4COM1042 [Computing Platforms]

Co-design Group Project D

A Grid World Robot Simulator (gRobot)

Notes

Mick Walters, 2016

Summary

The Grid Robot Simulator simulates a simple grid square based world, which enables the creation and control of one or more robots within the simple grid world simulator. Before trying to run or use either program file, they must first be extracted from the compressed .ZIP Archive! To do this, right click on the “RobotGridWorldSimulator” Zip Archive folder, as downloaded from Studynet, and select “Extract”. You can then follow the prompts to select and extract all files to your working folder (directory). As you may know already, there is a Python client library (grobot.py) which can be used to interface a python program to control the gRobot. There is now also a Logisim circuit (grobotIF.jar) which provides a useful subset of the Python client functionality, which can be used to interface Logisim circuits to the simulator and control a gRobot.

Preliminaries

The Grid World Robot Simulator can be obtained from the Platforms page on Studynet. All files are contained in a .zip archive. The most current version is V2, which differs from the version you may have encountered previously in another module, as several bugs have now been fixed. The zip archive should contain three files; RobotGridWorld.py, grobot.py and the Logisim library circuit, grobotIF.jar. Within the archive should also be a subdirectory containing some sample maps. Amongst these is Maze.map which contains the maze to be solved for this project.

Grid Robot Simulator (RobotGridWorld.pyw)

To run the simulator, once extracted from the zip archive, simply double click on the program icon from your favourite file manager (Linux) or File Explorer (Windows or Mac). If you want to run the simulator on your own laptop or pc, Python 3.5 should be installed. Python can be obtained as a free download for your OS (Windows, Linux or Mac) from www.python.org.

When run, the simulator program will initially display a blank map, which is a 30x30 grid of squares, with x and y co-ordinates marked along the lower and left sides respectively. There are several buttons available categorised as:

Map Editing Controls:

"New Map" - Clears the current Map ready for editng a new one

"Load Map" - Loads a previously saved map

"Save Map" - Saves the current (edited) map, for later loading

Clicking anywhere on the map grid area will toggle the building and removal of walls on the map. When the pattern of walls is as desired, the map can be saved (click "Save Map" to a file (extension <name>.map). A previously “Saved” map can be recalled by selecting "Load Map". There should be several example maps extracted to your working folder (directory) along with the RobotGridWorld.pyw program file

Simulation Controls:

"Toggle Trails" - Toggles the display of appropriately coloured trails for each robot

"Speed" - A slider control to adjust the speed of the simulated robot(s) movements.

Grid Robot Control Interface

The gRobot can perform five actions:

Home (or Init): This initialises a new or existing robot (red only) and displays it at grid square x = 1, y = 1.

Move Forward: Moves the robot one square forward in the direction the robot is currently facing

Turn Left: The robot will turn 90 degrees left

Turn Right: The robot will turn 90 degrees right

Look: The robot will return the current status of adjacent grid squares, whether they are blocked (Wall etc) or clear (ok to move into). Only the status of the five grid squares relative to the front and sides of the robot are returned: Left, Left Front, Front, Right Front and Right.

The grobotIF.jar Chip

The Logisim interface to the Robot Grid World Simulator is a .jar library circuit, which can be loaded into a current circuit design by selecting Project/Load Library/JAR Library, and selecting the grobotIF.jar file to load. This will make available a Grobot Interface section in the Circuits Menu, which has a single item, grobotIF. This component is a chip with one 8 bit Input port (left side) and one 8 bit Output port (right). The grobotIF chip input port bit functions are defined as follows:

Bits 0 to 4: grobot command or function to perform (see above), defined as:

> 0xb1000 = Home/Init. New or existing Robot is initialised and displayed facing North at grid square x=1, y=1. Currently there is no way to select the colour, or change the Home position of the robot from the Logisim interface.
>
> 0xb0100 = Fwd. Robot to move forward one grid square (see above). If the way is blocked, then the robot will crash and be disabled and will ignore any further commands until a Home/Init command is recived to reset the robot to the Home position.
>
> 0xb0010 = Left. Robot will turn 90 degrees left.
>
> 0xb0001 = Right. Robot will turn 90 degrees right.
>
> 0xb0000 = Look. Robot will return the status of the front five grid squares on the Least Significant bits of the grobotIF output port (see below). All the previous commands automatically return the status of the adjacent front grid squares, so this function is only useful if the sensors need to be interrogated without the robot making a move.

