Spring 2019

California State University, Northridge

Department of Electrical and Computer Engineering

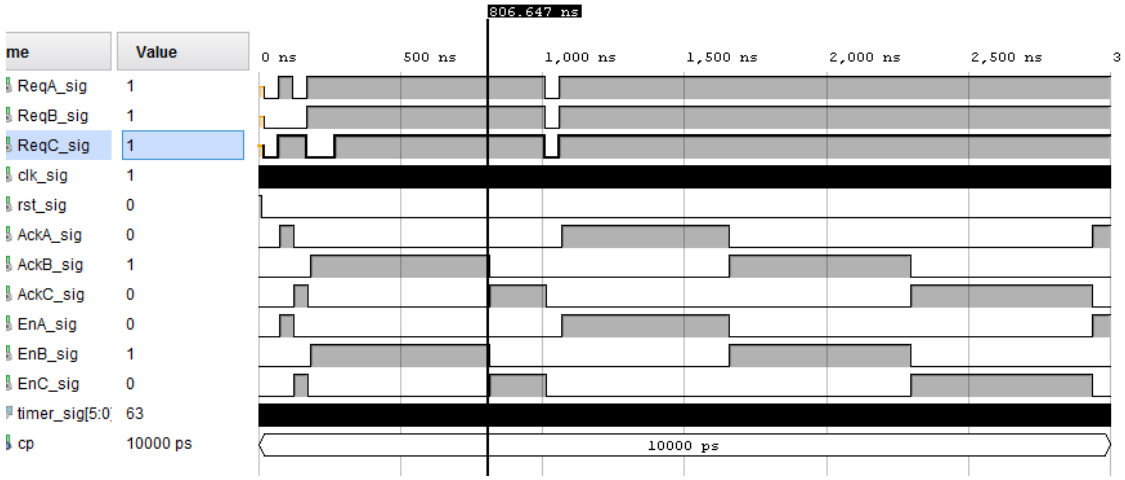
Computer Assignment 5: Round Robin Arbiter

May 01, 2019

Professor: Shahnam Mirzaei Ph.D

Authors:
Ridge Tejuco

Overall Result (Figure 1)



As a prerequisite to our design, Priority began with A.
Looking closer in Figure 2, the following values were input initially

ReqA: 1
ReqB: 0 - Processor A is acknowledged and given access.
ReqC: 1 - After this input priority moved from A to B.

ReqA: 0 - Processor B has priority, but is not requesting, so C
ReqB: 0 is given access
ReqC: 1 - Processor B still has priority after this

ReqA: 1 - C relinquishes access and ReqA and ReqB are high.
ReqB: 1 - Since, B still has priority and is requesting access
ReqC: 0 It is expected that AckB/ EnB is high as seen
in Figure 2.
- After this C should now have priority.

(Figure 2)

