ΗΡΥ203 Προχωρημένη Λογική Σχεδίαση
2η Εργαστηριακή Άσκηση

Ομάδα LAB20332002
Παντουράκης Μιχαήλ ΑΜ 2015030185
Τυχάλας Πέτρος ΑΜ 2015030169
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

26 Μαρτίου 2017

1 Σκοπός Εργαστηριακής Άσκησης

Η δεύτερη εργαστηριακή άσκηση συνεχίζει την εξοικείωση των φοιτητών στη γλώσσα VHDL, με το σχεδιασμό δομικών μονάδων με behavioral τρόπο και την μετέπειτα χρήση τους για την ιεραρχική σχεδίαση μεγαλύτερων κυκλωμάτων. Για το σκοπό αυτό η άσκηση ζητά την υλοποίηση δύο κυκλωμάτων: 1) ενός 4-bit αθροιστή Carry Look Ahead (CLA), ο οποίος αποτελεί ένα πρώτο παράδειγμα top-down ανάλυσης κυκλώματος και στη συνέχεια bottom-up υλοποίησής του με μικρότερες δομικές μονάδες, και 2) μιας μηχανής πεπερασμένων καταστάσεων, η οποία και αποτελεί χαρακτηριστικό παράδειγμα χρήσης του ρολογιού, των processes και case statements για σύγχρονα ακολουθιακά κυκλώματα.

2 Προεργασία-Περιγραφή

Η προεργασία της παρούσας άσκησης αφορούσε μόνο την συγγραφή του αντίστοιχου κώδικα σε VHDL, καθώς η ανάλυση και σχεδίαση των κυκλωμάτων με τα αντίστοιχα block diagrams έχουν ήδη δοθεί από την εκφώνηση της άσκησης.

2.1 4-bit CLA

Στο Παράρτημα 5.1 παρουσιάζουμε τον top module κώδικα του CLA. Σε αυτόν φαίνεται ξεκάθαρα η ιεραρχική σχεδίασή του με units τριών ειδών: τα Carry Generate-Propagate (CarryGP), Carry Look Ahead (CarryLA), Sum. Ειδικά στην υλοποίησή μας, αντί για μία 4-bit μονάδα CarryGP και Sum, στο top module χρησιμοποιήσαμε τέσσερα αντίγραφα των Sum και CarryGP, ένα για κάθε bit. Εναλλακτικά θα μπορούσαμε τα for-generate statements να τα χρησιμοποιήσουμε σε κατώτερο ιεραρχικά επίπεδα (δηλαδή σε άλλο αρχείο), όπου δηλαδή θα ορίζαμε τις αντίστοιχες 4-bit μονάδες και να χρησιμοποιήσουμε αυτές στο top module, χωρίς όμως να άλλαζε κάτι στη λειτουργία του τελικού μας κυκλώματος.

Περιγράφοντας τώρα τη λειτουργία του CLA, τα CarryGP (Παράρτημα 5.2) είναι υπεύθυνα για τον έλεγχο (και παραγωγή των αντίστοιχων σημάτων) δημιουργίας κρατουμένου στο εκάστοτε bit ή διάδοσης κρατουμένου που δημιουργήθηκε σε προηγούμενο bit. Με τη σειρά του, το CarryLA (Παράρτημα 5.3), έχοντας ως είσοδο τα generate και propagate σήματα των CarryGP και του κρατούμενου εισόδου, υπολογίζει ταυτόχρονα τα κρατούμενα της πρόσθεσης σε κάθε bit. Με βάση τα κρατούμενα και τους τελεστέους, γίνεται ο τελικός υπολογισμός του αθροίσματος σε κάθε bit από τις μονάδες Sum (Παράρτημα 5.4).

Ως υποσημείωση να τονίσουμε ότι για επιπλέον εξάσκηση στη VHDL, υλοποιήσαμε τις υπομονάδες του CLA τόσο με behavioural όσο και με structural αρχιτεκτονική. Για χάριν συντομίας, εδώ παραθέτουμε μόνο το CarryLA unit με ιεραρχική περιγραφή (Παράρτημα 5.5), όπου χρησιμοποιήθηκαν πύλες AND και OR πολλαπλών εισόδων, οι οποίες και αυτές με τη σειρά τους είχαν υλοποιηθεί ιεραρχικά από πύλες δύο εισόδων χρησιμοποιώντας δενδρική τοπολογία (tree topology).

