Specifications for the Attitude Control and Determination System-XCubeSat

Dhruv Sharma, Damien Seux, Quentin Lisack, Kevin Garanger 12 April 2015

Contents

1 Introduction 4

2 Architecture of the ADCS 5

3 External interfaces 10

4 Software Architecture 11

4.1 Division of the software 11

4.2 Structure of classes 12

4.3 Definition of ports 13

4.4 API Firmware 14

4.4.1 PWM 16

4.4.2 ADC 17

4.4.3 USART 17

4.5 Driver 17

4.6 Specialized Functions 17

5 Data formats 18

6 Testing 19

7 Referentials 20

List of Figures

4.1 *Division of the software in three layers each independent of the other* 11
4.2 *Class structure for ADCS Software* 12

List of Tables

4.1 List of peripherals with alternate functions and other configurable parameters . . 14
4.2 Clocks connected to each of the peripherals 15
4.3 PWM-ON 16
4.4 Configure the ports for the direction 16
4.5 Configure the ports as Timers 16
4.6 List of PWM Ports 17

Chapter 1

Introduction

The objective of this document is to summarize in one place the specifications related to the software of the Attitude Control and Determination System, henceforth ADCS of the nano-satellite X-CubeSat developed by the students of cole Polytechnique, Paris. We have divided the document into the following sections

- Definition of the architecture of the ADCS
- External Interfaces available on the ADCS
- Sensors and peripherals on the ADCS
- Data formats used for computing
- Protocols for testing

This document shall be useful for all people currently concerned with the conception of the ADCS and eventually for students who will take over the project in the months to come. This document is however, far from being complete and additions, corrections and deletions will be made till the day of delivery.

Chapter 2

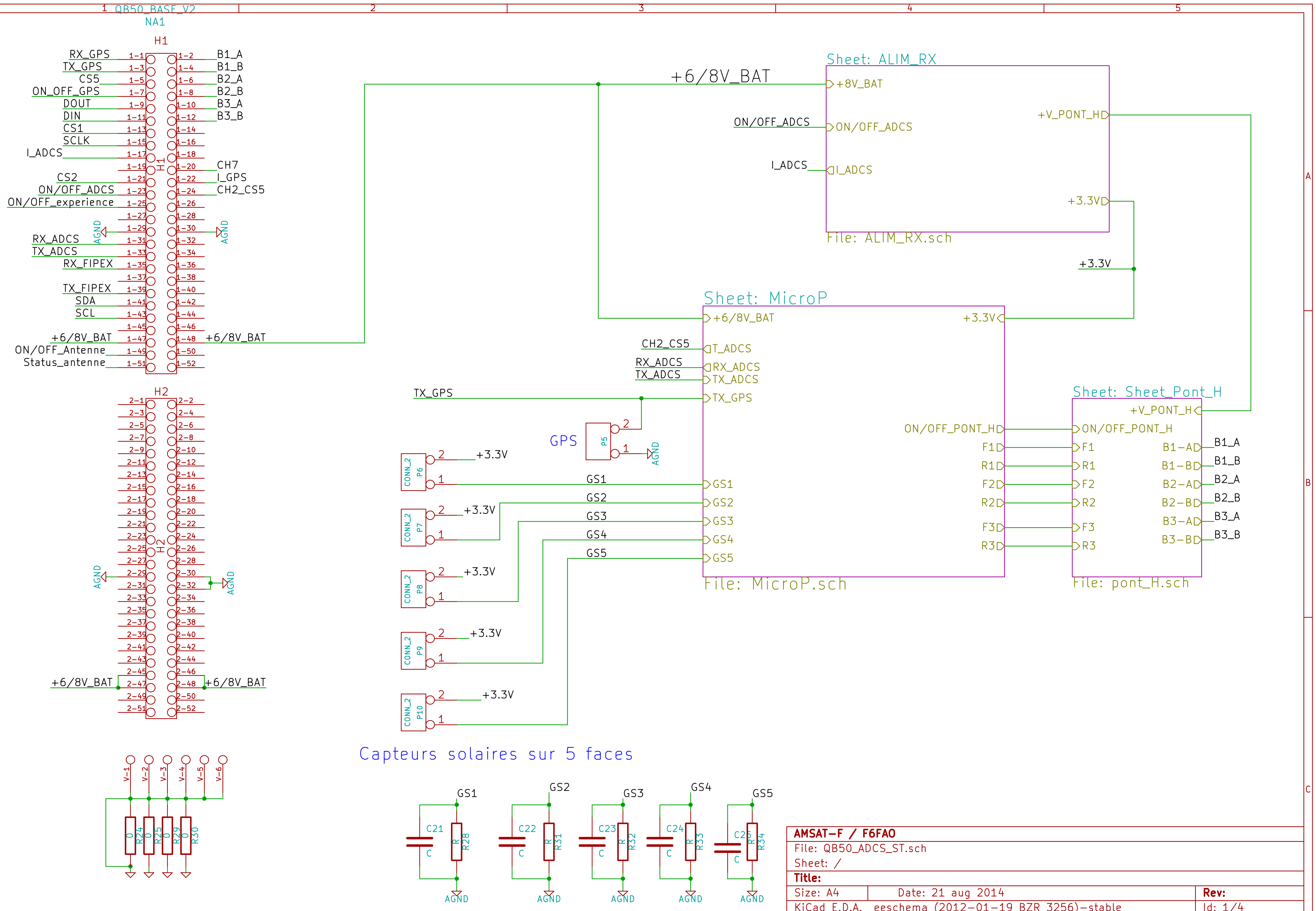
Architecture of the ADCS

The ADCS uses a microprocessor manufactured by ST Microelectronics based on the ARM Cortex-M4 Architecture. The model used is STM 32F405 RGT6

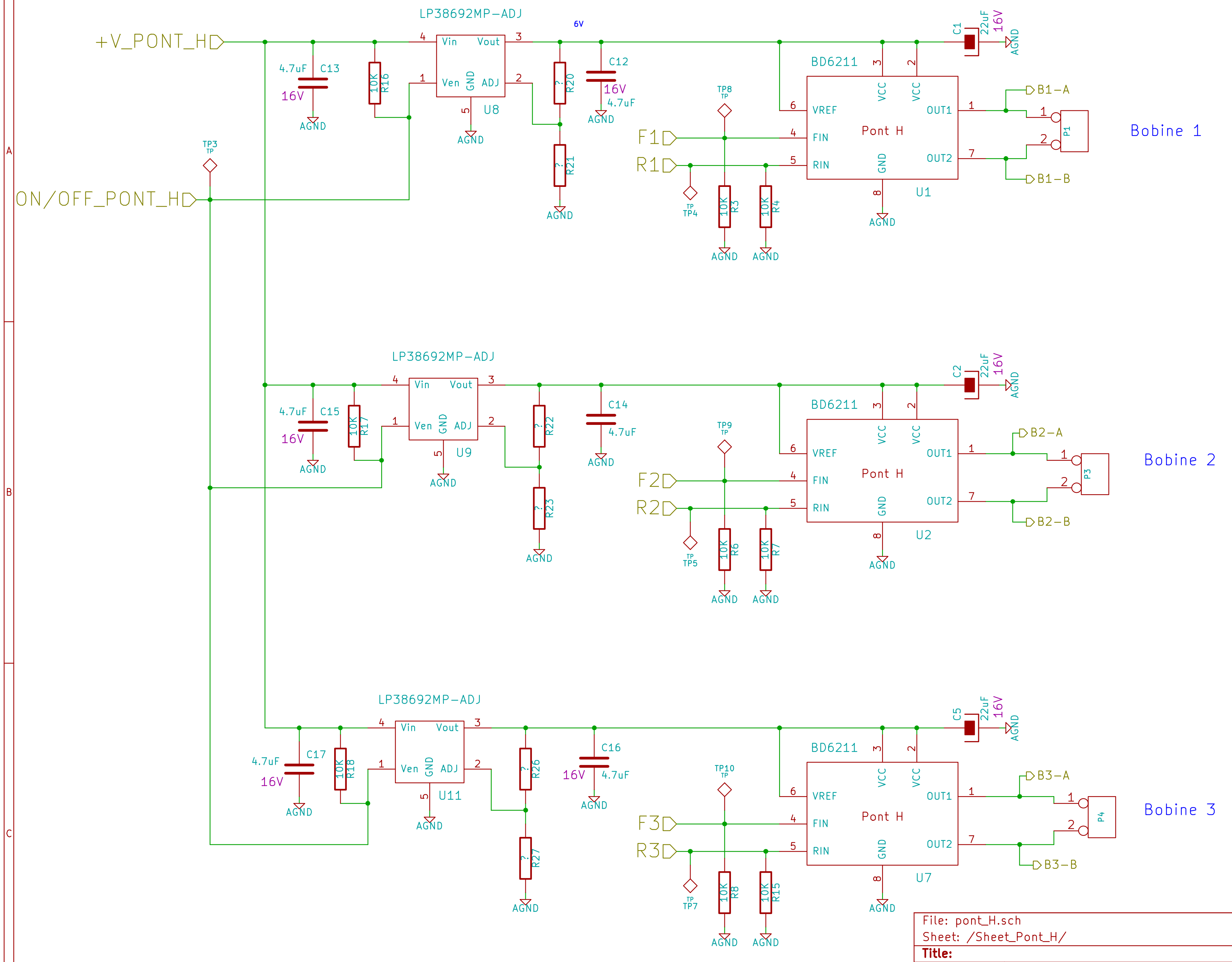
The available interfaces, sensors and other peripherals available on the ADCS are:

- 32 MB External Memory
- 7(or 9) sun sensors
- Magnetometer
- Gyrometer
- H-Bridges to modulate power to the magneto-torquers
- A Serial interface with the On-Board Computer (*Ordinateur de Bord* henceforth, ODB)

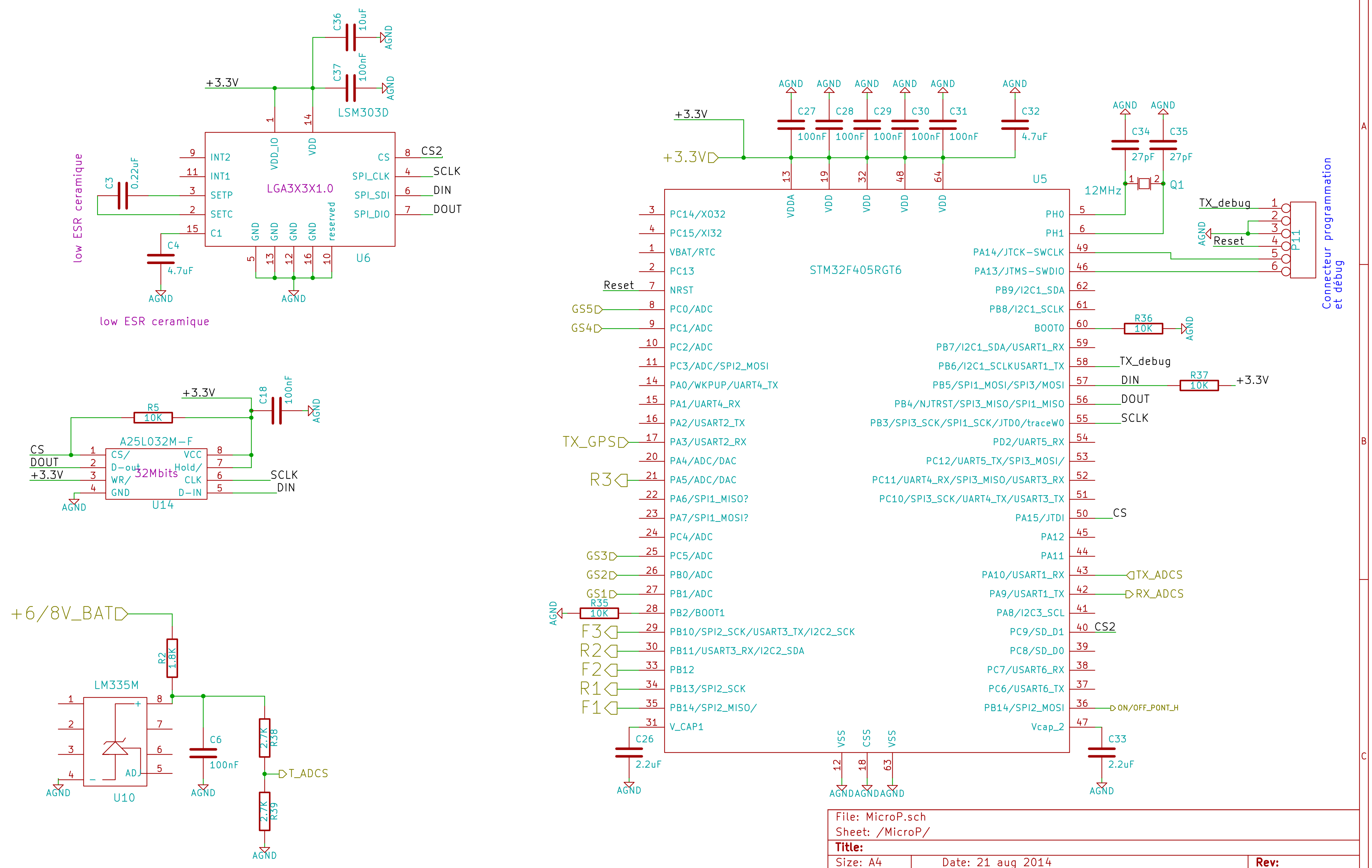
Following is the complete plan for the ADCS:



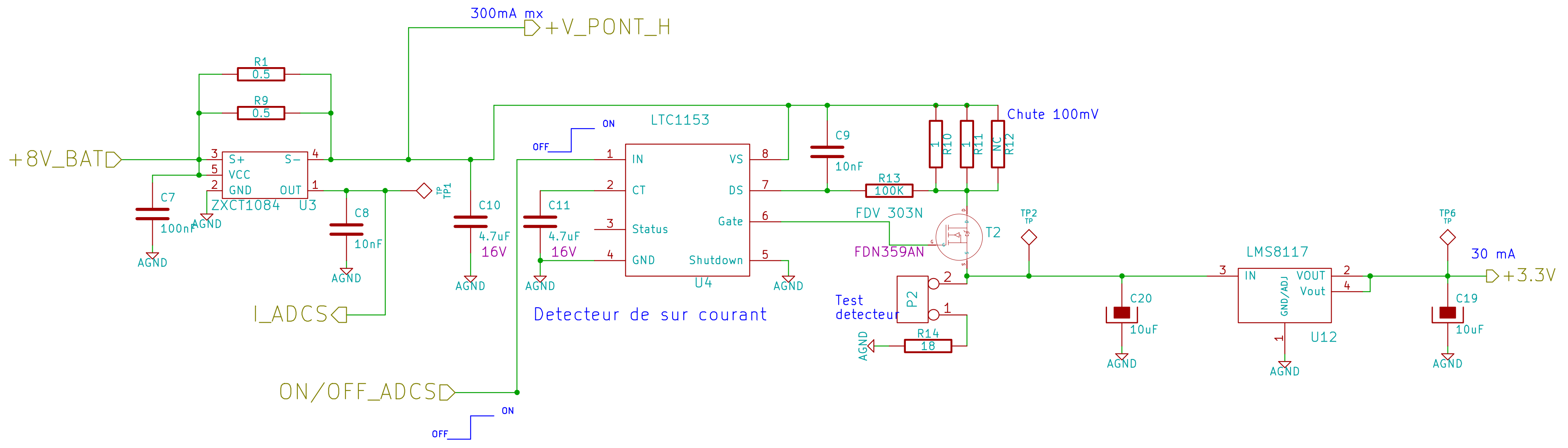
AMSAT-F / F6FAO
File: QB50_ADCS_ST.sch
Sheet: /
Title:
Size: A4 Date: 21 aug 2014 **Rev:**
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable Id: 1/4



File: pont_H.sch
Sheet: /Sheet_Pont_H/
Title:
Size: A4 Date: 21 aug 2014 **Rev:**
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable Id: 2/4



File: MicroP.sch
Sheet: /MicroP/
Title:
Size: A4 Date: 21 aug 2014 **Rev:**
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable Id: 3/4



File: ALIM_RX.sch
Sheet: /ALIM_RX/
Title:
Size: A4 Date: 21 aug 2014 **Rev:**
KiCad E.D.A. eeschema (2012-01-19 BZR 3256)-stable Id: 4/4

Chapter 3

External interfaces

Page 49-50 in the specifications for ODB

Chapter 4

Software Architecture

This is the heart of this document. We discuss here the architecture of the software, the various peripherals and the sensors mounted on the microprocessors, the various functions associated with them detailing every time the input, output and the configuration parameters for each of the peripherals.

We first begin by detailing how the software has been divided. Next we define the various classes that we have divided the software into. While the first division is an abstract division that we made to facilitate the development of the software, the structure of classes and their interdependance correspond to how the software will actually be implemented.

Once the abstract division and the structure of classes are defined we detail the correspondence of the various ports/ pins available on the microprocessor and the peripherals connected. Next, we define the functions corresponding to each of the divisions. Irrespective of their eventual use, all functions shall find themselves in one of three categories that we shall define in the section **Division of the software**.

4.1 Division of the software

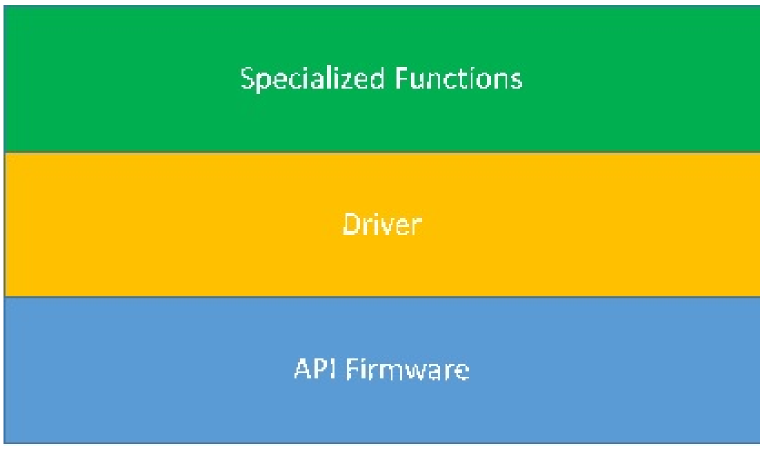


Figure 4.1: *Division of the software in three layers each independent of the other*

The ADCS software is divided into the following layers

1. **API Firmware**: In this layer, all the low-level functionalities are collected. By low-level functionalities, we mean all those functionalities which are not directly relevant to the functioning of the algorithms. This layer doesn't provide any data to the algorithms. Rather, this layer is concerned only with initializing and configuring the various peripherals that we have on the ADCS.

2. **Driver**: jhis layer acts as a bridge between the lower level API firmware and the higher-level specialized functions that we require to run our algorithms. Ideally, in this layer we find the functions which get the raw data from the sensors, convert or calibrate them as necessary and send them to the algorithms for further use.

3. **Specialized Functions**: The specialized functions are the functions which implement the algorithms for determining and controlling the attitude of the satellite. These functions are high-level functions and require the data provided to them by the *Driver* layer above to function.

4.2 Structure of classes

Once we have abstracted the software as being divided into layers depending on the kind of functions we will write for the software, we consider next the division of the software into dedicated classes. The class structure as envisaged for the ADCS is summarized in the graph below. The arrow head signifies that the a particular class gets a particular piece of data from the class where the arrow originates.

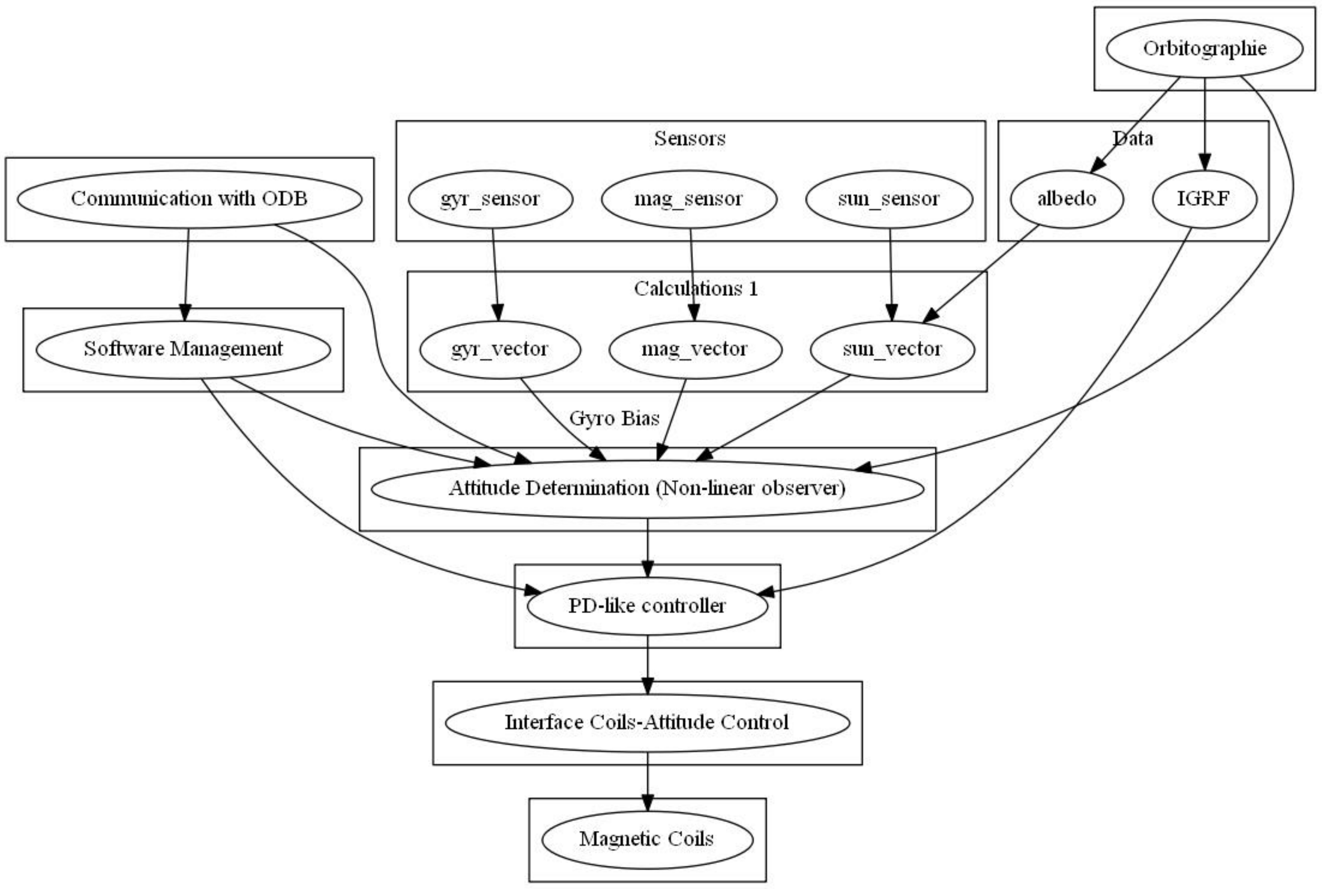


Figure 4.2: *Class structure for ADCS Software*

The various layers as shown in the class structure above are detailed below

1. **Sensors**: This layer corresponds to the API Firmware layer that we presented earlier. We have three types of sensors: Gyrometer, a magnetometer and 9 sun-sensors. This part of the software corresponds to the initialization and configuring the sensors (and the ports

to which they are associated on the microprocessor) so that they can start generating the data.

2. **Calculations 1**: This layer consists of a first layer of calculations effectuated once raw data is received from the sensors. In this layer, we receive the raw data from the sun-sensors, magnetometer and the gyrometer.

3. **Calculations 2**:This corresponds to a supplementary layer of calculations which is effectuated once the raw data has been received (not shown in the class structure). This step might take different forms for different sensors. For instance, once we have received the sun sensor raw data, the supplementary steps include converting the raw data into a voltage and further combining the data from the 9 sun-sensors into one sun-vector with three components. For the magnetometer however, we can eventually consider calibrating the data with the temperature. Identically for the gyrometer as well.

   Finally, this step also relates to reading the terrestrial magnetic field map and the albedo map from the external memory. All this data once collected, calibrated and treated will be given to the algorithms, which we explain below.

4. **Attitude Determination and Control Phase**: Once the data has been generated (after eventual calibrations), we have the layer where the actual control of the satellite takes place. The details of the algorithms will be presented in an ulterior section.

5. **PD-like controller**: This is what controls the satellite by providing us as output the magnetic moment to be generated in the magnetic coils. The controller then gives its output to the Coils-Attitude Control Interface.

6. **Interface Coils-Attitude Control**: This layer corresponds to the calculations to be done to configure the magneto-torquers. The magneto-torquers are controlled by Pulse-Width Modulation(PWM). For PWM, we require the frequency of modulation (in KHz) and the duty cycle (in percent). The result of the algorithms, however, is the magnetic moment to be generated by the magnetic torquers. Thus we require precise data on the behaviour of the magnetic coils used to convert this magnetic moment into a duty-cycle. (The frequency of modulation shall be fixed for all the coils)

7. **Magnetic Coils**: This is the layer which corresponds to the actual control that will be applied via the magnetic coils. The coils are powered via PWM for a fixed frequency usually aroun 15-20KHz.

4.3 Definition of ports

In this section we define the various ports, what their functionalities are and to which of the peripherals they are connected.

We also specify the alternate functions that need to be configured for each of the ports. These alternate functions could be Serial Communication (USART) for communicating with the ODB or an Analog to Digital Converter (ADC) for the sun-sensors.

Peripheral	Name of the port	Port on MPU	Logical State	Alternate Function(AF)	AF Details 1	AF Details 2
Sun Sensor	GS5	PC0	N/A	ADC	ADC_Resolution_8b	Channel10
	GS4	PC1	N/A	ADC	ADC_Resolution_8b	Channel 11
	GS3	PC5	N/A	ADC	ADC_Resolution_8b	Channel 15
	GS2	PB0	N/A	ADC	ADC_Resolution_8b	Channel 8
	GS1	PB1	N/A	ADC	ADC_Resolution_8b	Channel 9
PWM	F1	PB14	N/A	TIM1	Output Compare	
	R1	PB13	0 or 1	N/A		
	F2	PB12	0 or 1	N/A		
	R2	PB11	N/A	TIM2	Output Compare	
	F3	PB10	N/A	TIM2	Output Compare	
	R3	PA5	0 or 1	N/A		
Comm ODB	TX_ADCS	PA10	N/A	USART1_RX	Baud Rate	Bit size
	RX_ADCS	PA9	N/A	USART1_TX	Baud Rate	Bit size
MISC	On/Off Pont H	PB14	0 or 1	N/A		

Table 4.1: List of peripherals with alternate functions and other configurable parameters

4.4 API Firmware

This section details all the functions concerning the initialization and configuration of the peripherals.

Some generalities are in order. The initialization of any port goes through the following steps:

1. We first enable the peripheral clock corresponding to the sensor in question.
2. We then enable the GPIO clock corresponding to the sensor
3. We then enable the alternate function for the GPIO if required
4. We then configure the GPIO for use as an alternate function
5. We initialize the GPIO using its init function
6. Configure the functionalities for the alternate function
7. Finally initialize the alternate function using the corresponding init function.

One of the most important steps in the initialization of any port is to initialize the clock connected to the port. Each port is connected to one of two buses. We resume the various clocks connected to the various ports to serve as a reference to the reader.

Clock	Peripheral	RCC Peripheral
APB1	TIM2	RCC_APB1Periph_TIM2
	TIM3	RCC_APB1Periph_TIM3
	TIM4	RCC_APB1Periph_TIM4
	TIM5	RCC_APB1Periph_TIM5
	TIM6	RCC_APB1Periph_TIM6
	TIM7	RCC_APB1Periph_TIM7
	TIM12	RCC_APB1Periph_TIM12
	TIM13	RCC_APB1Periph_TIM13
	TIM14	RCC_APB1Periph_TIM14
	SPI2	RCC_APB1Periph_SPI2
	SPI3	RCC_APB1Periph_SPI3
	USART2	RCC_APB1Periph_USART2
	USART3	RCC_APB1Periph_USART3
APB2	TIM1	RCC_APB2Periph_TIM1
	TIM8	RCC_APB2Periph_TIM8
	USART1	RCC_APB2Periph_USART1
	USART6	RCC_APB2Periph_USART6
	ADC	RCC_APB2Periph_ADC
	ADC1	RCC_APB2Periph_ADC1
	ADC2	RCC_APB2Periph_ADC2
	ADC3	RCC_APB2Periph_ADC3
	TIM9	RCC_APB2Periph_TIM9
	TIM10	RCC_APB2Periph_TIM10
	TIM11	RCC_APB2Periph_TIM11
AHB1	GPIOA	RCC_AHB1_GPIOA
	GPIOB	RCC_AHB1_GPIOB
	GPIOC	RCC_AHB1_GPIOC
	GPIOD	RCC_AHB1_GPIOD
	GPIOE	RCC_AHB1_GPIOE
	GPIOF	RCC_AHB1_GPIOF
	GPIOG	RCC_AHB1_GPIOG
	GPIOH	RCC_AHB1_GPIOH

Table 4.2: Clocks connected to each of the peripherals

The configuration and initialization of all the functionalities presented below will follow the list of steps detailed.

4.4.1 PWM

4.4.1.1 Initialization

The following functions will be used for initialization purposes

1. **Setting on the H-Bridge**

   Syntaxe: void setPWMON()

   This function switches on the H-bridge which in its turn control the coils.

Name of the port	Port on MPU	Logical State
ON/OFF PONT H	PB14	1

Table 4.3: PWM-ON

2. **Configuring the ports which determine the current direction**

   Syntaxe: void configDirs()

Name of the port	Port on MPU	
R1	PB13	To be configured as General Purpose Input/Output
F2	PB12	
R3	PA5	

Table 4.4: Configure the ports for the direction

3. **Configure the ports which supply power**

   Syntaxe: void configTorquers()

   This function configures the ports connected to the coils to be used as Timers.

Name of the port	Port on MPU	
F1	PB14	TIM1
R2	PB11	TIM2
F3	PB10	

Table 4.5: Configure the ports as Timers

4. **Configure TIM1**

   Syntaxe: void configTIM1(uint32_t PWM_freq)

   Once the ports have been configured to be used as timers, we need to configure the timers to function at the frequency we require. The TIM1 is connected to the AHB1 Bus which has a clock frequency of 168MHz. Thus one first prescales the timer to be used as

5. **Configure TIM2**

   Syntaxe: void configTIM2(uint32_t PWM_freq)

6. **Configure PWM1**

   Syntaxe: void configPWM1(uint32_t duty_cycle)

7. **Configure PWM2**

   Syntaxe: void configPWM2(uint32_t duty_cycle)

8. **Configure PWM3**

   Syntaxe: void configPWM3(uint32_t duty_cycle)

4.4.1.2 Termination

4.4.1.2.1 Configuring the ports We use the following ports to be used for providing power to the magnetic coils using PWM. The ports: **PB14, PB11, PB10** will be configured to be used as timers, whereas **PB13, PB12** and **PA5** will be used as general input/output ports. The last three ports will be used to determine the direction of the current in the coils.

Peripheral	Name of the port	Port on MPU	Logical State	Alternate Function(AF)	AF Details 1	AF Details 2
PWM	F1	PB14	N/A	TIM1	Output Compare	
	R1	PB13	0 or 1	N/A		
	F2	PB12	0 or 1	N/A		
	R2	PB11	N/A	TIM2	Output Compare	
	F3	PB10	N/A	TIM2	Output Compare	
	R3	PA5	0 or 1	N/A		

Table 4.6: List of PWM Ports

4.4.2 ADC

4.4.3 USART

4.5 Driver

4.6 Specialized Functions

Chapter 5

Data formats

Chapter 6

Testing

Chapter 7

Referentials