Oblig 3 - Halvdupleksprotokoll

Fredrik Sandhei, Mathias Haukås [1]

1. mai 2017

[1]UiT - AUT-1001, obligatorisk innlevering 3

Innhold

Introduksjon . . . 2
Hensikt . . . 2
Teori . . . 2
    Protokoll . . . 2
    USART og SPI . . . 2
    Interrupts og Timer2 . . . 2
Løsning . . . 3
    Implementasjon . . . 3
Diskusjon . . . 3
Konklusjon . . . 3

Introduksjon

I samsvar med arbeidskravene til faget Programmering med mikrokontrollere", trengs det tre arbeidskrav, i form av tre obligatoriske innleveringer, godkjent. I dette dokumentet ligger vår besvarelse på den siste obligatoriske innleveringen - oblig 3.

Hensikt

Teori

Protokoll

Protokollen som skulle brukes i oppgaven er "halv-dupleks En datapakke sendes først fra datamaskinen til mikrokontrolleren ved hjelp RS-232 - kommunikasjon. Deretter forventer protokollen en transmit fra mikrokontrolleren. Dataen som sendes og mottas er forventet å være i ASCII-format. Hver Rx - prosedyre er kontrollert ved hjelp av rekkefølgen på pakkeelementene: Hver pakke begynner på en bokstav, enten A, B, C eller D etterfulgt med line feed, $\backslash n$. Når en Rx-prosedyre er gjennomført, avsluttes pakken med R og $\backslash n$, og protokollen er klar for å motta en Tx-prosedyre, som ikke er like strengt lagt opp. Den forventer igjen det samme som i Rx, men rekkefølgen spiller ingen rolle, og ingen R trengs ikke for å konkludere transmit.

USART og SPI

For å opprette kommunikasjon mellom PC og displaykortet er USART - Universal Synchronous Asynchronous Receive Transmit - et nyttig verktøy. Det er en type seriell kommuikasjon, der dataelementene/bitsene shiftes over en kabel mellom enhetene. Ved hjelp av en ekstra kabel kan en bruke en seriell klokke mellom enhetene for å synkronisere dataregistreringen hos mottakeren. I vårt tilfelle benyttes ikke den serielle klokken. Kommunikasjonen blir dermed asynkron - UART. Asynkron USART bruker et start-bit og et (evt. to) stopp-bit i tillegg til den dataen som sendes for å bestemme overføringshastigheten. UART-kommunikasjonen åpner for bruk av 'peripherene' på displaykortet. Kommunikasjonen mellom ATmega644A og de ulike komponentene kalles SPI. Serial Peripheral Interface er seriell kommunikasjon over korte avstander", og benyttes for å kommunisere mellom flere enheter og mikrokontrolleren. ATmega644 fungerer som master, og periferene fungerer som slaver. Dataoverføring ved SPI fungerer som en loop: Data blir sendt fra master til slave via MOSI samtidig som data mottas fra slave til master via MISO. På denne måten kan slaven og master veksle informasjon på en enkelt kommunikasjonslinje. SPI-kommunikasjon ble brukt mellom følgende enheter:

- LCD - display: Liquid Crystal Display - display
- LED - lys
- Potensialmeter
- Piezo-buzzer
- ISP-connector

Interrupts og Timer2

Flere av de periferene som ble brukt ble brukt i sammenheng med en interrupt. En interrupt, eller ISR - Interrupt Service Routine - 'forstyrrer' arbeidsflyten til prosessoren for å gjennomføre et sett med oppgaver implementert av programmereren til denne ISR-rutinen. Da settes alt annet på mikrokontrolleren på vent, og hovedprogrammet fortsettes etter at det nødvendige flagget fra interruptets opphavsregister registreres høyt. I dette tilfellet bruktes interrupts for UART-kommunikasjon, Timer2 - output-compare-mode - non-pwm-mode, samt ADC.

Løsning

Implementasjon

Til vår løsning av problemstillingen benyttet vi oss av interrupts til å behandle informasjon mottatt fra PCen og til å sende informasjon. Tanken for vår løsning var å holde mikrokontrolleren og PCen i en konstant kommunikasjonsloop i samsvar med protokollen: Data blir mottatt, og data blir sendt. Når noe ble mottatt, gikk RXC - flagget høyt $UCSR0A$ (USART Control And Status Register A), og RXC-ISR-rutinen mottok informasjonen shiftet fra PC og mottatt i UDR-registeret i ASCII-format. Datainnholdet ble lagret i et globalt array som ble sjekket for kjennetegnene for endring av pakkeinformasjon, i dette tilfellet en bokstav fra 'A' til 'D' eller 'R' og line-feed. Informasjonen ble lagret i ulike arrays (A_array,B_array,C_array,D_array) for videre behandling. Da det siste pakkeelementet 'R' og linefeed var mottatt, ble tilstandsvariabelen $receive_done$ satt høyt.

Da de ulike dataene hadde ulike hensikter, var det passende å systematisere informasjonen i en felles behandlingsfunksjon, *received*() (linje 115 - 178) som hadde det nullte elementet i hver pakke som argument. Funksjonen baserte seg på en switch-case som enten printet en linje på LCD-displayet, regulerte LED-lysene eller aktiverte TIMER2-ISR for bruk av OCR. For tilfellene 'A' og 'B', var det simpelthen kalling av funksjonen *lcd_printline*() fra *serlcd.h* - biblioteket. LED-lysene ble regulert ved å tukle litt med AND-operatoren mellom GPIO-pinnene og C_array. 'D'-tilfellet ble litt mer arbeid, da det krevdes å gjøre elementene i D_array om til desimal - 1 enn ASCII-1. Hvert element ble trukket fra med '0' og multiplisert med enten 1000, 100 eller 10, basert på deres posisjon i arrayet.

Diskusjon

Konklusjon

```
1  /*
2  * oblig3.c
3  * OBLIG 3
4  * Gjennomf rt av Fredrik Sandhei og Mathias HaukÂs
5  *
6  */
7
8  #define F_CPU 14745600
9
10 //Inkluderinger av ulike bibliotek
11 #include <avr/io.h>
12 #include <avr/interrupt.h>
13 #include "serlcd.h"
14 #include <util/delay.h>
15
16 //Funksjonsprotyper
17 void received(char input);
18 void uart_send();
19
20 char A_array[18], B_array[18], C_array[7], D_array[7], C_to_send[6], D_to_send[4], A_to_send
       [8], B_to_send[22] = { "BDerp herp\n" }, send_array[50];
21 char rec_array[20];
22 volatile unsigned char rx_counter = 0, sendcounter = 1, receive_done, add;
23 unsigned int freq, pwm_on, voltage, adc_reader;
24
25 ISR(ADC_vect) {
26   adc_reader = ADCW;
27   voltage = 5000*(long)adc_reader/1024;
28 }
```

```
29
30 ISR(TIMER2_COMPA_vect)
31 {
32   OCR2A = OCR2A + add;
33 }
34
35
36 ISR(USART0_TX_vect) {
37   //Dersom det er noen elementer i index sendcounter, shift ut på UDR.
38   if (send_array[sendcounter]) {
39     UDR0 = send_array[sendcounter++];
40   }//Hvis ikke, tilbakestill sendcounter
41   else {
42     sendcounter = 1;
43   }
44
45 }
46
47
48 ISR(USART0_RX_vect) {
49   unsigned char uart = UDR0;
50   unsigned char i = 0;
51   if (uart == '\n') {
52     if (rec_array[0] == 'A')
53       for (i = 0; i<16; i++)
54         A_array[i] = rec_array[i + 1];
55     else if (rec_array[0] == 'B')
56       for (i = 0; i<16; i++)
57         B_array[i] = rec_array[i + 1];
58     else if (rec_array[0] == 'C')
59       for (i = 0; i<4; i++)
60         C_array[i] = rec_array[i + 1];
61     else if (rec_array[0] == 'D')
62       for (i = 0; i<5; i++)
63         D_array[i] = rec_array[i + 1];
64     else if (rec_array[0] == 'R')
65       receive_done = 1;
66     rx_counter = 0;
67   }
68   else
69     rec_array[rx_counter++] = uart;
70 }
71
72 int main(void)
73 {
74   init_lcd();
75   lcd_cursoron();
76   //Oppsett av porter, input/output
77   DDRB = 0xf0;
78   PORTB = 0x0f;
79   DDRC = 0x10;
80   PORTC = 0x0f;
81   DDRD = 0xf0;
82
83   //Timer 2 - konfigurasjon
84   TCCR2B = (1<<CS22);
85   TIMSK2 = (1 << OCIE2A);
86   //UART - konfigurasjon
87   //Rx enable, Tx enable, Rx interrupt enable, Tx interrupt enable
88   UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0) | (1 << TXCIE0);
89   //Bitrate 9600 - standard hastighet pÂ rs-232
90   UBRR0H = 0;
```

```
91    UBRR0L = 95;
92    //ADC - konfigurasjon
93    //Bruker ADC0 som input
94    ADMUX = 0x00;
95    //ADC enable, interrupt enable, clock speed = 115200Hz
96    ADCSRA = (1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
97
98    sei();
99    while (1)
100   {
101     //Mottar data f rst fra PC
102     //For sÂ Â sende data fra uK til PC
103     //ved hjelp av halv-dupleks-protokollen/Bernt-protokollen
104     if (receive_done)
105     {
106       received('A');
107         received('B');
108         received('C');
109       received('D');
110       uart_send();
111       receive_done = 0;
112     }
113   }
114 }
115 void received(char input) {
116   /*Behandler informasjonen mottatt
117   basert pÂ arrayet som mottas*/
118
119   switch (input)
120   {
121   case 'A':
122     lcd_printline(0, 0, A_array);
123     break;
124
125   case 'B':
126     lcd_printline(1, 0, B_array);
127     break;
128
129   case 'C':
130     if (0x0f & C_array[0])
131       PORTB |= (0x0f & C_array[0]) << 4;
132     else
133       PORTB &= ~(0x10);