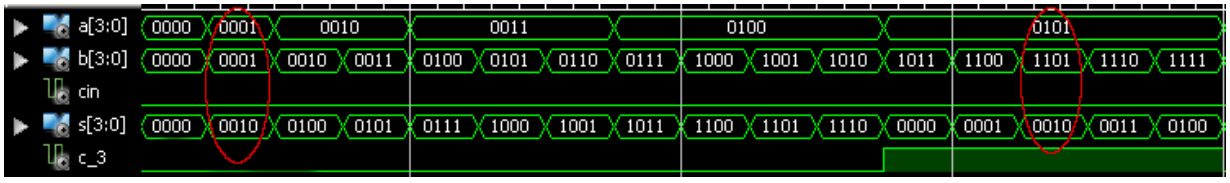
2.2 FSM

Στο Παράρτημα 5.6 παραθέτουμε τον behavioral κώδικα της ζητούμενης Moore FSM τριών καταστάσεων. Στο case statement φαίνονται ποιες είναι οι μεταβάσεις από το τωρινό (currentS) στο επόμενο (nextS) state, ανάλογα με το ποιο button από τα τρία της εισόδου είναι πατημένο (σήμα στο 1). Σε κάθε when επιπροσθέτως ορίζουμε και την έξοδο που στη συγκεκριμένη άσκηση αντιστοιχεί στα 8 leds του basys 2. Τέλος, ορίζουμε ότι η FSM εκκινεί από την κατάσταση A (κάτι που εξασφαλίζει και το when others), ενώ σε περίπτωση μη πατήματος κουμπιού (όπως και για το κουμπί 2) θεωρούμε ότι δεν αλλάζει κατάσταση.

Όπως ζητήθηκε κατά την εκτέλεση της άσκησης, αυτή η FSM έγινε instantiate και συνδέθηκε σε top module (Παράρτημα 5.7) με μία γεννήτρια παλμών διάρκειας ενός κύκλου (στην ουσία και αυτή μία FSM), η οποία και μεσολαβούσε μεταξύ εισόδων (κουμπιών) και FSM για την αποφυγή εισαγωγής σήματος διάρκειας πολλών κύκλων στην FSM (λόγω φυσικών περιορισμών του πατήματος κουμπιών απέναντι στη μεγάλη συχνότητα ρολογιού).

3 Κυματομορφές-Προσομοίωση

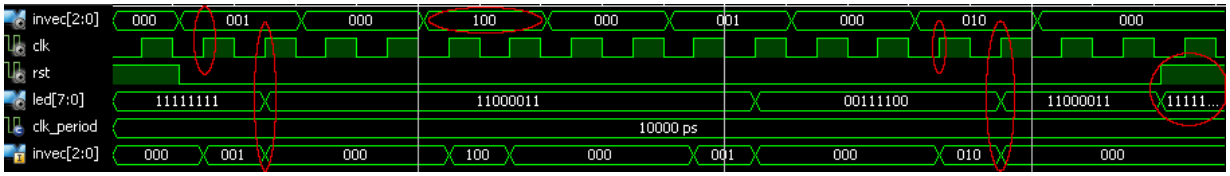
Μετά την υλοποίηση των παραπάνω κυκλωμάτων, προχωρήσαμε στη δημιουργία κώδικα test bench για τον έλεγχο της ορθής λειτουργίας τους μέσω προσομοίωσης. Και στις δύο περιπτώσεις η προσομοίωση παρήγαγε τα αναμενόμενα αποτελέσματα.



Σχήμα 1: Τμήμα του διαγράμματος χρονισμού προσομοίωσης της λειτουργίας του CLA για όλες τις πιθανές περιπτώσεις.

Ειδικότερα, στο Σχήμα 1 παραθέτουμε την κυματομορφή της προσομοίωσης του CLA. Στη συγκεκριμένη προσομοίωση καλύψαμε όλες τις δυνατές περιπτώσεις πρόσθεσης με ένα for loop, όπως φαίνεται και από το αντίστοιχο process του test bench (Παράρτημα 5.8). Στα κυκλωμένα παραδείγματα (από δεξιά προς τα αριστερά) φαίνονται τα αποτελέσματα των προσθέσεων 0001+0001 = 0010 με μηδενικό κρατούμενο εξόδου και 0101+1101=0010 με κρατούμενο εξόδου 1.

