Pausevarsler

Prosjektoppgave i ING1507

Kandidatnr. xx

Kandidatnr. yy

1. Introduksjon

1.1. Prosjektbeskrivelse

Ved bruk av ATmega32 mikrokontrollere og en rekke komponenter, skulle vi ut ifra de ulike aspektene vi har lært gjennom kurset *Datamaskinarkitektur* lage en modul/system som passer i et smarthjem.

Innhold

2. Prosjektidé

2.1. Utfordring

På skolen har vi pause fra undervisningen med omtrent 45 minutter intervaller, dette er det en fast satt plan på når pausene skal komme. Det er kadett kommandør (KK) som har dette ansvaret, jobben er å følge med på klokken og varsle foredragsholder med 5 minutter før en pause starter. Utfordringen for KK er at hen glemmer å følge med på klokken da dette vil ta over for fokuset på det instruktøren sier. Som resulterer i at pausene ikke kommer når de skal, som da igjen gjør at kadettene sliter med å fokusere når de ikke får avbrekk.

2.2. Løsning

Løsningen vi har kommet opp med er å lage et produkt som varsler KK samt foredragsholderen når disse pausene skal komme, og hvor lenge det er til.

2.2.1. Virkemåte

Det er et system bestående av en kk-modul og en lærer-modul. Disse kommuniserer med hverandre over blåtann.

2.2.2. KK-modul

Dette er masteren. Den har en Real Time Clock, som er programmert med en alarm som sender et signal ved skolestart. Mikrokontrolleren har ekstern interrupt på dette signalet og vekkes ved dette signalet. Da går den over i å lese av klokken konstant. Når timen starter sender den lengden på timen over blåtann til lærer-modulen. Når pausen starter, sender den lengden på pausen til lærer-modulen. Når skoledagen er over, sender den stop-kommando til lærer-modulen. All konfiurering av skolestart, pauser og skoleslutt konfigureres altså på KK-modulen. KK-modulen har også en LCD som viser gjenværende minutter og sekunder av pausen når det er pause, og gjenværende tid av timen når det er time.

2.2.3. Lærer-modul

Dette er slaven. Denne har RX-interrupt fra blåtannmodulen. Den sover fram til den får melding over blåtann. Da viser den på 7-segment-skjermen gjenværende tid til time/pause, basert på lengden på pausen/timen den får tilsendt hvor den trekker fra tid ved bruk av den interne klokken i mikrokontrolleren. Når det er pause hever en servo et flagg for å tydeliggjøre at det er pause. Når pausen er over, senkes flagget.

3. Metode

Kobling ble gjort for de ulike komponentene for å kunne opprette funksjonaliteten beskrevet innledningsvis. Deretter ble en og en komponent testet og det ble laget kode for rett funksjonalitet for denne komponenten. Deretter ble disse kodeutsnittene satt sammen i en sammensatt kode for funksjonaliteten på kk-modulen og en sammensatt kode for lærer-modulen.

3.1. Utstyr

3.1.1. KK-modul

- 1x ATmega32 mikrokontroller
- 1x HM-10 blåtannmodul
- 1x DS3231 Real Time Clock
- 1x 16x2 LCD
- 2x koblingsbrett
- Flere ledninger og motstander

3.1.2. Lærer-modul

- 1x ATmega32 mikrokontroller
- 1x HM-10 blåtannmodul
- 1x 7-segment-skjerm
- 1x Servo
- 3x koblingsbrett
- Flere ledninger og motstander

4. Teori

4.1. USART

Blåtannmodulen kommuniserer med en baud-rate på 9600. Dette skaper et nøyaktighetsproblem på ATmega32 uten videre konfigurering. Normal konfigurasjon av USART på ATmega32 med standard klokkehastighet på 1MHz og baud rate på 9600, gir en baud-prescaler med −7% avvik fra en baud-verdi på 9600 (ref. Figure 1). Dette kommer av utregningen på baud-prescaler som gir et så lavt tall at nøyaktigheten blir for dårlig til å nøyaktig kunne nå baudraten på 9600 med en heltalls baud-prescaler. For å overkomme dette dobles USART-klokkehastigheten ved å sette U2X-bit i UCSRA. Da må også utregningen av prescaler dobles. Dette gir et avvik på bare 0.2% på baud-raten, noe som er pålitelig nok til å få lest av data korrekt i svært stor grad.

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 Kbps		125 Kbps		115.2 Kbps		230.4 Kbps		125 Kbps		250 Kbps	

1. UBRR = 0, Error = 0.0%

Figure 1: Avvik i baudrate på ATmega32 (ref. ATmega32 datablad [1])

4.2. HM-10 Blåtannmodul

En HM-10 blåtannmodul sender UART data over bluetooth. Den har en intern krets som tar seg alt prosessering ved overføring over bluetooth og fungerer med standardinnstillinger så snart den blir koblet opp til en spenning på 3-6v. Den kan konfigrureres ved hjelp av AT-kommandoer for å endre innstillinger som baud-rate, enhetsnavn, eller om den skal være master eller slave. Det er verdt å merke seg at den interne kretsen kun er 3.3v tolerant. Logikken i ATmega32 er 5v. Dette går fint for signalet fra TX på blåtannmodul til ATmega32, da ATmega32 kan regirstrere et signal på 3.3v. Problematikken er på signalet fra TX på ATmega32 til blåtannmodulen. Den indre kretsen i blåtannmodulen kan skades dersom dette signalet er 5v. Dette løses ved å bruke en spenningsdeler med ulike motstander (se koblingsskjema) på dette signalet fra TX på mikrokontrolleren.

4.3. DS3231 Real Time Clock

DS3231 er en ekstremt nøyaktig klokke som fungerer over I2C. Denne har et minnebatteri, som gjør at klokken fortsetter å gå rett selv om ekstern strøm til enheten kobles fra, dette gjør at den vil kunne vise rett tid i svært lang tid. Enheten har en krystall og en temperatur-kompensert krystall. RTC-en holder styr på sekunder, minutter, timer, ukedag, dato, måned og år. Enheten har også to programmerbare alarmer. For å hente ut tiden fra RTC senders først adressen til RTC hvor least significant bit (LSB) er satt til 0 over I2C fra masteren (ATmega32 er masteren i dette scenarioet). Dette gjør at RTC-en begynner, og er klar til å respondere på kommende meldinger. Deretter sender adressen vi ønsker å lese fra (“word address”) dette er adressen i Figure 2 (for eksempel adresse = 03h for å lese av dagen). Deretter sendes repeated start, etterfulgt av adressen til RTC-en hvor LSB er satt til 1. Dette gjør at RTC-en sender byten med data fra denne adressen. Dersom master så sender ACK sendes byten fra neste adresse osv. For å avslutte sendes NACK og Stop. Denne prosessen følger vi i Figure 3, og er implementert i Listing 1.

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Figure 2: Tidsregistere i DS3231 (ref. DS3231 datablad [2])

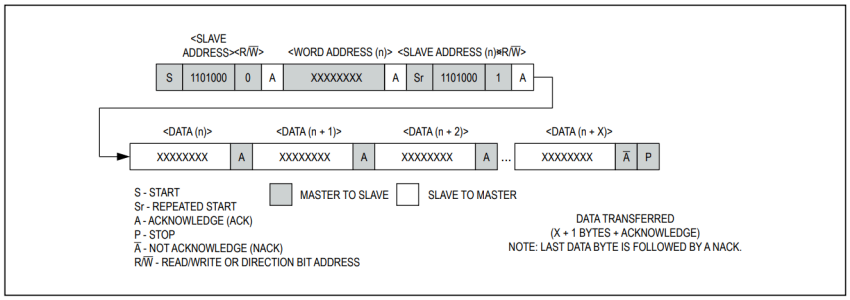

Figure 3: Prosess for avlesning DS3231

```
void RTC_Read_Clock(char read_clock_address) {
    I2C_Start(RTC_Write_address); // Start I2C communication with RTC
    I2C_Write(read_clock_address); // Write address to read
    I2C_Repeated_Start(RTC_Read_address);

    second = BDC2value(I2C_Read_Ack());
    minute = BDC2value(I2C_Read_Ack());
    hour = BDC2value(I2C_Read_Nack() & 0b00111111); //Last communication so nack.
The two MSB are not relevant
    I2C_Stop();
}
```

Listing 1: Kode for avlesning DS3231

4.4. 7-segment-skjerm

En 7-segment-skjerm er satt sammen av 7 led-lys med felles anode eller katode, hver av led-lysene har en egen pin. Hvis det er flere segmenter satt sammen vil det være en felles anode/katode for hvert segment og alle led-lysene for samme posisjon/samme bokstav er koblet sammen (ref. Figure 4). For å vise en bokstav på skjermen må de riktige pinnene settes høye eller lave.

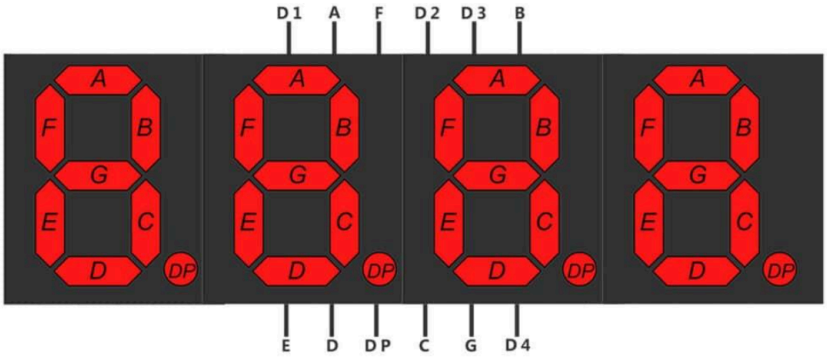

Figure 4: 7-segment-skjerm

5. Kodeprinsipper

5.1. KK-modul

5.1.1. Alarm på DS3231

Utklipp hentet fra databladet til DS3231 [2]

For prosjektet skal ATmega32 vekkes ved alarmen på DS3231 til kl.08.00. Alarmen settes ved å skrive til alarm-registrene (ref. Figure 2). For å få alarmen til å trigge på sekunder, minutter og timer, settes A1M4 i DS3231-register til 1 (ref. Figure 5) For å oppnå at når alarmen trigges settes SQW-pinnen til GND, slås Interrupt Control på ved å sette INTCN til 1. For at den skal trigge på alarm 1, slås også A1IE på. (ref. Figure 6). Alarm-initieringen er implementert i Listing 2. Setting av alarm er implementert i Listing 3. Når flagget for alarmen blir satt, vekkes ATmega32 og går over i å lese av klokken konstant. Flagget må nullstilles manuelt, og gjøres ved å sette bit-en for A1F i Figure 7 til 0. Dette er implementert i Listing 4.

DY/$\overline{DT}$	ALARM 1 REGISTER MASK BITS (BIT 7)				ALARM RATE
	A1M4	A1M3	A1M2	A1M1	
X	1	1	1	1	Alarm once per second
X	1	1	1	0	Alarm when seconds match
X	1	1	0	0	Alarm when minutes and seconds match
X	1	0	0	0	Alarm when hours, minutes, and seconds match
0	0	0	0	0	Alarm when date, hours, minutes, and seconds match
1	0	0	0	0	Alarm when day, hours, minutes, and seconds match

Figure 5: Konfigurering av alarm på DS3231

Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

Figure 6: Register for å konfigurere blant annet alarm på DS3231

```
    void RTC_Alarm_Init(){
        I2C_Start(RTC_Write_address);
        I2C_Write(0xE);    // Bet at the control register position (See datasheet)
        I2C_Write(0b00000101);  // Enabling interrupt Control and Alarm 1
Interrupt Enable
        I2C_Stop();
}
```

Listing 2: Kode for å initiere alarm på DS3231

```
void RTC_Alarm1_Time(char _hour, char _minute, char _second){ // The alarm
triggers at the spesified time every day.
    I2C_Start(RTC_Write_address);
    I2C_Write(7);         // Set alarm1 time in register 7
    I2C_Write(value2BDC(_second));
    I2C_Write(value2BDC(_minute));
    I2C_Write(value2BDC(_hour));
    I2C_Write(0b10000000); // Set A1M4 to trigger at that time set
    I2C_Stop();
}
```

Listing 3: Kode for å sette alarm1

Status Register (0Fh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	OSF	0	0	0	EN32kHz	BSY	A2F	A1F
POR:	1	0	0	0	1	X	X	X

