

```
1  /*
2   * FileName: main.c
3   * Version: 1
4   *
5   * Created: 11/9/2022 1:54 PM
6   * Author: Ethan Zeronik
7   *
8   * Operations: lcd test
9   *
10  * Hardware:
11  *   Atmega2560          micro controller
12  *   PORTL              LCD data
13  *   PORTD.0            LCD RS
14  *   PORTD.1            LCD R/W
15  *   PORTD.2            LCD E
16  */
17
18 /* NOTE: Includes */
19 #define F_CPU 16000000UL
20
21 #include <avr/io.h>
22 #include <util/delay.h>
23
24 #include "LiquidCrystalDisplay.h"
25
26 /* NOTE: Custom Macros */
27 // TODO: None
28
29 /* NOTE: Global Variables */
30 char message[] = {"Test"};
31
32 /* NOTE: Function prototypes */
33 // inits IO ports
34 void IO_init(void);
35
36 /* NOTE: Application implementation */
37 // the main loop of the function, provided to us
38 int main(void)
39 {
40     IO_init();
41     LCD_init(&DDRD, &PORTD, &DDRL, &PORTL);
42
43     LCD_sendInstruction(0x01);
44     _delay_ms(2); // can remove if use Busy Flag check
45
46     LCD_sendInstruction(0x02);
47     _delay_ms(2); // can remove if use Busy Flag check
48
49     LCD_sendInstruction(0x86);
50     _delay_ms(50);
51
52     LCD_sendString(message);
53     _delay_ms(50);
54
55     LCD_sendInstruction(0xC6);
56     _delay_us(50); // can remove if use Busy Flag check
57 }
```

```
58     LCD_sendString(message);
59
60     while(1)
61     {
62     }
63 }
64
65 /* NOTE: Function implementations */
66 void IO_init(void)
67 {
68     // do nothing
69 }
70
```

```
1  /*
2   * FileName: main.c
3   * Version: 1
4   *
5   * Created: 11/2/2022 1:39:55 PM
6   * Author: Ethan Zeronik
7   *
8   * Operations: basic serial
9   *
10  * Hardware:
11  *   Atmega2560          micro controller
12  *   PORTA              LED bar
13  *   PORTC              Button/Switches
14  */
15
16 /* NOTE: Includes */
17 #include <avr/io.h>
18 #include <avr/interrupt.h>
19
20 #include "Serial.h"
21
22 /* NOTE: Custom Macros */
23 // TODO: None
24
25 /* NOTE: Global Variables */
26 // TODO: None
27
28 /* NOTE: Function prototypes */
29 // inits IO ports
30 void IO_init(void);
31 // handler
32 void asyncGetHandler(char c);
33
34 /* NOTE: Application implementation */
35 // the main loop of the function, provided to us
36 int main(void)
37 {
38     IO_init();
39
40     // init async uart and bind an interrupt handler
41     SERIAL_uartInitAsync(USART0, 9600);
42     SERIAL_uartAsyncGetHandler(USART0, &asyncGetHandler);
43
44     sei();
45
46     while(1)
47     {
48         SERIAL_uartSendFixed(USART0, (char const * const)&PINC, 1);
49     }
50 }
51
52 /* NOTE: Function implementations */
53 void IO_init(void)
54 {
55     // set portA as an output
56     DDRA = 0xFF;
57     PORTA = 0x00;
```

```
58  
59     // set portC as an input  
60     DDRC = 0x00;  
61     PORTC = 0xFF;  
62 }  
63  
64 void asyncGetHandler(char c)  
65 {  
66     PORTA = c;  
67 }  
68
```

```
1  /*
2  * FileName: main.c
3  * Version: 1
4  *
5  * Created: 11/9/2022 2:26:50 PM
6  * Author: Ethan Zeronik
7  *
8  * Operations: uart to lcd
9  *
10 * Hardware:
11 *   Atmega2560          micro controller
12 *   PORTL               LCD data
13 *   PORTD.0             LCD RS
14 *   PORTD.1             LCD R/W
15 *   PORTD.2             LCD E
16 */
17
18 /* NOTE: Includes */
19 #include <avr/io.h>
20 #include <avr/interrupt.h>
21
22 #include "LiquidCrystalDisplay.h"
23 #include "Serial.h"
24
25 /* NOTE: Custom Macros */
26 // TODO: None
27
28 /* NOTE: Global Variables */
29 // buffer for uart
30 char message[24] = {0};
31 uint8_t messageIndex = 0;
32 uint8_t readFlag = 0;
33
34 /* NOTE: Function prototypes */
35 // inits IO ports
36 void IO_init(void);
37 // handler
38 void asyncGetHandler(char c);
39
40 /* NOTE: Application implementation */
