Figure 31: Results of the detailed design simulation of the test case 5 of the FSWS in Simulink

The I/O signals are illustrated now with the additional MANUAL_TRIP input in Fig. 31. The FSWS was also simulated for thirty seconds. Now, the PROCESS_TRIP is again activated after one second which initiates the six second operation sequence. After ten seconds, the MANUAL_TRIP is activated for one second, which initiates an additional two-second pulse in between the pulses of the basic sequence. The operation then continues normally, and overall the design seems to behave correctly.

The PL has more inputs but no temporal properties unlike the SSWS and FSWS. The required responses to all the different input combinations were specified in the requirements specification; there are sixty-four different ways that the six inputs can be combined. On a purely combinational design such as the PL, all input combinations could easily be tested. However, in this case study only a few of the key features were tested. The test cases for the PL are:

1. IN1 and EN1 active
2. IN2 and EN2 active
3. IN3 and EN3 active
4. IN1 active
5. EN1 active
6. IN2 active
7. EN2 active
8. IN3 active
9. EN3 active
10. IN1 and EN1 active, and IN2 and EN2 active
11. IN2 and EN2 active, and IN3 and EN3 active
12. IN1 and EN1 active, and IN3 and EN3 active

Since there are no temporal properties i.e. no long pulses or any other time-related functionality, multiple test cases could easily be simulated successively. In Fig. 32, the results of the simulation of the test cases 3 and 1 of the PL are illustrated respectively.

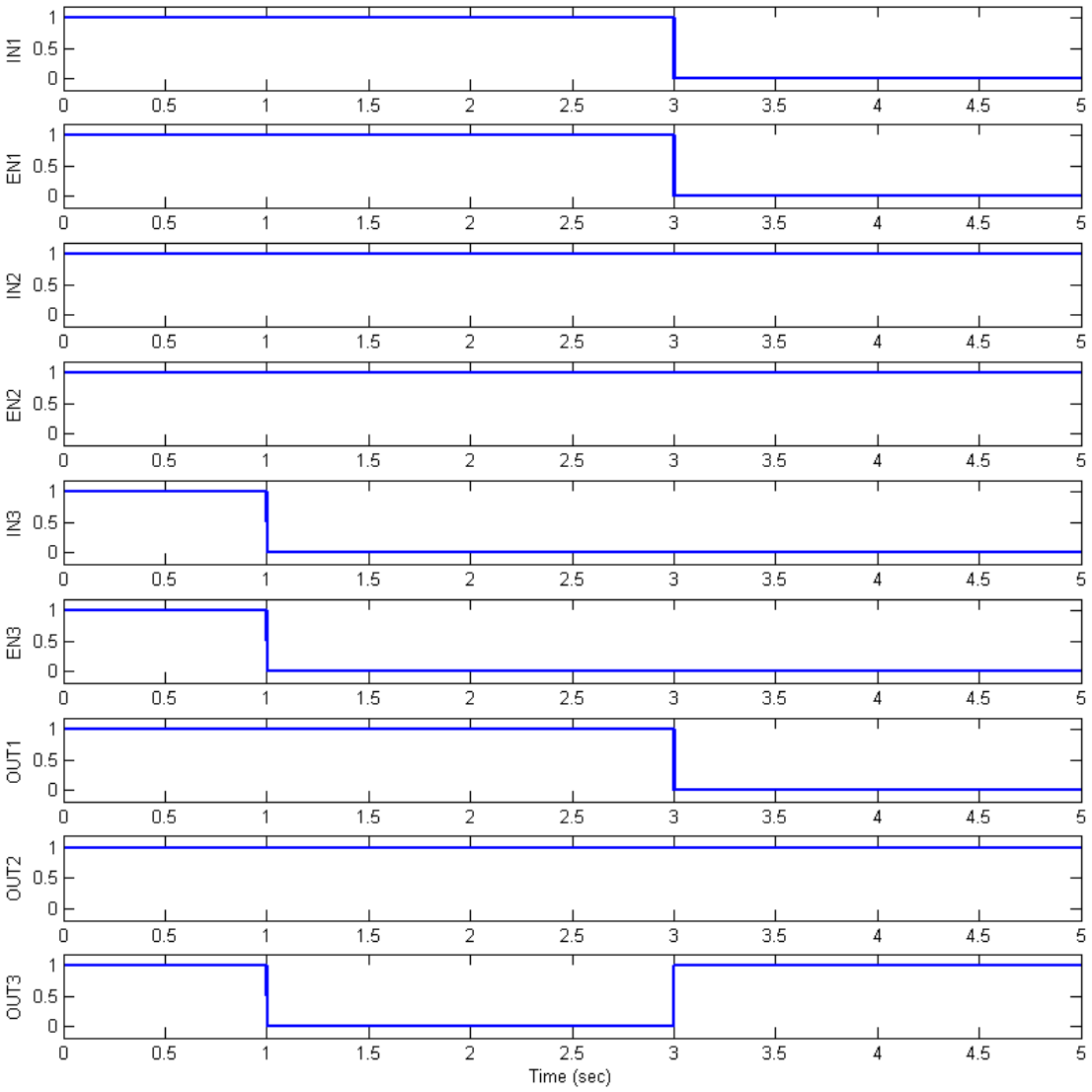

Figure 32: Results of the successive detailed design simulations of the test cases 3 and 1 of the PL in Simulink

In Fig. 32, all nine I/O signals of the PL are illustrated. Unlike the previous simulations, this one was only run for five seconds because there were no long pulses or other delays. First, the input signals IN3 and EN3 are activated after one second. This correctly activates the corresponding output OUT3. Then, after three seconds, the inputs EN1 and IN2 are activated which makes the output OUT1 active and OUT3 inactive, as required. Based on the simulations, the behaviour of the PL was correct.

Finally, the complete PLS was simulated. Test cases were devised for the PLS based on the requirements specification similarly to the subsystems. Thus, the PLS was simulated using the following test cases:

1. MANUAL activated after 1 second
2. ALARM_HIGH activated after 1 second
3. ACCELERATION activated after 1 second
4. ALARM_MED activated after 1 seconds
5. ALARM_HIGH activated after 1 second and deactivated after 2 seconds
6. ALARM_HIGH activated after 1 second, MANUAL activated after 15 seconds
7. ALARM_HIGH activated after 1 second, ACCELERATION activated after 4 seconds
8. ALARM_MED activated after 1 second, ALARM_HIGH activated after 2 seconds
9. ALARM_MED activated after 1 second and deactivated after 2 seconds, ALARM_MED re-activated after 8 seconds
10. ALARM_HIGH activated after 1 second and deactivated after 2 seconds, ALARM_MED activated after 3 seconds

Not all of the required features in the requirements specification can be verified using these test cases but they cover the key functionalities of the subsystems and the prioritisation. The results of test case 6 are illustrated in Fig. 33.