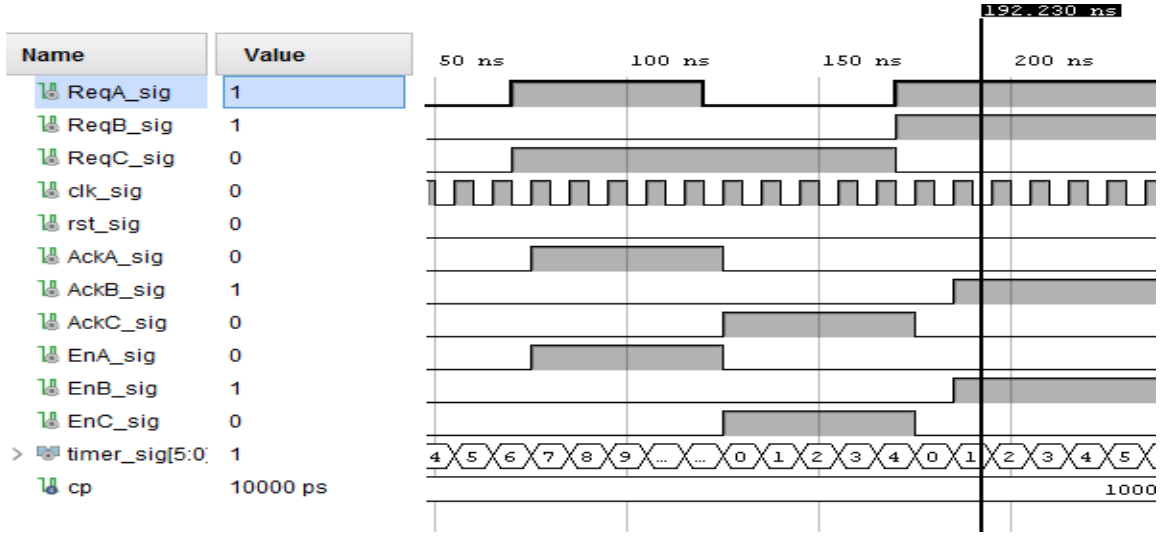
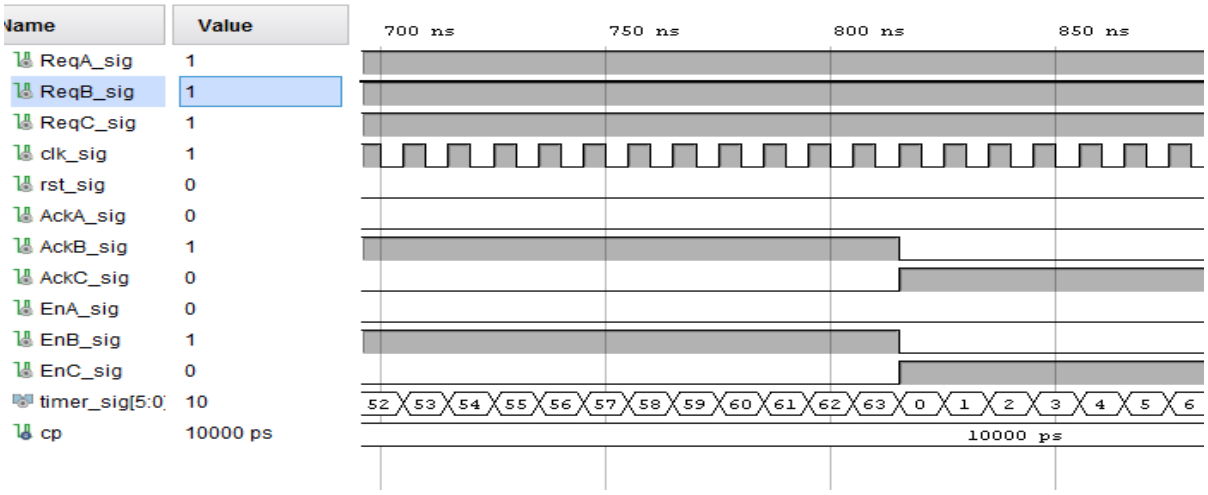


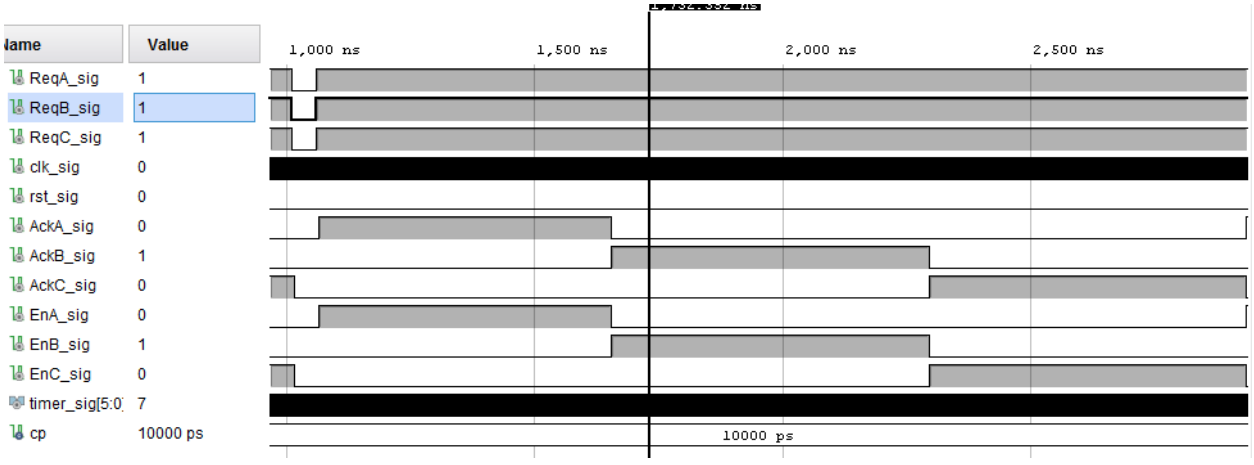
Figure 2 also shows how the timer resets after changing states before reaching the 64th cycle. (The timer counts from 0 to 4.)

The next input was seen in Figure 3.
ReqA: 1 - C still had priority, so as expected access was given
ReqB: 1 to processor C.
ReqC: 1 - Notice that this transition occurs at the 64th cycle
(Figure 3)



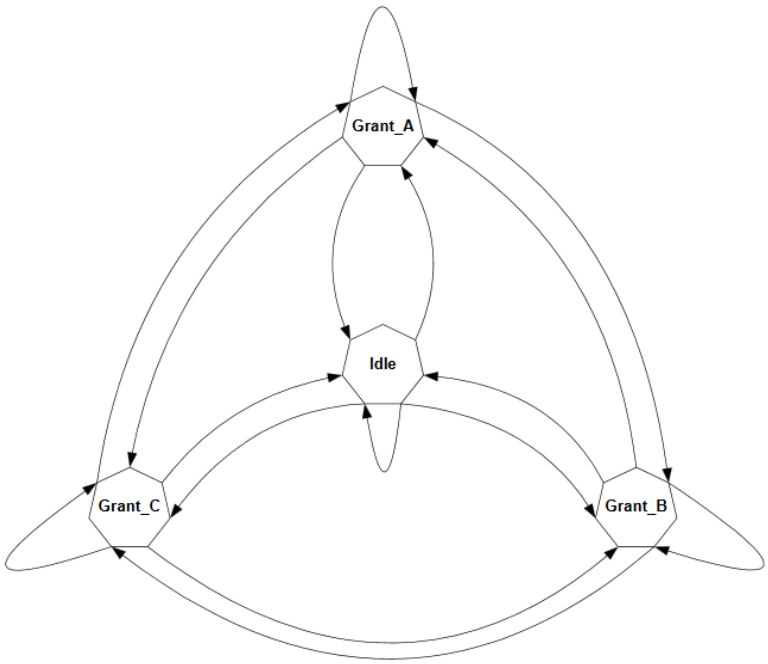
Then the input was transitioned to an Idle state by the following input.
ReqA: 0 - all access is relinquished.
ReqB: 0 - Priority has returned to A.
ReqC: 0

(Figure 4)



Then the Arbiter was set to run with all processors requesting access, which just shows that it changes access every 64 cycles by changing priority.