Στο Σχήμα 2 παρουσιάζουμε αντίστοιχα τα αποτελέσματα της προσομοίωσης της FSM. Περιγράφοντας τα κυκλωμένα σημεία με χρονική σειρά, βλέπουμε ότι το πάτημα του πρώτου button δημιουργεί ένα παλμό εισόδου (τελευταίο διάνυσμα invec) στην επόμενη θετική ακμή του ρολογιού (και όχι νωρίτερα). Με τη σειρά του ο παλμός εισόδου διαρκεί αναμενόμενα ένα κύκλο ρολογιού και οδηγεί σε αλλαγή κατάστασης (από A σε B, με τα μεσαία leds να σβήνουν) στην αμέσως επόμενη θετική ακμή. Παρόμοια λειτουργούν και τα υπόλοιπα κουμπιά, με το τρίτο να μην επηρεάζει την κατάσταση της FSM και το δεύτερο να κάνει τη μεταβολή από C σε B (όπου αλλάζουν όλα τα leds). Τέλος βλέπουμε και τη λειτουργία του Reset που οδηγεί στην κατάσταση A ανεξαρτήτως κατάστασης ή άλλων εισόδων.



Σχήμα 2: Διάγραμμα χρονισμού προσομοίωσης της λειτουργίας της FSM για τις περιπτώσεις πατήματος του Button 1 (πρώτο σήμα InVec(0)), Button 3 (InVec(2)), Button 2 (InVec(1)), Reset (κυκλωμένες από αριστερά προς δεξιά).

4 Συμπεράσματα

Στηριζόμενη ξανά σε απλά παραδείγματα κυκλωμάτων, η δεύτερη εργαστηριακή άσκηση μας προσέφερε την ευκαιρία να εξασκηθούμε στα εξής: 1) Στη χρήση ιεραρχικής σχεδίασης με top down ανάλυση λειτουργίας και bottom up υλοποίηση, όπου μικρότερες μονάδες δομούνται σε κατώτερο ιεραρχικά επίπεδο και

ενώνονται σε ανώτερο για να συνθέσουν ένα μεγαλύτερο κύκλωμα. 2) Στη χρήση τόσο behavioral όσο και structural αρχιτεκτονικών. 3) Στην υλοποίηση σύγχρονων ακολουθιακών κυκλωμάτων (με ρολόι), και συγκεκριμένα FSM με συμπεριφορικό μοντέλο. 4) Στην περαιτέρω εξοικείωση με τη σύνταξη case statements και processes στη VHDL.

5 Παράρτημα - Κώδικας VHDL

5.1 4-bit CLA

```
entity CLA4 is
    Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
           B : in  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           S : out  STD_LOGIC_VECTOR (3 downto 0);
           C_3 : out  STD_LOGIC);
end CLA4;
architecture Structural of CLA4 is
        component CarryGP is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           P : out  STD_LOGIC;
           G : out  STD_LOGIC);
        end component;
        component Sum is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           S : out  STD_LOGIC);
        end component;
        component CarryLA is
    Port ( C : out  STD_LOGIC_VECTOR (3 downto 0);
           Cin : in  STD_LOGIC;
           P : in  STD_LOGIC_VECTOR (3 downto 0);
           G : in  STD_LOGIC_VECTOR (3 downto 0));
        end component;
        signal P, G        : STD_LOGIC_VECTOR (3 downto 0);
        signal C               : STD_LOGIC_VECTOR (2 downto 0);
        begin
                CGPULoop : For j in 0 to 3 generate
                        CGPU: CarryGP Port Map (A => A(j),
                                                B => B(j),
                                                P => P(j),
                                                G => G(j));
                end generate;
                CLAU: CarryLA Port Map         (Cin => Cin,
                                                P => P,
                                                G => G,
                                                C(0) => C(0),
                                                C(1) => C(1),
                                                C(2) => C(2),
                                                C(3) => C_3);
                SumULoop : For j in 1 to 3 generate
                        SumU: Sum Port Map     (A => A(j),
                                                B => B(j),
                                                C => C(j-1),
                                                S => S(j));
                end generate;
                SumU0: Sum Port Map                    (A => A(0),
                                                B => B(0),
                                                C => Cin,
                                                S => S(0));
end Structural;
```

