Name ___________________________

Lab Section _____________________

PIC – Serial Peripheral Interface (SPI) to Digital Pot

Lab 6

Introduction: SPI is a popular synchronous serial communication protocol that allows ICs to communicate over short distances (PCB level communication). In this lab you will investigate using the Serial Peripheral Interface (SPI) bus to communicate with a peripheral that is external to the microcontroller.

Lab Requirements:

1. Demonstrate the use of SPI communication to control the wiper position of a digital potentiometer. Connect the digital potentiometer as a programmable attenuator and show how you can control the amplitude of a 5Vpp (0 to 5V) sine wave by sending SPI commands.

Demo Check (JK)______

About Serial Peripheral Interface:

SPI is a full-duplex synchronous serial communication protocol for data transfer between integrated circuits on a PCB. The protocol uses a master/ slave relationship where the master initiates all data transfers and generates the data synchronization clock. The standard bus connections are a Serial Clock (SCK), Master-Out-Slave-In (MOSI), Master-In-Slave-Out (MISO) and an active low Slave Select (nSS). Each slave device has its own Slave Select (nSS) signal which dictates when the device is participating in bus transfers. A typical connection diagram is shown below:

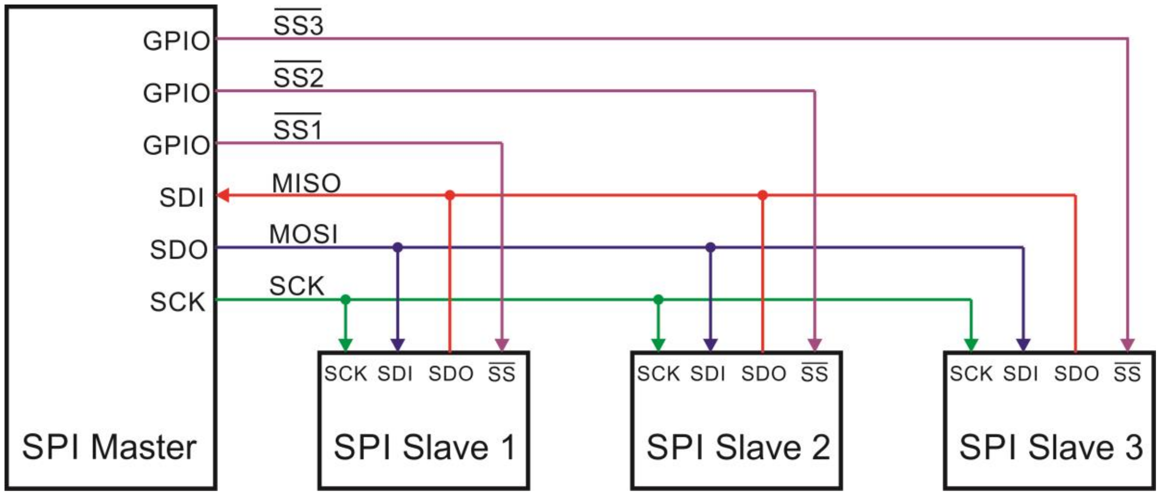


Figure 1 - SPI Connection Diagram

The SPI protocol operates as a shift register where data is transferred from the master to the slave and from the slave to the master at the same time. The data is shifted one bit at a time on the SCK. Unfortunately, there is no standardization on the clock edge for data shift or on the clock polarity so four different modes are possible. When connecting to a new peripheral it is important to study the waveform timing diagram to determine the clock polarity and phase.

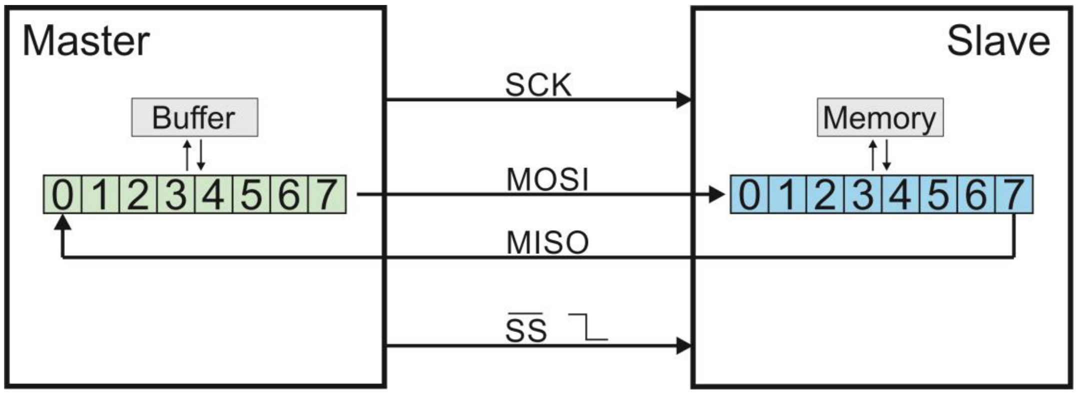


Figure 2 - SPI Shift Register

Before a data transfer begins the master must assert the nSS signal by driving it low. Then the data will be exchanged one bit at a time between the master and the slave. The Data Transfer diagram below illustrates a Mode 3 transfer where the SCK idles high and data is valid on the rising edge of the SCK. At the end of the data transfer the nSS signal is brought high to deselect the peripheral.

