NTNU
Norwegian University of
Science and Technology

Low Power Capacitive Touch Sensing

Edgar Leopold Elden

Master of Science in Electronics
Submission date: June 2009
Supervisor: Bjørn B. Larsen, IET
Co-supervisor: Øivind Loe, Energy Micro

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

Capacitive touch sensing is an attractive alternative to mechanical buttons and switches. The method allows electronics devices with user interfaces to be designed without moving parts, reducing cost and improving durability. It also makes more interesting-looking devices, as the iPod with its scroll-wheel possible.
For battery-powered devices, energy consumption is a big concern, so a low-power solution for capacitive touch sensing is essential.
The student will evaluate different schemes of capacitive touch sensing with focus on low power operation, and then implement a solution. The solution should consist of an analog external part, consisting of the capacitor(s) to be sensed and supporting circuitry, and an internal part consisting of digital logic written in verilog or VHDL that is responsible for automatically sensing the capacitor(s) and reporting with digital values whether touch(es) are detected or not.
The focus of the task is low power operation, and a trade-off must most certainly be made between response time and power consumption. The area of the digital logic should also be minimized to make an implementation of the system on a microcontroller plausible. Challenges include management of threshold values and noise rejection
The work can be split into three phases:

- Litterature study
  - o Evaluate existing methods for capacitive touch sensing
  - o Is there room for improvement in the existing methods with respect to power consumption?
  - o Select one or more methods to proceed with based on their potential for low power consumption and implementation size.
- Implementation and simulation
  - o Implement the sensing scheme(s) in verilog/VHDL. The code should be synthesizable.
  - o Simulate the system. Determine power consumption, noise rejection, reliability, etc.
- Implementation on FPGA
  - o Implement the external part of the system on a PCB/veroboard and run the internal part on an FPGA. Report any findings. Did the system behave as expected?

Assignment given: 15. January 2009
Supervisor: Bjørn B. Larsen, IET

Preface

This report is the result of the work associated with my Master's thesis at the Norwegian University of Science and Technology during the spring of 2009. I have specialized in Design of Digital Systems at the Master of Science program in Electronics Engineering at the Department of Electronics and Telecommunications.

This thesis describes the process of designing a capacitive touch system with focus on low power. The assignment has been given by Enercy Micro AS. During the work it has been a priority to identify how different design choices affect power, response time and noise rejection. This priority will hopefully make this thesis a good insight into how a capacitive touch system can be constructed to minimize power while performing acceptable to human interacting. A fully working system has been constructed and tested to see if simulations and theory coincide with the behaviour of the final product. It has been rewarding to see the system grow and witness how the design choices has had a positive affect on the power usage of the system.

I would like to thank my supervisor Associate Professor Bjørn B. Larsen for guiding me while writing this report. I would also like to thank my supervisor at Energy Micro AS, Øivind Loe, who has helped me with many technical questions.

Trondheim, June 2009

Edgar L. Elden

Abstract

This thesis will seek to design a capacitive touch sensor that uses as little power as possible while still having decent performance. The study will start by discussing oscillators and find that relaxation oscillators with a frequency dependent on an RC-circuit is of greatest interest. Thorough simulations and theory will show that it is power efficient for the RC-circuit to oscillate between two voltage levels close to the supply voltage. It will also show that it is only the resistance that affect the power dissipation in the RC-circuit. A Finite State Machine that monitors changes in the period of the oscillator is described and designed. It uses two IIR filters to reject noise from the oscillator and provide an average over time the input can be compared to. A prototype is built and tests establish that both the oscillator and FSM behave as expected. It is found that the response time of the FSM can be stated in sampling periods and that lower bit lengths give faster response time. Power estimations are done and it is found that the FSM uses two orders of magnitude less power than the oscillator. The full design is compared to a low power capacitive touch system currently on the market. Power estimations indicate that the design proposed uses an order of magnitude less power than the commercial implementation it is compared with. The results also indicate that the proposed design has a potential for even more power optimization.

Contents

List of Figures

List of Tables

Introduction

In recent years there have been an increased use of capacitive touch sensing in consumer products. These interfaces have some advantages over typical buttons as they contain no mechanical parts. This gives higher durability and enables sleeker user interfaces where buttons can be embedded in the product itself. An example is the buttons on new stoves that are embedded in the glass-ceramic layer of the cooktop. Embedding the button in the product like this is also of great advantage in environments that needs to be keept clean, e.g kitchens or medical equipment.

This thesis will seek to design a capacitive touch-system that is as power efficient as possible while keeping an acceptable level of performance. It will look at implementations that are in use or have been proposed before, and seek to optimize or tweak them for optimal power efficiency. This focus will most likely come at the expense of other characterisitcs of a specific design, such as noise rejection, reliability or especially responsetime. The responsetime of the system should be acceptable for a human operator.

Different implementations of a capacitive touch system will be discussed and one implemented for its capacity of improved power efficiency. This will be done either by modifying the design or make it operate in a special manner.

The main contributions from this thesis are:

- Chosen an oscillatordesign for use in the sensing scheme and chosen components for it.
- Written a program in C that simulates the response of the RC-circuit in such an oscillator.
- Designed hardware that monitors the oscillator and reports sudden changes in frequency.

- Designed a prototype of the oscillator on PCB and tested whether oscillators with different characteristics function according to theory.
- Implemented the digital hardware on an FPGA from Altera together with a Nios II-based testbench that monitors the funcionality of the hardware.

Chapter 1

Theory and selection of oscillator

This chapter will discuss how capacitive touch sensing works and then look at a few types of oscillators and how they operate. A choice will be made on what oscillator to use in the final design. Finally the different parts of the oscillator, how they operate and how their power consumption can be minimized will be discussed from a theoretical standpoint.

Capacitive touch sensing works by detecting the induced capacitance in a node when a finger touches that node. Most often this node is a pad of copper on a PCB with some material with an electrical permitivity on top that the finger touches. Other design exist though, e.g. lamps that use their whole exterior as a touch button thereby allowing operation by touching the lamp. In Figure 1.1 on the following page a regular touch button can be seen. Here the capacitance in the button is C_p when there is no finger close to the button. As soon as a finger is close enough a capacitance, C_f, between the pad and ground is induced through the finger. This capacitance comes in parallell with C_p resulting in a total capacitance in the node $C_{tot} = C_p + C_f$. This change in capacitance needs to be detected by some system.

The capacitance between two conductive materials is given in Equation 1.1. As can be seen the capacitance, C, is given as a relationship between the area of the materials, A, the relative permeability of the material between the conducting materials, ε_r, the permeability of free space, ε_0 and the distance between the conductive materials, d.

$$C = \frac{\varepsilon_0 \varepsilon_r A}{d} \tag{1.1}$$

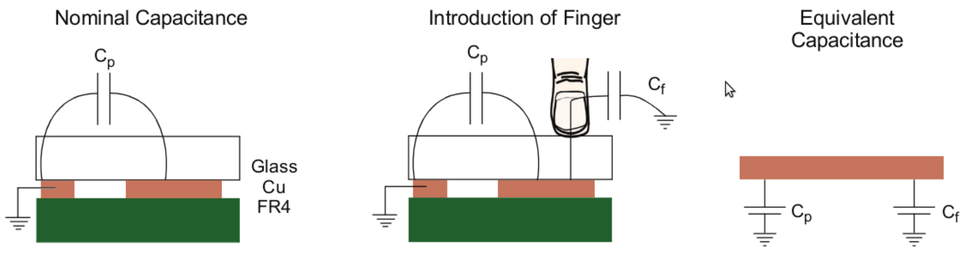


Figure 1.1: Capacitive touch sensing illustration.[13]

1.1 Oscillator

The oscillators purpose in the circuit is to generate a frequency that will change when the button area is touched by a finger. In practice this means that the frequency of the oscillator will have to be largely dependent on a capacitor. Different types of oscillators are abundant, and we will have to decide upon one to discuss further in this project. In the preliminary work for this thesis there has been found several potential oscillators that have been considered. There are some important demands that the oscillators will have to fulfill to be usable in this project. They have to be as small as possible, the frequency must be highly dependent on a capacitance and they have to have some potential for optimizing energy consumption.

Below a representative few of the oscillators found during the preliminary work are presented and discussed.

1.1.1 Harmonic Oscillators

Harmonic oscillators generate sine waves at a frequency determined by the magnitude of either resistors, capacitors, inductors or a combination of these. The basic premise of their function is an amplifier with an electronic filter connected between its output and input. At startupt the amplifier contains mostly noise, but the signal gets filtered and fed back into the amplifier continuously reducing the noise until only frequencies that the filter lets trough is left.

The Wien Bridge oscillator is one of several types of harmonic oscillators. Its design can be observed in Figure 1.2 and i has a frequency given in Formula

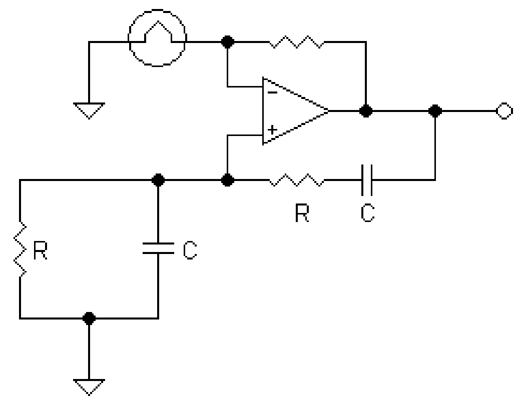


Figure 1.2: Wien Bridge Oscillator.[15]

1.2.

$$f = \frac{1}{2\pi RC} \tag{1.2}$$

This oscillator has several drawbacks. First, the frequency is determined by two identical capacitors. Preferably the frequency should be dependent on only one capacitor since changing one of several capacitors may have undesired effects other than changing the frequency. In this specific oscillator the two capacitors are listed as being of equal size. Simulations will have to be done to determine the effect of changing only one of the capacitors if this oscillator is chosen for further use in this project. Dependence on some sort of amplitude stabilization as shown in the upper left corner of Figure 1.2 may also prove to be a challenge. This complicates the design process beyond the scope of the problem description.

Other harmonic oscillators are disqualified because they contain inductors. These are too large to be practically implementet in a microcontroller, as this projects seeks to achieve. They also store all their energy in a magnetic field. This field can induce currents in other parts of the oscillator or the unit the capacitive touch sensing scheme will be part of due to mutual inductance[9]. One final unwanted aspect of harmonic oscillators is their sine output. In this design a square output is wanted as the frequency produces will be used as the clock signal for a digital counter.

All the harmonic oscillators found in the preliminary work has had one or more of the disadvantages discussed above and other solutions should be discussed in an effort to find a better alternative.

1.1.2 Relaxation oscillator

Relaxation oscillators are oscillators that depend on the charging and discharging of a single energy storage element. They rely on the nonlinear characteristic of the storage element, in our case a capacitor, and are typically easy to implement as a single module in an integrated circuit[1]. Also according to [1] they are prone to large random fluctuations in their period so thorough simulations of a design must be done before a prototype is built. The protoype should also be heavily tested to know what level of noise will most likely be present in a final design.

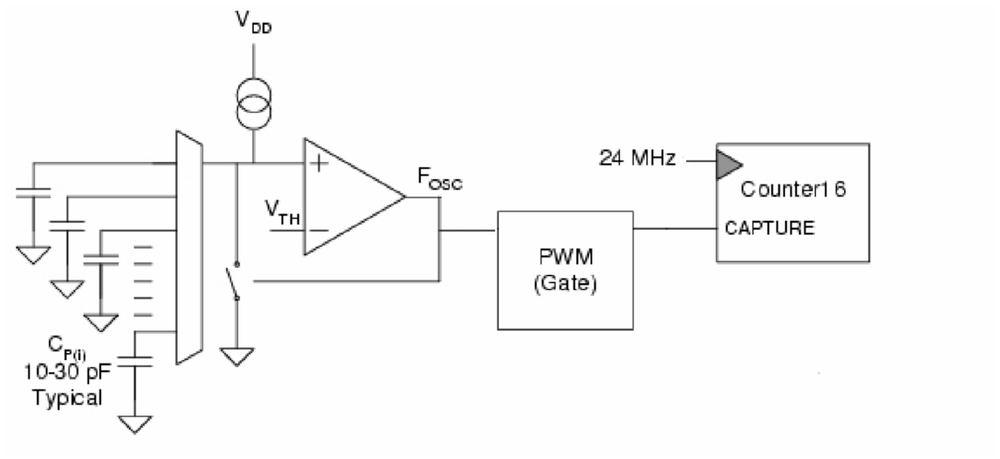


Figure 1.3: Relaxation oscillator design proposed by Planet Analog.[6]

In Figure 1.3 a design proposed by Planet Analog for a relaxation oscillator is shown on the left[6]. The digital logic and PWM on the right side are not part of the oscillator and not interesting in this section. This design functions by charging a capacitor until it reaches a specified voltagelevel, V_{TH}, at which point the comparator activates a transistor that quickly discharges the capacitor to ground. In the pictured design there are several capacitors (buttons) connected to the comparator through a mux. When the button is touched, another capacitance is inferred in the node leading to longer charge time. This will decrease the frequency of the oscillator.

This comparator is small and easy to implement, but it has a couple of major drawbacks though. Since the charge in the capacitor is dumped directly to ground the discharge is nearly instantaneous. The capacitor is then charging and drawing power nearly all the time, which is inefficient. A better solution would be one where the capacitor is slowly discharged, spending time discharging the energy it used while charging. The reason why this is more efficient is the fact that the RC-circuit does not consume energy during discharge, the energy used to drive the current through the resistor is already stored in the capacitor. Another drawback is that while the capacitor is discharging there is a short from V_{DD} to ground leading to an even higher powerdrain.

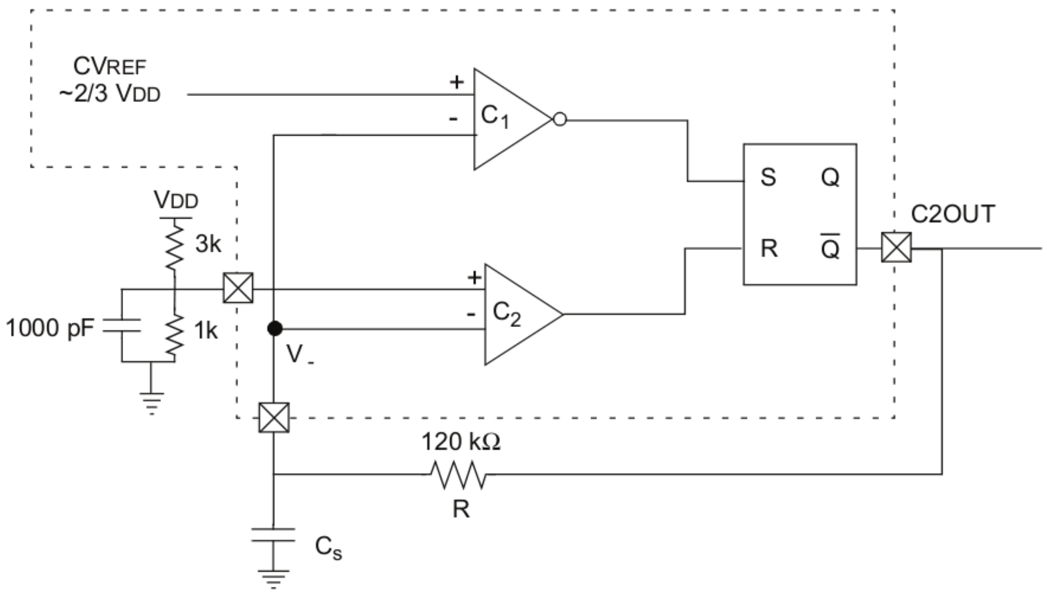


Figure 1.4: Relaxation oscillator design proposed by Microchip.[13]

The design proposed by Microchip in Figure 1.4 does not have the same problems as the design by proposed by Planet Analog, yet it functions on much the same way. In this design the voltagelevels that the capacitor oscillates between can be independently set at the inputs of the comparators. This gives more control over the power usage of the RC-part of the oscillator as will be shown in Section 3.2. As in Planet Analog's design the capacitor, C_s, is charged up until it reaches the voltage level at the positive input of comparator C_1. At this time C_1 sets it output low which sets the SR-latch's inverted output to zero. The RC-circuit then discharges into the output of the SR-latch until the voltage across the capacitor is as low as the positive input on C_2. This gives a positive output from C_2 which resets the inverted output of the latch to one restarting the cycle.

Since the RC-circuit is charged and discharged through the ouput of the SR-latch it is important that the SR-latch in use can sustain the current levels the RC-circuit demands or delivers.

Microchip's solution does not have the same shortcomings as Planet Analog's design. There are no short circuits during its operation, neither from V_{DD} or V_{-} to ground. When the capacitor is discharging the oscillator only consumes quiescent current in the comparator and latch. Thus the oscillator will not continuously draw power but spend some fraction of its time discharging. The solution from Planet Analog will continuously charge its capacitor as the discharge is done instantaneously.

1.1.3 Conclusion and choice

Due to the need for a square wave oscillator and the problems discussed in the last part this report will not use an harmonic oscillator. The choice has been settled on the design from Microchip in Figure 1.4 on the preceding page. This oscillator is somewhat complex compared to the others discussed in this chapter, but as will be shown in Chapter 3 the majority of power consumption in the oscillator comes from the RC-circuit. The other parts contribution is negligible compared to this so the added complexity is of no great concern. The fact that the comparator levels can be individually set will also prove to be a great asset in minimizing the energy consumption in the RC-circuit later in the report.

1.2 RC-filter

The RC-circuit is a vital part of the oscillator, and this section will look at how it reacts to input. A series RC-circuit acts as a low-pass filter with the voltage across the capacitor as the output and input at the opposite side of the resistor. A low-pass filter will block out any high frequencies in a signal, and in practice hinder any sudden changes in voltage at the output node. Figure 1.5 on the next page contains a schematic view of the circuit.

The definition of capacitance is the charge between two nodes divided by the voltage across them, as shown in Equation 1.3 on the facing page. By deriving the charge, Q, into the current needed to generate such a charge we get Equation 1.4 on the next page which gives the voltage across a capacitor

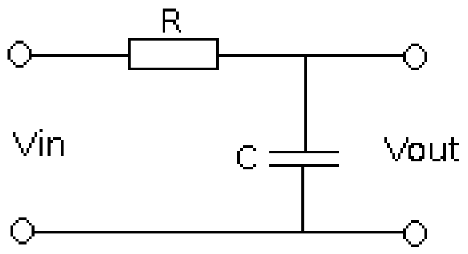


Figure 1.5: RC-circuit with input at the left hand side and output at the right.

as a function of time. As can be seen in Equation 1.4 the voltage across a capacitor is then given as the integral of the current into it divided by its capacitance. In the RC-circuit the current into the capacitor is given by the the voltage across the resistor. The formula is given in Equation 1.5.

$$C = \frac{Q}{V} \tag{1.3}$$

$$v_C(t) = \frac{1}{C}\int_{t_0}^{t} i_C(\tau)d\tau + v_C(t_0) \tag{1.4}$$

$$i_C = \frac{v_{in}(t) - v_C(t)}{R} \tag{1.5}$$

These functons are the basis for deriving V_{out} as a function of V_{in}. The result is given in Equation 1.6 for a sudden change from $0V$ to V_{in} and in 1.7 for a sudden fall from V_{in} to $0V$, with τ given in Equation 1.8. In the left part of Figure 1.6 on the following page the resulting output of the RC-circuit with a sudden rise in the input is shown. Similarly the response to a sudden drop on the input is shown at the right.

$$V_{out} = V_{in}(1 - e^{\frac{-t}{\tau}}) \tag{1.6}$$

$$V_{out} = V_{in}e^{\frac{-t}{\tau}} \tag{1.7}$$

$$\tau = RC \tag{1.8}$$

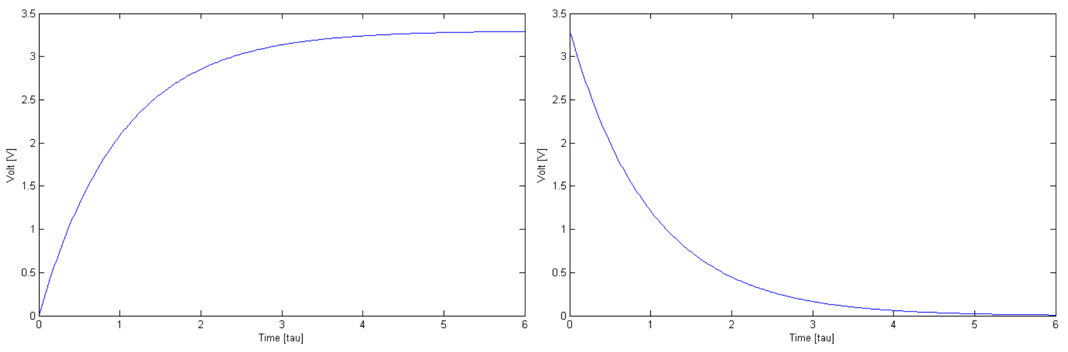


Figure 1.6: To the left the voltage across the capacitor in an RC-circuit is shown with a sudden increase from $0V$ to $3.3V$ at the input. To the right the same circuit experiences a sudden fall from $3.3V$ to $0V$.

In Figure 1.6 the response of an RC-circuit is shown with the time scale given in τ's. As can be noted the response of the circuit is dependent on τ. In fact, τ is the time needed for the capacitor to fall to $1/e$ of its initial value as can be easily derived from Equation 1.7 if t is substituted with τ. As the response of the RC-circuit is decided by τ it becomes apparent from Equation 1.8 that the frequency in the oscillator will be dependent on R and C.

In Figure 1.7 on the next page the response of two RC-circuits with the same τ is shown. The power dissipated in the resistor is given as a red line in Equation 1.9 on the facing page and is shown in the figure together with the output from the RC-circuit (blue line). The figure on the left has a capacitor half as large as the one to the right. Similarily, the RC-circuit on the right has a resistor with half the resistance to the one on the left. As can be seen, the response of the two circuits is exactly the same, due to the equal τ. The important thing to observe in the figure is how the amount of power being dissipated in the resistor is halved in the one on the left compared to the one with a larger capacitor. As the oscillator's frequency will be determined by the response time of the RC-circuit this figure shows that the power usage of the oscillator is not dependent on the frequency in a one-to-one fashion. Power consumption can then be optimized for a specific frequency.

Another thing to note from Figure 1.7 on the next page is that the power dissipation depends on the time. From Equation 1.9 and 1.10 it can be seen that the voltage across the resistor is dependent on the voltage across the capacitor, therefore the power dissipation in the resistor is also time dependent. This can also be seen in Figure 1.7 on the facing page where the power dissipation falls exponentially with time. In the oscillator design proposed above the comparator levels can be independently set. Thus it is possible to set these levels to values that minimizes the power dissipation in the resistor. How the comparator levels and RC-circuit affect the power usage will be shown in simulations in Section 3.2.

$$p(t) = i_R(t)v_R(t) = \frac{v_R(t)^2}{R} \tag{1.9}$$

where

$$v_R(t) = V_{in} - v_C(t) \tag{1.10}$$

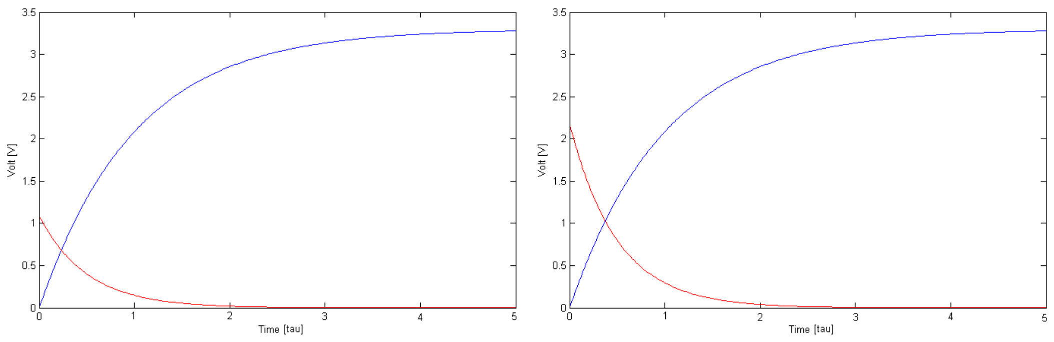


Figure 1.7: Voltagelevel over a capacitor during charging of an RC-circuit (blue). And the power dissipated in the resistor during the same time (red). On the left $R = 10\Omega$ and $C = 0.5F$. On the right $R = 5\Omega$ and $C = 1F$.

1.3 Comparators

This report will not discuss how comparators operate but focus on their functionality and characteristics. A comparator is an electronic unit with two inputs that outputs a logic value depending on which of the inputs has the highest voltage level connected to it. In Figure 1.8 on the next page a schematic view of a basic comparator can be seen. The output of the comparator will be V_{DD} as long as the input at node $+$ is larger than the input at node $-$. As soon as the input at node $-$ is larger than the input at $+$ the output switches to GND. Thus the output is dependent on whether the difference between the $+$ and $-$ is negative or positive.

An aspect of comparators that needs to be considered is whether they support rail-to-rail operation or not. If a comparator is said to have rail-to-rail operation it is supposed to cope with inputs that vary between V_{DD} and

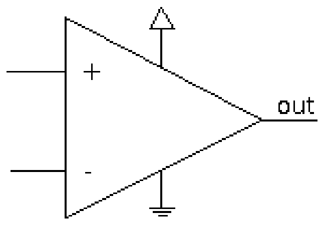


Figure 1.8: Schematic view of a basic comparator.

GND. If it does not support rail-to-rail it may be specified to work only if the inputs are smaller/larger than some specific value compared to V_{DD} or GND. E.g. the comparator may only guarantee correct operation if the inputs are at least $1V$ smaller than V_{DD}. If the comparator does not support rail-to-rail this will put restrictions on which comparator levels can be used in the oscillator.

A final thing to note about comparators is that the output does not switch momentarily when the inputs indicate change of the output. This small delay will often be stated in the datasheet and will set an upper limit for the frequency.

Chapter 2

Theory of the digital counter

The need for a digital circuit that can monitor the frequency of the oscillator will be discussed in this chapter together with ways to improve power usage of such a circuit. A basic premise of its operation and that of its main components will also be briefly discussed. As with the rest of the design the main goal will be to design the circuit to be as power efficient as possible. The problem description states that this will most likely come at the expense of noise rejection, response time and reliability and this will have to come into consideration. This chapter will discuss how the counter should work, what characteristics it can or must have and theoretical implications of these choices on the behaviour of the circuit.

For the counter to operate with as few problems as possible it is important to design it with noise on the input in mind. As was discussed in Chapter 1.1.2 a relaxation oscillator has problems with large variations in its period. Variations like these will lead to fluctuations in the frequency which will give variations in the value from the counter. Since a button push will contain other frequencies than noise some sort of filtering can be applied to the signal to seperate real from false button pushes. As with all else in this project the design will have to be made with a focus on power consumption.

The counter must be able to count the number of periods of the oscillator during a fixed time period and compare it with some known value. This way it will be able to determine if the frequency has decreased due to a finger touching the button pad. What value the counter compares with can be decided in several ways. One solution is to have a static number that it is compared to for every sample. This solution demands little hardware and will have low energy consumption, but this static value has to be preprogrammed in the circuit and therefore have to be measured accurately before construction. In mass production this would require a high level of consistency between each item or testing and individual programming of each. Another problem is the lack of dynamics in the system. The capacitance may change over time due to athmospheric conditions, temperature and regular wear and tear. This may cause the counter to report erronous button pushes or fail to sense real touches. If the unit is resistant to change by the environment, and the variance in the units produced in mass production is low, such a static comparison could be usable. The unit would be small and easy-to-implement and most likely use little energy. In Chapter 4 this thesis will look at the usability of such an implementation.

If the level of noise is quite large and the frequency of the oscillator varies over time an approach that dynamically responds to these changes will have to be made. The unit could over time monitor the average number of periods each sample and alter the comparison value to increase responsiveness and decrease noise. This way it will dynamically respond to changes in the oscillator that are not related to fingerpresses. An easy way to calculate an average over time is to use a low-pass filter on the input. The filter would hold a value that would not change with the high frequencies of noise, but would over time react to the low frequency changes of the environment or wear-and-tear. What implementation of a low-pass filter to use will be discussed and simulated in Chapter 4.

2.1 Metastability

Metastability is a problem that can occur when the data on the input of a register is not synchronized with the clock of the register. If the input changes during the setup or hold time of the register the output from the register may experience a pulse or may even oscillate[16]. A problem like this can lead to unwanted states in the system and problems ranging from erronous button pushes to hangups in the system. The hardware in this thesis will have two digital logic blocks with differenct clock signal that will communicate. Therefore this problem must be taken care of in the final design.

Metastability can be avoided by putting a chain of two or more registers between the digital circuits that are not synchronous. This improves the resistivity to metastability as the chance of metastability affecting the recieving circuit through several registers and clock periods are substantially smaller than through a direct connection.

2.2 Dynamic Power

This section will discuss the power usage in a digital circuit, and look at ways to minimize the different factors that contribute to the power usage.

Dynamic power is the power dissipation that occurs in the circuit when the logic state of the system changes. The power usage comes from the charging of parasitic capacitances in the circuit when the input to the circuit changes[11].

$$W = \frac{1}{2}CV^2 f\alpha \tag{2.1}$$

From Equation 2.1 the average dynamic power usage of a gate can be calculated[11]. The power usage depends on four factors that is explained below.

- C is the capacitance in the gate. It is affected by the gate size and trace length.
- V is the supply voltage of the system.
- f is the frequency of the system.
- α is the switching activity in the gate. I.e. a number for how often the gate switches. A value of 1 would indicate the gate swithces every clock cycle.

2.2.1 Capacitance and Voltage

The powerusage in a node is highly dependent on the voltagesupply, as can be noted in Equation 2.1 where the voltage is squared. The library used in the power estimations has defined the voltage supply to be 1.62V. Even though lowering the voltage supply is not a solution available to this project it is worth noting as the most potent way to lower the powerusage in the system.

The capacitance in the node is also something that is hard to affect from the point of view of a system designer. This is more a characteristic of the physical system.

2.2.2 Frequency

The frequency of the digital circuitry determines how often the system updates its registers and is then obviously something that influences the dynamic power usage in the system. After the system has been designed the lowest working frequency should be found. A low frequency is just one simple way to affect how the frequency influences the powerusage in the system. In the following sections a few other approaches to minimize the frequencys impact on powerusage will be discussed.

Clock gating is the process of only supplying a clock to a part of the system when that system needs to update its registers. This will decrease the powerusage in the registers, clock tree and gates in the fanout of the registers[11]. In Figure 2.1 the process can be seen with and without clock gating. The register on the left will only update its state if the signal *control* is set high. The register on the right will be updated every clock period consuming energy even when there is no change on the input. Clock gating will have to be implemented manually on every part of the system where such an approach may decrease the switching activity.

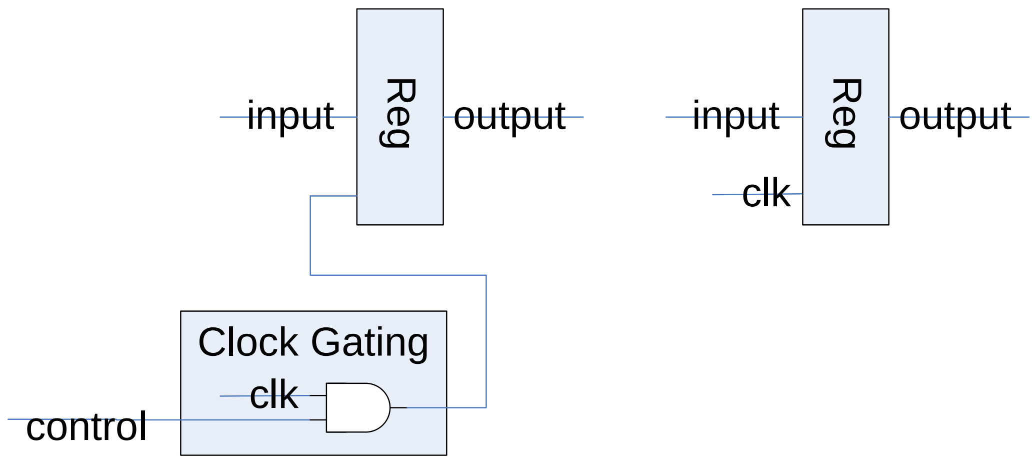


Figure 2.1: Clock gating of a register. The register on the left will only update its state if the signal *control* is high.

Another way to decrease the power usage in the system is to dynamically change the frequency in the system. The goal here is to give the system a frequency that is just high enough to make the system reach its deadlines. The digital part of this system is supposed to poll the frequency of the oscillator at regular intervals and calculate changes in frequency. Since most of the time is spent waiting for the next time to sample, the frequency could be lowered during this waiting period. While dynamically altering the frequency of the system is a very effective way to minimize the powerusage in a system it is not necessarily very applicable in this design though. The system in this thesis does not have specific deadlines, and is only supposed to do a few calculations everytime it samples the oscillator. The parts that samples the system can just as easily be clock gated eliminating dynamic power usage all together.

2.2.3 Switching activity

The switching activity is a measure on how often a specific gate switches. The best way to decrease this measure is algorithmic optimizations. In essence doing a task the most efficient way.

2.3 Leakage Power

While lower supply voltage decreases the dynamic power usage in a circuit it increases the delay in gates in the system. To compensate for this the threshold value of the transistor can be lowered too. This has the negative effect that the leakage current through the transistor increases. The dominant leakage current in transistors in current CMOS technologies is the leakage current through the channel of a sub-threshold transistor that is off [11]. In fact, lowering the threshold voltage of a transistor $100mV$ increases the leakage current tenfold[5]. To further complicate the problem the leakage current increases with temperature. This is a problem that has to be given attention when designing digital circuits.

Three approaches to lowering the leakage current is discussed in [11]. Two of them, power-gating and body bias control are out of the scope of this project as they require control over the hardware being used and this project will be implemented on an FPGA for testing. Another approach that may be useful is applying a minimum leakage vector to parts of the system while they are idle. An MLV is a vector that minimzes the leakage current by applying input to a circuit that minimizes the current through the transistors. Applying MLV will require hardware that uses both area and power and requires a great deal of simulations of the circuit to find such a vector.

Chapter 3

Design and simulations of oscillator

This chapter will discuss the oscillator chosen in Chapter 1 and seek to find optimal components and values for the RC-circuit and comparator levels.

3.1 Comparators

This section will look at the comparators and how they affect the oscillator. It will seek to find a comparator that is as energy efficient as possible, while still being able to operate at the frequencies required for capacitive touch sensing. In Table 3.1 a couple of potential comparators found at Farnell.com are presented.

Model	Quiescent Curr.	Response Time	Rail-to-rail	Ref
Maxim MAX921-924	4μA	$14\mu s/5\mu s$	No	[7]
National S. LMC7211	7μA	$11\mu s/4\mu s$	Yes	[8]
TI TLC193	22μA	$1.75\mu s/0.9\mu s$	Yes	[12]

Table 3.1: Comparison of three comparators found on Farnell.com. All are advertised as designed for low power applications. Responsetime is for high-to-low output with 10mV and 100mV overdrive. Second overdrive of comparator from TI is for 40mV overdrive.

In Table 3.1 the comparator from Texas Instruments can easily be excluded due to its high energy consumption compared to the others. Choosing between the two others are not that apparent though. The comparator from Maxim does use roughly 43% less power than the one from National Semiconductor, but it has a responsetime that is 27% longer. It is worth noting that the comparator from Maxim does not support rail-to-rail, it accepts inputs that swing from the negative supply rail to within $1.3V$ of the positive supply. What tips the scale in favor of Maxim in this project is the availability of a Spice-model. With a Spice-model it is possible to accurately simulate the behaviour of the oscillator. It also allows simulations of the comparator alone to learn more of its behaviour before a prototype is built. The prototype will use Maxim's comparator and all simulations in this documents will be based on the Spice-model from Maxim.

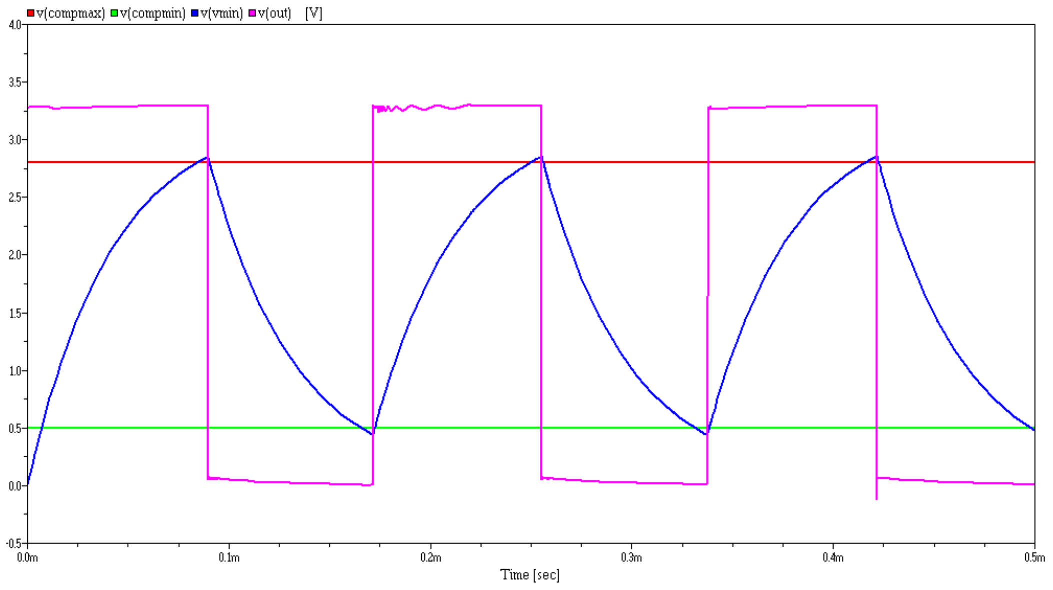


Figure 3.1: The response of the oscillator. $RC = 45.2\mu$s is 10-100 times as high as it will be in the final circuit, but is set unnaturally high in this simulation to make the figure is easier to comprehend.

In Figure 3.3 on page 22 the response and current consumption of the comparator is shown using the Spice-model provided by Maxim. As voltage at the negative input (green) increases beyond the voltage at the positive input (black) the output switches from high to low (blue). The red line shows the current that the comparator drains from the powersupply. The simulated current of nearly 1mA with high output and 0.5mA while low is far from the promised quiescent current of 4μA. This has been confirmed by Maxim Support as a fault in their Spice-model. The Spice-model is capable of modeling the behaviour of the comparator, but does not model the power usage correctly.

For that reason this thesis will assume the specified current of 4μA as correct and use the model to simulate delay and behaviour but not power.

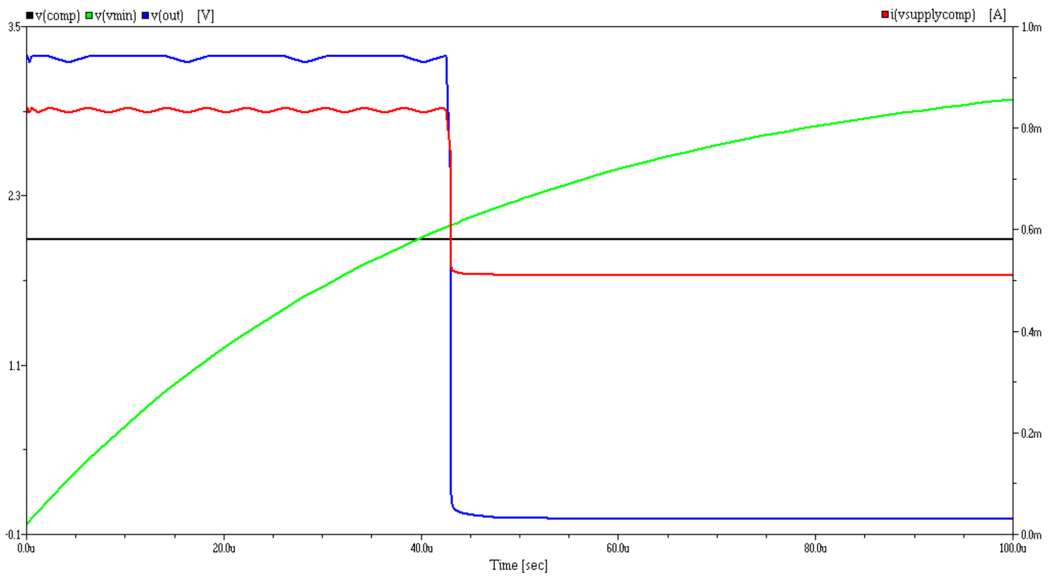


Figure 3.2: The response of and current drawn by the MAX921.

The final behaviour of the oscillator will also be determined by the latency of the comparator. As discussed above, the chosen comparator has a reported latency of 14μs. According to the datasheet this is the typical response time for a comparator in 3V operation and 10mV overdrive. Overdrive is the voltage difference between the two inputs of the comparator. For 100mV overdrive the latency decreases to 5μs.

The simulated results in Figure 3.3 on the next page shows some discrepancy between the the data from the datasheet and the data obtained from simulations on the spicemodel provided by Maxim. Some of this can be attributed to the increased V_{DD} in the simulated results, as this increases the response time[7]. Even though the datasheet gives these values as *typical*, the values may be overly pessimistic. As the Spice-model is presented as an exact representation of the comparator it is likely that the response time is lower than what the data sheet states. In Table 3.2 on page 24 values according to both the datasheet and simulations are presented for comparison. Note that there is noe difference between 1000mV and 100mV overdrive according to the Spice-model.

In the final oscillator the input will grow exponentially, and will have an overdrive that is lower than 100mV for a very short time. This can be seen in

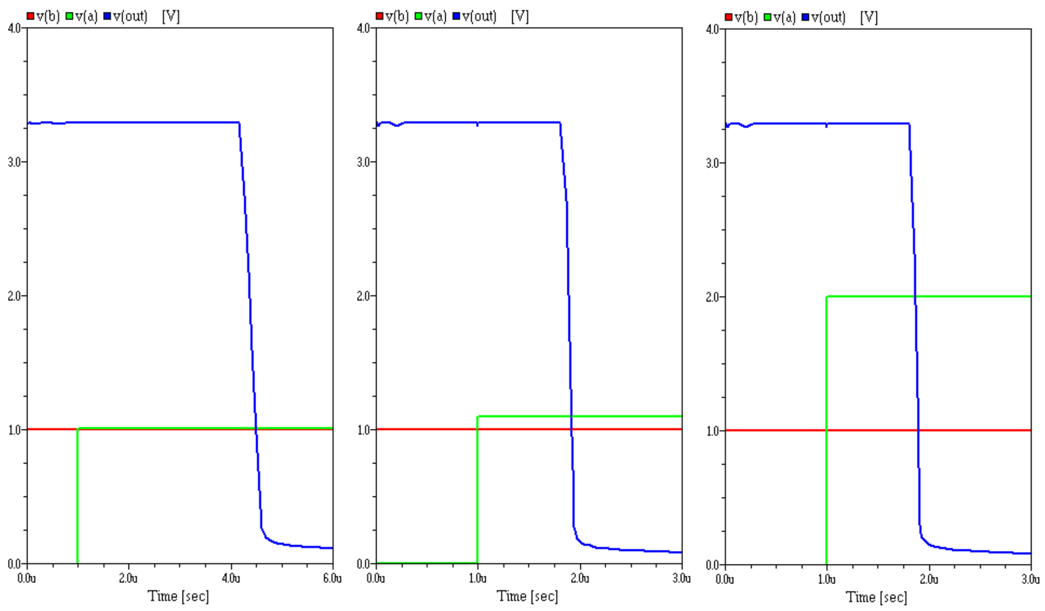


Figure 3.3: The response of the Maxim MAX921 comparator with 10mV, 100 mV and 1000mV overdrive. Note the difference in responsetime with different overdrives.

Figure 3.4 on the next page where the overdrive is over 100mV after roughly 0.5μs. As long as the oscillator reaches an overdrive in such a short time, it can be assumed that the responsetime will be closer to the results for 100mV than 10mV. From Figure 3.4 it can also be observed how the voltage across the capacitor does not vary between the comparator levels at high frequencies. Instead the latency in the comparator makes the RC-circuit oscillate between values lower than *compmin* and higher than *compmax*. To get a more correct estimation of the power usage in a specific oscillator it will therefore be important to measure the real voltages it oscillates between.

