**System Overview**

The Arduino DECADES code is intended to be installed on an Arduino UNO unit for connecting sensors to the FAAM on board data acquisition system: DECADES (<http://www.faam.ac.uk>). The code has expanded to a point where it is necessary to upgrade the standard processor used on the Arduino UNO unit where a greater amount of code space is available. The UNOPro has been used to this effect and can be found at <http://goo.gl/4yaaW2>.

The object of this document is to provide an overview of the system design and logic. The code is available on <https://github.com/mhobby/ArduinoDECADES>.

The DECADES system records data on two TCP servers and is available in flight to scientists on a UDP Multicast stream. An NTP server is made available to provide accurate time data for all instruments. The Arduino DECADES unit records the difference between two analogue voltages connected to the Arduino 10 bit ADC. Each ADC is unipolar and therefore produces a unsigned 10 bit representation. The difference is carried using 16 bit logic and therefore results in an equivalent 11 bit signed representation of the difference between the two voltages.

The Arduino unit transmits this data to the predefined servers of the DECADES system, which can be configured using an SD card inserted into the unit.

Timing in the Arduino unit is managed by a local clock in milliseconds and a local prediction of the NTP clock. This prediction is frequently adjusted by the NTP server to ensure that data is synchronised to UTC. Note that the local clock demonstrates significant drift and therefore frequent adjustments, using the NTP server, are required to maintain sample timing accuracy to within 1s of UTC. By taking this approach, fast responsive sampling can be maintained whilst remaining synchronised to a time standard for comparison with other measurements.

A diagram of the various functions used within the code is shown in Figure 1



Figure 1 – Function call diagram for the Arduino DECADES unit

**Global Variables**

Not all global variables are listed below. Those not listed are self-explanatory or commented in the code. Including them here would confuse the objective of this document to present the system logic for recording samples.

|  |  |  |
| --- | --- | --- |
| unsigned long | syncTimeout | timeout in milliseconds for waiting for a response from an NTP server. If not on the same subnet this can be quite long and will disrupt the responsiveness of the ADC sampling |
| unsigned int | T | sampled period in milliseconds |
| unsigned int | syncT | time between synchronisations of the local copy of NTP time (stored in lNTP) - internal clock has significant drift from NTP so resync is required at frequent intervals |
| unsigned long | lNTP | local copy of NTP time (whole seconds) in seconds since 1/1/1900 |
| unsigned long | lNTPfrac | local copy of NTP time (fractional seconds) - functionality not implemented, since sample accuracy to nearest second required only |
| unsigned long | tNow | local clock time in milliseconds since startup |
| unsigned long | tPrev | local clock time in milliseconds since startup of last sample |
| unsigned long | lastSync | local clock time in milliseconds since startup of last NTP sync |
| bool | clockResyncFlag | flag to indicate if/when lNTP is synced to NTP |
| int | sensorPin | hardware arduino pin that sensor should be connected. Sensor output should be connected to sensorPin and sensorPin+1 |
| signed int | sensorVal | difference between last measured ADC values on sensorPin and sensorPin+1 |

**void setup()**

*Variables:*

|  |  |  |
| --- | --- | --- |
| *type* | *name* | *Description* |
| char | ID | identity string for this data acquisition unit |
| byte | IPL | IP address for this data acquisition unit |
| byte | IPG | IP address for the local gateway (assuming all units on same subnet, this is not required to be set) |
| byte | IPA | IP address of TCP server A |
| byte | IPB | IP address of TCP server B |
| byte | IPC | IP address for UDP multicast |
| byte | IPntp | IP address for NTP server |
| char | mode | can be set to RLS or DBG. In RLS mode, the acquisition unit will wait for receipt of 'Weight on Wheel' (WoW) flags to progress |

*Process:*

1. setup serial comms
2. setup SD card
   1. setup file system
   2. calculate SD free space
   3. look for "arddConf.cfg" file to configure system
      1. configure system (e.g. - IP addresses, etc.)
   4. if “arddConf.cfg” not available, resort to defaults
      1. configure system
3. setup ethernet comms
4. connect to ntp server
5. connect to udp multicast stream
6. if mode=RLS
   1. wait for 5 consecutive wow flags (indicating aircraft is in the air)

AND

wait for 5 consecutive valid flight no

[ both of which are received on the UDP multicast stream ]

1. connect to TCP server A
2. connect to TCP server B
3. set local NTP time from NTP server - call *getNtpTime*
4. set NTP sync time to local clock value
5. open file on SD card for data backup

**void loop() [loop continuously]**

*Variables:*

|  |  |  |
| --- | --- | --- |
| *Type* | *name* | *description* |
| Float | fs | Free space on disk in mega bytes |
| unsigned 32bit int | volFree | Free clusters on volume (used to calculate *fs*) |

*Process:*

1. get local clock value
2. if local clock value is bigger than local time of last sample + sample period
   1. record local clock value
   2. take ADC value
   3. increase local NTP value
   4. if local NTP value needs resynchronising – call *getNtpTime* and update lNTP from server.
3. print data output to serial terminal
4. transmit data to TCP servers and UDP multicast stream – call *formDataPackets*
5. if sd card: write data to sd card
6. return to (1)

**bool getNtpTime(unsigned long \*epoch, unsigned long \*epochFrac)**

*Arguments:*

|  |  |  |
| --- | --- | --- |
| unsigned long\* | epoch | pointer to variable within which to store the NTP seconds obtained from the NTP server |
| unsigned long\* | epochFrac | pointer to variable within which to store the NTP fractional seconds obtained from the NTP server |

*Variables:*

|  |  |  |
| --- | --- | --- |
| unsigned long | xmt | NTP time when the packet departed the NTP server (whole seconds) |
| unsigned long | xmtFrac | NTP time when the packet departed the NTP server (fractional seconds) |
| unsigned long | syncStart | local clock time, in milliseconds, when the request was sent to the NTP server |
| unsigned long | elapsed | local clock time, in milliseconds, that have elapsed since the NTP request was sent to the server |

*Process:*

1. request time from NTP server
2. record time of request to syncStart
3. wait in loop for reply from NTP server for syncTimeout seconds – using elapsed to measure this time.
4. If NTP server reply:
   1. Strip NTP XMT time from NTP packet to xmt and xmtFrac. Only the server transmit time is used to update the local NTP time. No account for round trip delay is used since in the DECADES system the NTP server is on the same subnet and the round trip delay is negligible.
   2. Store xmt and xmtFrac to address pointed to by epoch and epochFrac.
   3. Return true
5. If no NTP server reply and syncTimeout seconds has passed, return false.

**void formDataPackets()**

*Variables:*

|  |  |  |
| --- | --- | --- |
| char | UDPBuf | Buffer for UDP data (in format defined by ARDD0001\_UDP\_v3.csv) |
| byte | TCPBuf | Buffer TCP data (in format defined by ARDD0001\_TCP\_v3.csv) |
| byte | UDPBufptr | pointer to variable within which to store the NTP seconds obtained from the NTP server |
| byte | TCPBufptr | pointer to variable within which to store the NTP fractional seconds obtained from the NTP server |
| static unsigned int | pktCount | Number of packets transmitted, used to correlate TCP and UDP packets |
| static bool | running | Toggle that alternates for every sample to indicate that code is running |
| char\* | tempStr | Temporary variable to convert binary data into human readable string |

*Process:*

1. Copy data into TCPBuf or UDPBuf, formatting from binary value to string value where necessary
2. Transmit TCPBuf to TCP Server A
3. Transmit TCPBuf to TCP Server B
4. Transmit UDPBuf to UDP Multicast address
5. Increment pktCount