My Experience with Software Defined Radio

Patrick Anglum

N6XTM

SDR Refresher

Software-defined radio (SDR) is a radio communication system where components that conventionally have been implemented in analog hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented or eliminated by means of software on a computer or embedded system.

In a SDR there is no IF section. The demodulation is performed directly on the RF frequency. This eliminates noise and distortion that would otherwise occur in the local oscillator, mixer and IF amplifier sections.

The user interface is usually implemented by some type of GUI interface – either on a PC screen or an LCD GUI display on the radio itself.

While the concept of SDR is not new, the rapidly evolving capabilities of digital electronics render practical many processes which were once only theoretically possible.

About Hermes Lite2 SDR

Direct conversion receiver, no Local Oscillator, Mixer, IF Frequency

RF signal is amplified, then sent directly to the demodulator.

Receiver frequency range 139kHz – 30mHz

Transmit power 5 watts

DC power requirements 11 - 16 Volts @ 2 Amps

Ethernet connectivity to PC, 100mHz or 1gHz

Free PC user interface, SDR Sharp, Thetis, Power SDR…

Based on Altera AD9866 broadband modem chip

Uses PC audio for microphone and headphones

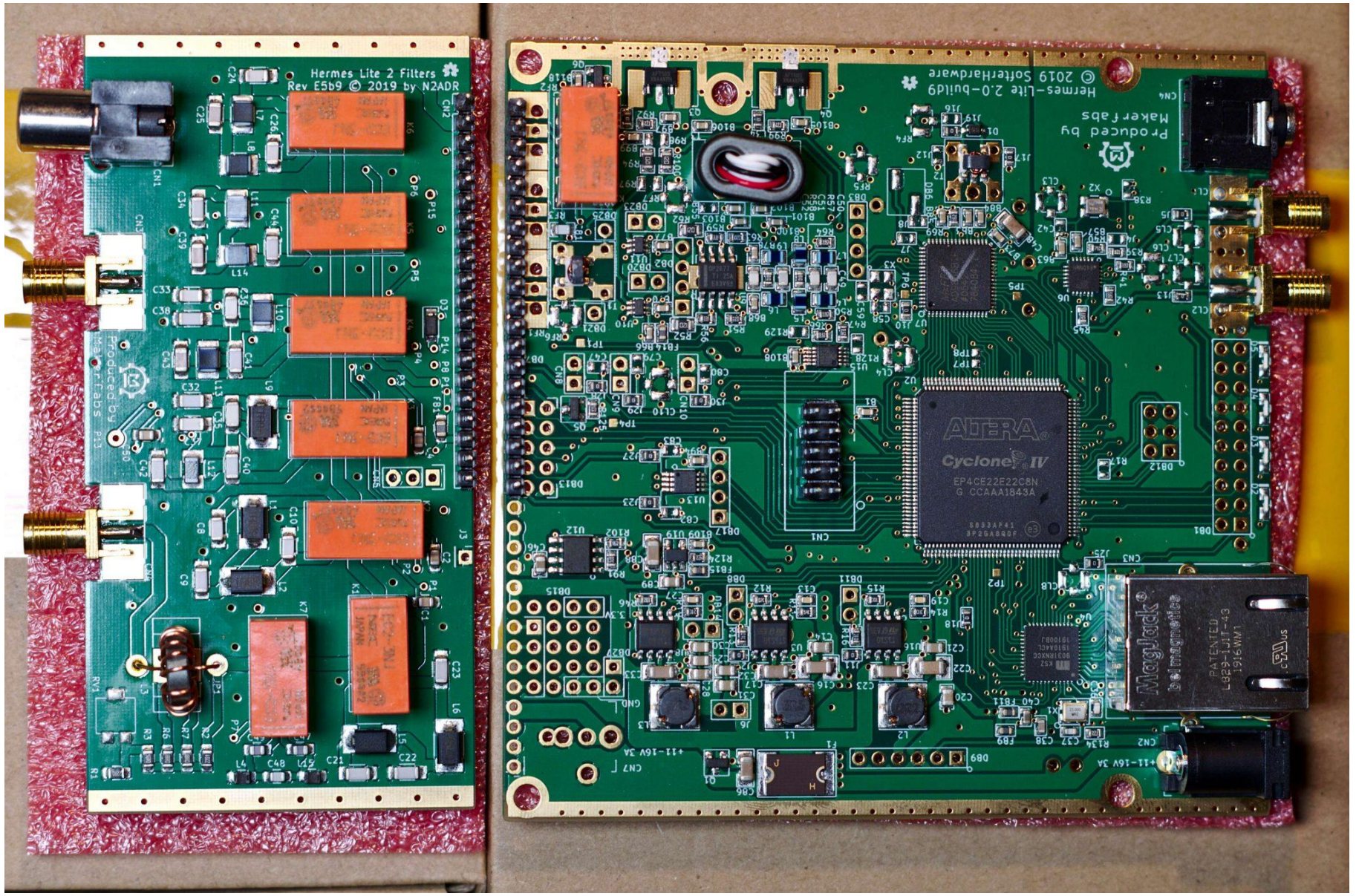
Open Source Software

Open Source Hardware

Available from http://hermeslite.com/

Additional questions: panglum@hotmail.com

Basic Radio Version



Low Pass Filter board

Hermes Lite2 main board

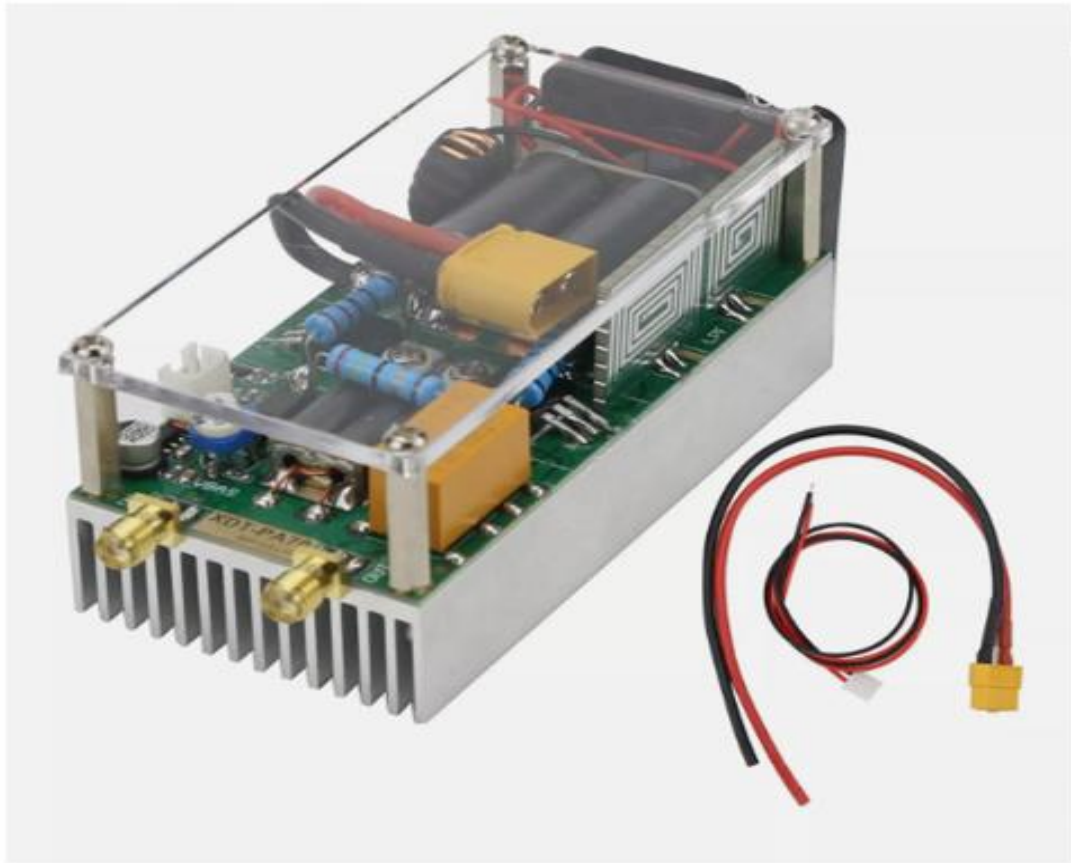
Enclosure

12 Volt Power Supply @ 2 Amps

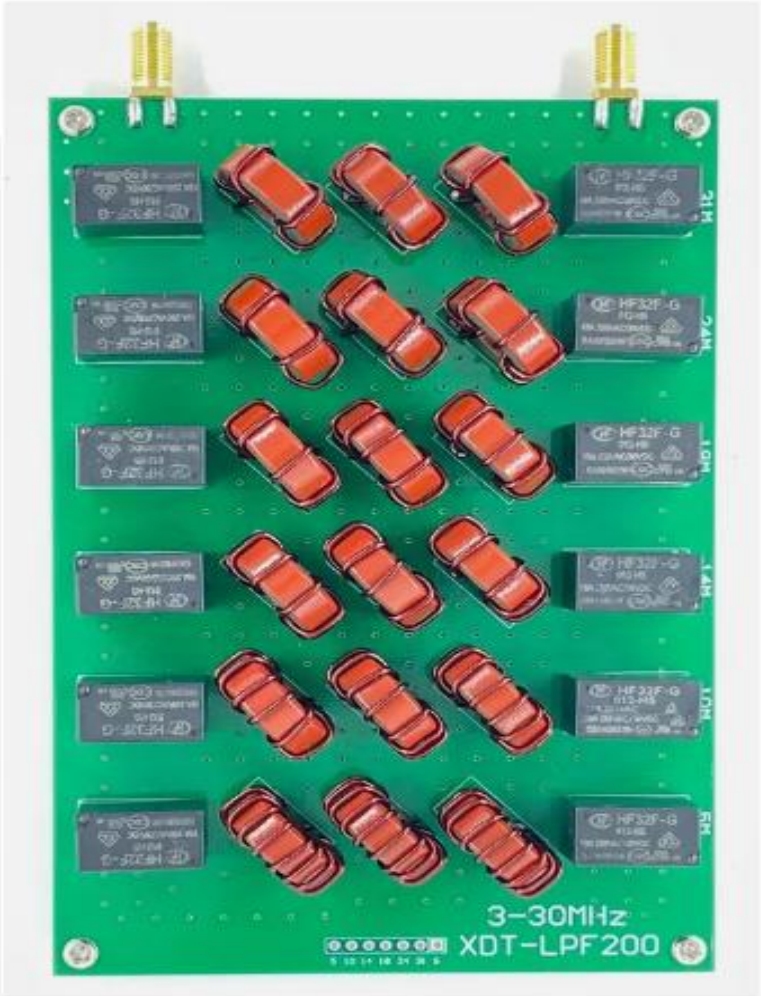
Front Panel

Rear Panel

100 Watt Version



100 Watt amplifier

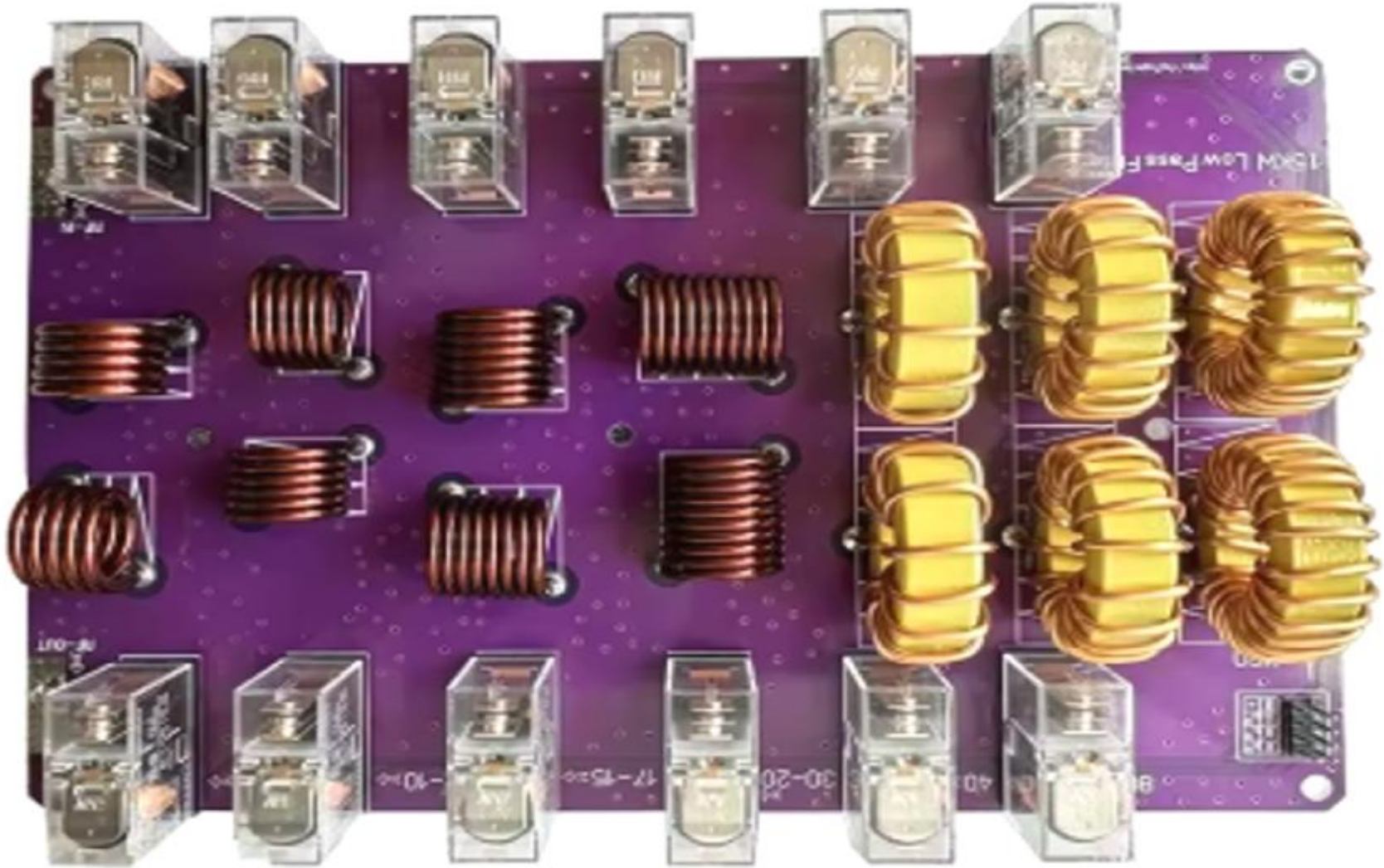


100 Watt low pass filter

12 Volt power supply @ 20 Amps

600 Watt Version

600 Watt Power Amp minus RF power transistors



600 Watt Low Pass Filter

60 Volt power supply (adjusted to 53V) @ 20 Amps

Cost

		Basic Radio	100W PA	600W PA
Hermes Lite 2	$270.00	X	X	X
N2ADR Board	$53.00	X	X	X
HL2 Enclosure	$17.00	X	X	X
GUI PC Software	$0.00	X	X	X
12V Power Supply	$20.00	X	X	X
60V Power Supply	$63.00			X
100W LPF	$100.00		X	
600W LPF	$150.00			X
100W Power Amp	$97.00		X	
600W Power Amp	$87.00			X
N Channel RF Power Transistor	$45.00			X
P Channel RF Power Transistor	$45.00			X
Panel Mount SO239 Cable @	$10.00	X	X	X
		$390.00	$560.00	$760.00

