The functional units on this demonstration bread board include a PIC18F4525 processor, a D/A converter, an LCD display, a waveform/noise generator, an analog ECG simulator, and an ECG amplifier. The system is programmed with the MPLab IDE v8.89 via the ICD 3 programmer. The schematics, bill of materials, and C source code are attached in the end. The breadboard is shown below with the ICD 3 programmer connected. The ECG leads are also shown.

The picture on the next page shows the connectors for attaching two oscilloscope probes (D/A out and Wave out), the potentiometer for adjusting the input voltage to A/D channel AN2, the potentiometer for adjusting the ECG gain, and the connector for attaching the ECG leads.
The system has been programmed to perform the following 10 functions.

**Function 00  
Binary counter**

The binary count is displayed via the 4 LED's driven by RB4-RB7. The count is also sent to the D/A converter to generate a ramp signal. The ramp can be used to verify the linearity and offset of the D/A converter. The offset can be eliminated by adjusting Vref- (pin 12) of the DAC0800. Currently this voltage is set at 1.2 V with a voltage divider circuit (1.5 KΩ and 4.7 KΩ).

**Function 01  
ECG Simulation**

Digitally simulated ECG is shown on channel 2 (cyan), compared to a real ECG waveform recorded from a human subject on channel 1 (orange). The timer TMR0 is set to generate interrupts at 1KHz, which provides a temporal resolution of 1 ms for the synthetic ECG waveform.

**Function 02  
Echo (A/D – D/A)**

The analog signal channel 1 (orange) is from the on-board waveform generator that produces a 9-Hz square wave mixed with simulated 60 Hz noise. The analog signal is acquired by the on-chip D/A via channel AN0 (pin 2) at a sampling rate of 240 Hz and directly outputted to the D/A converter as shown on channel 2 (cyan).
**Function 03**
**Echo @ fs 17 Hz**

The sampling rate can be adjusted via the 100 KΩ potentiometer connected to AN2 (pin 4). A 47 µF capacitor is temporarily connected between AN0 (pin 2) and ground to change the square wave closer to a sine wave. A set of sampling rates ranging from 16 Hz to 228 Hz is programmed into the processor. The purpose is to observe the effect of sampling rate. For example, aliasing is demonstrated in the right figure with a sampling rate of 17 Hz, which is just below the Nyquest sampling rate for the 9 Hz signal. The next figure shows the result with a sampling rate of 70 Hz.

**Function 04**
**Derivative**

A digital differentiator is implemented according to the following backward difference:

\[ y[n] = x[n] - x[n-1] \]

The sampling rate is 240 Hz.
**Function 05**

**Low-pass filter**

An FIR low-pass filter is implemented according to the following equation:

\[ y[n] = \frac{x[n] + 2x[n-1] + x[n-2]}{4} \]

The sampling rate is 240 Hz.

**Function 06**

**Hi-freq enhance**

An FIR high-frequency enhancement filter is implemented according to the following equation:

\[ y[n] = 2x[n] - \frac{x[n] + 2x[n-1] + x[n-2]}{4} \]

The sampling rate is 240 Hz.

**Function 07**

**60Hz notch filtr**

An FIR 60-Hz notch filter is implemented according to the following equation:

\[ y[n] = \frac{x[n] + x[n-2]}{2} \]

The sampling rate is 240 Hz.
**Function 08**
**Median filter**

A median filter is implemented by performing a bubble sort of the present sample point and the past 8 sample points. The output is the median of the 9 points. The result in the figure shows the effect of edge-preserving smoothing of the median filter. The 60 Hz noise is eliminated, while the edges are preserved. The filter output shows a delay of 25 ms, which is longer than those of the previous filters.

---

**Function 09**
**HR: 74 bpm**

A heart meter is implemented by executing the multiplication of backward differences (MOBD) algorithm for QRS detection. The sampling rate is set at 200 Hz. Figure on the right shows the ECG signal recorded from a human subject and the nonlinear transform of the MOBD algorithm. The MOBD algorithm significantly enhances the signal-to-noise ratio readily for the threshold detection. Except for the QRS complex other components such as the P wave, the T wave, and the noise are reduced to the baseline.