Figure 7: Register for å lese av / nullstille flagg

```
void RTC_Alarm_Clear(){
    I2C_Start(RTC_Write_address);
    I2C_Write(0xF);
    I2C_Write(0b10001000);  // Write 0 at the BSY, A2F and A1F flag.
    I2C_Stop();
}
}
```

Listing 4: Kode for å nullstille alarm-flagget

6. Diskusjon

6.1. Blåtannmodul

Konfigurering av modulen var utfordrende, da det er vanskelig å vite nøyaktig hvordan modulen opererer, særlig med lite dokumentasjon. Slike blåtannmoduler er også moduler det er finnes utallige kopier av der ulike produsenter har ulike standarder. Dette gjør det spesielt vanskelig å finne hvilken type modul med hvilken konfigurasjon man har, og deretter å finne adekvat dokumentasjon for denne. (Utseende på en HC-06, HC-05 og HM-10-modul er tilnærmet identisk, noe som gjør det vanskelig å identifisere hvilken man har). I vårt tilfelle fant vi med mye leting ut at våres modul var av typen HM-10, de var imidlertid ikke helt etter standard, for flere av AT-kommandoene som var å finne i ufullstendig dokumentasjon på internett returnerte våres blåtann moduler med “Error”. Tross dette var vi i stand til å endre rolle mellom slave og master. I vårt prosjekt ble blåtannmodulen brukt ved KK-modulen satt til å være master, og lærer-modulen slave. Etter det vi kunne finne av dokumentasjon på internett om automatisk oppkobling i slave-master-par fungerte ingen av kommandoene for å sette opp at master-modulen automatisk skulle koble seg opp til slave-modulen ved oppstart. Vi måtte derfor konkludere med at med mangel på rett dokumentasjon for våre blåtannmoduler var ikke automatisk sammenkobling ved oppstart mulig. Dette resulterte i at man ved oppstart må koble master modulen til seriell tilbobling til PC, for å via AT-kommandoer søke etter nære blåtann-moduler og velge rett, for så å koble opp til denne.

6.2. Virkemåte KK-modul

Det er hensiktsmessig at masteren har mest mulig nøyaktig tid, da Lærer-modulen settes etter denne. Det kunne blitt brukt interne timere i KK-modulen, og lest av klokken DS3231 med relativt lange intervallpauser, noe som kunne spart energi. Det ble alikevel besluttet at ettersom KK-modulen ikke

kjøres fra et batteri, ville det være mer hensiktsmessig å heller lese av klokken kontinuelig, for å få mest mulig korrekt tid. Noen optimaliseringer ble imidlertid implementert: 1. Alarmen på DS3231 settes til kl.08.00. Slik at når tidspunktet er 08.00 settes SQW-pinnen på DS3231 til GND. KK-modulen har dette signalet som et interrupt-signal, slik at den er i sleep fram til starten av dagen. Deretter leser den kontinuelig av klokken over I2C. Dette gjør den med bruk av while mens den sender og venter på ACK, grunnen til dette er at vi er nødt til å ha kontroll på hvor vi er i kommunikasjonsprosessen, da vi først sender adressen vi leser fra, for så å lese av et bestemt antall verdier, der vi får neste verdi ved å sende ACK og må ha kontroll på når vi skal sende NACK og STOP når all klokkedata er mottatt. Ved pause-/timestart senders lengden på pausen eller timen over USART. Denne USART-kommunikasjonen gjøres gjennom interrupts for å ikke blokkere hovedprogrammet. Dette gjør at KK-modulen kan fortsette å lese av klokken mens den sender data over USART. Dette gjør at KK-modulen har mest mulig korrekt tid, og kan sende data til Lærer-modulen på riktig tidspunkt.

7. Feilkilder

USART på ATmega32 ble konfigurert til å bruke dobbel hastighet. Dette gjorde at baud-prescaler fikk en verdi som ga den 0.2% avvik ved baud-rate på 9600. Selv om avviket er lite, er det ikke neglisjerbart. Dette kombinert med eventuelle feil under overføring med blåtann, gjør at det er en viss mulighet for at data som sendes fra KK-modulen blir mottatt feil på Lærer-modulen, og at denne derfor vil vise feil gjenværende tid.

Referanser

[1] Atmel Corporation, “ATmega32/L Datasheet.” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf

[2] Analog Devices, “DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal.” [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/DS3231.pdf