WCBM Mote Firmware Guide

Wireless CbM Mote Firmware Guide (1.0.1R Project Revision)

Rev.4

Code base: C code

Created: 25/10/2018 2:21 P.M.

Author: Tom Sharkey

Last Modified: 11/01/2021 3:43 PM

Modified by: Tom Sharkey

Contents

Revision History:

Rev.0 - Initial Document

Rev.1 – General Edits

Rev.2 – General Edits

Rev.3 – Addition of Tool Requirements and Environment Setup section

Rev. 4 – General Edits

1 Overview

This document provides an overview of the example firmware provided with the Condition Based Monitoring (CBM) kit. Companion documents are the "Getting Started Guide" provided with the CBM kit. This particular document version is based on the features available in the CBM code version referred to in the document title.

The purposes of this document are to enable the user to setup the firmware project in IAR Embedded Workbench, and to provide a high-level understanding of the programme structure, flow, and interactions, and to be able to customise the code. Please note that this code example is purely that – an example, and has not been tested or qualified in any way for production.

The embedded C code for the hardware in this project makes use of SmartMesh IP mesh network technology. The guide depicts a sample application that has been built on top of this SmartMesh framework. There are many functions which will not be discussed in this guide but can be investigated further using SmartMesh documentation. [https://dustcloud.atlassian.net/wiki/spaces/ALLDOC/overview?mode=glob]

A summary of the various C project files is given below:

- Main user application files:
    - main_prog.c *(top level application code with state machine)
    - scheduler.c (kernel module for managing scheduling timers, main loop)
    - SmartMesh_RF_cog.c (SmartMesh functions for mote communication and callbacks)
    - ADC_channel_read.c (initialise adc and holds adc sampling and fft math functions)
    - SPI0_ADXL362.c (SPI functions for interfacing with the ADXL362)
    - SPI1_AD7685.c (SPI functions for interfacing with the AD7685)
- ADuCM4050 drivers and libraries – provided in the relevant IAR CMSIS pack.
- SmartMesh library source files:
    - dn_endianness.c
    - dn_hdlc.c
    - dn_ipmt.c
    - dn_lock.c
    - dn_serial_mt.c
    - dn_uart_handle.c

In the following code structure section, the high-level functionality of the main user application files will be explained.

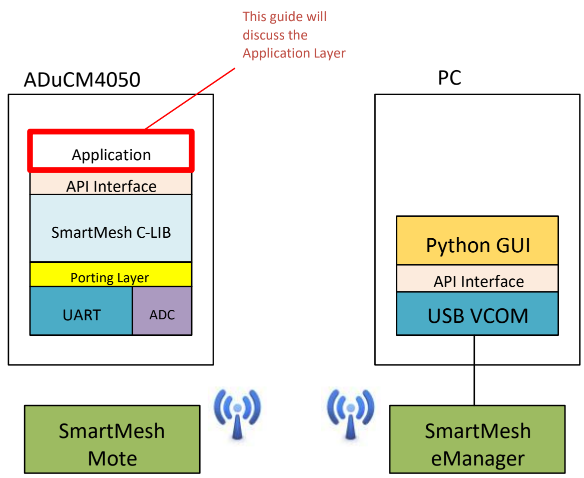


Figure 1: *Hardware Overview*

2 Tool Requirements and Environment Setup

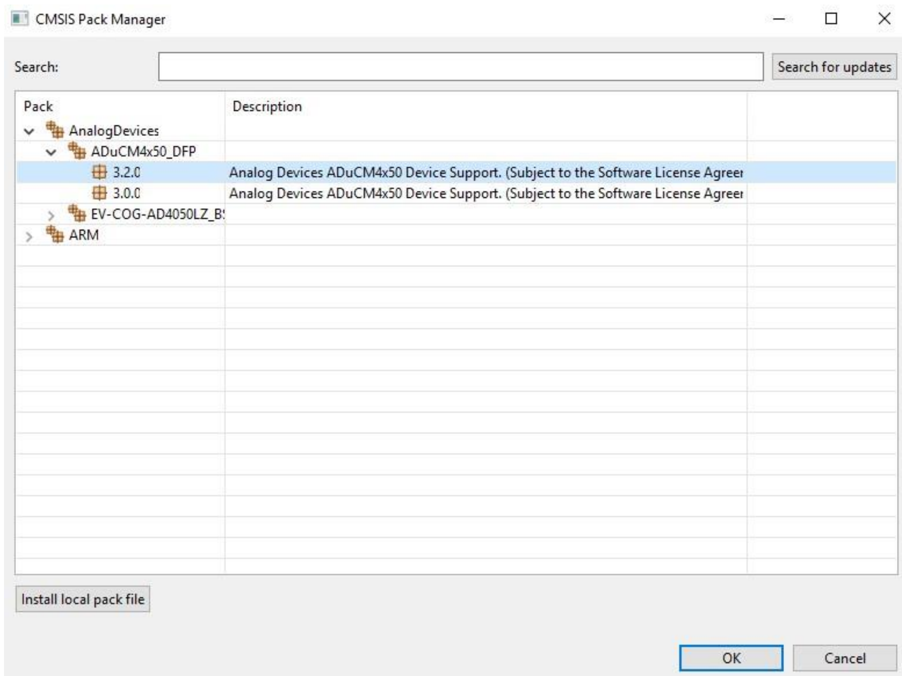
This project has been compiled and built in the IAR Embedded Workbench IDE version 8.22.2. The use of older versions may result in broken project options and/or project options not being fully transferred over and dependencies not working – these will then need to be manually set up.