The next figure shows the analog simulated ECG and the MOBD nonlinear transform. The simulated ECG contains 60 Hz noise, which is effectively eliminated by the MOBD algorithm.
### Bill of Materials

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Part number</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Processor</td>
<td>PIC18F4525</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>D/A converter</td>
<td>DAC0800</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Voltage regulator (5V, 1A)</td>
<td>LM7805</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Voltage Inverter</td>
<td>LMC7660</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Instrumentation amplifier</td>
<td>AD620</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Dual timer</td>
<td>LM556</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Timer</td>
<td>LM555</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Quad op amp</td>
<td>LM324</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>NPN transistor</td>
<td>2N2222</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Serial LCD display (16x2)</td>
<td>SerLCD</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Buzzer</td>
<td>Mini buzzer (12V)</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Ceramic resonator</td>
<td>4 MHz</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>LED</td>
<td>green</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>yellow</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>Resistor (x44)</td>
<td>220 Ω</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>330 Ω</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>470 Ω</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 KΩ</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>1.2 KΩ</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.5 KΩ</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>2.7 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>3.3 KΩ</td>
<td>4</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>4.7 KΩ</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>10 KΩ</td>
<td>12</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>15 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>22 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>27 KΩ</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>47 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>100 KΩ</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>330 KΩ</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>470 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>1 MΩ</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>Potentiometer (x2)</td>
<td>1 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>100 KΩ</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>Capacitor (x16)</td>
<td>.01 μF (103)</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>.1 μF (104)</td>
<td>1</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>.22 μF (224)</td>
<td>1</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>1 μF (electrolytic)</td>
<td>2</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>4.7 μF (electrolytic)</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>10 μF (electrolytic)</td>
<td>3</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td>100 μF (electrolytic)</td>
<td>1</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>220 μF (electrolytic)</td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>Switch</td>
<td>Momentary tact switch</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>Pushbutton on/off switch</td>
<td>1</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>SPDP slide switch</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>9V battery or AC adapter</td>
<td>1</td>
</tr>
<tr>
<td>43</td>
<td>Power supply</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Include Files

```c
#include "p18cxxx.h"
#include "timers.h"
#include "math.h"
#include "stdlib.h"
```

# Configure the Microcontroller PIC18f4525

```c
#pragma config OSC = XT
#pragma config WDT = OFF
#pragma config PWRT = OFF
#pragma config FCMEN = OFF
#pragma config IESO = OFF
#pragma config BOREN = ON
#pragma config BORV = 2
#pragma config WDTPS = 128
#pragma config PBADEN = OFF
#pragma config DEBUG = OFF
#pragma config LVP = OFF
#pragma config STVREN = OFF
#define _XTAL_FREQ 4000000
```

# Define Prototype Functions

```c
unsigned char ReadADC();
void rx_int (void);
void _highPriorityInt(void);
void Backlight(unsigned char state);
void SetPosition(unsigned char position);
void PrintLine(rom unsigned char *string, unsigned char numChars);
void PrintInt(int value, unsigned char position);
void PrintNum(unsigned char value1, unsigned char position1);
void SetupSerial();
void ClearScreen();
void SetupADC(unsigned char channel);
void Delay_ms(unsigned int x);
```

# Global Variables

```c
unsigned char function, mode, update, debounce0, debounce1, LEDcount, output, counter, i, j;
unsigned char data0, data1, data2, array[9], rank[9], do_median, do_MOBD, refractory, display;
unsigned char temp, sampling[16], TMRcntH[16], TMRcntL[16], sampling_H, sampling_L;
int dummy, d0, d1, d2, mobd, threshold, rri_count, hr;
```

```c
unsigned char ReadADC() { /* Start A/D, read from an A/D channel */
    unsigned char ADC_VALUE;
    ADCON0bits.GO = 1; // Start the AD conversion
    while(!PIR1bits.ADIF) continue; // Wait until AD conversion is complete
    ADC_VALUE = ADRESH; // Return the highest 8 bits of the 10-bit AD conversion
    return ADC_VALUE;
}
```

