14th December 2020

“ad9850 dds vfo”

Parallel access programing.

alastair john underwood, GW0AJU

For a while now I have been undertaking a “Packet Radio” modem project.

The RTTY application, there are many online offerings for the Arduino “C” coding, but I have often wondered what would be required to create ones own Packet Radio style modem.

The first study was how to use an Arduino processor device as a smart modem, or sort of. The need for a serial terminal to access the Arduino turned out not be such a problem, as the Arduino IDE provides a serial terminal, referred as the monitor.

However a while ago, I had a MS terminal program that run on windows XP, thought to have miss-layed the program file, now recently a while ago, discovered the “MS terminal” as a 1992 file, on a 64Mbyte memory stick.

Yes, a 64Mbyte memory stick. In those days, a 128Mbyte memory stick cost retail £128 pounds sterling. Now today, a 128GByte memory stick is some £30 quid or so.

Needless to say, win10 did not touch it, but with win7, it was ok.

Anyway, I thought I have ago at a Visual Basic version, using 2019 free Microsoft VB download.

It is a good job that there are many online websites with examples of a terminal modem programs. Viewing a few on websites, I decided to use this data and put together my own custom MS terminal emulator, to access control the Arduino processor board for the Packet Radio modem.

At first for the Arduino, in this case I used the Arduino Mega, only some 64bytes of text could be sent. I tried increasing the TX / RX buffers, however by altering the “serial.h” file, managed to get to work with a 1500 character buffer.

However, none of this has the ability to be used, if the tone encoder and decoder within the modem does not work sufficiently and stable enough.

To start with, a 555 timer circuit that was FM shift modulated by varying the voltage on pin 5, with the data signal. After trial and many tribulations later, I final came to the conclusion that the 555timer was not perhaps be sufficient enough.

The only thing I could consider, that is to say my brain eventually settled on, was to use the ad9850 dds vfo chip, found as usual available from Ebay or Amazon, as well as the Arduino Mega board, a 3rd party version does well enough though.

Determining the code values was not so much a problem while serial programming the dds vfo, but to parallel code program required a little scientific calculator work. In the end, after looking around, I found a nice simple well thought out Arduino code to determine the parallel code hex values.

By using an Arduino Nano as an “external PC maths co-processor”, a bit cheeky though, using the Arduino ide serial monitor to talk to the Nano, the Arduino code was put to work, Appendix “A”.

However, perhaps I though I should also modify the “C code” into BBC Basic coding, just to make things a little easier to use, Appendix “B”.

The next question, what was I going to use to interface the ad9850 with to parallel program.

The processor of choice for the external vfo, is the PICAXE 18m2, running at a 32MHz clock, coded using PICAXE Basic.

Appendix “C”, the 18m2 PICAXE test coding, arranged as a “packet radio” modem VFO. By the way, the 18m2 modem tone encoding program is only 96 Bytes long, out of the 2048 Bytes available on the 18m2 chip.

The 18m2 device used for the DDS controller, for packet Radio a “2400Hz” tone employed as an on/off keying principle.

With the on/off keying at 1200 Baud data rate, it is perhaps best to synchronously demodulate the Rx data tones, by multiplying the Rx data tone with itself.

At 1200 Baud rate, the highest analogue frequency is at most 600Hz, easier for sharp edged 600Hz lowpass filter to filter reject the 2400Hz data carrier tone, at the receiver detector stage.

At 1200 Baud rate, this equates to two cycles per bit at 2400Hz, or more cycles at a lower data rate.

With the 2400Hz on/off keying, the lower sideband of the packet modem tone encoder audio will only show through the radio set, due to the bandpass filtering of the voice audio circuits of the transmitter, with a SSB transmitter, ultimately the SSB “I.F.” filtering stage.

Appendix “D”, the Arduino Mega test coding. The value then in the example code, are shown to be for a 7·061445MHz carrier frequency.

Hope the info helps other hams.

Regards

Alastair GW0AJU

P.S.

