NTNU

Norwegian University of
Science and Technology

HW/SW Codesign of a Pedestrian Detection System

Magnus Rammel

Master of Science in Electronics
Submission date: January 2018
Supervisor: Per Gunnar Kjeldsberg, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Abstract

The HW/SW co-design philosophy introduced almost two decades ago provides solutions to common challenges in the electronic system design industry. It provides techniques for improving the efficiency of conventional design processes for electronic systems, by introducing methods for simultaneous design of hardware and software, as well as techniques for analyzing and finding correct implementation alternatives for selected system functions. This thesis describes the selection of a HW/SW co-design methodology for a software-based Pedestrian Detection System, as well as results from applying the methodology on the system itself.

In this design methodology, the system functions are evaluated from pre-defined design constraints for performance, energy consumption, excessive hardware and software parts, and system reliability. The functions not meeting the constraints are further selected for partitioning into either hardware or software architectures that result in potentially new implementation suggestions for the functions selected. The evaluation has been performed with evaluation criteria in the form of a cost function based on the constrained factors, and the results from applying the methodology on the system have been described. The data used for the evaluation criteria were counts of instructions executed, data reads/writes, as well as cache hit/miss counts of these. These counts are run time attributes of the Pedestrian Detection System's system functions, and are meant to indicate run time and performance demand on the system. The result of the evaluation indicated that no new implementation alternatives were needed.

Sammendrag

HW/SW co-design-filosofien som ble introdusert for omtrent to tiår siden tilbyr løsninger på vanlige utfordringer i industrien for elektronisk systemdesign. Den tilbyr teknikker som kan forbedre effektiviteten til konvensjonelle designprosesser for elektroniske systemer, ved å introdusere metoder for å designe maskinvare og programvare simultant, i tillegg til teknikker for å analysere og oppdage korrekte implementeringsalternativer for utvalgte systemfunksjoner. Denne oppgaven beskriver utvelgelsen av en HW/SW co-design-metodikk for et programvare-basert Fotgjengerdeteksjonssystem, i tillegg til resultater fra å bruke designmetodikken på systemet selv.

I denne metodikken blir systemfunksjonene evaluert ut fra forhåndsdefinerte designrestriksjoner for ytelse, energiforbruk, overflødig maskinvare og programvare, og systempålitelighet. Funksjonene med egenskaper som overgår restriksjonene blir videre valgt ut for å partisjoneres til enten maskinvare- eller programvarearkitekturer som resulterer i potensielt nye implementeringsforslag for de utvalgte funksjonene. Evalueringen har blitt utført med en kostnadsfunksjon utviklet fra de restrikterte faktorene, og resultatene fra bruken av metodikken på systemet har blitt beskrevet. Dataen brukt til evalueringskriteriene er tallverdier på antall instruksjoner utført, antall lesinger og skrivinger av data, i tillegg til tall på cache treff/ikke-treff for disse. Disse tallene er kjøretidsegenskaper for Fotgjengerdeteksjonssystemets systemfunksjoner, og er ment å indikere kjøretid og ytelsesbelastning på systemet. Resultatet fa evalueringen indikerte at ingen nye implementasjonsalternativer behøvdes.

Preface

This Master thesis project has taught me a great deal on the topic of HW/SW co-design. It has delivered a great learning outcome in terms of practical experience with previously acquired theoretical knowledge from both courses and the Specialization project with the same topic. The HW/SW co-design topic was chosen partly due to its relevance to my study programme specialization, but also because of the aforementioned practical learning outcome I assumed it would give me. The project has given me a deeper insight into the topic of HW/SW co-design, as I expected it would do.

During the thesis work I have gotten a new insight into performing computer program profiling in the Linux operating system. I have also learned a lot about the history of HW/SW co-design and thus how it has influenced electronic system design since its introduction.

I would like to thank my supervisor Per Gunnar Kjeldsberg for continually supporting and providing me with council on which steps to take forward, as well as for giving me feedback on the report at several instances during the project phase.

Magnus Rammel

Trondheim
17. January 2018

Problem Description

Candidate Name:
Magnus Rammel

Assignment Title:
HW/SW Co-design of a Pedestrian Detection System

Assignment Text:
Advanced Driver Assistance Systems (ADAS) have become an integral part of modern high-end cars. With data from cameras and other sensors, they perform services such as pedestrian detection, blind spot detection, and lane departure warning. The computational requirements for many of the algorithms needed to process the data are substantial. To have price, size, compute and energy efficient solutions, a combination of hardware and software is typically required. NTNU is partner in the EU research project Tulipp (Towards Ubiquitous Low-power Image Processing Platforms), where a system for pedestrian detection is one of the use-cases. A software implementation of the system is available in an open source repository.

In this master thesis assignment, the Pedestrian Detection System shall be investigated from a hardware/software co-design perspective. Based on an extensive literature study and early investigation of the system code, a design methodology shall be selected and described. Following the selected methodology, critical parts of the system shall be implemented in hardware using design tools suitable for Xilinx FPGAs. Overall improvement in performance and energy efficiency compared to an all software solution shall be estimated. To the extent time allows, test on an FPGA evaluation board shall be performed.

Supervisor:
Per Gunnar Kjeldsberg.

Contents

List of Abbreviations

FPGA Field-Programmable Gate Array

HW Hardware

IoT Internet of Things

LHT Loop Hierarchy Tree

PSO Particle Swarm Optimization

SW Software

UI User Interface

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Electronic devices of different shapes and sizes are a vital part of today's society. With ever increasing demands from the market, that require both increased performance as well as more advanced functionality from the products, the need to increase the efficiency in the design process for these devices has become increasingly important for developers and producers. By increasing the efficiency in the design process one can improve the products, thus enabling improvements in areas such as medical equipment for health concerns, automotive technology for the sake of traffic safety, and generally also other areas where electronic devices are used. The efficiency increase is also important to enable manufacturers to keep their position in what has become a tough and highly competitive market. To accommodate the market demands, several techniques for increasing the efficiency have been developed, and one particularly well regarded set of techniques are some of those derived from the Hardware/Software (HW/SW) co-design philosophy. HW/SW co-design is a design philosophy that focuses on improving the efficiency of traditional design processes and also the co-operation between the hardware and software parts of electronic systems.

The traditional design process for developing electronic systems has usually been considered sequential and time consuming, with few design processes being conducted in parallel. This has resulted in increasing cost and length of design schedules. The traditional design processes are thus not considered suitable for the modern market demands in the electronics industry. Additionally, the complexity of modern systems tends to grow with each new generation of technology, and it also seems like each new generation replaces the old one more quickly than before [20]. With all these new challenges, a new approach to the design philosophy behind electronic system development is considered necessary.

A HW/SW co-design process utilizes the synergy between hardware and software to improve the co-operation between the two. With the invention of the Field Programmable Gate Array (FPGA), the available target platforms on which the system functions can be mapped increased dramatically [25]. In an attempt to improve central factors in the design process such as the efficiency, cost, and length of design schedules, the HW/SW co-design philosophy has been developed. The aim with this philosophy is to decrease the length of design schedules, the design cost, and also the chip size, among other

factors also involved in the design process for electronic systems. The goal with this design philosophy is to enable designers to design systems in a shorter amount of time, which in turn is believed to result in a shorter time-to-market for their systems.

1.1 Goals of the thesis

The goals of this master thesis were to evaluate and analyze the Pedestrian Detection System from a HW/SW co-design perspective. The thesis is a continuation from a previous specialization project with the same topic. The results from the specialization project concluded that no new implementations were needed, and the goal with this thesis is thus to conclude on whether further studies into the system will yield a different result.

1.2 Overview of report Chapters

This section will provide an overview of the rest of the report Chapters.

- **Chapter 2:** Theory and Background is the first major Chapter in this report. This Chapter will provide the reader with a detailed overview of various aspects of HW/SW co-design. It will go into the major factors involved when one attempts to use the HW/SW co-design philosophy to design a system, and also the challenges one is usually faced with when making design choices, in the form of necessary trade-offs. This section will be used as basis for the theoretical groundwork behind the development of the design methodology.
- **Chapter 3:** Previous Work on HW/SW Co-design Methodologies contains information of notable previous work and contributions performed on the topic of HW/SW co-design, that can be considered relevant to the Pedestrian Detection System. It focuses on the work and contributions most relevant to the selected methodology described in this report, thus giving a brief historic overview of the previous development on the topic.
- **Chapter 4:** Pedestrian Detection System will describe the system that the design methodology is developed for. The Chapter will give a general overview of how the system is built up, what it was developed for, and also generally how it works.
- **Chapter 5:** Evaluation Tools will describe how the HW/SW co-design investigation was carried out, in the form of the methods or tools that were used. It will go into detail on each tool and thus provide some general information about them.
- **Chapter 6:** Suggested Design Methodology is where the selected design methodology will be described and reasoned for. Parallels to the Theory and background Chapter will be drawn, and thus the various factors that will be considered in the methodology will be described in terms of how they will affect the implementation choices.
- **Chapter 7:** Results of Design Methodology is the Chapter that will describe the results of the HW/SW co-design investigation into the Pedestrian Detection

System and thus reveal the implementation choices. It will also go into detail on how they were made and the reasons behind the choices.

