```
//Software for Zachtek for the WSPR-TX_LP1 product with product number 1011.
This board is also known as "WSPR Transmitter Experimenters version"
//There are two different flawors of firmware for this board.
//1: This code with hardcoded Callsign and Power data that just transmit on one
single band. It is smaller in size and simpler in many ways so for
//   experimenters that want to modify the code, expand, add sensors etc etc,
this might be easier to use.
//2: The regular version that is designed to be used with a PC software to set
all the data like callsign, power level, band hopping etc
//   To acheive this it uses a Serial API and stores settings in EEPROM. It can
do more but it is much bigger and can be hard to understand for the beginner.
//The Version of this software is stored in the constant "softwareversion" and is
displayed on the Serialport at startup
//
//To compile you need to install several libraries, se below in the #include
section what they are and download them using The Library manager in the Arduino
IDE or use the direct URL
//For Arduino Pro Mini 3.3V 8MHz.
//
//  Version History
//  1.05 initial Release
//  1.06 Disabled transmission if callsign was not changed (variable "call[]" on
row 115 sets callsign), added TXPause variable to set duty cycle of transmission.
//  1.07 Added more predefined frequency bands every band from 2190m to 4m are
now defined
//  1.08pk modified for 2 band selection via toggle switch on D6
//
//The WSPR coding is based on Jason Mildrums example code. See his original
Copyright text below the line.
//Harry "deBug" Zachrisson of ZachTek 2018
//-------------------------------------------------------------------
// WSPR beacon for Arduino, by Jason Milldrum NT7S.
// Original code based on Feld Hell beacon for Arduino by Mark
// Vandewettering K6HX, adapted for the Si5351A by Robert
// Liesenfeld AK6L <ak6l@ak6l.org>.
//
// Permission is hereby granted, free of charge, to any person obtaining
// a copy of this software and associated documentation files (the
// "Software"), to deal in the Software without restriction, including
// without limitation the rights to use, copy, modify, merge, publish,
// distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so, subject
// to the following conditions:
// The above copyright notice and this permission notice shall be
// included in all copies or substantial portions of the Software.

#include <TinyGPS++.h>      //TinyGPS++ library by Mikal Hart https://github.
com/mikalhart/TinyGPSPlus
#include <JTEncode.h>       //JTEncode  by NT7S https://github.
```

```
com/etherkit/JTEncode
#include <SoftwareSerial.h>  //Arduino SoftSerial

void i2cInit();
uint8_t i2cSendRegister(uint8_t reg, uint8_t data);
uint8_t i2cReadRegister(uint8_t reg, uint8_t *data);

#define I2C_START 0x08
#define I2C_START_RPT 0x10
#define I2C_SLA_W_ACK 0x18
#define I2C_SLA_R_ACK 0x40
#define I2C_DATA_ACK 0x28
#define I2C_WRITE 0b11000000
#define I2C_READ 0b11000001
#define SI5351A_H

#define SI_CLK0_CONTROL 16 // Register definitions
#define SI_CLK1_CONTROL 17
#define SI_CLK2_CONTROL 18
#define SI_SYNTH_PLL_A 26
#define SI_SYNTH_PLL_B 34
#define SI_SYNTH_MS_0 42
#define SI_SYNTH_MS_1 50
#define SI_SYNTH_MS_2 58
#define SI_PLL_RESET 177

#define SI_R_DIV_1 0b00000000 // R-division ratio definitions
#define SI_R_DIV_2 0b00010000
#define SI_R_DIV_4 0b00100000
#define SI_R_DIV_8 0b00110000
#define SI_R_DIV_16 0b01000000
#define SI_R_DIV_32 0b01010000
#define SI_R_DIV_64 0b01100000
#define SI_R_DIV_128 0b01110000

#define SI_CLK_SRC_PLL_A 0b00000000
#define SI_CLK_SRC_PLL_B 0b00100000

#define XTAL_FREQ 24999990 // Crystal frequency for a certain board in Hertz,
change to fit your induvidual TCXO
//#define XTAL_FREQ 26000003 // Crystal frequency for a certain board in Hertz,
change to fit your induvidual TCXO

//defines for TX Frequency of WSPR on the HAM bands
#define WSPR_TONE_SPACING       146          // ~1.46 Hz  Tone spacng in centiHz
