THE UNIVERSITY OF KANSAS

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

EECS 645 – Computer Architecture

Fall 2015

Homework 02 (Resource Sharing)

Student Name: Dustin Horvath Student ID: 2729265

Lecture Example (Template Answer):

Given a resource that is to be shared by two consumers such that only one consumer has access to the resource at any given time. The policy of access is non-preemptive with no priority. More specifically the policy is as follows:

- When the resource is idle and the consumers simultaneously request the resource, their requests are ignored until only one request is submitted
- When the resource is being accessed by any of the consumers, the consumers are scheduled as First-Come-First-Served (FCFS)

Design a two-consumer arbiter/controller that controls the access to the shared resource and implements the above policy. In the design process, provide the following:

1) The system architecture (block diagram) showing the interface ports to the arbiter including the clock and reset signals.
2) Finite State Machine (FSM) diagram showing all possible states, transitions, and output values.
3) K-maps for internal state and output variables.
4) Boolean expressions for internal state and output variables.
5) Detailed logic diagram using synchronous memory elements showing internal connections and external interfaces.
6) Complete description of the arbiter using VHDL code.
7) Complete VHDL testbench along with simulation results.

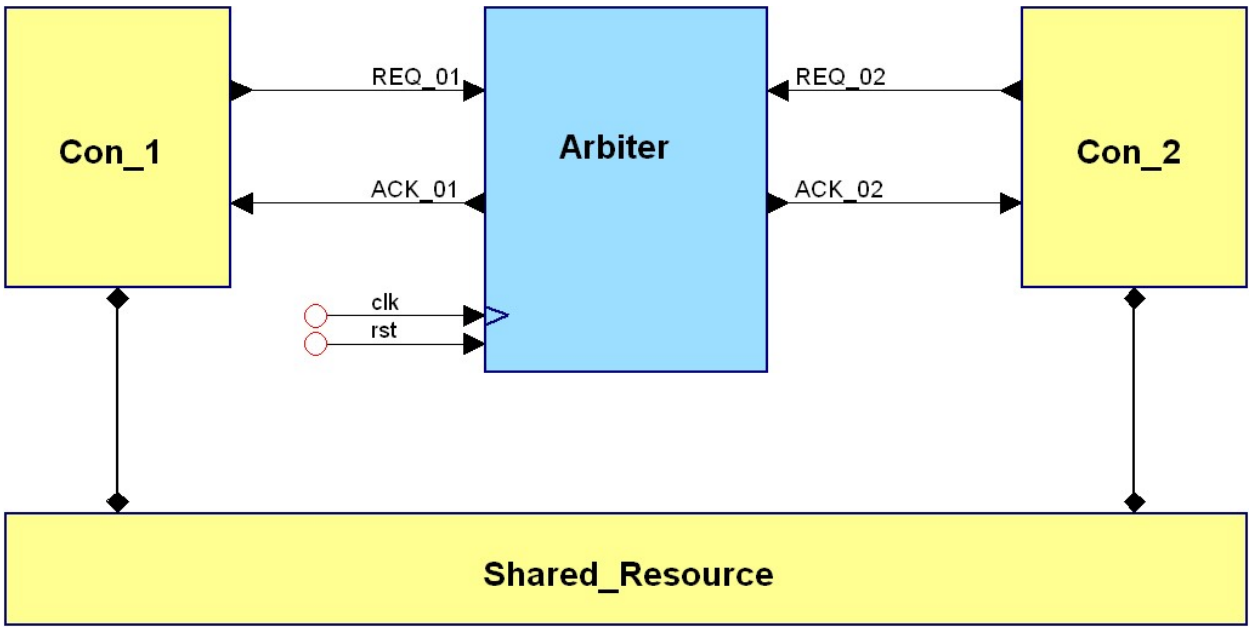


Figure 1: System architecture

Table 1: Input Code Assignment

Input Combinations (Input Codes)		Input Description
REQ_01	REQ_02	
0	0	No requests
0	1	Consumer 2 requests resource
1	0	Consumer 1 requests resource
1	1	Both consumers request resource

Table 2: Output Code Assignment

Output Combinations (Output Codes)		Output Description
ACK_01	ACK_02	
0	0	None granted access to resource
0	1	Consumer 2 granted access to resource
1	0	Consumer 1 granted access to resource
1	1	Forbidden output

Table 3: State Code Assignment

State Description		State Codes	
		S_01	S_02
Required States	Resource is idle (No access)	0	0
	Resource is used by consumer 2 (Con_02)	0	1
	Resource is used by consumer 1 (Con_01)	1	0
Unused States	Extra unused state (Unused)	1	1

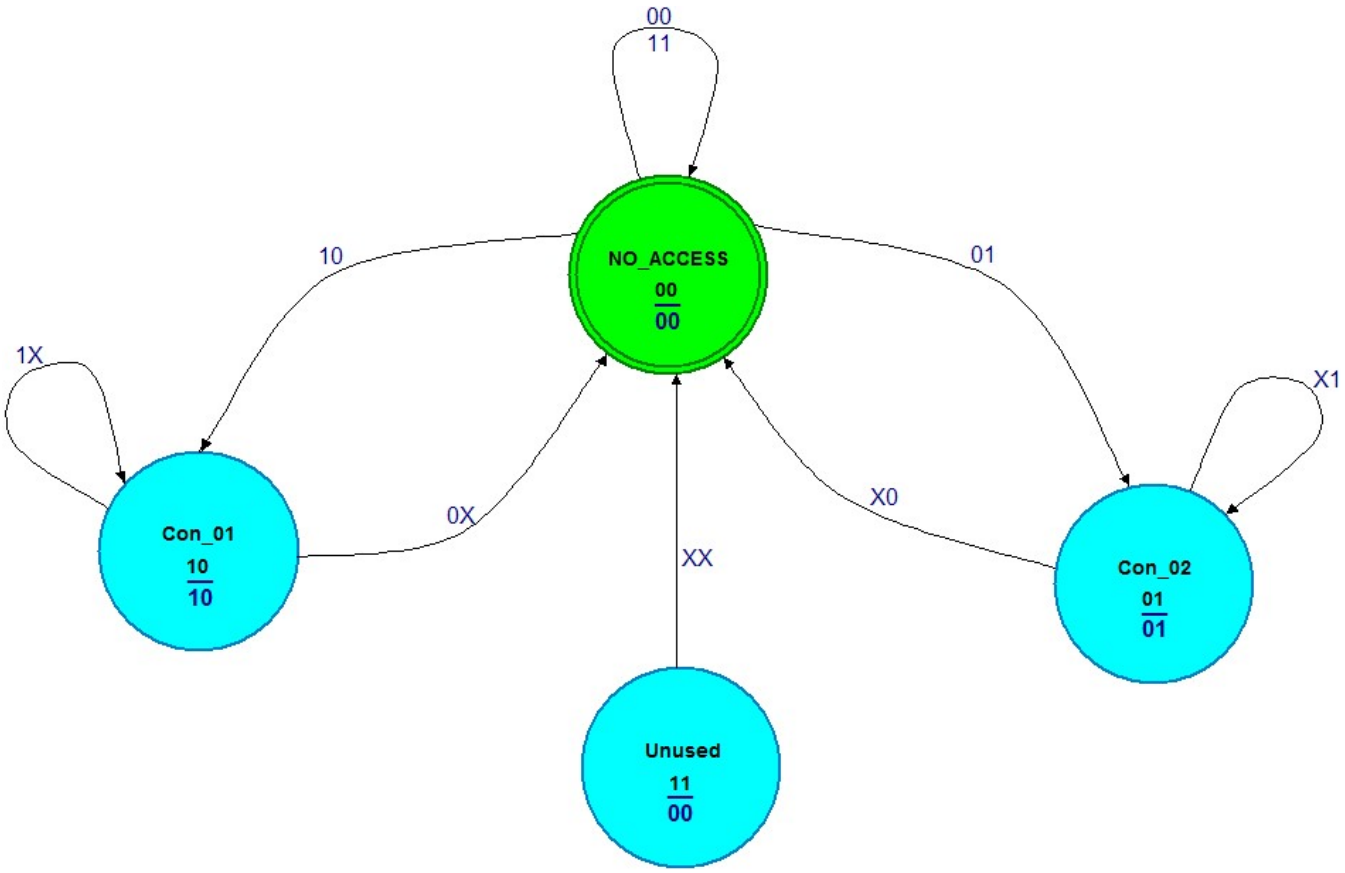


Figure 2: State transition diagram

Table 4: State transition table

Next State (S_1 , S_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00	00	01	00	10
	01	00	01	01	00
	11	00	00	00	00
	10	00	00	10	10

Table 5: K-map for the state variable S_1

Next State (S_1)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				1
	01				
	11				
	10			1	1

$$S_1^{next} = REQ_1 \cdot \overline{REQ_2} \cdot \overline{S_2^{current}} + REQ_1 \cdot S_1^{current} \cdot \overline{S_2^{current}}$$

Table 6: K-map for the state variable S_2

Next State (S_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00		1		
	01		1	1	
	11				
	10				

$$S_2^{next} = REQ_2 \cdot \overline{S_1^{current}} \cdot S_2^{current} + \overline{REQ_1} \cdot REQ_2 \cdot \overline{S_1^{current}}$$

Table 7: Output transition table

Outputs (ACK_1 , ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00	00	00	00	00
	01	01	01	01	01
	11	00	00	00	00
	10	10	10	10	10

Table 8: K-map for the output variable ACK_1

Outputs (ACK_1)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01				
	11				
	10	1	1	1	1

$$ACK_1 = S_1^{current} \cdot \overline{S_2^{current}}$$

Table 9: K-map for the output variable ACK_2

Outputs (ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01	1	1	1	1
	11				
	10				

