SPACELAB

TTC 2.0 Documentation

TTC 2.0 Documentation
SpaceLab, Universidade Federal de Santa Catarina, Florianópolis – Brazil

August 2024

TTC 2.0 Documentation
August, 2024

Project Chief:
Eduardo Augusto Bezerra <eduardo.bezerra@spacelab.ufsc.br>

Authors:
Gabriel Mariano Marcelino <gabriel.marcelino@spacelab.ufsc.br>
André Martins Pio de Mattos <andre.mattos@spacelab.ufsc.br>
Miguel Boing <miguel.boing@spacelab.ufsc.br>

Contributing Authors:
Sara Vega Martinez
Vitória Beatriz Bianchin
Yan Castro de Azeredo

Revision Control:

Version	Author	Changes	Date
0.0	G. M. Marcelino	Document creation	2021/04/01
0.1	G. M. Marcelino	First release	2021/06/16
0.2	G. M. Marcelino	General improvements and updates	2022/05/17
0.3	G. M. Marcelino, M. Boing	General improvements and updates	2022/11/04
0.4	M. Boing	Firmware updates	2023/03/19
0.5	M. Boing	Firmware updates	2023/11/01
0.6	M. Boing	Firmware updates	2024/03/14
1.0	M. Boing	Firmware updates	2024/08/08

List of Figures

List of Tables

Nomenclature

EM   *Engineering Model.*

EPS   *Electric Power System.*

FEC   *Forward Error Correction.*

FM   *Flight Model.*

GFSK   *Gaussian Frequency Shift Keying.*

GMSK   *Gaussian Minimum Shift Keying.*

GOLDS   *Global Open coLlecting Data System.*

HAL   *Hardware Abstraction Layer.*

I^{2}C   *Inter-Integrated Circuit.*

JTAG   *Joint Test Action Group.*

NGHam   *Next Generation Ham Radio.*

OBDH   *On-Board Data Handling.*

PCB   *Printed Circuit Board.*

RAM   *Random Access Memory.*

RF   *Radio Frequency.*

RS   *Reed-Solomon.*

RTOS   *Real Time Operating System.*

SDR   *Software-Defined Radio.*

SPI   *Serial Peripheral Interface.*

SRAM   *Static Random Access Memory.*

TTC   *Telemetry, Tracking and Command.*

UART   *Universal Asynchronous Receiver / Transmitter.*

uC   *MicroController.*

UFSC   *Universidade Federal de Santa Catarina.*

Contents

CHAPTER 1

Introduction

The TTC 2.0 is a Telemetry, Tracking and Command module designed for nanosatellites. It is one of the service modules developed for the GOLDS-UFSC CubeSat Mission [1].

The module is a direct upgrade from the TTC of FloripaSat-1 [2], which grants a flight heritage rating. The improvements focus on providing a cleaner and more generic implementation than the previous version, more reliability in software and hardware implementations, and adaptations for the new mission requirements. All the project, source, and documentation files are available freely on a GitHub repository [3] under the GPLv3 (firmware) and CERN OHLv2 (hardware) licenses.
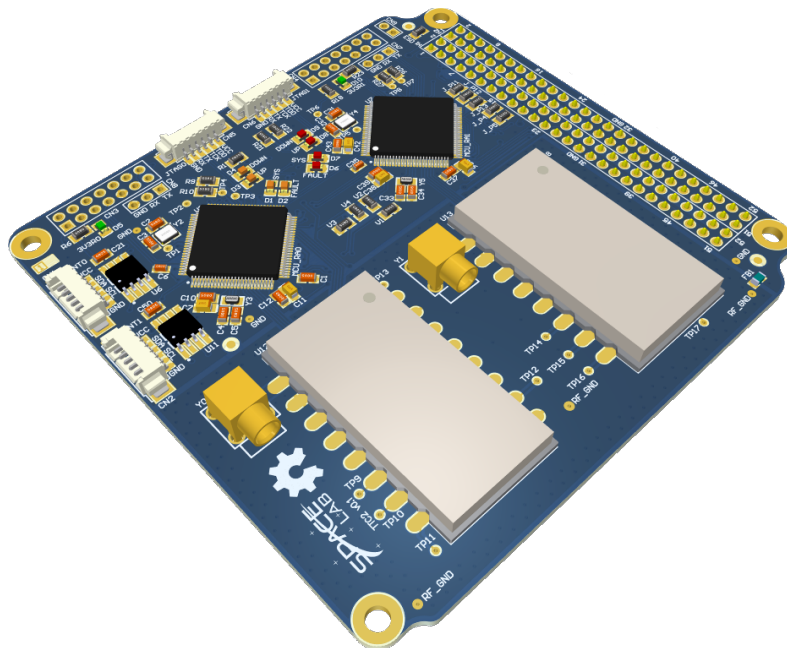
Figure 1.1: 3D view of the TTC 2.0 PCB.

1.1 Product tree

The product tree of the TTC 2.0 module can be divided into three branches: hardware, firmware, and documentation. A diagram of the product tree is available in Figure 1.2.

1
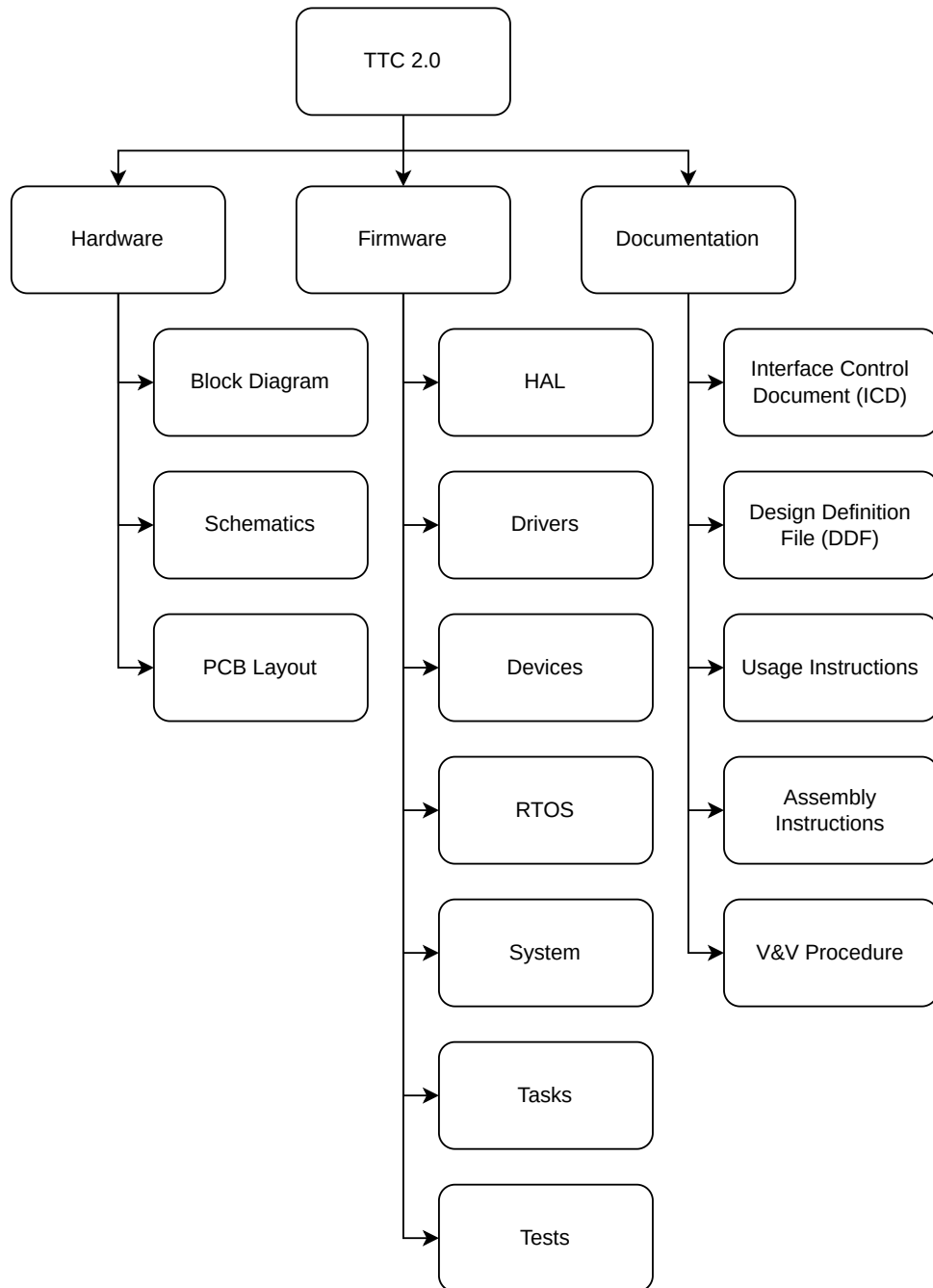
Figure 1.2: Product tree of the TTC 2.0 module.

CHAPTER 2

Hardware

This chapter presents a description of the hardware project of the TTC 2.0 module. As the primary reference, a block diagram can be seen in Figure 2.1. Also, a 3D model of both sides of the PCB is available in Figure 2.2.
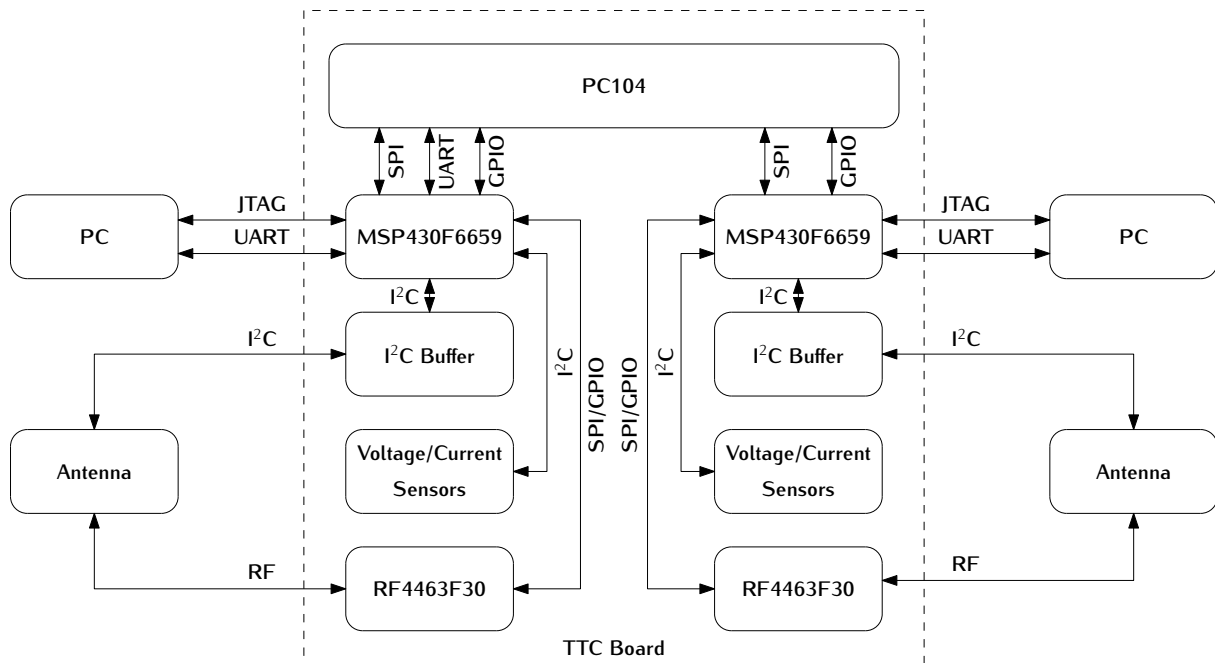
Figure 2.1: Block diagram of the TTC 2.0 hardware.

The following sections present a further description of the hardware project.

2.1 Specifications

The TTC 2.0 has two microcontrollers (MSP430FR6659) that run at a clock of 32 MHz, a RAM of 64 kB (SRAM), a flash memory of 512 kB for code storage, and another one of 128 kB for data storage. The TTC 2.0 also has current, voltage, and temperature sensors and two radio modules for RF communication. A brief description of the general specifications of the TTC 2.0 module is available below:

- **Microcontroller**: MSP430F6659

(a) Upper surface of the TTC 2.0 PCB.

(b) Lower surface of the TTC 2.0 PCB.

Figure 2.2: Surfaces of the 3D model of the TTC 2.0 board.

- **Clock**: 32 MHz

- **Memories**:

    - RAM: 64 kB (SRAM)
    - Flash: 512 kB (code) and 128 kB (storage)

- **Sensors**: Voltage, current and temperature

- **Modulation**: (G)FSK and/or (G)MSK

- **Baudrate**: 1200 to 9600 bps

- **Frequency**: 145-146 MHz, 435-438 MHz, and/or 450 MHz bands

- **Protocol**: NGHam

- **Interfaces**: UART, I^{2}C and SPI

- **Mass**: 73 g

- **Main bus**: PC-104 standard

2.2 Electrical interfaces

Below is a description of all available electrical interfaces of the TTC 2.0 module. The interfaces can be divided into PC-104 bus and dedicated interfaces.

2.2.1 PC-104 bus

The connector PC-104 is a junction of two double rows 26 pin headers (*SSW-126-04-G-D* by default). These connectors create a solid 104-pin interconnection across the different satellite modules. Table 2.1 provides the connector pinout[1] for the pins that are connected to the module. A reference of the pin's position can also be seen in Figure 2.3, and a description of the signal is available in Table 2.2.
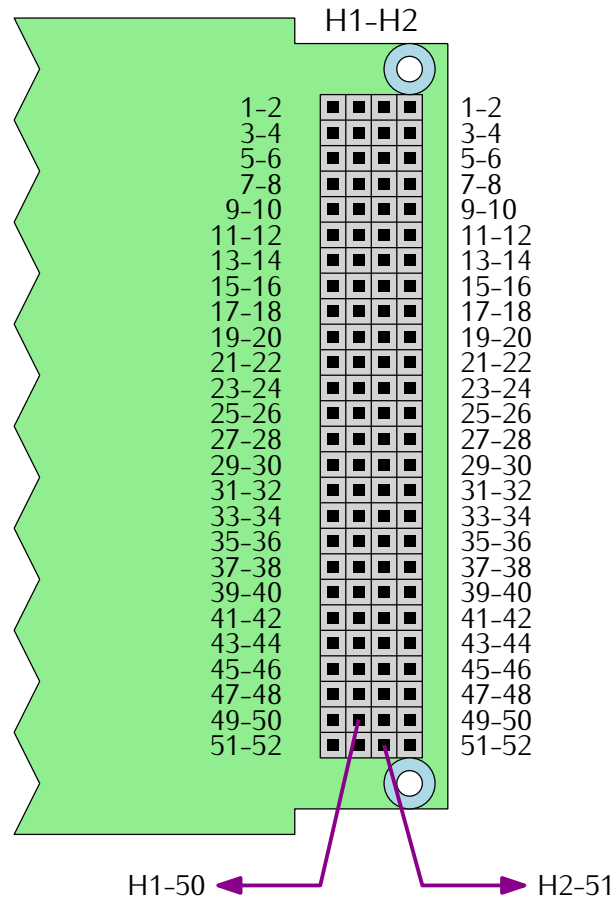
Figure 2.3: Reference diagram of the PC-104 bus (top view of a generic module).

This project's distribution pattern of pins is a mix of multiple patterns from CubeSat module manufacturers, like GomSpace, ISISpace, and EnduroSat. Some pins are positioned to attend to specific project requirements; this way, it is possible that the adopted pattern is only partially compatible with some commercial modules or other custom projects.

