WCBM Mote Firmware Guide

Wireless CbM Mote Firmware Guide (1.0.5R Project Revision)

Rev.7 Code base: C code

Created:	25/10/2018 2:21 P.M.
Author:	Tom Sharkey
Last Modified:	06/01/2025
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

Rev. 5 – Addition of download and debugging note, made document font consistent.

Rev. 6 – Updating descriptions after firmware version update to 0.5

Rev 7 – Minor fixes to diagrams

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)
  - shutdown.c (implements functions for putting microcontroller into low power sleep modes and waking it back up with a real-time clock)
  - ext_flash.c (functions which enable data storage and retrieval from the 1GB external flash chip which is connected to the microcontroller via SPI2)
  - 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.

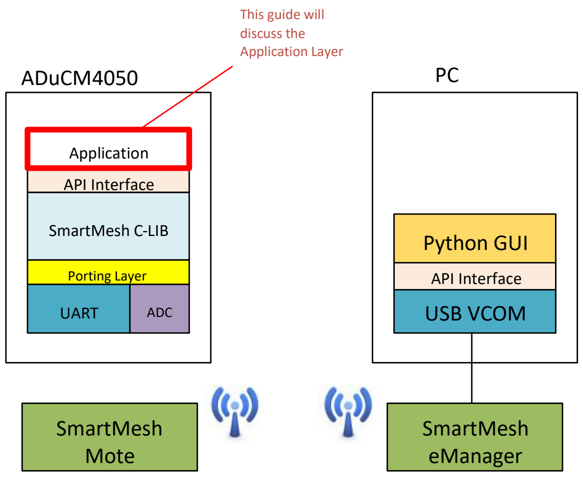


***Figure 1:** Hardware Overview*

2 Tool Requirements and Environment Setup

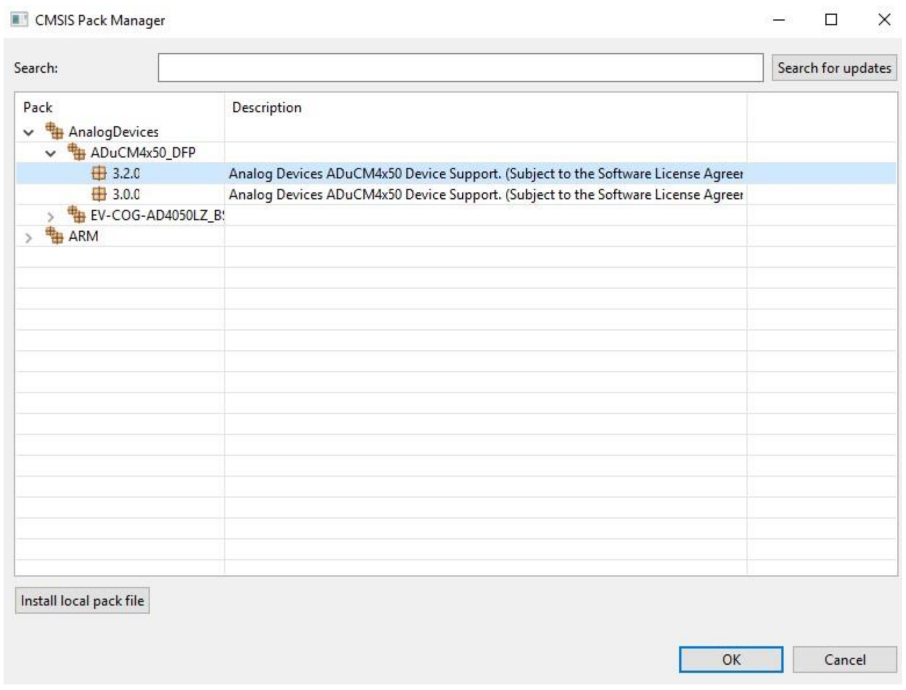
This project has been compiled and built in the IAR Embedded Workbench IDE version 8.22.2. The use of older versions may result in broken project options and/or project options not being fully transferred over and dependencies not working – these will then need to be manually set up.

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.

Download the project from the CBM Voyager GitHub (further details on this are included in the CBM Setup Guide). 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 3-8 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.

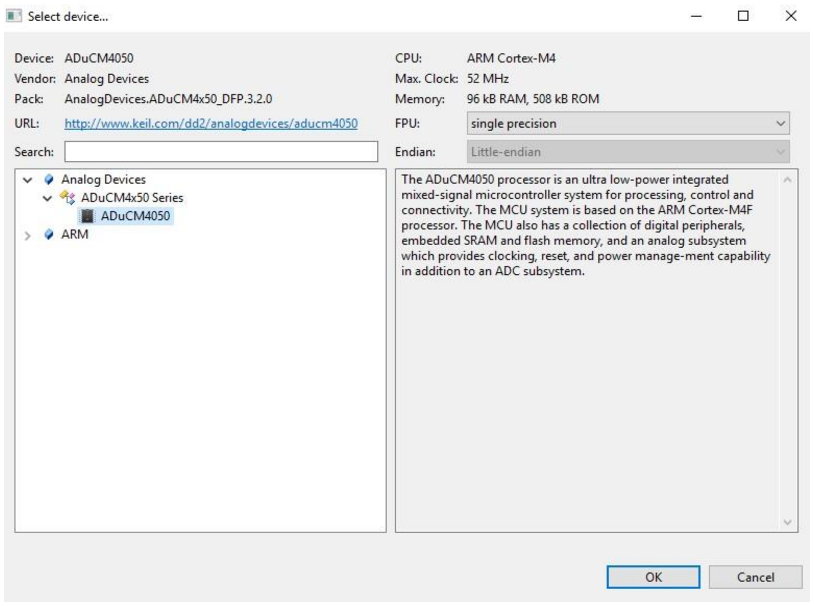
If the project cannot be opened successfully, follow all of the steps below to set up the project from scratch in IAR Embedded Workbench.

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.



Pack	Description
AnalogDevices	
ADuCM4x50_DFP	
3.2.0	Analog Devices ADuCM4x50 Device Support. (Subject to the Software License Agreer
3.0.0	Analog Devices ADuCM4x50 Device Support. (Subject to the Software License Agreer
EV-COG-AD4050LZ_B!	
ARM	

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



Select device...