I found that the ad9850 dds vfo, works quite well as an RF fsk modulator, example shown with the Arduino mega, is straight out on the 40m band. The PICAXE version, shown as the AF tone encoder, but the parallel program codes just as well be the 40m band settings.

Appendix “A”

```
/*
 * Arduino nano P.C. co-processor application
 * parallel programming codes for ad9850 dds vfo
 * By Alastair GW0AJU
 * date: 12th November 2020
 */


double DDS_CLOCK = 125000000;
double frequency = 7.061445E6;
uint32_t num;
char hex[8];

void setup()
{
Serial.begin(9600);
num = (frequency * pow(2, 32)) / DDS_CLOCK;
dec_hex();
Serial.print("w1 = ");
Serial.print(hex[7]);
Serial.println(hex[6]);
Serial.print("w2 = ");
Serial.print(hex[5]);
Serial.println(hex[4]);
Serial.print("w3 = ");
Serial.print(hex[3]);
Serial.println(hex[2]);
Serial.print("w4 = ");
Serial.print(hex[1]);
Serial.println(hex[0]);
}

void loop() {}

// decimal to hex converter

void dec_hex()
{
  int rem = 0,i=0;
  while(num > 0 && i >=0){
    rem = num%16;
    hex[i] = rem<10 ? (char)rem+48 : (char)rem+55;
    num/=16;
    i++;
  }
}
```

Appendix “B”

```
  5  REM BBC BASIC application
 10 REM calculator for ad9850 tuning data words
 20 clk_in = 125E6
 30 DIM hex(8)
 40 PRINT TAB(5);"w0 = ";"00 Hex";"  pwr_up"
 50 PRINT TAB(5);"w0 = ";"04 Hex";"  pwr_dwn"
 60 PRINT TAB(5);"w1, b31 - b25"
 70 PRINT TAB(5);"w2, b23 - b16"
 80 PRINT TAB(5);"w3, b15 - b8"
 90 PRINT TAB(5);"w4, b7 -  b0"
100 PRINT
110 PRINT
120
130 INPUT "input frequency required "; f_out
140 PROC_ad9850
150 GOTO 130
160 END
170
180
190 DEF PROC_ad9850
200 REM f_out = ( phase * clk_in ) / 2^32
210 num = INT(( f_out * 2^32 ) / clk_in)
220 rem = 0
230 i = 0
240 WHILE (num >0 AND i <=8)
250   rem = num MOD 16
260   IF rem < 10 THEN hexs = rem+48
270   IF rem >= 10 THEN hexs = rem+55
280   hex(i) = hexs
290   num = (num / 16)
300   i = i + 1
310 ENDWHILE
320 PRINT
330 PRINT TAB(5);"w1 = ";CHR$(hex(7));CHR$(hex(6))
340 PRINT TAB(5);"w2 = ";CHR$(hex(5));CHR$(hex(4))
350 PRINT TAB(5);"w3 = ";CHR$(hex(3));CHR$(hex(2))
360 PRINT TAB(5);"w4 = ";CHR$(hex(1));CHR$(hex(0))
370 PRINT
380 PRINT
390 ENDPROC
```