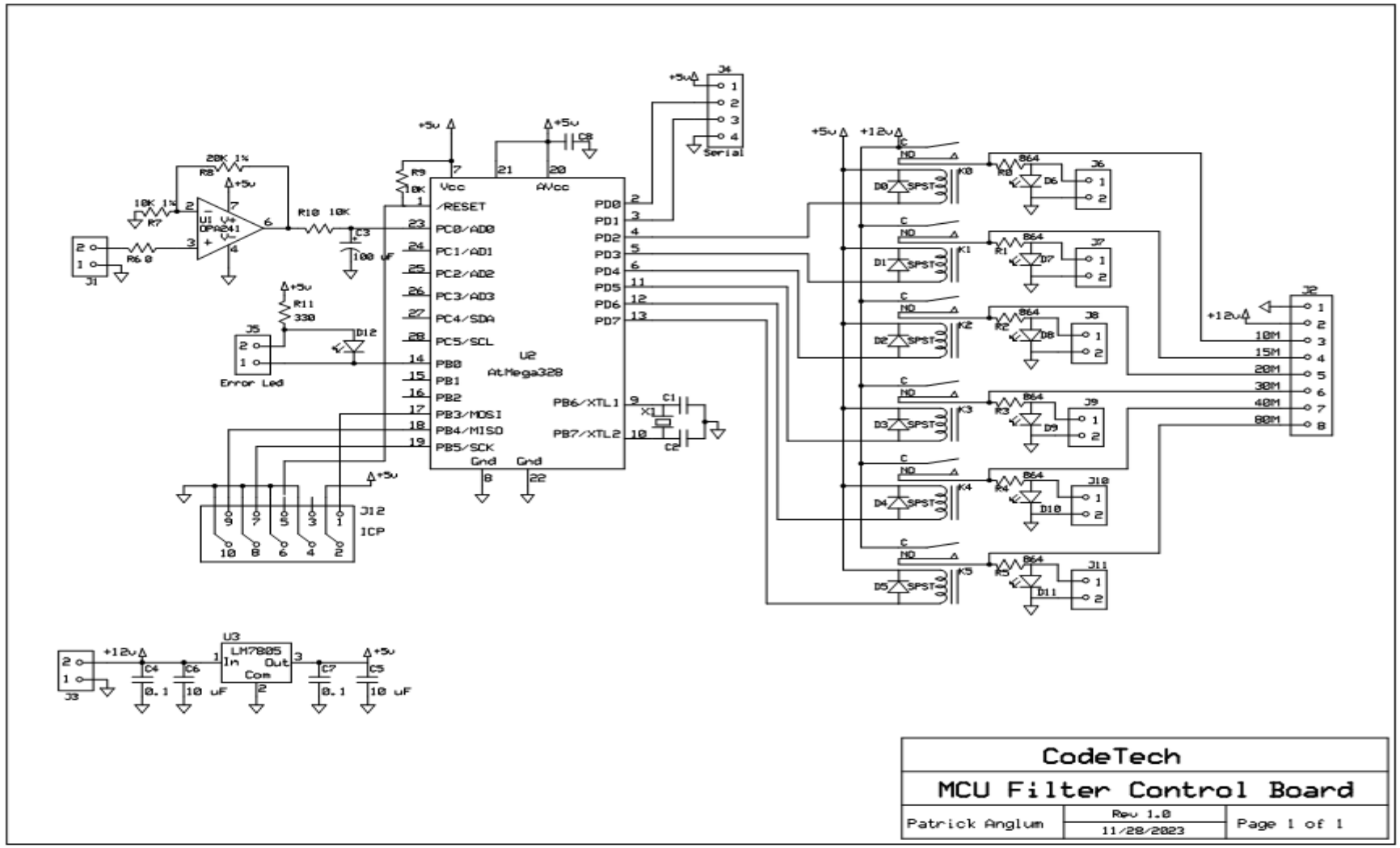
Auto Band Select Circuit

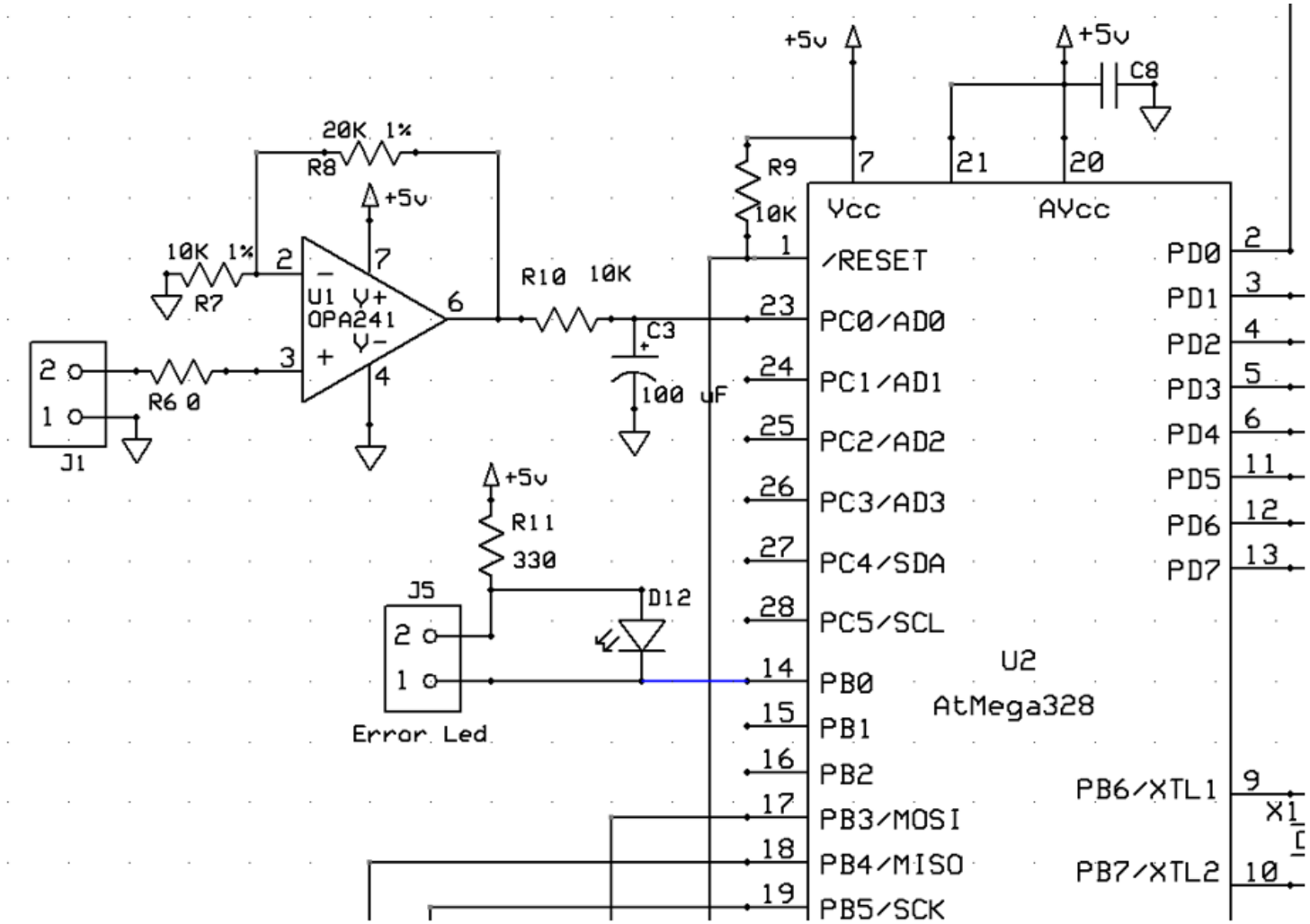
Hermes Lite2 outputs a Pulse Width Modulated signal based on the band selected. It varies from 230 mV to 2300mV.

Band	Level(mV)	Actual
1.8 MHz	230	270
3.8 MHz	460	502
5.0 MHz	690	720
7.0 MHz	920	958
10.0 MHz	1150	1183
14.0 MHz	1380	1402
18.0 MHz	1610	1620
21.0 MHz	1840	1836
24.0 MHz	2070	2048
28.0 MHz	2300	2257

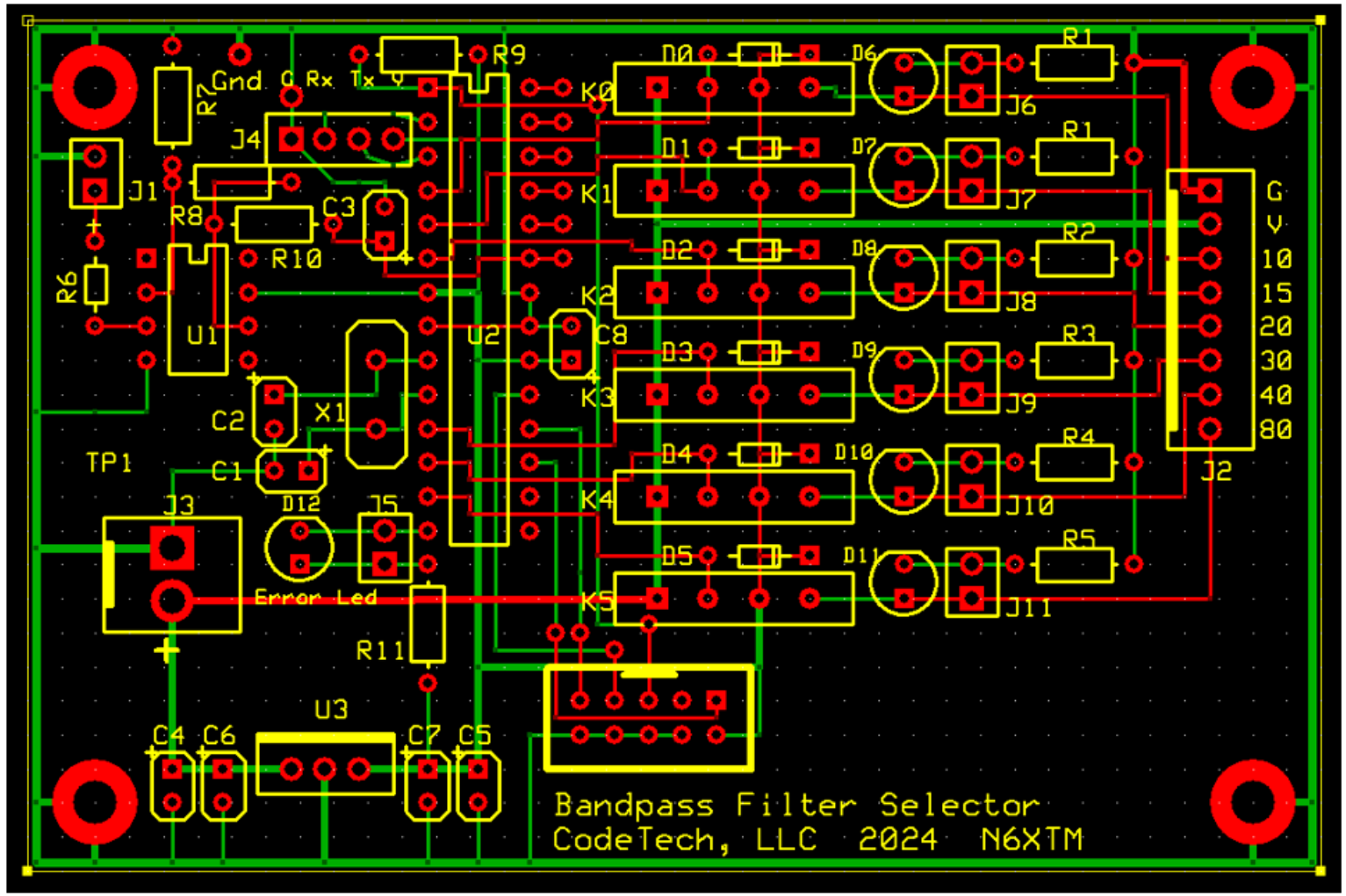
The PWM signal from the Hermes is fed to U1, an op-amp with a gain of two. R10 and C3 integrate the signal to provide a DC voltage to the A/D input on U2 (AtMega328). Based on the voltage, U2 outputs a logic low on one of six digital outputs. This logic low voltage actuates one of six relays which will output 12 volts. The 12 Volts is sent to the Low Pass Filter board, which selects the corresponding filter.

Schematic


CodeTech
MCU Filter Control Board
Patrick Anglum
Rev 1.0
11/28/2023
Page 1 of 1
ATMega328
Error Led
Serial
ICP
LM7805
OPA241
10M
15M
20M
30M
40M
80M


+5v
+5v
C8
20K 1%
R8
R9
10K
7
21
20
Vcc
AVcc
+5v
10K 1%
R7
2
7
U1 V+
OPA241
V-
6
R10 10K
C3
100 uF
/RESET
1
PD0
2
PD1
3
PD2
4
PD3
5
PD4
6
PD5
11
PD6
12
PD7
13
PC0/AD0
23
PC1/AD1
24
PC2/AD2
25
PC3/AD3
26
PC4/SDA
27
PC5/SCL
28
PB0
14
PB1
15
PB2
16
PB3/MOSI
17
PB4/MISO
18
PB5/SCK
19
PB6/XTL1
9
PB7/XTL2
10
X1
U2
AtMega328
2
1
J1
R6 0
3
4
+5v
R11
330
J5
D12
2
1
Error Led

PCB Layout


Gnd
Rx
Tx
R9
R7
J4
J1
R8
C3
R10
R6
U1
U2
C8
C2
X1
C1
TP1
J3
D12
J5
Error Led
R11
U3
C4
C6
C7
C5
K0
K1
K2
K3
K4
K5
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
R1
R1
R2
R3
R4
R5
J6
J7
J8
J9
J10
J11
J2
G
V
10
15
20
30
40
80
Bandpass Filter Selector
CodeTech, LLC 2024 N6XTM

Source Code

```
void setup()
{
  DDRD = 0xFF;   // all bits of PORTD are outputs
  PORTD = 0xFF;  // all bits of PORTD are one
  DDRB = 0x0F;   // 4 low bits of PORTB are outputs
  PORTB = 0x0F;  // 4 low bits of PORTB are one
}

void loop()
{
    uint16_t val = 0;  // variable to store the value read
    uint16_t analogPin = A0; //perform A/D conversion on AD0
    uint16_t average = 0;
    Serial.begin(9600);  //automatically uses D0 and D1



while(1)
{
    delay(500);
    val = analogRead(analogPin);  // read the input pin
    average = GetAverage();


    Serial.print("Raw Value = ");
    Serial.print(val);
    Serial.print("    ");
    Serial.print("Average = ");
    Serial.print(average);
    Serial.print("\n");
```

```
if (average > 0 && average <= 5)  AlarmLedOn();   // below range use B0 error LED
if (average > 5 && average <= 132)  AlarmLedOn();   // 160 meters use B0 error LED
if (average > 32 && average <= 100)
  {
    PORTD = 0b01111111;                    // 80 meters use D7 for 80M
    AlarmLedOff();
  }
  if (average > 100 && average <= 190)  AlarmLedOn();   // 60 meters use B0 error LED
  if (average > 190 && average <= 316)
  {
   PORTD = 0b10111111;                    // 40 meters use D6 for 40M
   AlarmLedOff();
  }
  if (average > 316 && average <= 386)
  {
   PORTD = 0b11011111;                    // 30 meters use D5 for 30M
   AlarmLedOff();
  }
  if (average > 386 && average <= 455)
  {
   PORTD = 0b11101111;                    // 20 meters use D4 for 20M
   AlarmLedOff();
  }
  if (average > 455 && average <= 525)  AlarmLedOn();   // 17 meters use B0 error LED
  if (average > 525 && average <= 599)
  {
   PORTD = 0b11110111;                    // 15 meters use D3 for 15M
   AlarmLedOff();
  }
  if (average > 599 && average <= 675)  AlarmLedOn();   // 12 meters use B0 error LED
  if (average > 675 && average <= 755)
  {
```

```
    PORTD = 0b11111011;                  // 10 meters use D2 for 10M
    AlarmLedOff();
   }
   if (average > 755 && average <= 1023) AlarmLedOn();  // above range use B0 error LED
  }//endof while
}//endof main



void AlarmLedOn() // PORTB0 low
{
  PORTB = PORTB & 0b11111110;
  PORTD = 0b11111111;
}

void AlarmLedOff() // PORTB0 high
{
  PORTB = PORTB | 0b00000001;
}

uint16_t GetAverage()
{
   uint32_t new1 = 0;
   uint32_t analogPin = A0; //perform A/D conversion on AD0
   uint32_t total = 0;
   uint32_t average = 0;


   for (int x = 1; x <= 300; x++)
   {
    new1 = analogRead(analogPin);
    total = total + new1;
   }

   average = (total / 300);
   total = 0;
   return average;
```