#define WSPR_DELAY              683         // Delay value for WSPR  delay in
milliseconds

#define WSPR_FREQ4m             7009250000ULL   //4m     70.092,500MHz
```

```
#define WSPR_FREQ6m            5029450000ULL   //6m    50.294,500MHz
#define WSPR_FREQ10m           2812610000ULL   //10m   28.126,100MHz
#define WSPR_FREQ12m           2492610000ULL   //12m   24.926,100MHz
#define WSPR_FREQ15m           2109610000ULL   //15m   21.096.100MHz
#define WSPR_FREQ17m           1810610000ULL   //17m   18.106,100MHz
#define WSPR_FREQ20m           1409710000ULL   //20m   14.097,100MHz
#define WSPR_FREQ30m           1014020000ULL   //30m   10.140,200MHz
#define WSPR_FREQ40m            704010000ULL   //40m    7.040,100MHz
#define WSPR_FREQ60m            528870000ULL   //60m    5.288,700MHz
#define WSPR_FREQ60mEU          536620000ULL   //60mEU  5.366,200MHz
#define WSPR_FREQ80m            359410000ULL   //80m    3.394,100MHz
#define WSPR_FREQ160m           183810000ULL   //160m   1.838,100MHz
#define WSPR_FREQ630m            47570000ULL   //630m     475.700kHz
#define WSPR_FREQ2190m           13750000ULL   //2190m    137.500kHz


// Hardware defines
#define StatusLED_Yellow 4  //Yellow LED to indicator that blinks at different
rates depending on GPS Lock, counting down to next TX
#define TransmitLED_RED 8   //Red LED next to RF out SMA that will turn on when
Transmitting

const char softwareversion[] = "1.08pk 40m / 30m" ; //Version of this program,
sent to serialport at startup

// Class instantiation
JTEncode jtencode;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial GPSSerial(2, 3); //GPS Serial port, RX on pin 2, TX on pin 3

void si5351aOutputOff(uint8_t clk);
void si5351aSetFrequency(uint32_t frequency);

//************************************************************************
*************
//--------- Set Transmit Frequency and Amateur callsign and TX Duty Cyle here
-----------------
//************************************************************************
*************
char call[] = "AA0BBB";                          //**********Set your call sign
here **************
uint64_t WSPR_DEFAULT_FREQ = WSPR_FREQ40m;   // Initial Band - Set this to one of
the defined WSPR band constants, or set to any frequency in centiHertz
uint64_t WSPR_DEFAULT_FREQ_Last = 0; // used in Auto toggle band
const uint64_t WSPR_BAND_1 = WSPR_FREQ40m; // Band 1 - Set this to one of the
```

```
defined WSPR band constants, or set to any frequency in centiHertz
const uint64_t WSPR_BAND_2 = WSPR_FREQ30m; // Band 2 - Set this to one of the
defined WSPR band constants, or set to any frequency in centiHertz
boolean WSPRband = 0; // used for toggling bands in AUTO mode
boolean WSPRband_Last = 0; // used for toggling bands in AUTO mode

uint8_t dbm = 23;                                   //Output power in dBm. The
WSPR-TX gives out about 24dBm after Low Pass filtering so 23 is closest value in
WSPR coding, change this if you have an attenuater or amplifier connected
unsigned long TXPause = 2000; //Pause to set TX duty cycle. 2000=100%, 20000=50%,
130000=33%
char loc[] =  "AB01";         //If you dont want to use the GPS provided locator
you can hard code a locator here, 4 chars only. You will still need the GPS for
timing - Only used if LocatorByGPS=false
boolean LocatorByGPS = true; //Set to false to disable automatic location by GPS
and switch to use the "loc" value
boolean Seeded = false; // flag to prevent reseeding random offset
boolean ShowClock = false; // set to false to stop clock updates to terminal
screen
boolean ShowDots = false; // set to True to show waiting for TX dots on terminal
screen
int BandSwitch = 0;
int BandSwitch_Old = 5;
int FreqToggle = 0;

//''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''
char MHLocator[] = "AA68BB";// Don't change, this is only here to allocate
memory, will be changed by the GPS position and Maidenhead calculations.
