마이크로프로세서 설계 무작정 따라하기 (4)

KAIST 전자전산학과 박사과정 배영돈(donny@ics.kaist.ac.kr)

6. 시뮬레이션

가) 테스트 모듈의 설계

그림 1 과 같이 테스트하고자 하는 설계를 테스트 모듈에 넣고(instance) 테스트 입력을 생성하여 인가한다.

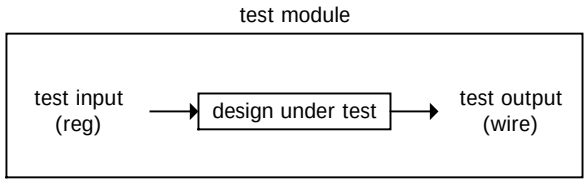

그림 1. 테스트 모듈의 구성

테스트 입력은 reg 형태로 선언하여 initial 또는 always 블록에서 임의의 값을 인가할 수 있도록 하며, 테스트 출력은 wire 를 사용한다.

SimpleCore 에 사용된 ALU 회로를 시뮬레이션 해보자. 다음은 ALU 를 시뮬레이션하기 위한 테스트 모듈이다.

ALU 테스트 모듈(alu_test.v)의 RTL 코드

```
module alu_test;

reg [15:0] aluAIn;     // alu input A
reg [15:0] aluBIn;     // alu input B
reg [ 2:0] aluCtl;     // alu control input
wire [15:0] aluOut;    // alu output
wire [ 3:0] condFlag; // condition flags

alu Ialu(
    aluAIn,
    aluBIn,
    aluCtl,
    aluOut,
    condFlag
);

always#10
begin
    aluAIn =$random;    // random value generation
    aluBIn =$random;    // random value generation
end
```

```
initial
begin
    $dumpvars;        // dump variables
    $monitor($time,"\t%h + %h = %h",aluAIn, aluBIn, aluOut); // variable monitor
    aluCtl = 3'b100;  // ADD
    #100 $finish;
end

endmodule
```

먼저 initial 부분을 살펴보면 $dumpvar 을 사용하여 시뮬레이션 결과를 저장하도록 하였다. (verilog.dump 에 저장된다.) 다음 $monitor 를 사용하여 변수의 값이 변경되면 화면에 표시하도록 하였다. ALU 가 덧셈 기능을 수행하도록 제어신호(aluCtl) 값을 설정하였다. 다음 100 ns 후에 시뮬레이션이 종료된다.

다음 always 블록에서 10ns 마다 ALU 의 입력(aluAIn, aluBIn)을 $random 을 사용하여 생성한다.

나) Verilog-XL 의 실행

$ verilog alu.v alu_test.v <enter>

시뮬레이션 결과 화면은 다름과 같다.

```
alu_test
                0       xxxx + xxxx = xxxx
               10       3524 + 5e81 = 93a5
               20       d609 + 5663 = 2c6c
               30       7b0d + 998d = 149a
               40       8465 + 5212 = d677
               50       e301 + cd0d = b00e
               60       f176 + cd3d = beb3
               70       57ed + f78c = 4f79
               80       e9f9 + 24c6 = 0ebf
               90       84c5 + d2aa = 576f
L28 "alu_test.v": $finish at simulation time 100
```

여러 개의 파일을 함께 실행하기 위해서는 –f 옵션을 사용하여 ‘verilog –f <command argument file>’의 형태로 사용하면 편리하다. SimpleCore 의 시뮬레이션에서는 simplecore.f 파일을 만들어서 사용하였다.

다) Waveform viewer 의 실행.

현재 많이 사용되는 waveform viewer 는 Signalscan, SimWaves, CWaves, Ut(Undertow)등이 있으며 사용방법은 대동소이하다. 본 강좌에서는 SimWaves 를 사용한다. 실행방법은 다음과 같다.

$ wd <enter>

그림 2, 그림 3 와 같이 verilog.dump 파일을 읽어와서 신호를 출력하면 그림 4 과 같은 결과를 볼 수 있다.

