ECE 1756

Assignment 3: RAM Mapping and RAM Architecture

"Memory is the mother of all wisdom."

- Aeschylus

Value: 21% of final mark

Purpose

This lab is intended to develop:

1. Experience writing a flexible CAD tool that can target various FPGA architectures and
2. Experience in quantitatively architecting an FPGA feature, in this case the RAM blocks.

Overview

1. You will develop a CAD tool that can read in the "logical RAMs" in each circuit of a benchmark set, and output a set of "physical RAMs" that can implement the logical RAMs for each circuit. Your tool should be able to target FPGAs with up to 3 types of physical RAMs (which may be 3 different block RAMs, or 2 block RAMs and "LUTRAM" created by using a logic block as RAM). Your CAD tool should attempt to find a solution that minimizes the area of the FPGA needed to fit each benchmark circuit.
2. You will use this CAD tool to compare the efficiency of various RAM architectures, and to suggest a good overall RAM architecture.

What to Hand In:

1. The RAM mapper CAD Tool and output:
    a. Submit the **source code** for your CAD tool on blackboard, via the assignment 3 tab.
    b. Include a **makefile** or equivalent (e.g. Visual Studio Project) that will build the code on the EECG Linux systems, or on some other system like MS Windows.
    c. Submit a **RAM mapping file** that was created by your program and which passes the checking program **for the Stratix IV-like RAM architecture** described in Section 2.d below.
2. Submit to blackboard a pdf document in which you:
    a. Describe the algorithm used by your tool.
    b. Derive the computational complexity of the tool, i.e. big O notation. Show your work.

c. Print out the source code and include it in your report.
d. This pdf file should be submitted outside a .zip or other archive, so turnitin can check it.
e. Your report should include results for a RAM architecture (similar to Stratix IV) that has:
    i. 50% of logic blocks can implement LUTRAM. When in LUTRAM mode, each logic block can implement sixty-four 10-bit wide words, or thirty-two 20-bit wide words.
    ii. 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, with each 16-bits wide.
    iii. 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 maximum word width is 64-bits (and there are 2048 such words in that mode).

- Fill in and include the table below 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.*

Table 1: RAM mapping results for example Stratix-IV like architecture.

Circuit #	LUTRAM blocks used	8192 bit RAM blocks used	128k bit RAM blocks used	Regular logic blocks used	Required Logic Block Tiles in Chip	Total FPGA Area (including unused blocks)
Circuit 0						
Circuit 1						
…						

- Give the total CPU time required by your program to map all the circuits, and the *geometric average* of the total FPGA area required over the benchmark set for this architecture.
- Your report must give the command line required to run your program for this architecture.
- Submit the RAM mapping output file (that passes the checking program) for this architecture.

f. 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 bits to 128k. 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 a table as shown below, and explain the various trends you saw, and why.

Table 2: Results without LUTRAM

Block RAM size	Max width	Logic blocks / RAM block	Geometric average FPGA area (min width trans.)
1 kbit	64	50	5.2×10^6
2 kbit	32	40	6.3×10^6
…			

g. Repeat the above experiment and include a similar table and discussion of results, but this time with 50% of the logic blocks capable of being used as LUTRAM.

Table 3: Results with LUTRAM

Block RAM size	Max width	Logic blocks / RAM block	Geometric average FPGA area (min width trans.)
1 kbit	128	30	8.9×10^6
…			
128 kbit	64	20	9.7×10^6

h. Try to find a better organization of RAM blocks than the best one you found in sections 2 and 3. 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 over the benchmark set by your RAM architecture.
- 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.

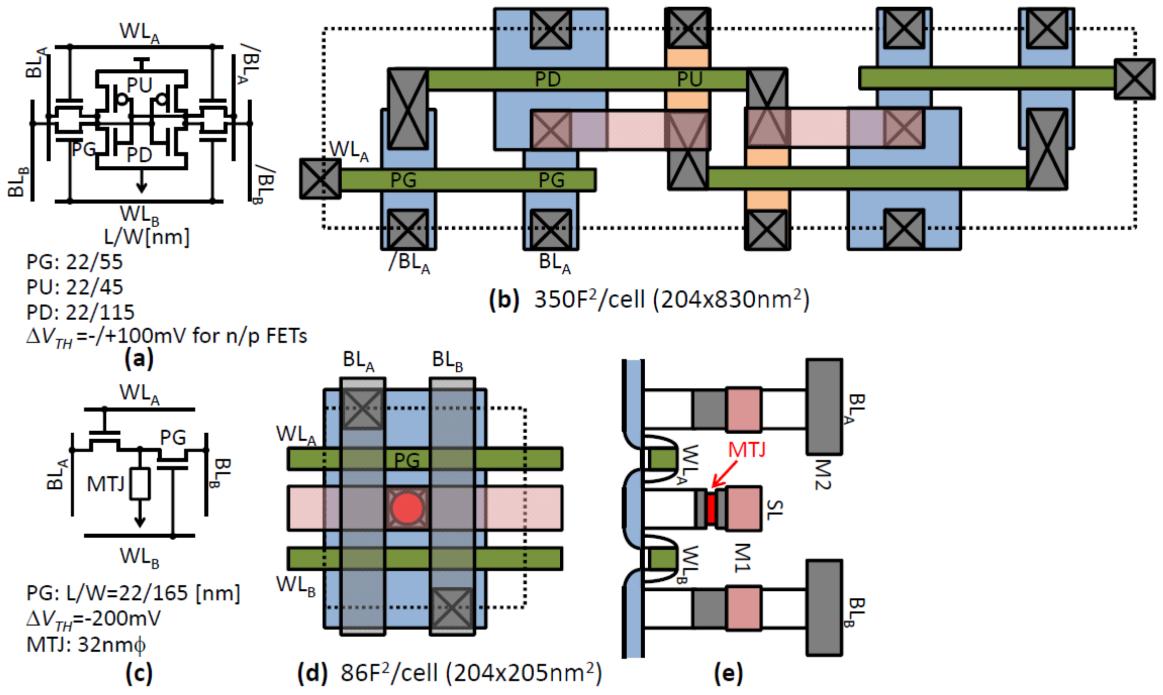


Figure 1: SRAM and MTJ cells: (a) Dual-port 8T SRAM cell and (b) its layout. (c) Dual-port 2T-1R MTJ cell, (d) its layout and (e) cross-section [2].

h. Future FPGAs may use emerging non-volatile memory technologies such as Magnetic Tunneling Junctions (MTJs) as a replacement for SRAM cells [2,3]. A MTJ is much more compact than SRAM and may offer a significant area advantage in large memory arrays. As shown in Figure 1, a single MTJ cell contains only two transistors instead of the 8 in a dual port SRAM cell, and is about 4 times smaller than its SRAM counterpart. Change your area model (as described in the Area Model section of the Detailed Specification that follows) to model an MTJ-based BRAM and repeat the experiment in part g.

- Give the geometric average area required over the benchmark set by your best (MTJ RAM) architecture.
- 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 how the best architecture changes when you change the memory technology. Why does the best architecture change / not change?

Background and Detailed Specification:

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)"
0            2941
1            2906
2            1836
...
```

The *logical_rams.txt* file give 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 fourty-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, allowing it to do 1 read and 1 write on a cycle, or 2 writes, or 2 reads.

2. Geometric Average

The geometric average of N numbers is computed by multiplying all N numbers together, and then taking the Nth 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 1e8) and then scale the final geometric average back up (e.g. by multiplying by 1e8).

3. Area Model

The area you should assume for various blocks (in minimum width transistor areas) is given below.

Logic block area = 35000 (with no LUTRAM support)

Logic block area = 40000 (if capable of supporting LUTRAM)

(SRAM-based) Block RAM area = 9000 + 5 * bits + 90 * sqrt(bits) + 600 * 2 * max_width

For **Part h** of the assignment use the following area model:

MTJ-based Block RAM area = 9000 + 1.25 * bits + 90 * sqrt(bits) + 600 * 2 * max_width

The logic block area is for a ten 6-LUTs in a cluster 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 SRAM cells takes 8 transistors (dual-port), but they are very efficiently laid out so we assume only 5 minimum-width transistor areas (5 * bits).
- Area for the row decoders, word line drivers, column muxing and sense ampliers. This area grows as the square root of the RAM. [90 * 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, and since there are two ports, one of which will need max_width inputs and the other which will need max_width outputs (in simple dual port mode), we multiply by 2: (600 * 2 * 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
    - One 64 kbit RAM for every 100 logic blocks
- Consider a benchmark circuit mapping solution that requires:
    - Twelve 18 kbit RAM blocks
    - Six 64 kbit RAM blocks

    - 400 logic blocks
- The limiting resource is the 64-kbit RAM blocks. We require an FPGA with:
    - Six 64 kbit RAM blocks
    - 6 * 100 = 600 logic blocks
    - 600/30 = twenty 18 kbit RAM blocks
    - Area = 6 * (area of 64-bit RAM) + 20 * (area of 18 kbit RAM) + 600 * logic block area

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 2. 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.

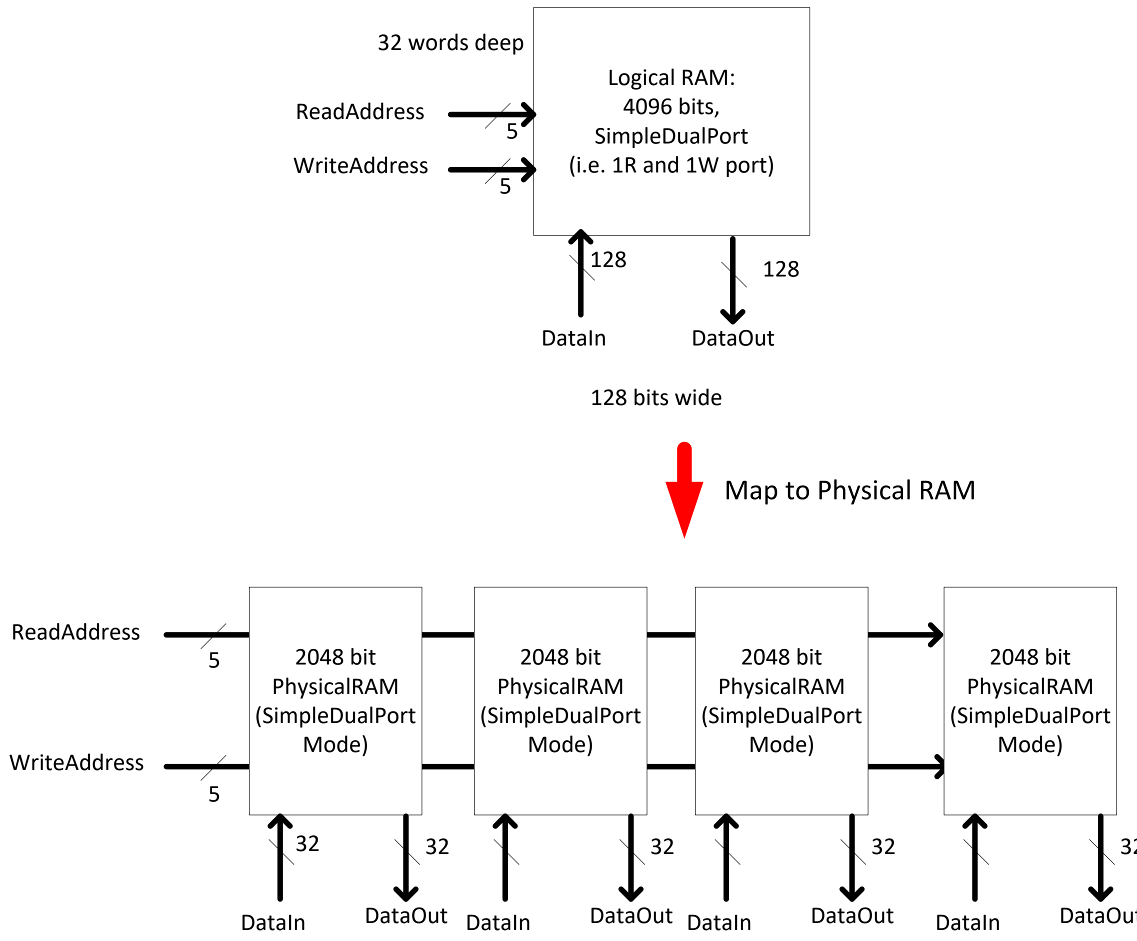


Figure 2: Mapping of a 2048 word x 32-bit wide Simple Dual Port RAM to Physical RAMs.

Sometimes we also have to combine multiple RAMs with some external logic to produce a deeper RAM. Consider the example of Figure 3 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).

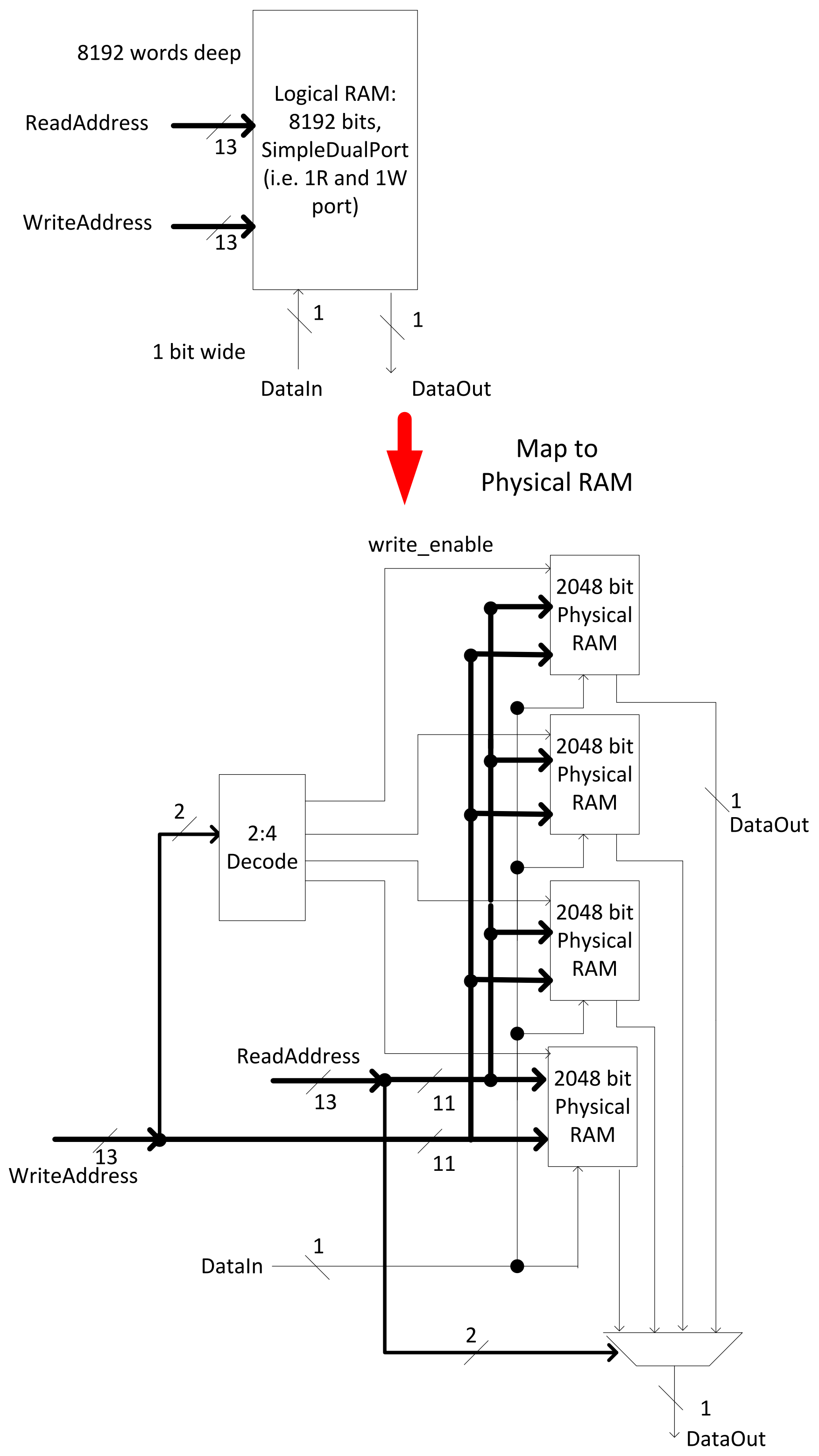


Figure 3: Mapping of a 8096 word x 1 bit wide logical RAM to physical RAMs.

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 number of words 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.
- LUTRAM for this assignment provides only two modes:
    - a 64-word, 10-bit wide RAM, or
    - a 32-word, 20-bit wide RAM.

  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.

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 provided by the RAM is a function of the word count required and the RAM mode, as listed above.
- The physical RAM can provide only two ports.
- LUTRAM cannot implement true dual port mode.

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.
- When multiple physical RAMs are combined "in parallel" to create a wider word, no extra logic is needed.
- When multiple physical RAMs are combined to produce 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 *width*$_{logicalRAM}$ *r:1* multiplexers will be required. 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.
3. The extra circuitry 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 muliplexers 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.

Example of How to Fill in Table 1:

You are given the following circuit:

2 logical RAMs:
RAM 1: 512 bit depth, 32 bits wide, simple dual port
RAM 2: 128 bit depth, 32 bits wide, simple dual port
Logic blocks (in the input files): 20

I choose a mapping of:

RAM 1: use 8192 bit block RAMs. I will need two 8192 bit block RAMs to get enough RAM bits (each will be in 512 x 16 mode).

RAM 2: use LUTRAM, plus some external muxing. I can make 64 deep x 10 bit wide RAMs from LUTRAM: I need to combine two LUTRAMs (plus external muxing) to make a 128 x 10 bit wide RAM. Then I need to replicate that four times to get 32 bits of width (I actually get 40 bits, but the extra is wasted). So that is 8 LUTRAM blocks, plus I need thirty-two 2:1 muxes (1 LUT each) and a 2:1 decoder (can be done in 1 LUT as well). So that is 33 LUTs --> 4 regular logic blocks (I round up, as I need more than 4 logic blocks).

This leads to me filling in Table 1 as shown below.

Circuit #	LUTRAM blocks used	8192 bit RAM blocks used	128k bit RAM blocks used	Regular logic blocks used	Required Logic Block Tiles in Chip	Total FPGA Area (including unused blocks)
Circuit 0	8	2	0	20 + 4 = 24	32	1.49e6

The information for Table 1 says I am targeting an FPGA with *one* 8k memory block for every *10* logic blocks. I also have 50% of my total logic blocks able to implement LUTRAM. So:

- 2 8k RAMs --> 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.

So now I need to compute the area of an FPGA with 32 logic blocks, and 32 * 0.1 = 3 (rounding off) 8k RAM blocks.
I also have one m128k RAM for every 300 logic blocks. 32 / 300 = 0 (rounding off) m128k RAM blocks.

Total area =
area of 32 logic blocks, half of which support LUTRAM: 32 * (35000 + 40000)/2 = 1 200 000
+ area of 3 m8K blocks = 3 * [9000 + 5 * (8192) + 90 * sqrt(8192) + 600 * 2 * 32] = 3 * 96506 = 289 518
+ area of 0 m128k blocks = 0

Total area = 1 489 518

Note that this is not the best mapping; choosing to map both RAM1 and RAM2 to 8192 RAMs would have let me fit this in an FPGA with three 8192 RAMs, and the area of such an FPGA would be smaller. Such an FPGA would require three 8192 RAMs, which necessitates 3 * 10 = 30 logic block tiles (and the three 8192 bit block RAMs that will come with such a chip) to get a big enough FPGA with the specified logic to RAM ratio, but this is still less than the 32 logic block tiles and three 8192 RAMs I had to include in my area above.

Running the RAM Legality Checker

We have prepared a checker to verify the correctness of your mapper; it is included on blackboard as `checker.exe` (MS windows) or `checker` (Ubuntu Linux) or `checker_mac` (MacOS). The checker ensures that your mapping has accounted for all the logical RAM and you have not violated the rules described in the assignment. Also, it checks the number of Look-Up Tables 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 Stratix IV-type architecture).

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. The first line 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 a Block RAM. 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 8kbit RAM and Type 3 RAM is the 128kbit 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 from the assignment, except for it does not calculate the area. 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 circuits. Error messages are printed to the standard error stream and it will indicate a pass or fail for each circuit it processed.

Mapping File Format

Open *logical_rams.txt* `and y`ou 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. There should be a mapping for every logical RAM in the logical ram file. Using this example, a simple mapping would map it to two LUTRAM in a 10-bit-wide-64-words-deep configuration in parallel. The mapping file format for such mapping, assuming default 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 number and `RAM ID` respectively. The third number is the number of additional LUT needed that you 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. It should be unique, otherwise the mapper assumes that you are using the same hardware for multiple logical RAMs. *S* and *P* are the number of RAM in series and in parallel respectively. Since we are putting two LUTRAM in parallel, *P* is 2. *S* is 1 because there's only 1 LUTRAM in series. You can think of the values *S* and *P* as tiling *S* x *P* RAM together to form a logical RAM. *Type* is the RAM Type number and it is the order which you have entered in the command. Using the example given in the previous section, LUTRAM is Type 1; 8kbit block RAM is Type 2 and 128kbit 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. *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 skip all leading white spaces. So 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.

This simple format should be sufficient for most mapping algorithms. If you would like to use different configurations to try to achieve even better results, please continue onto the next section (but be warned, it is more complex!).

<u>Mapping File Format – Advanced</u>

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, i.e.:

```
<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 #	RAM ID	Mode	Depth	Width
3	7	SinglePort	1025	30

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

```
// Split into 2 smaller logical RAM mapping
3 7 32 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 1:
RAM Configuration

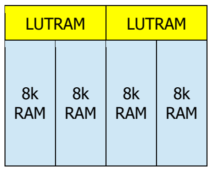


Figure 1 shows the arrangement of RAM described in the mapping above. Note that the combined *LD* of the two smaller mapping (1024 and 1) is the same as the large one (1025).

Mapping File Format – Another Advanced Example

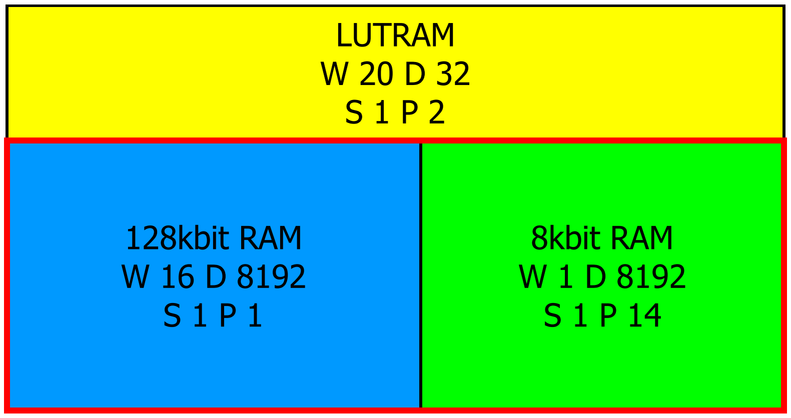
Since you can recursively split `<Logical RAM Mapping>` into two smaller ones, you can define almost any configuration possible.

Reusing the previous example, but this time, the logical RAM is a lot deeper:

Circuit #	RAM ID	Mode	Depth	Width
3	8	SinglePort	8200	30

Now it is too big for one 128kbit RAM but using two would be a waste. Instead, we can use one 128kbit RAM in 16-bit-wide-8192-words deep configuration in parallel with fourteen 8kbit RAM in 1-bit-wide-8192-words deep configuration. To compensate for the last 8 words, we top it off with two LUTRAM in parallel. This configuration is shown in Figure 2.

Figure 2: RAM Configuration for a 30-Bit-Wide-8300-Word-Deep Logical RAM



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
```

Mapping File Format – Sharing Block RAM

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

Circuit #	RAM ID	Mode	Depth	Width
1	0	SinglePort	100	8
1	1	ROM	50	16

RAM 0 and RAM 1 in this example can share a 8kbit 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
```

The highlighted portions describe the same physical RAM. A 8kbit RAM with ID number 7 in *TrueDualPort* mode. It is important that the highlighted 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 only count one 8kbit RAM 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 LUT needed. Note that due to port limitation, at most 2 logical RAM can be mapped onto a physical RAM. Also, *SimpleDualPort* RAM and *TrueDualPort* RAM need two ports already. Hence, they can't share a physical RAM with other logical RAM.

References

[1] P. Fleming and J. Wallace, "How Not to Lie with Statistics: The Correct Method to Summarize Benchmark Results," *Communications of the ACM*, March 1986. http://dl.acm.org/citation.cfm?id=5673

[2] K. Tatsumura, et al, "High Density, Low Energy, Magnetic Tunnel Junction Based Block RAMs for Memory-rich FPGAs" to appear in *International Conference on Field-Programmable Technology* 2016, pp. 1 – 8. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7929181

[3] Y. Guillemenet, et al. "A non-volatile run-time FPGA using thermally assisted switching MRAMS." *International Conference on Field Programmable Logic and Applications*. 2008.