ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΕΝΟΣ ΕΤΕΡΟΓΕΝΟΥΣ ΣΥΣΤΗΜΑΤΟΣ

Κωνσταντίνος Σκαμάγκας

Επιβλέπων: Φώτιος Βαρτζιώτης

Επίκουρος Καθηγητής, ΔΕΠ

Άρτα, Απρίλιος 2023

DESIGN AND IMPLEMENATION OF AN HETEROGENOUS SYSTEM

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 5 Ιουλίου 2023

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Φώτιος Βαρτζιώτης

2. Μέλος επιτροπής

   Γρηγόριος Δουμένης

3. Μέλος επιτροπής

   Σπυριδούλα Μαργαρίτη

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα προπτυχιακή 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.

Σκαμάγκας Κωνσταντίνος

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου που ήταν κοντά μου και με βοηθούσαν σε όλη την διάρκεια των σπουδών μου. Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Βαρτζιώτη που με υποστήριξε στην εκπόνηση της παρούσας πτυχιακής και ήταν πρόθυμος να λύσει όσες απορίες και προβλήματα είχα κατά την ανάλυση του θέματος που επιμελήθηκα.

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας πτυχιακής εργασίας ήταν ο σχεδιασμός και η υλοποίηση ενός ετερογενούς συστήματος με τη χρήση του Intel DevCloud και του OneAPI. Η έρευνα περιελάμβανε τη διερεύνηση των δυνατοτήτων του OneAPI, ενός ενοποιημένου μοντέλου προγραμματισμού λογισμικού που έχει σχεδιαστεί για την απλοποίηση της ανάπτυξης σε μια σειρά ετερογενών αρχιτεκτονικών. Το έργο χρησιμοποίησε επίσης το Intel DevCloud, μια πλατφόρμα βασισμένη στο cloud που παρέχει πρόσβαση σε μια σειρά αρχιτεκτονικών υλικού της Intel, όπως CPUs, GPUs, FPGAs και επιταχυντές AI. Η πτυχιακή περιγράφει λεπτομερώς τη διαδικασία σχεδιασμού, η οποία περιελάμβανε την επιλογή των κατάλληλων στοιχείων υλικού και λογισμικού, τη διαμόρφωση του συστήματος και τη βελτιστοποίηση των επιδόσεων. Η φάση της υλοποίησης περιελάμβανε την ανάπτυξη και δοκιμή ενός αριθμού εφαρμογών με τη χρήση του μοντέλου προγραμματισμού OneAPI. Οι εφαρμογές σχεδιάστηκαν για να παρουσιάσουν τις δυνατότητες του ετερογενούς συστήματος, συμπεριλαμβανομένης της ικανότητας αποδοτικής αξιοποίησης διαφορετικών τύπων υλικού και της κλιμάκωσης σε πολλαπλούς κόμβους. Τα αποτελέσματα του έργου καταδεικνύουν τις δυνατότητες του OneAPI και του Intel DevCloud ως πλατφόρμας για την ανάπτυξη εφαρμογών σε μια σειρά ετερογενών αρχιτεκτονικών. Συνοπτικά, η πτυχιακή παρέχει μια πολύτιμη συμβολή στον τομέα των ετερογενών υπολογιστών και καταδεικνύει τις δυνατότητες αυτής της τεχνολογίας για την αντιμετώπιση μιας σειράς υπολογιστικών προκλήσεων.

ABSTRACT

The purpose of this thesis was the design and implementation of a heterogeneous system using Intel DevCloud and OneAPI. The research involved exploring the capabilities of OneAPI, a unified software programming model designed to simplify development across a range of heterogeneous architectures. The project also used Intel DevCloud, a cloud-based platform that provides access to a range of Intel hardware architectures, including CPUs, GPUs, FPGAs, and AI accelerators. The thesis details the design process, which included selecting the appropriate hardware and software components, configuring the system, and optimizing performance. The implementation phase included the development and testing of a number of applications using the OneAPI programming model. The applications were designed to demonstrate the capabilities of the heterogeneous system, including the ability to efficiently utilize different types of hardware and scale to multiple nodes. The project results demonstrate the potential of OneAPI and Intel DevCloud as a platform for developing applications across a range of heterogeneous architectures. In summary, the thesis provides a valuable contribution to the field of heterogeneous computing and demonstrates the potential of this technology to address a range of computational challenges.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

1. Εισαγωγή

1.1 Πληροφορίες για τα ετερογενή συστήματα

Τα ετερογενή συστήματα στην επιστήμη των υπολογιστών είναι συστήματα τα οποία χρησιμοποιούν πάνω από ένα είδος επεξεργαστή. Αυτά τα συστήματα καταφέρνουν να έχουν καλύτερη απόδοση ή κατανάλωση με την χρήση διαφορετικών συνεπεξεργαστών που συνήθως ενσωματώνουν εξειδικευμένες δυνατότητες επεξεργασίας για τη διεκπεραίωση συγκεκριμένων εργασιών. Συνήθως η ετερογένεια στο πλαίσιο των υπολογιστών αναφέρεται στην λειτουργία διαφορετικών σετ εντολών αρχιτεκτονικής, όπου ο κύριος επεξεργαστής και οι υπόλοιποι επεξεργαστές έχουν πολύ διαφορετική αρχιτεκτονική. [1]

Ένα κατανεμημένο σύστημα περιέχει πολλά διαφορετικά είδη υλικού και λογισμικού που δουλεύουν ταυτόχρονα για την επίλυση προβλημάτων. Μπορεί να υπάρχουν πολλές διαφορετικές αναπαραστάσεις δεδομένων στο σύστημα. Αυτό μπορεί να περιλαμβάνει διαφορετικές αναπαραστάσεις για ακέραιους αριθμούς, ροές byte, αριθμούς κινητής υποδιαστολής και σύνολα χαρακτήρων. Τα περισσότερα από τα δεδομένα μπορούν να ομαδοποιηθούν από το ένα σύστημα στο άλλο χωρίς να χάσουν την σημασία τους. Οι προσπάθειες παροχής μιας καθολικής κανονικής μορφής πληροφοριών καθυστερούν σε απόδοση.

Μπορεί να υπάρχουν πολλά διαφορετικά σετ οδηγιών. Μια εφαρμογή που έχει μεταγλωττιστεί για ένα σύνολο εντολών δεν μπορεί να εκτελεστεί εύκολα σε υπολογιστή με άλλο σύνολο εντολών, εκτός εάν παρέχεται διερμηνέας συνόλου εντολών. Οι πρόσφατες εξελίξεις στον ιστό και την Java ενδέχεται να παρέχουν μια καθολική γλώσσα ερμηνείας στους περισσότερους υπολογιστές. Αν και μια γλώσσα υπολογιστή δεν είναι ένα σύνολο οδηγιών, αυτός είναι ένας καλός συμβιβασμός. [2]

Τα τελευταία χρόνια, ο ορισμός του ετερογενούς υπολογισμού έχει επεκταθεί στο να περιλαμβάνει επεξεργαστές που βασίζονται σε διαφορετικές αρχιτεκτονικές υπολογιστών. Για παράδειγμα, επεξεργαστές που βασίζονται στην αρχιτεκτονική ARM μπορεί να είναι καλύτεροι για κάποιες συγκεκριμένες διεργασίες, χάρη στον υψηλό αριθμό πυρήνων που έχουν, καθώς και στην καλύτερη κατανάλωση ισχύος και συμβατότητα με άλλες συσκευές που βασίζονται στην ίδια αρχιτεκτονική. Η υιοθέτηση μιας εναλλακτικής αρχιτεκτονικής μπορεί να φέρει εξυπνότερους τρόπους διαχείρισης των ήδη υπαρχόντων φόρτων εργασίας και υπολογιστικών διεργασιών. [3]

1.2 Σύντομη περιγραφή της πτυχιακής

Η συγκεκριμένη πτυχιακή εργασία πραγματεύεται το ευρύτερο αντικείμενο των ετερογενών συστημάτων αρχικά περιγράφοντας κάποια βασικά θεωρητικά κομμάτια που έχουν σχέση με αυτά τα συστήματα. Έπειτα γίνεται μια περιγραφή της υπάρχουσας κατάστασης που χρησιμοποιήθηκε για την συγκεκριμένη εργασία δηλαδή το πως λειτουργούν οι τεχνολογίες που βοήθησαν στην επίτευξη των στόχων της πτυχιακής. Στο τρίτο κεφάλαιο αναλύεται βήμα-βήμα η διαδικασία του κάθε μέρους της άσκησης σαν ένα εγχειρίδιο που θα μπορούσε να χρησιμοποιήσει κανείς για να την υλοποιήσει αν ενδιαφέρεται. Στο επόμενο κεφάλαιο δίνονται τα αποτελέσματα του κάθε βήματος με διάφορα στιγμιότυπα από τα προγράμματα που πραγματοποιήθηκαν οι έλεγχοι. Τέλος δίνονται κάποια συμπεράσματα από την συγκεκριμένη εργασία με ιδιαίτερη έμφαση στα σημεία που υπήρχε η μεγαλύτερη δυσκολία εύρεσης των λύσεων που έπρεπε να βρεθούν.

2. Περιγραφή της υπάρχουσας κατάστασης

2.1 DevCloud

Το DevCloud είναι ένα cloud service της Intel το οποίο δουλεύει πάνω σε επεξεργαστές Intel XEON και FPGA acceleration κάρτες. Το συγκεκριμένο cloud διαθέτει έναν αριθμό εργαλείων για ανάπτυξη εφαρμογών. Μερικά από αυτά είναι το Acceleration Stack και το Intel Quartus Prime Lite / Pro. Το DevCloud υποστηρίζει Acceleration κάρτες υψηλών αποδόσεων για να επιτρέπει στους χρήστες να πειραματίζονται με επιταχυνόμενα workloads τα οποία τρέχουν πάνω σε FPGA. [4]

Το Intel Developer Cloud παρέχει δωρεάν πρόσβαση σε ένα ευρύ φάσμα αρχιτεκτονικών της Intel το οποίο βοηθάει τους χρήστες να αποκτήσουν άμεση εμπειρία με το λογισμικό της. Επίσης προσφέρει την δυνατότητα εκτέλεσης workloads πάνω στα αντικείμενα του AI, του edge και του high-performance computing (HPC). Τέλος διαθέτει προ εγκατεστημένα frameworks, εργαλεία και βιβλιοθήκες της Intel, οπότε ο χρήστης δεν χρειάζεται να εγκαταστήσει κάτι από μόνος του. [5]

Κάτι που κάνει το DevCloud την καλύτερη επιλογή για να τεστάρει κάποιος τις εφαρμογές του είναι το χαμηλό κόστος και οι χαμηλές απαιτήσεις συστήματος. Πιο συγκεκριμένα αν κάποιος έπρεπε να αγοράσει το hardware και το software που τρέχει σε αυτά τα συστήματα θα έπρεπε να διαθέσει ένα μεγάλο κεφάλαιο για να τρέξει τα workloads του. Μια άλλη εναλλακτική θα ήταν τα Virtual Machines τα οποία δεν είναι επίσης μια καλή επιλογή καθώς μέσω αυτών κάποιος μπορεί δείξει πως τρέχει η εφαρμογή του σε ένα εικονικό περιβάλλον και όχι σε πραγματικό hardware. Το DevCloud προσφέρει την δυνατότητα να τρέξεις αυτά που θες σε ένα σύστημα το οποίο θα ήταν το ίδιο αν υπήρχε διαθέσιμο στο γραφείο σου. [6]

Για να συνδεθεί κάποιος στο DevCloud πρέπει είτε η συσκευή του να τρέχει πάνω στο λειτουργικό του Linux είτε αν τρέχει στα Windows να διαθέτει ένα ειδικό τερματικό όπως το MobaXterm που χρησιμοποιήθηκε για την συγκεκριμένη υλοποίηση. Επίσης άλλα συμβατά εργαλεία είναι το Cygwin και το PuTTY. Το MobaXterm διαθέτει έναν SSH Client ο οποίος χρειάζεται για να συνδεθούμε στο DevCloud και να χρησιμοποιήσουμε κάποιον server. Παρακάτω θα αναφερθούμε πιο αναλυτικά στο πως έγινε η διαδικασία εγκατάστασης και σύνδεσης σε αυτό.

Στην συγκεκριμένη εργασία χρειάστηκε να συνδεθούμε στο DevCloud και να διαμορφώσουμε τις απαραίτητες μεταβλητές και ρυθμίσεις που χρειάζονταν για την ανάπτυξη του AFU (Acceleration Functional Unit) μας. Ένα tutorial που βοήθησε πολύ στην διαμόρφωση του DevCloud για την δική μας περίπτωση ήταν το Using the Intel FPGA DevCloud for AFU Development.

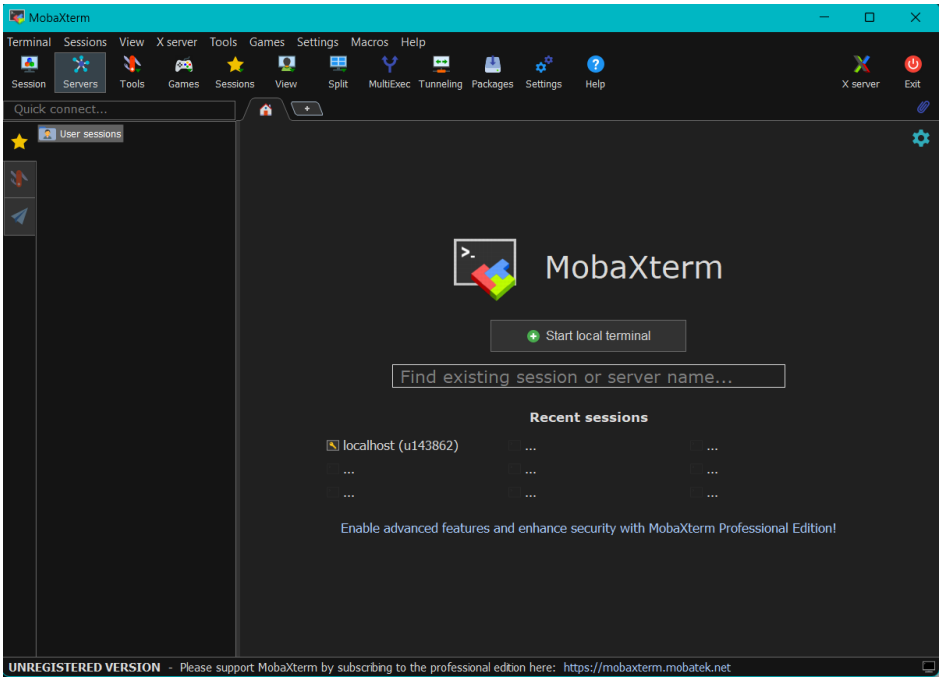
Η διαδικασία που ακολουθήσαμε για να το χρησιμοποιήσουμε είναι η εξής. Αρχικά έπρεπε να επισκεφτούμε την ιστοσελίδα της Intel για να δημιουργήσουμε λογαριασμό και να στείλουμε ένα αίτημα πρόσβασης σε αυτό μέσω της ιστοσελίδας https://intelsoftwaresites.secure.force.com/fpgadevcloud

Όταν τελειώσαμε με την διαδικασία της εγγραφής, μας στάλθηκε ένα e-mail με τα στοιχεία πρόσβασης στο DevCloud όπως το User ID το οποίο είναι το αναγνωριστικό του χρήστη μας και το UUID key το οποίο είναι το κλειδί με το οποίο θα έχουμε πρόσβαση στο DevCloud. Επίσης μας είχε δοθεί η πληροφορία ότι η πρόσβαση που θα έχουμε στο DevCloud θα ήταν για 120 ημέρες. Βέβαια μετά μπορούσαμε να κάνουμε νέα αίτηση ώστε να ανανεωθεί η πρόσβαση αυτή.