These simulations are useful for providing a pointer to what will most likely be a maximum frequency from the oscillator. As shown in Figure 3.1 on page 20 compared to Figure 3.4 on the next page the impact of the comparator on the frequency is dependent on the RC-part of the circuit. But based on the theoretical values from the datasheet it's impossible to achieve a frequency above 100kHZ assuming 100mV overdrive. This is assuming there is no time used to charging/discharging the RC-circuit. If we depend more on the results from the simulations we will be able to achieve frequencies in the region of 0.5Mhz. If the comparators response time fast enough for such frequencies we will have greater freedom in designing the RC-circuit to give a frequency

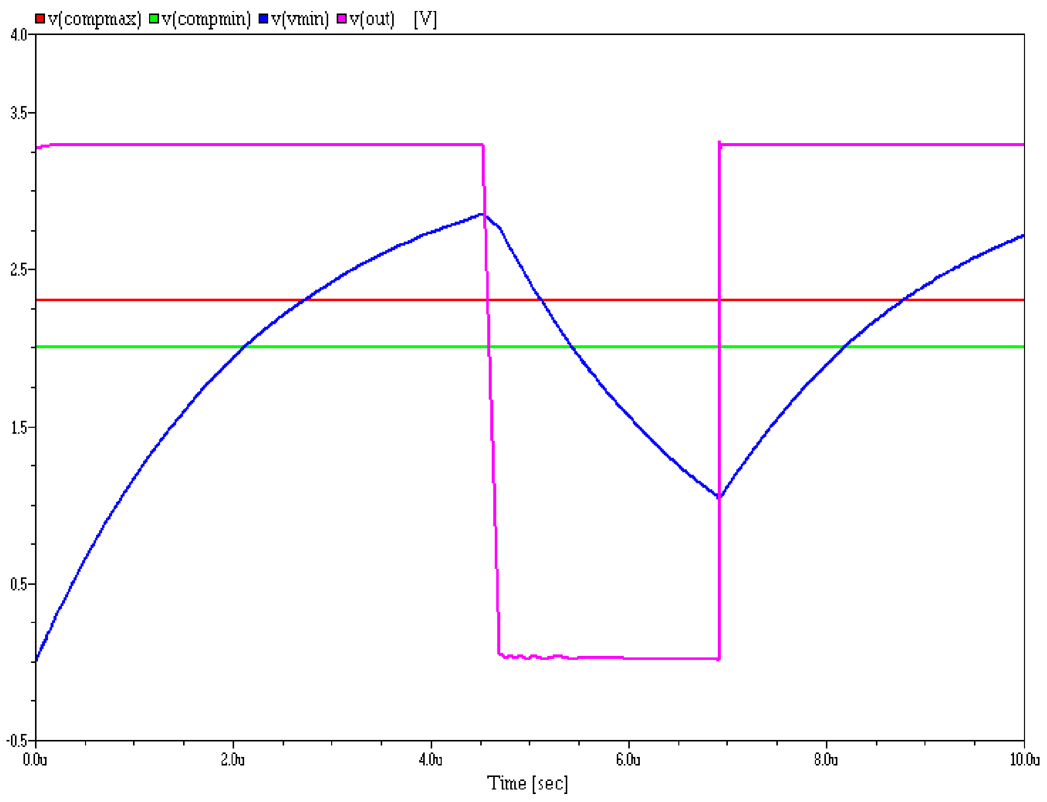


Figure 3.4: The response of the oscillator with C = 15pF, C = 150kHz, $compmin = 2$V and $compmax = 2.3$V.

that fits our needs for both response time and power usage. This report assumes that the spice-model of the comparator designed by Maxim is correct an that the maximum frequency is well beyond 100kHZ, if not as large as nearly 0.6MHz as indicated by the simulations.

	Theoretical 3V		Simulated 3.3V	
Overdrive	Responstime	Frequency	Responsetime	Frequency
10mV	14μs	35.7kHz	3.5μs	143kHz
100mV	5μs	100kHz	875ns	571.5kHz
1000mV	N/A	N/A	875ns	571.5kHz

Table 3.2: Theoretical and simulated response times and maximum frequencies for the comparators with different levels of overdrive.

3.2 RC-circuit

The RC-circuit is the part of the oscillator that will decide the frequency and it is important to simulate this part thoroughly to obtain data on how it affects power usage. How long it takes to charge the capacitor up to a certain voltage level is dependent on the size of the capacitor and the current going into it. The current into the capacitor is dependent on the resistance between the capacitor and the output of the SR-latch. In Chapter 1.2 the charging and discharging of an RC-circuit is explained thorougly and the relation between the RC-circuit, frequency and powerusage is discussed.

In Chapter 1.2 it was found that for a given τ the power usage is reduced inversely proportional to the size of the resistor. Hence the size of the resistor should be maximised when a τ that gives a wanted frequency is found. A smaller capacitance may lead to unwanted situations though. Since the voltage across the capacitor is dependent on the current into it as given in Equation 1.4 on page 9 noise in the system will affect the response of the RC-circuit more when the capacitance is small[2]. The size of the capacitor is also limited to the size of availiable capacitors and the capcitance in the button itself. A small capacitance is wanted as the capacitance inferred in the circuit with a button push will be relatively larger and thus easier to detect.

To simulate the characteristics of an RC-circuit with specific voltage supply, resistance, capacitance and comparator levels a computer program has been written in C. The program can be reviewed in Appendix A. This program simulates the RC-circuit during one cycle of the oscillation. I.e. the charging and discharging of the RC-circuit. It can then output frequency, duty cycle, energy usage per cycle or energy usage per second for all combinations of comparator levels.

It is of interest to know how changing the resistance and capacitance affects the power usage and frequency of the oscillator. It has already been discussed how the frequency depends on $\tau = RC$ and it is known that the frequency changes with τ. It has also been discussed how increasing the resistance and decreasing the capacitance so τ is constant decreases the power usage while keeping the frequency stable. How the circuit is affected by changing only one of the values is not known though.

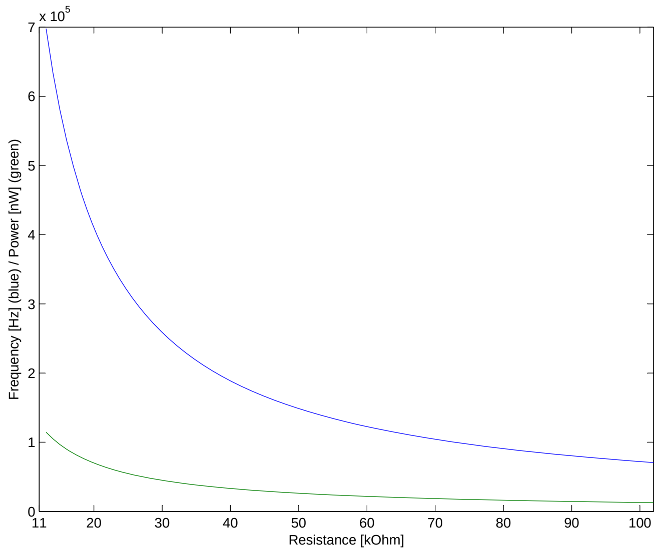


Figure 3.5: The response of the frequency and power usage as the resistance goes from 10kΩ to 100kΩ. Compmin = 0.5V. Compmax = 2.5V. C = 50pF.

In Figure 3.5 the resistance is increased from 10kΩ to 100kΩ while the other characteristics of the oscillator are kept constant. As can be noted both frequency and power sinks with increasing resistance. The decreasing frequency is obvious as increased resistance gives increased τ with constant capacitance. As the resistance is increased the current into the capacitor is also decreased leading to lower power dissipated in the resistor.

In Figure 3.6 on the following page the capacitance is increased from 10pF to 100pF while the other characteristics are kept constant. As with the in-

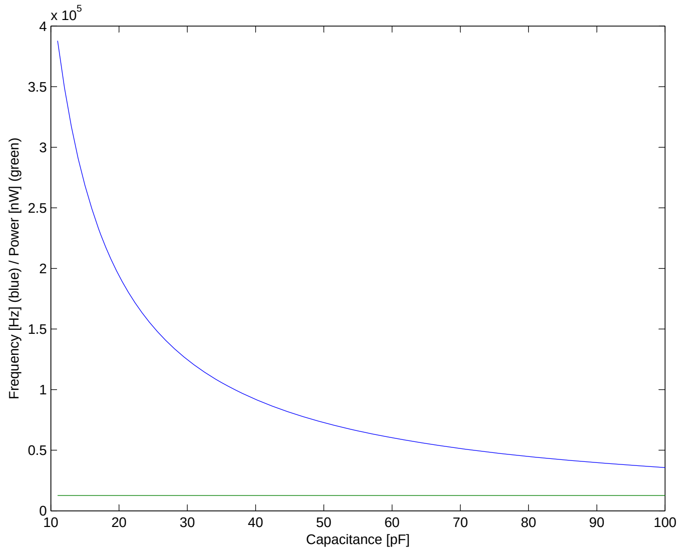


Figure 3.6: The response of the frequency and power usage as the capacitance goes from 10pF to 100pF. $Compmin = 0.5$V. $Compmax = 2.5$V. $R = 100\text{k}\Omega$.

creasing resistance in Figure 3.5 the frequency is decreasing with increasing capacitance. The interesting difference between altering the capacitance as opposed to the resistance is how the power usage remains constant. The reason for this behaviour is how decreasing capacitance affects the response of the RC-circuit. If the capacitance is decreased by 50% it will take 50% less energy to charge the capacitor from one voltage level to another. At the same time though, the frequency is doubled due to the 50% decrease in τ. As the energy used per cycle is halved, the number of cycles per second is doubled giving zero change in the power usage of the circuit.

On the following pages four simulations are shown. All these are simulations on the behaviour of an RC-circuit with $C = 44$pF and $R = 85\text{k}\Omega$. The choice of τ are more or less arbitrary, the important thing is the comparison between the figures, and the general shape of them. These figures will be used to explain the comparator levels influence on energy usage per cycle, frequency, power usage and duty cycle.

Figure 3.7 on the next page shows how much energy is consumed when charging the RC-circuit from the low comparator level to the high. The circuit behaves as expected from the theory in Chapter 1.2, as the circuit consumes most energy when charged from 0V to 3.3V. It is worth commenting the

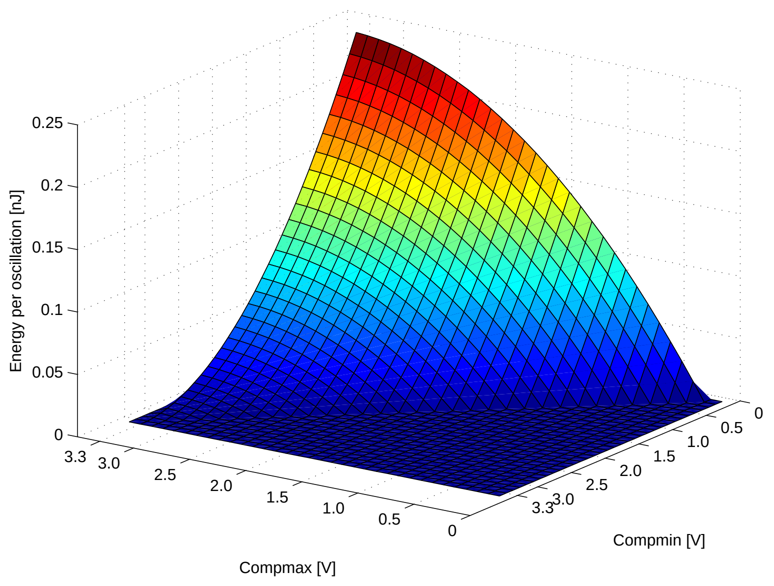


Figure 3.7: The energy usage for different values of *compmin* and *compmax*. $C = 47\text{pF}$ $R = 85\text{k}\Omega$

difference in how the energy consumptions falls depending on compmin and compmax. As was shown in Figure 1.7 on page 11 the power dissipation falls exponentially with rising voltage across the capacitor. Because of this it is more energy efficient per cycle to never discharge the capacitor completely. In the figure this can be observed as increased acceleration of energy consumption when compmin is decreased. On the other hand, the growth in power consumption slows down as compmax increases. These results are the same as foretold in the theory part and stand to confirm the theory. To summarize, the simulations indicate that to achieve an energy consumption as low as possible the RC-circuit should operate between voltage levels as high as possible. The question is how this affects the frequency of the system.

In Figure 3.8 on the next page the frequency at different comparator levels are shown. The figure is capped at 250kHZ to increase visibility as the frequency approaches infinty when the difference between *compmin* and *compmax* approaches zero. There are no surprises in this figure as it only shows that closer gap between *compmin* and *compmax* gives higher frequency. One thing to mention here is that a specific difference between comparator levels does not give a specific frequency. This is not easily observed in Figure 3.8, but can be seen in Table 3.3 on page 29. For instance the frequency while oscillating between 0.5V and 1V is different than between 2V and 2.5V. This comes from

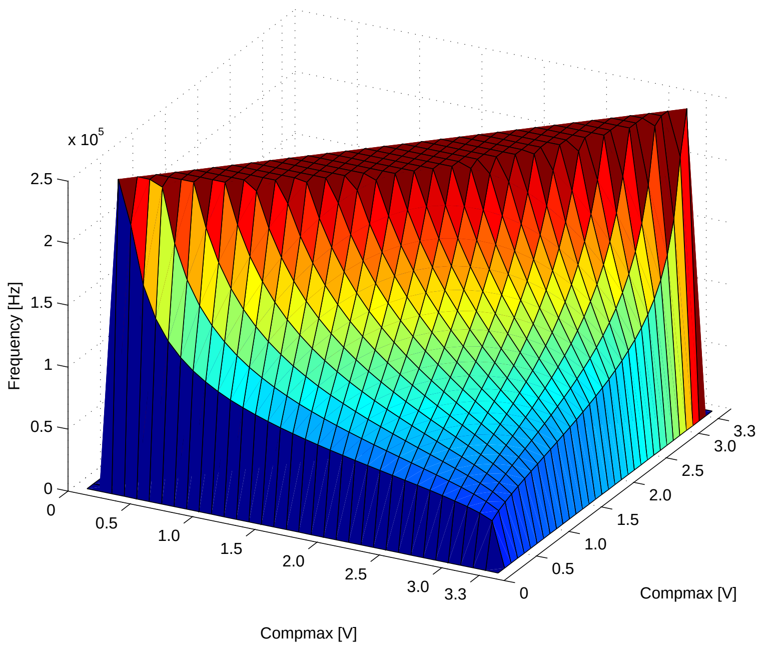


Figure 3.8: The frequency of the rc-circuit for different values of *compmin* and *compmax*. $C = 47\text{pF}$ $R = 85\text{k}\Omega$

the non-linearity in the RC-circuits response.

min\max[V]	1	1,5	2	2,5	3	3,2
0,5	281 135	162 443	116 198	87 443	62 174	48 239
1,0	0	384 467	197 981	126 887	79 821	58 224
1,5	0	0	407 830	189 322	100 705	68 606
2,0	0	0	0	352 982	133 690	82 467
2,5	0	0	0	0	215 100	107 573
3,0	0	0	0	0	0	215 146

Table 3.3: Theoretical and simulated resposnsetimes and maximum frequencies for the comparators with different levels of overdrive.

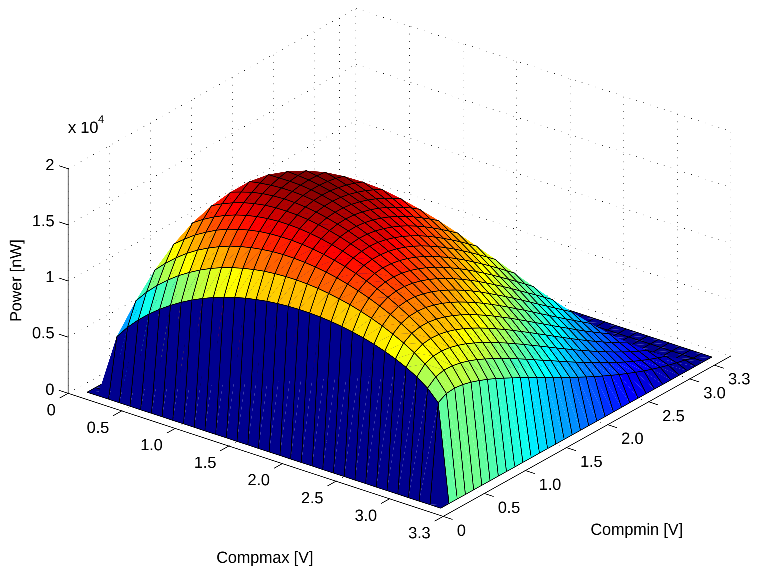


Figure 3.9: The powerusage of the RC-circuit for different values of *compmin* and *compmax*. $C = 47\text{pF}$ $R = 85\text{k}\Omega$

Figure 3.9 is the really interesting figure when it comes to power usage of the RC-circuit. It shows how much power the RC-circuit consumes, i.e. the amount of Jules/second. The most important aspect here is that the power usage is lowest when the comparator levels are set as high as possible. This is because it takes much longer time for the circuit to go from e.g. $2.5V$ to $2.6V$ than $0.0V$ to $0.1V$ with a $3.3V$ input. Even though the energy consumption to increase the voltage across the capacitor a specific value is the same no matter where in the voltage range you do the increase, the lower frequency gives a lower energy consumption per second. As can be observed from the figure, the difference between worst case power usage and best case for this specific oscillator is in the order of two magnitudes. Thus optimizing the comparator levels is a very efficient way of optimizing the power consumed in the oscillator. The most important deduction that can be made from the figure is that all the regions in the middle of the voltageregion give too high energy consumption. The oscillator will most likely have to oscillate between two high voltage levels, or two very low ones to achieve minimal power usage.

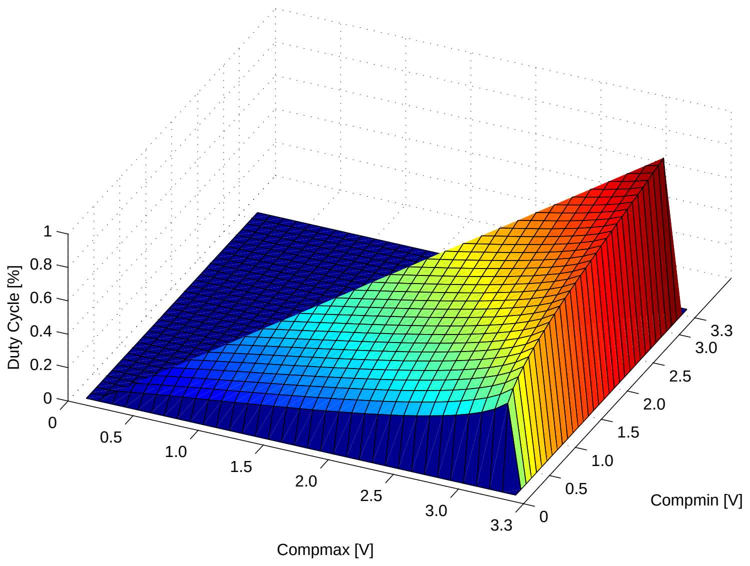


Figure 3.10: The dutycycle of the rc-circuit for different values of *compmin* and *compmax*. $C = 47\text{pF}$ $R = 85\text{k}\Omega$

The final data from the simulations of the RC-circuit can be observed in Figure 3.10. Here the duty cycle of the oscillator can be seen. Duty cycle is the percentage of time that a system is in an active state. In this case it is a number for how long the output of the oscillator is high, and how long it is low. If the dutcy cycle is low or high for a too small fraction of time during a period the digital circuit may have problems noticing the brief transition to this state. The results from Figure 3.9 indicate that to use as little power as possible the oscillator should operate between two very high voltages. According to Figure 3.10 this will lead to a very high duty cycle. Since the digital hardware will be implemented on an FPGA that supports a clock speed of 50 MHZ, and the oscillator will operate in the kHZ region duty cycle will probably not be of any great concern though.

3.3 SR-latch

In the design of the oscillator there is a SR-latch that holds the value from the oscillator and sinks/sources the current to the RC-circuit. This latch could be implemented in the hardware too if it is capable of coping with the current. But in an effort to keep the oscillator separate from the digital hardware it has been implemented on the PCB in this project. Since the latch can be incorporated in the the digital hardware at little cost to area and power consumption it has not been invested much time in finding a unit that is small and uses little power. The latch used in prototype is the MC14043B from On Semiconductor[10]. According to the datasheet this latch is capable of sinking or sourcing 10mA, more than enough to handle the current to and from the RC-circuit.

Chapter 4

Design and simulations of the digital counter

This chapter will discuss the digital part of the system. The digital part is in charge of counting the number of periods on the oscillator during a set time period and deciding if a finger is touching the button or not and signaling that to the general system it is a part of. First this chapter will discuss the counter and look at how long the sampling period should be. After that it will discuss filtering of the input signal and detection of change in frequency. Lastly, it will discuss the FSM and simulate its behaviour.

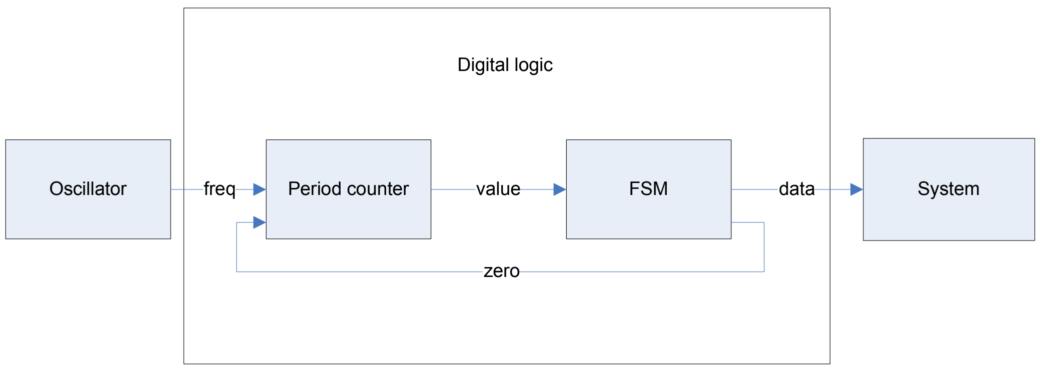


Figure 4.1: Overview of the digital logic. The oscillator functions as a clock for the Period Counter which sends it value to the FSM. The FSM monitors and resets the Period Counter and reports data and button pushes to the general system.

In Figure 4.1 on the previous page an overview of the digital system is shown. Since the design consists of two parts with different clock signals the design will be split into these two part. First there is a need for a counter that counts the periods of the oscillator. This unit is shown to the left in Figure 4.1. Secondly there have to be a Finite State Machine that polls this counter at regular intervals and monitors the change over time. This calls for some sort of memory in the FSM that it can compare the current value with. To avoid overflow in the counter the bit length has to be large enough for the counter to not overflow during a sampling period. When the FSM reads a new sample from the Period Counter it also resets the counter to zero.

4.1 Period Counter

Before the FSM is designed the Period Counter has to be discussed. As was noted in [1] a relaxation oscillator usually has a large level of noise, or so called jitter, in the duration of its period. It needs to be known to what degree this affects the value out from the counter. In Figure 4.2 on the facing page the coeffecient of variation is shown for different sample periods and different standard deviation on the period of the oscillator. The coeffecient of variation is shown in Equation 4.1 and as can be seen is defined as standard deviation, σ, divided by the mean, μ, of a set of data. Due to this independency on the mean, COV is a good way of comparing variance in systems with different mean.

$$COV = \frac{\sigma}{\mu} \tag{4.1}$$

The data used in the figure has been generated with Matlab and the period of the oscillator has gaussian noise with different standard deviations applied. The mean of the oscillators period has been 50 time units. Since the standard deviation of the noise goes from 1 to 50 it becomes apparent that noise levels with standard deviation in the high regions of those simulated are pretty unlikely. As gaussian noise has roughly 95% of its values between $\mu \pm 2\sigma$ the simulations with $\mu = 50$ and $\sigma = 50$ would indicate that the period would be between -50 and 150 95% percent of the time. Noise levels leading to negative period is of course impossible and periods that are three times as long as the mean is also unlikely.

In Figure 4.2 it can be observed that the COV is not particulary larger

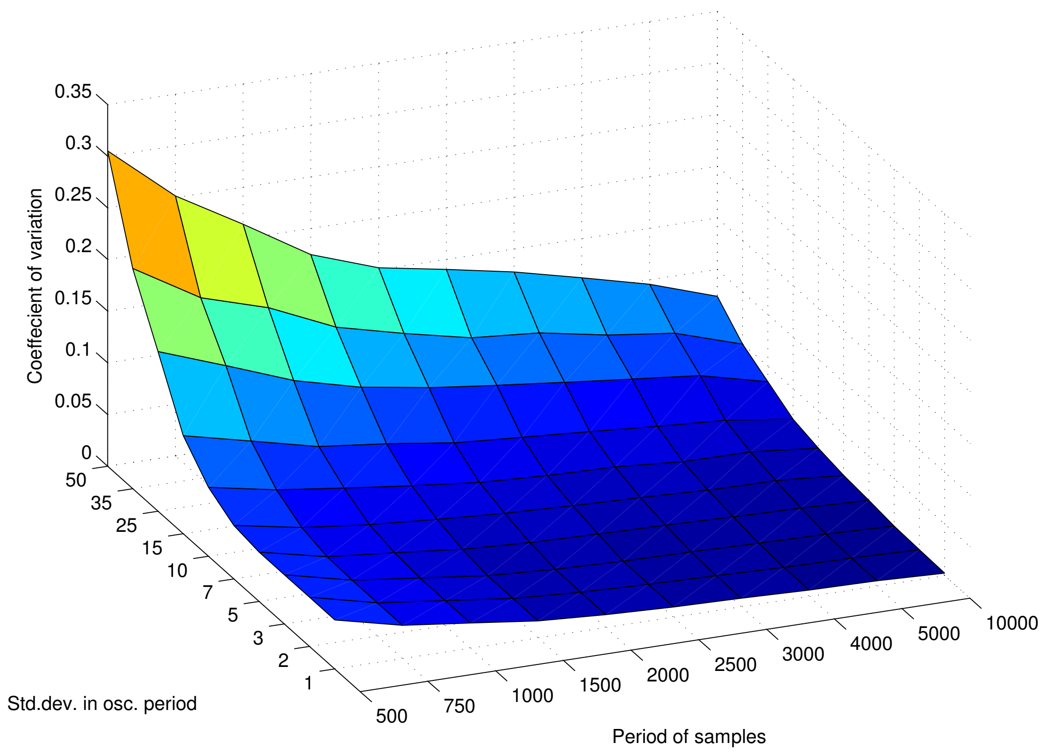


Figure 4.2: The coeffecient of variation for different sampling periods and levels of noise in the system.

with $\sigma = 25$ than $\sigma = 1$ for the larger sampling periods. Thus the period of sampling will itself be a sort of filter on the period from the oscillator as noise has $\mu = 0$. In other words, over time the variations in period will exclude each other leading to low level of noise. So a longer sampling period will give less noise on the value from the Period Counter. A table with the standard deviations in Figure 4.2 can be seen in Appendix G.

4.2 Detection of change in frequency

Detection of change in frequency is done through monitoring the value from the Period Counter which counts the number of periods in the last sample period. This detection is the main task of the FSM. This section will look at FIR- and IIR-filters and how good they are at filtering simulated noisy input from the Period Counter. As discussed in Chapter 2 the use of a dynamic comparison value will be explored. A filter will remove random fluctuations with high frequency but will keep the slow changes over time thus giving a value that can be used as an average over time.

4.2.1 FIR-filter

A FIR filter is a digital filter where there is no internal feedback. The output depends on a finite number of inputs time shifted trough the use of registers. If a FIR filter is to be implemented it is possible to give the different inputs different weight in the final results thereby making more recent samples more dominant than older inputs. Alternatively, all the inputs can be equally weighted so the output is merely an average of the k last inputs. The value of k determines the memory usage as each sample will have to be stored in registers. In Equation 4.2 the output from such a filter can be seen with k input.

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2) + ... + a_kx(n-k) \tag{4.2}$$

The coeffecients $a_0..a_k$ decide how much each of the last k values are weighted. Since there is no way to know which samples are closer to the real value it is impossible to weigh the samples according to expected levels of noise. But since the signal might change over time due to temperature og atmospheric changes it might be of value to weigh the most recent values higher than the older ones. Although, this depends on how many samples are stored at a time as with fever samples all the samples will be within a small time frame. It is important that the sum of the coeffecients is 1, that is the equation $\sum_{i=1}^{k} a_i = 1$ has to hold. If not the output will either approach infinty or zero depending on whether the sum is larger or smaller than 1. In Figure 4.3 on the facing page the filters in Table 4.1 on page 38 have been applied to a signal.

Since this is a digital system the choice of coeffecients has great impact on the size and power consumption of the system. If a coeffecient is set to be a random rational number there would be a need for multipliers in the system. If the coeffecients are set to be any number n where $n = 2^k$ and k is an integer it is possible to obtain the final result by shifting the signal. E.g. $2^4y_n = y(n) << 4$ which is free to implement in hardware. Every coeffecient should then be set to such a number to save power consumption.

A FIR filter like the one in Equation 4.2 can also be seen in Figure 4.10.

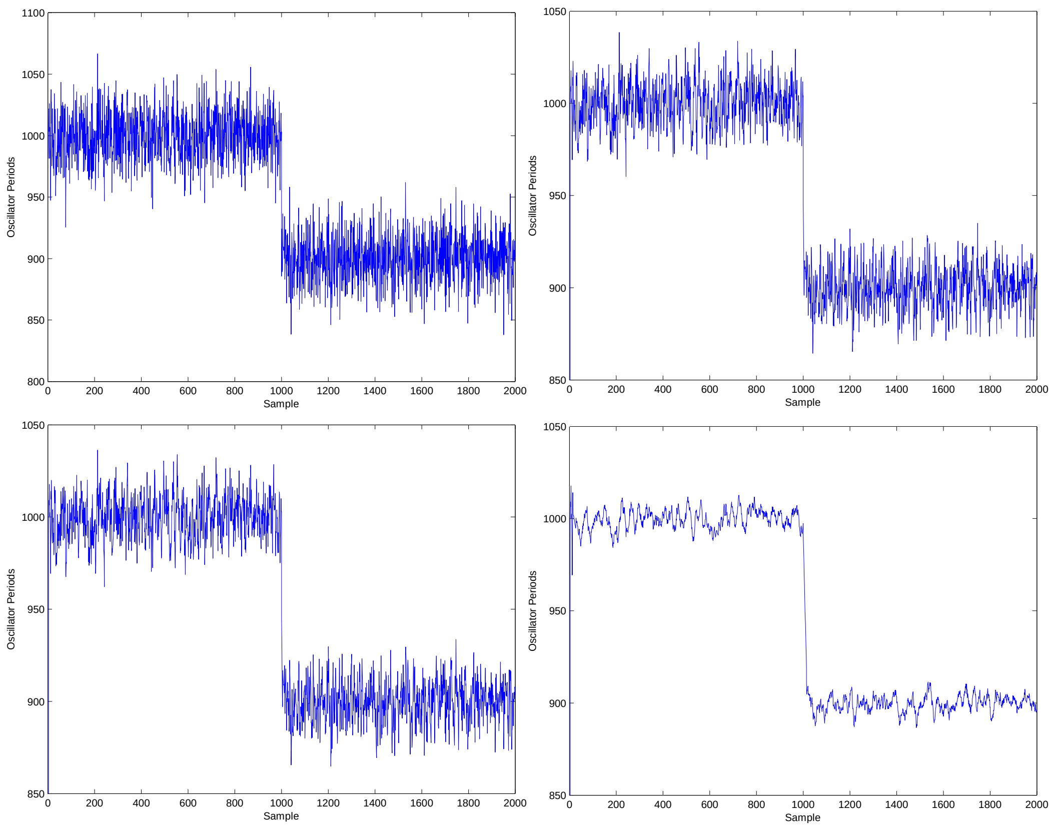


Figure 4.3: Input (top left) and outputs of different FIR-filters. Top right and bottom left has wheigthed inputs while bottom right is normal average of last 16 inputs.

Figure	Coeffecients	k	Registers	Adders
Upper right	$a_i = 2^{-i}$ where $a_k = a_{k-1}$	4	3	3
Lower left	$a_i = 2^{-i}$ where $a_k = a_{k-1}$	16	15	15
Lower right	$a_i = \frac{1}{16}$	16	15	15

Table 4.1: Comparison of the filters in Figure 4.3.

Here it can easily be observed that the filter needs $k-1$ storage elements with a bit length equal to the bit length of the counter. It also needs $k-1$ adders. This will in the end add up to quite alot of hardware that drains energy. In Figure 4.3 on the preceding page this kind of filter has been applied three times to a signal. The input consists of 1000 samples with $\mu = 1000$ and $\sigma = 20$ followed by 1000 samples with $\mu = 900$ and $\sigma = 20$. This simulates inputs from the Period Counter before and after a drop in the frequency by 10%. The input is then filtered by the filters shown in Table 4.1.

In Figure 4.3 on the preceding page the input signal and the output from the three FIR-filters are shown. Note that there is no great difference between the output from the upper right and the lower left filters. This is because the coeffecients quickly make the inputs irrelevant. E.g. coeffecient $a_3 = 2^{-3} = \frac{1}{8}$ while $a_{16} = 2^{-16} = \frac{1}{65536}$. In other words, the output is greatly dominated by the first few inputs. In fact, the most recent input has a weight of 50% on the output. In the filter on the lower right every input is weighted equal. The output from this filter is then a normal average of the last 16 inputs.

In Figure 4.4 on the next page the response of four filters where the inputs are weighted equal are shown on the right. On the left the input is shown. As can be seen in the figure the more inputs the filter has, the less noise there is on the output and the slower response the filter has. In the figure the red output has two inputs, green has four, blue has eight and black has sixteen.

4.2.2 IIR-filter

Another approach is to use an IIR filter. A IIR-filter has internal feedback that makes the response of the filter last indefinetely. In Equation 4.3 on the facing page the equation for a IIR-filter is shown. Here the output $y(n)$ depends on a former value of y, $y(n-1)$. Since the output depends on former outputs instant changes in the input will not affect the output instantaneously.

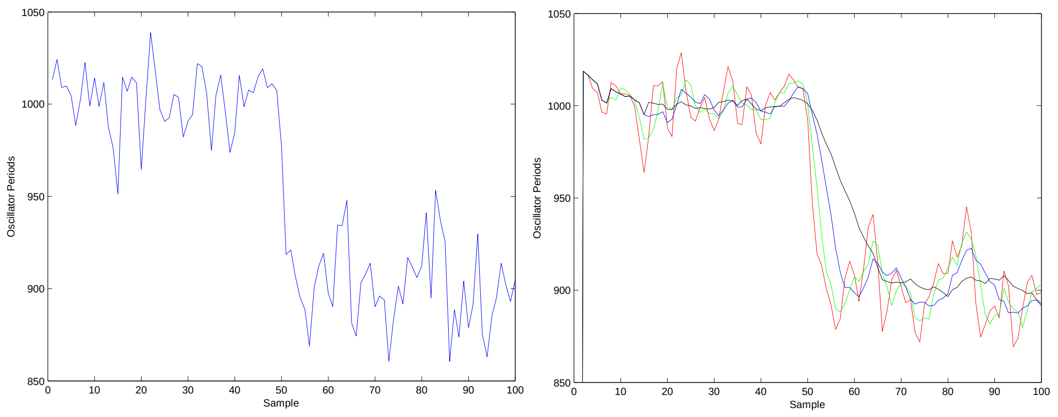


Figure 4.4: Response of FIR-filters with equal coeffecients. Red output is average over 2 last inputs. Green, 4 inputs. Blue, 8 inputs. Black, 16 inputs.

Instead, only changes to the input that last over time will affect the output. Unlike the FIR-filter discussed above this filter does not need to store inputs over time, it is only dependent on the current input and the old output in its calculations. Since the input and output will be stored in a registers either way the cost in hardware is only the calculation of $bx(n)$ and $ay(n-1)$ and an adder to sum the results.

The coeffecients of the filter should be discussed. In this project $a > b$ so the current input has less weight in the output than the last output. To avoid the ouput growing larger and larger or slowly approaching zero the sum of the coeffecients would have to equal 1. In other words, $a + b = 1$. With this relation between the coeffecients the output would be a weighted average of the former output and the current input.

$$y(n) = ay(n-1) + bx(n) \tag{4.3}$$

In Figure 4.5 on page 41 an input simulating the counts from the periodic counter has been filtered by three different IIR-filters on the form in Equation 4.3. The input can be seen on the upper left, it consists of 1000 samples with $\mu = 1000$ and $\sigma = 20$ simulating the counts from the period counter. After 1000 samples the frequency falls and the signal then consist of 1000 samples with $\mu = 900$ and $\sigma = 20$. Together this would be how the period counter would react to a rather noisy signal from the oscillator if a finger pushed it reducing the frequency by 10%.

The coeffecients have to be set in way to make it as easy as possible to compute the new output. As discussed above the coeffecients should be on the form 2^k where k is an integer. For a IIR-filter where the output only depends on two values this would only allow the coeffecient $\frac{1}{2}$. Since the former output should be weighted higher than the input just shifting the signals will not be sufficient in this case. The reason for setting $\frac{A}{B}y(n-1) = y(n-1) - \frac{1}{B}y(n-1)$ where $B - A = 1$ is to avoid the need for a multiplier in the logic. This way the fractions can be done by shifting and subtracting. The hardware needed to filter the signal is two adders assuming registers to hold the input and output already exists in the circuit.

Figure	Coeff. a	Coeff. b	Equation
Upper right	$\frac{3}{4}$	$\frac{1}{4}$	$y(n) = y(n-1) - \frac{1}{4}y(n-1) + \frac{1}{4}x(n)$
Lower left	$\frac{1}{8}$	$\frac{7}{8}$	$y(n) = y(n-1) - \frac{1}{8}y(n-1) + \frac{1}{8}x(n)$
Lower right	$\frac{1}{16}$	$\frac{15}{16}$	$y(n) = y(n-1) - \frac{1}{16}y(n-1) + \frac{1}{16}x(n)$

Table 4.2: Comparison of the filters in Figure 4.5. Every filter needs two adders and two registers.

In Table 4.2 the coeffecients and the final calculation needed to be done is shown for three filters. The resulting signals from the filters can be seen in Figure 4.5. The input is quite noisy and as can be seen the input before a button push is sometimes lower than the highest values after a button push. Thus the noise is too large to use a static value lower than 1000 as the threshold value for detecting button pushes. Even though the signal goes below e.g. 900 only after the button push the signal is so full of noise that there would be indicated several buttonpushes after eachother if a threshold of 900 was used.

After filtering the signal though, it would be possible to detect button pushes with a static value with all the filters. E.g. the system could be set to signal button push whenever the filtered signal was lower than 950. As has been discussed earlier a dynamic solution is desirable. The values from two filters with different coeffecients could be compared. In Figure 4.6 on page 42 the response of the three filters from Table 4.2 can be compared to the input. Here it can be observed how the filter with $b = \frac{1}{16}$ (green) has much slower response and contains much less noise than the filter with $b = \frac{1}{4}$ (blue). It could then be possible to compare the values from these two filters to determine button pushes thereby eliminating both the use of the noisy input signal and static values in the comparison. This solution is more complex leading to higher power usage and area. It is therefore important to do estimations of

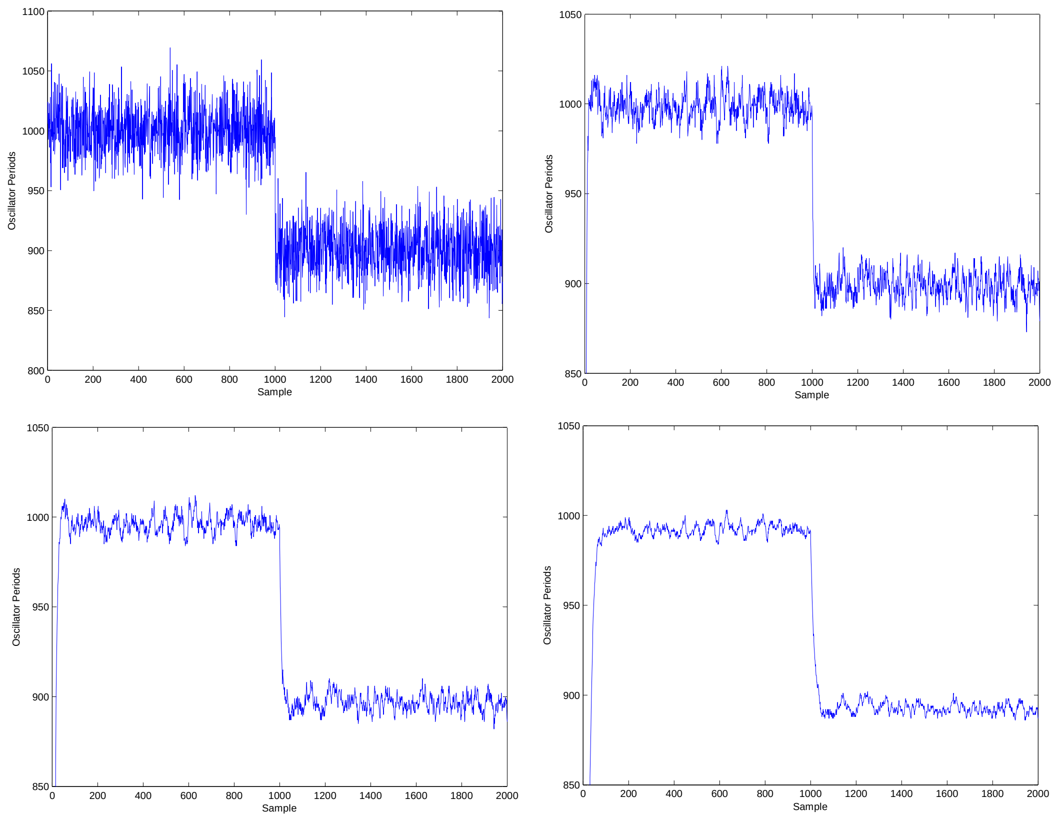


Figure 4.5: Input (top left) and outputs of different IIR-filters. Top right: $a = \frac{3}{4}$. Bottom left: $a = \frac{7}{8}$. Bottom right: $a = \frac{15}{16}$. Ref. Equation 4.3. $a + b = 1$ for all filters.

the power usage of both a static and a dynamic approach after the hardware has been written in Verilog.

4.2.3 Final considerations regarding filter

All the filters simulated above managed to filter the applied signal to a degree that it would be possible to detect button pushes. The choice of which filters to use later in the project depends only on the amount of power and area used by the different implementations. In Table 4.3 on the following page a comparison of all the filters simlated in the last two sections are shown. The table focuses on the hardware needed for the filters and assumes that the current input and last output are already stored in registers in the system.

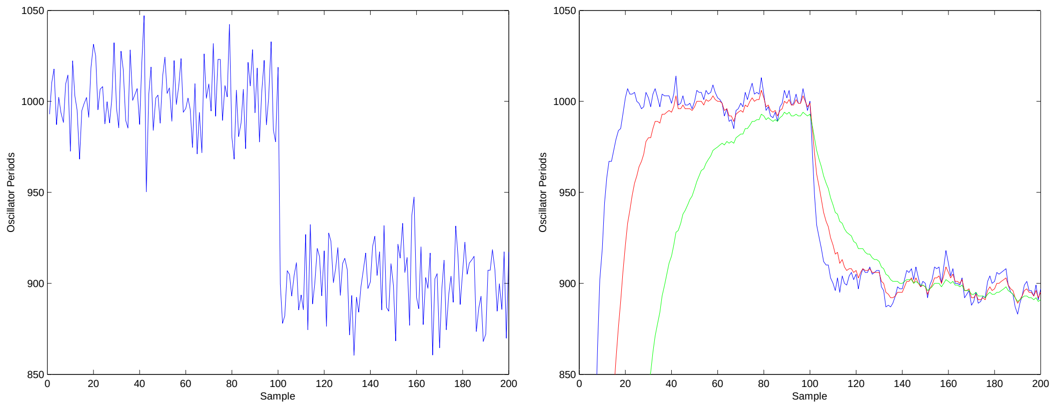


Figure 4.6: Response of different IIR-filters. Blue output has $a = \frac{3}{4}$. Red, $a = \frac{7}{8}$. Green, $a = \frac{15}{16}$.

Filter #	Type	Figure	Regs	Adders	Largest shift
1	FIR	4.3 UR	3	4	4
2	FIR	4.3 BL	15	15	16
3	FIR	4.3 BR	15	15	16
4	IIR	4.5 UR	0	2	2
5	IIR	4.5 BL	0	2	3
6	IIR	4.5 BR	0	2	4

Table 4.3: Comparison of the hardware needed in the filters simulated above. It is assumed that the system already has the current input and the last output stored in a register. UR = upper right. BL = bottom left.

As can be seen the IIR-filters generally need much less hardware. This mainly comes from the fact that they calculate their outputs from only two inputs. Another aspect that has yet to be discussed is the effect of shifting on the signals. In Chapter 4.1 it was found that a sampling period that gave a value from the Period Counter in the region of a few thousand would give a low level of noise. As every added bit in the sampling increases the size and power consumption it is in this projects interest to keep the bit length of the counter as low as possible. A bit length of 12 would give a maximum value from the counter of 4096. By setting the period of the FSM to such a value that the counter is as close as possible to 4096 without experiencing overflow due to noise the level of noise from the counter will be held at a minimum for that specific bit length. With a 12 bit signal filter two and three in Table 4.3 are useless as they shift the signal 16 bits. Likewise, filter six shifts the value 8 bits, that leaves only 4 bits left which will most likely give too low resolution on the signal. If filter one, four and five are compared in Figure 4.3 and 4.5 it can be seen that filter one has a much higher level of noise. When the increased size of FIR-filters are taken into account the use of IIR-filters seems apparent. It is therefore assumed that IIR-filters will be the most cost efficient and noise resistant solution and they will be used in the rest of the thesis.

4.3 Design of FSM

Now that it is known how the length of the sample period affects noise from the Period Counter choices can be made on the functionality of the FSM. This section will start with a look at the sequence of tasks the FSM is supposed to do. After that, it will discuss the size of the hardware with different filters. It will then look at a design choice that has had to be made before it finishes by simulating four different filtering techniques to find the response time of the system.

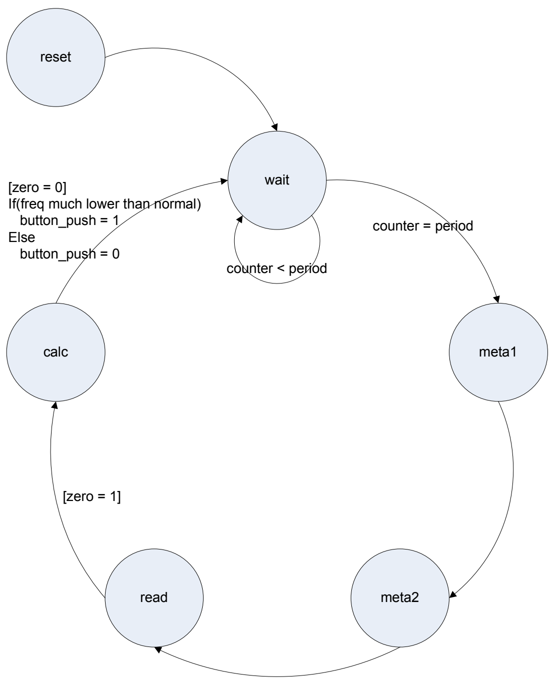


Figure 4.7: Finite State Machine of the system.

In Figure 4.7 on the previous page the Finite State Machine is shown. Once a signal, *counter*, has signaled that the FSM has waited a specific time period the FSM starts to do another sample of the period counter. As discussed in Chapter 2.1 the value read from the counter has to go through a chain of two registers to avoid metastability. This is done in state *meta1* and *meta2* and requires two clockperiods. In the next state, *read*, the unit reads the data from the registers and applies filters to the signal if such an approach is used. In the final state, *calc*, the signals are compared, and the unit will signal whether the button has been pushed or not. Finally, it goes back to the state *wait* and waits until it is time for a new sample. The *reset* state is the state the system goes to after a reset or at startup, in this state the FSM's registers are set to zero and the system prepared for operation.

The design of the FSM is more or less the same no matter what sort of filtering technique is used and whether the values are dynamically or statically compared. In Figure 4.8 a block view of the FSM and the logic it controls are shown with a generic filter module.

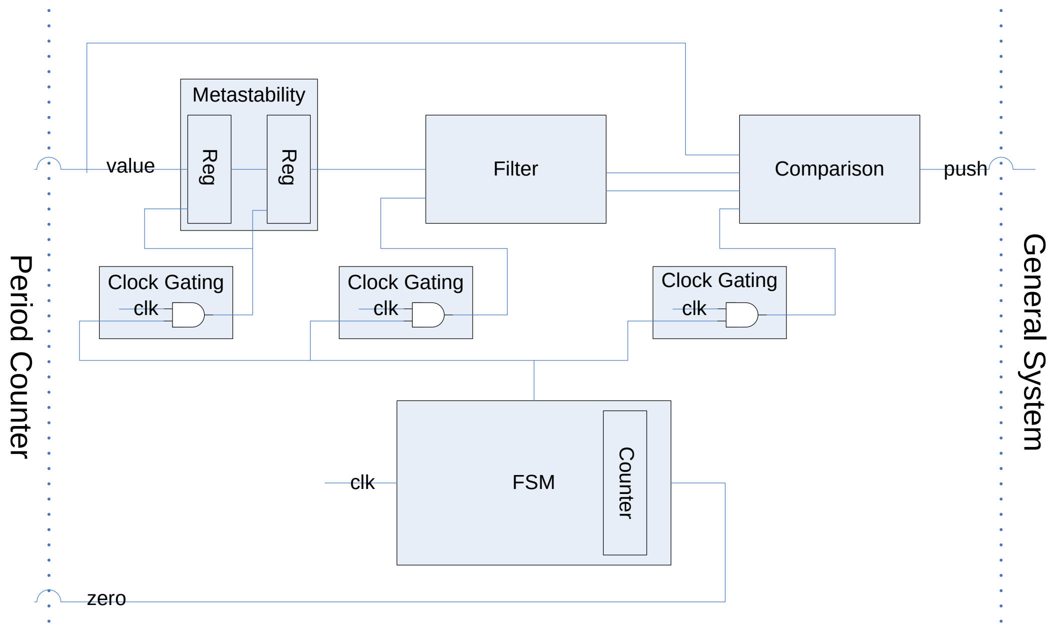


Figure 4.8: The Finite State Machine and the logic it controls.

The hardware required in this project is not very large as can be seen from the figure. There are in essence four seperate modules in the design. At the bottom is the Finite State Machine that controls the system and counts the time needed to wait between sampling periods. To the upper left in Figure 4.8

the chain of registers that will handle metastability is shown. As the values from these are only needed once during a sampling period a clock gate has been applied so they only update their state when a new sample should be read. In the middle the filter is shown. As noted above only a generic version is shown here. As the metastability module, the filter has been clock gates since it only needs to update its registers when a new sample is read. The final module is the comparison module which will do a comparison between filtered signal(s) and/or the current input depending on the filtering technique used. Like the other modules it has been clock gated to only update its registers when needed.

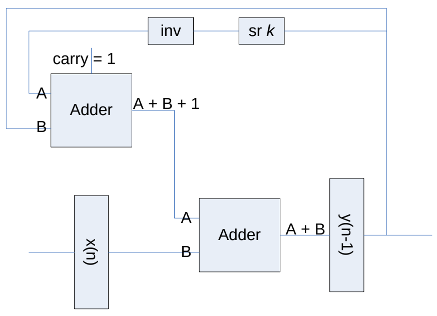


Figure 4.9: An IIR-filter with with output $y(n) = \frac{2^k-1}{2^k}y(n-1) + \frac{1}{2^k}x(n)$. sr k is a k-bit binary right shift, in other words division by a factor of 2^k.

In Figure 4.9 the hardware of an IIR-filter is shown. As can be seen there is feedback from the output that makes the filter have an infinite input response. For the special case $k = 2$ the inverter, shift operator and carry in may be removed and the input on the topmost adder replaced by $\frac{1}{2}y(n-1)+\frac{1}{4}y(n-1) = \frac{3}{4}y(n-1)$. This would save the inverter and carry in in the circuit. For k other than 2 this solution would require additional adders.

In Figure 4.10 on the following page the hardware of a FIR-filter is shown. As discussed in Chapter 4.2.1 this design uses k registers to store the last k inputs. In addition it uses several adders that not only consume power and area but give a very long critical path thus decreasing the maximum frequency of the system. This is not necessarily a problem in this design as low frequency is desirable, but it is worth noting.

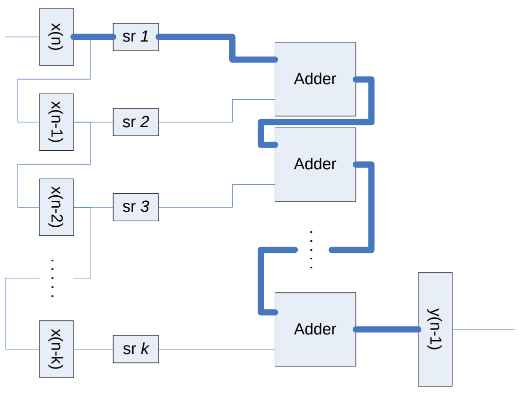


Figure 4.10: A FIR-filter with output $y(n) = \frac{1}{2}x(n) + \frac{1}{4}x(n-1) + ... + \frac{1}{2^k}x(n-k)$.

4.3.1 Erroneous button pushes due to filter response

Before the results from the simulations are presented a design choice has to be commented. In Figure 4.11 on the next page input and data from the filter of an implementation of the digital circuitry simulated in ModelSim is shown. The input (blue) drives the two IIR-filters, green and red. The detection of button pushes in this design is done when the fast filter is lower than slow filter. But as the filters rise to their final values it can be seen that the the slow filter stabilizes at a value greater than the slow one. This leads to a false button push at sample 45. Since the slow filter is supposed to be an average over time it should not be affected by button pushes. Therefore its value will not be updated as long as a button push is detected. As the slow filter stabilizes above the fast filter in Figure 4.11 this implementation will not work as the filter will continuously signal a button push after the filters have stabilized.

One way to rectifiy this could be to demand that the fast filter had to be a certain value lower than the slow, not just lower. But this would add an adder in the comparison module as can be seen in Equation 4.4.

$$fast < slow - value \tag{4.4}$$

Instead the three LSB of the fast and slow filter signal has been set to constantly be zero for the slow filter and one for the fast filter. This ensures that the fast filter signal will stabilize at a higher value than the slow filter. This has the added benefit of less switching in the circuit as three of the signals,

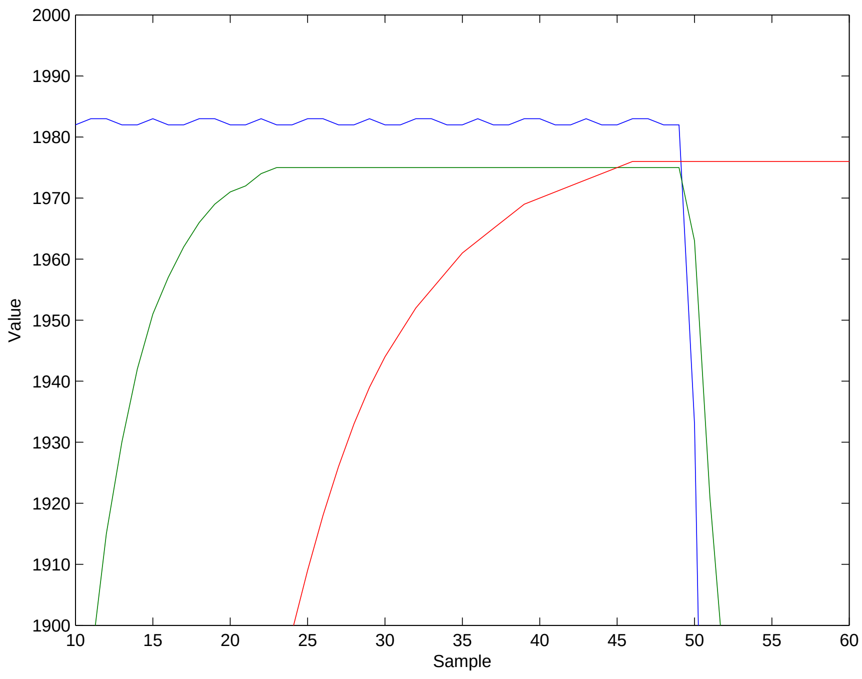


Figure 4.11: Input to the filter(blue) and response of two IIR-filters. One with $b = 2^{-2}$ (green) and one with $b =^{-3}$ (red). In this filter the comparison between the slow filter and the fast filter gives detection of button push when the filters stabilize. Frequency of oscillator: 80kHz. Sample period: 25.15ms.

in this case, twelve bit are static. In Figure 4.12 the result can be seen for a circuit and input that is the same as in Figure 4.11.

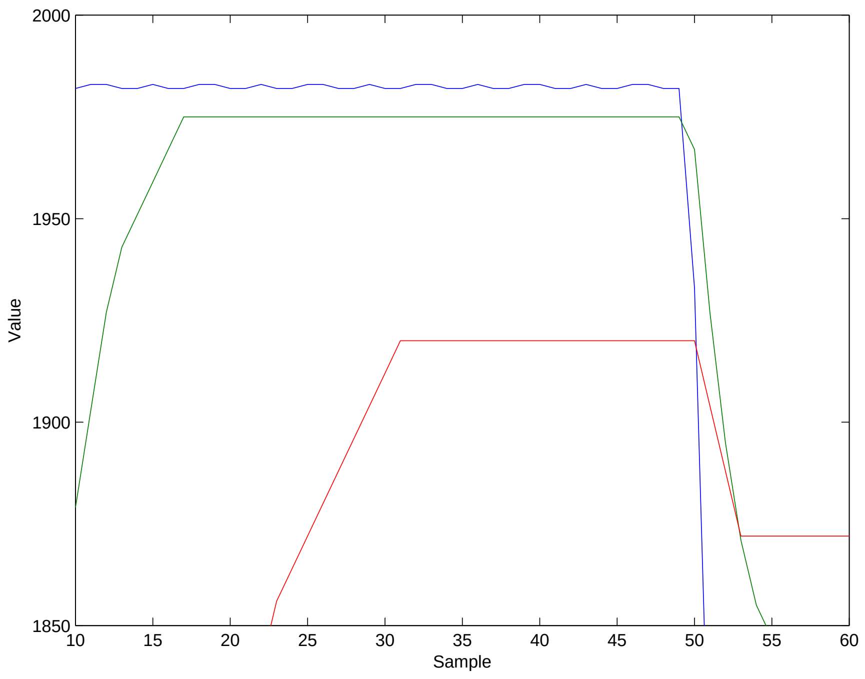


Figure 4.12: Input to the filter(blue) and response of two IIR-filters. One with $b = 2^{-2}$ (green) and one with $b =^{-3}$ (red). In this filter the slow and fast filters stabilize at values that avoids false output. Frequency of oscillator: 80kHz. Sample period: 25.15 ms.

As can be seen from the figure the filters stabilize at different values now. There are no erroneous button pushes detected. And the real button push is detected in a few sampling periods.

The choice of setting three LSB static as opposed to perhaps the two LSB must be decided experimentally for a particular design.

4.3.2 Response of the simulated hardware

The Verilog code for the FSM and the logic it controls can be viewed in Appendix B. The wave forms from ModelSim are not interesting themselves other than to prove that the hardware operates as desired. Therefore simulations can be reviewed in Appendix D How the digital filters responds are more interesting, and the data from simulations in ModelSim will be discussed in this section.

The response time of different filter designs are of interest as they give insight into the delay from a button push occurs until the hardware detects the decreased frequency. It is also of interest to know how the filter responds to button pushes at different frequencies of the oscillator. In this section four filtering techniques will be compared for four different frequencies in the oscillator. As discussed earlier the sampling period will be tuned to give a value from the Period Counter that lies as close to possible as the max value of a 12 bit signal without risking overflow.

A choice on which filters to compare and what frequencies from the oscillator to use must be made. Energy Micro AS has requested that the digital logic operates at 32.768kHZ or any 2^k division of that frequency. The reason for this is the availability of this frequency to logic in their microcontrollers while the microcontroller core is in sleep mode. In these simulations the digital circuits will operate a 8.192kHZ. In Section 4.2.3 it was found that IIR-filters with $b = 2^{-2}$ and $b = 2^{-3}$ are of greatest interest. Thus combinations of these, static comparison values and the unfiltered input will be tested with different frequencies.

Comparison \ kHz	Freq. drop	50	100	150	250
$IIR_{b=2^{-2}} < IIR_{b=2^{-3}}$	10%	368	82	40	35
	20%	140	43	40	19
$IIR_{b=2^{-2}} < 3600$	10%	297	208	118	99
	20%	143	92	66	35
$input < IIR_{b=2^{-2}}$	10%	67	14	14	3
	20%	67	14	14	3
$input < 3600$	10%	67	14	14	3
	20%	67	14	14	3
Sampling Period		76	38	25	15

Table 4.4: Response time of different filter techniques to detect drop i frequency. All numbers in ms. Digital clock is 8192Hz.

In Table 4.4 the response time of different filtering techniques are shown. The coloumn on the left shows the requirement for the FSM to signal that a button push has occured. For every combination of filter technique and frequency on the oscillator both a 10% drop and a 20% drop has been simulated. The sampling period has been tuned so a regular sample will be roughly 3800 before button push. In ModelSim the circuit has run with the oscillator at the specified frequency for a time long enough for the filters to stabilize. The frequency has then suddenly dropped and the time between the frequency droppes and the FSM signals a button push has been measured.

As can be seen there seems to be a strong correlation between frequency of the oscillator and response time of the digital circuit. This comes at the cost of increased switching activity in the circuit as a lower sampling period increases the frequency of updating registers in the filter. The results in the bottom two rows needs to be commented. In these comparisons the input value is compared to either an average over time (row three) or a static value (row four). Since the input has immidiate response time to the frequency change the module will signal that a button push has occured as soon as the FSM updates its registers. If the drop in frequency happens close enough to a sample the value from the counter may not drop enough to signal a button push though. Either way, the response time will never be larger than two sample periods, and usually shorter than one as long as the frequency change is large enough.

A well known rule of thumb is that the perceived response time should be less than 100 ms [4]. Recent research has found this to be too strict for cases where the user is pushing a button and that 150ms is closer to the average threshold of noticable delay[4]. Since the hardware described in this report is connected to some larger system its response time should be low enough for the general system to respond to the button push within reasonable time. The oscillators frequency will have to be determined according to which filtering technique is used and the bit length. Especially the solutions with filtered input (top two) needs higher frequency on the oscillator to achieve a responsetime that would not be perceived as delay by a human operator. In Section 3.1 the maximum frequency of the system is found to be 571.5kHz in the best simulated case, but might be lower than 100kHz according to the datasheet of the comparator used in this design. The maximum frequency of the oscillator will determine which of the implementations in Table 4.4 are usable when it comes to response time.

Chapter 5

Prototype and other hardware and tools

The problem description calls for a prototype of the capacitive touch system to be built and tested. Design of the prototype and the FPGA that is used for the digital part is discussed in this chapter.

5.1 Prototype

There are several ways to build a prototype for this project. One simple solution would be to build everything on a breadboard. A breadboard would give good opportunity to alter the design on the fly, and try different setups (e.g. changing the RC-circuit or comparator levels) after the prototype is built. A downside though, is the probability of errors as the system is constantly changed.

The solution this thesis has gone for is to design the layout of the oscillator and order a PCB from a supplier. As long as the layout is correct, the resulting PCB should have less error sources as there is less risk of bad soldering, loops in signal- or ground paths and better control over capacitance in signal paths. Another good reason is the need for buttons that are supposed to sense the button pushes, on a breadboard there is no way to make such surfaces unless they are bought from a supplier. As the goal of this project is to build the system from scratch store-bought sensors are unwanted.

In Figure 5.1 the layout of the prototype is shown side-by-side of the final PCB delivered from the supplier. On the card there are four oscillators that operate independently of eachother. By having four oscillators on each card an error in the design on one of them does not result in a useless PCB. By ordering a few cards there is also possible to test several combinations of RC-circuit and comparator levels. The whole card has a PATA-connector at the bottom that is used to connect the card to the development board discussed in Chapter 5.2. Connection between the PCB and prototype can then be more direct and potentially avoid issues with crosstalk and capacitance in the paths which could occur in a cable.

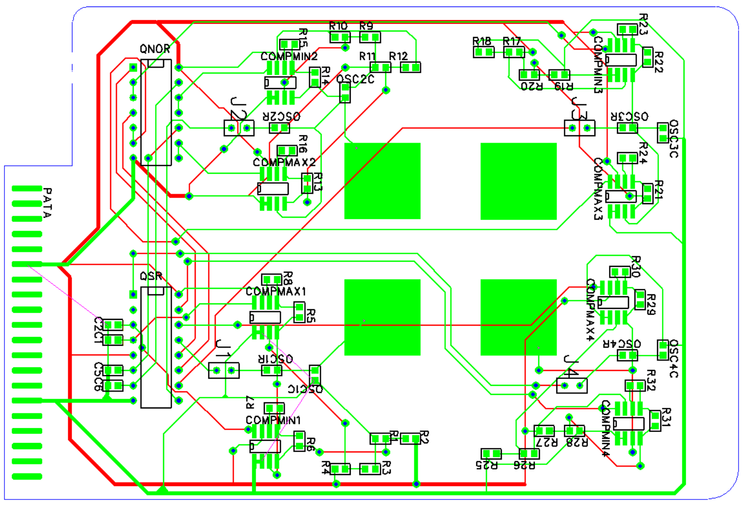


Figure 5.1: The layout of the PCB containing the prototype. It consists of four seperate oscillators connected to the development board through a PATA connector.

The four green squares in Figure 5.1 are the capacitive touch areas. These are connected to the V_- node in Figure 1.4 on page 7. Thus a finger touching the pad will induce a capacitance in that node increasing the τ of the RC-circuit. On the backplane of the PCB there is a groundplane that is as large as all four buttons. The reason for this is to give some capacitance in the pad it self. Due to concern that the capacitance in the pad would be so large that the fingers induced capacitance would not reduce the frequency significantly the backplane was removed on some of the pads.

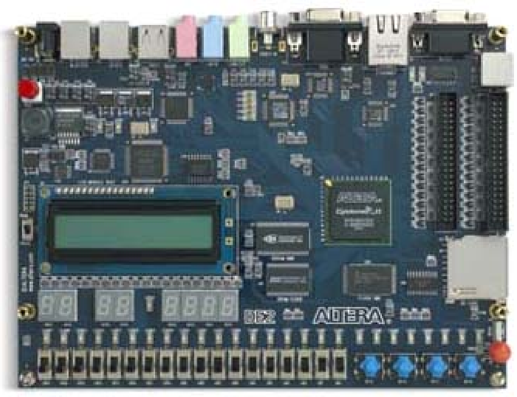


Figure 5.2: A picture of Altera's DE2 Development Board with the Cyclone II FPGA used to test the digital hardware.

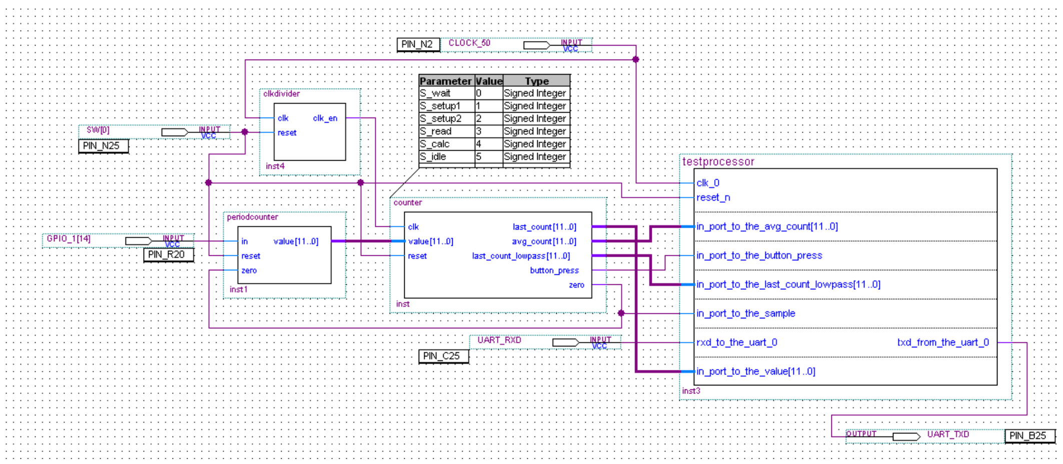
5.2 Development board

The digital part of the problem description is supposed to be implemented on an FPGA to test its functionality in practice. The FPGA used is a Cyclone II EP2C35 on a DE2 Development Board as this is known to the author. Together with the hardware designed in this project a soft-core CPU from Altera, Nios II, has been used to monitor the circuit. Due to the much higher clock frequency of the Nios II compared to the hardware designed in this project it has been possible to program the Nios to poll the output from the hardware and transmit every sample to a computer over RS-232. The program running on the Nios II can be reviewed in Appendix F.

In the Figure 5.3 on the next page the hardware implemented on the FPGA can be viewed. A larger image can be viewed in Appendix E The system consists of the hardware designed in this project and a Nios II that monitors the hardware and reports every sample to a computer.

5.3 Software

This section will give a brief description of the software used in this project. It will also discuss the program developed to give simulations of the behaviour of the RC-circuit depending on R, C and the comparator levels.



Parameter	Value	Type
S_wait	0	Signed Integer
S_setup1	1	Signed Integer
S_setup2	2	Signed Integer
S_read	3	Signed Integer
S_calc	4	Signed Integer
S_idle	5	Signed Integer

Figure 5.3: The hardware programmed to the FPGA. It consists of the Period Counter, FSM and Nios II.

5.3.1 Software

This section will list the software used in this project and briefly describe their uses.

- Verilog is a Hardware Description Language that has been used to describe all the hardware used in this project.
- ModelSim 6.3d from Mentor Graphics is a hardware simulation environment used to write and debug the digital hardware. ModelSim has also been used to simulate the switching activity in the circuit which has later been used to do power estimations in Design Vision.
- Design Vision Y2006.06 from Synopsys has been used to synthesize the digital hardware and generate netlists that ModelSim can run simulations on. The switching activity simulated in ModelSim has then been used to estimate the power usage.
- AimSpice Student Version 5.3c has simulated the behaviour of the oscillator and especially the comparator.
- Matlab R2009a by The Mathworks has been used to plot graphs from the data generated from ModelSim, recieved from the hardware or generated with the program written as part of this thesis.

5.3.2 Program for calculation behaviour of the RC-circuit

Since it has been vital to know how R, C and the comparator levels affect the power usage of the RC-circuit a program has been written in C that can calculate the behaviour of the circuit. This program has used Equation 5.1 and 5.2 to calculate the response of the RC-filter. This has been done by calculating the voltage across the capacitor with 1ns increments of time both when the capacitor is charged from *compmin* to *compmax*, and discharged.

$$V_C = V_{in}(1 - e^{\frac{-t}{\tau}}) \tag{5.1}$$

$$V_C = V_{in}e^{\frac{-t}{\tau}} \tag{5.2}$$

The program is capable of calculating the energy used during one cycle of the oscillator, frequency, duty cycle and average watt of an oscillator with specific R, C and comparator levels. Or it can output those values for every combination of *compmin* and *compmax* between ground and V_{DD} for an oscillator with specific R and C. The code can be reviewed in Appendix A.

Chapter 6

Results and discussion

The previous chapters have simulated different aspects of the system and given results according to theory and simulations. This chapter will look at real results from the actual prototype and hardware operating on an FPGA. It will also contain power estimations of the digital circuit according to Design Vision.

6.1 Resulting behaviour of the oscillator

This thesis has not measured the power consumption of the actual oscillator. Instead the results presented in this chapter will be measured frequencies of oscillators with and without button pushes. The power usage presented in this section will be from the program described in Section 5.3.2.

Which RC-circuits and comparator levels to test must first be discussed. Since the power usage of the circuit will not be measured on the prototype there is no need to test different circuits with power consumption in mind. The tests are just be to see if the circuit operates as expected.

In Table 6.1 on the next page an oscillators response to button pushes with different materials between the pad and the finger can be seen. The oscillators characteristics is listed in Table 6.2 on the following page and the characteristics of the mediums in Table 6.3 on page 59. The oscillator is tested both with and without a ground plane at the opposite side of the pad. Table 6.1 also contains measured voltage levels the RC-circuit oscillates between

Layer and Push			Measured				Theoretical	
GP	Layer	Push	Freq.	Drop	V_{min}	V_{max}	Freq.	Power
Y	-	N	44.4kHz	-	0.64V	2.44V	59.5kHz	20.8μW
Y	-	Y	26.7kHz	39.8%	0.64V	2.28V	-	-
Y	Paper	Y	36.7kHz	17.3%	0.64V	2.44V	-	-
Y	Glass	Y	42.5kHz	4.3%	0.64V	2.44V	-	-
N	-	N	46.8 kHz	-	0.6V	2.5V	55.6kHz	20.6μW
N	-	Y	22.5 kHz	51.9%	0.64V	2.33V	-	-
N	Paper	Y	35.7 kHz	23.7%	0.6V	2.5V	-	-
N	Glass	Y	41.6 kHz	11.1%	0.6V	2.5V	-	-

Table 6.1: This table shows how an oscillator with and without ground plane responds to button pushes with different materials between the pad and the finger.

R	C	*compmin*	*compmax*
100kΩ	68pF	0.72V	2.44V

Table 6.2: The oscillator used in Table 6.1.

and theoretical values for the frequency and power usage. As can be seen the frequency is lower in real life than in the theoretic values from the program discussed in Section 5.3.2. This can be attributed to the fact that the program does not take into account the delay in the comparators. In Equation 6.1 these values have been used to calculate the delay in the comparators. This value is for this specific oscillator as different comparator values will give different overdrives leading to a different delay.

$$\frac{\frac{1}{44400} - \frac{1}{59500}}{2} = 2.86\mu s \qquad (6.1)$$

In Chapter 3 the discrepancy between the simulated and theoretical (datasheet) delay in the comparator was discussed. The result from Equation 6.1 confirms that the delay is less in real life than in the simulations, but still not as low as the Spice-model implies. It is worth noting that the comparator delay will depend on the comparator voltages as these decide how long time it takes the voltage across the capacitor to reach 100 mV overdrive. As was also noted in Section 3 the comparator does not support rail-to-rail operation, yet in the results from Table 6.2 it can be seen that the limit in the input to 1.3V below positive supply is not that strict.

Material	Thickness	Relative Permeability
Glass	560μm	7 [14]
Paper	100μm	1.4[3]

Table 6.3: Characterisitcs of the glass and paper used between the pad and the finger.

When it comes to the resulting behaviour of the circuit to a finger pushing on the tab the response seems adequate. While touching directly on the pad does reduce the frequency by 50% this is not a likely use of the button. In a real product the pad will usually be covered by either glass or some sort of plastic as Apple's iPod. Covering the pad with a 560μm thick piece of glass only reduces the frequency by 4.3% and 11.1% for the oscillator with and without a ground plane respectively. As the drop in frequency depends on the relative change in capacitance due to the finger this can most likely be improved by using a smaller capacitance in the oscillator. This has the added benefit of improving power usage if the resistance is increased so τ stays constant. It is also apparent that not having a ground plane behind the pads increases drop in frequency. The reason for this is the increased capacitance in the pad when a ground plane is present.

One final thing to note in Table 6.1 is the change in *compmax* when the finger touches the pad directly. The cause to this has not been found, but it is of interest as the decrease in *compmax* further decreases the frequency beyond the change due to the inferred capacitance.

It is also of interest to know the max frequency of the system since increased frequency will give reduced sampling time for specific bit length. In Section 6.2 it is found that the response time of the IIR-filters can be given in a number of samples, independent of the sampling period. Thus the response time of the system is closely related to the frequency of the system and the bit length assuming the sampling period is tuned so the counter makes full use of the bit length. As described in Section 3.1 the comparators limit the max frequency of the system. The characteristics of the oscillator used to find f_{max} can be viewed in Table 6.4 on the next page. In this case the comparator levels will not affect the system as the input to the comparators will be either 0V or 3.3V due to the absence of an RC-filter between the output of the SR-latch and the inputs of the comparators.

It was found in Section 3.2 that by increasing R the power usage in the RC-circuit is decreased. If this is to be done while keeping the frequency constant

R	C	*compmin*	*compmax*	f_{max}
0Ω	0F	1.27V	1.89V	121kHz

Table 6.4: The characteristics used to find maximum frequency of the oscillator and the resulting frequency of the oscillator.

the capacitance needs to be decreased so τ remains constant. Decreasing the capacitance has the negative effect of making the system less noise resistant, as was also noted in Section 3.2. It is therefore of interest to find the difference in noise from oscillators with the same τ. The three circuits tested can be seen in Table 6.5.

Characteristics			Theory		Measured Values			
C[pF]	R[kΩ]	τ[μs]	Power	Freq.	Freq.	V_{min}	V_{max}	COV
33	390	12.87	3.59μW	102	25	0.64V	2.32V	1.5×10^{-4}
47	270	12.69	5.18μW	103	26	0.64V	2.32V	2.6×10^{-4}
100	135	13.50	10.8μW	101	26	0.64V	2.32V	5.2×10^{-4}

Table 6.5: The three oscillators tested to find the difference in noise on oscillators with the same τ, but different R and C. *compmin* = 0.64V and *compmax* = 1.97V. All frequencies in kHz.

The oscillator has been connected to the digital hardware and the output sent to a computer over UART. In Figure 6.1 on the next page the output from the oscillator can be seen.

As can be seen the result has been the exact opposite of what should be expected according to the theory. Increased capacitance in the RC-circuit seems to give increased noise in the value from the Period Counter. No good reason has been found for this. It goes against both theory (Equation 1.4 on page 9) and sources[2]. If this behaviour is correct it would actually be positive though, as a lower capacitance gives better sensitvity to the capacitance induced by a finger.

This section will sum up what has been found on the power usage of the oscillator. It was found in Section 3.2 that the oscillator should oscillate between voltage levels high in the available voltage domain. This is not possible with the components chosen in this thesis. While using the comparator from Maxim has been great for simulating the behaviour of the circuit its lack of rail-to-rail operation severly limits the voltage levels available as comparator voltages. This is not a big problem in this thesis though, as the goal has

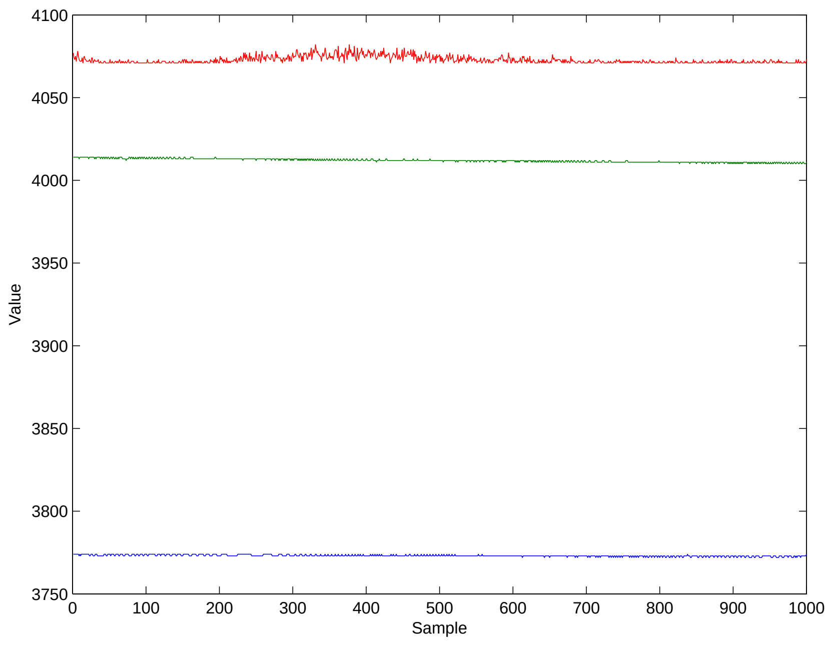


Figure 6.1: Difference in noise on the three oscillators in Table 6.5. Blue graph: $R = 390\text{k}\Omega$, green graph: $R = 270\text{k}\Omega$ and red graph: $R = 135\text{k}\Omega$.

been to identify how different parameters that affect power usage. Since comparator levels are such a parameter a unit built for production should choose comparators that have this capability.

The other main method of decreasing power usage in the oscillator is to maximize the resistance in the RC-circuit, with or without reducing the capacitance to keep the frequency constant. The power dissipated in the RC-circuit was found in Section 1.2 and is repeated in Equation 6.2. As can be seen the power is inversely dependent on the size of the resistor. The resistance should then be minimized for a specific τ in an effort to minimze the power dissipated in the resistor.

$$p(t) = i_R(t)v_R(t) = \frac{v_R(t)^2}{R} \tag{6.2}$$

6.2 Resulting behaviour of the digital circuit

When it comes to results from the digital circuit implemented on an FPGA the interesting data would be to see if the circuit responds similarily in real life as in the simulations in Chapter 4. This section will therefore get the output from the filters found to be relevant in Chapter 4. While the results in Chapter 4 where based on Verilog code simulated in ModelSim the results in this section is from actual hardware running on an FPGA with the oscillator as input to the Period Counter.

Throughout this section the digital hardware will be clocked at 8.192kHz. The sampling period will be tuned to achieve a normal value from the Period Counter that is close to the max allowed by the bit length. Since Chapter 4 found that the two IIR filters with $b = 2^{-2}$ and $b = 2^{-3}$ are of greatest interest only those two filters will be tested in this section. All the results are with the oscillator in Table 6.6 connected to the input of the Period Counter. The oscillator has no ground plane on the back of the PCB. With a finger pushed against the pad with no medium between the pad and the finger the frequency of the oscillator drops from 67kHz to 31kHz.

R	C	*compmin*	*compmax*	Freq.	Power
150kΩ	15pF	0.63V	1.97V	67kHz	9.7μW

Table 6.6: The characteristics of the oscillator used to test the hardware. Comparator levels have been measured on actual circuit and power is according to the measured frequency and the program described in Section 5.3.2.

Three different configurations of the system have been tested. One with a 12 bit counter, one with 11 bit and one with 10 bit. The goal is to see if there are any differences in number of samples used to detect the button pushes, and also to see if there is any difference in noise from the Period Counter. In this section only the data from the 12 bit counter will be presented in graphs, data from the two other bit lengths can be viewed in Appendix C.

In Figure 6.2 on the next page the response of the filters with 12 bit can be observed. The fast filter has stabilised after roughly 20 periods while the slow filter uses nearly 40 periods to stabilize. 20 periods is equal to 2124ms according to the sampling period in Table 6.8 on page 65. The slow filter uses nearly twice that to stabilize after a reset or boot-up of the system. If this is too slow depends on the boot-up time of the system the touch sensing is part of. In Figure 6.2 it can also be observed how the slow filter stabilizes at

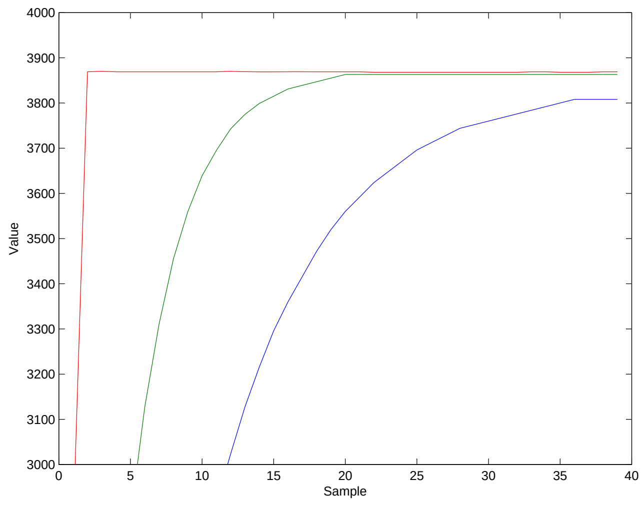


Figure 6.2: The response of the two 12 bit filters. Input is red, $IIR_{b=2^{-2}}$ is green and $IIR_{b=2^{-3}}$ is blue.

a much lower value than the fast filter due to the three LSB of the fast and slow filters being set to logic one and zero respectably.

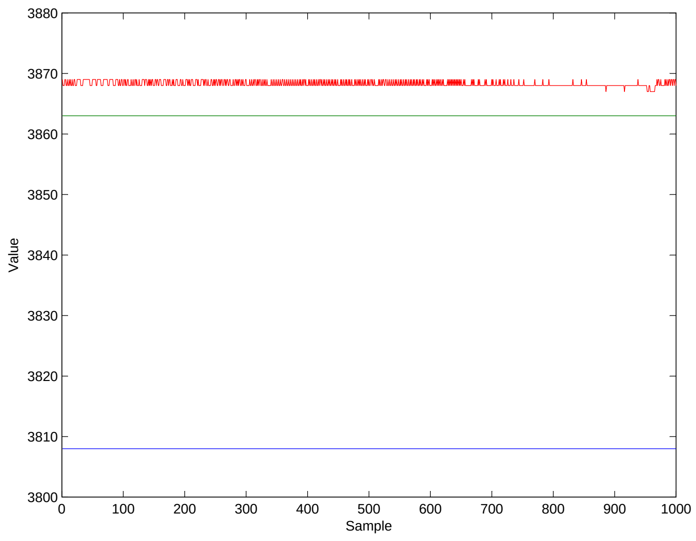


Figure 6.3: The behaviour of the 12 bit filters and input over long time.

Figure 6.3 shows the behaviour of the input and filters over long time. Note that the filters has stabilized and do not change even though the input varies. This figure shows that the filtering is more than adequate at filtering away the noise in the input signal. This is mainly due to the low level of noise in the input. It can therefore be assumed that a 12 bit counter will filter away most of the jitter in the oscillator.

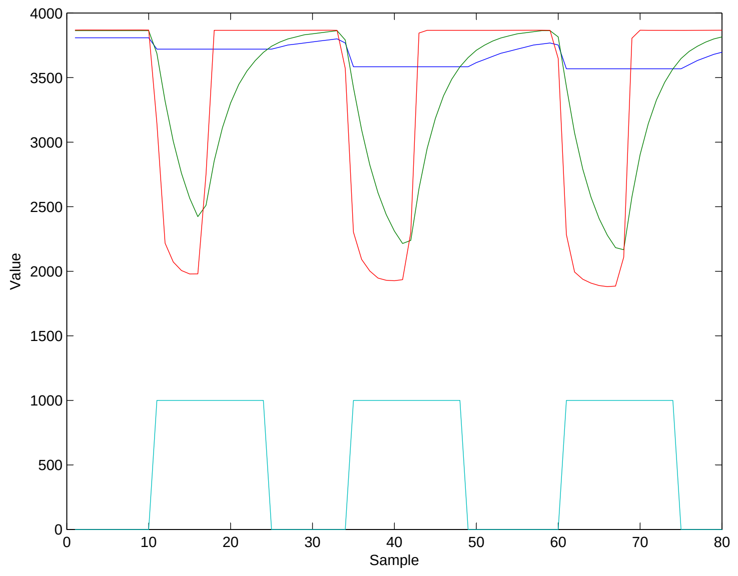


Figure 6.4: The behaviour of the 12 bit filters to button pushes. The light blue graph is the ouput from the hardware noting the button has been pushed.

In Figure 6.4 the output and input of the filters are shown when the button is pushed three times. In this sensing scheme the circuit signals that the button has been pushed when $IIR_{b=2^{-2}}$ is lower than $IIR_{b=2^{-3}}$. Note how the slow filter does not change when the button has been signaled as touched. This is done to filter out the input from finger pushes from the slow filters output as this should be an average over time. It also prevents the slow filter from falling below the fast one and signaling that the button push has enden before it indeed has.

It is of intereset to know the average response time to button pushes for the different bit lengths. In Table 6.7 on the next page ten samples has been taken on the response time of the systems. This has been done by touching the pad ten times and viewing the output to determine how many samples it takes the system to signal button push and release. The average response time has been calculated, as has the coeffecient of variation to the samples to know how stable the response time is.

Bit length												Mean	COV
10 bit	push	4	2	4	3	4	3	3	2	3	3	3.1	0.2380
	release	12	11	11	10	10	10	10	10	10	9	10.3	0.0799
11 bit	push	3	2	2	3	2	2	2	2	2	2	2.2	0.1917
	release	8	8	10	8	8	10	9	9	10	8	8.8	0.1049
12 bit	push	1	2	2	2	2	2	2	2	2	2	1.9	0.1664
	release	9	8	8	9	9	9	10	7	8	8	8.5	0.1000

Table 6.7: The digital systems response time to button pushes.

In Table 6.8 a summary of the values found from the simulations in Figure 6.2 to 6.4 can be seen. The table gives the average response time to button pushes and the level of noise on the filters. The most important thing to note her is how the average number of samples needed to notice pushes and releases sinks with increasing bit length. But as can be noted by multiplying the number of samples by the sample period the response time is shorter for 10 bit than 12 bit. E.g. the push response for 12 bit is $1.9 \cdot 111.5 = 212$ms, while it is $3.1 \cdot 27.94 = 86$ms for 10 bit. All in all the numbers indicate that reducing the bit length reduces the response time to button pushes.

When it comes to noise rejection all the filters manages to completely filter out the noise on the input. In other words, due to the reduced response time of the 10 bit filter and good noise rejection there is no reason to pick the 12 bit filter over the 10 bit version. Reducing the bit length will probably also reduce power usage in the system. This will be discussed in Section 6.3.

	COV			Response Time		
# bit	input	$IIR_{b=2^{-2}}$	$IIR_{b=2^{-3}}$	Push	Release	Sampling Period
10	0.4667	0	0	3.1	10.3	$27.94ms$
11	0.2513	0	0	2.2	8.8	$55.76ms$
12	0.1271	0	0	1.9	8.5	$111.5ms$

Table 6.8: Level of noise and response time of digital hardware with different bit length.

The results presented in this Section indicates that it is possible to decrease the bit length of the digital hardware below 10 bit. This will give better response time and a smaller circuit. As was discussed in Section 4.3.2 the response time should be less than 150 ms from the button is pushed until the general system has responded. In the case of 12 bit this is not possible with the oscillator used in this section. The systems response time can be improved either by increasing the frequency of the oscillator or decreasing the bit length of the digital hardware. Increasing the frequency of the oscillator will come at the cost of increased power usage, while decreasing the bit length will decrease size and power consumption. So the bit length should be set to the smallest value that is resistant to the noise levels from the Period Counter.

6.3 Power estimation of the digital circuit

The power used by the digital circuit must be estimated. In an effort to make data from this report comparable power has been estimated for hardware with the same filtering techniques as in Section 4.3.2. The hardware has also been simulated with the same sampling period as in Section 6.2. The input has been random values in an effort to make the switching activity in the circuit worst-case.

Comparison \ Bit length	Drain Type	10	11	12
$IIR_{b=4} < IIR_{b=8}$	Dynamic	9.1204	9.0324	9.4493
	Leakage	247.93	270.30	295.63
$IIR_{b=4} < 3600$	Dynamic	8.2925	8.7144	9.1420
	Leakage	176.37	193.12	210.90
$input < IIR_{b=4}$	Dynamic	8.2991	8.7190	9.1454
	Leakage	183.03	200.04	217.21
$input < 3600$	Dynamic	7.9428	8.3892	8.8273
	Leakage	100.46	108.52	117.71

Table 6.9: Power usage of different filtering techniques and bit length. All numbers in nW.

In Table 6.9 the power estimated by Design Vision is shown. These values give a leakage current that is between 92.7% and 96.9%, thus the leakage current far outweighs the dynamic power consumption. This is due to the low activity in the circuit. As the circuits are the same as in Section 6.2 the 12 bit circuits have a period of 111ms resulting in the registers being updated less than ten times per second.

In Figure 6.5 on the facing page the power usage of the Period Counter can be seen for a few different frequencies on the input, and either a 12, 11 or

10 bit counter. As can be seen the power consumption of the Period Counter increases linearly with the frequency. Higher bit levels also increase the power consumption, but the difference is not large. It should be noted that the Period Counter uses nearly the same amount of power as the FSM. This is due to the increased frequency of the Period Counter as well as the increased switching activity. The Period Counter updates its registers every clock period.

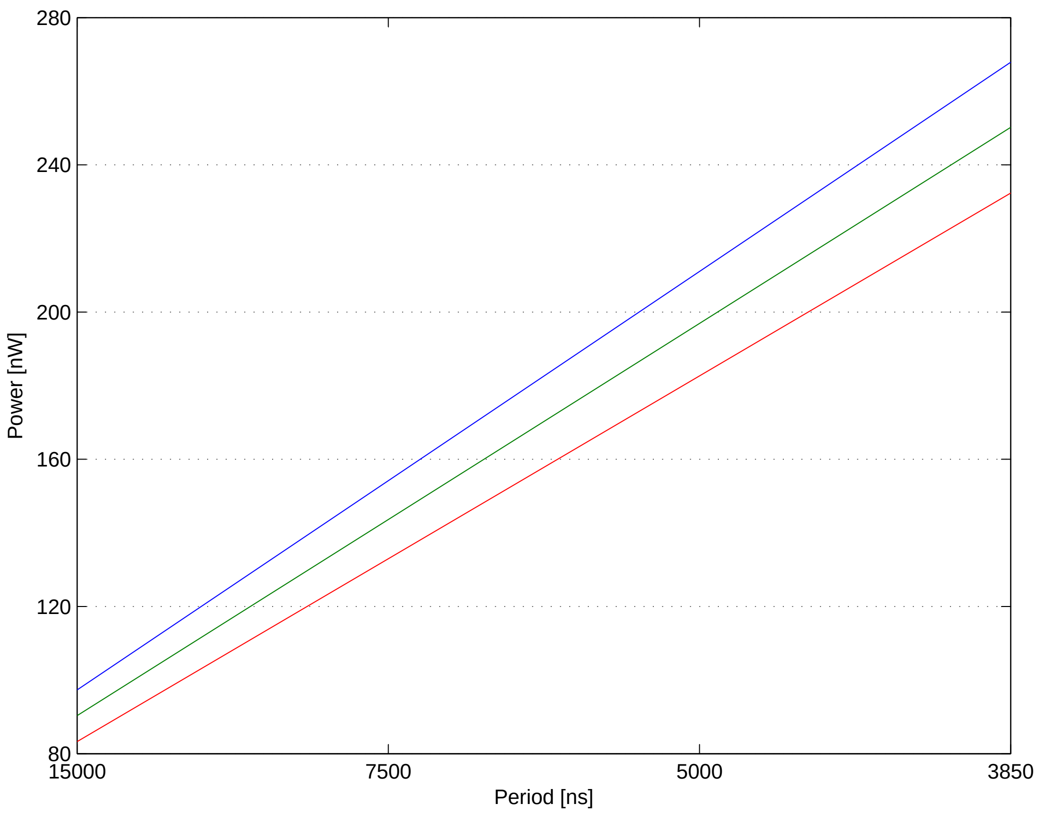


Figure 6.5: The power usage of the Period Counter with different frequency. Red line is for a 10 bit counter, green 11 bit and blue line 12 bit.

The main findings in this section is how the power usage in the digital hardware is dominated by leakage power. The difference in dynamic power for the different filtering techniques is minimal, the main reason that power is saved by switching filtering technique is the lowered leakage. It is also worth noting that the decrease in power consumption by lowering bit length is nearly constant. E.g. for $IIR_{b=2^{-2}} < IIR_{b=2^{-3}}$ the cost of adding one bit seems to be rougly 25 nW.

One final comment on the power usage of the digital circuit is the level of difference between the power usage of the digital circuit and the oscillator. All oscillators that have been discussed in this thesis has had a power usage between 3 μW and 21 μW. This is two orders of magnitude larger than the power consumption in the digital hardware. As the noise levels from the oscillator has been small with the oscillators tested in this thesis the power consumption can most likely be further lowered by increasing R with or without keeping τ the same.

6.4 Comparison with a system already on the market

In an effort to give an impression on the power usage of this module compared to others on the market a comparable module has been found. The QT100A from Quantum Research Group is a single capacitive touch button on an IC and is marketed as using 128μA on average over time with a 3V power supply resulting in a power usage of 384μW. When touched it sets its one bit output low for the duration of the touch. Its internal threshold is adjusted over time to compensate for changes from the environment. This units behaviour is then the same as the one in this thesis and should be a good benchmark for the power consumption.

Assuming the $IIR_{b=2^{-2}} < IIR_{b=2^{-3}}$ filtering is used the power consumption from the digital hardware has been found to be 248 nW for a 10 bit implementation and being lineary dependent on the bit length. The results found in Section 6.2 with regard to noise indicate that the bit length can be further lowered reducing both response time and power usage. The oscillator used together with the hardware simulated in Section 6.2 used 9.7μW and could potentially be lowered by increasing R or choosing better comparator levels. As can be noted from Table 6.5 on page 60 the comparator levels used are pretty inefficient with reference to Figure 3.9 on page 29. Finally the power usage of the comparators must be mentioned, these are listed as using 4μA resulting in a power drain of 13.2μW assuming 3.3V operation. This makes the comparators tha largest contributors to the power usage. The net power usage of the capacitive touch sensing scheme used in the last two sections is then $0.248 + 9.7 + 2 \cdot 13.2 = 36\mu$W which is an order of magnitude less than the solution from Quantum Research Group. As discussed in this paragraph there is also a potential for lowering this further.

Chapter 7

Conclusions and future work

The goal of this thesis has been to design a functional low power capacitive touch sensing system with focus on identifying which design choices affect response time, noise rejection and power usage. It has been found that the RC-circuit uses far more power than the digital hardware. Thus the comparator levels and RC-circuit in the oscillator are the characteristics that has the most impact on power usage. In the digital circuit noise rejection can be improved by applying the right filtering technique and response time can be shortened by reducing the bit length of the system. The results presented stands to confirm that the proposed design functions as specified. It has a response time that allows user interaction without perceived delay and the power usage is more than competitive with designs currently on the market.

One important aspect is how much the different parts of the sensing scheme contribute to the final power usage. The comparators and oscillator have power usage levels that are two orders of magnitude larger than the digital circuit. In other words, how the digital circuit is designed does not affect power usage to any significant degree. The digital circuit should therefore be optimized for response time and noise rejection while the oscillator is optimized with power consumption in mind.

7.1 Future work

Future work on this design should focus on further optimizing the power consumption of the oscillator. A good start would be to use comparators with rail-to-rail operation and test how the oscillator functions when it oscillates between voltages high in the voltage domain. How low the capacitance in the RC-circuit can be without experiencing large levels of noise could also be of interest. This would allow the resistance to be set as high as possible for a specific frequency. When an oscillator has been designed focus should shift to reducing the bit length of the digital hardware to a level that operates with acceptable noise rejection.

Bibliography

[1] Asad A. Abildi and Robert G. Meyer. *Noise in Relaxation Oscillators.* IEEE, 1983.

[2] Larry K. Baxter. *Capacitive Sensors Design and Applications.* Wiley-Blackwell, 1997.

[3] Jens Borch, M. Bruce Lyne, Richard E. Mark, and Charles Habeger. *Handbook of Physical Testing of Paper.* CRC Press, 2001.

[4] James R. Dabrowski and Ethan V. Munson. Is 100 milliseconds too fast? *Short Talks*, 2001.

[5] International Roadmap Committee. *2001 ITRS, Highlights and Challenges*, 1997.

[6] Cypress Semiconductor Corp. Mark Lee. *The art of capacative tough sensing.* http://www.planetanalog.com/features/showArticle.jhtml?articleID=181401898, 2006.

[7] Maxim Integrated Circuits. *MAX921-MAX924 Datasheet.*

[8] National Semiconductor. *LMC7211 Datasheet.*

[9] James W. Nilsson and Susan A. Riedel. *Electric Circuits, 5th edition.* Addison-Wesley Publishing Company, fifth edition, 1996.

[10] ON Semiconductor. *MC14043B & MC14044B Datasheet.*

[11] Massoud Pedram and Jan M. Rabaey. *Power Aware Design Methodologies.* Kluwer Academic Publishers, 2002.

[12] Texas Instruments. *TLC193 & TLC393 Datasheet.*

[13] Microchip Technology Inc. Tom Perme. *Introduction to Capacitive Sensing.*

[14] Microchip Technology Inc. Tom Perme. *Layout and Physical Design Guidelines for Capacitive Sensing.*

[15] Wikipedia.org. *Wien Bridge Oscillator.* http://en.wikipedia.org/wiki/Wien_bridge accessed 22.05.

[16] Xilinx, Inc. *Metastability Considerations*, 1997.

Appendix A

C program describing the RC-circuit

Listing A.1: C code of program for simulationg behaviour of RC-circuit.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>



//Config
#define NSAMPLES                                        10000000
            //Max number of samples. Defines length of arrays.
    Should be large.
#define SUPPLYVOLTAGE               3.3                          //
    Voltagelevel of powersupply


double samples;
double Vdd;
double R;
double C;
double tau;
double Vmin;
double Vmax;

double vt[NSAMPLES];
double it[NSAMPLES];
double pt[NSAMPLES];


```

```
void init()      //Initializes the values according to the config above
{
        samples = NSAMPLES;
        Vdd = SUPPLYVOLTAGE;
        tau = R * C / 1000000;
        tau = tau / 1000000;


}


//Calculates the voltage over the capacitor and current through the resistor during one period.
//Also calculates the powerdissipation in the resistor
void calcResponse(double min,double max)
{
        double t = 0;
        double exp = 0;
        int turn = 0;
        int i = 0;

        double e = M_E;

        //Iterates over the samples during charging of the capacitor
        for(i = 0; i < samples; i++)
        {
                t = (double)i/1000000000;        //Time is incremented in steps of nanoseconds
                exp = pow(e,-(t/tau));
                vt[i] = Vdd + (min - Vdd) * exp;
                pt[i] = (Vdd-vt[i])*(Vdd-vt[i])/R;
                if(vt[i] > max) //We've reached the high level, calculate last values and break
                {
                        turn = i;
                        break;
                }
        }

        //Calculates the discharging of the capacitor
        for(i = turn + 1; i < samples; i++)
        {
                t = (double)(i-turn)/1000000000;        //Time is reset to one and incremented in nanoseconds
                exp = pow(e,-(t/tau));
                vt[i] = max * exp;
                pt[i] = 0.0;
                if(vt[i] < min)                         //We've reached the low level, calculate last
```

```
                    values and break
                {
                        vt[i+1] = 0.0;
                        break;
                }

        }

}

//Calculates the energy needed when the capacitor is charged and
    decharged
//Returns the value in nanowatt
double calcEnergy(){

        double tmp = 0;

        int i;
        for(i = 0; i < samples; i++)
        {
                tmp = tmp + pt[i];
                if(pt[i] == 0.0){break;}
        }

        return tmp;     //Nanowatts!!!

}

//Outputs the energy needed for one cycle
void outputEnergy()
{
        int i;
        int j;
        double low;
        double high;
        double energy = 0;
        for(i = 0; i-1 < 33; i++)
        {
                low = (double)i * (Vdd / 33.0);
                for(j = 0; j-1 < 33; j++)
                {
                        if(i < j && i != 0 && j != 33)
                        {
                                high = (double)j * (Vdd / 33.0);
                                calcResponse(low,high);
                                energy = calcEnergy();
                                printf("%f␣",energy);
                        }
                        else
                        {
                                printf("0.000000␣");
                    }
            }
            printf("\n");
        }
}
//Calculates the duty cycle by iterating through the response and
    finding the turning and ending point
double calcDutyCycle()
{
        int i;
        int turn = 0;
        int size;
        for(i = 0; i < samples; i++)
        {
                if(vt[i+1] < vt[i] && turn == 0)
                {
                        turn = i;
                }
                if(vt[i] == 0.0 && i > 5)
                {
                        break;
                }
                size = i;
        }
        return (double)turn / (double)size;
}

//Calculates the duty cycle for all combinations of compmin/
    compmax and outputs them
void outputDutyCycle()
{
        int i;
        int j;
        double low;
        double high;
        double dc;
        for(i = 0; i-1 < 33; i++)
        {
                low = (double)i * (Vdd / 33.0);
                for(j = 0; j-1 < 33; j++)
                {
                        if(i < j && i != 0 && j != 33)
                        {
                                high = (double)j * (Vdd / 33.0);
                                calcResponse(low, high);
                                dc = calcDutyCycle();
                                printf("%f ",dc);
                        }
                        else
                        {
                                printf("0.000000 ");
                }
            }
            printf("\n");
        }
}

//Calculates the frequency by counting the number of nanosecond iterations done by calcResponse()
double calcFrequency()
{
        int i;
        for(i = 0; i < samples; i++)
        {
                if(vt[i] == 0.0 && i > 5)
                {
                        break;
                }
        }
        return 1000000000 / (double)i;
}

//Outputs the frequency for a specific RC-cicruit and all combinations of compmin/compmax
void outputFrequency()
{
        int i;
        int j;
        double low;
        double high;
        double freq;
        for(i = 0; i-1 < 33; i++)
        {
                low = (double)i * (Vdd / 33.0);
                for(j = 0; j-1 < 33; j++)
                {
                        if(i < j && i != 0 && j != 33)
                        {
                                high = (double)j * (Vdd / 33.0);
                                calcResponse(low, high);
                                freq = calcFrequency();
                                if(freq < 250000)
                                {
                                        printf("%f ",freq);
                                }
                                else
                                {
                                        printf("250000 ");//Cap the output at 250000 to make more readable graphs
                                }
                        }
                        else
                        {
                                printf("0.000000 ");//Output 0 for
                                    oscillations when compmin >
                                    compmax
                        }
                }
                printf("\n");
        }
}

//Calculate power usage by multiplying the frequency by the energy
    used per oscillation
double calcEnergySec()
{
        return calcFrequency() * calcEnergy();  //Nanowatt!!
}

//Outputs the power usage for a specific RC-cicruit and all
    combinations of compmin/compmax
void outputEnergySec()
{
        int i;
        int j;
        double low;
        double high;
        double energy;
        for(i = 0; i-1 < 33; i++)
        {
                low = (double)i * (Vdd / 33.0);
                for(j = 0; j-1 < 33; j++)
                {
                        if(i < j && i != 0 && j != 33)
                        {
                                high = (double)j * (Vdd / 33.0);
                                calcResponse(low,high);
                                energy = calcEnergySec();
                                if(energy < 350000)
                                {
                                        printf("%f ",energy);
                                }
                                else
                                {
                                        printf("350000 ");
                                }
                        }
                        else
                        {
                                printf("0.000000 ");
                        }
                }
                printf("\n");
        }
}

//Outputs the frequency and power usage of an RC-circuit as the
    capacitance increases
void outputFreqandWattCap(double low, double high, double startC,
    int steps, double startR)
{

        C = startC;
        R = startR;

        int i;
        for(i=0;i<steps;i++)
        {
                calcResponse(low, high);
                init();
                printf("%g %g\n",calcFrequency(),calcEnergySec());
                C = C++;
        }


}

//Outputs the frequency and power usage of an RC-circuit as the
    resistance increases
void outputFreqandWattRes(double low, double high, double startR,
    int steps, double startC)
{

        C = startC;
        R = startR;

        int i;
        for(i=0;i<steps;i++)
        {
                calcResponse(low, high);
                init();
                printf("%g %g\n",calcFrequency(),calcEnergySec());
                R = R + 1000;
        }


}


int main(int argc,char* argv)
{

        int type = 0;
        char ord[20];
        double min;
        double max;
        int steps;


        printf("Single oscillator, all comParator levels, sweep of
            R or sweep of C? (s/p/r/c)\n");
        while(type == 0)
        {
                scanf("%s",ord);
                if('s' == ord[0])
                {
                        type = 1;
                }
                else if('p' == ord[0])
                {
                        type = 2;
                }
                else if('r' == ord[0])
                {
                        type = 3;
                }
                else if('c' == ord[0])
                {
                        type = 4;
                }
                else
                {
                        type = 0;
                }
        }

        if(type == 1)
        {
                printf("Capacitance? (in pF)\n");
                scanf("%s",ord);
                C = atof(ord);

                printf("Resistance? (in kOhm)\n");
                scanf("%s",ord);
                R = 1000 * atof(ord);

                init();

                printf("compmin? (in Volt)\n");
                scanf("%s",ord);
                min = atof(ord);

                printf("compmax? (in Volt)\n");
                scanf("%s",ord);
                max = atof(ord);

                calcResponse(min,max);
                printf("Min: %g, Max: %g\n",min,max);
                printf("C: %g pF, R: %g Ohm\n",C,R);
                printf("Frequency       : %f \n", calcFrequency())
                    ;
                printf("Dutycycle       : %f \n", calcDutyCycle())
                    ;
                printf("Energy/cycle    : %f \n", calcEnergy());
                printf("Avg. nwatt      : %f \n", calcEnergySec())
                    ;
        }
        else if(type == 2)
        {
                printf("Capacitance? (in pF)\n");
                scanf("%s",ord);
                C = atof(ord);

                printf("Resistance? (in kOhm)\n");
                scanf("%s",ord);
                R = 1000 * atof(ord);

                init();

                printf("1 - Output Energy\n");
                printf("2 - Output Duty Cycle\n");
                printf("3 - Output Frequency\n");
                printf("4 - Output Energy pr Second\n");

                scanf("%s",ord);

                switch(atoi(ord))
                {
                        case 1 :
                                printf("Energy usage pr cycle for 
                                    low/high from 0V to 3.3V:\n");
                                outputEnergy();
                                break;
                        case 2 :
                                printf("Duty Cycle for low/high 
                                    from 0V to 3.3V:\n");
                                outputDutyCycle();
                                break;
                        case 3 :
                                printf("Frequency for low/high 
                                    from 0V to 3.3V:\n");
                                outputFrequency();
                                break;
                        case 4 :
                                printf("Energy usage pr second for low/high from 0V to 3.3V:\n");
                                outputEnergySec();
                                break;
                        default : printf("You have to choose 1-4!\n");
                }
        }
        else if(type == 3)
        {
                printf("Capacitance? (in pF)\n");
                scanf("%s",ord);
                C = atof(ord);

                printf("Starting resistance? (in kOhm)\n");
                scanf("%s",ord);
                R = 1000 * atof(ord);

                printf("Number of steps of 1000kOhm to simulate\n");
                scanf("%s",ord);
                steps = atof(ord);

                init();

                printf("compmin? (in Volt)\n");
                scanf("%s",ord);
                min = atof(ord);

                printf("compmax? (in Volt)\n");
                scanf("%s",ord);
                max = atof(ord);

                printf("Frequency and power usage of the circuit\n");
                outputFreqandWattRes(min,max,R,steps,C);

        }
        else if(type == 4)
        {
                printf("Resistance? (in kOhm)\n");
                scanf("%s",ord);
                R = 1000 * atof(ord);

                printf("Starting capacitance? (in pF)\n");
                scanf("%s",ord);
                C = atof(ord);

                printf("Number of steps of 1pF to simulate\n");
                scanf("%s",ord);
```

```
                    steps = atof(ord);

                    init();

                    printf("compmin? (in Volt)\n");
                    scanf("%s",ord);
                    min = atof(ord);

                    printf("compmax? (in Volt)\n");
                    scanf("%s",ord);
                    max = atof(ord);

                    printf("Frequency and power usage of the circuit\n
                        ");
                    outputFreqandWattCap(min,max,C,steps,R);

          }


          return 0;
}
```

Appendix B

Verilog code

Listing B.1: Verilog code of the period counter

```
module asynccounter(in, reset, zero, value);
    input in, reset, zero;
    output [11:0] value;
    reg [11:0] value;

    always @(posedge in, negedge reset, posedge zero)
    begin
        if(reset == 0)
           value = 0;
        else if(zero == 1)
           value = 0;
        else
           value = value + 1;
    end
endmodule
```

Listing B.2: Verilog code of the period counter

```
module counter(clk, value, reset, last_count,  button_press, zero)
    ;
        input clk, reset;
        input [11:0] value;
        output reg [11:0] avg_count;
        output reg [11:0] last_count_lowpass;
        output reg [11:0] last_count;
        output reg button_press, zero;
        reg [7:0] teller;


        parameter S_wait = 0, S_setup1 = 1, S_setup2 = 2, S_read =
            3, S_calc = 4, S_idle = 5;

        reg [3:0] State, StateNext;
        reg [11:0] tmp_meta_count;


        always @(*)
        begin
            StateNext = State;
            case(State)
                S_wait: begin
                    if(teller == 225) begin
                        StateNext = S_setup1;
                    end
                end
                S_setup1:begin
                    StateNext = S_setup2;
                end
                S_setup2: begin
                    StateNext = S_read;
                end
                S_read: begin
                    StateNext = S_calc;
                end
                S_calc: begin
                    StateNext = S_idle;
                end
                S_idle: begin
                    StateNext = S_wait;
                end
                default: begin
                    StateNext = S_idle;
                end
            endcase
        end


        always @(posedge clk, negedge reset) begin
            if(reset == 0) begin
                State <= S_wait;
                last_count_lowpass <= 0;
                avg_count <= 0;
                button_press <= 0;
                tmp_meta_count <= 0;
                last_count <= 0;
                teller <= 0;
                zero <= 0;

            end
            else begin

                case(State)
                    S_wait: begin
```

```
                                //Waits until next sampling period
                end
                S_idle: begin
                    teller <= 0;//Resets the counter
                end

                S_read: begin//Update the two filters
                    last_count_lowpass <= ((last_count_lowpass >> 2) + (last_count_lowpass >> 1)) + (last_count >> 2);
                    last_count_lowpass[0] <= 1;
                    last_count_lowpass[1] <= 1;
                    last_count_lowpass[2] <= 1;

                    last_count_lowpass <= (last_count >> 1) + (tmp1 >> 1) + (tmp2 >> 2) + (tmp >> 2) + (tmp3 >> 3);
                 if(button_press == 0)begin
                        avg_count <= (avg_count - (avg_count >> 3)) + (last_count >> 3);
                        avg_count[0] <= 0;
                        avg_count[1] <= 0;
                        avg_count[2] <= 0;

                    end
                    zero <= 1;               //Resets the period counter
                end
                S_calc: begin
                    if(last_count_lowpass < avg_count)
                        button_press <= 1;                //Button push detected
                    else
                        button_press <= 0;                //No button push this period
                    zero <= 0;
                end
                default: begin
                end
            endcase

            teller <= teller + 1;              //The counter is always updated

            if(State == S_setup1 || State == S_setup2)begin   //To avoid metastability
                tmp_meta_count <= value;
                last_count <= tmp_meta_count;
            end
```

```



                        State <= StateNext;                 //Updates
                            the state
                    end
                end

endmodule
```

Appendix C

Resulting behaviour of filters with 11 and 10 bit

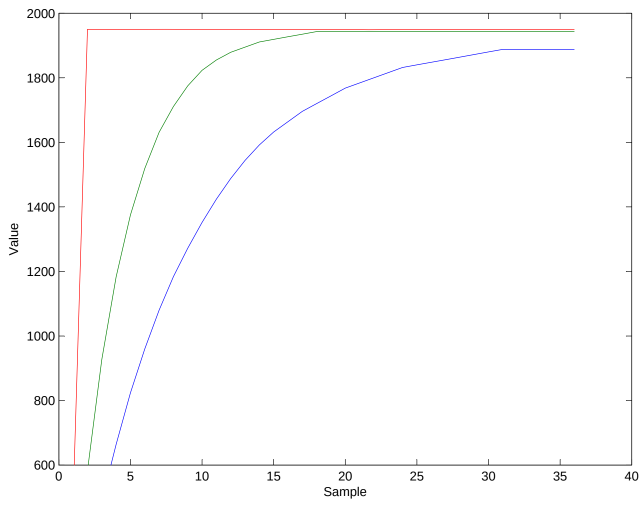


Figure C.1: The response of the two 11 bit filters. Input is red, $IIR_{b=2^{-2}}$ is green and $IIR_{b=2^{-3}}$ is blue.

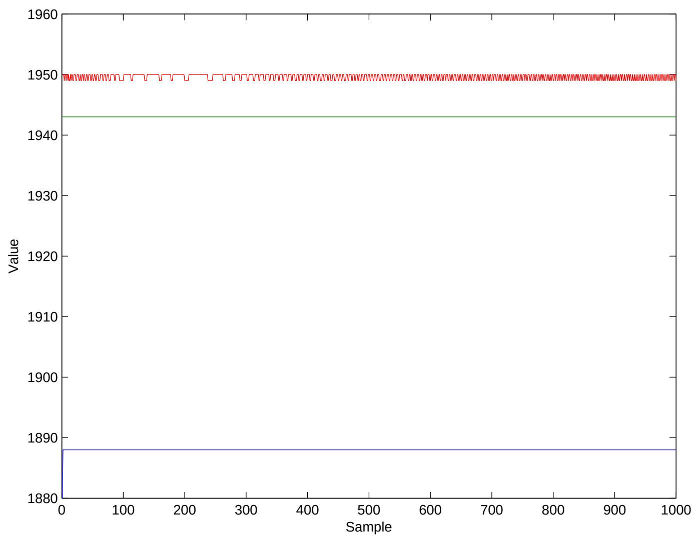


Figure C.2: The behaviour of the 11 bit filters and input over long time.

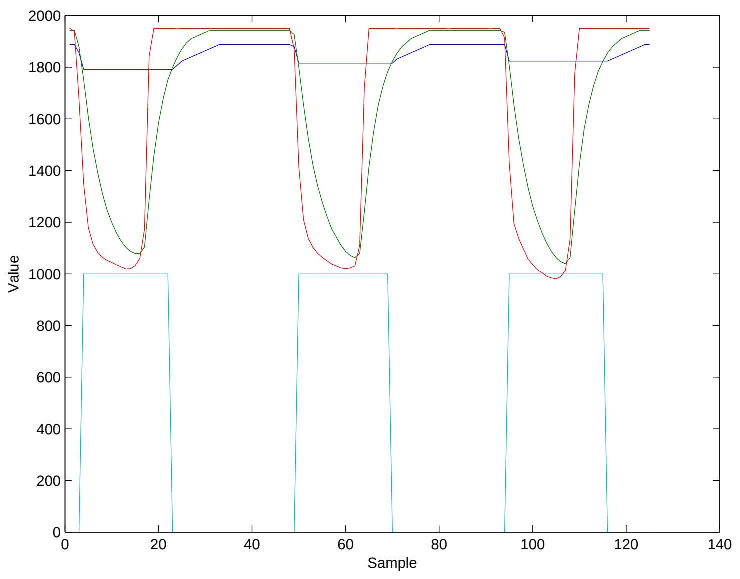


Figure C.3: The behaviour of the 11 bit filters to button pushes. The light blue graph is the ouput from the hardware noting the button has been pushed.

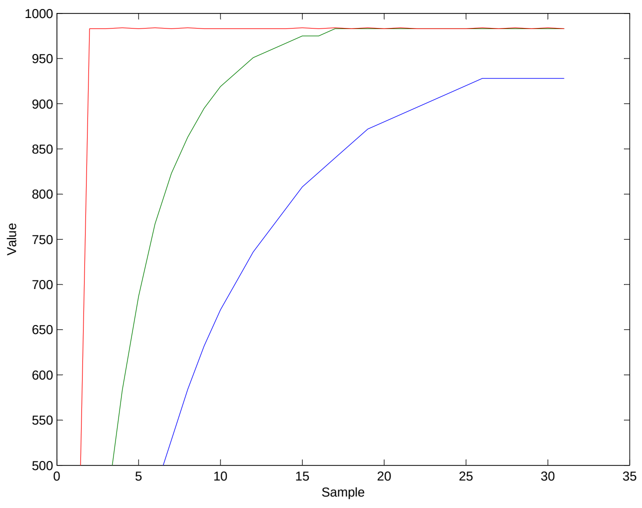


Figure C.4: The response of the two 10 bit filters. Input is red, $IIR_{b=2^{-2}}$ is green and $IIR_{b=2^{-3}}$ is blue.

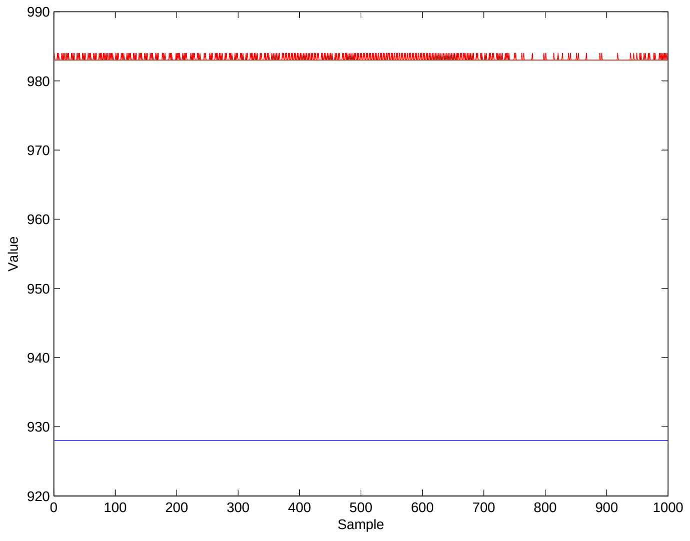


Figure C.5: The behaviour of the 10 bit filters and input over long time.

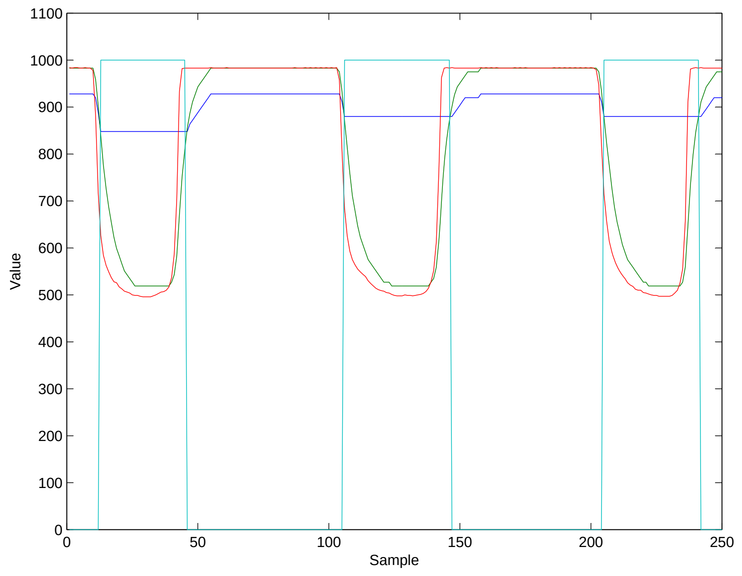
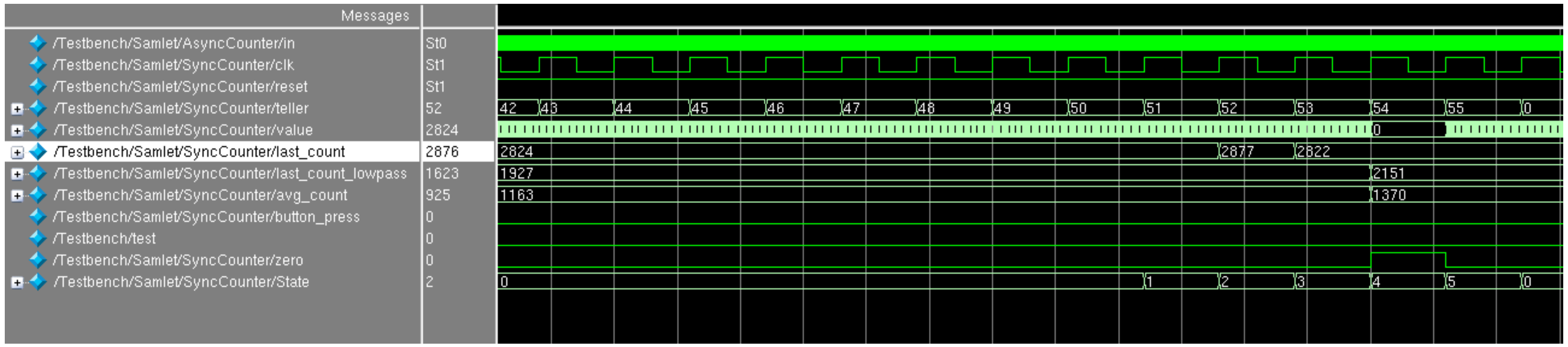


Figure C.6: The behaviour of the 10 bit filters to button pushes. The light blue graph is the ouput from the hardware noting the button has been pushed.

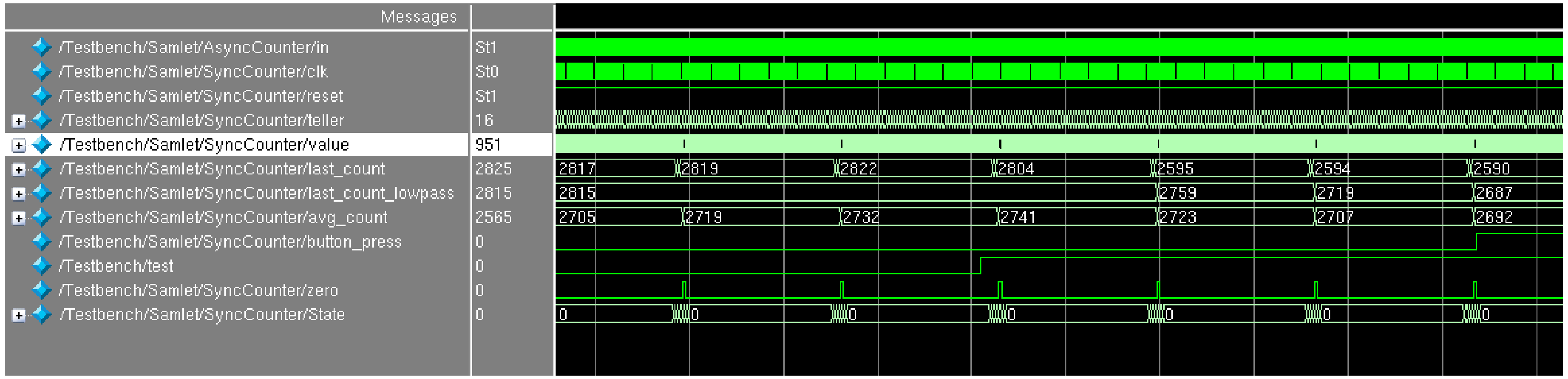
Appendix D

Simulations of the hardware from ModelSim



Messages	
/Testbench/Samlet/AsyncCounter/in	St0
/Testbench/Samlet/SyncCounter/clk	St1
/Testbench/Samlet/SyncCounter/reset	St1
/Testbench/Samlet/SyncCounter/teller	52
/Testbench/Samlet/SyncCounter/value	2824
/Testbench/Samlet/SyncCounter/last_count	2876
/Testbench/Samlet/SyncCounter/last_count_lowpass	1623
/Testbench/Samlet/SyncCounter/avg_count	925
/Testbench/Samlet/SyncCounter/button_press	0
/Testbench/test	0
/Testbench/Samlet/SyncCounter/zero	0
/Testbench/Samlet/SyncCounter/State	2

Figure D.1: Simulation showing one sample period of the hardware.

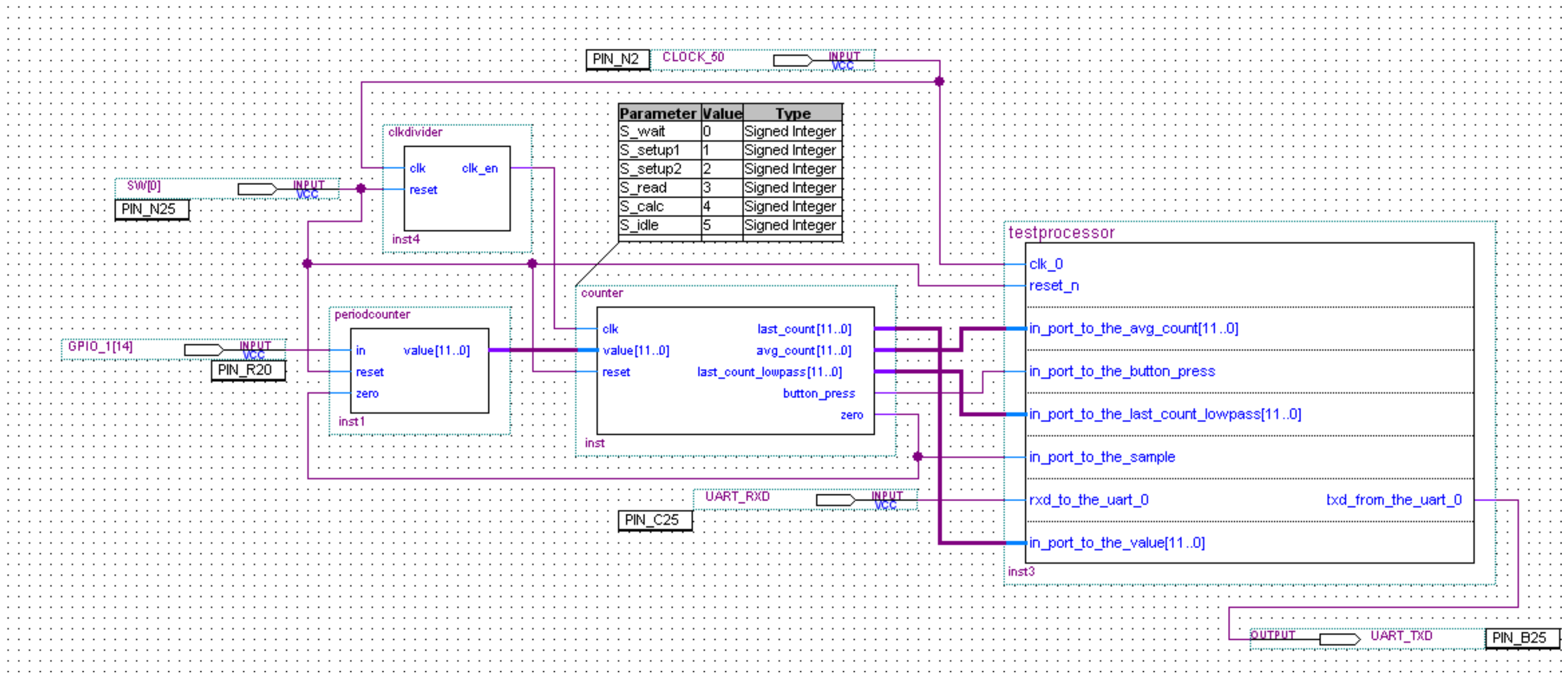


```
Messages
/Testbench/Samlet/AsyncCounter/in                      St1
/Testbench/Samlet/SyncCounter/clk                      St0
/Testbench/Samlet/SyncCounter/reset                    St1
/Testbench/Samlet/SyncCounter/teller                   16
/Testbench/Samlet/SyncCounter/value                    951
/Testbench/Samlet/SyncCounter/last_count               2825   2817  2819  2822  2804  2595  2594  2590
/Testbench/Samlet/SyncCounter/last_count_lowpass       2815   2815                    2759  2719  2687
/Testbench/Samlet/SyncCounter/avg_count                2565   2705  2719  2732  2741  2723  2707  2692
/Testbench/Samlet/SyncCounter/button_press             0
/Testbench/test                                        0
/Testbench/Samlet/SyncCounter/zero                     0
/Testbench/Samlet/SyncCounter/State                    0      0     0     0     0     0     0     0
```

Figure D.2: Simulation showing several sample periods of the hardware. Also shows detection of button push. The signal *test* indicates that a button push has been simulated by decreasing the frequency.

Appendix E

Design implemented on the FPGA



Parameter	Value	Type
S_wait	0	Signed Integer
S_setup1	1	Signed Integer
S_setup2	2	Signed Integer
S_read	3	Signed Integer
S_calc	4	Signed Integer
S_idle	5	Signed Integer

Figure E.1: The hardware implemented on the FPGA.

Appendix F

Code running on Nios II

Listing F.1: Code running on Nios II

```
#include "sys/alt_stdio.h"
#include "stdio.h"
#include "time.h"
#include "system.h"
#include "io.h"

int main()
{
    int avg_count;
    int last_count_lowpass;
    int last_count;
    int button_press;
    int zero;
    while(1){
        zero = IORD(SAMPLE_BASE,0); //Reads zero output from the
            FSM
        while(zero == 0){zero = IORD(SAMPLE_BASE,0);}
        while(zero == 1){zero = IORD(SAMPLE_BASE,0);}//Waits until
             after a positive edge
        avg_count = IORD(AVG_COUNT_BASE,0);
        last_count_lowpass = IORD(LAST_COUNT_LOWPASS_BASE,0);
        last_count = IORD(VALUE_BASE,0);
        button_press = IORD(BUTTON_PRESS_BASE,0);     //Reads all
            values
        printf("%d %d %d %d\r\n",avg_count,last_count_lowpass,
            last_count,1000*button_press);
    }

  return 0;
}
```

Appendix G

Standard deviation of samples in Figure 4.2

SD/Time	500	750	1000	1500	2000	2500	3000	4000	5000	10000
1	0,5002	0,5001	0,4990	0,5001	0,5001	0,5001	0,4998	0,5002	0,5002	0,5000
2	0,4985	0,5002	0,5001	0,5002	0,4993	0,5001	0,5080	0,5101	0,5160	0,6497
3	0,5001	0,5000	0,5021	0,5021	0,5218	0,5274	0,5606	0,6086	0,6712	0,9029
5	0,5032	0,5217	0,5403	0,6149	0,7059	0,7465	0,8379	0,9114	1,0377	1,4175
7	0,5533	0,6261	0,7070	0,7915	0,9213	1,0540	1,1538	1,2924	1,4293	1,9667
10	0,6960	0,8394	0,9470	1,1265	1,2741	1,4243	1,5483	1,7532	2,1087	2,8365
15	0,9829	1,2214	1,3356	1,6784	1,8103	2,1770	2,3250	2,6842	2,9078	4,2206
25	1,6459	1,9708	2,2758	2,7625	3,2171	3,6576	3,9521	4,5294	5,0902	7,2224
35	2,3702	2,8741	3,1723	3,8695	4,5365	5,0192	5,4297	6,2554	6,9411	9,6867
50	3,2247	4,0320	4,3041	5,6820	6,5371	7,0780	7,8395	9,1125	10,2585	14,1814

Table G.1: This table contains the standard deviation of the sampels in Figure 4.2 on page 35. As can be seen the standard deviation with a sampling period of 10000 is larger than for a sampling period of 500. But as can be observed in Figure 4.2 the COV indicates that the variations realtive to the size of the mean is smaller.