`#pragma code _highPriorityInt=0x8`

```c
void rx_int (void){// Set up highpriority interrup vector
    _asm goto _highPriorityInt _endasm
}
```
#pragma interrupt _highPriorityInt

void _highPriorityInt(void) {  /*------------------------------------------------------------------
** high priority interrupt service routine  -------------------------------------------------------------------**/
checkflags:
  if (INTCONbits.TMR0IF == 1) {  // When there is a timer0 overflow, this loop runs
    INTCONbits.TMR0IE = 0;  // Disable TMR0 interrupt
    INTCONbits.TMR0IF = 0;  // Reset timer 0 interrupt flag to 0
    if (debounce0 != 0) debounce0--;  // switch debounce delay counter for INT0
    if (debounce1 != 0) debounce1--;  // switch debounce delay counter for INT1
    TMR0H = 0xF0;  // Reload TMR0 for 4.167 ms count, Sampling rate = 240 Hz
    TMR0L = 0x7C;  // 0xFFFF-0xEFB8 = 0x1047 = 4167, adjust for delay by 68 us
    switch (function) {
      case 1:  // Function 1: ECG simulation
        TMR0H = 0xFC;  // Reload TMR0 for 1 ms count, sampling rate = 1KHz
        TMR0L = 0xF8;  // 0xFFFF-0xFC17 = 0x3E8 = 1000, adjust for delay by 227 us
        PORTBbits.RB4 = !PORTBbits.RB4;  // Toggle RB4 (pin 37) for frequency check
        switch (mode) {
          case 0:  // P wave up
            counter++;  output++;  if (counter == 30) mode++;
            break;
          case 1:  // P wave flat
            counter--;  if (counter == 0) mode++;
            break;
          case 2:  // P wave down
            counter++;  output--;  if (counter == 30) mode++;
            break;
          case 3:  // PR segment
            counter++;  if (counter == 100) mode++;
            break;
          case 4:  // QRS complex - Q
            counter++;  output -= 3;
            if (counter == 105) {  counter = 0;
              mode++;
            }
            break;
          case 5:  // QRS complex - R up
            counter++;  output += 6;  if (counter == 30) mode++;
            break;
          case 6:  // QRS complex - R down
            counter++;  output -= 6;  if (counter == 62) mode++;
            break;
          case 7:  // QRS complex - S
            counter++;  output += 3;
            if (counter == 71) {  counter = 0;
              mode++;
            }
            break;
          case 8:  // ST segment
            counter++;
            if (counter == 89) {    counter = 0;
              mode++;
            }
            break;
          case 9:  // T wave up
            counter++;  output++;  if (counter == 55) mode++;
            break;
          case 10:  // T wave flat
            counter++;  if (counter == 110) mode++;
break;  
    case 11:  
        // T wave down
        counter++;  
        output--;  
        if (counter == 165) mode++;  
        break;
    case 12:  
        // End ECG
        counter--;  
        if (counter == 0) mode++;  
        break;
    case 13:  
        // Reset ECG
        counter++;  
        if (counter == 202) {
            counter = mode = 0;  
            output = 50;
        }
        break;
    }
PORTD = output;
break;

case 2:  // Function 2: Echo
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    PORTD = data0;  // Echo back
    PORTBbits.RB4 = !PORTBbits.RB4;  // Toggle RB4 (pin 37) for frequency check
    break;

case 3:  // Function 3: Echo (vary rate)
    TMR0H = sampling_H;  // Reload TMR0 high-order byte
    TMR0L = sampling_L;  // Reload TMR0 low-order byte
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    PORTD = data0;  // Echo back
    PORTBbits.RB4 = !PORTBbits.RB4;  // Toggle RB4 (pin 37) for frequency check
    break;

case 4:  // Function 4: Derivative
    data1 = data0;  // Store previous data points
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    dummy = (int)data0 - data1 + 128;  // Take derivative and shift to middle
    PORTD = (unsigned char)dummy;  // Output to D/A
    break;

case 5:  // Function 5: Low-pass filter
    data2 = data1;  // Store previous data points
    data1 = data0;  
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    dummy = ((int)data0 + data1 + data1 + data2) / 4;  // smoother
    PORTD = (unsigned char)dummy;  // Output to D/A
    break;