uint8_t tx_buffer[180];
uint8_t symbol_count;
uint16_t tone_delay, tone_spacing;
uint64_t freq = WSPR_DEFAULT_FREQ;
uint64_t freq1 = WSPR_BAND_1;
uint64_t freq2 = WSPR_BAND_2;

static double Lat;
static double Lon;
static int GPSH;
static int GPSM;
static int GPSS;

void setup()
{
  // Set the proper frequency, tone spacing, symbol count, and
  // tone delay depending on mode
  // freq = WSPR_DEFAULT_FREQ;
  symbol_count = WSPR_SYMBOL_COUNT; // From the library defines
  tone_spacing = WSPR_TONE_SPACING;
```

```
 tone_delay = WSPR_DELAY;

 //bool i2c_found;
 //Initialize Arduinos Serial port
 Serial.begin(115200);
 delay(500);// Wait for Serialport to be initialized properly
 GPSSerial.begin(9600); // The GPS Serial port uses 9600
 //Output Initialization text on the serial port

 Serial.print(F("Zachtek WSPR-TX_LP1 -Hard Coded Info- Software version: "));
 Serial.println(softwareversion);
 Serial.print(F("Callsign set to "));
 Serial.println(call);
 Serial.println(F("Maidenhead postion set by GPS"));
 Serial.print(F("Power Data set to "));
 Serial.print(dbm);
 Serial.println(F("dBm"));
 Serial.print (F("WSPR Frequency #1 set to "));
 Serial.print (uint64ToStr((freq1 / 100ULL), false));
 Serial.println(F("Hz with a +/-100Hz random offset on every TX"));

 Serial.print (F("WSPR Frequency #2 set to "));
 Serial.print (uint64ToStr((freq2 / 100ULL), false));
 Serial.println(F("Hz with a +/-100Hz random offset on every TX"));

 Serial.println("Toggle Switch Selects  #1 / AUTO_Change / #2");

 //  Serial.println(F("Initializing.."));
 pinMode(TransmitLED_RED, OUTPUT);
 pinMode(StatusLED_Yellow, OUTPUT);
 pinMode(6, INPUT_PULLUP); // Toggle Switch # 1 - Band Change
 pinMode(7, INPUT_PULLUP); // Toggle Switch # 2 - Auto Frequency Hop

 Serial.print(F("Initializing I2C Interface "));
 i2cInit();
 Serial.println(F("and Si5351 Synth"));
 si5351aSetFrequency(2000000000ULL);

 si5351aOutputOff(SI_CLK0_CONTROL);



 // Encode the message in the transmit buffer
 // This is RAM intensive and should be done separately from other subroutines
 //set_tx_buffer();
 Serial.print(F("Waiting for GPS location fix."));
}

void loop()
{
```

```
 while (GPSSerial.available()) {
   gps.encode(GPSSerial.read());
   if (gps.location.isUpdated())
    {
      GPSH = gps.time.hour();
      GPSM = gps.time.minute();
      GPSS = gps.time.second();
      if (LocatorByGPS)
      {
        Lat = gps.location.lat();
        // Serial.println(F("Lat set"));
        Lon = gps.location.lng();
        Serial.println(" ");
        Serial.print(F("GPS Lat "));
        Serial.print (Lat);
        Serial.print(", Long ");
        Serial.print (Lon);
        calcLocator (Lat, Lon);
        Serial.print (F("  Maidenhead locator is "));
        Serial.print(MHLocator);
       loc[0] = MHLocator[0]; loc[1] = MHLocator[1]; loc[2] = MHLocator[2];
loc[3] = MHLocator[3];
        Serial.print (F(" (using "));
        Serial.print(loc);
        Serial.println (F(")"));
        LocatorByGPS = false; // do above code once, unless told to update it
        if (!Seeded) {
          long seed = (long) (((long)Lat * 10000) + ((long)Lon * 10000));
          randomSeed(seed);
          Seeded = true; // done, so do not re-seed
        }
        // Encode the message in the transmit buffer so we are prepared to
transmit
        // This is RAM intensive and should be done separately from other
subroutines
        set_tx_buffer();
        TimeStamp();
       Serial.println (F(" Waiting for even Minute"));
