DataLab. Internship Program (2023 Summer)

Week 1

Tutor: 배은태

학습 목표

5주 간의 연구실 인턴 교육 프로그램을 통해 FPGA(Field Programmable Gate Array) 기반의 하드웨어 설계의 기초부터 Bluespec System Verilog를 이용한 해시 알고리즘 가속기의 구현과 RISC-V Custom Instruction를 이용한 제어까지 학습하고 실습할 예정입니다.

하드웨어를 설계하기 위해서 직접 회로도(schematic)를 그리는 대신 하드웨어의 동작을 기술하는 일종의 프로그래밍 언어인 하드웨어 기술 언어(Hardware Description Language, HDL)를 이용할 수도 있습니다. 대표적인 HDL에는 Verilog HDL과 VHDL, System Verilog 등이 있으며, 이보다 고수준의 문법을 제공하는 Bluespec System Verilog(BSV)나 Chisel 등의 High-level HDL(HLHDL)도 존재합니다. 또한 고급 언어인 C/C++를 이용하여 하드웨어 커널을 기술하는 High-level Synthesis(HLS)도 널리 사용되고 있습니다.

HDL을 이용한 하드웨어 회로 설계는 프로그래밍 언어를 사용한다는 점에서 소프트웨어 개발과 닮은 점도 있지만 논리 합성(synthesis)이나 구현(implementation) 등 생소한 하드웨어 용어부터 순차적으로 실행되는 소프트웨어 코드와는 사뭇 다른 하드웨어 회로의 동작까지 많은 부분이 다소 낯설게 다가올 수 있습니다. 따라서, 1-2주차에는 기본적인 Verilog HDL 코딩과 시뮬레이션, Xilinx FPGA 사용법에 대해 익히고, 이후에 BSV를 이용하여 좀 더 복잡한 회로를 구성하고자 합니다.

1주차에는 논리회로에 대한 기초적인 지식과 Verilog HDL의 기본 문법을 익히고 시뮬레이션을 통해 회로의 동작을 검증해보는 것을 목표로 합니다.

실습 환경 준비

소프트웨어와 마찬가지로 하드웨어 역시 의도에 맞게 설계되었는지 동작을 검증하는 단계가 필요합니다. 회로를 직접 합성하기 앞서(단순한 회로도 합성하고 구현해서 보드에 올리려면 상당한 시간이 필요합니다), 시뮬레이터를 이용하면 회로의 입출력 신호를 모니터링할 수 있습니다. 시뮬레이션을 하려면 회로가 원하는 값을 출력하는지 확인할 수 있도록 입력될 값을 직접 정의해줘야 합니다. 이를 테스트벤치(testbench) 또는 테스트 벡터라고 합니다. 또한 Verilog HDL은 시뮬레이션을 위한 각종 시스템 태스크($display)와 시뮬레이션에서만 사용 가능한 문법들(initial 등)을 제공하고 있습니다. 이들 문법들은 논리 합성, 즉 하드웨어 회로로 구현은 불가능하지만, 회로의 기능(functionality)을 검증하기 위한 유용한 도구로 사용할 수 있습니다.

Vivado는 Xilinx(AMD에 인수)에서 제공하는 EDA tool입니다. HDL 코드에 대한 시뮬레이션은 물론이고, Xilinx FPGA를 대상으로 논리 합성(synthesis), 구현(implementation), 비트스트림 생성

(bitstream generation) 등 다양한 기능을 제공합니다. Vivado는 무료 버전인 Standard Edition과 유료 버전인 Enterprise Edition이 있으며, 더 많은 제품군(FPGA)을 지원하느냐의 차이입니다. 실습에서 사용되는 교육용 보드는 Standard Edition으로도 충분합니다. EDA 툴은 하드웨어에 맞게 선택하시면 됩니다. 예를 들어, Xilinx FPGA 대신 Altera(Intel에 인수) FPGA를 사용할 경우에는 Vivado 대신에 Quartus를 선택하면 됩니다. (**Vivado는 용량이 매우 크고, 설치 시간도 크고 아름답기 때문에 미리 설치해두시기를 권장합니다.**)

https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html

물론 Vivado에도 시뮬레이터(xsim)가 존재하지만, 가볍고 편리하게 이용할 수 있는 유용한 오픈소스 툴이 있어서 소개하고자 합니다. 이들 툴은 설치 방법도 크게 복잡하지 않고, CLI 환경에서 쉽게 실행할 수 있습니다.

