Arduino For Amateur Radio

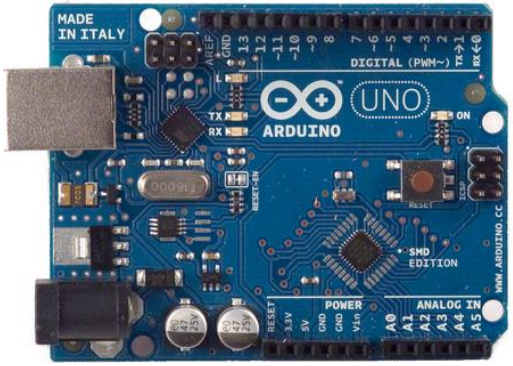
By Julie VK3FOWL and Joe VK3YSP

- Introduction to Arduino
- Arduino Models, Books and Add-Ons
- Amateur Radio Applications
- Arduino Pro Micro
- Introduction to Arduino Programming
- More on Arduino Programming
- Workshop

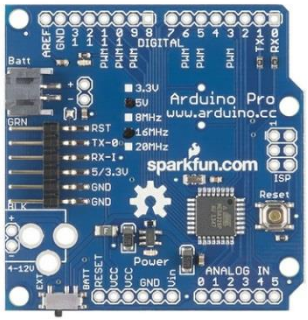
Introduction to Arduino

- Atmel 8, 16 or 32-bit microcontrollers
- Inexpensive: $3 Pro Mini - $6 Pro Micro
- Cross platform: Windows/Linux/MacOSX
- Open Source Hardware and Software
- Free Integrated Development Environment (IDE)
- USB programmer built in
- C/C++ Compatible (almost)
- I/O: Digital, Analog, I2C, SPI, USB, TTL Serial

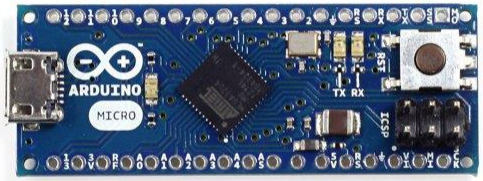
Arduino Models



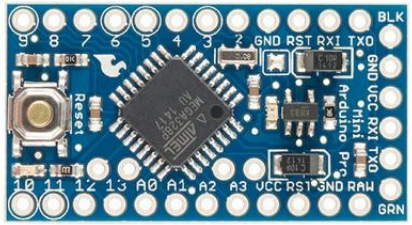
UNO



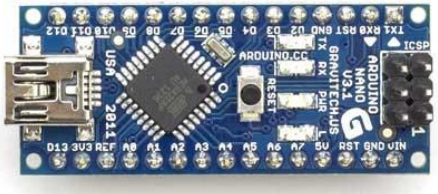
PRO



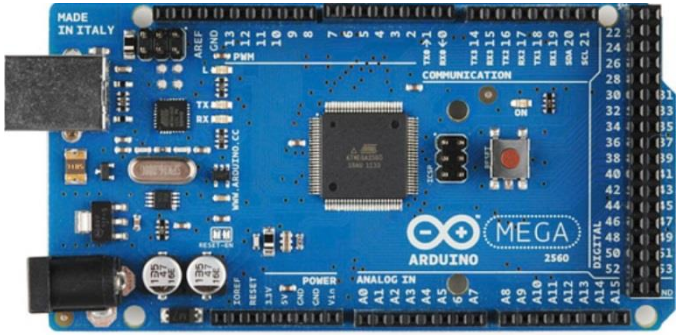
MICRO



PRO MINI



NANO



MEGA

Arduino Books

...PLUS MANY MORE

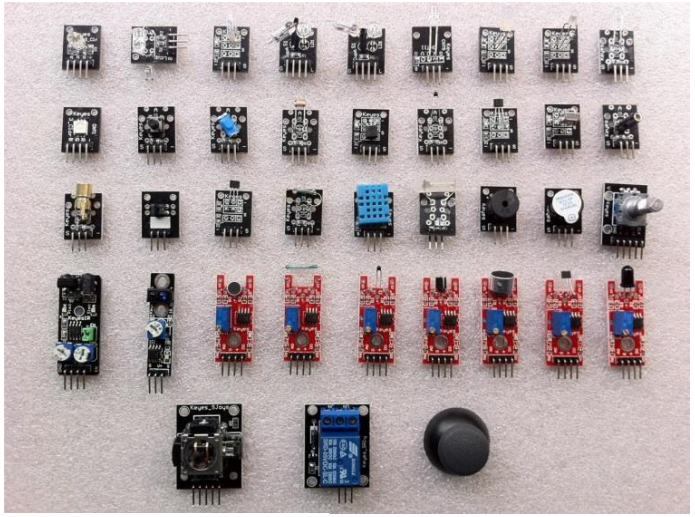
Arduino Add-Ons

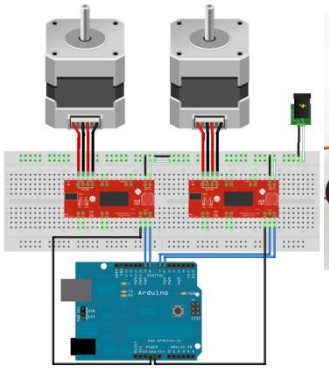
Sensors
- GPS Receiver
- Accelerometer
- Magnetometer
- Barometer/Altimeter
- Anemometer/Airspeed
- Rate Gyroscope
- Thermocouple/IR/PIR
- Vibration/Tilt
- Hygrometer/Humidity
- Acoustic/Ultrasonic
- Force/Flex/Weight/Strain
- Gas/Water/Light/Colour/UV
- Fluid Level/Fluid Flow/pH
- Proximity/Line/Touch
- Pulse/Heartbeat/BP
- ECG/Brainwave
- Ionizing Radiation
- Switch/Pot/Encoder
- Keyboard/Joystick/Touchpad
- Reed/Hall Effect Switch
- CAM/LIDAR/FLIR
- Gesture/Fingerprint
- Hydrophone/Geophone
- Barcode/RFID reader
- Mag/Memory card reader
- Coin Acceptor
- Voltage/Current/Power
- Analog to Digital Converter

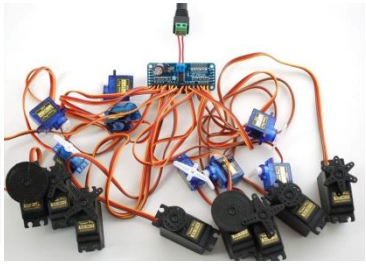
Drivers
- Linear/rotary actuators
- Stepper motor/Servomotor
- AC/DC Motor/Relay/Solenoid
- Solenoid Valve
- Buzzer/Bell/Siren
- Digital to Analog Converter

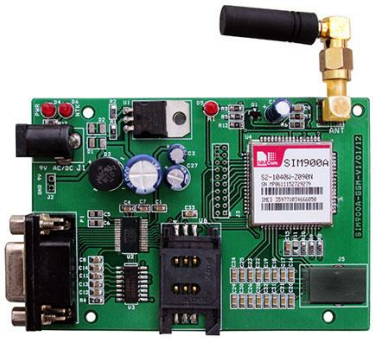
Displays
- Monochrome/Colour
- Text/graphics/LED/LCD/OLED

RF/Data Link/Remote Control
- DDS VFO
- WiFi/MANET
- GSM/3G/4G
- IR/RF Remote





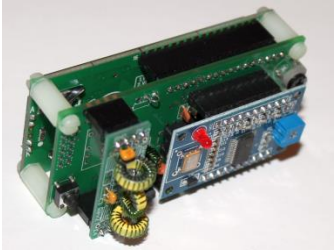




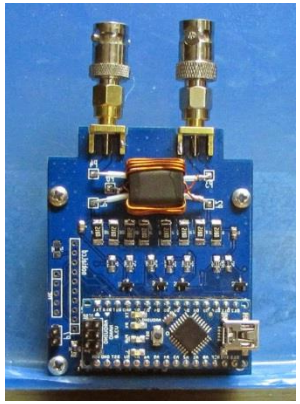
Amateur Radio Applications

- DDS VFO Controller

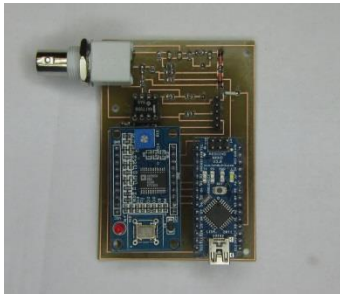
- WSPR/QRSS/APRS beacon



- Digital SWR/Power meter

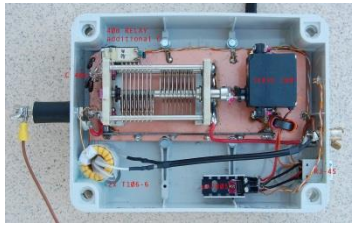


- Digital Antenna/Network Analyser

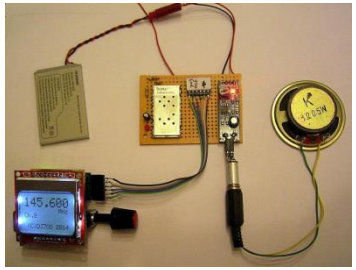


Amateur Radio Applications

- Automatic Antenna Tuner

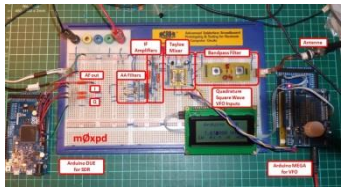


- Transceiver module controller



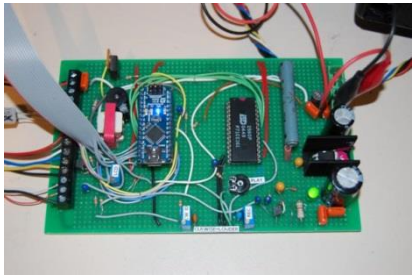
- Antenna rotator controller

- SDR Controller

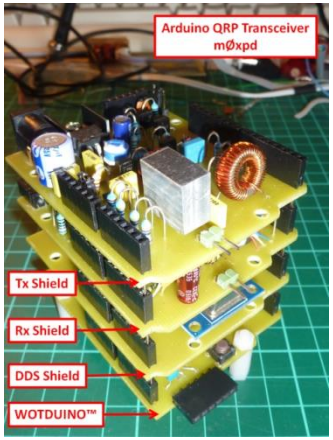


Amateur Radio Applications

- Repeater controller



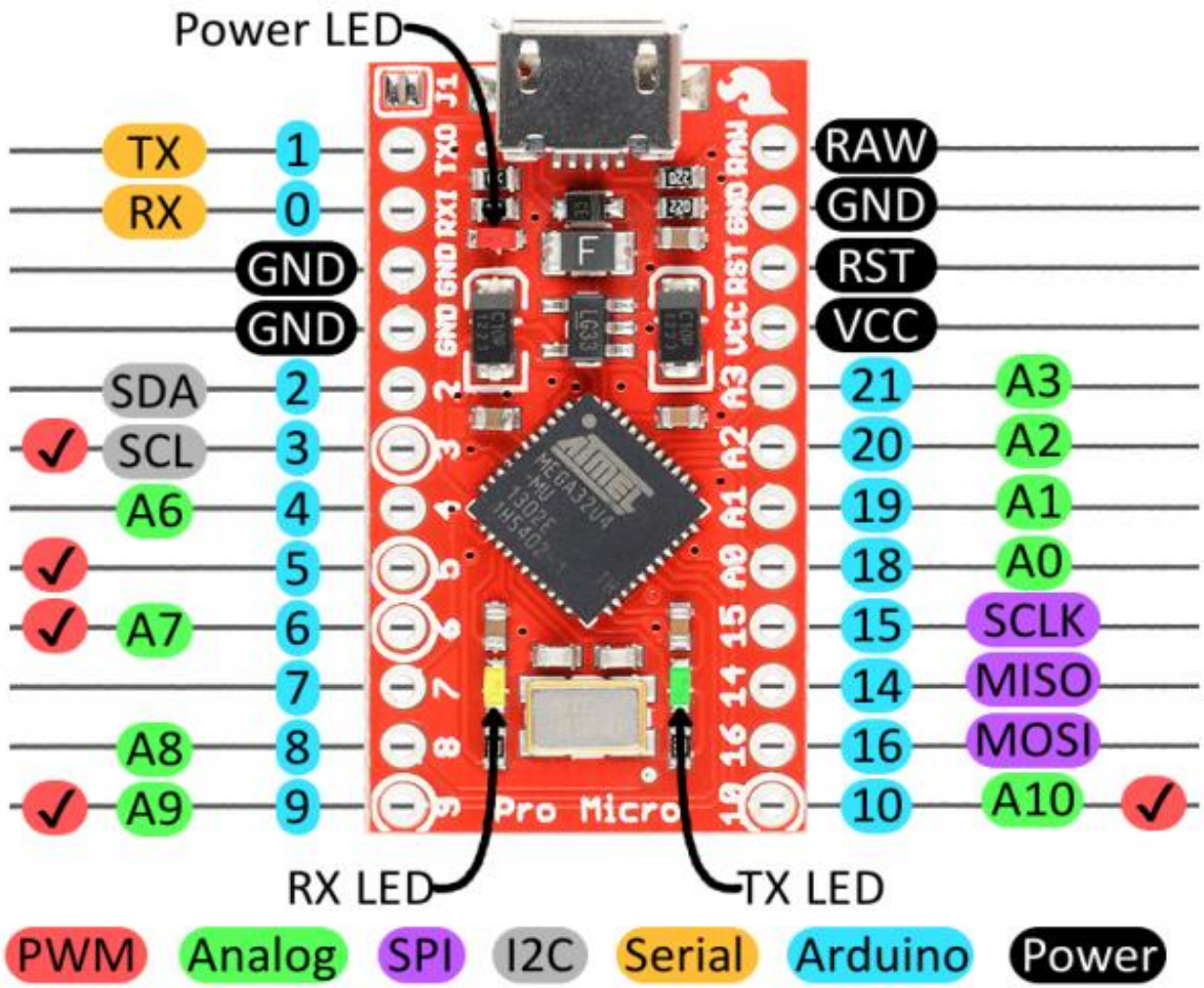
- QRP Transceiver



- Morse keyer/encoder/decoder

- APRS Display

Arduino Pro Micro



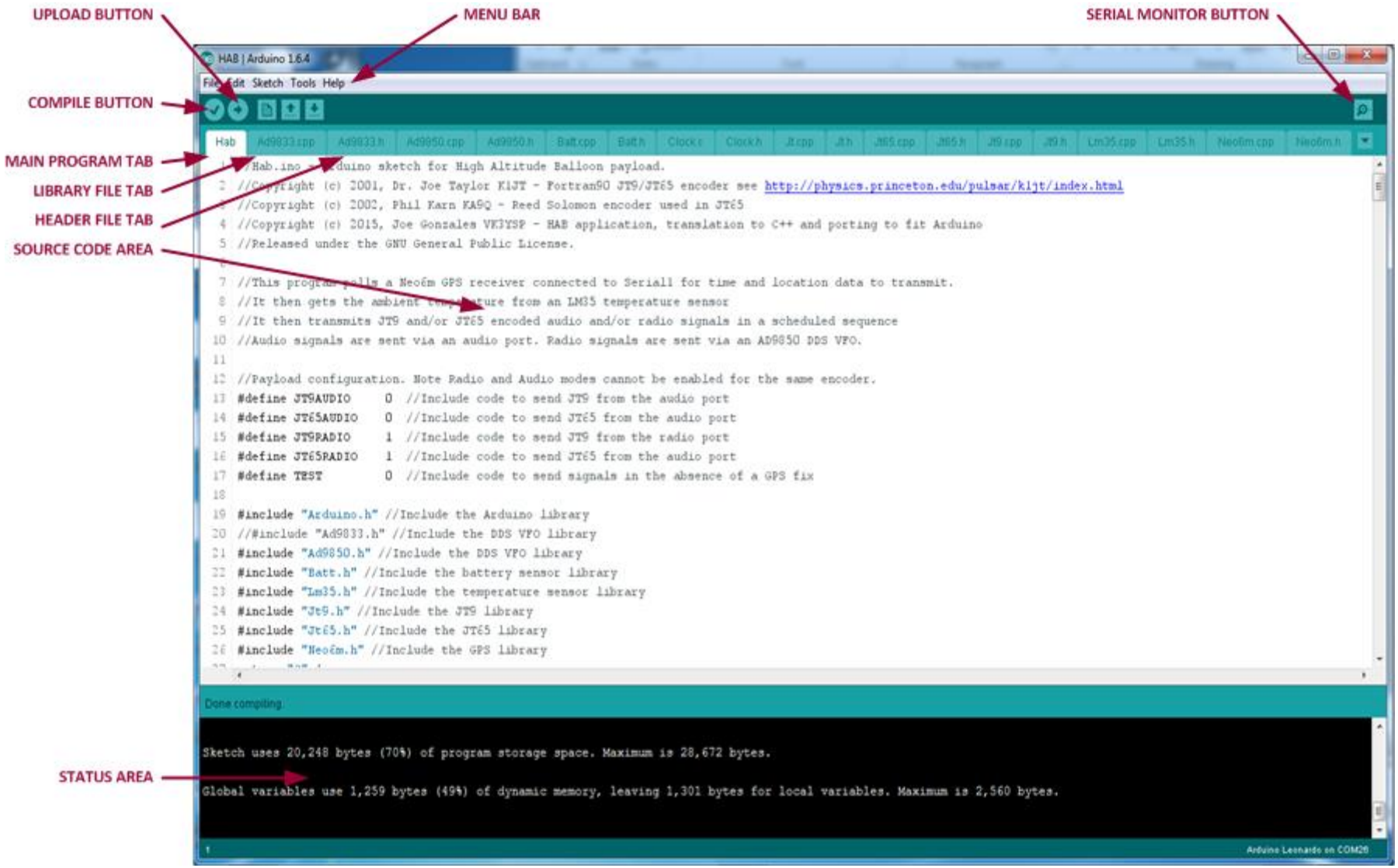
Arduino Pro Micro

- High Performance, Atmel ATmega32U4, 16MHz, 8-Bit Microcontroller
- Advanced RISC Architecture, Low Power AVR®
- 5V–12V External Power or USB Power.
- 32KB Program memory, 2.5KB SRAM, 1KB EEPROM
- One 8-bit, two 16-bit and one 10-bit Counter/Timer/Prescaler/Capture/Compare
- USB 2.0 port for programing/serial monitor (No switch required for programming)
- Eighteen 5V Input/Output Ports for Digital TTL I/O
- Five 8-bit PWM Ports for PWM or Analog Output
- Nine 10-bit ADC Port for Analog Input with 2.5V or 5V reference
- Programmable USART for Serial TTL I/O
- 3-wire, SPI Serial Perpheral Interface. 2-wire, I2C Inter-Integrated Circuit Interface.
- Interrupts with wake-up on pin change.

Introduction to Arduino programming

- Arduino Integrated Development Environment (IDE)
- Arduino Program Development
- Comments
- Statements
- Data Types
- Constants
- Variables
- Functions
- Programs
- Library Functions
- Program Design
- Program Code

Arduino IDE


UPLOAD BUTTON
MENU BAR
SERIAL MONITOR BUTTON
COMPILE BUTTON
MAIN PROGRAM TAB
LIBRARY FILE TAB
HEADER FILE TAB
SOURCE CODE AREA
STATUS AREA
HAB | Arduino 1.6.4
File Edit Sketch Tools Help
Hab
Ad9833.cpp
Ad9833.h
Ad9850.cpp
Ad9850.h
Batt.cpp
Batt.h
Clock.c
Clock.h
Jt.cpp
Jt.h
Jt65.cpp
Jt65.h
Jt9.cpp
Jt9.h
Lm35.cpp
Lm35.h
Neo6m.cpp
Neo6m.h
//Hab.ino - Arduino sketch for High Altitude Balloon payload.
//Copyright (c) 2001, Dr. Joe Taylor K1JT - Fortran90 JT9/JT65 encoder see http://physics.princeton.edu/pulsar/k1jt/index.html
//Copyright (c) 2002, Phil Karn KA9Q - Reed Solomon encoder used in JT65
//Copyright (c) 2015, Joe Gonzales VK3YSP - HAB application, translation to C++ and porting to fit Arduino
//Released under the GNU General Public License.
//This program polls a Neo6m GPS receiver connected to Serial1 for time and location data to transmit.
//It then gets the ambient temperature from an LM35 temperature sensor
//It then transmits JT9 and/or JT65 encoded audio and/or radio signals in a scheduled sequence
//Audio signals are sent via an audio port. Radio signals are sent via an AD9850 DDS VFO.
//Payload configuration. Note Radio and Audio modes cannot be enabled for the same encoder.
#define JT9AUDIO 0 //Include code to send JT9 from the audio port
#define JT65AUDIO 0 //Include code to send JT65 from the audio port
#define JT9RADIO 1 //Include code to send JT9 from the radio port
#define JT65RADIO 1 //Include code to send JT65 from the audio port
#define TEST 0 //Include code to send signals in the absence of a GPS fix
#include "Arduino.h" //Include the Arduino library
//#include "Ad9833.h" //Include the DDS VFO library
#include "Ad9850.h" //Include the DDS VFO library
#include "Batt.h" //Include the battery sensor library
#include "Lm35.h" //Include the temperature sensor library
#include "Jt9.h" //Include the JT9 library
#include "Jt65.h" //Include the JT65 library
#include "Neo6m.h" //Include the GPS library
Done compiling.
Sketch uses 20,248 bytes (70%) of program storage space. Maximum is 28,672 bytes.
Global variables use 1,259 bytes (49%) of dynamic memory, leaving 1,301 bytes for local variables. Maximum is 2,560 bytes.
Arduino Leonardo on COM26

Arduino Program Development

- Plug the Arduino into the computer USB port
- Start up the IDE application
- Load an existing source code (.ino/.c/.cpp/.h) file; or
- Type in a new source code file and save it
- Compile the source code to check it
- Fix any errors and re-compile it if necessary
- Upload the executable code to the Arduino
- Watch the Arduino reset and start running immediately
- Start the Serial Monitor for any serial I/O
- Unplug the Arduino and it will run stand-alone on power up

Comments

- Comments are used to provide explanatory notes inline with the source code. They are ignored by the compiler and are not executed by the processor.

```
//This is a single line comment. It is not executed.

/*
These are comments over several lines.
They are not executed either.
*/
```

Statements

- Statements are executable instructions.
- Statements are executed in order.
- Each statement finishes with a semicolon.
- Multiple statements executed as a group are enclosed in parentheses {}

```
{
        a = 1;          //Statement 1
        a = 2;          //Statement 2
        b = a + 2;      //Statement 3
}
```

Data Types

- Data types identify the format of information used in statements.
- Common data types:
  - char (character): A,a,B,b,C,c… !,@,#… 1,2,3…
  - String (strings of characters): “Hello”
  - int (integers): …-1,-2,0,1,2…
  - byte (8-bit bytes): 0..255
  - float (floating point numbers): 1.234
  - bool (boolean): true/false
  - void (nothing):
- Information with different data types is typically not compatible, but can sometimes be converted.
- Data types are coloured blue in the IDE.

