Special Vehicle for Transporting Unstable Chemicals

By

Gong, Zhangxiaowen

Ma, Jun

Zhou, Wenjia

Final Report for ECE 445, Senior Design, Spring 2012

TA: Fortier, Justine

1 May 2012

Project No. 15

Abstract

This project is to design a special vehicle for transporting volatile substances. The purpose is to minimize the vibration and inclination of the vehicle when moving on an uneven surface.

The design actualizes the following functions: an omni-directional movement system that not only eliminates the centrifugal force when the vehicle turns but also makes the vehicle agile in compact lab environment, an auto leveling system that allows the vehicle to adapt uneven surface thus reduces vibration, and a real time wireless communication channel between the vehicle and an upper computer that provides features such as remote control and sensor data recording for further analysis.

Most functionalities described above have been fulfilled as intended. However, a torque issue impedes the performance of the auto leveling. As a result, a re-design for the mechanical system is scheduled in the future.

Contents

1 Introduction

1.1 Purpose and Goal

Our project is to design a special vehicle for transporting chemicals. The motivation came to us during a chemistry lab session. We noticed that when using a normal cart to carry chemicals, the carrier experienced significant bumps. Since volatile and hazardous chemicals should be treated carefully, we decided to address the issue by building a robot that performs safe and efficient transportation for those unstable chemicals. The major goal for the design is to keep the chassis level and stable; therefore, the vehicle has to limit the vibration and the tilt of the chassis while traveling on different road conditions.

1.2 Functions

The system has 4 major functions:

1. **Motion analysis:** The on-board processor acquires data from MEMS (MicroElectroMechanical Systems) sensors and calculates real-time tilt angle according to the readings. In order to eliminate noises, the raw data from the sensors have to be filtered.
2. **AL (Auto Leveling) system:** A worm motor based auto leveling system allows the vehicle to adapt uneven surface thus reduces vibration. In order to effectively reduce vibration, the suspension has to respond to uneven surfaces within 0.3s. The maximum dynamic tilt angle between the chassis and the horizontal surface also has to be constrained in 5 degree.
3. **ODM (Omni-Directional Movement) system:** A system built by servos and PID controlled motors not only eliminates the centrifugal force when the vehicle turns but also makes the vehicle agile in compact lab environment. All 4 wheels in the system should be adjusted to the same direction, rotate at the same speed, and modify their speed respectively when encountering uneven surface.
4. **Real time wireless communication** A wireless link between the vehicle and an upper computer provides features such as remote control and sensor data recording for further analysis.

1.3 Design Block

The project consists of seven modules. Figure 1 is the main block diagram of the system.

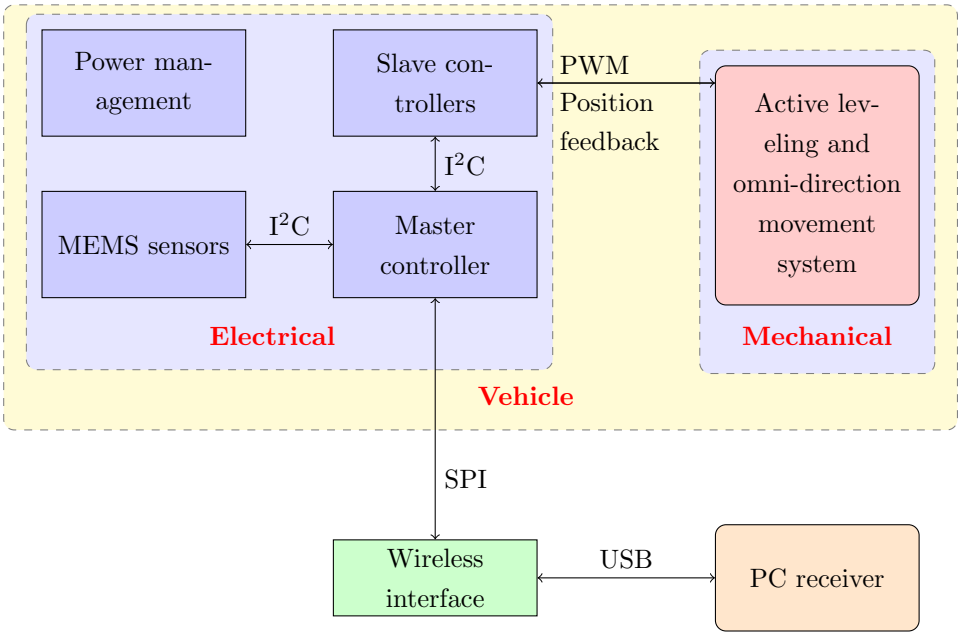


Figure 1: Main block diagram

1. **Master Controller:** The main controller is based on a 32-bit AVR microcontroller that performs various tasks including gathering data from MEMS sensors, making decision to balance the chassis, printing debug information on an OLED (Organic Light Emitting Diode) display, and communicating with a PC via wireless.
2. **Slave Controllers:** The side controllers is based on 8-bit AVR microcontrollers. Each MCU (Micro-Controlling Unit) receives commands from the main controller and controls 1 servo, 1 regular DC motor, and 1 worm motor.
3. **MEMS sensors:** An InvenSense MPU-6050 3-axis gyroscope and 3-axis accelerometer measures the state of the chassis. It performs 6-axis sensor fusion inside the chip and sends the results to the main controller in integer quaternions.
4. **Power Management:** The vehicle is powered by a 3-cell 11.1V 2200mAh LiPo battery. The on-board step-down and step-up regulators then provide 3.3V, 5V, and 13.8V power for different uses. The power management scheme for the model vehicle is designed to support the whole system for 1 hour of continuous operation on heavy load, which is enough for demo purpose.
5. **Mechanical system:** It includes the AL system and the ODM system. They take control signals from the slave controllers and deliver feedback for closed-loop control.
6. **Wireless Interface:** In order to analyze the system performance, the vehicle constantly send the sensor readings to a host computer. The wireless link for the transmission is built upon nRF24L01P 2.4GHz RF solution. The MCUs can send commands and packages to the RF chip via SPI (Serial Peripheral Interface).
7. **PC Receiver:** An 8-bit AVR microcontroller that has a built-in USB controller handles the computer side RF chip. In addition, a computer software is developed to visualize the sensor readings, to store/playback the data, and to remotely control the vehicle.

2 Design

2.1 Design Procedure

2.1.1 Mechanical System

The mechanical system consists of two parts – an ODM system and an AL system.

1. Traditionally, a typical ODM system is built up by mounting $3 \sim 4$ omni-wheels onto the chassis with $120°$ or $90°$ between each adjacent pair respectively. Since each omni-wheel has multiple passive wheels that allow it to move along the axis perpendicular to its active rolling direction[1], a vector calculation is able to yield the velocity of each omni-wheel that moves the vehicle towards any direction without rotating the chassis. However, the design introduces the following problems:
   (a) The passive wheels cause the omni-wheel having a non-perfect-round shape, which vibrates when rolling.
   (b) The small size of the passive wheels hinders the vehicle when encountering obstacles.
   (c) Since the passive wheels can rotate freely, an external horizontal force exerting on the chassis may result the vehicle to slide.

   As a result, we adopted a novel design that adjusts the rotation direction of each of the four regular wheels independently. For a full-size final product, the mechanism should be done by closed-loop controlled motor; nonetheless, we applied RC servos on the model system due to size issue.
2. The AL system is responsible for keeping the chassis level. Therefore, the height of each wheel has to be altered autonomously. To actualize the function, a closed-loop controlled worm motor with a $15kg \cdot cm$ torque is attached to each driving wheel on the model system. However, a hydraulic system is preferred for a real product.

Moreover, both the ODM system and the AL system follow the control scheme described in Figure 2. They provide rotational encoder or potentiometer feedback for speed or position PID control respectively, and the saturation filter confines the PID output in a reasonable range.

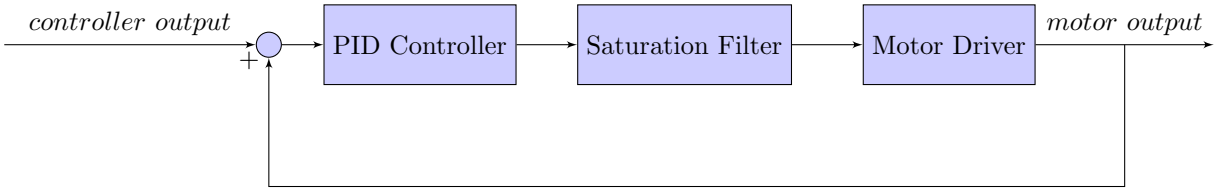


Figure 2: Control scheme for the ODM system and the AL system

2.1.2 Electrical/Electronic System

1. Since the on-vehicle system performs various functions ranging from analyzing sensor data to controlling motors, we adopted a master-slave structure multiple microcontroller approach. Since the master controller is required to execute multiple independent tasks simultaneously, a 32-bit microcontroller is preferred. Considering that most of the data analysis work in this design is performed upon floating point data, we eventually chose AT32UC3C2256C – an MCU with hardware FPU (floating point unit) – to fill the row. On the other hand, the slave controllers mainly deal with the actuators;

hence, a microcontroller model with high anti-noise ability is needed. Fortunately, the 8-bit AVR microcontroller family, which provides up to 40mA current output ability per I/O (Input/Output) pin and works at control-friendly 5V supply voltage, fulfills the requirements. Because the functions of the slave controller are relatively simple, we eventually chose a low-end model – ATmega8A.

2. MEMS sensor is an essential part of the design. We initially planed to obtain data from individual gyroscope, accelerometer, and magnetometer, so we can then perform a software sensor fusion based on Kalman filter to reduce noise. Nevertheless, we eventually decided to use an ASIC (Application-Specific Integrated Circuit) MPU-6050 that integrates a 3-axis gyroscope and a 3-axis accelerometer. The most valuable feature of MPU-6050 is that it applies a complementary filter[2] within the chip and outputs quaternion directly, which significantly frees the MCU's load. In addition, the single chip sensor also reduces both cost and power consumption. After test, we confirmed that the complementary filter is able to produce results with similar quality with that from the Kalman filter.
3. The rated voltage and current of the worm motor that we use on the model system are 12V and 1A respectively, so a typical integrated H-bridge is enough for driving it. However, since the design should later move towards a full-size vehicle, we still decided to make discrete H-bridges. The NMOS we adopted on the H-bridge can drive a motor with up to 55V and 110A rated voltage/current[3], and test results showed that the PCB traces with thick solder on them can hold 20A safely, which is enough for a big motor.
4. Although the output voltage of the 3-cell Li-Po battery used on the model system is within the V_{GS} range of the NMOS on the H-bridge, we use a step-up regulator to convert a regulated 5.0V voltage to 13.8V for the V_{GS}. The reason is also for compatibility with future designs. The current power scheme allows any voltage from 7V to 40V to power up the system.
5. The choices for wireless solution is wide; available ones includes Bluetooth, ZigBee, etc. Since the communication range of even a class 1 Blootooth transceiver is still quite short (about 100m)[4], we quickly eliminated it from the choice list. The ZigBee standard gives a relatively high effective range; however, on the unlicensed 2.4GHz carrier frequency, ZigBee only provides a 250Kbps data rate[5]. Because our project may introduce telepresence feature in the future, which demands a high band-width for real-time video transmission, we did not adopt ZigBee. Finally, we managed to find a solution with both qualified range and band-width. The nRF24L01P 2.4GHz wireless module that we finally chose integrates a PA (Power Amplification) circuit for transmitting and a LNA (Low Noise Amplification) circuit for receiving; consequently, a pair of such module can communicate in 500m at 2Mbps data rate. Furthermore, its 10Mbps SPI (Serial Peripheral Interface) bus allow high speed data exchange with the microcontrollers[6], which enhances the overall performance of the system.
6. In order to control the wireless interface on a computer, we need to set up a link between the RF chip and a computer USB (Universal Serial Bus) port. A common way is to use a FTDI chip that extends a virtual serial port on the computer and reconstruct the USB packages into UART (Universal Asynchronous Receiver/Transmitter) packages that are supported by most MCUs. Nonetheless, adding such chip not only raise the cost and power consumption but also limits the USB functionalities. As a result, we picked an AVR 8-bit MCU with built-in USB 2.0 device controller instead. To reduce the complexity of the computer side USB driver, we designed the MCU side USB driver following a CDC (Communication Device Class) pattern, which essentially emulates a virtual serial port. However, different from the FTDI approach, the driver does not really translate packages between USB and UART; therefore, the data rate is not limited by the serial port baud rate setting at all, thus ensures

the efficiency of the transmission.

2.2 Design Details

2.2.1 Schematics

The main controller uses AT32UC3C2256 as processor. The communication modules that are utilized in the design include I^2C (Inter-Integrated Circuit, a.k.a TWI, Two-Wire Interface), SPI, and USB. We also connect the external interrupt pins of the MCU to other modules such as the RF transceiver and the MEMS sensor. Figure 3 is the core circuitry that supports the MCU, including oscillator, power filtering circuit, USB interface for bootloader, reset circuit, etc.

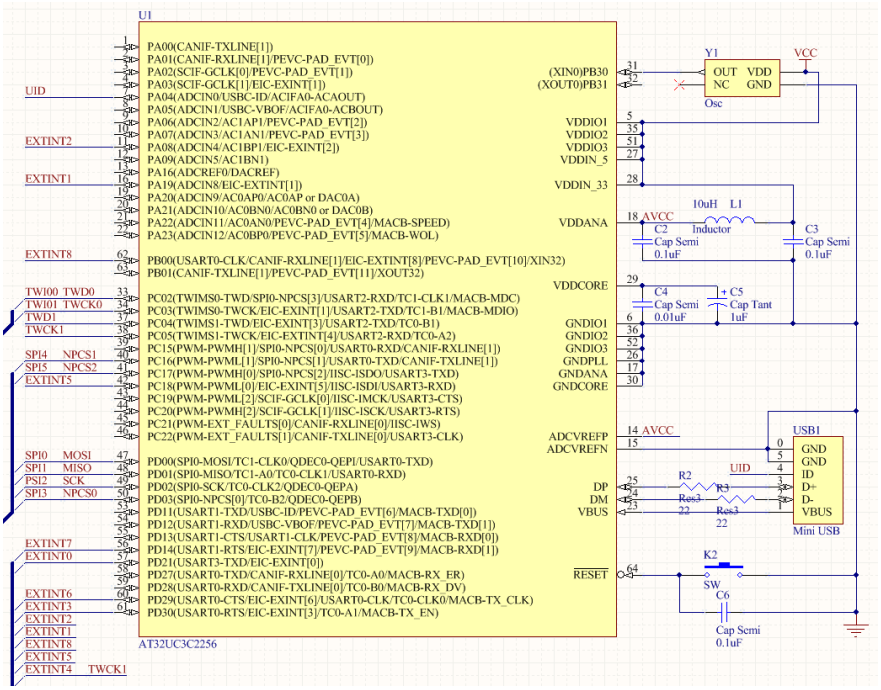


Figure 3: AT32UC3C2256 core system

Since the main MCU runs at 3.3V while the side ones work at 5V, a bidirectional level converter is required to establish the I^2C bus between them. We adopt a design that uses N-channel MOSFETs in a symmetric manner[7], which is presented in Figure 4. In the figure, VCC is 3.3V. Label TWD1 and TWCK1 belong to the 3.3V bus, and label TWI10 and TWI11 are the 5V ones. The circuit in the figure looks not symmetric; in fact, the 3.3V lines do have pull-up resistors although they are not shown in the figure due to being inside a resistor pack.

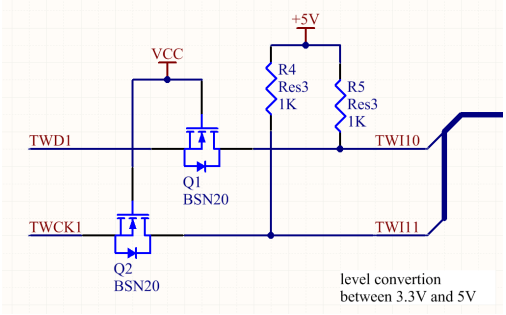


Figure 4: Bidirectional level shifter

The power management part on the main controller provides 3 distinct positive voltage levels: 5V, 3.3V, and 13.8V. The system first regulates the approximately 11.1V supply voltage to 5V and then step it up to 12V as well as down to 3.3V. The 5V regulator we use is LM2596, a highly efficient switching converter that works at 150KHz and has 3A current output capacity[8]. It supplies power for both generating the other two voltages and serving the side MCUs. Figure 5 includes its work state switch and output filtering circuit.

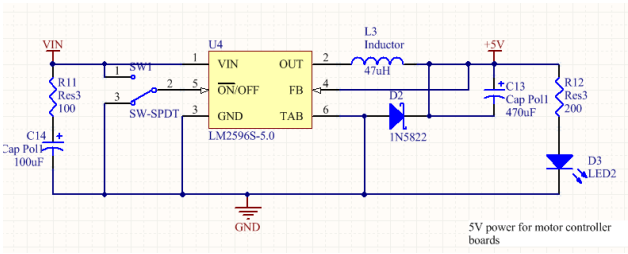


Figure 5: 5V regulator

We use a MAX629 switching step-up regulator for 13.8V supply. The chip needs two resistors to designate the output voltage from its internal 1.25V reference, $R_1 = R_2 \times (\frac{V_{out}}{V_{ref}} - 1)$[9] With 47KΩ as R_1 and 4.7KΩ as R_2, the resulting output voltage becomes 13.8V. Figure 6 shows the circuit.

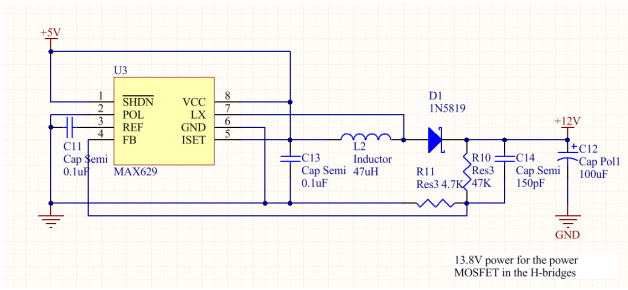


Figure 6: 13.8V regulator

Instead of using a switching regulator, we use LM1117, a linear regulator with 500mA maximum output current, for stepping 5V down to 3.3V. The reasons for this choice are:

1. The 3.3V system consists of the 32-bit MCU, 2.4GHz wireless transceiver, MEMS sensor, and the logic supply for OLED display, which draw a small total current.
2. The voltage drop from 5V to 3.3V is only 1.7V. Suppose the average current consumption by the 3.3V system is 100mA, the average dissipation power caused by the voltage drop is $1.7V \times 100mA = 170mW$, which is acceptable.
3. The linear regulator requires much less external components that a switching one does.

Figure 7 presents the circuit.

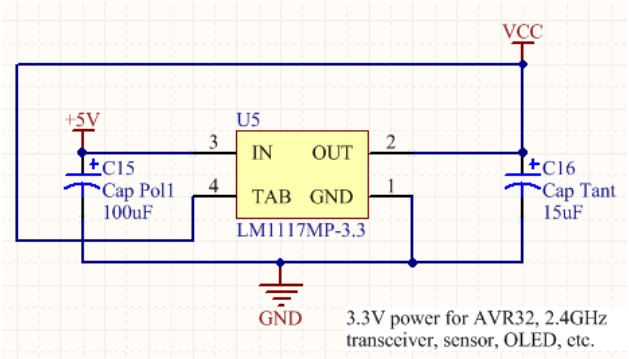


Figure 7: 3.3V regulator

Since we are using a module integrating the nRF24L01P wireless transceiver, the interface with the wireless module is just a header. The OLED display takes two voltage supplies: 5V for biasing the display and 3.3V for powering the logics. It takes data from the main MCU's SPI interface. Figure 8 shows the circuit.

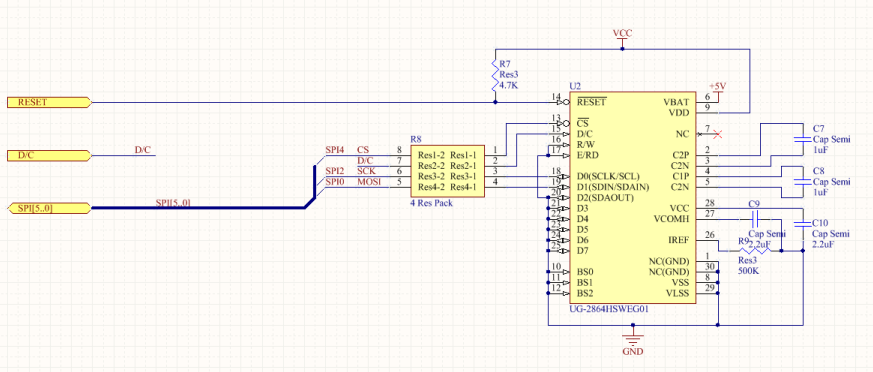


Figure 8: OLED display

Initially we planed to perform a 9-axis sensor fusion by attaching a 3-axis magnetometer HMC5883L to the MPU-6050 chip as an auxiliary. The magnetometer is useful for compensating the drift on the gyroscope's yaw output by taking reference to the Earth's magnetic field. However, later we found that the yaw data are not necessary for the system. As a result, we cut the magnetometer off from the design. Nonetheless, its place is still kept on the schematic as presented in Figure 9.

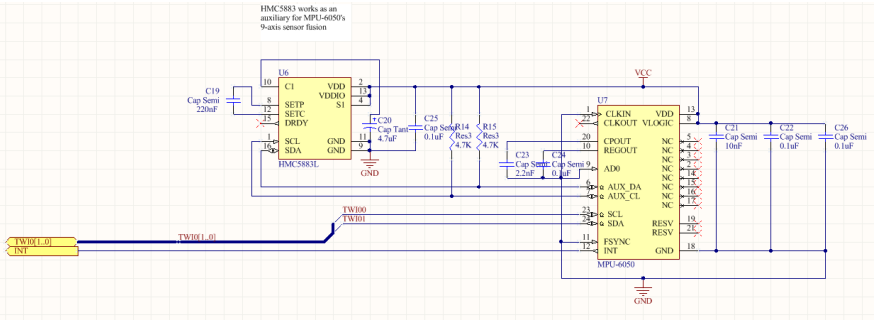


Figure 9: MEMS sensors

The side controllers use ATmega8A as processors. Besides GPIO (General Purpose Input/Output), we utilize the MCU's PWM (Pulse-Width Modulation) generator, external interrupt, and I^2C interface. Figure 10 is the core system, including oscillator, power filtering circuit, reset circuit, etc. Because we have totally 4 identical side controllers in the system, in order to apply consistent programs to every chip, we connect a 4-bit DIP switch to the MCU's GPIO so that each chip can determine the low 4-bit of its 7-bit I^2C slave upon booting up. The microcontroller is powered up by the 5V that comes from the main controller. However, each side controller still embeds a LM2596 regulator. The extra 3A current capability provided by the regulator goes to the 5V DC motor that drives the wheel.

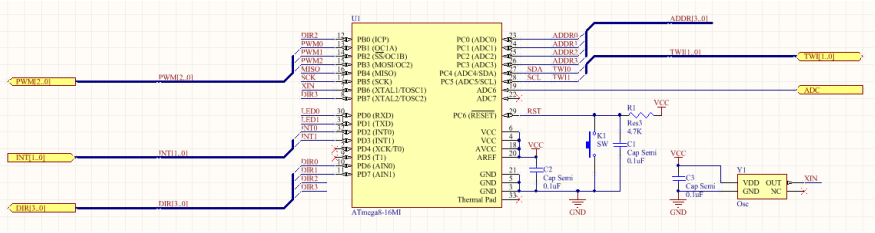


Figure 10: ATmega8A core system

Figure 11 is the design of the H-bridge for driving the worm motors. In the circuit, we use two IR2104 half-bridge chips to provide a 520 nanosecond deadtime to prevent instant short circuit when switching the direction of the motor. Besides, we also have other circuitry consists of Schottky diodes and resistors that protects the chips from reverse EMF (ElectroMotive Force). Note that IR2104 requires N-channel MOSFET instead of typical P-channel ones for the upper arms in the H-bridge. In order to provide the floating V_{GS} for the NMOS in the upper arms, a bootstrap circuit built from a diode, a Tantalum capacitor, and a charging resistor, is demanded. In order to charge the capacitor in the bootstrap circuit, we need a certain time of low level in the driving signal. As a result, the PWM signal provided to the H-bridge cannot reach a 100% duty cycle; instead, the peak duty cycle is limited at about 95%. The logics on the lower left side of the figure are responsible to translate the direction and PWM signals into logics that IR2104 accepts. Thanks to the logics, we are able to generate forward, backward, and breaking commands from two direction inputs

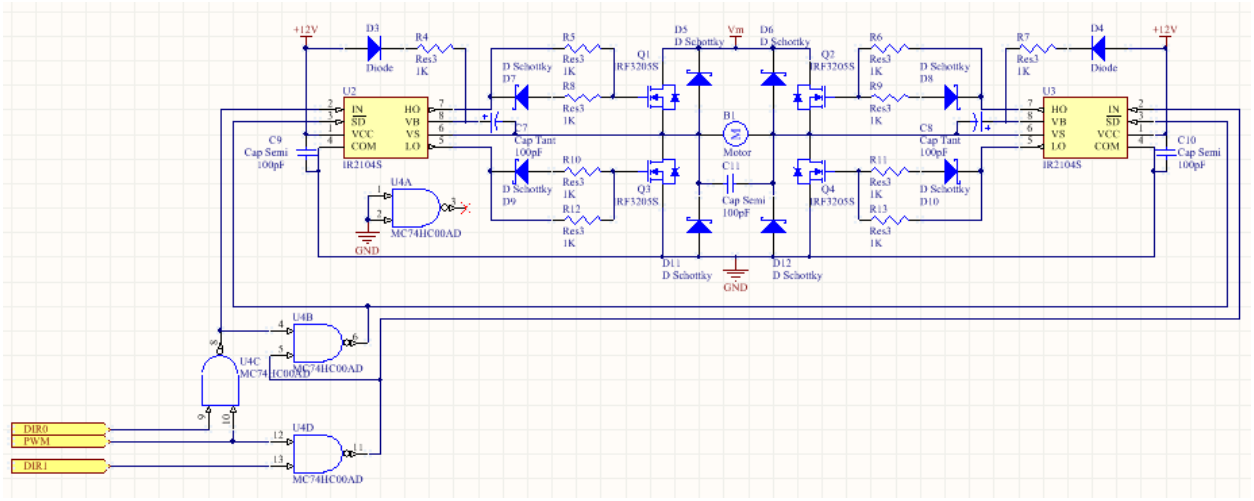


Figure 11: H-bridge

The MCU used on the PC side receiver is ATmega32U4. We use two communication modules provided by the chip: USB for PC communication and SPI for interfacing with the wireless chip. Figure 12 is the core system of the MCU. Its USB interface is also utilized by a built-in bootloader for programming the chip. The wireless module on the receiver board is identical to that on the main controller. The receiver board also has an LM1117 regulator for powering the wireless module up.

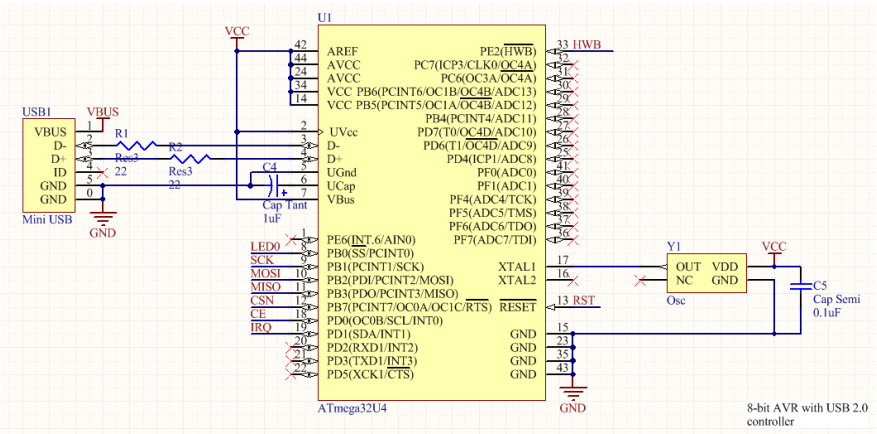


Figure 12: ATmega32U4 core system

2.2.2 Software

Because the master controller performs a number of independent tasks, we build the software upon an RTOS (Real-Time Operating System) to coordinate all the functions. The pre-emptive scheduling provided by the RTOS ensures high priority tasks to be served in real-time. Table 1 shows a list of tasks that are scheduled.

Table 1: RTOS tasks on the master controller

Task function	Priority	Wake up period
AL system algorithm	highest	50ms
Update sensor reading	high	10ms
Refresh OLED display	low	100ms
Process received command and coordinate the ODM system	low	100ms
Send sensor data over wireless	low	50ms
System idle task that cleans up dynamic memory and analyzes CPU usage	lowest	N/A

Among the tasks, the AL system algorithm has the highest priority. It's general flowchart is described in Figure 13.