$$ACK_2 = \overline{S_1^{current}} \cdot S_2^{current}$$

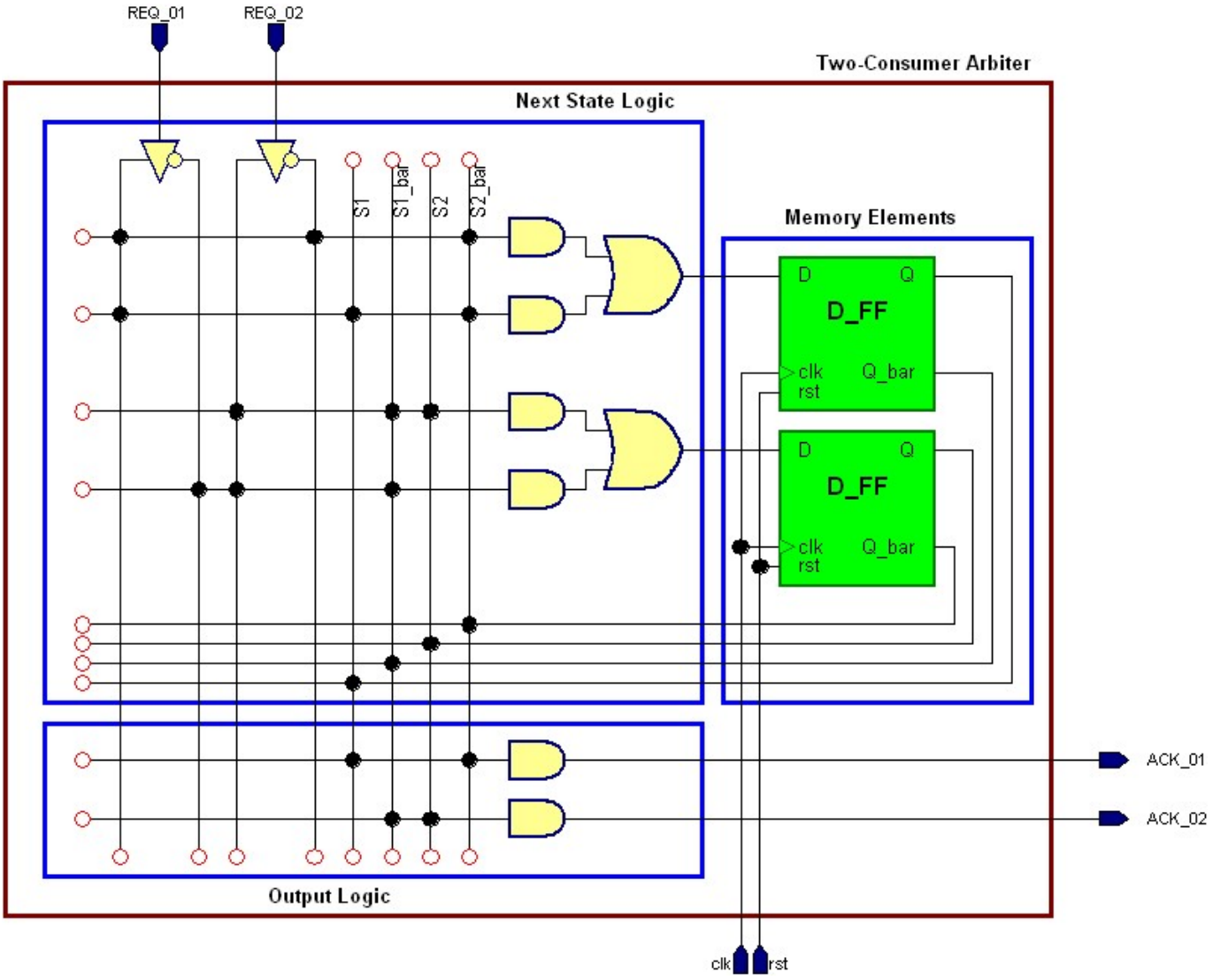


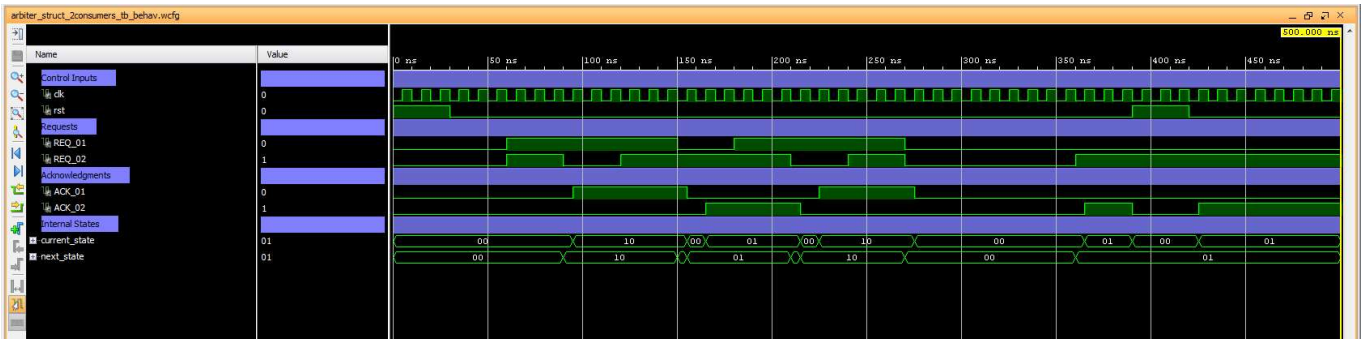
Figure 3: Synchronous logic diagram

Structural VHDL code

```
 1  LIBRARY ieee;
 2  USE ieee.std_logic_1164.all;
 3  USE ieee.std_logic_unsigned.all;
 4
 5  ENTITY arbiter_struct_2cons IS
 6     PORT(
 7        REQ_01 : IN     std_logic;
 8        REQ_02 : IN     std_logic;
 9        clk    : IN     std_logic;
10        rst    : IN     std_logic;
11        ACK_01 : OUT    std_logic;
12        ACK_02 : OUT    std_logic
13     );
14  END arbiter_struct_2cons ;
15
16
17
18  ARCHITECTURE struct_no_priority OF arbiter_struct_2cons IS
19
20     -- Declare current and next state signals
21     SIGNAL s1_current, s2_current : std_logic;
22     SIGNAL s1_next   , s2_next    : std_logic;
23
24  BEGIN
25
26     -------------------------------------------------------------
27     memory_elements : PROCESS(clk, rst)
28     -------------------------------------------------------------
29     BEGIN
30        IF (rst = '1') THEN
31           s1_current <= '0';
32           s2_current <= '0';
33           -- Reset Values
34        ELSIF (clk'EVENT AND clk = '1') THEN
35           s1_current <= s1_next;
36           s2_current <= s2_next;
37        END IF;
38     END PROCESS memory_elements;
39
40     -------------------------------------------------------------
41     -- state_logic
42     -------------------------------------------------------------
43     s1_next <= (REQ_01 and (not REQ_02) and (not s2_current)) or
44                (REQ_01 and s1_current and (not s2_current));
45
46     s2_next <= (REQ_02 and (not REQ_01) and (not s1_current)) or
47                (REQ_02 and s2_current and (not s1_current));
48
49     -------------------------------------------------------------
50     -- output_logic
51     -------------------------------------------------------------
52     ACK_01 <= s1_current and (not s2_current);
53     ACK_02 <= s2_current and (not s1_current);
54
55  END struct_no_priority;
```

Testbench and Simulation Results

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY arbiter_struct_2consumers_tb IS
END arbiter_struct_2consumers_tb;

ARCHITECTURE behavior OF arbiter_struct_2consumers_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT arbiter_struct_2cons
        PORT(
            REQ_01 : IN     std_logic;
            REQ_02 : IN     std_logic;
            clk    : IN     std_logic;
            rst    : IN     std_logic;
            ACK_01 : OUT    std_logic;
            ACK_02 : OUT    std_logic
        );
    END COMPONENT ;

    --Inputs
    signal REQ_01 : std_logic := '0';
    signal REQ_02 : std_logic := '0';
    signal clk    : std_logic := '0';
    signal rst    : std_logic := '0';

    --Outputs
    signal ACK_01 : std_logic;
    signal ACK_02 : std_logic;

    -- Clock period definitions
    constant clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: arbiter_struct_2cons PORT MAP (
                                        REQ_01 => REQ_01,
                                        REQ_02 => REQ_02,
                                        clk => clk,
                                        rst => rst,
                                        ACK_01 => ACK_01,
                                        ACK_02 => ACK_02
                                );

    -- Clock process definitions
    clk_process :process
    begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 3 clock periods.
                    rst <= '1';
        wait for clk_period*3;

                    rst <= '0';
        wait for clk_period*3;

        -- insert stimulus here
                    REQ_01 <= '1';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    REQ_01 <= '1';
                    REQ_02 <= '0';
        wait for clk_period*3;

                    REQ_01 <= '1';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    REQ_01 <= '0';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    REQ_01 <= '1';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    REQ_01 <= '1';
                    REQ_02 <= '0';
        wait for clk_period*3;

                    REQ_01 <= '1';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    REQ_01 <= '0';
                    REQ_02 <= '0';
        wait for clk_period*9;

                    REQ_01 <= '0';
                    REQ_02 <= '1';
        wait for clk_period*3;

                    rst <= '1';
        wait for clk_period*3;

                    rst <= '0';
        wait for clk_period*3;

        wait;
    end process;

END;
```



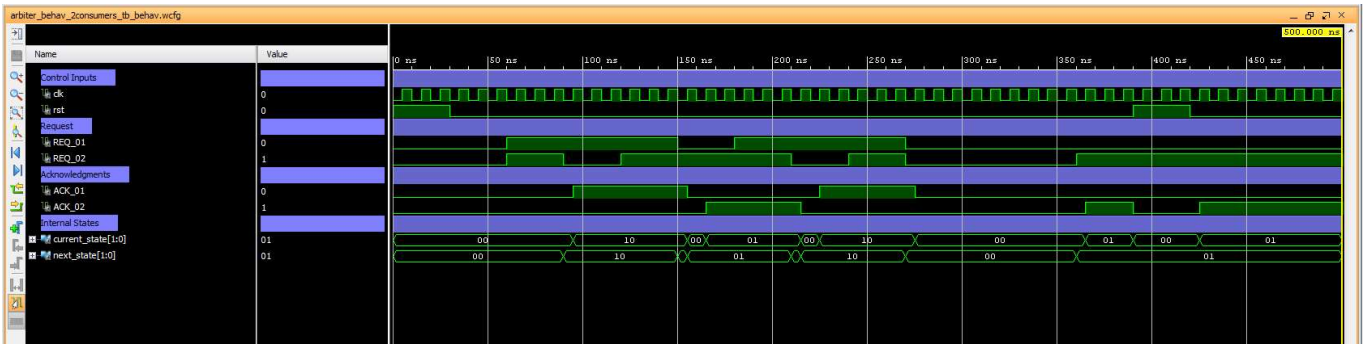
Behavioral VHDL code

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.std_logic_unsigned.all;
4
5  ENTITY arbiter_bahav_2cons IS
6     PORT(
7        REQ_01 : IN     std_logic;
8        REQ_02 : IN     std_logic;
9        clk    : IN     std_logic;
10       rst    : IN     std_logic;
11       ACK_01 : OUT    std_logic;
12       ACK_02 : OUT    std_logic
13    );
14 END arbiter_bahav_2cons ;
15
16
17
18 ARCHITECTURE behav_no_priority OF arbiter_bahav_2cons IS
19
20    -- Architecture Declarations
21    SUBTYPE STATE_TYPE IS
22       std_logic_vector(1 DOWNTO 0);
23
24    -- Hard encoding
25    CONSTANT NO_ACCESS : STATE_TYPE := "00" ;
26    CONSTANT Con_01 : STATE_TYPE := "10" ;
27    CONSTANT Con_02 : STATE_TYPE := "01" ;
28    CONSTANT Unused : STATE_TYPE := "11" ;
29
30    -- Declare current and next state signals
31    SIGNAL current_state : STATE_TYPE ;
32    SIGNAL next_state : STATE_TYPE ;
33
34    SIGNAL REQ_VEC : std_logic_vector(1 TO 2);
35
36 BEGIN
37
38    REQ_VEC <= (REQ_01 & REQ_02);
39
40    -----------------------------------------------------
41    memory_elements : PROCESS(clk, rst)
42    -----------------------------------------------------
43    BEGIN
44       IF (rst = '1') THEN
45          current_state <= NO_ACCESS;
46          -- Reset Values
47       ELSIF (clk'EVENT AND clk = '1') THEN
48          current_state <= next_state;
49       END IF;
50    END PROCESS memory_elements;
51
52    -----------------------------------------------------
53    state_logic : PROCESS (REQ_VEC, current_state)
54    -----------------------------------------------------
55    BEGIN
56       CASE current_state IS
57       WHEN NO_ACCESS =>
58          next_state <= NO_ACCESS;
59          IF (REQ_VEC = "10") THEN
60             next_state <= Con_01;
61          END IF;
62          IF (REQ_VEC = "01") THEN
63             next_state <= Con_02;
64          END IF;
65       WHEN Con_01 =>
66          next_state <= Con_01;
67          IF (REQ_VEC(1) = '0') THEN
68             next_state <= NO_ACCESS;
69          END IF;
70       WHEN Con_02 =>
71          next_state <= Con_02;
72          IF (REQ_VEC(2) = '0') THEN
73             next_state <= NO_ACCESS;
74          END IF;
75       WHEN OTHERS =>
76          next_state <= NO_ACCESS;
77       END CASE;
78    END PROCESS state_logic;
79
80    -----------------------------------------------------
81    output_logic : PROCESS (current_state)
82    -----------------------------------------------------
83    BEGIN
84       CASE current_state IS
85       WHEN Con_01 =>
86          ACK_01 <= '1';
87          ACK_02 <= '0';
88       WHEN Con_02 =>
89          ACK_01 <= '0';
90          ACK_02 <= '1';
91       WHEN OTHERS =>
92          ACK_01 <= '0';
93          ACK_02 <= '0';
94       END CASE;
95    END PROCESS output_logic;
96 END behav_no_priority;
```

Testbench and Simulation Results

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY arbiter_behav_2consumers_tb IS
END arbiter_behav_2consumers_tb;

ARCHITECTURE behavior OF arbiter_behav_2consumers_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT arbiter_bahav_2cons
       PORT(
          REQ_01 : IN     std_logic;
          REQ_02 : IN     std_logic;
          clk    : IN     std_logic;
          rst    : IN     std_logic;
          ACK_01 : OUT    std_logic;
          ACK_02 : OUT    std_logic
       );
    END COMPONENT ;

    --Inputs
    signal REQ_01 : std_logic := '0';
    signal REQ_02 : std_logic := '0';
    signal clk    : std_logic := '0';
    signal rst    : std_logic := '0';

    --Outputs
    signal ACK_01 : std_logic;
    signal ACK_02 : std_logic;

    -- Clock period definitions
    constant clk_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: arbiter_bahav_2cons PORT MAP (
                                       REQ_01 => REQ_01,
                                       REQ_02 => REQ_02,
                                       clk => clk,
                                       rst => rst,
                                       ACK_01 => ACK_01,
                                       ACK_02 => ACK_02
                                     );

    -- Clock process definitions
    clk_process :process
    begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
    end process;

```

```

    -- Stimulus process
    stim_proc: process
    begin
       -- hold reset state for 3 clock periods.
                rst <= '1';
       wait for clk_period*3;

                rst <= '0';
       wait for clk_period*3;

       -- insert stimulus here
                REQ_01 <= '1';
                REQ_02 <= '1';
       wait for clk_period*3;

                REQ_01 <= '1';
                REQ_02 <= '0';
       wait for clk_period*3;

                REQ_01 <= '1';
                REQ_02 <= '1';
       wait for clk_period*3;

                REQ_01 <= '0';
                REQ_02 <= '1';
       wait for clk_period*3;

                REQ_01 <= '1';
                REQ_02 <= '1';
       wait for clk_period*3;

                REQ_01 <= '1';
                REQ_02 <= '0';
       wait for clk_period*3;

                REQ_01 <= '1';
                REQ_02 <= '1';
       wait for clk_period*3;

                REQ_01 <= '0';
                REQ_02 <= '0';
       wait for clk_period*9;

                REQ_01 <= '0';
                REQ_02 <= '1';
       wait for clk_period*3;

                rst <= '1';
       wait for clk_period*3;

                rst <= '0';
       wait for clk_period*3;

       wait;
    end process;

