Arduino C++

Introduction to programming

Antony Watts M0IFA

Introduction

- Using the example code from a VFO sketch
- And along the way learn mainly about
  - Arduino functions
  - Libraries
- Open Arduino IDE,
  File > Sketchbook > My_VFO-Basic

Arduino IDE

- IDE = Integrated Development Environment
- Simple, not very powerful code editor
- Full scale compiler code -> binary
- Upload to board

Understand this?

```
void setup() {
  // encoder, button pins
  pinMode(DT, INPUT_PULLUP);
  pinMode(CLK, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);

  // setup interrupts DT & CLK for tuning
  attachInterrupt(digitalPinToInterrupt(DT), tune, CHANGE);
  attachInterrupt(digitalPinToInterrupt(CLK), tune, CHANGE);

// enable interrutps
interrupts();

// init LCD
lcd.begin();

// init dds si5351 module, "0" = default 25MHz XTAL, correction
dds.init(SI5351_CRYSTAL_LOAD_8PF, 0, 18100);

// set 8mA output drive
dds.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);

// enable VFO output CLK0, disable CLK1 & 2
dds.output_enable(SI5351_CLK0, 1);
dds.output_enable(SI5351_CLK1, 0);
dds.output_enable(SI5351_CLK2, 0);
```

Parts of the code

```
#include “library_name.h”
```

```
#define constant 20
```

```
create objects
```

```
Global variables
```

```
setup() and loop()
```

```
user functions
```

Parts of the code

- An Arduino sketch is made up from 8 parts

1. **Names of libraries to “include”, classes plus**
   Generation of objects based on classes
2. “define”d constants or macro’s
3. Global variables
4. Mandatory “setup()” and “loop()” functions
5. User functions

Libraries

```
// Si5351 V2, LCD and Rotary Encoder libraries
#include "si5351.h"
#include "LiquidCrystal_I2C.h"
#include "Rotary.h"
```

- Libraries are in your “`libraries`” folder, e.g. “`Si5351`”
- Two files, “`si5351.h`” and “`si5351.cpp`” - `header and functions`
- “`#include`” adds them as part of your sketch
- libraries can be downloaded from the internet, often on GitHub

class

name your object | library defines class functions | use the class

```
// dds object
Si5351 dds;

// lcd object
LiquidCrystal_I2C lcd(LCDADDR, LCDCOLS, LCDROWS);

// rotary Encoder object
Rotary rot = Rotary(DT, CLK);
```

class

name your object

library defines class functions

use the class

```
// init dds si5351 module, "0" = default 25MHz XTAL, correction
dds.init(SI5351_CRYSTAL_LOAD_8PF, 0, 18100);

// set 8mA output drive
dds.drive_strength(SI5351_CLK0, SI5351_DRIVE_8MA);

// enable VFO output CLK0, disable CLK1 & 2
dds.output_enable(SI5351_CLK0, 1);
dds.output_enable(SI5351_CLK1, 0);
dds.output_enable(SI5351_CLK2, 0);
```

class

name your object

library defines class functions

use the class

```
// init LCD
  lcd.begin();
====================
// display freq in kHz at col c, row r, f cHz, d decimal places
void dispFreq(uint8_t c, uint8_t r, uint64_t f, uint8_t d) {
  lcd.setCursor(c, r);
  lcd.print((float)f / 100000, d); // convert to float & kHz
  lcd.print("kHz ");
}
```

class

name your object

library defines class functions

use the class

```
// rotary interrupt service, pin 2 & 3
void tune() {
  unsigned char result;

  result = rot.process();

  if (result == DIR_CW) {
    freq += step;
    fChng = true;
  }
  else if (result == DIR_CCW) {
    freq -= step;
    fChng = true;
  }
}
```

Parts of the code

- An Arduino sketch is made up from 8 parts

1. Names of libraries to “include”, classes
   Generation of objects based on classes
2. **“define”d constants or macro’s**
3. Global variables
4. Mandatory “setup()” and “loop()”
   functions
5. User functions

define

```
// chose address of 0x27 or 0x3F for LCD
//#define LCDADDR 0x27
#define LCDADDR 0x3F
#define LCDCOLS 16
#define LCDROWS 2

// rotary Encoder pins 2 & 3 (DT & CLK), step change pin 4
(SW)
#define DT 2
#define CLK 3
#define SW 4
```

```
// lcd object
LiquidCrystal_I2C lcd(LCDADDR, LCDCOLS, LCDROWS);
```

```
// encoder, button pins
pinMode(DT, INPUT_PULLUP);
pinMode(CLK, INPUT_PULLUP);
pinMode(SW, INPUT_PULLUP);
```

