ECE 480, Team 2

Microchip MCP41XXX Digital Potentiometer

Implementation in Op-Amp Audio Amplifier Circuit and Interfacing to Microchip PIC18F4520 Microcontroller

Ahmad Alqudaihi

April 4th, 2010

Abstract

The MCP41XXX is a single-channel digital potentiometer. It can be implemented in Op-Amp audio amplifier circuit to change the gain or the input voltage of the amplifiers, which in turn, changes the sound level of the speaker output. Unlike the mechanical potentiometer, the user does not have to change the resistance of the potentiometer by hands or tuning tool. Programming can be used to change the resistance of the pot. This can be done by interfacing the MCP41XXX to a microcontroller.

Keywords: Microchip MCP41XXX, Microchip PIC18F4520, Serial Peripheral Interface Bus.

Table of Contents

1. Introduction:

Potentiometers are used widely in electronic circuit applications such as offset and gain adjustment. The most two common types of potentiometers are the mechanical potentiometer and the digital potentiometer. The digital potentiometer has advantages over the mechanical potentiometer:

i. The mechanical pot has moving parts while the digital one does not have any moving parts. This makes it less sensitive to environmental effects such as changes in temperature, humidity and to shock and vibration.
ii. The digital pot is more robust. It has a longer life than a mechanical pot.
iii. It offers off-hands programmability.

2. Objective:

There are three objectives of this application note:

i. It gives the reader a general idea of how the Microchip MCP41XXX digital potentiometer works.
ii. It gives the reader detailed instructions on how it is implemented in audio amplifier circuits for volume control.
iii. And how to interface it with Microchip PIC18F4520 microcontroller.

3. Microchip MCP41XXX Digital Potentiometer:

The MCP41XXX is a single-channel digital potentiometer. It comes in an 8-pin package as shown in figure 1. There are three different models of this device, MCP41010, MCP41050, and MCP41100. These three models are identical. They only differ in the value of the overall resistance. MCP41010 is a 10 KΩ pot; MCP41050 is 50 KΩ pot; and MCP41100 is a 100 KΩ pot (This can be noticed in the last three digits of the part number). For the device to function, it needs to communicate with the microcontroller via Serial Peripheral Interface Bus (SPI).

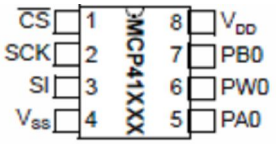

Figure 1: MCP41XXX

3.1 MCP41XXX Pin Descriptions:

Pin #	Name	Function/Description
1	$\overline{CS}$	This is the SPI port chip select pin. It is used to execute a new command after it is loaded into the device.
2	SCK	This is the SPI port clock. It used to clock in data into the device. The maximum clock frequency that the device can have is 10 MHz.
3	SI	This is the SPI port input pin. It is used to load data into the device.
4	V_{SS}	Ground.
5	PA0	The pot terminal A connection.
6	PW0	The pot wiper connection.
7	PB0	The pot terminal B connection.
8	V_{DD}	Power supply. It must fall between 2.7 V and 5.5 V.

Table 1: MCP41XXX Pin Descriptions

4. Implementing MCP41XXX in the Op-Amp Audio Amplifier Circuit:

The purpose of implementing pots in audio amplifier circuits is to adjust the gain or the input voltage of the amplifiers, which in turn, changes the sound level of the speaker output. One audio amplifier circuit that the MCP41XXX can be used in is the Op-Amp audio amplifier shown in figure 2. In this circuit, the user changes the input voltage of the non-inverting amplifier through the pot. As a result, the voltage across the speaker changes according to the following equation:

$$V_{speaker} = V_{in}\left(1+\frac{47K\Omega}{4.7K\Omega}\right) = 11\cdot V_{in}$$

Where:

$V_{speaker}$ is the voltage across the speaker

V_{in} is the variable input voltage of the non-inverting amplifier

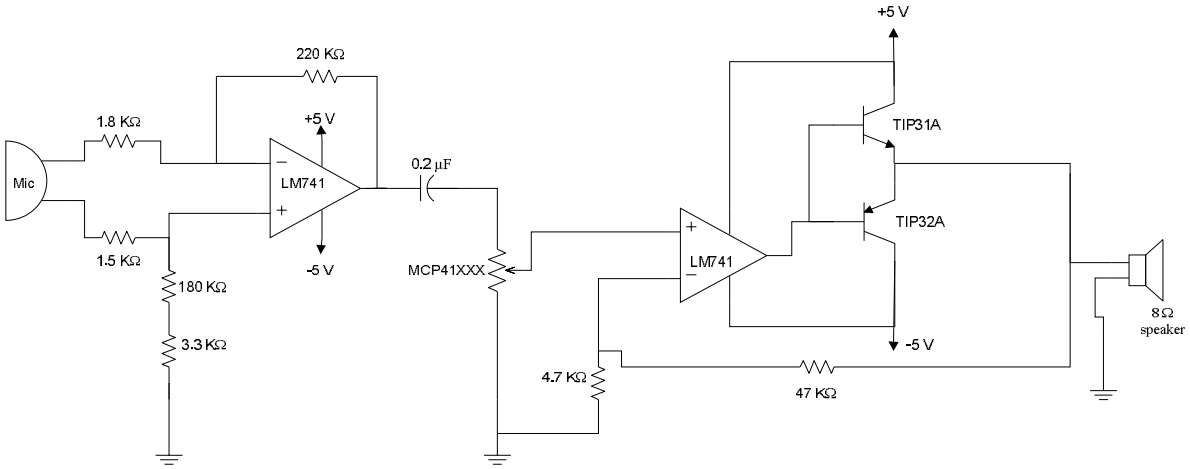

Figure 2: Op-Amp Audio Amplifier

4.1 Connections:

Pin#4 of the pot is connected to ground. Pin#8 is connected to the power supply (must be between 2.7 V and 5.5 V). The pot is a non linear load at its power pin (pin#8). This along with some inductance from the wires can create an oscillator. To squelch oscillations, a bypass capacitor (typical value of 0.1 µF) should be added very close to pin#8 as shown in figure 3.

Pin#5 of the pot is connected to ground. Pin#6 is connected to pin#3 of the op-amp. Finally, Pin#7 of the first pot is connected to the 0.2 µF capacitor.

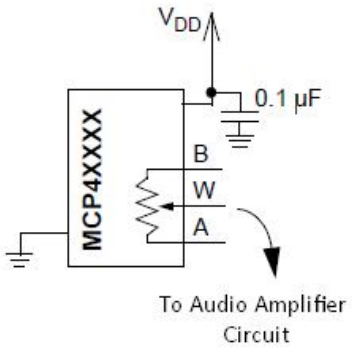

Figure 3: MCP41XXX with by-pass capacitor

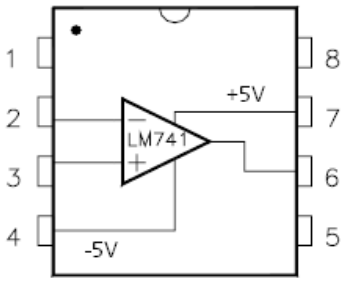

Figure 4: LM 471 Op Amp

4.2 Resistance of MCP41XXX:

The input voltage of the non inverting amplifier can be adjusted by changing the resistance between terminal A and the wiper and between terminal B and the wiper of the pot. This can be done by setting the position of the wiper to any of the 256 possible positions.

The user can calculate the resistance between both terminal and the wiper if the position of the wiper is known or vice versa, using the following:

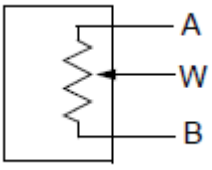

Figure 5: Model of the MCP41XXX

$$R_{WA}(D_n) = \frac{(R_{AB}) \cdot (256 - D_n)}{256} + R_W$$

$$R_{WB}(D_n) = \frac{(R_{AB})(D_n)}{256} + R_W$$

Where:

A is the A terminal of the pot (PA0 pin)
W is the wiper Terminal (PW0 pin)
B is the B terminal (PB0 pin)
R_{WA} is the resistance between terminal A and the wiper
R_{WB} is the resistance between terminal B and the wiper
R_{AB} is the overall resistance for the pot (10 KΩ for MCP41010, 50 KΩ for MCP41050, or 100 KΩ for MCP41100).
R_W is the resistance of the wiper (typically, 52 Ω for MCP41010, 125 Ω for MCP41050 and MCP41100).
D_n is the wiper setting, a value between 0 and 255

When the device is connected to the amplifier, R_W can be neglected, since the input impedance of the amplifier is very high. Therefore, the above equations can be reduced to:

$$R_{WA}(D_n) = \frac{(R_{AB}) \cdot (256 - D_n)}{256}$$

$$R_{WB}(D_n) = \frac{(R_{AB})(D_n)}{256}$$

5. Interfacing to Microchip PIC18F4520 Microcontroller:

When the pot is powered up, the wiper is reset to mid-scale position (D_n=128). If the user desires to change the position of the wiper, a microcontroller must be used to communicate with the pot via SPI interface.

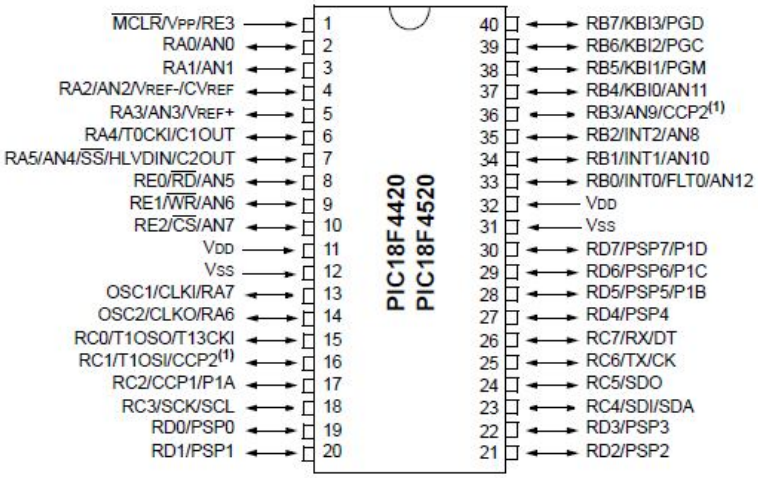

Figure 6: PIC18F4520 microcontroller

5.1 PIC18F4520 Pins Needed:

Pin #	Name	Description
17	RC2	This pin will be connected to pin#1 of the pot, the chip select. It is used to control commands executions on the pot.
18	SCK	This is the SPI clock output. It will be connected to pin# 2 of the pot. It is used to clock data out of the PIC and into the pot. The frequency of the clock output is the same as the frequency of the external clock oscillator (i.e. 10 MHz).
24	SDO	This is the SPI data output. It will be connected to pin# 3 of the pot. It is used to output the data from the PIC.

Table 2: PIC18F4520 pins needed for communications with MCP41XXX

5.2 Enabling the SPI mode on PIC18F4520:

Before the user starts writing the code for the communication with pot, he/she must enable the SPI mode on the PIC. This can be done by including the `spi.h` library at the beginning of the code. In this library, the SSPCON1 control register on the PIC is pre-defined and bit 5 of this register is pre-set to 1. This enables serial port and configures SCK, SDO as serial pins. This library also includes all the definitions for the functions and the variables that can be used to communicate with the pot via SPI interface.

5.3 Initializing the Serial Synchronous Serial Port (SSP) Module:

The following pre-defined function is used to initialize SSP module for SPI communications:

```
void OpenSPI(unsigned char sync_mode,
             unsigned char bus_mode,
             unsigned char smp_phase );
```

where `sync_mode` is one of the following defined values:

`SPI_FOSC_4`	SPI Master mode, clock = FOSC/4 (or Tcy)
`SPI_FOSC_16`	SPI Master mode, clock = FOSC/16 (or 4* Tcy)
`SPI_FOSC_64`	SPI Master mode, clock = FOSC/64 (or 16*Tcy)
`SPI_FOSC_TMR2`	SPI Master mode, clock = TMR2 output/2
`SLV_SSON`	SPI Slave mode, /SS pin control enabled
`SLV_SSOFF`	SPI Slave mode, /SS pin control disabled

`bus_mode` is one of the following defined values:

`MODE_00`	Setting for SPI bus Mode 0,0
`MODE_01`	Setting for SPI bus Mode 0,1
`MODE_10`	Setting for SPI bus Mode 1,0
`MODE_11`	Setting for SPI bus Mode 1,1

And `smp_phase` is one of the following defined values:

`SMPEND`	Input data sample at end of data out
`SMPMID`	Input data sample at middle of data out

5.3.1 `sync_mode`:

In this application, the PIC plays the master role since it controls the SCK that clocks data into the slave, the pot. Moreover, The input clock in the pot must be synchronous with the output clock from the PIC such that there are 16 clock cycles in the pot before the user execute the command (i.e. set $\overline{CS}$ =1). Therefore, `SPI_FOSC_64` is the value that the user should choose for `sync_mode`.

5.3.2 `bus_mode`:

The first digit of the mode is the clock polarity select (CKP) bit; 0 means that the idle state for clock is a high level and 1 is a low level. The second digit is SPI clock select (CKE) bit; 1 means that Transmission occurs on transition from active to idle clock state. In other words, the SDO data is valid before there is a clock edge on SCK. When it is set to 0, the opposite occurs. This is illustrated in the figure 7.

When, communicating with the pot, the data must be clocked in on the rising edge of the clock. Therefore, the `bus_mode` must be set to 0,0.

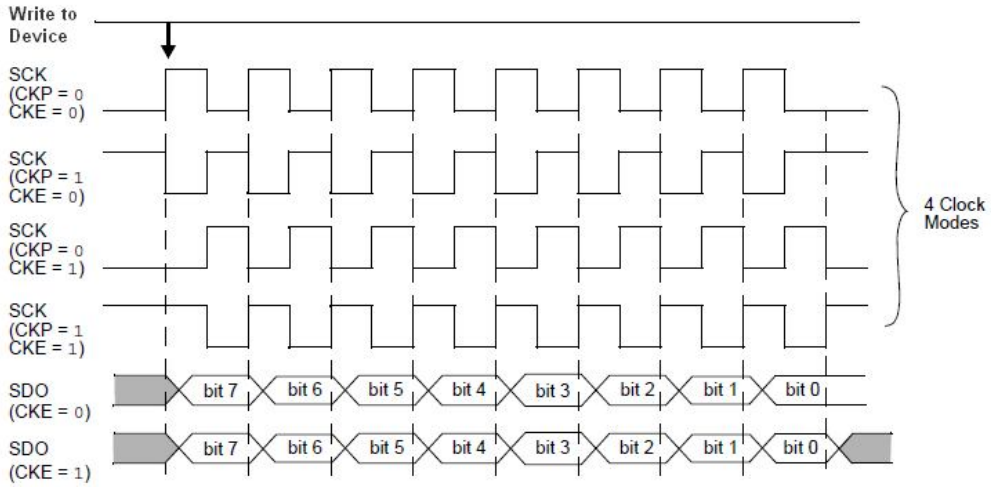

Figure 7: SCK waveforms

5.3.3 `smp_phase`:

The setting of this value is only important when the PIC is receiving data. In this application, the PIC is only transmitting data. Therefore, any of the two available values would not affect the operation of the pot.

5.4 Data Directionality:

The next step after configuring the pins in table 2 as serial ports pins is to set them as output pins. These pins are part of PORTC port on the PIC and their data direction can be set by

changing the bits of the TRISC data direction register on the PIC. Clearing bits 2, 3, and 5 will configure RC2, SDK and SDO respectively as outputs.

5.5 Writing Data to MCP41XXX:

To execute any command, a 16-bit Data must be sent from SDO of the PIC to SI of the pot. The 8 most significant bits of the data are called the command byte. It contains two command bits (4,5) and two potentiometer select bits (0,1) as shown in figure 8. If the two command bits are identical (0,0 or 1,1), No command will be executed. If bit 5 and 4 are set to 0 and 1 respectively, a new wiper setting will be written to the pot. Lastly, setting bit 5 and 4 to 1 and 0 respectively will set the pot on Shutdown mode.

The potentiometer bits, 0 and 1, determine which channel (If there are two) will the command be executed on; bit 0 is channel 1 and bit 1 is channel 2. Since MCP41xxx is a single channel pot, bit 0 of the command byte should be set to 1, and bit 4 and 5 to 1 and 0 respectively in order to execute the command. The rest of the bits are "don't care" bits.

The 8 least significant bits of the transmitted data are called the data byte. It is used to change the wiper setting when a binary (or hex) code corresponding to the decimal value of D_n is sent to the pot. For example, If D_n =255, then the data byte= FF (in hex) or 11111111 (in binary).

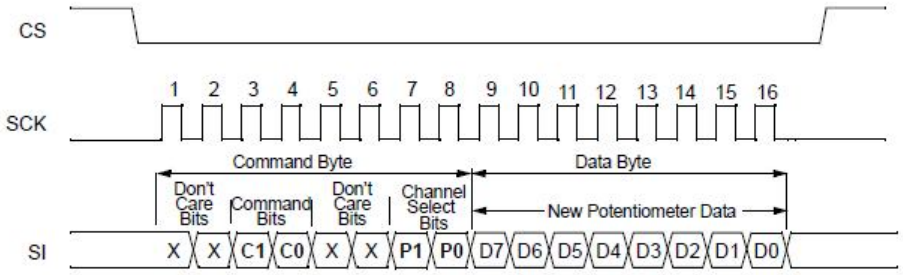

Figure 8. Instruction sequence for MCP41XXX

The following pre-defined function is used to write data into the pot:

```
unsigned char WriteSPI(  unsigned char data_out );
```

Where `data_out` is the data to be written into the pot.

One problem with this function is that `data_out` can only be an 8-bit value, and as stated above, a 16-bit data must be written to the pot. This problem can be easily solved by writing the function twice. Data are transferred from the PIC starting with the most significant bit and ending with the least significant bit. Therefore, In the first `WriteSPI` function, `data_out` = command byte, and in the second one, `data_out` =data byte.

5.5.1 Chip select:

As shown in figure 8, the $\overline{CS}$ must be set low (by sending a 0 from the RC2 pin) before the data is clocked into the pot. To execute the command, the $\overline{CS}$ must be set high (by sending a 1 from RC2). It is important to note that the number of clock rising edges must be 16

while $\overline{CS}$ is low, otherwise the command will be aborted. This is taken care of when `sync_mode` is set to `SPI_FOSC_64`.

5.6 MPLAB C18 C Compiler Code Lines Needed:

```
#include <p18cxxx.h>  // PIC18

#include <spi.h>  // spi library

#define SPI_CS LATCbits.LATC2  // Setting RC2 (pin#17)of the PIC to be
                               // SPI chip select select

void main()
{

TRISC=0x00;  // setting PORTC port pins to be outputs

SPI_CS = 1;  // chip select=1

OpenSPI(SPI_FOSC_64,MODE_00,SMPEND);  // Initializing SSP

SPI_CS = 0;      // chip select=0

WriteSPI(0x11);   // writing the command byte to the pot


WriteSPI(0xFF);  //  writing the data byte to the pot.
                   //In this case, Dn=FFh=255d

SPI_CS = 1;  // chip select=1, to excute the command

CloseSPI();   // disable SSP
while(1);
{
}
}
```

6. Conclusion:

Detailed explanation for implementing MCP41XXX in the Op-Amp circuit and developing the C code for PIC were given to allow the user to easily use the device with any circuit application. The Assembly language can also be used for coding the PIC to communicate with the device, if desired.

7. Refrences:

Microchip Technology Inc. “Comparing Digital Potentiometers to Mechanical Potentiometers” application note. 2000

Microchip Technology Inc. “Interfacing Microchip’s MCP41XXX and MCP42XXX Digital Potentiometers to a PICmicro Microcontroller” application note. 2001

Microchip Technology Inc. “MCP41XXX/42XXX” data sheet. 2003

Microchip Technology Inc. “MPLAB C18 C Compiler Libraries”. 2005

Microchip Technology Inc. “PIC18F2420/2520/4420/4520” data sheet. 2004

Wierzba, Gregory M. “ECE 203: Electric Circuits and Systems Laboratory”. 2007