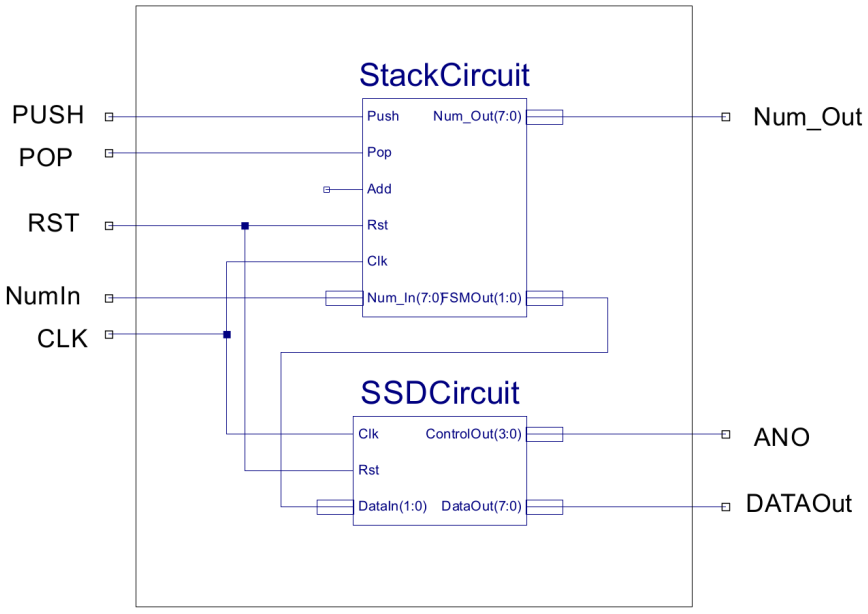
ΗΡΥ203 Προχωρημένη Λογική Σχεδίαση
3η Εργαστηριακή Άσκηση

