Oscilloskopprojekt



Teknisk Rapport

19-06-2017

Version 1.4.6

Denne opgave er udarbejdet af HOLD A og HOLD B

|  |  |  |  |
| --- | --- | --- | --- |
| HOLD A (Oscilloskop) | | HOLD B (Signalgeneratoren) | |
|  | Nicholas |  | Mathias |
|  | Christian |  | Christian |

30082-Projektarbejde i Digitaldesign



Indholdsfortegnelse

[1 Indledning 4](#_Toc485635124)

[2 Baggrund 4](#_Toc485635125)

[3 Opgaven 4](#_Toc485635126)

[4 Kravspecifikationer 4](#_Toc485635127)

[5 Design 4](#_Toc485635128)

[6 Oscilloskop 4](#_Toc485635129)

[6.1 Analog sampling 4](#_Toc485635130)

[6.2 UART TX/RX i mcu’en 4](#_Toc485635131)

[6.2.1 Modtagelse og Transmit 4](#_Toc485635132)

[6.2.2 Problemer med polling baseret transmit 4](#_Toc485635133)

[6.3 Labview protokol 4](#_Toc485635134)

[6.4 Signal generator protokol 4](#_Toc485635135)

[6.5 Checksum 4](#_Toc485635136)

[6.5.1 Zero16 4](#_Toc485635137)

[6.5.2 LRC8 4](#_Toc485635138)

[6.5.3 CRC16 4](#_Toc485635139)

[6.6 Bodeplot 4](#_Toc485635140)

[6.7 Signalgenerator 4](#_Toc485635141)

[6.7.1 Modulet SigGenControl 4](#_Toc485635142)

[6.7.2 Modulet SigGenDataPath 4](#_Toc485635143)

[6.7.3 Modulet SevenSeg5 4](#_Toc485635144)

[6.7.4 Modulet “DivClk” 4](#_Toc485635145)

[6.8 Indstilling af parametrene i LabView 4](#_Toc485635146)

[6.9 SPI 4](#_Toc485635147)

[6.10 Labview 4](#_Toc485635148)

[7 Implementering 4](#_Toc485635149)

[7.1 Oscilloskop 4](#_Toc485635150)

[7.1.1 C-blok diagram 4](#_Toc485635151)

[7.1.2 Dokumentation af c-moduler 4](#_Toc485635152)

[7.1.3 Databufferen 4](#_Toc485635153)

[7.1.4 Interrupt diagram 4](#_Toc485635154)

[7.1.5 Analog sampling 4](#_Toc485635155)

[7.2 Signal generator 4](#_Toc485635156)

[7.2.1 SigGenDataPath 4](#_Toc485635157)

[7.2.2 SigGenControl 4](#_Toc485635158)

[7.2.3 Shiftreg 4](#_Toc485635159)

[7.2.4 ReceiveDet 4](#_Toc485635160)

[7.2.5 ProControl 4](#_Toc485635161)

[7.3 Labview 4](#_Toc485635162)

[7.3.1 On/Off indikator 4](#_Toc485635163)

[7.3.2 Record length error 4](#_Toc485635164)

[7.3.3 Ekstra waveform 4](#_Toc485635165)

[7.4 Bodeplot 4](#_Toc485635166)

[8 Test 4](#_Toc485635167)

[8.1 Oscilloskop 4](#_Toc485635168)

[8.2 Signalgennerator 4](#_Toc485635169)

[8.2.1 Analog Filtre 4](#_Toc485635170)

[8.3 SPI Test 4](#_Toc485635171)

[8.3.1 Test af sikker SPI forbindelse ved modtagelse 4](#_Toc485635172)

[8.4 Test af “indstilling af parametre fra Labview Programmet” 4](#_Toc485635173)

[8.5 Test af VHDL koden 4](#_Toc485635174)

[8.6 Warnings når der Synthesizes 4](#_Toc485635175)

[8.7 Test af Bodeplot 4](#_Toc485635176)

[9 Konklusion 4](#_Toc485635177)

[10 Appendiks 4](#_Toc485635178)

[11 Flowchart over ProControl Tilstandsmaskine 4](#_Toc485635179)

# Indledning

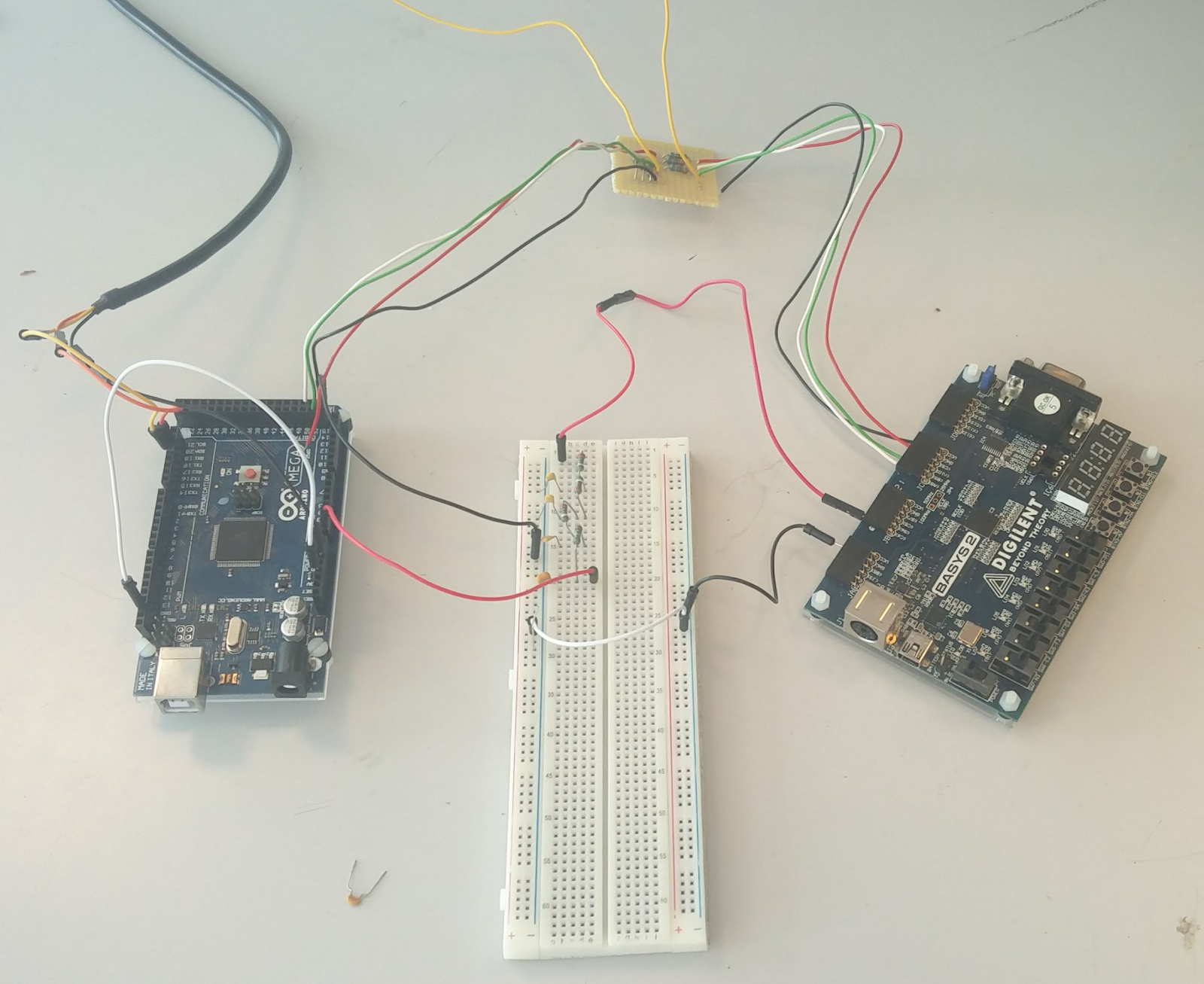
Formålet med dette projekt er at lave et system som består af et oscilloskop og en signalgenerator. Det som vores system skal bruges til, er at kunne generere og måle på forskellige typer signaler - med forskellige amplitude og frekvens. Vores kode er delt op i C kode og VHDL kode. Koden er skrevet modulbaseret, hvilket gør at kodestumper/moduler kan bruges til andre spændende projekter som indebærer nogle af de elementer som projektet har bestået af fx indenfor datakommunikation, protokoller, databehandling og signal generering. Med projektoplæget har gruppen fået udleveret allerede skrevet kode, både til FPGA’en og Labview for, at kunne komme i gang.

# Baggrund

I kurset “30082 Projektarbejde i digitaldesign” har gruppen skulle fremstille et digitalt design som skal udgøre et oscilloskop system med signalgenerator. Gruppen har i udarbejdelsen af systemet anvendt følgende værktøjer:

* Atmel Studio - Til C kode
* Xilinx ISE Design Suite - Til VHDL kode
* Basys 2 FPGA board
* Arduino MEGA 2560
* Picoscope
* Oscilloscopet i El-Lab
* LT-Spice

På nedenstående billede Figur 1, ses vores samlede system, her kan man se FPGA, MCU, SPI Shield (veroboard) og filteret (breadboard).



Figur 1 Billede af det fysiske produkt

# Opgaven

Denne gruppe har ved hjælp af givent materiale, lavet et system med flere funktioner. Systemet skal kunne generer forskellige signalformer. Signalformer som Trekant, Sinus, Konstant og savtakket, skal kunne vælges efter preference. Det skal også være muligt at kunne styre systemet i Labview fra computeren, og håndtere forskellige kommunikations protokoller. Nogle af signalerne skal også konverteres fra analog til digital før der kan arbejdes med dem i Atmel studio.

# Kravspecifikationer

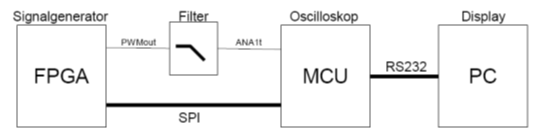
På nedenstående tabel ses funktionskrav til systemet. Vores gruppe har selv tilføjet nogle funktioner da de var relevante.

|  |  |  |  |
| --- | --- | --- | --- |
| **Oscilloskop** | | Verifikation | Krav opfyldt |
| ADC konvertering | Den analoge spænding fra signalgeneratoren skal måles fra 0 til 3.3 V med en opløsning på 8 bit. | **Info\*** | **JA** |
| Dataintegritet | Oscilloskopet skal ved alle nedenstående indstillinger kunne køre med kontinuert ubrudte målinger | **Test\* og Analyse** | **JA** |
| Parametre | Oscilloskopets samplerate og Record length skal kunne indstilles fra Labview programmet. | **Test\*** | **JA** |
| Samplerate min | Oscilloskopet skal kunne køre ned til 10 sps. | **Test\*** | **JA** |
| Samplerate max | Oscilloskopet skal kunne køre op til 5.000 sps. | **Test\*** | **JA** |
| Oscilloskopet skal kunne køre op til 10.000 sps. | **Test og Analyse** | **JA** |
| Record length min | Oscilloskopet skal kunne køre med ned til 10 ADC målinger i hver pakke. | **Test\*** | **JA** |
|  | Den minimale tilladelige record length skal tage højde for sampleraten. | **Test og Analyse** | **JA** |
| Record length max | Oscilloskopet skal kunne køre med op til 1000 ADC målinger i hver pakke for alle samplerate. | **Test/**  **Analyse\*** | **JA** |
| RS-232 baudrate | RS232 forbindelsen skal køre med en baud rate på 115.2 kbaud | **Info\*** | **JA** |
| RS-232 håndtering | Modtagelse af data fra LabView programmet skal foregå ved interrupt. Transmission kan foregå ved polling eller interrupt. | **Info\*** | **JA** |
| **Signalgenerator** | |  |  |
| PWM filter | Der skal designes et lav-pas filter der på passende vis udglatter de digitale PWM pulser. | **Test/**  **Analyse\*** | **JA** |
| Parametre | Signalgeneratorens signalform (SHAPE) , amplitude (AMPL) og frekvens (FREQ) skal kunne indstilles fra Labview programmet. | **Test\*** | **JA** |
| SHAPE, AMPL og FREQ kan gøres synligt på syv segment displayet. | **Test** | **JA** |
| Sinus signal | Der kan implementeres en look-up tabel i VHDL koden der gør det muligt at signalgeneratoren kan lave et sinus-formet signal | **Test** | **JA** |
| SPI baudrate | SPI forbindelsen skal køre med en baudrate på 500 kbaud | **Info\*** | **JA** |
| SPI håndtering | To-vejs SPI kommunikation kan implementeres f.eks. med acknowledge handshake | **Info** | **Nej** |
| SPI protokol | Der skal vælges og implementeres en robust protokol til at overføre SHAPE, AMPL og FREQ | **Analyse\*** | **JA** |
| SPI test | Der skal ved test demonstreres en sikker forbindelse ved modtagelse. Denne test kan laves som et separat projekt med moduler fra det endelige oscilloskop projekt. | **Test\*** | **JA** |
| **Større udvidelser** | |  |  |
| Sample rate | Større samplerate (op til 70 ksps) med brudt datastrøm. | **Valgfri** | **JA** |
| Trigger funktion | Indførelse af en trigger funktion og/eller en sync funktion (trigger fra FPGA'en) på Oscilloskopet (kræver at Labview koden ændres) | **Valgfri** | **NEJ** |
| Bodeplot | Automatisk frekvens scan til generering af bode plot (netværksanalysator) | **Valgfri** | **Ja** |
| Signalform | Ny signalform i signalgeneratoren. | **Valgfri** | **JA** |
| Højere ordens filter | Højere ordens filter for bedre filtrering. Helst højere end 2. ordens | **Valgfri** | **JA** |

Verifikation markeret med rødt og med \* er krav sat fra projektoplæggets side.

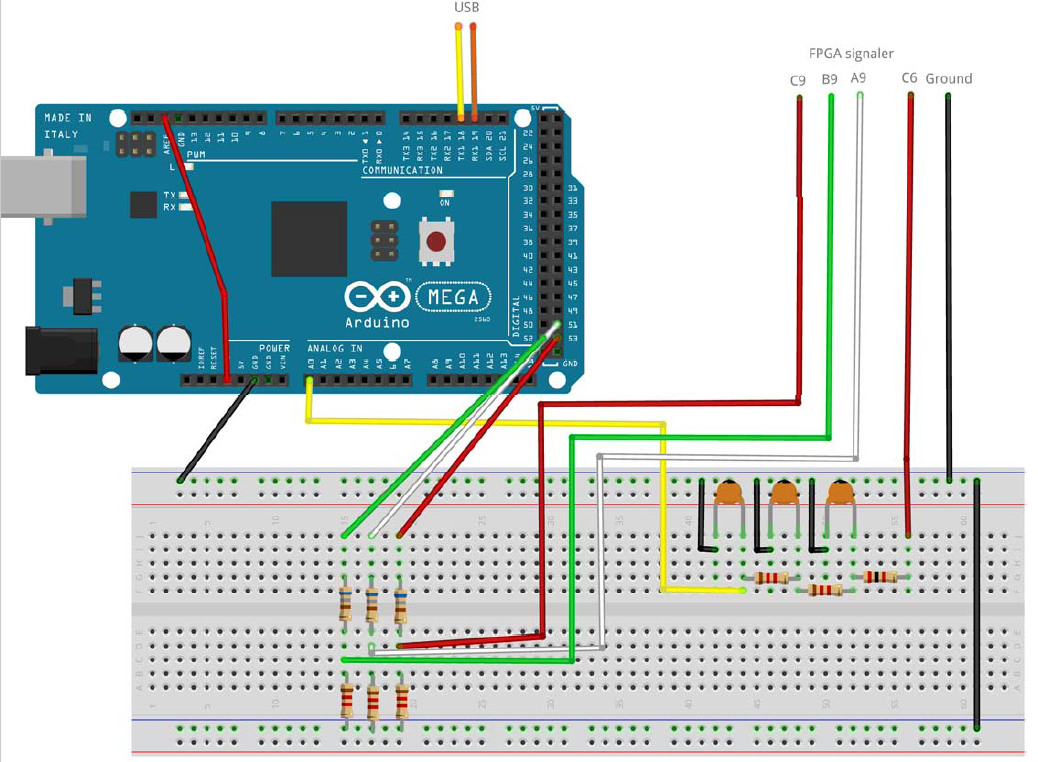
# Design

Systemet fungerer på den måde, at der er en PC blok som er Labview og det brugerinterface, som skal kunne ændre på styresignalerne og vise signalerne på PC skærmen. Blokken ændre styresignalerne ved at sende og modtage ved hjælp af en RS232 forbindelse til MCU blokken. MCU blokken modtager styresignalerne og sender informationer ved hjælp af SPI til FPGA’en. I MCU’en skal der laves håndtering af den data som kommer fra Labview, den data der sendes til FPGA’en og signalet der kommer fra FPGA’en. For at kunne sende til FPGA’en, skal der oprettes en sikker SPI forbindelse, der kan overføre data pakker. FPGA’en skal generer forskellige signaler, alt efter hvad den bliver bedt om af MCU’en. FPGA’en skal sende et PWM signal, som så skal filtreres ved hjælp af et analogt filter. Det filtreret signal skal sendes til MCU’en, som så sender tilbage til PC’en. På nedenstående billede Figur 2 ses skitsen af systemet.



