**MARS ROVER**

**Voice controlled vehicle**

**Abstract:**

The main function of this project was to incorporate what we have learned and developed in the previous projects and create something new. We started with the pulse width modulation (pwm) and motor control IP that we had created for project 2. In an effort to enhance this we chose multiple motors and decided upon a vehicle that could demonstrate our acquired knowledge. In the latter part of the term we studied redundancy, priority, and analog to digital conversion. We decided to incorporate these into our design. The last aspect we integrated for a challenge and enjoyment was utilizing voice commands as a method for control. While time constraints prevented developing voice control directly on the Nexys4 board, we opted for laptop processing while adding the wireless communication aspect to the project. Our path ultimately proved us successful in obtaining our proposed goals while granting us the opportunity to shine by completing our stretch goals as well.

**Proposal and Hardware:**

Our proposal was to build a wireless motor controller. The Nexys4 FPGA will accept inputs and then “control” in real time the variable motor speed output. We wanted to try Bluetooth as our preferred means of “remote control.” We were asked to include the vehicle collision avoidance as a required component of our design.

We started with a plexiglass base with a means to attach two motors and a skid swivel stabilizing wheel. To control these motors we used two HB3 Digilent motor controllers. We instantiated two copies of our designed IP from Project 2 to enable, set direction, and choose output speed based on a generated pulse width modulated (pwm) signal. These were connected in our generated Microblaze block. In addition we ended up including one other output which was a pwm signal that controlled a hitec HS-300 360° servo motor. The servo acted as a base for a solar charged searchlight and when activated was capable of rotating to any position.

The input hardware consisted of an Xbox Kinect, a MAXSONAR Pmod, a BT2 (Bluetooth) Pmod, a pair of Pololu Wixels, and an Arduino Mega. In addition we utilized the functionality of the Nexys4 board by using the ADC converter and the on board accelerometer. The control algorithm is discussed below.

**Control Algorithm:**

The control algorithm incorporates our different interface methods in a priority that also grants us redundancy. The lowest priority or base commands are the voice activation controls. This path takes us from talking through the Kinect device into voice processing utilizing the Microsoft Kinect SDK. We then run the created application in Visual Studios while opening a serial port link. This link is comprised of a pair of Pololu Wixels. At the Wixels’ core is the CC2511F32 microcontroller from Texas Instruments. Built upon this is a 2.4 GHz radio and USB connectivity. This supplies the wireless link between our laptop and the mobile Nexys4 board that will be mounted on the vehicle. To interface the Wixel with the Nexys4 we opted to run through another piece of hardware, the Arduino Mega. The Mega has multiple built in serials and allows us to use the Wixel as a simple serial passthrough. The Mega is also a complete microcontroller and we opted to actually process the serial string into its binary representation that we then passed onto the Nexys4 gpio pins. This binary code was decoded in the Cyclic Execution loop of the Nexys4 control program and used to drive the vehicle.

Built as the next level of priority and as a protection was the obstacle collision detection and avoidance. We felt this addition was necessary to prevent damage especially since the vehicle was being controlled remotely and interference could potentially cause the driver to lose control of the vehicle. This gave us a sort of Fail-Safe default so that our vehicle could stop itself to prevent harm if it lost connection to the host. To actualize this protection we chose the MAXSONAR Ultrasonic Range Finder Pmod from Digilent. If the front sensor detected an imminent collision object it took over the motor commands and stopped the motors. This in turn led to another problem. While keeping the vehicle safe it also disables the motor and prevents you from recovering the vehicle. In an effort to address this we added another higher priority designed with super user access.

This took the form of connection through the Bluetooth BT2 Pmod also from Digilent. The Bluetooth can be easily accessed from a laptop through the various serial terminals such as putty, tera term and arduino. We also discovered that any ready-made Android Bluetooth app such as Arduino Bluetooth Controller gives instant access to control our vehicle. This in turn gives us another aspect of redundancy control as the interface is compatible over a broad range of devices. The Bluetooth interface as a super user does contain its own risk. While allowing you to back up, turn or even drive through an obstacle, you are capable of crashing and doing harm not only to the vehicle but potentially to others. Therefore one must exercise caution when entering this mode and assuming control of the vehicle in this manner.