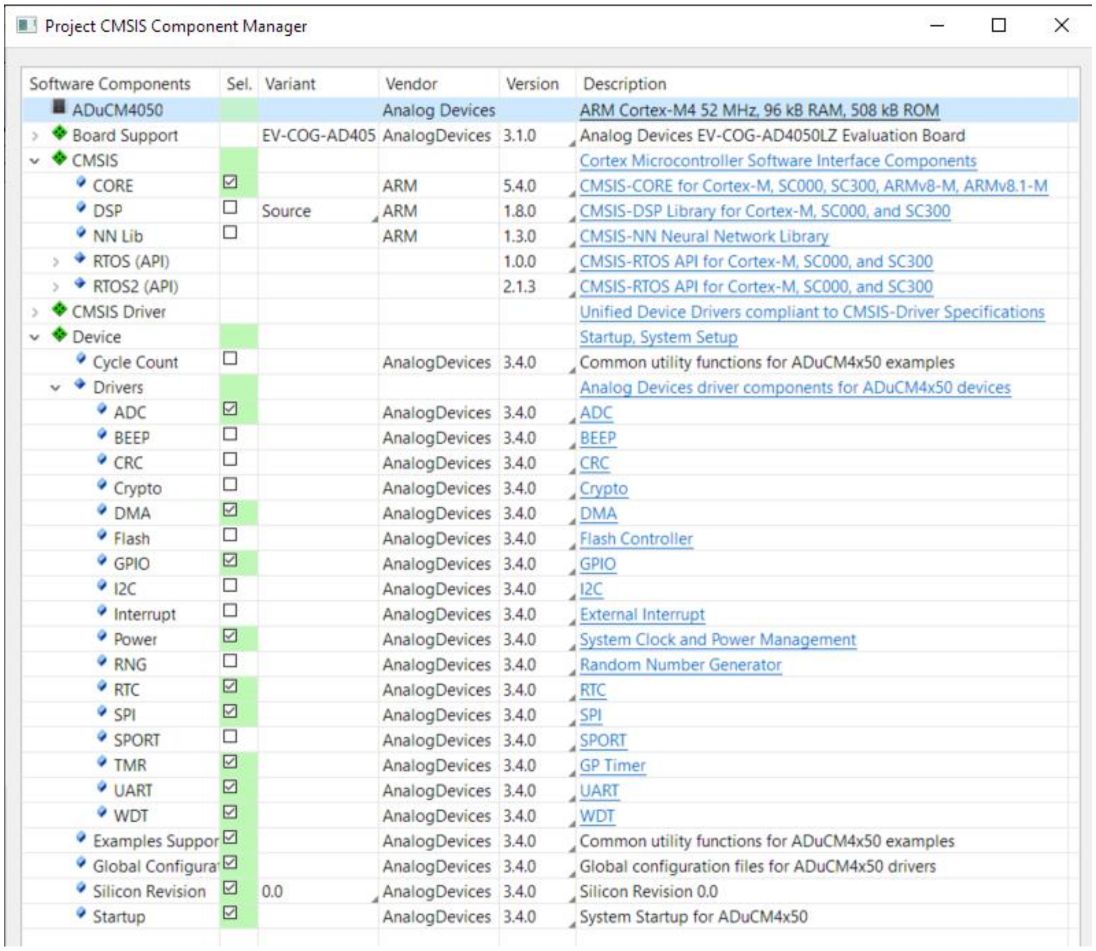
Device:	ADuCM4050	CPU:	ARM Cortex-M4
Vendor:	Analog Devices	Max. Clock:	52 MHz
Pack:	AnalogDevices.ADuCM4x50_DFP.3.2.0	Memory:	96 kB RAM, 508 kB ROM
URL:	http://www.keil.com/dd2/analogdevices/aducm4050	FPU:	single precision
Search:		Endian:	Little-endian

- Analog Devices
  - ADuCM4x50 Series
    - ADuCM4050
- ARM

The ADuCM4050 processor is an ultra low-power integrated mixed-signal microcontroller system for processing, control and connectivity. The MCU system is based on the ARM Cortex-M4F processor. The MCU also has a collection of digital peripherals, embedded SRAM and flash memory, and an analog subsystem which provides clocking, reset, and power manage-ment capability in addition to an ADC subsystem.

OK Cancel

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.

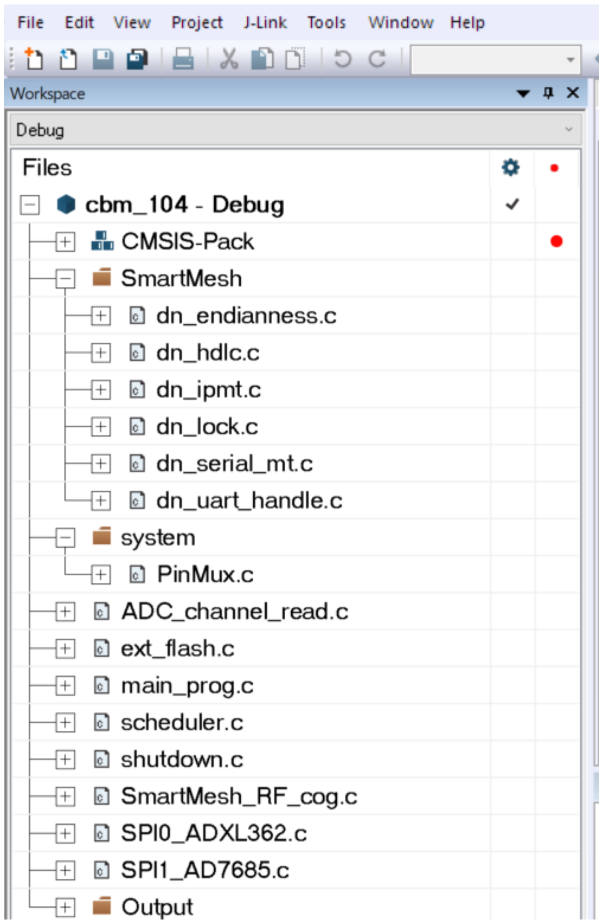


Project CMSIS Component Manager

Software Components	Sel.	Variant	Vendor	Version	Description
ADuCM4050			Analog Devices		ARM Cortex-M4 52 MHz, 96 kB RAM, 508 kB ROM
Board Support		EV-COG-AD405	AnalogDevices	3.1.0	Analog Devices EV-COG-AD4050LZ Evaluation Board
CMSIS					Cortex Microcontroller Software Interface Components
CORE	☑		ARM	5.4.0	CMSIS-CORE for Cortex-M, SC000, SC300, ARMv8-M, ARMv8.1-M
DSP	☐	Source	ARM	1.8.0	CMSIS-DSP Library for Cortex-M, SC000, and SC300
NN Lib	☐		ARM	1.3.0	CMSIS-NN Neural Network Library
RTOS (API)				1.0.0	CMSIS-RTOS API for Cortex-M, SC000, and SC300
RTOS2 (API)				2.1.3	CMSIS-RTOS API for Cortex-M, SC000, and SC300
CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifications
Device					Startup, System Setup
Cycle Count	☐		AnalogDevices	3.4.0	Common utility functions for ADuCM4x50 examples
Drivers					Analog Devices driver components for ADuCM4x50 devices
ADC	☑		AnalogDevices	3.4.0	ADC
BEEP	☐		AnalogDevices	3.4.0	BEEP
CRC	☐		AnalogDevices	3.4.0	CRC
Crypto	☐		AnalogDevices	3.4.0	Crypto
DMA	☑		AnalogDevices	3.4.0	DMA
Flash	☐		AnalogDevices	3.4.0	Flash Controller
GPIO	☑		AnalogDevices	3.4.0	GPIO
I2C	☐		AnalogDevices	3.4.0	I2C
Interrupt	☐		AnalogDevices	3.4.0	External Interrupt
Power	☑		AnalogDevices	3.4.0	System Clock and Power Management
RNG	☐		AnalogDevices	3.4.0	Random Number Generator
RTC	☑		AnalogDevices	3.4.0	RTC
SPI	☑		AnalogDevices	3.4.0	SPI
SPORT	☐		AnalogDevices	3.4.0	SPORT
TMR	☑		AnalogDevices	3.4.0	GP Timer
UART	☑		AnalogDevices	3.4.0	UART
WDT	☑		AnalogDevices	3.4.0	WDT
Examples Suppor	☑		AnalogDevices	3.4.0	Common utility functions for ADuCM4x50 examples
Global Configura	☑		AnalogDevices	3.4.0	Global configuration files for ADuCM4x50 drivers
Silicon Revision	☑	0.0	AnalogDevices	3.4.0	Silicon Revision 0.0
Startup	☑		AnalogDevices	3.4.0	System Startup for ADuCM4x50

