Application Note

Using SPI to Create a Configurable System

Author: Stefan Turkowski

Serial Peripheral Interface Protocol is used in many systems to control all kinds of subsystems. A protocol is simply a set of rules put in place to ensure that every circuit using SPI will be able to be controlled. If your code and your circuit follows these rules, you should have no trouble controlling many devices with one microcontroller. See Wikipedia for more information.

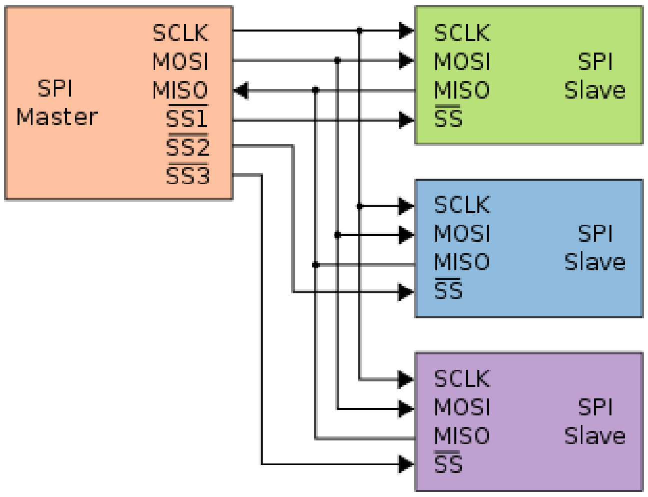
Benefits

- On-the-fly parameter adjustments (Such as digital potentiometers)
- DAC operation made easy
- IC control (such as Infineon BGT24 or power amplifiers)

Links to useful Chips

- Single potentiometer
- Dual potentiometer
- Digital to Analog Converter (Although a DAC with 16 bits instead of 12 is recommended because will produce a cleaner triangle wave)
- Infineon BGT24

How it Works



Understanding the diagram above

- Your microcontroller is the “master” because it controls the clock and slave selects.

- Your digital potentiometers and DAC are your “slaves”. You can only tell one slave what to do at one time. You control which slave you are talking to with the slave select (chip select, SS, ect…) pins.
- Master out, slave in (MOSI) is used to transmit data from the master to the slave.
- Master in, slave out(MISO) is used to transmit data from the slave to the master. You will probably not need to use this pin though.

Communicating

- You must set SS low to talk to that slave.
- To talk to a slave, you may use a command like:
  - SSIDataPutNonBlocking(SSI2_BASE, character); *This is included in TivaWare, which you should download.*
  - writeData(0x18); *A function found in our code, which sends 00011000 to the slave. It is recommended that you use this function.*
- The commands will start the clock, send data at the clock rate, and turn off the clock. The data is loaded into internal registers on each IC, so that the data is stored even after the clock is turned off. Every slave that has its SS high will ignore all data sent.
- You must set SS high once you are done talking to a slave.

Initializing Everything

- Initializing all the peripherals on a Tiva is the most difficult part. It must be done before any data can be sent. If you use the code that I provide, and understand which ports and pins you are using, it should not be a problem. Here are the important lines of code:
- ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2); enables SSI port 2. There is also SSI port 0 and SSI port 1.
- ROM_SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ | SYSCTL_OSC_MAIN); The internal clock is 200 MHz, but the highest division you can use is 2.5.
- SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, SysCtlClockGet()/20, 8); creates the settings for SSI2. SysCtlClockGet()/2 is the fastest that the SSI can operate (at half the speed of the processor), but it is recommended to use a higher division because some SPI devices cannot operate at such a high clock frequency anyway. This SSI is set to 8 bits, but 4 and 16 can also be used. If 16 bits is used, you can use writeData(0xF5AB);

Before You Design Your PCB

- BREADBOARD TEST EVERYTHING. Understand how communicating with your ICs works before making final decisions.
- Make sure that your SPI traces to not go under any signals, especially RF signals. SPI generates a lot of noise, and it should be turned off when possible.
- If possible, use SPI1 and SPI2, because talking to multiple devices with one SPI line does not always work out as cleanly as it sounds.

Caution

- Each IC has its own requirements for SPI, including max clock rate and mode:
    - You must send data under a certain frequency for each chip. This can be changed on the fly if you need to with SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, **SysCtlClockGet()/50**, 8);
    - SPI mode determines if the data is loaded in on the rising or falling edge of the clock. There are four different modes. Most chips appear to be mode 0, but you should definitely double check the data sheet.

Tiva or Stellaris Code

```
/*
Triangle wave and sync pulse generator to control a VCO for FMCW radar. This code also initializes the BGT24,
and set a digital potentiometer for coarse tune.
The MPC4921 DAC is used to generate a triangle wave with a period of 40ms.
For use with Tiva or Stellaris Launchpad

Stefan and Joe
*/

 #include <stdint.h>
 #include <stdbool.h>
 #include "inc/tm4c123gh6pm.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "driverlib/sysctl.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/ssi.h"
```

```
#define NUM_SSI_DATA                                    1

 uint32_t pui32DataTx[NUM_SSI_DATA];
 uint32_t pui32DataRx[NUM_SSI_DATA];
 uint32_t ui32Index;

void writeCommand(uint8_t c)                    //Not needed in this code, but might be useful later.
            {
            while(ROM_SSIDataGetNonBlocking(SSI2_BASE, &pui32DataRx[0]))
    {
    }
            // We want to set DC low because we are trying to write a command
            //ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, 0);
                                    NOT NEEDED

            // We only need to put(). We don't need to get() because we are not
            // looking for information from the Display
  SSIDataPutNonBlocking(SSI2_BASE, c);
            while(SSIBusy(SSI2_BASE))                                    //Hopefully this step will not be
needed when integrated into a code running with interrupts
    {
    }
}

void writeData(uint8_t c)
            {

            while(ROM_SSIDataGetNonBlocking(SSI2_BASE, &pui32DataRx[0]))
    {
    }
            // If we want to set DC (not direct current) high because we are trying to write data
            // DC is a GPIO: We are choosing PB6
            //ROM_GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_PIN_6);                    Not
needed
            // We only need to put(). We don't need to get() because we are not
            // looking for information from the OLED
  SSIDataPut(SSI2_BASE, c);                                    //Does the whole SPI process EXCEPT
Slave Select low
            while(SSIBusy(SSI2_BASE))                                    //Hopefully this step will not be
needed when integrated into a code running with interrupts
  {
  }
}

char data = 0;// A byte is an 8-bit number
unsigned int outputValue = 0;// A word is a 16-bit number
```

```
void setup()
{

            ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI2);          //for CLK      and
data
            ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);   //enable  slave select ports for DAC
and Infineon


            GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6); //Enable DAC SS pin as GPIO
            GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_4); //Enable Infineon SS pin as
GPIO
            GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_5); //Enable Digital Pot SS pin as
GPIO


            ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);         //Still enabling

            GPIOPinConfigure(GPIO_PB7_SSI2TX);           //MOSI transmit data
            //GPIOPinConfigure(GPIO_PB5_SSI2FSS);        //Predefined SS not needed

            GPIOPinTypeSSI(GPIO_PORTB_BASE, GPIO_PIN_4 | GPIO_PIN_7);
                            // Configure SSI Pins


            ROM_SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL |
SYSCTL_XTAL_16MHZ |
          SYSCTL_OSC_MAIN);                                   //80 MHz
            SSIConfigSetExpClk(SSI2_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, SysCtlClockGet()/20, 8); //Running SSI clock at 40 MHz. Should be too fast...
       GPIOPinConfigure(GPIO_PB4_SSI2CLK);               //Clock COnfigure
            ROM_SSIEnable(SSI2_BASE);

            //GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_5, 0);          //Set predefined SS to 0 (if
we used it)
            ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);   //Enable onboard LED port
            //ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1);     //Enable onboard
LED pins for testing
            //ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
            //ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0);


            //ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, 0);           //SS low for Pot
            //writeData(0x00);       //For Pot
            //ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, GPIO_PIN_5);         //SS
High for Pot
```

```
                    //Turning on Infineon
                              ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4, 0);
          //SS low

                              writeData(0x00);
          // Send the upper byte for BGT24


                              writeData(0x18);
          // Send the lower byte for BGT24

                              ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4, GPIO_PIN_4);
          //SS High
}


int highByte(int x)
{
          return (0xff00 & x)>>8;
}

int lowByte(int x)
{
          return (0x00ff & x);
}
int main(void)
{
          setup();

          for(;;)
          {
                    int a = 0;
   // Rising edge of the triangle wave
   while (a <= 4080)
   {
                              GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //Turn
LED on for testing Purposes

      outputValue = a;

                              ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0);
          //SS low

      data = highByte(outputValue);              // Take the upper byte
      data = 0x0F & data;
      data = 0x30 | data;
```

```
                                        writeData(data);
            // Send the upper byte

        data =lowByte(outputValue);                // Shift in the 8 lower bits

                                        writeData(data);
            // Send the lower byte

                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
            //SS High

                                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 0); //Turn LED off for
testing Purposes

                                a = a+64;
    }

    // Falling edge of the triangle wave, very similar to the above
                        int b = 4080;
    while (b >= 64)
    {
                                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, GPIO_PIN_2); //Turn
LED on for testing Purposes

                                 outputValue = b;

                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, 0);
            //SS LOW
        data =highByte(outputValue);
        data = 0x0F & data;
        data = 0x30 | data;

                                        writeData(data);
        data =lowByte(outputValue);

                                        writeData(data);

                                        ROM_GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, GPIO_PIN_6);
            //SS HIGH

                                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 1); //Turn LED on for
testing Purposes

                                b = b-64;
    }
            }
```

}