```
#include "STM32TimerInterrupt.h"
#include <SPIFlash.h>
#include <Wire.h>
#define SS_FLASHMEM PB12
uint16_t expectedDeviceID=0xEF40;
SPIFlash flash(SS_FLASHMEM, expectedDeviceID);
uint32_t memAddress;
int L = 31;
#define RXD 3
#define TXD 2
#define PTT PA12
#define PD PC14
HardwareSerial Serial1(USART1);
STM32Timer ITimer0(TIM1),ITimer1(TIM2); // Init STM32 timer
TIM1
unsigned short int dacBuffer[256]={0},tmpSmp=0;
volatile unsigned char head = 0;
volatile unsigned char tail = 0;
volatile uint16_t samples[12]={0};
volatile uint64_t basicBlkBytes={0};
volatile int indx = 0;
volatile byte nextsample=0;
unsigned short cnt=0;
bool flag=false;
volatile bool bitStuffFlag = false;
int waitState = 1;
unsigned char tmFrameCnt=0;
int16_t x0=0, x1=0, y01=0, y11=0;
int8_t delaySamp[8] = {0};
uint8_t NRZIDecBits=0,olbByte=0;
uint32_t sampledBits=0, actualBits=0;
uint8_t curr_phase=0;
int bytCnt=0;
```

```
uint16_t EOTTBytes=0;
bool debug = true;
volatile byte startTX = 0;
bool nextCS = false;
unsigned short crc = 0xffff;

unsigned char alias[] = "APRS";
char src[6] = {0},dest[6] = {0},digi[8][6] = {0};//,
digi2[6] = {0},digi3[6] = {0},digi4[6] = {0},digi5[6] = {0},
digi6[6] = {0},digi7[6] = {0},digi8[6] = {0};
unsigned char srcSSID = 0,destSSID=0,digiSSID[8]={0};
char msg[256] = {0};
volatile short int secondCnt=0, timeCount=0;
unsigned char digiCnt=0;
unsigned char CF = 0,CF_ind = 0;
unsigned char PID = 0,PID_ind = 0;
unsigned char Ndigi=0,APRSBytes=0,beaconId=0;

//---------------------- APRS Tx
int value=0;
int toneid= 0;
unsigned char countOnes = 0;

int count = 0;
int udelay = 50;

unsigned short int sineLUT[] =
{252,256,260,264,268,276,280,284,288,292,296,300,304,308,312
,316,324,328,332,336,340,344,348,352,356,360,364,368,372,376
,380,384,388,392,396,400,400,404,408,412,416,420,424,424,428
,432,436,440,440,444,448,448,452,456,456,460,464,464,468,472
,472,476,476,480,480,484,484,488,488,488,492,492,492,496,496
,496,500,500,500,500,504,504,504,504,504,504,504,504,504,504
```

```
,504,504,504,504,504,504,504,504,504,504,500,500,500,500,500
,496,496,496,492,492,492,488,488,484,484,480,480,476,476,472
,472,468,468,464,460,460,456,452,452,448,444,444,440,436,432
,432,428,424,420,416,412,412,408,404,400,396,392,388,384,380
,376,372,368,364,360,356,352,348,344,340,336,332,328,324,320
,316,312,308,304,300,296,288,284,280,276,272,268,264,260,256
,248,244,240,236,232,228,224,220,216,208,204,200,196,192,188
,184,180,176,172,168,164,160,156,152,148,144,140,136,132,128
,124,120,116,112,108,104,100,96,92,92,88,84,80,76,72,72,68,6
4,60,60,56,52,52,48,44,44,40,36,36,32,32,28,28,24,24,20,20,1
6,16,12,12,12,8,8,8,4,4,4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,4,4,4,4,8,8,8,12,12,12,16,16,16,20,20,24,24,28,28,
