Project Name Here
CSEE 4840 Project
Design Document

Thomas Chau tc2165@columbia.edu
Ben Sack bs2535@columbia.edu
Peter Tsonev pvt2101@columbia.edu

Table of contents:

1) **Introduction**

Our project seeks to achieve the following goal: create a hardware and software co-design that leverages the computational power and interface capabilities of the DE2 board to an autonomously control a stock RC car on a preprogrammed course while avoiding obstacles that might pose a threat for the car.

2) Block Diagram:

1. NIOS II processor module

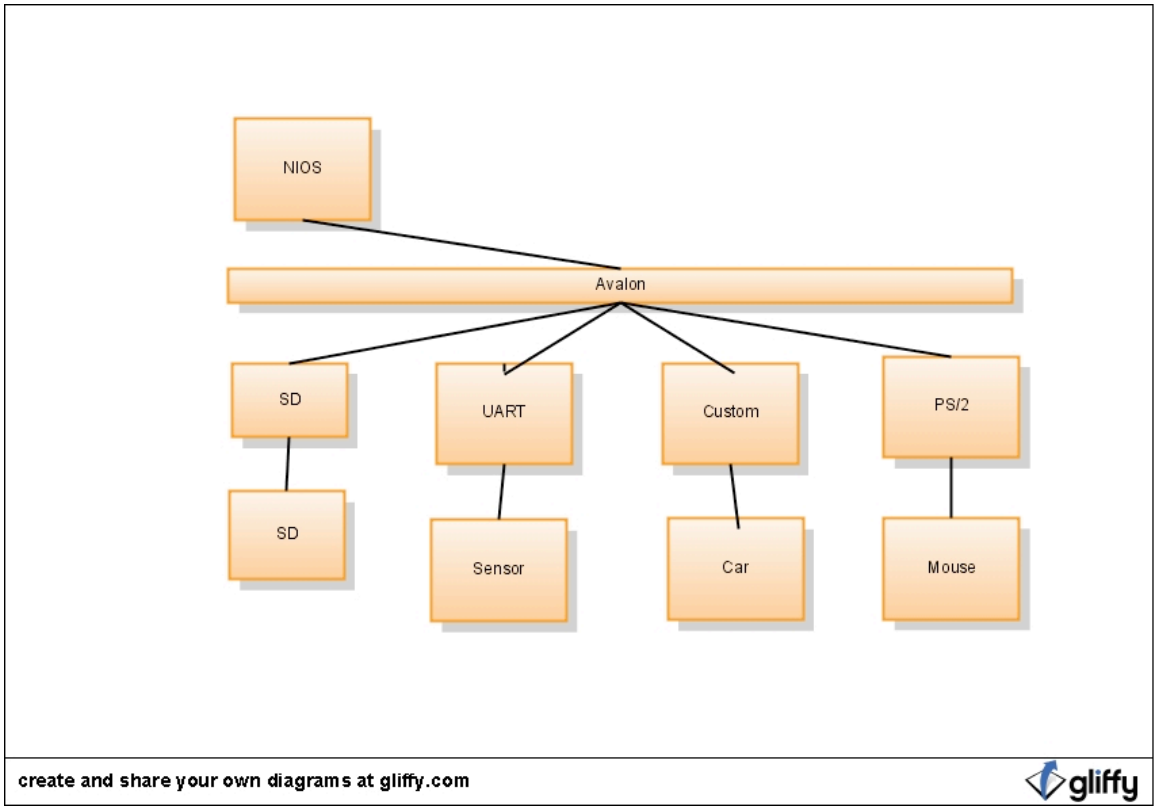
    1.1 Avalon Bus interface

        1.1.1 UART
            1.1.1.1 Proximity Sensor

        1.1.2 SDRAM Controller
            1.1.2.1 8MB SDRAM

        1.1.3 Custom Car Control Hardware (PWM?)
            1.1.3.1 Car Hardware: Motor and Steering Control

        1.1.4 PS/2 Controller
            1.1.4.1 PS/2 Mouse



The figure above describes the basic functionality. The NIOS processor will run a C program residing in the SDRAM. The C code will output left/right and forward/backward commands to two memory-mapped registers in a VHDL component. The latter will constantly read the registers and generate the proper signal for the car hardware (more on