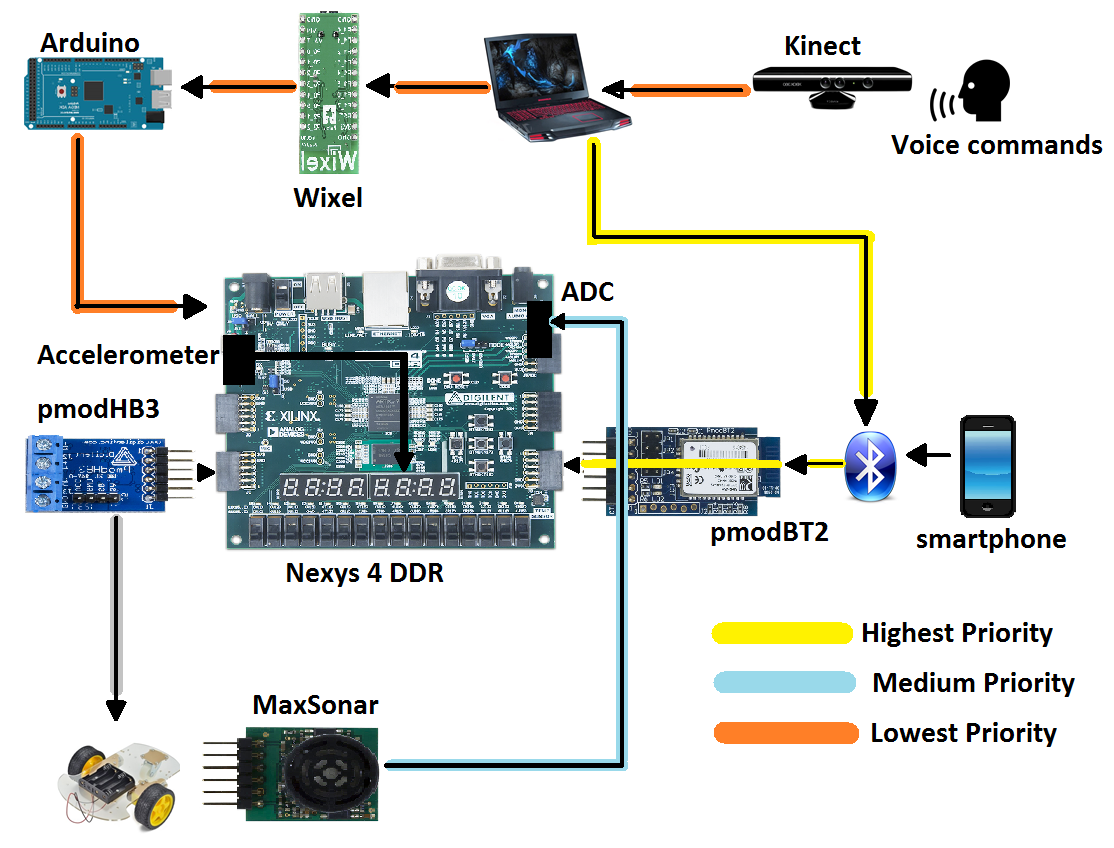


Figure 1. Block Diagram showing priority

**Vehicle**

The base Vehicle was an *INSMA Motor Smart Robot Car Chassis Kit*. In order to mount the battery, the Nexys4, and house all the connections we built up. We added two additional layers of plexiglass and then decided on a Mars Rover theme. Aluminum side panels were added as well as our countries’ flags. The motor cables had forced our battery to be further toward the front than desired and we added a front wheel skid system to stabilize the vehicle if it came to a sudden stop. This gave us sufficient room to place all of our components while maintaining a sleek design.