      }

     if ((GPSS == 0) && ((GPSM % 2) == 0))//If second is zero at even minute
then start WSPR transmission
      {
        TimeStamp();
        Serial.print(F(" WSPR TX for 110 Seconds on "));
        freq = WSPR_DEFAULT_FREQ + (100ULL * random (-100, 100));
       Serial.print(uint64ToStr(freq / 100, false));
        Serial.println(F("Hz"));
        encode ();
```

```
       Serial.print(F("Pause for ~"));
       Serial.print(TXPause / 1000);
       Serial.println(F(" Seconds for TX duty cycle."));
       smartdelay(TXPause);//Pause for some time to give a duty cycle on the
transmit. 2000=100%, 20000=50%, 130000=33%

      }
     else //Double-blink to indicate waiting for top of next even minute
      {
       checkBandSwitch(); // Read Band Switch, and check if Auto Freq change is
selected

       if (ShowClock) {
         TimeStamp();
       }
       if (ShowDots) {
         Serial.print('.');
         if (GPSS >= 59) { // wrap dots to new line at end of minute
           Serial.println(' ');
         }
       }
      digitalWrite(StatusLED_Yellow, HIGH);
       smartdelay (100);
      digitalWrite(StatusLED_Yellow, LOW);
       smartdelay (100);
      digitalWrite(StatusLED_Yellow, HIGH);
       smartdelay (100);
      digitalWrite(StatusLED_Yellow, LOW);
       smartdelay(300);
     }
    }

   if (gps.location.isValid())
    {
     // if (gps.satellites.value() != 0)// digitalWrite(StatusLED_Yellow, HIGH);
// turn the yellow LED on
     if (gps.satellites.value() == 0) {
       Serial.println("Check GPS!");
       // Slow Blink ON/OFF to say Check GPS !
       digitalWrite(StatusLED_Yellow, HIGH);   // turn the yellow LED on
       smartdelay(400);
       digitalWrite(StatusLED_Yellow, LOW);    // turn the yellow LED off
       smartdelay(400);
      }
    }
    else
    {
     //quick blink to indicate waiting to Aquire GPS
     digitalWrite(StatusLED_Yellow, HIGH);   // turn the yellow LED on
```

```
      smartdelay(300);
      digitalWrite(StatusLED_Yellow, LOW);    // turn the yellow LED off
      Serial.print(".");
      smartdelay(100);
    }
  }
}
// TimeStamp for Serial Monitor
void TimeStamp() {
  GPSH = gps.time.hour();
  GPSM = gps.time.minute();
  GPSS = gps.time.second();
  if (GPSH <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSH); Serial.print (F(":"));
  if (GPSM <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSM); Serial.print (F(":"));
  if (GPSS <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSS);
}
// requires manual patching for bands used / announcments, but works...
void checkBandSwitch() {
  int BandSwitch = (digitalRead(6) * 2) + digitalRead(7);
  if (BandSwitch != BandSwitch_Old) {
    BandSwitch_Old = BandSwitch;
    Serial.print("Next WSPR TX on ");
    switch (BandSwitch) {
      case 1: {
          Serial.println("40m");
          WSPR_DEFAULT_FREQ = WSPR_BAND_1;
          WSPRband = 0;
          FreqToggle = 0;
          break;
        }
      case 2: {
          Serial.println("30m");
          WSPR_DEFAULT_FREQ = WSPR_BAND_2;
          WSPRband = 1;
          FreqToggle = 0;
          break;
        }
      case 3: {
          Serial.println("Auto 30/40m");
       //   WSPRband = 0; // start new toggle with band WSPR_BAND_1
          FreqToggle = 1; // enable frequency toggle at the end of each TX block
          break;
        }
    }
  }
}
```

```
void setBand() {
  if (FreqToggle == 1) {
    WSPRband = !WSPRband; // don't use last band again !
    Serial.print("Frequency toggle, next WSPR TX on ");
    switch (WSPRband) {
      case 0: {
          Serial.println("40m");
          WSPR_DEFAULT_FREQ = WSPR_BAND_1;
           break;
        }
      case 1: {
          Serial.println("30m");
          WSPR_DEFAULT_FREQ = WSPR_BAND_2;
           break;
        }
    }
  }
}
// Loop through the string, transmitting one character at a time.
void encode()
{
  uint8_t i;
  unsigned long startmillis;
  unsigned long endmillis;
  boolean TXEnabled = true;

  // Send WSPR for two minutes
  // digitalWrite(StatusLED_Yellow, HIGH); // yellow LED
  if ((call[0] == 'A') && (call[1] == 'A') && (call[2] == '0') && (call[3] ==
'A') && (call[4] == 'A') && (call[5] == 'A')) //Do not actually key the
transmitter if the callsign has not been changed from the default one AA0AAA
  {
    Serial.println(F("Callsign is not changed from the default one, Transmit is
disabled"));
    Serial.println(F("Recompile this software with your Callsign to enable
transmission"));
    TXEnabled = false;
  }



  startmillis = millis();
  for (i = 0; i < symbol_count; i++)
  {
    endmillis = startmillis + ((i + 1) * (unsigned long) tone_delay) ;
    uint64_t tonefreq;
    tonefreq = freq + ((tx_buffer[i] * tone_spacing));
   if (TXEnabled) si5351aSetFrequency(tonefreq);
    //wait until tone is transmitted for the correct amount of time
```

```
    while (millis() < endmillis) {
      //just wait
    }
  }

  // Turn off the output
  // Switches off Si5351a output
 si5351aOutputOff(SI_CLK0_CONTROL);
 digitalWrite(StatusLED_Yellow, LOW);

  GPSS = 50; // GPS not read during TX, so fudge next clock time stamp after TX
period
  GPSM ++;
  if (GPSM >= 60) {
    GPSM = 0;
    GPSH++;
    if (GPSH >= 24) GPSH = 0;
  }

  if (GPSH <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSH); Serial.print (F(":"));
  if (GPSM <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSM); Serial.print (F(":"));
  if (GPSS <= 9) Serial.print('0'); //add leading zero
  Serial.print (GPSS);
  Serial.print(F(" TX Off, "));
  LocatorByGPS = true; // Request updated GPS Data for next transmission

  setBand(); // Check to see if we need to AUTO Toggle to other frequency
}

void set_tx_buffer()
{
  // Clear out the transmit buffer
 memset(tx_buffer, 0, sizeof(tx_buffer));
  jtencode.wspr_encode(call, loc, dbm, tx_buffer);
}



//Maidenhead code from Ossi Väänänen https://ham.stackexchange.
com/questions/221/how-can-one-convert-from-lat-long-to-grid-square
void calcLocator(double lat, double lon) {
  int o1, o2, o3;
  int a1, a2, a3;
  double remainder;
  // longitude
  remainder = lon + 180.0;
  o1 = (int)(remainder / 20.0);
```

```
  remainder = remainder - (double)o1 * 20.0;
  o2 = (int)(remainder / 2.0);
  remainder = remainder - 2.0 * (double)o2;
  o3 = (int)(12.0 * remainder);

  // latitude
  remainder = lat + 90.0;
  a1 = (int)(remainder / 10.0);
  remainder = remainder - (double)a1 * 10.0;
  a2 = (int)(remainder);
  remainder = remainder - (double)a2;
  a3 = (int)(24.0 * remainder);
  MHLocator[0] = char(o1 + 65);
  MHLocator[1] = char(a1 + 65);
  MHLocator[2] = char(o2 + 48);
  MHLocator[3] = char(a2 + 48);
  MHLocator[4] = char(o3 + 65);
  MHLocator[5] = char(a3 + 65);
}


// This custom version of delay() ensures that the gps object
// is being "fed".
static void smartdelay(unsigned long ms)
{
  unsigned long start = millis();
  do
  {
    while (GPSSerial.available())
      gps.encode(GPSSerial.read());
  } while (millis() - start < ms);
}


uint8_t i2cStart()
{
  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT))) ;
  return (TWSR & 0xF8);
}

void i2cStop()
{
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
  while ((TWCR & (1 << TWSTO))) ;
}

uint8_t i2cByteSend(uint8_t data)
{
```

```
  TWDR = data;
  TWCR = (1 << TWINT) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT))) ;
  return (TWSR & 0xF8);
}

uint8_t i2cByteRead()
{
  TWCR = (1 << TWINT) | (1 << TWEN);
  while (!(TWCR & (1 << TWINT))) ;
  return (TWDR);
}

uint8_t i2cSendRegister(uint8_t reg, uint8_t data)
{
  uint8_t stts;

  stts = i2cStart();
  if (stts != I2C_START) return 1;

  stts = i2cByteSend(I2C_WRITE);
  if (stts != I2C_SLA_W_ACK) return 2;

  stts = i2cByteSend(reg);
  if (stts != I2C_DATA_ACK) return 3;

  stts = i2cByteSend(data);
  if (stts != I2C_DATA_ACK) return 4;

  i2cStop();
  return 0;
}

uint8_t i2cReadRegister(uint8_t reg, uint8_t *data)
{
  uint8_t stts;

  stts = i2cStart();
  if (stts != I2C_START) return 1;

  stts = i2cByteSend(I2C_WRITE);
  if (stts != I2C_SLA_W_ACK) return 2;

  stts = i2cByteSend(reg);
  if (stts != I2C_DATA_ACK) return 3;

  stts = i2cStart();
  if (stts != I2C_START_RPT) return 4;
```

```
  stts = i2cByteSend(I2C_READ);
  if (stts != I2C_SLA_R_ACK) return 5;

  *data = i2cByteRead();

  i2cStop();

  return 0;
}

// Init TWI (I2C)
//
void i2cInit()
{
  TWBR = 92;
  TWSR = 0;
  TWDR = 0xFF;
  PRR = 0;
}

//
// Set up specified PLL with mult, num and denom
// mult is 15..90
// num is 0..1,048,575 (0xFFFFF)
// denom is 0..1,048,575 (0xFFFFF)
//
void setupPLL(uint8_t pll, uint8_t mult, uint32_t num, uint32_t denom)
{
  uint32_t P1; // PLL config register P1
  uint32_t P2; // PLL config register P2
  uint32_t P3; // PLL config register P3

  P1 = (uint32_t)(128 * ((float)num / (float)denom));
  P1 = (uint32_t)(128 * (uint32_t)(mult) + P1 - 512);
  P2 = (uint32_t)(128 * ((float)num / (float)denom));
  P2 = (uint32_t)(128 * num - denom * P2);
  P3 = denom;

  i2cSendRegister(pll + 0, (P3 & 0x0000FF00) >> 8);
  i2cSendRegister(pll + 1, (P3 & 0x000000FF));
  i2cSendRegister(pll + 2, (P1 & 0x00030000) >> 16);
  i2cSendRegister(pll + 3, (P1 & 0x0000FF00) >> 8);
  i2cSendRegister(pll + 4, (P1 & 0x000000FF));
  i2cSendRegister(pll + 5, ((P3 & 0x000F0000) >> 12) | ((P2 &
                  0x000F0000) >> 16));
  i2cSendRegister(pll + 6, (P2 & 0x0000FF00) >> 8);
  i2cSendRegister(pll + 7, (P2 & 0x000000FF));
}
```

```
//
// Set up MultiSynth with integer Divider and R Divider
// R Divider is the bit value which is OR'ed onto the appropriate
// register, it is a #define in si5351a.h
//
void setupMultisynth(uint8_t synth, uint32_t Divider, uint8_t rDiv)
{
  uint32_t P1; // Synth config register P1
  uint32_t P2; // Synth config register P2
  uint32_t P3; // Synth config register P3

  P1 = 128 * Divider - 512;
  P2 = 0; // P2 = 0, P3 = 1 forces an integer value for the Divider
  P3 = 1;

  i2cSendRegister(synth + 0, (P3 & 0x0000FF00) >> 8);
  i2cSendRegister(synth + 1, (P3 & 0x000000FF));
  i2cSendRegister(synth + 2, ((P1 & 0x00030000) >> 16) | rDiv);
  i2cSendRegister(synth + 3, (P1 & 0x0000FF00) >> 8);
  i2cSendRegister(synth + 4, (P1 & 0x000000FF));
  i2cSendRegister(synth + 5, ((P3 & 0x000F0000) >> 12) | ((P2 &
                  0x000F0000) >> 16));
  i2cSendRegister(synth + 6, (P2 & 0x0000FF00) >> 8);
  i2cSendRegister(synth + 7, (P2 & 0x000000FF));
}

//
// Switches off Si5351a output
// Example: si5351aOutputOff(SI_CLK0_CONTROL);
// will switch off output CLK0
//
void si5351aOutputOff(uint8_t clk)
{
 digitalWrite(TransmitLED_RED, LOW);
  i2cSendRegister(clk, 0x80); // Refer to SiLabs AN619 to see
  //bit values - 0x80 turns off the output stage
}



//
// Set CLK0 output ON and to the specified frequency
// Frequency is in the range 10kHz to 150MHz and given in centiHertz (hundredths
of Hertz)
// Example: si5351aSetFrequency(1000000200);
// will set output CLK0 to 10.000,002MHz
//
// This example sets up PLL A
// and MultiSynth 0
// and produces the output on CLK0
```

```
//
void si5351aSetFrequency(uint64_t frequency) //Frequency is in centiHz
{
  static uint64_t oldFreq;
  int32_t FreqChange;
  uint64_t pllFreq;
  uint32_t xtalFreq = XTAL_FREQ;
  uint32_t l;
  float f;
  uint8_t mult;
  uint32_t num;
  uint32_t denom;
  uint32_t Divider;
  uint8_t rDiv;

 digitalWrite(TransmitLED_RED, HIGH);
  // Serial.print (F("Freq is="));
  // Serial.println (uint64ToStr(frequency,false));
  if (frequency > 100000000) { //If higher than 1MHz then set output divider to 1
    rDiv = SI_R_DIV_1;
    Divider = 90000000000ULL / frequency;// Calculate the division ratio. 900MHz
is the maximum internal (expressed as deciHz)
    //Serial.print (F("Divider="));
    //Serial.println (Divider);
    pllFreq = Divider * frequency; // Calculate the pllFrequency:
    //the Divider * desired output frequency
    //Serial.print (F("pllFreq="));
   //Serial.println (uint64ToStr(pllFreq,false));
    mult = pllFreq / (xtalFreq * 100UL); // Determine the multiplier to
    //Serial.print (F("mult="));
    //Serial.println (mult);

    //get to the required pllFrequency
    l = pllFreq % (xtalFreq * 100UL); // It has three parts:
    //Serial.print (F("l="));
    //Serial.println (l);
    f = l; // mult is an integer that must be in the range 15..90
    //Serial.print (F("f="));
    //Serial.println (f);
    f *= 1048575; // num and denom are the fractional parts, the numerator and
denominator
    //Serial.print (F("f="));
    //Serial.println (f);
    f /= xtalFreq; // each is 20 bits (range 0..1048575)
    //Serial.print (F("f="));
    //Serial.println (f);

    num = f; // the actual multiplier is mult + num / denom
    //Serial.print (F("num="));
```

```
   //Serial.println (num);
   denom = 1048575; // For simplicity we set the denominator to the maximum
1048575
   num = num / 100;
  }
 else // lower freq than 1MHz - use output Divider set to 128
 {
   rDiv = SI_R_DIV_128;
   frequency = frequency * 128ULL; //Set base freq 128 times higher as we are
dividing with 128 in the last output stage
   Divider = 90000000000ULL / frequency;// Calculate the division ratio. 900,000,
000 is the maximum internal

   pllFreq = Divider * frequency; // Calculate the pllFrequency:
   //the Divider * desired output frequency
   mult = pllFreq / (xtalFreq * 100UL); // Determine the multiplier to
   //get to the required pllFrequency
   l = pllFreq % (xtalFreq * 100UL); // It has three parts:
   //Serial.print (F("l="));
   //Serial.println (l);
   f = l; // mult is an integer that must be in the range 15..90
   //Serial.print (F("f="));
   //Serial.println (f);
   f *= 1048575; // num and denom are the fractional parts, the numerator and
denominator
   //Serial.print (F("f="));
   //Serial.println (f);
   f /= xtalFreq; // each is 20 bits (range 0..1048575)
   //Serial.print (F("f="));
   //Serial.println (f);

   num = f; // the actual multiplier is mult + num / denom
   //Serial.print (F("num="));
   //Serial.println (num);
   denom = 1048575; // For simplicity we set the denominator to the maximum
1048575
   num = num / 100;
 }


 // Set up PLL A with the calculated  multiplication ratio
setupPLL(SI_SYNTH_PLL_A, mult, num, denom);

 // Set up MultiSynth Divider 0, with the calculated Divider.
 // The final R division stage can divide by a power of two, from 1..128.
 // reprented by constants SI_R_DIV1 to SI_R_DIV128 (see si5351a.h header file)
 // If you want to output frequencies below 1MHz, you have to use the
 // final R division stage
setupMultisynth(SI_SYNTH_MS_0, Divider, rDiv);
```

```
  // Reset the PLL. This causes a glitch in the output. For small changes to
  // the parameters, you don't need to reset the PLL, and there is no glitch
  FreqChange = frequency - oldFreq;

  if ( abs(FreqChange) > 100000) //If changed more than 1kHz then reset PLL
(completely arbitrary choosen)
  {
   i2cSendRegister(SI_PLL_RESET, 0xA0);
  }

  // Finally switch on the CLK0 output (0x4F)
  // and set the MultiSynth0 input to be PLL A
 i2cSendRegister(SI_CLK0_CONTROL, 0x4F | SI_CLK_SRC_PLL_A);
  oldFreq = frequency;
}

String  uint64ToStr (uint64_t p_InNumber, boolean p_LeadingZeros)
{
  char l_HighBuffer[7]; //6 digits + null terminator char
  char l_LowBuffer[7]; //6 digits + null terminator char
  char l_ResultBuffer [13]; //12 digits + null terminator char
  String l_ResultString = "";
  uint8_t l_Digit;

  sprintf(l_HighBuffer, "%06lu", p_InNumber / 1000000L); //Convert high part of
64bit unsigned integer to char array
  sprintf(l_LowBuffer, "%06lu", p_InNumber % 1000000L); //Convert low part of
64bit unsigned integer to char array
  l_ResultString = l_HighBuffer;
  l_ResultString = l_ResultString + l_LowBuffer; //Copy the 2 part result to a
string

  if (!p_LeadingZeros) //If leading zeros should be romeved
  {
   l_ResultString.toCharArray(l_ResultBuffer, 13);
    for (l_Digit = 0; l_Digit < 12; l_Digit++ )
    {
     if (l_ResultBuffer[l_Digit] == '0')
      {
       l_ResultBuffer[l_Digit] = ' '; // replace zero with a space character
      }
      else
      {
       break; //We have found all the leading Zeros, exit loop
      }
    }
   l_ResultString = l_ResultBuffer;
   l_ResultString.trim();//Remove all leading spaces
```

```
  }
  return l_ResultString;
}

static void printFloat(float val, bool valid, int len, int prec)
{
  if (!valid)
  {
    while (len-- > 1)
      Serial.print('*');
    Serial.print(' ');
  }
  else
  {
    Serial.print(val, prec);
    int vi = abs((int)val);
    int flen = prec + (val < 0.0 ? 2 : 1); // . and -
    flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
    for (int i = flen; i < len; ++i)
      Serial.print(' ');
  }
  smartdelay(0);
}

static void printInt(unsigned long val, bool valid, int len)
{
  char sz[32] = "*****************";
  if (valid)
    sprintf(sz, "%ld", val);
  sz[len] = 0;
  for (int i = strlen(sz); i < len; ++i)
    sz[i] = ' ';
  if (len > 0)
    sz[len - 1] = ' ';
 Serial.print(sz);
  smartdelay(0);
}

static void printDateTime(TinyGPSDate & d, TinyGPSTime & t)
{
  if (!d.isValid())
  {
    Serial.print(F("********** "));
  }
  else
  {
    char sz[32];
    sprintf(sz, "%02d/%02d/%02d ", d.month(), d.day(), d.year());
    Serial.print(sz);
```

```
  }

  if (!t.isValid())
  {
    Serial.print(F("******** "));
  }
  else
  {
    char sz[32];
    sprintf(sz, "%02d:%02d:%02d ", t.hour(), t.minute(), t.second());
    Serial.print(sz);
  }

  printInt(d.age(), d.isValid(), 5);
  smartdelay(0);
}

static void printStr(const char *str, int len)
{
  int slen = strlen(str);
  for (int i = 0; i < len; ++i)
    Serial.print(i < slen ? str[i] : ' ');
  smartdelay(0);
}
```