Fifth Stage Prototype of an Intelligent Material Handling and Routing System

Eric Lund

# Abstract\*

In this report, the final prototype of an intelligent material handling and routing system is designed and tested. The objective of the system is to be able to read data from an item and then rout it through a single intersection. While it is doing these tasks, it should be able to detect errors and produce a warning to the user. The system is designed in the C programming language and the prototype is implemented on a TERN board for debugging and testing. The prototype designed is based on a previous developed prototype. The new module has increased functionality and improved performance compared to the previous prototype. Furthermore, the model is built on the Microcos real time operating system.

# Introduction\*

As a part of *Factory Automation, Routing, and Tracking Systems, Ltd* we have been hired to create a material handling system for a new Dell Computer factory. This project builds upon the prototypes previously developed. The prototype still mimics a single node of the system to perform basic tasks, including material routing operations, error checking, and triggering warnings. The prototype was modified to user a clock to control its execution, accept data from peripheral systems, and display data to an external system. Our prototype will run in the Paradigm operating environment and will output results to LEDs, a speaker, and an LCD screen on a TERN board. Additionally, it will display data through a serial port to a computer terminal. The data will be displayed on a user-interface that will be implemented in java. The java interface will also allow the user to send commands to the system.

# Discussion of the lab

**Design Specification**

In this lab we are using the previously developed prototypes to design a new system with updated functionality. The prototype of the system models a single intersection of the greater material handling system. The prototype must still be able to correctly rout an item through the intersection and recognize error conditions. The system will provide output by turning on an LED to represent the direction of the item and display the item’s identification, direction, and time to rout to an LCD screen for a user to read. The system must run on five second intervals. If an item takes longer than five seconds to rout through the intersection than the next item will be sent on the next closest five second interval. Additionally, information will be sent and received through a serial communication port using a java application. This information displayed will vary depending on whether the item routs normally or an error occurs while the system is routing. Commands that govern the state of the system will be received from the java application. The system also will accept external signals that will model sensors detection a jam condition.

When the system recognizes an error it must be able to read user input from the TERN board keyboard. This will indicate that the error has been resolved and the system can move on. Given the increased user interaction through the serial communications, the system will continue to run when it receives a size error or a bad identification. If the jam sensor triggers, the system will wait for the jam to be resolved before accepting the next item.

**Software Implementation**

The software is to be implemented on a real-time operating system. Tasks will be run according to their respective priorities, which are assigned at startup, as opposed to the previous lists or queues. After a task has run it will block its execution for a static amount of time before it re-enters a ready state. When multiple tasks are ready, the task with the highest priority that is ready will run. New tasks can be dynamically added to the operating system using a task create or a task delete call.



Figure 1: High Level Block Diagram

The first step to designing the system was to map out its control flow. The control flow is demonstrated in the following sequence diagram.



Figure 2: Sequence Diagram of the material handling and routing system

The sequence diagram shows the process going from one ready task to the next. Since the task that has the highest priority and is in a ready state will run, it cannot be guaranteed that each task will run through each iteration unless the maximum total time for the tasks to run is less than the amount of time it takes for the first task to go from a blocked state to a ready state. In response to this, tasks that are fast and important were given lower priorities to ensure that they were not starved out of the system while tasks with long executions, for example the serial communications or routing tasks, were given lower priorities so they wouldn’t contribute to starving out other tasks. Each individual task can perform multiple options before returning to the main module. These actions often modify global variables in the program so that the other tasks may use the data generated at each step. These actions are mapped in the following functional decomposition diagram.

Figure 3: Sequence Diagram

The actions on the lines indicate which global variable is changed in the previous module that tells the next module to move forward. Also, although only one of the directional tasks, left, right, or forward, execute at a time; the order in which these three tasks execute is unimportant since they do not relate to each other. What these drawings do not show is the interval of the clock. The system requires that new data is only accepted on five second intervals. The timing of the system is controlled by an internal timer. This timer can run a maximum of around 6.5 milliseconds. Our system causes an interrupt whenever that timer reaches a designated max time, in this case five milliseconds. This interrupt increments a global counter accordingly in order to ensure the tasks only run every five seconds. Five milliseconds were chosen so that the system had sufficient time to run through its list. Shorter times cause the system to spend too much time in interrupts rather than main code.

Figure 4: Data and Control Flow Diagram

Once the control flow of the program is set, the individual modules can be designed. The full code for the system is located in Appendix D. The pseudo-code for the system is located in Appendix C. The referenced activity diagrams for the modules are located in Appendix E. The system prototypes and data structures for each of the modules designed for this system are located in Appendix F.

**Reader Task**

The reader task did not significantly change from the previous prototypes. The first task of the system is the reader task. The goal of the reader task is to obtain the item’s data and set the global variables of the program for future use. The system will only execute the reader task when it is ready to accept new data. In this regard, the reader task will not perform its function if the system is waiting for a delay to be met or if the system is currently in a jam. These cases were decided because the reader task sets the rest of the program in motion. If the program is already using the LCD screen and the reader task is called, there is a good chance the data from the new item will overwrite the display on the LCD screen before the previous data has been fully resolved. If the function is ready to accept new data than the reader task randomly generates a direction, item size, and item identification.

Our design allows for the item size to generate a negative number; however it does not allow an item size of zero. This was implemented because an item size of zero doesn’t throw an error, yet it is not a logical value for size. The item identification is created as a nine character array. The first eight characters hold randomly generated values and the last character holds a value that allows the item identification to be displayed as a string. The first of the eight randomly generated characters is allowed to either a positive number or a negative sign. This gives the system a chance to create an identification error in the routing task. Once these variables have been generated they are set to the global variables of the system. After this, reader task has finished executing and the system moves on to the next task.

**Directional Tasks**

The directional tasks did not significantly change from the previous prototypes. The left, right, and forward tasks are all considered directional tasks. Like the reader task, these only enter their actions if the system has accepted new data. The reasons are the same as mentioned above. If directional task the system is currently executing is the same as the direction in which the item is moving, the task sets a global variable to tell the future tasks that an item is present and it turns on its respective LED on the TERN board. Once complete, the directional task returns and the program move on. Though the system will run through all three directional tasks every iteration, only one of the tasks will actually affect the system.

**Routing Task**

There were only minor changes in the routing task between the previous and current prototype. The object of the routing task is to check for errors from the data acquired by read, and if none are found, display the data to the LCD screen. Additionally, the data task will also reenter appropriate error modules, if the necessary, and will reset the system when the item has finished routing. Finally, the routing task will create the parse task and send the data to it in order to format the data for serial communications.

First, if the system has accepted new data routing task will perform its main function. While performing its main function routing task gets the item size, direction, and identification from the global variables and randomly generates its own time variable. The time variable sets the amount of time, in seconds, that it will take for the item to rout through this intersection. This is set as a global variable for use later in this and other functions. The routing tasks will than look at the item identification and size to check whether they are negative and check if there is a system jam. If item size is negative than the system will enter a subsystem indicating there is a size error. If the item identification is negative the system will enter a subsystem indicating there is an incorrect identification. If the external sensors indicate there was a jam, the system will enter its jam warning. These subsystems will be discussed in detail later in the report. Before the routing task data can be sent to the serial port it must be properly formatted. The routing task creates the parse task in the operating system and sets a variable to tell the parse task which data to format for serial communications. The system will display the item data to the LCD in order of item identification, item direction, and time to rout.

The rout task will then check whether the system is currently in an error. If the system is in an error, the subsystem for each error will execute before rout task returns to the linked list. The identification error and size error will not stop the execution of the system, so, if necessary, the rout task can execute multiple error modules each iteration.

The routing task will reset the system to a ready state. It will only reset the system once the total delay time has surpassed the time it took the current item to rout and that the system has not reset itself in this item’s iteration before. This because resetting the system is a very intensive process. The system only resets once because of the intensive nature of resetting. Otherwise, the system would significantly slow down if the system passed its rout delay time but continues to delay in the scheduler.