Στη συνέχεια χρειάστηκε να κατεβάσουμε και να εγκαταστήσουμε το MobaXterm free edition από την ιστοσελίδα https://mobaxterm.mobatek.net/download-home-edition.html .

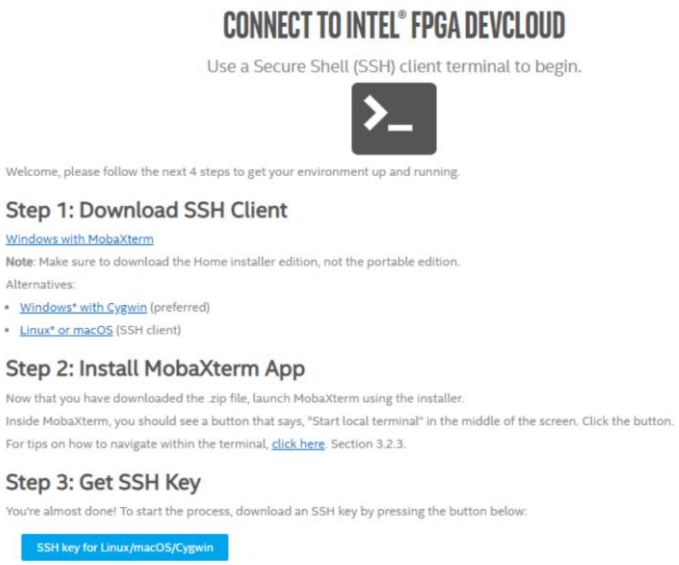
Αφού έγινε η λήψη αποσυμπιέσαμε το zip αρχείο και εκτελέσαμε ένα msi αρχείο υπεύθυνο για την εγκατάσταση του προγράμματος.

Όταν ανοίγουμε το MobaXterm μας εμφανίζεται το παρακάτω παράθυρο. Πατάμε στο “Start local terminal” για να ανοίξουμε ένα νέο τερματικό.



Εικόνα 1 Το παράθυρο του MobaXterm

Για να συνδεθούμε στο DevCloud μέσω του τερματικού θα πρέπει πρώτα να κατεβάσουμε ένα SSH κλειδί από το λινκ https://devcloud.intel.com/fpga/connect/# . Αφού συνδεθούμε στον λογαριασμό που φτιάξαμε πατάμε στο πεδίο “SSH key for Linux/macOS/Cygwin” για να κατεβάσουμε ένα SSH κλειδί που θα χρειαστούμε για να αποκτήσουμε πρόσβαση στην υπηρεσία του DevCloud.



Εικόνα 2 Τα βήματα εγκατάστασης του MobaXterm

Έπειτα πρέπει να δημιουργήσουμε την διαδρομή ~/.ssh εάν δεν υπάρχει ήδη και να μετακινήσουμε το κατεβασμένο κλειδί σε αυτήν την διαδρομή. Οι εντολές που θα εκτελεστούν είναι οι παρακάτω:

```
mkdir -p ~/.ssh

mv

/drives/c/Users/kosta/Downloads/devcloud-access-key-143682.txt ~/.ssh/
```

Στην συνέχεια πρέπει να τροποποιήσουμε το config αρχείο που υπάρχει μέσα στο directory του ssh. Αυτό θα το κάνουμε με την χρήση της εντολής "vim config". Έπειτα θα προσθέσουμε τις ακόλουθες εντολές:

```
Host devcloud

User u143862

IdentityFile ~/.ssh/devcloud-access-key-143862.txt

ProxyCommand ssh -T -i ~/.ssh/devcloud-access-key-143862.txt
guest@ssh.devcloud.intel.com
```

Τέλος θα χρειαστεί να διορθώσουμε τα permissions του SSH με τις εντολές:

```
chmod 600 ~/.ssh/devcloud-access-key-u143862.txt

chmod 600 ~/.ssh/config
```

Πλέον έχουμε ολοκληρώσει την παραμετροποίηση και εγκατάσταση του DevCloud οπότε θα πρέπει να συνδεθούμε σε αυτό με την εντολή *"ssh devcloud"*. Κατά την πρώτη απόπειρα σύνδεσης θα μας εμφανιστεί ένα μήνυμα για το αν θέλουμε να προσθέσουμε το host devcloud στην λίστα των γνωστών hosts οπότε πρέπει να πληκτρολογήσουμε *"yes"*. Έτσι τις επόμενες φορές που θα χρειαστεί να συνδεθούμε, απλά θα εκτελούμε την εντολή *"ssh devcloud"*.

Αφού ολοκληρωθεί η πρόσβαση στο DevCloud πληκτρολογούμε την εντολή *devcloud_login* και έπειτα τις επιλογές 1 και πάλι 1 για να χρησιμοποιήσουμε την συσκευή Arria 10. Έπειτα έχουμε πρόσβαση στον χρήστη μας και μπορούμε να περιηγηθούμε στα directories με την εντολή cd.

```
u143862@login-2:~$ devcloud_login

You are selecting an interactive compute server sesssion. Please consider using batch
mode submission using
devcloud_login -b to not tie up compute servers with idle sessions.
See the help menu using devcloud_login -h for more details.

What are you trying to use the Devcloud for?

1) Arria 10 PAC Compilation and Programming - RTL AFU, OpenCL
2) Arria 10 - OneAPI, OpenVINO
3) Stratix 10 PAC Compilation and Programming - RTL AFU, OpenCL
4) Stratix 10 - OneAPI, OpenVINO
5) Compilation (Command Line) Only
6) Enter Specific Node Number

Number: 1
Which Arria 10 PAC Development Stack release would you like to source?
0) 1.2
1) 1.2.1

Number: 1
```

Εικόνα 3 Σύνδεση και επιλογή συσκευής στο DevCloud

2.2 oneAPI

Το oneAPI είναι ένα ενοποιημένο, πολυαρχιτεκτονικό μοντέλο προγραμματισμού πολλών προμηθευτών το οποίο προσφέρει πολλές δυνατότητες πάνω στο developing με αρχιτεκτονικές επιταχυντών. [7] Μερικά είδη υλικού που περιλαμβάνει είναι κάρτες γραφικών, επιταχυντές AI (Artificial Intelligence) και FPGA (Field Programmable Gate Arrays. Ο σκοπός του είναι να αποκλείσει την ανάγκη από τους προγραμματιστές να διαθέτουν ξεχωριστές βάσεις κώδικα, πολλαπλές γλώσσες προγραμματισμού και διαφορετικά εργαλεία και ροές εργασίας για κάθε αρχιτεκτονική. Έτσι μπορούν να αναπτύξουν και να βελτιστοποιήσουν τις εφαρμογές τους σε διαφορετικές πλατφόρμες χρησιμοποιώντας το ίδιο σετ εργαλείων αποσφαλμάτωσης και ανάλυσης αποδόσεων π.χ. την λήψη δεδομένων χρόνου εκτέλεσης από τον host και επιταχυντές μέσω του προφίλ Vtune. [8]

Η προδιαγραφή του oneAPI επεκτείνει τα ήδη υπάρχοντα προγραμματιστικά μοντέλα με την ενεργοποίηση πολλαπλών αρχιτεκτονικών υλικού μέσα από μια γλώσσα παράλληλων δεδομένων, ενός σετ βιβλιοθηκών API και μιας διεπαφής χαμηλού επιπέδου για να υποστηρίζει cross-architecture προγραμματισμό. Είναι χτισμένο πάνω στα πρότυπα της βιομηχανίας και παρέχει μια ανοιχτή, cross-platform πλατφόρμα για προγραμματιστές. [9]

Το oneAPI υποστηρίζει τις δύο πιο διαδεδομένες εταιρείες πάνω στον χώρο των GPUs, οι οποίες είναι η NVIDIA και η AMD. Συγκεκριμένα, στις NVIDIA κάρτες που υποστηρίζουν CUDA, μπορεί να χρησιμοποιηθεί με τα ήδη υπάρχοντα toolkits του oneAPI που περιλαμβάνουν τους μεταγλωττιστές της Intel oneAPI DPC++/C++. Αυτοί μπορούν να μεταγλωττίσουν SYCL κώδικα ο οποίος είναι συμβατός με επιταχυντές υλικού και να τρέξει σε κάρτες γραφικών της NVIDIA. Επιπλέον στις κάρτες γραφικών της AMD υπάρχει η δυνατότητα χρήσης του oneAPI με τα ίδια toolkits και τους ίδιους compilers με την εξαίρεση ότι βρίσκονται ακόμα σε beta στάδιο. [10]

2.3 SystemVerilog

Στα ψηφιακά κυκλώματα τα οποία αποτελούνται από τρανζίστορ που συνδέονται μεταξύ τους όταν θέλουμε να σχεδιάσουμε και να αναλύσουμε ένα κύκλωμα, είναι πιο εύκολο να χρησιμοποιήσουμε λογικές πύλες. Αυτές όταν συνδυάζονται μας δίνουν τους καταχωρητές, τους πολυπλέκτες, τους αποκωδικοποιητές και πολλά άλλα γνωστά κυκλώματα. Αυτή η ιεραρχική σχεδίαση που ακολουθούμε για να σχεδιάσουμε τέτοια κυκλώματα ονομάζεται σχηματικό διάγραμμα. Βέβαια όταν έχουμε να κάνουμε με μεγάλο αριθμό συνδυαστικών η ακολουθιακών κυκλωμάτων στο ίδιο σχηματικό, δεν είναι το ίδιο πρακτικό να σχεδιάσουμε το σχηματικό με το να περιγράψουμε το κύκλωμα που θέλουμε με μια γλώσσα περιγραφής υλικού. [11]

Η SystemVerilog είναι μια από τις πιο γνωστές γλώσσες περιγραφής υλικού μαζί με την VHDL και την Verilog. Εμπλουτίζει πολλά χαρακτηριστικά από τις γλώσσες C και C++. Χρησιμοποιείται για την περιγραφή της συμπεριφοράς του κυκλώματος που θέλουμε να σχεδιάσουμε και δημιουργεί έτσι ψηφιακά μπλοκ που αποτελούνται από συνδυαστικές πύλες και ακολουθιακά στοιχεία. Για να ελέγξουμε ότι η περιγραφή που έχουμε κάνει για το υλικό μας είναι σωστή, χρειαζόμαστε μια γλώσσα επαλήθευσης υλικού που να διαθέτει πολύπλοκα τεστ επαλήθευσης. [12]

Στην SystemVerilog για να αναπαραστήσουμε τα δεδομένα ενός κυκλώματος συναντάμε πολλούς τύπους δεδομένων που υπάρχουν και στην C, C++. Πολλοί αλγόριθμοι που είναι φτιαγμένοι πάνω στην C μάλιστα μπορούν να μετατραπούν πιο εύκολα στην SystemVerilog αν οι δύο γλώσσες έχουν τους ίδιους τύπους δεδομένων. Ενώ στην απλή Verilog οι τύποι των μεταβλητών έχουν τέσσερις καταστάσεις: το κάθε μπιτ έχει τιμή 0,1,X ή Z, στην SystemVerilog συναντάμε δύο νέους τύπους δεδομένων που έχουν δύο καταστάσεις και το κάθε μπιτ είναι μόνο 0 ή 1. Όταν δεν χρειαζόμαστε τις τιμές X και Z όπως στα test benches και στα for-loops μπορούμε να χρησιμοποιήσουμε αυτούς τους τύπους. [13]

2.4 FPGA

Τα FPGAs ή αλλιώς Field Programmable Gate Arrays είναι συσκευές ημιαγωγών που βασίζονται σε ένα matrix ρυθμιζόμενων λογικών μπλοκ (CLBs) τα οποία συνδέονται μέσω προγραμματιζόμενων διασυνδέσεων. Τα FPGAs μπορούν να αναπρογραμματιστούν ανάλογα με τις προδιαγραφές της επιθυμητής εφαρμογής και λειτουργίας τους αφού κατασκευαστούν. Αυτή η λειτουργία ξεχωρίζει τα FPGAs από τα Application Specific Integrated Circuits (ASICs), τα οποία είναι κατασκευασμένα για να εκτελούν συγκεκριμένες τεχνικές διεργασίες. Παρόλο που υπάρχουν FPGAs τα οποία μπορούν να προγραμματιστούν μια φορά (one-time programmable ή αλλιώς OTP), τα επικρατέστερα σε θέμα λειτουργιών είναι τα SRAM που μπορούν να αναπρογραμματιστούν καθώς εξελίσσεται η σχεδίαση. [14]

Η δυνατότητα αναπρογραμματισμού που διαθέτουν τα FPGAs τα ξεχωρίζουν από τα υπόλοιπα ολοκληρωμένα κυκλώματα καθώς τους προσφέρουν ευελιξία και την δυνατότητα να προσαρμοστούν ανάλογα στις ανάγκες του μεγαλύτερου συστήματος με το οποίο συνδέονται. Είναι ιδανικά κατασκευασμένα για τις σημερινές ταχύτατα αναπτυσσόμενα εφαρμογές. Παραδείγματα αυτών είναι το edge computing, η τεχνητή νοημοσύνη (AI), η ασφάλεια συστημάτων, το 5G, ο εργοστασιακός αυτοματισμός και η ρομποτική. [15]

