David Forsling
david.forsling@telia.com
2015-12-27

Loggning av ljusintensitet

SAMMANFATTNING:
Ett system för loggning av ljusintensitet beskrivs i denna rapport. Systemet visar data från loggningen på en bildskärm med mellan 53 sekunder och 2800 minuter historik.

Innehållsförteckning

1 Inledning

Ett system som loggar ljuset i en lokal har konstruerats. Loggningen visas på en bildskärm med ett diagram där den vertikala axeln visar ljusets intensitet och den horisontella axeln visar tid. Hårdvaruplattform är kretskortet BeMicro Max 10 och den inbyggda fotoresistorn används till att läsa ljusets intensitet med fyra möjliga samplingsfrekvenser.

2 Kravspecifikation

Tabell 1: Kravspecifikation

Krav id	Förstudie	Utfört
Krav_1	Beskriv två äldre projekt ur Low level C synvinkel, se arkivet i itslearning under ”Konstruktionsmetodik_och _teknisk_dokumentation”. (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 och den ljuskänsliga komponenten. För mera information se boken. (Eget kapitel)	Ja
Krav_2d	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.*	*N/A*
Krav_4a	Projektet specifikationen ska minst innehålla ljusmätaren, knappar, lysdioder och VGA. Alla tryckknapparna behöver inte användas.	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 av ljusmätaren; 5 Hz, 2 Hz, 1Hz och 0,1Hz. En hel skärm ska kunna visas. 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 sagt att alla typerna i anropen ska vara exakt definierade, som tex ”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 (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);`	Ja

	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 två äldre projekt (som liknar det valda projektet) 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 delsystem), o.s.v.
	- Verktygs- ”settings”, eget kapitel
	  * Dokumentera alla settings
	  * Eventuella länkdirektiv
	- Verifiering, eget kapitel
	  * Fyll i testprotokollet och ev buggar antecknas. Konstruktionen behöver inte vara buggfri.
	- Analys, eget kapitel
	  * Felmeddelanden ”Warnings” från kompileringen
	  * Minimera ”foot print”, koden
	- Validering, eget kapitel
	  * Visa med resultat från testprotokollet (validering)
	  * Kombineras med en film (kort) som beskriver för en utomstående vad som genomförts (visa gärna en systemarkitektur) och demonstrera att det fungerar (en länk till YouTube).
	  * Sammanfattning, kända buggar och framtida förbättringar
	  * Referenser
		Ja

Krav_8	VGA-manual för SW-ingenjörer (I bilagan på rapporten och under doc i VGA IP-komponenten)	Ja

	Leveranskrav	

Krav_9	Leveransen ska innehålla tre foldrar:
	- Konstruktionsbeskrivning
	  * Konstruktionsrapport
	- Konstruktionsdokument
	  * Alla C-filer i (software/BSP och Projekt mapp)
	  * Alla IP-komponenter (IP mapp)
	  * SOF- och SOPCINFO-filen
	- Diverse
		Ja

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 schemat (för VG).	Ja

3 Projektplan

Beräknad tid för att genomföra projektet är 12 arbetsdagar. Eftersom ett flertal andra projekt pågår samtidigt måste arbetstiden spridas ut över veckorna. Två arbetsdagar per vecka under 6 veckor är den planerade fördelningen av tiden.

3.1 Tidplan

Vecka 1: Konstruktion av VGA-rutiner.
Vecka 2: Konstruktion av funktioner för A/D-omvandling.
Vecka 3: Konstruktion av funktioner som hanterar loggning.
Vecka 4: Systemkonstruktion och felsökning.
Vecka 5: Rapportskrivning.
Vecka 6: Rapportskrivning.

3.2 Budgeterad kostnad

Kostnaden för att genomföra projektet på 12 arbetsdagar med en timlön på 500 kronor blir 48000 kronor.

3.3 Verklig kostnad

Projektet skulle enligt planen genomföras på 96 timmar. Den verkliga arbetstiden blir ungefär 120 timmar vilket innebär att kostanden blir 60000 kronor.

4 Beskrivning av två projekt från tidigare år

Två projekt från tidigare år sammanfattas i detta kapitel.

4.1 5 i rad

I detta avsnitt sammanfattas spelet 5 i rad som är konstruerat av Lars Högberg.

Programmet är uppbyggt av tre moduler. Huvudprogrammet main.c använder header-filen femirad.h som innehåller applikationsspecifika makron och konstanter. Programmets funktioner ligger i en separat fil som heter funktioner.inc.

En tillståndsmaskin i main sköter spelets grundfunktioner som exempelvis att läsa tangenter, uppdatera vga och lcd samt att kontrollera om någon spelare har vunnit.

Ett antal åtgärder har utförts för att minimera exekveringstiden. Multiplikation och division har använts sparsamt och i de fall de behövs har de placerats i delar av koden som utförs sällan. Variabeltyper har valts så att type casting undviks så mycket som möjligt.

Minnesanvändning har minimerats genom att spelplanen utformats med ett 32 bitars ord i minnet per rad där varje bit representerar en av spelplanens 32 kolumner. Kompileringsinställningarna har justerats med inaktivering av C++ samt aktivering av optimering mot minskad storlek, reducerade drivrutiner och litet C-bibliotek. Storleken på koden blir med optimeringarna endast 13 kB.

4.2 Digital Labyrintspel

Det andra projektet som sammanfattas heter "Digitalt Labyrintspel" och är konstruerat av Patric Sjöberg. Spelet är en digital variant av det klassiska labyrintspelet där en kula ska rulla genom en labyrint utan att ramla ner i ett hål på vägen.

Tre moduler används i konstruktionen. Huvudfilen innehåller main() som styr själva spelet. Filen Drawings.c innehåller de funktioner som main använder och Drawings.h innehåller funktionernas deklarationer.

Huvudprogrammet börjar med att rita upp spelplanen som har blå bakgrund och svarta hål samt röda ytterkanter och labyrintväggar. Därefter ritas kulan som ska rulla genom labyrinten och huvudloopen startas.

Huvudloopen börjar med läsning av tangenter för att bestämma kulans färdriktning. När kulan rullar genom labyrinten fattas beslut om nästa steg genom att läsa färgen hos de pixlar som kulan kommer att rulla över. Om någon av pixlarna är svart tar spelet slut och om någon pixel är röd stannar kulan. Målet har nåtts om någon pixel är vit och kulan får rulla vidare i tänkt färdriktning när alla pixlar är blå.

