Arbiter Project

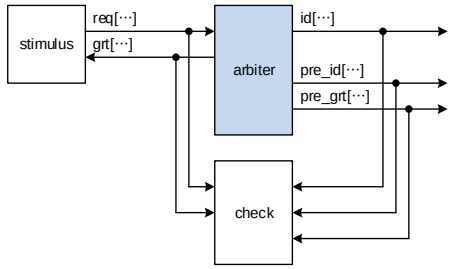
2017

Ando Ki, Ph.D.
(adki@future-ds.com)

Agenda

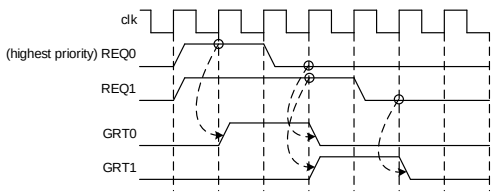
- Pure priority-based arbiter
- Arbiter with fairness

Testing environment



- arbiter
  - block under testing
- stimulus
  - generating stimulus
- check
  - checking validity

Signals and timing



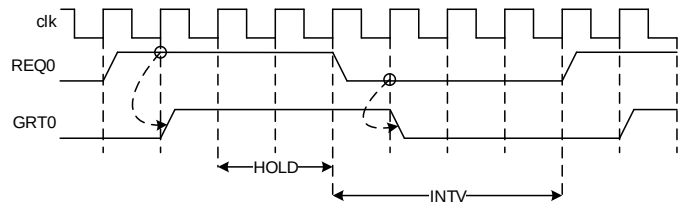
- Say N-bit request
  - i.e., N masters
- req[N-1:0]
  - Where req[0] has the highest priority
- grt[N-1:0]
  - grt[0] corresponds to req[0]
- id[M:0]
  - which master has grant
  - M = log2(N)
  - id[M] is valid flag
    - i.e., id[M]=0 means there is no winer

Prepare your own priority-based arbiter

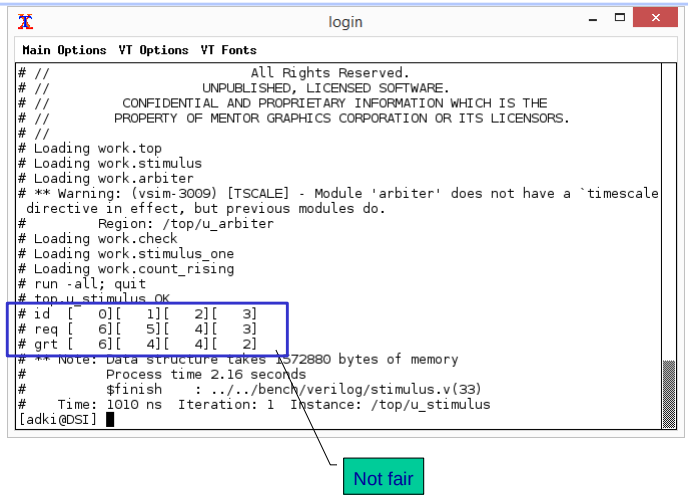
- Go to $(PROJECT)/arbiter_project/arbiter_priority directory
- You have to fill 'rtl/verilog/arbiter.v' file

- Then, go to 'sim/modelsim' directory
  - 'sim_define.v'
    - 'WIDTH' determines the number of maters, i.e., N
    - 'CYCLES' determines the number of cycles to test
    - 'HOLD' specifies the number of cycles to keep 'req' high after getting 'grt'
    - 'INTV' specifies the number of cycles to drive 'req' high again after de-asserting

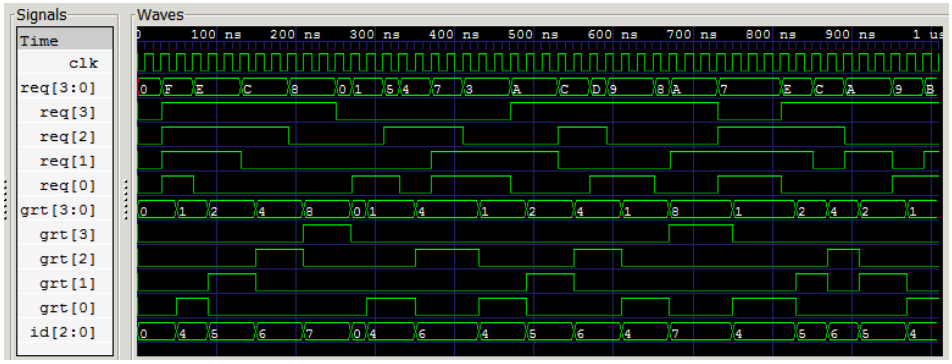
- Run simulator
  - $ make



Simulation results



Simulation results



Check for different 'WIDTH', 'HOLD', and 'INTV'.

Priority encoder

- Pure priority encoder
  - How to keep grant high while its request remains high.
    - Even a higher request joins.

```
module priority
(     input  wire [2:0] req
    , output reg  [2:0] grt
);
 always @ (req)  begin
        if (req[0]) grt <= 3'b001;
  else if (req[1]) grt <= 3'b010;
  else if (req[2]) grt <= 3'b100;
  else             grt <= 3'b000;
 end // always
endmodule
```

Note that 'req[0]' has highest priority.

Priority encoder

- Pure priority encoder

```
module priority
(     input  wire       rset_n
    , input  wire       clk
    , input  wire [2:0] req
    , output reg  [2:0] grt=3'h0
);
    always @ (posedge clk or negedge rst_n) begin
    if (rst_n==1'b0) grt <= 3'h0;
    else casex ({req,grt})
            // priority
            6'b1xx_000: grt <= 3'b100;
            6'b01x_000: grt <= 3'b010;
            6'b001_000: grt <= 3'b001;
            // stay
            6'b1xx_100: grt <= 3'b100;
            6'bx1x_010: grt <= 3'b010;
            6'bxx1_001: grt <= 3'b001;
            // last
            6'b000_xxx: grt <= 3'b000;
            // last and handover
            6'b01x_100: grt <= 3'b010;
            6'b001_100: grt <= 3'b001;
            6'b10x_010: grt <= 3'b100;
            6'b001_010: grt <= 3'b001;
            6'b1x0_001: grt <= 3'b100;
            6'b010_001: grt <= 3'b010;
            default   : grt <= 3'b000;
        endcase
    end // always
endmodule
```

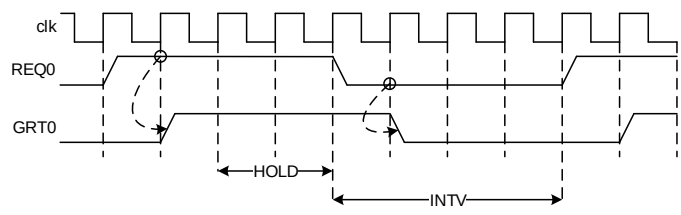
Note that 'req[2]' has highest priority.

Prepare your own improved arbiter

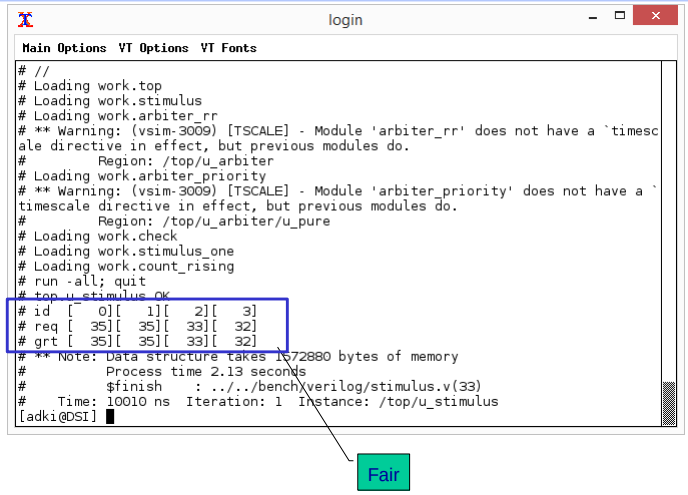
- Go to $(PROJECT)/arbiter_project/arbiter_fair directory
- You have to fill 'rtl/verilog/arbiter_fair.v' file

- Then, go to 'sim/modelsim' directory
  - 'sim_define.v'
    - 'WIDTH' determines the number of maters, i.e., N
    - 'CYCLES' determines the number of cycles to test
    - 'HOLD' specifies the number of cycles to keep 'req' high after getting 'grt'
    - 'INTV' specifies the number of cycles to drive 'req' high again after de-asserting

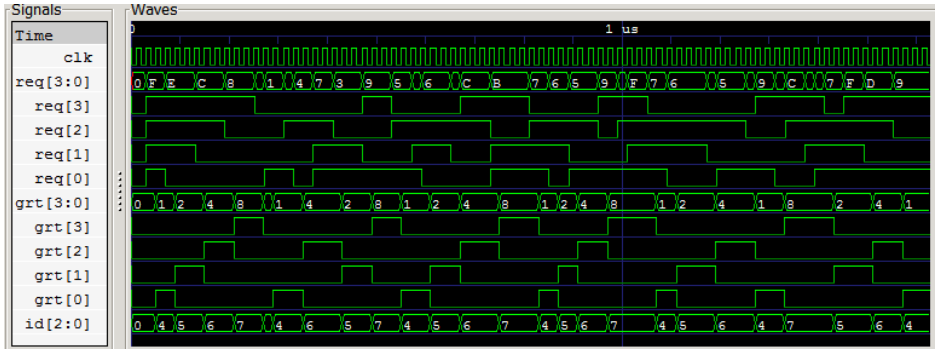
- Run simulator
  - $ make



Simulation results

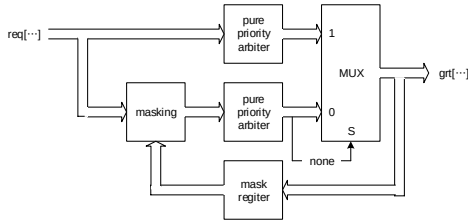


Simulation results



Check for different ‘WIDTH’, ‘HOLD’, and ‘INTV’.

Basic structure



- 'pure priority arbiter': priority-based arbiter (combinational logic)
- 'mask register': logical OR with old and new mask
- 'MUX': select one of two results, where masked results select