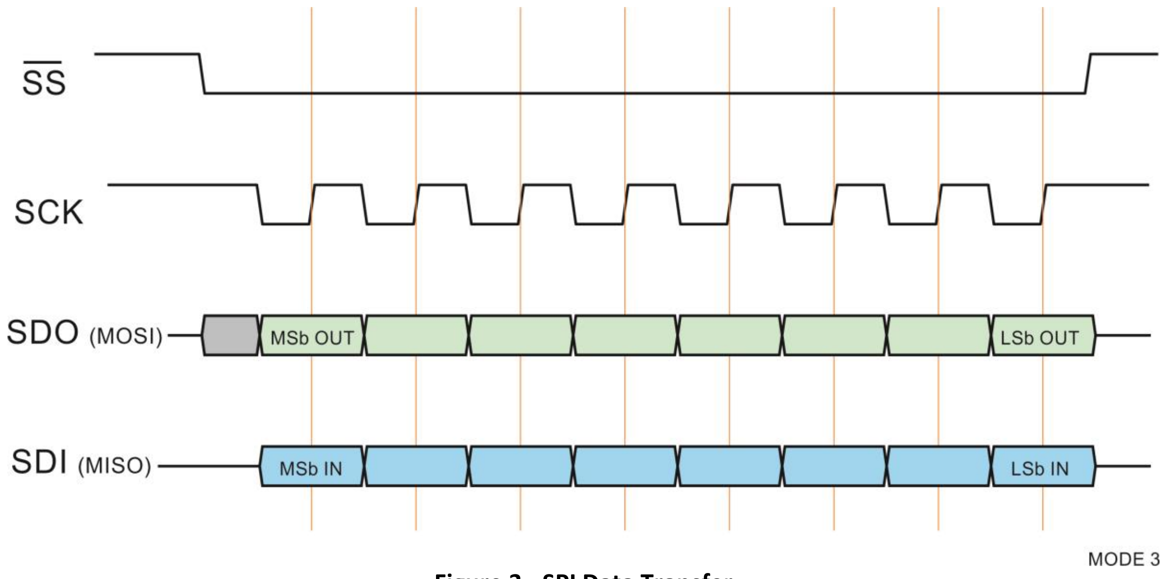


Figure 3 - SPI Data Transfer

Most modern microcontrollers contain dedicated hardware to implement a master or slave peripheral interface. In the PIC family of microcontrollers this functionality is located in the Master Synchronous Serial Port (MSSP) Module. The MSSP is capable of being configured to implement either the SPI or I2C bus protocol. The hardware can be configured to operate as either a master or as a slave device. Today we will be setting up the MSSP for SPI Master Mode operation.