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

Ημερομηνία Διεξαγωγής: 31 Μαρτίου 2017

1 Σκοπός Εργαστηριακής Άσκησης

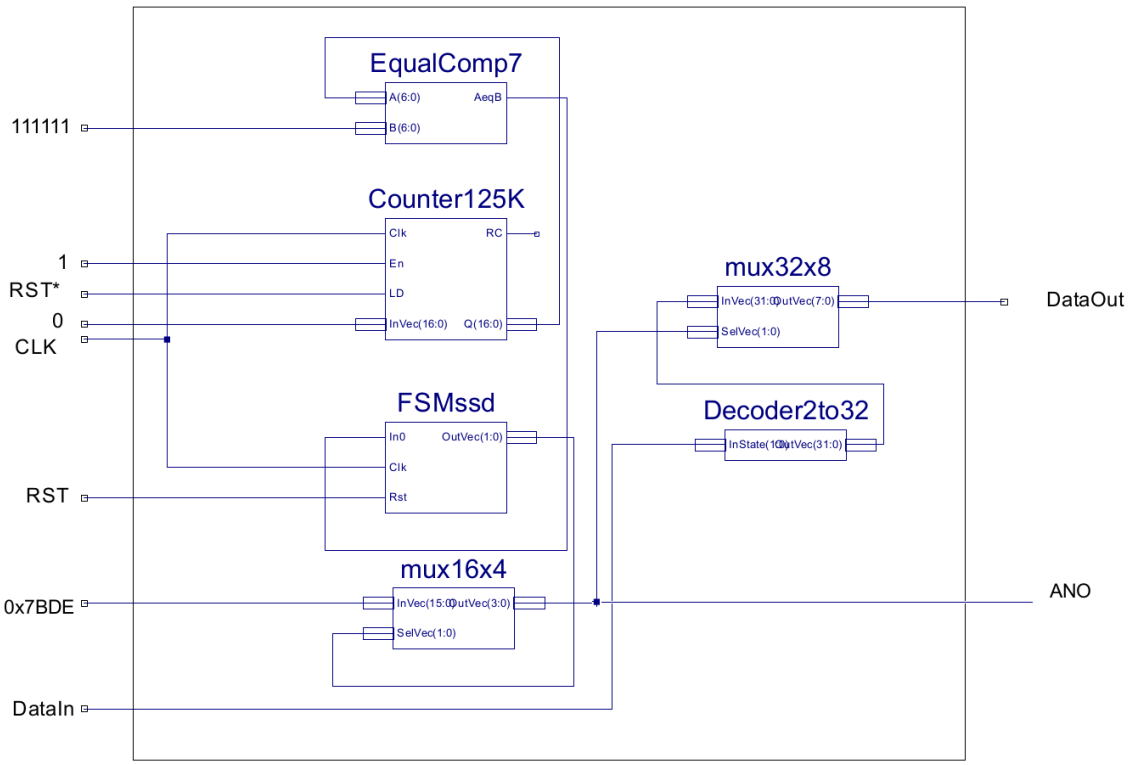
Με την τρίτη εργαστηριακή άσκηση ξεκινά ο δεύτερος κύκλος εργαστηριακών ασκήσεων του μαθήματος, όπου στόχος είναι η σταδιακή σχεδίαση μίας απλής αριθμομηχανής. Στο παρόν μέρος της άσκησης πραγματοποιείται το πρώτο βήμα υλοποίησής της, με το σχεδιασμό σε VHDL τόσο της βασικής λειτουργικότητας μίας post-increment, pre-decrement στοίβας ως μνήμη, όσο και της βασικής διεπαφής με τα Seven Segment Displays (SSD) του Basys 2. Για την κατασκευή τους καλούμαστε να χρησιμοποιήσουμε με ιεραρχικό τρόπο δομικές μονάδες, ορισμένες με τη σειρά τους με όλους τους τρόπους που μάθαμε στα προηγούμενα εργαστήρια (με behavioural και structural design). Συνεπώς, σε αυτό το εργαστήριο επιτυγχάνεται τόσο η βαθύτερη κατανόηση της ιεραρχικής σχεδίασης σε VHDL, όσο και της χρήσης κυκλωμάτων stack memory και SSD.



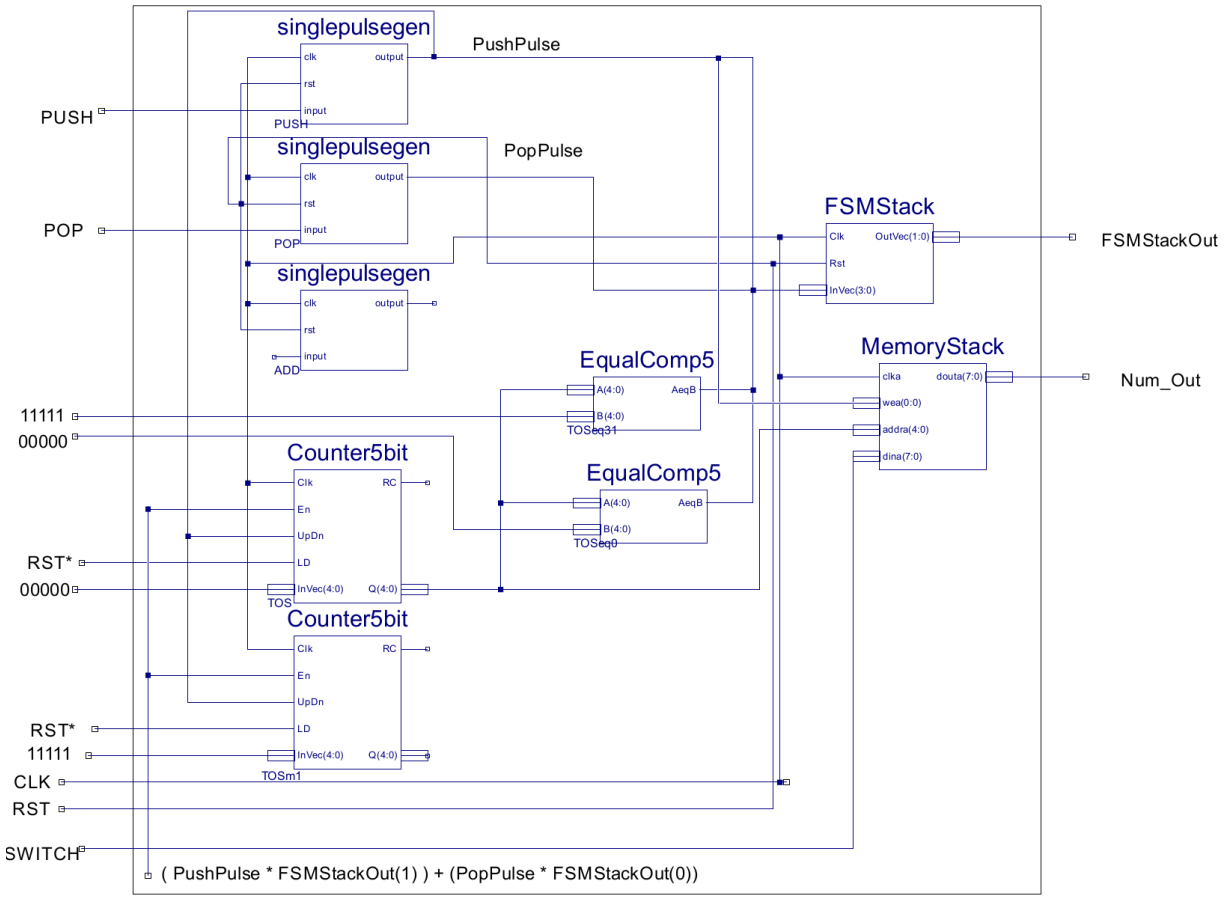
Σχήμα 1: Το πλήρες κύκλωμα στο ανώτατο ιεραρχικό επίπεδο. Το Σχήμα 2 απεικονίζει σε κατώτερο επίπεδο το SSDCircuit, ενώ το Σχήμα 3 απεικονίζει το StackCircuit.

2 Προεργασία-Περιγραφή

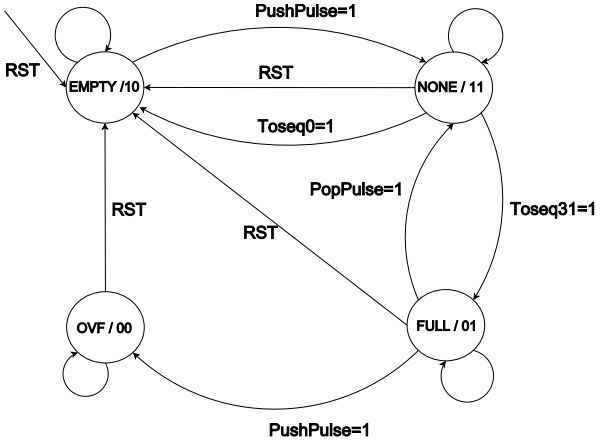
Στα ζητούμενα της προεργασίας, εκτός από τον κώδικα VHDL, περιλαμβάνονταν το block diagram της δική μας σχεδίασης, τα διαγράμματα καταστάσεων των FSM που χρησιμοποιήθηκαν, και το σχήμα



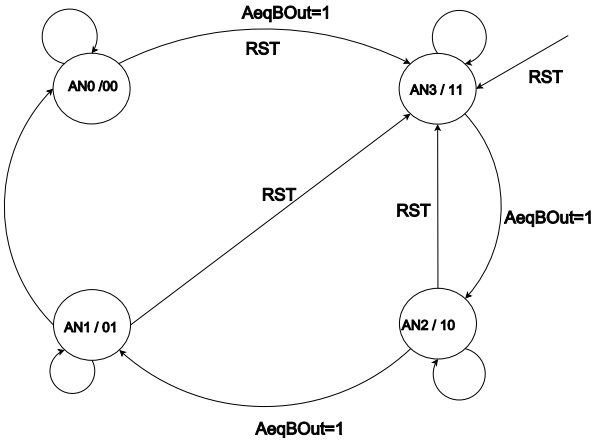
Σχήμα 2: Το κύκλωμα επικοινωνίας (SSDCircuit) της στοίβας με τα SSD. Η αριστερή στήλη περιέχει το control path ενώ η δεξιά το data path.



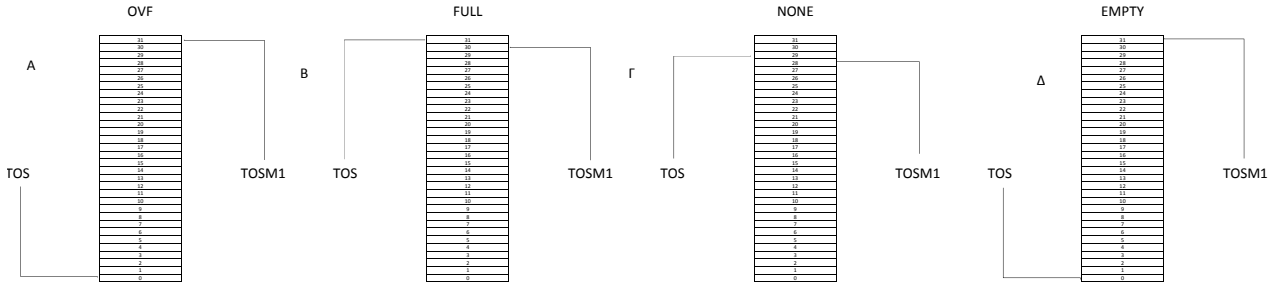
Σχήμα 3: Το κύκλωμα (StackCircuit) υλοποίησης της στοίβας. Στο κύκλωμα περιλαμβάνονται και τα κυκλώματα παλμών εισόδου (singlepulsegen) και η block RAM (MemoryStack). Το διάγραμμα καταστάσεων της FSMStack δίνεται στο Σχήμα 4.



Σχήμα 4: Η μηχανή πεπερασμένων καταστάσεων που ελέγχει την στοίβα. Όλες οι περιπτώσεις εισόδου που δεν καταγράφονται ρητώς οδηγούν σε διατήρηση της κατάστασης. Το πρώτο bit εξόδου της FSM επιτρέπει ή όχι το PUSH, και αντίστοιχα το δεύτερο bit το POP.



Σχήμα 5: Η μηχανή πεπερασμένων καταστάσεων που ελέγχει την απεικόνιση στα SSD. Όλες οι περιπτώσεις εισόδου που δεν καταγράφονται ρητώς οδηγούν σε διατήρηση της κατάστασης.



Σχήμα 6: Η μνήμη για την υλοποίηση της ουράς. Απεικονίζονται οι θέσεις των Top Of Stack και Top Of Stack -1 στην κατάσταση OVF (A), FULL (B), NONE (Γ, παράδειγμα για 29 αποθηκευμένους ακεραίους), και EMPTY (Δ).

της stack memory. Στη συνέχεια αυτής της ενότητας προχωράμε σε μία top-down παρουσίαση του κυκλώματός μας.

Ξεκινώντας με το Σχήμα 1, παρουσιάζουμε το ανώτατο ιεραρχικά επίπεδο της σχεδίασής μας. Σε αυτό φαίνεται ο σαφής διαχωρισμός του κυκλώματός μας σε δύο μέρη, σε άμεση αντιστοίχιση των μερών που περιγράψαμε στην εισαγωγή: 1) Στο StackCircuit (διατηρώντας τα ονόματα που χρησιμοποιήσαμε στη VHDL) υλοποιείται όλη η λειτουργικότητα της στοίβας. Όλες οι είσοδοι συνδέονται με αυτό το module, ενώ η έξοδος Num_Out της μνήμης συνδέεται απευθείας με τα LEDs. 2) Το SSDCircuit module μεσολαβεί μεταξύ του StackCircuit και των SSDs, χρησιμοποιώντας δηλαδή την έξοδο της FSM που ελέγχει την κατάσταση της στοίβας για να ελέγξει την πληροφορία που θα απεικονιστεί στα SSDs.

2.1 Stack module

Στο Σχήμα 3 παρουσιάζουμε αναλυτικότερα το κύκλωμα που υλοποιεί τη στοίβα. Τα κουμπιά εισόδου (PUSH, POP) συνδέονται με κυκλώματα singlepulsegen, τα οποία μας δόθηκαν στο προηγούμενο εργαστήριο και είναι υπεύθυνα για την μετατροπή των αντίστοιχων σημάτων εισόδου σε παλμούς (PushPulse, PopPulse) διάρκειας ενός κύκλου ρολογιού. Το υποκύκλωμα περιέχει δύο 5-bit μετρητές (συμπεριφορική υλοποίηση, Παράρτημα 5.1), έναν που δείχνει στο top of stack (TOS) και έναν που δείχνει στην αμέσως προηγούμενη θέση (top of stack -1, TOSm1). Επιπλέον χρησιμοποιήσαμε το εργαλείο Core Generator για τη δημιουργία block memory πλάτους 8-bit και βάθους 32 θέσεων (Σχήμα 6), ώστε συνολικά να αποθηκεύονται έως 31 ακέραιοι.

Για τον έλεγχο των ζητούμενων λειτουργιών της στοίβας χρησιμοποιήσαμε μια FSM τεσσάρων καταστάσεων που αντιστοιχούν και στις τέσσερις αντίστοιχες απεικονίσεις των SSDs (Σχήμα 4). Οι μεταβάσεις ελέγχονται από τα σήματα PushPulse, PopPulse και από την έξοδο συγκριτών που ελέγχουν το πότε ο TOS θα φτάσει τις τιμές 0 ή 31. Οι μετρητές, με τη σειρά τους, μετρούν ένα βήμα κατά πάνω ή κατά κάτω σύμφωνα με τα PushPulse, PopPulse αντίστοιχα, αλλά μόνο όταν η 2-bit έξοδος της FSM το επιτρέπει. Τελικά, η block memory δίνει την έξοδό του στα LEDs, ενώ η FSM με την έξοδό της ελέγχει επίσης το τι θα δοθεί στο υποκύκλωμα των SSDs για απεικόνιση.

2.2 SSD module

Στο Σχήμα 2 απεικονίζεται το SSDCircuit module. Το υποκύκλωμα αυτό αποτελείται από δύο modules που αντιστοιχούν στο control και data path.

Το control path περιέχει ένα μετρητή, ο οποίος μετράει έως το 125000. Επιπλέον περιέχει μια FSM (Σχήμα 5) που καθορίζει (μέσω ενός πολυπλέκτη 16x4) το SSD που θα λειτουργεί κάθε στιγμή. Ο έλεγχος της FSM από τον παραπάνω μετρητή επιτρέπει την απεικόνιση συχνότητας 100 Hz (με βάση το ρολόι του Basys 2, 50 MHz).

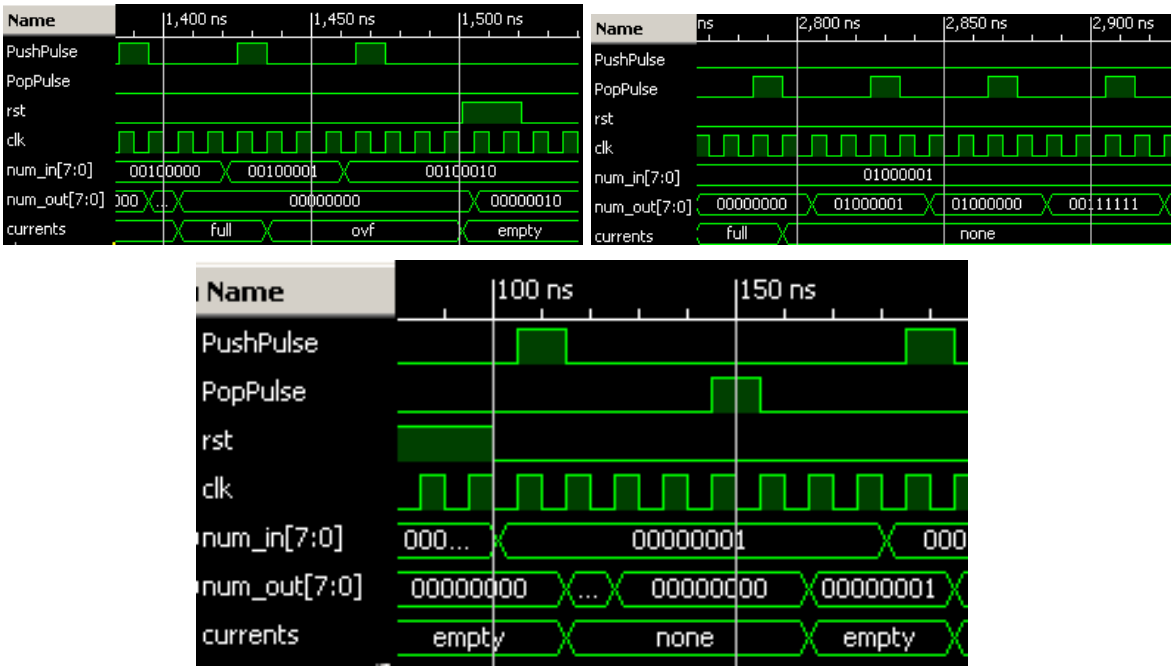
Από την άλλη πλευρά το data path περιέχει έναν αποκωδικοποιητή 2 to 32 (συμπεριφορική σχεδίαση, Παράρτημα 5.2), ο οποίος σύμφωνα με την έξοδο της FSMstack παράγει το αντίστοιχο σήμα της κοινής καθόδου. Μιας και η κάθοδος για τα SSDs είναι κοινή, στην έξοδο του αποκωδικοποιητή χρειάζεται ένας πολυπλέκτης 32x8 (υλοποιήθηκε ιεραρχικά από πολυπλέκτες 2x1, παράδειγμα Παράρτημα 5.3), ο οποίος και ελέγχεται από το control path.

3 Κυματομορφές-Προσομοίωση

Μετά την υλοποίηση των παραπάνω κυκλωμάτων, προχωρήσαμε στη δημιουργία κώδικα test bench για τον έλεγχο της ορθής λειτουργίας τόσο εξ΄ ολοκλήρου, όσο και τμήματος του κυκλώματος. Για οικονομία χώρου, στην παρούσα αναφορά παρουσιάζουμε τρεις περιπτώσεις κυματομορφών για τις καταστάσεις empty, full, ovf.

Ειδικότερα, στο Σχήμα 7-κάτω βλέπουμε την αρχική συμπεριφορά του κυκλώματος μετά το πάτημα του reset. Η στοίβα ξεκινά στην κατάσταση empty, με τον ερχομό παλμού push περνά στην κατάσταση none, ενώ με ένα pop ξαναεπαναφέρεται σε empty και εμφανίζεται στα LEDs η τιμή που δόθηκε στην πρώτη θέση της μνήμης.

Στο Σχήμα 7-πάνω-αριστερά παρουσιάζεται η περίπτωση της υπερχείλισης. Τη στιγμή που δίνουμε το 32ο push, η τιμή 33 δεν εγγράφεται στη μνήμη, και το κύκλωμα πλέον δεν αντιδρά (κατάσταση ovf) σε



Σχήμα 7: Τμήματα του διαγράμματος χρονισμού προσομοίωσης της λειτουργίας του κυκλώματος. Πάνω αριστερά: Γέμισμα μνήμης και υπερχείλιση. Πάνω δεξιά: Γέμισμα μνήμης και ανάκτηση από τη μνήμη. Κάτω: Έναρξη άδειας μνήμης.

οτιδήποτε μέχρι να δοθεί reset και η μνήμη να αδειάσει (στην πράξη δεν διαγράφουμε τις προηγούμενες τιμές, αλλά τις πανωγράφουμε στη συνέχεια). Τέλος, στο Σχήμα 7-πάνω-δεξιά δείχνουμε την περίπτωση διαβάσματος τιμών από την μνήμη, έχοντας φτάσει στην κατάσταση full.

4 Συμπεράσματα

Η τρίτη εργαστηριακή άσκηση αποτέλεσε το πρώτο βήμα για τη δημιουργία μιας απλής αριθμομηχανής. Πιο συγκεκριμένα: 1) εξασκηθήκαμε στη δημιουργία κυκλώματος ελέγχου της λειτουργίας SSDs και 2) στην ιεραρχική υλοποίηση μιας στοίβας. Αν και το κύκλωμά μας είναι σχεδιαστικά σωστό και λειτούργησε σωστά στην FPGA, θα μπορούσαμε να μειώσουμε τις καταστάσεις της FSMstack από τέσσερις σε δύο (για push, pop, φυσικά με επιπλέον module με λογική για τις λειτουργίες empty, full, ovf). Επιπλέον, θα μπορούσαμε να κρατάμε στην κατάσταση empty τον TOSm1 στη θέση 0, και να ελέγχουμε ξεχωριστά από τα seven segments το DP, αλλά για την παρούσα άσκηση αυτές οι αλλαγές δεν θα είχαν κάποια επίδραση.

5 Παράρτημα - Κώδικας VHDL

5.1 5-bit Counter

```
architecture Behavioral of Counter5bit is
signal count : STD_LOGIC_VECTOR (4 downto 0);
begin
process (Clk, En, LD, UpDn, InVec)
        begin
                if LD = '0' then count <= InVec; -- load
                elsif En = '1' then -- check enable
                        if rising_edge(Clk) then
                                if UpDn = '1' then count <=
                                  std_logic_vector (unsigned(count) + 1); -- increment register
                                else count <=
                                  std_logic_vector (unsigned(count) - 1); -- decrement register
                                end if;
                        end if;
                end if;
end process;

process (count, UpDn)
        begin
                Q <= count;
```

```
                if ( (count = "11111") and (UpDn = '1') )
                  or ( (count = "00000") and (UpDn = '0') ) then RC <= '1';
                else RC <= '0';
                end if;
end process;
end Behavioral;
```

5.2 Decoder2to32

```
architecture Behavioral of Decoder2to32 is
begin
        with InState select
                OutVec <= X"FF038371" when "00", -- printing OVF: F = X"71", V = X"83", O = X"03"
                          X"FFFFFF71" when "01", -- printing F
                          X"FFFFFF61" when "10", -- printing E: E = X"61"
                          X"FFFFFFFF" when others; -- printing NONE
end Behavioral;
```

5.3 Mux32x8

```
architecture Structural of mux32x8 is
        component mux4x1  is
                Port (        InVec : in  STD_LOGIC_VECTOR (3 downto 0);
                              SelVec : in  STD_LOGIC_VECTOR (1 downto 0);
                              Out0 : out  STD_LOGIC);
        end component;
        begin
                Loop1 : For i in 0 to 7 generate
                        mux0 : mux4x1 Port Map (InVec(0) => InVec(i),
                                                InVec(1) => InVec(i+8),
                                                InVec(2) => InVec(i+16),
                                                InVec(3) => InVec(i+24),
                                                SelVec => SelVec,
                                                Out0 => OutVec(i));
                end generate;
end Structural;
```