2.2.2 Dedicated electrical interfaces
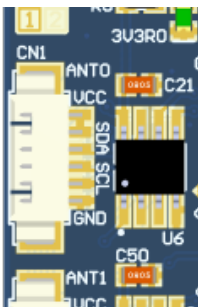
The module also contains other electrical interfaces through pin headers and PicoBlade connectors that are used to flash code and as debug port of microcontrollers (CN5, CN3, CN4, CN6) and serial communication for a status report and log messages (CN7, CN8). The C9 connector switches the TTC 2.0 power supply to the MSP-FET. The CN1 and CN2 are antenna interfaces (I^2C bus and power), and Y0, Y1 are MCX connectors to be used

---

[1]This pinout is simplified since additional interfaces were omitted.
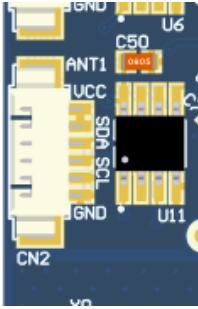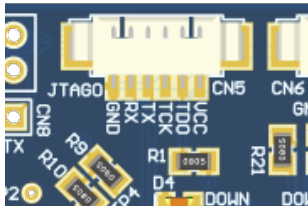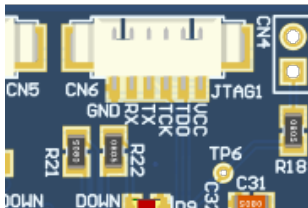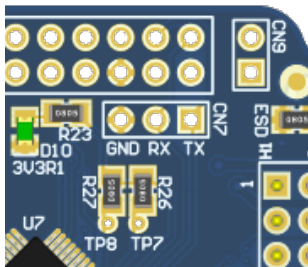
Pin Row	H1 Odd	H1 Even	H2 Odd	H2 Even
1–2	-	-	-	-
3–4	-	-	-	-
5–6	-	-	RA_1_UART_RX	-
7–8	GPIO_6	GPIO_7	RA_1_UART_TX	GPIO_0
9–10	RA_1_SPI_INT	RA_1_EN	-	-
11–12	RA_0_SPI_INT	RA_0_EN	RA_1_SPI_MOSI	RA_1_SPI_CLK
13–14	-	-	RA_1_SPI_CS	RA_1_SPI_MISO
15–16	-	-	-	-
17–18	-	-	-	GPIO_1
19–20	-	GPIO_2	-	GPIO_3
21–22	-	-	-	GPIO_4
23–24	-	-	-	-
25–26	-	-	-	-
27–28	-	-	VCC_3V3	VCC_3V3
29–30	GND	GND	GND	GND
31–32	GND	GND	GND	GND
33–34	-	-	-	-
35–36	RA_0_SPI_CLK	-	VCC_3V3_ANT	VCC_3V3_ANT
37–38	RA_0_SPI_MISO	-	-	-
39–40	RA_0_SPI_MOSI	RA_0_SPI_CS	-	-
41–42	-	-	-	GPIO_5
43–44	-	-	-	-
45–46	-	-	-	-
47–48	-	-	-	-
49–50	VCC_5V_RA_0	VCC_5V_RA_0	-	-
51–52	VCC_6V_RA_1	VCC_6V_RA_1	-	-

Table 2.1: PC–104 bus pinout.
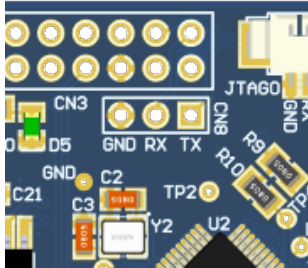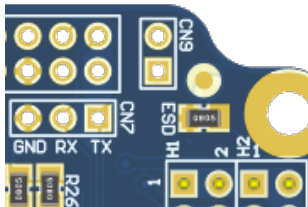
as the RF interface with the antennas. More details of each of these connectors, like the pinout, can be seen in Table 2.3.

Connector	Image	Interface	Type	Pins
CN1		I^{2}C	PicoBlade	3V3 3V3 I2C_SDA I2C_SCL GND GND

Continues on next page

CN2		I²C	PicoBlade	3V3 3V3 I2C_SDA I2C_SCL GND GND
CN5		JTAG	PicoBlade	3V3_JTAG TDO_TDI TCK UART_TX UART_RX GND
CN6		JTAG	PicoBlade	3V3_JTAG TDO_TDI TCK UART_TX UART_RX GND
CN7		UART	PinHeader	TX RX GND

Continues on next page

CN8		UART	PicoBlade	TX RX GND
CN9		Jumper	Pin Header	3V3_JTAG 3V3
CN3		JTAG	Pin Header	TDO_TDI 3V3_JTAG None None None None TCK None GND None None UART_TX None UART_RX

Continues on next page
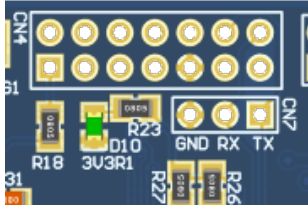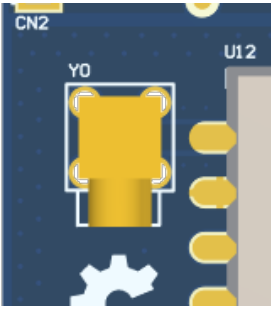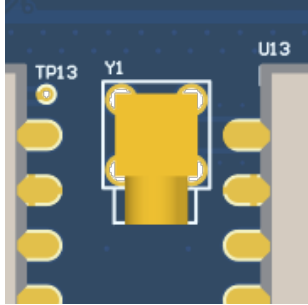
CN4		JTAG	Pin Header	TDO_TDI
				3V3_JTAG
				None
				None
				None
				None
				TCK
				None
				GND
				None
				None
				UART_TX
				None
				UART_RX
Y0		RF	MCX	RF_SIGNAL
				RF_GND
Y1		RF	MCX	RF_SIGNAL
				RF_GND

Table 2.3: Dedicated electrical interfaces.

2.3   Mechanical interfaces

The TTC 2.0 board has four mounting holes to fix the module into the CubeSat mechanical structure. These mounting holes are 3.2 mm in diameter and positioned on each corner of the PCB. These holes can be seen in Figures 2.2(a) and 2.2(b).

Signal	Pin(s)	Description
GND	H1–29/30/31/32, H2–29/30/31/32	Ground reference
VCC_3V3	H2–27, H2–28	TTC power supply (3,3 V)
VCC_3V3_ANT	H2–35, H2–36	Antenna power supply (3,3 V)
VCC_5V_RA_0	H1–49, H1–50	Radio 0 power supply (5 V)
VCC_6V_RA_1	H1–51, H1–52	Radio 1 power supply (6 V)
RA_0_SPI_CLK	H1–35	CLK signal of the radio 0 SPI bus
RA_0_SPI_MISO	H1–37	MISO signal of the radio 0 SPI bus
RA_0_SPI_MOSI	H1–39	MOSI signal of the radio 0 SPI bus
RA_0_SPI_CS	H1–40	CS signal of the radio 0 SPI bus
RA_0_SPI_INT	H1–11	INT signal of the radio 0 SPI bus
RA_1_SPI_CLK	H2–12	CLK signal of the radio 0 SPI bus
RA_1_SPI_MISO	H2–14	MISO signal of the radio 0 SPI bus
RA_1_SPI_MOSI	H2–11	MOSI signal of the radio 0 SPI bus
RA_1_SPI_CS	H1–13	CS signal of the radio 0 SPI bus
RA_1_SPI_INT	H1–9	INT signal of the radio 0 SPI bus
RA_1_UART_RX	H2–5	RX signal of the radio 1 UART
RA_1_UART_TX	H2–7	TX signal of the radio 1 UART
RA_0_EN	H1–11	Radio 0 power enable
RA_1_EN	H1–9	Radio 1 power enable
GPIO_N	H1–7/8/19, H2–8/18/20/22/42	GPIO pin (not used)

Table 2.2: PC–104 bus signal description.

2.4 Printed Circuit Board

This section presents some detailed information about the PCB of the TTC 2.0 module.

2.4.1 Dimensions

The TTC PCB dimensions follow the CubeSats industry standards defined by ISISpace and GomSpace, with a size of 89, 15 mm by 92, 13 mm [4]. The outline is specifically defined to improve the physical integration of modules and payloads into the mechanical structure. A drawing with the board dimensions is available in Figure 2.4.

2.4.2 Layout

The TTC 2.0 module contains only two layers: top and bottom, as presented in Figures 2.2(a) and 2.2(b).

To design the layout, the Altium Designer[2] software was used.

Simple PCBs with common manufacturing specifications were used to test and develop the module. These PCBs were defined as Engineering Models (EM). The Flight Model (FM, or the boards that fly with the satellite) of the boards have the following manufacturing specifications:

---

[2]https://www.altium.com/
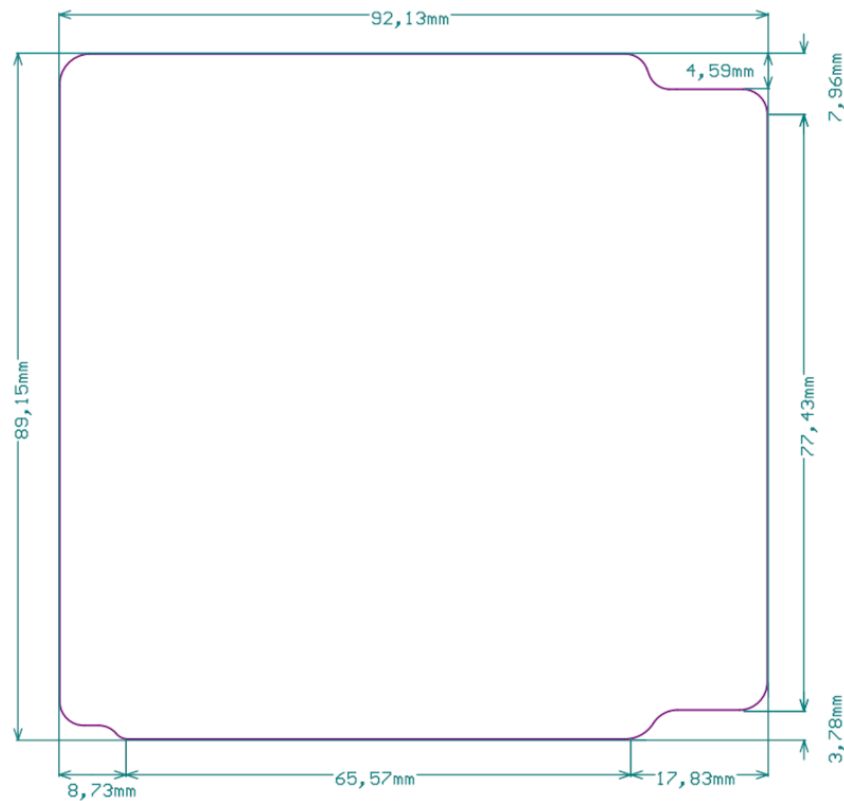
Figure 2.4: Board dimensions for the TTC 2.0 module.

- **PCB specs.**: IPC 6012 Class 3

- **PCB thickness**: 1,6 mm

- **Material**: TG170 FR-4

- **Surface finish**: ENIG

- **Board finish**: Conformal coating application

2.5 Peripherals

The TTC 2.0 have some peripherals besides the microcontrollers, like sensors, watchdog timers, and radio modules. These peripherals are better described in this section.

2.5.1 Power sensor

The board has power sensors (model INA226AQDGSRQ1 from Texas Instruments) that use an I^2C interface to communicate with the microcontroller. There are four of these sensors in the module, one for each microcontroller and one for each radio module. It monitors voltage and current from a given bus through a shunt resistor (with a resistance of $0,1\ \Omega$). The final measurement is continuous and made by an average of 128 measurements with an interval of $588\ \mu$s between each other.
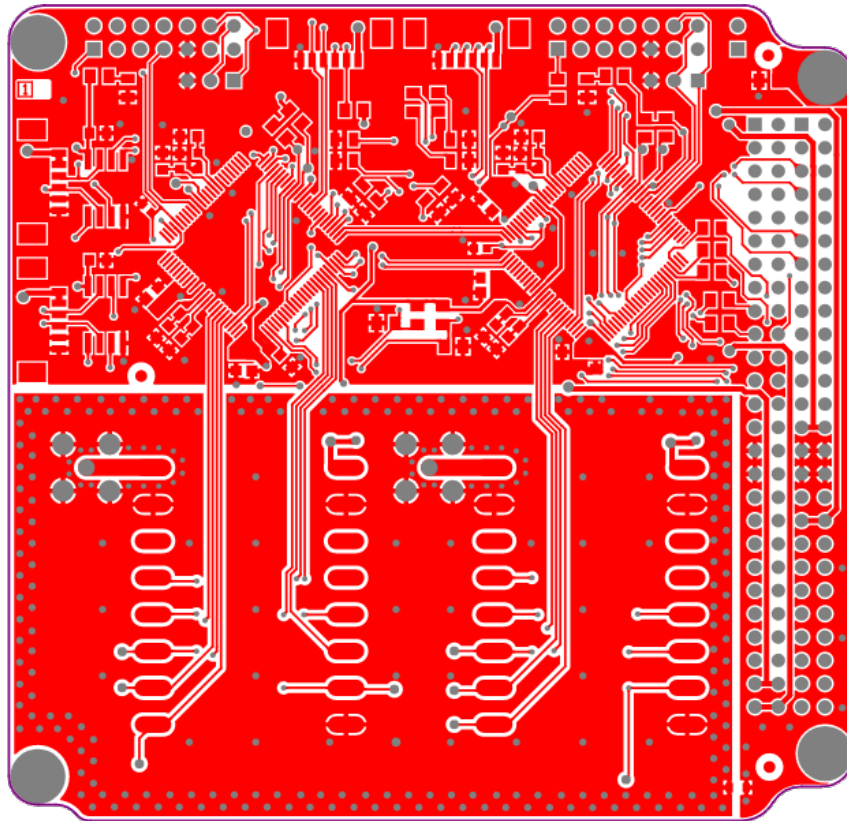
Figure 2.5: Top layer of the TTC 2.0 layout.

2.5.2   External watchdog timer

Besides the internal watchdog of the microcontroller, there is also a redundant external one. For that, the TPS3823 IC from Texas Instruments is used. It has an integrated voltage monitor and a watchdog timer, with a timeout of 1600 ms. The boards have two of these ICs, one for each microcontroller.

2.5.3   Radio module

One of the main components of the board is the radio module (NiceRF RF4463F30), which uses an SPI interface to be controlled by the microcontroller. The radio has a dedicated power source of 5V directly from the PC-104 bus. The output power of the RF signal generated by the radio is 30 dBm (1 W). The radio is half-duplex and has an integrated power amplifier and an RF switch, as shown in Figure 2.7.
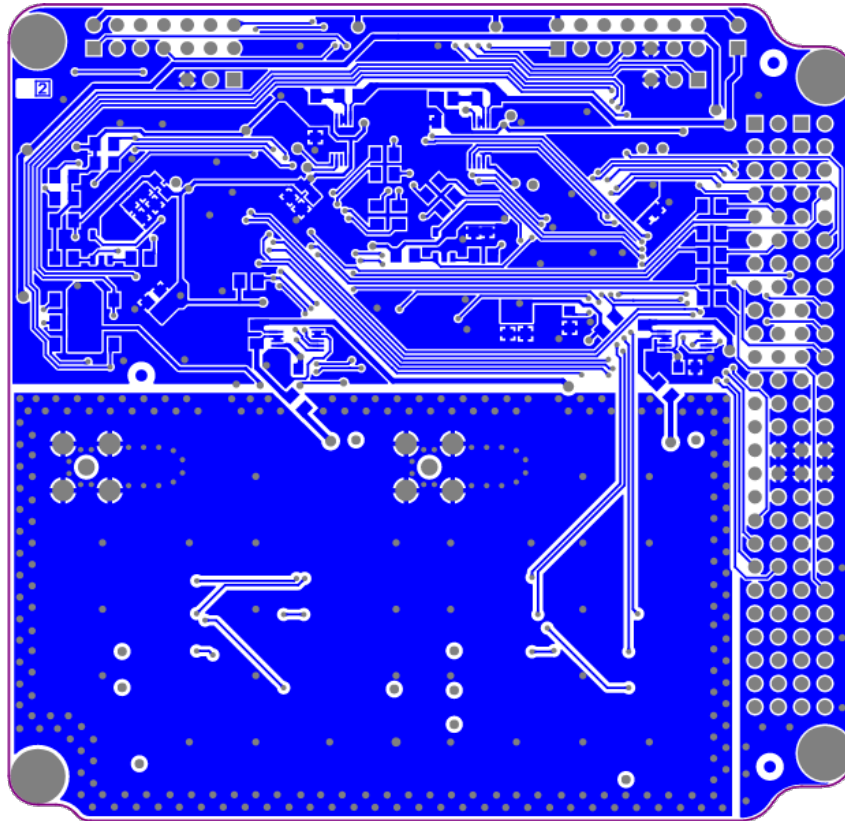
Figure 2.6: Bottom layer of the TTC 2.0 layout.
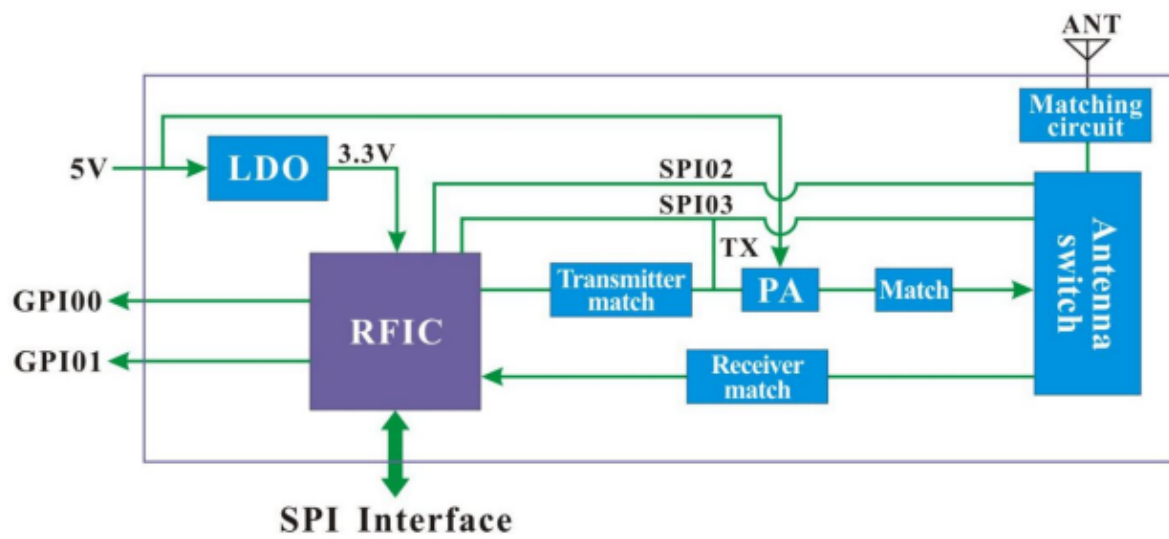
Figure 2.7: Block diagram of the RF4463F30 module.