5.2 Carry Generate - Propagate Unit

```
architecture Behavioral of CarryGP is
        begin
                P <= A xor B;
                G <= A and B;
end Behavioral;
```

5.3 Carry Look Ahead Unit - Behavioral Example

```
architecture Behavioral of CarryLA is
        begin
                C(0) <= G(0) or ( P(0) and Cin );
                C(1) <= G(1) or (P(1) and G(0)) or ( P(1) and P(0) and Cin);
                C(2) <= G(2) or (P(2) and G(1)) or ( P(2) and P(1) and G(0)) or
                        ( P(2) and P(1) and P(0) and Cin);
                C(3) <= G(3) or (P(3) and G(2)) or ( P(3) and P(2) and G(1)) or
                        (P(3) and P(2) and P(1) and G(0)) or
                        ( P(3) and P(2) and P(1) and P(0) and Cin);
end Behavioral;
```

5.4 Sum Unit

```
architecture Behavioral of Sum is
        begin
                S <= A xor B xor C;
end Behavioral;
```

5.5 Carry Look Ahead Unit - Structural Example (Bonus)

```
architecture Structural of CarryLA is
        component OrGate is
                 Port ( In0 : in  STD_LOGIC;
                                  In1 : in  STD_LOGIC;
                                  Out0 : out  STD_LOGIC);
        end component;
        component AndGate is
    Port ( In0 : in  STD_LOGIC;
           In1 : in  STD_LOGIC;
           Out0 : out  STD_LOGIC);
        end component;
        component Or3Gate is
                 Port ( In0 : in  STD_LOGIC;
                                  In1 : in  STD_LOGIC;
                                  In2 : in  STD_LOGIC;
                                  Out0 : out  STD_LOGIC);
        end component;
        component And3Gate is
    Port ( In0 : in  STD_LOGIC;
           In1 : in  STD_LOGIC;
                        In2 : in  STD_LOGIC;
           Out0 : out  STD_LOGIC);
        end component;
        component Or4Gate is
                 Port ( InVec : in  STD_LOGIC_VECTOR (3 downto 0);
                                  Out0 : out  STD_LOGIC);
        end component;
        component And4Gate is
    Port ( InVec : in  STD_LOGIC_VECTOR (3 downto 0);
           Out0 : out  STD_LOGIC);
        end component;
        component Or5Gate is
                 Port ( InVec : in  STD_LOGIC_VECTOR (4 downto 0);
                                  Out0 : out  STD_LOGIC);
        end component;
        component And5Gate is
    Port ( InVec : in  STD_LOGIC_VECTOR (4 downto 0);
           Out0 : out  STD_LOGIC);
        end component;
        signal Ponce : STD_LOGIC_VECTOR (3 downto 0);
        signal Ptwice : STD_LOGIC_VECTOR (2 downto 0);
        signal Pthree : STD_LOGIC_VECTOR (1 downto 0);
        signal Pfour : STD_LOGIC;
        begin
        -- All 2-input AND gates
        AndULoop : For j in 1 to 3 generate
                AndU1: AndGate Port Map (        In0 => P(j),
                                                 In1 => G(j-1),
                                                 Out0 => Ponce(j)); -- Propagate once
        end generate;
        AndU0: AndGate Port Map (        In0 => P(0),
```

```
                                                In1 => Cin,
                                                Out0 => Ponce(0));
        -- All 3-input AND gates
        And3ULoop : For j in 2 to 3 generate
                And3U1: And3Gate Port Map (        In0 => P(j),
                                                        In1 => P(j-1),
                                                        In2 => G(j-2),
                                                        Out0 => Ptwice(j-1));
        end generate;
        And3U0: And3Gate Port Map (        In0 => P(1),
                                                In1 => P(0),
                                                In2 => Cin,
                                                Out0 => Ptwice(0));
        -- All 4-input AND gates
        And4U1: And4Gate Port Map (        InVec(0 to 2) => P(1 to 3),
                                                InVec(3) => G(0),
                                                Out0 => Pthree(1));
        And4U0: And4Gate Port Map (        InVec(0 to 2) => P(0 to 2),
                                                InVec(3) => Cin,
                                                Out0 => Pthree(0));
        -- 5-input AND gate
        And5U: And5Gate Port Map (        InVec(0 to 3) => P,
                                        InVec(4) => Cin,
                                        Out0 => Pfour);
        -- C_0 calculation
        OrU:        OrGate         Port Map (In0 => Ponce(0),
                                                In1 => G(0),
                                                Out0 => C(0));
        -- C_1 calculation
        Or3U:        Or3Gate         Port Map (In0 => Ponce(1),
                                        In1 => Ptwice(0),
                                        In2 => G(1),
                                        Out0 => C(1));
        -- C_2 calculation
        Or4U: Or4Gate Port Map (        InVec(0) => Ponce(2),
                                                InVec(1) => Ptwice(1),
                                                InVec(2) => Pthree(0),
                                                InVec(3) => G(2),
                                                Out0 => C(2));
        -- C_3 calculation
        Or5U: Or5Gate Port Map (        InVec(0) => Ponce(3),
                                                InVec(1) => Ptwice(2),
                                                InVec(2) => Pthree(1),
                                                InVec(3) => Pfour,
                                                InVec(4) => G(3),
                                                Out0 => C(3));
end Structural;
```

