Project Report

Editor: Group 3

Version 1.0

Status

Reviewed		
Approved		

PROJECT IDENTITY

2018/VT, Group 3
Linköping University, Dept of Electrical Engineering (ISY)

Group members

Name	**Responsibility**	**Phone**	**Email**
Joakim Bertils	Project Leader (PL)	070-5203702	joabe408@student.liu.se
Lukas Franzon	Project Member	070-3905989	lukfr994@student.liu.se
Martin Hinnerson	Project Member	072-2104372	marhi386@student.liu.se
Johannes Grundell	Project Member	073-2515123	johgr505@student.liu.se
André Steen	Project Member	076-3099921	andst637@student.liu.se

Email list for the whole group:
Web site: http://www.isy.liu.se/edu/kurs/TSEK06/

Customer: ISY, Linköpings universitet, 581 83 Linköping
Customer contact: Martin Nielsen-Lönn, 013-288946, martin.nielsen.lonn@liu.se
Course leader: Atila Alvandpour, 013-285818, atila@isy.liu.se
Tutor: Martin Nielsen-Lönn, 013-288946, martin.nielsen.lonn@liu.se

Document history

Version	Date	Changes	Sign	Reviewed
0.1	2018-02-09	First version		
0.2	2018-03-08	Transistor design		
0.3	2018-05-18	Draft of final report		
1.0	2018-05-24	Final report		

Contents

List of Figures

List of Tables

Acronyms

BIST Built in self-test.

CMOS Complementary metal-oxide-semiconductor.

DC Direct current.

DFF D-type flip-flop.

MSB Most significant bit.

PRBS Pseudorandom binary sequence.

SPI Serial Peripheral Interface.

TFF T flip-flop.

VLSI Very-large-scale integration.

1 Introduction

The aim with this project is to design a fully-custom, high-performance, low power VLSI circuit in sub-micron CMOS technology. This report in specific contains the results of the high-level modeling design, transistor design and simulation results from both of these phases.

1.1 Purpose and goal

The group members should learn the different steps of the IC design flow, including system analysis, simulation, layout implementation and verification.

1.2 Delimitations

The restrictions in this project are the products/components available at ISY and the time budget for the group members.

1.3 Conventions

In this document all signals are typeset in *italics*. Data bits, vectors of data bits and names of subsystems are typeset in san serif.

2 High level design description

The adder circuit block diagram given in [2, Fig. 1] has been changed. A new block diagram of the system can be seen in fig. 1. All the blocks in fig. 1 is described in their corresponding subsections. A few noteworthy changes of the system is the SPI_in being integrated in the register array, thus saving space by reducing the total number of registers. The 16x4-bit shift register for *CorrSum* is also included in the register array so that all the registers can be loaded in series. All changes made will be explained more in detail in their corresponding subsections.

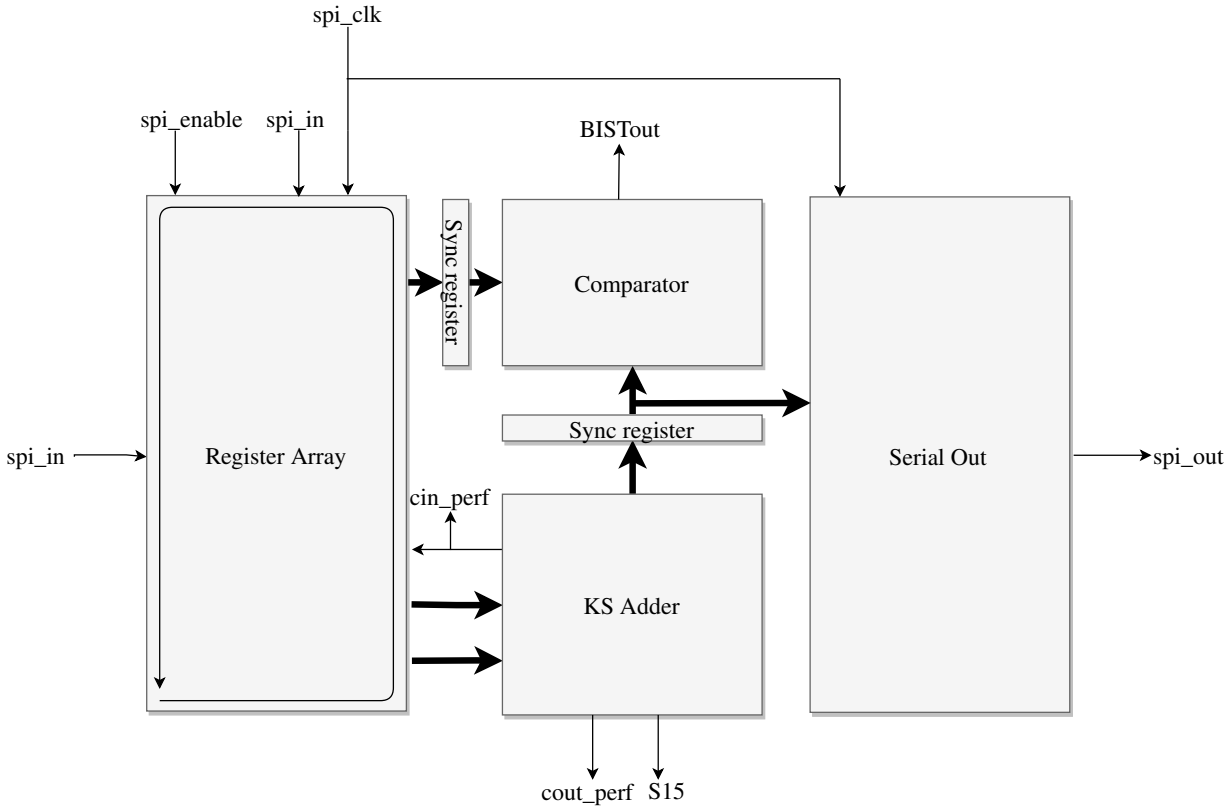

Figure 1: Overview of the system.

2.1 Definition of chip signals

Tab. 1 describe the signals going in and out from the system.

Table 1: Definition of chip signals.

Signal Name	Type	Size	Frequency	Description
spi_clk	IN	1bit	kHz	Input for serial clk
spi_in	IN	1bit	kHz	Input for serial data
spi_enable	IN	1bit	kHz	Signal to enable serial loading
clk	IN	1bit	200 MHz	Global clock signal
vdd_adder	IN	1 wire	DC	Power supply for the adder
vdd_aux	IN	1 wire	DC	Power supply for all components except the adder.
vss	IN	1 wire	DC	Global ground connection
spi_out	OUT	1bit	kHz	Output for serial data
BISTout	OUT	1bit	200 MHz	Output pin from the comparator
cin_perf	OUT	1bit	400 MHz	Carry in signal going to the adder. Used to measure full speed performance of the adder.
cout_perf	OUT	1bit	400 MHz	Carry out signal going from the adder. Used to measure full speed performance of the adder.
S<15>	OUT	1bit	400 MHz	MSB of the calculated sum from the adder. Used to measure full speed performance of the adder.

2.2 Shift-registers/PRBS

The shift registers are a big part of the system. The shift register is constructed to consist of one big register array with all the registers including the *A* vectors, *B* vectors, *Cin* bits, *CorrSum* vectors, and control register bits, see fig. 2. Register *1*, *2* and *CorrSum* contain 16x4 bits. Register *Cin* contains 4 bits and the control register 4 bits. Thus, the total register size is 200 bits.

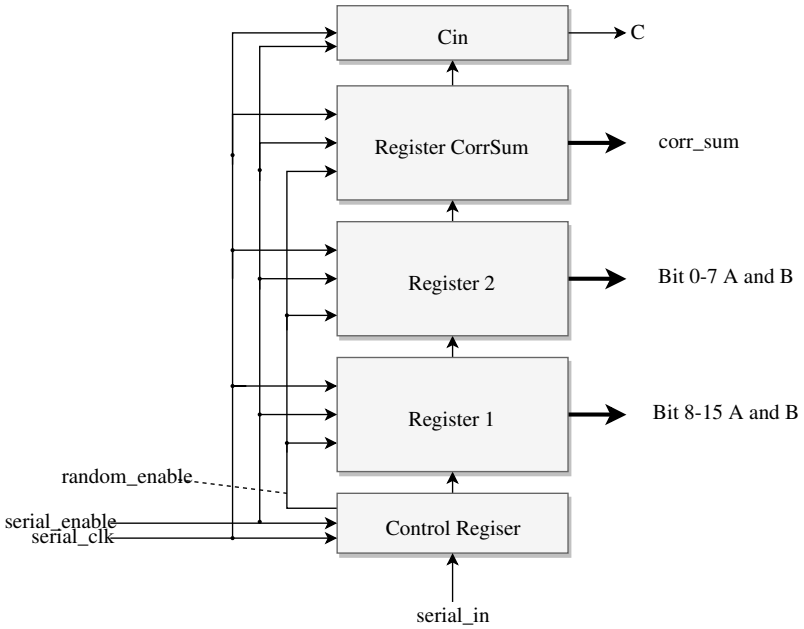

Figure 2: Overview of the shift register blocks.

The control register implemented has 4 bits of which two is used at the moment (*random_enable* and *LS_bypass*). Having extra bits here might be useful in the future for additional control signals. A description of the different bits in the control register and their use can be found in tab. 2.

Table 2: A description of the bits in the control register.

Bit number	Use	Active
1	Level shift bypass	high
2	Random enable	high
3	Unused	
4	Unused	

The serial interface to load the registers is built-in to the registers, meaning that all the registers are loaded in serial with the control register being the last bits in the array and the *CorrSum* bits being the first. This means that loading the registers has to be done in a specific sequence. This is explained in section *2.2.1*.

When loaded, the registers can function in two different modes depending on the control signal *random_enable*. When *random_enable* is low the registers shift the loaded vectors in parallel to the right. The vectors that are shifted out to the adder are also shifted in on the back again. This means the registers can continue shifting the vectors an infinite amount of time without losing them. When *random_enable* is high the registers work in PRBS mode. The registers are then shifted and modified in a pseudo-random manner. This is explained in section 2.2.2.

2.2.1 Loading sequence

Because the register array is loaded in a serial manner the registers at the front of the array will be loaded first and then shifted all the way through. Not until the entire register array is

loaded the bits will be in their correct place. The sequence of loading is described in tab. 3. A C-program is used to create a text file containing the test vectors so that the loading part is automated in a convenient manner.

Table 3: Loading order of the shift registers. Each row corresponds to an entire vector.

Order	**Length (bits)**
Cin 1	1
Cin 2	1
Cin 3	1
Cin 4	1
LSB for CorrSum 1, 2, 3 & 4	4
...	...
MSB for CorrSum 1, 2, 3 & 4	4
LSB for A1, A2, A3 & A4	4
LSB for B1, B2, B3 & B4	4
...	...
MSB for A1, A2, A3 & A4	4
MSB for B1, B2, B3 & B4	4
Control Bits	4

2.2.2 PRBS

When the registers are in the Pseudorandom binary sequence (PRBS) mode the A, B and Cin registers will use a PRBS in order to randomize the contents of the registers. Each register uses the bits inside itself for randomization which means that the A and B register use 64 bits and the Cin register uses 4 bits. The PRBS is achieved by serially shifting the bits inside the register each clock cycle and shifting in the XNOR of bits 64, 63, 61 and 60 for the A and B register and the XNOR of bits 4 and 3 for the Cin register into the firs bit. This way the contents of the registers are pseudorandom but the bits repeats them self in cycles due to it being a sequence. The time it takes for the cycle to repeat itself is however $2^{(number\ of\ bits)} - 1$ clock cycles. This means that the A and B registers repeats in $2^{64} - 1$ clock cycles and the Cin register repeats in $2^4 - 1$ clock cycles. The fact that the Cin register repeats in a shorter time than the A and B registers is not a problem.

2.2.3 Definition of Shift-register signals

Tab. 4 describe the signals going in and out from the Shift-Registers.

Table 4: Definition of shift-register signals.

Signal Name	Type	Size	Frequency	Fan-Out	Description
spi_in	IN	1bit	kHz	1	Input for serial data
spi_clk	IN	1bit	kHz	1	Input for serial clk
spi_enable	IN	1bit	kHz	2	Signal to enable serial loading
random_enable	IN	1bit	kHz	1	Control signal used to activate the PRBS functionality.
clk	IN	1bit	200 MHz	1	Global clock signal
vdd	IN	1 wire	DC	All	Power supply for the registers
vss	IN	1 wire	DC	All	Global ground connection
A	OUT	16bit	200 MHz	3	One of the vectors going to the adder.
B	OUT	16bit	200 MHz	3	One of the vectors going to the adder.
Cin	OUT	1bit	200 MHz	3	Carry in bit to the adder.
CorrSum	OUT	16bit	200 MHz	1	Vector with the correct sum going to the comparator.

