```
// Serial Example
// Jason Losh

//-----------------------------------------------------------------------------
// Hardware Target
//-----------------------------------------------------------------------------

// Target Platform: EK-TM4C123GXL Evaluation Board
// Target uC:       TM4C123GH6PM
// System Clock:    40 MHz

// Hardware configuration:
// Red Backlight LED:
//   PB5 drives an NPN transistor that powers the red LED
// Green Backlight LED:
//   PE4 drives an NPN transistor that powers the green LED
// Blue Backlight LED:
//   PE5 drives an NPN transistor that powers the blue LED
// Red LED:
//   PF1 drives an NPN transistor that powers the red LED
// Green LED:
//   PF3 drives an NPN transistor that powers the green LED
// Pushbutton:
//   SW1 pulls pin PF4 low (internal pull-up is used)
// UART Interface:
//   U0TX (PA1) and U0RX (PA0) are connected to the 2nd controller
//   The USB on the 2nd controller enumerates to an ICDI interface and a virtual COM port
//   Configured to 115,200 baud, 8N1

//-----------------------------------------------------------------------------
// Device includes, defines, and assembler directives
//-----------------------------------------------------------------------------

#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include "tm4c123gh6pm.h"

#define RED_LED      (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32 + 1*4)))
#define GREEN_LED    (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32 + 3*4)))
#define PUSH_BUTTON  (*((volatile uint32_t *)(0x42000000 + (0x400253FC-0x40000000)*32 + 4*4)))

//-----------------------------------------------------------------------------
// Subroutines
//-----------------------------------------------------------------------------

// Blocking function that returns only when SW1 is pressed
void waitPbPress()
{
    while(PUSH_BUTTON);
}

// Initialize Hardware
void initHw()
{
    // Configure HW to work with 16 MHz XTAL, PLL enabled, system clock of 40 MHz
```

```
57    SYSCTL_RCC_R = SYSCTL_RCC_XTAL_16MHZ | SYSCTL_RCC_OSCSRC_MAIN | SYSCTL_RCC_USESYSDIV | (4
  << SYSCTL_RCC_SYSDIV_S);
58
59    // Set GPIO ports to use APB (not needed since default configuration -- for clarity)
60    // Note UART on port A must use APB
61    SYSCTL_GPIOHBCTL_R = 0;
62
63    // Enable GPIO port A and F peripherals
64    SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOA | SYSCTL_RCGC2_GPIOF;
65
66    // Configure LED and pushbutton pins
67    GPIO_PORTF_DIR_R = 0x0A;  // bits 1 and 3 are outputs, other pins are inputs
68    GPIO_PORTF_DR2R_R = 0x0A; // set drive strength to 2mA (not needed since default
  configuration -- for clarity)
69    GPIO_PORTF_DEN_R = 0x1A;  // enable LEDs and pushbuttons
70    GPIO_PORTF_PUR_R = 0x10;  // enable internal pull-up for push button
71
72    // Configure UART0 pins
73    SYSCTL_RCGCUART_R |= SYSCTL_RCGCUART_R0;         // turn-on UART0, leave other uarts in
  same status
74    GPIO_PORTA_DEN_R |= 3;                           // default, added for clarity
75    GPIO_PORTA_AFSEL_R |= 3;                         // default, added for clarity
76    GPIO_PORTA_PCTL_R = GPIO_PCTL_PA1_U0TX | GPIO_PCTL_PA0_U0RX;
77
78    // Configure UART0 to 115200 baud, 8N1 format (must be 3 clocks from clock enable and
  config writes)
79    UART0_CTL_R = 0;                                 // turn-off UART0 to allow safe
  programming
80    UART0_CC_R = UART_CC_CS_SYSCLK;                  // use system clock (40 MHz)
81    UART0_IBRD_R = 21;                               // r = 40 MHz / (Nx115.2kHz), set
  floor(r)=21, where N=16
82    UART0_FBRD_R = 45;                               // round(fract(r)*64)=45
83    UART0_LCRH_R = UART_LCRH_WLEN_8 | UART_LCRH_FEN; // configure for 8N1 w/ 16-level FIFO
84    UART0_CTL_R = UART_CTL_TXE | UART_CTL_RXE | UART_CTL_UARTEN; // enable TX, RX, and module
85}
86
87// Blocking function that writes a serial character when the UART buffer is not full
88void putcUart0(char c)
89{
90    while (UART0_FR_R & UART_FR_TXFF);
91    UART0_DR_R = c;
92}
93
94// Blocking function that writes a string when the UART buffer is not full
95void putsUart0(char* str)
96{
97    int i;
98    for (i = 0; i < strlen(str); i++)
99      putcUart0(str[i]);
100}
101
102// Blocking function that returns with serial data once the buffer is not empty
103char getcUart0()
104{
105    while (UART0_FR_R & UART_FR_RXFE);
106    return UART0_DR_R & 0xFF;
```

```
107}
108
109//-----------------------------------------------------------------------------
110// Main
111//-----------------------------------------------------------------------------
112
113int main(void)
114{
115    // Initialize hardware
116    initHw();
117
118    // Display greeting
119    putsUart0("Serial Example\r\n");
120    putsUart0("Press '0' or '1'\r\n");
121    putcUart0('a');
122
123    // Wait for PB press
124    waitPbPress();
125
126    // For each received character, toggle the green LED
127    // For each received "1", set the red LED
128    // For each received "0", clear the red LED
129    while(1)
130    {
131        char c = getcUart0();
132        GREEN_LED ^= 1;
133        if (c == '1')
134            RED_LED = 1;
135        if (c == '0')
136            RED_LED = 0;
137    }
138}
139
```