Lei Liu
[redacted]

Assignment 3

ECE 1756
November 27, 2023

1. Description of the algorithm

The mapping tool is written in C++.

1.1 structures

- class: logicRam
  We will parse input:logical_rams.txt, and store the information in logicRam. This includes
  - RamID for this logic ram
  - mode for this logic ram
  - depth for this logic ram
  - width for this logic ram
- class: mappedRam
  This is the structure used to store the mapping result for each logic ram. We will directly retrieve the info stored here to generate the output mapping result. This structure includes
  - RamID: id for the corresponding logic ram
  - lram_mode: mode for the corresponding logic ram
  - logic_depth: depth for the corresponding logic ram
  - logic_width: width for the corresponding logic ram
  - mappedram_id: the unique id for this specific mapped result
  - type: the physical ram type assigned for this result
  - serial: how many serial needed for this result
  - parallel: how many parallel needed for this result
  - mapped_depth: depth assigned for this result
  - mapped_width: width assigned for this result
  - additional_lut: luts used to implement
  - cost: area cost (so far) if implemented in the circuit

    *The last two items are stored in order to compare different candidates during the analysis
- class: circuit
  This will further wrap the first two structure together as a circuit based.

- circuit_id: id for each circuit
- num_lb:number of existing logic blocks for each circuit
- logic_ram_list: circuit contains a list of logicRam that need to be taken care of. We will prepare all the info after parsing the input text file.
- mapped_ram_list: circuit needs a list of mappedRam which corresponds to each logicRam. We will fill this list during the analysis.

■ vector{circuit} logic_circuit_list
This vector contains 69 circuit info, which will be created after parsing input. During analysis, we will unpack circuit info and generate mapping during 69 iteration.

■ class: resource
Each available physical candidate will be stored as resource. Info for the resource are directly provided by user. This includes

- physical_ram: the type of the physical ram
- ratio: ratio respective to logic block
- max_width: the max width available
- size: size of the physical ram
- possible_comb: all possible depth and width combination for this physical ram will be stored as a vector here

■ vector{resource} arch_resource_list
This vector lists all the possible physical ram resource (with its details) available. During analysis, we will iterate through each possible physical ram resource, and find a best result.

1.2 algorithm

parse_input:
Use the file stream to parse two input text files. Circuit id, number of existing logic blocks, and all the required info for logic ram are prepared in the structure:vector{circuit} logic_circuit_list.

construct_resource:
based on user input, resource are prepared and inserted in the structure: vector{resource} arch_resource_list

perform_basic_core_mapper:
This is the core function to analyze each logic ram and decide the mapped result.
vector{circuit} logic_circuit_list is passed in, since it contains info for each circuit, each logic

ram. vector{resource} arch_resource_list is also passed in, since it provides options of different physical rams.
We will first iterate through each circuit. For each circuit, we will then iterate through each logical ram.
For each logical ram, iterate through each possible resource(physical ram candidate), and then iterate through each possible depth and width combination within this resource.
Therefore, for a logical ram, in each "round", it has a hypothesis of a possible physical ram, with a possible depth and width combination assigned. We will then calculate the area used so far in this circuit using the formula given in the handout.

$$\begin{aligned}&\text{area for analysis} = \text{area used for previous logical ram in this circuit} + \\ &\text{area used if the current logical ram used this physical ram with this combination}\end{aligned} \tag{1}$$

We will compare the area analyzed in each round, and choose the physical ram with the depth+width combination that achieves the lowest area among other candidates. The selected result will be stored in the mappedRam list for the corresponding logical ram. Then, we will move to the next logical ram and repeat the above analysis.

This algorithm is similar to the idea of "greedy". "For the current object, among all the possible results, take the one with the lowest cost and then move to the next object." However, it strictly follow the order of those logical rams being inserted. And when a logic ram has been decided with an output, we never look back and change it.

Other functions deal with input arguments, different structures, and generate output files. Please refer to README file for more details.

1.3 A derivation of the computational complexity

As mentioned in the previous section, we iterate through each circuit, each logic ram, each available physical resource, each possible depth+width combination.

{a{b{c{d}}}}
a = circuit, b = logic rams, c = physical resources, d = depth+width combination

Therefore, the complexity is: $O(C \times N^4)$
Although this complexity sounds horrible, in our test, a = 69 fixed, c = 3 in maximum, which still allows the program complete the execution in an acceptable amount of time.

