

MSc THESIS

Power Aware HW/SW Partitioning for a DVB-H Receiver Module

Ioannis Koryfidis

Abstract



CE-MS-2006-11

In this thesis, we consider a DVB-H receiver module and investigate effective means to find the most appropriate partition between software and hardware space (FPGA), while taking into account power consumption, hardware area constraints, and the communication between the blocks of the partition. To find the best partition for the considered application we utilize a simplified version of simulated annealing, which is a common method used in solving combinatorial optimization problems. In order to apply the simulated annealing methodology, we first split the DVB-H receiver in basic building blocks while considering the entire protocol stack. Subsequently, for each an every block, we acquire an estimation of the power consumption when implemented in software and when implemented in hardware. An ARM processor is used for the software implementations and a Stratix FPGA device for hardware implementations. The software power dissipation is estimated using Sim-Panalyzer, an augmentation to the SimpleScalar performance simulator of the ARM, while the hardware power dissipation is estimated using the PowerPlay Power Analyzer, a member of the Quartus II software's PowerPlay suite of power analysis and optimization tools. Based on

those per block estimates the annealing methodology provides a DVB-H system partition, which leads to a reduction in power consumption of up to 35% at the cost of up to 1703 logic elements of additional FPGA hardware area. Our findings are important as they provide the means for an energy effective practical realization of a DVB-H enabled mobile device, like for instance a mobile phone with TV capabilities ("TV-on-Mobile") or a portable TV.

PHILIPS

sense and simplicity

 **TU Delft**

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Power Aware HW/SW Partitioning for a DVB-H Receiver Module

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Ioannis Koryfidis
born in Drama, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Power Aware HW/SW Partitioning for a DVB-H Receiver Module

by Ioannis Koryfidis

Abstract

In this thesis, we consider a DVB-H receiver module and investigate effective means to find the most appropriate partition between software and hardware space (FPGA), while taking into account power consumption, hardware area constraints, and the communication between the blocks of the partition. To find the best partition for the considered application we utilize a simplified version of simulated annealing, which is a common method used in solving combinatorial optimization problems. In order to apply the simulated annealing methodology, we first split the DVB-H receiver in basic building blocks while considering the entire protocol stack. Subsequently, for each an every block, we acquire an estimation of the power consumption when implemented in software and when implemented in hardware. An ARM processor is used for the software implementations and a Stratix FPGA device for hardware implementations. The software power dissipation is estimated using Sim-Panalyzer, an augmentation to the SimpleScalar performance simulator of the ARM, while the hardware power dissipation is estimated using the PowerPlay Power Analyzer, a member of the Quartus II software's PowerPlay suite of power analysis and optimization tools. Based on those per block estimates the annealing methodology provides a DVB-H system partition, which leads to a reduction in power consumption of up to 35% at the cost of up to 1703 logic elements of additional FPGA hardware area. Our findings are important as they provide the means for an energy effective practical realization of a DVB-H enabled mobile device, like for instance a mobile phone with TV capabilities ("TV-on-Mobile") or a portable TV.

Laboratory : Computer Engineering
Codenummer : CE-MS-2006-11

Committee Members :

Advisor: Sorin Cotofana, CE, TU Delft

Chairperson: Stamatis Vassiliadis, CE, TU Delft

Member: Georgi Gaydadjev, CE, TU Delft

Member: Nick van der Meijs, CAS, TU Delft

Member: Joep van Gassel, Philips Research

Contents

List of Figures	v
List of Tables	vii
Acknowledgements	ix
1 Introduction	1
2 Background	5
2.1 DVB-H	5
2.1.1 Time-slicing	6
2.1.2 MPE-FEC	6
2.1.3 IPDC	7
2.2 IC Power Consumption	9
2.3 Software Power Consumption	11
2.3.1 Bus Power	11
2.3.2 Memory Power	11
2.3.3 CPU Power	12
2.3.4 Other Power Dissipation Sources	12
2.4 Hardware/Software Partitioning	13
2.5 Simulated Annealing	15
2.5.1 Overview	16
2.5.2 Pseudo-code	17
2.6 Conclusion	17
3 Methodology	19
3.1 DVB-H System Specification	19
3.1.1 Granularity	19
3.1.2 Functional Partitioning	20
3.1.3 Filtering	22
3.2 Hardware System Realization	22
3.2.1 Target Hardware	22
3.2.2 Tools	23
3.2.3 Level of Abstraction	23
3.2.4 Implementation	24
3.3 Power Estimation	26
3.3.1 Hardware	26
3.3.2 Software	28
3.3.3 Communication between HW-SW	30
3.4 Partitioning using Simulated Annealing	33

3.4.1	Algorithm Considerations	33
3.4.2	Cost Function	34
3.4.3	Other SA Parameters	34
3.5	Conclusion	35
4	Results	37
4.1	Experimental Results	37
4.1.1	Hardware	37
4.1.2	Software	39
4.2	Analysis of the Solution Space	40
5	Conclusion	47
5.1	Summary	47
5.2	Future Directions	48
	Bibliography	52

List of Figures

2.1	Protocol Hierarchy for DVB-H IPDC	8
3.1	DVB-H Functional Partitioning	21
3.2	Common Functionality Blocks	25
3.3	PowerPlay Power Analyzer High-Level Flow	27
4.1	Possible and Proposed Partitions for the H.264 path	44
4.2	Possible and Proposed Partitions for the AAC+ path	45
4.3	Possible and Proposed Partitions for the FLUTE path	45

List of Tables

3.1	IPDC Filtering Parameters	22
3.2	Stratix EP1S10 Features	23
3.3	Sim-Panalyzer Command File Parameters	30
3.4	Signal Toggle Rates	31
3.5	HW/SW Communication Power	33
4.1	Hardware design area values	37
4.2	Power dissipation values for HW implementation (H.264 path)	38
4.3	Power dissipation values for HW implementation (AAC+ path)	38
4.4	Power dissipation values for HW implementation (FLUTE path)	38
4.5	Total Power Dissipation and Hardware Area values for all-hardware im- plementation	39
4.6	Power dissipation values for SW implementation (H.264 path)	39
4.7	Power dissipation values for SW implementation (AAC+ path)	39
4.8	Power dissipation values for SW implementation (FLUTE path)	39
4.9	Total Power Dissipation values for all-software implementation	40
4.10	Power Dissipation reductions of an all-hardware solution over an all-SW solution	40
4.11	Maximum allowed Power Dissipations in order to have Power Reduction .	41
4.12	Results of the SA algorithm for the H.264 path	42
4.13	Results of the SA algorithm for the AAC+ path	42
4.14	Results of the SA algorithm for the FLUTE path	42
4.15	Partitioning Scenarios	42
4.16	Comparison of Partitioning Scenarios	43

Acknowledgements

Ioannis Koryfidis
Delft, The Netherlands
July 21, 2006

Introduction

Since the first television broadcast in England in 1936, technologists and researchers have shown great interest in improving the quality of services related to audiovisual content distribution. The creation of the European Broadcasting Union (EBU) in 1950 initiated a continuous provision of technical information to broadcasters all over Europe, so that knowledge and experience could be easily and freely shared. Through these ideals, the Digital Video Broadcasting (DVB) [1] Project was born, aiming primarily to standardize the technology involved in the creation and distribution of digital television signals. Distribution of data by satellite (DVB-S) and cable (DVB-C) means, was ratified in 1994, while distribution over terrestrial (DVB-T) means was ratified in 1997.

Following the evolution of broadcasting, new dynamics in the consumer electronics market were created. In the last decades, electronic devices got smaller, so that today they can even fit in someone's palm. The flourishing electronics market has been continuously penetrated by new mobile products and the broadcasting industry had no choice but to be involved. Digital Video Broadcasting - Handheld (DVB-H) [2] was introduced as a technical specification for bringing broadcast services to handheld receivers in November 2004. Among the new features of DVB-H in comparison with its predecessors, Time Slicing and MPE-FEC are the most important, since they facilitate the reduction of power consumption, the most significant problem of battery-powered handheld and mobile devices.

Power consumption has been a major bottleneck in the design of efficient mobile devices, since the technology involving the development of batteries with bigger lifetimes is advancing very slowly. The major power consumption sources in a mobile device, which is a type of an embedded system, are the ones related to frequent charging and discharging of the capacitive loads and the ones related to leakage currents of logic gates. Software applications running on an embedded platform are also responsible for the power dissipated in the platform's CPU, memory, and buses. Consequently, as the requirements for functionality in mobile devices are constantly increasing, technologists have to find efficient ways to cope with the power consumption problem.

Several ways, besides increasing battery life, have been proposed in order to solve the power consumption problem of embedded systems. Clock gating [3] techniques aim to inhibit the frequently switching clock and any other high activity signals when a logic block is not in use. Moreover, supply voltage reduction results in substantial power savings, since power reduces quadratically with supply voltage. Maximum performance is delivered only when supply voltage is high and when the performance demand is low, lower operating voltages are applied. Furthermore, leakage reduction has been extensively studied [4],[5],[6],[7] in order to reduce standby leakage in transistors. Despite their effectiveness, these techniques apply to lower levels of design abstraction and few

work has been done in the system level [8],[9].

Recently, Philips introduced its next generation DVB-H front-end System-in-Package solution, BGT215 [10]. In this competitive market it is of key importance to offer differentiating products to its customers. Philips Research works together with Philips Semiconductors on DVB-H solutions and they aim to differentiate from the competition by reducing the overall power consumption of a DVB-H enabled mobile device. Our project, in cooperation with Philips Research, was to find effective means to reduce the power consumption of a reference system consisting of a DVB-H module and an ARM-based application processor. A reference software implementation (written in C++) of the DVB-H protocol stack that could run on an ARM processor was already available and the idea that predominated over other power reduction techniques was that of extracting portions of the software running application as potential candidates for hardware realization, where they would consume less power. This type of problem is called hardware/software partitioning problem.

Hardware/Software System Partitioning techniques were initially targeting performance optimization of embedded systems, since generally hardware execution is faster than software. The same though applies for power consumption. The implementation of a software-running application in dedicated hardware (FPGA or ASIC) substantially reduces power consumption in the majority of the cases. Nevertheless, cost and size constraints of practical hardware systems are an obstacle in moving the whole functionality to dedicated hardware, while hardware/software partitioning provides a solution by extracting only a portion of an application to hardware, according to some design constraints.

The main partitioning technique that we primarily envisaged was to split the DVB-H protocol stack functionality in basic building blocks according to a chosen granularity and estimate the power consumed both by software and hardware implementations of each and every block. Then, we could identify which hardware/software partition would provide power consumption reductions over an all-software solution, using an efficient method that also examines hardware area as a constraint. Moreover, a significant factor that could not be omitted from our estimations was the communication between hardware and software. The initial partitioning contained building blocks corresponding to the DVB-H receiver's filtering functionality between the network and the application layer. Consequently, three data paths were considered; the H.264 video extraction, the AAC+ audio extraction, and the FLUTE file delivery paths, all belonging to the Internet Protocol Data Cast (IPDC) functionality of DVB-H.

In order to estimate the power consumption of hardware, a hardware implementation of the reference software was necessary. For this reason, the Altera's EP1S10 Stratix FPGA [11] was used as a reconfigurable device. The hardware design was done in VHDL under the HDL Designer Series suite from Mentor Graphics [12]. The implemented design was simulated using the ModelSim SE tool from Mentor Graphics [13] and for the estimation of the power dissipation of each building block, the Quartus II PowerPlay Power Analyzer [14] was used. The simulation provided Power Analyzer with the necessary signal activities for the power consumption calculation. A DVB-H specific MPEG-2 Transport Stream was used as an input to create these activities. Quartus II design synthesis also provided the hardware area of each building block, which was later used

as a design constraint for our partitioning algorithm.

For power consumption estimation of the software implementation, Sim-Panalyzer [15], an augmentation to the SimpleScalar [16] performance simulator of the ARM processor was utilized. In order to acquire the estimation results, individual software modules corresponding to the building blocks of the initial partition had to be prepared. Since Sim-Panalyzer accepts ARM executables as input, an ARM-Linux cross compiler had to be built in a Linux platform in order to translate the C++ code to executable binaries targeting the ARM processor. A combination of compiler tools and utilities assisted in this step.

The power consumption due to communication between hardware and software building blocks was estimated by considering power dissipated on the bus lines due to the transfer of data. Using the estimation scheme in [55], the switching activity of bus signals was necessary. In order to acquire that, the Hamming distances (how many bus lines are switched from 0 to 1 or from 1 to 0 per clock cycle) were calculated using simulation results from ModelSim. In [55] we could also find estimation methods for bus capacitive loads, corresponding to the technology that we employed.

Having power estimations for the hardware and software implementations, as well as power estimations for HW/SW communication and hardware areas required to include hardware building blocks to the FPGA, we could move on to our partitioning methodology. The size of the problem was relatively small (4-5 building blocks per data path), so it could be easily solved by exploring all the possible partitioning combinations. Nevertheless, Simulated Annealing, a common method used in solving combinatorial optimization problems, was utilized. Simulated Annealing's ability to escape from local minima would substantially reduce the exploration time for the best partitioning scenario in larger problems, which will potentially occur in future research. The employed Simulated Annealing algorithm provided us with the overall power consumption values and hardware areas for the best partitioning scenarios, according to a predefined cost function. This cost function was dependent on power consumption and hardware area and penalized moves that were crossing boundary values during searching, especially the ones corresponding to power consumption boundary violations, in order to speed up the execution time of the algorithm.

So as to identify the boundaries that we had to set to the Simulated Annealing algorithm, we simply assumed that power reduction over an all-software solution can occur only when the overall power consumption of the resulting partition does not surpass the FPGA's static power component. This component is present by the time we choose to implement any building block or blocks in hardware, so it is included in every partition except from the all-software case. Eventually, four power consumption constraints were used, corresponding to 10%, 20%, 30% and more than 30% power reduction over the all-software solution, expecting the Simulated Annealing algorithm to find the most appropriate nearby solution.

The results showed that, utilizing HW/SW partitioning with Simulated Annealing, power reductions of up to 35% in comparison to the all-software solution could be achieved, at a cost of up to 1703 Logic Elements of hardware area, and the Simulated Annealing algorithm was correctly choosing between nearby solutions, according to the cost function.

This thesis is organized as follows:

- Chapter 2 provides the necessary theoretical knowledge on DVB-H, IC and software power consumption, HW/SW partitioning, and Simulated Annealing.
- Chapter 3 introduces the system's specifications and presents the power estimation methodology for the hardware and software implementations and the hardware/software communication. Our Simulated Annealing algorithm is also discussed and its parameters are explained.
- Chapter 4 presents the power estimation results and the outcomes of the Simulated Annealing algorithm for HW/SW partitioning are discussed.
- Chapter 5 concludes this thesis with a summary, and provides possible directions for relevant future research.

Background

The purpose of this chapter is to provide the reader with the theoretical background on the key aspects of this thesis. First, an introduction to the DVB-H standard [2] is given in Section 2.1. Sections 2.2 and 2.3 introduce the fundamentals in IC Power Consumption and Software Power Consumption, respectively. The HW/SW partitioning problem is formulated in Section 2.4, while the basics of Simulated Annealing are presented in Section 2.5.

2.1 DVB-H

The Digital Video Broadcasting Project (DVB) [1], [2] is, since 1993, an industry-led consortium of over 270 broadcasters, manufacturers, network operators, software developers, regulatory bodies and others in over 35 countries committed to designing global standards for the global delivery of digital television and data services. Services using DVB standards are available on every continent with more than 120 million DVB receivers deployed.

DVB consists of a set of internationally accepted, open standards for digital television, defining the physical layer and data link layer of a distribution system. DVB systems distribute data by satellite (DVB-S and DVB-S2), cable (DVB-C) and terrestrial television (DVB-T). Besides audio and video transmission, DVB also defines data connections and all data are transmitted in MPEG-2 transport streams with some additional constraints.

In 1998, the Digital Video Broadcasting Project initiated research related to mobile reception of DVB-Terrestrial (DVB-T) [17] signals. Later in 2000 the Motivate (Mobile Television and Innovative Receivers) Project concluded that mobile reception was indeed a realistic target, but it would require dedicated broadcast networks, which would conform to the increased demands of mobile services. In 2002, via the EU-sponsored Multimedia Car Platform (MCP) Project, excellent antenna diversity reception behaviors were explored, sufficiently improving reception performance of DVB-T signals, especially for mobile devices. Starting from the year 2002 and considering market's inclination to mobile broadcast services, system requirements were examined, concluding officially to the creation of the Digital Video Broadcasting - Handheld (DVB-H) standard EN 302 304 by the European Telecommunications Standards Institute (ETSI) in November 2004 [18].

The features of such a transmission system serving mobile devices are derived from these devices' requirements as follows:

1. Since batteries constitute the power source of mobile devices, the transmission system must support power-offs in the receptor's chain for certain periods in order to increase battery lifetime.

2. Since mobile devices' users are not stationary, an easy access to services while changing from one transmission cell to another must be supported by the transmission system.
3. Since the working environment of mobile devices suffers from severe man-made noise and mobile multipath channels, the transmission system must assist on the reception quality and flexibility.
4. Additionally, since compatibility is a key issue, the transmission system must be backwards compatible with existing networks and implementations.

The DVB-H system meets the above requirements via its specification. It is important to mention that DVB-H is based upon the DVB-T standard and DVB-H is totally backwards compatible with DVB-T. The base of this backward compatibility is that the new features of DVB-H are not implemented in the DVB-T physical layer, but in the link layer instead. This means that DVB-T receivers are not affected by new DVB-H parameters.

We mention two significant elements of DVB-H in the link layer that are new compared to DVB-T: Time slicing and Multiprotocol Encapsulation - Forward Error Correction (MPE-FEC). In the remainder of this subsection we briefly discuss them and the fundamentals of IP (Internet Protocol) datacast over DVB-H, as they are relevant to the research performed in this thesis. The differences in the physical layer are not discussed here but the interested reader can find them in the DVB-H specification [18].

2.1.1 Time-slicing

The MPEG-2 Transport Stream (TS) [19] is the means of transmitting information in the DVB standards. The way of carrying IP datagrams in an MPEG-2 TS is realized using Multiprotocol Encapsulation (MPE) [20]. Each IP datagram is encapsulated into one MPE section. An Elementary Stream (ES) is then formed out of MPE sections with a particular program identifier (PID).

A typical requirement of DVB-H devices is to receive audio and video services that are transmitted over IP on ESs that have a relatively low bitrate in the order of 250kbps. However, the MPEG-2 TS supports bitrates up to 10Mbps, which means that the relevant ES occupies only a small part of the total MPEG-2 TS bitrate in mobile dedicated ESs. Demodulating and decoding only this part instead of the whole TS would be ideal for reducing the power dissipation of the system. This becomes reality with the use of Time-Slicing, where MPE sections are sent in bursts of high bitrate instead of continuous low bitrate. In the time interval between consecutive bursts the receiver can be turned-off, substantially reducing power dissipation. Synchronization errors are avoided via the delta-T parameter, which is transmitted in the header of each section in a burst and points to the beginning of the next burst.

2.1.2 MPE-FEC

Using the MPE-FEC scheme, DVB-H offers a very convenient way to protect data from transmission errors. MPE-FEC is based on Reed-Solomon parity data (RS data) cal-

culated from each burst's IP datagrams. IP data and RS data are encapsulated into MPE-FEC sections, which are transmitted right after the last MPE section of a burst in the same ES but with different table_id value, which makes it easy for the receiver to discriminate between these two different types of sections. An MPE-FEC frame is used for the calculation of the RS data, consisting of an application data table (ADT), and an RS data table. In the ADT the IP datagrams and possible padding are included. The number of rows of the MPE-FEC frame is 256, 512, 768 or 1024 and the number of columns is 191 for ADT and 64 for RS data (all numbers representing bytes). The direction layout of the IP datagrams in the frame is vertical, column-by-column, starting in the upper left corner. The 64 RS data bytes of one row are calculated from the 191 IP datagram bytes of the same row, making use of a (255,191) Reed-Solomon code. In the decoding side, this technique proves to be very powerful, resulting in nearly 10-12 ratio of uncorrected frames after MPE-FEC decoding, assuming 10% section loss probability, an initial CRC-32 detection of erroneous sections and an application of an erasure-based decoding of the RS (255,191) code [2].

2.1.3 IPDC

IP datacast (IPDC) [21] refers to the set of specifications created in order to fill the gap between independent networks. With IP datacast it is possible to have a single convergent terminal, which in case of DVB-H is a mobile device, that consumes a variety of broadcast content and services, with full compatibility of the networks involved. This happens in the higher ISO/OSI layers, by specifying the corresponding protocols. IP Datacast is already finalized by the ad-hoc group CBMS (Convergence of Broadcast and Mobile Services). The specification defines the electronic service guide, service access management, delivery protocols, bearer signaling, QoS, mobility and roaming. Figure 2.1 shows the protocol stack of the IP Datacast transmission system. Real-time content, such as real-time audiovisual content, is delivered via Real-Time Protocol (RTP) [22], while non-real-time content is delivered via File Delivery over Unidirectional Transport / Asynchronous Layered Coding (FLUTE/ALC) [23] data carousel. The navigation between services is realized via an XML based Electronic Service Guide (ESG), while H.264 is used as a video coding format and AAC+ is used as an audio coding format. In the rest of the subsection a brief introduction of the underlying protocols for real-time and non-real-time content transport is given.

2.1.3.1 Real-Time Content

The transport of audiovisual content is realized over RTP as defined in RFC 3550 [22]. This is a generic specification; independent of the content, while individual RFCs define how the transport of coded video and audio is realized.

Transport of H.264/AVC video The transport of coded video is described in RFC 3984 [24]. The H.264/AVC specification [25] introduces the Video Coding Layer (VLC) and the Network Abstraction Layer (NAL) as the basic elements describing video content. VCL is responsible for the video features of H.264, such as transform, quantization, motion compensation, loop filter, etc., while NAL is responsible for the formatting of

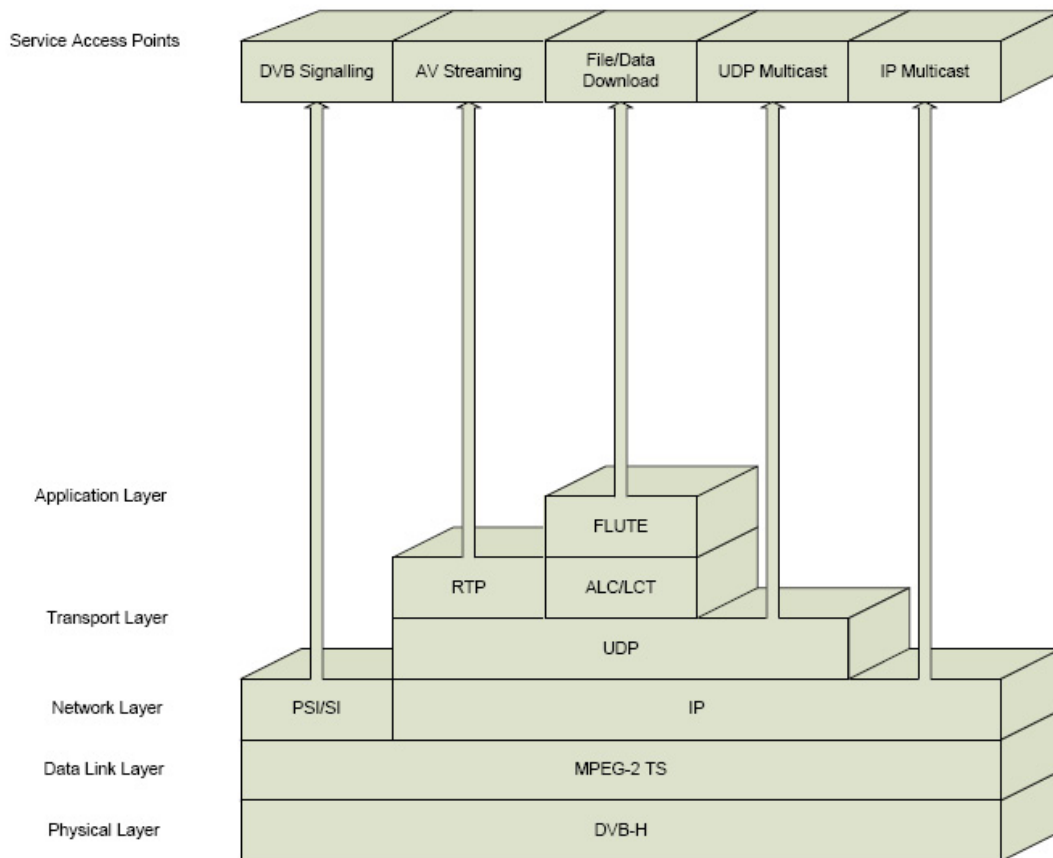


Figure 2.1: Protocol Hierarchy for DVB-H IPDC

VCL data into NAL units, appropriate for transport over the utilized network. The NAL unit's header before the payload is one byte indicating the type of the NAL unit, the presence of errors or syntax violations in the payload and additional information important for the decoding process.

Transport of HE AAC v2 audio The transport of High Efficiency AAC v2 audio is described in RFC 3640 [26]. The framing structure supports carriage of multiple AAC frames in one RTP packet. Each RTP packet contains an integer number of Access Units (AUs), which are byte aligned. An AU consists of an AU header and payload. Normally, the AU payload contains the data of one AAC frame, but the case of multiple AUs carrying data that were previously split is also supported. Moreover, interleaving for error resiliency improvement in packet loss is used.

2.1.3.2 Non-Real-Time content

For non-real-time content RTP is not suitable. Instead, UDP/IP as defined in RFC 768 [27], is used, independent of the IP version and the link layers underneath.

File Delivery Services over FLUTE IP Datacast File Delivery is based on FLUTE [23]. FLUTE resides over the Asynchronous Layered Coding (ALC) protocol instantiation [28]. For IPDC file delivery, ALC consists of a Layered Coding Transport (LCT) block [29] and a Forward Error Correction (FEC) block [30]. ALC uses the LCT building block to provide in-band session management functionality. LCT inherits from ALC several specified and non-specified fields. FEC is used to ensure reliable reception of data, even though its presence is not mandatory. ALC can be used to transfer binary objects of finite or variable length size. Special purpose objects, like the File Delivery Table (FDT) instances, are used to describe FLUTE sessions and provide the relevant parameters.

2.2 IC Power Consumption

In the digital electronics field, the term power consumption is referred as the electrical energy per time unit that has to be provided to a digital circuit in order to maintain its operation [31]. So, power consumption is the effect of the intended functionality of a device. One part of the power is dissipated in order to keep this functionality running, while another part is “wasted” as thermal dissipation.

Power consumption in digital circuits is measured in units of Watts. One Watt is equivalent to one Joule of energy per elapsed second. So, $1Watt = 1 \frac{Joule}{sec}$

We can identify three major sources of power dissipation in digital CMOS circuits [32], namely switching, short-circuit, and leakage. These are summarized using the following equation for the total average power:

$$P_{avg} = P_{switching} + P_{short-circuit} + P_{leakage} \quad (2.1)$$

The first term of Equation (2.1), which is frequently the dominant one, refers to the switching component of power and it is further defined as:

$$P_{switching} = C_L \cdot V_{dd}^2 \cdot f_{clk} \cdot a \quad (2.2)$$

where:

C_L is the load capacitance,

V_{dd} is the device supply voltage,

f_{clk} is the clock frequency and

a is the node transition activity factor, which can be further defined as the average number of transitions between 0-1 or 1-0 states in the output of the node per clock cycle.

$P_{switching}$, as expressed in Equation (2.2), is present when there is a charge of the capacitive load C_L due to the transition from 0 to supply voltage V_{dd} (in a PMOS transistor), or when there is a dissipation of the energy stored in the capacitor due to the transition from supply voltage V_{dd} to 0 (in a pull-down NMOS transistor). Switching does not often happen at the clock rate, unless we are referring to clock buffers, but rather in a probabilistic way characterized by the activity factor a . It is obvious that $P_{switching}$ is highly dependent on the overall activity of the digital circuit, and it is considered to be one of the defining factors for the whole system's power dissipation.

The second term of Equation (2.1) refers to the short-circuit power dissipation and it is further defined as:

$$P_{short-circuit} = I_{sc} \cdot V_{dd} \quad (2.3)$$

where:

I_{sc} is the direct-path short circuit current and

V_{dd} is the device supply voltage.

$P_{short-circuit}$, as expressed in Equation (2.3), is present when there is simultaneous activation of both NMOS and PMOS transistors, creating a direct current pathway between ground and supply voltage V_{dd} . This occurs in the interval between rise/fall time during switching in the input of the logic gates. Short circuit currents are significant when the rise/fall time at the input of a gate is much larger than the output rise or fall time. Hence, equal input and output edge times are advised for a low short circuit current and power dissipation.

Finally, the third term of Equation (2.1) refers to the leakage component of the power dissipation and it is further defined as:

$$P_{leakage} = I_{leakage} \cdot V_{dd} \quad (2.4)$$

where:

$I_{leakage}$ is the leakage current and

V_{dd} is the device supply voltage.

The main characteristic of $P_{leakage}$ is that it is defined primarily by fabrication technology and it is dependent on substrate injection and subthreshold effects. The substrate injection effect occurs when a transistor switches off and another transistor charges up or down the drain according to the bulk potential of the inactive transistor. On the other hand, the subthreshold effect occurs when the voltage between gate and source V_{gs} has exceeded the weak inversion point, while it is still smaller than the threshold voltage V_t , where carrier drift is dominant. This phenomenon is called carrier diffusion between the source and the drain. Since the supply voltages have gradually decreased the last years in order to compensate high power dissipation due to switching activities of ICs operating at high frequencies, threshold voltages had to be proportionally reduced. This reduction though raised subthreshold leakage exponentially, reaching in many today's ICs 50% of the total power dissipation.

2.3 Software Power Consumption

The overall power dissipation of an embedded system does not only originate from the application - specific hardware, but also from the CPU, the memory and the address and data buses when an embedded application is running on the platform's microprocessor, microcontroller or digital signal processor. We refer to the above power dissipation as software power dissipation. Obviously, the power is actually dissipated in the processor's hardware, but it is as a consequence of executing an application program. There are a number of sources of power dissipation influenced by software and contributing to the overall power dissipation of the system, which are explained below.

2.3.1 Bus Power

Busses in an embedded system consist of unidirectional address bus lines and instruction bus lines (opcodes to be executed) and bidirectional data bus lines. With these groups of interconnecting lines, the communication of the CPU with the memory, I/O circuits and peripheral modules is established. Each of the above lines can be modeled as an RC-transmission line, where R and C are the line's resistance and capacitance, respectively. Activation of a line prompts for the charging or discharging of the capacitive load, depending on the previous value that this line had. For example, a transition in an 8-bit data bus between words 00101011 and 11100111 implies charging of 3 lines and discharging of 1 line. Usually, bus charging and discharging of I/O lines can occur up to 80% of the software execution time. There exist coding techniques, like bus invert coding [33], which reduce the power dissipated due to this switching activity in an embedded system's bus. We discuss bus power consumption in more detail in Chapter 3.3, where we estimate the communication costs between hardware and software modules.

2.3.2 Memory Power

Power dissipated by memory read and write accesses is usually one of the dominating components (ranging from 10% - 25%) of the total software power dissipation for mobile devices and portable computers [34]. In the case of DSP applications, where a significant amount of data is processed, this contribution can be substantially higher. Memory power dissipation has a number of components, namely power dissipated in the cell array, in the decode logic and sense amplifier as well as power dissipated due to charging and discharging of the address or data lines capacitances. The type of access is also significant in how much it contributes to the overall power dissipation. In the ARM microprocessor, which is considered in our system, CPU cycles are divided in S-cycles, N-cycles, and I-cycles. The S-cycles refer to sequential memory accesses, where the next word is returned from the same buffer, dissipating a relatively small amount of power. Power dissipation in the address lines during sequential memory accesses is also small, due to the fact that the address word changes only in one bit. If the last memory location of a page is to be accessed though, more power has to be consumed in order to access the next page. The N-cycles refer to non-sequential memory accesses. In this kind of access, more power is dissipated relative to sequential accesses because consecutive address words are irrelevant, causing large activity in the address bus. In N-cycles usually different pages

have to be accessed, contributing to more power dissipation, as explained in the previous paragraph. Finally, the I-cycles refer to cycles that no memory access is involved, so no power is dissipated in the memory system. One more significant contributor to the power dissipated in the memory is the memory access patterns, affecting mainly the cache hits and misses. The cache, residing closer to the CPU than main memory, dissipates less power because the address and data lines are shorter and have less internal capacitance. Inappropriate memory access patterns lead to cache misses and, consequently, power expensive main memory accesses.

2.3.3 CPU Power

When instructions are executed in the CPU, they contribute significant power to the overall power dissipation. We can divide these instructions in four broad categories:

- Load / Store instructions
- Branch instructions
- Type-1 Arithmetic instructions, such as addition, subtraction, shift etc.
- Type-2 Arithmetic instructions, such as multiplication and division.

If we assume that the average power consumption to execute one instruction is W_j , where j represents one of the categories mentioned above, and I_j is the number of times this instruction is executed, we can easily derive the CPU power dissipation as:

$$P_{CPU} \propto \frac{\sum_{j=1}^4 (W_j \times I_j)}{\sum_{j=1}^4 I_j} \quad (2.5)$$

The power dissipated when an arithmetic instruction is executed depends primarily on the ALU or FPU data path that it instructed by software. Many different ways of optimization exist in this context, as for example replacing a division by power of two with corresponding right shifts.

Finally, instruction scheduling is very important, because unsuccessful scheduling can lead to pipeline stalls, which in turn consume a considerable amount of power.

2.3.4 Other Power Dissipation Sources

There is a number of additional sources of power dissipation during software execution that must be taken into account, as they contribute as an overhead to the overall power dissipation. These sources are the clock distribution and the control logic and they accompany code execution in each cycle. In [35] it was shown that short code sequences in a number of microcontrollers and DSPs always dissipated a smaller amount of power than longer sequences. The program in longer code sequences, which demanded extra

execution cycles, took more time to execute so that the overhead was more than that in shorter code sequences.

Despite this problem, power management techniques nowadays tend to eliminate such overhead by cleverly dealing with power dissipation that does not have a direct contribution to the involved computational tasks .

2.4 Hardware/Software Partitioning

In embedded system design, a popular way of achieving boosts in the system's performance is to extract portions of a software running application as potential candidates for hardware realization. Designers often augment a microprocessor with custom hardware to achieve that [36]. Unfortunately though, the cost and size constraints of practical hardware systems make it impossible to realize large applications only in hardware. Therefore, designers frequently consider solutions comprised of both software and custom hardware, working together to meet a set of constraints. This extraction of hardware and software components from a system specification is referred as hardware/software partitioning. Partitioning of a system affects many of its aspects, with execution time, hardware area and cost, and power dissipation being the most concerned. Due to the fact that, typically, system partitioning has to occur at a high-level in the design flow, the system's characteristics related to a resulting partition solution can only be estimated. These estimations are the input to a cost function that evaluates the quality of the solution and instructs for the appropriate decisions.

Hardware/Software partitioning belongs to the CAD optimization problems group. The partitioning methods attempt to divide a set of interconnected modules, and derive a solution that meets a number of design constraints. This methodology can be applied from the lowest to the highest level of abstraction. Starting from the lower level, partitioning is possible by grouping highly interconnected gates, reducing in that way the amount of interconnect delay on wires between gates. As we move to higher levels of abstraction, the elements of the system become substantially larger. From logic cells we pass to macro cells in the logic level, while in the highest architecture level we consider entire system blocks. It is obvious that the effect of a partition to the system's characteristics as we move to higher levels of abstraction becomes drastically bigger. For example, the interconnect delay becomes more pronounced, as it is orders of magnitude larger than in lower levels of abstraction. So, a correct partitioning is crucial for the overall quality of the system.

In literature, the hardware/software partitioning problem has been extensively studied. Initial work on hardware/software partitioning [37], [38] focussed mainly on optimizing the system's performance, while minimizing the amount of hardware area under timing constraints was a further target [39], [40]. Optimizing power dissipation using hardware/software partitioning was not extensively studied, as we can identify only a few partitioning methodologies addressing this problem. In [9], clusters of operations/instructions are mapped to a core that yields a high utilization rate of the involved resources (ALUs, multipliers, shifters, etc.) and thus minimizing power consumption, while in [41] a task-level co-design methodology is introduced that optimizes for power consumption and performance.

In the remaining of this section, we formally define the partitioning problem. Fundamentally, the k -way partitioning problem is the generic form of any partitioning problem and it can be formulated as:

Definition 2.1 *Given a set of modules $M = \{m_1, m_2, \dots, m_n\}$, the k -way partitioning problem finds a set of clusters $P = \{p_1, p_2, \dots, p_k\}$ such that $p_i \subseteq M$ for $i \in [1, k]$, $\bigcup_{i=1}^k p_i = M$, $p_i \cap p_j = \emptyset$ for $i \in [1, k]$, $j \in [1, k]$ and $i \neq j$ (i.e., the clusters are mutually disjoint).*

The resulting solution must satisfy a set of constraints by optimizing a problem-specific cost function. When $k = 2$, which is the case in hardware/software partitioning, the problem is called a bipartitioning problem.

The way partitioning is going to be made depends on some estimation function. The estimation function $E(P) = DP(dp_1, dp_2, \dots, dp_q)$ has a specific partition P as input and a set of design parameters DP as output. Hardware area and cost, performance and power dissipation belong to this group of design parameters. Using the estimation function we can evaluate if the constraints that we have initially put are met and accordingly decide whether a solution is better than another one. The quality of the partitioning is, therefore, highly dependent on the reliability of the estimation function. The clearest definition of hardware/software partitioning is presented in [42].

Definition 2.2 *Given a set of functions $F = \{f_1, f_2, \dots, f_n\}$, so that altogether constitute the complete functionality of the system, and a set of constraints $C = \{C_1, C_2, \dots, C_n\}$ where $C_i = \{G_i, V_i\}$, $G_i \subset F$ and $V_i \in \mathbb{Z}^*$ (V_i is a constant in function group G_i . Each function f_i can represent system functionality at nearly any granularity: as coarse-grained as an entire program task or as fine-grained as a single micro-operation on a processor.)*

Define: A hardware/software partition as $P = \{H, S\}$, where $H \subset F$, $S \subset F$, $H \cup S = F$ and $H \cap S = \emptyset$.

In the above partition P , H represents hardware clusters and S represents software clusters. The power dissipation (which is our point of interest) of a set of functions G is defined as the total power dissipation of the functions in G , or

$$Power(G) = \sum_{f_i \in G} power_dissipation(f_i) \quad (2.6)$$

We call all partitions that satisfy $Power(G) \leq V_i$, $i \in [1, m]$ as power-satisfying partitions. The hardware/software partitioning problem is to find a power-satisfying partition $P = \{H, S\}$ so that the hardware area $HA = min$.

Additionally we call all partitions that satisfy HA , where A is a hardware area constraint, as area-satisfying partitions. In this case the hardware/software partitioning problem would be to find an area-satisfying partition $P = \{H, S\}$ so that $Power(G) = min$.

The two problems previously described are just a subset of the general partitioning problem. Defining other constraint parameters, similar problems can be solved.

The Hardware/Software Partitioning problem is NP-complete, as it requires the exploration of a design space that is exponential in size (relative to the number of functions in F). Therefore, the time required to evaluate a partitioning solution is proportional to the cardinality of F ($\|F\|$), which implies that the granularity of the system has a big impact on the performance of the algorithm. Nevertheless, making function granularity coarser would potentially decrease the quality of the solution because the flexibility will decrease, meaning that a variety of pathways to the optimal solution will not be examined.

Finally, Hardware/Software Partitioning must be performed early in the design cycle; as for example estimations of hardware area must be derived before hardware synthesis. In general all design characteristics have to be evaluated in an early stage and they have to be as accurate as possible because they determine the quality of the solution.

Summarizing, we observe that the hardware/software partitioning problem is very complex, as the quality of the solution it provides is dependent on many non-trivial factors like the performance and accuracy of the estimators, the granularity of the functions involved, and the quality of the search heuristic.

2.5 Simulated Annealing

Simulated annealing (SA) [43] is a generic probabilistic meta-algorithm for global optimization problems. It locates a good approximation to the global optimum of a given function in a large search space.

It is known that all exact methods for determining an optimum solution to a given function require a computing effort that increases exponentially with the number of variables N influencing the solution. We categorize these kind of problems as NP-complete (non-deterministic polynomial time complete) problems, which cannot be solved in polynomial time. In complexity theory, the NP-complete problems are the most difficult problems to solve in the NP set of decision problems. Heuristic methods with computational requirements proportional to powers of N can be applied to give a solution in NP-complete class of problems, but because they are problem-specific, they do not guarantee effectiveness for a variety of problems.

The idea behind simulated annealing, as it originates from the annealing technique of heating and controlled cooling of a material in metallurgy, is that in every SA algorithm step, the current solution is replaced by a random nearby solution that is selected with a probability depending on the difference between consecutive solutions and a global parameter T , which is called temperature. Temperature is decreasing gradually during the algorithm, allowing even “wrong” random moves (accepting solutions that do not improve in comparison with the previous) for high temperatures and being more selective for lower temperatures. This acceptance is important because the SA algorithm manages to escape from local minimum/maximum values, which is mainly the case in greedy algorithms.

2.5.1 Overview

The goal of the SA algorithm is to bring the system characterized by some energy function $E(s)$ from an initial state to a state with minimum possible energy. The algorithm is iterative. In the basic iteration, the SA algorithm is examining a state s' which is a neighbor state from the current state s . Then, using probabilities, a decision is taken whether to move to that s' or stay in the current s . The overall direction of the algorithm is to move to states with lower energy.

One significant part of the SA algorithm is the probability of making a transition to another state. This probability is defined as a function of $P(e, e', T)$, where $e = E(s)$ and $e' = E(s')$ are the energies of the current state s and the candidate new state s' respectively. T is the global time-varying parameter that represents the system's temperature.

The key feature of the SA algorithm is that it allows the probability P to be non-zero when $e' > e$, meaning that transitions to states of higher energy, which are transitions that do not seem to converge to the minimum solution, are possible. With this feature, getting stuck in local minima is prevented.

The temperature parameter T is reduced from a high initial value (many times it is infinity) to zero or nearly zero during the algorithm, according to some annealing schedule. The rate and step of reduction are specified by the user with the most commonly used schedule being the exponential schedule, where the temperature decreases with a constant factor $a < 1$ at each step. When values of T close to zero are reached, probability P tends to zero when $e' > e$ and has a positive value when $e' < e$, meaning that for small values of T , the algorithm gives advantage more to “downhill” movements (movements with lower energy), than “uphill” movements that were required in the beginning to escape from a local minimum. It is worth mentioning that when T is actually zero, the SA algorithm acts like an equivalent greedy algorithm, which only accepts “downhill” movements.

Another significant part of the algorithm is the choice of the probability function P . Nonetheless, due to the fact that P is always dependent on the temperature factor T , in practice the same probability function is used in every problem, adjusting the annealing schedule accordingly. Originally, the method as formulated in [43] used a transition probability 1 for $e' < e$ (always accept a “downhill” movement) and $e^{\frac{e-e'}{T}}$ for all the other cases. This function originates from the Metropolis-Hastings algorithm [44], which is used to generate a sequence of numbers from the probability distribution of one or more variables.

Finally, one of the most significant parts of the SA algorithm is the choice of the energy function $E(s)$, which is called cost function. The cost function is substantially different for various types of problems, as the involved parameters and their weights vary. If for example we have to examine the behavior of five individual parameters each time we move from a state s to s' , we have to formulate a cost function that gives to each parameter the appropriate weight and combine all of them together in order to give a clear image of the condition of the system in that particular iteration and temperature. The choice of the appropriate cost function is in the responsibility of the SA algorithm user and it must be carefully chosen because small variations can substantially affect the

performance and correctness of the method.

2.5.2 Pseudo-code

To give a more clear view of the method described above, we present the following piece of pseudo-code as a characteristic implementation of the simulated annealing algorithm. The key variables of the pseudo-code are the following:

- *factor*: annealing temperature reduction factor,
- *temps*: number of temperature steps to try,
- *trials*: number of trials at each temperature.

The pseudo-code is the following:

```
initialize temperature
for i = 1 to temps do
  temperature  $\leftarrow$  factor  $\times$  temperature
  for j = 1 to trials do
    try moving to a neighbor solution
    delta  $\leftarrow$  current_cost - trial_cost
    if delta > 0 then
      make the move permanent
    else
      p  $\leftarrow$  random_number  $\in$  [0...1]
      m  $\leftarrow$  exp(delta/temperature)
      if p < m then
        make the move permanent {Metropolis criterion}
      end if
    end if
  return on equilibrium condition
end for
end for
```

The equilibrium condition refers to a predefined condition where we accept that the solution does not get better through time. More details about this condition are given in Section 3.4.

2.6 Conclusion

In this chapter a brief overview of the theoretical background required in order to understand the methodology used in our problem was given. In the next chapter this methodology is presented, based on an amalgamation of the topics discussed so far.

In this chapter we aim to introduce the methodology and steps we followed in order to acquire estimation values for hardware, software and communication between HW/SW, as well as how these values were used as an input to our Simulated Annealing algorithm, which provided the partitioning scenarios.

The chapter is organized as follows. In Section 3.1 a description of our target DVB-H system is given. The hardware implementation is considered in Section 3.2 and power estimation guidelines for hardware, software and communication are provided in Section 3.3. Finally, the actual partitioning methodology using Simulated Annealing is explained.

3.1 DVB-H System Specification

The DVB-H system that we wish to optimize in terms of power dissipation is realized both in software (C++) and in hardware (VHDL). A brief list of requirements for the understanding of the rest of the chapter is the following (more details will be given through the chapter):

1. The reference software application is running on an ARM processor.
2. The hardware implementation resides in an FPGA device.
3. The communication between software and hardware modules is realized through a system bus.
4. The partitioning methodology should be applied in the functional area between the output of the PID filter and the input of the H.264, AAC+ and FLUTE decoding blocks. MPE-FEC and PSI/SI extraction are not examined. This is our field of interest, while the research upon the rest of the fields is outside this thesis' scope.

In order to optimize power dissipation, our plan is to initially partition the system's functionality in distinct parts according to a chosen granularity and estimate the power dissipation for each part both for the software and hardware implementation. These estimations are input vectors to our partitioning methodology, which can calculate efficiently the most appropriate partitioning scenarios for the power optimization problem.

3.1.1 Granularity

Estimations that are used in HW/SW partitioning should happen at a specific and predefined level of granularity [45]. We can define granularity as follows:

Definition 3.1 *Given the behavioral description of a system, the granularity determines how to cut this system into a set B of n pieces $B = \{b_0, b_1, \dots, b_{n-1}\}$ for the purpose of distributing these pieces among the hardware part and the software part of a mixed hardware/software system.*

So, we can say that granularity determines the lower bound of a system's functionality segments.

In our case, examining the desired system's functionality, which is depicted in Figure 2.1, we chose to use the protocol layers of IP Datacast over DVB-H as building blocks. This choice defines the granularity, which belongs to a high-level of abstraction. The choice originates from the system's behavior, which consists from successive fragmentations of an initial MPEG-2 Transport Stream passing from the Data Link, Network, Transport and Application Layers.

Since granularity severely affects the potential communication overhead between blocks residing in different implementations (in our case hardware and software), our choice must have an appropriate reasoning. This reasoning is given by the fact that when communicating between the above-mentioned layers, the majority of the data exchanged between blocks are the payloads of the corresponding formats and very little additional information. The flow of payload data through the whole protocol stack is mandatory, since it is the minimum amount of required information for the next level, so any other choice of granularity would increase the communication overhead by exchanging excess data.

3.1.2 Functional Partitioning

The DVB-H system was partially implemented in hardware (until the output of the PID filter) and fully implemented in reference software. So, in order to have the same elements for comparison, the additional hardware blocks had to be implemented. We briefly describe this procedure in the next section. The requirement was to examine the functional area between the output of the PID filter and the input of the H.264, AAC+ and FLUTE decoding blocks. The functional partition of the envisioned DVB-H system in basic elements, according to the chosen granularity and requirements is presented in Figure 3.1. The blocks in dark color represent the blocks of interest while the blocks in light color represent the corresponding neighboring blocks. Moreover, the lines connecting the blocks represent the corresponding communication.

Figure 3.1 shows that the system has the form of a Directed Acyclic Graph (DAG), representing a call graph $CG = \{V, E\}$, where each vertex represents a building block in hardware or software and each edge represents the communication between two vertexes, and it is assumed to be zero if both of them are implemented in the same way (hardware or software) and non-zero otherwise. As we have stated before, this communication includes the payloads of the corresponding formats, for example the input of the IP filter block consists of MPE section payloads and the output of IP payloads, which becomes input to the UDP filter block and so on.

An MPEG-2 Transport Stream contains a collection of Elementary Streams of different kinds of data. In our system we are examining real-time video and audio and non-real-time file deliverables. If we envisage in our hardware system that each of the

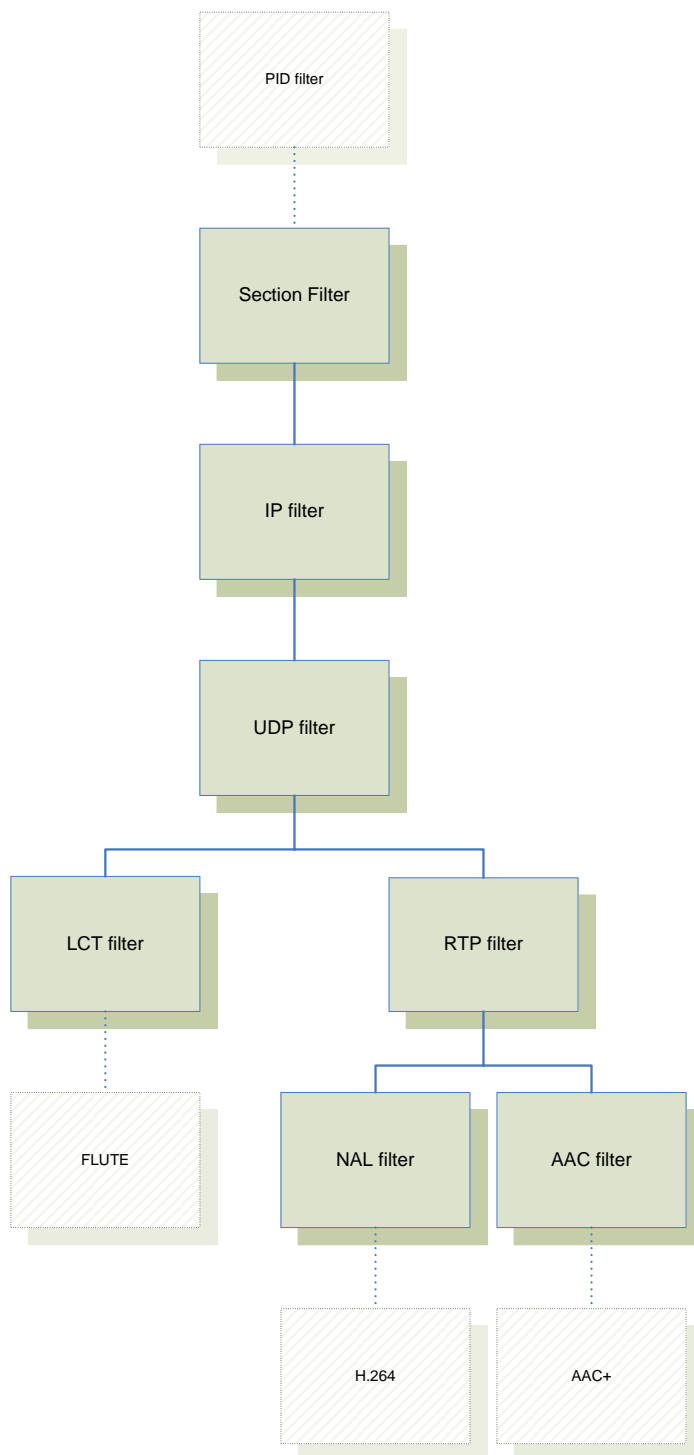


Figure 3.1: DVB-H Functional Partitioning

Table 3.1: IPDC Filtering Parameters

Block	Parameter	Length (bits)
Section filter	table_id	8
IP filter	destination_ipv4_address	32
UDP filter	destination_port	16
RTP filter	payload_type	7
LCT filter	transport_object_identifier	48

above blocks exists only one time in the system, i.e., there is only one Section filter, one IP filter etc., then there are three distinct paths in order to acquire specific content (video, audio or file) at a time and only one of the paths is activated. The three paths are the following:

1. *H.264 extraction*

Section filter \rightarrow IP filter \rightarrow UDP filter \rightarrow RTP filter \rightarrow NAL filter

2. *AAC+ extraction*

Section filter \rightarrow IP filter \rightarrow UDP filter \rightarrow RTP filter \rightarrow AAC filter

3. *FLUTE extraction*

Section filter \rightarrow IP filter \rightarrow UDP filter \rightarrow LCT filter

This distinction is very important, as we will see later in the partitioning methodology, because the partitioning solutions for all three paths must converge until the UDP filter, since it is common in every path. Moreover, convergence has to exist in the RTP filter for the video and audio paths. It is worth mentioning that in software the problem of obligatory activation of at most one path does not exist, since different threads can simultaneously filter many Elementary Streams.

3.1.3 Filtering

In order to acquire data from a specific elementary stream, filtering has to occur in all building blocks in Figure 3.1 (except NAL and AAC filters) using information residing usually in headers of the packets involved. For the part of the DVB-H system we are examining these filtering parameters are shown in Table 3.1.

Without the correct combination of values in the above parameter list, invalid data will be filtered. The filtering mechanism is described in Section 3.2.4.

3.2 Hardware System Realization

3.2.1 Target Hardware

For the realization of the hardware of the additional modules (from Section filter to NAL, AAC, and LCT filter) we used the Altera's Stratix family of FPGAs [11] as target device. Stratix devices are based on a 1.5V, 0.13 μ m, all-layer copper SRAM process,

Table 3.2: Stratix EP1S10 Features

Feature	EP1S10
Logic Elements (LEs) ¹	10,570
M512 RAM Blocks 512-bits + parity	94
M4K RAM Blocks 4-Kbits + parity	60
MegaRAM Blocks 512-Kbits + parity	1
Total RAM Bits	920,448
DSP Blocks	6
Embedded Multipliers	48
PLLs	6
Maximum User I/O Pins	422
<i>Static Power (mW)</i>	187.5

with densities ranging from 10,570 to 114,140 logic elements (LEs) and up to 10 Mbits of RAM. Stratix devices offer up to 28 DSP blocks with up to 224 (9-bit x 9-bit) embedded multipliers optimized for DSP applications that require high data throughput, as it is the case in our system. The EP1S10 model of the Stratix family of FPGAs [46] was used for analysis of the solution space in hardware/software partitioning. The device's features are illustrated in Table 3.2.

3.2.2 Tools

For the design creation of the hardware implementation presented in the previous section, the HDL Designer Series tool (version 2005.2 - build 37) suite from Mentor Graphics [12] was used. This suite provides a complete enterprise-level HDL development platform for ASICs and FPGAs.

For the simulation of the resulting design, the ModelSim SE VHDL 6.1b tool from Mentor Graphics [13] was used. ModelSim provides a comprehensive simulation and debug environment for complex ASIC and FPGA designs.

Finally, for the synthesis of the resulting design, the Quartus II software version 6.0 from Altera Corporation [14] was used. The same tool was used, as we will see in the next section, for the estimation of power dissipation of the hardware implementations.

3.2.3 Level of Abstraction

The programming language used to implement the functionality of the design was VHDL. VHDL is commonly used to describe hardware at many different levels of abstraction. When considering the application of VHDL to FPGA/ASIC design, three levels of abstraction, the algorithm, register transfer level (RTL), and gate level, are identified [47]. Algorithms are unsynthesizable, RTL is the input to synthesis, and gate level is the output from synthesis. The difference between these levels of abstraction is timing oriented.

¹Logic elements are the smallest and the most flexible building blocks of FPGAs. They are composed mainly of 16 bits of memory (16*1 configuration) with a 4 bit address bus and a flip-flop on the output.

- In the algorithm level there are no clock or specific delays, other than the sequential execution of instructions.
- The RTL description has an explicit clock, on which all operations are scheduled, without using specific information for gate delays below cycle level.
- In the gate level, technology-specific delay information for each gate is used.

The gate level of abstraction is a low level for describing behavioral hardware. We are more interested on what the hardware does and not on how it does it, so gate level is not a good choice. On the other hand, the algorithm level is too high, obstructing even the most comprehensive synthesis tools to produce hardware. So, the best choice is the RTL level, which traditionally gives the best results as input to synthesis tools.

3.2.4 Implementation

The whole implementation is clocked at 54MHz, which corresponds to the global transport stream clock, as defined by the system requirements. The input of the section filter, which is the first building block in the functional area we are examining, is an MPEG-2 transport stream, filtered at the PID (`program_id`) after passing through the PID filter. The MPEG-2 transport stream consists of an 8-bit data line clocked at 54MHz.

During the design process, we split the building blocks in Figure 3.1 into sub-blocks for clarity reasons. In all the building blocks we can identify some common functionality blocks:

1. *Header parser*

This functionality block performs parsing of the corresponding headers, identifying the key parameters, such as the payload length or the start/end of the payload, which usually reside in the headers of the corresponding packets. The same building block is responsible for the extraction of the equivalent payload, which is passed to the next building block.

2. *Delay buffer*

This functionality block performs buffering of the transport stream in case the extraction of a large variable from a header is required.

3. *Filter register*

This functionality block is present in all the building blocks except of the NAL filter and the AAC filter. Its purpose is to select a specific elementary stream by setting the corresponding variable that was presented in Table 3.1. All variables in every building block have to be correctly set in order to filter the MPEG-2 transport stream correctly. This implies outputting the appropriate payload only when the transport stream carries the same parameter with the filter register.

We can visualize the above in Figure 3.2.

In [48], it is stated that no buffering is needed on the streaming data on the receiver side when real-time content must be delivered. The terminal has to receive content and

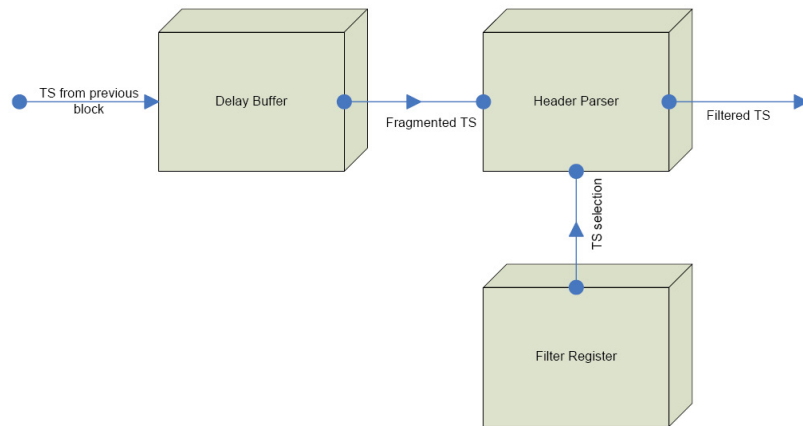


Figure 3.2: Common Functionality Blocks

immediately consume it. However, partitioning scenarios may require that two consecutive blocks be implemented in different ways (hardware or software), which requires communication. Our implementation is aware of this requirement introducing optional buffers in the form of internal Stratix RAMs at the outputs of the hardware building blocks, which can hold corresponding payload data to be sent to the next building block, which is potentially implemented in software. Equivalently, the option of inputting data from a software-implemented block is also realized. Payload data from the software block are put in an internal Stratix RAM and from there they are fed into the next hardware building block. In the case where a hardware building block has to communicate with another hardware building block, this happens simultaneously with the creation of the payloads to be sent. The minimum memory size required in order to efficiently hold payload data is defined by the MPE-section size, which includes all successive headers and payloads. The maximum MPE-section size is 4096 bytes (4080 bytes of datagram + 16 bytes of section overhead), matching with the maximum number of bytes in an MPEG-2 private section [19].

However, a single 4096 byte RAM is not enough in the above situation for the following reason. If two MPE sections arrive one after the other, while their size exceeds the 4096 bytes, the memory manager does not have sufficient time to send the first MPE-section to software before the second one requires a write operation to the internal memory. There are a number of solutions for this problem:

1. *Use of a double buffering memory technique*

This type of memory technique is based on a pair of memories working together. When the one is fully written, the second one is enabled for writing while data start being read from the first one. In our case a pair of 4096 byte single-port RAMs are required.

2. *Use of one dual-port RAM*

This type of memory allows multiple reads or writes to occur in the same time, or nearly the same time, unlike single-ported RAM which only allows one access at a time. In our case, an 8192-byte dual-port RAM is required.

Due to the fact that different clocks may be required for reading and writing (hardware clock runs at 54MHz while the communication bus is considered to be at 200MHz, which is the speed that software can perform reads), and this feature is supported by the dual-port RAM, this type is therefore chosen for the hardware implementation.

3.3 Power Estimation

3.3.1 Hardware

For the estimation of the power dissipation of the building blocks of our system depicted in Figure 3.1, the Quartus II PowerPlay Power Analyzer [49] was used. The factors affecting the power consumption and addressed from the Power Analyzer are the following:

1. *Device selection*

The Power Analyzer calculates the power consumption according to the target device. Target devices have different characteristics, such as process technology, supply voltage, electrical design, and device architecture.

2. *Number, type and loading of I/O pins*

Output pins drive off-chip components, which creates a high-load capacitance and consumes more power per signal transition.

3. *Number and type of Logic Elements, Multiplier Elements, and RAM blocks*

A design with more such circuit elements usually consumes more power than a design with fewer elements. The working mode of the element sometimes affects its contribution to the total static power consumption.

4. *Number and type of Global Signals*

Global signals are spread all over a design, consuming in that way significant amounts of dynamic power. One typical example of such signal is the global clock, which spans through the entire design.

5. *Signal Activities*

The transitions of internal signals play one of the most important roles contributing to the system's power dissipation. Two statistical factors affect signal activities, the toggle rate and the static probability. Toggle rate is defined as the number of times the signal makes a transition from 1 to 0 or 0 to 1 per second, while static probability is defined as the fraction of time the signal is in high state, during the device's operation. The toggle rate affects the frequency of charging

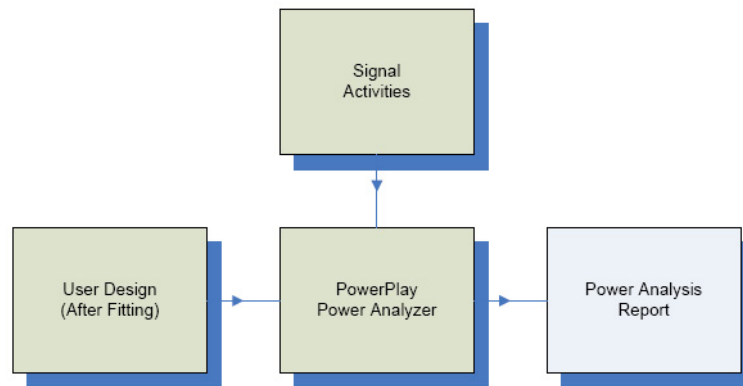


Figure 3.3: PowerPlay Power Analyzer High-Level Flow

and discharging of the capacitive load, and the dynamic power consumption is increased linearly as toggle rates increase.

The high-level flow of the Power Analyzer is depicted in Figure 3.3.

Power Analyzer requires synthesis of the design targeting a specific device and a collective signal activity file to be used for toggle rate extraction. As we have indicated in the previous section, synthesis is realized using the Quartus II software. Using the synthesis tool of Quartus II we also extract the hardware area that each building block requires in order to be synthesized, which will be used as an input to our partitioning methodology (see Section 3.4).

On the other hand, the most efficient way of deriving the signal activities is through simulation results. Simulation is performed using the ModelSim tool, which is considered a third-party simulator for the Quartus II. In this case, a script is generated from Quartus II, called Value Change Dumb file script, which instructs ModelSim to generate a Value Change Dumb (.vcd) file containing encoded waveforms while it is simulating the design. Through this procedure, the Power Analyzer derives toggle rates and static probabilities for the majority of signals. There are a number of signals corresponding to internal nodes that cannot be extracted from the ModelSim simulation. For these signals a default toggle rate is used to approximate the signal's activity, which is set to 12.5% by the Quartus II tool.

In order to evaluate the power dissipation for each individual block in Figure 3.1, a reference design consisting of all the modules properly interconnected was created. Then, applying separate MPEG-2 Transport Streams that contained data for H.264 video, AAC+ audio or FLUTE as an input, the power dissipation for each block was

estimated by the Quartus II, through hierarchical estimations that are supported from the Power Analyzer. These MPEG-2 Transport Streams were taken directly from a file that was generated from a DVB-H encapsulator.

3.3.2 Software

In order to estimate the power dissipated by the software executed on the ARM processor, we use Sim-Panalyzer [15], an augmentation to the SimpleScalar performance simulator [16]. ARM binaries were produced using an ARM-Linux cross-compiler.

3.3.2.1 SimpleScalar

The SimpleScalar tool set is a system software infrastructure used for program performance analysis, detailed microarchitectural modeling, and hardware-software co-verification. Through SimpleScalar, applications can be built and simulated on a range of processors. The type of simulator varies from fast functional simulators to detailed processor model simulators that are able to simulate caches, branch predictors, and many other features of modern processors. Since SimpleScalar can emulate the ARM instruction set, and because its reliability is in very high levels (in 2000 more than one third of all papers published in top computer architecture conferences used the SimpleScalar tools to evaluate their designs), it appears ideal for emulating the ARM processor in our case.

3.3.2.2 Sim-Panalyzer

The Sim-Panalyzer tool on the other hand is an infrastructure for microarchitectural power simulation. It is positioned above the “sim-outorder” component of the SimpleScalar simulator. The Sim-Panalyzer program contains components that model specific parts of the ARM processor. Sim-Panalyzer focuses efficiently on basic microarchitectural blocks and provides power information over a wide range of power dissipation sources, such as caches, clock trees, external I/O, on-chip memories and datapath and execution blocks. For microarchitectural blocks, the basic method to calculate the power dissipation is by multiplying the appropriate switching capacitance by the number of microarchitectural accesses. For external I/O accesses, a transaction model counts the I/O pin switches in a cycle accurate way. Moreover, support for more sophisticated and accurate power models is provided through libraries, containing basic building blocks for the embedded logic simulator and ability to extract switching capacitance for CMOS gates. The logic simulator accumulates the switching capacitance of internal nodes into a switching capacitance estimation of each functional block’s node.

3.3.2.3 ARM-Linux cross-compiler

Since the target architecture for our experiments is the ARM architecture, the inputs to the Sim-Panalyzer tool for program emulation and power dissipation calculation must be ARM binaries. In this case, a cross-compiler kit targeting the ARM should be built on a Linux platform in order to acquire an ARM executable from C++ code. A cross

compiler is a compiler capable of creating executable code for a platform other than the one on which the cross compiler runs, so it is suitable for our problem.

Gcc [50] is a free cross-compiler that supports a large number of platforms and languages, among them the ARM platform and the C++ language. Together with gcc, glibc [51], which is the GNU C library, mandatory for system calls and other basic facilities, is needed. Gcc also relies upon binutils [52], a collection of binary tools, with the most important of them being ld - the GNU linker and as - the GNU assembler. Finally, linux kernel headers [53], which are header files for the Linux kernel that define structures and constants that are needed for building most standard programs, are necessary.

As we mentioned before, building the tool-chain for embedded targets is in general a difficult procedure. New releases of the used utilities are not always supported and many times the gcc team relies on users to test upcoming releases. In [54] though, this procedure is simplified by becoming automated. The crosstool [54] is a collection of scripts to build and test several versions of gcc and glibc for most architectures supported by glibc. The resulting toolchain was arm-unknown-linux-gnu, constituting of the combination of the following versions of modules:

- Gcc-3.4.2,
- Glibc-2.3.3,
- Binutils-2.15,
- Linux-2.6.8.

3.3.2.4 Estimation Procedure

After building the cross compiler, the command line argument required in order to compile a C++ application (for example hello.cpp) for the ARM is the following:

```
> arm-unknown-linux-gnu-g++ -static -o hello hello.cpp
```

After building the Sim-Panalyzer tool, the following command is outputting the power dissipation report of the hello.cpp application:

```
> sim-panalyzer -config hello.cfg hello
```

The `-config` defines the name of the configuration file that contains architecture specific information for the ARM processor, such as the operating frequency, supply voltage etc. This configuration file is generated from a script provided by the Sim-Panalyzer tool, that parses a command file. The command file that we use in order to generate the configuration file is the default command file provided by Sim-Panalyzer. We mention a portion of the parameters in the command file that are important for our estimations in Table 3.3.

Table 3.3: Sim-Panalyzer Command File Parameters

Global (core) Parameters	Value
Supply Voltage	1.8V
Frequency	200MHz
Address/Data I/O Parameters	Value
I/O Voltage	3.3V
Frequency	200MHz
Microstrip length	10mm

3.3.3 Communication between HW-SW

In our methodology, we take into account the communication cost between two building blocks only when they reside in different implementations. We, therefore, assume the communication between hardware-hardware or software-software negligible. In order to estimate the power dissipation for the transfer of a specific amount of bytes from hardware to software and vice versa, we assume that power is dissipated only in the bus connecting the processor (in our case, ARM) and the reconfigurable logic (in our case, Stratix FPGA). In this line of reasoning, we use the models presented in [55] for the power dissipated in the interconnect between data path and external memory in an embedded system. In cases where software is sending data to hardware or hardware is sending data to software, the receiver of data can be seen as a memory component for the transmitter, so this model provides a good approximation of the actual power dissipation.

The power dissipated by a specific bus b , P_{bus} in mW is defined [55] as:

$$P_{bus} = W_{bus} \cdot a_{toggle} \cdot f_{access} \cdot (C_{driver} + C_{load}) \cdot V_{dd}^2 \cdot 0.001 \quad (3.1)$$

where:

W_{bus} is the width of bus b in bits,

a_{toggle} is the toggle rate of data in the bus,

f_{access} is the operating frequency of the bus in MHz,

C_{driver} is the capacitance of the buffer that drives bus b , in pF,

C_{load} is the total load capacitance of bus b , in pF,

V_{dd} is the bus power supply, in Volts and,

0.001 is the factor included to obtain the power in mW.

In the remaining of the section we estimate the power dissipation value for each communication between consecutive blocks in Figure 3.1 when residing in different implementations.

Table 3.4: Signal Toggle Rates

Block	Sum of Hamming Distances	a_{toggle}
Section filter	296,430	0.1813 (18.13%)
IP filter	268,999	0.1646 (16.46%)
UDP filter (NAL)	185,069	0.1132 (11.32%)
UDP filter (AAC)	81,917	0.0501 (5.01%)
UDP filter (LCT)	178,556	0.1092 (10.92%)
RTP filter (NAL)	172,816	0.1057 (10.57%)
RTP filter (AAC)	70,168	0.0429 (4.29%)
NAL filter	170,491	0.1043 (10.43%)
AAC filter	68,791	0.0421 (4.21%)
LCT filter	124,693	0.0763 (7.63%)

We assume that the width of the bus is 8 bits, so $W_{bus} = 8$. The bus power supply voltage and operating frequency are the same used in the software estimation, so $V_{dd} = 3.3V$ and $f_{access} = 200MHz$.

We evaluate the remaining parameters of Equation (3.1) as follows:

- The toggle rate a_{toggle} is defined [56] as

$$a_{toggle} = \frac{\sum_{t=0}^{L-1} H(B^{(t)}, B^{(t+1)})}{L - 1} \quad (3.2)$$

where:

$B^{(t)}$ is the encoded word on the bus at time t ,

$B^{(t+1)}$ is the encoded word on the bus at time $(t+1)$,

L is the total length of the generated stream and

$H(B^{(t)}, B^{(t+1)})$ is the Hamming distance between the encoded word on the bus at time t and $(t+1)$.

In order to acquire the parameter a_{toggle} for every communication between consecutive blocks, we run a process in VHDL calculating the sum of Hamming distances between consecutive words in the output stream for a specific amount of time. After that, we divide this value with the total length of the generated stream during this time and we present the results in Table 3.4. Note that the rates correspond to the output of the mentioned building block (i.e., the toggle rate for IP corresponds to the IP to UDP communication and so on) and the total length $L = 1,634,276$ bytes in every case. Moreover, for the UDP and the RTP filter there are three different directions according to the content (NAL, AAC or LCT) and the toggle rates change accordingly.

We observe from Table 3.4 that the sum of Hamming distances is reduced as we move lower into the hierarchy, which can be explained from the fact that less data are present at the output due to header removals.

