POLITECNICO DI TORINO

Master degree
in Electronic Engineering

Master Degree Thesis

Design of multipurpose measurement system for Nanosatellites testing

Supervisor
Prof. Reyneri Leonardo

Candidate
Borrelli Dario

2016/17

Index

SUMMARY

This thesis project focused on the design of a measurement system able to test attitude control of the academic AraMiS satellite. AraMiS, acronym for Modular Architecture for Satellites, is a project started in 2006 at the Politecnico di Torino. The aim of the project is to realize small satellites with a really modular structure. Modularity allows a significant decrease in the cost of the project itself and thus provides the university with the opportunity to become interested in space. The cost of a space mission is, in fact, the main obstacle facing the common interest in space, by companies and universities. The idea behind this project is to develop dedicated interconnected and distributed units, built with commercial off-the-shelf components (COTS) components, in order to increase fault tolerance and allow a decent performance degradation, while maintaining the costs at acceptable levels. Small artificial satellites are generally subdivided according to their weight. In particular, it is about micro-satellite, when the mass of the satellite is between 10 and 100 Kg, it is nano-satellite, when the mass is between 1 and 10 Kg, it is called pico-satellite, when the mass is between 0.1-1 Kg. The most efficient way to reduce the cost of a small satellite project is to reduce project costs as much as possible and not recurring to manufacturing. These costs, in fact, represent over 90% of the total budget. Their reduction can only be achieved through the sharing of design between a large number of space missions. The re-use of projects is the logic behind the AraMiS project, the development of a modular architecture consisting of a small number of flexible modules that can be reused in different missions. Reusing the same module over several times allows you to subdivide project, qualification, and testing costs, and reduce waiting times before launch.

The most critical part of low-cost spacecraft project is the almost complete lack of tests, due for several reason, like complexity, time and cost of test equipment. Hence the idea to realize a low cost multi-purpose measurement system able to model mainly attitude control of nano-satellite, but also other parameters. To provide a wide range of tests on nano-satellite has been chosen to design a measurement system capable to measure position, forces, magnetic field and temperature.

1 INTRODUCTION

1.1 TESTING PROBLEM

One of the major causes for bad result in a nano-satellite space mission is lack or insufficient testing operation. The nano-satellite are designed to be cheap for both realization and sent in the space, but the testing procedure can be very long and the the testing instrument can be very expensive. In nano-satellite space mission often the budget is limited in term of time and in term of costs. For that reason the first operation to be neglect are the testing operation. On nano-satellite can be performed a wide range of tests, the most important are:

- Dynamic test: check the dynamic behaviour of nano-satellite is fundamental for the success of the mission. Using a gimbal the nano-satellite can be rotated or moved, in this condition can be performed center of gravity test, spin-off stabilization test, attitude control test and more;
- Solar test: using lamps, is generated an artificial illumination that approximate the solar sun light, the aim is to test solar cells, sun screen or materials used;
- Thermal vacuum chamber: thermal resistance and dissipation in space is critical, can be simulated using a vacuum chamber and a heat source can be performed thermal simulation, in order to test the thermal resistance of materials or correct heat dissipation;
- Vibration testing: a nano-satellite during launch is subject to an high level of vibration, this condition can be simulated using a shaker. Can be tested the mechanical characteristics of the nano-satellite, like resistance of materials or structural resistance;
- Electromagnetic testing: there are different electromagnetic tests that can be performed, for example using a earth magnetic field simulator can be tested electronics or magnetic parts of nano-satellite like magnetorquer.

1.2 PROPOSED IDEA

The proposed solution intend to solve the problem of nano-satellite tests, creating a multi-purpose low cost measurement system designed specially for nano-satellite. The idea was to create a measurement system has much flexible possible, and able to take different type of measure for a wide range of application for nano-satellite testing. In particular the idea was to realize a multi sensor board and an acquisition board. The board for acquisition (Bk1C601E) have an on-board processor able to control, collect and elaborate data from others board connected through SPI interface. While the multi-sensor board (1C603A) is smaller provided with different sensors for acquire data, memory to store tare value and SPI interface. The choose of the sensors are mainly due to archive the maximum flexibility, in particular using IMU sensor (Inertial Motion Unit) give the opportunity to have three axes gyroscopes, three axes accelerometer, three axes magnetometer, on a single very small integrated circuit. This sensor joint with a temperature sensor permits to perform a wide range of tests for nano-satellite. Furthermore, commercial IMU sensors are becoming widely diffused and relatively low cost due to it's big applications in mobile environments. The accelerometers and gyroscopes permits to test the dynamic behaviour of nano-satellite, in particular the additional magnetometer can be useful to make different type of attitude control tests for magnetorquer, the precision needed to make this type of test is in capabilities of this sensors. The temperature sensor, instead, can be very useful to test element of the nano-satellite like solar panels in a solar simulation device.

1.3 PROPOSED SOLUTION

The proposed solution is a compact measurement system, in particular was necessary to design two different boards. The 1C603A Multi Sensor is very small and contain the IMU sensor and the temperature sensor, can be interfaced through SPI with a FFC cable. The Bk1C601E Acquisition Module contain a microcontroller, sixteen connector to connect to 1C603A Multi Sensor, the capability to acquire data from seven different analog inputs and can be controlled via CAN or via JTAG by an host computer as shown in figure 1.

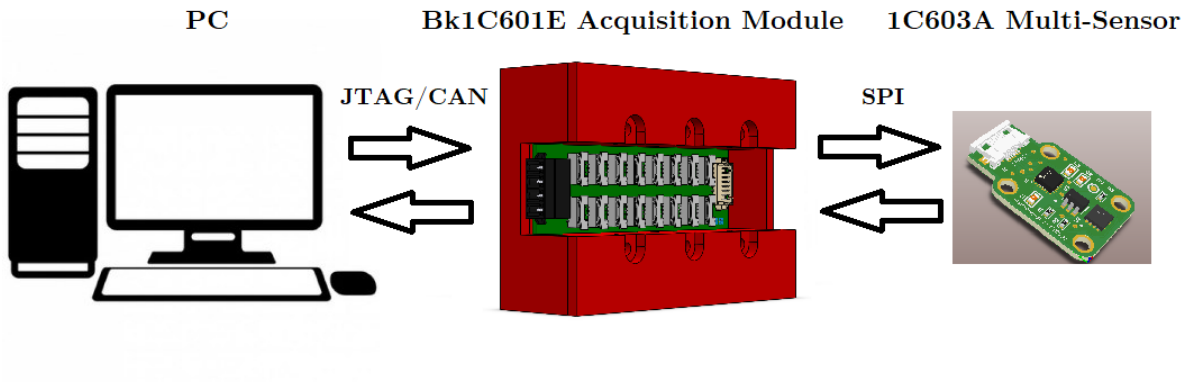


Figure 1: Designed measurement system structure

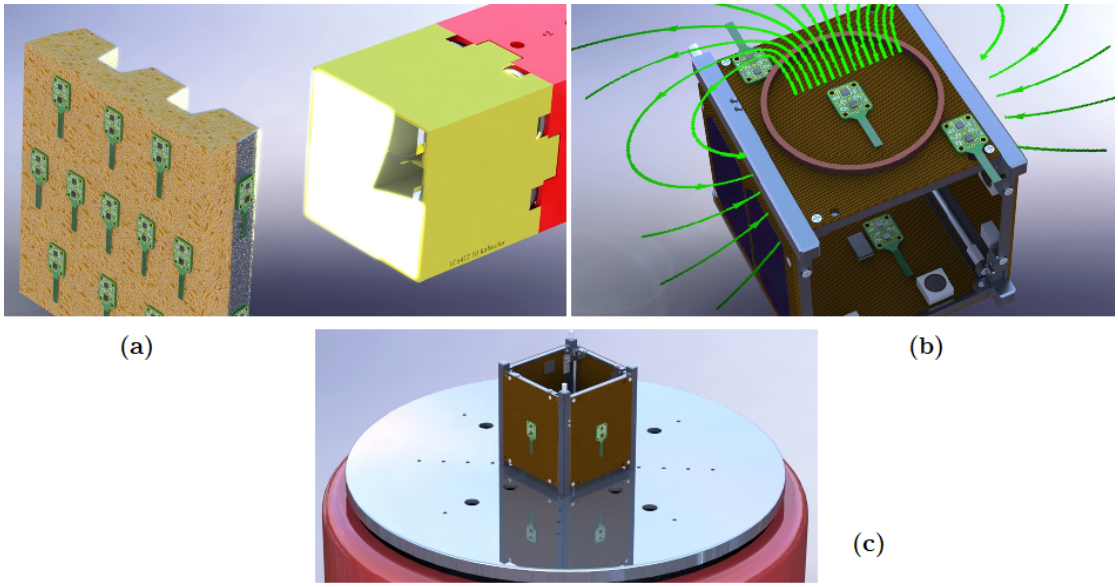


Figure 2: Examples of applications of the designed system

The application of the system are various in figure 2 are shown some examples, like the temperature distribution of objects heated by internal heat sources or caused by real or simulated solar irradiation (a), magnetic field distribution inside or around solenoids or magnetic circuits of systems or satellite attitude actuators such as magnetorquers (b), acceleration distribution on flat surfaces or on the side walls of geometrically complex systems during vibration tests or as a result of shocks (c). The applications are not limited to nano-satellite testing, in fact due to the compacts dimensions and to the versatility of the sensors mounted the system can be used for different application fields like for example automotive applications.

1.4 THESIS ORGANIZATION

This thesis project describe the system realized comprehending also an overview of the standard used, the description of the designed interfaces and testing procedures and future works. To realize the Bk1C601E Acquisition Module has been necessary to implement or optimize also other reusable blocks. In order to respect the AraMiS organization, all the designed blocks and the two boards are complexity reusable for different application. The schematic and PCB has been made using Altium Designer, the final board has been manufactured by EuroCircuit while all the components has been bought on Digikey. After the design and realization phase, has been realized the software using the C++ generator tool of Visual Paradigm and IAR Embedded Workbench for debugging and loading operation. Every part of this thesis project has been created also in classes with Visual Paradigm using the UML descriptive language, in order to better integrate it in the AraMiS project.
The thesis project is organized as:

- Chapter 2 : Contain an overview of the standards used in the project;
- Chapter 3 : Discuss the design of the 1C603A Multi-Sensor;
- Chapter 4 : Describe the optimization of the Bk1B4221WTile Processor;
- Chapter 5 : Discuss the design of the Bk1B4853 CAN Interface;
- Chapter 6 : Describe the design of the Bk1C601E Acquisition Module;
- Chapter 7 : Contain an overview on future works that can be performed with the designed system.

2 STANDARDS USED

To make the developed boards more compatible with different systems has been used known and wide diffused standard for communication, in particular SPI to connect Bk1C601E with 1C603A, I2C to connect the internal sensor of 1C603A, CAN and JTAG interface from an external host with the Bk1C601E. Furthermore, as a part of the AraMiS project has been necessary to respect all the standards for blocks names, interfaces pin-outs and for wires names in schematics.

2.1 SERIAL PERIPHERAL INTERFACE – SPI

The Serial Peripheral Interface is a communication system used between microcontroller with other integrated circuit or between microcontroller. This standard has been created and developed by Motorola. The transmission is between a device called master, and one or more device called slave. The master control the bus, provide the clock signal and can start or end the communication. The clock can be configured setting two parameters:

- CPOL: sets the clock polarity, if equal to zero the idle state of the clock is the low logic level, inversely if equal to one the idle state of the clock is the high logic level;
- CPHA: sets the clock phase, hence the receiver sampling clock edge. If is equal to zero the receiver sample the inputs on the clock edge between half clock cycle with the clock idle and half clock cycle with the clock asserted, inversely if is equal to one the receiver sample the inputs on the clock edge between half clock cycle with the clock asserted and half clock cycle with the clock idle.

The SPI bus is defined serial, synchronous and full duplex. There is not a minimum speed of communication but there is a maximum value due to the parasitic capacitance introduced on the line by connected devices. The bus is realized using 4 wires (5 considering ground) that are:

- SCLK: Serial Clock, provided by the master;
- MISO/SOMI: Master Input Slave Output, input for the master output for the

slave;

- MOSI/SIMO: Master Output Slave Input, output for the master input for the slave;
- CS/SS/nCS/nSS: Chip Select (Slave Select), active low signal provided by the master, cannot be share by more slave and activate the communication with that specific slave.

In figure 3 is shown a basic SPI interface.

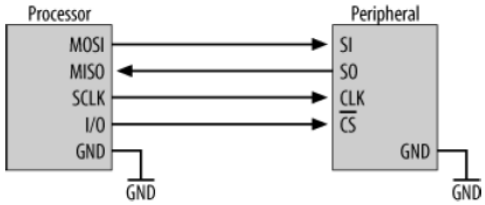


Figure 3: Basic SPI Interface

Both master and slaves contain a microcontroller that control serial shift registers. The master load a byte in it's shift register, and send it to the slave through the MOSI signal line, simultaneously the slave transfer the contents of its shift register to the master through the MISO signal line. In this way both write and read operation are performed simultaneously.

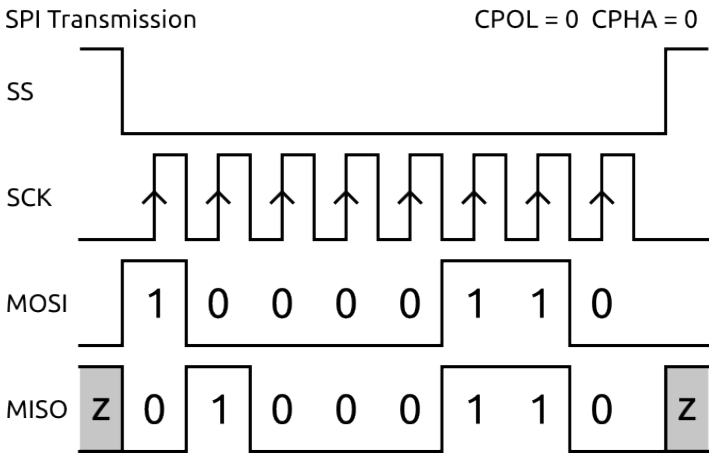


Figure 4: SPI Transmission

In Figure 4 is shown a SPI transmission. If is desired only a write operation, the master ignores the byte received, conversely if is desired a simple read operation the

master transfer a dummy byte to initiate a slave transmission. Some device can handle multiple byte transfers, in this case the chip select remain low for the entire duration of the transmission.

2.2 INTER INTEGRATED CIRCUIT – I2C

I2C (Inter-Integrated Circuit) is a very simple and cheap bus widely used for many years by manufacturer to interconnect peripheral devices mainly in small scale embedded systems. I2C bus is multi-master (each device have a unique address), bidirectional, low speed (100Kbps for standard mode and 400Kbps for fast mode) and synchronous to a common clock. In figure 5 is shown a generic I2C Bus, in the designed system has been used a I2C bus with only one master.

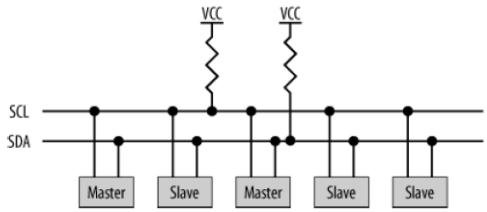


Figure 5: Generic I2C Bus

It use two wires for the bus, both open drain (so they need pull-up resistors) and bidirectional, the wires are:

- SDA: Serial Data
- SCL: Serial Clock

In idle state both wires are high, a transaction begin with the "start condition", that correspond to SDA going low followed by SCL, after that begin the transaction on SDA as shown in figure 6. The data on SDA is sampled on the rising edge of SCL and must remain valid till SCL goes low again. The transaction ends with "stop condition" obtained returning high first SCL and after SDA.

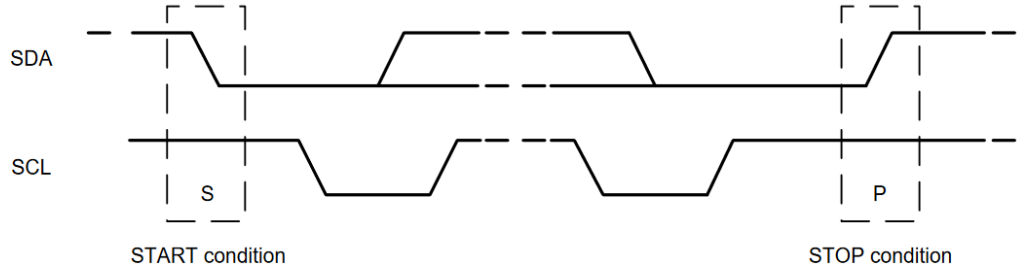


Figure 6: I2C start and stop condition

To reduce errors there is an acknowledge procedure. Every byte sent the sender provide an additional clock pulse on SCL, in which SDA is left high, in order to acknowledge the transmission. If the receiver acquire correctly the byte it low the SDA wire acknowledging the sender, if not the sender understand the error resend the byte. For each transmission the first byte represent the address of the slave, that is composed by seven bits plus one which indicate the direction of the operation (one for read or zero for write), while the second byte represent the data to be sent. In figure 7 is shown an I2C complete data transfer. If two master begin a transmission simultaneously the first master that recognize that the bus is pulled down by another master abort the transmission.

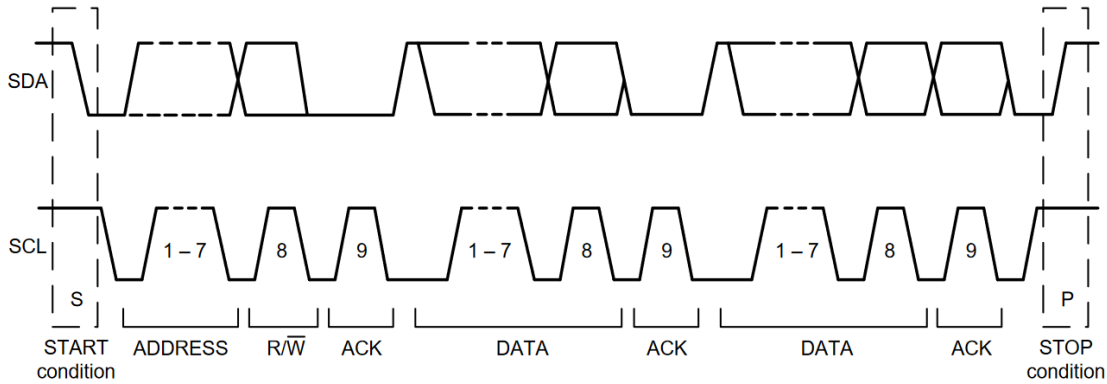


Figure 7: I2C Data transfer

2.3 JTAG

JTAG stand for Joint Test Action Group is defined under the IEEE standard 1149.1a (4), sometimes is known as a Test Access Port or TAP, this interface provides direct access to the internals of the processor, in particular allows real-time debugging of hardware and software, with single-step or multi-step code running directly on the target systems, can be set breakpoints in code or when a particular address or device is accessed, it also permits to examine and modify registers and memory locations. To utilize the JTAG interface is needed additional JTAG compliant hardware. The JTAG interface is defined on four signals:

- TDI : Test Data Input
- TDO : Test Data Output
- TMS : Test Mode Select
- TCK : Test Clock

2.4 ARAMIS STANDARDS - MODULE

The developed system is a part of the AraMiS project, so to make it fully compatible and fully reusable for other system must respect some AraMiS standards. The first standard used is the nomenclatures as can be seen each reusable block and each board have it's own name defined following the AraMiS nomenclature. All the reusable block are built considering that they have interface with a tile processor, in particular has been used the ***module*** structure to handle the signal, defined in the document (2). In figure 8 can be seen the module definition, each wire has a standard name that respect it's functionalities in the tile processor, for example the SPI wires correspond to D0 D1 D2 D3, while D6 and D7 can be used as analog inputs, furthermore all this wire are also general purpose input output so they can also be configured as logic signals.

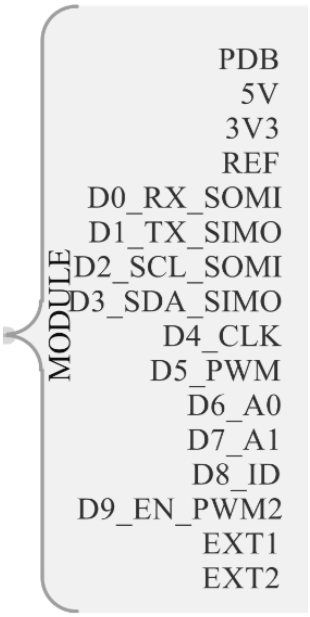


Figure 8: AraMiS module definition

2.5 CONTROLLER AREA NETWORK - CAN BUS

CAN bus stand for Controller Area Network is a serial standard for field bus officially released by Bosch in 1986. The latest CAN specification the 2.0 was released in 1991 (5). For this thesis project has been studied and used the specification 2.0b (3). The CAN network protocol was created to provide deterministic communication in complex systems, with the following features:

- Message priority assignment and fixed maximum latencies;
- Multicast communication with bit-oriented synchronization;
- Bus multi-master access;
- Error detection with automatic retransmission of corrupted messages;
- Detection of permanent failures in node, and automatic switch-off to isolate faulty node;

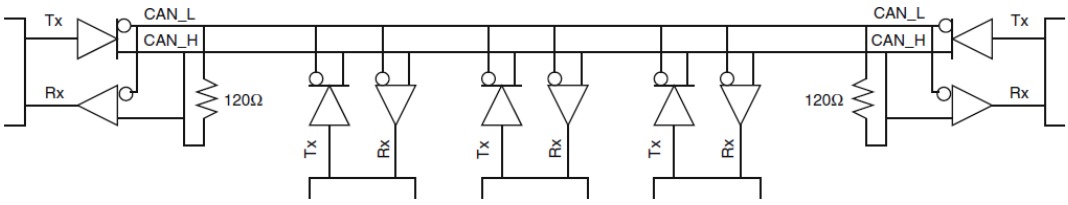


Figure 9: CAN Bus

In a multicast communication every node of the network is listening on the bus, so every node receive the frame and decide if the message have to accepted or not. The CAN protocol consent the simultaneous access to the bus by many nodes, in this case there is an not destructive arbitration based on bit, there is also a priority coded in the node identify transmitted.
In CAN bus definition is specified only the physical layer and the data-link layer, while the application layer is not standardized, so the designer have to define the user interface details.

