
**Politecnico di Milano**
**FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE**

**Sistemi Embedded 1**
**A.A. 2014-2015 – Exam date: 4 Feb 2015**

**Prof. William FORNACIARI**

| | |
|---|---|
| Surname (readable)..... | Name (readable)..... |
| Matr..... | Signature..... |

| Q1 | Q2 | Q3 | | TOT |
|---|---|---|---|---|
| | | | | |

**NOTES**

**It is forbidden to refer to texts or notes of any kind as well as interact with their neighbors. Anyone found in possession of documents relating to the course, although not directly relevant to the subject of the examination will cancel the test. It is not allowed to leave during the first half hour, the task must still be returned, even if it is withdrawn. The presence of the writing (not delivered) implies the renunciation of any previous ratings.**

## Question Q1

***(10 points)*** *Which are the main differences between a real and an ideal sensor?*
*Considering a MEMS inertial sensors tailored to measures accelerations, which are the most relevant aspects differentiating such solution to other available technologies?*

## Question Q2

***a) (5 points)*** *Consider an on-chip bus interconnect with support for pipeline transactions used for communicate between the CPUs and the memory.*
*Discuss the main limitations of such architecture in terms of bus utilization and overall system performance. Then describe the split transaction and burst transaction optimizations and how they can face such limitations. (Synthetic examples may help the answer, while cannot substitute the discussion).*

*SHORT SOLUTION*

*The standard pipelined bus architecture grants the bus to a transaction keeping it reserved until the end of the transaction. Since, all the transactions, i.e. reads and writes, requires access to memory, the variable memory latency degrades the bus utilization, since the bus is virtually kept busy for the granted transaction even if not actually used in all cycles due to memory access time. The split transaction optimization restores a deterministic shape to the transactions, both reads and writes, since each transaction is split between the request and the response stages. For each stage a bus arbitration is required thus it doubles the arbitration stages for each transaction. However, it avoids to keep the bus reserved to a transaction during memory access time that is unpredictable and can take time.*

*Write with split transaction optim  AR | A | G | Bad | AR | A | G | Back*
*Read with split transaction optim  AR | A | G | Ba | AR | A | G | Bd*

*Considering the standard pipelined bus architecture, the need to send multiple data from the same source-destination pair, i.e. write from CPU to memory of a cache line, requires multiple transactions one per data, where the data size is equal to the bus width.*

*Burst transaction optimization allows a transaction to arbitrate a single time on the bus to transmit multiple data, thus avoiding the need for multiple bus arbitrations. It can be combined with the split transaction optimization and the burst size is a parameter that can be set in a range reported in the bus specification document.*

***b) (5 points)*** *Considering the strategies tailored to support serial communication, the student is required to concisely explain the goal and main differences between the following protocols* SPI (*Serial Peripheral Interface*) and I2C (*Inter IC*).

## Question Q3

***a) (3 points)*** *Describe the main difference between blocking and non-blocking assignements in the always block in Verilog Hardware Description Language (HDL) . Considering the code fragment in Listing* ***Q2.a.1*** *identifies the blocking and non-blocking statements and the behavior of the circuit, possibly with a schematic.*

***Listing Q2.a.1***

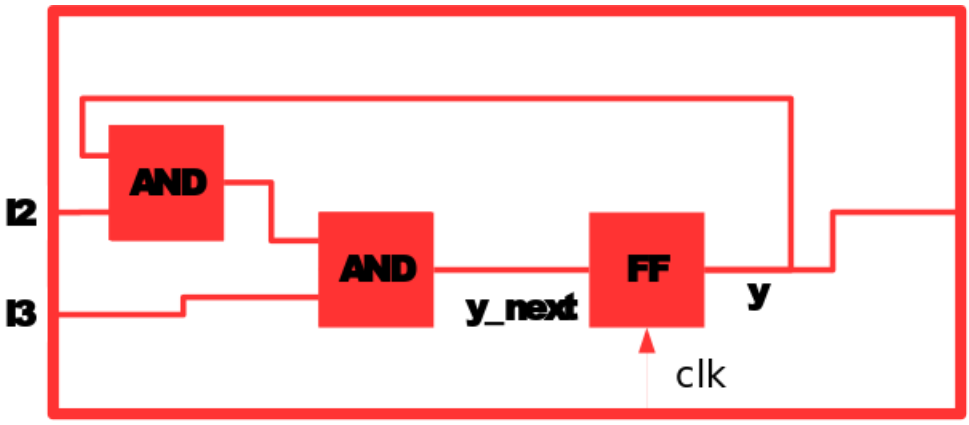
```
module m1( clk, i2 ,i3 , y );
        input wire clk, i2 ,i3;
        output reg y;

        reg y_next;
        always@(posedge clk)
        begin
                y<=y_next;
        end

        always@(*)
        begin
                y_next=y;
                y_next= y_next & i2;
                y_next= y_next & i3;
        end

endmodule
```

*SHORT SOLUTION*

*Considering a Verilog always block used to describe combinatorial circuits, all the non-blocking assignements in the block are evaluated in parallel using for the signals on the right side of the <= the values at the beginning of the block, and the left side signals are assigned once at the end of the block execution.*

*On the other hand, an always block composed of blocking assignements evaluates them one after the other. Thus, each assignement uses the most updated value for the right side signals eventually updated by the previous blocking assignements.*

*Considering blocking assignement Listing Q2.a.1, the equivalent circuit is:*

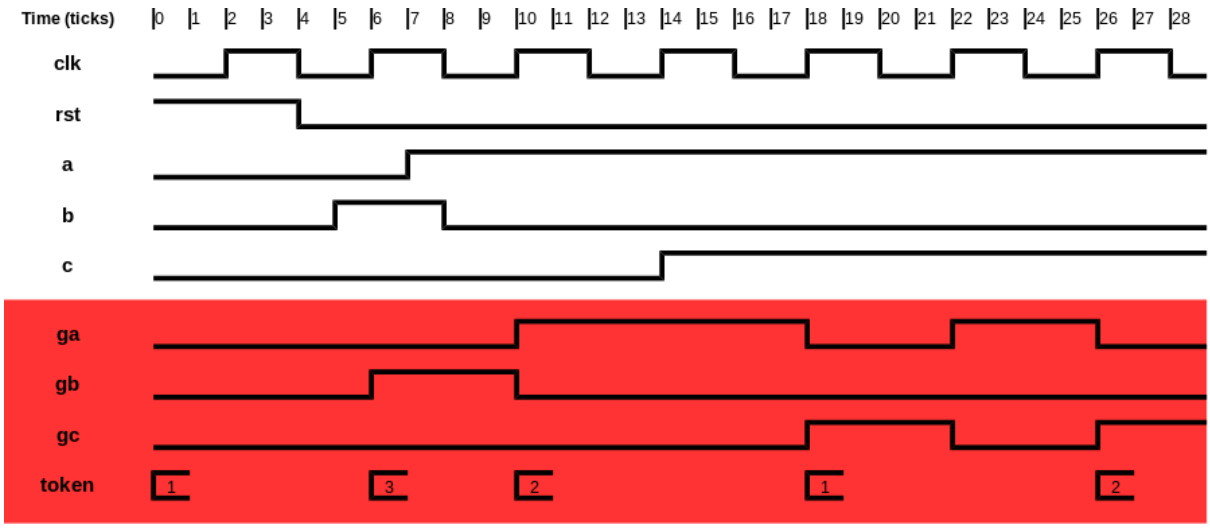


***b) (4 points)*** *Listing Q3.b.1 provides a behavioral description of a generic round-robin arbiter with three input requests, i.e. a,b,c and three grant output signals, one per each request. Complete the timing diagram provided in Figure Q3.b.2 according to the module implementation.*

***Listing Q3.b.1***

```
module m3(clk,rst,a,b,c,ga,gb,gc);
input clk;
input rst;
input a;
input b;
input c;
output ga,gb,gc;
reg [1:0] token, token_next;
reg [2:0] g, g_next;

        always@(posedge clk,posedge rst)
        begin
                if(rst==1)
                begin
                        g <= 0;  token <= 1;
                end
                else
                begin
                        g <= g_next; token <= token_next;
                end
        end
        always@(*)
        begin
                token_next=token;    g_next=0;
                case(token)
                        2'b01:
                                begin
                                        if(a==1) begin  g_next=3'b001; token_next=2;  end
                                        else if(b==1) begin g_next=3'b010; token_next=3; end
                                        else if(c==1) begin g_next=3'b100; token_next=1; end
                                end
                        2'b10:
                                begin
                                        if(b==1) begin g_next=3'b010; token_next=3; end
                                        else if(c==1) begin g_next=3'b100; token_next=1; end
                                        else if(a==1) begin  g_next=3'b001; token_next=2;  end
                                end
                        2'b11:
                                begin
                                        if(c==1) begin g_next=100; token_next=1; end
                                        else if(a==1) begin  g_next=3'b001; token_next=2;  end
                                        else if(b==1) begin g_next=3'b010; token_next=3; end
                                end
                endcase
        end
        assign ga=g[0];  assign gb=g[1]; assign gc=g[2];
endmodule
```

***Figure Q3.b.2***



***C (3 points)*** *Change the module implementation in Listing Q3.b.1 to obtain a priority arbiter where priority of the signals is as follow: a>b>c. The very same interface of the module has to be preserved as well as the end to end behavior in term of output signals, i.e. registered output. (Note that many implementations are possible).*

***Listing Q3.b.1_solution***

```
module m3(clk,rst,a,b,c,ga,gb,gc);
input clk;
input rst;
input a;
input b;
input c;
output ga,gb,gc;
 //reg [1:0] token, token_next;
reg [2:0] g, g_next;

        always@(posedge clk,posedge rst)
        begin
                if(rst==1)
                        g <= 0;
                else
                        g <= g_next;
        end
        always@(*)
        begin
                g_next=0;
                if(a==1)
                        g_next=3'b001;
                else if (b==1)
                        g_next=3'010;
                else if (c==1)
                        g_next=3'100;
        end
        assign ga=g[0];  assign gb=g[1]; assign gc=g[2];
endmodule
```