[FlexTrack](#_f7bcxlvdupdl)

[adc](#_yc4ekqu36wsq)

[aprs?](#_263l71j99ezw) (will look into later)

[ds18b20](#_t6nxcgn97fno)

# FlexTrack

**This is the main arduino script that runs on a continuous loop. It calls the functions in the other scripts. Things are commented out (disabled) or enabled and configured from this script.**

**Overall, the other scripts run conditionally depending on what is enabled (defined) or disabled in this script.**

|  |
| --- |
| PROGRAM START  // PREPROCESSING ----------------------------- INCLUDE AVR/PGMSPACE.H  DEFINE SETTINGS FOR HABDUINO (not applicable to us) DEFINE SETTINGS FOR RTTY (radioteletype - we're not going to use) PAYLOAD ID FREQUENCY BAUD RATE PWM SHIFT  DEFINE POWERSAVING (not applicable)  DEFINE SETTINGS FOR LORA (LowRange LowPower Wireless Network -- Not using)  DEFINE APRS SETTINGS CALLSIGN SSID (radio identifier code) PATH ALTITUDE (height where the transmitting parameter will switch from WIDE1-1 to WIDE2-1 (basically how many hops to digipeaters will be made to transmit location http://wa8lmf.net/DigiPaths/NNNN-Digi-Demo.gif) USE WIDE2 (setting that turns the WIDE2 option on or off) TX\_INTERVAL (how often to transmit in minutes) PRE-EMPHASIS (no idea what this is) RANDOM (value to randomize transmission time) COMMENT (optional comment to add to transmission) TELEM\_INTERVAL (how often to sent telemetry (sensor) packets; in minutes)  IF HS\_APRS\_300 HAS BEEN DEFINED: (not defined for our purposes) THEN DEFINE THE ARDUINO PIN ADDRESSES THE FOLLOWING: GPS\_I2C (I2C serial connection to GPS) LED\_STATUS LED\_TX APRS\_ENABLE (enable pin for transmitter [if applicable]) APRS\_DATA (pin connected to data pin on transmitter) A0\_MULTIPLIER (reference voltage for analog pin readings) WIREBUS (1-wire communication bus for sending data i.e. temperature data)  IF HS\_RTTY\_300 DEFINED: THEN DEFINE: ... etc. same stuff as above.  Does the same IFDEF blocks for: UAVANUT\_LORA HABDUINO (what we are using, kind of) UAB //-------------------------------------------------- IF GPS\_SERIAL IS NOT DEFINED AND IF GPS\_I2C IS NOT DEFINED THEN DEFINE GPS\_SERIAL AS 'Serial' (probably sets it so that it uses the RX and TX pins on the arduino rather than other comm ports)  IF DEBUG\_SERIAL IS NOT DEFINED THEN DEFINE DEGBUG\_SERIAL TO 'Serial'. (probably sets the debug mode to debug the correct serial ports, [arduino rx and tx?]  DEFINE EXTRA\_FIELD\_FORMAT as "%d,%d... (basically 5 integers) DEFINE EXTRA\_FIELD\_LIST AS:  ",(int)((GPS.Speed \* 13) / 7), GPS.Direction, GPS.Satellites, DS18B20\_Temperatures[0], Channel0Average"  (this is probably a list of parameters to use; casts gps speed with some math as an integer, takes gps direction, gps satellites?, ds18b20\_tempratures[0] (the zeroth index of the data from a temperature sensor with model number ds18b20?), and channel0average is probably the average analog readings from analog sensors.) -- I think this adds on extra values to the transmission that includes this data.  DEFINE SENTENCE\_LENGTH AS 100. (NMEA (Received) Sentence - The ASCII data stream received from navigation equipment (such as GPS receivers) conforming to the NMEA 0182 Version 2.0 specification. APRS supports five NMEA Sentences: GGA, GLL, RMC, VTG and WPT.) -- This probably just sets the length of data transmitted.  // GLOBALS (probably setting up global variables) ----------------------  CREATE STRUCTURE OF DATA TYPE "TBinaryPacket" (like an array but with different data types -- think matlab structures) STRUCTURE CONTAINS FOLLOWING VARIABLES: PayloadIDs (8bit unsigned integer) Counter (16 bit unsigned integer) BiSeconds (16 bit unsigned integer) Latitude (float) Longitude (float) Altitude (32 bit signed integer) END STRUCTURE  CREATE STRUCTURE NAMED "GPS" OF DATA TYPE "TGPS" CONTAINING THE FOLLOWING VARIABLES: Hours, Minutes, Seconds (all integers) SecondsinDay (unsigned long integers) Longitude and Latitude (floats) Altitude (long integer signed) Satellites (unsigned integer) Speed (integer) Direction (integer) [what number represents what direction?] FixType (byte? single byte?) psm\_status (byte) InternalTemperature (float) BatteryVoltage (float) ExternalTemperature (float) Pressure (float) BoardCurrent (unsigned integer) errorstatus (unsigned integer) FlightMode (byte) PowerMode (byte) END STRUCTURE  DECLARE SentenceCounter VARIABLE AND INITIALIZE TO 0.  // SETUP FUNCTION ----------------------------  IF GPS\_SERIAL IS DEFINED CALL GPS\_SERIAL.begin FUNCTION WITH INPUT (9600) (start serial console with baud rate 9600) END IF  IF DEBUG\_SERIAL IS DEFINED CALL DEBUG\_SERIAL.begin FUNCTION WITH INPUT (9600) (start serial console with baud rate 9600)  CALL serial.println FUNCTION WITH "" INPUT  (prints nothing on a new line) PRINT "FlexTrack Flight Computer, payload ID(s)" TO SERIAL CONSOLE  IF RTTY\_DATA IS DEFINED (does not apply to us) PRINT RTTY\_PAYLOAD\_ID TO SERIAL CONSOLE  IF LORA\_NSS IS DEFINED (does not apply to us) PRINT LORA\_PAYLOAD\_ID TO SERIAL CONSOLE  IF APRS\_DATA IS DEFINED (this applies to us) PRINT APRS\_CALLSIGN TO SERIAL CONSOLE  PRINT TWO NEWLINES TO SERIAL CONSOLE END DEBUG\_SERIAL IF  IF GPS\_I2C IS DEFINED PRINT "I2C GPS" ON A NEW LINE (likely will not happen) ELSE  PRINT "SERIAL GPS" ON A NEWLINE (this is likely what will happen) END IF  IF LORA\_NSS IS DEFINED (does not apply) PRINT "LoRa telemetry enabled"  IF RTTY\_BAUD IS DEFINED (does not apply) PRINT "RTTY telemetry enabled"  IF APRS\_DATA IS DEFINED (does apply) PRINT "APRS telemetry enabled" END IF STATEMENTS  PRINT "Free memory = " TO SERIAL CONSOLE PRINT RETURN VALUE OF freeRam() FUNCTION  CALL SetupLEDs() FUNCTION  CALL SetupGPS() FUNCTION  CALL SetupADC() FUNCTION  IF LORA\_NSS IS DEFINED (does not apply) CALL SetupLoRa() FUNCTION END IF  IF RTTY\_BAUD AND RTTY\_DATA IS DEFINED (does not apply) CALL SetupRTTY() FUNCTION END IF  IF APRS\_DATA IS DEFINED (does apply!) CALL SetupAPRS() FUNCTION END IF  IF WIREBUS DEFINED (may apply!) CALL Setupds18b20() FUNCTION END IF  END SETUP FUNCTION  // -------------- MAIN LOOP FUNCTION  CALL CheckGPS() FUNCTION  IF RTTY\_BAUD AND RTTY\_DATA ARE DEFINED (not applicable) CALL CheckRTTY() FUNCTION END IF  IF LORA\_NSS IS DEFINED (not applicable) CALL CheckLoRa() FUNCTION END IF  IF APRS\_DATA IS DEFINED CALL CheckAPRS() FUNCTION END IF  CALL CheckADC() FUNCTION  CALL CheckLEDs() FUNCTION  IF WIREBUS IS DEFINED CALL Checkds18b20() FUNCTION END IF  END MAIN LOOP FUNCTION  freeRam() FUNCTION DEFINITION: DECLARE INTEGER VARIABLE \_\_heap\_start AS EXTERNALLY DEFINED VARIABLE DECLARE INTEGER POINTER VARIABLE \*\_\_brkval AS EXTERNALLY DEFINED VARIABLE  DECLARE LOCAL INTEGER VARIABLE 'v' RETURN: ADDRESS OF v MINUS THE RESULT OF CONDITIONAL EXPRESSION: IF \_\_brkval IS ZERO OR NULL, VALUE IS ADDRESS OF \_\_heap\_start ELSE VALUE IS VALUE OF \_\_brkval VARIABLE. END FUNCTION DEFINITION  (my guess is that this calculate the amount of ram available by subtracting the address of a newly declared variable v and the start of the "stack" which is defined as \_\_heap\_start somewhere else (possibly in <avr/pgmspace.h> which was included at the top of this code.  END FLEXTRACK CODE PROGRAM |

# adc

**This script checks the values from the analog pins if A0\_MULTIPLIER is enabled. It saves values to an array, which it then sums up and averages every second and sends to main code**

|  |
| --- |
| DECLARE VARIABLES: CheckADCChannels (unsigned long) = 0  IF A0\_MULTIPLIER IS DEFINED DECLARE VARIABLES: Channel0Readings (integer array size 5) Channel0Average (unsigned integer) END IF  SetupADC() FUNCTION DEFINITION: IF A0\_MULTIPLIER IS DEFINED CALL FUNCTION analogReference() WITH DEFAULT INPUT (reference voltage probably set to 4.9 volts) SET A0 PIN TO INPUT MODE END IF END SetupADC() FUNCTION  CheckADC() FUNCTION DEFINITION IF millis() IS GREATER THAN OR EQUAL TO CheckADCChannels  (if time since last call is greater than 1 second)  IF A0\_MULTIPLIER IS DEFINED SET Channel0Average EQUAL TO RETURN VALUE OF ReadADC() WITH PARAMETERS: A0 (pin number), A0\_MULTIPLIER (multiplier value), AND Channel0Readings (array of readings from A0 pin) END IF  SET CheckADCChannels = millis() + 1000L (sets CheckADCChannels value to the time since the program started plus 1 second) END IF  (basically this part makes it so that it is averaged every second across 5 readings)  END FUNCTION  ReadADC FUNCTION DEFINITION - PARAMETERS (Pin (int), Multiplier (float), Readings Array Pointer (int)) DECLARE INTEGER VARIABLE "i", AND UNSIGNED INTEGER "Result"  FOR LOOP ( i goes from 0 to 3 [4 iterations], and increments up) Readings[i] = Readings[i + 1] (sets current value to the next element in array) END FOR LOOP  SET Readings[4] to the analogRead VALUE AT DEFINED PIN NUMBER (so this program shifts all values in array to the left, and in the last empty spot, it reads a new value)  SET Result = 0  FOR LOOP (i from 0 to 5 [5 iterations] incrementing up) SET Result EQUAL TO Result PLUS VALUE OF Readings[i] END FOR LOOP (sums up all elements in array)  RETURN Result \* Multiplier / 5.0 (float) (this takes the sum of all array values, multiplies it by a multiplier, and finds the average, which is returns as part of the function) END FUNCTION DEFINITION |

# aprs?

IF APRS\_DATA IS DEFINED

INCLUDE <util/crc16.h>

INCLUDE <avr/pgmspace.h>

DEFINE THE FOLLOWING:

* BAUD\_RATE (1200)
* TABLE SIZE (512)
* PREABME\_BYTES (50)
* REST\_BYTES (5)
* PLAYBACK\_RATE ( F\_CPU / 256)
* SAMPLES\_PER\_BAUD (PLAYBACK\_RATE / BAUD\_RATE)
* PHASE\_DELTA\_1200 ( ((TABLE\_SIZE \* 1200) SHIFTED 7 BITS TO LEFT) / PLAYBACK\_RATE)
* PHASE\_DELTA\_2200 ( ((TABLE\_SIZE \* 2200) SHIFTED 7 BITS TO LEFT) / PLAYBACK\_RATE)
* PHASE\_DELTA\_XOR (PHASE\_DELTA\_1200 ^ PHASE\_DELTA\_2200)
* APRS\_DEVID (“APEHAB”)

DECLARE THE FOLLOWING VARIABLES

* NextAPRS = 0 (unsigned long int)
* aprs\_mode = 0 (int)
* APRSSentenceCounter (unsigned int)
* \*\_txbuf = 0 (pointer variable of volatile type [can change by factors outside code] and static type [can be changed] through various iterations of for loop])
* \_txlen = 0 (non-pointer, has same data type as above)

IF WIREBUS IS DEFINED

DECALRE EXTERNAL INTEGER VARIABLE ARRAY DS18B20\_Temperatures

END IF

DECLARE STATIC CONSTANT ARRAY \_sine\_table[ ] = VALUES IN sine\_table.h AND STORE IN PROGRAM MEMORY (USING PROGMEM IDENTIFIER)

(basically this initializes an array and fills it with the hex values in sine\_table.h and stores this in program memory instead of SRAM. this doesn’t allow changing of this array)

// Start code ---------------------

# ds18b20

**This script defines the setup and reading behavior of DallasTemperature communication protocols for the OneWire bus system. May or may not be useful to us.**

|  |
| --- |
| IF WIREBUS IS DEFINED (this entire script doesn't run unless WIREBUS is defined) INCLUDE <OneWire.h> INCLUDE <DallasTemperature.h>  DEFINE MAX\_SENSORS TO BE 4  DECLARE VARIABLES: SensorCount = 0 (int) CheckDS18B20s = 0 (unsigned long int) GettingTemperature = 0 (int) oneWire(WIREBUS) (OneWire object?) (initializes oneWire variable to the pin to access oneWire?) defined in OneWire.h sensors(&oneWire) (function to read address of oneWire variable?) defined in DallasTemperature.h DS18B20\_Temperatures (array of size MAX\_SENSORS)  Setupds18b20() FUNCTION DEFINITION: CALL sensors.begin() function (Probably a function defined in DallasTemperature.h INCLUDE SET SensorCount TO RETURN VALUE OF sensors.getDeviceCount() function PRINT SensorCount VALUE TO SERIAL CONSOLE PRINT "DS18B20's on Bus" TO SERIAL CONSOLE AS NEW LINE USE min() FUNCTION TO DETERMINE WHETHER MAX\_SENSORS OR SensorCount IS SMALLER AND SET TO NEW SensorCount VALUE  IF SensorCount IS GREATER THAN ZERO CALL FUNCTION sensors.setResolution() INPUT 9 AS ARGUMENT END IF  END FUNCTION DEFINITION   Checkds18b20() FUNCTION DEFINITION: // This function alternates between updating temperatures in array, or requesting temperatures for the next time it is called. Probably related to the "aprs\_telemetry" setting which tells it to send sensors data every other minute rather than every minute.  IF millis() (time since program start) IS GREATER THAN OR EQUAL TO CheckDS18B20s VALUE (0 at start) AND IF GettingTemperature IS TRUE DECLARE INTEGER i  FOR LOOP (i from 0 to less than number of sensors [SensorCount], incrementing up) SET DS18B20\_Temperatures[i] = sensors.getTempCByIndex(i) (this sets the current array value to the temperature in Celsius from the temp sensor using a function from the DallasTemp.h file) PRINT CURRENT TEMPERATURE VALUE TO SERIAL CONSOLE END FOR LOOP  SET CheckDS18B20s VALUE TO TIME SINCE PROGRAM START PLUS 10 SECONDS END IF STATEMENT;  ELSE IF GETTING TEMPERATURE IS FALSE CALL sensors.requestTemperatures() (function in includes that initializes reading temperatures?) SET CheckDS18B20s EQUAL TO TIME SINCE PROGRAM START PLUS 1 SECOND END ELSE.  SET GettingTemperature TO FALSE.  END FUNCTION DEFINITION  END IF DEFINED BLOCK |