ECE1756 Assignment 3
RAM Mapping and RAM Architecture

"Memory is the mother of all wisdom."

— Aeschylus

Assigned on Wednesday, November 9 **Total marks = 22/100**

Due on Tuesday, November 29 @ 11:59 pm

1 Objectives

This assignment is intended to help you:

1	Experience writing a flexible CAD tool that can target various FPGA architectures

- Map logical memories to physical block RAMs (BRAMs) in an FPGA architecture with up to 3 different memory block types (e.g. 3 sizes of BRAMs or 2 sizes of BRAMs and LUTRAM)
- Optimize the mapping to minimize FPGA area

2	Understand how to quantitatively architect an FPGA feature

- Compare the efficiency of various architectures, and suggest a good overall RAM architecture.

2 Background

Typically, hardware designers declare various memory arrays in their Verilog/VHDL implementations in different modules of their system. In a custom hardware (i.e. ASIC) design, this would instantiate a memory array of the exact required depth and word size. However, in FPGAs the types and sizes of the on-chip memory blocks are predetermined by the FPGA vendor. FPGA architectures can embed blocks of SRAM-based memory (BRAMs) of different sizes and configurations, and can also use the look-up tables in the soft fabric as memory blocks (LUTRAMs) as discussed in lecture. Forcing designers to determine the best way to combine BRAMs and LUTRAMs for each memory instance they need and writing HDL to implement them would be laborious and also would tie the design to a specific FPGA architecture. Instead, the vendor CAD tools typically include a *RAM mapping* stage that implements the *logical* memories in the user's design using the *physical* BRAMs and LUTRAMs on the chip in the best possible way. The RAM mapper chooses the physical memory type and configuration (i.e. width/number/type of its ports) and generates any additional logic required to combine multiple BRAMs or LUTRAMs to implement each logical RAM.

In this assignment, you will develop a RAM mapper CAD tool that takes as an input a list of logical RAMs in a benchmark circuit and the specifications of the FPGA's memory architecture, decides on how to map the logical RAMs to physical RAMs, and finally produces the total area of the FPGA that would be needed to implement this benchmark circuit. Ultimately, we would like to minimize this area as much as possible, which depends on both the specified FPGA memory architecture and the quality of the RAM mapper itself (i.e. how well can the RAM mapper utilize the available physical RAMs to implement a set of logical RAMs). Therefore, in the first half of the assignment, you will develop a RAM mapper CAD tool and optimize it to best utilize a fixed FPGA memory architecture for a given set of benchmark circuits. Then in the other half, you will use your optimized RAM mapper to perform an architecture exploration and propose an FPGA memory architecture that achieves the best results across the given benchmarks. To make things easier, we also provide you with a *legality checker* which takes as an input a mapping produced by your RAM mapper and tells you whether it is a legal mapping or not.

3 Deliverables

For this assignment, you are asked to hand in the following:

1. **The RAM mapper CAD tool and output:**
   - The source code for your CAD tool written in any programming language you prefer (as long as we can compile it and run it on the UG systems).
   - A README with instructions on how to build and run your code on the UG systems.
   - A RAM mapping file that was created by your tool and passes the legality checker for the fixed (i.e. Stratix-IV-like) architecture as described later in the handout.
2. **A LaTeX-typed report** in PDF format using this template on Overleaf. You can also use this service to write your report if you do not want to install Latex on your machine. Your report should include the following:
   (a) A description of the algorithm used by your tool.
   (b) A derivation of the computational complexity of the tool (i.e. the big O notation). You should explain how you got this answer.
   (c) A listing of the source code of your tool in the appendix of your report.
   (d) Results for the fixed Stratix-IV-like architecture:
      - This architecture has:
        – 50% of logic blocks can implement LUTRAM. When in LUTRAM mode, each logic block can implement sixty-four 10-bit wide or thirty-two 20-bit wide words.
        – For every 10 logic blocks, there is one 8192-bit RAM block that can implement RAMs ranging from 8192 1-bit words to (in all modes but true dual port) 256 32-bit words. In true dual port mode, the widest RAM organization is 512 words, each of which is 16 bits wide.
        – For every 300 logic blocks, there is one 128kbit RAM block that can implement RAMs ranging from 128k 1-bit words to 1024 128-bit words (in all modes but true dual port). In true dual port mode the widest RAM organization is 2048 words, each of which is 64 bits wide.
      - Fill in and include **Table 1** in your report, which gives the mapping results for each circuit. Note that the *"Required Logic Block Tiles in Chip"* and the *"Total FPGA Area"* are the values for an FPGA that gives just enough of the limiting resource type, and the amount specified by the various RAM block to logic block ratios of everything else. Note that the `checker` program described in Section 5 can generate this table for you from a RAM mapping output file.
      - Give the total **CPU runtime** of your program, and the **geometric average of the total FPGA area** required over the benchmark set for this architecture.

