A wide range signal generator

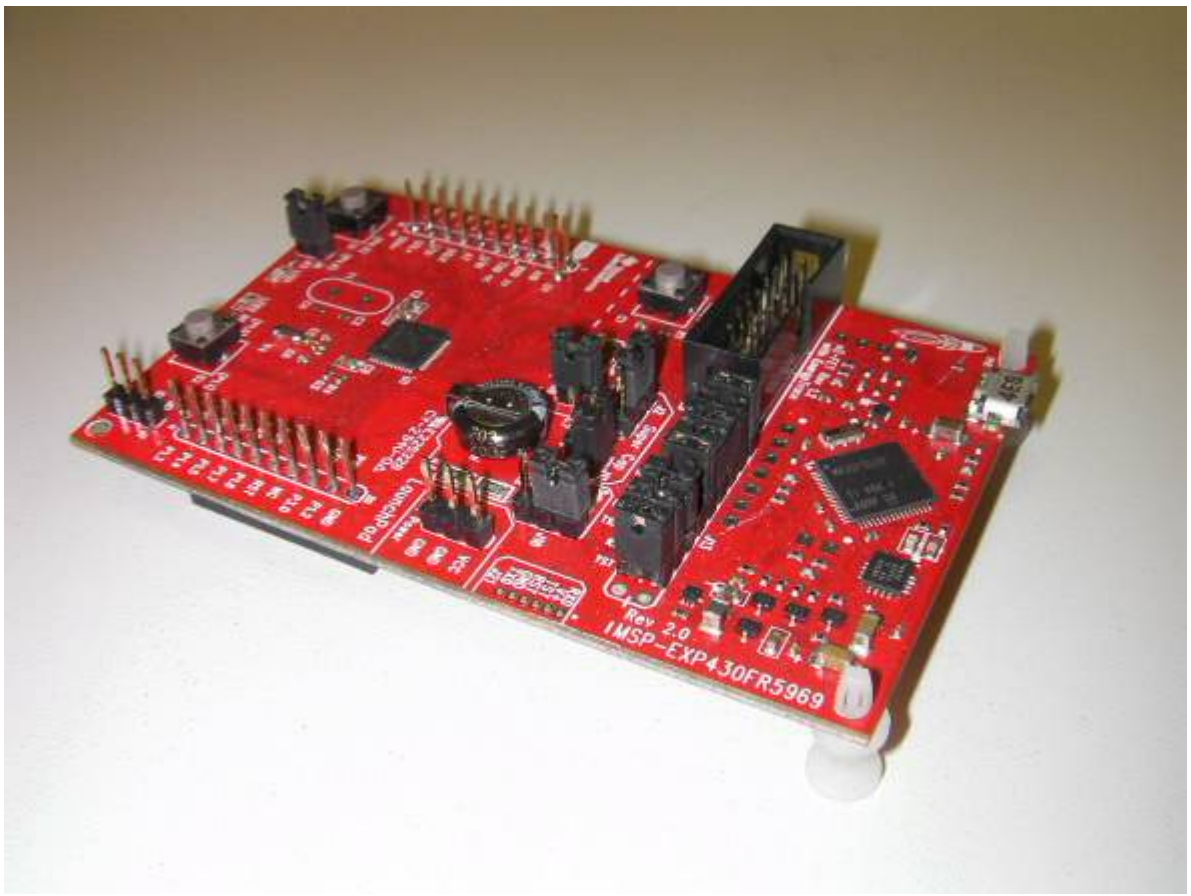
Using a series of ready built modules you can fairly easily build a 34MHz to 4400MHz signal generator.

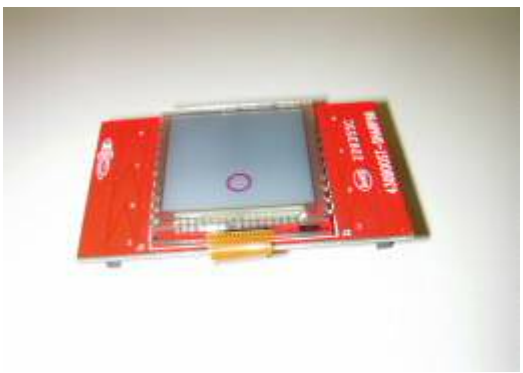
The main module is based on the ADF4351 (for a slightly higher starting frequency of 134MHz, the ADF4350 can be used without any other changes).