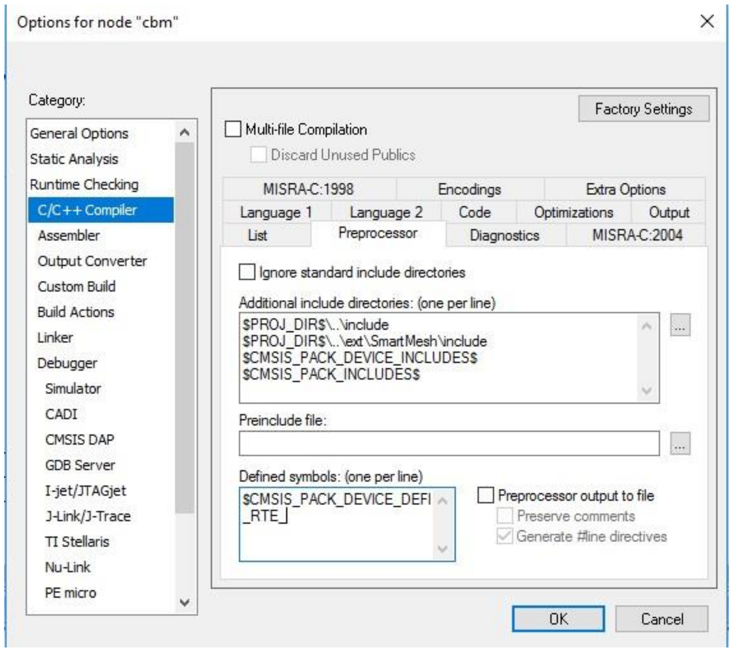
9. Save the project in the 'project' folder. The source files will then be at the correct level relative to the project.

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.



```
cbm_104 - Debug
  CMSIS-Pack
  SmartMesh
    dn_endianness.c
    dn_hdlc.c
    dn_ipmt.c
    dn_lock.c
    dn_serial_mt.c
    dn_uart_handle.c
  system
    PinMux.c
  ADC_channel_read.c
  ext_flash.c
  main_prog.c
  scheduler.c
  shutdown.c
  SmartMesh_RF_cog.c
  SPI0_ADXL362.c
  SPI1_AD7685.c
  Output
```

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



Multi-file Compilation
Discard Unused Publics
Ignore standard include directories
Additional include directories: (one per line)

```
$PROJ_DIR$\..\include
$PROJ_DIR$\..\ext\SmartMesh\include
$CMSIS_PACK_DEVICE_INCLUDES$
$CMSIS_PACK_INCLUDES$
```

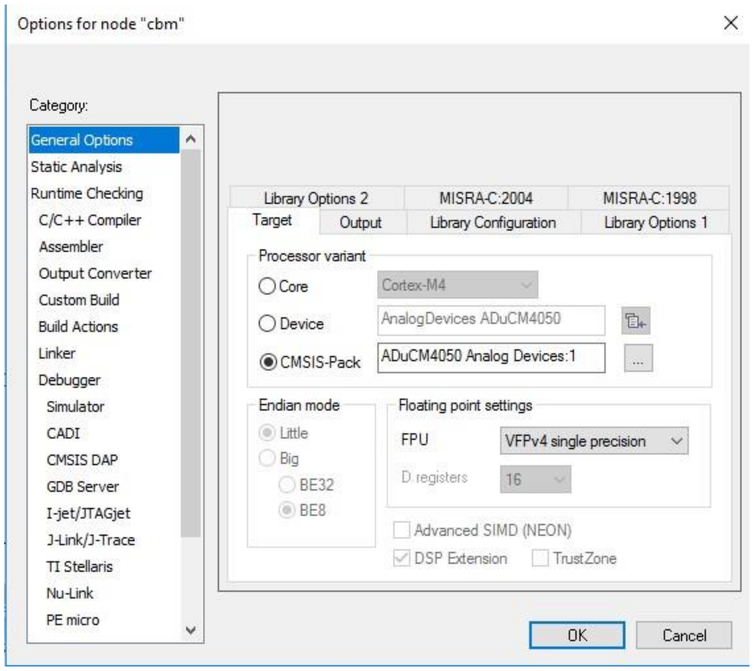
Preinclude file:

Defined symbols: (one per line)

```
$CMSIS_PACK_DEVICE_DEFI
_RTE_
```

Preprocessor output to file
Preserve comments
[✓ Generate #line directives]

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



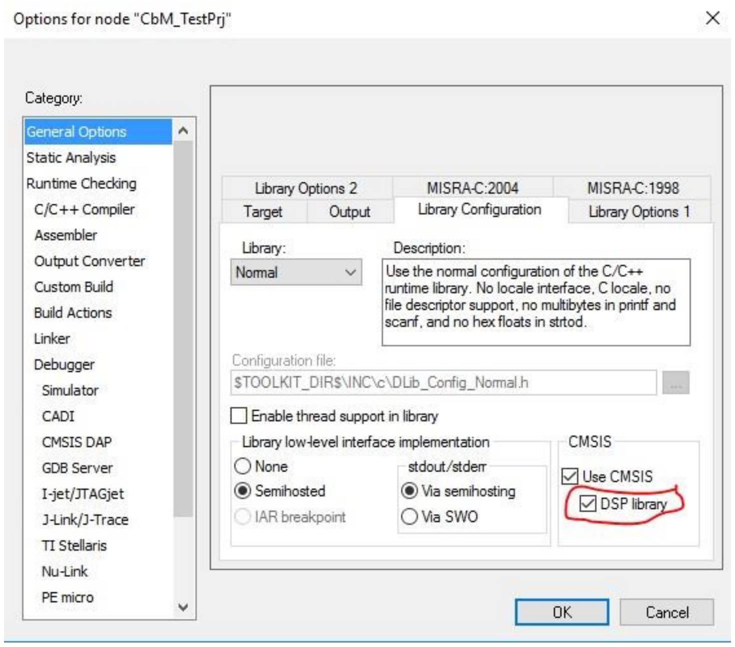
Processor variant
Core: Cortex-M4
Device: AnalogDevices ADuCM4050
[✓ CMSIS-Pack]: ADuCM4050 Analog Devices:1

Endian mode
[✓ Little]
Big
BE32
[✓ BE8]

Floating point settings
FPU: VFPv4 single precision
D registers: 16

Advanced SIMD (NEON)
[✓ DSP Extension]
TrustZone

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



Library: Normal

Description: Use the normal configuration of the C/C++ runtime library. No locale interface, C locale, no file descriptor support, no multibytes in printf and scanf, and no hex floats in strtod.

Configuration file: $TOOLKIT_DIR$\INC\c\DLib_Config_Normal.h

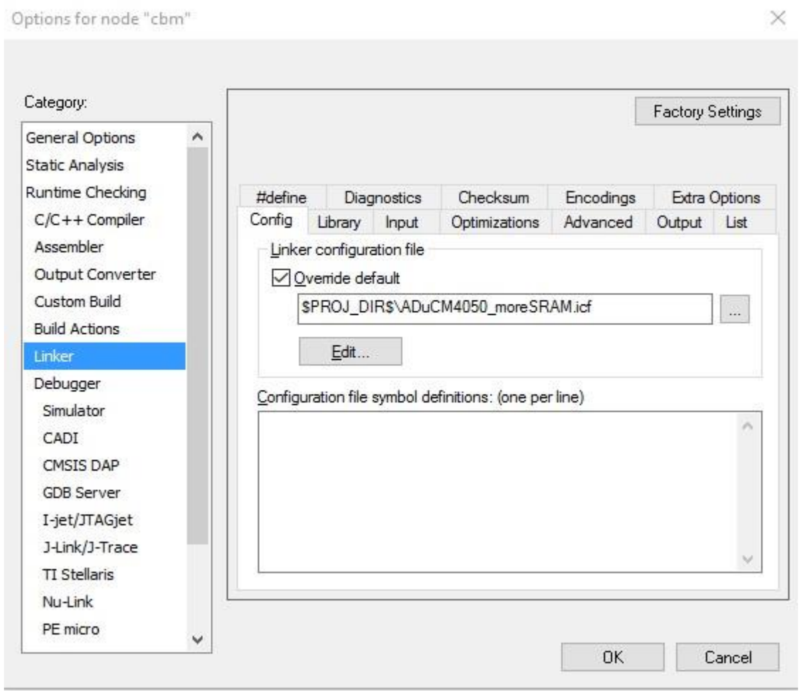
☐ Enable thread support in library

Library low-level interface implementation: ○ None [✓ ● Semihosted] ○ IAR breakpoint

stdout/stderr: [✓ ● Via semihosting] ○ Via SWO

CMSIS: ☑ Use CMSIS ☑ DSP library

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



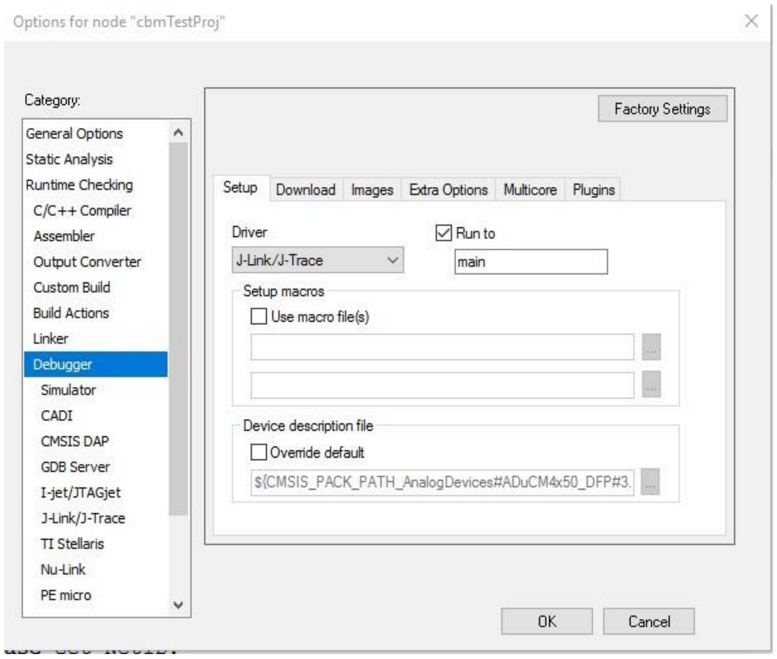
Linker configuration file

☑ Override default

$PROJ_DIR$\ADuCM4050_moreSRAM.icf

Configuration file symbol definitions: (one per line)

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.



Driver: J-Link/J-Trace

[✓] Run to: main

Setup macros

[ ] Use macro file(s)

Device description file

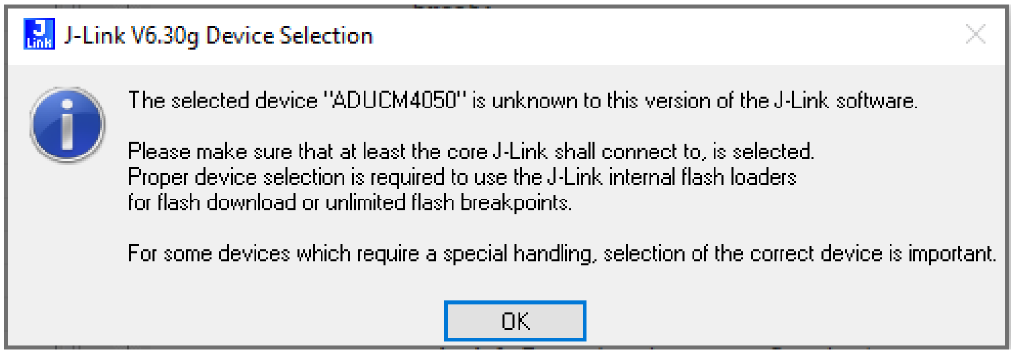
[ ] Override default

$(CMSIS_PACK_PATH_AnalogDevices#ADuCM4x50_DFP#3.

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.

Note, the first time you download and debug this program, you may get the following prompt:



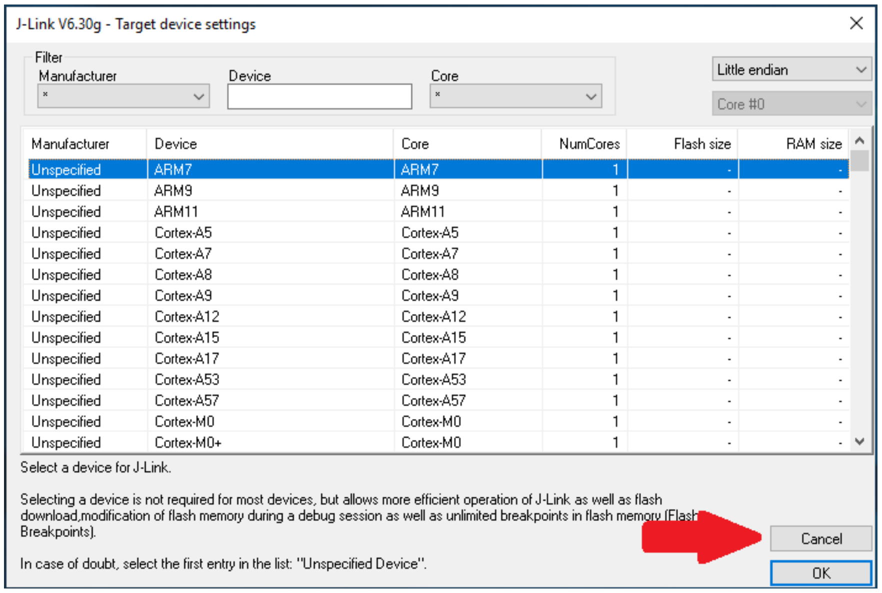
J-Link V6.30g Device Selection

The selected device "ADUCM4050" is unknown to this version of the J-Link software.

Please make sure that at least the core J-Link shall connect to, is selected.
Proper device selection is required to use the J-Link internal flash loaders
for flash download or unlimited flash breakpoints.

For some devices which require a special handling, selection of the correct device is important.

- Select OK, and proceed to the next page



J-Link V6.30g - Target device settings

Filter: Manufacturer *, Device (empty), Core *; Little endian; Core #0

Manufacturer	Device	Core	NumCores	Flash size	RAM size
Unspecified	ARM7	ARM7	1	-	-
Unspecified	ARM9	ARM9	1	-	-
Unspecified	ARM11	ARM11	1	-	-
Unspecified	Cortex-A5	Cortex-A5	1	-	-
Unspecified	Cortex-A7	Cortex-A7	1	-	-
Unspecified	Cortex-A8	Cortex-A8	1	-	-
Unspecified	Cortex-A9	Cortex-A9	1	-	-
Unspecified	Cortex-A12	Cortex-A12	1	-	-
Unspecified	Cortex-A15	Cortex-A15	1	-	-
Unspecified	Cortex-A17	Cortex-A17	1	-	-
Unspecified	Cortex-A53	Cortex-A53	1	-	-
Unspecified	Cortex-A57	Cortex-A57	1	-	-
Unspecified	Cortex-M0	Cortex-M0	1	-	-
Unspecified	Cortex-M0+	Cortex-M0	1	-	-

Select a device for J-Link.

Selecting a device is not required for most devices, but allows more efficient operation of J-Link as well as flash download,modification of flash memory during a debug session as well as unlimited breakpoints in flash memory (Flash Breakpoints).

In case of doubt, select the first entry in the list: "Unspecified Device".

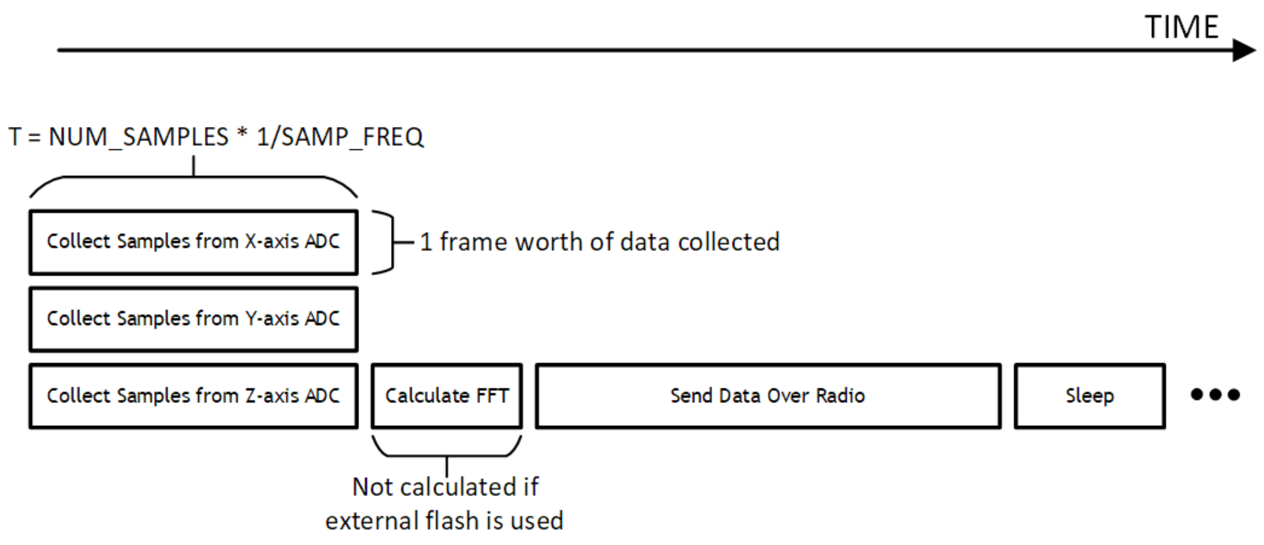
- Select Cancel. It is not necessary to specify a device. The flashing process will begin once this menu has been closed.

3 High Level Overview of Operation

A summary of the high-level operation of Voyager 3 can be seen in the below figure. Note, the sequence of events shown in this figure are collectively known as a **Stage**. The code goes through a simple order of operations: Idle, Sample, FFT, Transmit and optionally Sleep. 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 which is explained further in Section 5.4.

When a mote is connected to the manager via the GUI, Stages are repeated until the GUI is disconnected.

If the mote is connected to the manager through the Terminal Mode, the user passes in the argument NUM_STAGES to control how many times to repeat the sequence.


TIME
T = NUM_SAMPLES * 1/SAMP_FREQ
Collect Samples from X-axis ADC
1 frame worth of data collected
Collect Samples from Y-axis ADC
Collect Samples from Z-axis ADC
Calculate FFT
Send Data Over Radio
Sleep
Not calculated if external flash is used

VOYAGER 3 TIMELINE OF EVENTS. THIS COLLECTION OF EVENTS IS CALLED A STAGE

5 Code Structure

The function of the main program is to:

- Initiate a connection between mote and manager
- Sample acceleration data via ADC
- Store data in an external flash if the number of ADC samples per frames is greater than 1024
- Transmit raw and FFT data
- Put microcontroller into a low-power sleep mode for user-defined number of seconds
- 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.

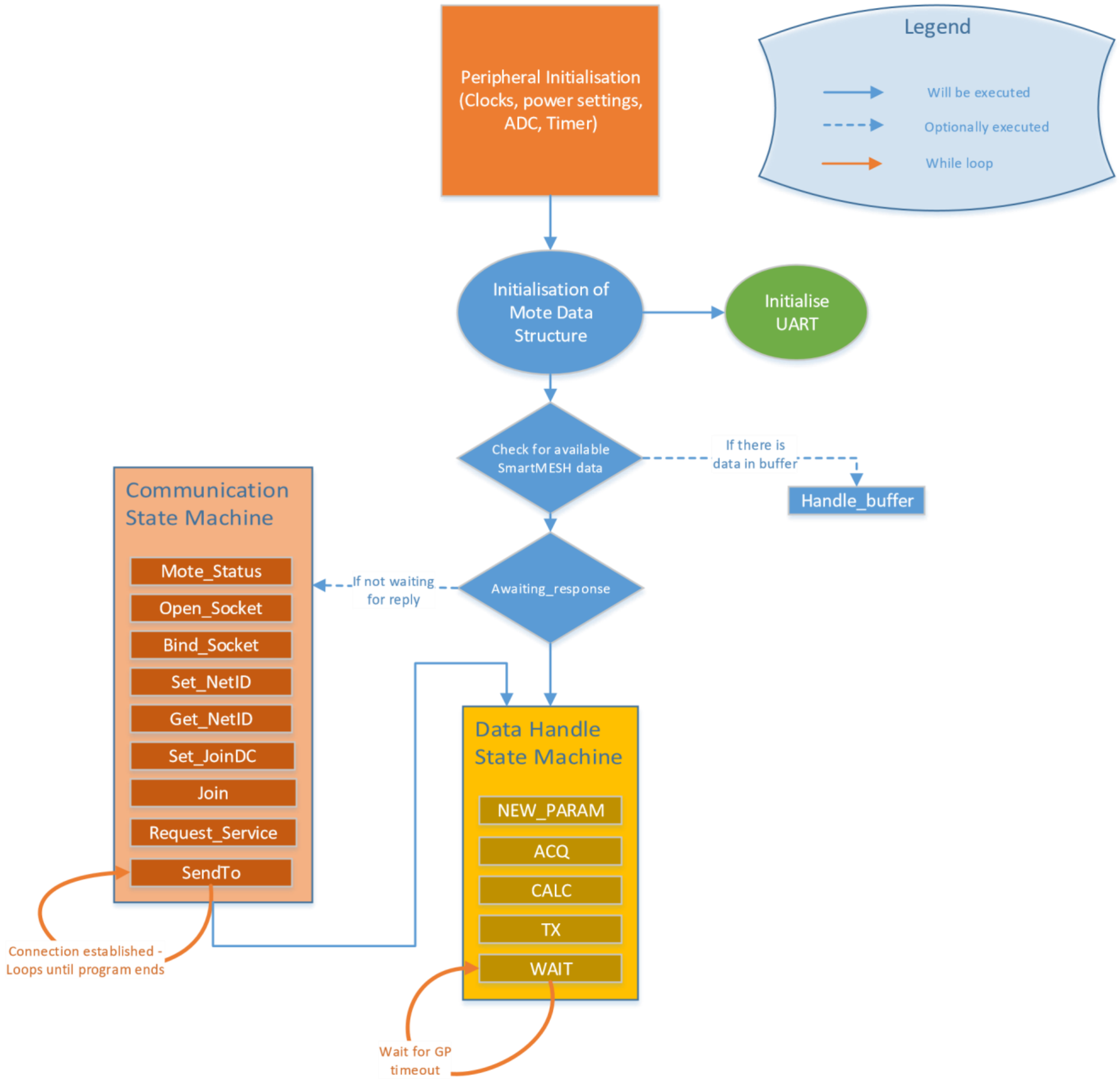

Legend
Will be executed
Optionally executed
While loop
Peripheral Initialisation (Clocks, power settings, ADC, Timer)
Initialisation of Mote Data Structure
Initialise UART
Check for available SmartMESH data
If there is data in buffer
Handle_buffer
Communication State Machine
Mote_Status
Open_Socket
Bind_Socket
Set_NetID
Get_NetID
Set_JoinDC
Join
Request_Service
SendTo
If not waiting for reply
Awaiting_response
Data Handle State Machine
NEW_PARAM
ACQ
CALC
TX
WAIT
Connection established - Loops until program ends
Wait for GP timeout

Figure 2: *High Level Diagram*

5.1 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).

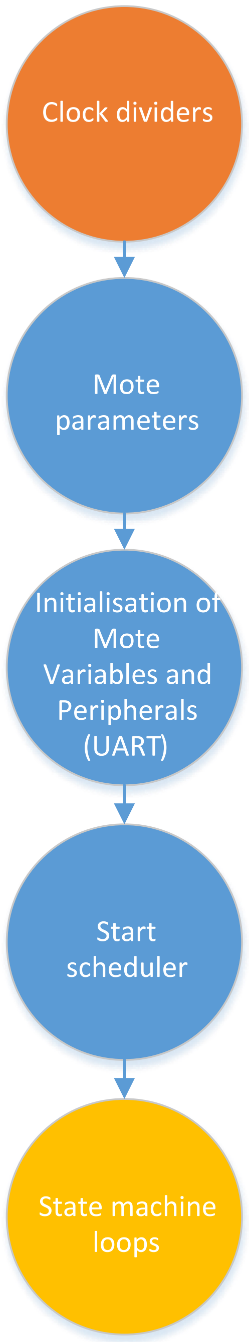


Figure 3: *Initialisation*

5.2 Communication State Machine:

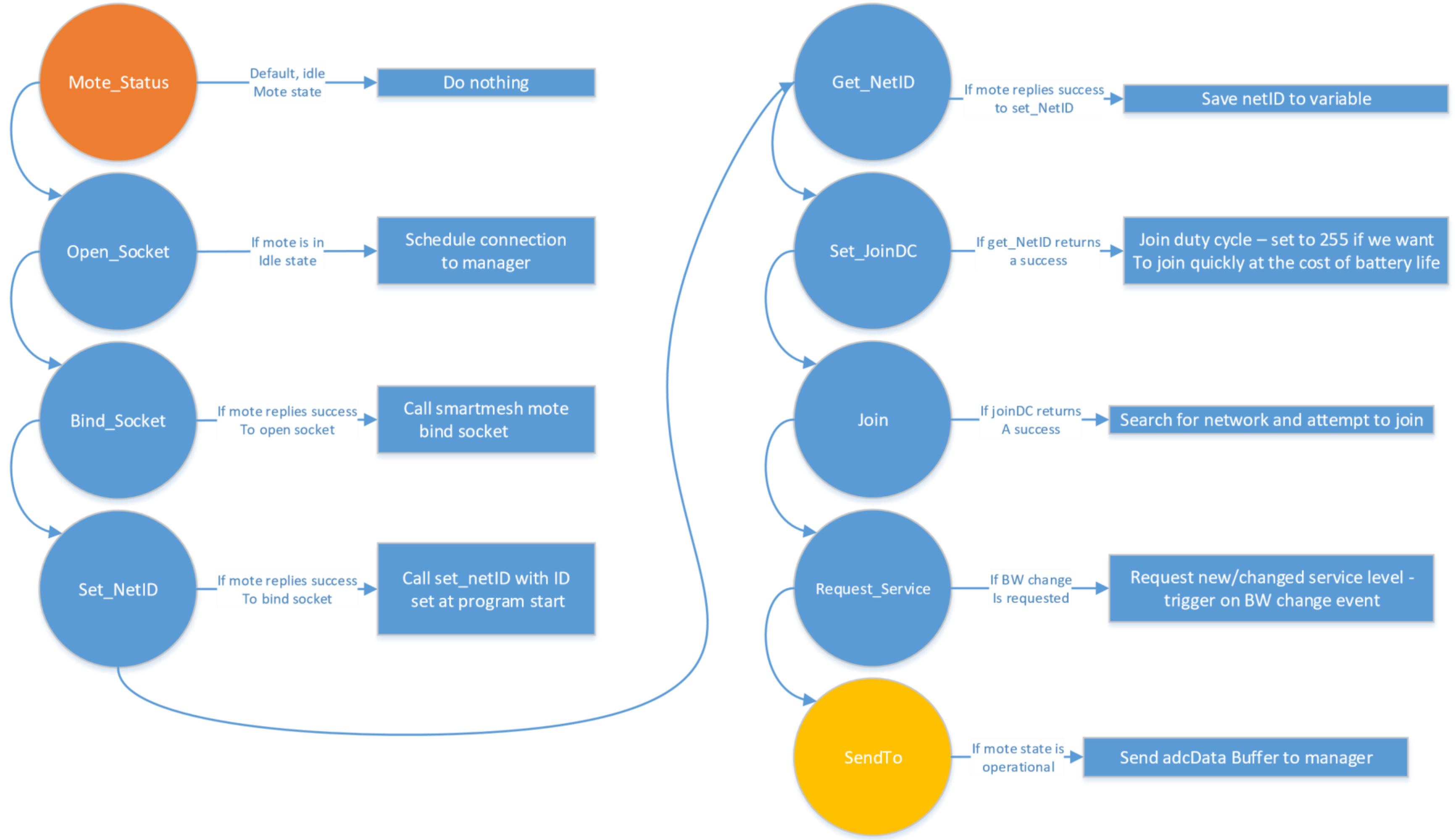
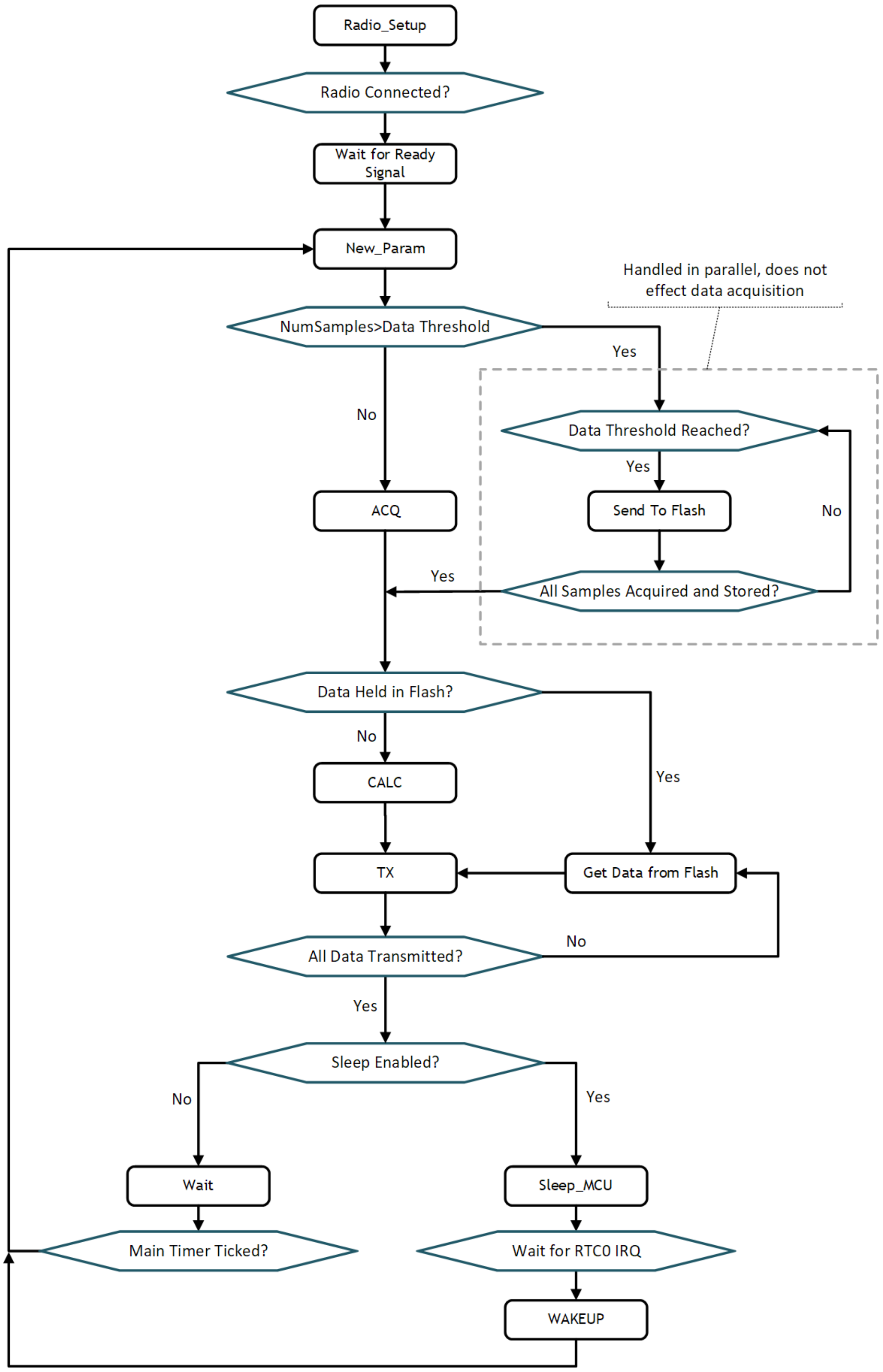

Mote_Status
Default, idle Mote state
Do nothing
Open_Socket
If mote is in Idle state
Schedule connection to manager
Bind_Socket
If mote replies success To open socket
Call smartmesh mote bind socket
Set_NetID
If mote replies success To bind socket
Call set_netID with ID set at program start
Get_NetID
If mote replies success to set_NetID
Save netID to variable
Set_JoinDC
If get_NetID returns a success
Join duty cycle – set to 255 if we want To join quickly at the cost of battery life
Join
If joinDC returns A success
Search for network and attempt to join
Request_Service
If BW change Is requested
Request new/changed service level - trigger on BW change event
SendTo
If mote state is operational
Send adcData Buffer to manager

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.

5.3 Data Handle, State Machine:

State	Description
RADIO SETUP	The microcontroller will wait in this state until the communication state machine reaches its SendTo state.
WAIT FOR READY	The microcontroller will remain in this state until it gets a ready signal from the manager so that it is not collecting samples which will never be observed.
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. The microcontroller will also not progress from this state unless it has received a ready signal from the manager.
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.
SEND TO FLASH	If the user sets the number of samples per frame to be greater than 1024 then the 1GB external flash will be used. Data is sent to the external flash via SPI2 port. Up to 1/3GB can be collected per axis at the user defined sampling frequency because the data storage operation happens in parallel with the data sampling operation. Note, if external flash is used then FFT calculations will not be performed and only the raw data will be transmitted over the radio.
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. Note, if external flash is used then FFT calculations will not be performed and only the raw data will be transmitted over the radio.
GET DATA FROM FLASH	If the user sets the number of samples per frame to be greater than 1024 then the 1GB external flash will be used. Data will be read from the flash one page at a time (1024 samples per page) into the microcontroller's RAM and transmitted via the radio.
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	This state is only entered if sleep duration has been defined to be zero seconds. Following transmission of the data, the mote enters the wait state. The mote remains in the wait state until a flag is set indicating that a timer has expired.
SLEEP MCU	This state is only entered if sleep duration has been defined to be greater than zero seconds. Following transmission of the data, the mote enters the low power sleep state. Upon entry to sleep a real-time clock (RTC0) is programmed and begins counting up to the target sleep duration. The minimum granularity for sleep duration is one second because RTC0 runs at 1Hz.
WAKEUP	This state re-enables certain functions that were disabled prior to entering the sleep state


Radio_Setup
Radio Connected?
Wait for Ready Signal
New_Param
NumSamples>Data Threshold
Handled in parallel, does not effect data acquisition
Yes
No
Data Threshold Reached?
Yes
Send To Flash
No
ACQ
Yes
All Samples Acquired and Stored?
Data Held in Flash?
No
CALC
Yes
TX
Get Data from Flash
No
All Data Transmitted?
Yes
Sleep Enabled?
No
Yes
Wait
Sleep_MCU
Main Timer Ticked?
Wait for RTC0 IRQ
WAKEUP

5.4 Packet Formatting

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

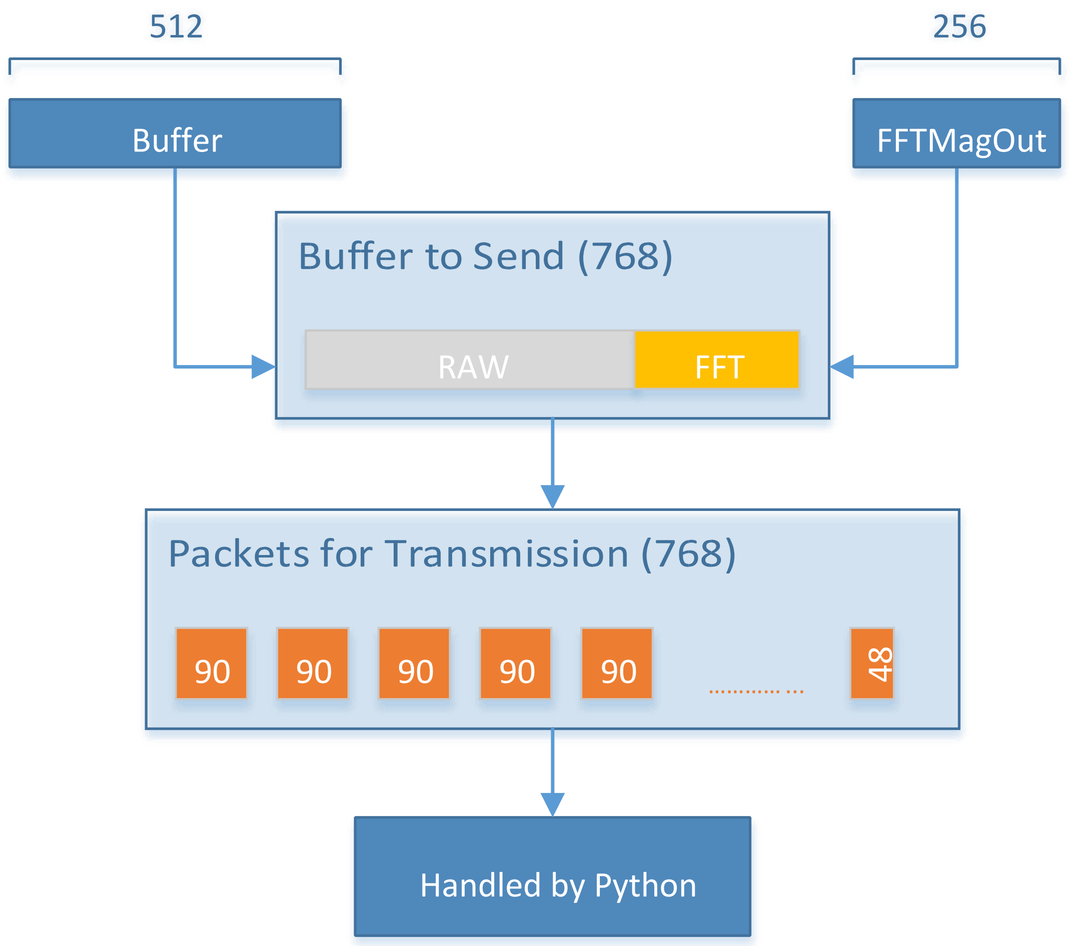


***Figure 8**: Packets for transmission*

If external flash is used then the FFT calculations will not be performed and only the raw data shall be transmitted.

6 External Flash

Voyager has a 1GB NAND flash memory device connected to the microcontroller via SPI2. The part name of the flash is "Micron MT29F1G01ABAFDWB-IT" and its datasheet can be found here.

Firmware Version 0.5 and onwards makes use of this flash memory which allows up to 1GB to be collected in a single Stage, without interruption to the selected sampling frequency.

If the user selects that more than 1024 samples should be collected per frame, then the firmware automatically stores and retrieves data to/from flash.

NOTE: If external flash is used then FFT will not be calculated.

The flash chip is divided into 64 pages. Each page contains 1024 blocks. Each block contains 2048B of data and 128B of error correction codes which are automatically enabled.

Each axis has 1/3 of the flash memory dedicated to it, as shown in **Error! Reference source not found.**:

FLASH DIVISION