Presentation Format for Final Exam
(Embedded Systems)

Eunseo Ko

IT Engineering
Sookmyung Women's University

Outline

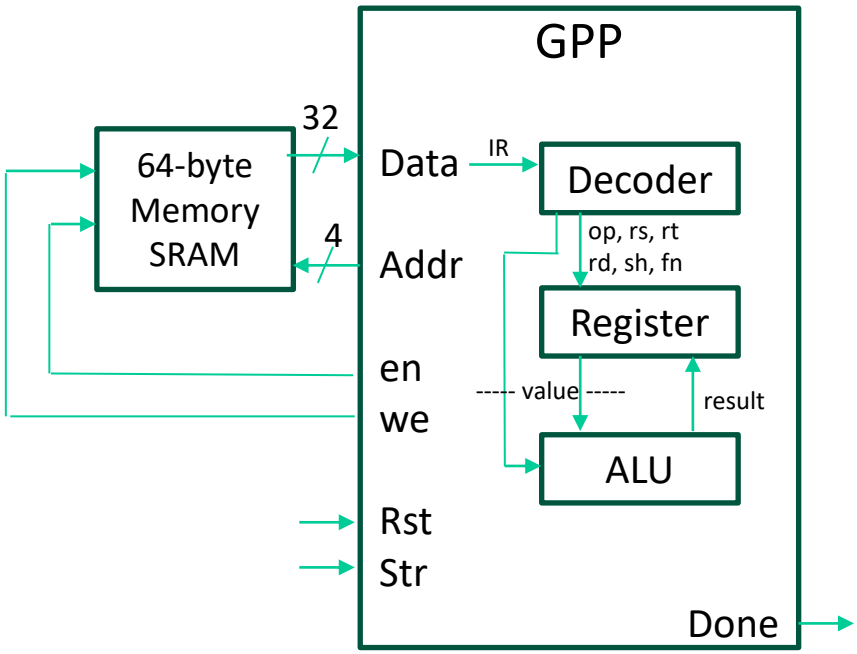
- About My GPP
- Block Diagram and HLSM for RTL Design of GPP
- Two-Procedure RTL Description of the Algorithm in Verilog
- Specification of Components from Xilinx CORE Generator
- Hierarchy of Verilog Files
- Test-Vector Generator
- Verilog-Testbench for Simulating RTL Design of the Algorithm
- Modelsim Simulation
- Xilinx ISE – RTL Synthesis by XST
- Hardware Performance Evaluation
- Attempt Review
- Improvements

About My GPP

- GPP (General Purpose Processor)
  - CPU (Central Processing Unit)
  - I-Cache, Decoder, ALU, Register 포함
  - +, -, *, /, <<, >> 연산 가능

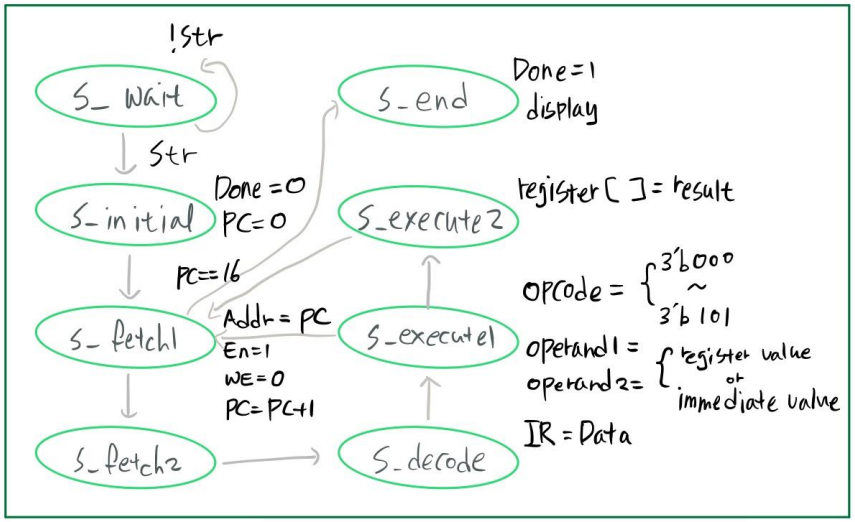
- Why I select GPP?
  - 컴퓨터의 필수 구성 요소
  - 임베디드 개발자 희망 : 하드웨어에 대한 이해 필요
  - GPP를 직접 만들며 컴퓨터 시스템의 구성을 직접 이해하고 싶음
  - 교수님의 허락을 맡아 하드웨어 가속기가 아닌 범용 프로세서 제작

Block Diagram and HLSM for RTL Design of GPP



Block Diagram for GPP algorithm

Inputs : Data (64 byte Memory), Str
Outputs : en, we, Done
Local registers : register (128 byte Memory),
IR (32 bit), op, rs, rt, rd, sh, fn,
PC, result



HLSM for GPP algorithm

Two-Procedure RTL Description of the Algorithm in Verilog

GPP

```
`include "define.h"

module GPP (Addr, Data, RW, En, Done, Clk, Rst, Str);
```

프로세서 동작/종료 제어

```
    input       Clk, Rst, Str;
    integer     I;

    // SRAM
    output reg  [(`SA_WIDTH-1):0]   Addr;
    input       [(`D_WIDTH-1):0]    Data;
    output reg  RW, En, Done;
```

I-Cache 구성 요소

```
    // register
    reg [(`D_WIDTH-1):0] regi [0:25];
```

Register

```
    parameter   S_wait      = 0,
                S_initial   = 1,
                S_fetch1    = 2,
                S_fetch2    = 3,
                S_decode    = 4,
                S_execute1  = 5,
                S_execute2  = 6,
                S_end       = 7;

    reg         [3:0] State, StateNext;
    reg         [(`D_WIDTH-1):0] IR;
    integer     PC;
```

프로세서 상태 및 명령어 구성 요소

```
    // im = rd+sh+fn
    wire [5:0] op;
    wire [4:0] rs;
    wire [4:0] rt;
    wire [4:0] rd;
    wire [4:0] sh;
    wire [5:0] fn;

    Decoder decoder (IR, op, rs, rt, rd, sh, fn);
```

Mips 명령어 분리

```
    reg     [2:0] op_code;
    reg     [(`D_WIDTH-1):0] operand1, operand2;
    wire    [(`D_WIDTH-1):0] result;
    reg     enable;

ALU alu (op_code, operand1, operand2, result, enable);
```

피연산자 지정 및 연산

```
// StateReg
always @(posedge Clk) begin
    if (Rst == 1)
        State <= S_wait;
    else
        State <= StateNext;
end
```

