A REPORT

ON

Development of the flight model of an inertial asset control system for ARAMIS satellite

Submitted by:-

AAKASH JAIN
Final Year Undergraduate Student
BITS Pilani, India

Guided by:-

Prof. Leonardo Reyneri
Department of Electronics,
Politecnico di Torino, Italy

Politecnico di Torino
Corso Duca degli Abruzzi, 24 - 10129
Torino, ITALY
June-August 2009

ACKNOWLEDGEMENTS

I hereby express my deep sense of gratitude to Prof. Leonardo Reyneri, Department of Electronics, Politecnico di Torino for providing me with a wonderful opportunity in the form of this internship program.

I take this opportunity to express our heartfelt thanks to Marco Borri and Maurizio Tranchero for their technical support and guidance.

I am also thankful to all the officials and my colleagues in my department for providing necessary technical assistance and enthusiastic work environment.

ABSTRACT

The present report discusses the work carried out on the project "Development of the flight model of an inertial asset control system for ARAMIS satellite". ARAMIS is the second satellite being developed at Politecnico di Torino after developing a satellite called PiCPoT (Piccolo Cubo del Politecnico di Torino) in July 2006, which had as its main purposes the test on the operation of commercial components (COTS) in space and the acquisition of images and spatial parameters. The present Satellite ARAMIS will be one step ahead of PiCPoT and the central concept kept in mind is to introduce the concept of modularity of the satellite which will significantly reduce costs of design and development of the satellites. Defining various standard modules which are compatible with each other and then assembling these modules in the quantities and manner required by the mission.

Among the various modules in the ARAMIS Satellite one is the Attitude and Orbit Control Systems (**AOCS**) subsystem which is one of the most important subsystems of a satellite. Its main function is to control the attitude and the orbit of the satellite. It is often the most complex of the satellite subsystems. Its function is to control the attitude and the orbit of the satellite. This is a critical function because without attitude or orbit control the satellite will gradually – sometimes in a matter of seconds, sometimes after several days – become unable to fulfil its mission objectives.

AOCS' chief task is to periodically collect measurements from sensors and convert them into commands for actuators.
There are two major **Attitude Actuators Subsystem**(1B21) in the ARAMIS which are:

1) *Magnetic Torque Actuator (1B211)*

2) *Reaction Wheel Actuator (1B212)*

Also there are two major ***Attitude Sensor Subsystem***(1B22) which are:

1) *Magnetometer Sensor (1B221)*

2) *Gyroscopic Sensor (1B222)*

This project aimed towards development of all the necessary Software and Hardware for the 'Reaction Wheel Actuator' and 'Gyroscopic Sensor'. While all the necessary software and hardware for other two module namely 'Magnetic Torque Actuator' and 'Magnetometer Sensor' are been developed in an another project.

The basic principal which governs the control of orbit and rotation of the satellite is the law of conservation of angular momentum. A reaction wheel placed inside the satellite is rotated at a controlled speed for a calculated duration. This in turns rotates the satellite in opposite direction in order the conserve the total Angular momentum of the satellite.

The code has been written for the microcontroller (TI MSP430) which reads the gyroscopic sensor, controls the reaction wheel and communicates with the central processor of the satellite. The verification has been done by downloading the code onto the circuit with the microcontroller TI MSP430F2274. All the modules necessary for the communication with the central processor have been written and tested. The modelling for the system has been done in UML through Visual Paradigm and the compilation, simulation and assembly code generation along with downloading onto the chip has been done by IAR Embedded Workbench for MSP430.

All the newly developed class-diagrams, Sequence diagrams and Use-case implementations have been integrated into the overall ARAMIS satellite code and model.

TABLE OF CONTENTS

1. INTRODUCTION

The Politecnico di Torino has developed a satellite called PiCPoT (Piccolo Cubo del Politecnico di Torino) in 2006, which had as its main purposes the test on the operation of commercial components (COTS) in space and the acquisition of images and spatial parameters of . In addition to these goals of scientific-technical, the establishment of this satellite has also had the advantage of involving many departments of the Politecnico di Torino in the design of the modules that make up the satellite involving teachers, researchers, students and doctoral candidates. The satellite PiCPoT has a cubic shape of side about 12 cm. In five of the six faces of PiCPoT there is a solar panels that convert solar energy into electrical energy used by electronics on board, while the sixth face of the cube have two antennas used for communication with the ground station and the cameras. The creation of this satellite was completed in July 2006 with the launch of the satellite through the Russian Dnepr carrier. Due to a problem of the hydraulic pitcher has not been possible to verify the behaviour of the satellite in orbit. With the experience gained during the design of the satellite PiCPoT have created the basis for a second project, the satellite **Aramis**. The project had the main PiCPoT "defect" to be developed ad-hoc for the mission, so we had a usability project very low which made the high cost of design. By ARAMIS satellite we want to introduce the concept of modularity of the satellite to remedy the defects in PiCPoT. Defining the standard modules are compatible with each other, the modules can be assembled in the quantities and manner required by the mission. This approach reduces costs of design and development as the modularity of design allows the use of modules in different missions with different architectures thereby spreading the cost of the project.

ARAMIS project contains various modules which have been divided into various categories. Figure 1 shows the various modules and sub-modules defined in ARAMIS.

This report deals with the design, implementation and testing of the 'Attitude and Orbit Subsystem' (1B2 in figure 1) **of the satellite ARAMIS.**

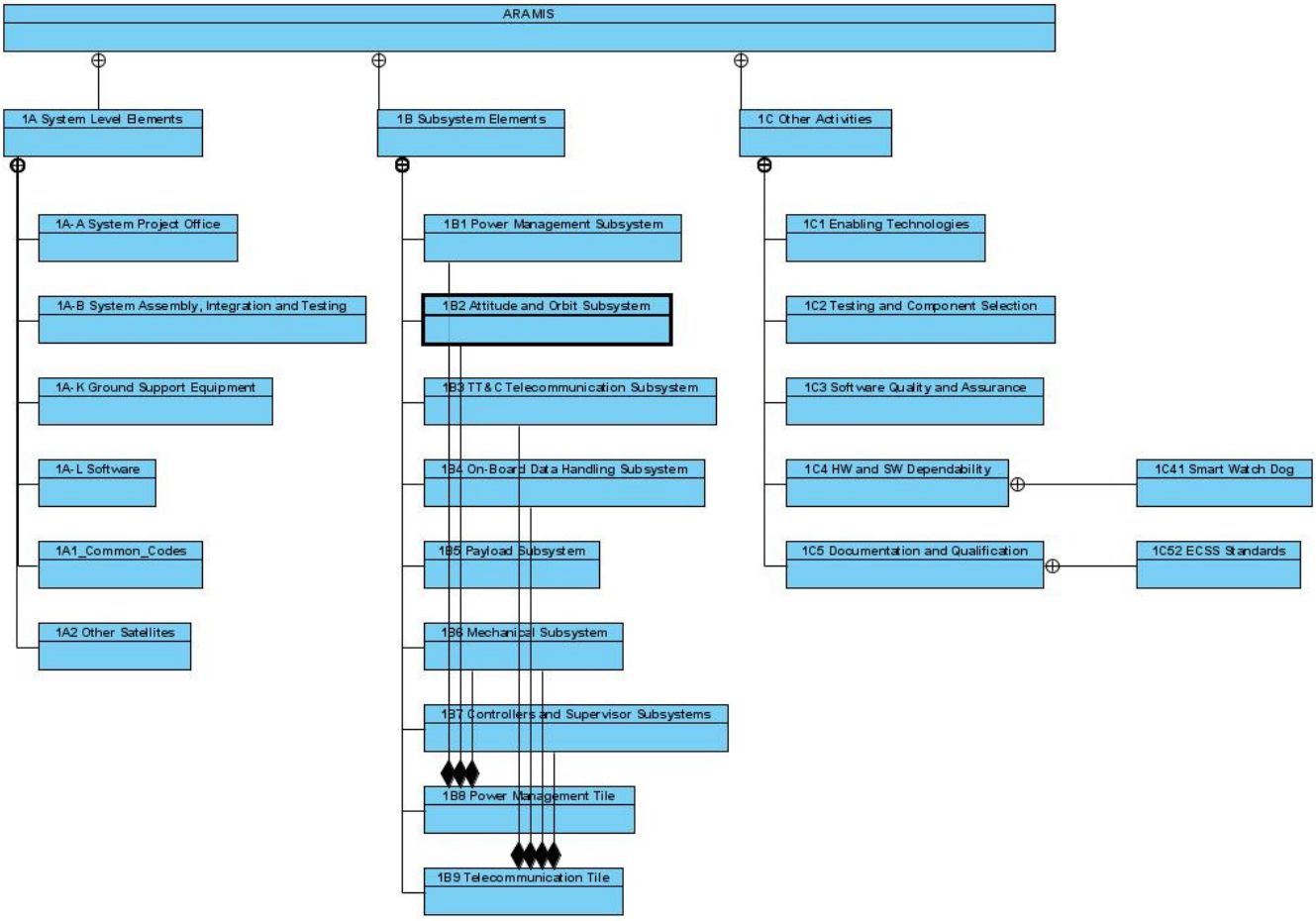


Figure 1: ARAMIS Modules

The detailed description of the ‘Attitude and Orbit Subsystem’ module is shown in Figure 2. Among the various other things, the present work focuses on design and implementation of the submodules ‘Reaction Wheel Actuator’ , ‘Gyroscopic Sensor’ and ‘Centralized Attitude Controller’ . The position and link between all these sub-modules can be seen in the figure 2.

