Oscilloskop projekt

August 2020

UDARBEJDET AF PROJEKTGRUPPE 1:

Hold A:

Kim H. Christensen - s181554

Jørgen D. Greve - s181519

Hold B:

Mads Ludvig Stender - s195130

Esben Leerbeck - s195131

DTU - DIPLOM

30082 - Projektarbejde i Digitaldesign

Indholdsfortegnelse

1 Indledning

1.1 Baggrund

Kurset 30082 Projektarbejde i Digitaldesign går ud på at designe og bygge et mindre digitalt system ved hjælp af en PC, et Arduino ATmega 2560 MCU board samt et Digilent Basys 2 FPGA board. Til det har vi brugt software værktøjer som Atmel studio, ISE Design Suite, Digilent Adept og LabView. Atmel Studio er blevet brugt til at programmere MCU boardet i C programmeringssproget. Digilent Adept er blevet brugt til at programmere FPGA boardet i programmeringssproget VHDL, hvor koden er blevet skrevet i ISE Design Suite og LabView har kørt et på forhånd udleveret Oscilloskop program på en PC. Af hardware værktøjer har vi brugt et Digilent Analog discovery 2 oscilloskop og breadboards. Oscilloskopet er et USB oscilloskop der kobles til en PC eller Mac hvor man gennem programmet Waveforms kan bruge det både som oscilloskop, men også som signalgenerator, voltmeter mfl. Breadboards er blevet brugt til at forbinde systemerne samt til opbygning af det analoge PWM filter.

1.2 Opgaven

Opgaven går ud på at designe et oscilloskop, der skal kunne måle en spænding mellem 0V - 3.3V og omsætte det til et digitalt signal der kan vises som graf i et program på en PC. Spændingen der skal måles på genereres af et PWM output fra en FPGA chip. Spændingen filtreres i et analogt filter og konverteres herefter til en digital værdi vha. en MCU. Den digitale værdi sendes derefter til PC'en. På PC'en kan man endvidere regulere forskellige parametre på både FPGA chippen og MCU'en.

1.3 Kravspecifikation

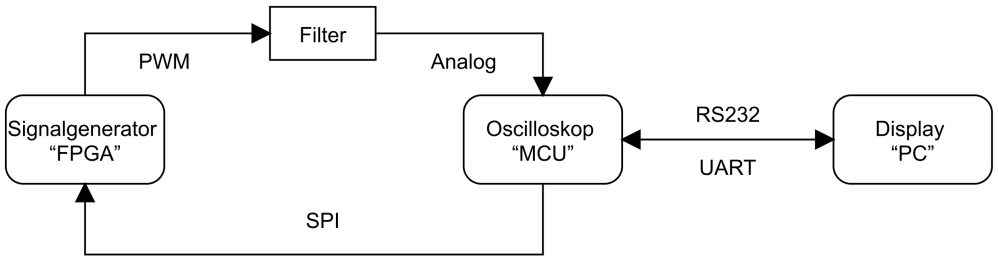
1.3.1 Oscilloskop kravspecifikation

ADC konvertering	Den analoge spænding fra signalgeneratoren skal måles fra 0 til 3.3 V med en opløsning på 8 bit.
Dataintegritet	Oscilloskopet skal ved alle nedenstående indstillinger kunne køre med kontinuert ubrudte målinger.
Parametre	Oscilloskopets samplerate og Record length skal kunne indstilles fra Labview programmet.
Samplerate min	Oscilloskopet skal kunne køre ned til 10 sps.
Samplerate max	Oscilloskopet skal kunne køre op til 5.000 sps. Må gerne køre op til 10.000 sps
Record length min	Oscilloskopet skal kunne køre med ned til 10 ADC målinger i hver pakke. Den minimale tilladelige record length skal tage højde for sampleraten.
Record length max	Oscilloskopet skal kunne køre med op til 1000 ADC målinger i hver pakke for alle samplerate.
RS-232 baudrate	RS232 forbindelsen skal køre med en baud rate på 115.2 kbaud
RS-232 håndtering	Modtagelse af data fra LabView programmet skal foregå ved interrupt. Transmission kan foregå ved polling eller interrupt.

1.3.2 Signalgenerator kravspecifikation

PWM filter	Der skal designes et lav-pas filter der på passende vis udglatter de digitale PWM pulser.
Parametre	Signalgeneratorens signalform (SHAPE) , amplitude (AMPL) og frekvens (FREQ) skal kunne indstilles fra Labview programmet. SHAPE, AMPL og FREQ kan gøres synligt på syv segment displayet.
Sinus signal	Der kan implementeres en look-up tabel i VHDL koden der gør det muligt at signalgeneratoren kan lave et sinus-formet signal.
SPI baudrate	SPI forbindelsen skal køre med en baudrate på 500 kbaud.
SPI håndtering	To-vejs SPI kommunikation kan implementeres f.eks. med acknowledge hand-shake.
SPI protokol	Der skal vælges og implementeres en robust byteorienteret protokol til at overføre SHAPE, AMPL og FREQ.
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.
PWM signal	PWM skal maksimalt køre med 10 kHz da ADC'en maksimalt kører med 10 ksps.

2 Design



Figur 1: Oscilloskop blokdiagram

Overordnet set er oscilloskop projektet sådan sat op, at FPGA boardet genererer et PWM signal, der passerer et analogt filter for at blive konverteret fra analogt til digitalt signal i MCU'ens ADC. Herefter sendes MCU'ens digital konverterede data via UART til programmet LabView, der kører på en PC. LabView viser herefter data som en graf på PC'en. Man kan via LabView sende instrukser om samplerate og record length til MCU'en via UART og om signalform, amplitude og frekvens gennem MCU'en til FPGA boardet via en SPI forbindelse.

2.1 Oscilloskop

MCU'en modtager en analog spænding fra 0 - 3.3V på ADC0 (Pin A0/97) hvorefter MCU'en vha. ADC, konverterer den analoge spænding til en digital værdi mellem 0 - 255. ADC'ens samplerate styres af MCU'ens timer/counter 1 (herefter "timer 1") ved at ADC'en konverterer en sample hver gang timer 1 genererer et interrupt og kører interrupt service routinen TIMER1 COMPB. Timer 1 er sat op til clear timer on compare match (CTC mode) således at timeren tæller fra 0 - TOP (TOP = OCR1A = 0 - 65535) hvorefter der generers et interrupt og timeren cleares og starter forfra.

Timer 1 frekvensen eller ADC sampleraten kan beregnes ud fra følgende formel hvor F_{CLK} er system clock frekvensen, N er ADC prescaleren og $OCRnA$ er TOP værdien.

$$f = \frac{F_{CLK}}{N \cdot (1 + OCRnA)}$$

For at kunne opnå 10 ksps er der blevet valgt en prescale værdi (N) på 64. Man kunne også have valgt 256, men i så fald kommer TOP værdien ret langt ned for at opnå en frekvens på 10 kHz.

For at oversætte samplerate outputtet fra LabView som er en værdi fra 10 til 10.000 til en ny TOP værdi til timer 1, som passer med den ønskede samplerate var det nødvendigt at opfinde en formel der kunne klare dette. Med den valgte prescaler skulle 10 sps fra LabView blive til en TOP værdi på 24.999 og samtidigt skulle 10.000 sps fra LabView blive til en TOP værdi på 24. Det viste sig at følgende formel kunne udføre den ønskede beregning.

$$TOP = \frac{250.000}{sps} - 1$$

ADC output værdien kan beregnes ud fra følgende formel hvor V_{IN} er input spændingen (0 - 3.3V), V_{REF} er referencespændingen og 256 er fordi der er tale om en 8 bit konvertering.

$$ADC = \frac{V_{IN} \cdot 256}{V_{REF}}$$

V_{REF} er valgt til at være 3.3V, hvorfor AREF er forbundet til MCU boardets 3.3V forsyningspin samt en kondensator til GND for at eliminere støj.

Forbindelsen mellem MCU'en og Labview foregår ved hjælp af seriel kommunikation, i projektet er der benyttet UART (Universal Asynchronous Receiver Transmitter).

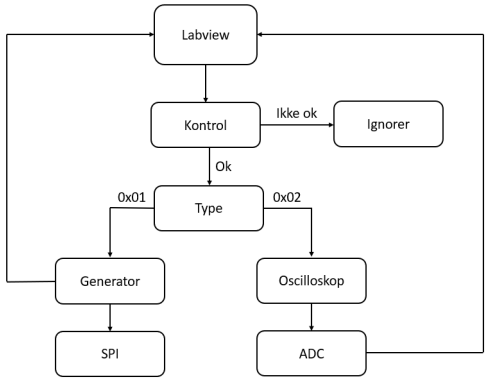
UART er konfigureret til:

- BAUD-rate = 115.2k.
- UBRRn = 16 (16.36).
- Full duplex, USCRnA = 1.
- 8 data bits, UCSZn1 = 1 og UCSZn0 = 1.
- 1 stop bit, USBSn = 0.
- Parity disabled, UPMn1 = 0 og UPMn0 = 0.
- Receive complete interrupt, RXCIEn = 1.

UBBRn er udregnet efterfølgende formel - MCU'en kører med en CPU-clock på 16 MHz:

$$UBBR = \frac{f_{OSC}}{8 \cdot BAUD} - 1$$

I ATMega-2560 databogen kan man på side 231, tabel 22-12, læse at overstående UBBR værdi har en fejlprocent på 2.1% med en CPU-clock på 16 MHz.
Diagrammet viser et simplificeret billede at forløbet for data mellem Labview og MCU'en.



Figur 2: Data flow

For at tilsikre det sendte data kan blive behandlet på ønskelig vis, bliver der sendt datapakker mellem MCU'en og Labview. Nedenstående tabel illustrer hvad disse datapakker består af:

Sync	Length	Type	Data	Checksum
2 bytes	2 bytes	2 bytes	X bytes	2 bytes

- Sync består altid af de to bytes "0x55AA", ofte i programmeringen er den blevet brækket op i enkelte bytes "0x55" og "0xAA".

- Length består også af to bytes, det er kun nødvendit at bruge to bytes for at sende datapakker indeholdende ADC målinger, da alle andre pakker har en længde der kan beskrives på en byte.

- Type kan enten være "0x01" eller "0x02" og henviser henholdsvis til pakker til/fra generator- eller oscilliskop fanebladet i Labview. Typen bliver således brugt af MCU'en til at identificere hvor data kommer fra og hvor data bliver sendt hen.

- Data består af det data der bliver sendt mellem MCU'en og Labview. Denne mængde kan variere fra 2 bytes og helt op til 1000 bytes.

- Checksum er en kontrolfunktion der minimerer risikoen for fejl i data under transmisionen. I dette projekt bliver der brugt LRC-8 hvilket er en "Longitudinal redundancy check".

Da det skal tilsikres at MCU'en modtager en hel datapakke før den begynder at behandle det modtagne data bruges der et UART recieve complete interrupt. I interrupt rutinen benyttes det at længden på pakken kan læses, hvorfor en for-løkke modtager netop de bytes som datapakken består af. Hver gang en byte er modtaget bliver denne gemt i en buffer. Derudover sættes et flag højt som initialiserer behandlingen af en modtaget pakke.

Behandlinen foregår således, i datapakken bliver det kontrolleret om sync er korrekt og der foretages en test af om checksummen stemmer overens med den modtagne checksum.

Udregningen af checksum foregår ved at hver byte, i datapakken, bliver XOR'et med den foregående.

Sync	0x55	0	1	0	1	0	1	0	1
Sync	0xAA	1	0	1	0	1	0	1	0
Length	0x00	0	0	0	0	0	0	0	0
Length	0x09	0	0	0	0	1	0	0	1
Type	0x01	0	0	0	0	0	0	0	1
Data	0x01	0	0	0	0	0	0	0	1
Data	0x02	0	0	0	0	0	0	1	0
Checksum	0x00	0	0	0	0	0	0	0	0
Checksum	0xF4	1	1	1	1	0	1	0	0

Tabellen viser et eksempel på en datapakke og skal for de to første rækker fra venstre skal de læses lodret, de resterende kasser viser den binære værdi at hexadecimal tallet i række to. Hvis man vil kontrollere om checksum er korrekt skal man først tælle de mindst betydende binære værdier lodret, checksum skal ikke tælles med, men viser hvilken værdi udregningen giver.

Hvis tallet er lige skal pågældende ciffer i checksum være 0 og hvis det er ulige skal det være 1.
På den måde kan det hurtigt kontrolleres om checksummen er korrekt.

Dog har det sine svagheder, for eksempel vil værdien nul ikke være medregnet da den ikke ændrer på det talte antal cifre. Desuden vil den heller ikke registrere hvis der er bit-fejl i et byte hvis der er endnu en bit-fejl i et efterfølgende byte, da det her vil resultere i det samme ciffer i checksummen.

Såfremt test af checksum accepteres vil pakken blive behandlet, type bestemmer om pakken skal behandles som modtaget fra generator fanebladet (0x01) eller oscilliskop fanebladet (0x02).

I generator fanebladet bruges de to data bytes der bliver modtaget med datapakken. Den første indikerer hvilken knap der er blevet aktiveret og den anden byte indeholder den, i Labview, indtastede switch værdi. Først vil fire if-statements afkode hvilken knap der er blevet trykket på og en efterfølgende switch-case vil afgøre hvad der skal ske med den modtagne switch-værdi:

- For "Enter" knappen vil den aktive LED (Har 0x00 som start værdi) i Labview afgøre hvilken parameter der arbejdes med, derfor er denne case opbygget af tre if statements der afgør hvilken tilstand den aktive LED har og om den indtastede værdi er indenfor grænserne af hvad tilstandene kan håndtere. Såfremt dataen ligger indenfor de acceptable værdier vil talværdien blive sendt videre til FPGA boardet vha. SPI. Her bliver de indtastede værdier også gemt på MCU og sendt retur til Labview hvor de vil opdatere sidepanelet med den konverterede switch værdi.

- For "Select" knappen vil den indtastede switch værdi (0x00 - 0x02), en if-else statement og en switch case afgøre om det er henholdsvis Shape, Amp eller freq der skal være aktiv. Igen vil der sendes en værdi til FPGA vha. SPI, dog er værdien ikke væsentlig da denne blot aktiverer den valgte tilstand på FPGA boardet.

- For "Run/Stop" knappen bliver der sendt en værdi til FPGA vha. SPI, igen her er værdien underordnet vi har blot valgt at det skal være switch værdien der bliver sendt. Da knappen skal kunne starte og stoppe FPGA, er der blevet benyttet en variabel "RS_flag" der skifter mellem nul og et for hver gang "Run/Stop" bliver aktiveret. Det fungerer på den måde at hvis flaget er 1 tolkes det som at processen er startet, her vil det ikke være muligt at indtaste nye værdier eller ændre tilstand for den aktive LED. Hvis falget er 0 tolkes det som at processen er stoppet og man har altså mulighed for at indtaste nye værdier inden endnu en gennemkørsel. - Den eneste knap der kan bryde ud af en Run cyklus er "Reset" knappen.

- For "Reset" knappen bliver variablerne nulstillet og sidepanelet i Labview bliver opdateret. Derudover bliver de nye værdier (0x00) for Shape, Amp og Freq sendt enkeltvis efterfulgt af send for på den måde at opdatere hver enkelt på FPGA.

I oscilloskop fanebladet bliver der sendt fire bytes data. De første to bytes indeholder den valgte sample rate og de sidste to indeholder record length. Grunden til at de bliver sendt som to bytes hver er, at det kun er muligt at sende 8-bit med UART. Derfor bliver eks. 0x2710 (10.000 dec) sendt som 0x27 i første byte og 0x10 i sidste byte.
Derfor skal der foretages en sammenklistring af de to sample rate- og record length-bytes. Efter dette bliver sample rate brugt til at opdatere sample hastigheden på ADC'en og record length styrer hvor store pakker der skal sendes retur til Labview.

For at håndtere det data ADC'en sender, benyttes et interrupt der bliver aktivt hver gang der en en sample klar. Derefter bliver det aktuelle sample gemt i en buffer. Når denne buffer er fuld vil en datapakke, af en størrelse Record bestemmer, blive sendt til Labview hvor oscilliskop displayet bliver opdateret.

2.2 Signalgenerator

FPGA bordet bruges som signalgenerator ved at få indstillet `Shape, Ampl, Freq` og når `SigEn` (run) signalet går højt, vil der sendes et PWM signal ud i den ønskede form, amplitude og frekvens. Dette indstilles gennnem en SPI forbindelse som der bliver sendt fra MCU'en gennem et 8-bit MOSI-signal. Protokollen, der er bliver brugt, er som følger, først kommer der en synkronisering-byte, der sørger for at det der bliver sendt igennem MOSI ikke bliver puttet ind i et forkert register, ved at tjekke om byten, der bliver sendt har den ønskede byte værdi. På denne måde sikre det, at tilstandsmaskinen i FPGA boardet starter det samme sted hver gang.

Derefter i protokollen kommer adresse-byten, der fortæller tilstandsmaskinen hvilke registrer, der skal skiftes til eller gerne vil ændre værdien i. Hvis værdierne ikke skal ændres, sendes adresse-byten med en værdi der fortæller programmet at den skal kører og dermed gerne vil sende et signal ud igennem PWM. Efter der er fortalt hvilke registre der ønskes ændringer i, kommer den del af byteorienterings protokollen, data-delen, der fortæller hvad værdien i det ønskede register skal ændres til.

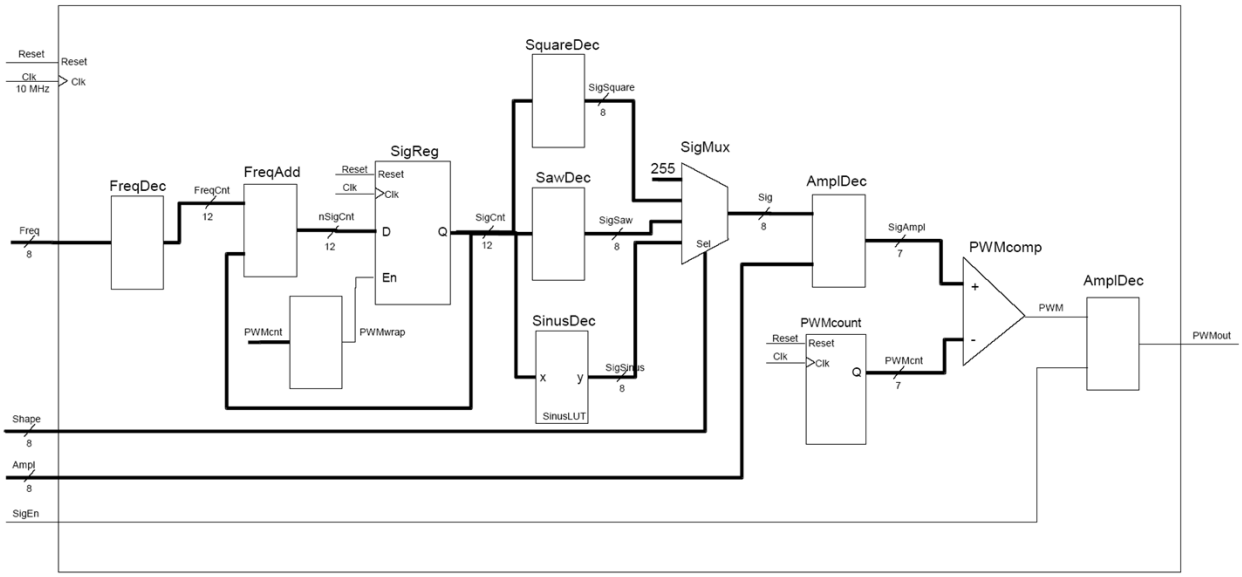
Til sidst kommer den vigtigste del er protokollen som er checksum. Hvis værdien, der kommer ind i checksum registret ikke stemmer overens med det som der allerede er beregnet fra de 3 andre registrer, vil ingen registre blive åbnet for og intet signal vil blive sendt ud igennem PWM, hvorimod hvis MOSI stemmer overens med den allerede er beregnet checksum vil der blive skabt et PWM signal med respektive ønskede værdier fra hver af registerne som der kommer ud til et lav-pass filter, hvor det så samlet fungerer som et DA-converter. Det konverterede signal sendes til MCU'en, som konvertere det analoge signal til et digital signal.

Sync	Adr	Data	Checksum

Grunden til der er valgt denne protokol, vist ovenfor, er fordi at med en synkroniserings byte som der sikrer at man altid starter med den rigtige tilstand og med adressen byte hvor den kan skifte imellem mange forskellige registre igennem en tilstandsmaskine hvor det dermed bliver nemmere at styre hele FPGA programmet fra Labview og at kunne få de er rigtige værdier ud til registerne; `Shape, Ampl, Freq` og `SigEn` (run), i data delen af protokollen. Til sidst er der en checksum byte som sikre at der ikke er sket fejl under overførslen mellem MCU og FPGA.

For at forstå SPI-forbindelsen skal man vide at den består af 3 ting; en clock, MOSI-signal og et SS-signal. Clock signalet er intuitivt at forstå da alt den sørger for er at kunne synkronisere master og slave, MOSI står for master out slave ind altså det er master-blockens output der bliver sendt hen til slavens input altså det

er den der fortæller hvad slaven skal ændre dette gør den igennem et 8 bit signal. Til sidst er der SS-signalet dette er signalet der fortæller om master-blocken er færdig med at overføre data, dette signal er aktiv lav.



Figur 3: SigGenDataPath Module

Denne del af VHDL koden står for at lave PWM-signalet, dette gør den ved at få de 4 indtastede værdier fra `Shape, Ampl, Freq` og `SigEn` (run). Med frekvensen kan der ses at den går ind i en dekoder der laver et 8 bit signal om til et 12 bit signal det 12 bit signal sendes så til en komponenten(`FreqAdd`) der sørger for at sammenligne signalerne `FreqCnt` og `SigCnt` for at enten få frekvensen længere op eller ned.

Hvor det så sendes ud gennem signalet `nSigCnt` hen til (`SigReg`), denne komponent får et Enabel-signal fra komponenten(`PWMcount`) så hele SigGenDataPath modulet er synkroniseret med PWM signalet fra bordet, men outputet fra komponenten(`SigReg`) sender ud til de 3 dekodere der hver især laver 3 forskælige signaler der alle bliver sendt til en multiplexer(SigMux) som bliver styret af Shape som bestemmer hvilken af de 3 signaler derefter skal gå videre til `AmplDec`.

I komponenten(`AmplDec`) bliver Ampl sat sammen med signalet fra multiplexeren der går videre ud til et 7 bit signal, `SigAmpl`, hvor den bliver sammenlignet i komponenten(`PWMcomp`) med signalet `PWMcnt`, `PWMcomp` output `PWM` går til et register der venter på at `SigEn` (run) går høj hvorden der efter sender PWM-signalet ud til filteret.

som beskrevet i Oscilloskop afsnittet vil der når Shape, Amp og freq indstilles i Labview sendes en type 1 datapakke via UART, til MCU'en. Pakkens datadel består af to bytes, den første byte indeholder hvilken knap der er trykket på og den anden indeholder switch-værdien der er blevet tastet.

- For "Enter" knappen vil den aktive LED (Har 0x00 som start værdi) i Labview afgøre hvilken parameter der arbejdes med. Såfremt dataen ligger indenfor de acceptable værdier vil talværdien blive sendt videre til FPGA boardet vha. SPI.

- For "Select" knappen vil den indtastede switch værdi (0x00 - 0x02), afgøre om det er henholdsvis Shape, Amp eller freq der skal være aktiv. Igen vil der sendes en værdi til FPGA vha. SPI, dog er

værdien ikke væsentlig da denne blot aktiverer den valgte tilstand på FPGA boardet.