Constants

- Constants are pre-defined words used instead of numbers in statements.
- They represent permanent values using more meaningful names.
- Constants are used to refer to Arduino pins
  - A0 refers to pin 18. SCLK refers to pin 15.
- Constants are used in Arduino functions
  - HIGH refers to 1 and LOW refers to 0
  - INPUT and OUTPUT refers to different pin modes

Variables

- Variables are used to store transient values.
- They are <u>declared</u> by name with their data type:

```
int cup1; float cup2;
```

- Variables can then be <u>assigned</u> values:

```
cup1 = 2; cup2 = 1.5;
```

- Variables can be <u>initialised</u> (once):

```
int cup1 = 2; float cup2 = 1.5;
```

- Variables can be <u>operated</u> upon:

```
cup1 = cup1 + 2;
cup1++; //same as cup1=cup1+1;
cup1+=1; //same as cup1=cup1+1;
```

Functions

- Functions define a set of frequently executed statements.
- Values can be passed to functions. Functions can return a value.
- Functions can be declared with different parameter and return data types.

```
float julie(int cup1, float cup2) {
  float cup3 = cup1 + cup2;    //Note: Here cup1, cup2 and cup3 are “local” variables
  return cup3;
}
```

- Functions can be called by passing parameters and assigning return values.

```
float cup3 = julie(2, 1.5); //Note: This cup3 is different. How about: float cup3 = julie(1.5, 2);?
```

- Functions can also be declared with no parameters and no return value.

```
void julie() {
//Statements
}
```

- These functions can be called as follows:

```
julie();
```

Programs

- All Arduino “sketches” are essentially C/C++ programs, but instead of the normal ANSI C main function, int main(), they must have two declared functions void setup() and void loop(). Each function takes no parameters and returns no result.

```
//This setup function is called once automatically on Arduino startup
void setup() {
  // put your setup code here, to run once:

}

//This loop function is called repeatedly after setup has completed
void loop() {
  // put your main code here, to run repeatedly:

}
```

Library Functions

- The Arduino IDE provides many useful pre-defined library functions including:

  Digital I/O, Analog I/O, Serial I/O, Advanced I/O, Time, Maths, Trig, Random numbers, Interrupts and USB emulation.

- The IDE provides a web-based reference: Select Help | Reference to browse it.
- Library functions are orange in the IDE.

More on Arduino programming

- Declaring more complex data types
- Compiler directives
  - #define, #include, #ifdef
- Control structures:
  - if/then/else/while/switch
- Operators:
  - comparison/arithmetic/boolean
  - bitwise/pointer/compound
- Interrupts
- More library functions
- Object oriented programming (C++)

Program 1 Design

This program will flash a LED on and off.

The LED is connected between Pin 2 (+) and GND (-).

The setup function initializes pin 2 as a digital output pin and sets it LOW (so the LED will be off).

The loop function will repeatedly:

- Set Pin 2 HIGH (the LED will go on) then wait 100ms.
- Set Pin 2 LOW (the LED will go off) then wait 100ms.

Note: A series current-limiting resister is required for the LED. R = (5 - Vf) / If

Library Function: pinMode

void pinMode(int pin, int mode);

Configures the specified pin to behave either as an input or an output.

- pin: The Arduino pin number
- mode: INPUT, OUTPUT or INPUT_PULLUP

Note: The pull-up mode enables internal pull-up resistors (20kΩ - 50kΩ) to +5Vdc on the specified pin.

Example:

```
pinMode(2, OUTPUT);    //Initialize pin 2 as an output
```

Library Function: digitalWrite

void digitalWrite(int pin, int value);

Set the voltage on an output pin to the specified value. HIGH = 5V, LOW = 0V.

- pin: The Arduino pin number
- value: HIGH, LOW

Examples:

```
digitalWrite(2, HIGH);  //Turn the LED on.
digitalWrite(2, LOW);   //Turn the LED off.
```

Library Function: delay

```
void delay(int ms);
```

Suspends program execution for the specified number of milliseconds.

- ms: The delay time in milliseconds

Example:

```
delay(100);        //Wait for 100ms
```

Program 1 Code

```
//This program flashes a LED on and off.
//Hardware: Arduino Pro Micro. LED with resistor connected between pin2 and GND
//This program is free. It is released under the GNU General Public License.

//Setup the Arduino.
void setup() {
  pinMode(2, OUTPUT);          //Initialize digital pin 2 as an output pin.
  digitalWrite(2, LOW);        //Turn the LED off.
}

//Flash the LED.
void loop() {
  digitalWrite(2, HIGH);       //Turn the LED on.
  delay(100);                  //Wait for 100ms.
  digitalWrite(2, LOW);        //Turn the LED off.
  delay(100);                  //Wait for 100ms.
}
```

Arduino Workshop

- IDE installation
- Discussion, Questions and Answers
- Program 1 Development and Test
- Program 2 Design and Coding
- Program 2 Development and Test
- Amateur Radio Application 1
- Amateur Radio Application 2

IDE Installation

- Install the IDE application on your laptop.
- Connect the Arduino to your laptop USB port.
- Start the IDE application and test the compiler.
- Test program upload and execution.
- Program 1: Blink a LED.
- Program 2: Measure the ambient light level.

Program 2 Design

Reusing the code from the previous example, this program will measure the voltage on an analog pin and display it using the IDE serial monitor. It will then flash the LED at a rate proportional to the ambient light level.

A light dependent resistor (2kΩ~50kΩ) is connected between pin A0 and GND to provide a dynamic voltage input.

The setup function will initialize pin A0 as an analog input pin and enable an internal pull-up resistor (20kΩ - 50kΩ).

The loop function will repeatedly:

Turn the LED on for 100ms.

Read the integer value of the 10-bit analog to digital converter (0 – 1023)

Compute the equivalent voltage = 5 * (value / 1024.0)

Send the voltage information to the serial port so it can be displayed by the serial monitor.

Use the analog value as the off delay time for the LED.

Library Function: analogRead

```
int analogRead(int pin);
```

Reads the value of the specified analog pin.

- pin: The Arduino pin number
- returns: 0 – 1023, representing 0 – 5 volts

Example:

```
int value = analogRead(A0);
```

Library Function: Serial.begin

void Serial.begin(speed);

Sets the speed of the Arduino serial port.

- speed: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200 bps.

Example:

```
Serial.begin(9600);
```

Library Function: Serial.print

void Serial.print(val); void Serial.println(val);

Sends data to the Arduino serial port.