- **Chapter 9:** Discussion will discuss any uncertainties involved in the implementation choices, and will further discuss possible uncertainties or issues with the measurements and how this has affected or not affected the end result.
- **Chapter 10:** Conclusion is the final content section. It sums up the project results and elaborates and concludes on how the results match the assignment description. It will also go into what can possibly be considered for future work.
- References features all the references used in this project report.
- Appendix provides the figures and tables that were deemed too big to be placed in the main part of the report.

Chapter 2

Theory and Background

2.1 Background

Ever since the creation of the first known electronic components, believed to be the creation of the first diode by Fleming in the year of 1904 [11,12], electronic devices have gradually filled more vital roles in various areas of society. Notable examples of these areas are information broadcasting, signal- and data processing, process control, as well as safety systems and medical applications. Since the 1960s and onwards, embedded systems in particular have undergone extensive developments in design and functionality, and have gradually seen more use in several areas of society [17].

According to W.Wolf in [23], the term Hardware/Software co-design was introduced in the beginning of the 1990s. The term was developed for the purpose of creating a common term for what he refers to as "a confluence of problems in integrated circuit (IC) design" [Wolf, 2003, p. 38]. As further stated by Wolf in [23], the HW/SW co-design field began to take shape when designers experimented with combining microprocessors and standard hardware components on circuit boards.

As quoted from [4]: "The Minuteman intercontinental ballistic missile was the first to use an embedded computing system, and was built in 1961 for the US Air Force. This embedded system was the Autonetics D-17 guidance computer". The Autonetics D-17 guidance computer is illustrated in Figure 2.1. As stated in [17], this computer was the first ever known embedded system to have been produced in large quantities. Following a series of additional usage of embedded systems for military- and aviation purposes, different industries began to utilize embedded systems for other purposes from the beginning of the 1990s.

As mentioned above, it was around this time that the HW/SW co-design term was introduced. As of the beginning of the 21st century, embedded systems as well as several other kinds of electronic systems have become increasingly involved in the lives of human beings from day to day. Notably with the introduction of a wide variety of electronic devices united under the common term Internet of Things (IoT), electronic systems, and particularly mobile and embedded systems, are increasingly taking part in areas of everyday life [24].

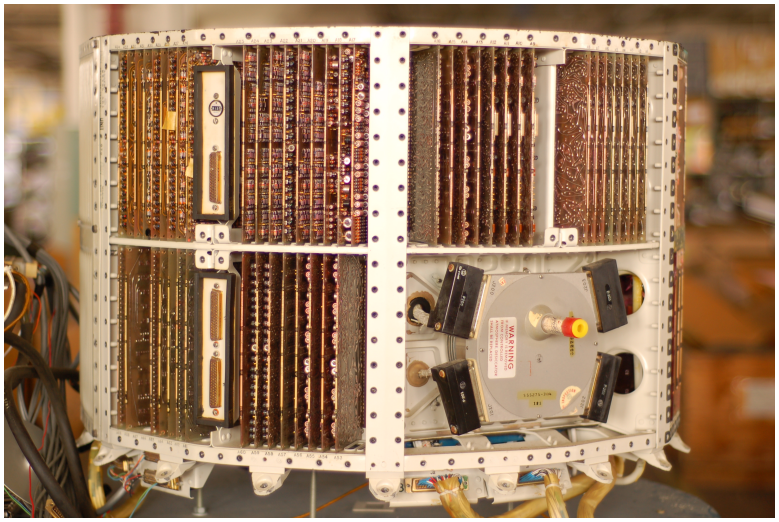

Figure 2.1: Photograph of the Autonetics D-17 guidance computer, mounted on a Minuteman intercontinental ballistic missile. It is believed to be the first ever embedded system produced in large quantities. [3]

As previously described, HW/SW co-design is a term that was developed in the electronic system design industry to describe a set of challenges in IC design. It is also described by Ha & Teich in [13] as a technology field, and as quoted, "strives to achieve system-level design objectives by leveraging the synergy between hardware and software through their concurrent design" [Ha & Teich, 2017, p. 3].

The term can also be thought of as a design philosophy, seeing as it is often used to develop new and improved design methodologies for electronic systems. Aside from focusing on exploiting the synergy of hardware and software parts in electronic systems, HW/SW co-design methodologies are characterized by attempts to design for the best trade-offs based on various design attributes characteristic of hardware and software [13]. The underlying intention behind a HW/SW co-design process is to improve the efficiency in various areas of traditional design processes to reduce design time, increase product quality, and lower production cost. A generalized design process for HW/SW co-design is given in Figure 2.2.

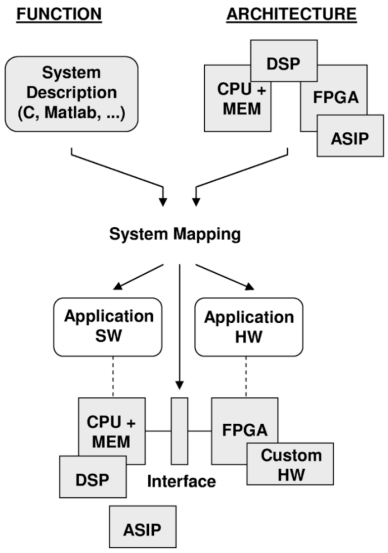

Figure 2.2: Flow chart for generalized HW/SW co-design process.

Since this work was performed, HW/SW co-design has evolved into a well established field [19]. Based on several aforementioned key factors, or design principles, the field has gradually seen more use in the industry, notably in the form of Field-Programmable Gate Arrays (FPGA) as well as in several other research projects conducted in the field [23]. Over the years, the HW/SW co-design term has evolved and is now characterized as a mainstream technology field in the area of embedded systems, especially with the introduction and development of the FPGA boards.

2.2 Traditional design processes for electronic systems

A simple model for the traditional design process for electronic systems is illustrated in Figure 2.3. As illustrated, a traditional design process begins with identifying the specifications and requirements of the system, before selecting the appropriate algorithm according to the system's functional specification. The design process then proceeds with the decision on the hardware architecture, based on the functional requirements, as well as a manual partitioning process for HW and SW. The steps that however separate the traditional design process from the HW/SW co-design approach is the development phases of the hardware and software parts. As illustrated, these steps are carried out separately, and the development is according to [13] often performed with two engineering teams that work on either developing the hardware or the software by themselves.

As illustrated in 2.3, the decisions for type and design of hardware architecture is the first step in the implementation phase. This results in the hardware being fixed before the system software is developed and tested, which removes the opportunities for

co-simulation of hardware and software to improve the synergy between them. Furthermore, this decision also results in increased design time schedules and economical cost for the designers, in the event that the design teams encounter implementation challenges that necessitate a re-design of the hardware or software to enable a better system integration for either the hardware or software.

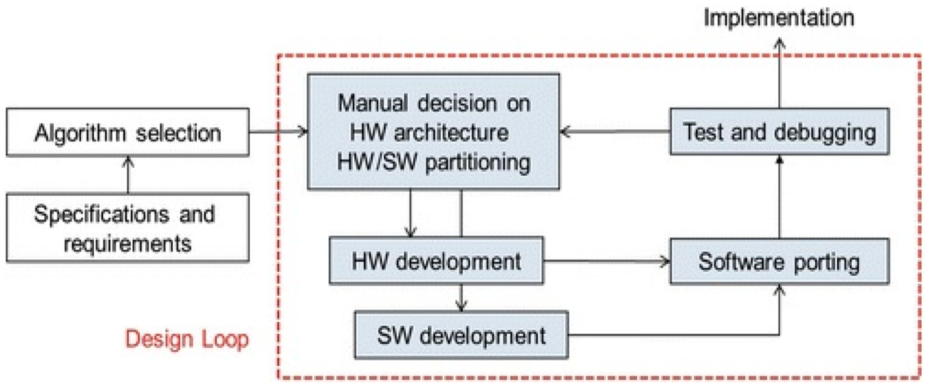

Figure 2.3: Model or flow chart for a traditional design process for electronic systems. [13]

2.3 The HW/SW Co-design approach

The aforementioned encounter of implementation challenges serves as a motivation for employing HW/SW co-design techniques. A typical design methodology for HW/SW co-design is illustrated in Figure 2.4. As illustrated, a typical HW/SW co-design process begins with a system description or rather a description of the system tasks or functions for the system to be designed, and also a selection of hardware and software architecture platforms on which the system functions are to be implemented. This phase in the design process is followed by the system mapping or partitioning phase, where the functions of the system are mapped onto the various hardware and software architectures, creating new hardware and software implementations for the system. The third and final phase is the system is the

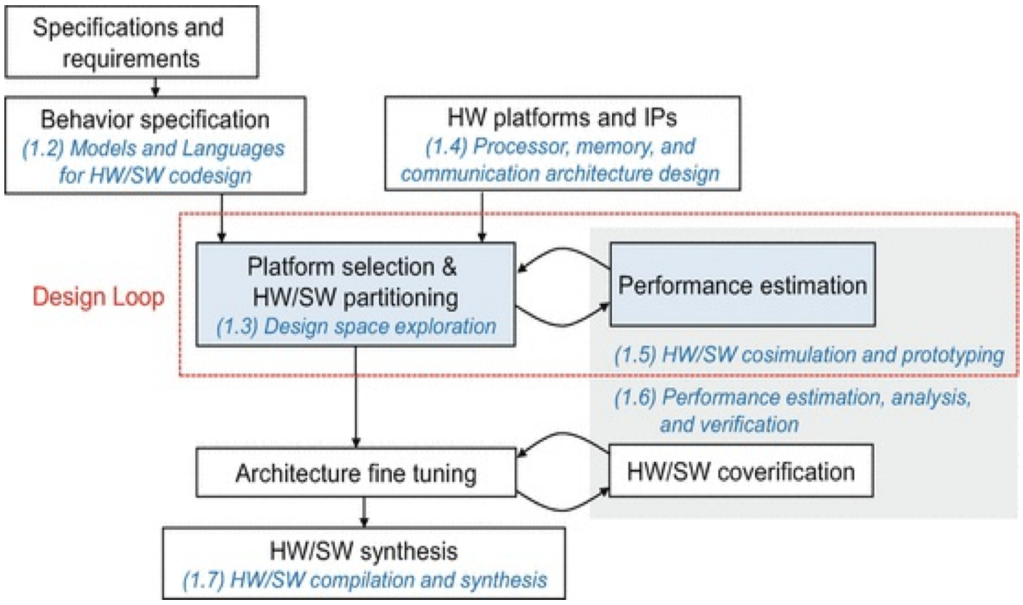

Figure 2.4: Model or flow chart for a HW/SW co-design process for electronic systems. [13]

2.4 Design process for HW/SW Co-design

The goal with a HW/SW co-design process is to co-ordinate the design of hardware and software, ultimately seeking to avoid the design challenges usually found with traditional design processes in the industry [14].

The fundamental goals of HW/SW co-design can be narrowed down to the following topics:

- Shortening design schedules
- Shortening time-to-market
- Reducing cost of design
- Improving product quality

2.5 HW/SW Partitioning

As mentioned in the previous Section, the different design processes performed with the HW/SW co-design philosophy all involve some kind of system mapping after the system functions and architectural components have been established. This process is about mapping the objects and components together, with the resulting implementations being in either hardware or in software, or in the form of an interface between the two. This mapping process is called HW/SW partitioning. The purpose with this step is to decide whether to make a hardware implementation or a software implementation for all the system functions to be implemented. [20].

According to [5], the ideal partition tool for hardware and software automatically produces a set of high-quality partitions within a short and predictable time frame for computation. The process rests on a solid foundation of practical and theoretical insight into the various alternatives of hardware and software components available for implementation [5]. Generally, this decision making process considers all the attributes of the different system functions, and concludes on an implementation choice based on the best fit of the available architecture components, as well as constraints from key HW/SW co-design factors such as design complexity, performance, energy efficiency, and reliability [18, 20]. From these factors, the choice of whether to implement in hardware or in software is made from different attributes of these factors [20]. The attributes and constraints that favor hardware or software implementations are all dependent on the system to be designed, but can to some extent be divided into general categories. These categories are listed below.

Attribute	**Implementation alternative**
Suitability/aptitude for parallel implementation	Hardware
Large amount of excessive software	Hardware
Small amount of calculations	Hardware
High speed requirement	Hardware
High desire for energy efficiency	Hardware
Suitability/aptitude for sequential implementation	Software
High level of complexity	Software
Large amount of calculations	Software
Low speed requirement	Software
Energy efficiency less important	Software

Table 2.1: Overview of the optimal implementation alternatives for the various program statements being part of the Pedestrian Detection Systems execution.

As stated by M. B. Abdelhalim et al. (2011) in [5], programmable software parts of embedded systems are often less costly and time consuming to develop and improve

than the application specific hardware parts of the system.

Following from this, the various design factors will be elaborated on in greater detail in the following Subsections below.

2.5.1 Parallel or Sequential solution

A fundamental decision involved in the system design is whether the functionality has an aptitude for being carried out in a parallel or a sequential manner. Implementation wise, a hardware implementation is well suited for parallel execution, while a software implementation is more suited for sequential execution of tasks [20]. For parallel implementations, hardware implementations can feature redundant components and circuits, and can thus easily be designed in a way that allows for parallel execution of tasks [20]. The trade-off in this regard is the added space requirement for the system. Running tasks in parallel in software is more complicated, and often requires dedicated processes for scheduling the system tasks in what is called a concurrent execution [21]. Issues such as interference, race conditions, deadlocks and livelocks are all fairly common in concurrent execution of software, in addition to the added design time necessary to implement concurrent execution of tasks [8]. It is therefore simpler to run a software implementation in a serial manner on a single CPU core. Running the implementation on a single core however leads to more concentrated accumulation of heat, which will require more cooling.

2.5.2 Level of complexity

Design complexity is another important factor. Producing hardware is expensive, and the more complicated a hardware implementation becomes, the more space it usually requires from the system. The decision making in terms of design complexity therefore involves much of the same considerations as when deciding from an abstraction level point of view. Solving complex system tasks is usually far easier to do with lines of code, and only takes up virtual space in registers and memory instead of demanding big physical footprints.

Level of abstraction is another important and fundamental factor in HW/SW co-design. All tasks meant to be implemented have a level of abstraction to them that normally results in a software implementation being more favorable the higher the abstraction level becomes. Implementing a task with a high abstraction level in hardware, such as for instance a complicated mathematical calculation, might result in the hardware becoming complicated and requiring a larger footprint, thus increasing design cost and space requirements for the system. Choosing a software implementation for a task featuring complicated calculations is far less complicated, as tasks can be implemented simply with the use of lines of code instead of big and complicated physical circuits.

Software libraries also often contain extensive amounts of functionality options, and a programmer also usually has several software languages to choose from that suits different abstraction levels of programming. In general, one could say that unless the

task is about transmitting signals or performing very simple calculations, a software implementation is the most optimal choice, if not the only choice. As further support to this claim, some typical operations done in software such as making lists or arrays, and performing manipulations in the form of adding and deleting elements, is more or less impossible to accomplish with a HW implementation alone.

(KOMMENTAR: Her og andre steder mangler det en del referanser, da det er et resultat av at jeg prøvde å bare få noen ord på papiret. Kan være noe av det er feil også.)

2.5.3 Requirements for performance/execution time

As mentioned before, the speed requirements or time constraints demanded from the task is, when considered on its own, a requirement that almost always leads to a HW implementation, due to the aforementioned nature of hardware. Proof of this statement is given in Figure 2.5.

