C-programmering

Ingenjörsjobb för slutbetyg C-kurs

Lasse Karagiannis

2016-12-18

Sammanfattning Ingenjörsjobb C-programmering för TEIS

Ett C-program som använder sig av objektbaserat koncept har utvecklats för en mätstation baserad på BeMicro 10 kortet från Arrow, som är bestyckad med sensorerna som använts. Temperatur, ljusstyrka och acceleration har uppmätts och presenterats på en VGA-skärm där VGA-interfacet utgjorts av ett tillhandahållet IP-block. Sensorer på kortet som använts är den enkla termistorn på framsidan av kortet, accelerometern X-axel sensor, samt ljusmätaren i form av en fotoresistor.

Innehållsförteckning

1 Kravspecifikationen

Kravspecifikationen redovisas i nedanstående Tabell 1

Tabell 1: Kravspecifikation från kund

Krav	Beskrivning	Utfört Ja/nej
Förstudie		
Krav_1	Beskriv kort och ge förbättringskommentarer på ett äldre projekt ur low level C-synvinkel, se ”archive older projects”. (Eget kapitel)	JA
Krav_2a	Beskriv BeMicro Max 10 kortet, som ska användas till projektet. (Eget kapitel)	JA
Krav_2b	Beskriv mikrodatorarkitekturen, alla I/O komponenterna, minneskartan m.m. (Eget kapitel)	JA
Krav_2c	Beskriv hårdvaran med A/D omvandlingen, drivrutinerna, temperatur och accelerometern komponenten. (Eget kapitel). I bilagan ska databladen för accelerometern och temperaturgivaren	JA
Krav_2d	Beskriv hur en accelerometer och ljusmätare är uppbyggd och fungerar.	JA
Krav_2e	Beskriv VGA-hårdvaran	JA
Krav_3	*I de fall konstruktören väljer att genomföra ett eget projekt ska en kravspecifikation för detta projekt skrivas och godkännas av chefen och övriga krav diskuteras.*	JA
Krav_4a	Projektet specifikationen ska minst innehålla ljusmätaren och accelerometern, knappar, lysdioder och VGA. Alla tryckknapparna behöver inte användas. Temperaturmätning är frivillig.	JA
Krav_4b	VGA skärmen ska först presentera systemet med en sida på skärmen. Därefter ska en ny sida visas som hanterar loggningen. Konstruktören utformar sidan.	JA
Krav_4c	VGA-skärmen ska kunna visa minst följande samplingsfrekvenser; 5 Hz, 1Hz och 0,2Hz. En hel skärm ska kunna visas i realtid (d.v.s. den uppdateras hela tiden med nya värden). Konstruktörens namn ska skrivas i undre högra hörnet.	JA
Funktionskrav		
Krav_5	Regler och riktlinjer för C ska följas.	JA
Krav_6	**Följande VGA drivrutiner är beställda:** En kort SW-manual för VGA-drivrutinerna (exempel se Sierra SW-manual). Även exekveringstiden ska vara definierad för varje funktion. Koden ska vara robust skriven. Kunden har också begärt att alla typerna i anropen ska vara exakt definierade, som t.ex. ”alt_u16” (inga ”int”). **Funktion :** `print_pix(unsigned int x,unsigned int y,unsigned int rgb);` **Funktionsbeskrivning:** Skriver en pixel med färgen *rgb* på koordinaten (x, y). **Funktion:** `print_hline(unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);` **Funktionsbeskrivning:** Skriver en horisontell linje med färgen *rgb* och med längden *len* vilken startar på koordinaten	JA

	(x_start, y_start). **Funktion:** `print_vline(unsigned int x_start, unsigned int y_start, unsigned int len, unsigned int RGB);` **Funktionsbeskrivning:** Skriver en vertikal linje med färgen *rgb* och med längden *len* vilken startar på koordinaten (x_start, y_start). **Funktion:** `print_char(unsigned int x,unsigned int y,unsigned int rgb,unsigned int BG_RGB,char Character);` **Funktionsbeskrivning:** Skriver tecknet "Character" med färgen *rgb* och med bakgrundsfärgen "BG_RGB" på koordinaten (x, y). Implementera minst de tecken som behövs för att skriva den texten som ska visas på skärmen. **Funktion:** `read_pixel_ram_int(unsigned int x_start, unsigned int y_start); return: unsigned pixel_data` (obs! enbart de sista tre bitarna är intressanta). **Funktionsbeskrivning:** Läser pixel_data från pixel RAM (3 bitar, RGB) från adress x och y (beräknat från x och y). **Funktion:** `print_circle(unsigned int radie, unsigned int x_centrum, unsigned int y_centrum, unsigned int rgb);` **Funktionsbeskrivning:** Skriver en cirkel med radien "radie" och färgen" rgb" på mittkoordinaten (x, y). Cirkeln ska fyllas med samma färg. **Funktion**: `clear_screen(int rgb);` **Funktionsbeskrivning**: Denna funktion rensar skärmen genom att RGB-värdet skrivs till alla pixlar på skärmen. **Funktion:** `print_str(alt_u16 x_start, alt_u16 y_start,alt_u8 rgb,char *str)` **Funktionsbeskrivning:** Print_str skriver ut en sträng på skärmen bokstav för bokstav med hjälp av print_char.	
Rapportkrav		

Krav_7	Utgå gärna från Rapportmall_TEIS, som ligger på Itslearning. Kunden kräver en standardiserad teknisk rapport med filnamn "förnamn_efternamn_C_ingenjorsjobb". Följande ska finnas med i slutrapporten:
	• Kravspecifikation, eget kapitel (krav)
	• Beskrivning av ett äldre projekt med några kommentarer om förbättringar för att förstå konstruktionen, eget kapitel (krav)
	• Projektplan, aktiviteter, budgeterad och verklig kostnad (500 SEK/tim), eget kapitel (krav)
	• Testprotokoll för verifiering/validering, grundat på kravspecifikationen. Protokollet behöver inte vara komplett, det räcker att visa hur det ska utföras och det behöver bara visas på systemnivå (inte på subnivåer), eget kapitel. (krav)
	• Konstruktionsbeskrivning, eget kapitel
	  ○ System arkitektur ("top-down")
	  ○ Delsystem
	  ○ Eventuella subsystem (delsystem av
		JA

	delsystem), o.s.v.
	• Verktygs- ”settings”, eget kapitel
	  o Dokumentera alla settings
	  o Eventuella länk direktiv
	• Verifiering, eget kapitel
	  o Fyll i testprotokollet och ev buggar antecknas. Konstruktionen behöver inte vara bugg fri.
	• Analys, eget kapitel
	  o Felmeddelanden ”Warnings” från kompileringen
	  o Minimera ”foot print”, koden
	• Validering, eget kapitel
	  o Visa med resultat från testprotokollet (validering)
	  o Kombineras med en kort film som beskriver för en utomstående vad som genomförts (visa gärna en systemarkitektur) och demonstrera att det fungerar (en länk till YouTube), som en kort presentation av hela jobbet.
	• Sammanfattning, kända buggar och framtida förbättringar
	• Referenser	

Leveranskrav		

Krav_9	Leveransen ska innehålla tre foldrar:
	• **<u>Konstruktionsbeskrivning</u>**
	  o Konstruktionsrapport
	  Filnamn:
	  förnamn_efternamn_C_ingenjorsjobb (pdf, word..)
	• **<u>Konstruktionsdokument</u>**
	  o software mappen med applikation och BSP
	  o Alla IP-komponenter (IP mapp)
	  o SOF- och SOPCINFO-filen
	• **<u>Diverse</u>**	

Krav_10	Leveransen ska ske till plattformen Itslearning. Namnet på filen ska vara ”förnamn_efternamn_C_ingenjorsjobb_x.zip”. Där x motsvarar version. Sista leveransdag se kurs schema (för VG).	

2 Kommentar äldre C-projekt

De redovisade projekten har alla gemensamt att de inte redovisat om och hur de använt idéer avseende objektorienterad analys och objektorienterad programmering. De två killarna Magnus Mårtensson och Magnus Flysjö, och som väldigt förtjänstfullt programmerat och dokumenterat spelet Black-Jack beskriver i sin videofilm att de använder *struct* för att gruppera sammanhörande information, men redovisar t.ex. inte vad som de anser vara sammanhörande.
Man får dock en vinkning avseende detta när man tittar på hur de grupperat innehållet i listningarna av innehållen i de header-filerna de skapat.
Objektorienterad programmering innebär dock mer än att gruppera variabler i struct och att lägga hjälpfunktioner i separata filer som inkluderas istället för att att ha allt i en samma fil som main-funktionen. Objekt orienterad programmering innebär att man även grupperar funktioner som objektet använder i den *struct* där relevanta variabler finns listade, där mycket stor tonvikt läggs på design av typer. Objektets typdefinition är den s.k. klass beskrivningen, som är lejonparten i objekt orienterad design.
Genomläsning av rapporterna ger slutsatsen att ingen laborant verkar ha använt det objektorienterade konceptet.

Mårtensson och Flysjö har mycket tydligt, mycket förtjänstfullt, redovisat den *state machine* som programmet använder genom att den grafiska redovisningen samt att man vid varje övergång skrivit övergångsvillkoret. Rapporten skvallrar om stor omsorg om detaljer.

Patric Sjöberg redovisar i sin rapport en C-kod, med en stor nästlad if-sats som ger väldigt lite information, och kunde ha utelämnats. Det hade dock varit intressant läsning om han hjälpt läsaren med en grafisk beskrivning av koden såsom författarna till Black-Jack gjort.

Jonas Torstensson har däremot listat C-koden för main-funktionen, för spelet PONG, i sin rapport. Hans main-funktion innehåller bara några få anrop, och detta är mycket elegant, och enligt denna rapports författare det sättet ett huvudprogram ska vara uppbyggt. Torstensson har även bifogat ett flödes schema.

3 Projektplan

Projektplanen har varit mycket enkel, och följt ”rapid prototyping” konceptet, dvs. att utan tanke på optimering försöka få till en fungerande prototyp så snabbt som möjligt. Följande naturliga steg följdes

1. Förstå kravspecifikationen
2. Testa och förstå exempelkoden
3. Bestäm användargränsnittet
4. Formulera testfall och skissa på ett testprotokoll
5. Bestäm mjukvaruarkitektur
6. Koda prototypen, validera och debugga. Versionshantera på Github.
7. Skriv rapporten
8. Leverera

Punkterna 1 till 6 klarades under en helg, ca 30 timmar.
Rapportskrivningen har i skrivande stund då 95% av rapporten skrivits ca 20 timmar. Under rapportskrivningen uppmärksammas detaljer i kravspecen som jag missat att implementera, såsom exempelvis att kravet är 3 olika samplingsperioder.

4 Testprotokoll

Test av varje enskild sensor för sig med plottning av graf som uppdateras, däremot har problem uppstått när samtliga grafer har skulle presenteras samtidigt, dvs. Accelerometer X, Accelerometer Y, Accelerometer Z, Temperatur och Ljusmätare. Här har programmet bettet sig mycket konstigt. Det har uppvisat många olika sjuksymtom. Grafer har skrivits ut dubbelt, programmet har frusit. Debuggsession har visat att pekare har värden 0xdeadbeef.

Ändringar i koden gjordes där funktionstyper med void-pekare ändrats till pekare av typen pekare till SENSOR_OBJECT.
Vidare har typen för köerna ändrats till att vara *static*. De här ändringarna har stabiliserat koden, men inte gjort den helt felfri.
Problemet att *x_origin* nollställdes, och efter ett tag även *y_origin* felsöktes. Under felsökning noterades av varibeln char description[80] nollställdes, och det verkade som att för varv i main-funktionen while-loop som gjordes, nollställdes en minnesposition tillhörandes variabeln SENSOR_OBJECT för accelerometer X, där nollställningen av *x_origin* nåddes fram under varv 49 av exekveringen av while-loopen i main-funktionen.
För att snabbare komma fram till momentet där nollställningen av x_origin inträffade, så minskades medlemsvariabbeln *const char description[80]* till *const char description[18]*, vilket gjorde att felsymptomen upphörde, men någon logisk förklaring till detta har inte kunnat nås.
En slarvig analys gjorde att jag spred den felaktiga informationen att felet spridit sig till att utskriften av tiden som görs av funktionen som väntar 0.2 sekunder, men detta berodde på att variabeln sub_seconds som räknar upp till 5 och uppdaterar simuleringstidsräknaren, inte var deklarerad som static.
Koden har en eller flera latenta fel, som kommer att ge sig till känna om *const char description[18]* ändras till att kunna hantera längre beskrivande ord.

Tabell 1 Testprotokoll

Krav	Testfall	Beskrivning
4b	1	”Splashscreen”då programmet startar
4b	2	Tangentryckning avslutar splash screen till grafpresentation
4b	3	Koordinatsystem presenteras utan sensordata
4b	4	Sensor presentationsgrafer fungerar var för sig
4b	5	Graferna fungerar tillsammans
4b	6	Sensordata ändras då Accelerometer skakas
4b	7	Sensordata ändras då Termistor värms
4b	8	Sensordata ändras då Fotoresistor täcks
4c	9	Knapptryckning byter samplingsperiod
	10	Klocka som visar förfluten tid för att kunna

5 Konstruktionsbeskrivning

Systemet består av FPGA-kortet BeMicro10 som innehåller de sensorer som använts. FPGA:n är programmerad med processorn Nios II, samt ett av TEIS tillhandahållet VGA-interface i mjukvara och hårdvara.
Mjukvaran som Nios II processorn exekverar åstadkommer läsning av sensorer och skrivning till VGA-interfacet som hanterar visning på VGA-skärm.
Mjukvarudelen av systemet har designats under beaktande av faktorer såsom kodåtervinning och komplexitets reducering i åtanke.
För att reducera komplexiteten och göra problemet hanterbart valdes objekorienterad C.

På samma sätt som man i C++ och Java kan definiera typer som är s.k. abstrakta klasser, dvs, typer man inte kan skapa objekt av, så kan man följaktligen göra analogt, även i C, där typen SENSOR_CLASS skapades med ett typalias SENSOR_OBJECT.
Om man följer strukturen i nedanstående struct-deklaration kan variabler inte skapas av typen SENSOR_CLASS, däremot av SENSOR_OBJECT.
SENSOR_CLASS blir här att likna vid en abstrakt klass.