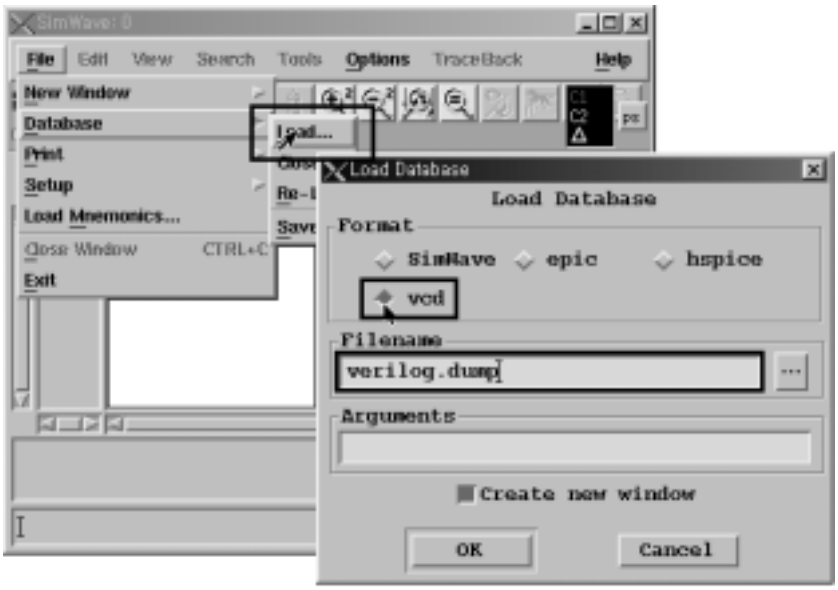

그림 2. Database Load

그림 3. 신호 선택

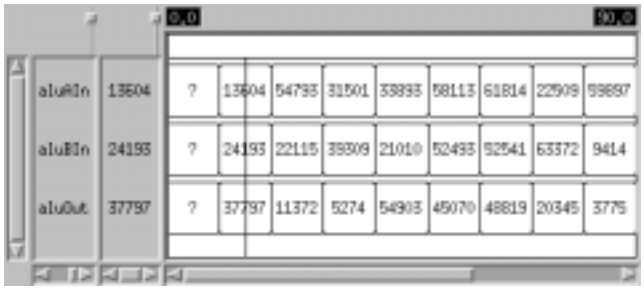

그림 4. 시뮬레이션 결과

7. SimpleCore 의 시뮬레이션

SimpleCore 를 시뮬레이션하기 위하여 그림 5 와 같은 간단한 환경의 테스트 모듈을 설계하였다.

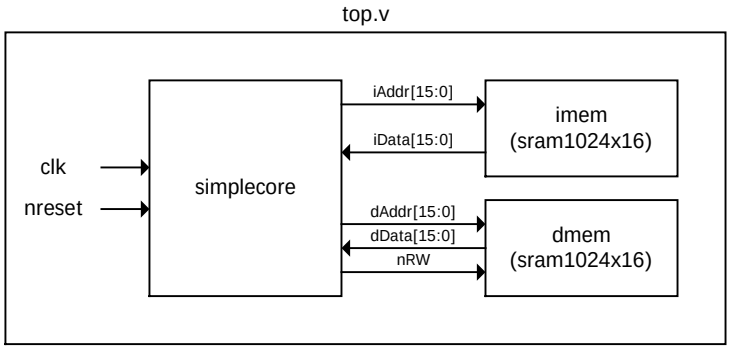

그림 5. SimpleCore 의 시뮬레이션 환경

명령어 메모리(imem)과 데이터 메모리(dmem)을 위하여 간단한 메모리 모델을 사용하였다. ALU 의 시뮬레이션과 다른점은 바로 이 메모리를 사용하였다는 점이며 메모리를 초기화하기 위하여 테스트 모듈(top.v)에 다음과 같이 기술한다. 즉, 시뮬레이션의 시작과 함께 imem.hex, dmem.hex 파일의 내용으로 메모리를 초기화 하는 것이다.

```
initial
        $dumpvars;
        $readmemh("imem.hex", imem.ram, 0);
        $readmemh("dmem.hex", dmem.ram, 0);
end
```

다음은 간단한 loop 프로그램 명령어 메모리의 내용이다.

```
// test_vector: loop
//
0000 // mov r0,   #0
// loop
0801 // add r0,   #1
2803 // cmp r0,   #3
dff8 // brne loop
4000 // nop
4000 // nop
3800 // end
```

이제 Verilog-XL 을 실행하고 시뮬레이션 결과를 확인하자.

```
$ verilog -f simplecore.f <enter>
```

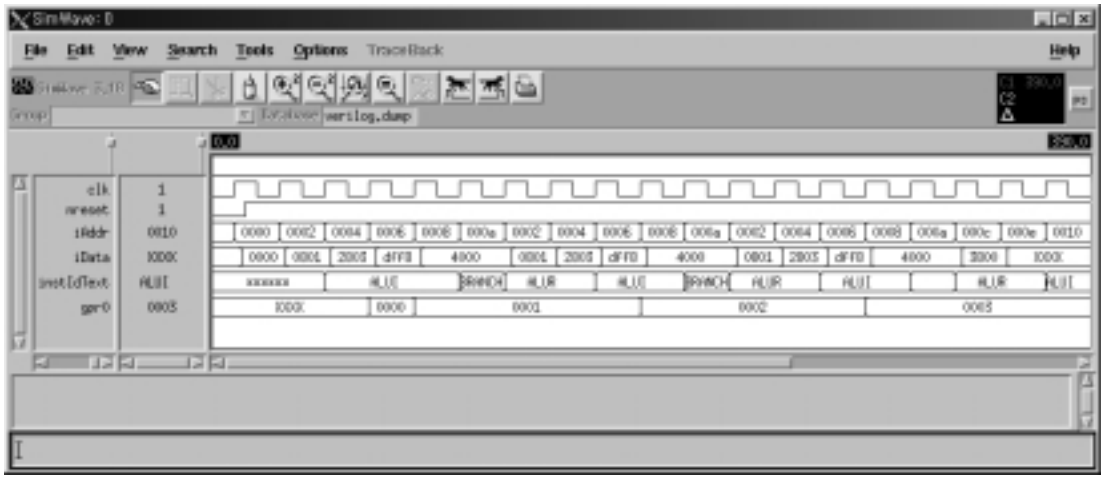

그림 6. loop 프로그램의 수행결과

시뮬레이션 결과(그림 6)를 살펴보면 gpr0 가 0 으로 초기화된 다음 3 까지 덧셈을 반복하는 것을 확인할 수 있다.

8. 맺음말

본 강좌에서는 4 회 동안 16 비트 마이크로프로세서의 설계에 대하여 연재하였다. 제한된 지면으로 보다 많은 내용을 싣지 못하여 아쉬움이 남는다. SimpleCore 가 실제 내장형 프로세서로 사용되기 위해서는 인터럽트, power-down, 테스트와 같은 기능이 추가적으로 요구된다. 이것은 독자들의 몫으로 남기고자 한다.

반도체 설계과정은 크게 전단계(front-end)와 후단계(back-end) 로 구분되며 본 강좌에서는 전단계까지의 내용을 설명하였다. IDEC 라이브러리를 사용하면 직접 후단계 설계를 해볼 수있으며, MPW 를 통해서 실제 칩을 제작할 수도 있다. 본 강좌를 계기로 앞으로 MPW 에서 독자적인 내장형 프로세서들이 많이 제작되고 더 나아가 실제 상품에도 사용되었으면 하는 바람이다.

* SimpleCore 의 전체 소스코드가 홈페이지(http://www.donny.co.kr/simplecore)를 통해 제공됩니다.