BotSim 3.1
Functional Specification

Revision 3.1

Table of Contents

Introduction

The Rojobot (**<u>Ro</u>**y and **<u>Jo</u>**hn's **<u>Bot</u>)** Simulator 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 robot (also called the "BOT or BotSim") gains information about its environment through two simulated sensors:

- Proximity sensors – The BOT has an 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 – The BOT has a line tracker sensor that consists of three pairs of LEDs and phototransistors (left, center, and right) that indicate the presence or absence of a black line below the front of the BOT. The value of the sensor is returned in 3 bits (BlkLL, BlkLC, and BlkLR). Each of the sensors will be set to '0' if it 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. Putting a sensor on top of a black line would cause no reflection and the sensor would return '0'.

Note the relationship between these two sensor types. The proximity sensors are looking ahead of the BOT so they can be used to keep the BOT from crashing into an obstacle. The line tracker sensors are looking directly under the front of the BOT so they can be used to guide the BOT along a predetermined path. Also note that the BOT only changes locations when it is moving forward or reverse. All turns are done in place.

The RojoBot lives and moves in a 128 x 128 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 floor – provides no obstacle for our intrepid little BOT
- Black line – indicates that the front of the BOT is over a black line
- Obstacle – indicates that the location is occupied by a barrier or 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. Makes your imagination run wild, doesn't it.

As a user of BotSim, you will interface your hardware to it through the byte-oriented register and input interface described in the next section. You can think of the BotSim as a "black box" and not delve into its implementation unless you are interested. This would be typical of IP that you procured from a 3rd party for your project.

New to BotSim 3.1

BotSim 3.1 (Bot 3.1) adds functionality to the BOT:

- The `Sensors` register now includes the value of the map location the BOT is on top of (2'b00= Open floor, 2'b01 = Black Line, 2'b10 = Obstruction (wall) and 2'b11 = Reserved). A BOT control application can read the `Sensors` register and take special action based on the value of the location in the world map. For example, let's say you want to use the "Reserved" code on the world map to represent a land mine instead of a Black Line. You could create a custom map with the BOT World Development Kit that litters the black line with land mines. Your BOT could read the Sensors register when the BOT moves to a new location and destroys the BOT (or maybe moves it to a new location) when the BOT is on top of the land mine. The possibilities are endless.

- Variable motor speed. The "standard" BOT ignores all but the lower bit for each wheel in the `MotCtl_in` register. This limits BOT operation to on/off and forward/reverse. When Variable Motor Speed is enabled, the BOT uses all three of the motor control bits, allowing the application to adjust its speed. The BOT still rotates in place when it is turning, but with higher speed settings the BOT will turn more quickly. The same holds true when the BOT is moving forward or backwards. Higher speed settings will cause the BOT to move more quickly. Even though the right motor speed and left motor speed bits could be set to different values, Bot 3.1 does not allow that. When the motor control speeds are different, the BOT will make both wheels turn at the faster of the two speeds. This simplification is in place to avoid having to account for veering and for gradual turns in the simulation.
- Adjustable Wheel Threshold. Bot 3.1 allows the user to adjust the Wheel threshold, thus providing an additional way to control the speed of the BOT. It is important to understand the function of the Wheel Threshold in the BOT simulation and how it relates to the motor speed.

Bot 3.1 BOT Movement Algorithm

The BOT simulation maintains two distance counters (one per wheel) that store the amount of simulated "distance" a wheel has moved since the `MotCtl_in` register was last sampled. In Bot 3.1 the `MotCtl_in` register is sampled roughly every 75msec (50msec 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 BotSim accumulates changes in the distance counters until they pass a "wheel move threshold." The "wheel move threshold" simulates enough wheel movement to cause the BOT to move to its next location or change orientation it is turning.

Each time the `MotCtl_in` register is sampled (once for each pass through the BOT simulator main loop) the BOT simulator adds the value of the motor speed bits to the distance counter. When the wheel threshold is exceeded the BOT advances forward or backwards or turns 45 or 90 degrees depending on the state of the forward/reverse bits for each motor. When the BOT is put in a slow right or left turn it will turn 45 degrees when the wheel threshold is exceeded. When the BOT is set to a fast right or left turn it turns 90 degrees when the wheel threshold is exceeded. When both wheels are turning forwards or backwards then the BOT will advance (or retreat) in position when the threshold is exceeded. Adjusting the Wheel Threshold can then be used to speed up or slow down the overall BOT movement relative to the motor speed control inputs. Setting the Wheel threshold to a higher number slows down the BOT because the simulated motors have to run longer (e.g. be sampled more times) to exceed the threshold and advance the BOT. Variable motor speed must be enabled in the `BotCtl` register to modify the wheel threshold.

The variable motor speed and the adjustable wheel threshold work together to allow much more flexibility in the way the BOT operates. For example, assume the wheel threshold is set to 5 for each wheel and both the left and right motor speed is set to 001_2. That means it will take .075 sec (the approximate delay in the BOT simulator's main loop) x 5 (the wheel threshold) = .375 seconds of real-time for the BOT to move one position or turn 45 degrees. Now, let's change the Wheel Threshold 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 BOT simulator 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 BOT).

BotSim 3.1 Register Interface

Bot31.v outputs six 8-bit registers that contain information about the BOT. The BOT is controlled by a single 8-bit input that is used to control the two motors that drive the wheels. The BOT is configured with a single 8-bit register that controls the wheel threshold and whether variable speed is enabled or disabled. The registers and the input are connected to your hardware through a simple port-based interface and are set to their initial values when BOT simulator is initialized.

The BOT simulator also implements an *upd_sysregs* signal which toggles when the BOT output registers are updated.

The *bot.v* registers are meant to consume a block of 8 I/O port addresses in an embedded processor SoC design. To do this, the Bot Configuration register shares the same Port ID as the BotInfo_reg register. The Bot Configuration register is read-only to the BotSim system and the BotInfo register is output-only so this is OK.

Since you will be designing the I/O interface your registers can be based at any I/O port address and you can reserve as many I/O port adddresses as you would like.

The BotSim register interface is summarized in the following table:

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

* Relative to the BotSim module.

BotConfig Input (BotConfig_in)

The 8-bit Bot Configuration register is used to alter the default configuration of the BOT. 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. See the "Bot 3.1 BOT Movement Algorithm" section earlier in this document for a more complete explanation.

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.

Motor Control Input (MotCtl_in)

The 8-bit Motor Control Input 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 BOT motors 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 BOT motors 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 BOT 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 address) of the BOT. This is an unsigned binary number running from 0 (left) to 127 (right). The BOT is placed in the middle of its world whenever the BOT simulator is reset. 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 BOT 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 address) of the BOT. This is an unsigned binary number running from 0 (top) to 127 (bottom). The BOT is placed in the middle of its world whenever the BOT simulator is reset. In a 128 x 128 world the initial Y-coordinate is 64 (0x40)

BOT Information Register (BotInfo_reg)

The BOT Information register contains the BOT's current orientation (heading) and movement (action caused by the wheel settings). 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 BOT. Mvmt is based on the direction the BOT's wheels are turning. The BOT is stopped and faces East in its world when the BotSim is reset.

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 BOT. The orientation of the BOT is the direction the BOT is moving in the world. The desired direction is achieved by setting the motor control inputs to turn the BOT either right or

left until it reaches the desired orientation and then changing them (either stopped, forward or reverse) to stop the turn once the BOT is pointed in the proper direction

The orientation (compass heading) is specified as follows:

Orient[2:0] (hex)	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) [Modified for BOTSim 3.1]

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. See the "New to BotSim 3.1" section earlier in this document for a more complete explanation

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

ProxL Proximity Sensor, Left. ProxL returns '1' if the BOT 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 Bot's current position.

ProxR Proximity Sensor, Right. ProxR returns '1' if the BOT 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 Bot's current position.

BlkLL Black Line Sensor, Left. BlkLL returns '0' if the left front of the Bot 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 Bot 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 BOT is over a black line. The sensor returns a '1' if no black line is present under the sensor.

IMPORTANT: IF A PHYSICAL ROJOBOT EXISTED ITS BLKLL AND BLKLR SIGNALS COULD BE USED TO SENSE WHEN THE BOT WAS VEERING AWAY FROM THE BLACK LINE. ALL THREE SENSORS ARE ASSERTED WHEN THE BOT IS SITTING DIRECTLY ON TOP OF A BLACK LINE. IN THE BOTSIM THE ROJOBOT OCCUPIES 1 LOCATION AND SO DOES A BLACK LINE SEGMENT SO THERE ARE ONLY TWO POSSIBLE VALUES FOR THE LINE TRACKER. {BLKLL, BLKLC, BLLR} = 3'B111 MEANS THAT THERE IS NO BLACK LINE UNDER THE BOT. {BLKLL, BLKLC, BLLR} = 3'B000 MEANS THAT THE BOT IS OVER A BLACK LINE.

Left Motor Distance Counter (LMDist_reg)

NOTE: THIS DISTANCE COUNTER FEATURE HAS BEEN DEPRECATED. DO NOT RELY ON THE DISTANCE COUNTERS IN YOUR BLACK LINE FOLLOWING ALGORITHM.

The Left Motor distance register indicates how far the left wheel has moved. The distance counter is incremented by 1 every time the motor control inputs are sampled and the left motor speed is greater than 0. The distance counter is cleared every time the BOT's location or orientation changes.

D7	D6	D5	D4	D3	D2	D1	D0

D[7:0]	Distance [7:0]. This is the 8-bit binary value of the left motor distance counter.

Right Motor Distance Counter (RMDist_reg)

NOTE: THIS DISTANCE COUNTER FEATURE HAS BEEN DEPRECATED. DO NOT RELY ON THE DISTANCE COUNTERS IN YOUR BLACK LINE FOLLOWING ALGORITHM.

The Right Motor distance register indicates how far the right wheel has moved. The distance counter is incremented by 1 every time the motor control inputs are sampled and the right motor speed is greater than 0. The distance counter is cleared every time the BOT's location or orientation changes.

D7	D6	D5	D4	D3	D2	D1	D0

D[7:0]	Distance [7:0]. This is the 8-bit binary value of the right motor distance counter.

“Update System Register” signal (upd_sysregs)

The BOT simulator produces an output called *upd-sysregs*. This output emits a 0->1->0 pulse every time the output registers are updated, whether any of the registers values have changed. *upd-sysregs* can be used as an interrupt source to an embedded processor or by hardware to indicate, for example, that a display should be updated.