32,32,36,40,40,44,48,48,52,56,56,60,64,64,68,72,76,80,80,84,
88,92,96,100,104,104,108,112,116,120,124,128,132,136,140,144
,148,152,156,160,164,168,172,176,180,188,192,196,200,204,208
,212,216,220,224,228,236,240,244,248,252};
double gtheta = 0;
double deltaTheta;
int val=0;
unsigned short deg;
volatile short int idx=0;
double g_fudge;
double sampRate = 52631;
double twoPibysampRate  = 360.0/sampRate;
double uSperSample = 1000000.0 /(sampRate);
const int S = 100; // Number of samples to read in block
volatile unsigned char dat=0,gotPkt = false;
//----------------------------PlayTone-------------------------
-----
void playtone(uint16_t tonefreq , double tonedur )
{
    uint16_t tonesamples, i ;
    double   deltatheta;
    tonedur += g_fudge ;
   tonesamples = tonedur/(uSperSample);

   deltatheta = twoPibysampRate * tonefreq ;

   for(i=1;i<=tonesamples;i++)
    {
         gtheta +=deltatheta;
         if (gtheta>360)
         gtheta -= 360;
         deg = (unsigned short) gtheta;
        while((head==0&&tail==255)||(tail==head-1));
         dacBuffer[tail] = sineLUT[deg];
         if (tail == 255)
           tail = 0;
         else
           tail++;
    }
    g_fudge = tonedur - ( tonesamples * uSperSample );

}  // end playtone

//-----------------------------------------------------------
----
void parseBytes()
{
  unsigned short xor_in;
  if (bytCnt==0){
    crc = 0xffff;
  digiCnt=0;
  if(debug)  Serial2.print("\n");
   }
  else if((((crc ^ 0xff)& 0xff)== olbByte) and (((crc >> 8)
^ 0xff)== APRSBytes) and APRSBytes != 0)
  {
    unsigned char msglen = (bytCnt-(PID_ind+2));
    crc = 0xffff;
    gotPkt = true;
  }
  else{
    msg[bytCnt] = APRSBytes;
    unsigned char inByte = olbByte;
      for(int i=0;i<8;i++)
  {
    xor_in = crc ^ (inByte & 1);
  crc >>= 1;

  if(xor_in & 0x01)
    crc ^= 0x8408;

    inByte >>= 1;
  }
  }
    if(bytCnt<6)
    {
      dest[bytCnt] = APRSBytes>>1;
      if(debug) Serial2.write(APRSBytes/2);
    }
    else if(bytCnt==6)
    {
      destSSID = APRSBytes;
      if (APRSBytes & 0x01)
      nextCS = false;
      else
      nextCS = true;
      if(debug)
      {
      Serial2.write('-');
     Serial2.write(0x30+(APRSBytes&0x1f)/2);
      }
    }
    else if(bytCnt==13)
    {
      srcSSID = APRSBytes;
      if (APRSBytes & 0x01)
      nextCS = false;
      else
      nextCS = true;
      if(debug)
      {
      Serial2.write('-');
     Serial2.write(0x30+(APRSBytes&0x1f)/2);
      }
    }
    else if(nextCS and (bytCnt==20 or bytCnt==27 or
bytCnt==34 or bytCnt==41 or bytCnt==48 or bytCnt==55 or
bytCnt==62 or bytCnt==69))
    {
      if (APRSBytes & 0x01){
      nextCS = false;
      CF_ind = bytCnt+1;
      PID_ind = bytCnt+2;
      }
      else
      {
      nextCS = true;
      }
     digiSSID[((bytCnt+1)/7)-3] = APRSBytes;
      if(debug){
```

```
      Serial2.write('-');
      Serial2.write(0x30+(APRSBytes & 0x1f)/2);
       }
    }
    else if(bytCnt>6 and bytCnt<13)
    {
      src[bytCnt-7] = APRSBytes>>1;
      if(debug) Serial2.write(APRSBytes/2);
    }
    else if(nextCS and (bytCnt>13 and bytCnt<20))
    {
      digi[0][bytCnt-14] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
    else if(nextCS and (bytCnt>20 and bytCnt<27))
    {
      digi[1][bytCnt-21] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
    else if(nextCS and (bytCnt>27 and bytCnt<34))
    {
      digi[2][bytCnt-28] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
    else if(nextCS and (bytCnt>34 and bytCnt<41))
    {
      digi[3][bytCnt-35] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
else if(nextCS and (bytCnt>41 and bytCnt<48))
           {
      digi[3][bytCnt-42] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
else if(nextCS and (bytCnt>47 and bytCnt<55))
          {
      digi[5][bytCnt-48] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
else if(nextCS and (bytCnt>55 and bytCnt<62))
         {
      digi[6][bytCnt-56] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
else if(nextCS and (bytCnt>62 and bytCnt<69))
          {
      digi[7][bytCnt-63] = APRSBytes>>1;
      if (debug)      Serial2.write(APRSBytes/2);
    }
    else if (bytCnt == CF_ind)
    {
      CF = APRSBytes;
      if (debug) Serial2.print(APRSBytes,HEX);
    }
    else if (bytCnt == PID_ind)
    {
      PID = APRSBytes;
    if (debug)    Serial2.print(APRSBytes,HEX);
    }
    else
    {
      msg[bytCnt-(PID_ind+1)] = APRSBytes;
      if (debug) Serial2.write(APRSBytes);
    }
}
```

```
//------------------------------Interrupt
SR----------------------
void TimerHandler0()
{
  if (!startTX){
  int16_t a0;
  int16_t a1;
  a1 = adc_read_value(PA_0,12);
  a0 = (int16_t) (a1>>4)-121;


  for (int ind=7; ind>0; ind--)
  {
    delaySamp[ind] = delaySamp[ind-1];
  }
  delaySamp[0] = a0;

  x0 = x1;
  x1 = ((int8_t) delaySamp[4]*a0)>>2;
  y01 = y11;
  y11 = x0+x1+(y01>>1);

  sampledBits <<= 1;
  sampledBits |= y11>0?1:0;


  if (((sampledBits ^ (sampledBits>>2))&0x03) == 0x03) //
signal transition LS 4 bits = 0011 or 1100
  {
    if (curr_phase < 32)
      curr_phase++;
    else
      curr_phase--;
  }
  curr_phase += 8;

  if (curr_phase >= 64)
  {
    curr_phase %= 64;
    actualBits <<= 1;

    uint8_t bits = sampledBits & 0x07;
    if (bits == 0x07 || // 111
        bits == 0x06 || // 110
        bits == 0x05 || // 101
        bits == 0x03    // 011
        ) {
        actualBits |= 1;
    }
    NRZIDecBits >>= 1;
   NRZIDecBits |= ((actualBits ^ (actualBits>>1))&0x01)?
