W3PM Universal MEPT

Modes: WSPR , FST4W, FT8, CW, QRSS
Timing: WiFi, GPS, RTC

Gene Marcus W3PM GM4YRE

This multifaceted Manned Experimental Propagation Transmitter (MEPT) uses an Arduino Nano IoT 33 and a Si5351A clock generator board to generate WSPR, FST4W, FT8, CW, or QRSS signals. Time accuracy is derived from either WiFi, GPS, or a DS3231N real time clock (RTC). Transmit times, modes of operation, and frequency are selected by a menu or by one of three schedules.

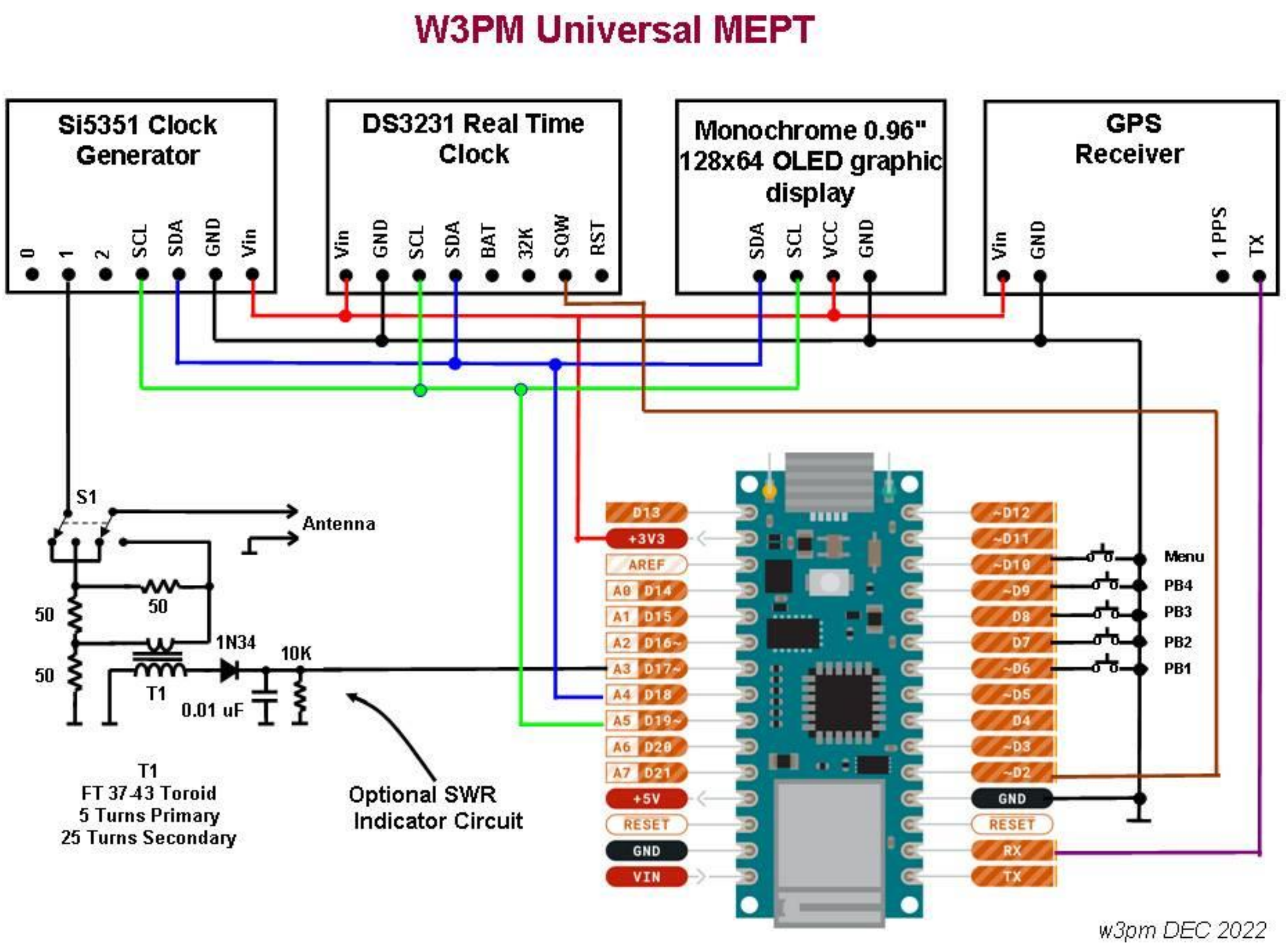

Figure 1
Wiring Diagram

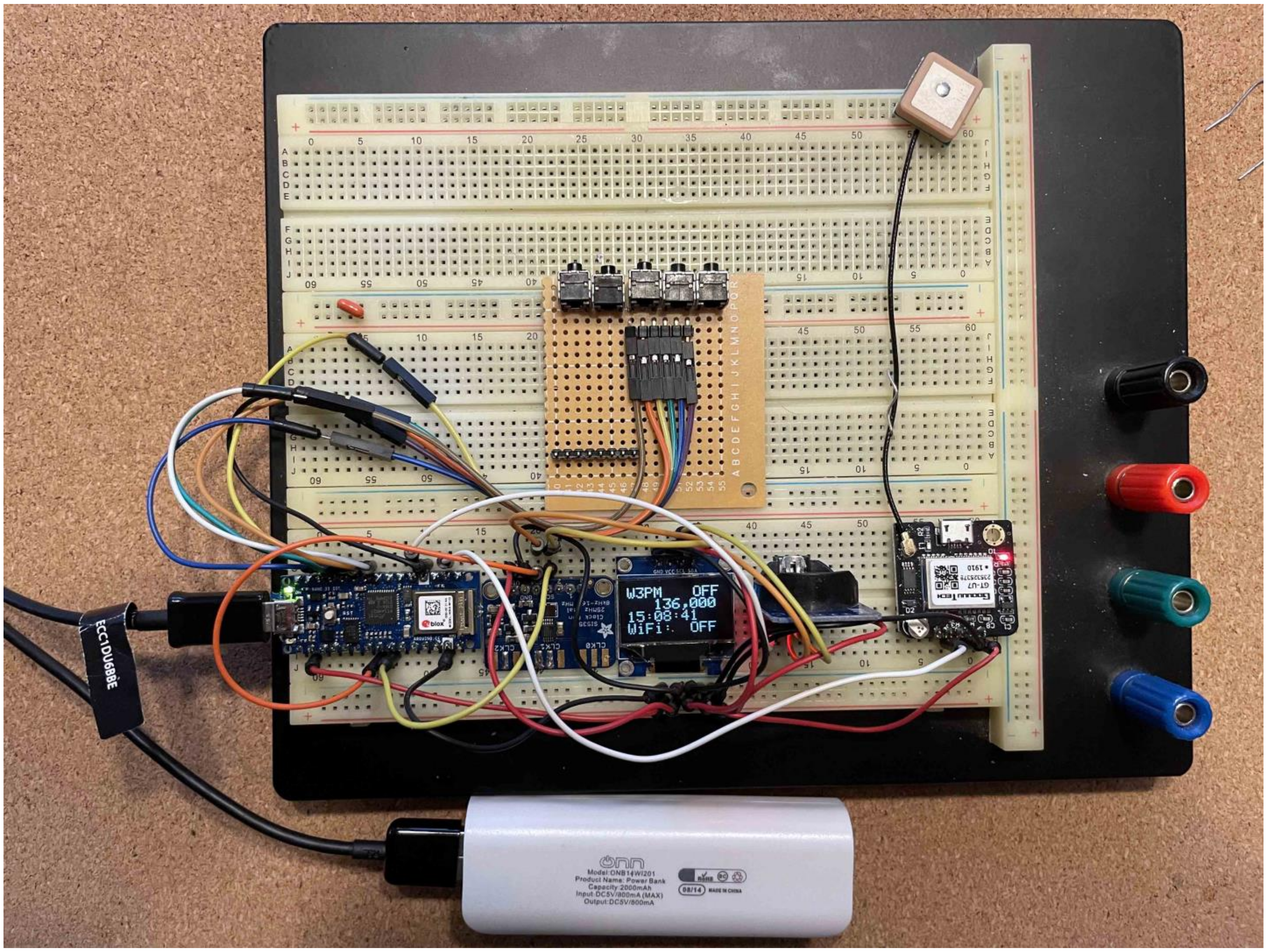

Figure 2
Breadboard

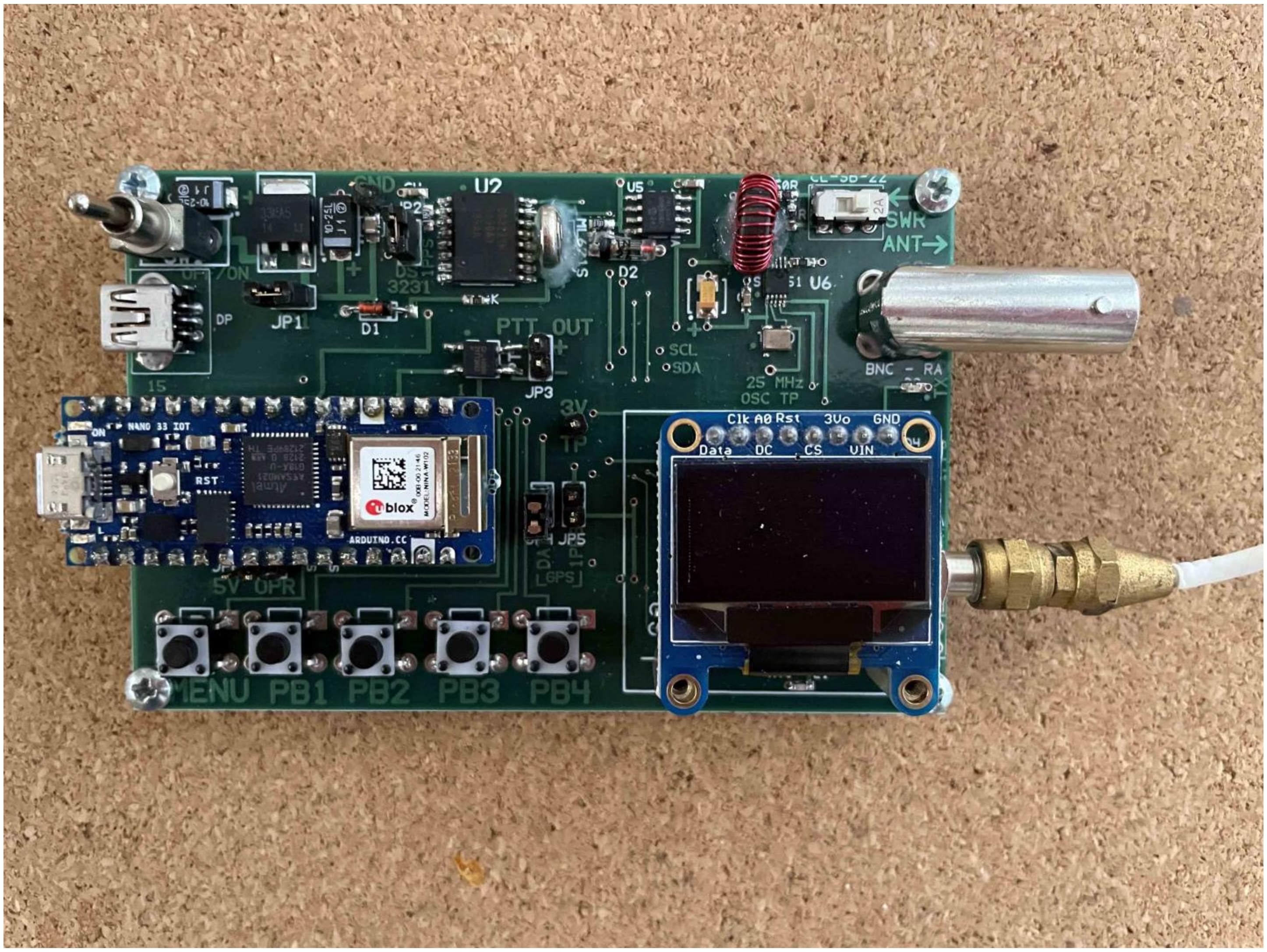

Figure 3
K4CLE board

Introduction

The Nano 33 IOT is NOT 5V tolerant!

NOTE: The Nano 33 IOT used in this project is a 3.3 volt device. Damage will occur if more than 3.3 V is applied to any of the input/output pins. The unit draws little power and can be powered by a USB cable connected to a computer, a USB charger, or by some 5V USB battery backup chargers. Alternately, you can use a 7-21VDC source connected to the Arduino Nano 33 IOT’s “VIN” pin. The 3.3 volt output pin “3V3” is capable of supplying 600 mA which easily powers all of the peripheral boards.

The Nano 33 IOT features WiFi connectivity, 448 KB of ROM, 520KB SRAM , 2MB Flash and a SAMD21 Cortex®-M0+ 32bit low power ARM MCU running at 48 MHz. This allows far more project flexibility to develop a multifaceted MEPT project.

This project is based individual circuit cards as shown in Figure 1. The building blocks for this project consist of an Arduino Nano 33 IOT microcontroller, a Si5351A clock generator breakout board, DS3231 real time clock (RTC) board , GPS receiver, and a 0.96 inch 128x64 serial I2C OLED display board. The Si535a, DS3231, and OLED boards communicate with the Nano 33 IOT microcontroller over an I2C bus using only three wires. The boards were chosen because they are highly capable, inexpensive, well documented, and available over the internet from a wide variety of vendors.

The GitHub files contain archival information about a circuit board developed by Doug, K4CLE (Figure 3). At this point in time he is unable to provide any populated or un-populated boards. He may be contacted at “doug at k4cle dot com” for any questions concerning the board.

This project relies heavily on timer accuracy derived from the Nano 33 IOT 48 MHz clock. Unfortunately, the board is not crystal controlled and relies on an LC oscillator. To overcome this shortcoming a correction factor is generated by comparing the millisecond timer to a 1pps output from the high accuracy DS3231 Real Time Clock.

EEPROM is not available on the Nano 33 IOT. The sketch for the project uses the FlashStorage_SAMD library to emulate EEPROM storage using flash memory. The K4CLE board uses an on board EEPROM IC so a modified sketch is included in the files.

Timing is derived from a DS3231 real time clock (RTC), GPS receiver, or WiFi. If WiFi or GPS timing is not selected or interrupted, the unit defaults to RTC timing. RTC timing is less accurate. The builder may anticipate an uncertainty of about 2 parts per million from 0 – 40 degrees C (32 to 104 degrees F) which will allow WSPR operation for about two weeks without correction. The GPS receiver may be omitted if the builder prefers WiFi or RTC timing.

All the necessary parameters required for operation can be changed by using 5 pushbuttons.

An antenna tune function is included. Although it will not provide the actual SWR reading, it will serve as an indicator to let you know when the antenna is tuned to a nominal 50 ohm impedance.

NOTE: Ensure your project uses the DS3231SN. Some vendors substitute the DS3231M version which is not as accurate as the “N” version.

Construction

Construction of the unit is not critical provided adequate RF techniques are used. Do not use long unshielded wires for RF connections. I discovered that The GPS reception will be adversely affected if the GPS receiver and/or GPS antenna is mounted too close to the Si5351. I first built and tested the system using a solderless breadboard (figure 2) without any problems. Mounting the unit in a metal cabinet with an external GPS antenna is the preferred way to house this project. To improve short and long term accuracy the unit should be kept at a constant temperature away from any drafts.

The Si5351A, GPS receiver, and the display modules are powered from the 3.3 volt pin of the Arduino. You can use the USB programming cable or a 7-21VDC source connected to the Arduino Nano 33 IOT’s “VIN” pin. Builders may require an external 3,3 V power supply if a different GPS receiver is used. I used the popular and inexpensive NEO-6M GPS module selling for about $10 USD on many internet sites. Ensure any substituted GPS receiver outputs the "$GPRMC" sentence at 9600 baud.

Software Installation and Setup

Before attempting to load the sketch the builder should check the Nano 33 IOT for proper operation. Some example sketches are provided with the open source Arduino software to test for Nano IOT 33 functionality. The simple “blink” example sketch will confirm that a sketch can be loaded and the Arduino board is functioning. The internet provides step-by-step instructions on how to set up the device.

Open the sketch with the Arduion IDE and replace the example WiFi, WSPR/FST4W/FT8, and CW/FSK configuration information with your customized configuration. The sketch is well commented to guide you through the changes. Don’t forget to save the sketch after you make the changes.

NOTE: The sketch contains two variables which hold unique data symbols pertaining to FST4W and FT8 operation. No action is required if the builder does not require FST4W or FT8 operation. The variables containing example data are shown below.

```
// Load FST4W channel symbols for WZ9ZZZ FN21 30dBm
const int  FST4Wsymbols[160] = {
  0, 1, 3, 2, 1, 0, 2, 3, 2, 3,
  3, 2, 3, 2, 3, 3, 0, 3, 1, 1,
  1, 2, 1, 1, 0, 0, 3, 3, 0, 3,
  3, 1, 0, 1, 3, 3, 0, 1, 2, 3,
  1, 0, 3, 2, 0, 1, 2, 0, 1, 0,
  3, 3, 1, 3, 3, 2, 3, 3, 0, 0,
  1, 2, 1, 1, 3, 0, 3, 2, 0, 3,
  2, 0, 3, 3, 0, 0, 0, 1, 3, 2,
  1, 0, 2, 3, 2, 2, 3, 1, 0, 2,
  2, 0, 2, 2, 2, 1, 0, 1, 0, 0,
  3, 1, 2, 0, 0, 3, 1, 0, 2, 2,
  1, 2, 2, 2, 2, 3, 1, 0, 3, 2,
  0, 1, 2, 1, 3, 2, 3, 3, 3, 2,
  1, 0, 1, 0, 2, 2, 3, 0, 2, 3,
  3, 0, 2, 0, 0, 1, 1, 3, 2, 1,
  1, 0, 0, 1, 3, 2, 1, 0, 2, 3
};
```

```
// Load FT8 channel symbols for "CQ WZ9ZZZ AA99"
const int FT8symbols[79] = {
  3, 1, 4, 0, 6, 5, 2, 0, 0, 0,
  0, 0, 0, 0, 0, 1, 1, 4, 6, 4,
  7, 3, 1, 1, 3, 7, 0, 0, 0, 2,
  0, 4, 3, 2, 1, 7, 3, 1, 4, 0,
  6, 5, 2, 0, 5, 7, 5, 1, 4, 0,
  7, 1, 7, 0, 0, 7, 4, 3, 5, 7,
  3, 0, 5, 4, 7, 4, 1, 7, 2, 0,
  0, 7, 3, 1, 4, 0, 6, 5, 2
};
```

The file to generate the data for "FST4Wsymbols" is part of the WSJT suite of programs. Open Windows Command Prompt. Navigate to directory C:\WSJTX\bin>. Enter the following command using your own callsign, grid square, and power level(dBM) enclosed in quotes as follows:

```
fst4sim "WZ9ZZZ FN21 30" 120 1500 0.0 0.1 1.0 0 99 F > FST4W_message.txt"
```

This will create a file named "FST4W_message.txt" in C:\WSJTX\bin. Open the "FST4W_message.txt" file into a text editor. Separate the "Channel symbols:" data with commas. Use cut and paste to replace the example data with the data generated with your station information.

The file to generate the data for "FT8symbols" is also part of the WSJT suite of programs. Open Windows Command Prompt. Navigate to directory C:\WSJTX\bin. Enter the following command using your own callsign and grid square enclosed in quotes as follows:

```
C:\WSJTX\bin> ft8code "CQ WZ9ZZZ  AA99"  > FT8_message.txt
```

A file named “FT8_message.txt” will be created in C:\WSJTX\bin. Open the FT8_message.txt file into a text editor (use “Channel symbols (79 tones)” data). Separate the "Channel symbols:" data with commas. Cut and paste the symbol message data to replace the example variable’s data.

The following libraries are used by the sketch:
Adafruit_GFX
Adafruit_SSD1306
FlashStorage_SAMD (not required for K4CLE board version)
WiFiNINA
RTCZero
TinyGPSplus
NTPClient
Time

The libraries are located in the Arduino IDE at: Sketch > Include Library > Manage Libraries. Windows users will find the menu bar on top of the active window. Linux and MAC users will find the menu bar at the top of the primary display screen.

In the unlikely event of operational problems, I suggest the builder check the Arduino Nano IOT and OLED display boards individually to ensure proper operation. The Arduino board can be checked using some of the example sketches provided with the open source Arduino software. The simple “blink” example sketch will confirm that a sketch can be loaded and the Arduino board is functioning. The OLED display may be checked by running one of the AvrI2C examples found with the SSD1306Ascii library.

Transmit Time, Mode, and Band Scheduling

Transmit time is determined by the variable TXinterval. If TXinterval = 3 the unit will transmit once every 3rd 2 minute transmit slot. Alternately, one of three transmit schedules may selected.

If a transmit schedule is selected the transmit times, modes of operation, and frequency are controlled by a 3 column, 1-60 row multi-dimensional array called “schedule_x [][3]” The format for “schedule_x [][3]” is {TIME, MODE, BAND},. Any number of rows up to 60 may be used. Ensure the last line is (-1,-1,-1),

Format: (TIME, MODE, BAND),

TIME: WSPR & FST4W are in 2 minute increments past the top of the hour
FST4W300 is in 5 minute increments past the top of the hour

MODE: 1-WSPR 2-FST4W120 3-FSR4W300 4-CW 5-FSK(QRSS) 6-FT8

BAND: is the 'TXdialFreq' band number i.e. 2 for 1836600 kHz

Note: FST4W modes are normally used only on LF and MF

Ensure times do not overlap when using FST4W300

FST4W 900 and 1800 operation are not supported because of Nano 33 IOT timing accuracy
And Si5351 board frequency stability.

Ensure the format is correct
i.e. brackets TIME comma MODE comma BAND comma brackets comma {24,1,11},

Band frequencies are stored in the variable “TXdialFreq []”. As the name implies, this is the dial frequency as used in WSJTX and not the actual transmit frequency. The offset frequency (1400 – 1600 Hz in the case for WSPR) is added to the dial frequency in the sketch to determine the actual transmit frequency.

Band frequencies are stored in Hz i.e 7.04 MHz = 7040000. Any number of bands may be stored. Any frequency below 41 MHz may be entered.

Scheduling example:

```
const int schedule [][3] = {
  {0, 1, 1},   // WSPR at top of the hour on 474.200 kHz
  {2, 2, 0},   // FST4W120 at 2 minutes past the hour on 136.000 kHz
  {4, 1, 6},   // WSPR at 4 minutes past the hour on 10138.700 kHz
  {20, 3, 0},  // FST4W300 at 20 minutes past the hour on 136.000 kHz
  {58, 1, 11}, // WSPR at 58 minutes past the hour on 28124600 kHz
  (-1,-1,-1),  // End of schedule flag
  };
```

Band frequencies example:

```
const unsigned long TXdialFreq [] =
{
  136000  , // Band 0
  474200  , // Band 1
  1836600 , // Band 2
  1836800 , // Band 3
  3568600 , // Band 4
  3592600 , // Band 5
  5287200 , // Band 6
  5364700 , // Band 7
  7038600 , // Band 8
  10138700, // Band 9
  14095600, // Band 10
  18104600, // Band 11
  21094600, // Band 12
  24924600, // Band 13
  28124600, // Band 14
  40620000, // Band 15 – experimental 8 meter WSPR
  40680000, // Band 16 – experimental 8 meter WSPR & FT8
  3569950,  // Band 17 - QRSS (mid band)
  7039950,  // Band 18 - QRSS (mid band)
  10140050, // Band 19 - QRSS (mid band)
  14096950, // Band 20 - QRSS (mid band)
  28074000, // Band 21 - 10 meter FT8
  0
};
```

Operation

The K4CLE board uses an accurate TCXO so calibration is not required. A one time frequency calibration is required for the individual board version. Connect the unit to an external frequency counter, select “FREQ CAL” from the menu then follow the instructions to adjust to 25 MHz.

When the unit is turned on, the auto-calibration algorithm for the millisecond timer of the Nano 33 IOT begins. This takes 100 seconds to complete.

The first screen to appear is the timing selection screen for WiFi, GPS, or RTC timing. The device will default to RTC timing if nothing is selected after 30 seconds. If WiFi is selected the device display “CONNECTED” along with WiFi details for a few seconds before going to the home screen. If unsuccessful, the device will default to RTC timing. If GPS is selected the device will search for four satellites to obtain valid data. If valid data cannot be found within two minutes the unit will default to RTC timing.

The home screen provides all the necessary transmit parameter information. The transmitter ON/OFF status is displayed in the upper right hand corner. Use PB2 to toggle the transmitter on and off.
.
Five pushbuttons are used to control the functions. “PB2” is normally used to select the function and “MENU” is used to return to the function selection. A shortcut to synchronize the time (only required if WiFi or GPS is not available) provided the displayed time is correct to within +/- 30 seconds. While in the “CLOCK SET” function, simply depress pushbutton 1 at the top of the minute.

The “CLOCK” function also displays the day of the week, month, and temperature in Centigrade and Fahrenheit. The displayed temperature is actually the DS3231 IC temperature so it may vary somewhat from actual ambient temperature.

The “ANT TUNE” function uses a SWR indicator circuit based upon the Dan Tayloe, N7VE, resistive SWR bridge. When selected, the display will not tell you what the actual SWR is, but will serve as an indicator to let you know when the antenna is tuned to a nominal 50 ohm impedance. Simply adjust the displayed bar graph for a minimum amount of bars.

The “MENU” pushbutton is used to select or exit a function when not transmitting.

Pushbutton pin allocations follow:

HOME SCREEN		**TRANSMIT**	**CLOCK**	**BAND**
PB1	N/A	N/A	N/A	Decrease band
PB2	Turn transmitter ON/OFF	Select HOME SCREEN	Select Clock display	Depress with PB3 to save
PB3	N/A	N/A	N/A	Depress with PB2 to save
PB4	N/A	N/A	N/A	Increase band

EDIT	**POWER**	**TX INTERVAL**	**SET OFFSET**
PB1	Decrease power	Increase TX interval	Increase offset
PB2	Depress with PB3 to save	Depress with PB3 to save	Depress with PB3 to save
PB3	Depress with PB2 to save	Depress with PB2 to save	Depress with PB2 to save
PB4	Increase power	Decrease TX interval	Decrease offset

TX HOPPING		**ANT TUNE**	**MODE**
PB1	"YES" increase offset	N/A	Depress to select mode
PB2	Depress with PB3 to save	Select Ant Tune	Depress with PB3 to save
PB3	Depress with PB2 to save	N/A	Depress with PB2 to save
PB4	"NO"	N/A	Depress to select mode

FREQ CALIB (adjust to 25 MHz)	
PB1	Decrease frequency
PB2	Depress with PB3 to save
PB3	Change resolution / Depress with PB2 to save
PB4	Increase frequency

(Clock settings are only required if WiFi or GPS are not available)

	CLOCK SET	DATE SET
PB1	Time sync* / Set Hour	Set Day
PB2	Set Minute	Set Month
PB3	N/A	Set Year
PB4	Hold to change time**	Hold to change date**

* Depress at top of the minute to synchronize

** Changes are saved when PB4 is released

	EDIT CALL	EDIT GRID
PB1	Depress to change selected character	Depress to change selected character
PB2	Depress with PB3 to save	Depress with PB3 to save
PB3	Depress with PB2 to save N/A	Depress with PB2 to save
PB4	Depress to select character	Depress to select character

Note: Use "#" to denote the end of edited call or gridsquare entries.

SOLAR DATA (only works with WiFi)

PB1	N/A
PB2	Select solar data
PB3	N/A
PB3	N/A