För att kunna ha en vektor med olika objekt så inför man i objektorientering konceptet abstrakt klass som man inte kan skapa objekt av, men som andra klasser/typer måste ”implementera”. Då går det att skapa en array av den abstrakta klasstypen, och ha i denna objekt som implementerar den abstrakta klassen. All information om varje objekt och alla funktioner som det kan tänkas använda är knutet till objektet.

```
typedef struct SENSOR_CLASS SENSOR_OBJECT;


struct SENSOR_CLASS {
    char description[80];
    alt_u32 x_origo;
    alt_u32 y_origo;
    alt_u32 time_base;
    alt_u32 normalization_factor;
    alt_u32 offset;
    alt_u32 rgb;
    QUEUE queue;
    QUEUE *q;
    void (*configure_time_base)(alt_u32, void*);
    void (*init_measurement)(void*);
    void (*read_sensor)(QUEUE*);
    void (*update_graph)(void*);
    void *this;
    void (*draw_graph)(void*);
};
```

Typdefinitioner i C är röriga. Typdefinitionen gäller structen, inte variablerna inne i den, t.ex. så har vi inte deklarerat typen rgb som synonym till typen alt_u32, utan det är en namnet på den variabel som måste finnas i struct-typen på just den positionen.

Att man är tvingad att ha ett namn, är bara för att hjälpa programmeraren, kallar det därför ”hjälpnamn” för att separera konceptet från det som kallas variabelnamn och som deklareras i en en variabeldeklaration. Detta är ingen variabeldeklaration i den meningen.
Vidare så måste funktionspekare finnas, men till skillnad från vanliga variabeldeklarationer som börjar med typen och därefter variabelnamnet, så deklareras inte hjälpnamnet efter typen, utan den återfinns i själva uttrycket för funktionstypen.
På samma sätt som rgb är ett ”hjälpnamn” av typen alt_u32, så är init_measurement ett ”hjälpnamn” av typen ”pekare-till-funktion-som-tar-en-void-pekare-och-returnerar void”.

Anledningen till att man måste ha void pekare, är att kompilatorn inte accepterar pekare till typ-alias SENSOR_OBJECT. Det går att deklarera pekare av typstructen, (det är ju så man kodar noden till en länkad lista), dvs. SENSOR_CLASS i det här fallet, men författaren ville ha en pekare av till SENSOR_OBJECT av estetiska skäl. Kompilatorn ger error, därför att den håller inte koll på typ alias, när den håller på att läsa in structen. Void-pekare behövs därför att dessa får typkastas till valfri typ. Här används void-pekare därför att avsikten är att typkasta denna till SENSOR_OBJECT.

Läsaren undrar kanske varför man ska krångla till det på detta sättet, och införa typ-alias, med följden att void-pekare måste användas. Svaret är lika självklart som enkelt:
Därför att det är snyggt och det ser ut som C++ och Java!

De funktioner/metoder som ska vara unika för varje objekt, som måste finnas för att SENSOR_OBJECT ska implementera SENSOR_CLASS är följande funktioner.

```
void config_time_base(alt_u32, SENSOR_OBJECT*); void
init_measurement(SENSOR_OBJECT*);
void update_graph(SENSOR_OBJECT*);
void draw_graph(SENSOR_OBJECT*); void
read_accelerometerX(QUEUE *); void
read_accelerometerY(QUEUE *q); void
read_accelerometerZ(QUEUE *q); void
read_temp(QUEUE *q);
void read_light(QUEUE *q);
```

Nu kan objekten allokeras/deklareras och populeras med variabler och funktioner(funktionspekare) av samma typ som den ”abstrakta klassen”.

```
QUEUE q1, q2,q3,q4,q5;
QUEUE* q11 = &q1;
QUEUE* q22 = &q2;
QUEUE* q33 = &q3;

QUEUE* q44 = &q4;
QUEUE* q55 = &q5;



SENSOR_OBJECT accelorometerX =
      {     "Accelerom. x",
            30, 50,1,1,25,4, q1,q11,
            config_time_base,
            init_measurement,
            read_accelerometerX,
            update_graph,
            &accelorometerX,
```

```
                draw_graph
        };

    SENSOR_OBJECT accelorometerY =
        {       "Accelerom. y",
                320/3 +30, 50,1,1,20,4, q2,q22,
                config_time_base,
                init_measurement,
                read_accelerometerY,
                update_graph,
                &accelorometerY,
                draw_graph
        };

    SENSOR_OBJECT accelorometerZ =
        {       "Accelerom. z",
                2*320/3 +30, 50,1,1,60,4, q3,q33,
                config_time_base,
                init_measurement,
                read_accelerometerZ,
                update_graph,
                &accelorometerZ,
                draw_graph
        };




    SENSOR_OBJECT temp_sensor =
        {       "Temperature",
                30,180,1,100,10,4,q4,q44,
                config_time_base,
                init_measurement,
                read_temp,
                update_graph,
                &temp_sensor,
                draw_graph
        };

    SENSOR_OBJECT light_sensor =
        {       "Light",
                320/3 +30, 180,1,100,10,4, q5,q55,
                config_time_base, init_measurement,
                read_light,
                update_graph,
                &light_sensor,
                draw_graph
        };
```

Nu kan en array av den SENSOR_OBJECT deklareras

```
SENSOR_OBJECT sensors[5]={accelorometerX,
                          accelerometerY,
                          acceleoromete,
                          temp_sensor,
```

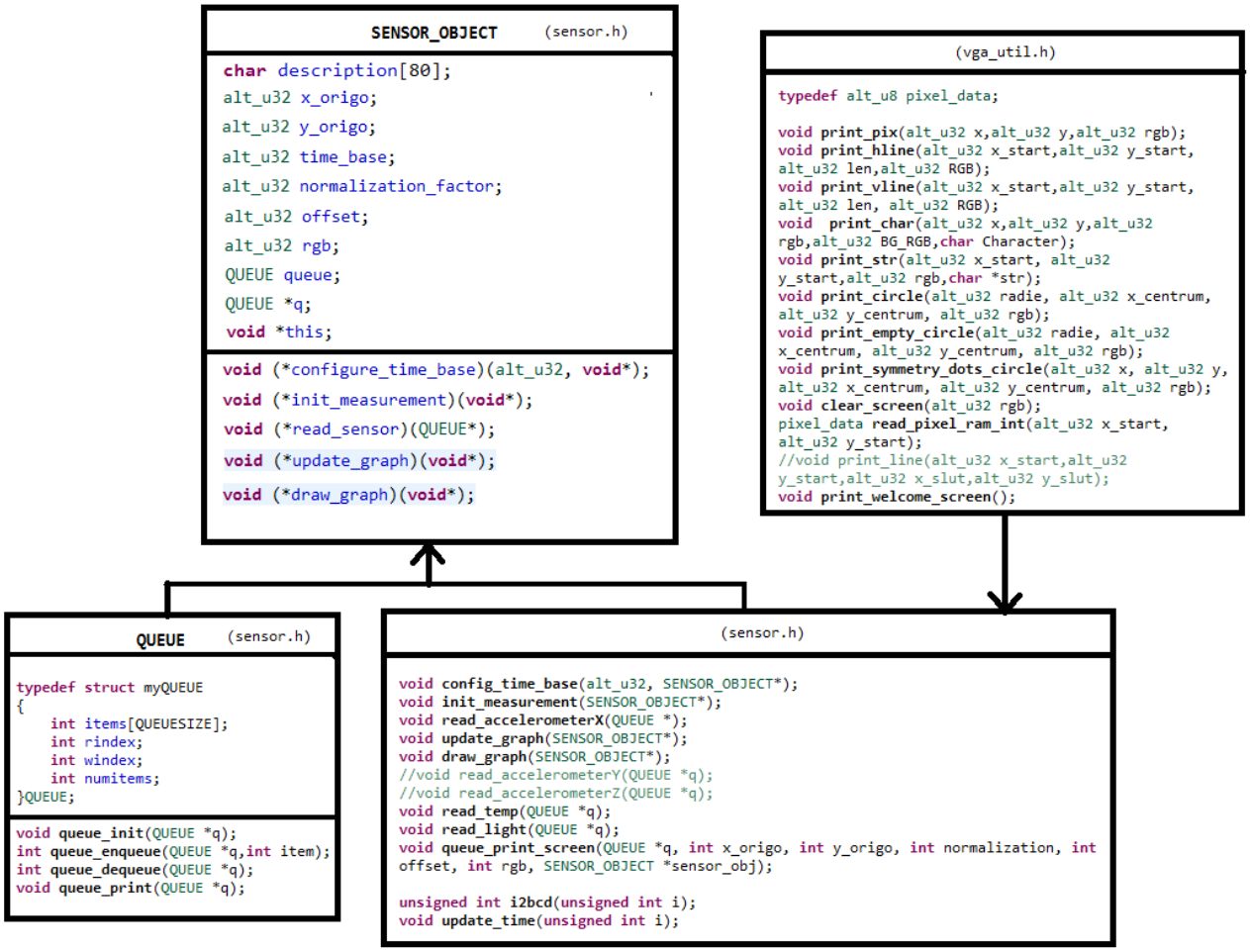
```
light_sensor
};
```

En for-loop kan nu ta hand om samtliga sensorer i tur och ordning på ett otroligt kompakt och lättförståeligt sätt

```
for(int i = 0; i<5; i++){
	sensors[i].configure_time_base(1,&sensors[i]);
	sensors[i].read_sensor(sensors[i].q);
	sensors[i].update_graph(&sensors[i]);
}
```

Funktionsnamnen som finns i klassbeskrivningen anropas, med de ”hjälpnamnen” från typbeskrivningen.

Figur 1 visar en graf över hur koden är uppbyggd. Sensorobjektet äger sin kö för att lagra mätvärden och har medlemsfunktioner för avläsning av sensordata samt grafpresentation. Medlemsfunktionerna utnyttjar hjälpfunktioner i vga_util.h, och sensor.h.

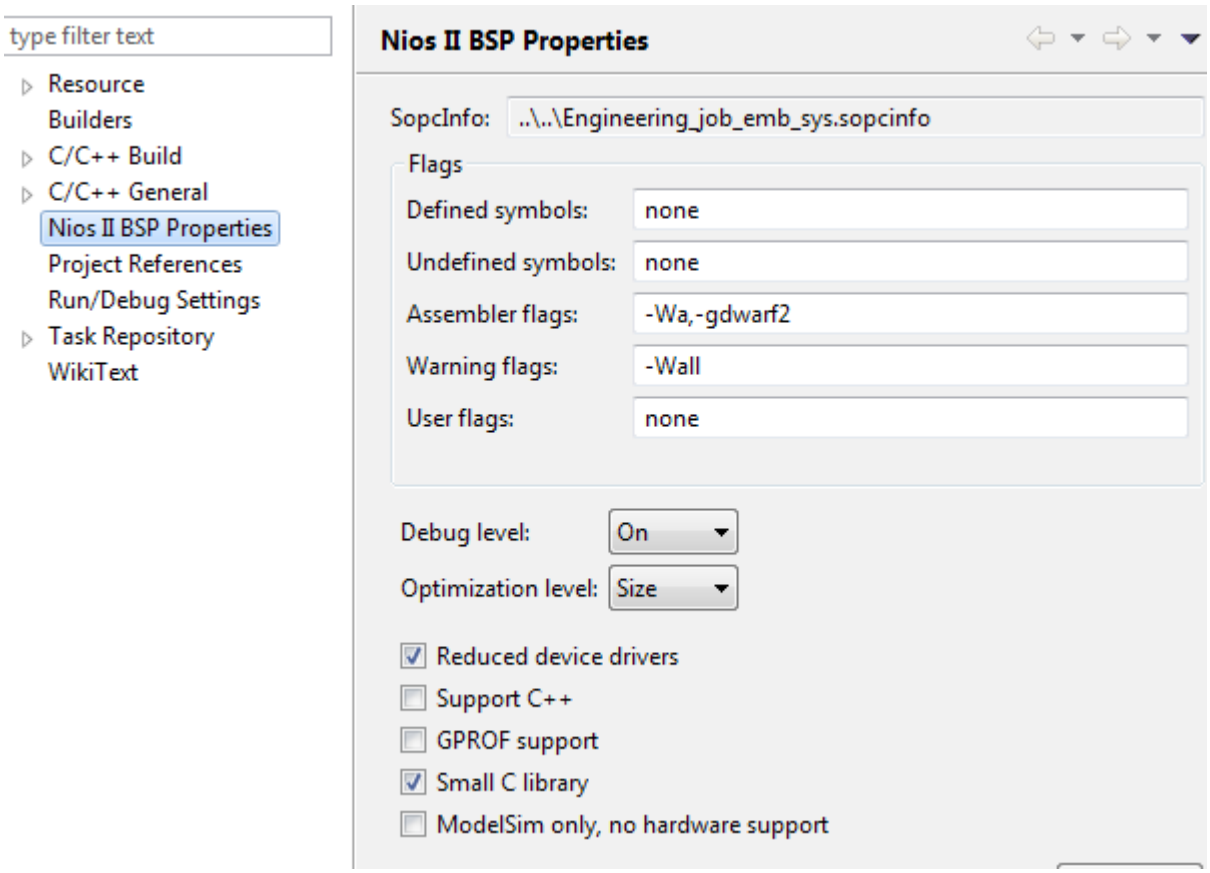


Figur 1 Grafisk presentation av kodstrukturen

Efter ovanstående text skrivits, så har felsökning lett fram till att void-pekarna i SENSOR_OBJECT har ändrats till pekare till SENSOR_OBJECT.

6 Verktygs -”settings”

Inga andra settings har använts förutom att optimera för storlek, reduced device drivers och small C-library, och debug ON, under utvecklingen. Se Figur 2.