- For "Run/Stop" knappen bliver der sendt en værdi til FPGA vha. SPI, igen her er værdien underordnet vi har blot valgt at det skal være switch værdien der bliver sendt.

- For "Reset" knappen bliver variablerne nulstillet og sidepanelet i Labview bliver opdateret. Derudover bliver de nye værdier (0x00) for Shape, Amp og Freq sendt enkeltvis efterfulgt af send for på den måde at opdatere hver enkelt på FPGA.

3 Implementering

3.1 Oscilloskop

I c-modulet er der udover main.c også følgende header filer:

- adc.h indeholder initialiseringen af analog til digital converteren. ADC'en er sat op til at køre 8 bit samples, auto trigger mode ved interrupt fra timer 1 og prescalet med 16 for at kunne køre maksimal hastighed ved 8 bit (1000 kHz)

- counter.h indeholder initialiseringen af 16 bit counter/timer 1 som styrer ADC sampleraten. Timer 1 er sat op til at køre CTC mode (Clear timer on compare), prescalet med 64, output compare B match interrupt enable samt OCR1A som TOP/sammenligningsværdi

- Labview.h som indeholder:

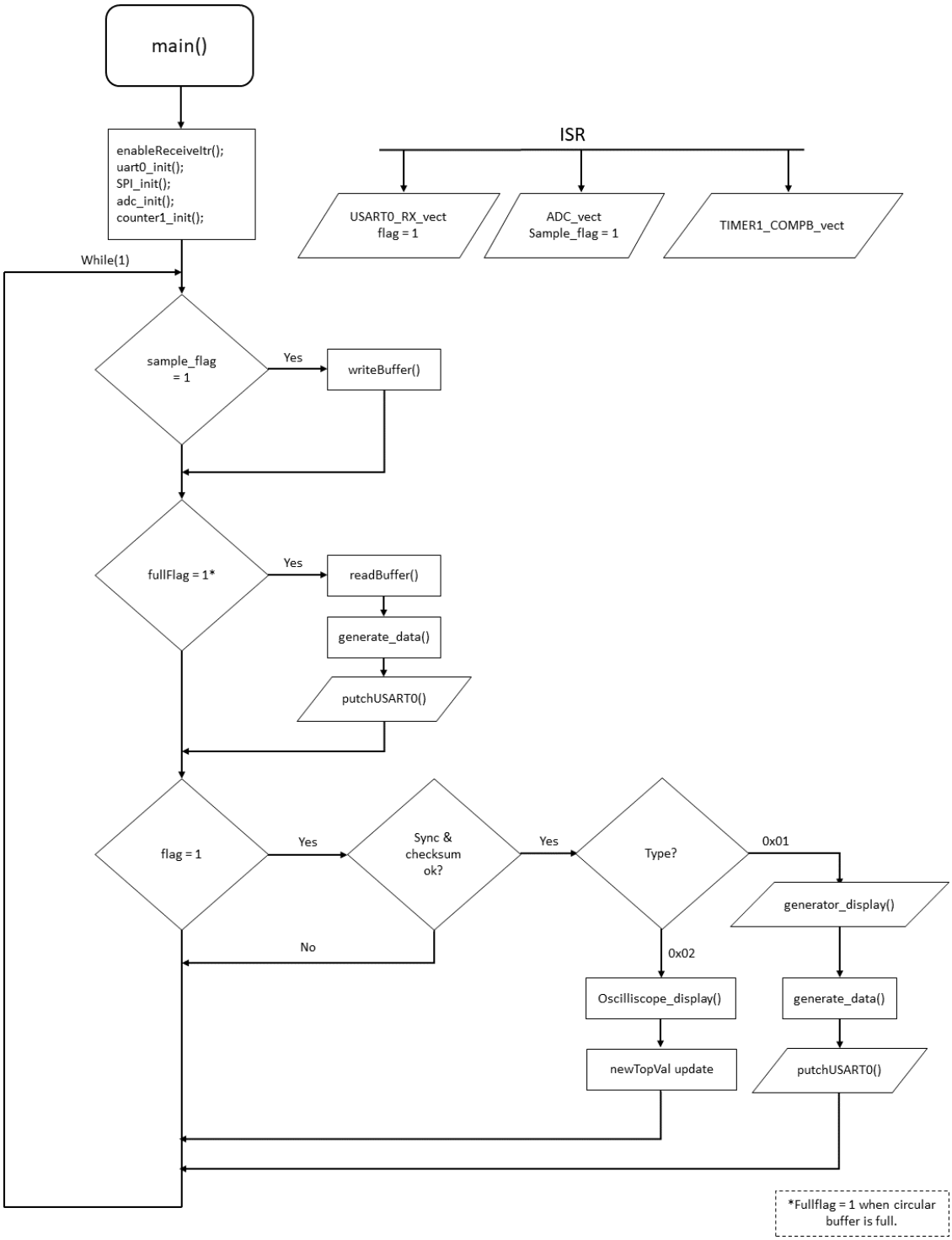
  1. generator_display() som afkoder modtagne datapakker, desuden opdaterer den værdier på Active-LED, Shape, Amp og Freq der skal bruges til at sende retur til Labview, for at opdatere sidepanelet i generator fanebladet, og til blive videresendt til FPGA boardet vha. SPI.
  2. oscilloscope_display() som sammensætter de to modtagne byteværdier Sample rate og Record.
  3. generate_data() sørger for at lave datapakker der skal sendes retur til enten generator- eller oscilliskop fanebladet bliver pakket.
  4. checksum() foretager LRC-8 checksum på modtagne datapakker og datapakker klar til afsendelse.

- uart.h der indeholder initialiserings funktionerne til UART i Design Oscillloskop afsnittet er det nærmere beskrevet hvordan det er sat op. Derudover er der funktionerne getchUSART0(), putchUSART0 (), enableReceiveItr(), disableReceiveItr(), der henholdsvis kan modtage en karakter, sende en karakter, aktivere "Receive complete interrupt" og deaktivere "Receive complete interrupt".

- SPI.h indeholder initialiseringen af SPI forbindelsen. Herunder at der kun anvendes signalerne MOSI, SCK og SS, samt sættes SPI baudraten på 500kbaud. Yderligere er der funktionen putcSPI_master(), der sørger for at sende en byte over SPI. Funktionerne sendShape(), sendAmpl(), sendFreq(), sendRun() og sendEnter() er kodet på den måde at de indeholder den passende byte-protokollen til signalgeneratoren.

Desuden er der også to funktioner i main.c, writeBuffer() og readBuffer(), disse styrer cirkulær buffer hvor ADC samples bliver læst ind i. Så længe writeIndex og readIndex ikke har samme værdi bliver der læst samples ind i denne, når readIndex eller writeIndex rammer højeste plads i arrayet bliver de nulstillet, så "brugte" samples bliver overskrevet. Når bufferen er fuld, readIndex = writeIndex, bliver fullFlag = 1.

3.1.1 Main løkken



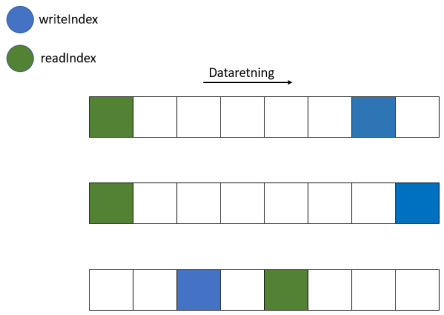
Figur 4: Flowchart main();

Ovenstående figur viser et flowchart af hvordan main løkken eksekveres.
Inden While(1) løkken starter bliver UART, Recieve Complete Interrupt, SPI, ADC, Counter1 og Counter 3 initialiseret.
Derefter starter while løkken, hvor der først bliver checket på om ADC har en sample klar til at blive læst, hvis der er det bliver denne værdi læst ind i databufferen. Dernæst kontrolleres der om fullFlag = 1, altså om databufferen er fyldt, såfremt det er sandt bliver en data med en længde bestemt af Record (Bestemt i Labview) læst fra databufferen og lagt i et nyt array. Herefter bliver data-delen af pakken, i generate_data() sat sammen med sync, length of package, type og checksum inden det sendes vha. putchUSART0() i en for-løkke.

Dernæst kontrolleres der om USART0_RX_vect modtaget en datapakke fra Labview, hvis det er aktuelt bliver flag = 1. Herefter bliver sync og checksum kontrolleret, såfremt disse ikke er korrekte returneres der til while løkken. Hvis de er korrekte kontrolleres datapakkens type, for type 0x01 (generator fanebladet) henvises der til generator_display() og type 0x02 henvises der til oscilloscope_display(), hvad der nøjagtigt sker i disse to funktioner funktion bliver beskrevet yderligere i underafsnittet "Labview kommunikation". Dog skal det nævnes at efter oscilloscope_display() bliver newTopVal opdateret, hvilket justerer ADC sample rate.

3.1.2 Databufferen

Som tidligere beskrevet fungerer databufferen ved at ADC samples bliver læst ind i bufferen indtil denne er fyldt. Hvis der i mellemtiden er blevet læst fra bufferen frigives de pladser der indeholder allerede brugte samples. Hvis writeIndex = readIndex, indikerer det at bufferen er fuld indtil der igen bliver læst data herfra - i dette tilfælde vil Fullflag = 1. Der er også en funktion der indikerer hvis bufferen er tom, denne er ikke blevet brugt da implementeringen af databufferen ikke lykkedes til fulde. Nedenstående figur illustrer hvordan bufferen fungerer.



Figur 5: Databuffer

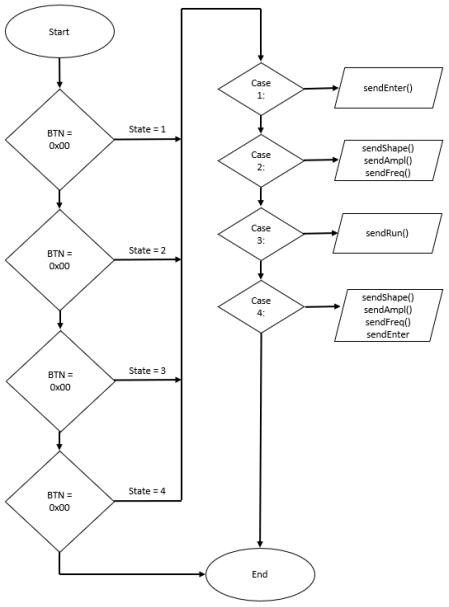
Ovenstående viser hvordan bufferen på øverste illustration er begyndt at læse data ind, på den midterste vil bufferen være fuld da der endnu ikke er læst data ud og på sidste illustration ses det at der nu er frigivet plads til at der endnu engang kan læses data ind i bufferen.

3.1.3 Analog sampling

Analog sampling udføres ved at timer 1 tæller fra 0 til TOP værdien i OCR1A. Når den rammer TOP værdien udløser det et interrupt og timeren starter fra 0 igen. Interruptet starter den analoge sampling og værdien af samplingen gemmes i ADCH. For globals og functions, se A.2.2 og A.2.3 i Appendix.

3.1.4 Labview kommunikation

Som beskrevet i underafsnittet "Main løkken" bliver flag = 1 når der er modtaget en datapakke fra Labview, dette sker vha. et "Recieve complete interrupt". Når sync og checksum er blevet kontrolleret bliver type analyseret, hvor type 0x01 er generator_display() og type 0x02 er oscilloscope_display().



Figur 6: Flowchart generator_display();

Overstående figur viser et flowchart over generator_display(). Her afgør fire if statements hvilken tilstand den efterfølgende switch case skal gå til, dette gøres ved at datapakkens første data byte indeholder en værdi på 0x00-0x03, henholdsvis "Enter", "Select", "Run/Stop" og "Reset".

Case 1: behandler "Enter", her kontrolleres der hvilken tilstand den aktive LED, i Labview sidepanelet har. Dernæst bliver det kontrolleret om den indtastede switch værdi er indenfor de rammer som den enkelte tilstand kan indeholde, såfremt dette er korrekt vil værdien blive læst ind i enten "Shape", "Amp" eller "Freq". Herefter vil den pågældende værdi blive gemt på MCU'en og videresendt til FPGA boardet vha. sendEnter() SPI funktionen. I tilfælde af at værdien overstiger rammerne for den aktive tilstand, vil værdien ikke blive opdateret, men forblive det tidligere værdi.

Case 2: behandler "Select", opdaterer hvilken tilstand der skal være aktiv for at indlæse data. Ved opstart og vil reset bliver tilstanden sat til 0x00, hvilket henviser til den øverste tilstand i Labview "Shape". Man vælger tilstand ved at taste en switch værdi på 0x00-0x02, henholdsvis "Shape", "Amp" eller "Freq". Denne switch værdi bruges som til endnu en switch case, der afgør tilstanden. Derefter bliver der sendt information om tilstand til FPGA boardet vha. enten sendShape(), sendAmpl() eller sendFreq() og tilstanden bliver gemt på MCU.

Case 3: behandler "Run/Stop", knappen fungerer ved at man trykker en gang for at starte signalgeneratoren og næste gang skal den stoppe. Det er kodet således at RS_flag = 0 fra start, ved første tryk i Labview testes der på to if-statements, hvor den ene tester på RS_flag = 0 og den anden RS_flag = 1. Såfremt RS_flag = 0 bliver sendRun() aktiveret og signalgeneratoren starter derefter bliver RS_flag = 1, for at det andet if-statement er aktivt næste ved næste cyklus. Er RS_flag = 1 bliver der testet på den sidst kendte værdi for den aktive led, signalgeneratoren stoppes og der returneres til den tilstand FPGA boardet havde inden start.- Alle tilstande undtaget "Reset" kontrollerer på RS_flags tilstand, således er det altså ikke muligt at

ændrer på værdierne eller den aktive tilstand så længe signalgeneratoren kører.

Case 4: behandler "Reset", her bliver værdierne active, amp, shape og freq nulstillet og disse værdier bliver sendt til de respektive tilstande på FPGA boardet.
Generate_data() bliver kaldt når der er data klar til at blive sendt tilbage til Labview. Den forbereder en buffer, tx_buffer med syncbytes, length of package, type, data og checksum. Igen er der forskel på hvordan pakkerne der sendes tilbage til Labview ser ud. Nedenstående figur illustrer hvorledes de to forskellige pakker kan se ud.

Type 1: Til generator fanebladet

Sync		Length		Type	Data				CRC	CRC
0x55	0xAA	1 byte	1 byte	1 byte	Aktive indikator	Shape	Amp	Freq	0x00	LRC8

Type 2: Til oscilliskop fanebladet

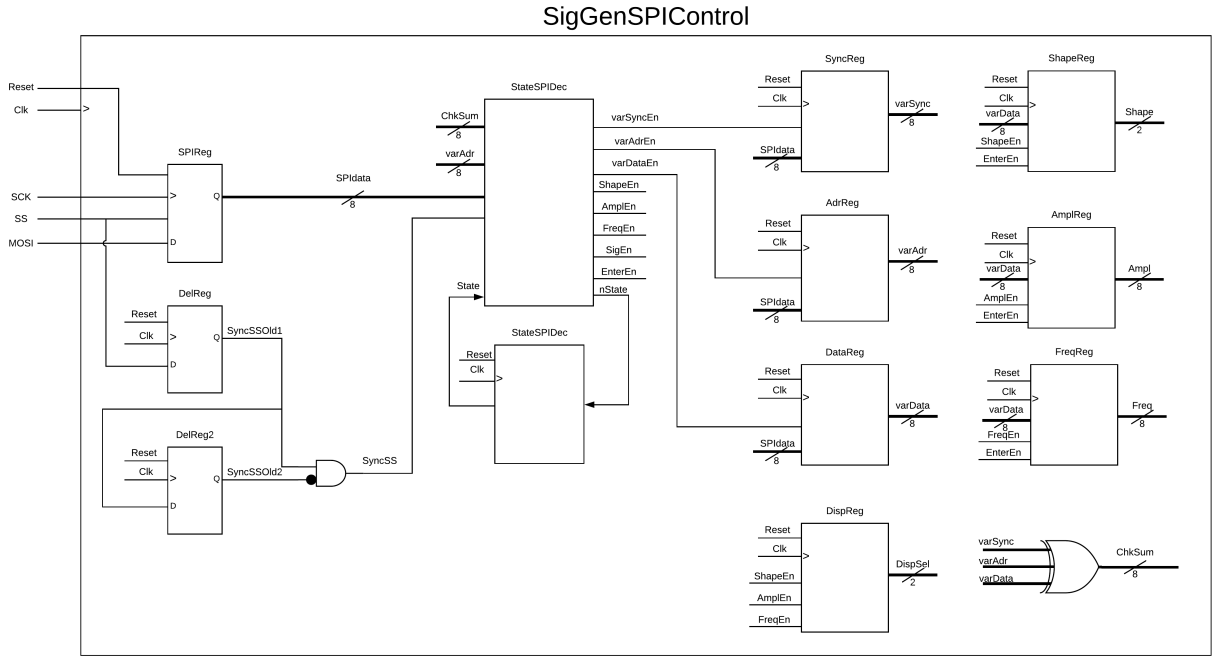
Sync		Length		Type	Data	CRC	CRC
0x55	0xAA	1 byte	1 byte	1 byte	x bytes	0x00	LRC8

Figur 7: Data typer

Her kan det ses at type 0x01 datapakker indeholder de data der bliver behandlet i generator_display() og disse bliver sendt retur til Labview hvor de opdaterer sidepanelet i generator fanebladet.
Type 0x02 datapakker indeholder data samples fra ADC'ens målinger på signalgeneratoren.

3.2 Signalgenerator

Der blev udleveret kode til en signalgenerator, hvor der her er blevet modificeret på *SigGenControl* således at dette blevet styret vha. SPI-kommunikation . Figur 8 viser funktionsdiagrammet over det modificeret modul.

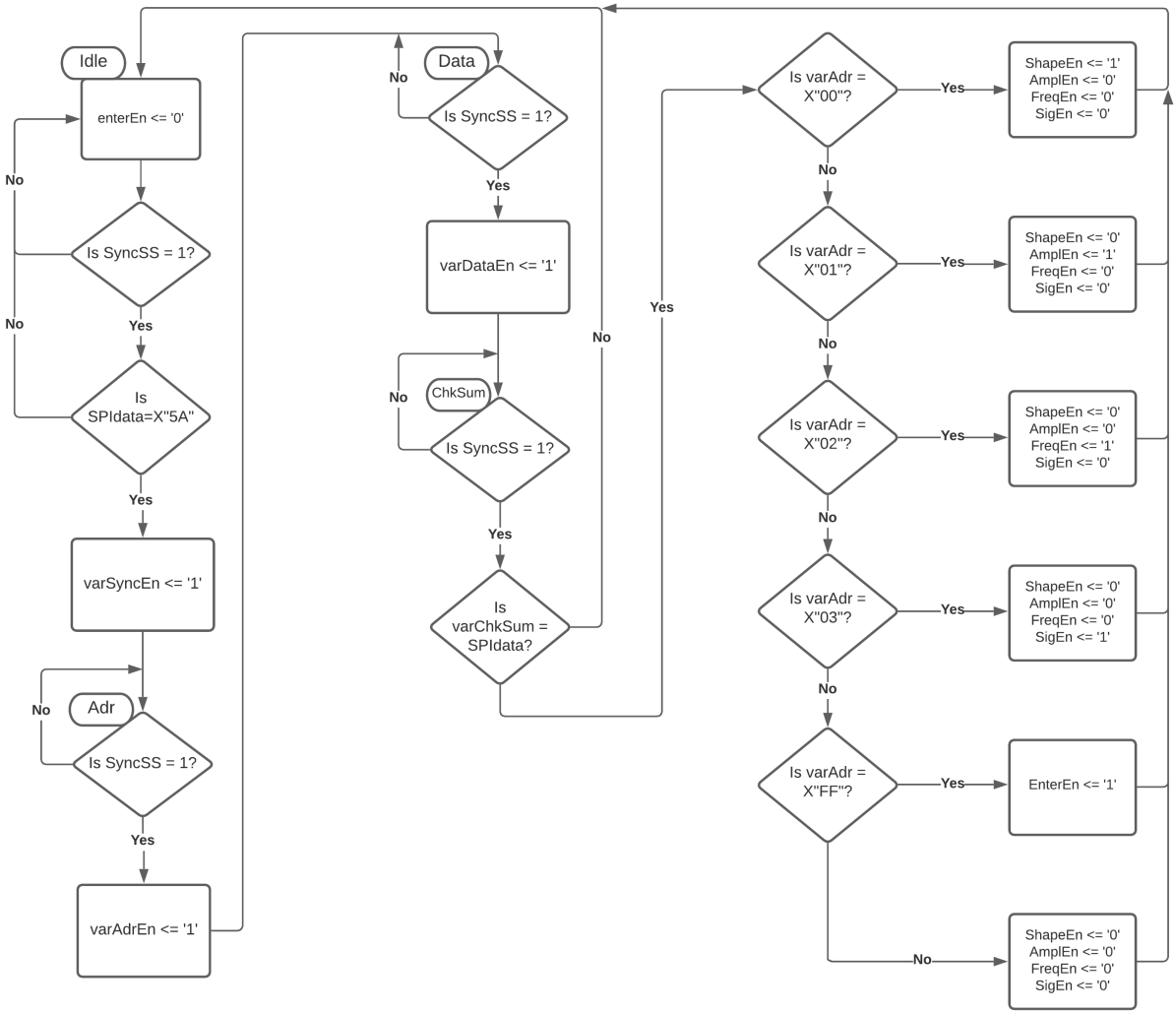


Figur 8: Funktionsdiagram over SigGenSPIControl

Det fungerer ved, at et shift-register skifter `MOSI` signalet ind på `SPIdata`. Dette gør den når `SS` er lavt og der er en opadgående flanke på `SCK`. Når byten er overført, vil SS gå høj og `SPIdata` vil blive brugt i `StateSPIDec`, `SyncReg`, `AdrReg` og `DataReg`. For at skifte tilstand i tilstandsmaskinen, bruges et synkroniseret `SS` puls. Dette sker ved brug af to D-flip-flops, som ses på figur 8 ved de to registre `DelReg` og `DelReg2`. `DelReg's` output går både til `DelReg2` og til AND-gaten, så når `SyncSSOld1` er lav kommer det ind på indputet af `DelReg2` og derefter ud af outputet en clock periode efter, hvor det så bliver inverterede, som gør at hvis `SyncSSOld1` skifter til at være høj i den næste clock periode så vil AND-gatens output blive høj. Det er vigtigt, for at få en sikker SPI forbindelse, at de forskellige clock/puls signaler er synkroniseret, da der ellers vil være mulighed for at `SS` rammer en clock som er lav.

Når der er valgt hvilken tilstand (shape, amplitude, frekvens) i tilstandsmaskinen, vil det gældende *enable* signal tænde for tilsvarende display i `DispReg` som sender `DispSel` videre til 7-segments modulet. Dette gør det muligt at på selve FPGA boardet, at se hvilken 'tilstand' programmet er i.

Yderligere så er der et asykront reset, som går til alle registere. Dette er BTN3 knappen på selve FPGA boardet. Det er også muligt at nulstille via LabView ved at sende en adresse-byte som ikke listet i tilstandsdiagrammet



Figur 9: Tilstandsdiagram over SigGenSPIControl

Der er fire tilstande: *Idle, Adr, Data* og *Checksum*. Dette er grundet den valgte byte protokol, at først sendes et synkroniseringsbyte, som er valgt til at være `X"5A"`, dernæst en adresse byte, en data byte og til sidst en checksumbyte, der er udregnet som ligning (1).

