Lab 6 Report

EEL4742C - 00446

Yousef Awad

September 2025

Contents

Introduction

In this lab, we learned how to use the UART module on the MSP430 as well as what UART is generally, via programming the backchannel UART link that connects the board to the PC of use.

6.1 Transmitting Data over UART

```
#include <msp430fr6989.h>

#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7

// UART Channels are P3.4 and P3.5 for transmit and recieve
    respectively
#define transmit BIT4
#define recieve BIT5

// WE LOVE DEFINES
#define FLAGS UCA1IFG // Contains the transmit & receive flags
#define RXFLAG UCRXIFG // Receive flag
#define TXFLAG UCTXIFG // Transmit flag
#define TXBUFFER UCA1TXBUF // Transmit buffer
#define RXBUFFER UCA1RXBUF // Receive buffer

void initialize_uart(void)
{
  // Configuring the pins to use backchannel uart
  P3SEL1 &= ~(transmit | recieve);
  P3SEL0 |= (transmit | recieve);

  // Setting the clock to SMCLK
  UCA1CTLW0 |= UCSSEL_2;

  // Setting the dividers and enabling oversampling
  UCA1BRW = 6;
  // setting the modulators and such
  UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;

  // Exiting the reset state
  UCA1CTLW0 &= ~UCSWRST;
}

void uart_write_char(volatile unsigned char ch)
{
  while (!(FLAGS & TXFLAG))
  {
    // Wait for transmission that is ongoing to complete
  }

  TXBUFFER = ch;
  return;
}

volatile unsigned char uart_read_char(void)
```

```
{
  if (!(FLAGS & RXFLAG))
  {
    return 0; // no byte was recieved
  }

  // Return the buffer
  volatile unsigned char return_char = RXBUFFER;
  return return_char;
}

int main(void)
{
  // Enabling the leds and other stuff
  WDTCTL = WDTPW | WDTHOLD; // Stop WDT
  PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
  // Pins as output
  P1DIR |= redLED;
  P9DIR |= greenLED;
  // Setting both leds to off
  P1OUT &= ~redLED;
  P9OUT &= ~greenLED;

  // doing what the function says
  initialize_uart();

  unsigned char count = '0';

  for (;;)
  {
    volatile unsigned char read_char = uart_read_char();
    if (read_char == '1')
    {
      P9OUT |= greenLED;
    }
    else if (read_char == '2')
    {
      P9OUT &= ~greenLED;
    }

    if (count > '9')
    {
      count = '0';
    }
    uart_write_char(count);
    uart_write_char('\n');
    uart_write_char('\r');
    count += 1;

    P1OUT ^= redLED;

    // delay
    _delay_cycles(100000); // 100,000 cycles
  }
}
```

6.2 Transmitting Integers & Strings over UART

```c
#include <msp430fr6989.h>
#include <stdint.h>
#include <string.h>

// UART Channels are P3.4 and P3.5 for transmit and recieve respectively
#define transmit BIT4
#define recieve BIT5

// WE LOVE DEFINES
#define FLAGS UCA1IFG // Contains the transmit & receive flags
#define RXFLAG UCRXIFG // Receive flag
#define TXFLAG UCTXIFG // Transmit flag
#define TXBUFFER UCA1TXBUF // Transmit buffer
#define RXBUFFER UCA1RXBUF // Receive buffer

// Reverses a given string
void strrev(char *str)
{
  int i = 0;
  int j = strlen(str) - 1;
  char temp;
  while (i < j)
  {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
  }
}

// Converts an unsigned 16-bit integer to a null-terminated string (base 10).
void custom_itoa(uint16_t number, char *buffer)
{
  int i = 0;

  // Handle the special case of 0
  if (number == 0)
  {
    buffer[i++] = '0';
    buffer[i] = '\0';
    return;
  }

  // Process individual digits
  while (number > 0)
  {
    int remainder = number % 10;
    buffer[i++] = remainder + '0'; // Convert digit to its ASCII character
    number = number / 10;
  }

```

```
53    buffer[i] = '\0'; // Null-terminate the string
54
55    // The digits are in reverse order, so we need to reverse the
        string
56    strrev(buffer);
57  }
58
59  void initialize_uart(void)
60  {
61    // Configuring the pins to use backchannel uart
62    P3SEL1 &= ~(transmit | recieve);
63    P3SEL0 |= (transmit | recieve);
64
65    // Setting the clock to SMCLK
66    UCA1CTLW0 |= UCSSEL_2;
67
68    // Setting the dividers and enabling oversampling
69    UCA1BRW = 6;
70    // setting the modulators and such
71    UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;
72
73    // Exiting the reset state
74    UCA1CTLW0 &= ~UCSWRST;
75  }
76
77  void uart_write_char(volatile unsigned char ch)
78  {
79    while (!(FLAGS & TXFLAG))
80    {
81      // Wait for transmission that is ongoing to complete
82    }
83
84    TXBUFFER = ch;
85    return;
86  }
87
88  unsigned char uart_read_char(void)
89  {
90    if (!(FLAGS & RXFLAG))
91    {
92      return 0; // no byte was recieved
93    }
94
95    // Return the buffer
96    volatile unsigned char return_char = RXBUFFER;
97    return return_char;
98  }
99
100 void uart_write_string(char *string)
101 {
102   int i; // counter
103   for (i = 0; i < strlen(string); i++)
104   {
105     uart_write_char(string[i]);
106   }
107   return;
108 }
```

```
109
110 void uart_write_uint16 (uint16_t number)
111 {
112   // Converting the number via snprintf
113   char buffer[6]; // 5 characters is the max amount of characters
      for 65,536
114   custom_itoa(number, buffer);
115   uart_write_string(buffer);
116   return;
117 }
118
119 int main(void)
120 {
121   // Enabling the leds and other stuff
122   WDTCTL = WDTPW | WDTHOLD; // Stop WDT
123   PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
124
125   // doing what the function says
126   initialize_uart();
127
128   // Printing 'Hello World!!' to the console
129   uart_write_string("Hello World!!");
130   uart_write_char('\n');
131   uart_write_char('\r');
132
133   uart_write_uint16(65432);
134   uart_write_char('\n');
135   uart_write_char('\r');
136 }
```

6.3 Modifying the UART Configuration

```c
#include <msp430fr6989.h>
#include <stdint.h>
#include <string.h>

// UART Channels are P3.4 and P3.5 for transmit and recieve respectively
#define transmit BIT4
#define recieve BIT5

// WE LOVE DEFINES
#define FLAGS UCA1IFG // Contains the transmit & receive flags
#define RXFLAG UCRXIFG // Receive flag
#define TXFLAG UCTXIFG // Transmit flag
#define TXBUFFER UCA1TXBUF // Transmit buffer
#define RXBUFFER UCA1RXBUF // Receive buffer

// Reverses a given string
void strrev(char *str)
{
  int i = 0;
  int j = strlen(str) - 1;
  char temp;
  while (i < j)
  {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
  }
}

// Converts an unsigned 16-bit integer to a null-terminated string (base 10).
void custom_itoa(uint16_t number, char *buffer)
{
  int i = 0;

  // Handle the special case of 0
  if (number == 0)
  {
    buffer[i++] = '0';
    buffer[i] = '\0';
    return;
  }

  // Process individual digits
  while (number > 0)
  {
    int remainder = number % 10;
    buffer[i++] = remainder + '0'; // Convert digit to its ASCII character
    number = number / 10;
  }

```

```
  buffer[i] = '\0'; // Null-terminate the string

  // The digits are in reverse order, so we need to reverse the
    string
  strrev(buffer);
}

// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal()
{
  // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

  // Reroute pins to LFXIN/LFXOUT functionality
  PJSEL1 &= ~BIT4;
  PJSEL0 |= BIT4;

  // Wait until the oscillator fault flags remain cleared
  CSCTL0 = CSKEY; // Unlock CS registers
  do
  {
    CSCTL5 &= ~LFXTOFFG; // Local fault flag
    SFRIFG1 &= ~OFIFG; // Global fault flag
  }
  while((CSCTL5 & LFXTOFFG) != 0);

  CSCTL0_H = 0; // Lock CS registers
  return;
}

void initialize_uart(void)
{
  // Configuring the pins to use backchannel uart (SAME)
  P3SEL1 &= ~(transmit | recieve);
  P3SEL0 |= (transmit | recieve);

  // Setting the clock to ACLK (SEL_1 and not SEL_2 [SMCLK])
  UCA1CTLW0 |= UCSSEL_1;

  // Setting the dividers and enabling oversampling
  UCA1BRW = 6; // SAME DIVIDER
  // setting the modulators and such
  UCA1MCTLW = UCBRS1 | UCBRS2 | UCBRS3 | UCBRS5 | UCBRS6 | UCBRS7;

  // Exiting the reset state
  UCA1CTLW0 &= ~UCSWRST;
}

void uart_write_char(volatile unsigned char ch)
{
  while (!(FLAGS & TXFLAG))
  {
    // Wait for transmission that is ongoing to complete
  }

  TXBUFFER = ch;
  return;
}
```

```
109
110 unsigned char uart_read_char(void)
111 {
112   if (!(FLAGS & RXFLAG))
113   {
114     return 0; // no byte was recieved
115   }
116
117   // Return the buffer
118   volatile unsigned char return_char = RXBUFFER;
119   return return_char;
120 }
121
122 void uart_write_string(char *string)
123 {
124   int i; // counter
125   for (i = 0; i < strlen(string); i++)
126   {
127     uart_write_char(string[i]);
128   }
129   return;
130 }
131
132 void uart_write_uint16 (uint16_t number)
133 {
134   // Converting the number via snprintf
135   char buffer[6]; // 5 characters is the max amount of characters
        for 65,536
136   custom_itoa(number, buffer);
137   uart_write_string(buffer);
138   return;
139 }
140
141 int main(void)
142 {
143   // Enabling the leds and other stuff
144   WDTCTL = WDTPW | WDTHOLD; // Stop WDT
145   PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
146
147   // Setting ACLK to 32KHz
148   config_ACLK_to_32KHz_crystal();
149
150   // doing what the function says
151   initialize_uart();
152
153   // Printing 'Hello World!!' to the console
154   uart_write_string("Hello World!!");
155   uart_write_char('\n');
156   uart_write_char('\r');
157
158   uart_write_uint16(65432);
159   uart_write_char('\n');
160   uart_write_char('\r');
161 }
```

6.4 Application: Airport Runway Control

```c
#include <msp430fr6989.h>
#include <stdint.h>
#include <string.h>

// LED BITS
#define red BIT0
#define green BIT7

// Button BITS
#define but1 BIT1
#define but2 BIT2

// UART Channels are P3.4 and P3.5 for transmit and recieve respectively
#define transmit BIT4
#define recieve BIT5

// WE LOVE DEFINES
#define FLAGS UCA1IFG // Contains the transmit & receive flags
#define RXFLAG UCRXIFG // Receive flag
#define TXFLAG UCTXIFG // Transmit flag
#define TXBUFFER UCA1TXBUF // Transmit buffer
#define RXBUFFER UCA1RXBUF // Receive buffer

// Global variables for states of runway1 and 2
volatile int red_state = 0; // runway 1 state
volatile int green_state = 0; // runway 2 state
volatile int blink_state = 0;

// Reverses a given string
void strrev(char *str)
{
  int i = 0;
  int j = strlen(str) - 1;
  char temp;
  while (i < j)
  {
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
  }
}

// Converts an unsigned 16-bit integer to a null-terminated string (base 10).
void custom_itoa(uint16_t number, char *buffer)
{
  int i = 0;

  // Handle the special case of 0
  if (number == 0)
  {
    buffer[i++] = '0';
```

```
54      buffer[i] = '\0';
55      return;
56    }
57
58    // Process individual digits
59    while (number > 0)
60    {
61      int remainder = number % 10;
62      buffer[i++] = remainder + '0'; // Convert digit to its ASCII
        character
63      number = number / 10;
64    }
65
66    buffer[i] = '\0'; // Null-terminate the string
67
68    // The digits are in reverse order, so we need to reverse the
        string
69    strrev(buffer);
70  }
71
72  // Configures ACLK to 32 KHz crystal
73  void config_ACLK_to_32KHz_crystal()
74  {
75    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
76
77    // Reroute pins to LFXIN/LFXOUT functionality
78    PJSEL1 &= ~BIT4;
79    PJSEL0 |= BIT4;
80
81    // Wait until the oscillator fault flags remain cleared
82    CSCTL0 = CSKEY; // Unlock CS registers
83    do
84    {
85      CSCTL5 &= ~LFXTOFFG; // Local fault flag
86      SFRIFG1 &= ~OFIFG; // Global fault flag
87    }
88    while((CSCTL5 & LFXTOFFG) != 0);
89
90    CSCTL0_H = 0; // Lock CS registers
91    return;
92  }
93
94  void initialize_uart(void)
95  {
96    // Configuring the pins to use backchannel uart
97    P3SEL1 &= ~(transmit | recieve);
98    P3SEL0 |= (transmit | recieve);
99
100   // Setting the clock to SMCLK
101   UCA1CTLW0 |= UCSSEL_2;
102
103   // Setting the dividers and enabling oversampling
104   UCA1BRW = 6;
105   // setting the modulators and such
106   UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;
107
108   // Exiting the reset state
```

```
109   UCA1CTLW0 &= ~UCSWRST;
110 }
111
112 void uart_write_char(volatile unsigned char ch)
113 {
114   while (!(FLAGS & TXFLAG))
115   {
116     // Wait for transmission that is ongoing to complete
117   }
118
119   TXBUFFER = ch;
120   return;
121 }
122
123 unsigned char uart_read_char(void)
124 {
125   if (!(FLAGS & RXFLAG))
126   {
127     return 0; // no byte was recieved
128   }
129
130   // Return the buffer
131   volatile unsigned char return_char = RXBUFFER;
132   return return_char;
133 }
134
135 void uart_write_string(char *string)
136 {
137   int i; // counter
138   for (i = 0; i < strlen(string); i++)
139   {
140     uart_write_char(string[i]);
141   }
142   return;
143 }
144
145 void uart_write_uint16 (uint16_t number)
146 {
147   // Converting the number via snprintf
148   char buffer[6]; // 5 characters is the max amount of characters
        for 65,536
149   custom_itoa(number, buffer);
150   uart_write_string(buffer);
151   return;
152 }
153
154 int main(void)
155 {
156   // Enabling the leds and other stuff
157   WDTCTL   = WDTPW | WDTHOLD; // Stop WDT
158   PM5CTL0 &= ~LOCKLPM5;       // Enable GPIO pins
159
160   // Configuring the LEDs as outputs and off
161   P1DIR |= red;
162   P9DIR |= green;
163   P1OUT &= ~red;
164   P9OUT &= ~green;
```

```
165
166 // Defining the buttons as inputs and their resistors as pull-up
167 P1DIR &= ~(but1 | but2);
168 P1REN |= (but1 | but2);
169 P1OUT |= (but1 | but2);
170
171 // Setting up button interrupts on FALLING EDGE
172 // P1IES |= (but1 | but2);
173 P1IE  |= (but1 | but2);
174 P1IFG &= ~(but1 | but2);
175
176 // Setting ACLK to 32KHz
177 config_ACLK_to_32KHz_crystal();
178
179 // Setting up the timer
180 TA0CCR0   = 32767;  // 1 second interrupts
181 TA0CCTL0 = CCIE;   // Enabling capture control interrupt
182 TA0CCTL0 &= ~CCIFG; // Clearing interrupt flag (if there)
183 // ACLK is used, no divider, up mode, and clear the current
      stored time
184 TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
185 // Sanity check for timer_a interrupt flag
186 TA0CTL &= ~TAIFG;
187
188 // doing what the function says
189 initialize_uart();
190
191 // Since we aren't using low power mode
192 _enable_interrupt();
193
194 // I think you'll know what this does...
195 uart_write_string("\033[m\033[1;1H\033[2J"
196                   "ORLANDO EXECUTIVE AIRPORT RUNWAY CONTROL
      \n"
197                   "\n"
198                   "\n"
199                   "                  Runway 1    Runway 2\n"
200                   "Request (RQ):       1           3\n"
201                   "Forfeit (FF):       7           9\n"
202                   "\n"
203                   "\n"
204                   "\n"
205                   "--------
      --------\n"
206                   "RUNWAY 1                        RUNWAY
      2\n"
207                   "--------
      --------\n"
208                   "\n"
209                   "\n"
210                   "\n"
211                   "\n"
212                   );
213
214 // Actual logic for selection
215 for (;;)
216 {
```

```
217     unsigned char given_char = uart_read_char();
218
219     switch(given_char)
220     {
221       case '1':
222         uart_write_string("\0337\033[13;HRequested\0338");
223         P1OUT |= red;
224         red_state = 1;
225         break;
226       case '3':
227         uart_write_string("\0337\033[13;34HRequested\0338");
228         P9OUT |= green;
229         green_state = 1;
230         break;
231       case '7':
232         uart_write_string("\0337\033[13;H                \033[14;H
                      \033[16;H                \0338");
233         P1OUT &= ~red;
234         red_state = 0;
235         blink_state = 0;
236         break;
237       case '9':
238         uart_write_string("\0337\033[13;34H
    \033[14;34H                \033[16;34H                \0338")
    ;
239         P9OUT &= ~green;
240         green_state = 0;
241         blink_state = 0;
242         break;
243     }
244
245     __delay_cycles(25000);
246   }
247 }
248
249 // button interrupts
250 #pragma vector = PORT1_VECTOR
251 __interrupt void P1_ISR()
252 {
253   if (P1IFG & but1)
254   {
255     if (red_state)
256     {
257       if(blink_state == 3 || blink_state == 4)
258       {
259         // Do Nothing
260       }
261       else if (blink_state == 2)
262       {
263         blink_state = 1;
264       }
265       else
266       {
267         blink_state = 2;
268       }
269     }
270     P1IFG &= ~but1;
```

```
271   }
272
273   if (P1IFG & but2)
274   {
275     if (green_state)
276     {
277       if(blink_state == 1 || blink_state == 2)
278       {
279         // Do Nothing
280       }
281       else if (blink_state == 4)
282       {
283         blink_state = 3;
284       }
285       else
286       {
287         blink_state = 4;
288       }
289     }
290     P1IFG &= ~but2;
291   }
292
293   __delay_cycles(200000); // 200 MS
294 }
295
296 #pragma vector = TIMER0_A0_VECTOR
297 __interrupt void TA0_ISR()
298 {
299   switch (blink_state)
300   {
301     case 1:
302       uart_write_string("\0337\033[16;H*** Inquiry ***      \0338")
    ;
303       // We then need to do #2
304     case 2:
305       P1OUT ^= red;
306       uart_write_string("\0337\033[14;HIn Use       \033[14;34H
              \0338");
307       if (green_state)
308       {
309         P9OUT |= green;
310       }
311       else
312       {
313         P9OUT &= ~green;
314       }
315       break;
316     case 3:
317       uart_write_string("\0337\033[16;34H*** Inquiry ***      \0338
    ");
318       // We need to do #3
319     case 4:
320       P9OUT ^= green;
321       uart_write_string("\0337\033[14;34HIn Use       \033[14;H
              \0338");
322       if (red_state)
323       {
```

```
324           P1OUT |= red;
325         }
326         else
327         {
328           P1OUT &= ~red;
329         }
330         break;
331     }
332 }
```

Student Q&A

1

Given: What's the difference between UART and eUSCI?

UART is a general serial protocol while the eUSCI is an above wrapper that can do UART, SPI, and I2C. It encompasses and enables all 3 forms of communication.

2

Given: What is the backchannel UART?

The backchannel UART is connected to the eUSCI module 1 in channel A.

3

Given: What's the function of the two lines of code that have P3SEL1 and P3SEL0?

These lines specifically change the pin multiplexers that allow you to use the UART module externally.

4

Given: The microcontroller has a clock at the frequency of 1,000,000 Hz and we're aiming to setup a UART connection at 9600 baud. How do we obtain a clock rate of 9600 Hz? Explain the approach at a high level.

When referencing the reference manual for the MSP430, we are shown a table that gives us the recommended values of the registers for given inputs and output frequencies. In that table it states what each register should be given if you want oversampling or not.

5

Given: A UART transmitter is transmitting data at at 1200 baud. What is receiver's clock frequency if oversampling is not used?

The reciever clock will have the exact same frequency as the transmitter, therefore it is 1200Hz.

6

Given: A UART transmitter is transmitting data at at 1200 baud. What is receiver's clock frequency if oversampling is used? What's the benefit of oversampling?

It depends on the amount of oversampling that can even be configured on the reciever. Assumingn we use a default of 16x oversampling, then the hclock rate would be 19,200Hz. The reason we oversample to begin with is to compensate for any offsets or drifts in the transmitter clock over time as well as to eliminate as much noise in the signal given out by averaging the reads of the signal over multiple samples.