```c
// UART0 Library
// Jason Losh

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

// Target Platform: EK-TM4C123GXL
// Target uC:       TM4C123GH6PM
// System Clock:    -

// Hardware configuration:
// 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

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

#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
#include "uart0.h"
#include "gpio.h"

// Pins
#define UART_TX PORTA,1
#define UART_RX PORTA,0

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------

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

// Initialize UART0
void initUart0()
{
    // Enable clocks
    SYSCTL_RCGCUART_R |= SYSCTL_RCGCUART_R0;
    _delay_cycles(3);
    enablePort(PORTA);

    // Configure UART0 pins
    selectPinPushPullOutput(UART_TX);
    selectPinDigitalInput(UART_RX);
    setPinAuxFunction(UART_TX, GPIO_PCTL_PA1_U0TX);
    setPinAuxFunction(UART_RX, GPIO_PCTL_PA0_U0RX);

    // Configure UART0 with default baud rate
    UART0_CTL_R = 0;                                    // turn-off UART0 to allow safe
    programming
    UART0_CC_R = UART_CC_CS_SYSCLK;                     // use system clock (usually 40
    MHz)
}

// Set baud rate as function of instruction cycle frequency
void setUart0BaudRate(uint32_t baudRate, uint32_t fcyc)
{
    uint32_t divisorTimes128 = (fcyc * 8) / baudRate;   // calculate divisor (r) in
    units of 1/128,
                                                        // where r = fcyc / 16 * baudRate
    UART0_CTL_R = 0;                                    // turn-off UART0 to allow safe
    programming
    UART0_IBRD_R = divisorTimes128 >> 7;                // set integer value to floor(r)
    UART0_FBRD_R = ((divisorTimes128 + 1) >> 1) & 63;   // set fractional value to
```

```
    round(fract(r)*64)
    UART0_LCRH_R = UART_LCRH_WLEN_8 | UART_LCRH_FEN;    // configure for 8N1 w/ 16-level FIFO
    UART0_CTL_R = UART_CTL_TXE | UART_CTL_RXE | UART_CTL_UARTEN;
                                                        // turn-on UART0
}

// Blocking function that writes a serial character when the UART buffer is not full
void putcUart0(char c)
{
    while (UART0_FR_R & UART_FR_TXFF);               // wait if uart0 tx fifo full
    UART0_DR_R = c;                                  // write character to fifo
}

// Blocking function that writes a string when the UART buffer is not full
void putsUart0(char* str)
{
    uint8_t i = 0;
    while (str[i] != '\0')
        putcUart0(str[i++]);
}

// Blocking function that returns with serial data once the buffer is not empty
char getcUart0()
{
    while (UART0_FR_R & UART_FR_RXFE);               // wait if uart0 rx fifo empty
    return UART0_DR_R & 0xFF;                        // get character from fifo
}

// Returns the status of the receive buffer
bool kbhitUart0()
{
    return !(UART0_FR_R & UART_FR_RXFE);
}

```