KYUNGPOOK NATIONAL UNIVERSITY

COMP311-1 Logic Circuit Design Chap. 12

Instructor: Taigon Song (송대건)

2019 Fall

Possible Logic Functions of Two Variables

- There are 16 possible functions of 2 input variables:
  - in general, there are 2**(2**n) functions of n inputs

한 게이트로 나타낼 수 없을때 쓸 함수가 필요

X	Y	16 possible functions (F0–F15)															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
		0	X and Y	X	Y	X xor Y	X or Y	X nor Y not (X or Y)	X = Y	not Y	not X	X nand Y not (X and Y)	1				

→ What if I need a custom function?

12.1 UDP Basics

What UDP (user-defined primitive) is

- Verilog's built-in primitives
  - Such as and, nand, or, nor, xor, and xnor
- Designers can build their own custom-built primitives in a design
  - We call these "User-Defined Primitives" (UDP)
  - Self-contained, and do not instantiate other modules or primitives
  - Similar as gate-lv primitives
  - Two types of UDPs
    - Combinational UDPs – when output is solely determined by combinational logic
    - Sequential UDPs – when output takes current inputs and current outputs for next outputs (e.g., latches and flipflops)

12.1 UDP Basics

input a,b,c (이렇게 써야함) , input[3:0] (불가능), output은 1bit 하나만 가능
z는 다루지 않음 x로 취급됨, 모듈 안에 선언 불가

12.1.2 UDP Rules

- UDPs follow certain rules:
  1. UDPs can take only scalar input terminals (1 bit). However, multiple input terminals are permitted
  2. UDPs can have only one scalar output terminal (1 bit). The output terminal must always appear first in the terminal list. Multiple output terminals are not allowed
  3. The output terminal is declared with the keyword “output”. In sequential UDPs, the output must also be declared as “reg”.
  4. The inputs are declared with the keyword input.
  5. The state in a sequential UDP can be initialized with an initial statement (optional). A 1-bit value is assigned to the output
  6. The state table entries can contain values 0, 1, or x. UDPs do not handle z values. ‘z’ values are treated as ‘x’ values
  7. UDPs are defined at the same level as modules. UDPs cannot be defined inside modules. UDPs are instantiated exactly like gate primitives
  8. UDPs do not support inout ports

12.2 Combinational UDPs

udp 설계시 다른 모듈과 구분 위해 앞에 udp_ 붙인다.
테이블 만들어서 설계

12.2.1 Combinational UDP Definition

Example 12-1 Primitive udp_and

```
//Primitive name and terminal list
primitive udp_and(out, a, b);

//Declarations
output out; //must not be declared as reg for combinational UDP
input a, b; //declarations for inputs.

//State table definition; starts with keyword table
table
   //The following comment is for readability only
   //Input entries of the state table must be in the
   //same order as the input terminal list.
  // a   b   :   out;
     0   0   :   0;
     0   1   :   0;
     1   0   :   0;
     1   1   :   1;

endtable //end state table definition

endprimitive //end of udp_and definition
```

Example 12-2 ANSI C Style UDP Declaration

```
//Primitive name and terminal list
primitive udp_and(output out,
                  input a,
                  input b);
--
--
endprimitive //end of udp_and definition
```

12.2 Combinational UDPs

12.2.2 State Table Entries

- If a certain combination occurs and the corresponding entry is not in the table, the output is x
- Use of default x output is frequently used in commercial models

테이블에 없는 인풋인 경우 아웃풋 x 출력

Example 12-3 Primitive udp_or

```
primitive udp_or(out, a, b);

output out;

input a, b;

table
  //  a   b   :   out;
      0   0   :   0;
      0   1   :   1;
      1   0   :   1;
      1   1   :   1;
      x   1   :   1;
      1   x   :   1;
endtable

endprimitive
```

Note: What's the output when (a=x, b=0)?

12.2 Combinational UDPs

12.2.3 Shorthand Notation for Don't Cares

- The '?' symbol represent don't care conditions
- A ? Symbol is automatically expanded to 0, 1, or x.
- The udp_or can be described as the below

```
primitive udp_or(out, a, b);

output out;
input a, b;

table
  //  a   b   :   out;
      0   0   :   0;
      1   ?   :   1; //? expanded to 0, 1, x
      ?   1   :   1; //? expanded to 0, 1, x
      0   x   :   x;
      x   0   :   x;
endtable

endprimitive
```

12.2 Combinational UDPs

12.2.4 Instantiating UDP Primitives

- UDPs are instantiated exactly like Verilog gate primitives

Example 12-4 Instantiation of udp Primitives

```
// Define a 1-bit full adder
module fulladd(sum, c_out, a, b, c_in);

// I/O port declarations
output sum, c_out;
input a, b, c_in;

// Internal nets
wire s1, c1, c2;

// Instantiate logic gate primitives
xor (s1, a, b);//use Verilog primitive
udp_and (c1, a, b); //use UDP

xor (sum, s1, c_in); //use Verilog primitive
udp_and (c2, s1, c_in); //use UDP

udp_or  (c_out, c2, c1);//use UDP

endmodule
```

12.2 Combinational UDPs

12.2.5 Example of a Combinational UDP

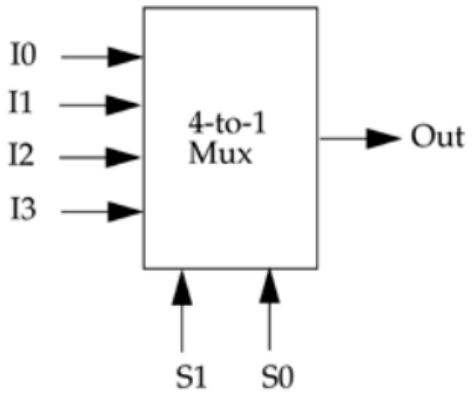
- An example of a 4-to-1 MUX in combinational UDP
  - Note
    - State table becomes large quickly as # of input increases
    - UDPs offer convenient feature to implement a function whose truth table is known, without extracting actual logic

S1	S0	Out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Example 12-5 Verilog Description of 4-to-1 Multiplexer with UDP

```
// 4-to-1 multiplexer. Define it as a primitive
primitive mux4_to_1 ( output out,
                      input i0, i1, i2, i3, s1, s0);
table
  //  i0   i1   i2   i3, s1   s0   : out
      1    ?    ?    ?    0    0    : 1 ;
      0    ?    ?    ?    0    0    : 0 ;
      ?    1    ?    ?    0    1    : 1 ;
      ?    0    ?    ?    0    1    : 0 ;
      ?    ?    1    ?    1    0    : 1 ;
      ?    ?    0    ?    1    0    : 0 ;
      ?    ?    ?    1    1    1    : 1 ;
      ?    ?    ?    0    1    1    : 0 ;
      ?    ?    ?    ?    x    ?    : x ;
      ?    ?    ?    ?    ?    x    : x ;
endtable

endprimitive
```

udp는 꼭 필요한경우가 아니면
잘 사용하지 않는다.
더 쉽게 나타낼 수 있기 때문



12.3 Sequential UDPs

Sequential UDPs vs. Combinational UDPs

- Sequential UDPs 시퀀셜 udp는 안쓰는걸 추천
  - The output of sequential UDP is always reg
  - Initial statement can be used to initialize output of sequential UDPs
  - The format of the state table is slightly different
  - There are three sections in a state table entry
    - Inputs, current state, and next state (separated by colon ":")
  - Input specification of state table entries can be in terms of input levels or edge transitions
  - The current state is the current value of the output register
  - The next state is computed based on inputs and the current state. The next state becomes the new value of the output register
  - All possible combinations of inputs must be specified to avoid unknown output
  - Two sequential UDPs
    - Level-sensitive sequential UDP/edge-sensitive sequential UDP

12.3 Sequential UDPs

12.3.1 Level-Sensitive Sequential UDPs

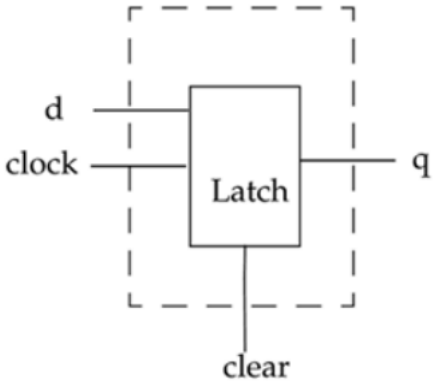
- Basically, “Latches”

Example 12-7 Verilog Description of Level-Sensitive UDP

```
//Define level-sensitive latch by using UDP.
primitive latch(q, d, clock, clear);

//declarations
output q;
reg q; //q declared as reg to create internal storage
input d, clock, clear;

//sequential UDP initialization
//only one initial statement allowed
initial
    q = 0; //initialize output to value 0

//state table
table
  //d clock clear : q : q+ ;

    ?   ?   1     : ? : 0 ; //clear condition;
                            //q+ is the new output value

    1   1   0     : ? : 1 ; //latch q = data  = 1
    0   1   0     : ? : 0 ; //latch q = data  = 0

    ?   0   0     : ? : - ; //retain original state if clock = 0
endtable

endprimitive
```

No change 그대로 냅둠

Figure 12-3. Level-Sensitive Latch with clear



Example 12-8 ANSI C Style Port Declaration for Sequential UDP

```
//Define level-sensitive latch by using UDP.
primitive latch(output reg q = 0,
                input d, clock, clear);
--
--
--
endprimitive
```

12.3 Sequential UDPs

12.3.2 Edge-Sensitive Sequential UDPs

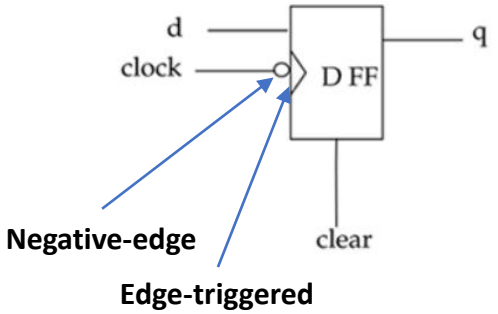
- Basically, Flip-flops

Example 12-9 Negative Edge-Triggered D-flipflop with clear

```
//Define an edge-sensitive sequential UDP;
primitive edge_dff(output reg q = 0,
                   input d, clock, clear);


table
    //  d clock clear : q : q+  ;

        ?   ?     1    : ? : 0 ; //output = 0 if clear = 1
        ?   ?     (10) : ? : - ; //ignore negative transition of clear

        1   (10)  0    : ? : 1 ; //latch data on negative transition of
        0   (10)  0    : ? : 0 ; //clock

        ?   (1x)  0    : ? : - ; //hold q if clock transitions to
unknown
                                 //state

        ?   (0?)  0    : ? : - ; //ignore positive transitions of clock
        ?   (x1)  0    : ? : - ; //ignore positive transitions of clock

        (??) ?    0      : ? : - ; //ignore any change in d when clock
                                   //is steady
endtable

endprimitive
```

(αβ): transition a - >b 로 변할때 엣지를 나타냄

Figure 12-4. Edge-Sensitive D-flipflop with clear




```
table
...
        (01) (10)  0    : ? : 1 ;
entry
...
endtable
```

Q) Is this possible?

12.3 Sequential UDPs

12.3.3 Example of a Sequential UDPs

- A 4-bit binary ripple counter

Example 12-10 T-Flipflop with UDP

```
// Edge-triggered T-flipflop
primitive T_FF(output reg q,
               input clk, clear);

//no initialization of q; TFF will be initialized with clear signal

table
  //  clk   clear :   q   : q+ ;
      //asynchronous clear condition
      ?       1   :   ?   : 0 ;

      //ignore negative edge of clear
      ?     (10)  :   ?   : - ;

      //toggle flipflop at negative edge of clk
      (10)    0   :   1   : 0 ;
      (10)    0   :   0   : 1 ;

      //ignore positive edge of clk
      (0?)    0   :   ?   : - ;
endtable
endprimitive
```

Example 12-11 Instantiation of T_FF UDP in Ripple Counter

```
// Ripple counter
module counter(Q , clock, clear);

// I/O ports
output [3:0] Q;
input clock, clear;

// Instantiate the T flipflops
// Instance names are optional
T_FF tff0(Q[0], clock, clear);
T_FF tff1(Q[1], Q[0], clear);
T_FF tff2(Q[2], Q[1], clear);
T_FF tff3(Q[3], Q[2], clear);

endmodule
```

UDP Table Shorthand Symbols

```
table
    //  d clock clear : q : q+  ;

        ?   ?     1   : ? : 0 ; //output = 0 if clear = 1
        ?   ?     f   : ? : - ; //ignore negative transition of clear

        1   f     0   : ? : 1 ; //latch data on negative transition of
        0   f     0   : ? : 0 ; //clock

        ?   (1x)  0   : ? : - ; //hold q if clock transitions to unknown
                                //state

        ?   p     0   : ? : - ; //ignore positive transitions of clock

        *   ?     0   : ? : - ; //ignore any change in d when
                              //clock is steady
endtable
```

Table 12-1. UDP Table Shorthand Symbols

Shorthand Symbols	Meaning	Explanation
?	0, 1, x	Cannot be specified in an output field
b	0, 1	Cannot be specified in an output field
-	No change in state value	Can be specified only in output field of a sequential UDP
r	(01)	Rising edge of signal
f	(10)	Falling edge of signal
p	(01), (0x) or (x1)	Potential rising edge of signal
n	(10), (1x) or (x0)	Potential falling edge of signal
*	(??)	Any value change in signal