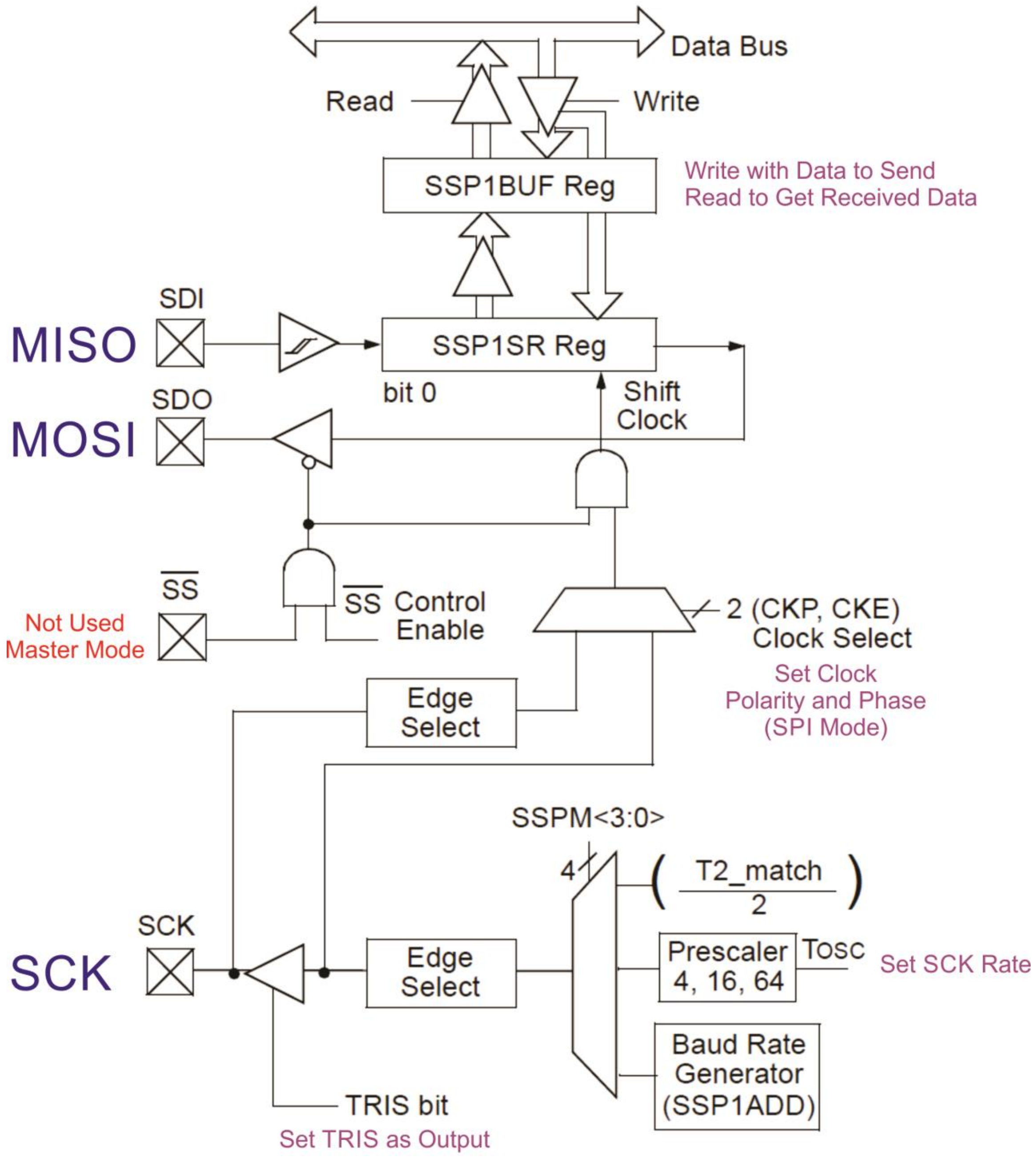
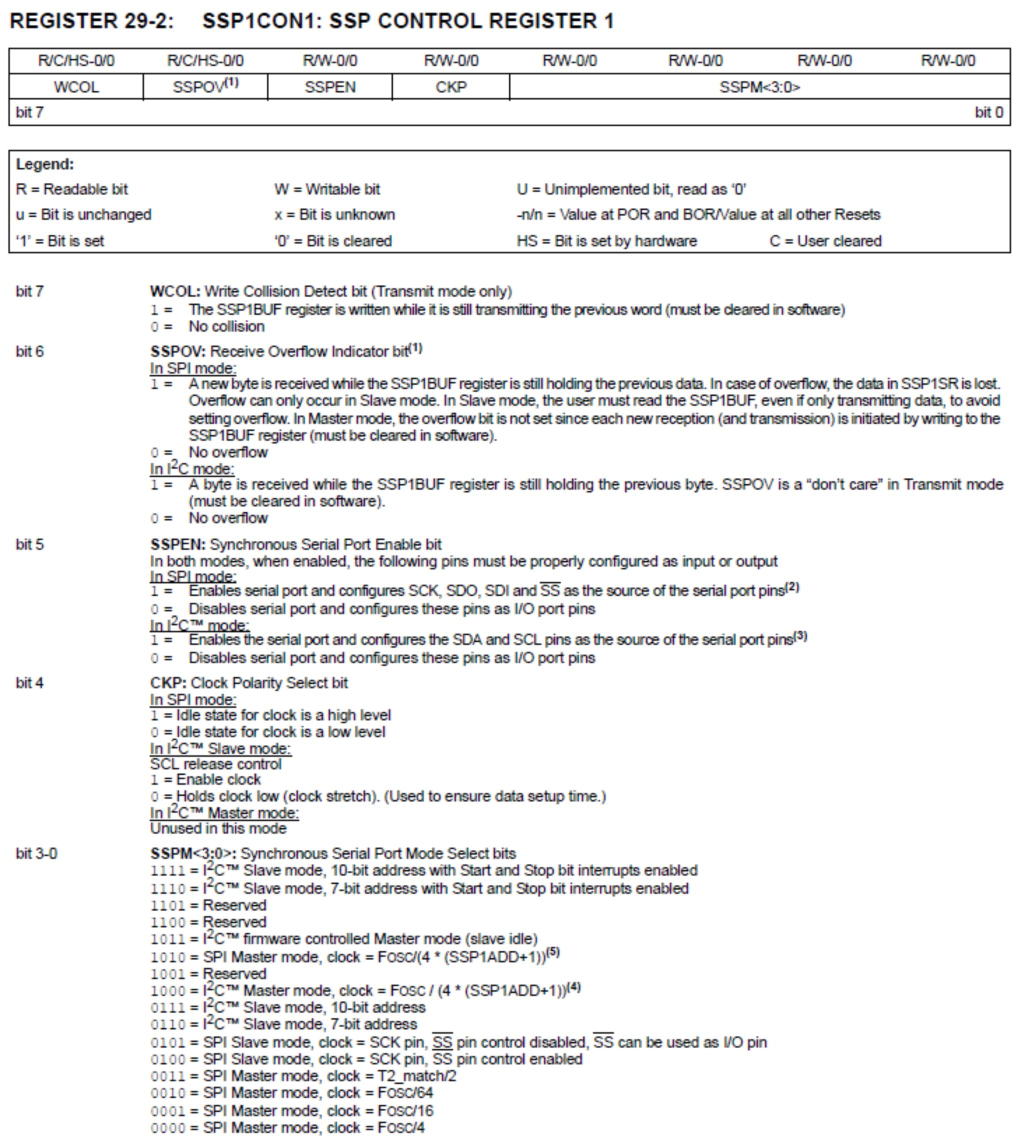


