Technical University of Crete

Diploma Thesis

Design of a Distributed Producer-Consumer System for Golomb Ruler Computation on a Reconfigurable Supercomputer

Author:
Giorgos Foteinopoulos

Supervisor:
Prof. Apostolos Dollas

Committee:
Prof. Dionisios Pnevmatikatos (NTUA)
Assoc. Prof. Eftihios Koutroulis

Microprocessors and Hardware Lab
School of Electrical & Computer Engineering

December 12, 2019

TECHNICAL UNIVERSITY OF CRETE

Abstract

Electronic and Computer Engineer

Design of a Distributed Producer-Consumer System for Golomb Ruler Computation on a Reconfigurable Supercomputer

by Giorgos Foteinopoulos

An optimal Golomb ruler is a set of distinct positive integers (marks) such that the differences, computed over all different pairs, are distinct, and they occupy less space than any other ruler with the same number of marks. Optimal Golomb rulers are utilized in a great variety of applications over many scientific fields, with their derivation being computationally expensive and extremmelly time-consuming, despite the continuous growth of technology. However, finding and proving a Golomb ruler to be optimal is a problem that can be parallelized. Therefore there is a great incentive to design a custom solution based on FPGA technology. Our design utilizes Convey HC-2ex, a heterogeneous multi-fpga computing platform, that allows us to effectively use many levels of parallelism, in order to improve previous architectures focused on solving this problem. Following the evaluation of the HC-2ex results, we relocated our design to two contemporary FPGAs, capitalizing on the evolution of HDL designing tools. Despite the greater scale of the HC-2ex platform, we confronted spatial limitations, due to the low performance of the older designing tool. On the other hand, the results from the two FPGAs are significantly more promising, as we achieve close to perfect utilization of their available resources, with a high clock, resulting to a potential speedup of 3X over the HC-2ex, down to 2X in the case of larger OGRs.

TECHNICAL UNIVERSITY OF CRETE

Abstract

Electronic and Computer Engineer

Σχεδίαση Κατανεμημένου Συστήματος Παραγωγού-Καταναλωτή για Υπολογισμό Κανόνων Golomb σε Αναδιατασσόμενο Υπερυπολογιστή

by Giorgos FOTEINOPOULOS

Ένας βέλτιστος κανόνας Golomb είναι ένα σύνολο απο διακριτούς θετικούς ακεραίους (σημεία), τέτοιους ώστε όλες οι διαφορές, που μετρώνται από κάθε διαφορετικό ζευγάρι, να είναι μοναδικές, και να καταλαμβάνουν το μικρότερο χώρο σε σχέση με οποιοδήποτε άλλο κανόνα που έχει τον ίδιο αριθμό σημείων. Οι βέλτιστοι κανόνες Golomb χρησιμοποιούνται σε ένα ευρύ φάσμα εφαρμογών πολλών επιστημονικών τομέων, αλλά η εύρεση τους είναι ακριβή υπολογιστικά και εξαιρετικά χρονοβόρα, παρά την συνεχιζόμενη ανάπτυξη της τεχνολογίας. Ωστόσο, είναι ένα πρόβλημα που μπορεί να παραλληλοποιηθει. Συνεπώς υπάρχει μεγάλο κίνητρο για να σχεδιάσουμε μία προσαρμοσμένη λύση βασισμένη στην τεχνολογία των FPGA. Η σχεδίασή μας υλοποιείται στο Convey HC-2ex, μια ετερογενή υπολογιστική πλατφόρμα πολλών FPGA, που μας επιτρέπει να χρησιμοποιήσουμε αποτελεσματικά πολλά επίπεδα παραλληλοποίησης, με σκοπό να βελτιώσουμε προηγούμενες αρχιτεκτονικές που εστίαζαν στην λύση αυτού του προβλήματος. Ακολουθώντας την αξιολόγηση των αποτελεσμάτων του HC-2ex, μεταφέραμε την σχεδίαση μας σε δύο σύγχρονες FPGA αξιοποιώντας την εξέλιξη των σχεδιαστικών εργαλείων βασισμένων σε γλώσσες περιγραφής υλικού. Παρά την μεγάλη κλίμακα της πλατφόρμας HC-2ex, αντιμετωπίσαμε περιορισμούς όσον αφορά τον χώρο, εξαιτίας της χαμηλής απόδοσης των παλιότερων σχεδιαστικών εργαλείων. Αντιθέτως, τα αποτελέσματα από τις δύο FPGA είναι αρκετά υποσχόμενα, καθώς πετυχαίνουμε σχεδόν τέλεια αξιοποίηση των διαθέσιμων πόρων, διατηρώντας υψηλό ρολόι, και καταλήγοντας σε ένα πιθανό speedup 3x σε σχέση με το HC-2ex, που πέφτει στο 2x στην περίπτωση μεγαλύτερων βέλτιστων κανόνων Golomb.

Acknowledgements

First of all, I would like to thank my supervisor Professor Apostolos Dollas for his valuable guidance during the course of this thesis, as well as for the opportunity to experiment with a top of the art, at the time, multi-FPGA system.

Furthermore I am extremely grateful to the deus ex machina Dr. Grigoris Chrysos, who always had an answer for every question, concerning the technical aspect of my thesis.

Moreover, I want to express my appreciation to Pavlos Malakonakis for his helpful advice and to Dr. Euripides Sotiriades for his guidance.

Additionally, I want to express my thanks to S. Apostolakis, A. Gardelakos, A. Sotiropoulos, K. Kalaitzis, E. Kousanakis, G. Tampakakis, K. Kuriakides, P. Toupas, G. Pitsis and L. Charisis for exchanging views over many problems I encountered during my thesis.

Also, I want to especially thank V.Tsamou for her encouragement.

Last but not least, I want to express my highest gratitude and appreciation to my parents for their endless support.

Contents

List of Figures

List of Tables

List of Abbreviations

AE	**A**pplication **E**ngine
AEEM	**A**pplication **E**ngine **E**xecution **M**ask
AEH	**A**pplication **E**ngine **H**ub
BRAM	**B**lock **R**andom **A**ccess **M**emory
BW	**B**and**W**idth
CAE	**C**ustom **A**pplication **E**ngine
CLB	**C**onfigurable **L**ogic **B**lock
CPI	**C**ycles **P**er **I**nstruction
CRC	**C**yclic **R**edundancy **C**heck
DDR	**D**ouble **D**ata **R**ate
DIMM	**D**ual **I**n-line **M**emory **M**odule
ECC	**E**rror-**C**orrecting **C**ode
FIFO	**F**irst **I**n **F**irst **O**ut
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FSM	**F**inite **S**tate **M**achine
GE	**G**olomb **E**ngine
HDL	**H**ardware **D**escription **L**anguage
ISE	**I**ntegrated **S**ynthesis **E**nvironment
LUT	**L**ook**U**p **T**able
MC	**M**emory **C**ontroller
MPSoC	**M**ulti-**P**rocessor **S**ystem-**o**n-**C**hip
NP	**N**on-deterministic **P**olynomial
OGR	**O**ptimal **G**olomb **R**uler
SoC	**S**ystem **o**n **C**hip
TID	**T**ransaction **ID**entification

Dedicated to my parents

Chapter 1

Introduction

In Computer Science there are many problems with very high computational complexity and, despite the increased pace of technological development, they have proved to be a formidable obstacle for the conventional CPUs. In some cases, these problems can be separated into a number of parallel tasks. In such a situation, scientists often deploy some version of a distributed system, like a computer cluster or an internet based distributed platform, or they take advantage of reconfigurable processors that can be optimized in solving a particular problem, as well as exploit the parallelized nature of it.

A direction in utilizing this technology has appeared over the past years, by combining multiple FPGAs (Field Programmable Gate Array) on the same platform, in order to increase the overall computational parallelism [29]. Therefore, it is of great importance to engineer a design that makes optimum use of the platform's resources for each applicable problem, as well as achieve the best parallelization we can.

1.1 Motivation

A Golomb ruler is a set of distinct positive integers (marks) such that the differences, computed over all different pairs, are distinct. A Golomb ruler is optimal if there is no shorter Golomb ruler with the same number of marks. Finding the optimal Golomb ruler (OGR) for a specified number of marks is computationally very challenging, probably an NP-hard problem, as well as the construction of Golomb Rulers since it is provably shown to be NP-hard [21]. However, finding and proving a Golomb ruler to be optimal is a problem that can be parallelized, when we use the appropriate algorithm. Therefore there is a great incentive to design an FPGA based solution.

Convey HC-2ex is a heterogeneous computing platform that combines a general purpose Intel processor with a reconfigurable coprocessor based on FPGA technology, along with 64GB of shared memory. The coprocessor includes four FPGAs, each one connected to two of the other FPGAs, forming a ring.

Although we cannot capitalize on the high-end memory interface of the Convey HC-2ex platform, since the algorithm we intend to implement is not memory-intensive, the synergistic quad-FPGA setup gives us the opportunity to invest in the implementation of a Producer-Consumer model, therefore exploiting the extensive logic resources of the platform.

Furthermore, based on previous designs, that were developed on Technical University of Crete, we have the opportunity to explore new ideas and assess the effectiveness of our approach.

1.2 Scientific Contribution

The contribution of this thesis is mainly the redesign of the Golomb Engine, a project that is developed in Technical University of Crete with the purpose of finding and proving Optimal Golomb Rulers, and its relocation to the Convey HC-2ex platform.

While in this process, we converted the datapath from single cycle to pipeline, and reviewed in both cases the results, mostly from a spatial point of view. This design is independent of the FPGA we are employing, and we can view this as a black box that can be effortlessly transferred to a different FPGA, provided a few minor changes.

Moreover, we had to utilize a big scale multi-FPGA system, specifically an interconnected system of four FPGAs, that provides direct communication between the FPGAs, in addition with a shared memory, that we made use of, so as to share the necessary information. It should be emphasized that we had a different design in every FPGA, and although we encountered serious problems with the platform, mainly with the simulation tools we were provided, we managed to successfully synchronize all the FPGAs, and to get satisfactory results considering the system's limitations, as well as the designing tool's restrictions.

In the final stage, we tested two top of the shelf FPGAs, and we compared them with the Convey HC-2ex platform, with the results being in favor of the new technology, as it was expected, and with the new designing tool, VIVADO, playing a considerable role.

In order to do the aforementioned, we had to use VHDL, Verilog, Assembly and C, as our programming languages, and to get involved with ISE® and VIVADO®, two complex designing suites provided by Xilinx, and with ModelSim®, a multi-language HDL simulation environment.

1.3 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2 we give the necessary theoretical background on Golomb Rulers and on the algorithm we are implementing in order to find and prove these rulers. Moreover we present numerous applications in various scientific fields, that are based on Golomb Rulers' properties. Chapter 3 presents the related work on Golomb Rulers that has been developed on Technical University of Crete, as well as an overview of the Convey HC-2ex platform, and its interfaces we are making use of. In Chapter 4 we make a brief description of our initial architecture and the reasons that led to the final version. Then, we give an extensive analysis of each module of our final design and the custom protocols we developed, that were necessary for migrating to Convey HC-2ex platform. Chapter 5 expounds the performance results, and the relocation to two contemporary FPGAs for

comparison purposes. Finally, in Chapter 6 we draw our conclusions from the overall work presented in this thesis and we propose some improvements that may be realized in future work.

Chapter 2

Theoretical Background

2.1 Golomb Ruler

In mathematics a Golomb ruler, named after Solomon W.Golomb, is a set of marks at integer positions, where every set of two marks measures a distance that no other different set of two marks measures. For a ruler with n marks there are

$$0 + 1 + 2 + 3 + ... + n - 1 = \sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

differences between them. The number of the marks is the **order** of the Golomb ruler and the difference between the first and the last mark is the **length** of the ruler. If a Golomb ruler measures all distances up to its length then it is called a **perfect** Golomb ruler. If there is no Golomb ruler of the same order with shorter length, then the ruler is **optimal** (OGR).

While looking into a Golomb ruler, it is very helpful to examine its difference triangle. A difference triangle is a data structure that contains all the distances measured by the ruler. The distances are arranged between marks of the ruler in the form of a triangular matrix. Each element $d_{i,m}$ in each row m of the matrix contains the distances formed by the $|a_{i+m+1} - a_i|$ marks of the ruler. In figure 2.1 we present the difference triangle for the OGR(4) [3, 6, 22, 34].

OGR (4)	0	1	4	6
		1	3	2
			4	5
				6

FIGURE 2.1: Difference Triangle for the OGR(4)

Every Golomb ruler has a mirror image (fig.2.2), which is the reversed version of the ruler. We can create the mirror image by placing the first mark of the ruler as last at the n position, and then by placing the rest of the marks, that are at the x position, at the n-x position . The ruler and its mirror image measure the same distances.

OGR (4)	0 1 4 6	mirror	0 2 5 6

FIGURE 2.2: OGR(4) and its mirror image

Also every part of a Golomb ruler is likewise a Golomb ruler with fewer marks and it is called a **stub**.

Optimal Golomb Rulers		
Order	**Length**	**Marks**
1	0	0
2	1	0 1
3	3	0 1 3
4	6	0 1 4 6
5	11	0 1 4 9 11 0 2 7 8 11
6	17	0 1 4 10 12 17 0 1 4 10 15 17 0 1 8 11 13 17 0 1 8 12 14 17
7	25	0 1 4 10 18 23 25 0 1 7 11 20 23 25 0 1 11 16 19 23 25 0 2 3 10 16 21 25 0 2 7 13 21 22 25
8	34	0 1 4 9 15 22 32 34
9	44	0 1 5 12 25 27 35 41 44
10	55	0 1 6 10 23 26 34 41 53 55
11	72	0 1 4 13 28 33 47 54 64 70 72 0 1 9 19 24 31 52 56 58 69 72
12	85	0 2 6 24 29 40 43 55 68 75 76 85
13	106	0 2 5 25 37 43 59 70 85 89 98 99 106
14	127	0 4 6 20 35 52 59 77 78 86 89 99 122 127
15	151	0 4 20 30 57 59 62 76 100 111 123 136 144 145 151
16	177	0 1 4 11 26 32 56 68 76 115 117 134 150 163 168 177
17	199	0 5 7 17 52 56 67 80 81 100 122 138 159 165 168 191 199
18	216	0 2 10 22 53 56 82 83 89 98 130 148 153 167 188 192 205 216
19	246	0 1 6 25 32 72 100 108 120 130 153 169 187 190 204 231 233 242 246
20	283	0 1 8 11 68 77 94 116 121 156 158 179 194 208 212 228 240 253 259 283
21	333	0 2 24 56 77 82 83 95 129 144 179 186 195 255 265 285 293 296 310 329 333
22	356	0 1 9 14 43 70 106 122 124 128 159 179 204 223 253 263 270 291 330 341 353 356
23	372	0 3 7 17 61 66 91 99 114 159 171 199 200 226 235 246 277 316 329 348 350 366 372
24	425	0 9 33 37 38 97 122 129 140 142 152 191 205 208 252 278 286 326 332 353 368 384 403 425
25	480	0 12 29 39 72 91 146 157 160 161 166 191 207 214 258 290 316 354 372 394 396 431 459 467 480
26	492	0 1 33 83 104 110 124 163 185 200 203 249 251 258 314 318 343 356 386 430 440 456 464 475 487 492
27	553	0 3 15 41 66 95 97 106 142 152 220 221 225 242 295 330 338 354 382 388 402 415 486 504 523 546 553

TABLE 2.1: All known OGRs as of October 2019

The derivation of optimal Golomb rulers is a very challenging computationally problem, as there is no known closed-form solution, and it is known to

be an NP-hard problem [28, 21], with an exponential growth in computational time of the exhaustive search, as the ruler's order increases. To make this clear, finding a Golomb ruler for a specific order to be optimal, can be proved only when no other ruler found is smaller than the OGR. However the bounds of the algorithms used are growing exponentially with respect to the solution size. It is worth noting that there are construction techniques for rulers that appear to be optimal and are guaranteed to be near-optimal. There are lists for all the best known Golomb rulers, for a considerable number of marks. For example the best known Golomb rulers with 3999 marks has a length of 15,875,644 [25].

There are many algorithms for constructing optimal Golomb rulers [34], like the Scientific American algorithm, the Token Passing algorithm, the Shift algorithm, the FLEGE (Feiri-Levet Enhanced GARSP Engine) algorithm and others. In this thesis the algorithm that we are using in our architecture is the Shift Algorithm.

Because of the very long computational time, the rulers with 24 to 27 marks, have been proven by distributed.net [13]. Distributed.net is an online, general purpose, distributed computing project. For the OGR-project they use a brute force search, but they do not give too much information on the algorithms they use in order to reduce the search space. Their current project is finding the OGR(28).

It is also worth noting that there is an increase in nature-inspired based, multi-objective, optimization algorithms for solving multi-objective, engineering design problems. FA (Firefly Algorithm), MOBA (Multi-Objective Bat Algorithm), and its extended form PHMOBA (Parallel Hybrid MOBA) and FPA (Flower Pollination Algorithm) are some of these nature-inspired algorithms that are used in solving optimal Golomb Rulers problems, mainly for channel allocation, bandwidth usage improvements and reduction in four-wave mixing (FWM) crosstalk in optical wavelength division multiplexing (WDM) systems [5, 15, 35].

2.2 Applications

Golomb rulers have a wide variety of applications and they find actual uses in various scientific fields [8].

2.2.1 Information Theory

In Information and Coding Theory Golomb rulers are used for error detection and correction [26]. The rulers are used to generate canonical self-orthogonal codes (CSOC). These codes use a set of difference triangles such that all the elements are distinct. This property gives to the CSOC the ability to correct independent errors and recover from decoding errors, considering there is an error-free period. The performance of these codes is very high.

Moreover in the research for efficiently obtaining sparse sensor arrays, Golomb rulers are very useful, since the placing of distinct difference bases allows for increased aperture, by reducing the number of redundant spacings in the array [18].

For the same reasoning, in the design of spatially or temporally distributed acoustic or underwater acoustic systems, placing the structural elements of the system, based on the idea of the unique differences between the marks of a Golomb ruler, improves its quality and performance, by avoiding the interference of signal components of the same spatial frequency [30, 31].

2.2.2 Telecommunications

In frequency usage, when there is a band of consecutive channels, there is interference created from third and fifth order intermodulation products. If the frequency difference between any pair of the operating channels is distinct from any other pair, then the intermodulation products, and especially the ones of third order, do not fall on other operating channels [4].

Furthermore, in a time-hopping multiple-access (THMA) communication system, a message packet of length T is encoded into n subpackets. The n subpackets are then transmitted in some n time slots. The first packet will be transmitted at the ith time slot, the second packet at the ith+t_1th time slot, and the nth packet at the ith+t_{n-1}th time slot. For system performance analysis purposes the n(n-1)/2 positive differences of every pair of time slots, needs to be distinct. Such a time-hopping pattern is said to be optimum if it is of minimum length. The optimal patterns coincide with Golomb rulers [17].

2.2.3 Radio Astronomy

Similar to subsection 2.2.1, in the field of Radio Astronomy, in order to observe extragalactic radio sources, while placing antenna arrays, we want to achieve the desired sensitivity and resolution. For a given number of elements there is an optimal way to place them. By using Golomb rulers, this is resolved, since it minimizes the number of redundant spacings present in the array [1, 7].

2.2.4 X-ray Crystallography

X-ray crystallography is a technique used to determine the atomic and molecular structure of proteins and biological macromolecules. Its aim is to obtain a three dimensional molecular image, by exposing the crystal to an x-ray beam from different directions, and then measure and process the resulting diffraction patterns. In order to eliminate ambiguities that occur when two different crystals produce the same diffraction patterns, we can take advantage of the Golomb rulers' properties. That is, with the exception of a single pair of six mark rulers, any Golomb ruler of the same length will have a different set of differences measured [8, 27].

2.2.5 Computer Science

2.2.5.1 Wireless Localization

There are many studies and publications concerning the problem of wireless localization, but still indoors wireless localization systems are unreliable and

inaccurate. The main reasons for that are the loss of line of sight that is much needed for angle-of-arrival based algorithms, and the high cost of a robust and accurate multi-antenna system, with high computational capabilities, since the typical requirements for an indoor system would be low in cost and power. Furthermore ranged-based algorithms' quality is degraded by interference and that leads to complex scheduling in the collection of the ranging information with high communication cost. By using Golomb rulers to optimize phase-difference of arrival (PDoA)-based ranging techniques, we manage to obtain a large input set, from a significantly smaller number of actual measurements, with the additional benefit of low energy consumption and latency since we need fewer number of samples collected [23].

2.2.5.2 Closed Hashing

In a hash table when we try to insert a new item, we calculate its hash value and insert it at the respective position. If there is an item already at this position, we have a collision. A solution is to try and store the collided item at a different location inside the hash table (closed hashing). We may move up the hash table one position at a time (linear probing) or a number of positions depending on the number of times that we have moved so far (quadratic probing), until there is no collision and we have a successful insertion. However if we choose to move in a different way, jumping a number of positions that is distinct than any other number that we have jumped so far (Golomb probing), then we minimize the overlapping of collision chains between two entries with a different hash value. That results in significantly less collisions than linear (58%) and quadratic probing (26%) [20].

2.2.5.3 Recovery Scheme in Distributed Systems

In a distributed computing system, fault tolerance is of great importance. Besides even distribution of the load, while all computers are up and running, we also expect the load to remain uniformly distributed even when the worst combination of computers break down. The problem of finding the optimal recovery scheme for a system with n computers corresponds to the mathematical problem of finding the longest sequence of positive integers, such that the sum is smaller than n and the sums of all subsequences are unique, which in essence describes the Golomb rulers [16].

2.3 Shift Algorithm

The Shift Algorithm was designed by professor Apostolos Dollas, William T. Rankin and David McCracken [3] and it is an enhancement of the Scientific American algorithm. The major advantage of this algorithm is that when a new mark is placed on the ruler, only the new distances it introduces have to be calculated. So instead of having to calculate N(N-1)/2 differences, it calculates only the N new differences created by the new mark. Moreover it allows for an efficient way to check if a candidate ruler is a Golomb one.

The Shift algorithm utilizes two bit vectors, that represent the positions of the placed marks on the ruler, and the distances measured by these marks, respectively. The first bit vector, **LIST**, holds the marks of the ruler, and the second bit vector, **DIST**, captures all the differences between the marks. The 2.3 figure illustrates these 2 bit vectors. Note that the LIST vector holds the marks from right to left, starting from number 0, while the DIST vector holds the distances from left to right, starting also from zero distance. In this example our ruler has marks on 0,1,3 and 7, and the differences developed from these marks are 0,1,2,3,4,6 and 7.

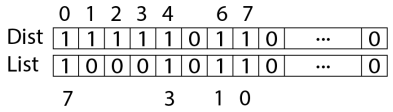

FIGURE 2.3: Illustration for DIST and LIST vectors for 4 marks

The LIST vector is initialized to 10x00, while the DIST vector is initialized to 0x00. As long as the marks at LIST vector do not intersect with the marks at DIST vector, we create the new LIST vector by adding an '1' at the first position of LIST and the new DIST vector using an OR function between LIST and DIST. In order to check if the marks intersect, we bit-wise AND-ing the 2 vectors together and then OR-ing all the AND results. If the marks between LIST and DIST intersect (meaning the OR returned 1) then we **shift** the LIST vector to the right, thus the name of the algorithm. Each time we successfully add a new mark to the ruler, therefore creating the N+1 mark ruler, we also push the, shifted to the right by one, N mark ruler onto a stack, and the algorithm tries to add another mark to the N+1 ruler. If we cannot add a new mark, then we pop from the stack the shifted N mark ruler, and we try to add another mark.

Figure 2.4 shows a flowchart of the algorithm. The array MAX[D] contains the maximum values we allow for each potential mark of the ruler (Maximum Position Reduction), while MIN[D] bounds the minimum location at which a mark may be placed (Minimum Ruler Preclusion). Both arrays will be discussed further at section 2.4.

Figure 2.5 demonstrates the first steps of the Shift algorithm.

2.4 Search Space Reductions

In order to improve the algorithm's efficiency we can apply a number of search space reductions.

- **First Mark Preclusion:** This technique relies on the fact that if a ruler has its second mark at the second position (measuring a distance of 1), there cannot be a mark at the second position of its mirror image ruler. So

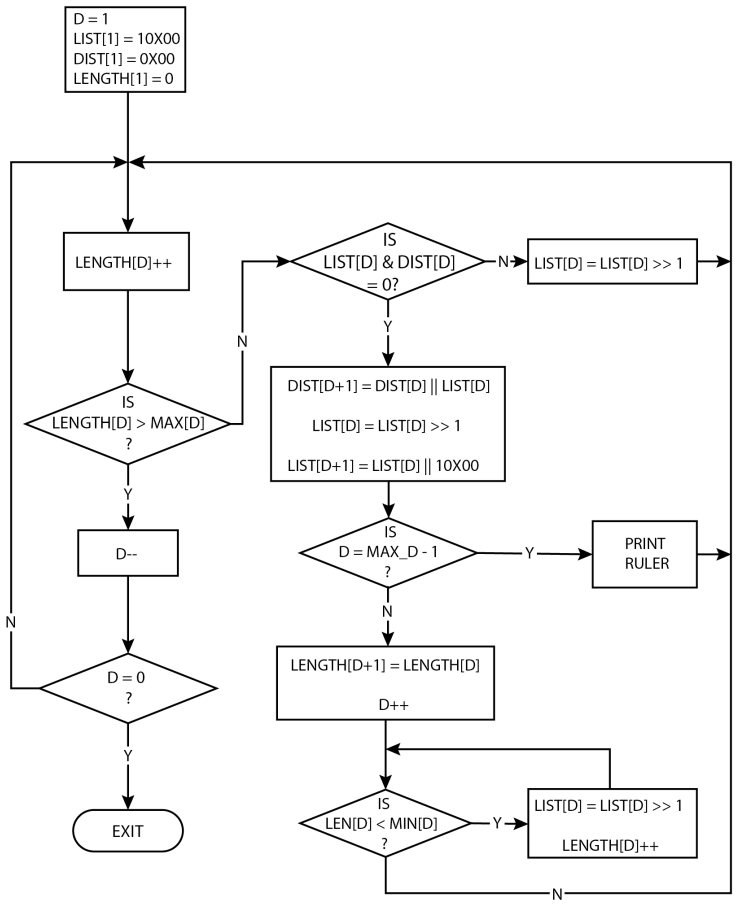

FIGURE 2.4: Flowchart for the Shift algorithm

we can place the second mark at the third position (measuring a distance of 2), instead of the second position. In that way we may not find the ruler if it has its second mark at the second position, but it is guaranteed that we will find its mirror image. With this technique we can decrease by 10% the search space, while still maintaining an exhaustive search[12].

- **Midpoint Reduction:** This technique relies on the fact that a ruler measures the same distances with its mirror image. So we can restrict the position of the middle mark of the ruler to the first half of its geometric center, since either the ruler or the mirror image must have the middle mark positioned before the geometric center of the ruler. This restriction reduces the search space by almost half. This technique cannot be used in combination with First Mark Preclusion technique.

- **Maximum Position Reduction:** This method is based on the fact that for every mark on the ruler, the available space for the remaining number of marks cannot be smaller than the space required from the

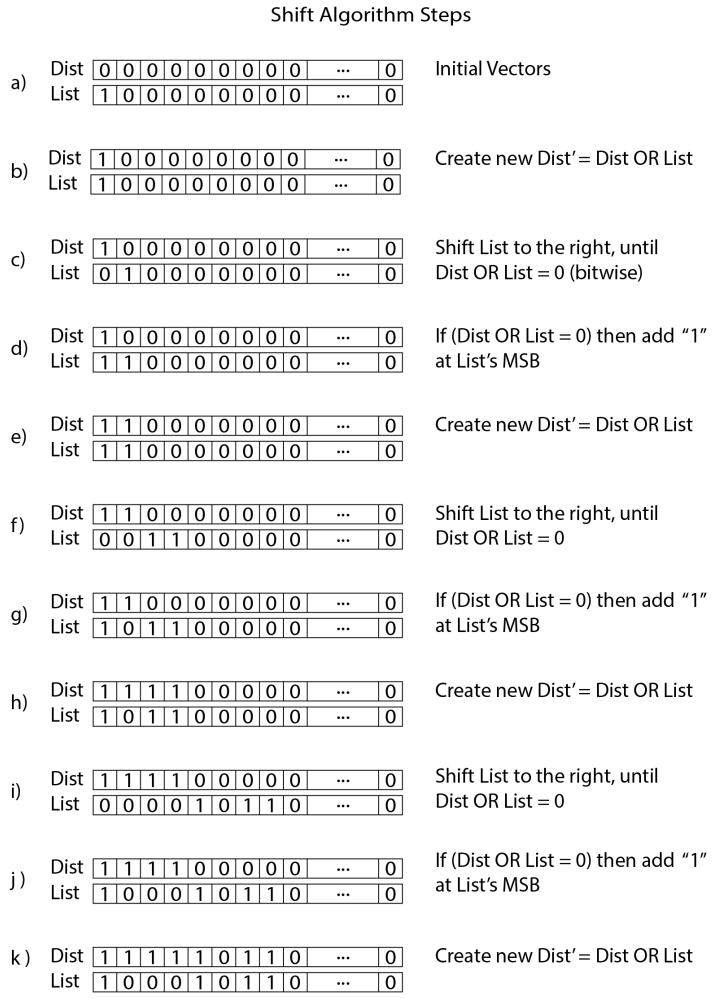

FIGURE 2.5: Demonstration of the Shift algorithm

equivalent OGR of the same number of marks. Therefore whenever we place a new mark, its maximum position has to be the length of the OGR we are looking for, minus the OGR of the number of the remaining marks, minus one. Furthermore, if we combine this technique with the Midpoint Reduction technique, we can reduce the upper bounds for each mark even more, by keeping a table of minimum ruler lengths as a function of their length and distances measured[3].

- **Minimum Ruler Preclusion:** The last technique is based on the fact that a ruler with a number of marks and a specific length measures certain distances. By keeping a table with this information, we reduce the value range of the possible positions of a new mark, therefore reducing the search space.

In our design we make use of Midpoint Reduction, Maximum Position Reduction and Minimum Ruler Preclusion.

Chapter 3

Related Work and Convey Platform

The Golomb Engine project is developed at Technical University of Crete under the guidance of professor Apostolos Dollas and aims at solving the OGR-n problem. It implements a parallel version of the Shift algorithm and it is based on FPGA devices.

3.1 Previous Designs

3.1.1 GE1

The first Golomb Engine was designed by prof. Apostolos Dollas, Euripides Sotiriades and Apostolos Emmanouelides in 1998 [2]. It consisted of twenty Xilinx 5000 series, 19 required for the datapath and 1 for the control path, and it could solve the 20 marks Golomb Ruler. Although it had a low clock frequency (11.96 MHz), it ran roughly 30 times faster than a high-end workstation. The control issued 3 instructions: SHIFT, PUSH and POP. Depending on the outcome of an OR comparison between DIST and LIST vectors, it shifted the list one position to the right, until there was no collision, then pushed the two vectors in two stacks, shifted one more bit to the right, and continued executing successive shifts. The reason of the extra shift before the push operation was to avoid an infinite loop of push and pop operations. In case of the list vector shifting too "far" to the right, so that it was impossible to produce an optimal ruler, then the previous list and dist vectors were popped from the stacks.

3.1.2 GE2

The second Golomb Engine [14] implemented a producer-consumer model. It was designed by Euripides Sotiriades and prof. Apostolos Dollas and it could solve the 24 marks Golomb Ruler. It produced 8-marks stubs on a personal computer, and each stub was processed by an FPGA. They compared their results with a 333MHz AMD K6. It had a 48 MHz clock, and, although for short length stubs the software was faster, the speedup that was calculated from the mean times of the performance of each range of stubs according to the percentage of the corresponding stubs and the total number of stubs, was MT_{GE2} / MT_{SW} = 8.25 .

3.1.3 GE3

The third Golomb Engine was designed by Pavlos Malakonakis, under the guidance of prof. Apostolos Dollas and Euripides Sotiriades [24, 38]. It introduced the idea of multiple shifts per cycle, which was a significant improvement to the existing datapath. Moreover both producer and consumers were designed in the same FPGA.

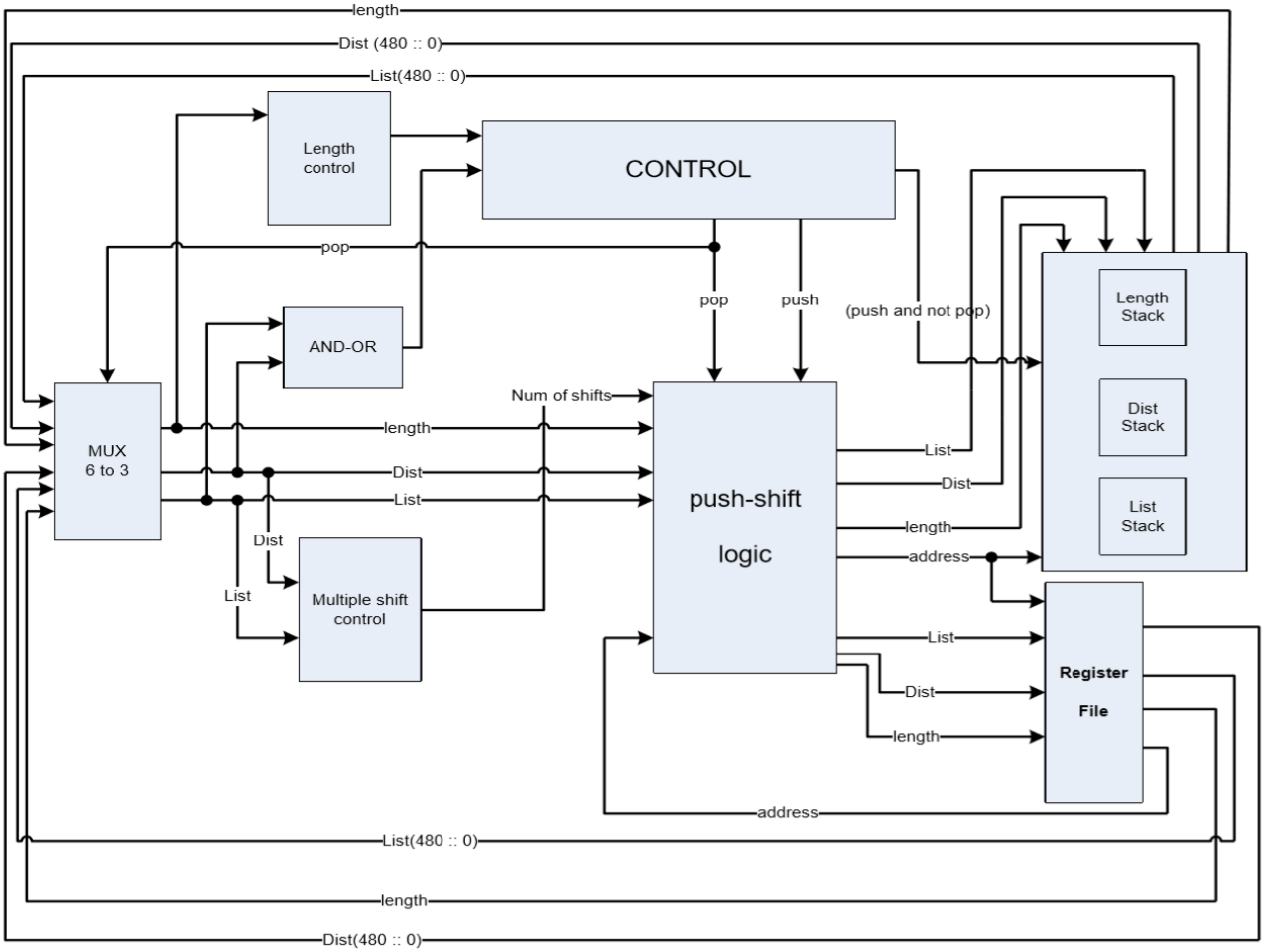

FIGURE 3.1: Datapath of Golomb Engine 3 with multiple shifts [38]

GE3, like GE2, executed 3 operations: SHIFT, PUSH and POP with 1 cycle per instruction. Considering that for OGR(25) 98% of the operations are SHIFT, and 60% of them are a shift of 16 or more positions, they created a multiple shift control unit, that could determine how many positions the list vector should shift, ranging from 1 to 16, in case of a SHIFT operation. This new unit tripled the utilization of resources on the FPGA, but it gave a speedup of 12 compared to the previous version of the Golomb Engine.

For the producer/consumer model a modified GE was used, that produced stubs, with shorter list and dist vectors than the OGR that we are looking for, which were written into a FIFO and then they were fed into the consumers.

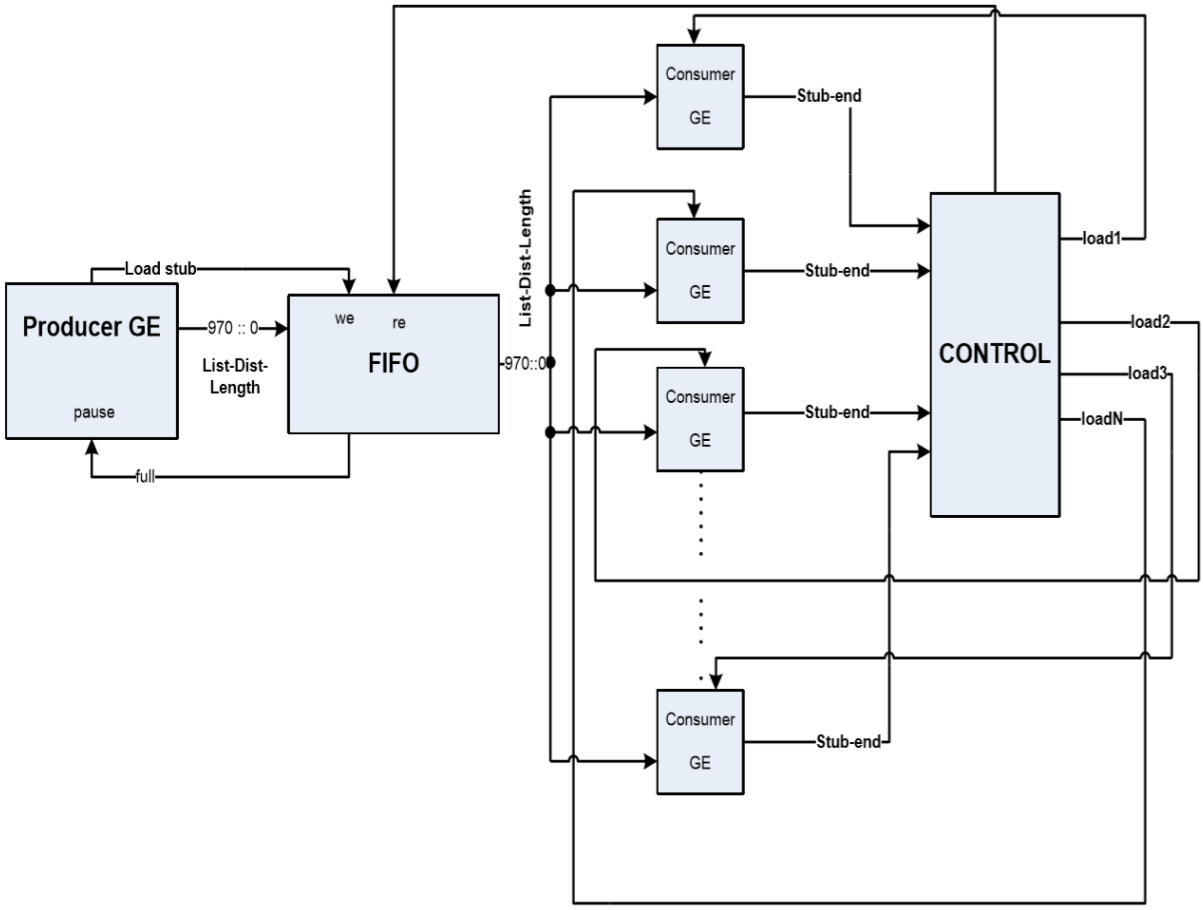

FIGURE 3.2: General Architecture of GE3 [38]

GE3 was modified to solve the OGR(25) on a Spartan3, XC3S1000 model, but due to the bigger utilization, there was enough room only for the producer and a single consumer, and both supported 1 to 3 position shifts instead of 16. This design had a 40 MHz clock. There was also another design on the XUP platform, on a XC2VP30 FPGA, that could accommodate either 2 consumers without multiple shifts, or 1 consumer with 1-8 position shifts. The second case had a 63 MHz clock. Furthermore, there was a design on a Virtex4, XC4V60 FPGA, with 2 consumers supporting 1-11 position shifts, and the producer creating stubs with 15 marks. This design had a 87 MHz clock. Finally, the last design was on a Virtex5, XC5V330XL FPGA, that could, theoretically, accommodate 18 consumers with 1-16 position shifts, with a 120 MHz clock. Unfortunately, due to memory limitations, this design could not be simulated with the tools Xilinx was offering at the time [38].

3.2 Convey HC-2ex

3.2.1 Overview

The Convey Hybrid Core is a heterogeneous computing platform that integrates two types of processor architecture in one system. It combines a host x86-64 Intel Xeon E5-2642 6-core processor and a reconfigurable Convey coprocessor based on FPGA technology. The coprocessor is an attached processor [1] that implements multiple instruction sets, referred to as personalities, including both Convey-defined and user-defined personalities. It utilizes 4 Virtex-6 LX760 FPGAs, called Application Engines (AEs), and a total of 64 GB DDR2 shared memory capable of providing up to 80 GB/s of memory bandwidth when all four AEs are deployed. It includes its own high bandwidth memory subsystem that is incorporated into the Intel global memory space, creating the Hybrid-Core Globally Shared Memory (HCGSM) [11].

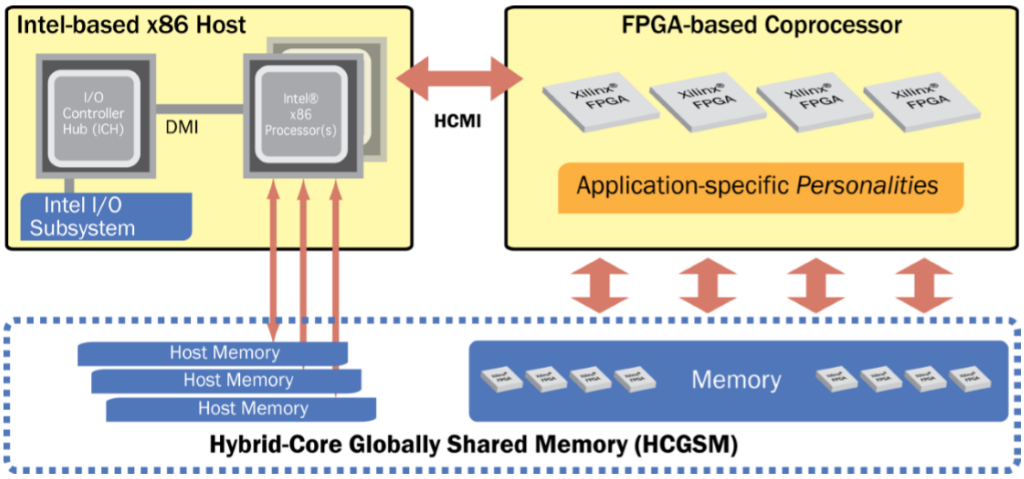

FIGURE 3.3: Convey Hybrid Core System Diagram [11]

3.2.2 Coprocessor

The Convey coprocessor has three major sets of components, as shown in fig 3.4: The Application Engine Hub (AEH), the Memory Controllers (MCs), and the Application Engines (AEs).

The AEH is the central hub for the coprocessor. It implements the interface to the host processor and to the Intel I/O chipset, fetches and decodes instructions, and executes scalar instructions. It processes coherence and data requests from the host processor, routing requests for addresses in coprocessor memory to the MCs.

The coprocessor has 8 Memory Controllers that support a total of 16 DDR2 memory channels, providing an aggregate of over 80GB/sec of bandwidth to ECC protected memory. The MCs translate virtual to physical addresses on

[1] The Convey coprocessor must be attached to a process before it can be used.

behalf of the AEs. They support standard DIMMs as well as Convey designed Scatter-Gather DIMMs. The Scatter-Gather DIMMs are optimized for transfers of 8-byte bursts, and provide near peak bandwidth for non-sequential 8-byte accesses. The coprocessor therefore not only has a much higher peak bandwidth than is available to commodity processors, but also delivers a much higher percentage of that peak for non-sequential accesses .Together the AEH and the MC's implement features that are present in all personalities, ensuring that important features such as memory protection, access to coprocessor memory, and communication with the host processor are always available.

The AEs are connected to the AEH by a command bus that transfers opcodes and scalar operands, and to the memory controllers via a network of point-to-point links that provide very high sustained bandwidth. Each AE instruction is passed to all four AEs and it is processed accordingly to the implemented personality.

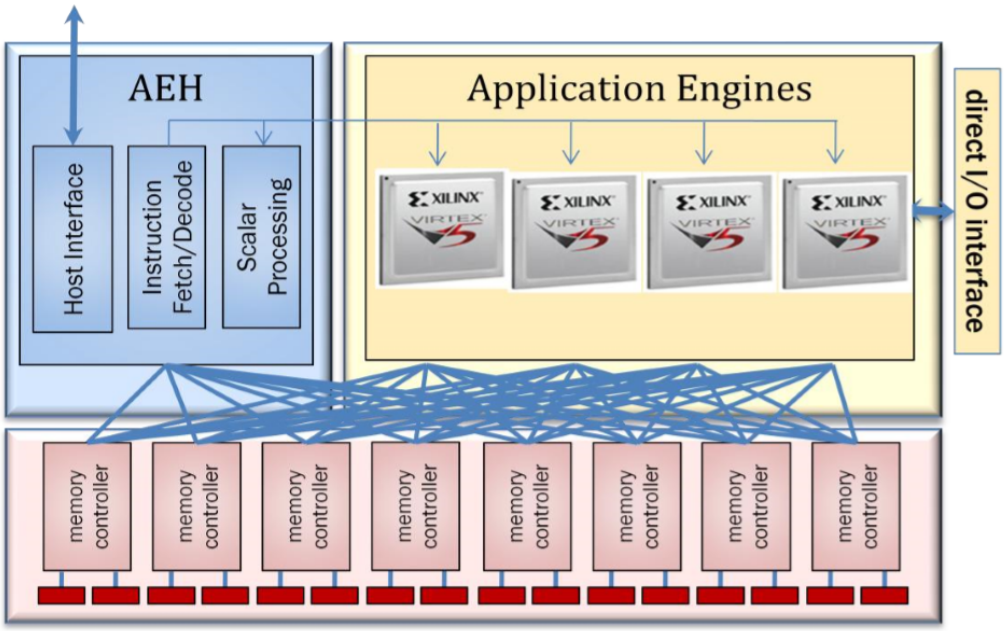

FIGURE 3.4: Coprocessor Diagram [11]

3.2.3 Personalities

A personality defines the set of instructions supported by the coprocessor and their behavior. A system may have multiple personalities installed and can switch between them dynamically to execute different types of code, but only one personality is active on a coprocessor at any one time. Each installed personality includes the loadable bit files that implement a coprocessor instruction set, a list of the instructions supported by that personality, and an ID used by the application to load the correct image at runtime.

The instruction sets can be of two types. The scalar instruction set includes all basic operations that implement basic scalar and control flow operations required to implement interfaces and manage the operation of the AEs and they are common among all personalities. The extended instruction set is unique for every personality. It is designed for particular workloads and provides customization for different applications. Scalar instructions are executed in the AEH, while extended instructions are passed to the AEs for execution.

3.2.4 Memory Controller Interface

The Memory Controller (MC) Interface gives the AEs direct access to coprocessor memory. Each of the 4 AEs is connected to each of the 8 MCs through a 300MHz DDR interface. The MC interface inside the AEs is provided by Convey. Each of 8 MC interfaces in the AE FPGA is directly connected to a single Memory Controller, and each MC physically connects to 1/8 of the coprocessor memory. Each Memory Controller is connected to 2 DIMMs. The AE personality must decode the virtual memory address so that only requests intended for a particular MC's attached memory are sent to that MC [9].

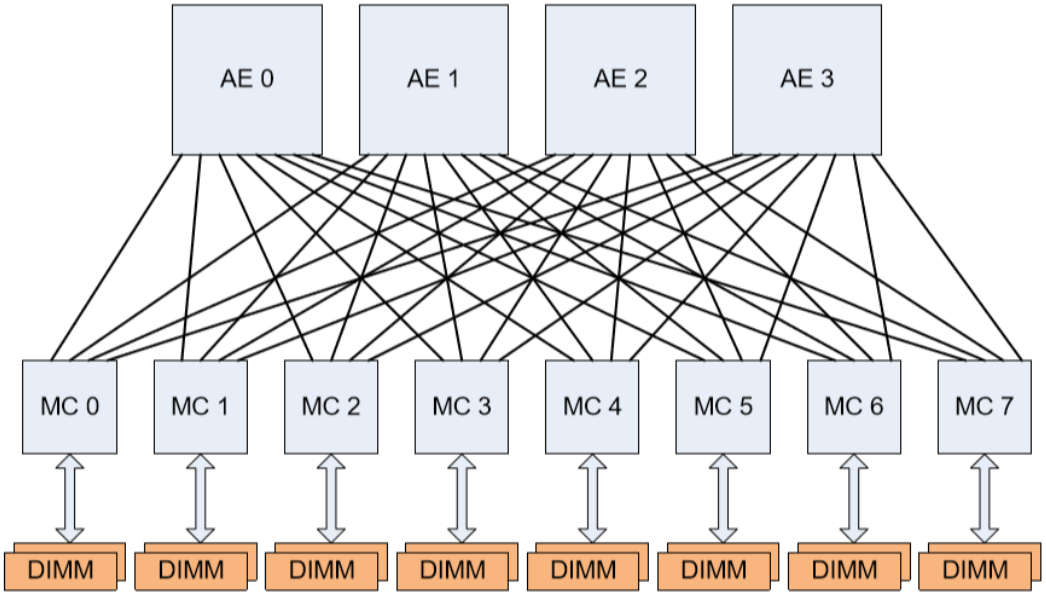

FIGURE 3.5: Coprocessor AE Memory Connections [9]

3.2.4.1 MC Interface Functionality

The 300 MHz interface is converted into two 150 MHz memory ports to/from the AE personality, in order to ease timing. Data from these two ports, the "even" port and the "odd" port, are multiplexed onto the same 300 MHz request channel in the MC interface. For write operations, the write data is stored in a first-in, first-out buffer until it is sent across the AE-MC link. No response is returned to the AE personality for write operations. For read operations, the

write data bus is used to store read return control information. This data is stored in the write data buffer until the read request is sent out. When the read request is sent to the MC, the read request data is removed from the write data buffer and stored in a read control buffer based on the transaction ID assigned to the request transaction. When the read is returned from the MC, the transaction ID (TID) is used to lookup the read control information from the read control buffer.

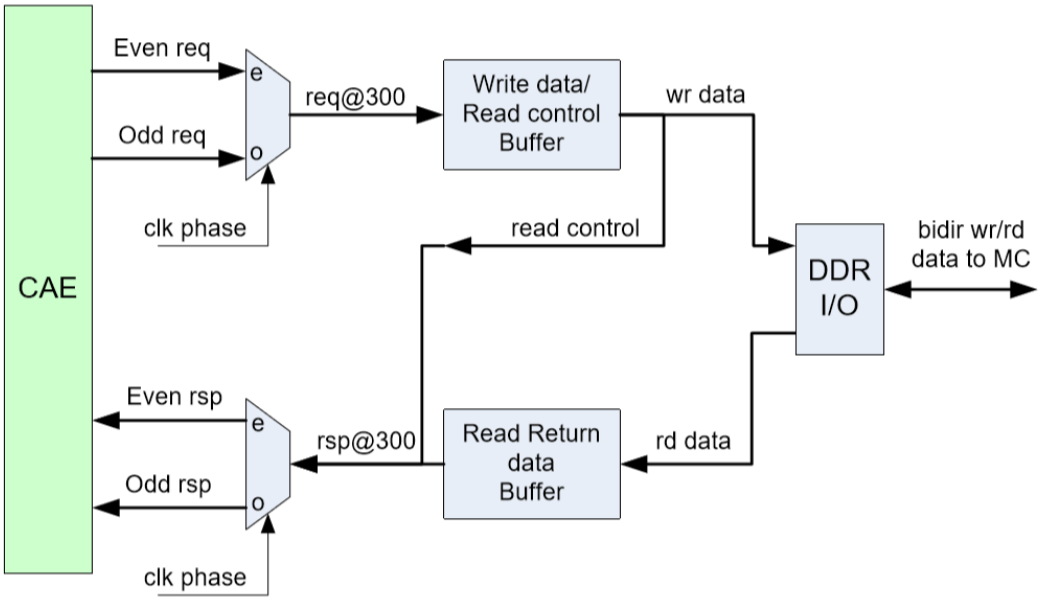

FIGURE 3.6: Functional Block Diagram of the MC Interface [9]

Also, fence instructions are sent by the AEH to enforce ordering of loads and stores to memory. Finally, stall requests are sent from the MC interface to stall read requests and write requests for two cycles to avoid overflowing buffers in the MC interface.

3.2.4.2 Signal Interface to Custom Personality

The signals between the custom AE personality and the MC interface are divided into the following categories:

- Even Request Port
- Odd Request Port
- Even Response Port
- Odd Response Port

Three operations can be sent to the MC interface: loads, stores and fences. To send a request, one of the corresponding signals (mc_req_ld_*, mc_req_st_* or mc_req_fnc_*) must be asserted. The mc_req_size_*<1:0> indicates

whether it is a byte, word, doubleword or quad-word request. The size, 48-bit address and write data/read control signals are valid on the same cycle as the load or store request.

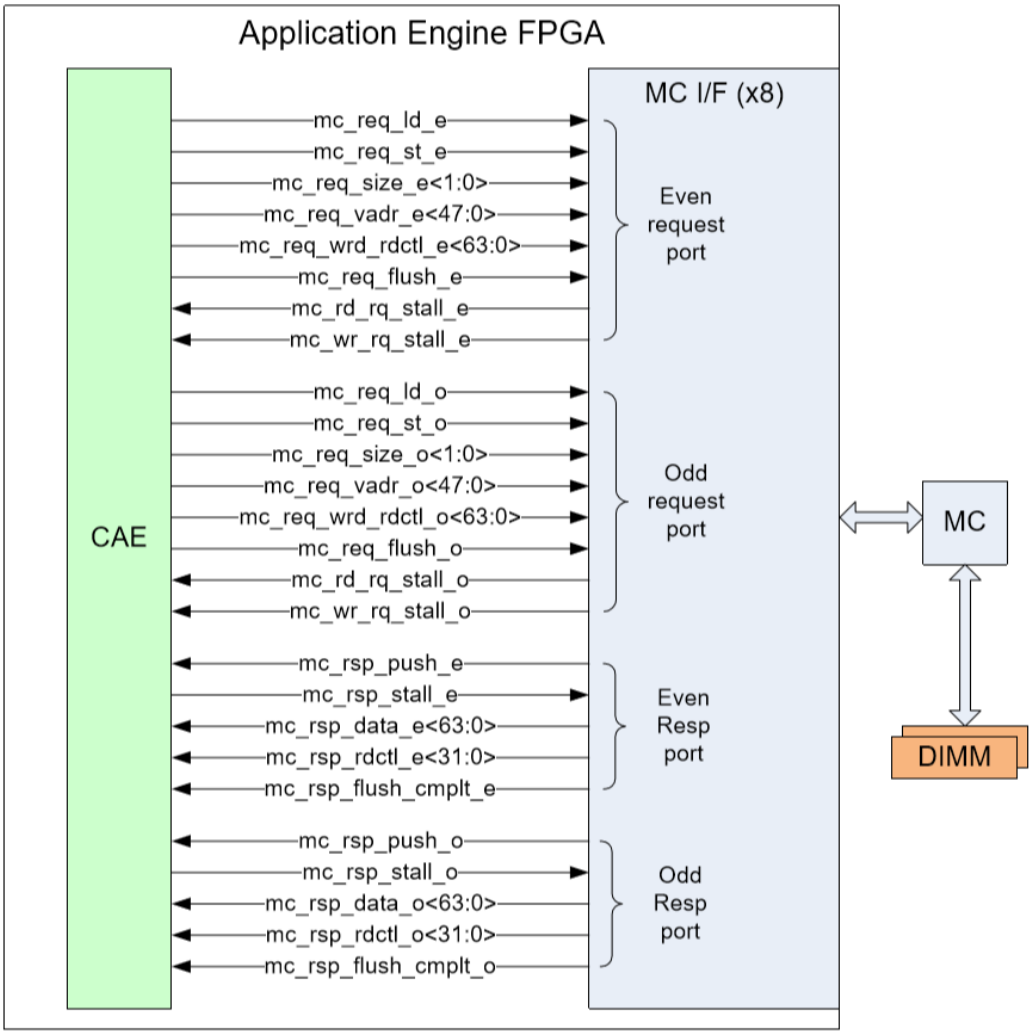

FIGURE 3.7: CAE to MC I/F Signal Interface [9]

In our thesis we are using the even response and even request ports. The signals for these ports and their definitions are shown in the figure above and in the tables 3.1 and 3.2.

3.2.5 Optional MC Interface Functionality

Optional MC Interface functionality can be instantiated to reduce complexity in the custom personality. This optional functionality is divided in three sections: response data ordering, request data ordering and memory abstraction. Instantiating any optional interface, requires some of the FPGA's resources. In our thesis we implement two of these optional interfaces: Read Order Cache and Crossbar.

Signal Name	*Type*	*Description*
mc_rsp_stall_e	output	Stall responses from MC
mc_rsp_push_e	input	MC response valid
mc_rsp_data_e<63:0>	input	Response data, aligned at least significant byte (byte 0)
mc_rsp_rdctl_e<31:0>	input	Response read control*
mc_rsp_flush_cmplt_e	input	Write flush complete

TABLE 3.1: MC Interface Signal Definitions – Even Response Port [9]

Signal Name	*Type*	*Description*
mc_req_ld_e	output	Load request
mc_req_st_e	output	Store request
mc_req_size_e<1:0>	output	Request size (0-byte, 1-word, 2-dbl, 3-quad)
mc_req_vadr_e<47:0>	output	48-bit virtual address
mc_req_wrd_rdctl_e<63:0>	output	Stores: [63:0] - Write data Loads: [63:32] - reserved [31:0] - read control*
mc_req_flush_e	output	Write flush request
mc_rd_rq_stall_e	input	Stall read requests to this MC
mc_wr_rq_stall_e	input	Stall write requests to this MC

TABLE 3.2: MC Interface Signal Definitions – Even Request Port [9]

3.2.5.1 Optional Read Order Cache MC Interface

The optional read order cache assures memory reads (loads) from the associated request port are returned to the custom personality in order. The read data returned from memory is queued and ordered before being returned to the personality. Order is only assured for reads and only between requests from the same request port. The Read Order Cache does not impact memory content. The read order cache is enabled using the MC_READ_ORDER variable.

3.2.5.2 Optional Crossbar MC Interface

The optional crossbar is used to connect each port of custom personality to every MC interface. The crossbar allows the personality to maintain an abstracted view of memory, since the address decode and request/response routing is handled by the crossbar. The crossbar is enabled with the MC_XBAR variable.

3.2.6 AE-AE Interface

The AE-to-AE interface allows data to be transferred directly from one AE to another. The AE to AE interface is an optional interface. Two types of AE–AE interfaces are available:

- A set of counter flowing rings
- Point to point connections for "next door AEs".

The AE–AE Interface is enabled by setting the AE_AE_IF variable to 1. The AE_AE_IF variable enables all AE-AE links (ring and point to point). Also, the AE-AE Interface, if enabled, utilizes some of the FPGA resources, though the company does not specify exactly what percentage of each AE's resources.

3.2.6.1 AE-AE Ring Interface

The AE–AE Ring Interface provides transparent transport for two counter flowing rings supporting data rates up to 600Mbps each. The PDK designer determines the protocol. CRC checking on the data and flow control are provided by the AE–AE Interface. Figure 3.8 shows the signal interface between the AE–AE Interface and the custom AE personality and table 3.3 defines these signals.

3.2.6.2 Next Door Point to Point Interface

The Next Door Point-to-Point Interface provides transparent transport for point to point connections between "next door AEs" supporting data rates up to 247.5 Mbps each. The HC-2ex supports two sets of point to point links. The PDK designer determines the protocol. Parity generation and checking on the data and flow control are provided by the AE – AE Interface. Figure 3.9 shows the signal interface between the AE–AE Interface and the custom AE personality and table 3.4 defines these signals.

3.2.6.3 AE – AE Transmit

To send a transaction to another AE,we have to assert *_tx_vld while driving *_tx_data. The *_tx_stall signal is provided as a backpressure mechanism to stall transactions from the custom personality. When stall is asserted, the personality must stop sending data within two cycles to avoid overflowing buffers in the AE-AE interface.

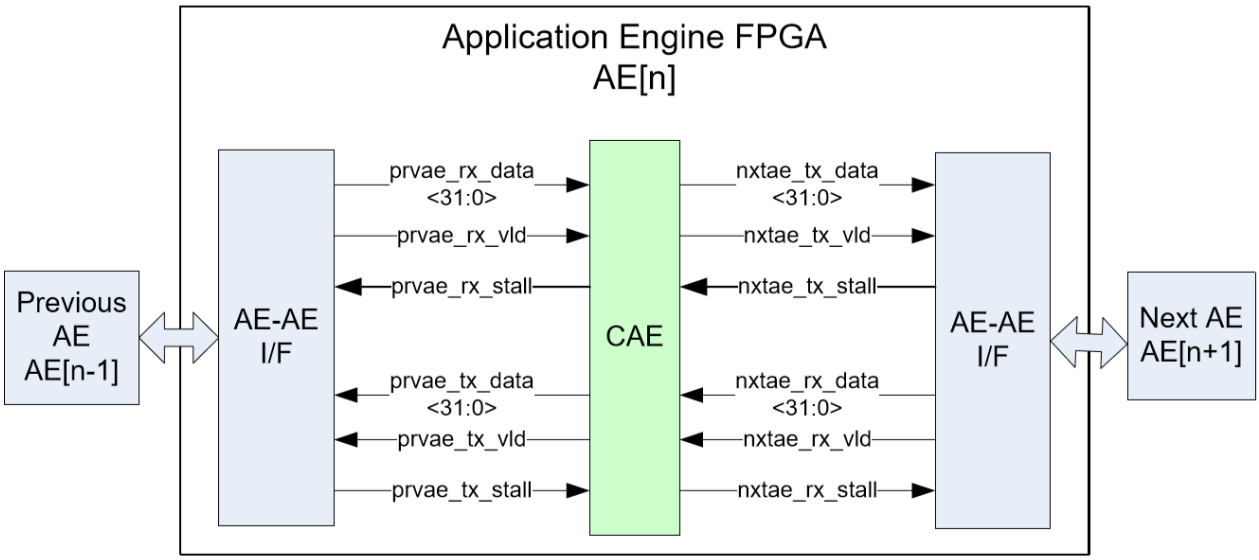

FIGURE 3.8: AE to AE Ring Interface Diagram [9]

Signal Name	*Type*	*Description*
nxtae_rx_data<31:0>	input	Next AE receive data
nxtae_rx_vld	input	Next AE receive data valid
nxtae_rx_stall	output	Next AE receive stall
nxtae_tx_data<31:0>	output	Next AE transmit data
nxtae_tx_vld	output	Next AE transmit data valid
nxtae_tx_stall	input	Next AE transmit stall
prvae_rx_data<31:0>	input	Previous AE receive data
prvae_rx_vld	input	Previous AE receive data valid
preae_rx_stall	output	Previous AE receive stall
preae_tx_data<31:0>	output	Previous AE transmit data
preae_tx_vld	output	Previous AE transmit data valid
preae_tx_stall	input	Previous AE transmit stall

TABLE 3.3: AE to AE Interface Ring Signal Definitions [9]

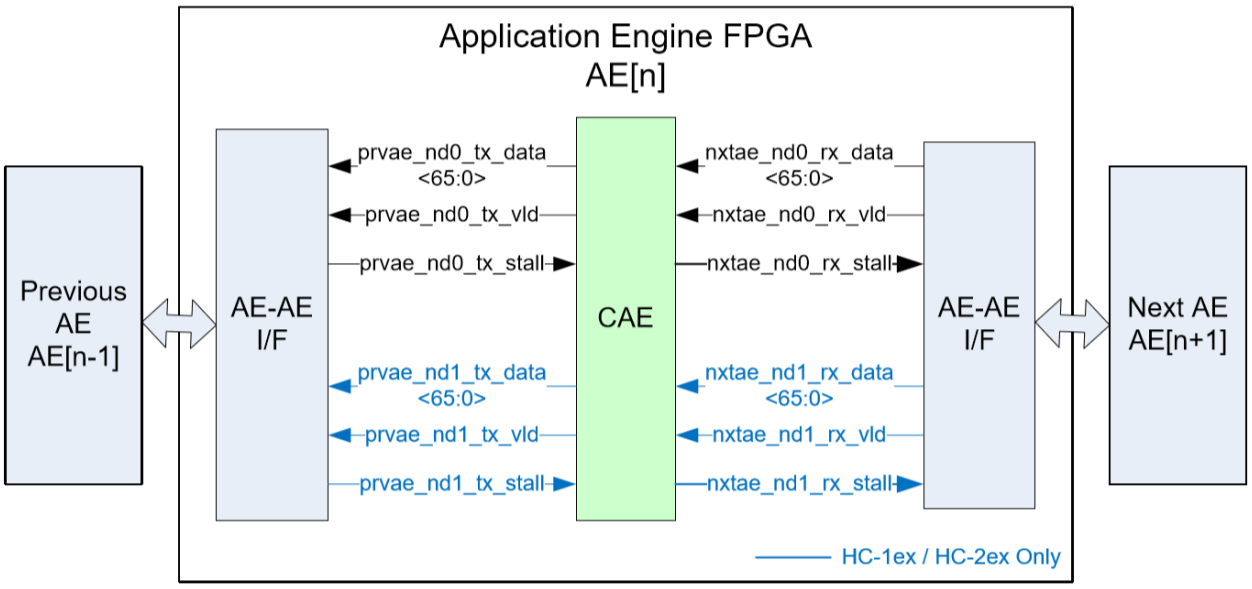

FIGURE 3.9: AE to Next Door AE Point to Point Interface Diagram [9]

Signal Name	*Type*	*Description*
prvae_nd0_tx_data<65:0>	output	Previous AE next door transmit data
prvae_nd0_tx_vld	output	Previous AE next door transmit data valid
prvae_nd0_tx_stall	input	Previous AE next door transmit stall
nxtae_nd0_rx_data<65:0>	input	Next AE next door receive data
nxtae_nd0_rx_vld	input	Next AE next door receive data valid
nxtae_nd0_rx_stall	output	Next AE next door receive stall

TABLE 3.4: AE to Next Door AE Interface Signal Definitions [9]

3.2.6.4 AE – AE Receive

When *_rx_vld is asserted, a receive transaction is valid and *_rx_data should be latched by the CAE. The custom personality must accept the transaction. The CAE can stall data from another AE by asserting *_rx_stall. The custom personality must be able to accept five additional transactions after asserting *_rx_stall.

Chapter 4

Implementation

4.1 First Approach - GE3 Redesign

The initial purpose of this thesis was to relocate GE3 to Convey HC-2ex platform and to take advantage of its high end capabilities. Based to Malakonakis' work, we redesigned GE3 from the ground up. The Golomb Engine was implemented using the ISE Design Suite 12.4, which is compatible with the synthesis tools that exist on the Convey server. Our GE3 works with 1 CPI and has 7 units as shown in figure 3.1:

- **Length control:** It implements the search space reductions and it checks how "far" a mark can be placed on the list vector. Also it checks if the list vector has exceeded the length of the OGR we are looking for.
- **AND-OR control:** It checks bitwise if the list and dist vectors have an "1" at the same position.
- **Multiple shift control:** It utilizes 16 AND-OR units and it gives the shortest number of shifts, so that the shifted-list and dist vectors will not have any collisions.
- **Control:** Based on the outcomes of the three previous units, control decides which operation will be executed. SHIFT, PUSH or POP.
- **Push-Shift logic:** It has the necessary logic to perform the SHIFT and PUSH operations.
- **Stack unit:** It consists of three stacks of 32 depth, and width proportional to the OGR we are looking for and stores the list, dist and length vectors, whenever a PUSH operation is executed. Each memory is a true dual port RAM.
- **Register file:** It stores the new list and the new dist in case of a PUSH or a SHIFT operation.

In our design, contrary to Malakonakis' actual GE3 implementations, we decided to use 1-16 position shifts and not less, since we did not face the same

utilization restrictions, considering the greater resources of a Virtex 6 compared to these of a Spartan 3 or a Virtex 4.

Initially we designed a producer and a single consumer, that solved the OGR(11), to run only on one of the four AEs of the HC-2ex. We produced stubs of seven marks, and the consumer tried to fill the remaining four. We designed 2 different versions of this scenario, in order to gain more expertise on HC-2ex.

In the first version we feed internally the consumer. We have a FIFO of 1024 depth, that stores the produced stubs, and its output is connected directly to the consumer. This design finishes in 3.88 seconds.

In the second version, we abandon the FIFO, and we write the produced stubs on the shared memory. Then we read them and we feed the consumer. In the meantime, we pause the producer until the consumer finishes. We use only one port of one memory controller. This design finishes in 5.02 seconds.

4.1.1 Multiple AEs

The next step was to use 2 AEs and more Consumers: a Producer at AE0 and four Consumer at AE1, with AE2 and AE3 being inactive. The general idea is to have the AEs communicate with each other through the AE-AE interface about control messages, and exchange stubs through the shared memory, as the diagram 4.1 illustrates. The reason for this designing choice is that the stubs for greater OGRs are much larger than a single 32-bit message, that is more than enough for the AEs to send the necessary information about producer's or consumers' status. As it was mentioned at 3.2.2 and 3.2.6.1, the BW of the shared memory even when using only 1 DIMM of 1 MC (5 GB/sec) is larger by one order of magnitude compared to the 600 Mbps (75 MB/sec) data rate of the AE-AE Ring interface.

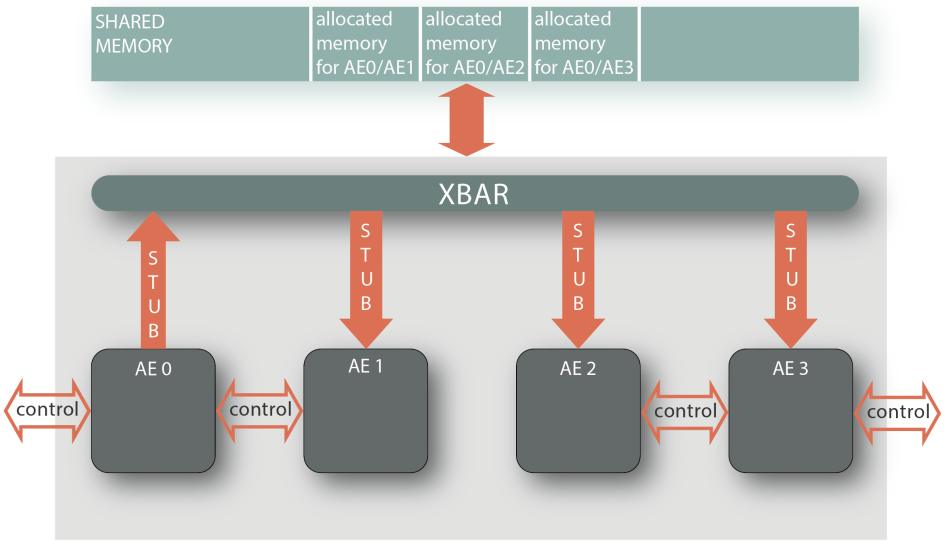

FIGURE 4.1: AE to AE and Memory Communication Diagram

Therefore we developed a basic communication protocol between AE0 and AE1. Also we made use of the Application Engine Execution Mask (AEEM) control register in order to enable only AE0 and AE1. This design finishes in 2.15 seconds.

4.1.1.1 Simulation Problems with HC-2ex

At this point a major problem emerged. Since we had to create two different designs, one with a Producer and its control for AE0, and another with four Consumers and their control for AE1, and subsequently two different bitstreams, we would like to have the tools to simulate the behavior and signal values of these designs. Unfortunately, the Convey platform does not provide such a functionality, for distinct designs in each AE.

So, for the simulation process only, we had to find a way to verify our design. Eventually, we had to compromise on having all the different designs loaded on every AE. By using the unique ID of every AE, we enabled the corresponding design, with the use of a multiplexer for each AE-AE or memory output or input of each design, while keeping a big portion of each AE disabled, since it was confined by the designs of the other AEs, thus having much bigger utilization and being unable to precisely compare our simulation with an actual run. Indicatively, considering that 10% of each AE was reserved due to various interfaces[1], in our final system architecture, where we utilized all the AEs, the resources allocated for every design, in a simulation run, were 20% of the total FPGA. This fact presented a huge drawback while trying to calculate the optimal number of marks our stubs should have, in order to keep a balance between producing stubs too fast and having idle consumers.

4.1.2 Specifying the number of marks of a stub

After completing successfully, a small OGR with two AEs, we decided to move on to OGR(20) with all AEs enabled. Our final goal at the time was OGR(28), and OGR(20) was the intermediate step, since it has about half the length of the OGR(28), 284 and 586 respectively[2].

When we create any stub, there must be enough space left for the remaining marks. The minimum length of this space is the length of the optimal Golomb ruler for the remaining number of marks. For example, if we try to find the OGR(20) with a Producer that creates stubs of 11 marks, and the Consumers fill the remaining 9, then the maximum length of the stubs can be:

$$stubs\ length = length\ of\ OGR(20) - length\ of\ OGR(9) = 284 - 45 = 239$$

Depending on how the stubs' marks will end up, the Consumer might have much more space to try and fit the remaining marks, with the maximum available space in our example being the case of the shortest stub:

$$length\ of\ OGR(20) - length\ of\ OGR(11) = 284 - 73 = 211$$

[1] Convey PDK manual states that 6% of the available logic resources and about 12% of the block rams are used by the basic Convey interfaces.

[2] Although we do not know the OGR(28) yet, we know the length it has.

So, in an actual run the Producer will create stubs with length ranging from 73 to 239. It is evident that a stub with 73 length has much more space for the remaining marks, than that of a 239 length, and it will take much more time for the Consumer to check if it can fit the rest of the marks, while the second will be over in a few cycles.

After taking into consideration the rate at which stubs are produced, the delay between AEs when they communicate directly, the time to write a stub to the shared memory and read it, and the number of Consumers we could fit theoretically in an AE, despite the technical difficulties due to the lack of proper simulation, we decided to set our Producer to create stubs with 11 marks, while the Consumers will fill in the remaining 9 marks.

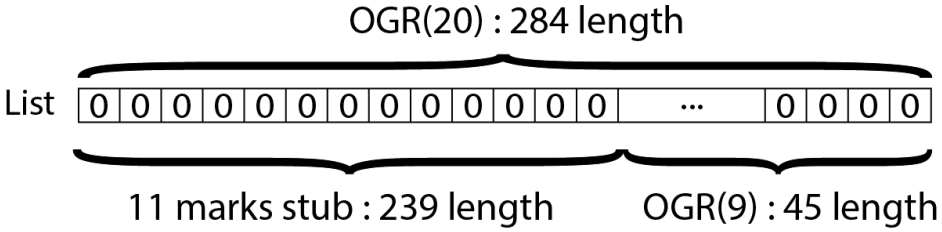

FIGURE 4.2: Size of 11-marks stubs for OGR(20)

4.1.3 Changing the Design

After finishing the design, and taking into account the resources needed for the producer and the consumer modules, we ended up on creating 20 Consumers in each AE. Since an actual run for the whole algorithm would take probably years, we modified our Golomb Engine to finish after one million stubs had been checked by the Consumers. Although the ISE Design Suite, that runs on the server, completed successfully the synthesize procedure, when it run the implementation, at "Place & Route", it popped this message, in all four AEs:

> The router has detected a very high timing score for this design. It is extremely unlikely the router will be able to meet your timing requirements. To prevent excessive run time the router will change strategy.The router will now work to completely route this design but not to improve timing. This behavior will allow you to use the Static Timing Report and FPGA Editor to isolate the paths with timing problems. The cause of this behavior is either overly difficult constraints, or issues with the implementation or synthesis of logic in the critical timing path.

Eventually the process ended with fully routed designs, and with clocks ranging from 54.8 MHz to 62 MHz. However when we attempted to run the designs we got nothing back, even after several hours had passed, although, judging from the simulation, we expected this design to finish in a few seconds.

Therefore we decided to reduce the number of Consumers to four, in order to alleviate the density of the design. Nevertheless, we got the same message at

"Place & Route", and still, when we tried to run the design, it seemed frozen. Our next attempt was to increase the number of stub's marks to 14. In that way, all the stubs would be carried out by the Consumers on AE0, where our Producer also is, thus "disregarding" the other AEs. Likewise the same message appeared at "Place & Route", but this time our design finished successfully after a few seconds. So, at first sight, the "Place & Route" problem was impacting the communication between the AEs.

Therefore we decided to change the design, and convert it into a pipeline, in order to achieve a better clock frequency.

4.2 Final Design

As it was mentioned at 4.1.1, the general idea, is to exchange messages through AE-AE interface, about the status of the Consumers, and to write and read the stubs through the shared memory. There were some changes to our control units, since our Consumer can execute 4 stubs at the same time, after we changed its datapath to pipeline.

4.2.1 Pipeline Architecture

In the HC-2ex platform, when it generates the synthesis and implementation reports, it is not very clear what is the critical path, since it involves signals and paths that belong to the various interfaces and control signals that each AE implements, and which are not in the part of the AE that we can modify. Nevertheless, we tried to insert the pipeline stages, where we believed it will make most impact. Also we tried to reduce the number of levels of logic. To conclude, we added four stages of pipelining to both Producer and Consumer.

4.2.1.1 Producer

Having four pipeline stages at our Producer, means that we simultaneously create four different stubs. In order to secure the coherence of the production process, we want to ensure the attainment of two goals. Firstly, that all four of the sub-producers running, will finish at about the same time, and secondly, to avoid creating the same stub twice.

We achieve the first of these two goals by initializing our Producer to, what we believe, is the most equal way to split the space of all the possible stubs that can be created. In order to do so, first we examine what is the maximum position the second mark of the stub can be placed. Then we split this space into four equal parts, with the intention of initializing each stage of our pipeline to a two-marks signal, each marking the start of these four equal parts. Considering though that when the available space to place the rest of the marks gets smaller, the time it takes for each stub to finish also gets shorter, we decided to not distribute the search space entirely equally. Instead, for each stage, we shortened the distance between the two marks of the initialization signal. This can be made more clear through the example of figure 4.3. As it can be seen, we place the second mark in each of the four initialization signals to positions 2,11,23

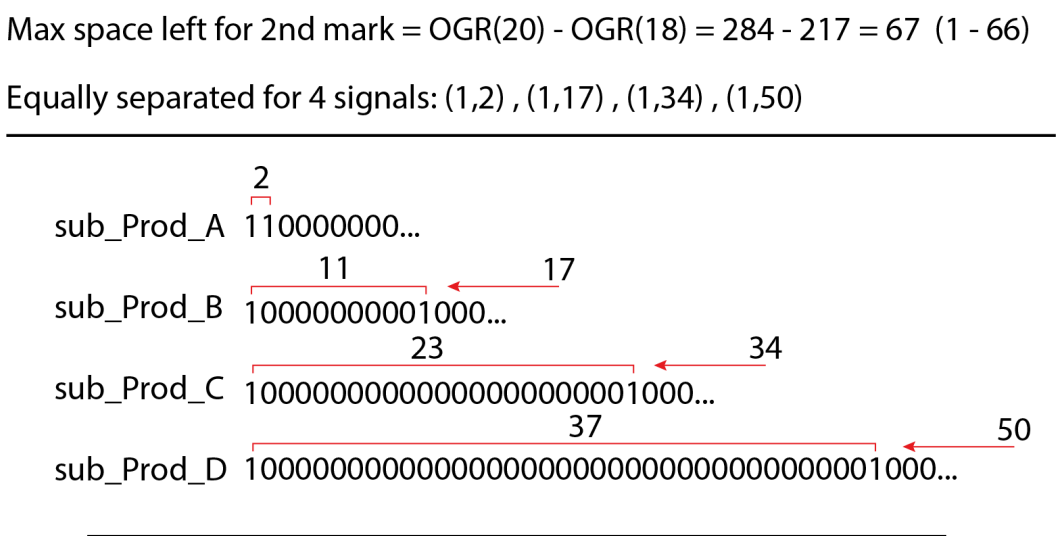

FIGURE 4.3: Initialization signals for the OGR(20) Producer with 11 marks stub

and 37 respectively, although the positions in order to split equally the space would be 2,17,34 and 50. Unfortunately, it is impossible to find the optimal way to split the stub space, since we cannot calculate the number of stubs with their second mark at each of the possible positions.

For the second goal, we set the finish condition for each of these parallel sub-Producers to the moment they enter the search space of the next sub-Producer. In other words if their second mark's position equals the next sub-Producers initialization signal's second mark position, the first sub-Producer finishes.

We added three pipeline registers to our original datapath, forming, with our initial multiplexer, a four-stage pipeline. The first register is placed after the AND-OR control and Length control units, and in the middle of the Multiple Shift Control unit. The MSC unit consists of 16 AND-OR units, with their outputs being checked for the first one to satisfy an if/elsif statement. After the placement of the register, we split the MSC into two units. The first one holds the 16 AND-OR units, while the second one, after the pipeline register, checks the AND-OR outputs and gives us the shortest number of shifts for the list vector, in order to have zero collisions with the dist vector. The second pipeline register was placed after the second MSC unit and before the Push-Shift Logic unit. The last pipeline register was placed before the Stack unit and the Register File. Our Control functions at the same stage as the Second Pipeline Register. The Producer's datapath can be seen at figure 4.4.

In our architecture we do not use a FIFO to store the stubs. When a stub is created it is either fed directly to an internal Consumer of AE0, or it is written to the shared memory for a Consumer of the other AEs. So, while a write command is being executed, or if there are no available Consumers, we have to stall the Producer's operation. To achieve this, we issue a stall command from our general AE0 Control unit, that sends a signal to all the pipeline registers of the Producer, as well as to some of its units, specifically to the Producer's Control, Push-Shift Logic Unit, Register File and Stack Unit.

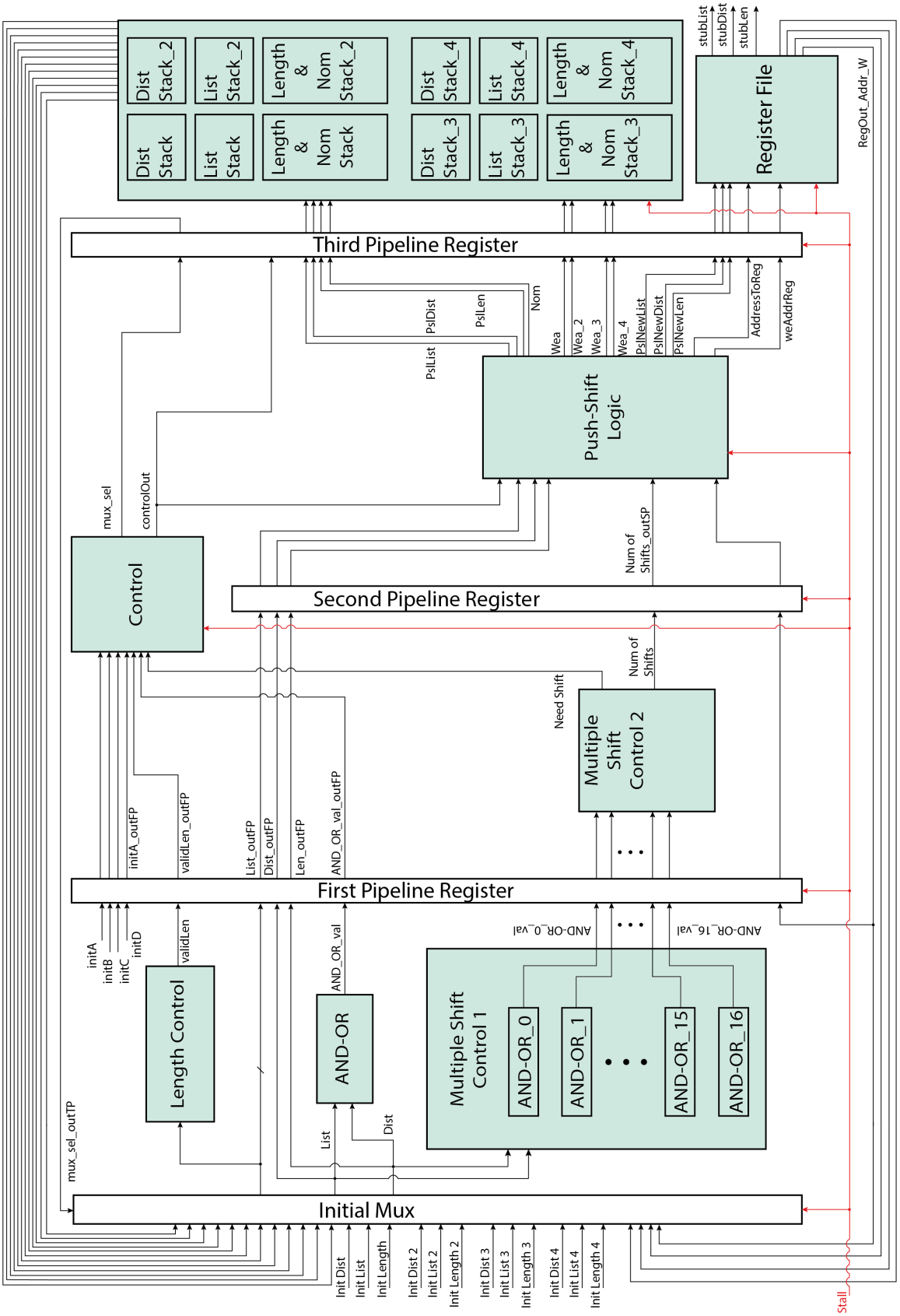

FIGURE 4.4: Producer's Pipeline Datapath

The memories of the Stack unit remain true dual port, like the ones in our single cycle datapath. We need to have true dual port memories, because of the stall functionality. Normally, the address for the corresponding stub of each pipeline stage changes, depending on the operation that is being ordered. If it is a PUSH command, we increment the address by one, and write the pushed list vector, and when we execute a POP we read the output of the memory and we decrement by one its address. However, when we stall the Producer's operation, while we perform a PUSH or POP command, then we will either get the wrong output if we execute a POP or write to the wrong address if we execute a PUSH. With true dual port memories we can solve this problem, by using the second address and the second output, to either write at the correct address or read the correct output. Also we have one multiplexer before the memories, in order to select the appropriate address, and a multiplexer after each memory, so as to get the desired output. At figure 4.5 we can examine the port differences between single port, simple dual port and true dual port memories.

All our memories have 32 depth and ranging width depending if it is the list/dist memories or the length memory. Moreover, in our pipeline datapath, we need to save the number of marks placed at the current stub. That needs an extra BRAM of 32 depth and 5 width. In order to save resources though, we concatenate the length and number-of-marks signals and store them to the same BRAM, thus creating a 32 depth and 13 width memory for OGR(20)'s Producer. The reason for this is that in Virtex-6 the minimum size, that BRAM can be configured, is as an 18 Kb RAM, which is more than enough in order to store both the length and the number of marks, and it saves us the extra utilization of four 18 Kb of BRAM (one for each stub being generated) [33].

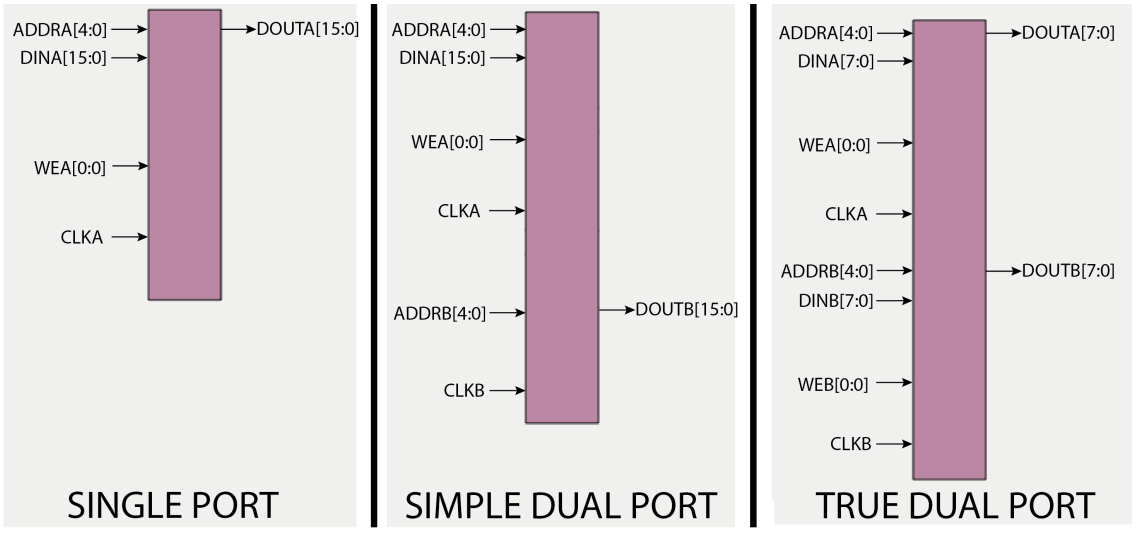

Figure 4.5: Available BRAM options

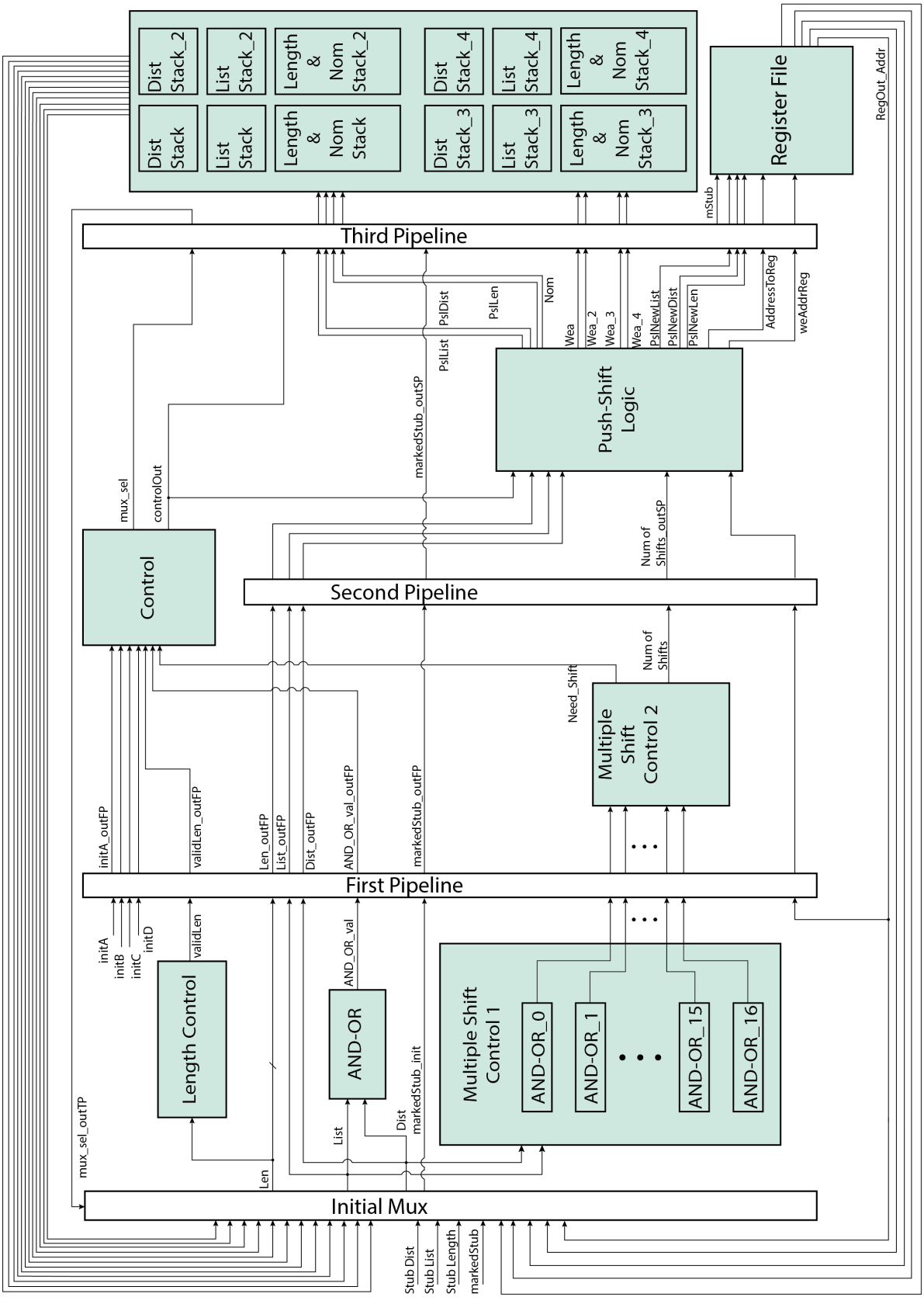

FIGURE 4.6: Consumer's Pipeline Datapath

4.2.1.2 Consumer

In general, the Consumer's datapath is almost identical with the Producer's. The main difference is that there is no stall functionality. Also the memories inside the Stack unit are single port instead of true dual port, that we used in our single cycle datapath. This is a great improvement, since the available BRAM is the factor that confines us to adding more Consumers, from a resource utilization perspective. So, by using single-port BRAM, we can add almost the double number of Consumers in each AE [19, 32, 33]. Furthermore, we also have twelve memories, three for each stub running. One for the list, one for the dist and one for the length/nom concatenated signals. Finally, there is an extra 1-bit signal, "markedStub", used for back up purposes.

The Consumer's main FSM, depending on how many stubs are concurrently running, and if a stub finishes or a new stub arrives, has a total of 17 states, as shown in figure 4.7.

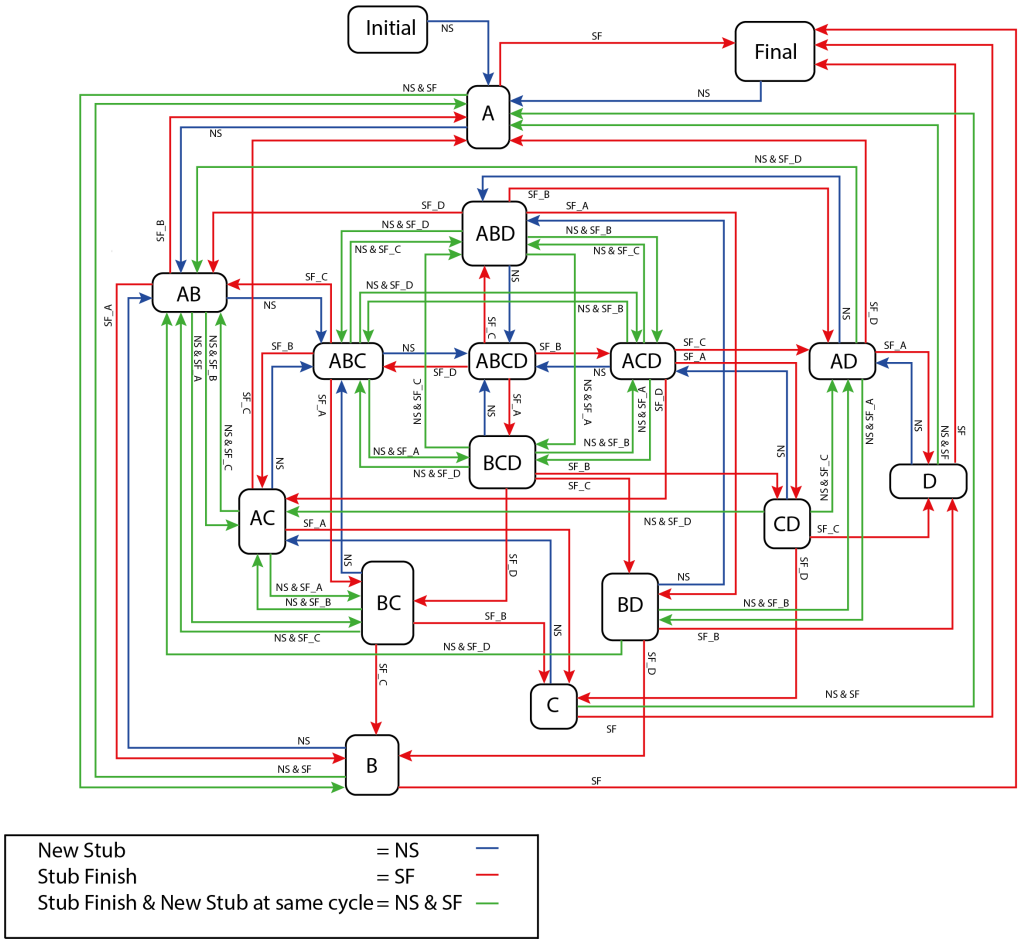

FIGURE 4.7: Consumer's FSM

In order to display this in a more clear way, we split the FSM to three sub-FSMs, as shown in figures 4.8, 4.9 and 4.10. Specifically 4.8 is the fsm for when a new stub arrives, 4.9 for when a stub finishes and 4.10 for when a stub finishes while a new stub arrives at the same cycle. When there are four stubs running,

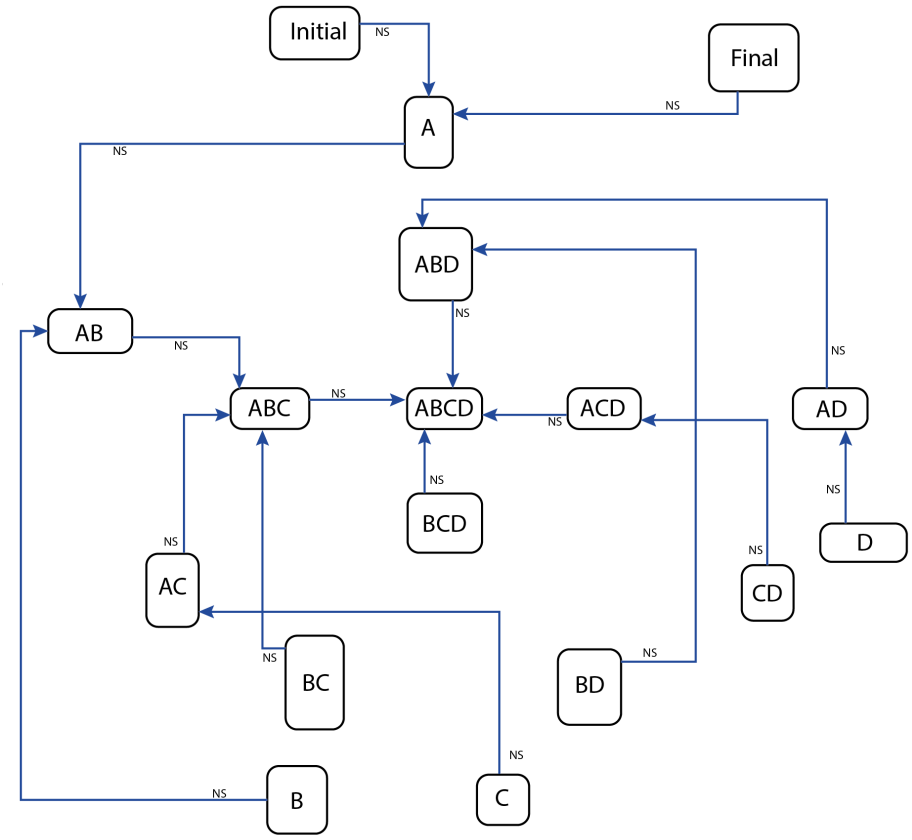

FIGURE 4.8: Consumer's FSM for new stubs

we are at the ABCD state, and the Consumer sends to the AE's Control unit a signal, stating that he cannot accept any more stubs. Moreover each Consumer has an output signal that informs its corresponding AE, that at least one stub is running, therefore it is still a working Consumer.

If a Consumer finds the ruler we are looking for, the list vector is stored in a register, and, after the current stub finishes, the Consumer informs the main AE control, which in turn informs the AE0's Control, in case the Consumer is at another AE. When the design finishes, the list vector is written to the shared memory, and the host reads it and prints the marks of the ruler to the console. As our Golomb Engine is designed, if a ruler is found it superscribes any ruler previously found, therefore at least two rulers will be found, and the final output list will be the mirror image of the first one.

4.2.2 AE to AE Communication

For the communication between the AEs, we used the ring interface, that worked best for our purposes, since it provides communication with both the neighbors of each AE. In our design AE0, where the Producer is placed, communicates directly with AE1 and AE3. The messages that are destined for AE2, are forwarded through AE3, as shown in diagram 4.1.

In AE0, we have a unit, Share Stubs And Back Up unit (SSABU), that is responsible for the communication, for deciding which Consumer, in the other

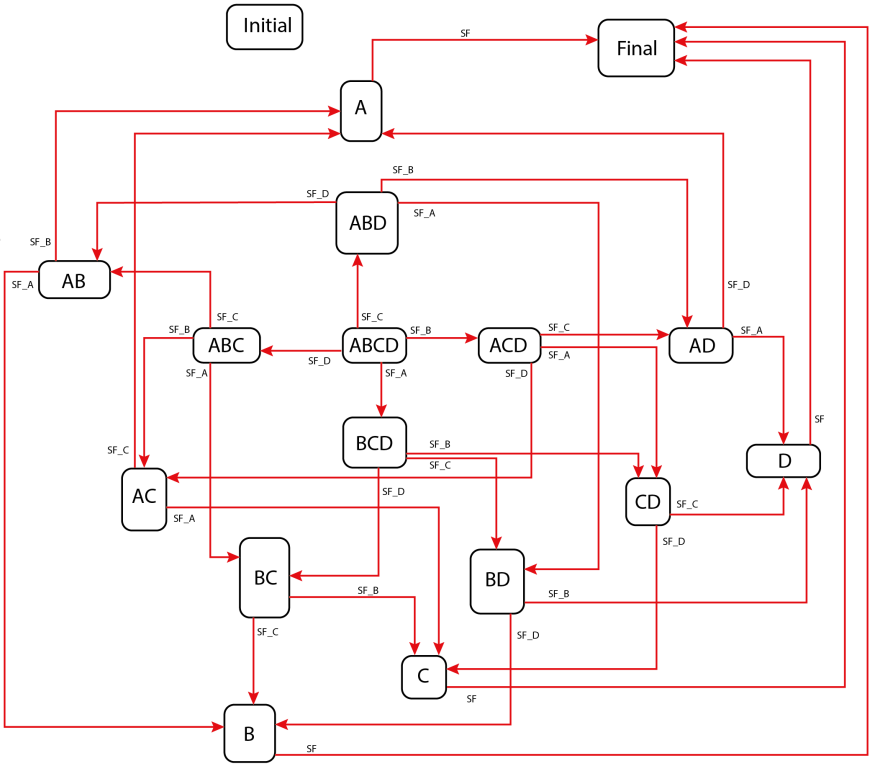

FIGURE 4.9: Consumer's FSM for stubs finishing

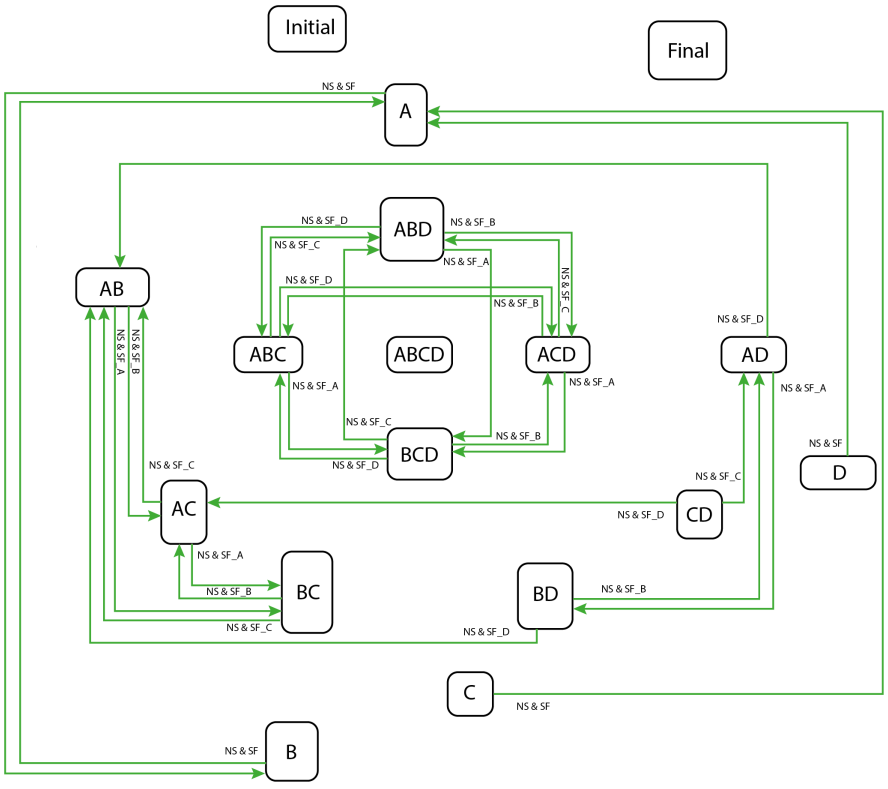

FIGURE 4.10: Consumer's FSM for stubs finishing while new stubs arrive

AEs, is to be "fed" with a stub, for holding the necessary info about which Consumers are empty, as well as for being our fail-safe unit, in case of a system crash.

SSABU has a vector, memWriteVector, with length equal to the number of Consumers that are at AE1, AE2 and AE3. Each bit corresponds to a specific Consumer and indicates if the Consumer is full or not. Whenever a stub is generated, and after we check that all the Consumers of AE0 are full, then we check the memWriteVector starting from the LSB. So, the priority in which we feed the other AEs is first AE1, then AE2 and last AE3. Figure 4.11 illustrates this vector.

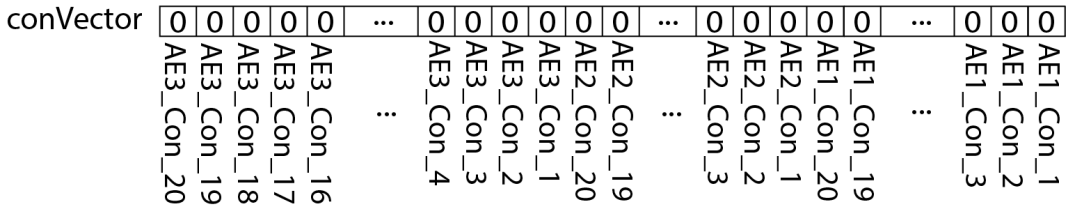

FIGURE 4.11: AE0 Vector for other AE's Consumer's status

The AEs exchange 32-bit messages through the AE-AE ring interface. As it was stated at sections 3.2.6.3 and 3.2.6.4, when an AE sends a message, it asserts the nxtae_tx_vld or prvae_tx_vld, depending on whether it wants to send it to the next or the previous AE respectively, and sends the message through nxtae_tx_data or prvae_tx_data. After an average of 25 cycles, as it was shown in simulation runs, the message arrives at the next AE, where the nxtae_rx_vld or prvae_rx_vld is asserted, and the nxtae_rx_data or prvae_rx_data, respectively, carries the message.

When our design stars running, AE1, AE2 and AE3 inform AE0 about the status of their Consumers. From that info we initialize the memWriteVector. AE2 sends the message to AE3, which forwards the message to AE0. In order for AE0 to distinguish the origins of the messages that receives from AE3, we use the 32nd and 31st bit of the 32-bit data signal. If the 32nd bit is an '1', then the message comes from AE2, and if the 31st bit is an '1, then the message comes from AE3. Likewise, when AE0 sends a message to AE2 or AE3, the 32nd or the 31st bit respectively will be an '1'. If both of these bits are an '1', this means that the message is addressed to both AE2 and AE3. This happens when the Producer is finished, and we want to inform the other AEs that after their Consumers finish, they should go to the final stage, write the OGR to memory, if it was found at one of them, and then inform AE0, so we can finish our execution. The structure of the messages exchanged between AE3 and AE0 is displayed at figure 4.12.

Furthermore, we have to mention how AE3 handles the messages from AE2. In the case that AE3 receives a message from AE2, that it needs to forward it to AE0, but at the same time AE3 wants to send a message to AE0, then it gives priority to the message from AE2 and at the next cycle it sends its own message.

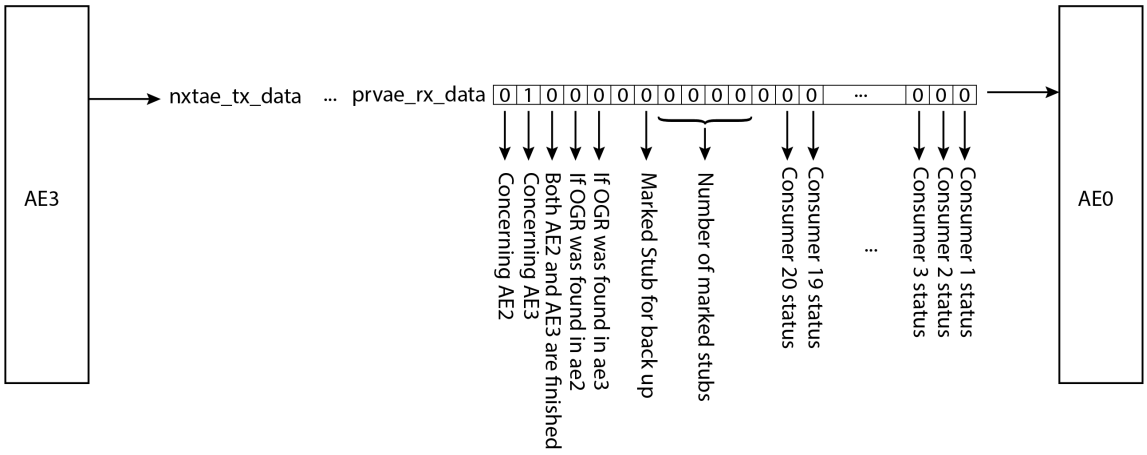

FIGURE 4.12: Display of the structure of AE to AE messages.

4.2.3 Shared Memory and Stub Distribution

In order to send the stubs to other AEs we concatenate the dist, the list and the length signals and then we write them to the shared memory. When our design starts running, the host processor calls the routine *void * cny_cp_malloc (size_t size)*, that allocates a number of bytes of coprocessor memory equal to the size argument [10]. For the OGR(20) we make three allocations of 1280 bytes, and we sent the pointer of each allocation to AE1, AE2 and AE3 respectively. AE0 gets all three pointers. The 1280 results from:

- The list and dist vector have length of 239 bits each.
- The length vector has a length of 8 bits.
- So, for every stub we need 239+239+8 = 486 bits.
- Load and store memory requests read and write data in 64-bit values, so for a stub, rounded up, we need 8 * 64 = 512 bits allocated in the shared memory.
- Our intention was to have 20 Consumers in every AE, so 20 * 512 = 10240 bits = 1280 bytes.

Therefore, we have allocated 64 bytes (512 bits) for each individual Consumer we have in AE1, AE2 and AE3, in order to share the produced stubs. It is not the most optimized way to allocate memory for our scheme, but since we have abundance of memory, 3840 bytes in total are paling in comparison to the available 64GB of shared memory. Moreover, in this way we can make targeted writes to each Consumer, thus simplifying the memory management. Figure 4.13 displays the way we allocate memory on the shared memory.

We mentioned in section 3.2.4.1, that no response is returned to the AE personality for write operations. Still, we need a mechanism to verify that the stubs are written and we can read them safely. Fortunately, the Convey

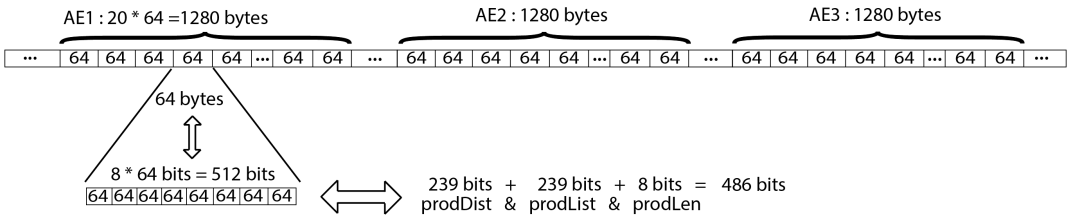

FIGURE 4.13: Allocated part of Shared Memory for each AE

platform has a write flush function. We can send a flush write request, after the last write that need to be complete. After a write flush has been issued at an MC request port, the corresponding flush-complete signal will be asserted for one cycle when all outstanding writes to that port have completed. Then we can safely inform the target AE, in order to read the stubs.The average number of cycles in simulation for the flush-complete signal to be asserted is 185 cycles.

We also thought of informing the other AE instantly, and utilizing a counter before we start reading the stub, thus saving the extra time of the 25 cycles, in average, for the AE to AE message to arrive. Although this design choice worked successfully when we tested it, in the final design we gave it up, since the simulation results on which we based the delays, for the AE to AE communication and the write-flush/flush-complete process, are not very reliable.

4.2.4 Back Up System

As we mentioned at section 4.2.2, SSABU unit is also our fail-safe unit in case of a system crash. In order to achieve this, we need a way to know at any point which stubs have finished, and if our system crashes, to be able to restore our producer from this point. Considering that for big OGRs, an entire execution of our design might take years, this functionality is considered necessary. In order to do that, first we create a counter that we can change depending on how fast Consumers finish checking the stubs. For our design we set that to one million stubs. Also, when we had a single cycle producer, it was easier to save the necessary info, since our stubs were produced sequentially. But now our Producer creates four completely different stubs and not necessarily in a circular order, meaning that if we check the last four stubs produced, it does not entail that each one will be originated from one of each of the four "sub-Producers". So, in order to ensure that we do not lose track of any of the sub-Producers' progress, when our counter hits one million stubs created, then we stop the counter and we "mark" the next ten stubs that we produce. When we execute these marked stubs, inside the Consumer there is a 1-bit signal that follows the stub, through the pipeline stages. When this stub is finished, we check if it was a marked one, and if that is the case, then we inform the SSABU unit. If the Consumer that receives the marked stub is on another AE, while informing that AE, that we wrote the stub to the shared memory, we assert the 27th bit in the *_tx_data message. In this way, that AE handles the stub as a marked one. When one or more marked stubs finish on that AE, it sends

a message to AE0, with high the 26th bit, and bits 25 down to 22 having the number of marked stubs that are finished. This happens if the marked stubs finish simultaneously. Otherwise it sends a message to AE0 whenever a marked stub finishes. When the SSABU receives ten acknowledgements, we start the counter again.

Moreover, when we create the marked stubs, AE0 saves their list vector to the shared memory, on a space allocated specifically for this purpose. The list vector is sufficient in order to retrieve also the dist and length vectors. When we allocate memory at the start of our execution, we allocate, for back up purposes, space equal to the length of twenty list vectors, separated in two parts. For the 11-mark stubs of OGR(20) that is

$$239\,bits * 10 = 2390\,bits \approx 299\,bytes$$

and since we write in values of 64 bits we need

$$5 * 64 = 320\,bytes$$

for each of the two spaces allocated. So, in total we allocate 640 bytes. At each time, alternately half of this space holds the lists of the stubs that have been executed successfully, and the other half holds the lists of the stubs that are currently being executed or that may have finished. In case of a system crash, we will back up from the older version, that is easy to find out which one is.

So, in short, our SSABU does not know which Consumer runs a marked stub. The only info that SSABU is interested in, is how many of these marked stubs have finished. When it gathers a total of 10 acknowledgements, it activates the counter again, and changes the destination for the next back up between the two allocated spaces.

Figure 4.14 shows the datapath of AE0 with all the major modules, modified for solving the OGR(20). The datapath of AE1, AE2 and AE3 is simpler than AE0. AE1 and AE2 are almost identical, with the only difference being the direction to which they communicate. AE1 communicates with the previous AE, that is AE0, while AE2 communicates with the next AE, that is AE3. AE3 is similar to AE2, but has some extra logic in order to forward the message from AE0 to AE2 and vice versa, as it was analysed at subsection 4.2.2 and illustrated in diagram 4.1.

4.2.5 28 Marks Golomb Engine Modification

After establishing that our design functions as it is intended, we modified it for solving the 28 marks OGR. The length of OGR(28) is 586 [25], as we already know. That means the ruler we are looking to prove is more than double the size of the OGR(20), that we used so far (284 size). So, in the best case scenario we can fit roughly 9 Consumers in every Application Engine of the Convey HC-2ex platform. The bottleneck responsible for this is the available BRAM. Every Consumer needs 2.4 Mbits of BRAM, while the Producer needs 4.2 Mbits. Since the BRAM in Virtex-6 is 25.92 Mbits, and the available BRAM, that equals 88% of the total, is 22.8 Mbits, we get the respective number of Consumers.

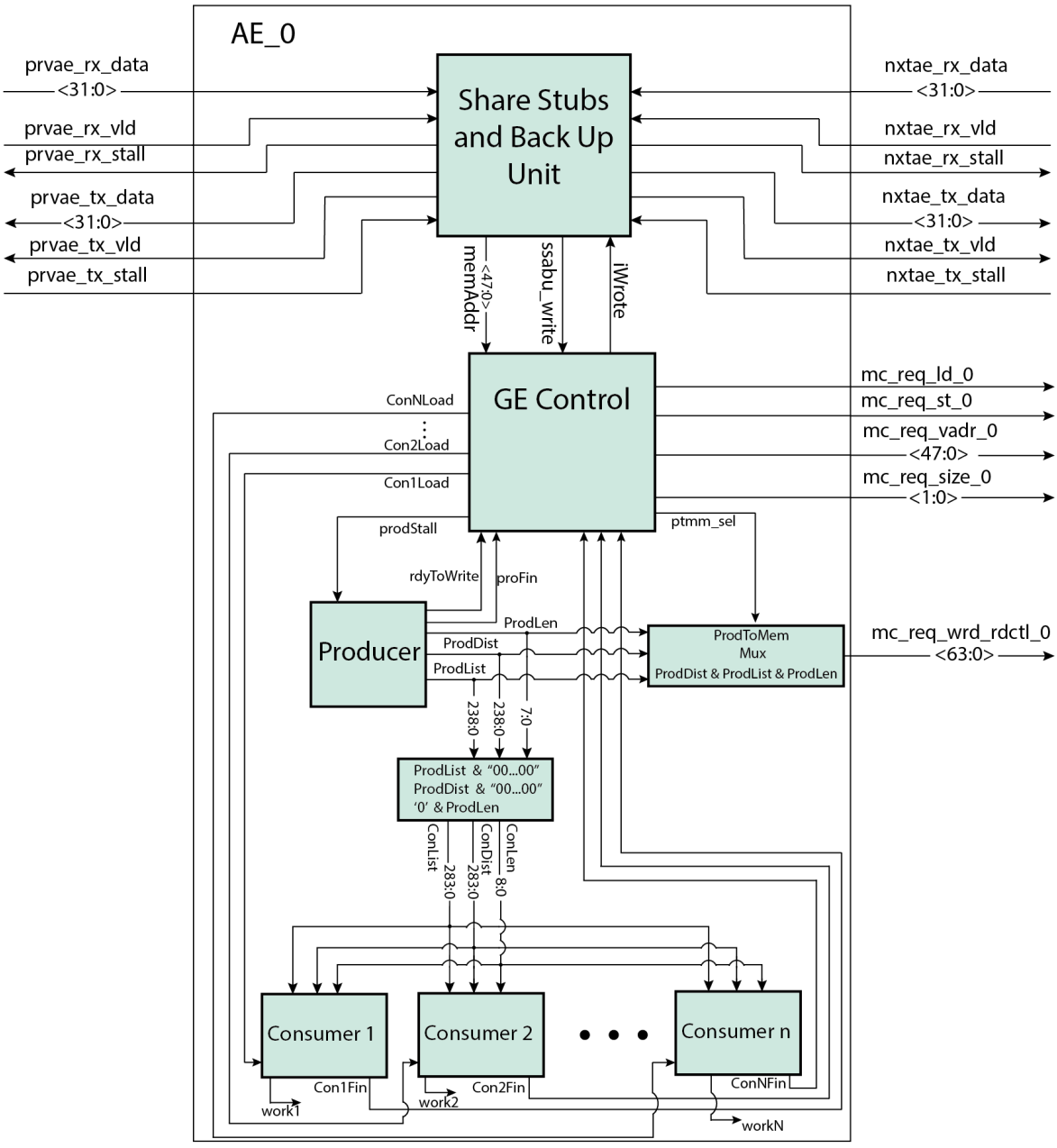

FIGURE 4.14: General Architecture of AE0 for OGR(20) & 11 marks stubs

Considering the simulation results for the optimal case of 9 Consumers per AE, and 7 Consumers in AE0, due to the Producer, meaning 34 Consumers in total, and 136 stubs running concurrently, we configured our Producer to create 18-marks stubs. That means that the Consumers have to fill the remaining 10 marks. So, the available space they require is at least the length of OGR(10) = 56. Therefore, our Producer creates stubs of maximum length 586 - 56 = 530. The basis behind the 18-marks stubs is to keep the Consumers always busy, meaning that our goal is to produce faster than we consume, since we prefer to have our Producer idle, rather than our Consumers.

As we mentioned at section 4.1.1.1, the Convey HC-2ex does not provide the user with a proper simulation tool, when we need to utilize different designs in each AE, so the optimal number of 9 Consumers per AE, was an estimation based on the resources needed for each Consumer module, and the stubs' production and consumption rates.

Furthermore, we allocate 1224 bytes for each AE, on the coprocessor memory, that is necessary for 9 Consumers. Each stub has 530+530+10 = 1070 bits => 17 * 64 = 1088 bits => 9792 bits for 9 Consumers = 1224 bytes.

4.2.6 Setting Up our Design at HC-2ex

At Convey server, inside the 'Makefile.include' file:

1. We declare which version of the platform we want to use. In this case "hc-2ex".

2. We declare the path where our VHDL and memory files are. The memories were generated on our personal computer using ISE's IP (CORE Generator & Architecture Wizard) and then the necessary files (.ngc, .xco, .v) were uploaded to the Convey server.

3. We set the MC_XBAR to '1', in order to enable the optional Crossbar MC Interface.

4. We set the MC_READ_ORDER to '1', so as to enable the optional Read Order Cache MC Interface

5. We set the AE_AE_IF variable to '1', in order to activate the AE-to-AE Interface.

First we place a smaller version of the four designs on the same project, for simulation purposes, as described at section 4.1.1.1. After we establish that everything is running correctly, considering the simulation's limitations, we create four different projects, one for each AE. From each project we will generate a bitstream, that we will upload to the HC-2ex Server. Then we will combine these four bitstreams into a specific format file that is compatible with the HC-2ex server and it will configure each AE to the corresponding design.

Chapter 5

System Evaluation

As it was mentioned at 3.1.3, most of the operations being executed are SHIFT operations, especially as the ruler gets larger. Moreover, a great percentage of them range between 1 and 16 shifts, and considering the resources needed for more than 16 shifts, the trade-off is not worth it. What is more, although on the previous work of Golomb Engine 3 they found that 1-16 shifts is the optimal number that the Multiple Shift Control unit should handle, in practice, and due to the limited capabilities of the tools and the inadequate memory capacities of machines at the time, they were not able to test GE3 on a top-shelf FPGA, like Virtex5. Since we have much greater processing power and memory capabilities, we chose to maintain the 1-16 multiple shifts and examine if our design meets our expectations.

Taking into account the nature of the problem of OGR derivation, it is not possible to compare accurately the performance of our design for stubs with different number of marks and draw the right conclusions, since we are incapable of calculating the exact number of stubs that will be produced. On the other hand, comparing a single FPGA versus a multi-FPGA run, for a specific OGR and a fixed number of stubs' marks, seems more meaningful.

Furthermore, we have to mention that, although in the context of testing the performance of Convey HC-2ex platform and getting familiar with the technology it offers, we had to modify our Producer's datapath into pipeline, as well as the Consumer's datapath, in reality it is unreasonable to pay the costs of AE-to-AE communication and memory utilization, whereas we could create four identical designs with one Producer in every AE, and initialize them differently, so that we split the stub space in four equal parts. Obviously, as it was stated at 4.2.1.1, we also have to deal with the problem of splitting the stub space in equal parts, in terms of the time required for them to be consumed, with the solution not being obvious, but rather intuitive, and certainly impossible to prove if we made an accurate decision.

What is more, when we tried to instantiate the maximum number of Consumers, according to the available resources, we had the same failing results in Place & Route stage, as we had with the single cycle datapath. Therefore, we reduced the number of Consumers, until the implementation procedure could complete.

5.1 Convey HC-2ex Performance Evaluation

Considering the above, at our initial measurement of the OGR(20) with 11 marks stubs, we concluded at having 2 Consumers at each AE, that can concurrently process 32 stubs. The next step was to modify our design, so that it run only on one AE. Therefore we disabled the AE-to-AE communication and any interaction with the shared memory, and set the AE2, AE3 and AE4 to idle state, thus having only AE1 active, with 1 Producer and 2 Consumers in total. In both cases we have a clock of 150 MHz.

We run both of these setups for 1 million and 16.7 million stubs. Table 5.1 details the execution times for both cases.

Number of Stubs (million)	**Execution time (sec.)**		**Speedup(X)**
	1 FPGA	**4 FPGAs**	
1	2.02	0.93	2.17
16.7	34.05	15.61	2.18

Table 5.1: Run times for single FGPA vs multi-FPGA for OGR(20) with 11-marks stubs

A number of conclusions can be deduced from this table. First of all, the performance gain by deploying multiple AEs is not as high as expected. Although we have an increase of the number of AEs from 1 to 4, the speedup we gain is 2.18. This behavior can be attributed to the latency of the AE to AE communication and the stub's distribution through the shared memory. Due to the balance that we try to achieve between the Producer and the Consumers, regarding the rate of creation and consumption of stubs, there is frequent communication between the Producer and the Consumers of the other AEs, resulting to numerous interruptions in the flow of the execution, that add a considerable overhead to the execution time. Regardless of the OGR we are looking for, as long as we aim to keep this balance, we cannot avoid this overhead. The solution might be to produce shorter stubs, thus reducing the communication time to a small percentage of the total execution time, since there will be produced fewer stubs, but the Producer will be in an idle state for most of the time.

Nevertheless, it is apparent that there is a linear relation between the execution time and the number of stubs, for a specific OGR and a fixed number of marks for its stubs. This analogy happens both in the single-fpga and in the multi-fpga executions. Therefore, although at our design we chose initially to utilize only one memory controller at each AE, eventually it seems more efficient to use more MCs, especially at greater OGRs with lengthy stubs, where we will split the stubs in many 64-bit parts, in order to store them to the shared memory and load them from the other AEs.

What is more, it is important to emphasize the low performance of the ISE 12.4, concerning the results in the actual number of Consumers we had versus our intended target of maximum number of Consumers. As it was stated at 4.2.5, the resource on which depends how many Consumers can fit in our design, is the available BRAM. With OGR(20) the BRAM required by every Consumer is 1.2 Mbits and 2.04 Mbits for the Producer. Since the available

BRAM in every AE is 22.8 Mbits, that results in 17 Consumers in AE0, and 19 Consumers in each of the rest AEs, adding up to 74 Consumers in total. However, in our successful implementation, we end up with only 8 Consumers, 2 in each AE.

Considering that for OGR(20) we had so poor results in Place & Route, our expectations for OGR(28) were low. Indeed, although in theory we expected to have 34 Consumers, in our final implementation we had:

- Only our Producer at AE0 without any Consumers, running at 96 MHz.
- Two Consumers at AE1, running at 96.7 MHz.
- One Consumer at AE2, running at 149.3 MHz.
- Two Consumers at AE3, running at 103.7 MHz.

As it was expected there is half the number of Consumers compared to OGR(20), since the length of OGR(28) is double the length of OGR(20). When it comes to the best case scenario we have 5 instead of 34 Consumers. Moreover at AE0 we have a clock of 96 MHz, although there is only our Producer and zero Consumers. Furthermore, we were unable to get any results from an actual execution, a behavior that, combined with the poor performance on the implementation stage, can be attributed to bad synchronization among the four FPGAs.

Since our problem evolved into a capacity one, it is obvious that by having 1-16 shifts to choose, in our Multiple Shift Control Unit, we cannon exploit the full potential of the resources we have in our disposal. Instead of examining how well the system would respond, if we reduced the number of multiple shifts, since this case was researched on the previous work on Golomb Engine 3 at 3.1.3, we decided to compare the Place & Route results from two contemporary FPGAs with the ones we got from the Convey HC-2ex. The two evaluation boards we utilized are ZCU102 that is based on the XCZU9EG MPSoC and ZC706 that is based on the XC7Z045 SoC.

5.2 ZCU102 & ZC706

5.2.1 Xilinx Zynq UltraScale+ MPSoC ZCU102

The ZCU102 is a general purpose evaluation board for rapid-prototyping based on the Zynq® UltraScale+ XCZU9EG-2FFVB1156E-2-i MPSoC (multiprocessor system-on-chip). High-speed DDR4 SODIMM and component memory interfaces, FMC expansion ports, multi-gigabit per second serial transceivers, a variety of peripheral interfaces, and FPGA logic for user customized designs provides a flexible prototyping platform [37].

5.2.2 Xilinx Zynq 7000 SoC ZC706

The ZC706 evaluation board for the XC7Z045 SoC provides a hardware environment for developing and evaluating designs targeting the Zynq®-7000

XC7Z045-2FFG900C SoC. The ZC706 evaluation board provides features common to many embedded processing systems, including DDR3 SODIMM and component memory, a four-lane PCI Express® interface, an Ethernet PHY, general purpose I/O, and two UART interfaces [36].

Table 5.2 presents the available resources for Virtex6, XCZU9EG and XCZ7Z045.

Model	CLB LUT (K)	CLB Flip-Flops (K)	BRAM (Mbits)	DSP Slices
Virtex-6 (HC-2ex)	474	948.5	25.92	864
XCZU9EG (ZCU-102)	274	548	32.1	2520
XC7Z045 (ZC706)	218.5	437	19.2	900

TABLE 5.2: Resources of Virtex-6-XC6VLX760, XCZU9EG-2FFVB1156 and XC7Z045-FFG900

5.2.3 VIVADO versus ISE

The ISE design environment offered by Xilinx has been superseded over the last years by VIVADO®. VIVADO is a design suite produced also by Xilinx, that we utilized in order to relocate our design to ZCU102 and ZC706. Besides the reasonable enhancements over ISE, regarding additional features, such as support for system on a chip development and high-level synthesis, VIVADO also offers improved results in runtime and memory utilization, and it delivers robust performance with low power consumption without affecting the system design's timing. The reason for all these is the way VIVADO has been built from the ground up to address the productivity bottlenecks in system-level integration and implementation.

It is also worth mentioning that the version of ISE, that the Convey HC-2ex platform utilizes, is admittedly very poor, when it comes to performance, in either running time or mapping, placing and routing a design, especially in very dense designs. Considering also that the users of the platform have very limited choices as opposed to the ISE on our personal computer, e.g. we could not apply any timing constraints, it is natural to have low expectations from the synthesis and implementation processes for dense designs.

The version we used is VIVADO 2018.2. As for the ISE utilized in the server where we create the bitstreams for the Convey HC-2ex platform, it is the 12.4 version.

5.2.4 ZCU102 & ZC706 relocation

Although at start the VIVADO design suite seemed more complex than ISE and we needed some time to become acquainted with it, eventually the migration from ISE proved to be quite easy. The relocation of our design was the same for both ZCU102 and ZC706. In its core, our design can be perceived as a black box, that consists of a Producer and a number of Consumers, with 2

inputs, clock and reset, and 2 outputs, a finish signal and the optimal golomb ruler (list signal). It is very similar to the single FPGA version that we executed at the HC-2ex platform, with the AE-to-AE interface's status disabled, where we feed internally the Consumers, without having to write and read from the coprocessor memory. Therefore, after the necessary modifications, and the memories upgrades, that VIVADO performs in an almost automated style, we were able to test and compare the implementation results with the ones from the Convey HC-2ex.

5.2.5 ZCU102 & ZC706 Implementation Results

Tables 5.3 and 5.4 present the maximum number of Consumers that each FPGA can hold based on their resources, and the actual number the FPGA implementation achieves, for OGR(20) and OGR(28) respectively. Moreover, as it was stated at 4.1.1.1, the basic Convey interfaces occupy 12% of the overall block ram of HC-2ex. The results for Virtex-6 and the Convey HC-2ex platform originate from the server provided from Technical University of Crete, that was custom built specifically for the HC-2ex platform, and it is powered by an Intel® Xeon® E5-2620 at 2.00GHz. The results for XCZU9EG and XC7Z045 derive from the VIVADO 2018.2 installed on our perconal computer that has an Intel® Core™ i7-6700HQ at 2.60GHz.

Model	**BRAM (K) Total / Available**	**Max # of Cons**	**Actual # of Cons**	**1M stubs (sec)**
Virtex-6 (HC-2ex)	25.92 / 22.8	17	2	2.02
Convey HC-2ex	103.68 / 91.2	74	8	0.93
XCZU9EG (ZCU-102)	32.1	24	24	0.19
XC7Z045 (ZC706)	19.2	13	13	0.35
Software	-	-	-	2250

Table 5.3: Maximum and actual number of Consumers for each FPGA for OGR(20) with 11-marks stubs

Model	**BRAM (K) Total / Available**	**Max # of Cons**	**Actual # of Cons**
HC-2ex single AE	25.92 / 22.8	7	0
HC-2ex 4 AEs	103.68 / 91.2	34	5
XCZU9EG (ZCU-102)	32.1	10	10
XC7Z045 (ZC706)	19.2	5	5

Table 5.4: Maximum and actual number of Consumers for each FPGA for OGR(28) with 18-marks stubs

The first observation that can be made is the superiority of VIVADO over ISE. It is unambiguously clear, in both cases, that VIVADO excels at implementing our design in the most optimal way. Specifically, on our single FPGA

implementation of the HC-2ex, we achieve a best case of 2 Consumers instead of 17 for the OGR(20) and an ineffective outcome of 0 Consumers instead of 7 for the OGR(28), since we could instantiate only the Producer. Regarding the results of a complete multi-FPGA implementation, ISE delivers 8 instead of 74 Consumers for the OGR(20) and 5 instead of 34 Consumers for the OGR(28), both results equivalent to their single-FPGA version. On the other hand, concerning XCZU9EG and XC7Z045, VIVADO delivers impressive results for both the OGRs we tested. For OGR(20) it implements 24 out of 24 Consumers at XCZU9EG, and 13 out of 13 Consumers at XC7Z045. As for the OGR(28), VIVADO manages one more time to meet the ideal scenario. At XCZU9EG we get 10 out of 10 Consumers and at XC7Z045 5 out of 5 Consumers. Furthermore, we compared our OGR(20) results with a C++ code, that we executed at a PC running on an Intel® Core™ i7-6700HQ at 2.60GHz, although the code is quite different than our hardware design, since it does not implement a Producer-Consumer model, nor does it support multiple shifts. The way that we count each stub on our software was with the utilization of a counter that we increased each time the 11th mark moved in the LIST vector. What is more, a conventional PC handles differently the list and dist vectors, in parts of 32-bits, while our design works at a 150 MHz cycle, where it checks the whole length of LIST and DIST vectors. Considering the long length of both vectors when they are modified for OGR(20), the x1100 speedup of our harware seems reasonable.

Moreover, VIVADO meets our timing constraints in every implementation, with a 7.5 ns clock period or 133 MHz frequency. Considering the significantly higher utilization on the two contemporary FPGAs, on OGR(20), at XCZU9EG we can have more than 3X speedup, since we have 24 versus 8 consumers, yet we eliminate the cost of communication between different FPGAs. As for the OGR(28), just from a spatial point of view, we can have more than 2X speedup. XC7Z045 as well can achieve a speedup of 1.6X for OGR(20), and, although we have the same number of Consumers at the OGR(28) implementation, since HC-2ex could not give any execution results, due to synchronization difficulties, XC7Z045 prevails.

What is also noteworthy is the time required by each design suite to finish the synthesis and implementation processes. While ISE can take up to 2 days and even then it may fail in completing the implementation stage, VIVADO needed less than 30 minutes to finish, with the additional benefit of delivering excellent results.

Chapter 6

Conclusions & Future Work

Optimal Golomb rulers are utilized in numerous applications originating from a wide range of scientific fields. Finding and proving an optimal Golomb ruler is a tedious process, that can benefit a great deal from FPGA based implementations that exploit the parallelized nature of the problem. We conclude this thesis by summarising the key aspects of our work and then we suggest a few ideas in an attempt to improve the current status of our Golomb Engine.

6.1 Conclusions

Our main goal was to advance the Golomb Engine project, through relocation to a big scale, multi-fpga, server based, hybrid-computing system, Convey HC-2ex. After redesigning the Golomb Engine from the ground up, we modified the initial, single-cycle datapath, to a four stages pipeline, in order to achieve a better clock cycle, and thus resolving a communication problem between the FPGAs, that we considered it was resulting from synchronization issues. Furthermore we maintained the Producer-Consumer model, with one Producer in one of the FPGAs, and as many Consumers as the designing tool succeeded at implementing.

The pipelined version, although it solved the communication problems, it produced poor results regarding the number of modules instantiated, due to limitations of ISE, the designing tool that the server is compatible with, especially as the OGR gets larger. Although in theory our design's bottleneck is the available BRAM, in reality we could not actually test its efficiency, because of the implementation results. Nevertheless the timing results were promising, though normally we would avoid turning our Producer into pipeline, but out of necessity, in order to maintain a short clock cycle in our design, we had to modify all our Golomb Engine modules into pipeline.

Since our main concern turned out to be the performance of the designing tool, we decided to relocate our design to two contemporary FPGAs, XCZU9EG and XC7Z045, and to use VIVADO instead of ISE. VIVADO has in essence replaced ISE over the last years. Considering that our design can be perceived as a black box, capable of finding Golomb Rulers, the relocation was straightforward. The results from both FPGAs were outstanding, since we had optimal utilization, in combination with a great clock frequency. Moreover we had a potential speedup of 2x for XCZU9EG, regarding OGR(28), even though it had to compete four Virtex-6.

The most challenging part of this thesis definitely was to synchronize four different designs, in four FPGAs, that communicate in a circular connection, and to find solutions in order to bypass any difficulties or problems that the Convey HC-2ex platform presented, such as the limitations of the simulation tool. Although in the final results the inadequacy of ISE is obvious, in general a pipeline version of the Golomb Engine, that implements a Producer-Consumer model, is promising, assuming the necessary upgrades on the designing tools that HC-2ex utilizes, or, if that is not possible, the shifting of research on modern FPGAs, that deliver optimal results.

6.2 Future Work

The work presented here can be advanced with several propositions that, due to lack of time, have been left for the future. First of all, despite the disappointing implementation results from Convey HC-2ex, the presented design can be improved by exploiting better the memory interface and the memory controllers provided to the user, in order to reduce the delay cost each time we share a stub to another FPGA.

Moreover, and since the advancement of VIVADO over ISE is clear, the same idea of one Producer with multiple Consumers, can be implemented in modern schemes that utilize multiple FPGAs and can exploit the dominance of the new designing tool. Furthermore, the single cycle version of Golomb Engine can be tested, or a hybrid version with pipelined Consumers and a single cycle Producer, that would give better control on the stubs that have been consumed, and simplify the back-up system.

References

[1] Moffet A. "Minimum-redundancy linear arrays". In: *IEEE Transactions on Antennas and Propagation* Vol.16, No 2, p.172–175 (1968). URL: https://ieeexplore.ieee.org/document/1139138.

[2] Apostolos Emmanouelides Apostolos Dollas Euripides Sotiriades. *Architecture and Design of GE1, a FCCM for Golomb Ruler Derivation*. 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.3974&rep=rep1&type=pdf.

[3] David McCracken Apostolos Dollas William T. Rankin. *A New Algorithm for Golomb Ruler Derivation and Proof of the 19 Mark Ruler*. 1998. URL: https://ieeexplore.ieee.org/document/651068.

[4] W. C. Babcock. *Intermodulation Interference in Radio Systems*. 1953. URL: https://ieeexplore.ieee.org/document/6768265.

[5] Gupta N. Bansal S. Singh A. K. "Optimal Golomb Ruler Sequences Generation for Optical WDM Systems: A Novel Parallel Hybrid Multi-objective Bat Algorithm". In: *Journal of The Institution of Engineers (India)* Series B, 98(1), 43–64 (2016). URL: https://link.springer.com/article/10.1007/s40031-016-0249-1.

[6] Brian Beavers. "Golomb Rulers and Graceful Graphs". In: (2015). URL: https://www.researchgate.net/publication/228807894_GOLOMB_RULERS_AND_GRACEFUL_GRAPHS.

[7] Ribes J. Biraud F. Blum E. "On optimum synthetic linear arrays with application to radioastronomy". In: *IEEE Transactions on Antennas and Propagation* Vol.22, No 1, p.108–109 (1974). URL: https://ieeexplore.ieee.org/document/1140732.

[8] S. W. Bloom G. S. Golomb. "Applications of numbered undirected graphs". In: *Proceedings of the IEEE* Vol.65, No 4, p.562–570 (1977). URL: https://ieeexplore.ieee.org/document/1454786.

[9] *Convey Personality Development Kit Reference Manual*. Version 5.2. Convey Computer, 2012. URL: https://wikis.ece.iastate.edu/cpre584/images/6/64/ConveyPDKReferenceManual.pdf.

[10] *Convey Programmers Guide*. Version 1.8. Convey Computer, 2010. URL: https://wikis.ece.iastate.edu/cpre584/images/e/e7/ConveyProgrammersGuide.pdf.

[11] *Convey Reference Manual*. Version 1.1. Convey Computer, 2012.

[12] A. K. Dewdney. "Computer Recreations". In: *Scientific American* 253, No.6. pp 16-29 (1985). URL: https://www.jstor.org/stable/24967868.

[14] Peter Athanas Euripides Sotiriades Apostolos Dollas. *Hardware-Software Codesign and Parallel Implementation of a Golomb Ruler Derivation Engine*. 2001. URL: https://www.researchgate.net/publication/3888248_Hardware-software_codesign_and_parallel_implementation_of_a_Golomb_ruler_derivation_engine.

[15] Singh A. K. Gupta N. Jain P. Bansal S. "Golomb Ruler Sequences Optimization for FWM Crosstalk Reduction: Multi-population Hybrid Flower Pollination Algorithm". In: (2015). URL: https://www.researchgate.net/publication/280571768.

[16] Lennerstad H. Klonowska K. Lundberg L. "Using Golomb rulers for optimal recovery schemes in fault tolerant distributed computing". In: (2003). URL: https://www.researchgate.net/publication/220952351_Using_Golomb_Rulers_for_Optimal_Recovery_Schemes_in_Fault_Tolerant_Distributed_Computing.

[17] Sarwate D. V. Lam A. W. "On optimum time-hopping patterns". In: *IEEE Transactions on Communications* Vol.36, No 3, p.380-382 (1988). URL: https://ieeexplore.ieee.org/document/1464.

[18] Tollis I. G. Linebarger D. A. Sudborough I. H. "Difference bases and sparse sensor arrays". In: *IEEE Transactions on Information Theory* Vol.39, No 2, p.716–721 (1993). URL: https://ieeexplore.ieee.org/abstract/document/212309.

[20] Klonowska K. Gustafsson G. Lundberg L. Lennerstad H. *Using Optimal Golomb Rulers for Minimizing Collisions in Closed Hashing*. 2004. URL: https://link.springer.com/chapter/10.1007/978-3-540-30502-6_11.

[21] Papakonstantinou P. A. Meyer C. "On the complexity of constructing Golomb Rulers". In: *Discrete Applied Mathematics* 157(4), 738–748 (2009). URL: doi:10.1016/j.dam.2008.07.006.

[22] Kiriakos Simon Mountakis. "Parallel Search for Optimal Golomb Rulers". 2010. URL: http://artemis.library.tuc.gr/DT2012-0120/DT2012-0120.pdf.

[23] Abreu G. Oshiga O. *Analysis of wireless localization with Golomb-optimized multipoint ranging*. 2014. URL: https://ieeexplore.ieee.org/abstract/document/6933336.

[24] Apostolos Dollas Pavlos Malakonakis Euripides Sotiriades. *GE3: a single FPGA client-server architecture for Golomb ruler derivation*. 2010. URL: https://ieeexplore.ieee.org/document/5681461.

[26] Bernstein A. Robinson J. "A class of binary recurrent codes with limited error propagation". In: *IEEE Transactions on Information Theory* Vol.13, No 1, p.106–113 (1967). URL: https://ieeexplore.ieee.org/document/1053951.

[27] Martin J.H.J. Smyth M.S. "x Ray Crystallography". In: *BMJ Journals* Molecular Pathology, Vol.53, Issue 1 (2000). URL: https://mp.bmj.com/content/molpath/53/1/8.full.pdf.

[28] Gary Lebby Stephen W. Soliday Abdollah Homaifar. *Genetic Algorithm Approach to the Search for Golomb Rulers*. 2000. URL: https://www.researchgate.net/publication/2532838_Genetic_Algorithm_Approach_To_The_Search_For_Golomb_Rulers.

[29] Qingshan Tang. "Methodology of Multi-FPGA Prototyping Platform Generation". 2015. URL: https://tel.archives-ouvertes.fr/tel-01256510/document.

[30] Riznyk V. "Application of Perfect Distribution Phenomenon for Acoustics and Music". In: p.581–585 (2000). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.541.8715&rep=rep1&type=pdf.

[31] Riznyk V. "Application of the Golden Numerical Rings for Configure Acoustic Systems of Fine Resolution". In: *Acta Phys. Polonica* Vol.119, No 6-A, p.1046-1049 (2011). URL: http://przyrbwn.icm.edu.pl/APP/PDF/119/a119z6Ap30.pdf.

[34] Rankin W.T. "Optimal Golomb Rulers : An Exhaustive Parallel Search Implementation". 1993. URL: https://pdfs.semanticscholar.org/025c/45dc22bf709e1ccfd1eb55f8cdf48fb35089.pdf.

[35] Baljeet Kaur Yogita Wadhwa Parvinder Kaur. "Golomb Ruler Sequence Generation and Optimization using Modified Firefly Algorithm". In: *SSRG International Journal of Electronics and Communication Engineering* Vol. 1, Issue 5 (2014). URL: https://pdfs.semanticscholar.org/08b3/6fcf33e94f543b3882087913e54ad8179b1b.pdf.

[38] Παύλος Μαλακωνάκης. "Μελέτη, Σχεδιασμός και Υλοποίηση σε Αναδιατασσόμενη Λογική, Αρχιτεκτονικής για τον παράλληλο Υπολογισμό Βέλτιστων Κανόνων Golomb (GE3)". 2009. URL: http://dias.library.tuc.gr/view/14993.

External Links

[13] *distributed net*. URL: http://www.distributed.net/OGR.

[19] *LogiCORE IP BlockMemory Generator v7.1*. URL: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v7_1/blk_mem_gen_ds512.pdf.

[25] *Possibly Optimal Golomb Rulers Calculated for 160 to 40,000 Marks*. URL: http://cube20.org/golomb/.

[32] *Virtex-6 Family Overview*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[33] *Virtex-6 FPGA Memory Resources*. URL: https://www.xilinx.com/support/documentation/user_guides/ug363.pdf.

[36] *ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC*. URL: https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf.

[37] *Zynq UltraScale+ MPSoC Data Sheet:Overview*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf.