Design and Characterization of the Time Division Multiplexing concept on a dual clock Imaging DSP

THESIS

submitted in partial fullfilment of the

requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Giannoula Ioanna Kyriakou

This work was performed in:

Delft University of Technology

Department of

Computer Engineering

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Design and Characterization of the Time Division Multiplexing concept on a dual clock Imaging DSP"** by **Giannoula Ioanna Kyriakou** in partial fulfillment of the requirements for the degree of **Master of Science**.

APPROVED:

Dr. Ir. Rene van Leuken

Dr. Stephan Wong

Dr. Amir Zjajo

Abstract

The market demand for quality imaging on portable devices has increased heavily in the last few years. This increase on the demands has a direct effect on the complexity of the imaging-dedicated devices, making the Digital Signal Processors (DSP) development a key factor in the race of imaging quality.

More complex devices have been developed to cope with the technical challenges in terms of computational capability, power consumption and area footprint of the devices, especially in the portable market. This challenge is the key to the success of the imaging processor in the market, forcing companies to develop new and creative ways to reduce area and power while keeping performance.

Experimental results showed that by applying the Time Division Multiplexing (TDM) concept, as a clock domain crossing technique, in an Imaging DSP (IDSP), the area that it occupies as well as the power that it consumes can be reduced. In this thesis, four new implementations of the IDSP are proposed and their functionality was proven to be correct. All these designs were synthesized and results about area and power were extracted and analyzed.

Acknowledgments

I would like to express my wholehearted gratitude to my supervisor Harm Peters and my co-advisor Noemi Quintana Afonso from Intel. Their patience, expertise and advising, contributed to the completion of this thesis. Furthermore, I would like to thank my supervisor, Prof. Rene van Leuken from TU Delft for his assistance at all levels of this project. The door to Prof. Rene van Leuken office was always open whenever I had a question about my research or writing.

Finally, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Delft, October 2016

Contents

Delft, October 2016

List of Figures

List of Tables

Delft, October 2016

List of Abbreviations

VLSI	Very Large System Integration
SIMD	Single Instruction Multiple Data
VLIW	Very Large Instruction Word
GPP	General Purpose Processors
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
FIFO	First In First Out
FF	Flip Flop
DSP	Digital Signal Processor
IDSP	Imaging Digital Signal Processor
CDC	Clock Domain Crossing
VRF	Vector Register File
TDM	Time Division Multiplexing
IS	Issue Slot
RTPG	Random Test Pattern Generation

1 Introduction

The general purpose processors are designed to execute multiple applications and perform multiple tasks. They can be very expensive especially for small devices that are designed to perform special tasks and they lack in terms of performance. Therefore, the application specific processors emerged as a solution for high performance, cost effective and low power consumption processors. They can have a better performance since they are designed for a large amount of math operations involved in signal processing. The application specific processors have become a part of our life and can be found in every device that is used on a daily basis. They can be classified into three major categories Digital Signal Processors, Application Specific Instruction Set Processors and Application Specific Integrated Circuit.

The digital signal processing cameras use a DSP chip to digitize analog video streams. Analog video streams are generated by charge-coupled device – or CCD – chips, which DSP chips then convert into a digital video signal. The DSP cameras provide a number of notable advantages over analog cameras, including increased brightness, added stability, greater sharpness and resolution, greater power efficiency, reduced sensitivity to noise and more. The smartphones, digital cameras and other devices, meanwhile, use digital signal processing to provide high-quality images or video in a portable and energy efficient fashion.

In a big chip not all the blocks need to be clocked at the highest possible frequency in order to achieve the desired level of performance. There can be few blocks which inherently work slowly and, therefore, can be clocked at slower clock than blocks like the core which require high frequency clock for maximum throughput. For the correct communication between these clock domains synchronizer techniques have to be implemented.

Nowadays, most of the companies aim to reduce the area that their processors occupy and the power that they consume, by keeping the same performance or by increasing it. One of their major factor that prevents them from this is the routing congestion due to chip size which is being reduced continuously, so there is less area available for the wires to be routed. The problem is especially acute as the interconnects are the performance bottleneck in integrated circuits and also they are damaging the quality of the designs.

More specifically, routing congestion is the most common reason for the unexpected increase in the delay of a net that lies on a critical path and this can cause a design to miss its frequency target. Moreover, a densely congested design is likely to result in a lower manufacturing yield than a similar uncongested design and that is because congested layouts tend to have larger critical areas for the creation of shorts and opens due to random effects. Also, it is observed that the routing congestion in future circuits is expected to be even more severe because of growing design complexity and continuous technology scaling.

In this thesis, the TDM concept was used as a clock domain synchronizer technique in order to reduce the amount of area that the IDSP occupies, as well as the amount of wires that it contains. Section 1.1 gives an overview of the related work while subsequently in Section 1.2 it is motivated how by applying the TDM concept in an IDSP the area that it occupies can be reduced. In Section 1.3 the thesis goals will be formulated. This chapter is concluded in Section 1.4 with an overview of the organization of this thesis.

1.1 Related work

FIFOs are often used to safely pass data from one clock domain to another asynchronous clock domain. In [9], one method that is used to design, synthesize and analyze a safe FIFO between different clock domains is explained. Since asynchronous FIFOs are clocked from two different clock domains, the clocks are running at different speeds. Another way to synchronize different clock domains is through handshaking protocol, by sending some control/status signals through the clock domain boundary to synchronize the transmitter and receiver. In [11], the handshake protocol was used in an asynchronous AHB Bus Interface. A completely different way to synchronize two different clock domains is described in [13]. This paper presents an interface technology, called Direct Conversion Method (DCM), that is specifically aimed at SoCs with multiple clock domains derived by dividing a master clock. Since the ratio of the resulting clock periods is rational, if the latching edged of two clock domains are aligned at some time, they are guaranteed to periodically align thereafter. Synchronization failure can be completely avoided without any handshake circuit between the clock domains.

1.2 Motivation

The IDSP is the Imaging DSP that is used in the Imaging Processing Unit of Intel. Current and past IDSPs have been causing problems with placement and routing in physical design. These problems are caused by a combination of many factors. Some of them are the wide vector busses (>=512 bits) that connect the different IS of the VLIW core, the currently register files placement because of its implementation and the placement of the vector store units away from the memory interface.

The problem of the placement of the vector store units away from the memory interface was solved by placing all the load/store units in separate ISs and place them closer to the interface. So the two main concerns are the wide vector busses and the implementation-placement of the register files. In this master thesis, the main architecture will be studied in order to find different architectures so that the width of those vector busses to be reduced and the implementation of the register file to be the most suitable for a better placement and performance.

In order to achieve this, our design had to be split into two clock domains and one of these domains has to operate in double frequency than the other one. The synchronization of this two clock domains will be done by using the Time Division Multiplexing concept, which is widely used in telecommunication systems. The TDM is a method of putting multiple data streams in a single signal by separating the signal into many segments, each having a very short duration. This kind of communication requires a multiplexer/demultiplexer at each end of the line.

In our case, the one clock domain will do the multiplexing of the signals and the other one the demultiplexing of them. By using this concept, it will be possible to reduce a big amount of wires that exist in the current IDSP and also to reduce the amount of area that it occupies with very little impact on performance.

1.3 Thesis Goals

The goals of this master thesis are the following:

- Verify that the TDM concept is applicable as a Clock Domain Crossing technique in the IDSP, by implementing it in the existing architecture and verify it.
- Use the TDM concept to find many different solutions in terms of design, so the number of wires to be reduced significantly.
- Synthesize the different architectures that will be designed, in order to collect results about area, power consumption and timing.
- Evaluate these results, so a conclusion will be extracted on which architecture is the most suitable in terms of area, power consumption and timing.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2

It provides the reader with the necessary background information of the most important terms that are used in the thesis. It includes information about the VLSI Design flow, about application specific systems, different CDC techniques and static timing analysis terminology.

Chapter 3

In this chapter, the entire architecture of the IDSP is presented and the process which leads us to the final modified architecture, by using the TDM concept, is explained briefly. Furthermore, the limitations of the new architectures are referred.

Chapter 4

The verification environment and the modifications that have to be applied on it are explained and results about area, power and timing for every different architecture are depicted and evaluated.

Chapter 5

It gives a reflection of what has been achieved during this research and is finished with an overview of future work.

2 Literature Review

This chapter will provide background information of the keywords present in this thesis. To start with, in Section 2.1 the VLSI Design Flow will be explained, as it will be used for the purpose of this thesis. In Section 2.2 different architectures are explained as well as different processors. Then a terminology about static timing analysis will be defined. At the end of the chapter, many Clock Domain Crossing (CDC) techniques will be introduced.

2.1 VLSI Design Flow

There are several steps in the implementation of a digital circuit, Figure 1, [1] . In this section, the full design flow is presented and each step of it is explained. Design specifications come first, they describe with an abstract way the entire architecture, functionality etc. of the digital IC to be designed. Next step is the RTL coding, in which, the RTL description is done using a Hardware Description Language and then, it is tested to verify the functionality of the circuit. Next step is synthesis, in which the RTL code is converted to a gate-level netlist and estimated data such as area and power can be extracted. In physical design step, the netlist is converted into a geometrical representation of shapes by using technology libraries that are provided by the fabrication houses. The next step after physical design is the fabrication process,in which, the designs are fabricated onto silicon dies and packed into ICs.

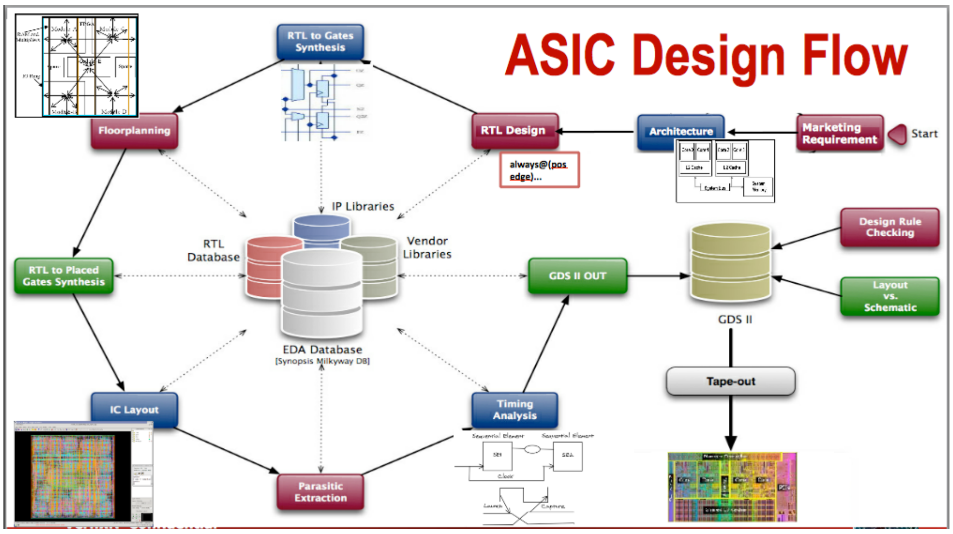


Figure 1: *The ASIC Design Flow*

2.2 Digital Signal Processor

A digital signal processor is an integrated circuit designed for high-speed data manipulations widely used in audio, video, voice and image manipulation. The DSPs have dedicated logic to perform basic mathematical functions like addition, subtraction, multiplication and division faster than the general purpose microprocessors. Some fundamental DSP algorithms that depend heavily on multiply-accumulate performance are the FIR filters and the Fast Fourier transform (FFT).

Typical DSP systems consist of a DSP chip, memory, an analog to digital converter, a digital to analog converter and communication channels.

DSP chip: In a DSP chip, there are the arithmetic units that implement the mathematical functions such as multiplication and addition. It is the part that makes the DSP so fast in comparison with traditional processors.

Memories: There are special memory architectures that are able to fetch multiple data and/or instructions at the same time. In many cases, the DSP reads some data, operates on it, and writes it back.

A/D and D/A converters: An Analog to Digital converter turns the analog input into digital data and a Digital to Analog converter performs the reverse process.

Moreover, the DSPs are using either fixed-point arithmetic or floating point arithmetic. Each one has its advantages and disadvantages. For example, with fixed point arithmetic there is a large speed benefit and cost benefit due to reduced hardware complexity, but with floating point, the cost and complexity of software development are reduced in exchange with more expensive hardware, since it is easier to implement algorithms in floating point.

2.3 SIMD and VLIW architectures

The SIMD (Single Instruction Multiple Data) is a technique of performing the same operation, be it arithmetic or otherwise, on multiple pieces of data simultaneously, Figure 2. Such machines exploit data level parallelism, but not concurrency. There are parallel computations, but only a single process (instruction) at a given moment. It is widely used for the 3D graphics and the audio/video processing in multimedia applications.

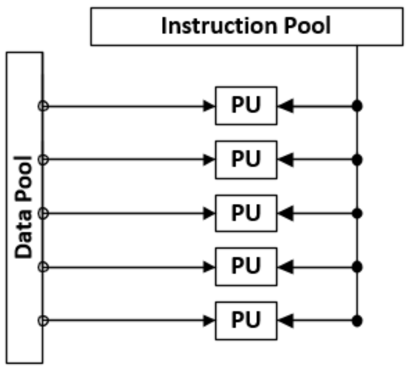


Figure 2: *SIMD*

The VLIW (Very Large Instruction Word) architectures take advantage of the instruction level parallelism (ILP). A VLIW processor executes multiple instructions in parallel by putting them into a very long instruction word, Figure 3, and then assign them on the appropriate functional units as it is shown in Figure 4. These functional units can be floating point units, integer ALU, load/store units, but the challenge is to design a compiler that decides how to build these very long instruction words in order to allocate useful work to every slot of every instruction.

Operation 1	**Operation 2**	**Operation 3**	**Operation 4**

Figure 3: *The instruction word format*

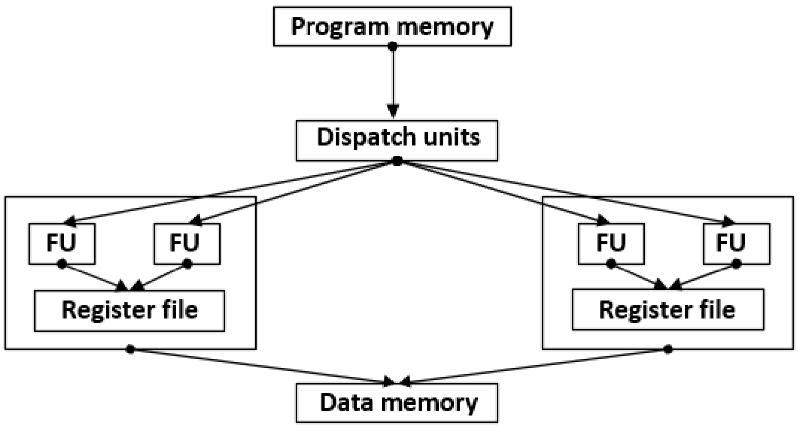


Figure 4: *A VLIW processor*

The VLIW processors are often used in digital signal processing applications where high performance is critical. Moreover, the SIMD cores can be combined with the VLIW architecture for further increasing of throughput and speed.

2.4 Application Specific Systems

Some of the typical approaches of building an application specific system or an embedded system are to use one or more of the following implementation methods: GPP, ASIC or ASIP.

GPP: The functionality of the system is exclusively build on the software level. Although the biggest advantage of such a system is the flexibility, it is not optimal in terms of performance, power consumption, cost, physical space and heat dissipation.

ASIC: Compared to a GPP, the ASIC-based systems offer better performance and power consumption but at the cost of flexibility and extensibility. Although it is difficult to use an ASIC for tasks other than what they were designed for, it is possible to use a GPP to perform the more general less demanding tasks in addition to an ASIC in the same system.

ASIP: In this approach, an ASIP is basically a compromise between the two extremes; the ASIC processors are being designed to do mostly a very specific job with high performance but with minimal room for modifications and the general purpose processors which cost a lot more than an ASIP but with extreme flexibility at what they do. Due to this flexibility and low price, the ASIP are great to be used in embedded and system-on-a-chip solutions.

Application Specific Instruction Set Processor

An ASIP (application-specific instruction set processor) is a component used in system-on-chip design. The instruction set of an ASIP is tailored to benefit a specific application. This specialization of the core provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC. For an ASIP design, there is a base architecture with specific ISA which is customized by adding extra components or by tweaking the existing ones, in order to make the execution of this application more efficient. There are two ways to design them, from scratch and from a base architecture. The second one is much easier than the first one and there are specific tools for the automatic generation of processor architecture (HW) and instruction set (SW).

In [8], an FFT ASIP is presented, in which, design choices related to memories and the instruction set of FFT FU had to be made, based on the architecture of AVISPA-EXP1 (processor) and the application to be executed. The results of area, power and performance are comparable to those of an ASIC solution. Moreover, an ASIP template was used in [12] for one of the most compute and memory-intensive functions in video processing, motion estimation (ME). An instruction-set suitable for performing a variety of ME functions was developed and the ASIP architecture was based on a VLIW processor. Due to the results on area and power, the ASIP approach clearly demonstrates the benefits of alternative space/time computing solutions as opposed to the traditional approach of ASICs and GPP.

2.5 Clock Domain Crossing

The SoCs are becoming more and more complex these days. The CDC (clock domain crossing) occurs whenever data is transferred from a flip flop driven by one clock to a flip flop driven by another clock. Depending on the relationship of the two clocks, there are different types of problems in transferring data from the source clock domain to the destination clock domain. More specifically, the main issues of clock domain crossing are metastability, data loss and data incoherence [2].

Every flip flop has a specified setup and hold time in which the data input is not permitted to change before and after a sampling clock edge. In case of a setup or a hold violation the output signal of FF oscillates for an indefinite amount of time and at the end the FF settles down to either 1 or 0. This phenomenon is known as metastability and the FF is said to have entered in metastable state.

In order to ensure no data loss, the source data should remain stable for some minimum time, so the setup and hold time requirements are met with respect to at least one active edge of destination clock.

In case of multiple signals being transferred from one clock domain to another and each signal being synchronized separately, if all the signals change simultaneously, some of them may get captured in the first cycle in the destination domain and some other in the second cycle. This may result in an invalid combination of values in the destination domain.

2.6 Synchronizer techniques for multi-clock domain SoCs

According to [3], there are many different standard synchronization techniques for multi-clock domain SoCs and each one has its own advantages and disadvantages. There are some less complex synchronizers such as 2-FF synchronizer and some more complicated as FIFO and Handshake synchronizers.

Two FF synchronizer

The metastability is a probabilistic phenomenon. The meta-stable output converges to a stable value with time. Therefore, even if the input to the stage-2 FF still remains meta-stable, the probability that the output of the stage-2 FF will remain meta-stable for a full destination clock cycle is asymptotically close to zero. More specifically, in Figure 5, while sampling B1d in clock B domain there is a probability of B1q to go into metastable state, but the probability of B2q to be metastable is very close to zero.

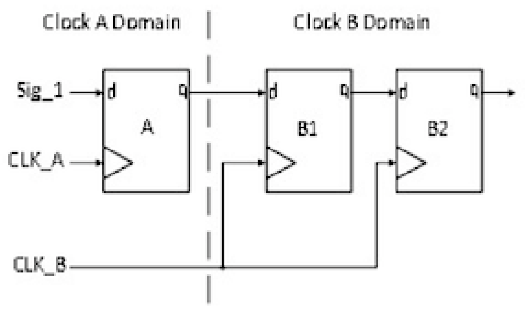


Figure 5: *The two FF Synchronizer.*

Toggle synchronizer

It is used in order to synchronize a pulse generating in source clock domain to destination clock domain and it cannot be synchronized using 2 FF synchronizer because the pulse can be skipped, which can cause the loss of pulse detection and the subsequent circuit may not function correctly.

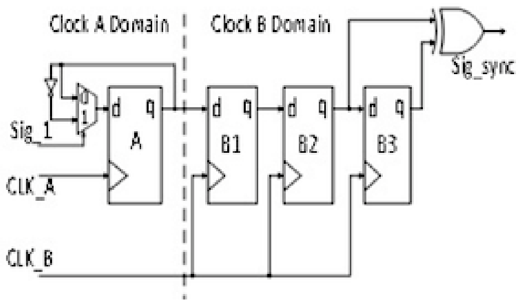


Figure 6: *The Toggle Synchronizer.*

Handshake synchronization

In this synchronization scheme, Figure 7, the request and acknowledgment mechanism is used to guarantee the sampling of correct data into destination clock domain is irrespective of the clock ratio between the source and destination clock domain and it is mainly used to synchronize vector signals that do not change very frequently. The handshaking works great but reduces the bandwidth at the clock crossing interface because each piece of data has many cycles of series handshaking.

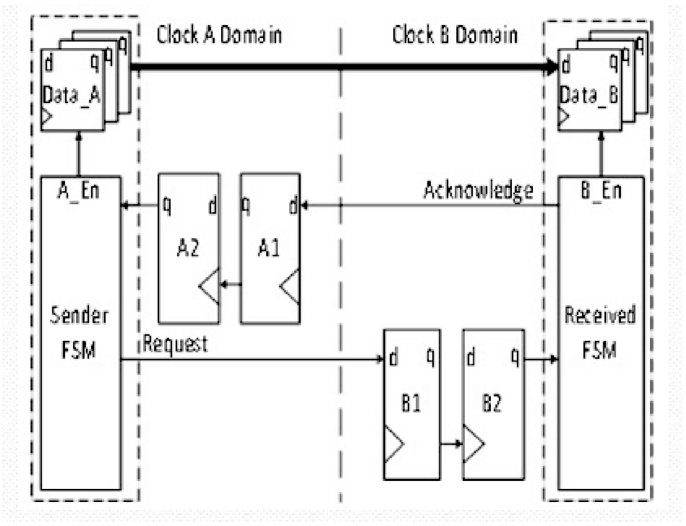


Figure 7: *The Handshake Synchronization.*

Asynchronous FIFO synchronization

An asynchronous FIFO, Figure 8, is the best way to synchronize vector data that change continuously between two asynchronous clock domains and provides full synchronization independent of clock frequency. Moreover, it can increase bandwidth across the interface and still remain a reliable communication scheme. The FIFO is managed as a circular buffer using pointers, write and read pointer, and whenever one catches the other, the FIFO is either empty or full. Furthermore, the FIFO synchronization uses gray encoding for the pointers to guarantee that at most one bit will change per cycle.

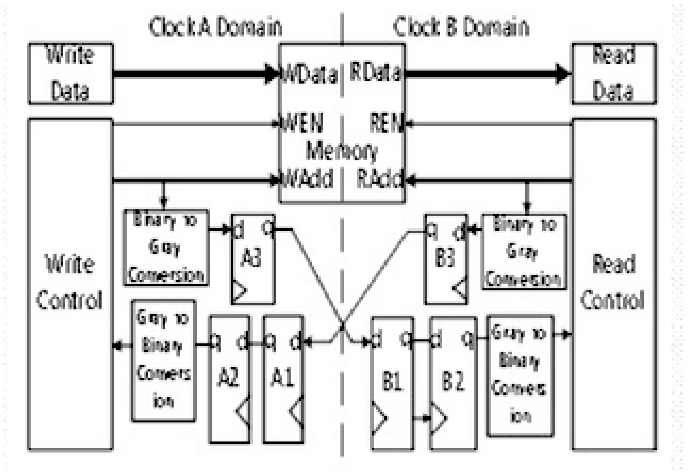


Figure 8: *The asynchronous FIFO Synchronization.*

2.7 Static timing analysis terminology

Propagation Delay

The propagation delay, symbolized t_{pd}, is the time required for a digital signal to travel from the input(s) of a logic gate to the output. Often on manufacturers' datasheets, this refers to the time required for the output to reach 50% of its final output level when the input changes to 50% of its final input level.

In a more complex circuit every input can go through many different paths until it reaches the output. Every path in the circuit also has a different delay and for a single input, the propagation delay is the delay of the path with the longest delay because it takes that much time for that input to give a valid output. For a multiple input circuit, the propagation delay is equal to the maximum delay in the circuit. This is because if the output of the circuit is read shorter than the maximum delay, in some cases the output might not be valid and can result in false interpretations and actions. The difference in propagation delay of different circuit paths can result in glitches on the output.

Skew

The clock of a circuit is distributed from a single source to all the memory elements of the circuit which are called registers of FF. Skew is a phenomenon in synchronous circuits in which the same sourced clock signal arrives at different components (latches or flip flops) at different times. This can be caused because of wire interconnect length, temperature variations, parasitic capacitive coupling.

Setup and Hold Time

Considering a D-type Flip Flop, just before and after the clock edge, there is a critical time region where the D input must not change. The region just before the clock edge is called setup time (t_{su}) and the region just after the clock edge is called hold time (t_h). If either is violated, correct operation of the FF is not guaranteed and metastability can result. Moreover, a flip-flop takes the value of the signal at its input, on the positive edge of the clock, and send it to the output of the FF after a small delay called $t_{clk->Q}$.

For a better understanding of the setup and the hold time, Figure 9 will be used as an example, [4]. More specifically, in order to propagate a valid signal from D2 to D3 the time that is required is $t_{clk->Q} + t_{logic}$ and in order the FF3 to latch it, the signal has to be stable at least t_{setup} time before the next positive edge of the clock. So in order to prevent setup violations the following equation has to be satisfied.

$$t_{clk->Q} + t_{logic} + t_{setup} <= t_{clock} - t_{skew}$$

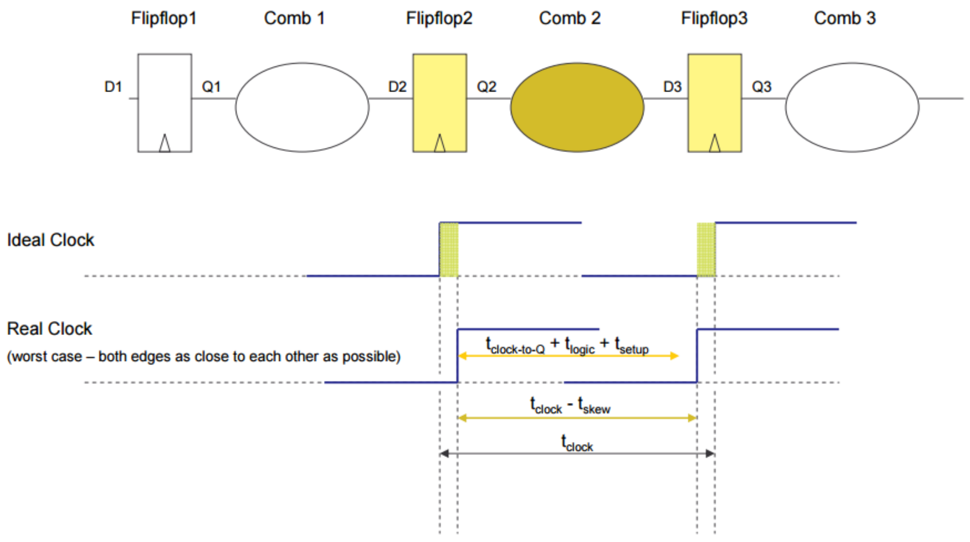

Flipflop1
Comb 1
Flipflop2
Comb 2
Flipflop3
Comb 3
D1
Q1
D2
Q2
D3
Q3
Ideal Clock
Real Clock
(worst case – both edges as close to each other as possible)
$t_{clock\text{-}to\text{-}Q} + t_{logic} + t_{setup}$
$t_{clock} - t_{skew}$
t_{clock}

Figure 9: *Example of setup time violation.*

Another important timing measurement is the hold time. By having as reference the Figure above and Figure 10, for D2 to be able to send its signal to Q2, it must be unchanged for t_{hold} time after the positive clock edge. That is, during this time, a signal from D1 should not be able to race through the combinational logic Comb1 and make it to D2. So in order to prevent the hold violations the following equation has to be satisfied.

$$t_{skew} + t_{hold} <= t_{clock->Q} + t_{logic}$$

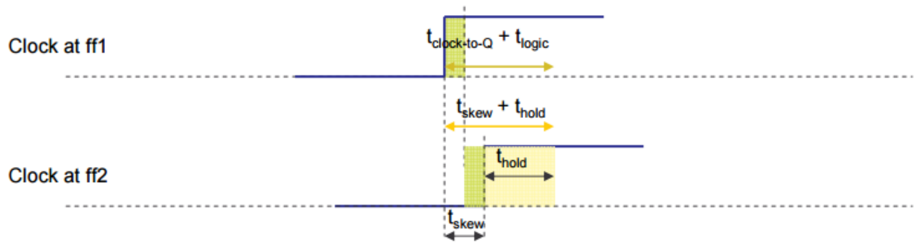


Figure 10: *Example of hold time violation.*

Slack and Critical path

The arrival time is the time that it takes for a signal to arrive at a certain point and the required time is the time that is the latest time that a signal can arrive without making the clock cycle longer than it is. Slack is the difference between the required time and arrival time, so a negative slack indicates that constraints have not been met, while positive slack indicates the opposite. Furthermore, the critical path is the path that creates the longest delay and violates the timing constraints, so it has negative slack.

3 Design and Implementation

Within this chapter, the design of the original IDSP will be presented and all the steps that were followed to achieve the final architecture of the IDSP will be explained. In the first three sections, the IDSP tile and the IDSP architecture will be explained while subsequently in Section 3.3 the approach of the TDM concept that was followed in order to achieve our aims is presented. In Section 3.4 and 3.5 details about the implementation of the TDM concept inside the VRF and around the vector IS of the IDSP will be given. Finally, in the last two sections, the two final architectures with the maximum reduction in terms of area will be presented and the limitations of the new design will be reported.

3.1 IDSP tile architecture

The IDSP processor is one of the main part of IDSP tile system as many other devices, such as an evq (Event queue with tracing capabilities), a gpc (Generic performance counter for tracing), a gp_reg (General-purpose registers), etc. Furthermore, the IDSP tile consists of buses, converters (CIO devices), pipelines, interfaces with memories and other devices and it is designed to work in one clock domain. According to Figure 11, the idsp_tile_logic is separated from memories, such as the program memory, vector memory, i-cache memory etc. There is a need to work with the idsp_tile, in order to be able to test the changes that are going to be applied in the generated RTL code of the idsp core.

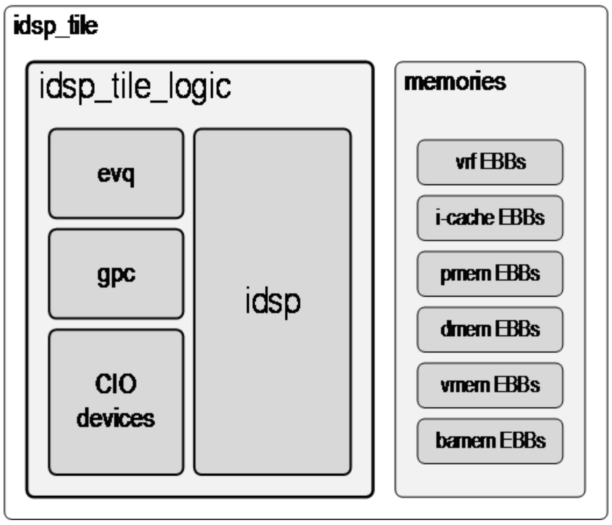


Figure 11: *The IDSP tile architecture*

3.2 IDSP architecture

The IDSP (Imaging Digital Signal Processor) is a VLIW processor. It has 512b vector datapath, 8/16/32 way SIMD vector support with 32/16/8 bits/element, 32b scalar datapath and 9 ISs. Also, the memory subsystem contains a program memory, an instruction cache, a data memory, a block access memory and a vector memory.

More specifically, the ISs in the IDSP core are grouped into 3 clusters of 3 ISs. As it is shown in Figure 12, there is a load/store cluster with 2 load/store units, a first compute cluster which contains a scalar IS and 2 vector ISs and a second compute cluster which contains the same amount of scalar and vector ISs. Each cluster according to its ISs contains a number of functional units, VRFs and scalar register files. The distribution of the function units over the ISs is made according to some criteria. For example, it is important to balance the ISs in terms of area or to have functional units in the same IS that do not interfere with each other much. Furthermore, the internal communication in a cluster is implemented with minimal latency, while the communication between different clusters is pipelined deeper to avoid timing issues. So, the communication between the IS in one cluster will become by using non-pipelined busses and the busses that cross from 1 cluster to another will be pipelined. It should be noted that with this architecture, the arrival time of the result of an IS into a register file will depend on the cluster in which the target register file resides.

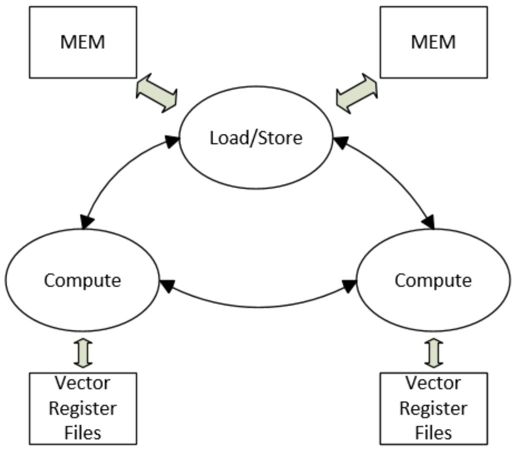


Figure 12: *Clustering of Issue Slots*

Finally, a scaled down architecture of the IDSP is going to be used, with one compute cluster, as it is shown in Figure 13, so the scalar and load/store clusters will be generated to support only one compute cluster. The compute cluster will contain one scalar IS, one vector IS, one scalar register file of size 32x32b connected to the scalar IS, one scalar register file of size 16x32b connected to the vector IS, one VRF of size 8x512b connected to the same vector IS and a bypass network connected to the VRF. The vector IS consists of 5 functional units. A better explanation of the architecture will be described afterward. Also, as it is depicted in the figure below the output of the IS of the compute cluster has a feedback to its own scalar register file and VRF, but it is also connected to the register files of other ISs located either on the scalar cluster or in the load/store cluster. These interconnections are controlled via the RSN network.

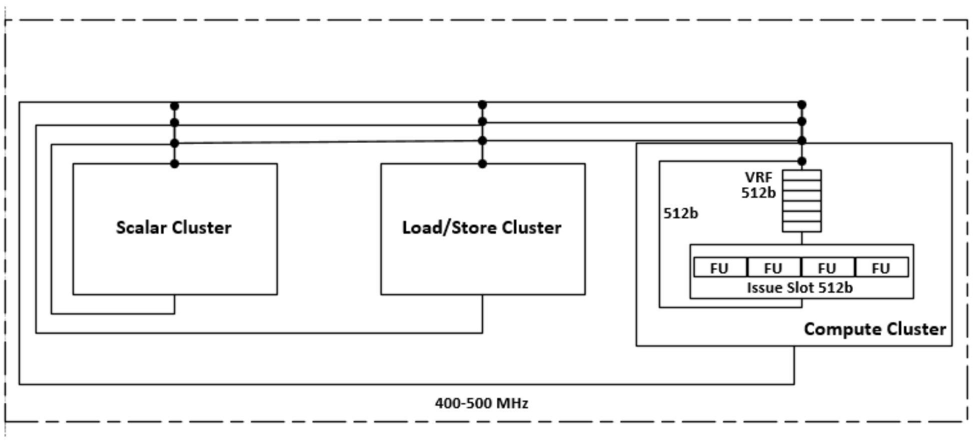


Figure 13: *The IDSP scaled down architecture*

3.2.1 Issue Slot architecture

A brief review of the structure of the IS has to be described in order to be easy to understand the changes that have to be made in order to apply the TDM concept inside the vector IS of the compute cluster. The vector ISs consists of the following components:

- ODEC
- ICN
- EXU's
- OCN

ODEC: This component is responsible for enabling the appropriate FU in each clock cycle, based on the opcode signal which specifies the operation to be performed.

ICN: This network is responsible for sending the data to the execution units.

EXU's: In this component the instructions are executed. It consists of three components, the ctrl component which produces an enable signal when the operation is valid, the isolator component which is responsible for which input in the EXU is valid each clock cycle and the functional units which have as inputs the data and the operation type and produce the output data. Also, there is the probe component which is used for verification purposes.

OCN: It has as inputs the valid and the output data signals that the execution units produce and according to those signals the appropriate output data are propagated to the output of the vector IS.

In Figure 14, the architecture of the vector IS is depicted. The inputs are redirected to the ODEC and ICN components and according to the control signal of these two components, the input data will be sent to the EXUs. Accordingly, the results of the EXUs are propagated to the OCN and then to the outputs of the vector IS.