This could be controlled by using any Arduino board, but my preference was to use an Energia MSP430FR5969 'LaunchPad' because this has the advantages of internal non-volatile memory and an LCD board which simply plugs into it. In other words, the frequency amplitudes, call sign and locator values do not disappear when the device is unplugged

To set up the parameters, a rotary encoder switch with an integrate push button was wired into the 'LaunchPad' along with a simple 0.1uF capacitor down to ground on each switch (x3). P1_4, P1_5 are connected to the encoder part and P1_3 to the pushbutton.

Pin P3_4 is the Slave Select for the ADF435x along with P2_2 (SCLK), P1_6 (MOSI)

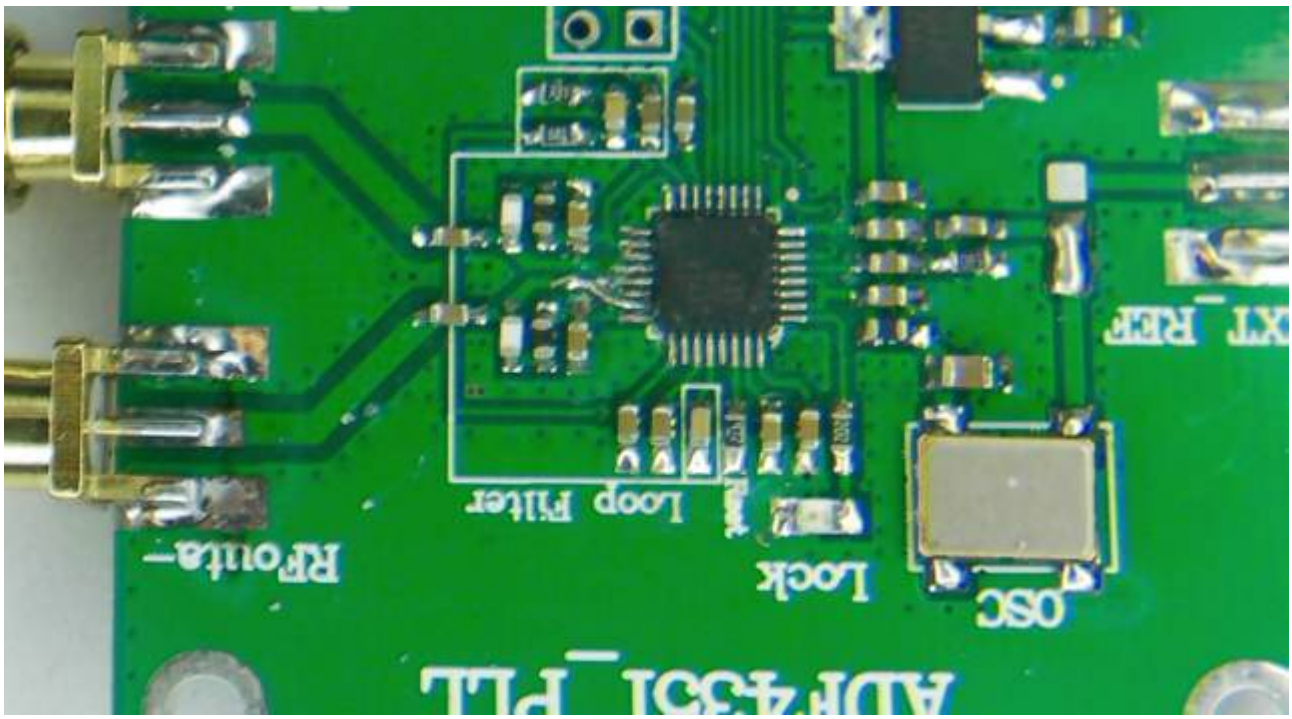




Like the Arduino range, Energia comes with some standard libraries for driving the graphical display and the SPI bus used to control the ADF435x. Once you have downloaded Energia (energia.nu) you will need to add two libraries; RTC-B-master (https://github.com/spirilis/RTC_B) and my ADF435x library (https://github.com/mcafzap/Energia-ADF435x) to the ~\documents\energia\libraries directory. Then copy the sketch into ~\documents\energia.

The selected menu item has a chevron ('<' or 'v') next to it; you can set the frequency, enter your callsign and locator and set the output levels from each of the 2 outputs (-4, -1, 2 and 5 dBm). Please see photos.
There is a simple time out implementation whilst adjusting the parameters; after 3 minutes of no activity the 'home' screen is restored and the beacon restarts.

At present, the auxiliary output transmits the callsign and locator in Morse once a minute, at a fixed 20wpm. However, this output is not routed to the SMA connectors. To do this, you need to carefully modify the ADF435x module by cutting the track from the main output pin 13 and reconnect it to an auxiliary output pin 14 or 15. (See below.)



Conceivably, you could modify the sketch to output WISPR or JT65 instead.

I am using the fractional-N mode for the PLL in the ADF device; this allows a greater resolution down to 100Hz but it ought to be relatively easy to improve this to 10Hz if you are prepared to amend the library. You could also add a phase modulation or improve the phase noise (at the expense of resolution).

Our tests reveal a typical square wave spectrum with 3rd and 5th harmonics available up to about 13GHz. Internally, there are 3 VCOs and depending upon the frequency chosen (i.e. no division of the VCO output) you may have very little harmonic output, but in general (and without a spectrum analyser) always use a low pass filter if you intend to transmit directly.

PARTS.

MSP430FR5969 LaunchPad – Farnell Components
LCD Booster Pack (430Boost SHARP96) – Farnell Components
ADF4351 breakout board – Ebay
12mm rotary encoder switch with keyswitch (model 10XSKU026777) – Ebay
Box about 165x85x55 – Maplin code N18GC (A clear box might avoid the need for machining slots.)
Knob – Maplin code YX02C

SOFTWARE
Energia sketch: 'SigGen' and the libraries mentioned in the links above.

The following is the main sketch SigGen.ino and should be cut and pasted into your Energia development environment. Please ensure you have all the required libraries in the right place or you will get errors.

```
// Include application, user and local libraries
#include "SPI.h"
#include "OneMsTaskTimer.h"
#include "LCD_SharpBoosterPack_SPI.h"
#include <ADF435x.h>
#include <RTC_B.h>
// Morse storage method -Mark VandeWettering http://brainwagon.org/2009/11/14/another-try-at-an-arduino-based-morse-beacon/

#define COM_PIN P3_4 // sets pin 32 to be the slave-select pin for PLL
#define PERSIST __attribute__((section(".text")))
ADF435x PLL(COM_PIN); // declares object PLL of type ADF4350. Will initialize it below.
#define SPEED  (20)
#define DOTLEN  (1200/SPEED)
#define DASHLEN  (3*(1200/SPEED))

char callsign[] = "M0SII\0";            // The CW callsign. Only A-Z, 0-9 and /
char locator[] = "IO83UL\0";            // The CW locator. Only A-X, 0-9 and /
char levels[4][7] = {
  "-4dBm\0", "-1dBm\0", " 2dBm\0", " 5dBm\0"};
LCD_SharpBoosterPack_SPI myScreen;
const int buttonPin     = P1_3;             // connect the pushbutton to this LP pin
const int encoderPin1   = P1_4;             // connect one encoder pin to this LP pin
const int encoderPin2   = P1_5;             // connect the other encoder pin to this LP pin
const uint32_t minVal   =   3437500;        // minimum value that will be allowed as input
const uint32_t maxVal   = 440000000;    // maximum value that will be allowed as input
const uint32_t startInc     = 1;            // values increase/decrease by this value to start
const long minInc       = 1;            // minimum increment/decrement allowed when turning
const long maxInc       = 100000000;        // maximum increment/decrement allowed when turning
const long divInc       = 10;           // factor for decreasing the increment/decrement
const char ltab[] = {
  0b101,            // A
  0b11000,          // B
  0b11010,          // C
  0b1100,           // D
  0b10,             // E
  0b10010,          // F
  0b1110,           // G
  0b10000,          // H
  0b100,            // I
  0b10111,          // J
  0b1101,           // K
  0b10100,          // L
  0b111,            // M
  0b110,            // N
  0b1111,           // O
  0b10110,          // P
  0b11101,          // Q
```

```
  0b1010,           // R
  0b1000,           // S
  0b11,            // T
  0b1001,           // U
  0b10001,           // V
  0b1011,           // W
  0b11001,           // X
  0b11011,           // Y
  0b11100           // Z
}
;

const char ntab[] = {
  0b111111,          // 0
  0b101111,          // 1
  0b100111,          // 2
  0b100011,          // 3
  0b100001,          // 4
  0b100000,          // 5
  0b110000,          // 6
  0b111000,          // 7
  0b111100,          // 8
  0b111110           // 9
}
;



uint8_t state = false, mMenu=0, timeout = 0;
uint8_t secs, mins, i, j, k = 0, level=0;
uint16_t count = 0;
volatile boolean encoderLH1 = false;
volatile boolean encoderLH2 = false;
uint8_t m, e, f, g, displayScreen=0, adjPointer = 0;
uint8_t x=0, diggy;
char mark[14];
char freq[14];
uint32_t fAddition = minInc;
uint32_t noddy;
// variables used to track whether or not a change has occurred
uint32_t lastEncoderVal = 1;
uint32_t lastEncoderInc = 0;

uint32_t encoderVal PERSIST;
uint32_t encoderInc PERSIST;
uint32_t frock PERSIST; // 0.1KHz freq to be set... 144,100,000Hz
uint8_t mainLevel PERSIST;
uint8_t auxLevel PERSIST;

// Add setup code
void setup() {
  pinMode(P3_6, OUTPUT);
  digitalWrite(P3_6, HIGH); // enable Tx
  Serial.begin (115200);
  Serial.print("Hello World\n");
  pinMode(buttonPin, INPUT_PULLUP);
  pinMode(encoderPin1, INPUT_PULLUP);
  pinMode(encoderPin2, INPUT_PULLUP);
```

```
  attachInterrupt(encoderPin1, ISR_Encoder1, CHANGE);  // interrupt for encoderPin1