Figur 2 BSP inställningar

Ytterliggare kompileringsflaggor har lagts på genom att editera makefilen.
Förutom -Wall, så har -Wpedantic och -Werror lagts på.

```
# Common arguments for ALT_CFLAGSs
APP_CFLAGS_DEFINED_SYMBOLS :=
APP_CFLAGS_UNDEFINED_SYMBOLS :=
APP_CFLAGS_OPTIMIZATION := -O0
APP_CFLAGS_DEBUG_LEVEL := -g
APP_CFLAGS_WARNINGS := -Wall -Wpedantic -Werror
APP_CFLAGS_USER_FLAGS :=

APP_ASFLAGS_USER :=
APP_LDFLAGS_USER :=
```

Storleken på elf-filen är 604213 bytes med debug-information och utan storleksoptimering.
Storleken på elf-filen är 568029 bytes utan debug-information och utan storleksoptimering.
Storleken på elf-filen är 564645 bytes utan debug-information och med storleksoptimering.

7 Verifiering

Programmet uppvisar inga fel vid körning.
Följande fel har dock uppstått då man kör koden, men dessa är åtgärdade

1. Programmet fungerar som det ska men del av en graferna visas återigen i något hörn.
2. Programmet fryser plötsligt, vid debug läge indikeras att en trap instruktion körts, utan att det finns någon kod för trap.
3. Programmet fryser plötsligt och vid debug indikeras att pekaren q har värdet 0xdeadbeef.
4. Programmet fryser då kösystemet har för stort samplingsdjup.
5. Programmet fungerar som de ska men efter att programmet körts i några minuter, så börjar det sist allokerade objektets graf att krångla och endast det senast samplade värdet ändras, dvs. kö systemet upphör att fungera för endast den grafen.

Problemen ovan har åtgärdas genom att definiera om pekartyper i structen SENSOR_OBJECT från pekare av typen pekare till void till pekare av typen pekare till SENSOR_OBJECT.
Vidare har köer deklarerats som static i main-funktionen. Man tycker att detta inte ska ha någon betydelse att deklarera en variabel som static i en main-funktion, men praktisk erfarenhet visar dock att obegripliga feltillstånd kan hävas genom att göra så. Detta kan beror på fel i koden eller bero på en kombination av fel i koden, fel i GCC4 sviten samt fel i Eclipse. Tabell 2 visar testprotokollet med resultat.

Tabell 2 Testprotokoll

Krav	Testfall	Beskrivning	Validering
4b	1	”Splashscreen”då programmet startar	OK
4b	2	Tangentryckning avslutar splash screen till grafpresentation	OK
4b	3	Koordinatsystem presenteras utan sensordata	OK
4b	4	Sensor presentationsgrafer fungerar var för sig	OK
4b	5	Graferna fungerar tillsammans	OK
4b	6	Sensordata ändras då Accelerometer skakas	OK
4b	7	Sensordata ändras då Termistor värms	OK
4b	8	Sensordata ändras då Fotoresistor täcks	OK
4c	9	Knapptryckning byter samplingsperiod	OK
	10	Klocka som visar förfluten tid för att kunna	OK

8 Analys

ELF-filens storlek är ca en halv megabyte. Dess storlek beror på hur mycket ytterliggare information verktyget genererar till målsystemet.

Storleken på elf-filen är 604213 bytes med debug-information, utan storleksoptimering.

Storleken på elf-filen är 568029 bytes utan debug-information, utan storleksoptimering.

Storleken på elf-filen är 564645 bytes utan debug-information och med storleksoptimering.

Aktivering av enable_lightweight_device_driver_api, deaktivering av enable_c_plus_plus, enable_clean_exit och enable_exit har gjorts.

Inga kompileringsvarningar har erhållits även efter att kompilatorn instruerats att göra en striktare kodanalys, genom att förse den med flaggorna -Wpedantic och -Werror.

9 Validering

Enligt Wikipedia så betyder validering:

Validering (av engelskans *validation*) är när man officiellt godkänner ett dokument och bekräftar att det som står i dokumentet är korrekt. Ofta består dokumentet av ett förslag och valideringen går då ut på att se till att förslaget verkligen stämmer med förslagsställarens önskemål.
Allra vanligast är att dokumentet innehåller krav på en produkt. Valideringen är då en del av kravhanteringen och syftet med valideringen är att se till att man verkligen utvecklar den produkt som intressenterna vill ha.

När utvecklingen sedan är klar verifieras den färdiga produkten mot de validerade kraven. Ett inte helt ovanligt missförstånd är att validering är en variant av verifiering, men det är alltså helt fel. Det korrekta är att man validerar kraven på den tänkta produkten mot kravställarens önskemål, men att man verifierar egenskaperna hos den färdiga produkten mot kraven.

Validering har åstadkommits genom frekventa förevisningar av prototypen för beställaren, som inte invänt en enda gång avseende prototypens funktion och dess gränssnitt emot användaren.
Slutsatsen som rapportförfattaren tagit att det testprotokoll som formulerats implementerar de facto beställarens krav.
För att se en tidig version av programmet, sök efter youtube-kanalen ”Lasse Karagiannis” och spellistan ”Program craches”. Release-versionen finns inte på youtube.

Den version av koden som körs i youtube-klippet har sedan dess uppdaterats.

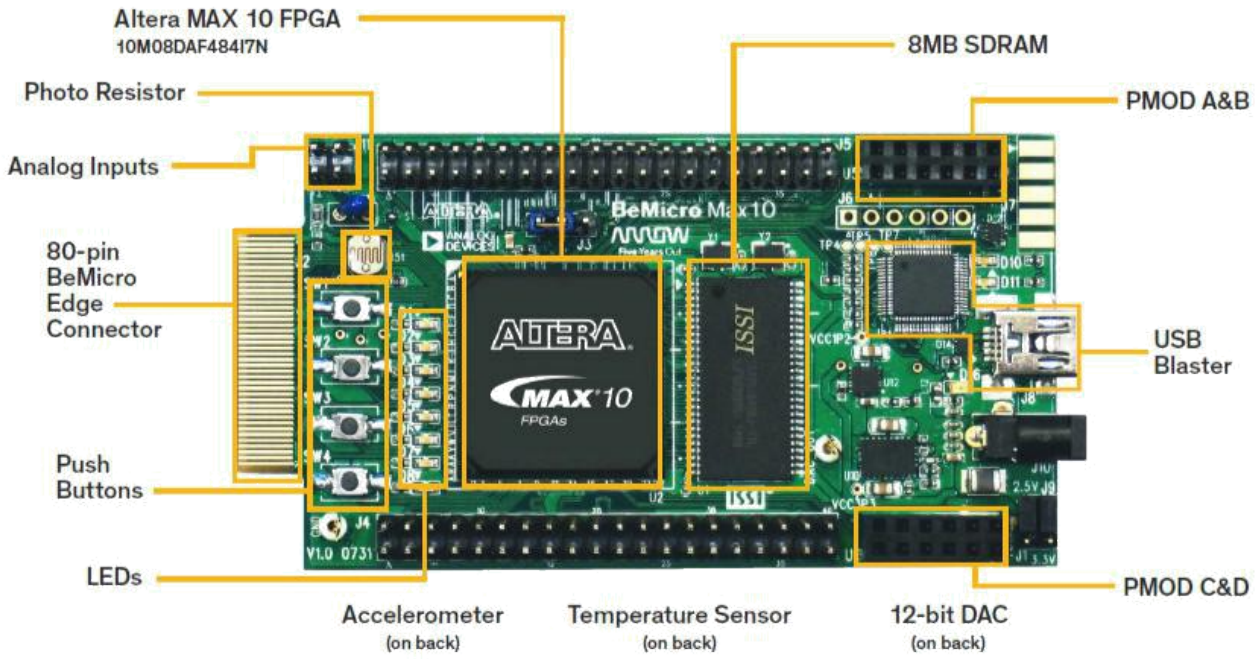
Testprotokollet Tabell 2 nedan visar att valideringen har varit framgångsrik.

Tabell 2 Testprotokoll

Krav	Testfall	Beskrivning	Validering
4b	1	”Splashscreen”då programmet startar	OK
4b	2	Tangentryckning avslutar splash screen till grafpresentation	OK
4b	3	Koordinatsystem presenteras utan sensordata	OK
4b	4	Sensor presentationsgrafer fungerar var för sig	OK
4b	5	Graferna fungerar tillsammans	OK
4b	6	Sensordata ändras då Accelerometer skakas	OK
4b	7	Sensordata ändras då Termistor värms	OK
4b	8	Sensordata ändras då Fotoresistor täcks	OK
4c	9	Knapptryckning byter samplingsperiod	OK
	10	Klocka som visar förfluten tid för att kunna	OK

10 BeMicro Max 10 kortet

Kortet som använts under ingenjörsjobbet är BeMicro Max 10 från Arrow Electronics. Projektet använder fotoresisistorn, accelerometern, temperatursensorn samt expansionsporten PMOD A&B som VGA-port.



Figur 1 BeMicro Max 10 kortet

Kortet har [1]

två expansionsportar av typen PMOD.

två stycken 40 bitars expansionsportar av instickstyp.J4 och J5.

En 80 bitars expansionsport ”BeMicro Edge Connector”

en 3-axlad MEMS Accelerometer från Analog Devices (ADXL362)

en12-bitars DAC från Analog Devices (AD5681R)

en digital temperatursensor från Analog Devices (ADT7420) 8 MB SDRAM

en MAX 10 FPGA (10M08DAF484xxx)

- 8,000 LEs
- 414 Kbit (Kb) M9K memory
  - 256 Kbit (Kb) user flash memory 2
  - phase locked loops (PLLs)
- 24 18x18-bit multipliers
  - 1 ADC block – 1 MSa/sec, 12-bit, 18-channels 17 analog inputs
- 1 temperature sense diode
- 250 general purpose input/output (GPIO)
- Non-volatile self-configuration with dual-boot support

På kortet går att utläsa portbeteckningar med kodade enligt Jx, eller Ux, där x är ett nummer.

J1: Jumper för systemspänningar
J2 : 80 bitars kant-konnektor

J3: Jumper för *boot select*

J4, J5: 40-pinnars digital port

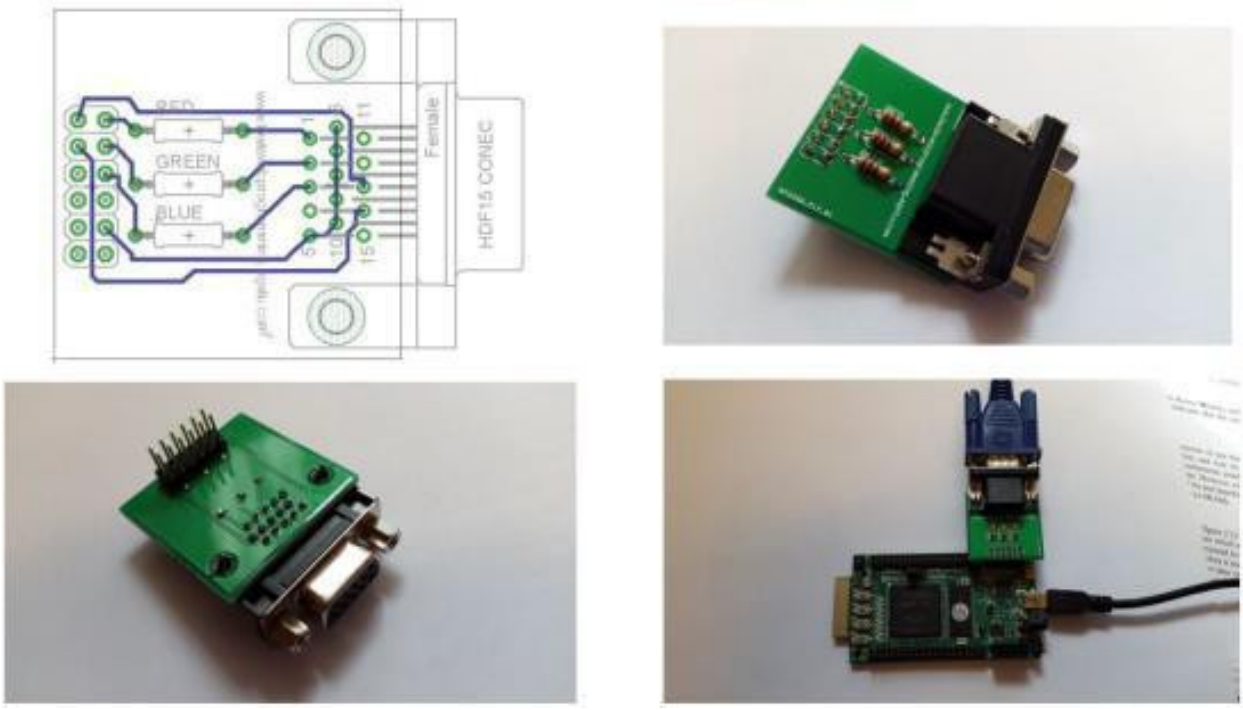
J6: Olödd port

J7: kantkontakt för matningsspäninga

J8: USB-kontakt

J9: Jumper för systemspänningar
J10: hylskontakt för matningssänning
J11: två analoga ingångar AIN1 och AIN2
U5: PMOD A&B
U6: PMOD C&D

Altera tillhandhåller intressant tutorials avseende detta kort med ett antal inbyggnadstillämpningar, såsom DC-DC omvandlare, PWM-styrning och Stegmotorstyrning.[2]



VGA-porten är ett TEIS-utvecklat kort som spänningsanpassar BeMicro signalerna till en VGA-skärms signalspecifikationer. Se bild nedan.

Figur 2 VGA expansionskort

[1] http://www.alterawiki.com/wiki/BeMicro_Max_10

[2] https://cloud.altera.com/devstore/platform/?family=max-10&board=4

11 Mikrodatorarkitekturen

Mikrodatorn är implementerad i en FPGA från Altera av typ Max 10, och använder ett antal fria IP-block såsom Nios II processorn och ett antal periferienheter såsom SDRAM, SDRAM kontroller och PIO. Dessa är konfigurerbara, och definieras i en .sof-fil. Hur konfigureringen görs finns beskrivet i tutorial tillhandahållen av Altera [3].