Υπάρχουν διάφορα είδη χρηστών παγκοσμίως που χρησιμοποιούν τα FPGAs για να καλύψουν τις όποιες ανάγκες τους. Αρχικά οι Developers χρησιμοποιούν FPGAs στον τομέα της ηλεκτρονικής για πολλές διαφορετικές εφαρμογές. Έπειτα οι ερευνητές τα χρησιμοποιούν για να εξερευνήσουν τον κλάδο της λογικής σχεδίασης και για να επιλύσουν δύσκολα προβλήματα και να κατασκευάσουν καινούριους αλγόριθμους. Έπειτα υπάρχουν οι φοιτητές οι οποίοι μπορούν να εξασκηθούν στον προγραμματισμό που επιτρέπουν αυτά τα κυκλώματα και τέλος υπάρχουν άτομα που έχουν σαν χόμπι την λογική σχεδίαση και τα FPGAs είναι πολύ ιδανικά για αυτούς. [16]

2.5 Γλώσσα προγραμματισμού C

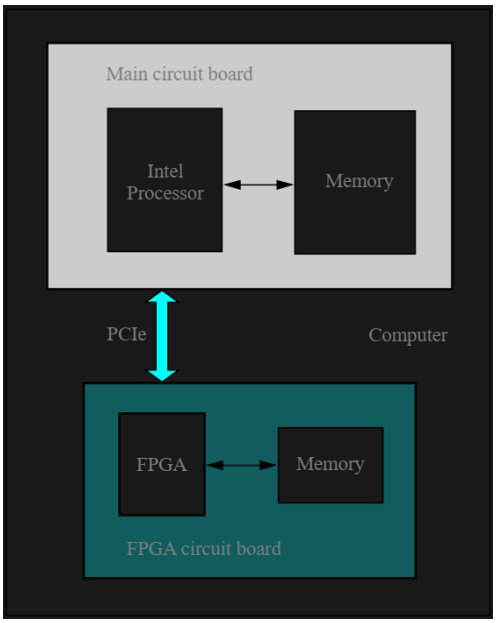
Η C είναι μία γλώσσα προγραμματισμού που χρησιμοποιείται για εφαρμογές γενικότερου σκοπού, η οποία εφευρέθηκε από τον Dennis Ritchie στα εργαστήρια Bell το 1972. Είναι μια πολύ διάσημη γλώσσα παρόλο που είναι αρκετά παλιά και είναι ισχυρά σχετιζόμενη με το λειτουργικό σύστημα UNIX καθώς ο λόγος που εφευρέθηκε ήταν για να γραφτεί το συγκεκριμένο λειτουργικό. [17]

Σχεδιαστικά, τα χαρακτηριστικά της C την καθιστούν ιδανική για πολλές λειτουργίες των επεξεργαστών και γι' αυτό την συναντάμε σε λειτουργικά συστήματα, οδηγούς συσκευών, σε επίπεδα πρωτοκόλλων και σε λογισμικό εφαρμογών. Η C χρησιμοποιείται συχνά σε αρχιτεκτονικές υπολογιστών που ποικίλλουν με τις μικρότερες σε μέγεθος να είναι οι μικροελεγκτές και τα ενσωματωμένα συστήματα και οι μεγαλύτερες να είναι οι υπερυπολογιστές. [18]

Λέγεται ότι η 'C' είναι η γλώσσα του Θεού και η βάση για προγραμματισμό καθώς αν κάποιος γνωρίζει να την χρησιμοποιεί μπορεί εύκολα να αποκτήσει γνώσεις άλλων γλωσσών προγραμματισμού που χρησιμοποιούν παρόμοια λογική με την C. Επιπλέον είναι αναγκαίο κάποιος να έχει ένα υπόβαθρο στους μηχανισμούς της μνήμης των υπολογιστών καθώς είναι ένα σημαντικό χαρακτηριστικό της χρήσης της C. [19]

3. Εγχειρίδιο σχεδίασης ενός Intel AFU

Στο ακόλουθο κεφάλαιο θα αναλύσουμε βήμα-βήμα το πως σχεδιάσαμε και πως υλοποιήσαμε ένα ετερογενές σύστημα με χρήση των τεχνολογιών της Intel. Στην δική μας υλοποίηση ο στόχος ήταν να σχεδιάσουμε ένα Acceleration Functional Unit ή αλλιώς AFU το οποίο θα δουλεύει πάνω σε ένα FPGA. Αυτό το FPGA που επιτρέπει την υλοποίηση γενικότερα άλλων κυκλωμάτων συνδέεται με έναν επεξεργαστή Intel. Η σύνδεση τους γίνεται μέσω μιας θύρας PCI express (PCIe). Αυτό σημαίνει ότι οι δύο αυτές συσκευές επικοινωνούν μεταξύ τους για να υπολογίσουν τα αποτελέσματα που θέλουμε. Στην παρακάτω εικόνα μπορούμε να δούμε σε μορφή σχεδιαγράμματος ποια είναι η δομή του ετερογενούς συστήματος που θα χρησιμοποιήσουμε:



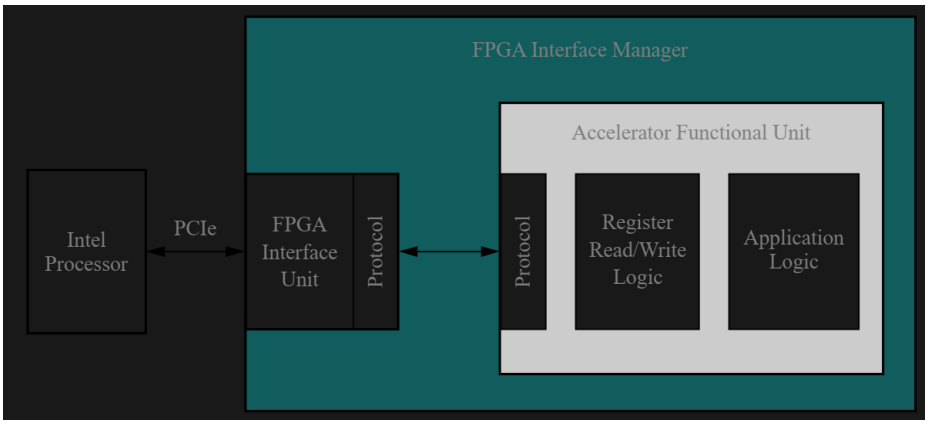
Εικόνα 4 Το block diagram του συστήματος. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Μιας και θα χρησιμοποιήσουμε τεχνολογίες της Intel για την ανάπτυξη του ετερογενούς συστήματος, αυτή μας παρέχει το Intel Acceleration Stack το οποίο είναι ένα σετ εργαλείων λογισμικού και υλικού όπως και διάφορες λειτουργίες για την σύνδεση τους. Το DevCloud

ανήκει σε αυτό το σετ και θα το αξιοποιήσουμε για να τρέξουμε ορισμένα προγράμματα σε C πάνω στο AFU το οποίο υλοποιήσαμε με την SystemVerilog. [20]

3.1 Πρώτο μέρος

Στο πρώτο μέρος θα σχεδιάσουμε ένα κύκλωμα το οποίο θα αποτελεί το κυρίως τμήμα του AFU. Σε αυτή την φάση θα έχουμε ένα απλό κύκλωμα το οποίο θα περιγράφεται σε γλώσσα SystemVerilog. Για να καταλάβουμε καλύτερα από τι μηχανισμούς θα αποτελείται το AFU μας και πως αυτό θα συνδέεται με το FPGA, έχουμε την παρακάτω εικόνα η οποία είναι ένα μπλοκ διάγραμμα υψηλού επιπέδου. Το FPGA Interface Manager περιέχει το σύστημα που θα λειτουργεί πάνω στο FPGA. Το Interface Unit (FIU) είναι ουσιαστικά μια γέφυρα μεταξύ της διεπαφής PCIe και του AFU.



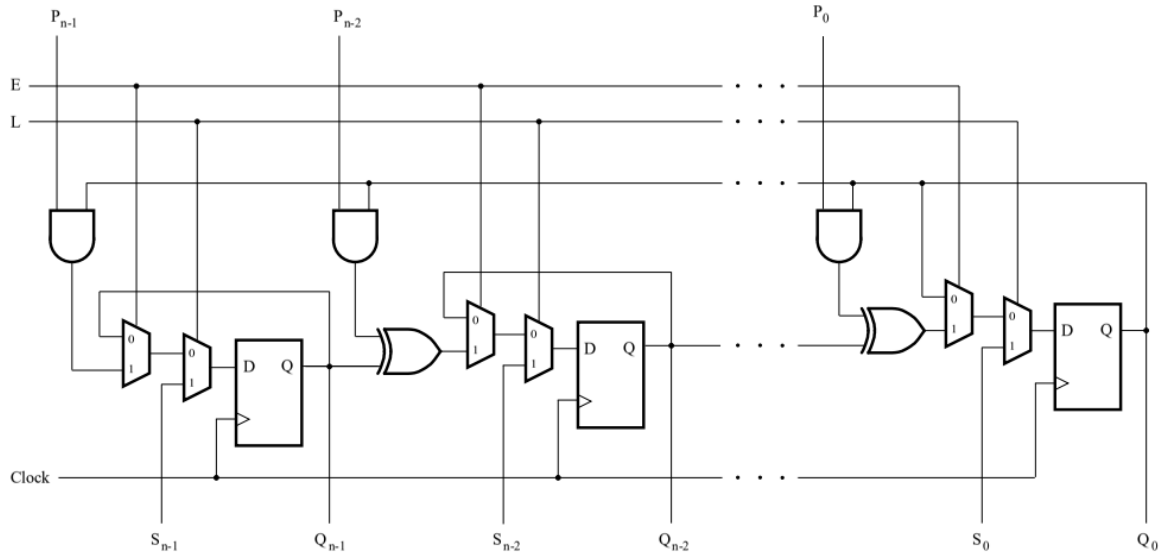
Εικόνα 5 Το block diagram του FPGA. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Το AFU περιλαμβάνει κάποια λογικά κυκλώματα που μέσω του πρωτοκόλλου CCI-P (Core Cache Interface Protocol) συνδέονται σε κάποιους καταχωρητές. Περισσότερες πληροφορίες για το CCI-P μπορούν να βρεθούν σε αυτήν την ιστοσελίδα. Αυτά τα κυκλώματα παρέχουν αποκωδικοποίηση διευθύνσεων που επιτρέπουν στον επεξεργαστή να

διαβάζει η να γράφει στους καταχωρητές του AFU χρησιμοποιώντας memory mapped I/O. Μερικοί από αυτούς τους καταχωρητές έχουν αποκλειστικούς σκοπούς που χρειάζονται για να είναι συμβατοί με το Intel Acceleration Stack. Άλλοι καταχωρητές είναι μέρος του Application Logic στο AFU, το οποίο χρησιμοποιείται για να εκτελεί υπολογισμούς μαζί με τον επεξεργαστή.

Ο σκοπός του AFU που κατασκευάσαμε είναι να παρέχει ένα σύστημα υλικού το οποίο θα παράγει τυχαίους αριθμούς. Ο επεξεργαστής μπορεί να διαμορφώνει το AFU έτσι ώστε να παράγει ακεραίους σε ένα συγκεκριμένο εύρος αριθμών ή να επηρεάζει την σειρά των τιμών που δημιουργούνται. Όποτε χρειάζεται να εκτελεστεί κάποιος υπολογισμός, ο επεξεργαστής μπορεί να διαβάσει μια τυχαία ακέραια τιμή από το AFU.

Στο πρώτο μέρος της μεθοδολογίας θα περιγράψουμε το πως σχεδιάσαμε το Application Logic Module του AFU. Το υπόλοιπο κομμάτι των κυκλωμάτων που αποτελούν το AFU θα το αναλύσουμε στο δεύτερο μέρος. Το Application Logic όπως προαναφέραμε παράγει τυχαίες τιμές. Το κύκλωμα που είναι υπεύθυνο για αυτήν την διεργασία ονομάζεται Linear Feedback Shift Register (LFSR).



Εικόνα 6 Το σχηματικό διάγραμμα του Linear Feedback Shift Register (LFSR). Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Στην παραπάνω εικόνα παρατηρούμε ένα σχηματικό του LFSR στο οποίο δημιουργείται ένας καταχωρητής n ψηφίων με όνομα Q. Η είσοδος φόρτωσης με το γράμμα L παίρνει την τιμή του seed. Όταν όμως η είσοδος L είναι 0 τότε η έξοδος Q εξαρτάται από την είσοδο ενεργοποίησης E. Όταν αυτή είναι 0 τότε η τιμή του Q δεν μπορεί να αλλάξει. Όταν όμως η είσοδος E είναι 1 τότε το Q έχει την συμπεριφορά ενός καταχωρητή ολίσθησης. Η τιμή η οποία ολισθαίνει στο κάθε φλιπ-φλοπ εξαρτάται από την τιμή του Polynomial (P) που χαρακτηρίζει το LFSR.

Για κάθε θέση bit i έχουμε στην παραπάνω εικόνα την είσοδο δεδομένων του κάθε φλιπ-φλοπ $D_{(i)}$ και την έξοδο του $Q_{(i)}$ αντίστοιχα. Όταν το LFSR λειτουργεί σαν καταχωρητής

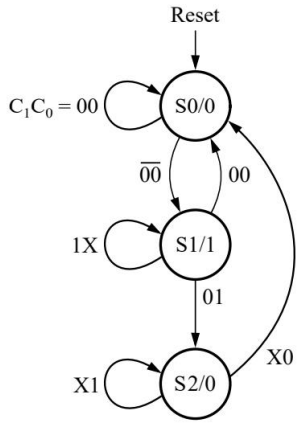
ολίσθησης, αν το P_i είναι 0 τότε το $D(_i)$ είναι $Q_{(i+1)}$. Όταν όμως το $P_{(i)}$ είναι 1 τότε το $D_{(i)}$ είναι $Q_{(i+1)}$ XOR $Q_{(0)}$. Αυτή η τοποθέτηση των πυλών XOR στο LFSR του επιτρέπει να παράγει μια ακολουθία τιμών n-bit που μπορούν να χρησιμοποιηθούν σαν «τυχαίοι» ακέραιοι. Να σημειώσουμε εδώ ότι στην θέση n-1 έχουμε $D_{(i)} = 0$ εάν $P_{(n-1)} = 0$ και $D_{(i)} = Q_{(0)}$ εάν $P_{(n-1)} = 1$.

Πέρα από τον καταχωρητή εξόδου Q της παραπάνω εικόνας, το Application Logic Module διαθέτει και έναν καταχωρητή n-bit για να αποθηκεύει την τιμή του πολυωνύμου P και έναν καταχωρητή ελέγχου 2-bit, C. Όταν γίνεται επαναφορά του κυκλώματος ισχύει $C = C_{(1)}C_{(0)} = 00$ και το LFSR είναι σε κατάσταση διακοπής. Το λογισμικό που τρέχει στον επεξεργαστή μπορεί να γράφει νέες τιμές στον καταχωρητή ελέγχου για να χρησιμοποιεί το LFSR σε δύο διαφορετικές καταστάσεις.

Όταν το bit $C_{(1)}$ είναι 1 του καταχωρητή ελέγχου, ενεργοποιείται η κατάσταση συνεχούς λειτουργίας. Τότε το LFSR μπορεί να παράγει έναν νέο «τυχαίο» ακέραιο αριθμό σε κάθε θετική ακμή του ρολογιού. Το LFSR μπορεί να επιστρέψει στην κατάσταση διακοπής αν ο καταχωρητής ελέγχου πάρει την τιμή C = 00. Επίσης όταν έχουμε $C_{(1)} = 0$ και $C_{(0)} = 1$ τότε το LFSR τοποθετείται σε κατάσταση step. Αυτό σημαίνει ότι μπορεί να ενεργοποιηθεί μόνο για έναν κύκλο ρολογιού ώστε να παράγει μόνο μία τιμή ψευδοτυχαίου αριθμού. Το LFSR και πάλι μπορεί να σταματήσει αν το C γίνει 00.

Μια εφαρμογή μπορεί να επιλέξει να βάλει το LFSR είτε σε συνεχή η βηματική κατάσταση ανάλογα τις απαιτήσεις του υπολογισμού που εκτελείται. Για να υποστηρίξει αυτές τις

καταστάσεις λειτουργίας το Application Logic Module πρέπει να έχει την δυνατότητα να ελέγχει το σήμα ενεργοποίησης στο LFSR. Αυτό μπορεί να γίνει με την χρήση μιας μηχανής πεπερασμένων καταστάσεων (FSM) που βλέπουμε στην παρακάτω εικόνα. Τα μπιτ ελέγχου $C_{(1)}C_{(0)}$ είναι οι είσοδοι της μηχανής από τις οποίες παράγεται η έξοδος z. Αυτή η έξοδος συνδέεται άμεσα με την είσοδο ενεργοποίησης E του LFSR.



Εικόνα 7 Το διάγραμμα καταστάσεων της FSM. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Η FSM (Finite State Machine) έχει σαν αρχική κατάσταση την S0 και παράγει την έξοδο z = 0, η οποία αναφέρεται στο διάγραμμα καταστάσεων ως S0/0. Όσο η είσοδος $C_{(1)}$ και $C_{(0)}$ είναι 0 η FSM παραμένει στην κατάσταση S0. Στην περίπτωση που αλλάξει το $C_{(1)}$ σε 1, στην επόμενη ενεργή ακμή ρολογιού η μηχανή περνάει στην κατάσταση S1 όπου η έξοδος z γίνεται 1 και ενεργοποιεί το LFSR. Η μηχανή παραμένει σε αυτήν την κατάσταση όσο ισχύει $C_{(1)}$ = 1 όπως δείχνει και το βέλος με την τιμή 1X (η τιμή αυτή καλύπτει τις περιπτώσεις $C_{(1)}C_{(0)}$ = 10 και $C_{(1)}C_{(0)}$ = 11). Αυτό το σενάριο βάζει το LFSR σε συνεχή λειτουργία, που παράγει έναν τυχαίο αριθμό ανά κύκλο ρολογιού. Όταν το C ξαναγίνει 00 η μηχανή καταστάσεων επιστρέφει στην αρχική κατάσταση S0.

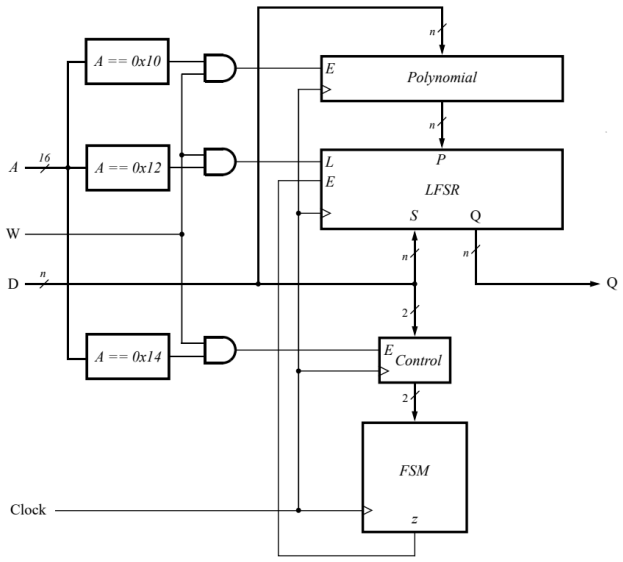
Μια άλλη περίπτωση είναι να έχουμε το $C_{(1)}$ = 0 και το $C_{(0)}$ να γίνει 1. Τότε στην επόμενη ακμή του ρολογιού η FSM περνάει στην κατάσταση S1 όπου ενεργοποιείται η έξοδος z αλλά μόνο για έναν κύκλο ρολογιού. Όπως βλέπουμε από το βέλος με την ετικέτα $C_{(1)}C_{(0)}$ = 01, η επόμενη ακμή ρολογιού φέρνει την μηχανή στην κατάσταση S2 όπου z = 0. Όσο το $C_{(0)}$

παραμένει 1, η μηχανή παραμένει στην S2 και επιστρέφει στην αρχική κατάσταση S0 μόνο αν αλλάξει το $C_{(0)}$ σε 0. Αυτή η περίπτωση κάνει το LFSR να παράγει μόνο έναν τυχαίο ακέραιο που συνεπάγεται την βηματική λειτουργία του.

Το διάγραμμα του κυκλώματος Application Logic που υπάρχει παρακάτω περιλαμβάνει τις εξής εισόδους:

Μία είσοδο 16-bit A, μία είσοδο δεδομένων n-bit D και μία είσοδο εγγραφής W που αναπαριστά τον τρόπο που το κύκλωμα θα συμπεριληφθεί σε ένα AFU το οποίο με τη σειρά του θα συνδεθεί σε έναν επεξεργαστή (δεύτερο βήμα). Η αποκωδικοποίηση διευθύνσεων, που γίνεται με την χρήση NOR πυλών εκχωρεί την διεύθυνση $A = 10_{16}$ στον καταχωρητή πολυωνύμου, την διεύθυνση $A = 12_{16}$ στο LFSR και την διεύθυνση $A = 14_{16}$ στον καταχωρητή ελέγχου. Οι πύλες AND του κυκλώματος διασφαλίζουν ότι κάθε καταχωρητής μπορεί να φορτώσει την τιμή δεδομένων D όταν χρησιμοποιείται η εκάστοτε διεύθυνση και η είσοδος εγγραφής W είναι 1. Όταν γίνεται εγγραφή δεδομένων στο LFSR, η είσοδος seed (S) καθορίζει τις εξόδους Q των flip-flop και η είσοδος ενεργοποίησης E ελέγχεται όπως αναφέραμε και πιο πάνω από την έξοδο z της μηχανής καταστάσεων.

Για την σχεδίαση του Application module χρησιμοποιήσαμε την γλώσσα της SystemVerilog και μιας και ο σκοπός του πρώτου μέρους ήταν να σχεδιάσουμε ένα μέρος του συστήματος σε επίπεδο υλικού, δεν χρησιμοποιήσαμε το DevCloud το οποίο θα χρειαστεί μόνο στα επόμενα μέρη. Για την μεταγλώττιση του κώδικα και για την προσομοίωση του application module χρησιμοποιήσαμε το ModelSim το οποίο υποστηρίζει πολλές γλώσσες περιγραφής υλικού.



Εικόνα 8 Το σχηματικό διάγραμμα του AFU. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

3.2 Δεύτερο μέρος

Στο δεύτερο μέρος ενσωματώσαμε το application module από το πρώτο μέρος σε ένα νέο αρχείο για να μπορέσουμε να δημιουργήσουμε το AFU το οποίο με την σειρά του θα υλοποιηθεί πάνω σε ένα FPGA. Σε αυτό το μέρος κάναμε επίσης χρήση της SystemVerilog όπως και μερικά εργαλεία ανάπτυξης υλικού που υπάρχουν στο DevCloud. Ένα AFU όπως είχαμε δει και σε μία από τις παραπάνω εικόνες χρησιμοποιεί μία θύρα πρωτοκόλλου και μερική λογική για να μπορεί να γράφει και να διαβάζει τις τιμές των καταχωρητών που βασίζονται σε αυτό το πρωτόκολλο. Στην δικιά μας περίπτωση χρησιμοποιήσαμε το Core Cache Interface Protocol (CCI-P) της Intel. Αυτό το πρωτόκολλο είναι μέρος της διεπαφής του FPGA στην δεύτερη εικόνα και αποτελεί μια μέθοδο σύνδεσης του επεξεργαστή με το AFU.

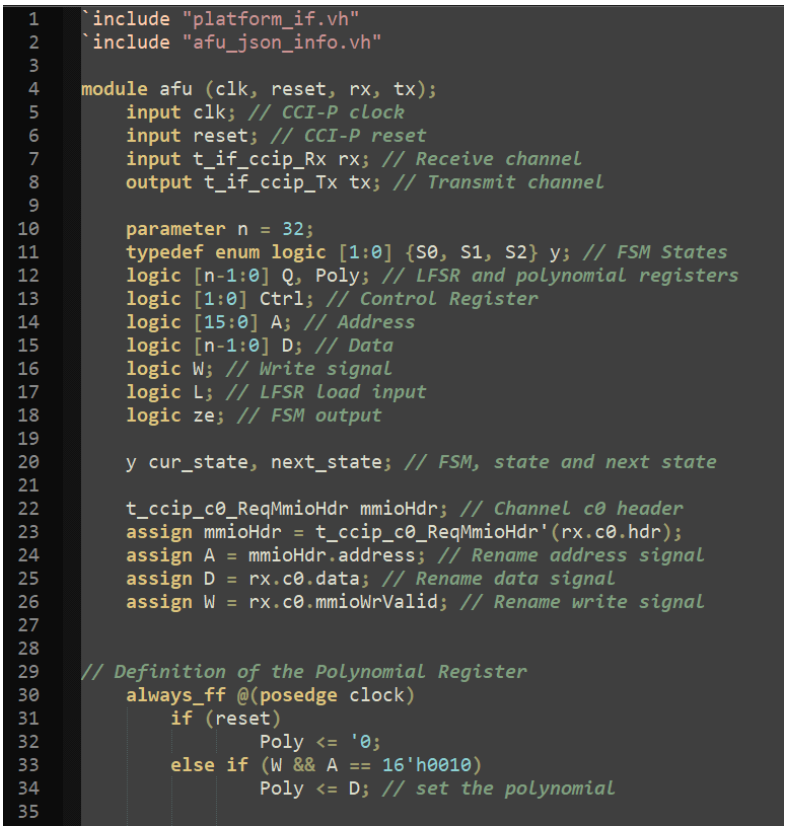
Στην εικόνα 9 μπορούμε να δούμε τον κώδικα για την υλοποίηση του AFU και θα αναλύσουμε τι μηχανισμούς και σήματα ενεργοποιούμε σε κάθε γραμμή του κώδικα.

Αρχικά στις γραμμές 1 και 2 χρησιμοποιούμε τις επικεφαλίδες `platform_if.vh` και `afu_json_info.vh`. Η πρώτη ορίζει κάποιους τύπους δεδομένων του CCI-P πρωτοκόλλου που χρησιμοποιούνται στον κώδικα, και η δεύτερη καθορίζει κάποιες χρήσιμες πληροφορίες για το AFU μας.

Ένα AFU μπορεί να έχει διαφορετικές θύρες ανάλογα την λειτουργία που είναι σχεδιασμένο να εκτελεί. Στην δική μας περίπτωση, πέρα από τις εισόδους clock και reset χρειαζόμαστε και άλλη μια είσοδο με όνομα rx για να λαμβάνουμε δεδομένα CCI-P και μια έξοδο με όνομα tx για την μετάδοση δεδομένων. Αυτές οι δύο θύρες έχουν ειδικούς τύπους δεδομένων όπως παρατηρούμε στις γραμμές 7 και 8 οι οποίες ορίζονται από το header file `platform_if.vh`.

Στις γραμμές 10 μέχρι 18 ορίζουμε μια παράμετρο n = 32 και τα σήματα που χρειαζόμαστε. Το σήμα Q αναπαριστά το LFSR το οποίο στο application module που φτιάξαμε στο πρώτο μέρος ήταν η έξοδος του ενώ για το AFU ο επεξεργαστής διαβάζει τα δεδομένα των καταχωρητών από την πόρτα εξόδου tx.