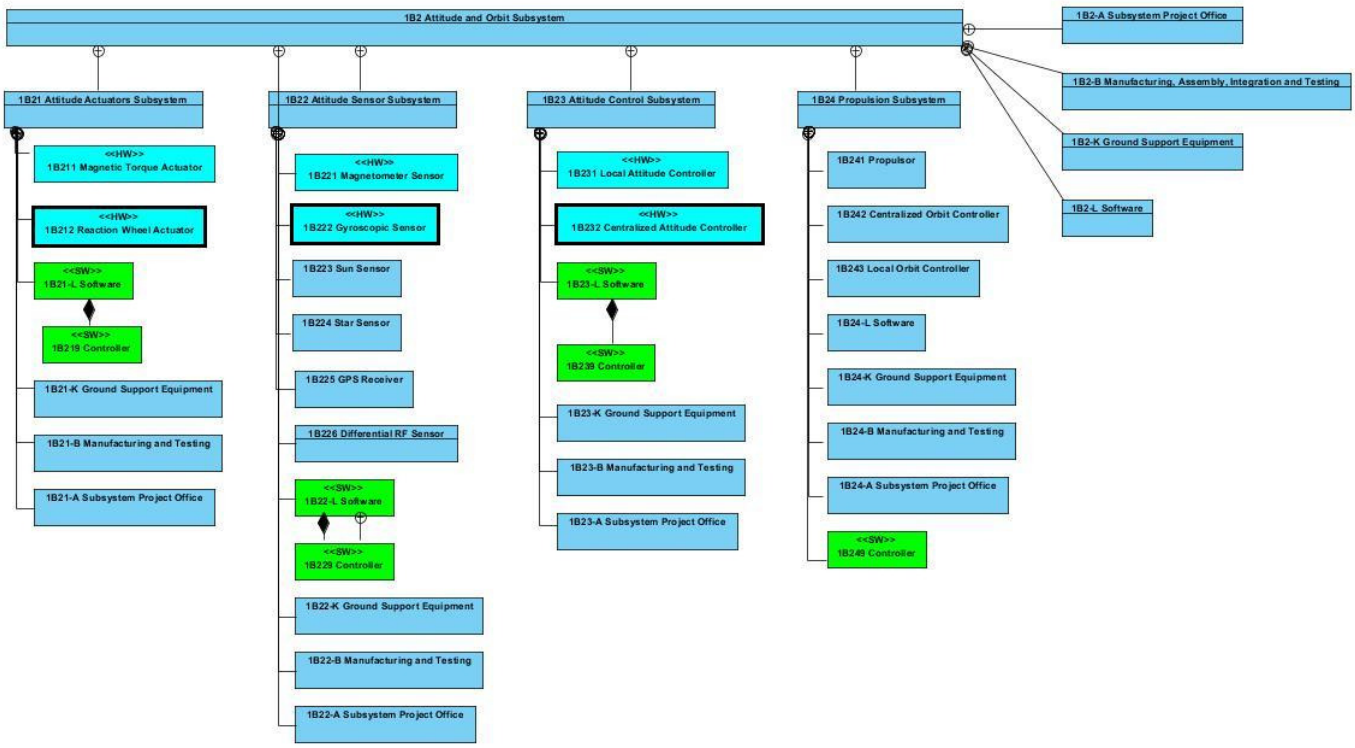


Figure 2: Attitude and Orbit Subsystem

2. DESCRIPTION OF THE DEVICES

Various devices has been used in designing and implementing the required system. The major devices among them are:

1) Microcontroller
2) Motor
3) Brushless Motor Driver
4) Gyroscope

2.1) Microcontroller

The controller must have the necessary features to manage the two drivers, sensors, A / D conversions and communication with the outside (in UML called AOCS).

The controller must have the following characteristics:

• 6-pin to operate the brushless motor driver

• 5 pin to operate the solenoid driver.

• 1 timer for timed feeding of the solenoid

• 1 timer for timed circuit set / reset of the magnetometer

• 2 pin to force the signals to set and reset the magnetometer

• 1 A / D converter.

• 2-channel (pin) for ADC for the signals from the gyroscope

• 2-channel (pin) for ADC for the signals from the magnetometer

• 1 channel (pin) for ADC for the signal from the solenoid driver.

• 1 UART communication interface used for communication and testing.

• low consumption.

So the minimum features that the controller must have are:

• at least 2 timers.

• an ADC with at least 5 channels.

• 12-pin general purpose.

• A UART interface.

• low power consumption.

The controller is chosen **MSP430F2274** (family MSP430x2xx) of Texas Instruments, which has the following characteristics:

• Low voltage (1.8 V to 3.6 V).

• Low Power (Active mode: 270μA at 1MHz, 2.2 V Standby mode: 0.7 μA).

CPU • 16-bit RISC.

• Configurable Clock up to 16MHz internally.

• 2 timers (Timer_A, Timer_B) to 16 bits.

• Universal Serial Communication Interface (UART, SPI, I2C).

• A / D 10-bit, 200-Ksps, internal voltage reference.

• Package 38-pin TSSOP.

The pinout of this microcontroller is in the Figure 3 below.

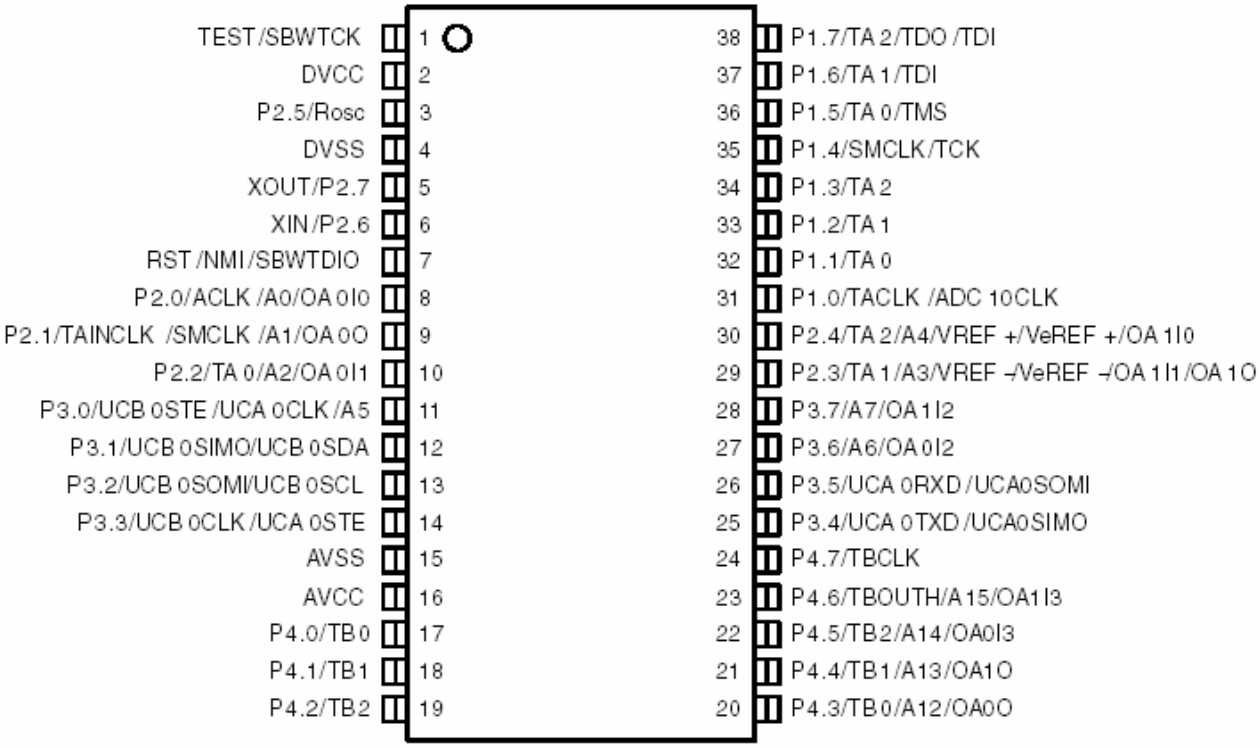


Figure 3: Pinout of the Microcontroller MSP430F2274

The schematic of this structure is shown in the figure 4:

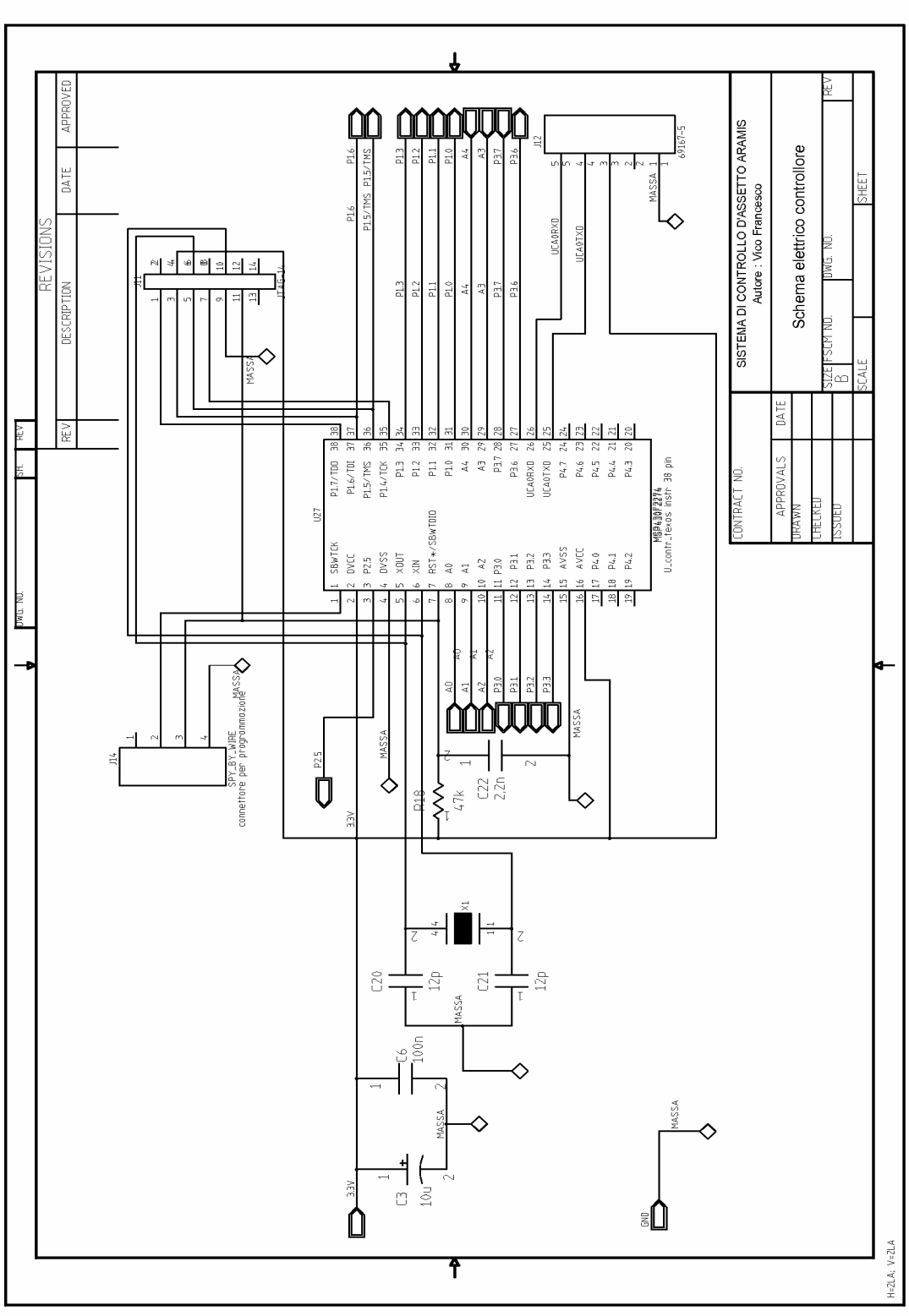


Figure 4: Schematic of the Microcontroller MSP430F2274

As can be seen in the schematic diagram, the microcontroller is powered with a voltage of 3.3 V via pins 2 and 16. On pin 5 and 6 has been connected to a quartz X1 (SE2418CT) with resonance frequency

of 32.768 KHz with two external capacitors (C20 and C21) of the value of 12pF, as recommended by the datasheet of quartz.
Connectors J11 and J14 connectors are used for programming (microcontroller) via 4-wire JTAG and JTAG 2-Wire (Spy Bi-Wire).
The J12 connector is the connector used for serial communication.
The A / D converter will be operated with a clock, using the internal oscillator to the microcontroller.