Complete State diagram



Modified Block diagram showing Priority combinational logic.

```
-- ARBITER_TOP.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY arbiter_top IS
	PORT (
		ReqA, ReqB, ReqC, clk, rst : IN std_logic;
		AckA, AckB, AckC, EnA, EnB, EnC : OUT std_logic;
		timerOut : OUT unsigned(5 DOWNTO 0)
	);
END arbiter_top;
ARCHITECTURE Behavioral OF arbiter_top IS
	TYPE state_type IS (idle, grant_a, grant_b, grant_c);
	SIGNAL timer64, timer_next : unsigned(5 DOWNTO 0);
	SIGNAL current_state, next_state : state_type;
	SIGNAL priority_A : std_logic := '1';
	SIGNAL priority_B, priority_C : std_logic := '0';
	SIGNAL PReqA, PReqB, PReqC : std_logic;
	SIGNAL req_A, req_B, req_C : std_logic;
	SIGNAL timer_reset : std_logic;
BEGIN
	-- current state logic and timer64
	PROCESS (clk, rst)
	BEGIN
		IF (rst = '1') THEN
			current_state <= idle;
			timer64 <= (OTHERS => '0');
		ELSIF (rising_edge(clk)) THEN
			current_state <= next_state;
			IF (timer_reset = '1') THEN
				timer64 <= (OTHERS => '0');
			ELSE
				timer64 <= timer_next;
			END IF;
		END IF;
	END PROCESS;
	timer_next <= timer64 + 1;
	-- current priority logic
	-- disables other requests
	PReqA <= ReqA AND priority_A;
	PReqB <= ReqB AND priority_B;
	PReqC <= ReqC AND priority_C;
	-- only one of these can be on if any are at all.
	req_A <= ReqA AND NOT(PReqB OR PReqC);
	req_B <= ReqB AND NOT(PReqC OR PReqA);
	req_C <= ReqC AND NOT(PReqA OR PReqB);

	PROCESS (current_state, req_A, req_B, req_C,
```

```
priority_A, priority_B, priority_C, timer64)
      BEGIN
            --next state logic with next priority logic
            timer_reset <= '0';
            CASE current_state IS
                  WHEN idle =>
                        IF (req_A = '1') THEN
                              next_state <= grant_a;
                        ELSIF req_B = '1' THEN
                              next_state <= grant_b;
                        ELSIF req_C = '1' THEN
                              next_state <= grant_c;
                        END IF;
                  WHEN grant_a =>
                        IF (req_A = '0' OR timer64 = "111111") THEN
                              timer_reset <= '1';
                              IF (priority_A = '1') THEN
                                    priority_A <= '0';
                                    priority_B <= '1';
                                    priority_C <= '0';
                              END IF;
                              IF (req_B = '1') THEN
                                    next_state <= grant_b;
                              ELSIF (req_C = '1') THEN
                                    next_state <= grant_c;
                              ELSE
                                    next_state <= idle;
                              END IF;
                        END IF;
                  WHEN grant_b =>
                        IF (req_B = '0' OR timer64 = "111111") THEN
                              timer_reset <= '1';
                              IF (priority_B = '1') THEN
                                    priority_A <= '0';
                                    priority_B <= '0';
                                    priority_C <= '1';
                              END IF;
                              IF (req_C = '1') THEN
                                    next_state <= grant_c;
                              ELSIF (req_A = '1') THEN
                                    next_state <= grant_a;
                              ELSE
                                    next_state <= idle;
                              END IF;
                        END IF;
                  WHEN grant_c =>
                        IF (req_C = '0' OR timer64 = "111111") THEN
                              timer_reset <= '1';
```

```
                                    IF (priority_C = '1') THEN
                                          priority_A <= '1';
                                          priority_B <= '0';
                                          priority_C <= '0';
                                    END IF;
                                    IF (req_A = '1') THEN
                                          next_state <= grant_a;
                                    ELSIF (req_C = '1') THEN
                                          next_state <= grant_b;
                                    ELSE
                                          next_state <= idle;
                                    END IF;
                              END IF;
                  END CASE;
                  --output logic
                  CASE current_state IS
                        WHEN idle =>
                              AckA <= '0';
                              AckB <= '0';
                              AckC <= '0';
                              EnA <= '0';
                              EnB <= '0';
                              EnC <= '0';
                        WHEN grant_a =>
                              AckA <= '1';
                              AckB <= '0';
                              AckC <= '0';
                              EnA <= '1';
                              EnB <= '0';
                              EnC <= '0';
                        WHEN grant_b =>
                              AckA <= '0';
                              AckB <= '1';
                              AckC <= '0';
                              EnA <= '0';
                              EnB <= '1';
                              EnC <= '0';
                        WHEN grant_c =>
                              AckA <= '0';
                              AckB <= '0';
                              AckC <= '1';
                              EnA <= '0';
                              EnB <= '0';
                              EnC <= '1';
                  END CASE;
                  timerOut <= timer64;
            END PROCESS;
END Behavioral;
```

```
-- Arbiter_tb.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
entity arbiter_tb is
--  Port ( );
end arbiter_tb;
architecture Behavioral of arbiter_tb is
      signal ReqA_sig,ReqB_sig,ReqC_sig: std_logic;
      signal clk_sig,rst_sig: std_logic := '0';
      signal AckA_sig,AckB_sig,AckC_sig,EnA_sig,EnB_sig,EnC_sig: std_logic;
      signal timer_sig: unsigned(5 downto 0);
      constant cp: time := 10ns;
      component arbiter_top is
      port(ReqA,ReqB,ReqC,clk,rst: in std_logic;
            AckA,AckB,AckC,EnA,EnB,EnC:out std_logic;
            timerOut:out unsigned(5 downto 0));
      end component;
begin
      UUT: arbiter_top
      port map(ReqA => ReqA_sig,ReqB => ReqB_sig,
                  ReqC => ReqC_sig,clk => clk_sig,
                  rst => rst_sig,AckA => AckA_sig,
                  AckB => AckB_sig,AckC => AckC_sig,
                  EnA => EnA_sig,EnB => EnB_sig, EnC =>EnC_sig,
                  timerOut => timer_sig);
      process(clk_sig)
      begin
      clk_sig <= not clk_sig after cp/2;
      end process;
      process
      begin
      rst_sig <= '1';wait for cp;rst_sig <= '0'; wait for cp;
      ReqA_sig <= '0'; ReqB_sig <= '0'; ReqC_sig <= '0';
      wait for 5*cp;
      ReqA_sig <= '1'; ReqB_sig <= '0'; ReqC_sig <= '0';
      wait for 5*cp;
      ReqA_sig <= '1'; ReqB_sig <= '1'; ReqC_sig <= '1';
      wait for 5*cp;
      ReqA_sig <= '0'; ReqB_sig <= '1'; ReqC_sig <= '1';
      wait for 69*cp;
      ReqA_sig <= '0'; ReqB_sig <= '0'; ReqC_sig <= '0';
      wait for 5*cp;
      ReqA_sig <= '1'; ReqB_sig <= '1'; ReqC_sig <= '1';
      wait for 192*cp;
      wait;
      end process;
end Behavioral;
```