다음 동작 지정 프로시저

```
// ComLogic
always @(State) begin
    case(State)
        S_wait: begin
            if (Str)
                StateNext <= S_initial;
            else
                StateNext <= S_wait;
        end
        S_initial: begin
            Done <= 1'b0;
            PC <= 0;
            regi[0] <= 0;
            StateNext <= S_fetch1;
        end
        S_fetch1: begin
            if (PC==`SL_WIDTH)
                StateNext <= S_end;
            else begin
                Addr <= {`SA_WIDTH{1'b0}};
                RW <= 1'b0;
                En <= 1'b0;

                op_code <= 3'b000;
                operand1 <= {`D_WIDTH{1'b0}};
                operand2 <= {`D_WIDTH{1'b0}};
                enable <= 1'b0;
                Addr <= PC;
                RW <= 1'b0;
                En <= 1'b1;
                PC <= PC+1;

                StateNext <= S_fetch2;
            end
        end
        S_fetch2: begin
            StateNext <= S_decode;
        end
        S_decode: begin
            IR <= Data;
            StateNext <= S_execute1;
        end
```

상태 별 행위 프로시저

Str(시작)을 기다리는 상태

GPP 연산이 시작되어 기초 변수 초기화

I-Cache의 내용을 다 읽을 때까지 진행, I-Cache에서 한 줄 씩 가져오기

Fetch를 위한 여분 State

Instruction에 저장

Decoder 가 명령어 분리

Two-Procedure RTL Description of the Algorithm in Verilog

```
97              S_execute1: begin
98                  case(op)
99                      0: begin
100                 case(fn)
101                     0: begin    // sll
102                         op_code <= 3'b100;
103                         operand1 <= regi[rt];
104                         operand2 <= sh;
105                     end
106                     2: begin    // srl
107                         op_code <= 3'b101;
108                         operand1 <= regi[rt];
109                         operand2 <= sh;
110                     end
111                     24: begin   // mult
112                         op_code <= 3'b010;
113                         operand1 <= regi[rs];
114                         operand2 <= regi[rt];
115                     end
116                     26: begin   // div
117                         op_code <= 3'b011;
118                         operand1 <= regi[rs];
119                         operand2 <= regi[rt];
120                     end
121                     32: begin   // add
122                         op_code <= 3'b000;
123                         operand1 <= regi[rs];
124                         operand2 <= regi[rt];
125                     end
126                     34: begin   // sub
127                         op_code <= 3'b001;
128                         operand1 <= regi[rs];
129                         operand2 <= regi[rt];
130                     end
131                 endcase
132                     end
133                 8: begin    // addi
134                     op_code <= 3'b000;
135                     operand1 <= regi[rs];
136                     operand2 <= {rd,sh,fn};
137                 end
138             endcase
```

실행1 : ALU 연산

연산을 위한 opcode와
피연산자 지정

```
140                 enable <= 1'b1;
141                 StateNext <= S_execute2;
142             end
143             S_execute2: begin
144                 case(op)
145                     0: regi[rd] <= result;
146                     8: regi[rt] <= result;
147                 endcase
148
149                 StateNext <= S_fetch1;
150             end
151             S_end : begin
152                 $write("regi : ");
153                 for (I=0; I<25; I=I+1) begin
154                     $write("%2d, ", regi[I]);
155                 end
156                 $display("%2d", regi[25]);
157
158                 Done <= 1;
159                 StateNext <= S_wait;
160             end
161         endcase
162     end
163
164 endmodule
```

Enable로 연산 시작

실행2 : 연산 결과 레지스터에 저장

모든 명령어 처리가 끝나면
레지스터를 출력하고 종료

Two-Procedure RTL Description of the Algorithm in Verilog

Decoder

```
1   `timescale 1ns/1ns
2
3   module Decoder(IC, op, rs, rt, rd, sh, fn);
4
5       input [31:0] IC;
6       output reg [5:0] op;
7       output reg [4:0] rs;
8       output reg [4:0] rt;
9       output reg [4:0] rd;
10      output reg [4:0] sh;
11      output reg [5:0] fn;
12
13      always @(IC) begin
14          {op, rs, rt, rd, sh, fn} <= IC;
15      end
16
17  endmodule
```

명령어 분리

ALU

```
1   `include "define.h"
2
3   module ALU(op_code, operand1, operand2, result, enable);
4
5       input [2:0] op_code;
6       input [(`D_WIDTH-1):0] operand1, operand2;
7       output reg [(`D_WIDTH-1):0] result;
8       input enable;
9
10      always @* begin
11          if (enable) begin
12              case(op_code)
13                  0: result <= operand1 + operand2;   // add, addi
14                  1: result <= operand1 - operand2;   // sub
15                  2: result <= operand1 * operand2;   // mult
16                  3: begin                            // div
17                      case (operand2[2:0])
18                          3'b100: result <= operand1 >> 2;
19                          3'b010: result <= operand1 >> 1;
20                      endcase
21                  end
22                  4: result <= operand1 << operand2;  // sll
23                  5: result <= operand1 >> operand2;  // srl
24              endcase
25          end
26      end
27
28  endmodule
```

Opcode에 따라 각 연산 진행

Specification of Components from Xilinx CORE Generator

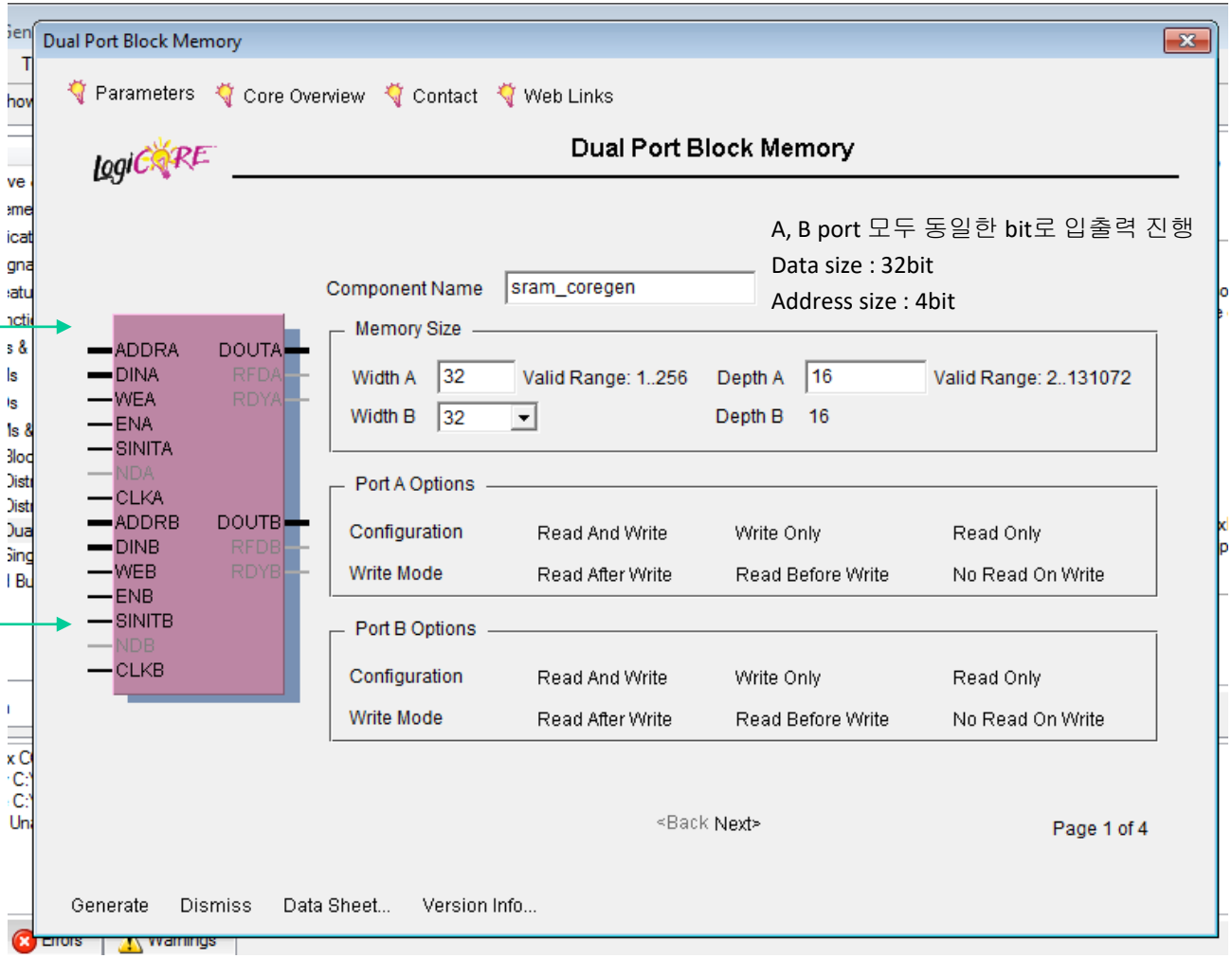
Interface for GPP

4-bit

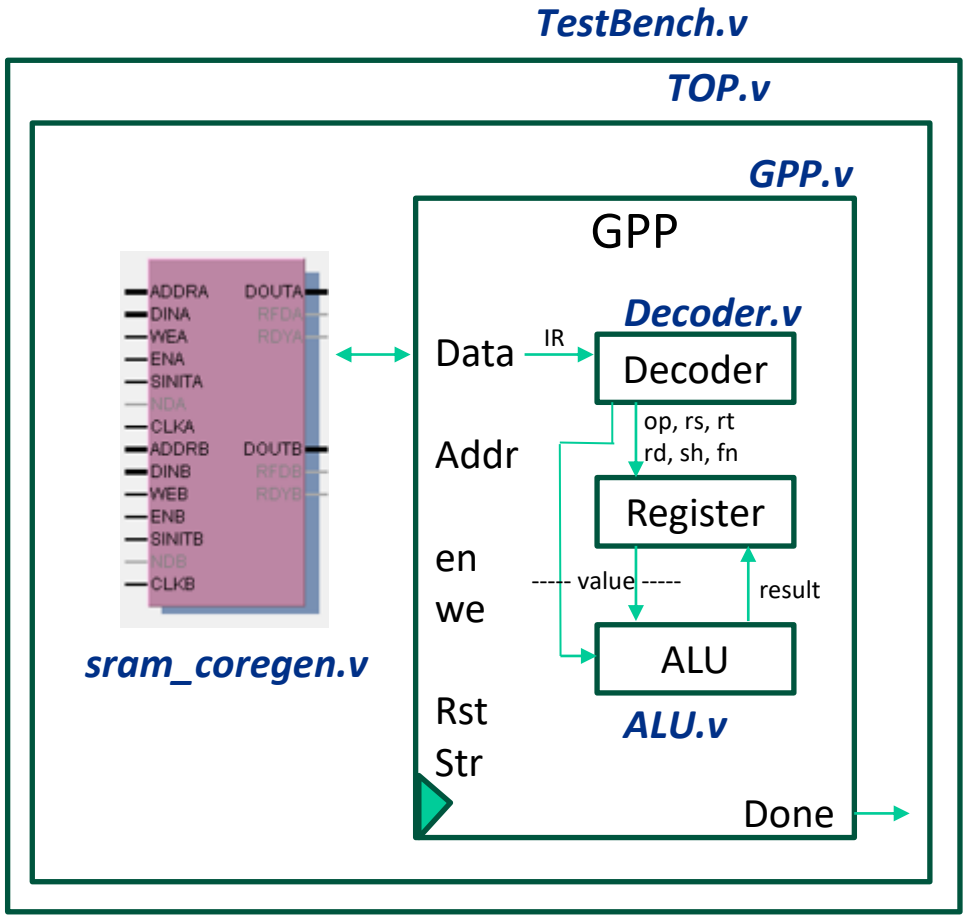
Sharing data

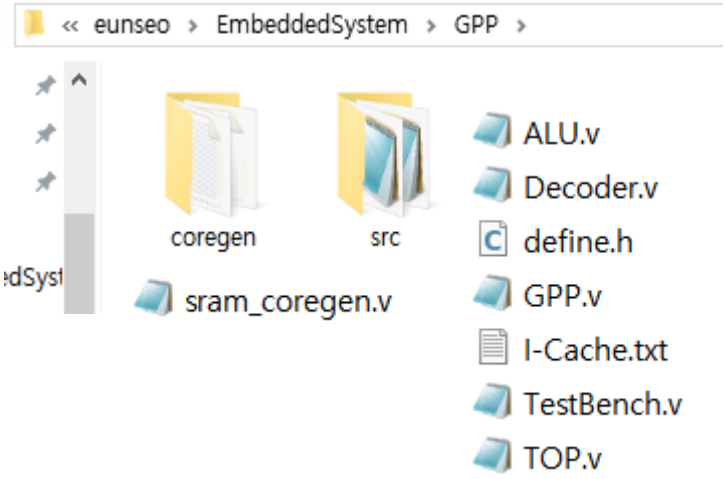
4-bit

Interface for Testbench

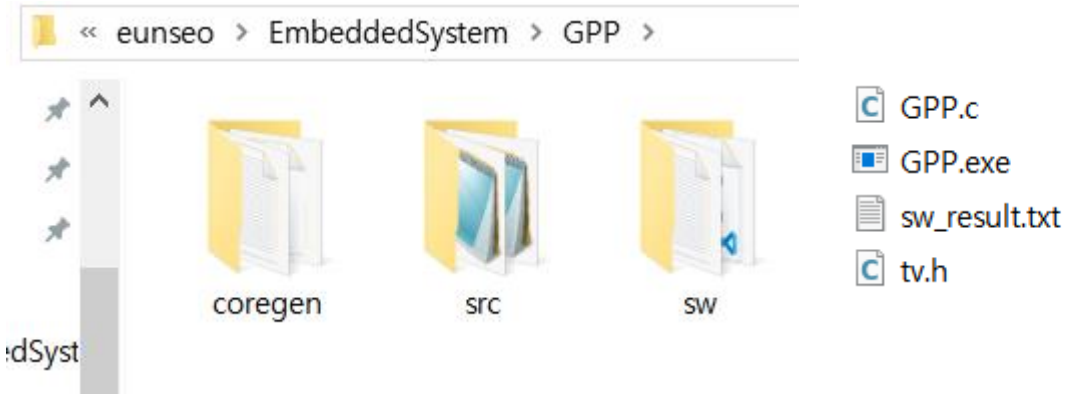


Hierarchy of Verilog Files





Test-Vector Generator



- GPP.exe (source code : GPP.c)
  - C-program for GPP algorithm generating sw_result.txt
- No tvgen.exe
  - 난수가 아닌 특정 계산을 위한 명령어가 필요하기 때문
  - 직접 t레지스터, s레지스터의 목적을 구분하여 16개 명령어 생성

Test-Vector Generator

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "tv.h"

int main(void){

    // 결과 레지스터 파일
    FILE *fp;
    int i, length=sizeof(IC)/sizeof(IC[0]);

    uint32_t regi[32] = {0,}; register 파일
    uint32_t op, rs, rt, rd, sh, fn, im;

    fp = fopen("sw/sw_result.txt", "w");
        if(fp==NULL){
                printf("error occurs when opening sw_result.txt!\n", i);
                exit(1);
        }

    for (i=0; i<length; i++){
        // opcode 추출, 1111 1100 0000 0000 0000 0000 0000 0000
        op = (IC[i] & 0xFC000000) >> 26;

        // rs 추출, 0000 0011 1110 0000 0000 0000 0000 0000
        rs = (IC[i] & 0x3E00000) >> 21;

        // rt 추출, 0000 0000 0001 1111 0000 0000 0000 0000
        rt = (IC[i] & 0x1F0000) >> 16;

        // immediate 추출, 0000 0000 0000 0000 1111 1111 1111 1111
        im = IC[i] & 0xFFFF;
```

결과 파일 생성
에러 시 프로그램 종료

IC에서 op, rs, rt, im 추출

C: > eunseo > EmbeddedSystem > GPP > sw > C tv.h > ...

```c
#include <stdint.h>

uint32_t IC[16] = {
    0x20080001,
    0x20090003,
    0x01098020,
    0x200A0002,
    0x020A8818,
    0x0230881a,
    0x02118022,
    0x001188C0,
    0x00119082,
    0x20130007,
    0x200B0004,
    0x026BA018,
    0x0013A882,
    0x02B1B018,
    0x200C0004,
    0x02CCA822
};
```

Test-Vector Generator

```
37          switch (op){
38           case 0:
39              // rd 추출, 0000 0000 0000 0000 1111 1000 0000 0000
40              rd = (IC[i] & 0xF800) >> 11;
41
42              // funct 추출, 0000 0000 0000 0000 0000 0000 0011 1111
43              fn = IC[i] & 0x3F;
44
45              switch (fn){
46
47              case 0: //0x00 shifh left
48                  // shamt 추출
49                  sh = (IC[i] & 0x7C0) >> 6; // 0000 0000 0000 0000 0000 0111 1100
50                  regi[rd] = regi[rt] << sh;
51                  printf("Shift Left $(%d), $(%d), %d : %d\n", rd, rt, sh, regi[rd
52                  break;
53
54              case 2: //0x02 shift right
55                  sh = (IC[i] & 0x7C0) >> 6; // 0000 0000 0000 0000 0000 0111 1100
56                  regi[rd] = regi[rt] >> sh;
57                  printf("Shift Right $(%d), $(%d), %d : %d\n", rd, rt, sh, regi[r
58                  break;
59
60              case 24: //0x18
61                  regi[rd] = regi[rs] * regi[rt];
62                  printf("Multiply $(%d), $(%d), $(%d) : %d\n", rd, rs, rt, regi[r
63                  break;
64
65              case 26: //0x1a
66                  regi[rd] = regi[rs] / regi[rt];
67                  printf("Divide $(%d), $(%d), $(%d) : %d\n", rd, rs, rt, regi[rd]
68                  break;
```

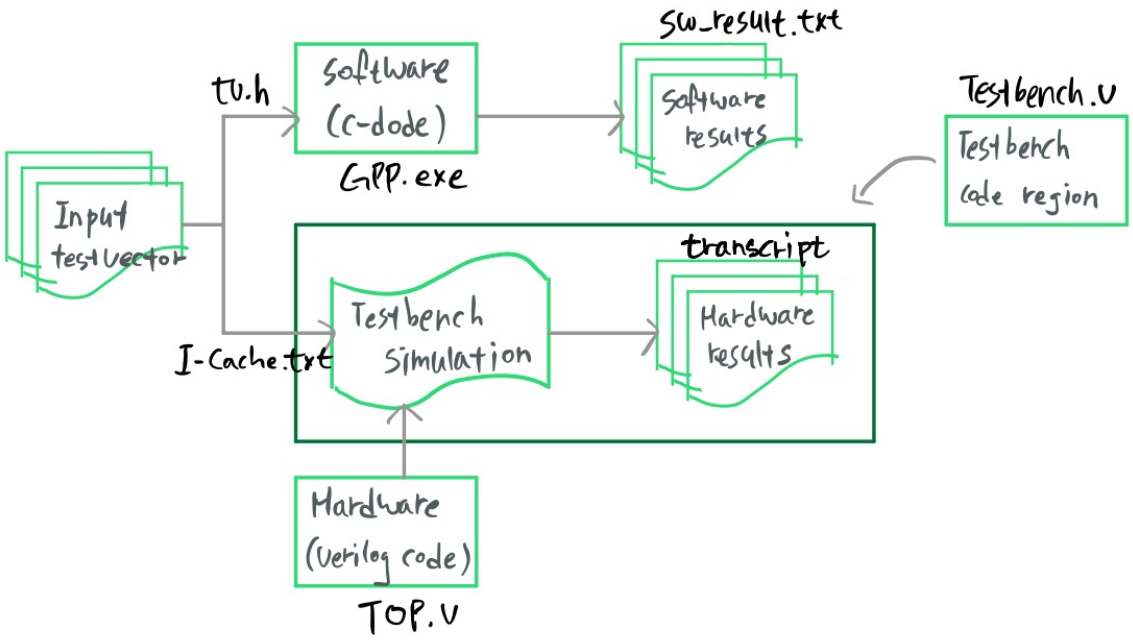
각 연산 수행 후 register에 저장

IC에서 sh 추출

```
70              case 32: // 0x20
71                  regi[rd] = regi[rs] + regi[rt];
72                  printf("Add $(%d), $(%d), $(%d) : %d\n", rd, rs, rt, regi[rd]);
73                  break;
74
75              case 34: //0x22
76                  regi[rd] = regi[rs] - regi[rt];
77                  printf("Subract $(%d), $(%d), $(%d) : %d\n", rd, rs, rt, regi[rd]);
78                  break;
79
80              }
81              break;
82
83          case 8:
84              printf("addi $(%d), $(%d), %d\n", rt, rs, im);
85              regi[rt] = regi[rs] + im;
86              break;
87          }
88      }
89
90      for (i = 0; i < 32; i++)
91          fprintf(fp, "%d: %x\n", i, regi[i]);
92
93      return 0;
94  }
```

register 출력

Test-Vector Generator



- Register는 GPP 내의 변수 -> Testbench에서 비교 불가능
- sw_result.txt와 Hardware results (modelsim의 transcript) 비교

Verilog-Testbench for Simulating RTL Design of the Algorithm

```
1   `include "define.h"
2
3   module TestBench();
4
5       reg Clk, Rst, Rst_M, Str;
6       wire Done;
7
8       reg [(`SA_WIDTH-1):0] Addr;
9       reg [(`D_WIDTH-1):0] Data_I;
10      wire[(`D_WIDTH-1):0] Data_O;
11      reg En, RW;
12
13      reg [(`D_WIDTH-1):0] regi[(`SL_WIDTH-1):0];
14      integer Index;
15      parameter ClkPeriod = 20;
16
17      GPP_TOP CompToTest(Clk, Done, Rst, Str, Addr, Data_I, Data_O, En, RW, Rst_M);
18
19      // Clock Procedure
20      always begin
21          Clk <= 1'b0;    #(ClkPeriod/2);
22          Clk <= 1'b1;    #(ClkPeriod/2);
23      end
24
25      initial $readmemh("src/I-Cache.txt", regi);
26
27      initial begin
28          Rst_M <= 1'b1; Rst <= 1'b0; Str <= 1'b0;
29          En <= 1'b0; RW <= 1'b0;
30          @(posedge Clk);
31
32          Rst_M <= 1'b0;
33          @(posedge Clk);
34
35          for (Index=0; Index<`SL_WIDTH; Index=Index+1) begin
36              $display("Writing to SRAM: Addr=%2d, Data=%h", Index, regi[Index]);
37              Addr <= Index;
38              Data_I <= regi[Index];
39              RW <= 1'b1;
40              En <= 1'b1;
41              @(posedge Clk);
42          end
43
44          En <= 1'b0; RW <= 1'b0;
45          @(posedge Clk);
46
47          Rst <= 1'b1;
48          @(posedge Clk);
50          // S_wait
51          Rst <= 1'b0; Str <= 1'b1;
52          @(posedge Clk);
53
54          // S_initial
55          Str <= 1'b0;
56          @(posedge Clk);
57
58          // when GPP is done
59          while (Done != 1'b1)
60              @(posedge Clk);
61
62          $stop;
63      end
64
65  endmodule
```

GPP와 SRAM을 연동한 최상위 모듈

Clock 생성

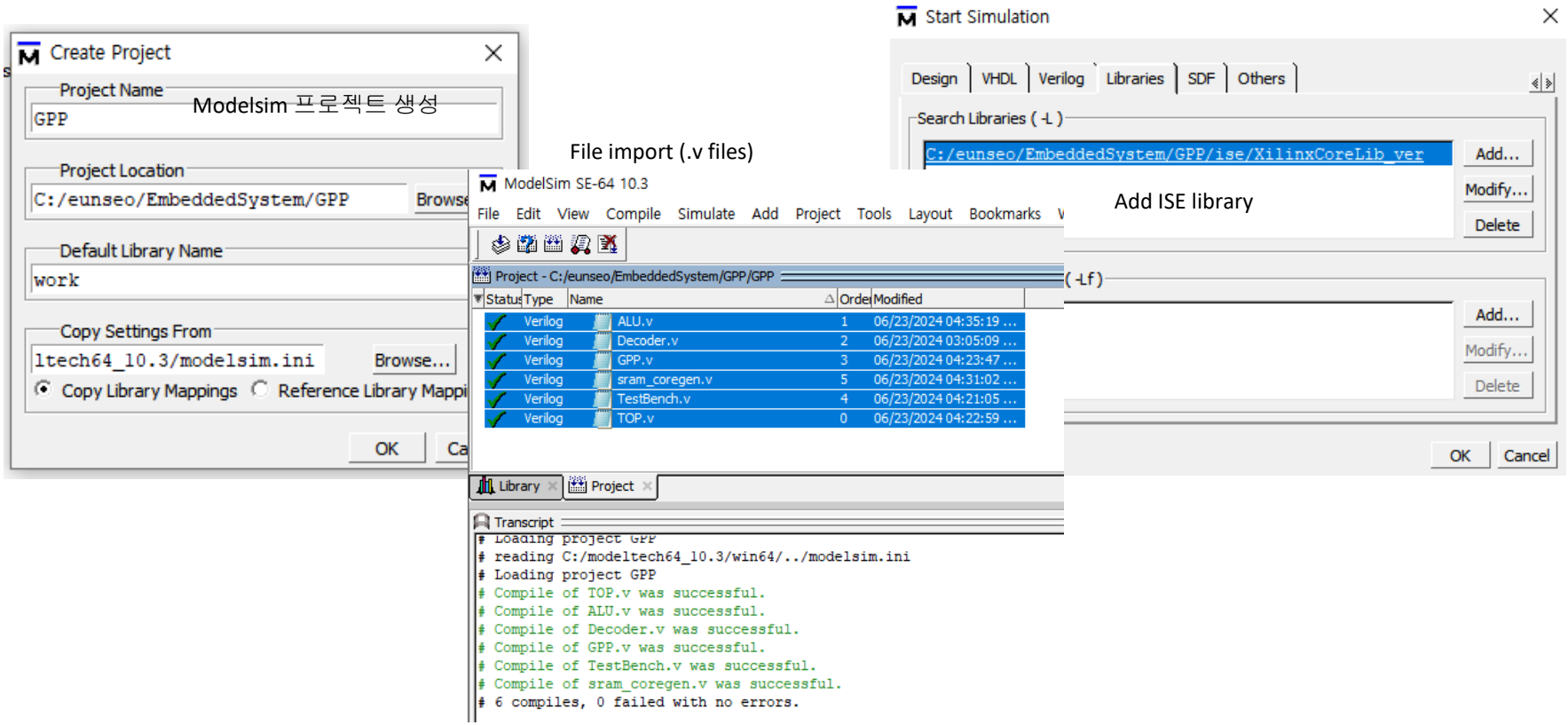
명령어가 저장된 txt파일을 가져와 SRAM에 순차적으로 저장

기본 Rst, Str 변수 설정 -> GPP 실행

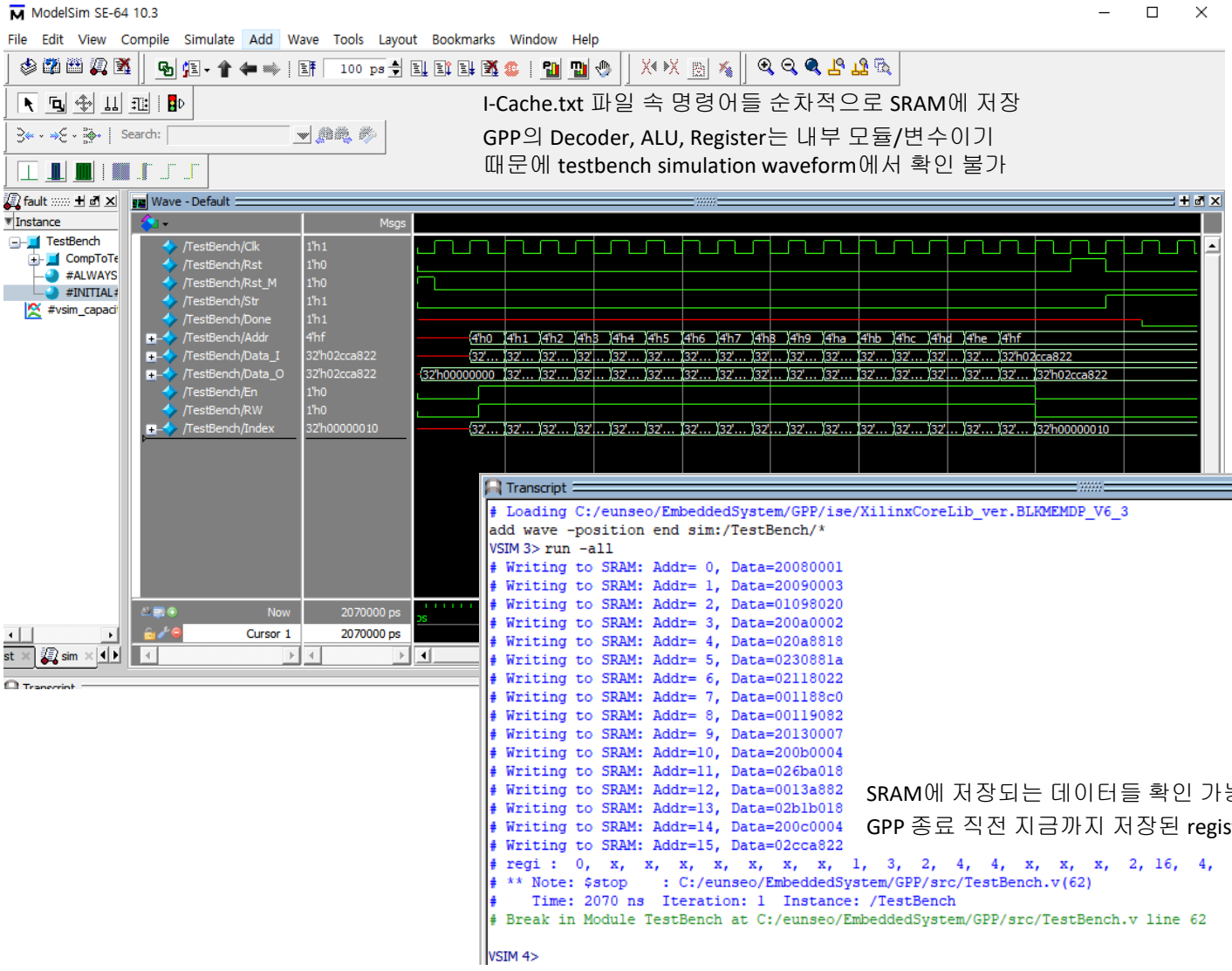
GPP 종료될 때까지 Clk 실행

Modelsim Simulation

- Xilinx ISE에서 modelsim 사용 불가
- 개인 노트북의 modelsim에서 진행



Modelsim Simulation



I-Cache.txt 파일 속 명령어들 순차적으로 SRAM에 저장

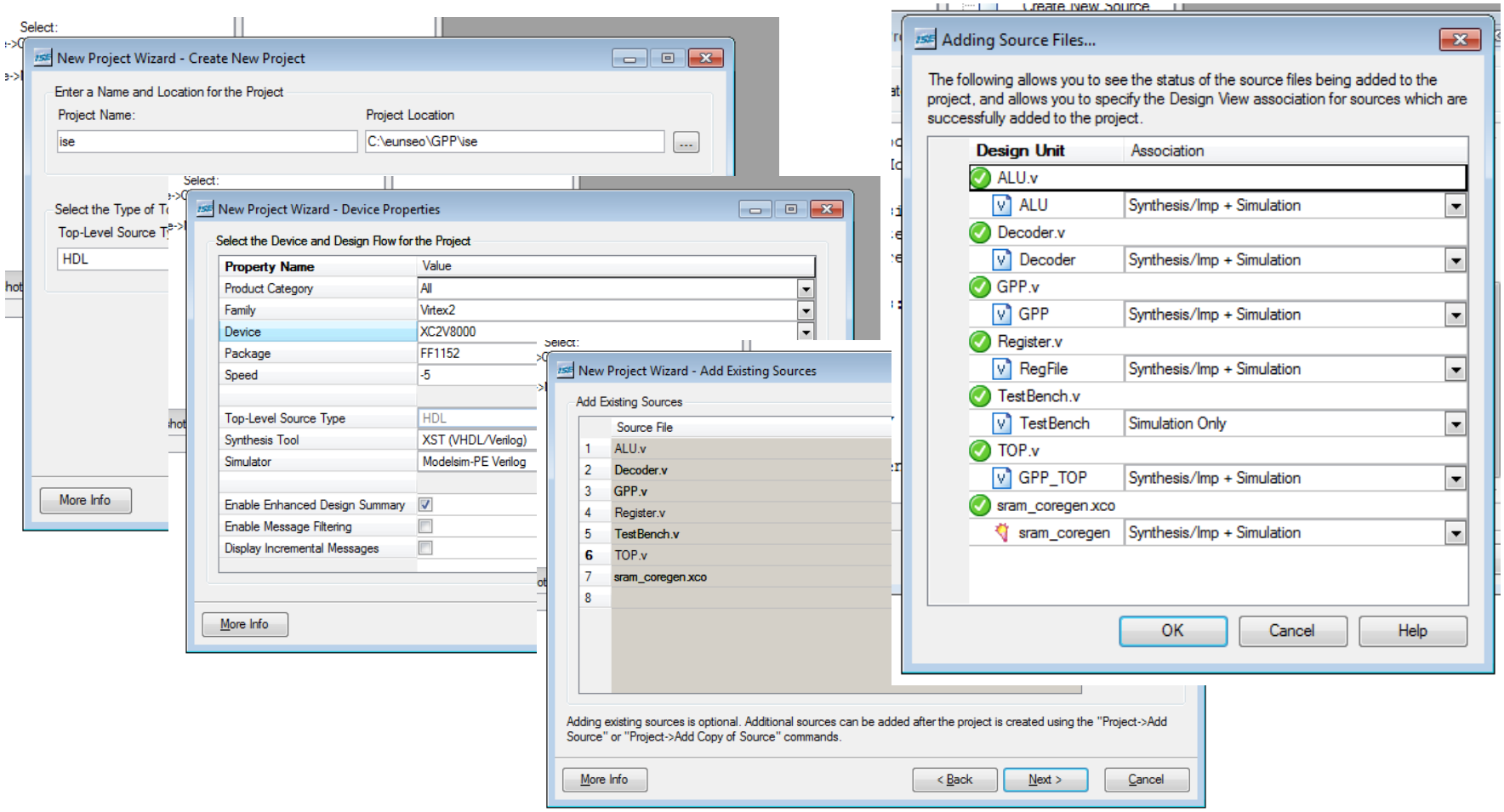
GPP의 Decoder, ALU, Register는 내부 모듈/변수이기 때문에 testbench simulation waveform에서 확인 불가

SRAM에 저장되는 데이터들 확인 가능

GPP 종료 직전 지금까지 저장된 register 변수 내용들 확인 가능

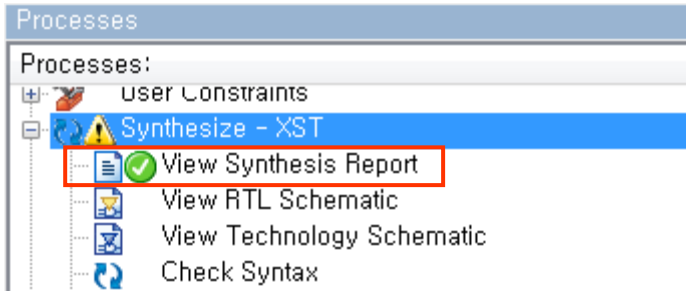
Xilinx ISE – RTL Synthesis by XST

- 프로젝트 생성 및 add files



Xilinx ISE – RTL Synthesis by XST

- 성공적으로 합성 완료




```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Multipliers                          : 1
 32x32-bit multiplier                  : 1
# Adders/Subtractors                   : 2
 32-bit adder                          : 1
 32-bit addsub                         : 1
# Registers                            : 8
 Flip-Flops                            : 8
# Latches                              : 35
 1-bit latch                           : 2
 3-bit latch                           : 1
 32-bit latch                          : 31
 4-bit latch                           : 1
# Multiplexers                         : 3
 32-bit 26-to-1 multiplexer            : 2
 32-bit 4-to-1 multiplexer             : 1
# Logic shifters                       : 2
 32-bit shifter logical left           : 1
 32-bit shifter logical right          : 1

=========================================================================
```

사용된 의미있는 Datapath Component 나열
Fpga 내의 resources를 사용한 것

```
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 2v8000ff1152-5

 Number of Slices:                       45  out of  46592     0%
 Number of Slice Flip Flops:             44  out of  93184     0%
 Number of 4 input LUTs:                 79  out of  93184     0%
 Number of IOs:                          75
 Number of bonded IOBs:                  75  out of    824     9%
    IOB Flip Flops:                       1
 Number of BRAMs:                         1  out of    168     0%
 Number of GCLKs:                         2  out of     16    12%


=========================================================================
```

CLB : Lookup table + flip-flop
Slices : CLB와 Lookup table 사이의 계층 (네 개의 slice가 한 개의 CLB)
Slice flip-flop을 사용하지 않는 Slice 존재
IO는 모두 bonded IOB (실체 칩 밖의 IO pin이 모두 매핑된 상태)
Block RAMs : core generator로 생성한 SRAM

Xilinx ISE – RTL Synthesis by XST

```
================================================================================
Timing constraint: Default period analysis for Clock 'GPP_Core/_not00071'
  Clock period: 5.730ns (frequency: 174.520MHz)
  Total number of paths / destination ports: 1552 / 32
--------------------------------------------------------------------------------
Delay:               5.730ns (Levels of Logic = 33)
  Source:            GPP_Core/PC_1 (LATCH)
  Destination:       GPP_Core/PC_31 (LATCH)
  Source Clock:      GPP_Core/_not00071 falling
  Destination Clock: GPP_Core/_not00071 falling

  Data Path: GPP_Core/PC_1 to GPP_Core/PC_31
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     LD:G->Q               3   0.586   0.878  GPP_Core/PC_1 (GPP_Core/PC_1)
     LUT1:I0->O            1   0.382   0.000  GPP_Core/PC_1_rt (GPP_Core/PC_1_rt)
     MUXCY:S->O            1   0.259   0.000  GPP_Core/Madd__addsub0000_cy<1> (GPP_Core/Madd__addsub0000_cy<1>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<2> (GPP_Core/Madd__addsub0000_cy<2>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<3> (GPP_Core/Madd__addsub0000_cy<3>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<4> (GPP_Core/Madd__addsub0000_cy<4>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<5> (GPP_Core/Madd__addsub0000_cy<5>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<6> (GPP_Core/Madd__addsub0000_cy<6>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<7> (GPP_Core/Madd__addsub0000_cy<7>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<8> (GPP_Core/Madd__addsub0000_cy<8>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<9> (GPP_Core/Madd__addsub0000_cy<9>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<10> (GPP_Core/Madd__addsub0000_cy<10>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<11> (GPP_Core/Madd__addsub0000_cy<11>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<12> (GPP_Core/Madd__addsub0000_cy<12>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<13> (GPP_Core/Madd__addsub0000_cy<13>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<14> (GPP_Core/Madd__addsub0000_cy<14>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<15> (GPP_Core/Madd__addsub0000_cy<15>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<16> (GPP_Core/Madd__addsub0000_cy<16>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<17> (GPP_Core/Madd__addsub0000_cy<17>)
```

Critical Path 구성

```
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<18> (GPP_Core/Madd__addsub0000_cy<18>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<19> (GPP_Core/Madd__addsub0000_cy<19>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<20> (GPP_Core/Madd__addsub0000_cy<20>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<21> (GPP_Core/Madd__addsub0000_cy<21>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<22> (GPP_Core/Madd__addsub0000_cy<22>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<23> (GPP_Core/Madd__addsub0000_cy<23>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<24> (GPP_Core/Madd__addsub0000_cy<24>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<25> (GPP_Core/Madd__addsub0000_cy<25>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<26> (GPP_Core/Madd__addsub0000_cy<26>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<27> (GPP_Core/Madd__addsub0000_cy<27>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<28> (GPP_Core/Madd__addsub0000_cy<28>)
     MUXCY:CI->O           1   0.046   0.000  GPP_Core/Madd__addsub0000_cy<29> (GPP_Core/Madd__addsub0000_cy<29>)
     MUXCY:CI->O           0   0.046   0.000  GPP_Core/Madd__addsub0000_cy<30> (GPP_Core/Madd__addsub0000_cy<30>)
     XORCY:CI->O           1   1.107   0.480  GPP_Core/Madd__addsub0000_xor<31> (GPP_Core/_addsub0000<31>)
     LUT3:I2->O            1   0.382   0.000  GPP_Core/_mux0026<31>1 (GPP_Core/_mux0026<31>)
     LD:D                      0.322          GPP_Core/PC_31
    ----------------------------------------
    Total                     5.730ns (4.372ns logic, 1.358ns route)
                              (76.3% logic, 23.7% route)
```

```
Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
---------------
Speed Grade: -5

   Minimum period: 5.730ns (Maximum Frequency: 174.520MHz)
   Minimum input arrival time before clock: 2.213ns
   Maximum output required time after clock: 6.648ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

최소 동작 속도 : 5.730ns

최대 주파수 : 174.520MHz

- 5.730ns : 가상 이상적인 Clock
- 실제 동작 시 delay로 인해 실행 시간이 더 늘어날 것

Hardware Performance Evaluation

- Modelsim simulation 결과
  - 2070ns 소요
  - Clock 주기 : 20ns
- 총 103.5 사이클
  - Testbench 초기 설정 : 20 사이클 (SRAM 데이터 입력 포함)
  - GPP 초기 설정 : 3 사이클
  - 한 명령어 당 5 사이클 (fetch1, fetch2, decode, execute1, execute2) x 16 : 80사이클
  - GPP 종료 1 사이클

Analysis / Type	**Total Cycle Counts**	**Critical Path Delay /Operating Frequency**	**Execution Time** (Cycle Count x Critical Path Delay)
Hardware (***Two-Procedure RTL)***	103.5	5.730ns / 174.520MHz	103.5 X 5.730 ns = 593.055 ns

Attempt Review

- Register module 합성
  - 합성 시 @* 민감도 문제 -> en로 변경

```
// CombLogic - Write Operation
always @* begin
    for(Index=0; Index<(2**`RA_WIDTH);Index=Index+1)
            RegFile_Next[Index] <= RegFile[Index];

    if(W_en == 1'b1)
        RegFile_Next[W_Addr] <= W_Data;
end
```

```
// CombLogic - Write Operation
always @(W_en) begin
    for(Index=0; Index<(2**`RA_WIDTH);Index=Index+1)
            RegFile_Next[Index] <= RegFile[Index];

    if(W_en == 1'b1)
        RegFile_Next[W_Addr] <= W_Data;
end
```

- S_execute -> S_execute1, S_execute2, S_execute3
  - 값 가져오기, 연산하기, 레지스터에 반영하기로 분리
  - Delay 제거를 위한 State 분리 작업 후 timing 문제 발생

    -> 개별 Register module 삭제 및 내부 변수로 변경

```
reg [(`RA_WIDTH-1):0] R1_Addr, R2_Addr, W_Addr;
wire [(`D_WIDTH-1):0] R1_Data, R2_Data;
reg [(`D_WIDTH-1):0] W_Data;
reg R1_en, R2_en, W_en, dis;

RegFile register(R1_Addr, R2_Addr, W_Addr, R1_en, R2_en, W_en,
R1_Data, R2_Data, W_Data, Clk, Rst, dis);
```

```
uint32_t regi[32] = {0,};
uint32_t op, rs, rt, rd, sh, fn, im;
```

Attempt Review

- Devide
  - / 연산자 사용하여 합성 시 에러 발생 -> BSR로 대체
- BSR module
  - 세부 모듈 output 에러 -> 세부 모듈 삭제 및 BSR에서 개별 진행
  - State 이동 시 delay 문제 발생

  -> 세부 모듈 삭제 및 ALU 내에서 case문으로 처리

```
module BSR(en, BS_AMT, D_IN, D_OUT);

input en;
input [2:0] BS_AMT;
input [`D_WIDTH-1:0] D_IN;
output reg [`D_WIDTH-1:0] D_OUT;
reg [`D_WIDTH-1:0] D_TMP;

BSR2 bsr2 (BS_AMT[1], D_IN, D_TMP);
BSR1 bsr1 (BS_AMT[0], D_TMP, D_OUT);
```

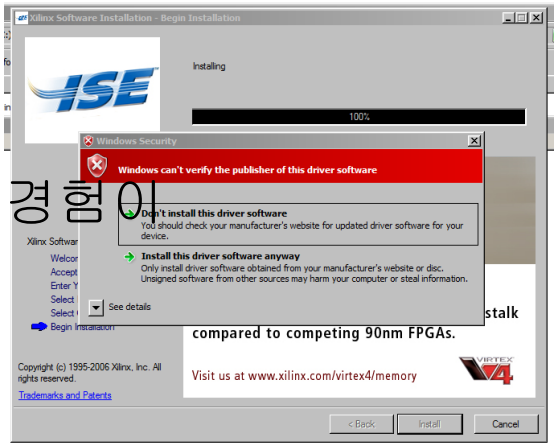
```
always @(en) begin
    D_TMP <= D_IN;
    if (en) begin
        if (BS_AMT[2])
            D_TMP <= {{2{D_TMP[`D_WIDTH-1]}}, D_TMP[`D_WIDTH-1:2]};

        if (BS_AMT[1])
            D_TMP <= {{D_TMP[`D_WIDTH-1]}, D_TMP[`D_WIDTH-1:1]};
    end
    D_OUT <= D_TMP;
end
```

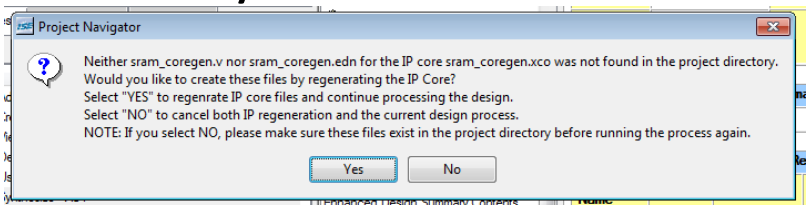
```
3: begin                                    // div
    case (operand2[2:0])
        3'b100: result <= operand1 >> 2;
        3'b010: result <= operand1 >> 1;
    endcase
end
```

Attempt Review

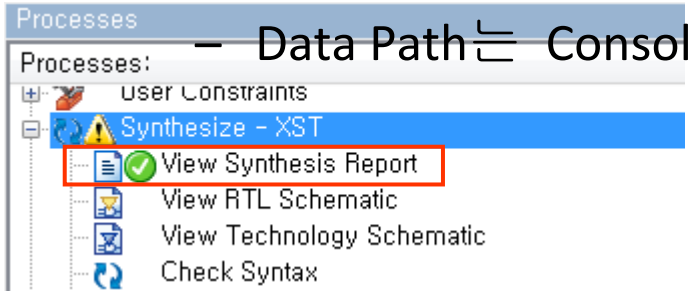
- Xilinx ISE install
  - 보안 경고 -> 설치 클릭
  - 설치 클릭 후 windows에 접근 불가능한 경험이 있어 해결하지 못하는 문제라 생각



- RTL Synthesis by XST



  - Alert : 파일을 프로젝트 내로 복사한다는 안내일 뿐
  - 이후 Data Path에 대한 report를 찾을 수 없어 합성 실패로 착각
  - Data Path는 Console에는 출력되지 않고, Report 창에서만 출력



Improvements

- Register : GPP 내 변수가 아닌, 실제 register 생성
- ALU : Divide 수행 시 BSR 사용
  - 추후 실수를 사용한 곱셈, 나눗셈 구현
- D-Cache(SRAM) 및 Memory(DRAM) 구현
  - 실제 GPP의 구조와 더 가까운 형태
  - Load, store 명령어 수행
- I-Cache 생성기 생성
  - 사용자가 명령을 입력하면 그에 맞는 마이크로 명령어를 자동으로 I-Cache에 추가하는 모듈 구현
  - a = b + c -> load b, load c, add b c a, store a
- 이상적인 GPP 구현이 최종 목표

Thank you

Eunseo Ko

IT Engineering
Sookmyung Women's University