Using Xilinx Data2MEM to Patch Block RAMs

Author **Message**

MichaelM **Post subject:** Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Fri Jul 19, 2013 1:41 pm

offline

Joined: Mon Apr 23, 2012 12:28 am
Posts: 555
Location: Huntsville, AL

enso started me down this path, so I thought I'd post the results of my investigations.

First, it is definitely possible to use Xilinx's Data2MEM tool to change the contents of one or more block RAMs.

A key to using Data2MEM to perform the gymnastics necessary is to define the memory space, and to correctly associate the netlist names of the block RAMs to their locations in the routed design.

This post (the one at the end of the chain from Walter Dvorak) put me on my way. The example in the Data2MEM User's Guide was good, but the issue was finding the locations of the Block RAMs that would not impact performance too much. The syntax of the Data2MEM BMM (Block RAM Memory Map) file is pretty straightforward and consistent with other Xilinx tools. The post at the link gave a significant clue on how to determine the locations post-PAR of the block RAMs which will contain the program image. Walter Dvorak indicates that if the BMM file provided to the tools does not lock (LOC =) or place (PLACE =) the memory blocks, but simply defines the address range, the partitioning (bus layout), and the type), then BitGen will generate a new BMM with the _bd suffix appended to the filename you gave. (*The file will be written by BitGen in the same directory that contains your BMM file.*)

The following is my BMM file (for my M16C5x PIC16C5x-compatible soft-core microcomputer project), which is part of the project like the UCF (User Constraints File):

Code:

```
////////////////////////////////////////////////////////////////////////////////
//
//  Block RAM Memory Map file: M16C5x Block RAM (4096 x 12) Program Memory
//
////////////////////////////////////////////////////////////////////////////////

ADDRESS_SPACE PROM RAMB16 [0x00000000:0x000017FF]

    BUS_BLOCK
        Mram_PROM1 [ 3:0] LOC = X0Y3;
        Mram_PROM2 [ 7:4] LOC = X0Y4;
        Mram_PROM3 [11:8] LOC = X0Y5;
    END_BUS_BLOCK;

END_ADDRESS_SPACE;
```

The key words ADDRESS_SPACE and END_ADDRESS_SPACE; define a range of addresses that will be loaded with user provided data if the "address" of the input data fits within the range. Following ADDRESS_SPACE, a name for the address range is required. It is simply a name, or tag, that has no relation to the netlist. In this case, I named the address range PROM because that's the name of the the ROM/RAM variable in my source file. Following the name is a memory type. This is the type of memory that is used. In other words, its the name given by Xilinx to the various block RAMs they have in their products. Spartan II, and Virtex family parts implement 4kb Block RAMs, which Xilinx refers to as RAMB4. The Spartan 3/3E/3A/3AN and Spartan 6 implement 18kb block RAMs. Thus, a RAMB18 would indicate to Data2MEM that you will be using the parity and the data lines, and RAMB16 indicates that the parity lines will not be used. To create a 4096 x 12 PROM for my PIC16C5x-compatible processor core, I need three block RAMs each organized as 4096 x 4. So I specified the memory type of the PROM address space as RAMB16 instead of RAMB18. (**NOTE:** *in my designs I infer ROMs/RAMs. I use the recommended coding styles found in the "Light Bulb" tool if ISE.*)

Following the memory type declaration you have to define the address range (in bytes) represented by your block RAM memory. In my case, the three block RAMs actually provide 6kB of storage. I initially made an error in specifying my memory array as having an address range [0x0000_0000:0x0000_0FFF], or 4096 12-bit words. When Translate read my BMM file, it reported that it expected an address range of 0x1800 (6kB) instead of the 0x1000 (4kW) I had specified. Thus, I had to set the address range of the PROM address space to [0x0000_0000:0x0000_17FF].