**Startup Task**

As the previous scheduling responsibilities are taken over by the operating system, a scheduler task was no longer necessary. In its place, a startup task was created to initialize the system. The startup task initializes timers and creates the statically scheduled tasks for the operating system. It is run only at the beginning of the program and then deletes itself from the system. The first timer is for use by the operating system. The second timer is used to create a pulse wave for the system’s motor. The third timer is for the system’s tasks to keep track of runtime. Each of these timers has an associated interrupt to carry out the aforementioned duties.

**Serial Communications Task**

The serial communications task was updated significantly in this prototype. The serial communications task relays data to a computer terminal through a serial port. In this case a java application was used to read the data. The task scheduling was changed from dynamic to static because the system can both read and write from the serial port.

The serial communications task has two separate functions, to receive data and to transmit data. Serial communications receives data when characters have been inputted into the system from the serial port, i.e. java application. The data from the serial port is formatted as a frame. The serial port gets this frame and calculates the check sum value of the frame by XORing the data in the frame. If the check sum value of the frame does not equal the check sum value that was calculated by the java application, the task discards the data and asks for the data to be retransmitted by sending the application a negative acknowledge (nak) frame. If this process fails 4 times than the system requests the user to re-initialize the serial connection. If the two values are equal the task sends an acknowledge (ack) frame to the application and continues on and decodes the frame. The payload of the frame is extracted and sent to the parse task to be verified.

If data to be sent is generated and has been properly parsed into a frame than the serial communications task transmits the data to the serial port. The task sends the data and waits for either an ack or nak response from the system. If a nak response is received the task retries to send the data up to four times. If a nak response is received four times the system asks the user to re-initialize the serial connection. If an ack response is received, the transmission was successful and transmit aspect is complete.

**Parse Task**

The parse task is new to this stage of the prototype. The parse task has two main functions, to format outgoing data to a frame or to verify incoming data to a legal command. The parse task is dynamically scheduled and is created whenever data needs to be transmitted to the serial port or data has been received and processed in serial communications.

Once serial communications extracts the payload from a received frame the payload is sent to the parse task. The parse task compares the received payload to all valid commands. If it matches a command, the data is sent to the command task. If the data does not match a command the parse task builds a frame with the ‘E’ character as its payload and sends the frame back to serial communications.

Data that has been generated needs to be formatted before it is transmitted. The parse task takes this data from a global buffer to format. If the data is in a pre-arranged format, in this case either as an ‘M’ command, to display the last item’s data, or an ‘A’ command to display warning information, than the parse task gets this data from global variables and builds the payload. Otherwise, if the data to be transmitted does not follow these formats the parse task simply creates the frame using the data in the parse buffer, as is. Once the proper payload has been assembled, the actual frame is built. The frame consists of five parts: a start character, the frame’s length, the payload information, the check sum of the data, and an end character. The check sum of the data is used to verify that the frame that was received is the same as the frame that was sent. It is calculated by xoring all the data from start to the end of the payload. Once the frame has been built the task signals the serial communications task that data is ready to be transmitted and the task deletes itself from the operating system.

**Command Task**

The command task is a new function to the prototype. The objective of the command task is to enact a command that has been received from the java application and verified by the parse task. The command task is statically scheduled. The available commands are: initialization, start, stop, data logging, get warning information, and get last item information. In order to start serial communications, the system must receive an initialization. This command only needs to be called at the beginning of operations and whenever the serial port requests a re-initialization. The start command activates all measurement tasks and all measurement interrupts. Likewise, the stop command deactivates all measurement interrupts and terminates measurement tasks. The stop command cannot stop the entire system, as you still must run serial communications to get a start command, so the system deletes the reader task so no new items can be read. The data logging command enables or disables data logging. If data logging is disabled than data will be generated but it will not be reported or recorded. The information commands ask the parse task to generate an A or M response and transmit this response to the serial port so that the user can read the latest data at any time.

**Stinger Missile Launcher**

The stinger missile launcher is a new task to this prototype. The goal of the stinger missile launcher is to automatically clear jams in the system. The missile launcher will only active if a jam is present and one or more missiles have been loaded. Missiles are loaded through the missile interrupt. The missile launcher can fire multiple missiles. Firing of the missile launcher is reported to the LCD screen, which tells the user how many missiles were fired. Finally, the missile launcher resets the jam and the number of missiles.

**Java Console**

A java console window was implemented in this version of the prototype for user interaction. The java console performs two main functions, to receive data to be logged and displayed to the window and to transmit commands to the system. All data that is transmitted or received is sent as information frames, as described in the serial communications section of this report. The frames are verified using a stop and wait protocol to ensure that data is successfully transferred in both directions.

The first function of the java console is to act as a user-interface into the system. Once the user inputs data the console formats this data into an information frame. The console waits for an ack or nak response from the serial communications task. If a nak response is received the console resends the data up to four times. If an ack response hasn’t been received in this time the console requests that the user re-initializes the serial connection. If an ack response is received the console finishes.

The second function of the java console is to log and display data to the console window. The data is received as an information frame. Once the console receives a frame from the serial task it must verify the data. It accomplishes this in the same method as the serial communications does; the task calculates the sum of the data received and compares this to the sum of the data recorded in the frame. If they aren’t equal the console sends a nak frame to request the data be resent. If they are equal the console sends out an ack frame and continues on to print the data to the screen. The java console can print the received data out in multiple ways, depending on the payload of the data. If the payload is an M or A response the console decodes each section of the response and prints out pre-decided identifiers and then the data contained in the payload. If the payload is an error command response the console prints out a notification to the user. Otherwise, if the data does not match any of the pre-decided formats the console simply prints out the contents of the payload.

**Warning Modules**

Three additional warning modules were added an addendums to the routing task. These are the jam warning task, the size warning task, and the identification warning task. These tasks were not significantly changed in the new prototype. The objective of these three modules is to provide warning when the system has encountered an error. The type of error each is triggered from can be identified by their respective module names. The size and identification error are detected by the systems software. In this case, if either of them is read as negative numbers the system will enter the appropriate warning module. The jam error is triggered by an external interrupt, which will be discussed in further detail in the Interrupt section. In this prototype, the error is resolved when a user presses a key on the TERN board. These three warnings share many similarities. The first time the warning is encountered the status variable is off. Using this, the error can reset the system from the effects of the previous tasks, display the warning to the LCD screen, and set the global variables to recognize that the system has encountered an error.

When the error has been resolved the tasks resets the necessary global variables and the TERN board. As opposed to the previous prototype, two of the errors do not keep the system from accepting new data; the size and identification warnings. However, the jam warning will keep the system from accepting new data.

Each warning has an individual pattern. The pattern is executed by an updating error state global variable. The warning moves onto its next state when the delay time for the given warning pattern has passed. As the identification warning moves through its states it turns a red LED on for one second, off for one second, on for one second, and off for seconds. The size warning is similar to the identification warning. It flashes the yellow LED on for two seconds and off for one second. The jam warning uses a speaker to alert the user when it occurs. In order to drive a speaker we need to send a wave to its input. We chose to send a square wave by changing a status variable between high and low. In order to generate an audible wave, this switch needs to occur rather frequently. Since we are no longer blocking out for significant amounts of time, the frequency of the wave is tied to how long it takes for timer two to interrupt. A five millisecond timer produces a quality, audible wave. The speaker annunciates on and off at one second intervals.

**Interrupt Service Requests**

Multiple interrupts were used to collect data from outside the system. The interrupts used were three for various timers, two to control the speed of the TERN board’s motor, and two to detect and handle jams in the system.