  attachInterrupt(encoderPin2, ISR_Encoder2, CHANGE);  // interrupt for encoderPin2
  attachInterrupt(buttonPin, ISR_Button, FALLING);     // interrupt for encoder button
  myScreen.begin();
  myScreen.clear();
  myScreen.flush();
  rtc.begin(WEDNESDAY, 1, 1, 2015, 12, 00, 00);  // 8/20/2014 15:21
  rtc.attachPeriodicInterrupt(1, Int2); //interrupt once a second
  mainLevel &= 0x0003;
  auxLevel &= 0x0003;
  PLL.setFeedbackType(0); //integer PLL...
  PLL.initialize(14400000, 10000000); // initialize the PLL - reference 10Mhz
  //  encoderVal = startVal;
  lastEncoderVal = encoderVal;
  displayScreen=0;
  //  frock = encoderVal;
  setDisplay();
  delay(1000);
  //  myScreen.setFont(0);
}

// Add loop code
void loop() {
  if ((mins == 1) && (m == 0)) {
    mins = 0;
    //    PLL.auxDisable();

    uint8_t grog;
    for (i=0;i<6;i++) {
      uint8_t greg = callsign[i];
      if (greg==0x0)  grog = 1;
      else  if (greg < 0x41) {
        grog = ntab[greg-48];
      }
      else grog = ltab[greg-65];
      sendMorse(grog);
    }
    for (i=0;i<7;i++) {
      uint8_t greg = locator[i];
      if (greg==0x0)  grog = 1;
      else  if (greg < 0x41) {
        grog = ntab[greg-48];
      }
      else grog = ltab[greg-65];
      sendMorse(grog);
    }

  }

  // check for change in encoder value
  if (lastEncoderVal != encoderVal) {
    Serial.print("\n Screen No ");
    Serial.println(displayScreen);
    // If on 'home' screen, select a new screen. Else alter value or Return
    if (displayScreen==0) {
      adjPointer = 0;
      switch (mMenu) {
      case 0:
```

```
    displayScreen = 1;
    break;
  case 1: // Callsign
    displayScreen = 2;
    break;
  case 2: // Location
    displayScreen = 3;
    break;
  case 3: // Output Levels
    displayScreen = 4;
    break;
  }
}

else {
  if (adjPointer == 0) {
    displayScreen = 0;
    m = 0;
  }

  if (displayScreen == 1) {
    i = adjPointer-1;
    fAddition = 10;
    while (i) {
      fAddition *= 10;
      --i;
    }
    Serial.print("\nAdded amount ");
    Serial.print(fAddition);
    if (encoderVal > lastEncoderVal) frock += fAddition;
    else frock -= fAddition;
    if (frock > maxVal) frock = maxVal;
    if (frock < minVal) frock = minVal;
  }

  if (displayScreen == 2) {
    if (adjPointer) {
      if (encoderVal > lastEncoderVal) {
        j = 5 - adjPointer;
        callsign[j] += 1;
        if (callsign[j] > 'Z') callsign[j] = '0';
        if ((callsign[j] > '9') && (callsign[j] < 'A')) callsign[j] = 'A';
      }
      else {
        callsign[j] -= 1;
        if ((callsign[j] < 'A') && (callsign[j] > '9')) callsign[j] = '9';
        if (callsign[j] < '0') callsign[j] = 'Z';
      }
    }
  }

  if (displayScreen == 3) {
    if (adjPointer) {
      j = 6 - adjPointer;
      if (encoderVal > lastEncoderVal) {
        locator[j] += 1;
        if (locator[j] > 'X') locator[j] = '0';
        if ((locator[j] > '9') && (locator[j] < 'A')) locator[j] = 'A';
```

```
        }
        else {
          locator[j] -= 1;
          if ((locator[j] < 'A') && (locator[j] > '9')) locator[j] = '9';
          if (locator[j] < '0') locator[j] = 'Z';
        }
      }
    }

    if (displayScreen == 4) { //set levels
      if (adjPointer==1) {
        if (encoderVal > lastEncoderVal) ++mainLevel;
        else --mainLevel;
        mainLevel &= 0x03;
      }
      if (adjPointer==2) {
        if (encoderVal > lastEncoderVal) ++auxLevel;
        else --auxLevel;
        auxLevel &= 0x03;
      }
    }
  }
  lastEncoderVal = encoderVal;
  setDisplay();
  setSynth();
}

// check for change in button
if (lastEncoderInc != encoderInc) {
  // Move pointer to next changeable value
  switch (displayScreen) {
  case 0: //selects whichever setup page is pointed at by mMenu
    ++mMenu;
    mMenu &= 0x03;
    break;
  case 1:
    ++adjPointer;
    if (adjPointer > 8) adjPointer = 0;
    break;
  case 2:
    ++adjPointer;
    if (adjPointer > 6) adjPointer = 0;
    break;
  case 3:
    ++adjPointer;
    if (adjPointer > 6) adjPointer = 0;
    break;
  case 4:
    ++adjPointer;
    if (adjPointer > 2) adjPointer = 0;
    break;
  }
  Serial.print("\nPointer is ");
  Serial.println(adjPointer);

  lastEncoderInc = encoderInc;
  setDisplay();
  m = 0;
```

```
  }
  delay(200);
}


void ISR_Button() {
  m = 1;
  timeout = 0;
  ++encoderInc;  // change the increment amount
}


void ISR_Encoder1(){
  m = 1;
  timeout = 0;
  // Low to High transition?
  if (digitalRead(encoderPin1)==HIGH) {
    encoderLH1 = true;
    if (!encoderLH2) {
      ++encoderVal;                       // increase the value+
    }
  }
  // High-to-low transition?
  if (digitalRead(encoderPin1) == LOW) {
    encoderLH1 = false;
  }
}


void ISR_Encoder2(){
  m = 1;
  timeout = 0;
  // Low-to-high transition?
  if (digitalRead(encoderPin2) == HIGH) {
    encoderLH2 = true;
    if (!encoderLH1) {
      encoderVal -= encoderInc;                       // increase the value+
    }
  }

  // High-to-low transition?
  if ((digitalRead(encoderPin2) == LOW)) {
    encoderLH2 = false;
  }
}


void setDisplay(void) {
  /*
   If 'displayScreen' == 0 then normal screen
   if 'displayScreen' == 1 then set freq screen
   If 'displayScreen' == 2 then set callsign screen
   if 'displayScreen' == 3 then set location screen
   If 'displayScreen' == 4 then set levels screen
   */
  myScreen.clear();
  myScreen.flush();
  setFreqString();
```

```
switch (displayScreen) {
case 0:
  //    Serial.println("Here0");
  myScreen.setFont(1);
  myScreen.text(1, 0,  "FREQ KHz");
  myScreen.text(0, 32, "C/S");
  myScreen.text(0, 48, "LOC");
  myScreen.text(0, 64, "LVL");

  myScreen.text(0, 80, "F C L O ");
  if (mMenu == 0) myScreen.text(11, 80,'<');
  if (mMenu == 1) myScreen.text(34, 80,'<');
  if (mMenu == 2) myScreen.text(57, 80,'<');
  if (mMenu == 3) myScreen.text(84, 80,'<');

  myScreen.setFont(0);
  myScreen.text(10, 18, freq);
  myScreen.text(50, 35, callsign);
  myScreen.text(50, 51, locator);

  myScreen.setCharXY(32, 66);
  for (i=0;i<5;i++)   myScreen.print(levels[mainLevel][i]);
  myScreen.setCharXY(65, 66);
  for (i=0;i<5;i++)   myScreen.print(levels[auxLevel][i]);
  break;

case 1: //set frequency
  myScreen.setFont(1);
  myScreen.text(1, 0,  "SET FREQ");
  myScreen.setFont(0);
  myScreen.text(10, 55, freq);
  for (e = 0; e < 14; e++) mark[e] = 0x20;
  if (adjPointer == 0) {
    mark[2] = 'R';
    mark[3] = 'e';
    mark[4] = 't';
    mark[5] = 'u';
    mark[6] = 'r';
    mark[7] = 'n';
    mark[8] = '\0';
  }
  else {
    if (adjPointer == 8) mark[0] = 'v';
    if (adjPointer == 7) mark[2] = 'v';
    if (adjPointer == 6) mark[3] = 'v';
    if (adjPointer == 5) mark[4] = 'v';
    if (adjPointer == 4) mark[6] = 'v';
    if (adjPointer == 3) mark[7] = 'v';
    if (adjPointer == 2) mark[8] = 'v';
    if (adjPointer == 1) mark[10] = 'v';
    mark[13] = '\0';
  }
  myScreen.text(10, 35, mark);
  break;

case 2: //set callsign
  myScreen.setFont(1);
```

```
  myScreen.text(1, 0,  "CALLSIGN");
  myScreen.setFont(0);
  myScreen.text(40, 45, callsign);

  for (e = 0; e < 14; e++) mark[e] = 0x20;
  if (adjPointer == 0) {
    mark[2] = 'R';
    mark[3] = 'e';
    mark[4] = 't';
    mark[5] = 'u';
    mark[6] = 'r';
    mark[7] = 'n';
    mark[8] = '\0';
  }
  else {
    if (adjPointer == 1) mark[7] = 'v';
    if (adjPointer == 2) mark[6] = 'v';
    if (adjPointer == 3) mark[5] = 'v';
    if (adjPointer == 4) mark[4] = 'v';
    if (adjPointer == 5) mark[3] = 'v';
    if (adjPointer == 6) mark[2] = 'v';
  }

  mark[13] = '\0';
  myScreen.text(22, 35, mark);
  break;

case 3: //set Maidenhead Locator
  myScreen.setFont(1);
  myScreen.text(1, 0,  "LOCATION");
  myScreen.setFont(0);
  myScreen.text(35, 45, locator);

  for (e = 0; e < 14; e++) mark[e] = 0x20;
  if (adjPointer == 0) {
    mark[2] = 'R';
    mark[3] = 'e';
    mark[4] = 't';
    mark[5] = 'u';
    mark[6] = 'r';
    mark[7] = 'n';
    mark[8] = '\0';
  }
  else {
    if (adjPointer == 1) mark[7] = 'v';
    if (adjPointer == 2) mark[6] = 'v';
    if (adjPointer == 3) mark[5] = 'v';
    if (adjPointer == 4) mark[4] = 'v';
    if (adjPointer == 5) mark[3] = 'v';
    if (adjPointer == 6) mark[2] = 'v';
  }

  mark[13] = '\0';
  myScreen.text(22, 35, mark);
  break;

case 4: //set output levels
  myScreen.setFont(1);
```

```
    myScreen.text(12, 0,  "LEVELS");
    myScreen.setFont(0);
    myScreen.text(10, 45, " Main  -  Aux");

    for (e = 0; e < 14; e++) mark[e] = 0x20;
    if (adjPointer == 0) {
      mark[4] = 'R';
      mark[5] = 'e';
      mark[6] = 't';
      mark[7] = 'u';
      mark[8] = 'r';
      mark[9] = 'n';
      mark[10] = '\0';
    }
    else {
      if (adjPointer == 1) mark[2] = 'v';
      if (adjPointer == 2) mark[10] = 'v';
    }
    mark[11] = '\0';

    myScreen.text(12, 35, mark);
    myScreen.setCharXY(10, 70);
    for (i=0;i<5;i++) {
      myScreen.print(levels[mainLevel][i]);
    }
    myScreen.setCharXY(60, 70);
    for (i=0;i<5;i++) {
      myScreen.print(levels[auxLevel][i]);
    }
    break;

  default:
    break;
  }
  myScreen.flush();
}


void setSynth(void) {
  PLL.setFreq(frock);
}

void setPointer (void) {
  // using encoderInc set location to 'v'
  for (e = 0; e < 14; e++) mark[e] = 0x20;

  if (encoderInc == 100000000) mark[0] = '*';
  if (encoderInc == 10000000) mark[2] = '*';
  if (encoderInc == 1000000) mark[3] = '*';
  if (encoderInc == 100000) mark[4] = '*';
  if (encoderInc == 10000) mark[6] = '*';
  if (encoderInc == 1000) mark[7] = '*';
  if (encoderInc == 100) mark[8] = '*';
  if (encoderInc == 10) mark[10] = '*';
  mark[11] = '\0';
  myScreen.print(mark);

}
```

```
void setFreqString (void)
{
  f = 0; // f is set if any previous digit > 0
  noddy = frock;
  diggy = noddy/100000000;
  freq[0] = diggy | 0x30;
  noddy = noddy % 100000000;
  freq[1] = ',';
  if (diggy > 0) f = 1;
  if (f==0) {
    freq[0] = ' ';
    freq[1] = ' ';
  }
  diggy = noddy/10000000;
  freq[2] = diggy | 0x30;
  noddy = noddy % 10000000;
  if (diggy > 0) f = 1;
  if (f == 0) freq[2] = ' ';
  diggy = noddy/1000000;
  freq[3] = diggy | 0x30;
  noddy = noddy % 1000000;
  if (diggy > 0) f = 1;
  if (f == 0) freq[3] = ' ';
  diggy = noddy/100000;
  freq[4] = diggy | 0x30;
  noddy = noddy % 100000;
  freq[5] = ',';
  if (diggy > 0) f = 1;
  if (f == 0) {
    freq[4] = ' ';
    freq[5] = ' ';
  }
  diggy = noddy/10000;
  freq[6] = diggy | 0x30;
  noddy = noddy % 10000;
  if (diggy > 0) f = 1;
  if (f == 0) freq[6] = ' ';
  diggy = noddy/1000;
  freq[7] = diggy | 0x30;
  noddy = noddy % 1000;
  if (diggy > 0) f = 1;
  if (f == 0) freq[7] = ' ';
  diggy = noddy/100;
  freq[8] = diggy | 0x30;
  noddy = noddy % 100;
  freq[9] = '.'; // decimal point
  diggy = noddy/10;
  freq[10] = diggy | 0x30;
  diggy = noddy % 10;
  freq[11] = diggy | 0x30;
  freq[12] = '0';
  freq[13] = '\0';
}


void Int2(void) { //Timer interrupt
```

```
  if (secs)  --secs;
  else {
    secs = 60;
    mins=1;
    if (m) {
      ++timeout;
      if (timeout > 2) {
        adjPointer = 0;
        m = 0;
       lastEncoderVal = encoderVal+1;
      }
    }
  }
}


void dash(void)
{
  PLL.auxEnable();
  if (m==0)
    delay(DASHLEN);
  PLL.auxDisable() ;
  if (m==0)
    delay(DOTLEN) ;
}

void dit(void)
{
  PLL.auxEnable();
  if (m==0)
    delay(DOTLEN);
  PLL.auxDisable() ;
  if (m==0)
    delay(DOTLEN);
}


void sendMorse(uint8_t p) {
  while (p != 1) {
    if (p & 1)
      dash() ;
    else
      dit() ;
    p = p / 2 ;
  }
  if (m==0)
    delay(2*DOTLEN) ;
  return ;
}
```