Στις γραμμές 22 και 23 ορίζουμε ένα σήμα ειδικού τύπου του CCI-P που ονομάζεται mmioHdr. Αυτό το σήμα παρέχει την διεύθυνση του καταχωρητή που χρησιμοποιείται εκείνη την στιγμή από τον επεξεργαστή και εκχωρείται στο 16-bit σήμα A στην γραμμή 24. Η διεύθυνση υλοποιείται στο CCI-P ως μια μετατόπιση από την βασική διεύθυνση που χρησιμοποιείται για είσοδο-έξοδο με αντιστοίχιση μνήμης. Κάθε διεύθυνση A είναι ευθυγραμμισμένη με μια double-word (32 bit) στον χώρο διευθύνσεων του επεξεργαστή. Στις γραμμές 25 και 26 εκχωρούμε τα σήματα n bit δεδομένων και εγγραφής του επεξεργαστή στις μεταβλητές D και W αντίστοιχα. Αυτά τα σήματα ενεργοποιούνται μόνο όταν ο επεξεργαστής γράφει κάποια τιμή σε έναν καταχωρητή του AFU. Χρησιμοποιώντας τον καταχωρητή πολυωνύμων σαν παράδειγμα, το μπλοκ always στην γραμμή 30 που δείχνει πως οι καταχωρητές μπορούν να οριστούν σε ένα AFU.



```
1   `include "platform_if.vh"
2   `include "afu_json_info.vh"
3
4   module afu (clk, reset, rx, tx);
5       input clk; // CCI-P clock
6       input reset; // CCI-P reset
7       input t_if_ccip_Rx rx; // Receive channel
8       output t_if_ccip_Tx tx; // Transmit channel
9
10      parameter n = 32;
11      typedef enum logic [1:0] {S0, S1, S2} y; // FSM States
12      logic [n-1:0] Q, Poly; // LFSR and polynomial registers
13      logic [1:0] Ctrl; // Control Register
14      logic [15:0] A; // Address
15      logic [n-1:0] D; // Data
16      logic W; // Write signal
17      logic L; // LFSR load input
18      logic ze; // FSM output
19
20      y cur_state, next_state; // FSM, state and next state
21
22      t_ccip_c0_ReqMmioHdr mmioHdr; // Channel c0 header
23      assign mmioHdr = t_ccip_c0_ReqMmioHdr'(rx.c0.hdr);
24      assign A = mmioHdr.address; // Rename address signal
25      assign D = rx.c0.data; // Rename data signal
26      assign W = rx.c0.mmioWrValid; // Rename write signal
27
28
29  // Definition of the Polynomial Register
30      always_ff @(posedge clock)
31          if (reset)
32                  Poly <= '0;
33          else if (W && A == 16'h0010)
34                  Poly <= D; // set the polynomial
35
```

Εικόνα 9 Ο πηγαίος κώδικας του δεύτερου μέρους

Στην εικόνα 10 που υπάρχει παρακάτω εμφανίζεται ένα μέρος απαραίτητου κώδικα για το AFU ώστε να μπορεί να υποστηρίξει ανάγνωση δεδομένων από τον επεξεργαστή σε συγκεκριμένες διευθύνσεις. Αυτές οι διευθύνσεις αποκωδικοποιούνται στο AFU με την χρήση μιας δήλωσης case στην γραμμή 127. Όπως φαίνεται κάθε διεύθυνση ανταποκρίνεται με την εισαγωγή δεδομένων στην έξοδο tx του AFU. Για την διεύθυνση A = 0 το AFU ανταποκρίνεται με μια συγκεκριμένη 64-bit data pattern που απαιτείται για κάθε AFU.

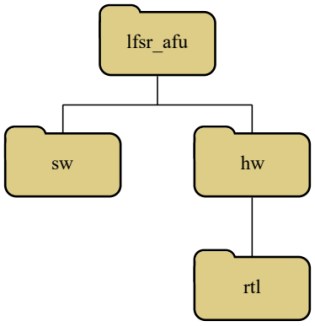
Οι διευθύνσεις A = 2_{16} και A = 4_{16} ανταποκρίνονται στα χαμηλά και υψηλά 64 bit του σήματος afu_id. Αυτό το σήμα αναπαριστά ένα τύπου global μοναδικό αναγνωριστικό για το AFU. Δηλώνεται στην γραμμή 108 και αρχικοποιείται με την συμβολική σταθερά AFU_ACCEL_UUID η οποία ορίζεται στο αρχείο `afu_json_info.vh`. Τέλος, οι διευθύνσεις A = 6 και A = 8 ανταποκρίνονται με 0 για το AFU μας. Περισσότερες λεπτομέρειες σχετικά με τις αναγκαίες διευθύνσεις που πρέπει να υποστηρίζονται σε ένα AFU μπορούν να βρεθούν αναζητώντας online documentation σχετικό με το πρωτόκολλο CCI-P.

Οι γραμμές 148 μέχρι 150 επιτρέπουν στον επεξεργαστή να διαβάσει τα περιεχόμενα των καταχωρητών πολυωνύμου, ελέγχου και LFSR στις διευθύνσεις $A = (10)_{16}$, $A = (12)_{16}$ και $A = (14)_{16}$ αντίστοιχα. Όπως αναφέρθηκε και παραπάνω κάθε διεύθυνση A αναπαριστά μια λέξη double των 32 bit η οποία καταλήγει σε μια διεύθυνση του επεξεργαστή. Αυτό συνεπάγεται ότι οι αποκωδικοποιημένες διευθύνσεις στην case δήλωση ευθυγραμμίζονται σε ένα όριο τετραπλής λέξης (64 bit) επειδή το λιγότερο σημαντικό μπιτ της διεύθυνσης A(0) είναι 0 για κάθε διεύθυνση. Οι προδιαγραφές του πρωτοκόλλου CCI-P απαιτούν ευθυγράμμιση για όλους τους καταχωρητές σε ένα AFU.

```
108     logic [127:0] afu_id = 'AFU_ACCEL_UUID; // from afu_json_info.vh
109
110     always_ff @(posedge clock) begin // respond to memory-mapped I/O reads
111         if (reset) begin
112             tx.c1.hdr <= '0;
113             tx.c1.valid <= '0;
114             tx.c0.hdr <= '0;
115             tx.c0.valid <= '0;
116             tx.c2.hdr <= '0;
117             tx.c2.mmioRdValid <= '0;
118         end
119         else begin
120             // clear read response flag in case there was a response last cycle
121             tx.c2.mmioRdValid <= 0;
122
123             // serve MMIO read requests
124             if (rx.c0.mmioRdValid == 1'b1) begin
125                 // copy TID, which host needs to map response to request
126                 tx.c2.hdr.tid <= mmioHdr.tid;
127                 // post response
128                 tx.c2.mmioRdValid <= 1;
129
130                 case (A)
131                     // AFU header
132                     16'h0000: tx.c2.data <= {
133                         4'b0001, // Feature type = AFU
134                         8'b0, // reserved
135                         4'b0, // afu minor revision = 0
136                         7'b0, // reserved
137                         1'b1, // end of DFH list = 1
138                         24'b0, // next DFH offset = 0
139                         4'b0, // afu major revision = 0
140                         12'b0 // feature ID = 0
141                     };
142                     16'h0002: tx.c2.data <= afu_id[63:0]; // AFU_ID_L
143                     16'h0004: tx.c2.data <= afu_id[127:64]; // AFU_ID_H
144                     16'h0006: tx.c2.data <= 64'h0; // DFH_RSVD0
145                     16'h0008: tx.c2.data <= 64'h0; // DFH_RSVD1
146
147                     // application logic registers
148                     16'h0010: tx.c2.data <= 64'(Poly);
149                     16'h0012: tx.c2.data <= 64'(Q);
150                     16'h0014: tx.c2.data <= 64'(Ctrl);
151
152                     default: tx.c2.data <= 64'h0;
153                 endcase
154             end
155         end
156     end
157 endmodule
```

Εικόνα 10 Συνέχεια του κώδικα του δεύτερου μέρους

Για να ολοκληρώσουμε την σχεδίαση του AFU μας δημιουργήσαμε έναν φάκελο με όνομα *lfsr_afu* στον οποίο υπάρχουν δύο υποφάκελοι με ονόματα hw και sw αντίστοιχα. Ο sw φάκελος αφορά το κομμάτι λογισμικού δηλαδή ό,τι προγράμματα χρησιμοποιήσαμε στα επόμενα μέρη της πτυχιακής τα οποία ως επί το πλείστον είναι στην γλώσσα C. Στον hw φάκελο που μας χρειάστηκε για τα δύο πρώτα μέρη έχουμε έναν ακόμη υποφάκελο με όνομα rtl στον οποίο τοποθετήσαμε το αρχείο afu.sv το οποίο δημιουργεί το AFU μας στην γλώσσα της SystemVerilog.



Εικόνα 11 Η δομή των φακέλων του LFSR. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Η δομή αυτή των φακέλων έχει επιλεχθεί ώστε να μπορεί το DevCloud να χρησιμοποιήσει τα αρχεία μας σωστά. Ένα αρχείο που χρειάζεται για το DevCloud είναι το *filelist.txt*. Σε αυτό έχουμε τα ονόματα των αναγκαίων αρχείων για το AFU και ένα από αυτά είναι το *lfsr_afu.json* το οποίο περιέχει κάποιες χρήσιμες πληροφορίες για το AFU σε μορφή JSON *(JavaScript Object Notation)*. Συγκεκριμένα το αρχείο καθορίζει τον τύπο της θύρας CCI-P που χρειάζεται για το AFU, η οποία ονομάζεται *ccip_std_afu*. Αυτή η διεπαφή αποτελείται από τις θύρες *clock, reset, receive (rx) & transmit (rx)*. Το αρχείο επίσης καθορίζει το όνομα του AFU *(lfsr_afu)* και το μοναδικό αναγνωριστικό του *(UUID)*. Το UUID που βλέπουμε παρακάτω είναι ένα παράδειγμα καθώς θα χρειαστεί να παράξουμε ένα καινούριο για το AFU μας με την χρήση της εντολής *uuidgen* στο DevCloud. Τα αρχεία πηγαίου κώδικα με κατάληξη .sv που παρατηρούμε στην εικόνα 12 καθορίζουν την διεπαφή του AFU και του CCI-P. Επίσης τα αρχεία *ccip_interface_reg.sv* και *ccip_std_afu.sv* πρέπει να βρίσκονται μέσα στον φάκελο *rtl* του AFU.

```
lfsr_afu.json

afu.sv
ccip_interface_reg.sv
ccip_st_afu.sv
```

Figure 12: The contents of *filelist.txt*.

```
{
    "version": 1,
    "afu-image": {
        "power": 0,
        "afu-top-interface": {
            "class": "ccip_std_afu"
        },
        "accelerator-clusters": [
            {
                "name": "lfsr_afu",
                "total-contexts": 1,
                "accelerator-type-uuid": "850adcc2-6ceb-4b22-9722-d43375b61c66"
            }
        ]
    }
}
```

Εικόνα 12 Τα περιεχόμενα του filelist και του json αρχείου. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

Σε αυτό το σημείο θα ξεκινήσουμε να χρησιμοποιούμε το DevCloud για το οποίο μιλήσαμε αναλυτικά στο 2ο κεφάλαιο. Αρχικά θα ανοίξουμε το πρόγραμμα MobaXterm για να μπορούμε να εκτελέσουμε τις εντολές που θα χρειαστούν. Η πρώτη θα είναι για να αντιγράψουμε τους φακέλους και τα αρχεία που φτιάξαμε στο DevCloud. Έχοντας αποθηκεύσει τα συγκεκριμένα στον φάκελο Desktop στον υπολογιστή μας εκτελούμε την εντολή `cd ../../drivers/c/users/kosta/Desktop` και μετά την εντολή `scp -r lfsr_afu devcloud:~/` . Στην συνέχεια συνδεόμαστε στο DevCloud μέσω ssh και εκτελούμε την εντολή `uuidgen` για να δημιουργηθεί ένα νέο UUID για το LFSR AFU. Αυτό το UUID πρέπει να το γράψουμε στο αρχείο *lfsr_afu.json* και να αντικαταστήσουμε αυτό που είχε ήδη μέσα. Έπειτα θα χρειαστεί να φορτώσουμε τα εργαλεία του Intel FPGA Lite με την εντολή `tools_setup -t QL 18.1` . Στην συνέχεια θα πάμε στο directory του lfsr_afu και θα τρέξουμε την εντολή `afu_synth_setup -s hw/rtl/filelist.txt build_synth` .

Αφού δημιουργηθεί ο φάκελος *build_synth* εκτελούμε την εντολή *cd* για να εισέλθουμε στο directory του και εκτελούμε την εντολή `run.sh` . Αυτή η εντολή θα χρησιμοποιήσει πληροφορίες από το *lfsr_afu.json* για να παράξει το αρχείο `afu_json_info.vh` . Η εντολή *run.sh* εκτελεί το πρόγραμμα της Intel Quartus Prime για να μεταγλωττίσει το AFU που

φτιάξαμε και να το μετατρέψει σε ένα κύκλωμα το οποίο μπορεί να ενσωματωθεί στο FPGA που επιλέξαμε.

Το Quartus Prime ξεκινάει με την εκτέλεση εργαλείων σύνθεσης τα οποία μεταγλωττίζουν τον πηγαίο κώδικα που φτιάξαμε στην SystemVerilog. Όταν υπάρχουν συντακτικά λάθη εμφανίζονται με κόκκινο χρώμα και πρέπει να τα διορθώσουμε και να εκτελέσουμε νέα μεταγλώττιση. Στην δική μας περίπτωση είχαμε αρκετά τέτοια λάθη το οποίο ήταν φυσιολογικό. Αφού τα διορθώσαμε και έγινε η μεταγλώττιση επιτυχώς το πρόγραμμα του Quartus δημιούργησε ένα αρχείο προγραμματισμού bit-stream για FPGAs με όνομα *lfsr_afu.gbs* . Στην φιλοσοφία των Intel FPGA ο τύπος gbs είναι γνωστός ως ένα *Green Bitstream* και αναπαριστά ένα αρχείο μερικής αναδιαμόρφωσης για το FPGA που επιλέχθηκε. Αυτό το αρχείο πρέπει να το κατεβάσουμε στην συσκευή του FPGA στο DevCloud, όπου θα εισέλθει στο κύριο bit-stream που υπάρχει ήδη στο FPGA. Η Intel αναφέρεται στο κύριο bit-stream ως *Blue Bitstream* . Για να κατεβάσουμε το AFU στο FPGA εκτελούμε τις εντολές:

```
PACSign PR -t UPDATE -H openssl_manager -i lfsr_afu.gbs -o lfsr_afu_unsigned.gbs
```

Πατάμε y (yes) για να απαντήσουμε στα ερωτήματα που προκύπτουν από την παραπάνω εντολή και έπειτα εκτελούμε την παρακάτω εντολή:

```
fpgasupdate lfsr_afu_unsigned.gbs
```

Τώρα που το AFU έχει κατέβει στην επιλεγμένη FPGA συσκευή μας, μπορούμε να αναπτύξουμε προγράμματα λογισμικού που θα τρέχουν στον επεξεργαστή και θα κάνουν χρήση του AFU μας.

3.3 Τρίτο Μέρος

Για την ανάπτυξη λογισμικού για AFUs η Intel παρέχει μία συλλογή εργαλείων ανοιχτής πηγής γνωστά ως *Open Programmable Acceleration Engine (OPAE)*. Σαν παράδειγμα ένα πρόγραμμα C που χρησιμοποιεί την υποδομή του OPAE για να αποκτήσει πρόσβαση στο AFU έχει αναπτυχθεί στην συγκεκριμένη εργασία. Ο παρακάτω κώδικας που αφορά το

πρόγραμμα αυτό περιέχει το αρχείο επικεφαλίδας *mmio.h* το οποίο είναι στην συλλογή του OPAE και απαιτείται για να πραγματοποιούνται I/O με αντιστοίχιση μνήμης.

Στις γραμμές 5 με 7 γίνεται ο καθορισμός των διευθύνσεων για να αποκτήσει πρόσβαση το λογισμικό στους καταχωρητές πολυωνύμου, ελέγχου και LFSR. Αυτές οι διευθύνσεις είναι οι ίδιες με αυτές που χρησιμοποιήσαμε στον κώδικα του δεύτερου μέρους αλλά έχουν μετατοπιστεί αριστερά κατά 2 μπιτ. Αυτή η μετατόπιση μπιτ γίνεται για να μετατραπούν οι διευθύνσεις τύπου double-word (32-bit aligned) σε διευθύνσεις byte που χρειάζονται για τον επεξεργαστή.

```
1  #include <stdio.h>
2  #include <opae/mmio.h>
3
4  // Application Logic register addresses (offsets)
5  #define POLY_REG    0X10 << 2
6  #define LFSR_REG    0X12 << 2
7  #define CTRL_REG    0X14 << 2
8
9  int open_AFU (fpga_handle *);
10 void close_AFU (fpga_handle);
11
12 int main(int argc, char *argv[])
13 {
14     fpga_handle handle = NULL;
15     if (open_AFU (&handle) < 0)
16         return -1;
17
18     uint32_t data;
19     (void) fpgaWriteMMIO32 (handle, 0, POLY_REG, 221);   // set polynomial
20     (void) fpgaReadMMIO32 (handle, 0, POLY_REG, &data);  // set seed
21     printf ("Polynomial set to: %d\n", data);
22
23     (void) fpgaWriteMMIO32 (handle, 0, LFSR_REG, 0x1);
24     (void) fpgaReadMMIO32 (handle, 0, LFSR_REG, &data);
25     printf ("Seed set to: %d\n", data);
26
27     bool found[256] = { false };
28     bool stop = false;
29     int length = 0;
30     while (!stop) {
31         if (found[data]) stop = true;
32         else {
33             ++length;
34             found[data] = true;
35
36             // get a new random integer from the LFSR //
37             (void) fpgaWriteMMIO32 (handle, 0, CTRL_REG, 0x1);  // step
38             (void) fpgaWriteMMIO32 (handle, 0, CTRL_REG, 0x0);  // stop
39             (void) fpgaReadMMIO32 (handle, 0, LFSR_REG, &data);
40             printf ("LFSR: %d\n", data);
41         }
42     }
43     printf("\nLength of random sequence: %d\n", length);
44
45     close_AFU (handle);
46     return 0;
47 }
48
```

Εικόνα 13 Ο πηγαίος κώδικας του τρίτου μέρους

Στις γραμμές 9 και 10 έχουμε δύο συναρτήσεις *open_AFU* και *close_AFU*. Η πρώτη είναι χρήσιμη στην ενεργοποίηση ενός μηχανισμού επικοινωνίας μεταξύ του προγράμματος και του AFU μέσω ενός προγράμματος οδήγησης του Linux. Με την συνάρτηση *close_AFU* απενεργοποιείται αυτός ο μηχανισμός. Η *open_AFU* καλεί μερικές λειτουργίες της βιβλιοθήκης του OPAE για να ελέγξει αν το AFU είναι διαθέσιμο και λειτουργεί κανονικά.

Αν ισχύει αυτή η συνθήκη, η παράμετρος handle που καθορίζεται μέσω του ειδικού τύπου του OPAE με όνομα `fpga_handle` στην γραμμή 14, ορίζεται ως δείκτης του AFU. Επίσης

η *open_AFU* χρησιμοποιεί αυτόν τον δείκτη για να εκτυπώσει στο Linux τερματικό τα περιεχόμενα των υποχρεωτικών διευθύνσεων των καταχωρητών στο AFU, τα οποία υπάρχουν στον κώδικα που υλοποιήσαμε στο δεύτερο μέρος της εργασίας.

Ο υπόλοιπος κώδικας της παρακάτω εικόνας χρησιμοποιεί I/O χαρτογραφημένης μνήμης μέσω της παραμέτρου handle για να έχει πρόσβαση στους καταχωρητές του LFSR AFU. Η συνάρτηση *fpgaReadMMIO32* επιτρέπει στο λογισμικό να διαβάζει τα περιεχόμενα ενός καταχωρητή AFU ενώ η *fpgaWriteMMIO32* επιτρέπει μια καινούρια μεταβλητή να γράφεται σε έναν καταχωρητή. Ο σκοπός του κώδικα είναι να χρησιμοποιήσει την while της γραμμής 30 για να κάνει το LFSR να αναπαράγει την σειρά των τυχαίων ακεραίων που θα δούμε στα αποτελέσματα αυτού του μέρους στο επόμενο κεφάλαιο. Η while τερματίζεται όταν το LFSR παράξει μια τιμή η οποία έχει αναπαραχθεί προηγουμένως έτσι ώστε να τερματιστεί η σειρά αυτή.

Για να μεταγλωττίσουμε τον κώδικα που φτιάξαμε παρακάτω, τον ονομάσαμε *part3.c* και τον τοποθετήσαμε στον υποφάκελο sw του lfsr_afu. Έπειτα τον αντιγράψαμε στο DevCloud και επίσης χρειαστήκαμε ένα αρχείο με όνομα *manage_afu.c* το οποίο έχει τον απαραίτητο κώδικα σε C για τις συναρτήσεις `open_AFU` και `close_AFU`. Επίσης μια προϋπόθεση για να γίνει σωστά η μεταγλώττιση ήταν να δημιουργήσουμε ένα συγκεκριμένο Makefile το οποίο χρησιμοποιεί την υποδομή OPAE στο Devcloud. Αφού συνδεθήκαμε στο DevCloud κάναμε αλλαγή διαδρομής στο sw και εκτελέσαμε την εντολή `make part3`. Στην συνέχεια το πρόγραμμα διάβασε το αρχείο *lfsr_afu.json* για να βρει το UUID του AFU μας και να παράξει το αρχείο επικεφαλίδας σε C με όνομα *afu_json_info.h* στο οποίο αναφερθήκαμε και προηγουμένως. Αυτό το αρχείο χρησιμοποιήθηκε και από το *manage_afu.c*. Για την εκτέλεση του προγράμματος μας εκτελέσαμε την εντολή `./part3` και τα αποτελέσματα όλων των εντολών θα τα αναλύσουμε στο επόμενο κεφάλαιο.

3.4 Τέταρτο Μέρος

Στο προηγούμενο μέρος το πρόγραμμα που φτιάξαμε παράγει 15 τυχαίες τιμές βάση του πολυωνύμου που έχει την τιμή 221 και ξεκινάει με την τιμή 1 για το seed. Γενικότερα ένα LFSR μπορεί να παράξει σειρές από διαφορετικά μήκη ανάλογα τον συνδυασμό των τιμών του πολυωνύμου και του seed. Ένα LFSR με n-bit το οποίο έχει ένα πολυώνυμο το οποίο έχει ως αποτέλεσμα να παράξει το μέγιστο μήκος σειράς δηλαδή 2^n διαφορετικές τιμές, ονομάζεται primitive πολυώνυμο. Σε αυτό το μέρος φτιάξαμε ένα πρόγραμμα σε C το οποίο χρησιμοποιεί το LFSR AFU μας για να βρει όλα τα primitive πολυώνυμα των 8 μπιτ. Το μικρότερο πολυώνυμο που έπρεπε να εξετάσουμε είναι το Poly = $(80)_{16}$ = 128 και το μεγαλύτερο ήταν το Poly = $(FF)_{16}$ = 255. Όπως βλέπουμε και στον κώδικα παρακάτω, πολλές εντολές μένουν ίδιες με τον κώδικα του τρίτου μέρους. Αρχικά προσθέσαμε έναν πίνακα `primitives` με 128 θέσεις στον οποίο αποθηκεύονταν τα πρωτόγονα πολυώνυμα που βρίσκαμε. Έπειτα ορίσαμε άλλη μια `int` μεταβλητή για να μετράει τον αριθμό τον πολυωνύμων που θα βρίσκαμε.

```c
#include <stdio.h>
#include <opae/mmio.h>

// AFU addresses shifted left by two to convert the double-word address to byte address
#define POLY_REG 0X10 << 2
#define LFSR_REG 0X12 << 2
#define CTRL_REG 0X14 << 2

int open_AFU (fpga_handle *); // Sets up a communication mechanism between the program and the AFU
void close_AFU (fpga_handle); // via a linux device driver and then terminates the connection

int main(int argc, char *argv[]){
    fpga_handle handle = NULL; // This handle variable sets up a pointer to the AFU
    if (open_AFU (&handle) < 0)
        return -1;

    int primitives[128] = {0};
    uint32_t data;
    int length, j = 0;

    for(int i=128; i<256; i++){
        (void) fpgaWriteMMIO32 (handle, 0, POLY_REG, i);
        (void) fpgaReadMMIO32 (handle, 0, POLY_REG, &data);
        printf ("\nPolynomial set to: %d\n", data);

        (void) fpgaWriteMMIO32 (handle, 0, LFSR_REG, 0x1);
        (void) fpgaReadMMIO32 (handle, 0, LFSR_REG, &data);
        printf ("Seed set to: %d\n", data);

        bool found[256] = { false };
        bool stop = false;

        while (!stop) {
            if (found[data]) stop = true;
            else {
                ++length;
                found[data] = true;

                // get a new random integer from the LFSR //
                (void) fpgaWriteMMIO32 (handle, 0, CTRL_REG, 0x1);  // step
                (void) fpgaWriteMMIO32 (handle, 0, CTRL_REG, 0x0);  // stop
                (void) fpgaReadMMIO32 (handle, 0, LFSR_REG, &data);
                // printf ("LFSR: %d\n", data);
            }
        }
    printf("Length of random sequence: %d\n", length);
        if(length == ((1 << 8)-1)) {
            printf("%d is a primitive polynomial with a length of %d! \n",i, length);
            primitives[j] = i; // Store the primitive polynomial
        j++ ;
        }

    length = 0; // Reset the length for the next iteration
```

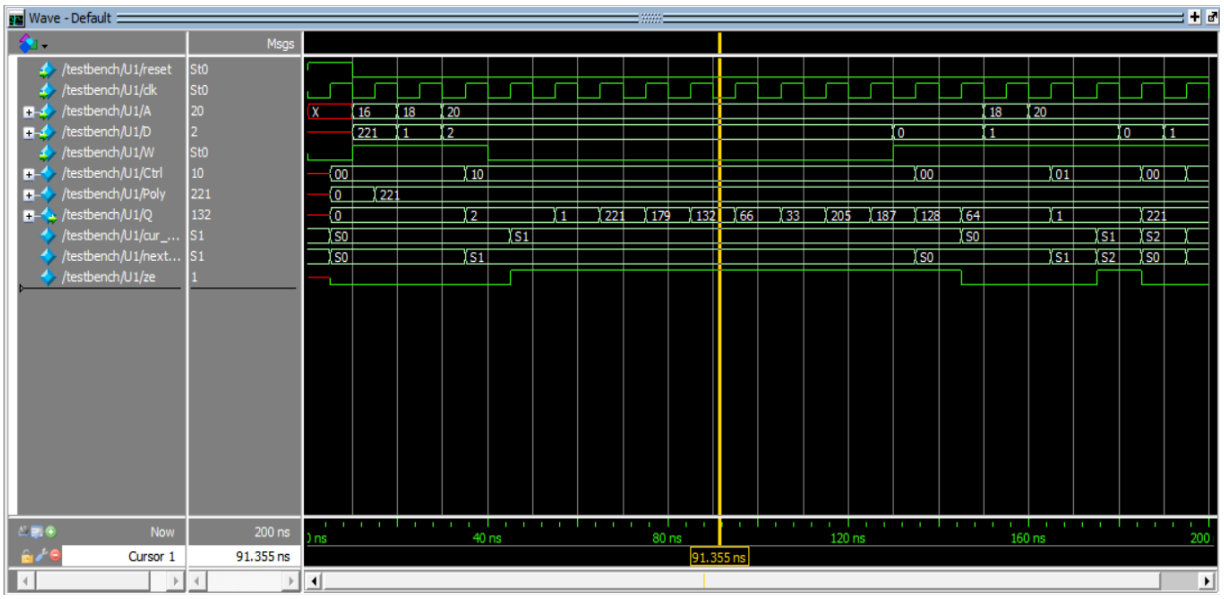
Εικόνα 14 Ο πηγαίος κώδικας του τέταρτου μέρους

Για να εξετάσουμε κάθε πολυώνυμο από την τιμή 128 μέχρι την τιμή 255 βάλαμε μια `for` loop στην οποία όπως και στο τρίτο μέρος γράφουμε στον καταχωρητή πολυωνύμου και LFSR τις τιμές για το πολυώνυμο και το seed αντίστοιχα. Στην `while` loop όπως αναφέραμε το LFSR παράγει νέες τιμές μέχρι να βρει κάποια που έχει αναπαραχθεί προηγουμένως και σταματάει. Έπειτα αφού εκτυπωθεί το μήκος της τυχαίας σειράς αριθμών εξετάσαμε με μία `if` αν αυτό είναι ίσο με 255 ώστε να αποθηκεύσουμε το πολυώνυμο στην λίστα με τα πρωτόγονα πολυώνυμα. Στο τέλος εκτυπώνουμε με μια `for` loop τα στοιχεία του πίνακα `primitives` ένα-ένα.

4. Αποτελέσματα σχεδίασης του LFSR AFU

4.1 Πρώτο μέρος

Παρακάτω βλέπουμε τα αποτελέσματα της προσομοίωσης του application module που φτιάξαμε. Τα σήματα που δίνονται σε κάθε χρονική στιγμή γίνονται μέσω του testbench.v αρχείου. Το ρολόι του κυκλώματος είναι χρονισμένο στα 10ns.

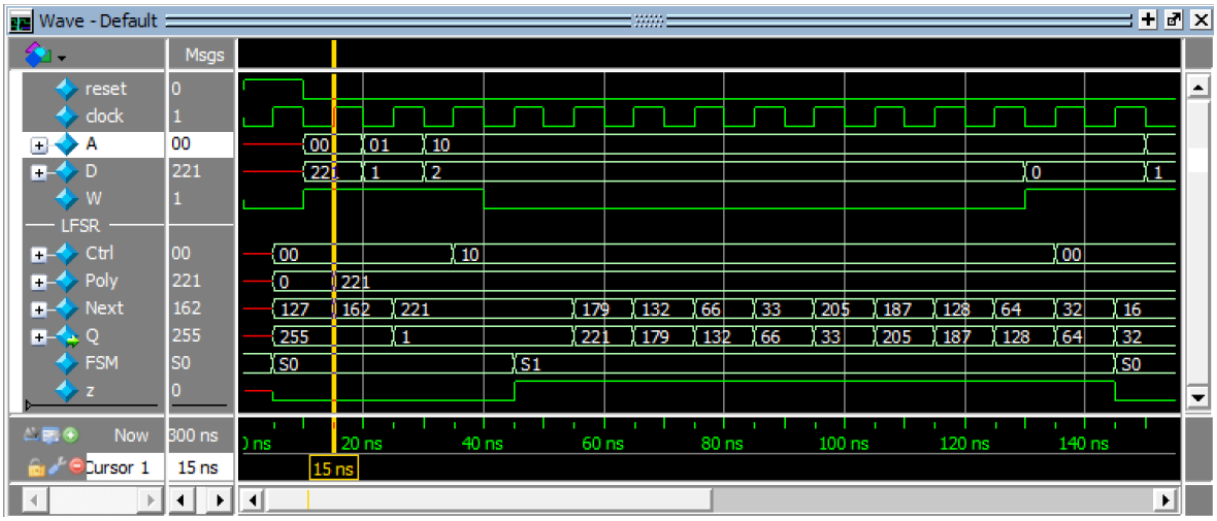


Εικόνα 15 Η κυματομορφή των εισόδων/εξόδων του Application module

Παρατηρούμε ότι το A παίρνει την τιμή 16_{10} την χρονική στιγμή 10ns και στην επόμενη ακμή του ρολογιού ενεργοποιείται ο καταχωρητής πολυωνύμου με τιμή 221 την οποία παίρνει λόγω της εισόδου D. Όταν το A παίρνει την τιμή 20_{10} ενεργοποιείται ο Control Register με τιμή 10_2 και τότε είναι που το Q αρχίζει να παράγει τυχαίες τιμές αφού το LFSR λειτουργεί σε continuous mode. Παρατηρούμε παράλληλα ότι την χρονική στιγμή 35ns αλλάζει η κατάσταση της FSM σε S1 λόγω της αλλαγής της τιμής του control register. Όταν το control επιστρέφει στην τιμή 00 η μηχανή καταστάσεων επαναφέρεται στην αρχική

κατάσταση στον επόμενο παλμό ρολογιού στα 135ns. Επίσης βλέπουμε ότι το Control αλλάζει την χρονική στιγμή 165ns σε 01_2 όπου αντίστοιχα η FSM πηγαίνει στις καταστάσεις S1 και S2 σε διαδοχικούς παλμούς ρολογιού. Έτσι επιτυγχάνεται το step mode όπου το LFSR παράγει μια μόνο τιμή την 221 την χρονική στιγμή 185ns. Έπειτα η FSM επιστρέφει στην κατάσταση S0 κατά την χρονική στιγμή 195ns.

Παρακάτω παρατηρούμε στην κυματομορφή του tutorial που έχουμε σαν επαληθευτικό μοντέλο ότι οι τιμές των εισόδων και των καταχωρητών συμπεριφέρονται παρόμοια με την δικιά μας υλοποίηση. Οι διαφορές που παρατηρούμε είναι στις τιμές του A οι οποίες είναι δυαδικής μορφής ενώ εμάς δεκαδικής. Επίσης ο συντάκτης του tutorial έχει επιλέξει να φαίνονται οι επόμενες τιμές που θα πάρει το Q ενώ εμείς διαλέξαμε να βλέπουμε τις επόμενες καταστάσεις της FSM.



Εικόνα 16 Η κυματομορφή του Application Module από το Tutorial. Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf

4.2 Δεύτερο μέρος

Παρακάτω θα δούμε κάποια αποσπάσματα από τα logs που κρατήσαμε κατά την διάρκεια της εκτέλεσης των εντολών `build_synth` και `run.sh`. Θυμίζουμε από το προηγούμενο

κεφάλαιο ότι η *build_synth* είναι υπεύθυνη για την δημιουργία των απαραίτητων αρχείων στα οποία θα δουλέψει η *run.sh* η οποία με την σειρά της θα κάνει το compile

του AFU ώστε να είναι συμβατό με το FPGA μας. Μιας και τα logs που εμφανίζονται στο terminal μας είναι αρκετά μεγάλα σε μέγεθος, κρατήσαμε κάποια στιγμιότυπα από την αρχή και το τέλος των εντολών που εκτελέστηκαν για την σύνθεση του AFU.

Όπως βλέπουμε στο πρώτο στιγμιότυπο το DevCloud ανοίγει ένα shell του Intel Quartus Prime ώστε να εκτελεστούν οι απαραίτητες εντολές μεταγλώττισης και αποσφαλμάτωσης.

```
u143862@s005-n007:~/lfsr_afu/build_synth$ run.sh
Restoring blue bitstream lib files
==================================
Info: ************************************************************
Info: Running Quartus Prime Shell
    Info: Version 19.2.0 Build 57 06/24/2019 Patches 0.01rc SJ Pro Edition
    Info: Copyright (C) 2019  Intel Corporation. All rights reserved.
    Info: Your use of Intel Corporation's design tools, logic functions
    Info: and other software and tools, and any partner logic
    Info: functions, and any output files from any of the foregoing
    Info: (including device programming or simulation files), and any
    Info: associated documentation or information are expressly subject
    Info: to the terms and conditions of the Intel Program License
    Info: Subscription Agreement, the Intel Quartus Prime License Agreement,
    Info: the Intel FPGA IP License Agreement, or other applicable license
    Info: agreement, including, without limitation, that your use is for
    Info: the sole purpose of programming logic devices manufactured by
    Info: Intel and sold by Intel or its authorized distributors.  Please
    Info: refer to the applicable agreement for further details, at
    Info: https://fpgasoftware.intel.com/eula.
    Info: Processing started: Fri Oct  7 06:37:16 2022
Info: Command: quartus_sh -t ./a10_partial_reconfig/flow.tcl -nobasecheck -setup_scrip
t ./a10_partial_reconfig/setup.tcl -impl afu_default
Info: Quartus(args): -nobasecheck -setup_script ./a10_partial_reconfig/setup.tcl -impl
afu_default
Info: flow.tcl version: #1
Info: Using setup script /home/u143862/lfsr_afu/build_synth/build/a10_partial_reconfig
/setup.tcl
Arria 10 Partial Reconfiguration Flow
--------------------------------------------------------------------------------
```

Εικόνα 17 Τα πρώτα logs της εντολής run.sh

Στο δεύτερο στιγμιότυπο παρατηρούμε ότι γίνεται χρήση ενός εργαλείου IP generation.

```
--------------------------------------------------------------------------------
   Project name                    : dcp
   Output directory                : output_files
   Base revision name              : dcp
   Reconfigurable partition names : green_region
   Implementation Revision : afu_default
      Reconfigurable Partition Name : green_region
Info: Compiling PR implementation afu_default.
Info: ************************************************************
Info: Running Quartus Prime IP Generation Tool
    Info: Version 19.2.0 Build 57 06/24/2019 Patches 0.01rc SJ Pro Edition
    Info: Processing started: Fri Oct  7 06:37:22 2022
Info: Command: quartus_ipgenerate dcp -c afu_default --run_default_mode_op
Info: Found 1 IP file(s) in the project.
    Info: IP file /home/u143862/lfsr_afu/build_synth/build/platform/AFU_debug/SCJIO.qs
ys was found in the project.
Info: Started running qsys-validate on Platform Designer system /home/u143862/lfsr_afu
/build_synth/build/platform/AFU_debug/SCJIO.qsys
Info: Performing Platform Designer system validation using the command line: /glob/dev
elopment-tools/versions/fpgasupportstack/a10/1.2.1/intelFPGA_pro/quartus/../qsys/bin/
q      sys-validate
/home/u143862/lfsr_afu/build_synth/build/platform/AFU_debug/SCJIO.qsys
Info: SCJIO: All Generic Component instances match their respective ip files.
Info: Finished running qsys-validate on Platform Designer system /home/u143862/lfsr_af
u/build_synth/build/platform/AFU_debug/SCJIO.qsys
Info: Elaborating Platform Designer system entity /home/u143862/lfsr_afu/build_synth/b
```

Εικόνα 18 Συνέχεια των logs της εντολής run.sh

Τέλος παρατηρούμε ότι αφού τελειώσει η εκτέλεση της μεταγλώττισης, μας εμφανίζεται το μήνυμα ότι δημιουργήθηκε το αρχείο *lfsr_afu.gbs*. Το αρχείο αυτό είναι ένα green bitstream που αναπαριστά ένα αρχείο μερικής αναδιαμόρφωσης για το FPGA στο οποίο πρόκειται να λειτουργήσει. Κατεβάσαμε αυτό το αρχείο στην FPGA συσκευή του DevCloud, ώστε να συνδεθεί με το κύριο bit-stream της συσκευής το οποίο λέγεται αλλιώς και *blue bitstream.*

```
    Info (332172): Hold clock transfer from mem|ddr4a|ddr4a_core_usr_clk (Rise) to
mem|ddr4a|ddr4a_phy_clk_1_1 (Rise) has uncertainty 0.320 that is less than the recommended
uncertainty 0.360
    Info (332172): Setup clock transfer from mem|ddr4a|ddr4a_core_usr_clk (Rise) to
mem|ddr4a|ddr4a_phy_clk_1_2 (Rise) has uncertainty 0.289 that is less than the recommended
uncertainty 0.360
    Info (332172): Hold clock transfer from mem|ddr4a|ddr4a_core_usr_clk (Rise) to
mem|ddr4a|ddr4a_phy_clk_1_2 (Rise) has uncertainty 0.320 that is less than the recommended
uncertainty 0.360
Warning (332088): No paths exist between clock target
"fpga_top|inst_fiu_top|inst_ccip_fabric_top|inst_fme_top|inst_fme_csr|bmc|spi_bridge|
spi_slave_to_avalon_mm_master_bridge_1|the_spislave_inst_for_spichain|the_SPIPhy|SPIP
hy_MOSIctl|filtered_sclk_negedge|q" of clock "filtered_sclk_negedge" and its clock
source. Assuming zero source clock latency.
Info (23030): Evaluation of Tcl script ./a10_partial_reconfig/report_timing.tcl was
successful
Info: Quartus Prime Timing Analyzer was successful. 0 errors, 184 warnings
    Info: Peak virtual memory: 3915 megabytes
    Info: Processing ended: Fri Oct  7 07:39:49 2022
    Info: Elapsed time: 00:00:16
Info (19538): Reading SDC files took 00:00:01 cumulatively in this process.
Wrote lfsr_afu.gbs

================================================================================
 PR AFU compilation complete
 AFU gbs file is 'lfsr_afu.gbs'
 Design meets timing
================================================================================
```

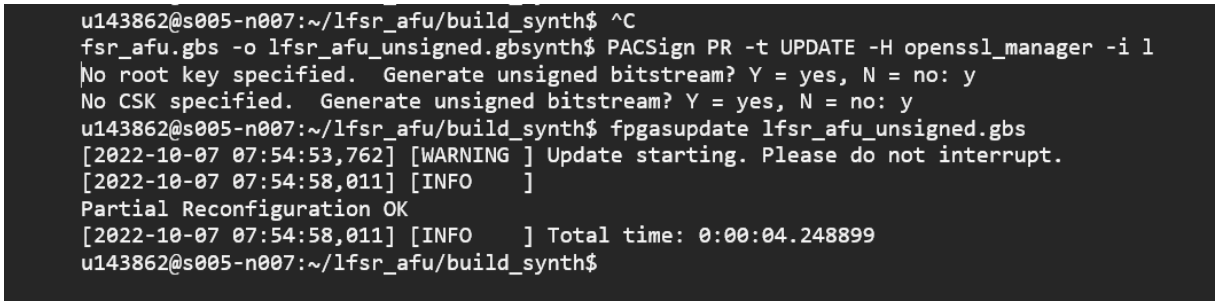
Εικόνα 19 Η επιτυχής δημιουργία του Green Bitstream

Για να κατεβάσουμε το AFU στο FPGA την εντολή:

```
PACSign PR -t UPDATE -H openssl_manager -I lfsr_afu.gbs -o lfsr_afu_unsigned.gbs
```

Έπειτα πληκτρολογήσαμε y (yes) στα ερωτήματα που εμφανίζονταν στο τερματικό από την εντολή και εκτελέσαμε την παρακάτω εντολή:

```
fpgasupdate lfsr_afu_unsigned.gbs
```



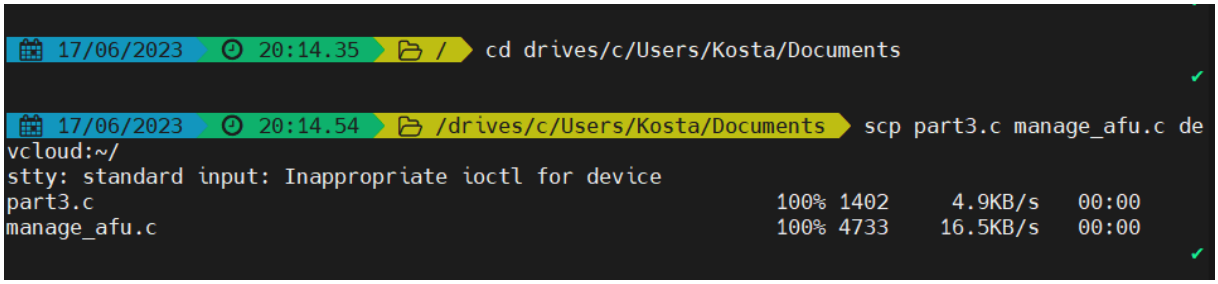
Εικόνα 20 Εκτέλεση των απαραίτητων εντολών για το κατέβασμα του AFU στο FPGA

Εφόσον ολοκληρώθηκε η διαδικασία του partial reconfiguration και το AFU κατέβηκε στην συσκευή FPGA, μπορούσαμε να συνεχίσουμε με την ανάπτυξη προγραμμάτων σε C τα οποία τρέχουν στον επεξεργαστή και κάνουν χρήση του AFU