2. Fixed Stratix-IV like architecture

Command line for this architecture:

./mapping 1
make sure you have compile the program and generate the mapping executable

Please see figure 1 and figure 2 for the completed table 1 of Stratix-IV like architecture.
Geometric Average Area = 2.162e+08
CPU runtime = 121 milliseconds

Circuit	Type 1	Type 2	Type 3	Blocks	Tiles	Area	
0	1003	414	13	3097	4140	2.0628e+08	Pass
1	1421	620	21	3349	6300	3.1494e+08	Pass
2	0	62	0	1836	1836	9.16224e+07	Pass
3	0	90	0	2808	2808	1.3999e+08	Pass
4	937	926	30	8279	9260	4.62174e+08	Pass
5	0	312	0	3692	3692	1.84285e+08	Pass
6	42	192	6	1874	1920	9.56414e+07	Pass
7	628	475	15	4107	4750	2.36746e+08	Pass
8	155	562	18	5342	5620	2.80322e+08	Pass
9	0	33	0	1636	1636	8.13408e+07	Pass
10	417	270	10	1587	3000	1.49971e+08	Pass
11	159	151	5	1358	1517	7.57197e+07	Pass
12	0	43	0	1632	1632	8.11908e+07	Pass
13	0	24	0	4491	4491	2.23672e+08	Pass
14	277	232	7	1885	2320	1.15354e+08	Pass
15	0	154	0	1956	1956	9.7281e+07	Pass
16	0	88	0	2181	2181	1.0879e+08	Pass
17	0	61	0	1165	1165	5.74392e+07	Pass
18	0	156	6	2036	2036	1.01053e+08	Pass
19	392	273	9	2316	2730	1.36389e+08	Pass
20	294	329	11	2808	3300	1.64968e+08	Pass
21	0	76	0	5100	5100	2.54951e+08	Pass
22	717	337	11	2553	3370	1.68269e+08	Pass
23	0	252	0	5230	5230	2.61081e+08	Pass
24	190	505	16	4391	5050	2.51743e+08	Pass
25	0	112	0	4517	4517	2.25691e+08	Pass
26	127	275	9	1444	2750	1.37332e+08	Pass
27	0	20	0	1496	1496	7.38885e+07	Pass
28	258	311	10	2115	3110	1.55158e+08	Pass
29	336	356	11	3120	3560	1.77229e+08	Pass
30	0	215	0	5419	5419	2.70757e+08	Pass
31	0	80	0	4347	4347	2.16824e+08	Pass
32	701	655	21	3705	6550	3.26728e+08	Pass
33	248	474	14	4166	4740	2.36274e+08	Pass
34	21	442	15	2005	4500	2.24957e+08	Pass
35	52	143	4	1376	1430	7.08342e+07	Pass
36	415	755	17	1789	7550	3.77286e+08	Pass
37	0	48	0	14969	14969	7.47457e+08	Pass
38	0	320	6	3202	3202	1.59477e+08	Pass
39	622	289	9	2104	2890	1.43934e+08	Pass
40	0	179	0	3060	3060	1.52801e+08	Pass
41	817	322	10	2337	3220	1.60345e+08	Pass
42	0	90	0	1337	1337	6.63812e+07	Pass
43	121	131	4	1212	1333	6.62312e+07	Pass
44	64	239	7	2143	2390	1.18655e+08	Pass
45	0	29	0	2782	2782	1.38822e+08	Pass
46	981	464	15	3627	4640	2.31559e+08	Pass
47	0	55	0	1439	1439	7.11717e+07	Pass
48	0	574	20	6882	6882	3.43215e+08	Pass
49	1595	1559	52	12193	15600	7.79851e+08	Pass
50	0	580	0	11884	11884	5.93526e+08	Pass
51	0	420	12	4204	4204	2.1011e+08	Pass
52	0	641	0	9603	9603	4.80021e+08	Pass
53	0	816	0	10817	10817	5.40631e+08	Pass
54	0	885	0	10903	10903	5.44725e+08	Pass
55	223	1056	35	10341	10564	5.27879e+08	Pass
56	0	374	0	4578	4578	2.28558e+08	Pass

Figure 1: Table 1(1) for Stratix-IV

57	0	466	0	7145	7145	3.56439e+08	Pass
58	0	790	16	7700	7900	3.94641e+08	Pass
59	4184	1817	60	13562	18170	9.07844e+08	Pass
60	0	558	0	20371	20371	1.01758e+09	Pass
61	0	1890	62	15079	18900	9.44819e+08	Pass
62	292	524	17	4936	5240	2.61553e+08	Pass
63	0	455	11	4846	4846	2.42065e+08	Pass
64	2072	1340	45	11216	13500	6.74871e+08	Pass
65	0	357	0	12721	12721	6.35576e+08	Pass
66	8	703	22	6355	7030	3.51064e+08	Pass
67	1588	539	15	3077	5390	2.68626e+08	Pass
68	0	192	0	4850	4850	2.42312e+08	Pass
Geometric Average Area: 2.16187e+08							

Figure 2: Table 1(2) for Stratix-IV

3. No LUTRAM, and one type of block RAM

BRAM Size	Max Width	LBs / BRAM	Geometric Average FPGA Area (min width transistors)
1 kbit	4	1	2.499e+08
2 kbit	16	2	2.348e+08
4 kbit	16	4	2.172e+08
8 kbit	32	6	2.142e+08
16 kbit	32	8	2.216e+08
32 kbit	64	12	2.428e+08
64 kbit	64	16	2.841e+08
128 kbit	128	25	3.519e+08

Table 1: Results without LUTRAM

From Table 1, we can observe that max width would increase with the BRAM size in order to achieve a good value of average area. If we limit the width to the same across all different BRAM Size, larger BRAM seems to waste more area than smaller BRAM during implementaion. Therefore, a larger value of Max Width is more suitable for larger BRAM.

Besides, high LB/BRAM ratio is perferred for larger BRAM. By increasing the ratio, we sort of "limit" the use of larger size BRAM as in most case, they occupy much area but the logical ram is not usually fully utilize the BRAM assigned.

Acorss the BRAM size from small to large, we can observe the trend that: average area decrease to the lowest at BRAM Size = 8 kbit and then start to increase. For BRAM that has size smaller than 8 kbit, they require more pieces to be implemented in serial/parallel, which leads to additional area cost of number of blocks used in luts. And for BRAM that has size larger than 8 kbit, as mentioned above, they cost area waste since they are usually not fully utilized.

4. one type of block RAM, with 50% LUTRAM support

As table 2 shows the result of structure with 50% lutram support, and one type of block RAM. Compared to table 1, it is noticed that, the structure with lutram support achieve a smaller area in general across all sizes of BRAM. LUTRAM handle small logical RAM better with smaller area cost.

BRAM Size	Max Width	LBs / BRAM	Geometric Average FPGA Area (min width transistors)
1 kbit	2	1	2.616e+08
2 kbit	8	2	2.267e+08
4 kbit	16	4	2.170e+08
8 kbit	32	6	2.159e+08
16 kbit	32	8	2.179e+08
32 kbit	64	15	2.268e+08
64 kbit	64	30	2.478e+08
128 kbit	64	40	2.712e+08

Table 2: Results with LUTRAM

5. Designed structure

LUTRAM with 1/4 of logic blocks that support LUTRAM
BRAM1: 8 kbit, max width = 32, ratio = 6
BRAM2: 64 kbit, max width = 64, ratio = 800
geometric average area = 2.109e+08

Followed the trends found in Table 1 and Table 2. 8 kbit BRAM with max width = 32, ratio = 6 is used as it achieves the minimum area among other types of BRAM. LUTRAM are inserted with 1/4 ratio of support to better serve small logical ram in the testcases.
64 kbit BRAM is chosen to provide options for large logical ram. We passed in a high ratio to limit so that it won't be frequently used.