0x00:0x80;
    cnt++;
      if (((NRZIDecBits & 0xfe) == 0x7c))
      {
      NRZIDecBits <<= 1;
      cnt--;
      bitStuffFlag = true;
      }
      else
      bitStuffFlag = false;
    if (NRZIDecBits == 0x7E and cnt>=8)
    {
      flag = true;
      cnt=0;
     NRZIDecBits = 0;
     olbByte = NRZIDecBits;
     bytCnt = 0;
    }
    else if (flag && cnt==8)
    {


      if(NRZIDecBits==0x7E and olbByte == 0x7E)
        bytCnt=0;
      else if((NRZIDecBits==0x7E and olbByte != 0x7E) or
bytCnt>=255)
      {
        flag = false;
      }
      else if(NRZIDecBits==0xff)
      {
        flag = false;
      }
      else{
      APRSBytes = NRZIDecBits;
      parseBytes();
      bytCnt++;
      }
      cnt = 0;
      olbByte = NRZIDecBits;
      NRZIDecBits = 0;
    }
}
  }
  secondCnt++;
  if(secondCnt>9600){ secondCnt=0; timeCount++;}
}
//-------------------------------------------------------------
----
```

```
//-----------------------------Interrupt
SR----------------------
void TimerHandler1()
{
  if (startTX)
  {
    if(head!=tail)
    {
      GPIOB->ODR = dacBuffer[head];
      head++;
      if(head>255)
        head = 0;
    }
    else
      GPIOB->ODR = 0;
  }
}
void AFSK(unsigned char Byte_in, bool flag)
{
 unsigned short xor_in;
 unsigned char test = Byte_in;
  for(int i=0;i<8;i++)
  {
    xor_in = crc ^ (Byte_in & 1);
  crc >>= 1;

  if(xor_in & 0x01)
    crc ^= 0x8408;



    if(Byte_in & 1)
    {
      // toneGen(toneid);
      if(toneid)
    playtone(1200,833.3);
  else
    playtone(2200,833.3);
      countOnes++;
      if(flag && (countOnes==5))
      {
        toneid = !toneid;
        if(toneid)
    playtone(1200,833.3);
  else
    playtone(2200,833.3);
        countOnes = 0;
      }
    }
    else
    {
      toneid = !toneid;
      if(toneid)
    playtone(1200,833.3);
  else
    playtone(2200,833.3);
      countOnes = 0;
    }
    Byte_in >>= 1;
  }
}

void APRS_TX() {
  digitalWrite(PTT,0);
  for(int i=0;i<50;i++) // zeros
    AFSK(0x00,LOW);
```

```
 // Initial Flags
 for(int i=0;i<5;i++)
   AFSK(0x7e,LOW);
 crc = 0xffff; // Seed for CRC

 //Dest
 for(int i=0;i<6;i++)
   AFSK(dest[i]<<1,1);
AFSK(destSSID,1);
 // SRC
 for(int i=0;i<6;i++)
   AFSK(src[i]<<1,1);
AFSK(srcSSID,1);
 // DIGI
 for (int j=0; j<8; j++)
 {
   if(j>(Ndigi-3))
   break;
   for(int i=0;i<6;i++)
     AFSK(digi[j][i]<<1,1);
   AFSK((digiSSID[j]),1);
 }

   // Ctrl Fld
AFSK(CF,1);
   // PID
AFSK(PID,1);

 // payload
 for(int i=0;i<(bytCnt-(PID_ind+3));i++) //
   AFSK(msg[i],1);

 // crc
```

```
  unsigned char crc_lo = crc ^ 0xff;
  unsigned char crc_hi = (crc >> 8) ^ 0xff;
  AFSK(crc_lo, 1);
  AFSK(crc_hi, 1);
  // Final flags
  for(int i=0;i<2;i++)
    AFSK(0x7e,0);
  idx = 0;
//delay(1000);
digitalWrite(PTT,1);
flag = false;
}
//----------------------------------------

// ------------- Audio Tx ---------------
void Audio_tx()