– val: Any integer, float, character or string value

Examples:

```
Serial.print(78);            //Prints 78
Serial.print(1.23456);       //Prints 1.23
Serial.print("N");          //Prints N
Serial.println("Hello");     //Prints Hello\r\n
```

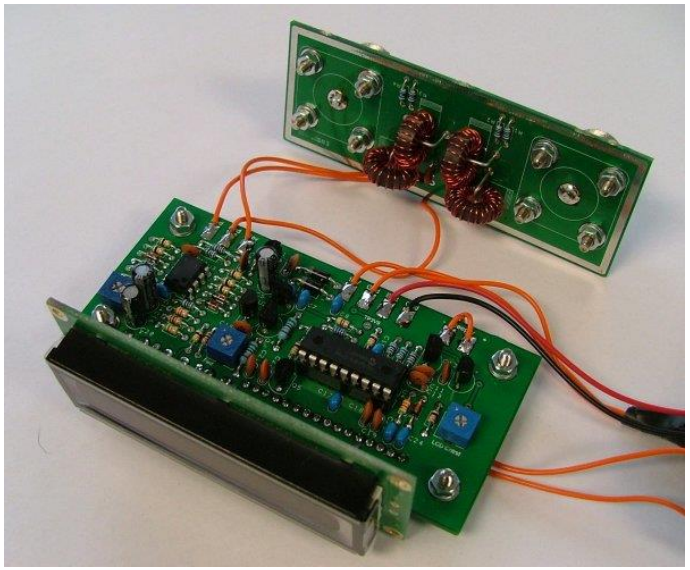
Program 2 Code

```
//This program will measure the voltage on an analog pin and display it using the IDE serial monitor.
//It will also flash the LED at a rate determined by the ambient light level
//Hardware: Arduino Pro Micro. LED with resistor connected between pin2 and GND. Photoresistor connected between pin A0 and GND.
//This program is free. It is released under the GNU General Public License.

//Setup the Arduino serial port speed and the analog input pin
void setup () {
  Serial.begin(9600);                    //Start up the serial port at a speed of 9600bps
  pinMode(A0, INPUT_PULLUP);             //Initialize analog pin A0 as an input pin with internal pullup resistor

  pinMode(2, OUTPUT);                    //Initialize digital pin 2 as an output pin.
  digitalWrite(2, LOW);                  //Turn the LED off.
}

//Repeatedly read and display the input voltage. Note: It varies with the ambient light intensity
void loop () {
  int value = analogRead(A0);            //Read the raw analog value from the A0 pin
  float voltage = 5 * value / 1024.0;    //Compute the actual voltage
  Serial.print("The voltage is: ");      //Print the heading over the serial port
  Serial.println(voltage);               //Print the voltage and a new line

  digitalWrite(2, HIGH);                 //Turn the LED on.
  delay(100);                            //Wait for one hundred milliseconds
  digitalWrite(2, LOW);                  //Turn the LED off.
  delay(value);                          //Wait for a variable amount of time determined by the ambient light level.
}
```

Amateur Radio Application 1

Simple Project: A High SWR Alarm Unit

Something like this, maybe, but we don’t have time for a fancy display. We just want an alarm that goes off every time the antenna falls down!

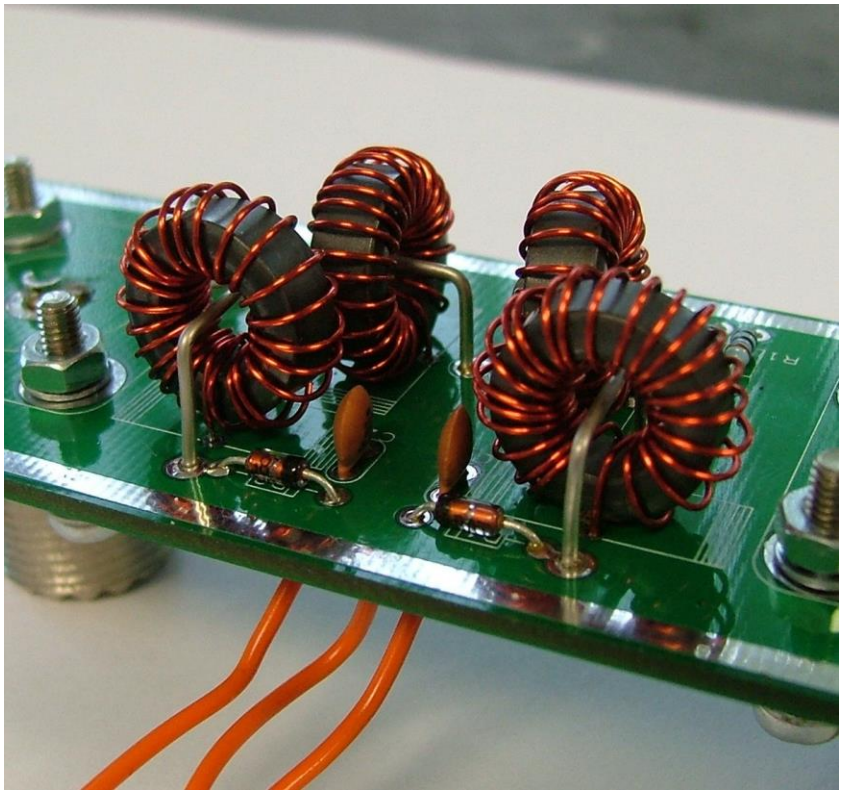


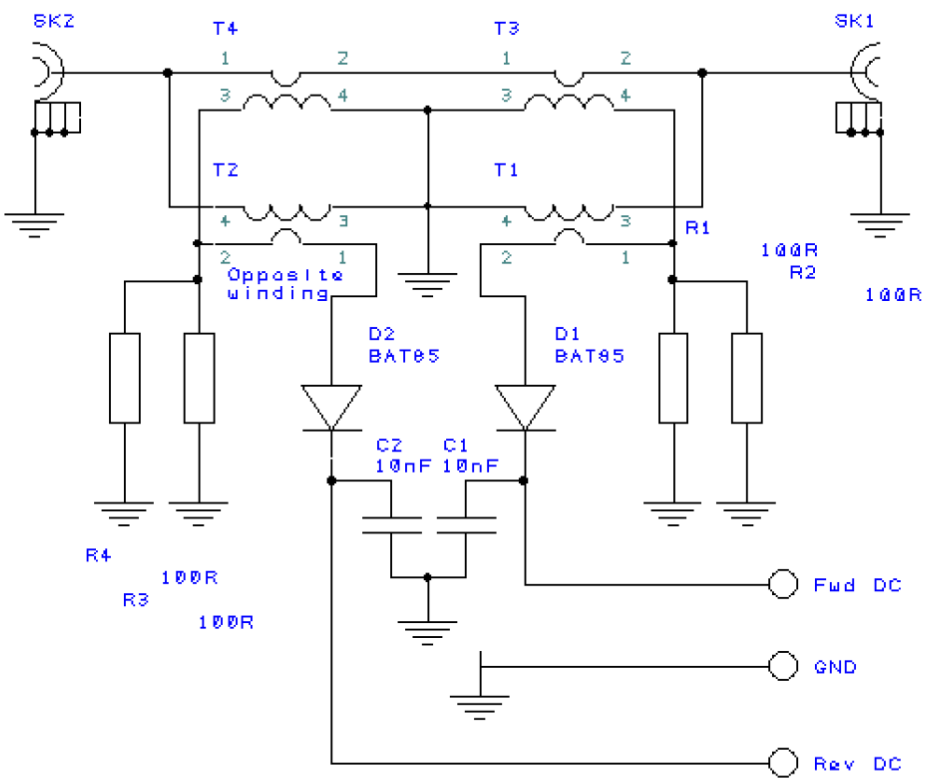
System Requirements

- The system shall sound an alarm whenever the SWR is too high
  - It shall monitor the forward and reverse voltages from an external SWR bridge
  - It shall compute the SWR at least every second
  - It shall sound and alarm while the SWR is greater than 1.5:1
  - It shall be powered by a 9 volt battery

The SWR bridge

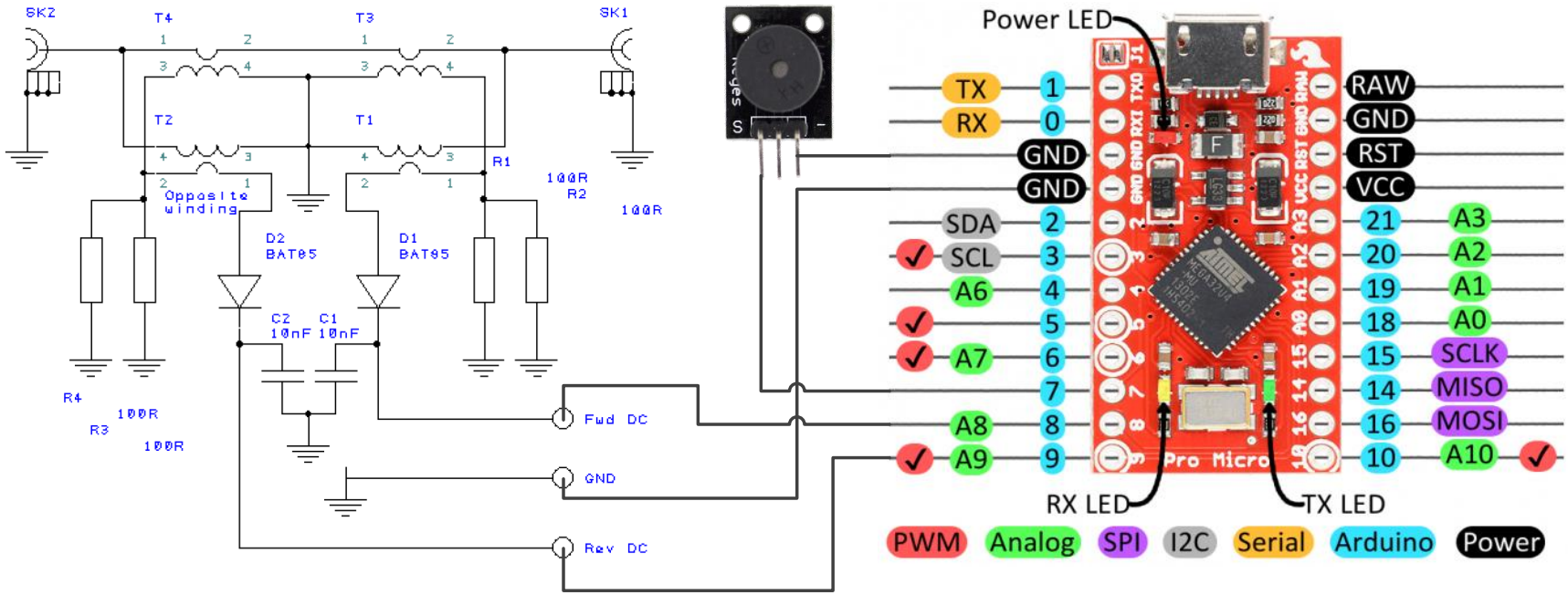
c/o http://www.radio-kits.co.uk/swr_meter/





Designing the Hardware

- Assume the Fwd DC and Rev DC voltage is <u>linear</u> with respect to power and is no more than <u>5 volts</u>.
- Connect Fwd DC and Rev DC to <u>analog input</u> pins A8 and A9, a 5V alarm to <u>digitial I/O</u> pin 7 and 9V to RAW. GND everything.



Testing the Buzzer

```
void setup() {
  pinMode(7,OUTPUT);          //Setup the alarm pin as an output
}

void loop() {
  digitalWrite(7,HIGH);       //Turn on
  delay(100);                 //Wait for a period
  digitalWrite(7,LOW);        //Turn off
  delay(100);                 //Wait for the same period
}
```

Refactoring the Code

```
void setup() {
  pinMode(7,OUTPUT);
}

void loop() {
  digitalWrite(7,HIGH);
 delay(100);
  digitalWrite(7,LOW);
  delay(100);
}
```

```
#define ALARMPIN 7 //Define the alarm pin
#define BEEPTIME 100 //Define the beep time

void beep(int alarmPin, int alarmTime) {
  //This function beeps the alarm just once
  digitalWrite(alarmPin,HIGH);          //Turn on
  delay(alarmTime);                     //Wait for a period
  digitalWrite(alarmPin,LOW);           //Turn off
  delay(alarmTime);                     //Wait for the same period
}

void setup() {
pinMode(ALARMPIN,OUTPUT); //Setup the alarm pin as an output
}

void loop() {
  // Beep the alarm continuously
  beep(ALARMPIN,BEEPTIME);
}
```

Testing the Analog Conversion

```
void setup() {
  pinMode(8,INPUT);
  Serial.begin(9600);
}

void loop() {
 int Vf = analogRead(8);
 int Vr = analogRead(9);
 Serial.print(Vf);
 Serial.print(‘,’);
 Serial.println(Vr);
 delay(100);
}
```

Getting ready to compute the SWR

```
#define ALARMPIN 7        //Define the alarm pin
#define BEEPTIME 100      //Define the beep time in milliseconds
#define SWRLIMIT 1.5      //Define the SWR limit
#define FWDPIN A8         //Define the FWD pin
#define REVPIN A9         //Define the REV pin

float computeSWR(int fwdPin, int revPin) {
  //This function reads and computes the SWR
  intVf = analogRead(fwdPin);  //Read the forward voltage
  int Vr = analogRead(revPin);  //Read the reverse voltage
  float swr = 1;            //Calculate the SWR
  Serial.print(Vf);         //Print the Vf
  Serial.print(',');
  Serial.print(Vr);         //Print the Vr
  Serial.print(',');
  Serial.println(swr);      //Print the SWR
  return swr;               //Return the SWR
}
```

Computing the SWR

$$SWR = \frac{V_f + V_r}{V_f - V_r}$$

```
float computeSWR(int fwdPin,int revPin) {
  float Vf = analogRead(fwdPin);
  float Vr = analogRead(revPin);
  float swr = (Vf + Vr)/(Vf - Vr);
  return swr;
}
```

Debugging

```
#define DEBUG        //Uncomment this line for debug mode

float computeSWR(int fwdPin, int revPin) {
  //This function reads and computes the SWR
  float Vf = analogRead(fwdPin);  //Read the forward voltage
  float Vr = analogRead(revPin);  //Read the reverse voltage
  float swr = (Vf + Vr) / (Vf - Vr); //Calculate the SWR
#ifdef DEBUG
  Serial.print(Vf);             //Print the Vf in debug mode
  Serial.print(',');
  Serial.print(Vr);             //Print the Vr in debug mode
  Serial.print(',');
  Serial.println(swr);          //Print the SWR in debug mode
#endif
  return swr;                   //Return the SWR
}
```

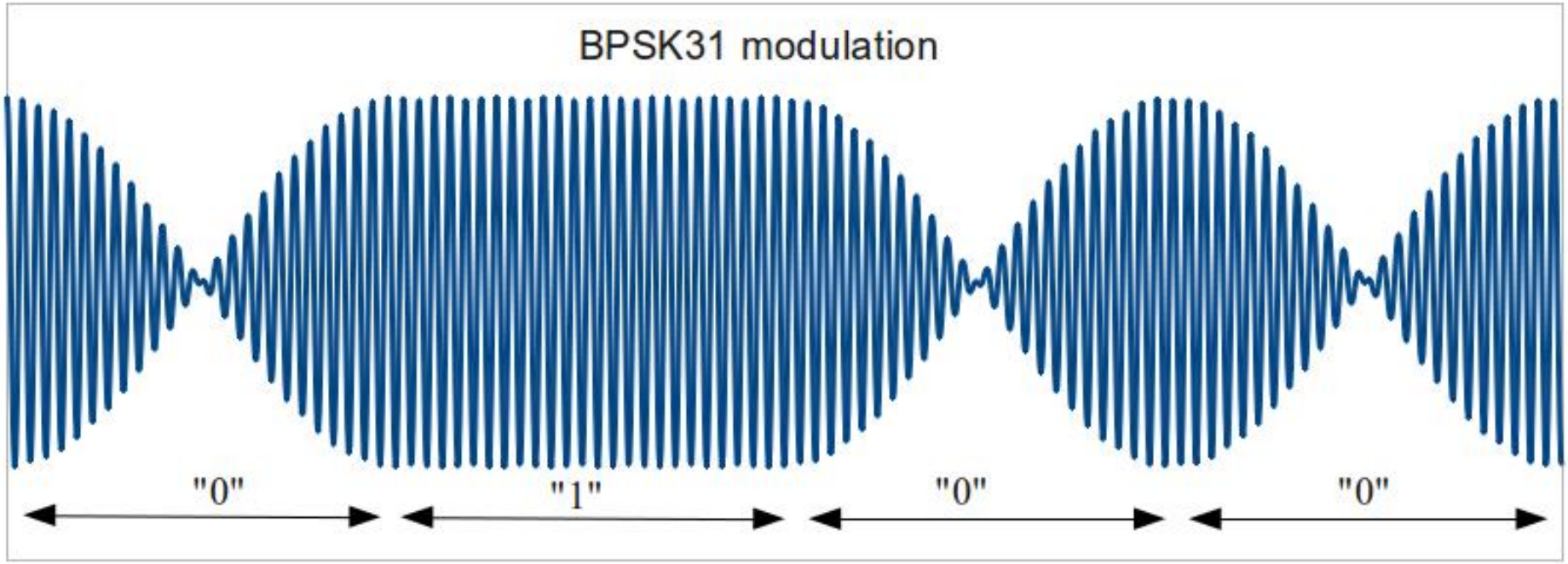
What can possibly go wrong?