Inside the ADDRESS_SPACE block, you need to specify the bit/byte lanes and the distribution/arrangement of the block RAMs. The example in Appendix A of the Data2MEM User's Guide shows a 2k x 64 arrangement. In that example, they are targeting RAMB4 block RAMs which support a 512 x 8 organization. With the 8 byte lanes required, 8 RAMB4 block RAMs are specified in four BUS_BLOCKs. Each BUS_BLOCK represents 512 x 64, for a full range of 4kB per BUS_BLOCK, or 16kB for the entire address space which consists of 32 (512 x 8) block RAMs.

In the case of my little core, the arrangement I want is 4096 x 12. Therefore, I assigned three 4-bit bus lanes. To do this I need to know the correct netlist name of each of the block RAMs automatically inferred by the synthesizer. My solution to this problem was two fold. First, I expect the Xilinx synthesizer to synthesize a multi-bit component and assemble them into aggregate/composite components LSB/lsb first. Therefore, I expected that the synthesizer would assign the LSN, i.e. bits 3:0, to the first, or lowest numbered block RAM.

Second, rather than using the floorplanner to manually assign the location of the three block RAMs, I simply ran the synthesizer and PAR tools (without the BMM included/added to the project) and then looked in the routed FPGA (with the FPGA Editor) for the three block RAMs that I know would be used for implementing my program memory array. I then took the assigned names and pasted them into the BUS_BLOCK section of my BMM file, assigned the three 4-bit bit lanes, and reran the tools with the updated BMM added to the project to generate the my_BMM_bd.bmm file that BitGen outputs when my BMM is provided without LOC constraints for the block RAMs.

In addition to using Project - New Source, Project - Add Source, or Project - Add Copy of Source to add the BMM file to the project, a modification is required to the process parameters of the netlist translator, Translate. So after adding the BMM file to the project, Translate's "Other NgdBuild Command Line Options" must be set to access the BMM file. In ISE, open the process properties pop-window and select the Translate Properties option. At the bottom the Translate properties window/pane is the "Other Ngdbuild Command Line Options" property. By default it is blank. Set the property to something like "**-bm** *myBMM<.bmm>*". (*The file extension is assumed to be bmm, so does not have to be included in the filename unless it's a BMM file which uses another file extension.*)

Generate the Mapped, Placed, and Routed design, and then run BitGen to generate the configuration bitstream file. If the BMM file provided does not include LOC or PLACE constraints, BitGen will find and output the locations of the placed block RAMs in the

my_BMM_bd.bmm file. I copied those placements into my BMM file and converted them to LOC = instead of PLACE = constraint that BitGen output.

There is likely a simpler way to get the netlist names for the block RAMs, but the above process was successfully applied. The Advanced HDL Analysis section of the Synthesis report also provides the names of the synthesized block RAMs. What it does not do is provide the name of the various block RAMs in a synthesized component consisting of multiple Block RAMs. But armed with that information, you can find (using FPGA Editor) the netlist names of the various block RAMs in a multi-RAMBx component.

With the BMM file in its final form, the next step is to generate a data file that Data2MEM can use to modify the FPGA configuration image. Data2MEM accepts two file types for the data file: (1) ELF - Execute and Link Format; and (2) MEM. In the case of the MicroBlaze soft-core processor, the Xilinx tools emit ELF files. In my case, neither the Kingswood assembler that I use for my M65C02 soft-core, nor the MPLAB assembler that I use for my M16C5x soft-core emit ELF files. Instead, both emit various (EPROM programming) output file formats. I use the binary output format with the Kingswood A65 assembler, and Intel Hex output format with the MPLAB PIC assembler.

Thus, to patch the bitstream file with Data2MEM I needed to create a data file compatible with the MEM file format. The MEM file format is very simple. It essentially is a file containing ASCII Hex addresses and data. Addresses are preceded by an @ symbol, and are separated from the data by a space or a newline. Multiple data elements may be placed on a single line if they are separated by spaces. Their addresses are assumed to be sequential until another @address is provided.

Both the SMRTool microprogram support tool and the Bin2Txt utility that I have released directly generates MEM-like files that I simply read with the Verilog $readmemh() system function to initialize all memories in any of my designs. These files, generally known as memory initialization files, have an even simpler format than MEM files. They are simply either ASCII Hex (0..9, A..F) or ASCII Binary (1, 0) files. Thus, I decided to use the opcode output file option from MPLAB instead of the Intel Hex file to implement the required MEM file. I figured that the order of the ASCII hex nibbles (characters) would be the same as that of the Verilog memory initialization file. That assumption was completely wrong. In fact, the order of the nibbles in each data word expected by Data2MEM is completely reversed. The following is the initial few lines from the MEM file I used:

Code:

```
@0000
FFC
500
600
E0C
A20
700
80C
F20
FE2
80A
```

And the following code fragment is that of the memory initialization file that Verilog's $readmemh() system function loads to initialize the same block RAMs during synthesis:

Code:

```
CFF
005
006
C0E
02A
007
```

```
C08
02F
2EF
A08
```

The two notable differences are the **@0000** at the start of the MEM file, and the ***reversal of all the nibbles***.

Thankfully, Ultra Edit has a column mode, and with a few simple key strokes its easy to take the first (leftmost) column (normally interpreted as the most significant nibble by everyone but Data2MEM) and swap it with the last (rightmost) column. With this contortion made, I changed the baud rate constant and used Data2MEM to patch the bitstream by running Data2Mem from the command line:

<path>**data2mem -bm** my.bmm **-bd** code.mem **-bt** my<.bit> **-o b** new<.bit>

Downloading the modified bit stream into the FPGA resulted in execution of the test program as expected with the changed baud rate in effect.

Rather than running Data2MEM from a command line, it is possible to set an option in BitGen, and get BitGen to automatically run Data2Mem prior to the generation of its output bitstream file. Like setting the option for Translate to use a BMM file, you can set an option in BitGen to update block RAMs using an ELF/MEM file. (**Note:** *most of the discussions/examples in the Data2MEM User's Guide are given as if an ELF file will always be used. I ignored that and simply specified the data files with the .mem file extension. As the name of the utility implies, MEM files were likely the first file type supported by the tool.*) To have BitGen patch one or more block RAMs in the bitstream file with an update, open the process options window for BitGen ("Generate Programming File"), and set the "Other Bitgen Command Line Options" (in the General Options category) to something like:

-bd myMEM**.mem**.

Apparently BitGen uses the BMM file specified in Translate to automatically call Data2Mem behind the scenes. Using Data2Mem in either manner, the patching of a design's block RAMs prior to downloading is a great time saver. Furthermore, it precludes re-synthesis and PAR of the design. Thus, if the changes are all restricted to the contents of block RAMs, the potential failure of PAR to meet timing can be avoided on large designs that require several MPP/Smart Explorer passes to close timing.

(**Note:** *with an ELF file, it may be possible to do the operation in real-time as the bitstream file is being loaded. But it appears that there is a caveat to this option: you have to be connected to an FPGA with a hard-core that shows up as part of the JTAG chain. Since that's not the case here, I have not pursued this option any further.*)

Summary:

(1) Determine the netlist names of the block RAMs to be patched.
(2) Create a BMM file and attach it to the project file list. (Loc/Patch constraints optional at this time)
(3) Set Translate's "Other Ngdbuild Command Line Options" to refer to the BMM file: -bm myBMM<.bmm>
(4) Run Synthesis, MAP and PAR, and BitGen. Examine BitGen's myBMM_db.bmm file, and constrain the block RAMs define in myBMM.bmm to match.
(5) Create MEM file to patch into bitstream file.
(6) Set BitGen's "Other Bitgen Command Line Options" to enable real-time patching: -bd myMEM.mem
(7) Re-run MAP, PAR, and BitGen, and download patched bitstream file to target. (At this point it's possible to create a PROM file for the FPGA.)

I believe the preceding discussions and the summary process checklist above are accurate.

Hope this is useful to those of us using embedded soft-cores in FPGAs. I expect that I will save considerable amount of time in working with my M65C02/M16C5x development board as a result of this discovery.

_____
Michael A.

Top | profile | quote

enso

offline

Joined: Sat Sep 29, 2012 10:15 pm
Posts: 660

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sat Jul 20, 2013 12:11 am

Good job, Michael. I am happy you have data2mem working. Once you figure out how to make it work, which is no picnic, your turnaround time for firmware development goes to seconds instead of minutes.

Have you had any luck using multiple BRAMS occupying sequential locations? For instance, an 8K x8bits block of RAM at 0000-1FFF hex. I was trying to assign 4 2K spaces to 4 BRAMS, but could not get it to work.

I suppose it would be wiser to assign 2 bits per BRAM using bitlanes and avoid muxing. I may want to use 3 BRAMS for 6K, though...

Top | profile | quote

MichaelM

offline

Joined: Mon Apr 23, 2012 12:28 am
Posts: 555
Location: Huntsville, AL

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sat Jul 20, 2013 1:33 am

The memory for the M16C5x is 3 block RAMs. With three block RAMs the only way to create a 4096 x 12 memory is to build it as 3 block RAMs of 4096 x 4. My understanding of the process is that Data2MEM interprets the ram type, the address space, and the bus blocks to define the memory array.

I suppose that it's possible that the XST synthesizer follows rules that allow the tools to "know" how the memory arrays are partitioned when more than one block RAM is required. For the case you appear to describe, 8k x 8, there are several ways that the synthesizer may decide to implement the composite memory from four block RAMs: (1) four block RAMs organized as 8192 x 2 (no external multiplexer); (2) four banks of one block RAM organized as 2048 x 8 (four multiplexers); (3) two banks of two block RAMs 2x (two 4096 x 4) (two multiplexers).

I haven't read anywhere yet on how the synthesizer lays out the RAM. Perhaps it does not, because the BMM file is not passed as a parameter to the synthesizer. In the Xilinx flow, the BMM file is passed to Translate, and I guess, to MAP and PAR since these tools execute after Translate. Perhaps the first constraint for the organization/geometry of the block RAMs is set by Translate, and then further refined by MAP. I have not found any information on this issue.

For my block RAMs I declared them as

Code:

```
reg [11:0] PROM [0:4095];
```

In the code where I inferred the program memory, I initialized them with a memory initialization file using

Code:

```
initial
    $readmemh(MemInitFile, PROM, 0, 4095);
```

The interesting thing was that I created the MemInitFile so that the most significant nibble was the first character on the line, and the least significant was the third (last) character on the line. The first MEM file I prepared was made from the MemInitFile. All I did was to add an @0000 to the first line of the MemInitFile and save it off as a MEM file. Using Data2MEM to apply that file to the bitstream should have resulted in a working system, but it did not. I knew that something was going on, because the patched bitstream did not work.

I then used Data2MEM to examine the block RAM in the working and non-working bitstream files:

data2mem -bm myBMM -bt myGoodBitFile -d
and
data2mem -bm myBMM -bt myBadBitFile -d

When I compared the block RAM contents of the working file and the non-working file, I found that they were not the same as I expected them to be. To get the Data2MEM MEM file prepared correctly, I had to swap the first and third characters of each instruction word (single line per instruction), but I also LOCed the block RAMs in a sensible order. The unconstrained placement of the block RAMs placed the middle nibble on the lowest rank and the first and third nibbles together on adjacent block RAMs. This is how the unconstrained Place put them into the design. The block RAM placement is not as important as the geometry, but I fixed the geometry to make it easier to determine that the first and third nibbles were swapped.

In developing my little utility to take the Intel Hex file and convert it to a MEM file, I found another thing to be careful on. Data2MEM work in bytes. Thus, on my first cut, the program would read the four bytes that MPLAB generated for each instruction word. The bytes are output in little endian form. The record address is related to the number of bytes not 12-bit words. The most significant four bits of the most significant byte are output as 0s by MPLAB. So when I truncate each line in the MEM file to three characters (to represent each instruction word), they represent only 1.5 bytes to Data2MEM. To output the reset vector, which "starts" on a 1.5 byte boundary, I had to output three dummy characters and a MEM file address 3 bytes back from the end of the memory block.

What this means is that an organization like 4x (8192 x 2), which does not require any LUT multiplexers may require that the bits be reversed in each nibble. In other words, the MEM file format is bit-lane reversed from the memory initialization file. For an 4x(8192 x 2) memory organization/geometry, reversing the bit-lanes may be too much work.

My recommendation at this time, is that if you can tolerate the performance degradation that occurs when LUT-based multiplexers are used to select the correct 2048 x 8 RAM, then use a x8 bit lane assignment. For the BMM file, I expect that you'll need to put each block RAM in its own BUS BLOCK, and to declare it as a 2048 x 8 RAM. In this way, the issues I ran into regarding the construction of the MEM file should not be an issue.

_____
Michael A.

Last edited by MichaelM on Sun Nov 24, 2013 4:07 pm, edited 1 time in total.

Top | profile | quote

enso

offline

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sat Jul 20, 2013 2:40 pm

I wonder if reversing the order of the 3 PROMS inside the BUS_BLOCK would change the order of nybbles in the memory file.

Joined: Sat Sep 29, 2012 10:15 pm
Posts: 660

Top | profile | quote

MichaelM

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sat Jul 20, 2013 3:33 pm

offline

Joined: Mon Apr 23, 2012 12:28 am
Posts: 555
Location: Huntsville, AL

Tried several combinations. The issue is related to how the synthesizer views memory and how the other tools interpret the memory data files.

After several fruitless attempts. I simply LOCed PROM1 (bits 3:0), PROM2 (bits 7:4), and PROM3 (bits 11:8) in a "natural" bottom to top arrangement in the FPGA and dumped the bitstream file using Data2MEM to determine the order of the data in the MEM file. Since the three PROMx block RAMs were synthesized with an initial test program, I could do a before and after comparison to determine the MEM file's bit-lane/character order.

The first few MEM files I used in testing, I generated manually using the program memory output feature of MPLAB, and the column mode of Ultra Edit. This dump consists of 3 ASCII Hex digits per line. Swapping the first and third columns and adding an @0000 address specifier to the top of file creates a MEM file that is "full", i.e. has a value for each nibble in the block RAM array. With this MEM file, I could edit the baud rate field, run Bitgen, and download to my heart's content.

_____
Michael A.

Top | profile | quote

ElEctric_EyE

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sat Jul 20, 2013 11:32 pm

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

Michael, I also am watching with interest. If it works, it would cut down my 65Org16.b software development time significantly.
I'll hopefully have spare time next week or so to try apply your results. Work is very demanding as colleague has been on vacation...
Thanks for posting your results!

Top | profile | quote

MichaelM

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sun Jul 21, 2013 3:44 pm

offline

Joined: Mon Apr 23, 2012 12:28 am
Posts: 555
Location: Huntsville, AL

EEyE:

I am sure it will work for you. It's working well for me for the testing that I am dong for a soft-core microcomputer that I'm currently working on. I have been able to patch various updates of my test program into the working bitstream without regenerating it through synthesis/MAP/PAR. To achieve the constraints I have levied on the design, I have had to resort to using ISE 10.1i's MPP. It needs about 3 passes, on average, to close timing. Even though the target is a small FPGA, XC3S200A-4VQG100I, the time to complete the process and generate a new bitstream is approximately 10 minutes. I haven't had to wait that long for an assembler or compiler to complete for about 25 to 30 years, and for FPGAs about 10 to 15 years.

Using Data2MEM, it only takes a few seconds to initiate Bitgen, click on Program in iMPACT, click Yes to reload the modified bitstream file, and download it into the FPGA. I

think if you set your project up correctly, and maybe implement a utility program like I did for converting Intel Hex files into MEM files, you should see a substantial speed up in your development cycle with the 65Org16 core.

I am not familiar with the output file for the assembler that you are using, but MEM files are about as simple as you can make a programming file. I think the key for you will be to determine the layout of your internal memory array. The synthesis report will give you a clue about the name of the block RAMs, and Bitgen will actually tell you where they are placed.

What I haven't determined at this time is how the BMM file interacts, if it does, with Translate and Map to map the synthesis provided block RAM specification onto the block RAMs in the FPGA. In other words, the block RAMs in my current project can only be arranged in one form to provide the 4096 x 12 arrangement that I require, and I haven't determined if this arrangement is dictated by the synthesizer, or implemented by the placer.

If it it implemented by the synthesizer, then for each project, the arrangement of the block RAMs will have to be discovered using the FPGA editor, or some other method: synthesis directives to LOC specific components in absolute or relative terms to specific FPGA resources. If it is implemented in the placer, then the arrangement specified in the BMM file would likely be the mapping that the placer will use to set the geometry and arrangement of the block RAMs in the FPGA. If I were Xilinx, I would have the placer perform this function instead of the synthesizer; too much FPGA-specific knowledge in the synthesizer probably has a negative impact on its portability across multiple FPGA families.

If I have some time over the next week, I might try to figure out whether it's the BMM file or the synthesizer that determines out how the block RAMs are configured (geometry) and allocated (arrangement/placement/numbered). If I get some results that may be applicable to the 65Org16 core, I'll post my findings.

I think, however, that you can use the procedure I outlined earlier in this thread to develop a BMM and use Data2MEM with 65Org16. If you run into any difficulties, or find issues with the procedure I outlined, I'll be glad to give a hand. I am also working an odd schedule these days, but I'll try and respond to any questions or issues as quickly as I can.

_____
Michael A.

Top profile quote

enso

offline

Joined: Sat Sep 29, 2012 10:15 pm
Posts: 660

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Sun Jul 21, 2013 4:45 pm

Some people call me paranoid, but I just don't trust large tools. I really prefer to instantiate BRAMs - then I know exactly how they are wired. Sometimes it's a little more work, but it's really not that much more, and you generally have to do it only once.

Top profile quote

ElEctric_EyE

offline

Joined: Mon Mar 02, 2009

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 2:25 pm

Sidenote: Development stages seem to be working for me at about 4 a month period, whether making boards, working out bugs in Verilog etc and now this post. Interesting.

7:27 pm
Posts: 3049
Location: NC, USA

So I am again interested in trying to build the bridge to use data2mem for my project to cut down development times. The reason for this desire is because currently it takes about 2 minutes to fully run the 4 stages (Synthesis, Implementation, Generate Program File, Configure Target Device). Even 1 small change to the 65Org16.b software portion and I have to re-run all 4 stages. EDIT: BTW 2 minutes is a highly variable depending on the machine. For reference I use an Intel i7 875K @3.8GHz, 4GB RAM and a 64GB SSD running WinXP SP3. The change to a SSD made a large difference...

My experience with ISE14.1 is limited. The only tool(s) I have used is primarily Xilinx ISE iMPACT to generate the FPGA PROM image. I have also used the SmartXplorer tool to find ways to keep project speeds maintained @100MHz in the past. Floorplanner is good when learning the syntax of how to develop a proper text .ucf constraints file.

In my project I use 2 1Kx16 BlockRAMs for zero-page and stack-page, and a 4Kx16 for the ROM (pre-initialized BRAM).

Now with Michael's and enso's help I guess the first step would be to open up the FPGA Editor tool and find the X,Y location of the BRAM used for the ROM? Then insert some code into the ROM module?