- The load capacitance C_{load} is estimated using the following formula [55]:

$$C_{load} = L_{on-chip} \cdot C_{on-chip} + C_{pin} + L_{off-chip} \cdot C_{off-chip} \quad (3.3)$$

where:

$L_{on-chip}$ is the on-chip length of the bus in mm,

$C_{on-chip}$ is the capacitance per mm on-chip interconnect (pF/mm),

$L_{off-chip}$ is the off-chip length of the bus in mm,

$C_{off-chip}$ is the capacitance per mm off-chip interconnect (pF/mm), and

C_{pin} is the capacitance of an I/O pin of the ASIC.

For the on-chip and off-chip lengths we use the same value as in the previous section, when we were defining the parameters for the ARM processor, thus

$$L_{on-chip} = L_{off-chip} = 10mm. \quad (3.4)$$

For 0.13 μ m technology, the on-chip interconnect capacitance is defined as:

$$C_{on-chip} = 0.24pF/mm. \quad (3.5)$$

The off-chip capacitance of a printed wiring board is crudely approximated as:

$$C_{off-chip} = 0.1pF/mm. \quad (3.6)$$

Typical values for the ASIC I/O pin capacitance are between 1 and 10 pF. We approximate:

$$C_{pin} = 5pF. \quad (3.7)$$

Using the above values, we calculate C_{load} from Equation (3.3):

$$C_{load} = 8.4pF. \quad (3.8)$$

- The bus driver capacitance is nearly 30% of its total load [55], so:

$$C_{driver} = 0.3 \cdot C_{load} = 2.52pF. \quad (3.9)$$

Using the values from Equations (3.8) and (3.9), together with the bus width and operating frequency values, Equation (3.1) becomes:

$$P_{bus} = 190.27 \cdot a_{toggle}. \quad (3.10)$$

Using Equation (3.10) and Table 3.4, the per block power dissipation estimated are presented in Table 3.5.

Table 3.5: HW/SW Communication Power

Block	a_{toggle}	HW/SW Communication Power (mW)
Section filter	18.13%	34.49
IP filter	16.46%	31.31
UDP filter (NAL)	11.32%	21.54
UDP filter (AAC)	5.01%	9.53
UDP filter (LCT)	10.92%	20.77
RTP filter (NAL)	10.57%	20.11
RTP filter (AAC)	4.29%	8.16
NAL filter	10.43%	19.84
AAC filter	4.21%	8.01
LCT filter	7.63%	14.51

We must note that the last three values (NAL, AAC, and LCT) are not used as an input to our methodology, since we do not examine the architecture of the H.264 decoder, AAC+ decoder, and FLUTE implementation, which are the corresponding next blocks, but they are presented for potential future use.

3.4 Partitioning using Simulated Annealing

Referring to Section 2.4 our primary goal is to find an area-satisfying partition $Par_1 = \{H, S\}$ so that power consumption $P = min.$ or to find a power-satisfying partition $Par_2 = \{H, S\}$ so that the hardware area $A = min.$ In the remaining of the section we will discuss the first objective, noting that for the second objective we act in an equivalent way. In Section 2.5 we presented the fundamentals for simulated annealing, but in order to adapt the algorithm to our problem we have to make some considerations.

3.4.1 Algorithm Considerations

First of all, the metrics for the cost function in our system are the hardware area in Logic Elements and the power dissipation in mW.

In HW/SW partitioning a random perturbation of a configuration implies a move of a single vertex (see Figure 3.1) from HW to SW or from SW to HW. This move alters the values in the edges between the neighboring vertexes, and the communication costs change together with the value of the moved vertex. We cannot really assume that a move of a vertex from SW to HW simply reduces the system's power consumption, because the resulting communication cost overhead may be higher than the power consumption gain (generally, hardware dissipates less energy than software). Nonetheless, this move definitely increases hardware area, as it is an independent factor. The effects of a move of a vertex from HW to SW can be discussed in a similar way. Finally, we consider a user-defined threshold value for hardware area C_{HA} , which is used as a constraint for our algorithm.

3.4.2 Cost Function

In the main iteration of the SA algorithm, the cost function corresponding to a potential move is evaluated. Simulated annealing inherently uses randomization to overcome local minima and moves that do not improve the optimization objective are accepted with some probability. There are several different ways to find the appropriate cost function. In [57] a multiobjective problem of hardware/software partitioning is addressed. Parameters are execution time and hardware area. The cost function in [57] is formed by dynamically weighting the components after normalization, with a number representing the distance from the desired solution region. Penalization is also used when a move leads to a partitioning that crosses the constraint boundaries. Typical cost functions, made out of a combination of normalized metrics are presented in [39], [58] and [40]. We decide to use a typical cost function together with some extra features introduced in [57] that we describe later. Our choice is based on the fact that the vertexes in our problem are few and we do not need to emphasize on the execution time of the algorithm, but on its reliability.

The hardware area and power consumption values in our cost function have to be normalized since they refer to different units. Normalization of hardware area is achieved by dividing the hardware area with the hardware area constraint. For the power consumption, normalization is achieved by dividing the power consumption with power consumption constraint. Instead of using the simplest scheme of just adding two normalized values to evaluate the cost function, we use a modified version of [57], where there is a penalty for moves that are crossing the constraint boundaries. Penalties are proportional to the distance from the boundary constraint. Moves causing boundary violations in both power consumption and hardware area are severely penalized. Moreover, since we aim to optimize power consumption, a 2x weight to the power consumption factor is given.

Based on the above discussion, our cost function is algorithmically described as:

```

cost_function  $\leftarrow$   $2 \times (cur\_power)/(power\_boundary) + (cur\_area)/(area\_boundary)$ 
if move causes only power boundary violation then
    cost_function  $\leftarrow$  (cost_function)  $\times$  [(cur_power)/(power_boundary)]
else if move causes only area boundary violation then
    cost_function  $\leftarrow$  (cost_function)  $\times$  [(cur_area)/(area_boundary)]
else if move causes area boundary AND power boundary violation then
    cost_function  $\leftarrow$  (cost_function)  $\times$  100
end if

```

3.4.3 Other SA Parameters

Following the cost function evaluation in significance for the efficiency of Simulated Annealing are the initial temperature, the cooling schedule, the number of iterations, and the global equilibrium.

- *Initial Temperature T*

The initial temperature is always chosen to have a relatively large value in order to allow the algorithm to escape local-minima more easily in the beginning. A fixed

initial temperature of $T = 500$ is used in our implementation.

- *Cooling Schedule*

We remind that the cooling schedule is responsible for decreasing the temperature at each step. The commonly used geometric schedule is used in our implementation, where $T_{new} = a \cdot T$, where a is a constant between 0.9 and 0.99. Due to the simplicity of our problem, we chose $a = 0.9$ which results in a fast cooling process.

- *Number of Iterations*

We choose a number of iterations at the current temperature using a simple constant multiple of the partitioning objects, as in [59]. Again, for simplicity reasons we choose a small constant multiple like 3.

- *Global equilibrium*

As a termination criterion we choose a commonly used method, which dictates that the algorithm must be stopped after a small number of constant temperature iterations without improvement in the quality of the solution [60]. We choose this number to be as much as the partitioning objects, so $\frac{1}{3}$ of the maximum iterations for a constant temperature according to the previous paragraph.

3.5 Conclusion

After providing all the necessary knowledge regarding the means for acquiring the power estimation inputs to the Simulated Annealing algorithm, as well as the algorithm itself, we can now move to the next chapter where we present our experimental results and the corresponding partitioning scenarios.

In this chapter, the results based on the so far discussed theory and methodology of Chapters 2 and 3 are introduced. Initially in Section 4.1, the power dissipation values measured for the hardware and software implementation are presented in detail. Then in Section 4.2, an analysis of the solution space is given and the partitioning scenarios according to our Simulated Annealing algorithm are discussed.

4.1 Experimental Results

As we have noted in Section 3.1.2, there are three paths in the partition of Figure 3.1, corresponding to specific content extraction (video, audio, files). For that reason, three experiments for every implementation (hardware and software) took place, setting the correct parameters given in Table 3.1 in order to acquire the relevant results. In Sections 4.1.1 and 4.1.2, the power dissipation values for hardware and software implementations are given, respectively. The power dissipation values for HW/SW communication have already been presented in Table 3.5.

4.1.1 Hardware

Initially, we present the hardware area required in the FPGA for the implementation of each of the blocks in Figure 3.1. These values originate from the synthesis of the implemented design and are shown in Table 4.1.

Block	Hardware Area (LEs)
Section Filter	571
IP Filter	532
UDP Filter	127
RTP Filter	165
NAL Filter	232
AAC Filter	51
LCT Filter	25

Table 4.1: Hardware design area values

In Tables 4.2 to 4.4, the power dissipation values of the H.264 extraction, the AAC+ extraction, and the FLUTE extraction path are presented, respectively. The I/O power corresponds to the power consumed due to switching activity and the Logic Cell power corresponds to the internal power dissipation of the logic gates, as explained in Section 2.2. The average toggle rates of the signals related to Logic cells and I/O are also provided in the Tables.

We note again that the static thermal power dissipation for the EP1S10 Stratix FPGA device is *187.5 mW*.

Block	Power of I/O (mW)	Power of Logic Cells (mW)	<i>Total Power (mW)</i>
Section Filter	4.69	4.60	<i>9.29</i>
IP Filter	5.00	3.63	<i>8.63</i>
UDP Filter	7.37	0.82	<i>8.19</i>
RTP Filter	7.40	1.10	<i>8.50</i>
NAL Filter	4.54	1.66	<i>6.20</i>

Average toggle rate for Logic Cells: 6.572 Mtransitions/sec

Average toggle rate for I/O: 2.635 Mtransitions/sec

Simulation duration: 100 ms

Table 4.2: Power dissipation values for HW implementation (H.264 path)

Block	Power of I/O (mW)	Power of Logic Cells (mW)	<i>Total Power (mW)</i>
Section Filter	4.69	4.60	<i>9.29</i>
IP Filter	5.00	3.63	<i>8.63</i>
UDP Filter	7.23	0.79	<i>8.02</i>
RTP Filter	7.09	1.02	<i>8.11</i>
AAC Filter	4.30	1.37	<i>5.67</i>

Average toggle rate for Logic Cells: 6.285 Mtransitions/sec

Average toggle rate for I/O: 2.571 Mtransitions/sec

Simulation duration: 100 ms

Table 4.3: Power dissipation values for HW implementation (AAC+ path)

Block	Power of I/O (mW)	Power of Logic Cells (mW)	<i>Total Power (mW)</i>
Section Filter	4.45	4.15	<i>8.60</i>
IP Filter	4.32	3.31	<i>7.63</i>
UDP Filter	6.74	0.98	<i>7.72</i>
LCT Filter	3.44	1.18	<i>4.62</i>

Average toggle rate for Logic Cells: 6.579 Mtransitions/sec

Average toggle rate for I/O: 5.382 Mtransitions/sec

Simulation duration: 300 ms

Table 4.4: Power dissipation values for HW implementation (FLUTE path)

We next introduce in Table 4.5 the total power dissipation and hardware area characteristics that were derived from Tables 4.1 to 4.4 for each of the three paths, when an all-hardware solution is realized. These values are important since they are used as constraint parameters in the partitioning methodology, as we will see in Section 4.2. Note again that for the total power dissipation, the static power of the Stratix (187.5 mW) is also included.

Path	Total Hardware Area (LEs)	Total Power Dissipation (mW)
H.264 (video)	1627	228.31
AAC+ (audio)	1446	227.22
FLUTE (files)	1255	216.07

Table 4.5: Total Power Dissipation and Hardware Area values for all-hardware implementation

4.1.2 Software

In this section, the power dissipation values for the software implementation are presented. These values were derived using the estimation procedure described in Section 3.3.2. In order to approximate the correct power dissipation values, we applied the estimation methodology to each individual block, taking into account the reading from and writing to a memory buffer of the relevant transport streams. In this line of reasoning, we applied appropriate modifications to the reference software implementation.

Block	Total Power (mW)
Section Filter	83.00
IP Filter	79.20
UDP Filter	77.40
RTP Filter	71.70
NAL Filter	60.60

Table 4.6: Power dissipation values for SW implementation (H.264 path)

Block	Total Power (mW)
Section Filter	83.00
IP Filter	79.20
UDP Filter	62.50
RTP Filter	59.20
AAC Filter	58.90

Table 4.7: Power dissipation values for SW implementation (AAC+ path)

Block	Total Power (mW)
Section Filter	78.40
IP Filter	70.90
UDP Filter	68.00
LCT Filter	64.10

Table 4.8: Power dissipation values for SW implementation (FLUTE path)

In Tables 4.6 to 4.8, the power dissipation values of the H.264 extraction, the AAC+ extraction, and the FLUTE extraction path are given, respectively.

Furthermore, in Table 4.9 the total power dissipation values that were derived from Tables 4.6 to 4.8 for each of the three paths is given, when an all-software solution is realized. These values are again used as constraint parameters in the partitioning methodology, as presented in Section 4.2. The static power of the ARM processor, as given by the Sim-Panalyzer tool is 18.4 mW and is also included in the values of Table 4.9.

Path	Total Power Dissipation (mW)
H.264 (video)	390.30
AAC+ (audio)	361.20
FLUTE (files)	299.80

Table 4.9: Total Power Dissipation values for all-software implementation

4.2 Analysis of the Solution Space

We intend to find the best HW/SW partitioning scenarios, so that the power reduction over an all-software solution is maximum. Initially, we acquire the power reductions when we move from an all-software solution to an all-hardware solution. This can be visualized in Table 4.10 using the data of Tables 4.5 and 4.9. The hardware area in this case for each path is maximum and the power reduction is the maximum that can be achieved.

Path	Hardware Area (LEs)	Power reduction (mW)	Power reduction (%)
H.264 (video)	1627	161.99	41.50
AAC+ (audio)	1446	133.98	37.09
FLUTE (files)	1255	83.73	27.93

Table 4.10: Power Dissipation reductions of an all-hardware solution over an all-SW solution

In an all-hardware implementation, the total amount of Logic Elements needed is derived from the summation of the hardware areas of all the building blocks, which is 1703 LEs . The available LEs of the EP1S10 Stratix FPGA (10,570) surpass the total hardware area required to have an all-hardware solution. Nevertheless, we examine the efficiency of our algorithm as if there were hardware area limitations. Hence, we aim to investigate the solution space for different power dissipation and hardware area constraints.

In order to have power dissipation reduction over an all-software solution, we explore the partitioning possibilities that do not surpass the all-software power dissipation presented in Table 4.9. However, the fixed power dissipation cost that is required in order to integrate the EP1S10 Stratix FPGA to our system is the FPGA's static power dissipation (187.5 mW). So, taking into account this lower limit, the maximum allowed power dissipations for our partitioning scenarios are derived from the difference between the power dissipations of an all-software solution and the FPGA's static power. These values are depicted in Table 4.11.