Sopcinfo-filen är det interface utvecklingsmiljön behöver visavi, Sof-filens konfigurering av FPGA:n.
Sopcinfo-filen innheåller information om målsystemet för utvecklingsmiljön, men innehåller inte den kompletta definitionen av målsystemet.
Sopcinfo-filen är möjlig att inspektera genom att besöka ”Properties” under BSP-mappen i Eclipse-baserade utvecklingsmiljön för Nios II. ”Nios II BSP Properties” och vidare till ”BSP editor”. I BSP-editor kan mindre omkonfigureringar (slå av/slå på befintliga moduler) göras i den befintliga mikrodatorarkitekturen som sopcinfo-filen utgör ett interface för.

Minnesmappen Figur 3, som finns att inspektera i BSP-editorn, beskriver systemarkitekturen genom att de ingående delarna av mikrodatorsystemet listas, samt var dessa placerats i minnesrymden.

Slave Descriptor	Address Range	Size	Attributes
jtag_uart_0	0x01081278 - 0x0108127F	8	printable
modular_adc_0_sequencer_csr	0x01081270 - 0x01081277	8	
TIMER_HW_IP_0	0x01081260 - 0x0108126F	16	
sys_pll	0x01081250 - 0x0108125F	16	
adc_pll	0x01081240 - 0x0108124F	16	
Led	0x01081230 - 0x0108123F	16	
Key_input	0x01081220 - 0x0108122F	16	
accelerometer_spi	0x01081200 - 0x0108121F	32	
modular_adc_0_sample_store_csr	0x01081000 - 0x010811FF	512	
BeMicro_VGA_IP_0	0x01000000 - 0x0107FFFF	524288	
SDRAM_Controller	0x00800000 - 0x00FFFFFF	8388608	memory

Figur 3 ***Memory map***

Timer_HW_IP _0 och BeMicro_VGA_IP_0 är periferienheter/hårdvaru IP-block, som tillhandahållits av TEIS.

[3] http://www.alterawiki.com/uploads/5/5e/BeMicroM10_Embedded_System_Lab.pdf

12 Beskrivning av några använda periferienheter

12.1 AD-omvandlaren

Fullständig dokumentation över AD-omvandlaren finns i dokumentet ”Max 10 Analog to digital converter user guide” [4]

Max 10 innehåller antingen en eller två fristående AD-omvandlare beroende på vilken modell av Max 10 man använder där respektive IP-block går under namnen ”Altera Modular ADC core” respektive ”Altera Modular Dual ADC core”, och är av typen SAR (Successiva approximationer) AD-omvandlaren har enligt databladet[5] för MAX10 FPGA:n en samplingsfrekvens på 1 MHz.

AD-omvandlaren har upplösningen 12 bitar och har 18 kanaler, enligt dokument på altera_wiki dvs. den kan AD-omvandla 18 analoga ingångar med samplingsfrekvensen 1 MHz.
16 av kanalerna är kopplade bidirektionella PIO-ingångar, vars digitala funktion försätts i tri-state då de används som analoga ingångar. Om det inte konfigureras som analoga ingångar så fungerar de som vanliga digitala ingångar.
En kanal är kopplad till en dedikerad analog pinne ANAIN1 och en kanal TSD är kopplad till en temperaturkänslig diod TSD, för monitorering av chip-temperaturen. Denna TSD är finns internt på FPGA-chippet, och är inte kopplad till någon pinne på chippet. Notera att denna inte är upptagen i tabellen över kanal pin mappningen ADC användarmanualen, se Figur 4.

Table 5-4: Altera Modular ADC IP Core Channel to Pin Mapping for Single ADC Devices

Channel Name	Pin Name
CH0	ANAIN1
CH1	ADC1IN1
CH2	ADC1IN2
CH3	ADC1IN3
CH4	ADC1IN4
CH5	ADC1IN5
CH6	ADC1IN6
CH7	ADC1IN7
CH8	ADC1IN8
CH9	ADC1IN9
CH10	ADC1IN10
CH11	ADC1IN11
CH12	ADC1IN12
CH13	ADC1IN13
CH14	ADC1IN14
CH15	ADC1IN15
CH16	ADC1IN16

Figur 4 AD-omvandlarens mappningar

AD-omvandlaren har ett försteg av ett lågpassfilter för att undertrycka frekvenser som annars speglas in i ett lägre frekvensintervall.

Enligt [4] sidan 1-3, så har Max 10 av modell 10M08DAF484xxx dock endast ett ADC-block med 16 kanaler, inte 18 stycken såsom angivet på altera_wiki.

Följande mappning finns mellan AD-omvandlaren och resurser på på BeMicro Max 10 [6]

8.4.1 Analog Input Header

One of the key features of the MAX 10 FPGA is the analog block. Connector J11 can be used to attach input signals to the MAX 10 FPGA device. The signals coming from the connector pass through a signal conditioning RC circuit and then are fed into the analog input pins of the MAX 10 FPGA device.

Signal Name	MAX 10 Pin	MAX 10 ADC channel number	Board Connection Point
AIN[0]	F5	Ch. 1	2x2 header J11, pin 1
AIN[1]	E4	Ch. 9	2x2 header J11, pin 2
AIN[2]	F4	Ch. 2	testpoint TP12 (silkscreen "AIN3" on back)
AIN[3]	E3	Ch. 16	testpoint TP13 (silkscreen "AIN4" on back)
AIN[4]	J8	Ch. 3	testpoint TP15 (silkscreen "AIN5" on back)
AIN[5]	G4	Ch. 11	testpoint TP14 (silkscreen "AIN6" on back)

8.4.2 Photo Resistor

The BeMicro Max 10 board contains a photo resistor attached to one of the analog input pins of the MAX 10 FPGA.

Signal Name	MAX 10 Pin	MAX 10 ADC channel number
AIN[6]	J9	Ch. 4

8.4.3 Thermistor (Thermal Resistor)

The BeMicro Max 10 board also contains a thermistor attached to one of the analog input pins of the MAX 10 FPGA.

Signal Name	MAX 10 Pin	MAX 10 ADC channel number
AIN[7]	F3	Ch. 12

8.4.4 Other MAX 10 ADC Inputs

The remaining MAX 10 ADC inputs are connected to various power rails on the BeMicro Max 10 kit.

Signal Name	MAX 10 Pin	MAX 10 ADC channel number	Board Connection Point
VCC3P3	J4	Ch. 5	3.3V power rail
VCC2P5	H4	Ch. 13	2.5V power rail
VCC1P8	H3	Ch. 6	1.8V power rail
VCC1P2	G3	Ch. 14	1.2V power rail
GND	K5, K6, J3, K4	Ch. 7, 8, 10, 15	ground

Figur 5 Pin-mappning BeMicro Max 10 kortet till Max 10 FPGA pinnar

12.1.1. AD-omvandlarens drivrutiner

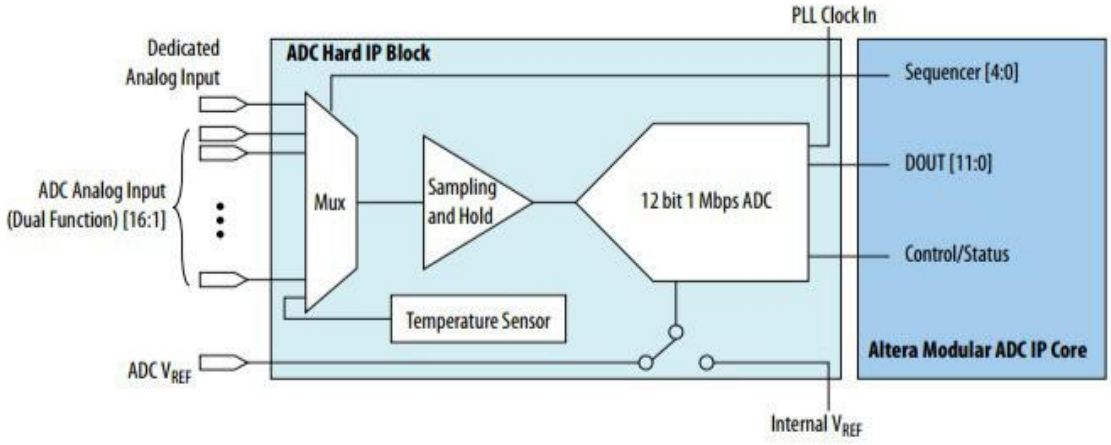
TEIS har drivrutiner för kommunikation med AD-omvandlaren bestående av makron som direkt skriver till register i AD-omvandlarens s.k. ”sequencer”, och använder inte alls de drivrutiner som BSP genererat i mappen ”drivers”. Mappen ”drivers” är en del av HAL men är inte en undermapp till HAL, vilket är ganska förvirrande.

Sequencerns uppgift är att multiplexa de 18 kanalerna in till ADC, se Figur 5. Sequencern har för

uppgiften upp till 64 s.k. slots, som allokeras och knyts till dedikerad kanal, vid konfigurationen av IP-blocket i Qsys. Om man bara är intresserad av 1 kanal t.ex kanal 0, kan man allokera slot 0 och knyta denna till kanal 0. Då kommer sequncern se till att endast kanal 0 multiplexas in till ADC.
Vill man instället att sequencerna ska multiplexa två kanaler in til ADC, t.ex. kanal 1 och kanal 2, men vill att kanal 2 ska AD -omvandlas 10 gånger oftare än än kanal 1, så kan man allokera 11 slots, och knyta slot 0 till kanal 1 och slottar 1 till 10 till kanal 2. När sequncern multiplexar in kanal 1 och kanal2 kommer det att ske efter detta slot-schema. Multiplexingen av kanalerna till ADC kommer då att ske cykliskt enligt följande cykel.
kanal 1, kanal 2, kanal2, kanal 2, kanal 2, kanal2, kanal 2, kanal 2, kanal 2, kanal 2, kanal2.
Samples Storage Core kommer att lagra detta i minnet i enlighet med definierat slot schema. På basadressen kommer i detta exempel att lagras resultatet från AD-omvandling av kanal , därefter kommer kanal 2 att lagras på basadressen +4*i, där i är slot-numret.

Figure 2-1: ADC Hard IP Block in MAX 10 Devices

Note: In dual ADC devices, the temperature sensor is available only in ADC1.



Figur 6 Sequncerns styrning av AD-omvandlarens kanal multiplexer

I ”drivers” finns följande headerfiler

```
altera_modular_adc_sample_store_regs.h
altera_modular_adc_sequencer_regs.h
altera_modular_adc.h
```

I `altera_modular_adc.h` finns HALs högnivå funktioner för konfigurering och läsning från ADC

```
void adc_stop(int sequencer_base);
void adc_start(int sequencer_base);
void adc_set_mode_run_once(int sequencer_base);
void adc_set_mode_run_continuously(int sequencer_base);
void adc_recalibrate(int sequencer_base);
void adc_interrupt_enable(int sample_store_base);
void adc_interrupt_disable(int sample_store_base);
void adc_clear_interrupt_status(int sample_store_base);
void adc_wait_for_interrupt(int sample_store_base);
int adc_interrupt_asserted(int sample_store_base);
alt_modular_adc_dev* altera_modular_adc_open (const char *name);
void alt_adc_register_callback(
```

```
      alt_modular_adc_dev  *dev,
      alt_adc_callback callback,
      void *context,
      alt_u32 sample_store_base);
int alt_adc_word_read (alt_u32 sample_store_base, alt_u32* source_ptr, alt_u32 len);

void altera_modular_adc_init(alt_modular_adc_dev* dev, alt_32 ic_id, alt_32
irq); /*
```

Dessa använder makron och definitioner definierade i
`altera_modular_adc_sample_store_regs.h`
`altera_modular_adc_sequencer_regs.h`
och `system.h`

TEIS använder inte dessa komplicerade funktioner, utan kringgår alltså dessa genom att skriva direkt till hårdvaran. Detta medger en snabbare exekvering, men gör tyvärr att mjukvaran måste förändras om den underliggande FPGA:n ändras.

TEIS använder sig av följande egendefinierade makron för Max 10:

```
#define ADC_INIT IOWR_32DIRECT(MODULAR_ADC_0_SEQUENCER_CSR_BASE,0,0x1) //
Start continuous sampling
#define ADC_READ_PHOTO IORD_32DIRECT(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, 0) // Read
ADC value
#define ADC_READ_TEMP IORD_32DIRECT(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, 4) // Read
ADC value
```

12.1.1.1 ADC_INIT

I ADC_INIT skrivs 0x01 på basadressen `MODULAR_ADC_0_SEQUENCER_CSR_BASE` +offset 0, vilken är definierad i system.h enligt
`#define MODULAR_ADC_0_SEQUENCER_CSR_BASE 0x1081270`

Basadressen är konfigurationsregistret, och vi kan läsa i `altera_modular_adc_sequencer_regs.h` att systemet tillhandahåller följande makro för skrivning till konfigurationsregistret

```
/*
 * IO Read Write helper Macros
 */
#define
  IORD_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base) \
  IORD(base, ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG)

#define IOWR_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base,
  data) \ IOWR(base, ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG,
  data)
```

Där finns också definitionen

```
#define ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG                    0
```

Således åstadkommer anropet ADC_INIT en skrivning av värdet 0x01 till adressen 0x108270.

Vill man använda de dedikerade makron som BSP genererat, som använder de ”helper Macros” ovan, för att starta kontinuerlig sampling så ska man istället använda följande makron listad i samma headerfil:

```
/*
 Set sequencer to run
continuously */
#define ALTERA_MODULAR_ADC_SEQUENCER_MODE_RUN_CONTINUOUSLY(base) \
   IOWR_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base, (             \
   IORD_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base)               \
    & ~(ALTERA_MODULAR_ADC_SEQUENCER_CMD_MODE_MSK) )             \
      | ALTERA_MODULAR_ADC_SEQUENCER_CMD_MODE_CONTINUOUS)


/*
 To Start the Sequencer Core
*/
#define ALTERA_MODULAR_ADC_SEQUENCER_START(base)      \
   IOWR_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base, (  \
   IORD_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base)    \
    & ~(ALTERA_MODULAR_ADC_SEQUENCER_CMD_RUN_MSK) )   \
      | ALTERA_MODULAR_ADC_SEQUENCER_CMD_RUN_START)
```

Anropet
`ALTERA_MODULAR_ADC_SEQUENCER_MODE_RUN_CONTINUOUSLY(MODULAR_ADC_0_SEQUENCER_CSR_BASE);`
gör följande
i. Skrivning till `MODULAR_ADC_0_SEQUENCER_CSR_BASE (0x1081270)` med makrot `IOWR_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG` som tar två argument *base* och *data*, som i sin tur inte är något annat än expansionen av makrot `IOWR(0x1081270, ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG, data)` dvs. `IOWR(0x1081270, 0, data)`

ii. Det som skrivs i styrregistret `data`, är det som redan finns där bitvis logiskt multiplicerat med inversen av `ALTERA_MODULAR_ADC_SEQUENCER_CMD_MODE_MSK (0x0000000E)` dvs.`0xFFFFFFF1`, vilket blir `0byyy...y0001`. Detta resultat görs sedan bitvis OR på med `ALTERA_MODULAR_ADC_SEQUENCER_CMD_MODE_CONTINUOUS (0x00000000)`, vilket resulterar i att deta data som skrivs till `0x1081270 är 0byyy...y0001`

Anropet
`ALTERA_MODULAR_ADC_SEQUENCER_START(MODULAR_ADC_0_SEQUENCER_CSR_BASE)` expanderas i sin tur till följande
`i. IOWR_ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG(base, data) som i sin tur expanderas till IOWR(MODULAR_ADC_0_SEQUENCER_CSR_BASE, ALTERA_MODULAR_ADC_SEQUENCER_CMD_REG, data)`

ii. Det som skrivs i styrregistret `data`, är det som redan finns där, logiskt bitvis multiplicerat med inversen av `ALTERA_MODULAR_ADC_SEQUENCER_CMD_RUN_MSK (0x00000001)`, dvs. `0xFFFFFFFE` vilket blir 0bxxx...xx0och därefter bitvis OR med `ALTERA_MODULAR_ADC_SEQUENCER_CMD_RUN_START (0x00000001)`. Resultatet blir 0bxxx..xx1

ADC_INIT åstadkommer samma resultat med den minst signifikanta nibbeln som anropen av de makron BSP tillhandahåller i avseendet dvs. först anropet

`ALTERA_MODULAR_ADC_SEQUENCER_MODE_RUN_CONTINUOUSLY(MODULAR_ADC_0_SEQUENCER_CSR_BASE);`
och därefter anropet
`ALTERA_MODULAR_ADC_SEQUENCER_START(MODULAR_ADC_0_SEQUENCER_CSR_BASE);`

Användarmanualen anger följande information angående konfigurationsregistrets bitfält

Altera Modular ADC Register Definitions

The registers in the generated Altera Modular ADC IP core provide the IP core with the control and settings during operation.

Sequencer Core Registers

Table 5-20: Command Register (CMD)

Address Offset: 0×0

Bit	Name	Attribute	Description	Value	Default
31:4	Reserved	Read	Reserved	—	0
3:1	Mode	Read-Write	Indicates the operation mode of the sequencer core. These bits are ignored when the run bit (bit 0) is set. In continuous conversion, the data will be overwritten after a complete sampling sequence.	• 7—Recalibrate the ADC • 6 to 2—Reserved • 1—Single cycle ADC conversion • 0—Continuous ADC conversion	0

Figur 7 Urklipp från ”Max 10 Analog to digital conversion users guide” avseede konfigurationsregistret

Vill man istället använda HAL API, så ska man istället använda rutinerna i
`altera_modular_adc.h`

```
void altera_modular_adc_init(alt_modular_adc_dev* dev, alt_32 ic_id, alt_32 irq);
```

följt av

```
void adc_set_mode_run_continuously(int sequencer_base);
```

och

```
void adc_start(int sequencer_base);
```

I `altera_modular_adc_init` registreras AD -omvandlaren i en lista, och om interupt ska genereras efter varje sampling, så registreras detta hos interrupt-kontrollern.
I `altera_modular_adc.c` kan man utläsa

```
void altera_modular_adc_init(alt_modular_adc_dev* dev, alt_32 ic_id, alt_32 irq)
{
    extern alt_llist altera_modular_adc_list; /*inte static således global
variabel definierad i annan header-fil*/
    alt_dev_llist_insert((alt_dev_llist*) dev, &altera_modular_adc_list);

      if ((0 <= ic_id) && (0 <= irq))
```

```
        {
            /* Install IRQ handler */
          #ifdef ALT_ENHANCED_INTERRUPT_API_PRESENT
               alt_ic_isr_register(ic_id, irq, alt_adc_irq,
                       dev, NULL);
            #else
               alt_irq_register(irq, dev, alt_adc_irq);
            #endif
        }
}
```

Vill man inte ha interrupt så ska irq-argumentet tydligen vara ett negativt tal. Argumentet ic_id skall förmodligen vara 0 eller 1 för Max10 därför att FPGA:n kommer med antingen en eller två ADC. I vårt fall ska ic_id förmodligen vara 0.

Vidare så måste man skicka in en pekare till den minnesarea som systemet avser att förvara information avseende ADC, och detta är en *struct* av typen `alt_modular_adc_dev`

vilken finns definierad i korresponderande headerfilen `altera_modular_adc.h`

```
/* Callback routine type definition */
typedef void (*alt_adc_callback)(void *context);

typedef struct alt_modular_adc_dev
{
    alt_dev              dev;
    /* Callback routine pointer */
    alt_adc_callback     callback;
    /* Callback context pointer */
    void                 *callback_context;
    /* Base address of the sample store micro core */
    alt_u32              sample_store_base;
   /* Base address of the sequencer micro core */
    alt_u32           sequencer_csr_base;
   /* Dual ADC mode enable status */
    alt_u8               DUAL_ADC_MODE;
} alt_modular_adc_dev;
```

Här blir det invecklat, därför att dev är en typsynonym till alt_dev_s enligt alt_dev.h
```
typedef struct alt_dev_s alt_dev;
```

där alt_dev_s är en struct med funktionspekare av filhanteringsfunktioner
```
struct alt_dev_s {
  alt_llist    llist;     /* for internal use */
  const char*  name;
  int (*open)  (alt_fd* fd, const char* name, int flags, int mode);
  int (*close) (alt_fd* fd);
  int (*read)  (alt_fd* fd, char* ptr, int len);
  int (*write) (alt_fd* fd, const char* ptr, int len);
  int (*lseek) (alt_fd* fd, int ptr, int dir);
  int (*fstat) (alt_fd* fd, struct stat* buf); int
  (*ioctl) (alt_fd* fd, int req, void* arg);
};
```

Detta behöver man inte bry sig om därför att API:et har en funktion som returnerar en pekare till en `alt_modular_adc_dev`

```
alt_modular_adc_dev* altera_modular_adc_open (const char *name)
```

```
{
    alt_modular_adc_dev *dev = NULL;

    /* Find requested device */
    dev = (alt_modular_adc_dev*) alt_find_dev (name, &altera_modular_adc_list);

    return dev;
}
```

Anropa helt enkelt **altera_modular_adc_open** med en konstant sträng som finns listad i system.h som sannolikt representerar ADC identiteten. Följande relevant strängar avseende ADC finns i system.h: ”CH0”, ”CH4”, ”CH12”,"/dev/modular_adc_0_sequencer_csr", "External VREF", "altera_modular_adc" och "/dev/modular_adc_0_sample_store_csr".
En kvalificerad gissning borde vara att den strängen som ska tillhandahållas funktionen antingen är "altera_modular_adc" eller "/dev/modular_adc_0_sequencer_csr", men kanske man också skall pröva "/dev/modular_adc_0_sample_store_csr.

Structen av typen alt_modular_adc_dev även en medlemsvariabeln *callback,* vilken är en funktion som man vill att CPU:n ska exekvera då ADC-kontrollern genererat ett interrupt. Har man inget interruptkrav så går det nog att att skicka in en pekare till en funktions som uppfyller typkraven, inte gör något alls, utan bara returnerar rätt typ.
Callback skall vara av typen alt_adc_callback, vilket är typsynonym för typen ”*pekare till funktion vars argument är en void-pekare och vars retur är av typen pekare till alt_adc_callback*”. Hur fixar man fram denna? Jo man definierar en funktion som tar som argument en pekare till void och returnerar denna pekare direkt. När man tilldelar stuktmedlemmen namnet på funktionen i fråga (namnet på funktionen = pekaren till funktionen) så typkastar man den till korrekt typ. Alla void pekare kan typkastas till vilken typ som helst så kompilatorn borde inte klaga alls.

Denna callback-rutin måste registreras hos interrupt kontrollern med funktionen **alt_adc_register_callback** som också fyller structen av typen alt_modular_adc_dev.

```
void alt_adc_register_callback(
      alt_modular_adc_dev  *dev,
      alt_adc_callback callback,
      void *context,
      alt_u32 sample_store_base)
{
    dev->callback         = callback;
    dev->callback_context = context; dev-
    >sample_store_base = sample_store_base;

      if(NULL != callback)
      {
               adc_clear_interrupt_status(sample_store_base);
            adc_interrupt_enable(sample_store_base);
      }
      else
      {
            adc_interrupt_disable(sample_store_base);
               adc_clear_interrupt_status(sample_store_base);
      }
```

```
    return ;
}
```

Vidare så är `callback_context` en pekare till information (context) som ADC tror att du är intresserad av. Allokera en void-pekare, och anropa funktionen **`alt_adc_register_callback`** med denna. Jag vet inte vad denna *context* är för något. Det skulle kunna vara information till sequencern om vilka slots som är definierade och till vilka kanaler dessa slots är knutna, men i såfall så skulle pekaren vara till en förutbestämd minnesadress, som innehåller information om mappningen. Vidare är det faktum att context ska vara en void-pekare något som skvallrar om att det finns någon rutin som kommer att använda denna och typkasta denna till något vettigare någonstans.

Dock är `sample_store_base` och `sequencer_csr_base` är värden som måste fås ifrån system.h. **`alt_adc_register_callback`** fyller structen med dessa värden, om de tillhandahålls.
Vi har från system.h

```
#define MODULAR_ADC_0_SEQUENCER_CSR_BASE 0x1081270
#define MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE 0x1081000
```

Om man inte har en Max 10 med två ADC, så kan man gissningsvis sätta `DUAL_ADC_MODE` till -1.

Efter denna förhållandevis svåra initaliseringen som det publika HAL API:et vill att man ska göra, så är de efterföljande anropen desto enklare att förstå.

Där
`void adc_set_mode_run_continuously(int sequencer_base)` som bara anropar makrot
`ALTERA_MODULAR_ADC_SEQUENCER_MODE_RUN_CONTINUO` som analyserats tidigare

```
                                                void
adc_set_mode_run_continuously(int sequencer_base)
{
    /* Note: Stop the ADC before calling this function, changing
       ADC mode while RUN bit is set has no effect. */
    ALTERA_MODULAR_ADC_SEQUENCER_MODE_RUN_CONTINUOUSLY(sequencer_base);
}
```

och **`adc_start`** som bara anropar makrot `ALTERA_MODULAR_ADC_SEQUENCER_START` som också analyserats tidigare

```
void adc_start(int sequencer_base)
{
    ALTERA_MODULAR_ADC_SEQUENCER_START(sequencer_base);
}
```

Sammanfattning sekvens av anrop för att använda HAL API för att initialisera ADC för att starta kontinuerlig sampling.

1. Deklarera en pekar-variabel till `alt_modular_adc_dev`

2. Anropa **`altera_modular_adc_open`** `(const char *name)` där name t.ex. är `”CH0”` och tilldela

funktionsreturen till pekarvariabeln från punkt 1.

3. Fyll den utpekade structen manuellt eller med anropet av `void alt_adc_register_callback(`

```
alt_modular_adc_dev  *dev,
alt_adc_callback callback,
void *context,
alt_u32 sample_store_base)
```

med dev som pekaren från punkt 1 för det första funktionsparametern. För det andra argumentet så ska man deklarera en pekare till en funktionstyp som accepterar void som inparameter och returnerar void. Denna typkastas till typen `alt_adc_callback` vid tilldelningen av structmedlemsvariabeln. Om interrupt inte ska användas, så skrivs istället NULL. Det tredje argumentet är en pekare till void som tidigare deklarerats utanför, och den fjärde inparametern är i vårt fall `0x1081000` eller dess synonym `MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE.`

4. Anropa v`oid altera_modular_adc_init(alt_modular_adc_dev* dev, alt_32 ic_id, alt_32 irq)` med argumentslistan: pekaren från punkt1, andra parametern är gissningsvis 0 och tredje parametern ett negativt tal om interrupt inte ska användas. Om positivt tal anges så är detta förmodligen prioriteten på interrupten. Om man kan ange prioritet på interrupt, så innebär det att Nios II implementerar en s.k. ”nested interrupt controller”, där högre interrupt prioritet kan avbryte lägre prioriterade interrupt.

5. Anropa `void adc_set_mode_run_continuously(int sequencer_base)` där argumentet är `MODULAR_ADC_0_SEQUENCER_CSR_BASE (0x1081270)`

6. Anropa `void adc_start(int sequencer_base)` där argumentet är `MODULAR_ADC_0_SEQUENCER_CSR_BASE (0x1081270)`

12.1.1.2 ADC_READ_PHOTO

Makrot ADC_READ_PHOTO expanderas enligt

```
#define ADC_READ_PHOTO IORD_32DIRECT(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, 0) // Read ADC value
```

där `MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE` är definierat i system.h enligt

```
#define MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE 0x1081000
```

Det är tydligt att detta makro läser 32 bitar från adressen `0x1081000`

Vill man använda HAL så ska man istället använda funktionen
`int alt_adc_word_read (alt_u32 sample_store_base, alt_u32* dest_ptr, alt_u32 len)`
Sequnecern arbetar med som mest 64 slots. Hur många som man allokerar väljer man när man konfiguerar ADC IP-blocket i Qsys.

```
/**
 * alt_adc_word_read
*
 * Reading from sample store core (RAM) in word addressing
*
 *slot |   address_ofset    | ADC1 sample data    |  ADC0 sample data
 *     | (word addressing) |  MS2B(31-16)         |  LS2B(15-0 bits)

 * 0   |      0             |  xxxx xxxx xxxx     |  xxxx xxxx xxxx
 * 1   |      4             |  xxxx xxxx xxxx     |  xxxx xxxx xxxx
 * Arguments:
 * - sample_store_base: Base address of sample store core
 * - *dest_ptr: destination buffer
 * - len: size of reading in 32 bits.
 *
 * Returns:
 * 0 -> success
 * -EINVAL -> Invalid arguments
**/
int alt_adc_word_read (alt_u32 sample_store_base, alt_u32* dest_ptr, alt_u32 len)
{
    alt_u32 word = 0;
    alt_u32 word_length = len;
    alt_u32* dest_buf = dest_ptr;
    alt_u32 base = sample_store_base;

    /* return -EINVAL if invalid arguments passed into function
    */ if(NULL == dest_buf)
    {
      return -EINVAL;
    }

    for(word = 0; word < word_length; word++)
    {
       *dest_buf = IORD_32DIRECT((base + (word * 4)),0);

       dest_buf++;
```

```
}
```

```
    return 0;

}
```

Hur koppligen mellan kanaler och slots är gjord i Qsys anges i system.h. Man kan se att kanal 0 har högst prioritet och AD-omvandlas 62 ggr, kanal 4 en gång och kanal 12 en gång under en sequencer cykel. Men enligt Figur 5, så är kanal 0 pinnen på Max 10 FPGA inte inkopplad till BeMicro Max 10 kortet. (:-)

```
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_0 "CH4"

#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_1  "CH12"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_10  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_11  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_12  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_13  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_14  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_15  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_16  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_17  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_18  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_19  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_2 "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_20  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_21  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_22  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_23  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_24  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_25  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_26  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_27  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_28  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_29  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_3 "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_30  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_31  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_32  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_33  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_34  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_35  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_36  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_37  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_38  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_39  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_4 "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_40  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_41  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_42  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_43  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_44  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_45  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_46  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_47  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_48  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_49  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_5 "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_50  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_51  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_52  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_53  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_54  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_55 "CH0"
```

```
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_56  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_57  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_58  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_59  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_6 "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_60  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_61  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_62  "CH0"
#define  MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_63  "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_7 "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_8 "CH0"
#define MODULAR_ADC_0_SEQUENCER_CSR_CSD_SLOT_9 "CH0"
```

Om man bara läsa från slott 0, som är knuten till kanal 4, så görs läsning av AD-omvandlaren med HAL Api med anropet
`alt_adc_word_read (alt_u32 sample_store_base, alt_u32* dest_ptr, alt_u32 len)` enligt
`alt_adc_word_read (MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, alt_u32* dest_ptr, 1)`, där resultatet fås fram genom att dereferera den pekaren som man använt som `dest_ptr`.

För att läsa från slott 1, som är knuten till kanal 12, så anropar man enligt
```
alt_adc_word_read (MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE +4, alt_u32* dest_ptr, 1);
```

Vill man hämta båda resultaten så anropar man enligt
```
alt_adc_word_read (MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, alt_u32* dest_ptr, 2);
int resultat_slot0 = *dest_ptr;
int resultat_slot1 = *(++dest_ptr);
```

Funktionen returnerar 1 om läsningen lyckas eller `-EINVAL` om läsningen misslyckas, där EINVAL är definierat i errno.h enligt
```
#define      EINVAL 22    /* Invalid argument */
```

12.1.1.3 ADC_READ_TEMP

Makrot läser 32 bitar från `MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE+4`, vilket förmodligen är kanal 1.

```
#define ADC_READ_TEMP IORD_32DIRECT(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE, 4) // Read
ADC value
```

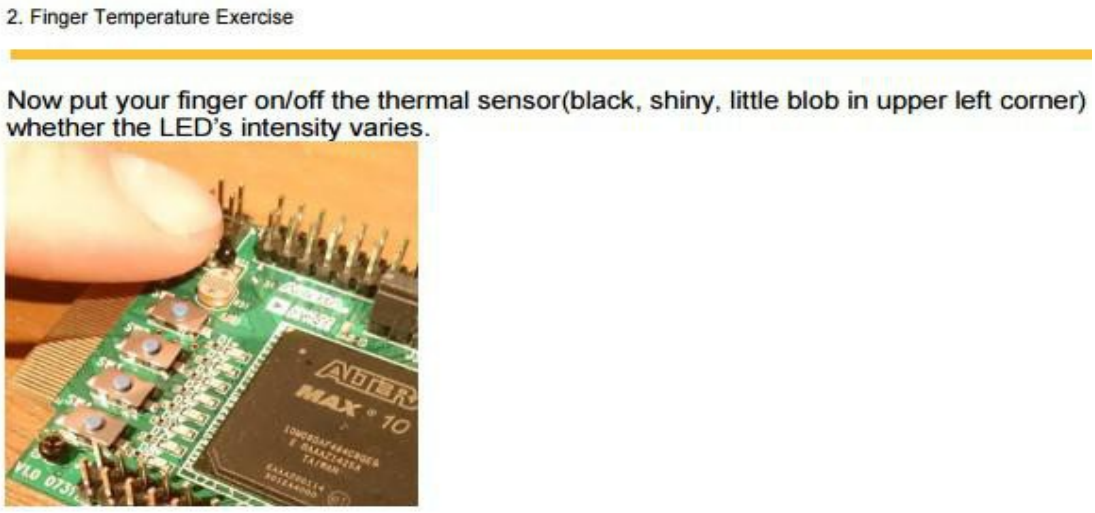

[4] https://www.altera.com/en_US/pdfs/literature/hb/max-10/ug_m10_adc.pdf

[5] https://www.altera.com/en_US/pdfs/literature/hb/max-10/m10_datasheet.pdf

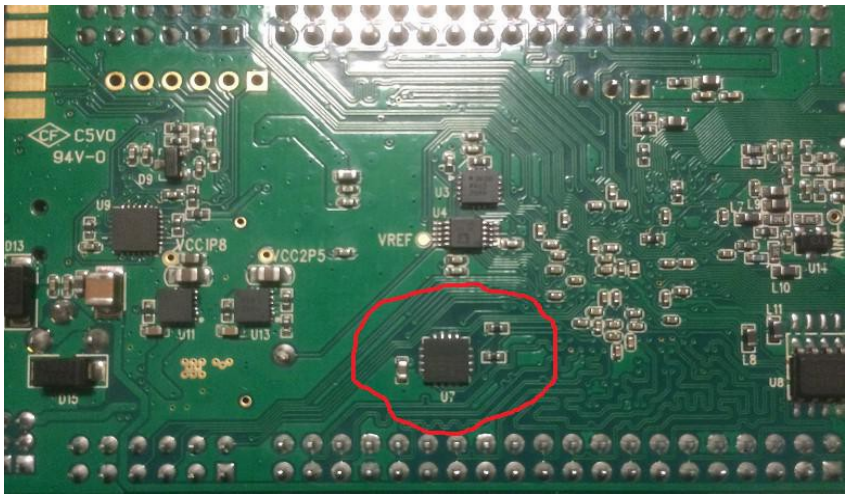
[6] http://www.tme.eu/en/Document/873710fa1254f968c852ae952b9f9132/bemicromax10.pdf

12.2 Temperaturgivaren

Temperaturgivaren är en ADT7420 från Analog Devices och är monterad på kortets undersida. Den har en intern 16-bitars AD-omvandlare av Sigma-delta typ, den kan dock konfigureras att endast använda 13 bitar.
Kommunikation med temperaturgivaren kan ske med I2C.
Se bifogat datablad i Appendix över detaljer kring givaren.
Programmet använder inte denna, utan termistorn på framsidan används.
Temperaturgivaren ska inte förväxlas med den enkla termistorn på kortets framsida. Se Figur 8 och Figur 9 för termistorn på kortets framsida repektive temperaturgivaren på kortets baksida. Inte heller bör man förväxla någon av dessa med den temperaturkänsliga dioden som är monterad ”*On chip*” och vars analoga spänning inte finns att tillgå på någon av FPGA:ns pinnar.



Figur 8 Termistor kopplad till Max 10 AD-omvandlare.

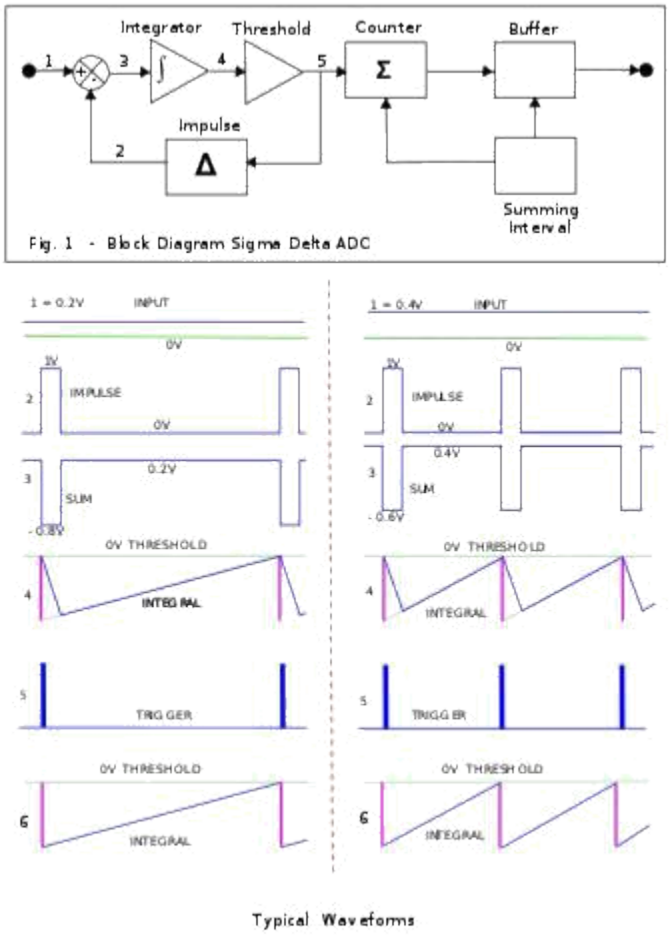


Figur 9 Tempgivare ADT7420 kopplad till I2C buss

12.2.1 Sigma delta omvandling för ADC

AD-omvandling av Sigma-delta typ är en förhållandevis komplicerad teori, som förmodligen ligger utanför ramen av denna utbildning, men i korthet så kan man säga att Sigma-delta omvandlaren generar ett pulståg som ackumuleras i en summator under en samplingsperiod, där pulståget har mycket högre frekvens än samplingsfrekvensen. Summan av pulståget antal pulser är proportionell emot spänningen som mäts. Ju högre spänning, desto högre summa ackumuleras.

För att åstadkomma detta krävs ett väldefinierat tidsavstånd mellan varje puls summering som representeras av samplingsperioden.
I början av varje samplingsperiod, genererar delta-enheten en positiv puls med bestämd magnitud. Refererandes till bilden från Wikipedia, så kommer den analoga signalen in vid (1), och under det korta samplingsintervall kan denna approximeras som konstant.
Vid början av varje samplingsperiod genererar Delta enheten en puls med bestämd magnitud. Skillnadsspänningen mellan (1) och (2) påföres integratorn som lämnar en spänning som är proportionell mot arean under grafen och den pålagda spänningen. Så en lägre skillnadsspänning åstadkommer en långsammare stigande ramp ut från integratorn än en högre spänning in till integratorn. Integrator spänningen kommer att stiga ända tills (4) är lika (1) enligt elementär reglerteori. När integratorns utspänning når den efterföljande komparatorns referensspänning s skapas en puls (5) som räknas i den efterföljande countern. Delta-enheten genererar en puls och cykeln börjar på nytt. Ett samplingsintervall innehåller alltså flera sådana cykler, som var och en resulterar i en puls. Summan av pulserna under ett sammplingsintervall är ett mått på mätsignalens spänningsnivå. I bilden nedan finns två olika inspänningar 0.2V och 0.4 Volt, och man kan tydligt se att den högre inspänningen kommer att resultera i ett högfrekventare pulståg, som i slutända kommer att resultera i en högra ackumulerad summa i slutet av varje samplingsintervall.



Figur 10 Typiska vågformer Delta-sigma AD-omvandalare

12.3 Accelerometern

Accelerometern är en ADXL362 från Analog Devices. Den kan detektera acceleration i tre vinkelräta axlar, så väl statiskt acceleration som dynamisk. Den åstadkommer detta genom att en minityriaserad vikt ändrar kapacitansen mellan två plattor, också dessa gjorde i MEMS teknologogi. Den kan konfigurereas för tre accelerations områden +/- 2g, +/-4g och +/-8g. Den har funktioner för självtest för att verifiera korrekt beteende. Troligtvis åstadkommes en elektromagnetisk påverkan av den miniatyriserade vikten vilket får kondensatorplattornas inbördes avstånd att anta ett bestämt läge, som skall resultera i en på förhand bestämd spänningsnivå för korrekt operation.

Gissningsvis åstadkommes de högre arbetsområdena genom en elektromagnetisk förspänning av viktens deflektionsarm.

Nogrannheten är 1 milli-g mellan axlarna, vilket betyder att den skulle kunna detektera en vinkelskillnad på 90 grader/1000 = 0.09 grader eller 1.5 milliradianer, vilket betyder att om man skulle använda denna som vattenpass, så är felet på avståndet 1 meter, bara 1.5 mikrometer.

Detta är tillräckligt noggrant för att kunna ersätta dyr laserbaserad precisionsmätutrustning för upprikting av CNC-maskiner.

Kommunikation med accelerometern sker enligt SPI-protokollet.

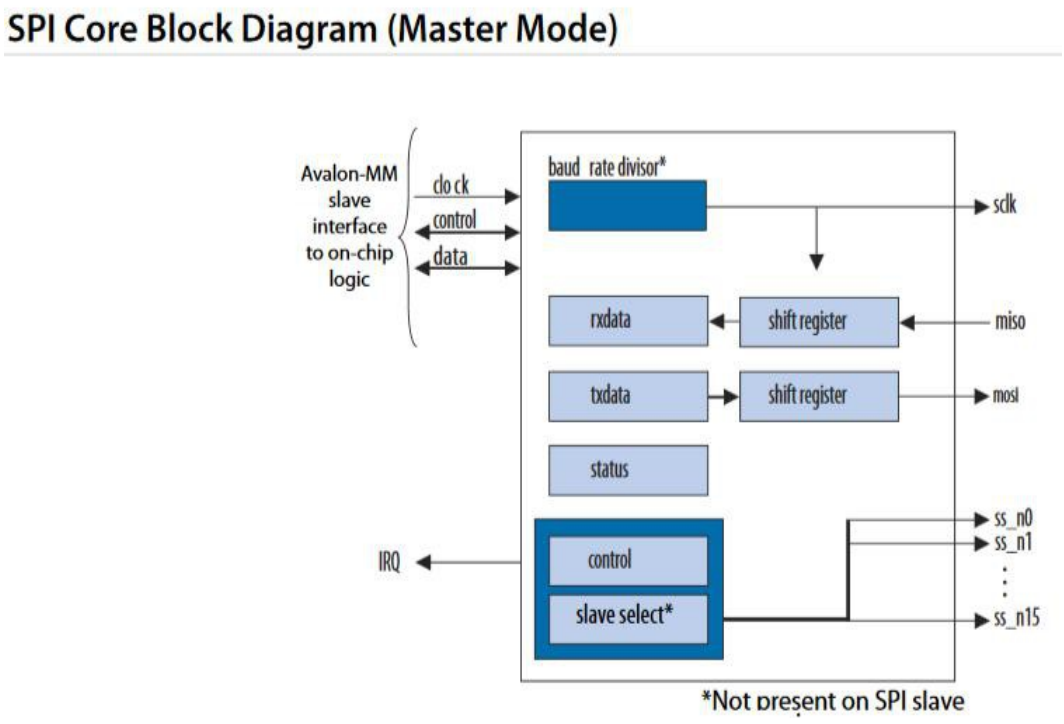
Wake-up mode, kontinuerlig sampling., skrivande till register.

De rigster som används i koden.

12.3.1 SPI kommunikation i Altera FPGA

SPI-kontrollern, interface till SPI-kontrollern, slav eller master, interrupt, svartlåda., tillhandahållna funktioner, konfigurering i Qsys.

Altera tillhandahåller SPI ipblock för MAX 10 FPGAN. Det är en SPI-kontroller som man konfigurerar i Qsys, till att vara antingen master-enhet eller slavenhet. SPI core, kallas denna kontroller och Altera erbjuder minnesmappad IO till dess kontrollregister och dataarea. SPI core kan också konfigureras att generera ett interrupt. Se Figur 11. På vänster sida ser vi interfacet emot host, som utgör programmerarens interface, och på höger sida syns interfacet till slav enheter. Om SPI core konfigurerats som slav finns inte slaveselect.



Figur 11 SPI core, interface emot host på vänster sida.

Minnesmappningen till SPI-cores interna register minnemappas enligt Figur 12, i fallet då SPI-core konfigurerats som master-enhet.

Register Map

An Avalon-MM master peripheral controls and communicates with the SPI core via the six 32-bit registers, shown in below in the **Register Map for SPI Master Device** figure. The table assumes an n-bit data width for `rxdata` and `txdata`.

Table 9-5: Register Map for SPI Master Device

Internal Address	Register Name	Type [R/W]	32-11	10	9	8	7	6	5	4	3	2-0
0	rxdata (8)	R	RXDATA (n-1..0)									
1	txdata (8)	W	TXDATA (n-1..0)									
2	status (6)	R/W			EOP	E	RRDY	TRDY	TMT	TOE	ROE	
3	control	R/W		SSO (7)	IEOP	IE	IRRDY	ITRDY		ITOE	IROE	
4	Reserved	—										
5	slaveselect (7)	R/W	Slave Select Mask									
6	eop_value(8)	R/W	End of Packet Value (n-1..0)									

Figur 12 SPI-kontrollerns register

Mappningen till minnet följer samma ordning, men man får tänka på att en minnes enhet är 8 bitar, så rxdata upptar 4 bytes med start vid SPI-controllerns mappade basadress, txdata hamnar då på adress basadressen +4.

I altera_avalon_spi_regs.h, ser vi överensstämmelse avseende ordningsföljden av SPI-kontrollerns interna register och dess minnesmappning.

```
#ifndef __ALTERA_AVALON_SPI_REGS_H__

#define __ALTERA_AVALON_SPI_REGS_H__

#include <io.h>

#define ALTERA_AVALON_SPI_RXDATA_REG                    0
#define IOADDR_ALTERA_AVALON_SPI_RXDATA(base)           __IO_CALC_ADDRESS_NATIVE(base,
ALTERA_AVALON_SPI_RXDATA_REG)
#define IORD_ALTERA_AVALON_SPI_RXDATA(base)             IORD(base,
ALTERA_AVALON_SPI_RXDATA_REG)
#define IOWR_ALTERA_AVALON_SPI_RXDATA(base, data)       IOWR(base,
ALTERA_AVALON_SPI_RXDATA_REG, data)

#define ALTERA_AVALON_SPI_TXDATA_REG                    1
#define IOADDR_ALTERA_AVALON_SPI_TXDATA(base)           __IO_CALC_ADDRESS_NATIVE(base,
ALTERA_AVALON_SPI_TXDATA_REG)
#define IORD_ALTERA_AVALON_SPI_TXDATA(base)             IORD(base,
ALTERA_AVALON_SPI_TXDATA_REG)
#define IOWR_ALTERA_AVALON_SPI_TXDATA(base, data)       IOWR(base,
ALTERA_AVALON_SPI_TXDATA_REG, data)

#define ALTERA_AVALON_SPI_STATUS_REG                    2
#define IOADDR_ALTERA_AVALON_SPI_STATUS(base)           __IO_CALC_ADDRESS_NATIVE(base,
```

```
ALTERA_AVALON_SPI_STATUS_REG)
#define IORD_ALTERA_AVALON_SPI_STATUS(base)          IORD(base,
ALTERA_AVALON_SPI_STATUS_REG)
#define IOWR_ALTERA_AVALON_SPI_STATUS(base, data)    IOWR(base,
ALTERA_AVALON_SPI_STATUS_REG, data)
#define ALTERA_AVALON_SPI_STATUS_ROE_MSK             (0x8)

#define ALTERA_AVALON_SPI_STATUS_ROE_OFST            (3)
#define ALTERA_AVALON_SPI_STATUS_TOE_MSK             (0x10)
#define ALTERA_AVALON_SPI_STATUS_TOE_OFST            (4)
#define ALTERA_AVALON_SPI_STATUS_TMT_MSK             (0x20)
#define ALTERA_AVALON_SPI_STATUS_TMT_OFST            (5)
#define ALTERA_AVALON_SPI_STATUS_TRDY_MSK            (0x40)
#define ALTERA_AVALON_SPI_STATUS_TRDY_OFST           (6)
#define ALTERA_AVALON_SPI_STATUS_RRDY_MSK            (0x80)
#define ALTERA_AVALON_SPI_STATUS_RRDY_OFST           (7)
#define ALTERA_AVALON_SPI_STATUS_E_MSK               (0x100)
#define ALTERA_AVALON_SPI_STATUS_E_OFST              (8)
#define ALTERA_AVALON_SPI_CONTROL_REG                3

#define IOADDR_ALTERA_AVALON_SPI_CONTROL(base)       __IO_CALC_ADDRESS_NATIVE(base,
ALTERA_AVALON_SPI_CONTROL_REG)
#define IORD_ALTERA_AVALON_SPI_CONTROL(base)         IORD(base,
ALTERA_AVALON_SPI_CONTROL_REG)
#define IOWR_ALTERA_AVALON_SPI_CONTROL(base, data)   IOWR(base,
ALTERA_AVALON_SPI_CONTROL_REG, data)
#define ALTERA_AVALON_SPI_CONTROL_IROE_MSK           (0x8)

#define ALTERA_AVALON_SPI_CONTROL_IROE_OFST          (3)
#define ALTERA_AVALON_SPI_CONTROL_ITOE_MSK           (0x10)
#define ALTERA_AVALON_SPI_CONTROL_ITOE_OFST          (4)
#define ALTERA_AVALON_SPI_CONTROL_ITRDY_MSK          (0x40)
#define ALTERA_AVALON_SPI_CONTROL_ITRDY_OFS          (6)
#define ALTERA_AVALON_SPI_CONTROL_IRRDY_MSK          (0x80)
#define ALTERA_AVALON_SPI_CONTROL_IRRDY_OFS          (7)
#define ALTERA_AVALON_SPI_CONTROL_IE_MSK             (0x100)
#define ALTERA_AVALON_SPI_CONTROL_IE_OFST            (8)
#define ALTERA_AVALON_SPI_CONTROL_SSO_MSK            (0x400)
#define ALTERA_AVALON_SPI_CONTROL_SSO_OFST           (10)
#define ALTERA_AVALON_SPI_SLAVE_SEL_REG              5

#define IOADDR_ALTERA_AVALON_SPI_SLAVE_SEL(base)     __IO_CALC_ADDRESS_NATIVE(base,
ALTERA_AVALON_SPI_SLAVE_SEL_REG)
#define IORD_ALTERA_AVALON_SPI_SLAVE_SEL(base)       IORD(base,
ALTERA_AVALON_SPI_SLAVE_SEL_REG)
#define IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(base, data) IOWR(base,
ALTERA_AVALON_SPI_SLAVE_SEL_REG, data)

#endif /* __ALTERA_AVALON_SPI_REGS_H__
```

Bas adressen *base* är adressen med offset noll, och fås ur system.h

```
/*
 * accelerometer_spi configuration
 *
 */

#define ACCELEROMETER_SPI_BASE 0x1081200
#define ACCELEROMETER_SPI_CLOCKMULT 1
#define ACCELEROMETER_SPI_CLOCKPHASE 0
#define ACCELEROMETER_SPI_CLOCKPOLARITY 0
#define ACCELEROMETER_SPI_CLOCKUNITS "Hz"
#define ACCELEROMETER_SPI_DATABITS 8 #define
ACCELEROMETER_SPI_DATAWIDTH 16 #define
ACCELEROMETER_SPI_DELAYMULT "1.0E-9" #define
ACCELEROMETER_SPI_DELAYUNITS "ns" #define
ACCELEROMETER_SPI_EXTRADELAY 0 #define
ACCELEROMETER_SPI_INSERT_SYNC 0 #define
ACCELEROMETER_SPI_IRQ 0
#define ACCELEROMETER_SPI_IRQ_INTERRUPT_CONTROLLER_ID
0 #define ACCELEROMETER_SPI_ISMASTER 1
#define ACCELEROMETER_SPI_LSBFIRST 0
#define ACCELEROMETER_SPI_NAME "/dev/accelerometer_spi"
#define ACCELEROMETER_SPI_NUMSLAVES 1
#define ACCELEROMETER_SPI_PREFIX "spi_"
#define ACCELEROMETER_SPI_SPAN 32
#define ACCELEROMETER_SPI_SYNC_REG_DEPTH 2 #define
ACCELEROMETER_SPI_TARGETCLOCK 128000u #define
ACCELEROMETER_SPI_TARGETSSDELAY "0.0" #define
ACCELEROMETER_SPI_TYPE "altera_avalon_spi"
#define ALT_MODULE_CLASS_accelerometer_spi altera_avalon_spi
```

12.3.1.1 rxdata

Minesmappade registret *rxdata* läses av host. Då contollerns mottagarskiftregister har mottagit n bitar där storleken på meddellande längdsenhet definieras under konfigurationen i Qsys, så sätts flaggan receive ready RRDY i statusregistret och skiftsregistrets innehåll överförs till det interna rx-registret (minnesmappat till basadressen)- Läsning av rxdata clearar nämnd statusflagga. Om läsning inte har gjorts och mottagarskiftregistret skriver data till rxdata medan RRDY är hög hissas recieve overrun ROE i statusregistret. Innehehållet i rxdata måste då ses som korrupt.

12.3.1.2 txdata

Om host vill sända data över SPI, så ska sänd-data skrivas till minnesmappade registret txdata. SPI-kontrollern indikerar att den är redo att ta emot ny sänd-data genom att sätta statusflaggan transmit ready TRDY. När skrivning har gjorts till txdata sätter SPI-kontrollern TRDY låg, och överför sänd-data till skiftregistret som är avsett för sändning. Skulle man skriva till txdata medan TRDY är låg hissar SPI-kontrollern transmit overrun TOE i statusregistret.

12.3.1.3 Status registret

SPI-kontrollerns interna statusregister är minnesmappat till basadressen + 8, där de utgörs av bitarna 3 till 9, och är i ordningsföljd ROE (Receive Overrun Error), TOE (Transmit overrun error), TMT (Transmit Shift Register Empty), TRDY (Transmit Ready), RRDY (Recieve Ready), E (Error) och EOP (End Of Packet). Förutom läsning av Status registret, så kan också skrivning göras. En skrivning nollställer flaggorna ROE, TOE, och E.

12.3.1.4 Kontroll registret

SPI-kontrollerns kontrollregister är minnesmappat till basadress + 12, och innehåller styrbitar på positionerna bit 3 till bit 10. De olika bitarna är avsedda att definiera interrupt baseradt på statusflagorna. Figur 13 beskriver resultat som fås om de olika bitarna ettställs.

Table 9-7: control Register Bits

#	Name	Description
3	IROE	Setting IROE to 1 enables interrupts for receive-overrun errors.
4	ITOE	Setting ITOE to 1 enables interrupts for transmitter-overrun errors.
6	ITRDY	Setting ITRDY to 1 enables interrupts for the transmitter ready condition.
7	IRRDY	Setting IRRDY to 1 enables interrupts for the receiver ready condition.
8	IE	Setting IE to 1 enables interrupts for any error condition.
9	IEOP	Setting IEOP to 1 enables interrupts for the End of Packet condition.
10	SSO	Setting SSO to 1 forces the SPI core to drive its ss_n outputs, regardless of whether a serial shift operation is in progress or not. The slaveselect register controls which ss_n outputs are asserted. SSO can be used to transmit or receive data of arbitrary size, for example, greater than 32 bits.

Figur 13 SPI-kontrollerns kontroll register

12.3.1.5 Status registret

Detta register är en bitfältsmask för att selektera den slavenhet som mastern väljer att kommunicera med. En slavenhet adresseras genom att sätta den korresponderande biten i maskregistret.
Till delningen av bitar för olika slavar görs vid konfigureringen i Qsys.