Figur 2 Skitse af systemet

På kredsløbsdiagrammet Figur 3 ses det endelig system. MCU’en modtager gennem TX og RX benene data fra labview. De 5 V SPI signaler (Grøn = SCK, hvid = MOSI og rød = SS) går til spændingsdeleren, hvor de bliver lavet om til 3.3V signaler som går til FPGA’en. FPGA’en sender udgangsignalet ud på C6 som går igennem 3. ordens RC filteret og går tilbage til den analoge pin A0 på MCU’en.

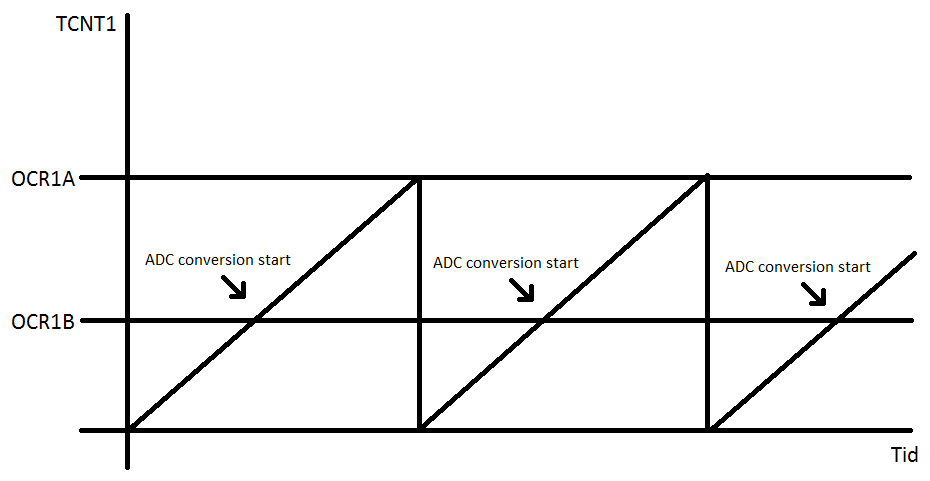


Figur 3 Kredsløbsdiagrammet lavet i Freitzing.

# Oscilloskop

## Analog sampling

For at måle den analoge spænding fra signal generatoren benyttes ATMega2560’erens indbyggede ADC. For at styre sampleraten benyttes en intern 16 bit timer som genererer et compare match interrupt.



Figur 4 ADC Timing diagram

Ovenstående billede Figur 4 viser hvordan vores ADC bliver sat i gang når comparematch b autotrigger i CTC mode bliver benyttet. Der indsamles 8 bit samples fra ADC’en hvilket gør det nemt at sende målingerne over UARTen da den sender i bytes. Målingerne gemmes i en FIFO ring buffer som minimum skal kunne indeholde vores maksimale record længde plus lidt ekstra for en sikkerheds skyld.

## UART TX/RX i mcu’en

### Modtagelse og Transmit

Modtagelse og transmit af data sker ved hjælp af recieve complete interrupt og transmit complete interrupt.

### Problemer med polling baseret transmit

Første gang der med succes blev sendt en datapakke med ADC målinger til labview, skete det hjælp af vores polling baserede sende funktion put\_char. som lægger en byte i UDR registret på UARTen og poller på UDRE1 flaget  som indikerer om dataregistret er tomt. ulempen ved dette er der ikke kan udføres andre funktioner i main mens der polles. Dette resulterede i at vores program ikke fungerede hvis man prøvede at sende til uarten mens den var i gang med at sende data til labview. Dette problem blev afhjulpet ved at implementere en interrupt service routine for transmit complete, som håndterer data der skal sendes.

## Labview protokol

Til at sende og modtage data fra labview og MCU’en benytter gruppen en protokol der er defineret på denne måde.

|  |  |
| --- | --- |
| **Størrelse** | **Indhold** |
| 2 bytes | Sync (0x55AA) |
| 2 bytes | Længden af hele pakken |
| 1 byte | Type (hvilken fane beskeden tilhører) |
| x bytes | Data |
| 2 bytes | Checksum (Zero/LRC/CRC) |

De første to bytes er sync bytes. De indikerer at nu kommer der en besked. De næste to bytes er længden af hele beskeden, så programmet ved hvornår den skal forvente beskeden slutter. Så kommer der en byte der beskriver hvor i labview beskeden kommer fra/sendes til. Derefter kommer alle data bytes’ne og til sidst to checksum bytes.

## Signal generator protokol

Til at sende data fra MCU’en til FPGA’en benyttes SPI med følgende protokol definition.

|  |  |
| --- | --- |
| **Størrelse** | **Indhold** |
| 1 bytes | Sync (0x45) |
| 1 bytes | Adresse |
| 1 byte | Data |
| 1 bytes | Checksum (LRC) |

Den første byte er sync, den næste henviser til hvilken data der skal ændres i FPGA’en, så kommer der en addresse, den data der skal indlæses og til sidst en checksum på adresse og data der benytter LRC.

## Checksum

Der findes forskellige måder, at implementer checksum på, hver med deres fordele og ulemper. Men idé med, at sende en checksum med noget data er for, at sikre sig at der ikke er blevet sendt noget forkert og teste for bitflip.

### Zero16

Zero16 er en meget simpel form for checksum, her er de to checksum bytes altid 0

### LRC8

LRC er et longitudinal redundancy check. Det check MCU’envi udfører fungere ved bitvis xor af alle byte i datapakken og resultatet bliver et 8 bit word der gemmes i den bagerste checksum byte.

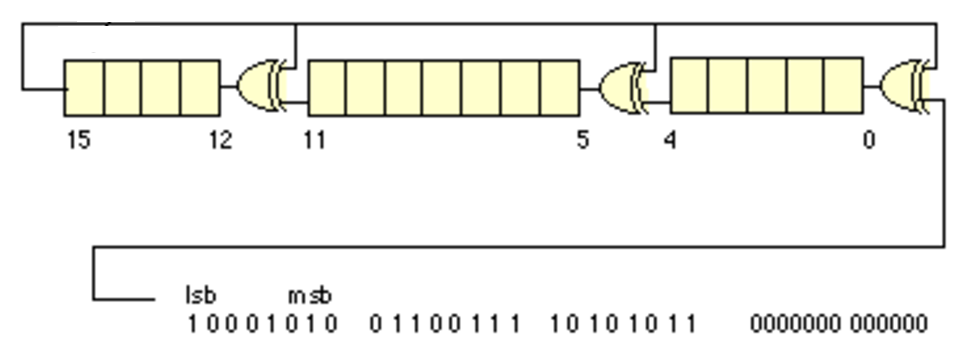
LRC=LRC xor byte

Hvor LRC starter med at være 0

Fordelen er at den er simpel en ulempe kan være at hvis der sker et dobbelt bit flip kan det være algoritmen ikke ser fejlen

### CRC16

Vi har et polynomium x16+x12+x5+1 Hvis vi skriver det ud får vi den binære værdi 1 0001 0000 0010 0001 så hvis vi trunkerer det mest betydende bit får vi et 16 bit tal. Hvis vi omskriver det til hex får vi 0x1021. Ved at benytte en initial værdi 0xFFFF kan sende hele beskeden gennem skifte registeret når alle de ekstra nuller er blevet skiftet ind er checksummen de 16 bit der står tilbage i skifte registeret.



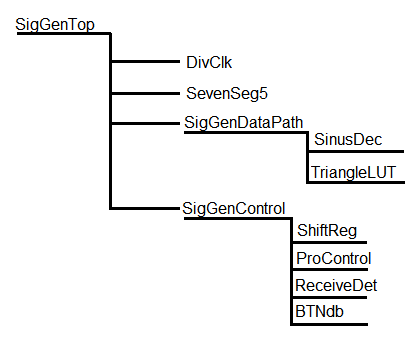
CRC checksummen benyttes til at detekte om der er fejl i dataen, hvilket algoritmen klarer i 99% af tilfældene. På grund af algoritmens matematiske egenskaber er det let at beregne en CRC justering til en vilkårlig ændring af data. Derfor kan denne algoritme kun anvendes til at verificere korrekthed men ikke dataintegritet.

## Bodeplot

Der ønskes implementeret en bodeplot funktion der gennemløber alle 255 frekvenser fra signal generatoren. Signal generatoren indstilles til sinus med maximum amplitude hvorefter udgangen at signal filteret tilsluttes et lavpasfilter med en lavere knækfrekvens. MCU’en skal herefter afgøre maksimum og minimum af signalet efter test-filteret for alle 255 frekvenser og placere differencen mellem de to værdier i et array. alle amplitude værdier sendes til labview til fremvisning.

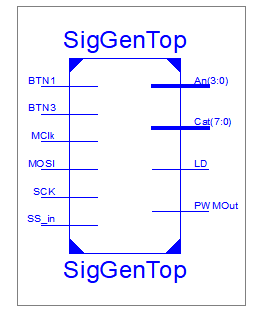
## Signalgenerator

Signalgeneratoren er programmeret på FPGA’en i VHDL. Koden skrives struktureret som moduler og under moduler. Hvert modul og under modul vil i dette afsnit blive beskrevet. Noget VHDL kode er med projektoplægget blevet udleveret samt manual til denne kode, som ligger grundstenen til, at gruppen kunne udføre test på de enkelte moduler projekte besod af. På Figur 5 ses modulerne hierarkisk:



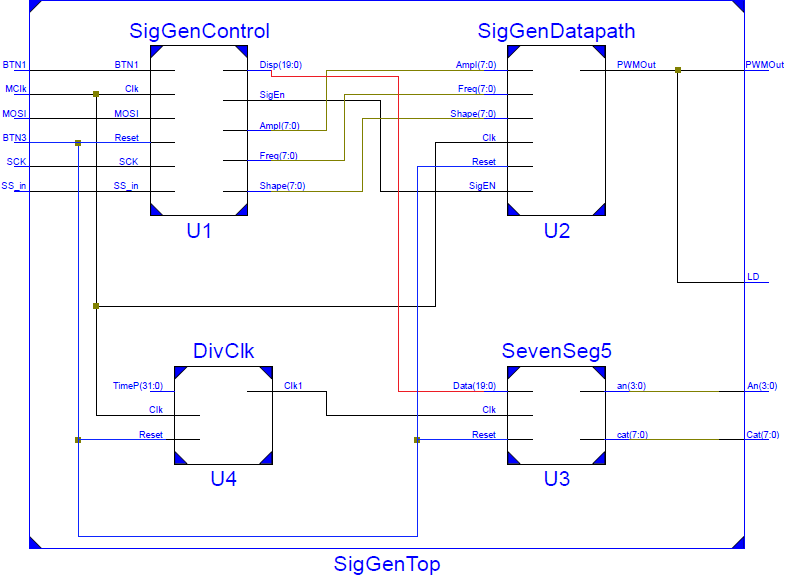
Figur 5 Modulhirakiet

SigGenTop er top modulet Figur 6, hvor I/O på FPGA’en bliver beskrevet. Gruppen har valgt, at bruge “BTN1” og “BTN3” som er trykknapper på FPGA boardet, som fungere som henholdsvis “skift menu tilstand” og “reset” knap. “MClk” som er FPGA’ens clock er sat til indbyggede standard crystal på 50 MHz. “MOSI”, “SCK” og “SS\_in” er SPI pins der bruges til envejs SPI kommunikation. “An” og “Cat” er porte som henholdsvis er Anode og Cathode til syv -segementsdisplayet på FPGA’en. “LD” er en lysdiode på FPGA’en, hvis lysstyrke svarer til “PWMout” portens PWM signal.



Figur 6 SigGenTop

Næste tegning viser SigGenTop’s 4 under moduler:



Figur 7 SigGenTop undermoduler

SigGenTop består af 4 undermoduler som beskrives kort:

### Modulet SigGenControl

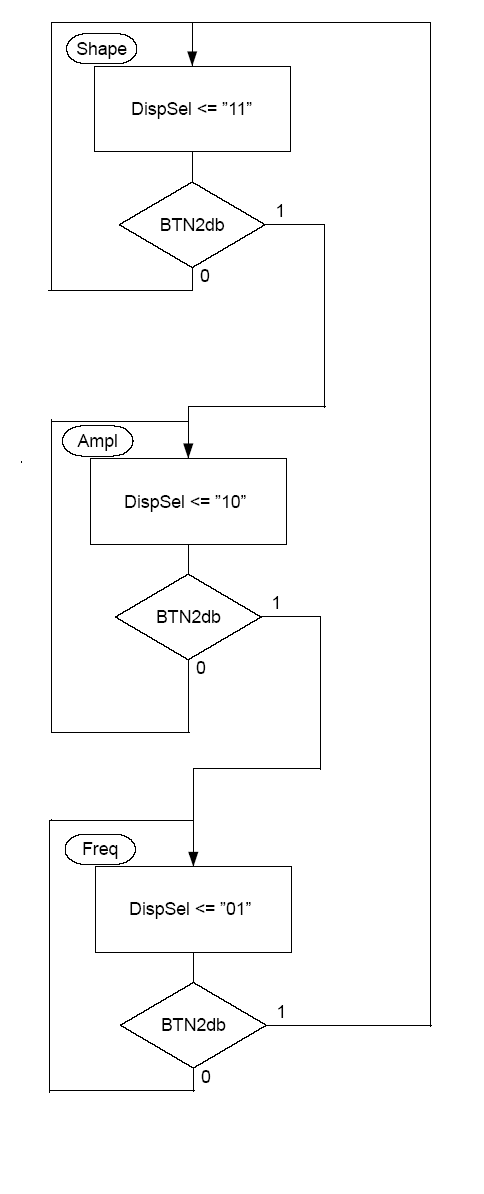
Modulet SigGenControl er signalgeneratorens controller som holder styr på følgende:

* Hvor brugeren og FPGA’en er henne i indstillingsmenuen og hvad der skal skrives på displayet.
* Modtage og fortolker den data som den modtager via SPI
* Bestemme om signal generatoren skal være tændt og hvilke parametre signal generatoren skal lave signalet ud fra.

Modulets I/O:

|  |  |  |
| --- | --- | --- |
| Indgange | Reset | Master Reset, Styret af BTN3 |
|  | CLK | Master Clock (50MHz) |
|  | BTN1 | Knap til at skifte menu i indstillingsmenuen |
|  | MOSI | SPI “Master In Slave Out” Data |
|  | SCK | SPI “Serial Clock” |
|  | SS\_in | SPI “Slave Select” |
| Udgange | Disp | 19 bit tal som er hvad der skal udskrives til displayet |
|  | SigEn | Enable signal om signalgeneratoren skal sende signalet ud eller om den er slukket |
|  | Ampl | 8 bit tal der angiver amplituden på signalet |
|  | Freq | 8 bit tal der angiver frekvensen på signalet |
|  | Shape | 8 bit tal der angiver formen på signalet |

ASM diagrammet Figur 8 er taget fra “SignalGeneratorManual”en og modificeret til den nye tilstandsmaskine der styrer, hvilken af menuerne displayet udskriver.



Figur 8 ASM Diagram over SigGenDatapath

### Modulet SigGenDataPath

Modulet SigGenDataPath er udleveret ved projektstarten og følgende I/O:

|  |  |  |
| --- | --- | --- |
| Indgange | Reset | Master Reset, Styret af BTN3 |
|  | CLK | Master Clock (50MHz) |
|  | SigEn | Enable signal om signalgeneratoren skal sende signalet ud eller om den er slukket |
|  | Ampl | 8 bit tal der angiver amplituden på signalet |
|  | Freq | 8 bit tal der angiver frekvensen på signalet |
|  | Shape | 8 bit tal der angiver formen på signalet |
| Udgange | PWMOut | PWM signalet |

Modulet har til formål at generer de forskellige signaler, udfra de 4 parametre som den modtager fra controlleren. Indstillings tabellen taget fra filen “SignalGeneratorManual”, viser hvilke indstillinger som modulet kan modtage:

|  |  |  |
| --- | --- | --- |
| Navn på parametre | Navn på signal på Figur 7 | Addresse i hex |
| Shape | Shape(7;0) | 0x00 |
| Amplitude | Ampl(7;0) | 0x01 |
| Frekvens | Freq(7;0) | 0x02 |
| Tændt/slukket | SigEn | 0x03 |

Første parametre er, hvilken slags “form” signalet har. Vores gruppe har som en del af de ekstra implementeringer man kunne vælge, valgt at tilføje et sinus og trekantet signal også:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec. Values** | **Data** | **Display** | **Description** |
| **0** | 00000000 | S 00 | Constant DC voltage |
| **1** | 00000001 | S 01 | Square-pulse |
| **2** | 00000010 | S 02 | Saw-tooth |
| **3** | 00000011 | S 03 | Sine wave |
| **4** | 00000100 | S 04 | Triangel |

Næste parametre er Amplituden, som går fra 0V til 3.3 V fordelt ud på 0 til 255 decimalt:

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec. Values** | **Data** | **Display** | **Description** |
| **0** | 00000000 | A 00 | 0V amplitude |
| **127.5** | 01111111 | A 7F | Half amplitude, about 1.65V |
| **255** | 11111111 | A FF | Full amplitude, about 3V |

Tredje parametre bestemmer ved hvilken frekvens signalet skal generes ved. Frekvensen går fra 0 Hz til 23 kHz fordelt ud på 0 til 255 decimalt.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dec. Values** | **Data** | **Display** | **Description** |
| **0** | **00000000** | A 00 | 0 Hz |
| **1** | **00000001** | A 01 | 24 Hz |
| **127** | **01111111** | A 7F | 11.5 kHz |
| **255** | **11111111** | A FF | 23 kHz |

En sidste parametre har gruppen selv implementeret, da gruppen valgte at fjerne switch knapperne, knap 0 og 2 på FPGA og dermed brugerens mulighed for selv at starte signalgeneratoren og ændre værdier for de 3 andre parametre. Parameteren bestemmer om signalgeneratoren sender signalet ud eller den bare sender ingenting ud.

|  |  |  |
| --- | --- | --- |
| **Dec. Values** | **Data** | **Description** |
| **0** | **00000000** | Off |
| **1** | **00000001** | On |

Den data som ændre på parametrene kommer fra modulet “SigGenControl”.

