**Abstract**

This design report details technical information regarding a project that enabled UART to allow wireless serial communication between a base station and a robot. To successfully accomplish these tasks, a program was written that enabled the base station and robot to communicate with each other. This robot acted as a warehouse inventory robot that received base station commands to carry out jobs within a given time limit. An assembly code was programmed to perform basic and advanced robot initialization, and a functional UART was improved for reliability, usability, and functionality. Movement of the robot was in conjunction with odometry and movement in the Cartesian coordinates. The main problem faced using the UART was that it was vulnerable to lose data or be confused by repeated data of the same value. Modifications were 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 was new and valid. What made this design unique was the fact that no buffer was required for the UART modification since the team could store and process the data as soon as it was available. While the robot successfully completed all eight jobs with fifteen of the 240 seconds to spare, the lack of perfect odometry resulted in several deductions in the raw score. The robot had to be physically picked up and returned to the home square several times. As a result, the project was only moderately successful since the goal of fully-autonomous behavior was not attained.

**Wireless Asynchronous Communication via UART on a DE2Bot**

# **Introduction**

This project implemented 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 was to convert bytes from parallel communication to serial communication and vice versa so that the two devices would be able to understand each other. The presented problem for this project was to implement asynchronous serial communication using UART to communicate between the base station and the robot. To create this communication, manipulating a block diagram to complete the UART interface was necessary. The UART interface was responsible for four main tasks, which included sending and receiving messages to and from the base station, carrying out jobs specified by the base station, and returning home. The goal of the robot was to carry out received base station commands, which specified pick-up and drop-off jobs in the warehouse. It then signaled completion back to the base station in order to provide feedback to the user. Odometry was used to navigate the robot according to the base station commands.

## General Methodology

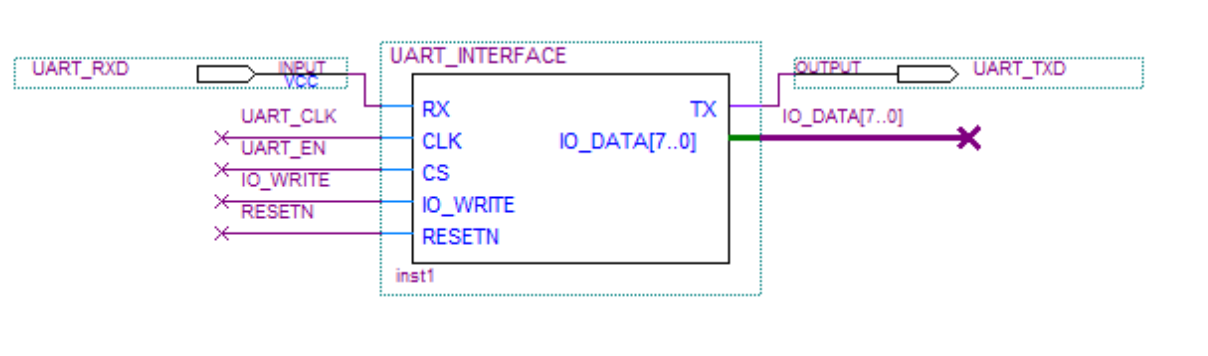
## Project Description

The method in which the UART processed the data from the base station happened in a two-phase process. First, the data was sent to the robot over the UART in order for it to execute the jobs given by the base station, and second, the responses were sent by the robot and received by the base station that the robot outputs as messages. Additionally, an assembly code was written that performed the following tasks:

1. Allowed the robot to carry out basic movement such as moving forward, backward, 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. Provided a method of reaching the pick-up and drop-off destinations according to the jobs received by the base station.

## Overall Design

The provided UART interface was modified from the original, as seen in Figure 1, and an I/O device was created to facilitate the transmission of data between the base station and the DE2 bot over UART. The I/O device created, read the DATA\_STREAM\_OUT\_STB output of the UART interface. This output would get asserted when there is a valid byte of data present in the UART. Once the output is asserted, the data would be retrieved to the assembly code without the need to modify the SCOMP. This would allow retrieval of valid data when required.



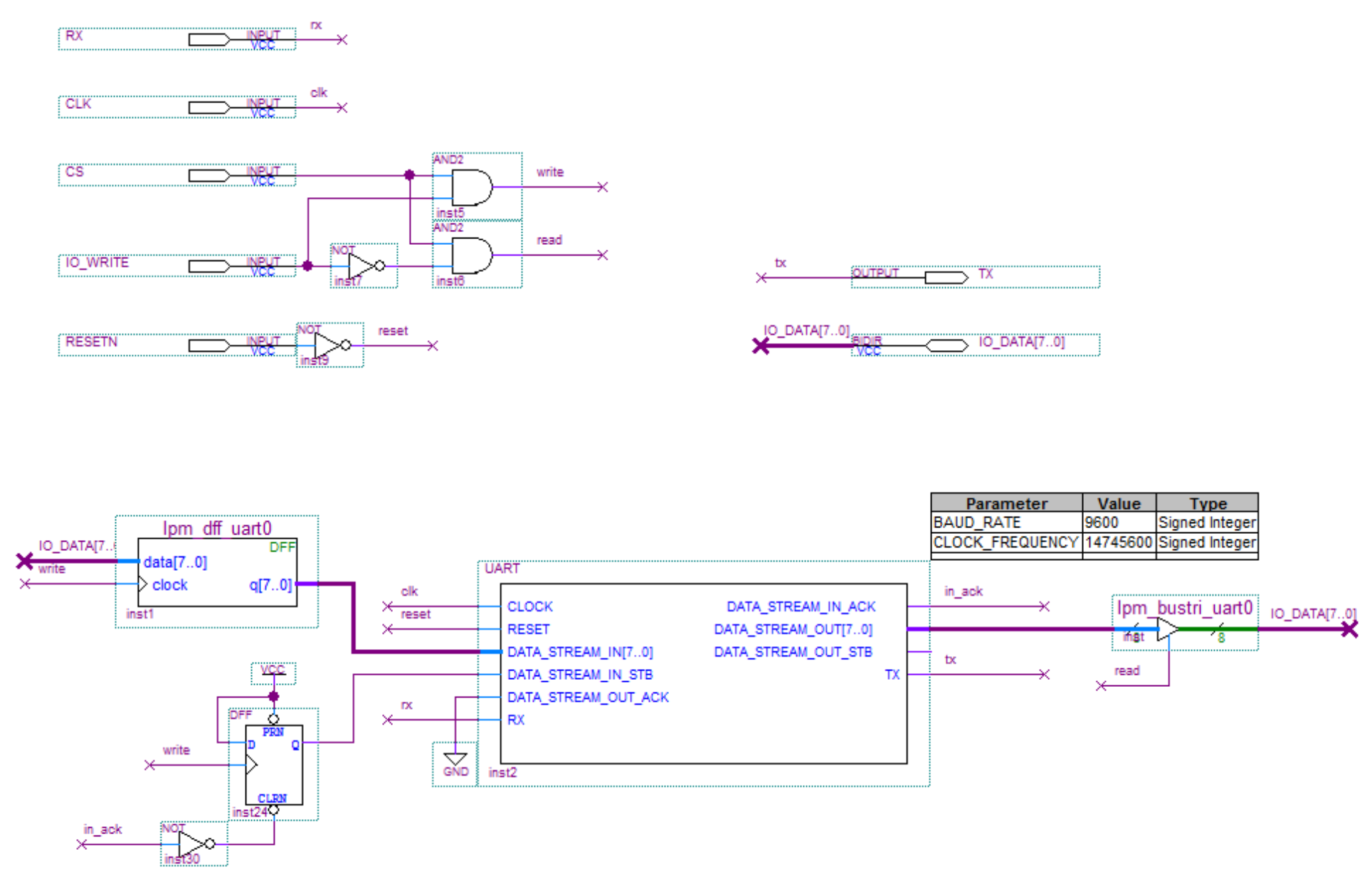
**Figure 1.** Original UART\_INTERFACE used to communicate with robot without any modification.

This design is very unique since a buffer system is not utilized when handling multiple jobs. When the robot began to execute a particular job, it would request data for that job. The X and Y coordinate for that job would be read into the assembly file and stored as variables. Since the data for the other jobs did not affect the current job, data was requested and retrieved only when required. This implementation resulted in the design being very simple and efficient.

For the odometry implementation, the robot moved to a particular location relative to its current position. The robot moved either forward or backward on a particular axis without the need to rotate 180 degrees. The robot moved in the X and Y-axis until the desired location is reached.

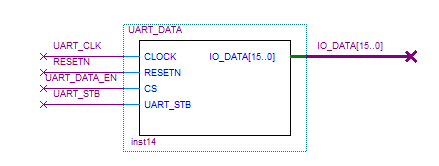
The project was approached by implementing the UART and the movement separately due to ease of testing. Having consistent error-free communication between the robot and the base station was vital for the success of this project. The provided UART interface already had an output called DATA\_STREAM\_OUT\_STB that was unused. Odometry was used so the robot simply moved parallel to either the X-axis or the Y-axis when picking up or dropping off jobs.

**UART Implementation**

An I/O device was created to read in the DATA\_STREAM\_OUT\_STB output of the UART, as seen in Figure 2. 

**Figure 2.** UART interface used to communicate with robot with added output DATA\_STREAM\_OUT\_STB.

This allowed the assembly code to read this output without the need to modify the SCOMP, as seen in Figure 3. This output was asserted when there was a valid byte of data in the UART. When the output was asserted, the data was retrieved and available to be used for further processing and moving the DE2Bot to the required destination. The DE2Bot sent a byte of data and the base station sent back multiple bytes of data containing its response. The base station sent back a minimum of two bytes of data. The first byte is the response and the second byte is the checksum, which was used to ensure that the data was proper. In the case of jobs, the first four bytes were the coordinates for the pick-up and drop-off locations for the jobs. The 5th byte is the checksum. If data was retrieved when it was not yet asserted, then the data cannot be valid. Therefore, the output was ensured as asserted before the data was inputted so that the data was deemed valid.



**Figure 3.** UART\_DATA interface modified with additional input UART\_STB.

**Odometry Implementation**

Movement was demonstrated with odometry rather than sonar because sonar had an inherent incapability of consistently functioning properly. The inventory robot received jobs from the base station and carried them out within a given time. The basic concept of navigation, as seen in Figure 4, was to first move along the X-axis and then along the Y-axis because implementing polar coordinates was inconsistent and time consuming. Additionally, handling rotation was more difficult due to its high sensitivity to small vibrations. As described in the assembly language code in Appendix A, the robot moved along the X-axis, rotated 90º counter-clockwise and then moved along the Y-axis to reach the pick-up location.

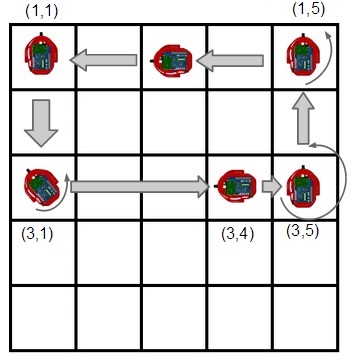


Figure 4. Example movement layout of the implemented algorithm.

Once the pickup location was reached, the robot faced the positive Y-axis. To navigate to the drop-off location, it moved along the Y-axis first and then rotated 270º counter-clockwise (to face the X direction) and then continued moving. Rotating the robot 270º counter-clockwise was used instead of rotating it 90º clockwise because the robot was more responsive and reliable when rotating counter-clockwise compared to when rotating clockwise. Counter-clockwise rotation was implemented over clockwise for consistency and as means for an efficient coding algorithm. Furthermore, if the robot must navigate along the negative X-axis or the negative Y-axis, the robot simply traversed in the reverse direction. Therefore, the robot remained in two states, for it always faced either the positive X-axis or the positive Y-axis. An example movement layout can be viewed in Figure 3 above. This allows quick movement between two locations without spending time for the robot to rotate. As the code in Appendix A suggested, odometry was used to move the robot forward or backward depending on where the final location was. After completing each job, the robot was sent home to reduce error propagation. Since each DE2 bot had different odometry calibrations, specific hard coded values were used with a specific robot that was tested through numerous trials.

# **Project Management**

A significant portion of the management plan was comprised of the Gantt chart. Appendix B 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, the UART implementation, the odometry execution, and presentation, testing, and executing. Typically, milestones indicated the start of the next major task, allowing the group to distinguish the focus of the project. During the last phase of the project multiple tasks were simultaneously executed to ensure that the presentation and final demo were completed in a timely manner.

**Technical Results**

During the two weeks leading up to the final demo, the robot experienced several issues when completing jobs. For instance, it went out of the expected pick-up and drop-off locations. It failed to turn 90° counter-clockwise when trying to return home after each job was completed. The robots turn constants and speeds had to be adjusted to accurately complete jobs. After the final demo, the results of the project were as follows:

* + - 1. Completed all eight jobs within 225 seconds.
      2. Averaged 20 to 25 seconds per job.
      3. Went out of the expected pick-up or drop-off locations seven times.
      4. Physically picked up the robot four times.
      5. Failed to turn 90° once when attempting to go home.

**Conclusions**

The robot successfully communicated wirelessly with the base station in order to carry out specific jobs using the UART. It is recommended to take a methodical approach otherwise overhead occurs. Having the best score in the assigned lab section was due to several major enhancements to the algorithm. The UART implementation was the strongest aspect of the algorithm in terms of communication with the base station. The I/O device created was simple and served as an efficient method of controlling whether or not the data was valid. As for the odometry aspect, the algorithm was fast and efficient in regards to movement speed and rotations. The rotations were carefully tested to create a perfect 90° turn, which accounted for the momentum of the robot leading to the overshoot in the turn angle.

On the other hand, the algorithm had several flaws, which proved to be detrimental to the final score. The odometry calculation was inconsistent and even having adjusted for these values, it still proved to be unreliable. The robot would undershoot or overshoot the pick-up and drop-off locations, which would lead the robot to higher error propagation. The robot was very sensitive and sometimes the robot would skip a subroutine since a condition was not met. This was shown during one of the jobs in which the robot forgot to turn 90° counterclockwise to head home after completing the job.

In the future, implementing polar coordinates could enhance the design of the algorithm. The algorithm used in this design moved along the X and Y-axis, but the use of polar coordinates would have the robot move along the hypotenuse of the triangle. Additionally having the robot finish one job and move directly to the next job improves the algorithm without having to return home after every job. Implementing both the polar coordinates and the continuous job execution would substantially save time.