that later). The C program will also constantly poll registers in another VHDL component that uses the RS232 interface to read the proximity sensors. Based on the values of those registers, the C code will be able to recognize upcoming obstacles and take corrective action. The car might also need to be aware of its direction of motion relative to previous directions so that if it turns left/right to avoid an obstacle, it can then make the opposite turn and tell if it is going again in the original direction. To achieve that we use a mouse which will interface through PS/2 and write to registers through a VHDL component. The registers will be then polled by NIOS to get their values into the C code.

3) Proximity sensor : Maxbotix LV-MaxSonar-EZ0

Our project will rely on an ultrasonic proximity sensor to gain awareness of obstacles in its path. Ultrasonic sensors emit sound pulses at a 42khz and wait to detect an echo up to a predefined timeout. Based on the time difference between the emitted pulse and the echo detection, the distance to an obstacle can be calculated. To avoid irrelevant results, most sensors are tuned to operate best for specific tasks. The tuning normally includes adjustment of the width of the sound pulses the sensors emits. For our application, the Communication is done via RS232 protocol, PWM or analog.

For the project purpose, we will use the RS232 format which the DE2 board natively supports.

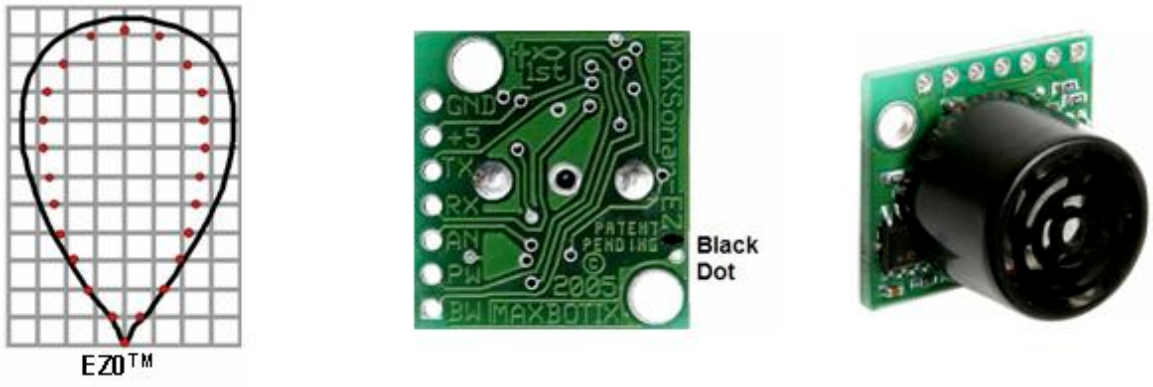


Figure 1 - LV-MaxSonar-EZ0[1]

Figure 1 contains front and back views of the sensors in addition to a graph which show the detection pattern for a large object. Each square on the grid represents 1 foot.

To use the sensor, we would need to connect only three pins: power, gnd and rx.
To communicate over the serial line, the onboard MAX232 chip would bridge between the UART module and the external sensor. The synthesized UART module would have to be set to communicate at 9600 baud, 8 bits, no parity and with one stop bit.
The DE2 board supports all three pins and communication rates without any modifications. To obtain readings from the sensor, all you need to do is connect it to the board. Once the sensor is powered up, it auto calibrates for 100ms. Once done, the sensor will produce measurements once every 49ms. Each measurement is outputted as a string consisting of the letter 'R' followed by the distance to the nearest obstacle in inches.

Serial communication is made possible by the NIOS II processor's UART module. The software controls the UART through the use of five memory mapped 16 bit registers.

[1] Image Taken from http://www.trossenrobotics.com/maxbotix-lv-maxsonar-ez0.aspx

Tx and Rx registers are used to read and transmit data, while the Control and Status registers are used to set and get the information related to the operation of the device. In addition, a dedicated devisor register is used to determine the baud rate of the module. In addition the above registers, the module supports interrupts in the event polling is not proffered. figure 2 below describes the timing diagram for one word transmitted over the UART line. This information is not needed for the current project as the exact behavior of the UART is abstracted thanks to Altera's built in support for serial communications.

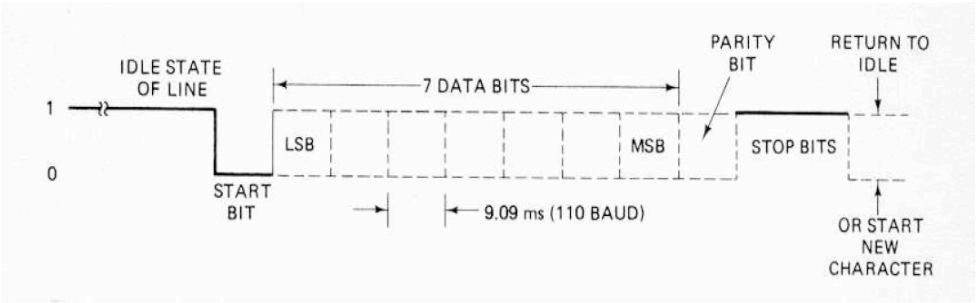
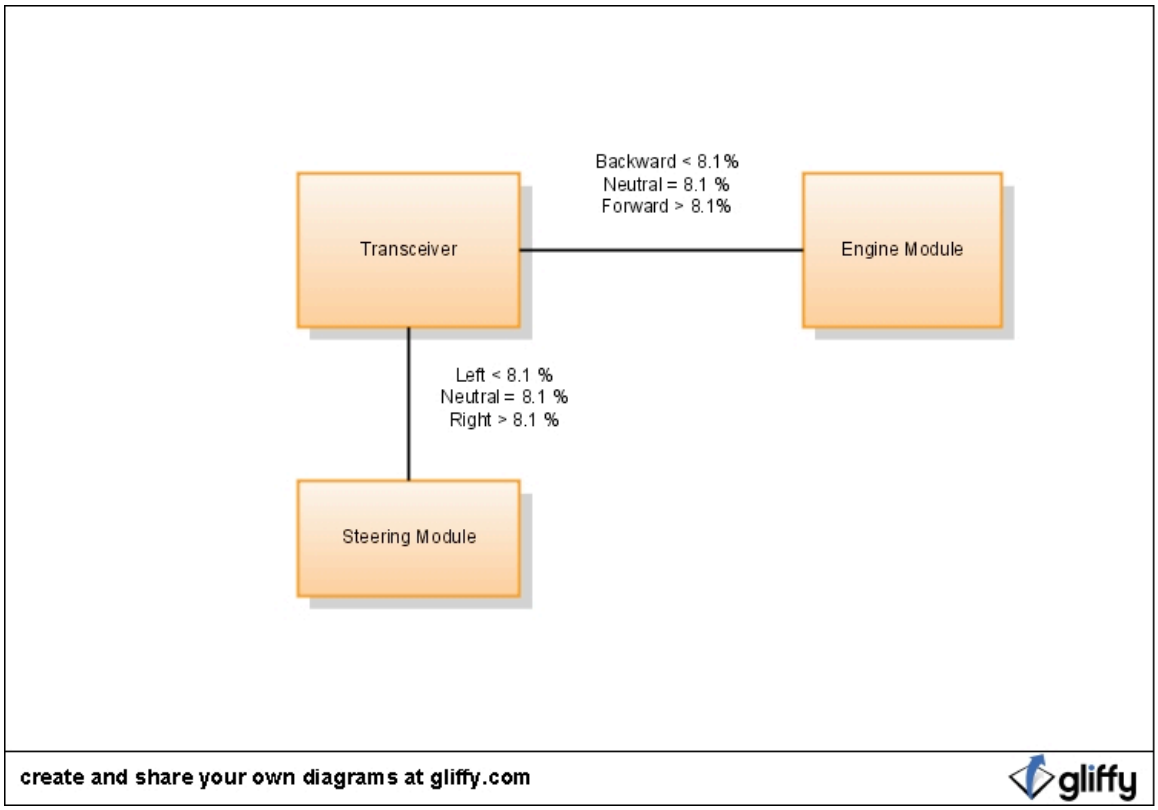


Figure 2 – RSR232 timing diagram[2]

[2] Image taken from UCSB's ECE153b lecture notes, by Professor Butner. http://vader.ece.ucsb.edu/ece153b/handouts/L15-Serial1.pdf

4) The Car:

The diagram below illustrates the hardware modules of the car and their interconnections. The percentages designate the duty cycle of the controlling signals for different commands (left/right and forward/backward). All of these will be now explained.

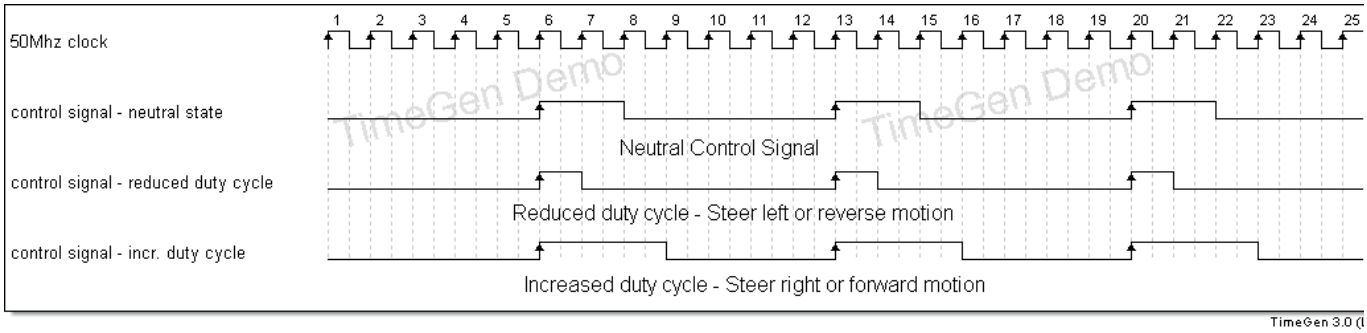


The DE2 needs to interface with the car hardware in order to control it. As it is, the car has an RF transceiver, steering module, and speed control unit. There is one connection from the transceiver to the steering module and a second connection from the transceiver to the engine module. Each connection uses three wires. Two of the wires are power (0.5V) and ground and the third one carries the control signal to the corresponding modules. Thus, in our case, we can leave all power connections intact and just drive the two control wires that leave the transceiver and go to the steering and speed control modules. Both control signals use the same waveforms, but are independent of each other. Each signal is a square wave of a constant frequency of 50Hz. Information is transmitted by changing the duty cycle of the wave (e.g. pulse width modulation). In the neutral state (when the car is doing nothing), the waveform is square with duty cycle of 8.1%. If there is a left/right or forward/backward command, the only thing that changes is the duty cycle on the corresponding control wire. It swings between 5.9% and 11%. Thus, if the car is to turn right, we have to drive the control wire for the steering module with a square waveform of duty cycle greater than 5.9%. If we want to turn left, we change the duty cycle to less than 5.9%. To go forward or backward, we drive the control wire of the speed module in the same way. This technique is called pulse width modulation and is

very useful because it enables one to use a single bit connection to control an analog device.

To enable NIOS to control the movements of the car, we need to connect a VHDL component to the Avalon bus and to a peripheral on the board (most likely the expansion header). The component will accept memory-mapped instructions from the processor and can use the 50MHz clock to derive a square output signal of the right frequency and duty cycle. We need three pins from the output interface – one ground, one for the steering control signal, and one for the engine control signal.

The figure below illustrates an exaggerated timing diagram. The control signal is supposed to be 50 Hz, and the clock – 50Mhz. When the car is motionless, both control signals will resemble the second signal in the figure (the first one below the clock). If it is going forward and left, the control signal for the engine will be like the signal on the bottom, and the control signal for the steering module would be like the third signal below.



We will need some external hardware to get the output signal to the voltage swing expected by the car hardware – 1/4V. The output voltage of the expansion header is either 3.3V or 5V and since the car hardware doesn't have to talk back to the board, we only need to worry about reducing the voltage to 1/4V.

5) Mouse and PS/2 Controller:

To provide the car with displacement feedback, we decided to explore the use of a cheap and readily available position tracking technology: the mouse. the DE2 board can support mice using both USB and PS/2 standards. To keep development overhead for the mouse component low, we chose to use the PS/2 connection. In addition there are two available technologies for the mouse hardware: trackball and optical. The optical interface is more accurate and might be possible to use without any the need to maintain a physical contact with the ground. Otherwise, a trackball mouse can be used instead, which will require physical contact with the ground at all times.

Communicating with the mouse is done with over a serial line and is achieved using only two signals: clock and data as shown in figure 5 below.

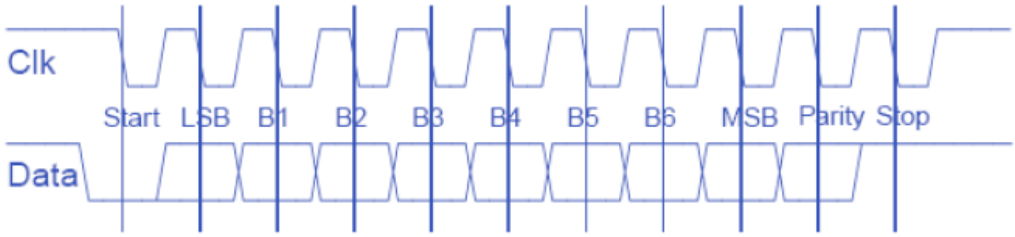


Figure 5 - Mouse communication timing diagram[3]

This communication pattern is used to transmit data packets consisting of 3 bytes. Each packet contains X and Y displacement values, 1 bytes each, in addition to 1 byte of status information such as sign bits, button state and overflow. The exact structure of a data packet is show in figure 6 below.

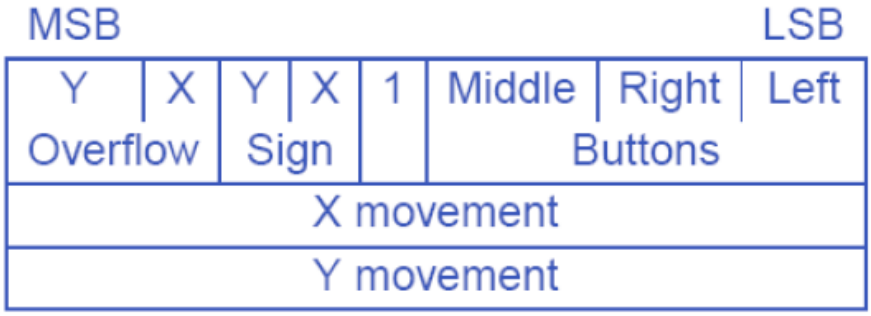


Figure 6 - Mouse data packet structure

[3] Figure 5 and 6 Taken from professor Stephan Edwards PS2 Lecture notes found at http://www1.cs.columbia.edu/~sedwards/classes/2009/4840/ps2-keyboard.pdf

6) Milestones:

1. Component interface and control:
   by this milestone, the device drivers for all systems components should be written and tested. The drivers should abstract the following:
   Car control: This driver should provide call to control the forward/reverse movement in addition to left and right steering control
   Proximity sensor: The driver will initialize the proximity sensor and collect data from its registers.
   Mouse: the mouse driver will obtain the change in the X and Y axis and contrast the readings with a timer to produce speed readings in addition to total displacement.
2. write software: software must be written to control the car movement along a path. this includes speed and direction control. In addition, sensory information must be processed to avoid obstacles. Finally, the software should be tested for reactiveness to insure the systems meet all deadlines, failure to do so may result in stale proximity readings which may cause the car to crash into obstacles.
3. project completion, the car can navigate through an obstacle course, our goal is to do so with speed.