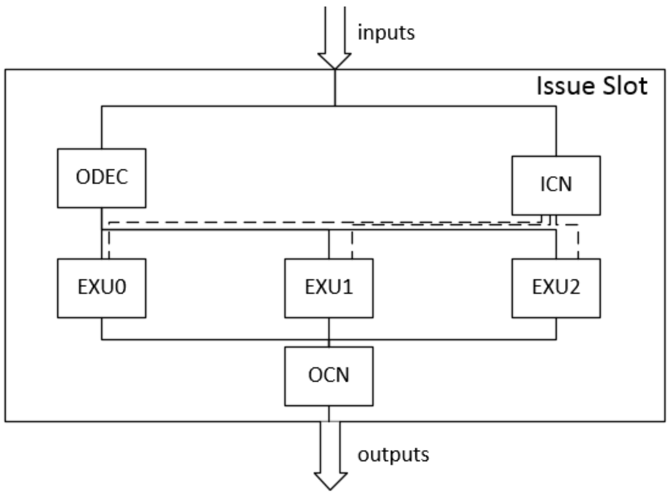


Figure 14: *The Vector Issue Slot architecture*

3.2.2 Bypass Network of the Vector Register Files

A bypass network is used when a simultaneous read and write to the same entry in the register file is happening, so the written data is bypassed to the read ports. The register file bypassing is widely used in modern pipelined architectures for minimizing the performance impact of the read-after-write (RAW) data hazards [10]. The main idea is to add multiplexers to the input of the ALU and supply the operands from the pipeline register instead of the register file.

In IDSP's VRF implementation there is a bypass network. In the specific architecture, when a read occurs, firstly, the data is read from the VRF and it is passed to the Bypass Network, where it is checked if the register that the data is going to be written is the same with the register that is going to be read, in order to bypass the written data.

To have a better view of how does this work, the exact connections between the VRF and the Bypass Network is depicted in the Figure 15. The write data and the registers are connected except for the VRF, also to the bypass network. In the bypass network, there is a control mechanism which checks if any of the registers of the two read ports is equal to any of the two registers of the write ports and according to this result, propagates the correct value of the data. It is obvious that there are going to be checked four equalities, because of the two read and two write ports.

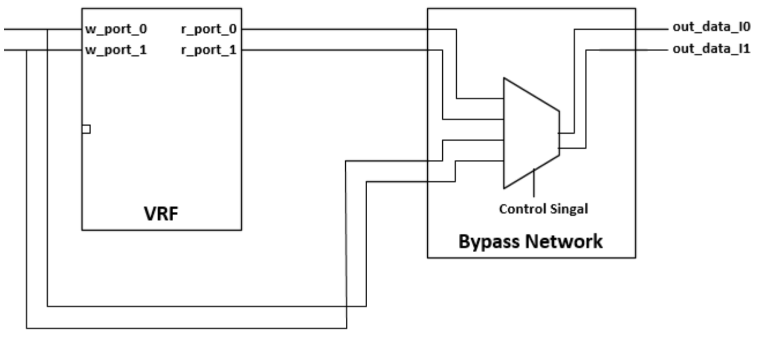


Figure 15: *Bypass Network Architecture*

3.3 Time Division Multiplexing concept

In order to achieve our first goal, which is to reduce the area that the IDSP occupies and consequently the amount of the wires that IDSP contains, the new design will consist of two different clock domains with the second one to operate in double frequency than the first one. For the efficient communication of these two domains, the TDM concept will be applied. In the TDM concept, which is widely used in telecommunication systems [5], incoming signals are divided into equal fixed-length time slots. After multiplexing, these signals are transmitted over a shared medium and reassembled into their original format after de-multiplexing, Figure 16, [6].

In our case, there will be a reduction by half in period time and the relationship between those two clocks, synchronous and aligned, is observed in Figure 17. In the first clock domain, the scalar and load/cluster will be included and the second clock domain will contain only the vector IS of the compute cluster.

According to this and the TDM concept, the way that the data was processed in the compute cluster will change. Considering that one clock cycle in the first clock domain matches with two clock cycles in the second one, the size of the vector IS in the compute cluster can be the half, because now the first half of the data path can be processed in the first clock cycle and in the second clock cycle the second half of the data will be processed.

Now the challenges are, except from the correct synchronization between the two clock domains, the communication of the components inside the compute cluster. For this reason, the communication of the VRF with the vector IS will change, as well as the communication of the components inside the vector IS.

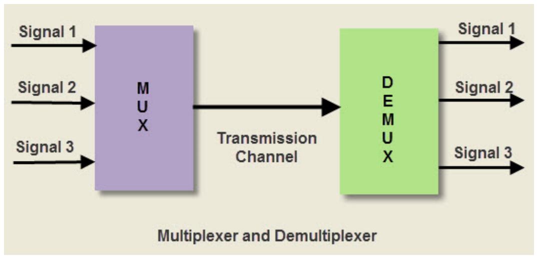


Figure 16: *Time Division Multiplexing process*

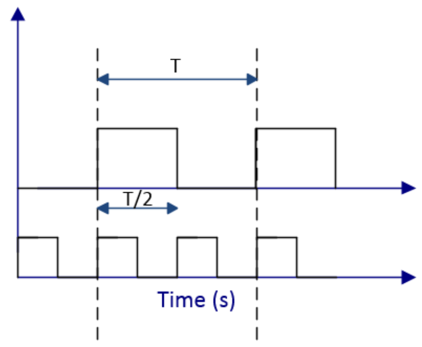


Figure 17: *Relationship between the two clocks*

3.4 IDSP modified architecture

The modified architecture consists of three different clusters, a scalar/control, a load/store and a compute cluster. In the new IDSP architecture, due to the TDM concept which leads the vector IS of the compute cluster to work on double frequency, the size of the vector IS in the compute cluster will be 256b. Consequently, its functional units will process data of 256b width, but the size of the VRF and the scalar register file will remain the same. The VRF is 8x512b wide and the scalar register file 16x32 wide.

In the new architecture, it is critical how the components inside the compute cluster will communicate because the data path between the two clock domains will be 512b but the size of the vector IS has to change, as well as the structure of the VRF. Another important issue is how the data will pass between the two clock domains. The modified architecture is depicted in the Figure 18, in which the two different clock domains are coloured differently and the paths related to them are presented.

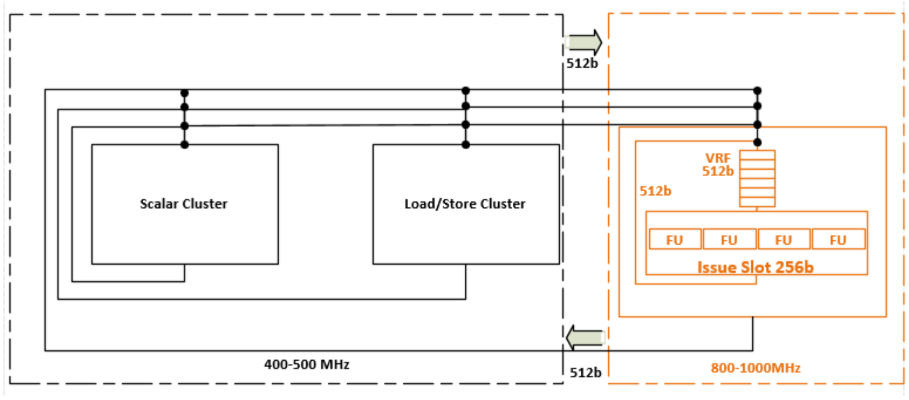


Figure 18: *The modified Architecture of the IDSP*

The TDM concept will be gradually introduced in the scaled down version of the IDSP. In the first step, the concept will be applied only to the VRF by implementing two different structures of it. In the second step, it will be extended around the vector IS which depends on the structure of the VRF and in the final step it would be implemented for the whole compute cluster that contains the vector IS and it will stop before the RSN network.

3.5 Introduce the TDM concept inside the vector register file

In this section, the first step of the implementation is explained. The TDM concept has to be applied first to the VRF and then to the rest parts of the compute cluster. For this reason, two different implementations of the VRF will be presented.

3.5.1 Two register files of size 8x256 bit

In the first step, the TDM concept will be implemented inside the VRF of the compute cluster which is connected to the vector IS. In the beginning, the size of the VRF, which contains two write ports and two read ports, was 8x512b, but for the purpose of the TDM concept, it will be implemented by two register files of size 8x256b. In the first register file, the most 256 significant bits of the data will be written while in the second register file, the 256 less significant bits will be stored. More specifically, in case of a write in the VRF, the first part of the data will be written in the first clock cycle in the first VRF and in the second clock cycle the second part will be written in the second register file, by using an extra signal which indicates in which part of the VRF the data will be written. In case of a reading from the register file, the data are read immediately by reading simultaneously from the two register files and by concatenating the data. Finally, in Figure 19, the write and read process that was described previously are depicted. Each part of the inputs is written in the specific VRF and the data are read with the same way.

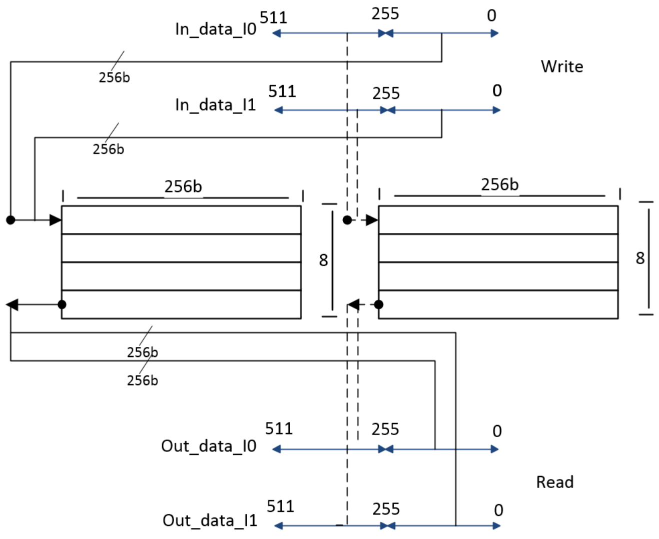


Figure 19: *The write and read process in the VRF of size 2x8x256b*

3.5.2 One register file of size 16x256 each

Another implementation of the VRF can be implemented by using one register file of size 16x256b. The way that the TDM concept can be applied is about the same, as the first part of the data is written in the first clock cycle, while in the second clock cycle the second part will be written. The big difference it that now there is only one register file with 16 entries, so the least significant bits of each register will be written in the first 8 entries of the VRF, while the most significant bits in the next 8 entries of the VRF. This will be implemented by placing the first 256 bit in the entry equal to the register id and the rest 256 bit in the entry equal to the register id plus 8 for consistency. When a read operation occurs, the first half of the data is read from the entry specified from the register id and the second half from the entry that is specified from the register id plus 8. The new structure of the VRF can be observed in Figure 20.

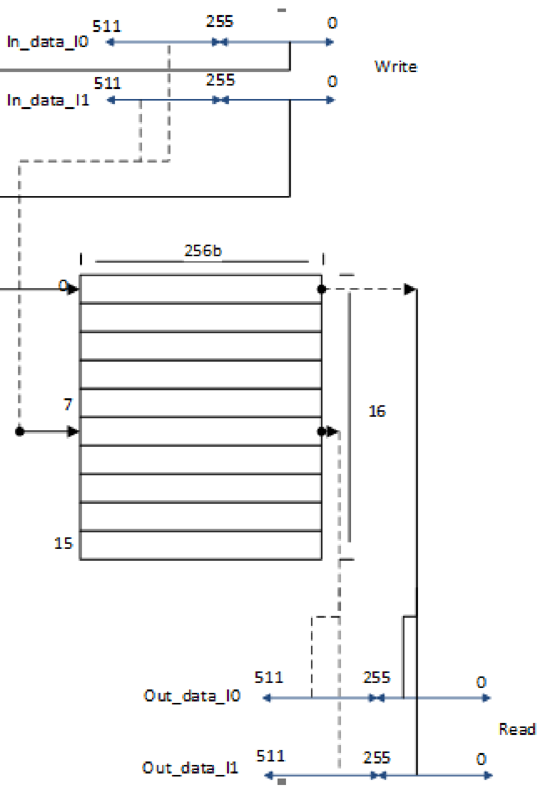


Figure 20: *The write and read process in the VRF of size 1x16x256b*

3.6 Introduce the TDM concept around the Issue Slot

The big advantage of using the TDM concept as a clock domain crossing technique is that the size of the vector IS in the compute cluster, that contains the functional units in which the computations are executed, will be reduced by half. This means that every component inside the IS will process data of size 256b, in comparison with the previous architecture that those components were processing data of size 512b.

Those components are the ICN network, all the execution units and the OCN network. Also, the size of the output data of the VRF has to change from 512b to 256b. In this way, in the first clock cycle the IS reads the first half of the data from the VRF and in the second clock cycle it reads the second half of the data.Accordingly, the FUs inside the IS will calculate the first part of the result in the first clock cycle and in second clock cycle the second part of the result. In order this to be implemented, extra multiplexers have to be added, so the appropriate part of the data to be processed in each clock cycle.

As it has already been referred, there are two different implementations of the VRF that has to be modified in order to be connected to the IS of size 256b. The VRF has two output ports of 512b each, so in order to be connected correctly with the vector IS, the VRF has to provide the first half of the data for both ports in the first clock cycle and the rest bits in the second clock cycle. To achieve this, a different implementation for each structure of the VRF has to be used. Although, the most important part of this implementation is the control signals in the ODEC, ICN and OCN components and inside the execution units of the vector IS. All the signals of these components which are related to the clock signal have to be modified. Firstly, the new connections between the VRF and the IS will be described and then the changes in the control signals that have to be made for the correct communication of the components of the vector IS will be explained.

3.6.1 Two vector register files of size 8x256 bit and 256 bit Issue Slot

In the new design the size of the IS will be 256b. This means that the VRF has to provide to the IS data of size 256b and for this reason, the current design has to be modified. In write process in the VRF, in the first clock cycle the first part of the data is written in the left sub-VRF while in the second clock cycle the second part is written in the right sub-VRF. On the other side, the data that was read from the VRF have to be concatenated before inserting the Bypass Network, because the Bypass Network remains of size 512b. According to the checks inside the Bypass Network, the appropriate data will be propagated to the vector IS. The input data in the vector IS has to be of size 256b because in this implementation the size of the vector IS has reduced to 256b, so it is able to process data of size 256b. For this reason, there are two multiplexers, for the two read ports, between the VRF and the vector IS, in order to send in the first clock cycle the first half of the data to the vector IS and in the second clock cycle the second half of the data.

The output of the IS has to be 512b because the data path between the compute cluster and the other two clusters is 512b. Accordingly, the first half of the output data of the vector IS will be stored for one clock cycle and then it will be concatenated with the second half of the data that it will be available in the next clock cycle and it will be propagated to the RSN network. The RSN network controls the data that are sent from one cluster to the other.

All the connections that were described previously are depicted in the Figure 21. The data that are read from the VRF is concatenated before inserting the Bypass Network, the multiplexer between the Bypass Network and the vector IS and the register that stores the first half of the data that will be concatenated with the second half of the data in the next clock cycle.

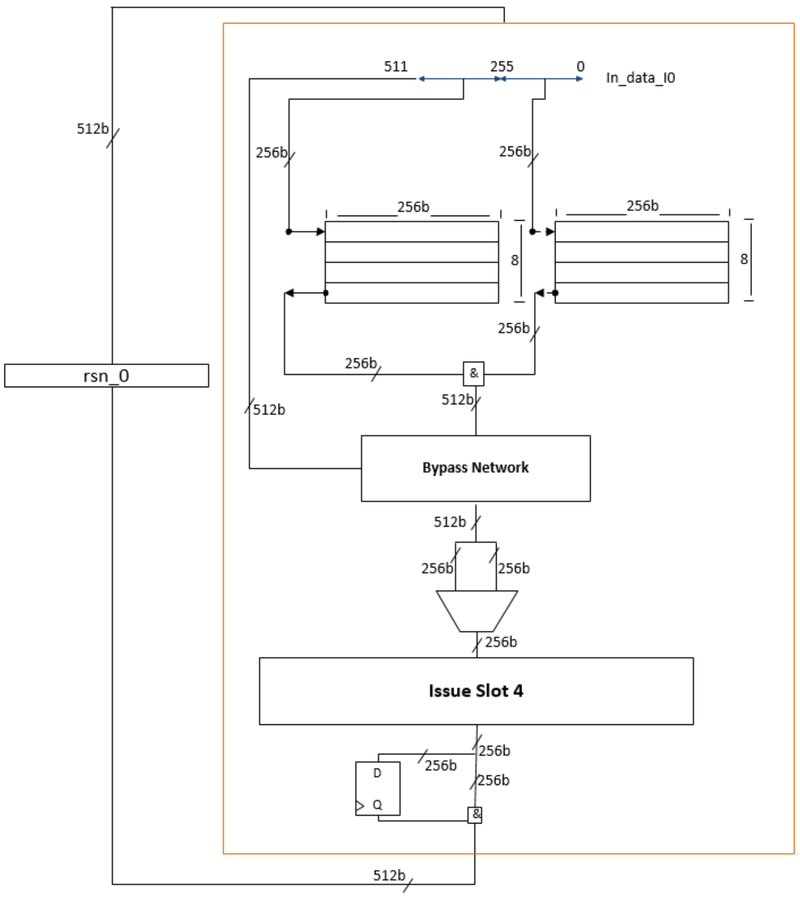


Figure 21: *Wire connections between the 2x8x256b VRF and the Issue Slot*

3.6.2 One register file of size 16x256 each and 256bit Issue Slot

In this structure, the first 256b is written in the first clock cycle in the entry of the VRF that the register id indicates while in the second clock cycle the data is stored in the entry register id plus 8 of the VRF. The read process is happening with a different way because the Bypass Network remains of size 512b. Firstly, the first half of the data is read from the VRF and is send to the bypass network extended with 0's in order to become 512b and then the second half of the data is sent to the bypass network extended with 0's. Between the Bypass Network and the vector IS there are two multiplexers each one for each read port, so in the first clock cycle the first half of the 512b will be sent to the IS and in the second clock cycle the second half will be sent.

The special part of this implementation is that the multiplexer before the bypass network has two inputs of size 512b. The first one is the data that are read from the VRF and the second one is the data that are going to be written in the VRF. If the bypass network is valid then the first half of data that are going to be written in the VRF has to be propagated and then, the second half of the data. If the bypass network is not valid, the correct part of the 512b data that is read from the VRF has to be propagated to the vector IS, because they have been extended with 0's, as has already been explained. When the two halves of the output data of the vector IS are available, they have to be concatenated and sent to the RSN network, which will send the data according to the control signals to the VRFs of other clusters or to its own VRF.

In this implementation, Figure 22, two multiplexers are added, one between the VRF and the Bypass Network and one between the Bypass Network and the vector IS. Finally, the first half of the output data has to wait the second half of the data that will be available in the next clock cycle in order to be concatenated.

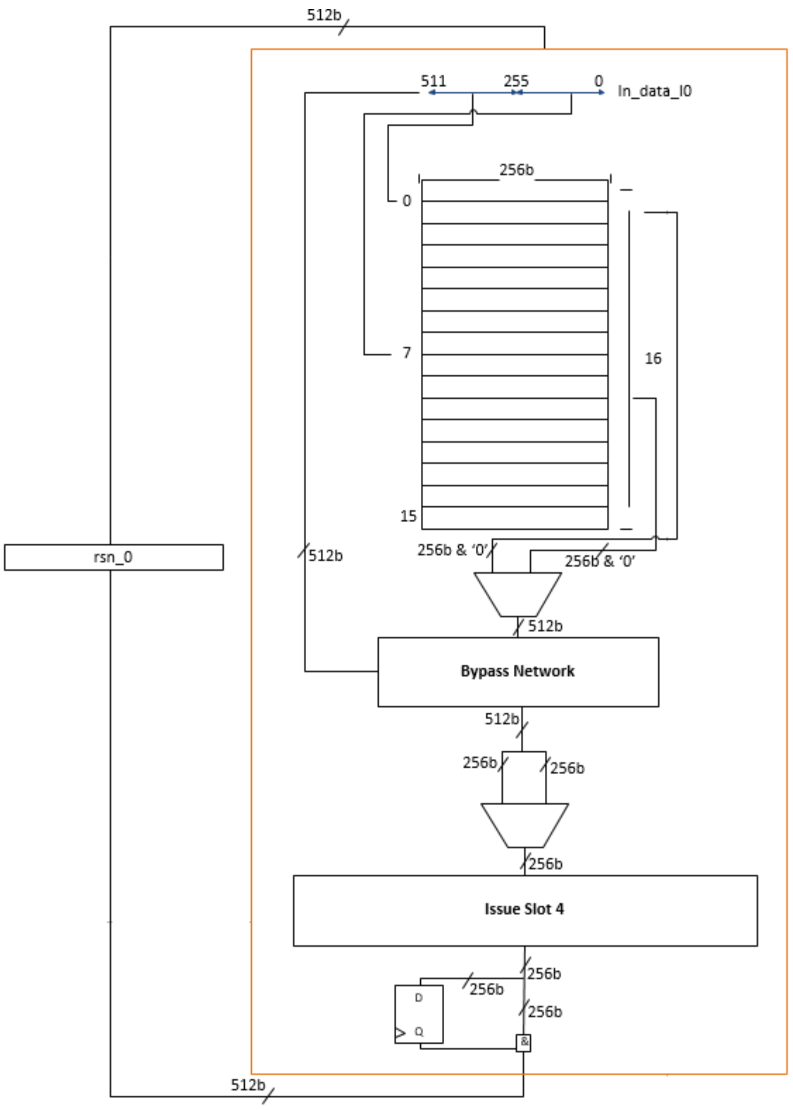


Figure 22: *Wire connections between the 1x16x256b VRF and the Issue Slot*

3.6.3 Changes on control signals of the Issue Slot

Control signals ensure the correct functionality inside the vector IS and the adjustment of these signals are fundamental. The control signals with dependencies on next clock cycle signals, need to be treated with special caution. Due to the double frequency, there are two positive edges of clock instead of one, that was previously. This has, as a result, all the signals that were changing on the next positive edge of the clock to be registered for one more clock cycle. To be easier to understand the whole changes in the control signals, it will be better to group them according to which component they belong and for a better understanding of the connections between those components, Figure 23 can be used.

- ODEC
  - out_fu_select
  - out_fu_select_next
  - out_optype
- ICN
  - argument_valid
  - argument_data
- EXU's
  - i_ctrl: output_enable
  - i_isolator: wire_input_mask
  - i_fu: output_data, output_valid
- OCN

ODEC

It is responsible for enabling the appropriate FU each clock cycle. According to the input signal in_opcode the appropriate out_fu_select_next signal becomes one in order to declare in which FU will be executed the instruction in the next clock cycle. The signal out_fu_select has to become one after one clock cycle than the out_fu_select_next . In our case, it has to be delayed one more clock cycle, because of the TDM concept. Another important signal that has to be delayed by one clock is the out_optype signal which is also an output of the ODEC component as the other two signals and it is an input to the EXUs.

ICN

This network is responsible for sending the input data of the vector IS to the appropriate EXU according to which argument is valid at each clock cycle. The valid signal is enabled one clock cycle before the data to be sent to the execution unit in order to indicate which arguments will be valid in the next clock cycle. So while the data in the original design were registered for one clock cycle, now they have to be registered for two clock cycles. Finally, the data will be sent to the appropriate EXU the same time with the out_optype and the out_fu_select signals that are sent from the ODEC component to the EXUs.

The most difficult part was to change with the proper way the signals which were responsible for transferring the data from the ICN network to the EXU. To achieve this except for the data that has to be registered, also the enable signal in the registers has to be delayed by one clock cycle. So in the first register the enable signal is delayed by one clock cycle and in the second register it is delayed for two clock cycle and this because otherwise, the first register will propagate the new value and not the second half of the previous value.

EXU

As has already been mentioned each execution unit consists of one controller, one isolator and the functional unit in which the computations are executed. In the picture below the architecture of each EXU is depicted.

The controller component has as an input the in_operation_valid signal that depends on signal fu_select signal and according to this produces an enable signal that afterward it is used together with the valid signal that the functional unit to indicate that the output of the execution unit is valid.

In the isolator component, the wire_input_mask signal, which depends on the signals in_input_valid_next and in_operation_valid_next, has to be registered for one more clock cycle, in order to propagate the correct data in the correct clock cycle to the FU. Considering that the signals with the suffix "next" indicate which input will be valid in the next clock cycle, in our case all the signals that depend on them have to be delayed for 2 clock cycles, because in one clock period of the original clock now there are two positive edges.

Furthermore, the functional unit component takes as inputs the operation type and the input data that the isolator propagates to it and it produces the output_data and an output_valid signal. In particular, each EXU calculates in the first clock cycle the first half of the output data and in the second half the second half of the output data and it propagates them together with the valid signal to the OCN component.

OCN

Finally, the OCN component has as inputs the valid signals, that the EXU's activate when their output data is ready, and the output data of the EXU's. When a valid signal becomes '1', the specific output data are propagated to the output of the vector IS.

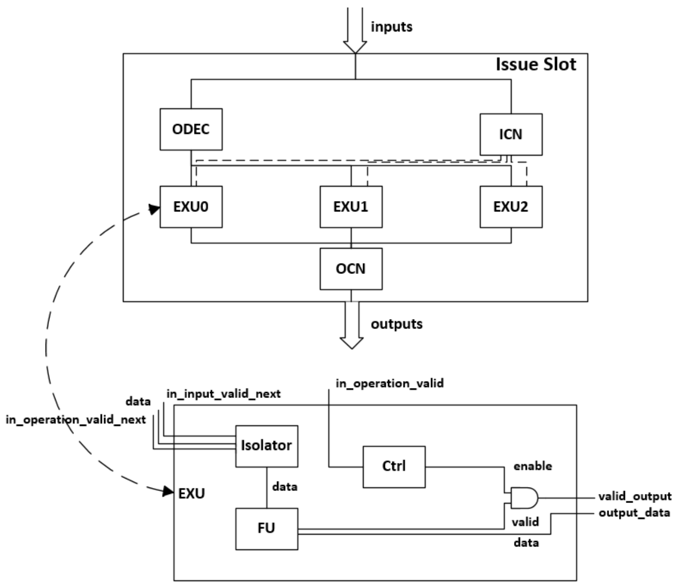


Figure 23: *EXUs' architecture*

3.7 Final Architecture

In the final implementation, the aim is to reduce as much as the architecture allows us the number of interconnections in the IDSP core. The solution is to reduce the number of input bits and output bits in the vector IS of the compute cluster because in the original architecture the RSN provides data of size 512b to the VRF and to the vector IS of the compute cluster.

More specifically, the RSN network, which is responsible for the input data that is sent in the compute cluster, in the final architecture it has to provide the compute cluster with data of size 256b. So the whole structure of the architecture has to change again. The input and output ports of the VRF, as well as, the output port of the vector IS have to become 256b. In this design, the concatenation of the first half of the data with the other half of it, it is done outside the vector IS and before the data is sent to the RSN network. Furthermore, a multiplexer has to be used between the RSN and the compute cluster, so in the first clock cycle the first half of the data to be sent to the compute cluster and in the second clock cycle the other half of the data. Again, two different designs will be implemented, because of the two different implementations of the VRF. Another important part of these changes is that the Bypass Network will be reduced from 512b to 256b, because now the data that will be written to the VRF it will be of size 256b and not 512b, so there is no need of a Bypass Network of size 512b. The two different implementations are presented in Figures 24, 25.

For the first structure with the VRF of 2x8x256 bit, the design has to change as follows. A demultiplexer will be needed for each write port in order to redirect the 256b data to the correct sub-register file. In the first clock cycle, the 256b will be written in the left sub-VRF and in the second clock cycle in the right sub-VRF. Also, one multiplexer will be needed when the data is read from the two sub-VRFs and is send to the Bypass Network. Furthermore, the Bypass Network is able to provide 256b data to the vector IS, so there is no need of a multiplexer between the Bypass Network and the vector IS.

For the second structure with the VRF of 1x16x256 bit, an extra multiplexer before the VRF, for the data that will be written in the VRF is not needed. The multiplexing is in the lowest hierarchical level of the VRF implementation and it is done between the entries that the data is going to be written. Furthermore, the 256b that are going to be written to the VRF, are provided to the new Bypass Network. The same happens also in the read process, the multiplexing is inside the VRF and provides the Bypass Network with data of 256b size. Accordingly, the vector IS process the data of 256b size and redirects the results outside of it.

This is the maximum reduction that can be achieved in terms of interconnections. To summarize the whole vector IS in the compute cluster and its interconnections were reduced from 512b to 256b, which leads to a big reduction in terms of area and interconnections.

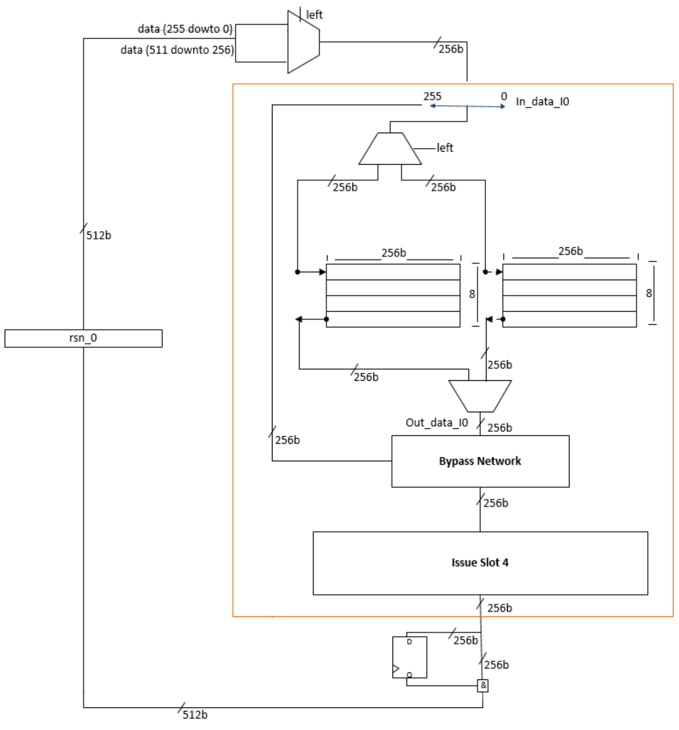


Figure 24: *Final architecture with 2x8x256bit VRF*

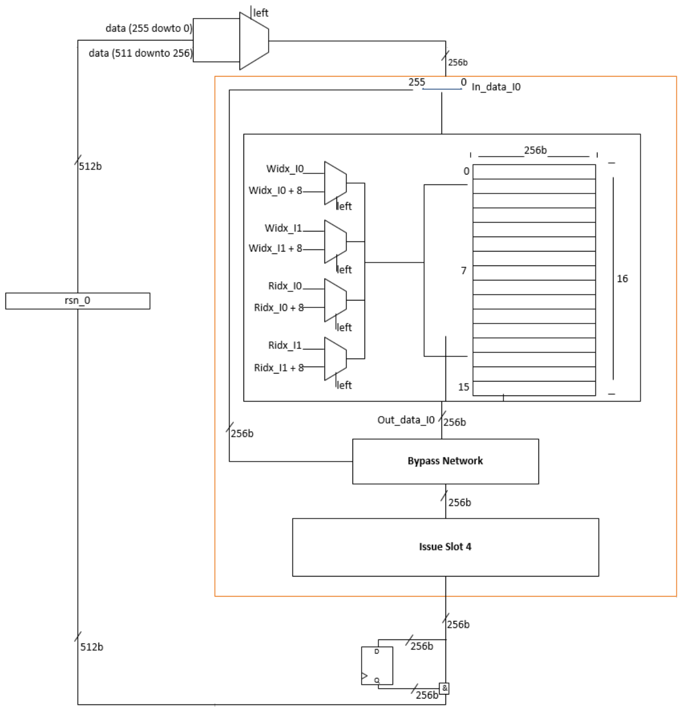


Figure 25: *Final architecture with 1x16x256bit VRF*

3.8 Latch based implementation of vector register file

In order to implement the VRF by using latches, the architecture has to change for the design that uses the VRF of 1x16x256b, because the changes that had to be applied in order to use the TDM concept in this structure were implemented in the lowest level of the design. In this occasion, the changes had to be made in the highest hierarchical component and replace the one that is already used by the latch based implementation of the VRF.

First important point is that in this structure the register id is 3 bits because there are 8 entries in the VRF. But now the register id has to become 4 bits because the VRF will have 16 entries. As it has been already referred in the previous implementations, the first part of the data has to be written in the first 8 entries and the second half in the next 8 entries. For this reason, the extension of the register id will be with ′0′ in the first clock cycle when the first half is written and it will be extended with ′1′ when the second half is written in the second half of the VRF. The same happens also when a read occurs. For this reason, two multiplexers have to be inserted into the design. Both of them to choose between ′0′ + register or ′1′ + register.

Latches and flip flops are the basic elements for storing information. More specifically, the flip flops are made from 2 latches. The basic advantage of using latch based implementation is that the area will be reduced further and that because each flip flop consists of 2 latches. Although, the main advantage is that synthesis flow provides the structural placement feature for the latch-based VRF implementation, Figure 26, that also helps in terms of timing. In case of VRF are being synthesized to flip flops, the flip flops are placed randomly, so extra buffers have to be used in order to achieve better timing. Taking into account that the implementation and the analysis of the latch based VRF are not straightforward, the use of this implementation depends on the size of the VRF. If the size is very small, it does not need to use latch based implementation, otherwise, it will be a better solution. However, on the other hand, if the capacity of the VRF is relatively high, hard macros(dedicated memories) are more area efficient implementation. The actual trade-off between flops, structural placed latches and hard macros are depending on the technology node and falls outside the scope of this work.

From all the various ram flavors, the gt_ram_sv_N_wr is the one that it is closer to the structure that the already existing VRF has. It is allowed to have from 0 to an infinite number of read ports and one or two write ports. In this occasion, two read ports and two write ports are needed. Furthermore, it takes one clock cycle for data to be available for reading.

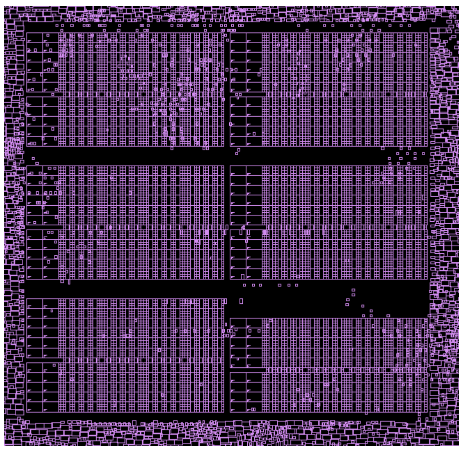
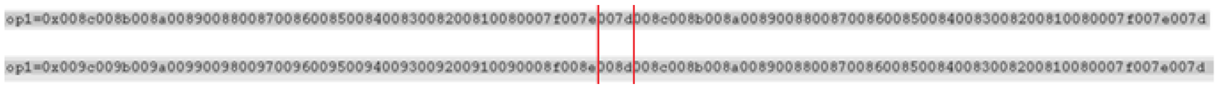


Figure 26: *The structural placement of a latch based VRF*

3.9 Limitations

After many experiments, the limitation of the new design was found. The limitation is that the TDM concept cannot be applied in all the functional units in the vector IS. This happens because when the TDM concept is applied in the vector IS, in the first clock cycle the first 256b are processed and in the second cycle the second half of the data is processed. This has as a result when there are instructions that use both parts of the data to calculate the final result, they do not have the other half of the data available.

For example there is an instruction that adds to each element the value of a 32b number by adding i, where i is the iteration. So the first half of the result is correct because i is increasing for each iteration, but in the second cycle, the second part of the result is wrong because the value of i is reset to 0. In the picture below, the first result is the correct result that RTL simulation has to print, but the result that actually the RTL simulation gives is the second one. With the red line, the two parts of the result are distinguished.



```
op1=0x008c008b008a00890088008700860085008400830082008100800007f007e007d008c008b008a00890088008700860085008400830082008100800007f007e007d

op1=0x009c009b009a00990098009700960095009400930092009100900008f008e008d008c008b008a00890088008700860085008400830082008100800007f007e007d
```

Figure 27: *RTL simulation results*

4 Results and Evaluation

In Section 4.1 the environment that was used in order to verify that the new different designs are functionally correct, will be explained. In the next section the synthesis flow that was used for the purpose of this thesis will be described and then the results about area, timing and power will be presented and evaluated.

4.1 Verification Environment

Functional verification verifies that a design meets the intent of its specification. In everyday terms, functional verification attempts to answer the question "Does this proposed design do what is intended?" and it is very difficult because of the sheer volume of possible test cases that exist in even a simple design.

In order to be able to verify that the new design is functional, an appropriate environment has to be set. For this purpose the IDSP tile will be used, and by using RTPG method [7] many different input sequences will be loaded on memories, program memory or instruction memory, that will be executed accordingly. To verify that the design meets its functional intent, four basic steps were needed:

- Compile files by using the ace environment
- Run Sched Simulation
- Run RTL Simulation
- Compare the outputs

More specifically, the ace environment was used to compile and run the tests that were generated by using a Perl script. With this script it is possible to set the number of test that will be generated and if those tests will be loaded from the program memory or the icache. Then by using the sched method the random generated C code will be compiled and simulated so a sched trace file will be exported. The RTL simulation produces a trace file too, that will be compared with the sched trace file, by using another Perl script. The output of this Perl script is a log file that includes the differences of those two files, such as, which operand's value is wrong and which functional unit produces this error.

In order to debug the design, the Synopsys DVE's waveform window was used. The output waveforms of the original circuit can be compared to them of the modified circuit, by using the vpd file that was generated from the RTL simulation. DVE is included in Synopsys VCS and it is an advanced, full-featured debug and visualization environment. Transaction-level debug is seamlessly integrated into DVE, allowing users to analyze and debug transactions in both list and waveform view. DVE further provides powerful capabilities for System Verilog testbench debug and it provides mixed HDL (System Verilog, VHDL and Verilog) and SystemC language debugging windows.

4.2 Verification Environment Modifications

As it has already been referred, inside the execution units in the vector IS, there is a probe component. This probe component is responsible for the probe file that contains the operands and the instructions that are executed in every clock cycle in the compute cluster. After the changes that had to be done so the TDM concept can be applicable on the IDSP core, the probe implementation has to change in order to be able to print with the correct format the data of 512b size. Then by using the Perl script it is easy to understand from the output log file if the new design is correct or not.

More specifically, the same concept of the TDM that was implemented in vector IS in the compute cluster has to be implemented in the SystemC code of probes, so the format of the data to be the appropriate one. The process is different for each functional unit and it depends on the way that each functional unit works. For example, the first execution unit is not so difficult to be modified because it is going to pass the data from the input port to the output and the size of those data is 32 bit, so there is no changes caused from the TDM concept. On the opposite, the second and third execution units do calculations by using two input data. The size of the first input is 256b and the second one is 32b. In order the probe to be able to print the output result with correct format, the first 256b of the result that are produced in the first clock cycle will be stored and in the next clock cycle will be concatenated with the rest 256b of the result. In some instructions this has to be done in both input ports and output ports and in some others the operand of 32b has to remain the same for both clock cycles.

4.3 Synthesis Flow

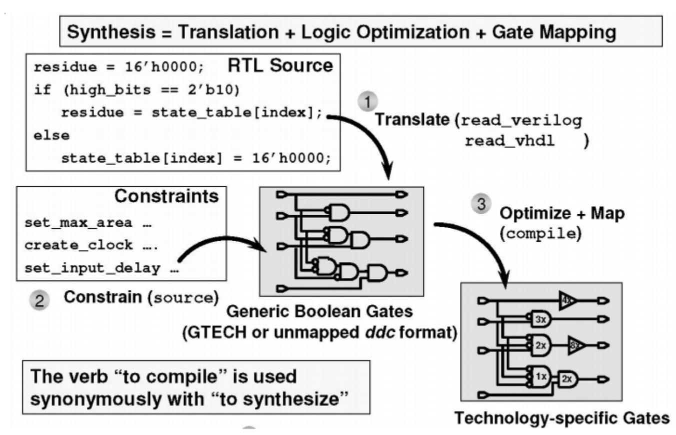
In this chapter the synthesis flow that was used will be explained step by step. All the different implementations will be evaluated with respect to area, power and timing and the results will be presented. In the first section, the setup information and procedure are presented. In the subsequent sections the different designs are compared in terms of different aspects, such as area, timing and power and in the last section the results are summarized.

4.3.1 Setup and procedure information

Synopsys Design Compiler will be used for the synthesis of the different designs. The output of synthesis is a VHDL/Verilog gate-level netlist and the design database. The first step is to specify the libraries that will be used.

The Design Compiler uses the target library to build a circuit. During mapping it selects specific cells from the target library taking into account timing, power, and area specific constraints. The target library specification should contain models for all cells, pads and custom cells that Design Compiler uses when is mapping the design. Furthermore, it uses the link library to resolve cell references. Default values and settings about the operating conditions (temperature, voltage and process variations), the wire load mode, timing ranges and net delay calculation can be found in these libraries. Also, the symbol library is used to generate the design schematic because it contains the schematic symbols for all the cells.

Then the synthesis flow begins, Figure 28. It starts by reading the design. Reading in an HDL design description consist of two tasks, analyzing and elaborating the description. With the analysis command, it reads the HDL files and checks them for errors and creates HDL objects in an HDL-independent intermediate format file. With the elaboration command, it translates the design into a technology independent design (GTECH) from the intermediate files. Next the constraints of the design has to be set. In this phase, area and timing constraints can be set. Setup-timing constraints can be specified, as well as constraints for input paths and output paths, environmental attributes, wire loads, input drivers, capacitive output loads. Finally, the design has to be optimized. The optimization step translates the HDL description into gate-level netlist using the cells available in the technology library. The optimization is done in several phases. In each optimization phase different optimization techniques are applied according to the design constraints.



```
residue = 16'h0000;
if (high_bits == 2'b10)
   residue = state_table[index];
else
   state_table[index] = 16'h0000;
```

```
set_max_area ...
create_clock ....
set_input_delay ...
```

Synthesis = Translation + Logic Optimization + Gate Mapping
RTL Source
1 Translate (read_verilog read_vhdl)
Constraints
2 Constrain (source)
Generic Boolean Gates (GTECH or unmapped ddc format)
3 Optimize + Map (compile)
Technology-specific Gates
The verb "to compile" is used synonymously with "to synthesize"

Figure 28: *Synthesis Flow*

The difference between the typical synthesis flow and the flow that was used is that in the Design Compiler the topographical option is used. This option uses physical constraints, including floorplan, options about utilization, more accurate wire load models and with this way provides to the IC physical guidance. The input files of the modified synthesis flow are presented in the figure below.

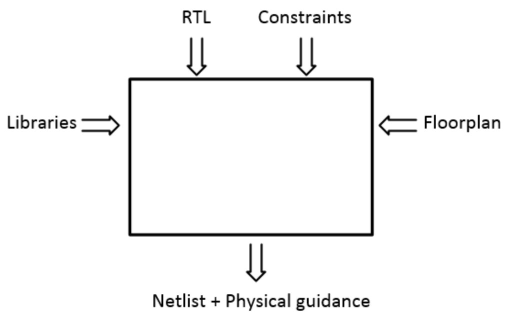


Figure 29: *Modified Synthesis Flow*

All the different designs that were described previously will be synthesized. More specifically, the designs that are going to be synthesized are the reference core, the designs with the maximum reduction in terms of area and the designs with the latch based implementation of the VRF. At the end of the synthesis flow, reports about timing, area and power are generated and were used for the evaluation of the designs. In the table below the different frequencies that were used in the synthesis flow are reported.

Table 1: *Frequencies*

Clk_I0	Clk_x2
200MHz	400MHz
250MHz	500MHz
275MHz	550MHz
300MHz	600MHz
350MHz	700MHz

4.4 Area Comparison

The main aim of this thesis was, by using the TDM concept as a clock domain crossing technique, to reduce the area that IDSP occupies and its wire length. The total netlenght of the nets in a design correlates well with the average congestion. Therefore, minimizing the netlength is often consistent with the goal of improving the congestion. Furthermore, the fact that in the specific synthesis flow the floorplan is loaded give us more accurate results because the netlength metric can be somewhat inaccurate if the placement changes.

For this reason, the TDM concept was used and after many changes in the basic architecture, two final designs which contain a compute cluster with a vector IS of size 256b and the maximum reduction in terms of wires were implemented. These two final architectures, except for the different VRF structure that they have, the VRFs have been implemented in two different ways. The first one is by using flip flops and the second implementation is latch based.

In terms of timing, the designs will be synthesized for the frequencies that have already been referred in the synthesis flow section. One important note is that in order to achieve better timing, the functional units that are not able to work on the TDM concept have not to be considered in the synthesis flow. For this reason, those functional units have to be set as false paths in the synthesis script in order not to be considered in timing calculations. Moreover,it was observed that the timing results are better when the VRF is implemented by latches, because of the structural placement that the synthesis flow enables for them.

The results in terms of gate count and wire length are presented in the graphs below.

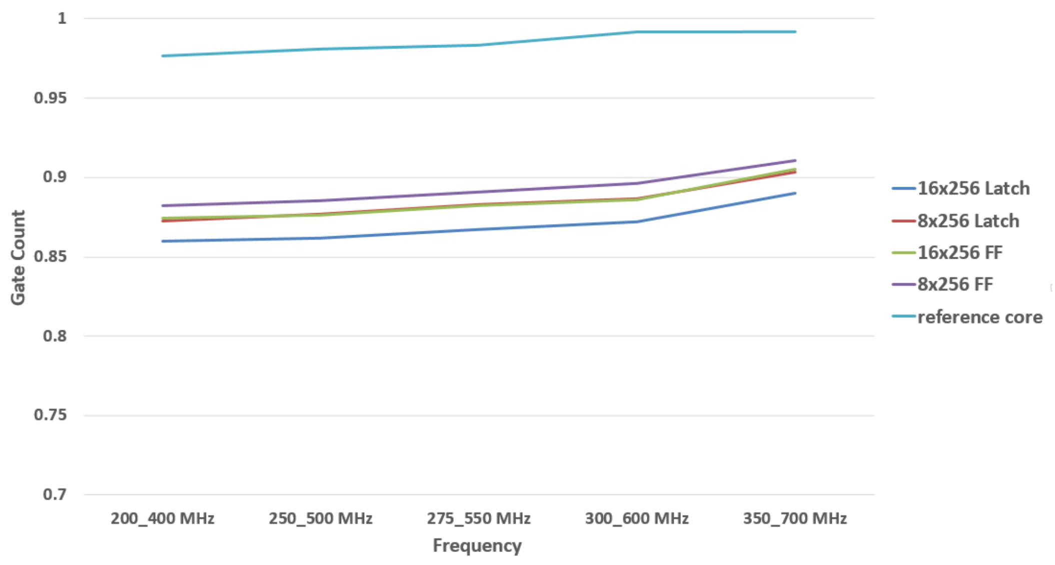


Figure 30: *Gate Count of the IDSP*

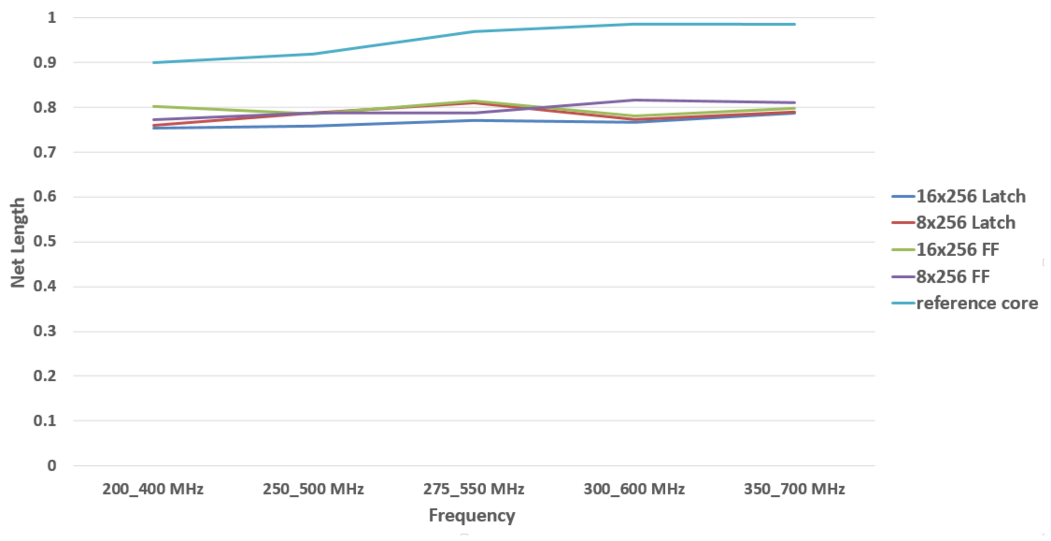


Figure 31: *Net Length of the IDSP*

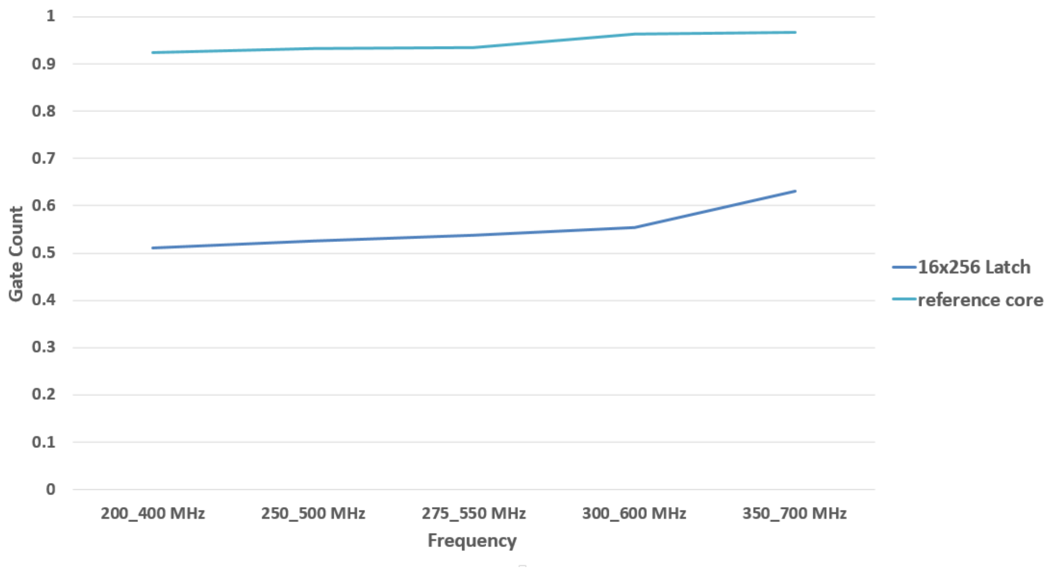


Figure 32: *Gate Count of the Vector Issue Slot*

The most important of the results which is also the goal of this project is the area reduction. It is obvious from the Figure 32 that there is a huge reduction in area in the vector IS that the TDM concept was applied. For this comparison, the reference core and the final architecture with a latch based implementation of the VRF were used. The reduction is about 42%, in terms of gate count, from the original design. It could be better, but in order for the second clock domain to achieve the 700MHz, more buffers for better timing results have to be used and for this reason, in 350/700MHz the reduction is about 34%.

From the Figure 30, the gradual reduction in terms of area is observed. Firstly, the area is reduced by using the final architecture with the entire vector IS to be connected to the other clusters with a data path of 256b instead of 512b and the reduction continues with the use of the latch based implementation of the VRFs. Also, another important note is that, from the two different structures of the design the best one, as it supposed to be, is the one that has one VRF of size 16x256b, because the multiplexing and demultiplexing of the signals is easier between the VRF and the Bypass Network, as well as, between the Bypass Network and the vector IS. In the total IDSP the improvement in terms of gate count is 13%, but in our occasion, the IDSP has only one vector IS. The normal IDSP contains 6 vector ISs and by using the synthesis results about the gate count of its vector IS, the entire improvement is approximately 30%.

Moreover, the reduction in terms of netlength,as it is observed in Figure 31 is about 20% for the entire IDSP. This is logical due to the fact that the reduction in terms of wires was applied only in the vector IS of the compute cluster, which occupies at about 17% of the entire design. Finally, the combination of 20% less wiring with 13% less area has, as a result, the reduction in wiring congestion.

4.5 Power Comparison

The strict limitation on power dissipation in portable electronics applications such as smartphones and tablet computers must be met by the VLSI chip designer while still meeting the computational requirements. Reducing the total power consumption in such systems is important since it is desirable to maximize the run time with minimum requirements on size, battery life and weight allocated to batteries.

There are several factors that contribute to the power consumption of a circuit. They include dynamic power consumption and power loss due to transistor leakage currents. By summing up this two powers, the dynamic and leakage power, the total power can be found.

$$P_{total} = P_{dynamic} + P_{leakage}$$

Dynamic power consumption depends on the activity of the logic gates in the design. When the logic gates toggle, energy is flowing as the capacitors inside them charged and discharged. Glitches are also another factor that dissipate power because are temporary changes in the value of the output and are caused due to the skew in the input signals of a gate. Accounts for 15%-20% of the global power. More specifically, dynamic power consumption is approximately proportional to the frequency that the design operates and to the square of the voltage that is used.

$$P_{dynamic} = C * V^2 * f$$

Where C is the capacitance, f is the frequency and V is the voltage.

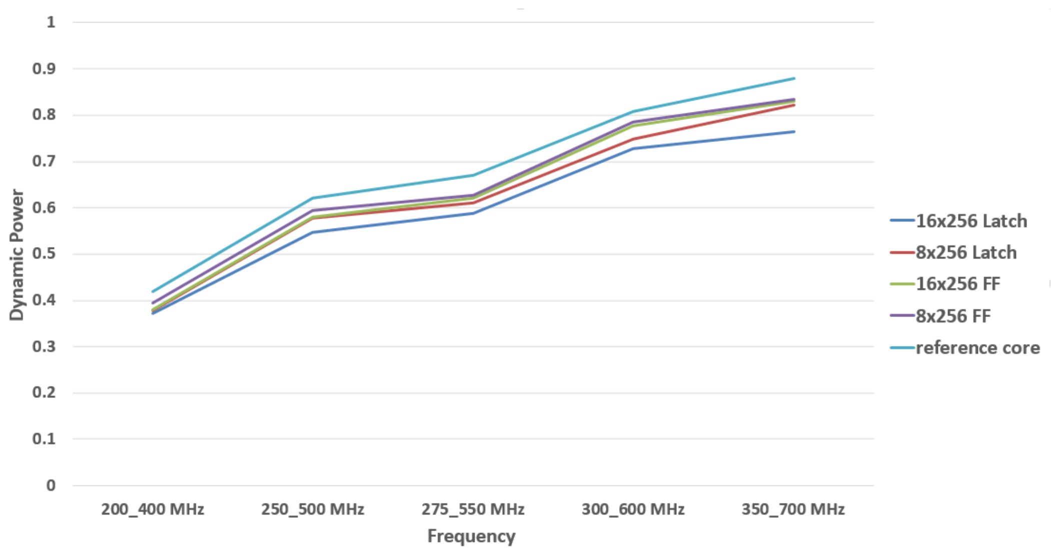


Figure 33: *Dynamic Power Consumption*

In the Figure above, the reduction in terms of dynamic power consumption is depicted. An average of 11% reduction is obtained in dynamic power consumption. Again the designs with the latch based implementation of the VRF have slightly better results than the designs with the FF implementation of the VRF.

Small amounts of currents are always flowing between the differently doped parts of the transistor and their magnitude depends on the state of the transistor, its dimensions and temperature. These currents are responsible for the power consumption due to transistor leakage currents. The leakage power $P_{leakage} = V * I$ is proportional to the leakage current which depends on the gate count. So according to the Figure 34 below and the theory it is normal that the leakage power is less in the new designs than the reference core, approximately 15%. Moreover, the leakage current increases when the frequency increases, because in higher frequencies the gate count is more, as has already been explained in the section of area comparisons.

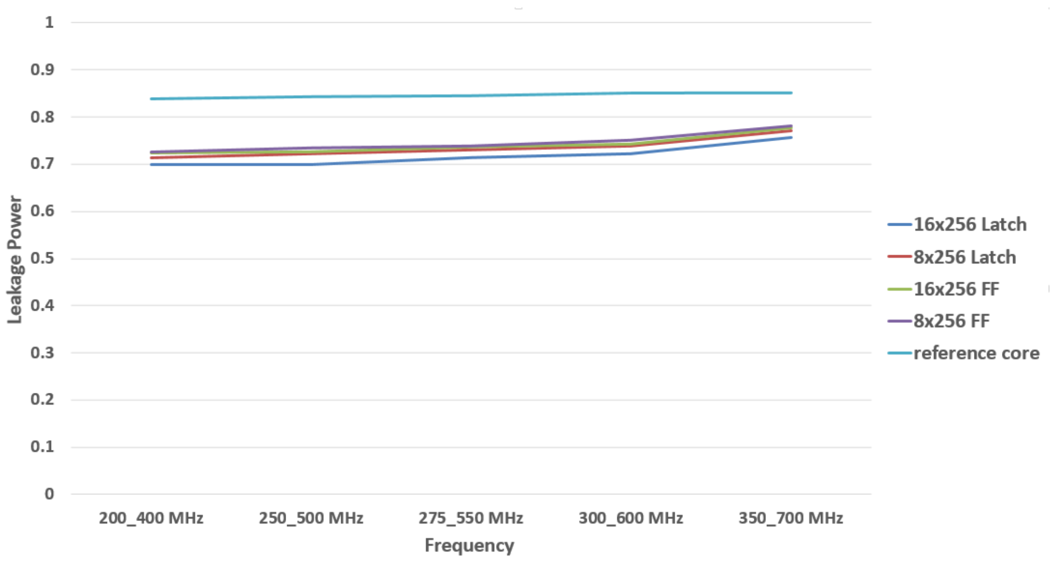


Figure 34: *Leakage Power*

5 Conclusion and Future Work

In conclusion, the TDM concept was applied successfully in the IDSP architecture and finally four different implementations of the vector IS in the compute cluster were proposed. In all of them the maximum reduction in terms of area and wires was achieved and, for better results in terms of area, the latch based implementation of the VRF was used.

In the modified architecture of the IDSP with one vector IS the area reduction was at about 13% and the wire length reduced by 20%. Firstly, this means that in terms of wiring congestion and routability the design is even better and that in a real IDSP that contains more than one vector ISs the area reduction is approximately 30%. Furthermore, the dynamic power consumption, as well as the leakage power, were reduced by 15% and 11%, respectively.

The future work that can be recommended is that those functional units that are not able to work under TDM concept have to be implemented differently. If this is not possible, then the functional units that the TDM concept is applicable must be grouped in one vector IS and those functional units that the TDM concept is not applicable to be grouped in another vector IS. A balance in terms of area and performance in vector ISs has to be found when a combination of different vector ISs implementation exists in the same architecture.

In terms of area and wires, the next goal could be to reduce the wires in the RSN network. The RSN network is the one that contains the biggest amount of wires because it is responsible for the interconnections between the different clusters in the IDSP architecture. The current implementation has stopped before the RSN network, because of the way that it is currently implemented. If there is any other way that the RSN network can be implemented, the gain in terms of wires and area will be big.

References

[1] `http://verifxn.blogspot.nl/`.

[2] `https://filebox.ece.vt.edu/~athanas/4514/ledadoc/html/pol_cdc.html`.

[3] `www.edn.com/electronics-blogs/day-in-the-life-of-a-chip-designer/4435339/Synchronizer-techniques-for-multi-clock-domain-SoCs`.

[4] `http://flickeringtubelight.net/blog/wp-content/uploads/2006/04/setupAndHoldViolations.pdf`.

[5] `https://en.wikipedia.org/wiki/Time-division_multiplexing`.

[6] `http://efxkits.com/blog/what-is-multiplexer-and-types/`.

[7] Michael Bushnell and Vishwani D Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science & Business Media, 2000.

[8] Ramesh Chidambaram, Rene van Leuken, Marc Quax, Ingolf Held, and Jos Huisken. A multi-standard fft processor for wireless system-on-chip implementations. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE, 2006.

[9] Clifford E Cummings. Simulation and synthesis techniques for asynchronous fifo design. In *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, 2002.

[10] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[11] Shen-Fu Hsiao, Chi-Guang Lin, Po-Han Wu, and Chia-Sheng Wen. Asynchronous ahb bus interface designs in a multiple-clock-domain graphics system. In *Circuits and Systems (APCCAS), 2012 IEEE Asia Pacific Conference on*, pages 408–411. IEEE, 2012.

[12] Harm Peters, Ramanathan Sethuraman, Aleksandar Beric, Patrick Meuwissen, Srinivasan Balakrishnan, Carlos A Alba Pinto, Wido Kruijtzer, Fabian Ernst, Ghiath Alkadi, Jef Van Meerbergen, et al. Application specific instruction-set processor template for motion estimation in video applications. *IEEE transactions on circuits and systems for video technology*, 15(4):508–527, 2005.

[13] Visvesh Sathe, Conrad Ziesler, Marios Papaefihymiou, S Kinfl, and Stephen Kosonocky. A synchronous interface for socs with multiple clock domains. In *SOC Conference, 2004. Proceedings. IEEE International*, pages 173–174. IEEE, 2004.