134     if (0x0f & C_array[1])
135       PORTB |= (0x0f & C_array[1]) << 5;
136     else
137       PORTB &= ~(0x20);
138     if (0x0f & C_array[2])
139       PORTB |= (0x0f & C_array[2]) << 6;
140     else
141       PORTB &= ~(0x40);
142     if (0x0f & C_array[3])
143       PORTB |= (0x0f & C_array[3]) << 7;
144     else
145       PORTB &= ~(0x80);
146     break;
147
148   case 'D':
149
150     if (D_array[0] == '1')
151     {
152       //Dersom pwm-flagget ikke er h y -> sett h y.
```

```
153         //Gj r det slik at denne kodesnutten bare kj res
154         //en gang.
155         if(pwm_on == 0)
156         {
157           TCCR2A |= (1 << COM2A0);
158           pwm_on = 1;
159         }
160 
161         freq = (D_array[1] - '0') * 1000;
162         freq += (D_array[2] - '0') * 100;
163         freq += (D_array[3] - '0') * 10;
164         freq += (D_array[4] - '0');
165         add = (long)115200/freq;
166       }
167       else
168         if (pwm_on)
169         {
170           //Toggler av OC2A (Output Compare 2A)
171           TCCR2A &= ~(1 << COM2A0);
172           pwm_on = 0;
173         }
174       break;
175     default:
176       break;
177     }
178 }
179 
180 //Legger til datapakkene som skal sendes fra uK til sine respektive pakker
181 //og deretter legger de sammen til et array som sendes ved UART.
182 //Denne metoden gj r ISR-transmit liten og enkelt implementert.
183 
184 void uart_send() {
185   //Appending data for transmit
186   unsigned char lf = 0, sep_cntr = 0;
187 
188   //Legger inn pakke A til sending'
189   //Start conversion for ADC
190   ADCSRA ^= (1<<ADSC);
191 
192   A_to_send[0] = 'A';
193   A_to_send[1] = (voltage/1000 + 0x30);
194   A_to_send[2] = ((voltage%1000)/100 + 0x30);
195   A_to_send[3] = ((voltage%100)/100 + 0x30);
196   A_to_send[4] = ((voltage%10) + 0x30);
197   A_to_send[5] = 'm';
198   A_to_send[6] = 'V';
199   A_to_send[7] = '\n';
200 
201   //Legger inn pakke C til sending
202 
203   C_to_send[0] = 'C';
204   for (unsigned char i = 1; i < 5; i++)
205   {
206     if ((PINB&(1 << (i - 1))) == 0)
207       C_to_send[i] = '1';
208     else
209       C_to_send[i] = '0';
210   }
211   C_to_send[5] = '\n';
212 
213 
214 //Legger inn pakke D til sending
```

```
215   char address_val = 0x30;
216   if(PINC&0x01)
217     address_val += 8;
218   if(PINC&0x02)
219     address_val += 4;
220   if(PINC&0x04)
221     address_val += 2;
222   if(PINC&0x08)
223     address_val += 1;
224
225   if(address_val >= 57)
226     address_val += 7;
227
228   D_to_send[0] = 'D';
229   D_to_send[1] = address_val;
230   D_to_send[2] = '\n';
231
232   //Legger alt sammen i et monster - array. Big time
233   for (unsigned char i = 0; i < 50; i++)
234   {
235     if (lf == 0)
236     {
237       send_array[i] = A_to_send[sep_cntr];
238       if (A_to_send[sep_cntr] == '\n')
239       {
240         //Dersom linefeed - inkrementer lf, nullstill sep_cntr og hoppe til neste iterasjon
241         lf = 1;
242         sep_cntr = 0;
243         continue;
244       }
245     }
246     if (lf == 1)
247     {
248       send_array[i] = B_to_send[sep_cntr];
249       if (B_to_send[sep_cntr] == '\n')
250       {
251         //Dersom linefeed - inkrementer lf, nullstill sep_cntr og hoppe til neste iterasjon
252         lf = 2;
253         sep_cntr = 0;
254         continue;
255       }
256     }
257     if (lf == 2)
258     {
259       send_array[i] = C_to_send[sep_cntr];
260       if (C_to_send[sep_cntr] == '\n')
261       {
262         //Dersom linefeed - inkrementer lf, nullstill sep_cntr og hoppe til neste iterasjon
263         lf = 3;
264         sep_cntr = 0;
265         continue;
266       }
267     }
268
269     if (lf == 3)
270     {
271       send_array[i] = D_to_send[sep_cntr];
272       if (D_to_send[sep_cntr] == '\n')
273       {
274         //Dersom linefeed - inkrementer lf, nullstill sep_cntr og hoppe til neste iterasjon
275         lf = 4;
276         sep_cntr = 0;
```

```
277             continue;
278           }
279       }
280       sep_cntr++;
281     }
282     //Begynner transmit.
283     //Dette vil f re til at TXC-flagget blir h y ,
284     //og en TXC ISR vil gjennomf res.
285     UDR0 = send_array[0];
286 }
```