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

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

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 28, 2014

**Table of Contents**

Abstract 1

Introduction 2

General Methodology 2

Project Description 2

Overall Design 3

UART Implementation 4

Odometry Implementation 5

Project Management 7

Technical Results 7

Conclusions 8

Appendix A: Final SimpleRobotProgram.asm Code

Appendix B: Final Gantt Chart

Appendix C: Brainstorming and Logbook Check-off Sheets

**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.

**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

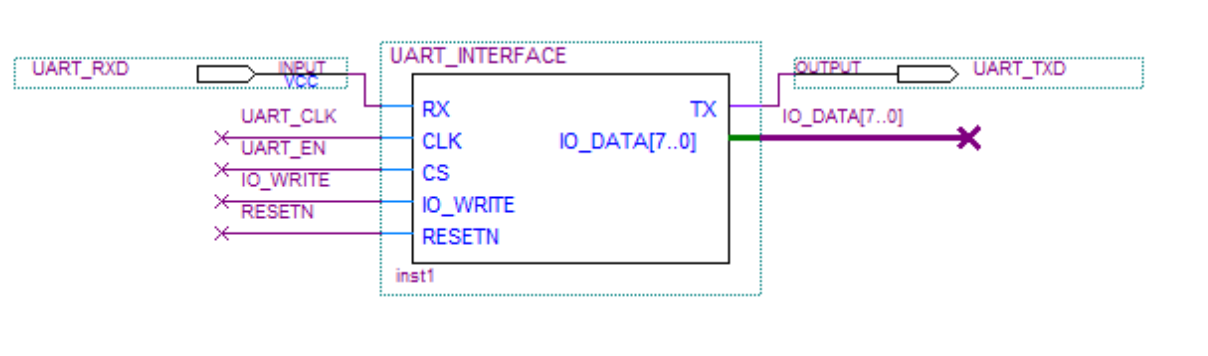
## Project Description

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.



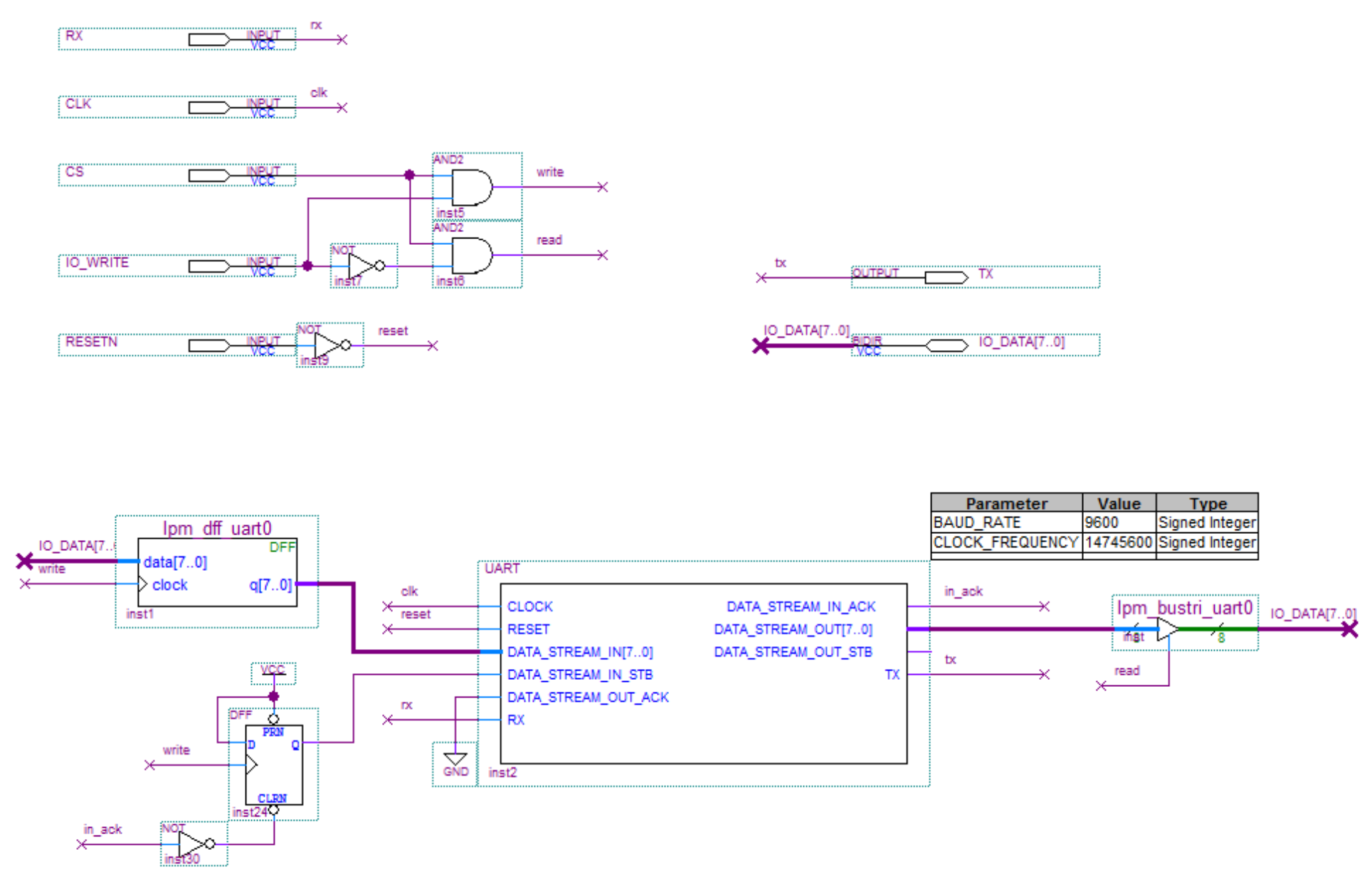
**Figure 1.** Original UART\_INTERFACE used to communicate with robot without any modification.