END;

```



Homework Problem I:

Given a resource that is to be shared by two consumers such that only one consumer has access to the resource at any given time. The policy of access is non-preemptive with priority determined by the consumer index. More specifically the policy is as follows:

- When the resource is idle and the consumers simultaneously request the resource, the consumers are prioritized according to their consumer number (i.e. consumer 2 has a higher priority than consumer 1)
- When the resource is being accessed by any of the consumers, the consumers are scheduled as First-Come-First-Served (FCFS)

Design a two-consumer arbiter/controller that controls the access to the shared resource and implements the above policy. In the design process, provide the following:

1) The system architecture (block diagram) showing the interface ports to the arbiter including the clock and reset signals.
2) Finite State Machine (FSM) diagram showing all possible states, transitions, and output values.
3) K-maps for internal state and output variables.
4) Boolean expressions for internal state and output variables.
5) Detailed logic diagram using synchronous memory elements showing internal connections and external interfaces.
6) Complete description of the arbiter using VHDL code.
7) Complete VHDL testbench along with simulation results.

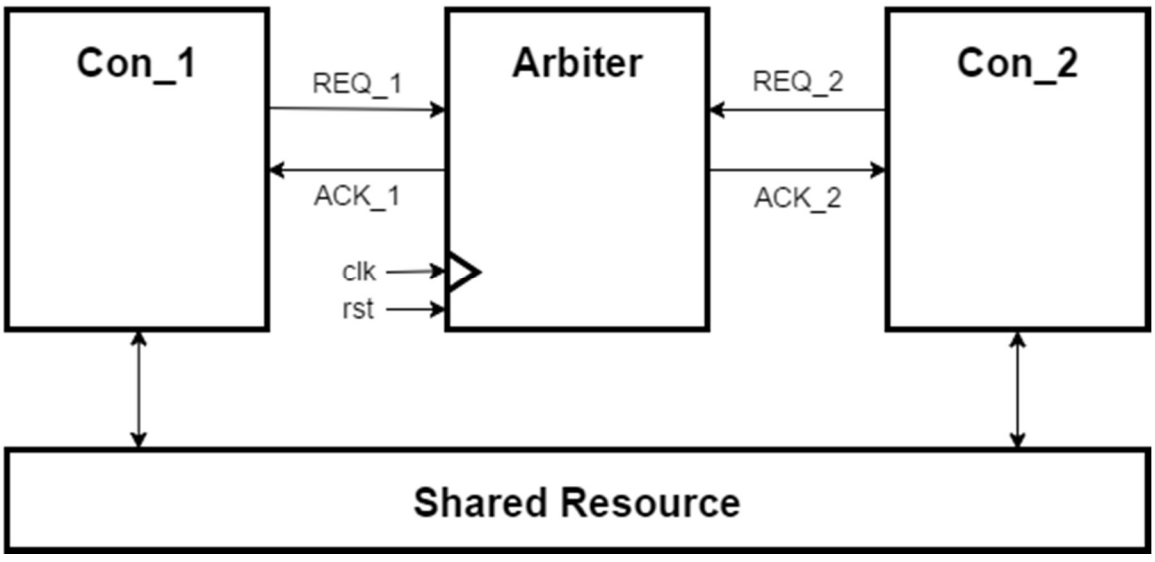


Figure 1: System architecture

Table 1: Input Code Assignment

Input Combinations (Input Codes)		Input Description
REQ_01	REQ_02	
0	0	No Requests
0	1	Consumer 2 requests resource
1	0	Consumer 1 requests resource
1	1	Both consumers request resource

Table 2: Output Code Assignment

Output Combinations (Output Codes)		Output Description
ACK_01	ACK_02	
0	0	None granted access to resource
0	1	Consumer 2 granted access to resource
1	0	Consumer 1 granted access to resource
1	1	Forbidden output

Table 3: State Code Assignment

State Description		State Codes	
		S_01	S_02
Required States	Resource is idle (No access)	0	0
	Resource is used by consumer 2 (Con_02)	0	1
	Resource is used by consumer 1 (Con_01)	1	0
Unused States	Extra unused state (Unused)	1	1

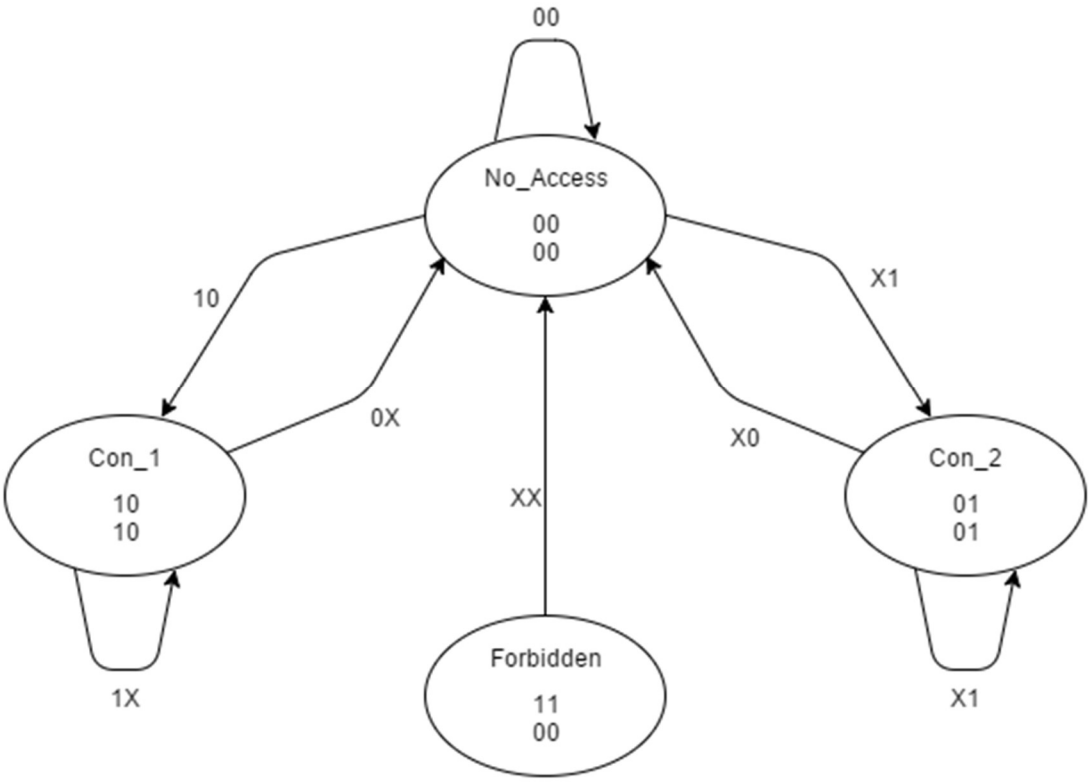


Figure 2: State transition diagram

Table 4: State transition table

Next State (S_1 , S_2)		Inputs (REQ_1 , REQ_2) 00	01	11	10
Current State (S_1 , S_2)	00	00	01	01	10
	01	00	01	01	00
	11	00	00	00	00
	10	00	00	10	10

Table 5: K-map for the state variable S_1

Next State (S_1)		Inputs (REQ_1 , REQ_2) 00	01	11	10
Current State (S_1 , S_2)	00				1
	01				
	11				
	10			1	1

$$S_1^{next} = REQ\_1 * \overline{REQ\_2} * \overline{S1\_current} + REQ\_1 * S1\_current * \overline{S2\_current}$$

Table 6: K-map for the state variable S_2

Next State (S_2)		Inputs (REQ_1 , REQ_2) 00	01	11	10
Current State (S_1 , S_2)	00		1	1	
	01		1	1	
	11				
	10				

$$S_2^{next} = \overline{S1\_current} * REQ\_2$$

Table 7: Output transition table

Outputs (ACK_1 , ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00	00	00	00	00
	01	01	01	01	01
	11	00	00	00	00
	10	10	10	10	10

Table 8: K-map for the output variable ACK_1

Outputs (ACK_1)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01				
	11				
	10	1	1	1	1

$$ACK_1 = S1\_current * \overline{S2\_current}$$

Table 9: K-map for the output variable ACK_2

Outputs (ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01	1	1	1	1
	11				
	10				

$$ACK_2 = \overline{S1\_current} * S2\_current$$

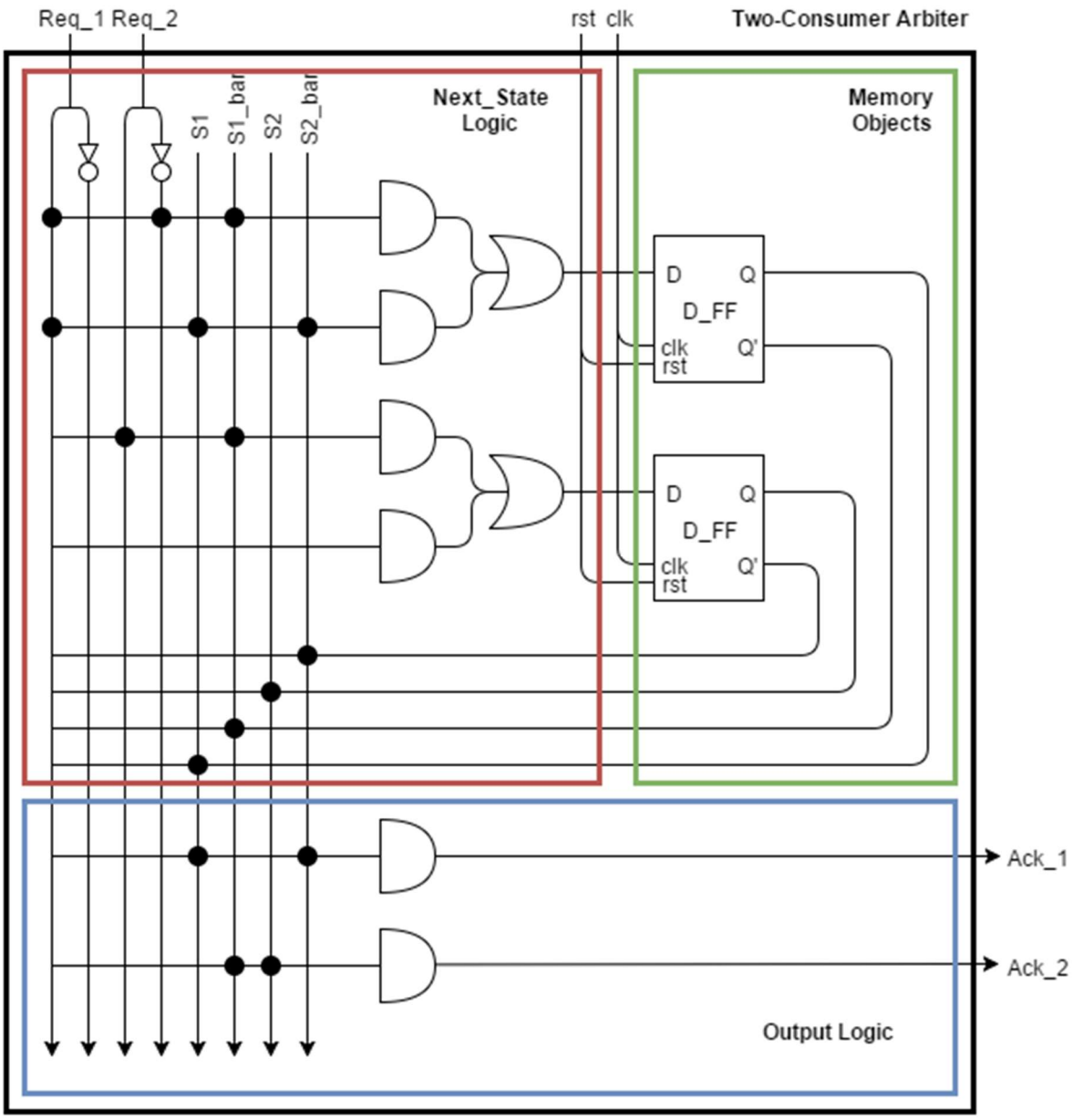


Figure 3: Synchronous logic diagram

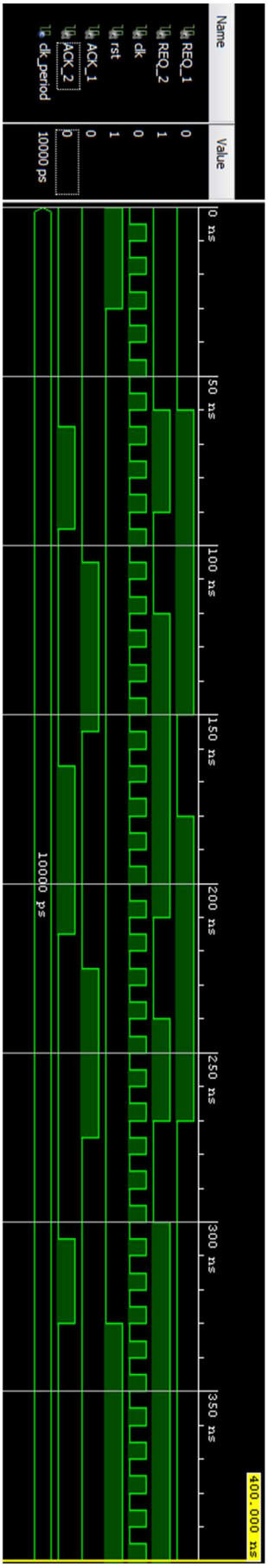
Structural VHDL code