The timer interrupts watch timer 0, timer 1, and timer 2, which are initialized in the startup task. Timer 0 is used exclusively by the operating system and its interrupt does nothing. Timer 1 controls the pulse width signal sent to the motor for the conveyor speed. Timer one is initialized with two max counts. When timer 1 reaches its first max count the motor is turned off. When timer 1 reaches its second max count the motor is turned on. The total of these two max counts is the period of the wave, 200 milliseconds. The speed of the motor is controlled by the amount of time the motor is set to on. Timer 2 controls the global time counter. This counter ensures that actions are only taken on five second intervals and only when the system is ready to accept a new item.

The speed of the motor can be changed from its initial state through the use of two other interrupts. One of these interrupts increases the speed of the motor while the other decreases it. Both these interrupts function by either incrementing or decrementing the first max count of timer 1 by 15%. To set the motor at full speed the first and second max counts must equal. To set the motor at off the first max count must be zero. These interrupts cannot increase or decrease the speed past 100% or 0%, respectively.

The jam error relies on external sensors to detect and handle the jam. In this system, one of the interrupts signals to the system that a jam has been detected. In this interrupt the system reads in two external signals to decipher which direction the jam has occurred at. The jam is cleared through use of the stinger missile launcher. The second interrupt queues up a missile each time it occurs.

# Presentation, discussion, and analysis of the results

In order to obtain results for our design we ran the program in the laboratory on a TD40 TERN board. The measureable results were the output of the TERN board, the output of the java console, and the synchronicity of times. We expected the TERN board outputs to perform multiple functions including that the item direction turned on the appropriate LED, the LCD screen displayed the data for the given amount of time, and that errors were displayed with proper warnings or annunciation. Multiple outputs could be simultaneously displayed to the LEDs if a warning and a routing were simultaneously occurring. If the warning and routing used the same LEDs then the warning should re-assert its pattern over the routing task. The serial terminal should display the proper data to the terminal and should only display the data once, when the data was first submitted from the routing task. The expected system only sent out data on intervals of five seconds. The java console should display either data sent from the routing task or data that was generated by commands. The system must be able to respond to a variety of commands that are sent from the java console.

The final prototype failed to meet expectations. While pieces of the prototype were fully functional, there were certain areas that did not perform their respective duties. The general structure of the system operated well once ported to the real-time system. The program was able to read in a tag, light the correct directional LED, rout the item and print its data, parse the data to the appropriate frame, and print this frame onto the java console window. The system was able to correctly handle identification and size warnings. The program was able run on the expected interval pattern.

The system had difficulties reading in commands from the java console and executing the proper actions, sending and receiving ack or nak control frames. Independently, each of these tasks was able to execute on their own, performing the expected actions. However, when the system was ran as a whole it would either bypass the task or it would crash the system.

The first error to look at is in receiving and executing commands. This task was shown to work when the program was run through in steps, rather than in its entirety. In this case, the serial task would read in the proper data, the parse task would verify the data, and the command task would execute the command given by data. However, it was determined that when the system was run in a normal execution the serial task would receive the data but the data would never be verified in the parse task. One possible cause is that the system was not entering in the parse task before the system re-entered the serial task and the data was being overwritten with a null character. If this is the case than adjusting the priorities or changing how the data is stored in the serial task may keep the system from losing the data.

The next error is the control frames. When the code for control frames is executed on the local and remote side the program will successfully run through one iteration before it stops executing. It is possible that the system is pre-empting the serial task as it waits for a control frame response or that the system is taking too long and it starves out the task. Testing demonstrated that when the system implemented control frames it would run through higher priority, shorter tasks at a much greater frequency than the longer tasks, but the execution would quickly cease without an explanation from the program. Giving the serial task a higher priority did not help the program; it simply stopped executing any tasks in the system much more quickly. A likely reason for this error is that the serial communications takes too long when the local and remote side needs to communicate. A possible fix for this problem would be to break up the execution of the longer tasks into multiple intervals. Rather than sending the data and waiting for a response in one iteration of the task, it could perform a piece of its task at each iteration with the next invocation of the task performing the next piece of the task. This process could continue until the task has finished executing. The task could keep track of what it has done and what it needs to do through the use of status variables. This could also be applied to any task that takes a relatively long amount of time to execute, including the tasks which write to the LCD screen.