Path	Max. allowed Power Dissipation (mW)
H.264 (video)	202.8
AAC+ (audio)	173.7
FLUTE (files)	112.3

Table 4.11: Maximum allowed Power Dissipations in order to have Power Reduction

The values in Table 4.11, the hardware power estimation values in Table 4.5, the software power estimation values in Table 4.9, and the HW/SW communication values in Table 3.5 become inputs to our Simulated Annealing algorithm, which derives the best partitioning scenario for each individual path. Afterwards, we explore the possibilities of further power dissipation reduction by decreasing the limits of Table 4.11. The hardware area constraints are set to the values of Table 4.1, since our algorithm will search for the solutions that are as below these constraints as possible. Tables 4.12, 4.13, and 4.14 present the outputs of our Simulated Annealing algorithm for the corresponding paths. The fields of these Tables are described as follows.

1. The first row represents the power dissipation boundary for our SA algorithm (without the FPGA's static power dissipation component). The numbers are chosen in order to correspond to specific reductions over the maximum allowed power dissipation values of Table 4.11, namely 10%, 20%, 30% and maximum reduction (minimum power dissipation boundary). The SA algorithm will search for solutions below that boundary that take up the least amount of hardware area.
2. The second row represents the actual power dissipation of the partition chosen by the SA algorithm. This value is always less than that of the first row and it does not include the FPGA's static power dissipation (187.5 mW).
3. The third row represent the hardware area taken up by chosen partition. It consists of the hardware areas of the hardware implemented building blocks.
4. The forth row introduces the total power reduction over the values of Table 4.9, which represent the power dissipation when an all-software solution is realized.
5. The fifth row represents the total power reduction percentage over the values of Table 4.9.
6. The sixth row gives a pointer to the corresponding partitioning scenario in Table 4.15.

In Table 4.14, we observe that the algorithm cannot provide any better results than the all-hardware solution as we reduce the power dissipation boundary after the first value. This situation is created because only two partitions can give power reduction, while all the others offer an increase of power dissipation over an all-software solution, due to low power dissipation boundary and communication costs.

The partitioning scenarios for all the paths, according to Tables 4.12, 4.13, and 4.14 are depicted in Table 4.15.

Power Dissipation Boundary (mW)	182.6	162.2	141.9	minimum
Actual Power Dissipation (mW)	177.18	149.01	115.32	40.81
Hardware Area (LEs)	1095	1056	1395	1627
Power reduction (mW)	25.62	53.79	87.48	161.99
Power reduction (%)	6.56	13.78	22.41	41.50
Partition (H.264 - Table 4.15)	a	b	c	d

Table 4.12: Results of the SA algorithm for the H.264 path

Power Dissipation Boundary (mW)	156.3	138.9	121.6	minimum
Actual Power Dissipation (mW)	147.92	135.04	101.11	39.72
Hardware Area (LEs)	875	1319	1395	1446
Power reduction (mW)	25.78	38.66	72.59	133.98
Power reduction (%)	7.14	10.70	20.09	37.09
Partition (AAC+ - Table 4.15)	a	b	c	d

Table 4.13: Results of the SA algorithm for the AAC+ path

Power Dissipation Boundary (mW)	110.1	89.84	78.61	minimum
Actual Power Dissipation (mW)	108.82	28.57	28.57	28.57
Hardware Area (LEs)	1230	1255	1255	1255
Power reduction (mW)	3.48	83.73	83.73	83.73
Power reduction (%)	1.16	27.93	27.93	27.93
Partition (FLUTE - Table 4.15)	a	b	b	b

Table 4.14: Results of the SA algorithm for the FLUTE path

Path	Partition	SF	IP	UDP	RTP	NAL	AAC	LCT
H.264 (video)	a	HW	SW	HW	HW	HW		
	b	SW	HW	HW	HW	HW		
	c	HW	HW	HW	HW	SW		
	d	HW	HW	HW	HW	HW		
AAC+ (audio)	a	SW	HW	HW	HW		HW	
	b	HW	HW	SW	HW		HW	
	c	HW	HW	HW	HW		SW	
	d	HW	HW	HW	HW		HW	
FLUTE (files)	a	HW	HW	HW				SW
	b	HW	HW	HW				HW

Table 4.15: Partitioning Scenarios

As we have stated in Section 3.1.2, the partitions for every path must converge in implementation for the Section Filter, the IP Filter, and the UDP Filter. The only partitioning scenario that offers that, except the one of an all-hardware solution that we call PS_0 , is the one consisting of H.264(c), AAC+(c) and FLUTE(a) or FLUTE(b)

in Table 4.15. We call this partition set PS_1 in the case of FLUTE(b) and PS_2 in case of FLUTE(a). However, we observe that the H.264 and AAC+ paths, which are dominating regarding power consumption, converge as well in H.264(b), AAC+(a). We investigate the power reductions also for the last case, even though the {SW, HW, HW, HW} partition of FLUTE (consuming 131.91 mW and taking up 684 LEs) is not included in the results obtained from the SA algorithm. We call this partition set PS_3 .

Table 4.16 presents a comparison between the aforementioned partition sets. Note that the negative sign of the power reduction for FLUTE in PS_3 means that there is an increase in power dissipation in comparison with an all-software solution. The hardware area values correspond to the hardware implemented building blocks of all three paths. Note that (v), (a) and (f) correspond to the H.264, AAC+, and FLUTE paths, respectively.

Partition Set	Hardware Area (LEs)	Power Reductions (%)	Avg Power Reduction (%)
PS_0	1703	41.50 (v) 37.09 (a) 27.93 (f)	35.51
PS_1	1420	22.41 (v) 20.09 (a) 27.93 (f)	23.48
PS_2	1395	22.41 (v) 20.09 (a) 1.16 (f)	14.55
PS_3	1132	13.78 (v) 7.14 (a) -6.54 (f)	4.79

Table 4.16: Comparison of Partitioning Scenarios

After examining Table 4.16, we come to the following conclusions:

1. The solution that offers the largest average power reduction, but also takes up the most hardware area is the all-hardware solution (PS_0)
2. If there is a hardware area constraint between 1420 and 1702 LEs, the best partitioning scenario is PS_1 .
3. Partitioning scenario PS_2 is dominated by PS_1 , since it offers considerably less power reduction in exchange of a very small hardware area reduction in comparison to PS_1 . This difference in power reduction originates from the fact that when the LCT block is implemented in software and the UDP block in hardware, the communication cost is added to the overall power dissipation.
4. Partitioning scenario PS_3 does not seem convenient, since we have an increase of power dissipation in the FLUTE path in comparison to an all-software solution, and the power reductions for the rest of the paths are poor in comparison to the other solutions.

5. However, the selection of the appropriate partition still highly depends on the hardware area constraints, meaning that even partitioning scenario PS_3 is considered appropriate when there is a hardware area constraint between 1132 and 1420 LEs.

In the remaining of the section, we verify the results obtained by the Simulated Annealing algorithm and we examine its efficiency.

The Simulated Annealing algorithm searches the solution space for the partition that satisfies best the selected criteria. This search, however, is not exhaustive and the final selection depends on a predefined stopping condition, as explained in Section 3.4. For this reason, and because the amount of our system's building blocks is small (meaning the exhaustive search will not take much time to complete), we verify the correctness of the algorithm by running a program that calculates all the possible partitions and compares the results with the ones obtained from Simulated Annealing.

The amount of possible partitioning scenarios p depends on the number of building blocks n under investigation from $p = 2^n$. The problem of finding the appropriate partition is categorized as NP-complete, since it can be solved in exponential time. Therefore, there are 32 possible partitions for the H.264 and AAC+ paths and 16 for the FLUTE path. In Figures 4.1, 4.2, and 4.3, the {hardware area, power consumption} pairs for every partition are depicted for the H.264, AAC+, and FLUTE path, respectively. The portion of the graphs that corresponds to pairs giving power reductions is highlighted and the proposed partitions from the Simulated Annealing algorithm are also presented. The letters below the proposed pairs correspond to the partitions of Table 4.15.

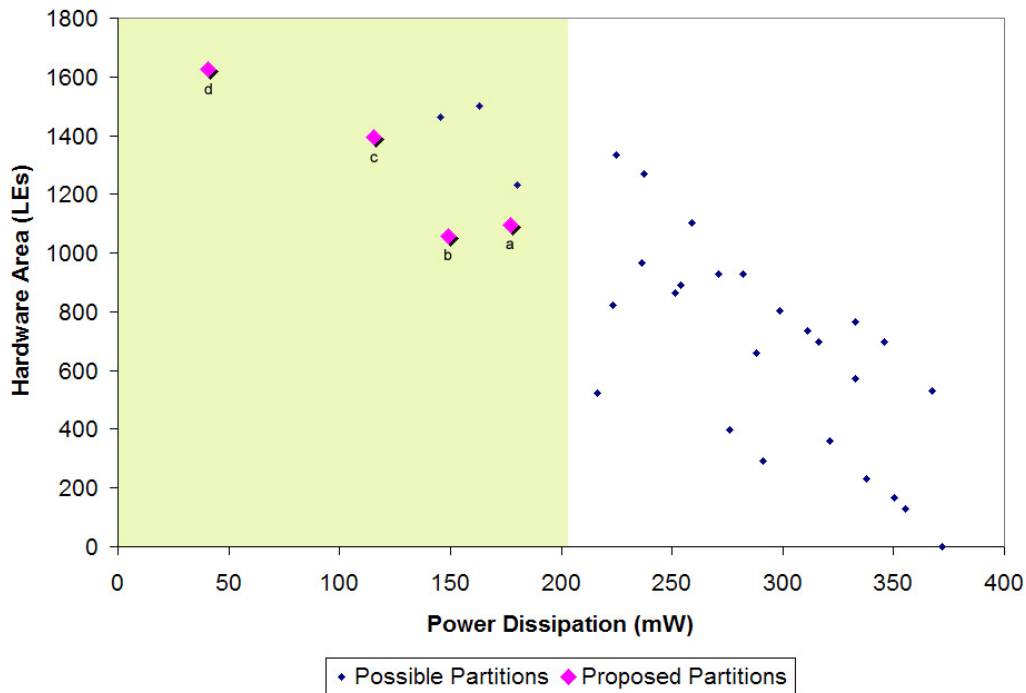


Figure 4.1: Possible and Proposed Partitions for the H.264 path

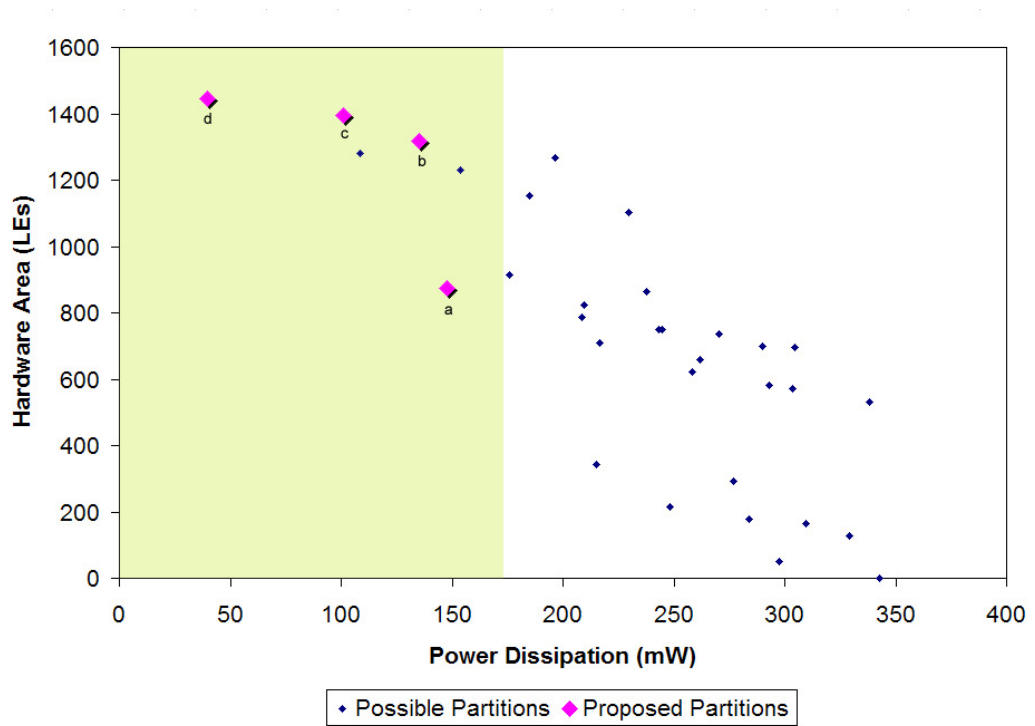


Figure 4.2: Possible and Proposed Partitions for the AAC+ path

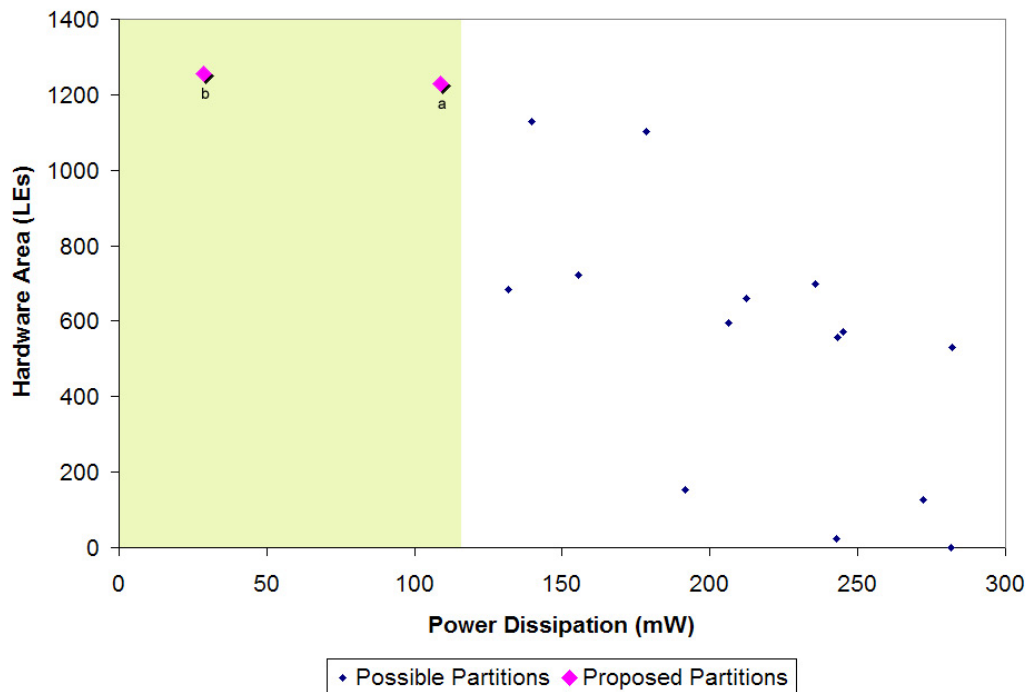


Figure 4.3: Possible and Proposed Partitions for the FLUTE path

We observe that in every graph the distribution of points starts from high hardware area - low power dissipation values and ends to low hardware area - high power dissipation values. Nevertheless, the distributions include several local minima, which makes the application of a traditional method for finding a minimum inappropriate. In this line of reasoning, a stochastic method like our Simulated Annealing that manages to escape from local minima is necessary.