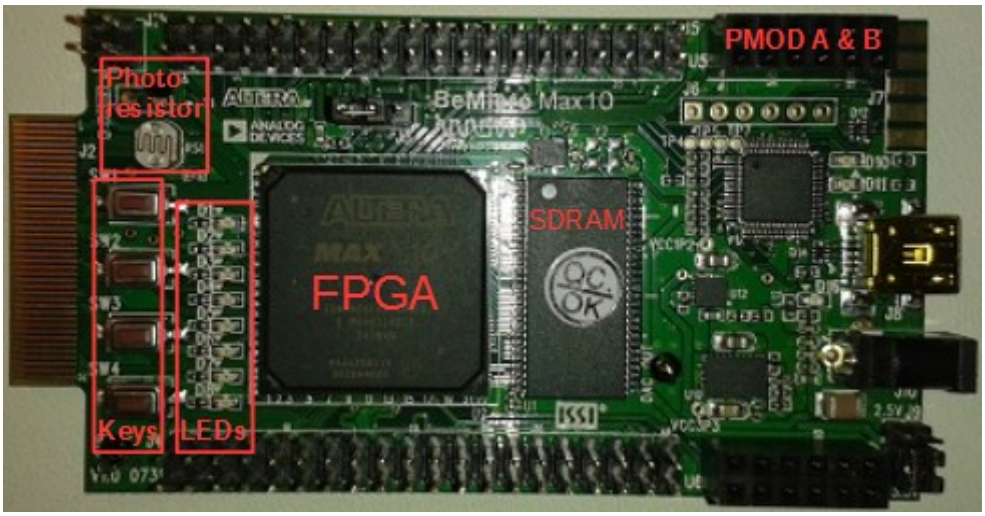
Kommentar: En variant skulle kunna vara att låta kulan fortsätta rulla tills ny riktning väljs. Spelet skulle också kunna utökas till att styras med en accelerometer.

5 BeMicro Max 10

Hårdvaruplattform för projektet är BeMicro Max 10 som tillverkas av Arrow Electronics. BeMicro är ett utvärderingskort med en Altera MAX 10 FPGA. Kortet har 8 MB SDRAM, en 12 bitars D/A-omvandlare, accelerometer, I2C-temperatursensor, temperaturresistor, fotoresistor samt ett antal knappar och lysdioder.

FPGA-kretsen har 8000 logiska element, 414 kbit RAM och 256 kbit flashminne som möjliggör självkonfigurering vid uppstart. En A/D-omvandlare med 18 ingångar är inbyggd i kretsen. En av ingångarna är kopplad till en temperaturkänslig diod och resten av kanalerna är tillgängliga via kretsens portar. Vidare finns två PLL, 18 multiplicerare och 250 GPIO.

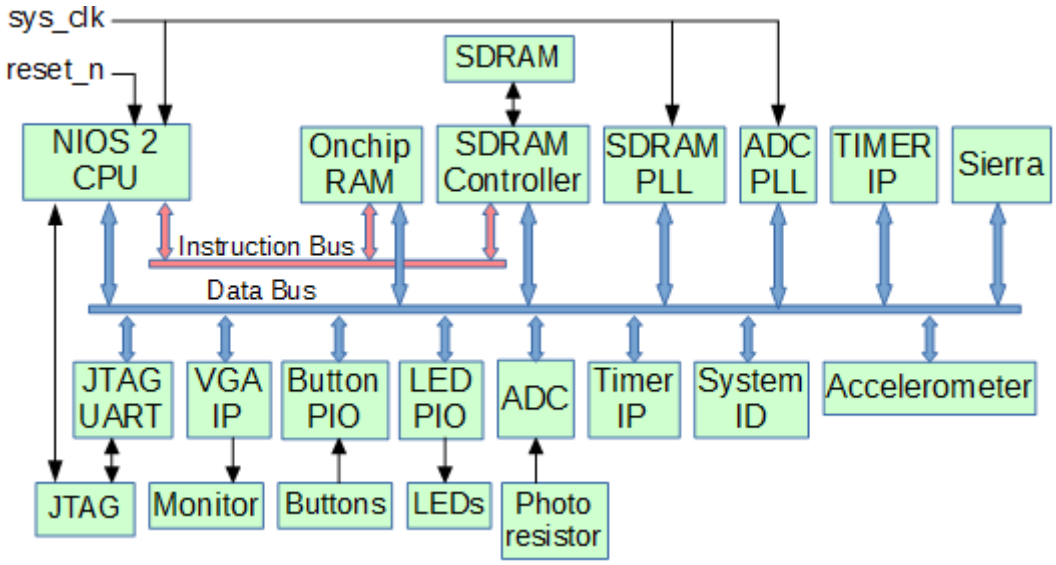
Kortet har två 40 pinnars kontaktlister som ger tillgång till 64 digitala I/O samt en kontakt på mönsterkortets kortsida med 80 anslutningar. Det finns två dubbla PMOD-anslutningar och kontakter som ger tillgång till 6 st analoga ingångar. Figur 1 visar BeMicro med de enheter som används i detta projekt markerade.



Figur 1: BeMicro Max 10

6 Mikrodatorarkitektur

Programkoden körs av ett mikrodatorsystem som är baserat på mjukvaruprocessorn Nios II Economy. Datorsystemet innehåller även ett antal I/O-komponenter varav flertalet används i konstruktionen som till exempel A/D-omvandlaren, VGA-komponenten, knapparna och lysdioderna. En illustration över datorarkitekturen visas i figur 2.



Figur 2: Mikdrodatorns arkitektur

6.1 I/O-komponenter

Nedan följer en beskrivning av systemets I/O-komponenter.

jtag_uart_0 : En serieport som kommunicerar över JTAG-gränssnittet.

sysid_qsys_0 : En komponent som innehåller information om hårdvaruversion.

modular_adc_0_sequencer_csr : Sequencern väljer ingång till A/D-omvandlaren.

TIMER_HW_IP_0 : Timerkomponent som mäter tid genom att räkna klockcykler.

ADC_PLL : Genererar klocksignal till A/D-omvandlaren.

SDRAM_PLL : Generar klocksignal till SDRAM-minnets styrenhet.

pio_key_1_3 : Parallell I/O-komponent för knapparna.

pio_LED : Parallell I/O-komponent för lysdioderna.

spi_accelerometer : Komponent som läser från accelerometern över SPI-bussen.

modular_adc_0_sample_store_csr : Minnesutrymme för sampeldata från A/D-omvandlaren.

onchip_memory2_0 : FPGA-kretsens inbyggda SRAM.

IP_VGA_0 : VGA-komponent för bildskärm ansluten till PMOD A och B.

sdram_controller : Styrenhet för SDRAM-minnet.

sierra_0 : Ett hårdvarubaserat realtidsoperativsystem.

6.2 Minneskarta

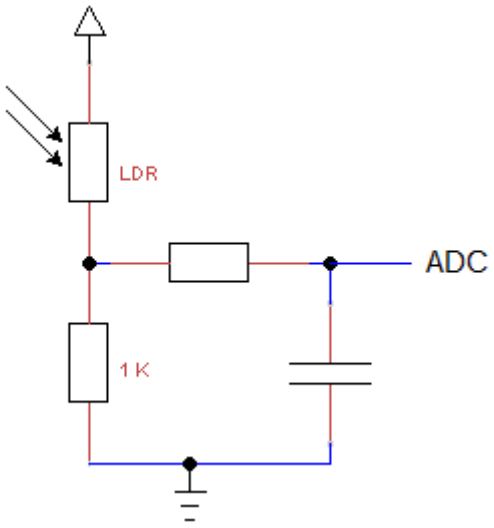
Tabell 2 visar de ingående komponenternas minnesområden.

Tabell 2: Minneskarta

Slave Descriptor	Address Range	Size	Attributes
jtag_uart_0	0x01203680 - 0x01203687	8	printable
sysid_qsys_0	0x01203678 - 0x0120367F	8	
modular_adc_0_sequencer_csr	0x01203670 - 0x01203677	8	
TIMER_HW_IP_0	0x01203660 - 0x0120366F	16	
ADC_PLL	0x01203650 - 0x0120365F	16	
SDRAM_PLL	0x01203640 - 0x0120364F	16	
pio_key_1_3	0x01203630 - 0x0120363F	16	
pio_LED	0x01203620 - 0x0120362F	16	
spi_accelerometer	0x01203600 - 0x0120361F	32	
modular_adc_0_sample_store_csr	0x01203400 - 0x012035FF	512	
onchip_memory2_0	0x01201000 - 0x01201FFE	4095	memory
IP_VGA_0	0x01000000 - 0x011FFFFF	2097152	
sdram_controller	0x00800000 - 0x00FFFFFF	8388608	memory
sierra_0	0x00000000 - 0x000003FF	1024	

7 A/D-omvandling

Ljusets intensitet registreras med hjälp av en fotoresistor (LDR) som bildar en spänningsdelare tillsammans med ett 1 kOhm motstånd. De båda motstånden är kopplade till A/D-omvandlarens ingång enligt figur 3. Fotoresistorns resistans minskar med ökad ljusstyrka vilket innebär att spänningen ökar för A/D-omvandlarens insignal när ljusstyrkan ökar.



Figur 3: Ljuskänsligt motstånd

FPGA-kretsens inbyggda A/D-omvandlare är en 12 bitars SAR ADC. SAR betyder Successive Approximation Register och innebär att omvandlingen sker genom att inspänningen jämförs med en spänning som ändras successivt tills de båda spänningarna är lika. Halva referensspänningen används vid första jämförelsen och resultatet avgör den mest signifikanta biten i värdet samt om nästa jämförelse ska göras mot 1/4 eller 3/4 av referensspänningen. Genom att fortsätta jämförelserna med halverat område för varje tillfälle närmar sig spänningarna varann och efter 12 upprepningar har rätt värde hittats för en 12 bitars ADC.

A/D-omvandlarens IP-block består av tre delar: sequencer, sample store och control block varav de två första är tillgängliga via minnesadresser enligt tabell 2. Sequencern styr vilka ingångar som ska läsas av omvandlaren samt ordningsföljden för ingångarna när omvandlingen sker. Denna sekvens ställs in i QSYS. Sequencerns register används för att starta en sekvens eller aktivera kontinuerlig uppdatering. Bit 1 startar en sekvens när värdet är 0 och bit 0 aktiverar kontinuerlig uppdatering när värdet är 1.

Resultaten från omvandlingen lagras i sample store och är tillgängliga för läsning via bussen. Värden lagras i den ordning som anges av sekvensen för läsning och sekvensens första värde lagras på basadressen med offset 0. För att aktivera kontinuerlig uppdatering och sedan läsa värdet från sekvensens första ingång används två HAL-kommandon:

```
IOWR_32DIRECT(MODULAR_ADC_0_SEQUENCER_CSR_BASE,0,0x1);
IORD_32DIRECT(MODULAR_ADC_0_SAMPLE_STORE_CSR_BASE,0);
```

8 VGA

En utgång för vga-bildskärm är kopplad till PMOD-anslutning A och B. Vga-signalerna R, G och B är anslutna till var sin pinne i kontakten och då PMOD-signalerna är vanliga digitala I/O kan varje färg endast vara av eller på. Det ger 2^3 = 8 möjliga färgkombinationer enligt tabell 3. Även signalerna h-sync och v-sync är anslutna till var sin pinne i kontakten.

Tabell 3: Färgkombinationer

Röd	Grön	Blå	Färg
0	0	0	Svart
0	0	1	Blå
0	1	0	Grön
0	1	1	Turkos
1	0	0	Röd
1	0	1	Magenta
1	1	0	Gul
1	1	1	Vit

Skrivning till bilden sker via ett bildminne där varje pixel i bilden motsvaras av en minnesadress med tre bitar data för de tre färgerna. Bildminnet är tillgängligt via vga-komponentens minnesadress i tabell 2. Vga-komponenten läser ur minnet och ritar bilden på skärmen genom att styra de fem vga-signalerna.

Bildens upplösning är 320 pixlar gånger 240 rader och bildminnet behöver därför innehålla 320*240 = 76800 rader med tre bitar data per rad. HAL-kommandon används för att skriva och läsa i bildminnet. Då gränssnittet mot bildskärmen i verkligheten arbetar med upplösningen 640*480 pixlar multipliceras adresserna med fyra enligt nedanstående exempel. Det första exemplet visar hur pixeln med koordinaterna x,y sätts till färgen rgb och det andra exemplet visar läsning av färgvärdet för pixeln.

```
IOWR_32DIRECT(IP_VGA_BASE, (x + y*320)*4, rgb);
rgb = IORD_32DIRECT(IP_VGA_BASE, (x + y*320)*4);
```

Vid upplösningen 320*240 kan x ha värden från 0 till 319 och y från 0 till 239.

9 Testprotokoll

Tabell 4: Testprotokoll för verifiering

Testfall	Beskrivning	Acceptanskrav	Verifiering
1	Tänder en pixel `print_pix(150,120,7);`	En vit pixel tänds.	OK

2	Läser en pixels färg
	`print_pix(150,120,7);`
	`rgb = read_pixel_ram_int(150,120);`
	`print_int(150,130,7,0,rgb,1);`
		En vit pixel tänds. Siffran 7 visas under pixeln.	OK

3	Ritar en horisontell linje `print_hline(150,120,100,7);`	En vit horisontell linje som är 100 pixlar lång ritas.	OK
4	Ritar en vertikal linje `print_vline(150,120,100,7);`	En vit vertikal linje som är 100 pixlar lång ritas.	OK
5	Rensar bilden `clear_screen(1);`	Hela bilden blir blå.	OK
6	Skriver ett tecken `print_char(150,120,7,0,'M');`	Bokstaven M skrivs med vit färg.	OK
7	Skriver en teckensträng `print_str(150,120,7,0,"LMN");`	Teckensträngen LMN skrivs med vit färg.	OK
8	Ritar en cirkel `print_circle(10,150,120,7);`	En fylld vit cirkel ritas.	OK

Tabell 5: Testprotokoll för validering

Testfall	Beskrivning	Acceptanskrav	Validering
1	Uppstart	Först visas en presentation av systemet.	OK
2	Loggning	Y - axeln visar 0 till 2000 X - axeln visar 0 till 53 sekunder Displayen visar "REAL TIME", "Freq 5 Hz" och rimligt värde för ljusintensitet.	OK
3	Uppdatering av diagram	Diagrammets graf visar lästa A/D-värden. Grafen scrollar åt höger och suddar inte ut bakgrunden.	OK
4	Reset (SW1 trycks ned)	Grafen rensas och loggningen börjar om.	OK
5	Växling till visning av medelvärde (SW2 trycks ned)	Displaytexten "REAL TIME" byts till "AVG" följt av ett tal som visar index för senast inlästa värde. Grafen uppdateras när 60 värden lästs in.	OK
6	Växling till visning av värden i realtid (SW2 trycks ned igen)	Visningsläget återgår till realtidsläge.	OK
7	Byte av samplingsfrekvens (SW3 trycks ned)	Samplingsfrekvens byts varje gång knappen trycks ned. Uppdatering av grafen sker med den aktuella frekvensen. Tillgängliga frekvenser är 5, 2, 1 och 0.1 Hz. X-axelns värden uppdateras med rätt tider vid varje ändring.	OK

10 Konstruktionsbeskrivning

Konstruktionen är uppdelad i fyra filer:

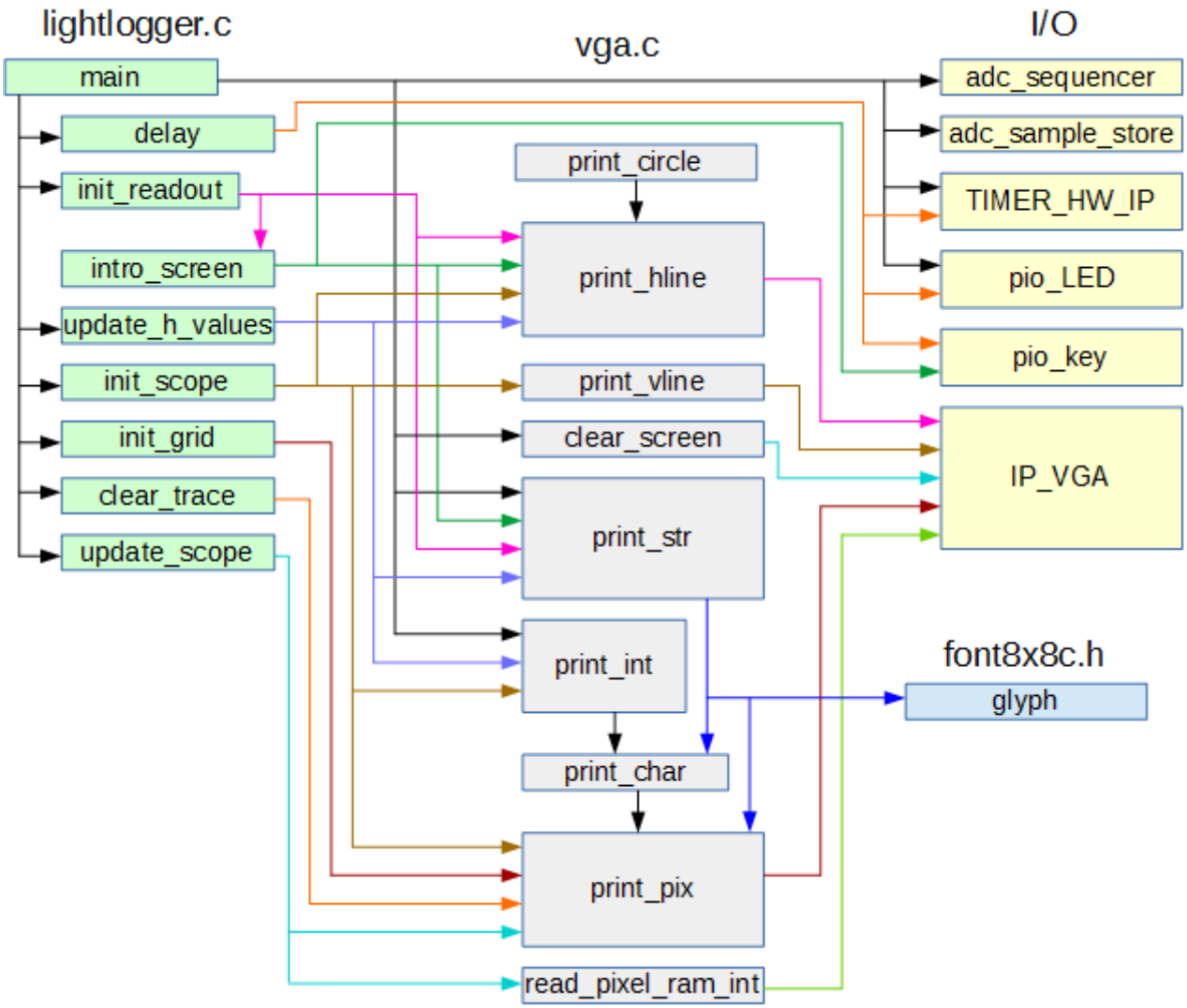
lightlogger.c är huvudprogrammet.

vga.c innehåller vga-rutinerna.

vga.h innehåller funktionsdeklarationer och inkluderas för att få tillgång till vga-rutinerna.

font8x8c.h innehåller teckenmatriser för de tecken som används av vga-rutinerna.

Figur 4 visar en översiktsbild över systemets arkitektur. Huvudprogrammet innehåller main och de funktioner som hanterar loggning av data, tidräkning och bildhantering. Funktionerna i vga.c används av huvudprogrammet för att utföra de grundläggande grafikfunktionerna som till exempel att tända pixlar, rita linjer och skriva tecken på skärmen. Grafikrutinerna använder i sin tur vga-utgångens IP-komponent till att skriva och läsa i grafikminnet. De övriga I/O-komponenterna används av huvudprogrammet till att kommunicera med A/D-omvandlaren, timern, lysdioderna och knapparna.

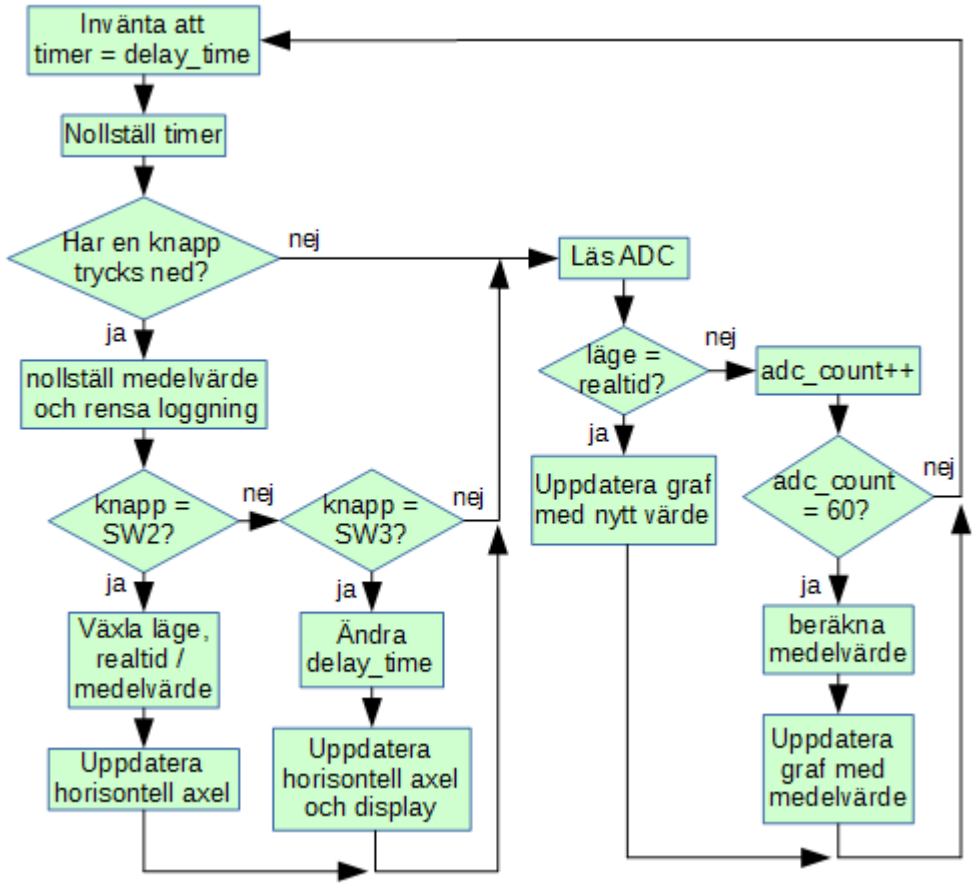


Figur 4: Systemarkitektur

10.1 lightlogger.c

Filen lightlogger.c innehåller alla programmets funktioner utom vga-rutinerna. Det första som sker efter start är att lysdioderna släcks samt att A/D-omvandlaren, timern och ett antal variabler initieras. Initieringen fortsätter med att loggningsdiagrammets axlar ritas med funktionen init_scope. Under diagrammet finns en display som visar visningsläge, samplingsfrekvens och senast lästa värde. Displayen initieras i funktionen init_readout som även anropar intro_screen innan displayens värden skrivs ut. Intro_screen visar en introduktionstext som presenterar systemet. Efter att användaren tryckt på en knapp slutförs initieringan med att den horisontella axelns värden skrivs av update_h_values, grafminnet initieras av clear_trace, diagrammets bakgrund ritas av init_grid och till sist nollställs timern.

Efter initieringen börjar huvudloopen som fungerar enligt flödesdiagrammet i figur 5. Det första som sker varje varv är att funktionen delay anropas för att invänta att en period av aktuell samplingsfrekvens förflutit. Ingången för kortets knappar pollas under väntetiden och om en knapp tryckts ned returneras knappens värde när funktionen avslutas. En av kortets lysdioder är tänd under en kort tid varje gång delayfunktionen anropas.



Figur 5: Main

Om returvärdet från delayfunktionen visar att en knapp tryckts ned sker funktionsval enligt knapptryckningen när exekveringen av main fortsätter. Loggningen nollställs om SW1 tryckts ned och visningsläge växlar mellan realtid och medelvärde om SW2 tryckts ned. Om SW3 tryckts ned byts samplingsfrekvens genom att stegning sker mellan frekvenserna 5 / 2 / 1 och 0,1 Hz.

Nästa steg är att läsa in ett nytt värde från A/D-omvandlaren. Diagrammet kan antingen visa värden i realtid med en pixel per läst värde eller som medelvärden av 60 inlästa värden. När aktuellt visningsläge är realtidsuppdatering sker uppdatering av grafen direkt med det nya värdet. När medelvärden visas räknas ett nytt värde fram om 60 värden har lästs sedan förra uppdateringen. Funktionen update_scope sköter uppdateringen av grafen som scrollas en pixel åt höger för varje nytt värde.

10.1.1 Funktioner

`void intro_screen()`
Skriver presentation av systemet och väntar på att SW1 trycks ned.

`void init_readout()`
Skapar displayen i bildens nedre del.

`void update_h_values(alt_u32 delay_time, char *htext)`
Uppdaterar den horisontala axelns värden enligt delay_time och skriver tidsenheten htext på skalan.

`void init_grid()`
Ritar loggningsdiagrammets bakgrund.

`void init_scope(alt_u32 delay_time)`
Ritar loggningsdiagrammets båda axlar med skalor och skriver värden vid den vertikalen skalan.

`void update_scope(alt_u16 adc)`
Uppdaterar loggningsdiagrammets graf med det ett nytt värdet från adc.

`void clear_trace()`
Raderar loggningsdiagrammets graf och rensar grafminnet.

`alt_u8 delay(alt_u32 delay_time, alt_u32 timer_offset)`
Väntar ett antal klockcykler enligt delay_time samt läser knapptryckningar och blinkar med en lysdiod.

`int main()`
Main innehåller huvudloopen och anropar övriga funktioner samt styr val enligt inmatning och läser nya värden från A/D-omvandlaren.

10.2 vga.c

Filen vga.c innehåller de vga-rutiner som beställts. Eftersom huvudprogrammet behöver en funktion som skriver ett heltal på skärmen har även funktionen print_int lagts till i filen. Funktionerna som ritar linjer och pixlar kontrollerar att skrivning utanför bilden inte sker. När övriga funktioner skriver i bilden används någon av funktionerna för pixlar eller linjer och därmed blir kontrollen utförd även för övriga funktioner. I filen vga.h definieras värden för X_MAX och Y_MAX. Felkontrollen utförs genom att koordinater jämförs med dessa värden. Antal pixlar per rad har begränsats genom att X_MAX har definierats till 317 på grund av att endast hälften av pixel 318 syns i bilden och pixel 319 hamnar i bildens vänstra kant i stället för till höger.

Funktionen print_circle används inte av huvudprogrammet, men finns med eftersom den ingår i beställningen.

10.2.1 Funktioner

```
void print_pix(alt_u16 x, alt_u16 y, alt_u8 rgb);
```
Tänder en pixel med färgen rgb på positionen x,y.

```
alt_u8 read_pixel_ram_int(alt_u16 x, alt_u16 y);
```
Läser färgvärdet för en pixel på positionen x,y. Vid första läsning från videominnet efter att skrivning utförts blir resultatet det senast skrivna färgvärdet oavsett adress. Läsning utförs därför två gånger för varje tillfälle funktionen anropas.

```
void print_hline(alt_u16 x_start, alt_u16 y_start, alt_u16 len, alt_u8 rgb);
```
Ritar en horisontell linje med startpositionen x,y och längden len samt färgen rgb.

```
void print_vline(alt_u16 x_start, alt_u16 y_start, alt_u16 len, alt_u8 rgb);
```
Ritar en vertikal linje med startpositionen x,y och längden len samt färgen rgb.

```
void clear_screen(alt_u8 rgb);
```
Rensar skärmen genom att sätta alla pixlar till färgen rgb.

```
void print_char(alt_u16 x, alt_u16 y, alt_u8 rgb, alt_u8 bg_rgb, char
character);
```
Skriver tecknet char på positionen x,y med färgen rgb och bakgrundsfärgen bg_rgb.

```
void print_str(alt_u16 x_start, alt_u16 y_start, alt_u8 rgb, alt_u8 bg_rgb, char
*str);
```
Skriver teckensträngen str med startpositionen x_start,y_start och färgen rgb samt bakgrundsfärgen bg_rgb. Funktionen avviker från kravspecifikationen avseende att parametern för bakgrundsfärg har lagts till.

```
void print_circle(alt_u16 radie, alt_u16 x_centrum, alt_u16 y_centrum, alt_u8
rgb);
```
Ritar en fylld cirkel med centrumpositionen x_centrum,y_centrum och radien radie samt färgen rgb.

```
void print_int(alt_u16 x_start, alt_u16 y_start, alt_u8 rgb, alt_u8 bg_rgb,
alt_u16 num, alt_u16 len);
```
Skriver en teckensträng med talet num och antal tecken enligt len på startpositionen x_start,y_start med färgen rgb och bakgrundsfärgen bg_rgb. Talet som skrivs ut kan som mest vara 16 bitar långt.

11 Verktygsinställningar

Ett antal inställningar har ändrats i BSP-editorn. Litet C-bibliotek, reducerade enhetsdrivrutiner och kompakt API för enhetsdrivrutiner har aktiverats. Stöd för C++, exit och clean exit har avaktiverats. Optimeringsnivå har ändrats till tre. Drivrutinerna för JTAG UART, accelerometer och Sierra har avaktiverats.

12 Verifiering

Vga-funktionerna har verifierats enligt testprotokollet i tabell 4.

12.1 Resultat från verifiering

Testfallen har kontrollerats genom att utföra testfallen och kontrollera resultat på en vga-skärm.

12.1.1 Testfall 1

Testfall 1 är att tända en vit pixel:

```
print_pix(150,120,7);
```

Figur 6 visar att testfallet är godkänt.

Figur 6: Pixel

12.1.2 Testfall 2

Testfall 2 är att tända en pixel, läsa färgvärdet för pixeln och skriva värdet på skärmen:

```
print_pix(150,120,7);
rgb = read_pixel_ram_int(150,120);
print_int(150,130,7,0,rgb,1);
```

Figur 7 visar att testfallet är godkänt.

Figur 7: Färgvärde

12.1.3 Testfall 3

Testfall 3 är att rita en horisontell linje:

```
print_hline(150,120,100,7);
```

Figur 8 visar att testfallet är godkänt.

Figur 8: Horisontell linje

12.1.4 Testfall 4

Testfall 4 är att rita en vertikal linje:

```
print_vline(150,120,100,7);
```

Figur 9 visar att testfallet är godkänt.

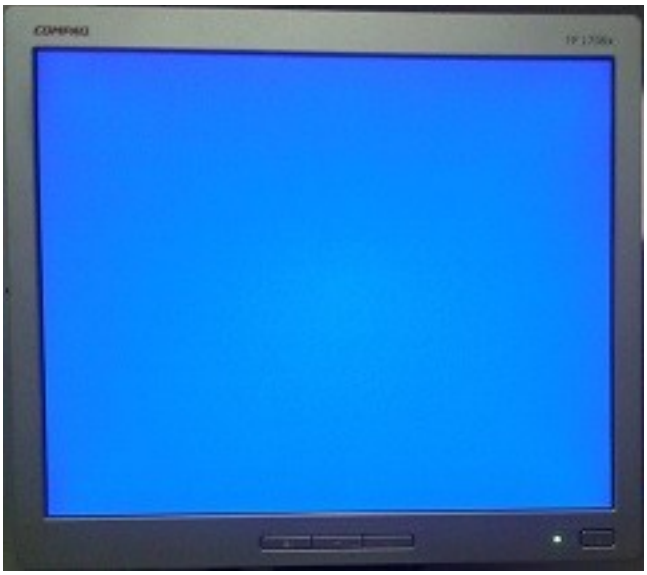
Figur 9: Vertikal linje

12.1.5 Testfall 5

Testfall 5 är att rensa skärmen med blå bakgrund:

```
clear_screen(1);
```

Figur 10 visar att testfallet är godkänt.



Figur 10: Rensad skärm

12.1.6 Testfall 6

Testfall 6 att skriva bokstaven M:

```
print_char(150,120,7,0,'M');
```

Figur 11 visar att testfallet är godkänt.

Figur 11: Tecken

12.1.7 Testfall 7

Testfall 7 är skriva teckensträngen "LMN":

```
print_str(150,120,7,0,"LMN");
```

Figur 12 visar att testfallet är godkänt.

Figur 12: Teckensträng

12.1.8 Testfall 8

Testfall 8 är att rita en fylld cirkel:

```
print_circle(10,150,120,7);
```

Figur 13 visar att testfallet är godkänt.

Figur 13: Fylld cirkel

13 Analys

13.1 Varningar

Vid kompilering av projektet genereras en varning för A/D-omvandlarens drivrutin:

```
drivers/src/altera_modular_adc.c: In function 'altera_modular_adc_init':
drivers/src/altera_modular_adc.c:131:14: warning: passing argument 3 of
'alt_irq_register' from incompatible pointer type
              alt_irq_register(irq, dev, alt_adc_irq);
              ^
```

Bedömningen är att det går bra att bortse från varningen eftersom A/D-omvandlaren fungerar och varningen genereras även för ett nytt "Hello World" med samma hårdvara.

13.2 Optimering

Programmets storlek har optimerats enligt tabell 6. Optimeringen utfördes i fyra steg och programmets storlek minskade från 86676 bytes till 23572 bytes.

Steg 1: Reduced device drivers och Small C library aktiverades. Support C++ avaktiverades.

Steg 2: Optimization Level 3 valdes.

Steg 3: Enable exit och Enable clean exit avaktiverades. Enable lightweight device driver api aktiverades.

Steg 4 Drivrutiner för JTAG UART, SPI Accelerometer och Sierra avaktiverades.

Tabell 6: Optimering av storlek

Minnestyp	Ej reducerad kod	Steg 1	Steg 2	Steg 3	Steg 4
text	77600	30616	30616	25312	22580
data	6960	796	796	312	76
bss	2116	2044	2044	2044	916
total	86676	33456	33456	27668	23572

Tider har mätts för att kontrollera att tillräckliga marginaler finns. Mätningen utfördes genom att lägga till en rad i funktionen delay:

```
printf("%d\n", TIMER_READ); // Print time
```

Utskriften visar hur lång tid som har förflutit under ett varv i huvudloopen och utförs just innan delayfunktionen väntar på att en period av inställd fördröjningstid ska ta slut:

```
while((TIMER_READ + timer_offset) < delay_time) // While not delay timeout...
```

Mätningarna visar att ett varv i huvudloopen som mest tar c:a 58 millisekunder vilket bedöms vara tillräckligt då den kortaste periodtiden är 200 millisekunder.

14 Validering

Konstruktionen har validerats enligt testprotokollet i tabell 5.

14.1 Resultat från validering

Validering har utförts genom att köra systemet som det är tänkt att användas. Resultatet redovisas genom avsnittets bilder och en demonstrationsfilm:

https://youtu.be/vCOzzwmwfKs

14.1.1 Testfall 1

Efter uppstart ska en presentation av systemet visas. Figur 14 visar presentationsbilden.

Figur 14: Presentationsbild

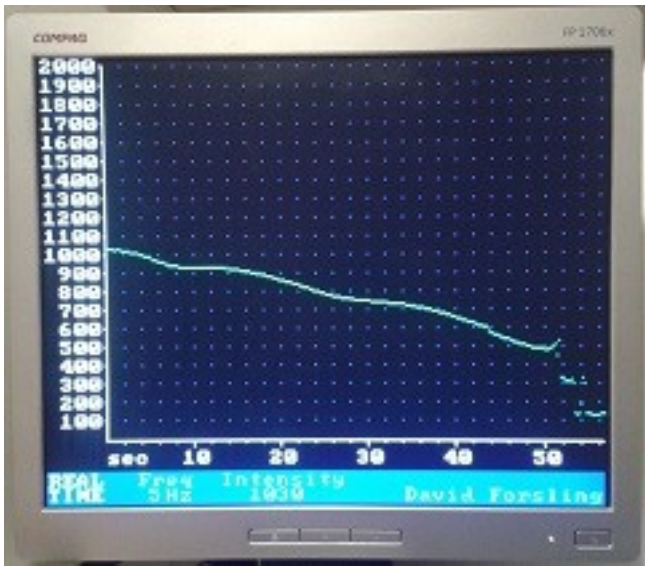
14.1.2 Testfall 2

När loggningen startas ska Y-axeln visa 0 till 2000 och X-axeln ska visa 0 till 53 sekunder. Displayen ska visa "REAL TIME", "Freq 5 Hz" och rimligt värde för ljusintensitet. Figur 15 visar bildskärmens utseende när loggningen just har påbörjats.

Figur 15: Påbörjad loggning

14.1.3 Testfall 3

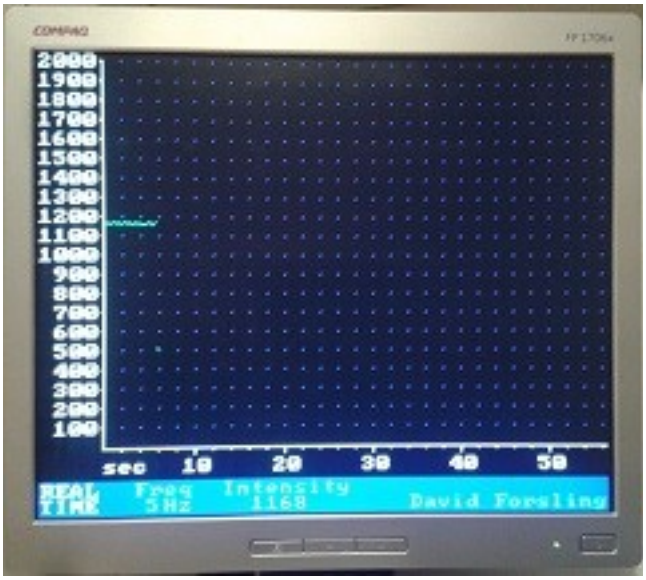
Diagrammets graf ska visa lästa A/D-värden. Grafen ska scrolla åt höger utan att sudda ut bakgrunden. Figur 16 visar att grafen har scrollats till höger jämfört med föregående bild.



Figur 16: Grafen

14.1.4 Testfall 4

När SW1 trycks ned ska grafen rensas och loggningen ska börja om. I figur 17 visas bildskärmens utseende efter att SW1 tryckts ned.



Figur 17: Efter reset

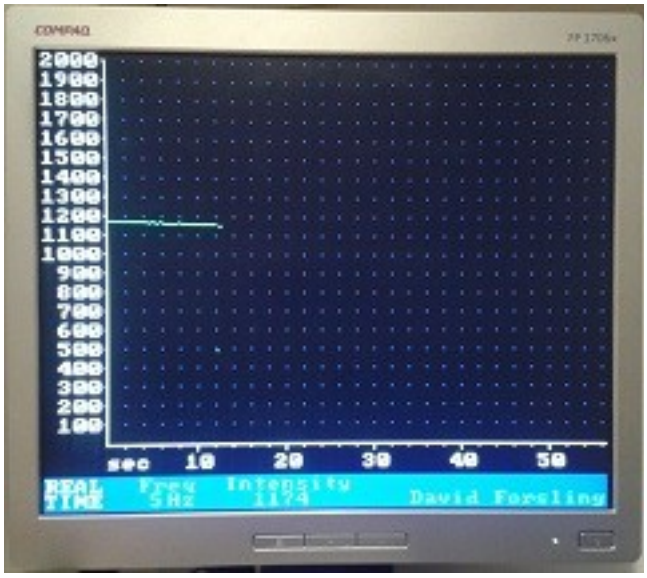
14.1.5 Testfall 5

När SW2 trycks ska växling ske till visning av medelvärden. Displaytexten "REAL TIME" ska bytas till "AVG" följt av ett tal motsvarande index för senast inlästa värde och grafen ska uppdateras när 60 värden lästs in. Figur 18 visar bildskärmens utseende efter att SW2 tryckts ned.

Figur 18: Medelvärde

14.1.6 Testfall 6

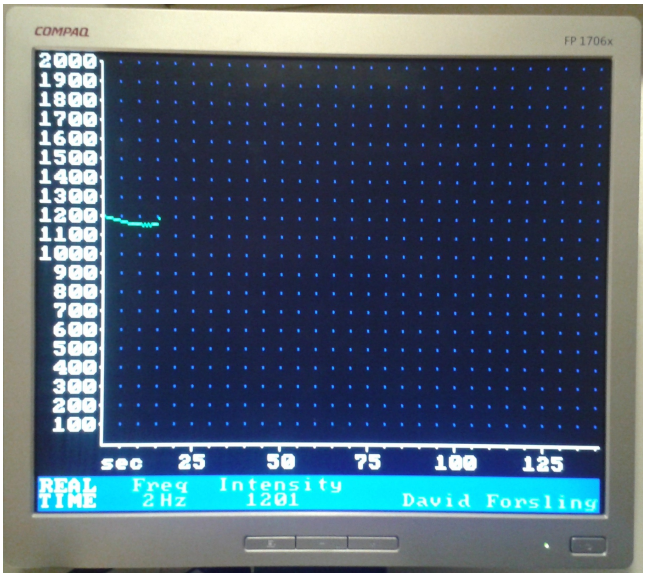
När SW2 trycks ned för andra gången återgår visningsläge till visning av värden i realtid. Figur 19 visar bildskärmens utseende efter att SW2 tryckts ned för andra gången.