Top profile quote

ElEctric_EyE

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 3:41 pm

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

> **MichaelM wrote:**
>
> ...
> Summary:
>
> (1) Determine the netlist names of the block RAMs to be patched.
> (2) Create a BMM file and attach it to the project file list. (Loc/Patch constraints optional at this time)
> (3) Set Translate's "Other Ngdbuild Command Line Options" to refer to the BMM file: -bm myBMM<.bmm>
> (4) Run Synthesis, MAP and PAR, and BitGen. Examine BitGen's myBMM_db.bmm file, and constrain the block RAMs define in myBMM.bmm to match.
> (5) Create MEM file to patch into bitstream file.
> (6) Set BitGen's "Other Bitgen Command Line Options" to enable real-time patching: -bd myMEM.mem
> (7) Re-run MAP, PAR, and BitGen, and download patched bitstream file to target. (At this point it's possible to create a PROM file for the FPGA.)..

I zoomed in and I see that the 4Kx16 'ROM' uses 4xRAMB16BWERs, so 4x (1Kx16), and I see their sites, so step 1 is complete.
I'll read Michael's post above on how to make a .bmm file.
EDIT:I'll go with this, step 2 complete:

Code:

```
////////////////////////////////////////////////////////////////////////////////
//
//  Block RAM Memory Map file: 65Org16.b Block RAM (4096 x 16) Program Memory
//
////////////////////////////////////////////////////////////////////////////////

ADDRESS_SPACE SYSROM RAMB16 [0x00000000:0x00000FFF]

    BUS_BLOCK
        Mram_RAM1 [15:0] LOC = X0Y20;
```

```
        Mram_RAM2 [15:0] LOC = X0Y16;
        Mram_RAM3 [15:0] LOC = X0Y18;
        Mram_RAM4 [15:0] LOC = X0Y24;
    END_BUS_BLOCK;

END_ADDRESS_SPACE;
```

EDIT: So I am realizing something here. Now I have specified the location of the BlockRAM. Initially I had left the tools to pick where to put it. Since my design is not complete, if I change the Verilog, the tools will try to relocate the BlockRAM which I do not want... I have seen enso's code in **another thread** on how 'lock-in' the location within the Verilog Module for the 'ROM'.
So I gather I would use something like:

Code:

```
(*LOC="RAMB16_X0Y0"*)
```

inside the 'ROM' Module. The problem for me now is how to spec the 4 BRAM locations inside this Module.

EDIT: Changed syntax error in .bmm file above.

Top profile quote

ElEctric_EyE

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 5:09 pm

Currently ngbuild is crashing, but I've noticed that the command line entry "-bm myBMM<.bmm>" in Translate seems to be uneccessary. ISE 14.1 seems to run it automatically if a .bmm file has been added as a source.

Top profile quote

ElEctric_EyE

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 5:50 pm

I see why it is crashing now. I did not understand how the blockRAM was organized.
Now I see how it is organized, but I am confused as to why ngdbuild is completing with an error message:
"ERROR::29 - Inconsistent address space size in ADDRESS_SPACE 'ROM'.
ADDRESS_SPACE was defined as 0x00001000 bytes, but the ADDRESS_RANGE total is 0x00002000 bytes."

Here is my .bmm file:

Code:

```
////////////////////////////////////////////////////////////////////////////////
//
//  Block RAM Memory Map file: 65Org16.b Block RAM (4096 x 16) Program Memory
//
////////////////////////////////////////////////////////////////////////////////

ADDRESS_SPACE ROM RAMB16 [0x00000000:0x00000FFF]

    BUS_BLOCK
        Mram_RAM1 [3:0] LOC = X0Y20;
        Mram_RAM2 [7:4] LOC = X0Y22;
        Mram_RAM3 [11:8] LOC = X0Y30;
```

```
        Mram_RAM4 [15:12] LOC = X0Y28;
    END_BUS_BLOCK;

END_ADDRESS_SPACE;
```

EDIT: Edited out my oversight that 4Kx16 needs 4x(4Kx4) BRAMs.