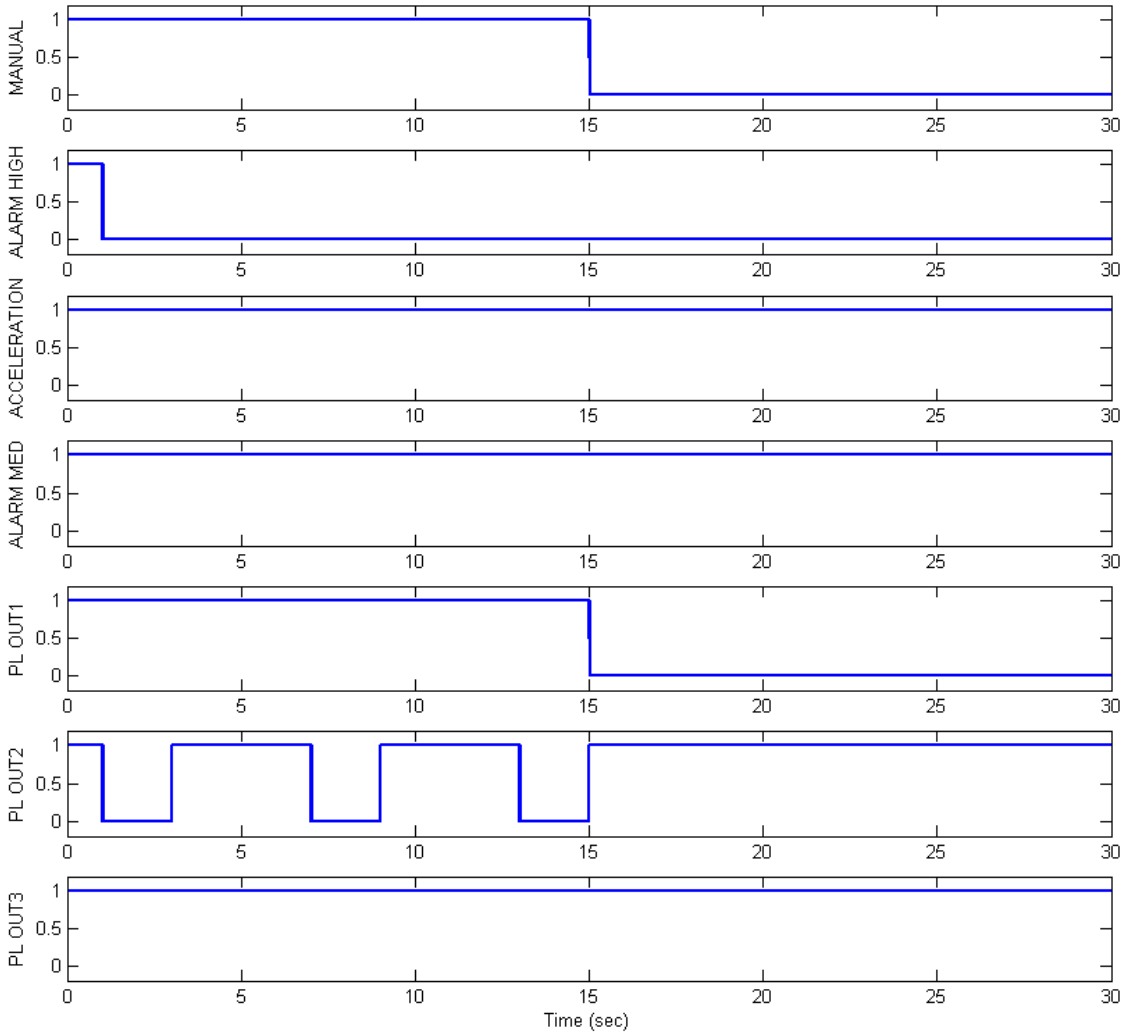

Figure 33: Results of the detailed design simulation of the test case 6 of the PLS in Simulink

The I/O signals of the PLS are illustrated in Fig. 33. The simulation was run for thirty seconds. In the test case, the input ALARM_HIGH is first activated after one second which initiates the functionality of the FSWS. This can be seen in the output PL_OUT2. Then, after fifteen seconds the input MANUAL is activated. The MANUAL is of a higher priority than the ALARM_HIGH and thus the higher PL_OUT1 is activated and the previously active PL_OUT2 is rendered idle. Based on the simulations, the behaviour of the PLS was also found correct.

In the detailed design simulations, the step time was one millisecond that is far longer than the actual step time of the device clock. The simulation would have taken too long to complete if the system had been simulated using a realistic nanosecond-scale step time. Therefore, a one millisecond step time was deemed sufficient due to the requirements that stated that there should be a response within one millisecond.

In detailed design V&V, some model checking was also done. A few requirements were tested on the PLS and the PL. The model checking tool was NuSMV (Cavada *et al.*, 2010) that has been used in other model checking projects at VTT (Lahtinen *et al.*, 2012). Models of the systems were created using the NuSMV input language, and the requirements were formalised using LTL.

The model for the PLS was created based on the logic circuit diagram in Fig. 12. The blocks were defined based on their functional descriptions i.e. they had nothing to do with the VHDL at this point. In this fully synchronous model, the whole circuit is updated at each time point unlike in the more detailed model checking models used in the subsequent phases. The model checker concluded that the design meets the specified requirements. As an example of LTL, below is one of the checked requirements written in natural language and the same requirement formalised in LTL.

- While Manual is inactive, while high Alarm is inactive, while acceleration is inactive, while Medium alarm is active, PS3 shall follow AS2 Output.

- ```
  G(! MANUAL &  ! ALARM_HIGH  & ALARM_MED -> (PL_OUT3 <->
  SSWS1.PT_OUT))
  ```

The PL was also checked in the same fashion. A model was built based on the logic circuit diagram in Fig. 15 in the NuSMV language. Again, only a couple of requirements were checked. As an example, one of the tested requirements is below written in both natural language and in LTL.

- while as2 output and as2 enable and ps3 are active, while MS output and MS enable are inactive, when as1 output and as1 enable are activated, ps3 shall be inactivated and ps2 shall be activated

- ```
  G(! PL_OUT3  & ! IN3 & ! EN3 & IN1 & EN1 & X(! IN2 & ! EN2 & IN1
  & EN1)
  -> X(PL_OUT3 & ! PL_OUT2))
  ```

The model checker concluded that the PL also meets the checked requirements. Models that are based on logic block diagrams have been checked using model checking before, so it was more interesting to try out model checking in the subsequent V&V phases.

Behaviour V&V

In the behaviour V&V, the main activity was the simulation of the VHDL code. A software tool called ModelSim by Mentor Graphics was used (Mentor Graphics, 2010). In order to do the simulations, an additional VHDL entity needed to be created to implement a test bench. A test bench VHDL entity typically has no I/O ports and contains instances of other VHDL entities that are tested. Different test cases are implemented inside processes where VHDL signals act as the I/O signals for the entity instances under test.

The actual clock cycle could not be used because the simulation would have taken too long. Instead, a hundred times slower clock was generated which was an improvement compared to the one millisecond cycle of the detailed design simulation. The same test cases were used in the behavioural simulation as in the detailed design simulation. In the following text, the results of the same test cases as in the detailed design simulations are illustrated.

Unlike in the detailed design simulation, the basic blocks were also simulated because they were manually implemented i.e. not ready-made like in Simulink. As an example of this, the results of the simulation of the two-input AND-block are in Fig. 34.

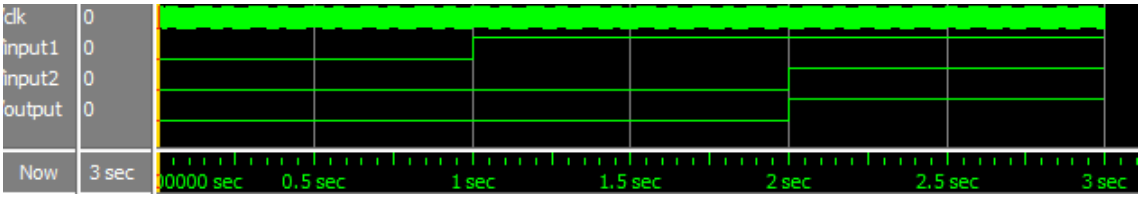

Figure 34: Results of the behavioural simulation of the basic functionality of the two-input AND-block in ModelSim

On the top of Fig. 34, the rapid clock signal (clk) can be seen, and the other I/O signals of the AND-block are below. The simulation was done using a generated 500 kHz clock and run for three seconds. Even a short simulation like this took over ten seconds to complete due to the rapid clock signal. After one second in the simulation, the first input is activated which has no effect on the output. Then, after two seconds, the second input is also activated, and now the output is activated as well. The block seems to function correctly i.e. implements the logical AND functionality.

After the individual blocks were tested, the subsystems were tested. The SSWS was tested first with the same test cases that were used in the detailed design simulation. The results of the test case 1 are illustrated in Fig. 35.

