# SYSTEM REPORT

## *SYSTEM DIAGRAM*

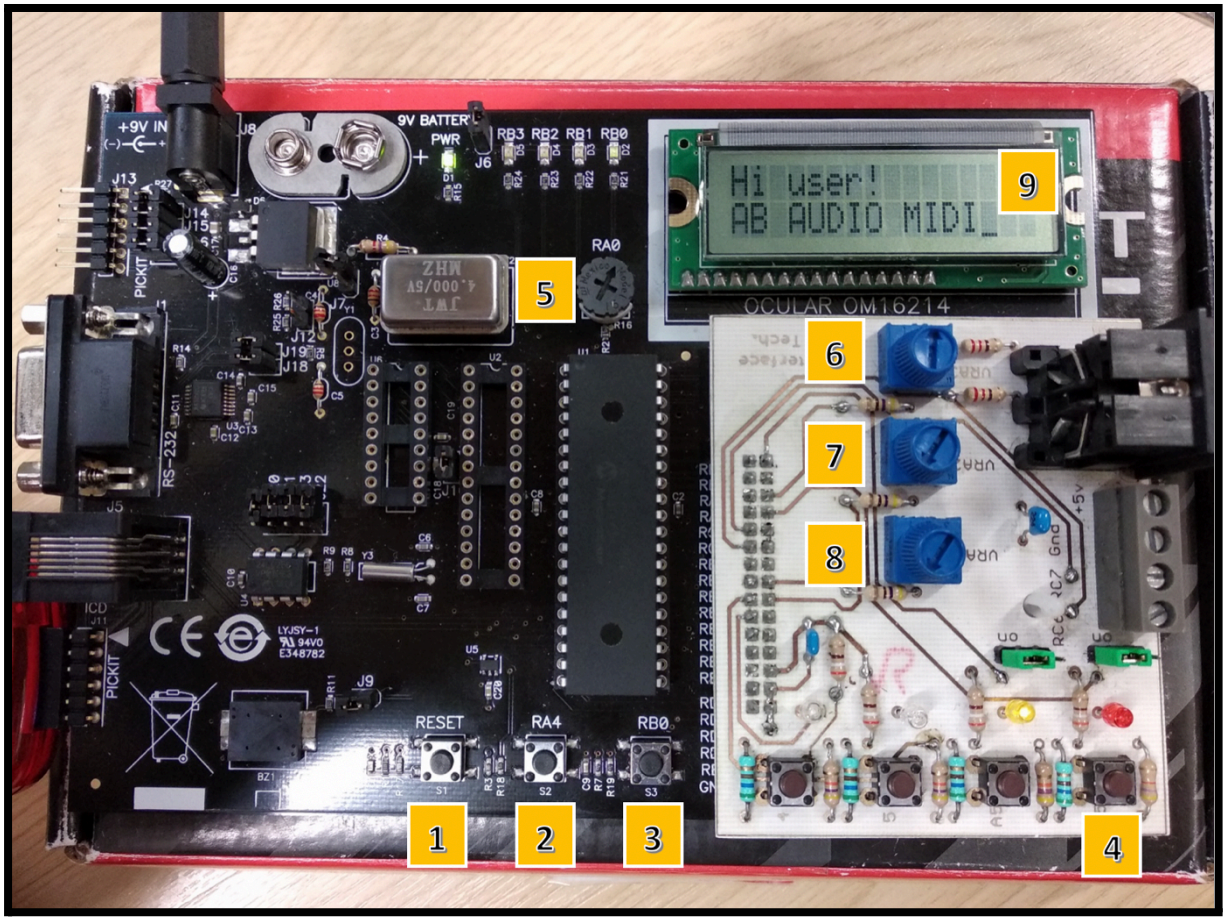



*Figure 1: System*

Figure 1 displays the system. Labelled are the interactive elements offered by it:

- 1 - **RESET**: pressing this button will reboot the system
- 2 - **RA4**: pressing this button allows the user to cycle through editable parameters
- 3 - **RB0**: pressing this button allows the user to cycle through MIDI Channels
- 4 - **RC5**: pressing this button will send a Note on MIDI message with the parameters set by the user, followed by a Note off MIDI message.
- 5 – **RA0**: this potentiometer allows the user to set the Note number, from 0 to 127.
- 6 – **VRA3**: this potentiometer allows the user to vary Pan.
- 7 – **VRA2**: this potentiometer allows the user to vary Modulation.
- 8 – **VRA1**: this potentiometer allows the user to set the Velocity value, from 1-127.
- 9 – **LCD** screen: this screen provides instant feedback to the user.

# *SYSTEM FLOWCHART*

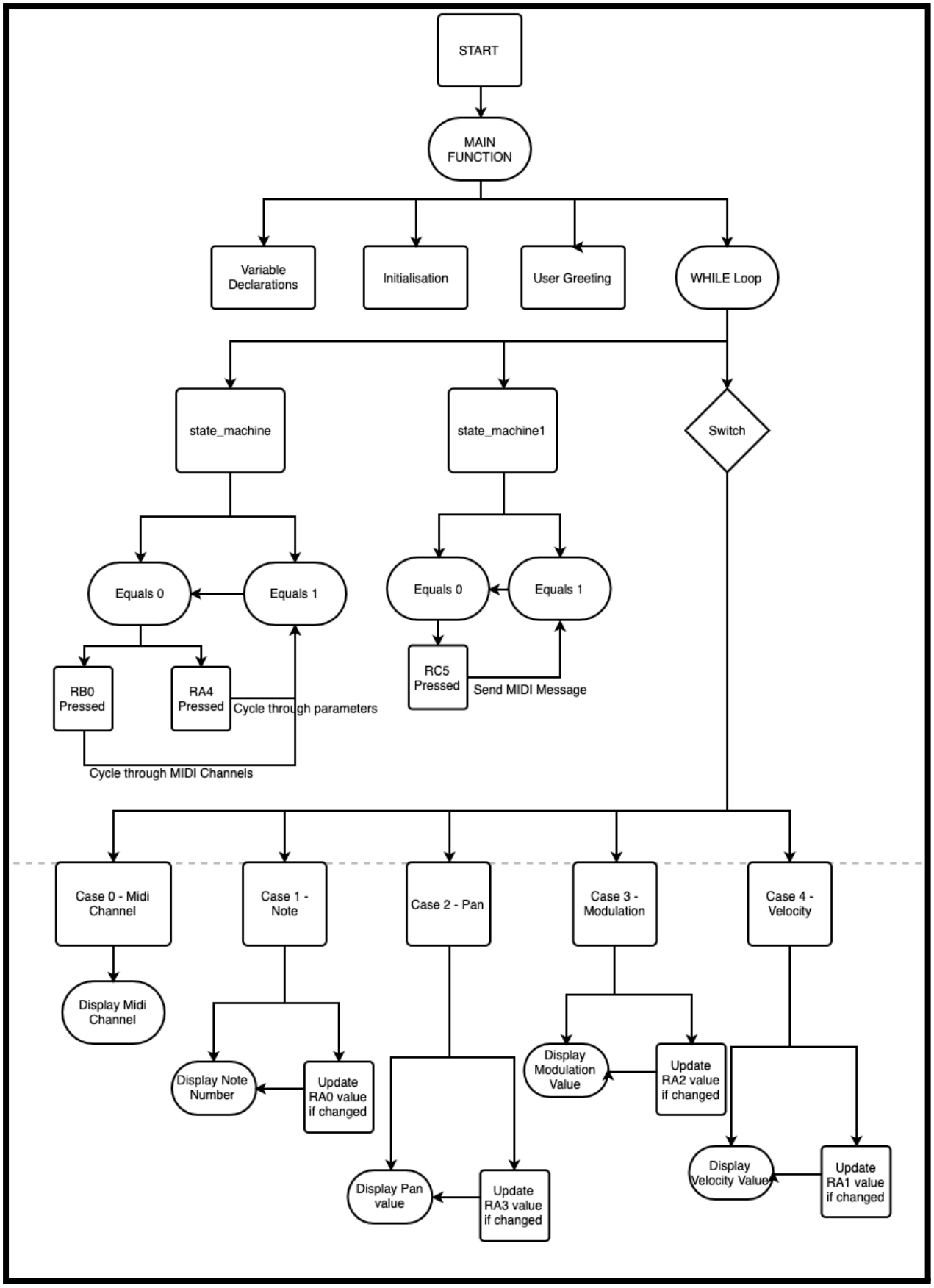



*Figure 2: System Flowchart*

# *USER GUIDE*

When started, the LCD screen will light up and greet the user. After a few seconds, the screen will change to display the selected MIDI Channel, defaulting to channel 1. The user can select from channel 1 to channel 10 by pressing RB0. After channel 10 the system will cycle back to channel 1 if RB0 is pressed.