case 6:  // Function 6: High-frequency enhancement filter
    data2 = data1;  // Store previous data points
    data1 = data0;  
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    dummy = ((int)data0 + data1 + data1 + data2) / 4;  // smoother
    dummy = data0 + data0 - dummy;
    PORTD = (unsigned char)dummy;  // Output to D/A
    break;

case 7:  // Function 7: 60Hz notch filter
    data2 = data1;  // Store previous data points
    data1 = data0;  
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    dummy = ((int)data0 + data2) / 2;  // 60 Hz notch
    PORTD = (unsigned char)dummy;  // Output to D/A
    break;

case 8:  // Function 8: Median filter
    data0 = ReadADC();  // Read A/D and save the present sample in data0
    do_median = 1;  // Flag main() to do median filter
    break;
case 9:// Function 9: Heart rate meter
TMR0H = 0xED; // Reload TMR0 for 5 ms count, sampling rate = 200 Hz
TMR0L = 0x44; // 0xFFFF-0xED44 = 0x12BB = 4795, (205 us delay)
data1 = data0; // Move old ECG sample to data1
data0 = ReadADC(); // Store new ECG sample from ADC to data0
PORTBbits.RB2 = !PORTBbits.RB2; // Toggle RB2 for sampling rate check
do_MOBD = 1; // Flag main() to do MOBD for QRS detection
break;
}
INTCONbits.TMR0IE = 1; // Enable TMR0 interrupt

if (INTCONbits.INT0IF == 1) {
  INTCONbits.INT0IE = 0; // Disable interrupt
  INTCONbits.INT0IF = 0; // Reset interrupt flag
  if (debounce0 == 0) {
    if (function <= 0) function = 9; // Set function range 0-9
    else function--;
    if (function == 9) SetupADC(1); // ECG comes from AN1 channel
    else SetupADCC(); // Others come from AN0 channel
    update = 1; // Signal main() to update LCD dispaly
    debounce0 = 10; // Set switch debounce delay counter decremented by TMR0
  }
  INTCONbits.INT0IE = 1; // Enable interrupt
goto checkflags; // Check again in case there is a timer interrupt
}

if (INTCON3bits.INT1IF == 1) {
  INTCON3bits.INT1IE = 0; // Disable interrupt
  INTCON3bits.INT1IF = 0; // Reset interrupt flag
  if (debounce1 == 0) {
    if (function >= 9) function = 0; // Set function range 0-9
    else function++;
    if (function == 9) SetupADC(1); // ECG comes from AN1 channel
    else SetupADCC(); // Others come from AN0 channel
    update = 1; // Signal main() to update LCD dispaly
    debounce1 = 10; // Set switch debounce delay counter decremented by TMR0
  }
  INTCON3bits.INT1IE = 1; // Enable interrupt
goto checkflags; // Check again in case there is a timer interrupt
}

void Transmit(unsigned char value) {
  while (!PIR1bits.TXIF) continue; // Wait until USART is ready
  TXREG = value; // Send the data
  while (!PIR1bits.TXIF) continue; // Wait until USART is ready
  Delay_ms (5); // Wait for 5 ms
}

void ClearScreen(){
  Transmit(254); // See datasheets for Serial LCD and HD44780
  Transmit(0x01); // Available on our course webpage
}

void Backlight(unsigned char state){
  Transmit(124);
  if (state) Transmit(0x9D); // If state == 1, backlight on
  else Transmit(0x81); // otherwise, backlight off
}

void SetPosition(unsigned char position){
  Transmit(254);
}
Transmit(128 + position);
}

void PrintLine(rom unsigned char *string, unsigned char numChars){    /**** Print characters ****/
unsigned char count;
for (count=0; count<numChars; count++) Transmit(string[count]);
}

