```
// SPI1 Library
// Jason Losh

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

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

// Hardware configuration:
// SPI1 Interface:
//   MOSI on PD3 (SSI1Tx)
//   MISO on PD2 (SSI1Rx)
//   ~CS on PD1  (SSI1Fss)
//   SCLK on PD0 (SSI1Clk)

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

#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
#include "spi1.h"
#include "gpio.h"

// Pins
#define SSI1TX PORTD,3
#define SSI1RX PORTD,2
#define SSI1FSS PORTD,1
#define SSI1CLK PORTD,0

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

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

// Initialize SPI1
void initSpi1(uint32_t pinMask)
{
    // Enable clocks
    SYSCTL_RCGCSSI_R |= SYSCTL_RCGCSSI_R1;
    _delay_cycles(3);
    enablePort(PORTD);

    // Configure SSI1 pins for SPI configuration
    selectPinPushPullOutput(SSI1TX);
    setPinAuxFunction(SSI1TX, GPIO_PCTL_PD3_SSI1TX);
    selectPinPushPullOutput(SSI1CLK);
    setPinAuxFunction(SSI1CLK, GPIO_PCTL_PD0_SSI1CLK);
    enablePinPullup(SSI1CLK);
    selectPinPushPullOutput(SSI1FSS);
    if (pinMask & USE_SSI_FSS)
    {
        setPinAuxFunction(SSI1FSS, GPIO_PCTL_PD1_SSI1FSS);
    }
    if (pinMask & USE_SSI_RX)
    {
        selectPinDigitalInput(SSI1RX);
        setPinAuxFunction(SSI1RX, GPIO_PCTL_PD2_SSI1RX);
    }

    // Configure the SSI1 as a SPI master, mode 3, 8bit operation
    SSI1_CR1_R &= ~SSI_CR1_SSE;                       // turn off SSI1 to allow
```

```
    re-configuration
70      SSI1_CR1_R = 0;                                 // select master mode
71      SSI1_CC_R = 0;                                  // select system clock as the
    clock source
72      SSI1_CR0_R = SSI_CR0_FRF_MOTO | SSI_CR0_DSS_8;  // set SR=0, 8-bit
73  }
74
75  // Set baud rate as function of instruction cycle frequency
76  void setSpi1BaudRate(uint32_t baudRate, uint32_t fcyc)
77  {
78      uint32_t divisorTimes2 = (fcyc * 2) / baudRate;  // calculate divisor (r) times 2
79      SSI1_CR1_R &= ~SSI_CR1_SSE;                      // turn off SSI1 to allow
    re-configuration
80      SSI1_CPSR_R = (divisorTimes2 + 1) >> 1;          // round divisor to nearest
    integer
81      SSI1_CR1_R |= SSI_CR1_SSE;                       // turn on SSI1
82  }
83
84  // Set mode
85  void setSpi1Mode(uint8_t polarity, uint8_t phase)
86  {
87      SSI1_CR1_R &= ~SSI_CR1_SSE;                      // turn off SSI1 to allow
    re-configuration
88      SSI1_CR0_R &= ~(SSI_CR0_SPH | SSI_CR0_SPO);      // set SPO and SPH as appropriate
89      if (polarity) SSI1_CR0_R |= SSI_CR0_SPO;
90      if (phase) SSI1_CR0_R |= SSI_CR0_SPH;
91      SSI1_CR1_R |= SSI_CR1_SSE;                       // turn on SSI1
92  }
93
94  // Blocking function that writes data and waits until the tx buffer is empty
95  void writeSpi1Data(uint32_t data)
96  {
97      SSI1_DR_R = data;
98      while (SSI1_SR_R & SSI_SR_BSY);
99  }
100
101 // Reads data from the rx buffer after a write
102 uint32_t readSpi1Data()
103 {
104     return SSI1_DR_R;
105 }
106
```