우선 Icarus Verilog입니다. Icarus Verilog(iverilog)는 입력된 소스코드(HDL 코드)를 실행 가능한 시뮬레이션 파일로 컴파일합니다. 이렇게 생성된 실행 파일은 vvp라는 런타임 엔진을 이용하여 실행시킬 수 있습니다. 시뮬레이션 파일을 실행하면 테스트벤치의 입력에 맞게 시뮬레이션을 진행하며, 그 결과를 터미널을 통해 확인할 수 있습니다. 또한 시뮬레이션 과정에 생성된 입력과 출력 신호를 파일 형태로(.vcd 파일) 덤프하여 파형을 확인할 수도 있습니다. 시뮬레이션 결과로 출력된 vcd 파일을 시각화 해주는 오픈소스 툴도 존재하며, gtkwave가 잘 알려져 있습니다.

https://steveicarus.github.io/iverilog/usage/installation.html

Icarus Verilog는 소스코드를 직접 컴파일하여 설치할 수도 있지만, 그냥 패키지 관리자로 설치할 수도 있습니다.

```
$ sudo apt-get install build-essential libboost-dev iverilog
$ sudo apt-get install gtkwave
```

iverilog의 사용법은 gcc 같은 CLI 환경에서 동작하는 컴파일러와 매우 비슷하기 때문에 gcc 사용에 익숙한 분들이라면 어렵지 않게 사용할 수 있을 것입니다. 예를 들어, ha.v라는 소스 파일과 이에 대한 테스트벤치인 tb_ha.v를 이용하여 시뮬레이션 파일을 생성하려면 다음과 같이 입력하면 됩니다(Verilog HDL의 소스 파일 확장자는 .v입니다).

```
$ iverilog tb_ha.v ha.v
$ vvp a.out
```

출력 파일을 지정하지 않으면 기본적으로 a.out이라는 파일이 생성되지만, (gcc와 마찬가지로) -o옵션을 이용하여 출력 파일명을 지정할 수도 있습니다.

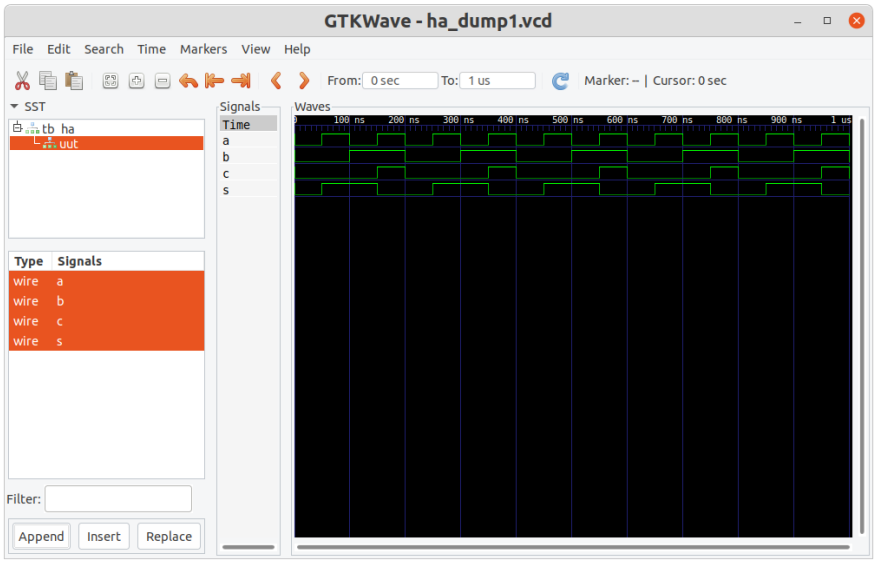
```
$ iverilog tb_ha.v ha.v -o ha
```

시뮬레이터를 실행하면 파형을 ha_dump1.vcd라는 파일로 덤프한다고 가정하면, 다음과 같이 gtkwave를 이용하여 확인할 수 있습니다.