Figure 4 - MSSP SPI Block Diagram



REGISTER 29-2: SSP1CON1: SSP CONTROL REGISTER 1

R/C/HS-0/0	R/C/HS-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
WCOL	SSPOV[1]	SSPEN	CKP	SSPM<3:0>			
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
'1' = Bit is set	'0' = Bit is cleared	HS = Bit is set by hardware C = User cleared

bit 7 **WCOL:** Write Collision Detect bit (Transmit mode only)
1 = The SSP1BUF register is written while it is still transmitting the previous word (must be cleared in software)
0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit[1]
In SPI mode:
1 = A new byte is received while the SSP1BUF register is still holding the previous data. In case of overflow, the data in SSP1SR is lost. Overflow can only occur in Slave mode. In Slave mode, the user must read the SSP1BUF, even if only transmitting data, to avoid setting overflow. In Master mode, the overflow bit is not set since each new reception (and transmission) is initiated by writing to the SSP1BUF register (must be cleared in software).
0 = No overflow
In I^{2}C mode:
1 = A byte is received while the SSP1BUF register is still holding the previous byte. SSPOV is a "don't care" in Transmit mode (must be cleared in software).
0 = No overflow

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
In both modes, when enabled, the following pins must be properly configured as input or output
In SPI mode:
1 = Enables serial port and configures SCK, SDO, SDI and $\overline{SS}$ as the source of the serial port pins[2]
0 = Disables serial port and configures these pins as I/O port pins
In I^{2}C™ mode:
1 = Enables the serial port and configures the SDA and SCL pins as the source of the serial port pins[3]
0 = Disables serial port and configures these pins as I/O port pins

bit 4 **CKP:** Clock Polarity Select bit
In SPI mode:
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level
In I^{2}C™ Slave mode:
SCL release control
1 = Enable clock
0 = Holds clock low (clock stretch). (Used to ensure data setup time.)
In I^{2}C™ Master mode:
Unused in this mode

bit 3-0 **SSPM<3:0>:** Synchronous Serial Port Mode Select bits
1111 = I^{2}C™ Slave mode, 10-bit address with Start and Stop bit interrupts enabled
1110 = I^{2}C™ Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1101 = Reserved
1100 = Reserved
1011 = I^{2}C™ firmware controlled Master mode (slave idle)
1010 = SPI Master mode, clock = FOSC/(4 * (SSP1ADD+1))[5]
1001 = Reserved
1000 = I^{2}C™ Master mode, clock = FOSC / (4 * (SSP1ADD+1))[4]
0111 = I^{2}C™ Slave mode, 10-bit address
0110 = I^{2}C™ Slave mode, 7-bit address
0101 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control disabled, $\overline{SS}$ can be used as I/O pin
0100 = SPI Slave mode, clock = SCK pin, $\overline{SS}$ pin control enabled
0011 = SPI Master mode, clock = T2_match/2
0010 = SPI Master mode, clock = FOSC/64
0001 = SPI Master mode, clock = FOSC/16
0000 = SPI Master mode, clock = FOSC/4

SSP1CON1 = 0b00110010;

To exchange data between the master and slave simply load the SSP data buffer then block until the transfer is finished. The code below also clears collision flag just in case it was set by a poorly timed write to the buffer. Don't forget assert the nSS line before calling the **SPI_SHIFT_8** function.

```
uint8_t  SPI_SHIFT_8  (uint8_t data)
{
   SSP1CON1bits.WCOL = 0;            // Clear Write Collision flag just in Case
   SSP1BUF = data;                   // Load Buffer with Data to Shift
   while (SSP1STATbits.BF == 0){}    // Block until 8b transferred
   return (SSP1BUF);                 // Return Data/Dummy
}
```

Digital Potentiometer:

Digital potentiometers can be useful in circuits where you need to control an analog function with a microcontroller. For this lab you will be interfacing the PIC16F18324 to a MCP4151-503 (50k) Potentiometer using the SPI Interface. The MCP4151 is available in an 8-pin DIP package which forces an unusual SPI interface due to the low pin count.

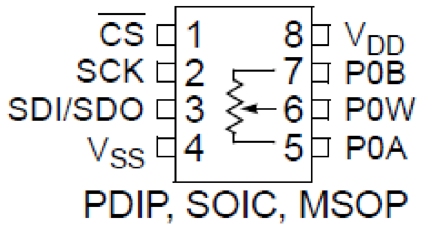


Figure 5 - MCP4151 Pinout

The MCP4151 uses a shared bi-directional SDI/SDO line in order to fit both pins on the small package. R1 must be sized to not limit the SDO voltage below the logic threshold of the SDI of the MCP4151. Since there is not much value in reading the registers of the digital pot we will simplify our design by not using the MISO (SDI) connection.

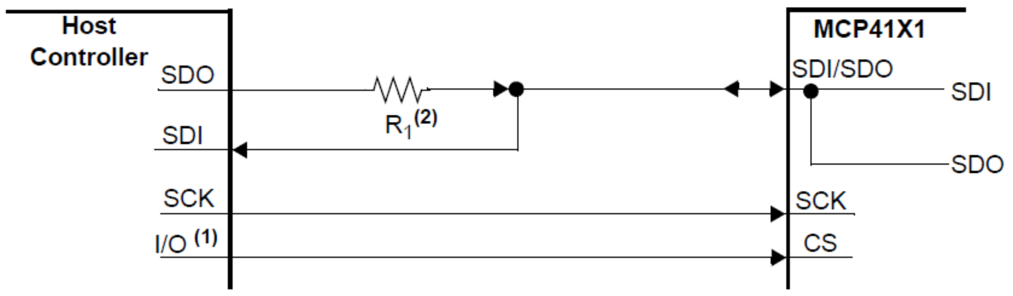


Figure 6 – MCP4151 Bi-Directional Hardware Configuration

If we only care about setting the digital potentiometers wiper position we can simplify the connection as shown in figure 7 below. If we us this simplified method to interconnect the parts we must be careful to only issue write commands from the master.

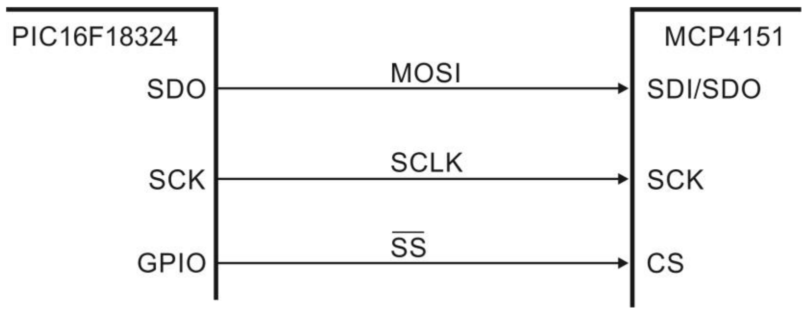


Figure 7 - Simplified Connection Diagram

Data packets can be either 8 or 16 bits in length depending on the function. The general memory map for Microchip digital potentiometers is show in Figure 9 below. For the MCP4151 we will only be writing to address **00h** which will set the wiper position of the potentiometer. The pot has 257 steps so the range of values to be written to the Data payload is 0 to 256.