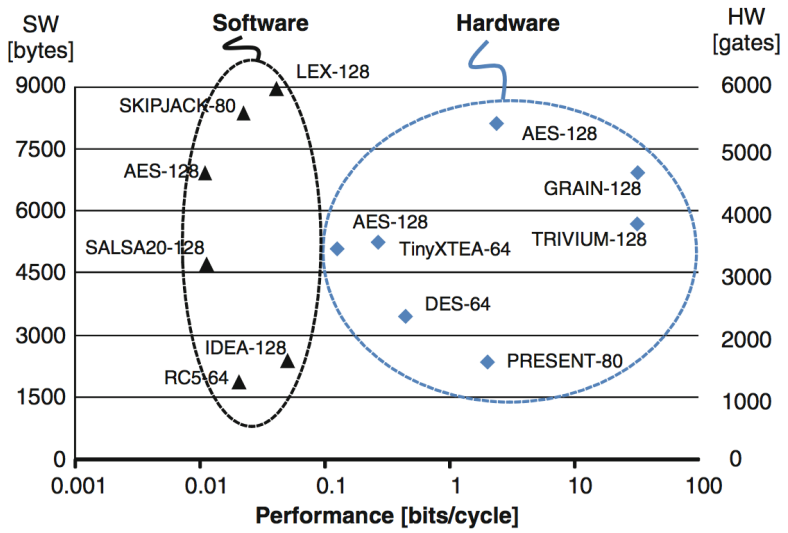

Figure 2.5: Various cryptographic implementations in software and hardware proposed from the year 2003 to 2008. [20]

2.5.4 Energy efficiency

Energy efficiency is one of the biggest and most important factors of HW/SW co-design. When designing a system to improve this factor, one can decrease heat dissipation or heat accumulation in the whole system, making the cooling demands lower. One can also improve the potential system performance because of a higher performance threshold due to the lower heat accumulation, and also improve battery life for mobile applications due to the smaller power consumption. In addition, one is also able to use smaller batteries that have a smaller footprint, thus taking up less space and leaving room for more parts on the circuit boards or similar. In terms of implementation choice, a HW implementation is generally favored due to the spreading of heat dissipation over several

locations rather than one concentrated location. By implementing in SW, all the work is usually performed by a single processor regardless of the amount of cores it has. This concentrates the heat accumulation and thus increases cooling demands. Despite this fact, implementing smaller and less CPU demanding tasks in SW might consume less power than a corresponding HW implementation due to the HW often having a standby current. This is in the context of situations where only very low amounts of power are tolerated.

2.5.5 Required reliability in design

Reliability concerns are a more separate topic from the others. According to [16], reliability is defined as "The probability that a required item will perform a required function without failure under stated conditions for a stated period of time." [16].
By this definition, one could state that a reliable system seldom fails. Basing an implementation choice on reliability is likely to result in a difficult decision making process with many factors to consider. In terms of potential for creating redundancy in the system, both a HW implementation and a SW implementation are viable options. Creating redundancy in hardware will often simply be solved by creating more parallelism in the hardware, meaning using more physical space by using more components than strictly necessary, which ensures the likelihood of continued task accomplishment even if certain components cease to function. Creating redundancy in software might be more complicated, but will probably demand less physical space in return. To make redundancy in a SW implementation one could design the software to feature backup recovery, meaning the executing program has a backup program that monitors the main program and takes over as the main program in the event of failure of the main program, and also in the process spawns a new backup program. Another way to design for redundancy in SW could be to use multiple registers, memory, CPUs, etc, and in the process also design the software to use all the resources indiscriminately. In the end, the reliability of the system is likely to depend on what is most cost efficient and best suits the physical space requirements for the system.

2.6 Cost function

As mentioned before, designing and applying a well thought out cost function to guide the design [15] is important in a HW/SW co-design methodology. The cost function is created as a mathematical equation based on functions presented in the work of [5] and [6], and contains the various design factors that are by the designer considered important in making the design as efficient as possible. The design factors are represented as addends in a large equation, where each design factor is constrained by a threshold value, and represented as a percentage of cost for the given implementation. For an optimally designed cost function, the computation result of the function provides a calculated cost that is close to the actual cost for the chosen system function and architecture component in question.

2.7 Co-simulation of HW and SW

A major technique used for shortening the design schedules and reducing cost is using co-simulation of HW and SW from the beginning of the design process. This is achievable through the use of modern hardware simulators, which have become accurate enough to replicate physical hardware testing to a large extent. Major benefits with this approach is the ability to handle design errors and other hardware related problems before the actual hardware is produced [9].

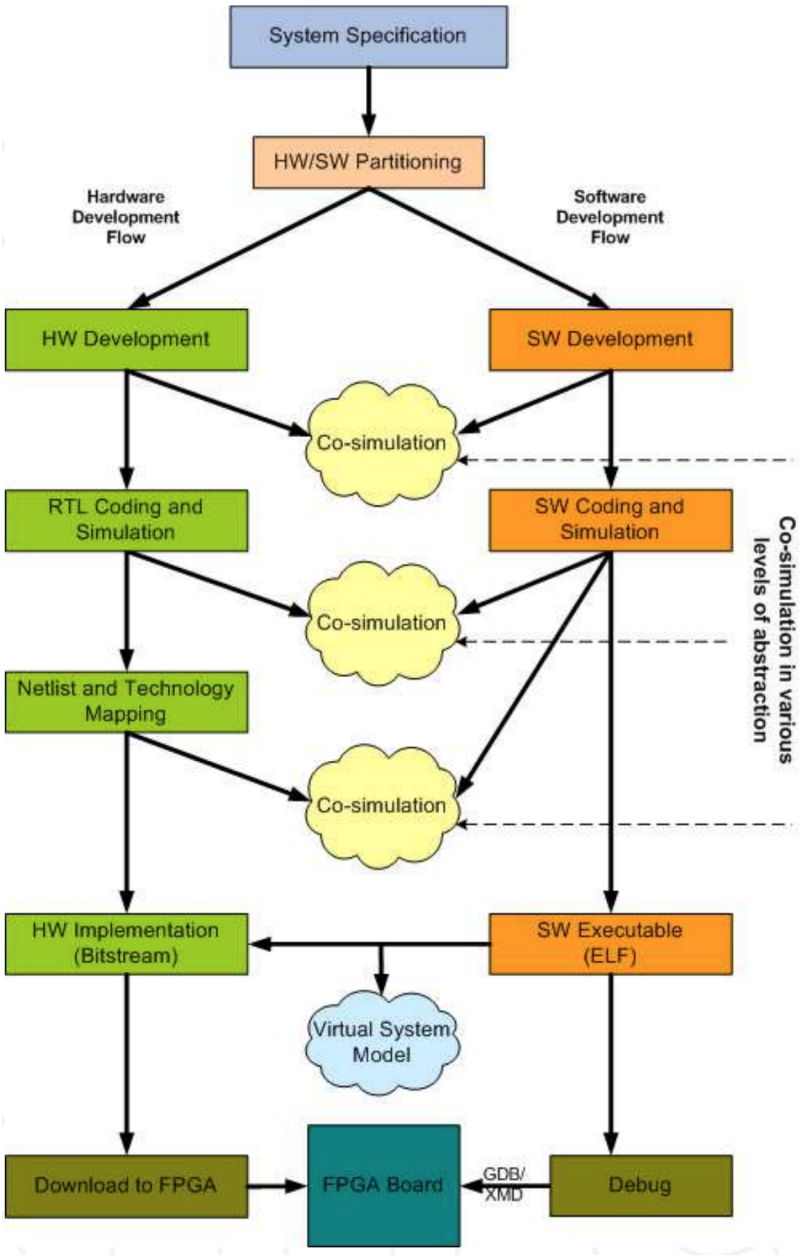

Figure 2.6: Illustration of an applied HW/SW co-design methodology for a SoC implementation [10]

2.8 FPGA

With the development of the Field-Programmable Gate Array, or FPGA for short, a new and simpler way of designing systems in HW/SW co-design arrived. [23] The FPGA platform, while differing in characteristics and design solutions depending on manufacturer, is by some viewed as the chip for which cosynthesis was created [23]. The architecture of the chip is built to be programmable, meaning it is well suited for HW/SW partitioning and thus for HW/SW co-design.

The FPGA circuit boards are capable of being programmed and re-programmed after production through the use of programmable on-board hardware resources [22]. The FPGAs feature many similarities to fixed specialized hardware, with features such as "application-tailored parallelism", "low power-consumption" and "integration advantages" [Teubner & Woods, 2013, p. 5]. Despite the fact that they also exhibit characteristics associated with simulated hardware such as increased power consumption and reduced clock speed, it is usually concluded that the benefits of an FPGA implementation outweigh the drawbacks in most cases.

Chapter 3

Previous Work on HW/SW Co-design Methodologies

This chapter will describe various HW/SW co-design methodologies presented in various publications the last three decades since the term was first introduced. The aim of this chapter is to illustrate the variety of HW/SW co-design methodologies that have been developed to illustrate the various ways in which the HW/SW co-design techniques can be employed. Notably the methodology described in [5] by M. B. Abdelhalim et al. (2011) has been used as inspirational material for the selection of the appropriate design methodology described in Chapter 6.

3.1 A Hardware-Software Codesign Strategy for Loop Intensive Applications

The generation of a Loop Hierarchy Tree LHT) was performed by Zhang & Kandemir in 2009 [25] in their work on developing a HW/SW co-design methodology for loop intensive applications. The authors believe they were the first to utilize a LHT in a HW/SW co-design methodology. The LHT is generated from an intermediate code representation, for the purpose of carrying out a co-design exploration through what is called a branch-and-bound search. The end goal for Zhang & Kandemir was to create a HW/SW partitioning that would reduce the delay in execution for an application to a minimum, while satisfying pre-defined constraints on hardware area.

This particular work's focus on targeting and improving the efficiency of applications that are loop and data intensive is one of the reasons it was included in this chapter. The mapping tree bears a close resemblance to the call graph representation generated by KCachegrind in the thesis project for the Pedestrian Detection System. Further, the mapping tree is generated from a breadth-first search through the LHT, which in a way resembles the way in which the call graph view in KCachegrind is generated from the Callgrind output file from the PDS.

3.2 An integrated high-level hardware/software partitioning methodology

A publication presented by M. B. Abdelhalim et al. in the year 2011 [5] proposed a HW/SW co-design methodology developed with the use of the Particle Swarm Optimization (PSO) technique for partitioning the hardware and software parts. The methodology also featured modeling of the hardware with two so-called "extreme implementations" (M. B. Abdelhalim et al., 2011, p. 19) using different hardware scheduling alternatives. The system functions are determined with the PSO technique. The target architecture is FPGAs.

The PSO technique utilized in this methodology was based on a previous work published in 2006 [7], which is believed by the authors to be the first ever HW/SW co-design work to employ the PSO technique.

The partitioning phase for this methodology was

1. Application domain: whether the partitioning algorithm is "multi-domain" (conceived for more than one or any application domain, thus not considering particularities of these domains and being technology-independent) or is "specific domain" algorithm.
2. The target architecture type.
3. Consideration of the HW/SW communication costs.
4. Possibility of choosing the best implementation alternative of HW nodes.
5. Possibility of sharing HW resources among two or more nodes.
6. Exploitation of HW-SW parallelism.
7. Single-mode or multi-mode systems.

Chapter 4

Pedestrian Detection System

As referred to in previous chapters, the objective of this master thesis project has been to develop a HW/SW co-design methodology for a Pedestrian Detection System. Since changing or altering the application was not a part of this thesis project, the whole system remains unchanged as the result of previous work performed by other or former project participants. The system used for the basis of this project's results, consists of a single code repository that is meant to be compiled and run on the Linux computer operating system. Further details about the system are outlined in the rest of this section.

As mentioned above, the system is created from a single open sorce code repository. As stated by the creator, the repository contains code that has been extracted from a much larger code base. The system therefore contains code that has been developed specifically for research purposes, meaning that it has been modified from the original release to be able to function separately without the rest of the original code base. The system has been in development for more than four years, with the focus being placed on "easy exploration of multiple variants" and "computational efficiency". Improving the compactness of the code has therefore been a secondary focus, thus one can expect to find some areas with potential for improvement at several places in the code.

The system was made to be compiled on two hierarchical levels, i.e. the first level had to be compiled first to enable one to compile on the second level. The first compilation level only makes use of what is referred to as "CPU only code", which is code that uses the computer's CPU alone to run the test applications. The second level uses what is called "test time code" i.e. it contains code that makes use of the CUDA platform for parallel computing developed by the Nvidia corporation, in addition to the CPU code from the first level.

The results presented in this report are all derived from the "CPU only code". There are two reasons for this, with the first being that the student decided that using code made for a parallel computing platform to do a HW/SW co-design investigation would make identifying the order of execution for the different processes harder.

The system compiled with CPU only code contains several test applications that il-

lustrate the functionality of the Pedestrian Detection System in different ways. These test applications are the ones used for generating the results presented in this report, and are in turn also the platforms that the HW/SW co-design investigation was carried out on, based on the proposed HW/SW co-design methodology. Figure 4.7 and Figure 4.8 display excerpts of screenshots from running the applications ground_estimation and stixel_world respectively.

4.1 Code structure

The Pedestrian Detection System is a software-only system, with multiple , and thus contains a substantial amount of program files. The amount of program files inherent in the PDS is rougly estimated to be slightly less than 3000, and the programming languages used in the files are mostly written in C++, C and Python, complemented with header files. Some other program file types exist as well, notably CMake files used for building, packaging and testing software [1], and also standard Makefiles and other files used for compilation purposes.

The profiling was collected from various runs of the different modes of the PDS, namely the ground_estimation and stixel_world modes. The application runs with Callgrind and cache simulation enabled produced various kinds of data that which was presented in KCachegrind. The Call Graph feature of KCachegrind presented the tree structure of the various kinds of functions and statements that were part of the application run.

FOllowing are illustrations of various features of the Pedestrian Detection System:

Figure 4.1: High level flow of Pedestrian Detection Algorithm

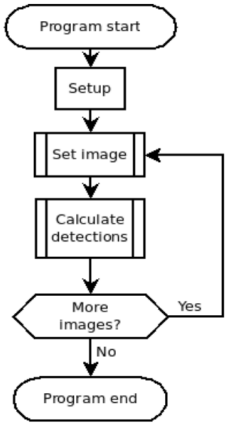

Figure 4.2: Flow chart for the main program loop. It passes the images selected for detection to the detector and initializes the detection.

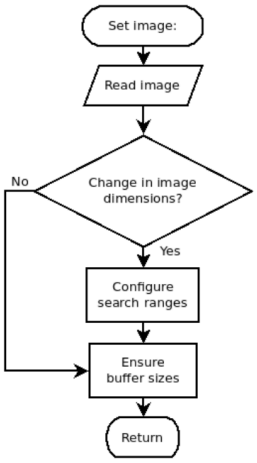

Figure 4.3: Flow chart for the Set Image subtask. Reads the input image and computes the scaling of the detectors.

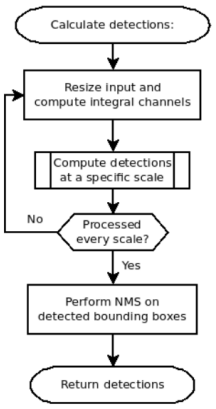

Figure 4.4: Illustration showcasing the most relevant parts of the PDS.

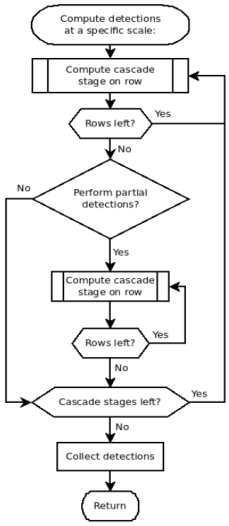

Figure 4.5: TRENGER INFO HER, sjekk kode eller bitbucket-repo.

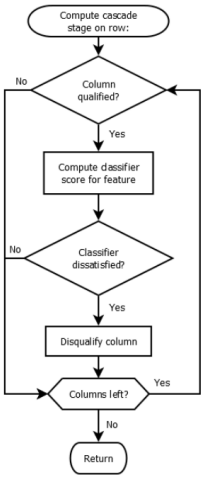

Figure 4.6: Displays general flow employed by the classifier.

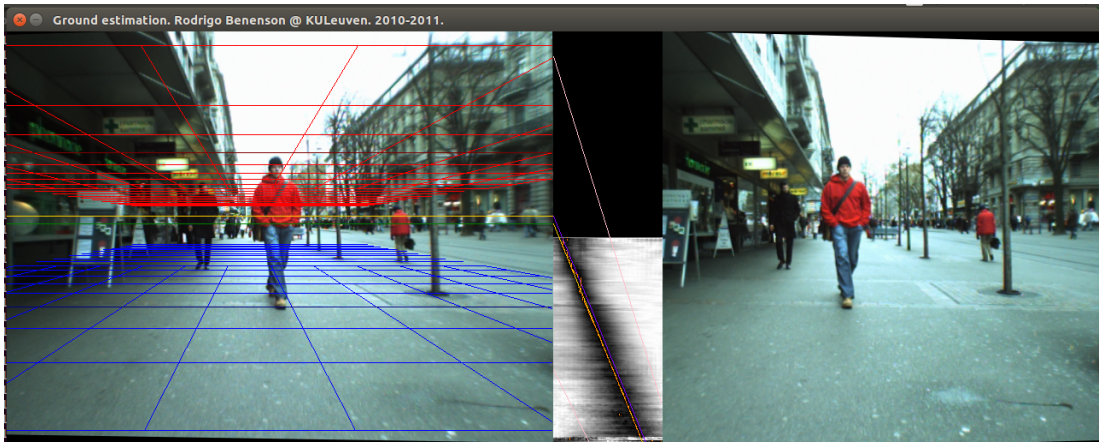

Figure 4.7: Run of the application ground_estimation, loaded with the configuration file test.config.ini.

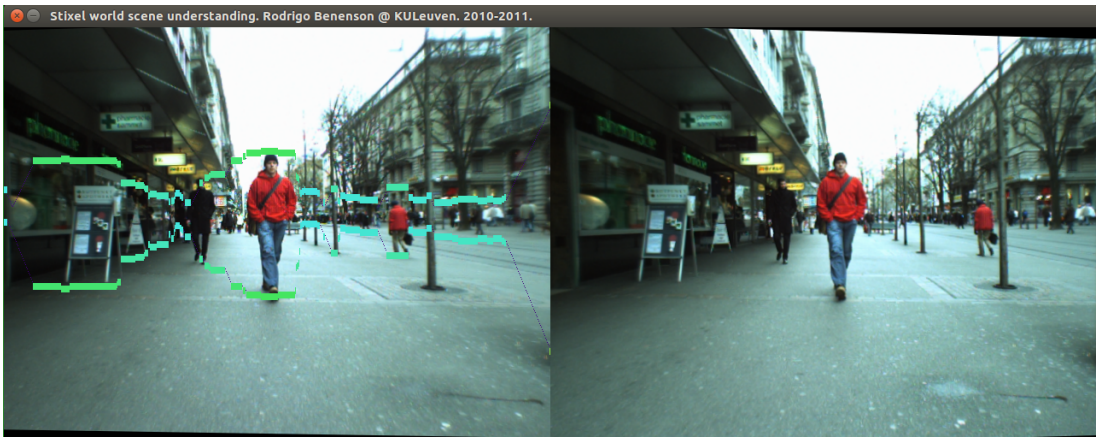

Figure 4.8: Run of the application stixel_world, loaded with the configuration file fast.config.ini.

Chapter 5

Evaluation Tools

This Chapter provides an overview of all the tools used for performing the HW/SW co-design analysis in the thesis.

5.1 Profiling and energy measurements

A central part in developing the design methodology was spent performing program profiling on the Pedestrian Detection System. The profiling was carried out with a selection of tools that are part of the Valgrind tool suite described below, as well as a separate profiling data visualization tool called KCachegrind.

5.1.1 Valgrind tool suite

The Valgrind tool suite was used for measuring the program thorugh its various applications while it was running, and for producing the corresponding profiling output data files.

Callgrind

The Callgrind tool is, as previously stated, a tool that a profiling output was used for producing the callgraph output files that were hence presented visually by the KCachegrind tool. As mentioned, Callgrind is a profiling tool that is part of the Valgrind tool suite. It is built and run in the Linux operating system and is enabled and executed along with the program in the Linux terminal with additional arguments. It runs along with the program being profiled. While the program is running, Callgrind monitors the following data:

- **Ir:** Instruction Fetch, meaning I cache reads and instructions executed.
- **Dr:** Data Read Access, meaning D cache reads in the form of memory reads.
- **Dw:** Data Write Access, meaning D cache writes in the form of memory writes.
- **I1mr:** L1 Instruction Fetch Miss, meaning cache read misses and that instruction wasn't in I1 cache but was in L2.
- **D1mr:** L1 Data Read Miss, meaning D1 cache read misses where data location was not in D1 cache, but in L2.

- **D1mw:** L1 Data Write Miss, meaning D1 cache write misses where location not in D1 cache, but in L2.
- **ILmr:** LL Instruction Fetch Misses.
- **DLmr:** LL Data Read Misses.
- **DLmw:** LL Data Write Misses.
- **L1 Miss Sum:** L1m = I1mr + D1mr + D1mw
- **Last-level Miss Sum:** LLm = ILmr + DLmr + DLmw
- **Cycle Estimation:** CEst = Ir + 10 L1m + 100 LLm

As presented, Callgrind monitors all the successful and unsuccessful instruction calls during the execution of the chosen program application. It was used for attempting to uncover the functionality of the Pedestrian Detection System, and also to pinpoint all the unsuccessful operations and loop-intensive program parts involved in the program execution.

When a program was executed with Callgrind enabled, the experience was that the program run time was slowed down, presumably due to the added processing and monitoring involved. During the project, the amount of experienced run time delay varied quite a lot, and seemed to depend on different factors, among them the amount of times the program had been run previously with profiling enabled.

5.1.2 KCachegrind

KCachegrind is a presentation tool for profiling data that supports data files from a number of different profiling tools. After receiving a profiling output file, it presents the data in a detailed user interface, as depicted in Figure 5.1.

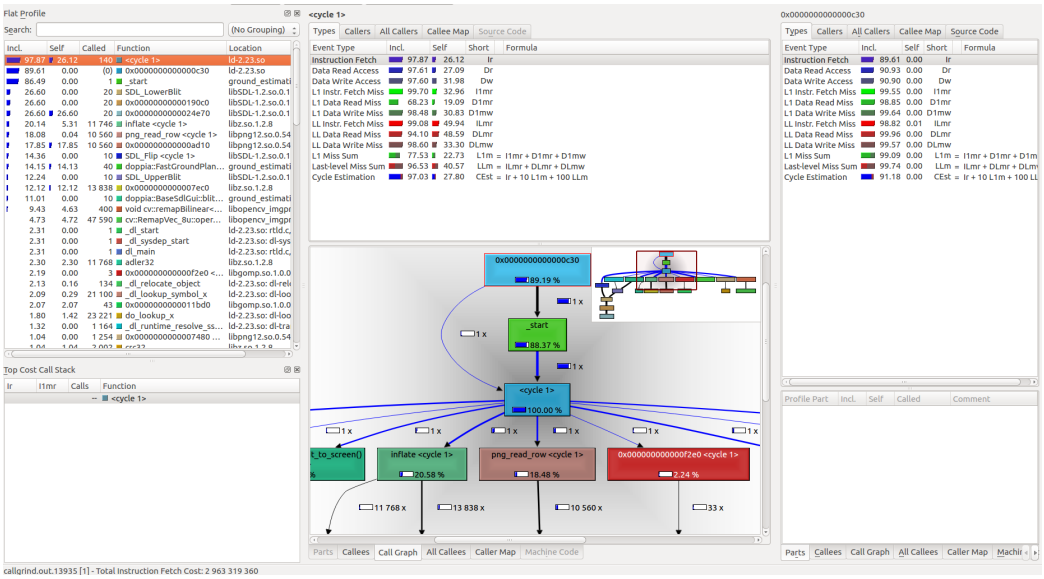

Figure 5.1: Picture of the KCachegrind program (original full resolution available in ZIP-file).

The KCachegrind tool was used for generating a visual call graph for the execution of the Pedestrian Detection System, as illustrated partly in Figure 5.1 and in detail in Figure 5.2.