void PrintInt(int value, unsigned char position){    /******** Print number at position **********/
int units, tens, hundreds, thousands;
setPosition(position);    // Set at the present position
if (value > 9999) {
    PrintLine((rom unsigned char*)"Over", 4);
    return;
}
if (value < -9999) {
    PrintLine((rom unsigned char*)"Under", 5);
    return;
}
if (value < 0) {
    value = -value;
    Transmit(45);
} else Transmit(43);
    thousands = value / 1000;    // Get the thousands digit, convert to ASCII and send
if (thousands != 0) Transmit(thousands + 48);
    value = value - thousands * 1000;
    hundreds = value / 100;    // Get the hundreds digit, convert to ASCII and send
Transmit(hundreds + 48);
    value = value - hundreds * 100;
    tens = value / 10;    // Get the tens digit, convert to ASCII and send
Transmit(tens + 48);
    units = value - tens * 10;
Transmit(units + 48);    // Convert to ASCII and send
}

void PrintNum(unsigned char value1, unsigned char position1){    /** Print number at position ***/
int units, tens, hundreds, thousands;
setPosition(position1);    // Set at the present position
hundreds = value1 / 100;    // Get the hundreds digit, convert to ASCII and send
if (hundreds != 0) Transmit(hundreds + 48);
else Transmit(20);
    value1 = value1 - hundreds * 100;
    tens = value1 / 10;    // Get the tens digit, convert to ASCII and send
Transmit(tens + 48);
    units = value1 - tens * 10;
Transmit(units + 48);    // Convert to ASCII and send
}

void SetupSerial(){    /*********** Set up the USART Asynchronous Transmit (pin 25) ************/
TRISC = 0x80;    // Transmit and receive, 0xC0 if transmit only
SPBRG = 25;    // 9600 BAUD at 4MHz: 4,000,000/(16x9600) - 1 = 25.04
TXSTAbits.TXEN = 1;    // Transmit enable
TXSTAbits.SYNC = 0;    // Asynchronous mode
RCSTAbits.CREN = 1;    // Continuous receive (receiver enabled)
RCSTAbits.SPEN = 1;    // Serial Port Enable
TXSTAbits.BRGH = 1;    // High speed baud rate
}

void SetupADC(unsigned char channel){    /******** Configure A/D and Set the Channel **********/
TRISA = 0b11111111;    // Set all of Port A as input
void main(){} //********** Generate a delay for x ms, assuming 4 MHz clockEEDED/  
unsigned char y;  
for(;x > 0; x--) for(y=0; y< 82;y++);  
}
ClearScreen();  // Clear screen and set cursor to first position
PrintLine((rom unsigned char*)" BME 361 Demo", 14);
SetPosition(64);  // Go to beginning of Line 2;
PrintLine((rom unsigned char*)" Biomeasurement",15); // Put your trademark here
Delay_ms(3000);
ClearScreen();  // Clear screen and set cursor to first position
PrintLine((rom unsigned char*)"Function", 8);
while (1) {
    if (update) {  // The update flag is set by INT0 or INT1
        update = 0;  // Reset update flag
        PrintNum(function, 8); // Update the function number on LCD display
        if (function != 0) LEDcount = PORTB = 0; // Reset LED's
        SetPosition(64);  // Go to beginning of Line 2;
        switch (function) {
            case 0: PrintLine((rom unsigned char*)"Binary counter ",16); break;
            case 1: PrintLine((rom unsigned char*)"ECG simulation ",16); break;
            case 2: PrintLine((rom unsigned char*)"Echo (A/D - D/A)",16); break;
            case 3: PrintLine((rom unsigned char*)"Echo @ fs Hz",16); break;
            case 4: PrintLine((rom unsigned char*)"Derivative ",16); break;
            case 5: PrintLine((rom unsigned char*)"Low-pass filter ",16); break;
            case 6: PrintLine((rom unsigned char*)"Hi-freq enhance ",16); break;
            case 7: PrintLine((rom unsigned char*)"60Hz notch filtr",16); break;
            case 8: PrintLine((rom unsigned char*)"Median filter ",16); break;
            case 9: PrintLine((rom unsigned char*)"HR = bpm ",16); break;
        }
    }
    switch (function) {
        case 0: // Function 0: Binary counter
            LEDcount++;
            PORTB = LEDcount & 0b11110000;
            // Mask out the lower 4 bits
            PORTD = LEDcount;
            // Output ramp to verify linearity of the D/A
            Delay_ms(10);
            // Delay to slow down the counting
            break;
        case 3: // Function 3: Echo (vary rate)
            INTCONbits.TMR0IE = 0;
            SetupADC(2);
            // Switch to A/D channel AN2
            counter = ReadADC();
            // Read potentiometer setting from AN2
            SetupADC(0);
            // Switch to A/D channel AN0
            counter = counter >> 4;
            // Scale it to 0-15
            temp = sampling[counter];
            // Display sampling rate
            PrintNum(temp,74);
            sampling_L = TMRcntL[counter];
            // Load TMR0 low-order byte
            sampling_H = TMRcntH[counter];
            // Load TMR0 high-order byte
            INTCONbits.TMR0IE = 1;
            // Enable TMR0 interrupt
            Delay_ms(1000);
            // Delay to slow down the counting
            break;
        case 8: // Function 8: Median filter (9-point)
            if (do_median) {
                do_median = 0;
                INTCONbits.TMR0IE = 0;
                // Disable TMR0 interrupt
                for (i=8; i>0; i--) array[i] = array[i-1];
                // Store the previous 8 points
                array[0] = data0;
                // Get new data point from A/D
                for (i=0; i<9; i++) rank[i] = array[i];
                // Make a copy of data array
                for (i=0; i<5; i++) {
                    // Perform a bubble sort
                    for (j=i+1; j<9; j++) {
                        if (rank[j] < rank[i]) {
                            temp = rank[i];
                            // Swap
                            rank[i] = rank[j];
                            rank[j] = temp;
                        }
                    }
                }
            }
PORTD = rank[4];  // Median is at rank[4] of rank[0-8]
INTCONbits.TMR0IE = 1;  // Enable TMR0 interrupt
break;
case 9:  // Function 9: Multiplication of Backward Differences (MOBD)
  if (do_MOBD){  // MOBD = Multiplication of Backwards Differences
    INTCONbits.TMR0IE = 0;  // Disable TMR0 interrupt
    do_MOBD = 0;  // Reset new_data flag
    d2 = d1;  // Move oldest difference to d2
    d1 = d0;  // Move older difference to d1
    d0 = (int)data0 - data1;  // Store new difference in d0, (int) casting important
    rri_count++;  // Increment RR-interval
    mobd = 0;  // mobd = 0, unless sign consistency is met:
    if (d0 > 0 && d1 > 0 && d2 > 0){  // (1) If 3 consecutive positive differences
      mobd = d0 * d1;  // Multiply first two differences
      mobd = mobd >> 3;  // Scale down (divide by 8)
      mobd = mobd * d2;  // Multiply the oldest difference
    }
    if (d0 < 0 && d1 < 0 && d2 < 0){  // (2) If 3 consecutive negative differences
      d0 = -d0;  // Take absolute value of differences
      d1 = -d1;
      d2 = -d2;
      mobd = d0 * d1;  // Multiply first two differences
      mobd = mobd >> 3;  // Scale down (divide by 8)
      mobd = mobd * d2;  // Multiply the oldest difference
    }
    if (refractory){  // Avoid detecting extraneous peaks after QRS
      refractory++;
      if (refractory == 40){  // Delay for 200 ms
        refractory = 0;  // Reset refractory flag to 0
        PORTBbits.RB3 = 0;  // Turn buzzer/LED off (Pin 36)
      }
    }
    else if (mobd > threshold){  // If a peak is detected,
      refractory = 1;  // Set refractory flag
      PORTBbits.RB3 = 1;  // Turn buzzer/LED on (Pin 36)
      display = 1;  // Set display flag
    }
  }
  PORTD = (unsigned char)mobd;  // Output mobd value to Port D
  INTCONbits.TMR0IE = 1;  // Enable TMR0 interrupt
}
if (display){  // Display Heart Rate in 3 digits
  hr = 12000/rri_count;  // 60/0.005 = 12000
  rri_count = 0;  // Reset RRI counter
  PrintNum(hr, 71);  // Isolates each digit and displays
  display = 0;  // Reset display flag
}
break;
}