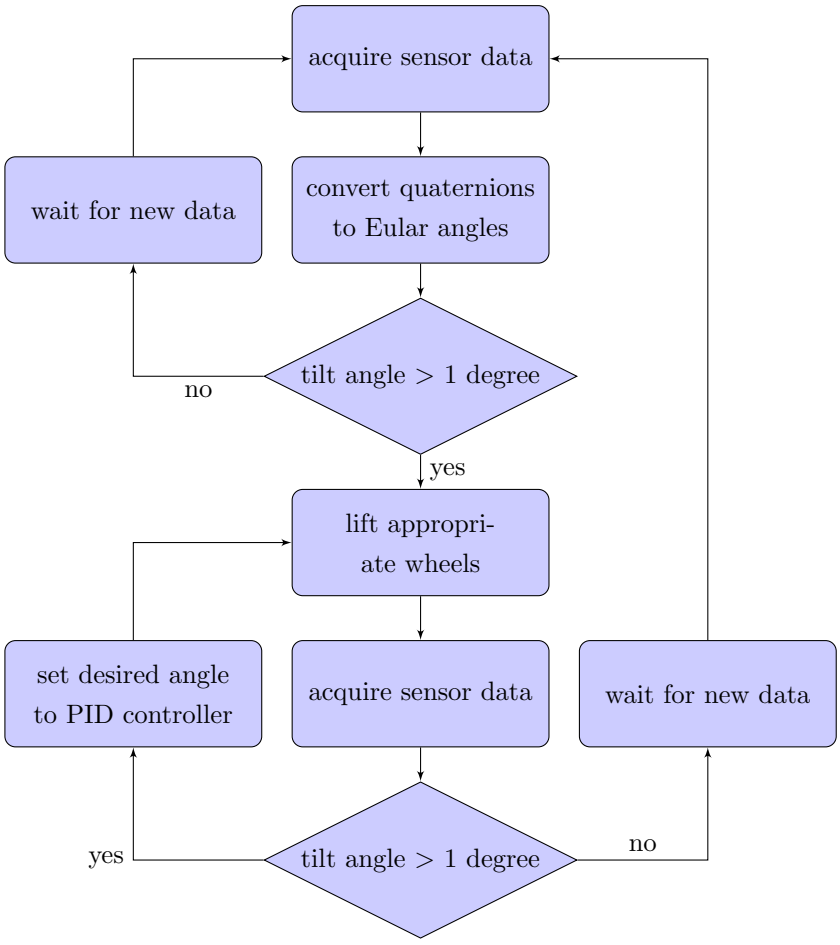


Figure 13: General AL system algorithm

Although the ODM system has a low priority, it's still one of the most essential components of the system. Figure 14 is the mounting coordinates of the servos in the system. The shaded parts in the graph are the effective angles that the servos can reach, and the numbers at the end points of each half-circle are the position commands sent to the servo for reaching those angles. In order to keep all the wheels parallel to each other, a coordinate translation is required for all servos. Imagine that the positive direction on the y axis in the graph stands for 0° in the whole system; hence, the positive direction on the x axis is 90°, and the negative direction on the y axis is −90°. An example of the coordinate translation is also illustrated in the figure. For the upper right servo, the system coordinate 0° is reinterpreted as 45° in the servo coordinate. Nonetheless, since each servo can only rotate within 180°, the system 180°, which is the negative direction on the y axis, cannot be directly translated into the upper right servo's coordinate. As a result, we can still map it to 45° in the servo coordinate and toggle the rotational direction of the corresponding driving wheel. Any other angle is also able to be translated in the same manner. Therefore, a pseudo 360° coordinates mapping is done. Table 2 is the complete mapping strategy for all servos. A negative sign at the front stands for a toggled direction is demanded for the corresponding driving wheel, and the position command sent to the servo is actually the value within the parentheses. If the calculated result exceeds the upper or lower limits (±180°), the final command should be decreased or increased by 360° respectively.

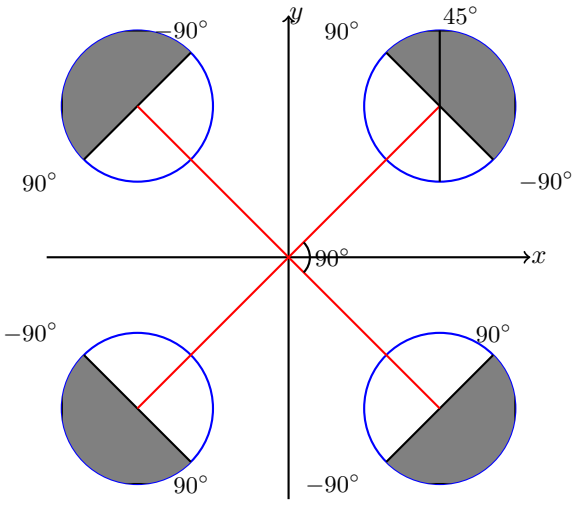


Figure 14: Coordinates of the ODM system

Table 2: Servo coordinates mapping strategy

System angle θ	Upper left	Upper right	Lower left	Lower right
$[-45, 135)$	N/A	$45 - \theta$	$-(45 - \theta)$	N/A
$(-180, -45) \cup [135, 180]$	N/A	$-(-\theta - 135)$	$-\theta - 135$	N/A
$[-135, 45)$	$-\theta - 45$	N/A	N/A	$-(-\theta - 45)$
$(-180, -135) \cup [45, 180]$	$-(-\theta + 135)$	N/A	N/A	$-\theta + 135$

In order to achieve maximum performance, all the peripheral and communication module drivers are designed with non-blocking algorithms.

1. The I^2C driver is optimized for RTOS, which means it suspends the calling task when a block read/write is required and resumes the task after the transaction finishes. This mechanism allows other task to obtain the CPU resource when a task is waiting for a transaction. The driver also utilizes a DMA (Direct Memory Access) channel that significantly reduces the CPU load. Furthermore, the fully automatic streaming algorithm hides the low level interrupt implementation from the RTOS, fulfilling an OOP (Object Oriented Programming) pattern.
2. The SPI driver also uses a DMA channel as well as a streaming scheme. Besides, the driver packs the operation mode of the target device into each packet. For instance, the RF chip works at 10MHz while the OLED is as fast as 33MHz. Therefore, the SPI hardware is able to switch among different modes autonomously. Different from the I^2C driver, the SPI one does not do the RTOS trick. The reason is that the SPI bus works at high speed comparing to the 400KHz I^2C bus; thus, frequent context switching among tasks may introduce additional overhead. Hence, whether or not to suspend the calling task has a performance trade off.

The wireless transceiver that we introduce to the project – nRF24L01P – uses a built-in protocol called Enhanced ShockBurst^{TM} to perform handshaking and error checking so that the MCU can save computing power for other tasks. The driver for the transceiver embraces the same design idea with the lower level drivers described above. The driver takes buffers from user and streams them out in a FIFO (First In First Out) manner. Since the Enhanced ShckBurst^{TM} protocol allows variable length package with maximum length at 32 bytes[6], if the buffer given by a user has content larger than 32 bytes, the API for appending package will first recurse to divide the buffer into multiple packages and then add them into the send queue. The transceiver has two work modes: PTX (Primary Transmitter) and PRX (Primary Receiver), our strategy is to configure it as PRX when it is idle, and only switch it to PTX when the send queue has packages waiting to be streamed out. The transceiver has 3 maskable interrupt sources, and uses a pulse on the IRQ pin to report an interrupt. The sources are: RX_DR (RX data ready), TX_DS (TX data sent), and MAX_RT (TX maximum retry reached). The driver makes use of all of them, and to handle them as follow:

1. RX_DR:
   (a) Read the length of the incoming package
   (b) Allocate a buffer that fits the length
   (c) If the receive queue has too many packages waiting for the user to fetch, then remove and free the head from the receive queue
   (d) Add the incoming package to the tail of the receive queue
2. TX_DS:
   (a) Check if more packages are in the send queue. If so, continue to send the next package
   (b) If not, go to PRX mode
3. MAX_RT, this interrupt is triggered when the transmitter loses connection with the receiver, so we handle it as:
   (a) Flush the TX FIFO in the transceiver
   (b) Rewrite the last package into the TX payload

In contrast to the master controller, the software for the slave ones is relatively simple. It adopts a traditional foreground-background design, which has an background infinite loop for executing commands and multiple interrupt service routines as foreground events. As an I^2C slave device, the slave controllers follow the I^2C register convention. Table 3 is the list of commands that the controller accept.

Table 3: I^2C slave commands

Command number	Read/write	Function
0x01	W	Set the PWM output of the driving wheel
0x02	W	Set the PWM output of the worm motor
0x03	W	Set the position of the servo
0x04	W	Set the calibration data to the servo
0x05	W	Set the threshold of the saturation filter
0x06	W	Break the worm motor
0x07	W	Break the driving wheel
0x08	W	Give the PID set point to the driving wheel
0xFE	R	Read the speed of the driving wheel
0xFF	R	Read the position of the worm motor

When the slave controller receives a command that sets the PID destination of the driving wheel (command 0x08), it loops to execute the control strategy demonstrated in Figure 15 until the error between the set point and feedback is reduced to a certain range. By tuning the PID constant K_p, K_i, K_d in the figure, the control scheme finally reaches a 300ms settling time. With this control system, the ODM system is able to output the same speed to all its 4 wheels, which successfully maintains the heading direction of the whole vehicle stable.

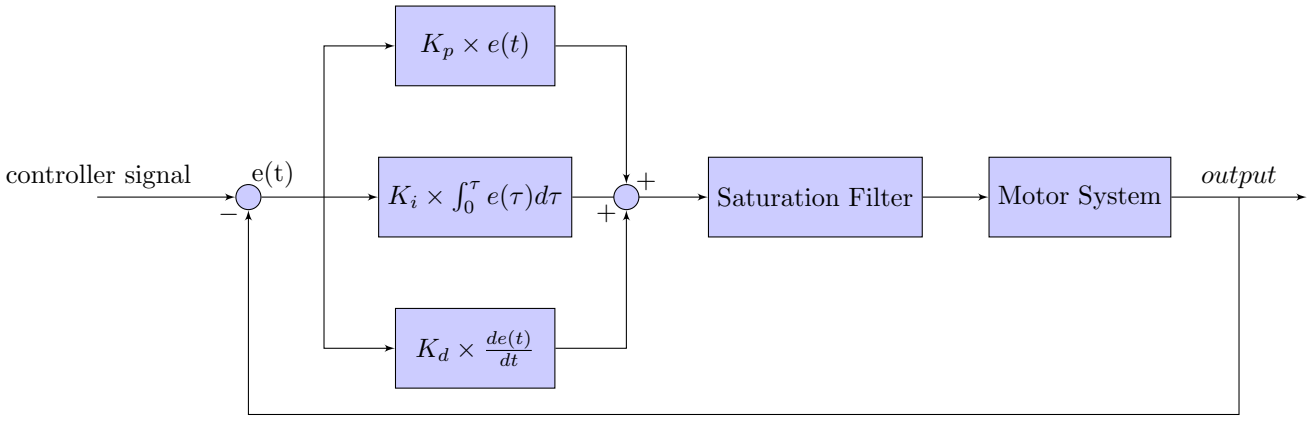


Figure 15: Motor control strategy

In order to pair with the main controller, the receiver on the PC side also uses an nRF24L01P. Although the driver for it is also based on stream, it handles the modes differently. Unlike the driver on the AVR32, which switches to PTX for sending packages, the driver on the receiver side always stays at PRX. When there are packages ready to send, it utilizes the acknowledgment package specified in the Enhanced ShckBurst^{TM} protocol to send them[6]. An acknowledgment package is the same with a normal TX package in term of length; however, it is sent passively when the receiver is acknowledging the transmitter. Therefore, it is never sent until the link is established between the transmitter and the receiver; thus, the MAX_RT interrupt will never happen under this configuration.

The USB controller in ATmega32U4 has a control endpoint (EP0) with a bank up to 64 byte and 6 other programmable endpoints. Besides the default EP0, the CDC specification uses other 3 endpoints. One is used by the CDC class interface, and the other two are used by the data interface. We configure the endpoints in the AVR as follow:

1. Endpoint 1 is configured in bulk IN mode and has two 64 byte banks in ping-pong mode. It sends the data from the wireless interface to the PC.
2. Endpoint 2 is configured in bulk OUT mode and has a 32 byte single bank. It receives data from the PC. The reason not using a 64 byte bank is that 32 byte matches the maximum package size of the RF chip. This configuration effectively prevents the incoming USB packages from overflow.
3. Endpoint 3 is configured in interrupt IN mode and has a 64 byte single bank. It works as the management endpoint for the CDC class interface. This endpoint is essentially a dummy one. We do not need to handle any event from it.

After finish the embedded software, we also need to install the PC driver to test it. Since Windows has a built-in driver for CDC devices, all we need to do is constructing an .inf file that describes the device. In order to identify our device, we have to set up the VID (Vendor ID) and PID (Product ID) first. Nonetheless, applying for valid VID and PID is expensive. Fortunately, ATMEL provides several sets of free VID and PID for products that uses AVR. The free set for CDC devices are VID=0x16C0 and PID=0x05DF. Figure 16 is a screen shot of the Windows device manager when the receiver is connected.

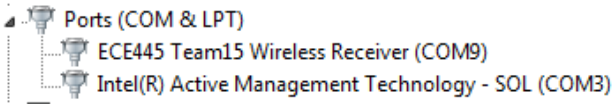


Figure 16: The Windows device manager identifies the receiver.

Since the receiver is working, we moved on to develop the PC interface software. The resulting GUI (Graphic User Interface) is shown in Figure 17. The program delivers the following functions so far:

1. Read from the virtual serial port and decode the received data into the real-time roll, pitch, and yaw information of the vehicle. The low level serial port communication support is provided by a third party library called QExtSerialPort.
2. Send encoded command to the virtual serial port for controlling the movement of the robot according the the interaction between the GUI widgets and the operator.
3. Provide intuitive illustration for the incoming data by reflecting the Eular angles on 3D models.
4. Real-time line charts of the data present the trends of the vehicle's movement.

5. Received data can be stored in a special compact file format for future playback and analysis.

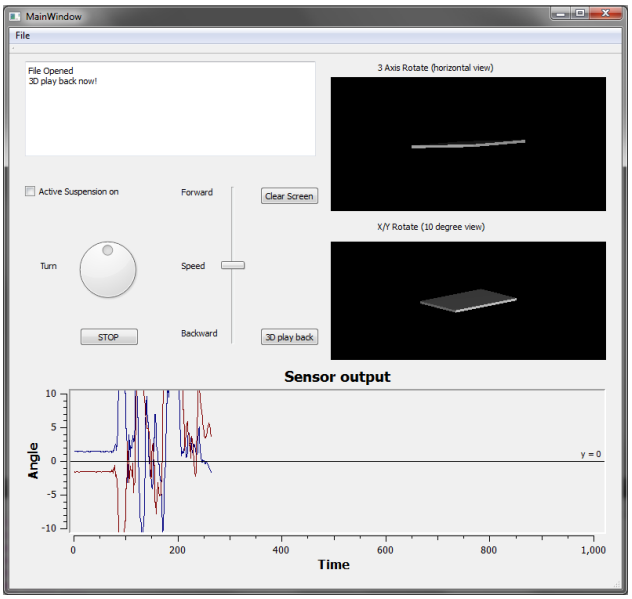


Figure 17: GUI software

3 Design Verification

This section discusses the general result that the major parts of the system yield. For the full requirements and verification table, please see the Appendix A.

3.1 Master Controller

The main controller is fully functional according to the verification table. Besides, thanks to the RTOS, we managed to develop a method to probe the CPU usage of the main processor by counting how much time the system idle task occupies the CPU comparing to the total time that the CPU runs. Two different counting methods delivered the ceiling and the floor of the performance. Figure 18 shows the surprising result of the test: because of the high performance non-blocking asynchronous drivers that we designed, the CPU usage is as low as around 0.1%$\sim$ 4.6.

1. **Performance ceiling:** We consider the CPU at idle only when the idle task occupies a whole OS tick (1ms).
2. **Performance floor:** We consider the CPU at idle if during an OS tick the idle task is ever resumed.