### Modulet SevenSeg5

Modulet SevenSeg5 er også et på forhånd opgivet modul. Det modtager Data fra SigGenControl og styrer hvad der skal skrives ud på de 4 syv-segmentsdisplayet, ved at styre Anode og Cathoden.

|  |  |  |
| --- | --- | --- |
| Indgange | Reset | Master Reset |
|  | CLK | Display clock (1 kHz) |
|  | Data | 19-bit data |
| Udgange | Anode | Signalet der styrer Anoderne på de 4 syv-segment displays |
|  | Cathode | Signalet der styrer Cathoderne på de 4 syv-segment displays |

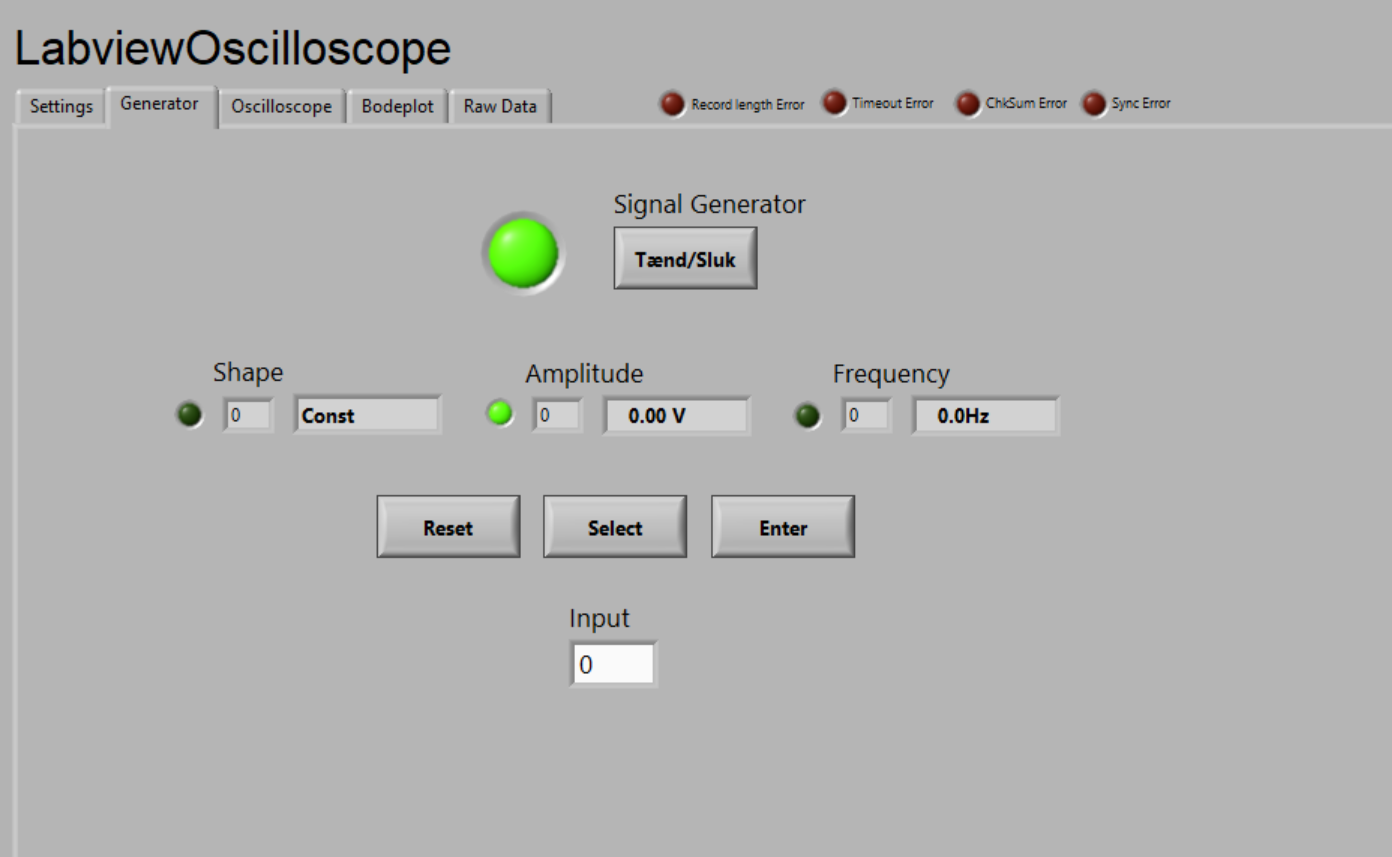
### Modulet “DivClk”

Modulet “DivClk” sørger for at dividere klokken med inputtet Ptime som er defineret som en kontant på 50.000. DivClk skal sørge for at SevenSeg5 modulet opdatere syv-segmentsdisplayet ved en passende frekvens på 1 kHz.

|  |  |  |
| --- | --- | --- |
| Indgange | Reset | Master Reset |
|  | CLK | Master Clock (50MHz) |
|  | TimerP | Nedskalerings værdi |
| Udgange | Clk1 | 1 kHz clk til syv-segmentsdisplayet |

## Indstilling af parametrene i LabView

Sådan som gruppen har valgt at kunne ændre på de forskellige parametre, er ved hjælp af Labview. På Figur 9 ses hvordan generator menuen ser ud i Labview. Det kan se at der er en “Tænd/Sluk” som tænder og slukker signal generatoren. “Reset” knappen resetter MCU’en og sender start parametrene til FPGA’en (Parametrene ses på billedet). “Select” bevæger sig mellem Shape, Amplitude og Frequency menuerne i labview. “Enter” sætter den indtastede værdi som den grønne lampe lyser ud for. I input feltet tastes en decimal værdier mellem 0 og 255. Er værdien der indtastes >=255 bruges værdien ‘255’. Dog ikke, hvis brugeren indsætter værdien i “shape”, da vil ‘4’ indlæses som er“triangle” signalet.



Figur 9 Labview

## SPI

I dette projekt, skulle gruppen sende til FPGA’en fra MCUen. Gruppen har valgt at bruge SPI til at realisere dette. SPI, som skulle være en synkron seriel forbindelse, skulle fungere som en punkt-til-punkt forbindelse. Masteren skal styrer kommunikationen og slaven skal kun i vores system modtage data. På Figur 10 er SPI hardware opsætningen vist. Gruppen har dog i dette projekt ikke 2 vejs kommunikation, så MISO (Master In Slave Out) signalet bruges ikke.



Figur 10 SPI

Link: <http://maxembedded.com/2013/11/serial-peripheral-interface-spi-basics/>

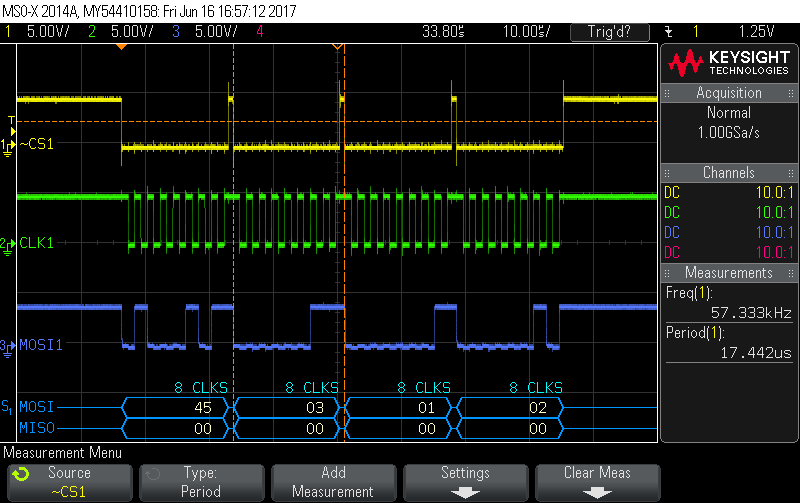
SPI forbindelsen brugt i systemet bruger derfor 3 ledninger:

SCK “Serial Clock” - Clock der er genereret af Masteren

SS “Slave Select” - Aktiv lavt signal, der indikere om Masteren sender eller ej.

MOSI “Master Out Slave in” - Seriel data overført 1 byte af gangen

Timing diagram # viser en overførsel fra MCU’en til FPGA’en af en pakke. Det ses, at SCK og SS (CLK1 og ~CS1 på diagrammet) signalerne er aktiv lave og hver gang 1 byte af pakken er overført går SS og SCK høj i kort tid, for derefter at gå lav igen og påbegynde at sende næste byte.



Figur 11 Timingdiagram

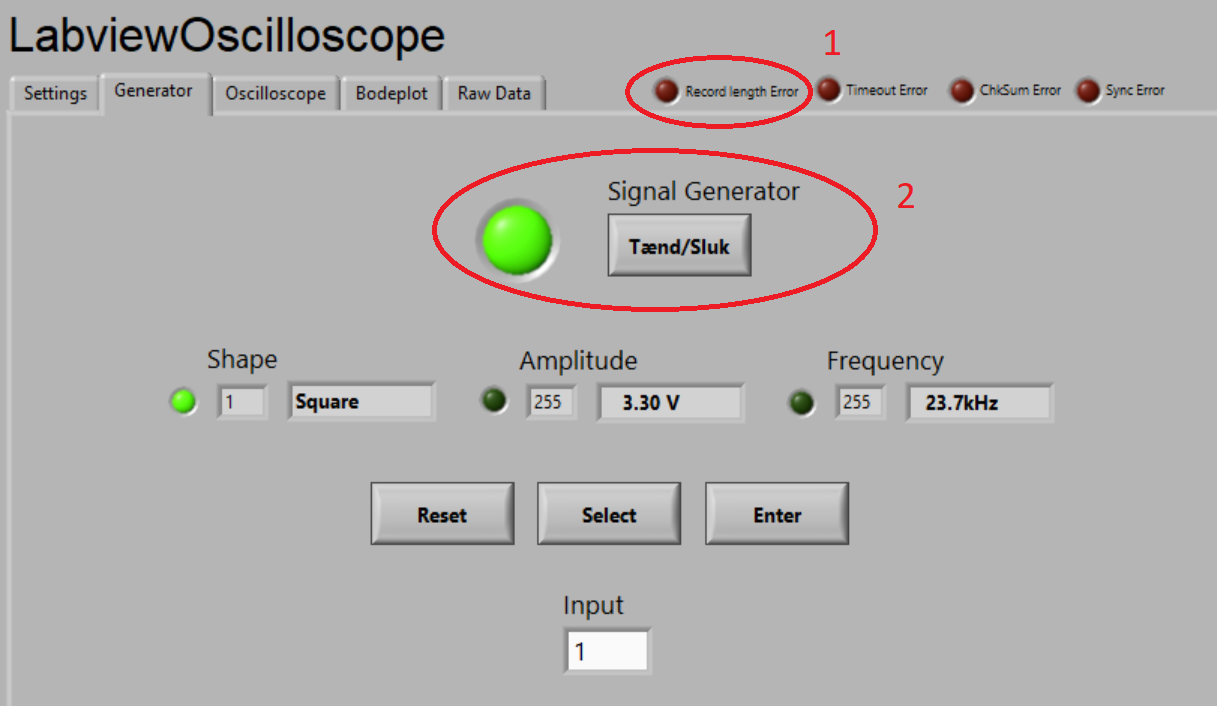
På Figur 11 ses timingdiagrammet for - SPI forbindelsen mellem MCU’en og FPGA’en vist ved hjælp af El-labs Oscilloskops “SPI Decoder” funktion.

Gruppen har dog forsøgt over 2 projektdage, at implementere 2 vejs kommunikationen og SPI handshake uden held. Det viste sig efterfølgende, at være en blanding mellem fejl på MCU’ens side pga. forkert SPI mode opsætning, ingen sammenhæng mellem hvad der blev sendt af test signaler og hvad der blev modtaget og vigtigst af alt grundet, at det var svært at opsætte “testbenche”s som reflekterede det der skete i virkeligheden. Implementeringen i form af koden til MCU’en og FPGA’en er lavet, men da gruppen ikke kunne få det til at virke i virkeligheden er det ikke med i rapporten. Det var dog en forholdsvis simpel implementeringen på MCU og FPGA siden det bare bestod af følgende:

* Opsætningen af parallel til seriel 8-bit skifteregister til “MISO” på FPGA’en samt en ny ekstra protokol og tilstande i tilstandsmaskinen til, at sende tilbage til MCU’en.
* Simple funktion til at modtage data fra MISO signalet over SPI på MCU’en, behandle dataen og vurderer om dataen der var blevet sendt til FPGA’en blev modtaget korrekt.

## Labview

Til Labview vil gruppen lave nogle simple udvidelser for at gøre programmet mere brugervenligt for brugeren. Gruppen har tilføjet en error lampe(1) til at vise om man har tastet én for lille record length i forhold til sampleraten. Den anden udvidelse er en indikator(2) for om signal generatoren er tændt eller slukket. Se Figur 12

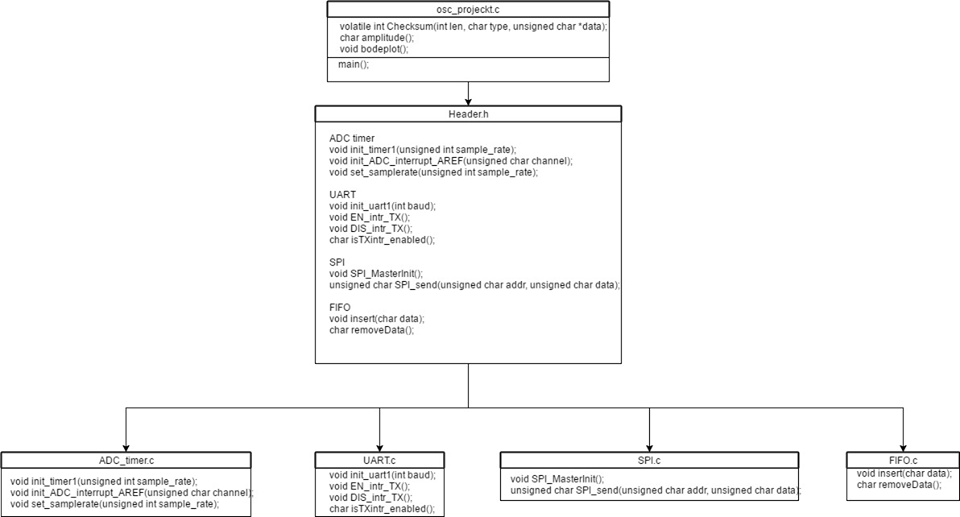


Figur 12 Ekstra features i Labview

# Implementering

## Oscilloskop

### C-blok diagram



Figur 13 Modul diagram over C-koden

### Dokumentation af c-moduler

FIFO.c

|  |  |
| --- | --- |
| Ansvar | Håndtering af ring buffer til dataindsamling |
| Brug af globale variable | Rear, front, itemCount, FIFO[MAX] |
| Funktioner og parametre | Void insert(char data)  insert sørger for at indsætte data givet til den på rear pladsen i FIFO arrayet samt inkrementere rear og itemCount.    Char removeData()  removeData sørger for at returnere data der ligger først i FIFO’en |

SPI.c

|  |  |
| --- | --- |
| Ansvar | Initialisering og håndtering af SPI kommunikation |
| Brug af globale variable | Ingen |
| Funktioner og parametre | Void SPI\_masterinit()  initialiserer SPI til 500 kbaud og master indstilling leading edge på falling.    Char SPI\_send(unsigned char addr, unsigned char data)  sender til FPGA ved hjælp af den udviklede protokol med en databyte og en adresse dertil. |

UART.c

|  |  |
| --- | --- |
| Ansvar | Initialisering af uart samt håndtering af interrupt indstillinger |
| Brug af globale variable | Ingen |
| Funktioner og parametre | Void init\_uart1(int baud)  initialiserer uart til 8bit no parity enabler recive og transmit interrupt  Void EN\_intr\_TX()  enabler transmit interrupt på uart 1.  ‘  Void DIS\_intr\_TX()  disabler transmit interrupt på uart 1.    Char isTXintr\_enabled()  returnerer 1 hvis transmit interrupt er enablet  returnerer 0 hvis transmit interrupt er disablet |

ADC\_timer.c

|  |  |
| --- | --- |
| Ansvar | Initialisering af timer og ADC samt håndtering af samplerate |
| Brug af globale variable | Ingen |
| Funktioner og parametre | Void init\_timer1(unsigned int sample\_rate)  initialiserer timer 1 med prescaler på 64 og compare match b interrupt enable samt udregner compareværdi    Void init\_ADC\_interrupt\_AREF(unsigned char channel)  initialiserer ADC med prescaler på 16, interrupt enable og voltage reference på AREF benet samt indstiller målekanal. ADC auto trigger på comparematch b.    Void set\_samplerate(unsigned int sample\_rate)  udregner ny compareværdi. |

|  |  |
| --- | --- |
| interrupt service rutiner | ISR(ADC\_vect)  aktiveres når der er en sample klar som lægges i FIFO bufferen ved hjælp af insert funktionen    ISR(TIMER1\_COMPB\_vect)  gør ingenting men den skal være der for korrekt operation af ADC auto-trigger. |

osc\_projekt.c

|  |  |
| --- | --- |
| ansvar | håndtering hvilke dele af programmet der kører |
| brug af globale variable | enum {Default,Generator, Oscilloskop, Bodeplot};  enum {Enter,Select,RunStop,Reset};  enum {ZERO16, LRC8, CRC16};  char checksum = ZERO16;    volatile unsigned char UART1=0, Generator\_data[5];  volatile int package\_len;  unsigned char buff1[50], msg\_type = Oscilloskop;  volatile unsigned char amp[256]; |
| funktioner og parametre | volatile int Checksum(int len, char type, unsigned char \*data);  udregner char checksum ud fra programindstillinger  char amplitude();  tager 5000 samples af en bølge og finder amplituden  void bodeplot();  gennemløber alle signal generatorens frekvens ved sinus og max amplitude og sinus form og kalder amplitude funktionen og organiserer dataene i et array som benytte til interrupt transmit. |
| interrupt service rutiner | ISR(USART1\_TX\_vect)  sørger for interrupt baseret transmit    ISR(USART1\_RX\_vect)  sørger for modtagelse og fortolkning af data |

### Databufferen

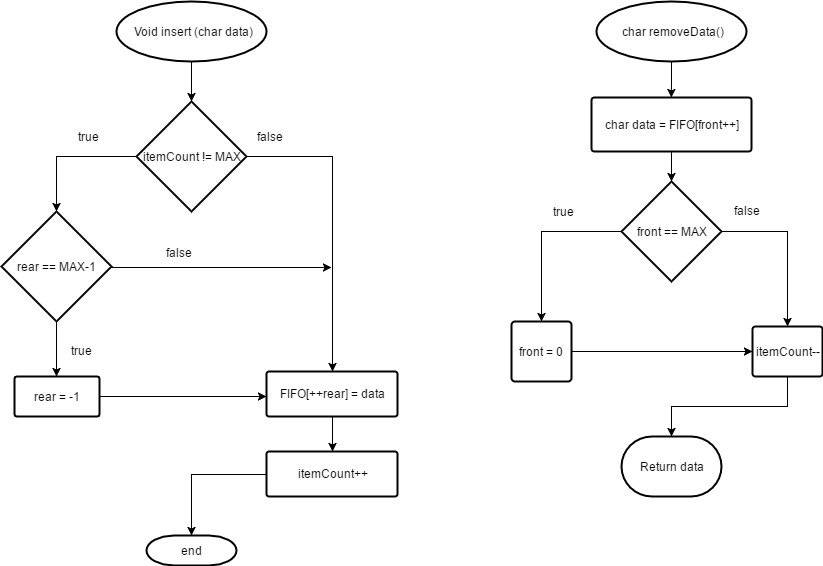
Som databuffer benyttes en FIFO ring buffer som er implementeret med et char array(FIFO[MAX] hvor MAX er en defineret macro), samt parametrerne front rear og itemcount. item count inkrementeres når der tilføjes data til FIFO bufferen og dekrementeres når der fjernes fra bufferen. front holder styr på hvor data elementet der er forrest i køen er placeret. rear holder styr på hvor det bagerste element er placeret således at ny data kan sættes bagerst i køen.

I forbindelse med FIFO bufferen er der udarbejde to funktion til interace med bufferen

Void insert(char data)

char removeData()

insert() har indsætter data i FIFO’en og removeData trækker data ud.

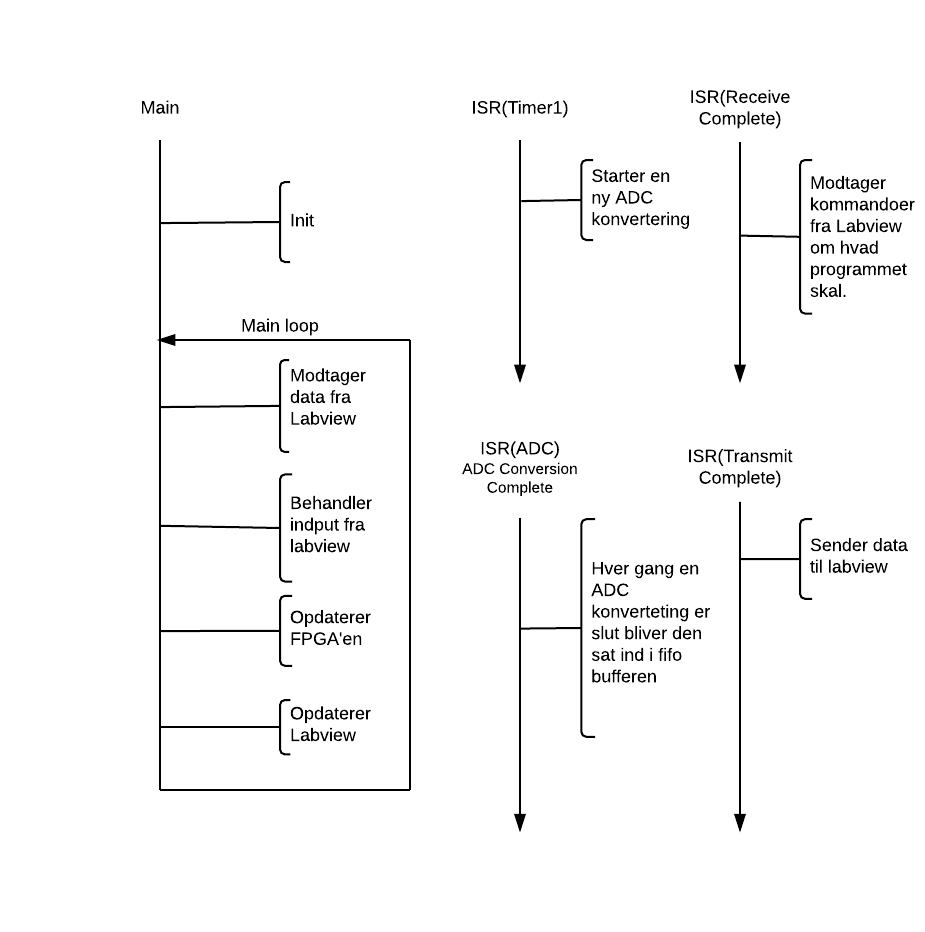


Figur 14 Flowchart for FIFO

Figur 14 viser flowchart for FIFO funktionerne. funktionerne håndterer at man begynder forfra i arrayet når man når enden. således at arrayet fungerer som en ring buffer.

### Interrupt diagram

På Figur 15 ses vores interrupt diagram. Der er fire service rutiner der kan intterupte vores main program. Service routinen ISR timer 1 skal være der, da vi benytter compare match interruptet til at starte en ny sample. Så har vi enablet transmit og receive comlpete på uart 1 så vi ikke poller når vi skal sende eller modtage noget.



Figur 15 Interrupt diagram

### Analog sampling

ADC’ens indstillinger.

ADC måler konstant på kanal 0.

ADC bliver clocket med MCU’ens 16 mhz clock-frekvens med en prescaler division på 16 hvilket giver en clock frekvens på 1 mhz.

ADC interrupt enables således at dataen kan indsamles i en service routine.

ADC’en sættes i autotrigger mode med trigger på timer 1 compare match b.

ADC’en har aktiveret “Left Adjust Result ” som sørger for at ADC’ens 8 mest betydende bit ligger i ADCH registret.

Timerens indstillinger

timer 1 benyttes.

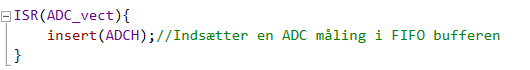
timeren bliver clocket med MCU’ens 16mhz clock-frekvens med en prescaler division på 64 hvilket giver en clockfrekvens på 250khz.

timeren kører i CTC mode hvor den ønskede top værdien lægges i OCR1A registrene. OCR1B sættes til 1 da denne definerer triggerpunktet for ADC’ens auto trigger funktion.

interrupt på compare match b enables da det kræves for ADC autotriggermode.

ADC service-rutine

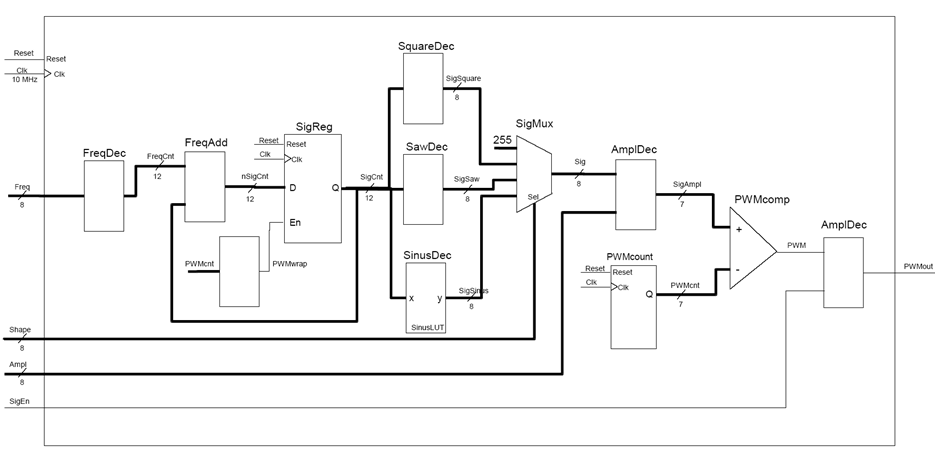
data fra ADC’en  lægges over i FIFO bufferen ved hjælp af insert funktionen. derfor benyttes der kun en linje kode i ADC service rutinen.



## Signal generator

### SigGenDataPath

På Figur 16 bliver der blandt andet genereret pwn tæller pulsen og signalerne. Sådan som det fungerer, er frekvensen som tæller op “FreqCnt”. FreqCnt bruges til at addere frekvensen sammen med FreqCnt til blandt andet det firkantet signal og savtak signalet. Siden Sinus og Trekant signalet er en LUT, så skal de dekodes, hvilket tager rigtig mange resourcer. Når signalerne  er kommet igennem SigMux og AmplDec så kommer PWMwrappet. PWMwrappet sker når PWM tælleren rammer compare værdien på comperatoren. På den måde bliver signalet kontinuert og periodisk. På nedenstående billede ses Datapath givet af Peter Brauer og Ole Schultz. Det som vores gruppe har modificeret ved dette modul, er ved at implementere et trekantet signal med en LUT.



Figur 16 SigGenDatapath undermoduler

Sådan som vores sinus- og trekant signal fungere, er ved en storA LUT (Look up table). I dette projekt har gruppen lavet både et trekantet signal og et sinus signal, ved hjælp af en LUT. Vores gruppe har fremstillet LUT i excel og konverteret det til en .coe fil. Selve coe filerne bliver placeret i modulet SigGenDataPath, som er der hvor signal generatoren er placeret. Modulet er opbygget af forskelliger tællere, som skal generere PWM signalet. Sådan som det fungere er at man dedikere ram, til at FGPA’en kan kigge i tabellen. For at kunne fremstille de forskellige LUT, så skal en formel bruges til at kunne generere de forskellige værdier. Værdierne kommer i decimal og de skal så konverteres før det kan bruges. Vores sinus signal var let nok at implementere, da formlen allerede var givet på forhånd. Det svære var at få formel som der skulle bruges for at kunne fremstille vores trekant signal. For at kunne fremstille et trekantet signal, skal top og bund værdien være kendt (V1 og V2). Derudover skal periodetiden også være kendt for at kunne fremstille signalet (T1 og T2). Sådan som man indstiller den, er ved at vide ens dT som er tiden inkrementeret med dT plus de andre parametre.

Se nedenstående link for Kilde.

<http://www.ecircuitcenter.com/VBA/Topics/Tri_Wave/Tri_Wave.htm>

Vores gruppe fandt ud af at LUT tager rigtig meget RAM. Så vores gruppe prøvede i en projektdag for at løse problemet, dog uden held. På nedenstående billede, ses design rapporten af VHDL koden. Her kan vi se at alt RAM bliver brugt, hvilket ikke er optimalt.

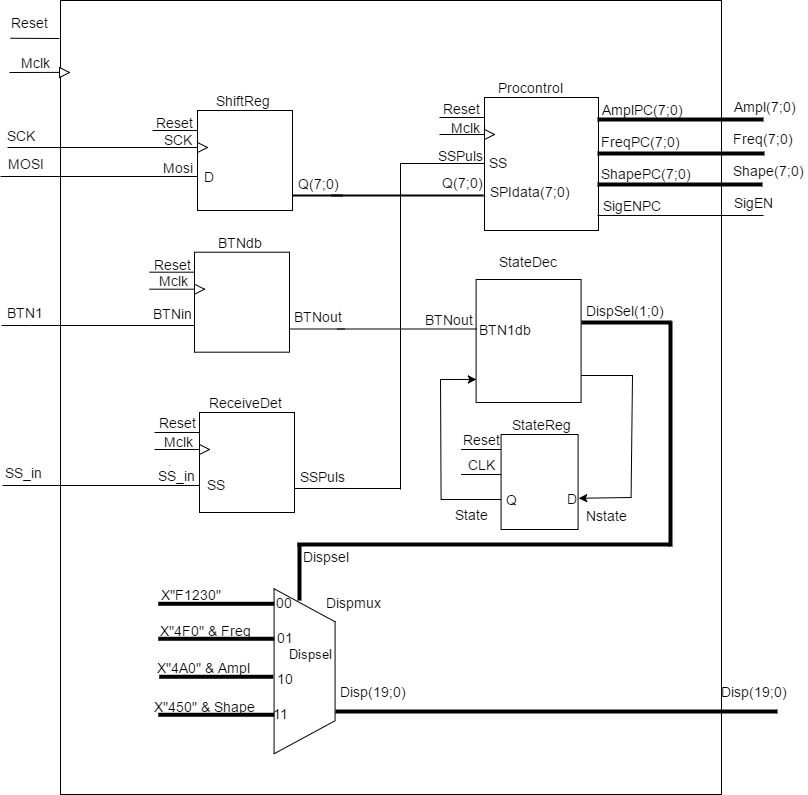


### SigGenControl

SigGenControl Modulets undermoduler og funktioner er beskrevet i tabellen og på funktions diagrammet på næste side.

|  |  |  |
| --- | --- | --- |
| Navn | Funktion | Undermoduler |
| ShiftReg | 8-bit skifteregister der modtager MOSI signalet og bliver clocket af SCK. | 8-bit Seriel-til-parallelt skifteregistrer |
| ReceiveDet | Input synkronisering af SS signalet. | To stk. 1-bit registrer |
| ProControl | SPI protokol fortolker der styrer de 4 signaler der går til datapath’en. | Tilstandsmaskine og registrer |

Som det kan ses på Figur 17 er der 3 moduler: “ShiftReg”, “ReceiveDet” og “ProControl” som er de moduler gruppen selv har implementeret. “StateDec”, “StateReg”, “BTNdb” og “DispMux” var opgivet på forhånd, men modificeret så de andre 3 moduler gruppen lavede passer med dem. Hvis man ser på manualen for Signal Generatoren gruppen fik udleveret, er de største rettelser: Fjernet Switches, BTN2 og BTN0 og at de 3 registre er blevet rykket ind i modulet “Procontrol”. Beslutningen om, at fjerne knapperne og switchesne var grundet, at gruppen ønskede at være sikre på at det kun var labview der kunne ændre på indstillingerne og derved sikre, labview og FPGA “altid” har de samme indstillinger. Det skal selvfølgelig siges, at det ikke garantere synkronisering mellem labview og FPGA’en da der ikke er et SPI handshake eller 2 vejs SPI kommunikation.