Appendix

Appendix I: Source code of the tool

Due to the page limit, only circuit.cpp which contains the core computation part is uploaded here. Please see other files along with the submission to get a full view of the code.

```
#include "circuit.h"

string logicRam::get_lram_mode() {
    string mode_result;
    switch (mode){
        case SimpleDualPort:
            mode_result = "SimpleDualPort";
        break;
        case ROM:
            mode_result = "ROM";
        break;
        case SinglePort:
            mode_result = "SinglePort";
        break;
        case TrueDualPort:
            mode_result = "TrueDualPort";
        break;
    }
    return mode_result;
}

void resource::prepare_combination(){
    for (int poss_width = max_width; poss_width >= 1; poss_width /= 2){
        unsigned int poss_depth = size/poss_width;
        pair<unsigned int, unsigned int> combination(poss_depth,
        ↪  poss_width);
        add_dw_combination(combination);
    }
}


void construct_resource(vector<resource>& resource_list, operationType op,
↪  input_parameter input_pack) {
    if(op == STRATIX_IV){
        //insert LUTRAM resource
```

```
        vector<pair<unsigned int, unsigned int> > comb;
        resource lut_resource = resource(LUTRAM, 2.0, 64, 20, 640, comb);
        //directly insert, since only two combination available
        pair<unsigned int, unsigned int> comb1(64, 10);
        pair<unsigned int, unsigned int> comb2(32, 20);
        lut_resource.add_dw_combination(comb1);
        lut_resource.add_dw_combination(comb2);
        resource_list.push_back(lut_resource);

        //insert 8192 BRAM
        comb.empty();
        resource bram8192_resource = resource(BRAM_8192, 10.0, 8192, 32,
        ↪  8192, comb);
        bram8192_resource.prepare_combination();
        resource_list.push_back(bram8192_resource);

        //insert 128k BRAM
        comb.empty();
        resource bram128k_resource = resource(BRAM_128K, 300.0, 131072, 128,
        ↪  131072, comb);
        bram128k_resource.prepare_combination();
        resource_list.push_back(bram128k_resource);
    }else if(op == NO_LUTRAM || op == WITH_LUTRAM){
        //insert the only type of BRAM according to the input
        vector<pair<unsigned int, unsigned int> > comb;
        unsigned int ratio = input_pack.get_input_bram_ratio()[0];
        unsigned int bram = input_pack.get_input_bram_size()[0];
        unsigned int m_width = input_pack.get_input_bram_mwidth()[0];
        resource bramcustom_resource = resource(BRAM_CUSTOM, ratio, bram,
        ↪  m_width, bram, comb);
        bramcustom_resource.prepare_combination();
        resource_list.push_back(bramcustom_resource);
        if(op == WITH_LUTRAM){
            //insert LUTRAM resource
            vector<pair<unsigned int, unsigned int> > comb;
            resource lut_resource = resource(LUTRAM, 2.0, 64, 20, 640,
            ↪  comb);
            //directly insert, since only two combination available
            pair<unsigned int, unsigned int> comb1(64, 10);
            pair<unsigned int, unsigned int> comb2(32, 20);
            lut_resource.add_dw_combination(comb1);
            lut_resource.add_dw_combination(comb2);
```

```
            resource_list.push_back(lut_resource);
        }
    }else if(op == MB_WITH_LUTRAM){
        //insert LUTRAM resource
        vector<pair<unsigned int, unsigned int> > comb;
        resource lut_resource =
        ↪  resource(LUTRAM,input_pack.get_input_lutram_ratio(), 64, 20,
        ↪  640, comb);
        //directly insert, since only two combination available
        pair<unsigned int, unsigned int> comb1(64, 10);
        pair<unsigned int, unsigned int> comb2(32, 20);
        lut_resource.add_dw_combination(comb1);
        lut_resource.add_dw_combination(comb2);
        resource_list.push_back(lut_resource);

        comb.empty();
        resource bramcustom_resource = resource(BRAM_CUSTOM,
        ↪  input_pack.get_input_bram_ratio()[0],

                ↪  input_pack.get_input_bram_size()[0],input_pack.get_input_bram_mwidth
               input_pack.get_input_bram_size()[0], comb);
        bramcustom_resource.prepare_combination();
        resource_list.push_back(bramcustom_resource);

        comb.empty();
        resource bramcustom2_resource = resource(BRAM_CUSTOM_2,
        ↪  input_pack.get_input_bram_ratio()[1],

                ↪  input_pack.get_input_bram_size()[1],input_pack.get_input_bram_mwidth
               input_pack.get_input_bram_size()[1], comb);
        bramcustom2_resource.prepare_combination();
        resource_list.push_back(bramcustom2_resource);
    }
}

void perform_basic_core_mapper(vector<circuit>& logic_circuit_list,
↪  vector<resource>& resource_list){
    for (auto& circuit: logic_circuit_list){
        unsigned int existing_LB = circuit.get_circuit_num_lb();
        unsigned int used_lutram = 0;
        unsigned int used_8192bram = 0;
        unsigned int used_128kbram = 0;
```

```
        int id_count = 0;
//        circuit.set_circuit_area(existing_LB * (35000+40000)/2); //MAGIC
↪  NUMBER HERE
        double circuit_areatested = 0.0;
//        cout<<"process circuit id: "<<circuit.get_circuit_id()<<endl;
//        cout<<"it has: "<<circuit.get_ram_list().size()<<" number of
↪  logic rams"<<endl;

        int logicram_count = 0;
        for(auto& logicram: circuit.get_ram_list()){
            //current cheapest mapped RAM for this current logic ram across
            ↪  all possible physical candidates
            vector<mappedRam> cheapest_map_list;
            double cheapest_area = DBL_MAX;

            for(auto physical_candidate: resource_list){
                if(physical_candidate.get_pram_type() == LUTRAM &&
                ↪  logicram.get_lram_mode() == "TrueDualPort"){
                    // lutram cannot support TDP
                    continue;
                }

                //iterate through all possible dw combinations of current
                ↪  physical candidate
                for(auto dw_pair: physical_candidate.get_comb_list()){
                    unsigned int curr_depth = dw_pair.first;
                    unsigned curr_width = dw_pair.second;

                    if(curr_width == physical_candidate.get_max_width() &&
                    ↪  logicram.get_lram_mode() == "TrueDualPort"){
                        //widest width is not available for TDP
                        continue;
                    }

                    int mapper_id = id_count ++;
                    //decide how to locate logic using physical candidate
                    unsigned int p = 1;
                    unsigned int s = 1;
                    unsigned int num_luts = 0;
                    if(logicram.get_lram_width() > curr_width){
                        p = (unsigned
                        ↪  int)ceil((double)logicram.get_lram_width()/(double)curr_widt
```

```
}
if(logicram.get_lram_depth() > curr_depth){
    s = (unsigned
    ↪  int)ceil((double)logicram.get_lram_depth()/(double)curr_dept
    if (s > 16){
        //dont consider any solution that is 16x deeper
        continue;
    }
    //compute extra logic needed(num_of_luts) when in
    ↪  serial
    if(s <= 4){
        if(s == 2){
            num_luts = 1*logicram.get_lram_width() + 1;
        }else{
            num_luts = 1*logicram.get_lram_width() + s;
        }
    }else if(s <= 7){
        num_luts = 2*logicram.get_lram_width() + s;
    }else if(s <= 10){
        num_luts = 3*logicram.get_lram_width() + s;
    }else if(s <= 13){
        num_luts = 4*logicram.get_lram_width() + s;
    }else{
        num_luts = 5*logicram.get_lram_width() + s;
    }
}
if(logicram.get_lram_mode() == "TrueDualPort") {// NOT
↪  SURE
    num_luts *= 2;
}
//test the area if use this pram, this combination
unsigned int num_pram_plan = s * p;
unsigned int extra_logic_LB = ceil((double)num_luts /
↪  10.0);//MAGIC NUMBER HERE
unsigned int LB_plan = 0; // total LB if use this pram
unsigned int bram8192_plan = 0;
unsigned int bram128k_plan = 0;
if(physical_candidate.get_pram_type() == LUTRAM){
    LB_plan = existing_LB + used_lutram + num_pram_plan
    ↪  + extra_logic_LB;
}else{
```

```
        LB_plan = existing_LB + used_lutram +
        ↪  extra_logic_LB;
        if(physical_candidate.get_pram_type() == BRAM_8192)
        ↪  {
            bram8192_plan = used_8192bram + num_pram_plan;
        }else {
            bram128k_plan = used_128kbram + num_pram_plan;
        }
    }

    unsigned int LBrequired_plan;
    if(LB_plan >= bram8192_plan * 10 && LB_plan >=
    ↪  bram128k_plan * 300){
