Όνομα : *Στέφανος Τζαφέρης*

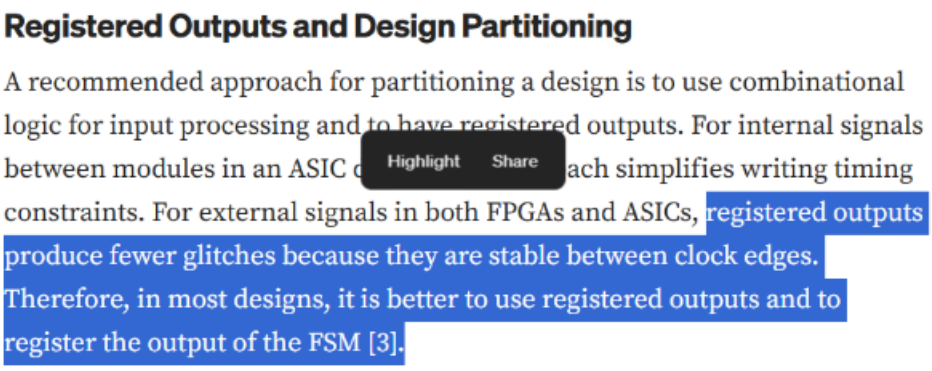
ΑΜ : *1115202200183*

Αναφορά εργασία 2

Ερώτημα 1 :

Αρχικά, ενώ είχα υλοποιήσει τον κώδικα κανονικά και το behavioral simulation είχε τα αναμενόμενα αποτελέσματα, όμως στο post-synthesis timing simulation η έξοδος γίνοταν 1 όταν είχα 2 άσσους συνεχόμενους (aligned στην αρχή των τριάδων) δηλαδή έδινε 1 στο 110 και 111. Μετά από πολύ προσπάθεια, βάζοντας registers σε είσοδο αλλάζοντας σε sync – async το reset και μη βλέποντας αποτέλεσμα, κατάλαβα ότι ουσιαστικά όταν βρισκόμασταν στην κατάσταση D1_is_1 πριν προλάβει να αλλάξει το Din σε 0 (στην περίπτωση του 110) ,γιατί στο simulation το αλλάζω σε ανερχόμενη ακμή του ρολογιού όπως στις διαφάνειες, υπήρχε μία metastable κατάσταση που έβλεπε το Din ως 1 και το state ως D1_is_1 και έκανε το Err = 1 πήγαινε στο Start και μετά έπιανε το 0 ως επομενη είσοδος άρα είχαμε λανθασμένη έξοδο. Το αντιμετώπισα εν τέλη με ένα register στο output, έτσι αυτό το πρόβλημα της κατάστασης είναι μέσα στο ERR_next και το βγάζω στην έξοδο ERR μονο στο clk rising edge. Έτσι η έξοδος είναι σωστή.

Ψάχνοντας βρήκα ότι αυτή η πρακτική είναι σύνηθες και προτείνεται Link: ***https://medium.com/@aiclab.official/finite-state-machines-fsm-aa0e5bf10b03***

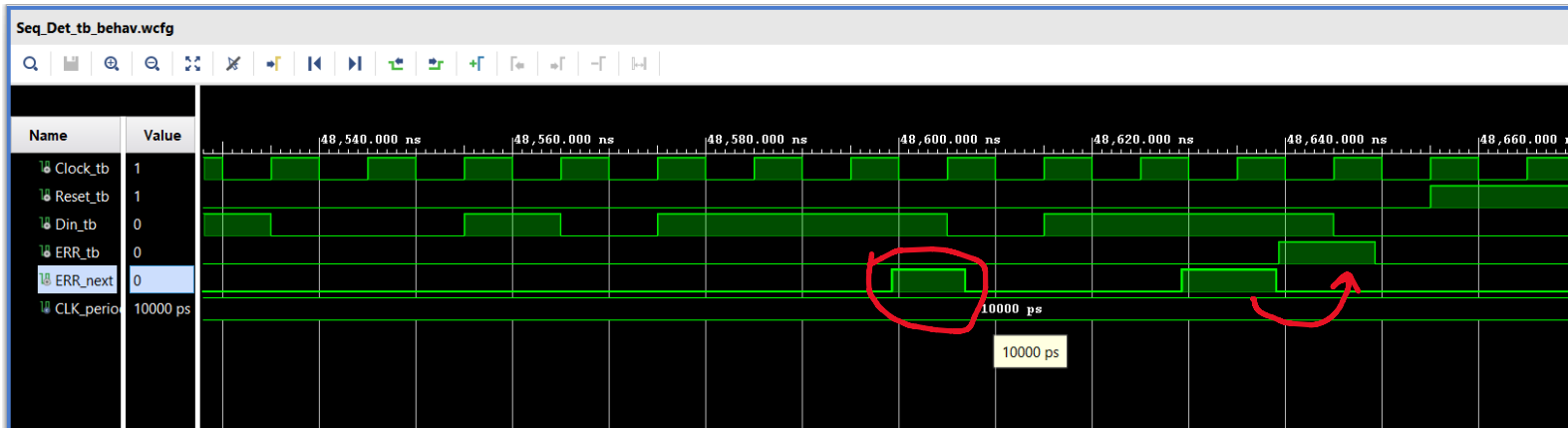


Registered Outputs and Design Partitioning

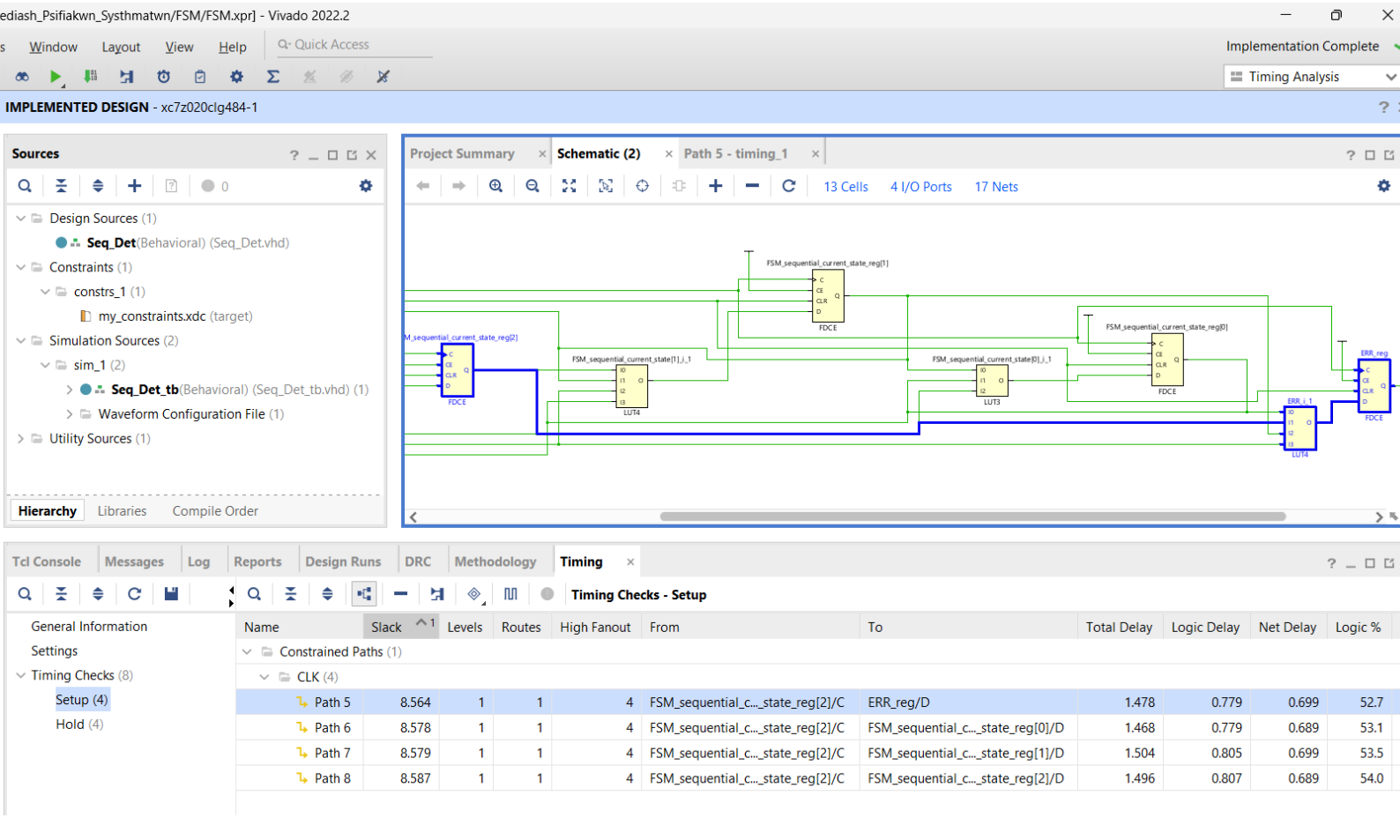
A recommended approach for partitioning a design is to use combinational logic for input processing and to have registered outputs. For internal signals between modules in an ASIC d ach simplifies writing timing constraints. For external signals in both FPGAs and ASICs, registered outputs produce fewer glitches because they are stable between clock edges. Therefore, in most designs, it is better to use registered outputs and to register the output of the FSM [3].

Highlight Share

Στην παρακάτω φωτογραφία βλέπουμε πως το ERR_next δεν βγαίνει στην έξοδο στην λανθασμλενη περίπτωση (1η) γιατι οι ασσοι έρχονται ως 101 110, αλλά στην περίπτωση που είναι 3 ασσοι ευθυγραμισμένοι σε 3αδα 111 και πρέπει ERR =1 περνάει κανονικά απλά φαίνεται στον επόμενο κύκλο.

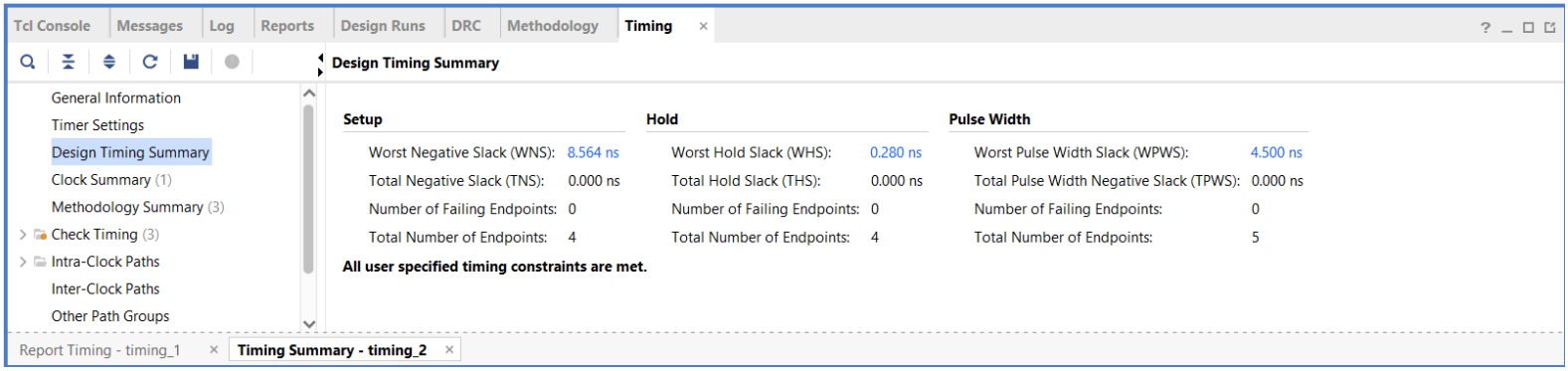


Σχετικα με τους χρόνους set up και την ταχύτητα του κυκλώματος, ερώτημα c:

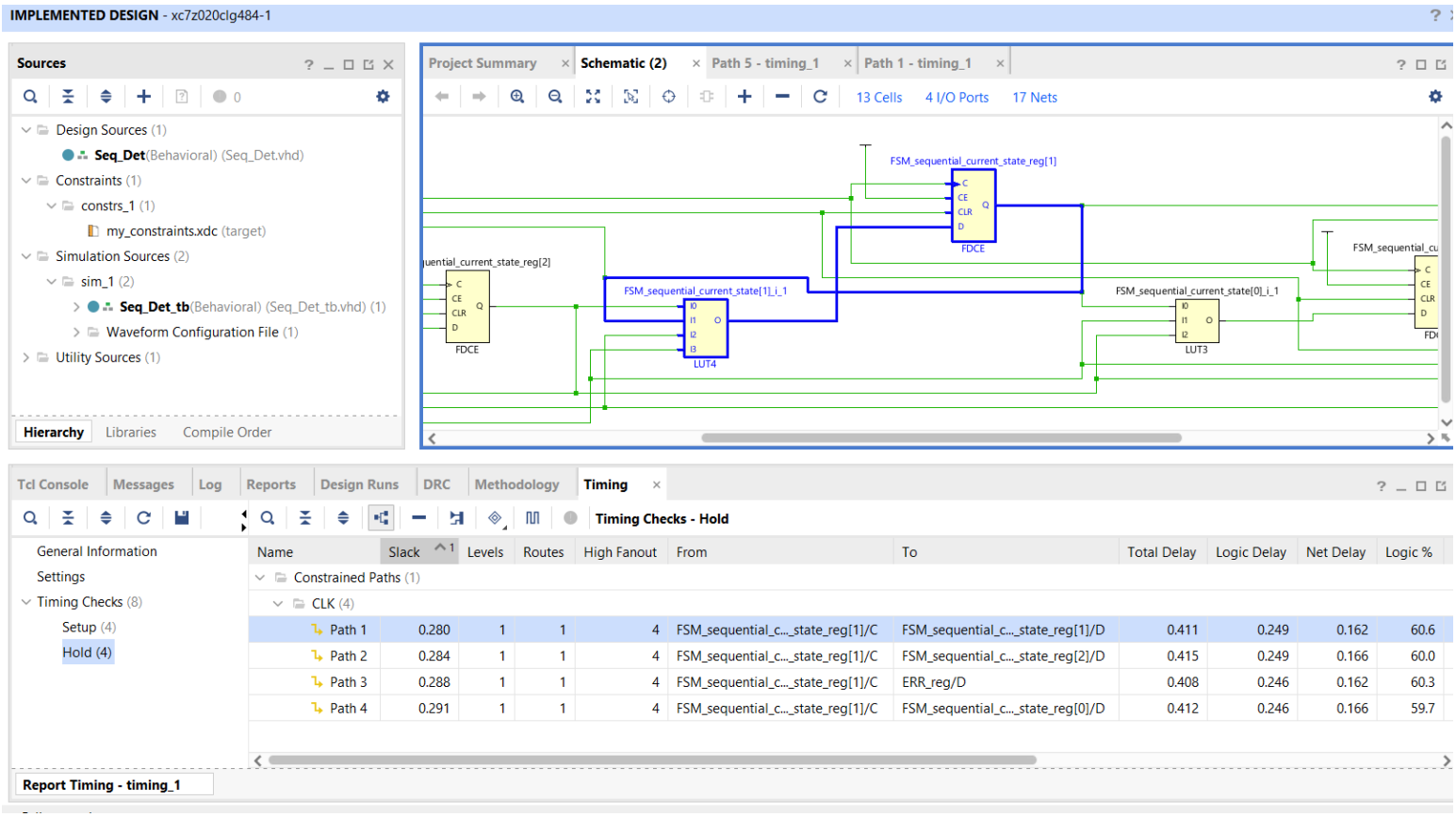


Το κρίσιμο μονοπάτι είναι το path 5 αυτό δηλαδή με το μικρότερο slack. Παρατηρούμε ότι το path 7 έχει το μεγαλύτερο total delay καθαρό, έτσι έψαξα και βρήκα ότι το μικρότερο slack προκύπτει από το skew time και το uncertainty time. Παρόλα αυτά αυτά ταυτίζονται στα δύο μονοπάτια, άρα όπως έχουμε αναφέρει και στο εργαστήριο η διαφορά (δηλαδή μονοπάτι με μικρότερο χρόνο να έχει λιγότερο slack) μπορεί να οφείλεται μέχρι και στη θέση των κυκλωμάτων πάνω στο chip. Όπως και να έχει το WNS = 8.564 **ns**.

Επομένως η ελάχιστη συχνότητα που μπορεί να έχει το κύκλωμα μας είναι 10.000 – 8564 = 1.436 **ns** και μέγιστη συχνότητα 696.37 **MHz.** Μια παρατήρηση είναι ότι όταν έθεσα αυτή την τιμή μέσω του edit timing constraints υπήρχε μία μικρή παραβίαση των περιορισμών του **Pulse Width** του FPGA (υπήρχε – στο timing report). Οπότε για να είμαστε ακριβείς η συχνότητα περιορίζεται λίγο ακόμα από τα φυσικά χαρακτηριστικά του FPGA.

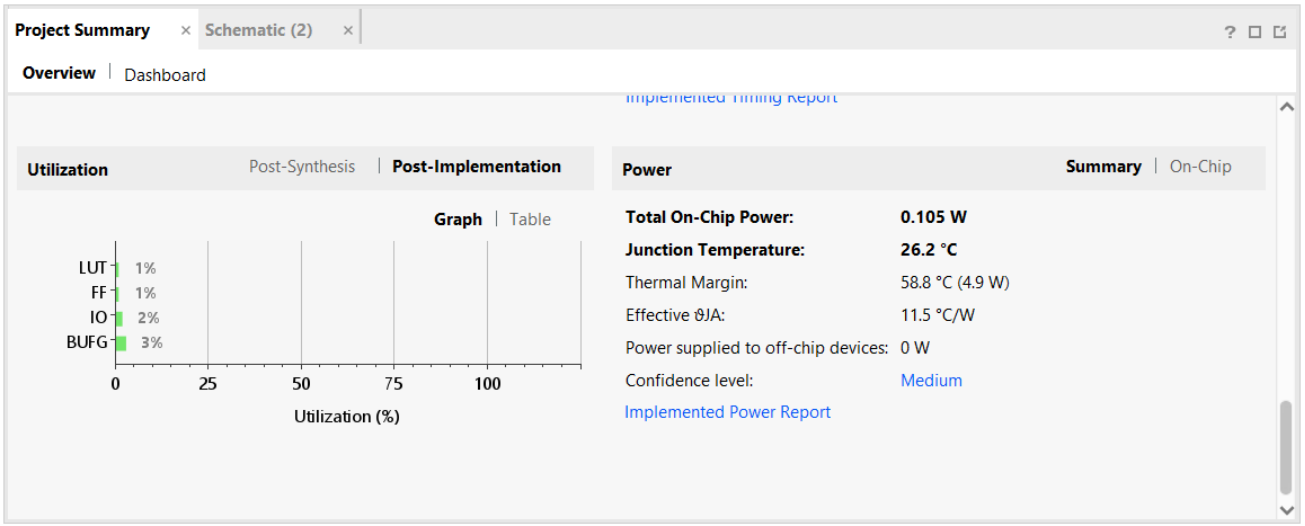


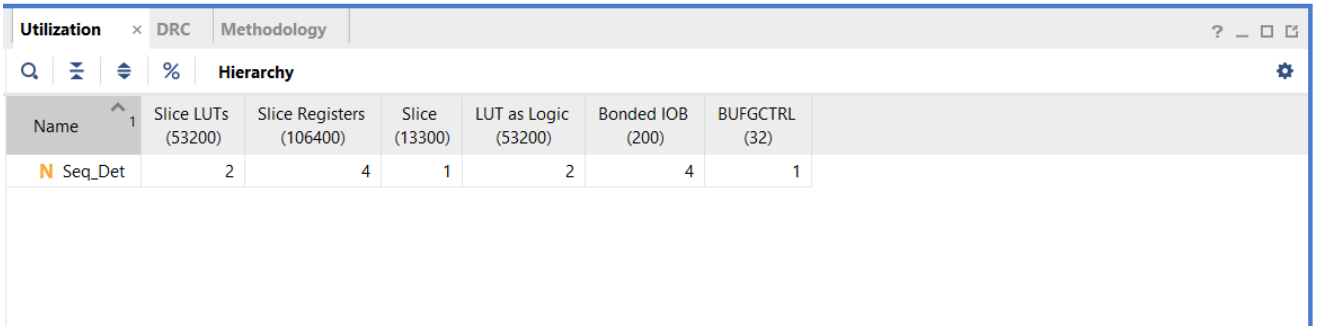
Αντίστοιχα η καθυστέρηση μόλυνσης είναι αυτή με το μικρότερο slack δηλαδή το path 1 με slack 0.280 και total delay 0.411 **ns**



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Logic %
Constrained Paths (1)										
CLK (4)										
Path 1	0.280	1	1	4	FSM_sequential_c..._state_reg[1]/C	FSM_sequential_c..._state_reg[1]/D	0.411	0.249	0.162	60.6
Path 2	0.284	1	1	4	FSM_sequential_c..._state_reg[1]/C	FSM_sequential_c..._state_reg[2]/D	0.415	0.249	0.166	60.0
Path 3	0.288	1	1	4	FSM_sequential_c..._state_reg[1]/C	ERR_reg/D	0.408	0.246	0.162	60.3
Path 4	0.291	1	1	4	FSM_sequential_c..._state_reg[1]/C	FSM_sequential_c..._state_reg[0]/D	0.412	0.246	0.166	59.7

Οι πόροι ,μετά την υλοποίηση, που χρησιμοποιούνται ειναι:





Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Bonded IOB (200)	BUFGCTRL (32)
N Seq_Det	2	4	1	2	4	1

Ερώτημα 2:

Αρχικά, έγινε γρήγορα αντιληπτό ότι δεν γίνεται απλά να έχουμε δύο loops, όπως στο software, και η πρώτη λύση που μου ήρθε στο μυαλό ήταν ότι ο αλγόριθμος έχει κάποια βήματα κοινά και έχει λίγες ομάδες βημάτων (compare, next_i, next_j, swap…) οι οποίες μπορούν να μεταφραστούν σε καταστάσεις FSM. Μία άλλη ιδέα που μου ήρθε είναι να διαβάζαμε όλες τις τιμές σε registers να τα περνάγαμε σε variables και μέσα σε process να εκτελούνταν κανονικά ο κώδικας. Όμως με την distributed ram μπορούμε να διαβάσουμε 1 δεδομένο την φορά (τουλάχιστον με την ram που μας έχετε δώσει που έχει ένα port) οπότε πάλι θα χρειάζοταν κάποιο είδους state για να μπορέσουμε να διαβάσουμε όλες τις τιμές. Επίσης, με αυτό τον τρόπο με ανησύχισε το θέμα κλιμάκωσης, δηλαδή με 1000 τιμές της ROM θα έπρεπε να έχω και 1000 variables και το κύκλωμα θα γινόταν τεράστιο **συνδυαστικό**… Εν τέλη, κατέληξα στην λογική του FSM.

Η ροή για τις καταστάσεις είναι η εξής:

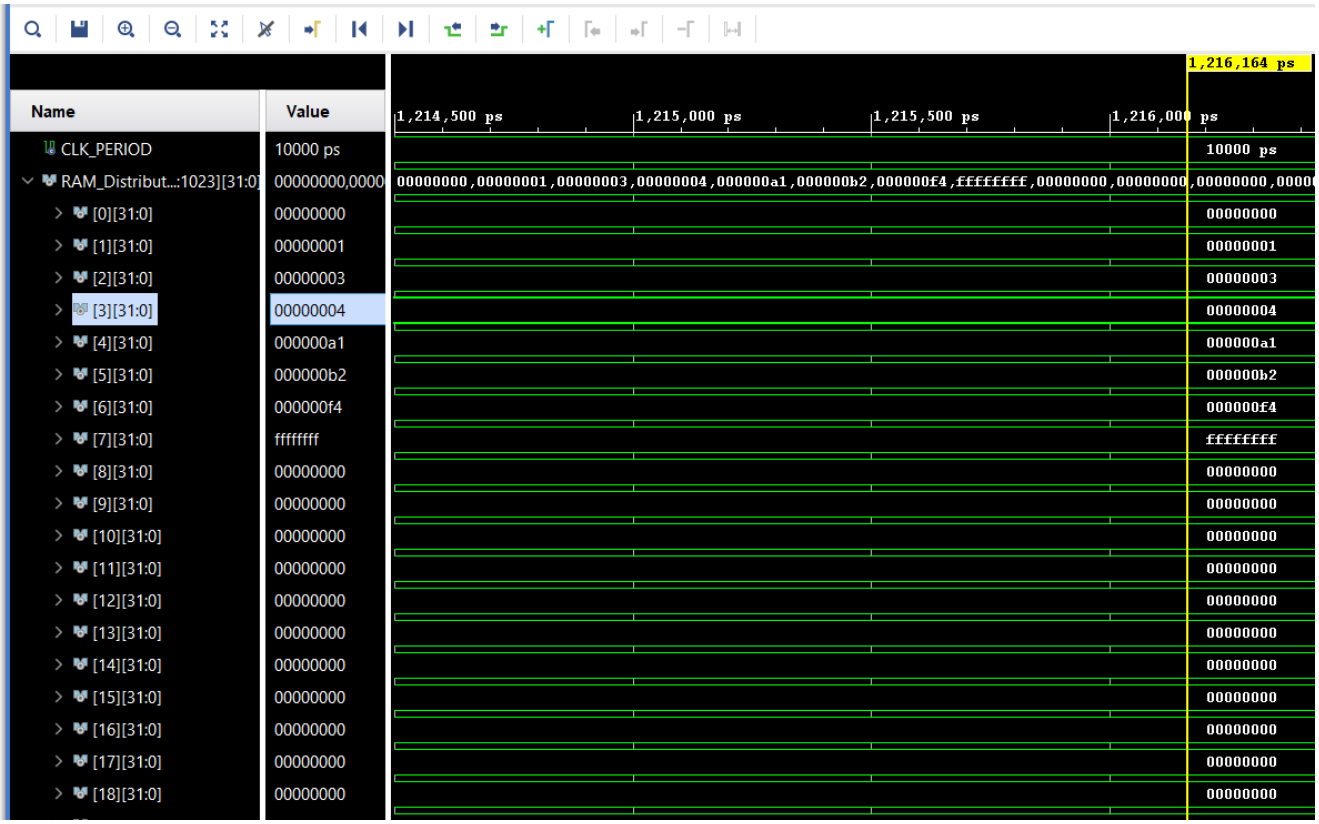
S_LOAD: Πριν ξεκινήσει η ταξινόμηση, γεμίζουμε τη RAM.
S_READ_1, S_READ_2: Επειδή δεν μπορούμε να διαβάσουμε δύο διευθύνσεις ταυτόχρονα από μια απλή RAM, σπάμε την ανάγνωση σε δύο κύκλους.

S_COMPARE: Συγκρίνουμε τους δύο καταχωρητές.Αν r_temp_val > r_temp_val_2, πρέπει να γίνει swap, αν όχι προχωράμε στο επόμενο ζεύγος (αύξηση του j) ή στον επόμενο γύρο (αύξηση του i), προσομοιώνοντας τους βρόχους for.

S_SWAP_1, S_SWAP_2: Όπως και στο read, η εγγραφή των νέων τιμών πρέπει να γίνει σε δύο βήματα.

Υπάρχει ένα σήμα done για να μπορούμε να δούμε στην προσομοίωση ότι τελείωσε ο αλγόριθμος. Σχετικά με την προσομοίωση λοιπόν, αρχικά είχα προσθέσει τον πίνακα της RAM και στο Behavioral simulation όπως φαίνεται και στην παρακάτω εικόνα, έτρεχα απλά τον αλγόριθμο απλά και έβλεπα το εσωτερικό σήμα

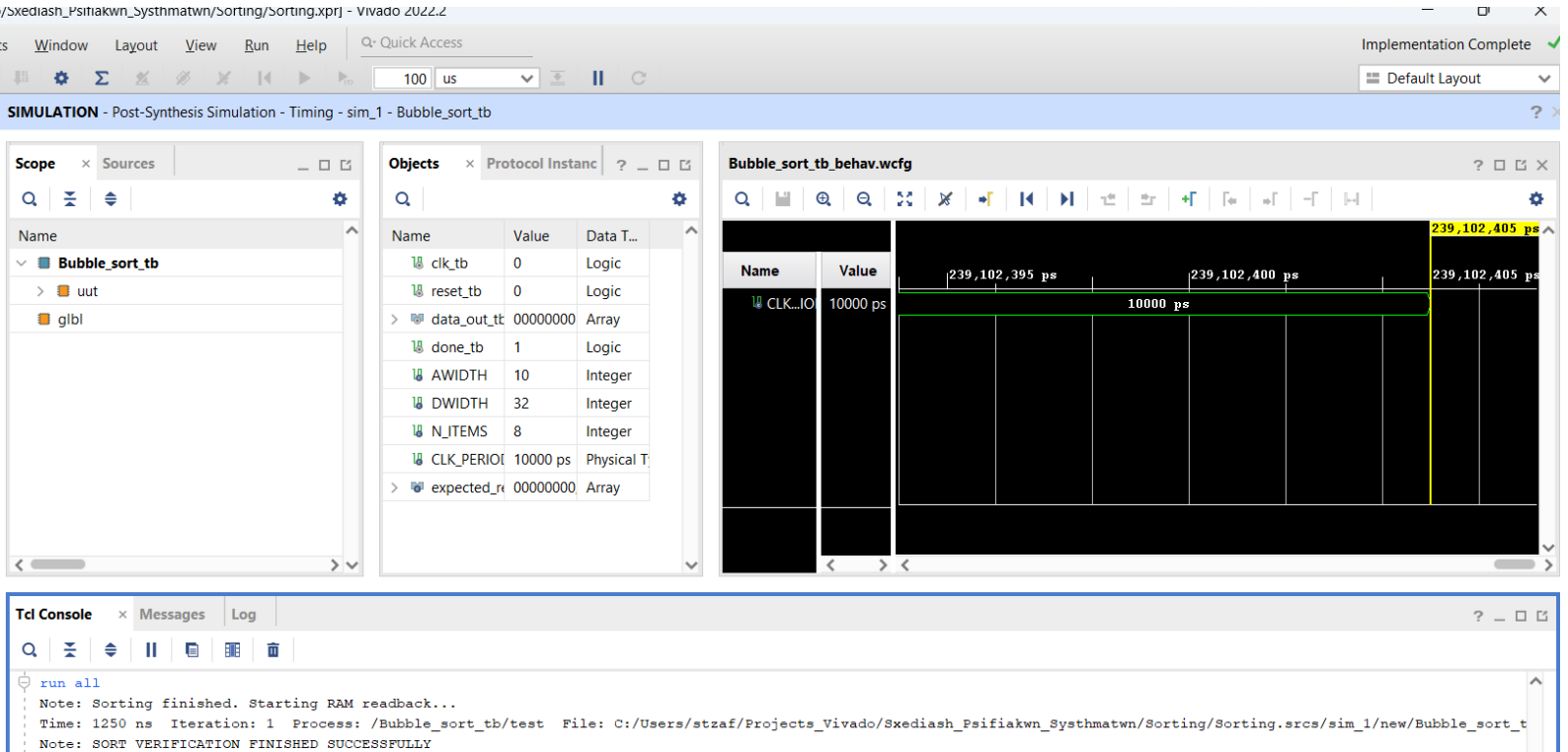
ram_distributed και μπορούσα να δω με το «μάτι» αν οι τιμές είναι ταξινομημένες.



Στο post-synthesis timing simulation όμως αυτή η δομή της RAM δεν είναι προσβάσιμη για να φανεί στις κυματομορφές, η RAM σπάει σε LUTS και είναι πολύ δύσκολο αν όχι αδύνατο να βρούμε ποια luts έχουν ποιες θέσεις μνήμες με τι τιμές. Επομένως, στο παραδοτέο simulation, υπάρχει ένας πίνακας constant με τα input της ROM ταξινομούνται μέσα στο process σε variables , και απλά περιμένουμε μόλις τελειώσει η ταξινόμηση να δούμε αν είναι σωστά. Στην αρχή σκέφτηκα μήπως θα μπορούσα να έχω πρόσβαση στον πίνακα της RAM μέσω του κώδικα της προσομοίωσης κάτι σαν uut.ram_distributed αλλα ψάχνοντας κατάλαβα ότι αυτό είναι διαθέσιμο μόνο στην VHDL 2008 και ονομάζεται **Hierarchical Reference (External Names)**. Έπειτα, σκέφτηκα να προσθέσω μία έξτρα κατάσταση (μετά το S_DONE), που να προσπελαύνει την μνήμη και να περνάει κάθε θέση στην RAM ώστε να μπορώ να ελέγξω αν είναι σωστή. Όμως, έτσι θα αύξανα τις καταστάσεις και την λογική του υλικού μόνο για την προσομοίωση. Εν τέλη επέλεξα μία ενδιάμεση λύση:
Δεν πρόσθεσα νέα κατάσταση απλά ένα σήμα done, και μέσα στην

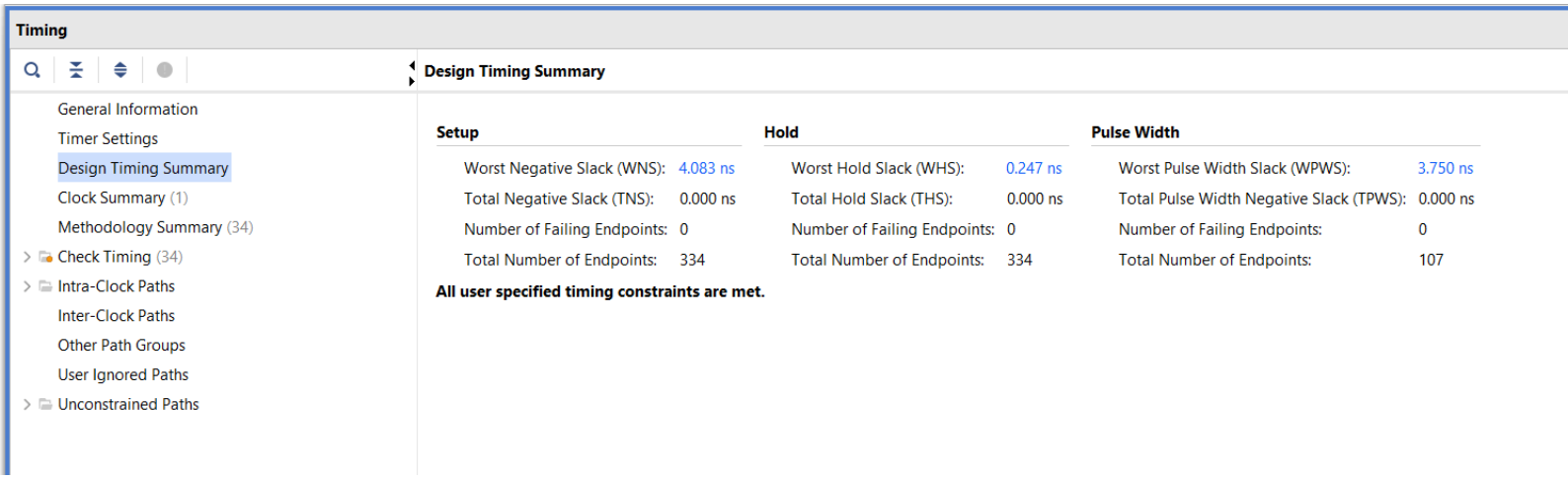
κατάσταση S_DONE περνάω μία φορά την κάθε θέση στην μνήμη, έτσι **a)** θα υπάρχει δυνατότητα να τσεκαριστεί αν η μνήμη είναι σωστή όπως αναμένουμε και **b)** η επιλογή να υλοποιηθεί η σάρωση της μνήμης εντός της κατάστασης S_DONE δεν επιβαρύνει τους πόρους του FPGA, αφού γίνεται πιθανόν ο synthesizer επαναχρησιμοποιεί τον μετρητή r_i που είναι ήδη απαραίτητος για την αρχική φάση φόρτωσης (S_LOAD). Έτσι, αποκτούμε δυνατότητα ανάγνωσης των αποτελεσμάτων χωρίς επιπλέον κόστος σε υλικό. Τέλος, επειδή και πάλι δεν ήταν εύκολο να δω τα αποτελέσματα του Data_out και να καταλάβω πότε τελείωσε το sort ώστε να τσεκάρω την μνήμη μέσω της κυματομορφής, στον κώδικα προσομοίωσης πρόσθεσα και ένα loop το οποίο ξεκινάει μόλις το done γίνει 1 δηλαδή τελειώσει η sort, διαβάζει συγχρονισμένα με το ρολόι αν η τιμή που βγάζει το κύκλωμα (data_out_tb) είναι ίση με την τιμή που έχουμε στο expected_results(i). Αν δεν είναι ίσα, τυπώνει ERROR στην κονσόλα και ότι το sort ήταν unsuccesful. Επίσης το κύκλωμα αλλάζει τα δεδομένα στην ανερχόμενη ακμή (rising edge). Το testbench διαβάζει τα δεδομένα στην κατερχόμενη ακμή (falling edge). Αυτό γίνεται για να είμαστε σίγουροι ότι τα δεδομένα έχουν σταθεροποιηθεί –setup time- πριν τα ελέγξουμε (όταν το είχα rising και στα δύο έπαιρνα λάθος και μέσω του μηνύματος στην κονσόλα κατάλαβα ότι κοίταγε κάποια προηγούμενη τιμή).

Το simulation για να τρέξει και να φαίνονται τα μηνύματα καθαρά, έχω βάλει ένα wait 100 sec μετά την πρώτη φορά που τρέχει άρα αν κάνετε run simulation -> clear console και run θα φανεί η παρακάτω έξοδος:

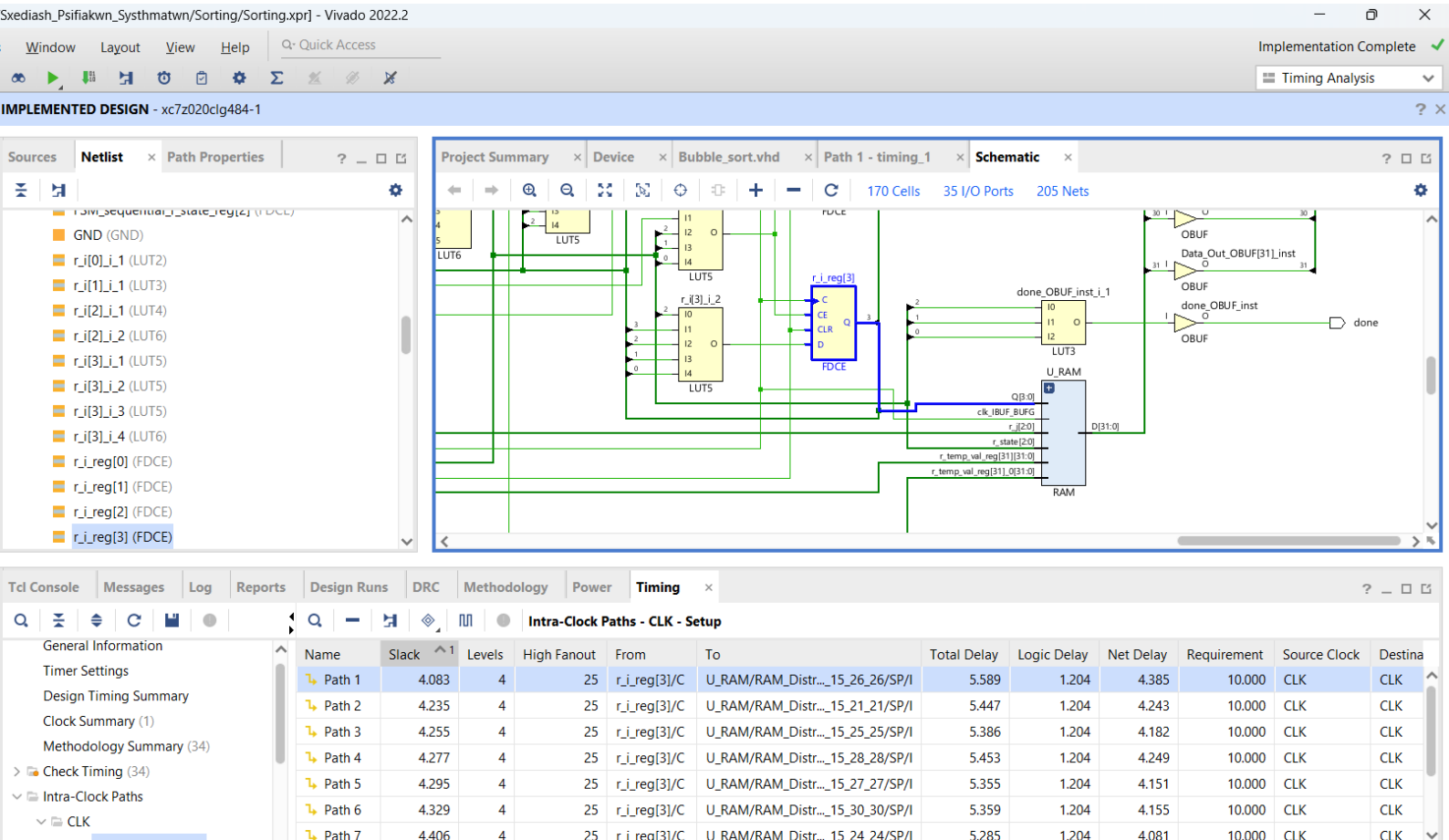


Μία διευκρίνιση για το simulation είναι ότι αν θέλουμε να αλλάξουμε τις τιμές στην ROM, πρέπει να βάλουμε και την είσοδο στο simulation στον constant array και αντιστοιχα αν αλλάξει το πλήθος των στοιχείων πρέπει να αλλάξει η constant N_ITEMS σε Bubble_sort.vhd και σε Bubble_sort_tb.vhd).

Χρόνοι - Πόροι

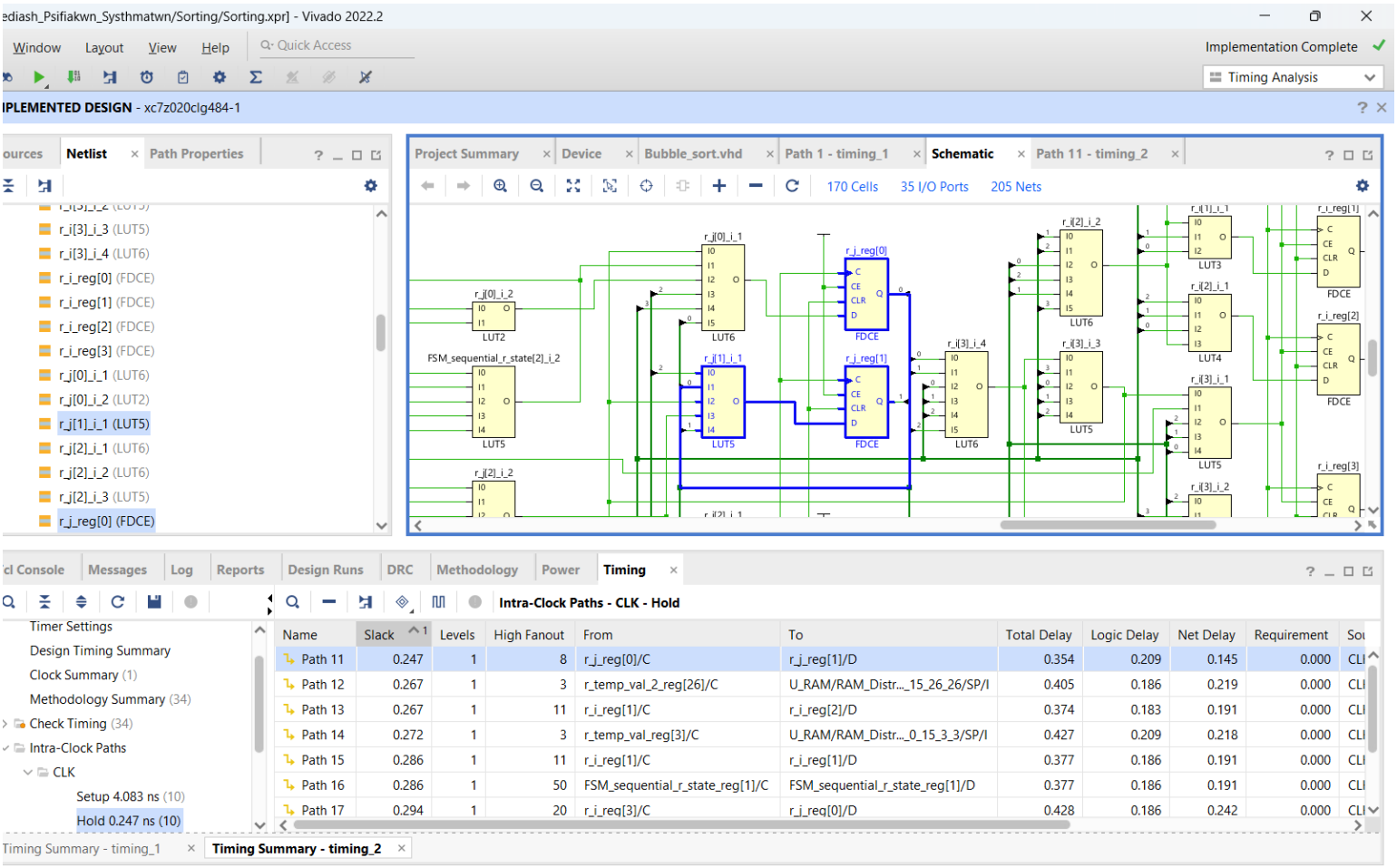


WNS : 4.083 ms άρα ελάχιστη περίοδος T = 10 – 4.083 = 5.917 **ns** με μέγιστη συχνοτητα F = 169 **MHz.** Κρίσιμο μονοπάτι είναι το **path 1** με καθυστέρηση **5.589 ns**

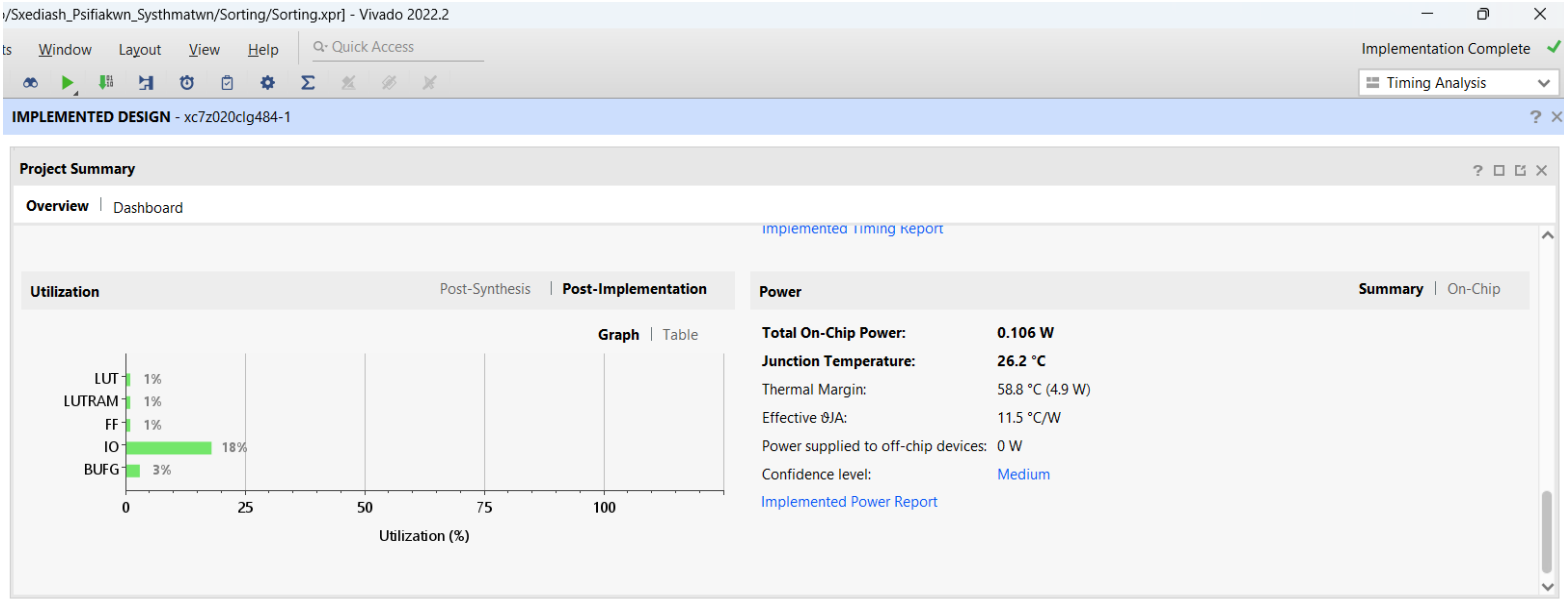


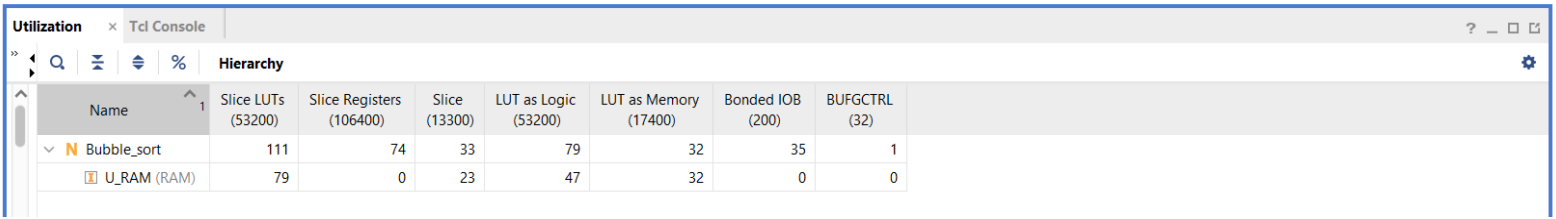
Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destina
Path 1	4.083	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_26_26/SP/I	5.589	1.204	4.385	10.000	CLK	CLK
Path 2	4.235	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_21_21/SP/I	5.447	1.204	4.243	10.000	CLK	CLK
Path 3	4.255	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_25_25/SP/I	5.386	1.204	4.182	10.000	CLK	CLK
Path 4	4.277	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_28_28/SP/I	5.453	1.204	4.249	10.000	CLK	CLK
Path 5	4.295	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_27_27/SP/I	5.355	1.204	4.151	10.000	CLK	CLK
Path 6	4.329	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_30_30/SP/I	5.359	1.204	4.155	10.000	CLK	CLK
Path 7	4.406	4	25	r_i_reg[3]/C	U_RAM/RAM_Distr..._15_24_24/SP/I	5.285	1.204	4.081	10.000	CLK	CLK

Αντίστοιχα για τους χρόνους μόλυνσης το **path 11** είναι το μονοπάτι και μας δίνει delay 0.354 **ns** και slack 0.247 **ns**



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 11	0.247	1	8	r_j_reg[0]/C	r_j_reg[1]/D	0.354	0.209	0.145	0.000
Path 12	0.267	1	3	r_temp_val_2_reg[26]/C	U_RAM/RAM_Distr..._15_26_26/SP/I	0.405	0.186	0.219	0.000
Path 13	0.267	1	11	r_i_reg[1]/C	r_i_reg[2]/D	0.374	0.183	0.191	0.000
Path 14	0.272	1	3	r_temp_val_reg[3]/C	U_RAM/RAM_Distr..._0_15_3_3/SP/I	0.427	0.209	0.218	0.000
Path 15	0.286	1	11	r_i_reg[1]/C	r_i_reg[1]/D	0.377	0.186	0.191	0.000
Path 16	0.286	1	50	FSM_sequential_r_state_reg[1]/C	FSM_sequential_r_state_reg[1]/D	0.377	0.186	0.191	0.000
Path 17	0.294	1	20	r_i_reg[3]/C	r_i_reg[0]/D	0.428	0.186	0.242	0.000





Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Bonded IOB (200)	BUFGCTRL (32)
Bubble_sort	111	74	33	79	32	35	1
U_RAM (RAM)	79	0	23	47	32	0	0