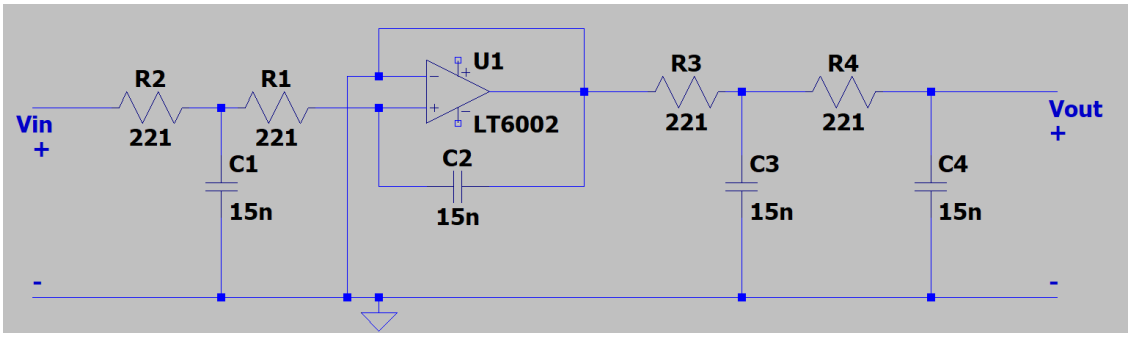
$$\text{CkkSum} = \text{varSync} \quad \text{XOR} \quad \text{varAdr} \quad \text{XOR} \quad \text{varData} \tag{1}$$

Derfor under `Idle` vil der, efter et højt synkroniseret `SS` puls, blive tjekket om den modtaget data stemmer overens med den ønskede byte. Igen, hvis dette er sandt, vil der åbnes for `SyncReg`, hvor at `SPIdata` vil indsættes på `varSync`. Efter burde de pakker der skal sendes være synkroniseret med de tilstande programmet er i. Der ses derfor kun efter et højt `SyncSS` signal, hvorved under `Adr`, vil der åbnes for `AdrReg`, og tilsvarende under `Data` åbnes `DataReg`. Sidste tilstand ses der om den udregnede checksum på MCU'en stemmer overens med FPGA'ens checksum, hvor at hvis det er sandt vil den gå vidre til at kigge på værdigen der er blevet indtastet på `varAdr`. Herefter vil der tændes for enten `Shape`, `Ampl`, `Freq` eller `SigEn` (som tænder for PWM signalet). Hvis `varAdr` er sat til `X"FF"` er der trykket på enter knappen i LabView, og `varData` vil blive sat ind på den tændte tilstand. Dette vil også sige at hvis `varAdr` er alt andet end `X"FF"`, vil dataen i `varData` være ligegyldig, da den ikke vil blive indsat i nogle register.

Derudover, bliver de diverse 'enable'-signaler kun sat under `ChkSum` tilstanden. Dette gøres for at holde den ønskede 'tilstand' tændt, selvom at tilstandsmaskinen er tilbage i `Idle`.

3.2.1 PWM filter

Som PWM filter blev der lavet ét 2. ordens aktivt lavpas filter, efterfulgt af to 1. ordens passive lavpas filtre, som er vist på figur 10.



Figur 10: PWM filter

Der blev valgt at bruge et aktivt filter, da der skal kunne arbejdes i meget lave frekvenser, og de højeste frekvenser vil være omkring 40kHz. For at få et stejlere fald per dekade, blev der tilføjet endnu 2 lavpas filtre, som er passive, men med samme knækfrekvens. Da alle modstande og kondensatorer har samme størrelse kan knækfrekvensen udregnes ved:

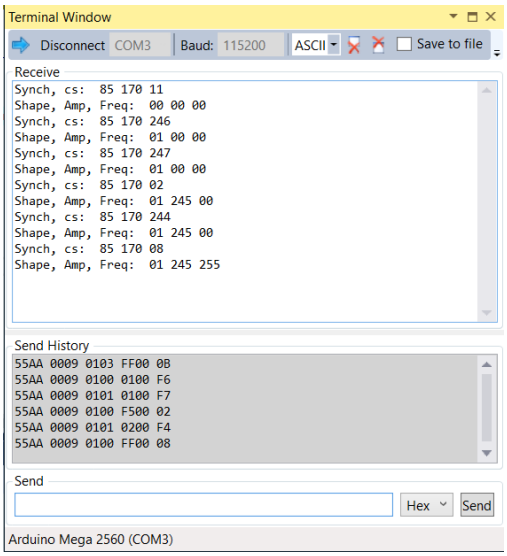
$$f_p = \frac{1}{2\,\pi\,R\,C} \tag{2}$$

Hvor at, hvis de brugte værdier indsættes i ligningen vil $f_p = 48\text{kHz}$, hvilket ligger lidt højere end signalgeneratorens højeste frekvens, men stadig under støjen fra PWM signalet. Og da det er et 4. ordens filter vil der være et fald på -80 dB/dekade og bør dermed skærer meget af PWM støjen fra.

4 Test

4.1 Oscilloskop

For at teste oscilliskopet skulle det først fastlægges hvordan MCU'en skulle håndtere datapakker fra Labview. Her blev simulatorene brugt for at bygge koden på MCU'en op på en måde der kunne håndtere at hente en hel datapakke. Måden modtagelse af data pakker blev testet på var at tage datapakker fra Labviews oscilliskop simulator og sende disse gennem terminalen i Atmel Studio, her kunne der så testet på om sync, length of package, type data og checksum blev læst ind i de rigtige variable.



Figur 11: UART test

Figuren viser at datapakker "sendt" fra Labview simulatoren bliver læst korrekt. Først bliver der sent en reset pakke, bemærk switch værdien på 255 (0xFF) bliver ignoreret og alle værdier bliver nulstillet. Herefter bliver switch værdien sat til 0x01 og et tryk på "Enter", da der har været trykket på reset er Shape aktiv og værdien bliver læst ind
Dernæst bliver en switch værdi på 0x01 samt "Select" sendt, det resulterer i at aplitude bliver aktiv og ved næste sendte pakke ses det at amplituden har fået ny værdi.
Sidst bliver 0x02 valgt med "Select" hvilket gør frekvensen aktiv og næste datapakke indeholdende switch værdi på 0xFF samt "Enter" bliver sendt til frekvent. For hver pakke ses det at sync og cs (Checksum) stemmer overens.

Databufferen har desværre ikke fungeret som ønsket, det har fungeret kontinuerligt at læse ind og ud af bufferen, dog er det ikke lykkedes at gøre forsendelsen af ADC samples helt kontinuert. Da det ikke lykkedes, blev det besluttet at bibeholde den afleverede opsætning af databuffer samt transmit af samples til Labview.

Det fungerer sådan nu, at når data bufferen er fyldt (fullFlag=1) bliver der sendt en pakke med data til Labview, herefter ventes der på at bufferen igen bliver fyldt, denne gang mangler der kun at fyldes det antal af pladser som tidligere er afsendt.

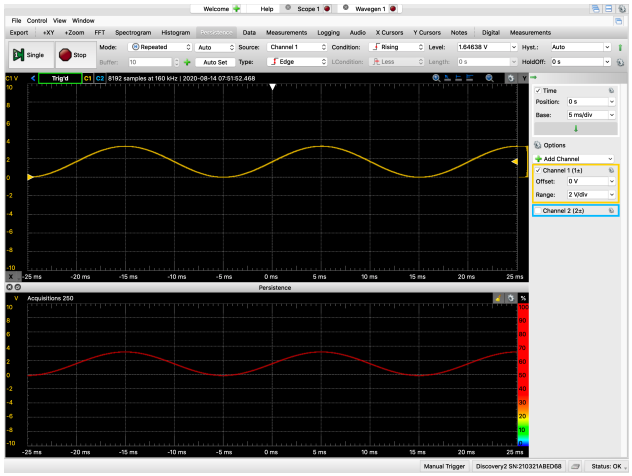
Det giver en skævvridning af data i oscilliskopet, dog kan man ved at sætte frekvens, samplerate og record length korrekt opnå ganske små afvigelser i kontinuiteten af de afsendte data. Fejlen giver tydligst udslag ved sinuskurven under stigning eller fald i kurven. Når der testes med FPGA signal generatoren gør PWM støjen på oscilliskop billedet at det bliver meget svært at pinpointe afvigelsen i billedet.

4.1.1 Dataintegritetstest

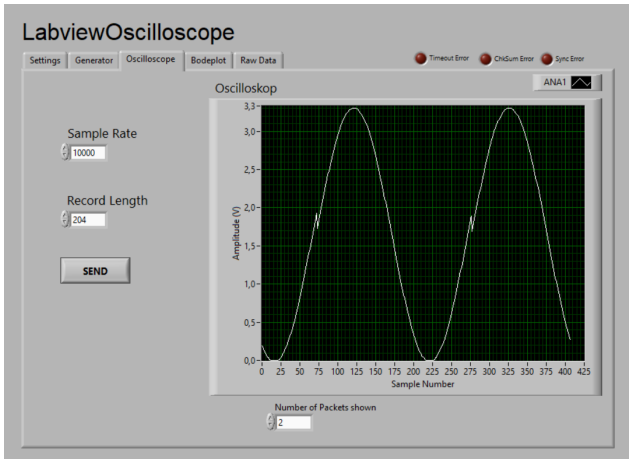
Dataintigritettesten er blevet udført ved at indstille en tonegenerator til en given shape, amplitude og frekvens, imens Labview blev indstillet til forskellige samplerates og record lengths.
Testene blev udført med følgende værdier:

Test nr:	**Samplerate:**	**Record length:**	**Signal:**	**Amplitude:**	**Frekvens:**
1	10000 sps	204	sinus	3,3 V	50 Hz
2	5000 sps	56	firkant	1 V	500 Hz
3	1000 sps	170	sinus	3,3 V	10k Hz
4	1001 sps	170	firkant	3,3 V	5k Hz
5	10000sps	100	sinus	3,3 V	500 Hz

Test 1:

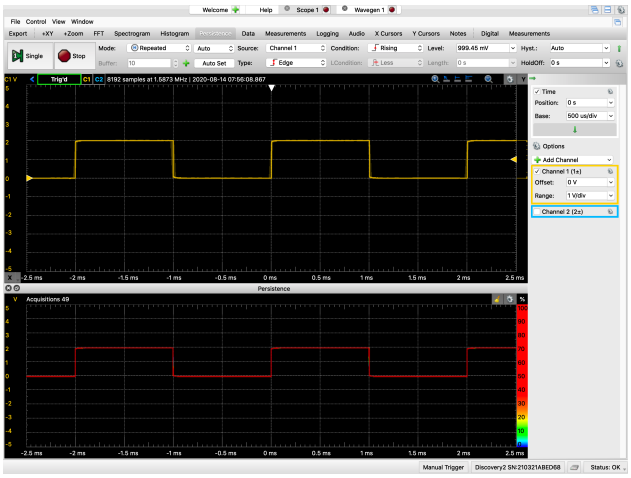


Figur 12: Test 1: Oscilliskop sinus signal

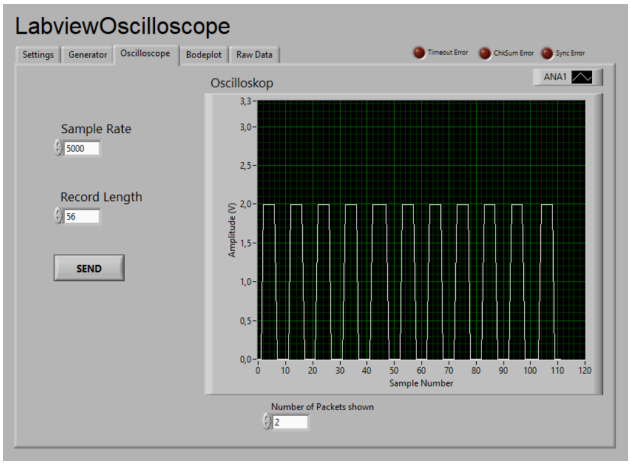


Figur 13: Test 1: Labview sinus signal

Test 2:

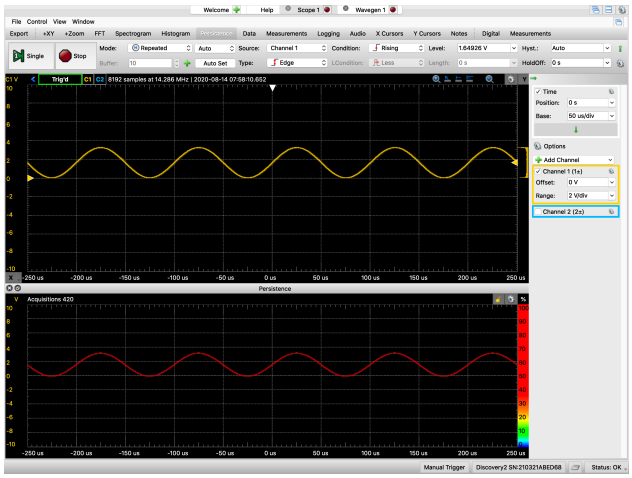


Figur 14: Test 2: Oscilliskop firkant signal

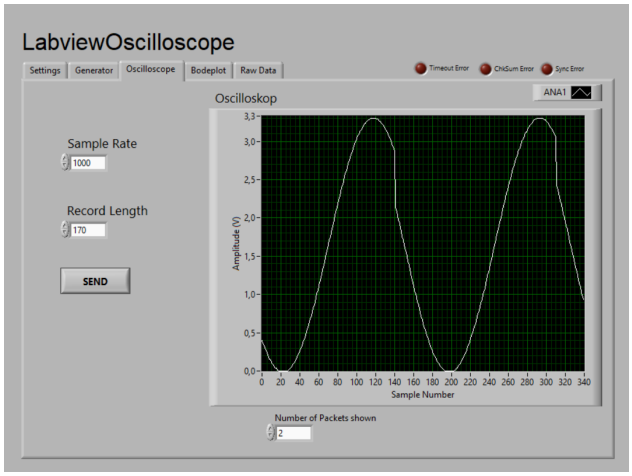


Figur 15: Test 2: Labview firkant signal

Test 3:

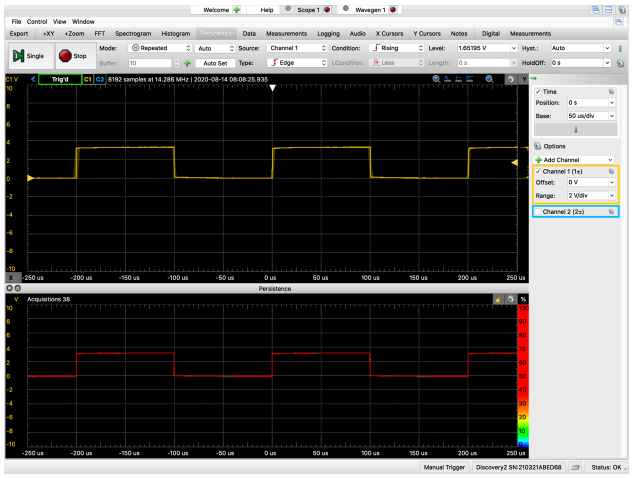


Figur 16: Test 3: Oscilliskop sinus signal

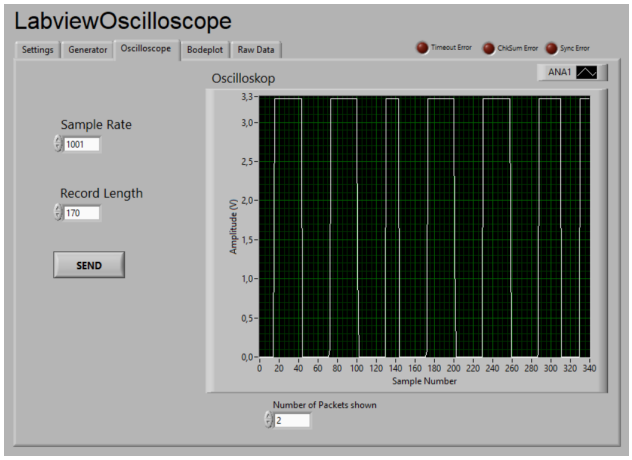


Figur 17: Test 3: Labview sinus signal

Test 4:

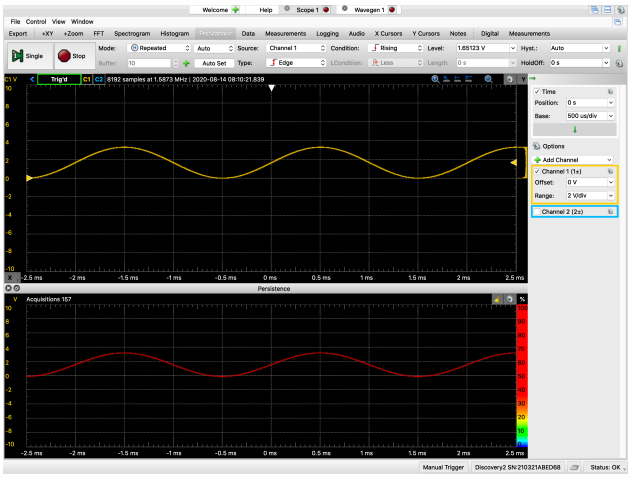


Figur 18: Test 4: Oscilliskop firkant signal

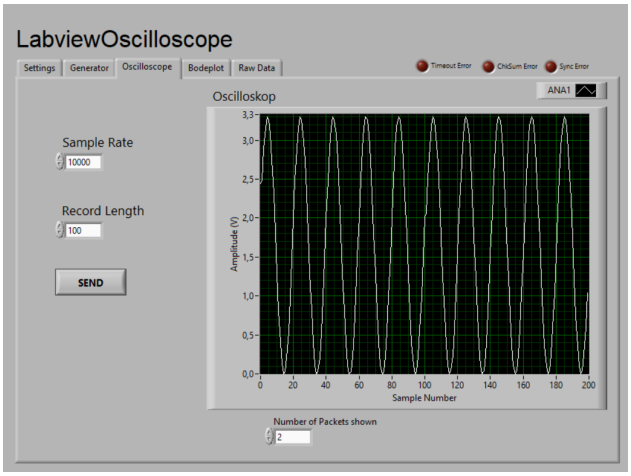


Figur 19: Test 4: Labview firkant signal

Test 5:



Figur 20: Test 5: Oscilliskop sinus signal



Figur 21: Test 5: Labview sinus signal

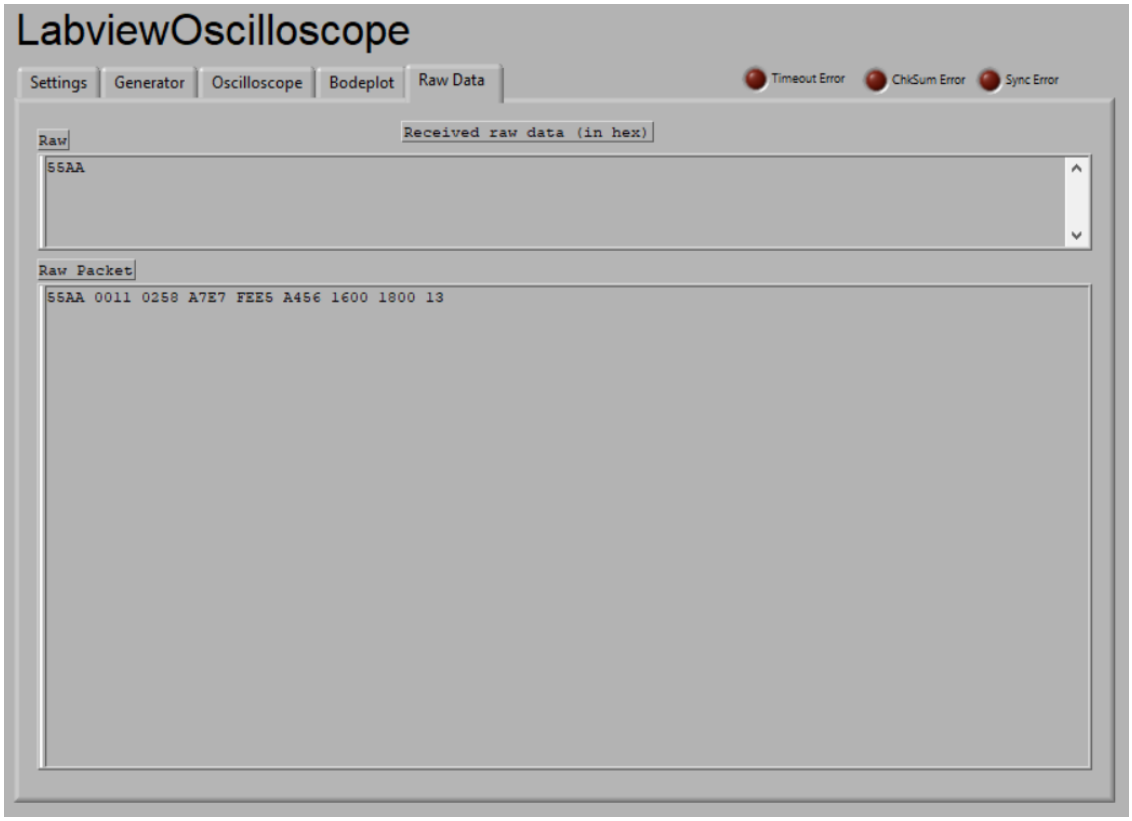
Som ovenstående tests viser, har vi et problem med kontinuitet af datapakkerne sendt til Labview. Det er igen mest tydeligt på sinus kurverne, men det blev valgt både at lave test med firkant og sinus signaler. Især i test 3 viser forskubbelsen af den faldende flanke på sinuskurven at datapakkerne halter bagefter, imens ADC målingerne fortsætter - det skal dog også bemærkes at med en frekvens på 10k Hz kræves der rigtig mange samples for at få en helt skarp kurve.

Som test 5 viser er det dog muligt at få en rigtig flot sinus kurve med god kontinuitet, selvom frekvensen ikke er så høj.

