Building your own BOT World
BOTWorld Development Kit

Revision 4
(Vivado Release)

Table of Contents

Introduction 1
List of Files 2
BOT World Construction Rules 3
Before you start 4
Building a new BOT World step-by-step 4

Introduction

This document describes the steps needed to build a new world map for the Rojobot BOT simulator. It is not a difficult task; most of the work is done by the Vivado IP Integrator (installed as part of the Vivado tool suite). The BOT World Development Kit includes several scripts and configuration files to simplify the process.

List of Files

The BOTWorld Development Kit is provided as a single .zip file that is included in the BOT 3.1 and BOTWorld Development Kit release package. The BOTWorld Development Kit includes the following files:

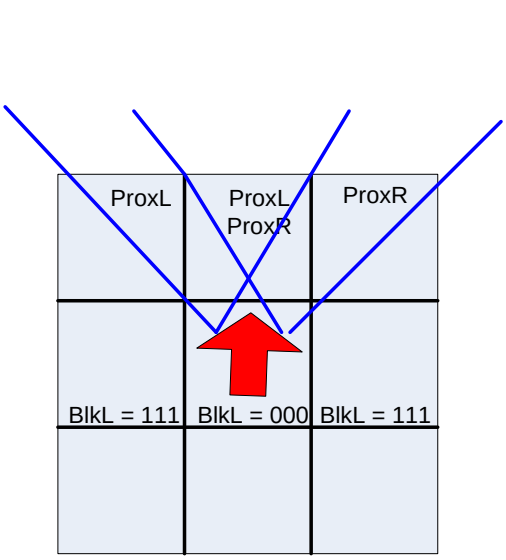
Name	Description
Documentation files	
Building your own BOT World (pdf)	Provides instructions for building your own BOT Worlds (this document)
readme.txt	Latest information on this release
Configuration files and scripts	
mapit_coe.pl	A PERL script that should be run prior to opening creatin the new World Map memory. This script converts a world map text file built from one of the template maps to the .coe file used by the Memory Generator to initialize the contents of the world map ROM.
Other files	
\world map templates (folder)	Contains the existing template.txt files and a blank template.txt file. One of these files can be used as the starting point for your new world.
\newmaps	Contains copies of the files used to create your own BotWorld. It is provided as a convenience to simplify the amount of directory and command line manipulation.

BOT World Construction Rules

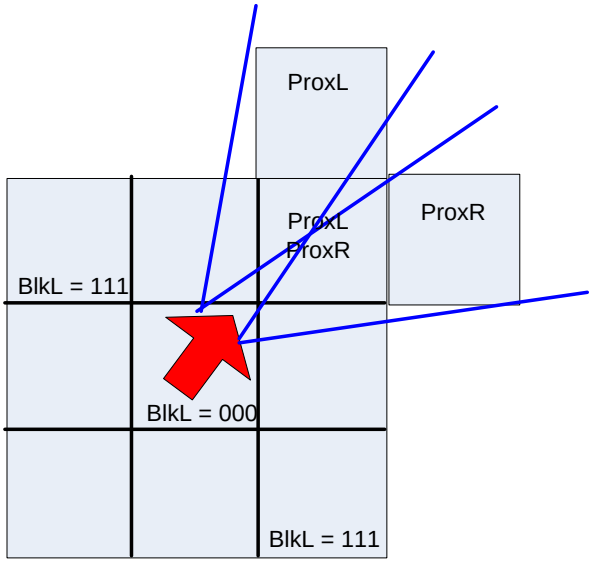
The BOT exists in a 128 (0 to 127) x 128 (0 to 127) grid with each {x,y} coordinate containing one of 4 values:

- 00 – Floor (Bare ground)
- 01 – Black line
- 10 – Obstruction/Wall
- 11 – Reserved

The BOT can sense and report whether it is on top of a black line location through its three black line sensors and can sense whether it is approaching an obstruction (a wall in the current implementation) through its two proximity sensors. The proximity sensors look ahead to the next location when it is facing North, South, East, or West. As an artifact of the implementation, the BOT looks ahead two locations on the diagonal when it is facing Northwest, Southwest, Northeast or Southeast. The sensor behavior can be summarized with the following illustrations:



North



Northeast

In order to properly handle the sensors, a new world map should conform to the following rules:

- The BOT is initialized to start in the middle of the world at coordinates {64, 64} ({40_{16}, 40_{16}} facing East. As a result, the start of your world should be at that location. Since the BOT is a line following robot, the assumption is that the black line for your world starts there.
- Leave some bare ground between black line and a wall
  - Minimum 1 location on straight runs
  - Minimum 2 locations on the diagonals
- Do not use the bottom 8 rows (120 to 127) of the world. Since the BOT world is intended to be displayed at VGA resolution of 640 x 480 w/ a resolution of 4 rows per pixel,

features put in the last 8 rows will not be displayed. The template files provided in the development kit include a surrounding wall to contain the BOT. This bottom of this wall is placed above those last 8 rows. Note: You may have to adjust the number of wall lines that are off the display if you change the display resolution.

Before you start

You will need a working *perl* interpreter to use mapit_coe.pl to generate the initialization file for the ROM created by Memory Generator. If you don't have PERL on your PC there are several Windows-based *perl* packages that can be downloaded from the internet. The CYGWIN environment, for example, provides Windows-friendly instances of most of the common Linux utilities including *perl*. ActiveState (http://www.activestate.com/activeperl/downloads) provides a *perl* interpreter and the common *perl* utilities and packages including the *perl* package manager (*ppm*) for 32-bit and 64-bit Windows systems. PERL is included in most of the popular Linux releases.

Building a new BOT World step-by-step

Building a BOT world for use by the BOT simulator is not hard. It will take longer to design your world then it will to create the world_map ROM. The steps are:

1. Modify one of the template files to create your new map and move the file to the *\newmaps* directory. This is best done by using a text editor (MS wordpad is fine…it's what I use) to build your map by writing 0, 1, 2, or 3 to each of the locations. The value 3 is reserved for your use should you wish to add an additional feature (say a simulated land mine) to your world.

DEPENDING ON THE CUSTOM FEATURE THAT YOU ADD AND HOW YOU IMPLEMENT THE ADDITIONAL FEATURE YOU MAY BE ABLE TO SUPPORT THE FEATURE WITHOUT MODIFYING THE BOT SIMULATOR SOURCE CODE, BUT IF NOT, THE BOT 3.1 ASSEMBLY LANGUAGE SOURCE IS INCLUDED IN THE BOT 3.1 RELEASE

Incidentally, the highlighted map files that are included in the project releases were generated by hand using the Text Highlighter tool in Microsoft Word. MS Word was set to Landscape mode and the text size was adjusted so that it was easy to see a section of the map on one screen. Each location was highlighted by clicking, dragging, or whatever to connect the dots.

2. Run the *mapit_coe.pl* PERL script to generate a ROM initialization file (.coe) for the Core Generator.

   mapit_coe.pl should be invoked from the command line as follows:

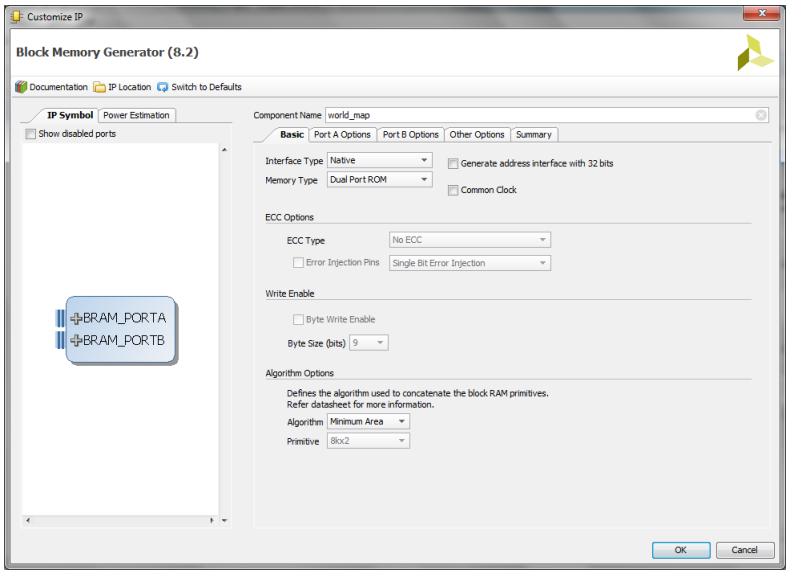
   > C:\ perl mapit_coe.pl <newmap.txt >newmap.coe

   Where *newmap.txt* is the world map file you created and *newmap.coe* is the initialization file you are going to specify to the Core Generator.

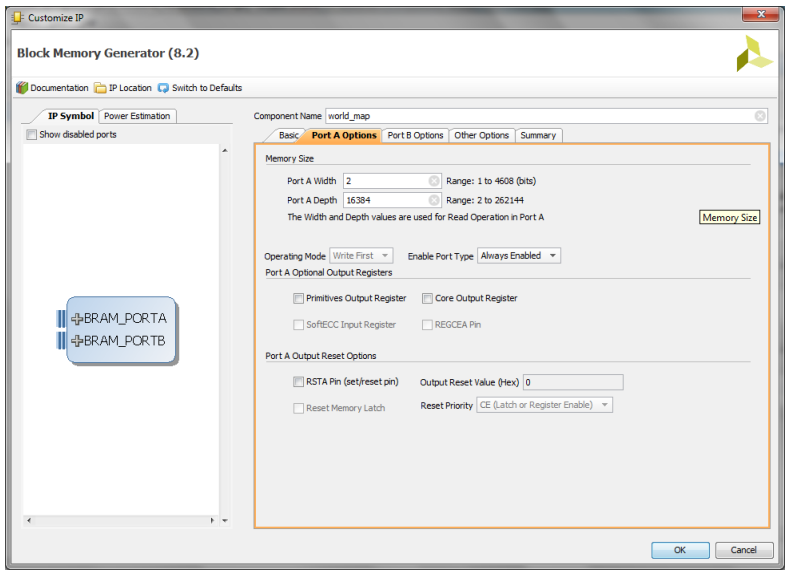
The PERL script uses *stdin* for the input and *stdout* for the output. '< 'on the command line redirects *stdin* to your .txt file and' >' redirects *stdout* to your .coe file.

3. Configure the Block Memory Generator to build the ROM and initialize it with the .coe file bits. This is done most simply by following these steps:
   a. Build your Rojobot systems as you have in the past except do not include `world_map.v` and `world_map.ngc`. These files will be provided by the Block Memory Generator. If you are building a BOT 3.x system, don't forget to add a port for the `Bot_Config_reg` to your bot.v instantiation.
   b. From the Flow Navigator select *Project Manager/IP Catalog*. This will bring up the IP Catalog.
   c. Type "Block Memory Generator" (the first few characters will suffice) in the Search window of the IP Catalog.
   d. Invoke the Block Memory Generator by double clicking on the item in the IP Catalog or by right-clicking and selecting *Customize IP*… This will open the Block Memory Generator

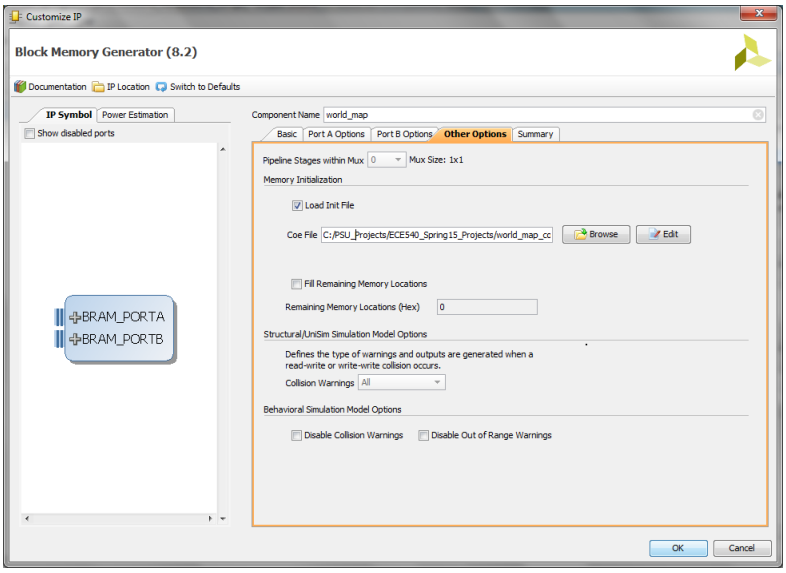
   e. **Block Memory Generator *`Basic`* tab** – Component name should be *world_map*. Interface type should be *native*. Memory type should be *Dual Port ROM*. *Generate address interface with 32 bits* and *Common Clock* check boxes should not be checked. The *Minimum Area* algorithm should be selected.



   f. **Block Memory Generator *`Port A`* `Options` and *`Port B`* `Options` tabs** – Port A and Port B options should be *Read Width: 2*, *Read Depth: 16384*. *Enable Port* should be set to *Always Enabled*. Accept the defaults for the other parameters.

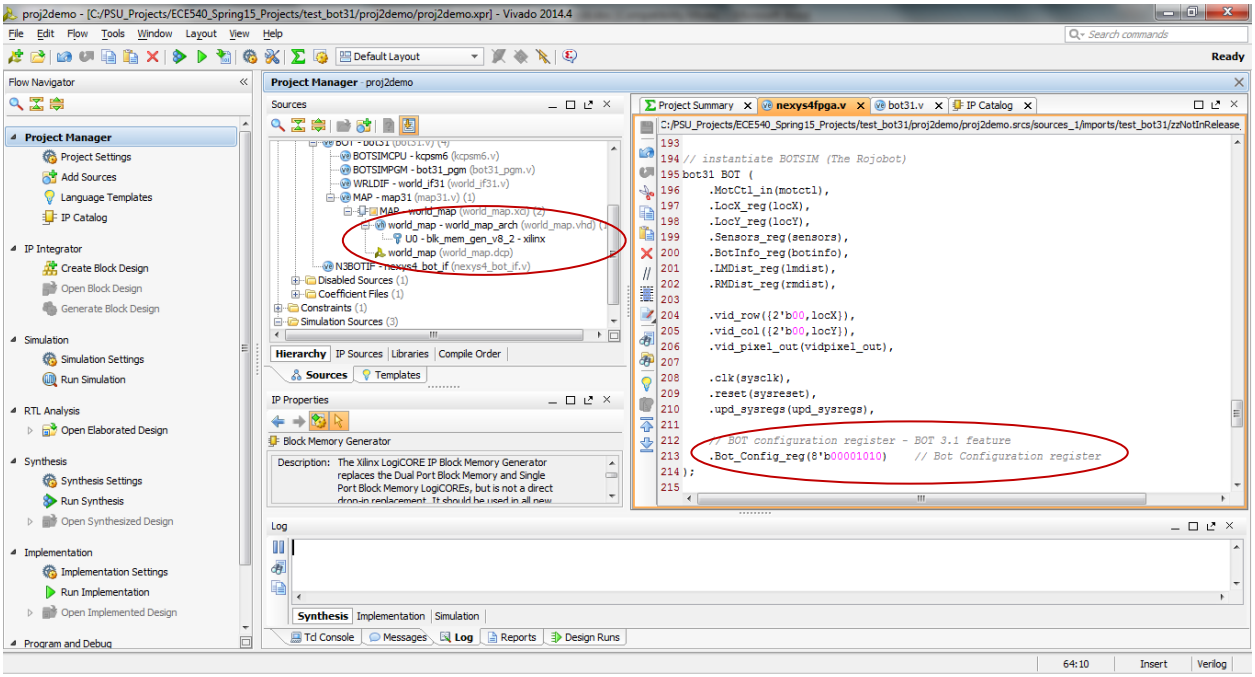


g. **Block Memory Generator** `Other Options` **tab**. Select *Load Init File*. Browse to and select your new *map.coe* file. Click on <OK>



h. (Optional) Generate the Output Products. This will launch a synthesis run which could take a few minutes to run. You should get a message saying the IP was successfully generated when the synthesis run completes.

4. As long as your new world map is named *world_map* the Vivado project manager should automatically add it into your project. NOTE: When I ran the Block Memory Generator its wrapper file was called `world_map.vhd` (VHDL) even though my target language in Setting/General was set to Verilog. While this was unexpected, it did not appear to cause any problems – Vivado manages mixed language projects with no problem.


proj2demo - [C:/PSU_Projects/ECE540_Spring15_Projects/test_bot31/proj2demo/proj2demo.xpr] - Vivado 2014.4
File Edit Flow Tools Window Layout View Help
Ready
Flow Navigator
Project Manager
Project Settings
Add Sources
Language Templates
IP Catalog
IP Integrator
Create Block Design
Open Block Design
Generate Block Design
Simulation
Simulation Settings
Run Simulation
RTL Analysis
Open Elaborated Design
Synthesis
Synthesis Settings
Run Synthesis
Open Synthesized Design
Implementation
Implementation Settings
Run Implementation
Open Implemented Design
Program and Debug
Project Manager - proj2demo
Sources
BOTSIMCPU - kcpsm6 (kcpsm6.v)
BOTSIMPGM - bot31_pgm (bot31_pgm.v)
WRLDIF - world_if31 (world_if31.v)
MAP - map31 (map31.v) (1)
world_map - world_map_arch (world_map.vhd)
U0 - blk_mem_gen_v8_2 - xilinx
world_map (world_map.dcp)
Disabled Sources (1)
Coefficient Files (1)
Constraints (1)
Simulation Sources (3)
Hierarchy IP Sources Libraries Compile Order
Sources Templates
IP Properties
Block Memory Generator
Description: The Xilinx LogiCORE IP Block Memory Generator replaces the Dual Port Block Memory and Single Port Block Memory LogiCOREs, but is not a direct
Project Summary nexys4fpga.v bot31.v IP Catalog
C:/PSU_Projects/ECE540_Spring15_Projects/test_bot31/proj2demo/proj2demo.srcs/sources_1/imports/test_bot31/zzNotInRelease
193
194 // instantiate BOTSIM (The Rojobot)
195 bot31 BOT (
196 .MotCtl_in(motctl),
197 .LocX_reg(locX),
198 .LocY_reg(locY),
199 .Sensors_reg(sensors),
200 .BotInfo_reg(botinfo),
201 .LMDist_reg(lmdist),
202 .RMDist_reg(rmdist),
203
204 .vid_row({2'b00,locX}),
205 .vid_col({2'b00,locY}),
206 .vid_pixel_out(vidpixel_out),
207
208 .clk(sysclk),
209 .reset(sysreset),
210 .upd_sysregs(upd_sysregs),
211
212 // BOT configuration register - BOT 3.1 feature
213 .Bot_Config_reg(8'b00001010) // Bot Configuration register
214);
215
Log
Synthesis Implementation Simulation
Tcl Console Messages Log Reports Design Runs
64:10 Insert Verilog

<finis>