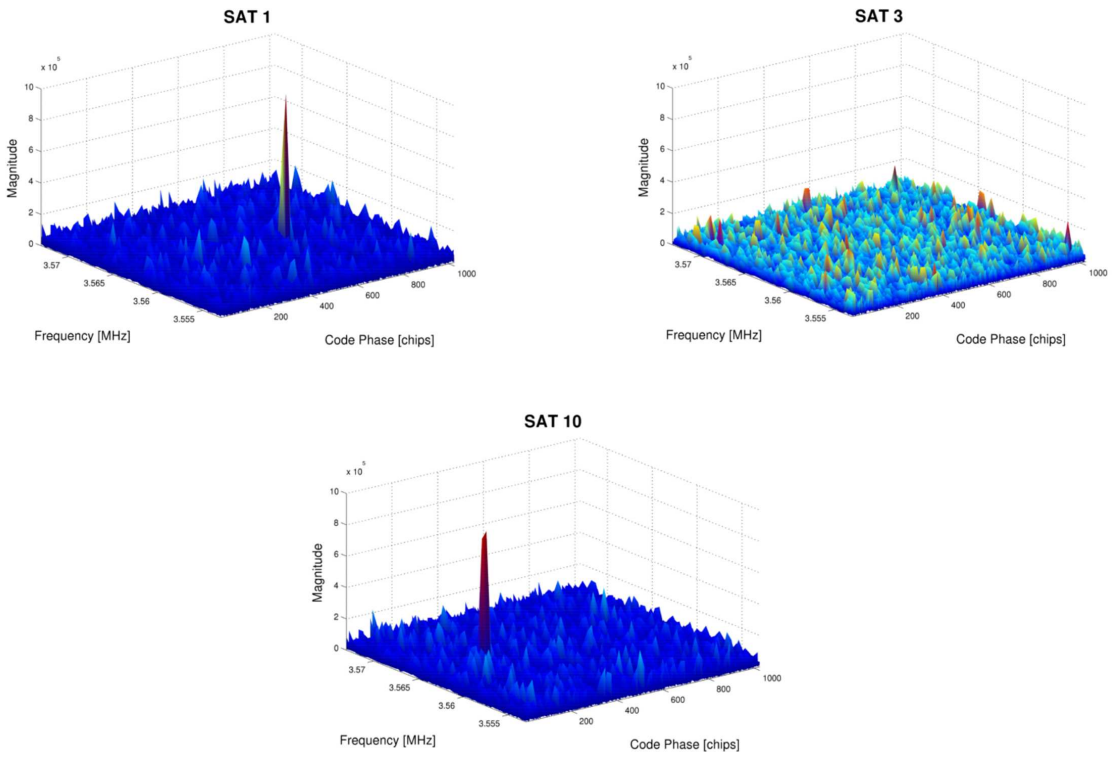
Acquisition for GPS receivers

An investigation for FPGA implementation



AALBORG UNIVERSITY

Applied Signal Processing and Implementation
Group 09gr944, Fall 2009
Faculties of Engineering, Science and Medicine
Department of Electronic Systems, Aalborg University

AALBORG UNIVERSITET

Department of Electronic Systems
9th Semester - ASPI3
Fredrik Bajers Vej 7
Telephone 96 35 86 90
http://es.aau.dk

Title:
Acquisition for GPS receivers: An investigation for FPGA implementation

Theme:
Optimal VLSI Signal Processing

Project period:
ASPI 3 - 2009

Project group:
09gr944

Group members:
Helge Glinvad Grøn

Supervisor:
Yannick Le Moullec

Publications: 3

Total number of pages: 55

Main report: 36

Appendix: 19

Supporting documents:
Attached DVD

Finished: 05.01.10

Synopsis:

The project have been an investigation of the acquisition block used in a GPS receiver for acquiring the visible GPS satellites present in sampled data. In the project the focus has been on investigating the Serial Search Algorithm with focus on execution time on a FPGA platform.
The investigation contains a Matlab implementation to verify the design, there after an implementation of two different methods using Mathworks Simulink and Xilinx System Generator.
The two designed models are based on serial and parallel principles. The two methods have been tested for their numerical properties with the (first) Matlab implementation.
After verifying both of the designs by they numerical properties, both have been synthesised using Xininx ISE and the results have been analysed. Here the serial method synthesis was successful and was ready for implementation, while the parallel method is using more IOBs than available on the selected FPGA.
The project concludes with a list of tasks for future works, with both short and long term perspectives.

Preface

This project report documents the work/research done through an 9th semester project on the Applied Signal Processing and Implementation (ASPI) specialization at Aalborg University. The project duration was 4 month and has been supervised by Associate Professor Yannick Le Moullec.

The documentation is composed of 55 pages report - the main report of 36 pages, 19 pages appendix and a DVD containing the Matlab files, test data, Xinlinx System Generator models, Xilinx project files and other file used in - or relevant for the project.

In the project following versions of the software tools have been used:

- Mathworks Matlab & Simulink 2009a
- Xilinx ISE 11.3
- Xilinx System Generator 11.3

To be sure that data on the DVD can be used is it recommended to used the same versions. All tools have been executed on a 64-bit Linux distribution and are know to work on the following versions:

- Personal: Ubuntu 9.10 Karmic Koala - 64-bit
- Lab: CentOS 5.3 Final- 64-bit

Helge Glinvad Grøn

Contents

1 Introduction

This chapter contains the motivation for the project, a description of the project and limitations for the project.

1.1 Motivation for the project

A company named GOM-Space[1], is placed in NOVI Aalborg and specialises in the development of small satellites. The company is interested in using GPS on its satellites, placed in the LEO orbit[2] of Earth for different purposes. The company are new to the GPS technology, and at first they have found two different application for GPS usage on satellites. The applications are listed below:

1. Extract time signals for exact time synchronization using the GPS network
2. Defined approximately position in near Earth orbit

Within the satellite development community, interest for FPGAs on small satellites is growing, so GOM-Space has contacted the students of A.S.P.I. on institute 8 at Aalborg University.

1.2 Project description

The project has its motivation focus on the GOM-Space's interest in using GPS in space. In other words, the project focuses on the conditions for being able to receive GPS signals. The GPS receiver and the algorithms for processing the signal are implemented on an FPGA platform. The project also focus on the implementation of the algorithms with focuses on execution of time.

Problem statement

- *Is it possible to explore the start condition blocks in a GPS receiver, by investigate their algorithmic properties and exploit them on an FPGA architecture, to make a time optimization of the start conditions?*

[1] www.gomspace.com
[2] Low Earth orbit (LEO)[**?**]: Geocentric orbits ranging in altitude from 0-2000 km

- *If possible - can the algorithm run in real time on an FPGA implementation?*

1.3 Limitations for the project

To build a complete GPS receiver from scratch is out of scope for this project. Therefore the focus is only on selected single blocks in the receiver. The project does not have focus on the whole GPS framework, but only on the blocks involved in the cold start up of a GPS receiver. The project considers an earth based receiver and for its development, data sampled and stored from Aalborg Universities GPS receivers are used.

2

GPS overview

This chapter contains an overview of the GPS framework - including GPS signals, GPS signal coding and different start conditions.

2.1 GPS - History & use

GPS is short for ”Global Positioning System”[**?**] and was introduced in 1972 with the first tests and the first satellites sent into orbit in 1978 - the system was fully operational in April 1995. The system has been continuously updated with newer satellites and features since 1995. The GPS network is mainly used for positioning. It was made for the U.S. Military, whom still have the main control of the system, but the GPS system is also used in a wide range of civil applications. The applications are spread over navigation, mapping, time synchronisation, tracking and many more. The first civil uses were for maritime navigation but now-a-days GPS receivers are found in an even wider amount of products such as mobile phones, personal data devices, laptops, cameras and many more - used to find the way, position tagging and just for fun.

2.2 GPS Receiver

A GPS receiver[**?**] is illustrated in Figure 2.1. The figure shows the receiver from the received signals in the RF-front end, to the Position calculation.

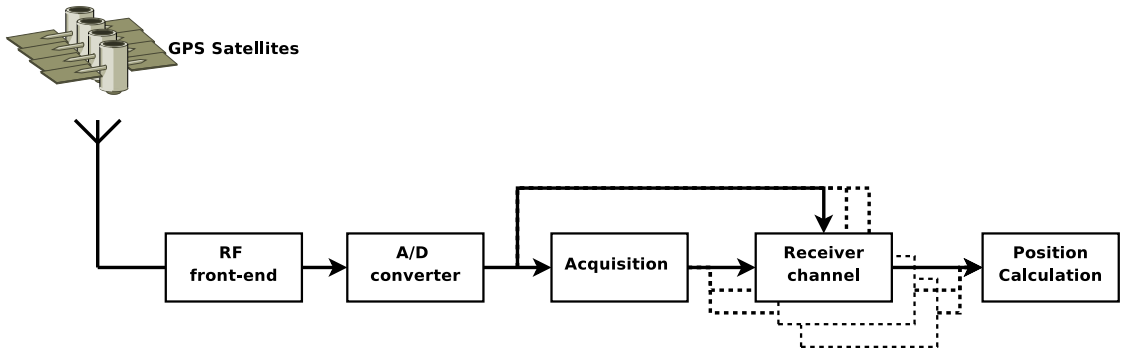


Figure 2.1: Block diagram showing a GPS receiver from RF front-end to data processing. In case of a warm start the Acquisition block is bypassed, and the data goes directly from the "ADC" to the "Receiver channels" - will in case of a cold start it will go from the "ADC" to the Acquisition block. There are marked multiple channels on the "Receiver channel"-block - this is done to illustrate that a finale receiver is able to receive signals from multiple satellites in parallel

The RF-front end and the ADC block receives the GPS signals and converts them to digital signals - after the ADC block is there two chooses, either send the signal though the Acquisition block or bypass it. The Acquisition block is used in what is called cold start described later in Section 2.3.4, while bypassing the block is called warm start. After the cold or warm start the next block are Receiver channels, this block can consist of multiple block in parallel and are able to receive the GPS signal from different channels - The more blocks in parallel, the more different channels can be received simultaneity. The last block in the receiver is the position calculation block - the block is based on received data able to extract different informations from the GPS satellites, the most common is the position and the distance to the different satellites and based on thus information is the block able to calculate the position of the GPS receiver.

2.3 GPS - Signals & coding

The GPS satellites are placed on the Medium Earth Orbit[1] at an altitude of approx 20200 km.

2.3.1 GPS operations frequencies

The satellites are made for both military and civil applications - This is made by using different frequencies in the L-band[**?**] and different codings. For the civil applications there are different frequencies based on one oscillator F_0 [**?**]:

$$\begin{aligned} F_0 &= 10,32 \text{ MHz} \\ L1 &= 150 \cdot F_0 = 1575,42 \text{ MHz} = 1,57542 \text{ GHz} \\ L2 &= 120 \cdot F_0 = 1227,60 \text{ MHz} = 1,22760 \text{ GHz} \end{aligned}$$

There are more frequencies for civil use, but $L1$ and $L2$ are the most widespread. The most focus for the project is on the signals on the L1 band - therefore L2 and other civil bands are not be used in this project.

[1] Medium Earth Orbit[**?**]: (MEO) Between 2000 Km and 35786 Km

2.3.2 GPS Signal Coding

The GPS system uses different coding systems - The C/A code[2] and P code[3] are used at the two frequencies L1 and L2 for different applications in GPS. Below is a list of which ones are used on the different frequencies:

- **L1**
  - C/A code
  - P code
- **L2**
  - P code

The project focuses on using the L1 frequency and the C/A code, hence there will not be a description of the P-code.

C/A Code

The C/A code is based on PRN[4] function and each GPS satellite has an unique C/A code. The C/A codes have a length of 1023 bits, but in this case, a bit is called a ”chip” - the reason for this is to indicate that the single bits do not hold any value or information - only as a part of the complete code do they have a value. The code repeats itself every millisecond and a single chip duration is approximate $\sim 1\ \mu$ S.
Below some facts to summon up about the C/A code in the GPS system:[**?**]

- Chipping rate: 1.023 MCPS[5]
- Length: 1023 Chips
- Chip duration: $\sim 1\ \mu$ S
- Wave length: $\sim$ 300 meter
- Repeated every millisecond
- 32 different sequences of C/A codes each assigned to one of maximum 32 GPS satellites

Navigation Data

When a receiver has ”tuned” in on a satellite by use of the L1 frequency and the C/A code, it can receive the navigation data package within the signal. The navigation data is mixed with the C/A code, by use of BPSK[6] modulation, which means that when a chip changes from high to low the phase changes 180°. An example can be seen in Figure 2.2.

The navigation data have a very low data rate - compared with the chipping rate of the C/A code - The navigation data is transferred with 50 bps[7] [**?**]. The navigation data contains information about the satellites time, position and health - and the data can be used for positioning of a receiver. For positioning, the data from four satellites are needed[**?**].

[2] Coarse Acquisition or Civilian Acquisition Code (also known as ”Clear Access”)
[3] Precise, Protected or Precision Code
[4] Pseudo Random Noise
[5] Mega Chips Per Second
[6] Binary Phase Shift Key
[7] bits per second

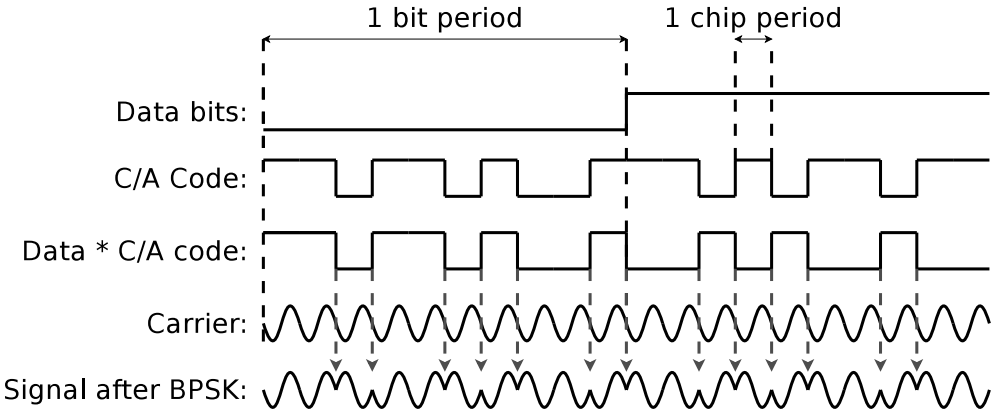


Figure 2.2: Example of Navigation data - The Data bits is low speed. The data and the coding is multiply, which means that if data is ”high” and code is ”high” equals signal is ”high” while data ”low” and code ”high” equals signal ”high”. The signal is multiplied with a carrier, which gives an BPSK - when the signal changes from ”high - low” or ”low - high” the carrier is phase change 180°. Figure is redrawn from [**?**].

2.3.3 Multi channel receiving of GPS signal

Because the GPS signal is using a C/A coding on the signal this enables the possibility of receiving signals from multiple satellites using CDMA[8][**?**]. When a signal is coded individually from the source and the receiver knowns the codes, it is possible to receive several signals at the same time by sorting them, or in other words use ”Code Division Multiple Access”.

2.3.4 Cold- & Warm start

A GPS receiver system can be started in two different ways, cold or warm. The difference is the time it takes. A cold start for receiving means that the receiver does not know where it is and therefore does not know which satellites to look for. Therefore a cold started receiver needs to run an acquisition block to detect which satellites to look for and where to find them in both code and frequency. A warm start can skip the acquisition, and by using data stored in a data storage in the receiver on the approximate position of the receiver, it can find the satellites and begin the receiving of the signals. For an hand-held GPS receiver like the ones used for hiking, it is normally only needed to make a cold start if one takes it from one country to another while turned off.

[8]Code Division Multiple Access

3 GPS Acquisition

This chapter includes a description of the acquisition block in a GPS receiver, the GPS receiver was illustrated in figure 2.1. The chapter focuses on one algorithm, testing it in Matlab and introduces initial considerations of a possible implementation.

3.1 Acquisition

The task of the Acquisition block is to "find" the GPS satellites visible to the receiver. The satellites need to be detected in two domains - the coding and the frequency domain.

3.1.1 Coding

In section 2.3.2 the C/A code have been described - There are 32 different codes of 1023 clips, each unique for one GPS satellite. The acquisition block's task is therefore to detect where in the coding each satellite signal is - so it can be synchronised with the receiver channel blocks.

3.1.2 Frequency

The carrier frequency the GPS L1 is known - but due to the fact that satellites are not stationary, seen from the receiver, the receiver frequency drifts. This effect is known as the Doppler effect [**?**] - therefore the acquisition block needs to search the frequency domain in the area $\pm$ 10 kHz of the receiver frequency. The step size of searching the frequency domain is set to 500 Hz [**?**].

3.2 Acquisition Algorithm

Based on the knowledge of the coding and frequency, different methods for searching both domains can be explored. This project only focuses on one method - the serial search acquisition algorithm.

Serial Search Acquisition Algorithm

A straight forward way of implementing the algorithm is by using a serial search algorithm [**?**]. The number of chips in the code and the number of different frequencies are known - which mean

that different combinations of frequencies and codes can be calculated by a simple equation:

$$\text{"number of chips in code"} \cdot (2 \cdot \frac{\text{Doppler offset}}{\text{Search step}}) = \text{number of combinations}$$

$$\underbrace{1023}_{\text{Code phase}} \cdot \underbrace{(2 \cdot \tfrac{10000}{500})}_{\text{Frequencies}} = 1023 \cdot 41 = 41943 \text{ combinations} \tag{3.1}$$

Equation 3.1 shows the numbers of combinations there will be for each C/A code. Each code starts in one of 1023 places, this starting point will hereby be known as the **Code Phase**.

The serial search acquisition algorithm is illustrated in Figure 3.1 and the four main blocks are described in the following paragraphs.

Initialization

The initialization block contains a number of different settings for the external settings. The system runs through 1 ms of collected data - in the Initialization block is the setting of from where the data comes, from an ADC connected to a RF front-end or from earlier collected data from storage. Based on the sampling rate the data length of 1 ms data is determine and the C/A codes are loaded from data storage and fitted to the data length of 1 ms data.

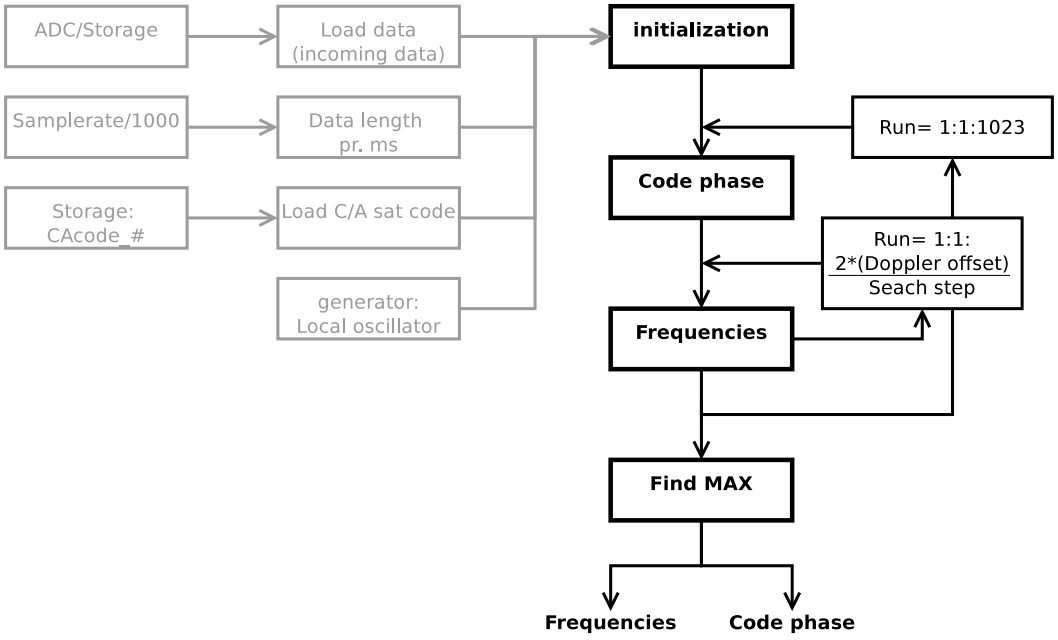


Figure 3.1: The figure shows a flow diagram over the Serial Search Acquisition Algorithm. The algorithm is composed of four blocks: Initialization, Code phase, Frequencies and Find MAX. The syntax used is from Matlab. Ex. Run=1:1:1023 means: start value=1 ; step size = 1 ; end value = 1023 ;

Code Phase & Frequencies

The code phase and frequencies blocks can be illustrated through a block diagram in Figure 3.2. The block diagram can be deviated in two loops, an inner one and an outer one. The inner loop is for the Frequencies block and the outer is for the Code phase. This means that for each code

phase, the frequencies block will be running for all frequencies[1]. When the inner loop has been executed, the outer loop runs through all the different combinations of chips in the CA code.

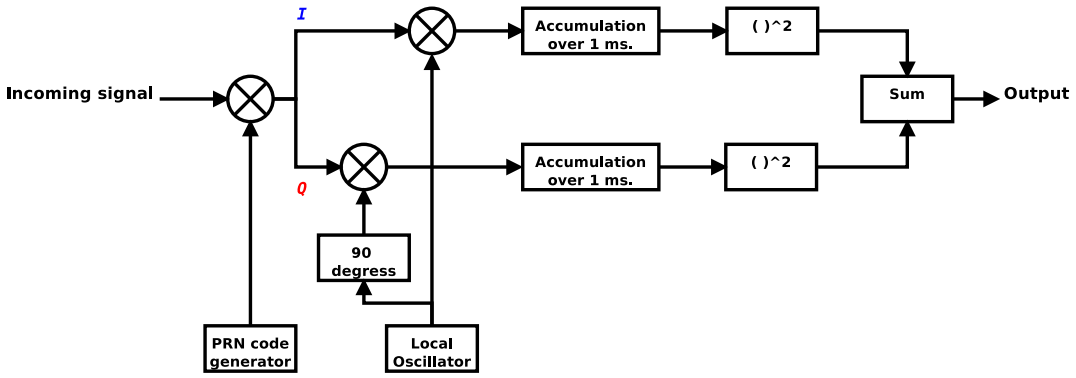


Figure 3.2: This block diagram shows the core of the acquisition algorithm - In the beginning the signal is mixed with the C/A code, then split and mixed with a sine and a cosine, then summed up and squared and at last summed. By changing the C/A code and the local oscillator, the blocks can identify the **frequencies** and the **Code phase**

Find MAX

The ”Find MAX” block is the final block. The input to the block is be an array of data generated in the ”Code phase” and ”Frequencies” block. The task of the block is to run through the data and to return the maximum of the ”Frequencies” and ”Code phase”. The maximum is then coded with a time stamp from the receiver and with the C/A code that corresponds to the execution of the algorithm.

3.3 Implementation in Matlab

This section contains a description of the Matlab implementation of the Serial Search Algorithm for the acquisition in the GPS receiver. The corresponding source code can be found in the attached DVD, appendix **??**. File *./code/Matlab Acquisition/Serial Search*. The Matlab test will be based on test data from the Aalborg University GPS development department [**?**].

3.3.1 From block diagram to Matlab implementation

The Matlab implementation was based on the flow chart in Figure 3.1 and the block diagram on Figure 3.2. The implementation included two loops, an outer for the code phase and an internal for the frequency. Within the internal loop would the algorithm design on the block diagram run. The Matlab implementation can be seen in appendix A. In the code can it be seen that the carrier are calculated as follows [**?**] :

$$\begin{aligned} expfreq &= exp(j \cdot 2 \cdot \pi \cdot fc(i) \cdot ts \cdot nn) \\ sine &= imag(expfreq) \\ cosine &= real(expfreq) \end{aligned} \tag{3.2}$$

Where:

- exqfreq : The carrier to the current frequency

[1] In this case, 41 times for IF $\pm$ 10 kHz with 500 Hz steps from the Doppler shift

- fc(i) : The frequency $\pm$ the Doppler steps, where (i) is the step
- ts : Sampling time
- nn : Total number of data samples

This calculation were in the first version calculated for every frequency and code phase, but by using Mathworks Profiler it were discovered that the code could be optimized by making the calculation at the side, and implement it as a lookup table into the design. The result was an execution time from 750 second to 170 second, which were an optimization of 4,4 times.

3.3.2 The C/A code

The C/A code repeats itself every millisecond, but with a sample rate higher than 1,023 MHz, the code signal has to be used at multiple samples.

Signal	$\frac{Samplerate}{1000}$ Bits/ms
Test Sample rate	11,999 MHz
Test Signal	11999 Bits/ms
C/A Code clock	1,023 MHz
C/A code	1023 Bits/ms

Table 3.1: Length of CA code versus signal length in bits/ms. Data is from test data from [?]

In table 3.1 it can be seen that the Code length in the test is shorter than that of the Test signals. This is due to the C/A code clock being lower than the Sample rate. There is a number of samples per ”chip” in the code. This can be compensated for by use of equation 3.3.

$$C = \text{round}(\frac{TS \cdot B)}{TC}); \tag{3.3}$$

where:

- C = Step(1 : length of signal in 1 ms)
- TS = $\frac{1}{\text{Sample frequency}}$
- B = Array with the size of data points in 1 ms data
- TC = $\frac{1}{\text{C/A code clock}}$
- Round() the elements to the nearest integers greater than or equal

The results of the test data can be seen in Figure 3.3. The figure shows that for every chip in the code, there is approximate 11 data points.

3.3.3 Results

It was possible to acquire some test data and C/A codes from NAVCOMM[2] at AAU. The data specifications and a fraction of the data can be seen in Figure 3.4.

The algorithm runs three times, one time per C/A code. The different C/A codes can be seen compared to each other in Figure 3.5

A graphical presentation of the Matlab results can be seen in Figure 3.6. The figure shows a 3-dimensional plot - The Y-axis is the frequency with $\pm$10 kHz to the L1 carrier frequency, the X-axis is the code phase from 1 to 1023 chips calculated as described in section 3.3.2 and the Z-axis is the power of the signal. As seen in the figure, SAT1 and SAT10 have a high peak, corresponding

[2]http://gps.aau.dk

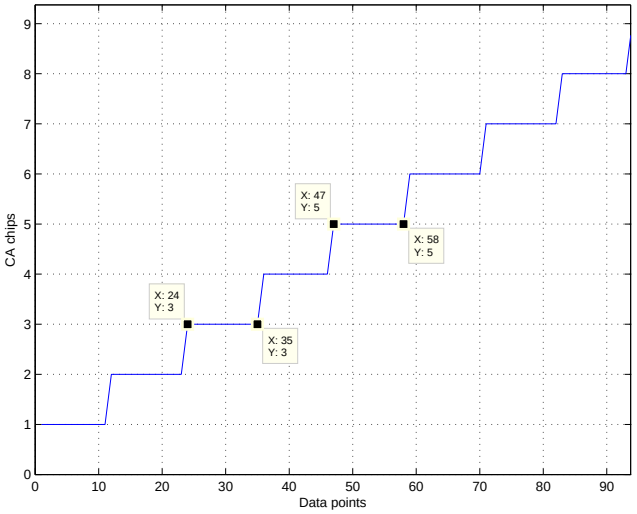


Figure 3.3: The figure shows a section of the final function for the test data. The step size is dependent on the sampling frequency. This case sample rate = 11,999 MHz

Test data specification		
Interminable frequency	3,563	MHz
Sampling frequency	11,999	MHz
Data pr. ms	11999	Samples

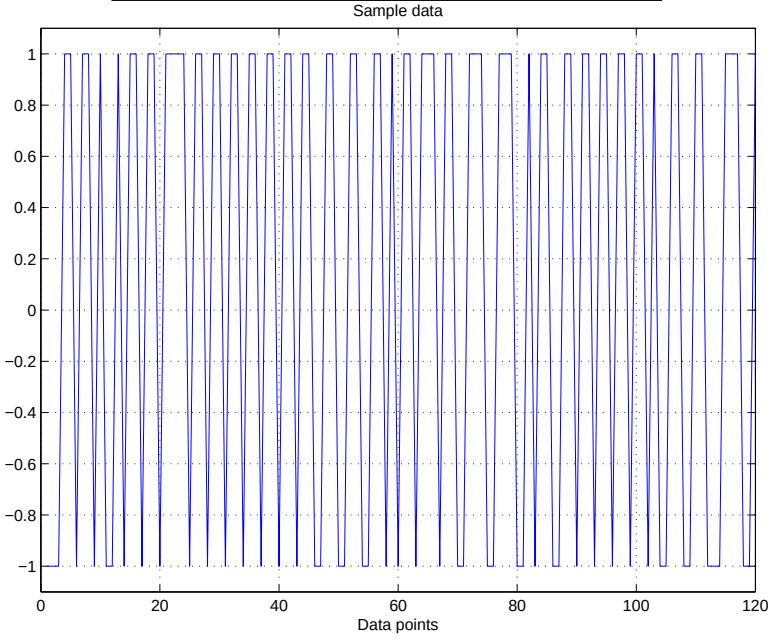


Figure 3.4: The figure shows approximately one percent of the GPS signal data and the table shows the data specification for the test data

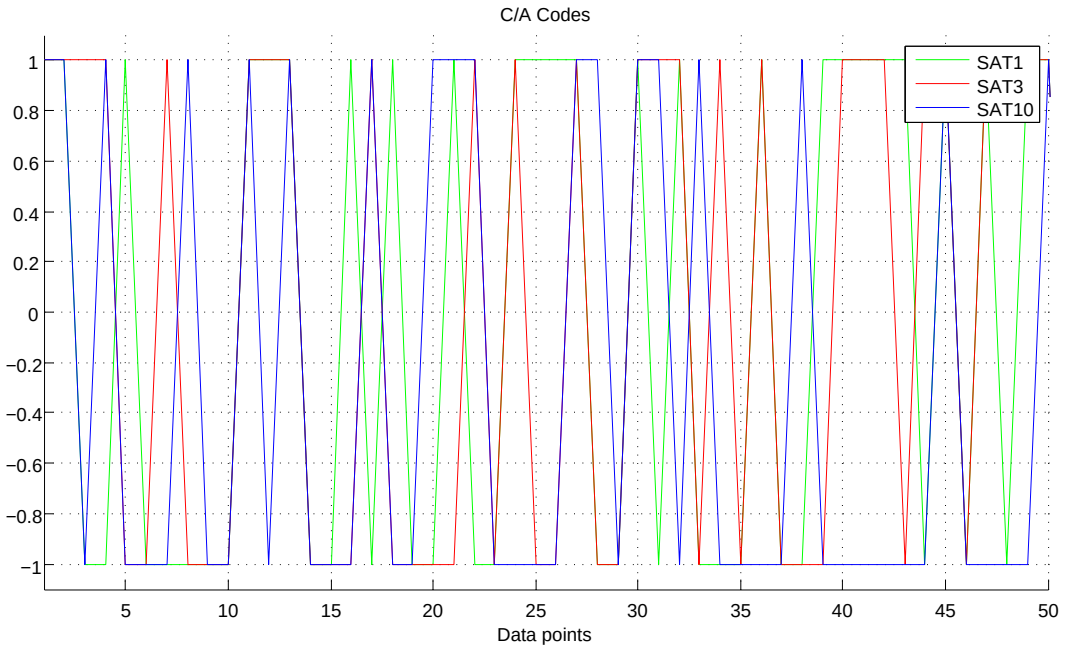


Figure 3.5: The figure shows three different C/A codes.

to the maximum code phase/frequency combination - while SAT3 is lower noise - This means that SAT1 and SAT10 are present with their C/A in the test data, while SAT3 is not. The maximum value on the Y-axis for SAT1 & SAT10 gives the ”FREQUENCIES” and on the X-axis it gives the ”CODE PHASE” which is needes for later tracking.

3.3.4 Test conclusion

The Matlab implementation gives satisfying results. It is possible to identify the satellites present in the test data. Therefore the design can continue with focus on implementation on a FPGA platform.

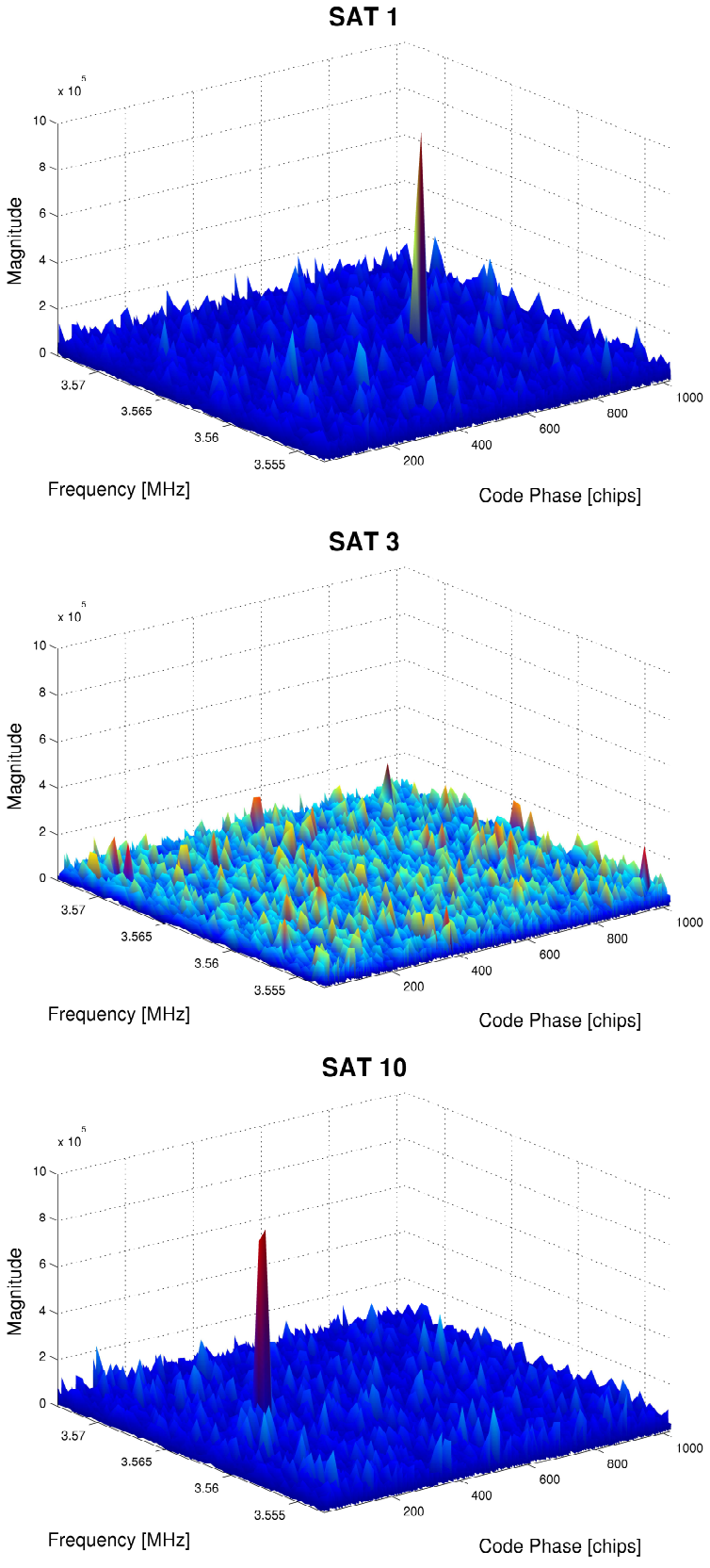


Figure 3.6: The figures shows the result of running the serial search algorithm on 1 ms sample data using three different C/A codes: Sat1, Sat3 & Sat10. The results show that Sat1 & Sat10 was visible i.e. the peak values for the receiver when the sampling was committed, while Sat3 was not.

4

FPGA implementation

This chapter contains a description of the mapping method and tools, from the Matlab implementation to an FPGA implementation. There are several ways to map an application onto the algorithmic domain and from there to the architecture domain. In the case of this project, the application have been mapped onto the algorithmic domain by a Matlab implementation. Based on the structure of the Matlab implementation, two algorithms for FPGA implementation are developed - One based on serial principle and one using a parallel principle.

4.1 Tools for mapping the algorithm onto the architecture

The chosen architecture is an FPGA. This mean there are different tools for implementing code onto the architecture. One way would be to analyse the algorithm and then develop a VHDL-implementation onto the FPGA using CAD tools for implementation. This would be a low level approach. Another way would be to use CAD[1] tools for both design and implementing of algorithms; this is a high level approach. A FPGA from the company Xilinx been chosen for this project, hence the development tool Xilinx ISE[**?**], Mathworks Simulink[**?**] og Xilinx System Generator[**?**] can be used.

4.1.1 Xilinx ISE

Xilinx ISE is a development tool made by Xilinx for their FPGAs. Figure 4.1 shows a diagram of the flow though the different build-in functions in ISE.

[1] Computer Aided Design

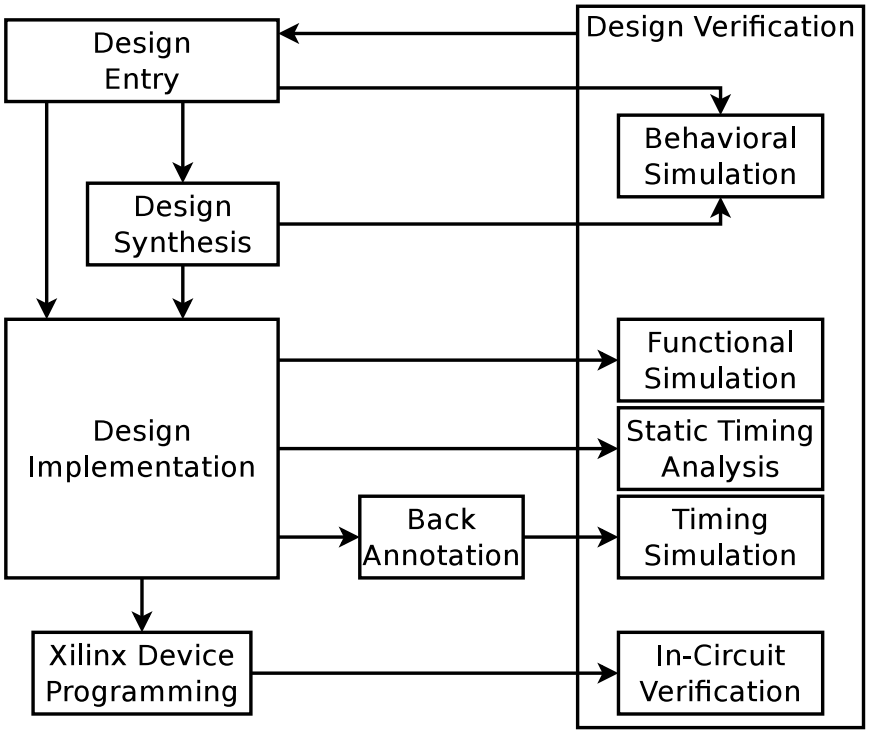


Figure 4.1: The figure shows the trough though the different states and functions build in to the development tool. The left side of the figure shows the high level functions - While the right side includes different design verifications with a feedback to the design

The tool contains a number of different tests, spread over different levels.

1. **Synthesis**
   - The synthesis makes an estimate of the need resource and speed for implementation on a selected FPGA. All results here are rough estimates.
2. **Implementation**
   - The Implementation include a mapping and routing inside the FPGA, which also means that the results are more accurate than the synthesis - amount the features more detail timing analyses, resource estimates and power analyse.
3. **Device Programming**
   - In the Device programming the output will be a programming file ready for transfer to the selected Xilinx FPGA

4.1.2 Xilinx System Generator

Xilinx System Generator is a plugin tool for Mathworks Matlab Simulink. In Simulink is it possible to make models based on different blocks - the plugin from Xilinx adds a number of different blocks with functionalities able to be executed within an Xilinx FPGA. The Xilinx blocks are unique for Simulink, normal Simulink block do not work directly with the Xilinx blocks and the data have to be ”gated” between Xilinx and Simulink blocks - The idea is then to make the algorithmic part of the model and as much of the control in Xilinx block - and the the outer shell by Simulink blocks. An example of Simulink and Xilinxs blocks can, in a simple working example,

be seen in figure 4.2. The Simulink and System Generator models can finally be exported to a Xilinx project, which is compatible with Xilinx ISE. It is also possible to generate the bit stream directly from Simulink/System Generator, in that case ISE is called transparently seen from the users point of view.

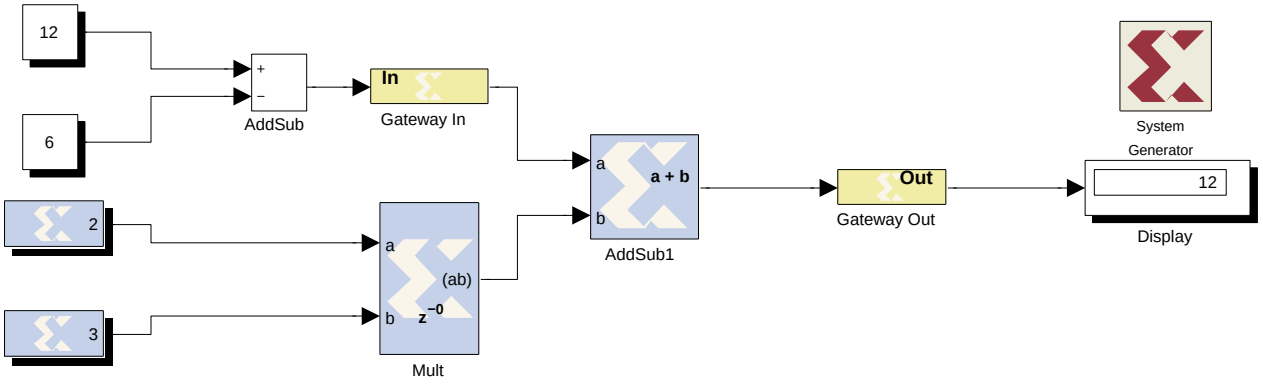


Figure 4.2: The figure shows an example of a model based on both Simulink and Xilinxs blocks. The Xilinx code is based on an input for the ”Gateway in” and output from ”Gateway out” - between the gateway the Xilinx code is executing and outside the normal simulink blocks can be used. The ”System Generator” block will be present in all system generator designs - within the block different settings for a specific FPGA can be set, and thereby can the model be exported to a ISE project for analyse and implementation.

4.2 Algorithmic analysis - From Matlab to Xilinx Simulink model

The Matlab implementation is based on the block diagram seen in figure 2.1. Considering it as blocks in a processing unit, the block diagram has been reorganized into the structure seen in figure 4.3 - As seen in the figure, the blocks have been replaced by multipliers and ALU’s. The figure is based on inputs which are time/sample dependent, *i.e. input(t) is dependent on t*. For having a picture of the data flow and operations within the block diagram, a modified precedence graph have been made - A normal precedence graph shows the critical path from input to output though a number of operators. This modified version has been augmented with a ”W” block, this is a wait block able to accumulate data until a X number of samples, thereafter sending the final value to the rest of the blocks. The modified precedence graph can be seen in figure 4.4.

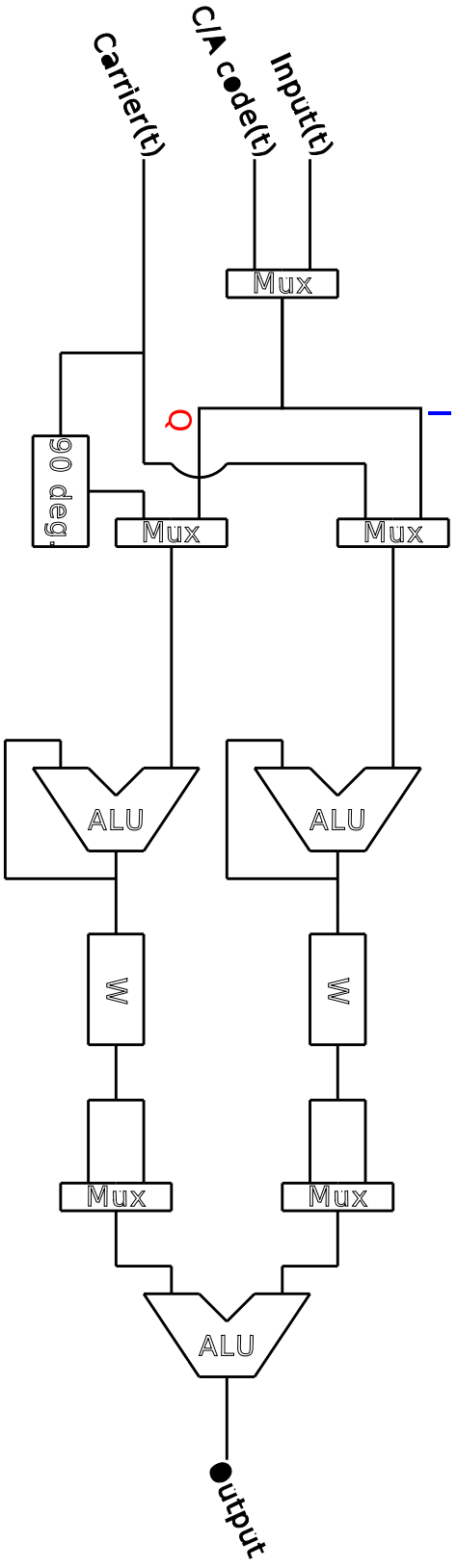


Figure 4.3: The Diagram shown in a reorganised version of the Serial Search Algorithm block diagram seen on figure 2.1. The input is depended of the time/sample. The block ”W” is a wait block, it is a sample based switch which will open after 1 ms of data samples have been accumulated. The operations of the block diagram can also be seen on the precedence graph on figure 4.4

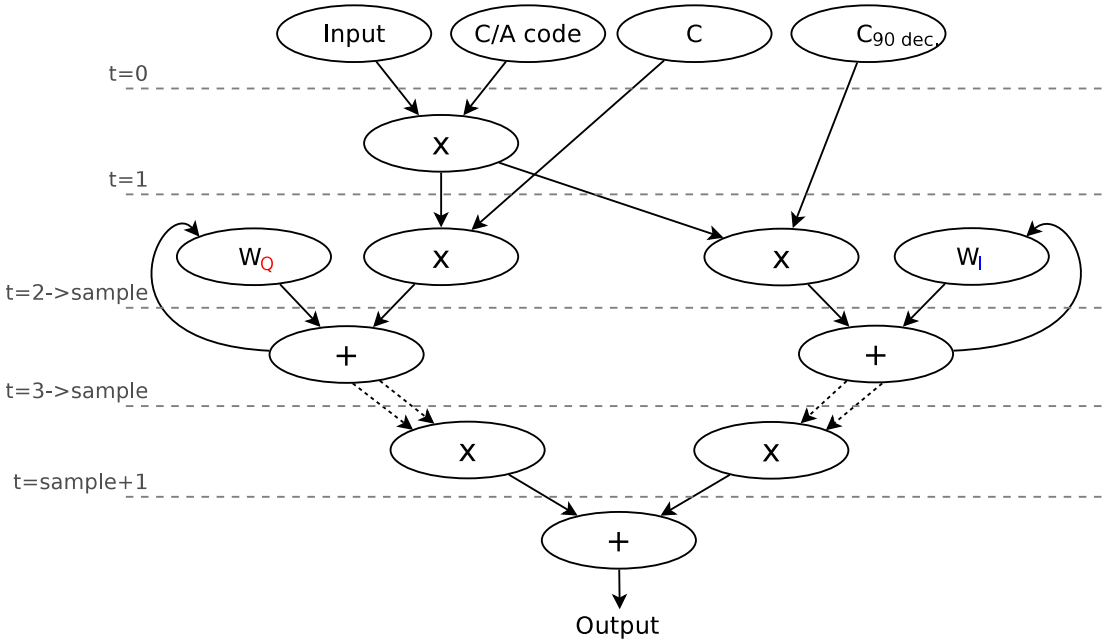


Figure 4.4: This modified precedence graph shows how the flow is within the block diagram. In a normal precedence graph is it possible to find the critical path - but in this modified version the ”W” blocks have been added. The ”W” blocks will hold the accumulated data until a X number of samples, the send the final value to the last part of the blocks. The modified version can show that the necessary number of processing units for this implementation will be 2 ALUs and 2 MUXs - this can be seen by looking at the time line, from to to button, the large number of ALUs and MUXs used at the same time in this design is 2 of each.

4.2.1 Simulink & System Generator: Timing & Word length

In the design phase from the ideal block diagram to a Simulink Xilinx model different adjustments were needed to make a realistic candidate for the FPGA implementation. The adjustments are listed below:

Timing:

Because the Xilinx blocks are made for implementation, some of the blocks include a latency by default. This means that in a system with multiple data inputs, all input time must be synchronised. To make the synchronisation, there are different methods - one method is to remove all latencies from the blocks, but this method would bring the model back towards an ideal environment and away from a realistic implementation. Another method is to add delay blocks in the model, to match delays from other blocks, thereby incorporate the default latencies from ex. multiplier blocks. The last method was chosen and implemented on the models (delay blocks can be seen on all Xilinx models).

Word length:

The Xilinx blocks have a default word length. At the Gateway In blocks, the input signal is as standard set to a value between -1 and 1, this set to the default value. The only place where the default word length does not fit the calculation values is within the accumulators. Here the value

is set at the recommended value in Simulink. Further elaboration on this topic is out of the scope for this project, and will be set as future work for the project.

4.2.2 Serial Method

The serial method is similar to the Matlab implementation. It is based on the block diagram, seen in figure 2.1 and the modified version in figure 4.3. It runs through the 41 different frequencies from the Doppler shift, one by one, for each of the 1023 different versions of each C/A code and finally for a GPS receiver the code will have to be run 32 times for the 32 different C/A codes used in the GPS network. This ends with a final number of executions of 1.362.176 times[2].

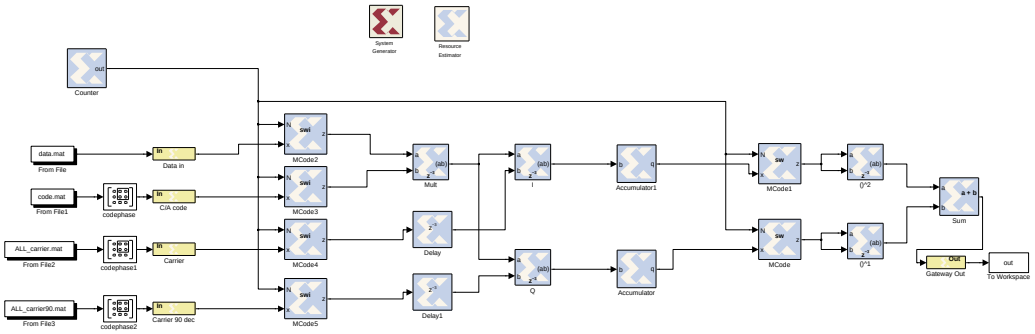


Figure 4.5: The figure shows the Mathworks Simulink & Xilinx System Generator model for the serial method. A larger figures can be found in AppendixB.1 on figure B.1

In other words the Simulink model build by Xilinx block replaces the code in the first Matlab implementation. The Simulink Xilinx model can be seen in figure 4.5.

4.2.3 Parallel Method

The second method implemented in Simulink Xilinx blocks is using a parallel principle. The parallel method is based on the serial method; the main Xilinx models is the same as the serial method. The difference is a simultaneous handling of the 41 models - the handler has been designed to simultaneous process of all frequencies for each code phase. The model has been divided into multiple models - an outer, a main and 41 subsystem block, they can be seen on figure 4.6. The parallel method will run though the 1023 different version of each of the 32 different C/A codes - this gives a finale number of execution on 32.736 times.

[2] 41 frequencies · 1023 different code options · 32 codes

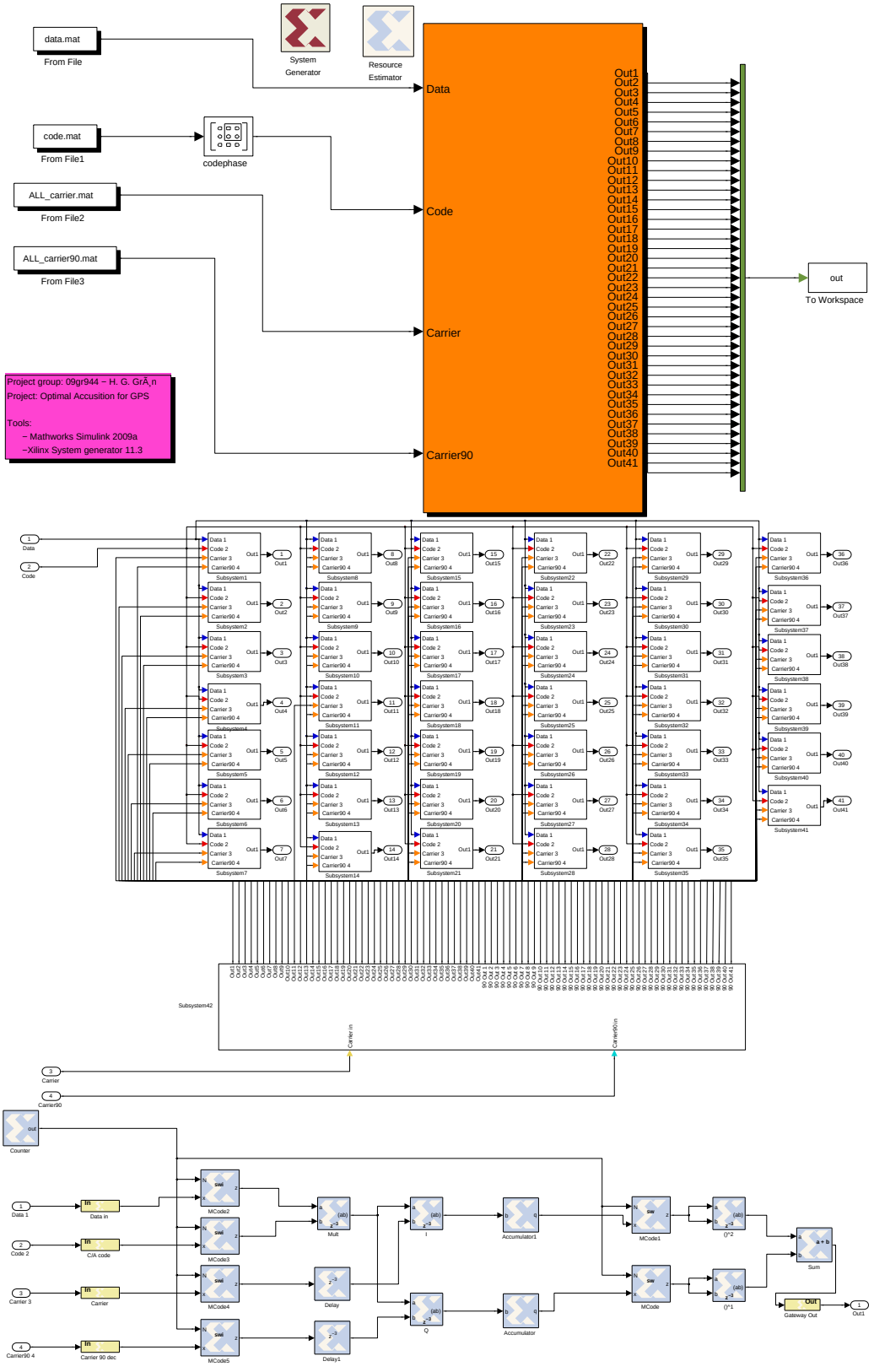


Figure 4.6: The figure shows the Mathworks Simulink & Xilinx System Generator model for the parallel method. The first figure shows the ”Outer block”, the middler figure shows the ”main block” and the last figure shows one of 41 subsystems. Larger figures can be found in Appendix B.2 on figure B.2, B.3 & B.3

4.3 Numerical testing of the models

Both models have been tested with the same test data, as used in the Matlab implementation (see section 3.3) and the final numerical output of the Matlab implementation, the Serial and the Parallel method ended up to be the same. Therefore it can be verified that the two methods are valid within the Simulink environment. The numerical values were tested using a little Matlab script, which ran though all values of the first Matlab implementation, described in chapter 3.3, and the result from the Simulink/System Generator model. The scripted returned a ”1” if the numbers were equal/true and a ”0” if they were unequal/falls. The result ended for both method as all ones, which means that the numerical values of the models and Matlab implementation are equal - This means that the models can be taken to the next phase. The next phase will be to generated the Xilinx Simulink model into a Xilinx FPGA project.
For the project a Xilinx Virtex-5 device number: 5vfx70tff1136-3[**?**] has been chosen. The device has been chosen because it is currently being used by Aalborg Universities GPS development in they SoftGPS project[**?**].
The generated Xilinx projects and the models for Mathworks Simulink can be found on the appendix DVD(*./code/Xilinx/*). The next chapter describes the synthesis process and results.

5 From model to FPGA implementation

*This chapter contains a description of the tool and results of the process - going from at Simulink model to a Xilinx FPGA implementation. The two methods, tested for their numerical properties, have been taken to the next phase - loading the generated Xilinx project into Xilinx ISE[**?**], for synthesizing the design and analysing the synthesis results.*

5.1 Synthesising the Projects

The two projects were synthesised in Xilinx ISE. Each of the results will be discussed in the following subsections. The Xilinx projects can be found on the appendix CD(*./code/Xilinx*).
The synthesis reports will give an estimate of the needed resources in the FPGA for implementation, a time estimate for the execution of the algorithm seen from the critical paths point of view and the corresponding clock speed for this time estimate. The results of the synthesis have been summed up in a ”Final Report” file which will be used as source for obtaining the results. Both method have been synthesised with a flag set for ”Speed” optimisation.
As a verification of the design can Xilinx ISE give a estimate of the design in functions blocks in the FPGA. A diagram showing the serial model implemented can be seen on in appendix C.1 on figure C.1 - compared with the Simulink/System Generator model, seen on the figure B.1, it can be concluded that the System Generator model and the ISE generated is very simulate in block, paths and design. The figure showing the parallel method have not been put into the report because the size of the design on paper, therefore can it be found on the appendix DVD(*./Design/*).

5.1.1 The Serial Method

The serial method’s final synthesis report can be seen in appendix C.1. In the next paragraphs the results of the resources and timing are summarized and described.

Resources

Summary of resources used for Serial implementation			
Device utilization	Used units	Total number	Used of total
Number of Slice Registers	132	44800	0%
Number of Slice LUTs	298	44800	0%
Number of bonded IOBs:	166	640	25%
Number of BUFG/BUFGCTRLs:	1	32	3%

Table 5.1: The table shows a summary of the resources used for the serial implementation, the full list can be found in appendix C.1. The serial implementation easily fits the selected FPGA

Timing

Summary of timing used for Serial implementation		
Clock Frequency	381,869	MHz
Total Routing	0,44	ns
Total Logic	2.179	ns
Total time for one period	2,619	ns

Table 5.2: The table shows the timing estimation from the synthesis, the full timing can be seen in appendix C.1. The timing result is only a rough estimation, for more accurate timing estimates Xilinx ISE should be used for "place-and-routing" of the design

A time estimate for the execution of the serial method can be calculated as follows:

$$\text{"Total time"} \cdot \text{ "Number of Doppler steps"} \cdot \text{ "Chips in code"} \cdot \text{ "Code sequences"} \tag{5.1}$$
$$2,619 \text{ ns} \cdot 41 \text{ frequencies} \cdot 1023 \text{ chips} \cdot 32 \text{ codes} = 3,515 \text{ ms per execution}$$

Synthesising conclusion for Serial implementation

The synthesis for the serial method was a success. As seen in table 5.1, the design fits within the size of the chosen FPGA. The timing calculation in equation 5.2 is made for one execution of the algorithm - which have be based on 1 ms data, the calculation estimates an execution time of 3,515 ms, this timing is without taking file handling into account. Based on this calculation and the resources requirements it can be concluded that the serial method is ready for implementation, but it will not be able to run in real time - therefore, some control taking this into account needs to be made for a final implementation. One possible way could be only to take a sample for every 5 ms, this would give time for execution of the design plus some data handling.

5.1.2 The Parallel Method

The parallel methods final synthesis report can be seen in appendix C.2, resource usage summary can be found in table 5.3 and a timing summary in table 5.4.

Resources

Summary of resources used for Parallel implementation			
Device utilization	Used units	Total number	Used of total
Number of Slice Registers	6560	44800	14%
Number of Slice LUTs	14514	44800	32%
Number of bonded IOBs:	7914	640	1236%
Number of BUFG/BUFGCTRLs:	1	32	3%

Table 5.3: The table shows a summery of the resources used in parallel implementation, the full list can be found in appendix C.2. Most of the design easily fits in the selected FPGA, apart from the IOBs.

Timing

Summary of timing used for Parallel implementation		
Clock Frequency	340,461	MHz
Total Routing	0,440	ns
Total Logic	2,497	ns
Total time for one period	2,937	ns

Table 5.4: The table shows the timing estimation from the synthesis, the full timing can be seen in appendix C.2. The timing result is only a rough estimation, for more accurate timing estimates will Xilinx ISE should be used to run the "place-and-routing"

A time estimate for the execution of the parallel method can be calculated as follows:

$$\begin{gathered} \text{"Total time"} \cdot \text{ "Chips in code"} \cdot \text{ "Code sequences"} \\ 2,937 \text{ ns} \cdot 1023 \text{ chips} \cdot 32 \text{ codes} = 0,0961 \text{ ms per execution} \end{gathered} \tag{5.2}$$

Synthesising conclusion for Parallel implementation

The parallel implementation was 41 times larger than the serial method by design. This size reference refers to the fact that the parallel design executes all 41 serial models at the same time, one for each of the 41 Doppler steps. This was no problem for the synthesis but the result of the larger design can be seen in the summary of the needed FPGA resources in Table 5.3. With one exception all resources needed were available on the FPGA - unfortunately the design uses 1236% of the available I/O ports in the FPGA. This means that design can not be mapped to the FPGA, and therefore some changes are needed for reducing the number of IOBs used.
If the focus is changed to the timing and the calculation of the execution time, seen in equation 5.2, it can be seen that the parallel method is 36.6 times faster that the serial method, based on the equation 5.2 & 5.2. With an execution time of 10% of the data sample it should be possible to execute the design in real time, but this require a more accurate timing analyse involving "place-and-routing", which means that the IOB problem should be solved.

5.2 Intermediate conclusion

It can be concluded that the serial method in the current form is ready to be taken to the next phase, which will be at first "place-and-routing" and the generations of bit stream file for the FPGA. On the other hand the parallel method can not be mapped before the IOBs problem has been solved. One solution for reducing the IOB use in both designs can be found by analysing the full synthesis

report[1]. The synthesis reports comments on a large number of IOB used for ”black box”-blocks - theirs blocks are related to the ”Mcode”-blocks seen on the models[2]. This means that the Matlab block have not been implemented optimally into the FPGA project, a more optimal solution could be to replace the Mcode-blocks with a VHDL design with the functionality.
The serial method have been successfully mapped, placed-and-routed and the bit stream file has been generated- because the generation was successful it can be concluded that the serial method is ready to implementation onto the FPGA, but for testing it will still need a design of a file handler for the input data for the FPGA.

[1]The full report can be found on the Appendix DVD, Appendix **??**
[2]Serial: see appendix figure B.1 - Parallel: see appendix figure B.4

6 Conclusion & future work

6.1 Conclusion

The project started with the motivation from the Danish satellite development company, GOM-Space. Their interest was to put GPS receivers onto small satellites - this wish from the company combined with an interest in FPGA implementations onto satellites ended up with a project well suited for the ASPI specialisation. The problems were then limited to focus on the acquisition block within a GPS receiver with focus on implementation onto a FPGA platform and with execution time in focus.
The problem was first analysed through a Mathworks Matlab investigation for verification of the algorithmic design, then a tool from the FPGA producer Xilinx called System Generator was used to map the the algorithm onto a Xilinx Virtex5 5vfx70tff1136-3 FPGA. Different methods have been developed, one build on parallel and one on serial principles, both with executions time in focus.
Based on the results from the Matlab implementation and the two System Generator models, it appears that the algorithms performs as expected. This conclusion is based on the fact that both methods are able to identify the satellites in the data and the numerical properties are equal for the models and the Matlab implementation.
After verifying the two method numerically they were synthesised to a Xilinx FPGA. Based on the results of the synthesis it can be concluded that the serial method can be fitted onto the selected FPGA with no problem, whereas the parallel method used 12 times the number of IOBs available on the FPGA. By this it can be concluded that the serial method is ready for final steps for the FPGA implementation, while the parallel method will have to be redesigned for an implementation.
Seen from another point of view, the parallel method will be able to run in real time on the selected FPGA, whereas the serial method will not. Based on this timing fact, it can be concluded that the parallel method should be redesigned, with focus on a lower use of IOBs.
Therefore the final conclusion is that the problem of time optimisation of start condition in the acquisition block of a GPS receiver, defined in section 1.2, has been investigated and ended with two different methods - where one is ready for implementation and another is not. It can also be concluded that the only one of the methods meets the optional problem of a real time implementations onto the FPGA platform.

For a final working implementation both methods will need a data handling module and finally should be tested on a FPGA platform; These tasks are described in the following section - Future Work.

6.2 Future work

The future work section is decided into two subsections - one with short term perspectives and one over long term. The short terms have a time frame less than one semester work, while the long term has an estimated time frame of minimum one semester work or more.

6.2.1 Short therm work

For implementation of each of the two designs, a data handler will be needed, therefore will one of the short term tasks will be to design a data handler which can manage the input and output of the implemented methods. This data handler could be make to run on a PC as a co-simulation to the FPGA or as a implementation onto the FPGA with either the serial or the parallel method.
Another task in the short term part would be to test the two models and the Matlab implementation with more data sets that the one used for this report - the data could come from the Aalborg University GPS Department or from other GPS departments in the world.

6.2.2 Long term

The long term tasks will contain different ideas for further investigations. The two methods are both designed with a black box implementation in the model, this model uses a lot of IOBs, which is the problem with the parallel method - therefore a more optimal way to do this task in model will be recommended as a task seen in a long term contains.
Another task could be to find the optimal word length used within the models, instead of using the recommended ones from Simulink. Finally, it be could interesting to use different optimization techniques to try to make more optimal models, with focus not only on execution time but also on power usage.
In this project the focus has been limited to an earth based GPS receiver, a task with a long term perspective could also be to change this from earth to a near earth orbit - which the satellites from GOM-Space are placed in. This might induce a focus on the Doppler steps used in the design.
So to sum up, a long term task could be to optimize the design from Earth to near Earth orbit, this means to focus on power optimization and on the receiver being on a fast moving satellite in space. It is expected that these ideas will be investigated next semester in my master thesis.
A very long term project would be to implement the design acquisition block onto a FPGA platform, along side all the other blocks used for a GPS receiver - optimized with execution time and power in focus, and place it all on a payload board for a GOM-Space satellite.

A

Matlab implementation

```
%% Serial Search Acquisition for use in the GPS standard
% % Project group 944 - Helge G. Gron
%
% BLOCK DIAGRAM
%                              ---------- X ------ intergration ------ ( )^2 ----
%                             |          |                                       |
% Incoming signal>---- X ------|          |                                   SUM ---- >Output
%                      |       |          |                                       |
%                             ---- X ------------ intergration ------ ( )^2 ----
%                      |             |     |
%                   PRN Code         90    |
%                   Generator      degree  |
%                                  |     |
%                                  Local
%                                  Oscillator
clc; clear; close all;

svnum=10; %%%% SELECT C/A code for GPS Satellite

% initial conditions
load ../testdata/data.mat
data=double(data');

fs=11.999e6;    %sampling freq
ts=1/fs;        %sampling time
n=fs/1000;      %data pt in 1 ms
nn=[0:n-1];     %total no. of pts
fc0=3.563e6;    %center freq without Doppler =IF

% Load PRN/CA/GOLD code for Satellite
gold_rate = 1.023e6;                %Gold code clock rate in Hz
ts=1/fs;
tc=1/gold_rate;
b=[1:n];
c=ceil((ts*b)/tc);

filename=sprintf('../testdata/gold%i.mat',svnum);
load(filename);

if length(code) ~= length(data)
fprintf('ERROR \n')
end

% Mulitipling local code with Incoming Data
```

```
c(length(data))=length(code(:,1));
load('expfreq.mat')

for q=1:1:(length(data))
    qq=c(q);
    singlecode=double(code(qq,:));
    x=singlecode.*data;

    clc
    Fremgang=round((q*100)/length(data))

    % Search though the frequens area +/- 10 KHz of IF
    for i=1:41
        fc(i) = fc0 + 0.0005e6*(i-21); %runs the +/- 10 kHz around Intermidie frequnzy
%       expfreq=exp(j*2*pi*fc(i)*ts*nn);
        expfreq=out(i,:);   %tabel opsalg laver ved hjelp af expfreq.m
        sine= imag(expfreq);            % generate local sine (direct)
        cosine= real(expfreq);  % generate local cosine (90 degree = cosine)
        I = sine.*x;
        Q = cosine.*x;
        Isum=sum(I)^2;
        Qsum=sum(Q)^2;

        IQsum(i,q)=Isum+Qsum;
    end
end


%% Results
result=IQsum;
[peak codephase]=max(max(result));
[peak frequency]=max(max(result'));

frequency = fc(frequency)

codephaseChips = round(1023 - (codephase/11999)*1023)
%codephaseChips = round((codephase/11999)*1023)

%% Ploting
gold_rate = 1.023e6;                    % Gold code clock rate in Hz
ts=1/fs;
tc=1/gold_rate;
b=[1:n];
c=ceil((ts*b)/tc);

figure(1)
x_axis=c;
y_axis=fc/1e6;
s=surf(x_axis,y_axis,result);
set(s,'EdgeColor','none','Facecolor','interp');
axis([min(x_axis) max(x_axis) min(y_axis) max(y_axis) 0 10e5]);
caxis([0 max(max(result))]);
xlabel('Code Phase [chips]');
ylabel('Frequency [MHz]');
zlabel('Magnitude');
text=sprintf('SVN %i',svnum);
title(text);
```

B

System generator Implementations

B.1 System generator: Serial implementation

- Figure B.1

B.2 System generator: Parallel implementation

- Figure B.2
- Figure B.3
- Figure B.4

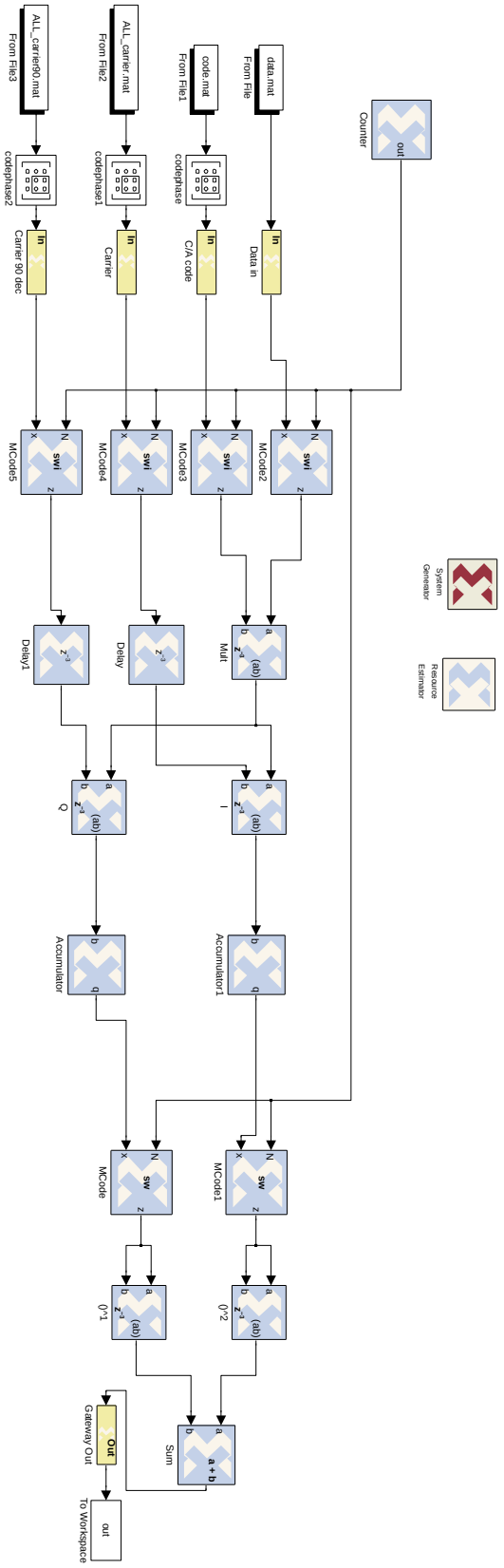


Figure B.1: The figure shows the Mathworks Simulink & Xilinx System Generator model for the serial method. The figure can be seen in the report on figure 4.5

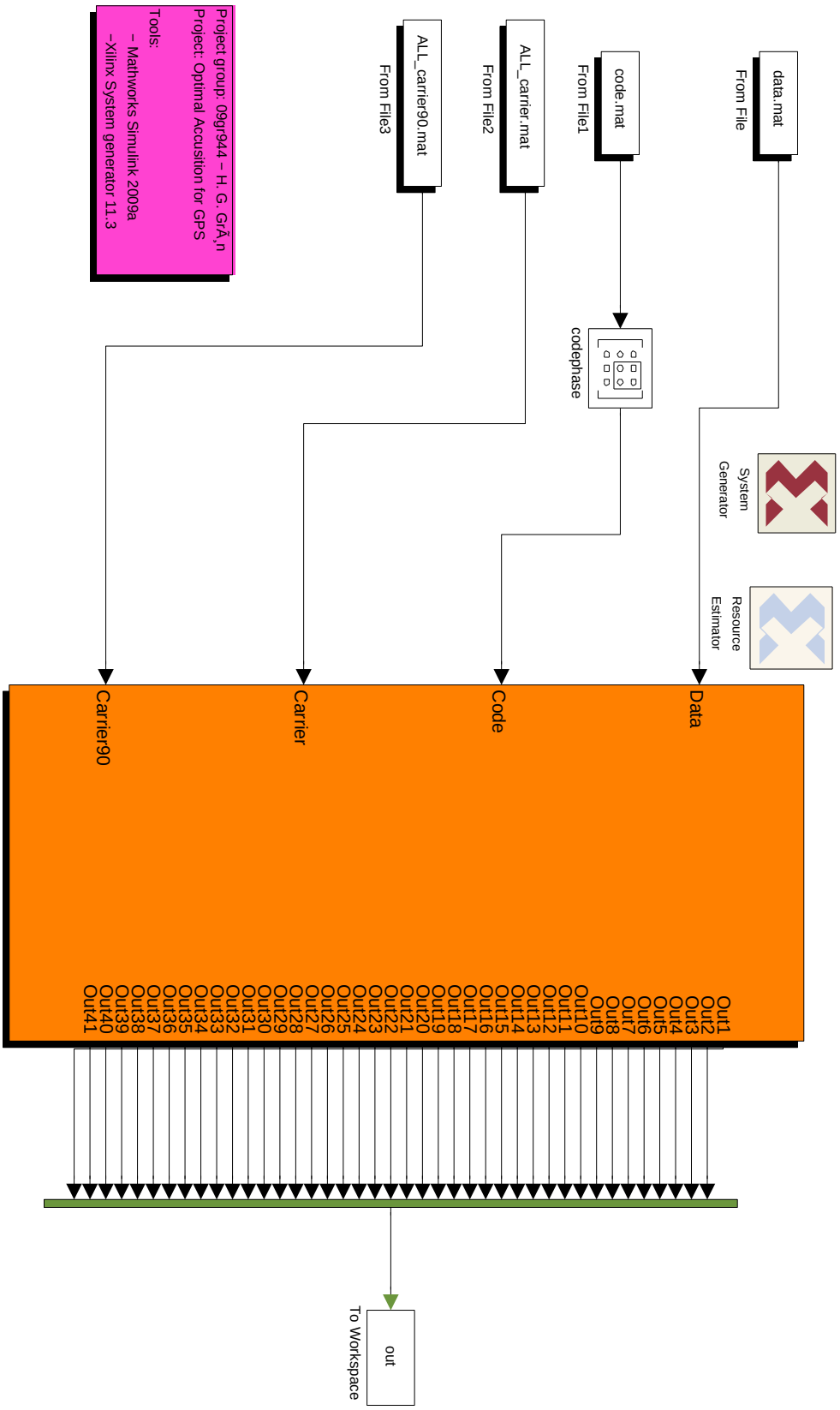


Figure B.2: The figure shows the Mathworks Simulink & Xilinx System Generator model for the parallel method. The figure shows the ”Outer block”. The figure can be seen in the report on figure 4.6

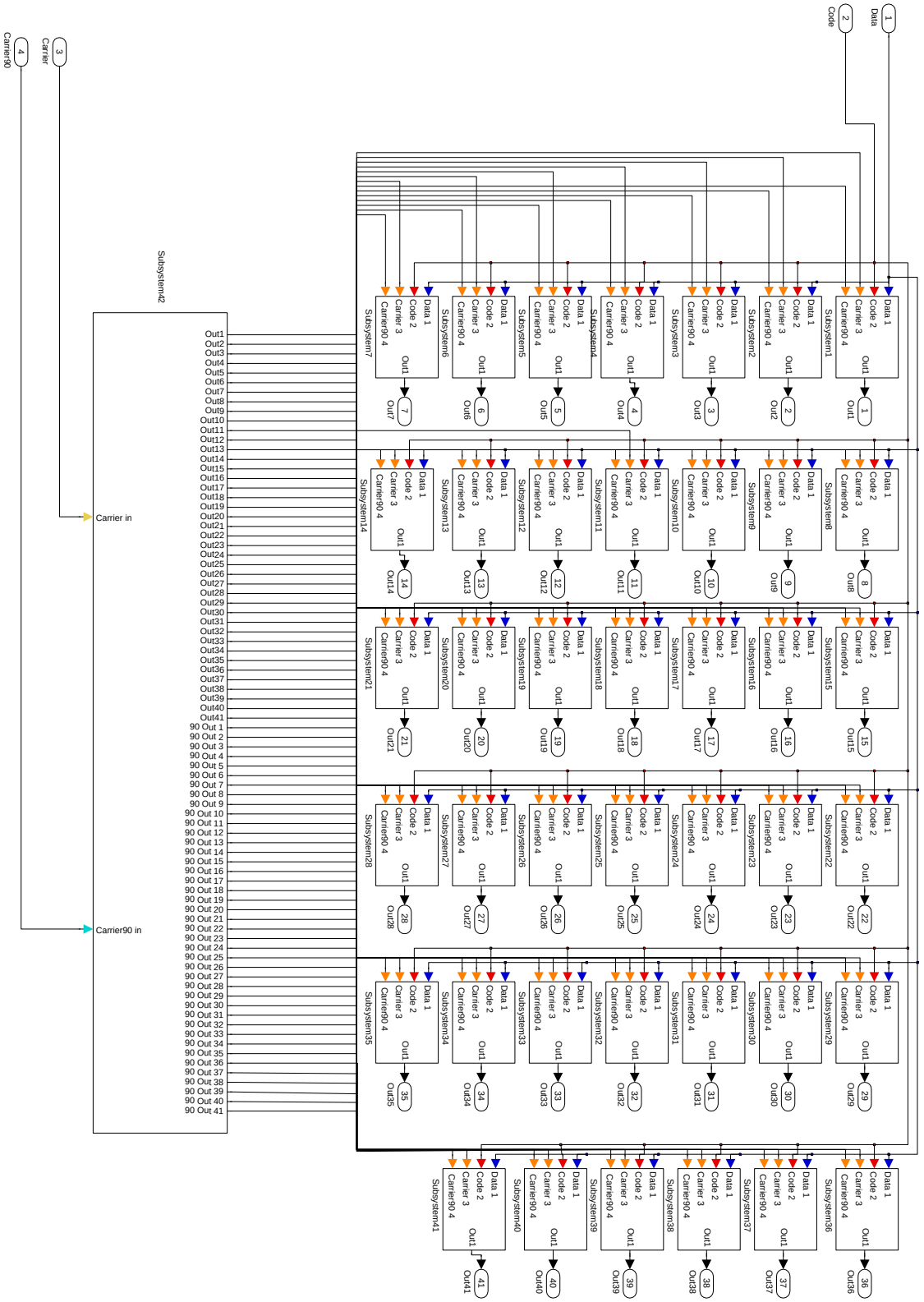


Figure B.3: The figure shows the Mathworks Simulink & Xilinx System Generator model for the parallel method. The first figure shows the ”Outer block”, the midder figure shows the ”main block” and the last figure shows one of 41 subsystems. The figure can be seen in the report on figure 4.6

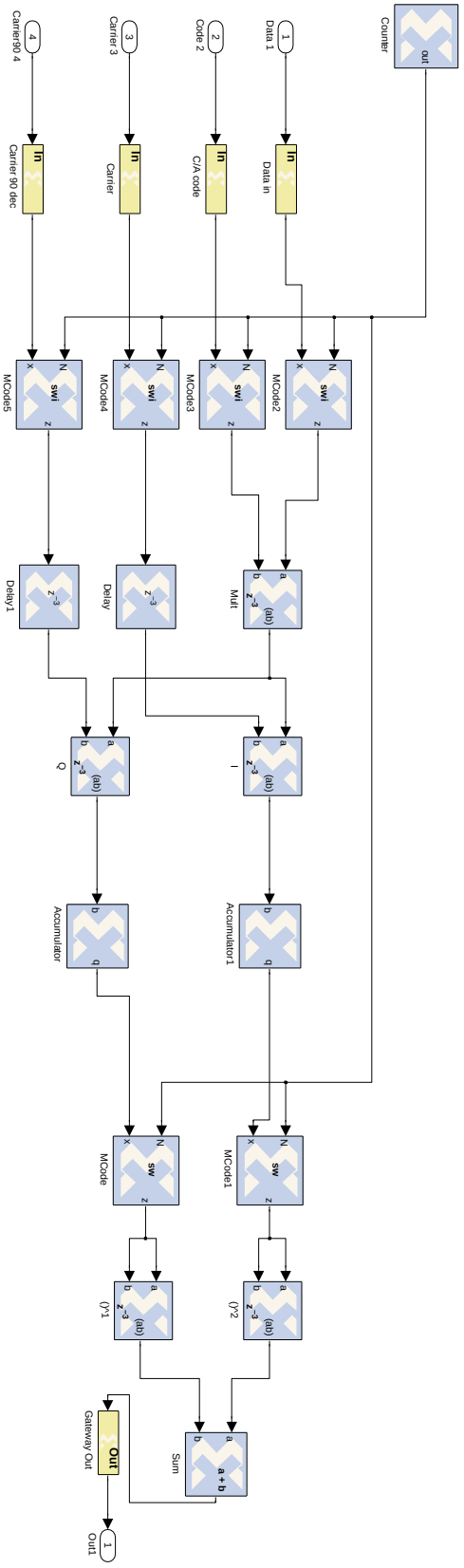


Figure B.4: The figure shows the Mathworks Simulink & Xilinx System Generator model for the parallel method. The figure shows one of 41 subsystems. The figure can be seen in the report on figure 4.6

C

Results from Synthesize final report

C.1 Serial method

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : testofsingleblock1_cw.ngr
Top Level Output File Name         : testofsingleblock1_cw
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 167

Cell Usage :
# BELS                             : 466
#      GND                         : 1
#      LUT2                        : 264
#      LUT6                        : 2
#      MUXCY                       : 98
#      VCC                         : 1
#      XORCY                       : 100
# FlipFlops/Latches                : 132
#      FDE                         : 132
# Shift Registers                  : 32
#      SRL16E                      : 32
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 165
#      IBUF                        : 64
#      OBUF                        : 101
# Others                           : 9
#      adder_subtracter_virtex5_11_0_96f86ec9539b26f9: 1
#      binary_counter_virtex5_11_0_07a214a68614af2b: 1
#      multiplier_virtex5_11_2_1c4d016bc87cc33d: 2
#      multiplier_virtex5_11_2_4a7e1b06d297d994: 1
#      multiplier_virtex5_11_2_7bae220c7b431dc2: 2
#      TIMESPEC                    : 1
#      xlpersistentdff             : 1
=========================================================================

Device utilization summary:
```

```
---------------------------

Selected Device : 5vfx70tff1136-3


Slice Logic Utilization:
 Number of Slice Registers:              132  out of  44800     0%
 Number of Slice LUTs:                   298  out of  44800     0%
    Number used as Logic:                266  out of  44800     0%
    Number used as Memory:                32  out of  13120     0%
       Number used as SRL:                32

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     298
   Number with an unused Flip Flop:      166  out of    298    55%
   Number with an unused LUT:              0  out of    298     0%
   Number of fully used LUT-FF pairs:    132  out of    298    44%
   Number of unique control sets:          1

IO Utilization:
 Number of IOs:                          167
 Number of bonded IOBs:                  166  out of    640    25%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 1  out of     32     3%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------



=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
+------------------------------------+------------------------+-------+
| Clock Signal                       | Clock buffer(FF name)  | Load  |
+------------------------------------+------------------------+-------+
| clk                                | BUFGP                  | 164   |
+------------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
-----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 2.619ns (Maximum Frequency: 381.869MHz)
   Minimum input arrival time before clock: 2.306ns
   Maximum output required time after clock: 0.847ns
   Maximum combinational path delay: 2.379ns

=========================================================================
Timing constraint: TS_clk_7880f197 = PERIOD TIMEGRP "clk_7880f197" 10 nS HIGH 5 nS
  Clock period: 2.619ns (frequency: 381.869MHz)
  Total number of paths / destination ports: 5032 / 132
  Number of failed paths / ports: 0 (0.00%) / 0 (0.00%)
-------------------------------------------------------------------------
Slack:                  7.403ns
```

```
Source:               testofsingleblock1_x0/accumulator1/accum_reg_41_23_0 (FF)
Destination:          testofsingleblock1_x0/accumulator1/accum_reg_41_23_49 (FF)
Data Path Delay:      2.619ns (Levels of Logic = 51)
Source Clock:         clk rising at 0.000ns
Destination Clock:    clk rising at 10.000ns

Data Path: testofsingleblock1_x0/accumulator1/accum_reg_41_23_0 (FF) to testofs...
                                Gate     Net
  Cell:in->out      fanout     Delay    Delay   Logical Name (Net Name)
  ----------------------------------------   ------------
    FDE:C->Q              2    0.396    0.440   testofsingleblock1_x0/accumulator1/...
    LUT2:I0->O            1    0.086    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:S->O            1    0.305    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           1    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    MUXCY:CI->O           0    0.023    0.000   testofsingleblock1_x0/accumulator1/...
    XORCY:CI->O           1    0.300    0.000   testofsingleblock1_x0/accumulator1/...
    FDE:D                     -0.022            testofsingleblock1_x0/accumulator1/...
  ----------------------------------------
  Total                        2.619ns (2.179ns logic, 0.440ns route)
                                       (83.2% logic, 16.8% route)

================================================================================
```

```
Total REAL time to Xst completion: 20.00 secs
Total CPU time to Xst completion: 9.50 secs

-->


Total memory usage is 405028 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :  231 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

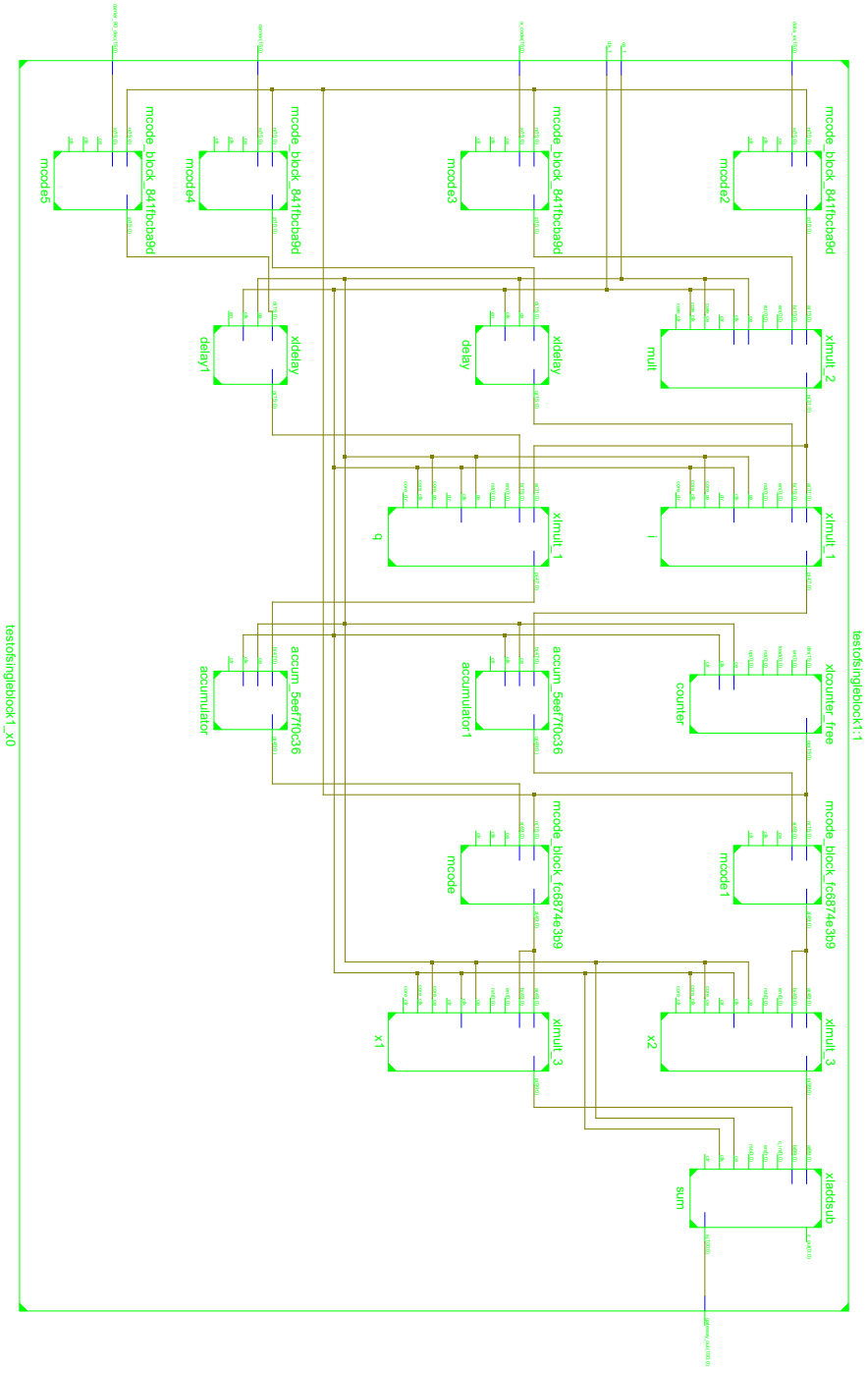


Figure C.1: The results after generation of Xilinx project with Xilinx Systemgenerator in Mathworks Simulink. The simulink model can be seen in block for on figure B.1

C.2 Parallel method

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : newlargeblock1_cw.ngr
Top Level Output File Name         : newlargeblock1_cw
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 7915

Cell Usage :
# BELS                             : 23618
#      GND                         : 1
#      LUT2                        : 13120
#      LUT6                        : 82
#      MUXCY                       : 5166
#      VCC                         : 1
#      XORCY                       : 5248
# FlipFlops/Latches                : 6560
#      FDE                         : 6560
# Shift Registers                  : 1312
#      SRL16E                      : 1312
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 7913
#      IBUF                        : 2624
#      OBUF                        : 5289
# Others                           : 289
#      adder_subtracter_virtex5_11_0_6ad7be66fd0ffb44: 41
#      binary_counter_virtex5_11_0_07a214a68614af2b: 41
#      multiplier_virtex5_11_2_4a7e1b06d297d994: 41
#      multiplier_virtex5_11_2_7bae220c7b431dc2: 82
#      multiplier_virtex5_11_2_d7bc6f69e434178f: 82
#      TIMESPEC                    : 1
#      xlpersistentdff             : 1
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vfx70tff1136-3


Slice Logic Utilization:
 Number of Slice Registers:            6560  out of  44800    14%
 Number of Slice LUTs:                14514  out of  44800    32%
    Number used as Logic:             13202  out of  44800    29%
    Number used as Memory:             1312  out of  13120    10%
       Number used as SRL:             1312

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  14514
   Number with an unused Flip Flop:    7954  out of  14514    54%
   Number with an unused LUT:             0  out of  14514     0%
   Number of fully used LUT-FF pairs:  6560  out of  14514    45%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                       7915
 Number of bonded IOBs:               7914  out of    640  1236% (*)

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               1  out of     32     3%
```

```
WARNING:Xst:1336 -  (*) More than 100% of Device resources are used

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 7872  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 2.937ns (Maximum Frequency: 340.461MHz)
   Minimum input arrival time before clock: 2.463ns
   Maximum output required time after clock: 0.847ns
   Maximum combinational path delay: 2.379ns

=========================================================================
Timing constraint: TS_clk_bf3829b0 = PERIOD TIMEGRP "clk_bf3829b0" 10 nS HIGH 5 nS
  Clock period: 2.937ns (frequency: 340.461MHz)
  Total number of paths / destination ports: 337184 / 6560
  Number of failed paths / ports: 0 (0.00%) / 0 (0.00%)
-------------------------------------------------------------------------
Slack:                  7.085ns
  Source:               newlargeblock1_x0/subsystem2_36a7e26a66/subsystem9_7...
  Destination:          newlargeblock1_x0/subsystem2_36a7e26a66/subsystem9_7...
  Data Path Delay:      2.937ns (Levels of Logic = 65)
  Source Clock:         clk rising at 0.000ns
  Destination Clock:    clk rising at 10.000ns

  Data Path: newlargeblock1_x0/... (FF) to newlargeblock1_x0/... (FF)
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDE:C->Q              2   0.396   0.440  newlargeblock1_x0/subsystem2_...
     LUT2:I0->O            1   0.086   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:S->O            1   0.305   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
     MUXCY:CI->O           1   0.023   0.000  newlargeblock1_x0/subsystem2_...
```

```
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          1    0.023    0.000   newlargeblock1_x0/subsystem2_...
    MUXCY:CI->O          0    0.023    0.000   newlargeblock1_x0/subsystem2_...
    XORCY:CI->O          1    0.300    0.000   newlargeblock1_x0/subsystem2_...
    FDE:D                    -0.022            newlargeblock1_x0/subsystem2_...
    ----------------------------------------
    Total                     2.937ns (2.497ns logic, 0.440ns route)
                                      (85.0% logic, 15.0% route)

=========================================================================


Total REAL time to Xst completion: 132.00 secs
Total CPU time to Xst completion: 102.48 secs

-->


Total memory usage is 654388 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings : 5420 (   0 filtered)
```

```
Number of infos    :    0 (    0 filtered)
```

D
DVD

Structure

- **Code**
  Contains Matlab implementation, System generator/Simulink Implementation, Test data and all files from Lab simulation
- **Design**
  Contains designs of both methods, design by Xilinx ISE
- **Extra Material**
  Contains different material used relevant for the project
- **Results**
  Contains Synthesis reports for both designs
- **Xilinx**
  Contains manuals for Xilinx ISE and System generator
- Project repport 09gr944.pdf
- timeplan.pdf