cs281: Introduction to Computer Systems

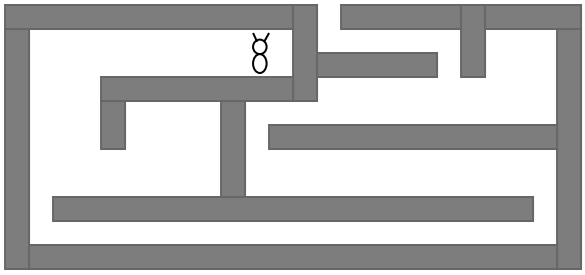
Lab07 - Sequential Circuits II: Ant Brain

1 Problem Statement

Obtain the file `ant.tar` from the class webpage. After you untar this file in an empty directory, you will find:

- `Lab07AntBrain.pdf`: This lab handout description.
- `MazeDriver.circ`: A main testing file. See later in this lab handout.
- `ant.circ`: Space for your project implementation. See later in this lab handout.
- `Maze1.in`: The image for the RAM in the `MazeDriver.circ` circuit. See later in this lab handout.

In this lab, you are tasked with building the controller for a robotic ant. The goal of the control program (aka, the ant *brain*) is for the ant to traverse out of a maze by sensing its surrounding with its antennae and controlling its motors to affect motion to move about and find its way out of the maze. A clock will dictate the steps of the control program and a sequential circuit, and its accompanying state machine, will model the desired behavior of the robotic ant. The problem is illustrated in the figure below.



To make the problem more tractable, we consider a simplified input/output interface for the ant brain.

Inputs: The ant has two antennae, which provide two boolean inputs to the ant.
L is 0 if the ant's left antenna **is not** touching a wall and is 1 if the left antenna **is** touching a wall.
R is 0 if the ant's right antenna **is not** touching a wall and is 1 if the right antenna **is** touching a wall.
Outputs: The ant brain outputs one of three discrete action choices.

action	**description**
forward	moves the ant forward one unit in the maze. If there is a wall immediately in front of the ant, the forward operation has no effect.
left	the ant makes a 90 degree turn to the left (counter clockwise), rotating in place.
right	the ant makes a 90 degree turn to the right (clockwise), rotating in place.

Goal: A desired state for the ant (at the exit).

Strategy: A general search process that helps the ant find the maze exit. This will be a function mapping inputs to outputs. You will also need internal states for the ant which will be part of your strategy/function.

2 Ant Behavior and Maze Characteristics

We consider mazes built from a 2D array of squares. Walls separate squares and define boundaries through which the ant cannot pass. Figure 1 shows a discrete maze that consists of 4 rows by 4 columns of squares along with wall separators. The ant is located in the upper left corner of the maze. A forward motion will always move the ant to the next full square (if there is not an intervening wall). A left or right turn will always turn the ant 90 degrees, thus it is always oriented North, East, South or West.

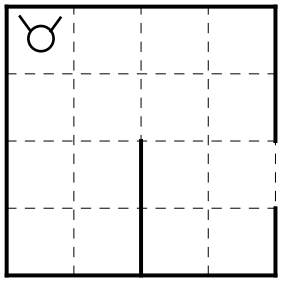


Figure 1: Example Maze

The ant's antennae will detect walls in the following way:

- If the ant is facing a wall directly in front of it, it will always detect this wall with both the left and right antennae. ($L = 1$ and $R = 1$).
- If the ant has a wall immediately to its left, it will detect this wall with the left antenna. ($L = 1$).
- If the ant has a wall immediately to its right, it will detect this wall with the right antenna. ($R = 1$).
- All other possible wall combinations will not engage the antennae detectors.
- The maze will never have a "narrow corridor" that causes the ant to detect both left and right walls with nothing in front. You can assume the only configuration that will activate both antennae is when a wall is directly in front of the ant.

Your maze may have wide open areas containing several open squares without walls. You can assume there always exist a single-spaced exit somewhere along the outer wall. You can also assume that all maze wall segments are connected; that is, there are no "islands" of inner walls in your maze. Thus you can implement a "wall following" strategy if you choose without concern of getting the ant in an infinite loop.

3 Your Design

As you did with the sequence detector, you must design a finite state machine that governs the operation of the ant brain. You must design and defend your own state machine. Think carefully about the inputs

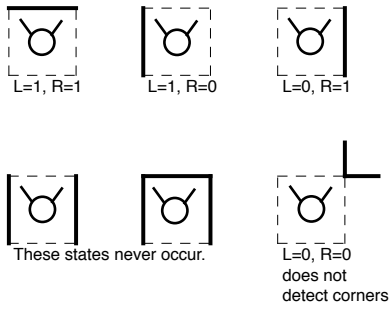


Figure 2: Wall Detection Examples

and the possible maze configurations. Plan your states carefully to capture necessary ant experience that will help guide it towards the exit. Here are some requirements and considerations.

1. Design a Moore-style finite state machine that describes the operation of this circuit.
2. Give a clear and concise description of the operation and design of your ant brain. Use coherent English prose to argue correct operation of the ant given your FSM.
3. Identify how many 1-bit memory elements are required to represent your set of states. Determine a mapping between the logical states in your FSM and an encoding of the set of 1-bit memory elements.
4. Draw a truth table that describes the next state function of the finite state machine.
5. Draw a truth table that describes the output function of the finite state machine.
6. Where appropriate, use K-maps to simplify the boolean expressions for each result of your next-state and output truth tables.
7. Design a sequential circuit from your boolean expressions. Use simple memory devices such as JK flip-flops, D-latches or T flip-flops as your memory elements.

4 Logisim

Open the `ant.circ` file. This is the design space in which you will build your ant brain circuit. It is important that you do **not** change the number and or position of the inputs and outputs. Your circuit must "fit" into the larger testing file and you will have difficulty if you change the inputs or outputs, either by position or by number.

This file contains the four inputs and one output for your circuit.

- **Left** This is a single-bit boolean input that is 1/0 indicating a detection on the ant's left antenna.
- **Right** This is a single-bit boolean input that is 1/0 indicating a detection on the ant's right antenna.
- **clock** This is the external clock from the main circuit. Use it to clock your memory devices (flip flops or latches).
- **reset** This is the external reset line from the main circuit. When the reset input is 1, your circuit should immediately reset its state so that it moves to the start state in your state transition diagram. This reset is "asynchronous" meaning that you should not wait for a clock edge; your circuit should reset immediately when the reset line goes high.

- **Action** This is the output from your circuit. It is a two-bit binary value:

Output	Action Description
`00`	move forward
`01`	rotate left in place
`10`	rotate right in place
`11`	not allowed

Your circuit should never output `11` as the action since this is not implemented.

This circuit initially contains a "dummy constant" providing a constant output of `0x01` (a left turn). You will delete this constant and replace it with your design logic. Along with your lab report, you should submit your `ant.circ` file. Be sure to keep the name of this file the same.

5 Testing

We first describe our scheme for encoding a maze. Figure 3 shows the numbering scheme. The boxes in this maze are numbered (in hex) starting in the upper left corner and moving left to right, then top to bottom.

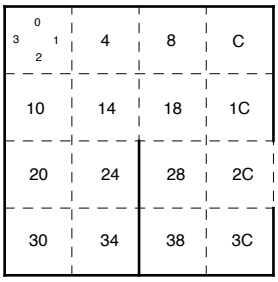


Figure 3: Maze Encoding Scheme

The ant can be located in any one of these 16 boxes. Within each box, the ant has four possible orientations (North, East, South, and West). Thus the ant's state is specified by a state number that corresponds to the ant's location and orientation. Note: this "state" of this system is not to be confused with your own state design within your own circuit; they interact, but they are not the same set of states. For example, state `0` corresponds to a location of the ant in the upper left box, facing North. State `25` has the ant in the box numbered 24, but now facing East (into the wall). State `2D` is the goal state of this maze; it has the ant at the exit, facing East towards the exit.

A very large, complex state transition diagram defines the action of the ant. The action choice (forward, left, right) determines the next state of the ant. For example, if the ant is in State `2A` and receives action 0 (forward), it will move from box 28 facing down, into box 38 still facing down. This will move the ant to State `3A` (box 38 facing down).

Each state has a corresponding "observation" that defines the ant's antenna. In State `25` the ant is in box 24 facing East. This will activate both antennae as it is facing a wall ($L = 1$, $R = 1$).

Now we turn to the logisim circuit which implements the ant system. Open the `MazeDriver.circ` file in logisim. This is the main testing file that allows us to simulate the ant's movement using your controller.

We give a brief description of the main parts of this circuit.

- There are two RAM memory devices in the middle of the circuit. The one on the left is the current state of the ant. (Again this is the state of the external circuit, not the state of your ant controller brain). It contains one of these hex numbers (`00` to `3F` for this maze) to specify a state. The RAM receives the next state input on the left (this will come from the output of your circuit) and sends the current state to the right as an output into the second RAM.
- The second RAM (on the right) implements the state transition table for the ant system. The current state arrives as input and "selects" a line of memory from the circuit. This line of memory contains 32 bits:

bits	**description**
Bits 24-31	These encode the two observations (left and right) of the current state.
Bits 16-23	This is the next state for action 2 (right turn).
Bits 8-15	This is the next state for action 1 (left turn).
Bits 0-7	This is the next state for action 0 (forward).

- The "State Decoder" separates these 32 bits into these four parts. The two observations are fed into your circuit as inputs for the antennae. The three next states are fed into a multiplexor. The mux is selected by your circuit's action choice; the output of the mux is fed into the next state input for the state RAM.
- The "ant" block is from your circuit. This is your logic design as described above. Notice now that it is essential you do not change the number or position of inputs because it must "fit" exactly into this circuit.
- A reset line is on the left. Lift this high to reset the circuit back to its starting state. Set the reset low to run the circuit.
- A clock is used to clock the circuit. The clock latches the next state into the current state. It also is an input into your circuit so that you can maintain and manage your own state information for the ant brain.
- A counter keeps track of how many steps your controller takes to reach the goal.
- A comparator and red LED at the top let you know when you've reached the goal state. The clock is disabled once the goal is achieved.

To operate this circuit you will need to perform two critical steps.

1. You will need to load the state transition table into the right-most RAM. Right-click on this RAM and select "load image". Select the file `Maze1.in`. You should now see the "data" loaded into the RAM. If you are so inclined, you can open the Maze1.in file in a text editor to see how all the information encoded.
2. You will need to load your library. Copy your `ant.circ` file into the same directory as the `MazeDriver.circ` file. Then, over on the menu listing on the left, right-click on the "ant" folder. Select "reload library". This will cause the main driver circuit to load the "guts" of your own ant brain design.

Now you are ready to test your design. Do a reset and then use the clock to move from state to state. You can follow the main information on the circuit to "see" where your ant is moving about.

6 Submission

There are two main components to this lab that must be submitted.

First you should submit your `ant.circ` file. Be sure it conforms and works with the `MazeDriver.circ`. Do all the testing first. You will lose a lot of credit if your circuit is not compatible. There is no reason your ant should fail to "solve" this first example maze.

Second submit a full lab report. Refer to the previous section on the required components of your lab report. Focus on presenting a full solution to the problem. For *this lab report* you do not have to fully describe the problem. It is complex so you can assume your reader has access to this document which describes the problem. But you should fully describe your design for the ant brain.

Both submissions are due via email by5pm on Friday, October 31.

Some design considerations:

- The most important criteria is to build a circuit that successfully navigates the ant to the exit.
- The second most important criteria an is exceptionally well-written lab report. It does no good to design a great circuit if you cannot communicate that design effectively. You are given two weeks for this lab in part so that you have adequate time to produce a high quality report. Make sure your report is free of errors (logical errors in mis-communicating the design as well as grammatical and typographical errors). As is the case with most professional projects, the lab report might take about 50% of the overall project time.
- Do not underestimate the importance of a clean, simple, efficient design for your ant brain controller. Your first design may not be the best solution. You may have to balance simplicity of design with efficiency of operation.
- Consider other possible mazes besides the one you are given in `Maze1.in`. You might even design your own mazes and build another memory image.
- Make your ant brain circuit neat and clean in your `ant.circ` file.
- Try to minimize the number of steps to goal. Keep in mind, your ant should be efficient in all mazes, not just the one as the given example. Also try to keep your ant from running into walls.