|  |  |
| --- | --- |
|  |  |
|  |  |

**Implementing Serial Communication (UART) in DE2 Bots**

Nisarg Himani

Sunny Patel

Thomas Saekao

Mohit Tandon

George Vellaringattu

ECE 2031, Digital Design Lab

Section L05

Georgia Institute of Technology

School of Electrical and Computer Engineering

Submitted

April 9, 2014

**Executive Summary**

This design report details technical information regarding a project that enables UART to allow wireless serial communication between a base station and a robot. To successfully accomplish these tasks, a program will be written that enables the base station and robot to communicate with each other. This robot will act as a warehouse inventory robot that receives base station commands to carry out with a given time limit. An assembly code will be programmed to perform basic and advanced robot initialization, and a functional UART will be improved for reliability, usability, and functionality. Movement of the robot will be in conjunction with odometry and movement in the Cartesian coordinates. Modifications are made to the UART interface code to work efficiently and to not break any other functionality as well as add in the ability to collect data only when the data is new and valid. No buffer is required for the UART modification since the team can store and process the data as soon as it is available.

**Implementing Serial Communication (UART) in DE2 Bots**

# **Introduction**

This project seeks to implement wireless serial communication to enable the sending and receiving of data using the UART between the DE2 robot and the base station. The purpose of the UART is to convert bytes from parallel communication to serial communication and vice versa so that the two devices will be able to understand each other. To successfully achieve these tasks, a program will be written to request, receive, and act upon commands from the base station. The goal of the robot is to act like a warehouse inventory robot and carry out received base station commands to pick up and drop off jobs at specific warehouse coordinates, and signaling completion back to the base station in order to provide feedback to the user. Odometry, rather than sonar, will be used to navigate the robot according to the base station commands.

# **Technical Approach**

## Project Description

The goal of this project is to manipulate a block diagram to complete the UART interface to meet the following specifications. Firstly, data needs to be sent to the robot over the UART in order for it to execute the jobs given by the base station, and then responses need to be sent by the robot and received by the base station that the robot outputs as messages. Additionally, an assembly code will be written that performs the following tasks:

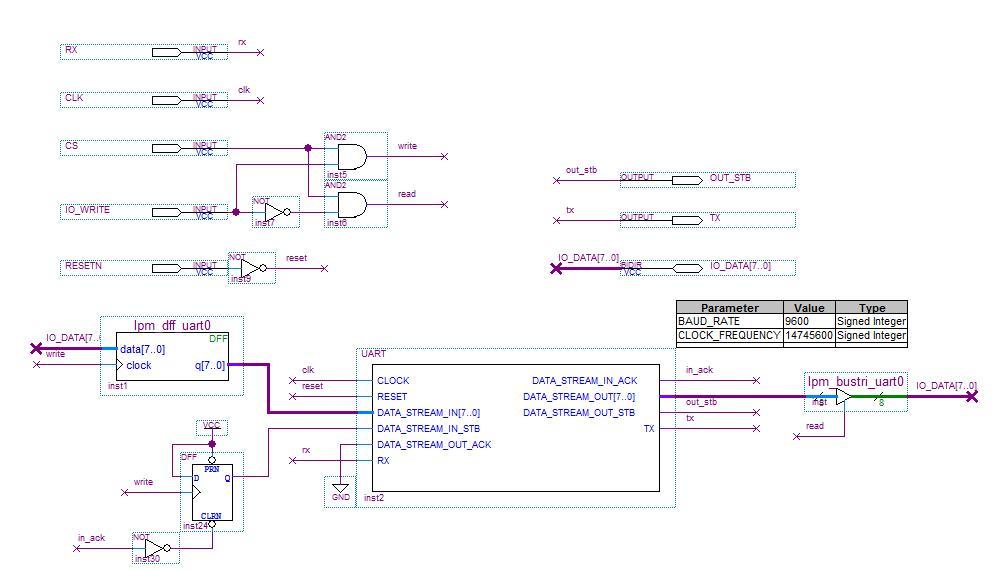
1. Allows the robot to carry out basic movement such as moving forwards, backwards, and rotating between 0 and 360 degrees so that the robot is constantly facing either the positive X direction or the positive Y direction.
2. Provides a method of reaching the pickup and drop off destinations according to the jobs received by the base station.

## Overall Design

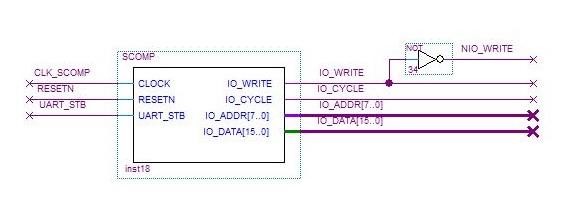
In order to properly implement the design, the SCOMP and UART interface will be manipulated to meet the specifications listed above. The project will be approached by implementing the UART and implementing the movement separately for ease of testing. Having error-free communication between the robot and the base station be consistent is vital. The provided UART interface already has an output called DATA\_STREAM\_OUT\_STB that is currently unused. It will be connected as an input to SCOMP so that it can be read and processed via the VHDL code. Odometry will be used so the robot will simply move parallel to either the X axis or the Y axis when picking up or dropping off its jobs.

**UART Implementation**

A SCOMP peripheral to interface with the UART and an assembly code will be written to communicate with the base station and act as a warehouse robot. The DATA\_STREAM\_OUT\_STB output will be utilized, causing the interface itself to assert that output when there is a valid byte of data. The UART interface can be seen in Figure 1. The output is connected to SCOMP as an input to be read from the written assembly file and then retrieve the data only when it is asserted. The result will be the base station sending a byte of data and the robot sending back bytes of data containing its response. The base station sends back a minimum of two bytes of data. The first byte is the response and the second byte is the checksum used to ensure the data is proper. In the case of jobs, the first four bytes are the coordinates for the pickup and drop off locations for the jobs. The 5th byte is the checksum. The SCOMP interface can be seen in Figure 2. If data is retrieved when it is not yet asserted, then the data may not be valid. Therefore, the output is ensured to be asserted before the data is inputted so that the data is valid.



**Figure 1.** UART interface used to communicate with robot with added output OUT\_STB.



**Figure 2.** SCOMP interface modified with additional input UART\_STB.

**Odometry Implementation**

Movement will be demonstrated with odometry rather than sonar. The inventory robot will receive jobs from the base station and carry them out within a given time. The basic concept of navigation is to first move in the X direction and then in the Y direction. As set in the assembly language code in Appendix A, the robot will move in the X direction, rotate 90 degrees counter-clockwise and then move in the Y direction to reach the pickup location. Once the pickup location is reached, the robot is facing in the positive Y direction.

Now to navigate to the drop off location, it will move in the Y direction first and then rotate 270 degrees counter-clockwise (to face the X direction) and then continue moving. Rotating the robot 270 degrees counter-clockwise will be used instead of rotating it 90 degrees clockwise because the robot is more responsive and reliable when rotating counter-clockwise compared to when rotating clockwise. Furthermore, if the robot must navigate towards the negative X direction or the negative Y direction, the robot simply moves backwards, in reverse. Therefore, the robot is remaining in two states, for it is always facing either the positive X direction or the positive Y direction. This allows quick movement between two locations without spending time for the robot to rotate. In either the original method for movement or the method stated in the contingency plan, a new medium speed will be created to allow the robot to move quickly and still increase its accuracy on landing in the correct position.

As the code in Appendix A suggests, odometry is used to move the robot forward or backward depending on where the final location is. This method moves a specific distance that is pre-calculated when movement is required after the robot receives jobs through the UART wireless communication.

# 

# **Management Plan**

A significant portion of the management plan comprised of the Gantt Chart. Appendix A contains a Gantt chart showing the major tasks and subtasks along with the milestones for the progress of the group since the project assignment was first introduced. The major tasks included the project introduction period, the UART implementation, the odometry execution, and testing and executing. Typically, milestones indicated the start of the next major task, allowing the group to distinguish the focus of the project.

Given that the group proposed to use the output of the provided UART interface called DATA\_STREAM\_OUT\_STB and connect it as an input to SCOMP symbol, an alternative method that could be implemented for the UART is that an I/O device will be used. The device will take in the output of the UART interface as the input and output it directly to the assembly language without the need to modify SCOMP. This alternative method will act as the contingency plan.

In addition to the plan for the UART implementation, a contingency plan may be used for the odometry implementation. A second method for movement will allow the robot to move to a particular location relative to its current position. It will keep the robot moving either forward or backward until it reaches the desired location. In this way, if the robot overshoots its position, it can stabilize and come to the right location.

**Appendix A: Modified Assembly Language Code for SimpleRobotProgram.asm**

-- SimpleRobotProgram.asm

-- Implementation of odometry movement in a portion of the assembly -- language code for the DE2 bot

-- Powers of 5

-- ECE 2031 L05

-- 04/09/2014

; Move to pickup location

; Start Pickup Location - X =================

LOAD FFast

STORE DirSpeed

LOAD X1

OUT SSEG1

SUB CurX

JPOS GETDISTX

JZERO PICKX

LOAD RFast

STORE DirSpeed

GETDISTX:

LOAD X1

SUB One

STORE Temp

LOAD TwoFeet

MULT Temp

STORE MovX

XMOVE:

LOAD DirSpeed

OUT LVELCMD

OUT RVELCMD

IN XPOS

STORE CXPOS

LOAD MovX

SUB CXPOS

OUT SSEG2

JPOS XMOVE

JNEG XMOVE

PICKX:

CALL STOP ; Stop the bot

LOAD X1 ; Update X value

STORE CurX ; Store new X value

CALL Rotate90 ; Rotate 90 anticlockwise

CALL STOP ; Stop the bot

; End Pickup Location X =======================

; Start Pickup Location Y ======================

; Now the bot is at the correct X value for Pickup.

; Move it along Y

LOAD FFast

STORE DirSpeed

LOAD Y1

OUT SSEG1

SUB CurY

JPOS GETDISTY

JZERO PICKY

LOAD RFast

STORE DirSpeed

GETDISTY:

LOAD Y1

SUB One

STORE Temp

LOAD TwoFeet

MULT Temp

STORE MovY

YMOVE:

LOAD DirSpeed

OUT LVELCMD

OUT RVELCMD

IN YPOS

STORE CYPOS

LOAD MovY

SUB CXPOS

OUT SSEG2

JPOS YMOVE

JNEG YMOVE

PICKY:

CALL STOP ; Stop the bot

LOAD Y1 ; Update Y value

STORE CurY ; Store new Y value

CALL STOP ; Stop the bot

; End Pickup Location Y ======================

CALL PICKUP ; Tell base station, job has been picked up

; Start Dropoff Location Y ====================

LOAD FFast

STORE DirSpeed

LOAD Y2

OUT SSEG1

SUB CurY

JPOS GETDIST2Y

JZERO DROPY

LOAD RFast

STORE DirSpeed

GETDIST2Y:

LOAD Y2

SUB One

STORE Temp

LOAD TwoFeet

MULT Temp

STORE MovY

YMOVE2:

LOAD DirSpeed

OUT LVELCMD

OUT RVELCMD

IN YPOS

STORE DYPOS

LOAD MovY

SUB DYPOS

OUT SSEG2

JPOS YMOVE2

JNEG YMOVE2

DROPY:

CALL STOP ; Stop the bot

LOAD Y2 ; Update Y value

STORE CurY ; Store new Y value

CALL Rotate270 ; Rotate 270 anticlockwise

CALL STOP ; Stop the bot

; End Dropoff Location Y =======================

; Start Dropoff Location - X =================

XMOVE2:

LOAD RFast

OUT LVELCMD

OUT RVELCMD

LOAD CurX

SUB One

JZERO DROPX

IN XPOS

JPOS XMOVE2

DROPX:

CALL STOP ; Stop the bot

LOAD One ; Update X value

STORE CurX ; Store new X value

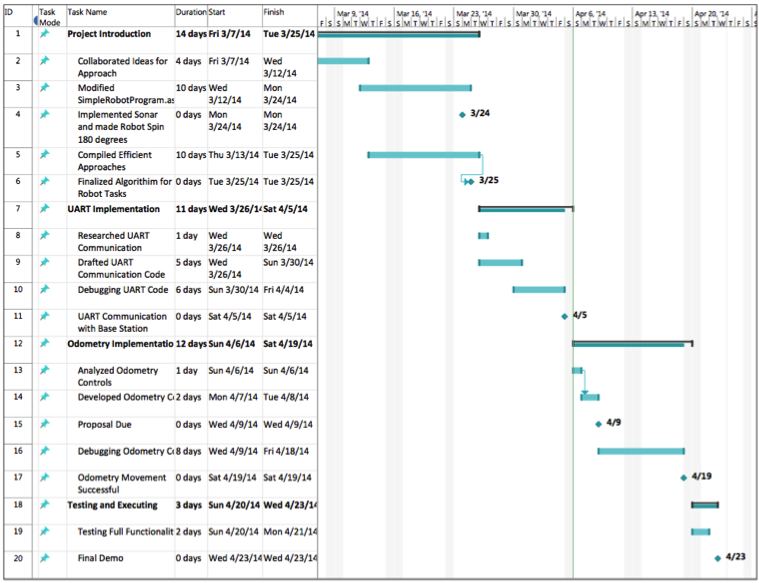
CALL STOP ; Stop the bot

; End Dropoff Location X =======================

CALL DROPOFF ; Tell base station, job has been dropped off

RETURN

**Appendix B: Gantt Chart**



**Figure 1.** Gantt Chart representing the timeline of events with estimated future tasks under the management plan.