ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΛΕΓΚΤΗΣ ΜΗΧΑΝΗΣ ΑΥΤΟΜΑΤΟΥ ΠΩΛΗΤΗ

Κορομπόκης Κωνσταντίνος

Επιβλέπων: Βαρτζιώτης Φώτιος

Επίκουρος καθηγητής

Άρτα, Νοέμβριος 2023

VENDING-MACHINE CONTROLLER

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 21/11/2023

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Βαρτζιώτης Φώτιος,

2. Μέλος επιτροπής

   Δουμένης Γρηγόριος,

3. Μέλος επιτροπής

   Στεργίου Ελευθέριος,

Δήλωσημηλογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Κορομπόκης, Κωνσταντίνος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, κ. Φώτιο Βαρτζιώτη για την πολύτιμη βοήθεια, υποστήριξη και την θετική συμβολή του για την υλοποίηση της συγκεκριμένης πτυχιακής εργασίας.

ΠΕΡΙΛΗΨΗ

Η πτυχιακή εργασία αναφέρεται στον σχεδιασμό και την υλοποίηση ενός αυτόματου πωλητή με χρήση της VHDL. Ο βασικότερος στόχος είναι η βελτίωση της λειτουργικότητάς και η δημιουργία ενός αξιόπιστου και αποδοτικού συστήματος που διαχειρίζεται με ακρίβεια τη διαδικασία διανομής γλυκών. Χρησιμοποιώντας τεχνικές FSM(Μηχανές Πεπερασμένων Καταστάσεων), ο αυτόματος πωλητής είναι σε θέση να διαχειρίζεται αποτελεσματικά συναλλαγές, εξασφαλίζοντας την ορθή επικύρωση κερμάτων, καθώς και την επιστροφή τους. Η δομή της VHDL και οι δυνατότητες των FSM χρησιμοποιούνται για τη μοντελοποίηση και υλοποίηση της συμπεριφοράς του αυτόματου πωλητή. Ενώ παράλληλα αναπτύσσετε μέσω διαφορετικών καταστάσεων με βάση τις εισόδους του χρήστη και τις συγκεκριμένες συνθήκες, διασφαλίζει και τον κατάλληλο χειρισμό της επικύρωσης κερμάτων, της αποδοχής κερμάτων αλλά και της διανομής γλυκών. Εκτός από τη βασική λειτουργικότητα, η εργασία εμβαθύνει στην μεθοδολογία σχεδιασμού, περιγράφοντας την δημιουργία των πρόσθετων καταστάσεων στην μηχανή, οι οποίες προστέθηκαν ώστε να αποτρέψουν λανθασμένες ή γρήγορες εισαγωγές κερμάτων. Στην συνέχεια, διευρύνετε το project με την ενσωμάτωση των μηχανισμών επικύρωσης και αποδοχής κερμάτων, οι οποίοι επικοινωνούν αλληλοδιάδοχα για να δημιουργήσουν έναν απλό και φιλικό προς το χρήστη αυτόματο πωλητή. Επιπρόσθετα σημαντική καθορίστηκε η χρήση των συστατικών(components), επιτρέποντας την ενεργοποίηση ενός μηχανισμού “χειραψίας” (handshake) μεταξύ του εξωτερικού κυκλώματος και του ελεγκτή του αυτομάτου πωλητή. Αυτός ο μηχανισμός επαληθεύει την σωστή διανομή των γλυκών και των συναλλαγών με κέρματα, ενισχύοντας την απόδοση του αυτόματου πωλητή και την ανταπόκρισή του στις ανάγκες των καταναλωτών. Εν κατακλείδι, η παρούσα πτυχιακή εργασία μπορεί να αποτελέσει ένα χρήσιμο σημείο αναφοράς για την κατανόηση της προσέγγισης FSM, της μοντελοποίησης συμπεριφοράς και των τεχνικών προσομοίωσης για το σχεδιασμό αυτόματων πωλητών.

Λέξεις-κλειδιά: Αυτόματος πωλητής, VHDL, FSM, Σχεδίαση κυκλωμάτων.

ABSTRACT

This senior thesis is about the design and implementation of a vending machine using VHDL. The main objective is to improve the functionality and create a reliable and efficient system that accurately manages the process of dispensing candies. Utilizing FSM(Finite State Machines) techniques, the vending machine is able to manage transactions efficiently, ensuring correct validation of coins as well as their return as change. The VHDL structure and the capabilities of FSMs are used to model and implement the behavior of the vending machine. While deploying through different states based on user inputs and specific conditions, it also ensures proper handling of coin validation, coin acceptance, and candy dispensing. In addition to the basic functionality, the thesis delves into the design methodology, describing the creation of additional states in the machine, which were added to prevent incorrect or fast coin inputs. Subsequently, the project is expanded by incorporating the coin validation and acceptance mechanisms, which communicate with each other to create a simple and user-friendly vending machine. Furthermore, the utilization of the components was defined as important, allowing the activation ofa handshake communication between the external circuit and the vending machine controller. This mechanism verifies the correct dispensing of candies and coin transactions, enhancing the vending machine's performance and responsiveness to consumer needs. In conclusion, this thesis can serve as a useful reference for understanding the FSM approach, behavioral modeling, and simulation techniques for designing vending machines.

Keywords: Vending Machine, VHDL, FSM, Circuit Design.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

1 Εισαγωγή

1.1 Εισαγωγή και σκοπός της πτυχιακής

Οι αυτόματοι πωλητές χρησιμοποιούνται για τη διανομή μικρών διαφορετικών προϊόντων, όταν εισάγεται ένα κέρμα. Αυτά τα μηχανήματα μπορούν να υλοποιηθούν με διάφορους τρόπους ανάλογα με την αρχιτεκτονική δομή και την σχεδίαση που απαιτείται, η οποία μπορεί να διαφέρει ανάλογα την χρήση και την εφαρμογή. Ωστόσο, ο σχεδιασμός και η υλοποίηση ενός αυτόματου πωλητή δεν αποτελεί ένα απλό project και απαιτεί προσεκτική εξέταση διαφόρων παραγόντων, όπως η διεπαφή χρήστη, η διαχείριση αποθεμάτων και η επεξεργασία πληρωμών. Σε αυτό το σημείο μπαίνει στο παιχνίδι η VHDL, ή γλώσσα περιγραφής υλικού ολοκληρωμένου κυκλώματος πολύ υψηλής ταχύτητας. Η VHDL είναι μια γλώσσα περιγραφής υλικού που χρησιμοποιείται για τη σχεδίαση ψηφιακών κυκλωμάτων και συστημάτων και μπορεί να χρησιμοποιηθεί για τη δημιουργία πολύπλοκων συστημάτων όπως των αυτόματων πωλητών με ευκολία. Στην παρούσα εργασία περιγράφεται ο σχεδιασμός ενός αυτόματου πωλητή για γλυκό, με την χρήση του μοντέλου μηχανής πεπερασμένων καταστάσεων. Η υλοποίηση περιλαμβάνει μια διαδικασία με 14 καταστάσεις, βασισμένη σε μηχανή τύπου Moore (Moore FSM). Ο αυτόματος πωλητής υποστηρίζει ένα προϊόν και τρία κέρματα, και είναι σε θέση να δέχεται κέρματα σε οποιαδήποτε σειρά. Όταν κατατίθεται το απαιτούμενο ποσό διανέμεται το γλυκό, ενώ επιστρέφει τα ρέστα αν το ποσό που έχει εισαχθεί είναι μεγαλύτερο από την τιμή του προϊόντος. Με βάση τον κώδικα ενός απλού ελεγκτή μηχανής αυτόματου πωλητή που βρίσκεται στο βιβλίο του VolneiA.Pedroni, CircuitDesignwith VHDL 3rd Edition, σκοπός της πτυχιακής εργασίας ήταν να ενσωματωθούν τα πρόσθετα χαρακτηριστικά από το πρόβλημα 9.3 (Κεφάλαιο 9), ώστε εξασφαλίζεται η ασφαλέστερη λειτουργία του μηχανήματος. Παρόλα αυτά, σχεδιάζοντας με βάση τον συγκεκριμένο κώδικα υπήρξαν κάποιοι περιορισμοί, οπότε έπρεπε να υλοποιηθεί το project όσο πιο απλά γίνονταν. [22](Pedroni, 2004, 231) Ακολουθώντας λοιπόν την ίδια λογική του αρχικού κώδικα, προστέθηκαν καταστάσεις που περιέχουν χαρακτηριστικά επιστροφής χρημάτων, χωρίς να χρειαστεί να αλλάξουν οι αρχικές καταστάσεις της μηχανής. Με αυτόν τον τρόπο αποτρέπονται πιθανές δυσλειτουργίες του αυτόματου πωλητή, στην περίπτωση που ο πελάτης εξακολουθεί να τοποθετεί κέρματα, και αφού έχει καταβληθεί το απαιτούμενο ποσόν. Επιπλέον περιγράφεται εκτενώς η λειτουργία και η μεθοδολογία σχεδίασης του κώδικα, αναλύοντας σε διάφορα κεφάλαια τους προσθέτους κώδικες που χρειάστηκαν για την επιτυχή υλοποίηση του αυτόματου πωλητή. Τέλος με προσομοιώσεις waveform υποδεικνύεται η ορθή λειτουργία του Αυτόματου πωλητή, καθώς επίσης σχολιάστηκαν καθ' όλη την διάρκεια του project οι προκλήσεις που αντιμετωπίστηκαν.

2 Γλώσσα περιγραφής υλικού VHDL

2.1 Ιστορία της VHDL

Τα αρχικά της γλώσσας VHDL προέρχονται από τα αρχικά VHSIC Hardware Description Language, όπου VHSIC προέρχεται από το Υπουργείο Αμύνης των Ηνωμένων Πολιτειών Αμερικής και σημαίνει Very High-Speed Integrated Circuits Program, δηλαδή υψηλής ταχύτητας πρόγραμμα ολοκληρωμένων κυκλωμάτων. Πρόκειται για μία γλώσσα περιγραφής υλικού η οποία

μπορεί να χρησιμοποιηθεί στη μοντελοποίηση και στην προσομοίωση της συμπεριφοράς και της δομής ψηφιακών συστημάτων σε πολλαπλά επίπεδα, ξεκινώντας από τις βασικές ρυθμίσεις συστήματος μέχρι λογικά σχήματα, σχεδιαστικές παραμέτρους, καταγραφή δεδομένων και σκοπούς επαλήθευσης. [27](*VHDL*, n.d.) Από το 1987, η VHDL έχει τυποποιηθεί από το Ινστιτούτων Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών (IEEE), με τον κωδικό IEEE Std1076-2019 να αποτελεί την τελευταία έκδοση της γλώσσας. Τα αρχεία .vhd αποτελούν βασικό τύπο αρχείου που χρησιμοποιούνται ως αρχεία εισόδου στα λογισμικά ψηφιακής σχεδίασης κυκλωμάτων (CAD) προκειμένου να σχεδιαστούν σύνθετα ολοκληρωμένα κυκλώματα. (Καλόμοιρος, 2015) Η VHDL χρησιμοποιείται ευρέως για την περιγραφή και υλοποίηση ψηφιακών συστημάτων σε λογικές διατάξεις, τύπου FPGAs (Field Programmable Gate Arrays-Διατάξεις πυλών προγραμματιζόμενες στο πεδίο) και CPLDs (Complex Programmable Logic Devices-Σύνθετες προγραμματιζόμενες λογικές διατάξεις). Επίσης, έχει καθιερωθεί ως πρότυπο στη σχεδίαση ηλεκτρονικών κυκλωμάτων ASICs (Application Specific Integrated Circuits). Ας αναφερθεί ότι εκτός από τη γλώσσα VHDL ευρύτατη χρήση και αποδοχή έχει βρει διεθνώς και η γλώσσα περιγραφής υλικού Verilog.

Η ανάπτυξη της VHDL ξεκίνησε το 1981 από το Υπουργείο Άμυνας των Ηνωμένων Πολιτειών για την αντιμετώπιση της "κρίσης" του κύκλου ζωής των υλικών. Το κόστος της επαναγοράς ηλεκτρονικού υλικού, λόγω των απαρχαιωμένων τεχνολογιών τους, έφτανε σε κρίσιμο σημείο, επειδή η λειτουργία των εξαρτημάτων δεν ήταν επαρκώς τεκμηριωμένη και τα διάφορα στοιχεία που συνθέτουν ένα σύστημα επαληθεύονταν μεμονωμένα, χρησιμοποιώντας ένα ευρύ φάσμα διαφορετικών και ασυμβίβαστων γλωσσών και εργαλείων προσομοίωσης. [10](*The History of VHDL*, 2018) Υπήρχε, λοιπόν, η απαίτηση για μια γλώσσα με ευρύ φάσμα περιγραφικών δυνατοτήτων που θα λειτουργούσε το ίδιο σε οποιονδήποτε προσομοιωτή και ήταν ανεξάρτητη από την τεχνολογία ή τη μεθοδολογία σχεδιασμού. Η διαδικασία τυποποίησης για τη VHDL είχε το μοναδικό χαρακτηριστικό ότι η συμμετοχή και η ανατροφοδότηση από τη βιομηχανία αναζητήθηκαν σε πρώιμο στάδιο. Η πρώτη γλώσσα βάσης (έκδοση 7.2) δημοσιεύτηκε 2 χρόνια πριν από το πρότυπο, έτσι ώστε η ανάπτυξη του εργαλείου να μπορεί να ξεκινήσει σοβαρά πριν από το πρότυπο. Όλα τα δικαιώματα για τον ορισμό της γλώσσας παραχωρήθηκαν από το Υπουργείο Άμυνας στο IEEE προκειμένου να ενθαρρυνθεί η αποδοχή και οι επενδύσεις του κλάδου. Το υπουργείο Αμύνης επιβάλει τη δημιουργία μιας VHDL περιγραφής με κάθε ASIC που παραδίδεται σε αυτό. Ο μόνος τρόπος για να το πετύχουν είναι η χρήση της VHDL σε όλη τη διαδικασία σχεδιασμού, καθιστώντας την VHDL ακόμα πιο δημοφιλή κατά τις διαδικασίες. Λόγω του ότι αποτελεί πρότυπο IEEE, η VHDL πρέπει να περνά από αξιολογήσεις κάθε πέντε χρόνια, ή και λιγότερα, προκειμένου να διασφαλιστεί το ότι συμβαδίζει με τη βιομηχανία στην οποία απευθύνεται. Η πρώτη φορά που ολοκληρώθηκε μία τέτοια αξιολόγηση ήταν το 1993, και η μορφή εκείνη αποτελεί ακόμη και σήμερα την πιο ευρέως υποστηριζόμενη έκδοση της VHDL. Ένα από τα χαρακτηριστικά που εισήγαγε η έκδοση VHDL-1993 ήταν οι κοινές μεταβλητές (shared variables). Δυστυχώς, κάτι τέτοιο δεν είχε νόημα και τρόπο να χρησιμοποιηθεί από τους χρήστες, με αποτέλεσμα το θέμα να λύνεται με την προσθήκη προστατευόμενων τύπων στην VHDL. Η έκδοση VHDL-2000 αποτελεί ακριβώς την έκδοση του 1993 με την προσθήκη αυτή. Η έκδοση του 2002 έμοιαζε σημαντικά με την προηγούμενη, έχοντας, όμως μία σημαντική διαφορά: οι όροι χρήσης των bufferports έχουν χαλαρώσει, καθιστώντας τες πολύ πιο χρήσιμες από όσο ήταν μέχρι τώρα. [29](*VHDL Tutorial*, n.d.) Το 2007, δημιουργήθηκε μια τροποποίηση στο VHDL 2002. Αυτό εισάγει το VHDL Procedural Interface (VHPI) και κάνει μερικές μικρές αλλαγές στα κείμενα του VHDL 2002. Εκτός από το ίδιο το VHPI, δεν προστέθηκαν νέες δυνατότητες. Το VHPI επιτρέπει προγραμματιστική πρόσβαση σε ένα μοντέλο VHDL πριν και κατά τη διάρκεια της προσομοίωσης. 4(*A Brief History of VHDL, 2014*) Με άλλα λόγια, μπορεί πλέον να εισαχθεί στον

προσομοιωτή VHDL και προγράμματα σε άλλες γλώσσες όπως η C. Το 2008 εγκρίθηκε και η4.0 Draft της VHDL, η οποία αντιμετώπιζε τα διάφορα ζητήματα που ανακαλύφθηκαν κατά τη δοκιμαστική περίοδο της έκδοσης 3.0. Σήμερα, η ανάπτυξη του VHDL συνεχίζεται με επιταχυνόμενους ρυθμούς με πολλά νέα χαρακτηριστικά που σίγουρα θα δούμε στο μέλλον.

2.2 Πότε χρησιμοποιείται η VHDL και πότε η Verilog;

Η κύρια διαφορά μεταξύ Verilog και VHDL είναι ότι η Verilog βασίζεται στη γλώσσα C ενώ η VHDL βασίζεται στις γλώσσες Ada και Pascal. Τόσο η Verilog όσο και η VHDL είναι Γλώσσες Περιγραφής Υλικού (HDL). Αυτές οι γλώσσες βοηθούν στην περιγραφή του υλικού ψηφιακών συστημάτων, όπως μικροεπεξεργαστές και flip-flops. Επομένως, αυτές οι γλώσσες είναι διαφορετικές από τις κανονικές γλώσσες προγραμματισμού. Η VHDL είναι μια παλαιότερη γλώσσα ενώ η Verilog είναι η πιο πρόσφατη γλώσσα. [26]*(VerilogVs VHDL: ExplainbyExamples, n.d.)* Οι απόψεις διίστανται για το ποια γλώσσα είναι καλύτερη, αλλά στην πραγματικότητα εξαρτάται από τη γλώσσα που προτιμάται από τον χρήστη. Θα πρέπει να αναφερθεί η ύπαρξη της SystemVerilog, η οποία σχετίζεται πολύ στενά με τη Verilog, δεν αποτελούν αντικείμενο μελέτης της παρούσας πτυχιακής εργασίας καθώς εφαρμόστηκε η χρήση της VHDL. VHDL σημαίνει VHSIC Γλώσσα περιγραφής υλικού και VHSIC σημαίνει ενσωματωμένο κύκλωμα πολύ υψηλής ταχύτητας. Έτσι, σε γενικές γραμμές, η VHDL στην πραγματικότητα είναι Γλώσσα περιγραφής υλικού ενσωματωμένου κυκλώματος πολύ υψηλής ταχύτητας. Ένα από τα βασικά χαρακτηριστικά του VHDL είναι ότι είναι μια γλώσσα με έντονη πληκτρολόγηση, πράγμα που σημαίνει ότι κάθε τύπος δεδομένων (ακέραιος, χαρακτήρας κ.λπ.) έχει προκαθοριστεί από την ίδια τη γλώσσα.[24]*(Prew, 2017)* Όλες οι τιμές ή οι μεταβλητές που ορίζονται σε αυτήν τη γλώσσα πρέπει να περιγράφονται από έναν από τους τύπους δεδομένων. Η VHDL είναι πιο αναλυτική από την Verilog και έχει επίσης σύνταξη που δεν μοιάζει με C. Με την VHDL, υπάρχουν πιθανότητες να γραφτούν περισσότερες γραμμές κώδικα. Η VHDL μπορεί επίσης να φαίνεται πιο φυσική για χρήση κατά καιρούς. Όταν κωδικοποιείτε ένα πρόγραμμα με VHDL, μπορεί να φαίνεται ότι ρέει καλύτερα. Αλλά ίσως αυτή είναι απλώς η προσωπική μου άποψη. Στη Verilog, η γλώσσα είναι πιο συμπαγής, καθώς η γλώσσα Verilog είναι μια γλώσσα μοντελοποίησης υλικού. Η δομή και η μορφή της έχουν πολλά κοινά με τη γλώσσα C. Η Verilog έχει καλύτερη κατανόηση της μοντελοποίησης υλικού, αλλά έχει χαμηλότερο επίπεδο δομών προγραμματισμού σε σύγκριση με τη VHDL. Η Verilog δεν είναι τόσο περίπλοκη όσο η VHDL, γι' αυτό είναι πιο συμπαγής. Συνολικά, η Verilog είναι αρκετά διαφορετική από την VHDL. Υπάρχουν κάποιες ομοιότητες, αλλά επισκιάζονται από τις διαφορές τους. Εξετάζοντας το παρακάτω παράδειγμα κώδικα, μπορούμε να συγκρίνουμε τον τρόπο με τον οποίο μπορεί να προγραμματιστεί ένα MUX μέσω VHDL και Verilog. [16]*(Minns&Elliott, 2008, 146)*

VHDL:

```
process (S0,S1,A,B,C,D)
begin
   case (S0,S1), is
      when "00" => Y <= A;
      when "01" => Y <= B;
      when "10" => Y <= C;
      when "11" => Y <= D;
      when others => Y <= A;
   end case;
end process;
```

Verilog:

```
always @({S0,S1}, A, B, C, D)
      case ({S0,S1})
         2'b00: Y = A;
         2'b01: Y = B;
         2'b10: Y = C;
         2'b11: Y = D;
      endcase
```

Εικόνα 1 Παράδειγμα διαφορών VHDL, Verilog

Η διάταξη αυτών των προγραμμάτων είναι παρόμοια. Μπορεί πολύ εύκολα κανείς να δει τι κάνει κάθε έκδοση του κώδικα. Η έκδοση VHDL είναι μεγαλύτερη από την Verilog, αλλά μπορεί να γίνει καλύτερα κατανοητή.

2.3 Τα 3 Στυλ μοντελοποίησης της VHDL

Η γλώσσα VHDL υποστηρίζει τρία είδη στυλ μοντελοποίησης: το στυλ ροής δεδομένων, το δομικό στυλ και το στυλ συμπεριφοράς. Η ροή δεδομένων και η δομική μοντελοποίηση χρησιμοποιούνται για τη μοντελοποίηση συνδυαστικών κυκλωμάτων, ενώ η μοντελοποίηση συμπεριφοράς χρησιμοποιείται τόσο για συνδυαστικά όσο και για διαδοχικά κυκλώματα.[17](*Modeling Concepts*, 2013) Μια μονάδα VHDL έχει μια καλά καθορισμένη δομή η οποία επιτρέπει τον ορισμό της ενότητας με σαφή και λογικό τρόπο. Μια τυπική μονάδα VHDL έχει δύο κύρια μέρη:

- δήλωση οντότητας και
- μπλοκ αρχιτεκτονικής.

Η δήλωση οντότητας ορίζει τις θύρες εισόδου και εξόδου της μονάδας μιας συσκευής. Το μπλοκ αρχιτεκτονικής στο VHDL καθορίζει τη λειτουργικότητα της συσκευής. Η οντότητα μπορεί να περιέχει τα ονόματα των θυρών, τα μεγέθη των θυρών και τις οδηγίες (είσοδος/έξοδος). Το μπλοκ αρχιτεκτονικής μπορεί να περιλαμβάνει στιγμιαία στοιχεία και τοπικά σήματα/δίκτυα. Το μπλοκ αρχιτεκτονικής χωρίζεται επιπλέον σε τρεις υπό-ενότητες:

- δηλώσεις components,
- δηλώσεις σημάτων και
- λειτουργικός κώδικας.