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.

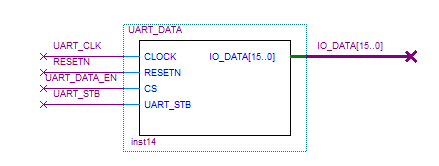
The project was approached by implementing the UART and the movement separately due to ease of testing. Having consistent error-free communication between the robot and the base station was vital for the success of this project. The provided UART interface already had an output called DATA\_STREAM\_OUT\_STB 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.

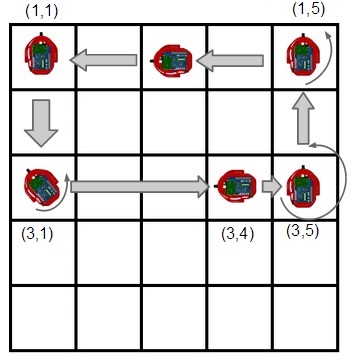


Figure 4. Example movement layout of the implemented algorithm.

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 the design of the algorithm could be enhanced by implementing polar coordinates. 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.

**Appendix A: Final SimpleRobotProgram.asm Code**

-- SimpleRobotProgram.asm

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

-- Powers of 5

-- ECE 2031 L05

-- 04/25/2014

ORG &H000 ;Begin program at x000

Init:

; Always a good idea to make sure the robot

; stops in the event of a reset.

LOAD Zero

OUT LVELCMD ; Stop motors

OUT RVELCMD

OUT SONAREN ; Disable sonar (optional)

OUT BEEP

CALL SetupI2C ; Configure the I2C

CALL BattCheck ; Get battery voltage (and end if too low).

OUT SSEG2 ; Display batt voltage on SS

LOAD Zero

ADDI &H17 ; arbitrary reminder to toggle SW17

OUT SSEG1

WaitForUser:

IN XPOS

OUT SSEG2

IN XIO

AND ResetMask

XOR Mask3

JNEG ForgetReset

JZERO ForgetReset

IN XIO ; contains KEYs and SAFETY

AND StartMask ; mask with 0x10100 : KEY3 and SAFETY

XOR Mask4 ; KEY3 is active low; invert SAFETY to match

JPOS WaitForUser ; one of those is not ready, so try again

Main: ; "Real" program starts here.

OUT RESETODO ; reset odometry in case wheels moved after programming

LOAD One

STORE CurX

STORE CurY

Game:

; CALL FORGETME ; Reset data associated with bot and base station

CALL ImAlive ; Let base station know I exist

CALL REPDUTY ; Let it know that I'm ready for duty

LOAD Zero

DoJob:

OUT RESETODO

CALL REQJOB ; Get Job #N and do it

CALL GOHOME

CALL Wait2

LOAD One

STORE CurX

STORE CurY

LOAD N

SUB One ; Decrement N

STORE N ; Store new N

JPOS DoJob ; Do all jobs until 0

;CALL GOHOME ; After all jobs, go home

CALL CLKOUT ; Clock out of base station

JUMP WaitForUser

;\*\*\*\*\* SUBROUTINES

ForgetReset:

CALL FORGETME

LOAD Eight

STORE N

JUMP WaitForUser

; Subroutine to wait (block) for 1 second

Wait1:

OUT TIMER

Wloop:

IN TIMER

OUT LEDS

ADDI -10

JNEG Wloop

RETURN

Wait2:

OUT TIMER

Wloop4:

IN TIMER

OUT LEDS

ADDI -10

JNEG Wloop4

RETURN

WaitHalf:

OUT TIMER

Wloop3:

IN TIMER

OUT LEDS

ADDI -5

JNEG Wloop3

RETURN

; Subroutine to wait (block) & beep for 3 seconds

BeepWait3:

OUT TIMER

Wloop2:

LOAD Two ; 360\*2 Hz

OUT BEEP

IN TIMER

OUT LEDS

ADDI -25

JNEG Wloop2

LOAD Zero

OUT BEEP

RETURN

; This subroutine will get the battery voltage,

; and stop program execution if it is too low.

; SetupI2C must be executed prior to this.

BattCheck:

CALL GetBattLvl

SUB MinBatt

JNEG DeadBatt

ADD MinBatt ; get original value back

RETURN

; If the battery is too low, we want to make

; sure that the user realizes it...

DeadBatt:

LOAD Four

OUT BEEP ; start beep sound

CALL GetBattLvl ; get the battery level

OUT SSEG1 ; display it everywhere

OUT SSEG2

OUT LCD

LOAD Zero

ADDI -1 ; 0xFFFF

OUT LEDS ; all LEDs on

OUT GLEDS

CALL Wait1 ; 1 second

Load Zero

OUT BEEP ; stop beeping

LOAD Zero

OUT LEDS ; LEDs off

OUT GLEDS

CALL Wait1 ; 1 second

JUMP DeadBatt ; repeat forever

; Subroutine to configure the I2C for reading batt voltage

; Only needs to be done once after each reset.

SetupI2C:

LOAD I2CWCmd ; 0x1190 (write 1B, read 1B, addr 0x90)

OUT I2C\_CMD ; to I2C\_CMD register

LOAD Zero ; 0x0000 (A/D port 0, no increment)

OUT I2C\_DATA ; to I2C\_DATA register

OUT I2C\_RDY ; start the communication

CALL BlockI2C ; wait for it to finish

RETURN

; Subroutine to read the A/D (battery voltage)

; Assumes that SetupI2C has been run

GetBattLvl:

LOAD I2CRCmd ; 0x0190 (write 0B, read 1B, addr 0x90)

OUT I2C\_CMD ; to I2C\_CMD

OUT I2C\_RDY ; start the communication

CALL BlockI2C ; wait for it to finish

IN I2C\_DATA ; get the returned data

RETURN

; Subroutine to block until I2C device is idle

BlockI2C:

IN I2C\_RDY; ; Read busy signal

JPOS BlockI2C ; If not 0, try again

RETURN ; Else return

; Subrotuine to inform the base station of existance

IMALIVE:

LOAD TSENDALV ; Load value to send

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive response from UART

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

;CALL INUART0 ; Wait until data is cleared

RETURN

; Subroutine to inform base station that I'm ready

REPDUTY:

LOAD SNDDUTY ; Load value to send

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive from UART

STORE RDUTY ; Store response

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

;CALL INUART0 ; Wait until data is cleared

RETURN

; Subroutine to clock out

CLKOUT:

LOAD CLKVAL ; Load value to send

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive data from UART

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

RETURN

; Subroutine to request job

REQJOB:

LOAD REQVAL ; Load value to send

ADD N ; Add specific job number

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive X1 from UART

STORE X1 ; Store X1

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive Y1 from UART

STORE Y1 ; Store Y1

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive X2 from UART

STORE X2 ; Store X2

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive Y2 from UART

STORE Y2 ; Store Y2

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

; Move to pickup location

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

OUT RESETODO

LOAD Zero

STORE Temp

LOAD X1

OUT SSEG1

SUB CurX

MULT TwoFeet

STORE Temp

JPOS MOVEXF

JNEG MOVEXR

JZERO PICKX

MOVEXF:

LOAD FMedium

OUT LVELCMD

ADD TurnConst

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JNEG MOVEXF

JUMP PICKX

MOVEXR:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS MOVEXR

JUMP PICKX

PICKX:

LOAD X1 ; Update X value

STORE CurX ; Store new X value

OUT RESETODO

CALL Rotate90 ; Rotate 90 anticlockwise

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

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

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

; Move it along Y

CALL STOP

OUT RESETODO

LOAD Zero

STORE Temp

LOAD Y1

OUT SSEG1

SUB CurY

MULT TwoFeet

STORE Temp

JPOS MOVEYF

JNEG MOVEYR

JZERO PICKY

MOVEYF:

LOAD FMedium

OUT LVELCMD

ADD TurnConst

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JNEG MOVEYF

JUMP PICKY

MOVEYR:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS MOVEYR

JUMP PICKY

PICKY:

CALL STOP ; Stop the bot

LOAD Zero

OUT RESETODO

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 ====================

OUT RESETODO

LOAD Zero

STORE Temp

LOAD Y2

OUT SSEG1

SUB CurY

MULT TwoFeet

STORE Temp

JPOS MOVEYF2

JNEG MOVEYR2

JZERO DROPY

MOVEYF2:

LOAD FMedium

OUT LVELCMD

ADD TurnConst

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JNEG MOVEYF2

JUMP DROPY

MOVEYR2:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS MOVEYR2

JUMP DROPY

DROPY:

LOAD Y2 ; Update Y value

STORE CurY ; Store new Y value

OUT RESETODO

CALL Rotate270 ; Rotate 270 anticlockwise

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

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

CALL STOP

OUT RESETODO

LOAD Zero

STORE Temp

LOAD One

OUT SSEG1

SUB CurX

MULT TwoFeet

STORE Temp

JNEG MOVEXR2

JUMP DROPX

MOVEXR2:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS MOVEXR2

JUMP DROPX

DROPX:

CALL STOP ; Stop the bot

LOAD Zero

OUT RESETODO

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

STOP:

LOAD Zero

OUT LVELCMD

OUT RVELCMD

CALL WaitHalf

RETURN

Rotate90:

LOAD RSlow

OUT LVELCMD

LOAD FSlow

OUT RVELCMD

IN THETA

OUT SSEG1

SUB Deg90

OUT SSEG2

JNEG Rotate90

RETURN

Rotate902:

LOAD RSlow

OUT LVELCMD

LOAD FSlow

OUT RVELCMD

IN THETA

OUT SSEG1

SUB Deg902

OUT SSEG2

JNEG Rotate902

RETURN

Rotate270:

LOAD RSlow

OUT LVELCMD

LOAD FSlow

OUT RVELCMD

IN THETA

OUT SSEG1

SUB Deg270

OUT SSEG2

JNEG Rotate270

RETURN

PICKUP:

LOAD PICKVAL ; Load value to send

ADD N ; Add specific job number

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive response from UART

;OUT SSEG2

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

CALL BeepWait3 ; Wait and beep for 3 seconds

RETURN

DROPOFF:

LOAD DROPVAL ; Load value to send

OUT UART ; Send to UART

CALL INUART0

CALL INUART1 ; Receive response from UART

;OUT SSEG2

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

;CALL INUART0 ; Wait until data is cleared

CALL BeepWait3 ; Wait and beep for 3 seconds

RETURN

GOHOME:

; Get Ready to go back to (1,1)

OUT RESETODO

LOAD Zero

STORE Temp

LOAD One

OUT SSEG1

SUB CurX

MULT TwoFeet

STORE Temp

JNEG HOMEXR

JUMP HOMEX

HOMEXR:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS HOMEXR

JUMP HOMEX

HOMEX:

CALL STOP ; Stop the bot

LOAD One ; Update X value

STORE CurX ; Store new X value

CALL Rotate902 ; Rotate 90 anticlockwise

CALL STOP ; Stop the bot

; Now the bot is at (1,N)

OUT RESETODO

LOAD Zero

STORE Temp

LOAD One

OUT SSEG1

SUB CurY

MULT TwoFeet

STORE Temp

JNEG HOMEYR

JUMP HOMEY

HOMEYR:

LOAD RMedium

OUT LVELCMD

SUB TurnConst2

OUT RVELCMD

IN XPOS

SUB Temp

OUT SSEG2

JPOS HOMEYR

JUMP HOMEY

HOMEY:

CALL STOP ; Stop the bot

LOAD Y1 ; Update Y value

STORE CurY ; Store new Y value

CALL STOP ; Stop the bot

RETURN

INUART1:

IN USTB

SUB One

JNEG INUART1

JPOS INUART1

IN UART

AND FIXIN

RETURN

INUART0:

IN USTB

JNEG INUART0

JPOS INUART0

RETURN

; Used for testing purposes only

FORGETME:

LOAD FRGTVAL ; Load value to send

OUT UART ; Send to UART

CALL INUART1 ; Receive data from UART

CALL INUART0 ; Wait until data is cleared

CALL INUART1 ; Receive checksum from UART

RETURN

ErrorVal: DW &HFF

; Constants for Im Alive

TSENDALV: DW &H0B

TESTN: DW &H0F

TESTALV: DW 0

; Constants for Reporting for Duty

SNDDUTY: DW &H10 ; CHANGE TO &H10 FOR ACTUAL JOBS --- TEST JOBS RANGE FROM 80 to 84

RDUTY: DW 0

CLKVAL: DW &H60

REQVAL: DW &H20

CXPOS: DW 0

CYPOS: DW 0

DYPOS: DW 0

PICKVAL: DW &H30

DROPVAL: DW &H40

FIXIN: DW &HFF

FRGTVAL: DW &H90

; N - Job number (8 to 1)

N: DW 8

; Job Coordinates

X1: DW 1

Y1: DW 1

X2: DW 1

Y2: DW 1

; Current location

CurX: DW 1

CurY: DW 1

; Move by X/Y

MovX: DW 0

MovY: DW 0

; Current Direction and Speed

DirSpeed: DW 0

TurnConst: DW 11

TurnConst2: DW 7

; This is a good place to put variables

Temp: DW 0 ; "Temp" is not a great name, but can be helpful

; Having some constants can be very useful

Zero: DW 0

One: DW 1

Two: DW 2

Three: DW 3

Four: DW 4

Five: DW 5

Six: DW 6

Seven: DW 7

Eight: DW 8

Nine: DW 9

Ten: DW 10

FSlow: DW 480 ; 100 is about the lowest value that will move at all

RSlow: DW -480

FMedium: DW 480

RMedium: DW -480

FFast: DW 500 ; 500 is a fair clip (511 is max)

RFast: DW -500

Pedro: DW 150

Deg90: DW 113

Deg902: DW 80

Deg270: DW 456

; Masks of multiple bits can be constructed by, for example,

; LOAD Mask0; OR Mask2; OR Mask4, etc.

Mask0: DW &B00000001

Mask1: DW &B00000010

Mask2: DW &B00000100

Mask3: DW &B00001000

Mask4: DW &B00010000

Mask5: DW &B00100000

Mask6: DW &B01000000

Mask7: DW &B10000000

StartMask: DW &B10100

ResetMask: DW &B01100

AllSonar: DW &B11111111

OneMeter: DW 476 ; one meter in 2.1mm units

HalfMeter: DW 238 ; half meter in 2.1mm units

TwoFeet: DW 265 ; ~2ft in 2.1mm units

MinBatt: DW 110 ; 11V - minimum safe battery voltage

I2CWCmd: DW &H1190 ; write one byte, read one byte, addr 0x90

I2CRCmd: DW &H0190 ; write nothing, read one byte, addr 0x90

; IO address space map

SWITCHES: EQU &H00 ; slide switches

LEDS: EQU &H01 ; red LEDs

TIMER: EQU &H02 ; timer, usually running at 10 Hz

XIO: EQU &H03 ; pushbuttons and some misc. inputs

SSEG1: EQU &H04 ; seven-segment display (4-digits only)

SSEG2: EQU &H05 ; seven-segment display (4-digits only)

LCD: EQU &H06 ; primitive 4-digit LCD display

GLEDS: EQU &H07 ; Green LEDs (and Red LED16+17)

BEEP: EQU &H0A ; Control the beep

LPOS: EQU &H80 ; left wheel encoder position (read only)

LVEL: EQU &H82 ; current left wheel velocity (read only)

LVELCMD: EQU &H83 ; left wheel velocity command (write only)

RPOS: EQU &H88 ; same values for right wheel...

RVEL: EQU &H8A ; ...

RVELCMD: EQU &H8B ; ...

I2C\_CMD: EQU &H90 ; I2C module's CMD register,

I2C\_DATA: EQU &H91 ; ... DATA register,

I2C\_RDY: EQU &H92 ; ... and BUSY register

UART: EQU &H98 ; The basic UART interface provided

USTB: EQU &H99 ; UART\_OUT\_STB

; 0x98-0x9F are reserved for any additional UART functions you create

SONAR: EQU &HA0 ; base address for more than 16 registers....

DIST0: EQU &HA8 ; the eight sonar distance readings

DIST1: EQU &HA9 ; ...

DIST2: EQU &HAA ; ...

DIST3: EQU &HAB ; ...

DIST4: EQU &HAC ; ...

DIST5: EQU &HAD ; ...

DIST6: EQU &HAE ; ...

DIST7: EQU &HAF ; ...

SONAREN: EQU &HB2 ; register to control which sonars are enabled

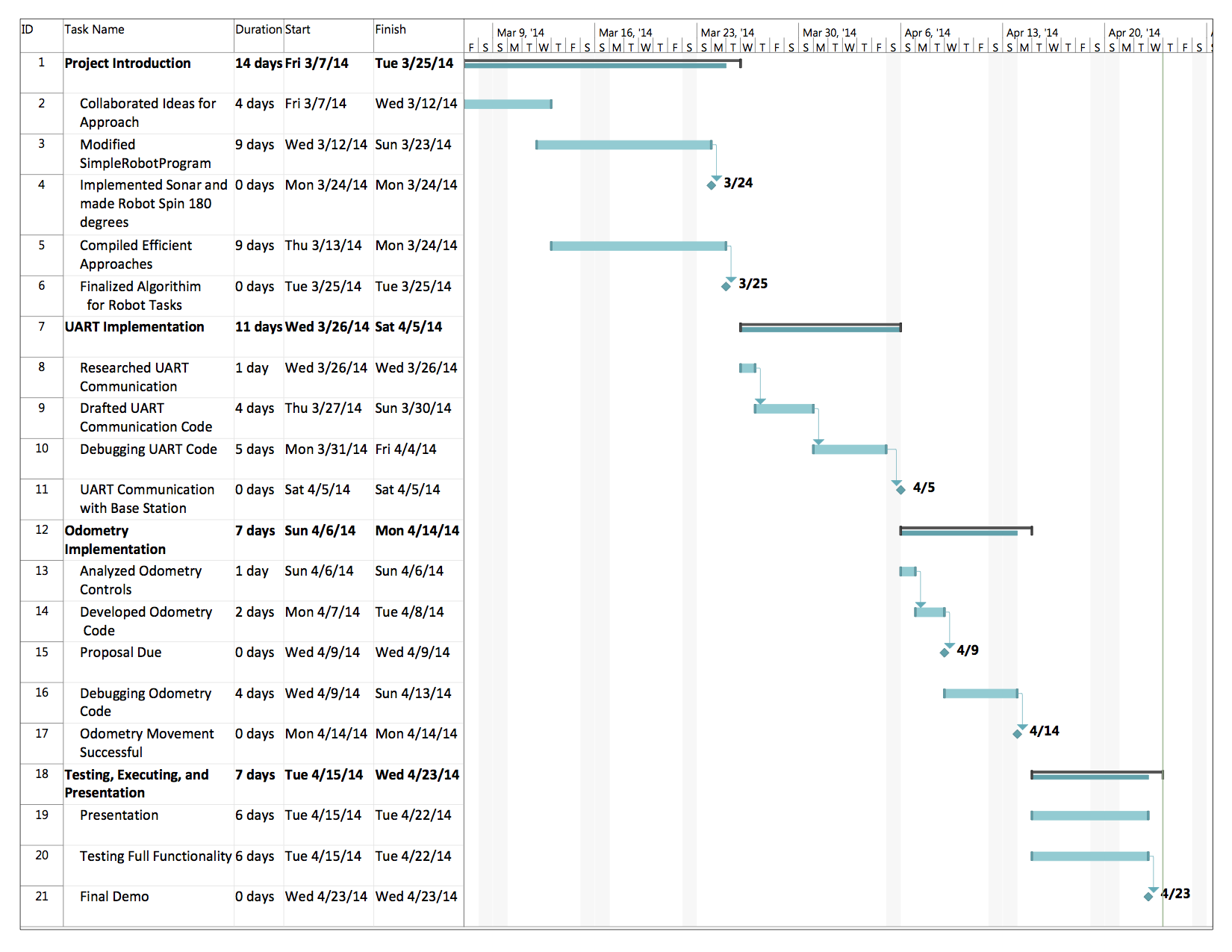
XPOS: EQU &HC0 ; Current X-position (read only)

YPOS: EQU &HC1 ; Y-position

THETA: EQU &HC2 ; Current rotational position of robot (0-701)

RESETODO: EQU &HC3 ; reset odometry to 0

**Appendix B: Final Gantt Chart**

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

**Appendix C: Brainstorming and Logbook Check-off Sheets**