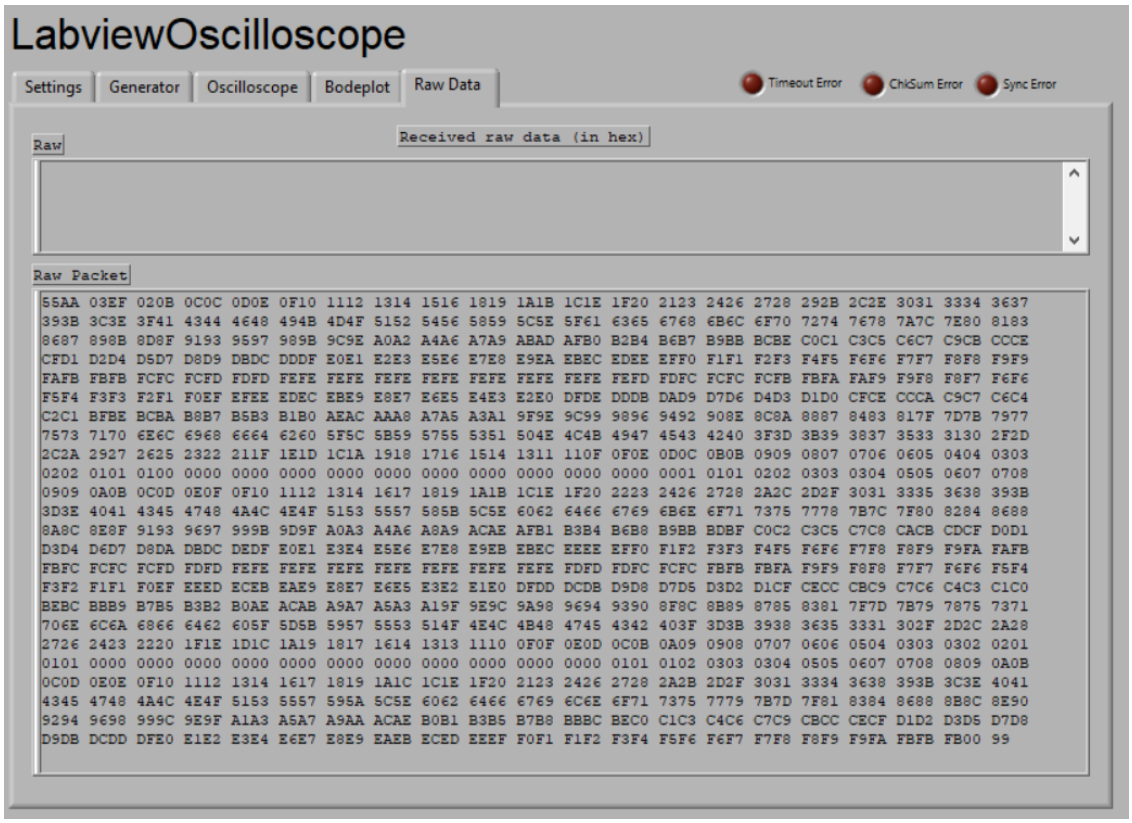
4.1.2 Parametertest

Parametertesten for samplerate er blevet udført ved at indstille samplerate i LabView og undersøge udgangssignalet som beskrevet under 4.1.3 10 sps, 5 ksps og 10 ksps blev indtastet i LabView og resultatet fra Digilent oscilloskopet kan ses på billederne under 4.1.4

Parametertesten for record length er blevet udført ved at indstille LabView til henholdsvis en længde på 10 og 1000 samples og se på LabView raw data output om det forventede blev returneret fra MCU'en



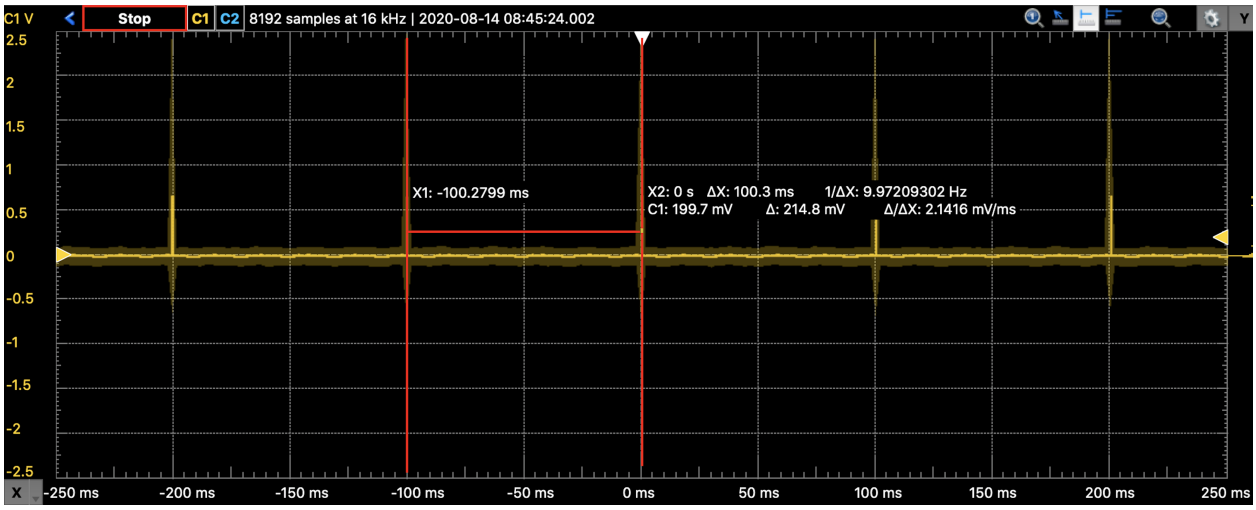
Figur 22: Parametertest record length 10



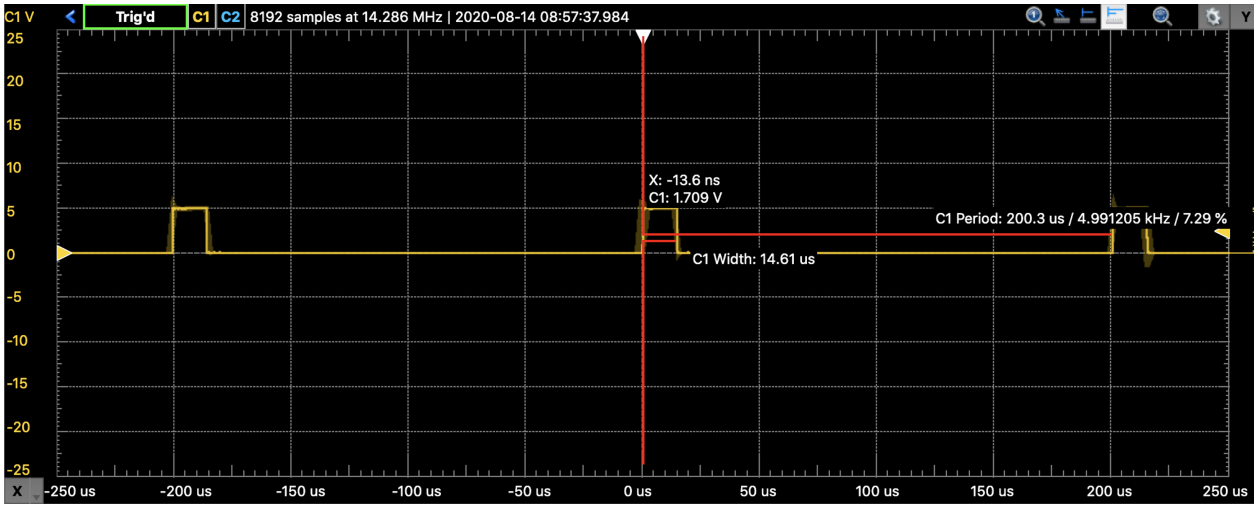
Figur 23: Parametertest record length 1000

4.1.3 ADC samplerate min/max modul test (standalone test)

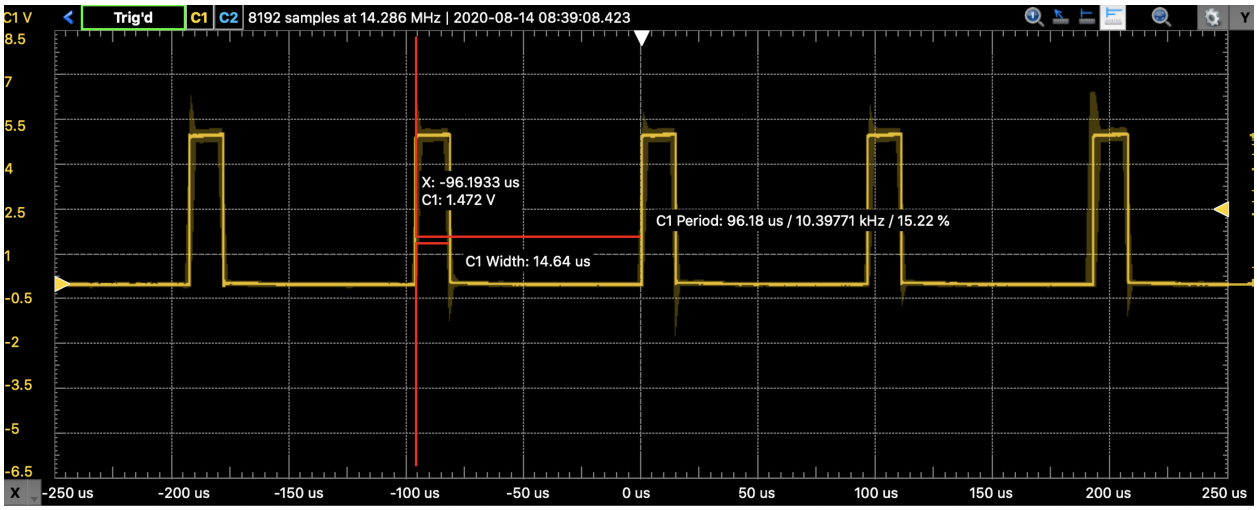
ADC'ens samplerate er blevet testet på to måder, både ved hardware (oscilloskop måling) og software (via terminal). Hardware testen foregik ved at måle med oscilloskopet på pin PB7. PB7 blev så sat op til at gå høj i interrupt service routine "TIMER1_COMPB_vect" dvs. når timer 1 har talt til TOP værdien, givet interrupt og derved har startet en sampling. Derefter blev PB7 sat lav igen i ISR(ADC_vect) altså når samplingen var udført. Dermed kunne samplingfrekvensen måles med oscilloskopet. Der blev målt ved de tre frekvenser fra kravspecifikationen 10 sps, 5 ksps samt 10 ksps. Målingerne er vist herunder som billeder fra oscilloskopet i Waveforms.



Figur 24: ADC samplerate modul test 10 sps

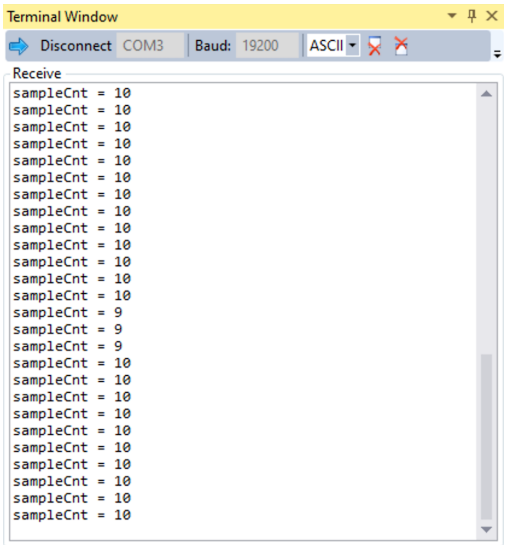


Figur 25: ADC samplerate modul test 5 ksps

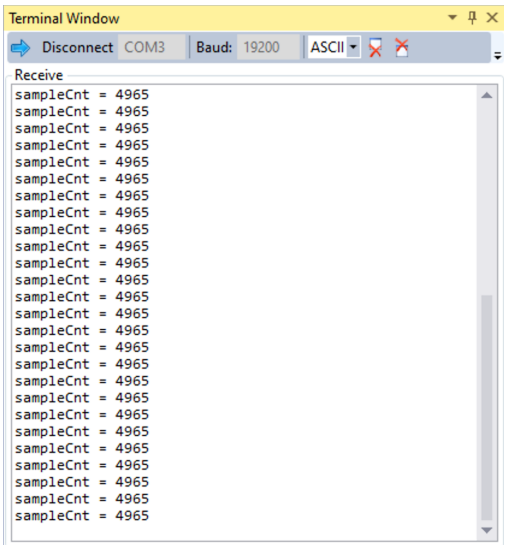


Figur 26: ADC samplerate modul test 10 ksps

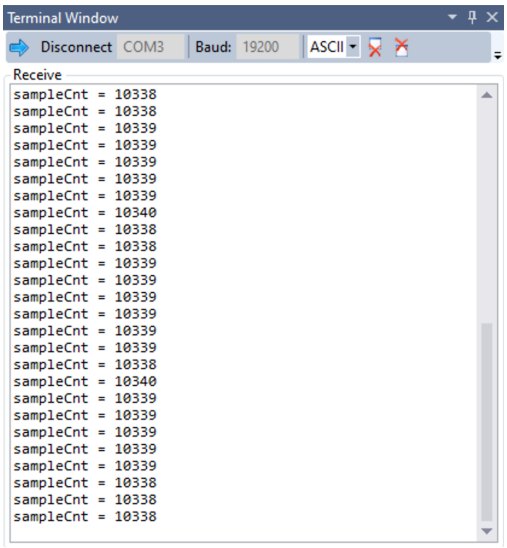
Software testen foregik ved at programmere timer 3 til at give interrupt 1000 gange i sekundet. I ISR(TIMER3_COMPB_vect) var der en variabel der talte en op hver gang der kom et interrupt. Samtidig blev der testet om variablen havde talt til 1000 og når det skete satte den et flag. I ISR(ADC_vect) var der en anden variabel der talte op hver gang der var en sample klar (ADC interrupt) og når timer 3 flaget var sat blev værdien af ADC interrupt variablen udskrevet til terminalen via UART. Derved blev antal samples pr. sekund talt og udskrevet på terminalen, se nedenstående billeder af terminalen.



Figur 27: ADC samplerate modul test 10 sps



Figur 28: ADC samplerate modul test 5 ksps



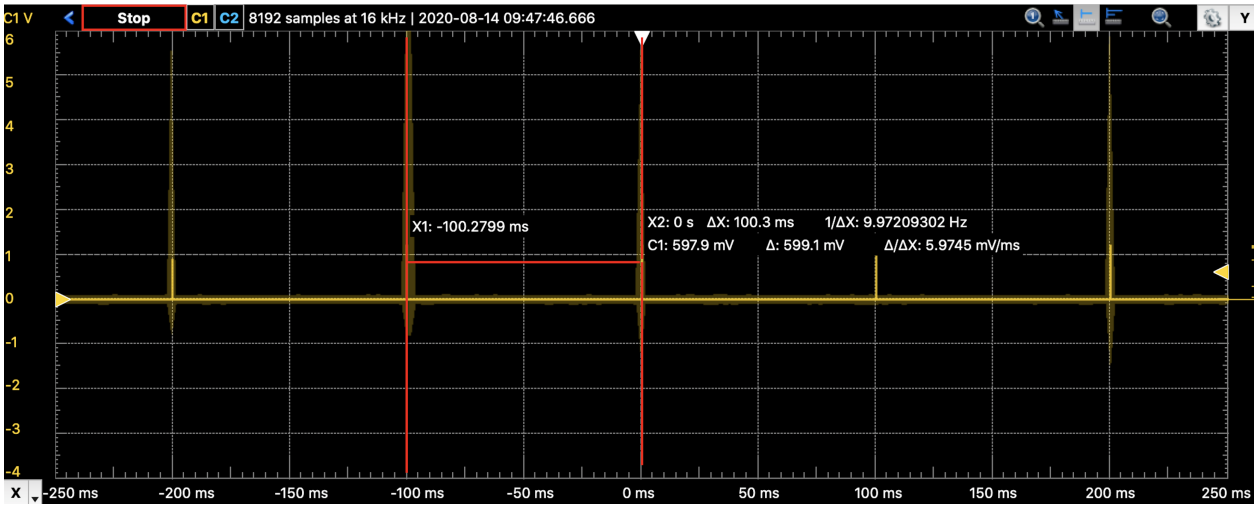
Figur 29: ADC samplerate modul test 10 ksps

I nedenstående tabel er krav og målinger listet for bedre overskuelighed.

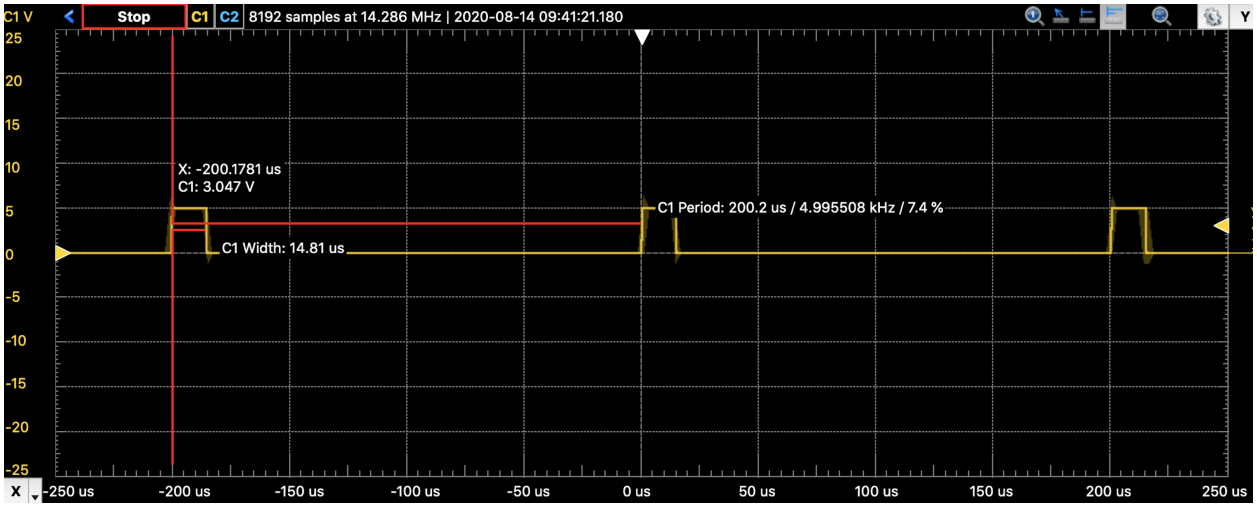
Krav:	**Målt (OSC):**	**Målt (TERM):**
10 sps	9,972 sps	9 - 10 sps
5 ksps	4,991 ksps	4,965 ksps
10 ksps	10,398 ksps	10,339 ksps

4.1.4 ADC samplerate min/max system test

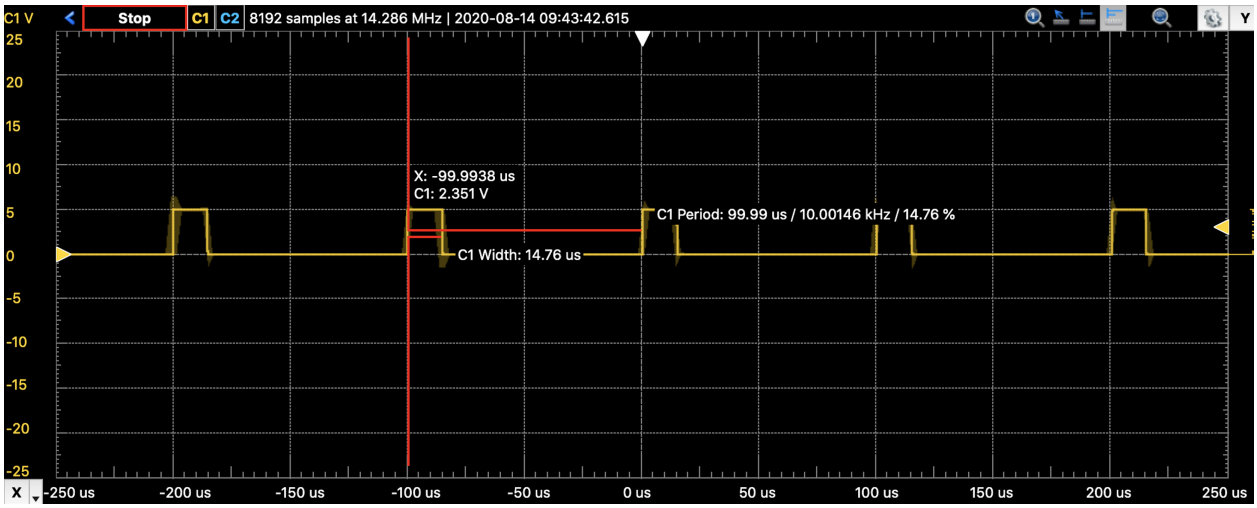
I system testen blev der ligesom i modul testen testet ved henholdsvis at sætte PB7 høj i ISR(TIMER1_COMPB_vect) og lav i ISR(ADC_vect) således at sampleraten kunne måles på PB7 med et oscilloskop. Se 4.1.3 for grundigere gennemgang af testopstillingen. Herunder ses billeder af testen fra oscilloskopet.



Figur 30: ADC samplerate Digilent oscilloskop system test 10 sps



Figur 31: ADC samplerate Digilent oscilloskop system test 5 ksps



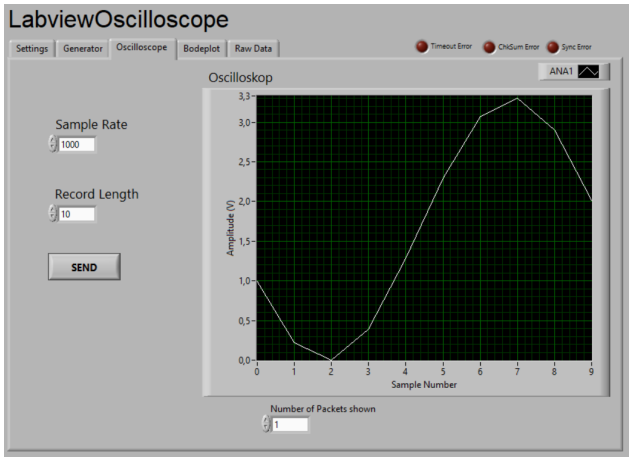
Figur 32: ADC samplerate Digilent oscilloskop system test 10 ksps

Sammenfattet i nedenstående tabel ses krav holdt op imod de målte frekvenser.

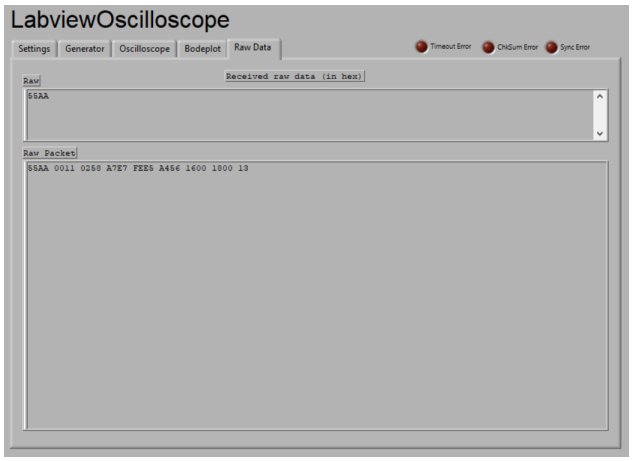
Krav:	**Målt (OSC):**
10 sps	9,972 sps
5 ksps	4,996 ksps
10 ksps	10,001 ksps

4.1.5 Record length test

Der er blevet testet om Oscilloskopet kan køre med ned til 10 ADC målinger i sekundet og op til 1000 ADC målinger for flere sample rates:
Test med 10 ADC målinger:



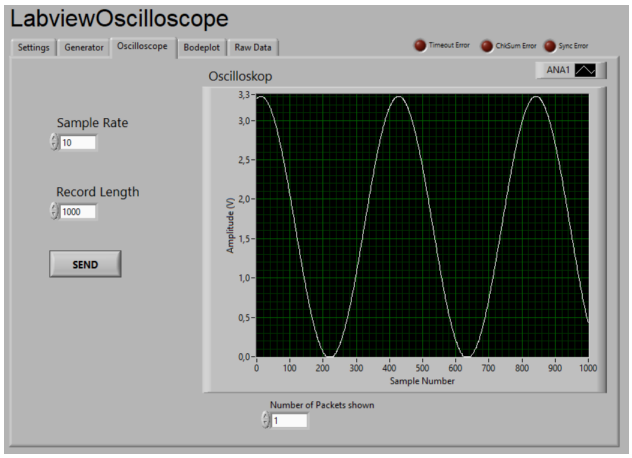
Figur 33: Oscilliskop 10 ADC målinger



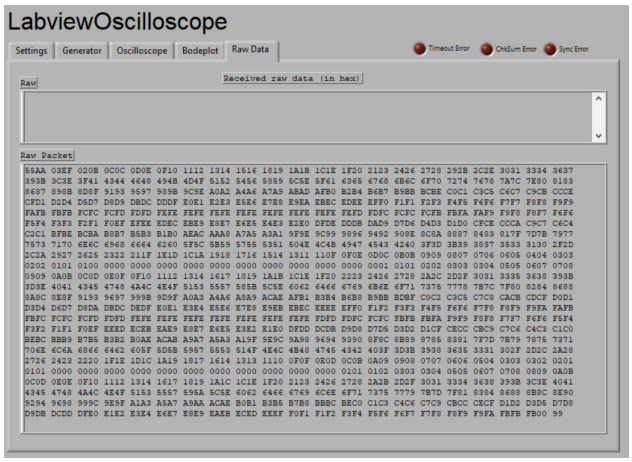
Figur 34: Raw 10 ADC målinger

Her ses det at det var muligt at komme ned på 10 sps med oscilliskopet.

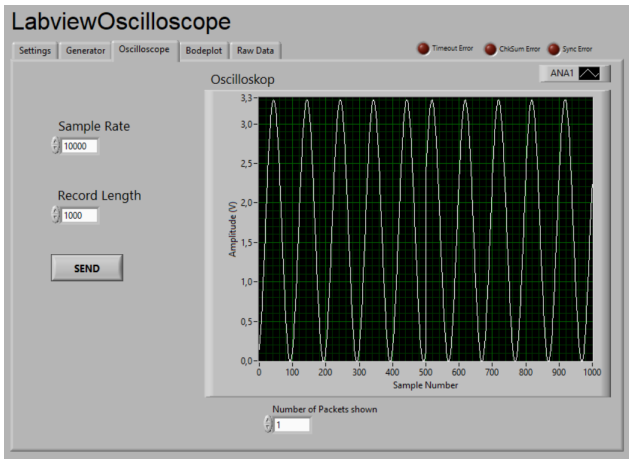
Test med 1000 ADC målinger:



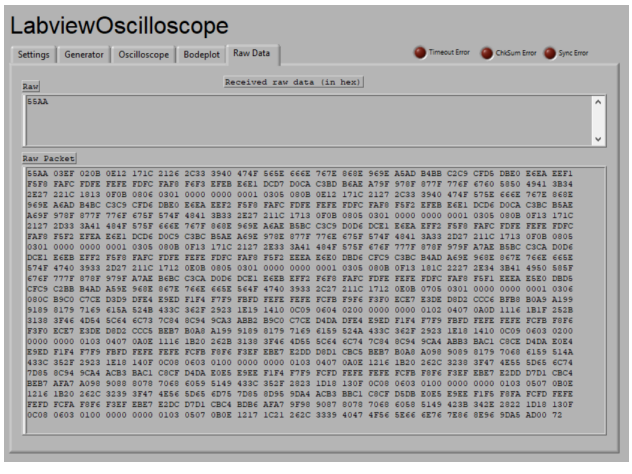
Figur 35: Oscilliskop 1000 ADC målinger, samplerate = 10



Figur 36: Raw 1000 ADC målinger, samplerate = 10



Figur 37: Oscilliskop 1000 ADC målinger, samplerate = 10000



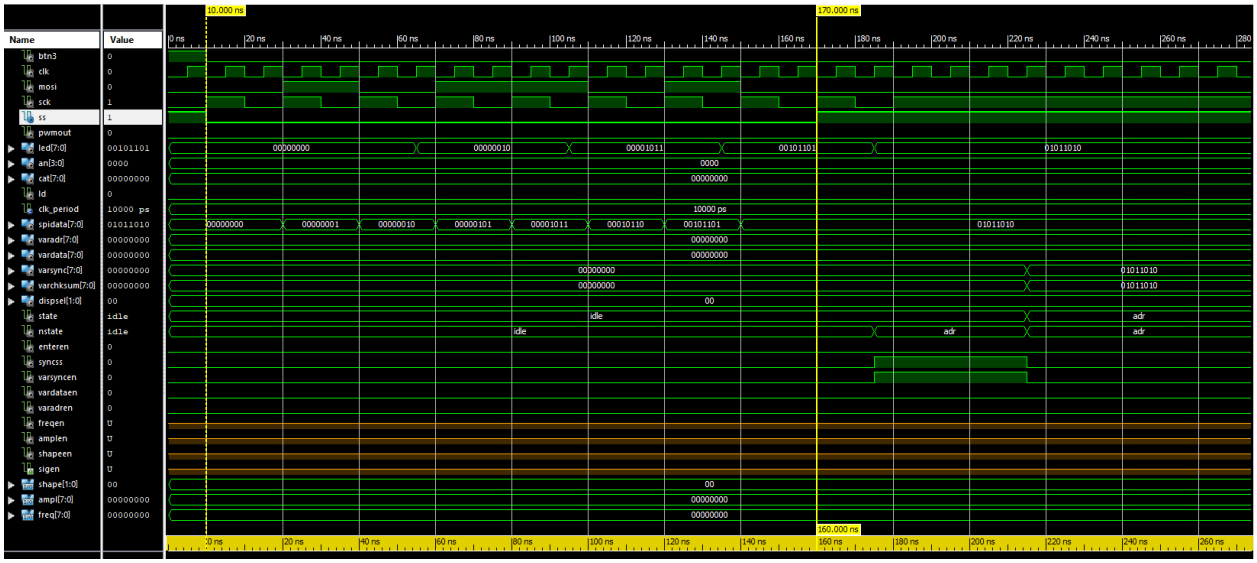
Figur 38: Raw 1000 ADC målinger, samplerate = 10000

Her ses det at det er muligt at køre med 1000 ADC målinger med højeste og laveste record length. Bemærk længden på pakken indikerer af den indeholder 1007 bytes, hvor de 7 er sync, record length, type og crc.

4.2 Signalgenerator

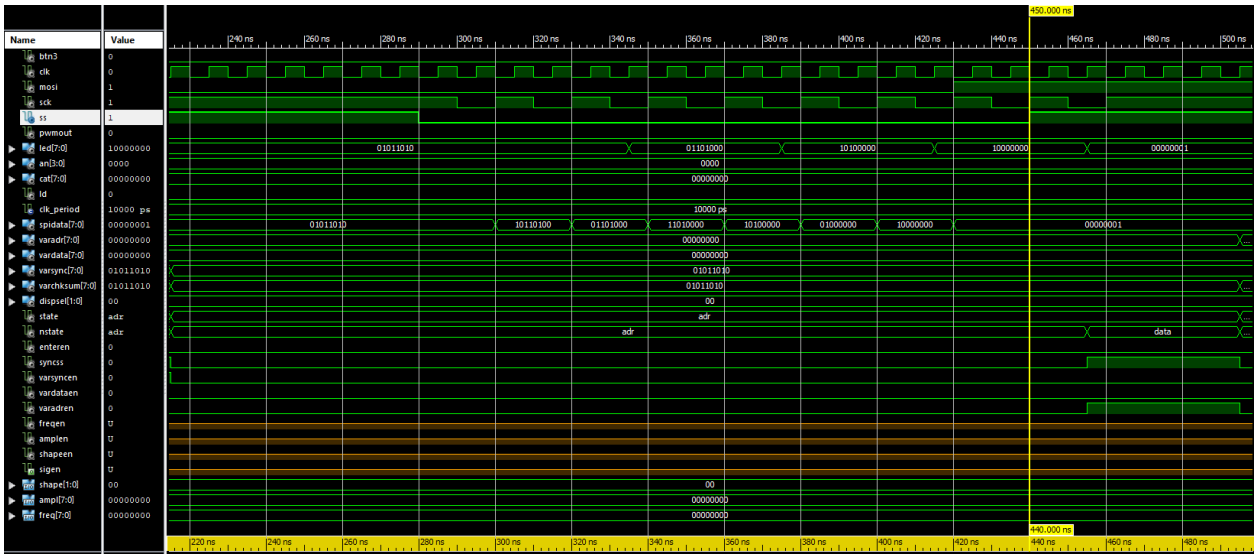
4.2.1 Simulering

Der ses, under simuleringen, indstilling af amplituden samt start af signalgeneratoren. Grundet byte-protokollen, der er valgt fra MCU til FPGA vil der være 4 billeder, henholdvis i tilstanden `Idle`, `Adresse`, `Data` og `Checksum`.



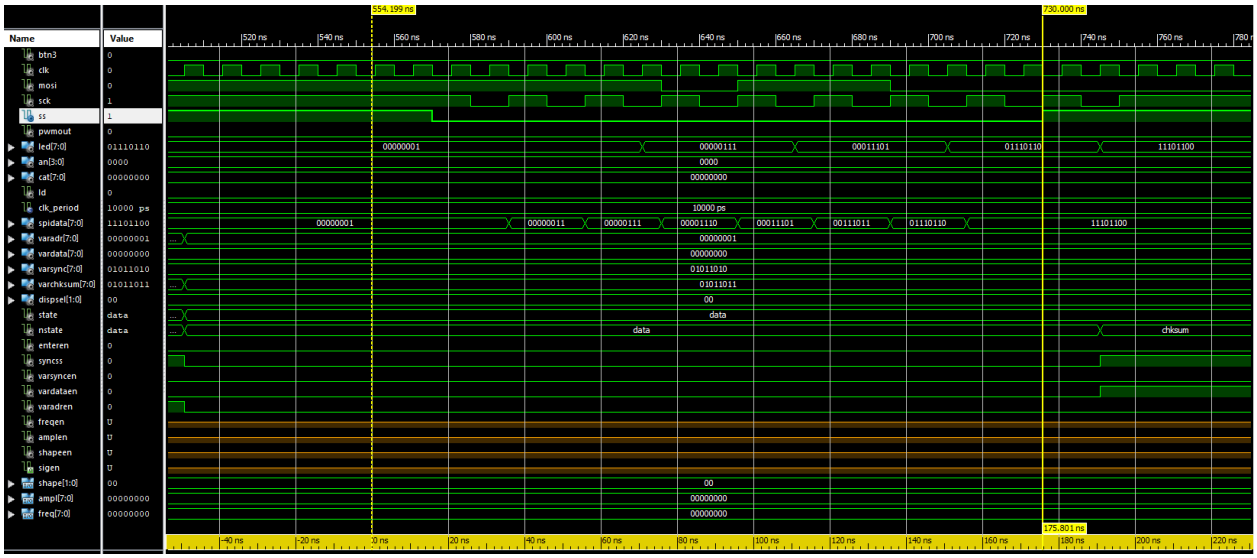
Figur 39: Simulering af signalgeneratorens opstart

Det kan ses på figur 39 hvordan SPI forbindelsen fungerer, samt registerer til at gemme den data, som er kommet via SPI-kommunikationen. Hvis der ses på `MOSI`, `SCK` og `SS` kan man se sammenhængen - når `SS` er lav, og der er en opadgående flanke på `SCK` vil det mestbetydende bit i `MOSI` blive rykket ind på `SPIdata`. Dette vil gentage sig indtil at `SS` bliver høj igen . Det kan dernæst ses der, at der bliver tjekket om det byte som er blevet overført stemmer overens med den ønskede synkroniserings byte, og hvis dette er tilfældet vil `nState` være adresse og der vil blive åbnet til synkroniserings registreret `SyncReg` hvor `SPIdata` vil sættes ind på `varSync`, som kan ses ved 222 ns.
Yderligere, ses det at `ShapeEn`, `AmplEn` og `FreqEn` er udefineret. Dette ændres så snart første indstilling er sat, som ses på figur 42.



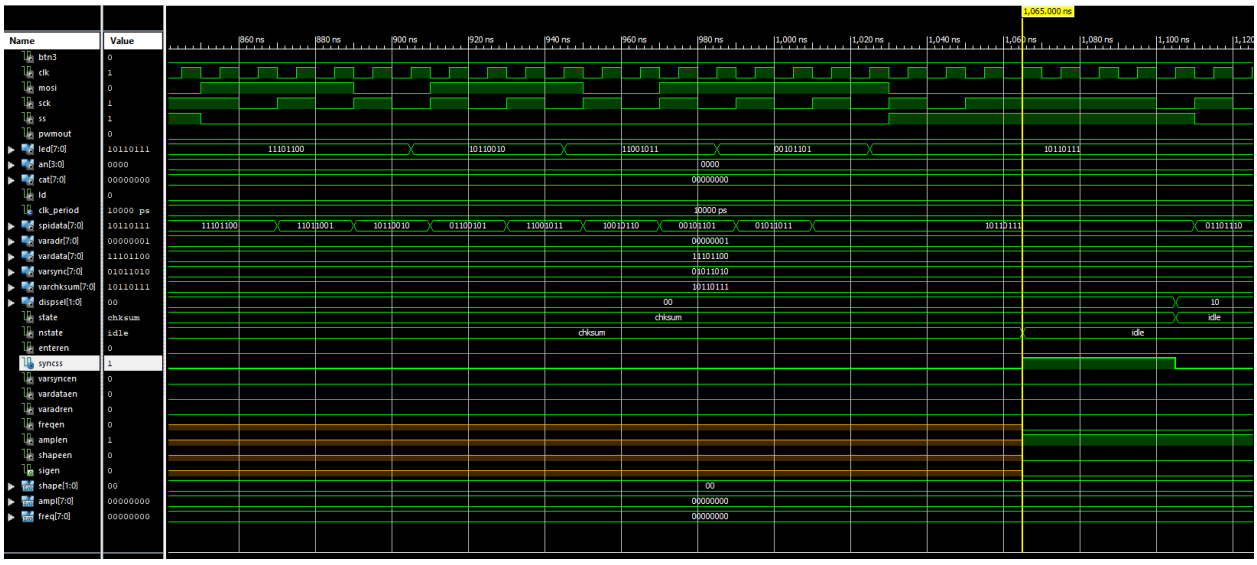
Figur 40: Simulering, der viser indstilling af adr

Samme fremgang som tidligere, data fra `MOSI` indsættes på `SPIdata`, hvor der ventes på en opadgående puls på en synkroniseret `SS`, `SyncSS`. Når denne er høj, vil der blive klart til næste tilstand, samt der vil åbnes for `AdrReg`, som vil gemme på den nye data fra SPI forbindelsen. Det ses at adresse har værdien 0x01 i hexadecimal, hvilket betyder at der ønskes at komme hen til amplituden.



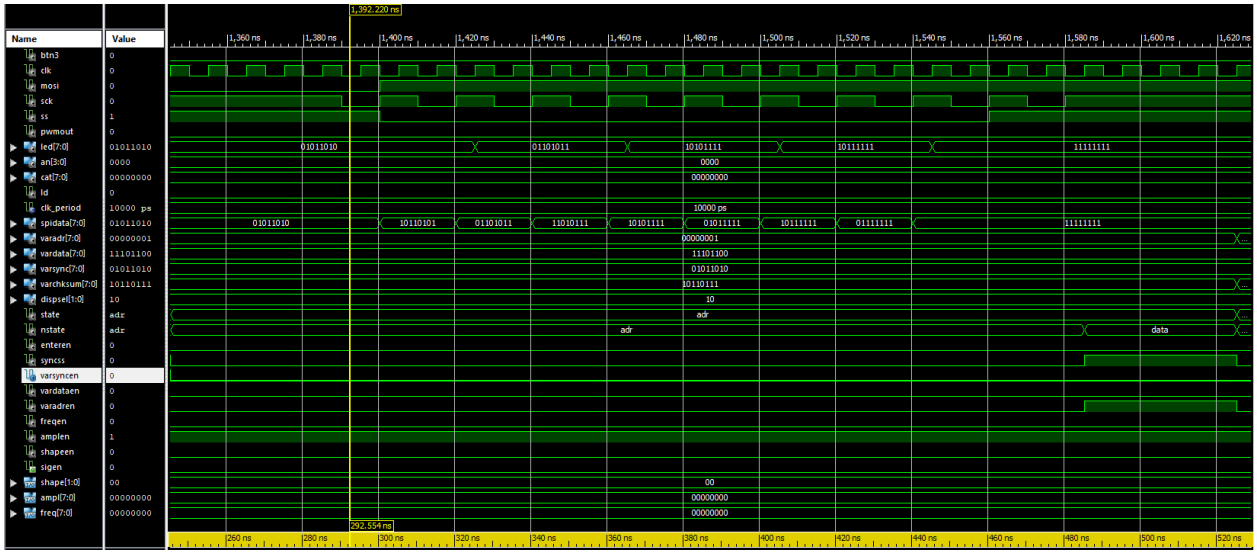
Figur 41: Data tilstanden

Som forklaret under implementering, betyder data ikke noget medmindre der bliver trykket på "enter" knappen i LabView. Derfor vil data værdien kun blive sat ind i et register, uden egentlig at blive brugt. Fremgangsmåden er identisk med adresse, med undtagelsen at data bliver sat ind i `DataReg` istedet.



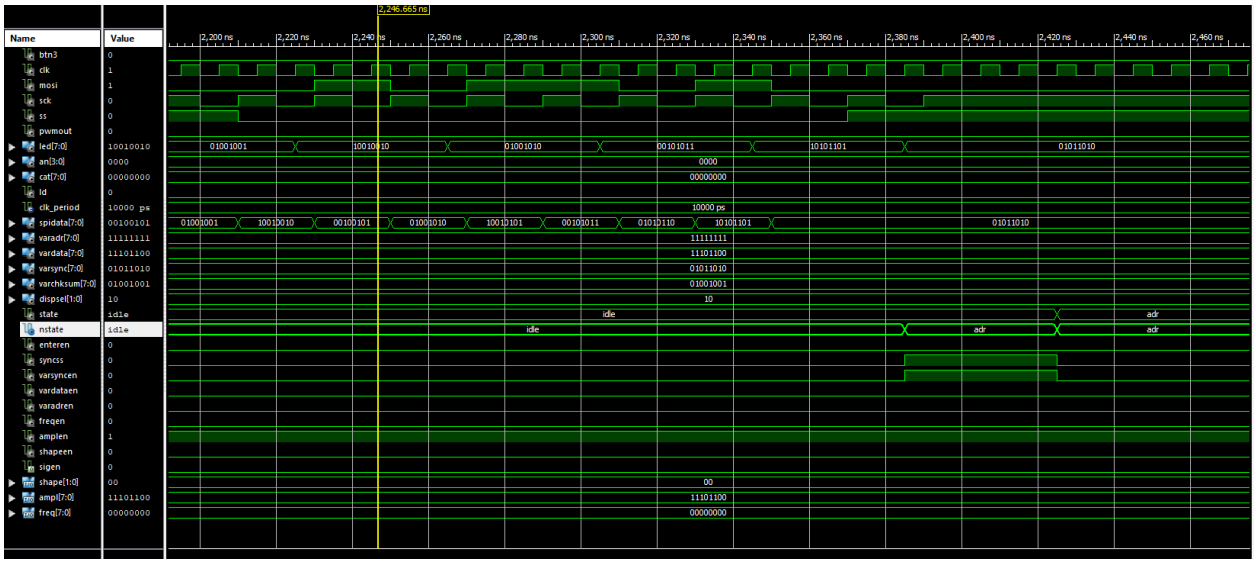
Figur 42: Simulering af checksum tilstanden

Dette er sidste stadie i byte protokollen, hvor der ses om checksummen fra MCU'en stemmer overens med FPGA'ens checksum. Hvis dette er sandt, vil der blive åbnet for det adresse byte der er blevet sendt. I dette tilfælde er det amplituden, hvilket også ses ved den gule streg (1.065 ns). Dette gør således også at de andre tilstande nu er defineret. For at få indsat værdier ind på amplituden skal adressen sættes til 0xFF. Dette ses på figur 43.



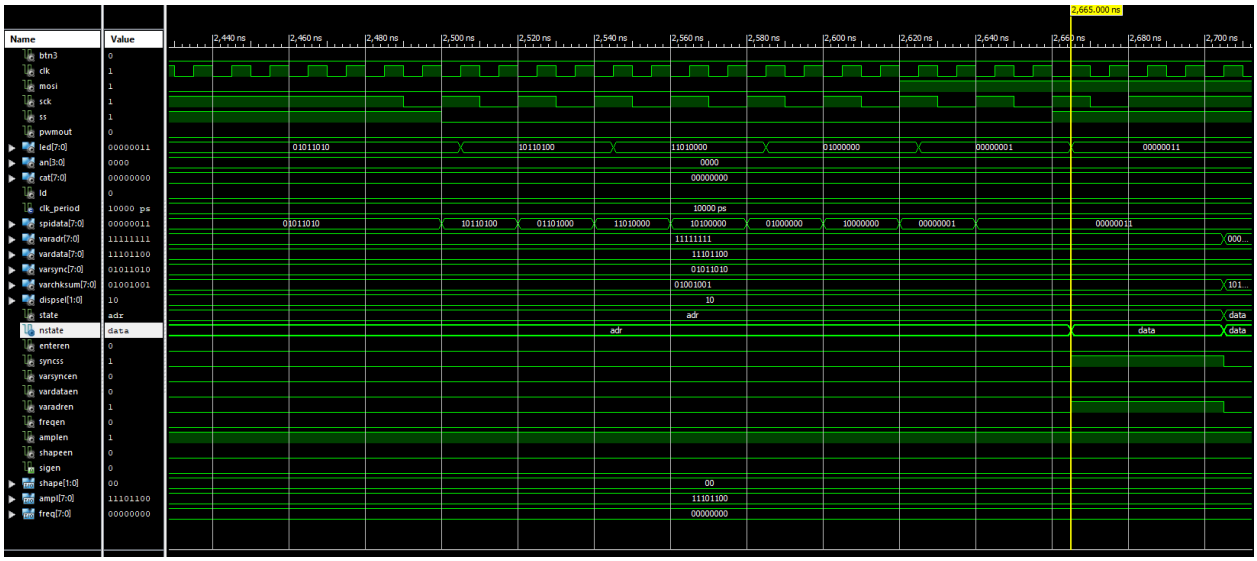
Figur 43: Simulering af indtastning af 'enter'

Det ses, at der stadig åbnet til amplituden via `AmplEn`, og at SPIdata slutter ved hexadecimalet 0xFF. Dette vil gøre at når byte protokollen er gennemført, vil `varData` indsættes på `ampl`, hvilket også ses på figur 44



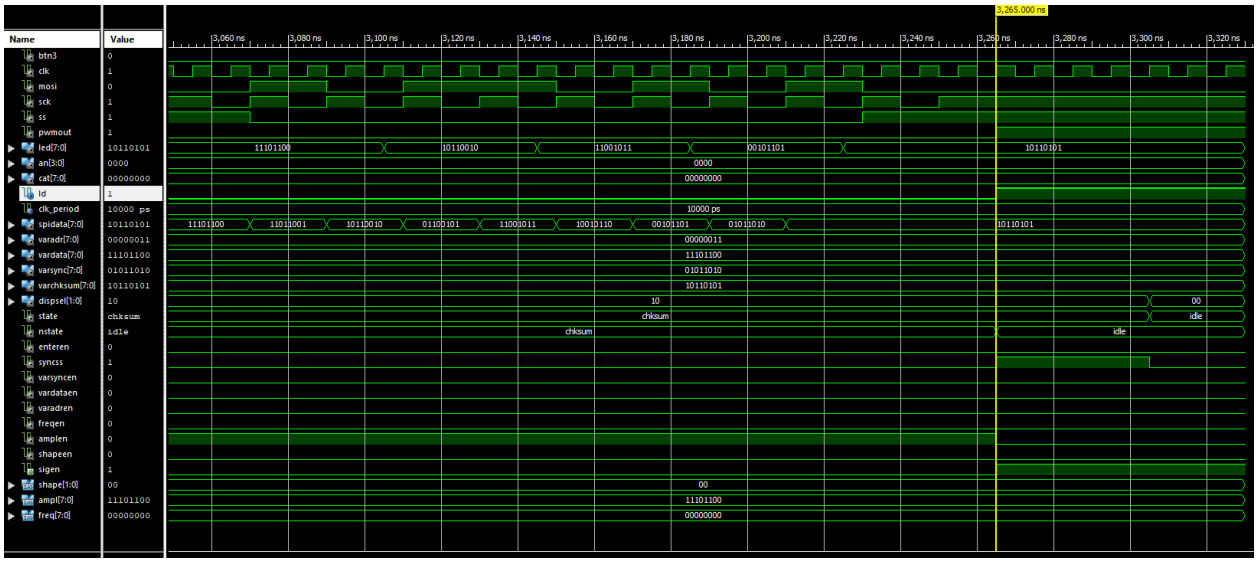
Figur 44: Simulering af indtastning af 'enter' efter gennemført byte protokol

Herefter simuleres der, at signalgeneratoren skal køre. Dette gøres ved at sætte adressen til 0x03, som vist på figur 45.