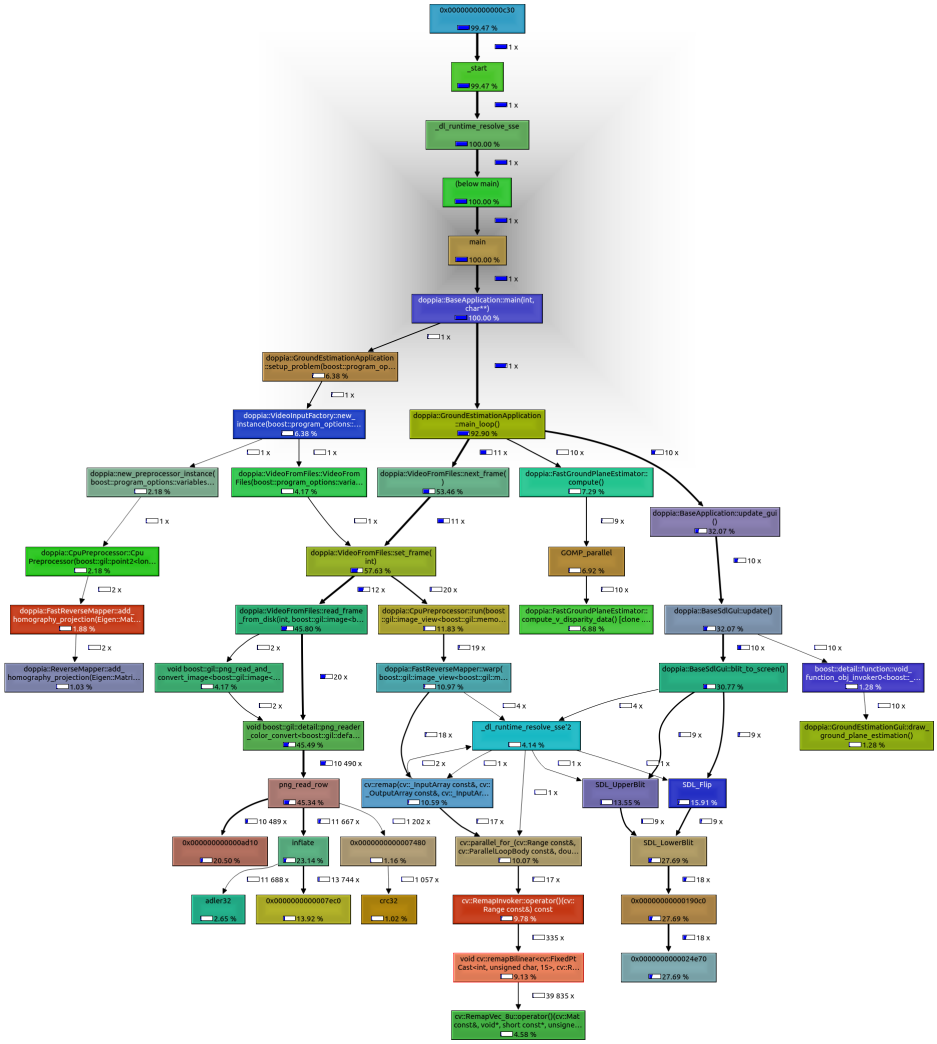

Figure 5.2: Call graph output image generated by KCachegrind, from a run of ground_estimation with the Callgrind tool and cache hit/miss count enabled. Full resolution in ZIP-file.

Chapter 6

Suggested Design Methodology

This Chapter will describe the selected design methodology for the Pedestrian Detection System. The material used for developing this methodology was primarily extracted from the various research work described in Chapter 3: Previous Work, as well as from the basic theoretical background material described in Chapter 2: Theory and Background. The methodology was further designed around the Pedestrian Detection System, and later applied on the system to produce suggestions for an improved design from a HW/SW co-design perspective. The results from the appliance of the methodology are described in Chapter 7.

In addition to the HW/SW co-design principles used as basis for developing the design methodology, the environment in which the Pedestrian Detection System is intended to operate in also influenced the design of the methodology. The PDS is intended to operate in cars, which meant concerns for overheating, physical space restrictions, and also response time to camera inputs were factored into the evaluation criteria as well. With the system's purpose being to spot pedestrians appearing in front of the car, for the sake of improving the traffic safety for pedestrians as well as the driver and passengers of the car. Incorporating this perspective into the methodology was thus considered a necessary course of action.

6.1 Technical approach

As mentioned, the Valgrind tool suite distribution [2] was used for the profiling performed on the system to uncover the PDS' main functions, along with the KCachegrind program for interpreting and presenting the profiling data in a more orderly fashion.

6.2 Code attributes leading to a hardware or software implementation

The program code being executed when running the Pedestrian Detection System in different modes contained several types of common program code statements whom are included in Table **??**. To make a proper HW/SW co-design evaluation of the program code, these statements were evaluated based on their execution mechanics, the amount

of instruction calls they performed during run-time, and also the precentage of time spent executing the statements in relation to the execution time as a whole.

Code functions	**Implementation alternative**
If/else tests	Hardware
Case structures	Software
While loops	Software
Mathematical operations	Hardware
Arrays	Hardware

Table 6.1: Overview of the optimal implementation alternatives for the various program statements being part of the Pedestrian Detection Systems execution.

6.3 Target architecture components

CPU
Memory
Xilinx Design tools for FPGA

6.4 HW/SW Partitioning

6.4.1 Relevant design factors

Energy consumption: Because of the expected scope of work for this thesis, with limited access to optimal measurement tools, the measurements of energy consumption was handeled with a 1. order model, stating that the amount of time spent executing a task provides a rough estimate of the overall energy spent to execute the task.

Design complexity: Viewed in the sense of chip area, maybe not the amount of operations involved.

Abstraction level

Aptitude for either parallel or sequential implementation

Design cost

6.4.2 Granularity depth

The granularity depth used for evaluating the attributes of the PDS was largely decided from the granularity depth available from the profiling tools. The call graph view from KCachegrind, illustrated in Figure 5.2, was set to present all function calls available from the profiling data. The configuration was thus set to:

- Caller depth unlimited
- Callee depth unlimited
- Min.Node cost 1 %
- Min.Call cost 10 % of node

6.4.3 Evaluation criteria for system functions

The evaluation criteria stated in this Subsection describe the criteria used for evaluating the various tasks involved in the call graph view generated from the Callgrind and KCachegrind tool sets. The purpose of this set of criteria is to create a judgement module for the various system functions that the Pedestrian Detection System contains, which is meant to decide whether or not certain functions should be considered for an alternative implementation. The criteria are developed primarily from a HW/SW co-design perspective, but also from the perspective of the environment which the system is to be operating in. The reason for the product environment perspective is the . The criteria are as follows:

HW/SW co-design perspective
-Energy: first order model; run time/instruction calls and data r/w operations
-HW complexity: footprint/chip area
-SW complexity: heavy use of SW functions and statements (if/else, case structures, for

and while loops)

Product environment perspective
-Low heat development
-Low power requirement
-Safety: if fails, fails to a safe state where user is notified of failure. Must then not be used.

Profiling perspective
- Ir, Dr, Dw counts, used to indicate run time or program demand/load/cost, the more it is the stronger
- Ir, Dr, Dw misses, the more the worse

As outlined in Chapter 2, the initial stage of a design process for a system developed with HW/SW co-design principles involves discovering and searching through the various components that the system is to contain, which in this case are the various system functions that are part of the program execution of the PDS. A method for searching through the various software-based system functions of the PDS was therefore developed. This method is formulated as such: 1. By using the Callgrind profiling tool suite for creating an output file, and then using the KCachegrind UI-representation tool to enhance the readability and presentation of the data, a call graph was generated with KCachegrind from the Callgrind output data with a granularity level for instruction fetch cost down to 1 % of the total amount of instructions executed during a program run.

6.4.4 Cost function

$$Cost = 100* \left(\begin{array}{c} A * \frac{Complexitycost}{ConstrainedComplexityCost} + B * \frac{Run - timeCost}{ConstrainedRun - timeCost} + \\ C * \frac{EnergyCost}{ConstrainedEnergyCost} + \\ D * \frac{PowerDissipationCost}{ConstrainedPowerDissipationCost} + E * \frac{Productioncost}{ConstrainedProductioncost} \end{array} \right) \tag{6.1}$$

The design of the cost function used for this methodology was inspired by the work of Abdelhalim et al.(2011) in [5] and [6]. The function is composed of the following design factors:

- Complexity cost
- Run-time cost
- Energy cost
- Power dissipation cost
- Production cost

Chapter 7

Results of Design Methodology

Skal omfatte resultatene fra bruk av designmetodikken på systemet. På grunn av dette ble det gjort sånn, på grunn av dette ble det gjort slik. This Chapter describes the measured results from applying the selected HW/SW co-design methodology on the Pedestrian Detection System.