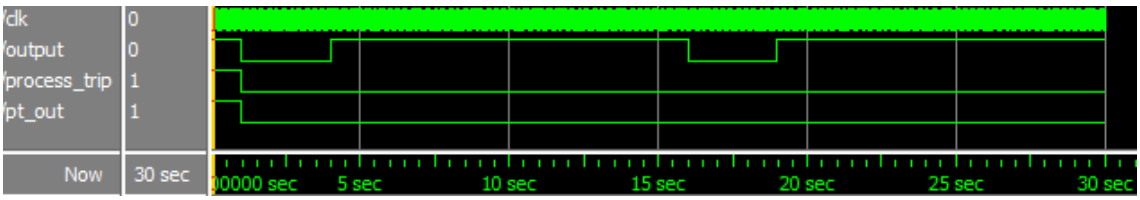

Figure 35: Results of the behavioural simulation of the test case 1 of the SSWS in ModelSim

The I/O signals in Fig. 35 behave similarly to the ones in the detailed design simulation in Fig. 30. Based on the simulations, the SSWS behaves correctly. A closer examination of the results in Fig. 35 reveals that there are delays of a few clock cycles between e.g. when the PROCESS_TRIP is activated and when the output is activated. This is illustrated in Fig. 36.

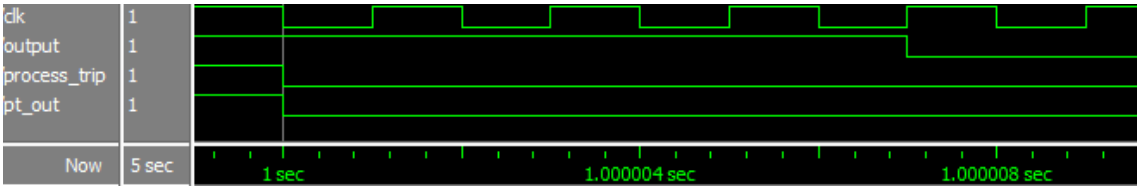

Figure 36: The response delay between the signals introduced by the synchronous design principle

The phenomenon in Fig. 36 occurs because the design is synchronous i.e. the blocks function only on the rising edges of the clock signal. This means that a signal propagates between the blocks only when there is a rising edge on the clock. Therefore, as the PROCESS_TRIP propagates through several blocks before affecting the OUTPUT, delays of a few clock cycles are introduced between the signals. Luckily these few clock cycle "sync delays" are meaningless with regard to the required response time of one millisecond.

It should be emphasised that these delays have nothing to do with the actual hardware-related propagation delays in the actual components. Those delays have not yet been modelled in this stage. First such delays, although still not completely realistic, are introduced in synthesis.

After the simulations of the SSWS, the FSWS was similarly simulated. The same test cases were used as in the detailed design simulation and the results of the simulation of the test case 5 are illustrated in Fig. 37.

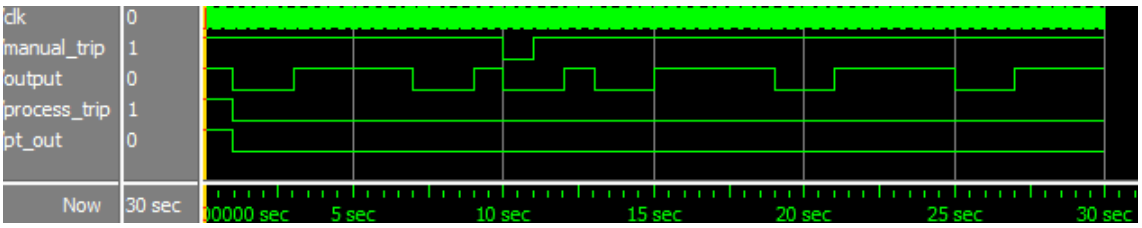

Figure 37: Results of the behavioural simulation of the test case 5 of the FSWS in ModelSim

In Fig. 37, the I/O signals of the FSWS behave similarly to the corresponding detailed design simulation results in Fig. 31. The clock frequency was the same 500 kHz used in the SSWS simulations, and the simulation was run for thirty seconds. The six-second operation sequence is correct and the additional pulse is correctly initiated when the MANUAL_TRIP is activated after ten seconds. Based on the simulations, the FSWS behaves correctly.

The third subsystem, PL, was tested next. The test bench was configured to run the test cases successively similarly to the detailed design simulation. It would be wise to test each of the test cases also individually because there is no guarantee that the internal state of the PL is correct after each test case. In Fig. 38, the results of the simulation of the test cases 3 and 1 are illustrated.

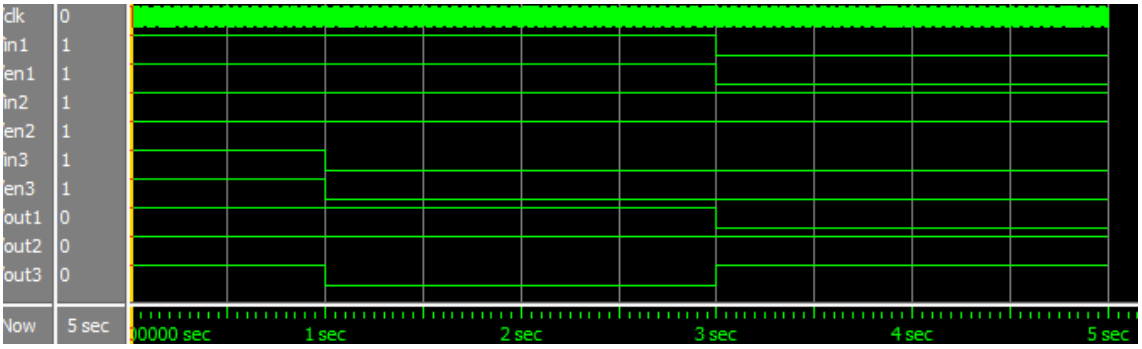

Figure 38: Results of the behavioural simulation of the test cases 3 and 1 of the PL in ModelSim

The signals in Fig. 38 look similar to the ones in the detailed design simulation in Fig. 32. The simulation was run using the 500 kHz clock and for five seconds. Like the previous subsystems, the PL also behaved correctly based on the simulation results.

After the behaviour of the subsystems was deemed correct, it was time to test the whole system. The results of the behavioural simulation of the test case 6 of the PLS are illustrated in Fig. 39.

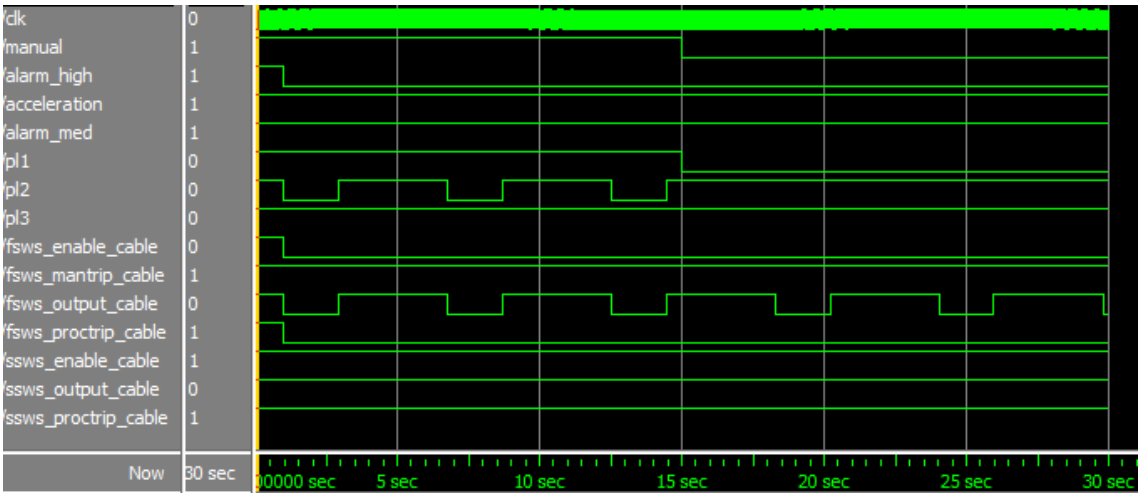

Figure 39: Results of the behavioural simulation of the test case 6 of the PLS in ModelSim

In Fig. 39, there are more signals than in the detailed design simulation in Fig. 33. This is because the I/O ports representing the pins of the cable used in the communication between the devices were also modelled. The ports representing the cable pins are the ones below the 'pl3' and have a '_cable' suffix. The different delays due to the synchronous design can be seen in Fig. 40.

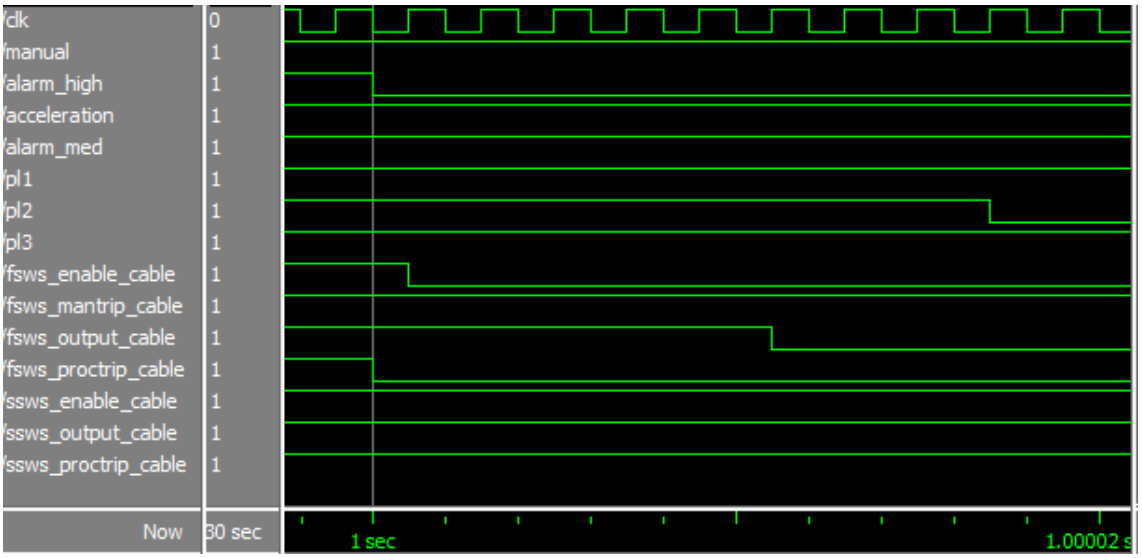

Figure 40: Delays introduced to the PLS simulations due to the synchronous design principle

It is interesting to see how the signals propagate in the system. First, ALARM_HIGH is activated and, on the same rising edge of the clock, the FSWS_PROCTRIP_CABLE that represents the pin where the ALARM_HIGH is conveyed to the FSWS is also activated.

The FSWS_ENABLE_CABLE is activated on the next clock cycle as it only goes through the FSWS without any special processing. It takes a while for the FSWS to form the output and after six cycles after the first activation of the ALARM_HIGH, the FSWS_OUTPUT_CABLE becomes active. Then, the PL processes the signals and after nine clock cycles after the initial activation of the ALARM_HIGH, the output PL2

finally becomes active. This does not violate the one millisecond response time requirement but it is still good to know that such a delay exists.

A Tcl script that was used in the previous case study (Lötjönen, 2012) was modified to enable the use of external input files as test inputs in the PLS as well. This enables more comprehensive testing as the test input vectors can be in a more generic form, and automatically processed in the test bench. However, this was not utilised in this case study because the aim was not to do comprehensive testing, but it could be used in future studies.

The behaviour of the PLS was also correct based on the simulation results. In the actual VHDL implementation there were still the higher level entities CYCLONE and IGLOO which were not simulated because they are merely used to house the already tested lower level entities. They were simulated in synthesis simulation because the design in synthesis is already on its way to becoming an actual FPGA application and should be realistic.

Model checking was also done in the behaviour V&V. The models of the same systems, the PLS and the PL, were checked as in the detailed design V&V. Also, the same requirements were checked as in the previous phase. Now, models of the systems were built based on the VHDL source code using the NuSMV input language. The VHDL code itself could not be used as the model in this case but in future studies this should be explored.

The models are rather similar to the ones built in the detailed design V&V phase but now the blocks operate similarly to the ones in the VHDL description meaning that each individual function block in the model takes one time point to update its outputs based on the inputs. Similarly to the simulations, a significantly slower time cycle had to be used because the checking would have taken too long when using a realistic time cycle. A time cycle of one hundred milliseconds was used.

The model checking concluded also in this stage that the designs meet the checked requirements. However, as the models are not necessarily formally equivalent to the VHDL models, the results may not be as convincing as if the VHDL models were used as such. Nevertheless, the model checking in the behavioural description V&V was deemed successful.

Synthesis V&V

In synthesis V&V, initially the main focus was on gate-level simulation. A quick review of the gate-level schematic diagrams did not reveal any errors introduced during the synthesis procedure. Because the design is very simple and as the design does not contain redundant blocks i.e. nothing for the synthesis tool to omit, it was to be expected that errors would not be introduced during synthesis.

As the synthesis design phase results in device-dependent products, the CYCLONE and the IGLOO entities could no longer be tested together as they reside on different hardware. This means that at least the complete PLS could not be tested through gate-level simulation.

Additionally, it turned out that for some reason, the pulse-blocks in the SSWS and FSWS did not function during gate-level simulation. As a result, the entire IGLOO could not be simulated. However, because the design should have been simulated using a realistic clock frequency, simulation would have been impossible either way because a slower clock signal would have been necessary in order for the simulations to be completed in a sensible time.

Simulating the design using a slower clock frequency defeats the purpose of the gate-level simulation as the purpose is to show that the behaviour of the design that is to be transformed into an FPGA application has not changed during synthesis. Therefore, after the behaviour of the design would have been deemed correct during the simulation using a slower clock, the synthesis should have been performed again using a realistic clock in order for the output product to be usable in the next phase in the design process. As the synthesis would now be performed again, the simulation results acquired previously using the slower clock could not be considered evidence of the correctness of the newly synthesised design.

Unfortunately, problems arose in the gate-level simulation of the CYCLONE as well. Although there would not have been problems with the time as the purely combinatorial Cyclone without the serial transmission entity could have been simulated using a realistic clock, the ModelSim version that was used was provided by Actel and could not be used for simulating the CYCLONE gate-level netlist synthesised for the Altera FPGA.

A version of ModelSim provided by Altera would have been available on their web site but not for the version of the Altera Quartus II software tool that was used in the design. Therefore, in future studies, the selection of the tools should be made in such a way that these issues can be avoided.

In the formal verification, only model checking could be done in synthesis V&V. The tools needed for the LEC were not available and thus it could not be conducted. Similarly to the detailed design and behavioural V&V phases, models were built based on the gate-level schematic diagrams output by the software tools. This time, however, a model of the PLS could not be constructed because the gate-level schematic diagram was too complex.

The complexity is caused by the pulse blocks in the FSWS and the SSWS. In the pulse blocks, large integers are needed as counters to manipulate time which means that many d-flip-flops are needed to implement the memory elements needed for storing and operating with the large integers. Intricate structures such as large timers easily lead to state explosion in the reachable space of the model checking model. Consequently, property verification on these models becomes infeasible.

In future studies, either the gate-level schematic should be properly exported or the use of the other output product, the EDIF netlist, should be looked into. The difficulties that were faced are in line with the previous experience of challenges regarding the manipulation of time when using FPGAs (Lötjönen, 2012). It makes simulation and apparently also model checking difficult when there is a need to spend time in the second scale.

The PL was checked successfully as it does not contain anything else than combinatorial logic blocks. The gate-level schematic is fairly similar to the logic circuit diagram; the synchronous blocks in the model are implemented so that the combinatorial part is performed instantaneously, and the result is assigned to a d-flip-flop whose output is updated after one clock cycle.

Although the synthesised gate-level schematic is logically equivalent to the descriptions in the previous phases, the synthesis tool removed one of the input ports of the PL. The removed port was the IN1 input. Although this seems alarming, when the logic circuit diagrams are inspected, it can be seen that the IN1 is after all redundant because both the IN1 and EN1 originate from the same signal MANUAL i.e. their values are always the same. If they could differ, the synthesis tool would most likely have left both ports intact.

The logical equivalence between the model of the PL based on the VHDL code and the model based on the gate-level schematic was verified by using the model checker. If only the PL entity was synthesised as a separate project, both ports would be preserved because there is no knowledge of where the signals originate from and could very well have different values.

Place and route V&V

As part of the place and route V&V, a review of the physical layout models of the CYCLONE and IGLOO was done which did not reveal any errors as there were no special requirements for the configuration and placement and of individual CLBs. Therefore, the only thing that needed to be verified was that the behaviour of the designs had not changed as a result of the PAR.

The software tools apparently performed some sort of static timing analysis during the PAR and some reports were provided containing information about the overall propagation delays etc. Without any deeper analysis, the overall propagation delays seem to be in the range of tens of nanoseconds which is good because the required response time was one millisecond. More emphasis on the timing analyses should be put in future studies especially if the systems then are more critical with respect to the timings.

The equivalence of the technology-level models with the gate-level schematics was of interest. As expected, the technology-level simulation was unsuccessful because the version of ModelSim that was used could not be used for the CYCLONE physical layout design mapped for the Altera FPGA.

The formal verification for the technology-level netlist or the physical layout model could not be conducted. Again, the lack of tools prohibited LEC, and the physical layout model could not be exported in such a way that would have enabled the creation of an SMV model for the model checking. The utilisation of the netlist and the use of the physical layout model in model checking should be looked into in future studies.

Configuration V&V

As the designs inside the two FPGAs were strongly dependent on each other, the FPGA testing that is supposed to be part of configuration V&V was done as part of integration V&V. In other words, hardware tests using the previously presented test cases were performed only after both FPGAs had been configured and connected together using the parallel cable.

The other part of configuration V&V, the verification of the configuration was not extensive because there were limitations in the read-back capabilities of the FPGAs. The configuration of the IGLOO could not be read back altogether (Microsemi, 2012b), and from the Cyclone, only the configuration of the non-volatile configuration memory could be read back. The actual configuration of the FPGA could not be read back.

The configuration was read back from the non-volatile configuration memory of the Cyclone development kit. The length of the file was 524,472 bytes and it had an eight-byte checksum. The file originally used for the configuration was one byte longer but had the same checksum. To inspect the bit streams, the configuration files were first converted into raw programming data (.rpd) files using the Quartus II, and then compared to each other to find differences using a tool called vBinDiff (Madsen, 2008). No differences between the files were found.

Integration V&V

In integration V&V, the functionality of the overall system was tested. The input signals were given using the switches and the buttons on the Cyclone development kit highlighted in Fig. 9. The outputs could be observed both by using the seven-segment displays on the Cyclone development kit and by using Matlab to process the serial data transmitted by the Cyclone development kit.

The same test cases were used as in the simulations although only the ones concerning the whole system could be tested because the subsystems were now packaged inside the CYCLONE and IGLOO entities. Some of the functionalities of the test cases of the individual subsystems could be tested on the entire system by only manipulating the inputs that initiate the particular subsystems. The functionality of the test case 1 of the SSWS was first tested, and the results are illustrated in Fig. 41.

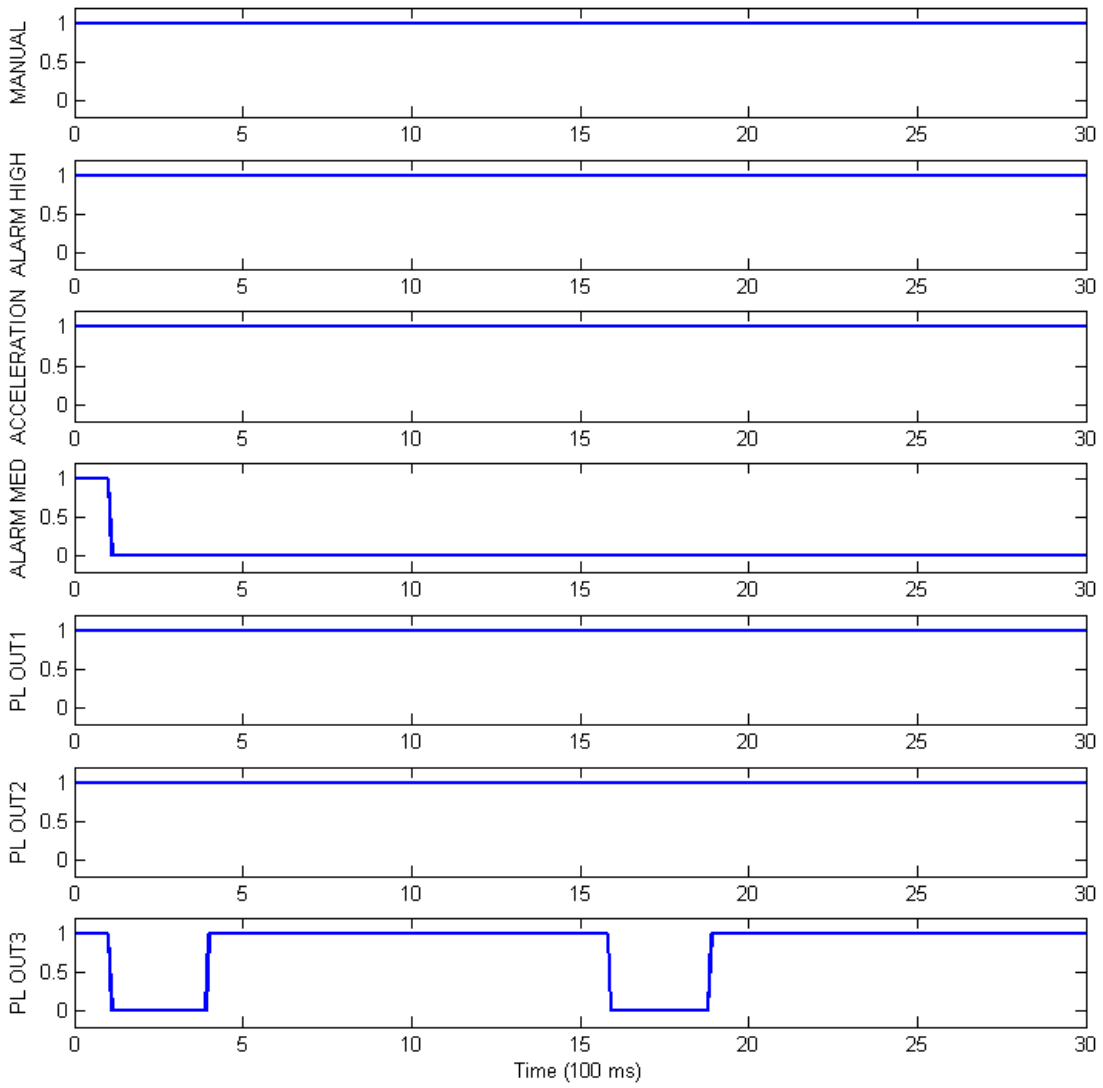

Figure 41: Results of the hardware tests of the functionality of test case 1 of the SSWS in Matlab

The results in Fig. 41 look rather similar to those in the detailed design simulation of the SSWS in Fig. 30 although the signals are now the I/O signals of the PLS. The transients are not as straight vertically as in the previous tests, which makes them look almost like

analogue signals. However, this is only an illusion caused by the low one hundred-millisecond resolution of the serial transmission.

Although the requirements specification required a one millisecond response time, it could not be tested because of the slow processing of the serial data in Matlab. The cycle time of one hundred milliseconds was used i.e. the serial transmitter in the Cyclone development kit was configured to transmit ten bytes in a second. A ten millisecond cycle was tried first but the serial data became distorted for an unknown reason. In future studies, better serial communication interface should be developed.

As the inputs were given manually by using the switches and buttons, the timings were not precise, so the results of the FPGA testing are merely indicative. In future studies, the serial transceiver could be made to not only transmit the signals but also to receive inputs generated e.g. by a PC. The signal values were also exported into a file that can be inspected e.g. using Microsoft Excel. A part of the output file of the first test case is in Table 2.

Table 2: Part of the output file of a hardware test of the functionality of the test case 1 of the SSWS

	MANUAL	ALARM_HIGH	ALARM_MED	ACCELERATION	PL_OUT1	PL_OUT2	PL_OUT3	CHECKBIT
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1
12	1	1	0	1	1	1	0	1
13	1	1	0	1	1	1	0	1
14	1	1	0	1	1	1	0	1
15	1	1	0	1	1	1	0	1
16	1	1	0	1	1	1	0	1

In Table 2, the time runs on the leftmost column. The resolution is one hundred milliseconds so e.g. the numeral value ‘10’ means 1000 ms i.e. one second. The rest of the columns represent the value of the signal indicated by the first row of the columns. In the first test case, the ALARM_MED was activated after one second. In Table 2, this is indicated by the value of ALARM_MED changing from ‘1’ to ‘0’ at time ‘12’ i.e. after 1.2 seconds.

The deviance from the 1.0 seconds is explained by both the inaccuracy of the manual input and the low resolution serial transmission. At the same time however, the PL_OUT3 is correctly activated. The exported files were not exploited any further but in future studies they could be used to do more detailed analysis of the hardware tests. However, this would require a significantly higher resolution and a way to use the serial port to also send inputs to the FPGA.

The rest of the testing concentrated on the PLS test cases. Based on the meagre hardware tests, the complete integrated system seemed to behave correctly. To make

sure that the system could function even a little longer, the system was left performing the FSWS functionality for a weekend. On the following Monday, the system was still functioning correctly. As a curiosity, it was reported that there was also a power outage during the weekend.

An interesting question related to this is that if an FPGA-based passive safety system was used in an NPP, and the FPGA was SRAM-based, how could the correctness of the re-configuration that follows a power outage be verified? In the case study, the system simply restarts and the functionality can be inspected but a passive safety system would not reveal the error as the system is only used on-demand. A read-back of the actual configuration in the FPGA is required to verify its correctness.

Despite operating mostly correctly, some anomalies were detected in the operation of the integrated system. When the FPGAs are powered up, the seven-segment displays sometimes show that one of the outputs is active (PL_OUT2 most frequently. This is probably caused by some of the d-flip-flops being in an arbitrary state after power-up. An asynchronous reset should have been designed into the system to avoid this. The reset would be enabled during power-up and then disabled after a specified time to start up the functionality correctly.

Additionally, when the interconnecting cable is disconnected from either end during operation, the active-low principle used in the signals should cause the highest priority output (the FSWS output) conveyed via the cable to become active. This does not happen however. Furthermore, sometimes when the cable is connected back during operation, the PL_OUT2 initiated normally by the FSWS becomes active for two seconds. The connecting of the cable must cause a voltage spike that is interpreted as an activation of the signals in the cable.

4.5 Summary of the case study

The PLS case study was partly successful and partly unsuccessful. The design process described in Section 2.3 was successfully followed through starting from the requirements and ending up with an apparently correctly functioning FPGA system. Unfortunately, some of the V&V methods described in Section 2.4 could not be tested due to lack of knowledge, experience, and the appropriate software tools. An illustration of what V&V activities were performed and what were not is in Table 3. The table is the same as Table 1 except that now the activities are coloured according to whether they were performed or not.

Table 3: Activities performed (green) and not performed (red) in the different phases of the V&V process of the PLS case study

	Review	Formal Verification	Testing
Requirements V&V	Requirements review, Formalised requirements review		
Architecture V&V	Architecture review		Simulation
Detailed Design V&V	Logic circuit review, Hardware specification review	Model checking	Logic Circuit Simulation
Behaviour V&V	HDL code review	Model checking	RTL simulation, static analyses
Synthesis V&V	EDIF netlist review, schematic diagram review	Model checking, logic equivalence checking	Gate-level simulation
Place and Route V&V	Physical layout model review	Model checking, logic equivalence checking	Technology-level simulation, static timing analysis
Configuration V&V			FPGA testing, verification of the configuration
Integration V&V			Operational tests, circuit board tests

The activities coloured green in Table 3 are the ones that were performed. One of the objectives, to get to FPGAs communicating with each other, was successfully completed. Also, the serial connection was established successfully. The testing of the V&V methods that could not be tested in this case study can be carried out in future studies.

The PLS case study provided verification for the impression that special knowledge about both software and hardware design is indeed required in FPGA design. Many of the challenges rose mainly from the lack of knowledge of the hardware-related details of the FPGA technology. With more knowledge and proper tools, many aspects of the design and V&V processes and the system itself can be improved and made more realistic. This offers a good basis for future studies.

5 Discussion

The design and V&V processes described in Chapter 2 were combined from different sources and presented in such a way that they should be understandable even to non-experts. This type of a wide collection of methods is rarely seen in other literature so the descriptions of the processes may prove useful to the reader when compiling an overall impression of what types of activities are typically included in the FPGA design and V&V processes.

There may be additional design and V&V activities to those described in Chapter 2. The activities related to different design and V&V methods described in the chapter are the ones most often found in the literature. A more comprehensive study of the methods and whether there still are some relevant activities omitted from this work should be conducted. However, the design and V&V processes illustrated in Chapter 2 should provide a good overview of their nature and using e.g. the V&V process should provide a good amount of evidence of the correctness of given applications.

The formal verification methods, especially model checking, are widely used in the IC industry for hardware verification. Model checking is also used in the verification of software-based systems but, as especially the later stages of the FPGA design process yield products similar to the ones in IC design, model checking may prove more applicable for verifying the products than currently perceived. This is a good subject for future studies.

As there is a need for internationally consistent guidance regarding the design and V&V of FPGA-based applications in NPP safety automation, the topics of Chapter 2 could be looked into more deeply and based on that, an NPP-related guide for FPGA design and V&V could be put together. The guide could incorporate Finnish regulations regarding digital I&C technology in NPPs and reflect on how, using the different design and V&V methods, those regulations could be satisfied using FPGA-based systems. As there are new NPP projects and modernisations starting and on-going in Finland, there seems to be demand for a guide regarding how the Finnish regulations in particular could be met by using FPGA as the implementation technology for the safety automation.

Also included in the guide could be an elaboration of the capabilities, advantages, and disadvantages of using FPGAs instead of software-based systems. The suitability and the advantages and disadvantages were explored in Chapter 3 of this document. However, many of the claims introduced in the chapter should be tested in order to validate them. For example, tests could be made where the performance of an application implemented in both software and FPGA is compared.

Especially the challenges faced during the V&V process of the system in the case study justified the impression that the FPGA design and V&V processes are more complex than the software processes. Knowledge of the hardware-related details is indeed required especially in the later phases of the design and V&V processes. This has been widely seen as one of the disadvantages of the FPGA technology because the experts in both NPP safety automation and FPGA are scarcer than their software counterparts.

As reported at the end of Chapter 4, many of the V&V methods such as gate- and technology-level simulation and LEC described in Chapter 2 could not be tested in the case study. Also, the VHDL design was done completely by manually writing the source code. In actual projects, reportedly the code is many times generated from a higher-level description, so it would be appropriate to employ this in future studies and see what kinds of new V&V requirements and challenges arise.

The problems regarding the spending of time seem to have not been reported in the literature previously. Therefore, it can be seen as an important result of this work that when systems are needed to deal with seconds or even longer periods of time, there are significant challenges in the V&V of those designs when using FPGAs. Many actual NPP safety functions deal with long periods of time, so some kind of solution to the time-problem should be found.

Regarding the V&V, for future studies proper tools that enable all the activities described in Chapter 2 and more knowledge of the specifics related to the activities need be acquired. Also in future studies, designs should be comprehensively tested using the methods in order to get a real impression of whether the methods yield credible evidence of the correctness of the design. To make the design process more realistic, also diverse design and V&V teams should be used.

In hardware testing, the test inputs should be fed into the FPGA by some other method than the imprecise manual use of switches and buttons on the development kits practised in the case study. The serial connection provides a way to feed arbitrary input vectors generated by a PC to the FPGA. Also, the exportation of the test results into files in a generic format enables e.g. statistical testing and analysis of the results.

The serial connection could also be used to connect the FPGA to a simulator running a model of an NPP. Then, different functions representing different subsystems of the NPP could be implemented and tested on the FPGA. Additionally, a software-based system running on a simple microcontroller performing the same function as the FPGA could be connected in parallel. The system would then mimic a realistic system where the FPGA is used as diverse backup for the primary software-based system.

6 Conclusions

As a result of this work, good models of the design and V&V processes were obtained by combining models found in several sources in literature. Although the design process seems to be sufficient, the V&V process will probably be expanded in the future as more V&V activities are identified. However, by using the methods that were identified in this work, a good amount of evidence of the correctness of an application can be acquired.

The increasing amount of operational FPGA-based systems in NPPs around the world indicates that the FPGA technology is suitable for use in the systems also in practice as well as in theory. However, the complexity of the design process and the need to have knowledge of the hardware-related specifics are real disadvantages although the actual application is less complex than the software counterpart.

The design of the case study system was successful as the system turned out just the way originally intended. The communication between the two different FPGA devices was implemented successfully and the system fulfils the requirements devised for the system. The explicit description of the whole design process that was followed through in the case study is useful to anyone who is interested in FPGA design. As the design consists of many phases, the practice of using the same design blocks as the examples in illustrating different phases helps the reader to understand what happens in the phases and what the differences between them are.

Although the design process itself was successful, several of the V&V activities in Table 1 could not be conducted due to the lack of proper tools and knowledge, and the problems with the time. An important result of the case study was the notion that when there is a need to deal with time in the second-scale such as in the pulse blocks in the case study, many problems arise in the V&V of such designs. As the clock frequencies of FPGAs are typically in the nanosecond-scale, implementing timers in the second-scale requires implementation of special integer counters. In FPGAs, there are no dedicated timer devices such as those found in many microcontrollers.

The fast clocks caused problems in the simulations and model checking. While the behavioural description simulations could be done using a slower clock, the gate- and technology-level simulations were not meaningful to do with a slower clock because at those phases the design should be such that it can be transformed into hardware. The purpose of these simulations was to verify that there were no errors introduced during the automated synthesis and PAR procedures and thus the design should be realistic i.e. such that reflects the intended final product.

As an additional result of the case study, many subjects for future studies emerged which can also be seen as one of the objectives of case studies in general. If the FPGA technology continues to gain ground in the nuclear power industry, there will be an increasing demand for new FPGA-related studies for which subjects can be looked for in this thesis.

7 References

Actel, 2010 — Actel Corporation. *Libero IDE v9.1 User's Guide*. Actel, 2010

Altera, 2006 — Altera Corporation. *Cyclone II FPGA Starter Development Board Reference Manual*. Version 1.0. San Jose, CA, 2006

Altera, 2011 — Altera Corporation. *Introduction to Quartus II Software.* Altera, 2011. Available: http://www.altera.com/literature/manual/quartus2_introduction.pdf (accessed: 22.4.2013).

Arndt *et al.*, 2012 — Ardnt, S. A., Dittmann, B. F., Dacruz, P., Glöckler, O., Naser, J., Nguyen, T., Salaun, P. Current Issues Associated with the Implementation of Field Programmable Gate Arrays in the Nuclear Power Industry. *The proceedings of the NPIC-HMIT 2012*. pp. 1738-1745, San Diego, California, USA, July 22-26 2012.

Ashenden, 1996 — Ashenden, P. J. *The Designer's Guide to VHDL*. ISBN 1-55860-270-4. Morgan Kaufmann Publishers Inc., California, USA, 1996

Bacon, 1998 — Bacon, J. *Concurrent Systems – Operating Systems, Database and Distributed Systems: An Integrated Approach*. Second edition. ISBN 0-201-17767-6. Addison-Wesley, UK, 1998

Balboni *et al.*, 1994 — Balboni, A., Mastretti, M., Stefanoni, M. Static Analysis for VHDL model Evaluation. *ACM 1994*, pp. 586-591, 1994

Björkman *et al.*, 2009 — Björkman, K., Frits, J., Valkonen, J., Heljanko, K., Niemelä, I. *Model-Based Analysis of a Stepwise Shutdown Logic*. MODSAFE 2008 Work Report. ISBN 978-951-38-7176-5. VTT, 2009

Bobrek *et al.*, 2007 — Bobrek, M., Woot, R. T., Ward, C. D., Killough, S. M., Bouldin, D., Waterman, M. E. Safe FPGA Design Practices for Instrumentation and Control in Nuclear Plants. *IEEE HPRCT 2007*. Monterey, California, USA, 2007

Boussinot, 1991 — Boussinot, F., De Simone, R. The ESTEREL Language. *Proceedings of the IEEE, Vol 79, No 9*. pp. 1293-1304, 1991

Burch *et al.*, 1992 — Burch, J., Clarke, E., McMillan, K., Dill, D., Hwang, L. *Symbolic model checking: 10^20 states and beyond.* Information and Computation 1992;98(2) pp. 142–70, 1992

Cavada *et al.*, 2010	Cavada, R., Cimatti, A., Jochim, C. A., Keighren, G., Olivetti, E., Pistore, M *et al. NuSMV 2.5 User Manual*. FBK-irst, 2010.
Clarke *et al.*, 1999	Clarke, E. M., Grumberg, O., Peled, D. A. *Model Checking.* ISBN 978-026-20-3270-4. The MIT Press, 1999
Daumas *et al.*, 2012	Daumas, F., Druilhe, A., Thuy, N. Justification Framework for the Formal Verification of a Microprocessor FPGA Emulator Application to the MC6800 µp. *5th FPGA IAEA Workshop*, Beijing, 2012
Doulos, 2013	Doulos. *The Designer's Guide to PSL.* Available: http://www.doulos.com/knowhow/psl/ (accessed 22.4.2013).
Druilhe *et al.*, 2010	Druilhe, A., Daumas, F., Nguyen, T. Formal Verification of an FPGA Emulation of the Motorola 6800 Microprocessor. *NPIC&HMIT 2010.* pp. 1316-1325, November 7-11, Las Vegas, Nevada, USA, 2010
EPRI, 1995	Electric Power Research Institute. EPRI TR-103331: *Guidelines for the Verification and Validation of Expert System Software and Conventional Software.* Volume 2: Survey and Assessment of Conventional Software Verification and Validation Methods (Revision 1). EPRI, Palo Alto, California, USA, 1995
EPRI, 2009	Electric Power Research Institute. EPRI TR-1019181: *Guidelines on the Use of Field Programmable Gate Arrays (FPGAs) in Nuclear Power Plant I&C Systems.* EPRI, Palo Alto, California, USA, 2009
ESA, 1994	European Space Agency – European Space Research and Technology Centre. *VHDL Modelling Guidelines.* Issue 1. Noordwijk, Netherlands, 1994
Fix, 2008	Fix, L. *Fifteen years of formal property verification in Intel. 25 Years of model checking*. Lecture Notes in Computer Science 2008;5000, pp. 139–44, 2008
Frhed, 2009	Frhed free hex editor. Available: http://frhed.sourceforge.net/en/ (accessed 22.4.2013). Latest version was released in 2009
Gaisler, 2002	Gaisler Research, Sandi Habinc. *Lessons Learned from FPGA developments*. Göteborg, Sweden. 36 p., 2002

IAEA, 1999 — International Atomic Energy Agency. *Verification and Validation of Software Related to Nuclear Power Plant Instrumentation and Control.* Technical Reports Series No. 384. IAEA, Vienna, Austria, 1999