Η δήλωση των component απαιτείται για την περιγραφή ενός ιεραρχικού σχεδιασμού. Οι δηλώσεις σήματος απαιτούνται για τοπικές συνδέσεις μεταξύ διαφόρων μπλοκ εντός του λειτουργικού κώδικα. Ο λειτουργικός κώδικας μπορεί να περιγραφεί με ποικίλους τρόπους. Το λειτουργικό μπλοκ του μπλοκ αρχιτεκτονικής είναι το σημείο όπου ορίζεται η λειτουργικότητα της μονάδας και ο τρόπος με τον οποίο αυτή υλοποιείται. Οι διεργασίες και οι άμεσες αναθέσεις μπορούν επίσης να είναι στο ίδιο μπλοκ αρχιτεκτονικής. Οι εκχωρήσεις εκτός διεργασιών είναι σχεδόν πάντα συνδυαστικές. Παραδείγματος χάριν, τέτοιες δηλώσεις χρησιμοποιούνται για την περιγραφή της μοντελοποίησης ροής δεδομένων. Οι διεργασίες χρησιμοποιούνται εάν πρέπει να εκτελεστούν διαδοχικές λειτουργίες όταν εναλλάσσονται συγκεκριμένα σήματα

Το στυλ ροής δεδομένων

Η μοντελοποίηση ροής δεδομένων μπορεί να χρησιμοποιηθεί για την περιγραφή συνδυαστικών κυκλωμάτων. Ο βασικός μηχανισμός που χρησιμοποιείται στο στυλ αυτό είναι η ταυτόχρονη ανάθεση. Σε μια ταυτόχρονη εκχώρηση δεδομένων, μια τιμή εκχωρείται σε ένα σήμα. Ένα παράδειγμα της σύνταξης μιας δήλωσης ταυτόχρονης ανάθεσης είναι:

LHS_signal<= RHS_expression;

Όπου το LHS_signal είναι ένα δίκτυο προορισμού ενός ή περισσότερων bit και το RHS_expression είναι μια έκφραση η οποία αποτελείται από διάφορους τελεστές. Ο στόχος στην RHS_expression μπορεί να είναι ένα βαθμωτό δίκτυο (net) ή ένα διανυσματικό δίκτυο (net) ή σταθερή επιλογή bit ενός διανύσματος ή σταθερή, μερική επιλογή ενός διανύσματος ή συνδυασμοί οποιουδήποτε από τα παραπάνω. Οι πράξεις ανάθεσης περιλαμβάνουν τις βασικές συναρτήσεις Boole (τελεστές): and, or, xor, nand, nor και xnor. Αυτές είναι (από προεπιλογή) δύο είσοδοι και μία έξοδος.

```
architecture dataflow of half_adder is
  begin
  sum <= a xor b;
  carry_out <= a and b;
end dataflow;
```

Εικόνα 2 DataflowModeling

Το στυλ μοντελοποίησης δομής

Στη δομική μοντελοποίηση, πολλά στιγμιαία αλληλένδετα κομμάτια συνδέονται για να δημιουργήσουν ένα κύκλωμα. Τα στιγμιαία στοιχεία μπορεί να είναι κομμάτια άλλων μονάδων ή/και πρωτογενών συσκευών. Η χρήση του επιπέδου πύλης (portmap) επιτρέπει την κατασκευή απλών συνδυαστικών κυκλωμάτων. Στο ανώτερο επίπεδο του αρχιτεκτονικού μπλοκ περιγράφεται η σχέση των κατωτέρων κυκλωμάτων και έτσι δημιουργούνται οι συνδέσεις. [28](*VHDL Modelling Styles*, n.d.) Η δομική μοντελοποίηση είναι η πιο χρήσιμη και αποτελεσματική μέθοδος στις περιπτώσεις που υπάρχουν σύνθετα και αλληλένδετα κυκλώματα.

```
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
  component xor_gate                    -- xor component declaration
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;

  component and_gate                    -- and component declaration
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;

begin
  u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
  u2: and_gate port map (i1 => a, i2 => b, o1 => carry_out);
```

Εικόνα 3 StracturalModeling

Στην VHDL, τα components δουλεύουν ως σύμβολα υποκατάστασης για την σχεδίαση της οντότητας, γι' αυτό και μοιάζουν αρκετά στην δήλωση. Ένας δομικός σχεδιασμός που

χρησιμοποιεί components καθορίζει απλώς τη διασύνδεση των components. Αλλά χρειάζεται την χρήση των portmap τα οποία ορίζουν την διασύνδεση των component με όλη την περιβαλλομένη αρχιτεκτονική.

Το στυλ μοντελοποίησης συμπεριφοράς

Η μοντελοποίηση συμπεριφοράς χρησιμοποιείται για την περιγραφή πολύπλοκων κυκλωμάτων. Στη VHDL, η μοντελοποίηση συμπεριφοράς γίνεται στο μπλοκ αρχιτεκτονικής. Μέσα στο μπλοκ αρχιτεκτονικής, οι διεργασίες ορίζονται για τη μοντελοποίηση διαδοχικών κυκλωμάτων. Οι μηχανισμοί (δηλώσεις) για τη μοντελοποίηση της συμπεριφοράς ενός σχεδίου ξεκινούν ως εξής:

```
begin
  ha: process (a, b)
  begin
    if a = '1' then
      sum <= not b;
      carry_out <= b;
    else
      sum <= b;
      carry_out <= '0';
    end if;
  end process ha;
```

Εικόνα 4 BehavioralModeling

Ένα τμήμα(module) μπορεί να περιέχει έναν οποιοδήποτε αριθμό δηλώσεων διεργασίας, και αυτές οι διεργασίες μπορεί να εμπεριέχουν μία ή περισσότερες δηλώσεις με τη σειρά τους. Οι δηλώσεις που εμφανίζονται στο σώμα δήλωσης διαδικασίας κατηγοριοποιούνται ως διαδικαστικές δηλώσεις. Οι διεργασίες εκτελούνται ταυτόχρονα μεταξύ τους (δηλαδή η σειρά με την οποία εμφανίζονται στο μοντέλο δεν έχει σημασία), ενώ οι διαδικαστικές δηλώσεις εκτελούνται με διαδοχικό τρόπο (δηλαδή η σειρά με την οποία εμφανίζονται έχει σημασία). Μια διαδικαστική δήλωση μπορεί να είναι:

1. Διαδικαστικές εργασίες (procedural)

2. Δηλώσεις υπό όρους (conditional)

3. Καταθέσεις υπόθεσης (case)

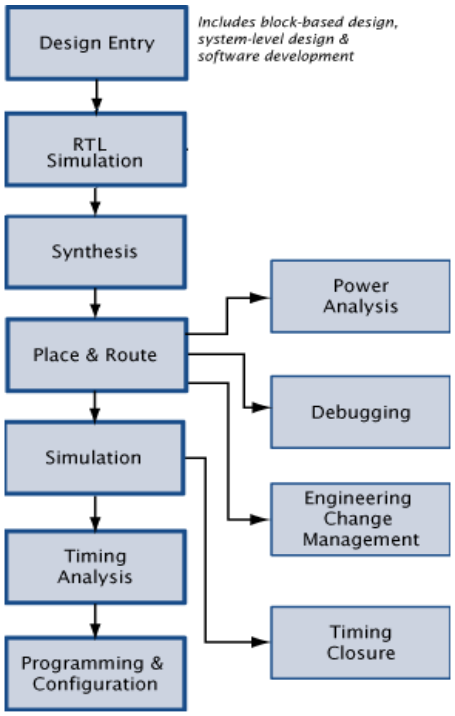
4. Δηλώσεις βρόχου (loop)

5. Δηλώσεις αναμονής (wait)

Οι δηλώσεις διεργασίας με λίστα ευαισθησίας εκτελούνται κατά τη διάρκεια του υπόλοιπου χρόνου, όποτε αλλάζει οποιοδήποτε από τα σήματα στη λίστα ευαισθησίας. Η δήλωση διεργασίας μπορεί να συντεθεί και το κύκλωμα που προκύπτει μπορεί να είναι ένα συνδυαστικό ή διαδοχικό. Προκειμένου το μοντέλο να δημιουργήσει ένα συνδυαστικό κύκλωμα, το μπλοκ διεργασίας δεν θα πρέπει να έχει εντολές ευαίσθητες στις ακμές και πρέπει να έχει όλη την έξοδο που δημιουργείται σε κάθε πρόταση υπό όρους ή πρόταση περίπτωσης εντός της διεργασίας.

2.4 Περιβάλλον Quartus II

Το λογισμικό ανάπτυξης Quartus II παρέχει ένα πλήρες σχεδιαστικό περιβάλλον για τη σχεδίαση συστήματος-σε-προγραμματιζόμενο-τσιπ (SOPC). Ανεξάρτητα από το αν χρησιμοποιείται σε προσωπικό υπολογιστή ή Linux, το λογισμικό Quartus II εξασφαλίζει εύκολη είσοδο στο σχεδιασμό, γρήγορη επεξεργασία και απλό προγραμματισμό συσκευών. Το Quartus II της Intel (πρώην AlteraQuartus II) είναι προγραμματιζόμενο λογισμικό σχεδιασμού λογικών συσκευών. Το Quartus επιτρέπει την ανάλυση και τη σύνθεση σχεδίων HDL, η οποία επιτρέπει στον προγραμματιστή να συντάξει τα σχέδιά του, να εκτελέσει ανάλυση χρονισμού, να εξετάσει διαγράμματα RTL, να προσομοιώσει την αντίδραση ενός σχεδίου σε διαφορετικά ερεθίσματα και να διαμορφώσει τη συσκευή-στόχο με τον προγραμματιστή. Το QuartusPrime περιλαμβάνει μια υλοποίηση VHDL καιVerilog για περιγραφή υλικού, οπτική επεξεργασία λογικών κυκλωμάτων και διανυσματική προσομοίωση κυματομορφής. [19](*NorskVersjon – Quartus II*, n.d.)Ο χρήστης μπορεί εύκολα να συνδυάσει διαφορετικούς τύπους αρχείων σχεδίασης σε ένα έργο, επιλέγοντας τη μορφή καταχώρησης σχεδίασης που λειτουργεί καλύτερα για κάθε λειτουργικό μπλοκ. Μπορεί να χρησιμοποιηθεί το πρόγραμμα επεξεργασίας μπλοκ Quartus II για να δημιουργηθεί το αρχικό μπλοκ διαγράμματος που περιγράφει το σχέδιό του χρήστη σε υψηλό επίπεδο και, στη συνέχεια, να χρησιμοποιηθούν πρόσθετα μπλοκ διαγράμματος, σχηματικά, αρχεία σχεδίασης κειμένου AHDL (.tdf), αρχεία εισαγωγής EDIF (.edf), αρχεία σχεδίασης VHDL (.vhd) και Verilog HDL Design Files (.v) για τη δημιουργία των στοιχείων σχεδίασης χαμηλότερου επιπέδου. Η είσοδος σχεδίασης ανεξάρτητα από την αρχιτεκτονική δίνει μεγάλη ελευθερία δημιουργίας στον χρήστη χωρίς να καταλήγει σε προβλήματα κατά την υλοποίηση της συσκευής. Ο compiler Quartus II βρίσκεται στην καρδιά του συστήματος, παρέχοντας ισχυρή επεξεργασία σχεδίασης που μπορεί να προσαρμοστεί κατάλληλα σε κάθε έργο. Ο αυτόματος εντοπισμός σφαλμάτων και η εκτενής τεκμηρίωση σχετικά με τα μηνύματα σφαλμάτων και προειδοποιήσεων καθιστούν τις τροποποιήσεις σχεδιασμού όσο το δυνατόν απλούστερες.



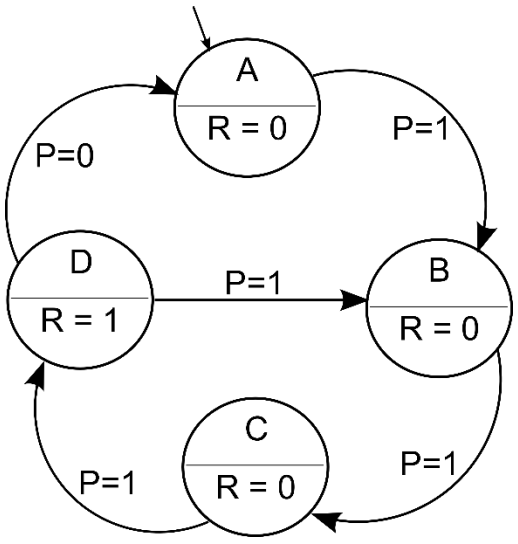
Εικόνα 5 Λογισμικό Quartus

Το λογισμικό Quartus II είναι ένα πλήρως ενσωματωμένο, ανεξάρτητο από αρχιτεκτονική πακέτο για τον σχεδιασμό λογικών δομών με τις προγραμματιζόμενες λογικές συσκευές της Altera (PLD), μερικοί τύποι από αυτές είναι: Arria II, Cyclone II , Cyclone V, MAX V, Stratix IV. Στο πρακτικό κομμάτι της πτυχιακής χρησιμοποιήθηκε Cyclone II και λίγες φορές Cyclone V για την αποφυγή τυχόν σφαλμάτων κατά την υλοποίηση και μεταγλώττιση. 3*(AlteraCorporation, 2008)*

3 Μηχανές πεπερασμένων καταστάσεων(FSM)

3.1 Βασική Θεώρηση

Στα ψηφιακά συστήματα, υπάρχουν δύο βασικοί τύποι κυκλωμάτων. Ο πρώτος τύπος είναι τα συνδυαστικά λογικά κυκλώματα. Στα κυκλώματα συνδυαστικής λογικής, οι έξοδοι εξαρτώνται αποκλειστικά από τις εισόδους. Παραδείγματα κυκλωμάτων συνδυαστικής λογικής είναι οι αθροιστές(adders), οι κωδικοποιητές(encoders) και οι πολυπλέκτες(multiplexers). Στους αθροιστές, η έξοδος είναι απλώς το άθροισμα των εισόδων χωρίς να έχει σημασία το ποιες ήταν οι προηγούμενες είσοδοι ή έξοδοι.[30]*(Williams, 2015)* Ο δεύτερος τύπος ψηφιακών λογικών κυκλωμάτων είναι τα διαδοχικά λογικά κυκλώματα. Στα διαδοχικά λογικά κυκλώματα, οι έξοδοι εξαρτώνται όχι μόνο από τις εισόδους, αλλά και από την παρούσα κατάσταση του συστήματος (δηλαδή, τις τιμές των εξόδων και τυχόν εσωτερικά σήματα ή μεταβλητές). Τα διαδοχικά λογικά κυκλώματα κυμαίνονται σε πολυπλοκότητα από απλούς μετρητές που μετακινούνται από τη μια κατάσταση στην άλλη με μια βασική ακολουθία (π.χ. 0,1,2,3…0,1,2,3…) έως κυκλώματα πολύ μεγάλης κλίμακας, όπως μικροεπεξεργαστές με εκατομμύρια διαφορετικές καταστάσεις κλπ. Στο παρόν κεφάλαιο θα γίνει μία ανάλυση των διαδοχικών λογικών κυκλωμάτων ως μηχανές πεπερασμένης κατάστασης και στον τρόπο μετατροπής αυτών των μηχανών πεπερασμένης κατάστασης στη γλώσσα περιγραφής υλικού VHDL.



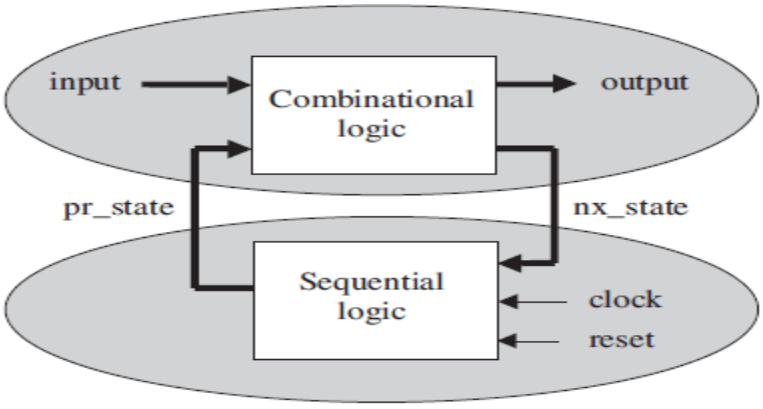
Εικόνα 6 Σχηματικό μιας FSM

Το σύστημα που παρουσιάζεται στο παραπάνω διάγραμμα έχει σαν σκοπό να εισαγάγει την ιδέα της μετατροπής μίας FSM σε VHDL. Αυτή η FSM έχει τέσσερις καταστάσεις: A, B, C και D. Το σύστημα έχει ένα σήμα εισόδου που ονομάζεται P και η τιμή του P καθορίζει σε ποια κατάσταση θα μετακινηθεί το σύστημα μετά. Το σύστημα αλλάζει κατάσταση από τη μία κατάσταση (A) στην επόμενη (B,C,D) εφόσον η είσοδος P είναι υψηλή (P=1). Εάν το P είναι χαμηλό και το σύστημα

βρίσκεται στην κατάσταση A, B ή C, η κατάσταση δεν αλλάζει. Εάν το σύστημα βρίσκεται στην κατάσταση D, τότε αλλάζει σε B εάν το P είναι υψηλό και σε A εάν το P είναι χαμηλό. Το σύστημα έχει επίσης μια έξοδο που ονομάζεται R που είναι 1 εάν βρίσκεται στην κατάσταση D, διαφορετικά είναι 0.

Όσον αφορά τον σχεδιασμό του FSM πρέπει να τονιστεί το εξής: Όλες οι καταστάσεις του διαγράμματος πρέπει να περιλαμβάνονται στις δηλώσεις της νέας state_type κατά τον προγραμματισμό. Για τη μετάβαση από τη μία κατάσταση στην άλλη πρέπει να χρησιμοποιηθεί η εντολή WHEN μέσα σε ένα CASE.1*(Γκουντέλλης, 2019)* Η μηχανή πεπερασμένων καταστάσεων είναι μία συλλογή από καταστάσεις, αποτελούμενη από μία αρχική κατάσταση και μια συνάρτηση μετάβασης η οποία καθορίζει τη μετάβαση από τη μία κατάσταση στην άλλη. Ορίζεται, δηλαδή από τη συνάρτηση μετάβασης, το πώς θα γίνει η διαδικασία από την αρχική κατάσταση στις επόμενες. Οι FSM είναι μηχανές που αποκρίνονται σ' ένα σύνολο ερεθισμάτων (γεγονότων) παράγοντας προβλέψιμες απαντήσεις (ενέργειες) οι οποίες βασίζονται σε μια ιστορία των προγενέστερων γεγονότων (επικρατούσα κατάσταση). Η ''ιστορία'' αυτή της μηχανής συνοψίζεται στην τιμή των εσωτερικών καταστάσεων της. Δημιουργείται ένα σύνολο από καταστάσεις και μεταβάσεις για να μοντελοποιηθούν οι αλληλεπιδράσεις ανάμεσα στις εισόδους και εξόδους. Οι μηχανές πεπερασμένων καταστάσεων χρησιμοποιούνται για να εκφράσουν διαδικασίες απόφασης και χρησιμοποιούνται στην υλοποίηση της μονάδας ελέγχου για να απλοποιήσουν την υλοποίηση.

Η μηχανή πεπερασμένων καταστάσεων αποτελείται από καταστάσεις και βέλη που δείχνουν την κατεύθυνση στη μετάβαση των καταστάσεων. Οι κατευθύνσεις καθορίζονται από τη συνάρτηση της επόμενης κατάστασης (next-state function), η οποία εξαρτάται από την παρούσα κατάσταση και τις εισόδους της καινούριας κατάστασης. Κάθε κατάσταση καθορίζει επίσης ένα σύνολο εξόδων που είναι ενεργές όταν η μηχανή βρίσκεται σε αυτή την κατάσταση. Στο παρακάτω διάγραμμα φαίνεται ένα δομικό διάγραμμα μιας FSM. Όπως είναι εμφανές, στο κάτω μέρος είναι η διαδοχική λογική (flip-flops), ενώ στο πάνω μέρος είναι η συνδυαστική λογική. Στη συνδυαστική λογική υπάρχουν δύο είσοδοι, με τη μία να είναι η παρούσα κατάσταση (pr_state) και η δεύτερη είναι η κατάλληλη κατάσταση εισόδου προκειμένου να έχουμε την επιθυμητή έξοδο. Αντίστοιχα, υπάρχουν και δύο έξοδοι στην συνδυαστική λογική, με την μία να αφορά την κατάλληλη κατάσταση εξόδου και τη δεύτερη να αφορά την κατάσταση εξόδου που υπεισέρχεται στην ακολουθιακή λογική (nx_state). Στην ακολουθιακή λογική (κάτω μέρος) έχουμε τρείς εισόδους (clock, reset και nx_state) και μία έξοδο (pr_state). Εφόσον όλα τα κομμάτια είναι μέρος του συστήματος, το clock, που είναι το ρολόι, και το reset, το οποίο είναι απαραίτητο για το σήμα αναφοράς, πρέπει να είναι συνδεδεμένα με την ακολουθιακή λογική.



Εικόνα 7 Διάγραμμα μηχανής καταστάσεων Mealy (Moore)

Το να διαχωρίζεται το κύκλωμα σε δύο κομμάτια, όπως στο παραπάνω διάγραμμα, βοηθά στο να διαχωριστεί και ο σχεδιασμός αντίστοιχα σε κομμάτια. Από τη λογική της VHDL είναι φανερό ότι το κάτω μέρος της ακολουθιακής λογικής θα απαιτεί Διεργασία PROCESS, ενώ το πάνω μέρος, μιας και είναι συνδυασμός μπορεί επίσης να γίνει με εντολή PROCESS, υπό τη σκέψη ότι στα ακολουθιακά κυκλώματα εντάσσονται και οι δύο τύποι λογικής, συνδυαστική και ακολουθιακή. Το clock και το reset ως σήματα εμφανίζονται στα σήματα ευαισθησίας του κάτω μέρους της PROCESS. [21]*(Pedroni, 2004, 186)* Όταν εκτελείται το reset, η παρούσα κατάσταση (pr_state) ορίζεται ως η αρχική κατάσταση του συστήματος. Διαφορετικά, στην κατάλληλη τιμή του clock τα φλιπ-φλοπ θα αποθηκεύσουν την τιμή της nx_state, τοποθετώντας την στην έξοδο του κάτω μέρους.

Μία τυπική δομή κώδικα όσων περιγράφηκαν πιο πάνω είναι η ακόλουθη:

```
process (rst, clk)
begin
    if (rst='1') then
        present_state <= st0;
    elsif (clk'EVENT and clk='1') then
        present_state <= next_state;
    end if;
end process;
```

Εικόνα 8 κάτω τμήμα της FSM

3.2 Τεχνικές Κωδικοποίησης

Στη VHDL, οι Μηχανές Πεπερασμένων Καταστάσεων (FSM) μπορούν να γραφτούν με διάφορους τρόπους. Η κωδικοποίηση των καταστάσεων ενός FSM επηρεάζει την απόδοσή της μηχανής όσον αφορά την ταχύτητα, τη χρήση πόρων, την κατανάλωση ενέργειας ακόμη και τον τύπο προγραμματιζόμενης λογικής. Η προτιμώμενη κωδικοποίηση εξαρτάται κάθε φορά από το ζητούμενο του σχεδιασμού. [15]*(Meeus, 2020)*
Οι αλγόριθμοι κωδικοποίησης κατάστασης περιλαμβάνουν:

- Binary κωδικοποίηση: οι καταστάσεις απαριθμούνται με δυαδικούς κωδικοποιημένους αριθμούς.
- One-hot κωδικοποίηση: οι καταστάσεις αντιπροσωπεύονται ως μοτίβα bit με ακριβώς 1 '1'.
- Gray κωδικοποίηση: η κωδικοποίηση διαδοχικών καταστάσεων διαφέρει μόνο κατά ένα bit.

state	**BinaryCode**	**One-HotCode**	**GrayCode**
s0	000	00001	000
s1	001	00010	001
s2	010	00100	011
s3	011	01000	010
s4	100	10000	110

Πίνακας 1 Παράδειγμα Κωδικοποίησης για μηχανή 4 καταστάσεων

Binary κωδικοποίηση

Σε ένα δυαδικό σχήμα κωδικοποίησης, η σχέση μεταξύ του αριθμού των μεταβλητών καταστάσεων (q) και του αριθμού των καταστάσεων (n) δίνεται από την εξίσωση:

$$q=\log 2\ (n)$$

Με αυτόν τον τύπο, προσδιορίζεται ο ελάχιστος αριθμός μεταβλητών κατάστασης που απαιτούνται για μια FSM με δυαδική κωδικοποίηση. Επισημαίνεται πως ο αριθμός των flip-flops που χρησιμοποιούνται είναι ίσος με τον αριθμό των μεταβλητών κατάστασης (q). Σε αυτή την τεχνική, οι καταστάσεις εκχωρούνται με δυαδική ακολουθία όπου οι καταστάσεις αριθμούνται ξεκινώντας από το δυαδικό 0 και πάνω. Η δυαδική κωδικοποίηση είναι συνήθως η προτιμώμενη μέθοδος όταν οι μηχανές υλοποίησης έχουν λιγότερες από 8 καταστάσεις. Η ευρεία «AND» αρχιτεκτονική τους επιτρέπει οποιονδήποτε αριθμό μεταβλητή κατάστασης (bit) που θα συμπεριληφθεί σε κάθε όρο προϊόντος χωρίς ποινή ταχύτητας (ή περιοχής). Τα μειονεκτήματα της χρήσης του binary encoding FSM περιλαμβάνουν το γεγονός ότι περισσότερα από ένα bit μπορεί να αναστραφούν σε οποιαδήποτε στιγμή και να οδηγήσουν σε glitch τον σχεδιασμό των μετρητών(counters). Απαιτεί επίσης μία πιο σύνθετη λογική κωδικοποίησης για τον προσδιορισμό της παρούσας κατάστασης. Επιπλέον, η μέθοδος είναι ακατάλληλη εάν το ζητούμενο είναι η εξισορρόπηση ενέργειας, καθώς όσο περισσότερο μεταβάλλονται οι καταχωρητές/flip-flop, τόσο περισσότερη ενέργεια καταναλώνει η συσκευή. [7]*(csun.edu, n.d.)*

One-hot κωδικοποίηση

Στην κωδικοποίηση one-hot μόνο ένα bit της μεταβλητής κατάστασης είναι "1" ή "hot" για οποιαδήποτε δεδομένη κατάσταση. Όλα άλλα bit κατάστασης είναι μηδέν. (όπως φαίνεται στον πίνακα 1) Επομένως, χρησιμοποιείται ένα flip-flop (register) για κάθε κατάσταση στη μηχανή, δηλαδή για n αριθμό καταστάσεων χρησιμοποιούνται n flip-flops. Χρησιμοποιώντας κωδικοποίηση one-hot, μπορούν να εξαχθούν οι εξισώσεις επόμενης κατάστασης από τα διαγράμματα κατάστασης. Η κωδικοποίηση One-hot απλοποιεί τις διαδικασίες μετάβασης και έτσι μπορούμε να δημιουργήσουμε τις εξόδους FSM και την επόμενη κατάσταση πιο γρήγορα. Τέλος,

είναι εύκολη στην σχεδίαση, προτείνετε για μεγάλα FSM και στην ανάθεση των καταστάσεων καταναλώνετε λιγότερη ενέργεια λόγω του ότι χρησιμοποιεί λιγότερες λογικές πύλες.