Factors to note before reading the results are as follows:

- The PDS was designed to be compiled in incremental steps. The first step provides the basic allows the system to run at all, and provides. The next compilation steps introduce more features that are optional. These features are as such not required for running the program, but are by the designers claimed to be was compiled with what the

7.1 Profiling results

7.1.1 Callgrind data

Function name	Ir counts Dr counts Dw counts	I1mr I2mr	D1mr D2mr	D1mw D2mw
???:0x0000000000024e70	788,103,340 290,361,960 75,283,480	640	576,050	672,010
???:0x000000000000ad10	528,811,917 56,461,526 20,327,928	168	35,169	1,479
???:0x0000000000007ec0	359,046,579 92,344,890 20,736,076	682	540,454	6,875
Funksjon nr 4 13935	277,867,551 39,360,464 350,964	368	422	19,534

Table 7.1:
Ir: Instructions executed
Dr: Memory reads
Dw: Memory writes
I1mr: I1 cache read misses (instruction wasn't in I1 cache but was in L2)
I2mr: L2 cache instruction read misses (instruction wasn't in I1 or L2 cache, had to be fetched from memory)
D1mr: D1 cache read misses (data location not in D1 cache, but in L2)
D2mr: L2 cache data read misses (location not in D1 or L2)
D1mw: D1 cache write misses (location not in D1 cache, but in L2)
D2mw: L2 cache data write misses (location not in D1 or L2)

As illustrated in the tables above, the profiling tools yield counts of instructions executed, data reads and writes, and also the amount of times these operations were unsuccessful. This data was used in combination with other types of data to measure the amount of time spent within each function of code.
With the use of the "Call Graph" view from the KCachegrind tool, the functions containing the most cost in terms of the amount of instruction calls they contained were selected from the final elements of the call tree of the detection system. This was due to

them having no calls to other functions, meaning that their cost did not become a sum of the costs associated with other called functions. The functions are listed in Table **??**.

Chapter 8

Discussions and future work

8.1 Applicability for other applications

With the methodology being very specific in its nature, it is unlikely that it will be applicable for many other systems. However, it might be applicable for other software only systems due to its sole focus on software functions.

8.2 Delay from running with callgrind

As mentioned earlier in the report, running the applications for the Pedestrian Detection System with the callgrind profiling tool severely increased the run time of the system. Both the ground_estimation and the stixel_world applications were measured to run for 1 second without any measurement tools enabled, but while the ground_estimation application spent 56 seconds with callgrind enabled, stixel_world spent approximately 10 minutes with the same callgrind configuration. The reason behind this is unclear, but it clearly shows that the two applications are more different than they could otherwise be considered to be. When comparing the annotated callgrind outputs from both applications, it can be seen that the ground_estimation output is severely more detailed, containing more than ten times the amount of lines of text than the output from the callgrind run with stixel_world. It is the student's belief that this might indicate that callgrind could have some difficulty with properly measuring the instruction calls and cache hits/misses from stixel_world. Aside from this issue, there are few observable differences between the two applications.

Chapter 9

Conclusions

The purpose and goals of this thesis wereto select and describe a HW/SW co-design methodology based on an extensive literature study and early investigation into the system code. The student was also tasked with simulating the system with design tools suitable for Xilinx FPGAs, and also if time allowed to implement the system on an actual FPGA evaluation board. Neither of these two tasks were accomplished by the student due to time constraints.

The results from the evaluation of the ground_estimation application points to leaving most of the application unchanged, albeit with some suggested changes to a more concurrent execution of the system.
The results from the evaluation of the stixel_world application also points to leaving most of the application unchanged, but with more suggested changes to a more concurrent execution of the system.
With the results of the evaluations elaborated on, the conclusion is that the Pedestrian Detection System works fairly well in its current state, as both of the tested applications performed well without any profiling tools enabled. It is however a bit unknown what could be the cause of the severe amount of delay experienced with the stixel_world application with the callgrind tool enabled.

References

[1] About cmake. `https://cmake.org/`. last visited: 2018-1-16.

[2] About valgrind. `http://valgrind.org/info/about.html`. last visited: 2017-06-21.

[3] D-17b. `https://en.wikipedia.org/wiki/D-17B`. last visited: 2018-1-12.

[4] Fasttrack to embedded systems. `https://www.digit.in/technology-guides/fasttrack-to-embedded-systems.html`. last visited: 2018-1-9.

[5] M. B. Abdelhalim and S. E.-D. Habib. An integrated high-level hardwaresoftware partitioning methodology. *Design Automation for Embedded Systems*, 15:19–50, 2011.

[6] M. B. Abdelhalim and S. E. –D. Habib. Fast hardware upper-bound power estimation for a novel fpga-based hwsw partitioning scheme. *2008 IEEE Computer Society Annual Symposium on VLSI*, 2008.

[7] M. B. Abdelhalim, A. E. Salama, and S. E.-D. Habib. Hardware software partitioning using particle swarm optimization technique. In *Hardware Software Partitioning using Particle Swarm Optimization Technique*, 2006.

[8] Gregory R. Andrews and Fred B. Schneider. Concepts and notations for concurrent programming. *ACM Computing Surveys (CSUR)*, 15:3–43, 1983.

[9] A. Ferrari and A. Sangiovanni-Vincentelli. System design: traditional concepts and new paradigms. *Proceedings 1999 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1999.

[10] K.M. Deliparaschos G.P. Moustris and S.G. Tzafestas. Feedback equivalence and control of mobile robots through a scalable fpga architecture. In Dr. Andon Topalov, editor, *Recent Advances in Mobile Robotics*, chapter 20, pages 401–427. InTech, 2011.

[11] M. Guarnieri. The age of vacuum tubes: Early devices and the rise of radio communications. *IEEE Industrial Electronics Magazine*, 6:41–43, 2012.

[12] M. Guarnieri. The history of electronics reviewed through the history of magneti marelli factory in pavia, italy. *2015 ICOHTEC/IEEE International History of High-Technologies and their Socio-Cultural Contexts Conference (HISTELCON)*, 2015.

[13] Soonhoi Ha and Jürgen Teich. *Handbook of HardwareSoftware Codesign.* Springer Netherlands, 2017.

[14] K. Keutzer, A.R. Newton, and J.M. Rabaey. System-level design: orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:1523 – 1543, 2000.

[15] M.L. Lopez, C.A. Iglesias, and J.C. Lopez. A knowledge-based system for hardware-software partitioning. *Proceedings Design, Automation and Test in Europe*, 1998.

[16] Patrick O'Connor and Andre Kleyner. *Practical Reliability Engineering.* John Wiley & Sons, 2011.

[17] Jivan S. Parab, Santosh A. Shinde, Vinod G. Shelake, Rajanish K. Kamat, and Gourish M. Naik. *Practical Aspects of Embedded System Design using Microcontrollers.* Springer, Dordrecht, 2008.

[18] A. Sangiovanni-Vincentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, 18:23–33, 2001.

[19] Patrick R. Schaumont. A senior-level course in hardware–software codesign. *IEEE Transactions on Education*, 51:306–311, 2008.

[20] Patrick R. Schaumont. *A practical introduction to Hardware Software Codesign.* Springer US, 2013.

[21] Fred B. Schneider. *On Concurrent Programming.* Springer New York, 1997.

[22] Jens Teubner and Louis Woods. *Data Processing on FPGAs.* Morgan & Claypool, 2013.

[23] Wolf Wayne. A decade of hardwaresoftware codesign. *Computer*, 36:38–43, 2003.

[24] Ass-Prof Dr Wortmann, Felix and Dr Flüchter, Kristina. Internet of things: Technology and value added. *Business & Information Systems Engineering*, 57:221–224, 2015.

[25] Yuanrui Zhang and Mahmut Kandemir. A hardware-software codesign strategy for loop intensive applications. *2009 IEEE 7th Symposium on Application Specific Processors*, 2009.

Appendix

Tips for getting Pedestrian Detection System to run

The necessary software libraries, as well as all system requirements, are described in the readme-file included in the Pedestrian Detection System repository available from bitbucket. This should be looked at before anything else.
As for tips and tricks with getting things to work, one should first and foremost avoid deleting files that are not possible to delete without being a sudo user. This might cause your Linux distribution to crash on start up, and you will probably have to reinstall the whole operating system.
Further one should ensure one has enough available space on the selected storage partition, as getting the Pedestrian Detection System to run from a fresh default installation will require downloading a fair amount of additional libraries and tools.
One should also preferably use a backup program that supports incremental backups, so that one can easily restore to a recent save state in the event of failure. A thing to watch out for in this regard is programs that have scheduled backups enabled by default. If your storage partition is already close to being full, this scheduled backup might run anyway, which will effectively remove all available space on your storage device, thus meaning you will have problems with booting up the system.
For removing files that one did not intend to download, or if one wishes to upgrade to a newer version of the file, consulting google is the way to go. Web pages such as stackoverflow, askubuntu, stackexchange, etc have a lot of answers in general. Some tips for removing are the use of the autoremove command, and also make uninstall.