Parts of the code

- An Arduino sketch is made up from 8 parts

1. Names of libraries to “include”, classes
   Generation of objects based on classes
2. “define”d constants or macro’s
3. **Global variables**
4. Mandatory “setup()” and “loop()”
   functions
5. User functions

global variables

- Variables have “scope”
- That is they can be seen in different parts of the sketch
- Global variables at top, used in any function

```
// use uint64_t to go above 40MHz
volatile uint64_t freq = 710000000; // start frequency cHz
volatile uint64_t step = 1000; // init 10Hz step
volatile bool fChng;
```

- Local variables only used in the function

```
// rotary interrupt service, pin 2 & 3
void tune() {
  unsigned char result;

  result = rot.process();
```

Parts of the code

- An Arduino sketch is made up from 8 parts

1. Names of libraries to “include”, classes
   Generation of objects based on classes
2. “define”d constants or macro’s
3. Global variables
4. **Mandatory “setup()” and “loop()” functions**
5. User functions

functions()

```
value nameOfFunction(input, input, …) {

 // function does these things, and optionally

 [return this;]
 }
```

- Functions return a value
  - can be void, char, int etc
- Functions have a name - spelling convention
- Function accept inputs, copied not changed
- Functions do things, local or global variables
- Function can return values

setup() and loop()

- All sketches must have two functions
- `setup()` - runs once
- `loop()` - runs over and over

```
void setup() {
  // encoder, button pins
  pinMode(DT, INPUT_PULLUP);
  pinMode(CLK, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);

  // —— etc etc
```

```
void loop() {
  // step?
  if (button()) {
    dispStep(step, 0, 1); // update step display
  }
  // —— etc etc
```

Parts of the code

- An Arduino sketch is made up from 8 parts

1. Names of libraries to “include”, classes
   Generation of objects based on classes
2. “define”d constants or macro’s
3. Global variables
4. Mandatory “setup()” and “loop()”
   functions
5. **User functions**

user functions

- You write your own
- Good way to break down a problem

```
// Output Freq for VFO, on CLK0, f cHz
void freqOut(uint64_t f) {
  dds.set_freq(f, SI5351_CLK0);
}
```

```
// change step?
bool button() {
  if (digitalRead(SW) == LOW) { // button pressed?
    while (!digitalRead(SW)); // wait for release
    if (step == 10000000) step = 1000; // back to 10Hz
    else step = step * 10; // or increase by x10
    return true;
  }
  else {
    return false;
  }
}
```

Arduino functions

- The Arduino development system automatically includes a set of functions for you to use
- Like
  - `pinMode(pin, MODE);`
  - `digitalWrite(pin, HIGH);`

Language Reference

Structure

- setup()
- loop()

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)

Variables

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short

Functions

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Advanced I/O

- tone()
- noTone()
- shiftOut()
- shiftIn()
- pulseIn()

loop

```
void loop() {
  // step?
  if (button()) {
    dispStep(step, 0, 1); // update step display
  }

  if (fChng) {
    freqOut(freq);
    fChng = false;
    dispFreq(0, 0, freq, 2); // display FREQ xxxxxx.xx kHz col 0 row 0
  }
  delay(20);
}
```

Let’s look through this word by word

button

```
// change step?
bool button() {
  if (digitalRead(SW) == LOW) { // button pressed?
    while (!digitalRead(SW)); // wait for release
    if (step == 10000000) step = 1000; // back to 10Hz
    else step = step * 10; // or increase by x10
    return true;
  }
  else {
    return false;
  }
}
```

Let’s look through this word by word

interrupts

```
void setup() {
  // encoder, button pins
  pinMode(DT, INPUT_PULLUP);
  pinMode(CLK, INPUT_PULLUP);
  pinMode(SW, INPUT_PULLUP);

  // setup interrupts DT & CLK for tuning
  attachInterrupt(digitalPinToInterrupt(DT), tune, CHANGE);
  attachInterrupt(digitalPinToInterrupt(CLK), tune, CHANGE);
```

- Arduino Uno has two interrupt pins (2 & 3) that generate interrupts logically named 0 and 1
- “define” values, **`DT = 2, CLK = 3`** is int 0 & int 1
- “**`tune`**” is function to call if interrupt occurs
- Interrupt occurs when DT or CLK “change” 0/1 or 1/0 That is when you turn the encoder!