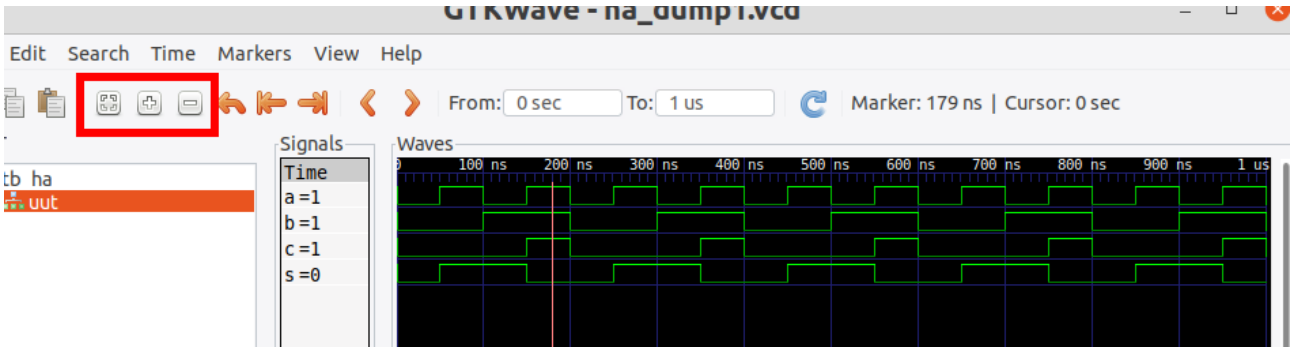
```
$ gtkwave ha_dump1.vcd &
```

실제 실행 예시:

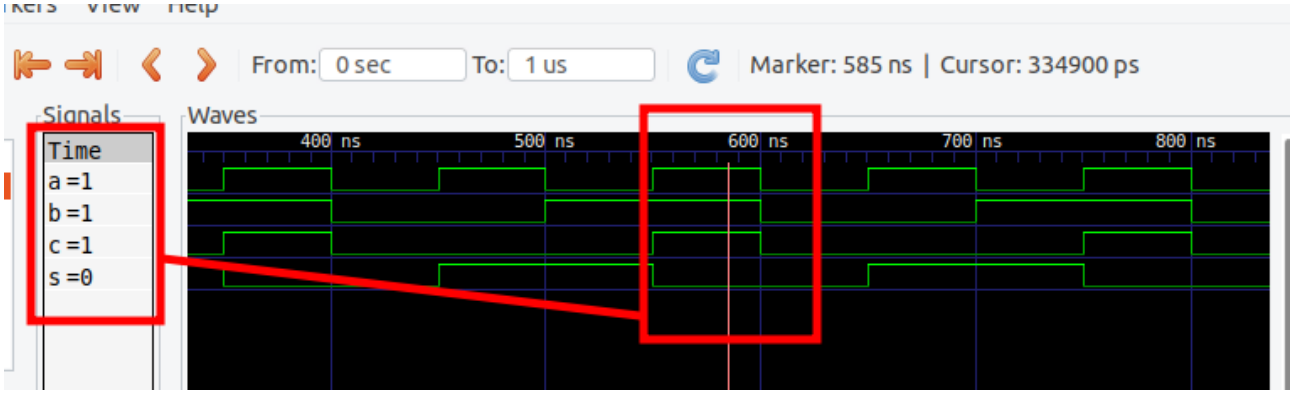
다음 그림은 입출력 신호 a, b, c, s를 선택하여 시각화 하고자 하는 신호 리스트에 append한 모습입니다.



파형의 모습은 좌측 상단의 버튼(빨간색 상자로 강조)을 이용하여 확대/축소할 수 있습니다. + 왼쪽에 있는 버튼은 현재 창 크기에 맞게 파형 폭을 조정합니다.



그리고 파형에서 특정 타이밍을 클릭하여 해당 타이밍의 신호 값을 확인할 수 있습니다. 다음 그림은 입력 a와 b가 모두 1일 때의 신호를 나타내고 있습니다. 반가산기(half adder)의 출력 c(carry)와 s(sum)은 각각 1과 0을 출력하고 있는 모습을 확인할 수 있습니다.



연습문제

1. C는 1비트 입력 A와 B의 논리 연산에 대한 출력을 나타냅니다. 진리표와 이에 대응하는 Verilog 코드(assign문)를 완성하세요. (논리 연산기호 •는 AND, +는 OR, ⊕는 XOR을 나타냅니다. 그리고 NOT은 A'나 $\bar{A}$ 등으로 나타냅니다.)

C=A • B

A	B	C
0	0	0
1	0	0
0	1	0
1	1	1

assign c = a & b;

C=A+B

A	B	C

0	0	
1	0	
0	1	
1	1	

assign c = a | b;

C=A⊕B

A	B	C
0	0	
1	0	
0	1	
1	1	

assign c = a ^ b;

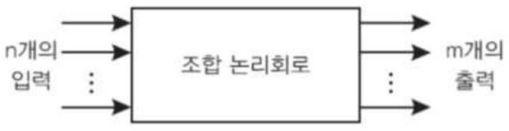
C=A'

A	C
0	
1	

assign c = ~a;

2. 조합회로(combinational circuit)와 순차회로(sequential circuit)의 차이점을 간단히 서술하세요.

조합회로는 기억 소자(플립플롭, 레지스터, 메모리 등등)를 포함하고 있지 않으므로, 이전 입력과 관계없이 현재의 입력으로부터 출력 값이 곧바로 결정됩니다. 대표적으로 CPU에 있는 ALU(Arithmetic and Logical Unit)가 있습니다.

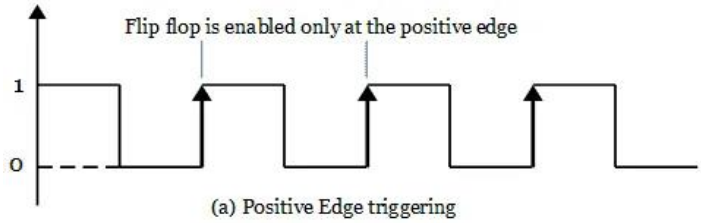


순차회로는 기억 소자가 있어서 이전의 입력이(그리고 현재의 입력이) 현재의 출력에 영향을 미칩니다. 즉, 상태(state, status)값을 가지는 회로라고 볼 수 있습니다.

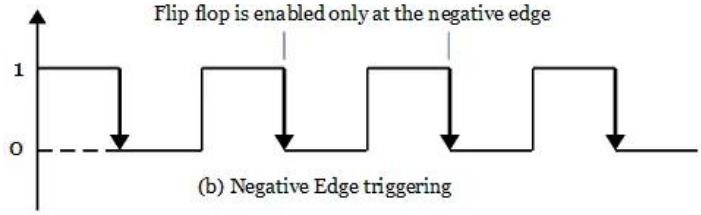
[그림 9-1] 순차 논리회로의 블록도

이런 순차회로에는 입력되는 클럭 신호에 동기(synchronization)되어 동작하는 동기식 순차회로와 클럭이 없는 비동기 순차회로(예를 들어 래치latch)가 있습니다.

