Laboratory 07: Asynchronous Serially Controlled Digital Potentiometer with SPI Interface

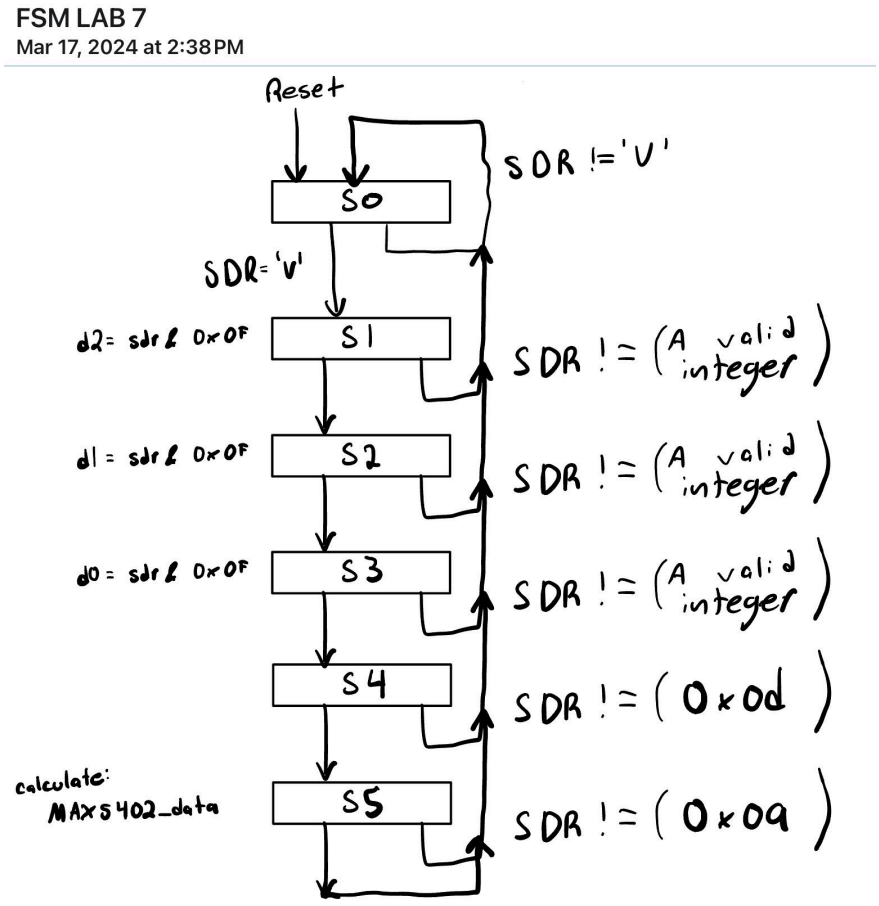
CHRISTOPHER NIELSEN + CHRISTOPHER SHAMAH
KENNETH SHORT
Bench No: 17
ID#114211318
ID#112229076
Lab Section L01

DT1 : program verification strategy as part of your prelab.

Our program verification strategy will be to first set breakpoints along the SPI0 init chain, ensuring that all necessary registers are filled out as proper, our voltage levels are correct for each step of the MOSI,CLK,CS and such, and then ensure that the data being transferred to the max5402 is correct, as well as track the voltage output with our oscilloscope to see a smooth even staircase function.

- ☑ ~~DT2~~

DT3 : Submit a state diagram for FSM case statements.



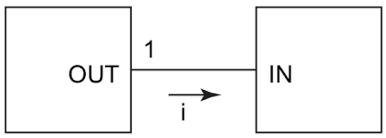
1)

