Design and Model Large VLSI System

林裕盛 Yu-Sheng Lin

Outline

- This slides does NOT discuss how to write Verilog.
- This slides focuses on high-level issues
  - Partitioning
  - Interfacing
  - Modelling
  - Verification
- It is about the *design pattern* and *best practice* for designing VLSI systems.

In realworld cases, you must think in high-level

- Research is NOT like IC contest, which provides detailed spec.

2.1 系統方塊圖

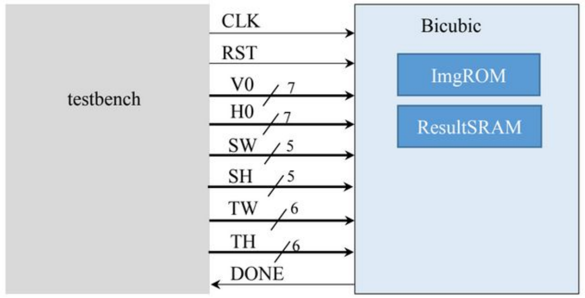
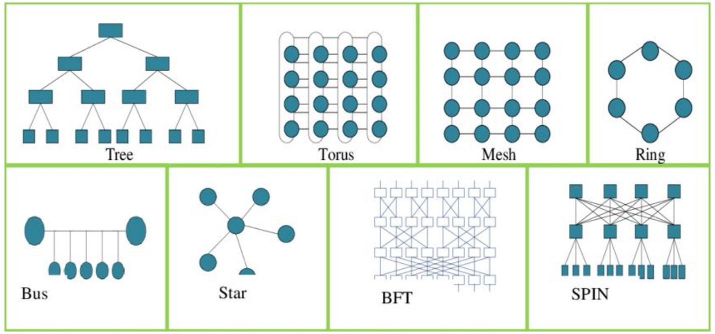


圖 1、系統方塊圖

IC Contest

Image processor with given spec

- In research, you have to explore large number of potential designs.
- Prematurely diving into Verilog wastes your time.



Research

Q: which one is good for my AI processor

In realworld cases, you must also think:

- If your boss ask you how long you need, and how do you progress, how do you let him/her know?
- If you are a PhD, and have 2 masters help you, can you speed up?
- How do you trust the code? Even if you graduate (in school) or are promoted (in a company).

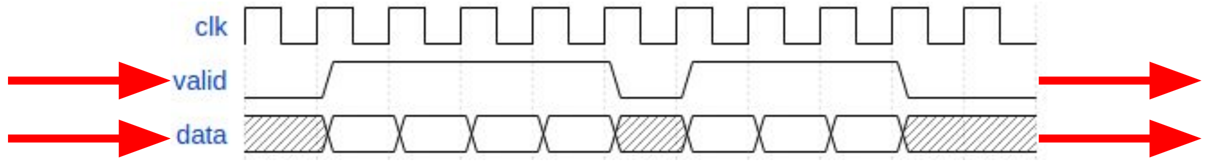
So, what is necessary?

- Test
  - Run by machine, not by you
- Unittest
  - Protect each piece of your code
- For this, you need good interfacing and partitioning

Interfacing: the less the better

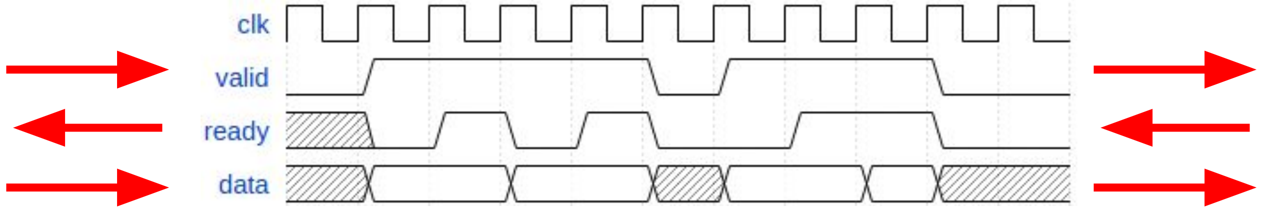
Personal feeling about how often each type appears

