TSEK06
Final Project Report

Group 5

Editor: Johannes Klasson

Version P1B

Status

Reviewed	Johannes Klasson	2016-05-28
Approved	Martin Nielsen-Lönn	-

PROJECT IDENTITY

VT, 2016, Group 5
Linköpings Tekniska Högskola, ISY

Group members

Name	**Responsibility**	**Phone**	**E-mail**
Johan Isaksson	Project Leader	070-2688785	johis024@student.liu.se
Johannes Klasson	Document Manager	073-8209003	johkl226@student.liu.se
Jonas Tarassu	VLSI Designer	070-5738583	jonta760@student.liu.se
Alexander Yngve	VLSI Designer	076-2749762	aleyn573@student.liu.se

Customer: ISY
Contact at customer: Martin Nielsen-Lönn
Course resposible: Atila Alvandpour
Consultant: Martin Nielsen-Lönn

Contents

Document history

Version	**Date**	**Changes**	**Performed by**
P1A	2016-05-26	First draft	Johan Isaksson
P1B	2016-05-28	Updated Simulation results	Johan Isaksson

1 Introduction

This document describes the 16-bit Kogge-Stone adder project in the course TSEK06. The project took the TOP-DOWN approach starting with a rough description of the behaviour of the system and ended up with a full custom design implemented in 0.35 µm CMOS technology. The main reason for doing this is to be able to simulate all logic to make sure everything works as intended before moving on to the next stage. A full in-depth description of the project can be found in section 2.

2 Project Description

The objective of this project was to implement a 16-bit Kogge-Stone Adder in 0.35 µm technology which from now on will be called KS adder. The KS adder belongs to the family of parallel-prefix adders which reduces the critical path compared to regular ripple-carry adders. The cost for this is paid in area and complex routing which prolongs development time.

Fast adders are very important in all types of integrated circuits since almost all algorithms one can come up with consists of at least one addition.

The official goal was to reach a speed of 200 MHz but the group decided on an unofficial goal to reach a speed of 400 MHz.

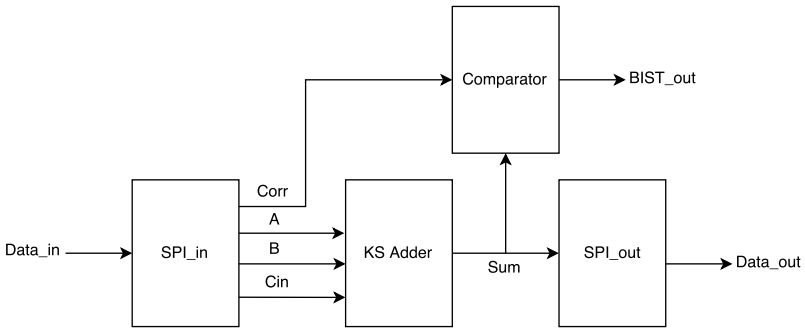

Figure 1 – Block diagram of the system after the first iteration.

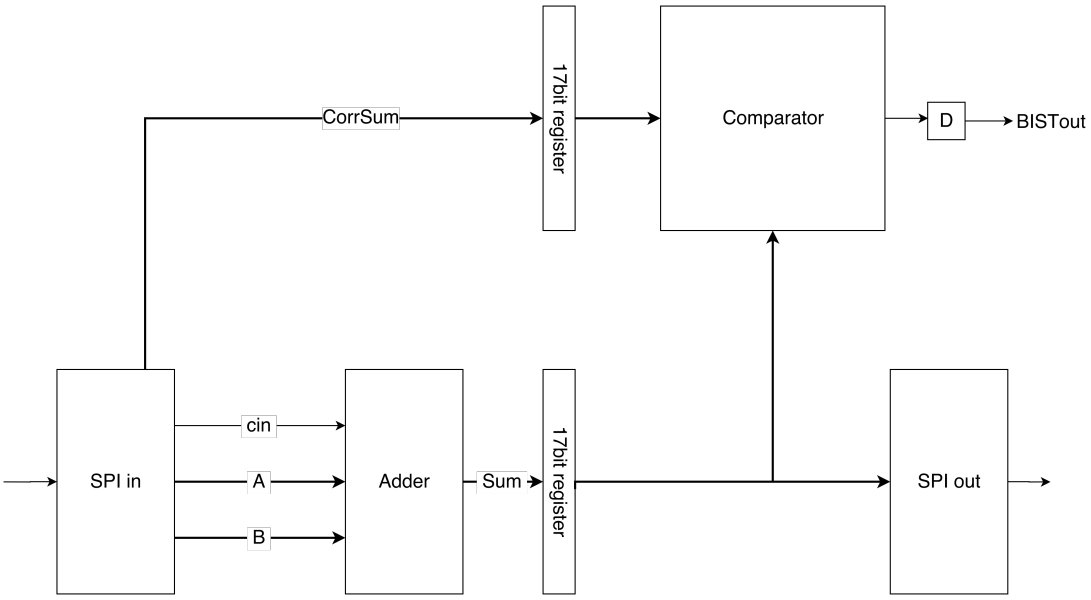

Figure 2 – Block diagram of the system after the second iteration.

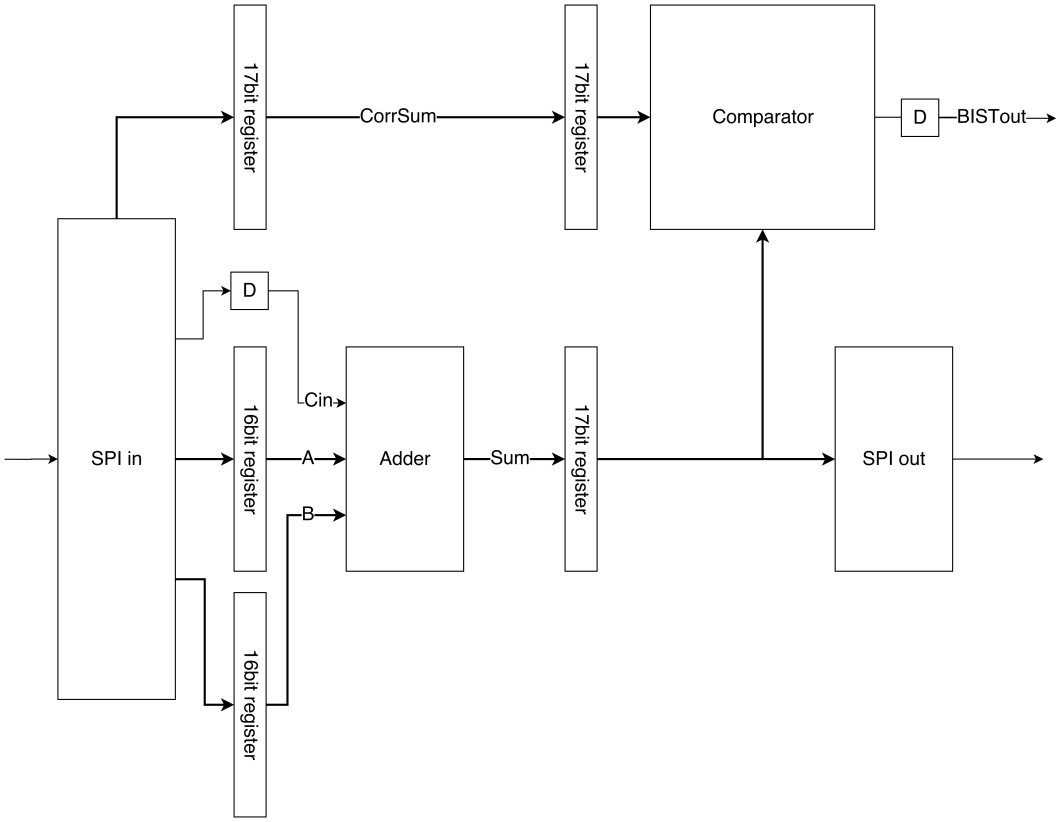

Figure 3 – Final block diagram of the system.

In Fig. 1 the block diagram of the system after the first design phase is depicted. One can note that the signal goes from one block to the next directly. In Fig. 2 some registers have been introduced to make sure BISTout stays the same between clock pulses when new data is

evaluated and to give a more stable signal to the comparator and SPI-out unit. In the final iteration additional register have been inserted between SPI in and the adder to cut the path from the registers in the SPI unit to the adder and to make sure that all bits in the same word arrives at the same time to the adder. The block diagram from the last iteration can be seen in Fig. 3.

2.1 SPI-in

The SPI-in module is supposed to serially receive four times four 16-bit numbers and to distribute these to the correct parts of the system.

2.1.1 SPI-recieve

The first part in the SPI-in module is SPI-receive, where we receive all the bits (see figure 4). It consists of 16 D flip-flops (DFF), that is connected one after another in a serial manner. They are clocked on the SPI clock, and on each positive clock edge, a new input bit is shifted in. After 16 pulses we have 16 bits stored, and a load signal is triggered so that each bit is moved to the correct test-register.

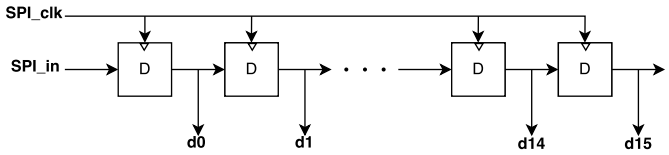

Figure 4 – Block diagram of the SPI-receive block.

2.1.2 Test-registers

The test-registers (see Fig. 5) consists of four flip flops (one for each addition). While SPI_enable is low, we load the registers with data from SPI-receive, and when it is high one of two things can happen, depending on what we want to test. If we want to test the energy consumption, then the new data to the registers are pseudo-random data, that we get from a XOR of the third and fourth bit in the register, meaning we get 15 different sequence combinations. And if we want to test the speed, then the same data is looped through the registers over and over, meaning we can slowly increase the clock rate and see when the system produces the wrong data.

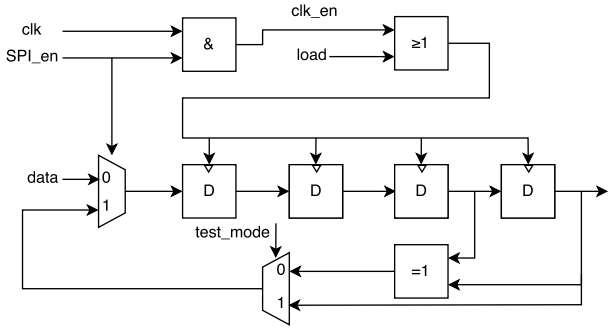

Figure 5 – Block diagram of the test-registers.

2.1.3 SPI-controller

The last part of this module is some control logic (see Fig. 6). The heart of the control block is a 6-bit counter, where the first four bits signalizes if we have read a 16 bit word or not and the last

two bits signalizes which of our four words we are currently reading. By combining these signals like in the figure we are able to produce the load signals that trigger the test-registers.

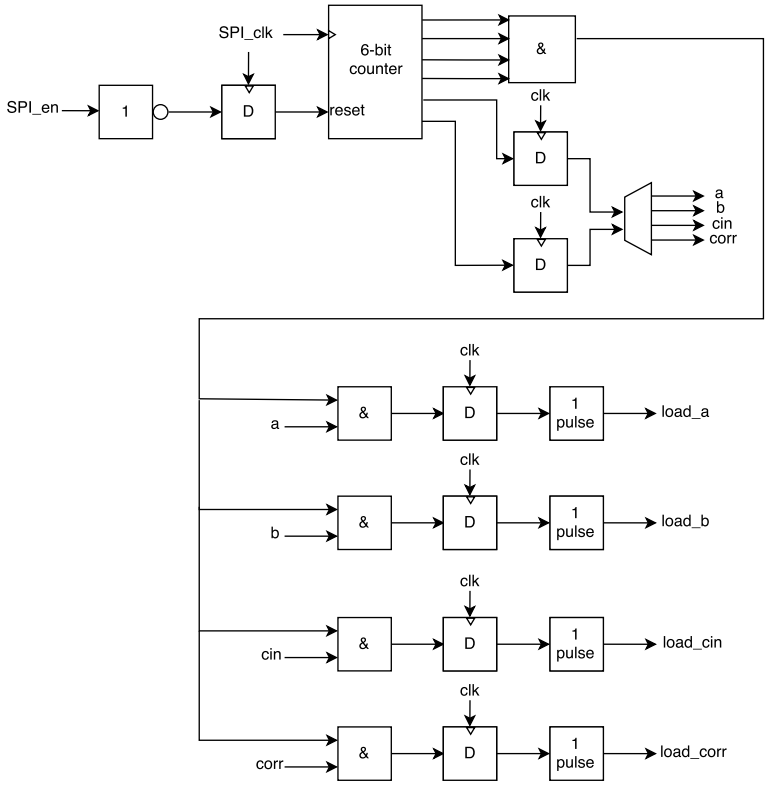

Figure 6 – Block diagram of the SPI-controller.

2.2 16-bit Kogge-Stone Adder

The KS adder consists of several simple blocks connected in a complex way. The *yellow* block has been split into two blocks *yellow_inv_in* and *yellow_inv_out*, which can be seen in Fig. 7. The *yellow_inv_in* block takes inverted input signals and gives non-inverted output. The *yellow_inv_out* block takes non-inverted inputs and gives inverted output. This arrangement saves a lot of gates. The *yellow_carry* block has been split in the same way.

Because of the inverted signals from *yellow_inv_out* some *sum* blocks have been replaced with XNOR gates instead of XOR gates which are the case in a standard KS adder design. A couple of inverters have also been added within the adder to make sure the new blocks gets the correct input.

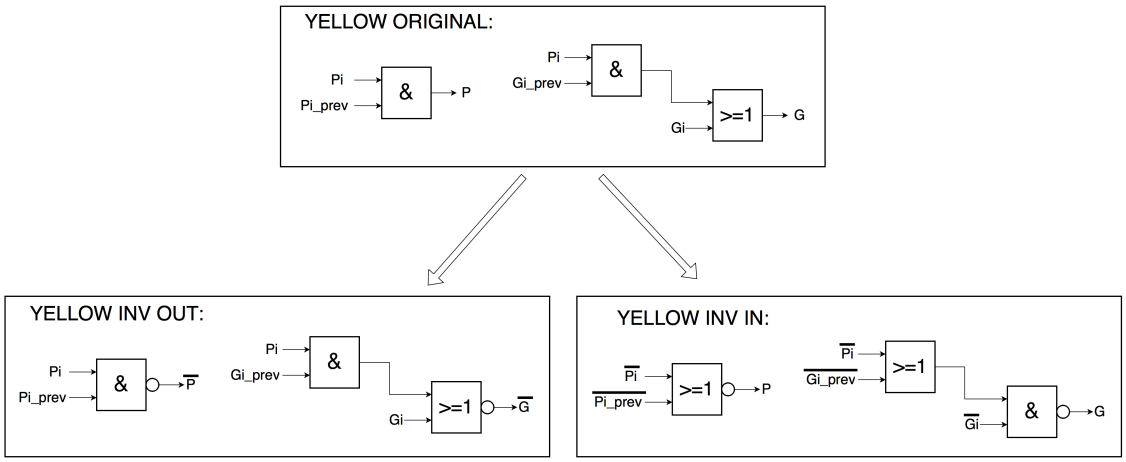

Figure 7 – The new yellow blocks.

To save space, new switch nets were created for the generate calculation of the *yellow* blocks. The new nets can be seen in Fig. 8 and 9. By doing this the transistor count is cut in half.

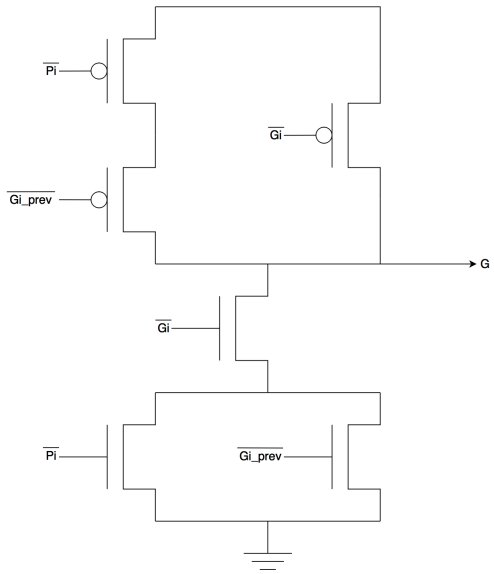

Figure 8 – Generate part of *yellow_inv_in*.

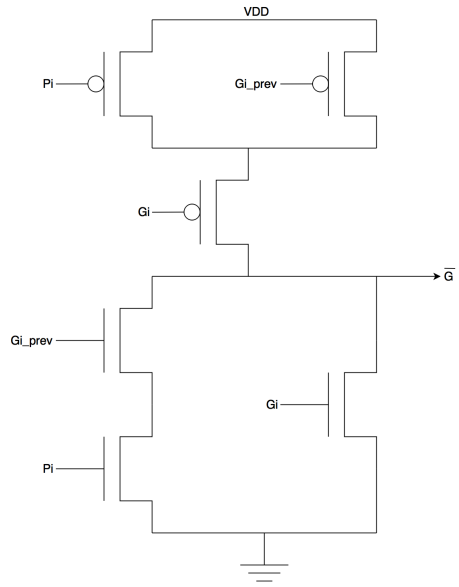

Figure 9 – Generate part of *yellow_inv_out*.

2.3 SPI-out

This chapter will describe the SPI-output module. An overview of the module can be seen in Fig. 12.

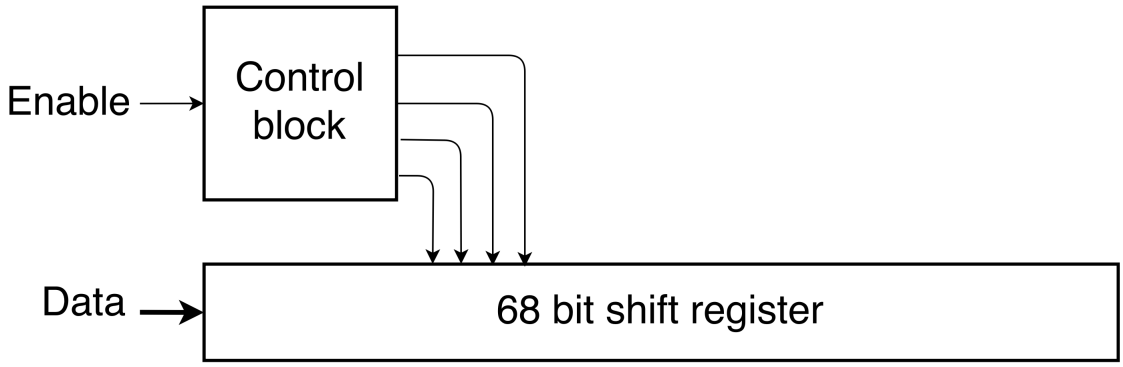

Figure 10 – SPI-out overview

2.3.1 Shift register

The major component is a 68-bit shift register, for the four 17-bit words, where each cell in the register contains one D flip-flop and one multiplexer. The schematic for a cell can be seen in Fig. 11.

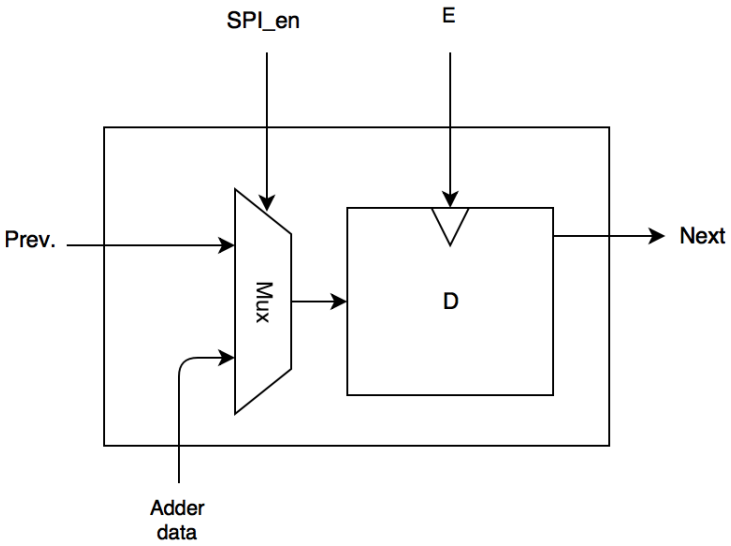

Figure 11 – Shift register cell

2.3.2 Control logic

The control logic is needed to load the long shift register with the correct data. Since the adder is built in such a way that it executes all the four additions as fast as it can after the SPI enable signal goes high, the output from one addition will only be available for one system clock cycle. This means that we need to quickly grab the data and put it in the correct place.

Our implementation uses a pulse generator, a 2-bit counter, a decoder to select what 17-bit part of the shift register to load, and some multiplexers to select if to clock the register on the spi clock or on the enable signals from the decoder. The pulse generator is triggered by the spi enable signal and creates a pulse that is four system clock cycles long. This is to only allow the counter to increment four times, and also to clip the pulse from the last decoder output. As can be seen in Fig. 12 the last output of the decoder is also fed back to the enable of the counter through an inverter. This prevents a possible glitch that is generated on E1 when the counter starts over, which would reload the section of the shift register containing the first sum.

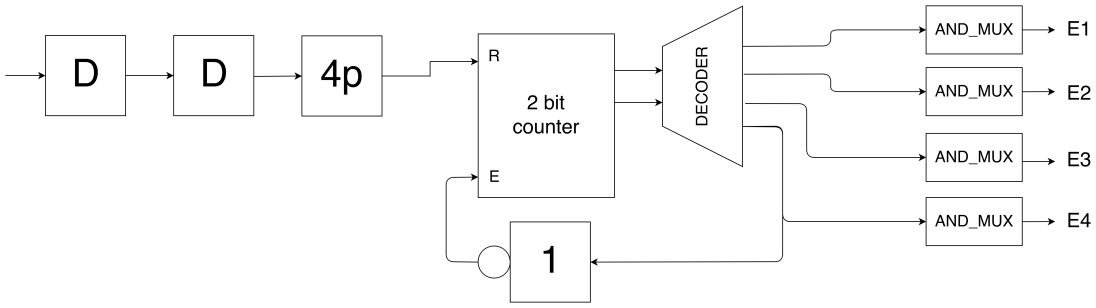

Figure 12 – Control logic for SPI-output

As can be seen to the left in Fig. 12, there are two D flip-flops in the beginning. They are needed to sync the SPI enable signal with the data from the adder as the data is delayed by two system clock cycles.
To the right there are four enable signals: E1, E2, E3, E4. Each one of those signals enables a specific part of the shift register. If SPI enable is low, all of the signals would be equal to the SPI clock so that data can be shifted out. Otherwise, if the SPI enable signal is high and, for example, E1 is high, the 17-bit part of the shift register corresponding to the sum of the first addition would be loaded with the output from the adder.
As each of the enable signals has a high fan out they all need a buffer which can be seen in Fig. 13.

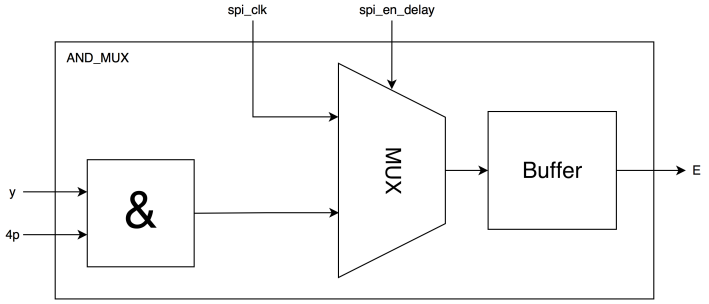

Figure 13 – Functional block containing an AND and a MUX.

3 Simulation Results

After running the simulation for different corners we retrieved the following data regarding system performance. The test data used can be seen in Tab. 3.

Table 1 – Corner results.

Corner	Nominal	Worst speed	Worst power	Unit
Simulated temperature	27	100	0	Celcius
Tested frequency	200	166	333	MHz
Power consumption chip	18.89	16.54	31.3	mW
Power consumption adder	0.93	0.807	1.436	mW
Propagation delay sum15	2.209	4.234	1.264	ns
Propagation delay cout	2.195	3.744	1.332	ns
Possible adder frequency	455	236	750	MHz

The propagation delay was measured between Cin and Cout and between Cin and Sum15. The possible adder frequencies in Tab. 1 are based on longest of these two. One thing to note is that the complete system did not function correctly during worst speed. The adder itself works fine and the correct data is outputted through SPI-out, but the built-in self test is failing. Simulation results, for the nominal corner with a frequency of 200 MHz, of the propagation delay, the SPI-out data and of BISTout can be seen in Appendix A.

4 Chip evaluation

This chapter will describe how to verify the functionality of the chip using a FPGA board and an oscilloscope.

4.1 Pad list

Tab. 2 shows the pin assignments for the chip. The pad frame with corresponding names and pin type is shown in Fig. 16. Fig. 17 shows how to connect the chip to the FPGA board and oscilloscope.

Table 2 – Pin assignments

Name	**Direction**	**Type**	**Description**
Vdd1	INOUT	Analog	Will provide most of the system with power and will be a steady 3.3 V.
Vdd2	INOUT	Analog	Will provide the adder with power and it might vary from 3.3 V downto threshold-voltage.
GND	INOUT	Analog	Ground.
Clk	IN	Analog	This is the clock for the adder, some registers and control logic. Should have a frequency of at least 200 MHz at 3.3 V. Will be lower as we decrease the voltage of Vdd2.
SPI_clk	IN	Digital	This clock is used by the input and output unit and should be at least four times slower than the system clock. Should also be low if SPI_en is inactive.
SPI_en	IN	Digital	Active low. Should go high on the first negative flank of SPI_clk after the last value is read.
SPI_in	IN	Digital	Updates it's value as soon as SPI_en goes low, and should have it's value ready on the first positive flank of SPI_clk, since this is when we read the value. The value of SPI_in should then be updated on every negative flank of SPI_clk.
SPI_out	OUT	Digital	The data is available for read on the first falling edge of SPI_clk after SPI_en has gone active.
BISTout	OUT	Digital	If the IN-data is correct, BISTout should be constant high after the first addition is done until the the PRBS-bit is set.
Cin	IN	Digital	Used to measure propagation delay through the adder.
Cout	OUT	Analog	Used to measure propagation delay through the adder.
Sum15	OUT	Analog	Used to measure propagation delay through the adder.

4.2 SPI Interface

This section describes the protocol used to communicate with the chip.
As an attempt to save time on implementation, the group decided to deviate from the standard SPI protocol and create an specialized protocol for the chip. The total amount of data that should be sent, during each active period of the SPI enable, is 4x4 16-bit words to the chip and four 17-bit words from the chip. This is illustrated in Fig. 14 where the 4x4 16-bit words are grouped into four 64-bit words.

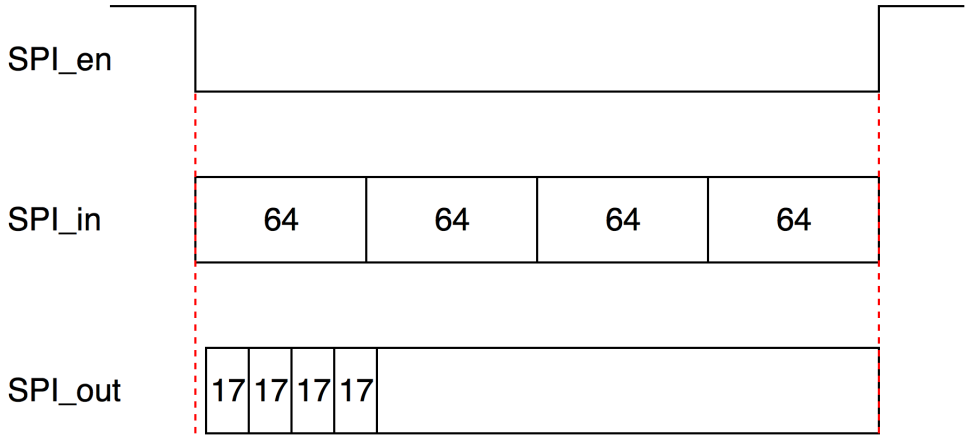

Figure 14 – SPI data diagram

The chip receive and send data with the most significant bit first. The order of the bits is illustrated Fig. 15. Data is both written and read by the chip on the rising edge of the the SPI clock. This means that the circuitry communicating with the chip has to read and write on the falling edge of the clock. The first input data should be available on the first rising edge after SPI enable goes low, and the first bit of the output will be available on the first falling edge after SPI enable goes low. This can also be seen in Fig. 15.

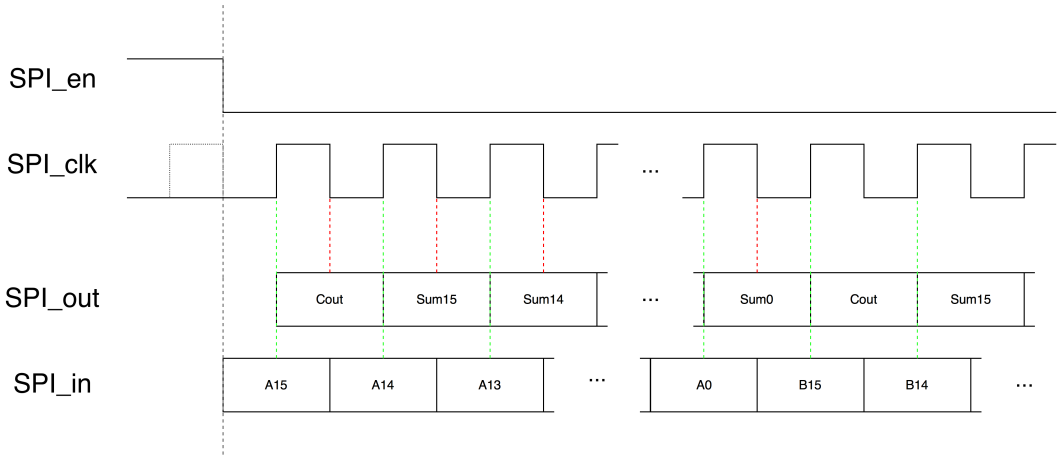

Figure 15 – SPI timing diagram

Worth mentioning is that the SPI clock should be kept low while SPI enable is high. This can also be seen in Fig. 15 but might not be as clear.

Tab. 3 shows the data that should be fed into the chip to test it's functionality.

Table 3 – Test data

A	**B**	**Config bits**	**Check sum**
1111111111111111	1111111111111111	0000000000000011	1111111111111111
0100010000110001	0001000100100011	0000000000000000	0101010101010100
1111111111111111	0000000000000000	0000000000000111	0000000000000000
0000000000000000	0000000000000000	0000000000000100	0000000000000000

4.3 Evaluation

To test the chip everything should be connected as shown in Fig. 17. The FPGA shown should be programmed to send out the data shown in Tab. 3 with the MSB first to be able to compare the results obtained during simulations before tape-out. The SPI clock should be at least four times slower than the system clock for the SPI-unit to function properly. While the data is being transferred into the chip clk_enable should be held low, which is for 256 SPI clock pulses. After this clk_enable must go high within 15 clock pulses to avoid overwriting previously transferred data.

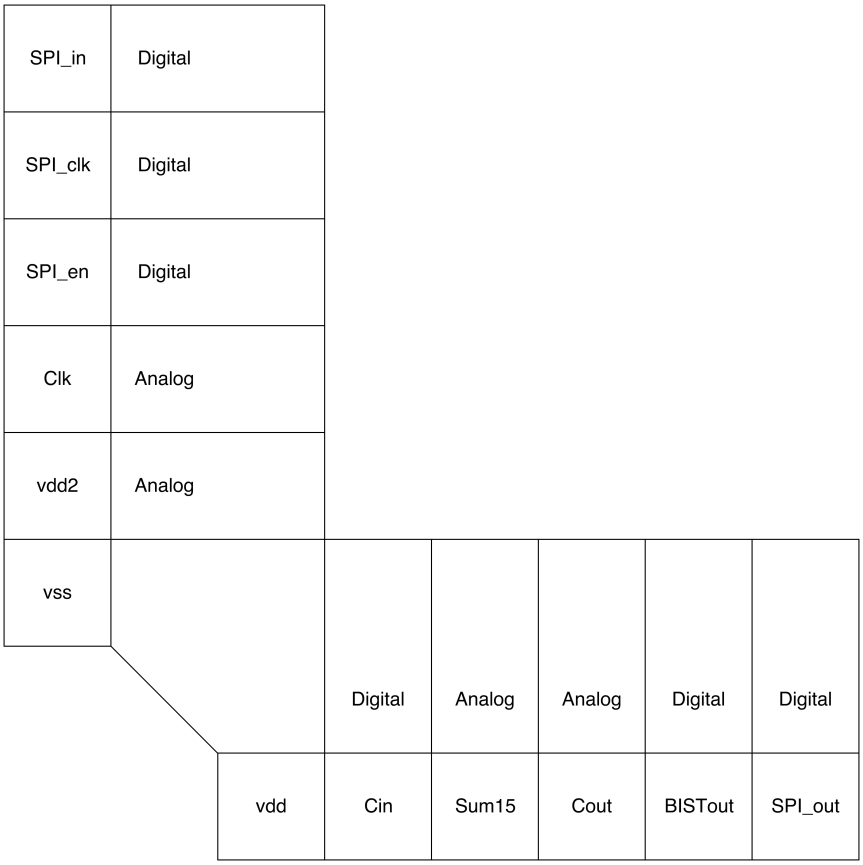

Figure 16 – Overview of the pad frame.

The chip will be tested in two different modes namely, PRBS- and TEST-mode. The PRBS-mode is used to measure the power consumption of the adder while being fed with pseudo random data at full speed. TEST-mode on the other hand is used to measure the propagation delay through the adder. The propagation delay can be determined by measuring the time between transition

on Cin to transition on Cout. For this to happen the input data must be chosen such that the adder overflows which causes Cout to go high. This happens for data number for in Tab. 3. While in TEST-mode it's possible to test how fast the system can be clocked by looking at BISTout while ramping up the frequency. As long as BISTout is high the system functions as intended but as soon as BISTout goes low something failed to keep up with the clock.

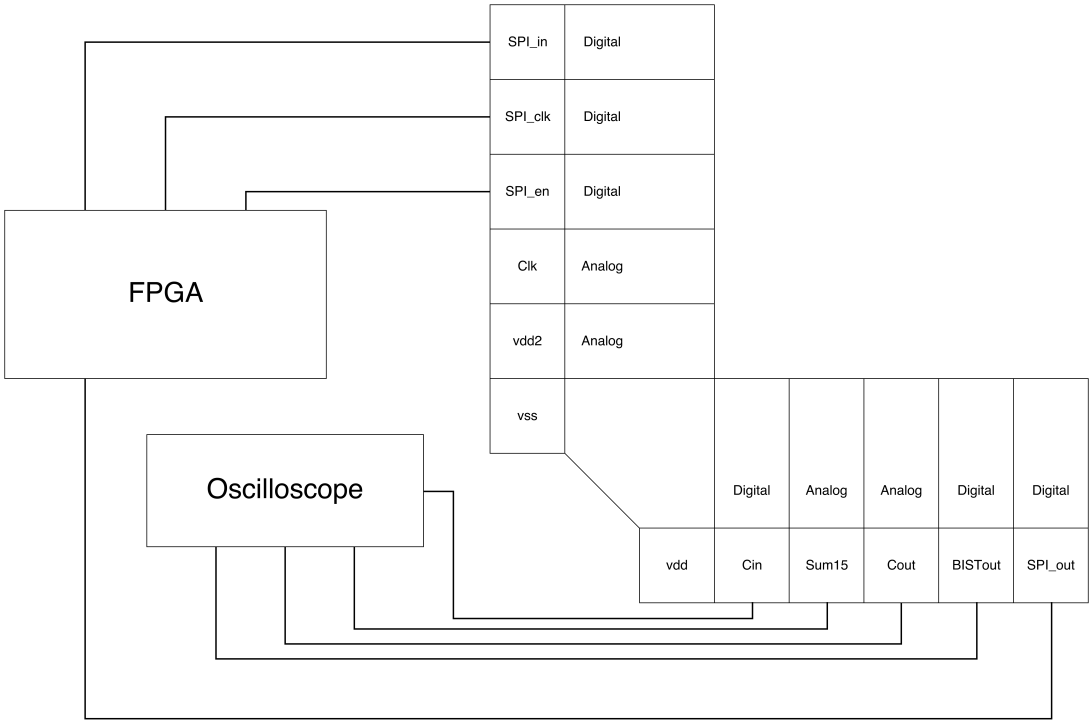

Figure 17 – Pad frame showing how to connect things.

5 Risks

Even though the project itself is done, there is still some risks that have to be taken into consideration before the evaluation of the chip.

1. There might be some small manufacturing errors that can cause the chip to malfunction.
2. The evaluation plan might not be extensive enough, so that much time must be wasted to remember exactly how the chip works and should be tested.
3. Simulations cannot 100% reflect the reality, so some small deviations might cause to chip to behave in an unexpected manner.
4. The simulations might not be extensive enough, and some untested input data might produce the wrong result.

6 Project Evaluation

During the course of the project we gained a lot of experience related to tools, design and also cooperation.

6.1 Cooperation

The group has functioned very well during the whole project except for two weeks during the beginning of the layout phase. There were some miscommunication since half of the group were on holiday the first week while the other half were on holiday during the second week of this period. This later led to some integration issues.

6.2 Tools

In retrospect, less time should have been spent on VerilogA simulations and more time on schematic simulations. The VerilogA models were very inaccurate and we had no idea what values to choose for propagation delays and rise times et.c. The schematic simulations were much more accurate and at the same time quite fast. Since the simulations when doing layout were very slow, a lot of time was spent on waiting for results. Doing more design work at the schematic level would have led to more effective work and possibly a better overall design.
Another tool related issue encountered was the problem with configurations. Using configurations in Cadence seemed like a very useful feature which could help us keep a clean structure of the project by having several schematics and layouts of different versions of the same cell in one place, instead of creating several very similar cells. It turned out however that it was very hard to maintain and the configuration editor in Cadence randomly crashed all the time when creating somewhat advanced configurations. We should have stuck to having only one schematic and one layout view in each cell, even though this led to many cells.
Quite early in the project we researched the possibility of having automatic tests of all cells in the project using a tool called Ocean. Unfortunately we could not get it to work, but it would have been very useful and could have helped us pinpoint errors with ease.

6.3 Design

The design work went smoothly until the integration phase. We expected some hardships during integration, but maybe not to the extent we experienced them. Things seemed to stop working for no apparent reason. Cells that worked yesterday, didn't function today and so on. Integration is hard, this is a lesson all of us will remember.
Looking back, there are several things we could have done much better. Clock distribution, interconnects and floor planning should have been considered much earlier in the project, during the same time as when layouting the basic gates. Floor planning and interconnects could have been done much better if we would have communicated better during the early layout phase, but due to the reasons mentioned in 6.1 this didn't happen. This led to a situation were individual parts functioned well but weren't optimally adapted to eachother. For example, some components of the chip had different widths and therefore the power rails and data interconnects couldn't be aligned easily. If the floor planning had been done earlier in the layout phase, we could also have designed the basic blocks better. When we designed the basic blocks we did it almost exclusivly using metal 1 and 2 layers, in order to leave room for power supply and clock signals on metal 3 and 4 layers. If we would have considered the power supply and clock signals at the same time as when we designed the basic blocks, we could probably have ended up with a tighter design.
Late in the project we discovered some timing issues probably related to the SPI control logic, but we didn't have time to make a new easier and faster control logic design since the layout was already done and the simulations took considerable time. If we had done more testing in the schematic stage, with more realistic loads and possible doing worst case corner testing in schematic simulations, this problem would have been discovered earlier and the control logic could have been redesigned. The problem was eventually solved without a new design but not in an optimal way.

A Simulation results

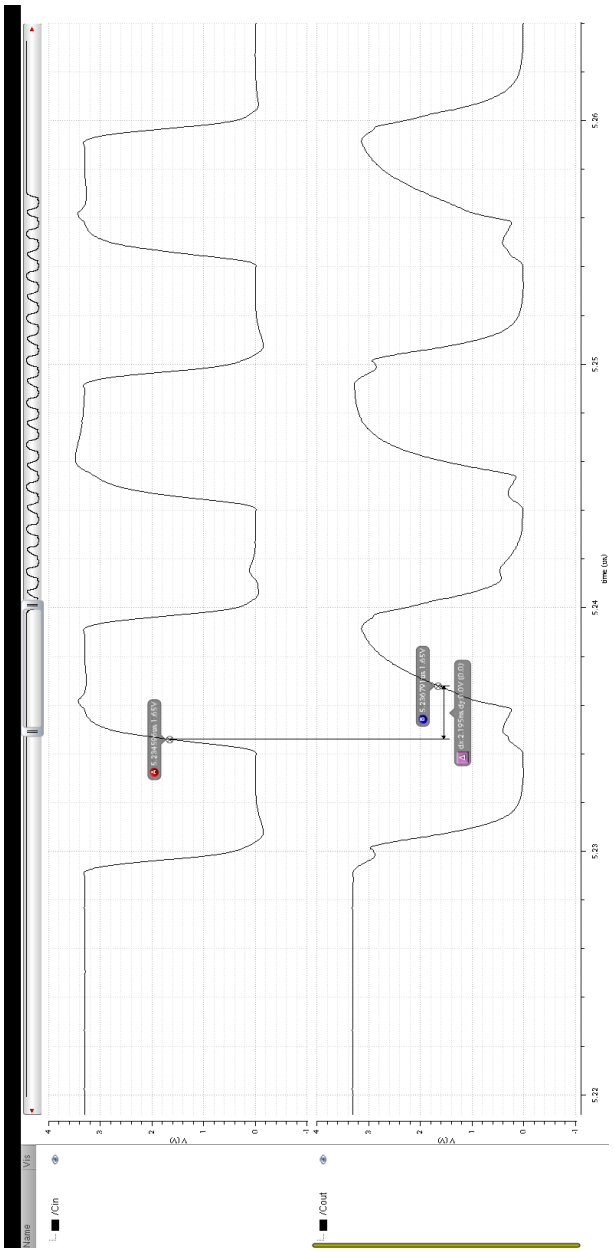

Figure 18 – Test data of propagation delay using the nominal corner and a frequency of 200 MHz

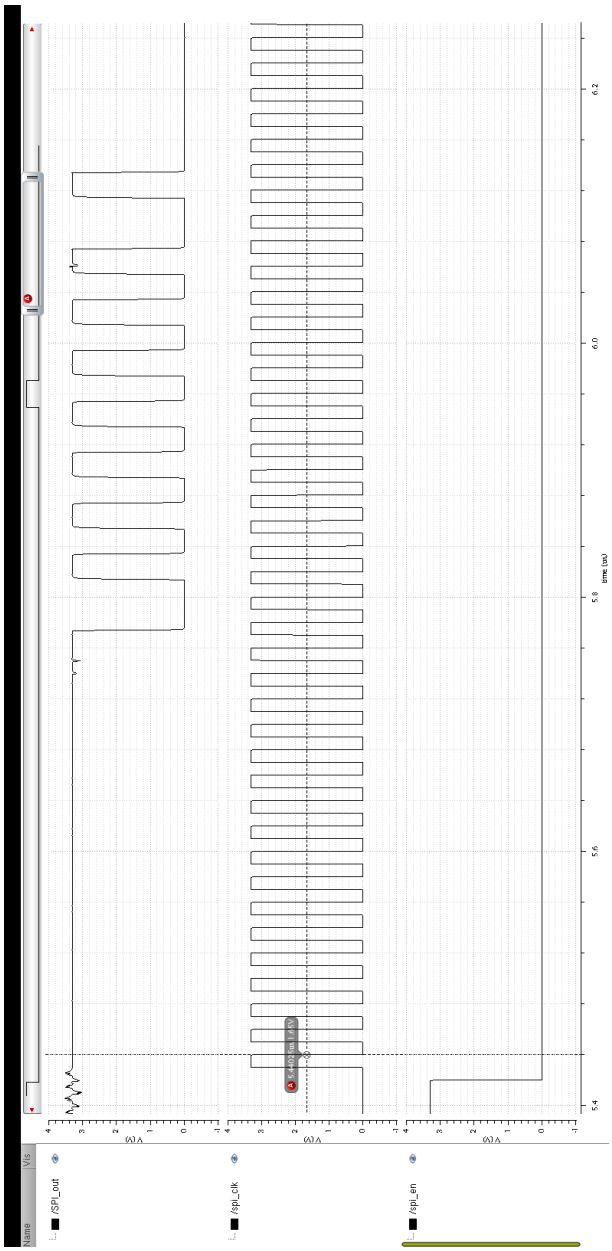

Figure 19 – Test data of SPI-out using the nominal corner and a frequency of 200 MHz

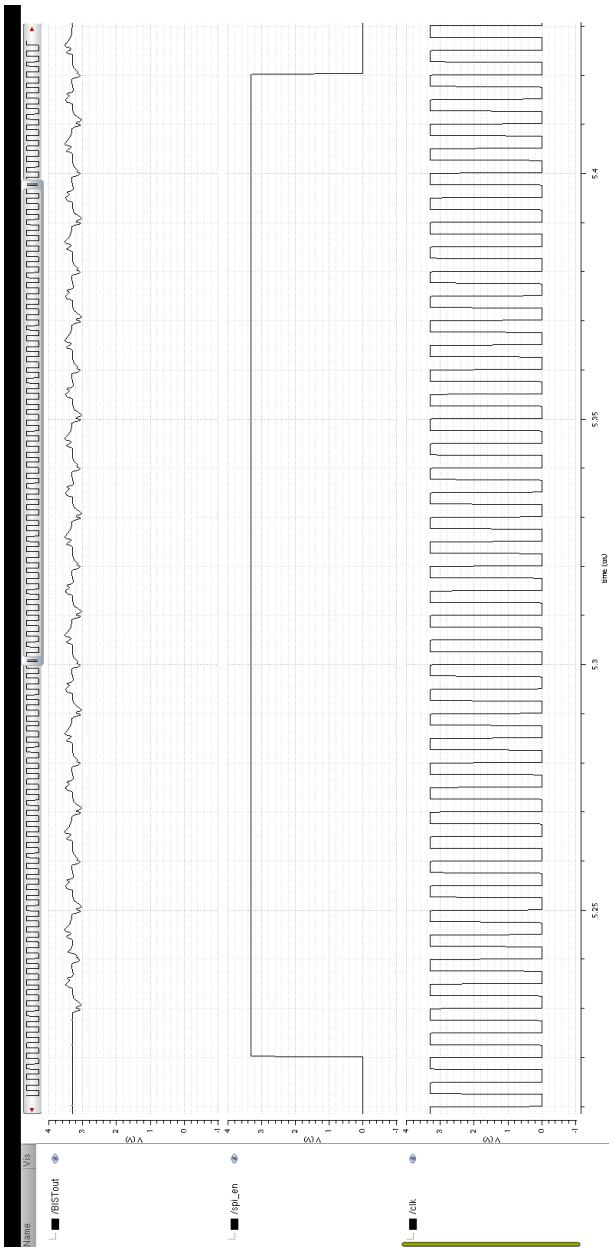

Figure 20 – Test data of BISTout using the nominal corner and a frequency of 200 MHz