2.5.1 Physical Layer

The standard ISO 11898 define the bit representation and the transmission medium.
For the bus CAN is used a single bidirectional channel, that usually is differential, the cable used is a twisted pair shielded or not depending of environment noise. The bit-encoding is non return to zero, are used two bits, bit zero called dominant and bit one called recessive. If is transmitted simultaneously a recessive and a dominant bit, the dominant will be the one present on the line. As shown in figure 10 the twisted pair is composed by two wires:

- CAN_L
- CAN_H

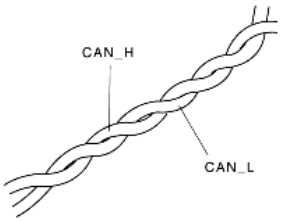


Figure 10: Twisted Pair Cable

The bit zero is represented with both line at 2.5V so the different between them is zero. While the bit one is represented bringing the CAN_H line to 3,5V and the CAN_L line to 1.5V, so the difference between the two line will be 2V. For synchronization is used the Bit Stuffing technique. During the transmission there must be at maximum five consecutive bits can have the same polarity. If there are more than five, a stuff bit of opposite polarity is inserted. The bit time is defined by four time segments. Each segment is composed by an integer number of time quantum that is a minimum of eight to a maximum of twenty five. The four segments are shown in figure 11 and are:

- SYNC_SEG : has fixed duration of one time quanta, and is used to synchronize different nodes;
- PROP_SEG : has programmable duration one to eight time quanta, is used to compensate delay in transmission;
- PHASE_SEG1 and PHASE_SEG2 : between them is located the sampling point, for that reason are programmed in order to have a correct sampling by all the hardware connected, both can vary between one and eight time quanta;

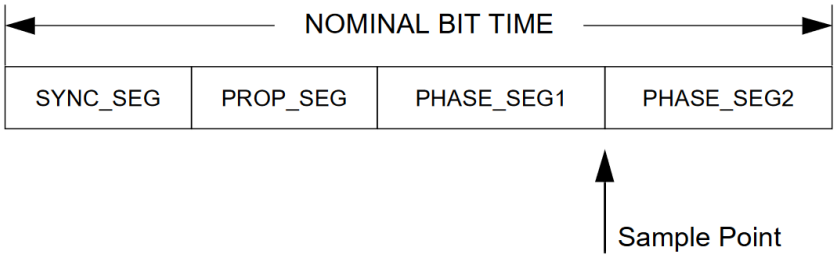


Figure 11: CAN Bit Time

The possibility to change the sampling point permits to optimize the bit timing with respect of the needs. The last specification is on the termination of the bus, that must have an impedance of 120Ω. In particular, in the design of Bk1B4853 CAN Interface has been used as termination two 62Ω with the ground connected between in the middle.

2.5.2 Data Link Layer

In this layer is defined other important aspects of CAN protocol, like data message frame formats, bus arbitration, message reception, and error management. In CAN there are four types of frame that can be transmitted, they differs by structure, content and function:

- Data frames: contain data information;
- Remote frames: are used to request transmission, don't contain data field;
- Error frames: are transmitted when a node detects an error;
- Overload frames: are used for flow control to request additional time delay before the transmission of other frames;

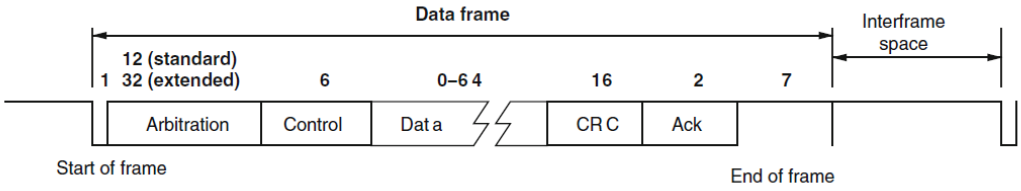


Figure 12: CAN Bus frame format

The frame format is shown in figure 12,it contains:

- Start of frame : is represented by a single dominant bit, is used to indicate the start of transmission and to synchronize all the receiver nodes;
- Arbitration field : can be composed by twelve or sixteen bits according to user defined format;
- Control field : is used to control the correctness of the previous sent bits and contain information on data field, like how many bits is long the data field;
- Data field : contain data, can contain from zero, for remote frames, to eight byte of data;
- CRC field : contain the fifteen bits of cyclic redundancy check of the data and one bit used as a CRC delimiter;
- Acknowledge field : is composed by two bit, one used by the receiver node as acknowledge, the bit is leaved recessive by the transmitter if the message is correctly received the receiver node write a dominant bit on that slot, the

second bit is used as acknowledge delimiter;

- End of frame : is composed by seven recessive bits, is used to divide frames;

The bus arbitration of CAN bus is both priority based and non-preemptive (a message cannot be preempted by higher priority message). The bus act like a wired-AND channel connected to all nodes. If the bus is idle two or more nodes can start the transmission at the same time, one of them issue a start of frame bit (a dominant bit) and other nodes synchronize on that bit edge and then they start transmit on the bus. To avoid the message collision every transmitting nodes compare the bit transmitted and the corresponding bit on the bus, if they coincide the node continue the transmission, instead if the bit transmitted is recessive (bit one) and the bit on the bus is dominant (bit zero), the unit stops the transmission and switches to listening mode only. Clearly the node transmitting the message with lowest identifier is always the winner of the arbitration, furthermore, at any time there can't be two message with the same identifier, but can be two or more nodes to receive the message. For that reason every node have an unit in the controller that using a mask registers permits the node to accept or not the received message.
The error management is performed in different ways, is used bit monitoring, the transmitter check on the line if the bit is correctly sent; bit stuffing, every five equal bits is sent a bit of opposite phase; CRC check, every node compute the CRC value and check if is equal with the one received; control on frame, if the receiver don't recognize the frame format is generated an error frame; acknowledge bits, in order to recognize error in receiving. Every node take a count of how many errors in transmission and in reception it generated, in particular every error has a different weight in the count. If one of this counter increase the node switch from a normal state to an error active state, in which the node can take part on the communications on bus, if increase more it switch to passive error state, in which the node must wait more time to transmit on the bus, if increase more it switch to bus off state, in which the node don't take part on the transmission on bus waiting to 127 recessive bits on bus to take part again on the communications.

2.6 COMPONENT LIBRARY

The first thing to set up to start a project using ***Altium Designer*** is a components library, because each component used in the project must be uniquely defined in a library. A PCB library, where are defined all the footprints, and symbols library, in which are defined all the symbols and all the parameters of the component. The first problem to overcome was the necessity to use the components already bought and present in the Politecnico. In particular was necessary for this board to set up a database library for capacitors, resistances. For the remaining components was necessary to create a custom library, in which all the footprints and symbols was modified to meet the datasheet specification, added component 3d model and revised in order to avoid errors. In particular for the Lead 24 QFN footprints was necessary to add small metal line (because for them the alignment is granted) to help the solder operation as can be seen marked as U1 in figure 15.

3 DESIGN OF 1C603A MULTI-SENSOR

The 1C603A Multi-Sensor is a multiple sensor able to measure the following physical parameters of a system under test:

- Temperature;
- Angular velocity along three orthogonal axes;
- Linear acceleration along three orthogonal axes;
- Magnetic field along three orthogonal axes;

In figure 13 is shown the class diagram of this board, comprehensive of all it's components and documentation. In the following paragraphs is explained the design work flow, starting from specifications, through choose of components, drawn of schematic, software, PCB design, to final product.

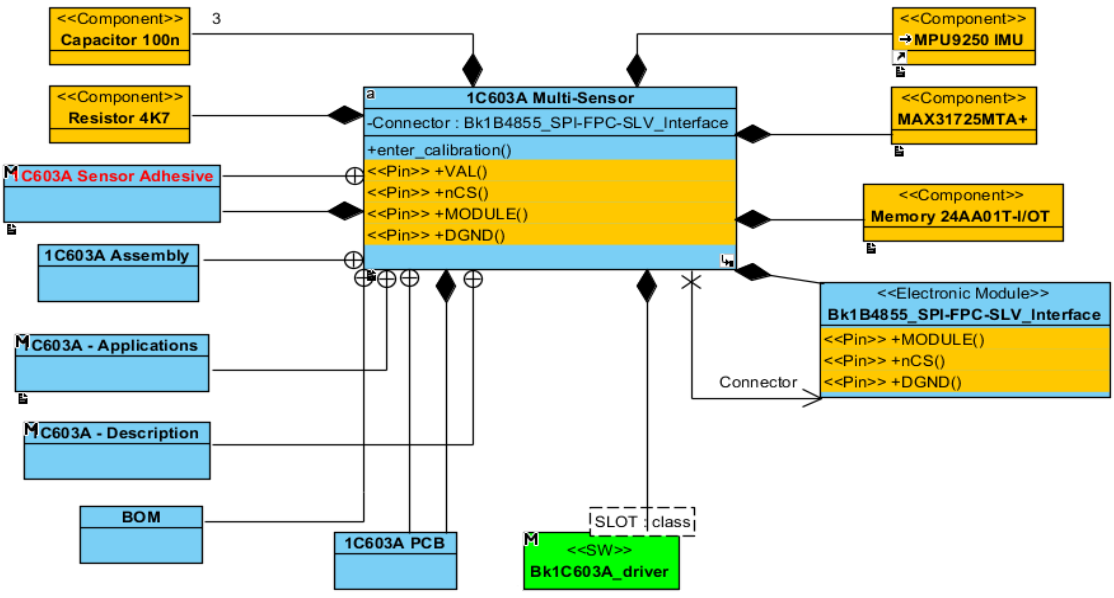


Figure 13: Bk1C603A Class Diagram

3.1 SPECIFICATION

The main specification for the 1C603E Multi-Sensor board are defined functional and the mechanical, trying also to obtain a low cost device, in terms of components and production costs.

3.1.1 Functional Specification

The board designed must have this functionalities:

- three a axis gyroscope, accelerometer and magnetometer;
- temperature sensor;
- controllable by a control board via SPI;
- capability to store on board calibration values, in order to correct the systematic error with the stored tare line.

3.1.2 Mechanical Specification

The board designed must be very small to obtain the maximum flexibility for make the board usable for also small application, and to permit to use more board on the same nano-satellite. For that reason the maximum dimensions set are 3x3cm. The final board dimensions are 1.4cm and 2.3cm. In order to be easily fixed must be present hole for fixing and all the bottom part of the board must be isolated to avoid shorts between different lines. The cable used to interface must be smallest possible and flexible but at the same time provide a stable connection with the board.

3.2 COMPONENTS

The components needed are an IMU sensor (three axis gyroscopes, accelerometer, magnetometer), a temperature sensor, a memory and a connector. To encounter the specification defined in paragraph 3.1 all the packages chosen are the smallest possible, the IMU sensor must have the capability to control other module (via I2C for the MPU9250) and at the same time must have an SPI interface to communicate

with the host processor, the memory must be capable to store all the tare value for each sensor, the connector must provide a stable connection to the host processor.

3.2.1 IMU sensor - MPU 9250

The IMU Sensor chosen is the Microchip MPU 9250, has been chosen for the high connectivity (this sensor have SPI and I2C interface and can control any I2c-compatible device), very small package 24 Lead QFN and relative low price (9-4€ each). From datasheet (6), it's a multi-chip module composed by two dies integrated into a single QFN package. In one die is present the 3 axis gyroscope and for the 3 axis accelerometer, while in the other is present the 3 axis magnetometer. The sensors are sampled with one16-bit ADC for each axes of gyroscope, accelerometer and magnetometer. For precision tracking of both fast and slow motions, the gyroscope full-scale range is user-programmable and also the accelerometer full-scale range is user-programmable. Communication with all registers of the device is performed using either I2C at 400kHz or SPI at 1MHz. For applications requiring faster communications, the sensor and interrupt registers may be read using SPI at 20MHz. For processing is present an embedded Digital Motion Processor (DMP) MPU-9250 that can be used to offloads computation of motion processing algorithms from the host processor. The DMP acquires data from accelerometers, gyroscopes, magnetometers and additional 3rdparty sensors, and processes the data. The resulting data can be read from the DMP's registers, or can be buffered in a FIFO. The DMP feature is not used in the 1C603A design. In figure 14 is present the block diagram representation of the module.
The features of triple-axis MEMS gyroscope are:

- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec and integrated 16-bit ADCs;
- Digitally-programmable low-pass filter;
- Gyroscope operating current: 3.2mA;
- Sleep mode current: 8μA;
- Factory calibrated sensitivity scale factor;

- Self-test;

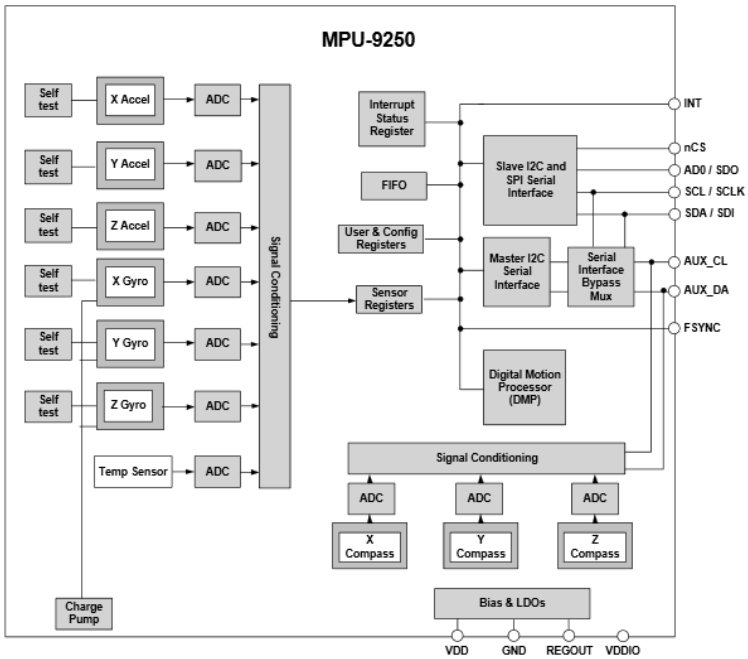


Figure 14: Block Diagram MPU 9250

The features of triple-axis MEMS accelerometer are:

- Digital-output triple-axis accelerometer with a programmable full scale range of ±2g, ±4g, ±8g and ±16gand integrated 16-bit ADCs;
- Accelerometer normal operating current: 450μA;
- Low power accelerometer mode current: 8.4μA at 0.98Hz, 19.8μA at 31.25Hz;
- Sleep mode current: 8μA;
- User-programmable interrupts;
- Wake-on-motion interrupt for low power operation of applications processor;
- Self-test;

The features of triple-axis MEMS magnetometer are:

- 3-axis silicon monolithic Hall-effect magnetic sensor with magnetic concentrator;
- Wide dynamic measurement range and high resolution with lower current consumption;

- Output data resolution of 14 bit (0.6μT/LSB);
- Full scale measurement range is ±4800μT;
- Magnetometer normal operating current: 280μA at 8Hz repetition rate;
- Self-test function with internal magnetic source to confirm magnetic sensor operation on end products;

Other features are:

- Auxiliary master I2C bus for reading data from external sensors;
- 3.5mA operating current when all 9 motion sensing axes and the DMP are enabled;
- VDD supply voltage range of 2.4–3.6V;
- VDDIO reference voltage for auxiliary I2C devices;
- Smallest and thinnest QFN package for portable devices:3x3x1mm;
- Minimal cross-axis sensitivity between the accelerometer, gyroscope and magnetometer axes;
- 512byte FIFO buffer enables the applications processor to read the data in-bursts;
- Digital-output temperature sensor;
- User-programmable digital filters for gyroscope, accelerometer, and temp sensor;
- 10,000g shock tolerant;
- 400kHz Fast Mode I2C for communicating with all registers;
- 1MHz SPI serial interface for communicating with all registers;
- 20MHz SPI serial interface for reading sensor and interrupt registers;
- MEMS structure hermetically sealed and bonded at wafer level;
- RoHS and Green compliant.

3.2.2 Temperature Sensor - MAX31725MTA+

The temperature sensor was chosen to meet the specification of maximum 0,5° C of uncertainty, an I2C interface, a small package (TDFN 8) and low cost (1,2-1,1€ each). From datasheet (7) the MAX31725MTA+ convert the temperature measurements to digital form using a high-resolution, sigma-delta, 16bit ADC. Accuracy is ±0.5°C from -40°C to +105°C. Communication is through an I2C-compatible 2-wire serial interface. The I2C serial interface accepts standard write byte, read byte, send byte, and receive byte commands to read the temperature data and configure the behaviour of the open-drain over-temperature shutdown output. The module features three address select lines with a total of 32 available addresses. The sensors have a 2.5V to 3.7V supply voltage range, low 600µA supply current, and a lock-up protected I2C-compatible interface. The device use an 8-pin TDFN package and operate over the -55°C to +150°C temperature range.

3.2.3 Memory - 24AA01T-I/OT

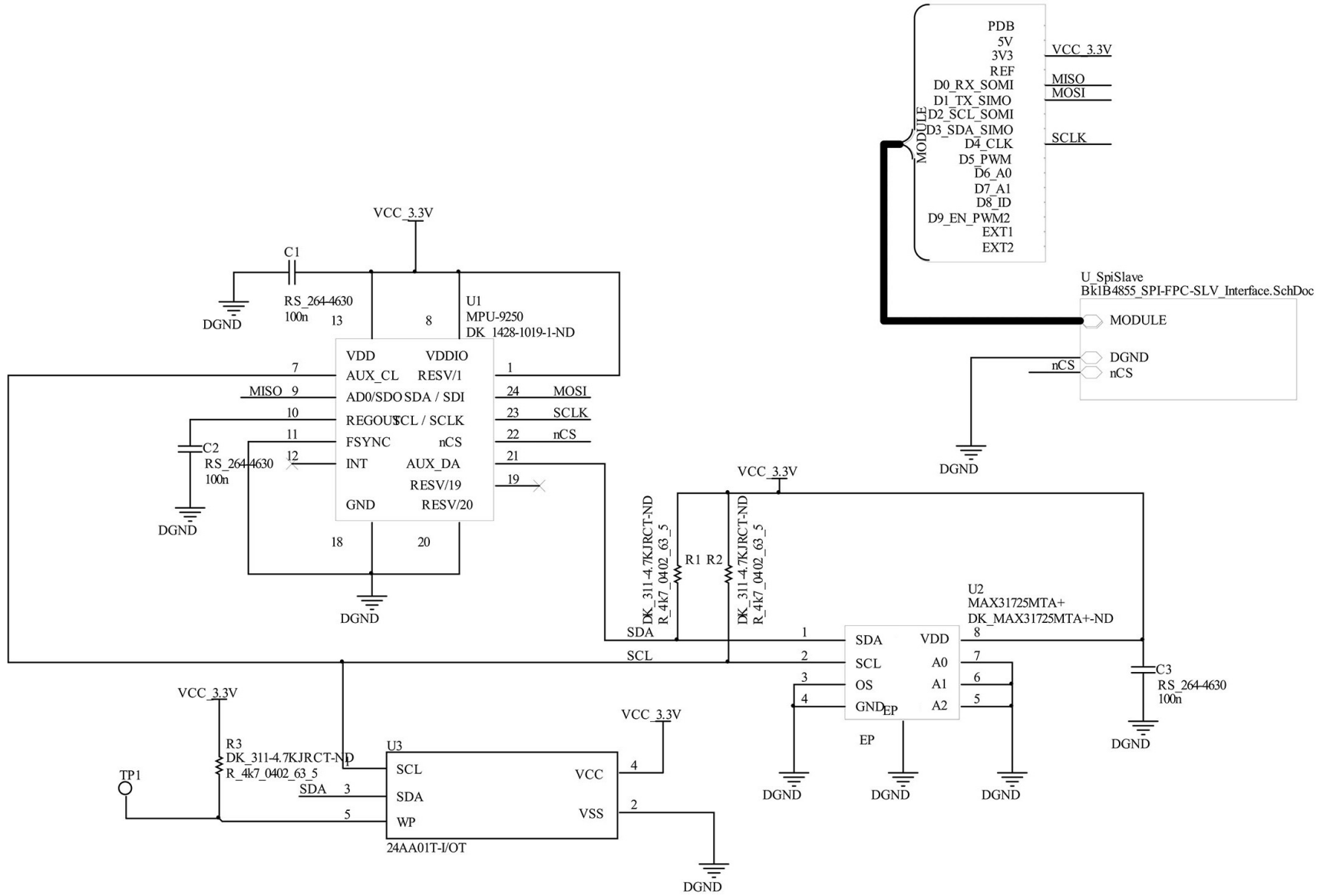
The memory have to be big enough to store all the calibration value for each sensor, for that reason have been chosen a 1Kbit (128 x 8) memory, with I2C interface, with a small package (SOT 23-5) and a low price (0,2€ each), due to it's compatibility is possible to choose for different application a bigger memory. From datasheet (8) the Microchip Technology Inc. 24AA01T-I/OT is a 1 Kbit Electrically Erasable PROM. The device is organized as one block of 128 x 8-bit memory with a 2-wire serial interface. Low-voltage design permits operation down to 1.7V with standby and active currents of only 1µA and 1 mA, respectively. The device also has a page write capability for up to 8 bytes of data. The package chosen is the 5-lead SOT-23.
Features:

- Single Supply with Operation down to 1.7V for 24AA01T-I/OT Devices;
- Low-Power CMOS Technology:
  - Read current 1 mA, max;
  - Standby current 1µA, max;
- 2-Wire Serial Interface, I2C™ Compatible;

- Schmitt Trigger inputs for Noise Suppression Output Slope Control to eliminate Ground Bounce;
- 100 kHz and 400 kHz Compatibility;
- Typical Page Write Time 3 ms;
- Hardware Write-Protect;
- ESD Protection >4,000V;
- More than 1 Million Erase/Write Cycles;
- Data Retention >200 Years;
- Factory Programmable Available;
- Package 5-lead SOT-23;
- Pb-free and RoHS Compliant;
- Temperature Ranges: 40°C to +85°C.

3.3 DESIGN – 1C603A MULTI-SENSOR

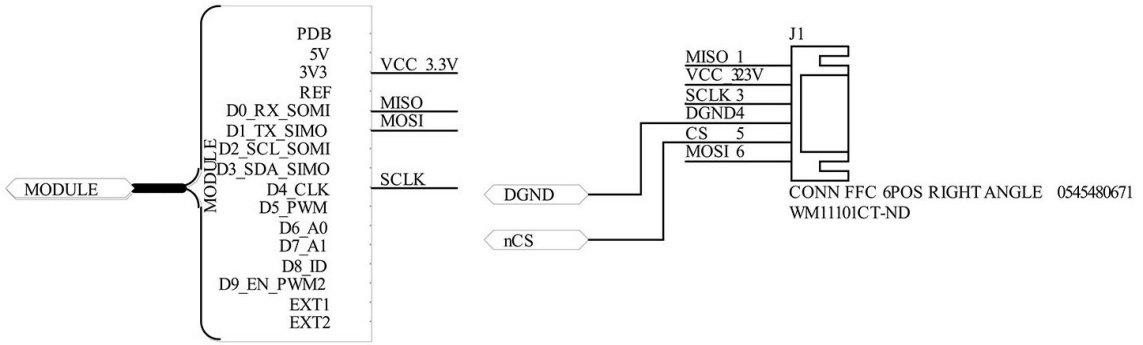
The board is designed to be connected via SPI with the Bk1C601E Acquisition module board. The SPI interface of the MPU9250 is used to control and handle the communication with the temperature sensor and memory via I2C, eliminating the necessity of additional hardware. The memory is normally in only read configuration to avoid the lost of data in case of error addressing, to make it writeable is necessary connect to ground the 1mm test point present on the board. The Max31725MTA+ device have a configurable slave address, connecting the pins A2 A1 A0 to ground has been selected as base address 144 (90 Hex). The design has been divided in two part, a top level in which there are all the main components and a reusable block the Bk1B4855_SPI-FPC-SLV, where is defined the pin out of the connector for the SPI slave interface. The top sheet, that is shown in schematic page 1, is arranged to make it fully compatible and reusable for future application, in particular to define signal has been used the standard ***AraMiS*** module definition described in paragraph 2.4 . The 100nF capacitors C1, C2 and C3 are decoupling capacitors, while the 4.7KΩ resistances R1 and R2 are pull-up resistor for I2C communication between devices. The memory write protection pin is connected to R3 a 4.7KΩ resistor, that acts as a pull-up resistance and to a test point, to make that pin easily accessible by the user.



Title	1C603A_Multi-Sensor_Project.PrjPcb : 1C603A_Multi-Sensor.SchDoc		
Size: A4	Number: 1 / 2		Revision: 1.0
Date: 16/10/2017		Sheet of	1C603A
File: C:\Users\..\1C603A_Multi-Sensor.SchDoc		Drawn By:	Dario Borrelli

3.3.1 Reusable Block – Bk1B4855_SPI-FPC-SLV

This reusable block in which is present the connector of FPC/FFC slave SPI interface. It takes as input a module, a chip select wire and ground, and connect it properly to match the master interface defined in Bk1B4855_SPI-FPC-MST. The pinout is elaborated to reduce cross-talk between different wires, keeping the high switching wires isolated between low switching wires. In page 2 of the schematic is shown the pinout for the FFC slave connector and how it's connect to the module signals.



Title	1C603A Multi-Sensor Project.PrjPcb : Bk1B4855 SPI-FPC-SLV Interface.SchDoc		
Size: A4	Number: 2 / 2		Revision: 1.0
Date: 16/10/2017		Sheet of	1C601A
File: C:\Users\..\Bk1B4855_SPI-FPC-SLV_Interface.SchDoc		Drawn By:	Dario Borrelli

3.4 PCB

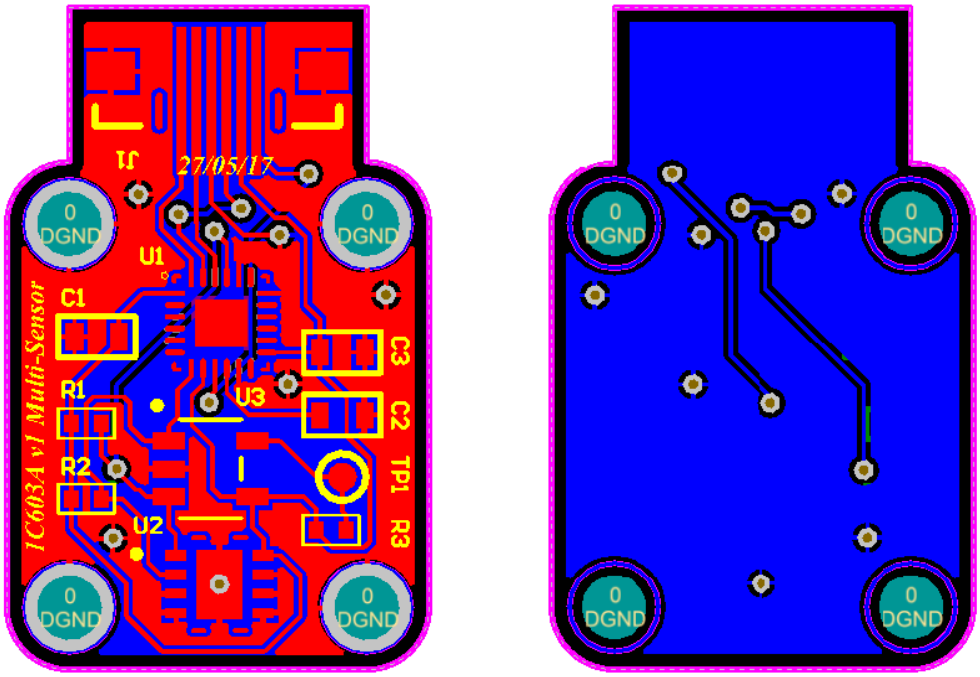


Figure 15: Bk1C603A Multi-Sensor Designed PCB

During the hardware realization has been interfaced different problems. The fist idea was to realize the board using a flexible-rigid PCB so without a female connector with a flexible part that have a male connector compatible with a chosen female connector on the Bk1C601E Acquisition module. The problem related to this idea were the cost of production, not compatible with the low cost specification, there were problems legated to the tolerances in the production of the flexible part of the PCB (in particular thickness and outline spacing) that was not compatible with any possible female connector on the Bk1C601E Acquisition module. For that reason has been chosen a female connector compatible with any 5 pins 3-5mm FFC cable, and the same standard has been used for the Bk1C601E Acquisition module. The PCB is realized considering rules not so strict in order to avoid the increase of the cost for production. The PCB is a two layer FR-4, for the wires thickness and for clearance is used 6mil as rules. Vias dimension are 0,7mm with 0,3mm for the hole.

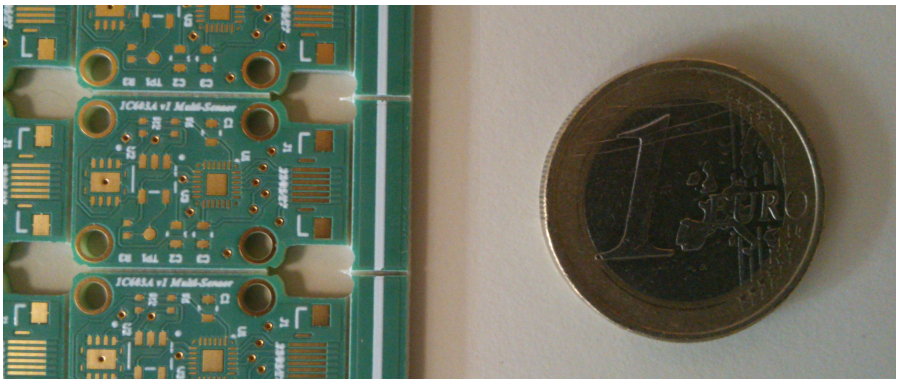


Figure 16: Produced 1C603A Multi Sensor PCB

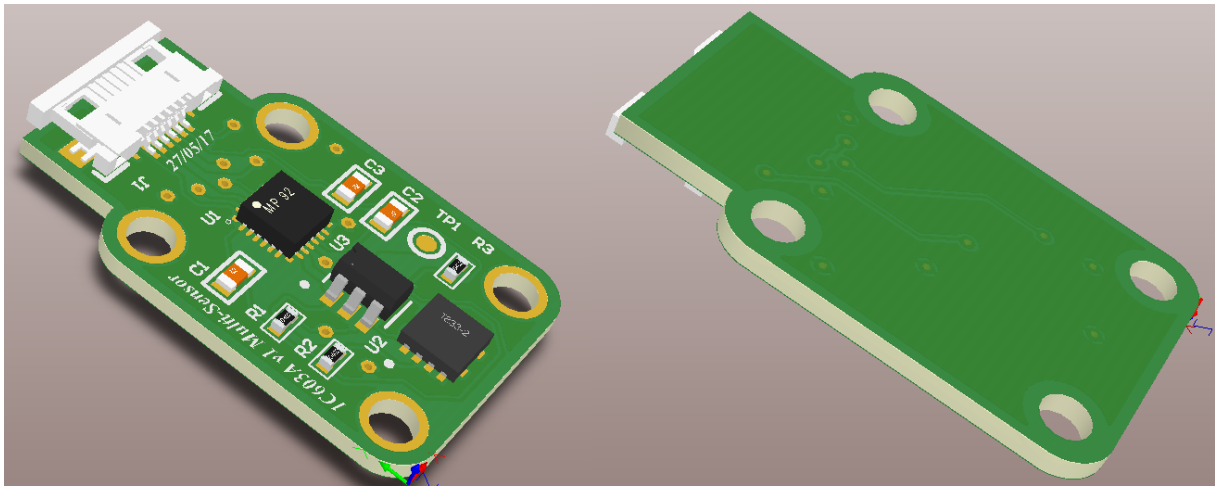


Figure 17: 3d Model of Bk1C603A Multi-Sensor

To isolate the bottom of the PCB all the vias are tented on the bottom. The capacitor C1, C2 and C3 are decoupling capacitor, for that reason must be placed closed to the supply voltage pins of the integrated circuit. There are also 4 mounting holes of 2,2mm connected to the ground. The resulting PCB is shown in figure 15. The blue and red polygon are copper pour connected to ground used to reduce cross-talk effect on the board. In figure 17 is shown the 3d model of the board.

3.4.1 Electrical Characteristics

In the following table are presented the electrical characteristic of the 1C603A Multi-Sensor. More information can be taken from the MPU9250 datasheet (6), in particular this integrated circuit have the capability to activate one sensor at once in order to reduce power consumption.

Parameter	**Min**	**Typical**	**Max**
Supply Voltage	2,4V	2,5V	3,6V
Supply Current (all sensor active)	3,8mA	4,4mA	7,63mA
Temperature Range	-40°C		+85°C
VIHmin	1,68V	1,75V	2,54V
VILmax	0,72V	0,75V	1,08V
VOHmax	2,16V	2,25V	3,24V
VOLmin	0,24V	0,25V	0,36V

3.5 SOFTWARE

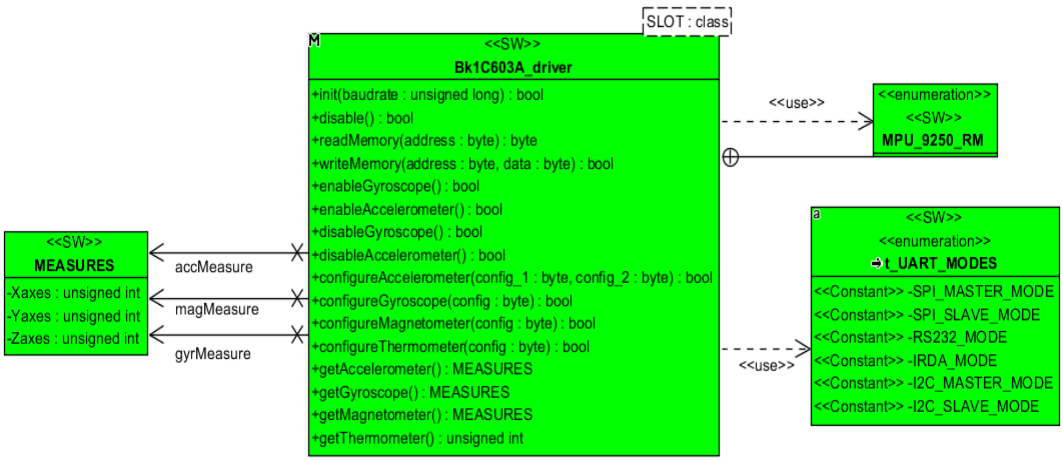


Figure 18: Software to interface with 1C603A

Due to it's structure this board doesn't need a preloaded firmware, but to correctly operate some procedure must be respected. This board must interact with a tile processor for that reason has been realized a driver class that use all the methods of

the tile processor used the Bk1B4221WTile Processor. In this paragraph are described every function created. In figure 18 is shown the software class diagram that contain all the function used to interface with Bk1C603A board, the enumeration called MPU_9250_RM instead represent the complete register map of the MPU9250. All the developed software can be find in Appendix A 1C603A.

3.5.1 Initializing Procedure

bool init(baudrate : unsigned long)

For the initializing procedure has been defined a function called init() that take as input the baudrate used for SPI communication for the tile processor. This function disable all unused features like FSYNC pin, interrupts and setup the I2C communication:

- Reset all register accessing ***PWR_MGMT_1*** at 107 (6Bh) and setting it to 80h, to reset the magnetometer access the register ***CNTL2*** 11 (0Bh) and set it to 01h. After that is mandatory to wait some time till the operation is complete;
- Access the register ***CONFIG*** at 26 (1Ah), configure the bit [5:3] to 000, to disable FSYNC;
- Set register ***INT_PIN_CFG*** at 55 (37 Hex) bit 2 to 0, in order to disable Fsync pin from causing an interrupt;
- Reset register ***INT_ENABLE*** at 56 (38 Hex) to disable all interrupts;
- To setup the MPU 9250 as the I2C master for the memory and temperature sensor is necessary to access ***USER_CTRL*** at 106 (6Ah) and setting it to 20h, enabling the bit 5;
- To set the I2C bus speed, in the register ***I2C_Master_Control*** at 36 (24h) can be selected different speed changing the first 3 bits, to set the speed to 400Khz must write in it 0Dh.
- Set the clock source for gyroscope accessing ***PWR_MGMT_1*** register at 107 (6Bh) and setting it to 01h, in order to select the best clock source;

If every operation is completed correctly, the function returns true, else it returns false.

3.5.2 Disable Function

bool disable()

This function disable the SPI communications with the 1C603A Multi-Sensor, it returns true if the operation is completely successfully otherwise it returns false.

3.5.3 Configuration Functions

The MPU9250 offer the possibility to configure several parameters for each sensor, like full-scale, precision and others. It permits also to activate or deactivate one sensor at time, reducing the power dissipation. For that reason was needed to create a series of function to configure or enable each sensor. These functions must be used before any acquisition operation from sensor, in particular for the temperature sensor it's mandatory to call the configuration before any other operation.

3.5.3.1 Enable Gyroscope

bool enableGyroscope()

This function enable the gyroscope, it returns true if every operation is completed correctly, else it returns false. To do that is necessary to:

- Access the ***PWR_MGMT_2*** register 108 (6Ch), read the register value and mask it with F8h in order to set the bits [0-2] to zero. The masking operation is necessary to keep activated the sensor already active.

3.5.3.2 Enable Accelerometer

bool enableAccelerometer()

This function enable the accelerometer, it returns true if every operation is completed correctly, else it returns false. To do that is necessary to:

- Access the ***PWR_MGMT_2*** register 108 (6Ch), read the register value and mask it with C7h in order to set the bits [3-5] to zero.

3.5.3.3 Disable Gyroscope

bool disableGyroscope()

This function disable the gyroscope, it returns true if every operation is completed correctly, else it returns false. To do that is necessary to:

- Access the ***PWR_MGMT_2*** register 108 (6Ch), read the register value and mask it with 07h in order to set the bits [0-2] to one.

3.5.3.4 Disable Accelerometer

bool disableAccelerometer()

This function disable the accelerometer, it returns true if every operation is completed correctly, else it returns false. To do that is necessary to:

- Access the ***PWR_MGMT_2*** register 108 (6Ch), read the register value and mask it with 38h in order to set the bits [0-2] to one.

3.5.3.5 Configure Accelerometer

bool configureAccelerometer(config_1 : byte, config_2 : byte)

This function takes as inputs two values, that will directly written in the corresponding configuration register:

- config_1 : will be written in the register ***ACCEL_CONFIG*** at 28 (1Ch), in this register can be set full-scale and self test operation for each axes;
- config_2 : will be written in the register ***ACCEL_CONFIG_2*** at 29 (1Eh), in this register can be set a low pass filter for the sensor and it's characteristics.

The function return true if everything goes fine, else it returns false.

3.5.3.6 Configure Gyroscope

bool configureGyroscope(config : byte)

This function takes as input one value, that will directly written in the corresponding configuration register:

- config : will be written in the register ***GYRO_CONFIG*** at 27 (1B), in this register can be set full-scale and self test operation for each axes;

The function return true if everything goes fine, else it returns false.

3.5.3.7 Configure Magnetometer

bool configureMagnetometer(config : byte)

This function takes as input one value, that will directly written in the corresponding configuration register:

- config : will be written in the register ***CNTL*** at 10 (0Ah), in this register can be set operating mode and measurement precision in bits;

The function return true if everything goes fine, else it returns false.

3.5.3.8 Configure Thermometer

bool configureThermometer(config : byte)

This function configure the MPU9250 to communicate through I2C with the temperature sensor, in order to write in it's configuration register the config parameter. The input config (type byte), will directly written in the corresponding configuration register, for that reason the user must configure it properly according to the temperature sensor datasheet (7).

In order to correctly communicate with the temperature sensor, the MPU9250 must be configured properly:

- Access the Slave 0 register ***I2C_SLV0_ADDR*** address 37 (25h), write in it 200 (C8h), to set up a write procedure to the external sensor, in particular the Msb=0 indicate a write procedure;
- In register ***I2C_SLV0_REG*** 38 (26h) must be set the register to write in this case must be set to 01, that correspond to the configuration register, in which can be set data format and operation mode;
- Access the register I2C_SLV0_DO 99 (63h), writing in it the config that will directly write in the configuration register of the sensor.
- Finally to start the I2C communication, in the register I2C_SLV0_CTRL 39 (27h) must write 129 (81h);

The function return true if everything goes fine, else it returns false.

3.5.4 Read Functions

To retrieve data from sensor has been provided a function each sensor, has been defined also type called ***"measures"*** which is composed by three unsigned integer one each axes, that will represent the read values.

3.5.4.1 Read Accelerometer

measures getAccelerometer()

This functions return the accelerometers read values in a object called measures, in particular the function access to:

- for axis X: register 3Ch for low bits and register 3Bh for high bits;
- for axis Y: register 3Dh for low bits and register 3Eh for high bits;
- for axis Z: register 40h for low bits and register 3Fh for high bits;

3.5.4.2 Read Gyroscope

measures getGyroscope()

This functions return the gyroscopes read values in a object called measures, in particular the function access to:

- for axis X: register 44h for low bits and register 43h for high bits;
- for axis Y: register 46h for low bits and register 45h for high bits;
- for axis Z: register 48h for low bits and register 47h for high bits;

3.5.4.3 Read Magnetometer

measures getMagnetometer()

This functions return the magnetometer read values in a object called measures, in particular the function access to:

- for axis X: register 03h for low bits and register 04h for high bits;
- for axis Y: register 05h for low bits and register 06h for high bits;
- for axis Z: register 07h for low bits and register 08h for high bits;

3.5.4.4 Read Thermometer

unsigned int getThermometer()

This function returns the thermometer, that will directly written in the corresponding configuration register. In order to correctly communicate with the temperature sensor, the MPU9250 must be configured properly:

- Access the Slave 0 register ***I2C_SLV0_ADDR*** address 37 (25h), write in it 200 (C8h), to set up a write procedure to the external sensor, in particular the Msb=1 indicate a read procedure;
- In register ***I2C_SLV0_REG*** 38 (26h) must be set the register to read in this case must be set to 00, that correspond to the temperature register;
- Finally must be set to 82h the register ***I2C_SLV0_CTRL*** 39 (27h) to start the read operation of two byte of data from the sensor;
- The read values can be retrieved accessing the MPU9250:
  - Register ***EXT_SENS_DATA_00*** : for low bits, address 73 (49h);
  - Register ***EXT_SENS_DATA_01*** : for high bits, address 74 (4Ah).

3.5.5 Onboard Memory Functions

The 1C603A Multi Sensor have also an on board EEPROM memory, to correctly use it has been defined two memory function one for write procedure and another for read procedure, both functions take as input the memory location address. To make the board easily to use and to avoid data lost, has been also defined a complete memory map.

3.5.5.1 Memory Map

This board, as described in the previous paragraphs, have mounted a 1Kb (128x8) memory. This memory is used to store the tare value in order to easily correct the measures. Approximating as linear the correction factor to apply is sufficient to have gain and offset to obtain the correct calibration. To have a better correction is possible to store three different value of calibration for each sensor, that correspond to different operating condition, like different operating temperature. In particular

has been defined three values of temperature, A, B and C that will be stored memory location shown in the memory map. This data is generated starting from the measured values, taken with a sensors read procedure, that are elaborated by the tile processor of the controlling board. There are also 8 bytes that can be defined by user. The memory mapping is represented in table 3.

Sensor	**Temperature A**				**Temperature B**				**Temperature C**			
	Offset		**Gain**		**Offset**		**Gain**		**Offset**		**Gain**	
	High	Low	High	Low	High	Low	High	Low	High	Low	High	Low
X-Axis Accelerometer	0B	0A	09	08	07	06	05	04	03	02	01	00
Y-Axis Accelerometer	17	16	15	14	13	12	11	10	0F	0E	0D	0C
Z-Axis Accelerometer	23	22	21	20	1F	1E	1D	1C	1B	1A	19	18
X-Axis Gyroscope	2F	2E	2D	2C	2B	2A	29	28	27	26	25	24
Y-Axis Gyroscope	3B	3A	39	38	37	36	35	34	33	32	31	30
Z-Axis Gyroscope	47	46	45	44	43	42	41	40	3F	3E	3D	3C
X-Axis Magnetometer	53	52	51	50	4F	4E	4D	4C	4B	4A	49	48
Y-Axis Magnetometer	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54
Z-Axis Magnetometer	6B	6A	69	68	67	66	65	64	63	62	61	60
Thermometer	77	76	75	74	73	72	71	70	6F	6E	6D	6C
Temperature	7A				79				78			
User-Defined	XX	XX	XX	XX	XX	XX	XX	7F	7E	7D	7C	7B

Table 1: Memory Map of Bk1C603E

3.5.5.2 Write Memory

bool writeMemory(address : byte, data : byte)

This function takes as inputs, the memory location address and the byte of data to write in the on-board memory.

To successfully perform the function:

- Configure IMU to communicate with the memory, in ***I2C_SLV1_ADDR*** address 40 (28h) writing in it 80 (50h), in particular the Msb=0 indicate a write procedure;
- In ***I2C_SLV1_REG*** 41 (29h) the user can select the memory location to write according with the memory map;
- The data that have to be written in the memory must be stored in the register ***I2C_SLV1_DO*** 100 (64h);
- Set to 81h the register ***I2C_SLV1_CTRL*** 42 (2Ah) to send one byte of data.

3.5.5.3 Read Memory

byte readMemory(address : byte)

To read the on-board memory is necessary to:

- Configure IMU to communicate with the memory, in ***I2C_SLV1_ADDR*** address 40 (28h) writing in it D0 (50h), in particular the Msb=1 indicate a read procedure;
- In ***I2C_SLV1_REG*** 41 (29h) the user can select the memory location to read according with the memory map;
- Set to 81h the register ***I2C_SLV1_CTRL*** 42 (2Ah) to read one byte of data;
- Access the ***EXT_SENS_DATA_03*** 76 (4Ch) to obtain the read data.

4 OPTIMIZATION OF BK1B4221WTILE PROCESSOR 4M V3

This reusable block was already present in the ***AraMiS*** library as a four module (AraMiS standard) tile processor. Has been necessary to redraw it in ***Altium,*** checking also all it's components, obtaining the version 3 of the schematic. In figure 19 is shown the class diagram, can be seen that this class is a part of the 1B422_Tile_processor.

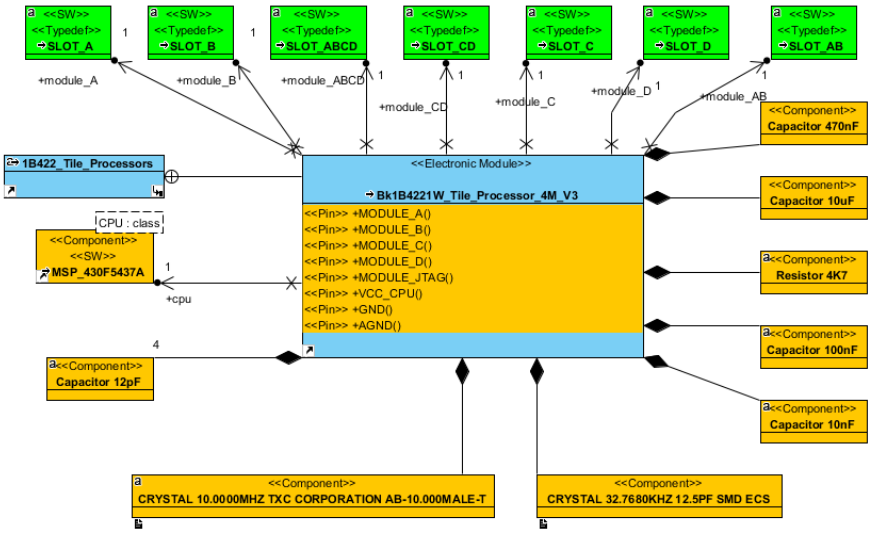


Figure 19: Bk1B4221W_Tile_Processor_4M_V3 Class Diagram

4.1 SPECIFICATION

This block must contain everything is needed to make work properly the MSP430F5437, and also:

- The block must be reusable in the AraMiS architecture;
- All the AraMiS standard for interfaces and modules have to be respected;

4.2 MAIN COMPONENTS

Main components of the board are the microcontroller, that has been chosen with integrated SPI interface and with a number of general purpose input output able to control all the devices on the board. The microcontroller chosen is optimized for ultralow-power operation reducing the impact on the needed voltage supply by the entire system.

4.2.1 Microcontroller - MSP430F5437

From datasheet (9) the Texas Instruments MSP430 family of ultralow-power microcontroller is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows the device to wake up from low-power modes to active mode in less than 5 µs.
The MSP430F5437 is microcontroller configuration with three 16-bit timers, a high-performance 12-bit analog-to-digital converter (ADC), up to four universal serial communication interfaces (USCIs), a hardware multiplier, DMA, a real-time clock (RTC) module with alarm capabilities, and up to 87 I/O pins.
Features:

- Low Supply Voltage Range: 2.2V to 3.6V;
- Ultralow Power Consumption;
- Active Mode (AM): All System Clocks Active:
    - 312 µA/MHz at 8 MHz, 3.0 V, Flash Program Execution (Typical);
    - 140 µA/MHz at 8 MHz, 3.0 V, RAM Program Execution (Typical);
- Standby Mode (LPM3):
    - Real-Time Clock (RTC) With Crystal, Watchdog, and Supply Supervisor Operational, Full RAM Retention, Fast Wakeup: 2.6 µA at 3.0 V (Typical);
    - Low-Power Oscillator (VLO), General-Purpose Counter, Watchdog, and Supply Supervisor Operational, Full RAM Retention, Fast Wakeup: 1.8

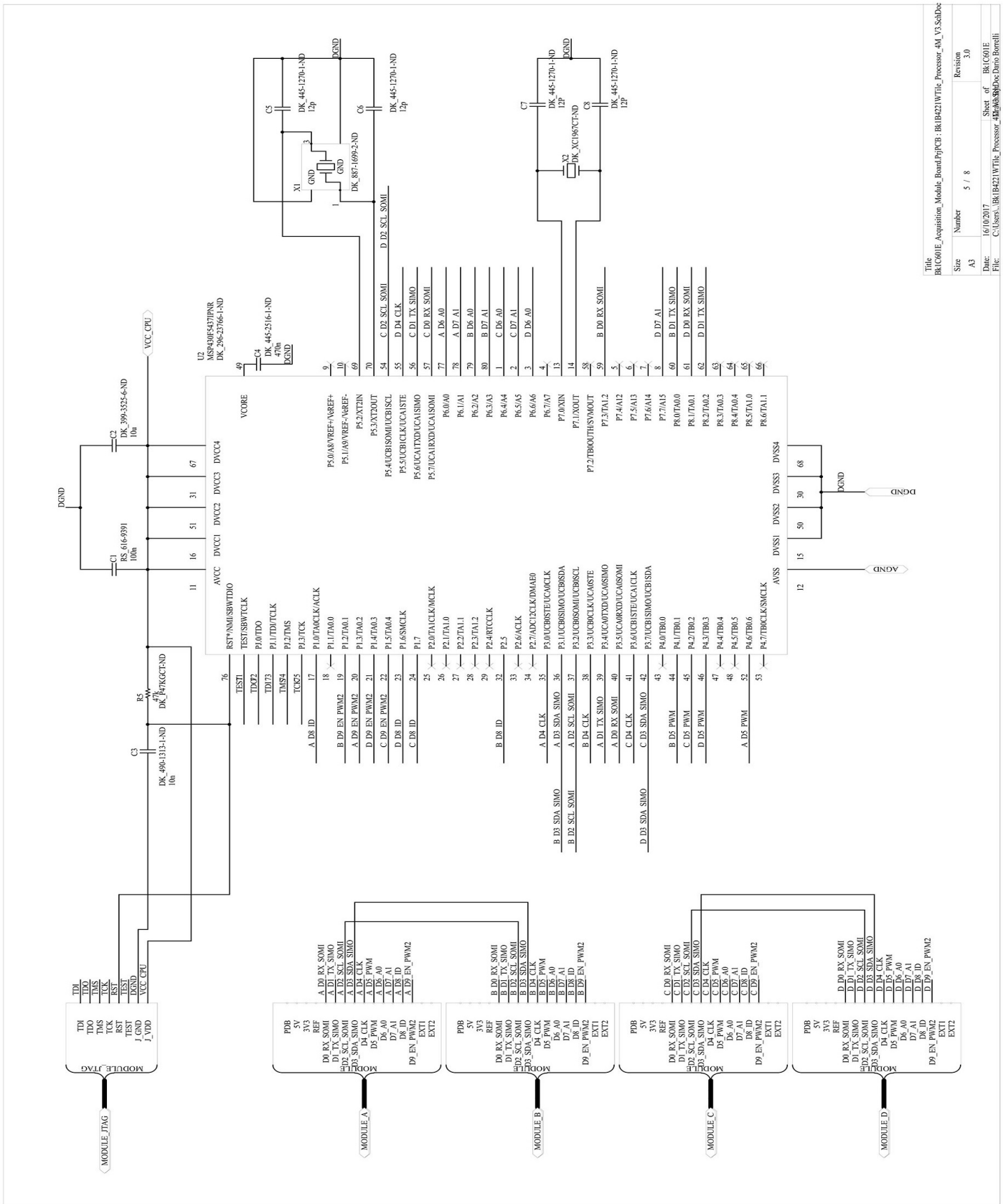
µA at 3.0 V (Typical);

- Off Mode (LPM4): Full RAM Retention, Supply Supervisor Operational, Fast Wakeup: 1.69 µA at 3.0 V (Typical);
- Wakeup From Standby Mode in Less Than 5 µs;
- 16-Bit RISC Architecture;
- Extended Memory;
- Up to 18-MHz System Clock;
- Flexible Power Management System;
- Fully Integrated LDO With Programmable Regulated Core Supply Voltage;
- Supply Voltage Supervision, Monitoring, and Brownout;
- Unified Clock System;
- FLL Control Loop for Frequency Stabilization;
- Low-Power Low-Frequency Internal Clock Source (VLO);
- Low-Frequency Trimmed Internal Reference Source (REFO);
- 32-kHz Crystals;
- High-Frequency Crystals up to 32 MHz;
- 16-Bit Timer TA0, Timer_A With Five Capture/Compare Registers;
- 16-Bit Timer TA1, Timer_A With Three Capture/Compare Registers;
- 16-Bit Timer TB0, Timer_B With Seven Capture/Compare Shadow Registers;
- Up to Four Universal Serial Communication Interfaces;
- USCI_A0, USCI_A1, USCI_A2, and USCI_A3 Each Support:
    - Enhanced UART Supports Automatic Baud-Rate Detection;
    - IrDA Encoder and Decoder;
    - Synchronous SPI;
- USCI_B0, USCI_B1, USCI_B2, and USCI_B3 Each Support:
    - I^2C;
    - Synchronous SPI;
- 12-Bit Analog-to-Digital Converter (ADC);
- Internal Reference;
- Sample-and-Hold;
- Auto-scan Feature;
- 14 External Channels, 2 Internal Channels;

- Hardware Multiplier Supporting 32-Bit Operations;
- Serial Onboard Programming, No External Programming Voltage Needed;
- Three-Channel Internal DMA;
- Basic Timer With RTC Feature.

4.3 DESIGN

This block contain everything is needed to make work properly the MSP430F5437. It's ports are the power supply VCC_CPU, the two ground (for the analog (ADC) and for digital circuits of the microcontroller), a JTAG module that contain all the signal needed for JTAG interface and 4 general-purpose module that maps all the remaining pin of the microcontroller. This mapping is already defined in ***AraMiS***.
There are two crystals, one at 32,765KHz used as ADC sampling frequency and another at 10MHz used as clock frequency of the microcontroller. The capacitors C1, C2, C3, C4 are decoupling capacitors, in particular three in PCB design phase they have to be placed the power supply pins of the microcontroller. Furthermore the capacitor C4, following the instructions contained in the datasheet (9), need an high temperature coefficient and low uncertainty, in this design has been chosen a X8R temperature coefficient capacitor with 10% of uncertainty.


Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : Bk1B4221WTile_Processor_4M_V3.SchDoc
Size
A3
Number
5 / 8
Revision
3.0
Date:
16/10/2017
Sheet of
Bk1C601E
File:
C:\Users\...\Bk1B4221WTile_Processor_4M_V3.SchDoc
Drawn By:
Dario Borrelli
U2
MSP430F5437IPNR
DK_296-23766-1-ND
VCC_CPU
DGND
AGND
C1
RS_616-9391
100n
C2
DK_399-3525-6-ND
10u
C3
DK_490-1313-1-ND
10n
R5
4k7
DK_P4.7KGCT-ND
C4
DK_445-2516-1-ND
470n
C5
DK_445-1270-1-ND
12p
C6
DK_445-1270-1-ND
12p
C7
DK_445-1270-1-ND
12P
C8
DK_445-1270-1-ND
12P
X1
DK_887-1699-2-ND
X2
DK_XC1967CT-ND
MODULE_JTAG
MODULE_A
MODULE_B
MODULE_C
MODULE_D

4.4 SOFTWARE

The software was already defined in the AraMiS project, so all the software used was checked and optimized to work properly. In figure 20 can be seen the class diagram of the software structure, in particular every module of the processor used is divided in slots. Each slot have different functions, based on capabilities supported by that module. In the designed system have been used slot A and slot B capabilities to control an SPI communications through UART0 and UART1, while other slot have been used the IOdriver functions to control properly the GPIO pins.

Figure 20: Bk1B4221 Tile processor software

5 DESIGN OF BK1B4853 CAN INTERFACE

This reusable block design is completely new. The purpose of this electronic module is to have a reusable block that easily permits to implements a fully flexible, stand-alone SPI-CAN interface. As can be seen in figure 21 where is shown the corresponding class diagram, contain everything is needed to use a CAN interface with an SPI communication, in particular it contains also a connector, configurable speed and filters for the transceiver, under voltage protection and has been created a software class for the tile processor to easily handle all the commands of the CAN controller.

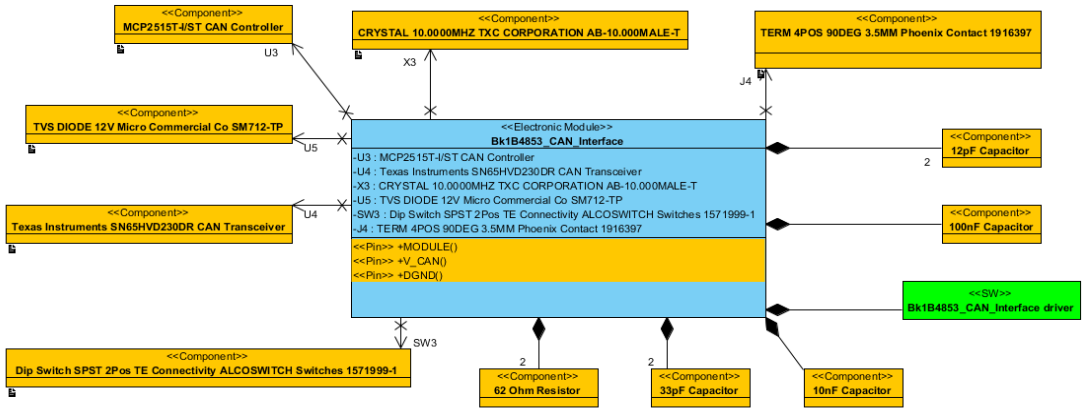


Figure 21: Bk1B4853_CAN_Interface Class Diagram

5.1 SPECIFICATION

The board designed must have everything is needed to realize a SPI-CAN interface, in particular:

- Must respect all the AraMiS standards;
- The CAN communication speed must be configurable;
- As a part of a CAN bus, must respect all the CAN specification, like the 120Ohm termination resistance, type of connector, protection of bus lines, etc..;
- To avoid noise problems, in case of long wires, must be considered the possibility to add filters on the transceiver TX and RX lines;
- Must have a fully compatible SPI interface.

5.2 MAIN COMPONENTS

The components of this block are a SPI-CAN interface a CAN transceiver to handle the CAN protocol.

5.2.1 CAN Interface - MCP2515T-I/ST

From datasheet (10) Microchip Technology's MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification, Version 2.0B. It is capable of transmitting and receiving both standard and extended data and remote frames. The MCP2515 has two acceptance masks and six acceptance filters that are used to filter out unwanted messages, thereby reducing the host MCU's overhead. The MCP2515 interfaces with microcontrollers (MCUs) via an industry standard Serial Peripheral Interface (SPI).
Features:

- Implements CAN V2.0B at 1 Mb/s:
  - 0 to 8-byte length in the data field;
  - Standard and extended data and remote frames;
- Receive Buffers, Masks and Filters:

  - Two receive buffers with prioritized message storage;
  - Six 29-bit filters;
  - Two 29-bit masks;
- Data Byte Filtering on the First Two Data Bytes (applies to standard data frames);
- Three Transmit Buffers with Prioritization and Abort Features;
- High-Speed SPI Interface (10 MHz):
  - SPI modes 0,0 and 1,1;
- One-Shot mode Ensures Message Transmission is Attempted Only One Time;
- Clock Out Pin with Programmable Pre-scaler:
  - Can be used as a clock source for other device(s);
- Start-of-Frame (SOF) Signal is Available for Monitoring the SOF Signal:
  - Can be used for time slot-based protocols and/or bus diagnostics to detect early bus degradation;
- Interrupt Output Pin with Selectable Enables;
- Buffer Full Output Pins Configurable as:
  - Interrupt output for each receive buffer;
  - General purpose output;
- Request-to-Send (RTS) Input Pins Individually Configurable as:
  - Control pins to request transmission for each transmit buffer;
  - General purpose inputs;
- Low-Power CMOS Technology:
  - Operates from 2.7V-5.5V;
  - 5mA active current (typical);
  - 1µA standby current (typical) (Sleep mode);
- Temperature Ranges Supported:
  - Industrial (I): -40°C to +85°C;

5.2.2 CAN Transceiver – SN65HVD230DR

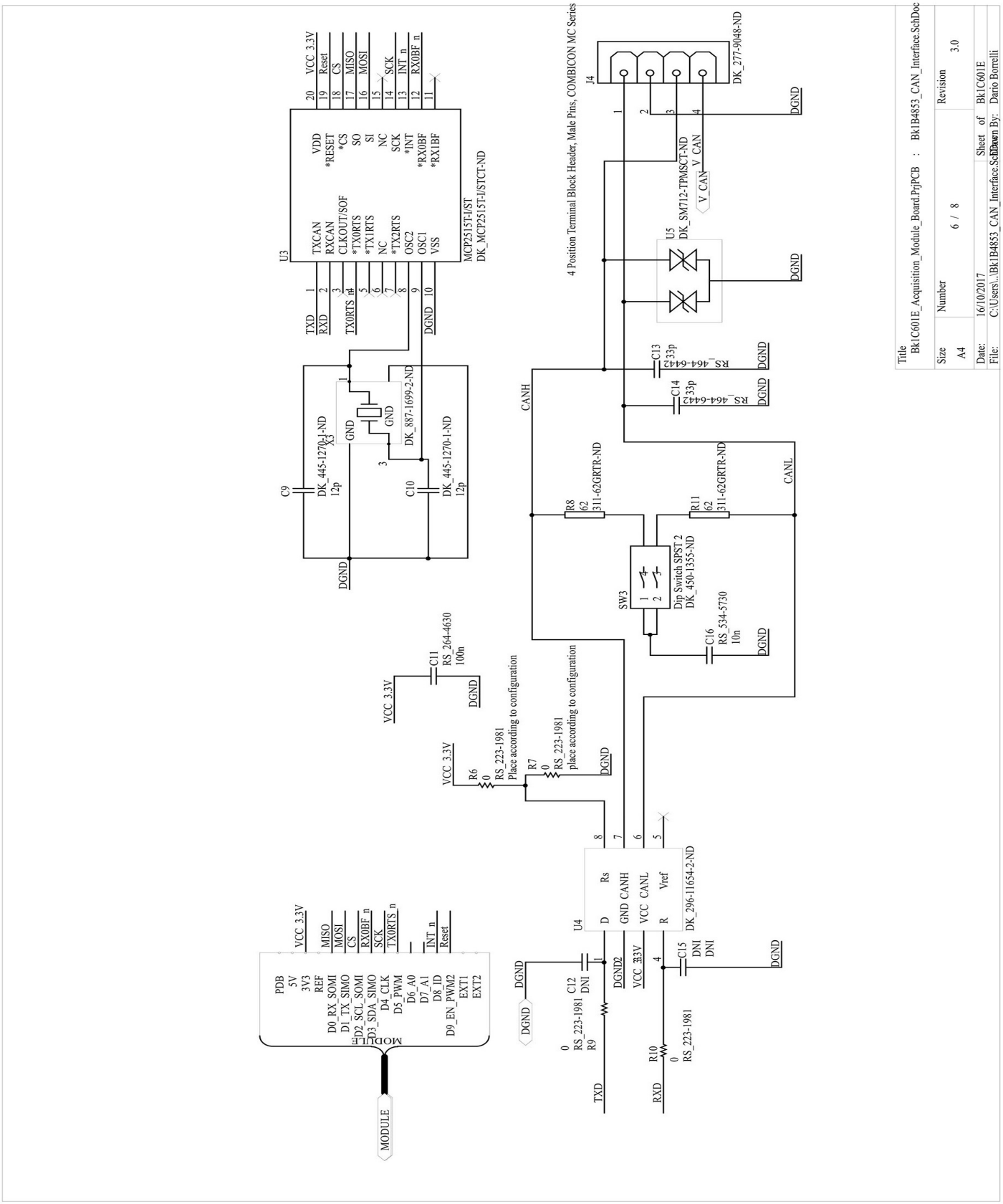
From datasheet (11) the SN65HVD230 controller area network (CAN) transceiver is compatible to the specifications of the ISO 11898-2 High Speed CAN Physical Layer standard (transceiver). These device is designed for data rates up to 1 megabit per second (Mbps), and include many protection features providing device and CAN network robustness. The SN65HVD23x transceivers are designed for use with the Texas Instruments 3.3VµPs, MCUs and DSPs with CAN controllers, or with equivalent protocol controller devices. The devices are intended for use in applications employing the CAN serial communication physical layer in accordance with the ISO 11898 standard. Designed for operation in especially harsh environments, these devices feature cross wire protection, loss of ground and over-voltage protection, over-temperature protection, as well as wide common mode range of operation.

The CAN transceiver is the CAN physical layer and interfaces the single ended host CAN protocol controller with the differential CAN bus found in industrial, building automation, and automotive applications. These devices operate over a -2 V to 7 V common mode range on the bus, and can withstand common mode transients of ±25V. The R_S pin (pin 8) on the SN65HVD230 and SN65HVD231 provides three different modes of operation: high speed mode, slope control mode, and low-power mode. The high speed mode of operation is selected by connecting the R_S pin to ground, allowing the transmitter output transistors to switch on and off as fast as possible with no limitation on the rise and fall slopes. The rise and fall slopes can also be adjusted by connecting a resistor in series between the R_S pin and ground. The slope will be proportional to the pin's output current. With a resistor value of 10kΩ the device will have a slew rate of ~15 V/µs, and with a resistor value of 100kΩ the device will have a slew rate of ~2 V/µs. The SN65HVD230 enters a low current standby mode (listen only) during which the driver is switched off and the receiver remains active if a high logic level is applied to the R_S pin. This mode provides a lower power consumption mode than normal mode while still allowing the CAN controller to monitor the bus for activity indicating it should return the transceiver to normal mode or slope control mode. The host controller (MCU, DSP) returns the

device to a transmitting mode (high speed or slope control) when it wants to transmit a message to the bus or if during standby mode it received bus traffic indicating the need to once again be ready to transmit.

5.3 DESIGN

The design realized takes as input a module in which are used all the needed signal to use SPI interface and GPIO pins to control the MCP2515T, in particular to make this integrated circuit work properly is needed a clock source that has been realized using a crystal and corresponding decoupling capacitors C10 and C9. The MCP2515T is connected to a CAN transceiver that handle the communications on the CAN_L and CAN_H wires. The transceiver communication speed is hardware configurable mounting or not the resistors R6, R7. Furthermore in case of noise problems on the TXD and RXD of the transceiver is already expected the possibility to set-up two low pass filter, replacing the 0Ω resistors R10,R9 and DNI capacitances C15, C12. To interface with the CAN wires is used a 4-wire CAN connector, a switch to configure this node as a termination, that is realized using a series of two 68Ω resistor R8,R11 tied in the middle to ground. To protect the transceiver in case of under-voltage of the CAN wires there are two diodes.


Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : Bk1B4853_CAN_Interface.SchDoc
Size A4
Number 6 / 8
Revision 3.0
Date: 16/10/2017
Sheet of Bk1C601E
File: C:\Users\..\Bk1B4853_CAN_Interface.SchDoc
Drawn By: Dario Borrelli
U3
MCP2515T-I/ST
DK_MCP2515T-I/STCT-ND
TXCAN
RXCAN
CLKOUT/SOF
*TX0RTS
*TX1RTS
NC
*TX2RTS
OSC2
OSC1
VSS
VDD
*RESET
*CS
SO
SI
NC
SCK
*INT
*RX0BF
*RX1BF
TXD
RXD
TX0RTS
VCC_3.3V
Reset
CS
MISO
MOSI
SCK
INT_n
RX0BF_n
X3
DK_887-1699-2-ND
DK_445-1270-1-ND
C9
C10
12p
DGND
4 Position Terminal Block Header, Male Pins, COMBICON MC Series
J4
DK_277-9048-ND
U5
DK_SM712-TPMSCT-ND
V_CAN
C13
C14
RS_464-6442
33p
CANH
CANL
R8
R11
62
311-62GRTR-ND
SW3
Dip Switch SPST 2
DK_450-1355-ND
C16
RS_534-5730
10n
C11
RS_264-4630
100n
R6
R7
0
RS_223-1981
Place according to configuration
U4
DK_296-11654-2-ND
D
Rs
GND
CANH
VCC
CANL
R
Vref
DGND2
VCC_3.3V
C12
C15
DNI
R9
R10
TXD
RXD
MODULE
PDB
5V
3V3
REF
D0_RX_SOMI
D1_TX_SIMO
D2_SCL_SOMI
D3_SDA_SIMO
D4_CLK
D5_PWM
D6_A0
D7_A1
D8_ID
D9_EN_PWM2
EXT1
EXT2
MISO
MOSI
CS
RX0BF_n
SCK
TX0RTS_n
INT_n
Reset

5.4 SOFTWARE

To use correctly the SPI-CAN interface, has been created a software class for tile processor. The controller generate all control bits and CRC field of the data frame, it have three buffer to send data and two buffer to receive data, so can be set different priority for each message and can be set two address for receiving data. In figure 22 is shown the corresponding class class diagram, in particular the MCP2515_RM is the the complete register map of the MCP2515. The software has been developed for the Bk1B4221WTile Processor, and handle the CAN controller in it's most important functions. All the developed software can be find in Appendix B Bk1C601E.

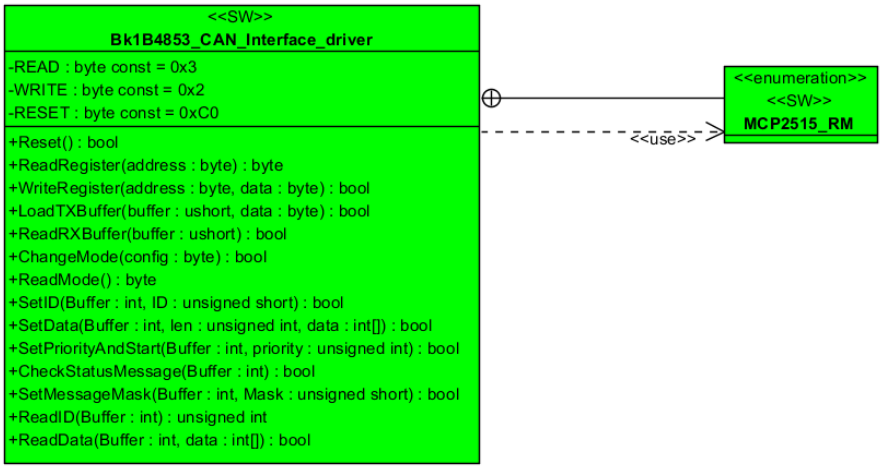


Figure 22: Bk1b4853 CAN interface software class

5.4.1 SPI Operation

The SPI read and write operation is performed writing to the MCP2515 using instruction set defined in datasheet (10), the following function implements the most important commands of the instruction set.

5.4.1.1 Reset

bool Reset()

This function when called reset all internal register and put the MCP2515 in the configuration mode, it returns true if the operation is performed successfully otherwise it returns false.

5.4.1.2 Read Register

byte ReadRegister(address : byte)

The function read the register pointed by the address received as input, it returns the read data if the operation is performed successfully otherwise it returns zero.
This is done writing the read command (equal to 3), thus is written the address of the register to read, finally is performed the read operation.

5.4.1.3 Write Register

byte WriteRegister(address : byte, data : byte)

The function write data (variable of type byte) to the register pointed by the address received as input, it returns true if the operation is performed successfully otherwise it returns false.
This is done writing the write command (equal to 2), thus is written the address of the register to write and finally is written the data.

5.4.1.4 Read RX Buffer

byte ReadRXBuffer(buffer : ushort)

This function use the Read RX Buffer instruction that permits to read the RX buffer directly, without performing a read operation on the corresponding buffer register, reducing the overhead of one byte (the register address byte). The MCP2515 have two different receiver buffer, furthermore can be read both identifier and data of the message, for that reason has been defined an input variable called buffer to choose from which buffer read and which will be the first data to be read.

buffer=0	Receiver buffer 0, start with the most significant bits of the identifier
buffer=1	Receiver buffer 0, start with data bits
buffer=2	Receiver buffer 1, start with the most significant bits of the identifier
buffer=3	Receiver buffer 1, start with data bits

Table 2: Read RX Buffer variable definition

5.4.1.5 Load TX Buffer

byte LoadTXBuffer(buffer : ushort, data : byte)

This function use the Load TX Buffer instruction that permits to load data in the TX buffer directly, without performing a write operation on the corresponding buffer register, reducing the overhead of one byte (the register address byte). The MCP2515 have three different transmission buffer, furthermore the data can be written starting from the most significant byte or from the least significant byte, for that reason has been defined an input variable called buffer to choose where the data should be written.

buffer=0	Transmission buffer 0, write the most significant byte
buffer=1	Transmission buffer 0, write the least significant byte
buffer=2	Transmission buffer 1, write the most significant byte
buffer=3	Transmission buffer 1, write the least significant byte
buffer=4	Transmission buffer 2, write the most significant byte
buffer=5	Transmission buffer 2, write the least significant byte

Table 3: Load TX Buffer variable definition

5.4.2 Configuration

The MCP2515 can operate in five different modes. For that reason has been realized a set of functions to access easily to the registers that control this parameters.

5.4.2.1 Change Mode

Bool ChangeMode(config : byte)

This function permits to change the operating mode of the MCP2515, in particular to change operating mode is necessary to access to the register CANCTRL at address Fh, the function access directly to the register and write in it the config variable. For that reason the user must configure the input variable config properly according to the datasheet (10).

5.4.2.2 Read Mode

byte ReadMode()

This function permits reads the status register (CANSTAT) of the MCP2515, in this register located at address Eh are stored information on the operating mode of the integrated circuit and interrupts flags.

5.4.3 Message Transmission

To use correctly the CAN bus protocol is necessary to set the identifier field and data field of the data frame that have to be sent on the bus. For that reason has been created a set of function with that purpose.

5.4.3.1 Set Identifier

bool SetID(Buffer : int, ID : unsigned short)

This function is used to set the 11 bits standard identifier of message to be sent, by the corresponding transmit buffer received as input in variable buffer, the id will be set in one of the three TXBnSIDH register located at 31h, 41h, 51h and in the TXBnSIDL register located at 32h, 42h, 52h. In particular in this register have to be set also the type of identifier of the message in this case standard. This function returns true if everything is performed successfully otherwise it returns false.

5.4.3.2 Set Data

bool SetData(Buffer : int, len : unsigned int, data : int[])

This function set the data field of the message that have to sent. It takes three input parameters, Buffer identify in which buffer the message to be sent is, len define the length of the data field, data is an array that contain the data the have to be written in the data field. Firstly is check if len have the correct dimension, after on the selected buffer is written the data field length in the register TXBnDLC located at address 35h, 45h, 55h. Finally the data to be written is split in byte and written in the corresponding registers TXBnDm. This function returns true if everything is performed successfully otherwise it returns false.

5.4.3.3 Set Priority and Start

bool SetPriorityAndStart(Buffer : int, priority : unsigned int)

This function permits to set the priority of the message and start the communication on the select buffer. It receive as inputs: Buffer that identify the buffer in which the message is located and the priority of the message that can be from 0-3 (3 represent the highest priority message). The operations are done accessing the register TXBnCTRL located at 30h, 40h, 50h and set it properly. This function returns true if everything is performed successfully otherwise it returns false.

5.4.3.4 Check Status Message

bool CheckStatusMessage(Buffer : int)

This function check if the buffer selected doesn't have messages pending, it return true if there are no message pending otherwise it returns false. This operation is performed accessing the register TXBnCTRL located at 30h, 40h, 50h and checking the third bit TREQ.

5.4.4 Message Reception

The message reception is handle by three buffer register that have a settable message filter to determine if the message should be loaded into the receiver buffer.

5.4.4.1 Set Message Mask

bool SetMessageMask(Buffer : int, Mask : unsigned short)

This function permits to set a message mask on the message identifier to determine if the message should be loaded into the receiver buffer. It receive as inputs: Buffer that identify the buffer in which mask should be applied and Mask that is the value that will used as mask. This operation is performed writing the mask in the register RXMnSIDH located at 00h, 04h, 08h, 10h, 14h, 18h, that contain the most significant bits of the mask to be applied, and in the register RXMnSIDL located at 01h, 05h, 09h, 11h, 15h, 19h, which contain the least significant bits of the mask. This function returns true if everything is performed successfully otherwise it returns false.

5.4.4.2 Read Identifier

unsigned int ReadID(Buffer : int)

This function read the identifier of the read message on the buffer defined by the parameter Buffer received as input, returning the read value in an unsigned int. This operation is performed reading the register RXBnSIDH located at 61h, 71h, for the most significant bits of the identifier, and reading the register RXBnSIDL located at 62h, 72h for the least significant bits.

5.4.4.3 Read Data

bool ReadData(Buffer : int, data : int[])

This function store in the variable data the read message on the buffer defined by the parameter Buffer received as input. This operation is performed reading the register RXBnDLC located at 65h, 75h to determine the data length, after are performed read operation on the registers RXBnDm in which the data is stored. This function returns true if everything is performed successfully otherwise it returns false.

6 DESIGN OF BK1C601E ACQUISITION MODULE

The Bk1C601E Acquisition Module is composed by two elements.
The Bk1C601E Acquisition Module board comprehend all the electronics, in particular the board contain:

- Bk1B4221WTile_Processor, that is used has core of the board to handle all the other elements on the board;
- Bk1B4853_CAN_Interface, that is used to handle the CAN bus protocol;
- SPI connectors, to interface with the 1C603A Multi-sensor;
- JTAG Connector, to permits to an PC to interface easily with the Bk1B4221WTile_Processor;
- Analog inputs Connectors, to use the analogic input pins of the Bk1B4221WTile_Processor, to make additional measures;
- Push buttons, to reset the microcontroller,;
- Led, the user can configure it to have a visual feedback of the operation of the microcontroller.

The second element is the Bk1C601E Acquisition Module box, that is an enclosure box for the electronic board. The entire system is represented in the class diagram shown in figure 23.

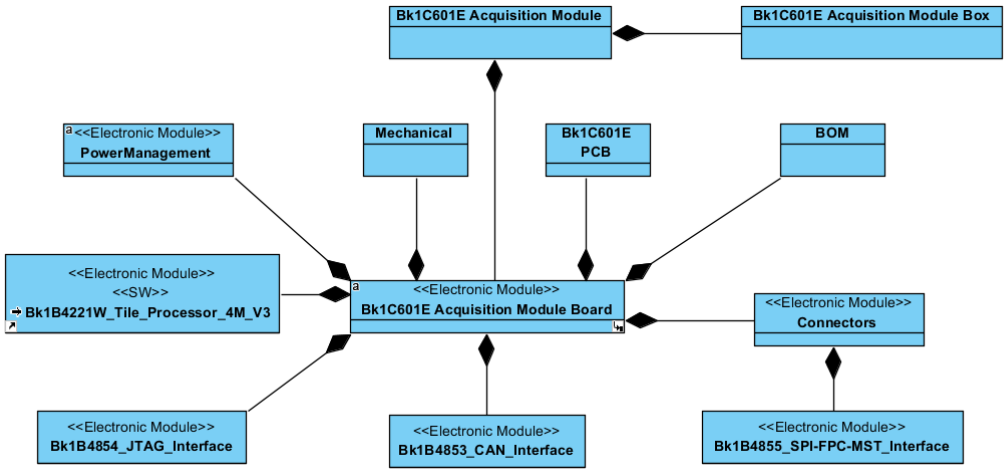


Figure 23: Bk1C601E Acquisition Module Class diagram

6.1 SPECIFICATION

This board is realized to control and acquire data from till sixteen Bk1C601A board connected through FFC cable using SPI protocol. The data acquired could be also sent to an host computer, and this board must be controllable from this host computer or from another control board, for this reason are also needed additional interfaces like JTAG and CAN.

6.1.1 Functional Specification

The board designed must have this functionalities:

- 16 SPI connector;
- CAN interface;
- JTAG interface;
- Capability to control and provide supply for sixteen 1C601A;
- Capability to store on board the read data;
- Capability to make calculation, in order to correct read data;
- Capability to make additional voltage measurement;
- Capability to easily control the behavior of the board processor .

6.1.2 Mechanical Specification

The board must be small enough to be portable, in particular has been identified a maximum dimensions of 10cmX10cm. However must be enough space to easily use the SPI connectors and at the same time enough space for all the other connectors and components. The final board design have smaller dimensions that are 5cmX7,5cm. In appendix is shown the drawing of the board. The enclosure must be bigger enough to contain the board and permits an easily access to the connectors and wires.

6.2 MAIN COMPONENTS

Starting from the specification this board is designed in many different block divided by functionalities, and all this block are made completely reusable for future applications. Main components are the Bk1B4221WTile_Processor described in chapter 4 and the Bk14853_CAN_Interface described in the chapter 5 . The final project has been realized with an hierarchical structure that is shown in figure 24.

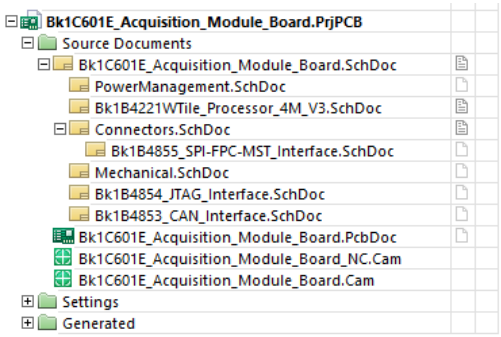
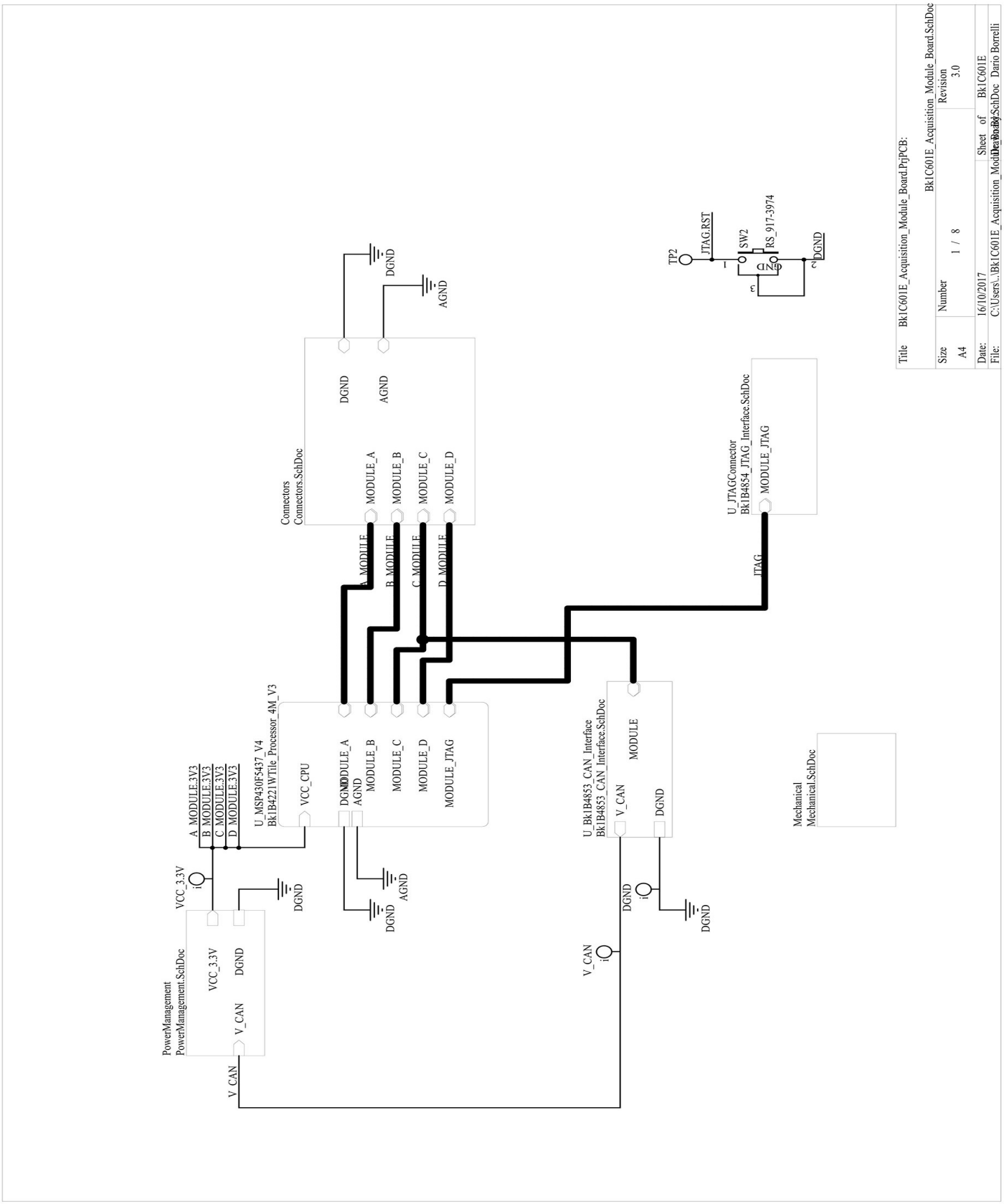


Figure 24: Sheet structure of Bk1C601E

6.3 DESIGN

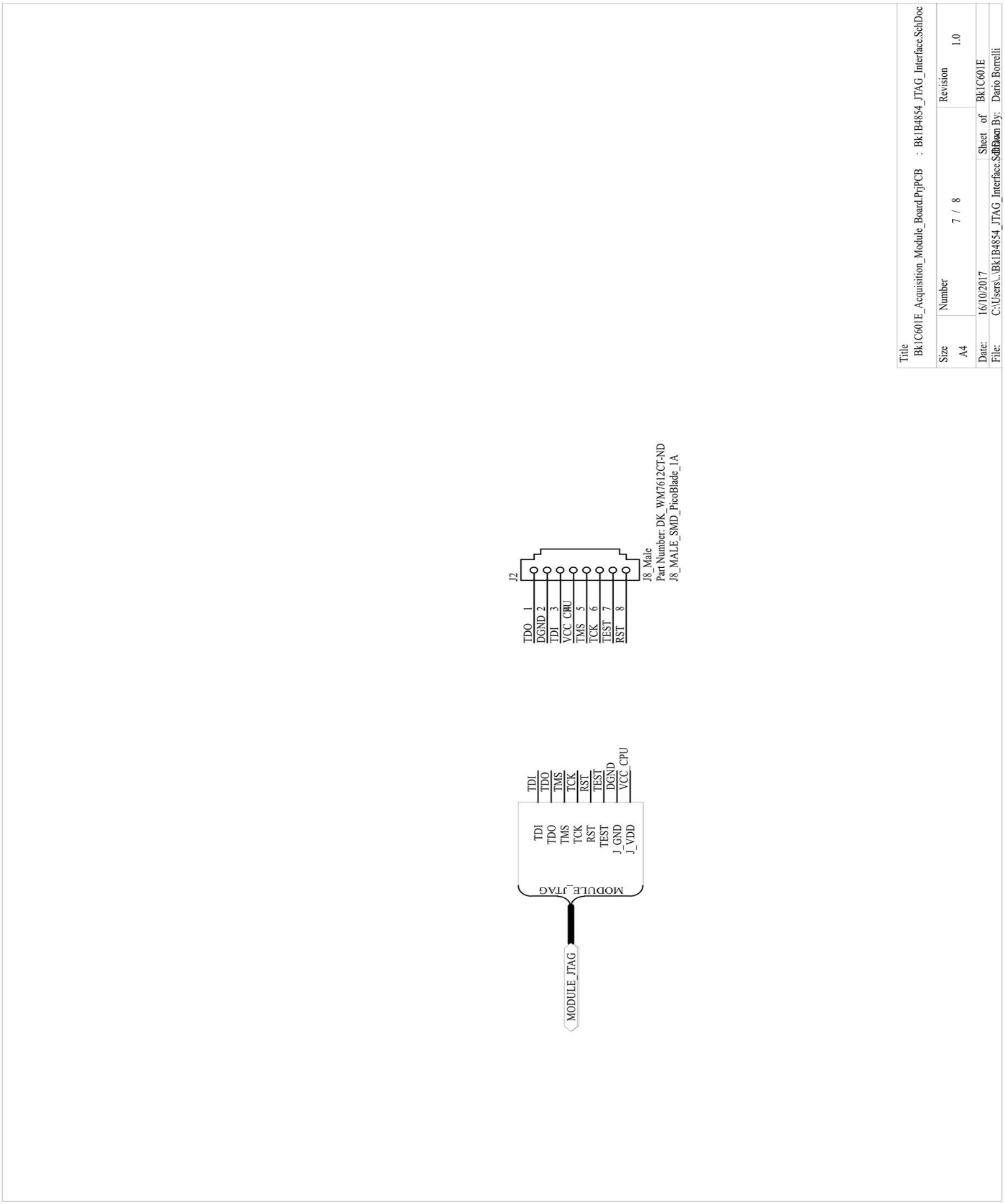
The first step in the board designing is the choose of microcontroller, from the specification has been calculated the numbers of general purpose input output port needed, the interfaces needed and the generally performance needed. In the AraMiS project there was already a class of processors called tile processors, they have also a standard definition of it's modules. Since they match all the needs and permit an easy integration in the AraMiS project has been decided to use the 4 module tile processor that has been optimized as described in chapter 4 . To make it easily programmable has been added a 8pin connector for the JTAG interface (comprehensive of dedicated 3,3Volts power supply, defined in a reusable blocks for future applications), a momentary switch linked with a 1mm test point and to reset the microcontroller that can be seen in page 1 of the schematic, a 8pin connector linked with seven analog

input and to analog ground of the microcontroller and two led connected to the microcontroller GPIO configurable by used in order to have a visual feedback of processor behaviour. However this processor doesn't have a CAN interface. To add the CAN interface has been necessary choose a CAN transceiver and CAN interface module, in particular has been chosen a SPI-CAN in order to maintain each block more reusable as possible, in fact the CAN interface (defined in a reusable block) is fully compatible with any microcontroller that have a SPI interface and it's hardware configurable to adapt to different CAN bus, the design of this module as been described in depth in the chapter 5 . This schematic represent the top level of the project, where all the different reusable blocks and schematic are connected together. In this schematic is possible to see the supply organization of the board. The power supply is taken from the CAN connector supply pin, is protected from over voltage and under voltage (power management schematic), and supply the entire board. The schematic connector contain all the SPI connectors, Analogic connector and the two led. The mechanical schematic contains a mechanical component defined in library in which there are the six fixing holes. In the next page is possible to see the top sheet of the project called Bk1C601E_Acquisition_Module_Board.


Title Bk1C601E_Acquisition_Module_Board.PrjPCB:
Size A4
Number 1 / 8
Bk1C601E_Acquisition_Module_Board.SchDoc
Revision 3.0
Date: 16/10/2017
Sheet of Bk1C601E
File: C:\Users\...\Bk1C601E_Acquisition_Module_Board.SchDoc
Dario Borrelli
Connectors
Connectors.SchDoc
DGND
AGND
MODULE_A
MODULE_B
MODULE_C
MODULE_D
A_MODULE
B_MODULE
C_MODULE
D_MODULE
U_JTAGConnector
Bk1B4854_JTAG_Interface.SchDoc
MODULE_JTAG
JTAG
TP2
JTAG.RST
SW2
RS_917-3974
U_MSP430F5437_V4
Bk1B4221WTile_Processor_4M_V3
VCC_CPU
A_MODULE.3V3
B_MODULE.3V3
C_MODULE.3V3
D_MODULE.3V3
VCC_3.3V
U_Bk1B4853_CAN_Interface
Bk1B4853_CAN_Interface.SchDoc
V_CAN
MODULE
PowerManagement
PowerManagement.SchDoc
Mechanical
Mechanical.SchDoc

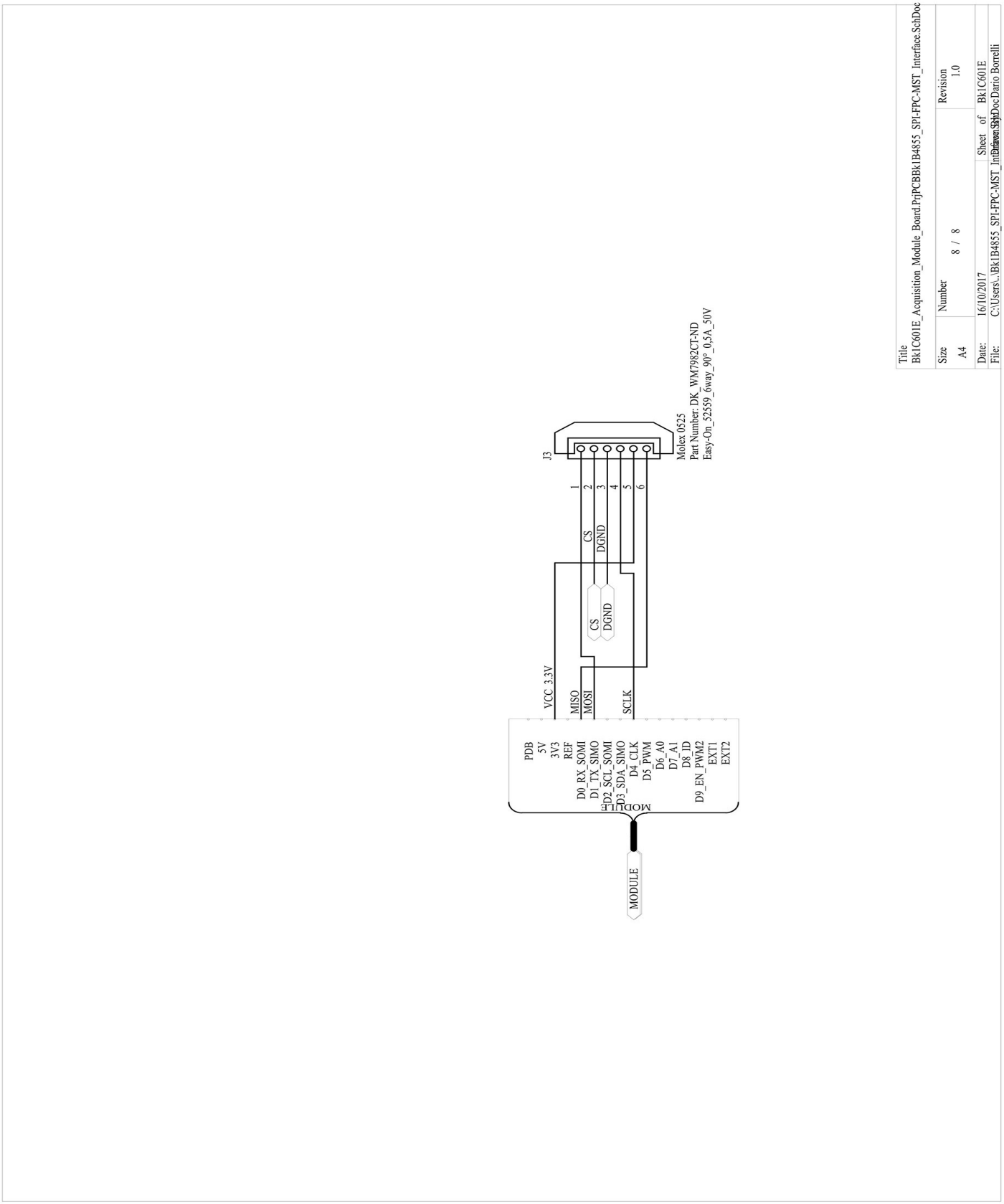
6.3.1 Reusable Block - Bk1B4854_JTAG_Interface

As it's shown in schematic page 7 this reusable block contain a 8 pin connector that provide all the wires necessary to use the JTAG interface. It takes as input a JTAG module and connect it to the correct pin out of the connector, the input module and the pin out are both a standard defined in AraMiS.


Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : Bk1B4854_JTAG_Interface.SchDoc
Size
A4
Number
7 / 8
Revision
1.0
Date:
16/10/2017
Sheet of
Bk1C601E
File:
C:\Users\...\Bk1B4854_JTAG_Interface.SchDoc
Drawn By:
Dario Borrelli
J2
TDO 1
DGND 2
TDI 3
VCC CPU
TMS 5
TCK 6
TEST 7
RST 8
J8_Male
Part Number: DK_WM7612CT-ND
J8_MALE_SMD_PicoBlade_1A
TDI
TDO
TMS
TCK
RST
TEST
DGND
VCC_CPU
MODULE_JTAG
TDI
TDO
TMS
TCK
RST
TEST
J_GND
J_VDD
MODULE_JTAG

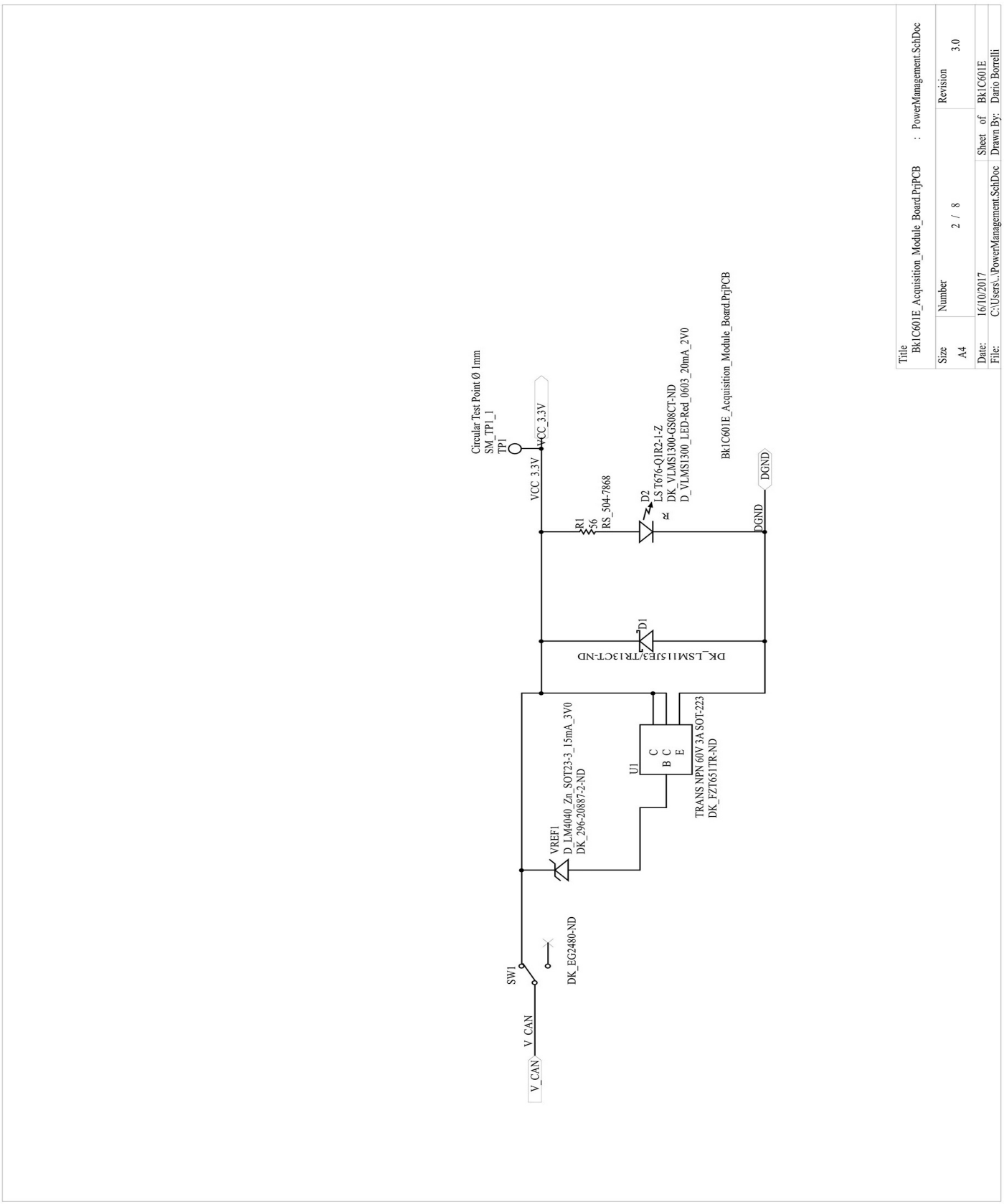
6.3.2 Reusable Block - Bk1B4855_SPI-FPC-MST_Interface

In this reusable block is present the connector of FPC master SPI interface. The schematic is shown in the next page, it takes as input a module, a chip select wire and ground, and connect its properly to match the slave SPI interface. The pinout for the FPC master connector to use not crossed cable is inverter with the respect of the slave connector. The connector used in this schematic a vertical mount in order to respect the mechanical specification.


Title
Bk1C601E_Acquisition_Module_Board.PrjPCBBk1B4855_SPI-FPC-MST_Interface.SchDoc
Size
A4
Number
8 / 8
Revision
1.0
Date:
16/10/2017
Sheet of Bk1C601E
File:
C:\Users\...\Bk1B4855_SPI-FPC-MST_Interface.SchDoc
Dario Borrelli
J3
Molex 0525
Part Number: DK_WM7982CT-ND
Easy-On_52559_6way_90°_0,5A_50V
1
2
3
4
5
6
CS
DGND
CS
DGND
VCC_3.3V
MISO
MOSI
SCLK
MODULE
PDB
5V
3V3
REF
D0_RX_SOMI
D1_TX_SIMO
D2_SCL_SOMI
D3_SDA_SIMO
D4_CLK
D5_PWM
D6_A0
D7_A1
D8_ID
D9_EN_PWM2
EXT1
EXT2
MODULE

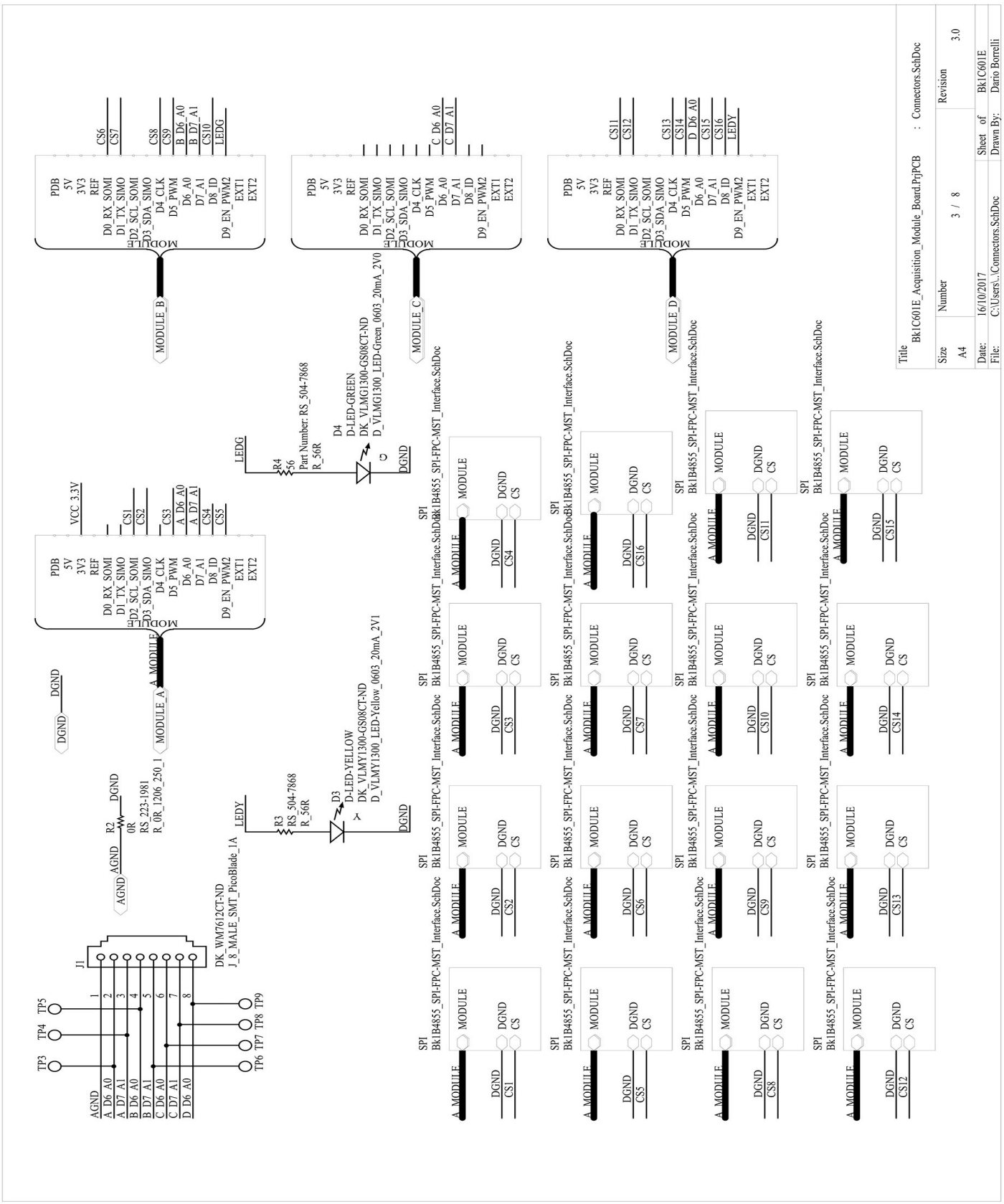
6.3.3 Power Management

For the power management are needed 3,3Volts to supply all the components on boards and all the connected 1C601A boards, at the beginning has been used a voltage regulator that from a 12 Volts source (coming from external source through a connector) and with a switch it was possible to select the 3,3Volts coming from the CAN connector supply pin, but to respect the mechanical specification has been necessary to remove the 12Volts connector to reduce the board dimension, for that reason was useless to have a voltage regulator so it was removed. Has been kept only the 3,3Volt coming from the CAN connector supply pin adding an additional over-voltage and under-voltage protection, with a switch that is used to switch-off the board removing the power supply to the board.
This schematic is realized custom for this board, it takes as input only the power supply and the ground, and give in output the power supply protected.
The over-voltage protection is realized using a 3Volt voltage reference diode and a NPN transistor, when the input voltage is over than 3,8Volt the transistor goes on insulating the board, blocking the maximum output supply voltage at 4,25Volts (the base emitter junction of the NPN transistor saturate at 1,25Volts). The under-voltage protection is realized using an high speed Schottky diode capable to sink up to 1A, when the input voltage goes under -0,3Volts the diode goes on sinking all the current and blocking the voltage. In addition there are a switch to remove the voltage supply to the board and a red led with a 1mm test point, in order to easily see if there is power supply in board or not. The schematic can be seen in the following page.


Circular Test Point Ø 1mm
SM_TP1_1
TP1
VCC_3.3V
VCC_3.3V
R1
56
RS_504-7868
D2
LS T676-Q1R2-1-Z
DK_VLMS1300-GS08CT-ND
D_VLMS1300_LED-Red_0603_20mA_2V0
Bk1C601E_Acquisition_Module_Board.PrjPCB
DGND
DGND
D1
DK_LSM115JE3/TR13CT-ND
VREF1
D_LM4040_Zn_SOT23-3_15mA_3V0
DK_296-20687-2-ND
U1
C
B C
E
TRANS NPN 60V 3A SOT-223
DK_FZT651TR-ND
SW1
DK_EG2480-ND
V_CAN
V_CAN
Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : PowerManagement.SchDoc
Size
A4
Number
2 / 8
Revision
3.0
Date:
16/10/2017
Sheet of
Bk1C601E
File:
C:\Users\..\PowerManagement.SchDoc
Drawn By:
Dario Borrelli

6.3.4 Connector

In this schematic there are all the connector organization, in particular it takes as inputs all the four modules of the microcontroller. Can be seen that for the SPI connectors is used the SPI wires of the module A, other wires are used for SPI chip selects, to control a yellow and a green led, and for the analog connector. For the analog connector every pin has been linked with a 1mm test point to easily sold cable or to control the acquired voltage. In this schematic is also present the two ground (analog and digital) connected together with 0Ω resistor. The schematic can be seen in the following page.


Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : Connectors.SchDoc
Size A4
Number 3 / 8
Revision 3.0
Date: 16/10/2017
File: C:\Users\...\Connectors.SchDoc
Sheet of Bk1C601E
Drawn By: Dario Borrelli
MODULE_A
MODULE_B
MODULE_C
MODULE_D
PDB
5V
3V3
REF
D0_RX_SOMI
D1_TX_SIMO
D2_SCL_SOMI
D3_SDA_SIMO
D4_CLK
D5_PWM
D6_A0
D7_A1
D8_ID
D9_EN_PWM2
EXT1
EXT2
VCC 3.3V
DGND
AGND
R2 0R RS 223-1981 R_0R_1206_250_1
J1 DK_WM7612CT-ND J_8_MALE_SMT_PicoBlade_1A
TP3 TP4 TP5 TP6 TP7 TP8 TP9
R3 RS_504-7868 R_56R
D3 D-LED-YELLOW DK_VLMY1300-GS08CT-ND D_VLMY1300_LED-Yellow_0603_20mA_2V1
R4 56 Part Number: RS_504-7868 R_56R
D4 D-LED-GREEN DK_VLMG1300-GS08CT-ND D_VLMG1300_LED-Green_0603_20mA_2V0
LEDY
LEDG
SPI
Bk1B4855_SPI-FPC-MST_Interface.SchDoc
A_MODULE
MODULE
DGND
CS1
CS2
CS3
CS4
CS5
CS6
CS7
CS8
CS9
CS10
CS11
CS12
CS13
CS14
CS15
CS16
A_D6_A0
A_D7_A1
B_D6_A0
B_D7_A1
C_D6_A0
C_D7_A1
D_D6_A0

6.3.5 Mechanical

This schematic contains a mechanical component defined in library in which there are the six fixing holes connected together. Having a component defined in library make possible to reuse this fixing holes (and so the case that are built for) for others boards. In the following page is shown the schematic.

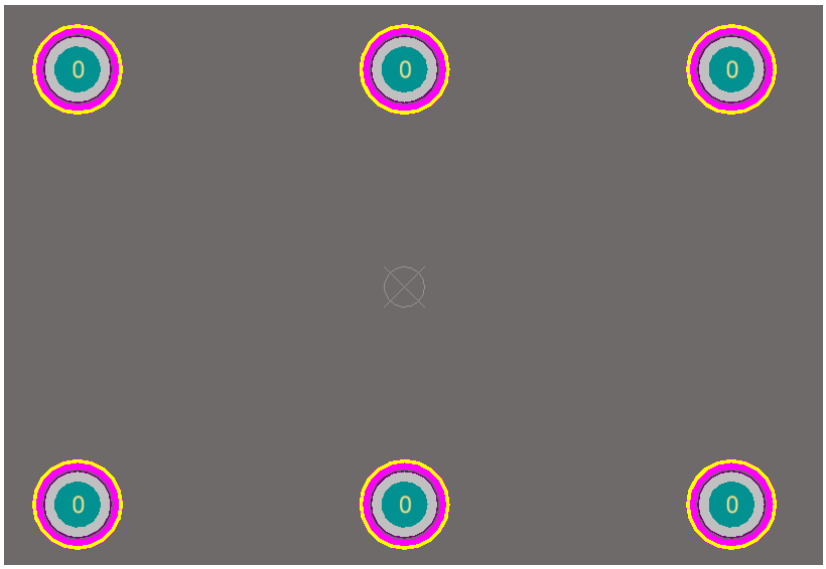
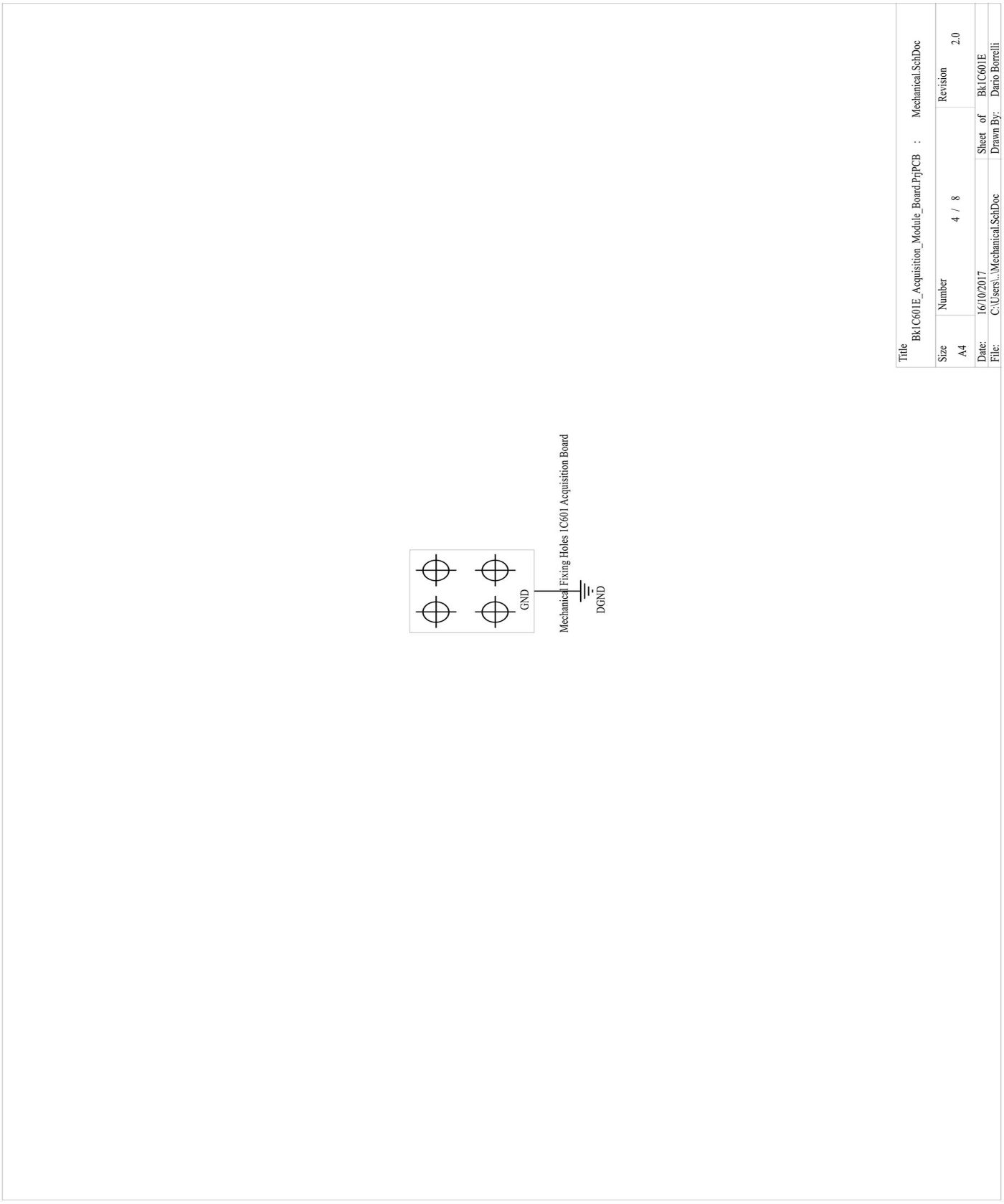


Figure 25: Mechanical component footprint


Title
Bk1C601E_Acquisition_Module_Board.PrjPCB : Mechanical.SchDoc
Size
A4
Number
4 / 8
Revision
2.0
Date:
16/10/2017
Sheet of
Bk1C601E
File:
C:\Users\..\Mechanical.SchDoc
Drawn By:
Dario Borrelli
GND
Mechanical Fixing Holes 1C601 Acquisition Board
DGND

6.4 PCB

The PCB has been realized considering the mechanical specification, in particular to have a smaller PCB has been necessary to use a 4 plane FR-4 with components placed on both side of PCB. The components that have to interact with user has been posed on top with also the higher components to have a board less thick on bottom (simplifying the fixing structure). The main rules used for this board are 6mil as clearance and wire thickness, and for Vias 0,7mm and 0,35mm for hole. The only wire thicker is the wire connect external power supply to the board internal power supply that is realized 0,5mm thick, to make it able to support over-current. The two internal plane are connected to internal power supply and to ground.

This two planes simplify a lot the routing procedure reducing from circa 200 connection to 60. Particular attention as been made to the communications part of the board, where for SPI-CAN and for the CAN transceiver the routing as been made manually only on the bottom of the board to ensure shorter wires and to avoid interferences with other switching part of the circuit (the 16 SPI wires are all placed on top). The CAN connector, the power supply switch and the CAN terminator switch are leaning out on the side of the board to make them usable cutting a hole on the side of the enclosure box.

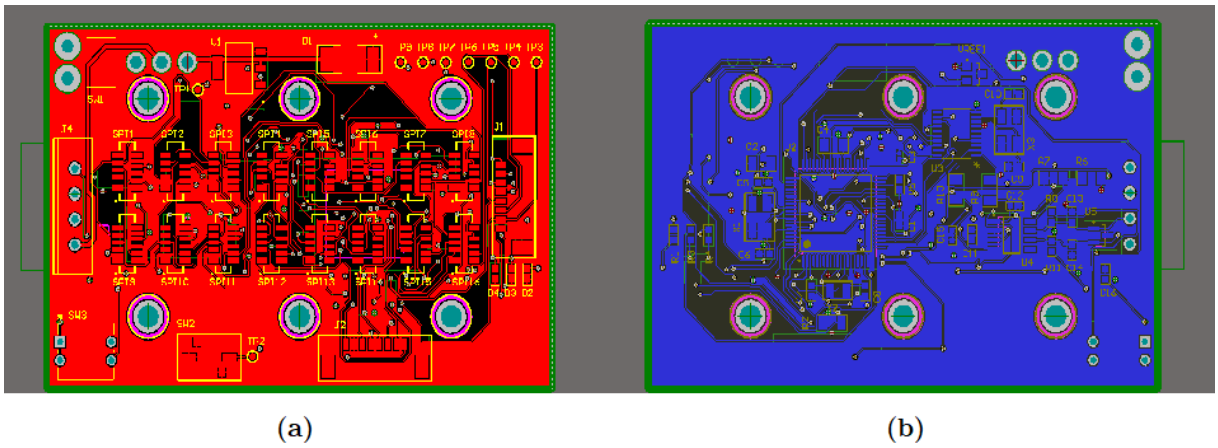
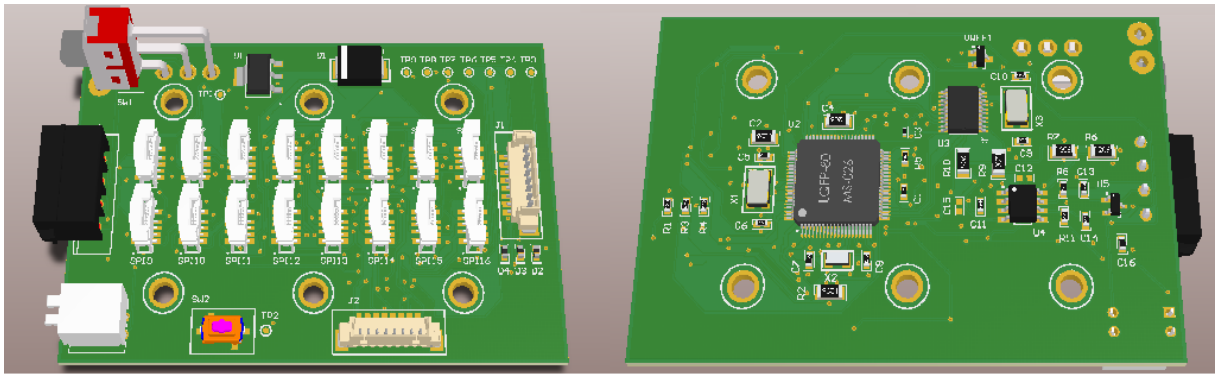


Figure 26: Bk1C601E Acquisition Module Board designed PCB, (a) top (b) bottom

To operate with till sixteen 1C601A board has been necessary to chose and find the space to insert sixteen FFC connector, to reduce the space needed on the board and to make it more easy to use has been chosen vertical mount connector. For all the components has been necessary to create symbols, footprints. For some components have a standard packages so has been necessary to create only the symbol and only link to the correct packages (checking on datasheet all the dimensions of pads and clearance between them), for other has been necessary to draw them and to find or build a specific 3d model to add to them. Also for this board all the common components are chosen considering which of them are already at Politecnico.
In figure 27 is shown the designed PCB 3d model, while in figure 28 is represented the final produced PCB.



(a) (b)

Figure 27: Bk1C601E Acquisition Module Board PCB 3d model, (a) top (b) bottom

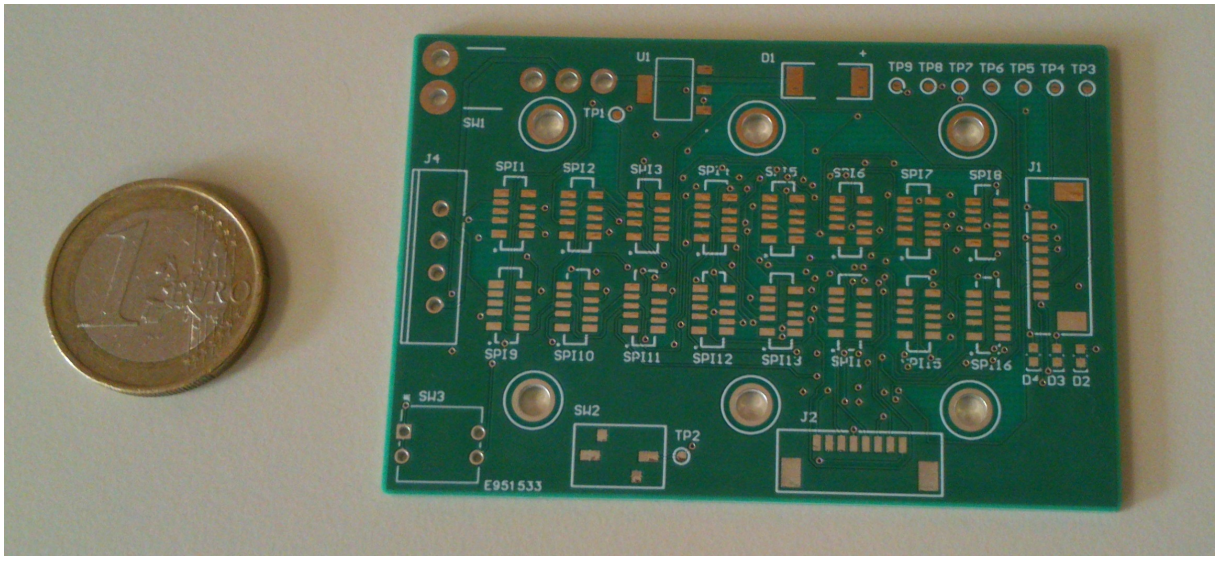


Figure 28: Bk1C601E Acquisition Module Board produced PCB

6.4.1 Electrical Characteristics

In the following table are presented the electrical characteristic of the 1C601E Acquisition Module Board. More information can be taken from components datasheet.

Parameter	**Min**	**Typical**	**Max**
Vcc Supply Voltage	2,2V	3,3V	3,6V
Supply Current (without 1c603A connected)			35,6mA
Supply Current (with 16 1c603A connected)			158mA
Temperature Range	-40°C		+85°C
Vin analog pins range	0V		Vcc
VOHmin	1,8V	2,70V	3V
VOLmax			0,6V
CAN_H	-4V		16V
CAN_L	-4V		16V

6.5 BK1C601E ACQUISITION MODULE BOX

To enclosure the Bk1C601E Acquisition Module Board has been already realized a box. This box is divided in three parts: a base, on which is mounted the electronics board, and two covers that with screws can be fixed to the base. In figure 29 can be seen the complete box designed.

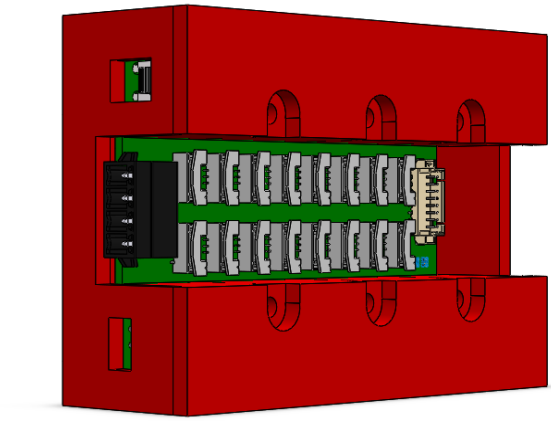


Figure 29: Bk1C601E Acquisition Module box

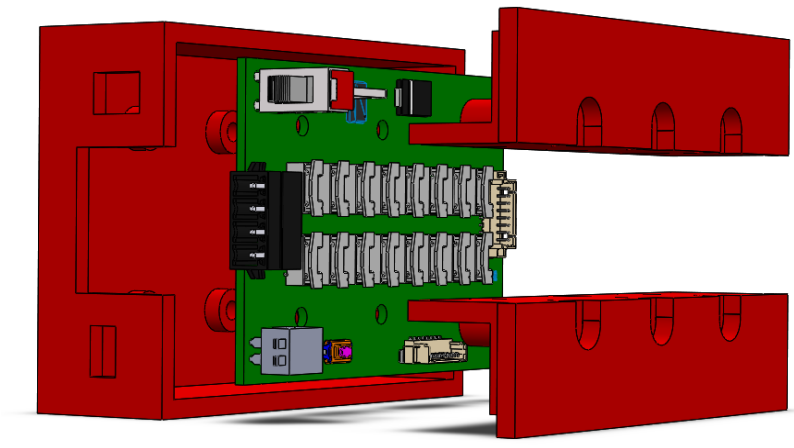


Figure 30: Bk1C601E Acquisition Module expanded view

7 CONCLUSIONS AND FUTURE WORKS

The designed system is a complete measurement system, however due to lack of time was not possible to develop the software for the Bk1C601E Acquisition Module Board and was not possible to test all the hardware created. This was manly due to the manufacturing time and component delivery time that not permits to test in time the board.

7.1 SOFTWARE BK1C601E

The software that have to be designed must implement the handling the measurement procedures, like the calibration procedure that have to perform for each 1C603A Multi-Sensor connected a measure in controlled environment elaborate the calibration values and store them in the 1C603A Multi-Sensor on board memory.
Have to be designed also the CAN protocol application level. In particular this level in CAN protocol is not standardized. For that reason is necessary to design the data field of the data frame. In particular is important to create a standard for the AraMiS project in order to make easily usable the 1B4853 CAN interface.

7.2 1C603A ELECTRICAL TEST

The first thing to do after soldering all components on the realized board, is to test if all the pins of every components are correctly connected and there isn't unwanted shorts between to adjacent pins. In particular, due to the small dimensions of the footprints, can be very difficult to correctly separate the central ground pad with pins. For that reason is very important to test the boards. Furthermore can be performed test on SPI communications with an oscilloscope in order to evaluate the presence of noises that can produce errors in SPI communications.

7.3 1C603A SOFTWARE TEST

The software realized for this board is used to send data and instruction to the IMU Sensor that handle all the aspect of the SPI communication and also all others components mounted on board.
The test can be realized also using a breakout board for the MPU9250, controlled by a developing board with the same processor (MSP430F5437) using a USB-JTAG debugger that permits to see real-time the registers value resulted by interaction of the two boards. In fact is important to check if the values generated by the 1C603A are compatible with measurement values.

7.4 BK1C601E ELECTRICAL TEST

After soldering all components on the realized board, the first thing to do is to test if all the pins of every components are correctly connected and there isn't unwanted shorts between to adjacent pins. Furthermore can be performed also test on SPI communications with an oscilloscope in order to evaluate the presence of noises that can produce errors in SPI communications, test on CAN communications to evaluate if the levels are noiseless and at the same time are comply with the CAN bus protocol. This is important to check the correct behaviour of the CAN transceiver. The supply over-voltage and under-voltage as not destructive protection, can be easily tester putting on the supply pin properly voltages value.

7.5 1B4853 CAN INTERFACE SOFTWARE TEST

The software realized handle the communication between the tile processor and the CAN controller. So all features implemented in the developed software must be tested to evaluate the correctly behaviour of the CAN controller. Also in this case can be used a breakout board for the MCP2515, controlled by a developing board with the same processor (MSP430F5437) using a USB-JTAG debugger that permits to see real-time the registers value resulted by interaction of the two boards.

APPENDIX A 1C603A

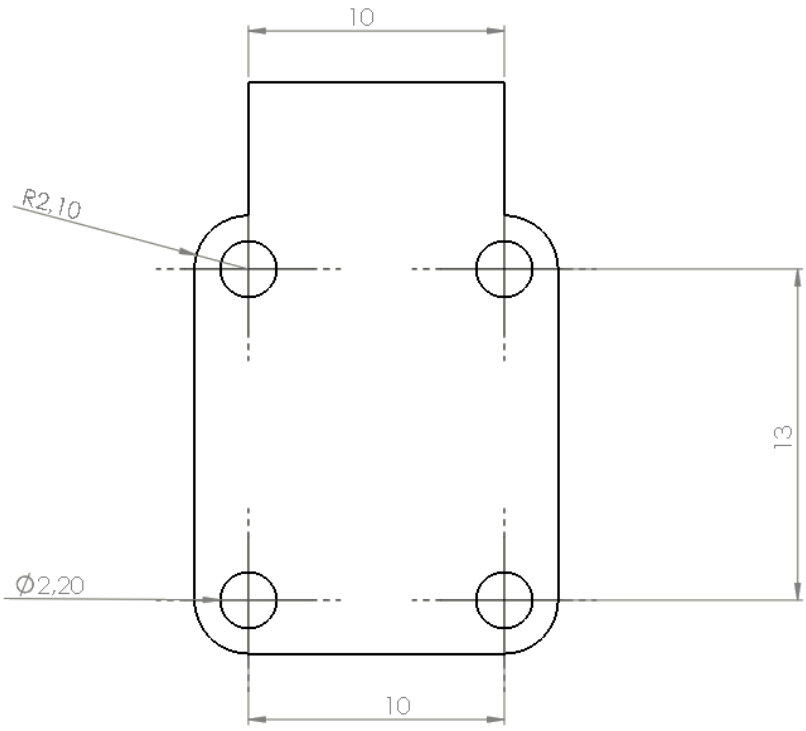
BOM

Quantity for 25 boards.

Quantity	Tot Qty	Tot Qty	Part Number	Part Labe	Part Name	Description
3	75	RS_616-9391	C_100n_0603_X7R_16	C_100n	Capacitor	C1, C2, C3
1	25	DK_WM11101CT-ND			Molex CONN FFC 6POS RIGHT ANGLE 0.50MM SMD - 0545480671	J1
3	75	DK_311-4.7KJRCT-ND	R_4k7_0402_63_5	R_4k7	Resistor	R1, R2, R3
1	25	DK_1428-1019-1-ND	MPU-9250		IMU GYRO/ACCEL/COMPASS/9-AXIS	U1
1	25	DK_MAX31725MTA+-ND	MAX31725MTA+		Digital Output Temperature Sensor	U2
1	25	DK_24AA01T-I/OTCT-ND			IC EEPROM 1KBIT 400KHZ SOT23-5	U3
1	25	DK_WM11405-ND			cavo FPC 8"	X

1C603A Dimensions

All measures are in millimetres.



1C603A Software

```
init(baudrate : unsigned long) : bool
{
byte temp;

SLOT::D0.init();
SLOT::D0.msbFirst();
SLOT::D0.enable(MSP_430::SPI_MASTER_MODE,baudrate);


for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(PWR_MGMT_1);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(0x80);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(CNTL2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(0x01);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
```

```
}
SLOT::D0.writeData(CONFIG);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(CONFIG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp && 0xC7);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(INT_PIN_CFG);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(INT_PIN_CFG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
```

```
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp && 0xFB);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(INT_ENABLE);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x00);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(USER_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x20);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_MST_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x0D);
```

```
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_1);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x01);
return true;
}

disable() : bool
{
for (unsigned long i=0;!SLOT::D0.isTXempty();i++){
    if (i>TIMEOUT)
        return false;
}
disable(SPI_MASTER_MODE);
return true;
}

readMemory(address : byte) : byte
{
byte data;
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV1_ADDR);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(0xD0);
```

```
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV1_REG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(address);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV1_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(0x81);


for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(EXT_SENS_DATA_03);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return 0;
}
data=SLOT::D0.readData();
return data;
}

writeMemory(address : byte, data : byte) : bool
```

```
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV1_ADDR);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x50);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV1_REG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(address);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV1_DO);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(data);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
```

```
}
SLOT::D0.writeData(I2C_SLV1_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x81);
return true;
}

enableGyroscope() : bool
{
byte temp;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp && 0xF8);
return true;
}
```

```
enableAccelerometer() : bool
{
byte temp;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp && 0xC7);
return true;
}

disableGyroscope() : bool
{
byte temp;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);
```

```
for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp || 0x07);
return true;
}

disableAccelerometer() : bool
{
byte temp;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return false;
}
temp=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
```

```
}
SLOT::D0.writeData(PWR_MGMT_2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(temp || 0x38);
return true;
}

configureAccelerometer(config_1 : byte, config_2 : byte) : bool
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(ACCEL_CONFIG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(config_1);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(ACCEL_CONFIG_2);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(config_2);

return true;
}
```

```
configureGyroscope(config : byte) : bool
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(GYRO_CONFIG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(config);
return true;
}

configureMagnetometer(config : byte) : bool
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(CNTL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(config);
return true;
}

configureThermometer(config : byte) : bool
{
//temperature sensor slave configuration
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV0_ADDR);
```

```
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0xC8);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV0_REG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x01);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV0_DO);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(config);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(I2C_SLV0_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
```

```
}
SLOT::D0.writeData(0x81);
return true;
}

getAccelerometer() : MEASURES
{
byte low,high;
MEASURES data;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_XOUT_H);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_XOUT_L);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
low=SLOT::D0.readData();
data.Xaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_YOUT_H);
```

```
for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_YOUT_L);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
low=SLOT::D0.readData();
data.Yaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_ZOUT_H);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(ACCEL_ZOUT_L);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
```

```
        return NULL;
}
low=SLOT::D0.readData();
data.Zaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);
return data;
}

getGyroscope() : MEASURES
{
byte low,high;
MEASURES data;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(GYRO_XOUT_H);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(GYRO_XOUT_L);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
low=SLOT::D0.readData();
data.Xaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
```

```
}
SLOT::D0.writeData(GYRO_YOUT_H);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(GYRO_YOUT_L);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
low=SLOT::D0.readData();
data.Yaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(GYRO_ZOUT_H);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(GYRO_ZOUT_L);
```

```
for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
low=SLOT::D0.readData();

data.Zaxes = ((unsigned int) high)<<8 || low;
data.Xaxes = high * 256 + low;
return data;
}

getMagnetometer() : MEASURES
{
byte low,high;
MEASURES data;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(HXH);
for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
SLOT::D0.writeData(HXL);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
       return NULL;
}
low=SLOT::D0.readData();
data.Xaxes= (unsigned int)((unsigned char) high<<8 || (unsigned char) low);
```

```
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(HYH);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(HYL);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
low=SLOT::D0.readData();
data.Yaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
SLOT::D0.writeData(HZH);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
high=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return NULL;
```

```
}
SLOT::D0.writeData(HZL);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
low=SLOT::D0.readData();
data.Zaxes = (unsigned int)((unsigned char) high<<8 || (unsigned char) low);
return data;
}
getThermometer() : MEASURES
{
byte low,high;
unsigned int data;

//temperature sensor slave configuration
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV0_ADDR);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(0x48);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV0_REG);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(0x00);
```

```
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(I2C_SLV0_CTRL);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(0x82);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}

SLOT::D0.writeData(EXT_SENS_DATA_00);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return 0;
}
low=SLOT::D0.readData();

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(EXT_SENS_DATA_01);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return 0;
}
high=SLOT::D0.readData();
data = (unsigned int)((unsigned char) high<<8 || (unsigned char) low)
return data;
}
```

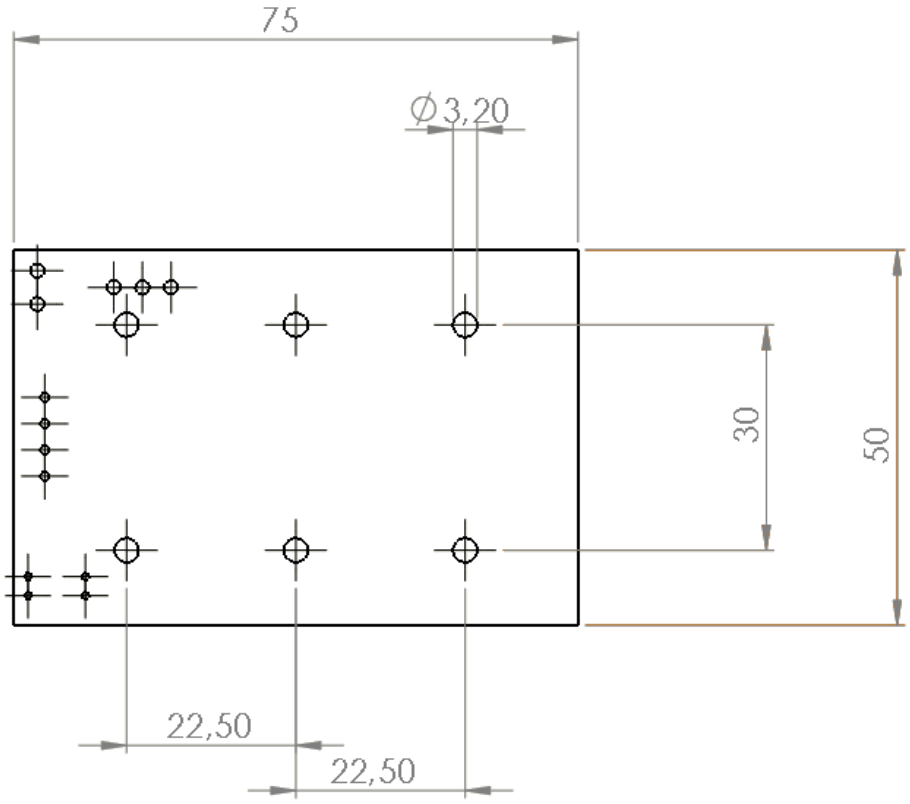
APPENDIX B BK1C601E

BOM

Quantity	LibRef	Part Label	Designator	Description
1	DK_277-9048-ND		J4	Phoenix Contact COMBICON MC Series
1	DK_296-11654-2-ND		U4	SMD CAN Transceiver
1	DK_296-20887-2-ND	D_LM4040_Zn_SOT23-3_15mA	VREF1	Zener Diode
2	DK_311-62GRTR-ND		R8, R11	SMD Resistor
1	DK_399-3525-6-ND	C_10u	C2	Capacitor
6	DK_445-1270-1-ND	C_12p, C_12p, C_12p_0603_C0	C5, C6, C7, C8, C9, C10	Capacitor
1	DK_445-2516-1-ND	C_470n	C4	Capacitor
1	DK_450-1355-ND	Dip_Switch_SPST_2_RightAngl	SW3	Dip Switch SPST 2 Position Through Hole, Ri
1	DK_490-1313-1-ND	C_10n	C3	Capacitor
2	DK_887-1699-2-ND		X1, X3	Crystal
1	DK_EG2480-ND	500SSP1S1M7QEA	SW1	Slide Switch SPDT Through Hole, Right Angle
1	DK_FZT651TR-ND	DK_FZT651TR-ND	U1	TRANS NPN 60V 3A SOT-223
1	DK_LSM115JE3/TR13CT-ND	LSM115JE3/TR13	D1	DIODE SCHOTTKY 15V 1A DO214BA
1	DK_MCP2515T-I/STCT-ND	MCP2515T-I/ST	U3	IC CAN CONTROLLER W/SPI 20TSSOP
1	DK_P47KGCT-ND	R_47K	R5	Resistor
1	DK_SM712-TPMSCT-ND	SM712-TP	U5	Anti-Parallel Zener 2x
1	DK_VLMG1300-GS08CT-ND	D_VLMG1300_LED-Green_0603	D4	LED Green
1	DK_VLMS1300-GS08CT-ND	D_VLMS1300_LED-Red_0603_2	D2	LED Red
1	DK_VLMY1300-GS08CT-ND	D_VLMY1300_LED-Yellow_0603	D3	LED Yellow
1	DK_WM7612CT-ND	J_8_MALE_SMT_PicoBlade_1A	J1	MOLEX PicoBlade
1	DK_WM7612CT-ND	J8_MALE_SMD_PicoBlade_1A	J2	MOLEX PicoBlade
16	DK_WM7982CT-ND	Easy-On_52559_6way_90°_0,5A	SPI1, SPI2, SPI3, SPI4, SPI5, SPI6,	MOLEX Easy-On™ 52559
1	DK_XC1967CT-ND		X2	Crystal
1	RS_223-1981	R_0R_1206_250_1	R2	Resistor
4	RS_223-1981	R_0R	R6, R7, R9, R10	Resistor
2	RS_464-6442	C_33p	C13, C14	Capacitor
3	RS_504-7868	R_56R	R1, R3, R4	Resistor
1	RS_534-5730	C_10n	C16	Capacitor
3	RS_616-9391, RS_264-4630, RS_616-9391	C_100n	C1, C11, C17	Capacitor
1	RS_917-3974		SW2	SMD Switch
1	XX_MSP430F5437IPNR	uP_MSP430F5437IPNR_LQFP80	U2	IC MCU 16BIT 256KB FLASH 80LQFP

Bk1C601E Dimensions

All measures are in millimetres.



BK1B4853CAN Software

```
Reset() : bool
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(RESET);

return true;
}

ReadRegister(address : byte) : byte
{
byte data;

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return  0;
}
SLOT::D0.writeData(READ);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return 0;
}
SLOT::D0.writeData(address);

for (unsigned long i=0;!SLOT::D0.isRXready();i++){
    if (i>TIMEOUT)
        return NULL;
}
data=SLOT::D0.readData();

return data;
}


WriteRegister(address : byte, data : byte) : bool
```

```
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(WRITE);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(address);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(data);

return true;
}

LoadTXBuffer(buffer : ushort, data : byte) : bool
{
if (buffer == 0)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x40);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(data);

return true;
}
```

```
if (buffer == 1)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(0x41);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(data);

return true;
}


if (buffer == 2)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(0x42);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(data);

return true;
}

if (buffer == 3)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
```

```
        return false;
}
SLOT::D0.writeData(0x43);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(data);

return true;
}

if (buffer == 4)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x44);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(data);

return true;
}

if (buffer == 5)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x45);

for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
```

```
        return false;
}
SLOT::D0.writeData(data);

return true;
}
return false;
}

ReadRXBuffer(buffer : ushort) : bool
{
if (buffer == 0)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x90);

return true;
}

if (buffer == 1)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
SLOT::D0.writeData(0x92);

return true;
}


if (buffer == 2)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
        return false;
}
```

```
SLOT::D0.writeData(0x94);

return true;
}

if (buffer == 3)
{
for (unsigned long i=0;!SLOT::D0.isTXready();i++){
    if (i>TIMEOUT)
       return false;
}
SLOT::D0.writeData(0x96);

return true;
}

return false;
}

ChangeMode(config : byte) : bool
{
return WriteRegister(CANCTRL, config);
}

ReadMode() : byte
{
return ReadRegister(CANSTAT);
}

SetID(Buffer : int, ID : unsigned short) : bool
{
bool var;

unsigned char msb = ID >> 3;
unsigned char lsb = (ID << 13) >> 8;
lsb= lsb && 0xE0;

switch (buffer) {

case 0 :
```

```
var=WriteRegister(TXB0SIDH,msb);
if (var==false)
break;
var=WriteRegister(TXB0SIDL,lsb);
break;

case 1 :
var=WriteRegister(TXB1SIDH,msb);
if (var==false)
break;
var=WriteRegister(TXB1SIDL,lsb);
break;

case 2 :
var=WriteRegister(TXB2SIDH,msb);
if (var==false)
break;
var=WriteRegister(TXB2SIDL,lsb);
break;

  default :
       return false;
}

return var;
}

SetData(Buffer : int, len : unsigned int, data : int[]) : bool
{
bool var;
unsigned char msb;
unsigned char lsb;

if (len> 8)
len=8;

lsb= lsb && 0xE0;

switch (buffer) {
```

```
case 0 :
var=WriteRegister(TXB0DLC,len);
if (var==false)
break;
for (int i=0;i<(len/2);i++){
msb = data[i] >> 8;
lsb = (data[i] << 8) >> 8;
switch (i) {

case 0 :
    var=WriteRegister(TXB0D0,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB0D1,msb);
    if (var==false)
    break;
break;

case 1 :
    var=WriteRegister(TXB0D0,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB0D1,msb);
    if (var==false)
    break;
break;

case 2 :
    var=WriteRegister(TXB0D2,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB0D3,msb);
    if (var==false)
    break;
break;

case 3 :
    var=WriteRegister(TXB0D4,lsb);
    if (var==false)
    break;
```

```
    var=WriteRegister(TXB0D5,msb);
    if (var==false)
    break;
break;

case 4 :
    var=WriteRegister(TXB0D6,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB0D7,msb);
    if (var==false)
    break;
break;

default:
break;
}
}
break;

case 1 :
var=WriteRegister(TXB1DLC,len);
if (var==false)
break;
for (int i=0;i<(len/2);i++){
msb = data[i] >> 8;
lsb = (data[i] << 8) >> 8;
switch (i) {

case 0 :
    var=WriteRegister(TXB1D0,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB1D1,msb);
    if (var==false)
    break;
break;

case 1 :
    var=WriteRegister(TXB1D0,lsb);
```

```
    if (var==false)
    break;
    var=WriteRegister(TXB1D1,msb);
    if (var==false)
    break;
break;

case 2 :
    var=WriteRegister(TXB1D2,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB1D3,msb);
    if (var==false)
    break;
break;

case 3 :
    var=WriteRegister(TXB1D4,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB1D5,msb);
    if (var==false)
    break;
break;

case 4 :
    var=WriteRegister(TXB1D6,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB1D7,msb);
    if (var==false)
    break;
break;

default:
break;
}
}
break;
```

```
case 2 :
var=WriteRegister(TXB2DLC,len);
if (var==false)
break;
for (int i=0;i<(len/2);i++){
msb = data[i] >> 8;
lsb = (data[i] << 8) >> 8;
switch (i) {

case 0 :
    var=WriteRegister(TXB2D0,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB2D1,msb);
    if (var==false)
    break;
break;

case 1 :
    var=WriteRegister(TXB2D0,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB2D1,msb);
    if (var==false)
    break;
break;

case 2 :
    var=WriteRegister(TXB2D2,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB2D3,msb);
    if (var==false)
    break;
break;

case 3 :
    var=WriteRegister(TXB2D4,lsb);
    if (var==false)
    break;
```

```
    var=WriteRegister(TXB2D5,msb);
    if (var==false)
    break;
break;

case 4 :
    var=WriteRegister(TXB2D6,lsb);
    if (var==false)
    break;
    var=WriteRegister(TXB2D7,msb);
    if (var==false)
    break;
break;

default:
break;
}
}
break;

return var;
}

SetPriorityAndStart(Buffer : int, priority : unsigned int) : bool
{
bool var;

if (priority > 3)
priority=3;

priority = priority & 0x8;

switch (buffer) {

case 0 :
var=WriteRegister(TXB0CTRL,priority);
break;

case 1 :
var=WriteRegister(TXB1CTRL,priority);
```

```
break;

case 2 :
var=WriteRegister(TXB2CTRL,priority);
break;

  default :
       return false;
}
}

CheckStatusMessage(Buffer : int) : bool
{
byte read;

switch (buffer) {

case 0 :
read=ReadRegister(TXB0CTRL);
if (read&0x8 != 0x8)
 return true;
break;

case 1 :
read=ReadRegister(TXB1CTRL);
if (read&0x8 != 0x8)
 return true;
break;

case 2 :
read=ReadRegister(TXB2CTRL);
if (read&0x8 != 0x8)
 return true;
break;

  default :
       return false;
}

return false;
```

```
}

SetMessageMask(Buffer : int, Mask : unsigned short) : bool
{
bool var;

unsigned char msb = Mask >> 3;
unsigned char lsb = (Mask << 13) >> 8;

switch (buffer) {

case 0 :
var=WriteRegister(RXM0SIDH,msb);
if (var==false)
break;
var=WriteRegister(RXM0SIDL,lsb);
break;

case 1 :
var=WriteRegister(RXM1SIDH,msb);
if (var==false)
break;
var=WriteRegister(RXM1SIDL,lsb);
break;

case 2 :
var=WriteRegister(RXM2SIDH,msb);
if (var==false)
break;
var=WriteRegister(RXM2SIDL,lsb);
break;

  default :
       return false;
}

return var;
}

ReadID(Buffer : int) : unsigned int
```

```
{
bool var;
unsigned char msb;
unsigned char lsb;
unsigned int;

switch (buffer) {

case 0 :
msb=ReadRegister(RXB0SIDH);
lsb=ReadRegister(RXB0SIDL);
break;

case 1 :
msb=ReadRegister(RXB1SIDH);
lsb=ReadRegister(RXB1SIDL);
break;


  default :
       return 0;
}
ID = (unsigned int)(msb<<8 ||  lsb)
return ID;
}

ReadData(Buffer : int, data : int[]) : bool
{
bool var;
int len;
unsigned char msb;
unsigned char lsb;

len= len && 0x7;

switch (buffer) {

case 0 :
len =readRegister(RXB0DLC);
for (int i=0;i<len;i++){
```

```
switch (i) {

case 0 :
data[0]=WriteRegister(RXB0D0);
break;

case 1 :
data[1]=WriteRegister(RXB0D1);
break;

case 2 :
data[2]=WriteRegister(RXB0D2);
break;

case 3 :
data[3]=WriteRegister(RXB0D3);
break;

case 4 :
data[4]=WriteRegister(RXB0D4);
break;

case 5 :
data[5]=WriteRegister(RXB0D5);
break;

case 6 :
data[6]=WriteRegister(RXB0D6);
break;

case 7 :
data[7]=WriteRegister(RXB0D7);
break;

default:
break;
}
}
break;
```

```
case 1 :
len =readRegister(RXB1DLC);
for (int i=0;i<len;i++){
switch (i) {

case 0 :
data[0]=WriteRegister(RXB1D0);
break;

case 1 :
data[1]=WriteRegister(RXB1D1);
break;

case 2 :
data[2]=WriteRegister(RXB1D2);
break;

case 3 :
data[3]=WriteRegister(RXB1D3);
break;

case 4 :
data[4]=WriteRegister(RXB1D4);
break;

case 5 :
data[5]=WriteRegister(RXB1D5);
break;

case 6 :
data[6]=WriteRegister(RXB1D6);
break;

case 7 :
data[7]=WriteRegister(RXB1D7);
break;

default:
break;
}
```

```
}
break;

case 2 :
len =readRegister(RXB2DLC);
for (int i=0;i<len;i++){
switch (i) {

case 0 :
data[0]=WriteRegister(RXB2D0);
break;

case 1 :
data[1]=WriteRegister(RXB2D1);
break;

case 2 :
data[2]=WriteRegister(RXB2D2);
break;

case 3 :
data[3]=WriteRegister(RXB2D3);
break;

case 4 :
data[4]=WriteRegister(RXB2D4);
break;

case 5 :
data[5]=WriteRegister(RXB2D5);
break;

case 6 :
data[6]=WriteRegister(RXB2D6);
break;

case 7 :
data[7]=WriteRegister(RXB2D7);
break;
```

```
default:
return false;
break;
}
}
break;
}

return true;
}
```

BIBLIOGRAPHY

(1) John Catsoulis, ***Designing Embedded Hardware***, O'Reilly Media (2005)

(2) M. Rizwan, De los Rios, Leonardo M. Reyneri, 1B48 ARAMIS Module Interface Control Document (2014)

(3) Marco Di Natale, Haibo Zeng, Paolo Giusto, Arkadeb Ghosal, ***Understanding and Using the Controller Area Network Communication Protocol Theory and Practice***, Springer Verlag, New York (2012)

(4) IEEE Standard Test Access Port: http://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf

(5) CAN Specification 2.0: http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf

(6) Datasheet MPU 9250: https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

(7) Datasheet MAX31725: http://datasheets.maximintegrated.com/en/ds/MAX31725.pdf

(8) Datasheet 24AA01T-I/OT: http://ww1.microchip.com/downloads/en/DeviceDoc/21711J.pdf

(9) Datasheet MSP430f5437: http://www.ti.com/lit/ds/symlink/msp430f5437.pdf

(10) Datasheet MCP2515: http://ww1.microchip.com/downloads/en/DeviceDoc/20001801H.pdf

(11) Datasheet SN65HVD230: http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf