BotSim 2.0
Functional Specification
(Last Updated: 16-Oct-2016)

Table of Contents

Introduction

The Rojobot (**Ro**y and **Jo**hn's **Bot)** 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.

Register Interface

Bot.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 registers and the input are connected to your hardware through a simple port-based interface and are set to their initial values when BotSim is reset.

The *bot.v* registers are meant to consume a block of 8 I/O port addresses in an embedded processor SoC design. However, you will be designing the I/O interface so your registers can be based at any I/O port address and you can reserve as many I/O port addresses as you would like.

BotSim also implements an *upd_sysregs* signal which toggles roughly every 50ms whether the Bot output registers are updated or not.

The BotSim register interface is summarized in the following table:

Direction*	Bits	Name	Description
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
Input or Output	8	*SPARE*	Spare register. You could, for example, use this register to hold the information needed to aim that death ray you've been thinking of implementing.

* Relative to the BotSim module.

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

IMPORTANT: BOTSIM 2.0 ONLY USES LMSPD[0] AND RMSPD[0]. THE OTHER SPEED BITS

ARE IGNORED. THIS MEANS THAT YOU CAN ONLY PROVIDE A SIMPLE ON/OFF CONTROL FOR EACH MOTOR. SET LMSPD[0] OR RMSPD[0] TO '1 TURN THE RESPECTIVE MOTOR ON. SET THE BIT TO '0' TO TURN THE MOTOR OFF.

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.

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.

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 BotSim 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 BotSim 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 the motor control inputs (either stopped, forward or reverse) to stop the turn once the Bot is pointed in the proper direction.

HINT: Turning the BOT is an application of closed loop control. Since the Bot is not guaranteed to complete a 45 degree turn in the interval between `upd_sysregs` *pulses the following steps should be followed:*

- *Read/save the starting orientation of the bot*
- *Start the Bot turn*
- *update/read the saved orientation on every* `upd_sysregs` *pulse*
- *Stop the Bot turn when the orientation has changed to your desired ending orientation*

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)

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
x	x	x	ProxL	ProxR	BlkLL	BlkLC	BlkLR

xxx Reserved. These bits are reserved for future enhancements (perhaps provided by you) They are 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 BOTSIM 2.0, THE ROJOBOT OCCUPIES 1 LOCATION ON THE MAP 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 BotSim produces an output called *upd_sysregs*. This output emits a 0->1->0 pulse whenever the output registers are updated. *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.

IMPORTANT: *UPD_SYSREGS* IS TOGGLED WHETHER OR NOT THE VALUE OF ANY OF THE OUTPUT REGISTERS HAS CHANGED SINCE THE LAST UPDATE. THIS HAS CAUSED SOME CONFUSION IN THE PAST.