Εργαστήριο Ψηφιακών Συστημάτων
Εργασία 1
7-Segment Display Driver

Ανδρικόπουλος Κωνσταντίνος

Περίληψη

Η εργασία έχει ως σκοπό την υλοποίηση ενός 7-segment LED Display driver. Χωρίζεται σε 4 μέρη.
Στο **Α μέρος** υλοποιείται ο κωδικοποιητής για τους χαρακτήρες που θα φανούν στην οθόνη.
Στο **Β μέρος** υλοποιείται το αργό ρολόι με το MMCME module, οδηγούνται οι άνοδοι κατάλληλα βάσει μετρητή και ένα σταθερό μήνυμα φαίνεται στο display.
Στο **Γ μέρος** υλοποείται βηματική μετάβαση του μηνύματος με την χρήση κουμπιού.
Στο **Δ μέρος** υλοποιείται βηματική μετάβαση του μηνήματος, αλλά τώρα με σταθερή καθυστέρηση 1,6777214 δευτερόλεπτα.

Μέρος Α: Κωδικοποιητής 7 τμημάτων

Στο πρώτο μέρος υλοποιήθηκε ο κωδικοποιητής που κωδικοποιεί 5-bit εισόδους (τους χαρακτήρες/ψηφία που θέλουμε να αναπαρασταθούν στα led) σε 7-bit εξόδους που οδηγούν τα τμήματα κάθε led segment. Η κωδικοποίησή μου περιλαμβάνει όλους τους αριθμούς και την αγγλική αλφαβήτα μέχρι το V (5'b00000-5'b11111). Αντιστοίχισα τα 7 bits του LED στα segments a-g από αριστερά προς τα δεξιά. Αφού τα segments είναι ενεργά όταν είναι στο 0 τα διάνυσματα LED έχουν φτιαχτεί ανάλογα. Για παράδειγμα, για να ανάψει το 0 σε ένα ψηφίο της οθόνης θα πρέπει να ανάψουν όλα τα segments εκτός από το g, δηλαδή το LED θα είναι 1000000.

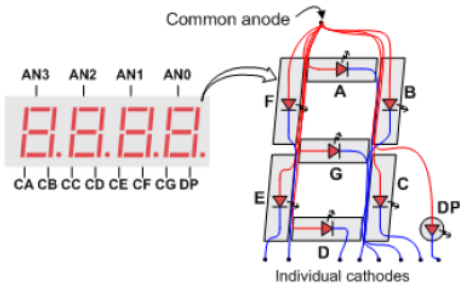

Figure 1: 7-segment display

Το κύκλωμα του κωδικοποιητή είναι συνδυαστικό και αποτελείται μόνο από μία case με όλες τις περιπτώσεις των 5-bit char.

Μέρος Β: Οδήγηση των ψηφίων

Το δεύτερο μέρος της εργασίας περιλαμβάνει την οδήγηση των 8 ανόδων και τον πρώτο πειραματισμό της πλακέτας. Παρακάτω φαίνεται ένα σχεδιάγραμμα για αυτό το μέρος:

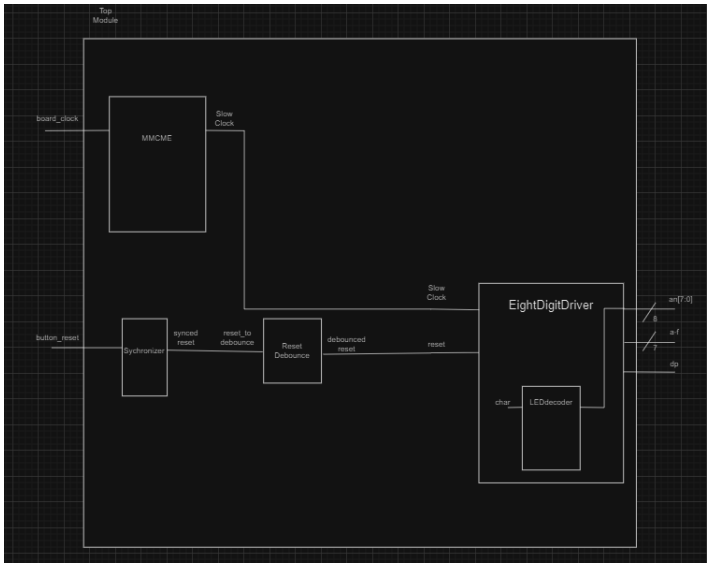

Figure 2: Σχεδιάγραμμα Β μέρους

Ανάγκη για αργό clock και MMCME module

Το ρόλοι 100MHz της πλακέτας είναι πολύ γρήγορο για το κύκλωμα των led λόγω της υψηλής χωρητικότητας τους. Για αυτό χρησιμοποίησα το mmcme module για να αυξήσω την περίοδο του ρολογιού από 10ns σε 0.2μs (200ns).
Στο module το CLKIN1 port συνδέθηκε με το clk (το ρολόι της πλακέτας) και το αργό ρολόι (μεταβλητή clock) που χρησιμοποιείται στην υπόλοιπη εργασία οδηγείται από το CLKOUT1. Συνδέθηκε κατάλληλα το feedback loop ανάμεσα σε CLKFBOUT και CLKFBIN.
Για να οριστεί η περίοδος του καινούριου ρολογιού αρχικά θέτουμε το port CLKIN1_PERIOD σε 10.0ns όσο και η περίοδος του ρολογιού και χρησιμοποιούμε τις παρακάτω εξισώσεις.

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D} \qquad \text{Equation 3-1}$$

$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O} \qquad \text{Equation 3-2}$$

Figure 3: MMCME equations

M είναι η τιμή του port CLKFBOUT_MULT_F
D είναι η τιμή του port DIVCLK_DIVIDE
O είναι η τιμή του port CLKOUT1_DIVIDE

Επίσης, η τιμή της συχνότητας V_{co} πρέπει να είναι ανάμεσα σε 600-1600MHz.
Τελικά για να παράξω το αργό ρολόι 5MHz που θέλουμε έβαλα αντίστοιχα τις τιμές M=6.0, D=1 και O=120.

Συγχρονισμός του reset

Το ασύγχρονο reset μπορεί να προκαλέσει προβλήματα μεταστάθειας και αξιοπιστίας των δεδομένων. Για αυτό χρησιμοποιώ δύο flip flop για να αποφευχθούν τέτοια προβλήματα. Το πρώτο flip flop έχει σκόπο την δειγματοληψία του σήματος και το δεύτερο βεβαιώνει ότι δεν θα υπάρχει μεταστάθεια.

Στον κώδικα αυτό πραγματοποιείται από το παρακάτω always block:

```
/*an always that sychronizes the reset from the reset button through two flip flops
to avoid possible metastabiliy*/
always@(posedge clock)
    begin
        reset_mid_ff<=reset;
        sync_out<=reset_mid_ff;
    end
```

Debouncing

Το πάτημα ενός κουμπιού δεν είναι ιδανικό. Λόγω της μηχανικής φύσης του μπορεί να δημιουργηθούν ταλαντώσεις και το κύκλωμα να δεχτεί πολλά σήματα μικρής διάρκειας πριν το σήμα κουμπιού σταθεροποιηθεί. Αυτό είναι ένα πρόβλημα που έχουμε με το κουμπί για το reset.

Για αυτό υλοποίησα ένα debouncing κύκλωμα που ελέγχει κάθε φορά αν η τιμή που βρίσκεται το reset την προκειμένη στιγμή είναι ίδια με την προηγούμενη. Αν αυτό συμβεί για πολλούς κύκλους σημαίνει ότι το σήμα του κουμπιού είναι σταθερό και το περνάει στο υπόλοιπο κύκλωμα. Αυτό το reset που έχει γίνει debounce χρησιμοποιείται στο υπόλοιπο κύκλωμα.

Παρακάτω φαίνονται κυματομορφές από δοκιμές του debounce κυκλώματος με μικρά σήματα πριν σταθεροποιηθεί το σήμα. Για λόγους προσομοίωσης, περιμένει μόνο για 20 κύκλους του ρολογιού για σταθερό σήμα, στον τελικό κώδικα είναι 256.

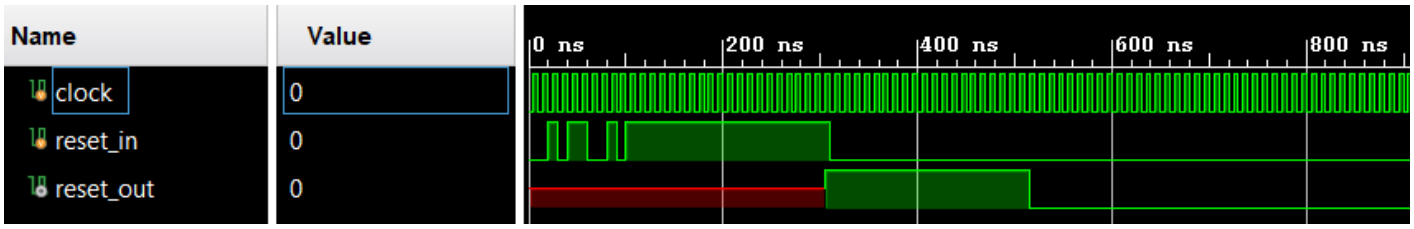

Figure 4: debounce test waveforms for 20 cycles

Οδήγηση των ανόδων

Για να ανάψει ένα ψηφίο του display πρέπει η αντίστοιχη άνοδος του να είναι στο 0. Επειδή, υπάρχει η δυνατότητα να ανάβουμε μόνο μία άνοδο κάθε φορά θα κάνουμε πολύπλεξη στον χρόνο. Δηλαδή εναλλάσω τις ανόδους αρκετά γρήγορα ώστε η αλλαγή να μην είναι αντιληπτή στο μάτι και το ψηφίο να φαίνεται συνέχεια αναμμένο. Επίσης, οι άνοδοι οδηγούνται με κενό 3 κύκλων μεταξύ τους ώστε να αποφευχθεί ghosting ψηφίων μέσα σε άλλα ψηφία. Και τα σήματα για τα segments κάθε ψηφίου δίνεται τουλάχιστον ένα κύκλο πριν το σήμα της ανόδου. Τα σήματα του decoder από το πρώτο μέρος εδώ είναι σταθερά.

Για να οδηγήσω τις ανόδους υλοποίησα έναν μετρητή με case με βάση τον παρακάτω πίνακα:

Τιμή μετρητή	**AN7**	**AN6**	**AN5**	**AN4**	**AN3**	**AN2**	**AN1**	**AN0**
11111	1	1	1	1	1	1	1	1
11110	**0**	1	1	1	1	1	1	1
11101	1	1	1	1	1	1	1	1
11100	1	1	1	1	1	1	1	1
11011	1	1	1	1	1	1	1	1
11010	1	**0**	1	1	1	1	1	1
11001	1	1	1	1	1	1	1	1
11000	1	1	1	1	1	1	1	1
10111	1	1	1	1	1	1	1	1
10110	1	1	**0**	1	1	1	1	1
10101	1	1	1	1	1	1	1	1
10100	1	1	1	1	1	1	1	1
10011	1	1	1	1	1	1	1	1
10010	1	1	1	**0**	1	1	1	1
10001	1	1	1	1	1	1	1	1
10000	1	1	1	1	1	1	1	1
01111	1	1	1	1	1	1	1	1
01110	1	1	1	1	**0**	1	1	1
01101	1	1	1	1	1	1	1	1
01100	1	1	1	1	1	1	1	1
01011	1	1	1	1	1	1	1	1
01010	1	1	1	1	1	**0**	1	1
01001	1	1	1	1	1	1	1	1
01000	1	1	1	1	1	1	1	1
00111	1	1	1	1	1	1	1	1
00110	1	1	1	1	1	1	**0**	1
00101	1	1	1	1	1	1	1	1
00100	1	1	1	1	1	1	1	1
00011	1	1	1	1	1	1	1	1
00010	1	1	1	1	1	1	1	**0**
00001	1	1	1	1	1	1	1	1
00000	1	1	1	1	1	1	1	1

Table 1: Πίνακας Ανόδων

Μέρος Γ - Βηματική Περιστροφή του Μηνύματος με χρήση Κουμπιού

Σε αυτό το μέρος το μήνυμα δεν είναι σταθερό αλλά προχωράει βηματικά με το πάτημα ενός κουμπιού. Παρακάτω φαίνεται ένα σχεδιάγραμμα για αυτό το μέρος:

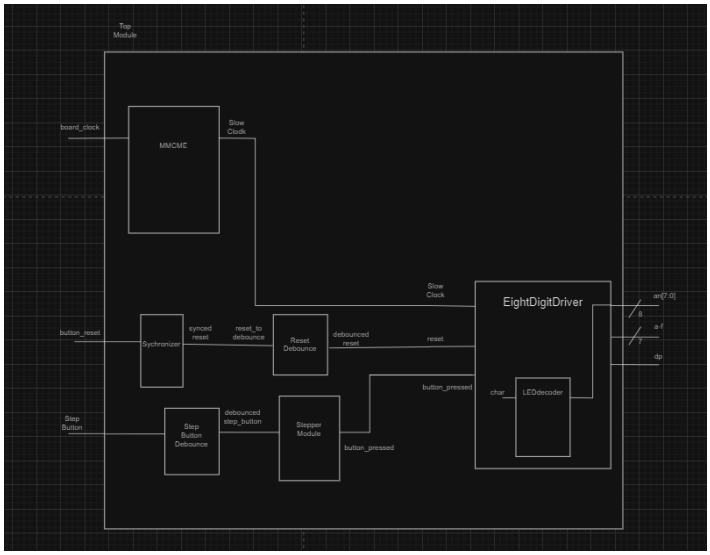

Figure 5: Σχεδιάγραμμα Γ μέρους

Αφού το μήνυμα δεν είναι σταθερό, θα το αποθηκεύσουμε σε μια μνήμη 16 γραμμών 5-bit λέξεων reg [4:0] message [0:15]. Έφτιαξα 8 reg μεταβλητές(char0-char7) που κρατάνε το μήνυμα για τύπωμα σε κάθε πάτημα του κουμπιού και παίρνουν τιμές με βάση την παρακάτω assign:

```
assign {char7,char6, char5, char4, char3, char2, char1, char0} = {message[address], message[
    address+1], message[address+2], message[address+3], message[address+4], message[address
    +5], message[address+6], message[address+7]};
```

Ένα πρόβλημα που μπορεί να προκύψει είναι ότι επειδή το πάτημα του κουμπιού για τον χρηστή κρατάει λίγη ώρα αλλά για το ίδιο το κύκλωμα κρατάει χιλιάδες κύκλους. Αν απλά αυξάναμε το address κάθε φορά που πέρναμε σήμα από το κουμπί τότε το address θα άλλαζε δεκάδες φορές στο πάτημα ενός κουμπιού και τα led θα αναβόσβηναν ή θα έκαναν ghosting μέσα σε άλλα ψηφία. Για αυτό στο stepper module υπάρχει η αντίστοιχη λογική ώστε το πάτημα του κουμπιού να λαμβάνεται υπόψη μια φορά: Κρατάω την προηγούμενη τιμή του σήματος του κουμπιού και ελέγχω αν η τωρινή τιμή του σήματος είναι 1 και η προηγούμενη 0, δηλαδή ελέγχω αν το σήμα μόλις πέρασε την θετική ακμή του (μόλις πατήθηκε το κουμπί) οπότε τότε στέλνω το σήμα button pressed στο driver και στον επόμενο κύκλο κάνω το ίδιο σήμα 0 για να αποφευχθεί το παραπάνω πρόβλημα.

Μέρος Δ - Βηματική Περιστροφή του Μηνύματος με σταθερή καθυστέρηση

Σε αυτό το μέρος προσομειωνούμε την λειτουργία του κουμπιού με έναν μετρητή 23-bit που μετράει 1,6777214 δευτερόλεπτα και τότε προχωράει βηματικά το μήνυμα. Παρακάτω φαίνεται ένα σχεδιάγραμμα για αυτό το μέρος:

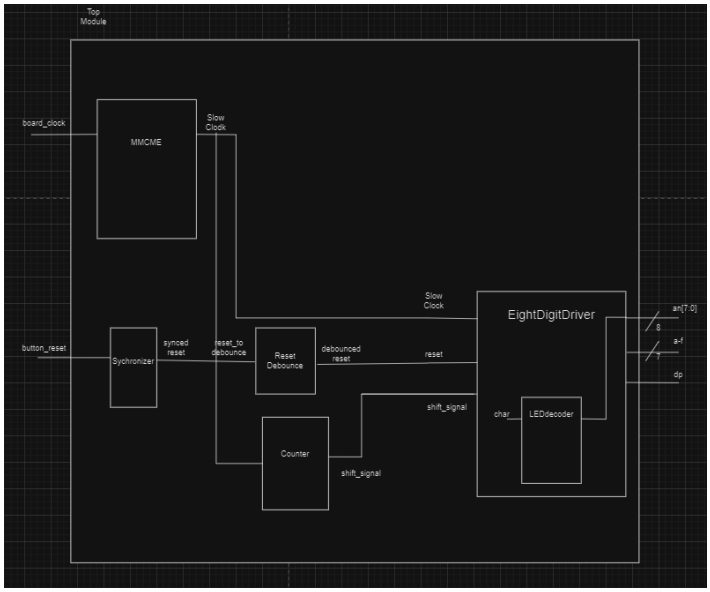

Figure 6: Σχεδιάγραμμα Δ Μέρους

Στο μέρος Δ απλά αντικαθιστούμε στον driver το button_pressed σήμα του κουμπιού με το shift_signal του μετρητή μόλις τελείωσει να μετράει 1,6777214 δευτερόλεπτα.