Group 4 Final Project

**Introduction**

Our project uses the gyroscope and bluetooth peripheral devices to wirelessly control an iRobot Create. The gyroscope determines the rotation angle of the iRobot, treating the FPGA board as a steering wheel. On-board buttons determine the speed and direction (forward or reverse) of the iRobot wheels. This information is sent to a laptop over the bluetooth peripheral as a UART. The laptop interprets the FPGA data to drive the iRobot over a serial connection.

**Design and Procedure**

***FPGA Design***

The FPGA design was based a top level module using the provided gyroscope and bluetooth modules for a Nexys 3 board as submodules. In essence, we took the z-axis data from the gyroscope and wired that information to the bluetooth modules using a separate signal; although we had planned to transfer the provided VHDL modules onto a Nexys 5 board, we ran out of time due to various difficulties we encountered. For one, our initial idea was to write a module to parse the z-axis data into useful commands to be sent to the iRobot; however, learning VHDL was extremely difficult and we were unable to successfully do so. Thus, we instead implemented a noise filter, using trial and error, to extract only useful input from the gyroscope since we did not want to flood the bluetooth connection with insignificant user input or errors from the gyroscope sensitivity that could overwhelm the iRobot.

User input was mapped to two buttons: one button controlled whether the iRobot would move forward or backwards and the other button controlled whether the iRobot was moving or not. To turn the iRobot, the gyroscope would sense user turn motions by measuring changes in z-axis rotation and send it to the bluetooth module for sending. Turning the board clockwise sent an “R” command to turn the iRobot to the right and turning it counter-clockwise would send an “L” to turn the robot to the left. Moreover, since we did not remove any of the provided gyroscope modules, various Nexys 3 board inputs was mapped to unused functions, such as the seven segment display, the start switch and the reset button; the exact mappings can be observed in the included ucf file.

*Gyroscope Peripheral*

The Pmod GYRO is a three-axis gyroscope that was configured to communicate with the FPGA via SPI. The gyroscope sends data for angular momentum across all three axes at a maximum resolution of 2000 data points per second. We used the basic gyroscope set-up, but isolated the z-axis data for a steering function. Since the gyroscope is extremely sensitive and constantly produces small non-zero values, we ignored smaller magnitudes.

Currently, the project only detects angular momentum. Ideally, the FPGA would work like a steering wheel where turning is based on angular position instead. We attempted to produce a simple integrated result of z-axis data to create z-position. However, due to the sensitivity and error accumulation, the z-position overflowed over time, regardless of buffer size. For an accurate angular position measurement, an accelerometer peripheral would likely need to be added to the project. The two peripherals would create an Inertial Measurement Unit that would be accurate for both small and large time-frames.

Reference VHDL code for the gyroscope can be found at:

https://reference.digilentinc.com/\_media/reference/pmod/pmodgyro/pmodgyro\_demo.zip

*Bluetooth Peripheral*

The Pmod BT2 uses the RN-42 chip for Bluetooth communication over UART. We used VHDL to set up the peripheral as a serial UART instead of the usual physical serial port. The chip was paired with the control laptop using the laptop’s Bluetooth hardware. Pairing the peripheral created two serial COM ports on the laptop - one for sending, one for receiving. When the FPGA has data to send (one of four direction commands), the Bluetooth peripheral sends this data to all paired devices. It operates at a baud rate of 9600 with eight-bit words.

Reference VHDL code for the Bluetooth device can be found at:

https://reference.digilentinc.com/\_media/nexys/nexys3/nexys3\_ise\_gpio\_uart.zip

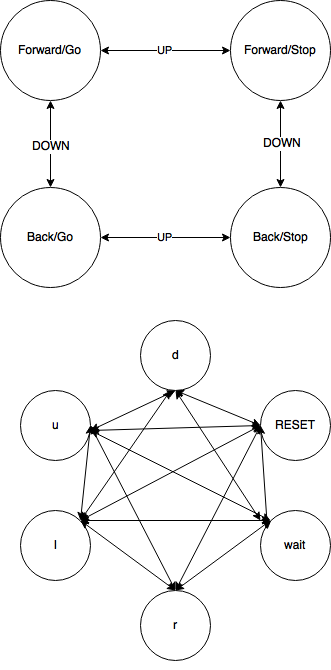
***iRobot Design***

*Laptop*

All programming on the laptop was encapsulated in one Python program. We decided to use Python for ease of development, given the short lifespan of this project. Python code is also portable, making development on multiple OS’s simple. The laptop is manually connected to the FPGA’s bluetooth module using the interface provided by the computer’s OS. From there, the bluetooth connection is used like any other serial connection. The PySerial library makes serial connection management easy in Python. The following code snippet shows the PySerial library in action. We wrote this script to test reading data from the bluetooth serial connection.

1. # Connection configuration
2. ser = serial.Serial(
3. port='/dev/tty.RNBT-6427-RNI-SPP',
4. baudrate=9600,
5. parity=serial.PARITY\_NONE,
6. stopbits=serial.STOPBITS\_ONE,
7. bytesize=serial.EIGHTBITS
8. )
10. ser.isOpen()
12. **while** True:
13. **if** (ser.inWaiting()>0):
14. bytesToRead = ser.inWaiting()
15. data = ser.read(bytesToRead)
16. **print**(data)
18. ser.close()

The commands received from the FPGA are ‘u’, ‘d’, ‘l’, ‘r’, and ‘RESET’. The ‘u’ and ‘d’ commands control the forward and backward movement of the iRobot. The forward and backward movements are controlled by a FSM. The ‘u’ and ‘d’ commands simply change the state of the iRobot. The 4 states are outlined in the FSM figure below. ‘u’ and ‘d’ correspond to UP and DOWN on the FSM edges. The starting state is Forward/Stop.



The ‘l’ and ‘r’ are commands are used to rotate the iRobot left and right respectively. At the Python level, iRobot rotation is stateless. In other words, a rotation command simply causes the iRobot to execute the rotation, either left or right at 1 degree. Unlike the FSM controlled forward/backwards movements, the program has no idea what angle the iRobot is moving afterwards.

Finally, the ‘RESET’ command received from the FPGA simply notifies the Python program to elegantly close its connections to both the FPGA and iRobot and exit the program.

*iRobot Control*

The iRobot is controlled serially with a default baud rate of 57600 and eight-bit words. Op-codes are used to perform various functions. For our project, we only used opcodes that control the wheels and movement of the iRobot. The python script sends present strings of op-codes. For our project, we created six basic commands: INIT, STOP, FWD, BWD, LEFT\_TURN, and RIGHT\_TURN.

* INIT: Initializes the iRobot by starting its open interface and setting it to Full Mode, granting access of other functions
* Wheel control commands
  + [137] [Vel. High byte] [Vel. Low byte] [Radius high byte] [Radius low byte]
  + STOP: Sets both velocity bytes to zero and both radius bytes to zero
  + FWD: Sets velocity to +100 mm/s, both radius bytes to zero (drives straight forward)
  + BWD: Sets velocity to -100 mm/s, both radius bytes to zero (drives straight backward)
* Wait angles: waits for iRobot to turn a specified degree
  + [156] [Dist. High byte] [Dist. Low byte]
  + LEFT\_TURN: Sets velocity to +100 mm/s, with the minimum positive radius. Then waits for iRobot to turn one degree counterclockwise.
  + RIGHT\_TURN: Set velocity to +100 mm/s, with the minimum negative radius. Then waits for iRobot to turn one degree clockwise

The iRobot only turns one degree for each turn command in order to maximize control responsiveness. Additionally, we experienced an error where the iRobot would turn indefinitely if the wait angle was set to a large amount and multiple turn commands were sent. We hypothesize this is due to the iRobot’s opcode buffer filling up, putting the iRobot in an indefinite ‘wait’ state with continuous turning.

A full list of opcodes can be found at:

https://www.irobot.com/filelibrary/pdfs/hrd/create/Create%20Open%20Interface\_v2.pdf

**Conclusion**

Overall, this project was an excellent learning experience. Not only were we able to create a project we were proud of, but we learned of many new technologies, such as bluetooth, VHDL, and serial connections. Although we encountered many technical difficulties, broken serial connectors, bluetooth modules and unknown VHDL errors, it exposed us to the hardware design process and how difficult it can be.