- Give the **command line** required to run your program for this architecture.
- Do not forget to submit the **RAM mapping output file** (that passes the legality checker program) for this architecture.

Table 1: RAM mapping results for example Stratix-IV-like architecture (LB: Logic Block).

Circuit #	**LUTRAM Blocks Used**	**8K BRAMs Used**	**128K BRAMs Used**	**Regular LBs Used**	**Required LB Tiles in Chip**	**Total FPGA Area**
Circuit 0						
Circuit 1						
...						

(e) Find the geometric average (over the benchmark suite given) of the FPGA area required for architectures with no LUTRAM, and one type of block RAM, as the size of the block RAM varies from 1k to 128k bits (in powers of 2). As you vary the block RAM size, (i) the maximum width and (ii) the number of logic blocks in the FPGA per RAM block should also vary. Experiment to find good values of these parameters for each RAM block size. Summarize the results in Table 2 as shown below, and explain the various trends you saw, and why.

Table 2: Results without LUTRAM.

BRAM Size	**Max Width**	**LBs / BRAM**	**Geometric Average FPGA Area (min width transistors)**
1 kbit	experiment to find a good value	experiment to find a good value	area
2 kbit	probably a different value	probably a different value	area
4 kbit			
...			

(f) Repeat the above experiment, fill in Table 3 and discuss the results, but this time with 50% of the logic blocks capable of being used as LUTRAM.

Table 3: Results with LUTRAM.

BRAM Size	**Max Width**	**LBs / BRAM**	**Geometric Average FPGA Area (min width transistors)**
1 kbit	experiment to find a good value	experiment to find a good value	area
...			
128 kbit	probably a different value	probably a different value	area

(g) Try to find a better organization of RAM blocks than the best ones you previously found in parts (e) and (f). You can use up to 3 types of RAM blocks (or 2 types of RAM blocks + LUTRAM), with any maximum bit count and maximum width you desire. You can also vary the percentage of logic blocks that support LUTRAM, if you choose to use LUTRAM in your architecture.

- Give the geometric average area required by your proposed RAM architecture over the benchmark set.
- Describe the parameters of your architecture. For each RAM block, give the maximum bit count, maximum width, and the number of logic blocks included in the FPGA for each RAM block of this type. Also indicate if you use LUTRAM or not, and if so, what fraction of the logic blocks support LUTRAM.
- Explain why your architecture is efficient.

4 Detailed Specifications

4.1 Benchmark Circuits

There are 69 benchmark circuits, numbered from 0 to 68. Your program will have to read in these benchmark circuits by parsing two files: `logic_block_count.txt` and `logical_rams.txt`.

The total number of logic blocks required for general logic (not RAM-related) in each circuit is given in the `logic_block_count.txt` file. There is a header line, and then each line gives the circuit number, followed by the number of logic blocks required by that circuit, e.g:

```
    Circuit     "# Logic blocks (N=10, k=6, fracturable)"
    0           2941
    1           2906
    2           1836
    ...
```

The `logical_rams.txt` file gives the data on all the "logical RAMs" in each design. A logical RAM is a RAM desired by the benchmark circuit, but not yet mapped to any "physical RAMs" on a specific FPGA. The first two lines give the number of circuits and column headings. Each line after that lists the circuit number, an integer to identify the logical RAM, the type of RAM desired, and its depth (number of words) and width (word size).

```
    Num_Circuits 69
    Circuit RamID   Mode            Depth   Width
    0       0       SimpleDualPort  45      12
    0       1       ROM             256     16
    0       2       SinglePort      2048    32
    1       0       TrueDualPort    32      36
...
```

The first RAM listed above, for example, is a RAM with one read port and one write port, and 540 bits organized as forty-five 12-bit words. There are four types of RAM:

- `ROM`: uses only one port and is never written to.
- `SinglePort`: uses only one port, as a r/w port.
- `SimpleDualPort`: uses one read port and one write port.
- `TrueDualPort`: uses two r/w ports to do 1 read and 1 write, 2 writes, or 2 reads each cycle.