Figur 45: Adresse ændring til 0x03

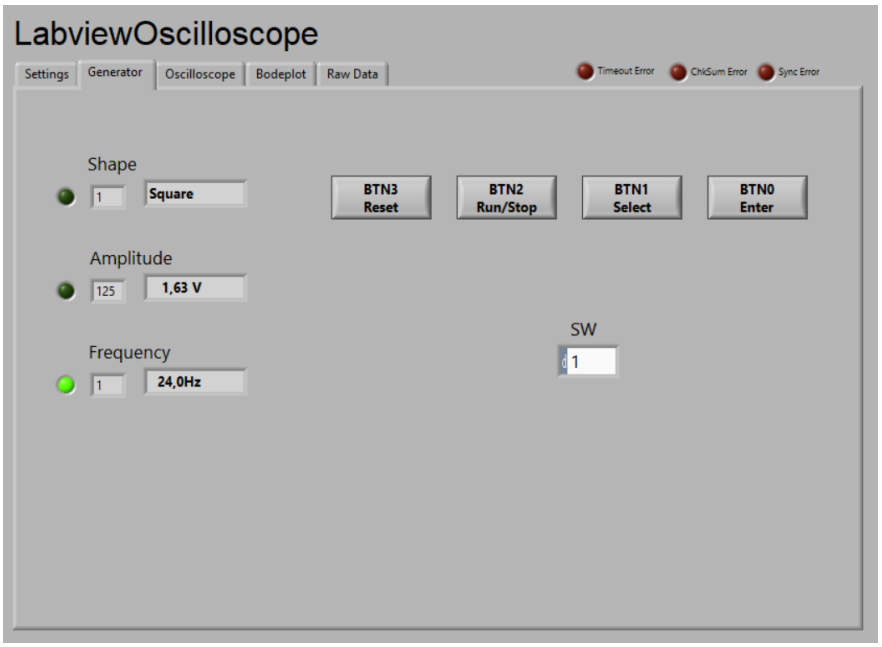
Det kan til sidst ses, at efter byte protokollen, vil `LD` blive høj, og ligeledes vil signalet `SigEn`, hvilket betyder at PWM signalet bliver sendt ud.



Figur 46: Simulering af indtastning af 'enter' efter gennemført byte protokol

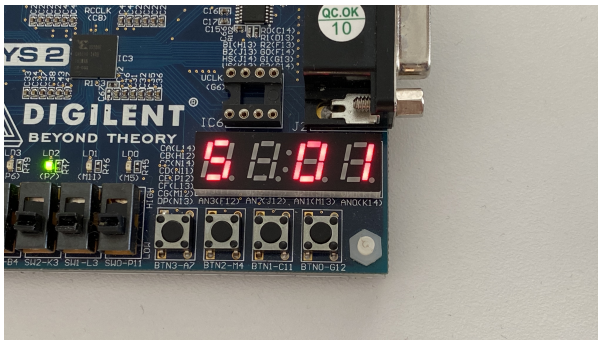
4.2.2 SPI kommunikation

Som det kunne ses ud fra simuleringen, skulle programmet fungere efter hensigten. Dermed bliver der testet fra LabView, ved at indstille shape, amplitude, frekvens og derefter køre programmet. Figur 47 viser hvordan opsætningen af denne test er sat op.

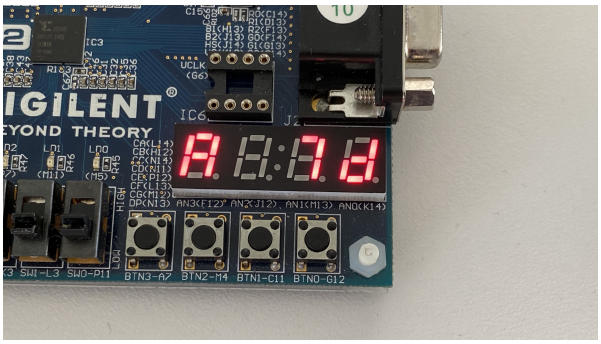


Figur 47: Opsætning signalgenerator fra LabView

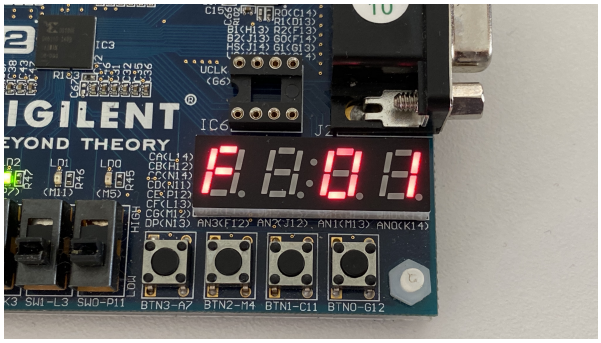
For at se om disse værdier faktisk kommer hen til FPGA boardet, kan der ses på figur 48. De viser således de indstillede tilstandesamt, dog skrevet i hexadecimal værdi istedet for decimal. Ligeledes viser displayet når programmet kører.



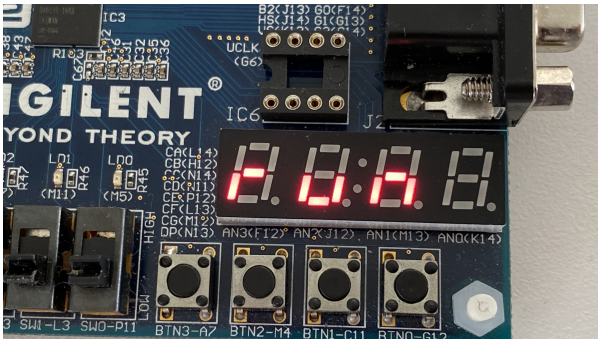
(a) Shape tilstanden



(b) Amplitude tilstanden



(c) Frekvens tilstanden

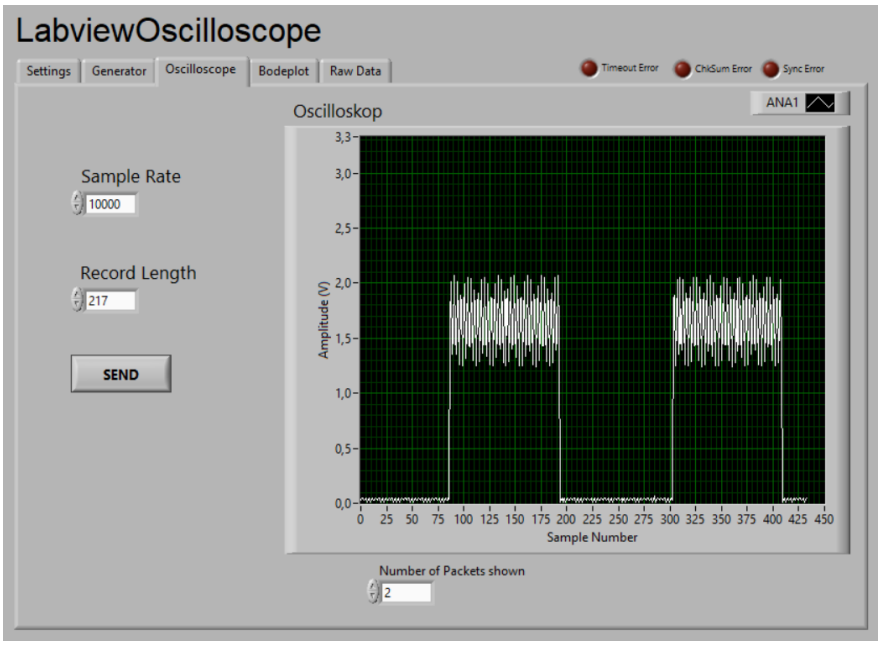


(d) Programmet imens det kører

Figur 48: FPGA board med de 4 tilstande, samt de enkeltes værdier

Dette viser, at der er en fungerende SPI forbindelse, samt at 7-segments displayet ligeledes fungerer.

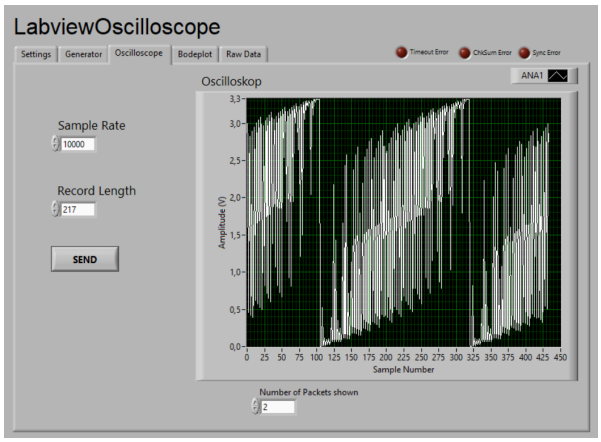
For at se om programmet også kan sende et PWM signal ud, når der trykkes på 'run' i LabView, ses der på figur 49 det ønskede signal. Denne viser et firkantet signal med en amplitude på 1,6V og en lav frekvens.



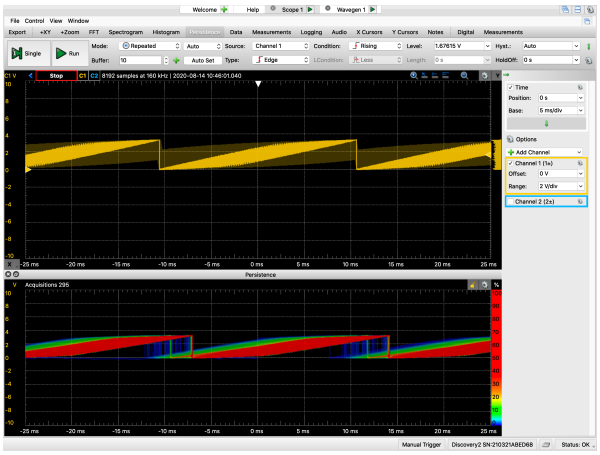
Figur 49: LabViews oscilloscop der læser på PWM signalet

Selvom der er støj når firkantsignalet er høj, kan det anes at middelværdien af støjen ligger omkring de 1,6V hvilket også er det ønskede, samt et relativt tydeligt firkant signal.

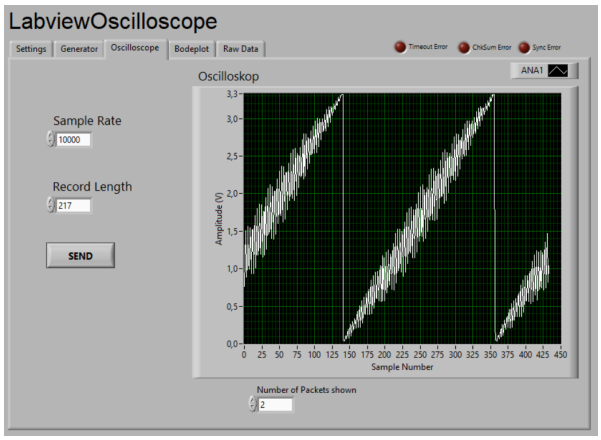
4.2.3 PWM filter



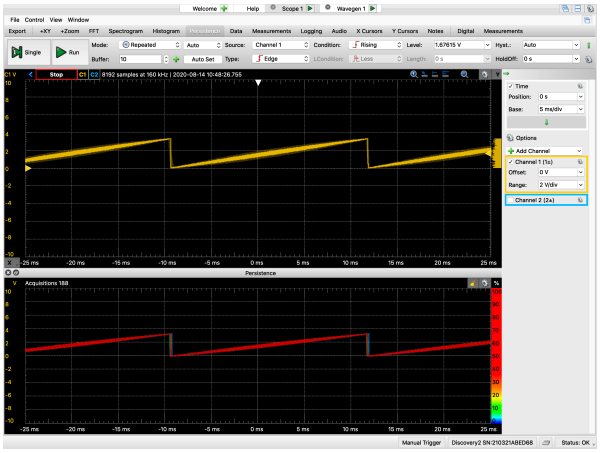
(a) Viser saw uden filter i Labview



(b) Viser PWM filtertest saw uden filter på oscilloscope



(c) Viser osc saw med filter i Labview



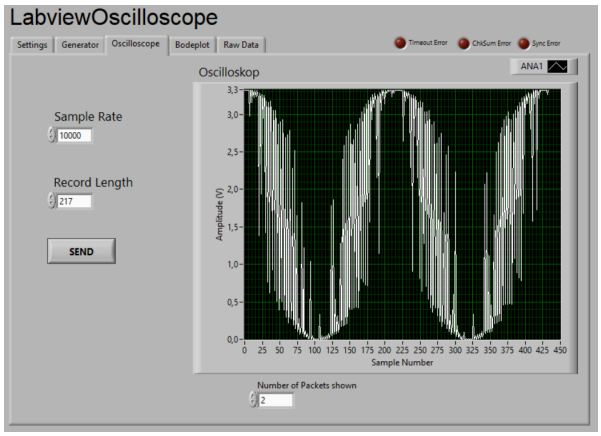
(d) Viser PWM filtertest saw med filter på oscilloscope

Figur 50: PWM savtak signal

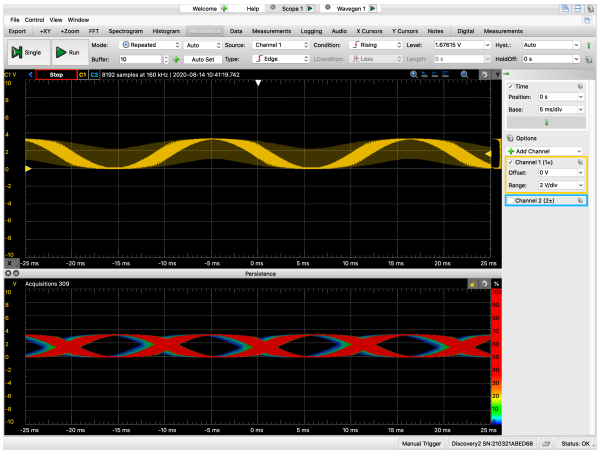
På figur 50 ovenfor, kan der ses savtak PWM signalet før og efter filteret er tilføjet, figur a og b er uden filter hvor man kan se en stor mængde støj, hvliket gør at det kan være svært at se hvliket signal det er, specielt på figur a, hvorimod efter filter er tilføjet på figur c og d er det meget tydeligere at se savtak signalet. Det kan dermed ses på testen at filteret hjælper på at fjerne støjen fra PWM, så der kan fås et renere savtak signal.

UDEN FILTER SAW		MED FILTER SAW	
Frekvens	24 Hz	Frekvens	24 Hz
Amplitude	3.3V	Amplitude	3.3V
Samplerate	10k	Samplerate	10k
Record length	217	Record length	217

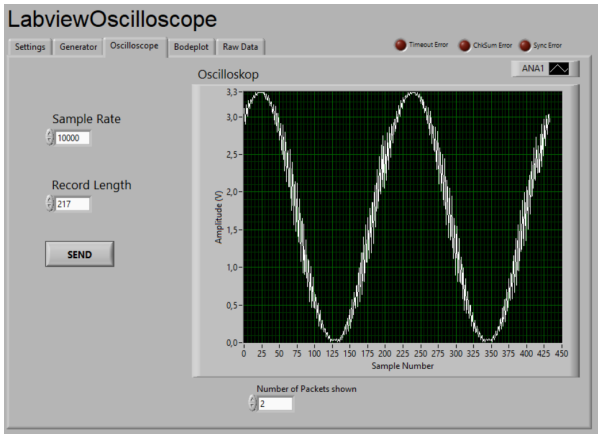
Tabellen ovenfor viser de indtastede værdier, der er blevet brugt under testen af savtak signalet.



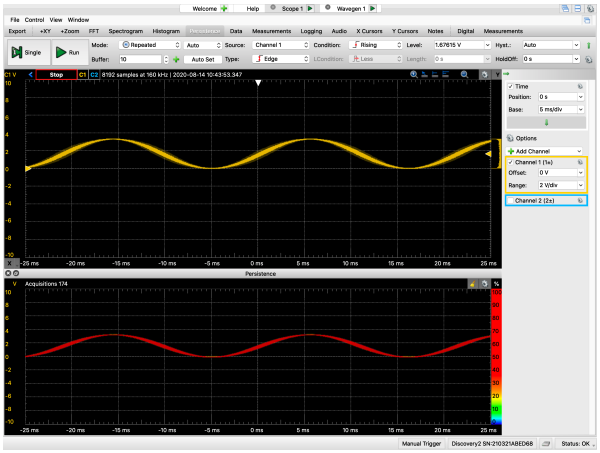
(a) Viser sinus uden filter i Labview



(b) Viser PWM filtertest sinus uden filter på oscilloscope



(c) Viser osc sinus med filter i Labview



(d) Viser PWM filtertest sinus med filter på oscilloscope

Figur 51: PWM sinus signal

Der er valgt at lave den samme test for sinus signalet, som kan ses på figur 51. Hvor det igen ses det samme resultat med at filteret sorterer langt det meste af støjen væk. Dette betyder, at filtrede i begge tilfælde ikke fjerner alt støj men gør signalerne pæne nok til at man tydeligt kan se en fin median igennem begge signaler.

UDEN FILTER SINUS		MED FILTER SINUS	
Frekvens	24 Hz	Frekvens	24 Hz
Amplitude	3.3V	Amplitude	3.3V
Samplerate	10k	Samplerate	10k
Record length	217	Record length	217

Tabellen ovenfor viser de indtastede værdier der er blevet brugt under testen af sinus signalet.

5 Konklusion

5.1 Oscilloskop krav oversigt

Parameter:	Krav:	Krav Opfyldt/Ej opfyldt:
ADC konvertering	Den analoge spænding fra signalgeneratoren skal måles fra 0 til 3.3 V med en opløsning på 8 bit.	Opfyldt
Dataintegritet	Oscilloskopet skal ved alle nedenstående indstillinger kunne køre med kontinuert ubrudte målinger.	Ej opfyldt
Parametre	Oscilloskopets samplerate og Record length skal kunne indstilles fra Labview programmet.	Opfyldt
Samplerate min	Oscilloskopet skal kunne køre ned til 10 sps.	Opfyldt
Samplerate max	Oscilloskopet skal kunne køre op til 5.000 sps. Må gerne køre op til 10.000 sps	Opfyldt
Record length min	Oscilloskopet skal kunne køre med ned til 10 ADC målinger i hver pakke. Den minimale tilladelige record length skal tage højde for sampleraten.	Delvist opfyldt
Record length max	Oscilloskopet skal kunne køre med op til 1000 ADC målinger i hver pakke for alle samplerate.	Opfyldt
RS-232 baudrate	RS232 forbindelsen skal køre med en baud rate på 115.2 kbaud	Opfyldt
RS-232 håndtering	Modtagelse af data fra LabView programmet skal foregå ved interrupt. Transmission kan foregå ved polling eller interrupt.	Opfyldt

5.1.1 Oscilloskop krav redegørelse

ADC konvertering
Kravet er opfyldt og skal ikke dokumenteres yderligere.

Dataintegritet
Dataintegritet er ikke blevet opfyldt, da der kræves kontinuerte målinger for alle opgivede tilstande. Det har ikke været muligt at knække koden til at få sendt datapakker kontinuerligt, dog kan der ved en del frekvenser findes flotte målinger (se test 5 i Dataintegritet) Desuden har PWM filtrets støj tendens til at "skjule" den manglende kontinuitet især ved lave frekvenser.

Parametre
Oscilloskopets samplerate eller med andre ord ADC'ens samplerate som styres af timer 1 interrupt og dermed timer 1's TOP værdi kan indstilles i LabView. Ligeledes kan record length indstilles i LabView som vist under 4.1.2

Samplerate min/max
Samplerate kan indstilles fra 9,972 sps til 10,001 ksps hvilket må siges at være så tæt på kravet om en samplerate på minimum 10 sps og ønskeligt maksimum på 10 ksps. Se 4.1.4 for yderligere information.

Record length min/max
Er delvis blevet opfyldt, da der ikke tages højde for samplerate ved den mindste record length.

RS-232 baudrate
Kravet er opfyldt og skal ikke dokumenteres yderligere.

RS-232 håndtering
Kravet er opfyldt og skal ikke dokumenteres yderligere.

5.2 Signalgenerator krav oversigt

Parameter:	Krav:	Krav Opfyldt/Ej opfyldt:
PWM filter	Der skal designes et lav-pas filter der på passende vis udglatter de digitale PWM pulser.	Opfyldt
Parametre	Signalgeneratorens signalform (SHAPE) , amplitude (AMPL) og frekvens (FREQ) skal kunne indstilles fra Labview programmet.	Opfyldt
	SHAPE, AMPL og FREQ kan gøres synligt på syv segment displayet.	Opfyldt
Sinus signal	Der kan implementeres en look-up tabel i VHDL koden der gør det muligt at signalgeneratoren kan lave et sinus-formet signal.	Opfyldt
SPI baudrate	SPI forbindelsen skal køre med en baudrate på 500 kbaud.	Opfyldt
SPI håndtering	To-vejs SPI kommunikation kan implementeres f.eks. med acknowledge handshake.	Ej opfyldt
SPI protokol	Der skal vælges og implementeres en robust byteorienteret protokol til at overføre SHAPE, AMPL og FREQ.	Opfyldt
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.	Opfyldt
PWM signal	PWM skal maksimalt køre med 10 kHz da ADC'en maksimalt kører med 10 ksps.	Delvist opfyldt

5.2.1 Signalgenerator krav redegørelse

PWM filter
Det kan ses ud fra afsnit 4.2.3, at PWM filteret reducerer støjen i høj grad og gør det muligt, at forskellen på de forskellige signaler og dermed må opfylde kravet.

Parametre
Ligeledes kan der konkluderes ud fra afsnit 4.2.2, at det er muligt at styre signalformerne via LabView, samt

det kan ses på FPGA boardets indbyggede 7-segments display

Sinus signal
Der er blevet tilføjet et 'look-up table' i VHDL koden, der gør det muligt at genererer et sinusformet signal. Denne kan ses under testafsnittet 4.2.3.

SPI baudrate
Der er i SPI.h, under `SPCR` registeret er der sat en høj forbindelse ved `SPI2X` samt `SPR1` hvilket sætter `SCK` frekvensen ned til 500kHz (udregningen ses på ligning (3)). Og da det er denne frekvens, der sender ét bit ved hver opadgående flanke, betyder det at der sendes 500k bit hvert sekund, ergo en SPI-baudrate på 500k baud.

$$\mathrm{SCK}_{freq} = \frac{f_{osc}}{32} \tag{3}$$

⇕

$$500 \cdot 10^3 = \frac{16 \cdot 10^6}{32}$$

SPI håndtering
Der er ikke blevet implementeret en to-vejs SPI kommunikation, grundet tidsmangel.

SPI protokol
Der er blevet implementeret sikker byte protokol som kan ses i tabellen. Denne sørger for der ikke sendes nogle bytes før at sync-byten er rigtig og checksummen sørger for, at alt overført data stemmer overens med FPGA'ens udregnede. Checksummen udregnes med ligningen (1). Og hvis bare én af de to ikke stemmer, vil der ikke blive overført/indsat noget data.

Sync	Adresse	Data	CheckSum

SPI test
Denne er bevist ved at kunne styre FPGA boardet igennem LabView, da al kommunikation til FPGA boardet er via SPI forbindelse. Dette ses i afsnit 4.2.2

PWM signal
Der er ikke ændret i `SigGenDatapath`, således at frekvensen er begrænset til 10kHz. Den eneste måde at begrænse PWM signalet, er ved at ikke sætte det højere end 10kHz i LabView programmet.