- The SWR is undefined when Vf = Vr
- The SWR is negative when Vf > Vr
- The readings are not taken at regular intervals
- The readings are neither peak nor average
- It may beep when you start transmitting
- There may be DC offsets and non-linearities
- There may be RFI to the electronics

Amateur Radio Application 2

Complex Project: BPSK31 Encoder

A BPSK31 Encoder first encodes text using Varicode then sends it at 31.25baud (symbols per second). This application just looks at how to generate the waveform using an Arduino.



Generating a waveform

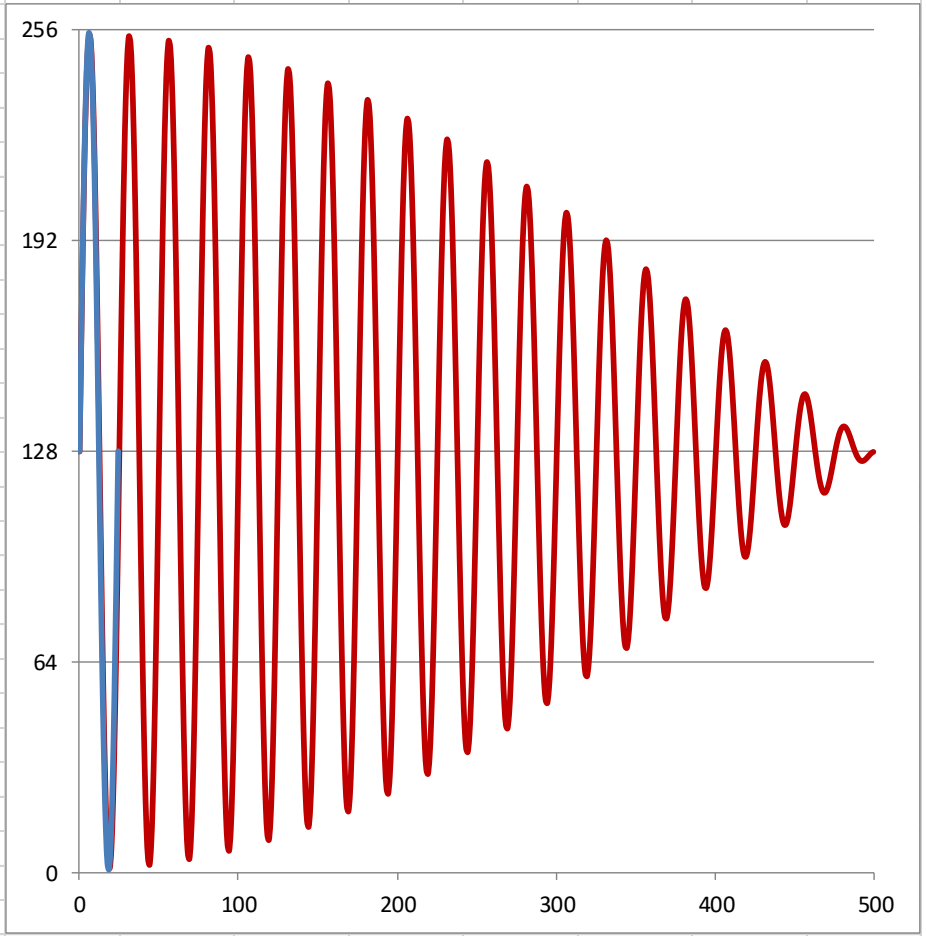
- Create a lookup table of waveform values
- Set a hardware timer to generate a phase-correct PWM square wave at the waveform sample rate.
- Use the timer's overflow Interrupt Service Routine to load a new waveform value into the PWM generator at the end of each PWM cycle.
- Use a Band Pass Filter to filter the PWM output.

Hint:

- Use a spreadsheet to create the waveform values
- The BPSK31 waveform is a cosine-modulated sinewave:

```
=COS(2*PI()*A12/$B$4/$B$5/2)*SIN(2*PI()*MOD(A12,$B$4)/$B$4)
```

Clock Frequency (Hz)	16000000	16000000	16000000	16000000		
Clock Divisor	512	512	512	512		
Samples per second	31250	31250	31250	31250		
Samples per cycle	25	50	40	20		
Cycles per symbol	40	20	25	50		
Samples per symbol	1000	1000	1000	1000		
Cycles per second	1250	625	781.25	1562.5		
Seconds per symbol	0.032	0.032	0.032	0.032		
Symbols per second	31.25	31.25	31.25	31.25		
Degrees to radians	0.017					
Sample	**Float**	**Byte**	**Hex**	**Float**	**Byte**	**Hex**
0	0.000	128	80	0.000	128	80
1	0.249	160	A0	0.249	160	A0
2	0.482	189	BD	0.482	189	BD
3	0.685	215	D7	0.685	215	D7
4	0.844	235	EB	0.844	235	EB
5	0.951	249	F9	0.951	249	F9
6	0.998	255	FF	0.998	255	FF
7	0.982	253	FD	0.982	253	FD
8	0.905	243	F3	0.905	243	F3
9	0.770	226	E2	0.771	226	E2
10	0.587	203	CB	0.588	203	CB
11	0.368	175	AF	0.368	175	AF
12	0.125	144	90	0.125	144	90
13	-0.125	112	70	-0.125	112	70
14	-0.368	81	51	-0.368	81	51
15	-0.587	53	35	-0.588	53	35
16	-0.770	30	1E	-0.771	30	1E
17	-0.904	13	0D	-0.905	13	0D
18	-0.981	3	03	-0.982	3	03
19	-0.996	1	01	-0.998	1	01
20	-0.949	7	07	-0.951	7	07
21	-0.842	21	15	-0.844	21	15
22	-0.683	41	29	-0.685	41	29
23	-0.480	67	43	-0.482	67	43
24	-0.248	97	61	-0.249	96	60
25	0.000	128	80	0.000	128	80



```
//This table is a template for the bpsk31 waveform. It is 20 cycles comprising 25 samples each.
//It is an 8-bit amplitude modulated sine wave.
//The envelope of the sine wave is a cosine shape from 0 to 90 degrees. Starting at 100% and finishing at 0%
//It represents only the first half of the zero symbol waveform. The second half is a mirror image.
//The first cycle (25 samples) of this waveform is also used as the template for the full-amplitude one symbol.
//The maximum diference between the full-amplitude wave and the cosine-modulated wave is only one lsb of the last entry on the first line.
//The phase of this waveform can be shifted 180 degrees by multiplying each value by -1.

volatile const char bpsk31Wave1250Hz[]  = {
  0x80, 0xA0, 0xBD, 0xD7, 0xEB, 0xF9, 0xFF, 0xFD, 0xF3, 0xE2, 0xCB, 0xAF, 0x90, 0x70, 0x51, 0x35, 0x1E, 0x0D, 0x03, 0x01, 0x07, 0x15, 0x29, 0x43, 0x60,
  0x80, 0x9F, 0xBD, 0xD7, 0xEB, 0xF8, 0xFE, 0xFC, 0xF2, 0xE1, 0xCA, 0xAE, 0x90, 0x70, 0x52, 0x36, 0x1F, 0x0E, 0x04, 0x02, 0x08, 0x16, 0x2A, 0x44, 0x61,
  0x80, 0x9F, 0xBC, 0xD6, 0xEA, 0xF7, 0xFD, 0xFB, 0xF1, 0xE0, 0xC9, 0xAE, 0x90, 0x70, 0x52, 0x37, 0x20, 0x10, 0x06, 0x04, 0x0A, 0x17, 0x2B, 0x44, 0x61,
  0x80, 0x9F, 0xBB, 0xD4, 0xE8, 0xF5, 0xFB, 0xF9, 0xEF, 0xDE, 0xC8, 0xAD, 0x8F, 0x71, 0x53, 0x38, 0x22, 0x12, 0x09, 0x07, 0x0D, 0x1A, 0x2D, 0x46, 0x62,
  0x80, 0x9E, 0xBA, 0xD2, 0xE6, 0xF2, 0xF8, 0xF6, 0xEC, 0xDC, 0xC6, 0xAC, 0x8F, 0x71, 0x54, 0x3A, 0x25, 0x15, 0x0C, 0x0A, 0x10, 0x1C, 0x2F, 0x47, 0x63,
  0x80, 0x9D, 0xB8, 0xD0, 0xE3, 0xEF, 0xF4, 0xF2, 0xE9, 0xD9, 0xC4, 0xAB, 0x8E, 0x72, 0x56, 0x3C, 0x28, 0x18, 0x10, 0x0E, 0x14, 0x20, 0x32, 0x49, 0x64,
  0x80, 0x9C, 0xB6, 0xCD, 0xDF, 0xEB, 0xF0, 0xEE, 0xE5, 0xD6, 0xC1, 0xA9, 0x8E, 0x72, 0x57, 0x3F, 0x2B, 0x1D, 0x14, 0x13, 0x18, 0x24, 0x35, 0x4C, 0x65,
  0x80, 0x9B, 0xB4, 0xCA, 0xDB, 0xE6, 0xEB, 0xE9, 0xE0, 0xD2, 0xBE, 0xA7, 0x8D, 0x73, 0x59, 0x42, 0x2F, 0x21, 0x19, 0x18, 0x1D, 0x28, 0x39, 0x4E, 0x66,
  0x80, 0x99, 0xB1, 0xC6, 0xD6, 0xE1, 0xE5, 0xE3, 0xDB, 0xCE, 0xBB, 0xA5, 0x8D, 0x74, 0x5B, 0x46, 0x34, 0x27, 0x1F, 0x1E, 0x23, 0x2E, 0x3D, 0x51, 0x68,
  0x80, 0x98, 0xAE, 0xC2, 0xD1, 0xDB, 0xDF, 0xDD, 0xD5, 0xC9, 0xB7, 0xA2, 0x8C, 0x74, 0x5E, 0x4A, 0x39, 0x2D, 0x26, 0x25, 0x29, 0x33, 0x42, 0x54, 0x6A,
  0x80, 0x96, 0xAB, 0xBD, 0xCB, 0xD4, 0xD8, 0xD6, 0xCF, 0xC3, 0xB3, 0xA0, 0x8B, 0x75, 0x60, 0x4E, 0x3E, 0x33, 0x2D, 0x2C, 0x30, 0x39, 0x47, 0x58, 0x6B,
  0x80, 0x94, 0xA7, 0xB8, 0xC5, 0xCD, 0xD0, 0xCF, 0xC8, 0xBD, 0xAF, 0x9D, 0x8A, 0x76, 0x63, 0x52, 0x44, 0x3A, 0x34, 0x34, 0x37, 0x40, 0x4C, 0x5C, 0x6D,
  0x80, 0x92, 0xA4, 0xB2, 0xBE, 0xC5, 0xC9, 0xC7, 0xC1, 0xB7, 0xAA, 0x9A, 0x89, 0x77, 0x66, 0x57, 0x4B, 0x42, 0x3C, 0x3C, 0x3F, 0x47, 0x52, 0x60, 0x6F,
  0x80, 0x90, 0xA0, 0xAD, 0xB7, 0xBD, 0xC0, 0xBF, 0xBA, 0xB1, 0xA5, 0x97, 0x88, 0x78, 0x69, 0x5C, 0x51, 0x49, 0x45, 0x44, 0x47, 0x4E, 0x58, 0x64, 0x72,
  0x80, 0x8E, 0x9B, 0xA7, 0xAF, 0xB5, 0xB7, 0xB6, 0xB2, 0xAA, 0xA0, 0x94, 0x87, 0x79, 0x6D, 0x61, 0x58, 0x51, 0x4E, 0x4D, 0x50, 0x56, 0x5E, 0x68, 0x74,
  0x80, 0x8C, 0x97, 0xA1, 0xA8, 0xAC, 0xAE, 0xAD, 0xA9, 0xA3, 0x9A, 0x90, 0x86, 0x7B, 0x70, 0x67, 0x5F, 0x5A, 0x57, 0x57, 0x59, 0x5E, 0x64, 0x6D, 0x76,
  0x80, 0x8A, 0x93, 0x9A, 0xA0, 0xA4, 0xA5, 0xA4, 0xA1, 0x9C, 0x95, 0x8D, 0x84, 0x7C, 0x74, 0x6C, 0x66, 0x62, 0x60, 0x60, 0x62, 0x66, 0x6B, 0x71, 0x79,
  0x80, 0x87, 0x8E, 0x93, 0x98, 0x9A, 0x9B, 0x9A, 0x98, 0x94, 0x8F, 0x89, 0x83, 0x7D, 0x77, 0x72, 0x6E, 0x6B, 0x6A, 0x6A, 0x6B, 0x6E, 0x72, 0x76, 0x7B,
  0x80, 0x85, 0x89, 0x8D, 0x8F, 0x91, 0x91, 0x91, 0x8F, 0x8D, 0x89, 0x86, 0x82, 0x7E, 0x7B, 0x78, 0x76, 0x74, 0x73, 0x74, 0x75, 0x76, 0x78, 0x7B, 0x7D,
  0x80, 0x82, 0x84, 0x86, 0x87, 0x88, 0x88, 0x87, 0x86, 0x85, 0x84, 0x82, 0x81, 0x7F, 0x7E, 0x7E, 0x7D, 0x7D, 0x7D, 0x7E, 0x7E, 0x7F, 0x7F, 0x80, 0x80
};
```

Interrupt Service Routine

```
void setup() {
  //Disable interrupts
  bitClear(TIMSK3, TOIE3);

  //Set PWM output pin
  pinMode(PWM, OUTPUT);                    //Timer 3 PWM output is on pin 5 for Arduino Pro Micro

  //Set Counter/Timer 3 Counter Control Registers
  TCCR3A = B10000001;                      //Set for 8-bit, Phase Correct, PWM generation
  TCCR3B = B00000001;                      //Set for no pre-scaling i.e. CLK = MCLK/512 = 31,250Hz

  //Enable interrupts
  bitSet(TIMSK3, TOIE3);
}

ISR(TIMER3_OVF_vect) {
  //Counter/Timer3 Overflow Interrupt Service Routine
  …
  //Output the waveform value at the current waveform index with the current phase
  OCR3A = bpsk31Wave1250Hz[bpsk31WaveIndex] * bpsk31Phase;
  …
}
```