Electgon

A Circuits' Particle

Xilinx BMM File

ELECTGON
www.electgon.com
contact@electgon.com

08.06.2019

BMM file is used by Xilinx tools to write binary data into specified RAM. This is used for instance in writing firmware and startup routines of an embedded system. It consists of description of the hardware structure of the target RAM as well as its available address space so that the binary data can be filled correctly in the right address.

In Xilinx, data2mem command line tool can be used to fill the instantiated RAMs. It needs bmm file that describes the layout of the RAM, an elf file that contains the binary data to be filled, and a bit file that contains final placed and routed design.

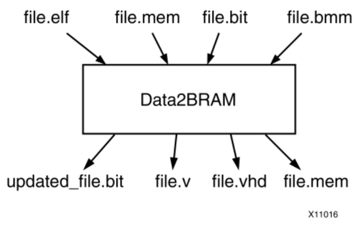

Figure 1: Data2mem Input and Output [1]

The general structure of a bmm file can be as shown below

```
ADDRESS_MAP map_name processor_type processor_ID
  ADDRESS_SPACE space_name mtype[start:end]

     BUS_BLOCK
       Bit_lane_definition
       Bit_lane_definition . .
     END_BUS_BLOCK;
.
.
  END_ADDRESS_MAP;
.
.
END_ADDRESS_MAP ;
```

- Running Processor maybe defined but it is not mandatory in this file.
- Space_name is a user defined name for the address space.
- Mtype is the type of the instance used to build the RAM. For Xilinx, the following ram types can be used : (RAMB16 - RAMB18 - RAMB32 - RAMB36 – MEMORY - COMBINED).
- [start:end] is the user defined address range
- Bus block refers to the data bus. If the processor is running on 32 bit data width, the bus block should contain clarification of where a 32-bit of the address space are defined. To make it clear, a 32 bit can be stored in one 32-bit width storage unit, or can be stored in 4 8-bit width storage unit, or can be stored in 2 16-bit width storage unit. This is what should be declared in a bus block, to describe how the hardware developer has designed storage unit(s) of the processor data bus. It is common to build the target RAM using 8-bit width storage unit.Figure 2 depicts more illustration.

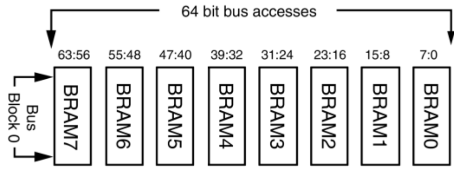

Figure 2: RAM Blocks [1]

So bit_lane_definition means which hardware instance is used to store one byte. In the example of figure 2, 8 RAM instances shall be defined for the block bus. The depth of each instantiated RAM unit reflects only how many bus blocks should be defined to cover the whole Address Space.

Lets assume we have Address Space 0x00000000:0x0003FFFF. This means the total size of the processor RAMs is 256Kx8 bit (K means 1024). Note here that one address is assigned for a 8 bit storage unit. Lets now assume the processor is running on 32 bit data bus and the building RAM unit is 2Kx8 bit (8 bit width, and 2048 bit depth). This means that one Bus Block will be formed by 4 RAM instances which will cover.

address space 0x00000000: 0x000007FF in BRAM0

address space 0x00000800: 0x00000FFF in BRAM1

address space 0x00001000: 0x000017FF in BRAM2

address space 0x00001800: 0x00001FFF in BRAM3

So at the end First Bus Block covers address space 0x00000000: 0x00001FFF

Second Bus Block covers address space 0x00002000: 0x00003FFF

Third Bus Block covers address space 0x00004000: 0x00005FFF

So one Bus Block will cover 8Kx8 bit (or simply 8K addresses). This means that 32 Bus Block are needed to cover the whole address space.

This Memory organization is the most important part for creating BMM file. Below is an example of how the Bus Block should look like

```
// Bus access map for next higher 16k, CPU address 0xFFFF8FFF-0xFFFFBFFF
BUS_BLOCK
  top/ram_cntlr/ram23 [63:56];
  top/ram_cntlr/ram22 [55:48];
  top/ram_cntlr/ram21 [47:40];
  top/ram_cntlr/ram20 [39:32];
  top/ram_cntlr/ram19 [31:24];
  top/ram_cntlr/ram18 [23:16];
  top/ram_cntlr/ram17 [15:8];
  top/ram_cntlr/ram16 [7:0];
END_BUS_BLOCK;

// Bus access map for next higher 16k, CPU address 0xFFFFC000 - 0xFFFFFFFF
BUS_BLOCK
  top/ram_cntlr/ram31 [63:56];
  top/ram_cntlr/ram30 [55:48];
  top/ram_cntlr/ram29 [47:40];
  top/ram_cntlr/ram28 [39:32];
  top/ram_cntlr/ram27 [31:24];
  top/ram_cntlr/ram26 [23:16];
  top/ram_cntlr/ram25 [15:8];
  top/ram_cntlr/ram24 [7:0];
END_BUS_BLOCK;
```

Figure 3: Bus Block Example [1]

Note that the instance name can be obtained from the synthesis report. For example the following was detected from the synthesis report

Found 2048x8-bit dual-port RAM <Mram_ram> for signal <ram>.

This means that The instance name is Mram_ram will be used to build the RAM unit. The [MSB_bit_num:LSB_bit_num] is used to indicate which byte lane the instance should store. i.e. for a 32 bus system, it will be distributed over 4 bytes lanes. This can be explained by the following example

ramg_g[0].ramB_g[0].ram_i/Mram_ram [7: 0] PLACED = X0Y0;
ramg_g[0].ramB_g[1].ram_i/Mram_ram [15: 8] PLACED = X0Y2;
ramg_g[0].ramB_g[2].ram_i/Mram_ram [23:16] PLACED = X0Y4;
ramg_g[0].ramB_g[3].ram_i/Mram_ram [31:24] PLACED = X0Y6;

So the instance
“ramg_g[0].ramB_g[0].ram_i/Mram_ram”
will store bits 7 downto 0, ramg_g[0].ramB_g[1].ram_i/Mram_ram will store bits 15 downto 8, etc.

It is possible also in the bmm file to specify which place the instantiated RAM should be place as shown in the previous example. The location is freely chosen by the developer according to the available tiles in the FPGA. One way to know that is by running synthesis and implementation (MAP, PAR) on the FPGA using Xilinx GUI tool (ISE, Vivado, PlanAhead, etc) then open the implemented design to see available RAM columns.

Bibliography

[1] Data2MEM User Guide - Xilinx UG658 - December, 2012.