Eamon Murphy

ECE 720

Finite Impulse Response Filter Accelerator Hardware Software Codesign

INTRODUCTION

The goal of this project was to produce a finite impulse response (FIR) filter accelerator for use in an RISCV SoC using Catapult High Level Synthesis (HLS), SystemC, and Nvidia's Matchlib. By using these resources, hardware and software can theoretically be designed in tandem with the complexity of communicating with the RocketChip tile abstracted away with SystemC's transaction level modeling (TLM). And, instead of describing the accelerator with RTL (and letting a synthesis tool like Synopsys Design Compiler produce the gate-level implementation), Catapult HLS uses the C++ description of this accelerator to produce RTL.

The final design does fulfill the original project goals. It uses the AXI 64-bit crossbar-style bus. It works with only modifications to the FIR Unit's SystemC model and the RISCV program. The redesigned software and accelerator also produce the same output as the original program. And, most importantly, it improves overall system performance. The cycles per instruction (equivalently the total delay) is less than the purely software implementation by around 30%. Additionally, offloading work to the direct-memory-access (DMA) unit and FIR unit allows the processor to coordinate activity on the high level and not waste cycles on repetitive computations and memory access.

Though certainly not as performant as hand-produced RTL, the final accelerator fulfills the project goals and served as a valuable learning experience.

SYSTEM ARCHITECTURE

The base system is shown below:

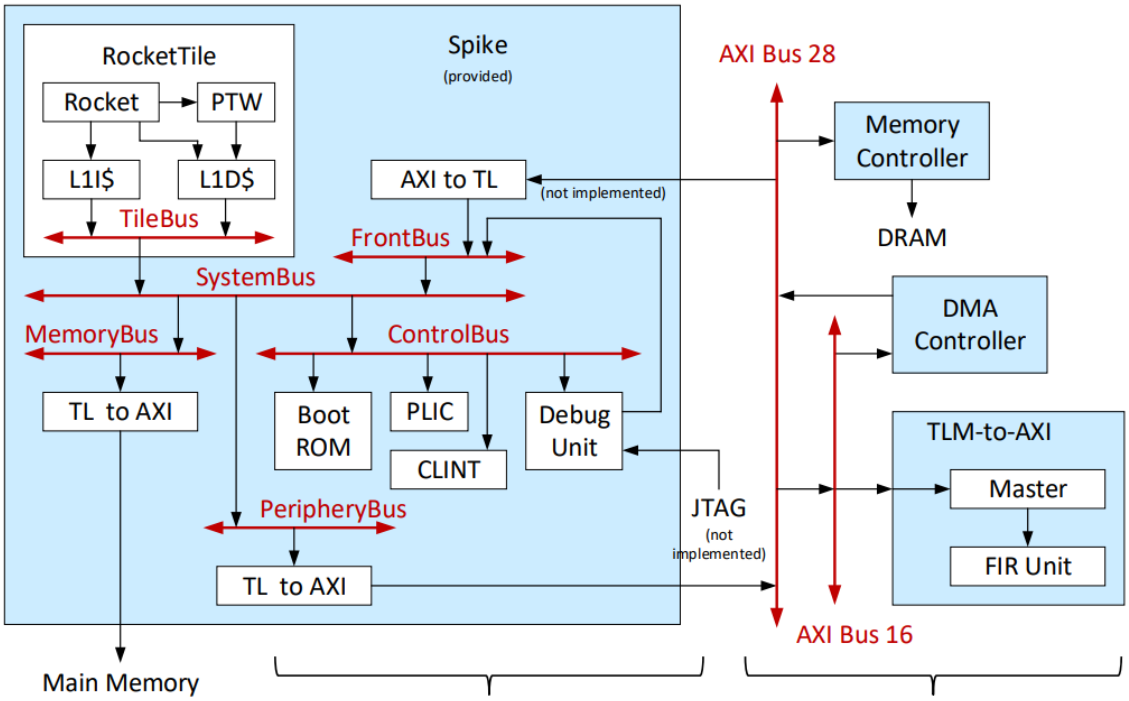

Figure 1: System Architecture (Davis)

Two parts of this base system were changed: the FIR unit, and the RISCV program ran on the Rocketchip tile.

One significant design choice concerned the FIR unit. Most of the delay from the base system is from memory accesses, so a few rare clock cycles of latency are insignificant. Rather, delay comes from slow operations repeated frequently. My initial suspicion was the optimal design for the FIR accelerator from every standpoint would be a pipelined version with 16 PEs (shown below) in series.