```
1  ----------------------------------------------------------------------------------
2  -- Company: University of Kansas
3  -- Engineer: Dustin Horvath
4  --
5  -- Create Date: 09/28/2015 09:08:39 PM
6  -- Design Name: arbiter_structural_2cons_highPri
7  -- Module Name: arbiter_structural_2cons_highPri - Structural
8  -- Project Name: EECS645 Homework 2
9  --
10 -- Revision: See latest git commit at http://git.nichnologist.net/dhorvath/eecs/tree/master/eecs645/hw2
11 -- Additional Comments: None
12 --
13 ----------------------------------------------------------------------------------
14
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.STD_LOGIC_UNSIGNED.ALL;
18
19 -- Define toplevel I/O
20 entity arbiter_structural_2cons_highPri is
21     PORT(
22         REQ_1 : IN      STD_LOGIC;
23         REQ_2 : IN      STD_LOGIC;
24         clk   : IN      STD_LOGIC;
25         rst   : IN      STD_LOGIC;
26         ACK_1 : OUT     STD_LOGIC;
27         ACK_2 : OUT     STD_LOGIC
28     );
29 end arbiter_structural_2cons_highPri;
30
31
32 architecture struct_high_index_priority of arbiter_structural_2cons_highPri is
33     -- Define interim signals
34     SIGNAL s1_current , s2_current  :  std_logic;
35     SIGNAL s1_next    , s2_next     :  std_logic;
36
37 begin
38
39
40     -- Define memory elements
41     memory_elements : PROCESS(clk, rst)
42
43     -- Define reset logic and state cycle behavior
44     BEGIN
45         IF (rst = '1') THEN
46             s1_current <= '0';
47             s2_current <= '0';
48         ELSIF (clk'EVENT AND clk = '1') THEN
49             s1_current <= s1_next;
50             s2_current <= s2_next;
51         END IF;
52     END PROCESS memory_elements;
53
54 -- Define next state logic
55     s1_next <= ((REQ_1 and (not REQ_2) and (not s2_current)) or (REQ_1 and s1_current and (not s2_current)));
56     s2_next <= ((not s1_current) and REQ_2);
57
58
59 -- Define output logic
60     ACK_1 <= s1_current and (not s2_current);
61     ACK_2 <= s2_current and (not s1_current);
62
63 end struct_high_index_priority;
64
```

Testbench and Simulation Results

```
1  ----------------------------------------------------------------------------------
2  -- Company: University of Kansas
3  -- Engineer: Dustin Horvath
4  --
5  -- Create Date: 09/28/2015 09:08:39 PM
6  -- Design Name: arbiter_structural_2cons_highPri_tb
7  -- Module Name: arbiter_structural_2cons_highPri_tb - Structural
8  -- Project Name: EECS645 Homework 2
9  --
10 -- Revision: See latest git commit at http://git.nichnologist.net/dhorvath/eecs/tree/master/eecs645/hw2
11 -- Additional Comments: None
12 --
13 ----------------------------------------------------------------------------------
14
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17
18 -- Uncomment the following library declaration if using
19 -- arithmetic functions with Signed or Unsigned values
20 --use IEEE.NUMERIC_STD.ALL;
21
22 -- Uncomment the following library declaration if instantiating
23 -- any Xilinx leaf cells in this code.
24 --library UNISIM;
25 --use UNISIM.VComponents.all;
26
27 LIBRARY ieee;
28 USE ieee.std_logic_1164.ALL;
29
30 -- Define toplevel I/O (empty)
31 ENTITY arbiter_structural_2cons_highPri_tb IS
32 END arbiter_structural_2cons_highPri_tb;
33
34
35 ARCHITECTURE behavior OF arbiter_structural_2cons_highPri_tb IS
36
37     -- Define ports of simulation design
38     COMPONENT arbiter_structural_2cons_highPri
39         PORT(
40             REQ_1  : IN     std_logic;
41             REQ_2  : IN     std_logic;
42             clk    : IN     std_logic;
43             rst    : IN     std_logic;
44             ACK_1  : OUT    std_logic;
45             ACK_2  : OUT    std_logic
46         );
47     END COMPONENT;
48
49     -- Define interim signals
50     signal REQ_1  :  std_logic  := '0';
51     signal REQ_2  :  std_logic  := '0';
52     signal clk    :  std_logic  := '0';
53     signal rst    :  std_logic  := '0';
54
55
56     signal ACK_1  :  std_logic;
57     signal ACK_2  :  std_logic;
58
59     constant clk_period  :  time  := 10 ns;
60
```

```
61 BEGIN
62
63     -- Unit Under Test
64     -- Map elements of simulation to elements of UUT.
65     uut: arbiter_structural_2cons_highPri PORT MAP(
66                     REQ_1 => REQ_1,
67                     REQ_2 => REQ_2,
68                     clk => clk,
69                     rst => rst,
70                     ACK_1 => ACK_1,
71                     ACK_2 => ACK_2
72                 );
73
74     -- Define clock behavior
75     clk_process :process
76     begin
77             clk <= '0';
78             wait for clk_period/2;
79             clk <= '1';
80             wait for clk_period/2;
81     end process;
82
83     -- Define simulation process
84     stim_proc: process
85     begin
86             rst <= '1';
87         wait for clk_period*3;
88             rst <= '0';
89         wait for clk_period*3;
90
91         --begin stimulus
92             REQ_1 <= '1';
93             REQ_2 <= '1';
94         wait for clk_period*3;
95             REQ_1 <= '1';
96             REQ_2 <= '0';
97         wait for clk_period*3;
98             REQ_1 <= '1';
99             REQ_2 <= '1';
100        wait for clk_period*3;
101            REQ_1 <= '0';
102            REQ_2 <= '1';
103        wait for clk_period*3;
104            REQ_1 <= '1';
105            REQ_2 <= '1';
106        wait for clk_period*3;
107            REQ_1 <= '1';
108            REQ_2 <= '0';
109        wait for clk_period*3;
110            REQ_1 <= '1';
111            REQ_2 <= '1';
112        wait for clk_period*3;
113            REQ_1 <= '0';
114            REQ_2 <= '0';
115        wait for clk_period*3;
116            REQ_1 <= '0';
117            REQ_2 <= '1';
118        wait for clk_period*3;
119
120            rst <= '1';
121        wait for clk_period*3;
122            rst <= '1';
123        wait for clk_period*3;
124
125        wait;
126    end process;
127
128 END;
```



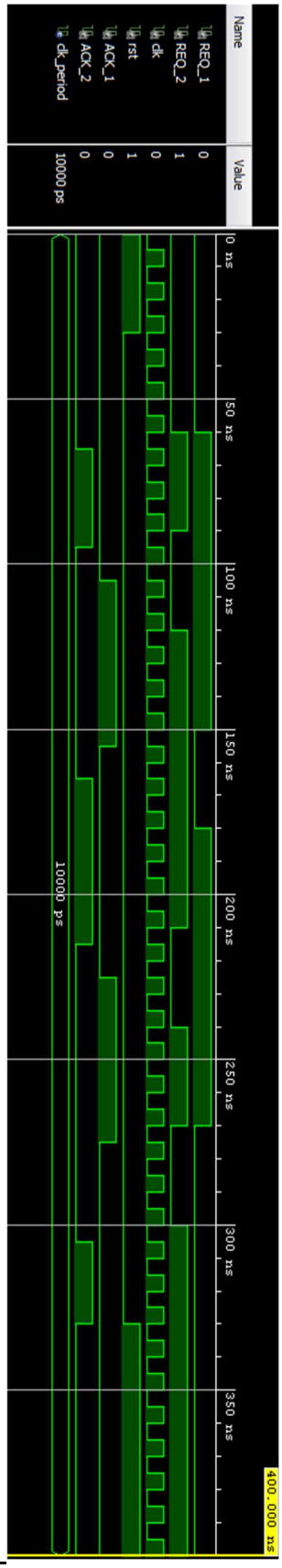
Behavioral VHDL code

```
1  ----------------------------------------------------------------------------------
2  -- Company: University of Kansas
3  -- Engineer: Dustin Horvath
4  --
5  -- Create Date: 09/28/2015 09:08:39 PM
6  -- Design Name: arbiter_behavioral_2cons_highPri
7  -- Module Name: arbiter_behavioral_2cons_highPri - Behavioral
8  -- Project Name: EECS645 Homework 2
9  --
10 -- Revision: See latest git commit at http://git.nichnologist.net/dhorvath/eecs/tree/master/eecs645/hw2
11 -- Additional Comments: None
12 --
13 ----------------------------------------------------------------------------------
14
15 LIBRARY ieee;
16 USE ieee.std_logic_1164.all;
17 USE ieee.std_logic_unsigned.all;
18
19 ENTITY arbiter_behavioral_2cons_highPri IS
20     -- Define toplevel I/O
21     PORT(
22         REQ_1  :  IN    std_logic;
23         REQ_2  :  IN    std_logic;
24         clk    :  IN    std_logic;
25         rst    :  IN    std_logic;
26         ACK_1  :  OUT   std_logic;
27         ACK_2  :  OUT   std_logic
28     );
29 END arbiter_behavioral_2cons_highPri;
30
31 ARCHITECTURE behavioral_highPri OF arbiter_behavioral_2cons_highPri IS
32
33     -- Alias STATE_TYPE is 2 bit vector
34     SUBTYPE STATE_TYPE IS
35         std_logic_vector (1 DOWNTO 0);
36
37     -- Hard coding for states
38     CONSTANT NO_ACCESS  :  STATE_TYPE  :=  "00"  ;
39     CONSTANT Con_01     :  STATE_TYPE  :=  "10"  ;
40     CONSTANT Con_02     :  STATE_TYPE  :=  "01"  ;
41     CONSTANT Unused     :  STATE_TYPE  :=  "11"  ;
42
43     -- Signals for states
44     SIGNAL current_state : STATE_TYPE;
45     SIGNAL next_state    : STATE_TYPE;
46
47     SIGNAL REQ_VEC       : std_logic_vector (1 TO 2);
48
49 BEGIN
```

```
50
51      REQ_VEC <= (REQ_1 & REQ_2);
52
53
54      -- Define memory elements and state cycle behavior
55      memory_elements  :  PROCESS(clk, rst)
56      BEGIN
57          IF (rst = '1') THEN
58              current_state <= NO_ACCESS;
59          ELSIF (clk'EVENT AND clk = '1') THEN
60              current_state <= next_state;
61          END IF;
62      END PROCESS memory_elements;
63
64      -- Define next state logic
65      state_logic  :  PROCESS (REQ_VEC, current_state)
66      BEGIN
67          CASE current_state IS
68          WHEN NO_ACCESS =>
69              next_state <= NO_ACCESS;
70              IF (REQ_VEC = "10") THEN
71                  next_state <= Con_01;
72              END IF;
73              IF (REQ_VEC = "01") THEN
74                  next_state <= Con_02;
75              END IF;
76              IF (REQ_VEC = "11") THEN
77                  next_state <= Con_02;
78              END IF;
79          WHEN Con_01 =>
80              next_state <= Con_01;
81              IF (REQ_VEC(1) = '0') THEN
82                  next_state <= NO_ACCESS;
83              END IF;
84          WHEN Con_02 =>
85              next_state <= Con_02;
86              IF (REQ_VEC(2) = '0') THEN
87                  next_state <= NO_ACCESS;
88              END IF;
89          WHEN OTHERS =>
90              next_state <= NO_ACCESS;
91          END CASE;
92      END PROCESS state_logic;
93
94      -- Define output logic
95      output_logic  :  PROCESS (current_state)
96      BEGIN
97          CASE current_state IS
98          WHEN Con_01 =>
99              ACK_1 <= '1';
100             ACK_2 <= '0';
101         WHEN Con_02 =>
102             ACK_1 <= '0';
103             ACK_2 <= '1';
104         WHEN OTHERS =>
105             ACK_1 <= '0';
106             ACK_2 <= '0';
107         END CASE;
108     END PROCESS output_logic;
109 END behavioral_highPri;
```

Testbench and Simulation Results

```
1  ----------------------------------------------------------------------------
2  -- Company: University of Kansas
3  -- Engineer: Dustin Horvath
4  --
5  -- Create Date: 09/28/2015 09:08:39 PM
6  -- Design Name: arbiter_behavioral_2cons_highPri_tb
7  -- Module Name: arbiter_behavioral_2cons_highPri_tb - Behavioral
8  -- Project Name: EECS645 Homework 2
9  --
10 -- Revision: See latest git commit at http://git.nichnologist.net/dhorvath/eecs/tree/master/eecs645/hw2
11 -- Additional Comments: None
12 --
13 ----------------------------------------------------------------------------
14
15 LIBRARY ieee;
16 USE ieee.std_logic_1164.ALL;
17
18 ENTITY arbiter_behavioral_2cons_highPri_tb IS
19 END arbiter_behavioral_2cons_highPri_tb;
20
21 ARCHITECTURE behavior OF arbiter_behavioral_2cons_highPri_tb IS
22
23     COMPONENT arbiter_behavioral_2cons_highPri
24         -- Define toplevel I/O
25         PORT(
26             REQ_1  : IN     std_logic;
27             REQ_2  : IN     std_logic;
28             clk    : IN     std_logic;
29             rst    : IN     std_logic;
30             ACK_1  : OUT    std_logic;
31             ACK_2  : OUT    std_logic
32         );
33     END COMPONENT;
34
35     -- Define interim signals
36     signal REQ_1  :  std_logic  := '0';
37     signal REQ_2  :  std_logic  := '0';
38     signal clk    :  std_logic  := '0';
39     signal rst    :  std_logic  := '0';
40
41
42     signal ACK_1  :  std_logic;
43     signal ACK_2  :  std_logic;
44
45     constant clk_period  :  time  := 10 ns;
46
47 BEGIN
48
49     -- Unit Under Test
50     -- Map elements of simulation to elements of UUT
51     uut: arbiter_behavioral_2cons_highPri PORT MAP(
52                     REQ_1 => REQ_1,
53                     REQ_2 => REQ_2,
54                     clk => clk,
55                     rst => rst,
56                     ACK_1 => ACK_1,
57                     ACK_2 -> ACK_2
58                 );
59
60     clk_process :process
61     begin
62             clk <= '0';
63             wait for clk_period/2;
64             clk <= '1';
65             wait for clk_period/2;
66     end process;
67
68
69     stim_proc: process
70     begin
71             rst <= '1';
72         wait for clk_period*3;
73             rst <= '0';
74         wait for clk_period*3;
75
76         --begin stimulus
77             REQ_1 <= '1';
78             REQ_2 <= '1';
79         wait for clk_period*3;
80             REQ_1 <= '1';
81             REQ_2 <= '0';
82         wait for clk_period*3;
83             REQ_1 <= '1';
84             REQ_2 <= '1';
85         wait for clk_period*3;
86             REQ_1 <= '0';
87             REQ_2 <= '1';
88         wait for clk_period*3;
89             REQ_1 <= '1';
90             REQ_2 <= '1';
91         wait for clk_period*3;
92             REQ_1 <= '1';
93             REQ_2 <= '0';
94         wait for clk_period*3;
95             REQ_1 <= '1';
96             REQ_2 <= '1';
97         wait for clk_period*3;
98             REQ_1 <= '0';
99             REQ_2 <= '0';
100        wait for clk_period*3;
101            REQ_1 <= '0';
102            REQ_2 <= '1';
103        wait for clk_period*3;
104
105            rst <= '1';
106        wait for clk_period*3;
107            rst <= '1';
108        wait for clk_period*3;
109
110        wait;
111    end process;
112
113 END;
```



Homework Problem II:

Given a resource that is to be shared by two consumers such that only one consumer has access to the resource at any given time. The policy of access is non-preemptive with priority determined by the consumer index. More specifically the policy is as follows:

- When the resource is idle and the consumers simultaneously request the resource, the consumers are prioritized according to their consumer number (i.e. consumer 1 has a higher priority than consumer 2)
- When the resource is being accessed by any of the consumers, the consumers are scheduled as First-Come-First-Served (FCFS)

Design a two-consumer arbiter/controller that controls the access to the shared resource and implements the above policy. In the design process, provide the following:

1) The system architecture (block diagram) showing the interface ports to the arbiter including the clock and reset signals.
2) Finite State Machine (FSM) diagram showing all possible states, transitions, and output values.
3) K-maps for internal state and output variables.
4) Boolean expressions for internal state and output variables.
5) Detailed logic diagram using synchronous memory elements showing internal connections and external interfaces.
6) Complete description of the arbiter using VHDL code.
7) Complete VHDL testbench along with simulation results.

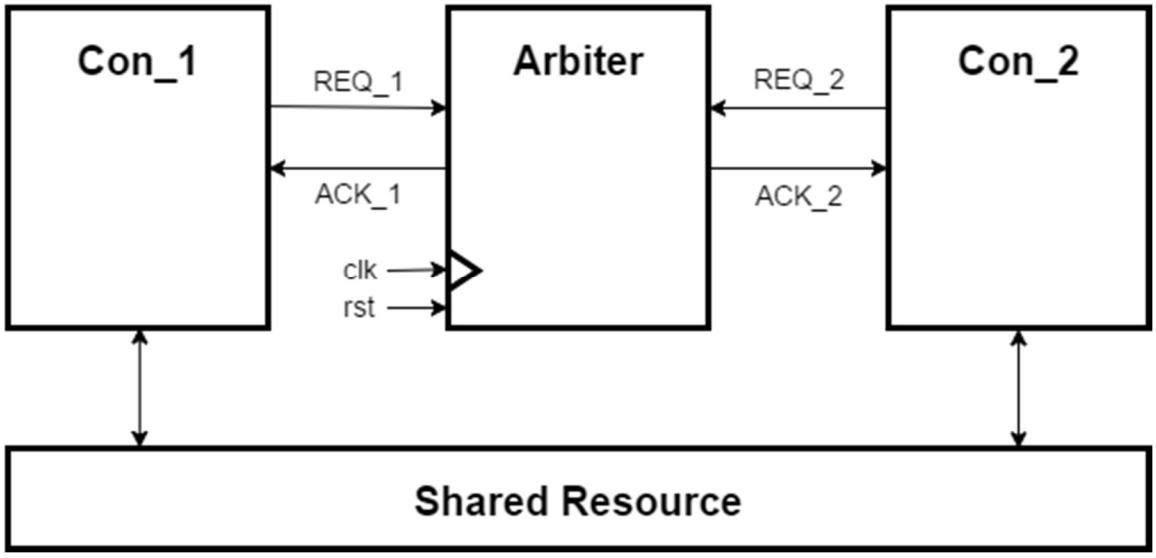


Figure 1: System architecture

Table 1: Input Code Assignment

Input Combinations (Input Codes)		Input Description
REQ_01	REQ_02	
0	0	No Requests
0	1	Consumer 2 requests resource
1	0	Consumer 1 requests resource
1	1	Both consumers request resource

Table 2: Output Code Assignment

Output Combinations (Output Codes)		Output Description
ACK_01	ACK_02	
0	0	None granted access to resource
0	1	Consumer 2 granted access to resource
1	0	Consumer 1 granted access to resource
1	1	Forbidden output

Table 3: State Code Assignment

State Description		State Codes	
		S_01	S_02
Required States	Resource is idle (No access)	0	0
	Resource is used by consumer 2 (Con_02)	0	1
	Resource is used by consumer 1 (Con_01)	1	0
Unused States	Extra unused state (Unused)	1	1

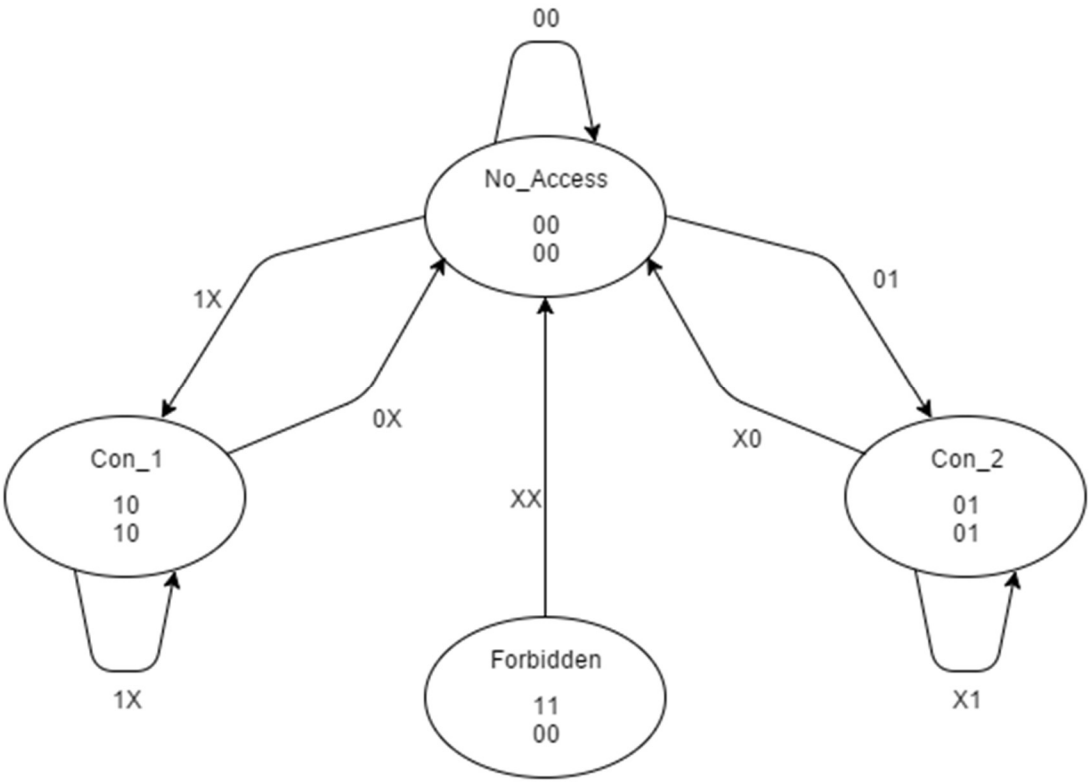


Figure 2: State transition diagram

Table 4: State transition table

Next State (S_1 , S_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00	00	01	10	10
	01	00	01	01	00
	11	00	00	00	00
	10	00	00	10	10

Table 5: K-map for the state variable S_1

Next State (S_1)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00			1	1
	01				
	11				
	10			1	1

$$S_1^{next} = \overline{S2\_current} * REQ\_1$$

Table 6: K-map for the state variable S_2

Next State (S_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00		1		
	01		1	1	
	11				
	10				

$$S_2^{next} = \overline{S1\_current} * \overline{REQ\_1} * REQ\_2 + \overline{S1\_current} * S2\_current * REQ\_2$$

Table 7: Output transition table

Outputs (ACK_1 , ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00	00	00	00	00
	01	01	01	01	01
	11	11	11	11	11
	10	10	10	10	10

Table 8: K-map for the output variable ACK_1

Outputs (ACK_1)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01				
	11				
	10	1	1	1	1

$$ACK_1 = S1\_current * \overline{S2\_current}$$

Table 9: K-map for the output variable ACK_2

Outputs (ACK_2)		Inputs (REQ_1 , REQ_2)			
		00	01	11	10
Current State (S_1 , S_2)	00				
	01	1	1	1	1
	11				
	10				

$$ACK_2 = \overline{S1\_current} * S2\_current$$

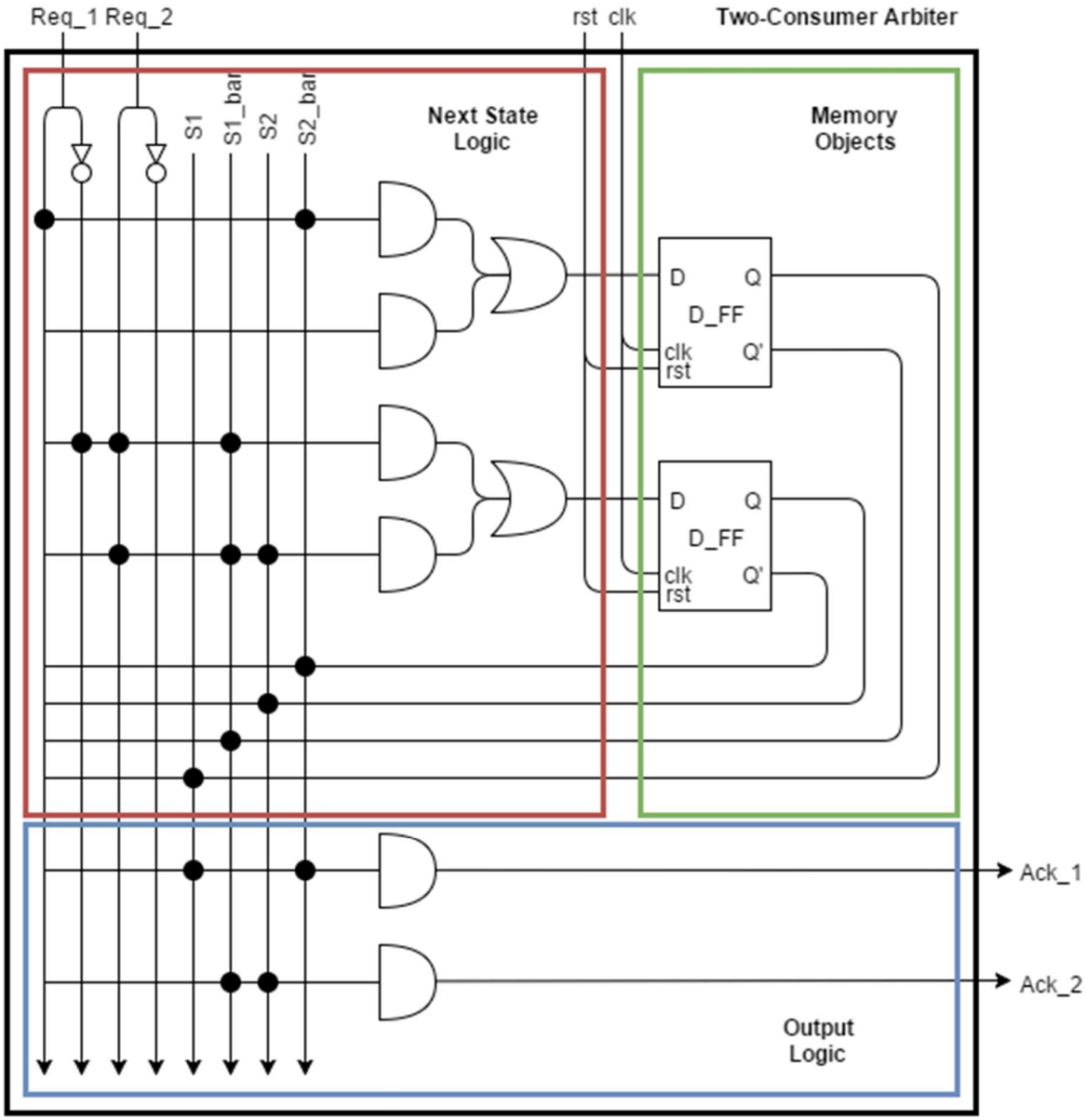


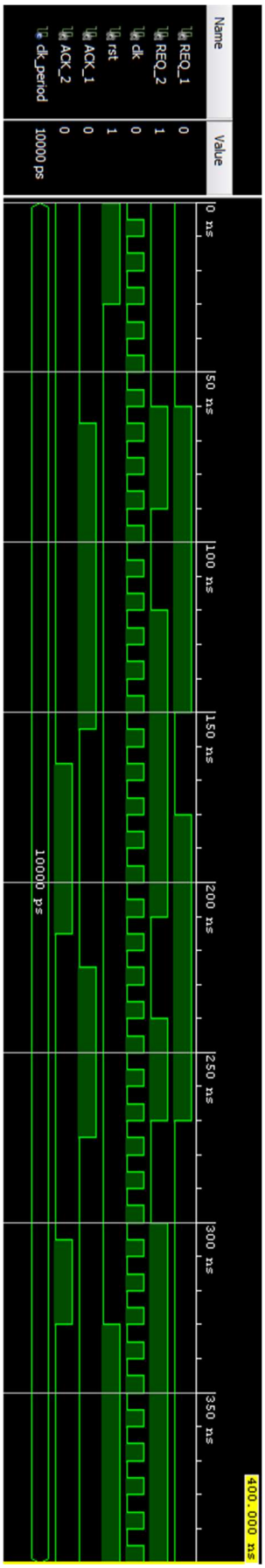
Figure 3: Synchronous logic diagram

Structural VHDL code

```
21
22 LIBRARY IEEE;
23 USE IEEE.STD_LOGIC_1164.ALL;
24 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
25
26 ENTITY arbiter_structural_2cons_lowPri IS
27     PORT(
28         REQ_1 : IN      STD_LOGIC;
29         REQ_2 : IN      STD_LOGIC;
30         clk   : IN      STD_LOGIC;
31         rst   : IN      STD_LOGIC;
32         ACK_1 : OUT     STD_LOGIC;
33         ACK_2 : OUT     STD_LOGIC
34     );
35 END arbiter_structural_2cons_lowPri;
36
37 ARCHITECTURE struct_low_index_priority of arbiter_structural_2cons_lowPri IS
38     SIGNAL s1_current , s2_current  :  std_logic;
39     SIGNAL s1_next    , s2_next     :  std_logic;
40
41 BEGIN
42     ---------------------------------
43     -- memory elements
44     ---------------------------------
45     memory_elements : PROCESS(clk, rst)
46
47     BEGIN
48         IF (rst = '1') THEN
49             s1_current <= '0';
50             s2_current <= '0';
51         ELSIF (clk'EVENT AND clk = '1') THEN
52             s1_current <= s1_next;
53             s2_current <= s2_next;
54         END IF;
55     END PROCESS memory_elements;
56
57     ---------------------------------
58     -- state logic
59     ---------------------------------
60
61     s1_next <= ((not s2_current) and REQ_1);
62     s2_next <= (((not s1_current) and (not REQ_1) and REQ_2) or ((not s1_current) and S2_current and REQ_2));
63
64     ---------------------------------
65     -- output logic
66     ---------------------------------
67
68     ACK_1 <= s1_current and (not s2_current);
69     ACK_2 <= s2_current and (not s1_current);
70
71 END struct_low_index_priority;
72
```

Testbench and Simulation Results

```
34 LIBRARY ieee;
35 USE ieee.std_logic_1164.ALL;
36
37 ENTITY arbiter_structural_2cons_lowPri_tb IS
38 END arbiter_structural_2cons_lowPri_tb;
39
40 ARCHITECTURE behavior OF arbiter_structural_2cons_lowPri_tb IS
41
42     COMPONENT arbiter_structural_2cons_lowPri
43      PORT(
44         REQ_1  : IN     std_logic;
45         REQ_2  : IN     std_logic;
46         clk    : IN     std_logic;
47         rst    : IN     std_logic;
48         ACK_1  : OUT    std_logic;
49         ACK_2  : OUT    std_logic
50     );
51     END COMPONENT;
52
53     signal REQ_1  :  std_logic  := '0';
54     signal REQ_2  :  std_logic  := '0';
55     signal clk    :  std_logic  := '0';
56     signal rst    :  std_logic  := '0';
57
58
59     signal ACK_1  :  std_logic;
60     signal ACK_2  :  std_logic;
61
62     constant clk_period  :  time  := 10 ns;
63
64 BEGIN
65
66     -- Unit Under Test
67     uut: arbiter_structural_2cons_lowPri PORT MAP(
68                     REQ_1 => REQ_1,
69                     REQ_2 => REQ_2,
70                     clk => clk,
71                     rst => rst,
72                     ACK_1 => ACK_1,
73                     ACK_2 => ACK_2
74                 );
75
76     clk_process :process
77     begin
78             clk <= '0';
79             wait for clk_period/2;
80             clk <= '1';
81             wait for clk_period/2;
82     end process;
83
84
85     stim_proc: process
86     begin
87             rst <= '1';
88         wait for clk_period*3;
89             rst <= '0';
90         wait for clk_period*3;
91
92         --begin stimulus
93             REQ_1 <= '1';
94             REQ_2 <= '1';
95         wait for clk_period*3;
96             REQ_1 <= '1';
97             REQ_2 <= '0';
98         wait for clk_period*3;
99             REQ_1 <= '1';
100            REQ_2 <= '1';
101        wait for clk_period*3;
102            REQ_1 <= '0';
103            REQ_2 <= '1';
104        wait for clk_period*3;
105            REQ_1 <= '1';
106            REQ_2 <= '1';
107        wait for clk_period*3;
108            REQ_1 <= '1';
109            REQ_2 <= '0';
110        wait for clk_period*3;
111            REQ_1 <= '1';
112            REQ_2 <= '1';
113        wait for clk_period*3;
114            REQ_1 <= '0';
115            REQ_2 <= '0';
116        wait for clk_period*3;
117            REQ_1 <= '0';
118            REQ_2 <= '1';
119        wait for clk_period*3;
120
121            rst <= '1';
122        wait for clk_period*3;
123            rst <= '1';
124        wait for clk_period*3;
125
126        wait;
127    end process;
128
129 END;
```

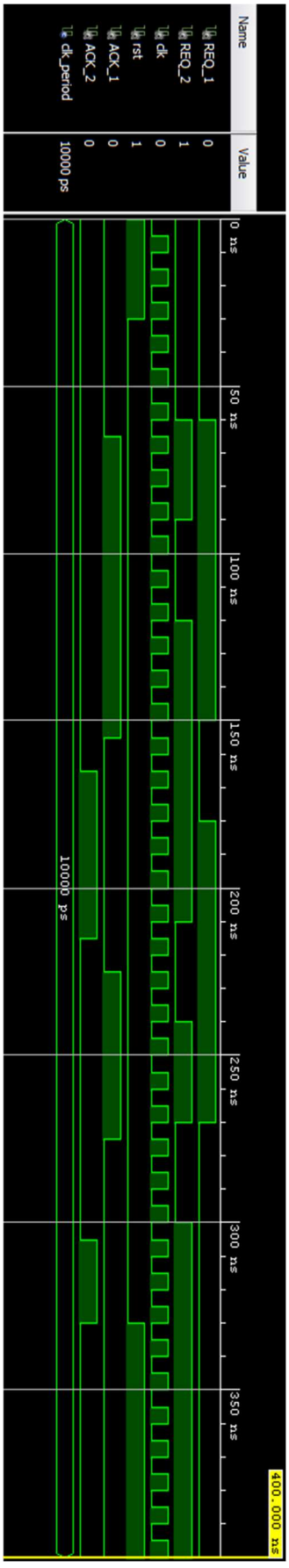


Behavioral VHDL code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY arbiter_behavioral_2cons_lowPri IS
    -- Define toplevel IO
    PORT(
        REQ_1  :  IN    std_logic;
        REQ_2  :  IN    std_logic;
        clk    :  IN    std_logic;
        rst    :  IN    std_logic;
        ACK_1  :  OUT   std_logic;
        ACK_2  :  OUT   std_logic
    );
END arbiter_behavioral_2cons_lowPri;

ARCHITECTURE behavioral_lowPri OF arbiter_behavioral_2cons_lowPri IS

    -- Alias STATE_TYPE is 2 bit vector
    SUBTYPE STATE_TYPE IS
        std_logic_vector (1 DOWNTO 0);

    -- Hard coding for states
    CONSTANT NO_ACCESS  :  STATE_TYPE  :=  "00"  ;
    CONSTANT Con_01     :  STATE_TYPE  :=  "10"  ;
    CONSTANT Con_02     :  STATE_TYPE  :=  "01"  ;
    CONSTANT Unused     :  STATE_TYPE  :=  "11"  ;

    -- Signals for states
    SIGNAL current_state : STATE_TYPE;
    SIGNAL next_state    : STATE_TYPE;

    SIGNAL REQ_VEC       : std_logic_vector (1 TO 2);

BEGIN

    REQ_VEC <= (REQ_1 & REQ_2);

    -- Define memory elements and state cycle behavior
    memory_elements  :  PROCESS(clk, rst)
    BEGIN
        IF (rst = '1') THEN
            current_state <= NO_ACCESS;
        ELSIF (clk'EVENT AND clk = '1') THEN
            current_state <= next_state;
        END IF;
    END PROCESS memory_elements;

    -- Define next state logic
    state_logic  :  PROCESS (REQ_VEC, current_state)
    BEGIN
        CASE current_state IS
        WHEN NO_ACCESS =>
            next_state <= NO_ACCESS;
            IF (REQ_VEC = "10") THEN
                next_state <= Con_01;
            END IF;
            IF (REQ_VEC = "01") THEN
                next_state <= Con_02;
            END IF;
            IF (REQ_VEC = "11") THEN
                next_state <= Con_01;
            END IF;
        WHEN Con_01 =>
            next_state <= Con_01;
            IF (REQ_VEC(1) = '0') THEN
                next_state <= NO_ACCESS;
            END IF;
        WHEN Con_02 =>
            next_state <= Con_02;
            IF (REQ_VEC(2) = '0') THEN
                next_state <= NO_ACCESS;
            END IF;
        WHEN OTHERS =>
            next_state <= NO_ACCESS;
        END CASE;
    END PROCESS state_logic;

    -- Define output logic
    output_logic  :  PROCESS (current_state)
    BEGIN
        CASE current_state IS
        WHEN Con_01 =>
            ACK_1 <= '1';
            ACK_2 <= '0';
        WHEN Con_02 =>
            ACK_1 <= '0';
            ACK_2 <= '1';
        WHEN OTHERS =>
            ACK_1 <= '0';
            ACK_2 <= '0';
        END CASE;
    END PROCESS output_logic;
END behavioral_lowPri;
```

Testbench and Simulation Results

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Define toplevel IO for simulation (empty)
ENTITY arbiter_behavioral_2cons_lowPri_tb IS
END arbiter_behavioral_2cons_lowPri_tb;

ARCHITECTURE behavior OF arbiter_behavioral_2cons_lowPri_tb IS

    COMPONENT arbiter_behavioral_2cons_lowPri
        -- Define IO for simulation
        PORT(
            REQ_1  : IN     std_logic;
            REQ_2  : IN     std_logic;
            clk    : IN     std_logic;
            rst    : IN     std_logic;
            ACK_1  : OUT    std_logic;
            ACK_2  : OUT    std_logic
        );
    END COMPONENT;

    -- Define interim signals
    signal REQ_1  :  std_logic  := '0';
    signal REQ_2  :  std_logic  := '0';
    signal clk    :  std_logic  := '0';
    signal rst    :  std_logic  := '0';


    signal ACK_1  :  std_logic;
    signal ACK_2  :  std_logic;

    constant clk_period  :  time  := 10 ns;

BEGIN

    -- Unit Under Test
    -- Map simulation elements to elements of UUT
    uut: arbiter_behavioral_2cons_lowPri PORT MAP(
                    REQ_1 => REQ_1,
                    REQ_2 => REQ_2,
                    clk => clk,
                    rst => rst,
                    ACK_1 => ACK_1,
                    ACK_2 => ACK_2
                );

    -- Define clock behavior
    clk_process :process
    begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
    end process;

    -- Begin simulation process
    stim_proc: process
    begin
        -- Prep with reset signal
            rst <= '1';
        wait for clk_period*3;
            rst <= '0';
        wait for clk_period*3;

        --begin stimulus
            REQ_1 <= '1';
            REQ_2 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
        wait for clk_period*3;

            rst <= '1';
        wait for clk_period*3;
            rst <= '1';
        wait for clk_period*3;

        wait;
    end process;