In Figure 4.1 we can identify the role of the Simulated Annealing cost function in the choice of the best partition. The possible pair of values above point (a) is not chosen, since it provides much higher hardware area value for approximately the same power dissipation value as point (a). A similar situation occurs for point (b), which dominates two points with much higher hardware area values and close power dissipation values. Finally, points (c) and (d) are chosen blindly from the Simulated Annealing algorithm for the corresponding constraints, since they do not have any other competitor points in their range.

In Figure 4.2 we observe again a similar case as that of point (a) in Figure 4.1 for the point above it. However, examining this graph makes it obvious that hardware area is not the only defining parameter regarding the selection between two close points. In the case of point (c), where the hardware area is higher than that of the point on the right of (c) but the power dissipation is lower, the Simulated Annealing algorithm selects (c) because the power dissipation has larger weight in comparison to hardware area in the cost function. This dictates that for points of so close values as it is in this case, the point corresponding to more hardware area - less power dissipation is chosen. Again, points (b) and (d) are chosen blindly, since they do not have any competitors in range.

Finally, in Figure 4.3 we identify only two solutions in the power reduction range. These two points have no competitors, so they are blindly chosen as in the previous figures. Point (a) is selected when the power dissipation boundary is close to the boundary of the power reduction range and point (b) is selected in all the other cases, which include power dissipation boundaries below that of point (a).

Conclusion

In this thesis, we disserted on the possibility to find the most appropriate partition between software and hardware space for a DVB-H receiver module, while aiming to reduce power consumption. This chapter is organized as follows. In Section 5.1, a summary of our investigation is given, while in Section 5.2, possible future directions are presented.

5.1 Summary

Chapter 2 provided the reader with the necessary background on the key aspects of this thesis. Initially, an overview of the DVB-H standard was given, highlighting its features. The IP Datacast (IPDC) protocol stack was presented and the layers under investigation were explained. Then, the main components of IC power dissipation and software power dissipation in an embedded microprocessor were presented. An introduction to the hardware/software partitioning problem was given and subsequently, the basics of the Simulating Annealing method, which was used as our partitioning methodology, were introduced.

In Chapter 3, we described the DVB-H system under investigation, which consists of an ARM processor and an Altera's Stratix FPGA, and we decomposed its functionality into basic building blocks, which constitute the partitioning objects of our methodology. This decomposition was realized taking into account the most appropriate level of granularity, and three paths were identified in the initial partitioning, corresponding to video (H.264), audio (AAC+), and file delivery (FLUTE) content in the IPDC protocol stack. Next, a brief explanation of the way the hardware system was realized in the chosen Altera's Stratix FPGA was given. The estimation methodology for the FPGA's power dissipation with the chosen tools (ModelSim for simulation - Quartus II software's PowerPlay suite for power estimation) was discussed next. The methodology for estimating power dissipation of the ARM processor executing an application followed. Sim-Panalyzer, an augmentation to the SimpleScalar performance simulator of the ARM, was used as an estimation tool. Since Sim-Panalyzer requires ARM-executables in order to perform the estimations, an ARM-Linux cross compiler was built using a combination of gcc, glibc, binutils, and linux kernel modules. Since the ARM processor and the Stratix FPGA communicate via the system's bus, a power estimation methodology for their communication was presented, based primarily on the bus signals' switching activities, which were calculated for our system. Finally, a description of the, Simulated Annealing based, partitioning methodology was given, highlighting among other basic parameters, its main component, the cost function.

Chapter 4 provided the results for the power estimations and the partitioning methodology. Initially, the hardware areas of the building blocks were given after the hardware

design synthesis. Then, for each an every block, two power estimation values, one when it resides in hardware and another when it resides in software, were presented. Then, setting a range of constraints for the overall system's power dissipation, we utilized the Simulated Annealing algorithm to search for the most appropriate partitioning solution, considering power dissipation, hardware area and hardware/software communication, for each of the three paths. Next step was to examine the partitioning scenarios and to identify the common solutions that would induce reduction of the system's power dissipation. Consequently, four partitioning scenarios were extracted, providing power reductions between approximately 5% - 35% depending on the hardware area available in the FPGA.

5.2 Future Directions

We conclude this chapter and the research performed in this thesis by providing the following directions for future research:

- We observed that the system's initial partition involved three paths (video, audio and files) that had some common building blocks (Section Filter, IP Filter and UDP filter). The Simulated Annealing algorithm, on the other hand, could only examine one path at a time, extracting the best partition without taking into account whether this partition is also appropriate for the rest of the paths. This obliged us to perform this work by hand, looking at which common solutions can be viable. An addition to the algorithm's functionality that would examine multiple paths with common building blocks is one possible direction.
- Since meeting timing constraints has been extensively studied as a hardware/software partitioning problem, while our Simulated Annealing algorithm considers only power dissipation and hardware area as constraints, the performance objective can be added to the cost function, allowing more comprehensive searches for the best solution according to system's specifications.
- While applying our partitioning methodology, we realized that the communication between hardware and software space consumes relatively considerable amounts of power and sometimes the communication factor becomes dominant over that of hardware and software implementations. A modified methodology that would only consider communication constraints between blocks would save the designer from time-costly power estimation procedures for hardware and software.
- In order to acquire power estimations for the hardware implementation, the building blocks had to be implemented in hardware from scratch, considering the reference software implementation. This inevitably implies that it is on the designer's choice what method to use to implement the software functions in hardware, creating possible variations in the exact functionality, affecting further the power estimations. A tool that would automatically generate a VHDL model describing as close as possible the functionality of the software implementation would solve this problem efficiently.

Bibliography

- [1] Digital Video Broadcasting (DVB). [Online]. Available: <http://www.dvb.org/>
- [2] G. Faria, J. A. Henriksson, E. Stare, and P. Talmola, "DVB-H: Digital Broadcast Services to Handheld Devices," in *Proc. IEEE*, vol. 94, no. 1, Jan. 2006, pp. 194–209.
- [3] S. Borkar, "Low Power Design Challenges for the Decade," in *ASP-DAC 2001*, pp. 293–296.
- [4] J. T. Kao and A. P. Chandrakasan, "Dual-Threshold Voltage Techniques for Low-Power Circuits," in *IEEE JSSC*, July 2000, pp. 1009–1018.
- [5] F. Hamzaoglu and M. R. Stan, "Circuit-Level Techniques to Control Gate Leakage for sub-100nm CMOS," in *Proc. ISLPED*, Aug. 2002, pp. 60–63.
- [6] A. Keshavarzi, S. Narendra, S. Borkar, C. Hawkins, K. Roy, and V. De, "Technology Scaling Behavior of Optimum Reverse Body Bias for Standby Leakage Power Reduction in CMOS IC's," in *Proc. ISLPED*, 1999, pp. 252–254.
- [7] C. H. Kim, J. Kim, S. Mukhopadhyay, and K. Roy, "A Forward Body-Biased Low-Leakage SRAM Cache: Device and Architecture Considerations," in *Proc. ISLPED*, Aug. 2003, pp. 6–9.
- [8] L. Benini and G. de Micheli, "System-level power optimization: techniques and tools," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, no. 2, pp. 115–192, 2000.
- [9] J. Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," in *Design Automation Conference*, 1999, pp. 122–127.
- [10] Philips DVB-H solution BGT215 (TV-on-mobile for Europe and Asian DVB-H markets). [Online]. Available: <http://www.semiconductors.philips.com/acrobat/literature/9397/75015522.pdf>
- [11] Altera Corporation - Stratix FPGAs. [Online]. Available: <http://www.altera.com/products/devices/stratix/stx-index.jsp>
- [12] Mentor Graphics - HDL Designer. [Online]. Available: http://www.mentor.com/hdl_designer/
- [13] Mentor Graphics - ModelSim. [Online]. Available: <http://www.model.com/>
- [14] Altera Corporation - Quartus II. [Online]. Available: <http://www.altera.com/products/software/products/quartus2/qts-index.html>
- [15] Sim-Panalyzer, The SimpleScalar-Arm Power Modeling Project. [Online]. Available: <http://www.eecs.umich.edu/~panalyzer/>

- [16] SimpleScalar Tool Set. [Online]. Available: <http://www.simplescalar.com/>
- [17] *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, ETSI Std. EN 300 744 v1.5.1, 2004.
- [18] *Transmission System for Handheld Terminals (DVB-H)*, ETSI Std. EN 302 304 v1.1.1, 2004.
- [19] *Generic coding of moving pictures and associated audio information*, ISO/IEC International Std. 13 818-1/13 818-2, 1994.
- [20] *Digital Video Broadcasting (DVB) - DVB specification for data broadcasting*, ETSI Std. EN 301 192 v1.3.1, 2003.
- [21] G. Martinez, "DVB-H Next Steps: the Convergence of Broadcast and Mobile Systems," in *Proc. IAB DVB World Conference*, 2005.
- [22] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 3550, July 2003.
- [23] T. Paila, M. Luby, R. Lehtonen, V. Roca, and R. Walsh, "FLUTE - File Delivery over Unidirectional Transport," IETF RFC 3926, Oct. 2004.
- [24] S. Wenger, T. Stockhammer, and D. Singer, "RTP payload for transport of H.264/AVC Content," IETF RFC 3984, Feb. 2005.
- [25] *Information Technology - Coding of audio-visual objects - Part 10: Advanced Video Coding*, ISO/IEC International Std. 14 496-10, 2005.
- [26] J. v. d. Meer, D. Mackie, V. Swaminathan, D. Singer, and P. Gentric, "RTP payload for transport of generic MPEG-4 content," IETF RFC 3640, Nov. 2003.
- [27] J. Postel, "User Datagram Protocol (UDP)," IETF RFC 768, Aug. 1980.
- [28] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft, "Asynchronous Layered Coding (ALC) Protocol Instantiation," IETF RFC 3450, Dec. 2002.
- [29] —, "Layered Coding Transport (LCT) Building Block," IETF RFC 3451, Dec. 2002.
- [30] —, "Forward Error Correction (FEC) Building Block," IETF RFC 3452, Dec. 2002.
- [31] M. Pedram, "Power Minimization in IC Design: Principles and Applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 1, pp. 3–56, 1996.
- [32] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," in *Proc. IEEE*, vol. 83, Apr. 1995, pp. 498–523.
- [33] M. Stan and W. Burlison, "Bus Invert Coding For Low Power I/O," *IEEE Transactions of VLSI Systems*, pp. 49–58, Mar. 1995.

- [34] E. P. Harris, S. W. Depp, W. E. Pence, S. Kirkpatrick, M. Sri-Jayantha, and R. R. Troutman, "Technology directions for portable computers," in *Proc. IEEE*, vol. 83, Apr. 1995, pp. 636–657.
- [35] V. Tiwari, S. Malik, A. Wolfe, and M. T. C. Lee, "Instruction level power analysis and optimization of software," *Journal of VLSI Signal Processing*, 1996.
- [36] A. Kaplan, M. Sarrafzadeh, and R. Kastner, "A Survey of Hardware/Software System Partitioning," Tech. Rep., 2003.
- [37] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design and Test*, Dec. 1993.
- [38] R. Gupta and DeMicheli, "System-level synthesis using re-programmable components," *EDAC*, 1992.
- [39] M. L. Vallejo and J. C. Lopez, "On the hardware-software partitioning problem: System Modeling and partitioning techniques," *ACM TODAES*, V-8, 2003.
- [40] T. Wiangtong, P. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Journal Design Automation for Embedded Systems*, V-6, 2002.
- [41] Track Hardware Software, "COSYN: Hardware-Software Co-synthesis of Embedded Systems."
- [42] F. Vahid, G. Jie, and D. D. Gajski, "A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning," *European Design Automation Conference (EuroDAC)*, 1994.
- [43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [44] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, pp. 1087–1092, 1953.
- [45] J. Henkel and R. Ernst, "The Interplay of Run-Time Estimation and Granularity in HW/SW Partitioning," *4th International Workshop on Hardware/Software Co-Design, Pittsburgh*, 1996.
- [46] Altera White Paper - Stratix Device Backgrounder. [Online]. Available: http://www.altera.com/literature/wp/wp_stx_backgrounder.pdf
- [47] Doulos - The Designer's Guide to VHDL. [Online]. Available: http://www.doulos.com/knowhow/vhdl_designers_guide/
- [48] "IP Datacast over DVB-H: Architecture," DVB BlueBook A098, Nov. 2005.

- [49] Altera - Quartus II PowerPlay Power Analysis & Optimization Technology. [Online]. Available: http://www.altera.com/products/software/products/quartus2/design/qts-power_analysis.html
- [50] gcc - the GNU Compiler Collection. [Online]. Available: <http://gcc.gnu.org/>
- [51] glibc - GNU C Library. [Online]. Available: <http://www.gnu.org/software/libc/>
- [52] binutils - GNU Binutils. [Online]. Available: <http://www.gnu.org/software/binutils/>
- [53] Debian - Linux Kernel Headers. [Online]. Available: <http://packages.debian.org/linux-kernel-headers>
- [54] Dan Kegel's Crosstool. [Online]. Available: <http://www.kegel.com/crosstool/>
- [55] F. Catthoor, E. d. Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.
- [56] W. Fornaciari, D. Sciuto, and C. Silvano, "Power estimation for architectural exploration of HW/SW communication on system-level buses," in *Seventh international Workshop on Hardware/Software Codesign (Rome, Italy). CODES '99*, 1999.
- [57] S. Banerjee and N. Dutt, "Very fast Simulated Annealing for HW-SW partitioning," CECS, Tech. Rep., June 2004.
- [58] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation Techniques," *IEEE Transactions on VLSI*, V-9, 2001.
- [59] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf Placement and Routing Package," *IEEE Journal Solid-State Circuits*, V-20, 1985.
- [60] P. Eles, Z. Peng, K. Kuchinski, and A. Doboli, "System Level Hardware/Software Partitioning based on simulated annealing and Tabu Search," *Journal Design Automation for Embedded Systems*, V-2, 1997.

Curriculum Vitae

Ioannis Koryfidis