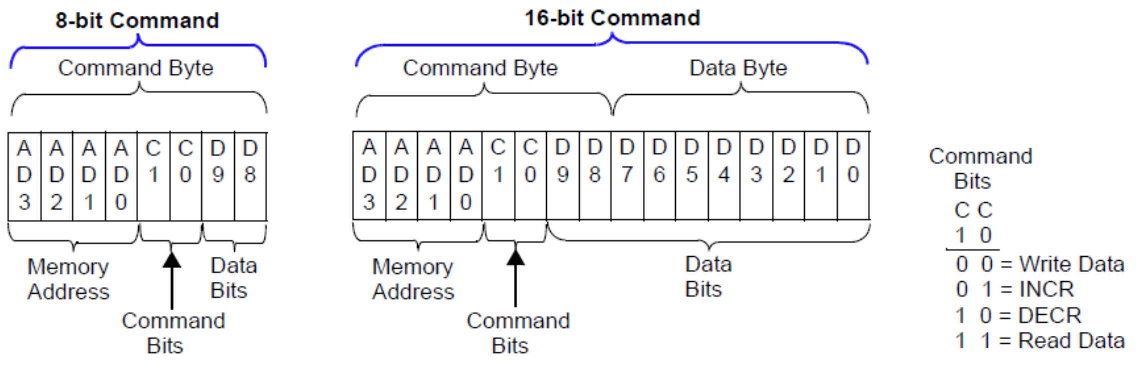


Figure 8 - MCP4151 Commands

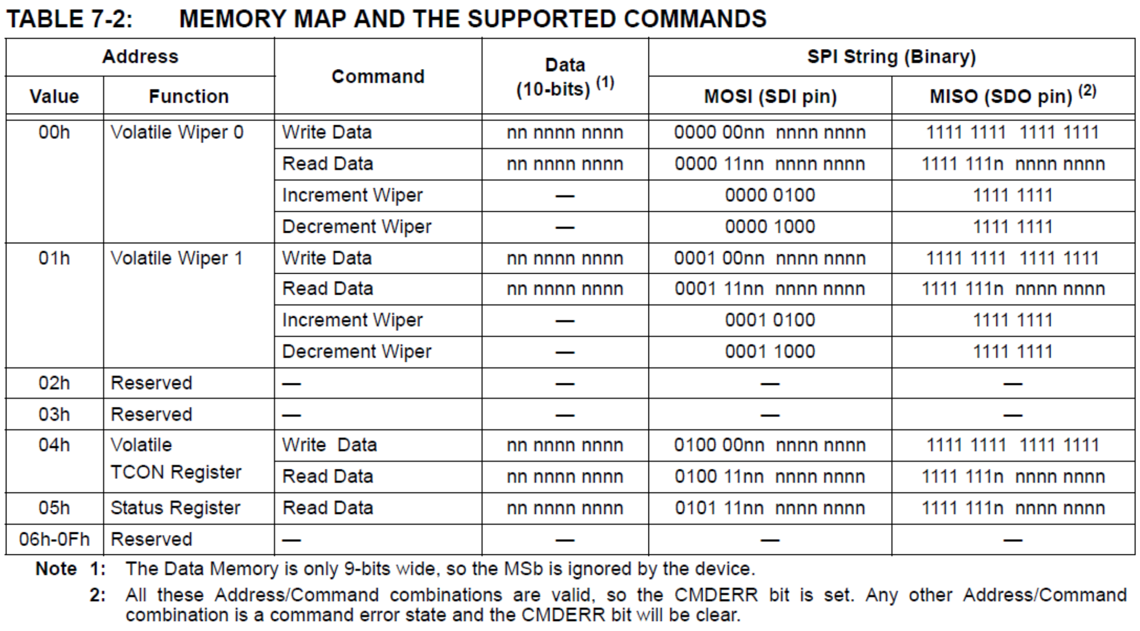


TABLE 7-2: MEMORY MAP AND THE SUPPORTED COMMANDS

Address		Command	Data (10-bits) [1]	SPI String (Binary)	
Value	Function			MOSI (SDI pin)	MISO (SDO pin) [2]
00h	Volatile Wiper 0	Write Data	nn nnnn nnnn	0000 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0000 11nn nnnn nnnn	1111 111n nnnn nnnn
		Increment Wiper	—	0000 0100	1111 1111
		Decrement Wiper	—	0000 1000	1111 1111
01h	Volatile Wiper 1	Write Data	nn nnnn nnnn	0001 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0001 11nn nnnn nnnn	1111 111n nnnn nnnn
		Increment Wiper	—	0001 0100	1111 1111
		Decrement Wiper	—	0001 1000	1111 1111
02h	Reserved	—	—	—	—
03h	Reserved	—	—	—	—
04h	Volatile TCON Register	Write Data	nn nnnn nnnn	0100 00nn nnnn nnnn	1111 1111 1111 1111
		Read Data	nn nnnn nnnn	0100 11nn nnnn nnnn	1111 111n nnnn nnnn
05h	Status Register	Read Data	nn nnnn nnnn	0101 11nn nnnn nnnn	1111 111n nnnn nnnn
06h-0Fh	Reserved	—	—	—	—

Note 1: The Data Memory is only 9-bits wide, so the MSb is ignored by the device.

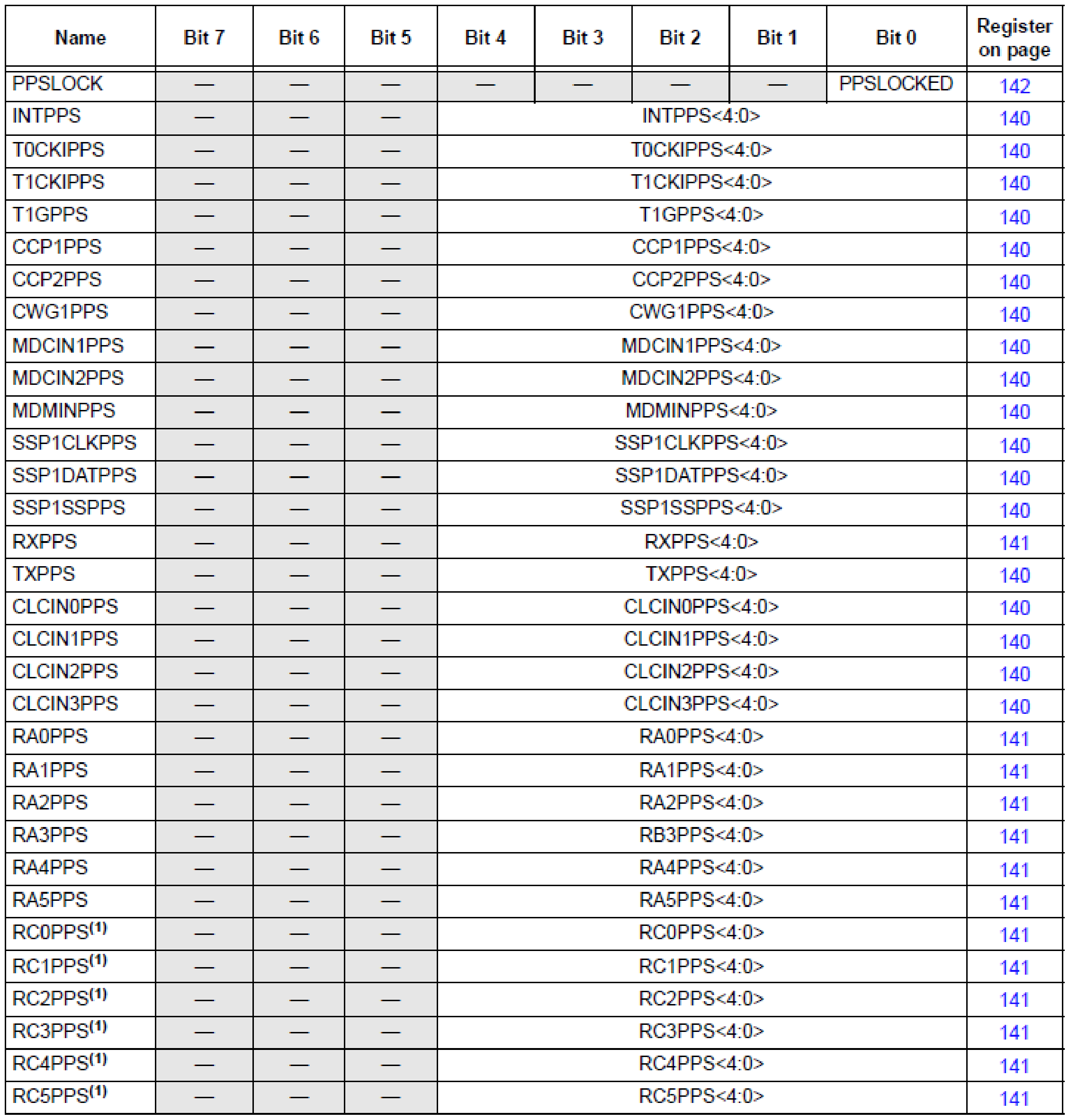
2: All these Address/Command combinations are valid, so the CMDERR bit is set. Any other Address/Command combination is a command error state and the CMDERR bit will be clear.

Figure 9 - MCP4151 Memory Map

Peripheral Pin Select:

For this lab you will need to map the SCK and SDO outputs using the PPS. If you want to try some SPI read commands by adding the resistor in the connection diagram you will also need to map the SDI input to a pin. When mapping peripheral inputs use table 12-1 to determine the register name for the peripheral function then associate the peripheral with the desired pin using the data in 12-8.

