Bus

Zihao Yu

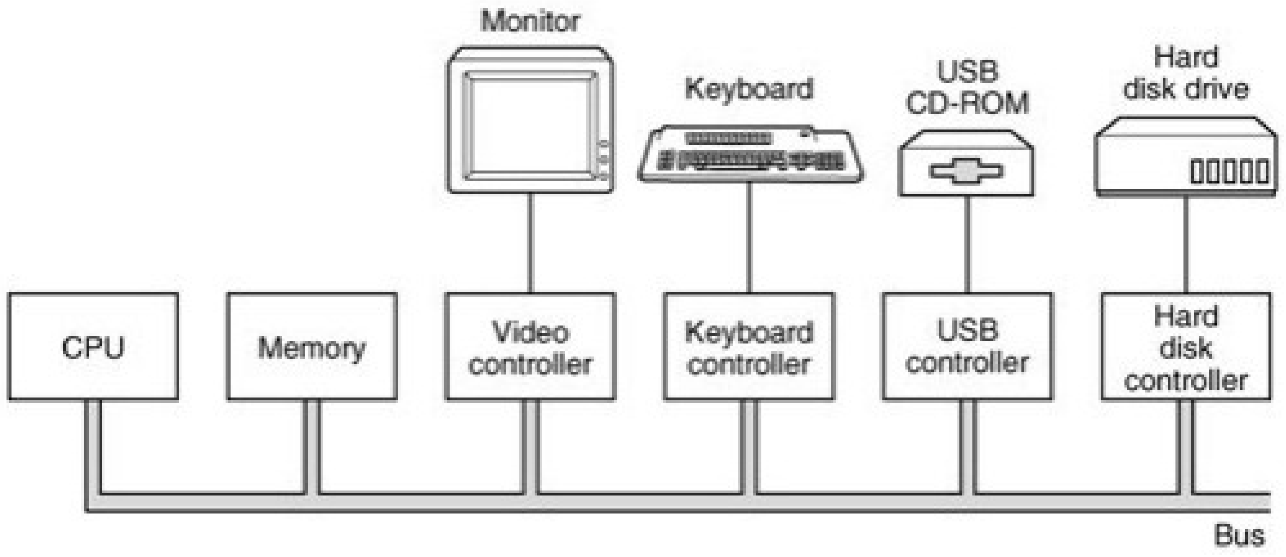
Institute of Computing Technology
Chinese Academy of Sciences

Introduction

Your CPU can communicate with some simple device models

- But in real hardware, device controllers are RTL models
- CPU should communicate with device in a hardware way

That is bus



Basic Concept

Bus = Protocol

Bus is a communication protocol

```
         +-----+            +-----+
         | CPU | <-------> | MEM |
         +-----+            +-----+
```

- The module which starts the communication is called `master`
  - CPU
- The module which repsonses to the communication is called `slave`
  - MEM
- Master and slave should reach the agreement about how to communicate with each other
  - When we design master and slave, we should use circuit to implement this aggrement

System Bus

Bus connecting between CPU and memory/devices

- We have already used DPI-C to emulate system bus
  - But it returns data immediately
- In the real hardware, it is not the case
  - It costs multiple cycles to get the data

This requires CPU to implement the following:

1. identify when the data returns from memory
2. wait until the data returns
   - after that, CPU can continue to execute the instruction
3. tell memory when CPU should fetch an instruction
   - When CPU is waiting, it should not issue any new fetch requests

Simple Bus (2)

Real chips usually adopt industry protocol like AXI

- It contains about 30 signals, which is complicated

For simplicity, we define a simple bus protocol called `SimpleBus`

- We will provide a bridge later to convert SimpleBus into AXI
  - Then integrate your CPU into SoC to communicate with other devices

Let us start with a simple case

Measure the Performance of CPU

Before starting to discuss bus, we first try to measure the performance of your CPU

TASK 1 - Computer IPC (Instruction Per Cycle) for your CPU

- Computer how many instructions and cycles your CPU costs to execute a program
- Then compute `inst / cycle`
  - It should be around `1`

You will measure IPC again after you implement bus

Fetch Instructions with Delay

The Protocol

Assume that the delay of accessing memory is 1 cycle

- CPU sends `raddr` to memory, and gets `rdata` in the next cycle

```
+-----+ raddr[log2(N)-1:0] ---> +-----+
| CPU | <---        rdata[31:0] | MEM |
+-----+                         +-----+
```

Both instruction fetch and load/store will access memory

- To distinguish between them, we refer the corresponding units as IFU and LSU
  - IFU = Instruction Fetch Unit, LSU = Load/Store Unit

For IFU, SimpleBus involves the following signals:

```
output [31:0] ifu_raddr,
input  [31:0] ifu_rdata,
```

Timing Diagram

```
                    --\ /----------\ /----------\ /----------\ /----------\ /--
          ifu_raddr    X 0x80000000 X     (1)    X 0x80000004 X            X
                    --/ \----------/ \----------/ \----------/ \----------/ \--
                    --\ /----------\ /----------\ /----------\ /----------\ /--
          ifu_rdata    X            X 0x00000413 X     (2)    X 0x80051137 X
                    --/ \----------/ \----------/ \----------/ \----------/ \--
```

Implement Waiting

```
                       Stage 1       Stage 2     Stage 1       Stage 2
                    --\ /----------\ /----------\ /----------\ /----------\ /--
          ifu_raddr    X 0x80000000 X     (1)    X 0x80000004 X            X
                    --/ \----------/ \----------/ \----------/ \----------/ \--
                    --\ /----------\ /----------\ /----------\ /----------\ /--
          ifu_rdata    X            X 0x00000413 X     (2)    X 0x80051137 X
                    --/ \----------/ \----------/ \----------/ \----------/ \--
```

We should let IFU know which stage it is at

- IFU should do different things in different stages

This can be implemented by `state machine` in digital circuit

Define two states - `idle` and `wait`

- At `idle` state: set `ifu_raddr` to the current PC, and switch to `wait` state
- At `wait` state: get `ifu_rdata` as an instruction to execute, and switch to `idle` state

Implement Waiting(2)

```
                        idle         wait         idle         wait
                  --\ /----------\ /----------\ /----------\ /----------\ /--
      ifu_raddr      X 0x80000000 X     (1)    X 0x80000004 X            X
                  --/ \----------/ \----------/ \----------/ \----------/ \--
                  --\ /----------\ /----------\ /----------\ /----------\ /--
      ifu_rdata      X            X 0x00000413 X     (2)    X 0x80051137 X
                  --/ \----------/ \----------/ \----------/ \----------/ \--
```

At `idle` stage, `ifu_rdata` is not a valid instruction

- There is no instruction for CPU to execute
- But how to make CPU not execute any instruction?

- The behavior of executing an instruction: change the state of CPU
  - change GPR and PC
- If we keep the state of PC unchanged, it is equivalent to not executing any instruction
  - Solution - disable the write enable signals for GPR and PC

Let IFU support SimpleBus

TASK 2 - let IFU support SimpleBus

- Let memory delay 1 cycle to return the result for instruction fetch

```
always @(posedge clock) begin
  ifu_rdata <= mem_read(ifu_raddr);
end
```

- Implement the SimpleBus Protocol in IFU
- Leave LSU unchanged now
  - We will change it in the following tasks
- Try to run some programs
  - The results should be the same as before
    - Since bus protocol should be transparent to the program
  - But you can check the waveform to confirm the timing of the communication
- Compute IPC

Access Data with Delay

Support Write Request

We should add new signals to the protocol to carry a write request

- `waddr` for write address
- `wdata` for write data
- `wen` for write enable
- `wmask` for write mask

```
         +-----+ addr[log2(N)-1:0]  ---> +-----+
         |     | wen                ---> |     |
         | CPU | wdata[31:0]        ---> | MEM |
         |     | wmask[3:0]         ---> |     |
         |     | <---        rdata[31:0] |     |
         +-----+                         +-----+
```

We can combine `raddr` and `waddr` to get `addr`

- Since LSU will not issue read request and write request at the same time

SimpleBus for LSU

For LSU, SimpleBus involves the following signals:

```
        output [31:0] lsu_addr,
        output        lsu_wen,
        output [31:0] lsu_wdata,
        output [ 3:0] lsu_wmask,
        input  [31:0] lsu_rdata,
```

Timing Diagram for Read

`wdata` and `wmask` can be any value

```
                     --\ /----------\ /-------------------------
          lsu_addr      X 0x80400000 X
                     --/ \----------/ \-------------------------
                     ---+            +--------------------------
          lsu_wen       |            |
                        +------------+
                     -------------------------------------------
          lsu_wdata
                     -------------------------------------------
                     -------------------------------------------
          lsu_wmask
                     -------------------------------------------
                     ---------------\ /---------\ /-------------
          lsu_rdata                  X 0x12345678 X
                     ---------------/ \---------/ \-------------
```

Timing Diagram for Write

```
          --\ /----------\ /------------------------
lsu_addr     X 0x80400000 X
          --/ \----------/ \------------------------
             +------------+
lsu_wen      |            |
          ---+            +-------------------------
          --\ /----------\ /------------------------
lsu_wdata    X 0x12345678 X
          --/ \----------/ \------------------------
          --\ /----------\ /------------------------
lsu_wmask    X     0xf    X
          --/ \----------/ \------------------------
          ------------------------------------------
lsu_rdata
          ------------------------------------------
```

Let LSU support SimpleBus

TASK 3 - let LSU support SimpleBus

- Let memory delay 1 cycle to return the result for load

```
always @(posedge clock) begin
  lsu_rdata <= (!lsu_wen) ? mem_read(lsu_addr) : 32'b0;
  if (lsu_wen) begin
    mem_write(lsu_addr, lsu_wdata, lsu_wmask);
  end
end
```

- Implement the SimpleBus protocol in LSU
  - For load, IFU should fetch next instruction after the load data is back
- Leave the device models unchanged now
  - We will change them to real device controllers in the SoC
- Try to run some programs
- Compute IPC

Support Delay > 1 cycle

Larger and Slower Memory

For more types of memory, access delay will be larger

- You may not know the actual delay in advanced
    - It depends on the internal state of the memory

- Memory should tell CPU when the access finishes
    - add a new signal `respValid`
- CPU should also tell Memory when there is a valid request
    - add a new signal `reqValid`

SimpleBus Supporting Delay > 1 cycle

```
+------+ reqValid              ---> +------+
|      | addr[log2(N)-1:0]    ---> |      |
|      | wen                   ---> |      |
| CPU  | wdata[31:0]           ---> | MEM  |
|      | wmask[3:0]            ---> |      |
|      | <---             respValid |      |
|      | <---           rdata[31:0] |      |
+------+                            +------+
```

Take LSU as an example, SimpleBus now involves the following signals:

```
output        lsu_reqValid,
output [31:0] lsu_addr,
output        lsu_wen,
output [31:0] lsu_wdata,
output [ 3:0] lsu_wmask,
input         lsu_respValid,
input  [31:0] lsu_rdata,
```

Timing Diagram for Read

`wdata` and `wmask` can be any value

```
                           +-------------+
          lsu_reqValid     |             |
                        ---+             +---------------------------------------------
----
                        --\ /----------\ /---------------------------------------------
----
          lsu_addr         X 0x80400000 X
                        --/ \----------/ \---------------------------------------------
----
                        --\               /---------------------------------------------
----
          lsu_wen          X             X
                        --/ ------------ \---------------------------------------------
----
                        ---------------------------------------------------------------
----
          lsu_wdata
                        ---------------------------------------------------------------
----
                        ---------------------------------------------------------------
----
          lsu_wmask
                        ---------------------------------------------------------------
----
                                                                      +-------------+
          lsu_respValid                                               |             |
```

Timing Diagram for Write

Although `rdata` is inrelevant for write, SimpleBus still sets `respValid` to high to indicate when a write request finished

```
                           +------------+
          lsu_reqValid     |            |
                        ---+            +---------------------------------------------
----
                        --\ /----------\ /--------------------------------------------
----
          lsu_addr         X 0x80400000 X
                        --/ \----------/ \--------------------------------------------
----
                        --\ ------------ /--------------------------------------------
----
          lsu_wen          X            X
                        --/              \--------------------------------------------
----
                        --\ /----------\ /--------------------------------------------
----
          lsu_wdata        X 0x12345678 X
                        --/ \----------/ \--------------------------------------------
----
                        --\ /----------\ /--------------------------------------------
----
          lsu_wmask        X     0xf    X
                        --/ \----------/ \--------------------------------------------
----
```

Extend the State Machine

To support `reqValid` and `respValid`, we should modify the state machine (take read as example):

- At `idle` state
  - If CPU wants to access memory, then set `reqValid` and `addr`, and switch to `wait` state
  - If CPU does not want to access memory, then stay at `idle` state
- At `wait` state
  - If `respValid` is high, then get the data from `rdata`, and switch to `idle` state
  - If `respValid` is low, then stay at `wait` state

For write, CPU should also wait for `respValid`

Support SimpleBus with Valid Signals

TASK 4 - support SimpleBus with valid signals

- Modify the slave for LSU as following

```
always @(posedge clock) begin
  lsu_rdata <= (lsu_reqValid && !lsu_wen) ? mem_read(lsu_addr) : 32'b0;
  if (lsu_reqValid && lsu_wen) begin
    mem_write(lsu_addr, lsu_wdata, lsu_wmask);
  end
  lsu_respValid <= lsu_reqValid;
end
```

- Modify the slave for IFU in a similar way, but without write
- Implement the new SimpleBus protocol in IFU and LSU
- Try to run some programs
- Compute IPC

More Testing

TASK 5 - do more testing to your code

1. Modify the access delay to 5/10/20 cycles
2. Perform some random testing
   - Implement a LFSR (Linear-feedback shift register)
     - Ask Google or AI
   - Take the random values generated by LFSR to determine the access delay
   - When `respValid` is low, fill `rdata` with random value
3. Add LFSR in IFU and LSU to add random delay to the `reqValue` signals

END