CHAPTER 3

Firmware

This chapter describes the main characteristics of the firmware part of the TTC module.

3.1 Product tree

The product tree of the firmware part of the TTC 2.0 module is available in Figure 3.1. This product tree follows the architecture of the firmware, being divided according to the firmware layers.

3.2 Commands

To externally control and access the TTC module, some commands are available through the serial interfaces of the board. The commands are almost identical for both microcontrollers (except the command answering behavior) and are available on all interfaces. A list with the commands is available in Table 3.1. The format of the commands answers can be seen in Table 3.2.

ID	Name	Content	Interface
0	NOP	None	SPI
1	Read parameter	Command ID (1B) + Parameter ID (1B)	SPI
2	Write parameter	Command ID (1B) + Parameter ID (1B) + Value (4B)	SPI
3	Transmit packet	Command ID (1B) + Packet length (1B) + Packet (1-220B)	SPI/UART
4	Receive packet	Command ID (1B)	SPI

Table 3.1: List of command's requests.

ID	Name	Content
1	Read parameter	Command ID (1B) + Parameter ID (1B) + Value (4B)
2	Write parameter	None
3	Transmit packet	None
4	Receive packet	Command ID (1B) + Packet(1-220B)

Table 3.2: Format of the command's answers.

A description of each command is available below and the details about the protocol communication are available at section 4.5.

- Read parameter/variable (ID = 1): This command is used to read a parameter or variable of the TTC module (see Table 3.3).

15
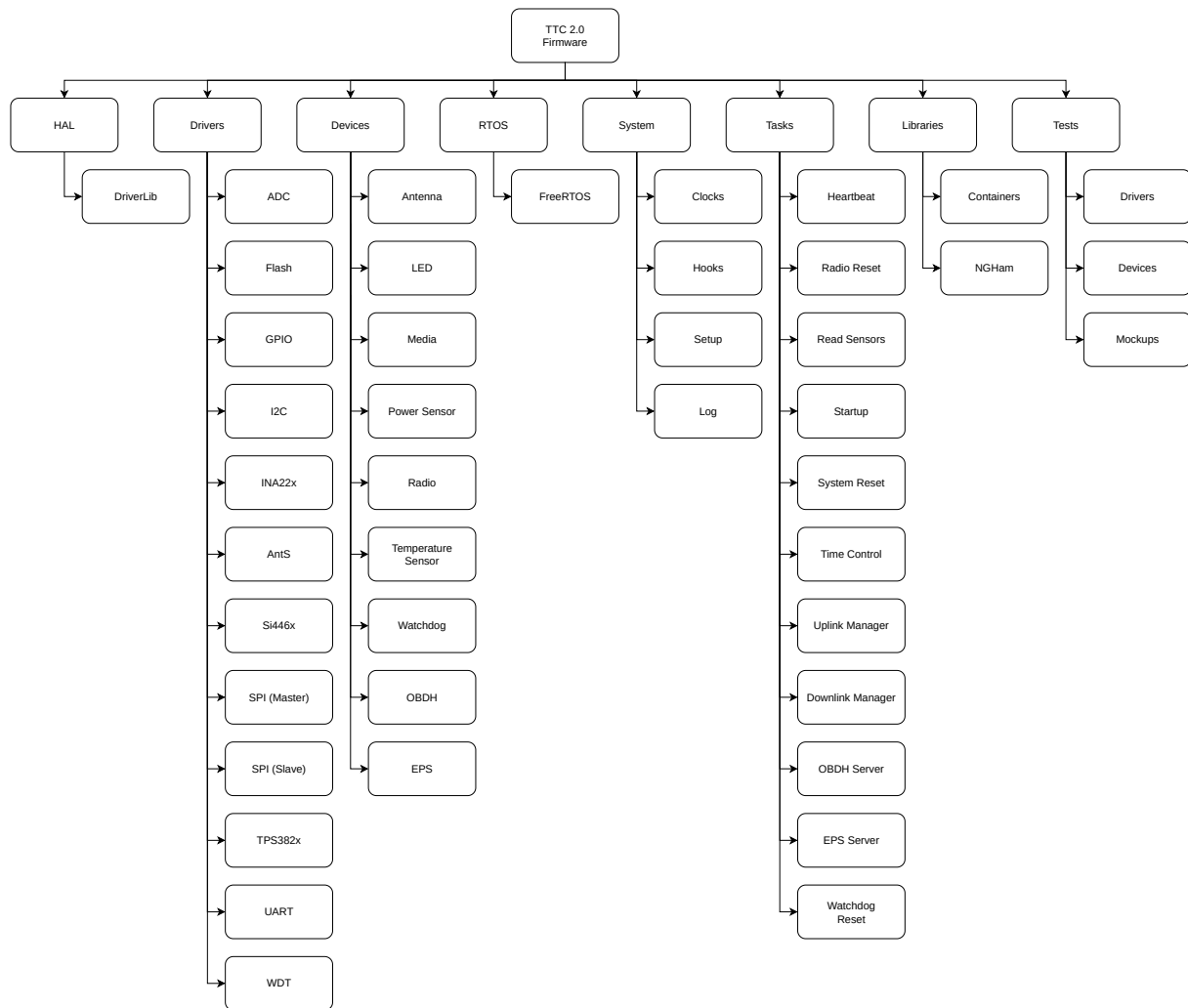
Figure 3.1: Product tree of the firmware of the TTC 2.0 module.

- Write parameter/variable (ID = 2): This command is used to write a value to a given parameter or variable when allowed (see Table 3.3).

- Transmit packet (ID = 3): This command is used to transmit a packet through the radio link of each microcontroller.

- Receive packet (ID = 4): This command is used to read a received packet through the radio link (stored in the internal FIFO of the microcontrollers of the TTC).

3.3   Variables and Parameters

A list of all the variables of TTC with their identification number (ID) and variable type that can be read from the sensors and peripherals can be seen in Table 3.3.

ID	Name/Description	Type	Access
0	Device ID (0xCC2A or 0xCC2B)	uint16	R
1	Hardware version	uint8	R

2	Firmware version (ex.: "v1.2.3" = 0x00010203)	uint32	R
3	Time counter in milliseconds	uint32	R
4	Reset counter	uint16	R
5	Last reset cause: – 0x00 = No interrupt pending – 0x02 = Brownout (BOR) – 0x04 = RST/NMI (BOR) – 0x06 = PMMSWBOR (BOR) – 0x08 = Wakeup from LPMx.5 (BOR) – 0x0A = Security violation (BOR) – 0x0C = SVSL (POR) – 0x0E = SVSH (POR) – 0x10 = SVML_OVP (POR) – 0x12 = SVMH_OVP (POR) – 0x14 = PMMSWPOR (POR) – 0x16 = WDT time out (PUC) – 0x18 = WDT password violation (PUC) – 0x1A = Flash password violation (PUC) – 0x1C = Reserved – 0x1E = PERF peripheral/configuration area fetch (PUC) – 0x20 = PMM password violation (PUC) – 0x22 to 0x3E = Reserved	uint8	R
6	Input voltage of the μC in mV	uint16	R
7	Input current of the μC in mA	uint16	R
8	Temperature of the μC in K	uint16	R
9	Input voltage of the radio in mV	uint16	R
10	Input current of the radio in mA	uint16	R
11	Temperature of the radio in K	uint16	R
12	Last valid command (uplink packet ID)	uint8	R
13	RSSI of the last valid telecommand	uint16	R
14	Temperature of the antenna module in K	uint16	R
15	Antenna module status bits: – Bit 15: The antenna 1 is deployed (0) or not (1) – Bit 14: Cause of the latest activation stop for antenna 1 – Bit 13: The antenna 1 deployment is active (1) or not (0) – Bit 11: The antenna 2 is deployed (0) or not (1) – Bit 10: Cause of the latest activation stop for antenna 2 – Bit 9: The antenna 2 deployment is active (1) or not (0) – Bit 8: The antenna is ignoring the deployment switches (1) or not (0) – Bit 7: The antenna 3 is deployed (0) or not (1) – Bit 6: Cause of the latest activation stop for antenna 3 – Bit 5: The antenna 3 deployment is active (1) or not (0) – Bit 4: The antenna system independent burn is active (1) or not (0) – Bit 3: The antenna 4 is deployed (0) or not (1) – Bit 2: Cause of the latest activation stop for antenna 4 – Bit 1: The antenna 4 deployment is active (1) or not (0)	uint16	R

	– Bit 0: The antenna system is armed (1) or not (0)		
16	Antenna deployment status (0=never executed, 1=executed)	uint8	R
17	Antenna deployment hibernation (0=never executed, 1=executed)	uint8	R
18	TX enable (0=off, 1=on)	uint8	R/W
19	TX packet counter	uint32	R
20	RX packet counter (valid packets)	uint32	R
21	TX packets available in the FIFO buffer	uint8	R
22	RX packets available in the FIFO buffer	uint8	R
23	Number of bytes of the first available packet in the RX buffer	uint16	R

Table 3.3: Variables and parameters of the TTC 2.0.

Each variable can be read or written using the commands "Read Parameter" and/or "Write Parameter". Some variables can just be read, as seen in the most right column of Table 3.3. When a variable is less than 32 bits long, it is left filled with zeros during a read or write operation (ex.: the value 0xAB becomes 0x000000AB).

3.4 Layers

The firmware flow of development goes from the low-level implementation (far right), with HA layer being register-level operation, to Tasks Layer with very abstract and high-level code.

3.4.1 Hardware Abstraction Layer

The HAL layer is the API DriverLib developed by Texas Instruments; it includes register manipulating functions to accelerate development. The TTC 2.0 uses HAL to handle GPIO operations and serial communications such as SPI, UART, and I^{2}C.

3.4.2 Drivers

Driver Layer is created to have the flexibility of the HAL layer but contains only the necessary abstraction to be still generic enough to support all the functionalities needed in Devices or other Drivers modules.

3.4.3 Devices

In this level of abstraction, the devices are used to create specific configurations with the Drivers Layer to build functions used in the Tasks layer. In opposition to the Drivers layer, the Devices do not communicate between themselves and are only used inside Tasks.

3.4.4 RTOS

The TTC 2.0 uses FreeRTOS kernel. Using an RTOS-based kernel enables the system to regularly maintain routine functions such as housekeeping, sensor reading, checking for receptions, and dealing with specific delays used in hardware, such as the radio. The priority level of a task dictates who has the most preference for execution. Initial delay

and period are used to determine the delay time to execute the task initialization after a boot; this regulates the time between the task executions. The stack is the amount of memory delimited to execute a task.

3.4.5 System

The System layer is used for housekeeping and management routines; it contains the system log, clock setup, and hooks. Its executions occur in almost all the TTC 2.0 abstraction layers (except the Tests layer), mostly for log purposes, as seen in Figure 3.2.

Figure 3.2: Example of log feedback received from TTC 2.0

3.4.6 Tasks

Tasks are the FreeRTOS threads equivalent and are the uppermost abstraction layer of code inside the TTC 2.0 flow of execution. Each task is designated a priority level, initial delay, period, and stack size as shown in Table 3.4.

Name	Priority[1]	Initial delay [ms]	Period [ms]	Stack [bytes]
Antenna Deployment	6	3600000	100	150
Downlink Manager	3	550	150	2000
EPS Server	3	10000	750	1000
Heartbeat	1	2000	500	160
OBDH Server	5	200	100	2000
Radio Reset	5	60000	60000	128
Read Antenna	2	2000	60000	150
Read Sensors	3	2000	60000	128
Startup	6	0	Aperiodic	500
System Reset	2	0	36000000	128
Time Control	3	1000	1000	128
Uplink Manager	3	500	300	2000
Watchdog Reset	1	0	100	150

Table 3.4: List of TTC 2.0 Tasks with configuration parameters.

Each of the tasks presented in Table 3.4 is described below:

- **Antenna Deployment**: Initialize the antenna sequence of deploy after

- **Downlink Manager**: Monitors for radio request from other tasks and manages down-link FIFO.

- **EPS Server**: Read only transmit requests from UART bus.

- **Heartbeat**: Blinks a status LED at a rate of 1 Hz. Both microcontrollers have a status LED. This LED indicates that the scheduler is up and running.

- **OBDH Server**: Read requests and send response from the SPI bus.

- **Radio Reset**: Resets the radio at 600 seconds.

- **Read Antenna**: Reads antenna current status and temperature.

- **Read Sensors**: Reads the uC and radio temperature and power consumption.

- **Startup**: Initializes all the devices and peripherals, and variables of the TTC 2.0 module (boot sequence).

- **System Reset**: Resets the microcontroller by software every 10 hours.

- **Time Control**: Manages the system time by loading the saving the time counter from/to the internal flash memory.

- **Uplink Manager**: Monitors the radio module for upcoming packages and stores it in memory.

- **Watchdog Reset**: Resets both watchdog timers (internal and external) at every 100 milliseconds.

3.4.7   Libraries

The Libraries are used for algorithm purposes and are not related to any hardware. Their function removes the redundancy of creating multiple identical structures for different driver modules.

3.4.8   Tests

Tests are used to verify and validate the Drivers and Devices' developed functionality. There are three types of tests: the static test uses the MISRA C: 2012 [5] guidelines to check for C safety standards, the unitary tests use the Cmocka[2] library with mockups of the hardware, to validate the module in an algorithmic level, and the last test verifies the integration between hardware and firmware.

---

[2]https://cmocka.org/

CHAPTER 4

Operation

This chapter describes general aspects of the TTC operation, like details of the used communication protocol, how to transmit a packet, how the telecommand reception works, and so on.

4.1   Packet transmission

To transmit a packet, the command "Transmit Packet" must be used (section 3.2). When the TTC receives this command through one of its command interfaces, a new NGHam packet is generated with the command content as the packet's payload. After the NGHam packet is generated, it is transmitted immediately through the air using the respective TTC's radio.

4.2   Telecommand reception

When a new valid telecommand is received, it is stored in an internal queue with five positions and 300 bytes available for each position. The new packet is discarded if the queue is full and there are no available positions to store the telecommands.

A telecommand is only stored if it is a valid NGHam packet, and only the payload of the NGHam packet is stored. Each telecommand can be up to 220 bytes long, according to the NGHam protocol definitions.

The received and stored telecommand can be read with the command read packet, as described in section 3.2. After a telecommand is read, it is deleted from the internal queue of the TTC module. The number of available telecommands and the length in bytes of the first queue position can be read using the command "Read Parameter", as also described in section 3.2 and 3.3.

4.3   Communication protocol

The communication protocol used with the satellite is the NGHam [6]. The NGHam protocol stands for Next Generation Ham Radio, a protocol designed for packet communication. It is similar to the classical AX.25 [7] protocol, but with the idea of using a FEC (Forward Error Correction) algorithm to increase the robustness, as defined in [8]:

> "...a link protocol partly inspired by AX.25. To improve the link reliability, it features Reed Solomon codes for Forward Error Correction (FEC). This makes

the data transmission more robust compared to, i.e., AX.25, which does not directly implement FEC on the link layer."

It was initially developed in the context of the CubeSat NUTS-1 (NTNU Test Satellite) development by the Norwegian University of Science and Technology (NTNU). This library is based on the implementation of Jon Petter Skagmo (LA3JPA), available in [9].

4.3.1 Packet fields

The Figure 4.1 presents a diagram with packet format and the data fields of the NGHam protocol.

Preamble 4 bytes	Sync Word 4 bytes	Size Tag 3 bytes	RS(255, 223) Code Block
			RS(223, 191) Code Block
			RS(191, 159) Code Block
			RS(159, 127) Code Block
			RS(111, 95) Code Block
			RS(79, 63) Code Block
			RS(47, 31) Code Block

| Header
1 byte | Payload
1-220 bytes | CRC
2 bytes | Padding
0-31 bytes | Parity Data
16 or 32 bytes |
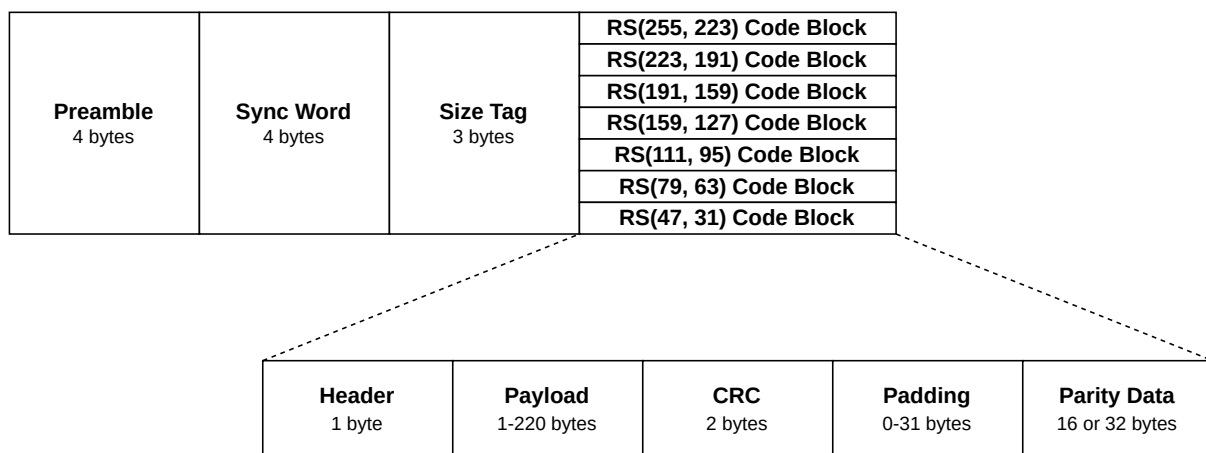
Figure 4.1: Format of an NGHam packet.

Next, there is a brief description of each one of those fields.

Preamble

Considering a data rate of 9600 bps, 4 bytes are typically used for the preamble. This sequence is an alternation of ones and zeros: 4× 0xAA (0b10101010).

Sync. word

The sync. word is a sequence of bits used for packet synchronization. With this sequence, the receiver can detect the start of a new packet. In NGHam, the sync. word is composed of 32 bits, following the pattern: 0x5D, 0xE6, 0x2A, 0x7E.

Size tag

This field indicates one of the seven possible packet sizes. It is a 24 bits tag and is made very robust by keeping a hamming distance of 13 bits between all vectors. The seven possible tags are listed in Table 4.1.

Size Num.	Tag	Reed–Solomon Config.	Max. Data Size
1	59, 73, 205	RS(47, 31)	up to 28 bytes of data
2	77, 218, 87	RS(79, 63)	up to 60 bytes of data
3	118, 147, 154	RS(111, 95)	up to 92 bytes of data
4	155, 180, 174	RS(159, 127)	up to 124 bytes of data
5	160, 253, 99	RS(191, 159)	up to 156 bytes of data
6	214, 110, 249	RS(223, 191)	up to 188 bytes of data
7	237, 39, 52	RS(255, 223)	up to 220 bytes of data

Table 4.1: NGHam packets sizes.

Reed–Solomon block

The Reed–Solomon code block (or just RS block) is the field with packet payload and parity data. It is divided into two parts: data and parity bytes. The data bytes are subdivided into four fields: header, payload, checksum, and padding. Each one of these fields is described below.

Header The header byte is the first data byte of the RS block. It is divided as presented in Table 4.2.

Bits	Purpose
7 to 6	Reserved
5	Extension on
4 to 0	Padding size (in bytes)

Table 4.2: NGHam header flags.

The extension bit indicates if the extension frame is enabled or not. The padding size bits are the number of padding bytes presented in the respective packet (0 to 31).

Payload The payload field is where the "useful" packet data is stored. As presented in the Size Tag field description above, each of the seven size groups allows a maximum number of bytes in the packet payload. The maximum possible length of the payload for an NGHam packet is 220 bytes. If more data need to be transmitted, it should be divided into chunks of 220 bytes and transmitted in separate packets.

Checksum There is a checksum field after the payload data to ensure data correctness and a first stage before running the Reed–Solomon correction algorithm. The used check-sum algorithm is the CRC16-CCITT 5, with the following configuration:

- **Polynomial**: 0x1021

- **Initial value**: 0xFFFF

- **Final XOR value**: 0xFFFF

The CRC16 value is computed from the header and the payload fields.

If the CRC16 value is correct, the Reed-Solomon chain is skipped, and the packet is directly considered valid. This way, the checksum field also allows a performance improvement.

Padding   To ensure the right packet length for the Reed-Solomon coding in use, if the payload content is less than the maximum allowed, the data field of the RS block is padded with zeros. The number of padding bytes is declared in the header byte (bits 4-0).

Parity data   This field is reserved for the computed parity bytes of the Reed-Solomon coding algorithm. The used implementation of the RS algorithm is based on the famous FEC library developed by Phil Karn (KA9Q) 6. This field can be 16 or 32 bytes long, depending on the payload length, and consequently, the adopted RS scheme described in Table 4.3.

Size Num.	Reed-Solomon Config.	Parity bytes
1	RS(47, 31)	16
2	RS(79, 63)	16
3	RS(111, 95)	16
4	RS(159, 127)	32
5	RS(191, 159)	32
6	RS(223, 191)	32
7	RS(255, 233)	32

Table 4.3: NGHam parity sizes.

For the Reed-Solomon framing, the following configuration is used:

- **Symbol size**: 8

- **GF polynomial**: 0x187 (coefficients form)

- **First root of RS code generator polynomial**: 112 (index form)

- **Primitive element**: 11

- **Number of roots**: 16 or 32 (table above)

4.3.2   Scrambling

Before transmitting a packet, the RS code block is scrambled by making a byte xor operation with a pre-generated table based on the polynomial $x^8 + x^7 + x^5 + x^3 + 1$ (defined in the CCSDS 131.0-B-3 standard [10]).

When the receiver receives a packet, it also performs the same operation to de-scramble the RS code block and gets the original content of the RS part of the packet.

By scrambling the packets, long sequences of ones or zeros are avoided by guaranteeing a good bit transition along the whole packet. More information about packet scrambling (or randomization) can be found in [10] (section 8.3).

4.4 Flow of execution

The TTC module operates accordingly to the flowchart from Figure 4.2. Its routine consists of frequent checks for uplink packages or requests from OBDH or EPS to downlink data. Also, by default TTC has protective measures to reset the radio between 600 seconds and the microcontroller after 10 hours.
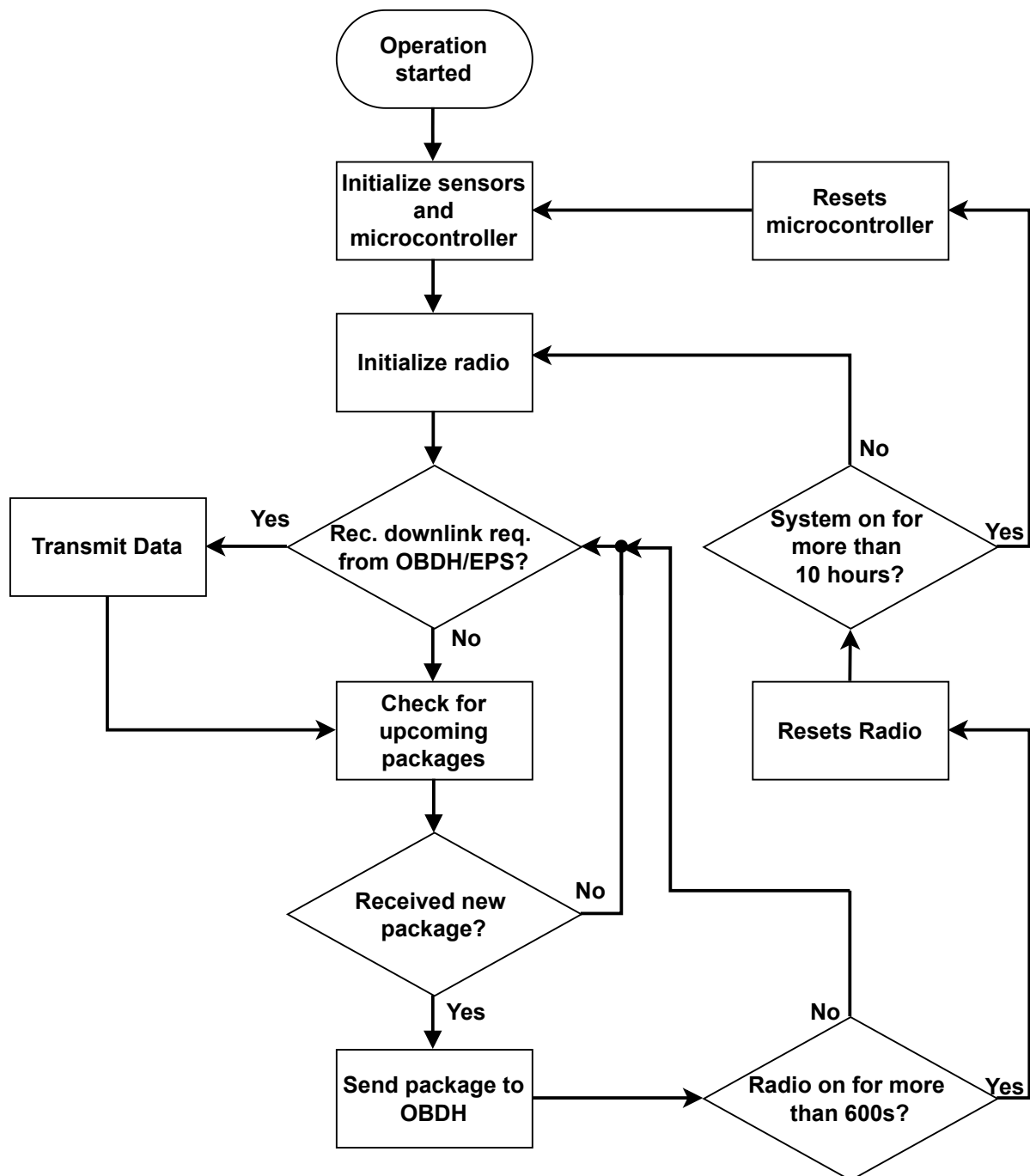
Figure 4.2: TTC 2.0 flowchart of execution.

4.5   Server Parameter

There are two main serial ports accessible to communicate with the TTC module a UART that is accessible only for the radio 1 and can only transmit packages. The other port is an SPI and is possible to communicate individually with each microcontroller and you can requisite all the commands and parameters (see Tables 3.1 and 3.3). The protocol communication is described below.

4.5.1   EPS Server

The EPS Server is in RX mode and only the Transmit packet command (ID=3) is available. To transmit a package the first byte shall be the command (0x03) followed by the packet. The UART configuration is resumed on Table 4.5. This serial port can be accessed via the pin 5 of the PC104-H2 pin 7 is connected to TX but is not utilized, see Table 2.1.

Configuration Parameter	Value
Clock	1 MHz
Baud Rate	115200 bps
Data Bits	8
Parity	No parity
Stop Bits	1

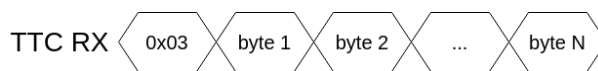Table 4.4: EPS Server configuration parameter.

Figure 4.3: EPS Server communication diagram description.

4.5.2   OBDH Server

The OBDH Server can be used for full communication and all the commands are implemented for this port. The TTC operates as half-duplex with the commands having a request and a followed by response (except for command Write parameter ID=2 that only uses a request). For commands that need a response it is required to leave a interval of at least 100ms from the request to the response. Each communication part, for both sides, in all commands starts with the preamble byte (0x7E) that can be used to detect and synchronize wrong transactions. When each side is communicating the listening side needs to send 0x00 while the other side transmits. All the request have seven bytes length including the preamble, the response for command ID=1 also has 7 bytes as fixed size, the response from commands ID=3 and ID=4 have a variable size accordingly to the request. The communication for each case is described in Figure 4.4. The TTC operates in SPI mode 0 and was tested with 1 MHz clock.
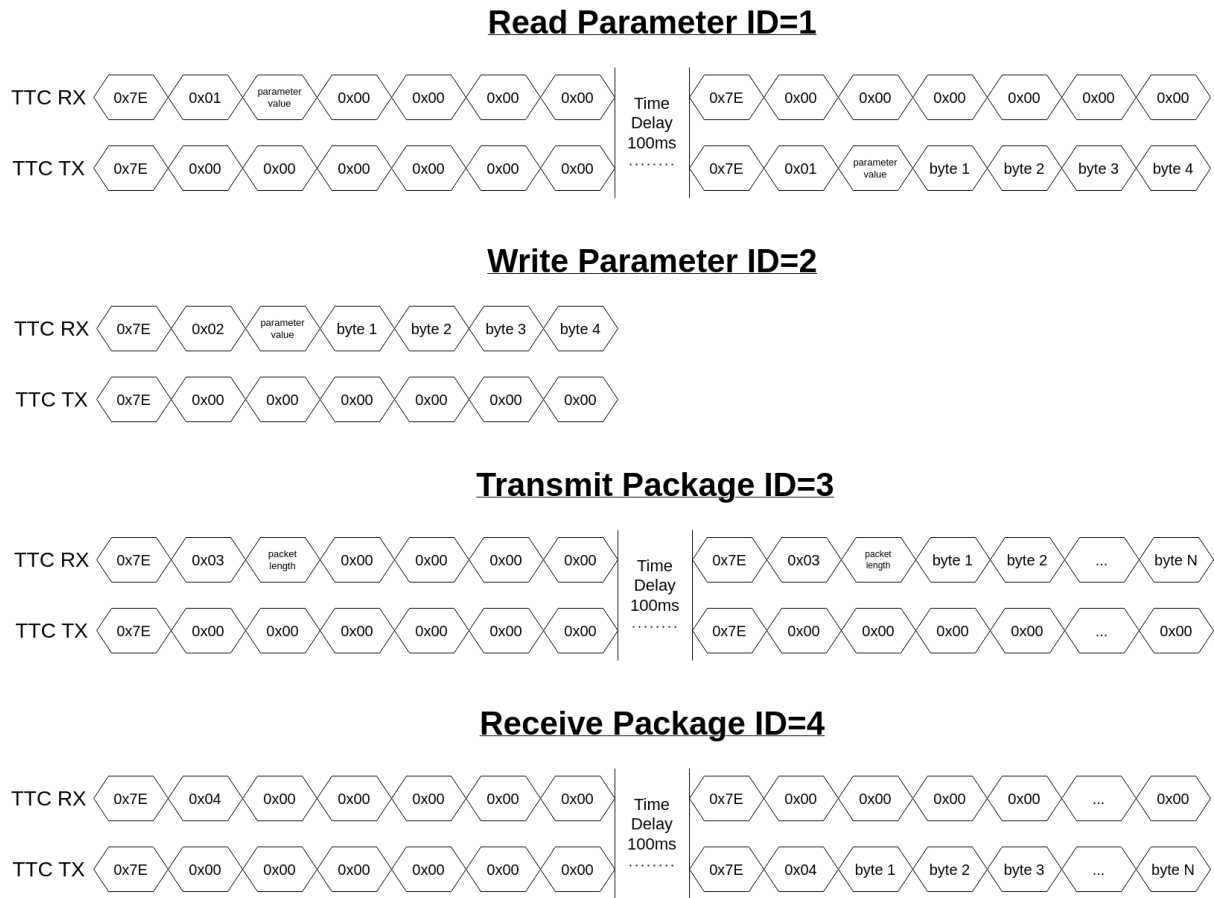
Read Parameter ID=1

| TTC RX | 0x7E | 0x01 | parameter value | 0x00 | 0x00 | 0x00 | 0x00 | Time Delay 100ms | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |
| TTC TX | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | .... | 0x7E | 0x01 | parameter value | byte 1 | byte 2 | byte 3 | byte 4 |

Write Parameter ID=2

| TTC RX | 0x7E | 0x02 | parameter value | byte 1 | byte 2 | byte 3 | byte 4 |
| TTC TX | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 |

Transmit Package ID=3

| TTC RX | 0x7E | 0x03 | packet length | 0x00 | 0x00 | 0x00 | 0x00 | Time Delay 100ms | 0x7E | 0x03 | packet length | byte 1 | byte 2 | ... | byte N |
| TTC TX | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | .... | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | ... | 0x00 |

Receive Package ID=4

| TTC RX | 0x7E | 0x04 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | Time Delay 100ms | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | ... | 0x00 |
| TTC TX | 0x7E | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | .... | 0x7E | 0x04 | byte 1 | byte 2 | byte 3 | ... | byte N |

Figure 4.4: OBDH Server communication diagram description.

Configuration Parameter	Value
Mode	0
Clock	200 kHz to 1 MHz

Table 4.5: OBDH Server configuration parameter.

CHAPTER 5

Assembly Instructions

This chapter will describe how to customize the radio configuration, power-on the board, import and flash the source code and finally alternatives to encode/decode to be transmitted/received packages from the TTC 2.0 module.

5.1 Radio Configuration

This section is a tutorial designed to generate a source code file with basic configuration parameters for the radio module. To achieve this, the WDS software from Silicon Labs (version 3.2.11.0), which is only available on Windows, will be used.

5.1.1 Steps

After the installation of the software, the procedures to configure the beacon radio are described bellow.

Step 1

1. Open the WDS software.

2. The following box will appear in the center of the window.

3. Click in "Simulate radio" and go to the next step.

Step 2

1. In the list of radios that appeared on the new window, select the chip type "Si4463".

2. In the revision column, select "B1".

3. Click on "Select Radio" to go to the next step.

Step 3

1. Select "Radio Configuration Application".

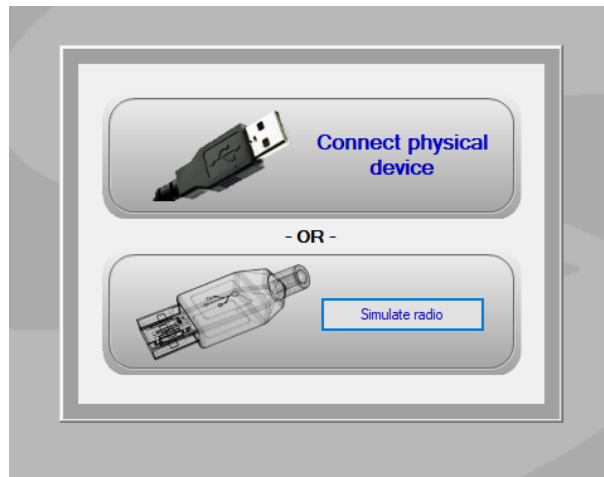2. Click on "Select Application" to go to the next step.
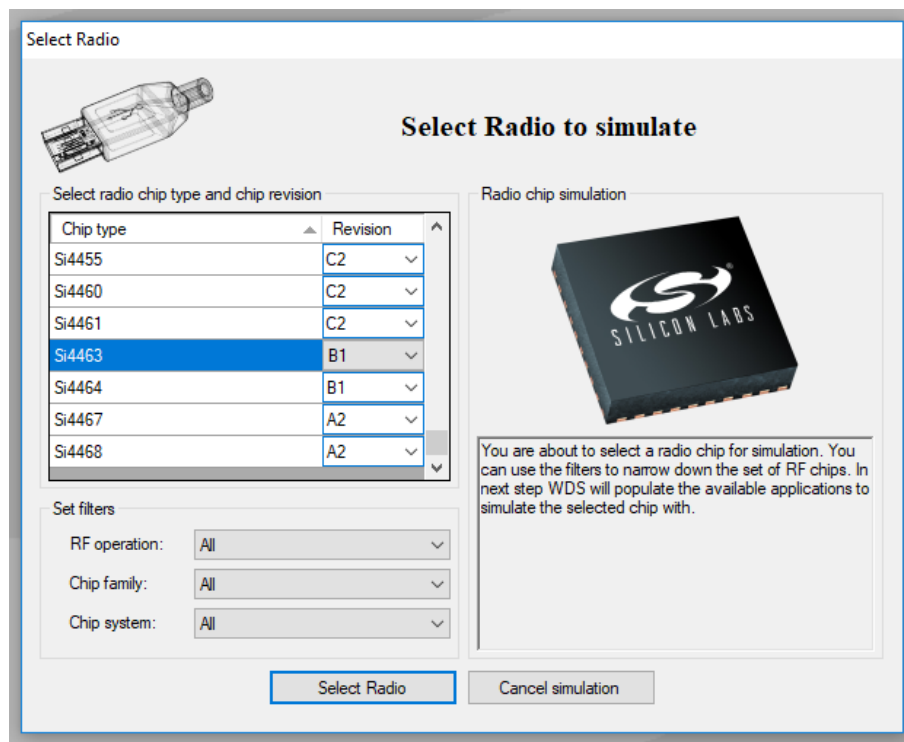
Figure 5.1: Step 1 of the radio configuration.

Figure 5.2: Step 2 of the radio configuration.

Step 4

1. In the "Frequency and power" tab, change the base frequency to 145,9 MHz.

2. Change the channel spacing to 0 kHz.

3. Change the crystal tolerance to 10,0 ppm (Both RX and TX).
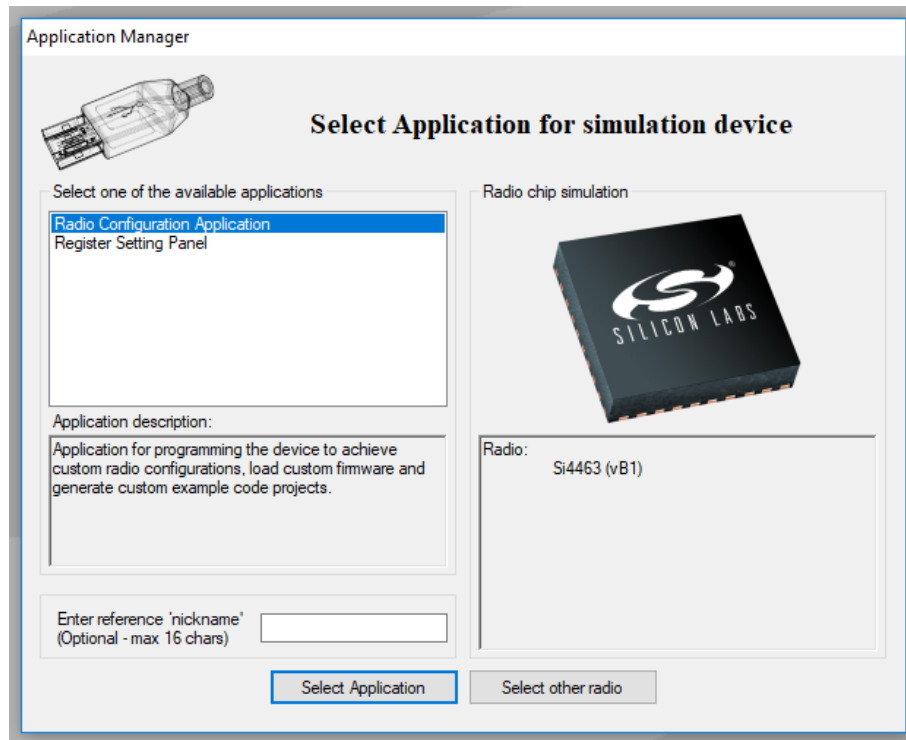
4. Go to the next step.

Figure 5.3: Step 3 of the radio configuration.

Step 5

1. In the "RF parameters" tab, change the modulation type to "2GFSK".

2. Change the the data rate to 1,2 kbps.

3. Change the deviation to $2, 5 \; kHz$.

4. Go to the next step.

Step 6

1. In the "Packet" tab, many subtabs will appear. In "Preamble" change the "Preamble TX length" to 4 bytes.

2. Again, in "Preamble", change the "Preamble pattern" to "Std. 1010 pattern ($>= 32$ and < 40 bits)".

3. Go to the next step.

Step 7

1. In the "Sync Word" tab, change the sync word length field to "4 bytes".

2. In the "Sync Word (on air int.)", enter the following sequence: 5D E6 2A 7E. This sequence is the sync word used by the NGHam protocol.
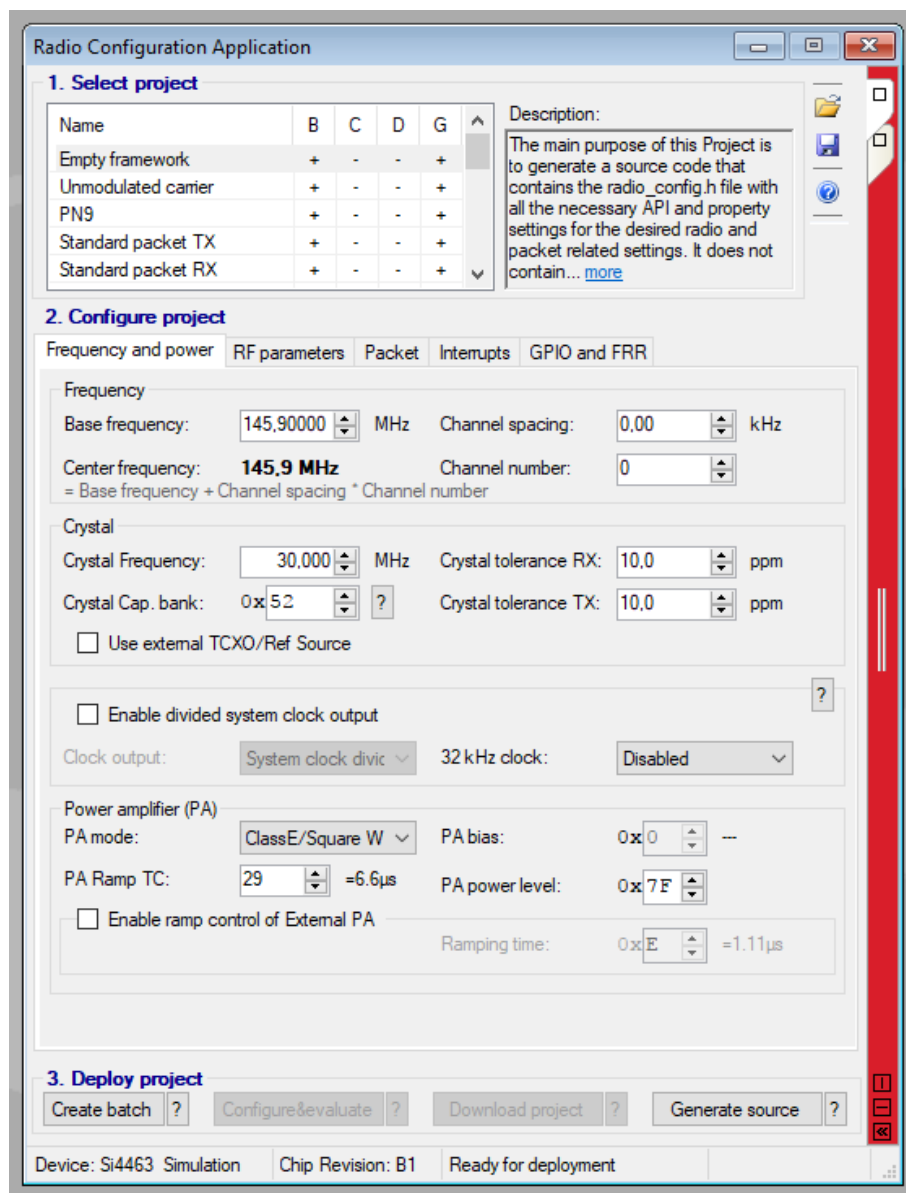
3. Go to the next step.

Figure 5.4: Step 4 of the radio configuration.

Step 8

1. In the "Field 1" tab, change the "Field length" to 50 bytes.

2. Go to the next step.

Step 9

1. In the "Variable length config" tab, there is no values to change.
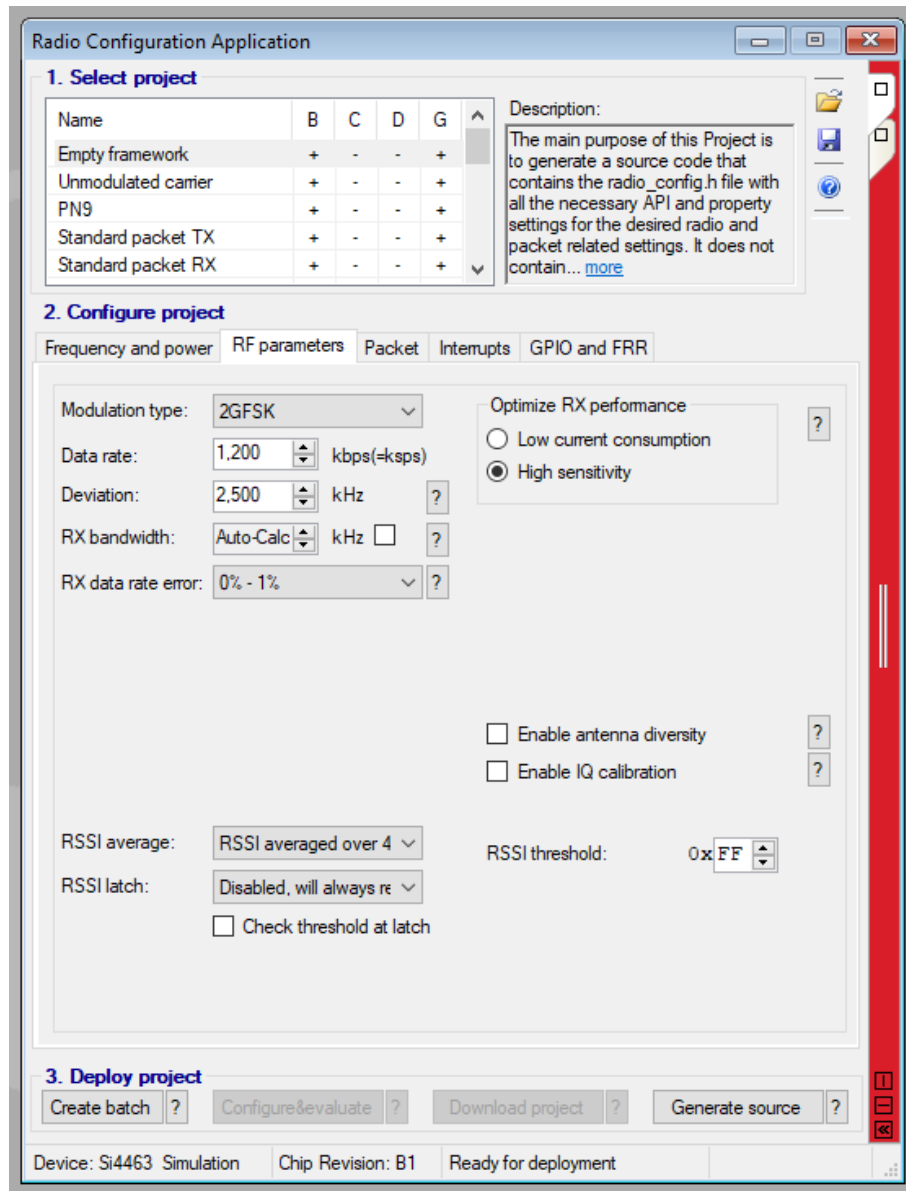
2. Go to the next step.

Figure 5.5: Step 5 of the radio configuration.

Step 10

1. In the "CRC config" tab, choose "No CRC." in "CRC polynomial".

2. Go to the next step.

Step 11

1. In the "Whitening config" tab, there is no values to change.
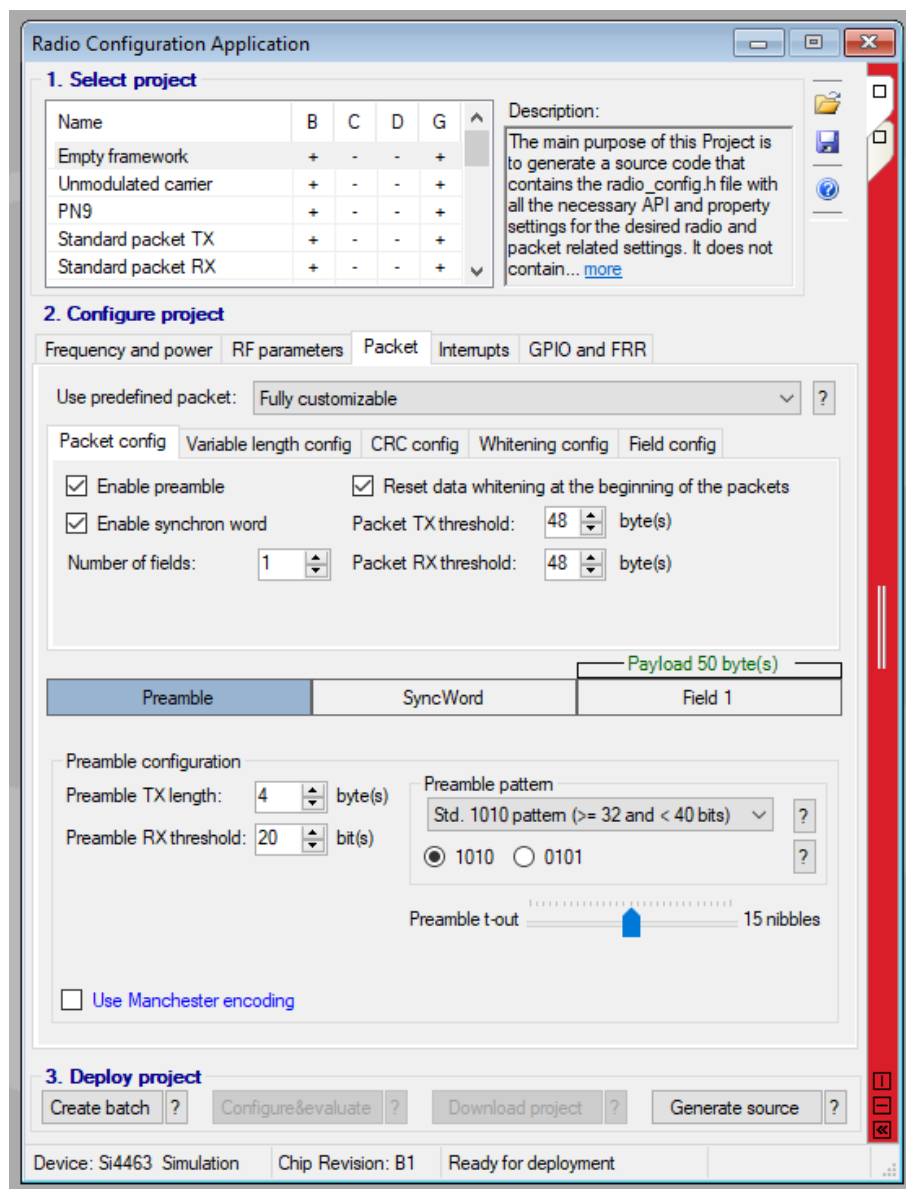
2. Go to the next step.

Figure 5.6: Step 6 of the radio configuration.

Step 12

1. In the "Field config" tab, there is no values to change.

2. Go to the next step.

Step 13

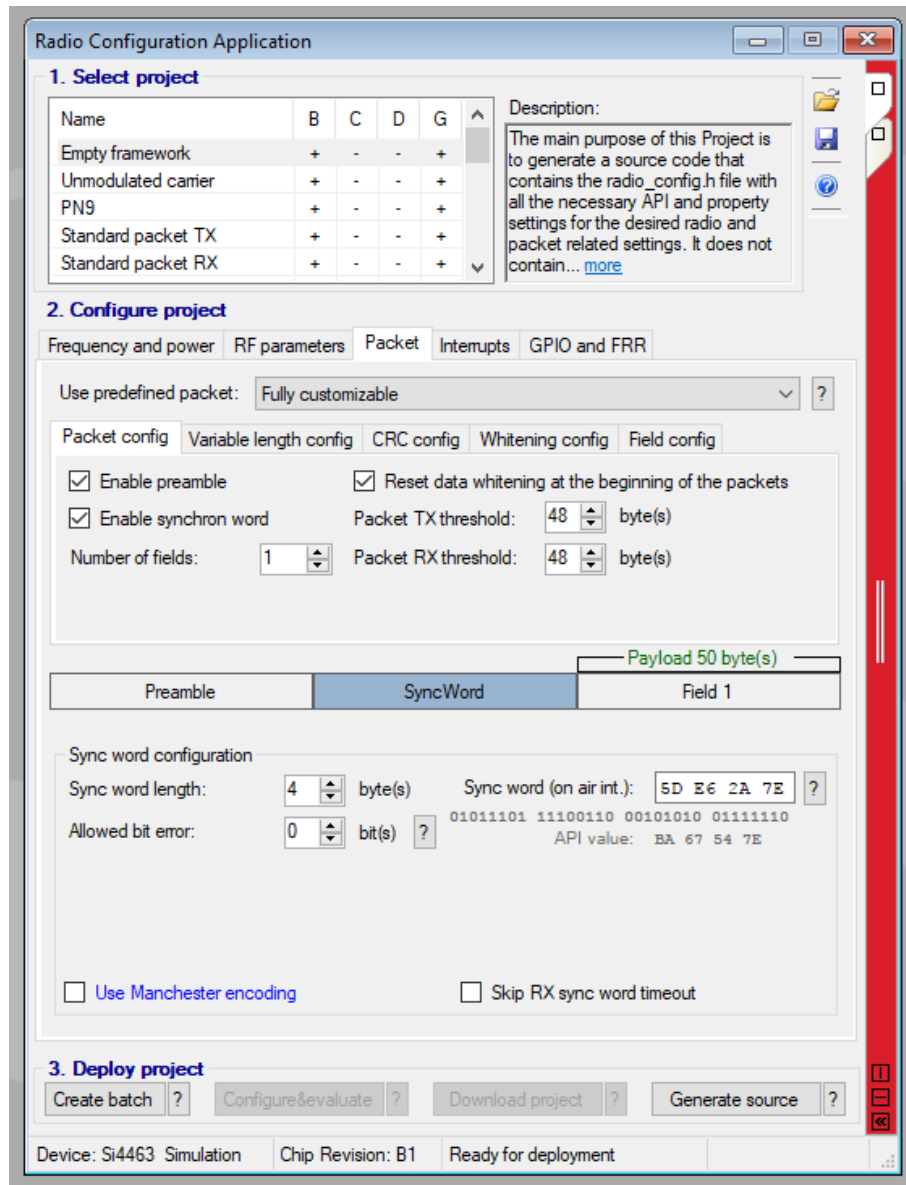1. In the "Interrupts" tab, there is no values to change.

2. Go to the next step.

Figure 5.7: Step 7 of the radio configuration.

Step 14

1. In the "GPIO and FRR" tab, enable pullup in GPIO1 and choose "TX_FIFO_EMPTY – This output is..." as functionality.

2. Enable pullup in GPIO2 and choose "RX_STATE – This output is..." as functionality.

3. Enable pullup in GPIO3 and choose "TX_STATE – This output is..." as functionality.

4. Enable pullup in NIRQ and choose "Active low interrupt signal" as functionality.

5. Enable pullup in SDO and choose "SDO – Output SPI Serial data out." as functionality.

6. Select "Global status" for the "Fast Response Register A".
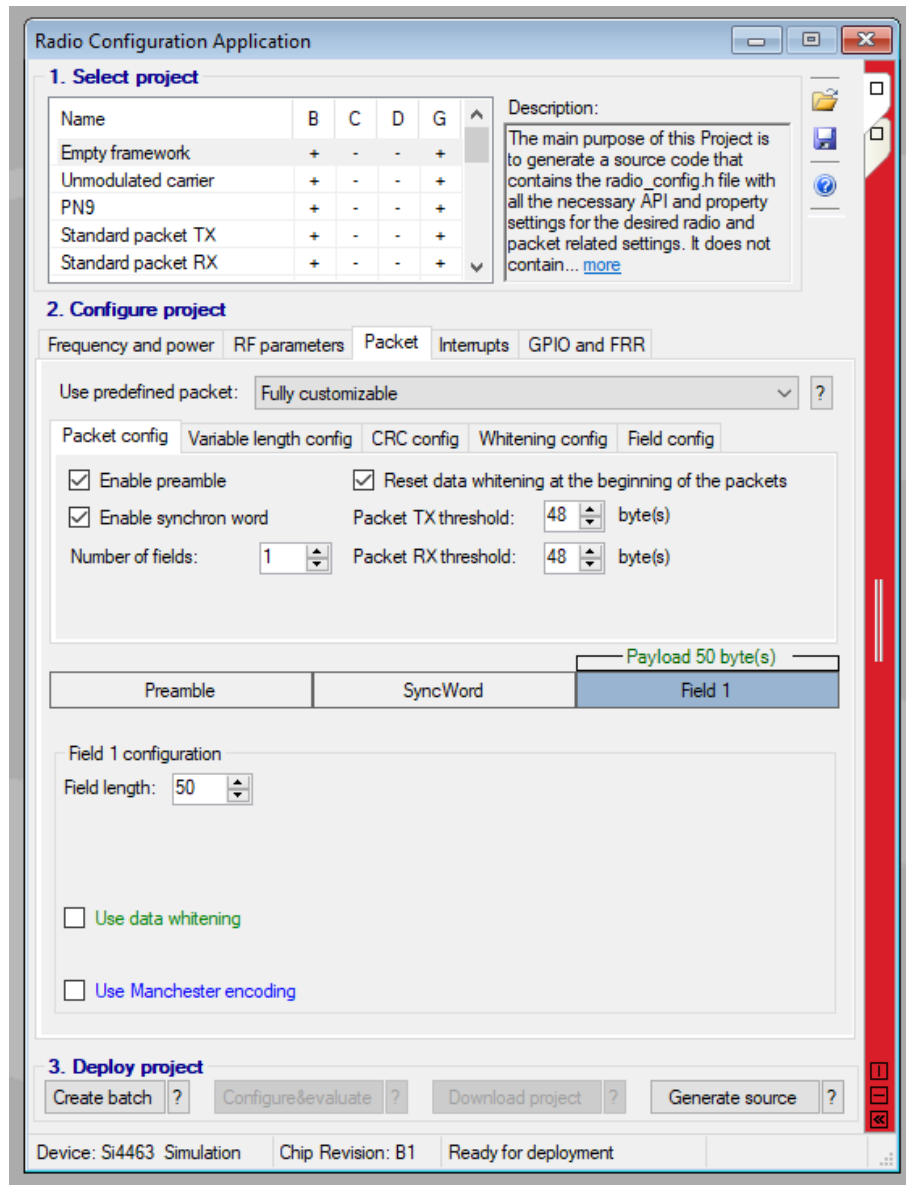
Figure 5.8: Step 8 of the radio configuration.

7. Select "Global interrupt status" for the "Fast Response Register B".

8. Select "Packet Handler status" for the "Fast Response Register C".

9. Select "Chip status status" for the "Fast Response Register D".

10. Go to the next step.

Step 15

1. Click in "Generate Source" and select the ".h" type of source.

2. The software will ask where to save the generated file.

3. The generated *.h file must be copied to the directory of the rf4463 driver.
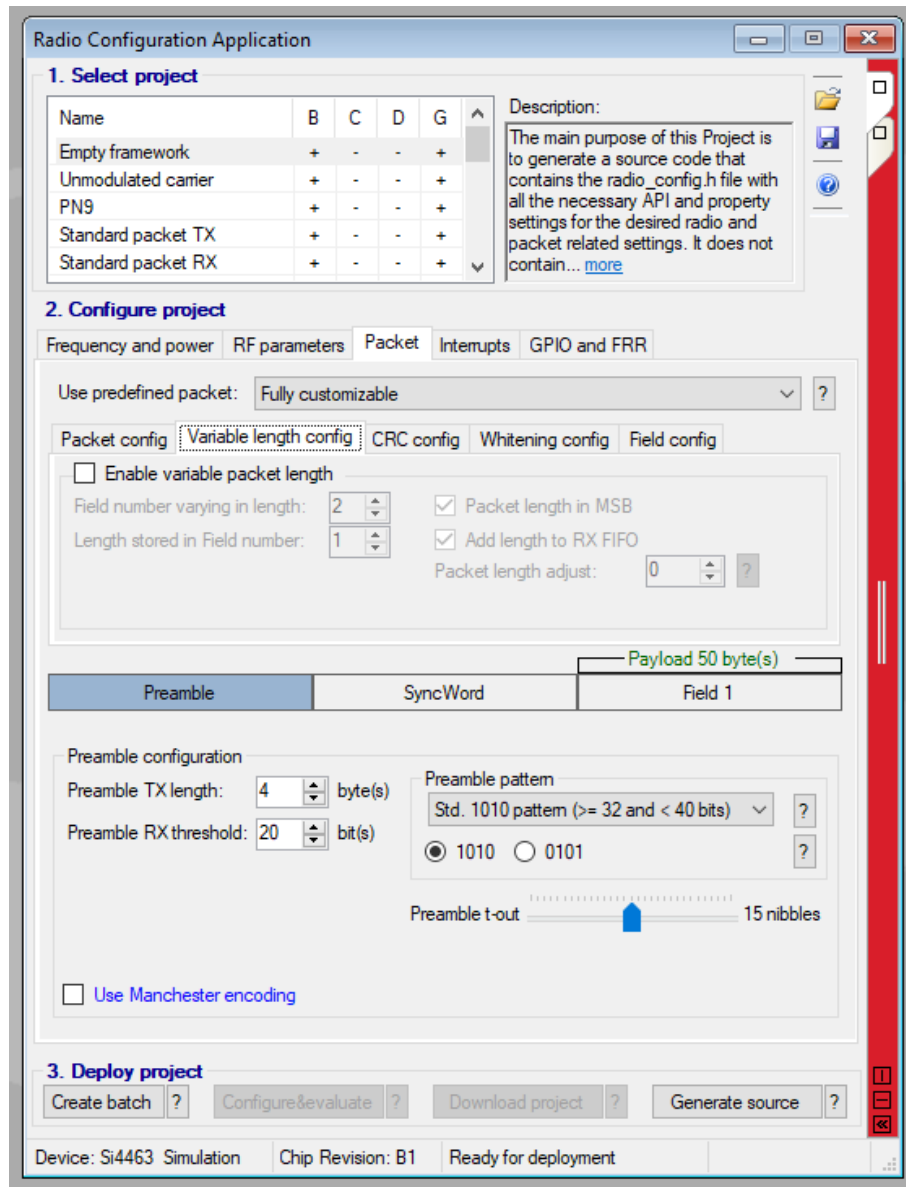
Figure 5.9: Step 9 of the radio configuration.

5.1.2 Final Remarks

This tutorial has the objective of generate a basic configuration parameters of the radio, some functionalities of the radio are not covered by the WDS software, and so, must be configured/controlled in the device driver.

5.2 Powering the module

The TTC 2.0 has three power supply inputs: one 3V3 supply to power the MCU and all the external components, two 5V inputs to power the transceivers.

The 3V3 input voltage rail is accessible at the PC-104 H2-27 and H2-28 pins. You can also supply the 3V3 rail with MSP-FET JTAG connector by connecting the jumper CN9. This last option should only be used for debug as it requires the MSP-430 FET to
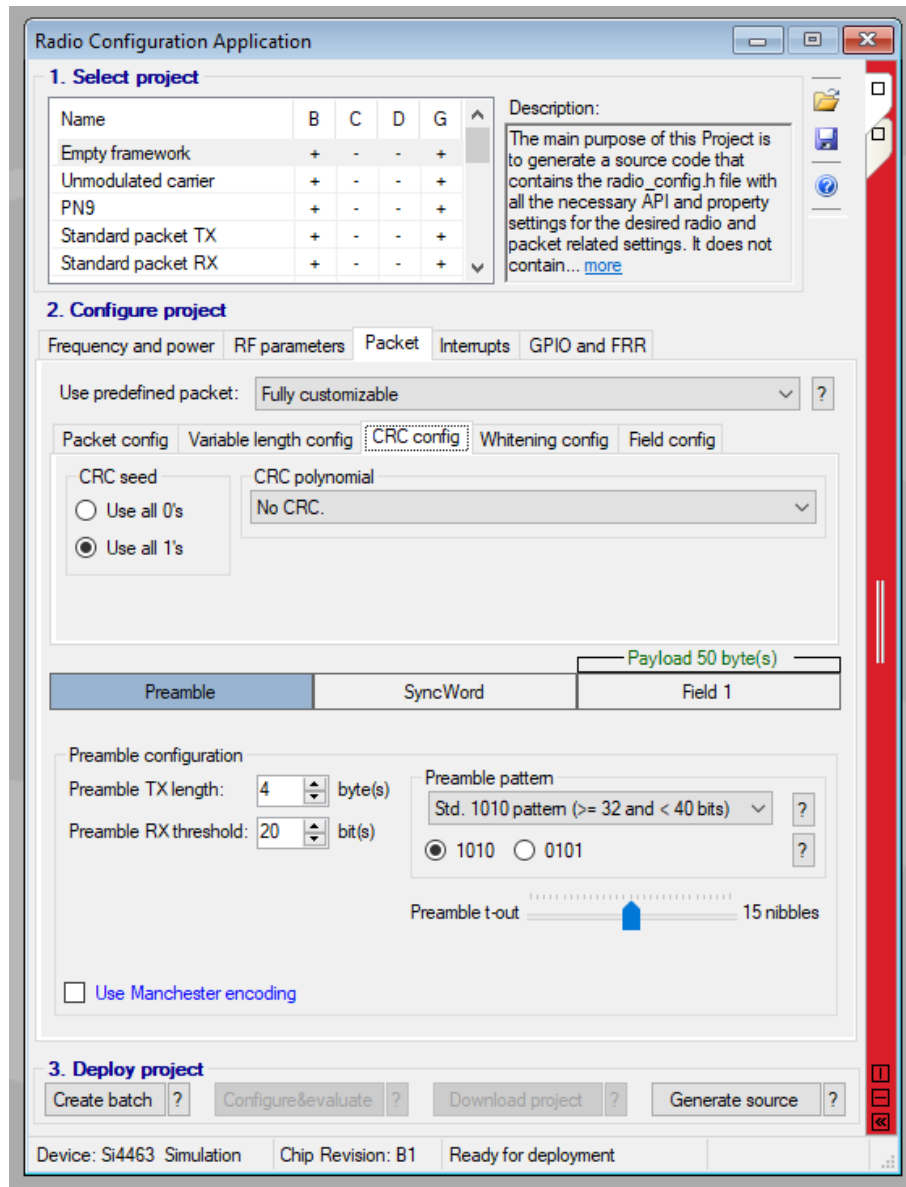
Figure 5.10: Step 10 of the radio configuration.

be connected.

The transceiver supply rails are only accessible on the PC-104: H1-49 and H1-50 for radio 0 and H1-51 and H1-52 for radio 1.

The TTC 2.0 module has only one common ground and it can be accessed by multiple pins on the PC-104 (see Table 2.2)

It is recommended, for safety reasons, to use different supply channels for each supply rail. Also, for extra safety it is recommended to add current cut-off limits: use $30mA$ for the 3V3 and $500mA$ for the radio 5V power rails.

5.3 Compiling, Building and Flashing the source code

This tutorial is a reference to compile, build and flash the firmware of the TTC 2.0 module using the Texas Instruments IDE: Code Composer Studio (CCS) version 12.
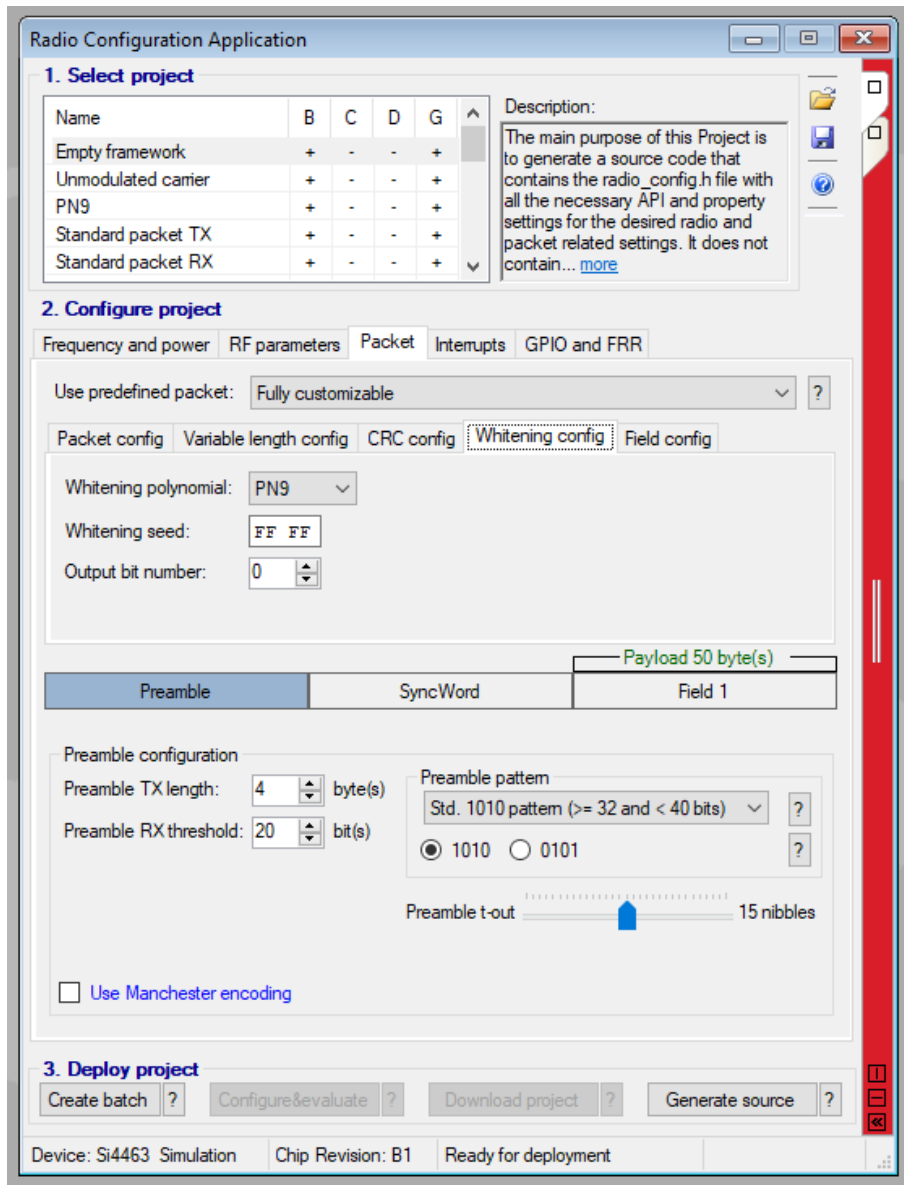
Figure 5.11: Step 11 of the radio configuration.

5.3.1 Importing the Source Code in the CCS IDE

The TTC 2.0 releases are available to download at the TTC 2.0 Github releases page [11]. The CCS IDE is available at the Texas Instruments website [12].

The steps bellow describe how to import the source code in the CCS IDE:

1. Extract the ttc2-x.x.zip file.

2. Open the CCS IDE.

3. Go to "Project" -> "Import CCS Projects...".

4. The window from the figure 5.15 will appear on the screen.

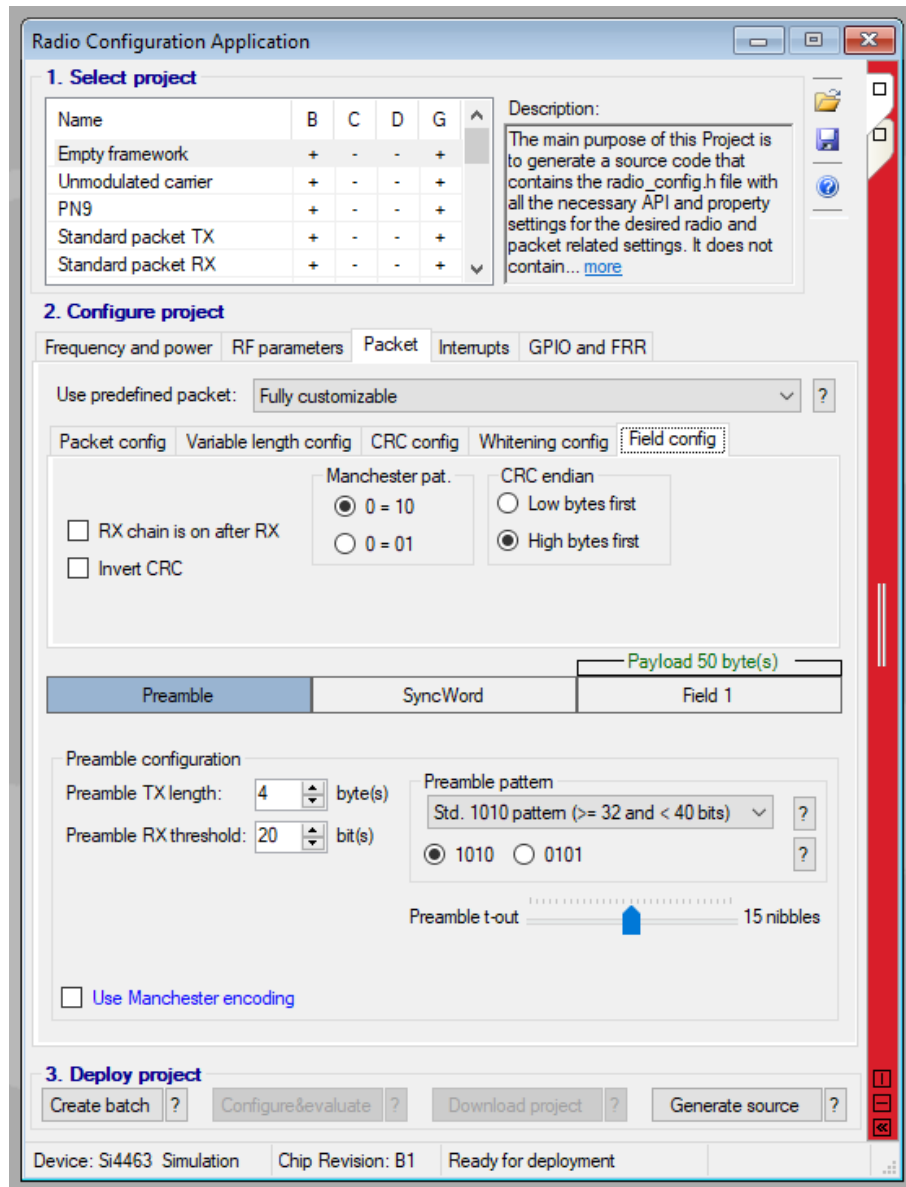5. On "Select search-directory:" select "*project_path/ttc2-x.x/firmware*".

Figure 5.12: Step 12 of the radio configuration.

6. Click on "Finish".

7. After importing the project, select the project on the "Project Explorer" and open the "Properties" tab (you can use the shortcut *Alt+Enter* or right-click on the project's name).

8. The window from the figure 5.16 will appear on the screen.

9. On "Variant", select the MCU partnumber (MSP430F6659 or MSP430F5659).

10. On "Compiler Version", select "TI v21.6.1.LTS" or higher.
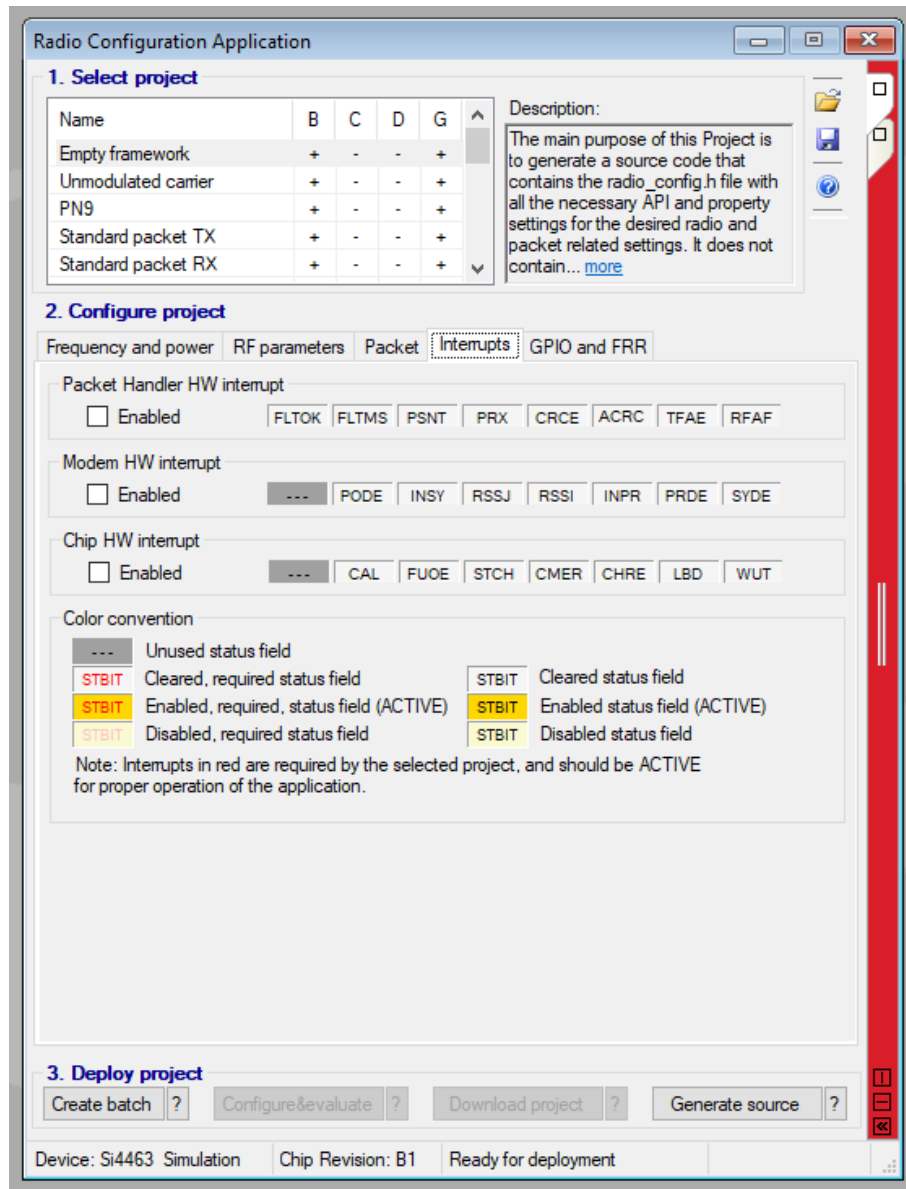
11. Click on "Apply and Close".

Figure 5.13: Step 13 of the radio configuration.

5.3.2 Customize the project

Since the TTC 2.0 has two MCU modules it is needed to specific which one you want to target. Access the configuration header file on the "Project Explorer" at "*config/config.h*", change the "RADIO_MODULE" macro between '0' or '1' (The module '1' is the one closest to the PC-104).

Each MCU module controls a corresponding transceiver, to customize the transceiver configuration (as shown on section 5.1) copy all the code on the header file generated by the WDS software and paste it at the "config/radio_x_config.h".

5.3.3 Compiling and Building

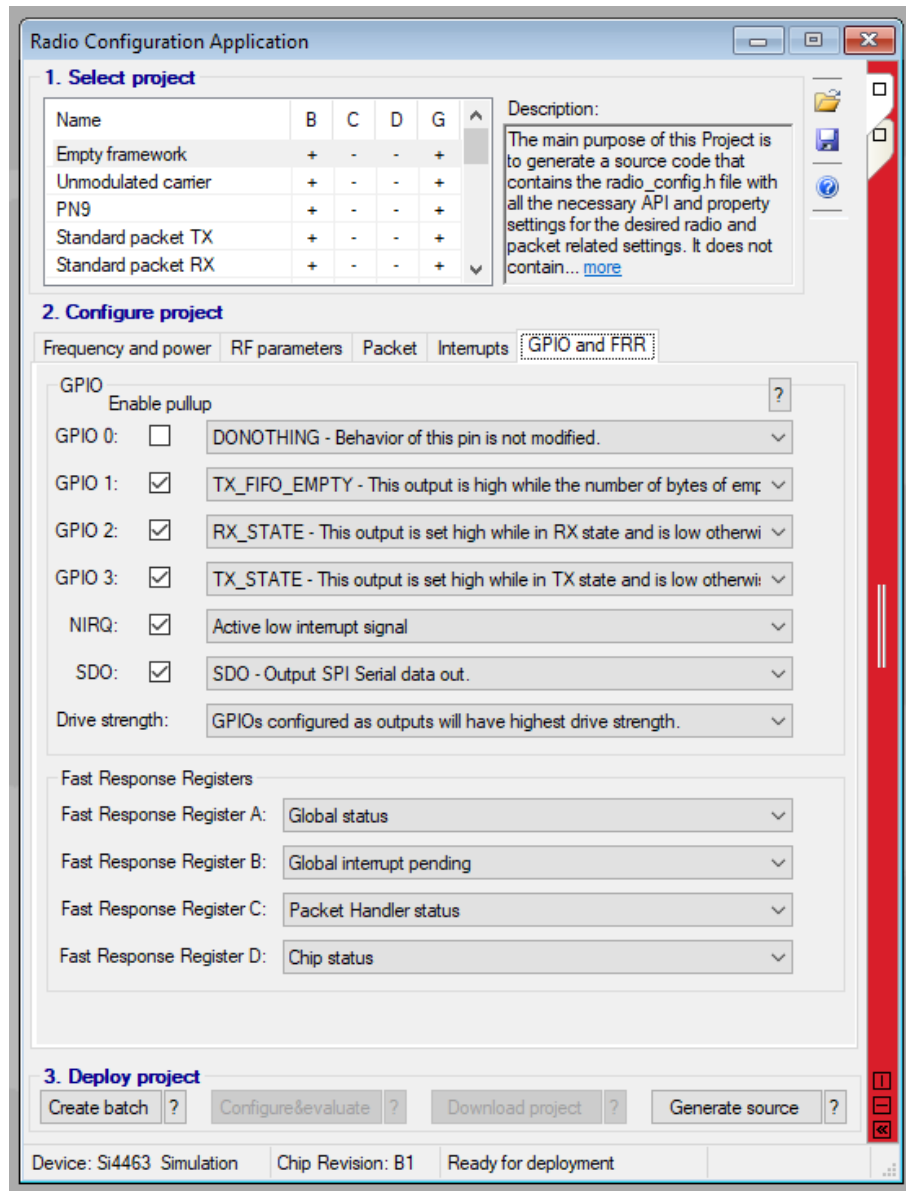To compile and build the firmware, click on "Project" -> "Build Project".

Figure 5.14: Step 14 of the radio configuration.

5.3.4   Flashing

With the board turned on and the MSP-FET connected, click on "Run" -> "Load". If no errors occur, the firmware was loaded successfully to the board.

5.4   Receiving and sending Telecommands

The structure of the telecommands are described at chapter 4. The standard solution to implement a ground-station is using the SpaceLab transmitter and decoder solutions [13] [14] combined with and RTL-SDR.
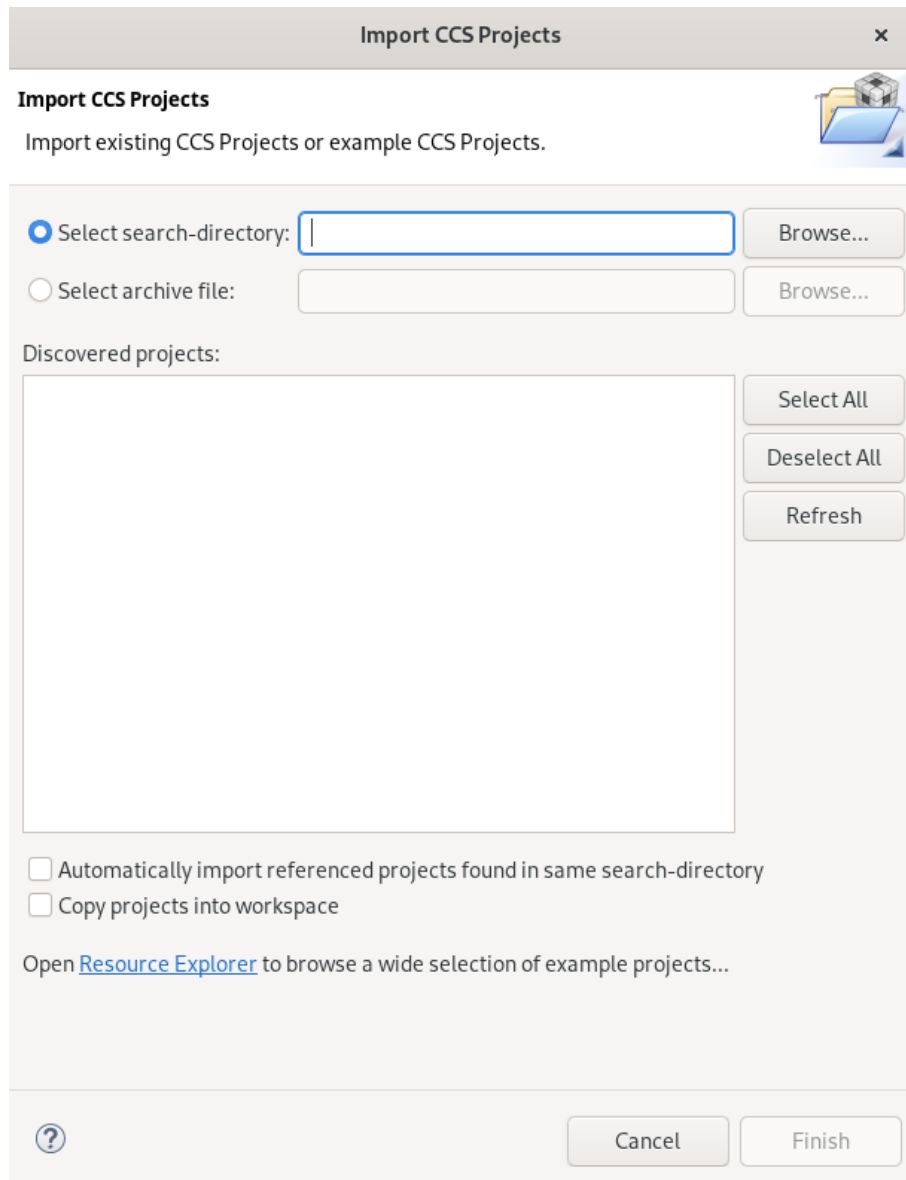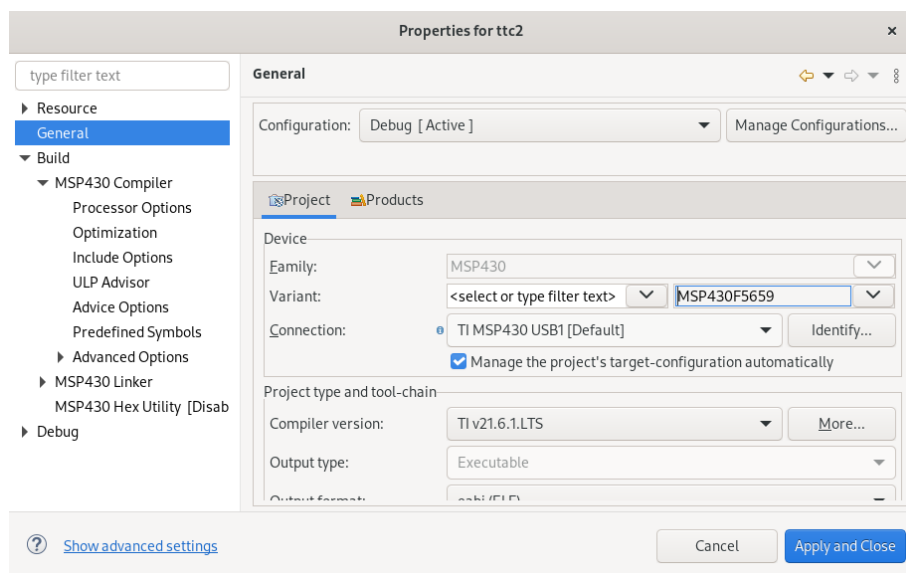
Figure 5.15: New CCS project window.

Figure 5.16: CCS project's properties window.

Bibliography

[1] SpaceLab. FloripaSat-2 Documentation, 2021. Available at <https://github.com/spacelab-ufsc/floripasat2-doc>.

[2] SpaceLab. Telemetry, Tracking and Command, 2019. Available at <https://github.com/floripasat/ttc>.

[3] SpaceLab. Telemetry, Tracking and Command 2.0, 2021. Available at <https://github.com/spacelab-ufsc/ttc2>.

[4] NASA. CubeSat101: Basic Concepts and Processes for First-Time CubeSat Developers, 2017. Available at <https://www.nasa.gov/sites/default/files/atoms/files/nasa_csli_cubesat_101_508.pdf>.

[5] MIRA Limited, Watling Street, Nuneaton, Warwickshire – United Kingdom. *MISRA C:2012 – Guidelines for the use of the C language in critical systems*, 2013.

[6] Gabriel Mariano Marcelino. PyNGHam library documentation, 2022.

[7] Peter Loveall. AX.25 Layer 2, 1984.

[8] André Løfaldli and Roger Birkeland. Implementation of a software defined radio prototype ground station for cubesats. 06 2016.

[9] Jon Petter Skagmo. NGHam Protocol, 2014.

[10] CCSDS. *TM Space Data Link Protocol*. The Consultative Committee for Space Data Systems, CCSDS Secretariat, National Aeronautics and Space Administration, Washington, DC, USA, 2021.

[11] SpaceLab. TTC 2.0 – Releases GitHub page, 2024. Available at <https://github.com/spacelab-ufsc/ttc2/releases>.

[12] Texas Instruments. Code Composer Studio™ integrated development environment (IDE), 2024. Available at <https://www.ti.com/tool/CCSTUDIO>.

[13] SpaceLab. SpaceLab Transmitter, 2024. Available at <https://github.com/spacelab-ufsc/spacelab-transmitter>.

[14] SpaceLab. SpaceLab Decoder, 2024. Available at <https://github.com/spacelab-ufsc/spacelab-decoder>.

APPENDIX A

Test Report of v0.1.1 Version

This appendix is a test report of the first manufactured and assembled PCB (version v0.1.1).

- **PCB manufacturer**: PCBWay (China)

- **PCB assembly**: PCBWay (China)

- **PCB arrival date**: 2022/04/18

- **Execution date**: 2022/04/22 to 2022/04/29

- **Tester**: Gabriel M. Marcelino, Vitória B. Bianchin and Miguel Boing

- **DNP components**: J_P10, J_P4, J_P11, J_P5, J_P9, J_P12, J_P13, J_P14, R7, R24, V2, V4, ESD, U12, U13

A.1 Visual Inspection

- **Test description/Objective**: Inspection of the board, visually and with a multimeter, searching for fabrication and assembly failures.

- **Material**:

  - Multimeter Fluke 17B+
  - Digital microscope (1000x)

- **Results**: The results of this test can be seen in Figures A.1 (top view of the board) and A.2 (bottom view of the board).

- **Conclusion**: No problems were identified on this test.

A.2 Firmware Programming

- **Test description/Objective**: Inspection of the board, visually and with a multimeter, searching for fabrication and assembly mistakes.

- **Material**:
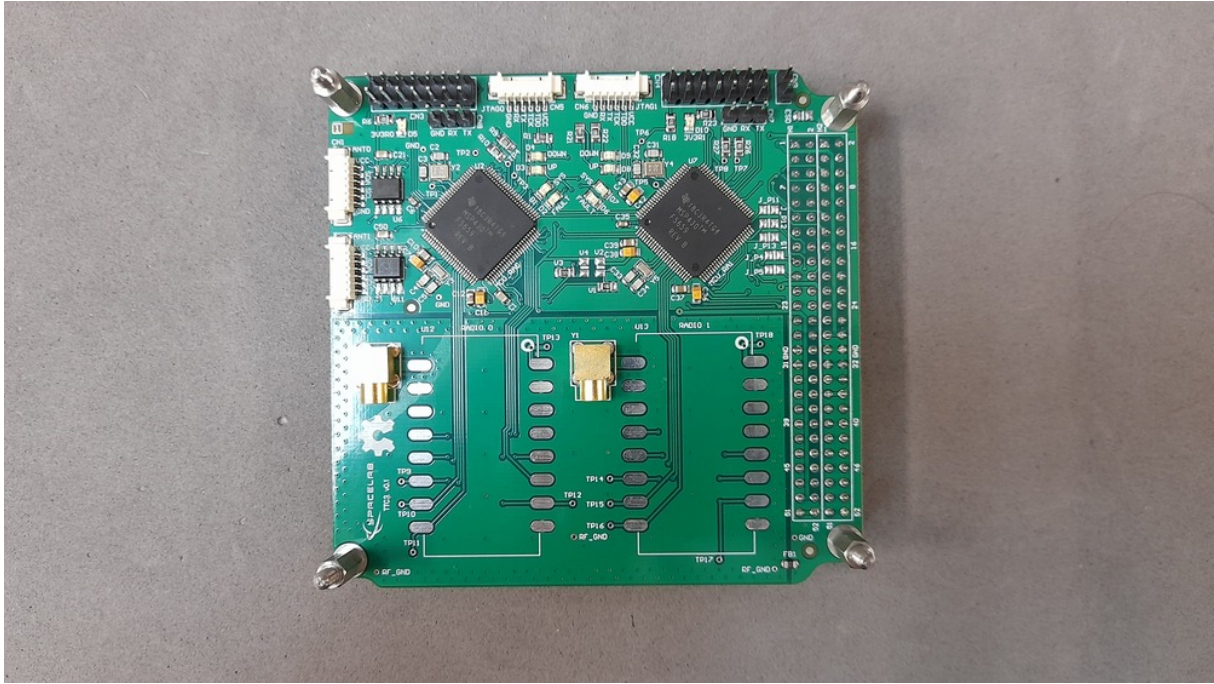
  - Code Composer Studio v11.2.0
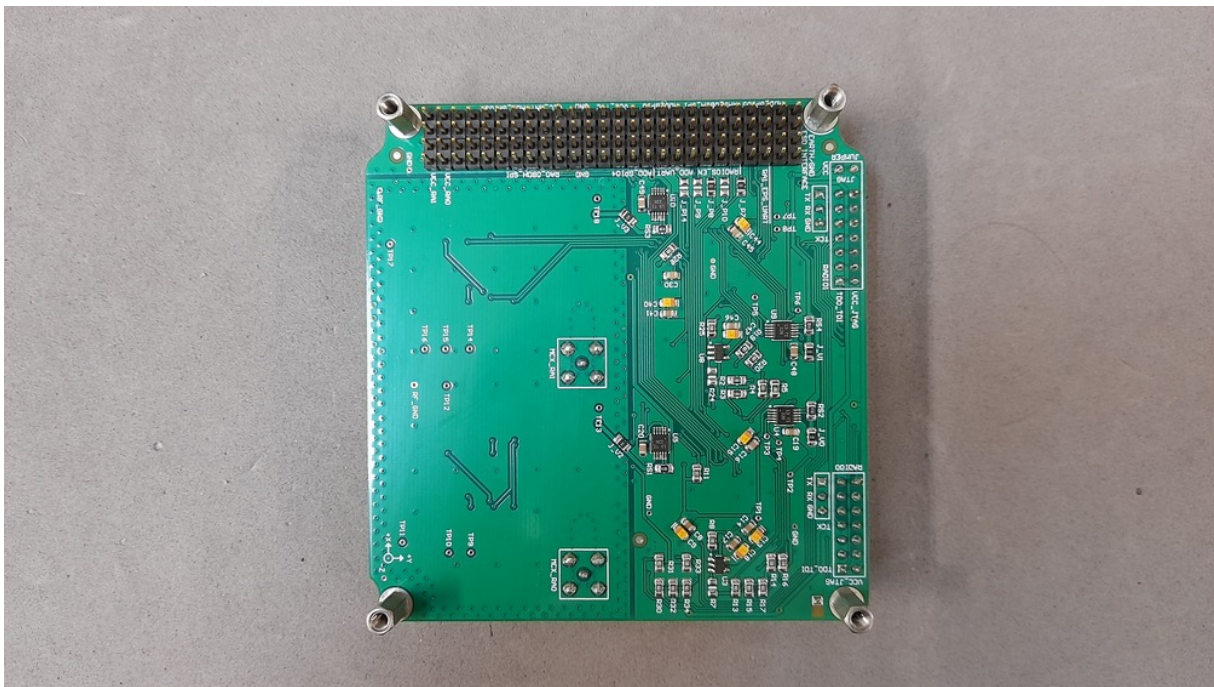
Figure A.1: Top view of the TTC 2.0 v0.1.1 board.

Figure A.2: Bottom view of the TTC 2.0 v0.1.1 board.

- MSP–FET Flash Emulation Tool
- USB–UART converter
- Screen (Linux software)

- **Results**:

- **Conclusion**:

A.3   Communication Busses

- **Test description/Objective**:

- **Material**:

  - Saleae Logic Analyzer (24 MHz, 8 channels)
  - Saleae Logic software (v1.2.18)
  - MSP-FET Flash Emulation Tool

- **Results**:

- **Conclusion**:

A.4   Sensors

A.4.1   Input Voltage

- **Test description/Objective**: Verify the input voltage measurements of the board.

- **Material**:

  - Code Composer Studio v11
  - MSP-FET Flash Emulation Tool
  - USB-UART converter
  - Screen (Linux software)

- **Results**: .

- **Conclusion:** No problems were identified on this test.

A.4.2   Input Current

- **Test description/Objective**: Verify the input current measurements of the board.

- **Material**:

  - Code Composer Studio v11
  - MSP-FET Flash Emulation Tool
  - USB-UART converter
  - Screen (Linux software)

- **Results**: .

- **Conclusion:** No problems were identified on this test.

A.5   Conclusion

No major problems were identified during the executed tests, all peripherals all working as expected.