TABLE 12-1: SUMMARY OF REGISTERS ASSOCIATED WITH THE PPS MODULE



Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Register on page
PPSLOCK	—	—	—	—	—	—	—	PPSLOCKED	142
INTPPS	—	—	—	INTPPS<4:0>					140
T0CKIPPS	—	—	—	T0CKIPPS<4:0>					140
T1CKIPPS	—	—	—	T1CKIPPS<4:0>					140
T1GPPS	—	—	—	T1GPPS<4:0>					140
CCP1PPS	—	—	—	CCP1PPS<4:0>					140
CCP2PPS	—	—	—	CCP2PPS<4:0>					140
CWG1PPS	—	—	—	CWG1PPS<4:0>					140
MDCIN1PPS	—	—	—	MDCIN1PPS<4:0>					140
MDCIN2PPS	—	—	—	MDCIN2PPS<4:0>					140
MDMINPPS	—	—	—	MDMINPPS<4:0>					140
SSP1CLKPPS	—	—	—	SSP1CLKPPS<4:0>					140
SSP1DATPPS	—	—	—	SSP1DATPPS<4:0>					140
SSP1SSPPS	—	—	—	SSP1SSPPS<4:0>					140
RXPPS	—	—	—	RXPPS<4:0>					141
TXPPS	—	—	—	TXPPS<4:0>					140
CLCIN0PPS	—	—	—	CLCIN0PPS<4:0>					140
CLCIN1PPS	—	—	—	CLCIN1PPS<4:0>					140
CLCIN2PPS	—	—	—	CLCIN2PPS<4:0>					140
CLCIN3PPS	—	—	—	CLCIN3PPS<4:0>					140
RA0PPS	—	—	—	RA0PPS<4:0>					141
RA1PPS	—	—	—	RA1PPS<4:0>					141
RA2PPS	—	—	—	RA2PPS<4:0>					141
RA3PPS	—	—	—	RB3PPS<4:0>					141
RA4PPS	—	—	—	RA4PPS<4:0>					141
RA5PPS	—	—	—	RA5PPS<4:0>					141
RC0PPS(1)	—	—	—	RC0PPS<4:0>					141
RC1PPS(1)	—	—	—	RC1PPS<4:0>					141
RC2PPS(1)	—	—	—	RC2PPS<4:0>					141
RC3PPS(1)	—	—	—	RC3PPS<4:0>					141
RC4PPS(1)	—	—	—	RC4PPS<4:0>					141
RC5PPS(1)	—	—	—	RC5PPS<4:0>					141

12.8 Register Definitions: PPS Input Selection

REGISTER 12-1: xxxPPS: PERIPHERAL xxx INPUT SELECTION

U-0	U-0	U-0	R/W-q/u	U-0	R/W-q/u	R/W-q/u	R/W-q/u
—	—	—	xxxPPS<4:0>				
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	q = value depends on peripheral

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4-0 **xxxPPS<4:0>:** Peripheral xxx Input Selection bits

```
11xxx = Reserved. Do not use.

1011x = Reserved. Do not use.
10101 = Peripheral input is RC5(1)
10100 = Peripheral input is RC4(1)
10011 = Peripheral input is RC3(1)
10010 = Peripheral input is RC2(1)
10001 = Peripheral input is RC1(1)
10000 = Peripheral input is RC0(1)
. . .
01xxx = Reserved. Do not use.
. . .
0011x = Reserved. Do not use.
00101 = Peripheral input is RA5
00100 = Peripheral input is RA4
00011 = Peripheral input is RA3
00010 = Peripheral input is RA2
00001 = Peripheral input is RA1
00000 = Peripheral input is RA0
```

Map the **SDO** and **SCK** to the I/O that you want to use for the SPI bus.

REGISTER 12-2: RxyPPS: PIN Rxy OUTPUT SOURCE SELECTION REGISTER

U-0	U-0	U-0	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u	R/W-0/u
—	—	—	RxyPPS<4:0>				
bit 7							bit 0

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’
u = Bit is unchanged	x = Bit is unknown	-n/n = Value at POR and BOR/Value at all other Resets
‘1’ = Bit is set	‘0’ = Bit is cleared	

bit 7-5 **Unimplemented:** Read as ‘0’

bit 4-0 **RxyPPS<4:0>:** Pin Rxy Output Source Selection bits

```
11111 = Rxy source is DSM
11110 = Rxy source is CLKR
11101 = Rxy source is NCO
11100 = Rxy source is TMR0
11011 = Reserved
11010 = Reserved
11001 = Rxy source is SDO/SDA(1)
11000 = Rxy source is SCK/SCL(1)
10111 = Rxy source is C2OUT(2)
10110 = Rxy source is C1OUT
10101 = Rxy source is DT(1)
10100 = Rxy source is TX/CK(1)
...
01101 = Rxy source is CCP2
01100 = Rxy source is CCP1
01011 = Rxy source is CWG1D(1)
01010 = Rxy source is CWG1C(1)
01001 = Rxy source is CWG1B(1)
01000 = Rxy source is CWG1A(1)
...
00111 = Reserved
00110 = Reserved
00101 = Rxy source is CLC2OUT
00100 = Rxy source is CLC1OUT
00011 = Rxy source is PWM6
00010 = Rxy source is PWM5
00001 = Reserved
00000 = Rxy source is LATxy
```

Note 1: TRIS control is overridden by the peripheral as required.

2: PIC16(L)F18323 only.