4.2 Geometric Average

The geometric average of N numbers is computed by multiplying all N numbers together, and then taking the N^{th} root of the result. This average equally weights all benchmarks, regardless of size, so it is the best average to use for experiments like this [1]. You should take care that the product does not overflow; a good way to do this is to scale the numbers before computing their product (e.g. by diving by 10^7) and then scale the final geometric average back up (e.g. by multiplying by 10^7).

4.3 Area Model

The area you should assume for various blocks (in minimum width transistor areas) is as follows:

- Logic block area = 35000 (with no LUTRAM support)
- Logic block area = 40000 (if capable of supporting LUTRAM)
- Block RAM area = $9000 + 5 \times bits + 90 \times \sqrt{bits} + 600 \times 2 \times max\_width$

The logic block area is for ten fracturable 6-LUTs in a clustered architecture, similar to Stratix III/IV/V and fairly similar to Virtex7. Note that logic blocks that can support LUTRAM (whether or not they are used as LUTRAM) have **approximately 14% higher area** than logic blocks that do not support LUTRAM.

The RAM block model combines:

- A fixed area to do basic functions like select clocks, create timed signals like pre-charge and sense-amp activate, and routing area for control signals and address inputs. (9000)
- The area for the SRAM cells in the RAM array. Each cell takes 8 transistors (dual-port), but they are very efficiently laid out so we assume only 5 minimum-width transistor areas ($5 \times bits$).
- Area for the row decoders, word line drivers, column muxing and sense amplifiers. This area grows as the square root of the RAM. ($90 \times \sqrt{bits}$)
- Routing area for the RAM inputs that vary as the RAM maximum word width varies. We assume 600 minimum-width transistor areas for a RAM input or output. Each RAM block has two ports, each of which is max_width bits wide. ($600 \times 2 \times max\_width$).

Note that in computing total FPGA area, you must respect the ratio of the various physical blocks as specified by your architecture. For example, consider an FPGA architecture with one 18 kbit RAM for every 30 logic blocks and one 64 kbit RAM for every 100 logic blocks. If a benchmark that requires 400 logic blocks, twelve 18 kbit BRAMs, and six 64 kbit RAMs is mapped to this FPGA, the limiting resource would be the 64 kbit BRAMs. Therefore the total FPGA area should be calculated as follows:

- Having six 64 kbit requires $6 \times 100 = 600$ logic blocks
- Having six 64 kbit requires $600/30 =$ twenty 18 kbit BRAMs
- Total area = (6×Area of 64 kbit RAM)+(20×Area of 18 kbit RAM)+(600×Area of logic block)

If you would like to know more about how block RAMs are built in order to understand this area model, [2] details the internals of block RAMs.

4.4 Logical to Physical RAM Mapping Rules

Often a single logical RAM will require multiple physical RAMs to implement. Consider for example a logical RAM of type `SimpleDualPort` with depth = 32 words and width = 128 bits, as shown in Figure 1. When mapped to a physical RAM with 2048 bits that can provide word widths up to 32 bits wide, we would need 4 physical RAMs (arranged in parallel to each provide a 32-bit word), and would waste half the bits in the RAMs.

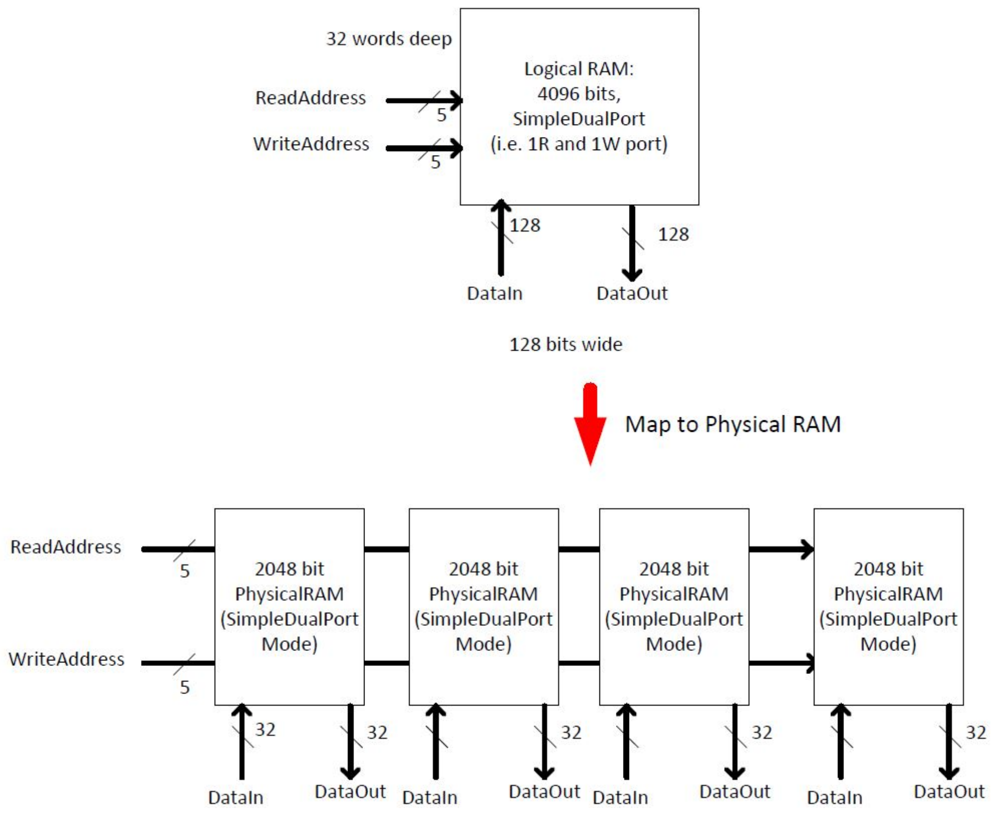

Figure 1: Mapping of a 2048 word × 32-bit wide Simple Dual Port RAM to Physical RAMs.

Sometimes we also have to combine multiple RAMs with some external logic to produce a deeper RAM. Consider the example of Figure 2 in which we are mapping a logical RAM of type `SimpleDualPort` with depth = 8192 words, and each word being only 1 bit wide. When mapped to a physical RAM with 2048 bits, we will require four physical RAMs to provide enough words. In addition, we will require some external circuitry: a 2:4 decoder to select the appropriate RAM to activate when writing (by decoding the top 2 bits of the address) and a 4:1 multiplexer to select the appropriate output data when reading. This will require five 6-LUTs (one to perform the 4:1 multiplexing, and four to provide the four 2-input and-gates needed in a 2:4 decoder), and hence requires 0.5 logic blocks (as we assume N = 10).

4.4.1 Physical RAM Capabilities

- A physical RAM can be organized to provide word widths of x1, x2, x4, x8, x16, ... up to the maximum width supported by the RAM. As the word width decreases, the depth of the RAM increases. For example, the 2048-bit RAM above supports 64 words when in x32 mode, but 2048 words when in x1 mode.
- Each physical RAM is dual-port. It can use its full width when in `SinglePort, ROM`, or `SimpleDualPort` modes. When in `TrueDualPort` mode, the widest width is not available, as both ports require both input (write) and output (read) ports to the routing – this doubles the demand for routing ports. The physical RAM listed above can only support x16 width words when in `TrueDualPort` mode, but can support x32 for all other modes, for example.

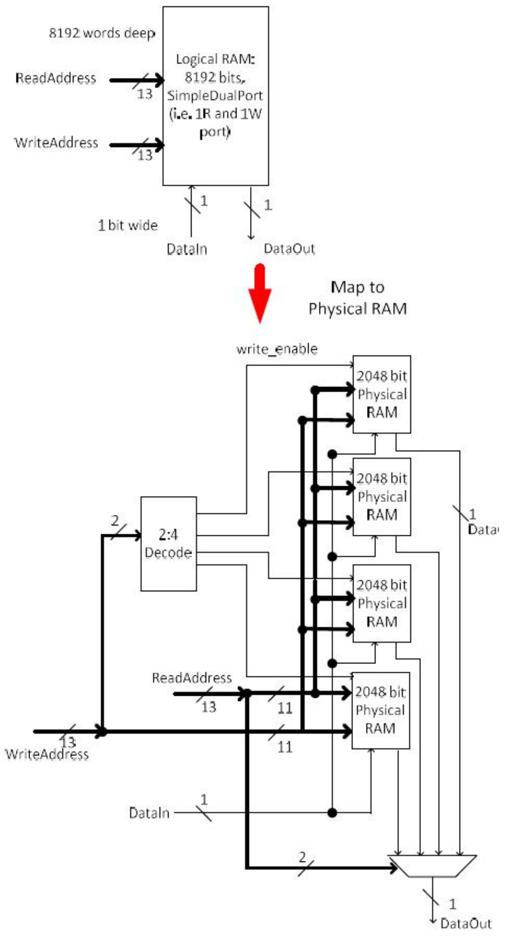

Figure 2: Mapping of a 8096 word × 1 bit wide logical RAM to physical RAMs.

- LUTRAM for this assignment provides only two modes: (i) 64-word, 10-bit wide, or (ii) 32-word, 20-bit wide. When used as LUTRAM, an entire logic block is converted and unavailable for logic use, even if the RAM implemented uses less than the full LUTRAM (e.g. a logical RAM of 64 five-bit wide words still uses an entire logic block). LUTRAM can implement ROM, single port, or simple dual port (1r and 1w) RAMs. It cannot implement true-dual port RAMs.

4.4.2 Physical RAM Legality

- The required bits must be less than or equal to the physical RAM capacity.
- The word width must be less than or equal to the width provided by the physical RAM. The width of the RAM is a function of the required word count and the RAM mode, as listed above.
- The physical RAM can provide only two ports.
- LUTRAM cannot implement true dual port mode.

4.4.3 Combining Multiple Physical RAMs to Implement a Logical RAM

As discussed above, very frequently multiple physical RAMs will have to be combined to implement a single logical RAM. Physical RAMs can be combined using the following rules:

- When physical RAMs are combined *in parallel* to create a wider word, no extra logic is needed.
- When physical RAMs are combined to implement a deeper RAM, extra logic is needed. If we combine R physical RAMs to make a logical RAM that is R times deeper than the maximum depth of the physical RAM, we will need to include extra logic to:
  1. Decode which RAM we must activate on a write. This requires a single $\log_2(R) : R$ decoder. This can be built using R LUTs, each of which must have $\log_2(R)$ inputs. Note also that for the special case of a 1:2 decoder, one of the two (1-input) LUTs is a buffer, so you really only need 1 (not 2) LUTs for that case.
  2. Multiplexers to select the appropriate read data word from the various physical RAM blocks. A total of W multiplexers of size $R : 1$ will be required, where W is the width of the logical RAM. A single 4:1 multiplexer can be implemented in a 6-LUT (and uses all 6 inputs). Larger multiplexers can be built by cascading 4:1 multiplexers together in a tree.

Keep in mind that the extra circuitry described above slows down the RAM. Assume that you can tolerate some slowdown, so combining physical RAMs to make an up to 16x deeper RAM (which will require 16:1 multiplexers and a 4:16 decoder) is permissible. You should not consider solutions that require larger than 16:1 multiplexers. The extra logic added should be included in your area results. Count the total number of LUTs you have added to the design, and divide by 10 to determine the number of extra logic blocks required.

4.4.4 Example for How to Fill Table 1

You are given a circuit that uses 20 logic blocks and has two logical RAMs:

- RAM 1: 512 bit depth, 32 bits wide, simple dual port
- RAM 2: 128 bit depth, 32 bits wide, simple dual port

We choose to implement the following mapping:

- RAM 1: use 8 kbit BRAMs. We will need two 8 kbit BRAMs to get enough RAM bits (each will be in 512 x 16 mode).
- RAM 2: use LUTRAM, plus some external muxing. We can make 64 deep x 10 bit wide RAMs from LUTRAM, so we need to combine two LUTRAMs (plus external muxing) to make a 128 x 10 bit wide RAM. Then we need to replicate that four times to get 32 bits of width (we actually get 40 bits, but the extra is wasted). So that is 8 LUTRAM blocks, plus we also need thirty-two 2:1 muxes (1 LUT each) and a 2:1 decoder (can be done in 1 LUT as well). So that adds up to a total of 33 LUTs for the extra circuitry.

In total across all the logic RAMs, we therefore need 33 LUTs in our mapping, or 3.3 logic blocks. We round this up to 4 (i.e. we are assuming the logic blocks already used by the circuit have no empty LUTs). This leads to us filling in Table 1 as shown below. The information for Table 1 says that we are targeting an FPGA with one 8 kbit BRAM for every 10 logic blocks and we also have 50% of my total logic blocks able to implement LUTRAM. So:

- 2 8 kbit BRAMs $\rightarrow$ need an FPGA with 20 logic blocks to get enough RAM blocks.

- 8 LUTRAM blocks → need an FPGA with 16 logic blocks to get enough RAM blocks.
- 8 + 24 = 32 total logic blocks → need an FPGA with at least 32 logic blocks. This is the limiting factor.

Circuit #	**LUTRAM Blocks Used**	**8K BRAMs Used**	**128K BRAMs Used**	**Regular LBs Used**	**Required LB Tiles in Chip**	**Total FPGA Area**
Circuit 0	8	2	0	20+4=24	32	1.49E+06

So now we need to compute the area of an FPGA with 32 logic blocks, and $32 \times 0.1 = 3$ (rounding off) 8 kbit RAM blocks (M8K). We also have one 128 kbit RAM block for every 300 logic blocks. Therefore, $32/300 = 0$ (rounding off) 128 kbit RAM blocks (M128K).

The total area can be calculated as follows:

- Area of 32 logic blocks, half of which support LUTRAM: $32 \times (35000 + 40000)/2 = 1,200,000$
- Area of 3 M8K blocks $= 3 \times [9000 + 5 \times 8192 + 90\sqrt{8192} + 600 \times 2 \times 32] = 289,518$
- Area of M128K blocks $= 0$
- Total area $= 1,489,518$

Note that this is not the best mapping; choosing to map both RAM 1 and RAM 2 to M8K RAMs would have let us fit this in an FPGA with three M8K RAMs, and the area of such an FPGA would be smaller. Such an FPGA with three M8K RAMs would necessitate $3 \times 10 = 30$ logic block tiles (and the three M8K RAMs that will come with them), which is still less than the 32 logic block tiles and three M8K RAMs we included for the mapping above.

5 Running the Legality Checker

We have prepared a checker to verify the correctness of your mapper; it is included in the `lab3.zip` archive on Quercus as `checker.exe` (for MS windows) or `checker` (for Ubuntu Linux) or `checker_mac` (for MacOS). The checker ensures that your mapping has accounted for all the logical RAMs and you have not violated the rules described in the assignment. Also, it checks the number of LUTs needed for muxing and decoders that you have calculated is acceptable (i.e. at least equal to the minimum required given your RAM mapping), and finally it computes the area of your solution. This document explains the format of the RAM mapping that your program should generate for the checker. Part of your submission of this assignment must include an input file for this legality checker (for the fixed Stratix-IV-like architecture).

5.1 How to Use the Checker

To use the checker for the RAM architecture described in the assignment, just type in the following line in the directory where the checker executable is located:

```
./checker -l 1 1 -b 8192 32 10 1 -b 131072 128 300 1 logical_rams.txt logic_block_count.txt <your
    mapping file>
```

or simply:

```
./checker -d logical_rams.txt logic_block_count.txt <your mapping file>
```

The `-d` flag automatically sets the RAM architecture parameters to the default one described in the assignment, while the first command does the same thing manually. The `-l` (a lowercase L) flag defines Type 1 RAM to be of LUTRAM and has a ratio of 1 to 1 with respect to the number of regular logic blocks. Likewise, the `-b` flags define block RAMs. The values after the `-b` flag are size, max width, and the ratio of logic blocks to block RAM respectively. RAM Type is the order that you input the flags into the checker. In this example, Type 1 RAM is the LUTRAM; Type 2 RAM is the 8 kbit RAM and Type 3 RAM is the 128 kbit RAM. The RAM Type number will be used in the mapping file. The checker supports RAM architectures with up to 3 different types of RAM. You can use the `-t` flag to print out a table like the one requested in the assignment. There also is a `-h` flag that explains how to use the checker. The checker will keep going until it reaches a syntax error or it finished processing all the test circuits. Error messages are printed to the standard error stream and it will indicate a pass or fail for each circuit it processed.

5.2 Mapping File Format - Basic

The basic mapping file format that you can input to the legality checker allows you to map each logical RAM to a *single type* of physical RAM (e.g. you might map a logical RAM to a set of 8 kb BRAMs, or a set of 128 kb BRAMs, but not a mix). This format is sufficient for most mapping algorithms.

Open `logical_rams.txt` and you will see many lines like the following:

```
1 2 SimpleDualPort 45 12
```

Each line is a logical RAM to be implemented. It lists the circuit number, RAM ID, the mode of the RAM, depth and width respectively. The mapping file must contain a mapping for every logical RAM in the `logical_rams.txt` file. For this example, a simple mapping would map it to two LUTRAMs in a 10-bit-wide-64-words-deep configuration in parallel. The mapping file format for such mapping, assuming the default Stratix-IV-like architecture, is:

```
1 2 0 LW 12 LD 45 ID 0 S 1 P 2 Type 1 Mode SimpleDualPort W 10 D 64
```

The first two numbers are the `Circuit` and `RamID` respectively. The third number is the number of additional LUTs needed, calculated according to the rules described in the assignment. `LW` and `LD` are logical width and depth. They should match the logical RAM's width and depth defined in `logical_rams.txt`. `ID` is a number you assigned to this group of physical RAM. This is usually just a unique id; if two logical RAMs use the same ID the mapper assumes they are sharing physical RAMs (which is legal in a few cases) as described in Section 5.3.2. `S` and `P` are the number of RAMs in series and in parallel respectively. Since we are putting two LUTRAMs in parallel, `P` is 2. `S` is 1 because there is only 1 LUTRAM in series. You can think of the values `S` and `P` as tiling `S`×`P` RAMs together to form one logical RAM. `Type` is the RAM type number and it matches the order of BRAM types you entered to the `checker` command. Using the example given in the previous section, LUTRAM is Type 1; 8 kbit block RAM is Type 2 and 128 kbit block RAM is Type 3. `Mode` is the mode that the physical RAM is in. It should be the same as the logical RAM's mode. `W` and `D` are the width and depth configuration that the physical RAM is in. Since we are using the 10-bit-wide-64-words-deep configuration, `W` and `D` are 10 and 64 respectively.

You can insert comments using "`//`" to help with debugging. The checker reads in a token at a time and skips all leading white spaces. Hence it can accept inputs separated into different lines if you find it easier to read. However, you should maintain the order of tokens as specified above.

5.3 Mapping File Format - Advanced *(Optional)*

If you would like to use multiple types of BRAM to map a single logical RAM, you will need to use this more complex, advanced mapping format. *Using this format is significantly more complex, and it is definitely not required – most good solutions use only the basic format.*

In general, a mapping for a logical RAM has the following components:

```
<Circuit> <RAM ID> <Additional LUT used> <Logical RAM Mapping>
```

`<Logical RAM Mapping>` is where you define your solution for a logical RAM.

`<Logical RAM Mapping>` expands into one of two forms. The simpler form:

```
<Logical RAM Mapping>
    LW <logical width> LD <logical depth> <Physical RAM>
```

where `<Physical RAM>` expands to

```
<Physical RAM>
    ID <ID num> S <series> P <parallel> Type <RAM type> Mode <mode> W <width> D <depth>
```

which is what you have seen in the previous section. This simply means that the logical RAM maps directly to a group of physical RAM with the same configuration. Alternatively, `<Logical RAM Mapping>` can be split into two smaller `<Logical RAM Mapping>` sections in series or in parallel. For example:

```
<Logical RAM Mapping>
    LW <width> LD <depth> [series|parallel] <Logical RAM Mapping> <Logical RAM Mapping>
```

For example, if we have to implement the following logical RAM:

```
Circuit RamID   Mode        Depth   Width
3       7       SinglePort  1025    30
```

We could simply map this logical RAM into one 128 kbit RAM, but it feels such a waste to use only one quarter of a 128 kbit RAM. So let's just use four 8k RAM in 8-bit-wide-1kwords-deep configuration and then top it off with two LUTRAMs. The mapping for that would be:

```
// Split into 2 smaller logical RAM mapping
3 7 31 LW 30 LD 1025 series
    // Four 8k RAM in Parallel
    LW 30 LD 1024 ID 0 S 1 P 4 Type 2 Mode SinglePort W 8 D 1024
    // Two LUTRAM in Parallel
    LW 30 LD 1 ID 1 S 1 P 2 Type 1 Mode SinglePort W 20 D 32
```

Figure 3 shows the arrangement of RAM described in the mapping above. Note that the combined `LD` of the two smaller mapping (1024 and 1) is the same as the large one (1025).

5.3.1 Advanced Mapping File Format - Another Example

Since you can recursively split `<Logical RAM Mapping>` into two smaller ones, you can define almost any configuration possible. Reusing the previous example, but this time, the logical RAM is a lot deeper:

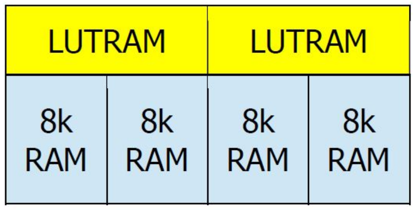

Figure 3: Combined RAM configuration.

```
Circuit RamID   Mode        Depth   Width
3       8       SinglePort  8200    30
```

Now it is too big for one 128 kbit RAM but using two would be a waste. Instead, we can use one 128 kbit RAM in 16-bit-wide-8192-words deep configuration in parallel with fourteen 8 kbit RAM in 1-bit-wide-8192-words deep configuration. To compensate for the last 8 words, we top it off with two LUTRAMs in parallel. This configuration is shown in Figure 4.

In the above configuration, the logical RAM is split into three logical mappings. To do that, first we split the logical RAM mapping into the yellow region and the region enclosed by the red border and connect them in series. Then, we split the red border region further into the blue and green region. Thus, the mapping for this configuration should be:

```
3 8 32 LW 30 LD 8200 series // The whole RAM split into 2 regions in series
    // Yellow LUTRAM Region
    LW 30 LD 8 ID 0 S 1 P 4 Type 1 Mode SinglePort W 20 D 32
    // Red Border Region, which further split into 2 regions in parallel
    LW 30 LD 8192 parallel
        // Blue 128kbit RAM Region
        LW 16 LD 8192 ID 1 S 1 P 1 Type 3 Mode SinglePort W 16 D 8192
        // Green 8kbit RAM Region
        LW 14 LD 8192 ID 2 S 1 P 14 Type 2 Mode SinglePort W 1 D 8192
```

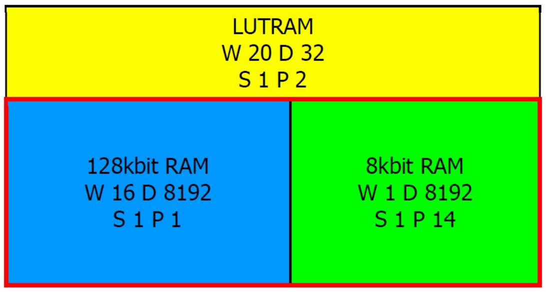

Figure 4: RAM Configuration for a 30-Bit-Wide-8300-Word-Deep Logical RAM.

5.3.2 Advanced Mapping File Format - Sharing Block RAM

There are times when you would like to share a physical RAM. Consider the following example:

```
Circuit RamID   Mode        Depth   Width
1       0       SinglePort  100     8
1       1       ROM         50      16
```

RAM 0 and RAM 1 in this example can share a 8 kbit block RAM in 16-bit-wide-512-word-deep configuration in TrueDualPort mode. The mapping for that would be:

```
// RAM 0
1 0 0 LW 8 LD 100 ID 7 S 1 P 1 Type 2 Mode TrueDualPort W 16 D 512
// RAM 1
1 1 0 LW 16 LD 50 ID 7 S 1 P 1 Type 2 Mode TrueDualPort W 16 D 512
```

You will notice that the two mappings above describe the same physical RAM. A 8 kbit RAM with ID number 7 in TrueDualPort mode. It is important that the blue portions are identical and the physical RAM is in TrueDualPort mode. This tells the checker that RAM 0 and RAM 1 reside in the same physical RAM and it should only count one 8 kbit RAM as used. Moreover, the aggregate depth of RAM 0 and 1 cannot be greater than the physical RAM's. Physical RAM's width must be equal or greater than RAM 0's width and RAM 1's width. Even though RAM 0 and 1 share the same physical RAM, they do not share whatever additional LUTs are needed. Note that due to port limitations, at most 2 logical RAMs can be mapped onto a physical RAM. Also, SimpleDualPort RAM and TrueDualPort RAM need two ports for a single logical RAM, and therefore they can't share a physical RAM with other logical RAM.

6 How You Will Be Graded

Your grade out of 22 will mainly depend on:

1. **The correctness and optimization quality of your RAM mapper: ~50%.** Part of this grade is for producing a legal result, but most of this grade will be for optimization quality on the Stratix IV-like architecture.

2. **The quality and completeness of your report and the quality of the RAM architecture you propose: ~45%.** You should answer all the questions and provide all the tables requested in the lab handout, and the architecture trends should be correct and explained well. You should thoroughly explore the RAM architecture space and suggest an improved FPGA RAM architecture.

3. **Coding style: ~5%.** Your code should be understandable and well structured. Use descriptive variable names and comment extensively. Break your code into moderate-length functions to avoid duplicated code and improve readability. Group related data into structures or classes.

7 References

[1] P. Fleming and J. Wallace, "How Not to Lie with Statistics: The Correct Method to Summarize Benchmark Results," Communications of the ACM, 1986. [paper link]

[2] S. Yazdanshenas, K. Tatsumura, and V. Betz, "Don't Forget the Memory: Automatic Block RAM Modelling, Optimization, and Architecture Exploration," FPGA 2017, pp. 115 - 124. [paper link]