디지털 논리회로

miniCPU 프로젝트 (6조)

중간보고서

담당교수: 박재근

조장: 임효준 (학점교류, 20246096)
조원: 김희균 (학점교류, 20246079)
김효경 (학점교류, 20246036)

제출일자: 2024.05.29.

목차

1) 역할분담

2024학년도 1학기 숭실대학교 '디지털 논리회로' 과목의 mini CPU 제작 팀 프로젝트를 진행함에 있어서 각 조원이 아래와 같이 역할을 분담하여 진행하기로 하였음.

임효준 - 프로그램 설계(코딩)
김희균 - 보고서 작성, 발표
김효경 - 자료조사, 내용기록

위의 역할분담은 해당 역할을 맡은 인원만이 그 역할을 수행한다는 것이 아닌 주로 그 역할을 수행하는 것으로, 조원끼리 부족한 부분을 서로 도와가며 프로젝트를 진행하기로 하였음.

2) 회의록

1차)

역할분담 실시

CPU의 기본 구조와 원리에 대해 살핌

2차)

각 조원이 인터넷 검색을 통하여 간단한 verilog CPU 코드들을 가져와 조사 후 분석하여 충분히 숙지하는 시간을 가짐

추후 조원들에게 설명해주고 서로 조사한 각 코드의 공통점과 차이점을 알아내어 토론하는 시간을 가질 예정

3차)

2차 때 회의했던 코드에 대해 각자 설명하였고 공통점과 차이점에 대해 토론함

해당주차 수업시간에 학습한 내용(Half-Adder, Full-Adder)에 대해 복습함

4차)

해당주차 수업시간에 학습한 내용(D Latch, D flip-flop)에 대해 복습함

5차)

지난주차에 학습했던 내용(D Latch, D flip-flop)에 관한 verilog 코드들을 Chat-GPT를 활용하여 구현해보고 vivado 프로그램을 사용하여 파형을 시뮬레이션 해봄

해당주차 수업시간에 학습한 내용(FSM)에 대해 심도 있게 복습함

6차)

ZOOM 회의 간 미래도서관 그룹스터디룸에서 대면 회의 계획

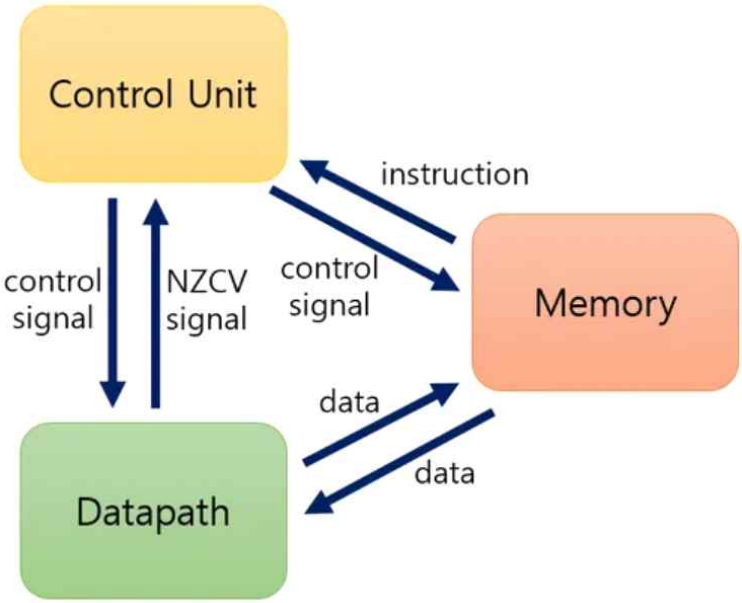
github로 16bit CPU 코드 작성 및 ALU 시뮬레이션 실행하였으나 오류발생 후 수정할 방법 고안하였음

3) 프로젝트 개요

mini CPU 프로젝트의 주제로 mini CPU를 Xilinx사의 vivado 또는 Intel사의 Quartus 프로그램을 활용하여 HDL(verilog)로 설계하는 프로젝트로 우리 6조는 vivado 프로그램을 활용하여 간단한 16bits CPU를 주제로 설계하기로 결정하였음.

Instruction fetch - Decode - Execution 순의 CPU 처리과정을 기반으로 하여 설계.

4) 프로젝트 진행현황



위 그림에서 볼 수 있듯이 CPU는 크게 컨트롤유닛, 메모리, 데이터패스로 분류할 수 있다. 이때 메모리는 데이터를 저장하는 역할을 하며 데이터패스는 메모리에서 데이터를 가져와(fetch) 연산을 수행(decode, execute)하는 역할을 한다. 컨트롤유닛은 이러한 전반적인 과정을 통제하는 역할을 수행한다.

우리 6조는 이중에서 데이터패스에 집중하여 vivado로 16bits CPU(데이터패스)를 설계하기로 결정했다.

데이터패스의 verilog code를 다음과 같이 작성하였다.

tb_16bits_cpu_datapath.v * | 16bits_cpu_datapath.v * | Untitled 1

C:/Users/dlagy/digital_logic_16bits_cpu/digital_logic_16bits_cpu.srcs/sources_1/nev

```
1   `timescale 1ns / 1ps
2   module datapath(
3    input clk,MB,MD,RW,
4    input [2:0] DA,AA,BA,
5    input [3:0] FS,
6    input [15:0] Datain,
7    output reg [15:0] Dataout,da,aa,ba
8   );
9   // register 초기값 '3'으로 설정.
10  reg [15:0] register[7:0];
11  integer i;
12  initial begin
13  for (i=0;i<8;i=i+1)
14        register[i] <= 16'd3;
15  end
16  always @(posedge clk)
17  begin
18  da <= register[DA]; aa<= register[AA]; ba <= register[BA];
19  //우선시 되는 신호들 설정 (FS 사용 안함)
20  if(~RW)
21      Dataout <= ba;     // no write
22  else if (MD)
23      da <= Datain;      // Data in
24  // constant 신호 (FS 사용)
25  else if (MB)
26      case(FS)                    // function code 설정
27      4'b0000: da <= aa;
28      4'b0001: da <= aa +1;
29      4'b0010: da <= aa + BA;
30      4'b0011: da <= aa + BA + 1;
31      4'b0100: da <= aa + ~BA;
32      4'b0101: da <= aa + ~BA + 1;
33      4'b0110: da <= aa - 1;
34      4'b0111: da <= aa ;
35      4'b1000: da <= aa & BA ;
36      4'b1001: da <= aa | BA ;
37      4'b1010: da <= aa ^ BA ;
38      4'b1011: da <= ~aa ;
39      4'b1100: da <= BA ;
40      4'b1101: da <= BA >> 1;
41      4'b1110: da <= BA << 1;
42  endcase
43  // function code 설정
44  else
45  case(FS)
46      4'b0000: da <= aa;
47      4'b0001: da <= aa +1;
48      4'b0010: da <= aa + ba;
49      4'b0011: da <= aa + ba + 1;
50      4'b0100: da <= aa + ~ba;
51      4'b0101: da <= aa + ~ba + 1;
52      4'b0110: da <= aa - 1;
53      4'b0111: da <= aa ;
54      4'b1000: da <= aa & ba ;
55      4'b1001: da <= aa | ba ;
56      4'b1010: da <= aa ^ ba ;
57      4'b1011: da <= ~aa ;
58      4'b1100: da <= ba ;
59      4'b1101: da <= ba >> 1;
60      4'b1110: da <= ba << 1;
61  endcase
62  end
63  endmodule
```

datapath module을 만들고 필요한 변수들 (clk, MB(메모리버스 신호), MD(메모리데이터 신호), RW(읽기/쓰기 신호))과 register address (DA, AA, BA), function select(FS)를 설정하였다. 그리고 MD=1일 때 작동하는 input Datain을 정의하고, RW=0일 때, no write로 register[BA]의 값을 반환하는 output Dataout을 정의하였다. MB=1일 때, register[BA]값을 사용하는 것이 아니라 상수 BA값을 계산에 사용하게 되므로 FS 에서 ba를 연산에 사용하는 것이 아니라 BA를 사용하여 연산하였다.

da, aa, ba는 각각 register[DA], register[AA], register[BA]를 의미하며, DA, AA, BA의 address에 해당하는 16bit register값을 의미한다.

reg[15:0] register[7:0];로 16bit register[0]~register[7] 8개의 register를 정의하였고, 초깃값들은 16'd3 으로 설정하였다. Positive edge clock 신호마다 동작하며, FS과 상관없이 우선시 되는 RW, MD를 if, else if 문으로 실행하고, FS 코드와 같이 동작하는 MB는 하단에 FS 코드를 사용하였으며, else 문에서 RW=1, MD=0, MB=0을 갖는 기본 FS만 실행된다.

아래의 표와 같은 연산을 수행할 데이터패스의 테스트벤치를 다음과 같이 작성하였다.

	operation	DA	AA	BA	MB	FS	MD	RW
1	R1=R2-R3	001	010	011	0	0101	0	1
2	R4=sl R6	100	XXX	110	0	1110	0	1
3	R7=R7+1	111	111	XXX	0	0001	0	1
4	R1=R0+2	001	000	XXX	1	0010	0	1
5	Dataout=R3	XXX	XXX	011	0	XXXX	X	0
6	R4=Datain	100	XXX	XXX	X	XXXX	1	1
7	R5=0	101	000	000	0	1010	0	1

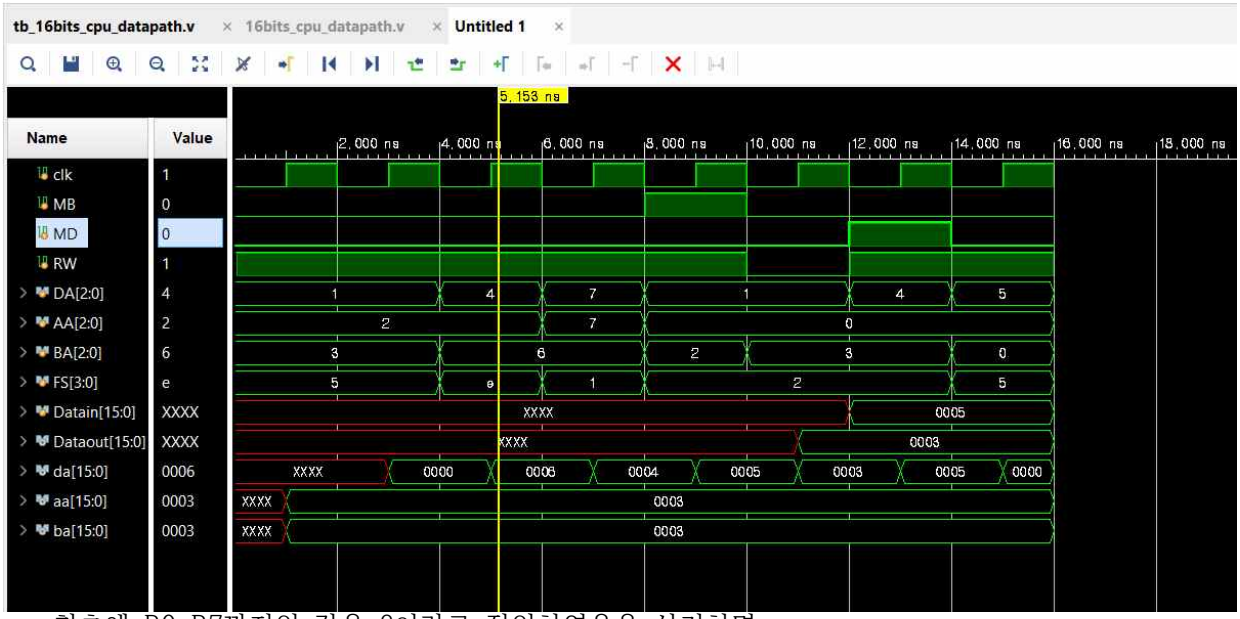
tb_16bits_cpu_datapath.v * | 16bits_cpu_datapath.v | Untitled 1

C:/Users/dlagy/digital_logic_16bits_cpu/digital_logic_16bits_cpu.srcs/sim_1/new/tb_16bits_cpu_datapath.v

```
1   `timescale 1ns / 1ps
2   module Test;
3       reg clk,MB,MD,RW;
4       reg [2:0] DA,AA,BA;
5       reg [3:0] FS;
6       reg [15:0] Datain;
7       wire [15:0] Dataout;
8       wire [15:0] da,aa,ba;
9           //testbench 변수와 source code counter 변수 연결
10  datapath uut(
11  .clk(clk),.MB(MB),.MD(MD),.RW(RW),.DA(DA),.AA(AA),.BA(BA),.FS(FS),.Dataout(Dataout),.Datain(Datain),.da(da),.aa(aa),.ba(ba)
12  );
13      always begin
14      #1 clk = ~clk; //clk 신호는 1ns마다 변화
15      end
16
17      initial begin
18          clk = 0;
19          DA = 3'b001; AA = 3'b010; BA = 3'b011; MB = 0; FS=4'b0101;MD=0; RW=1;
20          #4 DA=3'b100; BA = 3'b110; MB = 0; FS=4'b1110; MD = 0; RW = 1;
21          #2 DA = 3'b111; AA = 3'b111; MB = 0; FS = 4'b0001; MD = 0; RW = 1;
22          #2 DA = 3'b001; AA = 3'b000; MB = 1; FS=4'b0010; MD = 0; RW = 1; BA = 3'b010;
23          #2 BA = 3'b011; MB = 0; RW=0;
24          #2 DA = 3'b100; MD = 1; RW = 1; Datain = 16'd5;
25          #2 DA = 3'b101; AA = 3'b000; BA = 3'b000; MB=0; FS=4'b101; MD = 0; RW = 1;
26          #2
27          $finish;
28      end
29  endmodule
```

1ns 마다 clk 신호는 변화하므로 positive edge clock 주기는 2ns다. 처음 clk=0 으로 설정하고, 변수들을 입력하였다. 처음에 #4에 해당하는 시간동안 실행한 이유는 AA, BA의 address의 값들이 2ns 동안 지정되고, 이후 이 값들을 이용하여 FS code가 실행되어 DA값에 저장이 되므로 총 4ns가 필요하기 때문이다. 이후에는 clock주기 2ns만큼 간격을 두고 실행 하였다. XXX로 표현되는 코드들은 FS 에서 영향을 미치지 못하므로 이전 값을 그대로 갖도록 설정하였다. Dataout, Datain 값도 FS에 의해 fetch되지 않더라도 이전 값들을 그대로 유지한다.

위의 verilog code를 vivado 프로그램을 이용하여 파형을 구하면 다음과 같다.



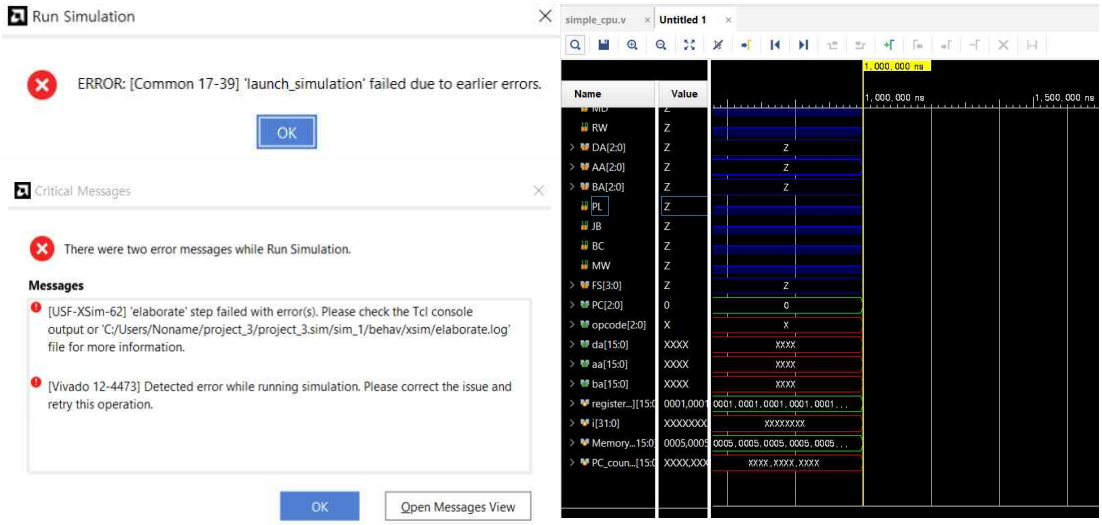
최초에 R0~R7까지의 값을 3이라고 정의하였음을 상기하면,

1. 3ns에서 R1=R2-R3이 일어나고 여기서 R2, R3가 3이므로 R1=3-3=0이다.
   R1(da)=0, R2(aa)=3, R3(ba)=3
2. 5ns에서 R4=sl(shift left)R6이고, 이때 R6=16'd3이므로 sl0000…011(2)->0000…0110(2)가 되므로 R4=6이다.
   R4(da)=6, R6(ba)=3
3. 7ns에서 R7=R7+1이고, 이때 R7=3이므로 R7=3+1=4이다.
   R7(da)=4
4. 9ns에서 R1=R0+2이고, 이때 R0=3이며 상수값 2는 BA값을 그대로 가져왔으므로, R1(da)=3+2=5이다.
   R0(aa)=3, BA=3'b010
5. 11ns에서 Dataout=R3이고, 이때 R3=3이므로, Dataout=3이다.
   R3(ba)=3
6. 13ns에서 R4=Datain이고, Datain은 임의의 값 16'd5라고 정했으므로 R4(da)=5이다.
   Datain=16'd5
7. 15ns에서 R5=0이고, 이때 같은 값을 xor하면 0이므로 R5(da)=R0 xor R0=0이다,
   R0(aa)=3, R0(ba)=3

예상한 값과 같이 데이터패스가 동작했음을 확인할 수 있다.

5) 실패사례 및 애로사항

<실패사례>



1. 구문 오류

- 코드를 작성하면서 종종 세미콜론을 빠뜨리거나, 괄호의 짝이 안 맞거나, 잘못된 키워드를 사용하여 오류가 발생했다.

  ->오류 메시지를 확인하고, 해당 부분을 수정하였다.

2. 논리 오류

- 코드가 의도한 대로 동작하지 않았다.

  ->시뮬레이션 결과와 기대한 결과를 비교하여 코드의 논리를 점검하여, 잘못된 논리 연산과 조건문을 사용하였다는 것을 발견하고 수정하였다.

3. 코딩 오류

- 'reg'타입 변수를 'wire'타입 변수에 할당하여 오류가 발생했다.

  -> 변수 타입을 정확하게 확인하고 일관성 있게 사용하였다.

- 테스트 벤치에서 초기화되지 않은 변수를 사용하여 오류가 발생하였다.

  -> 초기화되지 않은 변수는 기본값이 X(unknown)상태일 수 있어, 테스트 벤치 코드의 모든 변수와 레지스터가 초기화되었는지 확인 후 시뮬레이션을 통해 검증하였다.

4. 타이밍 오류

CPU는 클록 신호에 동기화되어 동작한다. 클록 주기 내에 모든 신호가 안정되어야 하는데, 클록 신호가 올바르게 설정되지 않거나 시간지연이 잘못 설정되면 타이밍 오류가 발생한다.

- 클록 주기가 짧아 데이터 전파 지연이 발생하였다.

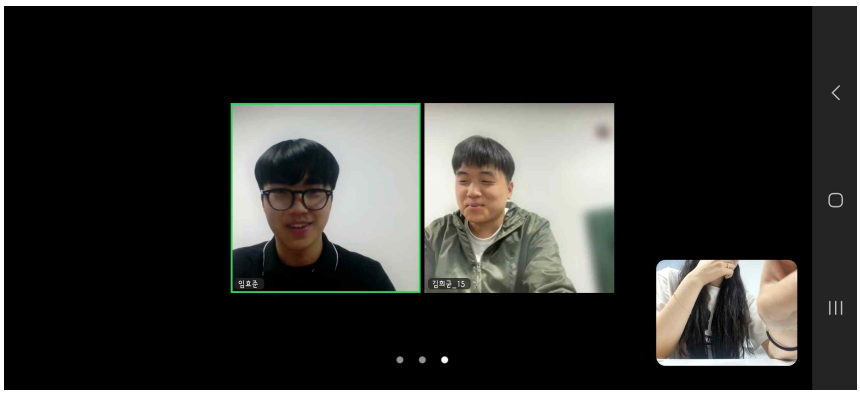
  -> 타이밍 분석 도구를 사용하여 타이밍 제약을 확인하여 타이밍을 개선하였다.

<설계 시 겪은 애로사항>

1. 복잡한 설계

CPU는 많은 기능 블록으로 구성되어 있어, 각각의 블록이 정확하게 동작하도록 설계하는 것이 매우 어려운 작업이다. 따라서, 블록 단위로 설계를 나누고, 각 블록을 검증한 후 통합하는 방식을 사용하여 모듈화된 설계와 반복적인 테스트로 복잡성을 줄였다.

6) 모임 사진



2024.05.22.(수) ZOOM회의

2024.05.28.(화) 대면회의