5.3 Gruppearbejde

Der har været et godt samarbejde i gruppen, hvor der problemfrit blev uddelt arbejdsopgave og delgrupper. Der har under hele forløbet været god kommunikation iblandt gruppen og dette har gjort, at der er blevet arbejdet optimalt og uden at lave noget dobbeltarbejde eller på anden måde spildt hinandens tid.

5.4 Overordnet konklusion

Alt i alt kan der konkluderes, at næsten alle strenge krav er opnået, med undtagelse af dataintegritet. Det er muligt at styre FPGA boardet via LabView, og at se et genkendeligt, filteret signalform på LabViews oscilloscop, hvilket også betyder at ADC'en samt UART forbindelsen fungerer. Derudover er der også opnået flere blødekrav, herunder op til 10.000 samples per sekund, signalformernes tilstand kan ses på 7-segmentsdisplayet, det er muligt at genererer et sinusformet signal og protokollen fra LabView til MCU'en er skiftet fra Zero16 til LRC-8 som giver en mere sikker overførsel.

Derimod er der stadig meget støj på PWM signalet, som kunne havde været mindsket, hvis der blev modificeret på `SigGenDatapath` og sænket signalform frekvensen og øget PWM frekvensen. Dette blev ikke lavet, grundet tidsmangel og prioritering af de strengekrav.

A Appendix

A.1 Oversigt over c-modulerne

A.1.1 main.c

Globals:
volatile char flag
char rx_buffer[11]
char adc_tx[1000]
char fullFlag
char transmitBuf[1010]
int buf_ctr
int readIndex
int writeIndex
int i
unsigned int sample_flag
unsigned int newTopVal
float sample_rate

Functions:
void writeBuffer();
char readBuffer(void);

A.1.2 adc.h

Functions:
void adc_init()

A.1.3 counter.h

Functions:
void counter1_init()

A.1.4 labview.h

Globals:
char RS_flag
char active
char shape
char amp
char freq
int LOP
int sync
char type
char cs
int sample
int record
char tx_buffer[1010]

Functions:
void sendAmpl(char data)
void sendShape(char data)
void sendEnter(char data)
void sendFreq(char data)
void sendRun(char data)
void generator_display(char *buffer)
void oscilloscope_display(char * buffer)
void generate_data(char *buffer)
void checksum(char *buffer)

A.1.5 spi.h

Functions:
void putcSPI_master(char cx);
void SPI_init()
void putcSPI_master(char cx)
void sendEnter(char data)
void sendShape(char data)
void sendAmpl(char data)
void sendFreq(char data)
void sendRun(char data)

A.1.6 uart.h

Functions:
void uart0_init()
unsigned char getchUSART0(void)
void putchUSART0 (char tx)
void enableReceiveItr()
void disableReceiveItr()

A.2 MCU kode

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\main.c 1

```
/*****************************************************************************************
 * MAIN                             DTU - DIPLOM                                         *
 *****************************************************************************************
 * Author(s):       Kim Holmberg Christensen & Jørgen Drelicharz Greve (s181554 / s181519)*
 * Creation date:   04.08.2020                                                           *
 * Update date:     04.08.2020                                                           *
 * Filename:        main.c                                                               *
 * Version:         1.0                                                                  *
 *                                                                                       *
 *****************************************************************************************/

#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include "uart.h"
#include "Labview.h"
#include "SPI.h"
#include "adc.h"
#include "counter.h"


void writeBuffer();
char readBuffer(void);


volatile char flag = 0;
char rx_buffer[11] = {0};
char adc_tx[1000] = {0};                            // Array for circular buffer
char fullFlag = 0;                                  // Flag to prohibit overwriting of circular buffer
char transmitBuf[1010] = {0};
int buf_ctr = 0;
int readIndex = 0;                                  // Index for reading circular buffer
int writeIndex = 0;                                 // Index for writing to circular buffer
int i = 0;
unsigned int sample_flag = 0;
unsigned int newTopVal = 0;                         // Variable for new counter TOP value
float sample_rate;                                  // Used for converting sample rate

int main(void)
{

```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\main.c 2

```
sei();
enableReceiveItr();
uart0_init( MYUBRRF );
SPI_init();
adc_init();                                         // Initialize ADC
counter1_init();                                    // Initialize counter/timer 1 for ADC interrupt

    while (1)
    {
        if ( sample_flag == 1)                      // When ADC sample is ready the value is stored in the circular buffer.
        {
            writeBuffer(ADCH);

            sample_flag = 0;
        }

        if(fullFlag == 1)                           // When the buffer is full, data will be transferred to the "transmitBuf".
                                                    // The amount of data is controlled by the record length
            {
                type = 0x02;

                for (i = 0; i < record; i++)        // Read data from circular buffer
                {
                    transmitBuf[i+5] = readBuffer();
                }

                generate_data(transmitBuf);

                for (i = 0 ; i < record + 7 ; i++)  // Send data to LabView
                {
                    putchUSART0(tx_buffer[i]);
                }
            }

        if (flag == 1)                              // UART recieve complete
        {
            flag = 0;

            checksum(rx_buffer);                    // Check sum of recieved package

            if(rx_buffer[0] == 0x55 && rx_buffer[1] == 0xAA && rx_buffer[rx_buffer[3]-1] == cs) //Check sync and check sum
            {                                                                                  // If the last byte in the package match
                type = rx_buffer[4];                                                           // cs it will be accepted
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\main.c 3

```
87
88                  if (type == 0x01)                   // Type = Generator tab
89                  {
90                      generator_display(rx_buffer);   // The received data will be examined, prepare data back to LabView and relay data to
91                                                      // FPGA board.
92
93                      generate_data(rx_buffer);       // Data package created
94
95                      for (i = 0 ; i < tx_buffer[3] ; i++) // Send data to LabView
96                      {
97                          putchUSART0(tx_buffer[i]);
98                      }
99                  }
100                 else if (type == 0x02)              // Type = Oscilloscope tab
101
102                 {
103                     oscilloscope_display(rx_buffer); // Received 2 bytes per data field (Sample rate & Record length) converted into one
104                                                      // 16-bit integer.
105
106                     sample_rate =(2.5*100000/sample)-1;
107
108                     newTopVal = sample_rate;        // Enter new TOP value
109                                                     // newTop = 24980 = 10 sps
110                                                     // newTop = 49 = 5.000 sps
111                                                     // newTop = 23 = 10.000 sps
112                     if (newTopVal < 23)             // ----------------------------------
113                     {
114                         newTopVal = 23;
115                     }                               // To make sure that newTopVal is within 10-10.000 sps
116                     if (newTopVal > 24980)
117                     {
118                         newTopVal = 24980;
119                     }                               // ----------------------------------
120                 }
121             }
122         }
123     }
124
125
126
127 }
128
129 ISR(USART0_RX_vect)                                 // Interrupt Service Routine
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\main.c 4

```
130 {
131     rx_buffer[buf_ctr++] = UDR0;
132
133     if (buf_ctr == rx_buffer[3] && rx_buffer[3] != 0)
134     {
135         flag = 1;
136         buf_ctr = 0;
137     }
138 }
139
140 ISR(ADC_vect)
141 {
142     // ---------------------------- ONLY FOR TESTING  -----------------------------
143     sample_flag = 1;
144     // ------------------------------------------------------------------------------
145
146     OCR1A = newTopVal;                              // Set new TOP value from main
147
148     PORTG &=~ (1<<PG5);
149 }
150
151 ISR(TIMER1_COMPB_vect)                              // ADC auto trigger source
152 {
153     PORTG |= (1<<PG5);
154 }
155
156 void writeBuffer(int data)
157 {
158     if(fullFlag == 1)
159     {
160
161     }
162     else
163     {
164         adc_tx[writeIndex] = data;
165
166         if (writeIndex == sizeof(adc_tx)/sizeof(adc_tx[0]))
167         {
168             writeIndex = 0;
169         }
170         else
171         {
172             writeIndex++;
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\main.c 5

```
173         }
174
175         if(writeIndex == readIndex)
176         {
177             fullFlag = 1;
178         }
179     }
180 }
181
182 char readBuffer(void)
183 {
184     char data = 0;
185
186     if (readIndex == writeIndex && fullFlag != 1)
187     {
188
189     }
190
191     else
192     {
193         data = adc_tx[readIndex];
194
195         if (readIndex == sizeof(adc_tx)/sizeof(adc_tx[0]))
196         {
197             readIndex = 0;
198         }
199
200         else
201         {
202             readIndex++;
203         }
204
205         fullFlag = 0;
206     }
207     return data;
208 }
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\adc.h 1

```
/*************************************************************************************************
 * ADC                                      DTU - DIPLOM                                         *
 *************************************************************************************************
 * Author(s):       Jørgen Drelicharz Greve s181519                                              *
 * Creation date:   04.08.2020                                                                   *
 * Update date:     04.08.2020                                                                   *
 * Version:         1.0                                                                          *
 * Filename:        adc.h                                                                        *
 *************************************************************************************************
 * INFORMATION:                                                                                  *
 *************************************************************************************************
 * - 8 bit ADC                                                                                   *
 * - CTC mode                                                                                    *
 * - Auto trigger on TIMER1_COMPB_vect                                                           *
 *                                                                                               *
 *************************************************************************************************/

void adc_init()
{
    ADMUX  |= (1<<ADLAR);                          // Left adjust result for 8 bit precision
    ADCSRA |= (1<<ADEN);                           // ADC enable p.292
    ADCSRA |= (1<<ADIE);                           // Interrupt enable p.293
    ADCSRA |= (1<<ADATE);                          // Auto trigger enable p.293
    ADCSRA |= (0<<ADPS0) | (0<<ADPS1) | (1<<ADPS2); // Prescaler p.293
    ADCSRB |= (1<<ADTS0) | (0<<ADTS1) | (1<<ADTS2); // Auto trigger source p. 295
    DIDR0   = 0xff;                                // Digital input disable (0xff = 11111111 = off)
    DIDR2   = 0xff;                                // Digital input disable (0xff = 11111111 = off)
}


```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\counter.h 1

```
/*****************************************************************************************
 * TIMER/COUNTER                          DTU - DIPLOM                                   *
 *****************************************************************************************
 * Author(s):       Jørgen Drelicharz Greve s181519                                      *
 * Creation date:   04.08.2020                                                           *
 * Update date:     04.08.2020                                                           *
 * Version:         1.0                                                                  *
 * Filename:        counter.h                                                            *
 *****************************************************************************************
 * INFORMATION:                                                                          *
 *****************************************************************************************
 * TIMER/COUNTER 1 SETTINGS                                                              *
 * Prescale 64                                                                           *
 * OCR1A = 24980 => 10 Hz                                                                *
 * OCR1A = 49    => 5 kHz                                                                *
 * OCR1A = 23    => 10 kHz                                                               *
 *                                                                                       *
 *****************************************************************************************/

void counter1_init()                                    // ADC control counter
{
    TCCR1B |= (0<<WGM10)  | (0<<WGM11) | (1<<WGM12);    // Waveform generation mode p.159
    TCCR1B |= (1<<CS10)   | (1<<CS11)  | (0<<CS12);     // Prescale p.161
    TIMSK1 |= (1<<OCIE1B);                              // Output compare match B
    OCR1A   = 24980;                                    // 16 bit TOP value

}

```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 1

```
/*****************************************************************************************
 * LABVIEW                              DTU - DIPLOM                                     *
 *****************************************************************************************
 * Author(s):       Kim Holmberg Christensen s181554                                     *
 * Creation date:   10.08.2020                                                           *
 * Update date:     10.08.2020                                                           *
 * Filename:        Labview.h                                                            *
 * Version:         1.0                                                                  *
 *                                                                                       *
 *****************************************************************************************/

#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char  RS_flag = 0;
char active = 0;
char shape = 0;
char amp = 0;
char freq = 0;
int LOP = 0;
int sync = 0x55AA;
char type = 0;
char cs = 0x0000;

int sample = 0;
int record = '0';

char tx_buffer[1010] = {0};

void sendAmpl(char data);
void sendShape(char data);
void sendEnter(char data);
void sendFreq(char data);
void sendRun(char data);

void generator_display(char *buffer)
{
    char state;
    char BTN;

```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 2

```
    BTN = buffer[5];


    //------------------------------- CHECK BUTTON ----------------------------------------------//

    if (BTN == 0x00)// Enter button pressed
    {
        state = 1;
    }

    if (BTN == 0x01) // Select button pressed
    {
        state = 2;
    }

    if (BTN == 0x02) // Run/Stop button pressed
    {
        state = 3;
    }

    if (BTN == 0x03) // Reset button pressed
    {
        state = 4;
    }


    switch(state)
    {
        case 1: //Enter state

        {
            //------------------------------- SHAPE ----------------------------------------------//

            if (active == 0x00 && buffer[6] < 0x04 && RS_flag == 0) // Test state of active "LED". For shape, 3 is the maximum number handled.
                                                                    // RS_Flag = 0 ensure that FPGA is not in run cycle.
                {
                    shape = buffer[6];
                    sendEnter(shape);
                }
            else
                {
                    shape = shape;                                  // If wrong input shape will remain unchanged
                }
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 3

```
87
88              //------------------------------- AMPLITUDE ---------------------------------------------//
89
90              if (active == 0x01 && buffer[6] <= 0xFF && RS_flag == 0)// test state of active "LED". For amplitude, value has to be between
                  0-255.
91                                                                      // RS_Flag = 0 ensure that FPGA is not in run cycle.
92                  {
93                      amp = buffer[6];
94                      sendEnter(amp);
95                  }
96              else
97                  {
98                      amp = amp;
99                  }
100
101                 //------------------------------- FREQUENCY ---------------------------------------------//
102
103             if (active == 0x02 && buffer[6] <= 0xFF && RS_flag == 0) // test state of active "LED". For frequency, value has to be between
                  0-255.
104                                                                       // RS_Flag = 0 ensure that FPGA is not in run cycle.
105                 {
106                     freq = buffer[6];
107                     sendEnter(freq);
108                 }
109             else
110                 {
111                     freq = freq;
112                 }
113
114                 //---------------------------------------------------------------------------------------//
115             break;
116         }
117
118         case 2: // Select state
119         {
120             if (buffer[6] < 0x03 && RS_flag == 0)   // SW value used for selecting "SHAPE", "AMPLITUDE" or "FREQUENCY".
121                                                     // A value is sent to the FPGA board in order to show the current state on the 7-seg
                                    display.
122                                                     // The value sent is not used by the FPGA board.
123             {
124                 active = buffer[6];
125                 sendShape(active);
126             }
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 4

```
127             else
128             {
129                 active =  active;
130             }
131
132                 switch(active)
133                     {
134                         case 0:
135                             {
136                                 if(RS_flag == 0)
137                                     {
138                                         sendShape(active);
139                                     }
140                                 break;
141                             }
142
143                         case 1:
144                             {
145                                 if(RS_flag == 0)
146                                     {
147                                         sendAmpl(active);
148                                     }
149                                 break;
150                             }
151
152                         case 2:
153                             {
154                                 if(RS_flag == 0)
155                                     {
156                                         sendFreq(active);
157                                     }
158                                 break;
159                             }
160
161                         default:
162                             {
163                                 break;
164                             }
165
166                 }
167         break;
168         }
169
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 5

```
170         case 3: //Run/Stop state
171         {
172                 if(RS_flag == 0)    // RS flag is used as a toggle switch, used for the FPGA board.
173                                     // First time the button is pressed the first if case is activated, next cycle the second and so on.
174                     {
175                         sendRun(buffer[6]);
176                         RS_flag = 1;
177
178                         break;
179                     }
180
181                 if(RS_flag == 1) // If  the button is in "Stop" state, the last state of LED (active) will be lit and the state will be
                      forwarded to the FPGA board.
182                     {
183                         if (active == 0x00) // Shape state
184                             {
185                                 sendShape(shape);
186                             }
187
188                         if (active == 0x01) // Amplitude state
189                             {
190                                 sendAmpl(amp);
191                             }
192
193                         if (active == 0x02) // Frequency state
194                             {
195                                 sendFreq(freq);
196                             }
197
198                 RS_flag = 0;
199                 break;
200             }
201         }
202
203         case 4: // Reset state
204         {
205             active = 0; // Reset the active "LED" in LabView to "Shape"
206             amp = 0;    // Reset amplitude to 0
207             shape = 0;  // Reset Shape to 0 (Constant signal)
208             freq = 0;   // Reset frequency to 0
209
210             sendAmpl(amp);  //Required for resetting the FPGA board.
211             sendEnter(amp); //Required for resetting the FPGA board.
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\Labview.h 6

```
212
213             sendFreq(freq); //Required for resetting the FPGA board.
214             sendEnter(freq); //Required for resetting the FPGA board.
215
216             sendShape(shape); //Required for resetting the FPGA board.
217             sendEnter(shape); //Required for resetting the FPGA board.
218
219             RS_flag = 0;
220
221             break;
222         }
223
224         default:
225         {
226             break;
227         }
228     }
229 }
230
231
232 void oscilloscope_display(char * buffer)
233 {
234     sample = (buffer[5] << 8) | buffer[6]; // Convert received sample rate for two 8-bit integers into one 16 bit integer
235     record = (buffer[7] << 8) | buffer[8]; // // Convert received record length for two 8-bit integers into one 16 bit integer
236 }
237
238
239 void generate_data(char *buffer)
240 {
241     LOP = 0;
242     int i = 0;
243     cs = 0;
244
245     memset(tx_buffer, 0 , sizeof(tx_buffer)); //Reset tx_buffer to 0
246
247     tx_buffer[0] = sync >> 8; // First sync byte
248     tx_buffer[1] = sync & 0xFF; //Second sync byte
249
250     if(type == 0x01) // Data returning to LabView "Generator tab"
251     {
252         LOP = 0x0B;
253         tx_buffer[4] = 0x01;
254         tx_buffer[5] = active;
```

```
255         tx_buffer[6] = shape;
256         tx_buffer[7] = amp;
257         tx_buffer[8] = freq;
258     }
259
260     if (type == 0x02) // Data returning to LabView "Oscilloscope tab"
261     {
262         LOP = record + 7; // Record = Record length of the expected data package.
263
264         tx_buffer[4] = type;
265
266         for (i = i ; i < record ; i++) // Fill in the transmit buffer with the data from the ADC buffer
267             {
268                 tx_buffer[i+5] = buffer[i+5];
269             }
270     }
271
272     tx_buffer[2] = LOP >> 8; // First "Length of packet" byte
273
274     tx_buffer[3] = LOP & 0xFF; // Second "Length of packet" byte
275
276     for (i = 0 ; i < LOP-2; i++) //Create LRC 8 check sum
277         {
278             cs = cs^tx_buffer[i];
279         }
280
281     tx_buffer[LOP-2] = cs >> 8; // First check sum byte
282
283     tx_buffer[LOP-1] = cs & 0xFF; // Second check sum byte
284 }
285
286 void checksum(char *buffer)
287 {
288     int i = 0;
289     cs = 0;
290
291     LOP = (buffer[2] << 8) | buffer[3];
292
293     for (i = 0 ; i < LOP-2; i++)
294         {
295             cs = cs^buffer[i];
296         }
297 }
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\SPI.h 1

```
/*****************************************************************************************
 * SPI.h                            DTU - DIPLOM                                         *
 *****************************************************************************************
 * Author(s):       Kim Holmberg Christensen s181554                                     *
 * Creation date:   11.08.2020                                                           *
 * Update date:     11.08.2020                                                           *
 * Filename:        SPI.h                                                                *
 * Version:         1.0                                                                  *
 *                                                                                       *
 *****************************************************************************************/

#define F_CPU 16000000UL                        //CLK speed
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void putcSPI_master(char cx);

void SPI_init()
{
        DDRB |= (1<<DDB2)|(1<<DDB1)|(1<<DDB0);
        SPCR |= (1<<SPE)|(1<<MSTR);
        SPCR |= (1<<SPI2X)|(1<<SPR1);           //SPI-baudrate 500kHz
        PORTB |=(1<<PB0);
}

void putcSPI_master(char cx)
{
    PORTB &=~(1<<PB0);                          //Wake slave

    SPDR = cx;

    while(!(SPSR&(1<<SPIF)));                   //Wait for data to be shifted out

    PORTB |=(1<<PB0);                           //sleep slave
}

void sendEnter(char data)
{
    char enterChkSum;
    putcSPI_master(0x5A);
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\SPI.h 2

```

    putcSPI_master(0xFF);

    putcSPI_master(data);

    enterChkSum = (0x5A^0xFF)^data;
    putcSPI_master(enterChkSum);
}

void sendShape(char data)
{
    char shapeChkSum;
    putcSPI_master(0x5A);
    putcSPI_master(0x00);
    putcSPI_master(data);
    shapeChkSum = 0x5A^0x00^data;
    putcSPI_master(shapeChkSum);
}

void sendAmpl(char data)
{
    char amplChkSum;
    putcSPI_master(0x5A);
    putcSPI_master(0x01);
    putcSPI_master(data);
    amplChkSum = 0x5A^0x01^data;
    putcSPI_master(amplChkSum);
}

void sendFreq(char data)
{
    char freqChkSum;
    putcSPI_master(0x5A);
    putcSPI_master(0x02);
    putcSPI_master(data);
    freqChkSum = 0x5A^0x02^data;
    putcSPI_master(freqChkSum);
}

void sendRun(char data)
{
    char runChkSum;
    putcSPI_master(0x5A);
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\SPI.h 3

```
87      putcSPI_master(0x03);
88      putcSPI_master(data);
89      runChkSum = 0x5A^0x03^data;
90      putcSPI_master(runChkSum);
91  }
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\uart.h 1

```
/*****************************************************************************************
 * UART                                    DTU - DIPLOM                                  *
 *****************************************************************************************
 * Author(s):       Kim Holmberg Christensen & Jørgen Drelicharz Greve (s181554 / s181519)*
 * Creation date:   26.03.2020                                                            *
 * Update date:     05.08.2020                                                            *
 * Filename:        uart.h                                                                *
 * Version:         1.1                                                                   *
 *****************************************************************************************
 * INSTRUCTIONS:                                                                         *
 *****************************************************************************************
 * Include uart.h (this file) in main.c:                                                 *
 *                                                                                       *
 *  USE FOLLOWING FUNCTIONS:                                                             *
 *  - uart0_init()              to initialize USART                                      *
 *  - getchUSART0(void)         to receive character                                     *
 *  - putchUSART0 (char tx)     to send character                                        *
 *  - getsUSART0(char *ptr)     to receive string                                        *
 *  - putsUSART0(char *ptr)     to send string                                           *
 *                                                                                       *
 *  Page numbers in this file (p.XXX) refers to pages in Atmel ATmega2560 data book.     *
 *                                                                                       *
 *  See bottom of this file for main.c example code.                                     *
 *****************************************************************************************/

#define F_CPU 16000000UL                    // Defines F_CPU as clock speed
#include <avr/io.h>
#include <avr/interrupt.h>                  // Needed for interrupts
#define BAUD 115200                         // Sets the baud rate
#define MYUBRRF F_CPU/8/BAUD-1              // Scales the clock speed (p.233)

// ---------------------------- INITIALIZE USART 0 -------------------------------------------------------
void uart0_init()
{
    UCSR0A =    (1<<U2X0);                  // Full duplex (p.223)
    UCSR0B |=   (1<<RXEN0) | (1<<TXEN0);    // Enable receive and transmit (p.224)
    UCSR0C |=   (1<<UCSZ01)| (1<<UCSZ00);   // Set frame format, 8 data bit (p.226)
    UCSR0C &=~  (1<< UMSEL00);
    UCSR0C &=~  (1<< UMSEL01);
    UCSR0C &=~  (1<<USBS0);                 // Stop bit select (1 or 2 bit) (p.226)
    UCSR0C &=~  (1<<UPM01);                 // Parity mode (p.226)
    UCSR0C &=~  (1<<UPM00);                 // Parity mode (p.226)
    UBRR0   =   MYUBRRF;                    // Clock generation (p.233)
```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\uart.h 2

```

}

// ---------------------------- GET CHARACTER ------------------------------------------------------------
unsigned char getchUSART0(void)
{
    while(!(UCSR0A & (1<<RXC0)));           // Wait until character is received

    return UDR0;
}

// ---------------------------- SEND CHARACTER -----------------------------------------------------------
void putchUSART0 (char tx)
{
    while(!(UCSR0A & (1<<UDRE0)));          // wait for empty transmit buffer
    UDR0 = tx;
}



void enableReceiveItr()
{
    UCSR0B |=   (1<<RXCIE0);                // Enable Receive complete interrupt (p.224)
}

void disableReceiveItr()
{
    UCSR0B &=~  (1<<RXCIE0);                // Enable Receive complete interrupt (p.224)
}



// ---------------------------- MAIN.C EXAMPLE CODE ------------------------------------------------------
/*
#include "uart.h"

int main()
{
    uart0_init();                           // Initialize USART

    char array[50] = " It works :) ";

```

C:\Users\jorge\Desktop\TestUARTLabViewfinalfinal\TestUARTLabView\TestUARTLabView\uart.h 3

```
87      while (1)
88      {
89          putsUSART0(array);                  // Sends string, view in terminal window
90      }
91  }
92  */
```

A.3 VHDL kode

```
1   ----------------------------------------------------------------------------------
2   -- Company:
3   -- Engineer:
4   --
5   -- Create Date:    16:17:22 01/05/09
6   -- Design Name:
7   -- Module Name:    SigGenTop - Behavioral
8   -- Project Name:
9   -- Target Device:
10  -- Tool versions:
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  ---- Uncomment the following library declaration if instantiating
26  ---- any Xilinx primitives in this code.
27  --library UNISIM;
28  --use UNISIM.VComponents.all;
29
30  entity SigGenTop is
31    Port ( BTN3   : in std_logic;
32           Clk    : in std_logic;
33           MOSI   : in std_logic;
34           SCK    : in std_logic;
35           SS     : in std_logic;
36           LED    : out std_logic_vector(7 downto 0);
37           An     : out std_logic_vector(3 downto 0);
38           Cat    : out std_logic_vector(7 downto 0);
39           LD     : out std_logic;
40           PWMOut : inout std_logic);
41  end SigGenTop;
42
43  architecture Behavioral of SigGenTop is
44
45  signal Mclk, DispClk, SigEn: std_logic;
46  signal Disp: std_logic_vector(19 downto 0);
47  signal Ampl, Freq: std_logic_vector(7 downto 0);
48  signal Shape: std_logic_vector(1 downto 0);
49
50  begin
51
52  U0: entity WORK.DivClk
53      port map(Reset => BTN3, Clk => Clk, TimeP => 4, Clk1 => Mclk);
54
55  U4: entity WORK.DivClk
56      port map(Reset => BTN3, Clk => Clk, TimeP => 50e3, Clk1 => DispClk);
57
```

```
58   U1: entity WORK.SigGenSPIControl -- Knapperne BTN2 - BTN0 er blevet udskiftet med
     MOSI signal
59       port map(Reset => BTN3, Disp => Disp, Shape => Shape, Ampl => Ampl, Freq => Freq,
     SigEN=> SigEN,
60       LED => LED, MOSI => MOSI, SCK => SCK, SS => SS, Clk => Mclk);
61
62
63   U2: entity WORK.SigGenDataPath generic map (PWMinc => "0000001")
64       port map(Reset => BTN3, Clk => Mclk, Shape => Shape, Ampl => Ampl, Freq => Freq,
     SigEN=> SigEN, PWMOut => PWMOut);
65
66   U3: entity WORK.SevenSeg5
67       port map(Reset => BTN3, Clk => DispClk, Data => Disp, An => An, Cat => Cat);
68
69   U5: LD <= PWMOut;
70
71
72   end Behavioral;
73
```

```
1   ---------------  Clock divider  ----------------------
2   library IEEE;
3   use IEEE.std_logic_1164.all;
4   use IEEE.STD_LOGIC_ARITH.ALL;
5   use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7   entity DivClk is
8       port ( Reset: in STD_LOGIC;      -- Global Reset (BTN1)
9              Clk: in STD_LOGIC;     -- Master Clock (50 MHz)
10             TimeP: in integer;     -- Time periode of the divided clock (50e6)
11             Clk1: out STD_LOGIC);   -- Divided clock1 (1 Hz)
12  end DivClk;
13
14  architecture DivClk_arch of DivClk is
15  --constant MaxCnt1: integer:= 14;
16  signal Cnt1: integer range 0 to 25000000;  -- 24 bit counter
17  signal Clear1: STD_LOGIC;
18  signal Clk1_D: STD_LOGIC;
19
20  begin
21
22    -- T-register with enable and async.reset
23    Div1Reg: process(clk,Reset)
24    begin
25      if Reset = '1' then Clk1_D <= '0';     -- async. reset
26      elsif (clk'event and clk ='1') then
27        if Clear1= '1' then                -- enable
28          Clk1_D <= not Clk1_D;
29        end if;
30      end if;
31    end process Div1Reg;
32
33    Div1Dec: process(Cnt1, TimeP) -- Kombinatorisk
34    begin
35      Clear1 <= '0';
36      if Cnt1 = TimeP/2 then
37        Clear1 <= '1';
38      end if;
39    end process Div1Dec;
40
41    -- 24 bit up-counter with clear and async. reset
42    Div1Cnt:process(clk,Reset)
43    begin
44      if Reset = '1' then Cnt1 <= 1;        -- async. reset
45      elsif (clk'event and clk ='1') then
46        if Clear1 = '1' then Cnt1 <= 1;   -- clear
47        else Cnt1 <= Cnt1 + 1; end if;
48      end if;
49    end process Div1Cnt;
50
51    Clk1 <= Clk1_D;
52
53  end DivClk_arch;
54
55
```

```
1   ----------------------------------------------------------------------------
2   -- Company:        DTU
3   -- Engineer:       Esben Leerbeck & Mads Stender
4   --
5   -- Create Date:    10:07:10 05/12/09
6   -- Design Name:
7   -- Module Name:    SigGenControl - Behavioral
8   -- Project Name:   Signal Generator
9   -- Target Device:  Spartan 3
10  -- Tool versions:
11  -- Description:    Control circuit for the Signal Generator system
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  entity SigGenSPIControl is
26    Port ( Reset  : in std_logic;
27           Clk    : in std_logic;
28           MOSI   : in std_logic;
29           SCK    : in std_logic;
30           SS     : in std_logic;
31           LED    : out std_logic_vector(7 downto 0);
32           Disp   : out std_logic_vector(19 downto 0);
33           Shape  : inout std_logic_vector(1 downto 0);
34           Ampl   : inout std_logic_vector(7 downto 0);
35           Freq   : inout std_logic_vector(7 downto 0);
36           SigEn  : out std_logic);
37  end SigGenSPIControl;
38
39  architecture Behavioral of SigGenSPIControl is
40
41  signal ShapeEN, AmplEN, FreqEN, varAdrEn, varDataEn, varSyncEn, SyncSS, SyncSSOld1,
    SyncSSOld2, enterEn: std_logic;
42  signal SPIData, varAdr, varData, varSync, varChkSum: std_logic_vector(7 downto 0);
43  signal DispSel: std_logic_vector(1 downto 0);
44  type StateType is (Idle, Adr, Data, ChkSum);
45  signal State, nState: StateType;
46
47  begin
48
49  ----------------------------------------------------------
50  -- Logikblokke
51
52  varChkSum <= (varSync XOR varAdr) XOR varData;
53
54  SyncSS <= SyncSSOld1 AND NOT SyncSSOld2;
55
56  ----------------------------------------------------------
```

```
57   -- Display multiplexer
58
59   DispMux: Disp <= X"F1230" when DispSel = "0" else
60                    X"4F0" & Freq when DispSel = X"1" else
61                    X"4A0" & Ampl when DispSel = X"2" else
62                    X"450" & "000000" & Shape;
63
64   ------------------------------------------------------------
65   -- Clock synchronizer
66
67   DelReg: process(Reset, Clk, SS)
68   begin
69      if Reset = '1' then SyncSSOld1 <= '0';
70      elsif Clk'event and Clk = '1' then
71         SyncSSOld1 <= SS;
72      else
73         SyncSSOld1 <= SyncSSOld1;
74      end if;
75   end process;
76
77   DelReg2: process(Reset, Clk, SS)
78   begin
79      if Reset = '1' then SyncSSOld2 <= '0';
80      elsif Clk'event and Clk = '1' then
81         SyncSSOld2 <= SyncSSOld1;
82      else
83         SyncSSOld2 <= SyncSSOld2;
84      end if;
85   end process;
86
87   ------------------------------------------------------------
88   -- Register
89
90   ShapeReg: process (Reset,  Clk, varData)
91   begin
92     if Reset = '1' then Shape <= "00";
93     elsif Clk'event and Clk= '1' then
94       if ShapeEn = '1' then
95         if enterEn = '1' then
96            Shape <= varData(1 downto 0);
97         else
98            Shape <= Shape;
99         end if;
100      else
101         Shape <= Shape;
102      end if;
103    end if;
104  end process;
105
106  AmplReg: process (Reset, Clk, varData)
107  begin
108    if Reset = '1' then Ampl <= X"00";
109    elsif Clk'event and Clk = '1' then
110      if AmplEn = '1' then
111        if enterEn = '1' then
112           Ampl <= varData;
113        else
```

```
114             Ampl <= Ampl;
115           end if;
116         else
117           Ampl <= Ampl;
118         end if;
119       end if;
120     end process;
121
122     FreqReg: process (Reset, Clk, varData)
123     begin
124       if Reset = '1' then Freq <= X"00";
125       elsif Clk'event and Clk = '1' then
126         if FreqEn = '1' then
127           if enterEn = '1' then
128             Freq <= varData;
129           else
130             Freq <= Freq;
131           end if;
132         end if;
133       end if;
134     end process;
135
136     DispReg: process(ShapeEn, AmplEn, FreqEn, varAdr, varData, Reset, Clk) -- Viser
        signalform tilstanden
137     begin
138        if Reset = '1' then DispSel <= "00";
139        elsif Clk'event and Clk = '1' then
140           if ShapeEn = '1' then
141              DispSel <= "11";
142           elsif AmplEn = '1' then
143              DispSel <= "10";
144           elsif FreqEn = '1' then
145              DispSel <= "01";
146           else
147              DispSel <= "00";
148           end if;
149        end if;
150     end process;
151
152     SPIReg: process(SPIdata, MOSI, SCK, SS, Reset)  -- Shiftregister som tager MOSI
        signal og shifter 8-bit ind på SPIdata
153     begin                                           -- Serial in - parallel out
154        if Reset = '1' then SPIdata <= X"00";
155        elsif SCK'event and SCK ='1' then
156           if SS = '0' then
157              SPIdata <= SPIdata(6 downto 0) & MOSI;
158           else
159              SPIdata <= SPIdata;
160           end if;
161        end if;
162     end process;
163
164     SyncReg: process(Reset, varSync, Clk, SPIdata) -- Gemmer SPIdata som varSync
165     begin
166        if Reset = '1' then varSync <= X"00";
167        elsif Clk'event and Clk = '1' then
168           if varSyncEn = '1' then
```

```
169            varSync <= SPIdata;
170         else
171            varSync <= varSync;
172         end if;
173      end if;
174   end process;
175
176   AdrReg: process(varAdr, Reset, Clk, SPIdata) -- Gemmer SPIdata som varAdr
177   begin
178      if Reset = '1' then varAdr <= X"00";
179      elsif Clk'event and Clk = '1' then
180         if varAdrEn = '1' then
181            varAdr <= SPIdata;
182         else
183            varAdr <= varAdr;
184         end if;
185      end if;
186   end process;
187
188   DataReg: process(Reset, varData, Clk, SPIdata) -- Gemmer SPIdata som varData
189   begin
190      if Reset = '1' then varData <= X"00";
191      elsif Clk'event and Clk = '1' then
192            if varDataEn = '1' then
193               varData <= SPIdata;
194            else
195               varData <= varData;
196            end if;
197      end if;
198   end process;
199
200   -- Kan bruges til at se SPI dataen
201   --LEDReg: process(Reset, Clk, varSync, varAdr, varData, varChkSum)
202   --begin
203   -- if Reset = '1' then LED <= X"00";
204   -- elsif Clk'event and Clk = '1' then
205   --    LED <= SPIdata;
206   -- end if;
207   --end process;
208
209   StateReg: process (Reset, Clk)
210   begin
211     if Reset = '1' then State <= Idle;
212     elsif Clk'event and Clk = '1' then
213       State <= nState;
214     end if;
215   end process;
216
217   -------------------------------------------------------------------------------
218   -- Tilstandsmaskine
219
220   StateSPIDec: process(state, SPIdata, nState, SyncSS, varSync, varAdr, varData,
      varChkSum, enterEn)
221   begin
222      nState <= Idle;
223      varSyncEn <= '0';
224      varAdrEn <= '0';
```

```
225      varDataEn <= '0';
226      enterEn <= '0';
227
228
229      case State is
230
231         when Idle =>   -- EnterEn nulstilles, så den kun er høj under ChkSum
    tilstanden
232          enterEn <= '0';
233            if SyncSS = '1' then -- Der ses efter en SS puls
234               if SPIdata = X"5A" then -- Hvis SPIdataen er lig med syncByten åbnes
    syncReg
235                  varSyncEn <= '1';
236                  nState <= Adr;
237               else
238                  nState <= Idle;
239               end if;
240            else
241               nState <= Idle;
242            end if;
243
244         when Adr =>
245
246            if SyncSS = '1' then -- SS puls vil åbne adrReg
247               varAdrEn <= '1';
248               nState <= Data;
249            else
250               nState <= Adr;
251            end if;
252
253         when Data =>
254
255            if SyncSS = '1' then -- SS puls vil åbne dataReg
256               varDataEn <= '1';
257               nState <= ChkSum;
258            else
259               nState <= Data;
260            end if;
261
262         when ChkSum =>
263            if SyncSS = '1' then
264               if  varChkSum = SPIdata then -- Hvis MCU chksum stemmer overens med FPGA
    chksum, vil der ses på varAdr
265
266               -- Her tændes der for en signalform og denne forbliver tændt, medmindre
    der kommer en ny adresse værdi,
267               -- på nær enterknappen (X"FF")
268                  if varAdr= X"00" then -- Shape state
269                     ShapeEn <= '1';
270                     AmplEn <= '0';
271                     FreqEn <= '0';
272                     SigEn <= '0';
273                     nState <= Idle;
274                  elsif varAdr = X"01" then -- Ampl state
275                     ShapeEn <= '0';
276                     AmplEn <= '1';
277                     FreqEn <= '0';
```

```
278                     SigEn <= '0';
279                     nState <= Idle;
280                  elsif varAdr = X"02" then -- Freq state
281                     ShapeEn <= '0';
282                     AmplEn <= '0';
283                     FreqEn <= '1';
284                     SigEn <= '0';
285                     nState <= Idle;
286                  elsif varAdr = X"03" then -- Run state
287                     ShapeEn <= '0';
288                     AmplEn <= '0';
289                     FreqEn <= '0';
290                     SigEn <= '1';
291                     nState <= Idle;
292                  elsif varAdr = X"FF" then -- Enter buttom
293                     enterEn <= '1';
294                     nState <= Idle;
295                  else
296                     ShapeEn <= '0';
297                     AmplEn <= '0';
298                     FreqEn <= '0';
299                     SigEn <= '0';
300                     nState <= Idle;
301                  end if;
302               else
303                  nState <= Idle;
304               end if;
305            else
306               nState <= ChkSum;
307         end if;
308      end case;
309   end process;
310
311   end Behavioral;
312
```

```
1   ----------------------------------------------------------------------------
2   -- Company:        DTU
3   -- Engineer:       Peter Brauer
4   --
5   -- Create Date:    29/4/15
6   -- Design Name:
7   -- Module Name:    SigGenDatapath - Behavioral
8   -- Project Name:   Signal Generator
9   -- Target Device:  Spartan 3
10  -- Tool versions:
11  -- Description:    Datapath circuit for the Signal Generator system
12  --
13  -- Dependencies:   Lookup table "SinusLUT" for generating a sinus signal
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  entity SigGenDatapath is
26    generic( PWMinc : std_logic_vector(6 downto 0) := "0010000" );
27    Port ( Reset  : in std_logic;
28           Clk    : in std_logic;
29           SigEN  : in std_logic;
30           Shape  : in std_logic_vector(1 downto 0);
31           Ampl   : in std_logic_vector(7 downto 0);
32           Freq   : in std_logic_vector(7 downto 0);
33           PWMOut : out std_logic);
34  end SigGenDatapath;
35
36  architecture Behavioral of SigGenDatapath is
37
38  signal SigCnt, nSigCnt, FreqCnt: std_logic_vector(11 downto 0);
39  signal Sig, SigSquare, SigSaw, SigSinus : std_logic_vector(7 downto 0);
40  signal SigAmpl: std_logic_vector(6 downto 0);
41  signal PWMcnt: std_logic_vector(6 downto 0) := "0000000";
42  signal PWM, PWMwrap : std_logic;
43
44  begin
45
46  FreqDec: FreqCnt <= "00" & Freq(7 downto 6) & Freq(5 downto 4) & '0' & Freq(3 downto 2
    ) & '0' & Freq(1 downto 0);
47
48  FreqAdd: nSigCnt <= SigCnt + FreqCnt;
49
50
51  SigReg: process (Reset, Clk)
52  begin
53    if Reset = '1' then SigCnt <= X"000";
54    elsif Clk'event and Clk = '1' then
55      if PWMwrap = '1' then
56        SigCnt <= nSigCnt;
```

```
 57       end if;
 58     end if;
 59   end process;
 60
 61   SinusDec : entity WORK.SinusLUT PORT MAP (clka => Clk, addra => SigCnt, douta =>
      SigSinus);
 62
 63   PWMcount: process(Reset, Clk)
 64   variable PWMcntvar: std_logic_vector(7 downto 0);
 65   begin
 66     if Reset = '1' then PWMcntvar := "00000000";
 67     elsif Clk'event and Clk = '1' then
 68       PWMcntvar := PWMcntvar + PWMinc;
 69         if PWMcntvar > "10000000" then
 70            PWMcntvar := "00000000";
 71         end if;
 72     end if;
 73     PWMcnt <= PWMcntvar(6 downto 0);
 74   end process;
 75
 76   PWMdec: PWMwrap <= '1' when PWMcnt = "0000000" else '0';
 77
 78   SquareDec: SigSquare <= "00000000" when SigCnt < X"800" else "11111111";
 79
 80   SawDec: SigSaw <= SigCnt(11 downto 4);
 81
 82   SigMux: Sig <= X"FF" when Shape = "00" else
 83                  SigSquare when Shape = "01" else
 84                  SigSaw when Shape = "10" else
 85                  SigSinus;
 86
 87
 88   AmplDec: process(Ampl, Sig)
 89   variable MulA, MulB, MulC: std_logic_vector(15 downto 0);
 90   --variable MulC: std_logic_vector(6 downto 0);
 91   begin
 92     MulB := X"00" & Sig;
 93     MulA := X"00" & Ampl;
 94     MulC := X"0000";
 95     for j in 0 to 15 loop
 96       if MulA(j) = '1' then
 97         MulC := MulC + MulB;
 98       end if;
 99       MulB := MulB(14 downto 0) & '0';
100     end loop;
101   --  MulC := MulC + 1;
102     SigAmpl <= MulC(15 downto 9);
103   --  SigAmpl <= "1111111";
104   end process;
105
106
107   PWMcomp: PWM <= '1' when PWMcnt <= SigAmpl else '0';
108
109   PWMon: PWMout <= PWM when SigEn = '1' else '0';
110
111   end Behavioral;
112
```

```
1   ----------  Driver to sevensegment display  ---------------
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.STD_LOGIC_ARITH.ALL;
5   use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7   entity SevenSeg5 is
8       Port ( Reset,Clk: in  std_logic;
9              Data :   in  std_logic_vector (19 downto 0); -- Binary data
10             cat :    out std_logic_vector(7 downto 0);  -- Common cathodes
11             an :     out std_logic_vector(3 downto 0)); -- Common Anodes
12  end SevenSeg5;
13
14  architecture SevenSeg_arch of SevenSeg5 is
15  signal DispCount: integer range 0 to 3;
16  signal DataN: std_logic_vector (3 downto 0);
17  signal CatInt, CatData, CatSign: std_logic_vector (7 downto 0);
18  signal AnInt: std_logic_vector (3 downto 0);
19
20  begin
21
22
23     DispCountReg: process(Reset, Clk)
24     begin
25      if Reset = '1' then
26      DispCount <= 0;
27      elsif Clk'event and Clk = '1' then
28        if DispCount = 3
29          then DispCount <= 0;
30          else DispCount <= DispCount + 1; end if;
31        end if;
32     end process DispCountReg;
33
34     DispCountDec: process(DispCount, Data)
35     begin
36       case DispCount is
37         when 0 =>
38           AnInt <= "1110";         --  Display 1 activated
39           DataN <= Data(3 downto 0);
40         when 1 =>
41           AnInt <= "1101";         --  Display 1 activated
42           DataN <= Data(7 downto 4);
43         when 2 =>
44           AnInt <= "1011";         --  Display 1 activated
45           DataN <= Data(11 downto 8);
46         when others =>
47           AnInt <= "0111";         --  Display 1 activated
48           DataN <= Data(15 downto 12);
49       end case;
50      end process DispCountDec;
51
52     with DataN SELect       --  Activate segments acc. to Data
53       CatData <= "11000000" when "0000",   --0
54              "11111001" when "0001",   --1
55              "10100100" when "0010",   --2
56              "10110000" when "0011",   --3
57              "10011001" when "0100",   --4
```

```
58                "10010010" when "0101",    --5
59                "10000010" when "0110",    --6
60                "11111000" when "0111",    --7
61                "10000000" when "1000",    --8
62                "10011000" when "1001",    --9
63                "10001000" when "1010",    --A
64                "10000011" when "1011",    --b
65                "11000110" when "1100",    --C
66                "10100001" when "1101",    --d
67                "10000110" when "1110",    --E
68                "10001110" when "1111",    --F
69                "11111111" when others;  --blank
70
71      with DataN SELect       --  Activate segments acc. to Data
72        CatSign <= "11111111" when "0000",    --Blank
73                "10101111" when "0001",    -- "r"
74                "11100011" when "0010",    --"u"
75                "10101011" when "0011",    --"n"
76                "10011001" when "0100",    --4
77                "10010010" when "0101",    --5
78                "10000010" when "0110",    --6
79                "11111000" when "0111",    --7
80                "10000000" when "1000",    --8
81                "10011000" when "1001",    --9
82                "10001000" when "1010",    --A
83                "10000011" when "1011",    --b
84                "11000110" when "1100",    --C
85                "10100001" when "1101",    --d
86                "10000110" when "1110",    --E
87                "10001110" when "1111",    --F
88                "11111111" when others;  --blank
89
90       CatInt <= CatData when Data(DispCount+16) = '0' else CatSign;
91
92       process(Reset, Clk)
93       begin
94         if Reset = '1' then Cat <= "00000000"; An <= "0000";
95         elsif Clk'event and Clk = '1' then
96           Cat <= CatInt;
97           An <= AnInt;
98         end if;
99       end process;
100
101   end SevenSeg_arch;
102
```