Bits 4, 5, and 6: Unused at present. Normally they should be connected to Ground or Logic zero for correct chip operation*.

---

* This is a shortcoming of my expertise in Java, and also due to the fact that the Logisim Java interface is not well documented! These input bits are reserved for the future addition of functionality to the grobotIF chip.

Bit 7: act = Action Trigger. A high signal (logic 1, rising edge) will trigger the grobotIF chip to send the function/command asserted on bits 0 to 4 to the grobot simulator. Note, if the rdy bit on the grobotIF output port is low (zero), any action triggered, apart form Home(/Init) will be ignored (see below)

The grobotIF output port bit functions are defined as:

Bit 7: rdy = Ready. When this signal is asserted high, the chip is ready to receive a command and action trigger on the input port (see above). While the robot is moving or processing the command, this line will go low until the move is completed.

Bit 6: err = Fatal Error. If a grobot command is unsuccessful due to a serious error (i.e. the simulator is not running, or cannot create a TCIP port etc.), this bit will assert high.

Bit 5: coll = Collision. Will return a high signal to indicate that the robot has collided with a wall or other obstacle in the simulated grid world. Note, this will only occur with a forward move command.

Bits 0 to 4: Sensor status. Will indicate the status of the five forward grid squares relative to the robot's current heading (N, S, E, or W); Bit 4 = Left, Bit 3 = Front Left, Bit 2 = Front, Bit 1= Front Right, Bit 0 = Right. A 1 indicates the grid square is blocked or occupied, a 0 that the grid square is clear of obstacles.

As the simulated robot operates asynchronously, in order to ensure correct operation of the grobotIF chip the following sequence of operation should occur:

1) Wait for the Rdy bit (7) to be high on the grobotIF output

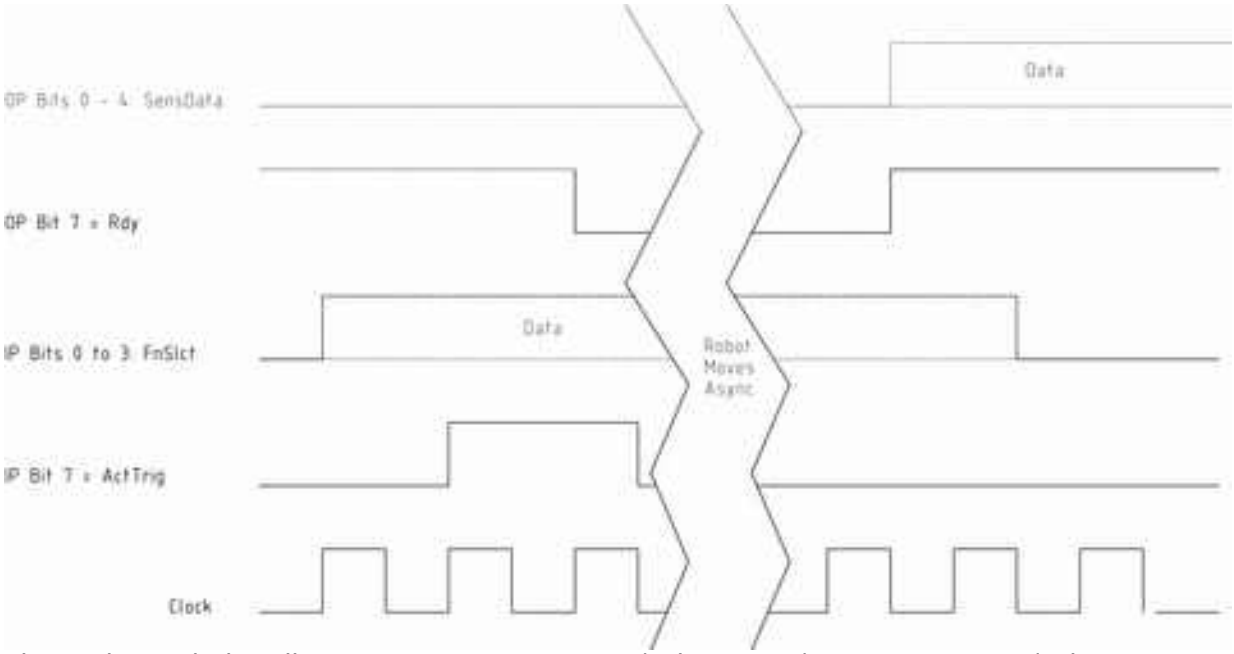
2) Assert the desired command bit pattern on bits 0 to 3 of the grobotIF input port.

3) Assert the ActTrig bit (7) high on the grobotIF input port.

4) The Rdy bit (7) on the grobotIF output port will go low to indicate that the robot is moving.

5) The act bit (7) IP should remain high until the Rdy bit(7) OP goes low.

6) When the Rdy bit (7) OP goes high after the robot has moved, the return status of the grobotIF can then be safely read on bits 0 to 6 of the grobotIF OP.



The grobotIF timing diagram. OP = output port (CdM8 Input). IP = Input Port (CdM8 Output port)

Logisim Hardware Design

The Logisim hardware circuit design is mainly straight forward, and consists of circuits you have met before including Button inputs, and two numeric displays and various LED indicators. The grobotIF chip input signals can be driven entirely by software, though it is convenient to “hard wire” the Action Trigger (ActTrig) via a one or two clock tick delay to an RS Latch in order to reduce software complexity and speed of response. The latch is reset by the Rdy signal after the robot has completed its action. The CdM8 program can simply monitor the RDY (OP Bit 7) signal to indicate that the robot action has completed.

The addresses and function of the input and output registers to be wired to the CdM8 IO bus are as follows:

CdM8 I/O Address	Circuit	Input/Output	Notes
0xf0	GrobotIF Driver	Input and Output	See above for details.
0xf1	Heading Indicator LEDs (N, S, E, W)	Output only	Only least sig. 4 bits are used.
0xf2	X position numeric display	Output only	) }Can also include hex } to dec conversion?
0xf3	Y position numeric display	Output only	)
0xf4	User Buttons: Run, Home, Left, Right, Forward.	Input only	Hint: Priority encoder can simplify circuit.
0x05	Handy debug port (optional!)	Output only	Latch output with manual button reset

Other indicator LEDs can be wired/driven directly from the grobotIF outputs. These are:

Fatal Error LED

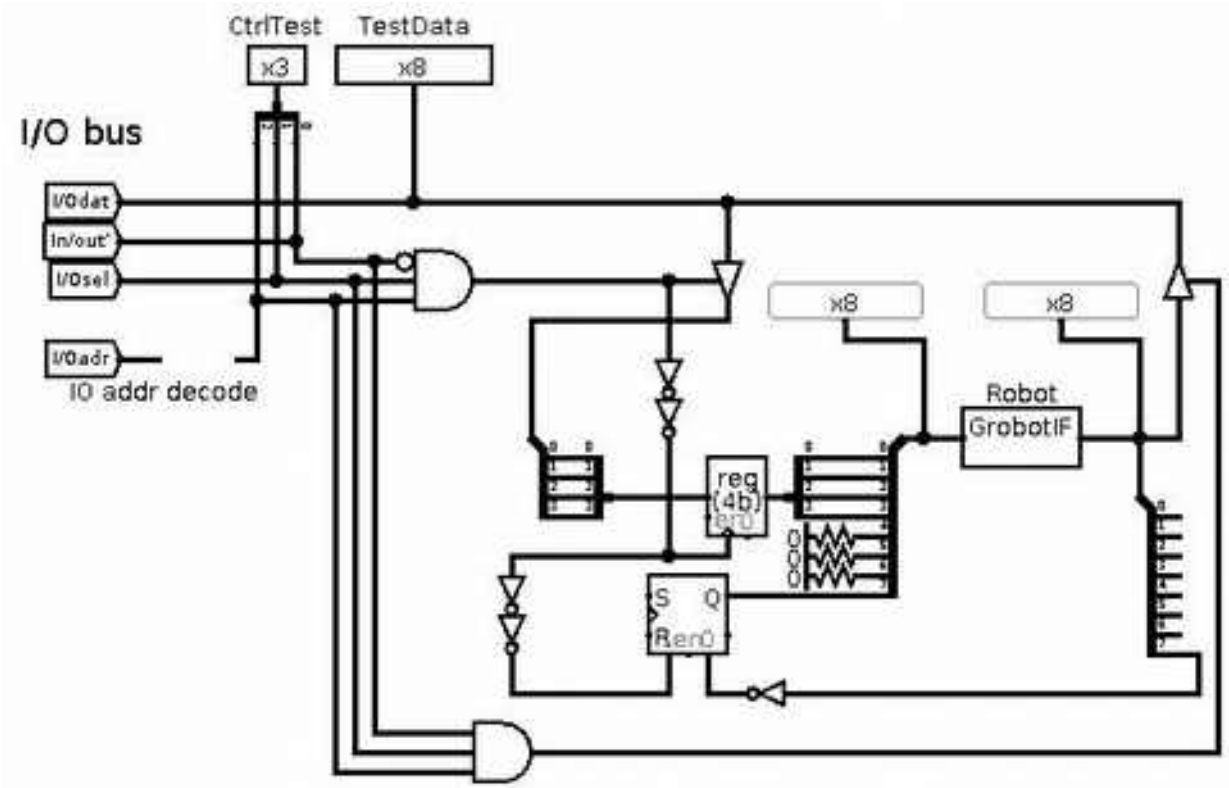
Collision Indicator LED

Sensor Data LEDS: Left, Left Front, Front, Right Front, and Right sensor status (1 = blocked, 0 = clear)

Simulation problems:

The grobotIF input lines are floating by default, so unconnected inputs (Bits 4 to 7) should be connected to weak pull down resistors to ensure correct operation.

Example I/O Driver Circuit for the grobotIF chip, with test inputs for IO control and Address select signals.



Software

Software development represents the main challenge of the gRobot project. The eventual main aim of this project is to produce a program that will control a gRobot to enter the provided Maze (Maze.map) at grid square x=1, y=1, and find the exit to the maze (at grid square x=0, y=15). Although the algorithm to accomplish this is the simple “Left Hand Rule” and can be encoded in less than 25 lines of CdM8 assembly language. The program needs to monitor the sensor and error data from the grobotIF chip so that the robot does not collide with a wall or obstacle at any time. The robot should also be able to recognise when it has reached the maze exit and stop automatically. In order for the robot to recognise when it has found the maze exit, the control program needs to keep track of the X and Y positions at every move and compare them to the (preset X and Y maze exit co-ordinates). To update the X and Y co-ords correctly, your your program also needs to track the (N, S, E or W) direction of each move, so it should keep track of the heading of the robot at every move. Also, updating of the X, Y and heading variables should not occur if there is an error which has prevented a move from being made. Therefore, your program should also monitor the grobotIF error signals and respond appropriately if an error does occur.

Suggested Design Progression

Software and hardware (Logisim circuit) development can be carried on in parallel to a large extent, with the software developers testing each hardware component as they are completed by the hardware designers. The recommended progression is as follows:

1. The hardware designers should concentrate on implementing a working grobotIF driver circuit first and connecting this to the CdM8 IO bus. As soon as this circuit is ready, the software developers can test it out with short test program to make sure that the circuit functions, the grobot can be reliably controlled, and also to gain experience at writing code that can communicate both send instructions to the grobotIF chip, and also recieve sensor data back successfully.

2. Once both software and hardware developers are siatified that the grobotIF chip is working satisfactorily, the hardware designers should implement the five user buttons, so the software developers can then develop a simple manual remote control program. Note, the program should also handle errors.

3. At this stage, the hardware designers can go ahead and finish the other output devices, including the X and Y numerical displays, Heading Indicator LEDs and also the sensor data and Error and Collision Indicator LEDs.

4. In parallel, the software developers can now develop the rest of the maze solving program, using variables to keep track of the robots X, Y positions and current Heading, but with no X, Y and Heading displays. Much of this work can be done on the CdM8 emulator, using temporary variable locations (as the emulator does not have split instruction and data memory) and checking that output register values at memory locations 0xf0 to 0xf4, are set correctly, with regular iterative testing of the memory .img on the partly completed Logisim circuits and robot simulator.

5. Once these software and hardware parts are completed, the software developers can then add the necessary code to display the X and Y positions, and update the Heading LED indicators. The hardware developers should finally incorporate additional circuitry (hex to dec) to show the X and Y positions in user friendly decimal number format on the two numeric displays.

Appendix: