Digital Design with Verilog

Verilog

Lecture 13: User-Defined Primitives

Learning Objectives

- Understand UDP definition rules and parts of a UDP definition.
- Define sequential and combinational UDPs.
- Explain instantiation of UDPs.

Introduction

- Built-in Primitives:
  - and, or, not, nand, nor, xor, xnor, buf.

- Verilog provides the ability to define custom-built primitives or User-Defined Primitives (UDP).

- These primitives are self-contained and do not instantiate other modules or primitives.

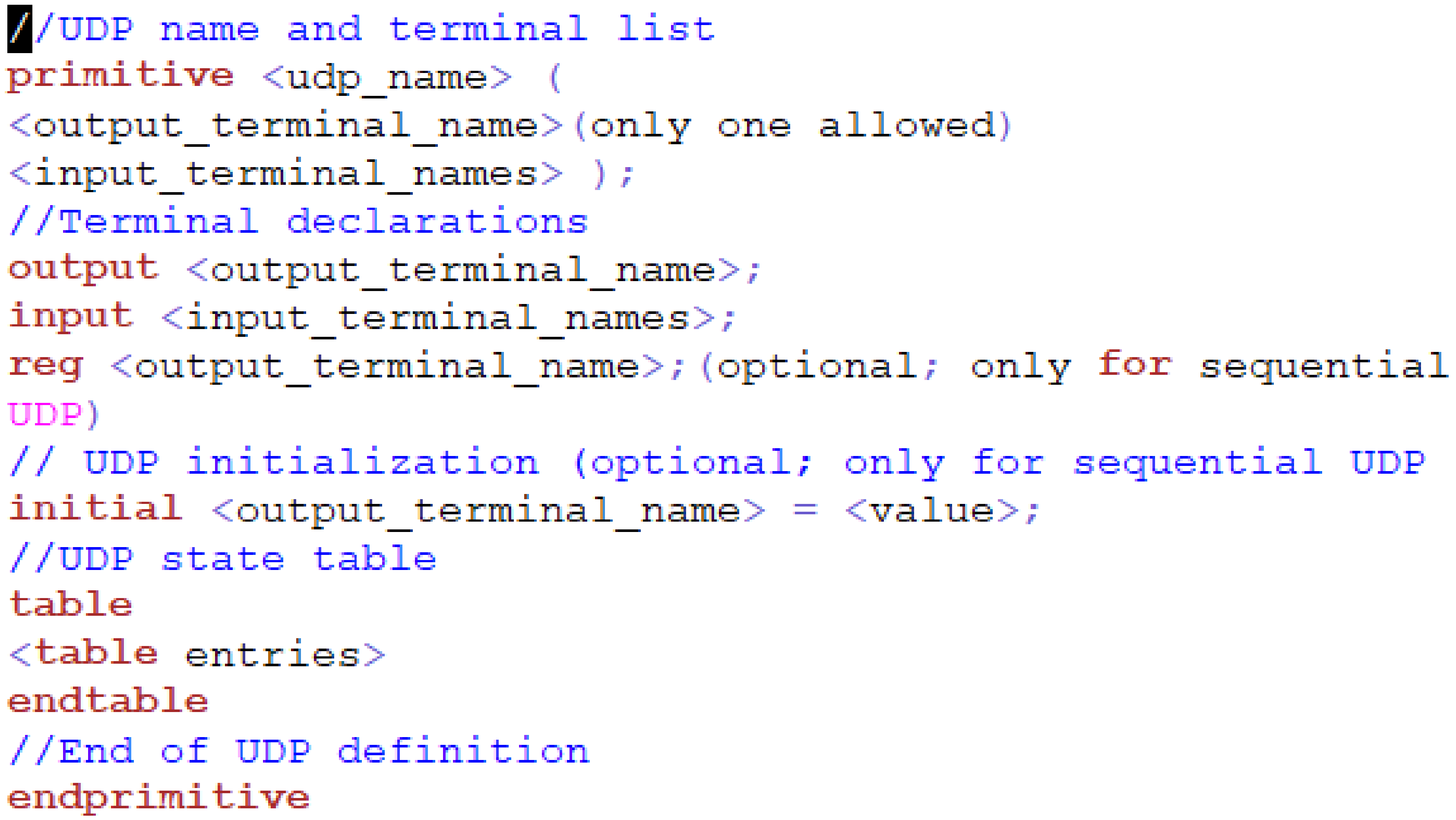
- UDPs are instantiated exactly like gate-level primitives.

Introduction

There are two types of UDPs: combinational and sequential.

- Combinational UDPs are defined where the output is solely determined by a logical combination of the inputs.
- Sequential UDPs take the value of the current inputs and the current output to determine the value of the next output.
  - The value of the output is also the internal state of the UDP.

Parts of UDP Definition



```
//UDP name and terminal list
primitive <udp_name> (
<output_terminal_name>(only one allowed)
<input_terminal_names> );
//Terminal declarations
output <output_terminal_name>;
input <input_terminal_names>;
reg <output_terminal_name>;(optional; only for sequential
UDP)
// UDP initialization (optional; only for sequential UDP
initial <output_terminal_name> = <value>;
//UDP state table
table
<table entries>
endtable
//End of UDP definition
endprimitive
```

UDP Rules

- UDPs can take only scalar input terminals (1 bit).
  - Multiple input terminals are permitted.
- 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.
- In the declarations section, the output terminal is declared with the keyword output. Since sequential UDPs store state, the output terminal must also be declared as a reg.

UDP Rules

- The inputs are declared with the keyword input.

- The state in a sequential UDP can be initialized with an initial statement.
  - This statement is optional. A 1-bit value is assigned to the output, which is declared as reg.

- The state table entries can contain values 0, 1, or x. UDPs do not handle z values. z values passed to a UDP are treated as x values.

UDP Rules

- UDPs are defined at the same level as modules.
- UDPs cannot be defined inside modules.
- They can be instantiated only inside modules.
- UDPs are instantiated exactly like gate primitives.
- UDPs do not support inout ports.

Combinational UDPs

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
```

Combinational UDPs

State Table Entries

- The <input#> values in a state table entry must appear in the same order as they appear in the input terminal list.
- Inputs and output are separated by a ":".
- A state table entry ends with a ";".
- All possible combinations of inputs, where the output produces a known value, must be explicitly specified. Otherwise, if a certain combination occurs and the corresponding entry is not in the table, the output is x.

Combinational UDPs

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
```

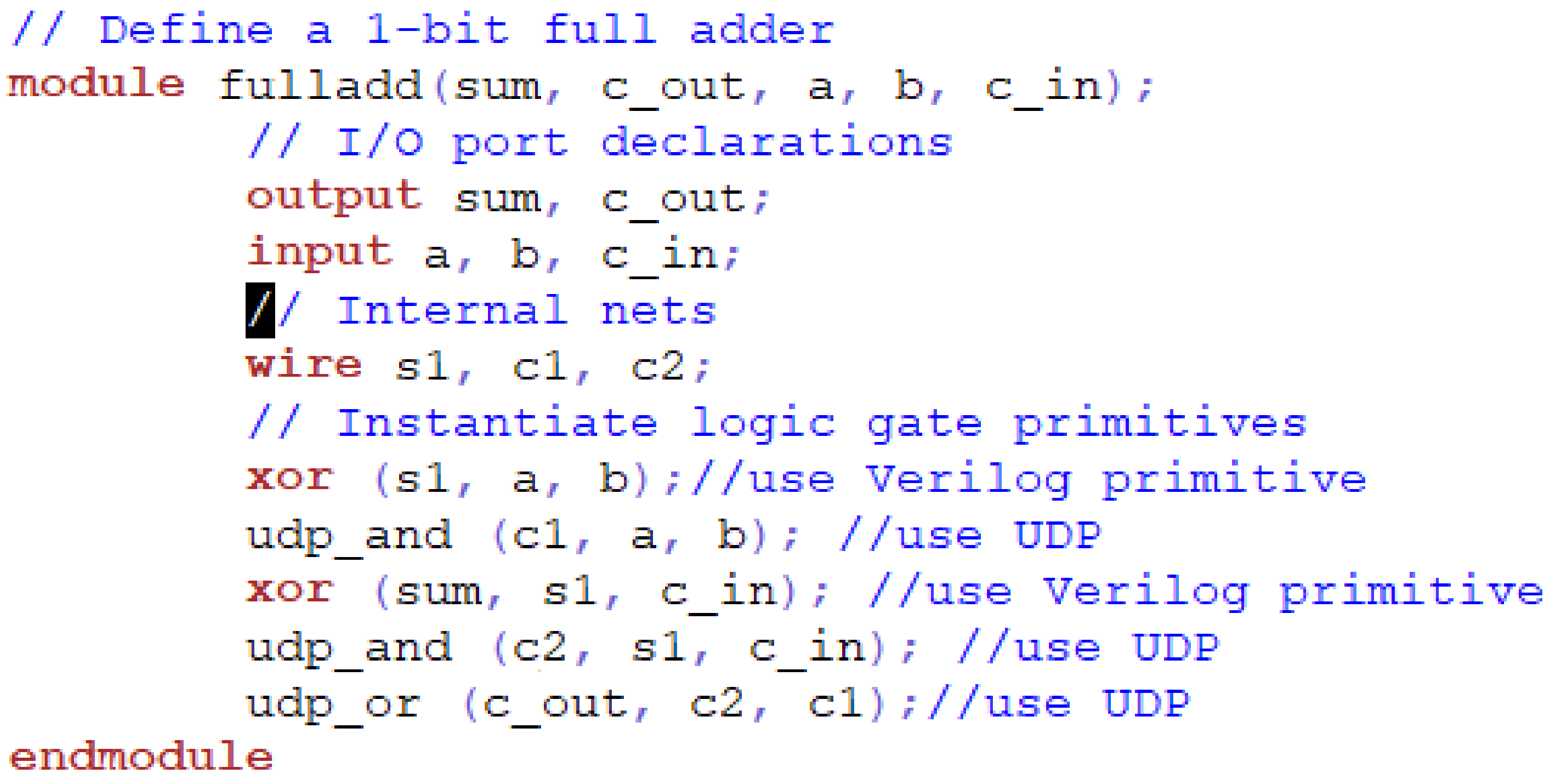
Combinational UDPs

Shorthand Notation for Don't Cares

```
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
```

Combinational UDPs

Instantiating 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
```

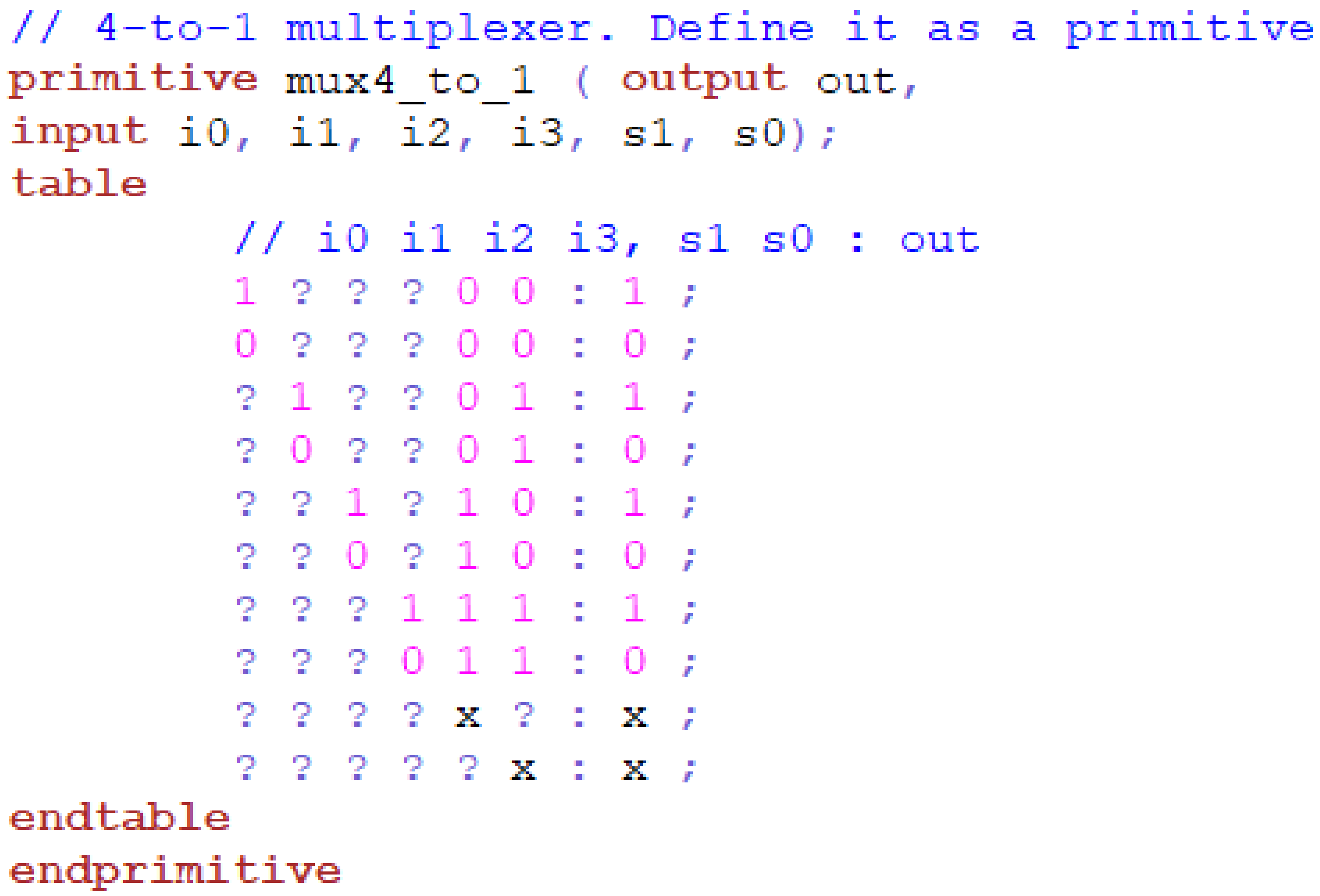
Combinational UDPs

```
primitive multiplexer(mux, control, dataA, dataB ) ;
output mux ;
input control, dataA, dataB ;
table
// control dataA dataB mux
0 0 0 : 0 ;
0 0 1 : 0 ;
0 1 0 : 1 ;
0 1 1 : 1 ;
1 0 0 : 0 ;
1 0 1 : 1 ;
1 1 0 : 0 ;
1 1 1 : 1 ;
endtable
endprimitive
```

If the input signal combinations are not fully specified, output may have X!! (e.g. 00X → output X)

Source: Chihhao Chao, NTU

Combinational UDPs



```
// 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
```

Sequential UDPs

Sequential UDPs have the following differences in comparison to combinational UDPs:

- The output of a sequential UDP is always declared as a reg.
- An initial statement can be used to initialize output of sequential UDPs.
- The format of a state table entry is slightly different.

```
<input1> <input2> ..... <inputN> : <current_state> :
<next_state>;
```

Sequential UDPs

- There are three sections in a state table entry: inputs, current state, and next state.

- The three sections are separated by a colon (:) symbol.

- The 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.

Sequential UDPs

- 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 values.

- If a sequential UDP is sensitive to input levels, it is called a level-sensitive sequential UDP.

- If a sequential UDP is sensitive to edge transitions on inputs, it is called an edge-sensitive sequential UDP.

Sequential UDPs

- Level-sensitive UDPs change state based on input levels.

```
//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
```

Sequential UDPs

- Edge-sensitive sequential UDPs change state based on edge transitions and/or input levels.

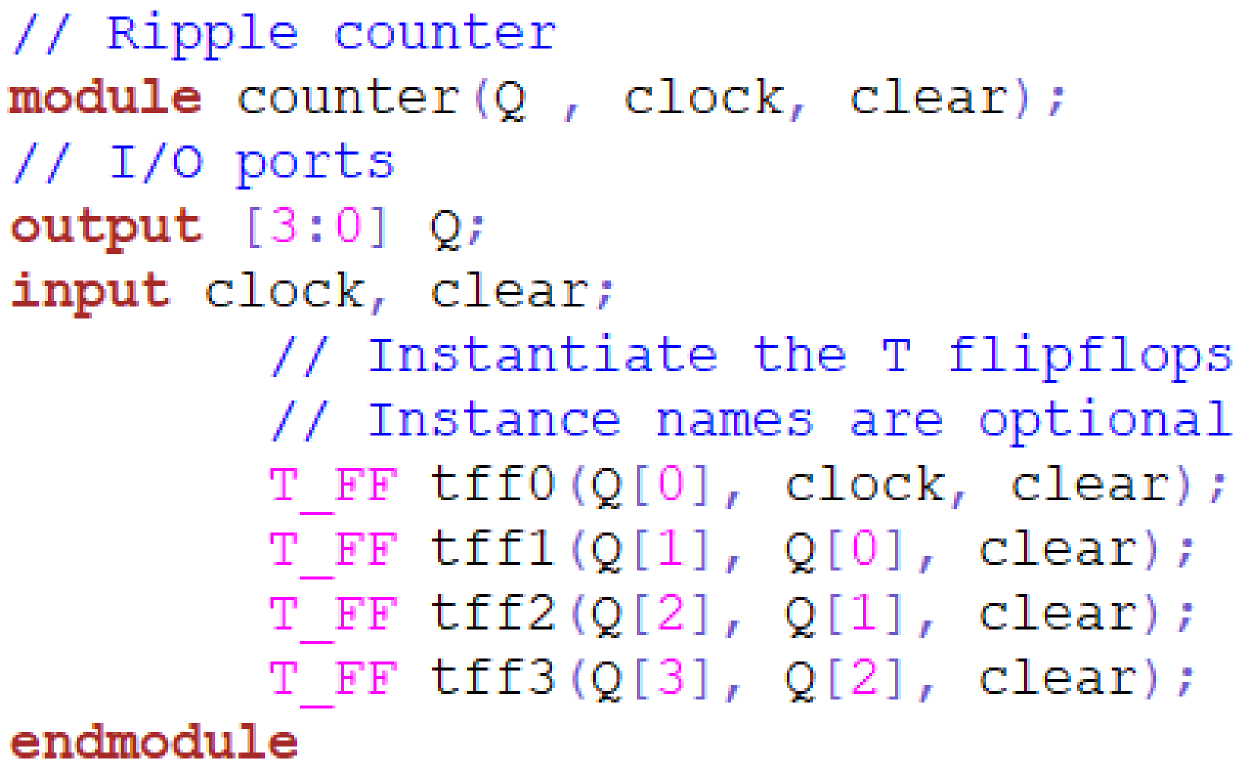
```
//Define an edge-sensitive sequential UDP;
primitive edge_dff(output reg q = 0,
input d, clock, clear);
table
// d clock clear : q : q+ ;
   ? ? 1 : ? : 0 ; //output = 0 if clear = 1
   ? ? (10): ? : - ; //ignore negative transition of clear
   1 (10) 0 : ? : 1 ; //latch data on negative transition 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
```

Sequential UDPs

```
// 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
```

Sequential UDPs

- To build the ripple counter with T-FFs, four T-FFs are instantiated in the 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
```

Abbreviations for UDP

Symbol	Interpretation	Explanation
?	0 or 1 or X	? means the variable can be 0 or 1 or x
b	0 or 1	Same as ?, but x is not included
f	(10)	Falling edge on an input
r	(01)	Rising edge on an input
p	(01) or (0x) or (x1) or (1z) or (z1)	Rising edge including x and z
n	(10) or (1x) or (x0) or (0z) or (z0)	Falling edge including x and z
*	(??)	All transitions
-	no change	No Change

References

- Chapter 12, Verilog HDL by Samir Palnitkar

Thank you