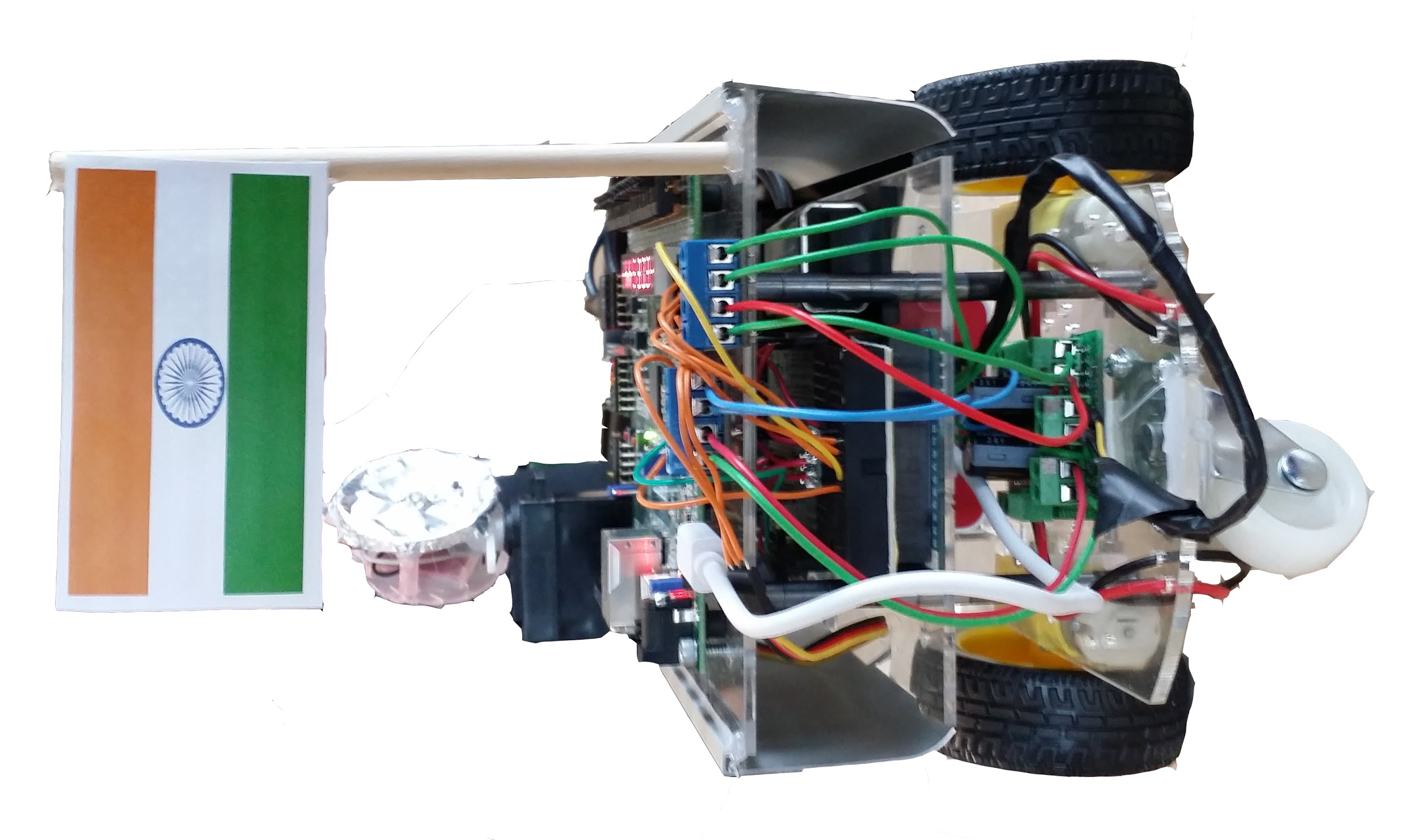
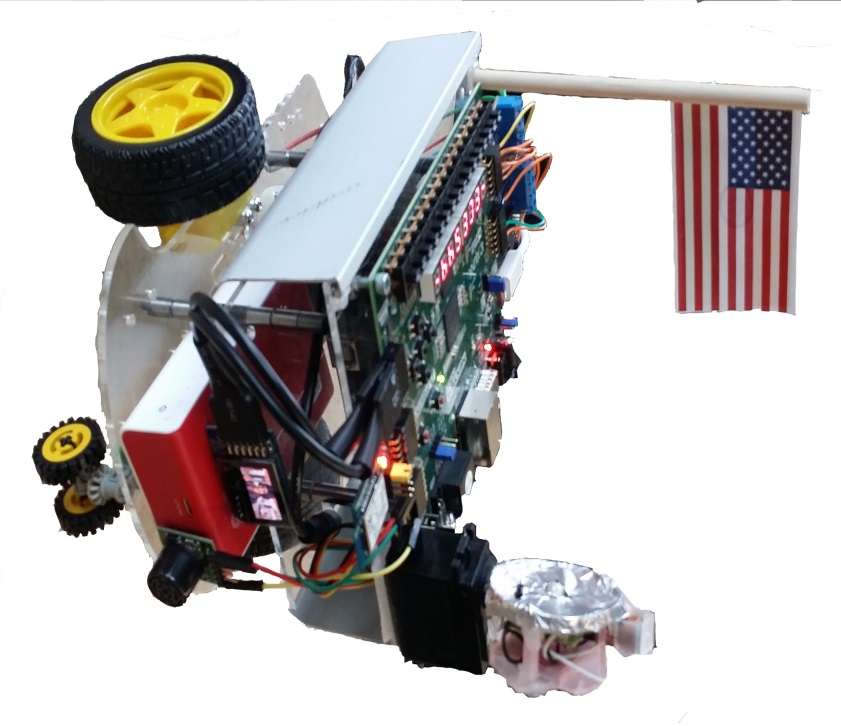


Figure 2. Front View Figure 3. Rear View

**Arduino:**

An Arduino Mega microcontroller was used in this project as a serial interface between the laptop processing the voice commands and the Nexys4. One Wixel was connected to the laptop via USB with the com port connected through Visual Studios. The second paired Wixel interfaced with the Arduino Mega through a shield. The shield simplifies the interface by auto connecting the Wixel to the main “Serial” in the Arduino. This gives access to pin 0, the RX, and pin 1, the TX, while providing voltage shifting due to the 3.3 V limit on the Wixel and the 5 V output of the Arduino. The Mega also has predefined Serial1 and Serial2 pins and this allows for a serial pass through which we had initially implemented. We felt that the hardware deserved some task and we then used it as a decoder that translated the string char input into a binary form that the Nexys4 could interpret as a simple read from the gpio.

- 0001 for Right turn,

- 0010 for Left Turn,

- 0011 for slow,

- 0100 for fast,

- 0101 for stop,

- 0110 for reverse mode,

- 0111 for Servo motor. “Controlling the search light rotation”

These values are read by Nexys4 through its GPIO JC[0:3] after one more voltage shift back to 3.3 V and the respective action is implemented on the Rover.

**Kinect:**

The Kinect provides a lot of versatility and interface options into the design. While we started this design with the idea of having voice control, the Kinect could in the future provide gesture recognition. The Kinect was interfaced through Visual Studios and uses the libraries provided by the Microsoft Kinect SDK. This gives us the ability to link a voice command with any desired form of serial output. Since our operational control was simple, we chose simple character strings. The C# program uses *Microsoft.Speech.Recognition* which gives us access to the classes contained within and makes the interface trivial for adding our own speech.

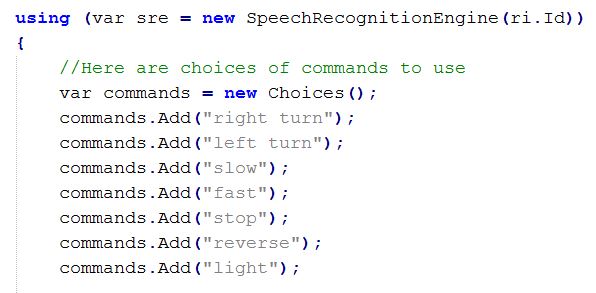


Figure 4. Adding voice commands

The sounds are processed and then given a confidence rating that gives you flexibility in deployment. We chose a baseline rating of >.7 although, as you can see from the next clip, our confidence was actually over .9.

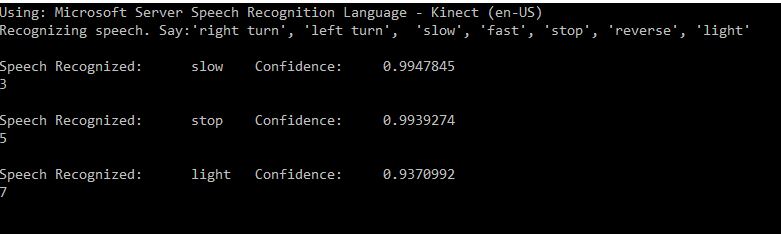


Figure 5. Console output from Visual Studios

The output is then fed to a configurable com port which gives us wireless access to our Mars Rover.  *SerialPort send = new SerialPort("COM12", 9600);*

The defaults for parity and stop bits match our current interface and we now have connected the Kinect to our project.

**Collision Avoidance:**

The collision avoidance feature implemented for our final project used a MaxBotix HRLV-MaxSonar sensor. This is an ultrasonic sensor with noise rejection, automatic calibration, and factory calibrated beam patterns. It is a high-resolution sensor with a resolution of 1mm with a 10 Hz reading rate. The sensor can detect objects from 1mm to 5 meters. The sensor interface outputs PWM, analog voltage, or TTL serial digital. For this project, analog voltage was chosen as our interface format.

The analog-to-digital (ADC) converter inside the Artix-7 was used to measure the voltage. The ADC is a dual-channel 12-bit converter capable of operating at 1 MSPS (Million Samples per Second). In addition, it has 16 differential signal auxiliary channels, however the Nexys4 development board only utilizes AUX channels 2,3,10, and 11. The input range of the ADC is 0-1V. However, this did not a pose a problem since we are using voltages much less than 1V. The ADC IP core was instantiated with an AXI Lite interface for ease of use with the Xilinx XADC driver and the settings according to Figure 6.

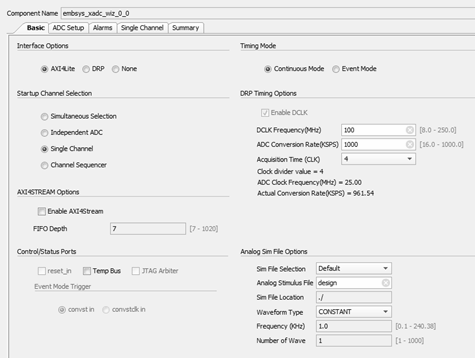


Figure 6. XADC IP core configuration.

The actual driver used in the SDK was the sysmon driver which can be used to monitor system voltages and temperatures. It is also capable of reading the AUX channels of the XADC and is straightforward to use. Only a few lines of code were needed to read the ADC, code snippet is shown in Figure 7.

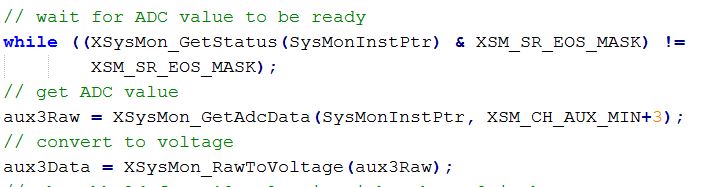


Figure 7: Code snippet to read the raw data from the ADC and convert to voltage.

Once the raw data is read from the ADC, it is then converted to a voltage. The distance at which the bot engages the collision avoidance system is approximately eight inches which corresponds to about 0.18V.

**Accelerometer:**

The accelerometer is used to measure the acceleration of the bot. This feature has no real impact on the performance of the bot. However, it is a nice feature to be able to see the acceleration. The original idea behind using the accelerometer was to measure velocity by integrating the sensor data. Yet, after some research, it was determined that method to measure velocity was very inaccurate.

The onboard accelerometer ADXL362 provides 12-bit output resolution that can measure ranges of ±2*g*, ±4*g*, and ±8*g* and has a SPI interface. For this project the range was limited to just ±1g because pulling more than 1g in the horizontal direction was not possible with the bot. The driver for the accelerometer was done completely in hardware for one reason only: because it was a last minute addition to the project and there was no time to write a low level driver for this device. The device driver was stripped out from the Digilent Nexys4 demo project, which was entirely done in VHDL. The driver takes care of the SPI interface and formatting the data into a useable format. It was packaged into an IP module and the 9-bit sensor data was output into a GPIO block to be read by the Microblaze. According to the VHDL commented code, -1*g* corresponds to 0, 0*g* = 255, and 1*g* = 511 decimal. Using this information, an equation was formulated to convert the raw data to *g*’s as shown below.



Code snippet is shown in Figure 8. The final output is then shown on the 7-segment display.

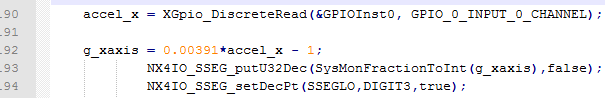


Figure 8. Calculating the acceleration

**SSEG**:

We had thought about displaying the velocity on the OLED screen but after our demo we realized just how hard it would be to see, especially on a moving vehicle. We then opted to display the velocity on the seven segment display which was clear enough to actually be read from a distance as the vehicle was moving. As this was not in our .bit for the demo, we have included a video of its performance along with the hard to see motion indicators of the OLED.

<https://youtu.be/WpCXBphi_Qg>

**OLED:**

The OLED is a Pmod from Digilent that we had used in our previous Projects. Our first idea was to have two headlights, each possibly with its own indicator. As we planned out our board we quickly realized that we were running out of Pmod ports and after some initial testing we were unhappy with the intensity of the light provided. After we had added the search light we still wanted to use the OLED and decided to just have a visual representation of the turn signals. Through a counter in our Cyclic Executive we created a state machine of sorts that would “blink” our indicators when we were executing a turn.

**Bluetooth:**

The Bluetooth Pmod BT2 uses the RN42 Roving Networks chip to create the wireless connection with the Nexys4 through UART using the AXI4-Lite interface. The device defaults to slave mode and allows any Bluetooth to detect and pair with it. Digilent provided the IP interface to the Microblaze, so adding the device to the block was trivial. We did end up utilizing the jumpers for functionality. Jumper 4 forces the device to run at a 9600 Baud rate and this made initialization more predictable. We choose to operate in polled mode and after initialization the functional interface was extremely simple.

BT.JPG

Figure 9. Capture and save any new data over the Bluetooth

Due to the ease of device connectivity and the multitude of devices that can operate using Bluetooth, this seemed ideal for our super user priority. This was demonstrated in our demo when the com port had become corrupted and we could just connect from our phone and take control of the system.

**Problems:**

1. While extremely useful, it took us some time to figure out the Bluetooth interface. At first we had tried interrupt mode and while we were able to receive data from the Nexys4 through the Bluetooth we were unable to write to it. We then dove deeper into the schematics and found it easier to use the jumper 4 setting to set the baud rate instead of the software. Even after we had connected and were able to transmit in both directions we encountered a problem when opening the Bluetooth com port from Visual Studios. After we had initiated a send, the system would lock up and generate an IO exception within the next 5 seconds. This persisted and eventually led us to our redundant control through the Arduino.
2. The other obstacles we overcame were more tool-related. We found it difficult and not at all predictable when trying to update the hardware platform in the SDK. Quite often we had to delete both the Board Support Package and sometimes even create a brand new application.

**Results:**

We were able to build a responsive system that operated wirelessly through voice commands. In addition, we successfully interfaced the MAXSONAR to provide collision avoidance. We were even able to add extra functionality to our design, namely redundancy, servo operation, and successful integration of the accelerometer. Given more time we hopefully could have pressed the gesture commands, possibly through some semaphore flag code. We came through this exercise with a fuller understanding of implementing a useful Microblaze package and we are looking forward to exploring future possibilities.

**References:**

* <https://youtu.be/WpCXBphi_Qg>

* <https://reference.digilentinc.com/reference/pmod/pmodmaxsonar/reference-manual>

* <https://reference.digilentinc.com/reference/pmod/pmodbt2/reference-manual>

* <https://reference.digilentinc.com/_media/reference/pmod/pmodbt2/pmodbt2_sch.pdf>
* Vivado 2015.2 CUSTOM IP PART I - Creating and Packaging Your IP Vivado <https://www.youtube.com/watch?v=BEQXV3eAZNs>
* <https://www.xilinx.com/support/documentation/university/XUP%20Boards/XUPNexys4DDR/documentation/Nexys4-DDR_rm.pdf>