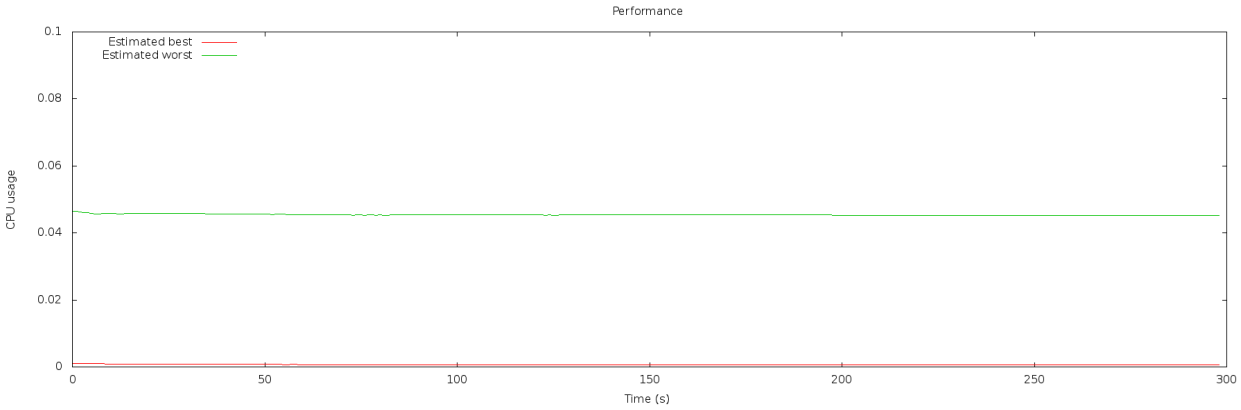


Figure 18: CPU usage chart

3.2 MEMS Sensors

The MEMS sensor is able to provide roll, pitch and yaw tilt in quaternion format. With its built-in filter, the results show no pattern of drift and notable noise. The data is verified by keeping the robot still for a certain period and tracking the sensor reading. After test, we found the standard deviation of total 978 data points is 0.0534, which is satisfying. Furthermore, the long term drift is almost zero in the result chart (Figure 19). In the plot, the blue trace is the roll reading while the violet one the from pitch. Since the yaw data are irrelevant to the project, we exclude them from the graph. Conclusively, the sensor output is robust and credible.

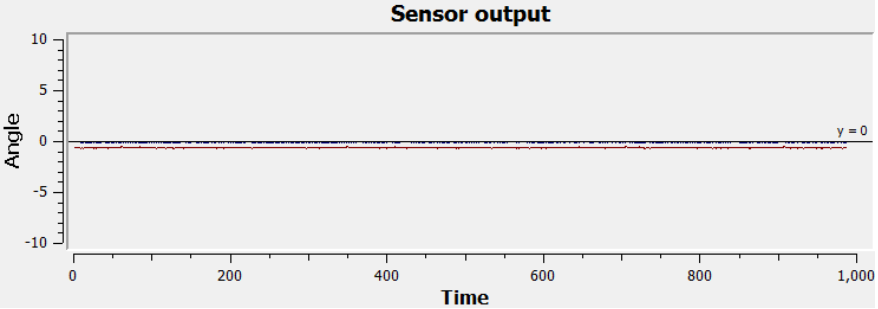


Figure 19: MEMS sensor test

3.3 ODM System

The overall ODM system has achieved a stable performance. The PID control system for the driving wheel is able to keep a 300ms settling time. The servos are calibrated to output the same angle with errors confined in 5°.

3.4 AL System

Due to the mechanical limitation of the worm motor, we could not test the active suspension system of the vehicle in real world scenarios. However, we still tested the system using software simulation and later proved that our design is viable. The first verification conducted was to test the settling time of the control system. The original requirement for it was 50ms. After a careful reconsideration, we changed the requirement to 300ms. Several reasons exist for this change. First, our control feedback sampling period is 50ms, a greater settling time should be allowed for feedback to become effective. Second, a 50ms settling time will result in big PID constants, which would make the system unstable by introducing larger overshoot. Last but not least, a 300ms settling time is quick enough for the system to react. Figure 20 presents the simulation results. The white signals are randomly generated ramps, and the violet ones shows the system's reaction. The graph justifies the quick response rate of the AL algorithm.

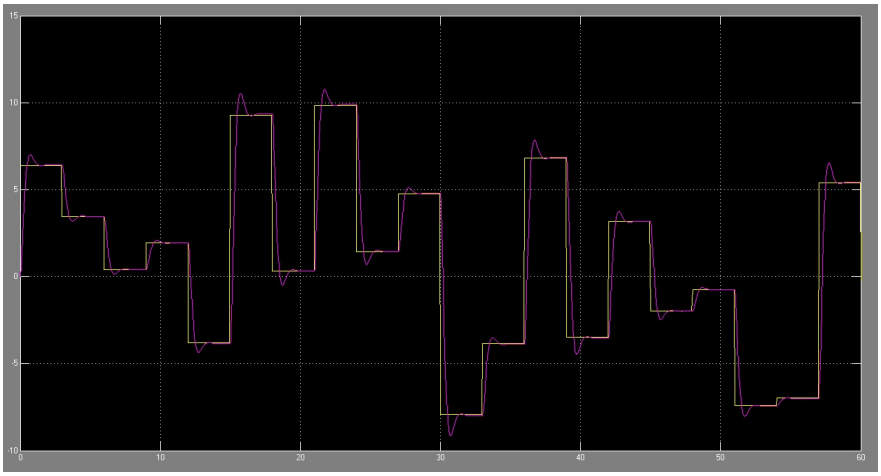


Figure 20: AL algorithm simulation

4 Cost

4.1 Parts

Table 4: Itemized budget

Part name	**Unit cost ($)**	**Manufacturer**	**Number required**	**Subtotal ($)**	**Actual Cost ($)**
AT32UC3C2256	13.78	ATMEL	1	13.78	13.78
ATmega8A	1.84	ATMEL	4	7.36	7.36
ATmega32U4	3.38	ATMEL	1	3.38	3.38
Worm motor	7	from China	4	28	28
Servo	4	from China	4	16	16
DC motor	4	from China	4	16	16
Encoder	2	from China	4	8	8
Battery	8.2	from China	1	8.2	8.2
nRF24L01P RF module	10.0	Nordic Semiconductor	2	20.0	20.0
MPU-6050	7.14	InvenSense	1	7.14	29.28
IRF3205S	0.42	International Rectifier	16	6.72	6.72
IR2104	0.62	International Rectifier	8	4.96	4.96
MAX629	1.54	MAXIM	1	1.54	0
LM2596S-5.0	0.31	Texas Instrument	4	1.24	0
LM1117-3.3	0.05	Texas Instrument	2	0.10	0.10
LM2594-3.3	1.54	Texas Instrument	1	1.54	1.54
OLED Display	4.92	Univision	1	4.92	4.92
oscillator	0.77	from China	6	4.62	4.62
PCB	20	from China	3	60	60
Misc. components	from China	N/A	N/A	5	5
Total				218.5	237.86

4.2 Labor

Machine shop estimation: $1000
Zhangxiaowen Gong: $35/hr × 12hr/week × 10 weeks = $4200
Wenjia Zhou: $35/hr × 12hr/week × 10 weeks = $4200
Jun Ma: $35/hr × 12hr/week × 10 weeks = $4200
Total: $4200 × 3 × 2.5 = $31500
The total expense is $31500 + $237.86 = $1000 = $32737.86

5 Conclusion

5.1 Accomplishments and Future Work

We are satisfied with the performance of our vehicle. The ODM system, on-board task management, and the wireless interface all work flawlessly. Our design has fulfilled most of the requirements and functioned as expected except that the AL system does not work due to the insufficient torque from the worm motor that we use. Nonetheless, the simulation results give promising data to justify the AL algorithm. As a result, a redesign to the physical AL system should deliver successful outcome. A possible alternative approach is to replace the worm motor with linear actuator. The latter one is slower yet more stable and stronger. We may also attach a pressure sensor to each driving wheel to examine whether the wheel is firmly touching the ground so that the measured vehicle state is more credible. In addition, the open-loop controlled RC servos are hard to be calibrated to output the same angle; hence, we may later adding sensors such as Hall effect sensors to transform the system into a closed-loop controlled one. Another possible improvement is on the processor side. Since the CPU usage shows that the main microcontroller is significantly overpowered, we may also substitute it with a low end ARM Cortex-M3 processor, which may cost only around $3 each.

5.2 Uncertainties

The mechanical part on the small size model system deviates from an actual full size vehicle notably. Therefore, although we successfully implement the design on the model system, all the parameters in the control system have to be re-calculated for the real system. Furthermore, an operating system based software tends to hide subtle synchronization bugs, which may result in hanging the system. For critical operations like carrying dangerous chemicals, such system failure is intolerable. Consequently, the software has to be carefully examined to eliminate any of such bugs.

5.3 Ethical considerations

We commit ourselves to the IEEE code of ethics, the ethical concerns we have is as follows:

> 'to accept responsibility in making decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;'

In the case of our project, our design is intended for laboratory use. Nonetheless, we cannot neglect the possibility that our design is used by unauthorized person or group for transporting unexpected and dangerous chemical, for instance, any sort of poisons or explosives. If our product is to be used, we will make sure we only hand it to authorized person or group, and make sure only they know how to operate the system.

References

[1] "Omnix Technology, Inc. - Directional Components & Integrated Systems," Web page, accessed May 2012. [Online]. Available: http://www.omnixtechnology.com/direct_components.html

[2] *App Note: 9-Axis MotionFusion & Calibration Algorithms*, InvenSense Inc., Sunnyvale, CA, 2011. [Online]. Available: http://www.invensense.com/developers/index.php?_r=downloads&ajax=dlfile&file=AppNote%20-%209-Axis%20MotionFusion%20and%20Calibration%20Algorithms.pdf

[3] "IRF3205S IRF3205L," Datasheet, International Rectifier, 2002. [Online]. Available: http://www.irf.ru/pdf/irf3205s.pdf

[4] "Class 1 Bluetooth Dongle Test," Web page, accessed May 2012. [Online]. Available: http://www.amperordirect.com/pc/r-electronic-resource/z-reference-bluetooth-class1-myth.html

[5] *IEEE 802.15.4d-2009 IEEE Standard for Information Technology - Specific Requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, IEEE, New York, NY, 2009. [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf

[6] "nRF24L01+ Product Specification," Datasheet, Nordic Semiconductor, 2008. [Online]. Available: http://www.nordicsemi.com/kor/nordic/download_resource/8765/2/23366062

[7] *Bi-directional Level Shifter for I^2C-bus and Other Systems*, Philips Semiconductors, 1997. [Online]. Available: http://ics.nxp.com/support/documents/interface/pdf/an97055.pdf

[8] "LM2596 SIMPLE SWITCHER Power Converter 150 kHz3A Step-Down Voltage Regulator," Datasheet, Texas Instrument, 2002. [Online]. Available: http://www.ti.com/lit/ds/symlink/lm2596.pdf

[9] "MAX629 28V, Low-Power, High-Voltage, Boost or Inverting DC-DC Converter," Datasheet, Maxim Integrated Products, 1997. [Online]. Available: http://pdfserv.maxim-ic.com/en/ds/MAX629.pdf

Appendix A Requirement and Verification Table

A.1 Main Controller

Table 5: Requirement and verification for the main controller

Item	Requirement and verification (Y/N)	Test/calibration procedure
OLED display	1. Display characters at specific coordinates **(Y)**	1. (a) Command the OLED to display strings with various lengths at different location (b) See if all 128×64 pixels are operational (c) See if all ASCII characters are mapped to pixels correctly (d) See if the designated coordinates are translated to right positions (e) Check if the strings wrap around correctly
DSP filter (if present)	1. Finish each filter computation within 10ms **(Software filter is abandoned)** 2. Filter at least 90% vibrational noise out	1. (a) Turn on the built-in 16-bit T/C (timer/counter) before starting the calculation and record the value in the T/C register right after the computation finishes (b) Calculate the delay from the T/C reading and its clock source setting 2. (a) Send both filtered and raw data to a computer (b) Calculate and compare the standard deviation of each set of data

Continued on next page

Table 5 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
I^2C interface	1. Signals have almost vertical edges maximum clock speed at 400KHz **(Y)** 2. No malfunction in high-noise environment **(Y)** 3. Communication between 3.3V I^2C bus and 5V I^2C bus **(Y)** 4. The streaming algorithm is able to save over 70% CPU cycles from simple synchronous drivers **(Y)**	1. (a) Set the I^2C bus to 400KHz and then use an oscilloscope to probe the SCL and SDA lines (b) Check the edges on the signals and determine of the curve on the edges, which are caused by the capacitance on the trace/wire and the pull-up resister specified in the I^2C standard, is in an acceptable shape 2. Test the transmission when the H-bridges and motors are turned on, and see if any data loss occurs 3. Test the level shifting circuit to see if it converts signals bidirectionally 4. Loop to append 100 I^2C transactions and see how many packages remain in the queue after the appending operation is done; the more packages still stay in the queue the better the algorithm is

Continued on next page

Table 5 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
SPI interface	1. Internal shift register is functional **(Y)** 2. No malfunction in high-noise environment **(Y)** 3. When addressing one slave device, no interference coming from the others **(Y)** 4. Utilize the DMA channel for both transmitting and receiving data **(Y)**	1. Check if the slave data shift to the master correctly when the master shift its data out 2. Test the transmission when the H-bridges and motors are turned on, and see if any data loss occurs 3. (a) Pull down the chip select lines on different slaves and see if data go to correct receiver (b) Check if the master changes SPI mode (clock phase, clock polarity, etc.) correctly when switching among slaves that adopt different modes (c) Keep addressing various slaves for 10 minutes and capture any malfunction 4. (a) Turn on the inner-loop function of AVR32's SPI module, which interconnect its input to its output (b) Set one DMA channel to transmit data in a block of memory (c) Set the second DMA channel to receive data and to store them in another block of memory (d) After the transmission finishes, check the consistency of the content in the two memory blocks

A.2 MEMS Sensors

Table 6: Requirement and verification for the MEMS sensors

Item	Requirement and verification (Y/N)	Test/calibration procedure
MPU-6050	1. Be configured with proper work mode (sensitive range, interrupt output, etc.) **(Y)** 2. Allow the main controller to configure its auxiliary sensor (HMC5883L) **(HMC5883L is abandoned)** 3. Accelerometer and gyro scope data are consistent **(Y)**	1. (a) Read values from all registers from the sensors' register map and check with the default value stated in their datasheets (b) Change values in several registers and then read back to see if the operations succeed (c) Move/turn the sensor at certain rate and check if the output is in the expected range 2. (a) Use the sensor's internal MUX to hand the auxiliary sensor to the main controller temporarily (b) Let the main controller to verify if it can change the auxiliary sensor's registers correctly (c) Get the auxiliary sensor back from the main controller and try to read its output (d) Let the main controller to address the auxiliary sensor again; the operation should fail now 3. (a) Put sensor unit on the table, tilt a little bit, and record the output (b) Calculate θ from accelerometer's x-axis (x) and z-axis (z) data: $\theta = \arctan(-\frac{x}{z})$ (c) Calculate γ from gyroscope's z-axis (ω) output: $\gamma = \omega \times t$ (d) θ should be close to γ

Continued on next page

Table 6 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
HMC5883L	1. Be configured with proper work mode (gain, interrupt output, etc.) **(HMC5883L is abandoned)** 2. Error is controlled within 2°	1. Perform similar procedure that checks the MPU-6050 2. (a) Place the sensor on the table and tilt 45° at a time (b) Record the output and compare it to the actual angle; $error = actual - output$ (c) Repeat the step (a) and (b) until the total angle tilted reaches 720° (d) Calculate the peak and average of the errors

A.3 Side Controllers

Table 7: Requirement and verification for the side controllers

Item	Requirement and verification (Y/N)	Test/calibration procedure
I^2C interface	1. Robust communication with no data loss and bus errors in high-noise environment **(Y)** 2. Multiple side MCUs can communicate with the main controller via a single I^2C interface without suffering bus conflict **(Y)**	1. Perform the test 1 and 2 in the I^2C part in section 3.1.1 2. (a) Send commands to different slave MCUs to verify the addressability (b) Let the main controller keep changing the slave MCU for 10 minutes and see if the bus fails in the duration
PWM	1. Generate 3 PWM signals with different duty cycles simultaneously **(Y)**	1. (a) Write a program that generate 3 PWM signals with different duty cycles and changes the duty cycle every 10 seconds (b) Use 2 oscilloscopes to probe the 3 PWM output pins (c) Check the correctness of each signal over time

Continued on next page

Table 7 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
H-bridge	1. Require no heat sink when driving current stables at the worm motor's rated current, which is 1.10A **(Y)** 2. Respond to PWM signals that have duty cycles ranging from 15% to 95% **(Y)**	1. (a) Connect an ammeter between the H-bridge output and the worm motor (b) Set the H-bridge to output at full speed (95% duty cycle PWM) and increase the load on the worm motor until the ammeter reading reaches 1.10A (c) Keep the motor spinning for 2 minutes (d) Put on antistatic wrist strap and feel the temperature on the driving MOSFET with bare finger; the finger should not feel hot 2. (a) Set the PWM signal to 15% (b) Let the H-bridge drive a motor with no load for 1 minute and see if the speed is constant (c) Repeat (a) and (b) with a 10% increase in duty cycle each time untile the duty cycle reaches 95%

A.4 Wireless Communication

Table 8: Requirement and verification for the wireless communication

Item	Requirement and verification (Y/N)	Test/calibration procedure
nRF24L01P	1. Be configured with proper work mode (channel, address, data rate, etc.) **(Y)** 2. Communication range is higher than 100m outdoor **(Y)** 3. Communication is not affected if the transceivers have a wall in between **(Y)**	1. Perform the test 1 in the MPU-6050 part in section 3.1.2 2. (a) Put the transmitter and receiver at the two corners of the Quad (b) Turn on both transceivers and check for stable transmission (c) If fail to establish the data link, hold the transmitter and walk towards the receiver until stable data appears (d) Mark the final locations of the transceivers on a scaled map and measure the distance 3. (a) Put the two transceivers in two adjacent rooms and start the transmission (b) If the data is stable, move the transceivers to rooms that have two walls in between (c) Increment the number of walls in between and perform the test until either the transmission is blocked of the number reaches 5 (d) Repeat the test with transceivers placed in different floors

Continued on next page

Table 8 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
Driver	1. Connect/disconnect automatically when the paired transceivers are in-range/out-of-range **(Y)** 2. Automatically resend packages that is not acknowledged by the receiver **(Y)** 3. The streaming algorithm is able to save over 70% CPU cycles from simple synchronous drivers **(Y)**	1. (a) Make the transmitter keep sending packages (b) Switch the on/off state of the receiver several times (c) See if the link is reestablished automatically 2. In the above test, check if any package is lost during the off state of the receiver 3. Perform the test 3 in the I^2C part in section 3.1.1

A.5 Omni-directional Movement

Table 9: Requirement and verification for the omni-directional movement

Item	Requirement and verification (Y/N)	Test/calibration procedure
DC motor	1. With speed control algorithm, all motor respond the speed command ($0 \sim 255$) identically **(Y)**	1. (a) Set the speed commands for all motors to 50 (b) Use a tachometer to record the speed of each motor (c) Repeat (a) and (b) with a 50 increase in speed command each time until the command reaches 250

Continued on next page

Table 9 – continued from previous page

Item	Requirement and verification (Y/N)	Test/calibration procedure
Servo	1. All servos respond to position commands identically, which means turning to the same angle **(Y with error in 5^{circ})** 2. Turn 60° with regular load within 0.25 second **(Y)**	1. (a) Set PPM signals equal to the raw position commands (b) Set the duty cycle of the raw command to 2.5% (0.5 millisecond high level) and provide it to all servos (c) Record the position of all servos (d) Repeat (b) and (c) with a 1.0% increase in duty cycle each time until the duty cycle reaches 12.5% (2.5 millisecond high level) (e) Pick one servo as standard, and then calculate scalar and offset errors of the other servos according to the results (f) Apply proportional and offset factors to the raw command to generate modified signals and do (b) to (d) several times until all servos respond to the raw command identically 2. (a) Put the vehicle on the table and let one servo turn 60° (b) Use a video camera with 60fps frame rate to capture the movement (c) Playback the video frame by frame and count the number of frames taken to finish the turning (d) Calculate the actual delay in second by $delay = \frac{frametaken}{60}$

A.6 Active Suspension

Table 10: Requirement and verification for the active suspension

Item	Requirement and verification (Y/N)	Test/calibration procedure
PID algorithm	1. Settling time is controlled in 50ms **(Requirement changed to 300ms; confirmed in simulation)** 2. Error between desired angle and actual turned angle is smaller than 2° **(Verified in simulation)**	1. (a) Run Simulink in MATLAB and set the desired angle x as input to the system (b) Plot step response in MATLAB and find the time when the output reaches 95% of steady state value (c) Load PID parameter into the vehicle and run it over a ramp with $slope = x$ (d) Collect potentiometer reading from the controller's ADC (analog to digital converter), which indicates the angle turned, and import them to MATLAB (e) Mark the delay between the initial position and the final position by analyzing the data 2. (a) Set 10° as the target angle to the PID system (b) Measure the actual turned angle via a protractor after the system reaches stable state (c) Record the error (d) Repeat from (a) to (c) with a 10° increase in target angle each time until the target angle reaches 50°

A.7 PC Receiver

Table 11: Requirement and verification for the PC receiver

Item	Requirement and verification (Y/N)	Test/calibration procedure
USB interface	1. Both PC side driver and MCU side driver are functional **(Y)** 2. Data rate is independent from the virtual serial port's baud rate setting **(Y)**	1. (a) Connect the receiver to the PC's USB port and see if the PC recognize it with the name specified in the .inf file we write (b) Open the corresponding virtual serial port in the HyperTerminal (c) Exchange a group of data and verify the correctness 2. (a) Set the baud rate of the virtual serial port to 240bps (b) Loop to send 1KB of random data to the PC (c) Check whether the transmission completes instantly instead of taking around 4 seconds
PC software	1. Receive real-time data and reflect them on line charts **(Y)** 2. Store data and time in file and playback afterwards **(Y)**	1. (a) Establish the wireless link between the vehicle and the PC (b) Move the vehicle randomly by hand (c) Check if the line charts correctly display the movement without recognizable delay 2. (a) Write received data to a non-existed file and check if the file is created successfully in the file system (b) Open the file and check the consistency with the original data

A.8 Power Management

Table 12: Requirement and verification for the power management

Item	Requirement and verification (Y/N)	Test/calibration procedure
Voltage regulation	1. Correctly generate 12V (10% tolerance), 5V (1% tolerance), and 3.3V (1% tolerance) voltages **(Y 12V requirement is changed to 13.8V)** 2. Voltages are stable (5% tolerance) during potential instant high load **(Y)**	1. Use voltmeter to probe all 3 generated voltages 2. (a) Build a voltage divider to translate all three voltages proportionally to 2V range (b) Temporarily connect the three translated voltages to the main controller's ADC input pins (c) Run the vehicle for 5 minutes and send the ADC readings real-timely to the host computer (d) Calculate the standard deviation and the minimum of the data (e) Replace the bypass capacitors with ones having higher capacitance if instant voltage drops are observed, and then repeat from (a) to (d)
Battery	1. Support the vehicle for 1 hour of continuous operation **(Y)**	1. (a) Fully charge the battery (b) Run the vehicle until the battery dies (c) Record the duration

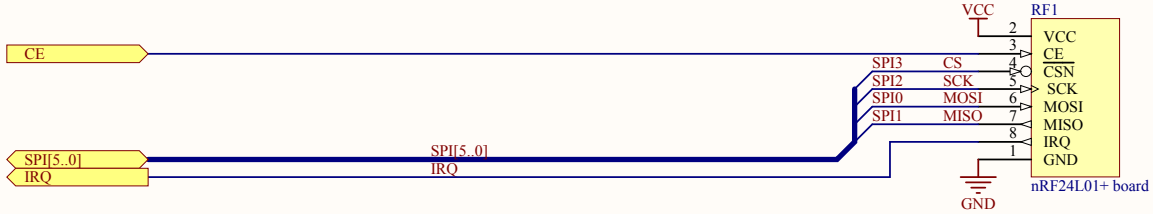
Continued on next page

Table 12 – continued from previous page

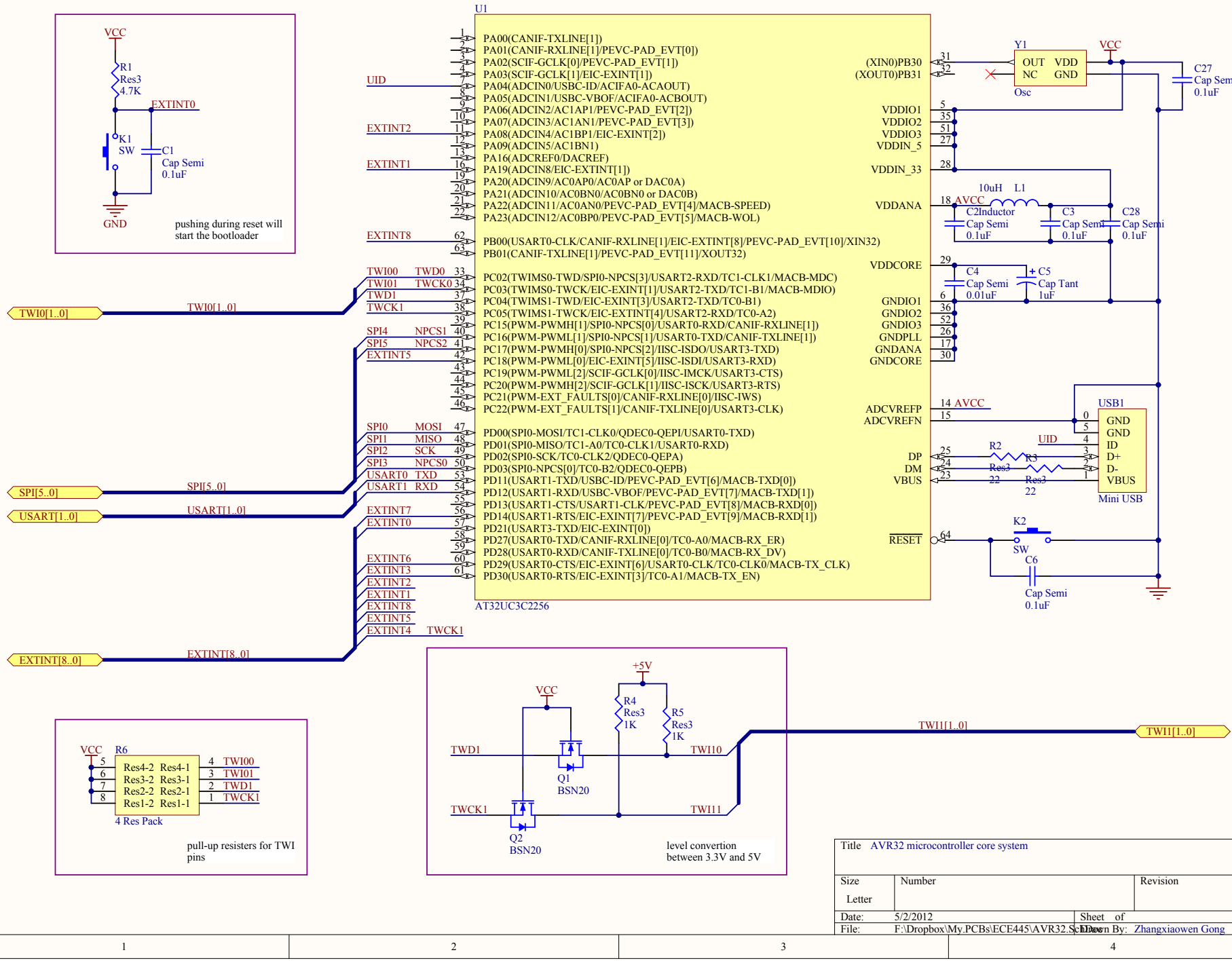
Item	Requirement and verification (Y/N)	Test/calibration procedure
Power consumption	1. Peak current is lower than 6A **(Unable to test)** 2. Average current is lower than 2A **(Unable to test)**	1. (a) Connect a 0.25Ω 10W sensing resister in the battery loop (b) Use AVR32's differential ADC channel to probe the voltage drop on the resister (c) Run the vehicle for 10 minutes and send the ADC readings real-timely to the host computer (d) Find the peak current among the data 2. In the above test, continue to calculate the average of all data

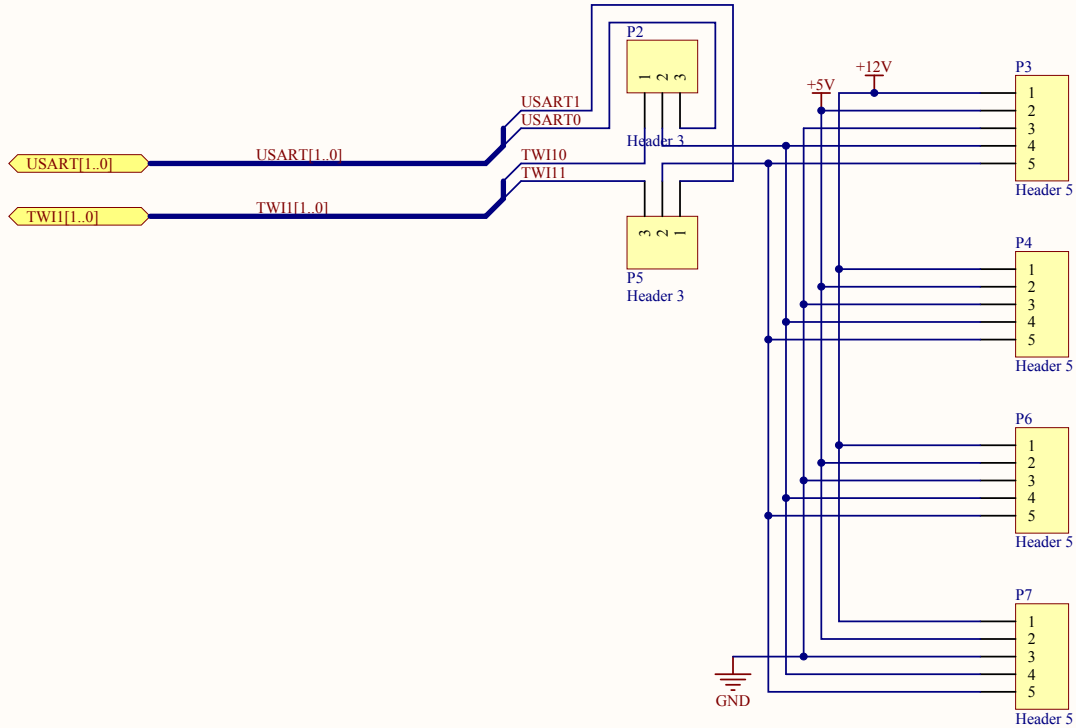
Appendix B Full Schematics and PCB Layouts

B.1 The Master Controller

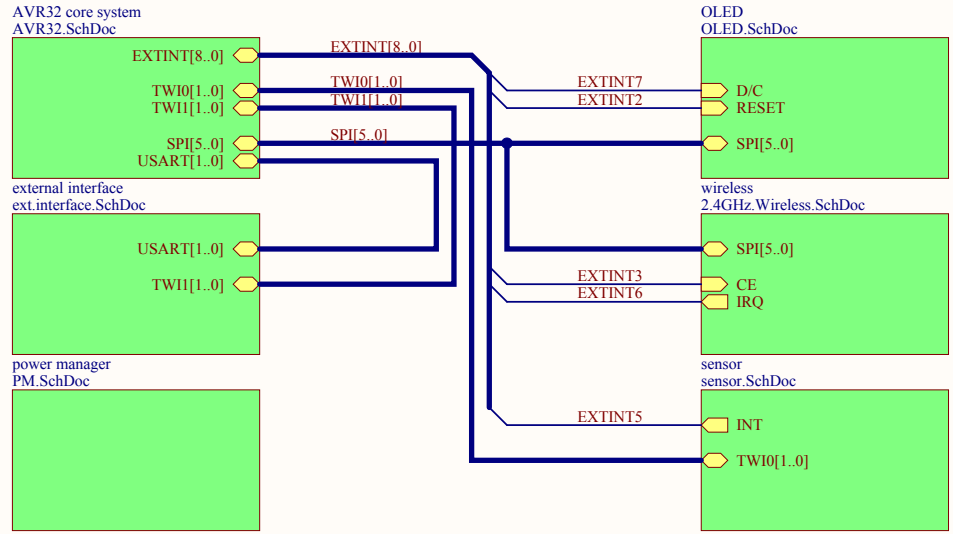


Title 2.4GHz wireless interface		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\2.4GHz.Wireless.SchDoc		Drawn By: Zhangxiaowen Gong

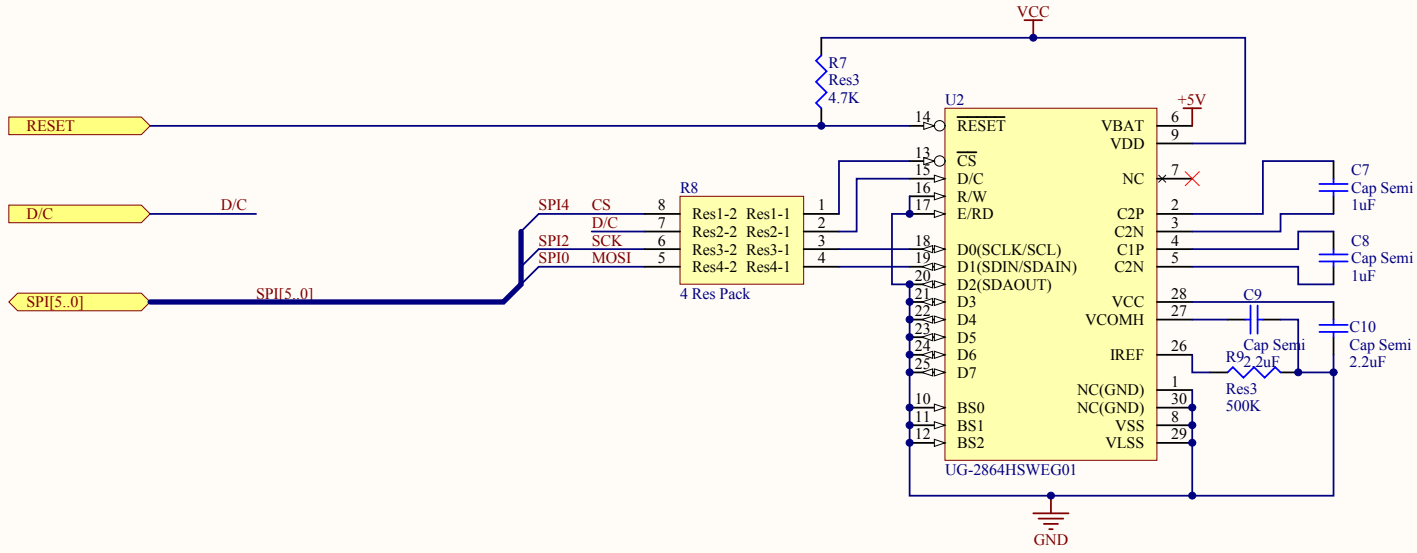

U1
AT32UC3C2256
PA00(CANIF-TXLINE[1])
PA01(CANIF-RXLINE[1]/PEVC-PAD_EVT[0])
PA02(SCIF-GCLK[0]/PEVC-PAD_EVT[1])
PA03(SCIF-GCLK[1]/EIC-EXINT[1])
PA04(ADCIN0/USBC-ID/ACIFA0-ACAOUT)
PA05(ADCIN1/USBC-VBOF/ACIFA0-ACBOUT)
PA06(ADCIN2/AC1AP1/PEVC-PAD_EVT[2])
PA07(ADCIN3/AC1AN1/PEVC-PAD_EVT[3])
PA08(ADCIN4/AC1BP1/EIC-EXINT[2])
PA09(ADCIN5/AC1BN1)
PA16(ADCREF0/DACREF)
PA19(ADCIN8/EIC-EXTINT[1])
PA20(ADCIN9/AC0AP0/AC0AP or DAC0A)
PA21(ADCIN10/AC0BN0/AC0BN0 or DAC0B)
PA22(ADCIN11/AC0AN0/PEVC-PAD_EVT[4]/MACB-SPEED)
PA23(ADCIN12/AC0BP0/PEVC-PAD_EVT[5]/MACB-WOL)
PB00(USART0-CLK/CANIF-RXLINE[1]/EIC-EXTINT[8]/PEVC-PAD_EVT[10]/XIN32)
PB01(CANIF-TXLINE[1]/PEVC-PAD_EVT[11]/XOUT32)
PC02(TWIMS0-TWD/SPI0-NPCS[3]/USART2-RXD/TC1-CLK1/MACB-MDC)
PC03(TWIMS0-TWCK/EIC-EXINT[1]/USART2-TXD/TC1-B1/MACB-MDIO)
PC04(TWIMS1-TWD/EIC-EXINT[3]/USART2-TXD/TC0-B1)
PC05(TWIMS1-TWCK/EIC-EXTINT[4]/USART2-RXD/TC0-A2)
PC15(PWM-PWMH[1]/SPI0-NPCS[0]/USART0-RXD/CANIF-RXLINE[1])
PC16(PWM-PWML[1]/SPI0-NPCS[1]/USART0-TXD/CANIF-TXLINE[1])
PC17(PWM-PWMH[0]/SPI0-NPCS[2]/IISC-ISDO/USART3-TXD)
PC18(PWM-PWML[0]/EIC-EXINT[5]/IISC-ISDI/USART3-RXD)
PC19(PWM-PWML[2]/SCIF-GCLK[0]/IISC-IMCK/USART3-CTS)
PC20(PWM-PWMH[2]/SCIF-GCLK[1]/IISC-ISCK/USART3-RTS)
PC21(PWM-EXT_FAULTS[0]/CANIF-RXLINE[0]/IISC-IWS)
PC22(PWM-EXT_FAULTS[1]/CANIF-TXLINE[0]/USART3-CLK)
PD00(SPI0-MOSI/TC1-CLK0/QDEC0-QEPI/USART0-TXD)
PD01(SPI0-MISO/TC1-A0/TC0-CLK1/USART0-RXD)
PD02(SPI0-SCK/TC0-CLK2/QDEC0-QEPA)
PD03(SPI0-NPCS[0]/TC0-B2/QDEC0-QEPB)
PD11(USART1-TXD/USBC-ID/PEVC-PAD_EVT[6]/MACB-TXD[0])
PD12(USART1-RXD/USBC-VBOF/PEVC-PAD_EVT[7]/MACB-TXD[1])
PD13(USART1-CTS/USART1-CLK/PEVC-PAD_EVT[8]/MACB-RXD[0])
PD14(USART1-RTS/EIC-EXINT[7]/PEVC-PAD_EVT[9]/MACB-RXD[1])
PD21(USART3-TXD/EIC-EXINT[0])
PD27(USART0-TXD/CANIF-RXLINE[0]/TC0-A0/MACB-RX_ER)
PD28(USART0-RXD/CANIF-TXLINE[0]/TC0-B0/MACB-RX_DV)
PD29(USART0-CTS/EIC-EXINT[6]/USART0-CLK/TC0-CLK0/MACB-TX_CLK)
PD30(USART0-RTS/EIC-EXINT[3]/TC0-A1/MACB-TX_EN)
(XIN0)PB30
(XOUT0)PB31
VDDIO1
VDDIO2
VDDIO3
VDDIN_5
VDDIN_33
VDDANA
VDDCORE
GNDIO1
GNDIO2
GNDIO3
GNDPLL
GNDANA
GNDCORE
ADCVREFP
ADCVREFN
DP
DM
VBUS
RESET
UID
EXTINT2
EXTINT1
EXTINT8
TWI00
TWD0
TWI01
TWCK0
TWD1
TWCK1
SPI4
NPCS1
SPI5
NPCS2
EXTINT5
SPI0
MOSI
SPI1
MISO
SPI2
SCK
SPI3
NPCS0
USART0
TXD
USART1
RXD
EXTINT7
EXTINT0
EXTINT6
EXTINT3
EXTINT2
EXTINT1
EXTINT8
EXTINT5
EXTINT4
TWCK1
TWI0[1..0]
SPI[5..0]
USART[1..0]
EXTINT[8..0]
Y1
OUT
VDD
NC
GND
Osc
C27
Cap Semi
0.1uF
10uH
L1
C2Inductor
Cap Semi
0.1uF
C3
Cap Semi
0.1uF
C28
Cap Semi
0.1uF
C4
Cap Semi
0.01uF
C5
Cap Tant
1uF
USB1
GND
GND
ID
D+
D-
VBUS
Mini USB
R2
R3
Res3
22
K2
SW
C6
Cap Semi
0.1uF
R1
Res3
4.7K
EXTINT0
K1
SW
C1
Cap Semi
0.1uF
GND
pushing during reset will
start the bootloader
R6
Res4-2 Res4-1
Res3-2 Res3-1
Res2-2 Res2-1
Res1-2 Res1-1
TWI00
TWI01
TWD1
TWCK1
4 Res Pack
pull-up resisters for TWI
pins
+5V
R4
Res3
1K
R5
Res3
1K
TWD1
TWI10
Q1
BSN20
TWCK1
TWI11
Q2
BSN20
level convertion
between 3.3V and 5V
TWI1[1..0]
Title AVR32 microcontroller core system
Size
Letter
Number
Revision
Date: 5/2/2012
Sheet of
File: F:\Dropbox\My PCBs\ECE445\AVR32.SchDoc
Drawn By: Zhangxiaowen Gong



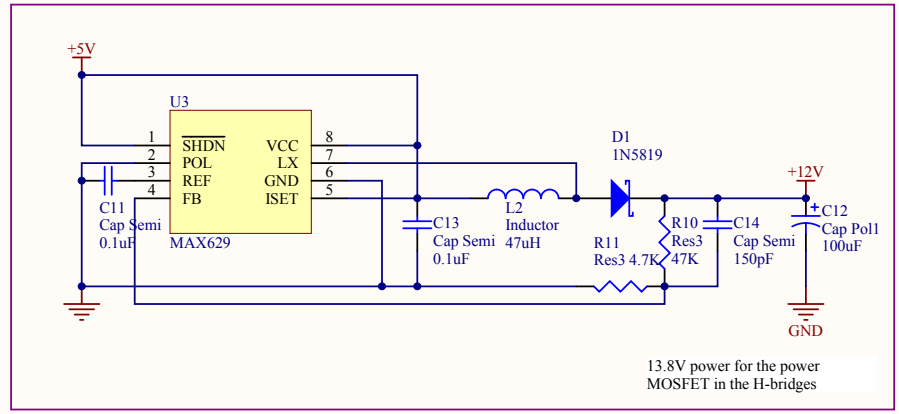
Title External interface to motor control boards		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\ext.interface.SchDoc		Drawn By: Zhangxiaowen Gong

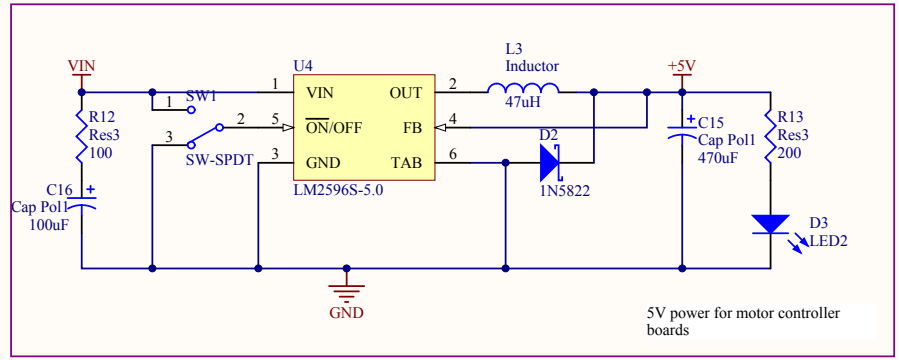


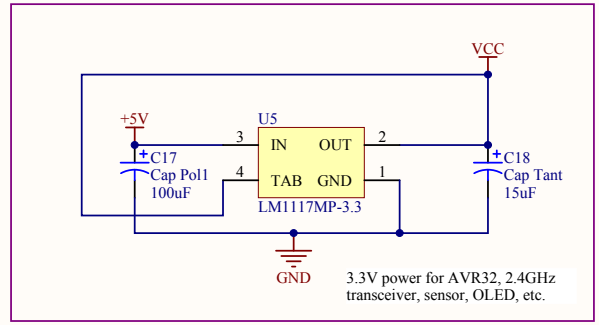
Title Main sheet		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\...\main.board.SchDoc		Drawn By: Zhangxiaowen Gong

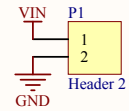


Title 0.9" OLED display interface		
Size: Letter	Number:	Revision:
Date: 5/2/2012	Sheet of	
File: F:\Dropbox\My PCBs\ECE445\OLED.SchDoc	Drawn By: Zhangxiaowen Gong	

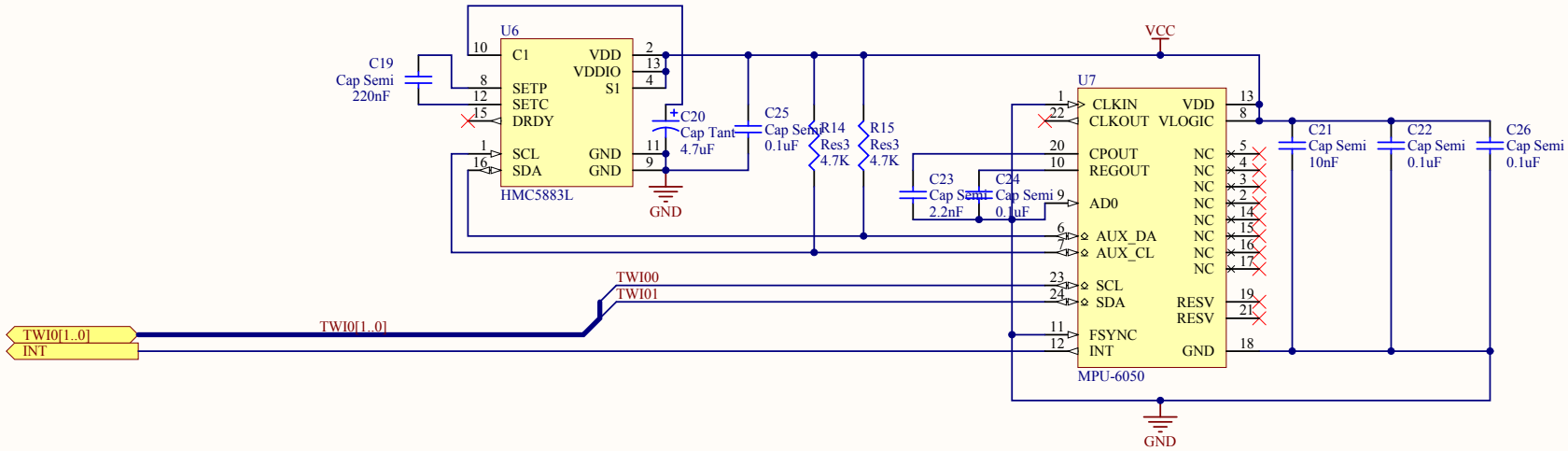




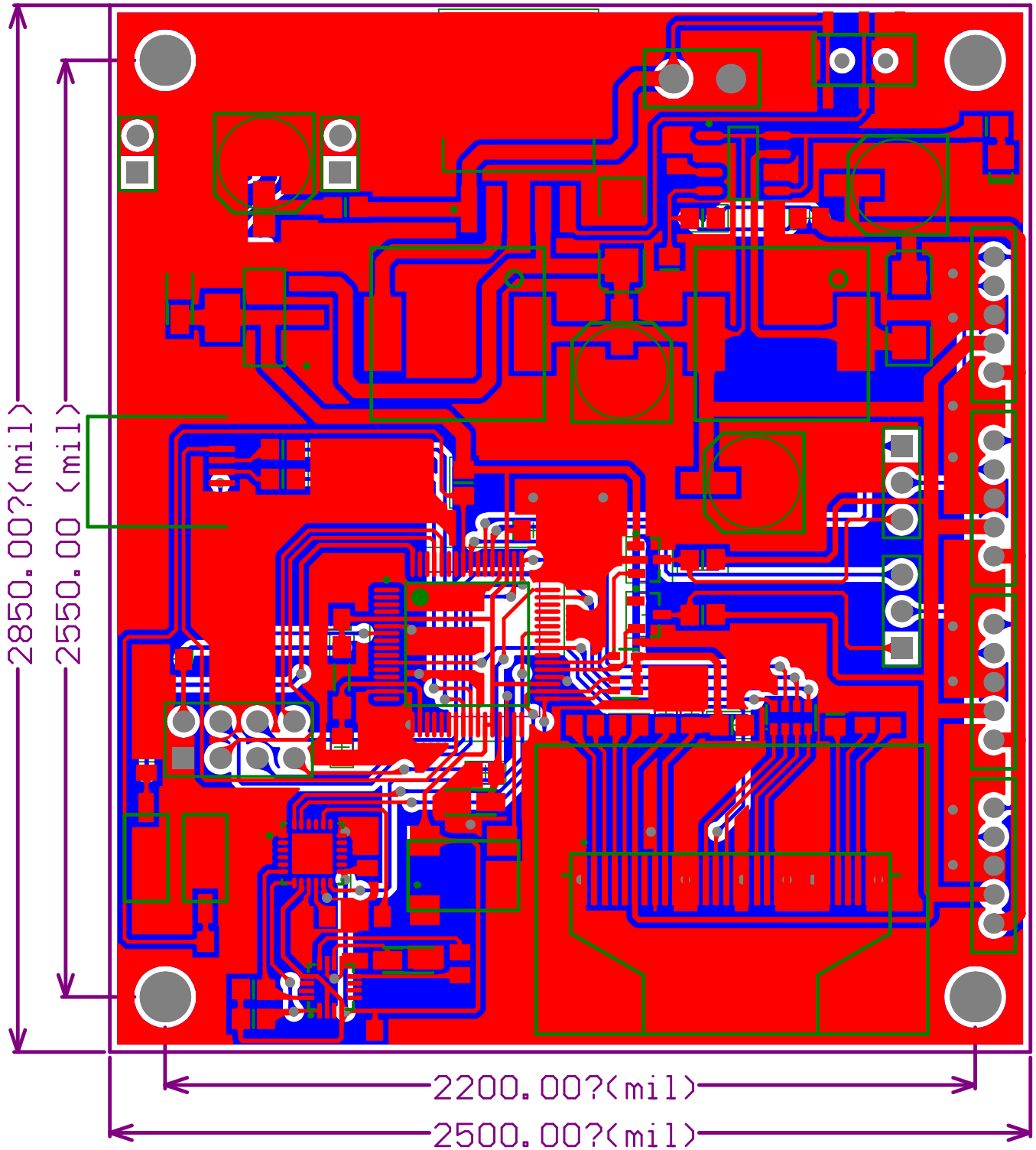




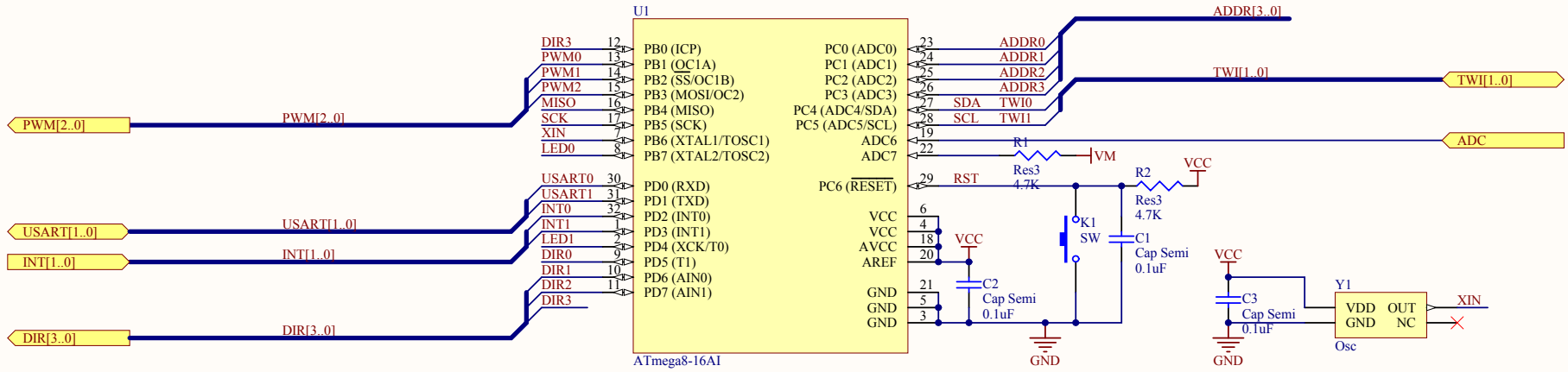
Title: Power management providing +3.3V, +5V, and +12V		
Size: Letter	Number:	Revision:
Date: 5/2/2012	Sheet of	
File: F:\Dropbox\My PCBs\ECE445\PM.SchDoc	Drawn By: Zhangxiaowen Gong	



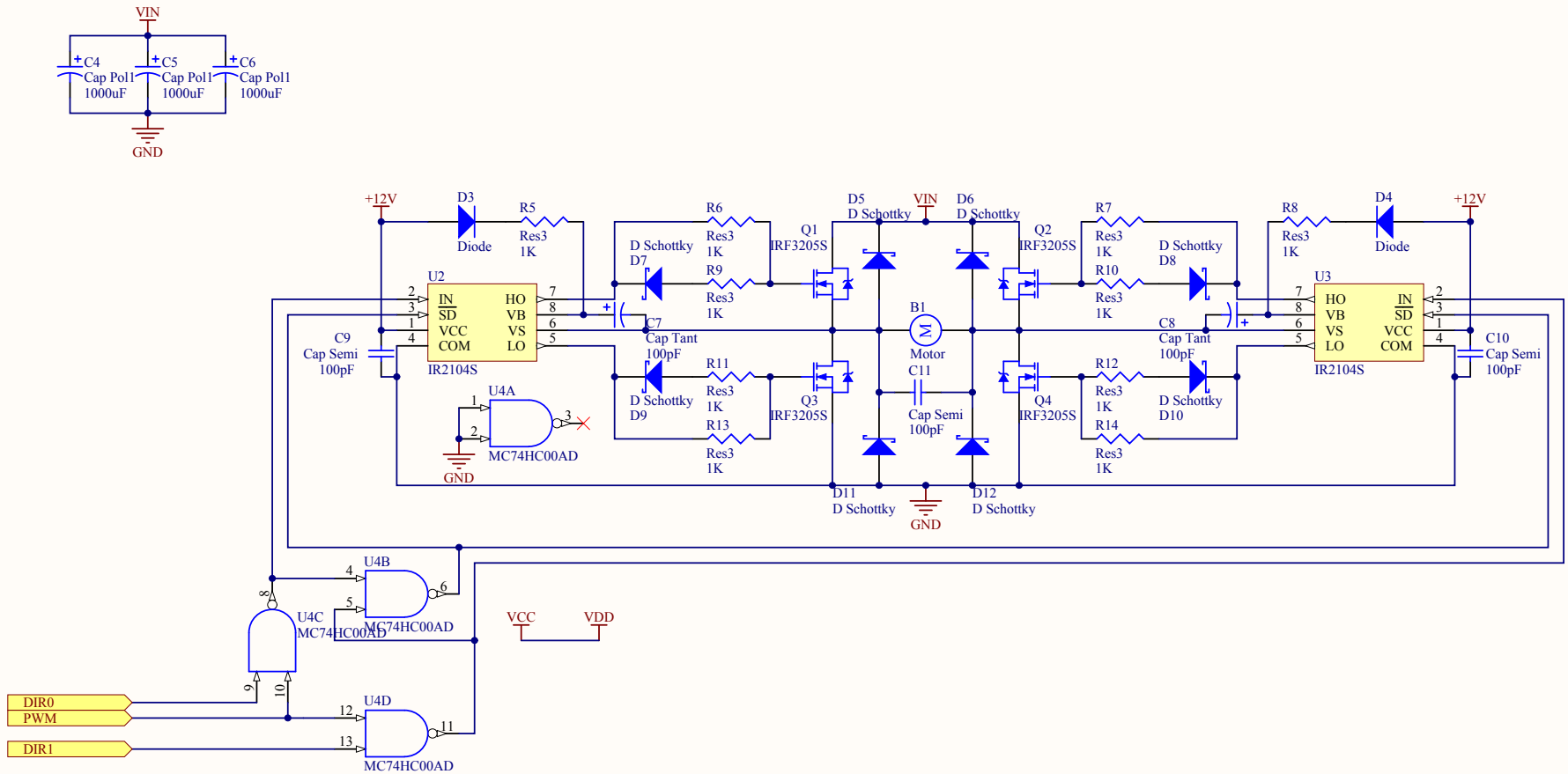
Title	6-axis gryoscope and accelerometer interface	
Size: Letter	Number	Revision
Date: 5/2/2012	Sheet of	
File: F:\Dropbox\My PCBs\ECE445\sensor.SchDoc	Drawn By: Zhangxiaowen Gong	


2850.00?(mil)
2550.00 (mil)
2200.00?(mil)
2500.00?(mil)

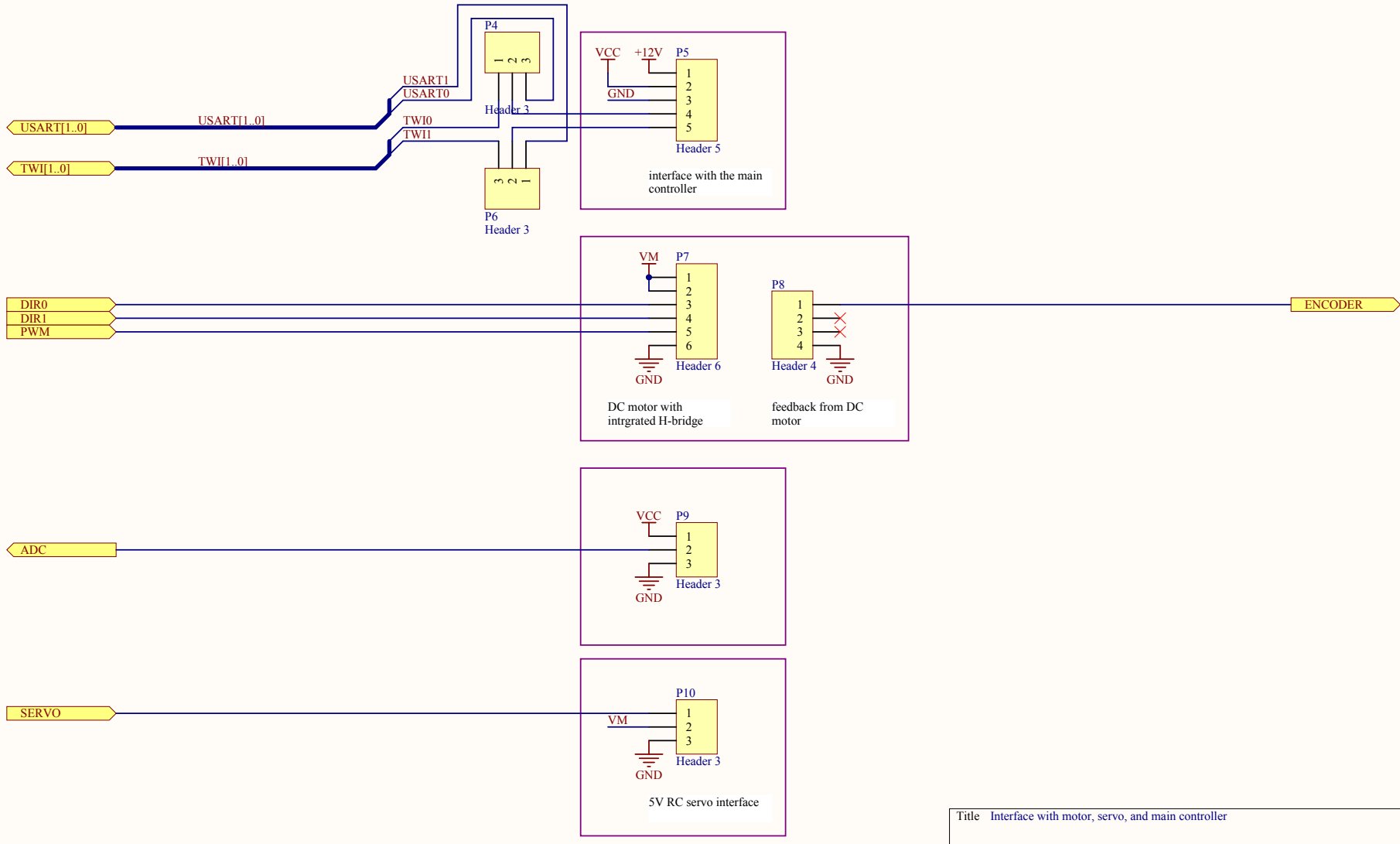
B.2 The Slave Controllers



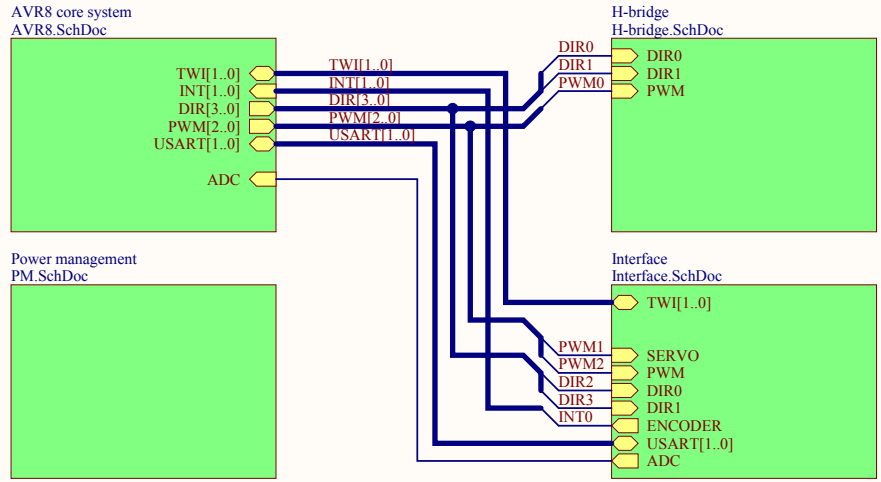
Title AVR8 microcontroller core system		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\AVR8.SchDoc		Drawn By: Zhangxiaowen Gong



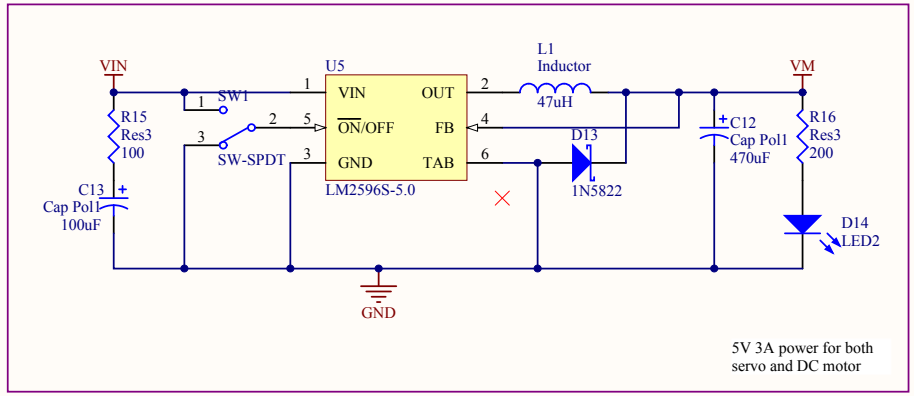
Title H-bridge DC motor driver		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\H-bridge.SchDoc		Drawn By: Zhangxiaowen Gong

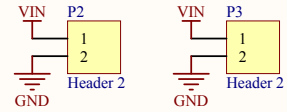


Title: Interface with motor, servo, and main controller		
Size: Letter	Number:	Revision:
Date: 5/2/2012		Sheet of
File: F:\Dropbox\...\Interface.SchDoc		Drawn By: Zhangxiaowen Gong

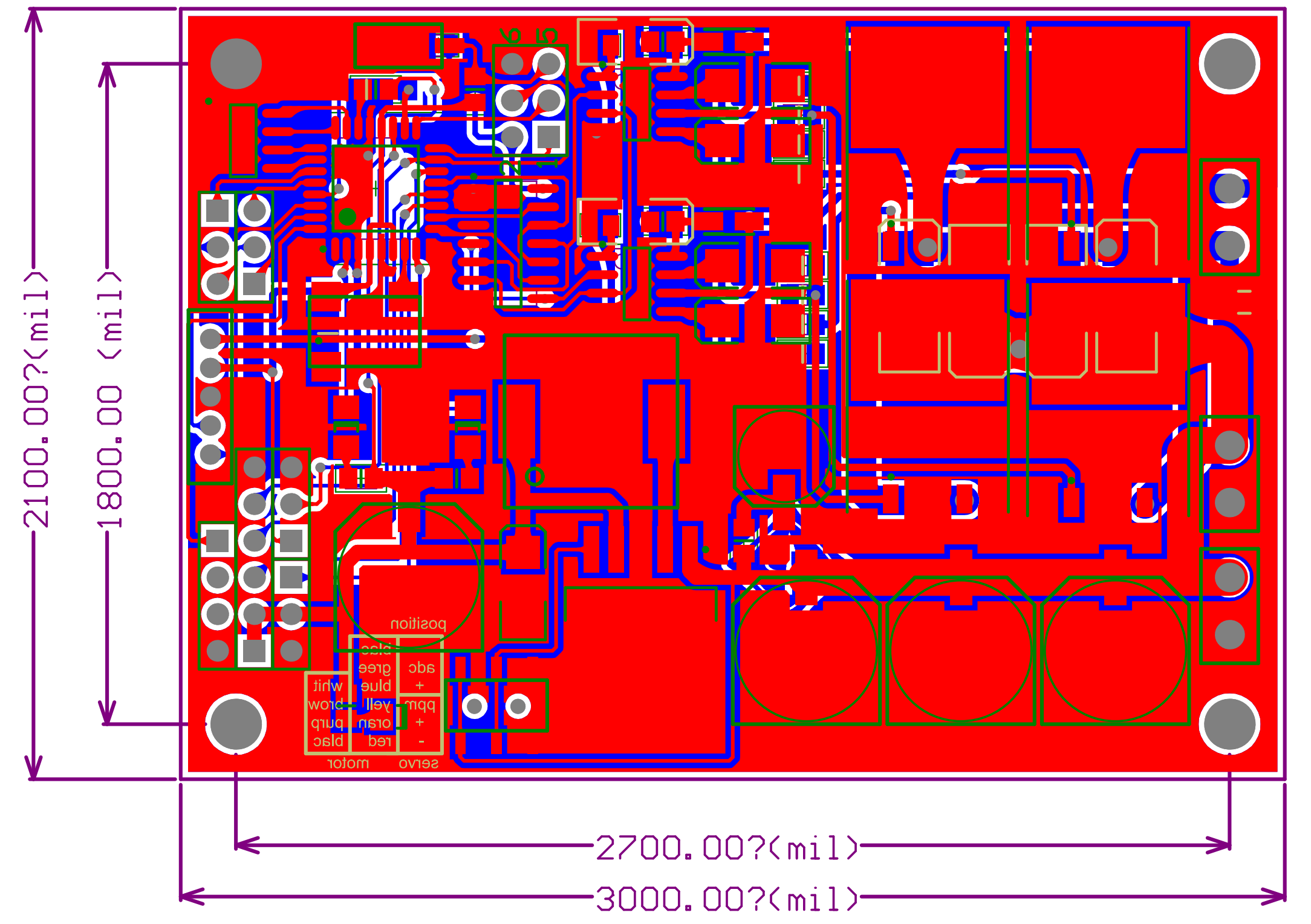


Title Main sheet		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\motor.controller.SchDoc		Drawn By: Zhangxiaowen Gong

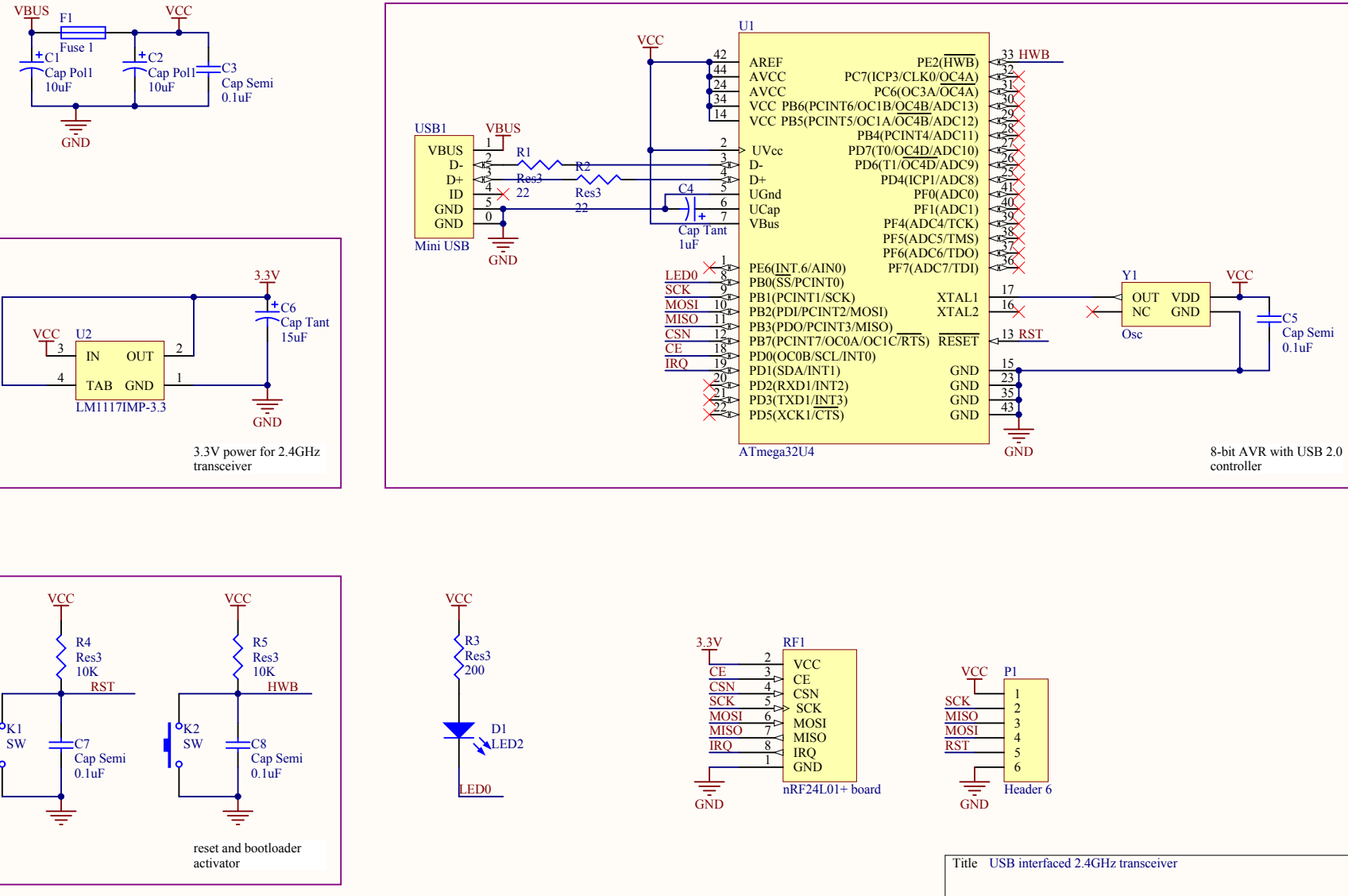




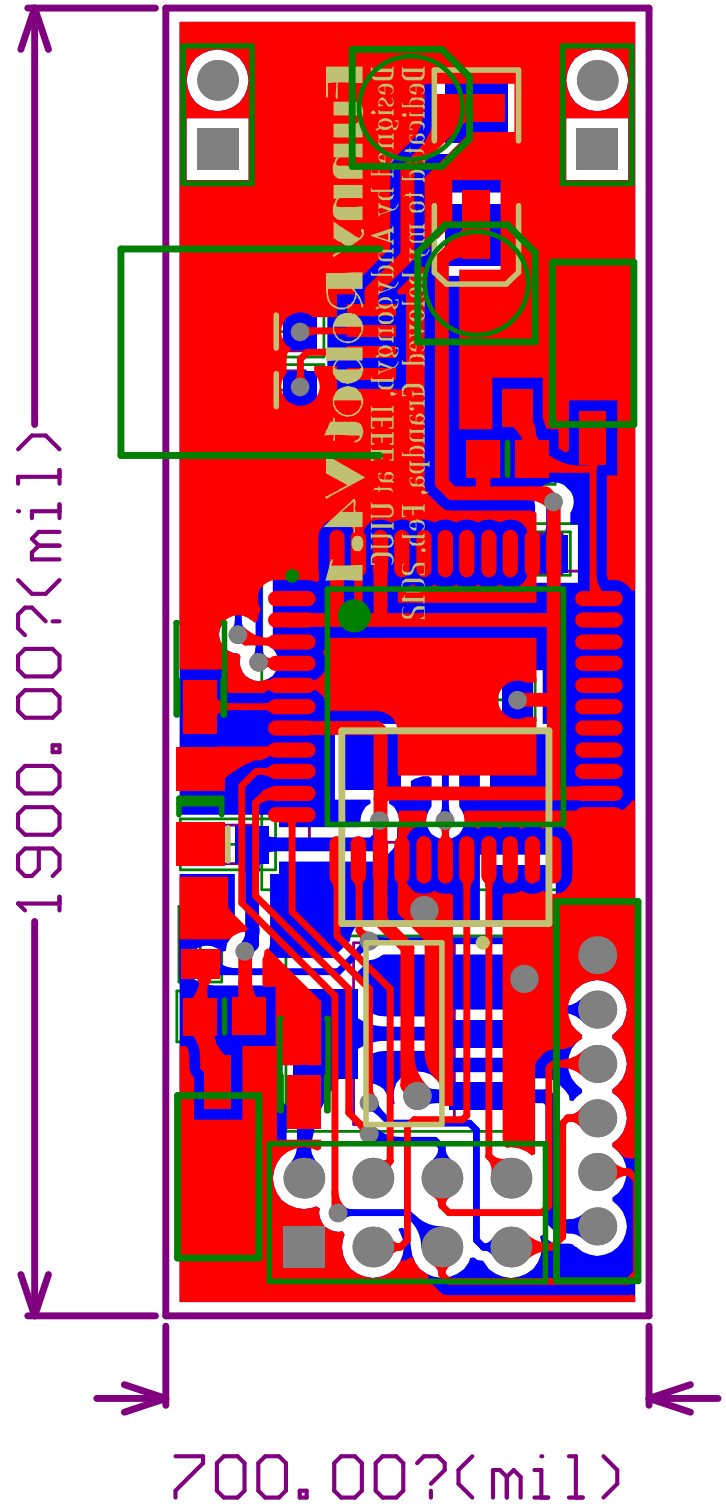
Title Power management providing +5V		
Size Letter	Number	Revision
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\PM.SchDoc		Drawn By: Zhangxiaowen Gong


2100.00?(mil)
1800.00 (mil)
2700.00?(mil)
3000.00?(mil)
position
adc
+
-
ppm
+
-
servo
motor

B.3 The PC Receiver



Title	USB interfaced 2.4GHz transceiver	
Size: A4	Number:	Revision:
Date: 5/2/2012		Sheet of
File: F:\Dropbox\..\Main.Sheet.SchDoc		Drawn By: Zhangxiaowen Gong


1900.00?(mil)
700.00?(mil)