Evaluation of the Achronix picoPIPE™ Architecture in High Performance Applications

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan vid Linköpings universitet av

Christoffer Peters

LiTH-ISY-EX--12/4645--SE

Linköping 2012
Evaluation of the Achronix picoPIPE\textsuperscript{TM} Architecture in High Performance Applications

Examensarbete utfört i Elektroniksystem vid Tekniska högskolan i Linköping av

Christoffer Peters

LiTH-ISY-EX--12/4645--SE

Handledare: Mario Garrido
isy, Linköpings universitet
Gunnar Stjernberg
Synective Labs AB

Examinator: Oscar Gustafsson
isy, Linköpings universitet

Linköping, 30 November, 2012
Evaluation of the Achronix picoPIPE™ Architecture in High Performance Applications

In this thesis the new Speedster HP FPGA from Achronix is analyzed. It makes use of a new type of interconnection technology called picoPIPE™. By using this new technology, Achronix claims that the FPGA can run at clock frequencies up to 1.5 GHz. Furthermore, they claim that circuits designed for other FPGAs should work on the Speedster HP after some adjustments. The purpose of this thesis is to study this new FPGA and test the claims that Achronix make about it.

This analysis is carried out in four steps. First an analysis of how the new interconnection technology works is given. Based on this analysis, a number of small test circuits are designed with the purpose of testing specific aspects of the new FPGA. To analyze circuit reusability an image filter designed by Synective Labs AB for a different FPGA architecture is adapted and evaluated on the Speedster HP. Lastly, an encryption circuit is designed from scratch. This is done in order to test what can be achieved on the Speedster HP when the designer is given full freedom.
Abstract

In this thesis the new Speedster HP FPGA from Achronix is analyzed. It makes use of a new type of interconnection technology called picoPIPE™. By using this new technology, Achronix claims that the FPGA can run at clock frequencies up to 1.5 GHz. Furthermore, they claim that circuits designed for other FPGAs should work on the Speedster HP after some adjustments. The purpose of this thesis is to study this new FPGA and test the claims that Achronix make about it.

This analysis is carried out in four steps. First an analysis of how the new interconnection technology works is given. Based on this analysis, a number of small test circuits are designed with the purpose of testing specific aspects of the new FPGA. To analyze circuit reusability an image filter designed by Synective Labs AB for a different FPGA architecture is adapted and evaluated on the Speedster HP. Lastly, an encryption circuit is designed from scratch. This is done in order to test what can be achieved on the Speedster HP when the designer is given full freedom.
I would like to start by dedicating this master thesis to my grandfather Torsten. His never-ending curiosity for new technology will always remain an inspiration to me in my engineering endeavors.

I would like to thank my two supervisors Gunnar Stjernberg and Mario Garrido for all their help during the work on this thesis. I would also like to thank Magnus Peterson at Synective Labs AB for giving me the opportunity to do this thesis. Furthermore, I would like to thank Achronix and Greg Martin for providing the tools and support needed to work with the Speedster HP FPGA.

I also want to thank all my friends, especially Oskar Holstensson, Ludvig Lindblom, Gustav Wallin and Gabriel Kulig, Jonathan Liss and Josef Larsson for making my time at the university very fun.

Last but not least, I would like to thank my fiancée Ariel, my mother Anne-Marie, my father Björn and my sister Emelie for all their love and support.
## Contents

1 Introduction .................................................................................. 3
   1.1 Background ............................................................................ 3
   1.2 Purpose .................................................................................... 3
   1.3 Outline ..................................................................................... 4
   1.4 Scope ....................................................................................... 5

2 Field Programmable Gate Arrays .................................................. 7
   2.1 General functionality and terminology ..................................... 7
   2.2 Virtex-6 .................................................................................. 8
   2.3 Speedster 22i HP ..................................................................... 9

3 Analysis of the picoPIPE fabric ................................................... 11
   3.1 The picoPIPE stage ............................................................... 11
   3.2 Interconnection using picoPIPE ............................................. 14
   3.3 Improvements and modifications ............................................ 15
   3.4 picoPIPE usage in FPGA ....................................................... 16
   3.5 Limitations with picoPIPE ..................................................... 18

4 Initial test designs ........................................................................ 21
   4.1 Test design and motivation .................................................... 21
       4.1.1 Distributed logic ............................................................. 21
       4.1.2 Multipliers ....................................................................... 22
       4.1.3 Simple filter structures ................................................... 22
       4.1.4 Resets .............................................................................. 23
       4.1.5 Loops .............................................................................. 23
   4.2 Methodology ............................................................................ 23
   4.3 Test circuit considerations ..................................................... 24
   4.4 Analysis of distributed logic .................................................. 25
   4.5 Analysis of multipliers ......................................................... 26
       4.5.1 The Speedster HP MACC block ....................................... 26
       4.5.2 The Virtex-6 DSP block .................................................. 27
       4.5.3 Multiplier experiments ................................................... 27
   4.6 Analysis of a simple filter ..................................................... 31
   4.7 Analysis of resets ................................................................... 33
   4.8 Finite state machines .......................................................... 36
5 Guidelines for hardware design on the Speedster HP

6 Median filter
   6.1 Algorithm description
   6.2 System description
      6.2.1 Top module
      6.2.2 FIFO
      6.2.3 Median filter
      6.2.4 Line buffers
      6.2.5 Bubblesort kernel
   6.3 Identified problems and tested solutions
      6.3.1 Adapting the RAM for Achronix FPGAs
      6.3.2 Analysis and redesign of the FIFO memories
      6.3.3 Redesigning the sorting kernel
      6.3.4 Redesigning the line buffers
      6.3.5 Redesigning the median filter block
   6.4 Conclusions

7 Data encryption standard
   7.1 Introduction to DES
   7.2 Implementations
      7.2.1 Direct implementation
      7.2.2 High-performance implementation
   7.3 Conclusions

8 Conclusions and future work
   8.1 Conclusions
   8.2 Future work

Bibliography
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>Achronix CAD Environment</td>
<td>17</td>
</tr>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
<td>9</td>
</tr>
<tr>
<td>ASC</td>
<td>Asynchronous-Synchronous Converter</td>
<td>16</td>
</tr>
<tr>
<td>CLB</td>
<td>Configurable Logic Block</td>
<td>9</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
<td>61</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processing</td>
<td>9</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In, First Out</td>
<td>10</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
<td>3</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
<td>36</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
<td>3</td>
</tr>
<tr>
<td>HLC</td>
<td>High Logic Cluster</td>
<td>9</td>
</tr>
<tr>
<td>LLC</td>
<td>Light Logic Cluster</td>
<td>9</td>
</tr>
<tr>
<td>LUT</td>
<td>Look-Up Tables</td>
<td>7</td>
</tr>
<tr>
<td>MACC</td>
<td>Multiply-and-Accumulate</td>
<td>9</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivative</td>
<td>41</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
<td>7</td>
</tr>
<tr>
<td>RLB</td>
<td>Reconfigurable Logic Block</td>
<td>9</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-Only Memory</td>
<td>7</td>
</tr>
<tr>
<td>SAC</td>
<td>Synchronous-Asynchronous Converter</td>
<td>16</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single Instruction Multiple Data</td>
<td>9</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
<td>23</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
<td>23</td>
</tr>
<tr>
<td>XP</td>
<td>Extra Pipelining</td>
<td>17</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Background

The Achronix company has developed a new technology called picoPIPE™ that they use in the core of their Speedster HP Field Programmable Gate Array (FPGA). By utilizing this new technology, they claim that they can achieve several times higher performance compared to conventional FPGAs from companies such as Xilinx and Altera [6]. They also claim that this new architecture is almost completely transparent to the designer, and that high performance can be achieved on their systems without having to do an extensive rewrite of the Hardware Description Language (HDL) code.

1.2 Purpose

The overall purpose that Synective Labs had with this thesis was to evaluate the new FPGA architecture that Achronix provide in order to find out if and in that case when they should use it. This has been divided into two main purposes. First the new architecture needs to be studied in order to explain how it works compared to a traditional FPGA architecture. Since the core technology differs much, the two types of FPGAs are not expected to behave in a similar way. To be able to analyze and explain these differences, a good understanding of both architectures is essential.

The second purpose is to evaluate the claims that Achronix makes. To do this, they have been summarized into three main questions:

1. What speed is achievable with the Speedster HP FPGA?
2. What is needed when designing a circuit to be able to achieve this speed?
3. What modifications are needed in a circuit designed for a traditional FPGA to make it work efficiently on the Speedster HP?
Using the first two questions as a starting point, several more specific ones have been formulated:

- Does the speed differ for different types of typical circuits?
- If so, what is the maximum speed for each type?
- What kind of design choices affect the performance?
- What is the impact of using the picoPIPE technology to automatically pipeline a circuit?
- What are the limitations?
- etc...

It is necessary to answer these questions so that a description of the practical behavior of the Speedster HP FPGA can be given and a list of programming guidelines can be compiled.

The purpose of the third main question is to determine how much previously written HDL code can be reused when working with the Speedster HP. This is a very important question because if Achronix claims are true, the performance of a circuit can be increased by simply replacing a traditional FPGA with the Speedster HP. On the other hand, if the code needs to be rewritten to get good performance on the Speedster HP, then that must be taken into account when deciding on whether or not to use this FPGA in a project.

1.3 Outline

The work in this thesis has been divided into a theoretical part and a practical part.

The theoretical part consists of chapters 2 and 3 where the goal is to fulfill the first purpose of this thesis. Data sheets, patents, white papers and other documents have been studied to get a detailed understanding of a traditional FPGA as well as the Speedster HP. A Xilinx Virtex-6 FPGA has been used to represent a currently available state-of-the-art traditional FPGA. It was chosen because it is designed for high performance and high bandwidth [4], the same target as Achronix has with Speedster HP. Apart from the logic resources in the two FPGAs, a specifically detailed and thorough study is done on the picoPIPE technology in chapter 3. This resulted in an explanation of how data is processed inside the asynchronous core of the Speedster HP FPGA.

For the practical part, the goal is to answer the questions about the claims that Achronix make. In chapter 4 the questions are further elaborated so that each of them only covers a specific aspect. Then a number of test circuits have been designed. The purpose is to isolate a certain behavior of the FPGA, so that questions about it can be answered reliably. In all tests the results for the Speedster HP are compared to those for the Virtex-6 to find in what way its behavior differ from a traditional FPGA. Using the conclusions from the tests as well as the
knowledge gathered in the theoretical work, a list of programming guidelines is produced in chapter 5. It contains recommendations on what to do and what to avoid in order to achieve maximum performance when designing circuits for the Speedster HP.

Next, in chapter 6 a larger high performance circuit which had previously been designed for a traditional FPGA by Synective Labs AB is analyzed to find if anything needs to be modified to make it run fast on the Speedster HP FPGA. The main goal is to find design choices that cause problems for the picoPIPE technology and then redesign the circuit with help from the guidelines. This will give, for this particular circuit, an evaluation of to what extent Achronix claims of code reusability were true.

Lastly, in chapter 7 a second large high performance circuit is designed from scratch to give full freedom to adapt it to the behavior of the Speedster HP. The choice of circuit has been done in collaboration with Achronix to assure that it is one that they expected good performance from.

1.4 Scope

Doing a complete analysis of the performance of such a complex circuit as a modern FPGA is clearly not possible in the scope of a master thesis. Furthermore, the FPGA studied in this thesis uses new technology that first has to be studied and understood before an analysis of the FPGA can be done. For this reason, it is important to set up a number of limitations for what should be covered. This also helps to focus the attention to the areas that are deemed most interesting.

Designing circuits for use in an FPGA is usually a trade-off between area (number of resources used) and performance. However, the main focus in all parts of this thesis has been on high performance, because that is what the Speedster HP FPGA was designed for.

The test circuits have also been designed with the picoPIPE technology in mind. They are either circuits that are expected to benefit from this technology and perform very well, or circuits that should cause problems and reveal the limitations of it. Furthermore, they test specific parts of the FPGA that are commonly used in high performance circuits.

For the analysis of larger circuits, two feedforward circuits are chosen because that is what the Speedster HP is intended to be used for.

Very little time has been spent on working with the settings in the tools used because that would have been too time consuming, and also would have shifted the focus away from the study of the core technology. For the same reason the code generators in each of the tools are not analyzed. They provide the possibility to generate code for components such as memories or multipliers by only setting a few parameters. They can be very useful when a very specific component is needed, but the code that they generate in not portable since it has been tailored for a certain FPGA.
Chapter 2

Field Programmable Gate Arrays

This chapter gives an introduction to both the conventional FPGA architecture and the Achronix picoPIPE architecture. Basic concepts such as logic blocks and interconnections are introduced and their function is explained.

2.1 General functionality and terminology

An FPGA is a circuit that can be programmed to carry out any logic function. The two most essential parts of an FPGA are the switching matrix and the logic blocks. The logic blocks consists of a number of Look-Up Tables (LUT), registers and multiplexers. It is also common that carry chain logic is added to speed up full adder implementations. A LUT normally behaves as an asynchronous Read-Only Memory (ROM) with a 4 to 6-bit address input and a 1-bit data output. It stores the truth table for the programmed boolean function. The output from the LUT can be synchronized by connecting it to the register, or kept asynchronous by bypassing the register. In figure 2.1 a simplified logic block can be seen.

To implement a logic function, it is partitioned into small enough boolean functions that can be programmed into the LUTs. The logic blocks are then connected through the switching matrix to form the complete logic function.

Certain functions are difficult to implement efficiently using only general logic blocks. Therefore, hard blocks that can only carry out specific functions are also included in an FPGA. A multiplier is one example of a very common hard block. The hard blocks are connected to the switching matrix and used in the same way as the logic blocks.

There are also memory blocks in an FPGA. Dual port block Random Access Memory (RAM) circuits with a few kilobytes of storage each are found in almost any FPGA. They can be used to store data in a much more efficient manner than using the registers in the logic blocks. In certain designs, a LUT can be configured as a very small RAM.
What determines the data throughput in a traditional FPGA is the clock of the system. All registers in a clock domain are controlled by the same global clock. When several logic blocks are connected to produce a more complex logical function, the clock frequency is limited by the critical path, i.e. the path between any two registers that has the highest propagation delay. In a synchronous design, all registers must be clocked at the same speed for the circuit to function properly. An example is given in figure 2.2. Assuming that all LUTs have the same delay, Data 1 passes through the critical path and the propagation delay from Register 2 to Register 3 determines the clock speed. Data 0 has a shorter path and could theoretically be clocked through faster, but since it shares the clock with Data 1, they have the same throughput. To achieve a higher throughput the designer needs to make the critical path as short as possible.

2.2 Virtex-6

In this master thesis, the Xilinx Virtex-6 XC6VLX75T-1-FF484 FPGA is used to represent a traditional high performance architecture. In Xilinx terminology, logic
blocks are called Configurable Logic Block (CLB). In Virtex-6, a CLB contains slices, and each slice contains LUTs, carry chain logic, multiplexers and registers. There are two different types of slices. In SLICEL the LUTs can only be used to implement a logic function. In SLICEM the LUTs can also be used as small RAMs [7].

The multiplier hard blocks have been replaced by Digital Signal Processing (DSP) blocks in Virtex-6. These DSP blocks contain a 18x25 multiplier, but can also perform several other functions [3]. It has a preadder placed before the multiplier and an Arithmetic Logic Unit (ALU) with an accumulator register placed after the multiplier. Apart from implementing a Multiply-and-Accumulate (MACC), the ALU is also capable of Single Instruction Multiple Data (SIMD) addition and logic functions with up to 4 operands. The MACC functionality is especially useful when the FPGA is used for signal processing. However, due to the complex operation, it is split up into a four stage pipeline. The number of pipeline stages that are used can be configured, but for maximum performance multiplication 3 stages should be used. The DSP blocks can be cascaded to increase the data width.

Each block RAM in Virtex-6 is dual port [2], meaning that two read or write operations can be done at the same time. It can be split into two independent memories of half the size. It can also be configured into one memory of double the size, but then it must have only one read-only and one write-only port. Furthermore, two neighboring block RAMs can be combined into one memory.

<table>
<thead>
<tr>
<th>Component</th>
<th>Speedster HP</th>
<th>Virtex-6</th>
</tr>
</thead>
</table>
| Logic     | LLC: 2 LUTs with registers  
           | HLC: 2 LUTs with a carry chain adder and registers | SLICEL: 4 LUTs with carry chain logic, multiplexers and registers  
           | SLICEM: Same as SLICEL, but LUTs can be used as RAM |
| Multiplier| 28x28 MACC   | 18x25 DSP |
| Memory    | Dual-port BRAM and single-port LRAM | Dual-port BRAM |

Table 2.1: A comparison of the components in the two FPGAs.

2.3 Speedster 22i HP

The Speedster 22i HP360 is the circuit that will be used to evaluate the picoPIPE architecture from Achronix. In Achronix terminology, logic blocks are called Reconfigurable Logic Block (RLB). Each RLB contains LUTs and registers, which are organized into Light Logic Cluster (LLC) and High Logic Cluster (HLC) [5]. An LLC is made up of LUTs and registers. A HLC is an LLC expanded with an adder and a carry chain.

Instead of full DSP blocks, Speedster HP has MACC blocks. These blocks contain a 28x28 multiplier, an adder and an accumulator register [5]. If only
multiplication is needed, the adder and accumulator register can be bypassed. The MACC block has a 3-stage configurable pipeline.

There are two types of RAM: block RAM and logic RAM [5]. The block RAM is dual port. It has a built-in First In, First Out (FIFO) controller and configurable geometry. The logic RAM has one read and one write port that can be used as a simple dual port or a single port memory.
Chapter 3

Analysis of the picoPIPE fabric

As explained in the previous chapter, the critical path is what limits the clock speed of a design. In a traditional FPGA a long critical path is typically formed when there is a long combinational path. When two points very far away from each other need to be connected it can cause a routing delay. The traditional solution is to manually pipeline a long combinational path into several shorter paths by inserting registers into the combinational path. The same thing can be done if a routing delay causes problems, and is then referred to as geometrical pipelining. These solutions will enable higher clock speeds, but needs to be done manually and will alter the logic function of the design. All registers in this clock domain must also be clocked at the same speed.

3.1 The picoPIPE stage

In the picoPIPE fabric, data is handled differently. Special pipeline stages called picoPIPE are built directly into the interconnection fabric of the FPGA. There is no global clock for the core of the FPGA. Instead, there is a local handshaking protocol between the individual picoPIPEs [1].

![C-element symbol](image)

Figure 3.1: C-element symbol.

The handshaking protocol is controlled in each picoPIPE by a C-element [16]. It is an asynchronous circuit with an internal feedback loop that can store its state.
The C-element symbol is shown in figure 3.1 and the schematic is shown in figure 3.2. From the schematic, the behavior in table 3.1 can be derived. The output signal will only change when both input signals are equal. Otherwise, the current output signal will remain unchanged.

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: Logic behavior of a C-element.

A single picoPIPE stage can be seen in figure 3.3. Note that the C-element is modified so that the input for the Ack in signal is inverted. The modified C-element controls the state of the stage, and the latch is used to store the actual data that is being transferred. In table 3.2 the relationship between the input and output signals is listed.

The transfer of data through this stage is done with a 4-phase handshaking protocol [16]. Table 3.3 contains a step-by-step description of an example transfer.
3.1 The picoPIPE stage

![Diagram of a picoPIPE stage](image)

Figure 3.3: A picoPIPE stage.

<table>
<thead>
<tr>
<th>Step</th>
<th>Ready in</th>
<th>Ack in</th>
<th>Ready out</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Initial state</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Data ready at input</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Latch closed</td>
</tr>
<tr>
<td>3 or 4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Ack from next stage</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Ack from previous stage</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Latch opened</td>
</tr>
</tbody>
</table>

Table 3.3: Data transfer cycle in a 4-phase picoPIPE.

In the initial state, the latch is open, so the Ready out signal is 0.

Step 1 of the transfer is that the previous stage signals that data is ready at the input of the latch by setting the Ready in signal to 1. This triggers a change in the Ready out signal from 0 to 1 according to the behavior in table 3.2, which in turn leads to three things that make up step 2 of the transfer.

First, the inverted Ready out signal is used to control the latch. When it makes a transition from 1 to 0 it closes the latch. Secondly, the Ready out signal is used as Ready in in the next stage, so it signals that data is now ready to be sent to the next stage. Thirdly, the Ready out signal is also the Ack out signal which is connected to the previous stage, so at the same time as the latch is closed it acknowledges that data has been received.

Now the data in the latch is valid, and step 3 and 4 is to get acknowledge from the two neighboring picoPIPE stages. The previous stage sets the Ready in signal to 0 as a reaction to the Ack out signal. The next stage acknowledges that data has been latched in it by setting Ack in to 1. These two steps can happen in any order, but both events need to occur before the transfer can move on to step 5.

In step 5 both neighboring stages have acknowledged the transfer, setting the Ready in signal to 0 and the Ack in signal to 1. Again, according to the behavior
in table 3.2, this triggers a change in the \textit{Ready out} signal from 1 to 0, leading to step 6 in the transfer.

The latch is opened again in step 6 because of the change in the \textit{Ready out} signal. This also signals to the next stage that the data at the output of the latch is no longer valid. Since that stage is also going through the same transfer cycle, but with a delay compared to this stage, it will set \textit{Ack in} to 0 when it reaches step 6 in its transfer. This will put this stage back at step 0 again, and the whole cycle can repeat.

The reason why it is called a 4-phase protocol even though it is described as having 6 steps here is that 4 transitions on the input signals are needed in each transfer cycle.

### 3.2 Interconnection using picoPIPE

In figure 3.4 three picoPIPE stages connecting a sending circuit with a receiving circuit are shown. To demonstrate the domino effect of this handshaking protocol when several stages are connected in series, the waveform in figure 3.5 has been drawn. In the initial state, \textit{Ready 1}, \textit{Ready 2} and \textit{Ready out} are 0, meaning that all latches are open and none of the stages contain any valid data. To initiate the transfer of data, the circuit connected to \textit{Ready in} signals that new data is ready at the input by setting it to 1. As described in the previous example, this will close latch 1, signal to the sending circuit that data has been received and signal to the next picoPIPE stage that data is valid. As soon as the first stage acknowledge that data has been received, the sending circuit sets \textit{Ready in} to 0, and starts calculating the next data to be sent. The transfer of the data through the picoPIPE stages happens automatically, without any external control, in accordance with the behavior described earlier. Finally, the receiving circuit sends an acknowledge on \textit{Ack in} as a reaction to the \textit{Ready out} signal and the transfer is complete.

![Figure 3.4: Three pipeline stages.](image-url)
3.3 Improvements and modifications

The example transfer above describes the principal behavior of the picoPIPE architecture. However, to improve performance and simplify certain parts of the pipeline circuit, 2-phase [16] and 1-phase [12] handshaking protocols, as well as modified C-elements [11] are used.

2-phase handshaking is created by modifying the handshaking pipeline, so that it is triggered on both rising and falling edges of the triggering input signal, instead of only on rising or falling edge as is the case in 4-phase handshaking [16].

In 1-phase logic, the acknowledge signal of a stage is disregarded. During synthesis an analysis is done on the circuit to find which stages are idle, i.e. empty stages that will immediately transfer the data to the next stage. Because these stages always can receive data, the acknowledge signal is disregarded.

An extra input can be added to a C-element by inserting an extra PMOS and NMOS transistor into the respective stack in figure 3.2. Extra inputs are needed if data from one stage is sent to several stages or if one stage receives data from several stages. When sending to several stages, the sending stage needs to have the acknowledge signals from all receiving stages connected to its C-element. Vice versa, if data from several stages are needed in one stage, that stage needs to have the ready signals from all the sending stages connected to its C-element. Furthermore, by inserting parallel transistors to an input, that particular input’s effect on the C-element can be switched on or off, making the handshaking configurable.

To use the data in a stage, the latch is replaced by either an RLB or a hard block with a fixed function. These blocks have a longer propagation delay than the latch, and it varies depending on what function they carry out. Because of
this, modified pipeline stages are added to the path of the handshaking signals and the path is made programmable so that it can match the propagation delay of the data path [11]. This ensures that the ready signal does not arrive at the output before the data is actually ready.

### 3.4 picoPIPE usage in FPGA

Now that the low level details have been explained, the effect of the picoPIPE on the FPGA can be discussed. The asynchronous core of the FPGA is surrounded by a clocked frame. This frame contains converters called *Synchronous-Asynchronous Converter* (SAC) and *Asynchronous-Synchronous Converter* (ASC) that handle clocking data in and out of the asynchronous core. Thanks to this frame, the FPGA will behave like a synchronous circuit when viewed from the outside.

![Figure 3.6: A simple clocked circuit.](image)

Figure 3.6 contains a simple example of a clocked circuit. An example of how the resulting implementation could look like in Speedster HP is shown in figure 3.7. The combinational logic has been implemented in two RLBs, and the interconnection between them contains a number of picoPIPE stages. When the clock goes high, the SAC will read the input data and convert it to input signals for the picoPIPE fabric. The output will be the data itself and the handshake signal. The data will then be passed on into the first RLB where its programmed logic function will be applied to the data, and the handshake signal will pass through a path with the same delay as the RLB. The data will then pass through a number of picoPIPE stages as it is sent through the interconnect fabric to the second RLB. How many stages it will pass through depends on how long the interconnection is. When the handshake signals that the data has reached the second RLB, its programmed logic function will be applied to the data before it is passed on to the ASC. As soon as the clock goes high after the data has arrived, it will be converted back into synchronously clocked data at the output of the ASC.

Considering only the basic behavior of this simple circuit, some observations can be made. For the ASC to be able to function properly, it needs to have valid data every time it is clocked. This means that the data sent into the circuit by the SAC must reach the input of the ASC before it is clocked. To make a comparison to a regular circuit, the SAC and ASC can be seen as registers, and the circuit between them like some combinational logic. That would mean that the clock frequency would be limited in a traditional manner by the delay of the combinational path between the registers.
However, thanks to the inclusion of the picoPIPE stages, the behavior is quite different. Each picoPIPE stage can hold one valid data, or data token. This can be exploited through something called Extra Pipelining (XP). When the circuit in figure 3.7 is initialized, all the picoPIPE stages will be empty. To make this explanation simple, it will be assumed that there are 3 picoPIPE stages between the two RLBs. If data is allowed to be clocked into the circuit for 3 clock cycles, while no data is clocked out, the picoPIPEs can be filled with data. This is what Achronix refers to as inserting extra pipeline stages, in this case XP equals 3. Once they are filled, the minimum period for the ASC will only be the delay through the second RLB, since new data is already available in the picoPIPE right next to it. The same is true for the SAC; as soon as the data has reached the picoPIPE directly after the first RLB, new data can be sent in. The effect is that the critical path is shortened, resulting in a higher maximum frequency. When the circuit is synthesized for the Speedster HP FPGA, Achronix tool called Achronix CAD Environment (ACE) will analyze the circuit and automatically determine how many extra pipeline stages should be used for maximum performance.

Another important aspect of the picoPIPE architecture is how registers are handled. Figure 3.8 depicts a manually pipelined version of the circuit in figure 3.6. The combinational logic has been split into two blocks and a pipeline register has been inserted between them. This is the normal way to increase performance for a circuit in a traditional FPGA. To understand how registers are handled by the picoPIPE fabric, it can be assumed that this circuit also is synthesized into what is shown in figure 3.7. The two logic blocks are implemented in one RLB each. The pipeline register will be converted into a picoPIPE stage. This is done by initializing the specific picoPIPE as non-empty. It will contain valid data when the circuit is started. The other 2 picoPIPEs will be left empty, meaning that a
maximum of 2 extra pipeline stages can be inserted. If that is the case, then the end result will be the same as for the circuit in figure 3.6. The only difference between the two is that in the first case the tool did the pipelining automatically.

The advantages of the picoPIPE technology can be summarized into three main points:

1. A long interconnection will not slow down the circuit since it will be made up of many short interconnections between picoPIPEs. This can be seen as automatic geometrical pipelining.

2. A circuit can be automatically pipelined by using the picoPIPEs that are already in the interconnection fabric. Furthermore, this will not affect the behavior of the circuit.

3. The whole core of an FPGA that uses the picoPIPE technology will be asynchronous. This means that there is no need for a clock distribution network, which makes up a big part of the power consumption in a traditional FPGA.

### 3.5 Limitations with picoPIPE

In the previous section, the details of the picoPIPE architecture were discussed. This architecture is very well suited for pure feed-forward circuits, since any number of picoPIPE stages can be used as extra pipeline stages anywhere in the circuit without the need to redo the timing analysis. The latency in terms of clock cycles will of course increase if the picoPIPEs are used as extra pipelines.

![Figure 3.9: Circuits with a feedback loops.](image-url)

However, there are two basic circuit constructs for when picoPIPEs can not be used as extra pipeline stages. The first problematic circuit is a loop, as seen in figure 3.9. In the top circuit, the data passes through the loop in one clock cycle. The critical path through the combinational logic will set the limit on how fast the
loop can run. In the bottom circuit, the combinational logic has been pipelined to speed up the loop. This will, however, change the function of the circuit, because now the latency in terms of clockcycles through the loop is doubled. No matter where inside the loop the pipeline register is placed, it will still affect the functionality. This can be directly translated into using the picoPIPEs in the loop as extra pipeline stages. The effect will be the same. For this reason, the clock frequency of loops can not be increased by using the picoPIPE stages.

Figure 3.10: Circuit with an unbalanced reconvergent path.

The second problematic circuit is an unbalanced reconvergent path. A reconvergent path appears when the circuit is split up into two branches that process the same data in parallel and then reconverge. In figure 3.10 an example is shown. The small boxes represent picoPIPEs and the black squares represent valid data, also referred to as data tokens. When this circuit is initialized, all the picoPIPEs are empty, as can be seen in the top part of the figure. In the bottom part of the picture, by using XP the picoPIPEs have been filled with as much data as possible from the input. It is clear that the maximum XP setting is 2 because after two clock cycles all the picoPIPEs in the shorter top path will contain data tokens. The path with the fewest number of picoPIPEs will limit the performance.

To solve this problem, it might be possible to balance the two paths by routing the top path so that it includes one more picoPIPE. Then all the picoPIPEs can be fully utilized. This case is shown in figure 3.11.

Figure 3.11: Circuit with an balanced reconvergent path where all picoPIPEs are utilized.
Chapter 4

Initial test designs

This chapter describes the basic circuits used in a first round of tests. These tests have been performed in order to understand the benefits and limitations of the two FPGA architectures. The goal is to answer the first two questions stated in the purpose section of this thesis in chapter 1.

4.1 Test design and motivation

Any FPGA contains a number of different blocks that are programmable to various degrees. To be able to evaluate the performance of the FPGA, it is reasonable to first analyze each specific type of block by itself and then test more complex circuits where different types of blocks are combined. Furthermore, the core architecture and especially the interconnection fabric must be taken into consideration since it is very different for the two FPGAs used in this thesis. The focus is on high performance and how to achieve maximum clock frequency. Area information is included only when it is a relevant part of the test results.

In this chapter a number of different circuit concepts are studied using test circuits. The following sections will give a motivation to why they are chosen for analysis as well as how the test circuits are designed.

4.1.1 Distributed logic

To implement a logic function in an FPGA, the RLBs (or CLBs) are used. A logic function can be split up into a number of sub-functions that are distributed among the RLBs, and these can then be connected through the programmable interconnections to form the complete function. For this reason this is called distributed logic. It is the most essential part of an FPGA, and therefore it is important to evaluate its performance.

To evaluate this type of logic, a circuit that calculates the sum of a number of 16-bit values has been designed. This circuit is chosen because addition is a common arithmetic function. It can also easily be expanded into a summation that can be used to test if the clock frequency is dependent on the number of
terms in the sum. If it is not, then automatic pipelining works in this case. The word length is set to 16 bits because that represents a realistic use of an FPGA.

The purpose with the experiments done on distributed logic is to answer the following questions:

1. What is the maximum clock frequency for distributed logic in the Speedster HP?

2. Can Speedster HP use the picoPIPE technology to automatically pipeline distributed logic?

4.1.2 Multipliers

Multipliers are needed to implement many different algorithms, and at the same time they are relatively complex. Implementing them using LUTs is possible, but that would consume a lot of area and not yield good performance. Therefore hard block multipliers that can carry out fixed point multiplication are found in almost any FPGA. The experiments in this section were done to provide answers to the following questions:

1. What is the highest performance of a single multiplier, and what is needed in order to achieve it?

2. How does the word length of the input data affect the performance?

3. Can Speedster HP use the picoPIPE technology to automatically pipeline a long chain of multipliers?

To answer the first two questions, a test circuit consisting of a multiplier with an adjustable number of input and output registers as well as configurable width has been designed. The word length is varied from 2 up to 32 bits, to find both when the synthesis tool choose to use a hardware multiplier and what happens when the word length is longer than what can fit in a single multiplier. Several multipliers connected in series are used as a test circuit to provide answers to the third question in the same way as the summation is used in the distributed logic case.

4.1.3 Simple filter structures

To test a combination of distributed logic and multipliers, a simple filter structure has been designed. It is closer to real-world usage of an FPGA than the circuits used in the previous experiments. The goal with this experiment is to test if the automatical pipelining works in a more complex circuit. Also, the filter coefficients have been made configurable in order to test if they affected the performance.
4.1.4 Resets

In some of the previous experiments it was observed that including reset functionality would sometimes affect the performance of the circuit. Therefore an analysis of resets in the Speedster HP is needed. To do this, the circuits used in the previous tests are evaluated with asynchronous and synchronous reset functionality.

4.1.5 Loops

Loops are common in many types of circuits, for example as part of a control structure or in calculations that require a feedback. As previously mentioned in chapter 2, a loop circuit structure is problematic for the picoPIPE architecture since it can not be automatically pipelined. In a loop some part of the output is used as input, so if the latency in the loop is changed then the functionality will also change. For this reason it is important to analyze how loops affect the performance of the Speedster HP FPGA. Two types of common loops are analyzed: finite state machines and mathematical circuits with feedback.

4.2 Methodology

Before an explanation of the methodology used in these experiments can be given, it is necessary to explain the work-flow of test circuit development for the two FPGAs.

First the test circuit is described in VHSIC Hardware Description Language (VHDL) code. VHDL is a hardware description programming language used to describe the behavior or structure of digital circuits, or more specifically Very High Speed Integrated Circuit (VHSIC). This code is then compiled and a simulator is used to verify that the circuit functions as expected.

Next, the test circuit code is synthesized for each of the two FPGAs. Synthesis is a process where the goal is to find a way to program and connect parts in the FPGA so that they match the description in the code. How this is done is of course completely dependent on the FPGA, so different tools have to be used for different FPGAs.

For the Xilinx FPGA, Xilinx development suite called ISE is used. It can carry out the whole synthesis process from compiling the VHDL code to a creating a programming file for the FPGA.

Achronix has chosen a different approach for their development tools. First the VHDL code needs to be compiled and then synthesized into a netlist, which is a list of connections between parts found in the FPGA. This is done in a third party tool customized for Achronix FPGAs. In this thesis Precision Synthesis from Mentor Graphics has been chosen for this task. The netlist is then loaded into Achronix own development tool called ACE. This tool is used to do a place-and-route of the netlist onto the FPGA while taking the picoPIPE technology into consideration.

To get the performance numbers from each test, the timing analysis tools in ISE and ACE are used. Timing analysis can be performed at different steps in the
synthesis procedure. The most reliable numbers are given by the post-place-and-route timing analysis, since it is performed on the final result of the synthesis. For this reason, only the post-place-and-route timing analysis is used.

The synthesis tools are very complex and have many settings that affect the final result. The goal of this master thesis is not to find the optimal settings for a given design, but rather to evaluate and compare the performance of the FPGAs. The only setting that is changed from its default value is the speed grade. For the Virtex-6 it is set to -1, meaning the cheapest and slowest in the family. For the Speedster HP it is set to standard.

In both ISE and ACE, timing constraints can be specified for the clock signals in the design. In ISE this can be done directly in the tool or by including a file that specifies the constraints. In ACE, a file containing the constraints needs to be included first, but can be edited directly in the tool after that. For the Speedster HP FPGA the number of extra pipeline stages used is also specified in this file.

To get the highest performance from either of the FPGAs, a special approach is needed in order to force the tools to do their best. If the timing constraints are too relaxed, the optimization will stop after they were met. If they are set too hard, the tool will give up prematurely. In both cases the resulting maximum clock frequency will be lower than the actual maximum.

The performance evaluation process in ISE for the Virtex-6 has been as follows. First, the circuit is synthesized with an initial timing constraint on the clock period. Then, the timing analysis reports if the constraint is met or not, and the achieved minimum period. If the constraint is met, it is further lowered until a failure is reported. When a failure occurs, the constraint is relaxed until it can be met. In this way a good approximation of the maximum frequency can be found.

When evaluating the performance of the Speedster HP in ACE, the process has been slightly different. As in ISE, a clock period timing constraint can be specified, but the number of XP can also be set. After synthesizing in ACE, the timing analysis tool will list the settings that should be used according to its analysis to get the highest performance. However, this has been found to not always be accurate and the same iterative process as with ISE is sometimes needed to find the best settings.

4.3 Test circuit considerations

When a circuit is synthesized as a top module, the number of input and output pins used can affect the results if the circuit is very small. To remove this bias from the test results, a data source with one input and a generic number of outputs is used in the tests where this is an issue. It consists of a shift chain of registers where each register output is also connected to an input on the test circuit. See figure 4.1 for a schematic. A data sink is also created for the outputs by simply connecting them to an AND gate. This prevents the synthesis tool from removing any part of the circuit during optimization.
4.4 Analysis of distributed logic

The circuit used for the distributed logic experiments is shown in figure 4.2. It consists of an binary tree of adders, where the critical path is increased with one adder when the number of input values is doubled. The performance of this circuit has been evaluated with 2, 4, 8 and 16 inputs. Table 4.1 contains the results.

![Figure 4.2: A 4-input adder tree.](image)

To analyze if distributed logic can be automatically pipelined in the Speedster HP, the case with two inputs should be considered. In the Virtex-6, the achieved clock frequency is around 600 MHz. However, on the Speedster HP it is more than 1.3 GHz, which means that it has more than double the performance. This clearly shows that the Speedster HP can outperform traditional state-of-the-art FPGAs.

To analyze the maximum clock frequency for distributed logic in the Speedster HP, the case with two inputs should be considered. In the Virtex-6, the achieved clock frequency is around 600 MHz. However, on the Speedster HP it is more than 1.3 GHz, which means that it has more than double the performance. This clearly shows that the Speedster HP can outperform traditional state-of-the-art FPGAs.

To analyze if distributed logic can be automatically pipelined in the Speedster HP, the case with two inputs should be considered. In the Virtex-6, the achieved clock frequency is around 600 MHz. However, on the Speedster HP it is more than 1.3 GHz, which means that it has more than double the performance. This clearly shows that the Speedster HP can outperform traditional state-of-the-art FPGAs.

To analyze the maximum clock frequency for distributed logic in the Speedster HP, the case with two inputs should be considered. In the Virtex-6, the achieved clock frequency is around 600 MHz. However, on the Speedster HP it is more than 1.3 GHz, which means that it has more than double the performance. This clearly shows that the Speedster HP can outperform traditional state-of-the-art FPGAs.
<table>
<thead>
<tr>
<th>Inputs</th>
<th>Virtex-6</th>
<th>Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>597 MHz</td>
<td>1319 MHz</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>304 MHz</td>
<td>1319 MHz</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>213 MHz</td>
<td>1319 MHz</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>154 MHz</td>
<td>1319 MHz</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 4.1: Maximum frequencies for the adder trees.

HP, if the performance with two inputs is used as a baseline, a clear pattern can be seen for the performance of the Virtex-6. Four inputs results in half the performance, 8 inputs reduces it further to a third and finally with 16 inputs it becomes a fourth of the performance with two inputs. This can easily be explained by looking at the adder tree structure: with two inputs there is one adder in the critical path, with four inputs there are two, with 8 inputs three, and with 16 inputs 4. On the Speedster HP on the other hand the clock frequency stays constant independent of how many inputs are used because the circuit is automatically pipelined. This shows that XP can be used very efficiently in distributed logic.

4.5 Analysis of multipliers

A brief description of the multipliers found in the two FPGAs used in this thesis was given in chapter 2. In this section, relevant details of each block are described so that the experimental results can be explained.

4.5.1 The Speedster HP MACC block

![Speedster HP MACC block diagram](image)

Figure 4.3: Speedster HP MACC block.

As explained earlier, the multipliers in the Speedster HP are inside the MACC blocks, which also contain an adder and an accumulator register. In figure 4.3 a complete MACC block is shown. Each input of the multiplier has a word length of 28 bits, and the adder and accumulator are 56 bits wide. All of the registers as well as the adder can be bypassed depending on how the multiplexers are programmed [5]. Using all the registers and bypassing the adder should give the highest performance when using only the multiplier.
4.5.2 The Virtex-6 DSP block

The multipliers in the Virtex-6 FPGA are located in the DSP48E1 blocks. These blocks are more complex than the Speedster HP MACCs, and can carry out many different types of operations [3], as discussed in chapter 2. However, since the DSP48E1 is used as a comparison to the Speedster HP MACC block, only the MACC functionality has been analyzed. A simplified schematic for the DSP block is shown in figure 4.4, where all other parts have been omitted. There are two registers for each input because in the full DSP block input B can be connected to a preadder with an accumulator register. There are a number of other differences compared to the Speedster MACC block. If the adder after the multiplier is not used, zeros will be added instead of bypassing the adder. This adder and the accumulator register are 48 bits wide, so the multiplier result is sign extended. The 5 extra bits in the accumulator can be used as guard bits to check if the MACC operation caused an overflow. Since the adder can not be bypassed, it should be necessary to use both registers after the multiplier to achieve maximum performance.

4.5.3 Multiplier experiments

The first test circuit that is used in the multiplier experiments is a 16-bit fixed point multiplier with different numbers of input and output registers. The purpose is to answer the question about how to get the highest performance from the multipliers, and to get a baseline to compare the results in the other multiplier experiments with. The word length is set to 16 bits because that fits into a single multiplier in both FPGAs.

Table 4.2 shows the performance achieved on the Virtex-6 in this experiment. In the register column, “X in” means that X registers were used at each input, and
### Table 4.2: 16-bit fixed point multiplication performance for the Virtex-6.

<table>
<thead>
<tr>
<th>Registers</th>
<th>$f_{max}$</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 in, 1 out</td>
<td>263 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>1 in, 2 out</td>
<td>473 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>1 in, 3 out</td>
<td>473 MHz</td>
<td>1 DSP, 32 FF</td>
</tr>
<tr>
<td>2 in, 1 out</td>
<td>263 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>2 in, 2 out</td>
<td>473 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>2 in, 3 out</td>
<td>473 MHz</td>
<td>1 DSP, 32 FF</td>
</tr>
</tbody>
</table>

“Y out” means that Y registers were used at the output. In the resources column, DSP means the DSP48E1 block, and FF means flip-flop.

It is clear that both output register are needed for maximum performance, but using either one or two input registers does not affect the performance. That both output registers are required for maximum performance needs to be taken into consideration when implementing algorithms. Furthermore, using flip-flops to create a third output register outside the DSP block only uses extra resources and does not improve the performance further. Thus it can be concluded that the maximum performance is around 470 MHz, which can be used as a comparison for other tests.

### Table 4.3: 16-bit fixed point multiplication performance for Speedster HP.

<table>
<thead>
<tr>
<th>Registers</th>
<th>$f_{max}$</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 in, 1 out</td>
<td>1499 MHz</td>
<td>6</td>
<td>1 BMULT</td>
</tr>
<tr>
<td>1 in, 2 out</td>
<td>1499 MHz</td>
<td>6</td>
<td>1 BMULT, 32 FF, 32 LUT</td>
</tr>
<tr>
<td>1 in, 3 out</td>
<td>1499 MHz</td>
<td>8</td>
<td>1 BMULT, 32 SHIFTER</td>
</tr>
<tr>
<td>2 in, 1 out</td>
<td>1499 MHz</td>
<td>6</td>
<td>1 BMULT, 32 DFF, 32 LUT4</td>
</tr>
<tr>
<td>2 in, 2 out</td>
<td>1499 MHz</td>
<td>8</td>
<td>1 BMULT, 32 SHIFTER</td>
</tr>
<tr>
<td>2 in, 3 out</td>
<td>1499 MHz</td>
<td>7</td>
<td>1 BMULT, 32 DFF, 32 LUT4, 32 SHIFTER</td>
</tr>
</tbody>
</table>

The same tests have been done on the Speedster HP, and the results are shown in table 4.3. The register and $f_{max}$ columns are the same as in table 4.2. The XP column shows the number of XP used. In the resources column, BMULT means the Speedster HP MACC block, FF means flip-flop, LUT means look-up-table and SHIFTER means hardware shift register.

It is possible to reach 1.5 GHz in all cases. The number of registers used does not affect the performance, but it affects the number of resources used and the number of XP. If more than one input or output register is used, resources outside the MACC block are needed, so the second internal output register in the MACC can not be used in this case. Therefore to get the best hardware utilization only one input and one output register should be used.

In the next experiment the goal is to answer the second question about the multipliers: how does the word length of the input data affect the performance?
4.5 Analysis of multipliers

To test this, the number of input registers is set to one for both FPGAs, and the number of output registers is set to one for the Speedster HP and two for the Virtex-6. These have been determined as the best settings in the previous experiment.

<table>
<thead>
<tr>
<th>Word length</th>
<th>$f_{max}$</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>700 MHz</td>
<td>12 FF, 6 LUT</td>
</tr>
<tr>
<td>4</td>
<td>461 MHz</td>
<td>22 FF, 23 LUT</td>
</tr>
<tr>
<td>8</td>
<td>473 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>16</td>
<td>473 MHz</td>
<td>1 DSP</td>
</tr>
<tr>
<td>32</td>
<td>138 MHz</td>
<td>4 DSP, 34 FF, 7 LUT</td>
</tr>
</tbody>
</table>

Table 4.4: Fixed point multiplication performance for Virtex-6 as a function of the wordlength.

The results for Virtex-6 are shown in table 4.4. For 2 and 4-bit input the multiplication is carried out in distributed logic. Multiplication with only 2-bit words is a very simple operation, so the critical path between the input and output registers will be very short, hence the higher clock frequency in table 4.4. With 4-bit words more LUTs are required to be able to carry out the calculation, which in turn creates a longer critical path resulting in a lower clock frequency. When 8-bit words are used, the synthesizer starts using the multiplier in the DSP block. Changing the word length to 16 bits does not affect the performance since the same hardware is used. For 32-bit words, 4 DSP blocks are needed, as well as some distributed logic. The clock frequency is considerably reduced because the registers inside the DSP blocks are no longer used in the best way.

<table>
<thead>
<tr>
<th>Word length</th>
<th>$f_{max}$</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1499 MHz</td>
<td>4</td>
<td>8 FF, 8 LUT</td>
</tr>
<tr>
<td>4</td>
<td>1319 MHz</td>
<td>6</td>
<td>9 ALU, 16 FF, 24 LUT</td>
</tr>
<tr>
<td>8</td>
<td>1499 MHz</td>
<td>5</td>
<td>1 BMULT</td>
</tr>
<tr>
<td>16</td>
<td>1499 MHz</td>
<td>6</td>
<td>1 BMULT</td>
</tr>
<tr>
<td>32</td>
<td>1319 MHz</td>
<td>11</td>
<td>44 ALU, 4 BMULT, 64 FF, 27 LUT</td>
</tr>
</tbody>
</table>

Table 4.5: Fixed point multiplication performance for Speedster HP.

In table 4.5 the results from the same experiment done on the Speedster HP are shown. ALU in the resources column means the carry chain adder in the HLC. In the same way as the Virtex-6, for 2 and 4-bit words distributed logic is used. The interesting thing to note is that when the ALU is needed, the tool reports that it limits the clock frequency to approximately 1.3 GHz. For 8 and 16-bit words one MACC block is used, and the performance is independent of the word length. The most interesting result comes from the 32-bit multiplication. As in the Virtex-6, 4 hardware multipliers and some distributed logic are needed to realize the circuit. However, the performance is still good because the number of XP that can be
used is increased. This shows that the picoPIPE technology can be very useful in practice, since the designer does not have to manually adjust the circuit when doing multiplications of larger width than the width of a single multiplier.

To analyze the automatic pipelining, a new test circuit has been designed. It is shown in figure 4.5 and consists of a number of multipliers connected in series. Registers are only used at the input and output of the circuit, so the chain of multipliers form a combinational path. Its purpose is to test the automatic pipelining using picoPIPEs. If the automatic pipelining works, then the clock frequency should stay the same, independent of the number of multipliers in the circuit. The same tests are done on the Virtex-6 so that a comparison between a traditional FPGA and the Speedster HP can be done. 16-bit fixed point words are used so that each multiplication can be carried out by a single multiplier. Furthermore, the coefficient and the input are connected to a data source. It is omitted in the figure for clarity. If constant values are used for the coefficients, it gives the synthesizer the opportunity to simplify the multiplication, which in turn could have an affect on the results.

![Figure 4.5: Chain of multipliers with clocked input and output.](image-url)

Table 4.6 shows a comparison of the clock frequency for different numbers of multipliers. Starting with the traditional FPGA, the pattern for the Virtex-6 is very clear. When the number of multipliers is doubled, the length of the critical path is also doubled. This leads to a 50 percent decrease in clock frequency. However, for the Speedster HP the clock frequency stays constant, independent of the number of multipliers. Instead, the number of XP increases. Thus, it can be concluded that the picoPIPE technology works perfectly in this simple test circuit.

<table>
<thead>
<tr>
<th>Multipliers</th>
<th>Virtex-6</th>
<th>Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>263 MHz</td>
<td>1499 MHz</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>124 MHz</td>
<td>1499 MHz</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>60 MHz</td>
<td>1497 MHz</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>30 MHz</td>
<td>1499 MHz</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>15 MHz</td>
<td>1497 MHz</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 4.6: Maximum clock frequencies for the multiplier chain.
4.6 Analysis of a simple filter

The previous tests were done on very deliberate test circuits designed to test a specific part of the FPGAs. To test something closer to real-world usage, a basic filter circuit has been designed. It can be seen in figure 4.6. It is a straightforward implementation of an image filter for a 3x3 matrix of pixels and it has not been pipelined. The reason for not doing this is to test if a more complex circuit can be automatically pipelined.

There are many different filters that can be implemented with this circuit by just changing the multiplication coefficients, also known as the filter kernel. A purpose of this experiment is to test if the choice of filter coefficients affects the performance. Three common filters are chosen for this. The coefficients are shown in a matrix to make it more easy to understand which constant is applied to a certain pixel.

The operation of this filter circuit can be described as follows. First 9 pixels that form a 3x3 block in the image are clocked in. Then each pixel is multiplied by some constant value, determined by the filter kernel implemented. Finally, all the products of the pixel multiplications are accumulated and the result is clocked out. Both the input pixels and the output pixel, as well as the multiplication coefficients, are 8-bit fixed-point two’s complement numbers. The sum of the 16-bit multiplication results is calculated and then rounded to 8 bits before it is sent out as the output pixel.

\[
\begin{bmatrix}
-\frac{1}{8} & -\frac{1}{4} & -\frac{1}{8} \\
0 & 0 & 0 \\
\frac{1}{8} & \frac{1}{4} & \frac{1}{8}
\end{bmatrix}
\]  

(4.1)

<table>
<thead>
<tr>
<th>FPGA</th>
<th>(f_{\text{max}})</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>203 MHz</td>
<td>-</td>
<td>66 FF, 81 LUT</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>1319 MHz</td>
<td>13</td>
<td>11 ALU, 24 FF, 104 LUT, 32 SHIFTER</td>
</tr>
</tbody>
</table>

Table 4.7: Maximum clock frequencies for the Sobel filter.
The first filter kernel that is tested is a Sobel filter. This is a filter used for edge detection in computer vision. The coefficients seen in equation (4.1) are used for this type of filter. The middle row of coefficients are zeroes, so the synthesis tool should remove the corresponding logic from the circuit when it is optimized. The multiplication by the other six coefficients result in a division by an even power of two, so they can be performed as shift operations. Thus, only adders to calculate the sum should be needed. Both ISE and ACE are able to exploit this fact, and only use flip-flops and LUTs. The results in table 4.7 show that the Speedster HP can use XP efficiently in this more complex circuit. The clock frequency is the same as that achieved in the experiments in section 4.4.

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{4}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16}
\end{bmatrix}
\] (4.2)

<table>
<thead>
<tr>
<th>FPGA</th>
<th>(f_{\text{max}})</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>165 MHz</td>
<td>-</td>
<td>105 FF, 86 LUT</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>1319 MHz</td>
<td>13</td>
<td>18 ALU, 72 FF, 168 LUT, 8 SHIFTER</td>
</tr>
</tbody>
</table>

Table 4.8: Maximum clock frequencies for the Gaussian filter.

Next, a Gaussian blur kernel is tested. The coefficients used can be seen in equation (4.2). Once again they are such that only shifts and adders are needed for the multiplication, but now 9 coefficients are used, instead of 6 in the Sobel case. Looking at the results in table 4.8, the Virtex-6 performance is affected by the higher number of coefficients. The clock frequency is lowered by the longer critical path through the adders. The performance for the Speedster HP, however, is not dependent on the number of coefficients used since XP can be used efficiently here too.

\[
\begin{bmatrix}
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{bmatrix}
\] (4.3)

<table>
<thead>
<tr>
<th>FPGA</th>
<th>(f_{\text{max}})</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>58 MHz</td>
<td>-</td>
<td>64 FF, 8 LUT, 9 DSP</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>1319 MHz</td>
<td>14</td>
<td>101 ALU, 147 FF, 192 LUT, 8 SHIFTER</td>
</tr>
</tbody>
</table>

Table 4.9: Maximum clock frequencies for the mean value filter.

Lastly, a mean value filter is tested by using the coefficients in equation (4.3). They are clearly not integer powers of two, so the result of each multiplication can not be calculated by just using a shift register, as is possible with the other kernels. For this reason ISE chooses to use DSP blocks when it synthesizes the
circuit. This leads to lower performance than with the other filter kernels, so the clock frequency is not only affected by the number of filter coefficients, but also by their value.

When the same filter is synthesized for the Speedster HP, the synthesis tool chooses to use distributed logic for the multiplications. This is possible because the coefficients are constant. Any multiplication with a constant value can be calculated with shifters and adders [15]. It is done by splitting the multiplication up into several multiplications with integer powers of two (shift operations), and then adding up the results of these multiplications to get the final result. This of course results in a much more complex circuit than when the Sobel or Gaussian blur kernels are used, but it does not affect the clock frequency. The picoPIPEs can be used here as well to enable the same performance as with the other kernels.

Thus it can be concluded that for this filter, the complexity of this circuit does not affect its performance on the Speedster HP. The same can not be said for the Virtex-6, where the performance is dependent both on the number of coefficients and their value.

4.7 Analysis of resets

When a circuit is powered on, the contents of all volatile memories such as flip-flops/register and RAMs are unknown. To bring the circuit to a known state, a reset signal is commonly used to load the memories with valid data. There are two ways to perform a reset, synchronously or asynchronously. With a synchronous reset, the memories are loaded if the reset signal is high when the clock goes high. With an asynchronous reset, the memories are loaded whenever the reset signal goes high, independent of the clock.

For this experiment, some of the previous test circuits have been redesigned so that they have either synchronous or asynchronous reset. The purpose is to find if having a reset affects the performance, and if there is a difference between having a synchronous and an asynchronous reset.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No reset</th>
<th>Synchronous reset</th>
<th>Asynchronous reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit mult.</td>
<td>473 MHz</td>
<td>473 MHz</td>
<td>186 MHz</td>
</tr>
<tr>
<td>32-bit mult.</td>
<td>138 MHz</td>
<td>138 MHz</td>
<td>96 MHz</td>
</tr>
<tr>
<td>4-input add.</td>
<td>304 MHz</td>
<td>296 MHz</td>
<td>296 MHz</td>
</tr>
<tr>
<td>8-input add.</td>
<td>213 MHz</td>
<td>213 MHz</td>
<td>213 MHz</td>
</tr>
</tbody>
</table>

Table 4.10: Performance on the Virtex-6 of some test circuits with different types of reset.

The results for Virtex-6 are shown in table 4.10. The Virtex-6 handles synchronous resets well, there is only a very slight reduction in clock frequency for the 4-input adder tree. With asynchronous reset the behavior is different. In both multiplier circuits the maximum clock frequency is much reduced compared to the synchronous reset. This happens because the internal pipeline registers in the
DSP block does not support asynchronous reset. For the adder tree circuits, however, the choice of reset does not affect the performance since they only contain distributed logic which supports asynchronous reset.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No reset</th>
<th>Synchronous reset</th>
<th>Asynchronous reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit mult.</td>
<td>1499 MHz</td>
<td>1410 MHz</td>
<td>1335 MHz</td>
</tr>
<tr>
<td>32-bit mult.</td>
<td>1319 MHz</td>
<td>1319 MHz</td>
<td>1040 MHz</td>
</tr>
<tr>
<td>4-input add.</td>
<td>1319 MHz</td>
<td>1319 MHz</td>
<td>1319 MHz</td>
</tr>
<tr>
<td>8-input add.</td>
<td>1319 MHz</td>
<td>1274 MHz</td>
<td>1319 MHz</td>
</tr>
</tbody>
</table>

Table 4.11: Performance on the Speedster HP of some test circuits with different types of reset.

The same tests were done on the Speedster HP and the results can be found in table 4.11. The behavior is somewhat similar to that of the Virtex-6. For the multiplier circuits, the worst case is the asynchronous reset, although for the 16-bit case the relative reduction in clock frequency is much less than in the Virtex-6. Including reset of either type in the adder trees has a negligible effect on the performance.

Thus it can be concluded that for both FPGAs, the performance is affected negatively if asynchronous reset is used in the multipliers. Therefore the best choice if reset is needed for the multiplier circuits is the synchronous reset. For the adder trees, the choice of reset does not affect the performance in a considerable way, so it can be chosen freely.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No reset</th>
<th>Synchronous reset</th>
<th>Asynchronous reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sobel</td>
<td>210 MHz</td>
<td>205 MHz</td>
<td>210 MHz</td>
</tr>
<tr>
<td>Gaussian</td>
<td>165 MHz</td>
<td>164 MHz</td>
<td>164 MHz</td>
</tr>
<tr>
<td>Mean</td>
<td>58 MHz</td>
<td>58 MHz</td>
<td>53 MHz</td>
</tr>
</tbody>
</table>

Table 4.12: Performance on the Virtex-6 of the filter kernels with different types of reset.

Moving on to the filter kernels, the clock frequencies for the Virtex-6 with different reset configurations are shown in table 4.12. Both the Sobel kernel and the Gaussian kernel contain only LUTs and flip-flops, and neither of them are affected by the choice of reset. This is coherent with the results for the adder trees that also only use distributed logic. The mean value filter, that uses DSP blocks, also retains most of its performance even when asynchronous reset is used. This differs from the previous results from the pure multiplication circuits, where asynchronous reset lowers the clock frequency very much. However, the clock frequency for the mean value filter is very low with any choice of reset, so the performance is not lowered much by only using registers external to the DSP blocks.
### Table 4.13: Performance on the Speedster HP of the filter kernels with different types of reset.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No reset</th>
<th>Synchronous reset</th>
<th>Asynchronous reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sobel</td>
<td>1309 MHz</td>
<td>996 MHz</td>
<td>895 MHz</td>
</tr>
<tr>
<td>Gaussian</td>
<td>1319 MHz</td>
<td>923 MHz</td>
<td>868 MHz</td>
</tr>
<tr>
<td>Mean</td>
<td>1319 MHz</td>
<td>1319 MHz</td>
<td>1319 MHz</td>
</tr>
</tbody>
</table>

The results in table 4.13 lists the performance of the filter kernels on the Speedster HP. The behavior differs much from that of the Virtex-6. When reset is used it is apparent that the choice of filter kernel affects the performance. With the mean value kernel there is no problem to achieve maximum clock frequency independent of the type of reset used. With the Sobel and Gaussian blur kernels, however, the clock frequency is considerably reduced with both synchronous and asynchronous reset. This does not correspond to the behavior observed when reset is used in either the mean value kernel or the adder trees, even though all four test circuits are made up entirely of distributed logic. Seeing these results prompts further exploration of the behavior.

Figure 4.7: Performance as a function of the number of non-zero coefficients for the Gaussian blur filter with synchronous reset.

Synchronous reset in the Gaussian blur kernel is chosen as a test case for further analysis. Setting all coefficients in the kernel except one to zero, and then retesting the circuit with one more non-zero coefficient added each time provides the results shown in figure 4.7. The filter is also expanded to use 4x4 pixels and tested with 15 and 16 coefficients to provide further data for the analysis. The figure shows...
that the performance is dependent on the number of coefficients, contrary to what was previously found when comparing the Gaussian blur kernel to the Sobel kernel with no reset. It is not a linear dependence, but the trend is that a higher number of coefficients results in lower performance. The case with 8 coefficients is a clear exception to this trend and should be connected to it being an integer power of two. Several other tests where the value and positions in the matrix of the coefficients have been tested, but no clear answer as to what exactly cause this performance problem could be found. The mean value kernel has also been tested in the 4x4 filter, but with all coefficients kept at 1/9 so that they still are not integer powers of two. Independent of the reset configuration, the performance remains the same as in the 3x3 filter. The conclusion thus is that when reset is used in this filter, both the value and the number of kernel coefficients affect the performance. As seen in the first filter experiment, this dependence is completely removed when reset is not used.

A more general conclusion that can be drawn from the tests with resets is that the effect of including reset in the Speedster HP is different compared to the Virtex-6. When designing circuits for the Speedster HP, the designer needs to be aware of the performance problems that could arise from introducing reset functionality in a circuit.

### 4.8 Finite state machines

A very common control loop structure is the Finite State Machine (FSM). An FSM consists of a finite number of states, where each state represents a certain control step in the operation that is implemented with the FSM. In each state the FSM outputs control signals to the controlled circuit in order to make it carry out the operation of that step in the process. The transitions between states in the FSM itself are either controlled by external control signals, or happen when a specified condition is fulfilled, such as a counter reaching a certain number. There are also FSMs where the transitions are not controllable and occur in a preprogrammed order.

For the FSMs this section aims to answer the following questions:

1. Does the controllability an FSM affect its performance?

2. Can the performance of a larger FSM be improved by splitting it up into several smaller FSMs?

To answer the first question, three different versions of a simple FSM intended to control a mathematical operation are analyzed. The FSM has four states that each perform a part of the operation using control signals. Since it is the performance of the FSM that is evaluated, the control signals are not connected to a circuit that actually performs the operation.

The most basic version can be seen in figure 4.8a. This version loops unconditionally and it works as follows: when the circuit is reset, it will go to the “Idle” state where all control signals are turned off. This state is marked with a thicker
4.8 Finite state machines

![Finite state machines](image)

Figure 4.8: Three versions of a small FSM. (a): An uncontrolled FSM. (b): A controlled one-way FSM. (c): A controlled two-way FSM.

circle to show that it is the reset state. In the next clock cycle the FSM goes to the “Read” state where the corresponding control signal is turned on, representing that the data needed for the operation is loaded into registers. Then it goes to the “Calc” state, turning the calculation control signal on to represent that the results are calculated. Finally it goes to the “Write” state where the results are written into memory by turning the write control signal on, and then it goes back to the “Idle” state.

In the version seen in figure 4.8b an input control signal called go has been added. It is used to control when the FSM goes to the next state, simulating that several clock cycles could be used to perform each part of the operation. Still, the FSM always traverses the states in the same order.

Finally, the FSM is expanded with an extra control signal so that it can go both back and forward, as shown in figure 4.8c. If both signals are turned off or on at the same time the FSM will stay in its current state.

<table>
<thead>
<tr>
<th>FSM</th>
<th>$f_{max}$ Virtex-6</th>
<th>$f_{max}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncontrolled</td>
<td>700 MHz</td>
<td>1451 MHz</td>
<td>6</td>
</tr>
<tr>
<td>Controlled one-way</td>
<td>700 MHz</td>
<td>1499 MHz</td>
<td>6</td>
</tr>
<tr>
<td>Controlled two-way</td>
<td>700 MHz</td>
<td>1499 MHz</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.14: Maximum clock frequencies for the three versions of the simple FSM.

The performance numbers for the three versions of this simple FSM are shown in table 4.14. Neither the Virtex-6 nor the Speedster HP are affected by the varying level of control over the FSM. Only in the case of the uncontrolled FSM in the Speedster HP the placement is a little different and the feedback of the current state lowers the performance slightly.

To answer the second question about FSMs, a bigger and more complex FSM is needed. For this purpose, the two memory controllers in figure 4.9 and figure
4.10 have been designed. They are both implemented to use a 1024-word 16-bit memory. The FSM in figure 4.9 operates as follows: when the circuit is reset, it starts in the “Idle” state. A read or write operation is initiated by sending a request via their respective control signals. If, for example, the write_req signal is turned on, the FSM will send a signal to the memory to read the write address. When it gets an acknowledge from the memory, it will signal to the memory to write the input data to that address. After it gets a second acknowledge from the memory, the FSM will signal that the writing is done before returning to the “Idle” state.

One way to reduce the complexity of a larger FSM is to split it up into several smaller FSMs that are controlled by one main FSM with very few states. Such an FSM is sometimes referred to as an hierarchical FSM. This has been done with the memory controller FSM as shown in figure 4.10. It carries out the same operations as the FSM in figure 4.9, but the implementation is split up into two separate FSMs for the reading and writing operations. They are controlled by a main FSM with only three states. Only the FSM for the write operation is included in the figure, since the FSM for the reading operation is identical, apart from the control signals. When a write request is signaled, the main FSM will go to the “write” state, and this will trigger the “Write” FSM that will write data to the memory in the same way as the previous FSM. The “Write” FSM will then signal that it is done writing to the main FSM, which will return to the Idle state.

Figure 4.9: A memory controller implemented in an FSM.

<table>
<thead>
<tr>
<th>FSM</th>
<th>$f_{\text{max}}$ Virtex-6</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single, without memory</td>
<td>700 MHz</td>
<td>1189 MHz</td>
<td>5</td>
</tr>
<tr>
<td>Hierarchical, without memory</td>
<td>700 MHz</td>
<td>816 MHz</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.15: Maximum clock frequencies for the memory controllers.

Table 4.15 shows the achieved maximum clock frequencies for the two versions of the memory controller. With the Virtex-6, it is clear that the type of FSM does not affect the performance. On the other hand, the performance of the Speedster HP is very different for the two types of FSMs. With both FSMs ACE report that the loop structure limits the performance. These results thus suggest that when
4.9 Circuits with data feedback

To further analyze the effect of loops, a number of different mathematical circuits with feedbacks have been designed and evaluated. These types of loop structures are common when implementing mathematical algorithms.

![Diagram of FSMs](image)

Figure 4.10: A memory controller implemented in several FSMs.

Designing an FSM for use in the Speedster HP, it is better to make one large FSM than to split it up into several smaller ones.

### 4.9 Circuits with data feedback

To further analyze the effect of loops, a number of different mathematical circuits with feedbacks have been designed and evaluated. These types of loop structures are common when implementing mathematical algorithms.

![Diagram of accumulator](image)

Figure 4.11: An accumulator with an enable signal.

The circuit in figure 4.11 is a controllable accumulator that adds the input value to the accumulator register when the CE signal is high. This circuit was designed in order to answer these questions:
1. Does a simple accumulator work well, despite its feedback structure?

2. What happens if the additions in the accumulator is replaced with a subtraction?

Question 2 might seem like a strange question to ask, but during the work on this thesis there had been some weird results that indicated that the behavior of addition and subtraction is not the same in the Speedster HP.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>$f_{\text{max}}$ Virtex-6</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit</td>
<td>700 MHz</td>
<td>1319 MHz</td>
<td>7</td>
</tr>
<tr>
<td>8-bit, subtract input</td>
<td>700 MHz</td>
<td>1255 MHz</td>
<td>6</td>
</tr>
<tr>
<td>8-bit, subtract feedback</td>
<td>700 MHz</td>
<td>939 MHz</td>
<td>6</td>
</tr>
<tr>
<td>16-bit</td>
<td>700 MHz</td>
<td>1242 MHz</td>
<td>6</td>
</tr>
<tr>
<td>16-bit, subtract input</td>
<td>700 MHz</td>
<td>1190 MHz</td>
<td>6</td>
</tr>
<tr>
<td>16-bit, subtract feedback</td>
<td>700 MHz</td>
<td>939 MHz</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.16: Maximum clock frequencies for different versions of the accumulator.

The results from the tests of the accumulator are in table 4.16. The configuration column shows the word length and the calculation done in the adder. For example, “subtract input” means that the input value is subtracted from the feedback value.

When subtraction is not done, a high clock frequency is achieved in both the Virtex-6 and the Speedster HP. Even though this circuit has a feedback loop, XP can still be used efficiently. This answers the first question.

In the four configurations when subtraction is done, there is no effect on the performance in the Virtex-6. The Speedster HP on the other hand gets lower performance, and it is different depending on which value is subtracted. There is a possible explanation as to why the performance is lower when the feedback value is subtracted from the input value. Note that this is just a hypothesis and it has not been confirmed by Achronix that this is what really happens. For this explanation it is needed to assume that a subtraction in the Speedster HP could be done by first negating the value that is to be subtracted, and then adding the negated value to the other value to get the result. If this is true and the hardware that calculates the negated value is placed in such a way that XP can not be used to speed up the circuit, in this case inside a loop, then that would lead to a longer critical path and lower performance.

Three additional circuits with feedback have been designed to further analyze the effect of feedback loops on the performance of the Speedster HP. The purpose of this experiment is to answer the following questions:

1. Does the type of calculation performed inside the loop affect the performance?

2. For each of these circuits, is it the feedback that limits the performance?
3. If different calculations are done in parallel inside a loop, does the slowest calculation determine the performance of the whole loop?

The idea for these circuits comes from a Proportional-Integral-Derivative (PID) regulator common in control systems [9]. This type of regulator consists of three main parts, a multiplier to calculate the proportional term, an accumulator to calculate the integral term and a subtractor to calculate the discrete derivative term. In a real regulator these terms are individually multiplied with some constants to give them different weights, and then the sum of the weighted terms is used as a control signal for the regulated system. The output of the regulated system is then sent back and subtracted from the input value (which is the wanted output from the system) to calculate the error. However, since it is the loop performance that is of interest in these tests, the regulated system as well as the individual weighing of the terms have been removed. 16-bit 2’s complement fixed point values are used in all calculations.

$$\text{Reg Reg Reg Reg Reg}$$

$$\begin{array}{c}
x_{\text{in}} & 16 \\
+ & 16 \\
\times & 32 \\
\text{Constant} & 16 \\
\text{Bit 30 to 15} & 16 \\
\text{Reg Reg Reg Reg} \\
\rightarrow & y_{\text{out}}
\end{array}$$

Figure 4.12: Multiplier circuit with a feedback loop that calculates the P term.

In figure 4.12, a circuit with only the proportional term can be seen. To give the loops of all the circuits the same latency in terms of clock cycles, two registers are put after the multiplier. The multiplication is done in a DSP block and a MACC block respectively, and the coefficient is set at synthesis time. The result from the multiplication is truncated before it is used in the feedback and output register.

$$\text{Reg Reg Reg Reg Reg}$$

$$\begin{array}{c}
x_{\text{in}} & 16 \\
+ & 16 \\
+ & 16 \\
\text{Reg Reg Reg} \\
\rightarrow & y_{\text{out}}
\end{array}$$

Figure 4.13: Accumulator circuit with a feedback loop that calculates the I term.

A circuit with only the integral term is shown in figure 4.13. This circuit contains two loops, a short one in the accumulator and a long one for the feedback.

To evaluate only the discrete derivative term, the circuit in figure 4.14 has been designed. The difference compared to the accumulator is that instead of a short feedback there is a feedforward and a subtraction instead of an addition.

The answer to the first question can be found by looking at the results in table 4.17. Each of the test circuits have also been tested without feedback in order to
answer question 2. This is done by connecting input ports to both inputs of the first adder. By comparing the different results from the Virtex-6, there is a clear difference in performance between the P term circuit and the other two. The P term is calculated using a DSP block and that is what limits the performance. The other two circuits only use distributed logic and, therefore, have roughly the same performance. Furthermore, there is no considerable change in the performance when the feedback is removed. This leads to the conclusion that in the Virtex-6 it is the content of the loop, not the loop itself, that limits the performance. Once again, the results from the Speedster HP indicate a different behavior. All three circuits have about the same performance, with the I term circuit being somewhat slower than the other two. However, if the feedback is removed, the performance of the P term circuit and the I term circuit goes up to the maximum of what the ALU blocks that make up the adders can handle. If the feedback is removed in the D term circuit the performance is also increased, but not as much as in the other two circuits. Thus, the conclusion for the Speedster HP is that both the feedback itself and the content of the loop limits the performance in these circuits.

The results for the D term circuit on the Speedster HP prompts further tests to find out what limits the performance in the circuit when the feedback is removed. In the previous experiment with the accumulator it was found that using subtraction has an impact on the performance. Therefore, three additional versions of the D term circuit without feedback have been designed: (a) where the subtraction in the first adder is replaced by an addition, (b) where the subtraction in the second adder is replaced, and (c) where both subtractions are replaced.

The results from the tests with the different versions of the D term circuit

---

**Figure 4.14:** Subtractor circuit with a feedback loop that calculates the D term.

**Table 4.17:** Maximum clock frequencies for the individual terms, with and without feedback.
### 4.9 Circuits with data feedback

<table>
<thead>
<tr>
<th>Circuit</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>D term, no feedback</td>
<td>972 MHz</td>
<td>4</td>
</tr>
<tr>
<td>D term, no feedback, (a)</td>
<td>1319 MHz</td>
<td>8</td>
</tr>
<tr>
<td>D term, no feedback, (b)</td>
<td>1319 MHz</td>
<td>8</td>
</tr>
<tr>
<td>D term, no feedback, (c)</td>
<td>1319 MHz</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 4.18: Maximum clock frequencies for the different versions of the D term circuit.

without feedback are in table 4.18. They show that if any of the subtractions is removed, the performance is improved to the same level as the P term and I term circuits without feedback. Again, this demonstrates that carrying out subtractions in the Speedster HP may lead to strange problems with performance. There is no information available from Achronix that explains why this happens.

Finally the circuit in figure 4.15 with all the three terms combined has been designed. An adder and a register have been included to sum up the terms before they are sent to the feedback and the output. The purpose of this circuit is to find an answer to question 3: if different calculations are done in parallel inside a loop, does the slowest calculation determine the performance of the whole loop? The circuit was also tested without a feedback to see how much the performance is limited by the loop.

Table 4.19 shows the results. The circuit also has been modified in a few different ways to test if there is a certain part that limits the performance. With the “no I term” modification, the I term is replaced by a D term calculation to see if the performance is improved by removing the term that is individually slowest. The “D term addition” modification is done by replacing the subtraction in the D term calculation with an addition. This is tested to analyze if the term that has the lowest performance when no feedback is used can be removed to make the whole
circuit faster. Finally the whole circuit is tested with the feedback removed to see how much it limits the performance. Also, the “D term addition” modification is tested without feedback to find if it can improve the performance further.

The conclusion for the Virtex-6 is that the performance stays the same, independent of the various modifications. It is the DSP block that limits the performance in all cases. Several conclusions can be drawn from the tests on the Speedster HP. First of all, for the unmodified version of the circuit, the performance is lower than that of the slowest individual term (I term in table 4.17). By removing that term, the performance is increased, but not to the level of the second individually slowest term (D term). If the feedback is removed, however, that results in a higher performance than what is achieved in the D term circuit with removed feedback (see table 4.17). Lastly, removing the subtraction in the calculation of the D part does not affect the performance whether or not feedback is used. In summary this shows that both the performance and behavior changes when these calculations are done in parallel on the Speedster HP.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>$f_{\text{max}}$ Virtex-6</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>394 MHz</td>
<td>546 MHz</td>
<td>3</td>
</tr>
<tr>
<td>PID, no I term</td>
<td>391 MHz</td>
<td>618 MHz</td>
<td>2</td>
</tr>
<tr>
<td>PID, D term addition</td>
<td>394 MHz</td>
<td>553 MHz</td>
<td>3</td>
</tr>
<tr>
<td>PID, no feedback</td>
<td>379 MHz</td>
<td>1236 MHz</td>
<td>13</td>
</tr>
<tr>
<td>PID, no feedback, D term addition</td>
<td>379 MHz</td>
<td>1236 MHz</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.19: Maximum clock frequencies for the combined P, I and D term circuit with different modifications.
Chapter 5

Guidelines for hardware design on the Speedster HP

In this chapter the test results from chapter 4 and the conclusions that can be drawn from those tests are used to compile a number of design guidelines for the Speedster HP FPGA. In some of the initial tests the behavior as well as the achieved performance differed considerably between the Speedster HP and the Virtex-6, therefore it is necessary to take the special characteristics of the Speedster HP FPGA into consideration when designing a circuit for it.

- Distributed logic
  1. Distributed logic does not need to be pipelined for performance reasons on the Speedster HP. The same is true when it is combined with BMULTs.

- Multipliers
  1. To minimize hardware utilization, only one register should be placed at each input and at the output of the multiplier. Otherwise, extra distributed logic is added.
  2. Numbers that are less than 8 bits wide should be sign extended to 8 bits before multiplication. This will make the synthesis tool choose a BMULT instead of distributed logic, which in turn will maximize the performance.
  3. Multiplications by numbers that are wider than 28 bits (the width of the multiplier inside the BMULT) are handled automatically by the synthesis tool. It divides it into partial multiplications on several BMULTs in a way that still maintains good performance. Therefore it is not necessary for the designer to spend time on doing it manually.
  4. Contrary to traditional FPGAs, it is not necessary to use pipeline registers to achieve good performance when several multiplications are done
in series. The designer should take this advantage into consideration when implementing circuits on the Speedster HP FPGA.

- Resets

  1. The designer should be careful with resets in the Speedster HP, and avoid using them if possible. Including reset functionality can lead to decreased performance.

  2. It is not clear exactly in which situations it will cause problems.

- Finite state machines

  1. The controlability of an FSM does not affect its performance, so the designer does not need to take this into consideration when designing it.

  2. It is better to use one big FSM than to split the functionality into several smaller FSMs.

- Circuits with data feedback

  1. A good approach to maximize the performance of a circuit with data feedback is to first synthesize an initial design, and then use the results to iteratively improve it. This is needed because the performance is determined by a combination of the individual components speed and how they are connected inside the loop.

  2. The designer should pay special attention to subtractions, because they have been shown to lead to performance degradation.
Chapter 6

Median filter

This chapter describes the work that has been done on a median filter that was provided by Synective Labs AB. It had been designed earlier as a demonstration circuit and had been developed for the Xilinx FPGAs. The first goal is to make it run on the Achronix FPGAs. Then problems in the design that limited the speed on the Achronix FPGAs are identified and isolated. These problems are studied in detail to understand what caused them to limit the speed of the circuit. In an iterative process for each part of the circuit where problems are present, several different redesigns are tested to find new solutions that remove the bottlenecks.

6.1 Algorithm description

A median filter is a widely used algorithm in image processing [8] [13]. Before a description of the details of the implementation analyzed in this thesis can be given, an introduction to what a median filter is and how it is used in image processing is necessary.

The input to a median filter is a square block of pixels from an image. In the filter the pixels are sorted from highest to lowest intensity. From the sorted pixels, the median is picked to be the pixel for that block in the filtered image. This is done for all possible blocks of the specified size so that a filtered picture of only median pixels is created. Since there is no data dependency between the output of one block and the input of another, the algorithm is massively parallelisable and well suited for implementation on an FPGA.

Median filters are used to reduce or remove noise in a picture. The noise pixels will differ much in intensity from the surrounding pixels, and will thus end up in the top or bottom of the sorted pixels.

The implementation of the median filter analyzed in this chapter works on gray-scale pictures, but the filter can be expanded to color pictures by filtering each color channel one at a time. The size of the pixel block is 5x5 pixels.
6.2 System description

In this section the different parts of the median filter and their respective functions are described.

6.2.1 Top module

In figure 6.1 a simplified schematic of the whole system including median filter and buffers, can be seen. When it is used, it is programmed onto an FPGA and connected to a memory bus on a host system. Software running on the host system can control the filter, as well as read and write data to it, using a specified address range.

To use the filter, the aforementioned software is used to write pixels into the input FIFO memory. 8 pixels of 8 bits each are written as a 64-bit word. The pixels are treated as gray-scale, meaning that they only have an intensity and no color. A simple control block regulates the data flow using a counter and the status flags from the input and output FIFO. The counter is needed to count the number of clock cycles since the last pixel word was fed into the filter. One pixel is consumed each clock cycle. The control block will thus wait 8 clock cycles before it tries to read from the FIFO again, since 8 pixels are fed into the filter at every read. When the control block tries to read from the input FIFO it checks that it is not empty, and that the output FIFO is not almost full. If the read fails, it will try again until it succeeds. Writing of the filtered pixels into the output FIFO is controlled by the median filter itself. The pixels are then read from the output FIFO using the controlling software.

Figure 6.1: Overview schematic of the whole system.
6.2 System description

6.2.2 FIFO

The FIFO memories enable the controlling software to read or write many pixels consecutively. Without them the data management would be more difficult. Figure 6.2 is a simplified schematic of the FIFO. The Writing and Reading internal control signals are generated asynchronously, whereas the rest of the signals are clocked. The RAM acts as a synchronous dual-port memory that can do reading and writing in parallel. The address generators are simply counters controlled by the Writing and Reading signals respectively. The data counter is also controlled by these signals. If only Writing is high, the counter is incremented one step since one more data word has been added to the FIFO, and decremented one step for the case when only Reading is high. If both signal are the same, the data counter is unchanged.

![Schematic of the FIFO memories.](image)

Figure 6.2: Schematic for the FIFO memories.

There are actually four independent blocks for generating the status flags, but only the Empty flag block has been included since they all have identical connections and control logic. For example, if there is only one data word in the FIFO, and only Reading is high, the last word is being read from the FIFO so the Empty flag goes high. Once the Empty flag has been set it will stay high until only Writing goes high. The Almost empty and Almost full levels can be changed with generics in the code.
6.2.3 Median filter

The contents of the median filter block in figure 6.1 can be seen in figure 6.3. Five consecutive lines of the image are needed to construct each 5x5 pixel block. The line buffers store the 4 previous lines and combine them with the incoming line to get the 5 lines needed. Because 8 pixels are written in each 64-bit word, the lowest 3 bits in the Line length signal are not connected to the line buffer.

![Figure 6.3: Schematic of the median filter block.](image)

When all the lines are valid, the Neighborhoods block reads them and constructs 5x5 neighborhoods around each new pixel. There are four Neighborhood signals because the code from Synective Labs supports sorting up to four pixel blocks in parallel in four sorters. In the work done in this thesis one sorter is used.

The output register combines the filtered pixels into 64-bit words before sending them to the output FIFO as seen in figure 6.1. The Median valid signal is generated by a counter that counts how many unsorted neighborhoods has been inserted into the sorting kernel. The pipeline in the kernel will be stalled when there is no valid input, so the first sorted pixel will appear at the output once the kernel is completely filled. The Output valid signal is also controlled by a counter which counts the number of valid median pixels currently in its buffer. When 8 pixels have arrived in the buffer, the output will be valid.

6.2.4 Line buffers

Pixels are fed in and out of the line buffer in 64-bit words containing 8 pixels each. To fill the line buffers correctly, the read and write pointers are controlled in a special way. They are basically counters that trigger each other.

The first incoming line is stored in the first line buffer by incrementing Write pointer 1 by one step every time Pixel valid goes high. Once the first line has been written, Read pointer 1 and Write pointer 2 also starts to increment one step when Pixel valid goes high. Because of this, the first line will be copied to the second line buffer while the second line is being written to the first line buffer. Once all buffers are filled and the fifth line starts to be fed in, the line buffer will signal that the output is valid.
6.2 System description

6.2.5 Bubblesort kernel

The most essential part of the median filter block is the sorter kernel. A pipelined version of the bubblesort algorithm called “Batcher’s odd-even mergesort” [10] is used in the Synective Labs implementation. It was considered easy to implement and also area efficient compared to other alternatives.

![Diagram of line buffers](image)

Figure 6.4: Schematic of the line buffers.

![Diagram of sorter block](image)

Figure 6.5: The sorter block used in the sorting network.

The sorter kernel is made up of two-input sorters seen in figure 6.5. When
**Input enable** is high, the unsorted values are sorted and clocked into the output registers. The sorters are arranged in lines, or pipeline stages, that are connected to its neighbors to form the complete sorting network, see figure 6.6. All blocks and registers are connected to the **Input enable** signal and clock, but for simplicity these connections have not been included in the schematic. From the output of the sorting network only the median pixel is used, so all unused sorting blocks are removed by the synthesis tool and have been grayed out in the figure.

Since all the sorter blocks are controlled by the **Input enable** signal, the whole pipeline is stalled when there is no valid input. A counter keeps track of how many blocks of data has been fed into the pipeline, and once the first block is sorted it will signal that the output is valid.

![Figure 6.6: Schematic of the sorter network.](image)

### 6.3 Identified problems and tested solutions

After the original design has been analyzed and understood, the next step is to synthesize it to get the initial performance. The goal for this part of the thesis is to find common design decisions that cause bottlenecks when synthesizing for the Speedster HP FPGA, and then try to remove them by redesigning the circuit.

#### 6.3.1 Adapting the RAM for Achronix FPGAs

Since the circuit had been designed with Xilinx FPGAs in mind, the RAM in the FIFO are instantiated as inferred RAM, meaning that they are described in such a way that ISE can understand that block RAMs should be used. In practice the RAM used acts as full synchronous memory, but it has a few synchronous signals that are copied asynchronously to the outputs.

As can be seen in section 2.3, the block RAM used in Achronix Speedster HP does not support any asynchronous access, and because of this Precision Synthesis has to use LUTs and registers to create the RAM in the FIFO. This leads to problems both with speed and utilization. In order to solve this, all asynchronous
6.3 Identified problems and tested solutions

signals are removed without changing the functionality of the RAM. Some other unnecessary signals, such as the read and write byte masks are also removed to simplify the design. After doing this the synthesis tools is able to use block RAMs for these memories.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>f_{max}</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>278 MHz</td>
<td>-</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>402 MHz</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6.1: Maximum clock frequency for the whole median filter system.

The achieved minimum clock periods on both the Speedster HP and the Virtex-6 can be seen in table 6.1. One other thing to note is that the whole median filter system is a quite large circuit, so synthesizing it is time consuming. Therefore not as much effort is put into optimizing the constraints as in the initial tests in chapter 4. In the synthesis report produced by ACE, the type and location of the critical path is stated. In the case of the complete median filter, a loop in one of the blocks that produce the status flags in the FIFO is reported as the critical path. Therefore the next part chosen for redesign is the FIFO memory.

6.3.2 Analysis and redesign of the FIFO memories

Figure 6.7 is a detailed schematic of the Empty status flag generator used in the FIFO. The reset is asynchronous. The only difference between the flag generators is the number used in the comparator.

Figure 6.7: Schematic of the Empty flag generator.

If the Empty flag is low it will be inverted when there is one data value in the FIFO and it is being read but no new data value is being written. If the flag is high it will be inverted when a new data value is being written into the memory but not read at the same time. As can be seen in the schematic, the output of the
<table>
<thead>
<tr>
<th>Configuration</th>
<th>$f_{max}$ Virtex-6</th>
<th>$f_{max}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous flags</td>
<td>400 MHZ</td>
<td>467 MHz</td>
<td>3</td>
</tr>
<tr>
<td>Asynchronous flags</td>
<td>390 MHz</td>
<td>398 MHz</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.2: Maximum clock frequency for the FIFO with two versions of flag generators.

register is connected to both the inverted input and the multiplexer, and this is what creates a loop. The achieved performance when only synthesizing the FIFO is stated as the “Synchronous flags” configuration in table 6.2.

This design will generate the flags synchronously, and because of this it becomes rather complicated. It must be decided if the FIFO is going to be empty (or almost empty, etc.) the next clock cycle, so the flags can not be generated by comparing the Data count signal to the number of data that each flag represents.

When analyzing the loop in the generator, another less obvious loop is discovered in the FIFO. It can be seen in figure 6.2. To generate the Reading signal, the Empty flag is used. The Reading signal is then used to control the Empty flag generator. The same is true for the Writing signal and the Full flag. Thus removing the internal loop in the flag generators would only help so much, and a redesign that remove both loops is needed.

A redesign is tested where the flags are generated asynchronously using only the Data count signal. This removes the internal loops in the flag generators. However, there is still another loop, since the Data count signal is controlled by the Reading and Writing signals, and they in turn are determined using the flags. By removing the flag generators the critical loop actually gets shorter, which impacts the performance in a negative way, see the “Asynchronous flags” configuration in table 6.2.

One way to remove the Data count signal would be to instead compare the read and write addresses. The problem with that solution is that the addresses wrap around when they reach the end of the address space, so the comparison would be wrong as soon as one of them did that. Preventing them from wrapping around would create its own loops as well as other problems. No solution that completely removes the loops can be found, so it is necessary to change the approach. The FIFO memories are only used to make the data management in the software easier. If the control software is redesigned they are not needed. When they are removed only the median filter core remains, and the new approach is to redesign it in a bottom up way.

### 6.3.3 Redesigning the sorting kernel

The first thing that is discovered when analyzing the design of the sorting kernel is that it has a loop in the counter that produces the Median valid signal. As long as the counter is less than the number of stages in the sorting network, it will be incremented by the Input enable signal. Once it is equal to the number of stages
<table>
<thead>
<tr>
<th>Configuration</th>
<th>$f_{\text{max}}$ Virtex-6</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>372 MHz</td>
<td>506 MHz</td>
<td>10</td>
</tr>
<tr>
<td>Improved</td>
<td>401 MHz</td>
<td>844 MHz</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 6.3: Maximum clock frequency for the two versions of the sorter kernel.

in the sorting network, it will directly copy the *Input enable* signal to the *Median valid* signal. A detailed schematic can be seen in figure 6.8.

![Schematic](image)

Figure 6.8: Schematic of the *Median valid* signal generator in the sorting kernel.

The *Input enable* signal also controls the data flow in the pipeline. When there is no valid input, the pipeline is stalled. This means that this signal must be connected to every block in the pipeline, giving it a huge fanout. In table 6.3 the “Original” configuration show the performance for this design.

To speed up the sorting kernel, the loop should be removed, and the rest of the circuit should be simplified as much as possible. The loop is completely removed by replacing the counter for the *Input enable* signal with a pipeline of 1-bit registers of the same length as the number of stages in the sorting network. The output from the last pipeline register is then used as the *Output valid* signal. Because the pixels and the Input enable are propagated through the same number of pipeline stages they will arrive at the output at the same clock cycle. After doing this redesign, the stalling functionality can also be removed. Instead of completely filling the pipeline with data the *Output valid* signal can be used. Another signal with a big fanout that can be modified is the reset signal. Instead of having it reset the whole pipeline, it is now enough to only reset the *Input enable* pipeline.

The performance achieved with these improvements implemented is stated in table 6.3 as the “Improved” configuration. As can be seen, the performance on the Speedster HP is much higher than without the redesign. The performance on the Virtex-6 on the other hand is not affected much. This shows that very specific design decisions are needed in order to achieve good performance.

### 6.3.4 Redesigning the line buffers

In table 6.4 the performance for the original line buffer can be seen. The output from the circuit is 5 64-bit words of pixels, so this would give erroneous results if it is synthesized as a top module. To avoid this problem, the line buffer is


<table>
<thead>
<tr>
<th>Configuration</th>
<th>( f_{\text{max}} ) Virtex-6</th>
<th>( f_{\text{max}} ) Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>301 MHz</td>
<td>486 MHz</td>
<td>4</td>
</tr>
<tr>
<td>Improved</td>
<td>293 MHz</td>
<td>794 MHz</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.4: Maximum clock frequency for the line buffer.

instantiated in a top module where the 5 outputs are connected to a bit-wise AND circuit, and the resulting 64-bit word is then connected to the output of the top module. A bit-wise AND has the lowest possible logic depth while still making use of all the outputs from the line buffer.

The problem in the Speedster HP with the original line buffer design is the control block in figure 6.4. For example, the counters for Write pointer 2 and Read pointer 1 are controlled by the counter for Write pointer 1. After the first full line has been written to the first buffer, they will start to increment. The same is true for the rest of the counters. To determine if a full line has been written, the Length signal is compared with the counter. The line length is configurable during runtime, so that images of different size can be filtered without having to resynthesize the whole filter. The Lines valid signal is also controlled by the counters, and will be kept low until all 4 buffers have been filled.

Since all the lines stored in the buffers when processing an image will be of the same length, this control block can be simplified to a large degree. If the behavior of the pointers are changed, only one pointer is needed. This can be achieved by using the same read and write pointer for all buffers. The distance between the single read and write pointer will be constant because a buffer should always store as many pixels as there is in one line. Therefore the read pointer value can be calculated using the write pointer value and the Length signal.

The Lines valid signal should also be generated in an alternative way to improve the performance. The solution chosen here is to insert a “magic” pixel before the first pixel of the real image when the line buffer is reset, and detect when it appears at the output from buffer 4. The value used for this pixel is the maximum value that can be stored in the Pixels in signal. The Lines valid signal is kept low until the “magic” pixel is detected, after that the Pixels valid signal is copied to the Lines valid signal every clock cycle. To prevent any erroneous behavior, all incoming pixels are compared to the “magic” pixel and if they match, the incoming pixel is changed to have one step lower intensity. This could introduce a small error, but the difference that might occur in the output is very hard to detect with the human eye.

Removing all the loops and simplifying the design makes a big difference for the performance on the Speedster HP, see the “Improved” configuration in table 6.4. The same is true about the line buffer as with the sorter kernel. By making very deliberate design decisions, the performance on the Speedster HP can be greatly improved while still leaving the Virtex-6 performance unaffected.
### 6.3 Identified problems and tested solutions

<table>
<thead>
<tr>
<th>Configuration</th>
<th>$f_{\text{max}}$ Virtex-6</th>
<th>$f_{\text{max}}$ Speedster HP</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>286 MHz</td>
<td>461 MHz</td>
<td>8</td>
</tr>
<tr>
<td>Improved blocks</td>
<td>264 MHz</td>
<td>469 MHz</td>
<td>9</td>
</tr>
<tr>
<td>Redesigned median filter block</td>
<td>259 MHz</td>
<td>505 MHz</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 6.5: Maximum clock frequencies for the different median filter blocks.

#### 6.3.5 Redesigning the median filter block

In the median filter block the line buffer and 1, 2 or 4 sorting kernels are instantiated and connected to some control logic. Because of the lengthy synthesis time, only 1 sorting kernel is used in all the tests in this section. The achieved performance for the original design is stated in table 6.5 as the “Original” configuration. When the line buffer and sorting kernel are replaced with the improved ones, the performance shown for the “Improved blocks” configuration is achieved. When using the improved blocks, the counter that generates the \textit{Median valid} signal in the original design of the median filter block cannot be used, because that solution assumes that the pipeline will be stalled. Instead the valid signal from the sorting kernel itself is used.

To further improve the performance of the median filter block, new ways to generate the \textit{Neighborhood valid} signal and the \textit{Output valid} signal is implemented, since in the original design they also use counters with feedback loops.

A solution similar to the \textit{Data valid} pipeline used in the sorting kernel is designed for the \textit{Neighborhood valid} signal. In the neighborhoods block, the median filter block stores 12x5 pixels which are used to create 5x5 neighborhoods around 8 pixels. In figure 6.9 the stored pixels are shown, and the dashed box contains the 5x5 pixels that make up the neighborhood to pixel 1. Every time the line buffer signals that the lines are valid, the control logic in the median filter block copies the lines into the rightmost 8x5 pixels stored. Then the 8 neighborhoods are fed into the sorting kernels one by one, two by two or four by four depending on the number of sorting kernels instantiated. In the original design of the median filter block a counter is used to keep track of when all the neighborhoods have been used. This counter is replaced by a 8 bits long shift register. When new pixels are
copied into the neighborhoods, the shift register will be loaded with 8 1’s when one sorting kernel is used, 4 1’s for 2 kernels and 2 1’s for 4 kernels. The rest of the shift register is filled with 0’s. Every time pixels are fed into the sorting kernels, a 1 is shifted out of the shift register and the last element in the shift register is used as the Neighborhood valid signal.

In the redesign of the Output valid signal generator, the counter used in the original design is kept, but its functionality is altered such that the feedback loop is removed. In the original design the counter is incremented one step each time median pixels are written into the output buffer. The buffer will be full when the counter reaches 7 if one sorting kernel is used, 3 if two kernels are used, and 1 if 4 kernels are used. Once the buffer is full, Output valid is set high and the counter is reset.

In the redesign the need for the counter to be reset once the desired number of pixels are in the output buffer is removed by incrementing a different number of steps depending on the number of sorting kernels used. The counter uses 3 bits, meaning that it can count to 7. When one kernel is used, one pixel at a time will be put in the output buffer, so the counter will be incremented one step. Once the counter equals 7 when a new pixel is put into the buffer, the output buffer will be full so the Output valid signal is set high. At the same time the counter will be incremented one more step to 8, but since it only consists of 3 bits this will in effect reset the counter to 0. For two kernels the counter is incremented two steps, and the buffer is full once it reaches 6. Lastly, for four kernels the counter is incremented four steps, and the buffer is full when the counter reaches 4. Using this intrinsic trait of digital arithmetics, the feedback loop is completely removed in the redesign.

Unfortunately both these redesigns of the median filter block does not improve the performance very much, see the “Redesigned median filter block” configuration in table 6.5.

6.4 Conclusions

The most general conclusion that can be drawn from the work on this median filter is that the performance that Achronix offers with their Speedster HP series FPGAs can not in the general case be utilized by simply synthesizing a previously designed circuit using their tools. This is especially true if the circuit has been designed with some other FPGA in mind, such as Xilinx Virtex. However, the clock frequency is higher in Speedster HP in all tested cases in this chapter, and some remarkable performance can be achieved if the circuit is redesigned to better suit the Speedster HP architecture.

Looking at the clock frequencies in table 6.3, a few interesting things can be seen. First of all, by just synthesizing the original sorter kernel for Speedster HP, the clock frequency is increased more than 35 percent compared to the Virtex-6. That is a significant performance advantage, but when the kernel is redesigned to make better use of the picoPIPE architecture, the clock frequency for the Speedster HP is more than double that of the Virtex-6. Although the marketed 1.5 GHz
seems to be difficult to achieve in practice, this shows that the Speedster HP has a potential to vastly outperform comparable FPGAs if the circuit can be tailored to take advantage of the architecture.

In the case of the line buffer, the difference in performance is even bigger between the two FPGAs, see table 6.4. With the redesigned circuit the clock frequency for the Speedster HP is nearly triple that of the Virtex-6. Two other things should be noted when looking at these results. Firstly, the original and redesigned line buffers both have almost the same performance on the Virtex-6. This means that designing in a way that makes the circuit perform well on the Speedster HP does not necessarily affect the performance of the circuit when used on other FPGAs. Secondly, even though the core component of the line buffer is a block RAM, it can still run at almost 800 MHz on the Speedster HP FPGA. This means that for example big look-up tables for complex mathematical functions could be utilized in a design while still being able to achieve good performance.

When the redesigned line buffer and the sorting kernel are used in the median filter block, there is a performance increase, but it is not as big as would be expected from the previous results. The achieved clock frequencies can be compared in table 6.5. Even after redesigning the control logic in the median filter block to better suit the Speedster HP, the clock frequency goes just over 500 MHz and does not come near the 800 MHz achieved by the line buffer and sorting kernel. A very long reconvergent path is somehow created when the two blocks are connected that limits the performance. It is possible that by further analysing the circuit this path can be removed, so that the performance can be further improved.
Chapter 7

Data encryption standard

The purpose of this chapter is to test what performance is practically achievable when a circuit that is well suited for the Speedster HP is implemented from scratch. This gives the designer full freedom to find the best possible implementation. The algorithm chosen for this is Data Encryption Standard (DES). It is expected to perform well on the Speedster HP because it is a completely feedforward algorithm. At the same time it is complex enough to represent real-world usage of the FPGA. In this chapter first an introduction to the DES and the encryption algorithm used in it are given. This is followed by a description of how the algorithm is implemented and the achieved performance.

7.1 Introduction to DES

A short introduction to general encryption theory and terminology is needed to make the description of the algorithm more easy to understand. There are five essential terms when discussing ciphers; cleartext, key, ciphertext, encryption and decryption. Cleartext is the message that should be protected by the cipher. The key is the secret that is needed in order to read the message after it has been encrypted. To encrypt the message, the cleartext and the key are used as input to the encryption algorithm. The output is called ciphertext. To turn the ciphertext back into cleartext, the key is used in a decryption algorithm to reverse the effects of the encryption algorithm.

The Data Encryption Standard became effective in July 1977 [14]. It describes an encryption algorithm with the same name. The algorithm was developed by IBM for the American National Institute of Standards and Technology. In 1999 it was replaced by the more secure Triple DES algorithm, and is now only used in legacy systems.

In DES the cleartext is divided into 64-bit blocks before being used as input to the encryption algorithm. The key is also 64 bits long, but the effective key length is only 56 bits because 8 bits are used as parity bits. A flowchart of the encryption algorithm is shown in figure 7.1.
Before the encryption of the data can start, 16 sub-keys need to be generated from the original 64-bit key input. In the *Permuted choice 1* box the 8 parity bits are discarded and the order of the remaining 56 bits is changed. The details of how the order of the bits is changed in the different permutations that are used in the
algorithm can be found in the documentation of the standard. What is important to know in relation to the work done in this thesis is that all the permutations are fixed, meaning that a specific input bit will always be connected to one specific output bit.

After the permutation, the output is divided into the leftmost and rightmost 28 bits, called the left half-key and right half-key. To produce the first sub-key, each of the two parts are rotationally left shifted one step, i.e. rotate every bit one step to the left and move the leftmost bit to the rightmost position. Then they are combined into one 56-bit word again and sent to the Permutation Choice 2 box where 8 more bits are discarded and the order of the other 48 bits is changed to make the first sub-key. To make the next sub-key, the output from the two shift operations are shifted one more step before being sent to a Permutation choice 2 box. This process is repeated 14 more times to produce 16 sub-keys.

To encrypt a 64-bit block of cleartext, first the order of the individual bits in the input data is changed in the Initial permutation. The result is then split into the 32 leftmost and the 32 rightmost bits, which will be referred to as the left half-block and right half-block respectively. These two parts are processed through 16 Feistel rounds, each with a different sub-key.

![Figure 7.2: Contents of Feistel box](image-url)

In each Feistel round the right half-block is sent into a Feistel box, the contents of which is shown in figure 7.2. The right half-block is also sent unchanged to the next Feistel round where it will be used as that rounds left half-block (see figure 7.1). Inside the Feistel box the half-block is expanded to 48 bits by copying some of the bits, as described in the standard. The expanded half-block and the sub-key
of that stage is then sent through an XOR gate. The output from the XOR gate is split up into eight 6-bit numbers and each of the numbers are used as input to their respective substitution box, or S-box. In each of the eight S-boxes the 6-bit number is substituted with a 4-bit number according to tables given in the standard. The output from all the S-boxes together form a 32-bit number that is permuted in the Permutation box as the last step inside the Feistel box. Finally, to finish the Feistel round the left half-block and the output from the Feistel block are sent through an XOR gate. The result will be used as the right half-block in the next Feistel round. After the cleartext has passed through all the 16 Feistel rounds a last reordering of the bits is done in the Final Permutation box to produce the ciphertext.

To decrypt the ciphertext, the same algorithm is used. The only thing that is different from the encryption is the order in which the sub-keys are used. By simply reversing the order of the sub-keys, i.e. use the 16th sub-key in the first Feistel round, the 15th sub-key in the second round and so on, the ciphertext can be decrypted into cleartext. This is a very useful feature if the algorithm is to be implemented in hardware, since only very little extra hardware is need in order to provide both encryption and decryption functionality.

7.2 Implementations

Two different implementations of the algorithm are done in the work of this thesis. First a direct implementation is done. The purpose of this implementation is to test if the picoPIPE architecture can be used efficiently without any customization of the implementation. As seen in the introductory section of this chapter, the algorithm itself is completely feedforward. Therefore the Speedster HP might be able to achieve good performance with a direct implementation. A second implementation is done based on the first implementation. The purpose of this implementation is to achieve as high performance as possible. The details of both of these implementations are presented in this section.

7.2.1 Direct implementation

In [14] a complete description of the algorithm used in the DES is given. Apart from the mathematical description, flowcharts of the encryption computation and the key scheduling are also included. These flowcharts are used as a basis for the direct implementation, since they closely resemble a high level hardware description of the algorithm.

The different permutations in the algorithm are fixed. That means that in hardware they can be performed by rerouting signal wires. In an FPGA this can be done directly in the routing matrix. The shift operations in the sub-key generator can also be done using only routing since the number of shifts is fixed. The XOR calculation is a common hardware operation that can be implemented with RLBs/CLBs. Finally, the substitution boxes, or S-boxes, that are part of the Feistel operation can be interpreted as small read-only memories with a 6-bit address input and a 4-bit data output. This means that they can be implemented
either with LUTs or BRAMs. In summary, this encryption algorithm is well suited for an FPGA implementation.

The direct implementation is done by simply translating the behavior of the flowchart into VHDL code. To be able to measure the performance of the circuit it needs to be clocked. Therefore the input of cleartext and key, as well as the output of ciphertext is clocked in the direct implementation. The rest of the circuit does not contain any clocked components.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>$f_{\text{max}}$</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>44 MHz</td>
<td>-</td>
<td>184 FF, 1473 LUT</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>114 MHz</td>
<td>6</td>
<td>8 BRAM, 184 FF, 4670 LUT</td>
</tr>
</tbody>
</table>

Table 7.1: Maximum clock frequencies for the direct implementation.

The performance and resources needed for the direct implementation are shown in table 7.1. Neither of the FPGAs can achieve high performance with this design. For the Virtex-6 the limiting factor for the clock frequency is the very long critical path from the input registers to the output registers. On the Speedster HP on the other hand, the clock frequency is limited by a reconvergent path in the key generator. By studying figure 7.1 more closely a reconvergent path can be found in connection to the key generator. It is shown in figure 7.3.

![Figure 7.3: Two reconvergent paths in the direct implementation](image)

Figure 7.3: Two reconvergent paths in the direct implementation

Consider that the *Permuted choice 1*, *Permuted choice 2* and the shift operations are all just rerouting of signals. The Feistel operation, on the other hand, needs to use RLBs to perform the XOR and S-box operations. Therefore the length of the two routes in terms of picoPIPEs are quite different and this leads
to performance issues as explain in section 3. By studying the synthesis results further it is also found that 8 BRAMs are used. They are each configured to have a 6-bit address and a 4-bit data output, so it is clear that they are used for 8 of the S-boxes. However, there are in total 128 S-boxes in the 16 stages of the direct implementation, which means that the rest of the S-boxes have been implemented with LUTs. LUTs only have one output bit each, so to implement an S-box requires many LUTs connected in a big net with a considerable logic depth. This is what creates the big unbalance in the two reconvergent paths shown in figure 7.3. If all S-boxes are implemented using BRAMs instead of LUTs the paths will be more balanced and the performance increased.

7.2.2 High-performance implementation

To improve performance of the direct implementation it is necessary to find a way to use BRAMs to implement the S-boxes. After testing some different redesigns it is found that by placing registers at the input and output of the S-boxes, ACE starts to implement them using BRAMs. If register are inserted there, the rest of the stage also needs to be pipelined. In figure 7.4 a pipelined stage is shown. The components of the Feistel operation are included to make it clear exactly where all the pipelining registers are inserted.

Figure 7.4: A stage of the pipelined DES implementation
7.3 Conclusions

The performance and resource usage for the redesigned DES implementation is shown in table 7.2. On both FPGAs the performance is greatly increased compared to the direct implementation. In the case of the Virtex-6 inserting pipeline registers in every stage removes the long critical path that is present in the direct implementation. For the Speedster HP FPGA the problem with the reconvergent path is removed in the pipelined implementation. Instead, the clock frequency of the circuit is limited by the performance of the BRAM. Furthermore, comparing the resources used in the redesign with the direct implementation, the number of LUTs used is almost halved, and 128 BRAMs are used to implement the S-boxes.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>$f_{\text{max}}$</th>
<th>XP</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-6</td>
<td>530 MHz</td>
<td>-</td>
<td>3136 FF, 3064 LUT</td>
</tr>
<tr>
<td>Speedster HP</td>
<td>1092 MHz</td>
<td>42</td>
<td>128 BRAM, 1280 FF, 2528 LUT, 1697 SHIFTER</td>
</tr>
</tbody>
</table>

Table 7.2: Maximum clock frequencies for the pipelined implementation.

7.3 Conclusions

To summarize, some conclusions will be drawn from the results presented in this chapter. The direct implementation of the DES algorithm does not achieve good performance in either of the two FPGAs. In the Virtex-6 case this is easy to understand, considering the very long critical path from input registers to output register. However, in the case of the Speedster HP FPGA it is interesting to see that even though the circuit is completely feed-forward it can not run at a high clock frequency. Instead, it is limited by unbalanced reconvergent paths.

Reconvergent paths are common in larger circuits. It is difficult to predict at design time which ones of these will cause problems when synthesized. Since the encryption circuit implemented in this chapter is rather small compared to the median filter in chapter 6, it is possible to find the cause of the problem and resolve it within the work of this thesis. With the unbalanced path removed in the improved implementation, the performance is in line with what Achronix claim that the Speedster HP can achieve. This shows that also larger circuits can work very well on this FPGA.
Chapter 8

Conclusions and future work

In this chapter the general conclusions that can be drawn from the whole thesis are presented. A section describing possible future work is also included.

8.1 Conclusions

The work done should be concluded by going back to the questions that were stated as the purpose in section 1.2 for doing this thesis.

The first question was how fast the Speedster HP FPGA is. It has been shown in the various experiments that it very much depends on the circuit. The blocks inside the FPGA, i.e. LUTs or BRAMs, is one part of that. That is the same as in a traditional FPGA. However, even more important in the Speedster HP is the design of the circuit. In a traditional FPGA how the circuit is implemented will also affect its performance. The difference from the Speedster HP is that it is generally predictable which design choices will yield good performance. For example, having a long path between two registers will limit the clock frequency. In the Speedster HP the long path might limit the clock frequency, or it might not because it can be automatically pipelined. That depends on the blocks used in the specific path and how they are connected as well as how the surrounding circuit is designed.

The second question was what is needed in order to achieve high speed on the Speedster HP. The most important aspect is the type of circuit. Pure feedforward circuits have been shown to get very good performance on the Speedster HP, and the opposite is true for circuits that have a feedback structure or loops. In addition to this, reconvergent paths in the circuit should be designed so that they are balanced. It is difficult to predict which paths will be unbalanced when designing the circuit, but by being aware of the problem design decisions can be made based on that knowledge. Also, the synthesis tool will provide information about unbalanced paths that will make it easier to find them.

The third question was how an already designed circuit should be adapted to the Speedster HP. This is evaluated in chapter 6 where the median filter is redesigned to better suit the Speedster HP. The most important thing to do in a
redesign is to check if there are loops in the circuit, because they are most likely to limit the performance. If it is possible, loops that are found should be removed. Furthermore, if the circuit has a control mechanism it should be simplified as much as possible, both to remove loops and to reduce the dependency between different control signals.

To summarize, it has been demonstrated that the Speedster HP FPGA can achieve great performance. In order to do so it requires a circuit that is suited for it and carefully designed.

8.2 Future work

There are many areas that can be considered for further study of this technology.

For the tools, it could be analyzed how all the different settings affect the performance. The purpose would be to find what the best settings are depending on the design goal and circuit.

Also, an evaluation of the code generators would be valuable. It would be interesting to know which blocks can be generated, and what the quality of the generated code is in terms of performance and readability.

One part of the FPGAs that was not studied in isolated test circuits was memory. There are different types of memory resources in the FPGAs, so a study where their possibilities and limitations are compared could be done.

Since clock signals are treated in a very special way in the Speedster HP FPGA, it would be of interest to explore how it handles circuits with more than one clock domain. Especially clock domain crossing and synchronization issues should be tested with the picoPIPE technology.

Only feedforward circuits were used in the analysis of larger circuits. By doing further analysis of circuits that have feedbacks it might be possible to find ways to redesign them so that their performance can be improved.

A general analysis of how the speed versus area trade-off is affected by the picoPIPE technology is also a possibility for future work. There are many techniques to minimize the area that make use of feedbacks, so that might lower the performance of the circuit more on the Speedster HP than it would on a traditional FPGA.
Bibliography

[1] WP001 Introduction to Achronix FPGAs, Rev. 1.6, 7 August 2008.  