IAEA, 2000 — International Atomic Energy Agency. NS-G-1.1: *Software for Computer Based Systems Important to Safety in Nuclear Power Plants.* IAEA, Vienna, Austria, 2002

IAEA, 2002 — International Atomic Energy Agency. NS-G-1.3: *Instrumentation and Control Systems Important to Safety in Nuclear Power Plants.* IAEA, Vienna, Austria, 2002

IAEA, 2007 — International Atomic Energy Agency. *IAEA Safety Glossary – Terminology Used in Nuclear Safety and Radiation Protection*. IAEA, Vienna, Austria, 2007

IAEA, 2013 — International Atomic Energy Agency. *Application of Field Programmable Gate Arrays (FPGAs) in Instrumentation and Control Systems of NPPs*. Available: http://www.iaea.org/NuclearPower/News/2013/2013-02-15-npe.html (accessed 22.4.2013)

IEC, 2012 — International Electrotechnical Commission. IEC Standard 62566 Ed. 1: *Nuclear power plants – Instrumentation and control important to safety – Development of HDL-programmed integrated circuits for systems performing category A functions*. IEC, 2012

IEEE, 1996 — Instititute of Electrical and Electronics Engineers. IEEE 91-1984: *IEEE Standard Graphic Symbols for Logic Functions.* IEEE, New York, USA, 1996

IEEE, 2004 — Instititute of Electrical and Electronics Engineers. IEEE 1086.6-2004: *IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis.* IEEE, New York, USA, 2004

IEEE, 2006 — Institute of Electrical and Electronics. IEEE 1364-2005: *IEEE Standard for Verilog® Hardware Description Language.* IEEE, New York, USA, 2009

IEEE, 2007 — Instititute of Electrical and Electronics Engineers. IEEE 1800: *IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language.* IEEE, New York, USA, 2007

IEEE, 2009 — Institute of Electrical and Electronics. IEEE 1076-2008: *IEEE Standard VHDL Language Reference Manual.* IEEE, New York, USA, 2009

IEEE, 2010	Institute of Electrical and Electronics Engineers. IEEE 1850-2010: *IEEE Standard for Property Specification Language.* IEEE, New York, USA, 2010
IEEE, 2012	Instititute of Electrical and Electronics Engineers. IEEE 1666-2011: *IEEE Standard for Standard SystemC® Language Reference Manual.* IEEE, New York, USA, 2012
ISO/IEC, 2012	International Organization for Standardization/International Electrotechnical Commission. ISO/IEC 8652: *Information technology – Programming languages – Ada.* ISO, 2012
Lahtinen *et al.*, 2012	Lahtinen, J., Valkonen, J., Björkman, K., Frits, J., Niemelä, I., Heljanko, K. Model-checking of safety-critical software in the nuclear engineering domain. *Reliability Engineering and System Safety 105.* pp. 104-113, Elsevier, 2012
Lötjönen, 2012	Lötjönen, L. *FPGA implementation of the Stepwise Shutdown System*. VTT Research report VTT-R-06053-12. 73 p., VTT, Espoo, 2012
Madsen, 2008	Madsen, C. J. *vBinDiff – Visual Binary Diff*. Available: http://www.cjmweb.net/vbindiff/ (accessed 22.4.2013). Latest version was released in 2008
Mathworks, 2012	Mathworks. *Simulink – Simulation and Model-Based Design*. Available: http://www.mathworks.se/products/datasheets/pdf/simulink.pdf (accessed 24.4.2013). Mathworks, 2012
Maxfield, 2004	Maxfield, C., *The Design Warrior's Guide to FPGAs.* ISBN 0-7506-7604-3. Elsevier, 2004
Mentor Graphics, 2008	Mentor Graphics Corporation, *FormalPro – Datasheet*. Mentor Graphics, 2008
Mentor Graphics, 2010	Mentor Graphics Corporation, *ModelSim 6.6d – User's manual*. Mentor Graphics, 2010
Microsemi, 2012a	Microsemi Corporation. *IGLOO Low Power Flash FPGAs Datasheet*. Revision 21. Microsemi, 2012
Microsemi, 2012b	Microsemi Corporation. *In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X*. Microsemi, 2012
Microsemi, 2012c	Microsemi Corporation. *ARM Cortex-M1-Enabled IGLOO Development Kit User's Guide*. Microsemi, 2012

Microsoft, 2013	Microsoft Corporation. Web site for Microsoft Excel. Available: http://office.microsoft.com/en-us/excel (accessed 24.4.2013)
NRC, 2009a	United States Nuclear Regulatory Commission. NUREG/CR-6992: *Instrumentation and Controls in Nuclear Power Plants: An Emerging Technologies Update.* NRC, 2009
NRC, 2009b	United States Nuclear Regulatory Commission. NUREG/CR-6991: *Design Practices for Communications and Workstations in Highly Integrated Control Rooms*. NRC, 2009
NRC, 2010a	United States Nuclear Regulatory Commission. NUREG/CR-7006: *Review Guidelines for Field-ProgrammableGate Arrays in NuclearPower Plant Safety Systems*. 94 p. NRC, 2010
NRC, 2010b	United States Nuclear Regulatory Commission. NUREG/CR-7007: *Diversity Strategies for Nuclear Power Plant Instrumentation and Control Systems*. 225 p. NRC, 2010
Poiksalo, 2007	Poiksalo, P-K. *Digitaalitekniikan Perusteet*. ISBN 978-952-5511-02-4. Modeemi ry, Tampere, 2007
Ranta, 2012	Ranta, J., 2012, *The current state of FPGA technology in the nuclear domain*. VTT Technology Issue 10, 62 p., VTT, Espoo, 2012
Ranta, 2013	Ranta, J. *Multi-Core Processing from NPP I&C Perspective*. VTT Research Report VTT-R-00177-13. 12p., VTT, Espoo, 2013
SAFIR2014, 2013	Web site of the Finnish Research Programme on Nuclear Power Plant Safety 2011-2014. Available: http://virtual.vtt.fi/virtual/safir2014/ (accessed 23.4.2013)
SAREMAN, 2012	SAREMAN, *Easy Approach to Requirements Syntax (EARS) brochure*. Available: http://cse.aalto.fi/wp-content/uploads/2012/01/EARSmaterial-2xA4-Final.pdf (accessed 19.4.2013)
Simpson, 2010	Simpson, P. *FPGA Design – Best Practices for Team-based Design.* ISBN 978-1-4419-6338-3. Springer, New York, USA, 2010
Smith, 2010	Smith, G. *FPGAs 101*. ISBN 978-1-85617-706-1. Elsevier, 2010

STUK, 2009	Säteilyturvakeskus STUK, Health and Safety Executive HSE, Autorité De Sûreté Nucléaire ASN. *Joint Regulatory Position Statement of the EPR Pressurised Water Reactor.* 22.10.2009
Tommila, 2012	Tommila, T., Valkonen, J., *"A control engineer's first guide to requirements engineering"*. Working report (draft 31.1.2013). VTT, 2013
Waage, 2012	Waage, H. FPGA Technology Used at the Temelin Nuclear Power Plant. *The proceedings of the NPIC-HMIT 2012*. pp. 821-828, San Diego, California, USA, July 22-26, 2012
Valkonen *et al.*, 2008	Valkonen, J., Karanta, I., Koskimies, M., Heljanko, K., Niemelä, I., Sheridan, D., Bloomfield, R. E. *NPP Safety Automation Systems Analysis – State of the Art*. VTT Working Papers 94 (VTT-WORK-94). ISBN 978-951-38-7158-1. 62 p., VTT, 2008
Wikipedia, 2013	Wikipedia. *Integrated Circuit Design*. Available: http://en.wikipedia.org/wiki/Integrated_circuit_design (accessed 22.4.2013)
Yarom *et al.*, 2006	Yarom, I., Zuckerman, M., Seligman, E., Sokolover, A. Cadence Conformal LEC – The Intel Experience. *CDNLive! EMEA*. 25-27 June, 2006