A final error in the system was its unusual response to jam warnings. The jam warning successfully behaved as expected, in triggering, execution, and clearing. However, once the jam warning was cleared the system did not correctly reset. Most noticeably, the system would not behave correctly on the next jam. This error was most likely caused by an oversight in how the jam warning reset.

**// TIMER STUFF DO TOMORROW**

In order to measure the efficiency of the program the execution time of each task was empirically measured. This was accomplished by triggering a high voltage output port on the TERN board and measuring the results on an oscilloscope. The port was toggled between high and low in order to produce a wave with a period equal to the execution time of the program. The high and low toggles were then placed into the while(1) loop in the main program. The wave was only generated when the program whose execution time we were measuring was currently being executed. The programs could have different execution times depending on exactly how much of the program was being executed. The execution times for the programs are as follows: Schedule Task: 29µs, Reader Task: 35.5µs, Direction Tasks: 12 or 17µs, Routing Task: 27µs, 31.5µs, or 70µs, Serial Communications Task: 140µs, 170µs, or 195µs. Unfortunately, these execution times cannot be compared to the previous prototypes execution times because the previous execution times were measured incorrectly. However, these times will be useful for determining the efficiency of future improvements on the design.

# Test Plan

The testing of the design can be carried out through the TERN board, java console, and Paradigms debugging tools. In order to meet the original requirements, the design will need to properly route items, identify errors, respond to external commands, and display proper outputs to the LCD and java console. When an item is routed, the LED and LCD screen should be active for as long as the generated time to transverse the route and the computer terminal should display data. Each route should only occur on the nearest five second interval. If the system receives an incorrect item size or identification is must throw a warning and not display the item data to the LCD screen. Data must still be sent to the computer terminal and printed correctly. When a warning is thrown, the output of the TERN board must match the expected output of the error, both in the type of output and the correct timing pattern. When the computer receives a command, it must be able to execute the proper action related to that command.

# Test Specification

The timing of the system must be tested so new data is read only on five second intervals. If a routing delay is under five seconds, the next data must be read at the five second system time. If a routing delay is over five seconds, new data must not be read in at five seconds or as soon as the rout finishes. Instead, it must be read at the next interval of five seconds. If a routing delay is exactly five seconds than new data must be read in immediately.

The system also must be able to handle errors. The system must be able to handle errors occurring one after another, multiple errors, and a rout while an identification error or size has not been resolved.

Finally, the system must be able to respond to data sent from the java console. Data should be sent in all different varieties.

# Test Cases

In order to test the timing of the system, time delays can be forced to have desired values by hard-coding them into the program. The rout delay time needs to be checked at a value less than five, a value equal to five, a value greater than five that is not a multiple of five, and a value greater than five that is a multiple of five. The delay time of the system can be observed by viewing the global variables as the program runs. The global variables must be dynamically updated as the program runs in order to observe results. The system delay time can be compared to the rout delay time in these variables. The program should only reset the system delay time when it is equal to or has surpassed the rout delay time, and only when the value modded by five is equal to zero.

The system’s ability to handle errors can be checked by forcing the item size and/or the item identification to hold a negative value. The system can then be visually observed to check whether it operates with continuous or multiple errors being thrown. Additionally, the global variables can be observed while the system is in an error. This will allow the user to watch the delay time surpass the count that it could reach in non-error iteration.

The command’s can be tested by entering data with different numbers, characters, capitalizations, and string lengths. One or more of these cases should be sent in at a time to make sure the system only responds to proper commands and otherwise returns an error.

# Summary and Conclusion

The final stage of the prototype greatly increases the functionality of the system. Compared to previous prototypes, the new design

In conclusion,

# Appendix A: Requirement Specifications

**System Description**

This specification describes and defines the requirements for an automated material handling system. The material handling system must be able to read an item’s identification, destination, and size from an attached RFID tag. The material handling system can move the item in three different directions at each node and signal warnings or alarms if a complication occurs. The system is also to be fully automated. A user will be able to read the item’s route and time to destination from a computer terminal. The material handling system is to be robust, reliable, low power, and inexpensive.