클럭이 있으면 여러 하드웨어 모듈들이 클럭 신호에 맞춰 동작하고, 출력과 레지스터 값도 클럭에 맞춰 업데이트 되므로 하드웨어가 더욱 안정적으로 동작하고 설계도 더욱 쉬워집니다. 클럭의 정확히 어떤 순간에 입력신호를 반영하여 회로가 동작하고 업데이트 해야 하는지도 결정해야 합니다. 방식에 따라 에지 트리거(edge trigger)와 레벨 트리거(level trigger)가 있습니다. 레벨 트리거는 클럭 신호가 1(High)인지 0(Low)인지만을 보고 판단하여 동작합니다. 반면, 에지 트리거는 클럭의 상승(rising) 또는 하강(falling) 에지가 발생하는 순간에 해당 사이클의 동작을 수행합니다. 여기서 말하는 에지(edge)란 그림과 같이 신호가 전이되는 순간을 의미합니다. low에서 high로 상승하는 순간을 상승 에지, high에서 low로 하강하는 순간을 하강 에지라고 합니다.



(a) Positive Edge triggering



(b) Negative Edge triggering

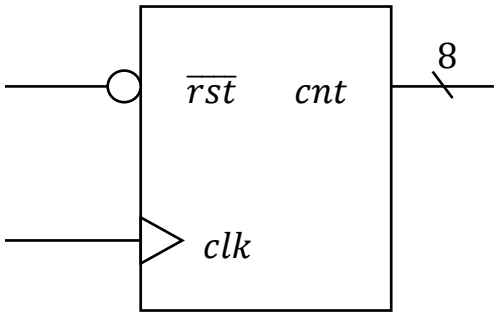
트리거링 방식 지정은 Verilog HDL에서도 문법적으로 지원하니 개념을 잘 이해해두시면 좋습니다. 예를 들어, 다음 코드는 8비트 카운터를 나타냅니다. always 문 다음에 있는 괄호를 주목하세요. 괄호 안에 들어가는 신호들의 리스트를 sensitivity list라고 합니다. 지정된 신호들의 변화를 감지해서 감지될 때마다 always 구문 안에 있는 코드를 실행하는 것입니다. 예제의 경우에는 posedge clk가 인수로 들어가 있는 모습을 볼 수 있습니다. 클럭 신호 clk의 변화를 감지하겠다는 의미입니다. 이때 앞에 붙은 posedge가 트리거링 방식을 의미합니다. posedge는 상승 에지를 가리킵니다. 반대로 negedge는 하강 에지를 의미합니다. 클럭 신호 clk의 상승 에지가 발생할 때마다 always 구문에 있는 동작이 실행될 것입니다. 리셋 신호인 rst가 LOW인지 검사하고, LOW가 아니면 카운터 값을 1씩 증가시킵니다.

```
module counter8(clk, rst, cnt);
    input clk, rst;
```

```
    output reg [7:0] cnt;

    always @(posedge clk) begin
        if (!rst)
            cnt <= 0;
        else
            cnt <= cnt + 1;
    end
endmodule
```

한편 rst의 검사는 clk에 맞춰 동기적으로 이뤄집니다. 그런 의미에서 위 코드와 같은 구성은 동기적(synchronous) 리셋이라고 할 수 있습니다. 또한 리셋 신호 rst가 HIGH (1)가 아니라 LOW (0)일 때 활성화된다는 점도 주목할 지점입니다. 이처럼 신호가 HIGH가 아니라 LOW일 때 동작하는 경우를 보고 active-low라고 부릅니다. 반대로, HIGH일 때 활성화되면 active-high라고 합니다. 회로도에서 신호에 NOT을 의미하는 bar가 붙어있거나 입력 신호에 방울이 붙어있는 경우에 해당 신호는 active-low라고 이해하시면 됩니다.



클럭 속도가 빠르다(클럭 주파수가 높다)는 것은 그만큼 회로가 빠르게 동작하고 빠르게 업데이트 됨을 의미합니다. 즉, 클럭이 빠르면 주어진 시간 동안 더 많은 양의 작업을 할 수 있다는 뜻입니다. 한편 사이클 내에 수행해야 하는 연산의 양 대비 사이클 주기가 너무 짧으면 타이밍 위반이 발생할 수 있습니다. 아무리 논리적으로 문제없는 코드를 작성해도 물리적인 특성 때문에 정상적인 출력을 보장할 수 없게 됩니다. 고속 회로를 설계하는 것이 까다로운 이유입니다.

