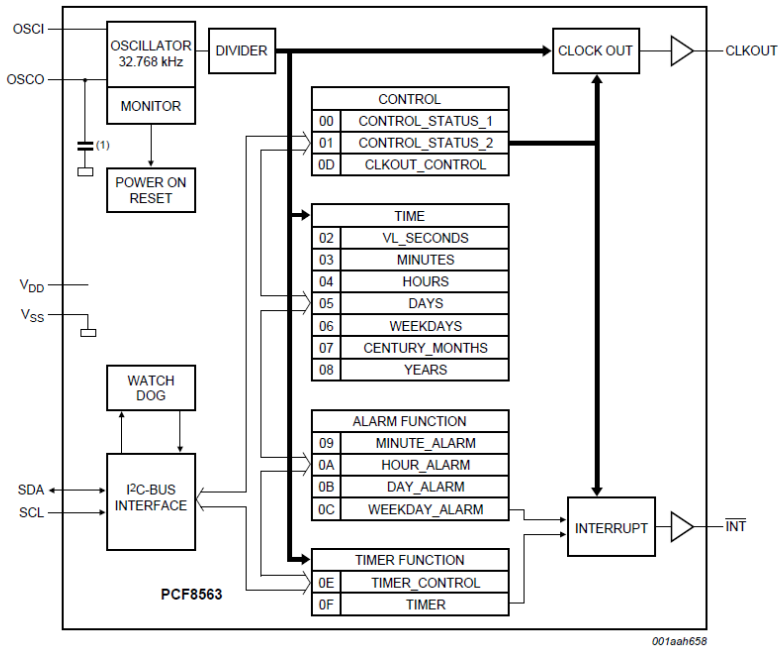
TSEA29 Konstruktion Med Mikrodatorer

I^2C-laboration

2022-10-04

2022 version 1.0 (Olov Andersson)

6. Block diagram



(1) C_{OSCO}; values see Table 30.

Fig 1. Block diagram of PCF8563

Inledning

Detta är en laboration för att skapa förståelse för hur I²C -protokollet kan implementeras på mikrokontrollern ATmega16 [1]. Laborationen är anpassad för den som redan har arbetat lite med ATmega16, och är förtrogen med dess datablad. Tanken är att ge stöd inför kursens utvecklingsprojekt. ATmega16 har hårdvarustöd för att vara både master och slave på I²C, men i den här laborationen fokuseras bara på master. Steget till att implementera slave-funktionalitet är förhållandevis lågt. Den som vill vara väl förberedd kan läsa på i relevanta datablad inför laborationen. Även att läsa detta dokuments appendix kan rekommenderas som förberedelse.

Dokumentets upplägg:

1) Presentation av uppgiften.
2) Konkreta laborationsuppgifter.
3) Appendix om I²C-protokoll och introduktion till RTC.

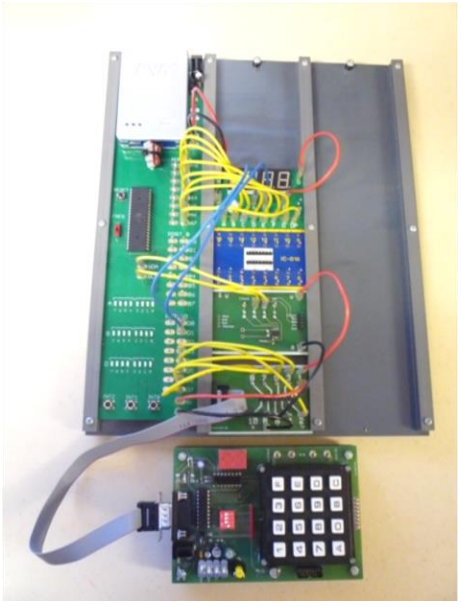
Syfte och mål

Syftet med laborationen är att göra det lättare att komma igång med I²C-kommunikation. Syftet är också att repetera samt bygga vidare på kunskaper från den inledande Mät-laborationen.
Efter genomförd laboration förväntas det en ökad förståelse för:

- Hur I²C-protokollet fungerar
- Hur I²C-hårdvaran i ATmega16 kan användas

Uppgiften i stort

I tidigare kurser så som datorteknik, TSEA82/TSEA57, har bland annat ett digitalur konstruerats. Tanken är nu att bygga vidare på detta koncept och införa en krets, PCF8563 [2]. Kretsen innehåller en fristående realtidsklocka (eng. ”RTC”) med alarmfunktionalitet. Vi har valt denna för att kommunikationsprotokollet med kretsen följer I²C-standarden. En separat krets av den här typen erbjuder en möjlighet att strömförsörja RTC-kretsen separat så att klockan ska kunna gå även om resten av systemet stängs av. Vi kommer att utnyttja kretsen på så vis att vi hämtar sekund- och minut-information med en sekunds intervall och lägger ut detta på den ena displayen. Intervallet bestäms av en hårdvarutimer som finns i ATmega16. I uppgiften ingår också ett hextangentbord, med vars hjälp man ska kunna styra klockan enligt beskrivning på nästa sida:



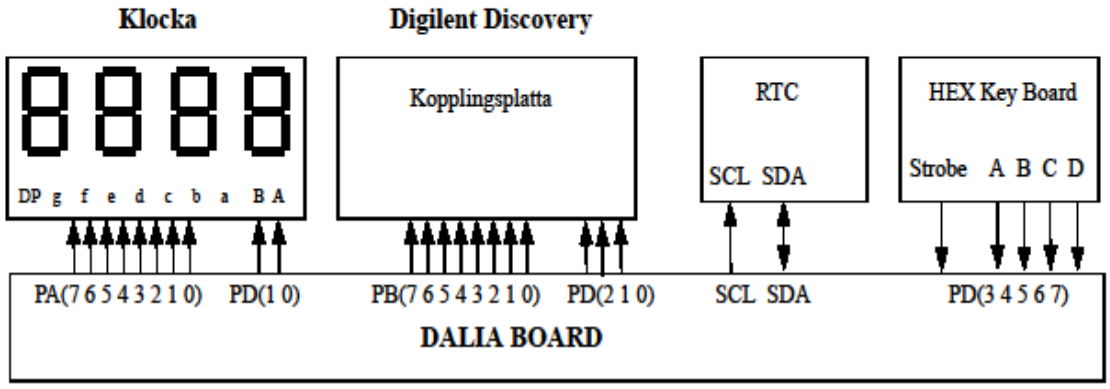
Grundfunktioner:

”C”: Gå till visningsläget (Decimalpunkten ligger mellan minuter och sekunder.)
”E”: Gå till ändringsläget (Decimalpunkten utgör markör.)

Funktioner i ändringsläget:

”D”: Spara tiden i RTC och gå till visningsläget
”F”: Flytta fram markören utan att ändra tiden
Siffrorna 0 – 9: Ändra siffran och flytta fram markören. Observera att enbart 0-5 ska accepteras som tiotalssiffra.

Kopplingsschema:



Laborationsskal

Till den här laborationen finns det ett laborationsskal som innehåller den grundfunktionalitet som behövs. Laborationsskalet innehåller också kod som underlättar mätuppgifterna. Laborationsskalet finns att hämta på kursens websida.

I grundfunktionaliteten ingår:

- Att hämta tiden en gång per sekund (ungefär) från RTC
- Att presentera tiden på en multiplexad 4x7-segmentsdisplay
- Att ta emot data från ett hextangentbord och avkoda knapparna enligt specifikation.

Laborationsskalet är uppdelat i följande moduler:

Main: Main är den del av programmet som körs när inga avbrottsrutiner körs. Förutom initiering av de övriga modulerna så finns här en huvudloop. Huvudloopen gör ingenting eftersom all funktionalitet ligger i ett antal avbrottsrutiner.

Display: Display-modulen innehåller funktioner för att uppdatera den multiplexade 4x7-segmentsdisplayen. Jobbet utförs av en avbrottsrutin, `ISR(TIMER0_COMP_vect)`, som triggas i 1 kHz. Detta ger en total uppdateringsfrekvens för displayen på 250 Hz. Siffrorna finns lagrade i `DISP_value[n]`, där n=0 för längst till höger och n=3 för längst till vänster. Variabeln `DISP_mux` håller reda på vilken siffra som just för tillfället uppdateras enligt algoritmen för multiplexning.

RTC: RTC-Modulen innehåller funktioner för att kommunicera med RTC. Dessa funktioner anropar i sin tur funktioner i modulen I2C eftersom RTC:n ansluts via en I²C-buss.

I2C: I2C-modulen innehåller funktioner för att genomföra kommunikation enligt I²C-standarden.

De funktioner som exporteras är:

- `I2C_RTC_read(adr, n)` **:** Hämtar n antal bytes från RTC med start från adress adr i RTC:ns interna minne. Resultatet hamnar i `I2C_read_buffer`.
- `I2C_RTC_write(adr, n)` **:** Skriver n antal bytes till RTC med start från adress adr i RTC:ns interna minne. Datan hämtas från `I2C_write_buffer`.

Några globala variabler som finns är:

- `I2C_write_buffer[I2C_BUFF_SIZE]` och `I2C_read_buffer[I2C_BUFF_SIZE]` : Två databuffertar som används vid sändning och mottagning.
- `I2C_is_write_mode` : En flagga som indikerar pågående sändning. Nollställs när sändningen är avslutad.
- `I2C_is_read_mode` : En flagga som indikerar pågående mottagning. Nollställs när mottagningen är avslutad.

Uppgifter del A: Multiplexad Display

Lite större displayer som ska presentera information, och som ska observeras av ett mänskligt öga, brukar låta informationen vara multiplexad. Det går då åt färre signalutgångar från en processor vilket kan vara nödvändigt i mikrodatortillämpningar. I den här applikationen är det de decimala siffrorna som skickas ut en i taget tillsammans med en adress som pekar ut på vilken 7-segmentsdisplay som just den aktuella siffran ska visas. Tempot för detta bestäms med hjälp av en intern timer som genererar ett avbrott med ett lämpligt tidsintervall. För att åstadkomma en flimmerfri visning behöver displayen uppdateras med minst 60 Hz. Eftersom en display består av fyra siffror ska alltså timern ställas in på minst 240 Hz.

Laborationsuppgift

1) Skapa ett nytt projekt på X: och välj "GCC C Executable Project".
2) Ladda ner filen: `I2C_skal.c` och lägg in den i projektet.
3) Koppla upp enligt kopplingsschemat samt slå på strömmen.
4) Kompilera laborationsskalet, och kör på ATmega16.

Klockan ska nu börja att gå och undersök gärna att knapparna hos tangentbordet fungerar enligt uppgiftens specifikation.

Mätuppgifter

Koppla in och mät på signalerna som skickas till 7-segmentsdisplayen. Eftersom kodskelettet lägger ut samma information på PORTA som på PORTB så går det bra att använda den föreslagna uppkopplingen, dvs PD(2-0) & PB(7-0) ansluts till logikanalysatorn med hjälp av en kopplingsplatta. Anslut också SCL, SDA och Gnd på RTC-modulen till logikanalysatorn. Den här uppkopplingen kan användas igenom hela laborationen.

Segmentinformationen läggs ut enligt principen: släck segmenten, välj display och tänd segmenten.

- Mät upp under hur lång tid som alla segmenten är släckta.
- Mät också upp frekvensen, dvs hur ofta som man väljer en ny display. (Enligt kodskelettet är detta approximativt 1 kHz.)
- Om ni vill Jämföra segmentinformationen med vad det borde vara så kan det underlätta om ni först trycker på knappen ”E”.

Avbrottsrutinen som hanterar multiplexningen lägger också ut en spårsignal på PD(2). På detta sätt kan tiden som programmet ligger i avbrottsrutinen mätas.

- Mät upp hur många % av CPU-tiden som avbrottsrutinen tar.
- Ändra muxningsfrekvensen till ca 2 kHz genom att konfigurera om Timer-0. I funktionen DISP_init() kan ni ändra på antingen TCCR0 eller OCR0 eller både och, se databladet över ATMega16 för detaljer.
- Mät återigen upp hur många % av CPU-tiden som avbrottsrutinen tar.

Skillnaden bör bli ganska liten eftersom avbrottsrutinen är kort.

Uppgifter del B: I²C-bussen (pollad)

I databladet för RTC [2, Avsnitt 9.5] finns tre olika kommunikationsmoder beskrivna och vi ska fokusera på två av dem.

Förberedelseuppgift

Läs i [2, kapitel 9] om I²C-kommunikationen, samt [2, avsnitt 8.2 – 8.4], om hur 8-bitars registren för sekunder och minuter är formaterade.

I²C-hårdvaran i ATmega16

ATmega16 är utrustad med dedikerad hårdvara för att hantera I²C, med multimaster och så kallad arbitrering (att reda ut situationen om flera enheter börjar agera master precis samtidigt). I det givna kodskelettet är den enklaste varianten redan implementerad, dvs den där en while-loop används för att vänta på TWINT-flaggan. Ca en gång/sekund hämtas tiden från RTC och det finns ett timer-avbrott, från Timer-1, som initierar och genomför hämtningen. Se funktionen `ISR(TIMER1_CAPT_vect)` i koden.

Mätuppgifter

Använd logikanalysatorn för att mäta upp en transaktion på I²C-bussen.

- Hur lång tid tar en transaktion, dvs mät tiden mellan start-villkor och stopp-villkor på bussen.
- Undersök den information som skickas på I²C-bussen. Jämför med den kommunikationsmod som används till/från RTC.
- Ändra i koden så att timmar, minuter och sekunder hämtas. Visserligen kommer fortfarande bara minuter och sekunder att visas på displayen, men med logikanalysatorn går det att se all information som överförs.
- Sätt tiden till 59:55 och verifiera med logikanalysatorn att timmar räknas upp med ett.

Tittar ni lite mer noga på informationen från logikanalysatorn så går det att se att en I²C-transaktion stör multiplexningen av displayen. Den här störningen är så pass kortvarig så att ögat inte märker detta, men med logikanalysatorns hjälp kan ni se att den trots allt finns där. Orsaken är att transaktionen görs i en avbrottsrutin, och då är alla andra avbrott blockerade. Det kan åtgärdas genom att tillåta avbrott även i en avbrottsrutin.

- Fixa detta och studera resultatet med logikanalysatorn.

En förutsättning för att detta ska fungera är att det inte finns någon konflikt mellan det data som hämtas via I²C och det som visas på displayen. Kodskelettet är skrivet så att det är först när hela I²C-transaktion är avslutad som klockinformationen flyttas till displayen.

- Skulle ögat upptäcka om det blev någon konflikt vid överföringen av information till displayen?

Mätuppgifter

Nu ska busskommunikationens prestanda mätas. För att underlätta det är kodskelettet förberett med något som kallas för villkorlig kompilering. Det går alltså väldigt enkelt att ställa om så att en annan del av koden kompileras. I början av kodskelettet finns en rad som ser ut så här:

```
//#define I2C_DEBUG
```

ändra den raden till:

```
#define I2C_DEBUG
```

Resultatet är dels att utsignalen PD(2) nu i stället innehåller en spårsignal som gör att det nu går att mäta upp hur lång tid som er I²C-transaktion befinner sig i while-looparna. Resultatet innebär också att den information som skickas till PORTB ändras så att vi nu även kan läsa av värdet på statusregistret (TWSR). För det används de fem högsta bitarna PB(7-3), varför de tre minsta bitarna kan användas till att lägga ut en siffra (0-7) och som blir unik för varje while-loop i I²C-transaktionen. Gå gärna in i kodskelettet och titta på hur detta är implementerat.

Nu kan ni:

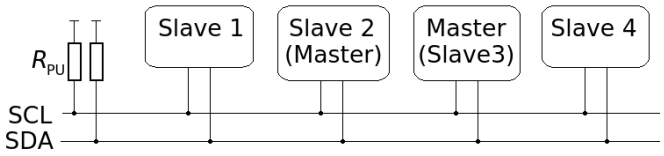
- Mäta tiden för varje while-loop som ingår i I²C-transaktionen. (Kommer att vara olika för olika delar av transaktionen.)
- Summera dessa tider och få fram en %-sats på hur mycket CPU-tid som slösas bort under I²C-transaktionen.

While-looparna i den här implementationen av I²C-transaktionen syftar till att vänta på en TWINT, dvs på att en avbrottsflagga ska bli satt. Ett smartare sätt att implementera detta på är att använda en avbrottsrutin, men en sådan implementation ingår inte i den här laborationen . Noterbart är att omedelbart efter varje while-loop så läses ett statusregister (TWSR) av, och med ett tränat öga går det att se att det förväntade värdet är olika på olika ställen i I²C-transaktionen. TWSR innehåller alltså information om var i I²C-protokollet som transaktionen befinner sig, och det är detta som utnyttjas vid en avbrotsstyrd implementation. I databladet för ATMega16 [1, kapitel: ”Two-wire Serial Interface”, sid 172-200] finns det omfattande dokumentation till er hjälp.

- Använd logikanalysatorn för att jämföra statusinformationen från TWSR, med vad det borde vara enligt databladet för AVR. Det finns några trevliga tabeller att studera där det framgår vad varje statuskod betyder.

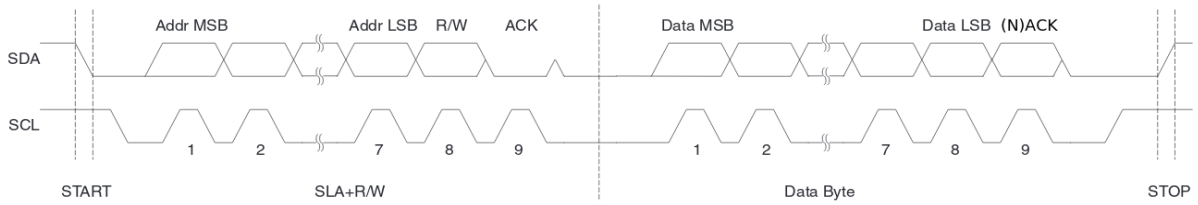
Appendix: I²C-protokollet

I²C är ett kommunikationsprotokoll från Philips. Av upphovsrättsskäl används ofta namnet TWI (two wire interface). Det finns beskrivet i ATmega16's datablad [1, kapitel: "Two-wire Serial Interface"].



- I²C är en tvåtrådsbuss. Den har seriella klocka (SCL) och seriell data (SDA).
- Upp till runt 100 enheter kan anslutas till samma I²C-buss (7-bitars adress, och ett antal reserverade adresser ger ca 100 adresser kvar att utnyttja).
- Kommunikationen initieras av en master och adresserar en slav.
- Det kan finnas flera mastrar, men bara en kan vara aktiv åt gången. För detta krävs att anslutna mastrar har multimaster-stöd. Ofta används dock bara en master, och en eller flera slavar.
- Master adresserar önskad slav i läs- eller skriv-läge, och därefter följer en eller flera bytes i ett paket. Data kan alltså skickas antingen slav till master eller master till slav. Man pratar därför om sändare (transmitter) och mottagare (receiver).
- De två trådarna är "öppen kollektor", d.v.s. de dras mot hög (etta) av resistorer. Vilken enhet som helst kan sedan tvinga dem låga, genom att ansluta dem till jord. Protokollet innehåller strikta regler för hur detta får göras.

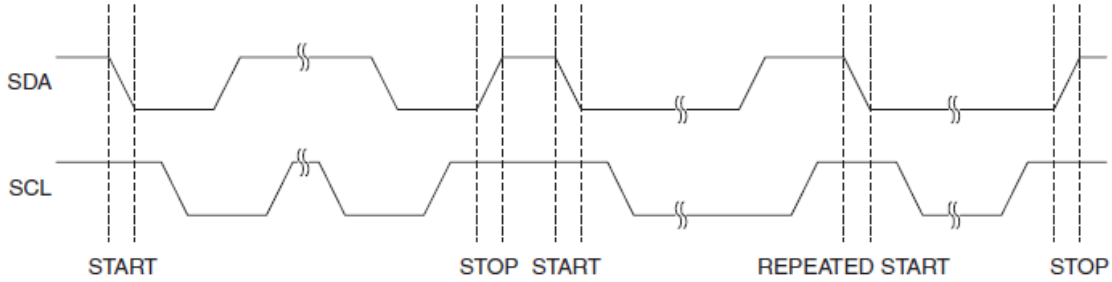
När bussen inte används ligger båda trådarna höga (d.v.s. ingen skriver Noll på dem). Under överföring matar master klockpulser på SCL. Sändaren uppdaterar SDA när SCL är låg, och mottagaren läser SDA när SCL är hög. Efter varje överförd byte bekräftar mottagaren genom att lägga ut en nolla, vilket kallas för A=ACK. Om linan inte är låg då, så ska sändaren tolka detta som en A=NACK (not ACK).



Bilden illustrerar ett paket på I²C-bussen (i början är SCL=SDA=hög).

1. S = Startvillkor: Master initierar genom att sänka SDA till låg medan SCL är hög.
2. SLA+R/W: Master skickar sju slavadressbitar och en R/W-bit.
3. ACK: Slaven bekräftar genom att tvinga SDA till låg.
4. Sedan ett antal byte enligt:
    1. Data: Sändaren skickar ut en byte.
    2. ACK: Mottagaren bekräftar. I master read, så svarar master med NACK sista byten.
5. P = Stoppvillkor: Master avslutar paketet genom att dra SDA från låg till hög när SCL är hög.

Istället för 5. STOP, så kan Master välja att göra en repeated start (Sr eller Rs) genom att lägga på ett nytt startvillkor. Bilden på nästa sida illustrerar skillnaden mellan START, STOP och REPEATED START.



Efter repeated start, så fortsätter ett paket precis som efter en ordinarie start. Tack vare repeated start kan en master byta read/write-läge utan att behöva avsluta sändningen, och på så sätt garanteras att ingen annan enhet hinner blir master emellan.

Kommunikation med RTC

Följande tre exempel beskriver den typ av överföring som RTC-kretsen är designad för. Variant ett och tre är de som är implementerade i laborationsskalet:

Master skriver N byte:

S	SLA+W	A	Data1	A	…	Data*N*	A	P

Master läser N bytes:

S	SLA+R	A	Data1	A	…	Data*N*	A	P

Master skriver 1, läser N bytes:

S	SLA+W	A	Data1	A	Sr	SLA+R	A	Data1	A	…	Data*N*	A	P

Här är start- och stoppvillkoren gråmarkerade, och bitar som skickas av slaven är understrukna. RTC:ns registeruppsättning kan ses som ett litet SRAM om 16 bytes. Dessa bytes används till att innehålla tid, datum och alarminformation. Detta innebär att när man kommunicerar med RTC så måste man först skicka över en startadress, vilket förklarar behovet av variant tre ovan. Notera gärna att master svarar med NACK efter att sista byten har lästs. Slaven blir då förvarnad om att överföringen kommer att avslutas med ett stoppvillkor. Rent praktiskt går det att avsluta överföringen utan ett föregående NACK, men det blir snyggare att följa standarden.

Referenser:

[1] ATmega16 Datasheet
http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

[2] PCF8563 Real-time clock/calendar
https://www.nxp.com/docs/en/data-sheet/PCF8563.pdf