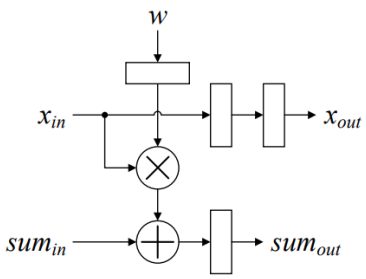

Figure 2: Processing Element (Davis)

With this scheme, a modest amount resources (registers, multipliers, adders) would be produced, so the area and power consumption would likely be lower. However, implementing this with cascaded SystemC modules negated the point of HLS to me. To fully realize the benefit (and fully experience the pain of HLS), I should not structurally describe things applying the same paradigm as

Verilog. My hope was that by framing the module in terms of buffers, loops, and transactions, HLS could produce many possible designs with vastly different topologies. The code that best illustrates this is in the run task of the FIR Unit:

```
        //copy weights
        for(int i = 0; i < 4; i+= 1) {
                weights[i + 0] = regOut_chan[2].read().slc<16>(i*16);
                weights[i + 4] = regOut_chan[3].read().slc<16>(i*16);
                weights[i + 8] = regOut_chan[4].read().slc<16>(i*16);
                weights[i + 12] = regOut_chan[5].read().slc<16>(i*16);
            }
            //shift old inputs
            for(int i = 0; i < 16; i+=1) {
              inputBuffer[i] = inputBuffer[i + 16];
            }
            wait();

            //process new inputs
            for(int i = 0; i < 4; i+= 1) {
              inputBuffer[i + 16] = regOut_chan[6].read().slc<16>(i*16);
              inputBuffer[i + 20] = regOut_chan[7].read().slc<16>(i*16);
              inputBuffer[i + 24] = regOut_chan[8].read().slc<16>(i*16);
              inputBuffer[i + 28] = regOut_chan[9].read().slc<16>(i*16);
            }

            //FIR computation
            for (int n=0; n<32; n++) {
              outputArray[n]=0;
              for (int m=0; m<TAPS; m++) {
                if (n+m-TAPS+1 >= 0) {
                  outputArray[n]+=weights[m]*inputBuffer[n+m-TAPS+1];
                }
              }
            }
```

This is the heart of the FIR Unit. I wanted to write the most algorithmic description possible to see where HLS failed or succeeded. The rest of the FIR unit is either boilerplate initialization code, transactions to send results to the output registers (which required nonintuitive approaches like stalling multiple times), or syntactic sugar.

The program running on the RISCV core is quite simple in comparison. To use the FIR filter, it instructs the DMA to copy weights and inputs to the FIR unit, resets a status bit in the FIR unit, sets a control bit, and then polls the status register for completion. A snippet of code to do this is shown below:

```
  printf("Copying second batch of inputs to FIR unit\n");
 llpp=(volatile long long**)0x70000010; // dma sr
```

```
*llpp=(volatile long long*)(8192+32); // memctl coef1 address
llpp=(volatile long long**)0x70000018; // dma dr
*llpp=(volatile long long*)0x10010030; // fir input
llp=(volatile long long*)0x70000020;   // dma len
*llp=(volatile long long)32; // starts transfer
llp=(volatile long long*)0x70000000;   // dma st
while (*llp);

printf("Starting second round of FIR computation\n");
llp=(volatile long long*)0x70010000;  //reset status register
*llp = 0x07;
llp=(volatile long long*)0x70010008; // fir ctrl
*llp=(volatile long long)0x03;       // Start computation cycle 1
*llp=(volatile long long)0x02;       // Start computation cycle 1
llp=(volatile long long*)0x70010000;  //reset status register
while(*llp != 0x03) {printf("Core waiting for FIR unit\n");};

printf("Copying second batch of FIR outputs to memory\n");
llpp=(volatile long long**)0x70000010; // dma sr
*llpp=(volatile long long*)(0x10010050); // memctl coef1 address
llpp=(volatile long long**)0x70000018; // dma dr
*llpp=(volatile long long*)0x00001020; // fir input
llp=(volatile long long*)0x70000020;   // dma len
*llp=(volatile long long)32; // starts transfer
llp=(volatile long long*)0x70000000;   // dma st
while (*llp);
```

PERFORMANCE AND IMPLEMENTATION

The final implementation synthesized by Catapult from the original SystemC proved to be quite inefficient. As shown below, the area score from HLS was quite high though over half the area comes from the AXI slave:

Solution	Registers	MUX	Functional	Logic	Memory	FSM Reg	FSM Comb	FSM	Datapath	Total Reg	Total Area	Rom
solution.v1 (new)												
firUnit.v1 (extract)	**16018.52**	**6659.24**	**1293.55**	**1243.89**	**3241.37**	**235.00**	**11.07**	**246.07**	**28456.58**	**16253.52**	**28702.64**	**0.00**
allocate	15656.76	7160.43	2787.80	6.38	3241.37	0.00	0.00	0.00	28852.74	15656.76	28852.74	0.00
dpfsm	16018.52	10861.99	2487.07	1322.89	3241.37	235.00	10.00	245.00	33931.84	16253.52	34176.84	0.00
extract	16018.52	6659.24	1293.55	1243.89	3241.37	235.00	11.07	246.07	28456.58	16253.52	28702.64	0.00
slave	12363.68	4844.50	416.31	811.46	0.00	10.00	1.53	11.53	18435.95	12373.68	18447.48	0.00
firUnit:inst	3654.84	1814.74	877.24	432.44	3241.37	225.00	9.53	234.53	10020.63	3879.84	10255.16	0.00

Figure 3: HLS Area Report

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Iterations	Total Cycles
solution.v1 (new)						
firUnit.v1 (extract)	**1728**	**8640.00**	**1728**	**8640.00**		
firUnit						
AxiSlaveToReg2<axi::cfg::standar...						
run	2	10.00	1	5.00		4
run:rlp	2	10.00	1	5.00		4
while	2	10.00	1	5.00		3
firUnit:run						
run	1726	8630.00	1728	8640.00		1793
run:rlp	1726	8630.00	1728	8640.00		1793
inputBuffer:vinit			64	320.00	32.00	64
while	1726	8630.00	1728	8640.00		1728
while:if:if:for			4	20.00	4.00	4
while:if:if:for#1			48	240.00	16.00	48
while:if:if:for#2			20	100.00	4.00	20
while:if:if:for#3			1632	8160.00	32.00	1632
while:if:if:for#3:for			1536	7680.00	16.00	1536

The primary source of throughput cycles is shown as the third for loop in the FIR Unit. This makes sense as the third loop does the actual FIR computation. Other sources of delay include reading and processing inputs and weights, which are insignificant in comparison. Though certainly higher than manually written Verilog, the FIR computation consumes less cycles in the dedicated accelerator than when done in the RISCV core. The full codesigned implementation using the accelerator takes 10027ns, the original takes 14328ns.

Resource usage from the RTL report for the base system and implementation is shown below:

Resource	Base System	Implemented
Mgc_add	9	19
Mgc_and	8	15
Mgc_mul	0	2
Mgc_mux	14	17
Mgc_mux1hot	3	9
Mgc_nand	5	5
Mgc_nor	8	13
Mgc_not	5	5
Mgc_or	5	13
Mgc_reg_pos	12	14
RAM, RAM_rwport_en	0	2

The full logs for the RTL reports are listed in the appendix. Resource counts were tallied by grep and counting the lines.

CONCLUSION

Ultimately, what I learned from this project was more about the limits and use cases of ESL than producing the most performant accelerator. Both the hardware/software codesign and HLS portions in this project illustrated some important points.

First, HLS is great in theory, but the synthesized design quality is difficult to control at that level of abstraction. To use an analogy, the tricky part about writing quality RTL is implying the desired implementation without structurally describing it. However, if the RTL does not produce the desired

implementation, there is always the ability to describe the hardware more granularly. For example, if I were making the FIR accelerator in Verilog, I could control the pipelining by manually placing flip flops at the outputs or inputs of select processing elements. However, when an HLS result was undesirable, the necessary changes in code to fix problems were mercurial. Controlling the level of loop unrolling or pipelining did not always show why HLS produced a result. The fundamental idea of HLS as I understand it is to enable greater productivity by describing systems at a higher level of abstraction. But, in cases where the output is lacking or more granular control is necessary, the core trouble with HLS shows.

Another area I learned about through this project was the value and difficulties of hardware software codesign. Courses like ECE564 showed how to build optimized accelerators with Verilog but integrating them into a full system proved to be an entirely different challenge. System performance was clearly improved by the accelerator, and the RISCV core could orchestrate tasks without wasting time on repetitive memory accesses and MACs. However, sometimes nonintuitive measures like stalling the accelerator between writes to the output registers or wasting cycles in the RISCV program to prevent compiler optimizations removing instructions were necessary to get proper simulation results. Without this codesign environment, properly integrating the accelerator and RocketChip tile would have been far more difficult.

