Adam Pietrewicz
November 11, 2024

FPGA Project: RAM Mapping and RAM Architecture

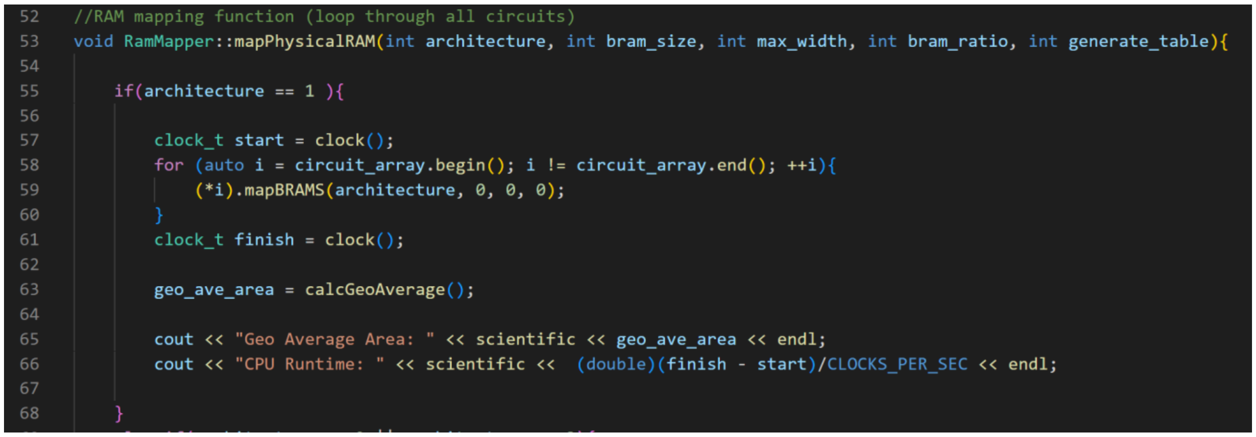
1. Description of the algorithm used in RAM Mapping tool

This RAM mapper uses a simple but effective algorithm that optimizes to minimize FPGA area. For every circuit, the program loops through all logical RAMs it needs mapped and does the following: it tries to find a physical RAM configuration (size, width, depth) that has the smallest local RAM area. So for every logical RAM, the smallest area solution is found and stored as the mapping. Thus, this splits the problem into optimizing each individual mapping to have the smallest area. For the Stratix-IV-like architecture, the program loops through all 3 possible physical RAM options: LUTRAM, 8k BRAM and 128k BRAM. It chooses a mapping out of the 3 options that takes the least area to map. When all logical RAMs are "mapped", they have stored the amount of LUTRAM, 8K , 128K BRAM, and additional LUT blocks (for extra circuitry when adding blocks together in series) needed to map the RAMs, and they are added up together to find the limiting factor and total FPGA area for a circuit. Lastly, all circuit areas are averaged together to produce a Geometric Average of the 69 benchmark circuits to gauge the success of the mapping.

Another algorithm was developed, however it was not able to pass all legality checks when testing different BRAM sizes in part 4 and 5. First, the list of logical RAMs in each circuit was sorted in order of largest to smallest size in order to first map the largest RAMs. Very large logical RAMs could usually be mapped to 128k BRAMs because they offer the most memory, and extra routing would be eliminated compared to if we tried mapping to 8k BRAMS or 640 bit LUTRAMS. Similarly, this algorithm then tries every possible configuration of physical RAMs for ever logical RAM (LUTRAM, 8k BRAM, 128k BRAM) but calculates the current total FPGA area rather than a "local BRAM" area. In other words, it chose a mapping by calculating the total area of the FPGA needed to map all logical RAMs so far iterated through. This did produce a mapping with good area results but was not developed further as it was not passing legality tests later on for part 4 and 5.

2. Derivation of the computational complexity of the tool

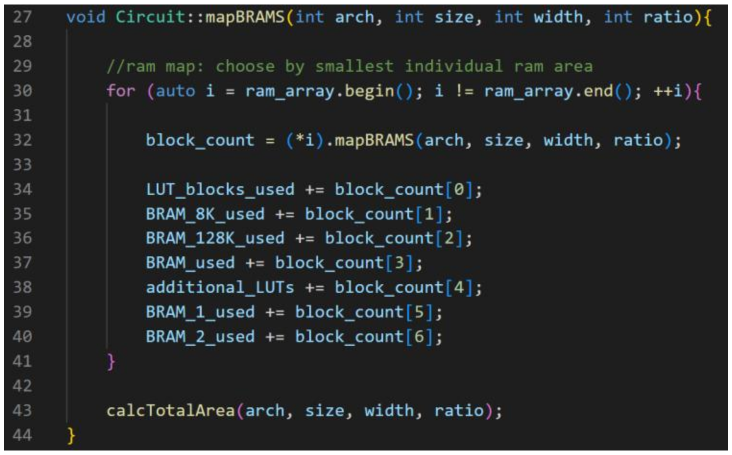
Let N represent the number of circuits and M represent the number of logical RAMs they may contain. First, the mapping tool (*RamMapper* class) loops through all circuits and calls a function (*Circuit::mapBRAMS*) that maps each circuit to the FPGA. The *RamMapper* class contains all circuits as a vector: *vector<Circuit> circuit_array*. This takes **O(N).**



```
//RAM mapping function (loop through all circuits)
void RamMapper::mapPhysicalRAM(int architecture, int bram_size, int max_width, int bram_ratio, int generate_table){

    if(architecture == 1 ){

        clock_t start = clock();
        for (auto i = circuit_array.begin(); i != circuit_array.end(); ++i){
            (*i).mapBRAMS(architecture, 0, 0, 0);
        }
        clock_t finish = clock();

        geo_ave_area = calcGeoAverage();

        cout << "Geo Average Area: " << scientific << geo_ave_area << endl;
        cout << "CPU Runtime: " << scientific <<  (double)(finish - start)/CLOCKS_PER_SEC << endl;

    }
```

Figure 1: Function that loops through all circuits and maps them to physical RAM.

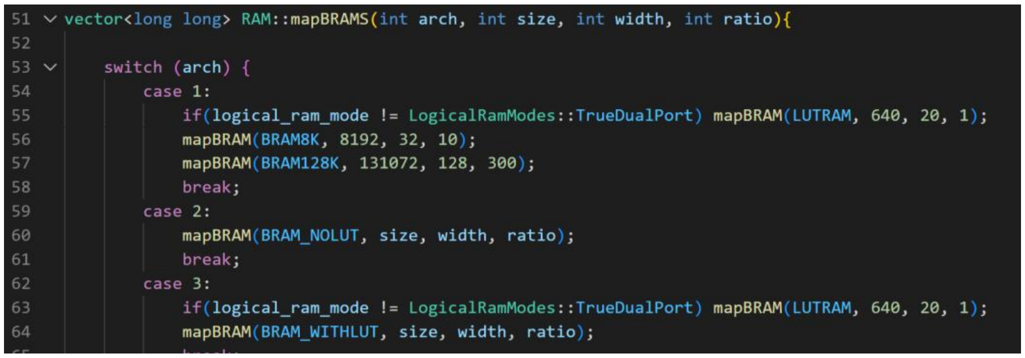
Next, each circuit object iterates through their vector of logical RAMs: *vector<RAM> ram_array* and calls the *RAM* class function *RAM::mapBRAMS* M times. Here in this loop, the number of physical RAM blocks are tracked and accumulated, and finally the function *calcTotalArea()* looks at all the accumulated physical RAM blocks and calculated the total FPGA area. Thus, this loop is **O(M),** making the complexity O(NM) so far.



```
void Circuit::mapBRAMS(int arch, int size, int width, int ratio){

    //ram map: choose by smallest individual ram area
    for (auto i = ram_array.begin(); i != ram_array.end(); ++i){

        block_count = (*i).mapBRAMS(arch, size, width, ratio);

        LUT_blocks_used += block_count[0];
        BRAM_8K_used += block_count[1];
        BRAM_128K_used += block_count[2];
        BRAM_used += block_count[3];
        additional_LUTs += block_count[4];
        BRAM_1_used += block_count[5];
        BRAM_2_used += block_count[6];
    }

    calcTotalArea(arch, size, width, ratio);
}
```

Figure 2: Circuit function that loops through its vector of logical RAMs and maps them.

Finally, the *RAM* class function *RAM::mapBRAMS* goes through multiple function calls which attempt to map using different physical RAM types (using *RAM:: mapBRAM*). Here the time complexity can be classified as constant because the number of iterations are constant (number of physical RAM types: 3, number of possible widths/depths for each RMA type: constant (powers of 2)). Therefore, the total computational complexity of the tool can be expressed as: **O(NM).**



```
vector<long long> RAM::mapBRAMS(int arch, int size, int width, int ratio){

    switch (arch) {
        case 1:
            if(logical_ram_mode != LogicalRamModes::TrueDualPort) mapBRAM(LUTRAM, 640, 20, 1);
            mapBRAM(BRAM8K, 8192, 32, 10);
            mapBRAM(BRAM128K, 131072, 128, 300);
            break;
        case 2:
            mapBRAM(BRAM_NOLUT, size, width, ratio);
            break;
        case 3:
            if(logical_ram_mode != LogicalRamModes::TrueDualPort) mapBRAM(LUTRAM, 640, 20, 1);
            mapBRAM(BRAM_WITHLUT, size, width, ratio);
```

Figure 3: RAM physical mapping function. Each architecture tries mapping to different physical RAM types.

3. Results for the fixed Stratix-IV-like architecture

Circuit #	**LUTRAM Blocks Used**	**8K BRAMs Used**	**128K BRAMs Used**	**Regular LBs Used**	**Required LB Tiles in Chip**	**Total FPGA Area**
0	1118	221	0	2941	4059	2.02373E+08
1	664	228	40	2906	12000	5.99885E+08
2	93	0	0	1836	1929	9.59789E+07
3	53	45	1	2808	2861	1.42556E+08
4	495	642	20	7907	8402	4.19995E+08
5	31	288	0	3692	3723	1.85737E+08
6	76	160	0	1853	1929	9.59789E+07
7	263	361	18	3947	5400	2.69948E+08
8	134	576	0	5342	5760	2.87775E+08
9	1	32	0	1636	1637	8.13783E+07
10	378	49	16	1418	4800	2.39954E+08
11	233	57	1	1329	1562	7.78900E+07
12	11	4	2	1632	1643	8.16998E+07
13	6	20	0	4491	4497	2.23897E+08
14	53	62	22	1808	6600	3.29937E+08
15	43	63	4	1956	1999	9.92797E+07
16	8	49	2	2181	2189	1.09090E+08
17	2	59	0	1165	1167	5.75142E+07
18	175	12	8	2034	2400	1.19977E+08
19	159	157	17	2230	5100	2.54951E+08
20	201	169	7	2679	2880	1.43462E+08
21	18	46	1	5100	5118	2.55723E+08
22	200	401	0	2320	4010	2.00150E+08
23	0	106	11	5230	5230	2.61081E+08
24	131	339	16	4325	4800	2.39954E+08
25	99	53	0	4517	4616	2.30369E+08
26	43	120	21	1323	6300	3.14940E+08
27	32	0	0	1496	1528	7.62288E+07
28	98	137	15	1993	4500	2.24957E+08
29	302	182	9	3025	3327	1.66174E+08
30	241	4	0	5419	5660	2.82209E+08
31	128	0	0	4347	4475	2.22879E+08
32	182	295	32	3476	9600	4.79908E+08
33	30	256	20	4006	6000	2.99943E+08
34	51	0	40	1705	12000	5.99885E+08
35	0	160	0	1360	1600	7.97012E+07
36	230	141	46	1561	13800	6.89868E+08
37	0	48	0	14969	14969	7.47457E+08

38	19	48	23	3190	6900	3.44934E+08
39	162	236	7	1807	2360	1.17240E+08
40	36	144	1	3060	3096	1.54440E+08
41	156	250	11	1955	3300	1.64968E+08
42	94	11	2	1337	1431	7.08717E+07
43	370	0	0	1212	1582	7.88330E+07
44	185	64	6	2114	2299	1.14277E+08
45	2	12	1	2782	2784	1.38897E+08
46	352	252	24	3360	7200	3.59931E+08
47	47	18	0	1439	1486	7.34169E+07
48	92	48	48	6851	14400	7.19862E+08
49	96	1088	96	11883	28800	1.43972E+09
50	186	458	0	11884	12070	6.03186E+08
51	10	425	0	4204	4250	2.12318E+08
52	991	0	0	9603	10594	5.29294E+08
53	1326	0	0	10817	12143	6.06599E+08
54	761	128	0	10903	11664	5.82301E+08
55	1539	16	0	10341	11880	5.93376E+08
56	100	212	6	4578	4678	2.33273E+08
57	851	0	0	7145	7996	3.99110E+08
58	1180	55	2	7700	8880	4.43405E+08
59	0	2400	0	11888	24000	1.19977E+09
60	10	552	0	20371	20381	1.01805E+09
61	0	2076	0	15079	20760	1.03763E+09
62	301	259	20	4888	6000	2.99943E+08
63	0	7	39	4846	11700	5.84888E+08
64	1118	515	64	10451	19200	9.59816E+08
65	289	176	0	12721	13010	6.50064E+08
66	244	85	45	6310	13500	6.74871E+08
67	94	114	57	2461	17100	8.54836E+08
68	192	0	0	4850	5042	2.51346E+08

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

Geometric Average Area: 2.63421e+08
CPU Runtime: 4.303000e-03 s (on UG machines)
Command line function required to run the program for this architecture:

- *make*
- *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 1*

4. Geometric average for architectures with no LUTRAM and one type of block RAM

BRAM Size	Max Width	LBs / BRAM	Geometric Average FPGA Area (min width transistors)
1 kbit	4	2	3.289566e+08
2 kbit	8	2	2.476834e+08
4 kbit	16	4	2.221212e+08
8 kbit	32	6	2.156838e+08
16 kbit	32	8	2.216362e+08
32 kbit	64	12	2.427788e+08
64 kbit	64	16	2.841202e+08
128 kbit	128	26	3.518061e+08

Table 2: Results without LUTRAM. BRAM configuration that maps to smallest area.

Command line function that runs the mapping tool and calculated the above table:

- *make*
- *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 2 table*

The RAM mapping program mapped the benchmark circuits using BRAMs of sizes ranging from 1k bits to 128k bits, max widths of 2 to 512 (powers of 2) and ratio of Logic Blocks per BRAM from 1 to 100. Above in Table 2 is shown the best BRAM configuration for each size that produced the smallest FPGA area. The 8k bit, 32 bit wide configuration managed to map the benchmark circuits to an area of 2.156838e+08 compared to 2.63421e+08 using the Stratix-IV-like architecture.

A trend seen from Table 2 is that the best max width of the BRAM increases as the size of the BRAM increases, which highlights how it is better to allow wider BRAMs if they are able to hold more bits, so that large RAMs can be stored using less BRAM blocks. Also, the ratio of Logic Blocks to BRAMs increased also along with the BRAM size. This relationship shows how it is valuable to have smaller BRAM blocks densely available on the FPGA since they take less space (area) compared to big BRAMs such as the 64k and 128k. These BRAMs are better to disperse sparsely across the FPGA because they add up to a lot of area due to their bit size. A large BRAM uses a lot of Logic Blocks to be constructed, therefore the FPGA should have a smaller ratio of them compared to LUTRAMs and small BRAMs.

Finally, a trend seen from Table 2 is that the Geometric Area first decreases from the first row, reaches a minimum at the 8k bit BRAM configuration, and then increases as the size gets bigger. This highlights how a BRAM configuration that is not too big and not too small works best for a wide variety of benchmark circuits that include very small and big logical RAMs. If only large 128k BRAMs are available for RAM, then a lot of them will be required to map all logical RAMs, and they would not be using all available bits as well due to the large excess of unused bits. It is better to have smaller sized BRAMs that have a couple thousand bits available to map RAMs. Here, some extra circuitry might be required to combine multiple BRAMs in series or parallel, but it wouldn't increase the FPGA area too much (compared to filling it with huge 128k BRAMs).

5. Geometric average for architectures with LUTRAM and one type of block RAM

BRAM Size	Max Width	LBs / BRAM	Geometric Average FPGA Area (min width transistors)
1 kbit	4	2	3.260844e+08
2 kbit	8	3	2.537343e+08
4 kbit	8	4	2.282952e+08
8 kbit	16	7	2.200825e+08
16 kbit	32	12	2.224975e+08
32 kbit	64	21	2.341142e+08
64 kbit	64	33	2.559382e+08
128 kbit	128	64	2.802511e+08

Table 3: Results with LUTRAM. BRAM configuration that maps to smallest area.

Command line function that runs the mapping tool and calculated the above table:

- *make*
- *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 3 table*

Here when mapping the benchmark circuits using LUTRAM and 1 type of BRAM, the 8k bit size, 16 bit wide and 7:1 ratio configuration maps the best Geometric Average of 2.200825e+08 compared to 2.63421e+08 using the Stratix-IV-like architecture and 2.156838e+08 of the single BRAM mapping. Surprisingly using LUTRAM increases the area of RAM mappings, which is due to the specific algorithm used in the tool. It chooses a mapping depending on the "local" area of mapping each RAM individually. The area of a logic block is 40000 compared to 35000 when it is capable of supporting LUTRAM, thus using LUTRAM can increase the total area needed by the FPGA (especially if LUTRAM requires 50% of the board's total logic blocks in this architecture).

The same trends regarding the increase in max width and ratio as the BRAM size increases is the same as in the previous mappings without LUTRAM. Interestingly, the Geometric Average is smaller than the previous exploration for BRAM sizes of 32, 64 and 128 kbits (the larger sizes). Having access to LUTRAM decreases the mapping area a lot, especially when there are small logical RAMs to map. LUTRAM is effective for that. The 640 bit LUTRAMs can map a wide range of small sized RAMs with less excess circuitry compared to the huge 128k bit BRAMs. Thus, this exploration highlights how it is valuable to have some LUTRAM availability and large-sized BRAMs to map a wide range of logical RAMs.

6. Best custom FPGA RAM architecture

    a. No LUTRAM
    b. BRAM1: 4k bits, 16 bit max width, 4:1 ratio
    c. BRAM2: 64k bits, 64 bit max width, 64:1 ratio
    d. Geo Average Area: 2.246549e+08
    e. CPU Runtime: 3.991000e-03

Above are the parameters of the best RAM architecture I found using my RAM mapping tool. I decided to use 2 BRAMS, a smaller 4k bit and a larger 64k bit BRAM that offer different max widths. The ratios for each were picked according to the trend discovered in parts 4 and 5, where smaller sized BRAMs can be placed more densely on the FPGA unlike the large ones because they take up a lot of area. In addition, I did not use LUTRAM in my mapping because it actually increased the mapping area due to my algorithm taking big consideration of the fact that logic block area is 14% larger when using LUTRAMs. I also tested the mapping when a smaller fraction of logic blocks supported LUTRAM, for example 20% rather than 50%, but abandoning LUTRAM gained the most savings in area for me. I managed to find a good pair of BRAMs that can successfully balance the need for small RAMs and larger RAMs. The ratio of 4k bit BRAMs is much smaller than for the 64k bit BRAMs in order to not waste mapping unused memory bits on the FPGA due to the larger BRAMs. Mapping with a single type of BRAM (8k) ultimately had the smallest mapped area in this assignment, but I think using at least 2 types of physical RAM is necessary for a good mapping architecture in order to efficiently map logical RAMs of a variety of sizes. For the future, I would increase the complexity of the mapping algorithm to include checking the size of the logical RAMs and mapping the larger ones using larger BRAMs and using LUTRAM every so often (as long as it doesn't become the limiting resource when calculating area). I would use a function that calculates the global total FPGA area so far as we map each logical RAM as described in part 1, this method would find more potential area savings and decrease the number of required Logic Block tiles in the chip.

Appendix

Listing of source code of RAM mapping program:

- README
  - includes description and walkthrough of running program (on UG machines)
- Makefile
  - Compile code using *make*, clean using *make clean*
- Main.cpp
  - Main file, program starts here taking in arguments from command line
- RamMapper.cpp/.h
  - RamMapper class defined, includes vector of Circuit objects
- Circuit.cpp/.h
  - Circuit class defined, includes vector of RAM objects
- RAM.cpp/.h
  - RAM class defined, includes both logical and physical RAM data variables
- Checker
  - Executable for UG machines
- Logic_block_count.txt
  - Input file for program
- Logical_rams.txt
  - Input file for program
- Mapping.txt
  - Mapping file generated by program (for Stratix-IV-like architecture)
- Table1.txt
  - Output of running program for Stratix-IV-like architecture functions:
    - *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 1*
    - *./checker -d logical_rams.txt logic_block_count.txt mapping.txt*
- Table2.txt
  - Output of running program command for testing single BRAMs with no LUTRAM:
    - *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 2 table*
- Table3.txt
  - Output of running program command for testing single BRAMs with LUTRAM:
    - *./RAM_mapper logical_rams.txt logic_block_count.txt mapping.txt 3 table*