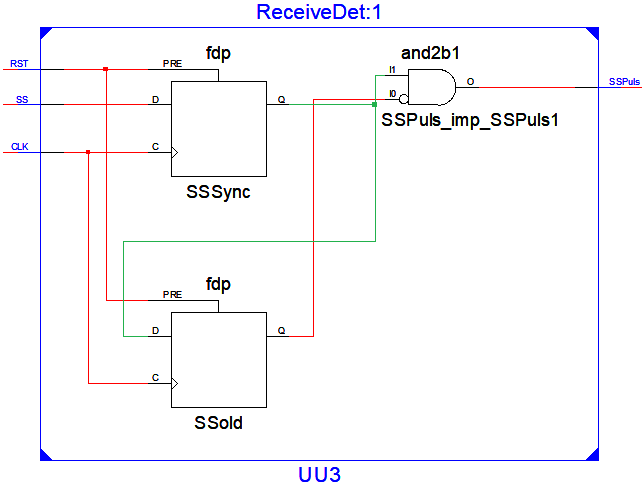
Figur 17 Funktionsdiagram over SigGenControl

### Shiftreg

Undermodulet “ShiftReg” er et 8-bit serielt til parallel skifteregister der bliver clocket med “SCK” clocken på nedadgående flanke. Den nedadgående flanke er valgt, da SCK er idle høj, defineret udfra SPI mode valgt på MCU’en. Modulet sender parallelt SPIdataen videre til Procontrol på en 8-bit bus.

### ReceiveDet

Undermodulet “ReceiveDet” input synkroniser SS signalet, da FPGA’en kører med en 50 MHz clock og MCU’ens clock kører ved en lavere 16 MHz. Undermodulet skal derfor sørger for at MCU’ens SPI “SCLK” clock bliver synkroniseret med FPGA’ens clock. Clock domainerne bliver synkroniseret (Clock Synchronization) ved hjælp af registrer og en AND gate med en inverter på et af benene, som det kan ses på Figur 18.



Figur 18 Funktionsdiagram over ReceiveDet

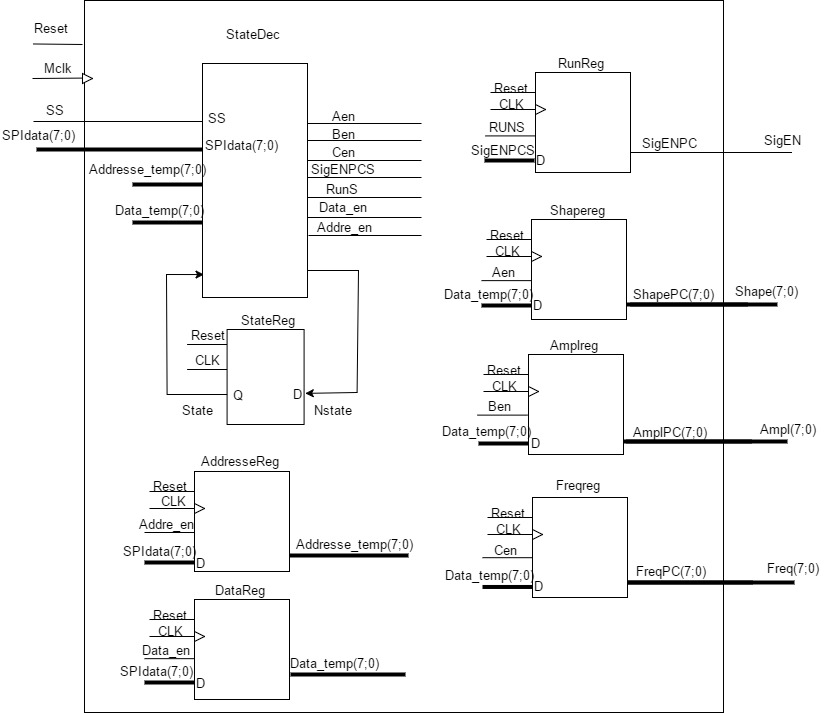
De to “kasser” SSSync og SSold er standard registrer, der ved et reset af systemet bliver til ‘1’ i stedet for 0. Et tidsdiagram fra TestBenchen viser, hvordan SS signalet (der er aktivt lavt) med den “langsomme” SCK clock bliver clock synkroniseret med FPGA’ens clock og outputter  SSpuls, som er ‘1’ i kun en FPGA clock periode. Se Figur 19



Figur 19 Timingdiagram

### ProControl

På funktionsdiagrammet kan undermodulet “ProControl” ses. Modulet indeholder protokolfortolkeren, registrene til de 4 parametre og 2 registrer til at holde midlertidige addresse og data værdier. Se Figur 20

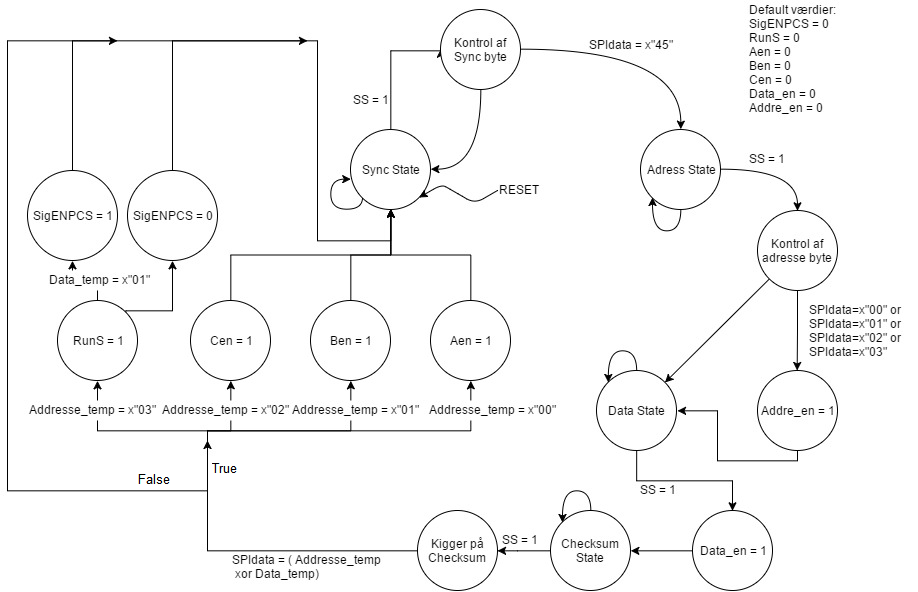


Figur 20 Funktionsdiagram over ProControl

På Figur 20 ses Mealy tilstandsmaskinen. Tilstandsmaskinen har 6 inputs, 7 outputs og styrer følgende:

* Enable signalerne til de midlertidige 8-bit addresse og data registre.
* Enable signalerne til de fire parametre registrer.

På Figur 21 ses tilstandsdiagrammet over tilstandsmaskinen i ProControl. Sådan som den fungere er ved default værdierne bliver sat og tilstandsmaskinen starter i Sync State. Så når SS (SSpuls signalet fra ReceiveDet) går højt, så går den over til “Kontrol af Sync Byte”. Herefter bliver der kigget om SPIdata har modtaget synkroniserings byten. Hvis Sync byte er lig med x45 så skiftes state til Adress State. Her ventes der igen til SS = ‘1’ og adressen kontrolleres om den er gyldig eller ej. Hvis den er gyldig (x00, x01, x02 eller x03) sættes “Addre\_en” enable signalet højt og adressen sættes ind i “AddresseReg”, hvor den bliver opbevaret midlertidigt. Derefter går den til “Data State” og afventer SS signalet. Data’en bliver lagt ind i “DataReg” hvor den opbevares midlertidigt. Derefter gås til “CheckSUm State” hvor der igen ventes på SS signalet. Når SS signalet kommer gås der til “Kigger på Checksum” hvor checksummen bliver udregnet og sammenlignet med den checksum den lige har modtaget. Hvis checksummen er korrekt, bliver enable signalet til det register der er skrevet til (bestemt af adressen) sat højt og registeret indlæser værdien fra “Data\_temp”. Registeret “RunReg” er 1-bit så det går bare højt eller lavt afhængig af om der er skrevet 0x01 eller 0x00 i data\_temp. Efter at der er skrevet til et register/checksummen er blevet erklæret forkert går tilstandsmaskinen tilbage til “Sync State” og afventer sync byten.



Figur 21 Tilstandsdiagram over tilstandsmaskinen

Der er også blevet lavet et ASM Diagram over tilstandsmaskinen. Diagrammet ligger i Appendiks. Det er mere detaljeret men fylder meget. Se

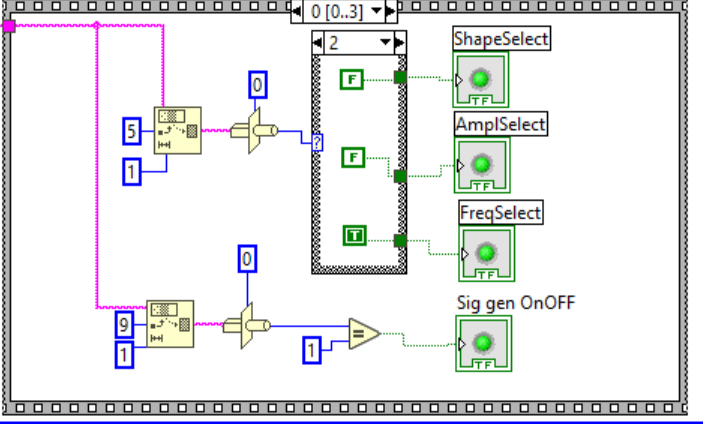
## Labview

### On/Off indikator

For at kunne kontrollere om signal generatoren er tændt eller slukket har vi tilføjet en “true-false” indikator. Den er lavet ved at vi har udvidet pakken vi sender fra MCU’en så den indeholder informationen om signal generatoren er tændt/slukket. Pakken ser ud som følger

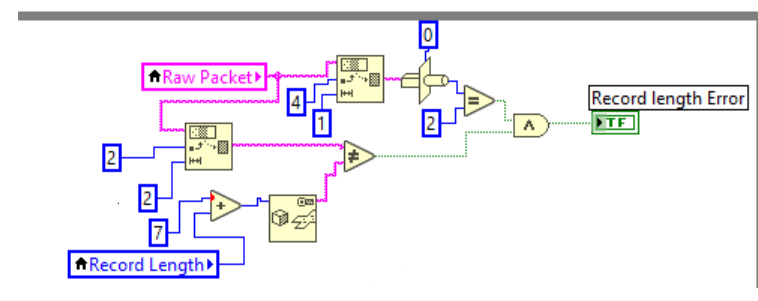
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0x55 | 0xAA | 0x00 | 0x0C | 0x01 | Indicator | Shape | Amp. | Freq. | On/off | check | sum |

Derfor tager kigger vores funktion i labview på det 9. element og ser om det er 1 eller 0.

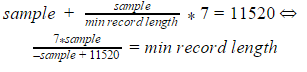


### Record length error

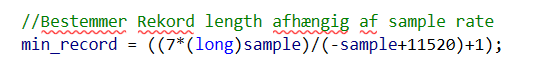
For at kunne bestemme om record length er noget den ikke må være, kigger vi i labview på om den record length man har indtastet, matcher med den længde pakken MCU’en sender faktisk er.



For at bestemme hvad den minimum record length i MCU’en kan være, har vi opstillet en ligning der beskriver hvor mange dette. Da vi sender med en baudrate på 115200 og der er en start og stop bit hver gang vi sender sender vi 10 bit. Hver gang vi sender en pakker koster der 7 ekstra byte. Vi kan derfor opstille denne ligning og isolere for rekord length og så ved vi hvad minimum record length i forhold til sample raten skal være.

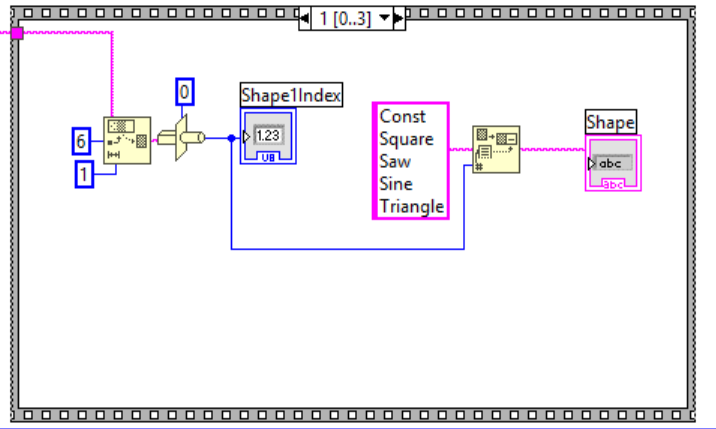


Derved får vi denne funktion for min record length i c vi lægger 1 til for at runde op til nærmeste hel tal.



### Ekstra waveform

Vi har i drop down menuen hvor man kunne vælge waveform tilføjet et trekant signal.



## Bodeplot

Bodeplotet implementeres ved hjælp af en funktion som kaldes når MCU’en modtager beskeden fra labview om at der ønskes et bodeplot.

I forbindelse med dette implementeres funktioner void bodeplot() og char amplitude().



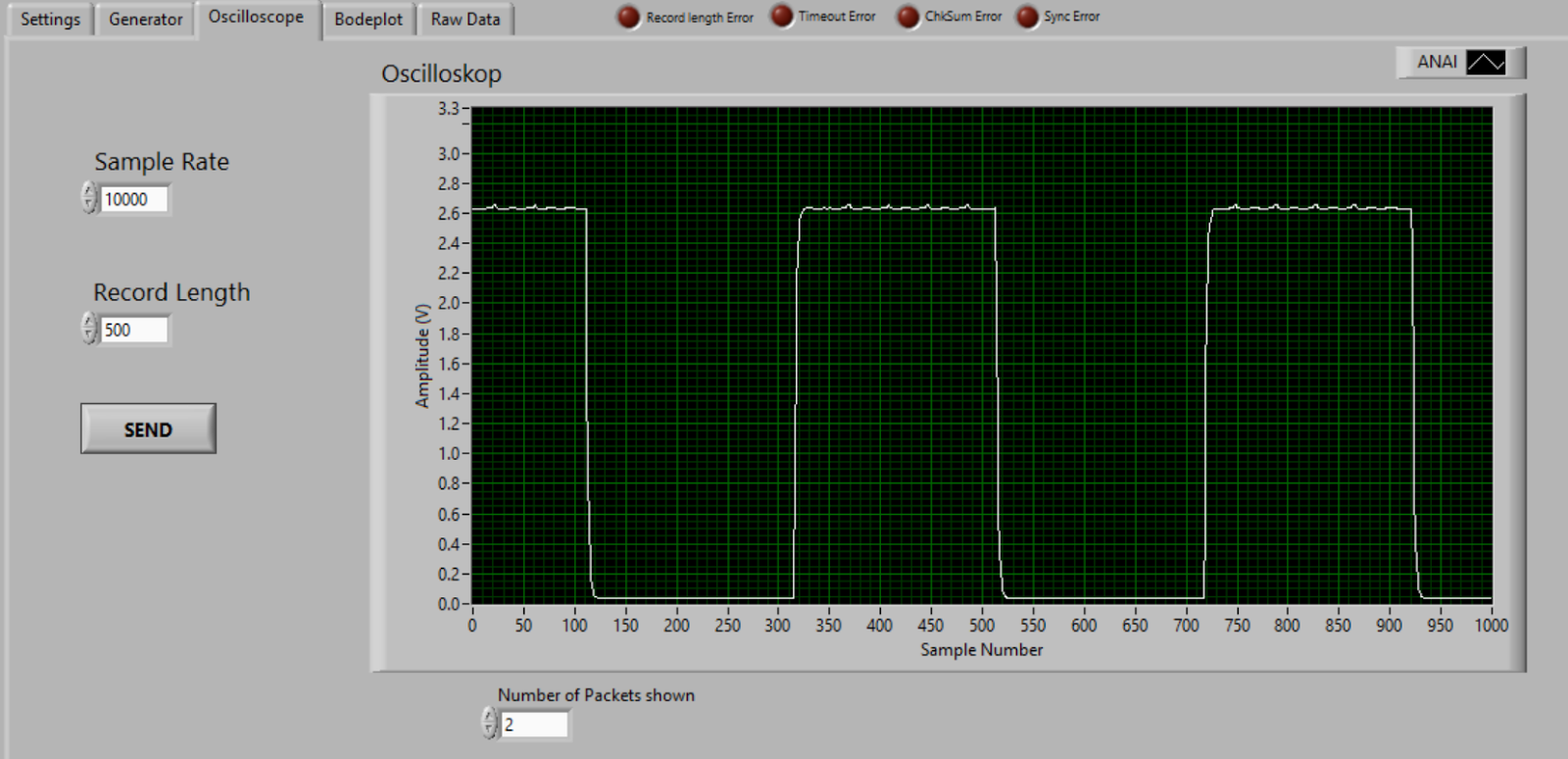
Figur 22 Flowchartet af Bodeplot

Figur 22 viser virkemåden af funktionerne. Bodeplot funktionen kalder amplitude 255 gange og lægger amplitude værdien i et array som senere bliver brugt til transmit af dataene. frekvensen bliver skiftet løbende undervejs. når alle frekvenser er gennemløbet blive transmit complete interrupt enablet og serviceroutinen dertil sørger for at sende dataen til labview.

# Test

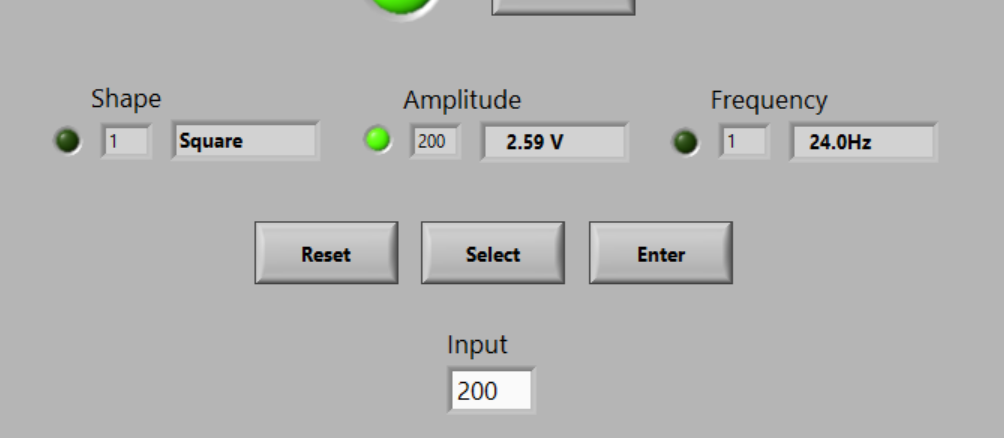
## Oscilloskop

Vi har testet oscilloskopet og det virker efter hensigt.

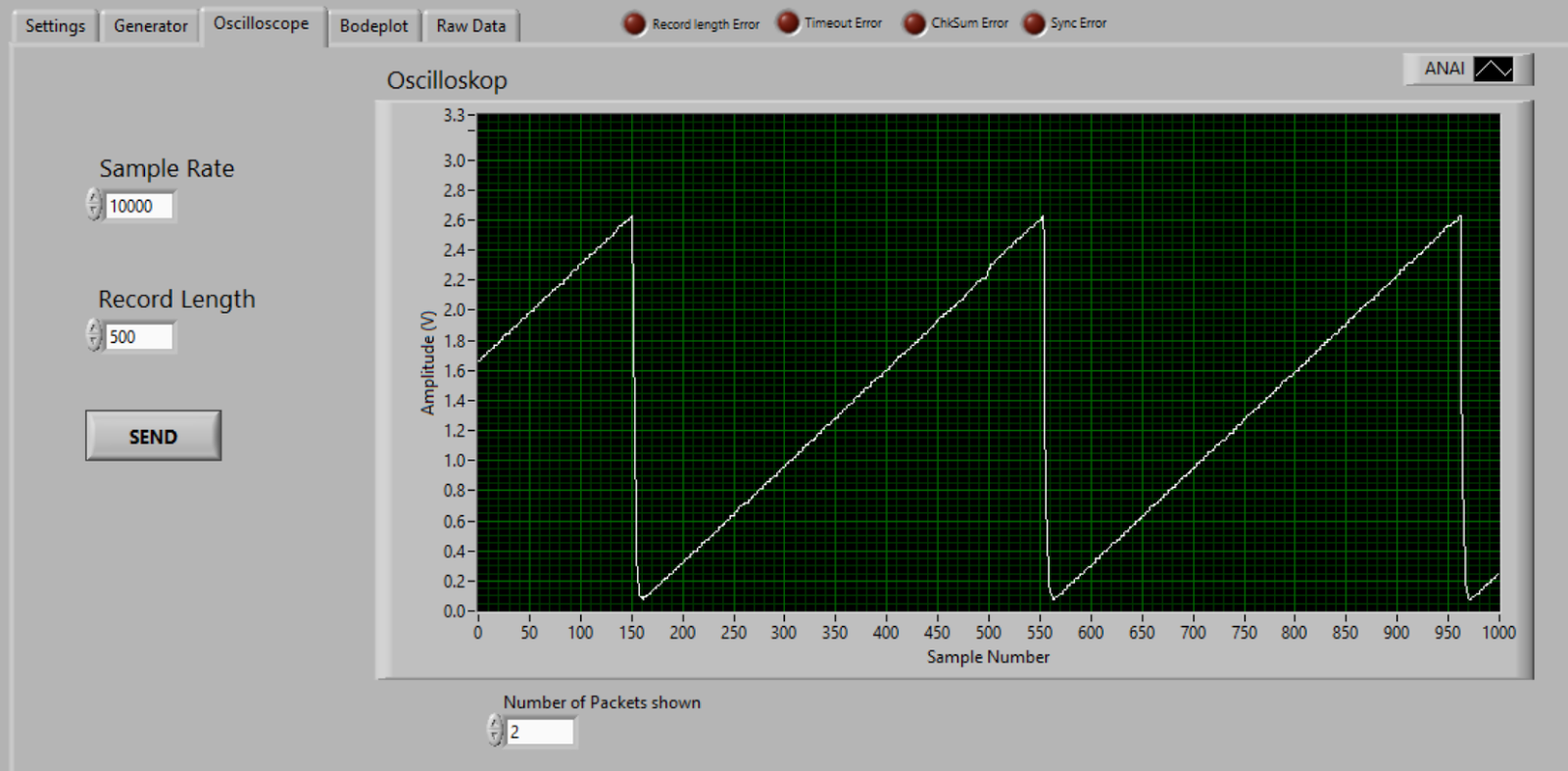


Figur 23 Udgangen på oscilloskopet

Figur 23 viser udgangen på oscilloskopet med nedenstående indstillinger.

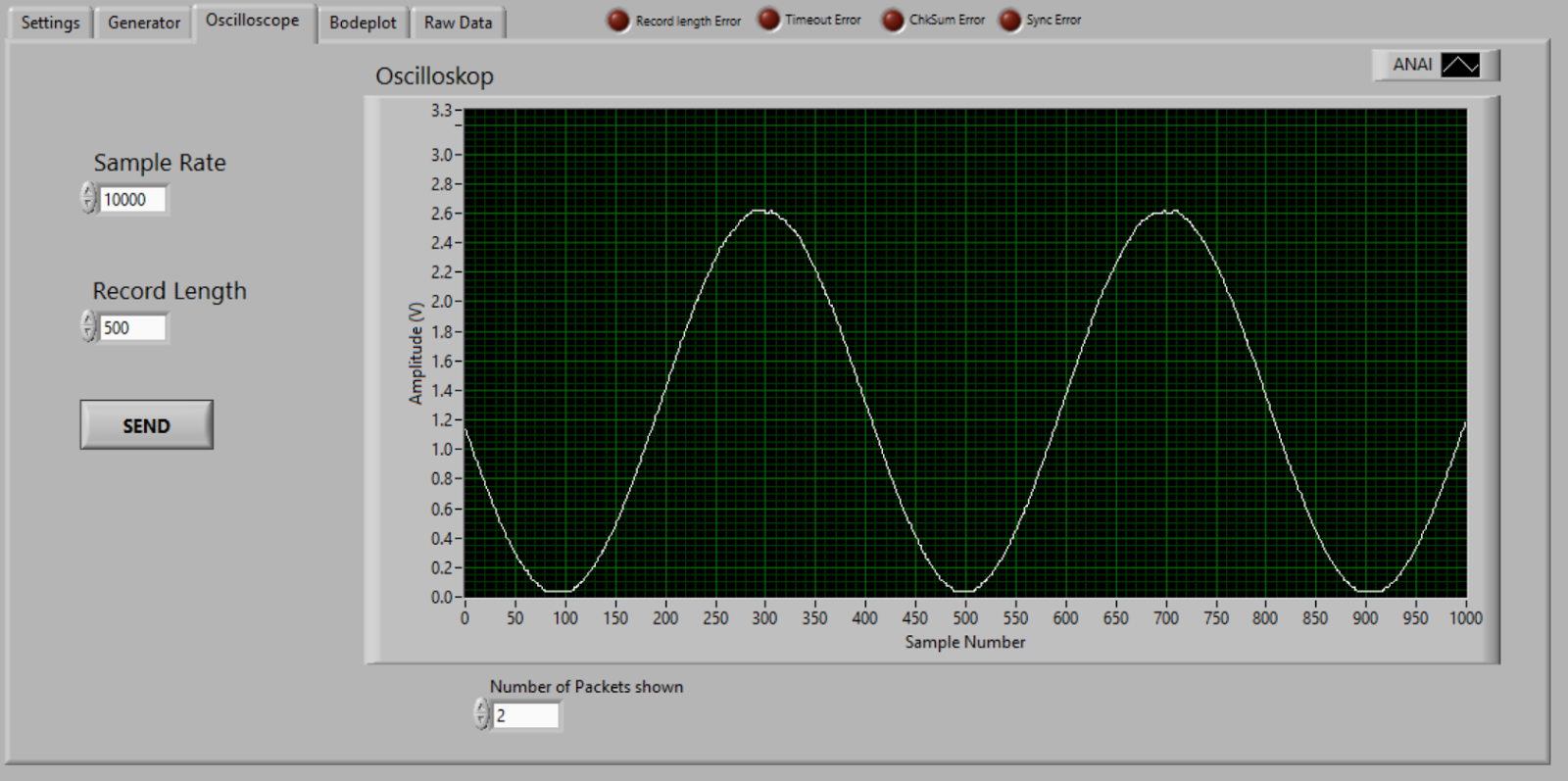


Figur 24 Indstillingerne i Labview



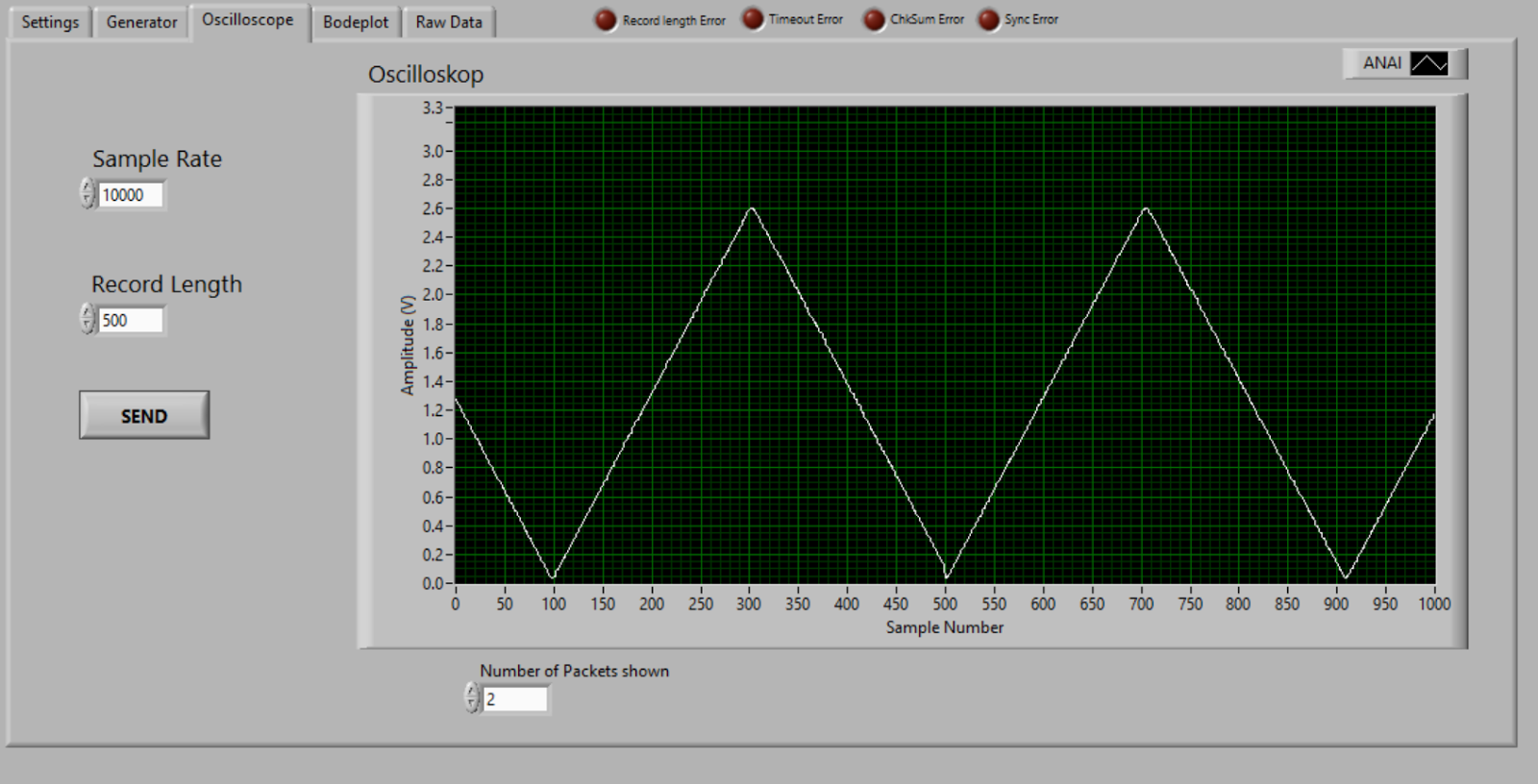
Figur 25 Savtak signal i Labview

På Figur 24 er signalformen ændret til savtak



Figur 26 Sinus signal i Labview

På Figur 26 er signalformen ændret til sinus



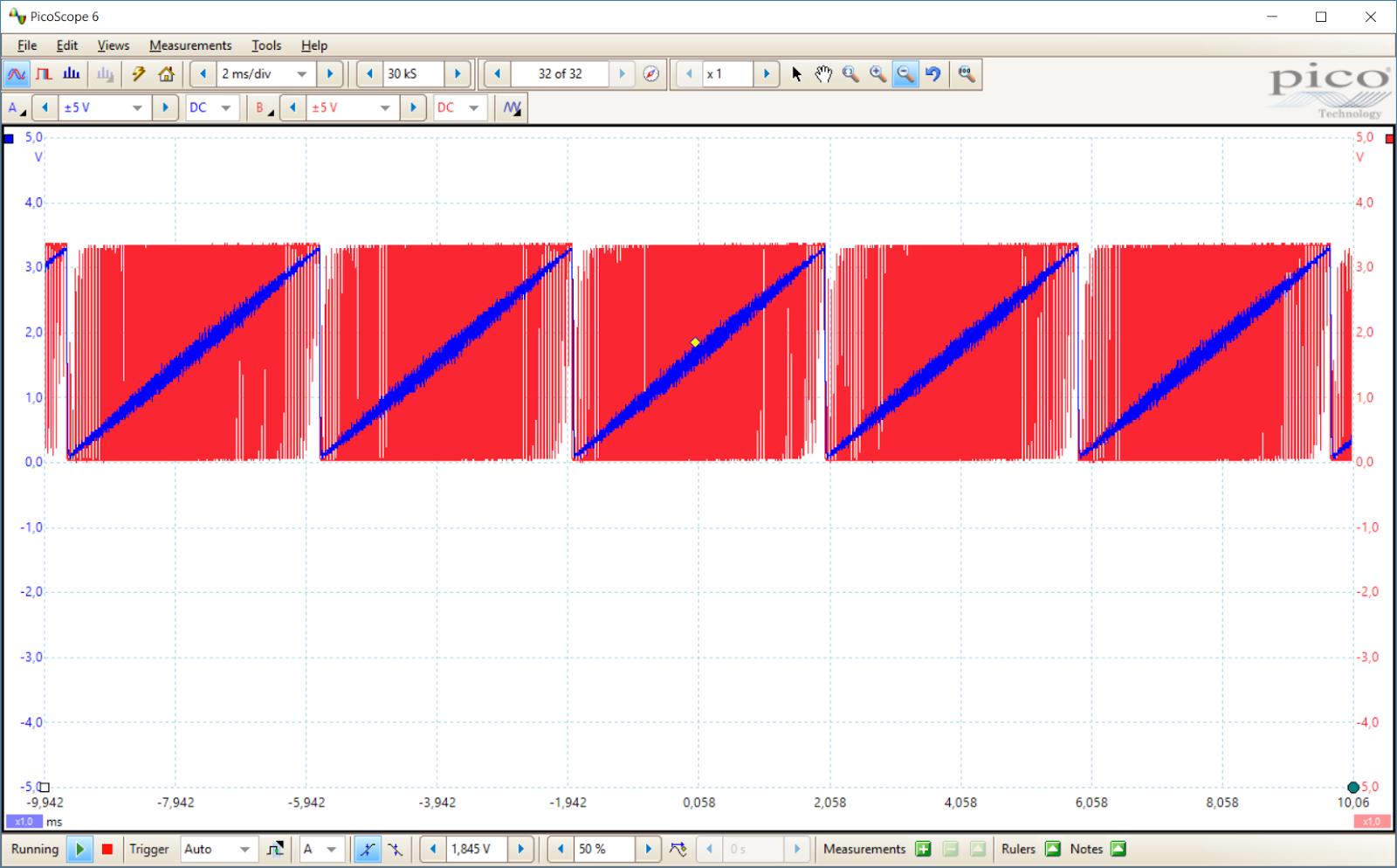
Figur 27 Trekant signal i Labview

På Figur 27 er signalformen ændret til trekant.

## Signalgennerator

### Analog Filtre

Vores gruppe har ved hjælp af signal generatoren på FPGA’en, kunne fremstille forskellige PWM signaler. Der var dog det problem, at det ikke var filtreret. Der skulle fremstilles et passende lavpasfilter. Vi startede med at lave et simpelt RC lavpasfilter. Der blev lavet nogle målinger for at se forskellen mellem de ufiltreret- og filtreret signaler. På Figur 28, ses forskellen på de ufiltreret og de filtreret signaler. Det filter som vores gruppe har brugt på nedenstående billede, er et 1. Ordens RC filter. Det er dog ikke vores endelige filter.

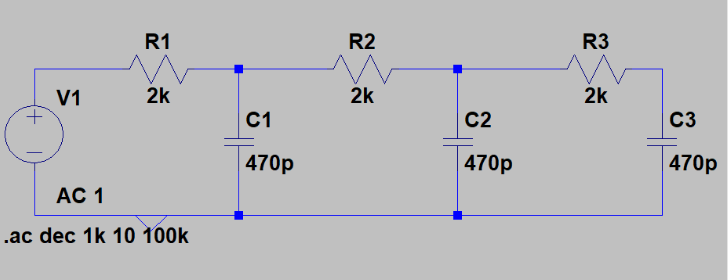


Figur 28 Forskel mellem ufiltreret og filtreret

Det røde er det ufiltreret og det blå er det filtreret. Det skal forestille en savtak signal. Det er muligt at se en tydelig forskel på det filtreret og det ufiltreret. Dog bliver det tydeligt at filtreret ikke frasortere alt støj.

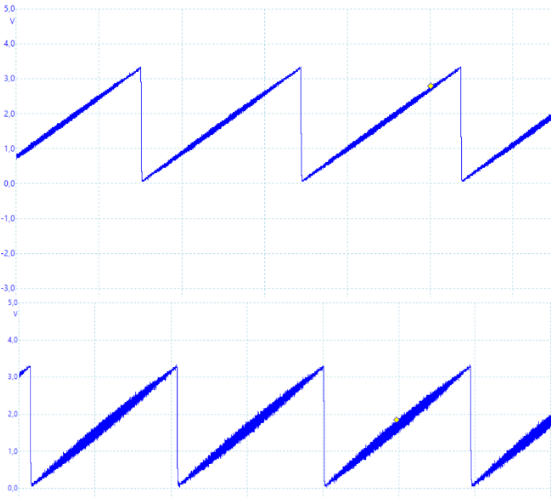
Det vil vores gruppe nemlig prøve at fixe, med et 3. ordens RC filter. Problemet med dette filter er når rangorden forøges, så rykker knækfrekvensen også. Knækfrekvensen startede på 24 kHz men efterhånden som rangorden forøges, blev knækfrekvensen også skubbet. Knækfrekvensen endte med at være på 33 kHz.

På Figur 29 ses kredsløbet af 3. ordens RC filter.



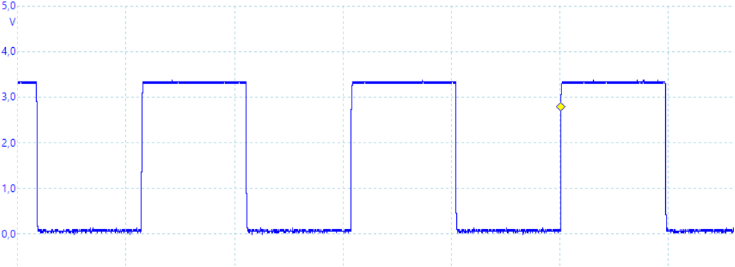
Figur 29 Kredsløbs diagram over filtret

Det var meget vigtigt at dæmpningen var god i filtrene. Der blev prøvet med flere forskellige ordner, men 3. ordens filter var den bedste. På Figur 30, ses forskellen mellem 3. ordens og 1. ordens RC filter. Den øverste er 3. ordens og den nederste er 1. ordens. Det er tydeligt at der er en forskel i støjreducering.



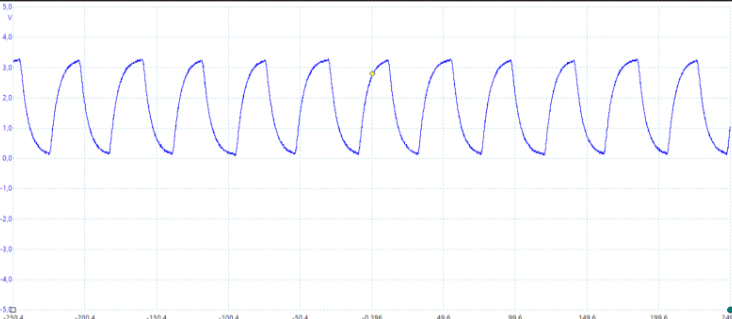
Figur 30 Forskel i filtrene

Vores gruppe prøvede også med firkant signalet, både ved lave frekvenser og høje frekvenser. Firkant signalet så rigtigt godt ud ved lave frekvenser, som man kan se på Figur 31.



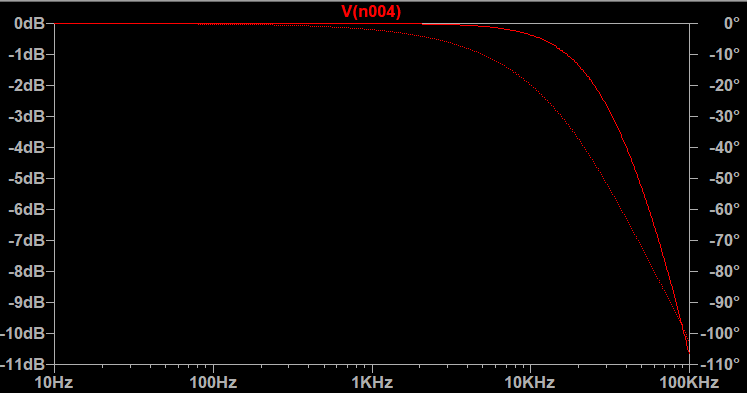
Figur 31 Firkant signal lav frekvens

Problemet opstår når frekvensen bliver for høj. På Figur 32 ses når frekvensen bliver for høj. Her er den maksimale frekvens blevet sat og filtret bliver sat på prøve.



Figur 32 Firkant signal med høj frekvens

For at kunne lave et godt 3. ordens filter, som tilfredsstiller gruppens krav, skulle filtret simuleres i LT-Spice. Derfra kunne der laves en Bodeplot analyse af de forskellige ordener. På Figur 33, ses den 3. ordens filter. Alle tre ordener, kan ses i Appendiks, hvor det er tydeligt at se en forskel på rangeringen af filtrene. Se Figur 44 Alle Bodeplot

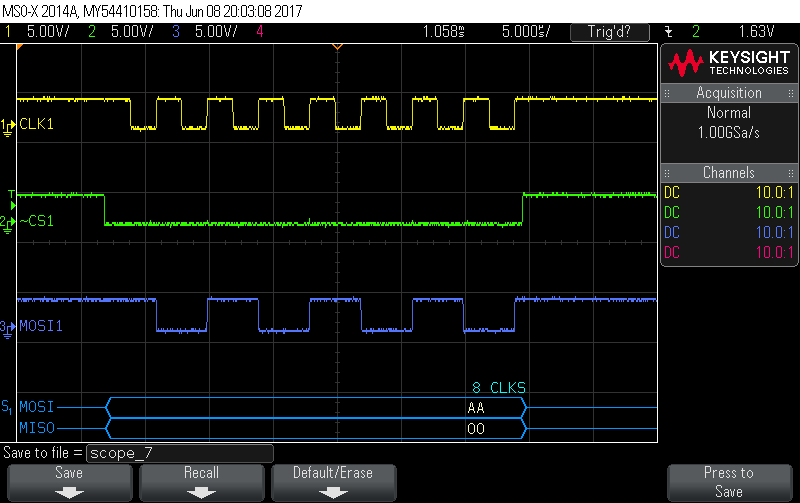


Figur 33 Bodeplot af filtret vi bruger

Vores gruppe kan sige at der er blevet fremstillet et passende lavpasfilter, som opfylder de givne krav.

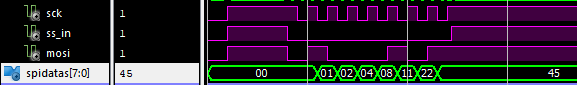
## SPI Test

I forbindelse med, at MCU’en og FPGA’ens SPI forbindelse skulle sættes sammen, skulle der laves test og simuleringer af hvordan SPI signalet opførte sig. På Figur 34 ses en af de første test, hvor det genkendelige bit mønster 0xAA eller 0b1010 1010 sendes fra MCU’en. Her ses det, at når SS går lavt betyder det, at MCU’en (Masteren) vil sende til FPGA’en (Slave) og på nedadgående flanke, bliver en bit læst.



Figur 34 Timingdiagram

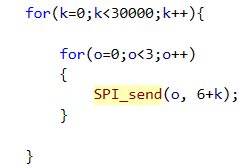
SPI signalerne skulle gruppen så simulerer i form af en testbench i VHDL, for at se om skifte registeret der var blevet lavet, modtog de 8 bits korrekt. Det kan ses på Figur 35, hvor SPIdata[7;0] er en databus der er udgangen på skifteregisteret. Det ses at skifteregisteret ændre outputtet hver gang SCK går lav og skifter de 0x45 eller 0b01000101 ind som er synkroniserings byten i vores SPI protokol.



Figur 35 SPIdata simuleret i testbench

### Test af sikker SPI forbindelse ved modtagelse

Gruppen skulle vise ved test, at der var blevet lavet en sikker SPI forbindelse. Det blev lavet ved, at modificere en kopi af vores færdige VHDL kode så der var en tæller der udskrev til displayet, hver gang der var en checksum fejl. Testen kunne laves ved at tilføje et for loop i c koden på MCU’en. Med en tælle variabel som kørte 30.000 gang, hvor den sendte forskellige adresser og data afsted hver gang, kunne vi vise en sikker SPI forbindelse da displayet rigtigt nok sagde der var 0 fejl. Den simple test funktion fra MCU’en ses på Figur 36, hvor SPI\_send() er en funktion der modtager (adresse, data) som parametre og sender en pakke afsted.



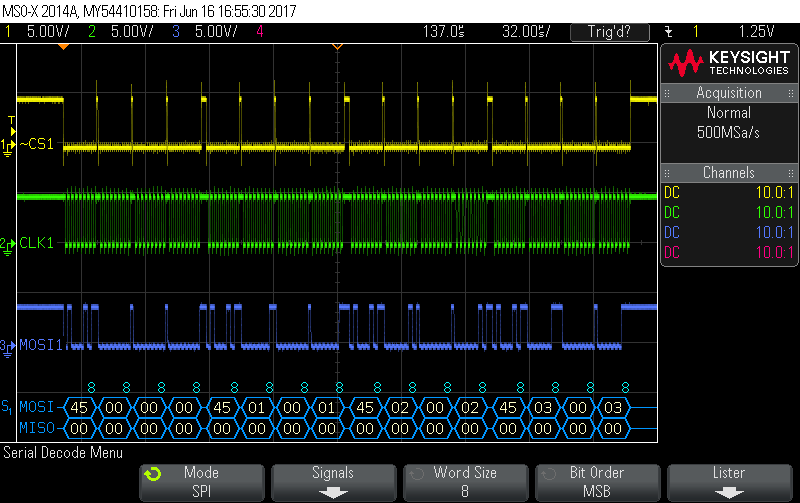
Figur 36 Test at overførsel

## Test af “indstilling af parametre fra Labview Programmet”

Test af Labview programmet skete ved brug af El labs Oscilloskopers SPI Decoder funktion. Derudover kunne den data der blev sendt aflæst via Uarten på terminalen og de ændrede parametre kunne så ses opdateret på FPGA’ens display.

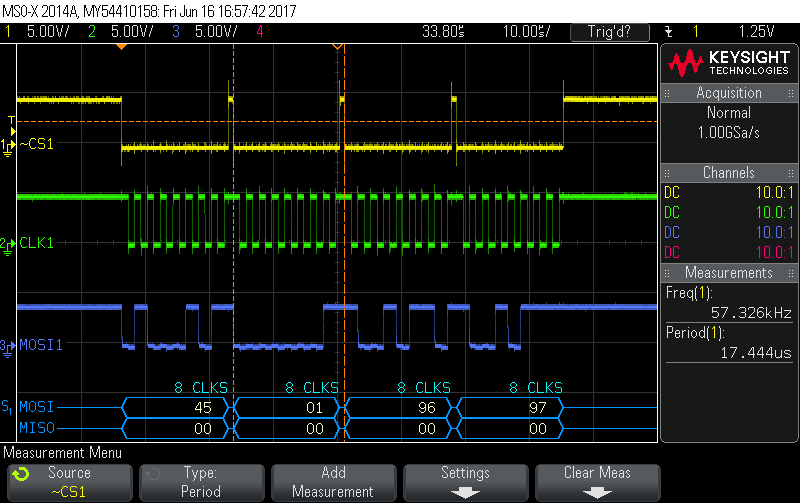
På billede Figur 37 er der blevet trykket på Reset i Labview og der ses, de 3 SPI signaler og det nederste signal, som er SPI decoderens output i hex. Der kan aflæses følgende udfra protokollen der er mellem MCU’en og FPGA’en:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sync byte | Adresse Byte | Data Byte | CheckSum | Funktion |
| 45 | 00 | 00 | 00 | Shape sættes til 0 |
| 45 | 01 | 00 | 01 | Amplitude sættes til 0 |
| 45 | 02 | 00 | 02 | Frekvens sættes til 0 |
| 45 | 03 | 00 | 03 | Signal Generatoren slukkes |



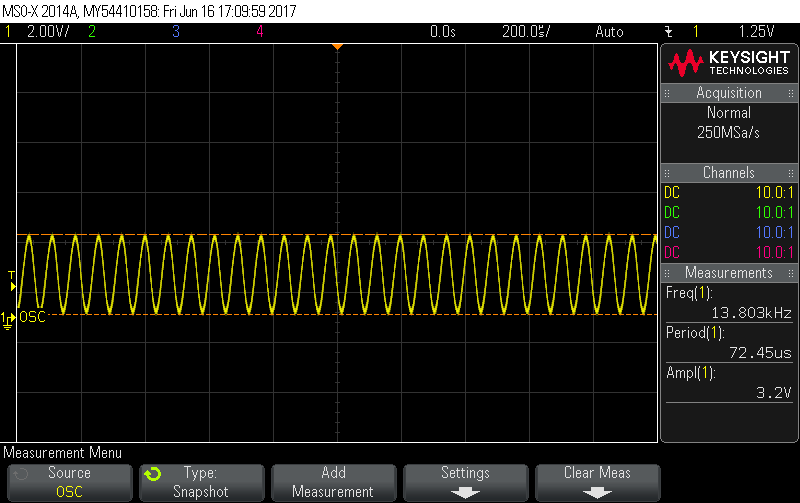
Figur 37 Timingdiagram SPI

Derefter sendte labview 0x01 som addresse, 0x96 som data og til sidst 0x01 xor 0x96 = 0x97 som checksum, for at sætte amplitude parameteren til 150 decimalt, hvilket også kunne ses på displayet på FPGA’en.



Figur 38 Timing diagram zoomet

Efter at have ændret parametrene et par gang og set FPGA displayet opdater med samme værdier, testes et sinus signal med parametrene: Trejde shape, 3.3 V amplitude og ved 13,8 kHz. Se Figur 39.



Figur 39 Test af signal

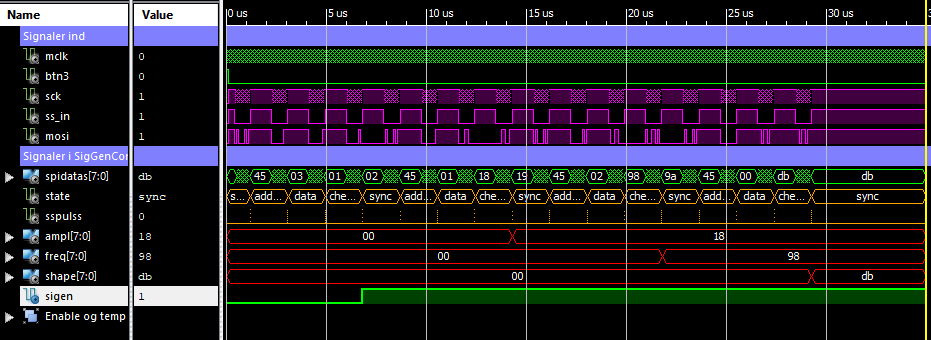
Det ser jo udmærket ud.

## Test af VHDL koden

I udviklingen af VHDL modulerne, er der løbende skrevet test benches og simuleret, hvordan systemet forfører sig i forskellige tilstande med forskellige inputs. Den første test, viser hele systemet der modtager data til at ændre alle 4 parametre. Signalerne er farvekodet og opdelt for at gøre det mere overskueligt at kigge på. De pink signaler i toppen, er de 3 SPI signaler, herefter kommer SPIdata[7;0] som viser byten der er modtaget. State viser, hvilken tilstand tilstandsmaskinen er i og SSpulss er signalet fra SS signalet der går fra lav til høj når en byte er transmitteret. De røde signaler er registrene, hvor det kan ses at den byte der bliver loaded ind når tilstandsmaskinen er i ‘data’ tilstanden er den værdi registrene antager. Til sidst er der sigen signalet som indikere at signal generatoren udsender et signal/er tændt. Se Figur 40

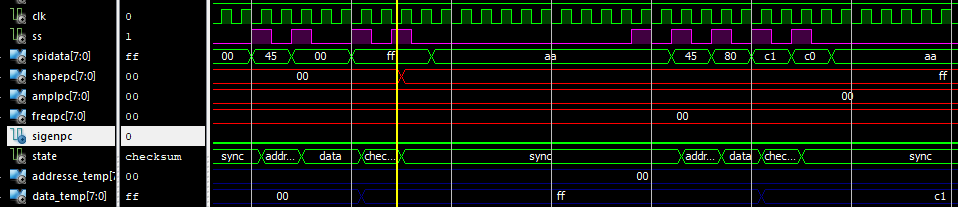
Bemærk den første pakke der bliver sendt: 45   03   01    02

Der skrives til ‘RunReg’ registeret på adresse 0x03, at den skal tændes 0x01.



Figur 40 Test bech til kontrol af koden

Den næste test, viser hvad der sker hvis der fra Labviews side skrives til en adresse der ikke findes. Her er tilføjet addresse\_temp og Data\_temp signalerne som holder på den midlertidige adresse og data. Første pakke skrives: “45  00   FF  FF” og det kan ses at addresse\_temp forbliver 0 og der efter “data” tilstanden bliver lagt “FF” ind i data\_temp registeret. Efter “Checksum” tilstanden med en korrekt udregnet checksum indsættes FF i ShapePC[7;0] registeret som det skal. Efter noget tid modtager SPIdata sync byten 0x45 igen og tilstandsmaskinen ved, at der er en pakke på vej. Dog modtager tilstandsmaskinen en ugyldig addresse og sætter derfor ikke noget ind i addresse\_temp. Efter checksummen bliver udregnet og er rigtigt ses det, at der ikke bliver skrevet til noget register og tilstandsmaskinen går tilbage til “sync” tilstanden, hvor den afventer sync byten. Se Figur 41



Figur 41 Kontrol ved fejl af VHDL kode

## Warnings når der Synthesizes

WARNING:Xst:2211 - “SigGenDatapath.vhd" line 50: Instantiating black box module <SinusLut>.

WARNING:Xst:2211 - "SigGenDatapath.vhd" line 52: Instantiating black box module <TriangleLUT2>.

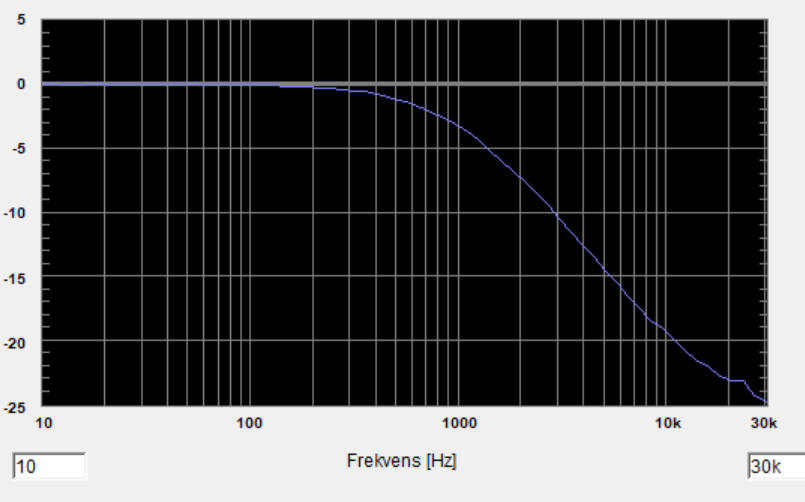
Phase 12  : 0 unrouted; WARNING:Route:455 - CLK Net:U4/Clk1\_D may have excessive skew because

Warning 1 og 2: Programmet advarer om, at “SinusLut” og “TriangleLut2” filerne bliver implementeret som black boks moduler, da filerne ikke er lavet udfra logik i VHDL, men er implementeret via Excel og de formler der er brugt.

Warning 3: Fejlkode der advarer om at timings i FPGA ’kan’ komme til at køre uoptimalt. Dog burde det ikke have nogen betydning for vores system. Ved højhastigheds applikationer der kræver præcis timings vil dette være et problem.

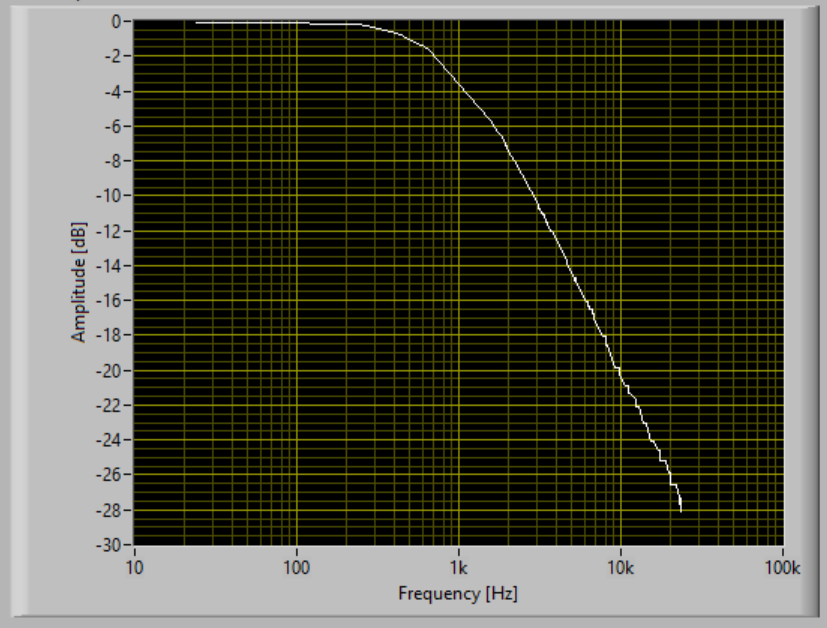
## Test af Bodeplot

Da Bodeplot funktionen skulle testes aktiverede vi funktionen via Labview og sammenlignede resultatet med hvad vi forventede at se. hvor at vide præcis hvad vi bør forventede benyttede vi DTU Ballerups elektronik laboratirieudstyr til at genrere et reference Bodeplot af filteret der testes og PWM-lavpasfilteret som helhed da dette tager fuldt højde for udgangsmodstanden i PWM-filteret. Vores reference Bodeplot så ud som følger. Se Figur 42.



Figur 42 Bodeplot på DTU udstyr

På nær det lille i slutningen af plottet repræsenterer dette plot den gyldne standard for hvad vores Bodeplot funktion skal kunne generere. Den endelige version af programmet genererer et Bodelplot der ser ud således.



Figur 43 Bodeplot i Labview

På Figur 43 viser Bodeplottet samme amplitude respons som DTU’s meget dyre udstyr fra Keysight. Test af Bodeplot er hermed en succes.

# Konklusion

Gruppen har fremstillet et digital system til aflæsning og generering af forskellige signaler. efter test kan det konkluderes at de implementerede funktioner virker efter hensigten og overholder alle tilhørende kravspecifikationer. Til videreudvikling kunne man implementere spi to vejs kommunikation evt. handshake med dertilhørende protokoller.

Formålet med dette projekt er at lave et system som består af et oscilloskop og en signalgenerator. Det som vores system skal bruges til, er at kunne generere og måle på forskellige typer signaler - med forskellige amplitude og frekvens. Vores kode er delt op i C kode og VHDL kode. Koden er skrevet modulbaseret, hvilket gør at kodestumper/moduler kan bruges til andre spændende projekter som indebærer nogle af de elementer som projektet har bestået af fx inden for datakommunikation, protokoller, databehandling og signal generering. Med projektoplægget har gruppen fået udleveret allerede skrevet kode, både til FPGA’en og Labview for, at kunne komme i gang.

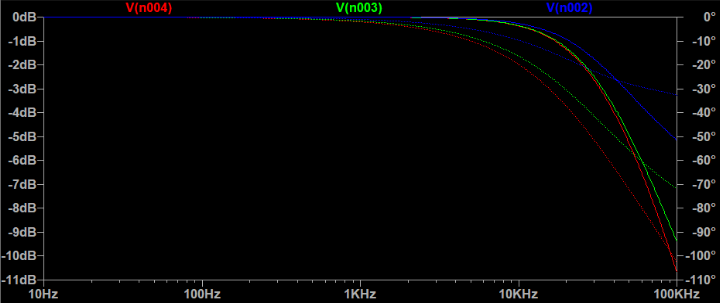
# Appendiks

Bilag filter

Rød= 3. orden

Grøn= 2. orden

Blå= 1. orden

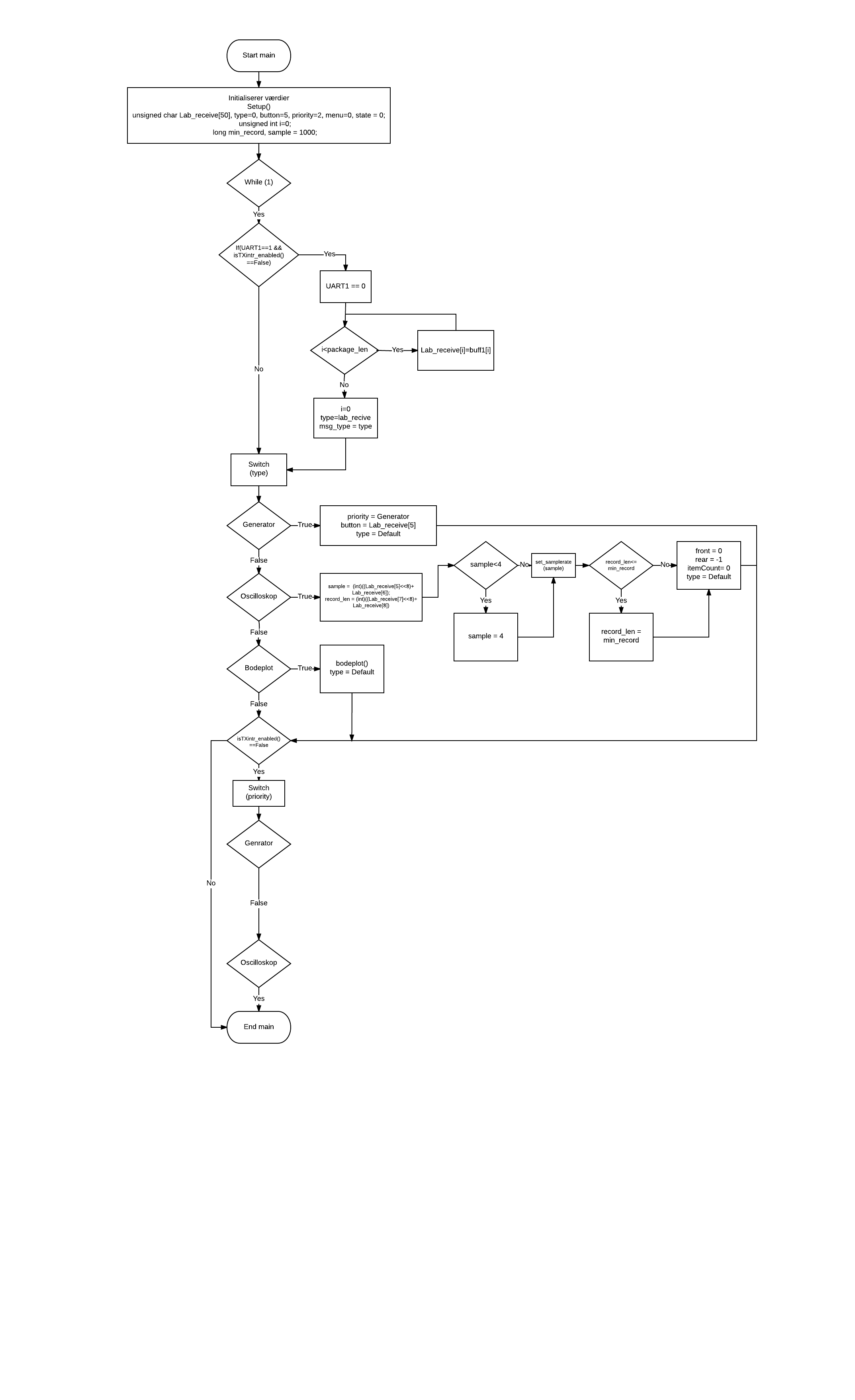


Figur 44 Alle Bodeplot

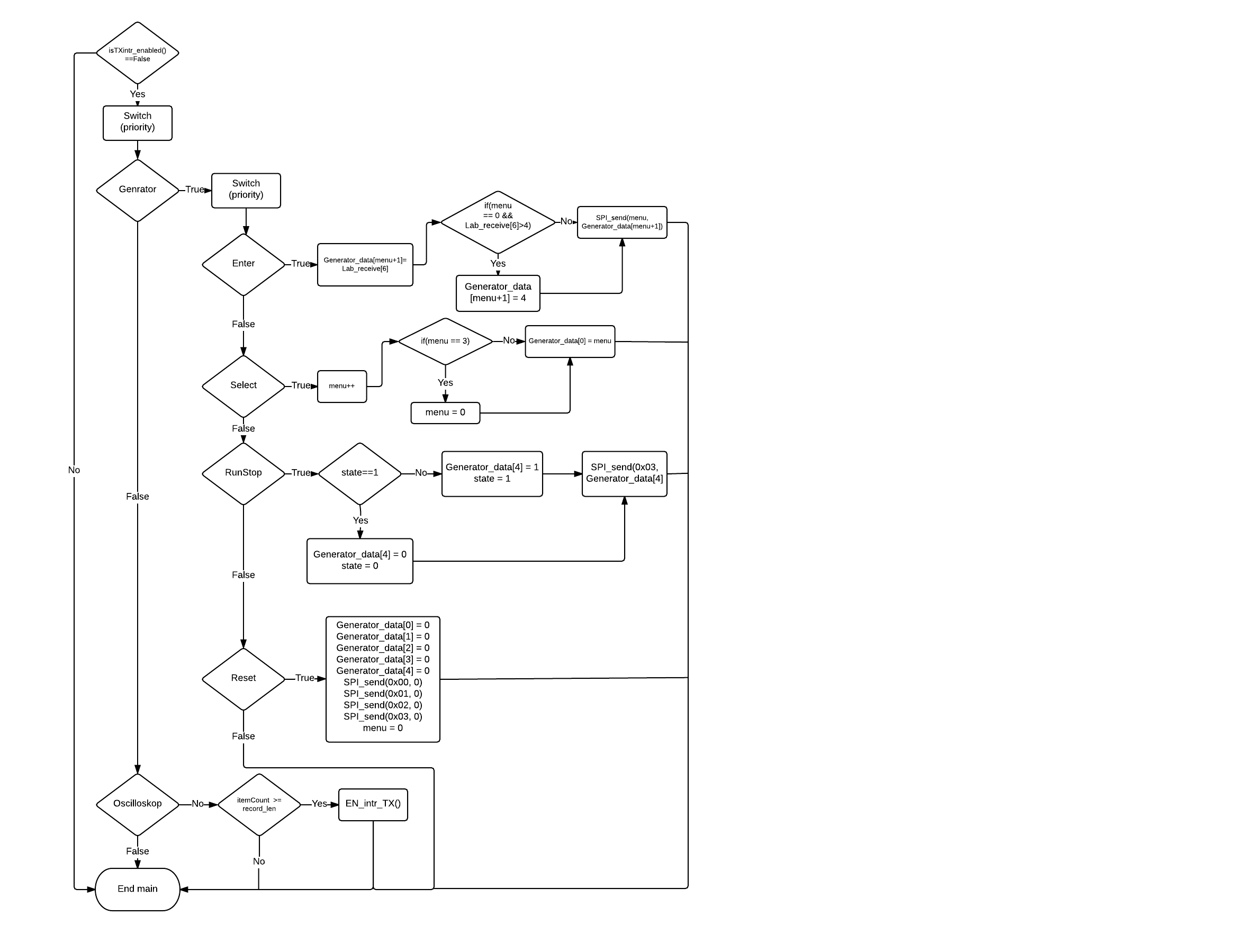
## Flowchart over ProControl Tilstandsmaskine



## Del 1 af main Flowchart



Del 2 af Main



## VHDL og C kode