Ultimately, this project showed me a different side of ESL than theoretical discussions or trivial examples. By implementing the accelerator with HLS and doing the codesign, I saw how ESL is different from other methodologies, where it falters, where it works, and how the paradigm differs from RTL.

REFERENCES

WR Davis, S 2019, *Project 2 Requirements,* lecture notes, Electronic System Level and Physical Design ECE 720, NC State University

APPENDIX

Implementation rtl.rpt

Bill Of Materials (Datapath)

Component Name	*Area Score*	*Area(Combinational)*	*AreaSeq*	*Delay*	*Post Alloc*	*Post Assign*
[Lib: VIRTUAL]						
VIRTUAL.while:unequal	*102.209*	*0.000*	*0.000*	*0.449*	*1*	*0*
[Lib: ccs_connections]						
ccs_conn_in_wait(21,44,0,0,0,0)	*0.000*	*0.000*	*0.000*	*0.000*	*1*	*1*
ccs_conn_in_wait(22,44,0,0,0,0)	*0.000*	*0.000*	*0.000*	*0.000*	*1*	*1*
ccs_conn_in_wait(23,71,0,0,0,0)	*0.000*	*0.000*	*0.000*	*0.000*	*1*	*1*
ccs_conn_in_wait(25,73,0,0,0,0)	*0.000*	*0.000*	*0.000*	*0.000*	*1*	*1*

ccs_conn_out_wait(24,71,0,0,0,0) *0.000* *0.000* *0.000* *0.000* *1* *1*

ccs_conn_out_wait(26,6,0,0,0,0) *0.000* *0.000* *0.000* *0.000* *1* *1*

ccs_conn_out_wait(35,71,0,0,0,0) *0.000* *0.000* *0.000* *0.000* *1* *1*

[Lib: nangate-45nm_beh]