The user can cycle between 5 different parameters – MIDI Channel, Note, Pan, Modulation and Velocity – by pressing RA4.

The 4 potentiometers offered by the system allow the user to change the parameters:

- RA0 changes note value (0:127)
- VRA3 changes Pan (-64:64)
- VRA2 changes Modulation (0:127)
- VRA1 changes Velocity (1:127)

Each of these parameters can be changed even when not being displayed, ie. If the Pan potentiometer is changed whilst Velocity is being displayed on the screen, the Pan value will still change. However, in order to confirm the change, the user must display the Pan value on the screen.

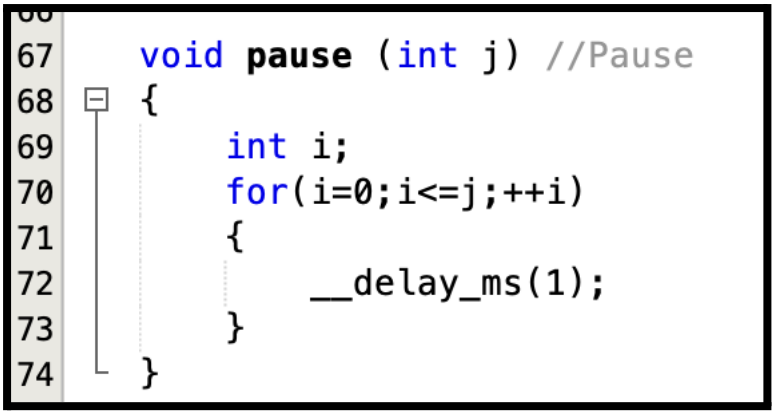
If RC5 is pressed, the system will send a MIDI note on message with the set values, followed shortly by a note off message.

# *TECHNICAL DESCRIPTION OF CODE*

```
61  #include <xc.h> // Include xc
62  #include <stdio.h> // Include stdio
63  #include "lcd.h"  // Include lcd
64  #include "usart.h" // Include usart
```

*Figure 3*

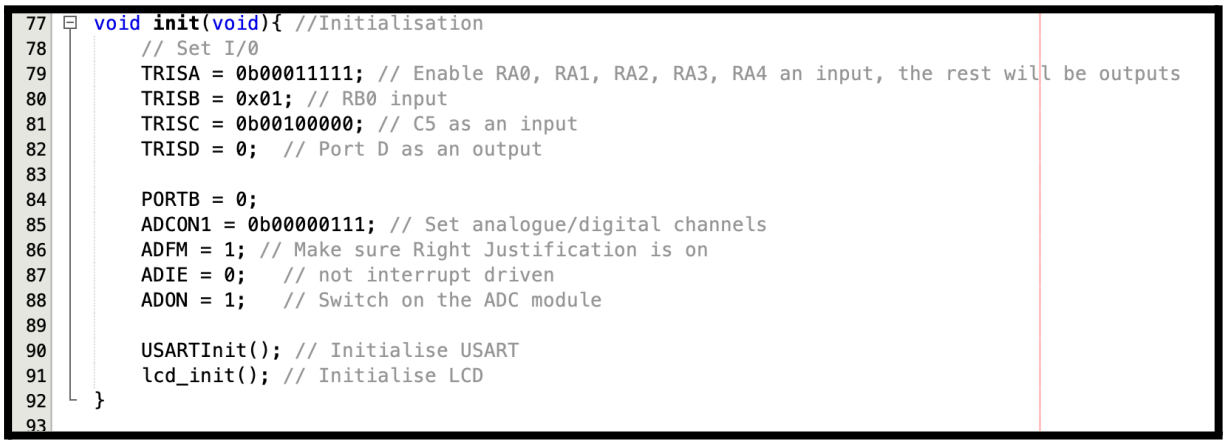
Figure 3 shows the #include statements at the start of the code, which are necessary in order for the whole program to work correctly.



```
67    void pause (int j) //Pause
68    {
69        int i;
70        for(i=0;i<=j;++i)
71        {
72            __delay_ms(1);
73        }
74    }
```

*Figure 4*

Figure 4 shows the declaration of the pause function, which will be used later to hold the user greeting for a few seconds.



```
77  void init(void){ //Initialisation
78      // Set I/0
79      TRISA = 0b00011111; // Enable RA0, RA1, RA2, RA3, RA4 an input, the rest will be outputs
80      TRISB = 0x01; // RB0 input
81      TRISC = 0b00100000; // C5 as an input
82      TRISD = 0;  // Port D as an output
83
84      PORTB = 0;
85      ADCON1 = 0b00000111; // Set analogue/digital channels
86      ADFM = 1; // Make sure Right Justification is on
87      ADIE = 0;   // not interrupt driven
88      ADON = 1;   // Switch on the ADC module
89
90      USARTInit(); // Initialise USART
91      lcd_init(); // Initialise LCD
92  }
93
```

*Figure 5*

Figure 5 shows the init function (initialisation). Inputs and Outputs are set on Ports A, B, C and D. Furthermore, the ADC parameters are set to work accordingly to our needs. The potentiometers are set as analogue channels whereas everything else is set as digital. Finally the USARTInit and lcd_init functions are called, initialising the USART and the lcd screen.

```
95  void main(void){
96      unsigned char midiChannel = 0; // MIDI Channel variable
97      unsigned char note = 0; // Selected note variable
98      unsigned char pan = 0; // Pan variable
99      unsigned char mod = 0; // Modulation variable
100     unsigned char velocity = 1; // Velocity variable
101     unsigned char outString[15] = ""; // OutString for writing into lcd
102
103     unsigned char adc_left, adc_right; // ADC values
104
105     int adc_new, last_value; // Manipulated ADC values
106
107     int state_machine = 0; // Variables that will be used for menu system
108     int state_machine1 = 0; // Variables that will be used for menu system
109     int updatePan = 0; // Used to update Pan
110     int updateMod = 0; // Used to update Mod
111     int parameter = 0; // Variables that will be used for menu system
112
113     init();
```

*Figure 6*

Figure 6 shows the start of the main function, where all the variables are declared. midiChannel, note, pan, mod and velocity will be used to hold the values of each parameter (MIDI Channel, Note, Pan, Modulation and Velocity). outString is used for writing into the LCD screen. adc_left and adc_right will be used to hold values read from the ADC. adc_new and last_value will be used for manipulated ADC values. The integer variables will be used on if statements for menu systems and so the system knows if to update values or not.
Finally, the init function is called, initialising the system.

```
115     lcd_goto(0);    // select first line
116     lcd_puts("Hi user!"); // Greet user
117     lcd_goto(0x40); // select second line
118     lcd_puts("AB AUDIO MIDI"); // Device name
119
120     pause(3000); // Pause for a bit
121     lcd_clear(); // Clear screen
122
```

*Figure 7*

Figure 7 shows the code responsible for greeting the user. After 3000 miliseconds, the screen is cleared.

```
124         while(1){ // While system is running
125 
126             if((PORTBbits.RB0==0)&&(state_machine==0)){ // RB0 button press, state_machine = 0
127                 midiChannel++; // Increase MIDI Channel by 1
128                 state_machine = 1; // Change state machine
129             }
130             if((PORTAbits.RA4==0)&&(state_machine==0)){ // RA4 button press, state_machine = 0
131                 lcd_clear(); // Clear screen
132                 parameter++; // increase parameter
133                 state_machine = 1; // Change state machine
134             }
135             if((PORTAbits.RA4==1)&&(PORTBbits.RB0==1)&&(state_machine==1)){ // No button press, state_machine = 1
136                 state_machine = 0; // Change state machine back to its original value
137             }
138 
139             if(parameter > 4) // If parameter = 5 (5th menu element)
140             {
141                 parameter = 0; // Go back to first element (cycle completed)
142             }
143             if(midiChannel > 9) // If Midi Channel > 10
144             {
145                 lcd_clear(); // Clear screen
146                 lcd_goto(0);    // select first line
147                 lcd_puts("Midi Channel:"); // Write Midi Channel
148                 midiChannel = 0; // Cycle through back to first channel
149             }
150 
```

*Figure 8*

Figure 8 shows the start of the while loop.
When state_machine equals 0, if RB0 is pressed, the value of midiChannel will increase by 1. state_machine will then equal 1, making the if irrelevant so that one RB0 button press increases midiChannel by only 1, no matter how long it is held for. Once the user lets go of RB0 state_machine will change back to 0, allowing for another button press that will increase midiChannel.
The same system is used with RA4, which will vary parameter, a variable that will be used later in the menu system on the switch.
The two if statements at the end are used to limit the number of MIDI channels to 10, and the parameters to 5 (as there are 5 possible parameters).

```
152             //Menu System
153             switch (parameter) // Menu System
154             {
155                 case 0: // MIDI Channel
156                 lcd_goto(0);    // select first line
157                 lcd_puts("Midi Channel:"); // Write Midi Channel
158 
159                 int channelNumber = 1+midiChannel; // As 0 = Channel 1, we need to equate the values
160                 sprintf(outString,"%d",channelNumber); // Prepare to write the channel
161                 lcd_goto(0x40); // select second line
162                 lcd_puts(outString); // Write the channel
163 
164                 break;
```

*Figure 9*

Figure 9 shows the start of the switch code and the first and default case (MIDI Channel). As the screen was last cleared, text can be written into the LCD using lcd_puts. channelNumber is used so that the channel number displayed is the same as the channel selected, as a value of 0 is channel 1. The channel number is then written onto the second line of the LCD screen.

```
166                 case 1: // Note
167                 lcd_goto(0);    // select first line
168                 lcd_puts("Note:"); // Write Note
169 
170                 sprintf(outString,"%d",note); // Prepare to write the note
171                 lcd_goto(0x40); // select second line
172                 lcd_puts(outString); // Write the note
173 
174                 //ADC bit
175                 ADCON0=0b00000011; // Read from RA0
176                 NOP(); // Wait 4 uS NOP() as detailed in the data sheet
177                 NOP(); //
178                 NOP(); // We need to wait a minimum of 4uS between switching channels
179                 NOP();
180                 GODONE=1; // Start a conversion on channel 0
181                 while(GODONE)continue;
182                 ADIF=0; // Clear the end of conversion flag
183                 adc_left=ADRESL; // Read the lower bits of the ADC results
184                 adc_right = ADRESH; // Read the upper bits of the ADC results
185 
186                 adc_new = adc_right<<8;
187                 adc_new = adc_left + adc_new;
188                 adc_new = adc_new>>3;
189                 note = adc_new; // set note as ADC value
190 
191                 if(adc_new!=last_value){
192                     lcd_clear();    // Clear screen if the value is different
193                 }
194                 last_value=adc_new;
195                 break;
```

*Figure 10*

Figure 10 shows the code for case 1 (Note). The first set of lines are basically the same as those in case 0, handling the writing onto the LCD screen. Starting on line 175 is code used for ADC handling. The comments describe what each line does. Starting on line 186 is code that manipulates the ADC values to get a value that can be usable by MIDI. The if statement on line 191 will clear the lcd screen if the note value has been changed.

```
197                 case 2: // Pan
198                 lcd_goto(0);    // select first line
199                 lcd_puts("Pan:"); // Write pan
200 
201                 int panValue = pan - 64; // Change how pan value is displayed
202                 sprintf(outString,"%d",panValue); // Prepare to write pan
203                 lcd_goto(0x40); // select second line
204                 lcd_puts(outString); // Write the pan value
205 
206                 //ADC bit
207                 ADCON0=0b00001111; // Read from VRA3
208                 NOP(); // Wait 4 uS NOP() as detailed in the data sheet
209                 NOP(); //
210                 NOP(); // We need to wait a minimum of 4uS between switching channels
211                 NOP();
212                 GODONE=1; // Start a conversion on channel 0
213                 while(GODONE)continue;
214                 ADIF=0; // Clear the end of conversion flag
215                 adc_left=ADRESL; // Read the lower bits of the ADC results
216                 adc_right = ADRESH; // Read the upper bits of the ADC results
217 
218                 adc_new = adc_right<<8;
219                 adc_new = adc_left + adc_new;
220                 adc_new = adc_new>>3;
221                 pan = adc_new; // set pan as ADC value
222 
223                 if((updatePan == 0)&&(pan==last_value)){
224                     USARTWriteByte(176 + midiChannel); // Control Value on Channel 1
225                     USARTWriteByte(0x0A); // Control Pan
226                     USARTWriteByte(pan); // pan value
227                     updatePan = 1;
228                 }
229                 if(adc_new!=last_value){
230                     lcd_clear();    // Clear screen if the value is different
231                     updatePan = 0;
232                 }
233                 last_value=adc_new;
234                 break;
235 
```

*Figure 11*

Figure 11 shows the code for case 2 (Pan). It's basically the same as that for case 1, except for the if statement on line 223. This if statement will send MIDI messages updating the pan value if it has been changed.

```
236                 case 3: // Modulation
237                 lcd_goto(0);    // select first line
238                 lcd_puts("Mod Value:"); // Write Mod Value
239 
240                 sprintf(outString,"%d",mod); // Prepare to write mod value
241                 lcd_goto(0x40); // select second line
242                 lcd_puts(outString); // Write the mod value
243 
244                 //ADC bit
245                 ADCON0=0b00001011; // Read from VRA2
246                 NOP(); // Wait 4 uS NOP() as detailed in the data sheet
247                 NOP(); //
248                 NOP(); // We need to wait a minimum of 4uS between switching channels
249                 NOP();
250                 GODONE=1; // Start a conversion on channel 0
251                 while(GODONE)continue;
252                 ADIF=0; // Clear the end of conversion flag
253                 adc_left=ADRESL; // Read the lower bits of the ADC results
254                 adc_right = ADRESH; // Read the upper bits of the ADC results
255 
256                 adc_new = adc_right<<8;
257                 adc_new = adc_left + adc_new;
258                 adc_new = adc_new>>3;
259                 mod = adc_new; // set mod as ADC value
260 
261                 if((updateMod == 0)&&(mod==last_value)){
262                     USARTWriteByte(176 + midiChannel); // Control Value on Channel 1
263                     USARTWriteByte(0x01); // Control Mod
264                     USARTWriteByte(pan); // mod value
265                     updateMod = 1;
266                 }
267 
268                 if(adc_new!=last_value){
269                     lcd_clear();    // Clear screen if the value is different
270                     updateMod = 0;
271                 }
272                 last_value=adc_new;
273                 break;
274 
```

*Figure 12*

Figure 12 shows the code for case 3 (modulation). It's basically the same as that for case 2 but using different variables.

```
275                 case 4: // Velocity
276                 lcd_goto(0);    // select first line
277                 lcd_puts("Velocity:"); // Write velocity
278 
279                 sprintf(outString,"%d",velocity); // Prepare to write velocity value
280                 lcd_goto(0x40); // select second line
281                 lcd_puts(outString); // Write the velocity value
282 
283                 //ADC bit
284                 ADCON0=0b00000111; // Read from VRA1
285                 NOP(); // Wait 4 uS NOP() as detailed in the data sheet
286                 NOP(); //
287                 NOP(); // We need to wait a minimum of 4uS between switching channels
288                 NOP();
289                 GODONE=1; // Start a conversion on channel 0
290                 while(GODONE)continue;
291                 ADIF=0; // Clear the end of conversion flag
292                 adc_left=ADRESL; // Read the lower bits of the ADC results
293                 adc_right = ADRESH; // Read the upper bits of the ADC results
294 
295                 adc_new = adc_right<<8;
296                 adc_new = adc_left + adc_new;
297                 adc_new = adc_new>>3;
298                 velocity = adc_new + 1; // Set velocity as ADC value + 1 in order to not get velocity = 0
299 
300                 if(velocity>127){ // If velocity = 128
301                     velocity=127; // set velocity as 127
302                 }
303                 if(adc_new!=last_value){
304                     lcd_clear();    // Clear screen if the value is different
305                 }
306                 last_value=adc_new;
307                 break;
308 
309             }
310
```

*Figure 13*

Figure 13 shows the code for case 4 (Velocity). It's basically the same as that of case 1 except for the if statement on line 300. On line 298 the minimum velocity is set to 1 as if the user sets it to 0 using the potentiometer it will be 1 as 1 has been added to it. The if statement limits the maximum to 127, as due to the previous addition it could now be 128.

```
311             //Send MIDI Message
312             if((PORTCbits.RC5==0)&&(state_machine1==0)){ // RC5 putton press, state_machine1 = 0
313             state_machine1=1; // Set state_machine1 as 1
314             }
315 
316             if ((PORTCbits.RC5==1)&&(state_machine1==1)){ // No RC5 press, state_machine1 = 1
317                 state_machine1=0; // set state_machine1 back to its original value
318 
319                 USARTWriteByte(144 + midiChannel); // NoteOn on selected channel
320                 USARTWriteByte(note); // Note
321                 USARTWriteByte(velocity); // Velocity
322                 pause(1000);
323                 USARTWriteByte(128 + midiChannel); // NoteOff on selected channel
324                 USARTWriteByte(note); // Note
325                 USARTWriteByte(velocity); // Velocity
326 
327             }
328 
329 
330 
331         }
332     }
333
```

*Figure 14*

Figure 14 shows the final code of the program, which handles MIDI message sending. If RC5 is pressed, a MIDI note on message will be sent using the variables

declared, which will have values set by the user. After a second, a MIDI note off message is sent.

# *CHECKLIST*

| CRITERIA | MET? |
|---|---|
| Multiplexing of 3 or more inputs | Yes. |
| Two or more analogue inputs | Yes. |
| One external control | Yes. |
| 1 or more forms of feedback | Yes. |
| Effective programming of ADCs | Yes. |
| Signal conditioning on controls | Yes. |
| MIDI routines functional and demonstratable | Yes. |
| Dynamic control | Yes. |
| Selection of a different MIDI channel under user control | Yes. |
| Appropriate MIDI message depending on controller types required to control an external source | Yes. |
| Effective programming of the PIC using interrupts where applicable | No interrupts. |
| Control Pitch, Amplitude of one or more audio inputs | Yes. |