**Specification of External Environment**

The material handling system is to operate in a factory environment. It will be implemented on a grid of mag-lev tracks.

***System Input and Output Specification***

*System Inputs*

The system will be able to read data in multiple forms, such as integers, strings, and high precision numbers.

*Events –*Events will occur on five second intervals.

*System Outputs*

The system will display the following data on a computer screen

* Route taken
* Time to transverse the intersection

**User Interface**

The user will only be able to turn the system on, off, or reset. The system shall be fully automated and handle its own errors.

# Appendix B: System Design Specifications

**System Description**

This specification describes and defines the basic requirements for an intelligent material handling system. The material handling system must be able to read an item’s identification, destination, and size from an attached RFID tag. The material handling system can move the item in three different directions at each node and signal warnings or alarms if a complication occurs. The system is also to be fully automated. A user will be able to read the item’s route and time to destination from a computer terminal. The material handling system is to be robust, reliable, low power, and inexpensive.

**Specification of External Environment**

The material handling system is to operate in a factory environment. It will be implemented on a grid of mag-lev tracks.

*User*

Turn system on/off

Supply feedback to error warning with keyboard

*System*

Generate item size, identification tag, and direction

Check direction

If direction is left, light green LED

If direction is forward, light yellow LED

If direction is right, light red LED

Check item size

If item size is negative, turn off directional LEDs, issue written warning and flash yellow LED. Enter error loop.

If keyboard is not pressed, keep giving alarm

If keyboard is pressed, clear display and reset warning

If size is not negative, continue normally

Check Identification tag

If first character in identification tag is a negative sign, turn off directional LEDs, issue written warning and flash red LED Enter error loop

If keyboard is not pressed, keep giving alarm

If keyboard is pressed, clear display and reset warning

If identification is not negative, continue normally

Check Jam Status Data

If jam present is true issue written and aural warning and enter loop

If keyboard is not pressed, keep giving alarm

If keyboard is pressed, clear display and reset warning

If jam present is false, continue normally

Generate time to destination value

Display identification tag, direction, and time to destination for the length of time to destination in seconds.

Display Data on Computer Terminal.

Clear TERN board display and reset variables for another loop.

**System Functional Specification**

The material handling system is intended to read data from an RFID tag, move the item in one of three different directions, check for errors in the system, and display the contents of the tag to a computer screen.

The system comprises of seven major blocks as given in the high level block diagram.

***Read Data Subsystem*** The read data function will be implemented by taking the data from the RFID tag and storing the data for identification, direction, and size of the item. The system must be able to read integers, strings, and high precision numbers.

***Directional Subsystem*** The direction functions will be implemented to determine which direction the item is moving, from the data stored by the read function, move the item on the appropriate path, and store in data that an item is currently present at the node.

***Routing Subsystem*** The rout function will be implemented to handle the package while it’s in transit. The function will check the data stored by the read function and trigger a warning/alarm for item jams or incorrect identifications. If the data stored by the read function is accurate, the rout function must display the time for the item to transverse the node and the identification of the package currently in the node to an interface.

***Scheduling Subsystem*** The Schedule system will be implemented to start a timer and a timer interrupt the first time it is activated. These two processes will only activate on start up. Otherwise, the schedule system will be able to add or remove other modules as needed.

***Serial Communications Subsystem*** The serial communications subsystem will be implemented by sending data from global variables to a computer terminal via a serial port. The serial communications subsystem will be a dynamic task, and as such will only be included in the list when necessary.

# Appendix C: Pseudocode

// Lab3.c Pseudo-code

Declare Global Variables

Left, Right, Fwd, itemSize, itemID, itemPresent, schedule delay time, TOTALDELAYTIME

Declare Function Prototypes

Create TCB structure

Create Data Structs

Main void(main)

// Local Variables

Delcare TCBs: Left, Right, Fwd, Routing, Reading, Scheadule

TCB Taskqueue

Declare Data Structs: Left, Right, Fwd, Routing, Reading

Initialize TCBs

TCB.taskDataPtr = TaskData

TCB.taskPtr = Task

Initialize Data structs

\*.\*Ptr = &\*

Initialize Hardware: TD40 Board and LCD

Initialize external interrupts for jam

Initialize Task Queue to TCBs

Create Linked list using insert command

While(1)

Run task at aTCB

Set aTCb to its Next

}

Left Task(data)

If(Delay interval has passed)

DataPtr gets data (Recast)

If(Left is True)

itemPresent gets true

Turn on LED

}

Fwd Task(data)

If(Delay interval has passed)

DataPtr gets data (Recast)

If(Fwd is True)

itemPresent gets true

Turn on LED

}

Right Task(data)

If(Delay interval has passed)

DataPtr gets data (Recast)

If(Right is True)

itemPresent gets true

Turn on LED

}

Reader(data)

If(Delay interval has passed)

ReadData gets data

Get item size from random

Set Global Size

Get direction from random

Set gotten Global direction to True

Create ID array

For (i<8)

Get char from random

Set char to array

Set Global ID from ID

}

Routing(data)

If(delay interval has passed)

Routdata gets Data

If(Item Present)

Get item Size

Get Item ID

Get time from random

If(size is negative)

Call Size Warning

Insert Serial Comm

If( ID is negative)

Call ID warning

Insert Serial Comm

If(jam present)

Call jam warning

Insert Serial Comm

Else

Print ID to LCD

Print Direction to LCD

Print Time to LCD

Send data to Serial Comm

Insert Serial Comm

Delay Time

If(display delay up)

Reset LEDs, Global Variables, LCD

}

Schedule()

If(first time)

Start timer 2 and its interrupt

If(add)

Add Serial node to list

If(remove)

Remove Serial node to list

}

ID Warning(Data)

If(first time through)

Reset Old System

Set System To ID Warning

If(error resolved)

Reset system to ready state

Else

If(warning state and delay time up)

Move to Next State

Turn on/off LED

}

Jam Warning(data)

If(first time through)

Reset Old System

Set System To ID Warning

If(error resolved)

Reset system to ready state

Else

If(warning state and delay time up)

Move to Next State

Turn on/off alarm wave

Create Alarm wave

}

Size Warning(Data)

If(first time through)

Reset Old System

Set System To ID Warning

If(error resolved)

Reset system to ready state

Else

If(warning state and delay time up)

Move to Next State

Turn on/off LED

}

Serial Communications(data) {

If(size warning)

Put(Size,Direction)

If(ID warning)

Put(ID,Direction)

If(Jam warning)

Put (line)

Else

Put(ID,Size,Direction)

Remove from list

}

Timer2\_ISR() {

Increment global delay time counter(s) by 5

If(5000)

Reset counter

}

JamDataSensor\_sX\_ISR() {

Set sX to 1

Update line string

Set jam present to TRUE

}

# Appendix D: Source Code

# Appendix E: Activity Diagrams



Figure 5: Directional Activity Diagram



Figure 6: Reading Activity Diagram

Figure 7: Routing Activity Diagram

Figure 8: Scheduling Activity Diagram



Figure 9: Warning Activity Diagrams

Figure 10: Serial Communications Task Activity Diagram

# Appendix F: Data Sturcts and ProtoTypes

# Appendix H: Other

Sources:

AMD, “AM186 and AM188 Family Instruction Set Manual”, <http://ee.washington.edu/class/472/peckol/reference/AM188ES_instruction_set.pdf>, February, 1997

Jered Aasheim, “rand1.c”, [http://ee.washington.edu/class/472/peckol/ assignments/lab2/](http://ee.washington.edu/class/472/peckol/%20assignments/lab2/), September 8, 2000

Estimate Hours

Design

Test/Debug

Documentation

X\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_