Vid reset nollställs samtliga bitar i maskregistret utom bit noll, som ställs. Således är slav-enhet noll förvalt.

12.3.1.6 END OF PACKET registret

Altera låter användaren välja utseendet på det bifältet som ska utgöra end-of-packet, dvs. indikeringen att transmissionen är slut. I EOP-registret definieras detta bitfält.

12.3.1.7 Högnivå anrop för kommunikation över SPI

Eftersom konfigureringen av SPI-kontrollern sker i Qsys, så behövs inte en massa funktioner och makron för att hantera SPI-kommunikationen. Altera tillhandahåller en funktion för avseendet.

```
int alt_avalon_spi_command(alt_u32 base, alt_u32 slave,
                           alt_u32 write_length, const alt_u8 * write_data,
                           alt_u32 read_length, alt_u8 * read_data,
                           alt_u32 flags);
```

för avseendet.Parametrarna för funktionen är base basadressen för den minnesmappning SPI-controllern har och som återfinns i system.h, i vårt fall0x1081200, därefter slave vilken slav man avser kommunicera med, vilket resulterar i en skrivning till SPI-kontrollerns interna Slave select mask register, därefter write_length som är längden på meddellandet, därefter write_data en pekare till skrivdata där databredden max får vara en byte.
Funktionen kommer själv att skriva konsekutiva write-data till txdata-adressen.
Vidare read_length längden på läsmeddelandet, read_data en pekare som kommer att peka till första adressen för läsmeddellandet, där funktionen själv hämtar data från rxdata-adressen och skriver till det ställa vars startadress read_data kommer att peka till.
Sista parametern är flags, där skrivning resulterar i nollställning av ROE, TOE, och E.

12.3.2. Kommunikation med accelerometern över SPI

Databladet för ADXL362 anger följande ordning för kommunikation

Kommandona är

0x0A, skrivning till register

0x0B, läsning från register

0x0D, läsning från FIFO.

Protokollet för kommunikation är:
Skrivning från mastern:
<kommando 0x0A><adress byte(adress till registret)> <data byte>

Läsning av register till mastern:
<kommando 0x0B><adress byte(adress till registret)>

Läsning av FIFO till
mastern: <kommando 0x0D>

12.3.2.1 Analys av exempelkoden

Nedan följer exemplekoden med de delar som inte är relevanta för SPI-kommunikationen borttagna.

```
#include <stdio.h>
#include <io.h>
#include <system.h>
#include <alt_types.h>
#include <altera_avalon_spi_regs.h>
#include "altera_avalon_spi.h"

#define WRITE_COMMAND 0x0A //SPI write command.
#define READ_COMMAND 0x0B //SPI read command.
#define POWER_CONTROL_REGISTER
0x2D #define MEASUREMENT_MODE 0x02
#define X_AXIS_MSB_REGISTER 0x08

int main()
{
      //Variables
```

```
alt_u8 spi_command_tx[3];
alt_u8 spi_command_rx[3];
```

```
      typedef struct mystruct
            { alt_8 x;
            alt_8 y;
            alt_8 z;
      } ACCELEROMETER;

      ACCELEROMETER accel_data;

/* START ACCELEROMETER MEASUREMENT MODE */
      spi_command_tx[0] = WRITE_COMMAND; // write command
      spi_command_tx[1] = POWER_CONTROL_REGISTER; // Power controregister
      spi_command_tx[2] = MEASUREMENT_MODE; // Measurement mode/start.

      //SPI command.
      alt_avalon_spi_command(ACCELEROMETER_SPI_BASE
      ,
                  0,3,spi_command_tx,0,spi_command_rx,0);

/* START READING FROM ACCELEROMETER VIA SPI INTERCONNECTION */
      spi_command_tx[0] = READ_COMMAND; // read command
      spi_command_tx[1] = X_AXIS_MSB_REGISTER; // Start reading from
      X-Axis MSB register.




//SPI command.
      alt_avalon_spi_command( ACCELEROMETER_SPI_BASE,
                              0,2, spi_command_tx,3, &accel_data,0);

//Print to console.
      printf("X-Axis: %d\t Y-Axis: %d\t Z-Axis:
      %d\n",accel_data.x,accel_data.y,accel_data.z);

      return 0;

      }
```

Vi har behandlat de tre kommandona i tidigare stycke.
Programmet definierar POWER_CONTROL_REGISTER som värdet 0x2D, och MEASUREMENT_MODE som värdet 0x02.
Anledningen ser vi i databladet

POWER CONTROL REGISTER

Address: 0x2D, Reset: 0x00, Name: POWER_CTL

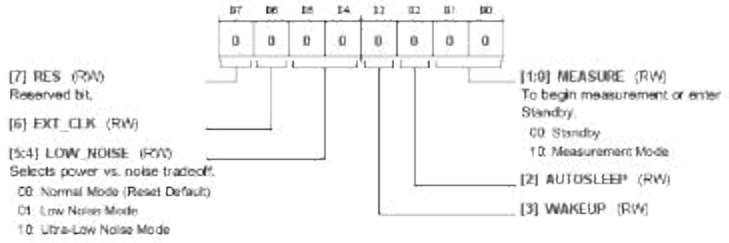


Table 18. Bit Descriptions for POWER_CTL

Bits	Bit Name	Settings	Description	Reset	Access
7	Reserved		Reserved.	0x0	RW
6	EXT_CLK		External Clock. See the Using an External Clock section for additional details. 1 = the accelerometer runs off the external clock provided on the INT1 pin.	0x0	RW
[5:4]	LOW_NOISE		Selects Power vs. Noise Tradeoff:	0x0	RW
		00	Normal operation (reset default).		
		01	Low noise mode.		
		10	Ultralow noise mode.		
		11	Reserved.		
3	WAKEUP		Wake-Up Mode. See the Operating Modes section for a detailed description of wake-up mode. 1 = the part operates in wake-up mode.	0x0	RW
2	AUTOSLEEP		Autosleep. Activity and inactivity detection must be in linked mode or loop mode (LINK/LOOP bits in ACT_INACT_CTL register) to enable autosleep; otherwise, the bit is ignored. See the Motion Detection section for details. 1 = autosleep is enabled, and the device enters wake-up mode automatically upon detection of inactivity.	0x0	RW
[1:0]	MEASURE		Selects Measurement Mode or Standby.	0x0	RW
		00	Standby.		
		01	Reserved.		
		10	Measurement mode.		
		11	Reserved.		

Figur 14 Power Ctrl registret

Skrivmeddellandet sammanställs därför enligt

```
spi_command_tx[0] = WRITE_COMMAND; // write command spi_command_tx[1]
= POWER_CONTROL_REGISTER; // Power controregister spi_command_tx[2] =
MEASUREMENT_MODE; // Measurement mode/start.
```

Meddelandet sänds med Altera-funktionen

```
alt_avalon_spi_command(ACCELEROMETER_SPI_BASE,
                0,3,spi_command_tx,0,spi_command_rx,0)
                ;
```

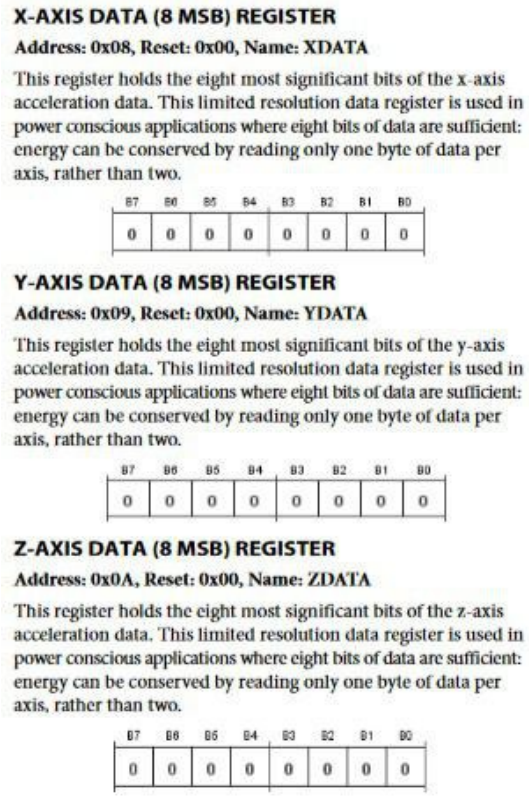
Första parametern *ACCELEROMETER_SPI_BASE* är basadressen till minnesmappningen av SPI-kontrollerns interna register. Slavens identitet är den med identitetssiffran 0, längden på meddelandet är 3, därefter pekaren *spi_command_tx* till första byten i meddelandet. Namnet på en array är synonymt med pekaren till första adressen i arrayen. Läsmeddelandet längd är 0. Pekaren

till läsdata är *spi_command_rx* men man kunde ha skrivit NULL, eftersom det utkrävda läsmeddelandets längd är noll. Slutligen nollställs status registret.

Därefter sätts ett nytt meddelande samman enligt

```
spi_command_tx[0] = READ_COMMAND;
spi_command_tx[1] = X_AXIS_MSB_REGISTER;
```

Läskommandot är 0x0B, register angivelsen är 0x08, vilket innehåller de 8 mest signifikanta bitarna för accelerationen i x-led, se Figur 15



X-AXIS DATA (8 MSB) REGISTER

Address: 0x08, Reset: 0x00, Name: XDATA

This register holds the eight most significant bits of the x-axis acceleration data. This limited resolution data register is used in power conscious applications where eight bits of data are sufficient: energy can be conserved by reading only one byte of data per axis, rather than two.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	0

Y-AXIS DATA (8 MSB) REGISTER

Address: 0x09, Reset: 0x00, Name: YDATA

This register holds the eight most significant bits of the y-axis acceleration data. This limited resolution data register is used in power conscious applications where eight bits of data are sufficient: energy can be conserved by reading only one byte of data per axis, rather than two.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	0

Z-AXIS DATA (8 MSB) REGISTER

Address: 0x0A, Reset: 0x00, Name: ZDATA

This register holds the eight most significant bits of the z-axis acceleration data. This limited resolution data register is used in power conscious applications where eight bits of data are sufficient: energy can be conserved by reading only one byte of data per axis, rather than two.

B7	B6	B5	B4	B3	B2	B1	B0
0	0	0	0	0	0	0	0

Figur 15 MSB register

Därefter sänds meddelandet med Alteras tillhandahållan SPI-funktion

```
alt_avalon_spi_command( ACCELEROMETER_SPI_BASE,
                        0,2, spi_command_tx,3, &accel_data,0);
```

Argumenten är basadressen, slav id, meddelandets längd, meddelandet, tre bytes utkrävs som mastern vill läsa, pekaren till vart läsdata ska levereras, och slutligen nollställning av status-registret.

12.4 VGA-hårdvaran

VGA kontrollen fungerar så att du har en basadress och sen räknas pixeladressen fram med "Pixel RAM Address = X + Y*320;

13 SW-manual VGA-IP

void print_pix(alt_u32 x,alt_u32 y,alt_u32 rgb);
Ritar en pixel vid koordinat x, y, med färgen rgb, där rgb-värdet är 0 till 7.

```
void print_hline(alt_u32 x_start,alt_u32 y_start, alt_u32 len,alt_u32 RGB);
Ritar en horisontell linje med längd len med start x, y, och färg RGB där rgb-värdet
är 0 till 7.
```

```
void print_vline(alt_u32 x_start,alt_u32 y_start, alt_u32 len, alt_u32 RGB);
Ritar en vertikal linje med längd len med start x, y, och färg RGB där rgb-värdet är
0 till 7.
```

```
void print_char(alt_u32 x,alt_u32 y,alt_u32 rgb,alt_u32 BG_RGB,char Character);
Skriver ut ett ascii-tecken, vid koordinat x, y, med färg rgb mot
bakgrundsfärgen BF_RGB, där rgb-värdena är 0 till 7.
```

```
void print_str(alt_u32 x_start, alt_u32 y_start,alt_u32 rgb,char *str);
Skriver ut en sträng, mot svart bakgrund, där färgen på tecknen är rgb, och där
rgb-värdena är 0 till 7.
```

```
void print_circle(alt_u32 radie, alt_u32 x_centrum, alt_u32 y_centrum, alt_u32 rgb);

Skriver ut en fylld cirkel med centrum, x, y och radie r, med färg rgb, där rgb-
värdena är 0 till 7.
```

```
void print_empty_circle(alt_u32 radie, alt_u32 x_centrum, alt_u32 y_centrum, alt_u32
rgb);
Skriver ut en ofylld cirkel med centrum, x, y och radie r, med färg rgb, där
rgb-värdena är 0 till 7.
```

```
void clear_screen(alt_u32 rgb);
Rensar skärmen.Skärmen färgas svart.
```

```
pixel_data read_pixel_ram_int(alt_u32 x_start, alt_u32 y_start);
Returnernar ett värde av typ pixel_data, som är ett värde 0 till 7 och
representerar färgen på pixeln vid x,y.
Typen typkastas till alt_u8.
```

14 Projektkostnad

Tiden som gått åt för projektet kan inte längre uppskattas, p.g.a. ovana eller avsaknaden av datorstöd för att dokumentera tidsåtgång.
Därför sätts en schablonkostnad till 4 arbetsveckor dvs. totalt 160 timmars arbete, vilket troligtvis är i underkant. Detta motsvarar kostnaden 56 000 kronor.
Men inte ens i skrivande stund är projektet klart, utan kvarstår är att göra i ordning detta för inlämning, vilket kräver ytterligare några timmar.

15 Referenser

[1] http://www.alterawiki.com/wiki/BeMicro_Max_10

[2] https://cloud.altera.com/devstore/platform/?family=max-10&board=4

[3] http://www.alterawiki.com/uploads/5/5e/BeMicroM10_Embedded_System_Lab.pdf

[4] https://www.altera.com/en_US/pdfs/literature/hb/max-10/ug_m10_adc.pdf

[5] https://www.altera.com/en_US/pdfs/literature/hb/max-10/m10_datasheet.pdf

[6] http://www.tme.eu/en/Document/873710fa1254f968c852ae952b9f9132/bemicromax10.pdf