41 // the main loop of the function, provided to us
42 int main(void)
43 {
44     IO_init();
45     LCD_init(&DDRD, &PORTD, &DDRL, &PORTL);
46
47     // init async uart and bind an interrupt handler
48     SERIAL_uartInitAsync(USART0, 9600);
49     SERIAL_uartAsyncGetHandler(USART0, &asyncGetHandler);
50
51     sei();
52
53     LCD_sendInstruction(0x01);
54     LCD_sendInstruction(0x02);
55
56     while(1)
57     {
```

```
58         if(readFlag)
59         {
60             LCD_sendInstruction(0x01);
61             LCD_sendInstruction(0x02);
62             LCD_sendString(message);
63
64             readFlag = 0;
65         }
66     }
67 }
68
69 /* NOTE: Function implementations */
70 void IO_init(void)
71 {
72     // do nothing
73 }
74
75 void asyncGetHandler(char c)
76 {
77     if(c != 0x0d && c != 0x0a && c != '\0')
78     {
79         if(messageIndex < 23)
80         {
81             // add to array
82             message[messageIndex] = c;
83             message[messageIndex + 1] = '\0';
84
85             messageIndex++;
86         }
87     }
88     else
89     {
90         for(uint8_t i = 0; i < 24; i++)
91         {
92             message[i] = '\0';
93         }
94
95         messageIndex = 0;
96     }
97
98     // set update flag
99     readFlag = 1;
100 }
101
```

```
1  /*
2   * FileName: LiquidCrystalDisplay.h
3   * Version: 1
4   *
5   * Created: 11/9/22 1:59 PM
6   * Author: Ethan Zeronik
7   *
8   * Operations: lcd definition
9   */
10
11 #ifndef LiquidCrystalDisplay_h_INCLUDED
12 #define LiquidCrystalDisplay_h_INCLUDED
13
14 #if defined(__cplusplus)
15 extern "C" {
16 #endif
17
18 #pragma message("WARNING: this module uses lower 3 bits of control and entire data buffer
19 assigned")
20
21 #include <stdint.h>
22
23 /* NOTE: Custom Types */
24 // custom type for charcater definition
25 typedef uint8_t LcdCustomCharacter_t[8];
26 // typing for the allowed character addresses
27 typedef enum LcdCharacterAddress_t
28 {
29     // the first charcater slot
30     lcdFirstSlot = 0x40,
31     // the second charcater slot
32     lcdSecondSlot = 0x48,
33     // the third charcater slot
34     lcdThirdSlot = 0x50,
35     // the fourth charcater slot
36     lcdFourthSlot = 0x58,
37     // the fifth charcater slot
38     lcdFifthSlot = 0x60,
39     // the sixth charcater slot
40     lcdSixthSlot = 0x68,
41     // the seventh charcater slot
42     lcdSeventhSlot = 0x70,
43     // the eighth charcater slot
44     lcdEighthSlot = 0x78,
45 } LcdCharacterAddress_t;
46
47 /* NOTE: Function prototypes */
48 // init for the lcd
49 // the data register is entirely used
50 // the control register is only used for the lower 3 bits
51 void LCD_init(uint8_t volatile * const pControlRegister, uint8_t volatile * const
52 pControlPort, uint8_t volatile * const pDataRegister, uint8_t volatile * const pDataPort);
53 // sends the given instruction
54 void LCD_sendInstruction(uint8_t input);
55 // send a single char
56 void LCD_sendChar(char c);
57 // send a string
```

```
56 void LCD_sendString(char const * const pData);
57 // creates the custom character at the given address
58 void LCD_createCharacter(LcdCharacterAddress_t address, LcdCustomCharacter_t custom);
59 // gets the created character
60 // to be used as a char in a astring
61 char LCD_getCharacter(LcdCharacterAddress_t address);
62
63 #if defined(__cplusplus)
64 } /* extern "C" */
65 #endif
66
67 #endif // LiquidCrystalDisplay_h_INCLUDED
```



```
1  /*
2  * FileName: LiquidCrystalDisplay.c
3  * Version: 1
4  *
5  * Created: 11/9/22 1:59 PM
6  * Author: Ethan Zeronik
7  *
8  * Operations: lcd implementation
9  */
10
11 /* NOTE: Includes */
12 #include "LiquidCrystalDisplay.h"
13
14 #if !defined(F_CPU)
15     #define F_CPU 16000000UL
16 #endif
17
18 #include <avr/io.h>
19 #include <util/delay.h>
20
21 /* NOTE: Global Variables */
22 // instance pointer to the control logic
23 static uint8_t * sControlPort;
24 // instance pointer to the data port
25 static uint8_t * sDataPort;
26
27 /* NOTE: Local function implementations */
28 void LCD_init(uint8_t volatile * const pControlRegister, uint8_t volatile * const
pControlPort, uint8_t volatile * const pDataRegister, uint8_t volatile * const pDataPort)
29 {
30     // configure port register and turn off port
31     *pDataRegister |= 0xff;
32     *pDataPort = 0x00;
33
34     // configure port register and turn off port
35     *pControlRegister |= 0x07;
36     *pControlPort = (*pControlPort & 0xf8) | 0x00;
37
38     sControlPort = (uint8_t *)pControlPort;
39     sDataPort = (uint8_t *)pDataPort;
40
41     // wait for lcd to power up
42     _delay_ms(35);
43
44     // set lcd to 8 bits, 2 lines, display off
45     LCD_sendInstruction(0x38);
46     _delay_us(50);
47
48     // set lcd to display on, cursor off, blink off
49     LCD_sendInstruction(0x0C);
50     _delay_us(50);
51
52     // clear the display
53     LCD_sendInstruction(0x01);
54     _delay_ms(2);
55
56     // increment mode
```

```
57     LCD_sendInstruction(0x06);
58 }
59
60 void LCD_sendInstruction(uint8_t input)
61 {
62     // set controls to RS = 0 E = 0, R!/W=0 then take E high
63     *sContolPort = (*sContolPort & 0xf8) | 0x00;
64     *sContolPort |= 0x04;
65
66     // send data then delay for at least 50us
67     *sDataPort = input;
68     _delay_us(50);
69
70     // take E low
71     *sContolPort = *sContolPort & 0xf9;
72
73     _delay_ms(5);
74 }
75
76 void LCD_sendChar(char c)
77 {
78     // set controls to RS = 1 E = 0, R!/W=0 then take E high
79     *sContolPort = (*sContolPort & 0xf8) | 0x01;
80     *sContolPort = *sContolPort | 0x04;
81
82     // send data then delay for at least 50us
83     *sDataPort = c;
84     _delay_us(50);
85
86     // take E low
87     *sContolPort = (*sContolPort & 0xf8) | 0x01;
88
89     _delay_ms(5);
90 }
91
92 void LCD_sendString(char const * const pData)
93 {
94     char * localPointer = (char * const)pData;
95
96     // set controls to RS = 1 E = 0, R!/W=0
97     *sContolPort = (*sContolPort & 0xf8) | 0x01;
98
99     while(*localPointer != '\0')
100     {
101         // take E high
102         *sContolPort = *sContolPort | 0x04;
103
104         // send data then delay for at least 50us
105         *sDataPort = *localPointer++;
106         _delay_us(50);
107
108         // take E low
109         *sContolPort = (*sContolPort & 0xf8) | 0x01;
110
111         _delay_us(50); /* Delay REQUIRED */
112     }
113 }
```

```
114     _delay_ms(5);
115 }
116
117 void LCD_createCharacter(LcdCharacterAddress_t address, LcdCustomCharacter_t custom)
118 {
119     // make sure that the character is correct and valid
120     LcdCustomCharacter_t safety = {
121         0x40 | (custom[0] & 0x1f),
122         0x40 | (custom[1] & 0x1f),
123         0x40 | (custom[2] & 0x1f),
124         0x40 | (custom[3] & 0x1f),
125         0x40 | (custom[4] & 0x1f),
126         0x40 | (custom[5] & 0x1f),
127         0x40 | (custom[6] & 0x1f),
128         0x40 | (custom[7] & 0x1f),
129     };
130
131     // set the address of the cgram (must be between 64 - 127)
132     LCD_sendInstruction(address);
133
134     // send the sterile character
135     LCD_sendString((char const * const)safety);
136 }
137
138 char LCD_getCharacter(LcdCharacterAddress_t address)
139 {
140     return (address - 64) / 8;
141 }
```

```
1  /*
2   * FileName: Serial.h
3   * Version: 1
4   *
5   * Created: 11/2/2022 1:41 PM
6   * Author: Ethan Zeronik
7   *
8   * Operations: serial definition
9   */
10
11 #ifndef Serial_h_INCLUDED
12 #define Serial_h_INCLUDED
13
14 #if defined(__cplusplus)
15 extern "C" {
16 #endif
17
18 #pragma message("WARNING: this module defaults to TX0/RX0 for interrupts and serial
19 communication")
20
21 #include <stdint.h>
22
23 /* NOTE: Custom Types */
24 // typing for the stepper motor enum
25 typedef enum SerialPortSelector_t
26 {
27     // usart 0 selector
28     serialUsart0 = 0,
29     // usart 1 selector
30     serialUsart1 = 1,
31     // usart 2 selector
32     serialUsart2 = 2,
33     // usart 3 selector
34     serialUsart3 = 3,
35 } SerialPortSelector_t;
36
37 // typing for the handler function
38 typedef void (*SerialAsyncGetHandler_t)(char);
39
40 /* NOTE: Function prototypes */
41 // init for the serial sync mode
42 void SERIAL_uartInit(SerialPortSelector_t port, uint32_t baud);
43 // init for the serial async mode
44 void SERIAL_uartInitAsync(SerialPortSelector_t port, uint32_t baud);
45 // sends the buffer to the desired port
46 // the string must be null terminated
47 void SERIAL_uartSend(SerialPortSelector_t port, char const * const pTransmitString);
48 // sends the buffer to the desired port
49 void SERIAL_uartSendFixed(SerialPortSelector_t port, char const * const pTransmitString,
50 uint16_t length);
51 // get a char from the serial buffer
52 char SERIAL_uartGetSync(SerialPortSelector_t port);
53 // set the async handler
54 // will run on every character to the uart buffer
55 void SERIAL_uartAsyncGetHandler(SerialPortSelector_t port, SerialAsyncGetHandler_t cb);
56
57 #if defined(__cplusplus)
```

```
56 | } /* extern "C" */  
57 | #endif  
58 |  
59 | #endif // Serial_h_INCLUDED
```

```
1  /*
2  * FileName: Delay.c
3  * Version: 1
4  *
5  * Created: 11/2/2022 1:41 PM
6  * Author: Ethan Zeronik
7  *
8  * Operations: serial implementation
9  */
10
11 /* NOTE: Includes */
12 #include "Serial.h"
13
14 #if !defined(F_CPU)
15     #define F_CPU 16000000UL
16 #endif
17
18 #include <avr/io.h>
19 #include <avr/interrupt.h>
20
21 /* NOTE: Local declarations */
22 // a helper to poll for the given registers then set the given value
23 // register and mask are related
24 // set will be set to the value at read once polling finishes
25 void pollThenSetHelper(uint8_t volatile * const pRegister, uint8_t mask, uint8_t volatile *
const pSet, uint8_t volatile * const pRead);
26 // a helper to remove redundant logic for sending data
27 void sendCharHelper(SerialPortSelector_t port, char const * const value);
28
29 /* NOTE: Global Variables */
30 // the handlers for each main uart channel
31 static SerialAsyncGetHandler_t interruptCallback[4];
32
33 /* NOTE: Local function implementations */
34 void SERIAL_uartInit(SerialPortSelector_t port, uint32_t baud)
35 {
36     uint16_t baudCalc = ((F_CPU / baud) / 16) - 1;
37
38     switch(port)
39     {
40     default:
41     case serialUsart0:
42     {
43         UCSR0A = 0x00;
44         // enable UART TX and RX with interrupt flag
45         UCSR0B = 0x18;
46         // set the UART for N, 8, 1
47         UCSR0C = 0x06;
48         // set BAUD Rate for 16MHz clock
49         UBRR0L = baudCalc;
50         UBRR0H = (baudCalc >> 8) & 0x0f;
51     }
52     break;
53     case serialUsart1:
54     {
55         UCSR1A = 0x00;
56         UCSR1B = 0x18;
```

```
57         UCSR1C = 0x06;
58         UBRR1L = baudCalc;
59         UBRR1H = (baudCalc >> 8) & 0x0f;
60     }
61     break;
62     case serialUsart2:
63     {
64         UCSR2A = 0x00;
65         UCSR2B = 0x18;
66         UCSR2C = 0x06;
67         UBRR2L = baudCalc;
68         UBRR2H = (baudCalc >> 8) & 0x0f;
69     }
70     break;
71     case serialUsart3:
72     {
73         UCSR3A = 0x00;
74         UCSR3B = 0x18;
75         UCSR3C = 0x06;
76         UBRR3L = baudCalc;
77         UBRR3H = (baudCalc >> 8) & 0x0f;
78     }
