30. Application - VFO

This is very flexible VFO, using a synthesiser and producing 3 outputs up to 150MHz, each separately tuned in steps from 10Hz to 1MHz. It is based on the Si5351 digital synthesiser. The chip is controlled over an I2C bus. The display shows the tuned frequency, clock output number and step. The control input is from a rotary encoder. Turn to set frequency, short push to change steps and long push to select the clock output.

Connections

Si5351

SDA	A4
SCL	A5
+5V	
GND	

OLED

SDA	A4
SCL	A5
+5V	
GND	

Rotary Encoder

CLK	D2
DT	D3
SW	D4
+	+5V
GND	

The sketch must handle all three clocks, 0, 1 & 2, for frequency (freq0, freq1 & freq2), the frequency step changes and the correct display. For this a variable clk = 0, 1 or 2 stores the active clock.

The libraries used are included, and the rotary encoder pins defined. Also defined is the hold time for the switch push needed to switch the active clock.

The VFO is limited to 100kHz to 150MHz. Outputs start at 1MHz, the initial step is 10kHz.

The LCD display has an I2C bus address of 0x3F (this may be different for your display...)

```
#include "si5351.h"                                     // include libraries
#include "Rotary.h"
#include "LiquidCrystal_I2C.h"

#define CLK 2                                           // rotary connections
#define DT 3
#define SW 4
#define HOLD 500                                        // long SW hold time (0.5sec)

#define CALIBRATION 5000                                // frequency CALIBRATION

#define FREQMIN 10000000                                // tune range 100k - 150 Mhz
#define FREQMAX 15000000000

Si5351 dds;                                             // create dds object
Rotary enc = Rotary(CLK, DT);                           // create enc object
LiquidCrystal_I2C lcd(0x3F, 16, 2);                     // create lcd object, I2C addr 0x3F

byte clk = 0;                                           // current CLK 0, 1, 2

uint64_t freq0 = 100000000;                             // init 1MHz (cHz)
uint64_t freq1 = 100000000;
uint64_t freq2 = 100000000;

uint64_t freqStep = 1000000;                            // init 10kHz (cHz)
```

setup()
This initialises both the lcd display and the Si5351.

```
void setup() {
  pinMode(CLK, INPUT_PULLUP);                           // encoder inputs, with pull-ups
  pinMode(DT, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);

  lcd.begin();                                          // init lcd

  dds.init(SI5351_CRYSTAL_LOAD_8PF, 0, CALIBRATION);    // set xtal capacitance, 25MHz, & CALIBRATION

  dds.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);    // output drive (~+10dBm)
  dds.drive_strength(SI5351_CLK1, SI5351_DRIVE_8MA);
  dds.drive_strength(SI5351_CLK2, SI5351_DRIVE_8MA);

  dds.set_freq(freq0, SI5351_CLK0);                     // preset freqs
  dds.set_freq(freq1, SI5351_CLK1);
  dds.set_freq(freq2, SI5351_CLK2);

  dispUpdate();
}
```

loop()

The first part of the loop deals with the push button switch. if it is held pushed for more than the HOLD time (set to 0.5sec) this switches the clock number to be adjusted and displayed.

Other wise a short push changes the frequency step for the rotation of the encoder..

```
long hold;
unsigned char result;

if (digitalRead(SW) == LOW) {                          // enc button push
  hold = millis();                                     // start hold count
  while (!digitalRead(SW));                            // wait release
  if (millis() - hold > HOLD) {                        // button hold > HOLD time
    if (clk == 2)  clk = 0;
    else clk++;
    dispUpdate();
  }
  else if (freqStep == 1000) freqStep = 100000000;     // update step
  else freqStep = freqStep / 10;                       // step down
  dispUpdate();                                        // display
}
```

The second part of the loop handles the tuning for the selected CLK output. Up or down by the step, and updates the display. It uses the switch( ) and case to select the correct output to control.

```
result = enc.process();                                  // read encoder
switch (clk) {
  case 0:
    if (result == DIR_CW && freq0 < FREQMAX) {           // freq up
      freq0 += freqStep;
    }
    if (result == DIR_CCW && freq0 > FREQMIN) {          // freq down
      freq0 -= freqStep;
    }
    dds.set_freq(freq0, SI5351_CLK0);                    // set & display
    dispUpdate();
    break;
  case 1:
    if (result == DIR_CW && freq1 < FREQMAX) {           // freq up
      freq1 += freqStep;
    }
    if (result == DIR_CCW && freq1 > FREQMIN) {          // freq down
      freq1 -= freqStep;
    }
    dds.set_freq(freq1, SI5351_CLK1);                    // set & display
    dispUpdate();
    break;
  case 2:
    if (result == DIR_CW && freq2 < FREQMAX) {           // freq up
      freq2 += freqStep;
    }
    if (result == DIR_CCW && freq2 > FREQMIN) {          // freq down
      freq2 -= freqStep;
    }
    dds.set_freq(freq2, SI5351_CLK2);                    // set & display
    dispUpdate();
    break;
}
```

Finally there is the lcd display update which has a “quirk”. The lcd library cannot handle 64 bit variables, so it cannot display using lcd.print(freq0 / 100), so each time a 64 bit number is referenced it is converted to a 32 bit uint32_t variable to print.

The display is very simple, and could perhaps include MHz, kHz notation, but is limited to16 characters...

```
void dispUpdate() {
  uint32_t freq;                                    // convert uint64_t to uint32_t for lcd.print

  lcd.clear();
  lcd.setCursor(3, 0);                              // freq col3, row0
  switch (clk) {
    case 0:
      freq = freq0 / 100;
      lcd.print(freq);
      break;
    case 1:
      freq = freq1 / 100;
      lcd.print(freq);
      break;
    case 2:
      freq = freq2 / 100;
      lcd.print(freq);
      break;
  }

  lcd.setCursor(3, 1);                              // clock
  switch (clk) {
    case 0:
      lcd.print("CLK0");
      break;
    case 1:
      lcd.print("CLK1");
      break;
    case 2:
      lcd.print("CLK2");
      break;
  }

  lcd.setCursor(12, 1);                              // step
  freq = freqStep / 100;
  lcd.print(freq);
}
```