Rojobot31 Functional Specification

Revision 3.0

Table of Contents

Introduction

The Rojobot (**<u>Ro</u>**y's and **<u>Jo</u>**hn's **<u>Bot</u>)** models a simple robot moving through a simple environment. The robot is based on the UP1-Bot described in Hamblen/Furman's *Rapid Prototyping of Digital Systems – A Tutorial Approach*, Kluwer Academic Publishers, 2001. As described in Hamblen/Furman, the robot is a platform with two wheels, each driven by an independent motor. A Teflon skid serves to stabilize the platform.

The Rojobot (also called the "BOT" when we're talking about it) gains information about its environment through two emulated sensors:

- Proximity sensors – Rojobot31 has an emulated infrared proximity detector that is activated by two off-angle LED's. The value of the sensor is returned in two bits (ProxL and ProxR) in the Sensors register. ProxL will be set to '1' if the BOT senses an obstacle to the left front of it. ProxR will be set to '1' if the BOT senses an obstacle to the right front of it. Both ProxL and ProxR will be set to '1' if they sense an obstacle directly in front of the BOT.
- Line tracker sensors – Rojobot31 has an emulated black line detection sensor that emulated as three LED/phototransistors pairs (left, center, and right) that indicate the presence or absence of a black line below the front of the Rojobot. The value of the sensor is returned in 3 bits (BlkLL, BlkLC, and BlkLR). Each of the sensors will be set to '0' if that sensor detects a black line below it and '1' if it doesn't. While it may seem strange to have an active sensor return a '0', consider the implementation. The sensor works by detecting a reflection from the floor so the sensor output would be active ('1') when it detects a reflection. A sensor that is placed on top of a black line would cause no reflection and, thus, the sensor returns '0'.

It is important to note the relationship between these two sensor arrays. The proximity sensors are looking ahead of the Rojobot and are by the Rojobot to avoid obstructions. The black line tracker sensors are looking directly under the front of the Rojobot and can be used to guide the Rojobot along a black line.

The Rojobot lives and moves in a 128 x 128 emulated world. The "map" of the world is read from a text file. Each location on the map can have one of the following characteristics:

- Open space – provides no obstacle for our intrepid little Rojobot
- Black line – indicates that the front of the Rojobot is over a black line
- Obstacle – indicates that one or more locations in front of the Rojobot are obstructed (e.g., a wall).
- *RESERVED* –We left this one available for you to create a world of your own. Just think, you could use this type to indicate a land mine or a target for your simulated photon death ray, or however you want to use it.

As a user of Rojobot31 you will interface your hardware to it through the byte-oriented register and input interface described in the next section. You can treat the Rojobot as a "Black Box" and not delve into its implementation details unless you are interested. In fact, Rojobot 3.1 is packaged as a Xilinx Vivado IP block which can be added to your system, and used without understanding the details of the implementation. This would be typical of IP that, for example, you may have procured from a 3rd party for your project.

From the application's perspective the Rojobot should be thought of as a slave peripheral. As is common in embedded systems, the application CPU communicates to the Rojobot through the I/O registers described later in this specification. The "world" that the Rojobot operates in is modeled in a separate dual-port block RAM (called the *worldmap*) which connects to both the Picoblaze emulating the Rojobot and a video controller that can be used to display the map.

Rojobot31 Movement Algorithm

The Rojobot emulation maintains two distance counters (one per wheel) that store the amount of emulated "distance" a wheel has moved since the `MotCtl_in` register was last sampled. In Rojobot31 the `MotCtl_in` register is sampled roughly every 75 msec (50 msec delay plus processing time in the main loop). Practically speaking, the wheels on a physical robot would not move very far in less than a tenth of a second unless they were turning very quickly. To account for this, the emulator accumulates changes in the distance counters until they pass a "wheel move threshold."

Each time the `MotCtl_in` register is sampled (once for each pass through the main loop in the emulation) the emulation code adds the value of the motor speed bits in the Bot Configuration register to the distance counter. When the "wheel move threshold" is exceeded and both motors are turning in the same direction (either forward or reverse) the Rojobot moves to a new location. When the Rojobot motors are turning in opposite directions or when one wheel is turning and the other is stopped the Rojobot will turn. A slow turn (one wheel turning the other stopped) changes the Rojobot's orientation by 45º. A fast turn (motors turning in opposite directions) changes the Rojobot's orientation by 90º.

Application CPU Register Interface

Rojobot31.v outputs six 8-bit registers that contain the state of the Rojobot. The Rojobot is controlled by a single 8-bit input that is used to control the two motors that drive the wheels. The Rojobot can be configured with a single 8-bit register that sets the wheel threshold and enables or disables variable speed control. All of the registers and the control and configurations inputs can be connected to your application CPU through a simple port-based interface and all of the registers are set to their initial values when Rojobot emulation is first initialized.

Rojobot 3.1 implements an *upd_sysregs* signal which toggles every time the state registers are updated. *upd_sysregs* can be used to drive an interrupt on the Application CPU or it can be polled in the main loop of the application.

The BotSim register interface is summarized in the following table:

Direction*	**Bits**	**Name**	**Description**
Input	8	`BotConfig_in`	Rojobot Configuration
Input	8	`MotCtl_in`	Motor Control
Output	8	`LocX_reg`	Rojobot location X (column) coordinate
Output	8	`LocY_reg`	Rojobot location Y (row) coordinate
Output	8	`BotInfo_reg`	Rojobot orientation and movement (action)
Output	8	`Sensors_reg`	Rojobot sensor values
Output	1	`upd_sysregs`	Update system registers (control signal)

* Relative to the Rojobot.

BotConfig Input (BotConfig_in)

The 8-bit Bot Configuration register is used to alter the default configuration of the Rojobot emulation. Writing a value to the `WheelThrsh` field can alter the overall speed of the Bot relative to the values of the `LMSpd` and `RMSpd` (left and right motor speed) in the `MotCtl_in` register. Setting the `WheelThrsh` to a higher number slows down the Rojobot because the motors have to run longer (e.g. be sampled more times) to exceed the wheel threshold and move or turn the Rojobot.

Bit 7	6	5	4	3	2	1	Bit 0
0	0	WheelThrsh					VMS

WheelThrsh	Wheel Threshold. This 5-bit binary value sets the wheel movement threshold for both the left and right motor. Writing a large value slows the BOT down. Writing a small value speeds the BOT up. The minimum wheel threshold should be 4. Anything smaller than that may result in incorrect behavior. The WheelThrsh field is only valid when Variable Motor Speed is enabled (VMS = 1). The default wheel movement threshold is 5.
VMS	Variable Motor Speed Enable. This bit enables or disable variable motor speed. Setting the bit to '1' causes all three of the left and right motor speed bits to control the simulated speed of the wheel motor. Setting the bit to '0' causes only the least-significant bit of the left and right motor speed bits to be read (on/off control). The default is VMS disabled.

Enabling variable motor speed control and the adjusting the wheel threshold provides flexibility in the way the BOT operates. For example, assume the `WheelThrsh` is set to 5 for each wheel and both the left and right motor speed are set to 001_2. That means it will take .075 sec (the approximate delay in the main loop) x 5 (the wheel threshold) = .375 seconds of real-time for the Rojobot to move one position or turn 45 degrees. Now, let's change `WheelThrsh` to 10 but leave the motor speeds at 001_2. It will take .075 sec x 10 = .75 seconds of real-time for the BOT to move or turn. Next, let's say we leave the Wheel Threshold at 10 but set the left motor speed to 011_2 and the right motor speed to 100_2. First the Rojobot emulator will adjust both motor speeds to 100_2 (The faster of the two speeds). Each time the `MotCtl_in` register is sampled the simulator will add 4 (100_2) to each of the wheel distance counters. The 3rd time through the main loop the wheel distance counters will exceed the threshold of 10 and the BOT will advance. Doing the math yields .075sec (main loop delay) x 3 times through the loop makes the accumulated distance 12 > 10 = . 225 sec (which is pretty fast – you might as well add a cape to your icon).

Motor Control Input (MotCtl_in)

The 8-bit Motor Control input port is used to control the left and right wheel motors. The two motors are controlled independently and can cause the Rojobot to move forward or backwards, stop, or make slow or fast right or left turns. The input contains the following control bits:

Bit 7	6	5	4	3	2	1	Bit 0
LMSpd[2:0]			LMDir	RMSpd[2:0]			RMDir

LMSpd[2:0]	Left motor speed. This 3-bit binary value sets the speed of the left motor, and thus how fast the left wheel is moving only if the Variable Speed enabled bit in the Bot Configuration register is set. If the Variable Speed bit is cleared (set to 0) then only LMSpd[0] is used This limits the motor control to on/off and direction.
LMDir	Left motor direction This bit sets the rotation direction of the left wheel. Setting the bit to '1' causes the wheel to move clockwise (forward). Setting the bit to '0' causes the wheel to move counterclockwise (reverse)
RMSpd[2:0]	Right motor speed. This 3-bit binary value sets the speed of the right motor, and thus how fast the right wheel is moving only if the Variable Speed enabled bit in the Bot Configuration register is set. If the Variable Speed bit is cleared (set to 0) then only RMSpd[0] is used This limits the motor control to on/off and direction.
RMDir	Right motor direction This bit sets the rotation direction of the right wheel. Setting the bit to '1' causes the wheel to move clockwise (forward). Setting the bit to '0' causes the wheel to move counterclockwise (reverse)

Location X Register (LocX_reg)

The Location X register returns the X (column) coordinate of the Rojobot on the world map The map is 128 rows x 128 columns. Columns are numbered from 0 (leftmost column in the world) to 127 (rightmost column). The register has the following contents:

Bit 7	6	5	4	3	2	1	Bit 0
0	X[6]	X[5]	X[4]	X[3]	X[2]	X[1]	X[0]

X[6:0]	X coordinate (column) of the Rojobot's location. This is an unsigned binary number running from 0 (left) to 127 (right). The Rojobot is placed in the middle of its world whenever the emulator is reset or restarted. In a 128 x 128 world the initial X-coordinate is 64 (0x40)

Location Y Register (LocY_reg)

The Location Y register returns the Y (row) coordinate of the Rojobot's location in the 128 x 128 map. Rows are numbered from 0 (top row in the world) to 127 (bottom row in the world). The register has the following contents:

Bit 7	6	5	4	3	2	1	Bit 0
0	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]

Y[6:0]	Y coordinate (row) of the Rojobot's location. This is an unsigned binary number running from 0 (top) to 127 (bottom). The Rojobot is placed in the middle of its world whenever the emulator is reset or restarted. In a 128 x 128 world the initial Y-coordinate is 64 (0x40)

Rojobot Information Register (BotInfo_reg)

The Rojobot Information register contains the current orientation (which way the front of the Rojobot is facing) and movement (based on the wheel motor direction and speed) of the Rojobot. The register has the following contents:

Bit 7	6	5	4	3	2	1	Bit 0
Mvmt[3:0]				0	Orient[2:0]		

Mvmt[3:0]	Current movement (or action) of the Rojobot. The Rojobot is stopped and faces East when the Rojobot emulator is reset or restarted.

The movement is specified as follows:

Mvmt[3:0] (hex)	**Action**	**Motor Settings**
00	Stopped	Left and right motor off, direction is "don't care"

04	Forward	Left and right motor forward (clockwise)
08	Reverse	Left and right motor reverse (counterclockwise)
0C	Slow left turn (1X)	Left motor off, Right motor forward -or- Left motor reverse, Right motor off
0D	Fast left turn (2X)	Left motor reverse, Right motor forward
0E	Slow right turn (1X)	Left motor forward, Right motor off, –or- Left motor off, Right motor reverse
0F	Fast right turn (2X)	Left motor forward, Right motor reverse

Orient[2:0] Orientation (or heading) of the Rojobot. The orientation is the direction the front of the Rojobot is facing, and hence the direction the Rojobot will move either forward or backwards. The desired heading is achieved by setting the motor control inputs to turn the Rojobot either right or left until it is facing in the desired orientation. The turn is ended by stopping both motors.

The orientation (compass heading) is specified as follows:

Orient[2:0]	**Heading (degrees)**	**Direction**
00	0	North
01	45	Northeast
02	90	East
03	135	Southeast
04	180	South
05	225	Southwest
06	270	West
07	315	Northwest

Sensors Register (Sensors_reg)

The Sensors register returns information about the environment around the BOT. The feedback is in the form of readings from the proximity and line tracking sensors. The register has the following contents:

Bit 7	6	5	4	3	2	1	Bit 0
Map[1]	Map[0]	x	ProxL	ProxR	BlkLL	BlkLC	BlkLR

Map[1:0] These bits contain the value of the map bits for the current location. They can be used by your application to take special action(s) based on the current location.

x Reserved. This bit is reserved for future enhancements (perhaps provided by you). The bit is driven to 0 in the current implementation

ProxL Proximity Sensor, Left. ProxL returns '1' if the Rojobot detects an obstacle

	to the left or in front of it. The sensor returns ‘0’ if there is no obstacle. As mentioned earlier in the specification, The Proximity sensors are detecting obstacles one location ahead of the Rojobot’s current position.
ProxR	Proximity Sensor, Right. ProxR returns ‘1’ if the Rojobot detects an obstacle to the right or in front of it. The sensor returns ‘0’ if there is no obstacle. As mentioned earlier in the specification, The proximity sensors are detecting obstacles one location ahead of the Rojobot’s current position.
BlkLL	Black Line Sensor, Left. BlkLL returns ‘0’ if the left front of the Rojobot is over a black line. The sensor returns a ‘1’ if no black line is present under the sensor.
BlkLC	Black Line Sensor, Center. BlkLC returns ‘0’ if the center front of the Rojobot is over a black line. The sensor returns a ‘1’ if no black line is present under the sensor.
BlkLR	Black Line Sensor, Right. BlkLR returns ‘0’ if the right front of the Rojobot is over a black line. The sensor returns a ‘1’ if no black line is present under the sensor.

Important: For a physical Rojobot the BlkLL and BlkLR signals could be used to sense when the Bot was veering away from the black line. All three sensors are asserted only when the Bot is sitting directly on top of a black line. In the emulation the RojoBot occupies 1 location, and so does a black line segment so there are only two possible values for the line sensor. {BlkLL, BlkLC, BlLR} = 3’b111 means that there is no black line under the Rojobot. {BlkLL, BlkLC, BlLR} = 3’b000 means that the Rojobot is over a black line.

“Update System Register” signal (upd_sysregs)

The Rojobot produces an output called *upd-sysregs*. This output emits a 0->1->0 pulse every time the Rojobot state registers are updated, whether or not any of the registers values have changed. *upd-sysregs* can be used polled or used as an interrupt source to the application CPU to trigger the application to examine the new state and take appropriate action.