이처럼 하드웨어를 설계할 때는 프로그래밍 언어라는 형식을 사용하더라도 하드웨어가 가진 특성을 항상 염두에 두고 주의 깊게 코드를 작성해야 합니다.

3. Behavioral modeling과 data-flow modeling, structural modeling에 대해 간단히 서술하세요(설명과 함께 교재나 웹에서 예시 코드를 가져와도 좋습니다).

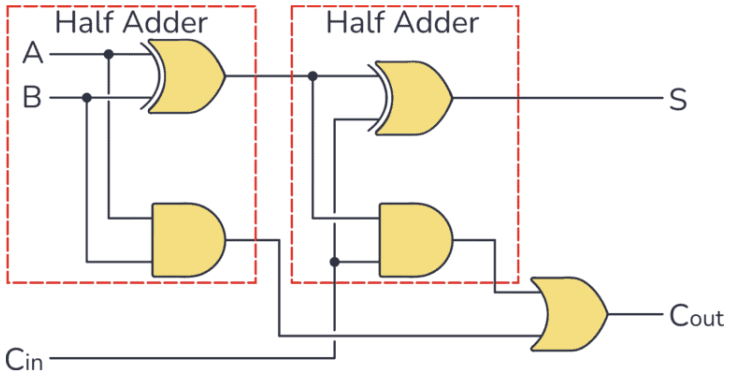
Behavioral modeling(행위적 모델링)은 functional(기능) 또는 알고리즘 레벨의 기술 방식입니다. 소프트웨어 코드를 짜듯이 하드웨어의 동작을 기술합니다. initial이나 always 구문을 주로 사용하는 방식입니다.

Dataflow modeling은 입력과 출력 사이의 신호 흐름을 연산자를 사용하여 직접적으로 나타내는 방식입니다. assign 문을 이용하여 조합회로를 기술하는 방식이라고 이해하시면 좋을 것 같습니다.

Structural modeling은 모듈들을 조합하여 더 큰 모듈을 정의하는 방식입니다. 모듈 안에 들어가

는 더 작은 모듈들을 서브 모듈(sub module)이라고 부릅니다. 그리고 이런 모듈들의 인스턴스를 정의하는 것을 instantiation이라고 말합니다. 작은 모듈들을 블록 조립하듯이 조합해서 큰 시스템을 구성한다는 점에서 구조적 모델링이라고 합니다.

사실 코딩 스타일이 달라진다고 해서 전혀 다른 하드웨어로 합성되는 것은 아닙니다(물론 합성기에 따라 더 잘 최적화되는 코딩 스타일이 있을 수는 있습니다). HDL 코드의 결과물은 결국 하드웨어 회로라는 점을 이해해야 합니다. 전가산기(full adder)를 예로 들어봅시다. 전가산기는 반가산기 두 개를 이용하여 structural modeling으로 구성할 수도 있지만, dataflow modeling으로도 구성할 수 있습니다.



즉, 다음 두 버전의 전가산기는 모두 같은 하드웨어로 합성된다는 것입니다.

```
// Structural Modeling
module fa(a, b, cin, s, cout);
    input a, b, cin;
    output s, cout;
    wire s1, c1;
    wire c2;

    ha h1(a, b, s1, c1);
    ha h2(.a(s1), .b(cin), .sum(s), .carry(c2));
    assign cout = c1 | c2;
endmodule

// Dataflow Modeling
module fa(a, b, cin, s, cout);
    input a, b, cin;
    output s, cout;
    wire s1, c1;
    wire c2;

    assign s1 = a ^ b;
    assign c1 = a & b;
    assign s = s1 ^ cin;
    assign c2 = s1 & cin;
    assign cout = c1 | c2;
endmodule
```

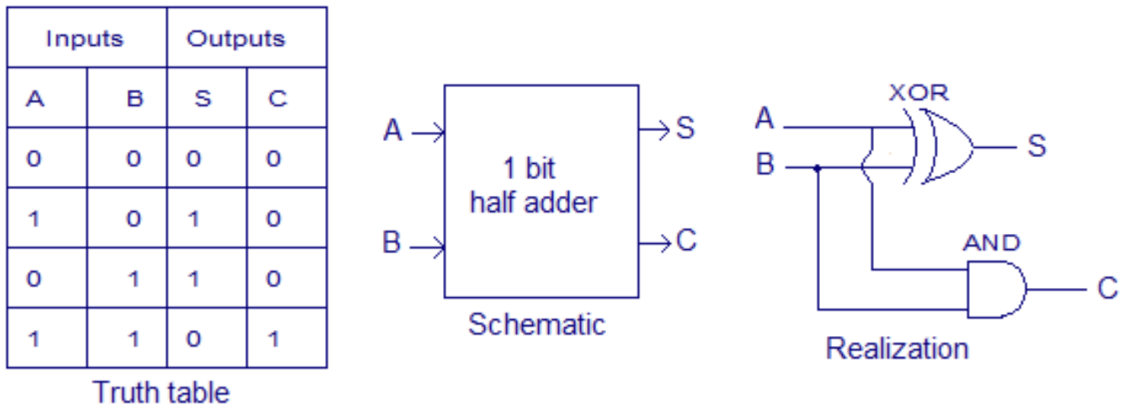
4. wire와 reg의 차이점에 대해 간단히 서술하세요.

Verilog의 자료형에는 크게 net형과 variable형이 있습니다. 둘은 서로 다른 하드웨어 구조를 표현(representation)하기 위해 사용되는데요, net형은 하드웨어 소자 간의 물리적인 연결(전선)을 나타냅니다. 일반적인 변수들과 달리 값을 저장하는 능력은 없습니다. net형에는 대표적인 예시로 wire가 있습니다. 어떤 모듈에 들어가고 나가는 값은 연속적으로 변화하기 때문에 입력 포트와 출력 포트는 (별도로 reg라고 선언하지 않는 이상) 기본적으로 wire형입니다.

5. initial과 always의 차이점에 대해 간단히 서술하세요.

initial은 시뮬레이션을 위한 구문으로, 처음 시작할 때 한 번만 실행됩니다. 시뮬레이션용이기 때문에 하드웨어로 합성되지는 않습니다. initial문과 다르게 always문은 계속해서 실행됩니다. always문의 괄호를 sensitivity list라고 부릅니다. always문은 sensitivity list에 들어가는 신호들의 변화가 감지될 때마다 실행됩니다. initial문과 다르게 always문은 하드웨어로 합성 가능합니다.

6. 실습: 다음 그림은 반가산기(half adder)의 진리표와 회로도입니다. 그림을 참고하여 반가산기의 design ha.v와 그 테스트벤치인 tb_ha.v를 작성하고, iverilog로 시뮬레이션 해봅시다. 그리고 gtkwave를 통해 파형을 직접 확인해봅시다.



참고: 파형을 파일로 덤프(dump)하려면 코드에 해당 구문을 추가하면 됩니다.

```
initial begin
  $dumpfile("ha_dump1.vcd");
  $dumpvars(0, tb_ha);
end
```

시스템 태스크 $dumpfile은 지정된 이름(ha_dump1.vcd)으로 파형 파일을 출력(dump)하라는 뜻이고, $dumpvars를 통해 모니터링하고자 하는 모듈을 지정합니다. 테스트벤치 모듈 이름이 tb_ha라면 tb_ha를 전달하면 됩니다.

7. 심화: 반가산기 모듈 2개를 이용하여 전가산기를 설계해봅시다. 6번 문제와 마찬가지로 gtkwave를 통해 입출력 파형을 확인해봅시다. (Hint: structural modeling)