4.3 Τρίτο Μέρος

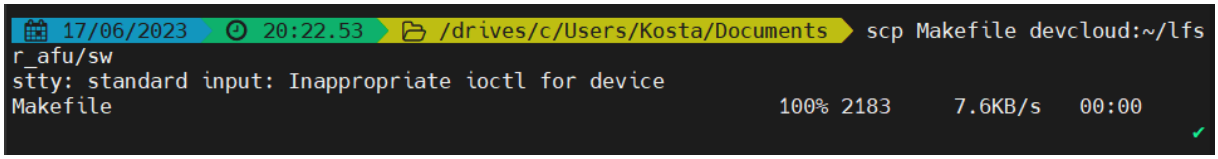
Σε αυτό το υποκεφάλαιο θα περάσουμε στα αποτελέσματα των εντολών που χρησιμοποιήσαμε για την εκτέλεση του προγράμματος `part3.c`. Αρχικά αντιγράψαμε το

αρχείο αυτό όπως και το `manage_afu.c` όπως αναφέραμε και στο δεύτερο κεφάλαιο με την παρακάτω εντολή:



Εικόνα 21 Αντιγραφή των αρχείων του τρίτου μέρους στο DevCloud

Στην συνέχεια αντιγράψαμε επίσης το Makefile που χρειάζεται για την υποδομή του OPAE στον φάκελο του software όπως βλέπουμε παρακάτω:



Εικόνα 22 Αντιγραφή του Makefile στο DevCloud

Εφόσον συνδεθήκαμε στο DevCloud, πήγαμε στο directory του sw και τρέξαμε την εντολή make part3. Ένα από τα προγράμματα που εκτελεί η εντολή make είναι το *afu_json_mgr*. Το συγκεκριμένο πρόγραμμα διαβάζει το αρχείο *lfsr_afu.json* για να βρει το UUID για το AFU και μετά παράγει το αρχείο επικεφαλίδας C *afu_json_info.h*. Αυτό το αρχείο επικεφαλίδας χρησιμοποιείται από το *manage_afu.c*. Στην συνέχεια λοιπόν για να εκτελέσουμε το πρόγραμμα που μεταγλωττίσαμε, εκτελέσαμε την εντολή `./part3` και παρακάτω μπορούμε να δούμε τα αποτελέσματα της.

```
u143862@s005-n007:~/lfsr_afu/sw$ ls
afu_json_info.h  defines.h  Makefile  manage_afu.c  manage_afu.o  part3.c  part4.c  part4.o
u143862@s005-n007:~/lfsr_afu/sw$ make part3
gcc -fstack-protector -fPIE -fPIC -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror -g
 -O2 -std=c99 -Wall -Wno-unknown-pragmas -c -o part3.o part3.c
gcc -fstack-protector -fPIE -fPIC -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror -g
 -O2 -std=c99 -Wall -Wno-unknown-pragmas -o part3 part3.o manage_afu.o -z noexecstack -z relro
-z now -pie -luuid -lpthread -lopae-c
u143862@s005-n007:~/lfsr_afu/sw$ ls
afu_json_info.h  Makefile      manage_afu.o  part3.c  part4.c
defines.h        manage_afu.c  part3         part3.o  part4.o
u143862@s005-n007:~/lfsr_afu/sw$ ./part3
Opening lfsr_afu
AFU DFH REG  = 0x1000010000000000
AFU ID HI    = 0xa72eee454bda4a55
AFU ID LO    = 0xbe0544c1b17512b0
AFU NEXT     = 0x0000000000000000
AFU RESERVED = 0x0000000000000000
Polynomial set to: 221
Seed set to: 1
LFSR: 221
LFSR: 179
LFSR: 132
LFSR: 66
LFSR: 33
LFSR: 205
LFSR: 187
LFSR: 128
LFSR: 64
LFSR: 32
LFSR: 16
LFSR: 8
LFSR: 4
LFSR: 2
LFSR: 1

Length of random sequence: 15
```

Εικόνα 23 Μεταγλώττιση και εκτέλεση του προγράμματος part3.c

4.4 Τέταρτο μέρος

Για να εκτελέσουμε το πρόγραμμα που φτιάξαμε όπως και στο τρίτο μέρος αρχικά αντιγράψαμε το αρχείο στο DevCloud από τον υπολογιστή μας με χρήση της παρακάτω εντολής:

```
29/06/2023  23:05.13  /drives/c/Users/kosta/Desktop  scp part4.c devcloud:~/
stty: standard input: Inappropriate ioctl for device
part4.c                                                    100% 2154     8.0KB/s   00:00
```

Εικόνα 24 Αντιγραφή του αρχείου του τέταρτου μέρους στο DevCloud

Με τον γνωστό τρόπο συνδεθήκαμε στο DevCloud στην συνέχεια, εκτελέσαμε την εντολή και εκτελέσαμε τις παρακάτω εντολές για να μεταγλωττιστεί σωστά το πρόγραμμα μας:

```
u143862@s005-n004:~/lfsr_afu/sw$ tools_setup -t QL 18.1
sourcing /glob/development-tools/versions/intelFPGA_lite/18.1/init_quartus.sh

u143862@s005-n004:~/lfsr_afu/sw$ make part4
gcc -fstack-protector -fPIE -fPIC -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror -g -O2 -std=c99 -Wall -
Wno-unknown-pragmas -c -o part4.o part4.c
gcc -fstack-protector -fPIE -fPIC -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror -g -O2 -std=c99 -Wall -
Wno-unknown-pragmas -o part4 part4.o manage_afu.o -z noexecstack -z relro -z now -pie -luuid -lpthread -lopae-c
```

Εικόνα 25 Μεταγλώττιση του προγράμματος part4.c

Έπειτα με την χρήση της εντολής `./part4` είχαμε την έξοδο που επιθυμούσαμε. Σε αυτήν την περίπτωση διαλέξαμε να μην εμφανίζονται η τιμές που έπαιρνε το lfsr για κάθε πολυώνυμο ώστε να μην έχουμε ένα υπερβολικά μεγάλο log file στην έξοδο. Διαλέξαμε για κάθε επανάληψη του βρόγχου να μας εμφανίζει μόνο τις τιμές του πολυωνύμου και του seed όπως και το μήκος της ακολουθίας. Όπως βλέπουμε στο παρακάτω στιγμιότυπο τα πρωτόγονα πολυώνυμα είναι 16.

```
Polynomial set to: 250
Seed set to: 1
Length of random sequence: 255
250 is a primitive polynomial with a length of 255!

Polynomial set to: 251
Seed set to: 1
Length of random sequence: 105

Polynomial set to: 252
Seed set to: 1
Length of random sequence: 85

Polynomial set to: 253
Seed set to: 1
Length of random sequence: 126

Polynomial set to: 254
Seed set to: 1
Length of random sequence: 127

Polynomial set to: 255
Seed set to: 1
Length of random sequence: 9
Primitive polynomials are:  142 149 150 166 175 177 178 180 184 195 198 212 225 231 243 250
```

Εικόνα 26 Εκτέλεση του προγράμματος part4.c

Και με την ολοκλήρωση της εκτέλεσης του part4 ολοκληρώσαμε τα αποτελέσματα που θέλαμε για την συγκεκριμένη πτυχιακή εργασία.

5. Δυσκολίες-Συμπεράσματα

Για την υλοποίηση της συγκεκριμένης πτυχιακής εργασίας παρουσιάστηκαν αρκετές δυσκολίες τις οποίες έπρεπε να αντιμετωπίσω τόσο στην κατανόηση των προβλημάτων που έπρεπε να λύσω όσο και στην διαχείριση του λογισμικού που χρειαζόταν για την εκπόνηση της. Αρχικά δυσκολεύτηκα στην κατανόηση της λειτουργίας των μηχανισμών του AFU καθώς δεν είχα απασχοληθεί με ένα κύκλωμα το οποίο να συνδυάζει τόσους μηχανισμούς. Επίσης ήταν το πρώτο μου project πάνω στην ανάπτυξη ετερογενών συστημάτων οπότε δυσκολεύτηκα να βρω τον τρόπο με τον οποίο θα συνδύαζα την χρήση διαφορετικών τεχνολογιών για την υλοποίηση του. Έπειτα κατά την σχεδίαση και δοκιμή του AFU στην γλώσσα της System Verilog έπρεπε να χρησιμοποιήσω το πρόγραμμα του ModelSim με το οποίο είχα διάφορες δυσλειτουργίες ως προς την επεξεργασία του project το οποίο είχα δημιουργήσει σε αυτό από διαφορετικούς υπολογιστές. Έπειτα σχετικά με την πλατφόρμα του DevCloud υπήρχαν και εκεί δοκιμασίες που έπρεπε να αντιμετωπίσω καθώς είχα ελάχιστη εμπειρία πάνω σε εικονικά περιβάλλοντα του Linux. Παρ’ όλα αυτά μου φάνηκε αρκετά ενδιαφέρον το πως λειτουργεί το συγκεκριμένο εργαλείο και ήξερα πως χρειάζεται κάποιος να έχει ασχοληθεί με παρόμοια περιβάλλοντα πάνω στον τομέα των ενσωματωμένων συστημάτων τον οποίο έχω ακολουθήσει στην μέχρι τώρα εργασιακή μου εμπειρία.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Wikipedia, 2014. Heterogenous computing [online] Διαθέσιμο στην ιστοσελίδα: https://en.wikipedia.org/wiki/Heterogeneous_computing [Ανακτήθηκε στις 12 Ιανουαρίου 2023]

[2] Infolab Stanford, 1998. Heterogenous computing [online] Διαθέσιμο στην ιστοσελίδα: http://infolab.stanford.edu/~burback/dadl/node95.html [Ανακτήθηκε στις 13 Ιανουαρίου 2023]

[3] Gigabyte, Heterogenous Computing [online] Διαθέσιμο στην ιστοσελίδα:

https://www.gigabyte.com/Glossary/heterogeneous-computing [Ανακτήθηκε στις 14 Ιανουαρίου 2023]

[4] GitHub, Devcloud Access Instructions [online] Διαθέσιμο στην ιστοσελίδα:

https://github.com/intel/FPGA-Devcloud/tree/master/main/Devcloud_Access_Instructions#30-access-from-your-pc-via-mobaxterm-or-from-linux-terminal [Ανακτήθηκε στις 17 Ιανουαρίου 2023]

[5] Intel. Intel Developer Cloud [online] Διαθέσιμο στην ιστοσελίδα: https://www.intel.com/content/www/us/en/developer/tools/devcloud/overview.html [Ανακτήθηκε στις 1 Φεβρουαρίου 2023]

[6] zdnet. AI Developer why you should be using Intel Devcloud [online] Διαθέσιμο στην ιστοσελίδα:

https://www.zdnet.com/paid-content/article/ai-developer-why-you-should-be-using-intel-devcloud/ [Ανακτήθηκε στις 2 Φεβρουαρίου 2023]

[7] oneAPI. OneAPI Programming Model [online] Διαθέσιμο στην ιστοσελίδα:

https://www.oneapi.io/ [Ανακτήθηκε στις 2 Φεβρουαρίου 2023]

[8] Bittware. oneAPI Software Tool Flow Frame for FPGAs by Intel –

https://www.bittware.com/ip-solutions/intel-oneapi/ [Ανακτήθηκε στις 2 Φεβρουαρίου 2023]

[9] Wikipedia. oneAPI (compute acceleration) [online] Διαθέσιμο στην ιστοσελίδα:

https://en.wikipedia.org/wiki/OneAPI_(compute_acceleration) [Ανακτήθηκε στις 2 Φεβρουαρίου 2023]

[10] Codeplay Software Ltd. oneAPI [online] Διαθέσιμο στην ιστοσελίδα:

https://codeplay.com/solutions/oneapi/ [Ανακτήθηκε στις 3 Φεβρουαρίου 2023]

[11] All About Circuits. What is a Hardware Description Language (HDL)? [online] Διαθέσιμο στην ιστοσελίδα: https://www.allaboutcircuits.com/technical-articles/what-is-a-hardware-description-language-hdl/ [Ανακτήθηκε στις 3 Φεβρουαρίου 2023]

[12] Chipverify. SystemVerilog Tutorial [online] Διαθέσιμο στην ιστοσελίδα: https://www.chipverify.com/systemverilog/systemverilog-tutorial [Ανακτήθηκε στις 3 Φεβρουαρίου 2023]

[13] Doulos. SystemVerilog Data Types [online] Διαθέσιμο στην ιστοσελίδα: https://www.doulos.com/knowhow/systemverilog/systemverilog-tutorials/systemverilog-data-types/ [Ανακτήθηκε στις 4 Φεβρουαρίου 2023]

[14] Xilinx. What is an FPGA? Field Programmable Gate Array [online] Διαθέσιμο στην ιστοσελίδα: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html [Ανακτήθηκε στις 5 Φεβρουαρίου 2023]

[15] Lattice Semiconductor. What is an FPGA [online] Διαθέσιμο στην ιστοσελίδα: https://www.latticesemi.com/en/What-is-an-FPGA [Ανακτήθηκε στις 5 Φεβρουαρίου 2023]

[16] Intel. Intel FPGA Basics and Getting Started [online] Διαθέσιμο στην ιστοσελίδα: https://www.intel.com/content/www/us/en/support/programmable/support-resources/fpga-training/getting-started.html [Ανακτήθηκε στις 5 Φεβρουαρίου 2023]

[17] W3Schools. Introduction to C [online] Διαθέσιμο στην ιστοσελίδα: https://www.w3schools.com/c/c_intro.php [Ανακτήθηκε στις 5 Φεβρουαρίου 2023]

[18] C (programming language) [online] Διαθέσιμο στην ιστοσελίδα: https://en.wikipedia.org/wiki/C_(programming_language) [Ανακτήθηκε στις 7 Φεβρουαρίου 2023]

[19] What is C Programming Language? Basics, Introduction, History [online] Διαθέσιμο στην ιστοσελίδα: https://www.guru99.com/c-programming-language.html [Ανακτήθηκε στις 7 Φεβρουαρίου 2023]

[20] OPAE Laboratory Exercise 1 [online] Διαθέσιμο στην ιστοσελίδα: https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Laboratory_Exercises/Heterogeneous_Computing/lab1.pdf [Ανακτήθηκε στις 11 Φεβρουαρίου 2023]