381 lab 7 interface
Mar 11, 2024 at 6:30 PM

Interface Checklist (revised 10/09/17)
ESE280 Ken Short

VDD = 3V B VDD = 5

Avr128 Max6402



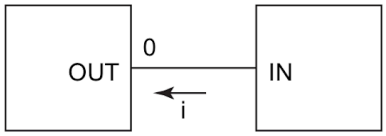
$V_{OHmin(A)} > V_{IHmin(B)}$

VDD-.7 > .7×VDD

$I_{OHmax(A)} > I_{IHmax(B)}$

6 mA > ±1.0 uA

Avr128 Max6402



$V_{OLmax(A)} < V_{ILmax(B)}$

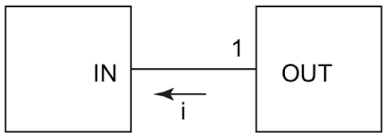
.6 V < .3×VDD

$I_{OLmax(A)} > I_{ILmax(B)}$

10 mA > ±1.0 uA

not used for lab 7

Avr128 Max6402



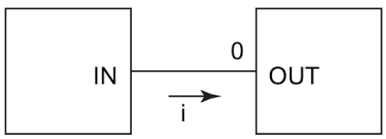
$V_{IHmin(A)} < V_{OHmin(B)}$

.7×VDD Not listed

$I_{IHmax(A)} < I_{OHmax(B)}$

5nA < Not listed

Avr128 Max6402



$V_{ILmax(A)} > V_{OLmax(B)}$

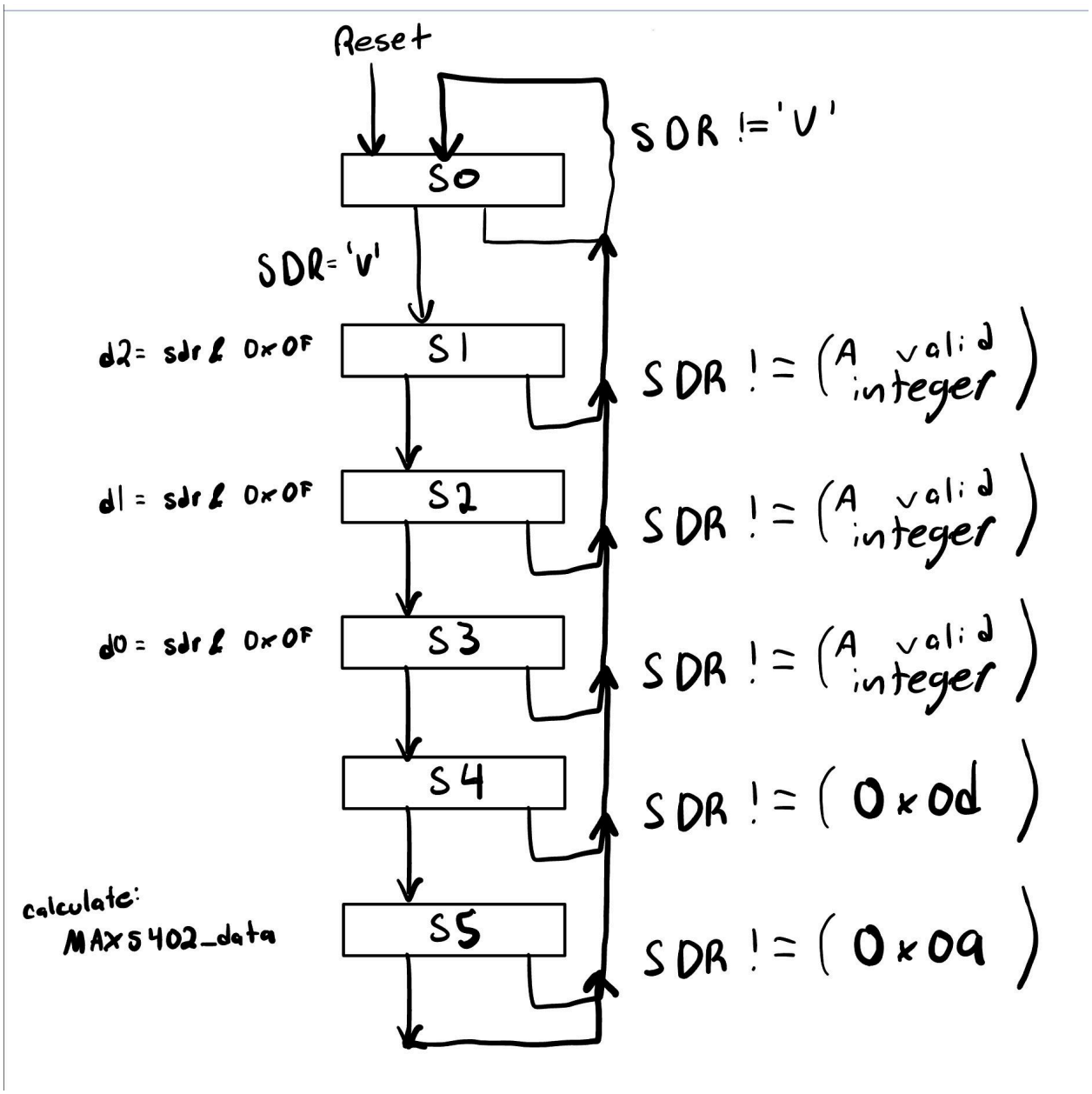
.3×VDD Not listed

$I_{ILmax(A)} < I_{OLmax(B)}$

5nA < Not listed

2) the max SCLK frequency is 1/100ns which is limited by the MAX5402 Datasheet

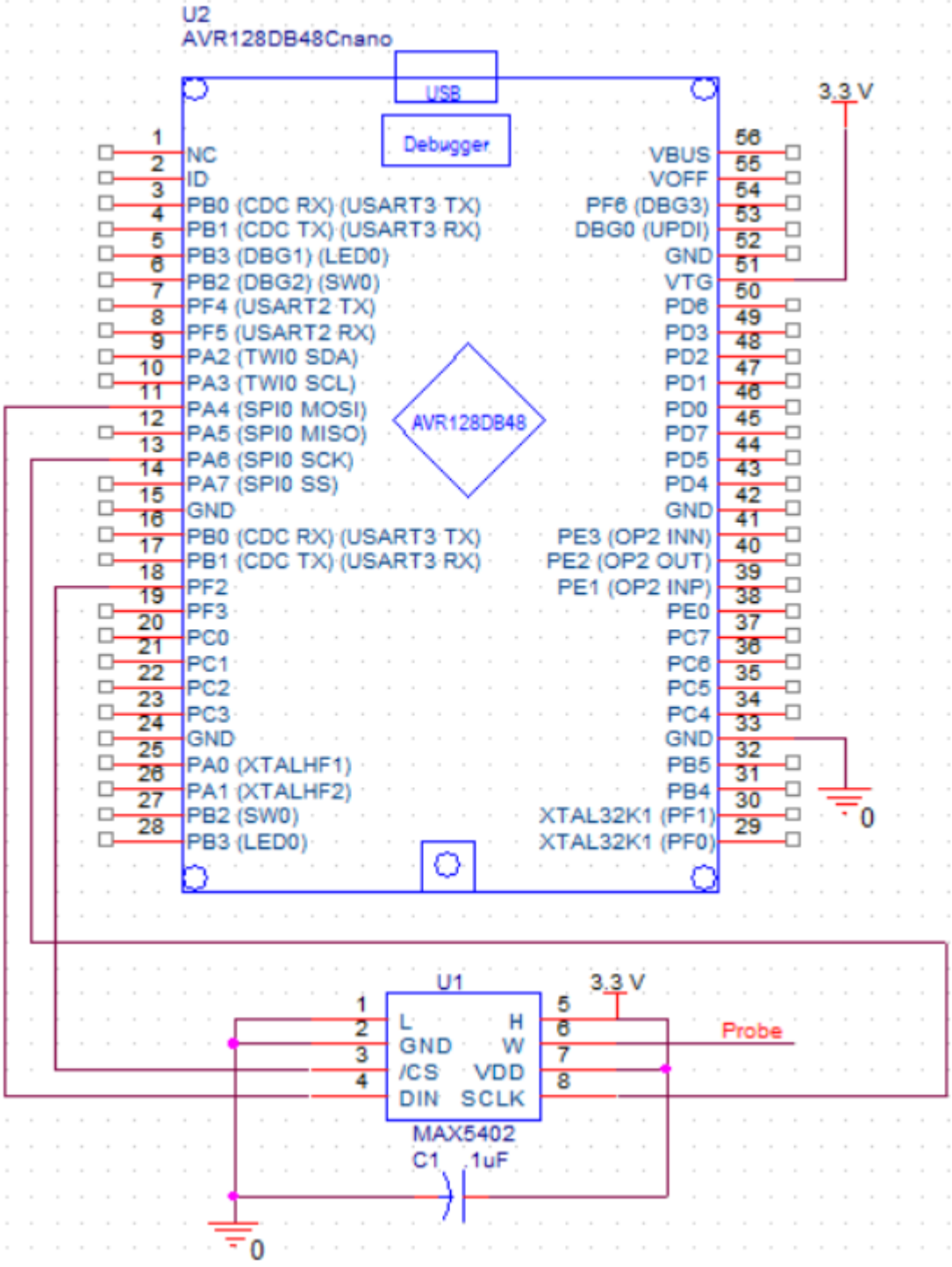
3) There is a diagram in the MAX5402 Datasheet that shows the expected input and response of the Chip, this diagram shows that the CPHA should be 0 and the CPOL should be 0 because it should trigger on the leading edge of the clock and the clock should be inactive at 0



5) in the fsm, if a invalid command is sent then the 5402 is never instructed to do anything. Eventually in the FSM path it is detected that the instruction is invalid, and through that it is sent back to state 0.

6)

Decimal is defined as a 32 bit value seeing as we need to translate at least 3 bytes worth of data, and the best form of data type to use to hold all the information of 3 bytes would be a 32 bit integer, as there is no uint24_t.


U2
AVR128DB48Cnano
USB
Debugger
AVR128DB48
1 NC
2 ID
3 PB0 (CDC RX) (USART3 TX)
4 PB1 (CDC TX) (USART3 RX)
5 PB3 (DBG1) (LED0)
6 PB2 (DBG2) (SW0)
7 PF4 (USART2 TX)
8 PF5 (USART2 RX)
9 PA2 (TWI0 SDA)
10 PA3 (TWI0 SCL)
11 PA4 (SPI0 MOSI)
12 PA5 (SPI0 MISO)
13 PA6 (SPI0 SCK)
14 PA7 (SPI0 SS)
15 GND
16 PB0 (CDC RX) (USART3 TX)
17 PB1 (CDC TX) (USART3 RX)
18 PF2
19 PF3
20 PC0
21 PC1
22 PC2
23 PC3
24 GND
25 PA0 (XTALHF1)
26 PA1 (XTALHF2)
27 PB2 (SW0)
28 PB3 (LED0)
56 VBUS
55 VOFF
54 PF6 (DBG3)
53 DBG0 (UPDI)
52 GND
51 VTG
50 PD6
49 PD3
48 PD2
47 PD1
46 PD0
45 PD7
44 PD5
43 PD4
42 GND
41 PE3 (OP2 INN)
40 PE2 (OP2 OUT)
39 PE1 (OP2 INP)
38 PE0
37 PC7
36 PC6
35 PC5
34 PC4
33 GND
32 PB5
31 PB4
30 XTAL32K1 (PF1)
29 XTAL32K1 (PF0)
3.3 V
0
U1
1 L
2 GND
3 /CS
4 DIN
5 H
6 W
7 VDD
8 SCLK
MAX5402
C1 .1uF
3.3 V
Probe
0

```c
/*
 * ASCII_str_to_MAX5402.c
 *
 * Created: 3/9/2024 11:44:32 PM
 * Author : MysticOwl
 */

#define F_CPU 4000000 //freq
#define USART3_BAUD_RATE(BAUD_RATE) ((float)((F_CPU * 64 )/ (16 * (float) BAUD_RATE)) + .5)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdint.h>
#include <string.h>

volatile uint8_t commandreceived;

/* UART Buffer Defines */
#define USART_RX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_TX_BUFFER_SIZE 16     /* 2,4,8,16,32,64,128 or 256 bytes */
#define USART_RX_BUFFER_MASK ( USART_RX_BUFFER_SIZE - 1 )
#define USART_TX_BUFFER_MASK ( USART_TX_BUFFER_SIZE - 1 )

#if ( USART_RX_BUFFER_SIZE & USART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#if ( USART_TX_BUFFER_SIZE & USART_TX_BUFFER_MASK )
#error TX buffer size is not a power of 2
#endif

//#define RX_BUFFER_MARGIN ((uint8_t)(0.1 * USART_RX_BUFFER_SIZE + 0.5))

/* Static Variables */
static unsigned char USART_RxBuf[USART_RX_BUFFER_SIZE];
static  uint8_t USART_RxHead;  //orig. declared volatile - kls
static  uint8_t USART_RxTail;  //orig. declared volatile - kls
//static unsigned char USART_TxBuf[USART_TX_BUFFER_SIZE];
static  uint8_t USART_TxHead;  //orig. declared volatile - kls
static  uint8_t USART_TxTail;  //orig. declared volatile - kls

volatile uint8_t cntrlcBM ;

uint8_t sdr;    //serial data received
uint8_t MAX5402_data;   //data to be written to MAX5402
uint8_t pstate = 0; //present state
uint8_t d2, d1, d0; //digits of the decimal value received
uint32_t decimal;       //binary value equal to decimal value received
```

```
49
50
51  // Function to initialize USART3
52  void USART3_Init(uint16_t baud, uint8_t data_bits, unsigned char parity ){
53
54      PORTB.DIR = 0x01; // make the whole port an input.; // make the single pin
          an output.
55
56      USART3.BAUD =  (uint16_t)USART3_BAUD_RATE(baud);      //baud rate
57
58      USART3.CTRLB =  0b11000000;    //transmitter and receiver enabling  as
          output
59
60
61      cntrlcBM =  0x00 ;    //frame format
62      //data bits format:
63
64      switch(data_bits) {
65          case 5:
66          cntrlcBM |= 0x00;
67          break;
68          case 6:
69          cntrlcBM |= 0x01;
70          break;
71          case 7:
72          cntrlcBM |= 0x02;
73          break;
74          case 8:
75          cntrlcBM |= 0x03;
76          break;
77          case 9:
78          cntrlcBM |= 0x06;
79          break;
80          case 10:
81          cntrlcBM |= 0x07;
82          break;
83          default:
84          cntrlcBM |= 0x00; //not valid choice
85          break;
86      }
87
88      //stop bit mode:
89
90      cntrlcBM |= 0x00; //1 stop bit
91      //cntrlcBM |= 0x04; // 2 stop bits