Appendix "C"

```
' By Alastair GW0AJU' Packet Memo tone generator
' date : 14th December 2020

      'B.0 to B.7 are outputs for DDS data input
      dirsB = %11111111
      'pinsB = %11111111

      'C.0 = FQ_UD low/high, C.1 = W_CLK and C.2 = RESET pulse high = outputs,
       'c.3 = output, c.4 to c.7 = inputs
      dirsC = %00000111
      'pinsC = %00000111

       symbol detect_on = b1

       detect_on = 1  ' to run tone_off key, then to reset "detect_on" to zero

       pause 200  ' delay for 200ms to allow the AD9850 DDS to reset and startup

       goto sub_tone_off ' to run tone_off key, then to reset "detect_on" to zero

      main:

       ptt_sense_loop:
       IF pinC.7 = 1 then goto Packet_Memo_modem_data ' sample the modem ptt sense
       goto ptt_sense_loop

       end

       ' to on/off key the modem Tx tone by sensing the "serial3 Tx"

       Packet_Memo_modem_data:
       IF pinC.6 = 1 then goto sub_tone_on
       IF pinC.6 = 0 then goto sub_tone_off
       return

      ' Packet Memo terminal tone on
      ' RF carrier output = 2400Hz, 0000421F Hex "W1 - W4"

      sub_tone_on:

       IF detect_on = 1 then goto key_on_end

      HIGH 7 ' serial rtty output
      LOW 0  ' FQ_UD low

      LET pinsB = $00  ' DDS W0
      pulsout 1,1

      LET pinsB = $00  ' DDS W1
      pulsout 1,1

      LET pinsB = $00  ' DDS W2
      pulsout 1,1

      LET pinsB = $42  ' DDS W3
      pulsout 1,1

      LET pinsB = $1F  ' DDS W4
      pulsout 1,1

      HIGH 0  ' FQ_UD High

       detect_on = 1

       key_on_end:

       return
```

```
' Packet Memo terminal tone off
      ' RF carrier output = 2400Hz, 0000421F Hex "W1 - W4"

      sub_tone_off:

       IF detect_on = 0 then goto key_off_end

      HIGH 7 ' serial rtty output
      LOW 0  ' FQ_UD low

      LET pinsB = $04  ' DDS W0
      pulsout 1,1

      LET pinsB = $00  ' DDS W1
      pulsout 1,1

      LET pinsB = $00  ' DDS W2
      pulsout 1,1

      LET pinsB = $42  ' DDS W3
      pulsout 1,1

      LET pinsB = $1F  ' DDS W4
      pulsout 1,1

      HIGH 0  ' FQ_UD High

       detect_on = 0

       key_off_end:

       return
```

Appendix “D”

```
/*
 * Arduino mega parallel access for ad9850 dds vfo
 * By Alastair GW0AJU
 * date 12th November 2020
 */

void setup()
{
DDRA = B11111111;  // pins 22-29
DDRC = B11111111;  // pins 37-30
PORTA = 0x00;
PORTC = 0x00;
PORTC = B00000000;  // FQ_UD = low; rst = low; w_clk = low
PORTC = B01000000;  // FQ_UD = low; rst = high; w_clk = low
PORTC = B00000000;  // FQ_UD = low; rst = low; w_clk = low
}

void loop()
{
delay(1000); // transmission keyboard simulation delay
// Null bytes test signal for RTTY RF VFO modem
mark();
delay(20);
space();
delay(20);
space();
delay(20);
space();
delay(20);
space();
delay(20);
space();
delay(20);
space();
delay(20);
mark();
delay(30);
delay(1000); // transmission keyboard simulation delay
}

  // Logic zero space tone carrier
  // RF carrier output = 7.061275MHz, 000E76244AHex

 void space()
    {
      PORTC = B10000000;  // FQ_UD = high; w_clk = low
      PORTC = B00000000;  // FQ_UD = low; w_clk = low

      PORTA = 0x00;   // DDS W0
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x0E;  // DDS W1
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk
```

```
      PORTA =  0x76;  // DDS W2
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x24;  // DDS W3
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x4A;  // DDS W4
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTC = B10000000;  // FQ_UD High; w_clk = low
    }


      // Logic one mark tone carrier
      // RF carrier output = 7.061445MHz, 000E763B1BHex

void  mark()
      {

      PORTC = B10000000;  // FQ_UD = high; w_clk = low
      PORTC = B00000000;  // FQ_UD = low; w_clk = low

      PORTA =  0x00;  // DDS W0
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x0E;  // DDS W1
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x76;  // DDS W2
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x3B;  // DDS W3
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTA =  0x1B;  // DDS W4
      PORTC = B00000001; // w_clk
      PORTC = B00000000; // w_clk

      PORTC = B10000000; // FQ_UD High; w_clk = low
      }
```