Gray κωδικοποίηση

Όπως συζητήθηκε παραπάνω, η Grey κωδικοποίηση μπορεί να χρησιμοποιηθεί για την επίτευξη σχεδίασης χαμηλότερης ισχύος και την μείωση των Glitch. Ένα επιπλέον πλεονέκτημα της κωδικοποίησης αυτής είναι η προστασία των ασύγχρονων εξόδων από δυσλειτουργίες. Η Gray κωδικοποίηση είναι μια εκχώρηση κατάστασης στην οποία οι κωδικοί διαδοχικών καταστάσεων διαφέρουν μόνο κατά ένα bit. Δηλαδή, κατά την αλλαγή διαδοχικών καταστάσεων, αλλάζει μόνο ένα flip- flop.Ο αριθμός των flip-flops που χρησιμοποιούνται καθορίζεται από τη παρακάτω σχέση:

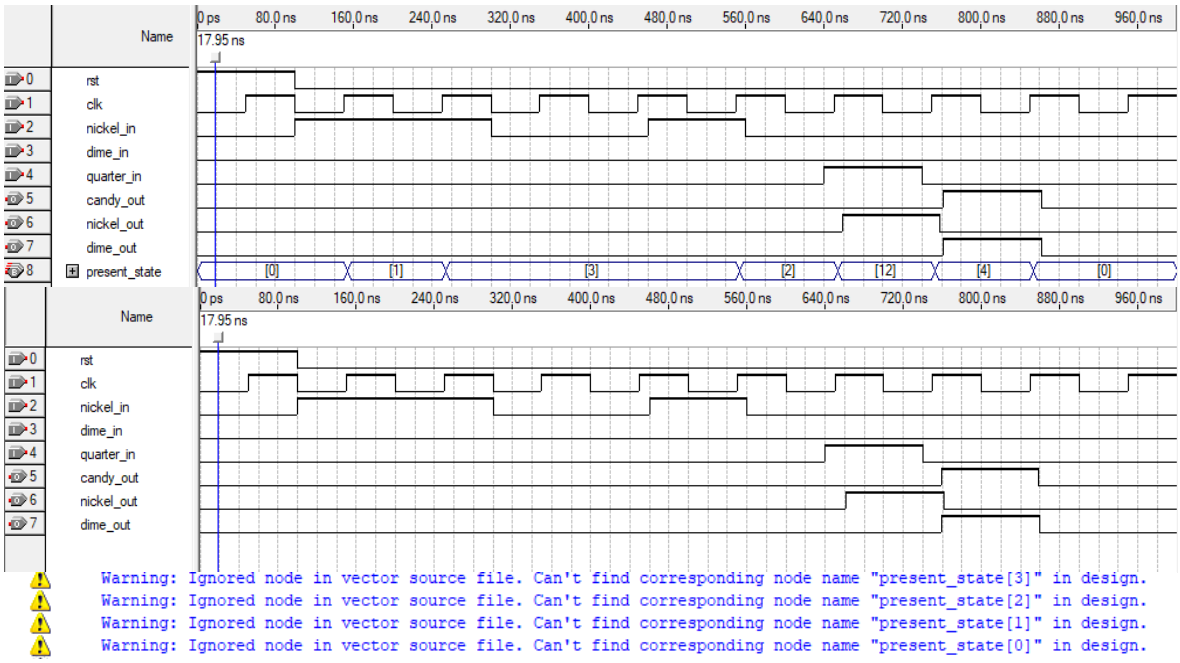
Αριθμός μεταβλητών κατάστασης = αριθμός flip-flops = log 2 (αριθμός καταστάσεων)

Η μέθοδος gray χρησιμοποιεί τον ίδιο αριθμό καταχωρητών (flip-flops) με τη δυαδική μέθοδο κωδικοποίησης. Η αποκωδικοποίηση της μπορεί να είναι εξίσου περίπλοκη, αν όχι περισσότερο. Συνοψίζοντας, με την κωδικοποίηση Gray, μόνο ένα bit αλλάζει όταν γίνεται μετάβαση μεταξύ γειτονικών καταστάσεων. Ως αποτέλεσμα, αυτή η τεχνική κωδικοποίησης μπορεί να μειώσει την κατανάλωση ενέργειας ενός FSM. Ωστόσο, η κωδικοποίηση Gray καθιστά τις ασύγχρονες εξόδους ενός FSM ανθεκτικές σε δυσλειτουργίες και προβλήματα.

Σε έναν κώδικα VHDL οι τεχνικές κωδικοποίησης ορίζονται και δηλώνονται μετά την δήλωση των καταστάσεων στην αρχή της αρχιτεκτονικής . Βοηθάνε σε μια πιο συγκεκριμένη κωδικοποίηση και εκπροσωπούν τον τύπο των καταστάσεων ενός FSM, με σκοπό να απλοποιήσει τον κώδικα, παρόλο που μπορεί να δημιουργηθούν προβλήματα σε μεγαλύτερα και πιο περίπλοκα projects και πιο πρακτικά στην σχεδίαση των καταστάσεων. [11]*(IntelCorporation, n.d.)* Η VHDL χρησιμοποιεί την σύνθεση των “attribute” για να καθορίζει κωδικοποιήσεις, η οποίες διαμορφώνονται από έναν τύπο απαρίθμησης (enum_encoding, εικόνα 9). Το “attribute” παίρνει μια “string” τιμή που αποτελείται από μία τεχνική κωδικοποίησης. Στην εικόνα 10 φαίνεται ένα παράδειγμα κυματομορφής από ένα απλό FSM 10 καταστάσεων, όταν έχει οριστεί τεχνική κωδικοποίησης gray και όταν δεν έχει οριστεί.

```
10  architecture fsm of vending_machine is
11      type state is (st0, st5, st10, st15, st20, st25,
12          st30, st35, st40, st45);
13      signal present_state, next_state : state;
14      attribute enum_encoding : string; --optional attribute
15      attribute enum_encoding of state : type is "gray";
```

Εικόνα 9 Ορισμός των attribute



Εικόνα 10 Παράδειγμα με Χρήση των attribute

3.3 Σύγχρονες – Ασύγχρονες

Τα ψηφιακά κυκλώματα τα οποία αποτελούνται από συνδυαστική και ακολουθιακή λογική ταυτόχρονα, περιγράφονται ως μηχανές πεπερασμένης κατάστασης (FSM), όπως είδαμε και από την αρχή του παρόντος κεφαλαίου. Μια μηχανή ονομάζεται σύγχρονη όταν οι μεταβάσεις από τη μία κατάσταση στην άλλη ελέγχονται ή συγχρονίζονται από ένα σήμα ρολογιού (clock). Όταν η λειτουργία μιας μηχανής δεν εξαρτάται από το σήμα ρολογιού, τότε ονομάζεται ασύγχρονη. Η παρούσα κατάσταση (present_state) μιας μηχανής κατάστασης καθορίζεται από τις αποθηκευμένες μεταβλητές στα flip-flops της διαδοχικής ενότητας. Η επόμενη κατάσταση (next_state) της μηχανής κατάστασης ορίζεται από το κύκλωμα, στο τμήμα της συνδυαστικής λογικής. [18]*(Ndjountche, 2016, 213-214)*Για τον σχεδιασμό μιας σύγχρονης μηχανής πεπερασμένων καταστάσεων, χρησιμοποιούνται τα ακόλουθα βήματα:

1) Εξαγωγή διαγράμματος κατάστασης. Το διάγραμμα αυτό θα περιέχει όλες τις καταστάσεις της μηχανής και θα δίνει τις συνθήκες υπό τις οποίες το κύκλωμα θα μεταβαίνει από τη μία κατάσταση στην άλλη.

2) Σύνταξη του πίνακα κατάστασης.

3) Αντιστοίχιση των συνδυασμών των bit στις μεταβλητές για να αναπαρασταθούν οι διάφορες καταστάσεις και να συνταχθεί ο αντίστοιχος πίνακας καταστάσεων.

4) Επιλογή του τύπου των flip-flop που θα χρησιμοποιηθούν στο κύκλωμα.

5) Εξαγωγή εξισώσεων εισόδου με βάση τους χάρτες Karnaugh.

6) Αναπαράσταση του λογικού κυκλώματος που προκύπτει.

Όπως αναφέρθηκε ήδη, η λειτουργία των ασύγχρονων μηχανών κατάστασης, σε αντίθεση με τις σύγχρονες, δεν απαιτούν σήμα ρολογιού. Η μεταφορά δεδομένων ή ο συγχρονισμός των ασύγχρονων μηχανών πραγματοποιείται μέσω της αμφίδρομης ανταλλαγής σημάτων αιτήματος και της αναγνώριση σημάτων, που ονομάζεται «επικοινωνία χειραψίας».Οι ασύγχρονες μηχανές προσφέρουν τα πλεονέκτημα μιας απλούστερης λογικής ελέγχου και τηνπιο γρήγορη ανταπόκριση σε χρόνους αποκατάστασης από συνθήκες επαναφοράς. Ωστόσο, είναι πιο ευαίσθητα σε σφάλματα συγχρονισμού (όπως διάδοση καθυστερήσεων ή κινδύνου, ταλάντωση). Ως αποτέλεσμα, είναι πολύ πιο δύσκολο να σχεδιαστεί μία αξιόπιστη ασύγχρονη μηχανή κατάστασης. Στη βασική λειτουργία μπορεί να αλλάξει μία μόνο είσοδος ανά πάσα στιγμή και η μηχανή βρίσκεται σε σταθερή κατάσταση.

Σύγχρονο - Ασύγχρονο reset

Στον ψηφιακό σχεδιασμό, οι επαναφορές (reset) χρησιμοποιούνται για να φέρουν ένα κύκλωμα σε μια προκαθορισμένη κατάσταση μετά την ενεργοποίηση. Οι επαναφορές σχεδιάζονται σε σύγχρονα (χρονισμένα) μέρη του σχεδιασμού. Η επαναφορά (reset) είναι είτε ασύγχρονη είτε σύγχρονη. Μια ασύγχρονη επαναφορά ενεργοποιείται μόλις βεβαιωθεί το σήμα επαναφοράς. Ενώ μια σύγχρονη επαναφορά ενεργοποιείται στο τέλος του ενεργού ρολογιού όταν επιβεβαιωθεί το σήμα επαναφοράς. Η επιλογή μεταξύ σύγχρονης ή ασύγχρονης επαναφοράς εξαρτάται από τη φύση της λογικής που επαναφέρεται και τις απαιτήσεις του project.

Τα πλεονεκτήματα και τα μειονεκτήματα της σύγχρονης επαναφοράς περιλαμβάνουν:

- Οι σύγχρονες επαναφορές είναι προβλέψιμες (στην άκρη του ρολογιού)
- Οι σύγχρονες επαναφορές είναι ισχυρές π.χ. κατά των δυσλειτουργιών
- Στην τεχνολογία ASIC, μπορούν να χρησιμοποιηθούν μικρότερα flip- flops
- αλλά η επαναφορά υλοποιείται σε μία επιπλέον λογική ακολουθία, η οποία μπορεί να προσθέσει καθυστέρηση
- Το χρονοδιάγραμμα κλεισίματος ενός μεγάλου reset μπορεί να είναι δύσκολο

Τα πλεονεκτήματα και τα μειονεκτήματα της ασύγχρονης επαναφοράς περιλαμβάνουν:

- Η επαναφορά μπορεί να συμβεί όταν το ρολόι δεν λειτουργεί, π.χ. σε πρώιμα στάδια εκκίνησης του κυκλώματος ή όταν χρησιμοποιείται πύλη ρολογιού
- Το κύκλωμα επαναφοράς δεν αποτελεί μέρος της διαδρομής δεδομένων και μπορεί να δημιουργήσει καθυστέρηση
- Σφάλματα στο δίκτυο επαναφοράς μπορεί να οδηγήσουν σε άκυρες επαναφορές
- Πρέπει να διασφαλιστεί ότι η απενεργοποίηση της επαναφοράς δεν συμβαίνει πάνω ή κοντά σε μια άκρη του ρολογιού. [15]*(Meeus, 2020)*

Συμπερασματικά, συνιστώνται οι σύγχρονες επαναφορές εκτός εάν το συγκεκριμένο κύκλωμα απαιτεί ασύγχρονη επαναφορά. Η επιλογή μπορεί να εξαρτάται από την τεχνολογία που χρησιμοποιείται, π.χ. ορισμένα μπλοκ FPGA ενδέχεται να υποστηρίζουν μόνο μια σύγχρονη επαναφορά.

```
p_synchronous_reset : process (clk) is
begin
   if rising_edge(clk) then
      if rst = '1' then              -- do reset
         q <= '0';
      else                           -- normal operation
         q <= d;
      end if;
   end if;
end process p_synchronous_reset;
```

Εικόνα 11 Σύγχρονο Reset

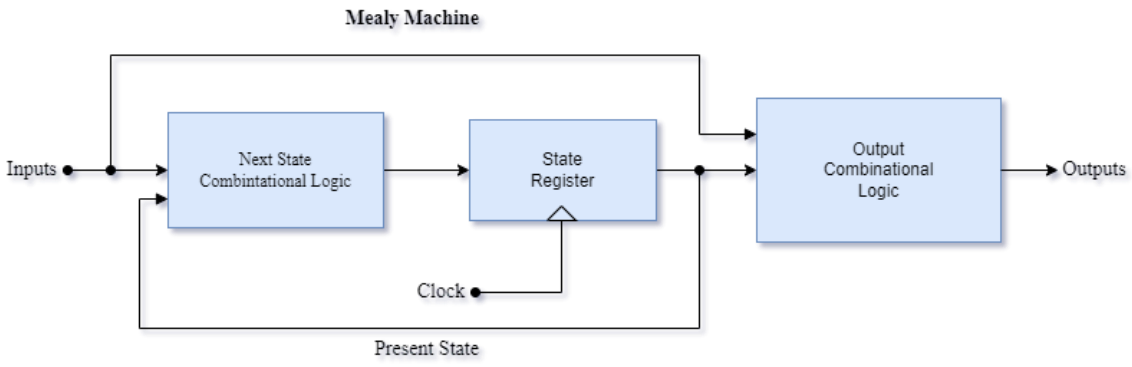
```
p_asynchronous_reset : process(clk, rst) is
begin
   if rst = '1' then                 -- do reset
      q <= '0';
   elsif rising_edge(clk) then       -- normal operation
      q <= d;
   end if;
end process p_asynchronous_reset;
```

Εικόνα 12 Ασύγχρονο Reset

3.4 Moore – Mealy

Οι μηχανές πεπερασμένων καταστάσεων, όπως ήδη αναφέρθηκε, χωρίζονται στους τύπους Moore και Mealy. Πολύ συνοπτικά στους τύπους FSM Mealy, οι έξοδοι εξαρτώνται τόσο από τις εισόδους όπως και από τις ενδιάμεσες καταστάσεις, ενώ στις μηχανές Moore οι έξοδοι εξαρτώνται μόνο από την παρούσα κατάσταση. Με άλλα λόγια, η έξοδος δεν επηρεάζεται άμεσα από την είσοδο (η είσοδος μπορεί να επηρεάσει μόνο την επόμενη κατάσταση του μηχανήματος).

Παρακάτω παρουσιάζεται διαγραμματικά μία μηχανή τύπου Mealy:

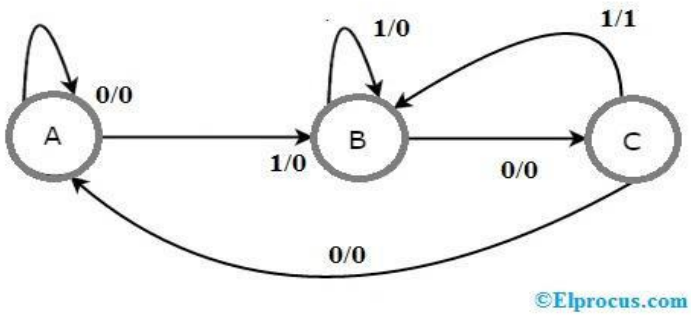


Εικόνα 13 Μηχανή Mealy

Όπως είναι εμφανές, μία Mealy FSM διαχωρίζεται σε δύο τμήματα: Την συνδυαστική λογική των καταστάσεων και τις ταυτόχρονες δηλώσεις της εξόδου. Το StateRegisterbox ως κομμάτι της μηχανής μπορεί να τροφοδοτήσει το τμήμα της λογικής με προηγούμενες εισόδους. Οι έξοδοι είναι συναρτήσεις και των εισόδων και των καταστάσεων και κάθε έξοδος είναι αποτέλεσμα του αντίστοιχου συνδυασμού εισόδου και κατάστασης. Έτσι, οι έξοδοι μπορούν να είναι χρήσιμα σαν σήματα μόνο όταν έχουν θετικό πρόσημο. [8]*(ElProCus, n.d.)*Για την επεξήγηση των μηχανών αυτών θα χρησιμοποιηθούν διαγράμματα καταστάσεων (statetransitiondiagram). Τα διαγράμματα αυτά αποτελούν την γραφική απεικόνιση των ακολουθιών και των λειτουργιών μιας μηχανής και θα πρέπει υποχρεωτικά να έχουν τα εξής τρία χαρακτηριστικά:

1. Να περιλαμβάνουν όλες τις πιθανές καταστάσεις
2. Να περιλαμβάνουν όλες τις συνθήκες μετάβασης από τη μία κατάσταση στην άλλη
3. Τα σήματα εξόδου πρέπει να είναι τα ίδια σε κάθε κατάσταση.

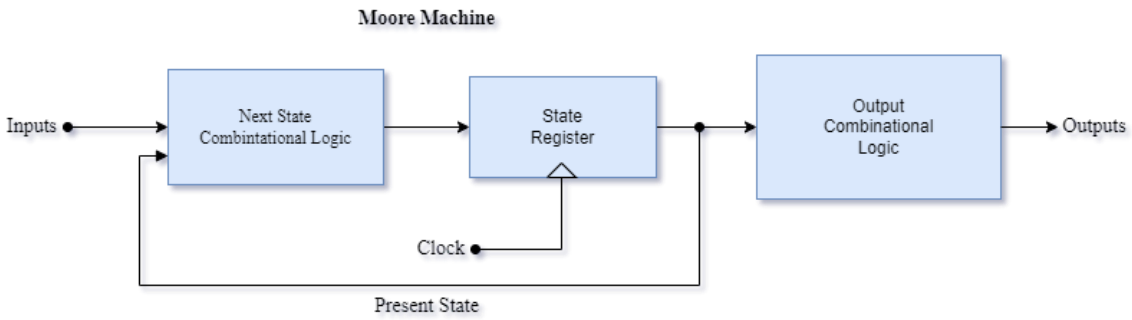
Παρακάτω φαίνεται το διάγραμμα κατάστασης μιας μηχανής Mealy:



Εικόνα 14 Διάγραμμα μηχανής Mealy

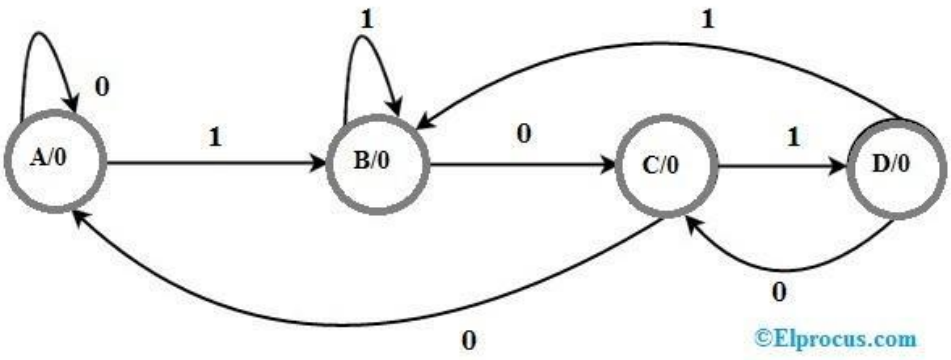
Όπως επεξηγείται στο διάγραμμα, οι τρεις πιθανές καταστάσεις μίας μηχανής Mealy είναι οι A, B και C ή 0/0, 1/0 και 1/1. Όλες οι καταστάσεις επικοινωνούν μεταξύ τους, και ανάλογα με την είσοδο αλλά και την τωρινή κατάσταση (A, B, C) γίνεται η μετάβαση στην επόμενη κατάσταση. Στον ίδιο χρόνο, η παραγωγή μίας ασύγχρονης εξόδου αλλάζει την κατάστασή της σε σύγχρονη, μιας και οι τιμές της εξόδου μίας μηχανής Mealy επηρεάζονται τόσο από την τρέχουσα κατάσταση όσο και από τις εισόδους. Ως κανόνας, ισχύει ότι μπορεί να χρησιμοποιηθεί είτε μία Moore είτε μία Mealy μηχανή, ανάλογα με τις ανάγκες του χρήστη. [22](Pedroni, 2013, 4-6)

Το παρακάτω διάγραμμα επεξηγεί μία FSM Moore:



Εικόνα 15 Μηχανή Moore

Τα αποτελέσματα/έξοδοι εξαρτώνται μόνο από τις τωρινές καταστάσεις, επομένως στο διάγραμμα βλέπουμε ότι οι έξοδοι αυτής της μηχανής θα παραχθούν μετά τη μετατροπή των καταστάσεων. Η μετατροπή αυτή θα γίνει μόνο στην θετική ακμή του ρολογιού όπως αναφέρθηκε και παραπάνω. Στο αντίστοιχο διάγραμμα καταστάσεων μιας μηχανής τύπου Moore, έχουμε 4 καταστάσεις A, B, C και D, με τις αντίστοιχες εξόδους τους. Σε κάθε κατάσταση, ανάλογα με την είσοδο, έχουμε δύο διαφορετικά αποτελέσματα, τα οποία οδηγούν τη μηχανή στην εκάστοτε κατάσταση κάθε φορά. Σε αυτού του τύπου τις μηχανές, υπάρχουν περισσότερες ή ίσες σε αριθμό καταστάσεις με την εφάμιλλη μηχανή Mealy.



Εικόνα 16 Διάγραμμα μηχανής Moore

3.5 Διαφορές

Όπως αναφέρθηκε και παραπάνω, η επιλογή ανάμεσα σε μία μηχανή τύπου Moore ή Mealy εξαρτάται από τα ζητούμενα που έχει ο εκάστοτε σχεδιαστής από τη μηχανή του, καθώς υπάρχουν εμφανής διαφορές ανάμεσα σε αυτούς τους δύο τύπους. Μία διαφορά, αρχικά, είναι τα σήματα εξόδου στις μηχανές. Μία μηχανή Moore έχει εξόδους που εξαρτώνται μόνο από την κατάσταση του κυκλώματος, ενώ σε μία μηχανή Mealy η έξοδος εξαρτάται τόσο από την κατάσταση όσο και από τις εισόδους του κυκλώματος. Πρακτικά, αυτό σημαίνει ότι τα σήματα μίας μηχανής Moore αλλάζουν όταν το σήμα του ρολογιού αλλάξει την κατάσταση της μηχανής, ενώ στις μηχανές Mealy, τα σήματα εξόδου μπορεί να αλλάξουν ανά πάσα στιγμή μετά από ένα σήμα εισόδου. Όσον αφορά την θεωρητική επιστήμη των υπολογιστών, οι δύο τύποι FSM θεωρούνται παρόμοιοι, καθώς και οι δύο αναγνωρίζουν «κανονικές εκφράσεις». Ωστόσο, όταν η μηχανή πρόκειται να χρησιμοποιηθεί ως κύκλωμα ελέγχου, τα αποτελέσματα που έχει μία μηχανή Moore έναντι μίας μηχανής Mealy είναι κρίσιμα για το χρονισμό του ελέγχου του κυκλώματος. 5*(Chiuchisanetal., 2010)*

Υπάρχουν τρεις μεγάλες διαφορές μεταξύ των δύο τύπων μηχανών. Πρώτον, σε μια μηχανή Mealy χρειάζονται λιγότερες καταστάσεις να εκτελούν την ίδια εργασία επειδή οι έξοδοι είναι συνάρτηση των καταστάσεων και των εισόδων και, επομένως, πολλές πιθανές τιμές εξόδου μπορούν να καθοριστούν σε μία κατάσταση. Δεύτερον, μία μηχανή Mealy παράγει πιο γρήγορα αποτελέσματα. Δεδομένου ότι μια έξοδος της μηχανής Mealy είναι συνάρτηση και της εισόδου, η έξοδος αντιδρά αμέσως κάθε φορά που η είσοδος εκπληρώνει την καθορισμένη κατάσταση.Αντίθετα μια μηχανή Mooreανταποκρίνεται με έμμεσο τρόπο στις αλλαγές της εισόδου. Η τρίτη διαφορά αφορά τον έλεγχο του πλάτους και του χρονισμού του σήματος εξόδου. Σε μια μηχανή Mealy, το πλάτος ενός σήματος εξόδου ποικίλλει ανάλογα με την είσοδο και μπορεί να είναι πολύ «στενό». Μία μηχανή

Mealy είναι ευαίσθητη στις διαταραχές του σήματος εισόδου και αυτό αντανακλάται στην έξοδο. Η έξοδος μιας μηχανής Moore συγχρονίζεται με το σήμα του ρολογιού και το πλάτος του είναι περίπου το ίδιο με την περίοδο του ρολογιού. [25]*(Thakur, 2023)* Συμπερασματικά, η επιλογή τύπου μηχανής από την οπτική της δημιουργίας μίας μηχανής ελέγχου γίνεται αν διαχωριστούν τα σήματα ελέγχου σε edge-sensitive και level-sensitive, όπου στην πρώτη περίπτωση προτιμάται η μηχανή Mealy καθώς δημιουργεί λιγότερες καταστάσεις σε λιγότερο χρόνο, ενώ στη δεύτερη περίπτωση η κατάλληλη μηχανή είναι η τύπου Moore καθώς μπορεί να απομονώσει το σήμα εξόδου από παρεμβολές των εισόδων. Οι μηχανές Mealy είναι προτιμότερες για σύγχρονα συστήματα που απαιτούν σύστημα «χωρίς καθυστέρηση και χωρίς σφάλματα», αλλά απαιτείται προσεκτικός σχεδιασμός για τα ασύγχρονα συστήματα. Επομένως, η μηχανή Mealy μπορεί να είναι πιο πολύπλοκη σε σύγκριση με τη μηχανή Moore. Στα πλαίσια του πρακτικού κομματιού της πτυχιακής εργασίας χρησιμοποιήθηκε moore τύπου προσέγγιση, πρώτων λόγο του μεγάλου αριθμού των καταστάσεων που απαιτούσε το project και δεύτερων διότι τα αποτελέσματα των εξόδων εξαρτώνται τόσο από τις καταστάσεις όσο και την θετική ακμή του ρολογιού.