END;
```



Homework Problem III:

Given a resource that is to be shared by three consumers such that only one consumer has access to the resource at any given time. The policy of access is non-preemptive with no priority. More specifically the policy is as follows:

- When the resource is idle and the consumers simultaneously request the resource, their requests are ignored until only one request is submitted
- When the resource is being accessed by any of the consumers, the consumers are scheduled as First-Come-First-Served (FCFS)

Design a three-consumer arbiter/controller that controls the access to the shared resource and implements the above policy. In the design process, provide the following:

1) The system architecture (block diagram) showing the interface ports to the arbiter including the clock and reset signals.
2) Finite State Machine (FSM) diagram showing all possible states, transitions, and output values.
3) K-maps for internal state and output variables.
4) Boolean expressions for internal state and output variables.
5) Detailed logic diagram using synchronous memory elements showing internal connections and external interfaces.
6) Complete description of the arbiter using VHDL code.
7) Complete VHDL testbench along with simulation results.

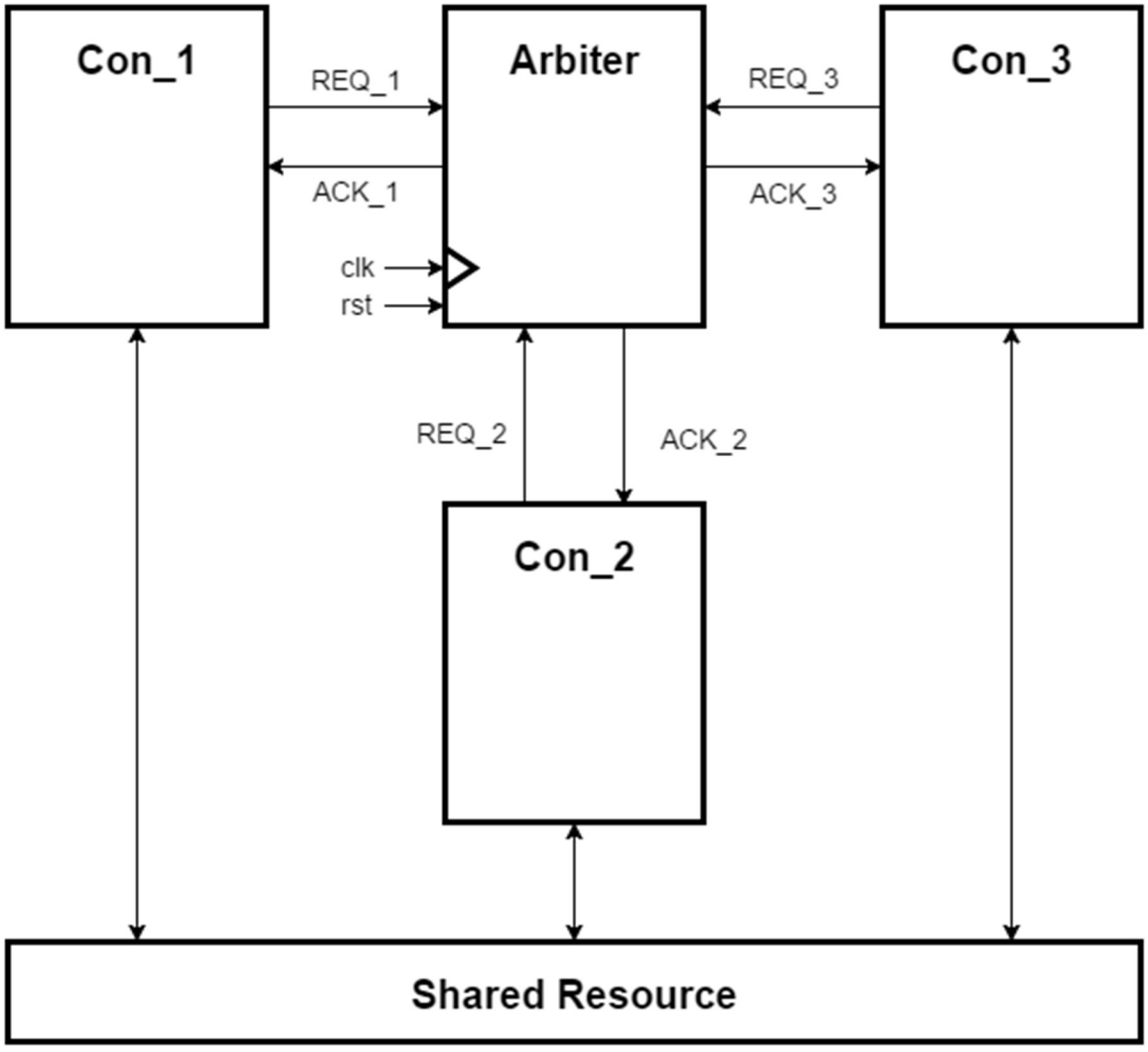


Figure 1: System architecture

Table 1: Input Code Assignment

Input Combinations (Input Codes)			Input Description
REQ_01	REQ_02	REQ_03	
0	0	0	No Requests
0	0	1	Consumer 3 requests resource
0	1	1	Consumers 2 and 3 requesting
0	1	0	Consumer 2 Requests resource
1	1	0	Consumers 1 and 2 requesting
1	1	1	Consumers 1, 2, and 3 requesting
1	0	1	Consumers 1 and 3 requesting
1	0	0	Consumer 1 requests resource

Table 2: Output Code Assignment

Output Combinations (Output Codes)			Output Description
ACK_01	ACK_02	ACK_03	
0	0	0	None granted access to resource
0	0	1	Consumer 3 granted access to resource
0	1	1	Forbidden output
0	1	0	Consumer 2 granted access to resource
1	1	0	Forbidden output
1	1	1	Forbidden output
1	0	1	Forbidden output
1	0	0	Consumer 1 granted access to resource

Table 3: State Code Assignment

State Description	State Codes	
	S_01	S_02
Resource is idle	0	0
Resource is used by consumer 3	0	1
Resource is used by consumer 1	1	0
Resource is used by consumer 2	1	1

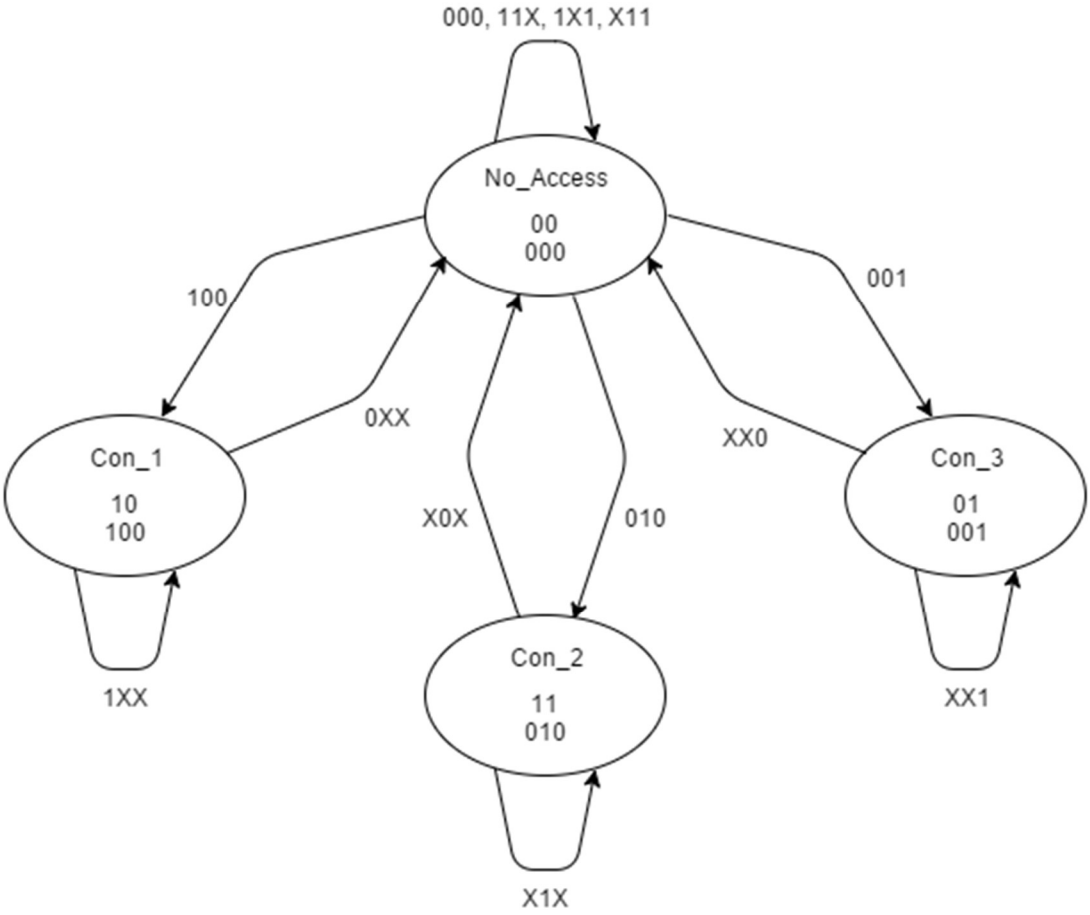


Figure 2: State transition diagram

Table 4: State transition table

Next State (S_1 , S_2)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00	00	01	00	11	00	00	00	10
	01	00	01	01	00	00	01	01	00
	11	00	00	11	11	11	11	00	00
	10	00	00	00	00	10	10	10	10

Table 5: K-map for the state variable S_1

Next State (S_1)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00				1				1
	01								
	11			1	1	1	1		
	10					1	1	1	1

$$S_1^{next} = S1\_current * S2\_current * REQ\_2$$
$$+ S1\_current * \overline{S2\_current} * REQ\_1$$
$$+ \overline{S2\_current} * REQ\_1 * \overline{REQ\_2} * \overline{REQ\_3}$$
$$+ \overline{S1\_current} * \overline{S2\_current} + \overline{REQ\_1} * REQ\_2 * \overline{REQ\_3}$$

Table 6: K-map for the state variable S_2

Next State (S_2)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00		1		1				
	01		1	1			1	1	
	11			1	1	1	1		
	10								

$$S_2^{next} = \overline{S1\_current} * \overline{REQ\_1} * \overline{REQ\_2} * REQ\_3$$
$$+ S2\_current * \overline{REQ\_1} * REQ\_2 * REQ\_3$$
$$+ S1\_current * S2\_current * REQ\_2$$
$$+ \overline{S1\_current} * S2\_current * REQ\_1 * REQ\_3$$
$$+ \overline{S1\_current} * \overline{S2\_current} * \overline{REQ\_1} * REQ\_2 * \overline{REQ\_3}$$

Table 7: Output transition table

Outputs (ACK_1 , ACK_2 , ACK_3)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00	000	000	000	000	000	000	000	000
	01	001	001	001	001	001	001	001	001
	11	010	010	010	010	010	010	010	010
	10	100	100	100	100	100	100	100	100

Table 8: K-map for the output variable ACK_1

Outputs (ACK_1)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00								
	01								
	11								
	10	1	1	1	1	1	1	1	1

$$ACK_1 = S1\_current * \overline{S2\_current}$$

Table 9: K-map for the output variable ACK_2

Outputs (ACK_2)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00								
	01								
	11	1	1	1	1	1	1	1	1
	10								

$$ACK_2 = S1\_current * S2\_current$$

Table 10: K-map for the output variable ACK_3

Outputs (ACK_3)		Inputs (REQ_1 , REQ_2 , REQ_3)							
		000	001	011	010	110	111	101	100
Current State (S_1 , S_2)	00								
	01	1	1	1	1	1	1	1	1
	11								
	10								

$$ACK_3 = \overline{S1\_current} * S2\_current$$

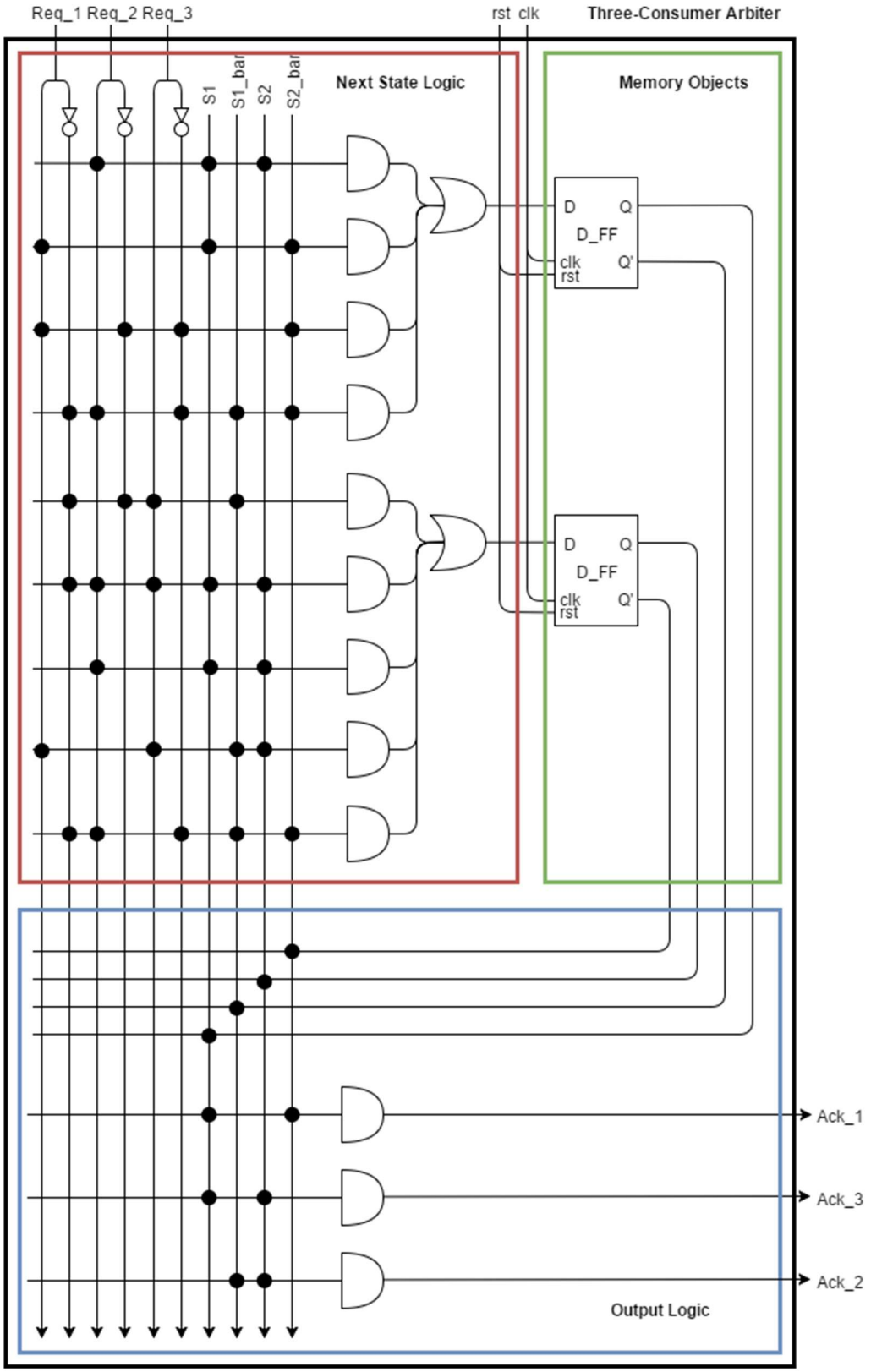


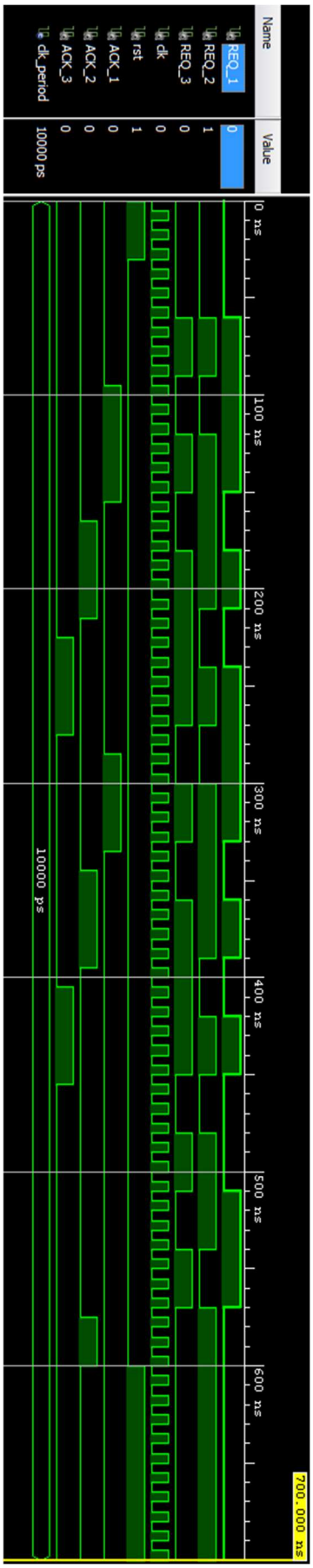
Figure 3: Synchronous logic diagram

Structural VHDL code

```
16 library IEEE;
17 use IEEE.STD_LOGIC_1164.ALL;
18 use IEEE.STD_LOGIC_UNSIGNED.ALL;
19
20 entity arbiter_structural_3cons is
21     -- Define toplevel IO
22     PORT(
23         REQ_1 : IN      STD_LOGIC;
24         REQ_2 : IN      STD_LOGIC;
25         REQ_3 : IN      STD_LOGIC;
26         clk   : IN      STD_LOGIC;
27         rst   : IN      STD_LOGIC;
28         ACK_1 : OUT     STD_LOGIC;
29         ACK_2 : OUT     STD_LOGIC;
30         ACK_3 : OUT     STD_LOGIC
31     );
32 end arbiter_structural_3cons;
33
34 architecture struct_no_priority of arbiter_structural_3cons is
35     SIGNAL s1_current , s2_current  :  std_logic;
36     SIGNAL s1_next    , s2_next     :  std_logic;
37
38 begin
39
40     -- Define memory elements
41     memory_elements : PROCESS(clk, rst)
42
43     BEGIN
44         IF (rst = '1') THEN
45             s1_current <= '0';
46             s2_current <= '0';
47         ELSIF (clk'EVENT AND clk = '1') THEN
48             s1_current <= s1_next;
49             s2_current <= s2_next;
50         END IF;
51     END PROCESS memory_elements;
52
53     -- Define state logic
54     s1_next <= (    (s1_current and s2_current and REQ_2) or
55                     (s1_current and (not s2_current) and REQ_1) or
56                     ((not s2_current) and REQ_1 and (not REQ_2) and (not REQ_3)) or
57                     ((not s1_current) and (not s2_current) and (not REQ_1) and REQ_2 and (not REQ_3))
58                     );
59     s2_next <= (    ((not s1_current) and (not REQ_1) and (not REQ_2) and REQ_3) or
60                     (s2_current and (not REQ_1) and REQ_2 and REQ_3) or
61                     (s1_current and s2_current and REQ_2) or
62                     ((not s1_current) and s2_current and REQ_1 and REQ_3) or
63                     ((not s1_current) and (not s2_current) and (not REQ_1) and REQ_2 and (not REQ_3))
64                     );
65
66     -- Define output logic
67     ACK_1 <= s1_current and (not s2_current);
68     ACK_2 <= s1_current and s2_current;
69     ACK_3 <= (not s1_current) and s2_current;
70
71 end struct_no_priority;
```

Testbench and Simulation Results

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY arbiter_structural_3cons_tb IS
END arbiter_structural_3cons_tb;

ARCHITECTURE behavior OF arbiter_structural_3cons_tb IS

    COMPONENT arbiter_structural_3cons
     PORT(
        REQ_1  : IN     std_logic;
        REQ_2  : IN     std_logic;
        REQ_3  : IN     std_logic;
        clk    : IN     std_logic;
        rst    : IN     std_logic;
        ACK_1  : OUT    std_logic;
        ACK_2  : OUT    std_logic;
        ACK_3  : OUT    std_logic
    );
    END COMPONENT;

    signal REQ_1  :  std_logic  := '0';
    signal REQ_2  :  std_logic  := '0';
    signal REQ_3  :  std_logic  := '0';
    signal clk    :  std_logic  := '0';
    signal rst    :  std_logic  := '0';


    signal ACK_1  :  std_logic;
    signal ACK_2  :  std_logic;
    signal ACK_3  :  std_logic;
    constant clk_period  :  time  := 10 ns;

BEGIN

    -- Unit Under Test
    uut: arbiter_structural_3cons PORT MAP(
                    REQ_1 => REQ_1,
                    REQ_2 => REQ_2,
                    REQ_3 => REQ_3,
                    clk => clk,
                    rst => rst,
                    ACK_1 => ACK_1,
                    ACK_2 => ACK_2,
                    ACK_3 => ACK_3
                );

    clk_process :process
    begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
    end process;

    stim_proc: process
    begin
            rst <= '1';
        wait for clk_period*3;
            rst <= '0';
        wait for clk_period*3;

        --begin stimulus
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;


        -- Final reset
            rst <= '1';
        wait for clk_period*3;
            rst <= '1';
        wait for clk_period*3;

        wait;
    end process;

END;
```

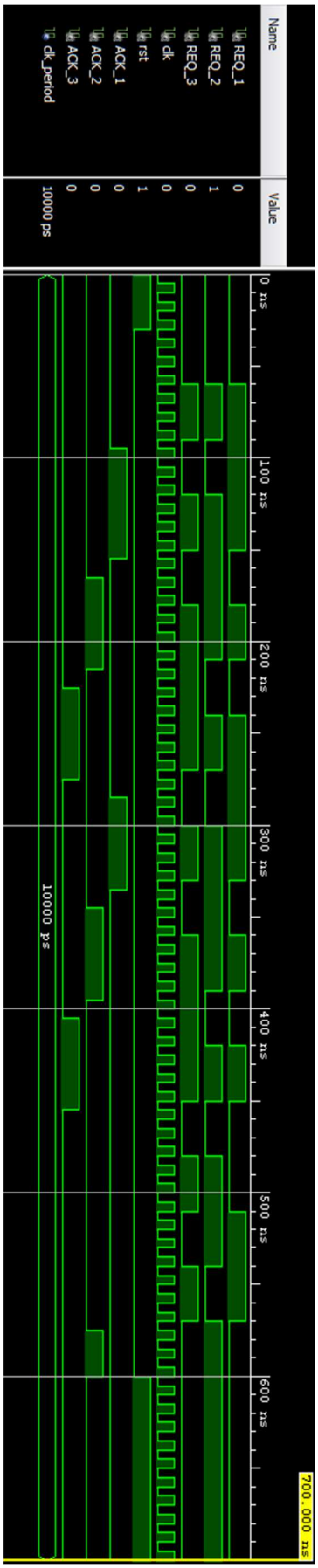

Name
Value
REQ_1
0
REQ_2
1
REQ_3
0
clk
0
rst
1
ACK_1
0
ACK_2
0
ACK_3
0
clk_period
10000 ps
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700.000 ns
10000 ps

Behavioral VHDL code

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY arbiter_behavioral_3cons IS
    PORT(
        REQ_1  :  IN    std_logic;
        REQ_2  :  IN    std_logic;
        REQ_3  :  IN    std_logic;
        clk    :  IN    std_logic;
        rst    :  IN    std_logic;
        ACK_1  :  OUT   std_logic;
        ACK_2  :  OUT   std_logic;
        ACK_3  :  OUT   std_logic
    );
END arbiter_behavioral_3cons;

ARCHITECTURE behavioral_3cons OF arbiter_behavioral_3cons IS

    SUBTYPE STATE_TYPE IS
        std_logic_vector (1 DOWNTO 0);

    -- Hard coding for states
    CONSTANT NO_ACCESS  :  STATE_TYPE  :=  "00"  ;
    CONSTANT Con_01     :  STATE_TYPE  :=  "10"  ;
    CONSTANT Con_02     :  STATE_TYPE  :=  "11"  ;
    CONSTANT Con_03     :  STATE_TYPE  :=  "01"  ;

    -- Signals for states
    SIGNAL current_state : STATE_TYPE;
    SIGNAL next_state    : STATE_TYPE;

    SIGNAL REQ_VEC       : std_logic_vector (1 TO 3);

BEGIN

    REQ_VEC <= (REQ_1 & REQ_2 & REQ_3);

    ---------------------------------
    -- Memory elements defined
    ---------------------------------

    memory_elements  :  PROCESS(clk, rst)
    BEGIN
        IF (rst = '1') THEN
            current_state <= NO_ACCESS;
        ELSIF (clk'EVENT AND clk = '1') THEN
            current_state <= next_state;
        END IF;
    END PROCESS memory_elements;

    ---------------------------------
    -- State logic defined
    ---------------------------------

    state_logic  :  PROCESS (REQ_VEC, current_state)
    BEGIN
        CASE current_state IS
        WHEN NO_ACCESS =>
            next_state <= NO_ACCESS;
            IF (REQ_VEC = "100") THEN
                next_state <= Con_01;
            END IF;
            IF (REQ_VEC = "010") THEN
                next_state <= Con_02;
            END IF;
            IF (REQ_VEC = "001") THEN
                next_state <= Con_03;
            END IF;
        WHEN Con_01 =>
            next_state <= Con_01;
            IF (REQ_VEC(1) = '0') THEN
                next_state <= NO_ACCESS;
            END IF;
        WHEN Con_02 =>
            next_state <= Con_02;
            IF (REQ_VEC(2) = '0') THEN
                next_state <= NO_ACCESS;
            END IF;
        WHEN Con_03 =>
            next_state <= Con_03;
            IF (REQ_VEC(3) = '0') THEN
                next_state <= NO_ACCESS;
            END IF;
        WHEN OTHERS =>
            next_state <= NO_ACCESS;
        END CASE;
    END PROCESS state_logic;

    ---------------------------------
    -- Define output logic
    ---------------------------------

    output_logic  :  PROCESS (current_state)
    BEGIN
        CASE current_state IS
        WHEN Con_01 =>
            ACK_1 <= '1';
            ACK_2 <= '0';
            ACK_3 <= '0';
        WHEN Con_02 =>
            ACK_1 <= '0';
            ACK_2 <= '1';
            ACK_3 <= '0';
        WHEN Con_03 =>
            ACK_1 <= '0';
            ACK_2 <= '0';
            ACK_3 <= '1';
        WHEN OTHERS =>
            ACK_1 <= '0';
            ACK_2 <= '0';
            ACK_3 <= '0';
        END CASE;
    END PROCESS output_logic;
END behavioral_3cons;
```

Testbench and Simulation Results

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY arbiter_structural_3cons_tb IS
END arbiter_structural_3cons_tb;

ARCHITECTURE behavior OF arbiter_structural_3cons_tb IS

    COMPONENT arbiter_structural_3cons
     PORT(
        REQ_1  : IN     std_logic;
        REQ_2  : IN     std_logic;
        REQ_3  : IN     std_logic;
        clk    : IN     std_logic;
        rst    : IN     std_logic;
        ACK_1  : OUT    std_logic;
        ACK_2  : OUT    std_logic;
        ACK_3  : OUT    std_logic
    );
    END COMPONENT;

    signal REQ_1  :  std_logic  := '0';
    signal REQ_2  :  std_logic  := '0';
    signal REQ_3  :  std_logic  := '0';
    signal clk    :  std_logic  := '0';
    signal rst    :  std_logic  := '0';


    signal ACK_1  :  std_logic;
    signal ACK_2  :  std_logic;
    signal ACK_3  :  std_logic;
    constant clk_period  :  time  := 10 ns;

BEGIN

    -- Unit Under Test
    uut: arbiter_structural_3cons PORT MAP(
                    REQ_1 => REQ_1,
                    REQ_2 => REQ_2,
                    REQ_3 => REQ_3,
                    clk => clk,
                    rst => rst,
                    ACK_1 => ACK_1,
                    ACK_2 => ACK_2,
                    ACK_3 => ACK_3
                );

    clk_process :process
    begin
            clk <= '0';
            wait for clk_period/2;
            clk <= '1';
            wait for clk_period/2;
    end process;


    stim_proc: process
    begin
            rst <= '1';
        wait for clk_period*3;
            rst <= '0';
        wait for clk_period*3;

        --begin stimulus
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '0';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;
            REQ_1 <= '1';
            REQ_2 <= '0';
            REQ_3 <= '1';
        wait for clk_period*3;
            REQ_1 <= '0';
            REQ_2 <= '1';
            REQ_3 <= '0';
        wait for clk_period*3;


        -- Final reset
            rst <= '1';
        wait for clk_period*3;
            rst <= '1';
        wait for clk_period*3;

        wait;
    end process;

END;
```


Name
Value
REQ_1
REQ_2
REQ_3
clk
rst
ACK_1
ACK_2
ACK_3
clk_period
0
1
0
0
1
0
0
0
10000 ps
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700.000 ns
10000 ps