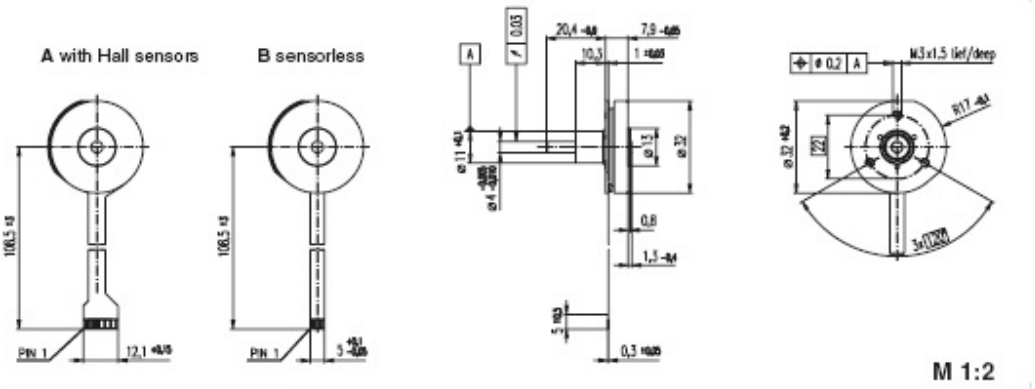
2.2) Motor

In order to rotate the wheel of inertia a reaction wheel or motor is required. The motor chosen to be used in ARAMIS is brushless motor of the EC 32 flat Maxon Motor. This motor fits best to our requirements of high reliability, small weight, high torque and easily integrable into the satellite.

Various details of the motor is in the next diagram:

maxon flat motor

EC 32 Flat motor ∅32 mm, brushless, 6 Watt



Stock program
Standard program
Special program (on request!)

			Order Number	
		with Hall sensors	200187	
		sensorless		200138
Motor Data				
1	Assigned power rating	W	6	6
2	Nominal voltage	Volt	9.0	9.0
3	No load speed	rpm	8600	8600
4	Stall torque	mNm	20	20
5	Speed / torque gradient	rpm / mNm	479.0	479.0
6	No load current	mA	110	110
7	Terminal resistance phase to phase	Ohm	4.50	4.50
8	Max. permissible speed	rpm	12000	12000
9	Max. continuous current at 5000 rpm	A	1.03	1.03
10	Max. continuous torque at 5000 rpm	mNm	8.70	8.70
11	Max. efficiency	%	80.0	80.0
12	Torque constant	mNm / A	9.5	9.5
13	Speed constant	rpm / V	1007	1007
14	Mechanical time constant	ms	70.0	70.0
15	Rotor inertia	gcm²	13.9	13.9
16	Terminal inductance phase to phase	mH	1.070	1.070
17	Thermal resistance housing-ambient	K / W	6.8	6.8
18	Thermal resistance winding-housing	K / W	7.4	7.4
19	Thermal time constant windings	s	3.7	3.7
20	Thermal time constant stator	s	16.1	16.1

Specifications

- Axial preload > 5 N
- Max. ball bearing loads
  axial (dynamic) 2.8 N
  radial (7.5 mm from flange) 5.5 N
  Force for press fits (static) 50 N
  (static, shaft supported) 1000 N
- Ambient temperature range -40 ... +100°C
- Max. permissible winding temperature +125°C
- Weight of motor 32 g
- Version with and without Hall sensors
- 8 pole permanent magnet
- 3 phased coil stator with 2 pole shoes each
- Values listed in the table are nominal.
- Connection with Hall sensors sensorless

Pin 1	3.5 ... 20 VDC	Motor winding 1
Pin 2	Hall sensor 3	Motor winding 2
Pin 3	Hall sensor 1	Motor winding 3
Pin 4	Hall sensor 2	⅄ neutral point
Pin 5	GND	
Pin 6	Motor winding 3	
Pin 7	Motor winding 2	
Pin 8	Motor winding 1	

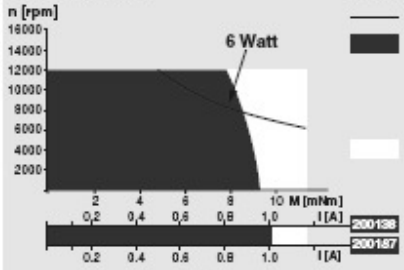
- Adapter — Order number — Order number
  see p. 275 — 220300 — 220310
- Connector — Article number — Article number

AMP	1-487951-1	487951-4
MOLEX	52207-1190	52207-0490
MOLEX	52089-1110	52089-0410

Pin for design with Hall sensors:
FPC, 11 pole, pitch 1.0 mm, top contact style

- For wiring diagram for Hall sensors, see p. 26

Operating Range



Comments

Details on page 149

Curve of constant assigned power rating

Continuous operation
In observation of above listed thermal resistances (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.
= Thermal limit.

Short term operation
The motor may be briefly overloaded (recurring).

Recommended Electronics:

AECS 35/3	page 262
DEC 24/1	263
DEC 50/5	264
DECV 50/5	265
EPOS 24/1	271
Notes	17

April 2005 edition / subject to change

Figure 5: Motor Datasheet

The schematic of the connections of the overall circuit made in order to control the reaction wheel is as shown in the figure:

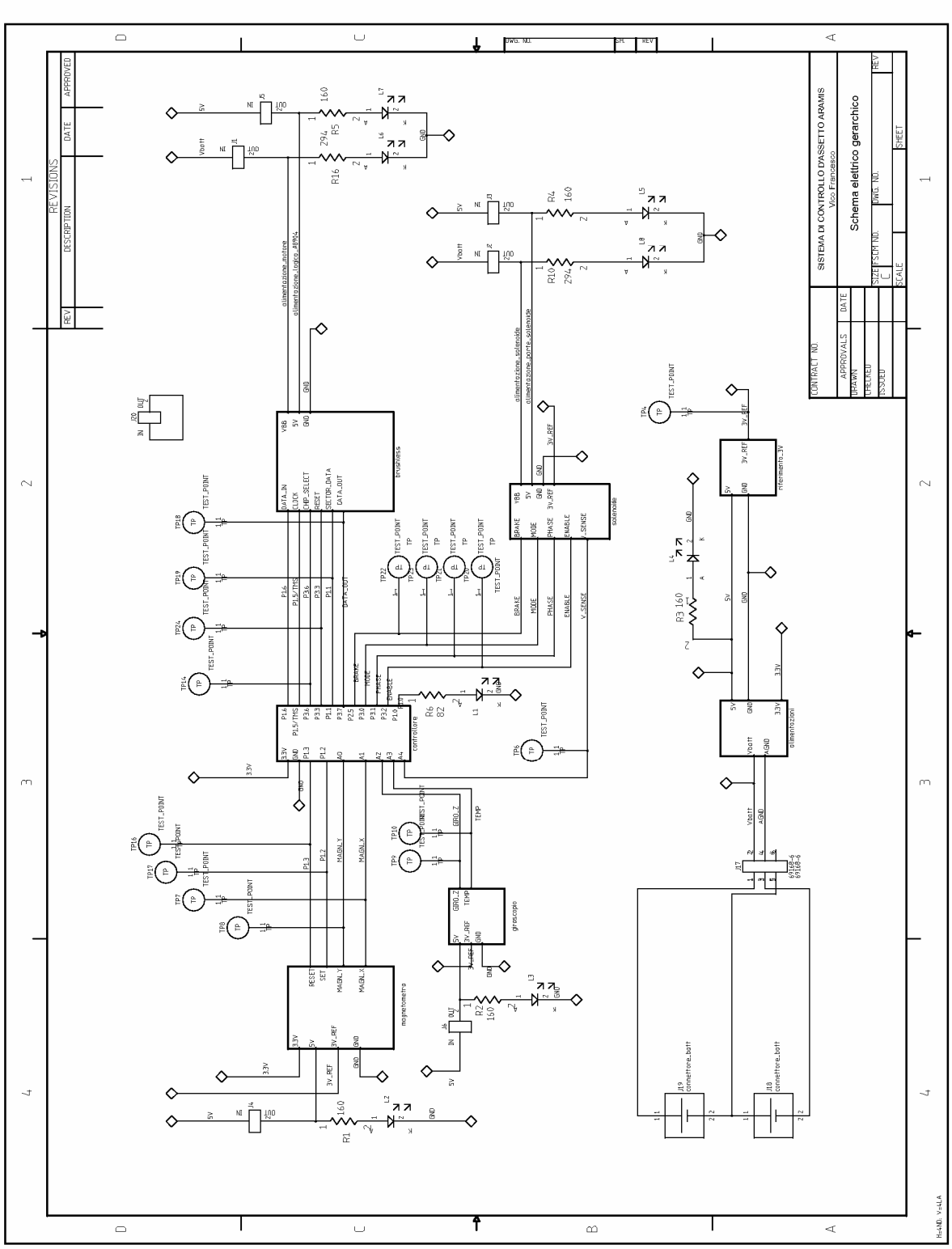


Figure 6: Schematic of overall circuit

2.3 Brushless Motor Driver

It is necessary to control the brushless motor so that it does rotate the wheel of inertia at a speed imposed by the desired angular momentum. The control is imposed by the means of a Motor Driver which can communicate with the processor and accordingly gives signals to the motor.

To drive the motor you chose to use the driver for brushless motors A8904 of Allegro Microsystems.

Its main features are :

- Typical voltage power control logic at 5V.
- Supply voltage of the load (motor) from 4 to 14V.
- Integrated three-phase bridge.
- Ability to provide output currents up to 1.4 A (3A peak)
- Programming serial speed and direction of rotation
- Serial programming of the maximum current absorbed by the load
- Programming of the serial number of poles of motor
- Integrated speed control based on back-EMF detection
- Package 24-pin SOIC.

The pin driver is shown in Figure 7.

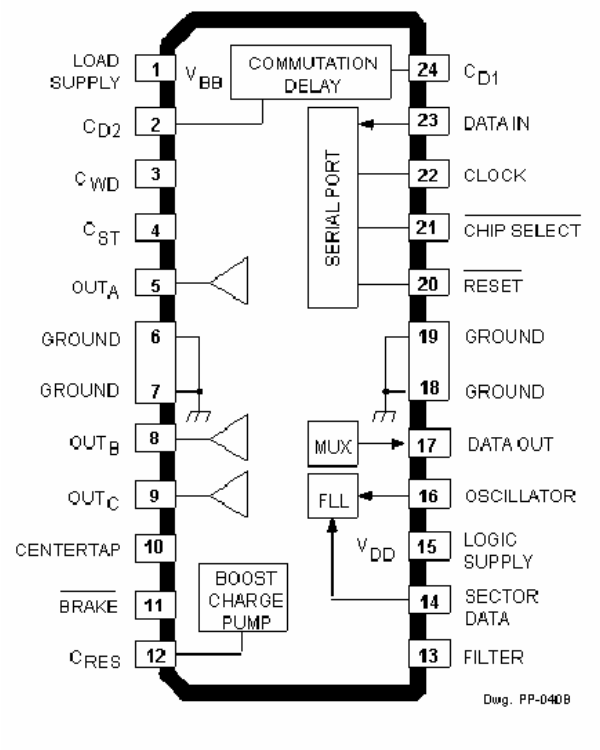


Figure 7: Pin Driver A8904

Serial Programming of the Brushless Motor Driver

The commands are sent from the microcontroller through the serial 29 bit.
When CHIP SELECT is low, the data is placed on the serial port on the rising edge of the clock with the MSB (D28) first.
At the end of the programming of the driver CHP SELECT returns to the logical value high.
The timing to be observed are given in the following figure 8.

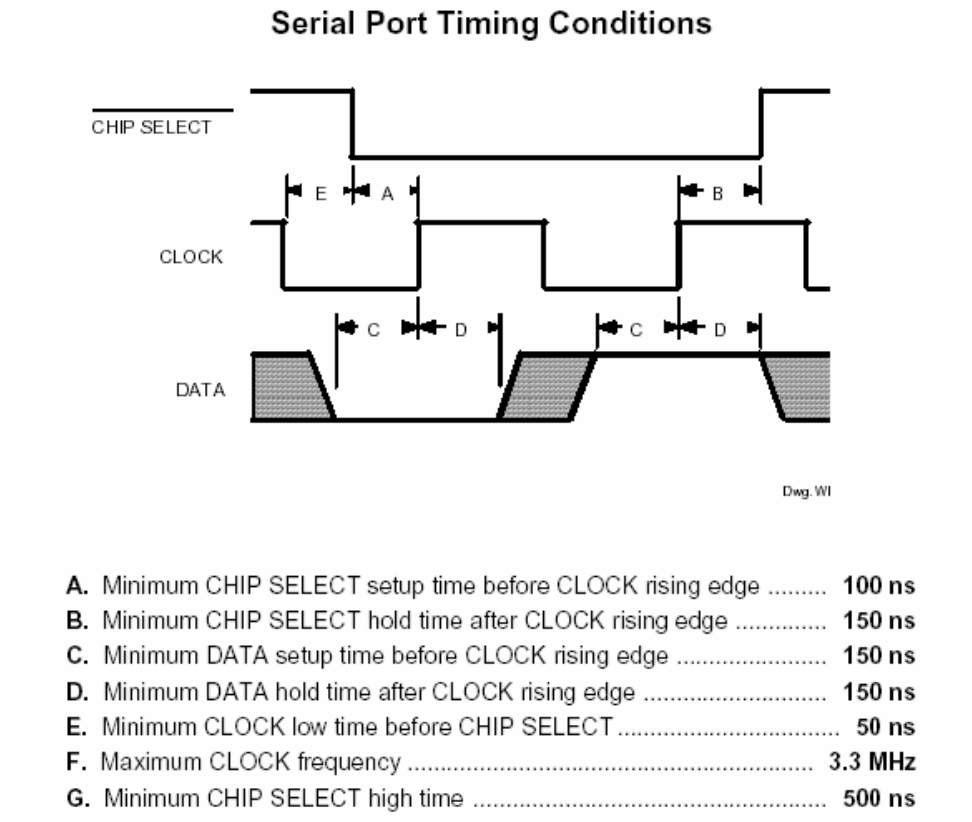


Figure 8: Timing for programming driver A8904

Through the 29-bit (D0. .. D28) programming, you can set the maximum current that can absorb the engine, the number of poles of the motor, the direction and speed of rotation and more. The programming of the desired speed is achieved using 14-bit (D5. .. D18), these bits are used to set the total count expresses the number of oscillations must count the speed control to the desired speed, i.e. :

$$total\ count = \frac{(60 \cdot f_{osc})}{desidered\ motor\ speed\ (rpm)}$$

To set this speed should be set to logical high value bits D14, D13, D11, D10, D9, D8, D7, D6 and D5.

Bit number	Count number
D5	16
D6	32
D7	64
D8	128
D9	256
D10	512
D11	1,024
D12	2,048
D13	4,096
D14	8,192
D15	16,384
D16	32,768
D17	65,536
D18	131,072

Figure 9: bit values associated with the programming

The schematic of the Brushless Motor Driver is as follows:

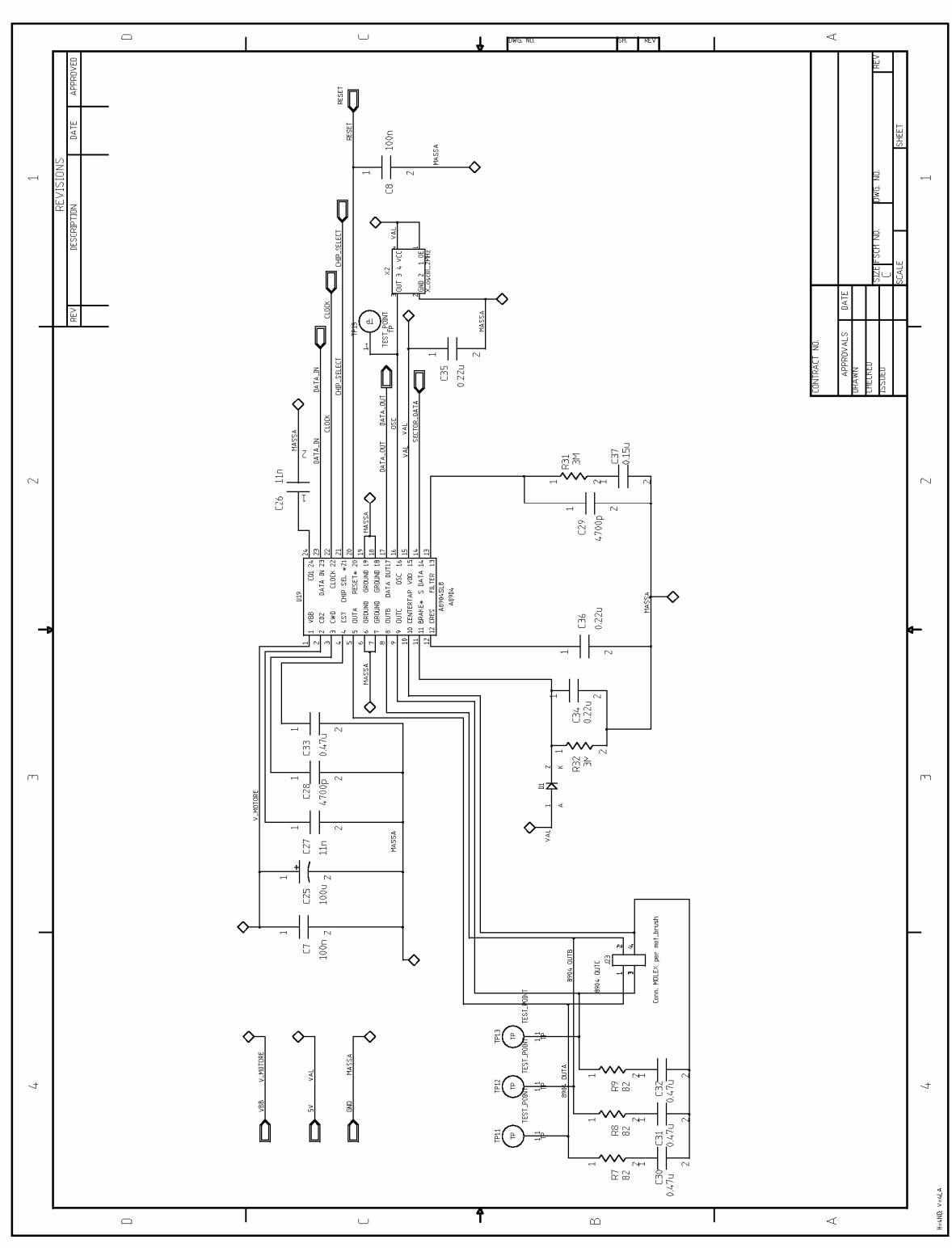


Figure 10: Schematic of the Brushless Motor Driver

2.4 Gyroscope

Gyroscope measure the temperature of the system and the angular rate of the satellite.
The gyro sensor must be able to measure the angular velocity around the axis perpendicular to the tile of PowerSupply. Maintaining the same system of reference of the magnetometer, i.e., assuming that the solenoid and then printed lies at x, y, the gyroscope must acquire the angular velocity around the z axis as shown in following figure 11.

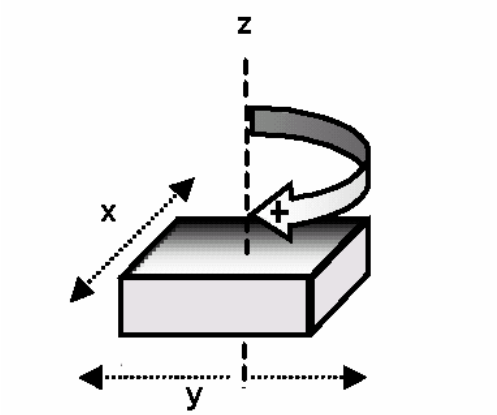


Figure 11: Axis measuring gyro sensor

This sensor must be capable of measuring speed of rotation of the satellite equal to ± 2 rpm (± 12 ° / s).

2.4.1) Gyro sensor

The characteristics that the required sensor must have are:
• Measuring z axis (yaw-rate).
• Low consumption.
• Measuring range greater than or equal to ± 12 ° / s.

The sensor chosen is the ADXRS401 Analog Device.

Its main features are:
• Measurement of the angular velocity around the z axis, i.e. the axis perpendicular to the surface mounting of the sensor.
• Supply Voltage 5V typical.
• Measuring range ± 75 ° / s.
• Output voltage at 0 ° / s typical 2.5 V.
• Typical Sensitivity 15mV / ° / s.
• Integrated temperature sensor.
• Package 7mm x 7mm x 3mm 32-pin BGA surface mount.
• Consumption of up to about 8MA.

The pinout of this sensor is shown in following figure which represents the component side seen from the bottom:

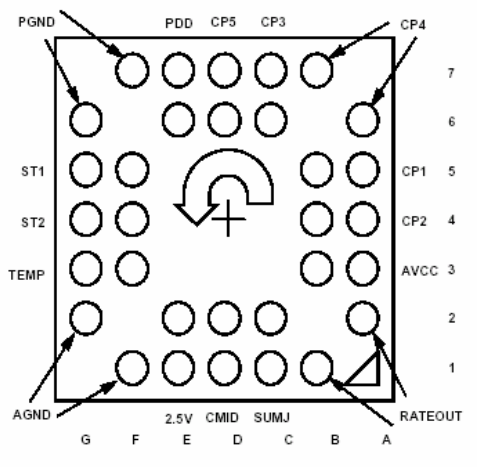


Figure 12: Gyroscopic sensor seen from the bottom

The 32-pin BGA component are reported in pairs in 16-pin logic is 12-pin physical interior is a replica of the external pin.
This solution probably has been adopted to make the PCB mounting more stable, without increasing the footprint of the component.
Referring to the pinout above, the gyro sensor output will be::

$$V_{ADXRS401_{RATEOUT}} = V_0 + S \cdot K_S$$

where:

- K_S represents the sensitivity of the gyro sensor in mV / ° / s.
- S is the angular velocity in ° / s, which varies from -12 ° / s at +12 ° / s.
- V_0 is the tension $V_{ADXRS401_{RATEOUT}}$ with $S = 0$.

Built in gyro sensor is a temperature sensor, whose output voltage is:

$$V_{ADXRS401_{TEMP}} = K_T \cdot (T - 25) + 2{,}5V$$

where:

- K_T represents the sensitivity of the temperature sensor in mV / ° C.
- T is the temperature expressed in ° C ranging from -40 ° C to +85 ° C. 85°C.

2.4.2) Conditioning circuit

The schematic circuit packaging is the figure below.

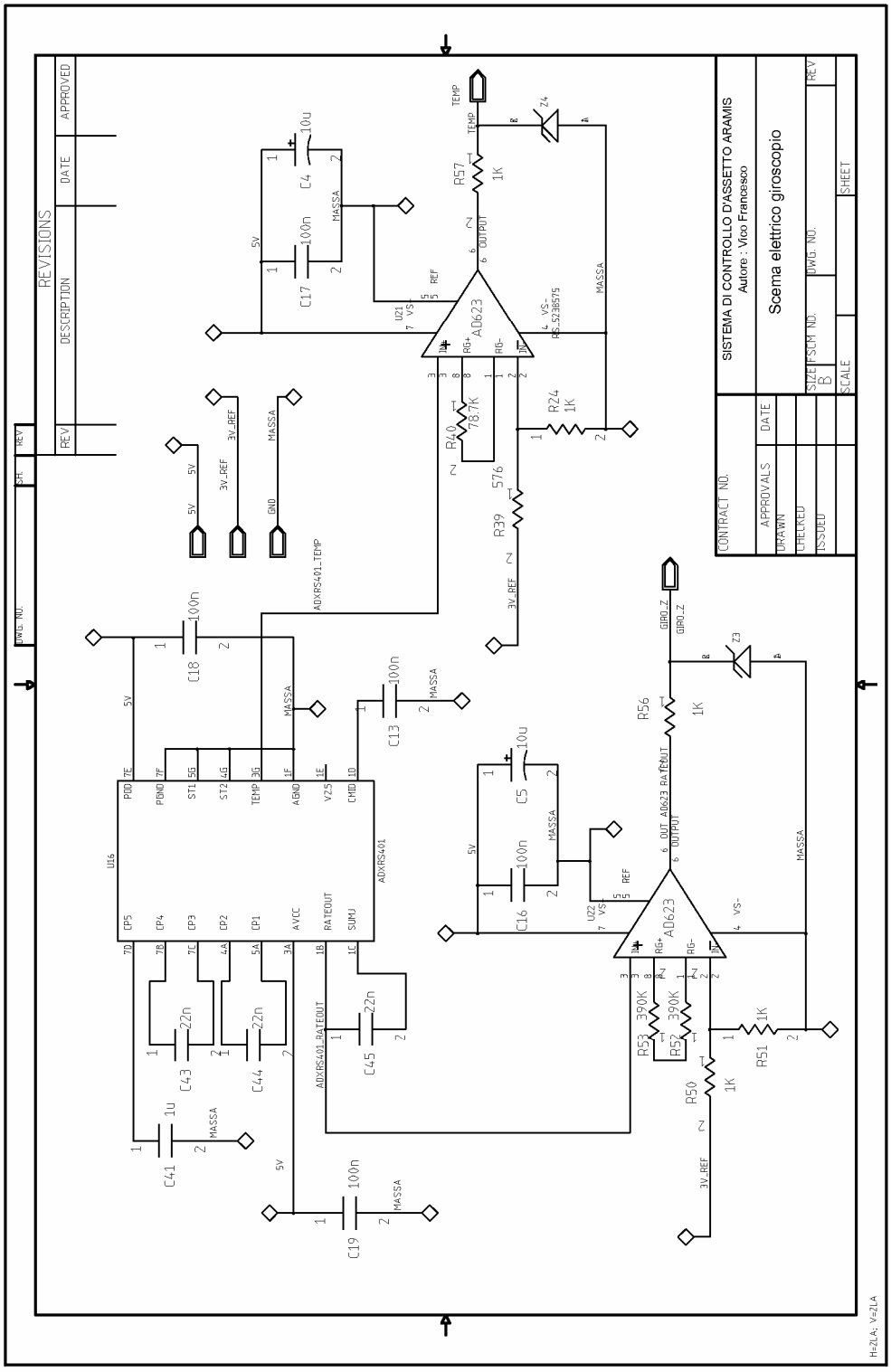


Figure 13: Gyroscopic schematic

The output voltages of the sensor must be scaled so that it can be acquired from the converter analog / digital controller.
It is therefore necessary to constrain the two voltages so that they remain in a range between 0 and 2.5 V.

The 0V represent the minimum output of the gyroscope and the 2.5V represents the maximum output of the gyroscope.

The above is an important condition as the code in the Gyroscope Class has been written assuming the above condition is fulfilled.

3. SYSTEM SPECIFICATIONS

Use Case Diagram

ATTITUDE CONTROL SUBSYTEM (1B232 CENTRALIZED ATTITUDE CONTROLLER)

The following Use-Case diagram shows the use-cases which has to be implemented in the 'Centralized Attitude Controller' subsystem.

Various Use-Cases has been categorized in terms of the action they perform, i.e. send a command to the module, get the satellite housekeeping data from the module containing the information, send a command to read / write some data, set the configuration of the module etc.

Figure 14: Use Case Diagram

The detail/purpose of each Use-case is as follows :

Name	Documentation
1B232 Centralized Attitude Controller	The object running on the On-Board Computer which controls the attitude of the whole satellite.
Module Commands	The **CPU** actor sends a data-less command to the **System**. The **CPU** can use up to 8 different **Module Command** commands, to issue as many commands to the system. It uses the **SPI Protocol - Command Only** use case, by issuing the **CMD_COMMAND_x** command, where **x** (0..7) identifies the command type; **x** does not identify the sequence in which messages are written, but the **Designer**-defined type (therefore the effect of the command). The **Designer** can use as many message types he wants. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
EnableWheel	Enables reaction wheel. Any further action of reaction wheel will be carried on. Uses **Module_Commands** use case of 1B45 package, with the **CMD_COMMAND_0** command.
DisableWheel	Disables reaction wheel. Any further action of reaction wheel will be disregarded. If the reaction wheel is in process of rotation while the **DisableWheel** command is given, the wheel is stopped immediately. Uses **Module_Commands** use case of 1B45 package, with the **CMD_COMMAND_1** command.
deactivateGyroscope	Disables gyroscope (removing power supply). Uses **Module_Commands** use case of **1B45** package, with the **CMD_COMMAND_2** command.
activateGyroscope	Enables gyroscope (giving power supply and sampling its output). Uses **Module_Commands** use case of 1B45 package, with the **CMD_COMMAND_3** command.

Get Module Housekeeping	Returns last measured housekeeping data (see use case **Housekeeping** for details). It uses the **SPI protocol - Read Data** use case by issuing the **CMD_GET_HOUSEKEEPING** command. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
getSpin	It measures and returns the rotational speed around z axis with a signed 16-bits integer; unit is 0.0001 rad/s. It provides housekeeping data (spin) to**Get Module Housekeeping** use case of 1B45 package, in the **Housekeeping_commons[0]** variable.
getTemperature	It measures and returns the temperature with a signed 16-bits integer; unit is 0.01°C. It provides housekeeping data (temperature) to**Get Module Housekeeping** use case of 1B45 package, in the **Housekeeping_commons[1]** variable.
Read Module Data	The **CPU** actor reads up to 256B of **Designer**-defined data from the **System**. The **CPU** can use up to 8 different **Read Module Data** commands, to read messages from as many different subsystems. It uses the **SPI Protocol - Read Data** use case, by issuing the **CMD_READ_DATA_x** command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are read, but the **Designer**-defined type (therefore the source subsystem). The **Designer** can use as many message types he wants. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
isRotating	Checks whether the reaction wheel is in process of rotation with respect to satellite or is stopped. Returns 1 if the motor is running and returns 0 if the motor is stopped. Uses **Read Module Data** use case of 1B45 package, with the **CMD_READ_DATA_0** command which returns a 1-byte message containing the status of the reaction wheel.

Write Module Data	The **CPU** actor sends up to 256B of **Designer**-defined data to the **System**. The **CPU** can use up to 8 different **Write Module Data** commands, to issue messages to as many different subsystems. It uses the **SPI Protocol - Write Data** use case, by issuing the **CMD_WRITE_DATA_x** command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the **Designer**-defined type (therefore the destination subsystem). The **Designer** can use as many message types he wants. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
actuateReactionWheel	It generates a user-defined angular momentum by means of the reaction wheel. The momentum is generated around z axis. Positive momentum will push x axis towards y axis If angular momentum is zero, the wheel is stopped. If momentum is higher than max allowed, wheel will rotate at max speed. If momentum is lower than min allowed, wheel will rotate at min speed. Angular momentum is signed 16-bits integer in units of 10 gcm2/s. Uses **Write Module Data** use case of 1B45 package, with the **CMD_WRITE_DATA_0** command with a 2-byte message containing the angular momentum with MSB first/last (TBD).
rotateReactionWheel	rotates reaction wheel by a user-defined number of turns. Angle is a signed 16-bits integer; unit is 0.1kgcm2rad Uses **Write Module Data** use case of 1B45 pacvkage, with the **CMD_WRITE_DATA_1** command with a 2-byte message containing the speed (in rpm) at which the reaction wheel has to be rotated, with MSB first/last (TBD).
onflightCalibration	Allows to change value of wheel inertia, offset and sensitivity of gyroscope, using three signed 16-bits integers. Units are: TBD, TBD, TBD. Uses **Set Module Configuration** use case of 1B45 package, with: • gyroscope spingain in field gysoscope_gain_s, • gyroscope spin offsetin field gysoscope_offset_s,

	• gyroscope temperature gain in field gysoscope_gain_T, • gyroscope temperature offsetin field gysoscope_offset_T, • Moment of Inertia of the reaction wheel in Inertia_motor.
Set Module Configuration	Changes some TBD **Designer**-defined configuration parameter (if any). The **Designer** shall define all configurable parameters (if any) in the type **t_Configuration**. This use case is different from the **Configure Module** use case, as the latter allows a compile-time configuration, while this use case allows run-time configuration changes (if foreseen). This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.

4. SYSTEM DESIGN

Sequence Diagram

MOTORDRIVER - ROTATE

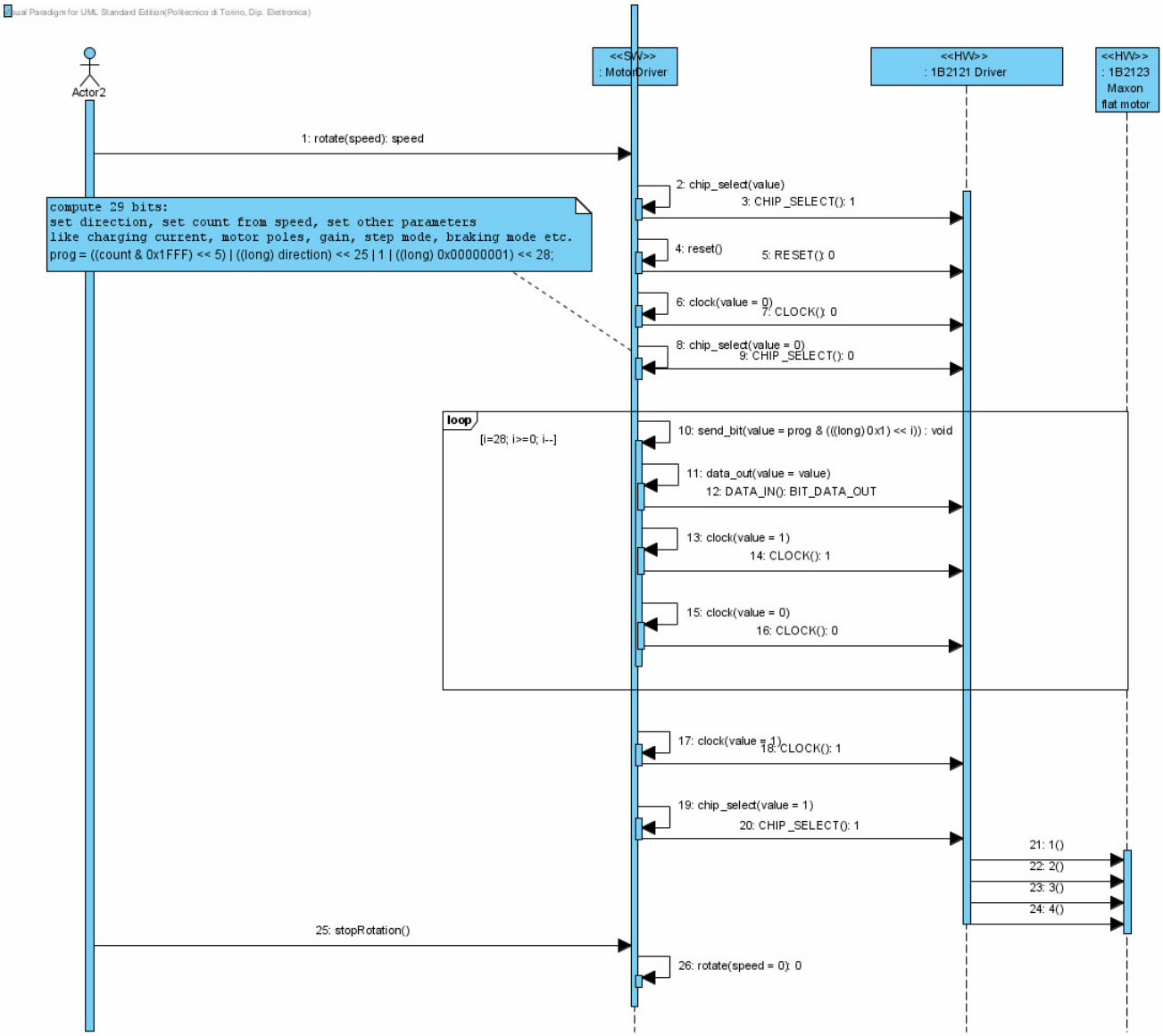


Figure 15: Timing Diagram: Rotate

5. IMPLEMENTATION

1B21 Class Diagram :

The following is the class diagram developed for implementing the system. The details for each specific class can be found in the next section.

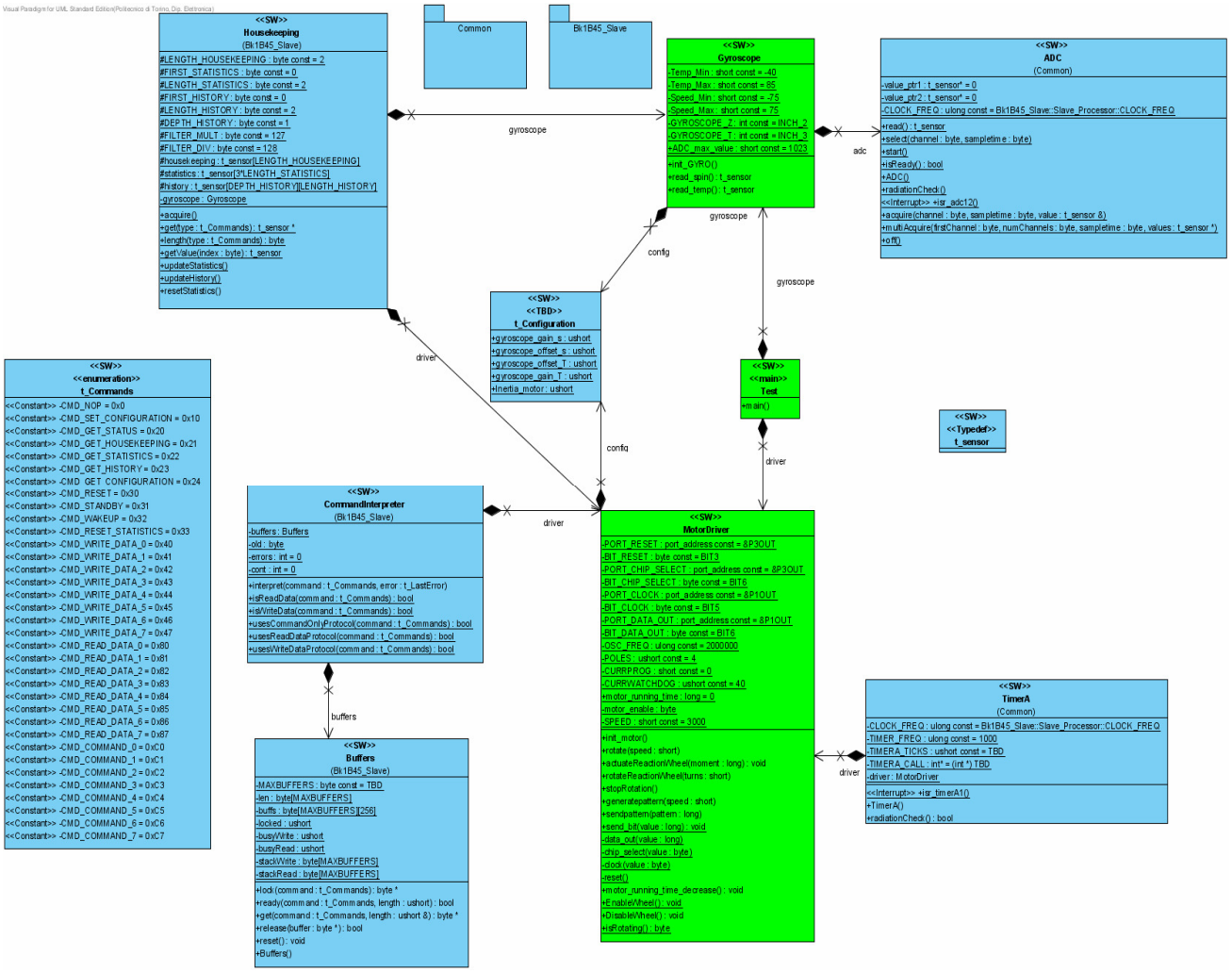


Figure 16: Class Diagram

5.1 Description of Classes and Packages

The description of various classes and packages used in the implementation of the system is as follows:

Name	Documentation
Common	
MotorDriver	Controls the motor by setting the desired speed to the motor. It generates and sends necessary signals to the motor driver 8904 which in turns control the motor.
Bk1B45_Slave	This package contains all specifications of the **Peripheral** (slave) side of the Bk1B45 Subsystem Serial Data Bus. In these diagrams, the **System** is the Peripheral side. It comprises two Use Case diagrams: • **Housekeeping Use Case Diagram**, which incorporates all use cases related to reading from the slave its housekeeping data and the related history and statistics, system status; to issue commands to the system like wake-up, standby and reset, and to read/write application-specific (**Designer**-defined) data. • **Supply, Enable, Configuration Use Case Diagram**, which incorporates use cases related to power supply, static configuration and testing of the system. The document only describes commonly-used functions of a **Peripheral** (slave), and the Designer may add as many functions as he requires. Yet any added function should comply as much as possible to the basic protocol described herein. Furthermore,not all functions described in these use case diagrams need be implemented. Most use cases are optional. If used, they shall be implemented as specified. If not used, they can be disabled by removing appropriate attributes from the relevant classes, as indicated in the **Configure Module** use case.
Buffers	This class implements a set of fixed-length buffers, to be used for **Write Module Data** and **Read Module Data** use cases, respectively. Up to **MAXBUFFERS** buffers of 256 bytes each can be allocated (provided that the processor has enough memory). Each buffer can be used for either **Write** or **Read** operation, although the class guarantees that at least one buffer is allocated for both **Write** and **Read** use case. In particular, for either **Write** (if **command** =

	CMD_WRITE_DATA_x) or **Read** (for **command** = CMD_READ_DATA_x): • throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) • finds the first available buffer of that type • locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is queued... • returns a pointer to it Returns **null** if no buffer of the chosen type is available or command is not supported.
t_Commands	Lists all available command codes and the corresponding value. There is one command for each use case, Removing a command from this class removes the corresponding use case. Adding additional commands requires adding the appropriate code to properly interpret and execute the command.
Test	for testing the motordriver class. It sends various commands to the motordriver class.
t_sensor	The type to be used to store sensor data
CommandInterpreter	Interprets various commands.
Housekeeping	Does the housekeeping related activities, constantly monitors housekeeping parameters.
Gyroscope	Measures the spin and the temperature.
ADC	Converts analog signals into Digital signals capable of being read/write by processor.
t_Configuration	Defines various on-flight configurable parameters.
TimerA	Timer for measuring and calculating time.

5.2 Details of major classes

Some of the classes like MotorDriver have been newly developed, some other classes like Housekeeping, CommandInterpreter etc. have been derived from 1B45 project and redefined or extended, while few classes like Timer, ADC & Buffer have been simply used from 1B45 project.

The classes which have been newly developed are:

1) MotorDriver
2) Gyroscope

5.2.1 MotorDriver

Controls the motor by setting the desired speed to the motor. It generates and sends necessary signals to the motor driver 8904 which in turns control the motor.

MotorDriver Class consists of :

1) Attributes
2) Operations

MotorDriver ->Attributes

private PORT_RESET : port_address = &P3OUT	
Documentation	Selects the **port** on which the **reset** signal will be sent to the motor driver 8904.
private BIT_RESET : byte = BIT3	
Documentation	Select the pin for the **BIT_RESET** function on the **PORT** specified.
private PORT_CHIP_SELECT : port_address = &P3OUT	
Documentation	Selects the **port** on which the **chip select** signal will be sent to the motor driver 8904.
private BIT_CHIP_SELECT : byte = BIT6	
Documentation	Select the pin for the **CHIP_SELECT** function on the **PORT** specified.
private PORT_CLOCK : port_address = &P1OUT	
Documentation	Selects the **port** on which the **clock** signal will be sent to the motor driver 8904.
private BIT_CLOCK : byte = BIT5	
Documentation	Select the pin for the **CLOCK** function on the **PORT** specified.
private PORT_DATA_OUT : port_address = &P1OUT	
Documentation	Selects the **port** on which the **data out** signal will be sent to the motor driver 8904.
private BIT_DATA_OUT : byte = BIT6	
Documentation	Select the pin for the DATA_OUT function on the **PORT** specified.

private OSC_FREQ : ulong = 2000000	
Documentation	Frequency of Oscillator used in Hz.
private POLES : ushort = 4	
Documentation	Number of poles of motor. Shall be either 4, 8, 12 or 16.
private CURRPROG : short = 0	
Documentation	Current limitation and transconductance gain: 0:1.2A; 500mA/V 1:0.6A;250mA/V 2:1A; 500mA/V 3:0.5A;250mA/V 4:0.6A; 500mA/V 5:0.3A;250mA/V 6:0.25A; 500mA/V 7:0.125A;250mA/V
private CURRWATCHDOG : ushort = 40	
Documentation	Absolute value of charge current (in uA) for the watchdog timer. Shall be either 10, 20, 30 or 40.
public motor_running_time : long = 0	
Documentation	this variable stores the time remaining in ms for which the motor has to be rotated.
private motor_enable : byte	
Documentation	it decides where the motor is enabled or disabled. **motor_enable==0** means that motor is disabled and any command given to motor will be disregarded. **motor_enable==1**means that motor is enabled and any command given to motor will be executed.
private SPEED : short = 3000	
Documentation	speed in rpm at which of the motor should be rotated by the commands rotateReactionWheel() and actuateReactionWheel(). presently choosen as 3000 More efficient decision of the motor speed according to the running time can be taken at a later stage by the Designer.
public BIT_MOTOR_ROTATING : byte const = BIT4	
Documentation	set the bit in t_Status:statusWord which corresponds to the state of the motor(rotating/stop).

MotorDriver ->Operations

public init_motor()	

Code	*(PORT_CHIP_SELECT + (&P1DIR-&P1OUT)) |= BIT_CHIP_SELECT;
	
	*(PORT_CLOCK + (&P1DIR-&P1OUT)) |= BIT_CLOCK;
	
	*(PORT_DATA_OUT + (&P1DIR-&P1OUT)) |= BIT_DATA_OUT;
	
	*(PORT_RESET + (&P1DIR-&P1OUT)) |= BIT_RESET;
	clock(0);
	chip_select(1);
	reset();
	EnableWheel();
	motor_running_time=0;
	status.statusWord = status.statusWord & (~BIT_MOTOR_ROTATING);
	//set the MOTOR_ROTATING status to stop
	config.Inertia_motor = 413.9;

Documentation	Initializes the signals necessary to run the motor.

public rotate(speed : short)		
Parameter	**speed**	
	Multiplicity	Unspecified
	Documentation	the speed in rpm at which the motor should be rotated.
	Type	short
	Direction	in

Code	if(speed==0)
	status.statusWord = status.statusWord & (~BIT_MOTOR_ROTATING);
	else
	status.statusWord = status.statusWord | BIT_MOTOR_ROTATING;
	
	generatepattern(speed);	

Documentation	Drives the motor to rotate clockwise (if **speed** is positive) or counterclockwise (if**speed** is negative) at angular speed equal to **speed** rpm. The routine terminates immediately, while the motor keeps on rotating until next command.	

public actuateReactionWheel(moment : long) : void	
Parameter	**moment**

	Multiplicity	Unspecified
	Documentation	the **moment** which the motor should provide. it is used for calculating the time at which the motor has to be rotated at **SPEED** rpm. motor_running_time =1000* moment / ((config.Inertia_motor + Inertia_Satellite) * SPEED * (2 * 3.14 / 60));
	Type	long
	Direction	in

Code	//computer the time in ms for which motor has to be rotated.
	
	motor_running_time =1000* moment / (config.Inertia_motor * SPEED * (2 * 3.14 / 60));
	
	rotate(SPEED);	

Documentation	turns the satellite by the specified **moment** using the rotation of motor. the motor speed is set to **SPEED** rpm. If doesn't check if the motor is already running or not, which has to be checked by the processor giving the command. The routine terminates immediately, while the motor keeps on rotating for the required time.	

public rotateReactionWheel(turns : short)		
Parameter	**turns**	
	Multiplicity	Unspecified
	Documentation	Number of turns the wheel should rotate.
	Type	short
	Direction	in
Code	motor_running_time = 1000 * turns / (SPEED *60); rotate(SPEED);	
Documentation	Rotates the reaction wheel by a user-defined number of turns specified in the parameter **turn**. The routine terminates immediately, while the motor keeps on rotating for the required time.	

public stopRotation()	
Code	rotate(0);
Documentation	Drives the motor to stop immediately.

public generatepattern(speed : short)		
Parameter	speed	
	Multiplicity	Unspecified
	Documentation	speed is the angular speed in rpm with which the motor should be rotated.
	Type	short
	Direction	in

Code

```
unsigned long prog=0, count=0;
char stop, direction;

//init_motor();

if (speed == 0) {
  stop = 1;
  count =0;
  direction = 0;
 }
else if (speed > 0) {
       count = 60*OSC_FREQ/speed;
       direction = 0;
        stop = 0;
}
else {
       count =  60*OSC_FREQ/ (- speed);
       direction = 1;
       stop = 0;
}

// compute the programming word
prog = (((count/16) & 0x1FFF) << 5) | ((long) direction) << 25 | ((long) stop) <<
2 | (motor_enable & 0x1);
prog |= (CURRPROG & 0x4) << 1 | (CURRPROG & 0x2) << 3 |
((long)CURRPROG & (long)0x1) << 28;

switch (CURRWATCHDOG) {
case 10: break;
case 20: prog |= (long)0x2 << 26; break;
case 30: prog |= (long)0x1 << 26; break;
case 40: prog |= (long)0x3 << 26; break;
```

	}
	
	switch (POLES) {
	case 4: prog |= (long)0x1 << 21; break;
	case 8: break;
	case 12: prog |= (long)0x3 << 21; break;
	case 16: prog |= (long)0x2 << 21; break;
	}
	
	clock(0);
	chip_select(0);
	sendpattern(prog);
	
	chip_select(1);

Documentation	This function generates a 29-bit pattern code and using **sendpattern()** send it to the 8904(motor driver) in order to program it to rotate the motor at the specified **speed.**

public sendpattern(pattern : long)		
Parameter	**pattern**	
	Multiplicity	Unspecified
	Documentation	**pattern** is the 29-bit programming word which is sent to the motor driver(8904) in order to set various parameters of the motor.
	Type	long
	Direction	inout
Code	for(i = 28; i >= 0; i--) send_bit(pattern & (((long) 0x1) << i));	
Documentation	Send the 29 bit programming word **pattern** to 8904 motor driver. This 29-bit word controls various parameters of reaction wheel like: rotation speed, direction, watchdog current, brake, sleep mode, step mode etc.	

public send_bit(value : long)		
Parameter	**value**	
	Multiplicity	Unspecified
	Type	long
	Direction	in

Code	data_out(value);
	clock(1);
	clock(0);	

Documentation	Send a bit to the motor driver's data_in port.	

private data_out(value : long)		
Parameter	**value**	
	Multiplicity	Unspecified
	Documentation	**value**will be checked for zero or non-zero condition.
	Type	long
	Direction	in
Code	*PORT_DATA_OUT =(*PORT_DATA_OUT & ~ BIT_DATA_OUT) | (value ? BIT_DATA_OUT: 0);	
Documentation	Send the data (29-bit programming word) serially to the motor driver (8904).	

private chip_select(value : byte)		
Parameter	**value**	
	Multiplicity	Unspecified
	Documentation	**value**will be checked for zero or non-zero condition.
	Type	byte
	Direction	in
Code	*PORT_CHIP_SELECT = (*PORT_CHIP_SELECT & ~ BIT_CHIP_SELECT) | (value ? BIT_CHIP_SELECT: 0) ;	
Documentation	Controls the chip select signal on the motor driver (8904). Sets chip select to 1 when **value**<>0; else sets chip select to 0.	

private clock(value : byte)		
Parameter	**value**	
	Multiplicity	Unspecified
	Documentation	**value**will be checked for zero or non-zero condition.
	Type	byte
	Direction	in
Code	*PORT_CLOCK = (*PORT_CLOCK & ~ BIT_CLOCK) | (value ? BIT_CLOCK: 0) ;	
Documentation	Send the Clock signal to the motor driver (8904).	

private reset()	
Code	*PORT_RESET = (*PORT_RESET & ~BIT_RESET) | 0; *PORT_RESET = (*PORT_RESET & ~BIT_RESET) | BIT_RESET;

Documentation	Controls the **reset** signal on the motor driver (8904).

public motor_running_time_decrease() : void	

Code	if(motor_running_time>0) {
	  motor_running_time--;
	}
	else if (motor_running_time==0) {
	  stopRotation();
	}

Documentation	Decreases the **motor_running_time** by 1 ms.

public EnableWheel() : void	
Code	motor_enable = 0x1;
Documentation	Enables reaction wheel. Any further action of reaction wheel will be carried on.

public DisableWheel() : void	
Code	motor_enable = 0x0;
Documentation	Disables reaction wheel. Any further action of reaction wheel will be disregarded.

5.2.2 Gyroscope

Controls the motor by setting the desired speed to the motor. It generates and sends necessary signals to the motor driver 8904 which in turns control the motor.

Gyroscope Class consists of :

1) Attributes
2) Operations

Gyroscope ->Attributes

private Temp_Min : short = -40	
Documentation	This is the temperature corresponding to which the ADC10 temperature output is 0 (minimum). This is the minimum temperature in °C which the Gyroscope (ADXRS401) can read.
private Temp_Max : short = 85	
Documentation	This is the temperature corresponding to which the ADC10 temperature output is 1023/4095(maximum). The maximum temperature in °C which the Gyroscope (ADXRS401) can read.
private Speed_Min : short = -75	
Documentation	This is the spin corresponding to which the ADC10 spin output is 0 (minimum). This is the minimum spin in °/s which the Gyroscope (ADXRS401) can read.
private Speed_Max : short = 75	
Documentation	This is the spin corresponding to which the ADC10 spin output is 1023/4095 (maximum). This is the maximumspin in °/s which the Gyroscope (ADXRS401) can read.
private GYROSCOPE_Z : int = INCH_2	
Documentation	the ADC channel on which gyroscope spin output is connected.

private GYROSCOPE_T : int = INCH_3	
Documentation	the ADC channel on which gyroscope temperature output is connected.
public ADC_max_value : short = 1023	
Documentation	Maximum output of ADC. 1023 for 10-bit ADC. 4095 for 12-bit ADC.

Gyroscope ->Operations

public init_GYRO()	

Code	config.gyroscope_gain_s = (unsigned short) (1000.0 * (Speed_Max - Speed_Min) / ADC_max_value) ;
	config.gyroscope_offset_s = (unsigned short) (1000.0 * (0 - Speed_Min) / config.gyroscope_gain_s) ;
	
	config.gyroscope_gain_T = (unsigned short) (1000.0 * (Temp_Max - Temp_Min) / ADC_max_value) ; // ADC_max_value = 1023 for 10-bit ADC
	config.gyroscope_offset_T = (unsigned short) (1000.0 * (0 - Temp_Min) / config.gyroscope_gain_T) ; //gyroscope_offset_T is 327 which is the ADC o/p for 0°C
	
	//adc.ADC();

Documentation	It initializes*gyroscope_gain_s*,*gyroscope_gain_T*,*gyroscope_offset_s* and *gyroscope_offset_T* to the required value. These values should be adjusted with correct ones obtained by calibration process.

public read_spin() : t_sensor	

Code	t_sensor val;
	t_sensor spin;
	
	adc.select(GYROSCOPE_Z,4);
	adc.start();
	while (adc.isReady()==false) ;
	val=adc.read();
	
	spin = (t_sensor)((val - config.gyroscope_offset_s) * (long) config.gyroscope_gain_s) ;
	return spin;

Documentation	It measures and returns the rotational speed around z axis with a signed 16-bits integer; unit is 0.0001 rad/s. This operation uses the ADC(from Common) that returns a digital value. This value is software adjusted, offsetted by *gyroscope_offset_s and then* multiplied by *gyroscope_gain_s* and to obtain correct value for spin measure.

public read_temp() : t_sensor	

Code	t_sensor val;
	t_sensor temperature;
	
	adc.select(GYROSCOPE_Z,4);
	adc.start();
	while (adc.isReady()==false) ;
	val=adc.read();
	
	temperature = (t_sensor)(((val - config.gyroscope_offset_T) * (long) config.gyroscope_gain_T) / 10) ;
	return temperature;

Documentation	It measures and returns the temperature with a signed 16-bits integer; unit is 0.01°C. This operation calls **ADC** that returns a value between **0** and **ADC_max_value**(1023 or 4095). This value is software adjusted, divided by *gyroscope_gain_T* after being offset by *gyroscope_offset*_T which is the ADC output for temperature = 0°C.

CONCLUSION

All the modules for ‘Reaction Wheel Actuator’ and ‘Gyroscopic Sensor’ have been successfully coded using UML in C++. The verification has been done by downloading the code onto the circuit with the microcontroller TI MSP430F2274. All the modules necessary for the communication with the central processor have been written and tested. The modelling for the system has been done in UML through Visual Paradigm and the compilation, simulation and assembly code generation along with downloading onto the chip has been done by IAR Embedded Workbench for MSP430.

Personal Comments on Internship and Stay in Torino

I was fortunate enough to get this offer to come to Politecnico di Torino, Italy for a summer internship. This internship has increased my technical, interpersonal and communication skills significantly through the enriching exposure of the technology, culture and society. The technology and infrastructure present here is as expected 'great'. Working on the development of a satellite is quite interesting. The thought that 'you are among the team which is developing a satellite, which is going to be launched soon' is itself great and motivating.

I learnt many new software, tools , techniques and analysed various circuits. Further I developed modules and programmed the microcontroller to control the spin of the satellite. I gained a much better understanding of the satellites and how they are developed. The meetings which I attended(even though quite a few) gave me an idea of professional discussions and how multi-disciplinary projects are coordinated and handled. All these things will surely help me in my future.

My stay at Torino, Italy has been very comfortable and pleasant. I have been living at Borsellino Residence near Politecnico. The residence is quite nice providing all the necessary facilities including well furnished room, Internet, Game Room, TV Room, 24 hrs Reception, Laundry etc.

Also the transport system of the city is nice and convenient to use. The railway station Torino Porta Nuova have trains to all major places in Italy and in Europe. Due to it, very few of my weekends have been in Torino, mostly I went for some visit to some good place on the weekends. I liked this country a lot(as I did a lot of travelling) and even the people are quite friendly.

I will surely like to come here again in future if I get a chance (for Masters or another internship).

Actually the way I got this internship was little amazing and surprising.
I was in my final year of my graduation B.E. (Hons) Electrical & Electronics from BITS Pilani, India. I applied to the professor long back for a 6-months internship from January-June 2009, but unfortunately didn't get through. No regrets, I did the internship in a company in New Delhi, India near my home.

Fortunately I have also got a Job in a reputed multinational company in Bangalore (CISCO Systems Inc., Bangalore, www.cisco.com). I was just waiting for the starting of the job and came to know that it will be probably from 1st July, 2009 while my current internship would be ending by 14th June, 2009.

I was just waiting for the internship to end and job to begin. Suddenly one morning I checked my mail and was surprised to see an email from the Professor Leonardo which said that he has got extra funding and wants to call me for an internship and asked if I am still available?? ☺

The first thought that came to my mind is… 'ohhh no, how is this possible'. Alas, I can't go, because my job joining is just after the current internship and I don't have any time in between.☹ I tried to postpone my joining, but that wasn't an option I had. Still I told the professor that I am interested and asked for the details. But unluckily didn't get any reply for many days. I simply interpreted that I can't go for this and should better forget it.

But suddenly few days after I came to know(unofficially) that the job joining is getting postponed (because of recession, or company policy or whatever) and simultaneously got a confirmed reply from

the professor about the internship. There I was: everything getting rearranged automatically and paving way for me to go Italy. Few more formalities and discussion over the dates of internship and 'All set'.

Finishing previous internship in Delhi on 13th June, fly to Italy on 14th June, back to home from Italy on 13th August and fly to Bangalore for job on 16th August , everything well, internship fits exactly in the time which otherwise would have been vacations.
A trip to Italy (and even to Switzerland, Spain, Belgium, France….) and that too officially , along with a nice experience with the latest technology in the field and chance to work on the Satellites, what else can be better than this??….