92
93      //parity format:
94
95
```

```
 96      switch(parity) {
 97          case 'D':
 98          cntrlcBM |= 0x00;
 99          break;
100          case 'E':
101          cntrlcBM |= 0x20;
102          break;
103          case 'O':
104          cntrlcBM |= 0x30;
105          break;
106          default:
107          cntrlcBM |= 0x00; //not valid choice
108          break;
109      }
110
111  }
112
113
114
115
116  void MAX5402_SPI0_init (){
117
118      PORTA.DIR |= PIN4_bm; /* Set MOSI pin direction to output */
119      PORTA.DIR &= ~PIN5_bm; /* Set MISO pin direction to input */
120      PORTA.DIR |= PIN6_bm; /* Set SCK pin direction to output */
121      PORTA.DIR |= PIN7_bm; /* Set SS pin direction to output */
122
123      PORTF.DIR |=                                                          ↵
           PIN2_bm;////////////////////////////////////////////////////////// ↵
           ////////////////////////////////////
124      PORTF.OUT |=                                                          ↵
           PIN2_bm;////////////////////////////////////////////////////////// ↵
           ////////////////////////////////////
125
126
127      //control a
128      // SPI0.CTRLA |= (SPI_ENABLE_bm | SPI_MASTER_bm); //enable spi
129      // SPI0.CTRLA |= SPI_CLK2X_bm; // we want the fastest clock possible so no  ↵
           prescalar
130      //SPI0.CTRLA |= SPI_MASTER_bm; // master mode
131      // Data order needs no bm change, as we want MSB first
132
133
134      /* Enable module */
135      /* SPI module in Master mode */
136
137
138
139      //control b
```

```
140     SPI0.CTRLB |= SPI_SSD_bm ;//| SPI_MODE0_bm; // mode zero as per the MAX5402 init waveforms
141
142     SPI0.CTRLA = SPI_ENABLE_bm | SPI_MASTER_bm;
143 }
144
145
146 void MAX5402_SPI0_write (uint8_t written){
147
148     PORTF_OUT = 0b00000000; //Slave select ON
149
150     SPI0.DATA = written;
151
152     while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* waits until data is exchanged*/
153     {
154         ;
155     }
156
157     //
158     PORTF_OUT = 0b00000100; //Slave select OFF ////////////////////////////////////////////////////////////////////
159
160 }
161
162
163
164 void parseANDsend(char bufferdata){
165
166  sdr = (uint8_t)bufferdata;
167
168
169     //The switch statement labeled FSM creates a FSM to parse the command
170     //string received in Task 3. You will have to analyze its operation
171     //to answer some of the questions for this laboratory.
172  switch (pstate)
173     {
174         case 0:
175         if (sdr == 'V')
176         pstate = 1;
177         else
178         pstate = 0;
179         break;
180
181         case 1:
182         if ((sdr >= '0') && (sdr <= '9'))
183         {
184             d2 = sdr & 0x0F;
185             pstate = 2;
```

```
186         }
187         else
188         pstate = 0;
189         break;
190 
191         case 2:
192         if ((sdr >= '0') && (sdr <= '9'))
193         {
194             d1 = sdr & 0x0F;
195             pstate = 3;
196         }
197         else
198         pstate = '0';
199         break;
200 
201         case 3:
202         if ((sdr >= '0') && (sdr <= '9'))
203         {
204             d0 = sdr & 0x0F;
205             pstate = 4;
206         }
207         else
208         pstate = 0;
209         break;
210 
211         case 4:
212         if (sdr == 0x0d)
213         pstate = 5;
214         else
215         pstate = 0;
216         break;
217 
218         case 5:
219         if (sdr == 0x0a)
220         {
221             pstate = 0;
222             decimal = (((d2 * 10) + d1) * 10) + d0;
223             // Assuming H = 3.30V, max output is 3.2872V when
224             // decimal is 329.
225             if (decimal > 329)
226             MAX5402_data = 255;
227             else
228             MAX5402_data = (uint8_t)(((decimal) * 255)/329);
229             MAX5402_SPI0_write(MAX5402_data);
230 
231             _delay_ms(1);
232         }
233         else
234         pstate = 0;
```

```
235         break;
236
237         default:
238         pstate = 0;
239     }
240
241
242
243
244
245 }
246
247
248
249 /* Interrupt handlers */
250
251 ISR (USART3_RXC_vect)       //Receive complete interrupt
252 {
253     uint8_t data;
254
255     //The following variable is not necessary if you are not going to take any
        action
256     //for an overflow that requires keeping the old index. Instead just use
257     //USART_RxHead instead of tmphead.
258     uint8_t tmphead;
259
260     cli();      // Clear global interrupt flag
261
262     /* Read the received data */
263     data = USART3.RXDATAL;
264
265     /* Calculate buffer index, increment and possibly roll over index */
266     tmphead = ( USART_RxHead + 1 ) & USART_RX_BUFFER_MASK;
267
268
269     //************************************************************************
270     /*
271     The following condition could be changed to
272     if ( (tmphead >= (USART_RxTail + RX_BUFFER_MARGIN)) || (USART_RxTail >=
        (tmphead + RX_BUFFER_MARGIN)));
273     {
274     //Use flow control to stop flow of characters:
275     (a) hardware unasserts CTS
276     (b) software send XOFF
277     }
278     */
279     //************************************************************************
280
281     if ( tmphead == USART_RxTail )
```

```
282     {
283         // ERROR! Receive buffer overflow
284     }
285
286     USART_RxBuf[tmphead] = data; // Store received data in buffer
287     //Alternate position B for USART_RxHead = tmphead;
288     USART_RxHead = tmphead;      // Store new index (was prev. in position A)
289     sei();      // re enable global interrupts
290 }
291
292
293
294 /* Read function */
295 unsigned char USART3_Receive( void )
296 {
297     uint8_t tmptail = USART_RxTail;
298     uint8_t tmphead = USART_RxHead;
299
300     while ( USART_RxHead == USART_RxTail ){  /* Wait for incomming data */
301
302         asm volatile ("nop");
303
304     }
305     tmptail = ( USART_RxTail + 1 ) & USART_RX_BUFFER_MASK;/* Calculate buffer
        index */
306     USART_RxTail = tmptail;                /* Store new index */
307     return USART_RxBuf[tmptail];           /* Return data */
308     //USART3.CTRLA |= USART_DREIE_bm;                   /* Enable UDRE
        interrupt */
309 }
310
311
312 int main(void)
313 {
314     /* Replace with your application code */
315 // uint8_t data = 1;
316 // uint8_t i ;
317 MAX5402_SPI0_init();
318 //MAX5402_SPI0_write(data);
319
320  sei();
321  SREG |=  CPU_I_bm;
322
323 USART_RxTail = 0x00;    //clear buffer indexes, not really necessary
324 USART_RxHead = 0x00;    //because they are automatically cleared since
325 USART_TxTail = 0x00;    //declared as global uninitialized variables
326 USART_TxHead = 0x00;
327
328 // SW0 pin an input, must be pressed to transfer data from Rx to Tx buffer
```

```
329  PORTB.DIR &= ~PIN2_bm;      // SW0 pin pushbutton input
330  PORTB.PIN2CTRL = 0x08;      //enable pull up
331  USART3_Init( 9600 , 8, 'D');   // Initialize USART3
332  sei();           // Enable global interrupts => enable USART interrupts
333  USART3.CTRLA |= USART_RXCIE_bm; /* Receive Complete Interrupt must be enabled
       */
334  //  USART3.CTRLA |= USART_DREIE_bm;
335
336
337  for( ; ; )        // Forever
338  {
339      //Uncomment next statement to have operation independent of SW0
340      parseANDsend(USART3_Receive());
341
342  }
343
344
345  return 0;
346
347  }
348
349
```