The project depends on the installation of the appropriate CMSIS pack in IAR Embedded Workbench, which enables the correct ADuCM4050 drivers and libraries to be installed, as these are not shipped with the source code. The relevant SmartMESH libraries are included in the source code. If the project does not open correctly due to version mismatch – for example – the project can be set up from scratch as shown in the next steps.

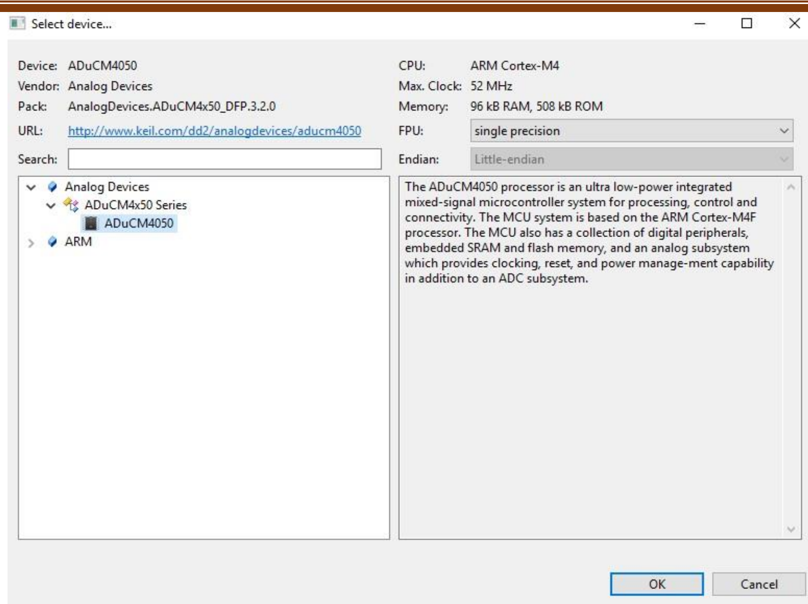
Copy the directories provided in the 'C firmware' folder to your PC (src, ext, include, project) maintaining the same directory structure. If you are using IAR EWB ver. 8.22.2 or newer, the project should open successfully. The project is found in the 'project' directory of the provided files. Open the 'cbm.eww' workspace file in IAR EWB. If this opens successfully, with no broken project options, follow steps 3-7 below to install the correct CMSIS driver package for the ADuCM4050 microcontroller. It is also worth checking the remaining steps to ensure that the Project Options are set correctly and all of the required source files are present.

If the project cannot be opened successfully, follow all of the steps below to set up the project from scratch in IAR Embedded Workbench.

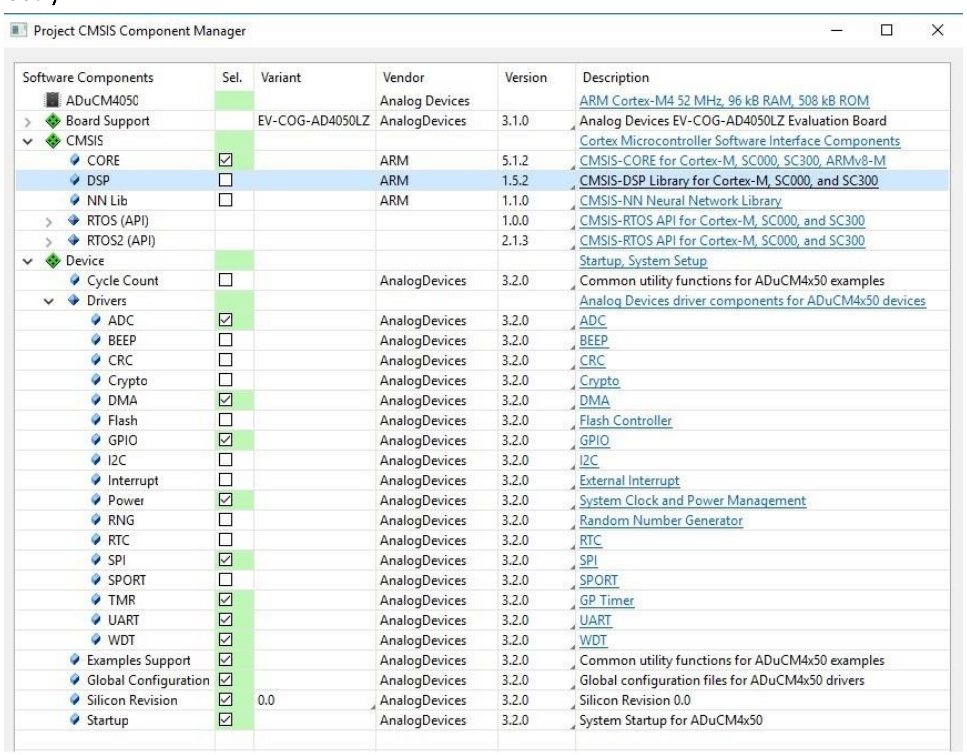
1. Delete all of the files with the name ‘cbm.*’ in the ‘project’ folder (e.g. cbm.eww, cbm.ewt etc). Delete the ‘project\settings’ folder. These will be recreated in the next steps. Ensure that the ‘ADuCM4050_moreSRAM.icf’ file is NOT deleted.
2. File->New Workspace
3. Project->CMSIS-Pack->Pack Installer
4. Select ADuCM4x50_DFP->3.2.0, right-click this package, and select install
5. Select ARM -> CMSIS -> 5.7.0, right-click and install, then press OK. Versions may vary for the DFP and CMSIS packages, but the version installed should be the same or newer than the versions shown.