Top profile quote

ElEctric_EyE

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 10:57 pm

> **MichaelM wrote:**
>
> ...Following the memory type declaration you have to define the address range (in bytes) represented by your block RAM memory. In my case, the three block RAMs actually provide 6kB of storage. I initially made an error in specifying my memory array as having an address range [0x0000_0000:0x0000_0FFF], or 4096 12-bit words. When Translate read my BMM file, it reported that it expected an address range of 0x1800 (6kB) instead of the 0x1000 (4kW) I had specified. Thus, I had to set the address range of the PROM address space to [0x0000_0000:0x0000_17FF]...

Ahhh. Exactly the mistake I'm making! Great tutorial so far!
I have an odd multiple errors about memory leakage now. I will try to figure it out.

Another point I had overlooked:

> **ElEctric_EyE wrote:**
>
> ...EDIT: So I am realizing something here. Now I have specified the location of the BlockRAM. Initially I had left the tools to pick where to put it. Since my design is not complete, if I change the Verilog, the tools will try to relocate the BlockRAM which I do not want...

> **MichaelM wrote:**
>
> ...Second, rather than using the floorplanner to manually assign the location of the three block RAMs, I simply ran the synthesizer and PAR tools (without the BMM included/added to the project) and then looked in the routed FPGA (with the FPGA Editor) for the three block RAMs that I know would be used for implementing my program memory array. I then took the assigned names and pasted them into the BUS_BLOCK section of my BMM file, assigned the three 4-bit bit lanes, and reran the tools with the updated BMM added to the project to generate the my_BMM_bd.bmm file that BitGen outputs when my BMM is provided without LOC constraints for the block RAMs...

Meaning the .bmm file will tell the tools where to put this BRAM.

Top profile quote

enso

offline

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 11:39 pm

If you are instantiating the BRAMs (my preferred method), you can loc them easily to a known position like this:

Joined: Sat Sep 29, 2012 10:15 pm
Posts: 660

Code:

```
(*LOC="RAMB16_X0Y0"*) RAMB16_S9 #(  .INIT(9'h000),  .SRVAL(9'h000))
 ROM1( .CLK(CLK), .ADDR(A[10:0]), .DO(DO[7:0]), .DI(DI[7:0]), ...
```

Once that's done, you can hardware the bmm file and not worry about it again.

Top | profile | quote

ElEctric_EyE

offline

Joined: Mon Mar 02, 2009 7:27 pm
Posts: 3049
Location: NC, USA

Post subject: Re: Using Xilinx Data2MEM to Patch Block RAMs **Posted:** Tue Dec 03, 2013 11:52 pm

Thanks enso, but I've read that inferring is best for letting the tools take the greatest advantage. I think at this level though, it probably won't matter.

BTW, here is the way I coded my ROM. I posted it somewhere else but should probably post here as well. I don't think this code is causing the errors I currently see. I will work on it tomorrow.

Code:

```
//4Kx16 ROM, i.e. initialized FPGA blockRAM
`timescale 1ns / 1ps

module ROM ( input clk,
             input rst,
             input [11:0] addr,
             output reg [15:0] dout
             );

reg [15:0] ROM [0:4095];                      //or reg [15:0] ROM [4095:0],
address alignment doesn't matter

initial $readmemh("C:/65016PVBRAM2/boot.coe",ROM,0,4095);

always @(posedge clk)
   if (rst)
      dout <= 0;
      else
         dout <= ROM[addr];

endmodule
```

Top | profile | quote

Display posts from previous: All posts ▼ Sort by Post time ▼ Ascending ▼ Go

newtopic | postreply **Page 1 of 3** [45 posts] **Go to page 1, 2, 3 Next**

Board index » 6502.org Users Forum » Programmable Logic All times are UTC

Who is online

Users browsing this forum: No registered users and 1 guest

You **cannot** post new topics in this forum
You **cannot** reply to topics in this forum
You **cannot** edit your posts in this forum
You **cannot** delete your posts in this forum
You **cannot** post attachments in this forum