{
  uint8_t Byte;
  unsigned short int sampleByte;
    digitalWrite(PTT,0);
        memAddress = 0x618000;
        digitalWrite(PA4,0);
        SPI.transfer(3);
        SPI.transfer(memAddress >> 16);
        SPI.transfer(memAddress >> 8);
        SPI.transfer(memAddress);
      for(uint32_t i=0; i<486161; i++) //3449433
      {
        Byte = SPI.transfer(0);
        sampleByte = Byte;
       for(uint8_t j=0;j<5;j++)
       {
       while((head==0&&tail==63)||(tail==head-1));
```

```
          dacBuffer[tail] = sampleByte<<2;
          if (tail == 63)
            tail = 0;
          else
            tail++;
       }
      }
   digitalWrite(PA4,1);
   digitalWrite(PTT,1);
}
// ------------ Audio Tx Ends ----------------

// ------------------ SSTV (PD120_tx) Tx -------------------
void PD120_tx()
{
  uint8_t Byte;
 digitalWrite(PTT,0);
   playtone( 1900 , 100000 ) ; // you forgot this one
   playtone( 1500 , 100000 ) ;
   playtone( 1900 , 100000 ) ;
   playtone( 1500 , 100000 ) ;
   playtone( 2300 , 100000 ) ;
   playtone( 1500 , 100000 ) ;
   playtone( 2300 , 100000 ) ;
   playtone( 1500 , 100000 ) ;

   // VIS lead, break, mid, start
   playtone( 1900 , 300000 ) ;
   playtone( 1200 ,  10000 ) ;
 //   playtone( 1500 , 300000 ) ;
   playtone( 1900 , 300000 ) ;
   playtone( 1200 ,  30000 ) ;
   // VIS data bits (PD120)
   playtone( 1100 ,  30000 ) ;
   playtone( 1100 ,  30000 ) ;
   playtone( 1100 ,  30000 ) ;
   playtone( 1100 ,  30000 ) ;
   playtone( 1100 ,  30000 ) ;
   playtone( 1300 ,  30000 ) ;
   playtone( 1100 ,  30000 ) ;
   playtone( 1300 ,  30000 ) ;

   // VIS stop
   playtone( 1200 ,  30000 ) ;
  playtone(1500,20000);

       digitalWrite(PA4,0);
       SPI.transfer(3);
       SPI.transfer(memAddress >> 16);
       SPI.transfer(memAddress >> 8);
       SPI.transfer(memAddress);

   // Image Transmission
   for(int x=0;x<248;x++)
   {
     playtone(1200,20000);
     playtone(1500,2080);
     for (int y=0; y<640; y++)
     {
       Byte = SPI.transfer(0);
       playtone(1500+Byte*3,190);
     }
     for (int y=0; y<640; y++)
     {
       Byte = SPI.transfer(0);
```

```
        playtone(1500+Byte*3,190);
      }
      for (int y=0; y<640; y++)
      {
         Byte = SPI.transfer(0);
        playtone(1500+Byte*3,190);
      }
      for (int y=0; y<640; y++)
      {
         Byte = SPI.transfer(0);
        playtone(1500+Byte*3,190);
      }
    }
    digitalWrite(PA4,1);
    digitalWrite(PTT,1);
}
// ------------------- SSTV (PD120_tx) Ends
-------------------

// -------------------  SSTV (Robot-72) Tx -------------------
void Robot72_tx()
{
  uint8_t Byte;
  digitalWrite(PTT,0);
    playtone( 1900 , 100000 ) ; // you forgot this one
    playtone( 1500 , 100000 ) ;
    playtone( 1900 , 100000 ) ;
    playtone( 1500 , 100000 ) ;
    playtone( 2300 , 100000 ) ;
    playtone( 1500 , 100000 ) ;