mgc_add(13,0,1,0,13,7) *36.176* *0.000* *0.000* *0.571* *0* *1*

mgc_add(13,0,2,1,13,4) *102.857* *11.943* *0.000* *0.383* *1* *0*

mgc_add(16,0,16,0,16,4) *119.740* *26.940* *0.000* *0.428* *1* *0*

mgc_add(16,0,16,0,16,7) *68.096* *0.000* *0.000* *1.216* *0* *1*

mgc_add(16,0,8,1,16,4) *123.415* *46.669* *0.000* *0.436* *1* *0*

mgc_add(17,0,16,0,17,4) *121.677* *31.604* *0.000* *0.437* *2* *0*

mgc_add(17,0,16,0,17,7) *70.756* *0.000* *0.000* *1.251* *0* *4*

mgc_add(4,0,1,0,5,7) *12.236* *0.000* *0.000* *0.258* *0* *1*

mgc_add(4,0,2,1,5,4) *35.601* *32.034* *0.000* *0.211* *1* *0*

mgc_add(5,0,2,1,5,3) *45.265* *0.000* *0.000* *0.211* *1* *0*

mgc_add(5,0,2,1,6,3) *45.265* *55.636* *0.000* *0.211* *1* *0*

mgc_add(5,0,4,0,5,4) *33.730* *0.000* *0.000* *0.219* *1* *0*

mgc_add(5,0,4,0,5,7) *19.684* *0.000* *0.000* *0.390* *0* *1*

mgc_add(5,0,5,1,6,2) *61.493* *79.468* *0.000* *0.196* *1* *0*

mgc_add(5,0,5,1,6,7) *22.947* *4.153* *0.000* *0.458* *0* *1*

mgc_add(7,0,7,0,7,3) *68.558* *0.000* *0.000* *0.246* *1* *0*

mgc_add(7,0,7,0,7,7) *29.792* *0.000* *0.000* *0.567* *0* *2*

mgc_add(8,0,1,1,8,6) *52.513* *0.000* *0.000* *0.347* *1* *0*

mgc_add(8,0,1,1,8,7) *37.524* *0.000* *0.000* *0.676* *0* *1*

mgc_and(1,2,4) *1.064* *0.000* *0.000* *0.070* *0* *196*

mgc_and(1,3,4) *1.330* *0.334* *0.000* *0.078* *0* *28*

mgc_and(1,33,4) *14.896* *15.710* *0.000* *0.331* *0* *1*

mgc_and(1,4,4) *1.596* *0.979* *0.000* *0.087* *0* *30*

mgc_and(1,5,4) *2.062* *1.582* *0.000* *0.095* *0* *19*

mgc_and(1,6,4) *2.527* *2.159* *0.000* *0.104* *0* *6*

mgc_and(1,7,4) *2.993* *2.719* *0.000* *0.112* *0* *2*

mgc_and(1,8,4) *3.458* *3.265* *0.000* *0.121* *0* *1*

mgc_and(1,9,4) *3.857* *3.802* *0.000* *0.129* *0* *2*

mgc_and(16,2,4) *17.024* *21.699* *0.000* *0.072* *0* *5*

mgc_and(2,2,4) *2.128* *0.913* *0.000* *0.070* *2* *2*

mgc_and(3,2,4) *3.192* *2.236* *0.000* *0.070* *0* *1*

mgc_and(4,2,4) *4.256* *3.584* *0.000* *0.071* *0* *1*

mgc_and(5,2,4) *5.320* *4.957* *0.000* *0.071* *0* *1*

mgc_and(64,2,4) *68.096* *130.031* *0.000* *0.080* *0* *1*

mgc_equal(65,4) *135.926* *167.034* *0.000* *0.450* *0* *1*

mgc_mul(16,0,16,0,16,4) *800.731* *801.511* *0.000* *0.782* *1* *0*

mgc_mul(16,0,16,0,16,7) *618.353* *753.300* *0.000* *1.433* *0* *1*

mgc_mux(1,1,2,2)	*0.000*	*24.647*	*0.000 0.090*	*0*	*219*
mgc_mux(1,1,2,3)	*0.000*	*4.079*	*0.000 0.082*	*19*	*4*
mgc_mux(13,1,2,5)	*24.206*	*13.128*	*0.000 0.140*	*4*	*6*
mgc_mux(16,1,2,5)	*29.792*	*21.368*	*0.000 0.162*	*16*	*32*
mgc_mux(16,2,4,5)	*59.318*	*60.557*	*0.000 0.287*	*4*	*4*
mgc_mux(16,4,16,5)	*245.784*	*312.151*	*0.000 0.376*	*1*	*1*
mgc_mux(2,1,2,3)	*3.325*	*11.479*	*0.000 0.088*	*1*	*2*
mgc_mux(20,1,2,5)	*37.240*	*31.955*	*0.000 0.154*	*0*	*1*
mgc_mux(3,1,2,5)	*5.586*	*0.000*	*0.000 0.090*	*2*	*2*
mgc_mux(4,1,2,5)	*7.448*	*0.000*	*0.000 0.098*	*3*	*6*
mgc_mux(44,1,2,5)	*81.928*	*91.370*	*0.000 0.158*	*0*	*1*
mgc_mux(5,1,2,5)	*9.310*	*0.000*	*0.000 0.106*	*0*	*1*
mgc_mux(64,1,2,5)	*119.168*	*138.778*	*0.000 0.163*	*11*	*16*
mgc_mux(64,4,16,5)	*1086.400*	*1207.005*	*0.000 0.407*	*2*	*2*
mgc_mux(68,1,2,5)	*126.616*	*148.153*	*0.000 0.164*	*0*	*1*
mgc_mux(73,1,2,5)	*135.926*	*159.835*	*0.000 0.165*	*0*	*1*
mgc_mux(8,1,2,5)	*14.896*	*0.000*	*0.000 0.130*	*2*	*11*
mgc_mux1hot(1,3,1)	*0.000*	*12.622*	*0.000 0.123*	*0*	*6*
mgc_mux1hot(1,4,4)	*3.458*	*0.000*	*0.000 0.152*	*0*	*1*
mgc_mux1hot(16,17,4)	*238.424*	*281.510*	*0.000 0.216*	*0*	*1*
mgc_mux1hot(16,3,4)	*42.560*	*41.940*	*0.000 0.139*	*0*	*1*
mgc_mux1hot(2,4,4)	*6.916*	*0.000*	*0.000 0.152*	*0*	*1*
mgc_mux1hot(2,5,4)	*8.512*	*0.000*	*0.000 0.161*	*0*	*1*
mgc_mux1hot(3,4,4)	*10.374*	*0.000*	*0.000 0.152*	*0*	*1*
mgc_mux1hot(5,16,4)	*70.490*	*78.515*	*0.000 0.213*	*0*	*1*
mgc_mux1hot(64,3,4)	*170.240*	*226.799*	*0.000 0.139*	*15*	*0*
mgc_nand(1,2,4)	*0.798*	*0.000*	*0.000 0.054*	*0*	*63*
mgc_nand(1,3,4)	*1.064*	*0.191*	*0.000 0.061*	*0*	*4*
mgc_nand(1,4,4)	*1.330*	*0.932*	*0.000 0.069*	*0*	*3*
mgc_nand(1,5,4)	*1.929*	*1.605*	*0.000 0.076*	*0*	*1*
mgc_nand(3,2,4)	*2.394*	*1.587*	*0.000 0.054*	*0*	*1*
mgc_nor(1,13,4)	*5.719*	*7.504*	*0.000 0.184*	*0*	*1*
mgc_nor(1,15,4)	*6.517*	*8.399*	*0.000 0.196*	*0*	*1*
mgc_nor(1,16,4)	*6.916*	*8.783*	*0.000 0.202*	*0*	*1*
mgc_nor(1,2,2)	*0.798*	*0.000*	*0.000 0.109*	*0*	*102*
mgc_nor(1,3,2)	*1.596*	*3.412*	*0.000 0.112*	*0*	*18*
mgc_nor(1,4,4)	*2.134*	*0.000*	*0.000 0.129*	*0*	*6*
mgc_nor(1,5,4)	*1.929*	*0.451*	*0.000 0.135*	*0*	*4*
mgc_nor(1,6,4)	*2.527*	*1.903*	*0.000 0.141*	*0*	*1*
mgc_nor(1,63,1)	*6.149*	*24.452*	*0.000 0.231*	*0*	*1*
mgc_nor(1,7,4)	*3.126*	*3.099*	*0.000 0.147*	*0*	*1*

mgc_nor(1,8,4)	3.724	4.109	0.000	0.153	0	2
mgc_nor(1,9,4)	4.123	4.975	0.000	0.159	0	3
mgc_nor(3,2,2)	2.394	1.538	0.000	0.109	0	1
mgc_not(1,1)	0.532	0.532	0.000	0.039	0	262
mgc_not(13,1)	6.916	10.463	0.000	0.039	0	2
mgc_not(16,1)	8.512	15.605	0.000	0.039	0	2
mgc_not(3,1)	1.596	1.005	0.000	0.039	0	1
mgc_not(7,1)	3.724	3.369	0.000	0.039	0	2
mgc_or(1,13,4)	5.453	7.404	0.000	0.154	0	1
mgc_or(1,14,4)	5.852	8.059	0.000	0.158	0	1
mgc_or(1,2,2)	1.064	2.503	0.000	0.086	0	150
mgc_or(1,3,2)	2.261	5.820	0.000	0.089	0	24
mgc_or(1,4,4)	2.528	0.000	0.000	0.115	0	5
mgc_or(1,5,4)	2.062	0.000	0.000	0.120	0	4
mgc_or(1,6,4)	2.527	0.772	0.000	0.124	0	1
mgc_or(1,64,3)	0.000	29.001	0.000	0.316	0	2
mgc_or(1,7,4)	2.993	2.070	0.000	0.128	0	1
mgc_or(1,8,4)	3.458	3.203	0.000	0.133	0	1
mgc_or(2,2,2)	2.128	3.726	0.000	0.086	0	1
mgc_or(2,2,3)	2.128	0.000	0.000	0.085	1	0
mgc_or(5,2,1)	5.257	10.858	0.000	0.084	0	2
mgc_reg_pos(1,1,0,0,0,0,0,1)	5.320	0.000	5.320	0.122	0	13
mgc_reg_pos(1,1,0,0,0,1,1,1)	5.320	0.000	5.320	0.122	0	51
mgc_reg_pos(13,1,0,0,0,1,1,1)	69.160	0.000	69.160	0.122	0	2
mgc_reg_pos(16,1,0,0,0,1,1,1)	85.120	0.000	85.120	0.122	0	34
mgc_reg_pos(2,1,0,0,0,1,1,1)	10.640	0.000	10.640	0.122	0	1
mgc_reg_pos(20,1,0,0,0,1,1,1)	106.400	0.000	106.400	0.122	0	1
mgc_reg_pos(3,1,0,0,0,1,1,1)	15.960	0.000	15.960	0.122	0	4
mgc_reg_pos(4,1,0,0,0,1,1,1)	21.280	0.000	21.280	0.122	0	7
mgc_reg_pos(44,1,0,0,0,1,1,1)	234.080	0.000	234.080	0.122	0	1
mgc_reg_pos(5,1,0,0,0,1,1,1)	26.600	0.000	26.600	0.122	0	2
mgc_reg_pos(64,1,0,0,0,1,1,1)	340.480	0.000	340.480	0.122	0	33
mgc_reg_pos(68,1,0,0,0,1,1,1)	361.760	0.000	361.760	0.122	0	1
mgc_reg_pos(73,1,0,0,0,1,1,1)	388.360	0.000	388.360	0.122	0	1
mgc_reg_pos(8,1,0,0,0,1,1,1)	42.560	0.000	42.560	0.122	0	1
mgc_shift_r(64,0,6,16,2)	238.267	333.649	0.000	0.505	4	0
[Lib: ram_nangate-45nm-singleport_beh]						
RAM(32,32,16,5,0,0,0,0,0,1,1,32,16,1)	1620.685	0.000	0.000	0.900	0	1
RAM(34,32,16,5,0,0,0,0,0,1,1,32,16,1)	1620.685	0.000	0.000	0.900	0	1
RAM_rwport_en(32,32,16,5,0,0,0,0,0,1,1,32,16,1)	1620.685	0.000	0.000	0.900	1	0
RAM_rwport_en(34,32,16,5,0,0,0,0,0,1,1,32,16,1)	1620.685	0.000	0.000	0.900	1	0

TOTAL AREA (After Assignment): 28456.577 14555.000 16019.000

Base system rtl.rpt:

Bill Of Materials (Datapath)

Component Name	Area Score	Area(Combinational)	AreaSeq	Delay	Post Alloc	Post Assign
[Lib: ccs_connections]						
ccs_conn_in_wait(21,44,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
ccs_conn_in_wait(22,44,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
ccs_conn_in_wait(23,71,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
ccs_conn_in_wait(25,73,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
ccs_conn_out_wait(24,71,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
ccs_conn_out_wait(26,6,0,0,0,0)	0.000	0.000	0.000	0.000	1	1
[Lib: nangate-45nm_beh]						
mgc_add(13,0,1,0,13,7)	36.176	0.000	0.000	0.571	0	1
mgc_add(13,0,2,1,13,4)	102.857	11.943	0.000	0.383	1	0
mgc_add(16,0,8,1,16,4)	123.415	46.669	0.000	0.436	1	0
mgc_add(17,0,16,0,17,4)	121.677	31.604	0.000	0.437	2	0
mgc_add(17,0,16,0,17,7)	70.756	0.000	0.000	1.251	0	4
mgc_add(7,0,7,0,7,3)	68.558	0.000	0.000	0.246	1	0
mgc_add(7,0,7,0,7,7)	29.792	0.000	0.000	0.567	0	2
mgc_add(8,0,1,1,8,6)	52.513	0.000	0.000	0.347	1	0
mgc_add(8,0,1,1,8,7)	37.524	0.000	0.000	0.676	0	1
mgc_and(1,2,4)	1.064	0.000	0.000	0.070	0	156
mgc_and(1,3,4)	1.330	0.334	0.000	0.078	0	27
mgc_and(1,4,4)	1.596	0.979	0.000	0.087	0	30
mgc_and(1,5,4)	2.062	1.582	0.000	0.095	0	19
mgc_and(1,6,4)	2.527	2.159	0.000	0.104	0	6
mgc_and(1,7,4)	2.993	2.719	0.000	0.112	0	2
mgc_and(1,9,4)	3.857	3.802	0.000	0.129	0	2
mgc_and(2,2,4)	2.128	0.913	0.000	0.070	2	0
mgc_mux(1,1,2,2)	0.000	24.647	0.000	0.090	0	217
mgc_mux(1,1,2,3)	0.000	4.079	0.000	0.082	17	4
mgc_mux(13,1,2,5)	24.206	13.128	0.000	0.140	4	6
mgc_mux(16,1,2,5)	29.792	21.368	0.000	0.162	0	1
mgc_mux(2,1,2,3)	3.325	11.479	0.000	0.088	1	0
mgc_mux(20,1,2,5)	37.240	31.955	0.000	0.154	0	1
mgc_mux(3,1,2,5)	5.586	0.000	0.000	0.090	2	2
mgc_mux(4,1,2,5)	7.448	0.000	0.000	0.098	3	4

mgc_mux(44,1,2,5)	81.928	91.370	0.000	0.158	0	1
mgc_mux(64,1,2,5)	119.168	138.778	0.000	0.163	10	16
mgc_mux(64,4,16,5)	1086.400	1207.005	0.000	0.407	2	2
mgc_mux(68,1,2,5)	126.616	148.153	0.000	0.164	0	1
mgc_mux(73,1,2,5)	135.926	159.835	0.000	0.165	0	1
mgc_mux(8,1,2,5)	14.896	0.000	0.000	0.130	2	11
mgc_mux1hot(1,3,1)	0.000	12.622	0.000	0.123	0	6
mgc_mux1hot(1,4,4)	3.458	0.000	0.000	0.152	0	1
mgc_mux1hot(64,3,4)	170.240	226.799	0.000	0.139	15	0
mgc_nand(1,2,4)	0.798	0.000	0.000	0.054	0	57
mgc_nand(1,3,4)	1.064	0.191	0.000	0.061	0	4
mgc_nand(1,4,4)	1.330	0.932	0.000	0.069	0	3
mgc_nand(1,5,4)	1.929	1.605	0.000	0.076	0	1
mgc_nand(3,2,4)	2.394	1.587	0.000	0.054	0	1
mgc_nor(1,15,4)	6.517	8.399	0.000	0.196	0	1
mgc_nor(1,2,2)	0.798	0.000	0.000	0.109	0	52
mgc_nor(1,3,2)	1.596	3.412	0.000	0.112	0	9
mgc_nor(1,4,4)	2.134	0.000	0.000	0.129	0	4
mgc_nor(1,5,4)	1.929	0.451	0.000	0.135	0	1
mgc_nor(1,7,4)	3.126	3.099	0.000	0.147	0	1
mgc_nor(1,8,4)	3.724	4.109	0.000	0.153	0	2
mgc_nor(1,9,4)	4.123	4.975	0.000	0.159	0	3
mgc_not(1,1)	0.532	0.532	0.000	0.039	0	235
mgc_not(13,1)	6.916	10.463	0.000	0.039	0	2
mgc_not(16,1)	8.512	15.605	0.000	0.039	0	2
mgc_not(3,1)	1.596	1.005	0.000	0.039	0	1
mgc_not(7,1)	3.724	3.369	0.000	0.039	0	2
mgc_or(1,2,2)	1.064	2.503	0.000	0.086	0	116
mgc_or(1,3,2)	2.261	5.820	0.000	0.089	0	19
mgc_or(1,4,4)	2.528	0.000	0.000	0.115	0	2
mgc_or(1,8,4)	3.458	3.203	0.000	0.133	0	1
mgc_or(2,2,3)	2.128	0.000	0.000	0.085	1	0
mgc_reg_pos(1,1,0,0,0,0,0,1)	5.320	0.000	5.320	0.122	0	9
mgc_reg_pos(1,1,0,0,0,1,1,1)	5.320	0.000	5.320	0.122	0	46
mgc_reg_pos(13,1,0,0,0,1,1,1)	69.160	0.000	69.160	0.122	0	2
mgc_reg_pos(16,1,0,0,0,1,1,1)	85.120	0.000	85.120	0.122	0	1
mgc_reg_pos(20,1,0,0,0,1,1,1)	106.400	0.000	106.400	0.122	0	1
mgc_reg_pos(3,1,0,0,0,1,1,1)	15.960	0.000	15.960	0.122	0	2
mgc_reg_pos(4,1,0,0,0,1,1,1)	21.280	0.000	21.280	0.122	0	6
mgc_reg_pos(44,1,0,0,0,1,1,1)	234.080	0.000	234.080	0.122	0	1
mgc_reg_pos(64,1,0,0,0,1,1,1)	340.480	0.000	340.480	0.122	0	31

mgc_reg_pos(68,1,0,0,0,1,1,1)	*361.760*	*0.000*	*361.760*	*0.122*	*0*	*1*
mgc_reg_pos(73,1,0,0,0,1,1,1)	*388.360*	*0.000*	*388.360*	*0.122*	*0*	*1*
mgc_reg_pos(8,1,0,0,0,1,1,1)	*42.560*	*0.000*	*42.560*	*0.122*	*0*	*1*
TOTAL AREA (After Assignment):	*18435.950*	*11362.000*	*12364.000*			