5.6 FSM

```
architecture Behavioral of Lab2FSM is
        Type state is (A, B, C);
        signal currentS, nextS: state;
        begin
                fsm_combi: process (currentS, InVec)
                  begin
                        case currentS is
                                when A =>        Out0 <=         "11111111";
                                                                if InVec = "001"
                                                                        then nextS <= B;
                                                                elsif InVec = "010"
                                                                        then nextS <= C;
                                                                else          nextS <= A;
                                                                end if;

                                when B =>        Out0 <=         "11000011";
                                                                if InVec = "001"
                                                                        then nextS <= C;
                                                                elsif InVec = "010"
                                                                        then nextS <= A;
                                                                else          nextS <= B;
                                                                end if;

                                when C =>        Out0 <=         "00111100";
```

```
                                                        if InVec = "001"
                                                                then nextS <= A;
                                                        elsif InVec = "010"
                                                                then nextS <= B;
                                                        else            nextS <= C;
                                                        end if;
                                when others =>          Out0 <=         "11111111";
                                                                nextS <= A;
                        end case;
                end process;
                fsm_synchr: process (Clk, Rst)
                        begin
                                if (Rst = '1')
                                        then currentS <= A;
                                elsif (rising_edge(Clk))
                                        then currentS <= nextS;
                                end if;
                end process;
end Behavioral;
```

5.7 FSMtop

```
architecture Structural of Lab2FSMTop is
        component singlepulsegen is
                Port (          clk                     : in std_logic;
                                        rst                     : in std_logic;
                                        input           : in std_logic;
                                        output           : out std_logic);
        end component;
        component Lab2FSM is
                Port        (        InVec : in  STD_LOGIC_VECTOR (2 downto 0);
                                        Clk : in  STD_LOGIC;
                                        Rst : in  STD_LOGIC;
                                        Out0 : out  STD_LOGIC_VECTOR (7 downto 0));
        end component;
        signal        PulseVec : STD_LOGIC_VECTOR (2 downto 0);
        begin
                PulseLoop : For i in 0 to 2 generate
                        PulseGen : singlepulsegen  Port Map (   rst => Rst,
                                                                input => InVec(i),
                                                                clk => Clk,
                                                                output => PulseVec(i));
                end generate;
                FSMU : Lab2FSM                Port Map (        InVec => PulseVec,
                                                                Clk        => Clk,
                                                                Rst        => Rst,
                                                                Out0        => Led);
end Structural;
```

5.8 CLA test bench

```
   stim_proc: process
   begin
      wait for 50 ns;
                for i in 0 to 15 loop -- loop that checks all possible input values
                        A <= STD_LOGIC_VECTOR (unsigned(A) + 1);
                        for j in 0 to i loop
                                B <= STD_LOGIC_VECTOR (unsigned(B) + 1);
                                wait for 50 ns;
                        end loop;
                end loop;
      wait;
   end process;
```