Figur 19: Realtidsläge

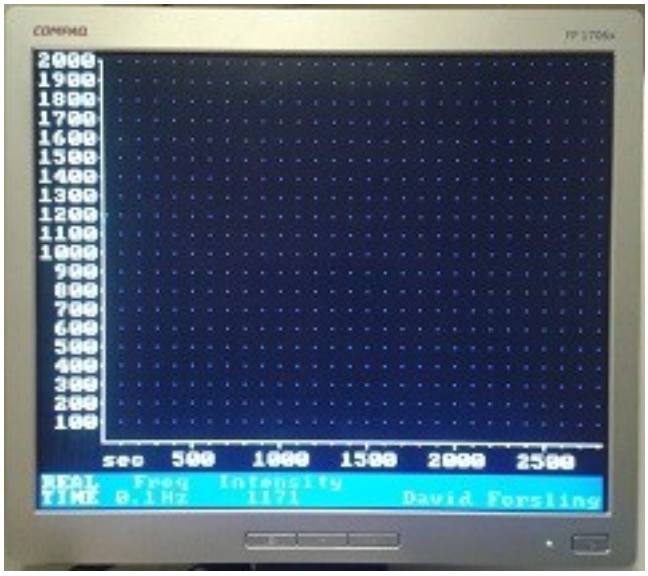
14.1.7 Testfall 7

Samplingsfrekvens ska bytas varje gång SW3 trycks ned. Stegning ska ske mellan frekvenserna 5, 2, 1 och 0,1 Hz. X-axelns värden ska uppdateras med rätt värden vid varje ändring. Figur 20 till 22 visar bildskärmens utseende efter växling mellan de olika frekvenserna.



Figur 20: 2 Hz

Figur 22: 1 Hz



Figur 21: 0,1 Hz

15 Sammanfattning

Systemet är en första prototyp och ett färdigt system för loggning bör kunna exportera data. En enkel lösning skulle kunna vara att skicka data till en PC via serieporten. Data skulle kunna lagras i SDRAM under loggningen och kopieras till PC:n vid behov.

En kalibrering av systemet behöver utföras för att få en riktig enhet för den uppmätta ljusstyrkan.

Snabbare grafikrutiner skulle förbättra det visuella intrycket av programmet. Funktionerna som skriver tecken och rensar bildskärmen upplevs som långsamma.

16 Referenser

Low Level C Programming for Desingers, 2015, Lars Bengtsson och Lennart Lindh

Arrow BeMicro Max 10 Getting Started User Guide, Version 14.0

http://www.helixsoft.nl/articles/circle/sincos.htm

https://github.com/dhepper/font8x8

5 i rad, Rapport skriven 2015 av Lars Högberg

Digitalt Labyrintspel, Rapport skriven 2013 av Patric Sjöberg

17 Bilagor

17.1 Bilaga A: Användarmanual för VGA-rutiner

Denna manual beskriver användning av nio grafikrutiner för vga-skärm i upplösningen 318 * 240 pixlar med åtta färger. De parametrar som anger färg vid nedanstående funktionsanrop är heltal mellan 0 och 7 med följande innebörd: 0 = svart, 1 = blå, 2 = grön, 3 = turkos, 4 = röd, 5 = magenta, 6 = gul och 7 = vit.

print_pix

Beskrivning

Tänder en pixel

Deklaration

```
void print_pix(alt_u16 x, alt_u16 y, alt_u8 rgb);
```

Argument

x Pixelns horisontella position

y Pixelns vertikala position

rgb Pixelns färg

Returvärde

Inget

Exempel

```
print_pix(150,120,7);
```

Exekveringstid för exempelkod

33,36 us

read_pixel_ram_int

Beskrivning

Läser färgvärdet för en pixel

Deklaration

```
alt_u8 read_pixel_ram_int(alt_u16 x, alt_u16 y);
```

Argument

x Pixelns horisontella position

y Pixelns vertikala position

Returvärde

rgb Pixelns färg

Exempel

```
rgb = read_pixel_ram_int(150,120);
```

Exekveringstid för exempelkod

52,48 us

print_hline

Beskrivning

Ritar en horisontell linje

Deklaration

```
void print_hline(alt_u16 x_start, alt_u16 y_start, alt_u16 len, alt_u8 rgb);
```

Argument

x_start	Linjens horisontella startposition
y_start	Linjens vertikala startposition
len	Linjens längd
rgb	Linjens färg

Returvärde

Inget

Exempel

```
print_hline(150,120,100,7);
```

Exekveringstid för exempelkod

2,74 ms

print_vline

Beskrivning

Ritar en vertikal linje

Deklaration

```
void print_vline(alt_u16 x_start, alt_u16 y_start, alt_u16 len, alt_u8 rgb);
```

Argument

x_start	Linjens horisontella startposition
y_start	Linjens vertikala startposition
len	Linjens längd
rgb	Linjens färg

Returvärde

Inget

Exempel

```
print_vline(150,120,100,7);
```

Exekveringstid för exempelkod

2,92 ms

clear_screen

Beskrivning

Rensar bilden

Deklaration

```
void clear_screen(alt_u8 rgb);
```

Argument

rgb Färg

Returvärde

Inget

Exempel

```
clear_screen(1);
```

Exekveringstid för exempelkod

534 ms

print_char

Beskrivning

Skriver ett tecken

Deklaration

```
void print_char(alt_u16 x, alt_u16 y, alt_u8 rgb, alt_u8 bg_rgb, char
character);
```

Argument

x	Horisontell position för tecknets översta vänstra hörn
y	Vertikal position för tecknets översta vänstra hörn
rgb	Tecknets färg
bg_rgb	Bakgrundsfärg
character	Tecknet som ska skrivas

Returvärde

Inget

Exempel

```
print_char(150,120,7,0,'M');
```

Exekveringstid för exempelkod

3,2 ms

print_str

Beskrivning

Skriver en teckensträng

Deklaration

```
void print_str(alt_u16 x_start, alt_u16 y_start, alt_u8 rgb, alt_u8 bg_rgb,
char *str);
```

Argument

x_start	Horisontell position för första tecknets översta vänstra hörn
y_start	Vertikal position för första tecknets översta vänstra hörn
rgb	Tecknens färg
bg_rgb	Bakgrundsfärg
str	Teckensträngen som ska skrivas

Returvärde

Inget

Exempel

```
print_str(150,120,7,0,"LMN");
```

Exekveringstid för exempelkod

9,63 ms

print_circle

Beskrivning

Ritar en fylld cirkel

Deklaration

```
void print_circle(alt_u16 radie, alt_u16 x_centrum, alt_u16 y_centrum,
alt_u8 rgb);
```

Argument

radie	Cirkelns radie
x_centrum	Cirkelns horisontella centrumposition

y_centrum	Cirkelns vertikala centrumposition
rgb	Cirkelns färg

Returvärde

Inget

Exempel

```
print_circle(10,150,120,7);
```

Exekveringstid för exempelkod

12,57 ms

print_int

Beskrivning

Skriver ett 16 bitars heltal

Deklaration

```
void print_int(alt_u16 x_start, alt_u16 y_start, alt_u8 rgb, alt_u8 bg_rgb,
alt_u16 num, alt_u16 len);
```

Argument

x_start	Horisontell position för första tecknets översta vänstra hörn
y_start	Vertikal position för första tecknets översta vänstra hörn
rgb	Tecknens färg
bg_rgb	Bakgrundsfärg
num	Talet som ska skrivas
len	Antal tecken som ska skrivas

Returvärde

Inget

Exempel

```
print_int(150,120,7,0,100,3);
```

Exekveringstid för exempelkod

9,47 ms