79     break;
80 }
81 }
82
83 void SERIAL_uartInitAsync(SerialPortSelector_t port, uint32_t baud)
84 {
85     SERIAL_uartInit(port, baud);
86
87     // turn on the rx interrupt
88     switch(port)
89     {
90     default:
91     case serialUsart0:
92         UCSR0B = 0x98;
93         break;
94     case serialUsart1:
95         UCSR1B = 0x98;
96         break;
97     case serialUsart2:
98         UCSR2B = 0x98;
99         break;
100    case serialUsart3:
101        UCSR3B = 0x98;
102        break;
103    }
104 }
105
106 void SERIAL_uartSend(SerialPortSelector_t port, char const * const pTransmitString)
107 {
108     char const * pWorker = (char const *)pTransmitString;
109
110     // while we are not at the end of the string
111     while(*pWorker != '\n')
112     {
113         // wait for uart tx to be ready then send out uart
```

```
114     sendCharHelper(port, pWorker);
115
116     pWorker++;
117 }
118 }
119
120 void SERIAL_uartSendFixed(SerialPortSelector_t port, char const * const pTransmitString,
uint16_t length)
121 {
122     for(uint16_t i = 0; i < length; i++)
123     {
124         // wait for uart tx to be ready then send out uart
125         sendCharHelper(port, pTransmitString + i);
126     }
127 }
128
129 char SERIAL_uartGetSync(SerialPortSelector_t port)
130 {
131     char ch;
132
133     // wait for uart rx to be ready and save to the char
134     switch(port)
135     {
136         default:
137         case serialUsart0:
138             pollThenSetHelper(&UCSR0A, (1 << RXC0), (uint8_t volatile * const)&ch, &UDR0);
139             break;
140         case serialUsart1:
141             pollThenSetHelper(&UCSR1A, (1 << RXC1), (uint8_t volatile * const)&ch, &UDR1);
142             break;
143         case serialUsart2:
144             pollThenSetHelper(&UCSR2A, (1 << RXC2), (uint8_t volatile * const)&ch, &UDR2);
145             break;
146         case serialUsart3:
147             pollThenSetHelper(&UCSR3A, (1 << RXC3), (uint8_t volatile * const)&ch, &UDR3);
148             break;
149     }
150
151     return ch;
152 }
153
154 void SERIAL_uartAsyncGetHandler(SerialPortSelector_t const port, SerialAsyncGetHandler_t cb)
155 {
156     // set the internal callback pointer to the one we were given
157     interruptCallback[port] = cb;
158 }
159
160 /* NOTE: Local function implementations */
161 void pollThenSetHelper(uint8_t volatile * const pRegister, uint8_t mask, uint8_t volatile *
const pSet, uint8_t volatile * const pRead)
162 {
163     // wait for the register
164     while((*pRegister & mask) == 0)
165     {
166     }
167
168     // save to the pSet
169     *pSet = *pRead;
```



```
170 }
171
172 void sendCharHelper(SerialPortSelector_t port, char const * const value)
173 {
174     switch(port)
175     {
176         default:
177         case serialUsart0:
178             pollThenSetHelper(&UCSR0A, (1 << UDRE0), &UDR0, (uint8_t volatile * const)value);
179             break;
180         case serialUsart1:
181             pollThenSetHelper(&UCSR1A, (1 << UDRE1), &UDR1, (uint8_t volatile * const)value);
182             break;
183         case serialUsart2:
184             pollThenSetHelper(&UCSR2A, (1 << UDRE2), &UDR2, (uint8_t volatile * const)value);
185             break;
186         case serialUsart3:
187             pollThenSetHelper(&UCSR3A, (1 << UDRE3), &UDR3, (uint8_t volatile * const)value);
188             break;
189     }
190 }
191
192 ISR(USART0_RX_vect)
193 {
194     interruptCallback[serialUsart0](UDR0);
195 }
196
197 ISR(USART1_RX_vect)
198 {
199     interruptCallback[serialUsart1](UDR1);
200 }
201
202 ISR(USART2_RX_vect)
203 {
204     interruptCallback[serialUsart2](UDR2);
205 }
206
207 ISR(USART3_RX_vect)
208 {
209     interruptCallback[serialUsart3](UDR3);
210 }
```