4 Σχετικά Projects

4.1 Πως λειτουργούν components και packages

Τα τμήματα του κώδικα στη VHDL που χρησιμοποιούνται συχνά μπορούν να οργανωθούν σε: τμήματα, συναρτήσεις και πακέτα. Τα τμήματα αυτά βρίσκονται στον κεντρικό κορμό του κώδικα, ωστόσο τα πακέτα ολοκληρώνονται στη βιβλιοθήκη του κώδικα όπου γίνεται και το compile τους. Ένα πακέτο (Package) χρησιμοποιείται για να οργανώσει τις δηλώσεις και τις περιγραφές των τύπων, των υπό-τύπων, των σταθερών τύπων, των περιεχομένων ή των υπό-προγραμμάτων ενός κυκλώματος προκειμένου να αποθηκευτούν στη βιβλιοθήκη (library). Η βιβλιοθήκη μπορεί να αποθηκεύσει περισσότερα από ένα πακέτα. Έτσι, αρχιτεκτονικά, ο κώδικας αποτελείται από τις αρχικές δηλώσεις που προέρχονται από τη βιβλιοθήκη, την ενότητα (entity), την αρχιτεκτονική δομή του κώδικα και τελικά την παραμετροποίηση. Σαν πακέτο μπορεί να δηλωθεί οτιδήποτε από τα παρακάτω:

- Τύπος (Type)
- Σταθερά (Constant)
- Σήμα (Signal)
- Συνάρτηση (Function)
- Διαδικασία (Procedure)

Με λίγα λόγια η δομή ενός πακέτου αποτελείται από τον ορισμό του Package με τις δηλώσεις του (μεταβλητών, τύπων, κλπ.) καθώς και το Package Body, το οποίο θα εμπεριέχει τις "περιγραφές" των υπό-προγραμμάτων. Το πακέτο και το body του πακέτου πρέπει να έχουν το ίδιο όνομα. Για να γίνει το πακέτο μέρος της βιβλιοθήκης θα πρέπει πρώτα να μεταγλωττιστεί και ύστερα να προστεθεί με την χρήση της εντολής USE στον κυρίως κώδικα.

```
-----------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
-----------------------------------
ENTITY...
...
ARCHITECTURE...
...
-----------------------------------
```

Εικόνα 17 Παράδειγμα κώδικα για Packages

Η δήλωση δομικού στοιχείου (component) είναι ένα κομμάτι συμβατικού κώδικα αποτελούμενο από τις δηλώσεις βιβλιοθήκης, την ενότητα και την αρχιτεκτονική. Ωστόσο, δηλώνοντας ένα τμήμα ως δομικό στοιχείο, τότε μπορεί αυτό να χρησιμοποιηθεί και μέσα σε κάποιο άλλο κύκλωμα, δίνοντας τη δυνατότητα ιεραρχικού σχεδιασμού. Για να χρησιμοποιηθεί ένα δομικό στοιχείο πρέπει αυτό αρχικά να έχει δηλωθεί, και να είναι στιγμιαίο. Τα components μπορούν να δηλωθούν είτε στον κύριο κορμό του κώδικα είτε μέσα σε ένα πακέτο. Υπάρχουν δύο τρόποι να χαρτογραφηθούν οι πύλες (ports) ενός δομικού στοιχείου. Η χαρτογράφηση κατά τη θέση και η χαρτογράφηση κατά όνομα. Οι πύλες μπορούν και να μη συνδεθούν με κάτι, χρησιμοποιώντας την εντολή OPEN. [9]*(GonzanaUniversity, n.d., 14)* Ακολουθεί παράδειγμα:

```
COMPONENT inverter IS
   PORT (a: IN STD_LOGIC; b: OUT STD_LOGIC);
END COMPONENT;
...
U1: inverter PORT MAP (x, y); -- positional mapping
...
U2: inverter PORT MAP (x => a, y =>b);  -- nominal mapping
```

Εικόνα 18 Παράδειγμα κώδικα για Components

Η στιγμιαία παρουσίαση ενός δομικού στοιχείου component είναι μια δήλωση που αναφέρεται, ουσιαστικά, στο στοιχείο που βρίσκεται στο χαμηλότερο επίπεδο στη σχεδίαση, με την έννοια ότι δημιουργείται ένα μοναδικό αντίγραφο (ή instance) αυτού του στοιχείου Component. Μια δήλωση παρουσίασης στοιχείου είναι ταυτόχρονη δήλωση, επομένως δεν έχει σημασία η σειρά με την οποία αναφέρονται τα στοιχεία. Πρέπει, ωστόσο, να δηλωθούν τυχόν στοιχεία τα οποία αναφέρονται είτε στην περιοχή δηλώσεων της αρχιτεκτονικής (πριν από το BEGIN) ή σε ένα εξωτερικό πακέτο που είναι ορατό στην αρχιτεκτονική. [6]*(csun.edu, n.d., 7)*

4.2 Γιατί χρησιμοποιούνται τα Testbenches

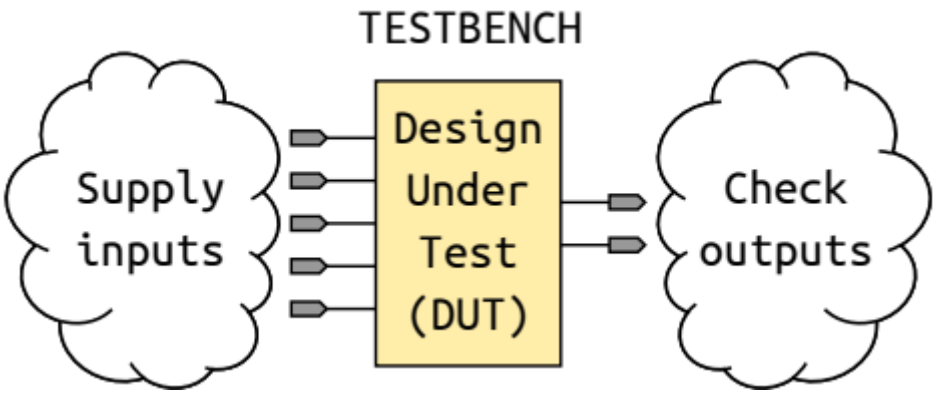
Κάθε φορά που σχεδιάζουμε ένα κύκλωμα ή ένα σύστημα, ένα από τα σημαντικότερα βήματα είναι η προσομοίωση, είτε παρέχοντας σ' αυτήν τα σήματα χειροκίνητα, είτε φτιάχνοντας ένα testbench. Γράφουμε ένα testbench για να εισάγουμε εισόδους (Stimulus) στην υπό δοκιμή μονάδα(DUT) και για να διαβάζουμε την έξοδό της. Επομένως, αν ο αριθμός των σημάτων εισόδου είναι πολύ μεγάλος ή/και πρέπει να εκτελέσουμε την προσομοίωση αρκετές φορές, τότε αυτή η διαδικασία μπορεί να είναι αρκετά πολύπλοκη, χρονοβόρα και ενοχλητική. Σε τέτοιες περιπτώσεις, απαιτούνται τα μοντέλα δοκιμής ώστε να επαληθευτεί εάν το σχεδιασμένο σύστημα λειτουργεί όπως αναμένεται ή όχι πριν το τοποθετήσουμε στο πραγματικό υλικό, π.χ. σε ένα FPGA. [13]*(Joshi, 2020)* Το testbench μπορεί να διαβάσει τα δοκιμαστικά σήματα από ένα αρχείο και να τα εφαρμόσει στο μοντέλο υπό δοκιμή(DUT). Στη συνέχεια, τα σήματα εξόδου ανακτώνται για ανάλυση. Αυτός είναι και ο βασικός ρόλος των testbenches που τα κάνει απαραίτητα κατά τη δημιουργία ενός κυκλώματος. Τα testbenches υφίστανται αξιολόγηση με τη χρήση του ουσιαστικότερου τρόπου δοκιμής, προσαρτώντας ακόμη ένα λογισμικό, γνωστό ως Modelsim. Το Modelsim ανήκει στην κατηγορία των προγραμμάτων επαλήθευσης και προσομοίωσης και αποτελεί ιδιαίτερα ισχυρό εργαλείο για την ανάπτυξη και επαλήθευση υλικού που περιγράφεται σε γλώσσες όπως η VHDL. Το ModelSim επίσης διευκολύνει τη διαδικασία εύρεσης ελαττωμάτων στο σχεδιασμό, αφού διαθέτει ένα έξυπνα σχεδιασμένο περιβάλλον εντοπισμού σφαλμάτων, που εμφανίζει αποτελεσματικά δεδομένα του σχεδιασμού για ανάλυση και εντοπισμό σφαλμάτων όλων των γλωσσών περιγραφής υλικού. Επιπλέον, τα testbenches παράγουν αποτελέσματα στην μορφή αρχείων csv (comma separated files), τα οποία μπορούν να διαβαστούν σε πληθώρα λογισμικών για περαιτέρω επεξεργασία, όπως Python, Excel, Matlab κλπ. Ακόμη, η μορφή αυτή του αρχείου, επιτρέπει σε ένα testbench να διαβάζει ένα αρχείο που έχει δημιουργηθεί παραδείγματος χάριν στην Matlab και να εξάγει σήματα ή τιμές. Για απλούστερα κυκλώματα, μπορούν χειροκίνητα να παραληφθούν κάποιες τιμές εισόδου, δημιουργώντας τες στο waveform, όπως εφαρμόστηκε στην παρούσα εργασία μέσω του προγράμματος Quartus II. [20]*(Patel, 2017)*

4.3 Projects που χρησιμοποιούν μοντέλα δοκιμής (Testbenches)

Τα μοντέλα δοκιμής (testbenches) είναι μέθοδοι με τις οποίες ένα μοντέλο VHDL επιτρέπει στον χρήστη να προσομοιώσει μία άλλη ενότητα. Οι μονάδες RTL εξαρτώνται από εξωτερικά σήματα για να λειτουργήσουν, είτε πρόκειται για σήματα εισόδου από άλλη μονάδα είτε σήματα ρολογιού. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, το testbench είναι μία αυτόνομη μονάδα δοκιμής χωρίς σήματα εισόδου ή εξόδου. Περιέχει, αντιθέτως, τοπικά σήματα, την μονάδα DUT(DesignUnderTest) και stimuli στις οποίες με κώδικα δημιουργούνται σήματα εισόδου.[12]*(JensenTech, n.d.)* Αυτό αποτελεί και τη βάση της VHDL, η οποία αποτελείται από μεγάλα τμήματα που προορίζονται μόνο για προσομοίωση. Για να χρησιμοποιηθούν όλες οι δυνατότητες του testbench ενός VHDL κώδικα, καθώς και για την βέλτιστη αποτελεσματικότητα τους, η χρήση του open-source προγράμματος Modelsim απαιτείται. Ωστόσο, η ενσωμάτωση του Modelsim μπορεί να δημιουργήσει αρκετά προβλήματα στην κατάσταση δοκιμής (DUT) και κατά την προσομοίωση. Το βασικότερο πρόβλημα είναι ότι τα σήματα φεύγουν από τον τομέα της VHDL, επομένως, θα πρέπει να γίνει επικοινωνία μαζί τους χρησιμοποιώντας διαφορετική γλώσσα κώδικα, όπως παραδείγματος χάριν είναι η TCL. Για να αποφευχθεί αυτό το πρόβλημα, μπορεί να δημιουργηθεί μία ξεχωριστή μονάδα δοκιμής (Testbench) VHDL μόνο για τον προσομοιωτή. Επειδή το testbench βρίσκεται στο υψηλότερο επίπεδο ιεραρχίας της σχεδίασης, θεωρείται ευέλικτο ως προς την παραμετροποίηση του καθώς εκτός από τις βασικές δυνατότητες

επαλήθευσης που περιέχει, μπορεί να επιστρέφει και μηνύματα τυχόν λαθών, αλλά και να διαβάζει δεδομένα μέσω ενός αρχείου TextIO. Παρ ‘όλα αυτά το ίδιο το testbench μπορεί να περιέχει σφάλματα ή να είναι ελλιπές, οδηγώντας σε εσφαλμένα αποτελέσματα. Επομένως, είναι ζωτικής σημασίας η επαλήθευση του ίδιου του testbench και η διασφάλιση ότι είναι αρκετά ολοκληρωμένο ώστε να ανιχνεύει πιθανά σφάλματα στη σχεδίαση. Υπάρχουν πολλοί τρόποι ανάπτυξης ενός testbench όσον αφορά την αρχιτεκτονική του δομή, η οποία διαφέρει ανάλογα την χρήση και την εφαρμογή. [14]*(Kashani-Akhavan, 2017)* Ένας από τους οποίους φαίνεται στο παρακάτω σχήμα() και αποτελείται από τρία 3 μέρη:

1. Το στοιχείο που πρόκειται να μπει υπό δοκιμή, δηλαδή το DesignUnderTest (DUT).
2. Ένας μηχανισμός για την παροχή εισόδων στο DUT.
3. Ένας μηχανισμός για τον έλεγχο των εξόδων του DUT έναντι των αναμενόμενων εξόδων.



Εικόνα 19 TestbenchDUT

Το ακόλουθο σχήμα δείχνει τη γενική αρχιτεκτονική του testbench. Άλλοι μέθοδοι σχεδίασης ενός Testbenchπεριγράφονται παρακάτω:

1. Self-Checking Testbench: Είναι ένα αυτοματοποιημένο πρόγραμμα δοκιμών, το οποίο εκτελεί μια ακολουθία δοκιμών και στο τέλος εκτυπώνει "OK" ή "Not OK". Η βασική του χρήση είναι να επαληθεύει ότι όλες οι ενότητες θα έχουν την προβλεπόμενη συμπεριφορά ανά πάσα στιγμή.
2. StimulusOnly Testbench: Περιέχει μόνο τον driver του stimulus και το DUT και δεν περιέχει καμία επαλήθευση αποτελεσμάτων. Αυτός ο τύπος testbench είναι χρήσιμος στην αρχή ενός προγράμματος σχεδίασης όταν δεν υπάρχουν γνωστά καλά διανύσματα ή για έναν γρήγορο έλεγχο μιας οντότητας.
3. Full Testbench: Περιέχει τον driver stimulus, γνωστά καλά αποτελέσματα και σύγκριση αποτελεσμάτων. Που σημαίνει ότι έχει και την δυνατότητα να ελέγχει την έξοδο του DUT. Αυτό που το κάνει να διαφέρει είναι ότι μπορεί να εκτελεί μια λειτουργία σύγκρισης μεταξύ των τιμών εξόδου από το DUT και των τιμών που έχουν διαβαστεί από το αρχείο.
4. Simulator specific Testbench: Είναι ένα Testbench γραμμένο σε ειδική για τον προσομοιωτή μορφή. Επιτρέπει στον σχεδιαστή να μεταγλωττίζει και να φορτώνει σχέδια, να δημιουργεί βιβλιοθήκες, να ορίζει σημεία διακοπής, να εκτελεί την προσομοίωση και πολλά ακόμη χρησιμοποιώντας τη γλώσσα εντολών του προσομοιωτή. [23](Perry, 2002, 330-345)

Τα testbenches έχουν το χαρακτηριστικό ότι, εφόσον γράφονται σε VHDL, είναι προσιτά από περισσότερα από ένα εργαλεία προσομοίωσης. Επιπλέον, μπορούν να προσαρμοστούν εύκολα σε

διάφορες εφαρμογές με τον ίδιο σχεδιασμό. Παρακάτω παρουσιάζεται η αρχική κωδικοποίηση ενός Testbench για ένα απλό vending machine με Process:

Όπως αναφέρθηκε και παραπάνω ένα Testbench είναι και αυτό ένα πρόγραμμα ή αρχείο VHDL, ακολουθεί δηλαδή τους κανόνες της γλώσσας VHDL. Γι' Αυτό, λοιπόν αρχικά ορίζεται η βιβλιοθήκη, τα κατάλληλα πακέτα και μια Entity χωρίς δηλώσεις. Πριν από το Begin πρέπει να δηλωθεί η δομή του component DUT, όπως επίσης τα τοπικά σήματα και οι σταθερές. Στο συγκεκριμένο Testbench ορίζονται τα τοπικά σήματα εισόδου, τα σήματα εξόδου και η χρονική περίοδος του ρολογιού στα 10 ns.

```
1   Library ieee;
2   Use ieee.std_logic_1164.all;
3
4   Entity tb is
5   end tb;
6
7   Architecture behavior of tb is
8
9       Component vending_machine
10      port(
11          clk : IN std_logic;
12          rst : IN std_logic;
13          nickel_in : IN std_logic;
14          dime_in : IN std_logic;
15          quarter_in : IN std_logic;
16          candy_out : OUT std_logic;
17          nickel_out : OUT std_logic;
18          dime_out : OUT std_logic);
19      end Component;
20
21      --Inputs
22      signal clk : std_logic := '0';
23              .
24              .
25      signal quarter_in : std_logic := '0';
26
27      --Outputs
28      signal candy_out : std_logic;
29              .
30      signal dime_out : std_logic;
31
32      --Clock period definitions
33      constant clk_period : time := 10 ns;
34
35
```

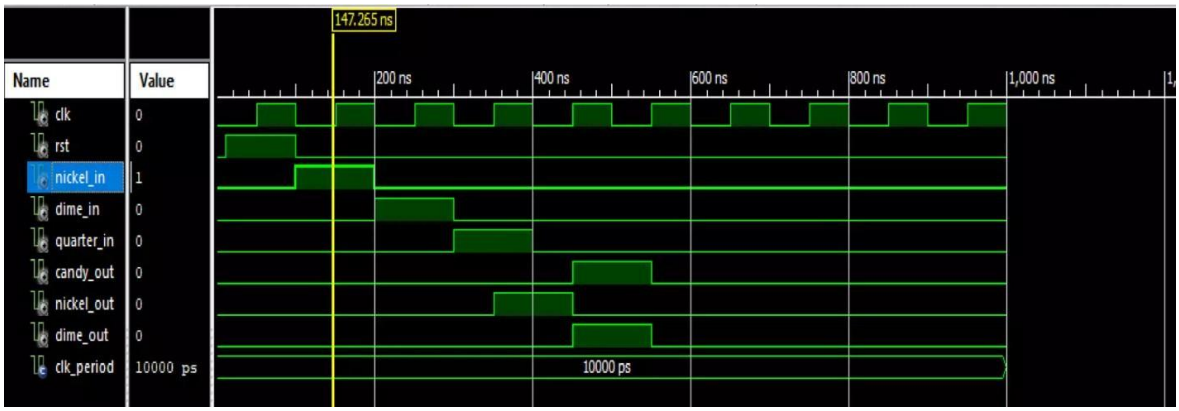
Εικόνα 20 Παράδειγμα κώδικα Testbench 1

Στη συνέχεια προστίθεται η ενότητα που παράγει τις εισόδους του DUT, όπου στη συγκεκριμένη περίπτωση εισάγεται ένα portmap με όλες τις αντιστοιχίες των εισόδων και εξόδων. Στους ορισμούς του ρολογιού περιμένουμε 5 ns για να αντιδράσει το DUT καθώς και για την έξοδο να σταθεροποιηθεί. Το τελευταίο βήμα είναι η περιγραφή της διαδικασίας stimulus η οποία εκτελείται διαδοχικά με την βοήθεια της εντολής wait. Σκοπός της είναι να επιβεβαιώσει τις εξόδους συγκρίνοντας τες με αναμενόμενες τιμές, αλλά παράλληλα και να ελέγξει πως θα αντιδράσει το DUT εξαρτώμενο από τις διάφορες εισόδους.

```
36      Begin
37
38      --Instantiate the Unit Under Test (UUT)
39      uut: vending_machine port map (
40              clk => clk,
41              rst => rst,
42              nickel_in => nickel_in,
43                      .
44                      .
45              dime_out => dime_out
46            );
47
48      --Clock process definitions
49      clk_process : process
50      begin
51          clk <= '0';
52          wait for clk_period*5;
53          clk <= '1';
54          wait for clk_period*5;
55      end process;
56
57
58      --Stimulus process
59      stim_proc : process
60      begin
61      wait for 10 ns;
62          rst<='1';
63      wait for 90 ns;
64          rst<='0';
65          nickel_in<='1';
66          dime_in<='0';
67          quarter_in<='0';
68      wait for 100 ns;
69          nickel_in<='0';
70          dime_in<='1';
71          quarter_in<='0';
72                .
73                .
74      wait for 100 ns;
75          nickel_in<='0';
76          dime_in<='0';
77          quarter_in<='0';
78
79          wait;
80      end process;
81
82  End;
```

Εικόνα 21 Παράδειγμα κώδικα Testbench 2

Το τελικό αποτέλεσμα της παραπάνω κωδικοποίησης φαίνεται στην εικόνα. Παρατηρείται ότι οι τιμές εξόδου συμφωνούν με τις τιμές εισόδου. Άρα με αυτόν τον τρόπο επαληθεύεται ότι ο κώδικας VHDL για τον οποίο εκτελέστηκε το Testbench είναι σωστός.



Εικόνα 22 Αποτέλεσμα Προσομοίωσης με χρήση Testbench

5 Ελεγκτής μηχανής αυτόματου πωλητή

5.1 Η Λειτουργία του αυτόματου πωλητή

Σκοπός του ελεγκτή μηχανής αυτόματου πωλητή είναι να προσομοιώσει τη λειτουργία ενός αυτόματου πωλητή γλυκών. Το μηχάνημα δέχεται πεντάλεπτα(nickel_in), δεκάλεπτα(dime_in) και εικοσι-πεντάλεπτα(quarter_in) κέρματα και διανέμει το γλυκό(candy_out) μόνο όταν εισαχθεί το ποσό των 25 λεπτών. Εκτός από τον μηχανισμό διανομής γλυκών εμπεριέχει και έναν μηχανισμό επιστροφής κερμάτων για ρέστα, καθώς και μηχανισμό εντοπισμού διαθέσιμων δεκάλεπτων στο κουτί των κερμάτων ώστε να επιστρέφει για ρέστα. Επιπλέον, με μία επικοινωνία τύπου "χειραψίας" μεταξύ του ελεγκτή και ενός εξωτερικού κυκλώματος εξακριβώνεται ότι τα κέρματα που έλαβε είναι έγκυρα ώστε να προχωρήσει στην επόμενη κατάσταση. Χρησιμοποιήθηκαν πολύ απλά βήματα κατά τον σχεδιασμό και την υλοποίηση έτσι ώστε να επιβεβαιωθεί η αποτελεσματικότητα του αυτόματου πωλητή κατά την παράδοση του γλυκού καθώς και για να μπορέσουν να εφαρμοστούν τα απαραίτητα πρόσθετα χαρακτηριστικά ασφάλειας. Γι' αυτό για το project χρησιμοποιήθηκαν 3 κέρματα τύπου boolean τα οποία περιμένουν να εισαχθούν κατά τις πρώτες 6 καταστάσεις. Στις υπόλοιπες καταστάσεις, εάν δηλαδή ο πελάτης εισάγει κέρματα υψηλότερης αξίας από την τιμή του γλυκού, τότε το μηχάνημα μεταβαίνει στην κατάσταση που διανέμει το προϊόν μαζί με τα ρέστα. Επομένως, γίνεται αντιληπτό ότι είναι ένα μηχάνημα που χρησιμοποιεί τη σύγχρονη ακολουθιακή λογική, που σημαίνει ότι η έξοδος συγχρονίζεται και εξαρτάται από την παρούσα κατάσταση (present_state). Το πιο γενικό μοντέλο μιας ακολουθιακής μηχανής έχει εισόδους, εξόδους και εσωτερικές καταστάσεις. Δεδομένου ότι είναι αδύνατο να υλοποιηθούν μηχανές που έχουν άπειρες δυνατότητες αποθήκευσης, εξετάστηκε η προσέγγιση ως προς τις μηχανές πεπερασμένων καταστάσεων (FSM). Όπως αναφέρθηκε και στα παραπάνω κεφάλαια οι μηχανές πεπερασμένων καταστάσεων είναι ακολουθιακά κυκλώματα των οποίων η παρούσα κατάσταση μπορεί να επηρεάσει τη μελλοντική τους συμπεριφορά μόνο με πεπερασμένο αριθμό τρόπων, δηλαδή είναι μηχανές με σταθερό αριθμό καταστάσεων. Συνεπώς, λαμβάνοντας υπόψη την λειτουργία του μηχανήματος, ο σχεδιασμός του αυτόματου πωλητή επιτυγχάνεται βασισμένος σε μια FSM Moore προσέγγιση. Αν και μια μηχανή Mealy μπορεί να μοντελοποιηθεί με τον ίδιο τρόπο όπως η μηχανή Moore, στην προκειμένη περίπτωση θα έκανε πιο πολύπλοκο το project κατά την σχεδίαση λόγω του ότι είναι μηχανές που εξαρτώνται από τις εισόδους αλλά και

την παρούσα κατάσταση. Εκτός από αυτό μια μηχανή Mealy λειτουργεί καλύτερα με λιγότερες καταστάσεις. Η διαδικασία των 14 μηχανών πεπερασμένων καταστάσεων (st0, st5, st10, st15, st20, st25, st30, st35, st40, st45, st_idle, st_return_5, st_return_10, st_return_25) έχει μοντελοποιηθεί με τέτοιο τρόπο ώστε να περιλαμβάνει διαφορετικούς μηχανισμούς ανά κατάσταση. Δημιουργήθηκαν καταστάσεις για να παρακολουθούν την τρέχουσα κατάσταση του αυτόματου πωλητή, αλλά και να εντοπίζουν εάν διανέμει γλυκά, επίσης αν είναι σε αδράνεια για λόγους ασφαλείας, ακόμα και αν έχει ξεμείνει από ρέστα. Όλες οι αλλαγές στις εξόδους "candy_out", "nickel_out" και "quarter_out" γίνονται στην παρούσα κατάσταση. Στο "coin_accepted" ωστόσο σήματα κατάστασης (state signals) χρησιμοποιήθηκαν σε συνδυασμό με το ρολόι, ώστε να γίνει αποδεκτό το κέρμα, αφού πρώτα έχει ενημερωθεί από το "coin_valid" ότι λήφθηκε ένα έγκυρο κέρμα. Η μηχανή πεπερασμένων καταστάσεων μπορεί στη συνέχεια να υλοποιηθεί σε μια διεργασία με μια δήλωση Case. Η εντολή Case περιέχει μια εντολή When για κάθε μια από τις πιθανές καταστάσεις, προκαλώντας το πρόγραμμα να ακολουθήσει διαφορετικές διαδρομές για κάθε κατάσταση. Η δήλωση When μπορεί επίσης να περιέχει κώδικα που πρέπει να εκτελεστεί ενώ βρίσκεται στη συγκεκριμένη κατάσταση. Η κατάσταση θα αλλάζει στη συνέχεια όταν ικανοποιείται μια προκαθορισμένη συνθήκη. Εκτός από αυτό, άλλος ένας μηχανισμός που υλοποιήθηκε και λειτουργεί στην διαδικασία της FSM είναι το "no_dime". Είναι άλλη μία είσοδος η οποία όταν ενεργοποιηθεί, δηλώνει ότι δεν μπορούν να βγουν άλλα δεκάλεπτα (dime) κέρματα από τον αυτόματο πωλητή διότι το κουτί των κερμάτων είναι άδειο. Επομένως μόνο πεντάλεπτα (nickel) είναι διαθέσιμα για ρέστα. Αυτός ο μηχανισμός επηρεάζει και την συμπεριφορά της επόμενης κατάστασης, αλλάζοντας της το μονοπάτι μέχρι να μοιράσει όλα τα ρέστα. Προκειμένου να αποφευχθεί πιθανή σύγχυση, η οποία μπορεί να προκύψει όταν τα κέρματα παραμένουν στην είσοδο της μηχανής FSM για περισσότερο από έναν κύκλο ρολογιού, είσοδοι, έξοδοι, σήματα και components επικοινωνούν μεταξύ τους μέσω μιας διαδικασίας τύπου "Handshake". Το "coin_valid" και το "coin_accepted" επικοινωνούν μεταξύ τους από το εξωτερικό κύκλωμα προς τον ελεγκτή του αυτόματου πωλητή, και ενημερώνουν αντιστοίχως ότι μία νέα είσοδος κέρματος είναι έτοιμη προς ανάγνωση και ότι η επεξεργασία της εισόδου λήφθηκε επιτυχώς αναγνωρίζοντας το κέρμα που καταχωρήθηκε. Τέλος, διαθέτει και μηχανισμό με χαρακτηριστικά επιστροφής χρημάτων σε περίπτωση που εισάγονται κέρματα γρήγορα ακόμα και εάν έχει καταβληθεί το απαιτούμενο ποσό και έχει διανεμηθεί το γλυκό, αποφεύγοντας έτσι τυχόν λανθασμένη εισαγωγή κερμάτων.

Ο δεύτερος κώδικας VHDL που χρησιμοποιήθηκε υλοποιεί ένα εξωτερικό κύκλωμα που δέχεται κέρματα, ανιχνεύει την αξία του κέρματος και εξάγει ένα σήμα που δείχνει αν το κέρμα είναι έγκυρο ή όχι ,έχοντας σαν έξοδο το "coin_valid" και σαν είσοδο το "coin_accepted". Η αρχιτεκτονική αυτουνού του προγράμματος περιλαμβάνει τρεις διεργασίες που ενεργοποιούνται τόσο με το σήμα επαναφοράς όσο και με την ανοδική ακμή του σήματος ρολογιού. Οι διεργασίες χρησιμοποιούν τα σήματα εισόδου για την ανίχνευση των ανερχόμενων ακμών και τις αντιστοιχούν σε σήματα που αντιπροσωπεύουν κάθε τύπο νομίσματος. Ένας τρίτος κώδικας VHDL επίσης χρειάστηκε για να συνδέσει τον ελεγκτή του αυτόματου πωλητή με το εξωτερικό κύκλωμα. Ο κώδικας VHDL για αυτό το πρόγραμμα είναι γραμμένος σε ιεραρχική σχεδίαση η οποία χωρίζεται σε πολλαπλά components. Αυτό επιτρέπει μια σπονδυλωτή και κλιμακωτή προσέγγιση στη σχεδίαση ψηφιακών κυκλωμάτων. Η αρχιτεκτονική ουσιαστικά συνδυάζει τον προηγούμενο κώδικα του εξωτερικού κυκλώματος (external_circuit_new) με τον κώδικα του ελεγκτή αυτόματου πωλητή (Vending_Machine). Το component του αυτόματου πωλητή είναι υπεύθυνο για τον έλεγχο της διανομής των γλυκών και των κερμάτων, ενώ το component του εξωτερικού κυκλώματος είναι υπεύθυνο για την ανίχνευση της εισαγωγής ενός κέρματος και τον προσδιορισμό αν πρόκειται για έγκυρο νόμισμα. Η συνδυασμένη αρχιτεκτονική αντιπροσωπεύει

ουσιαστικά τη συνολική λειτουργικότητα ενός αυτόματου πωλητή γλυκών που δέχεται κέρματα ως πληρωμή.

5.2 Μεθοδολογία Σχεδιασμού

Η χρήση οποιουδήποτε αυτόματου πωλητή ακολουθεί συνήθως συγκεκριμένα μοτίβα, συμπεριλαμβάνοντας κάποιες αλλαγές ανάλογα με τις απαιτήσεις της σχεδίασης. Τα μηχανήματα τις περισσότερες φορές λειτουργούν όταν κάποια χρήματα (κέρματα, χαρτονομίσματα, και πλέον στα πιο σύγχρονα, ακόμα και πιστωτική κάρτα) τοποθετούνται σε μια θυρίδα ή σε μία συσκευή ανάγνωσης καρτών. Στη συνέχεια, πρέπει να ενεργοποιηθεί ένα κουμπί ή να τραβηχτεί ένας μοχλός εκκίνησης. Μετά την εισαγωγή του ακριβούς ποσού στη θυρίδα του αυτόματου πωλητή, το επιθυμητό αντικείμενο θα αποδοθεί στον πελάτη μέσω ενός μηχανισμού διανομής. Συνοπτικά, η μεθοδολογία του σχεδιασμού στην συγκεκριμένη πτυχιακή εργασία πραγματοποιήθηκε πληρώντας τα εξής βήματα λειτουργίας:

1. Αυτόματος πωλητής που πουλάει γλυκά αξίας 25 λεπτών.
2. Χρήση Moore τύπου μηχανή πεπερασμένων καταστάσεων (FSM).
3. Καταστάσεις που υποδηλώνουν το ποσό που έχει εισαχθεί και πράττουν αναλόγως.
4. Μηχανισμός ελέγχου εγκυρότητας κερμάτων, ο οποίος ενημερώνει ότι ένα κέρμα είναι έτοιμο για ανάγνωση.
5. Μηχανισμός αποδοχής κερμάτων που ενημερώνει ότι έγινε επιτυχώς η επεξεργασία του κέρματος.
6. Μηχανισμός επιστροφής κερμάτων για ρέστα.
7. Μηχανισμός που ενημερώνει ότι το κουτί των δεκάλεπτων είναι άδειο και δίνει για ρέστα μόνο πεντάλεπτα.
8. Μηχανισμός επιστροφής χρημάτων, σε περίπτωση τοποθέτησης κερμάτων ενώ ο αυτόματος πωλητής είναι σε αδράνεια. (Μετά από διανομή γλυκού)
9. Διαδικασία επικοινωνίας “χειραψίας” μεταξύ του ελεγκτή του αυτόματου πωλητή και του εξωτερικού κυκλώματος.
10. Τρίτος κώδικας που πραγματοποιεί τον συνδυασμό μέσω των component.

Στη συνέχεια περιγράφεται η μέθοδος σχεδίασης κάθε κώδικα που χρησιμοποιήθηκε ξεχωριστά. Ξεκινώντας από:

vending_machine.vhd

Το μηχάνημα δέχεται αποκλειστικά κέρματα των 5, 10 και 25 λεπτών, τα οποία αποτελούν αντίστοιχα τις εισόδους τύπου boolean “nickel_in”, “dime_in”, “quarter_in”. Είναι επίσης απαραίτητα τα πρόσθετα σήματα “clk” και “rst”, τα οποία υποδεικνύουν το σήμα του ρολογιού και τη ασύγχρονη επαναφορά του μηχανήματος, η οποία ενεργοποιείται με την ανίχνευση θετικού σήματος. Με την επόμενη είσοδο “no_dime”, η μηχανή ενημερώνεται ότι έχει εξαντληθεί η διαθέσιμη ποσότητα δεκάλεπτων και δίνει ρέστα μόνο σε πεντάλεπτα. Σημειώνεται επίσης ότι σε αυτόν τον σχεδιασμό θα αγνοηθεί η χωρητικότητα του αποθέματος των πεντάλεπτων, πράγμα που σημαίνει ότι θα υποθέσουμε ότι θα υπάρχει πάντα γεμάτο το κουτάκι των πεντάλεπτων. Άλλη μία είσοδος είναι το “coin_valid”, με της οποίας την λήψη του σήματος μπορεί ο αυτόματος πωλητής να λάβει κέρμα χωρίς να το απορρίψει. Ωστόσο, αμέσως μόλις κατατεθεί ένα έγκυρο κέρμα, τότε θα ενεργοποιηθεί η έξοδος “coin_accepted”, η οποία θα το αποδεχτεί αυτομάτως. Άλλοι έξοδοι που περιλαμβάνονται είναι τα “nickel_out”, “dime_out”, και “candy_out” τα οποία υποδηλώνουν τα κέρματα για ρέστα και το γλυκό που περιμένει να διανεμηθεί. Ο ακόλουθος κώδικας συνεχίζει με τον ορισμό των σημάτων τα οποία χρησιμοποιήθηκαν και στις διεργασίες ως εσωτερικά

σήματα. Επιπλέον, καθορίστηκαν επίσης και καταστάσεις απαριθμημένου τύπου, οι οποίες περιγράφουν την συμπεριφορά του κυκλώματος καθώς αλλάζουν κατάσταση ανάλογα με τα σήματα εισόδου. Ο κώδικας VHDL που χρησιμοποιήθηκε για τη μηχανή είναι πολύ απλός και εξαρτάται από τον κύκλο ρολογιού. Οι διεργασίες εκτελούνται σε κάθε κύκλο ρολογιού, επιτρέποντας στη μηχανή κατάστασης να ενημερώνει τη συμπεριφορά της ακολουθιακά με βάση τα σήματα εισόδου και την τρέχουσα κατάσταση του συστήματος. Μια από τις διεργασίες χρησιμοποιείται για να πάρει την τιμή του σήματος "ca" και να την ενσωματώσει στην έξοδο "coin_accepted". Η επόμενη διεργασία συγχρονίζει τη μηχανή κατάστασης με το σήμα ρολογιού και διασφαλίζει ότι η παρούσα κατάσταση ενημερώνεται σωστά με βάση τη λογική μετάβασης καταστάσεων και των σημάτων επαναφοράς (ασύγχρονο reset). Μία άλλη διεργασία που χρησιμοποιήθηκε για να επηρεάσει τη συμπεριφορά της μηχανής κατάστασης μετά τη διανομή του γλυκού. Πιο συγκεκριμένα, με τη χρήση ενός μετρητή (counter), δημιουργείται μια καθυστέρηση στο κύκλωμα για δύο κύκλους ρολογιού και επαναφέρεται στη συνέχεια στην αρχική του τιμή, δηλαδή το 0 (όταν counter=2). Οι μετρητές είναι συχνά χρήσιμοι για καθυστερήσεις, διαίρεση ρολογιών και πολλές άλλες χρήσεις. Στην συγκεκριμένη περίπτωση η χρήση του μετρητή προέκυψε για να μετρήσει 2 χτύπους του ρολογιού μετά από την διανομή του γλυκού. Προχωρώντας στο επάνω τμήμα της FSM διασφαλίζεται ότι τα απαραίτητα σήματα βρίσκονται στη λίστα ευαισθησίας της διεργασίας και επίσης ότι καθορίζονται επιτυχώς οι συνδυασμοί εισόδων/εξόδων και σημάτων. Παρατηρείτε επίσης στον κώδικα ότι τυχόν αλλαγές και μεταβάσεις πραγματοποιούνται στην παρούσα κατάσταση.Με αυτόν τον τρόπο μπορεί να αντιληφθεί ότι η μηχανή είναι τύπου Moore, επειδή οι έξοδοι εξαρτώνται αποκλειστικά και μόνο από την αποθηκευμένη τρέχουσα κατάσταση. Αρχικώς, η ως έχει μηχανή ευρισκόμενη στην αρχική κατάσταση st0 αναμένει να τοποθετηθεί κέρμα προκειμένου να πραγματοποιηθεί μετάβαση σε επόμενη κατάσταση. Όπως αναφέρθηκε και παραπάνω κάθε μία από τις καταστάσεις υποδηλώνει και την τιμή η οποία αντιστοιχεί σε μια εισαγωγή κέρματος που είναι κατατεθειμένη στην παρούσα κατάσταση. Εκ των οποίων στις st0, st5, st10, st15, st20 γίνεται μόνο εισαγωγή κερμάτων, στην st25 γίνεται η διανομή του γλυκού, αλλά για λόγους ασφαλείας η επόμενη κατάσταση είναι η st_idle η οποία μπαίνει στην πρώτη κατάσταση αδράνειας του μηχανήματος πριν μεταβεί στην πραγματική αρχική κατάσταση της μηχανής st0, ώστε να ξεκινήσει η έναρξη του νέου κύκλου λειτουργίας της. Στις επόμενες καταστάσεις st30, st35, st40, st45 ενεργοποιούνται και τα ρέστα, αλλά και ο μηχανισμός που ελέγχει αν υπάρχουν δεκάλεπτα στον αυτόματο πωλητή μέσω του σήματος no_dime. Όπου όταν βρίσκεται στο θετικό '1' αλλάζει τα μονοπάτια των καταστάσεων ώστε να συμπληρώσει τα ρέστα με πεντάλεπτα. Για παράδειγμα, αν βρίσκεται στην κατάσταση st35 και είναι ενεργοποιημένο το no_dime='1', τότε θα επιστρέψει για ρέστα ένα πεντάλεπτο(nickel_out), και η επόμενη μετάβαση θα' ναι στην st30, η οποία θα απελευθερώσει το γλυκό μαζί με άλλο ένα πεντάλεπτο(nickel_out). Η κατάσταση st_idle υποδεικνύει την πρώτη κατάσταση ασφάλειας του μηχανήματος μετά από την διανομή του γλυκού. Δημιουργήθηκε η λειτουργία αυτής της κατάστασης ώστε να διασφαλίσει ότι για 2 κύκλους του ρολογιού δεν θα τοποθετηθεί κέρμα προτού προλάβει ο αυτόματος πωλητής να ανασυγκροτηθεί. Όσο βρίσκεται σε αυτήν την κατάσταση ότι κέρμα εισάγει ο πελάτης θα το επιστρέφει, μεταβαίνοντας ανάλογα με το κέρμα που εισαχθεί στην αντίστοιχη κατάσταση επιστροφής χρημάτων (δηλαδή τις st_return_5, st_return_10, st_return_25). Αντίστοιχα αν σε οποιαδήποτε στιγμή ενεργοποιηθεί το σήμα του ασύγχρονου 'rst', η μηχανή θα επιστραφεί στην αρχική κατάσταση "st0".Πιο λεπτομερής περιγραφή των καταστάσεων περιγράφεται σε αντίστοιχο κεφάλαιο παρακάτω.

external_circuit_new.vhd

Αυτό το δεύτερο πρόγραμμα VHDL υλοποιεί ένα εξωτερικό κύκλωμα που δέχεται κέρματα για να ανιχνεύει την αξία τους, και εξάγει ένα σήμα που δείχνει αν το κέρμα είναι έγκυρο ή όχι. Η βασική του λειτουργία είναι να εξασφαλίσει την ορθή λειτουργία της επικοινωνίας μεταξύ του μηχανισμού ελέγχου εγκυρότητας κερμάτων και του μηχανισμού επεξεργασίας κερμάτων. Γι' αυτόν τον λόγο, σε αντίθεση με τον κώδικα 'vending_machine.vhd', το 'coin_valid' είναι η μόνη έξοδος και το 'coin_accepted' λειτουργεί σαν είσοδος. Η αρχιτεκτονική 'schematic' ορίζει τρεις διεργασίες που ανιχνεύουν τις ανοδικές ακμές για κάθε τύπο νομίσματος καθώς και σήματα που ενημερώνονται αντίστοιχα με βάση τα σήματα εισόδου (nickel_in, dime_in, quarter_in). Τα σήματα 'nickel_rising_edge_detect', 'dime_rising_edge_detect' και 'quarter_rising_edge_detect' τίθενται σε 'true' μόνο στις θετικές ακμές της ανίχνευσης των κερμάτων και κλείνουν κατευθείαν στις ψευδείς. Τέλος, υπολογίζεται το σήμα 'input' για το κύκλωμα θέτοντας το σε ψευδή τιμή (false) εάν ένα νόμισμα έχει ήδη γίνει αποδεκτό, διαφορετικά θέτοντάς την σε αληθή τιμή(true) εάν έχει ανιχνευθεί μια ανερχόμενη ακμή για οποιοδήποτε από τα νομίσματα. Το σήμα 'coin_valid' τίθεται σε '1' όταν το 'input' είναι αληθές και σε αντίθετη περίπτωση σε '0', υποδεικνύοντας εάν έχει εισαχθεί έγκυρο νόμισμα ή όχι. Με αυτόν τον τρόπο, προσφέρεται η δυνατότητα στον αυτόματο πωλητή να λειτουργεί με αυξημένη αποδοτικότητα, επιτρέποντας την άμεση αποδοχή του επόμενου νομίσματος, με αποφυγή πιθανών καθυστερήσεων. Επίσης, με τον συγκεκριμένο κώδικα επιβεβαιώνεται ότι ένα νόμισμα θα λαμβάνεται υπόψη από τον ελεγκτή μόνο κατά την ανερχόμενη ακμή του ρολογιού. Αυτό τον καθορίζει και σημαντικό διότι αποφεύγεται πιθανή σύγχυση, η οποία μπορεί να προκύψει όταν τα κέρματα παραμένουν στην είσοδο της μηχανής για περισσότερους από έναν κύκλο του ρολογιού.

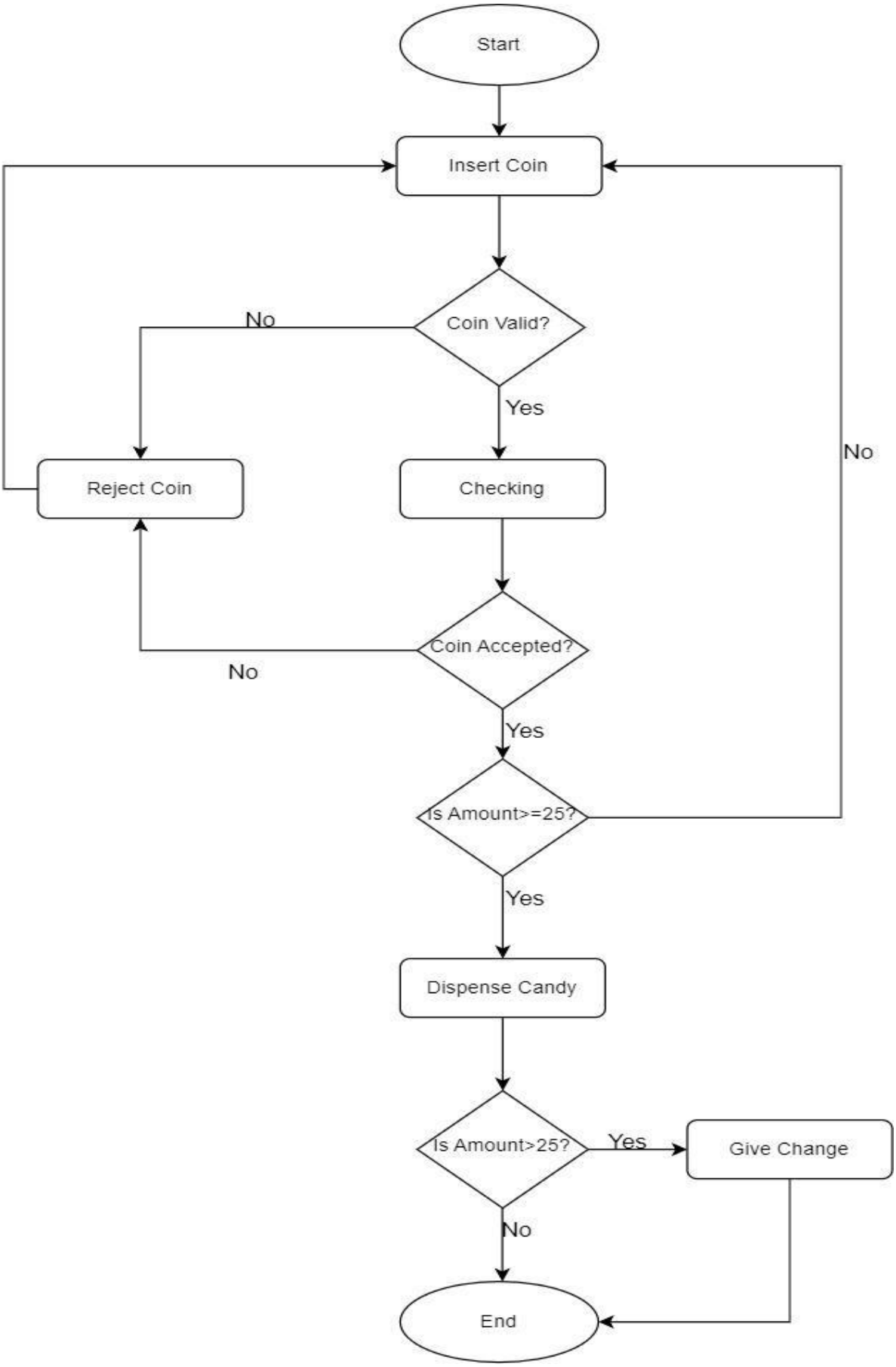
machine.vhd

Ένας τρίτος κώδικας χρειάστηκε για να συνδυάσει με την χρήση των component τους 2 κώδικες, ο οποίος βρίσκεται στο υψηλότερο επίπεδο ιεραρχίας ώστε να κληρονομήσει τις λειτουργίες των 2 συστατικών. Στο πλαίσιο αυτού του κώδικα, χρησιμοποιήθηκαν τα "coin_valid" και "coin_accepted" ως έξοδοι, διότι δεν απαιτείται να οριστούν ξανά οι προϋποθέσεις των μηχανισμών για την αποδοχή ενός κέρματος, όπως περιγράφεται στον κώδικα "vending_machine.vhd" και στον "external_circuit_new". Αυτό οφείλεται στον τρόπο επικοινωνίας και στην ροή δεδομένων στους κώδικες, καθιστώντας τους υπεύθυνους για την επικύρωση και επεξεργασία των κερμάτων παράλληλα. Το συστατικό "external_circuit_new" αναπαρίσταται με την ετικέτα "ec" και το συστατικό "vending_machine" αναπαρίσταται με την ετικέτα "vm". Τα σήματα εισόδου και εξόδου αντιστοιχίζονται στις ανάλογες θύρες για κάθε component ξεχωριστά. Η συνδυασμένη αυτή αρχιτεκτονική επικοινωνίας "handshake" παρέχει την απαραίτητη ασφάλεια στην λειτουργικότητα του Αυτόματου πωλητή.

5.2.1 Διάγραμμα Ροής Αυτόματου Πωλητή

Το παρακάτω διάγραμμα ροής δείχνει τη ροή των διαφόρων εργασιών που εκτελούνται κατά τη λειτουργία του αυτόματου πωλητή. Αρχικά, ο αυτόματος πωλητής είναι σε κατάσταση αδράνειας μέχρι να τοποθετηθεί κάποιο κέρμα, στη συνέχεια ο μηχανισμός ελέγχου εγκυρότητας κερμάτων ελέγχει αν είναι έγκυρο. Σε περίπτωση που δεν είναι, το κέρμα απορρίπτεται, ενώ αν είναι έγκυρο, προχωράει από την ανάγνωση της αξίας του κέρματος στην αποδοχή του. Η μέτρηση αυξάνεται καθώς εισάγονται έγκυρα κέρματα. Έπειτα, το μηχάνημα ελέγχει το συνολικό ποσό που έχει κατατεθεί και εάν αυτό είναι ίσο με 25 λεπτά, τότε το μηχάνημα διανέμει το γλυκό. Σε περίπτωση που το ποσό είναι μεγαλύτερο από την αξία του γλυκού τότε το μηχάνημα διανέμει το γλυκό μαζί

με τα ρέστα. Εάν δεν έχει συμπληρωθεί το πόσο της αξίας του γλυκού, τότε το μηχάνημα περιμένει από τον πελάτη να εισάγει περισσότερα κέρματα.

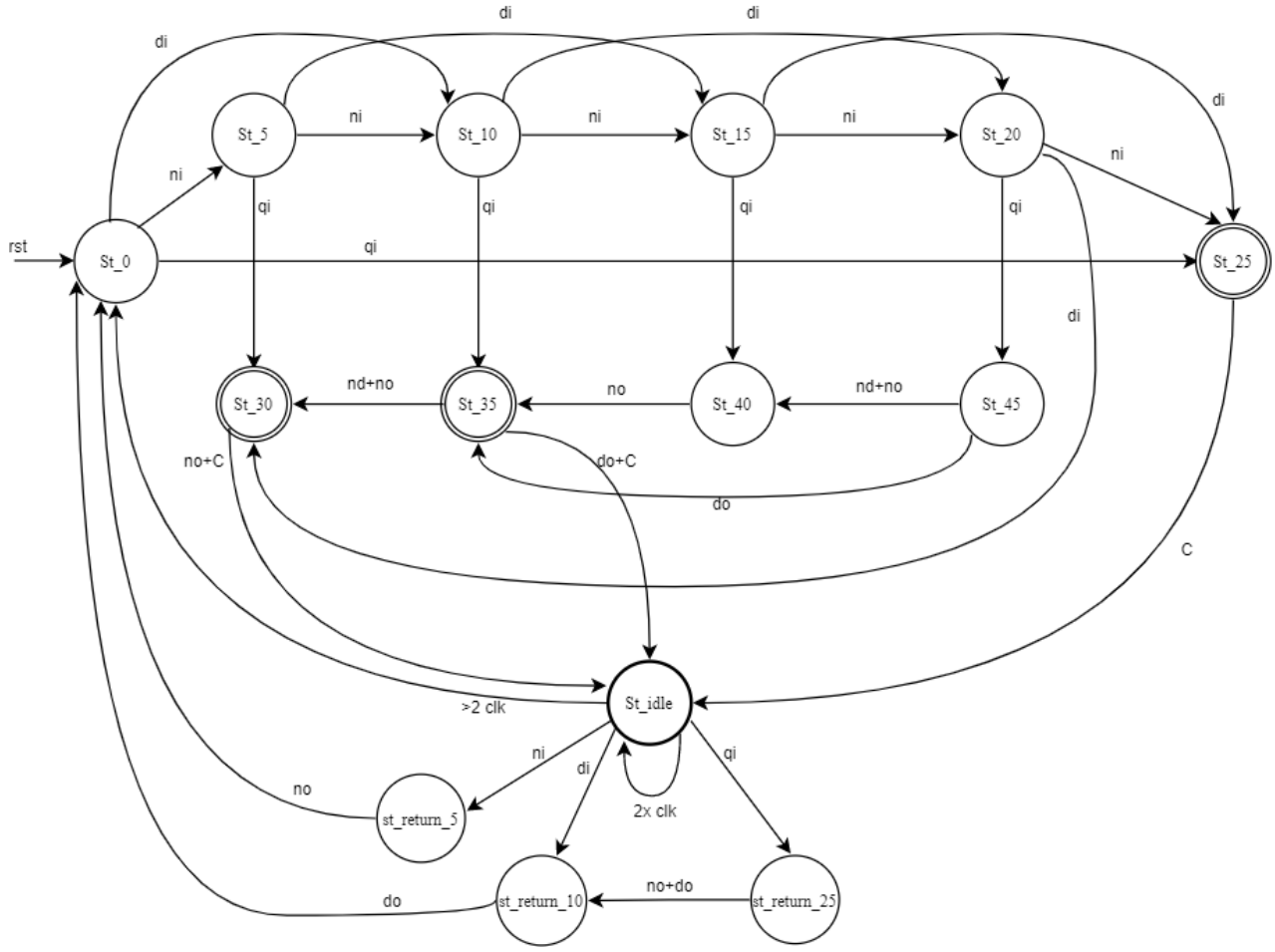


Εικόνα 23 Διάγραμμα ροής Αυτόματου πωλητή

5.2.2 Περιγραφή των καταστάσεων

Οι μηχανές κατάστασης στη VHDL είναι χρονισμένες διεργασίες των οποίων οι έξοδοι ελέγχονται από την τιμή ενός σήματος κατάστασης. Το σήμα κατάστασης χρησιμεύει ως εσωτερική μνήμη του τι συνέβη στην προηγούμενη επανάληψη. Εξαρτώνται από τα σήματα του ρολογιού και της σύγχρονης επαναφοράς. Αυτό σημαίνει ότι η μηχανή λειτουργεί κατά τη θετική ακμή του ρολογιού και επιστρέφει στην αρχική της κατάσταση όταν ενεργοποιείται το σήμα επαναφοράς. Ο απαριθμημένος τύπος state περιλαμβάνει την λίστα όλων των καταστάσεων της FSM. Στην κάτωθι μηχανή υπάρχουν 14 καταστάσεις, άρα απαιτούνται 4 bit για την κωδικοποίηση τους. Συνεπώς, λαμβάνοντας υπόψη τις καταστάσεις, παράχθηκαν 4 φλιπ-φλοπ, διότι ο μεταφραστής κωδικοποιεί τις καταστάσεις με τη σειρά που παρατίθεται: για st0 ="0000"(δεκαδικό 0), st5="0001"(δεκαδικό 1), ..., st_Return_25="1101"(δεκαδικό 13). Η συμπεριφορά του αυτόματου πωλητή καθορίζεται από τα τμήματα των διεργασιών της μηχανής καθώς και της παρούσα κατάσταση και τα σήματα εισόδου. Η διεργασία περιλαμβάνει μια δήλωση ‘case’ που καθορίζει την επόμενη κατάσταση με βάση την παρούσα κατάσταση και τις εισόδους. Οι είσοδοι λαμβάνονται αντίστοιχα ως τα κέρματα που περιμένουν να τοποθετηθούν. Σε όλες τις καταστάσεις εάν εισαχθεί άκυρο νόμισμα ή δεν εισαχθεί καθόλου, το μηχάνημα παραμένει στην αντίστοιχη παρούσα κατάσταση. Όταν ένα κέρμα εισάγεται στο μηχάνημα και είναι έγκυρο (το ‘coin_valid’ είναι υψηλό), το κέρμα γίνεται αποδεκτό (το ‘coin_accepted’ τίθεται σε υψηλό). Τα ‘ca’,’cv’ είναι τα αντίστοιχα εσωτερικά σήματα τα οποία μπορούν να ενημερώνονται σε κάθε μετάβαση. Επιπλέον, το σήμα ‘nd’ ελέγχεται επίσης για να διαπιστωθεί αν το μηχάνημα δεν έχει άλλα 10λέπτα. Στην περίπτωση που ενεργοποιηθεί η είσοδος ‘no_dime’ η μηχανή αλλάζει το μονοπάτι μετάβασης ώστε να συμπληρώσει το ποσό που απαιτείται για ρέστα, επιστρέφοντας 5λεπτα αντί για 10λέπτα. Σε κάθε περίπτωση που διανεμηθεί το γλυκό, η αμέσως επόμενη κατάσταση θα είναι η ‘st_idle’. Η οποία θεωρείται μία κατάσταση αδράνειας για δύο κύκλους του ρολογιού. Εάν εισαχθεί κέρμα όσο η μηχανή βρίσκεται σε αυτήν την κατάσταση τότε ενεργοποιούνται και οι αμέσως επόμενες καταστάσεις (‘st_return_5’, ‘st_return_10’, και‘st_return_25’) στις οποίες η μηχανή επιστρέφει το αντίστοιχο ποσό και επιστρέφει στην αρχική κατάσταση ‘st0’. Οι συγκεκριμένες καταστάσεις λειτουργούν με την συμβολή των εσωτερικών σημάτων ‘candy_dispensed’, το οποίο ενεργοποιείται όταν διανέμεται το γλυκό, και το σήμα ‘counter’ που αυξάνεται για να μετρήσει τον αριθμό των κύκλων ρολογιού που έχουν περάσει από τότε που διανεμήθηκε το γλυκό.

Για να διευκρινισθεί παραπάνω η λειτουργιά της μηχανής καταστάσεων κατασκευάστηκε ένα διάγραμμα καταστάσεων, στο οποίο καταστάσεις της μηχανής αναπαρίστανται μέσα σε κύκλους. Η κατάσταση ‘st_0’ είναι η αρχική κατάσταση της μηχανής. Οι καταστάσεις με τον μονό κύκλο είναι αυτές που δέχονται κέρματα, ενώ οι καταστάσεις ‘st40’, ‘st45’ έχουν μονό κύκλο επειδή απλώς απελευθερώνουν ρέστα και δεν πραγματοποιούν κάποια άλλη ενέργεια. Στις καταστάσεις που παρουσιάζονται με διπλούς κύκλους γίνεται η διανομή του γλυκού. Ενώ στις καταστάσεις ‘st30’, ‘st35’, ‘st40’, και ‘st45’ δημιουργούνται τα πρόσθετα μονοπάτια, τα οποία μεταβάλλουν τις καταστάσεις βάσει το σήματος no_dime(nd). Η κατάσταση ‘st_idle’ που αναπαρίσταται με πυκνότερο bold κύκλο ,αντιστοιχεί στην κατάσταση στην οποία η μηχανή θα μεταβεί κάθε φορά που γίνεται απελευθέρωση του γλυκού. Πιο συγκεκριμένα είναι η πρώτη κατάσταση αδράνειας, η οποία είναι υποχρεωτική για το μηχάνημα ώστε να επιβεβαιώσει την ορθή λειτουργία του αυτόματου πωλητή και να αποτρέψει τυχόν λανθασμένες ή γρήγορες εισαγωγές νομισμάτων. Αν παραμείνει σε αυτήν την κατάσταση για 2 κύκλους του ρολογιού, τότε το μηχάνημα μεταβαίνει αυτόματα στην αρχική κατάσταση ‘st0’, ώστε να ξεκινήσει ο νέος κύκλος της λειτουργίας της μηχανής FSM, και για να μπορέσει να δεχτεί το επόμενο νόμισμα. Το διάγραμμα κατάστασης (εικόνα 24) υποδεικνύει τη ροή των διαφόρων σημάτων και καταστάσεων κατά τη διάρκεια της λειτουργίας του αυτόματου πωλητή.



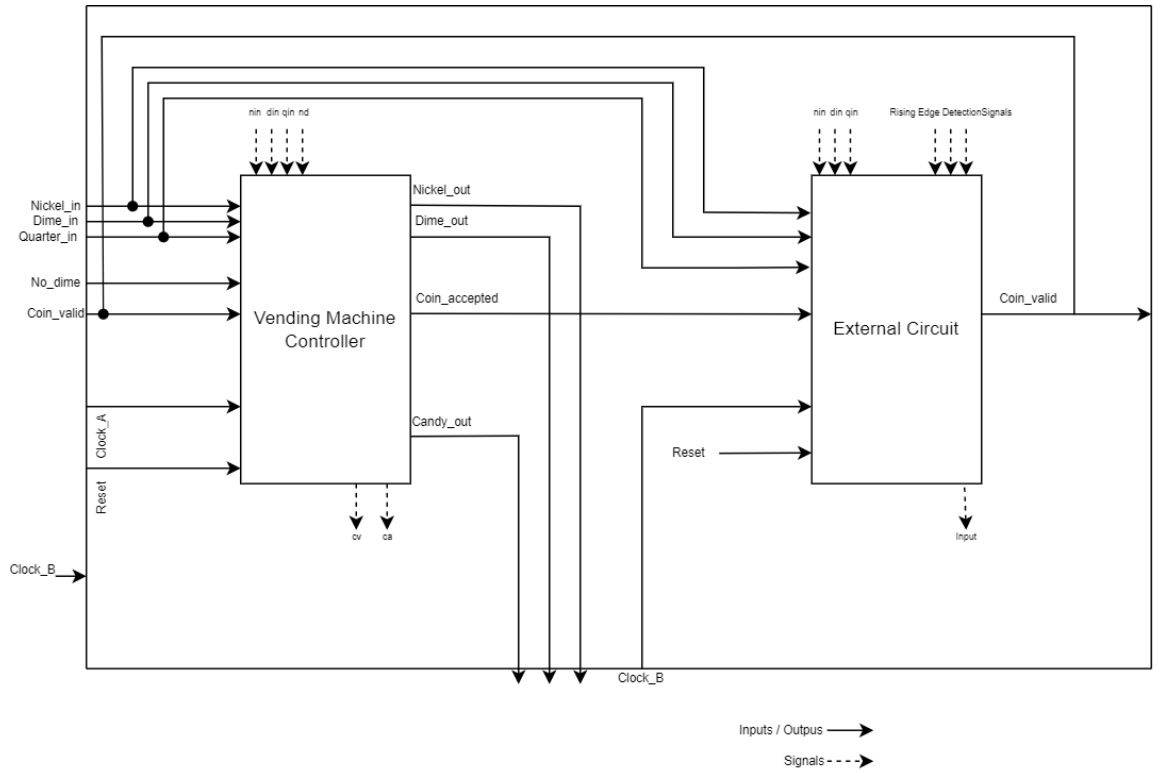
Εικόνα 24 Επάνω τμήμα και διάγραμμα καταστάσεων

Αντιστοίχισησημάτων: ni=nickel_in, di=dime_in, qi=quarter_in, nd=no_dime, no=nickel_out, do=dime_out, c=candy_out

5.2.3 Διαδικασία Handshake

Μια "χειραψία" μεταξύ δύο VHDL κώδικων αναφέρεται σε έναν μηχανισμό συγχρονισμού της επικοινωνίας μεταξύ των δύο κωδίκων. Αυτό μπορεί να επιτευχθεί μέσω της χρήσης σημάτων και διεργασιών. Ένας συνηθισμένος τρόπος υλοποίησης μιας χειραψίας είναι η χρήση ενός σήματος αίτησης και ενός σήματος επιβεβαίωσης. Ο πρώτος κώδικας στέλνει ένα αίτημα θέτοντας το σήμα αίτησης σε μια συγκεκριμένη τιμή και ο δεύτερος κώδικας απαντά θέτοντας το σήμα επιβεβαίωσης σε μια διαφορετική τιμή. Στη συνέχεια, ο πρώτος κώδικας ελέγχει την τιμή του σήματος επιβεβαίωσης για να διαπιστώσει αν το αίτημα έχει εκπληρωθεί. Για παράδειγμα, στην περίπτωση της πτυχιακής εργασίας, ορίστηκαν ο κώδικας του ελεγκτή αυτόματου-πωλητή (vending_machine.vhd) και ο κώδικας του εξωτερικού κυκλώματος (external_circuit.vhd). Οι οποίοι πρέπει να επικοινωνήσουν μεταξύ τους μέσω μιας χειραψίας. Ο κώδικας του ελεγκτή στέλνει ένα αίτημα στον κώδικα του εξωτερικού κυκλώματος θέτοντας το σήμα αίτησης σε '1' και στη συνέχεια ελέγχοντας την τιμή του σήματος επιβεβαίωσης. Ο κώδικας του εξωτερικού κυκλώματος μπορεί να απαντήσει στο αίτημα θέτοντας το σήμα επιβεβαίωσης σε '1' και στη

συνέχεια περιμένοντας να επανέλθει το σήμα αίτησης σε '0'. Το παρακάτω διάγραμμα απεικονίζει ένα block diagram των 2 κώδικων, το οποίο εξηγεί την επικοινωνία που πραγματοποιείται μεταξύ των εισόδων/εξόδων και των σημάτων.



Εικόνα 25 Μπλοκ διάγραμμα της επικοινωνίας Handshake

5.3 Μηνύματα τυχόν σφαλμάτων και αντιμετώπιση

Παρακάτω θα αναφερθούν επίσης οι προκλήσεις και οι περιορισμοί της υλοποίησης του έργου του αυτόματου πωλητή και πως αντιμετωπίστηκαν. Επιπλέον θα παρουσιαστούν οι προσπάθειες δημιουργίας των πρόσθετων καταστάσεων στο vending_machine.vhd με προτεινόμενους κώδικες και γιατί δεν επιλέχθηκαν για το συγκεκριμένο project. Η αρχική ιδέα ήταν να φτιαχτεί μια κατάσταση, στην οποία θα μεταβαίνει η μηχανή όταν διανεμηθεί επιτυχώς το γλυκό και για λόγους ασφαλείας θα απορρίπτει το κέρμα. Οπότε φτιάχτηκαν οι καταστάσεις ‘st_creject’ και ‘st_cchange’ που με την συνθήκη ‘if’ έλεγχε αλλά μόνο για την πρώτη θετική ακμή αν τοποθετηθεί κέρμα και απλά το απέρριπτε. Αυτή η λογική δεν συμφωνούσε παρόλ’αυτά με τους όρους των μηχανών πεπερασμένων καταστάσεων με αποτέλεσμα να δημιουργούνται αρκετά errors.

```
        when st_creject =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '0';
            if (nin) then
            ca <= '1';
            nickel_out <= '1';
            elsif (din) then
            ca <= '1';
            dime_out <= '1';
            elsif (qin) then
            ca <= '1';
            dime_out <= '1';
            nickel_out <= '1';
            next_state <= st_cchange;
            else next_state <= st0;
            end if;
        when st_cchange =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '1';
            next_state <= st0;
        end case;
    end process;
-------------------------------------------------
```

Εικόνα 26 Παράδειγμα λανθασμένου κώδικα

Enum_encoding

Δίχως την χρήση πόρων υλικού ελάχιστων διαστάσεων, δεν θεωρήθηκε αναγκαίο να καθοριστούν οι ελάχιστες διαστάσεις υλικού, επομένως το project δεν εξαρτάται από κάποια τεχνική κωδικοποίησης. Συνεπώς, ο τύπος enum_encoding δεν έχει καθοριστεί στο κύκλωμα και έχει χρησιμοποιηθεί η προεπιλεγμένη ρύθμιση του προγράμματος Quartus II. Ωστόσο, παρατηρείται ότι στον κώδικα vending_machine.vhd, στις γραμμές 18-19, έχει γίνει σχολιασμός της τεχνικής που χρησιμοποιήθηκε μόνο κατά τη σχεδίαση του συγκεκριμένου κώδικα για δοκιμαστικούς λόγους, προκειμένου να διευκολυνθεί η κατανόηση της συμπεριφοράς της μηχανής και, ειδικότερα, της λειτουργίας των καταστάσεων από την τρέχουσα προς την επόμενη κατάσταση.

Glitches

Δεν είναι glitch-free και σε μερικές σπάνιες περίπτωσης εισαγωγής κερμάτων στον αμέσως επόμενο κύκλο του ρολογιού δημιουργείται ανεπιθύμητη συμπεριφορά χωρίς να επηρεάζουν την λειτουργία του αυτόματου πωλητή αλλά πιο πολύ τον καθυστερούν στο να λάβει το επόμενο κέρμα. Αυτές οι δυσλειτουργίες συνήθως προκύπτουν όταν η ψηφιακή έξοδος ενός κυκλώματος χρησιμοποιείται ως ρολόι σε ένα άλλο κύκλωμα (π.χ. για έναν μετρητή ή για έναν καταχωρητή). Επίσης σε εισαγωγή ίδιου κέρματος (παράδειγμα εικόνα 41) ο αυτόματος πωλητής παίρνει παραπάνω κύκλους ρολογιού για να ελέγξει την εγκυρότητα και να αποδεχτεί το κέρμα. Αυτό έχει ως αποτέλεσμα να μην δέχεται τα κέρματα αν τοποθετηθούν σε μικρό χρονικό διάστημα.

candy_dispensedsignal

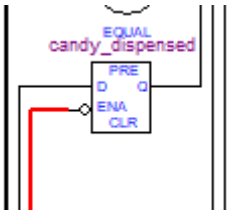
Το ‘candy_dispensed’signal πρέπει να γίνεται ‘1’ μόνο στην πρώτη κατάσταση αδράνειας ‘st_idle’, και να παραμένει στο ‘0’ σε οποιοδήποτε άλλη κατάσταση. Αυτό δημιουργούσε το μεγαλύτερο πρόβλημα στον κώδικα διότι πριν εντοπιστεί το error, είχεοριστεί το σήμα σε ‘1’ σε όλες τις καταστάσεις που το ‘candy_out’ γινόταν ‘1’. Το παρακάτω error εμφανιζόταν:

Clock Hold: 'clk'

	Minimum Slack	From	To	From Clock	To Clock	Required Hold Relationship	Required Shortest P2P Time	Actual Shortest P2P Time
1	Not operational: Clock Skew > Data Delay	vending_machine:vm|present_state.st_idle	vending_machine:vm|candy_dispensed	clk	clk	None	None	2.490 ns

Εικόνα 27 Μήνυμα σφάλματος

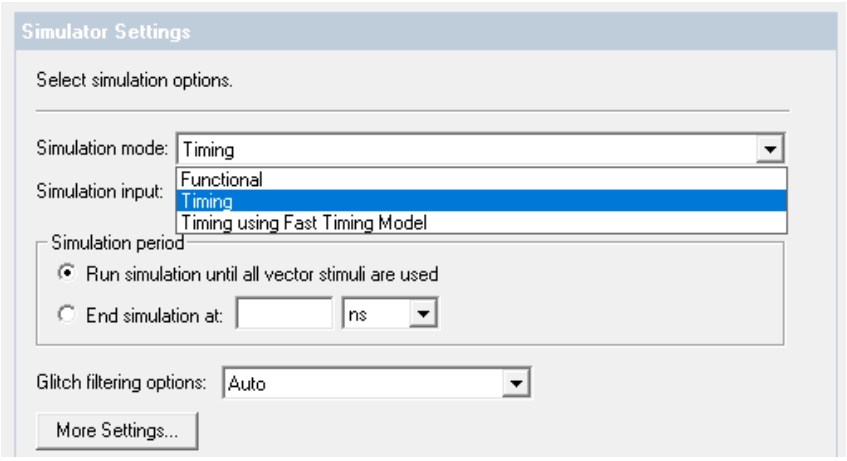
Για να μειωθούν τα μηνύματα προειδοποίησης, προσδιορίστηκαν από τον RTL Viewer τα latches ή αλλιώς οι μανδαλωτές, οι οποίοι δημιουργούνταν κάθε φορά που δεν άρεσε στον κώδικα συγκεκριμένο κομμάτι. Για τον εντοπισμό του προβλήματος και την αποσφαλμάτωση του πρέπει να ακολουθηθεί η διαδρομή των εισόδων του latch έως το σημείο που τερματίζει, και να εντοπιστεί και στον κώδικα το πιθανό λάθος.



Εικόνα 28 Μανδαλωτής

5.4 Αποτελέσματα Προσομοίωσης

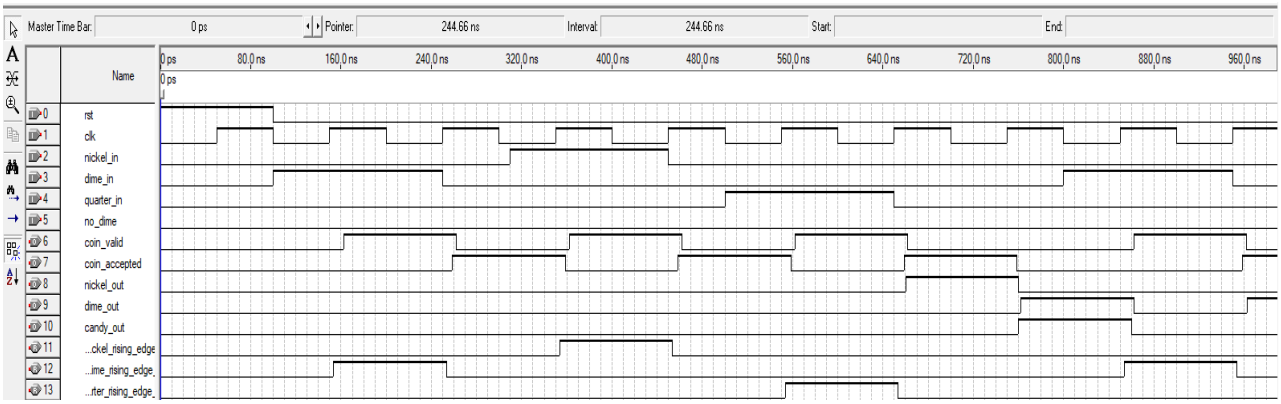
Η υλοποίηση και εκτέλεση της προσομοίωσης πραγματοποιήθηκαν με την χρήση της οικογένειας Cyclone II και την συσκευή EP2C35F672C8. Η οποία επιλέχτηκε με βάση τις απαιτήσεις των αναγκών του project και για τον λόγο ότι παρέχει προγραμματιζόμενα λογικά στοιχεία τα οποία μπορούν να διαμορφωθούν και να διασυνδεθούν μεταξύ τους για τη δημιουργία προσαρμοσμένων ψηφιακών κυκλωμάτων. Κατά την προσομοίωση με την χρήση του Waveform editor προτιμήθηκε η ρύθμιση της προσομοίωσης χρονισμού(Timing) αντί της λειτουργικής(Functional), διότι έχει την δυνατότητα να παρέχει μια ακριβέστερη αναπαράσταση της συμπεριφοράς του κυκλώματος, με το να δημιουργεί καθυστερήσεις κατά την φόρτωση των διαφόρων καταχωρητών καθώς και στην παραγωγή έγκυρων σημάτων στις θύρες εξόδου. Αντιθέτως, η λειτουργική προσομοίωση επικεντρώνεται στον έλεγχο της συμπεριφοράς του κυκλώματος χωρίς να λαμβάνονται υπόψη οι χρονικές καθυστερήσεις που μπορεί να προκαλούνται από φυσικά στοιχεία όπως πύλες, flip-flops και διασυνδέσεις.



Εικόνα 29 Ρυθμίσεις του προσομοιωτή

Είναι σημαντικό να αναφερθεί επίσης ότι σε όλα τα αποτελέσματα προσομοίωσης έχει ρυθμιστεί η περίοδος του ρολογιού στα 50ns, ενώ το ασύγχρονο reset είναι ενεργό για τα πρώτα 100ns. Στην παρακάτω προσομοίωση παρουσιάζεται μέσω του waveform του Quartus II η λειτουργία του αυτόματου πωλητή στην περίπτωση που τοποθετούνται κέρματα με την σειρά:

>10(dime_in) + 5(nickel_in) + 25(quarter_in) = 40. Οπότε εφόσον δεν είναι ενεργοποιημένο το σήμα ‘no_dime’ η μηχανή δίνει για ρέστα: 5(nickel_out) + 10(dime_out) + candy_out διότι συμπληρώνεται το ποσό που απαιτείται για να διανεμηθεί το γλυκό. Σε όλη την διαδικασία μπορεί να παρατηρηθεί ότι τα ‘coin_valid’ + ‘coin_accepted’ συνεργάζονται για να αποδεχτούν μία έγκυρη είσοδο κέρματος. Τέλος, μπορεί να φανεί ότι γίνεται μία γρήγορη τοποθέτηση ενός 10λεπτου(dime_in) κέρματος, το οποίο η μηχανή το απορρίπτει λόγο γρήγορης τοποθέτησης, επιστρέφοντας το σαν ρέστα (dime_out).

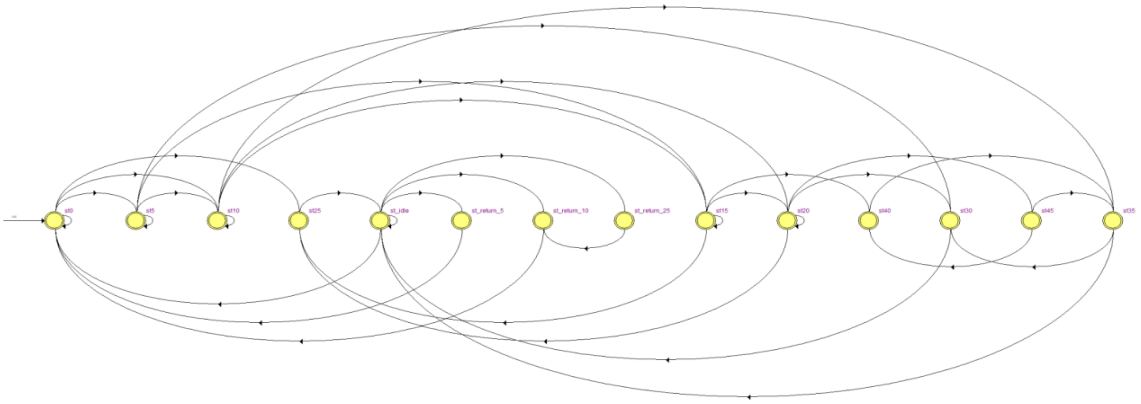


Εικόνα 30 simulationwithwaveform.vwf

6 Ο κώδικας του αυτόματου πωλητή

6.1 States

Η μηχανή καταστάσεων αποτελεί ένα από τα σημαντικότερα στοιχεία του project. Με τη χρήση του εργαλείου State Machine Viewer που παρέχεται από το Quartus II, μπορούμε να κατανοήσουμε ακόμη καλύτερα τον σχεδιασμό. Αυτό το εργαλείο παρέχει τη δυνατότητα απεικόνισης των καταστάσεων, των μεταβάσεων και της σχετικής λογικής της μηχανής. Επιπλέον, με το διάγραμμα καταστάσεων μπορούμε να αναλύσουμε τη συμπεριφορά και τη ροή της μηχανής σε κάθε πιθανή κατάσταση που έχει οριστεί στον σχεδιασμό.



Εικόνα 31 Διάγραμμα καταστάσεων μηχανής

Οι συνθήκες που υποδεικνύονται στον πίνακα με το θαυμαστικό (!) υποδηλώνουν την άρνηση ή το αντίστροφο της συγκεκριμένης συνθήκης. Επίσης, όπως φαίνεται και στον κώδικα(εικόνα 32) οι τιμές nin, din, qin είναι τα boolean σήματα που δηλώνουν ότι ένα κέρμα είναι έγκυρο.

```
29    nin <= coin_valid = '1' and nickel_in;
30    din <= coin_valid = '1' and dime_in;
31    qin <= coin_valid = '1' and quarter_in;
```

Εικόνα 32 Δήλωση σημάτων

SourceState	DestinationState	Condition
st0	st0	(!qin).(!din).(!nin)
st0	st5	(nin)
st0	st10	(din).(!nin)
st0	st25	(!nin).(!din).(qin)
st5	st5	(!qin).(!din).(!nin)
st5	st10	(nin)
st5	st15	(din).(!nin)
st5	st30	(!din).(!nin).(qin)
st10	st10	(!qin).(!din).(!nin)
st10	st15	(nin)
st10	st20	(din).(!nin)
st10	st35	(qin).(!din).(!nin)
st15	st15	(!qin).(!din).(!nin)
st15	st20	(nin)
st15	st25	(!nin).(din)

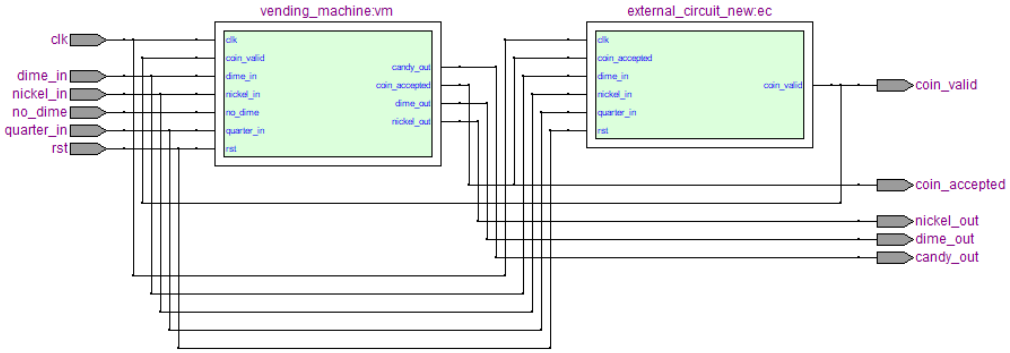
st15	st40	(qin).(!din).(!nin)
st20	st20	(!qin).(!din).(!nin)
st20	st25	(nin)
st20	st30	(din).(!nin)
st20	st45	(qin).(!din).(!nin)
st25	st_idle	
st30	st_idle	
st35	st30	(no_dime)
st35	st_idle	(!no_dime)
st40	st35	
st45	st35	(!no_dime)
st45	st40	(no_dime)
st_idle	st0	(!counter[0]).(counter[1])
st_idle	st_idle	(!qin).(!din).(!nin).(!counter[0]).(!counter[1])+ (!qin).(!din).(!nin).(counter[0])
st_idle	st_return_5	(nin).(!counter[0]).(!counter[1]) + (nin).(counter[0])
st_idle	st_return_10	(din).(!nin).(!counter[0]).(!counter[1])+ (din).(!nin).(counter[0])
st_idle	st_return_25	(qin).(!din).(!nin).(!counter[0]).(!counter[1])+ (qin).(!din).(!nin).(counter[0])
st_return_5	st0	
st_return_10	st0	

st_return_25	st_return_10	

Πίνακας 2 Μεταβάσεις καταστάσεων

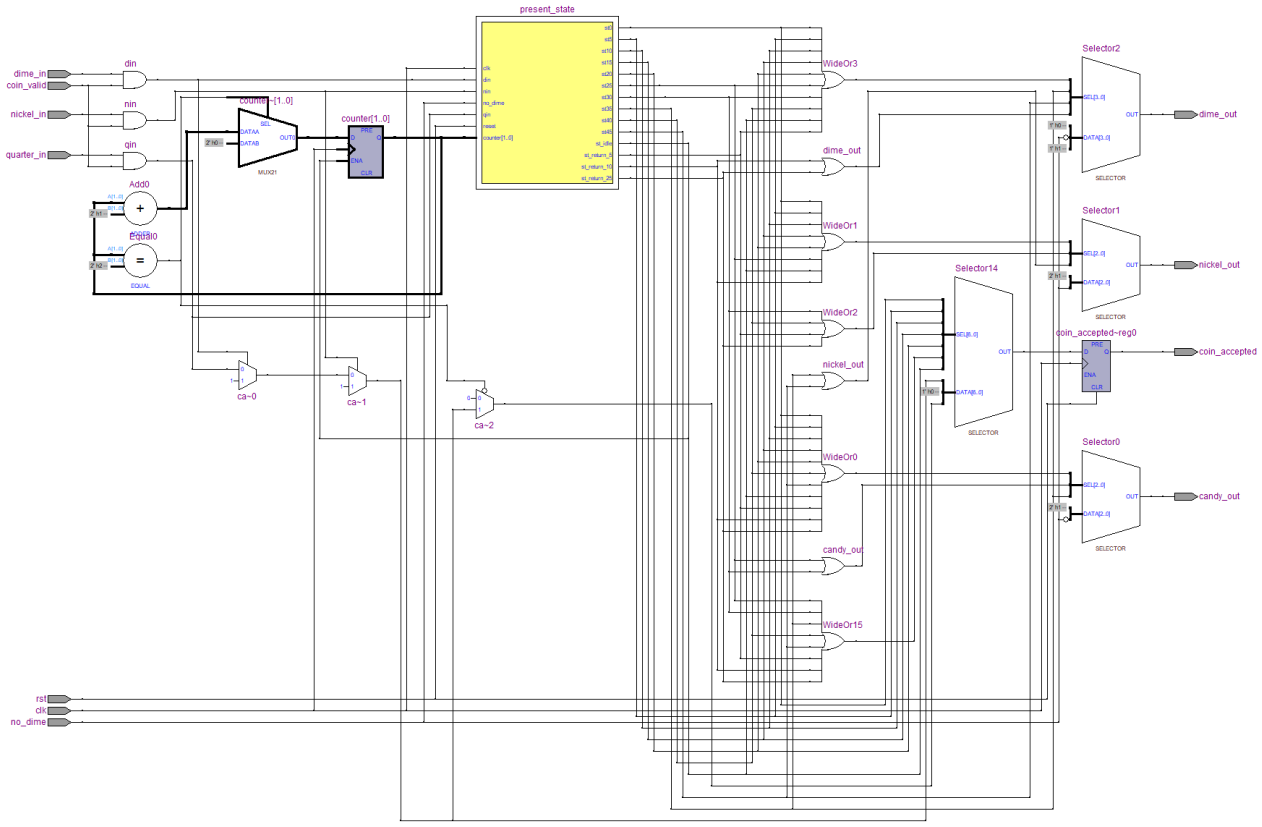
6.2 Schematic& RTL

Το επίπεδο μεταφοράς καταχωρητών RTL είναι ένα επίπεδο περιγραφής μιας ψηφιακής σχεδίασης στο οποίο η ωρολογιακή συμπεριφορά της σχεδίασης περιγράφεται ρητά με όρους μεταφοράς δεδομένων μεταξύ στοιχείων αποθήκευσης σε ακολουθιακή λογική, η οποία μπορεί να υπονοείται, και συνδυαστική λογική, η οποία μπορεί να αντιπροσωπεύει οποιαδήποτε λογική υπολογιστικών ή αριθμητικών λογικών μονάδων. Η μοντελοποίηση RTL επιτρέπει την ιεραρχία σχεδιασμού που αντιπροσωπεύει μια δομική περιγραφή άλλων μοντέλων RTL. Πρώτα απεικονίζεται στην εικόνα 33 το blockdiagram του κώδικα machine.vhd με την συνολική δομή σχεδιασμού του project. Σε αυτό το blockdiagram η οντότητα ανώτερου επιπέδου "machine" ενσαρκώνει τα δύο components και τα διασυνδέει μεταξύ τους.



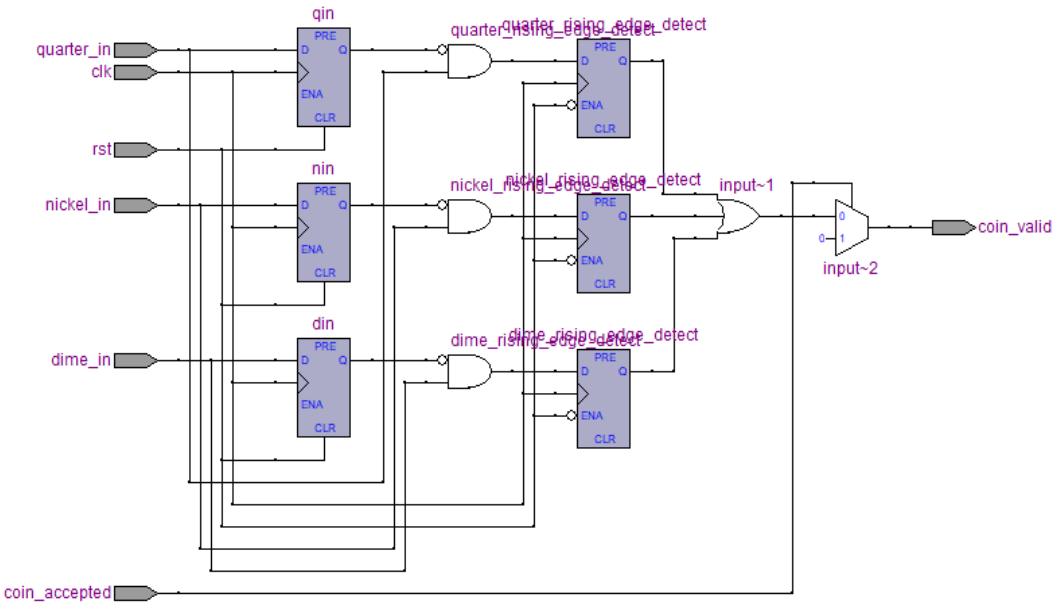
Εικόνα 33 RTL Viewer: Machine

Με την απλή ενέργεια του κλικ πάνω σε ένα από τα δύο μπλοκ, ο RTL viewer μεταβαίνει στο εσωτερικό του επιλεγμένου κώδικα. Η αρχιτεκτονική RTL του vending machine (εικόνα 34) αποτελείται από καταχωρητές και συνδυαστικά λειτουργικά μπλοκ (όπως οι αθροιστές και πολυπλέκτες) τα οποία καθορίζουν την διαδρομή δεδομένων. Η μηχανή πεπερασμένων καταστάσεων ξεχωρίζεται με το κίτρινο χρώμα, και είναι ο ελεγκτής που ελέγχει την μεταφορά δεδομένων μέσω των λειτουργικών μπλοκ και μεταξύ των καταχωρητών.



Εικόνα34 RTL Viewer: Vending Machine

Από την άλλη πλευρά η σχεδίαση RTL του External_circuit_new (εικόνα 35) είναι πιο απλή καθώς μεταφέρει τα σήματα μέσω των καταχωρητών για να εντοπίσει τις θετικές και αρνητικές ακμές των κερμάτων qin,din,nin. Με έναν πολυπλέκτη καθορίζει την έξοδο του coin_valid. Το αποτέλεσμα της σύνθεσης της διαδρομής δεδομένων είναι:

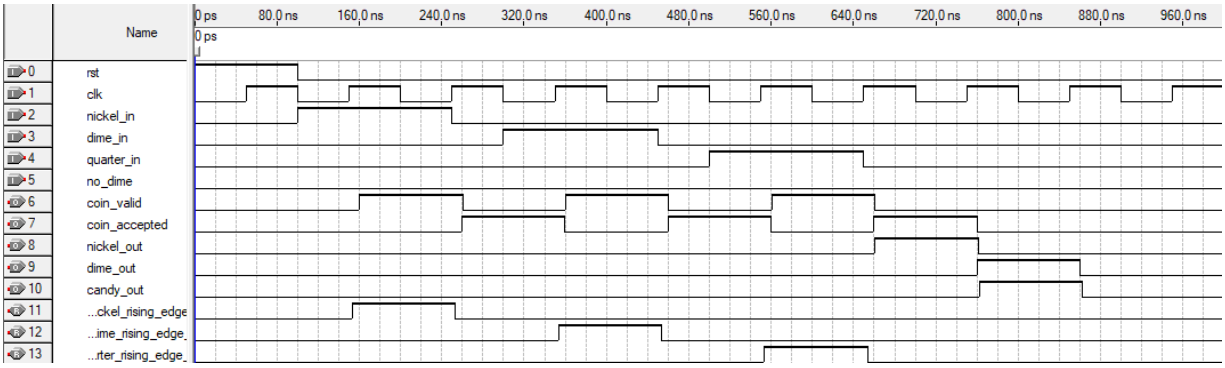


Εικόνα 34 RTL Viewer: External Circuit

6.3 Προσομοίωση με waveform

6.3.1 Απλή προσομοίωση

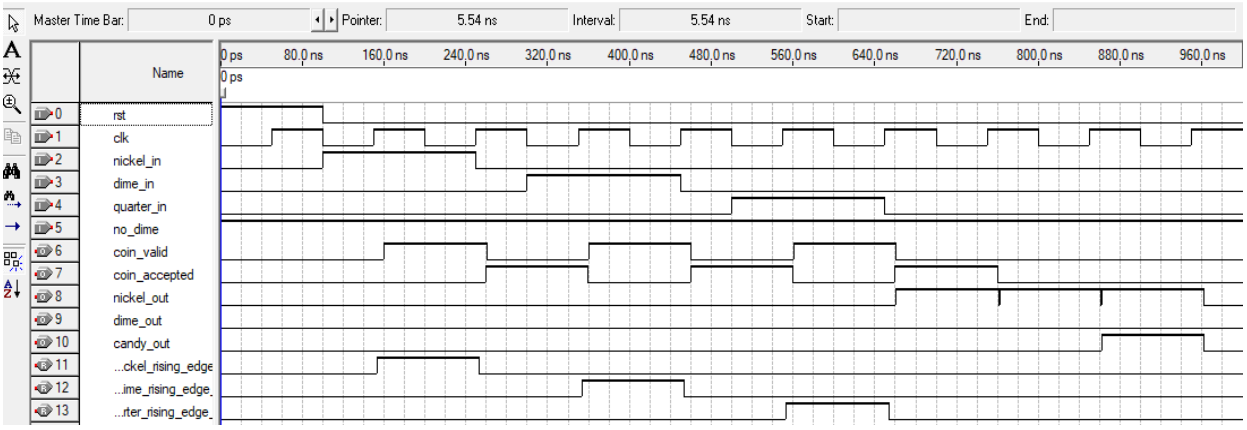
Υλοποίηση:>5(nickel_in)+10(dime_in)+25(quarter_in)=40
out:>>5(nickel_out)+10(dime_out)+(candy_out)



Εικόνα 35 simple simulation.vwf

6.3.2 Προσομοίωση όταν τοποθετηθούν 40 λεπτά και έχουν τελειώσει τα 10 λεπτά για ρέστα

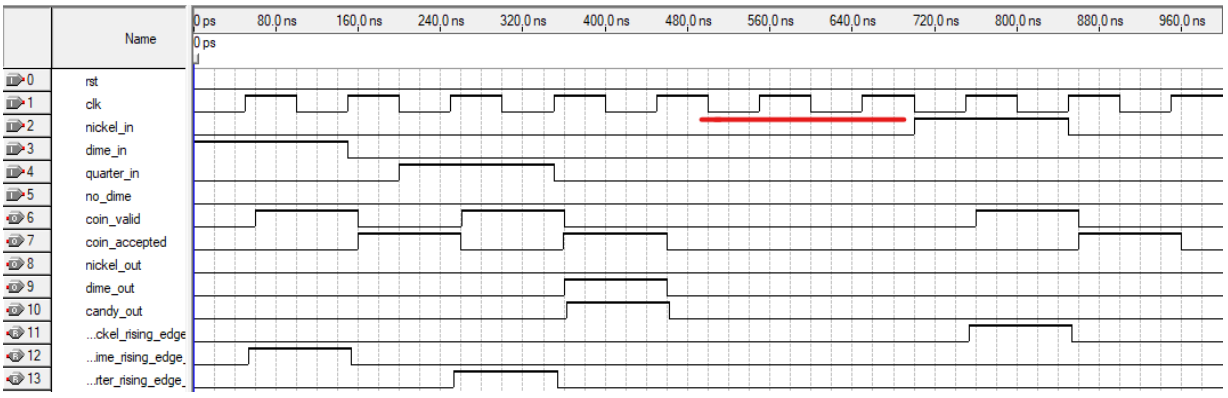
Υλοποίηση:>5(nickel_in)+10(dime_in)+25(quarter_in)=40
out: >>5(nickel_out) + 5(nickel_out) + 5(nickel_out) + (candy_out)



Εικόνα 36 simple simulation with no dime enabled.vwf

6.3.3 Προσομοίωση όταν συνεχίζει να δέχεται παραπάνω κέρματα ακόμα και όταν έχει καταβληθεί το ποσό

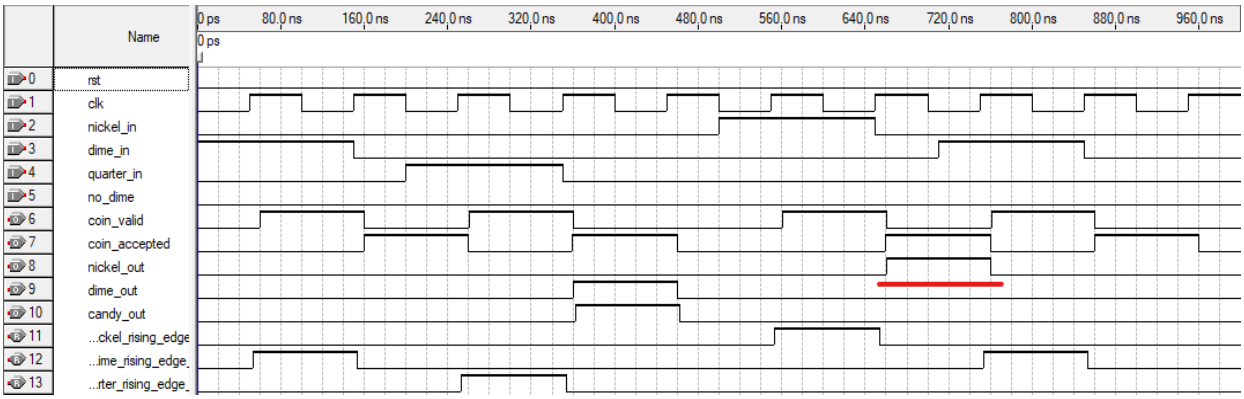
Υλοποίηση:>10(dime_in)+25(quarter_in)=35
out:>>10(dime_out)+(candy_out)
next: >5(nickel_in) Τοποθετήθηκε μετά από 2 κύκλους του ρολογιού για να μην απορριφτεί από την μηχανή



Εικόνα 37 simulation when it accepts coins after a candy dispension.vwf

6.3.4 Προσομοίωση όταν πραγματοποιηθείγρήγορη τοποθέτηση κέρματος, ύστερα από διανομή γλυκού

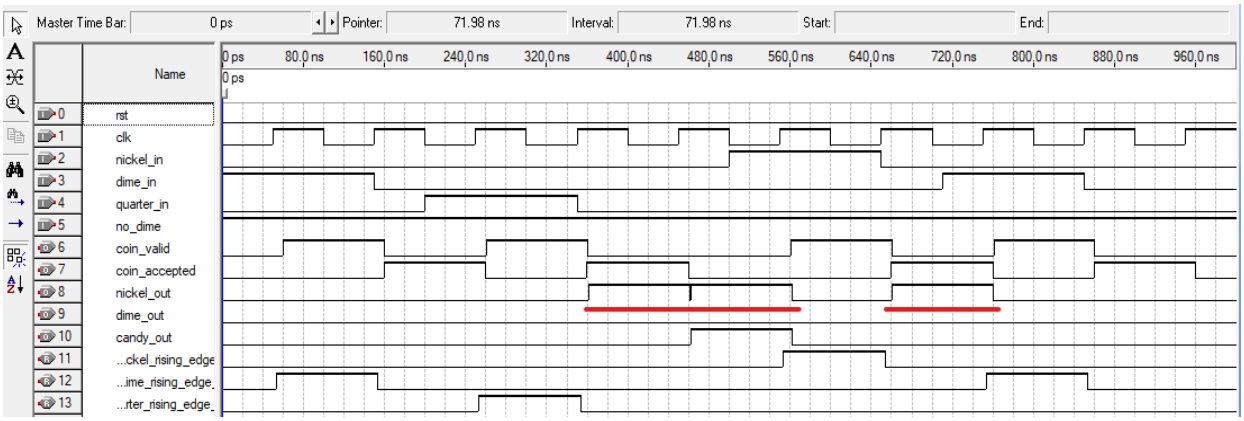
Υλοποίηση:>10(dime_in)+25(quarter_in)=35
out:>>10(dime_out)+(candy_out)
next:>5(nickel_in)
out:>>5(nickel_out)Απορρίφθηκε
next: >10(dime_in)



Εικόνα 38 Simulation when quick coin insertion is made.vwf

Ας δούμε πως αλλάζουν οι έξοδοι όταν ενεργοποιηθεί και το σήμα no_dime στην παραπάνω προσομοίωση

Υλοποίηση:>10(dime_in)+25(quarter_in)=35
out:>>5(nickel_in)+5(nickel_in)+(candy_out)
next:>5(nickel_in)
out:>>5(nickel_out)Απορρίφτηκε
next: >10(dime_in)

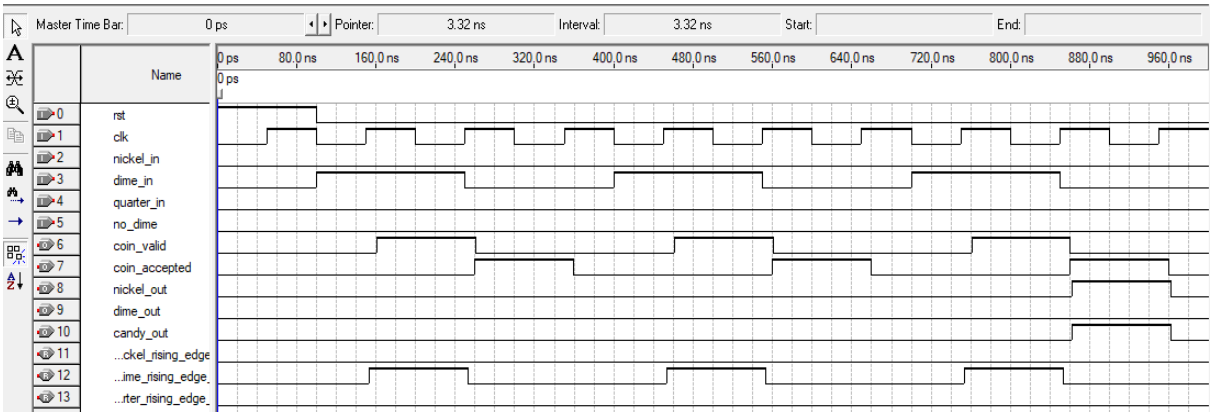


Εικόνα 39 Simulation when quick coin insertion is made and no dime is enabled.vwf

6.3.5 Προσομοίωση όταν τοποθετούνται ίδια κέρματα

Υλοποίηση:>10(dime_in)+10(dime_in)+10(dime_in)=30

out:>>5(nickel_out)+(candy_out)



Εικόνα 40 simulation when the same coins inserted.vwf

6.3.6 Προσομοίωση σε 50.0nsκύκλους του ρολογιού περίοδο

Υλοποίηση:>5(nickel_in)+10(dime_in)+5(nickel_in)=15

Τοδεύτερο 5λεπτο δεν γίνεται αποδεκτό γιατί τοποθετήθηκε ενώ έκανε ακόμα επεξεργασία το 10λεπτο η μηχανή του αυτόματου πωλητή

next:>25(quarter_in)=15+25=40

out:>>5(nickel_out)+10(dime_out)+(candy_out)

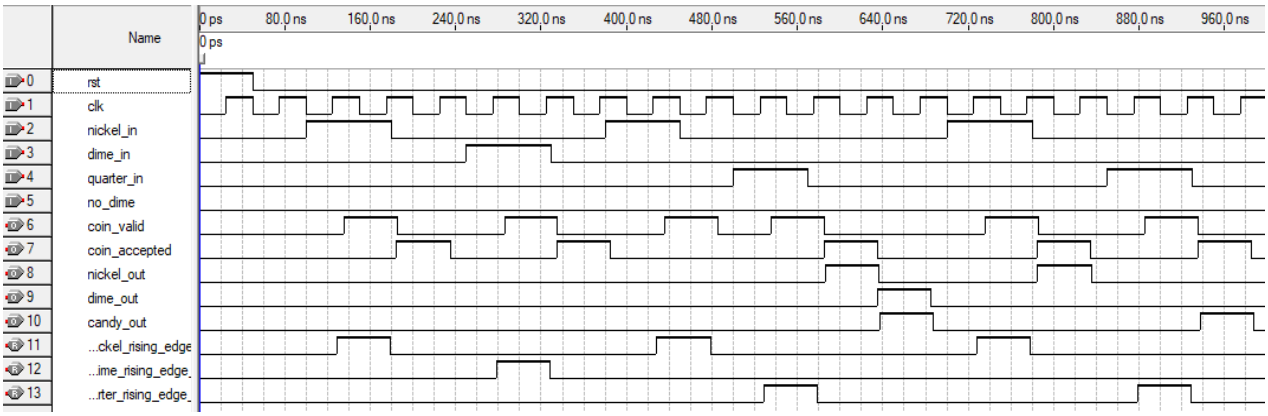
next:>5(nickel_in)

out:>>5(nickel_out)

Απορρίπτεται γιατί τοποθετήθηκε ενώ η μηχανή βρίσκεται σε κατάσταση αδράνειας για 2 κύκλους του ρολογιού

next:>25(quarter_in)

out: >>(candy_out)



Εικόνα 41 simulation on 50ns clock circles.vwf

7 Συμπέρασμα

Εν ολίγοις, το κύκλωμα του αυτόματου πωλητή που υλοποιήθηκε σε γλώσσα VHDL αποδεικνύει την ικανότητα προσομοίωσης της συμπεριφοράς ενός πραγματικού απλού αυτόματου πωλητή. Ο σχεδιασμός χρησιμοποίησε τις έννοιες των μηχανών πεπερασμένων καταστάσεων, της συνδυαστικής και της ακολουθιακής λογικής. Καθώς και πραγματοποιήθηκε η χρήση των πολυπλεκτών και καταχωρητών για τον έλεγχο της ροής των δεδομένων και των ενεργειών εντός του συστήματος. Συνοψίζοντας οι μηχανισμοί ασφαλείας καθόρισαν το βασικότερο κομμάτι του κώδικα, με την εφαρμογή του σχεδιασμού ενός μηχανισμού επικύρωσης κερμάτων και αποδοχής κερμάτων. Βασίζοντας τον συνδυασμό του μηχανισμού και τη λειτουργία των καταστάσεων της μηχανής, ο αυτόματος πωλητής εξασφαλίζει ασφαλείς και ακριβείς συναλλαγές. Το project παρουσίασε επίσης τις προσομοιώσεις με την χρήση κυμματομορφών σε διάφορες περιπτώσεις τοποθέτησης κερμάτων, ώστε να επαληθευτεί ότι η εφαρμογή του κώδικα αντικατοπτρίζει τις προδιαγραφές του σχεδιασμού. Σε γενικές γραμμές, η επίτευξη αυτού του project αναδεικνύει τις δυνατότητες της γλώσσας VHDL στη σχεδίαση σύνθετων-πολύπλοκων ψηφιακών συστημάτων όπως είναι οι αυτόματοι πωλητές. Με περαιτέρω επεξεργασίες και βελτιστοποιήσεις, το project ίσως μπορέσει να επεκταθεί για να ενσωματώσει πρόσθετα χαρακτηριστικά και λειτουργίες, καθιστώντας το ικανό να προχωρήσει σε φυσική υλοποίηση σε ένα ολοκληρωμένο κύκλωμα ,όπως είναι το FPGA.

ΠΑΡΑΡΤΗΜΑ

Α. Κώδικας

Vending_machince.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity vending_machine is
    port ( clk, rst : in std_logic;
           nickel_in, dime_in, quarter_in : in boolean;
           no_dime : in std_logic;
           candy_out, nickel_out, dime_out : out std_logic;
           coin_valid : in std_logic;
           coin_accepted : out std_logic);
end vending_machine;

architecture fsm of vending_machine is

---------------- States--------------------------------
    type state is (st0, st5, st10, st15, st20, st25, st30, st35, st40, st45, st_idle, st_return_5, st_return_10, st_return_25);
    signal present_state, next_state : state;

---------------- Signals ----------------------------
    signal ca, cv, nd: std_logic;
    signal nin, din, qin: boolean;
    signal candy_dispensed : std_logic;
    signal counter : integer range 0 to 2;


begin

nin <= coin_valid = '1' and nickel_in;
din <= coin_valid = '1' and dime_in;
qin <= coin_valid = '1' and quarter_in;
nd <= no_dime;



process (rst, clk)
    begin
        if (rst='1') then
            coin_accepted <= '0';
        elsif (clk'EVENT and clk='1') then
            coin_accepted <= ca;
        end if;
end process;

process (clk)
    begin
        if(candy_dispensed ='1') then
            if (clk'EVENT and clk='1') then
                counter <= counter + 1;
                    if (counter = 2) then
                    counter <= 0;
                    end if;
            end if;
        end if;
end process;

---- katw tmhma ths FSM : ------------------
process (rst, clk)
    begin
        if (rst='1') then
            present_state <= st0;
        elsif (clk'EVENT and clk='1') then
            present_state <= next_state;
        end if;
end process;
---- panw tmhma ths FSM : ------------------
    process (present_state, nickel_in, dime_in, quarter_in, no_dime)
    begin
        ca <= '0';
        case present_state is
        when st0 =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '0';
            candy_dispensed <= '0';
            if (nin) then next_state <= st5; ca <= '1';
            elsif (din) then next_state <= st10; ca <= '1';
            elsif (qin) then next_state <= st25; ca <= '1';
            else next_state <= st0;
            end if;
        when st5 =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '0';
            candy_dispensed <= '0';
            if (nin) then next_state <= st10; ca <= '1';
            elsif (din) then next_state <= st15; ca <= '1';
            elsif (qin) then next_state <= st30; ca <= '1';
            else next_state <= st5;
            end if;
        when st10 =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '0';
            candy_dispensed <= '0';
            if (nin) then next_state <= st15; ca <= '1';
            elsif (din) then next_state <= st20; ca <= '1';
            elsif (qin) then next_state <= st35; ca <= '1';
            else next_state <= st10;
            end if;
        when st15 =>
            candy_out <= '0';
            nickel_out <= '0';
            dime_out <= '0';
            candy_dispensed <= '0';
            if (nin)  then next_state <= st20; ca <= '1';
```

```
106             elsif (din) then next_state <= st25; ca <= '1';
107             elsif (qin) then next_state <= st40; ca <= '1';
108             else next_state <= st15;
109             end if;
110         when st20 =>
111             candy_out <= '0';
112             nickel_out <= '0';
113             dime_out <= '0';
114             candy_dispensed <= '0';
115             if (nin) then next_state <= st25; ca <= '1';
116             elsif (din) then next_state <= st30; ca <= '1';
117             elsif (qin) then next_state <= st45; ca <= '1';
118             else next_state <= st20;
119             end if;
120         when st25 =>
121             candy_out <= '1';
122             nickel_out <= '0';
123             dime_out <= '0';
124             candy_dispensed <= '0';
125             next_state <= st_idle;
126         when st30 =>
127             candy_out <= '1';
128             nickel_out <= '1';
129             dime_out <= '0';
130             candy_dispensed <= '0';
131             next_state <= st_idle;
132         when st35 =>
133             if (nd='1') then
134             candy_out <= '0';
135             nickel_out <= '1';
136             dime_out <= '0';
137             candy_dispensed <= '0';
138             next_state <= st30;
139             else
140             candy_out <= '1';
141             nickel_out <= '0';
142             dime_out <= '1';
143             candy_dispensed <= '0';
144             next_state <= st_idle;
145             end if;
146         when st40 =>
147             candy_out <= '0';
148             nickel_out <= '1';
149             dime_out <= '0';
150             candy_dispensed <= '0';
151             next_state <= st35;
152         when st45 =>
153             if (nd='1') then
154             candy_out <= '0';
155             nickel_out <= '1';
156             dime_out <= '0';
157             candy_dispensed <= '0';
158             next_state <= st40;
159             else
160             candy_out <= '0';
161             nickel_out <= '0';
162             dime_out <= '1';
163             candy_dispensed <= '0';
164             next_state <= st35;
165             end if;
166         when st_idle =>
167             candy_out <= '0';
168             nickel_out <= '0';
169             dime_out <= '0';
170             candy_dispensed <= '1';
171             if (counter /= 2) then
172                 if (nin) then next_state <= st_return_5; ca <= '1';
173                 elsif (din) then next_state <= st_return_10; ca <= '1';
174                 elsif (qin) then next_state <= st_return_25; ca <= '1';
175                 else next_state <= st_idle;
176                 end if;
177             else next_state <= st0;
178             end if;
179         when st_return_5 =>
180             candy_out <= '0';
181             nickel_out <= '1';
182             dime_out <= '0';
183             candy_dispensed <= '0';
184             next_state <= st0;
185         when st_return_10 =>
186             candy_out <= '0';
187             nickel_out <= '0';
188             dime_out <= '1';
189             candy_dispensed <= '0';
190             next_state <= st0;
191         when st_return_25 =>
192             candy_out <= '0';
193             nickel_out <= '1';
194             dime_out <= '1';
195             candy_dispensed <= '0';
196             next_state <= st_return_10;
197         end case;
198     end process;
199 ------------------------------------------------------------
200
201 end fsm;
202
```

External_circuit_new.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity external_circuit_new is
5      port (clk, rst : in std_logic;
6              coin_accepted : in std_logic;
7              nickel_in, dime_in, quarter_in : in boolean;
8              coin_valid : out std_logic);
9  end external_circuit_new;
10
11 architecture schematic of external_circuit_new is
12
13 SIGNAL input : boolean;
14 signal nin, din, qin: boolean;
15 signal nickel_rising_edge_detect, dime_rising_edge_detect, quarter_rising_edge_detect : boolean;
16
17 begin
18
19 process (rst, clk)
20     begin
21         if (rst='1') then
22             nin <= false;
23         elsif (clk'EVENT and clk='1') then
24             nin <= nickel_in;
25             nickel_rising_edge_detect <= nickel_in and (not nin);
26         end if;
27 end process;
28
29 process (rst, clk)
30     begin
31         if (rst='1') then
32             din <= false;
33         elsif (clk'EVENT and clk='1') then
34             din <= dime_in;
35             dime_rising_edge_detect <= dime_in and (not din);
36         end if;
37 end process;
38
39 process (rst, clk)
40     begin
41         if (rst='1') then
42             qin <= false;
43         elsif (clk'EVENT and clk='1') then
44             qin <= quarter_in;
45             quarter_rising_edge_detect <= quarter_in and (not qin);
46         end if;
47 end process;
48
49 input <= false when (coin_accepted = '1') else (nickel_rising_edge_detect or dime_rising_edge_detect or quarter_rising_edge_detect);
50 coin_valid <= '1' when input else '0';
51
52 end schematic;
53
```

Machine.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity machine is
5      port ( clk, rst : in std_logic;
6           coin_valid : out std_logic;
7           no_dime : in std_logic;
8           nickel_in, dime_in, quarter_in : in boolean;
9           candy_out, nickel_out, dime_out : out std_logic;
10          coin_accepted : out std_logic);
11 end machine;
12
13 architecture fsm of machine is
14
15
16 component vending_machine is
17     port(
18          clk, rst : in std_logic;
19          coin_valid : in std_logic;
20          no_dime : in std_logic;
21          nickel_in, dime_in, quarter_in : in boolean;
22          candy_out, nickel_out, dime_out : out std_logic;
23          coin_accepted : out std_logic
24     );
25     end component;
26
27 component external_circuit_new is
28     port(
29          clk, rst : in std_logic;
30          coin_accepted : in std_logic;
31          nickel_in, dime_in, quarter_in : in boolean;
32          coin_valid : out std_logic
33     );
34     end component;
35
36 signal cv, ca, nd : std_logic;
37 signal c_out, n_out, d_out : std_logic;
38
39 begin
40
41 candy_out <= c_out;
42 nickel_out <= n_out;
43 dime_out <= d_out;
44 coin_accepted <=ca;
45 coin_valid <= cv;
46
47
48 ec : external_circuit_new port map(clk=>clk, rst=>rst, nickel_in=>nickel_in, dime_in=>dime_in,
49       quarter_in=>quarter_in, coin_accepted=>ca, coin_valid=>cv);
50 vm : vending_machine port map(clk=>clk, rst=>rst, coin_valid=>cv, no_dime=>no_dime, nickel_in=>nickel_in,
51       dime_in=>dime_in, quarter_in=>quarter_in, candy_out=>c_out, nickel_out=>n_out, dime_out=>d_out, coin_accepted=>ca);
52
53
54 end fsm;
55
56
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Γκουντέλλης, Ο. (2019, Ιανουάριος). *Μηχανές Πεπερασμένων Καταστάσεων*.
2. Καλόμοιρος, Ι. Α. (2015). *Εισαγωγή στη γλώσσα VHDL*. http://teachers.cm.ihu.gr/kalomiros/Proigmena_Psifiaka_Theoria/didaktiko_yliko/Intro_VHDL_Kalomiros_teuxos2.pdf
3. Altera Corporation. (2008). *Cyclone II Device Handbook* (Volume 1 ed.).
4. *A Brief History of VHDL*. (n.d.). Doulos. https://www.doulos.com/knowhow/vhdl/a-brief-history-of-vhdl/
5. Chiuchisan, I., Potorac, A. D., & Graur, A. (2010). *Finite State Machine Design and VHDL Coding Techniques*. dasconference. http://www.dasconference.ro/cd2010/data/papers/C80.pdf
6. csun.edu. (n.d.). *Chapter Three :VHDL Fundamentals*. http://www.csun.edu/edaasic/roosta/ECE595_Chap3.pdf
7. csun.edu. (n.d.). *State Machine Encoding Techniques Synthesis*. http://www.csun.edu/edaasic/. http://www.csun.edu/edaasic/roosta/SME_Tech.pdf
8. ElProCus. (n.d.). *Finite State Machine (FSM) : Types, Properties, Design and Applications*. ElProCus. https://www.elprocus.com/finite-state-machine-mealy-state-machine-and-moore-state-machine/
9. Gonzana University. (n.d.). *VHDL: Packages and Components*. http://web02.gonzaga.edu/faculty/talarico/CP430/LEC/lec14-bari.pdf
10. *The History of VHDL*. (2018, May 17). HardwareBee. https://hardwarebee.com/the-history-of-vhdl/

11. Intel Corporation. (n.d.). *syn_encoding VHDL Synthesis Attribute*. https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/hdl/vhdl/vhdl_file_dir_syn_encoding.htm
12. Jensen Tech. (n.d.). *VHDL and FPGA terminology - Testbench*. VHDLwhiz. https://vhdlwhiz.com/terminology/testbench/
13. Joshi, D. (2020). *Testbenches in VHDL - A complete guide with steps*. Technobyte. https://technobyte.org/testbench-vhdl-types-examples-steps/
14. Kashani-Akhavan, S. (2017).*VHDL Testbench Tutorial*. https://moodlearchive.epfl.ch/2018-2019/pluginfile.php/1833321/mod_resource/content/1/vhdl_testbench_tutorial.pdf
15. Meeus, W. (2020). *Finite State Machine (FSM) encoding in VHDL: binary, one-hot, and others*. Sigasi Insights. https://insights.sigasi.com/tech/vhdl-onehot-fsm/
16. Minns, P., & Elliott, I. (2008). *FSM-based Digital Design using Verilog HDL*.
17. *ModelingConcepts*. (2013). xilinx university. https://www.xilinx.com/content/dam/xilinx/support/documents/university/ISE-Teaching/HDL-Design/14x/Nexys3/VHDL/docs-pdf/VHDL-Lab1.pdf
18. Ndjountche, T. (2016). *Digital Electronics 3: Finite-state Machines*. Wiley.
19. *Norsk versjon – Quartus II*. (n.d.). i.ntnu.no. https://i.ntnu.no/wiki/-/wiki/English/Quartus+II
20. Patel, M. K. (2017). *10. Testbenches — FPGA designs with VHDL documentation*. FPGA designs with VHDL. https://vhdlguide.readthedocs.io/en/latest/vhdl/testbench.html
21. Pedroni, V. A. (2004). *Circuit Design with VHDL* (3rd Edition ed.).
22. Pedroni, V. A. (2013). *Finite State Machines in Hardware: Theory and Design (with VHDL and SystemVerilog)*. MIT Press.
23. Perry, D. L. (2002). *VHDL: Programming By Example*. McGraw-HillEducation.
24. Prew, R. (2017). *Battle Over the FPGA: VHDL vs Verilog! Who is the True Champ?*Digilent. https://digilent.com/blog/battle-over-the-fpga-vhdl-vs-verilog-who-is-the-true-champ/

25. Thakur, S. (2023). *9 Differences Between Mealy And Moore Machine*. Unstop. https://unstop.com/blog/difference-between-mealy-and-moore-machine
26. *Verilog vs VHDL: Explain by Examples*. (n.d.). FPGA4student.com. https://www.fpga4student.com/2017/08/verilog-vs-vhdl-explain-by-example.html
27. *VHDL*. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/VHDL
28. *VHDL Modelling Styles: Behavioral, Dataflow, Structural*. (n.d.). Buzztech. https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/
29. *VHDL Tutorial*. (n.d.). Javatpoint. https://www.javatpoint.com/vhdl#History
30. Williams, D. (2015, December 23). *Implementing a Finite State Machine in VHDL*. All About Circuits. https://www.allaboutcircuits.com/technical-articles/implementing-a-finite-state-machine-in-vhdl/