6. Project->Create New Project->Empty CMSIS Pack Project
7. Select ADuCM4x50

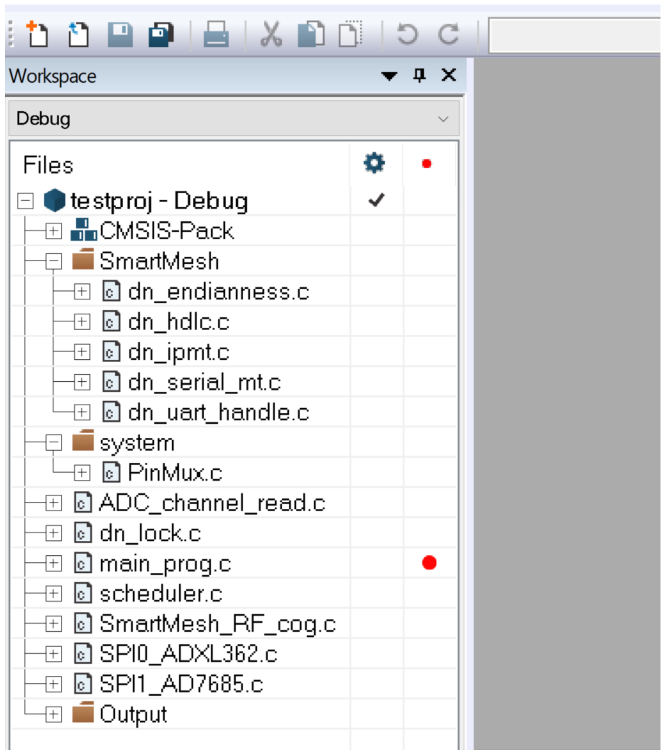


8. Select Pack Components shown below. These components may be highlighted in orange until all the correct options are highlighted, but should turn green when all options are selected correctly.

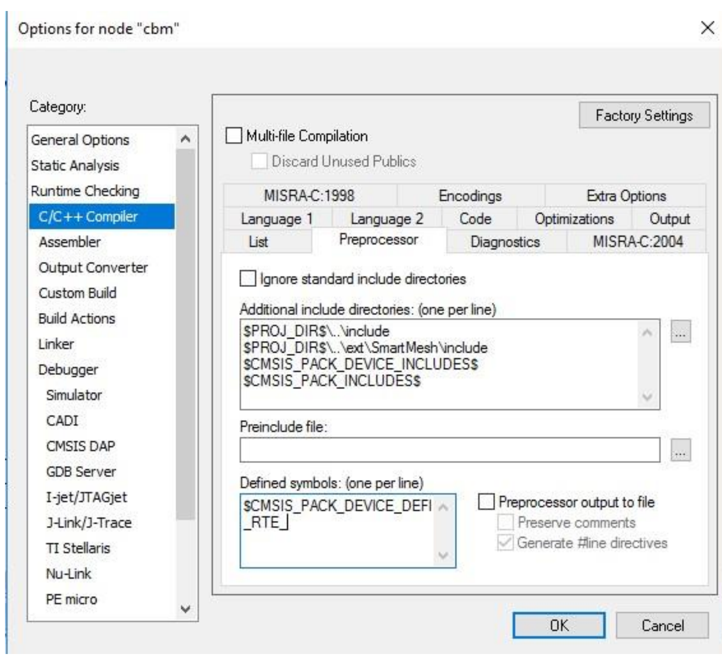


9. Save the project in the 'project' folder. The source files will then be at the correct level relative to the project.

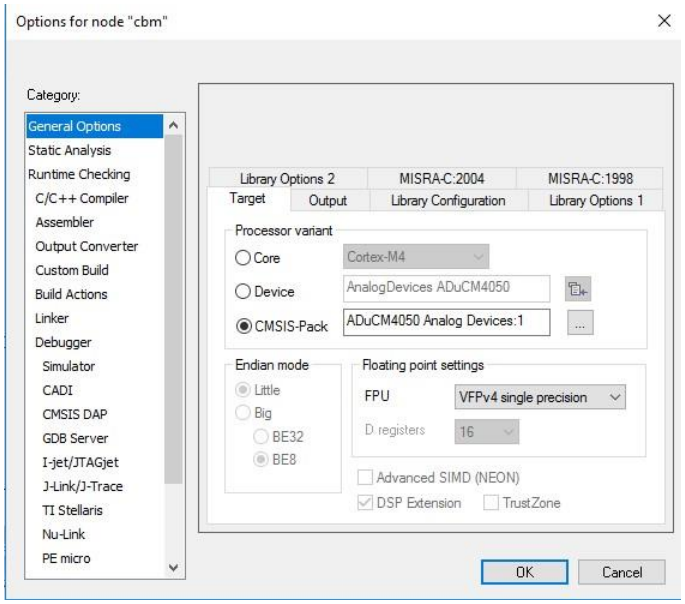
10. Project->Add Files: Add the source files provided, to the project (user application files, and SmartMESH library files) from the src, src/system, and ext/SmartMesh/src folders. These can be organized in Groups if desired (Project->Add Group). There is no need to add the header files in the various 'include' folders to the project explicitly as these will be added as compiler search options in step 10.



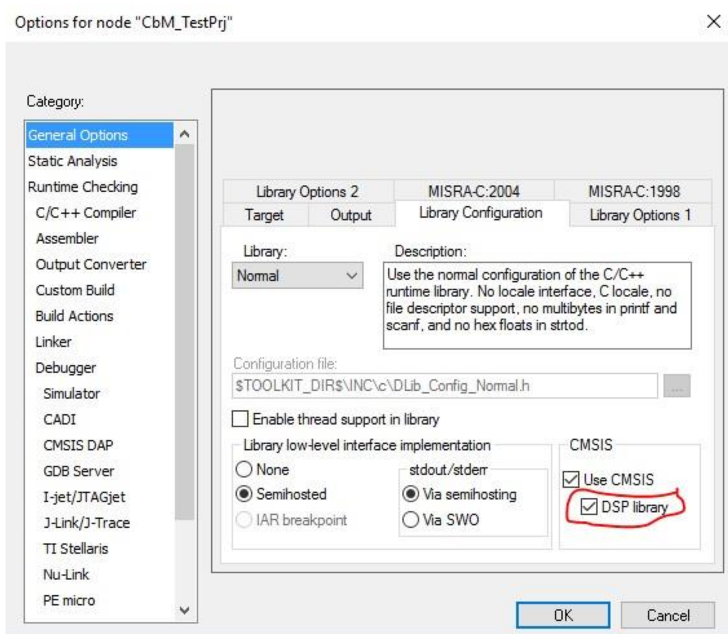
11. Project->Options->C/C++ Compiler: Add the include directories shown below $PROJ_DIR$\..\include and $PROJ_DIR$\..\ext\SmartMesh\include



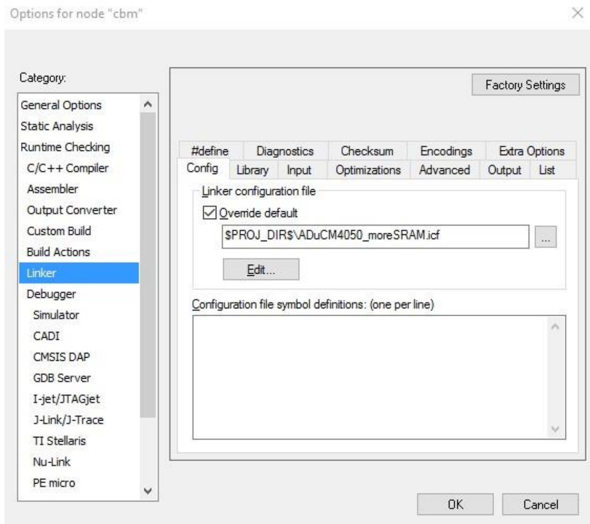
12. Project->Options->General Options->Target: Select the floating-point unit VFPv4 single precision



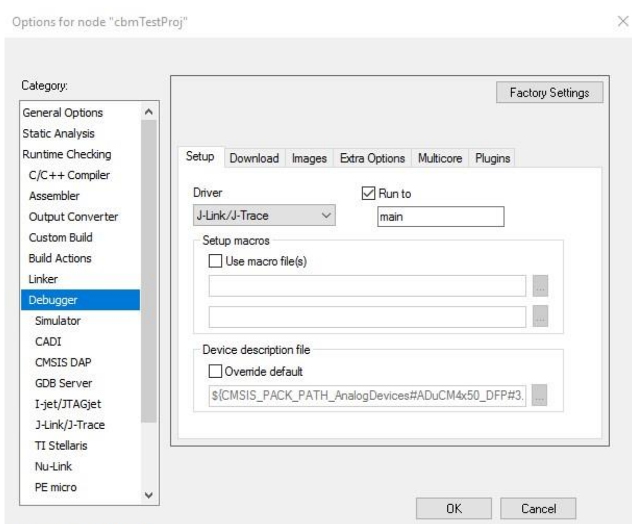
13. Project->Options->General Options->Library Configuration: Ensure that DSP library is checked



14. Project->Options->Linker, Select Override default and enter $PROJ_DIR$\ADuCM4050_moreSRAM.icf



15. Project->Options->Debugger->Setup. Select J-Link/J-Trace in the Driver selection. Ensure also that in the Download tab for the Debugger options, Use flash loader is selected.



16. File->Save Workspace (in the ‘project’ directory)
17. Project->Rebuild All

The project should compile and link without errors and be ready to download and debug to the mote.

3 Code Structure

The function of the main program is to:

- Initiate a connection between mote and manager
- Sample acceleration data via ADC
- Transmit raw and FFT data
- Handle changing sampling parameters from the manager

The overall operation of the program is depicted in Figure 2, with the initialization, communication state machine, and data handling state machine, scheduler timing, and data format depicted in the following sections.

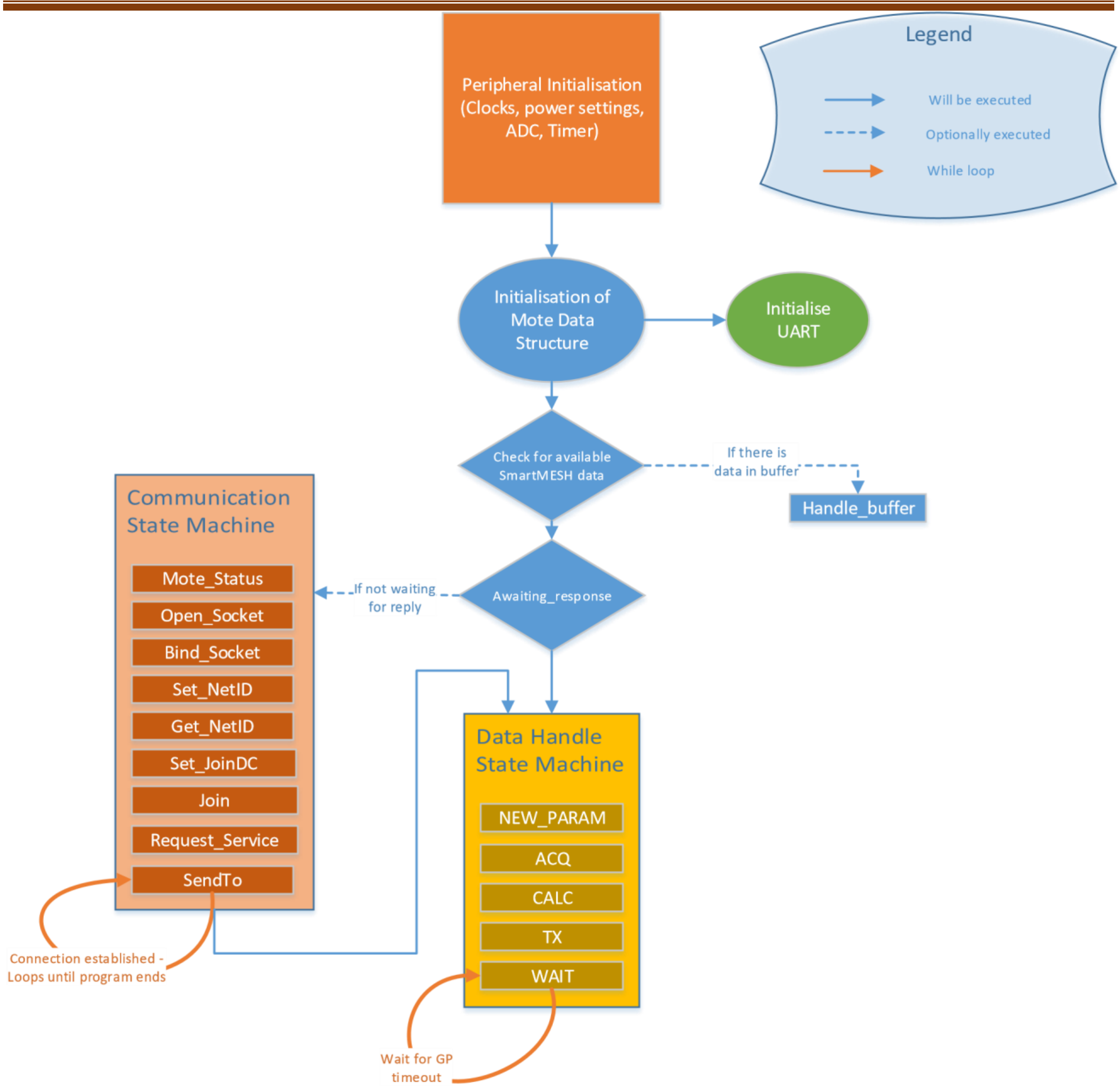


Figure 2: *High Level Diagram*

3.1 Initialisation:

Name	Description
Clock dividers	Simple constants that set frequencies corresponding to hardware clock frequencies. These are used throughout the code, such as when setting clock dividers later in initialization.
Mote parameters	My reply and my error are variables set for the user to make use of. They are currently not used in this version of the program. NET_ID is used to store the network ID number of the manager. Join_duty is a value between 0 and 255, and decides how quickly the mote will join the network, at the cost of greater energy usage. Ms_per_packet is used to set the millisecond gap between packets received.
Initialisation of Mote and Variables and Peripherals (UART)	Initializes variables necessary for application function. Calls dn_uart_init which initializes the UART of the microcontroller.
Start Scheduler	Begins and verifies correct function of the timer.
State machines	The main body of the program. The CONNECT and TRANSMIT state machines are described in further detail below. They are responsible for joining the mote to the network and beginning the flow of data between mote and manager (both directions).

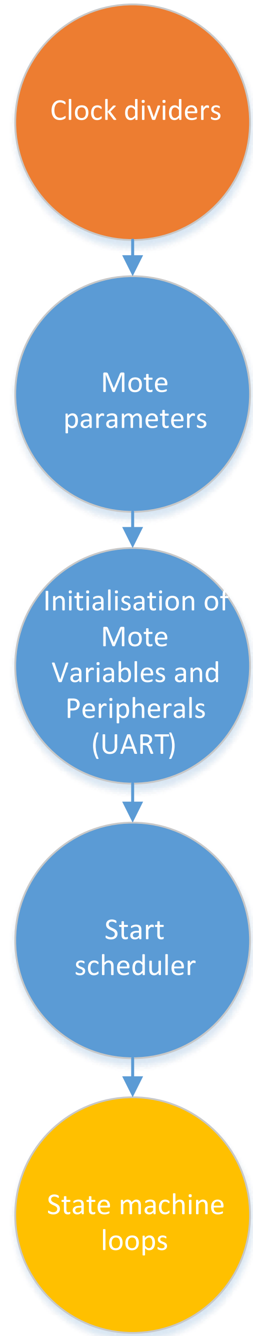


Figure 3: *Initialisation*

3.2 Communication State Machine:

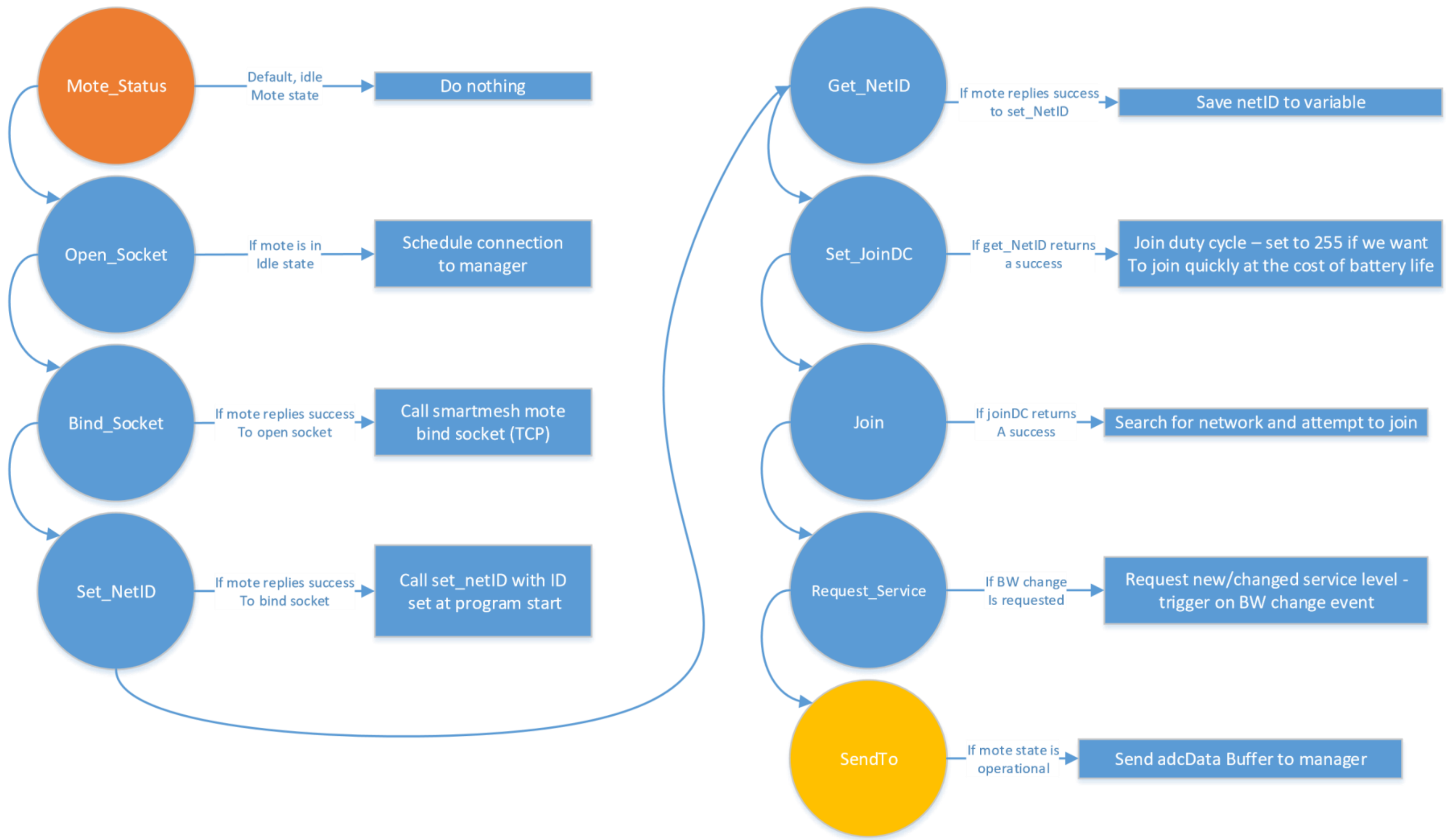


***Figure 4**: Communication State Machine*

State	Description
Boot Status	A state that is only used for program startup.
Mote Status	This is the "idle" state for the mote. The mote begins the state machine here (after moving on from boot status. The program will return to this state if it reaches a program timeout, meaning it could not successfully connect to the network.
Open Socket	Here the mote schedules a connection to the manager. The mote is initializing a TCP three-way handshake by sending a request to begin a session with the manager.
Bind Socket	If the manager returns an OK to the open socket request, the mote moves into the bind socket state. A socket must be bound before data can be streamed from mote to manager.
Set NetID	If the bind socket state returns a success, the set NetID state is entered. This sets the ID of the network with a constant (NETID) chosen before program launch– the default is 2425.
Get NetID	Stores the network ID chosen by the set NetID state as a variable.
Set JoinDC	Sets the join duty cycle. This variable goes between 1 and 255 and determines how quickly a mote joins the manager. Choosing a high duty cycle (255 -> 100%) will allow the network to form more rapidly but the motes will also use more power.
Join	The mote will attempt to search for a network with the same NetID and join it.
Request Service	Request new / changed service level – triggers on bandwidth change event.
SendTo	Final state of the mote. In this state, the mote will send the adcData Buffer to the manager whenever the data is available.

3.3 Data Handle, State Machine:

State	Description
NEW_PARAM	This state acts as a “double buffer” for parameter changes to the mote. It checks for updates to the sampling frequency and resolution at the beginning of the state machine. These parameters cannot be changed again until we return to the new parameter state. This ensures that key parameters do not change during the execution of the state machine as a result of multithreading.
ACQ	Here the mote acquires data from the ADC. It reads in the data from the UART to the adcBuffer. When this stage has completed, we have a buffer that is full of “RAW” adc data, but the FFT has not yet been calculated and appended to the buffer.
CALC	In this state we make use of some basic ARM math functions to fourier transform the raw data. It then appends this fft data to the adcBuffer, which is now ready to be transmitted to the manager. The transmission of data also begins in this state.
TX	TX is the transmission state, and the mote will remain here until it detects that the entire ADC data buffer has been successfully sent.
WAIT	Following transmission of the data, the mote remains in the wait state until a flag is set indicating that more data is available for reading. The mote then returns to the NEW_PARAM phase. This cycle will continue until the program closes or the mote loses connection to the network.

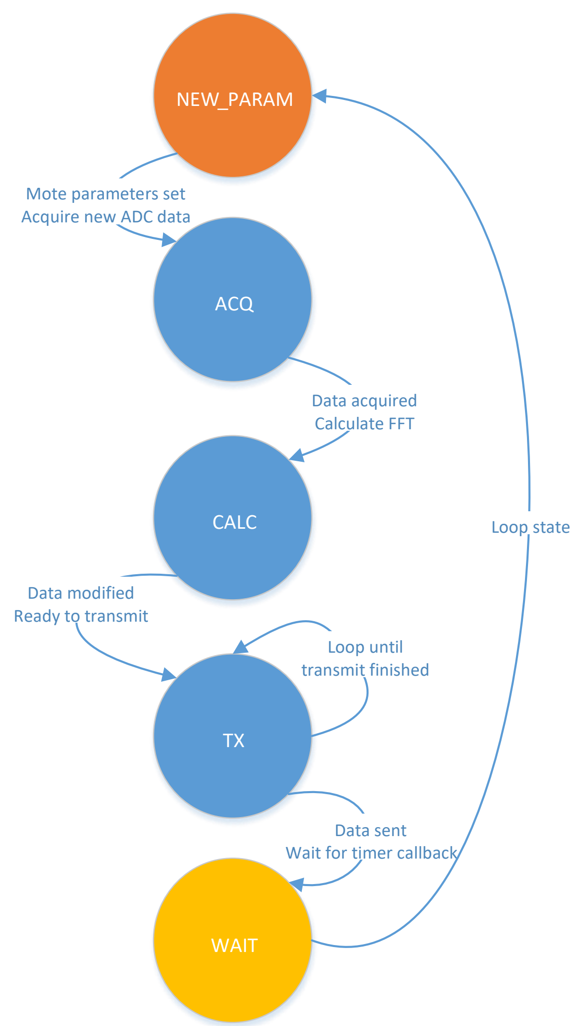


***Figure 5**: Data Handle State Machine*

Scheduler:

The scheduler timing for the embedded C code is outlined in fig.5. The code goes through a simple order of operations: Idle, Sample, FFT, and Transmit. However, because SmartMesh IP has a packet size limit of 90 bytes, any data sent that is greater than 90 bytes in length must be broken into smaller packets which are transmitted separately. This is further described in fig.6.

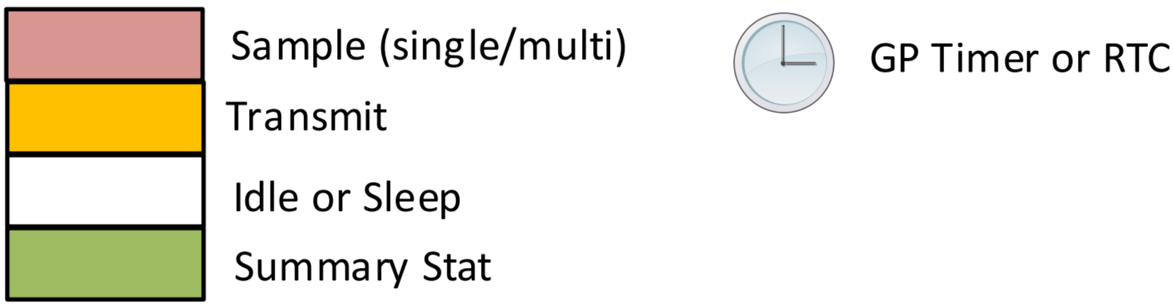


***Figure 6**: Legend*

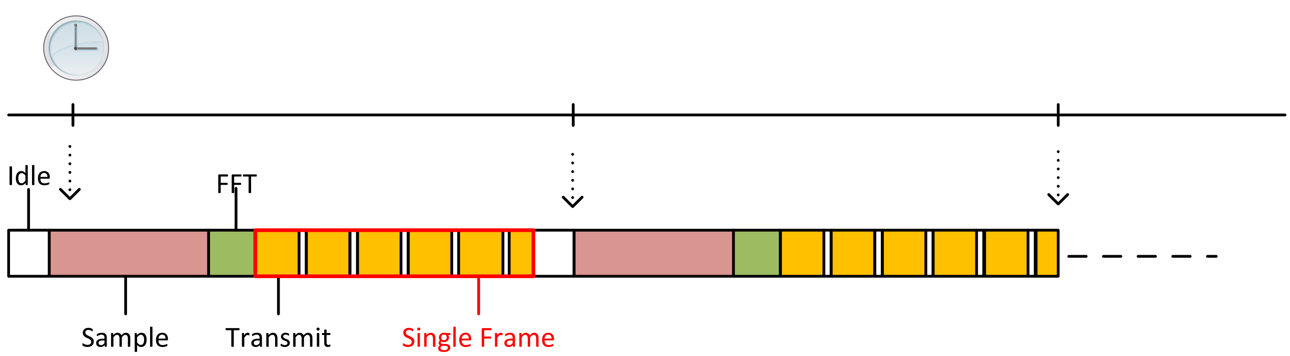


***Figure 7**: Timing Diagram*

Formatting data to be sent (shown for a sample size of 512 words):

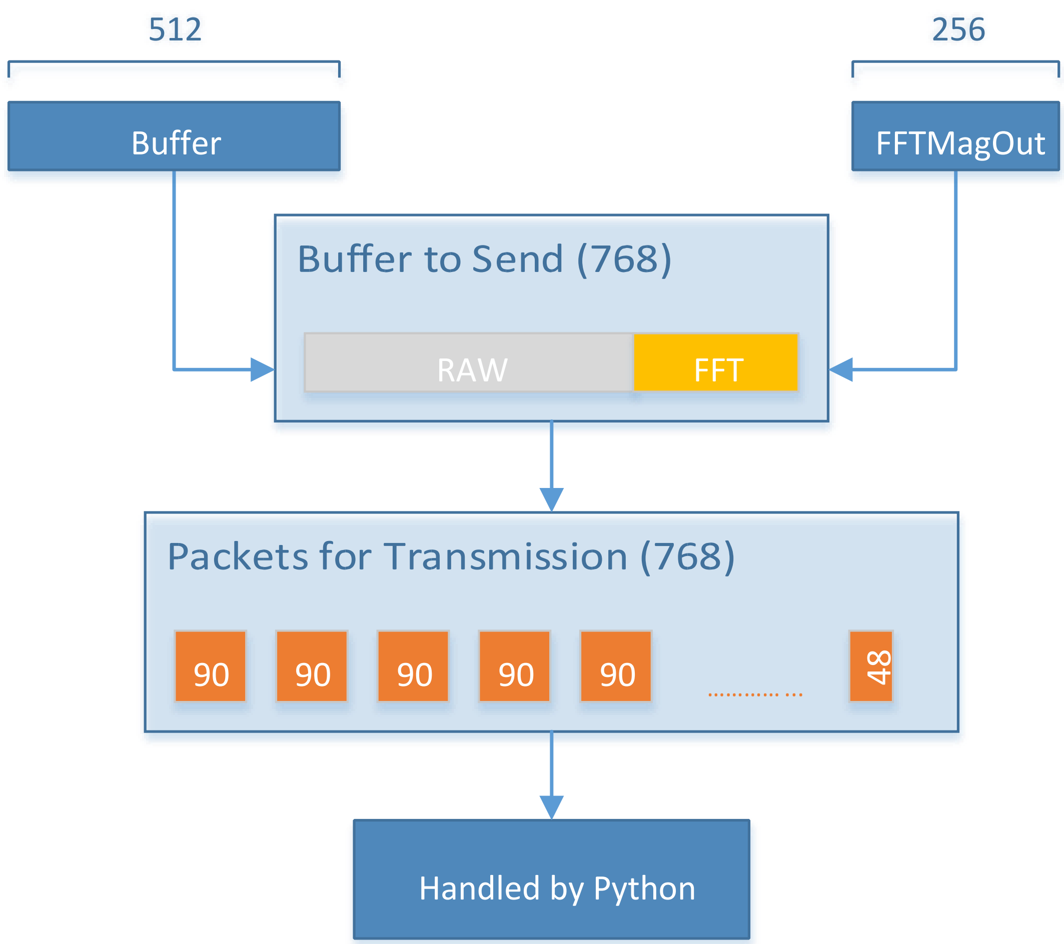


***Figure 8**: Packets for transmission*