    playtone( 2300 , 100000 ) ;
    playtone( 1500 , 100000 ) ;
```

```
 // VIS lead, break, mid, start
 playtone( 1900 , 300000 ) ;
 playtone( 1200 ,  10000 ) ;
 playtone( 1500 , 300000 ) ;
 playtone( 1900 , 300000 ) ;
 playtone( 1200 ,  30000 ) ;

 // VIS data bits (Robot-72)
 playtone( 1300 ,  30000 ) ; //0
 playtone( 1300 ,  30000 ) ; //0
 playtone( 1100 ,  30000 ) ; //1
 playtone( 1100 ,  30000 ) ; //1
 playtone( 1300 ,  30000 ) ; //0
 playtone( 1300 ,  30000 ) ; //0
 playtone( 1300 ,  30000 ) ; //0
 playtone( 1300 ,  30000 ) ; //0

 // VIS stop
 playtone( 1200 ,  30000 ) ;
playtone(1500,20000);

     digitalWrite(PA4,0);
     SPI.transfer(3);
     SPI.transfer(memAddress >> 16);
     SPI.transfer(memAddress >> 8);
     SPI.transfer(memAddress);

 // Image Transmission
 for(int x=0;x<248;x++)
 {
   playtone(1200,9000);
   playtone(1500,3000);
   for (int y=0; y<640; y++)
```

```
      {
        Byte = SPI.transfer(0);
         if(y%2)
        playtone(1500+Byte*3,431.25);
      }

     playtone(1200,4500);
     playtone(1500,1500);
      for (int y=0; y<640; y++)
      {
        Byte = SPI.transfer(0);
         if(y%2)
        playtone(1500+Byte*3,215.625);
      }

     playtone(1200,4500);
     playtone(1500,1500);
      for (int y=0; y<640; y++)
      {
         Byte = SPI.transfer(0);
          if(y%2)
        playtone(1500+Byte*3,215.625);
      }

      for (int y=0; y<640; y++)
      {
         Byte = SPI.transfer(0);
        //playtone(1500+Byte*3,190);
      }
    }
   digitalWrite(PA4,1);
   digitalWrite(PTT,1);
}
```

```
// ------------------- SSTV (Robot-72) Ends
-------------------
void SSTV_tx(uint8_t mode, uint32_t start_Addr)
{
  memAddress = start_Addr;
  switch(mode)
  {
    case 0: PD120_tx(); break;
    case 1: Robot72_tx(); break;
  }
}

//---------------QSLStarts--------------------------------
void QSL_tx()
{
  memAddress = 0x4E0000;
  Robot72_tx();
}
//-----------------------------SETUP-----------------------------
--
void setup()
{

cnt = 0;
  Serial1.begin(9600);
//  DDRB |= 0b00000100;
// pinMode(data_out,OUTPUT);
  pinMode(PTT,OUTPUT);
  pinMode(PD,OUTPUT);
  pinMode(PB3, OUTPUT);
  pinMode(PB4, OUTPUT);
  pinMode(PB5, OUTPUT);
  pinMode(PB6, OUTPUT);
  pinMode(PB7, OUTPUT);
  pinMode(PB8, OUTPUT);
  pinMode(PB9, OUTPUT);

  pinMode(PA4, OUTPUT);
  SPI.setMOSI(PA7);
  SPI.setMISO(PA6);
  SPI.setSCLK(PA5);
  digitalWrite(PA4,1);
 SPI.beginTransaction(SPISettings(18000000, MSBFIRST,
SPI_MODE0));
 // pinMode(HbyL,OUTPUT);
 pinMode(LED_BUILTIN,OUTPUT);
//  pinMode(SQ,INPUT);
  Serial2.begin(115200);
//  Audio.begin(sampRate, 100);
  delay(100);

  digitalWrite(PTT,0);
  delay(10);
  digitalWrite(PTT,1);
  digitalWrite(PD,1);
 // digitalWrite(HbyL,0);
  delay(400);
   // Current Sensor
  Wire.setSDA(PB11);
  Wire.setSCL(PB10);
  Wire.begin();

  Serial1.println("AT+DMOSETGROUP=1,144.8000,144.
5000,0000,1,0000");
   Serial1.println("AT+DMOSETVOLUME=8");
    delay(100);
    Serial2.println("Initializing EOTT");
    delay(100);
  // Interval in microsecs
  ITimer0.attachInterruptInterval(104, TimerHandler0);
  ITimer1.attachInterruptInterval(19, TimerHandler1);
}
//--------------------------------------------------------------
----

//-------------------TM_APRS()
--------------------------------
void TM_APRS()
{
  uint16_t Vbus=0, Vshunt=0, Temp=0;
  strcpy(dest,"CQ    ");
  destSSID = 0x60;
  strcpy(src,"ARPIT ");
  srcSSID = 0x60;
  Ndigi = 3;
  strcpy(digi[0],"WIDE2 ");
  digiSSID[0] = 0x63;
  strcpy(msg,"Greetings from BGS-ARPIT Payload!");
  CF = 0x03;
  PID = 0xf0;
  bytCnt = 36;
  PID_ind = 0;
  APRS_TX();

  strcpy(dest,"VU3TJD");
  destSSID = 0x60;
  strcpy(src,"ARPIT ");
  srcSSID = 0x60;
  Ndigi = 3;
  strcpy(digi[0],"WIDE2 ");
  digiSSID[0] = 0x63;
  // form TM frame
  Wire.beginTransmission(0x40); // transmit to device #112
  Wire.write(byte(0x4));      // sets register pointer to
echo #1 register (0x02)
  Wire.endTransmission();      // stop transmitting
  Wire.requestFrom(0x40, 2);    // request 2 bytes from
slave device #112
  if(2 <= Wire.available())    // if two bytes were received
  {
    Vshunt = Wire.read();  // receive high byte (overwrites
previous reading)
    Vshunt = Vshunt << 8;    // shift high byte to be high
8 bits
    Vshunt |= Wire.read(); // receive low byte as lower 8
bits
  }

  Wire.beginTransmission(0x40); // transmit to device #112
  Wire.write(byte(0x5));      // sets register pointer to
echo #1 register (0x02)
  Wire.endTransmission();      // stop transmitting
  Wire.requestFrom(0x40, 2);    // request 2 bytes from
slave device #112
  if(2 <= Wire.available())    // if two bytes were received
  {
    Vbus = Wire.read();  // receive high byte (overwrites
previous reading)
    Vbus = Vbus << 8;    // shift high byte to be high 8
bits
    Vbus |= Wire.read(); // receive low byte as lower 8 bits
  }
  Wire.beginTransmission(0x40); // transmit to device #112
  Wire.write(byte(0x6));      // sets register pointer to
echo #1 register (0x02)
  Wire.endTransmission();      // stop transmitting
  Wire.requestFrom(0x40, 2);    // request 2 bytes from
slave device #112
  if(2 <= Wire.available())    // if two bytes were received
  {
    Temp = Wire.read();  // receive high byte (overwrites
previous reading)
    Temp = Temp << 8;    // shift high byte to be high 8
bits
    Temp |= Wire.read(); // receive low byte as lower 8 bits
  }

  tmFrameCnt++; if(tmFrameCnt>99) tmFrameCnt=0;
  String TmFrm = "TM#" + String(tmFrameCnt,DEC) + "," +
String(Vbus,HEX) + "," + String(Vshunt,HEX)+ "," +
String(Temp,HEX)+ "   ";
 strcpy(msg,TmFrm.c_str());
  //
  CF = 0x03;
  PID = 0xf0;
  bytCnt = 23;
  PID_ind = 0;
  APRS_TX();
}

//---------------------------LOOP---------------------------
----
void loop()
{
```

```
while(1)
{
  if(gotPkt)
  {
    gotPkt = false;
    startTX=1;
    Ndigi = PID_ind/7;
    Serial2.println(PID_ind);
    Serial2.print("Frm ");
    for(int i=0; i<6; i++)
      if((src[i]>>1)!= 0x20)
        Serial2.write(src[i]>>1);
    Serial2.print("-");
    Serial2.write(0x30+((srcSSID & 0x1f)>>1));

    Serial2.print(" To ");
    for(int i=0; i<6; i++)
      if((dest[i]>>1)!= 0x20)
        Serial2.write(dest[i]>>1);
    Serial2.print("-");
    Serial2.write(0x30+((destSSID & 0x1f)>>1));

    if(Ndigi!=0)
    {
      Serial2.print(" Via ");
      for(int j=0;j<(Ndigi-2); j++)
      {
        if (j>0)
          Serial2.print(", ");
        for(int i=0; i<6; i++)
          if((digi[j][i]>>1)!= 0x20)
            Serial2.write(digi[j][i]>>1);
        Serial2.print("-");
```

```
          Serial2.write(0x30+((digiSSID[j] & 0x1f)>>1));
        }
      }
      Serial2.print("\n");

     Serial2.write(msg,bytCnt-(PID_ind+3));
      Serial2.write("\n");

      bool gotalias = false;

      for (int j=0;j<Ndigi-2;j++)
      {
        if((digi[j][0]== 'S') && (digi[j][1]== 'J') &&
(digi[j][2]== 'C') && (digi[j][3]== 'S') && (digi[j][4]==
'A')&& (digi[j][5]== 'T'))
        {
          digiSSID[j] |= 0x80;
          gotalias = true;
        }
      }

      if(!gotalias)
      {
        for (int j=0;j<Ndigi-2;j++)
        {
          if((digi[j][0]== 'A') && (digi[j][1]== 'R') &&
(digi[j][2]== 'I') && (digi[j][3]== 'S') && (digi[j][4]==
'S'))
          {
            digiSSID[j] |= 0x80;
            gotalias = true;
          }
        }
      }

      if(!gotalias)
      {
        for (int j=0;j<Ndigi-2;j++)
        {
          if((digi[j][0]== 'A') && (digi[j][1]== 'P') &&
(digi[j][2]== 'R') && (digi[j][3]== 'S')&& (digi[j][4]==
'A')&& (digi[j][5]== 'T'))
            {
              digiSSID[j] |= 0x80;
              gotalias = true;
            }
        }
      }

      if(gotalias){
        Serial2.println("Digipeated");
        Serial1.println("AT+DMOSETGROUP=1,144.5000,144.
5000,0000,1,0000");
        int delayCnt = 0;
        while(delayCnt<2000)
        Serial2.println(delayCnt++);
        APRS_TX();
        Serial1.println("AT+DMOSETGROUP=1,144.8000,144.
5000,0000,1,0000");
      }
   //   if((dest[0]== 'M') && (dest[1]== 'Y') && (dest[2]==
'C') && (dest[3]== 'A') && (dest[4]== 'L')&& (dest[5]==
'L'))
      if((dest[0]== 'S') && (dest[1]== 'J') && (dest[2]==
'C') && (dest[3]== 'S') && (dest[4]== 'A')&& (dest[5]==
'T'))
```

```
    QSL_tx();

    Serial2.println("Got here");
  startTX=0;
 }
if (timeCount>60)
 {
  startTX=1;
  beaconId++;
  if(beaconId>22) beaconId=1;
  Serial2.println(beaconId);
  switch(beaconId)
  {
    case 6: Audio_tx(); break;
    case 1: SSTV_tx(0,0x000000); break;
    case 4: SSTV_tx(1,0x09C000); break;
    case 8: SSTV_tx(0,0x138000); break;
    case 10: SSTV_tx(0,0x1D4000); break;
    case 12: SSTV_tx(0,0x270000); break;
    case 14: SSTV_tx(1,0x30C000); break;
    case 16: SSTV_tx(0,0x3A8000); break;
    case 18: SSTV_tx(1,0x444000); break;
    case 20: SSTV_tx(0,0x4E0000); break;
    case 22: SSTV_tx(1,0x57C000); break;

    case 2:
    case 3:
    case 5:
    case 7:
    case 9:
    case 19:
    case 21:
    TM_APRS(); break;
```

```
        //case 4: memAddress = 0x160000; SSTV_tx();
timeCount=0; break;
       // case 5: memAddress = 0x200000; SSTV_tx();
timeCount=0; break;
       // case 6: memAddress = 0x2A0000; SSTV_tx();
timeCount=0; break;
      }
      startTX=0;
      timeCount = 0;
    }
  }
}
```