- Valid: used by AXI stream



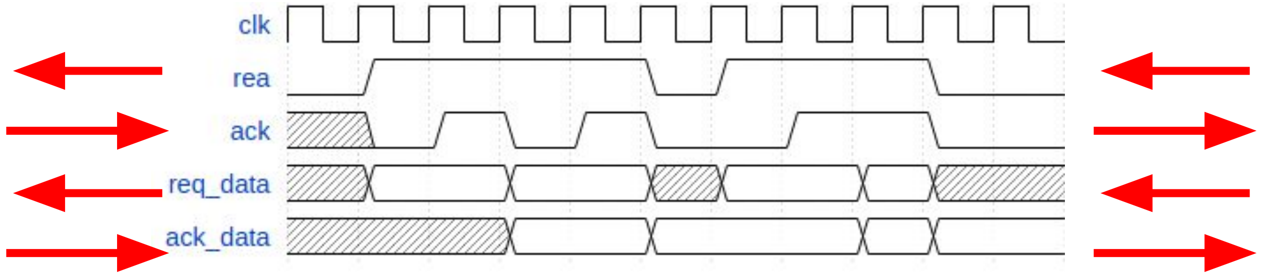
5%

- Valid/ready: used by AXI



90%

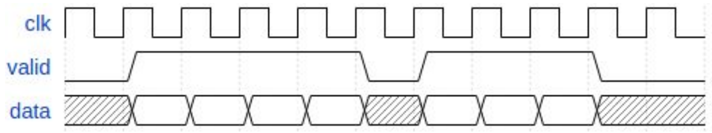
- Valid/ready (bidirectional): used by AHB



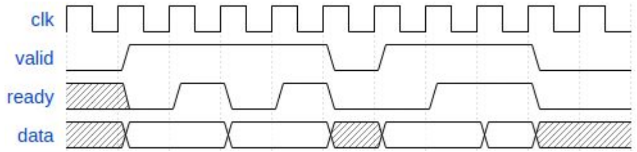
3%

Properties of the interfaces

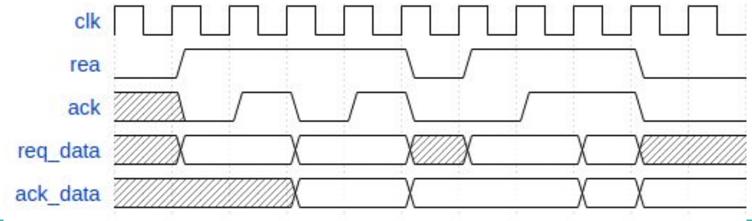
- The next stage shall always receive
  - Used when hardware resources is allocated



- The next stage can pause the output, most commonly used
  - Model the FIFO interface

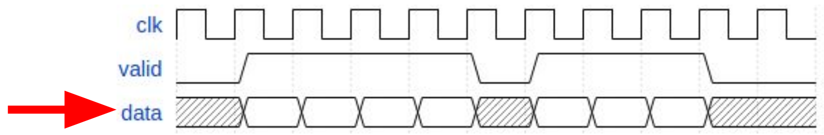


- Usually used when model cycle-accurate parts



Use struct to simplify interface for data ports

- This is suggested by Synopsys
- Make your Verilog much much cleaner



```
typedef struct packed {
   logic [1:0] a;
   AnotherStruct [2:0] b;
} TheStruct;
```

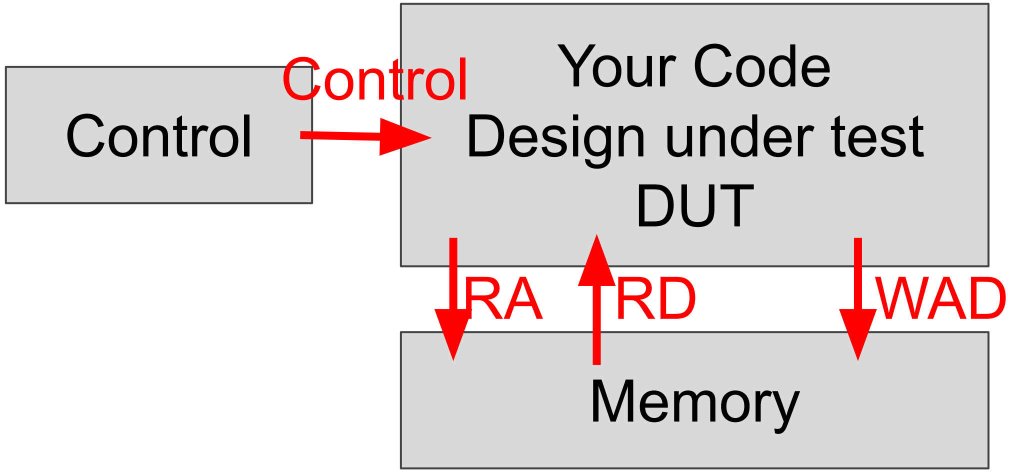
```
module A(
   input TheStruct s
);
endmodule
```

```
TheStruct data;
A a(
   .s(data)
);
```

Example: simple 1D CNN module

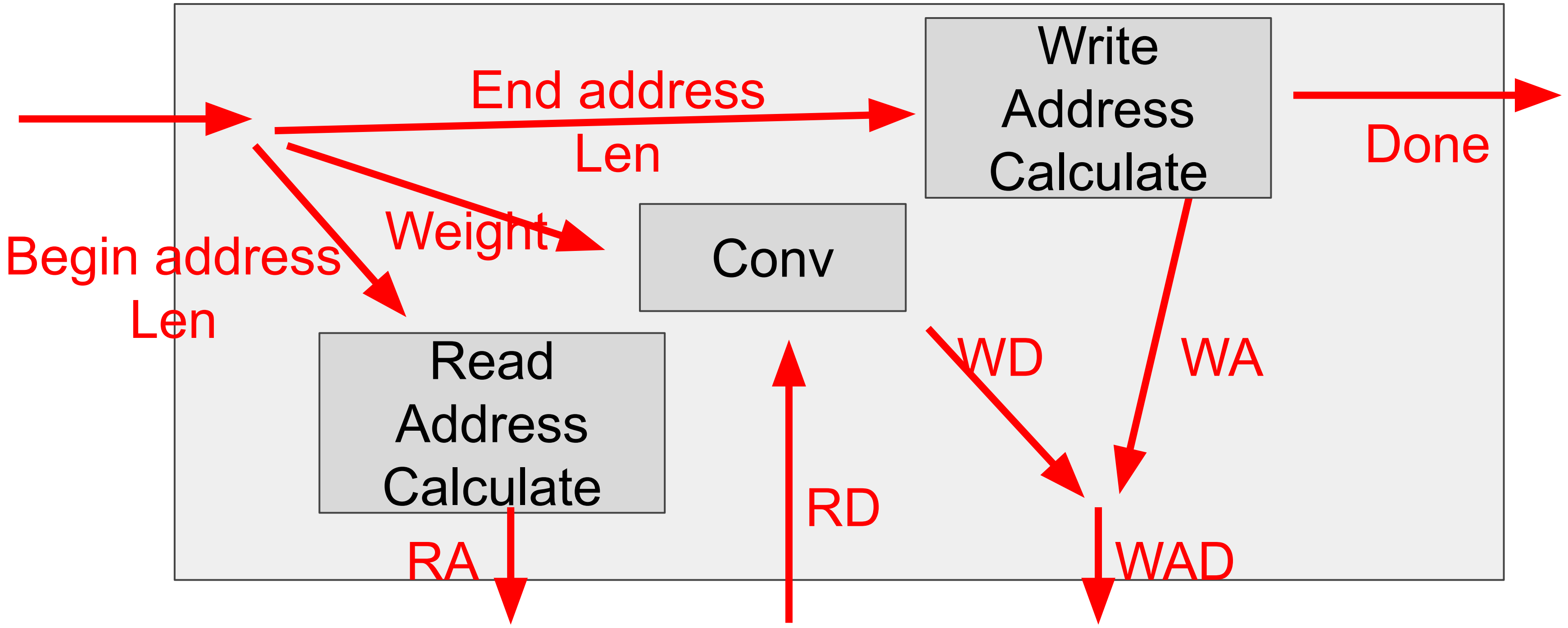
```
typedef logic [31:0] addr_t;
typedef struct {
  addr_t beg_addr;
  addr_t end_addr;
  logic [15:0][2:0] w;
} control_t;
```

- Control:
  - Begin address, end address, length, weight [3]
- Memory:
  - Read address (RA), Read data (RD), Write address+data (WAD)



Testable modularization

- Valid/Ready interfaces is used everywhere
- Individual module is testable



A good partition divide large design into testable subparts

Read Address Calculate =

```
def RdAddr(len, addr):
    ret = list()
    for i in range(len_):
        for j in range(3):
            ret.append(addr+i+j-1)
    return ret
```

Write Address Calculate =

```
def WrAddr(len, addr):
    ret = list()
    for i in range(len_):
        ret.append(addr+i)
    return ret
```

Conv =

```
def Conv(data):
    return sum(
        w[i]*data[i]
        for i in range(3)
    )
```

Practive by yourselves

- Write the I/O of every module in SystemVerilog
- Connect the toplevel module
- Note: don't need to implement the internal
- Think: how to test individual modules?