2.3 Kogge-Stone Adder

The Kogge-Stone adder is a variant of an adder designed to reduce delay through the circuit while sacrificing chip area. It calculates group carry bits in parallel to speed up the process. The adder is designed to add two binary encoded integer numbers as well as a carry bit, which represents an additional added value of one. The result is encoded as a binary number as well. When two N-bit wide numbers are added, the result is in general $N+1$ bits wide. The additional output bit is represented by an additional output separate from the sum bits called carry out. The calculation of the sum bits and the carry out bit can be done in many different ways, but to reduce the delay in the critical path, a parallel approach to the problem is needed. Since a Kogge-Stone adder is a parallel-prefix adder, it is well suited to the problem. It reduces the propagation delay to $O(log_2(N))$ instead of $O(N)$ for a naive implementation such as the ripple carry adder.

The need for a fast adder is very high as the adder is a main part of almost every CPU architecture as well as many other processing circuits. The delay does in many cases directly translate to the speed of the circuit, which is one of the main design concerns in circuits.

2.3.1 Description

This Kogge-Stone adder builds on the idea presented by Kogge and Stone[1] that a recurrence equation can be broken down into two equally complex sub-terms by recursive doubling. The idea is that these sub-terms can be computed in parallel. When the recursive doubling is done in multiple stages, the algorithm can be extended to 2^N sub-terms done in parallel and $log_2(N)$ stages, where *N* is the number of bits in the input of the adder.

The processing is done in multiple steps, each divided into their own group of blocks. The first step pre-processes the signal by calculating a propagate and generate bit for each block. The second step is the carry look-ahead network, which generate the final generate and propagate signals. The last step post-processes the signals and generates the final sum bits. The steps of the adder is described in detail below. Each bit in the input of the adder is placed in a column, where column *N* represents the input with a value of 2^N. *N* ranges from 0 to 15, since the adder

is designed to add 16-bit numbers. The stages in the adder is alternating normal and inverted signals to reduce the delay through the block.

Pre-processing The first step of the adder is to pre-process the input signals and create a generate and propagate signal for each input bit. The generate bits represent whether a carry should be created at this bit, which happens for example when both inputs *A* and *B* are one. The propagate bit represents whether the carry out bit should be set if there is a carry in bit. This happens in the case when one of the inputs is one and the carry bit is set. These bits are then sent into the carry look-ahead network, which will be described in detail below. To reduce the delay of the block, the output is inverted. The logical function of the pre-process block is described by

$$\overline{G} = \overline{A \cdot B}, \overline{P} = \overline{A \oplus B} \tag{1}$$

where $\overline{G}$ and $\overline{P}$ is the inverted generate signal and the inverted propagate signal respectively, *A* and *B* are the inputs for the block.

Carry look-ahead network The carry look-ahead network is the part if the circuit which calculates the final generate and propagate signals by successively grouping the generate and propagate signals. It consists of $log_2(N)$ stages where N is the number of bits in the input of the adder. This is done by combining carry and propagate signals from the current bit and a previous bit in the stage above. It works because of the fact that you can look at two input bit and calculate if that group of bits will generate a propagate and generate signal. The calculations can be done for all powers of two bits, and are done in parallel, which gives the adder its logarithmic performance. To reduce the delay, the first stage works on inverted inputs and outputs normal signals and the second stage works on normal signals and outputs inverted signals. This pattern is repeated for every stage in the carry look-ahead network. The carry in bit is inverted appropriately in each stage.

There are four types of blocks in the carry look-ahead network. The first one is the block that calculates propagate and generate for two input bits and outputs inverted signals, and its function is given by the logic equations

$$\overline{G'} = \overline{G + G_p \cdot P}, \overline{P'} = \overline{P \cdot P_p}, \tag{2}$$

where $\overline{G'}$ and $\overline{P'}$ are the inverted generate signal and inverted propagate signal respectively created by the block, *G* and *P* are the generate and propagate for the bit in the previous stage and G_p and P_p are the generate and propagate of a previous bit in the previous stage.

There also exists a variant of this block which do not have a previous propagate in the previous stage. The logical equation for that block is

$$\overline{G'} = \overline{G + P \cdot G_p}, \tag{3}$$

with the same definitions as the block above.

The two remaining blocks are identical to the first two blocks with the exception that the input bits are inverted instead of the output bits. The logical equation for the block with previous propagate signal is given by

$$G' = \overline{G} + \overline{G_p} \cdot \overline{P}, P' = \overline{P} \cdot \overline{P_p}, \tag{4}$$

with the definitions of the input and output signals identical to the block with inverted output signal.

The last block is the block with inverted input signals and no previous propagate signal and its logical function is given by

$$G' = \overline{G} + \overline{P} \cdot \overline{G_p}, \tag{5}$$

with above definitions.

Post-processing The last step processes the result by combining the propagate bit and the previous carry to compute the final sum bit. This is done in two different blocks, depending on which layer the input bits come from. The first block works on an inverted generate signal and its logical equation is given by

$$S = \overline{P} \oplus \overline{G_p}, \tag{6}$$

where *S* is the sum bit of the column, $\overline{P}$ is the inverted propagate from the pre-process step in the current column and $\overline{G_p}$ is the inverted generate bit of the previous column.

The second block works on the normal generate signal and its logical function is given by the logical equation

$$S = \overline{\overline{P} \oplus G_p}, \tag{7}$$

where G_p is the generate bit in the previous column and the other signal is defined as above.

The carry out is also generated in this step, and is the group generate of the carry in and the generate and propagate of the total adder group.

2.3.2 Definition of signals - Adder

Tab. 5 describe the signals going in and out from the Adder.

Table 5: Definition of adder signals.

Signal Name	Type	Size	Frequency	Fan-Out	Description
vdd_adder	IN	1 wire	DC	All	Power supply for the adder
vss	IN	1 wire	DC	All	Global ground connection
A	IN	16bit	400 MHz	2	One of the vectors going to the adder.
B	IN	16bit	400 MHz	2	One of the vectors going to the adder.
Cin	IN	1bit	400 MHz	3	Carry in bit to the adder.
S	OUT	16bit	400 MHz	3	Vector with the sum generated in the adder.
Cout	OUT	1bit	400 MHz	3	Carry out bit from the adder.

2.4 Comparator

The comparator is the main part of the Built in self-test (BIST). It enable us to verify adder result in full speed. The comparator check for equality of the correct sum stored in the *CorrSum* register and the calculated sum of the adder, *S*. If *CorrSum* and *S* from the adder are equal it pulls *BISTout* high, otherwise, an incorrect result, outputs zero. The *BISTout* will be available off chip.
The performance requirement of this block is that it completes the comparison in one clock cycle which corresponds to $5\,\mathrm{ns}$. Both *S* and *CorrSum* will be available at beginning of the cycle. The *BISTout* signal will connected to a flip-flop to eliminate glitches on the output.

2.5 Serial Out

The Serial Out is a small system with three subsystems, SPI_counter, SPI_counter_5bit and SPI_out_regs, whose main task is to send data from the adder to an external device. It receives data parallelly from the adder and sends it away serially. The SPI_out_regs consist of five registers, four of which are used as a cyclic buffer, and the last register is used as an output register where the data is shifted out serially one bit a time. Since the data coming out from the Serial Out is controlled by a serial clock which is much slower than the system clock, the system uses a 3-bit counter, the SPI_counter, that controls the storing of data from the adder. As long as the SPI_counter has not counted up to four, the SPI_out_regs will shift in new data from the adder on every system clock cycle. When SPI_counter has counted to four, it will send a signal to the SPI_out_regs telling it to start shifting and sending serially.

The serial shifting is controlled by the SPI_out_regs and as long as the $serialshift$ signal from the SPI_counter_5bit is low, it will shift and send data serially on every . When the $serialshift$ signal goes high, all data has been sent and the SPI_out_regs will start to parallel shift the buffer registers one step so that the output register receives new data to send. The data that is sent to the output register is also sent to a buffer register. How the shifting in SPI_out_regs is done can be seen in fig. 3.

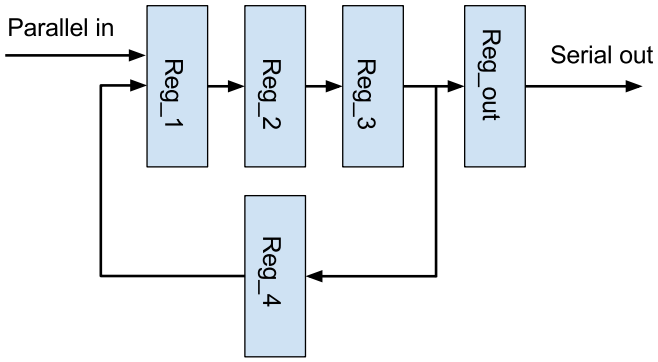

Figure 3: Overview of the shifting in SPI_out_regs.

2.5.1 Definition of signals - Serial Out

Tab. 6 describe the signals going in and out, as well as the internal signals, from the Serial Out system.

Table 6: Definition of SPI_out signals.

Signal Name	Type	Size	Frequency	Fan-Out	Description
vdd	IN	1 wire	DC	All	Power supply for the Serial Out
vss	IN	1 wire	DC	All	Global ground connection
clk	IN	1bit	200 MHz	All	Global system clock.
serial_clk	IN	1bit	kHz	21	Global serial clock
count_enable	IN	1bit	kHz	2	Enable bit to the SPI_counter and SPI_counter_5bit
parallel_in	IN	16bit	200 MHz	1	Data from the adder
spi_out	OUT	1bit	kHz	1	Serial data out
parallel_shift	INTERNAL	1bit	kHz	64	Parallel shift bit from SPI_counter to SPI_out_regs

2.6 Simulation results

Low level blocks used in the system such as 16-bit register, inverters, gates etc. have had their functionality confirmed but no simulation results for them will be displayed here. The blocks simulated are the high level blocks created such as the Shift/PRBS-registers, the adder, the comparator. A complete system simulation confirming its functionality is also displayed.

2.6.1 Serial Loading

In fig. 4 a part of a simulation when loading the PRBS registers is seen. As seen in the figure, when *spi_enable* is high, the vectors A, B and corr_sum is being loaded. As soon as serial_enable goes low, the serial shifting stops, the parallel shifting start and the PRBS-registers starts to listen to the system clock instead of the serial clock.

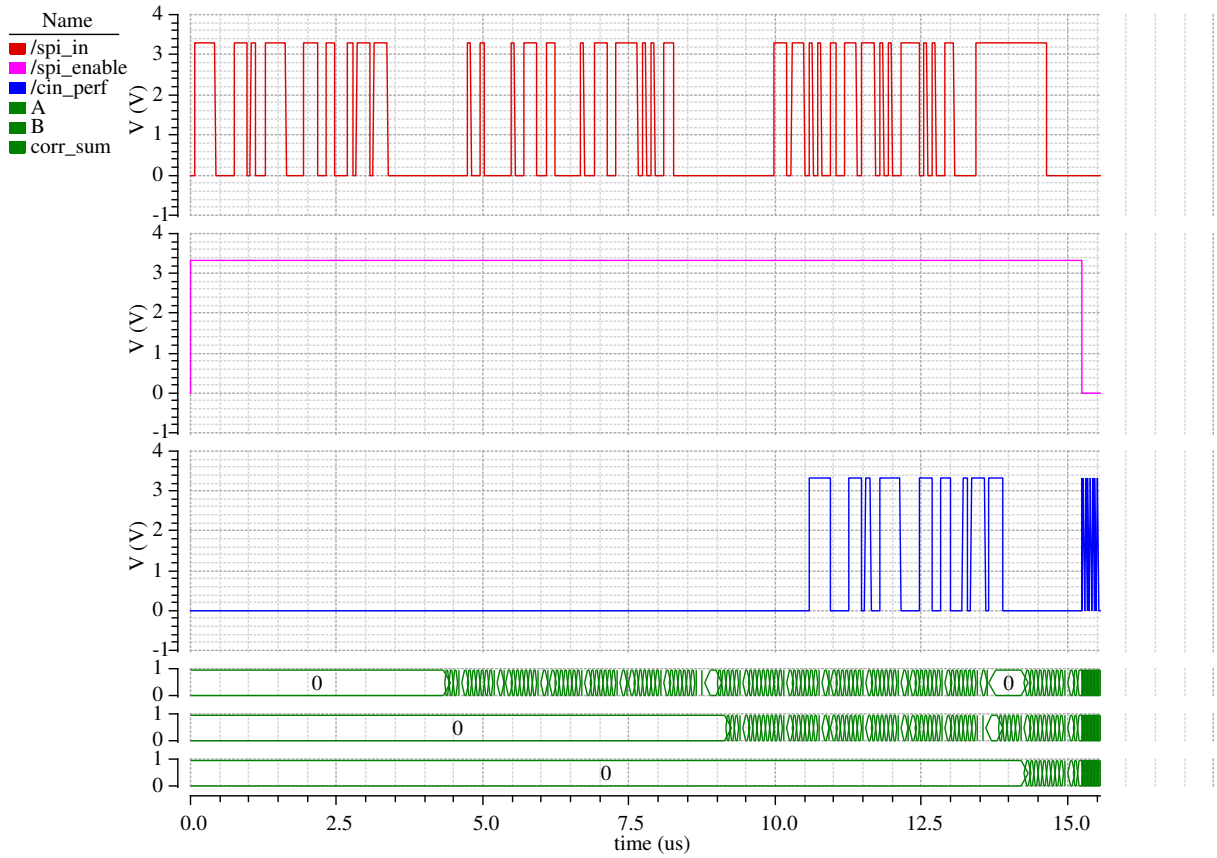

Figure 4: Serial loading of shift registers.

2.6.2 Adder

In fig. 5, four additions are shown. The input vectors A and B are represented as a bus and the input carry is shown as a wave. The sum vector is represented as a bus as well, and the result can be seen as the value of the bus. The delay time of the adder is very short, since it it specified to operate at a very high frequency. The adder have been tested on 8000 values and gives the correct result each time.

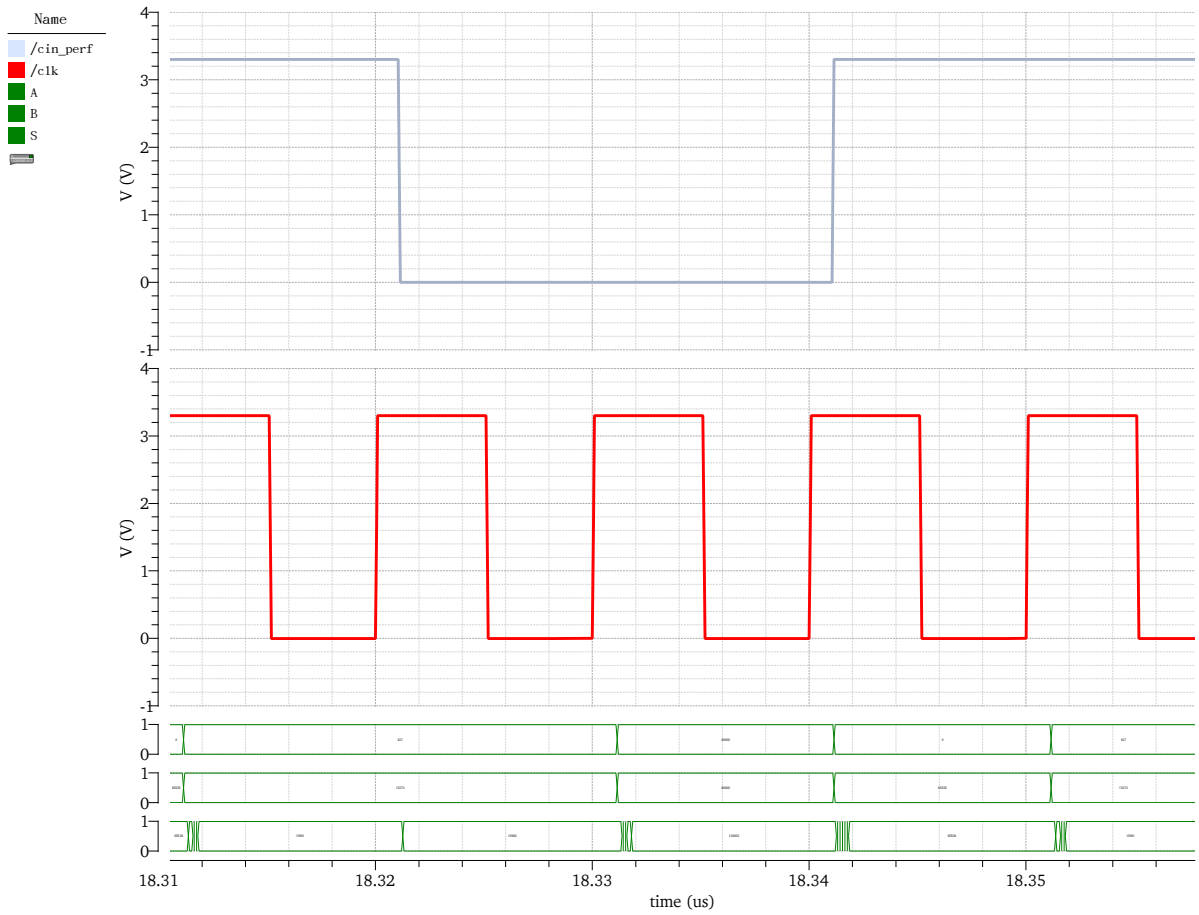

Figure 5: Simulation of adder.

2.7 Risks and delays

There are risks in the high-level design that might affect the project in the later design stages. First of all the high-level design that is created might not be practically possible to implement later on for different reasons. The high level design might not fit on the assigned area on the chip, the assumed speed of different parts might not be practically possible to achieve and it is possible that the initial plan is not achievable in the later stages. This would force us to return to the high level design at a later stage and redo it. These risks can be reduced by careful planning in the early stages of the project and by estimating the performance of the circuit to evaluate if the circuit is designed reasonably. A risk analysis and a what-if plan can be written to assess consequences of each risk and a potential solution to each scenario that can arise.

3 Transistor design

This section describes the transistor level implementation of the circuit and the simulation results as well as sizing strategies.

3.1 Gates

Almost all of the leaf-cells are basic gates. The gates used are listed in tab. 7. The AND, OR and MUX are not real leaf cells since they consists of combinations of the other gates, but since they are used frequently in the project they are mentioned here. Other blocks used in the project are combinations of these basic gates (except for the DFF and the level-shifter).

Table 7: Basic gates used.

Gate	Inputs	Outputs
INV	in	out
AND	A, B	out
NAND	A, B	out
OR	A, B	out
NOR	A, B	out
XOR	A, B	out
XNOR	A, B	out
MUX	A, B, control	out

Since the transistor widths of the gates will be different depending on where they are used, the simulations were done just to verify the functionality of the gates. The sizing was done using global variables in each block where the gates where used. This is documented in the following chapters. The only exception to this is the DFF. Transistor widths were kept at $1\,\mu\mathrm{m}$ in all blocks except the levelshifter and the DFF.

3.1.1 DFF

The D-type flip-flop (DFF) used in the project was copied from the course lab. The transistor schematic of the DFF and the transistor widths can be seen in appendix A, fig.22. The length of all the transistors is $0.35\,\mu\mathrm{m}$.

3.1.2 Level shifter

When testing the adder in sub-threshold the adder supply voltage will be lower then for the rest of the circuit. The output of the adder has to be translated from the low voltage domain before entering the high level domain of the serial-out registers. The conversation are handled by the level shifter. The schematic of the level shifter is shown in fig. 23. Since the shifting is only needed while running in sub-threshold it can be bypassed in normal operation. Bypass is activated by setting the *bypass* signal high. This signal is generated in the status register.

The topology of the level shifter is based on the work of [3]. That circuit was implemented in $0.18\,\mu\mathrm{m}$-technology and the transistor sizes had therefore to be changed to comply our $0.35\,\mu\mathrm{m}$-technology. The schematic with transistor sizes is shown in 23. The bypass circuit is realised

with transmission gates. Both input and output of the level shifter are pulled to ground when bypassed to keep it from floating.

The level shifter block is optimised for adding minimal delay when bypassed and maximal voltage shift with reasonable delay when active. I our case an operating frequency of a couple of kHz is allowed in sub-threshold.

3.2 Shift-registers/PRBS

There are five different registers that are used in the circuit. These are 16-bit registers, a 4-bit status register, two 64-bit PRBS registers for the two input vectors, a smaller 4-bit PRBS register for the carry-input and a 64-bit register for the correct sum. These are implemented differently depending on what functionality is needed, e.g. the correct-sum register is a stripped down PRBS-register since the PRBS functionality is not needed. This is done to save space on the chip. Saving size is also the main goal of the transistor sizing. The strategy has been to minimise the size of the transistors in all of the registers to make them as small as possible, while still reaching the speed goals.

3.2.1 16-bit register

The 16-bit register is implemented as a regular 16-bit parallel register by using 16 DFF:s in parallel. The DFF:s are clocked on the system clock and the main function of this register in the circuit is to sync signals before sending them to the comparator.

3.2.2 4-bit status register

The 4-bit status register is implemented using 4 DFF:s in series with the output of the previous DFF being the input of the next DFF. The output of every DFF is also available as output since these are the signals being used to control different parts of the circuit, e.g. if the PRBS-mode is activated or not. It is important that the hold-time and setup-time of the DFF:s are good enough in order to serially connect them. This has been tested and confirmed in simulations.

3.2.3 64-bit PRBS register

The 64-bit PRBS register is split into 16 identical 4-bit register blocks. Each of these blocks are implemented with four DFF:s in serial each. Additionally each 4-bit block contains one 2 input multiplexer. The PRBS-register has to be able to shift in new data or shift the current data around between the DFF:s and in order to obtain this functionality a multiplexer is present on the input of the first DFF.

Additionally the 64-bit PRBS register contains a 4-bit input XNOR in order to randomise the content of the register. The bits 64, 63, 61 and 60 are input to the XNOR and the output of the XNOR is fed into a input multiplexer which controls if the content is randomised or not. The 64-bit register also contains inverters to buffer signals.

3.2.4 4-bit carry-input register

The 4-bit carry input is very similar to the 4-bit register blocks of the PRBS register. The main difference is that the PRBS functionality is handled in the 4-bit register by a two input XNOR. The input to the XNOR is the bits 4 and 3 and the output of the XNOR is fed into two 2 input multiplexers who control if the content of the register is randomised or not and if new data is loaded or not.

3.2.5 64-bit correct-sum register

The 64-bit correct-sum register is very similar to the 64-bit PRBS register. The only difference is that the 64-bit correct-sum register does not have the PRBS functionality that the 64-bit PRBS register does. This means that it does not contain any XNOR gate for randomisation.

3.3 Kogge-Stone Adder

The transistor design has been very straight-forward in the sense that the high level gates only had to be translated to the corresponding transistor schematic. The main difficulty of the transistor design has been to size the transistors to achieve the target speed. This has for the most part been done with cadence, by using the built-in optimising tools with expressions for the propagation delay. The power consumption of the block has also been calculated. The final results of the optimisation can be seen in section 3.6.4. Only the width of the transistors has been considered when sizing. The lengths of the transistors has been kept at the process minimum at $0.35\,\mu\mathrm{m}$ since increasing the length of the transistors only have negative effects on the speed, which is the main design factor. The standard size for all transistors used in the adder is $18\,\mu\mathrm{m}$ for the PMOS transistors and $9\,\mu\mathrm{m}$ for all the NMOS. Only the XOR and the XNOR gates have used the specification maximum size of $20\,\mu\mathrm{m}$ and $10\,\mu\mathrm{m}$ for the PMOS and the NMOS size, respectievly. In total, the adder circuit is composed of 1216 transistors.

3.4 Comparator

The comparator was implemented with a tree structure of XOR-, NOR- and NAND-gates. These where ordered XOR, NOR, NAND, NOR, NAND, inverter from input to output. This topology reduces the required number of inverter stages compared to using only XOR- and AND-gates. Because of that the propagation delay and area on chip is minimised.

The gates was sized equivalent to an inverter with a PMOS width of $1\,\mu\mathrm{m}$ and a NMOS width of $0.4\,\mu\mathrm{m}$. This resulted in rise and fall times about the same and a small layout due to the minimum sized NMOS.

A fast mock-up of the block layout indicates that $6000\,\mu\mathrm{m}^2$ or $2.3\,\%$ of available chip area will be occupied by this block.

3.5 Serial Out

The SPI_counter_5bit has been moved out from the SPI_out_regs block. All the counter cells in both counters are modified to be as small as possible with as few transistors as possible. The

standard counter cell is made up by a T flip-flop (TFF) and an AND gate. A TFF is built up by a D-type flip-flop (DFF) and an XOR gate. The modified counter cells have either their $bout$ or $cout$ signal removed depending on where in the circuit they are located. The output of the SPI_counter that tells the system when all the data from the adder has arrived, is fed back to its input through a NAND together with its $enable$ signal. This NAND-gate is sized so that the SPI_counter does not continue counting when the next rising edge of the clock comes.

The transistor sizes of the DFFs used in SPI_out_regs are the same as in section 3.1.1. The reason for this is that there is no need for the DFFs to be larger than necessary. Also, with these sizes the DFFs can be directly connected together. MUXs used in the SPI_out_regs are sized small but large enough for the overall system to work as intended.

A clock buffer is used for the internal $parallel\_shift$ signal since it has a very large fan-out. The buffer is made from two inverters. In order for the last inverter to be able to drive all the registers, the inverters are sized as follows. The first inverter sizes are $18\,\mu\mathrm{m}$ PMOS width and $9\,\mu\mathrm{m}$ NMOS width and the seccond inverter have the sizes $48\,\mu\mathrm{m}$ PMOS width and $24\,\mu\mathrm{m}$ NMOS width.

All the blocks that responds to the $serial\_clk$ are sized to be small, since they don't need to be fast. Both counters in the Serial Out block start counting when the serial loading is done, i.e. when the spi_enable signal goes low.

The SPI_out_regs has been broken down into smaller blocks and the internal parallel_shift signal has been moved out to simplify routing and placing in layout view of the top level of the Serial Out.

3.6 Simulation results

This section describes the results of the simulations for each circuit.

All circuits in the project are tested in nominal operation, and process corners. Nominal operation is the manufacturer defined standard mode of operation with manufacturing mean properties. This includes a supply voltage of $3.3\,\mathrm{V}$ and a temperature of $27\,^{\circ}\mathrm{C}$. The process corners which in some way represents worst case scenarios such as worst speed and worst power. It may also represent different temperatures and input voltages. The temperature tests ranges from $25\,^{\circ}\mathrm{C}$ to $110\,^{\circ}\mathrm{C}$ and supply voltage is tested for the range $3.3\,\mathrm{V} \pm 10\%$

3.6.1 Level shifter

The level shifter is simulated in two modes, with active level shifter circuit and bypassed. We are interested in the delay added in both operation modes and the lowest possible input voltage in sub-threshold mode.

With bypass activated the level shifter adds a delay of $102\,\mathrm{ps}$ on rising edge and $23\,\mathrm{ps}$ on falling edge. The rise and fall times are $436\,\mathrm{ps}$ and $329\,\mathrm{ps}$ respectively. The input to output response is shown in fig. 6.

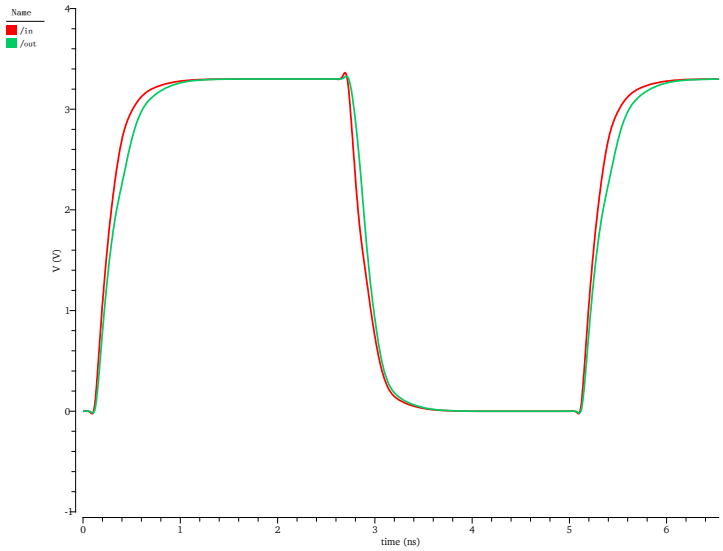

Figure 6: Input to output for level shifter in bypass mode.

The level shifter is functional down to an input voltage of $0.4\,\mathrm{V}$. For that input voltage a delay of $70\,\mu\mathrm{s}$ on rising edge and $55\,\mu\mathrm{s}$ on falling edge. The rise and fall times are $0.33\,\mu\mathrm{s}$ and $2.3\,\mathrm{ps}$ respectively. The input to output response is shown in fig. 7.

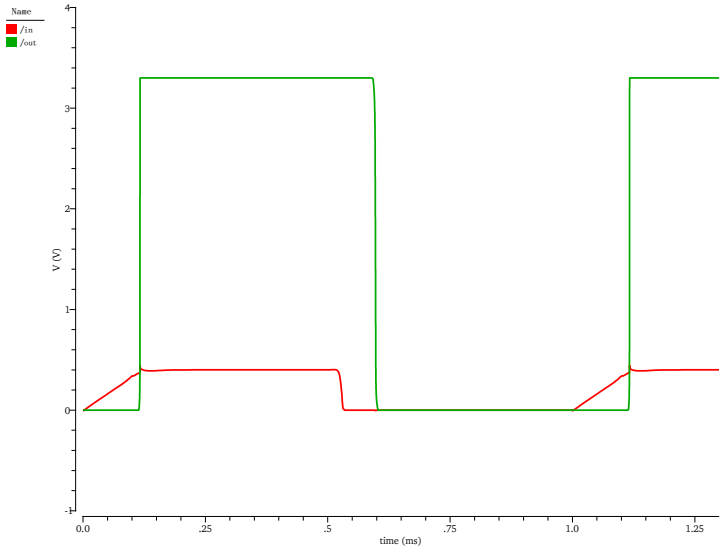

Figure 7: Input to output for level shifter in active mode with an input signal of $0.4\,\mathrm{V}$.

3.6.2 D Flip Flop

The D flip flop has a nominal rise and fall time below $100\,\mathrm{ps}$. This means it can handle speeds well above the specified $200\,\mathrm{MHz}$. Even in corners such as Worst case speed and when the voltage is dropped to $3\,\mathrm{V}$ the rise and fall times never exceeds $250\,\mathrm{ps}$. Tab. 8 shows the simulation results for the D flip flop, both nominal and in some of the simulated corners.

Table 8: Rise and fall time of the D flip-flop.

	Nominal	Worst case speed	Vdd = 3.0 V
Rise delay	62.93 ps	112.5 ps	167.4 ps
Fall delay	67.91 ps	137.6 ps	132.5 ps

3.6.3 Shift-registers/PRBS

The 64-bit PRBS registers and the 4-bit PRBS register blocks are the most complex registers that are used as the other registers are scaled down versions of these. Due to this most tests and simulations have been done one the PRBS register and simply confirmed on the other registers. If the 64-bit PRBS register and the 4-bit PRBS register block passed the requirements, so did the other registers.

For the registers the required speed of operation is at least 200 MHz and the timing in question is between the clock flank and the updated data on the output. The tests and simulations have been ran with the goal to have some margin to the required speed since the actual speed will decrease when the registers are created in the layout with parasitics. This means that higher speeds has been tested and simulations have been run in worse corners than the nominal corner in order to confirm functionality. The goal was to keep the propagation delay between the clock flank and the updated data flank below 2 ns.

The simulation results for simulations for the 4-bit PRBS register block in different process corners can be seen in tab. 9. The register was tested at three different voltages and in two different temperatures in the typical mean, the worst speed case and the worst power case corners. As expected the worst delays appears in the worst speed corner at the highest temperature and at the lowest voltage but the delay is still within the requirements.

*Table 9: Speed of the PRBS circuit over process corners. The goal was to keep the delay under 2*ns.

		Vdd = 3 V	Vdd = 3.3 V	Vdd = 3.6 V
Corner	Temp. [°C]	Delay [ps]	Delay [ps]	Delay [ps]
Typical mean	25 110	905 1125	839 1049	789 989
Worst case speed	25 110	1376 1716	1254 1575	1161 1465
Worst case power	25 110	534 661	509 631	490 607

Fig. 8 shows the simulation results of the 4-bit PRBS register block in the typical mean corner at a temperature of 25 °C and a voltage of 3.3V.

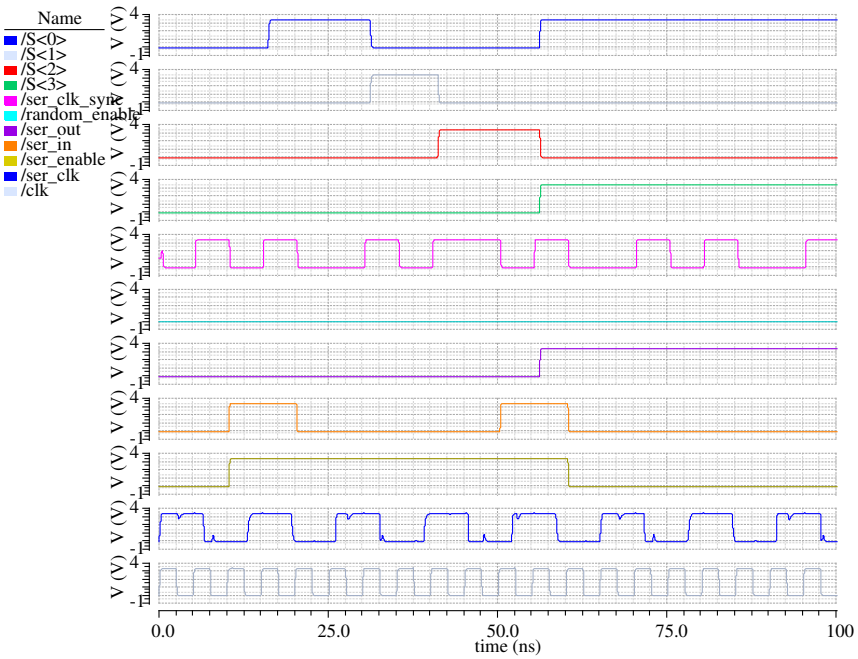

Figure 8: Simulation result of the 4-bit PRBS register block.

3.6.4 Kogge-Stone Adder

The speed of the Kogge-Stone adder is calculated as one over the highest propagation delay from the input bit to the output bits. The test is constructed in such a way that in the initial condition, the A vector is only zero, the B vector is only one and the carry-in is zero. Since the carry-in is not in the critical path, the carry-in to carry-out performance will be higher than the result of this test. In the first time step, the first bit in the A vector is set to one. This will cause the carry to propagate through the whole circuit. The delay to all output bits is measured. In the next time step, the first bit of the A vector is set to zero again. These cases are the slowest cases for the adder for falling and rising times respectively. The highest propagation delay is inverted and the speed of the circuit is obtained. The slowest speed of the circuit in different corners as well as the average power consumption of the adder circuit can be seen in tab. 10.

Table 10: Speed of the adder circuit over process corners.

		Vdd = 3 V		Vdd = 3.3 V		Vdd = 3.6 V	
Corner	Temp. [°C]	Speed [MHz]	Power [mW]	Speed [MHz]	Power [mW]	Speed [MHz]	Power [mW]
Typical mean	27	745	11.1	821	13.6	889	16.4
	50	702	11.3	771	13.9	835	16.7
	75	661	11.5	725	14.1	783	17.0
	110	611	11.8	670	14.4	722	17.3
Worst case power	27	1200	11.0	1290	13.5	1370	16.3
	50	1130	11.6	1210	13.7	1370	16.3
	75	1060	11.3	1140	13.9	1200	16.8
	110	980	11.6	1040	14.2	1100	17.1
Worst case speed	27	510	11.7	570	14.5	625	17.3
	50	480	11.9	535	14.5	586	17.5
	75	451	12.0	502	14.7	549	17.7
	110	417	12.3	463	15.1	505	18.1

The speed of the adder is over the specified speed $400\,\mathrm{MHz}$, even in the worst case. The worst case scenario provides a lower limit on the performance of the adder which must be taken into account when designing the circuit. The adder was also simulated in sub-threshold, where the input voltage is lower than the threshold voltage of the transistors in the process. In this region the linear approximation of the transistor gives zero current through the transistor, but higher level approximations gives an exponential decrease in current with respect to the source-drain voltage. Designing a circuit for sub-threshold will decrease the current draw of the transistors in the current and sacrifice the circuit speed, with the goal that the energy used per cycle should be minimized. In sub-threshold the simulated speed is slower, but this behaviour is expected and the only requirement in sub-threshold mode is that the circuit gives the correct result eventually. The power calculation for all but the sub-threshold operation is performed on a circuit that switches at $200\,\mathrm{MHz}$, which is the normal mode of operation for the adder. In sub-threshold operation, the circuit switches at $20\,\mathrm{kHz}$.

Fig. 9 shows the nominal transient response of the sum output bits as well as the carry-out bit in the first case where the first bit of the A vector is set to one at the time $10\,\mathrm{ns}$. The bits are initially flipping to a high state and after an amount of time the output sum bits reach the correct value of zero and the carry-out signal reaches the correct value of one. The propagation in this case can be calculated to approximately $1.2\,\mathrm{ns}$.

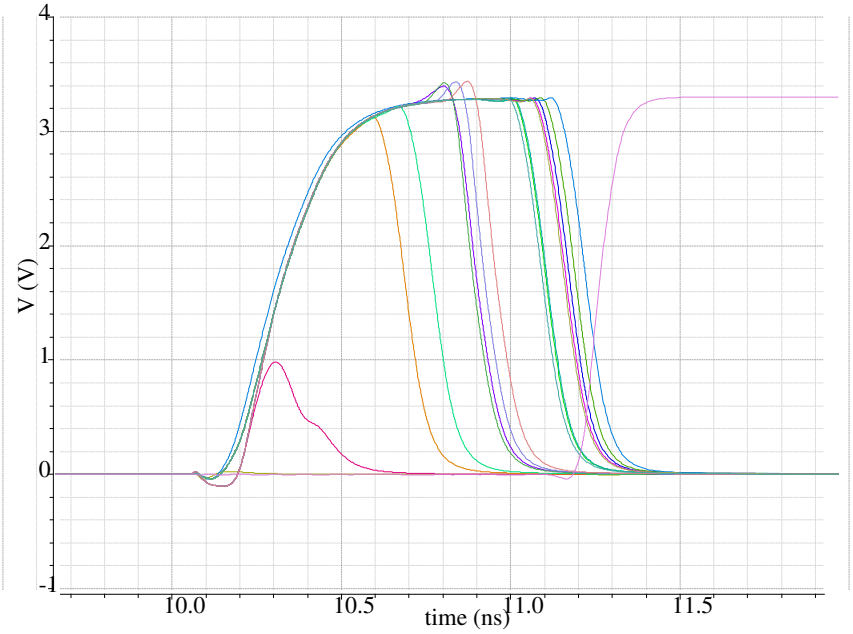

Figure 9: Nominal simulation of the Kogge-Stone adder 1.

Fig. 10 shows the nominal transient response of the output sum bits as well as the carry-out bit in the second case where the first bit is set to zero at the time $20\,\mathrm{ns}$. This time, the bits do not initially flip, but the rise time is slower than in the first case. The propagation delay in this case can be calculated to approximately $1.3\,\mathrm{ns}$.

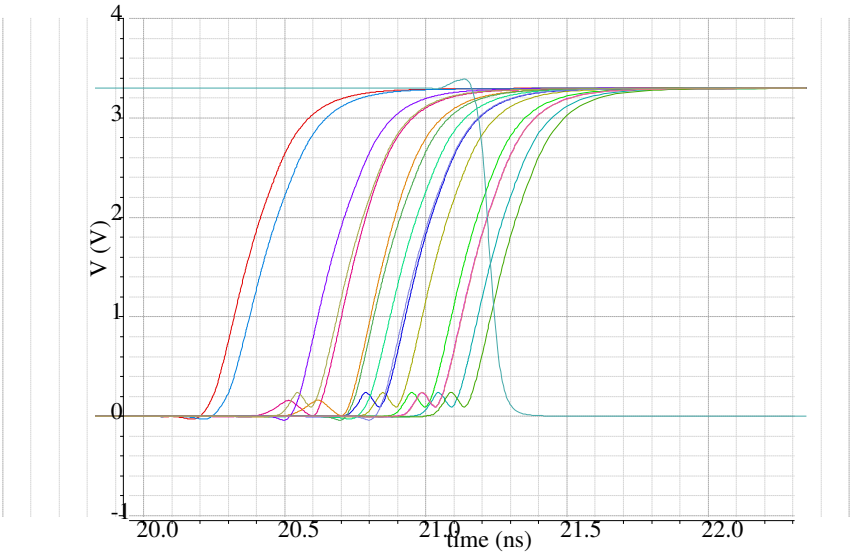

Figure 10: Nominal simulation of the Kogge-Stone adder 2.

Fig. 11 shows the sub-threshold transient response in the first case described above. The input voltage is in this case set to $0.5\,\mathrm{V}$. The circuit is considerably slower, but the function of the adder is still correct. The propagation delay can be calculated to $29\,\mu\mathrm{s}$.

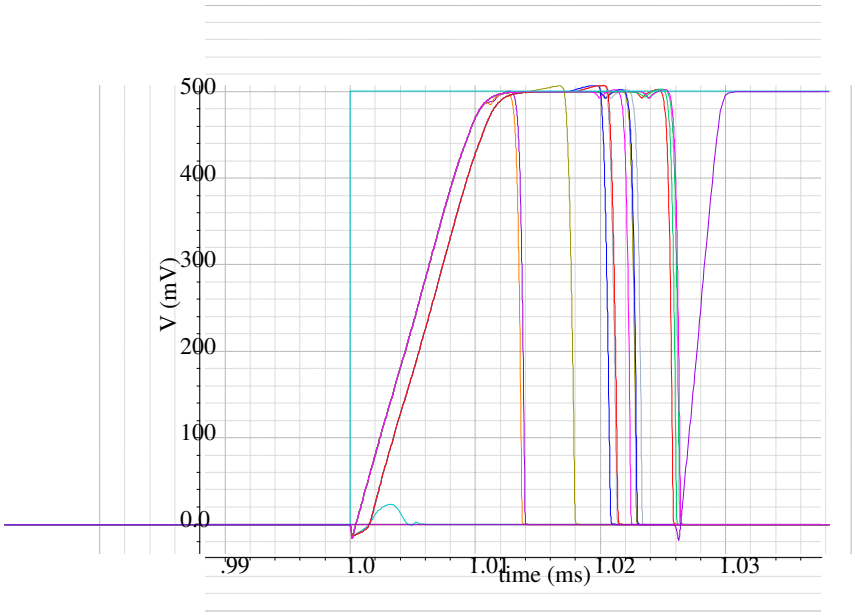

Figure 11: Sub-threshold simulation of the Kogge-Stone adder 1.

In fig. 12, the sub-threshold transient response for the second case can be seen. The input voltage is again set to $0.5\,\mathrm{V}$. The propagation delay in this case can be calculated to $35\,\mu\mathrm{s}$.

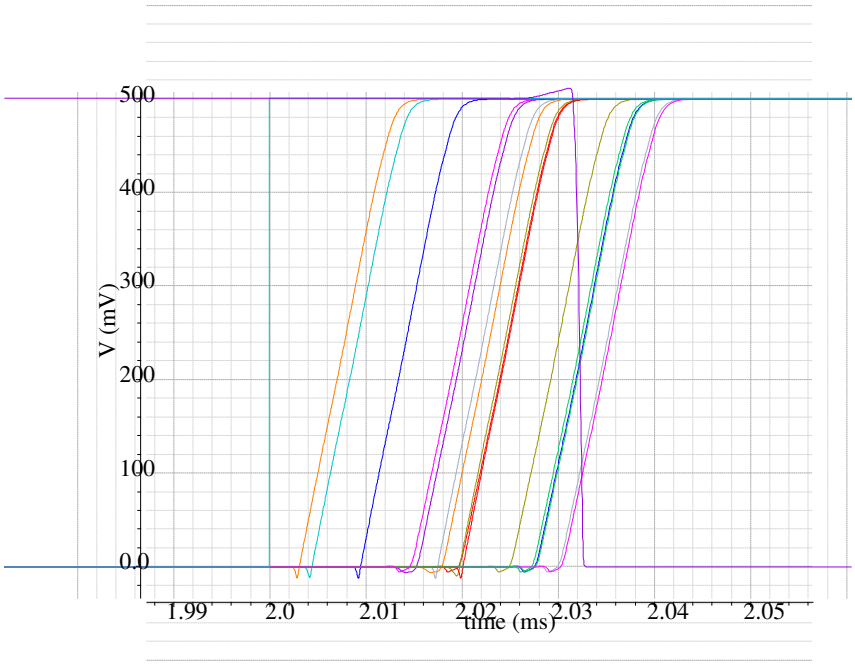

Figure 12: Sub-threshold simulation of the Kogge-Stone adder 2.

3.6.5 Comparator

The critical path of the comparator can be analysed by flipping one bit on the input. Since all paths have the same length and delay any input will activate the longest path. The test vectors was changing one bit to one with all other as zero and changing one out of all ones to zero.

The simulation result is shown in fig. 13. The internal signal before the flip-flop (blue) are glitching in transition but settles well before the next rising clock edge (red). The output signal (yellow) has passed through the flip-flop and is free of glitches. The delay are measured from rising clock edge to the point where the internal signal is $10\,\%$ from its final value.

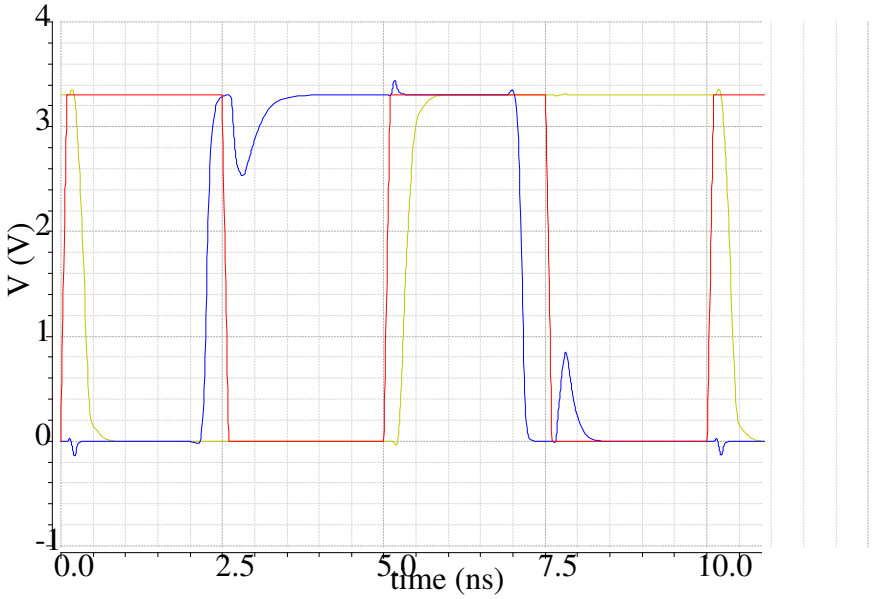

Figure 13: Delay of comparator output before DFF.

Nominal and worst case delay are shown in tab. 11. Worst case is realised with a temperature of $110\,^{\circ}\mathrm{C}$, supply voltage of $3\,\mathrm{V}$ and Monte Carlo-simulation of mismatch "worst zero" and "worst one". For all cases the delay are well below $5\,\mathrm{ns}$.

Table 11: Comparator delay and power consumption.

		Vdd = 3 V			Vdd = 3.3 V			Vdd = 3.6 V		
Corner	Temp. [°C]	Rise [ns]	Fall [ns]	Power [µW]	Rise [ns]	Fall [ns]	Power [µW]	Rise [ns]	Fall [ns]	Power [µW]
Typical mean	27	2.436	2.257	0.928	2.199	2.049	1.359	2.199	2.049	1.359
	50	2.523	2.331	0.900	2.275	2.114	1.319	2.275	2.114	1.319
	75	3.119	2.409	0.943	2.357	2.183	1.301	2.357	2.183	1.301
	110	3.192	2.517	0.901	2.472	2.28	1.391	2.472	2.28	1.391
Worst one	27	2.478	2.225	0.955	2.231	2.017	1.401	2.231	2.017	1.401
	50	2.565	2.293	0.926	2.306	2.077	1.359	2.306	2.077	1.359
	75	3.188	2.365	0.971	2.388	2.142	1.421	2.388	2.142	1.421
	110	3.267	2.463	0.946	2.502	2.231	1.383	2.502	2.231	1.383
Worst power	27	1.845	1.724	0.877	1.744	1.64	1.3	1.744	1.64	1.3
	50	1.893	1.765	0.884	1.788	1.676	1.383	1.788	1.676	1.383
	75	1.946	1.808	0.943	1.836	1.715	1.392	1.836	1.715	1.392
	110	2.017	1.869	0.956	1.902	1.77	1.407	1.902	1.77	1.407
Worst speed	27	3.746	3.394	0.946	3.253	2.636	1.377	3.253	2.636	1.377
	50	3.923	3.552	0.953	3.364	3.11	1.386	3.364	3.11	1.386
	75	4.109	3.721	0.961	3.519	3.23	1.396	3.519	3.23	1.396
	110	4.36	3.953	0.971	3.742	3.432	1.41	3.742	3.432	1.41
Worst zero	27	2.264	2.145	0.906	2.073	1.979	1.33	2.073	1.979	1.33
	50	2.344	2.215	0.913	2.142	2.041	1.339	2.142	2.041	1.339
	75	2.429	2.292	0.921	2.217	2.108	1.348	2.217	2.108	1.348
	110	2.545	2.398	0.932	2.322	2.202	1.362	2.322	2.202	1.362

3.6.6 Serial out

One of the most critical parts of the Serial Out is the signal that signals when all the data from the adder has arrived, called $count\_stop$. The delay compared to the clock edge has to be sufficiently small in order for the system to function correctly. This delay has been measured over several corners and temperatures. In fig. 14 this signal can be seen where red is nominal and yellow worst case speed and as seen, the signal has correct behaviour even in worst case speed with a temperature of $110\,^{\circ}\mathrm{C}$.

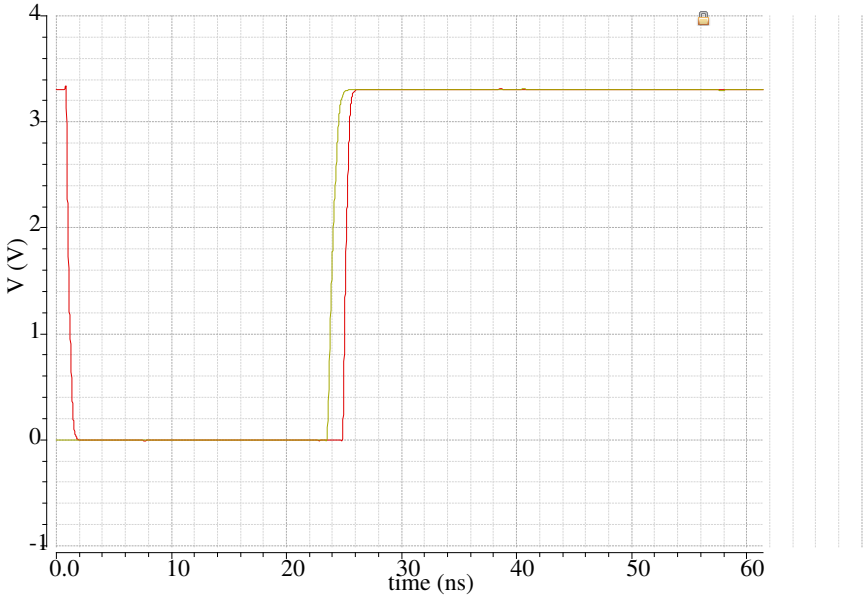

Figure 14: Count_stop simulation in nominal and worst case speed.

In tab. 12 we can see the rise time and delay in both nominal and worst case speed for the $count\_stop$ signal.

Table 12: Delay times for the count_stop signal in different temperatures and corners.

Temperature	Nominal	WSpeed	WPower	WOne	WZero	Monte Carlo	Typ. Mean
27 °C	557.2 ps	754.5 ps	632.2 ps	634.3 ps	469.9 ps	553.4 ps	557.2 ps
110 °C	660.4 ps	936.9 ps	439.3 ps	768.9 ps	581.8 ps	678.3 ps	683.2 ps

In tab. 13 the power consumption in the Serial Out is shown, measured as the average current over the voltage source multiplied with a Vdd of 3.3 V.

Table 13: Power consumption for the Serial Out in different temperatures and corners

Temp.	Nominal	WSpeed	WPower	WOne	WZero	Monte Carlo	Typ. Mean
27 °C	10.9 mW	11.1 mW	11.7 mW	11.7 mW	10.2 mW	11.2 mW	10.9 mW
110 °C	3.5 mW	24.3 mW	12.6 mW	12.4 mW	11.0 mW	11.9 mW	11.9 mW

3.6.7 Top level

After all the subsystems where done, simulated and verified, the complete system was simulated. Fig. 15 shows a simulation of the system where the registers has just finished loading (*spi_enable* gets pulled low). When the serial loading is done the registers starts shifting their bits in parallel to the adder and the adder calculates the sum. As seen in fig.15 the calculated *sum* and *CORRSUM* is identical and the *bist_out* is constant high. This verifies the functionality of the system.

This simulation is done in nominal mode with Vdd at 3.3V.

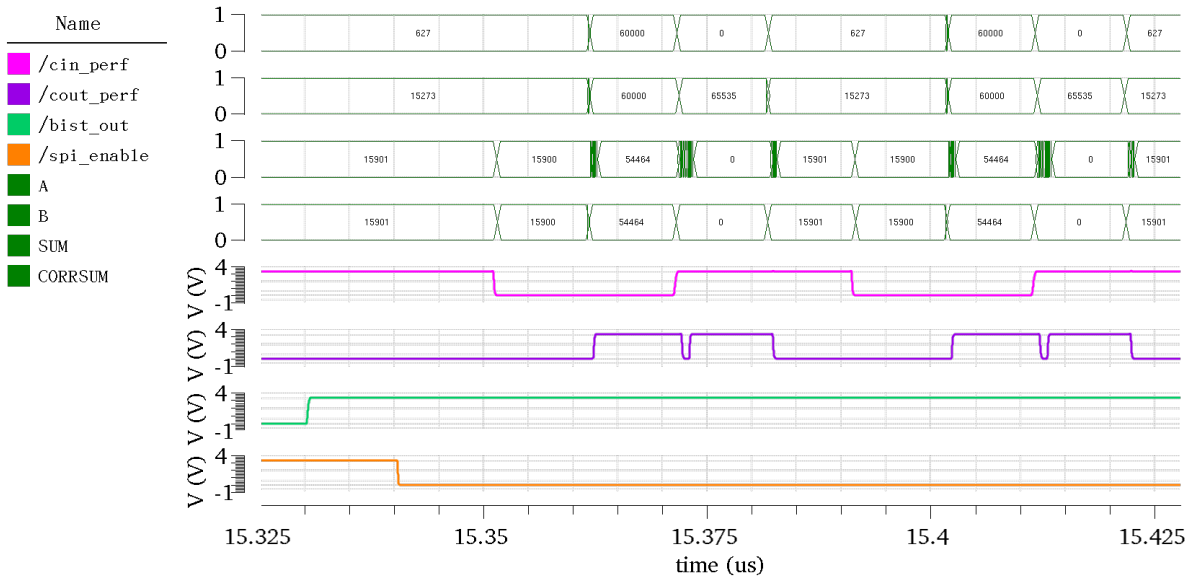

Figure 15: Complete system simulation verifying the functionality.

3.7 PAD assignment / Early test plan

The pad assignment of the signals to the chip is listed in tab. 1 in the high level design. These signal are the signals in to and out from the chip, which is equivalent to the pins/pads of the chip. The table also includes the nominal frequency of the signal mapped to the pin in normal adder operation. All external voltages are assumed to be the nominal voltage of the process, that is $3.3\,\mathrm{V}$. The only exception is in sub-threshold mode, when the voltage of the vdd_adder, cin_perf, $cout\_perf$ and $S < 15 >$ will be lower than the threshold voltage of the transistors in the process, which is below $0.5\,\mathrm{V}$. The nominal clock speed is $200\,\mathrm{MHz}$.

To test the chip in the non-random mode serial data must be loaded to the chip through the serial interface of the chip. The external test bed must provide the serial clock in addition to the serial data. To initiate a serial transfer to the chip, the spi_enable signal must be pulled high. The desired numbers to be added must be loaded during the next 204 serial clock cycles to the spi_in pin. The order of the data is specified in tab. 3. The data on the spi_in must be changed on the negative flank of the serial clock. This is due to the circuit reading data on the positive flank of the spi_clk. The spi_enable must in all other cases be low. After the transfer is complete, the adder will perform the additions of the numbers and start to send the calculated sums back to through spi_out pin.

To test the chip in random mode, the loading of the chip is identical to the loading of the chip in non-random mode, with the exception that the random-shift bit in the control register should be set. In this mode, the chip won't send any signals back through the spi_out pin, but will still be calculating the values internally. The performance of the chip may be tested by measuring the delay between stable data on cin_perf and $cout\_perf$ or $S < 15 >$.

In sub-theshold mode, the adder circuit will not be able to drive the buffer to the serial out block. The function can still be tested with the help of the level shifter included in the chip. The level shifter bypass must be disabled to use the level shifter. Otherwise the chip works just as in other operating modes with the exception of much lower speed at which the adder operates. This must be compensated for in the test bed by reducing the clock speed of the chip. The sub-threshold

performance may vary and the clock frequency used in test of the adder in sub-threshold mode must be measured with the physical chip.

3.8 Risks and Delays

During the high level design phase, it may be possible that some circuits have requirements which are not possible to implement due to process restrictions and regulations or due to the space limitation on the chip. In the transistor design phase, some of those requirements may be tested more thoroughly and the function of the circuits and the constraints may be verified with accurate models of the transistors. It is easier to check if the requirements are fulfilled with these models. This is not true for all risks. One risk that was identified during the high level design phase was that the circuit may not fit on the given chip, which is hard to verify in transistor design. The one thing that may be checked is the fill area of the transistors which gives a rough estimate, but to get more accurate answers the design must be done in layout. If the design do not fit on the chip, the transistor design and in worst case the high level design must be redone, which introduces a large delay in the project.

4 Final Report

This section describes the layout implementation as well as the full chip implementation and simulation results.

4.1 Project Description

The goal with this project is to design a fully-custom, high-performance, low power VLSI circuit in sub-micron CMOS technology.

The group members should learn the different steps of the IC design flow, including system analysis, simulation, layout implementation and verification.

A block diagram of the final system can be seen in fig. 16.

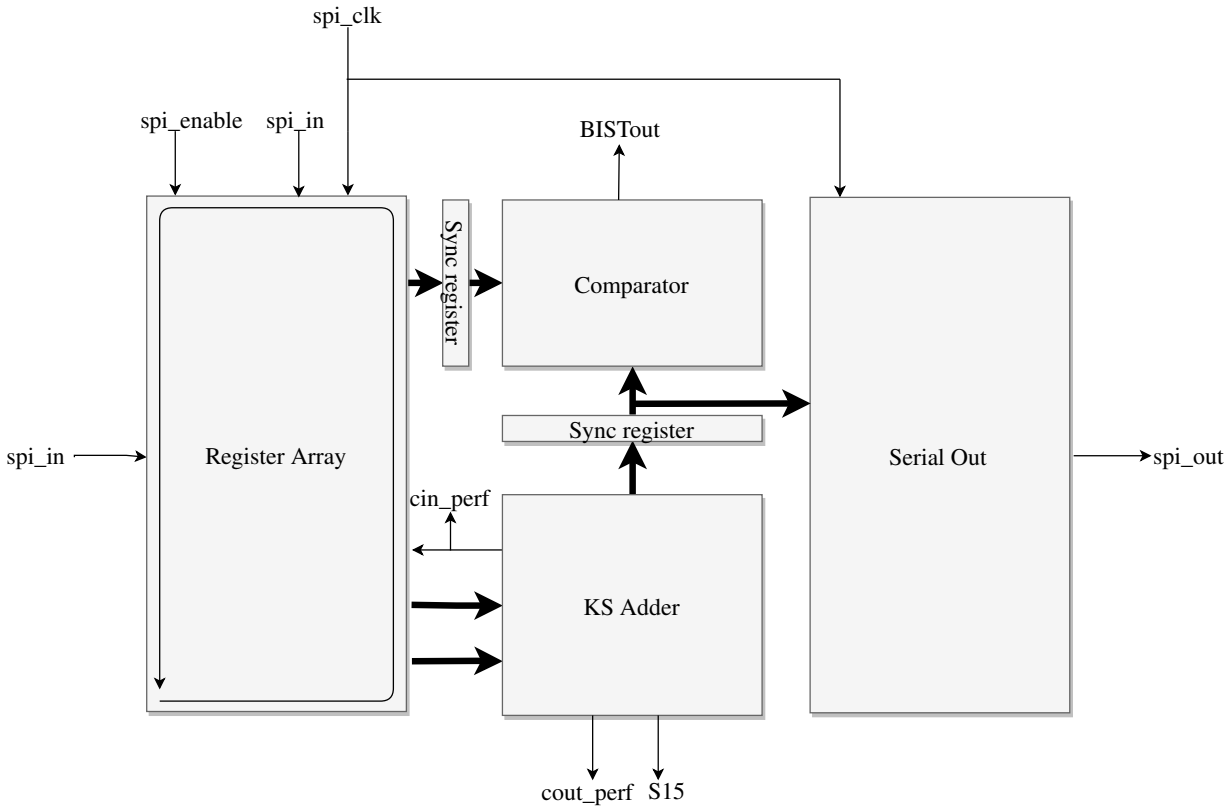

Figure 16: Overview of the system.

4.2 Simulation Results

The performance of the adder was measured by, in the initial state, setting the A register to 0, the B register to 65535, which is all ones, and the carry in to 0. The following clock cycle, the carry in bit was set to 1. This will measure the worst case scenario for the carry in to carry out and the carry in to S_{15} performance when the carry in signal is rising. In the next clock cycle, the carry in bit is set to its initial state 0, and this will measure the worst case for the carry in to carry out and the carry in to S_{15} performance when the carry in signal is falling. All test values can be seen in tab. 14.

Table 14: Test values used.

Value	A	B	Cin	Sum
1	0	1111111111111111	0	1111111111111111
2	0	1111111111111111	1	0
3	0	1111111111111111	0	1111111111111111
4	0	1111111111111111	1	0

By taking the highest propagation delay of the four measurements, a lower bound on the performance can be achieved. The timing of the test signals can be seen in fig. 17. The carry in is represented by the second waveform. The carry out and bit 15 of the sum is represented by the third and fourth waveform, respectively.

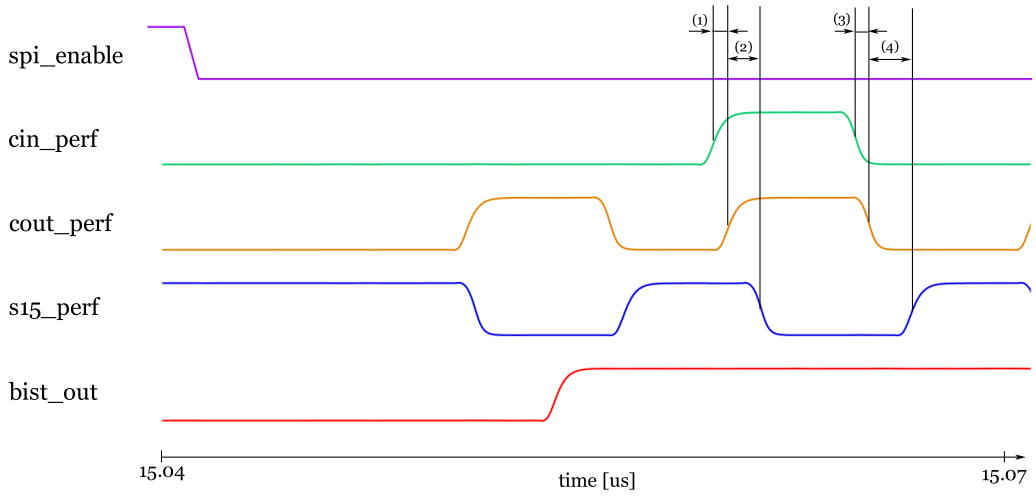

Figure 17: Simulation of the adder with performance test vectors.

The results of the performance measurements performed on the simulated chip layout can be seen in tab. 15. As can be seen, the performance is better than the target performance.

Table 15: Nominal propagation delay of the adder.

Case	Cin to Cout	Cin to S15	Target
Rising	511 ps	1.63 ns	2.50 ns
Falling	473 ps	2.03 ns	2.50 ns

The serial loading and the registers were tested with the same values as the adder test above. The main tests that was performed during the final parts of the project were to make sure that the serial loading and the internal shifting of the registers worked as intended and that the data was loaded correctly. This functionality was tested in the nominal, the worst speed and the worst power corners and both high temperatures around 110 °C and at lower temperatures around 25 °C were tested. The functionality was also tested at different voltage levels ranging from 3 V to 3.6 V. The results of a simulation of the serial loading in the nominal corner can be seen in fig. 18. The serial loading and the internal shifting worked as intended in all of the above cases.

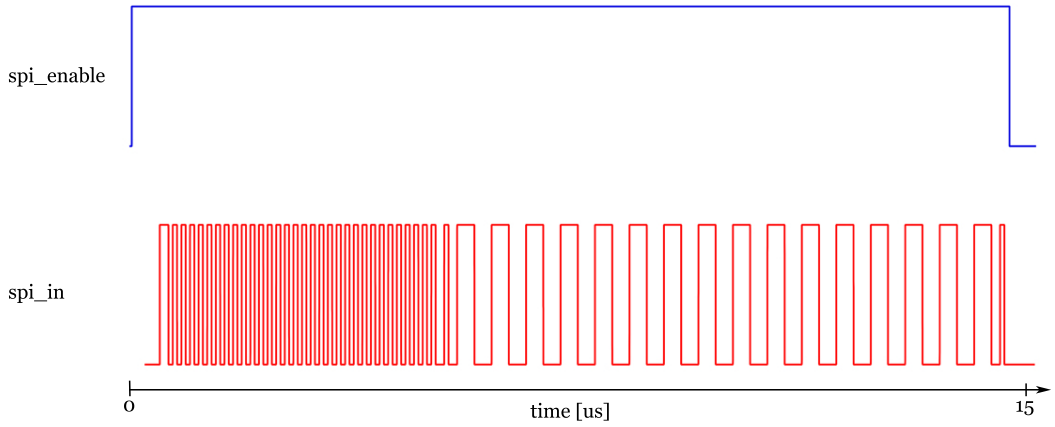

Figure 18: The dataflow during serial load. The bottom graph is the data channel and the top graph is the serial enable signal.

The functionality of SPI_out was tested with the same values as the adder test above. The SPI_out starts collecting values from the adder after the spi_enable signal goes low, it then loads values on the following 5 clock cycles. The first value is a random value that lies in the intermediate register and is not of interest. The other four values are the calculated values. Therefore the first value that is sent from the SPI_out is not of interest, by the same reason. The result from the test can be seen in fig. 19. The spi_enable signal is represented by the top waveform while the result is represented by the bottom waveform. As seen in the figure, the result shows that after spi_enable goes low, a high sequence is sent out on spi_out before the desired sum in tab. 14 is received. SPI_out will continue to send the result until other values are fed into the adder or until powered down.

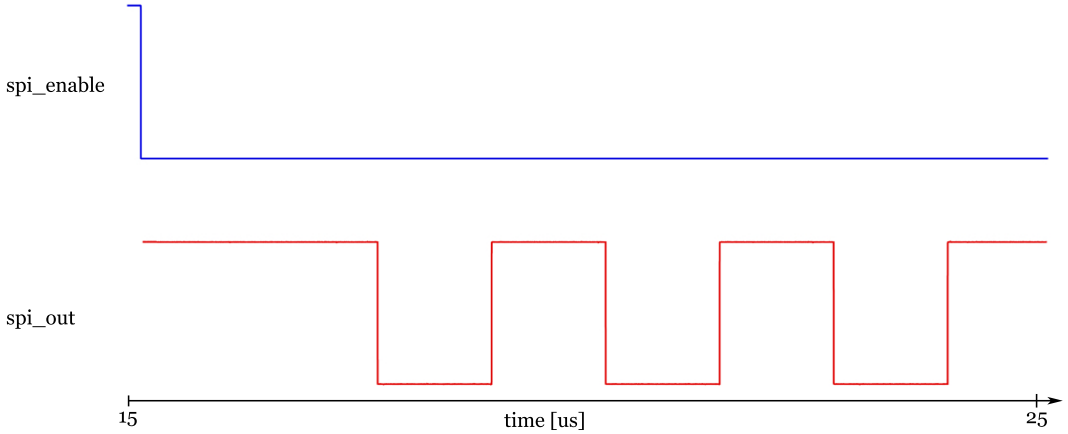

Figure 19: The dataflow during serial transfer from the chip. The bottom graph is the data channel and the top graph is the serial enable signal.

4.3 Evaluation Plan and PAD List

This section explains the evaluation and test plan for the finished chip and contains information about the connectivity. Fig. 20 shows the final layout of the chip including the pads.

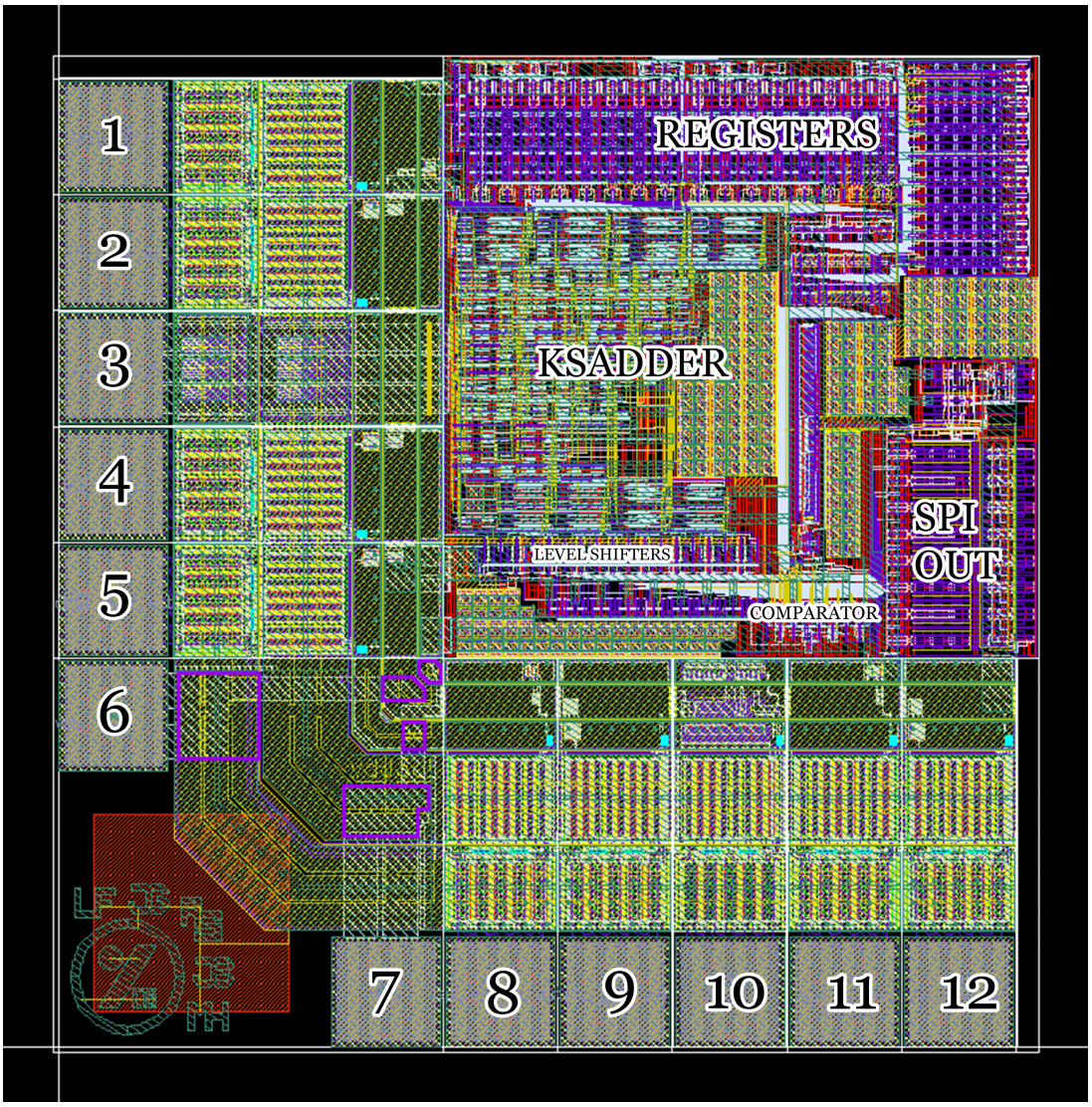

Figure 20: The dataflow during serial transfer from the chip. The bottom graph is the data channel and the top graph is the serial enable signal.

4.3.1 Pins and Pads

Tab. 16 shows a list of the PADs used on the chip and the properties of the respective signals.

Table 16: Pins and their corresponding pads. the numbers can be seen in fig. 20.

Pin Number	Pin	In/Out	Pad	Frequency
1	serial_in_ext	In	ICP	$\sim$10 kHz
2	cin_perf_ext	Out	BUP8P	200 MHz
3	vddKS_ext	InOut	APRIOWP	DC
4	s15_perf_ext	Out	BUP8P	$>$400 MHz
5	cout_perf_ext	Out	BUP8P	$>$400 MHz
6	vdd	InOut	VDDGNDCORNER	DC
7	vss	InOut	VDDGNDCORNER	DC
8	spi_enable_ext	In	ICP	$\sim$10 kHz
9	bist_out_ext	Out	BUP8P	200 MHz
10	clk_ext	In	ICCK8P	200 MHz
11	serial_clk_ext	In	ICP	$\sim$10 kHz
12	spi_out_ext	Out	BUP8P	$\sim$10 kHz

Fig. 21 shows a PCB sketch that explains how each pin should be connected to external parts.

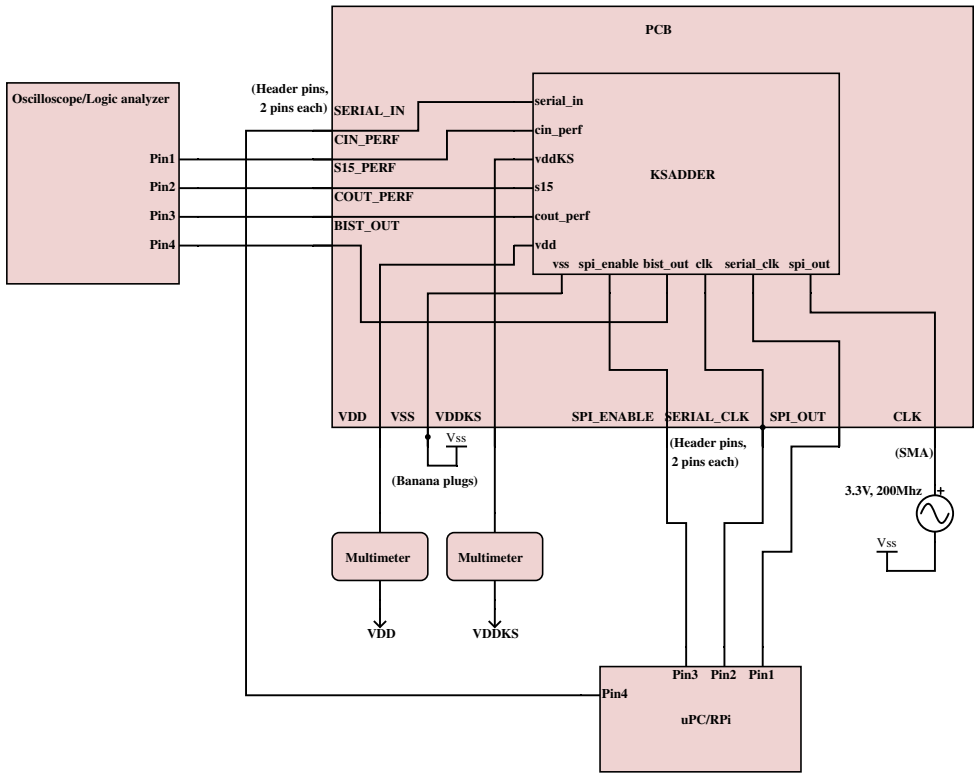

Figure 21: PCB sketch with external connections.

4.3.2 Test Plan

The following is a step by step guide on how to power up and test the finished circuit for the first time. The Arduino/RPi that is used should already be programmed and ready to transmit correct data to the PCB and chip.

1. Power on the multimeters and the oscilloscope or the logic analyser.

2. Power on the PCB with vdd and vddKS at 3.3 V. The power consumption should be 0.
3. Power on the external clock generator and generate a sinus wave at a low frequency ($10 - 100\,\mathrm{kHz}$).
4. Power on the Arduino/RPi.
5. Load the registers on the chip by serially transferring data from the Arduino/RPi. The LS-bypass bit should be 1 and the random-enable bit should be 0. The same values that were used in the layout simulations can be used and can be found in table 14.
6. Confirm that the chip works as intended by checking that the bist_out signal is high.
7. Check the delay between cin_perf_ext and both s15_perf_ext and cout_perf_ext using the oscilloscope or the logic analyser. Extract the speed of the adder-circuit using these values.
8. Check the data that is being sent from the spi_out_ext and confirm that the data being sent is correct.
9. Increase the frequency of the sinus wave from the external clock generator to $200\,\mathrm{MHz}$.
10. Confirm the chip functionality for this frequency using the instructions in steps 6-8.

In order to test the PRBS mode of the chip the test instructions above can be used with only slight adjustments. For point 5. the random-enable bit should be set to 1 in order to enable the random mode. Additionally the data in the registers should only be zeros. It is very important that no register contains only ones as that case would lock the PRBS sequence and the register would be locked into only containing ones. For safety it is recommended to only load zeros in the whole bit sequence, apart from the random-enable bit and the LS-bypass bit. The current draw for the vdd and vddKS pins were $14\,\mathrm{mA}$ and $3\,\mathrm{mA}$ respectively in nominal simulation.

The following is a step by step guide on how to power up and test the finished circuit for sub-threshold functionality. The Arduino/RPi that is used should already be programmed and ready to transmit correct data to the PCB and chip.

1. Power on the multimeters and the oscilloscope or the logic analyser.
2. Power on the PCB with vdd and vddKS at 3.3 V. The power consumption should be 0.
3. Power on the external clock generator and generate a sinus wave at a low frequency ($1\,\mathrm{kHz}$).
4. Power on the Arduino/RPi.
5. Load the registers on the chip by serially transferring data from the ARduino/RPi. The LS-bypass bit should be 0 and the random-enable bit should be 0. The same values that were used in the layout simulations can be used and can be found in table 14.
6. Confirm that the chip works as intended by checking that the bist_out signal is high.
7. Check the delay between cin_perf_ext and both s15_perf_ext and cout_perf_ext using the oscilloscope or the logic analyser. Extract the speed of the adder-circuit using these values.

8. Check the data that is being sent from the spi_out_ext and confirm that the data being sent is correct.
9. Decrease the vddKS voltage level towards 0.3 V in steps of 0.1 V.
10. Confirm the chip functionality for the selected vddKS voltage level using the instructions in steps 6-8.

A very early version of the testcode for an Arduino can be found in appendix B.

4.4 Risks

The main risks that could lead to the produced chip not fully working as intended mostly come from the uncertainty that exists during the production of small integrated circuits. One of the main risks is that the vias that are used to connect different layers on the chip break during production. This has been a concern during the design of the chip and multiple vias has been used when possible in order to minimise this risk.

Another big risk is that our chip will get produced on a silicon chip that has lower than nominal electrical properties. This would lead to higher propagation delays for signals and overall worse performance which could lead to chip malfunction. This has also been considered during the design and the chip has been simulated in order to make sure that it should work in worse chip-corners as well.

One main risk that was brought up during the transistor design of the chip was that the circuit might not fit on the actual chip due to over-sizing of transistors. This was a small problem in the beginning of the layout design phase but was solved and is no longer a problem. Due to this, its is no longer a risk. All of the risks that had to do with design have been resolved and are no longer valid.

4.5 Project Evaluation

The main thing the group members have learned are IC design in Cadence and working together in a project. Group communication has worked good. Most of the communication was direct when working together in SoB-lab. It was very good to have our own lab for this course so that we always had a place to work and meet. When not in SoB we used Slack to write to each other. It was not used a lot but good to have it structured.

The most challenging deadline for us was the one on transistor level. We realised quite late that we were missing one important sub-systems, the level shifters and the serial out-block was not fully functional.

The deadline for the high level design was pretty short after the project started, which was a little bit stressful but it was still doable. We thought that it might be a good idea to have more thorough lab examination. The reason for this is because we had one member removed from the group for lack of knowledge. With a more thorough lab examination this could have been detected earlier causing the group less trouble and not wasting the removed member's time in the project letting the removed member focus on other courses.

For the tape-out deadline we were ahead in schedule but problems with pads in DRC and LVS were hard to get rid off and caused some trouble with chip simulations that could have been

avoided. The problems were partly our own fault in form of lacking labels on certain wires but also because the padframe contained errors.

5 Time plan

The time plan is extracted to a separate document, which is updated every week.

References

[1] *A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations* P. M. Kogge and H. S. Stone. IEEE Transactions on Computers, vol. C-22, no. 8, pp. 786-793, Aug. 1973.

[2] *Project Description and Requirement Specification. Project: A 16-bit Kogge-Stone Adder. Version 1.1* Martin Nielsen-Lönn. Document, 2016.

[3] *An Ultralow-Voltage Energy-Efficient Level Shifter* Lanuzza, Marco and Crupi, Felice and De Rose, Raffaele and Strangio, Sebastiano and Rao, Sandro and Iannaccone, Giuseppe. IEEE TRANSACTIONS ON Circuits and Systems II-EXPRESS Briefs, vol. 64, no. 1, pp. 61-65, Jan. 2017.

Appendices

A Schematics

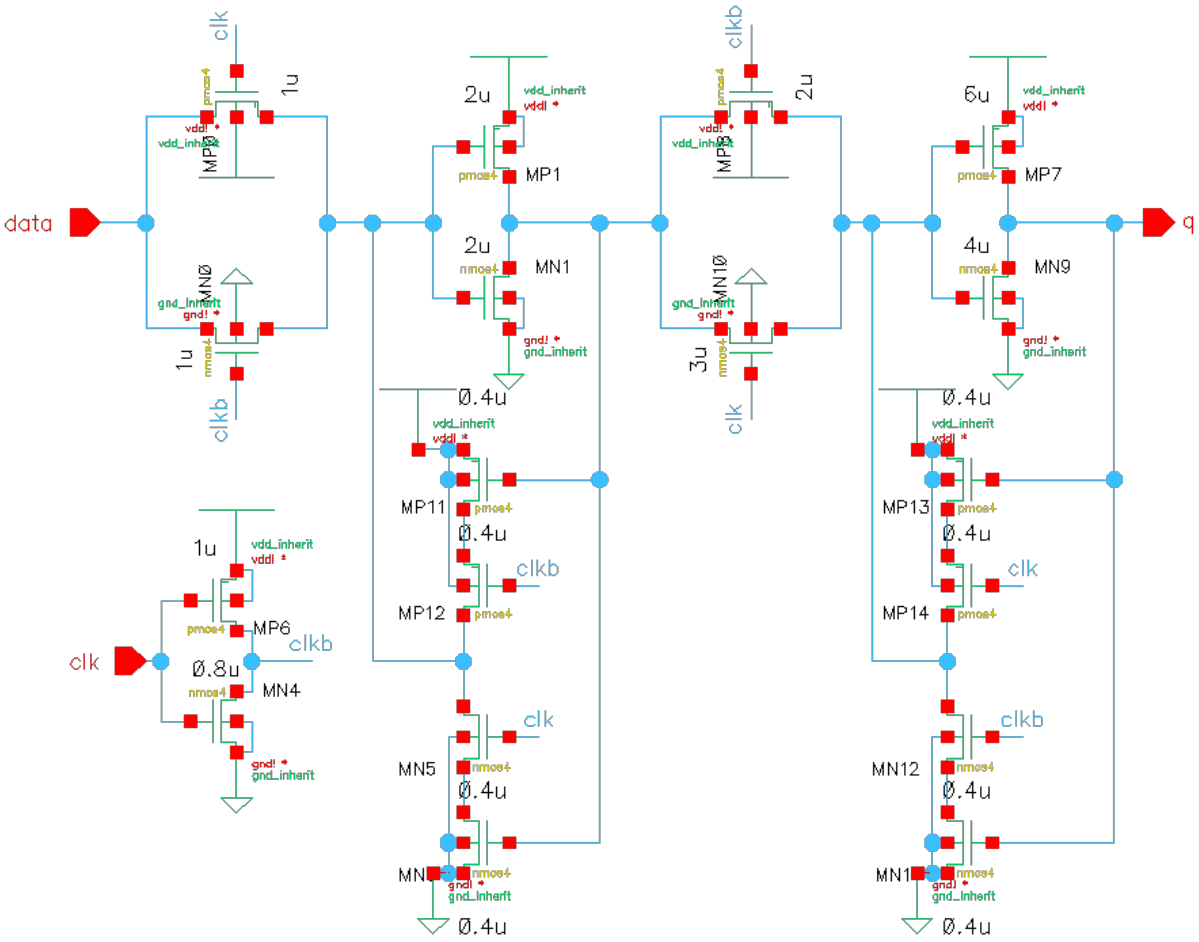

Figure 22: Schematic of the DFF. Transistor widths can be seen in figure. Length is $0.35\,\mu\mathrm{m}$ *for all transistor.*

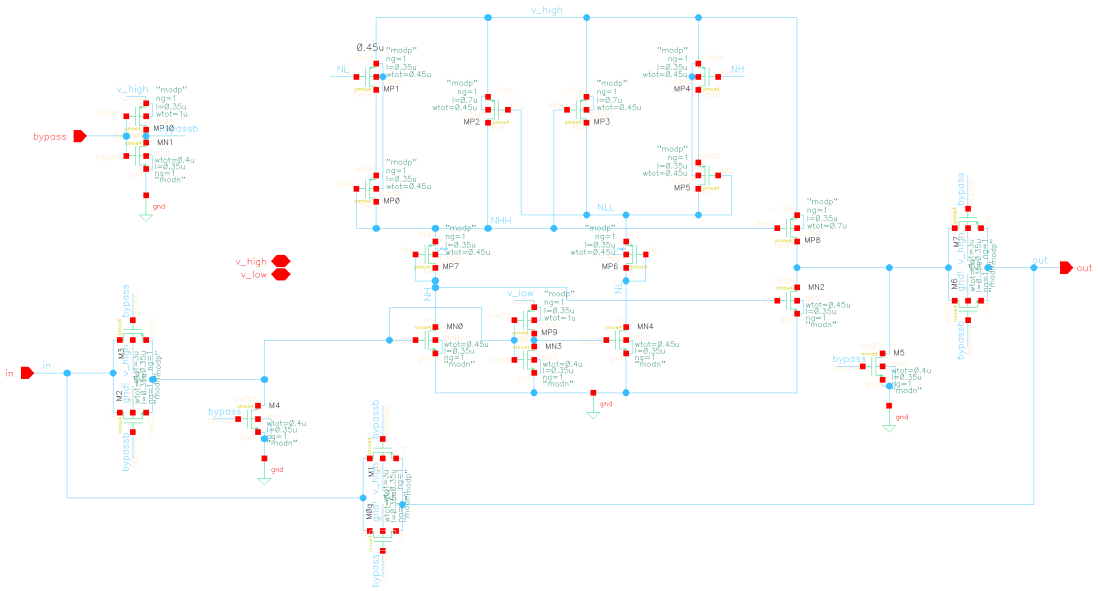

Figure 23: Schematic of the level shifter. Length is $0.35\,\mu\text{m}$ *for all transistors except MP2 and MP3 that is* $0.7\,\mu\text{m}$*. Widths are given in figure.*

B Early test code

```
/*
 * First draft of a testprogram for an arduino
 */

#include <SPI.h>

#define SPI_IN_PIN 3
#define SPI_OUT_PIN 4
#define SPI_ENABLE_PIN 5
#define SPI_CLK_PIN

int A[16];
int B[16];
int Corrsum[16];
int Cin[4];

byte bitSequence[200];
byte byteSequence[25];

SPISettings settingsA(10000, MSBFIRST, SPI_MODE1);


void setup() {
  pinMode(SPI_IN_PIN, INPUT);
  pinMode(SPI_OUT_PIN, OUTPUT);
  pinMode(SPI_ENABLE_PIN, OUTPUT);
  pinMode(SPI_CLK_PIN, OUTPUT);

  Serial.begin(9600);
  SPI.beginTransaction();
}

void loop() {
/* Main function (not finished)
 *
 * Call loading sequence on command from Serial
 *
 * Read the bit sequence on the spi_in_pin
 */


}

/*
 * Start the serial loading to the chip
 */
```

```
void startLoadingSequence() {
  digitalWrite(SPI_ENABLE_PIN, HIGH); //Pull the serial enable high

  for (int i = 0; i < 25; i++) {
    SPI.transfer(byteSequence[i]);
  }

  digitalWrite(SPI_ENABLE_PIN, LOW); //Pull the serial enable low
}


/*
 * Convert bit sequence generated by out generator script to a sequence o
 */
byte[] toByteArray(byte[] bits){
  int bitLength = bits.length();
  int numBytes = bitLength / 8;
    if (bitLength; % 8 != 0) numBytes++;

    byte bytes[numBytes];
    int byteIndex = 0, bitIndex = 0;

    for (int i = 0; i < bitLength; i++) {
        if (bits[i]){
            bytes[byteIndex] |= (byte)(1 << (7 - bitIndex));
        }
        bitIndex++;
        if (bitIndex == 8) {
            bitIndex = 0;
            byteIndex++;
        }
    }

    return bytes;
}
```