        LBrequired_plan = LB_plan;
    }else if(bram8192_plan * 10 >= LB_plan && bram8192_plan
    ↪  * 10 >= bram128k_plan * 300){
        LBrequired_plan = bram8192_plan * 10;
    }else{
        LBrequired_plan = bram128k_plan * 300;
    }
    int required_8192 =
    ↪  floor((double)LBrequired_plan/(double)10);
    int required_128k =
    ↪  floor((double)LBrequired_plan/(double)300);
    double try_area = LBrequired_plan * 37500 +
    ↪  required_8192 * (9000 + 5 * 8192 + 90 *
    ↪  sqrt((double)8192) + 600 * 2 * 32)
            + required_128k * (9000 + 5 * 131072 + 90 *
            ↪  sqrt((double)131072) + 600 * 2 * 128);
    if(try_area < cheapest_area){
        if(try_area == 0){
            cout<<"Something went wrong, area should not be
            ↪  zero!!!"<<endl;
            exit(1);
        }
        cheapest_area = try_area;
        cheapest_map_list.insert(cheapest_map_list.begin(),
                mappedRam(logicram.get_lram_id(), mapper_id,
                ↪  num_luts, logicram.get_lram_depth(),
                        logicram.get_lram_width(), s, p,
                        ↪  physical_candidate.get_pram_type(),
```

```
                                logicram.get_lram_mode(),
                                ↪ curr_depth, curr_width,
                                ↪ try_area));
            }
        }// iterate through each combination of the same physical
        ↪ ram
    }// iterate through all physical candidates, we have found the
    ↪ cheapset map for this logic ram

    if(cheapest_map_list.empty()){
        cout<<"No available mapped result found!!!"<<endl;
        cout<<"Something went wrong"<<endl;
        exit(1);
    }
    mappedRam cheapest_map = cheapest_map_list[0];

    ↪ logic_circuit_list[circuit.get_circuit_id()].add_mapped_ram(cheapest_map

    existing_LB += ceil((double)(cheapest_map.get_lut()) /
    ↪ 10.0);//MAGIC NUMBER HERE
    if(cheapest_map.get_map_type() == LUTRAM){
        used_lutram += cheapest_map.get_s() * cheapest_map.get_p();
    }else if(cheapest_map.get_map_type() == BRAM_8192){
        used_8192bram += cheapest_map.get_s() *
        ↪ cheapest_map.get_p();
    }else{
        used_128kbram += cheapest_map.get_s() *
        ↪ cheapest_map.get_p();
    }
    logicram_count++;
    if(logicram_count == circuit.get_ram_list().size()){
        circuit_areatested = cheapest_map.get_total_cost();
        if(circuit_areatested == 0){
            cout<<"area got from mapped is zero!!!"<<endl;
            cout<<"Something went wrong"<<endl;
            exit(1);
        }
        //print for debug purpose: this current circuit info
        cout<<"circuit: "<<circuit.get_circuit_id()<<" used LUTRAM:
        ↪ "<<used_lutram
        <<", used 8192BRAM: "<<used_8192bram
        <<", used 128k BRAM: "<<used_128kbram
```

```
                <<", my tested area is: "<<circuit_areatested<<endl;
            }
        }// all logic ram in this circuit have been mapped

        ↪ logic_circuit_list[circuit.get_circuit_id()].set_circuit_area(circuit_areate
    }
}

void perform_custom_core_mapper(vector<circuit>& logic_circuit_list,
↪ vector<resource>& resource_list, operationType op,
                                vector<unsigned int>ratio_list){
    for (auto& circuit: logic_circuit_list){
        unsigned int existing_LB = circuit.get_circuit_num_lb();
        unsigned int used_lutram = 0;
        unsigned int used_cusbram = 0;
        unsigned int used_cusbram2 = 0;
        int id_count = 0;
        double circuit_areatested = 0.0;

        int logicram_count = 0;
        for(auto& logicram: circuit.get_ram_list()){
            //current cheapest mapped RAM for this current logic ram across
            ↪ all possible physical candidates
//            cout<<"processing circuit: "<<circuit.get_circuit_id()<<",
↪ logic ram id: "<<logicram_count<<endl;
            vector<mappedRam> cheapest_map_list;
            double cheapest_area = DBL_MAX;

            for(auto physical_candidate: resource_list){
                if(physical_candidate.get_pram_type() == LUTRAM &&
                ↪ logicram.get_lram_mode() == "TrueDualPort"){
                    // lutram cannot support TDP
                    continue;
                }

                //iterate through all possible dw combinations of current
                ↪ physical candidate
                for(auto dw_pair: physical_candidate.get_comb_list()){
                    unsigned int curr_depth = dw_pair.first;
                    unsigned curr_width = dw_pair.second;
```

```
if(curr_width == physical_candidate.get_max_width() &&
↪  logicram.get_lram_mode() == "TrueDualPort"){
    //widest width is not available for TDP
    continue;
}

int mapper_id = id_count ++;
//decide how to locate logic using physical candidate
unsigned int p = 1;
unsigned int s = 1;
unsigned int num_luts = 0;
if(logicram.get_lram_width() > curr_width){
    p = (unsigned
    ↪  int)ceil((double)logicram.get_lram_width()/(double)curr_widt
}
if(logicram.get_lram_depth() > curr_depth){
    s = (unsigned
    ↪  int)ceil((double)logicram.get_lram_depth()/(double)curr_dept
    if (s > 16){
        //dont consider any solution that is 16x deeper
        continue;
    }
    //compute extra logic needed(num_of_luts) when in
    ↪  serial
    if(s <= 4){
        if(s == 2){
            num_luts = 1*logicram.get_lram_width() + 1;
        }else{
            num_luts = 1*logicram.get_lram_width() + s;
        }
    }else if(s <= 7){
        num_luts = 2*logicram.get_lram_width() + s;
    }else if(s <= 10){
        num_luts = 3*logicram.get_lram_width() + s;
    }else if(s <= 13){
        num_luts = 4*logicram.get_lram_width() + s;
    }else{
        num_luts = 5*logicram.get_lram_width() + s;
    }
}
if(logicram.get_lram_mode() == "TrueDualPort") {// NOT
↪  SURE
```

```
    num_luts *= 2;
}
//test the area if use this pram, this combination
unsigned int num_pram_plan = s * p;
unsigned int extra_logic_LB = ceil((double)num_luts /
↪  10.0);//MAGIC NUMBER HERE
unsigned int LB_plan = 0; // total LB if use this pram
unsigned int bram_plan = 0;
unsigned int bram1_plan = 0;
unsigned int bram2_plan = 0;
if(physical_candidate.get_pram_type() == LUTRAM){
    LB_plan = existing_LB + used_lutram + num_pram_plan
    ↪  + extra_logic_LB;
}else{
    LB_plan = existing_LB + used_lutram +
    ↪  extra_logic_LB;
    if(op == MB_WITH_LUTRAM){
        if(physical_candidate.get_pram_type() ==
        ↪  BRAM_CUSTOM){
            bram1_plan = used_cusbram + num_pram_plan;
        }else{
            bram2_plan = used_cusbram2 + num_pram_plan;
        }
    }else{
        bram_plan = used_cusbram + num_pram_plan;
    }
}

unsigned int LBrequired_plan;
if(op == MB_WITH_LUTRAM){
    if(LB_plan >= bram1_plan * ratio_list[1] && LB_plan
    ↪  >= bram2_plan * ratio_list[2]){
        LBrequired_plan = LB_plan;
    }else if(bram1_plan * ratio_list[1] >= LB_plan &&
    ↪  bram1_plan * ratio_list[1]
    >= bram2_plan * ratio_list[2]){
        LBrequired_plan = bram1_plan * ratio_list[1];
    }else{
        LBrequired_plan = bram2_plan * ratio_list[2];
    }
}else{
```

```
        if(LB_plan >= bram_plan *
        ↪  physical_candidate.get_ratio()){
            LBrequired_plan = LB_plan;
        }else{
            LBrequired_plan = bram_plan *
            ↪  physical_candidate.get_ratio();
        }
    }

    int required_bram =
    ↪  floor((double)LBrequired_plan/(double)physical_candidate.get_rat
    double try_area;
    unsigned int bram_bits =
    ↪  physical_candidate.get_pram_size();
    unsigned int bram_mwidth =
    ↪  physical_candidate.get_max_width();
    if(op == NO_LUTRAM){
        try_area = LBrequired_plan * 35000 + required_bram *
        ↪  (9000 +
                5 * bram_bits + 90 * sqrt((double)bram_bits)
                ↪  + 600 * 2 * bram_mwidth);
    }else if(op == WITH_LUTRAM){
        try_area = LBrequired_plan * 37500 + required_bram *
        ↪  (9000 +
                5 * bram_bits + 90 * sqrt((double)bram_bits)
                ↪  + 600 * 2 * bram_mwidth);
    }else{
        unsigned int bram1_bits =
        ↪  resource_list[1].get_pram_size();
        unsigned int bram1_mwidth =
        ↪  resource_list[1].get_max_width();
        unsigned int bram2_bits =
        ↪  resource_list[2].get_pram_size();
        unsigned int bram2_mwidth =
        ↪  resource_list[2].get_max_width();

        int required_bram1 =
        ↪  floor((double)LBrequired_plan/(double)ratio_list[1]);
        int required_bram2 =
        ↪  floor((double)LBrequired_plan/(double)ratio_list[2]);
```

```
                        try_area = LBrequired_plan *
                        ↪  ((35000/(double)ratio_list[0]*(ratio_list[0]-1))
                        ↪  + (40000/(double)ratio_list[0])) +
                                required_bram1 * (9000 + 5 * bram1_bits + 90
                                ↪  * sqrt((double)bram1_bits) + 600 * 2 *
                                ↪  bram1_mwidth) +
                                required_bram2 * (9000 + 5 * bram2_bits + 90
                                ↪  * sqrt((double)bram2_bits) + 600 * 2 *
                                ↪  bram2_mwidth);
                    }
//                      cout<<"try area this round is: "<<try_area<<endl;
//                      cout<<"cheapest area this round is:
↪  "<<cheapest_area<<endl;
                    if(try_area < cheapest_area){
                        if(try_area == 0){
                            cout<<"Something went wrong, area should not be
                            ↪  zero!!!"<<endl;
                            exit(1);
                        }
                        cheapest_area = try_area;
                        cheapest_map_list.insert(cheapest_map_list.begin(),

                                                 ↪  mappedRam(logicram.get_lram_id(),
                                                 ↪  mapper_id, num_luts,
                                                 ↪  logicram.get_lram_depth(),

                                                        ↪  logicram.get_lram_width()
                                                        ↪  s, p,
                                                        ↪  physical_candidate.get_pr

                                                        ↪  logicram.get_lram_mode(),
                                                        ↪  curr_depth,
                                                        ↪  curr_width,
                                                        ↪  try_area));
                    }
                }// iterate through each combination of the same physical
                ↪  ram
            }// iterate through all physical candidates, we have found the
            ↪  cheapset map for this logic ram
            if(cheapest_map_list.empty()){
                cout<<"No available mapped result found!!!"<<endl;
                cout<<"Something went wrong"<<endl;
```

```
        exit(1);
    }
    mappedRam cheapest_map = cheapest_map_list[0];

    ↪ logic_circuit_list[circuit.get_circuit_id()].add_mapped_ram(cheapest_map

    existing_LB += ceil((double)(cheapest_map.get_lut()) /
    ↪ 10.0);//MAGIC NUMBER HERE
    if(cheapest_map.get_map_type() == LUTRAM){
        used_lutram += cheapest_map.get_s() * cheapest_map.get_p();
    }else if(cheapest_map.get_map_type() == BRAM_CUSTOM_2){
        used_cusbram2 += cheapest_map.get_s() *
        ↪ cheapest_map.get_p();
    }else{
        used_cusbram += cheapest_map.get_s() * cheapest_map.get_p();
    }
    logicram_count++;
    if(logicram_count == circuit.get_ram_list().size()){
        circuit_areatested = cheapest_map.get_total_cost();
        if(circuit_areatested == 0){
            cout<<"area got from mapped is zero!!!"<<endl;
            cout<<"Something went wrong"<<endl;
            exit(1);
        }
        //print for debug purpose: this current circuit info
        cout<<"circuit: "<<circuit.get_circuit_id()<<" used LUTRAM:
        ↪ "<<used_lutram
            <<", used customed BRAM: "<<used_cusbram
            <<", used customed BRAM2: "<<used_cusbram2
            <<", my tested area is: "<<circuit_areatested<<endl;
    }
    if(op == NO_LUTRAM && used_lutram != 0){
        cout<<"Lutram used is not zero in this operation mode, which
        ↪ is not expected!!!"<<endl;
        cout<<"Something went wrong"<<endl;
        exit(1);
    }
    if(op == NO_LUTRAM || op == WITH_LUTRAM){
        if(used_cusbram2 != 0){
            cout<<"Two types of bram are used in this operation
            ↪ mode, which is not expected!!!"<<endl;
            cout<<"Something went wrong"<<endl;
```

```
                    exit(1);
                }
            }
        }// all logic ram in this circuit have been mapped

        ↪ logic_circuit_list[circuit.get_circuit_id()].set_circuit_area(circuit_areate
    }
}
```