Parallella Epiphany Cluster

Technical Report

Aviv Burshtein

300383387

aviv.bur@campus.technion.ac.il

aviv.bur@gmail.com

Instructor: Mr Oz Shmueli

Academic Supervisor: Prof. Tsahi Birk

Table of Contents

Abstract

As the race to make faster and stronger CPUs slowed to a pace, recent years marked a large leap in parallel computation platform development. Parallel applications rapidly become more and more popular, while large scale multicore computers remain expensive and difficult to manage. The Parallella platform tries to fill this void, presenting low budget, fast and efficient computational power. The question is, however, is it scalable? Will it be capable of handling future massively parallel applications?

This paper presents an attempt to use multiple Parallella boards to construct a scalable cluster, that makes use of Parallella's superb communication networks and computational power. This task is not easy, as the open source platform does not fully support it yet - both from the hardware and software points of view. This paper discusses the main problems in constructing such a cluster, the possible layouts and the limitations. Several attempts have been made and conclusions have been documented accordingly. Finally, the paper presents a user manual to guide users in recreating an Epiphany level Parallella cluster - from manufacturing PCB connectors to running a parallel application on the cluster.

Note that the cluster uses 2015.1 ESDK with 7020 ZYNQ and a corresponding FPGA build version. An attempt to apply it to other ESDK versions or builds might lead to driver mismatches, and would probably require some adjustments to be made.

Project Goals

1. *Study and setup the Parallella environment.*
2. *Design and produce a PCB e-link connector for dual board connection.*
3. *Find a way to allow Epiphany level communication between boards.*
4. *Write an example program that runs on an Epiphany cluster.*

Parallella Platform

General Overview

The Parallella board is an "affordable, energy efficient, high performance, credit card sized computer" [1] that aims to provide a platform for developing and implementing high performance parallel processing. The 18-core (16-Epiphany and 2 ARM cores) board can reach 32 GFLOPS using only about 5 Watts. Parallella uses a customized ARM implementation of Linux (Ubuntu 14.04) and runs it on the ZYNQ platform, which comprises of 2 ARM cores and FPGA logic. The Parallella has a 1-Gbps Ethernet port allowing a large amount of information to be passed quickly over the internet. The aim of creating the Parallella board was to make parallel computing more accessible by creating an affordable, open-source, and open-access platform. The price of a Parallella board starts at $99. The board's central platform is the Epiphany. It comprises of 16 high performance RISC cores (denoted e-cores) and 3 communication networks.

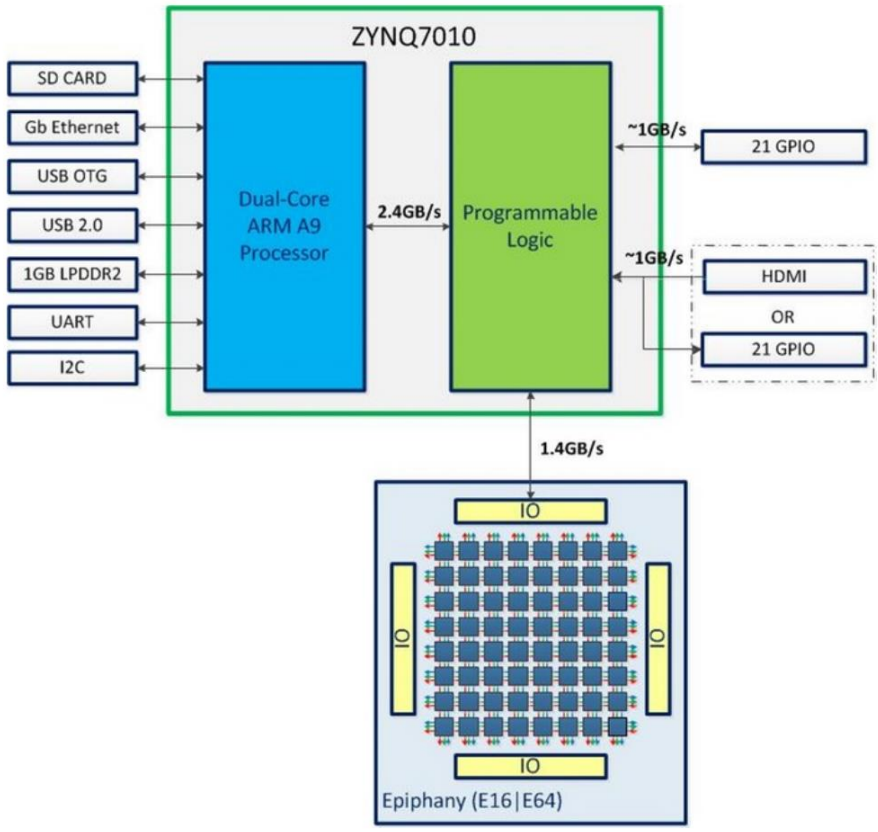

Figure 1 - Parallella general blocks [1]

Specifications

- C/C++ and OpenCL programmable.
- 32-bit IEEE floating point support.
- 512KB on-chip distributed shared memory.
- 32 independent DMA channels.
- Up to 1GHz operating frequency.
- 32 GFLOPS peak performance.
- 512 GB/s local memory bandwidth: The access speed of each RISC core's local RAM.
- 64 GB/s Network-On-Chip bisection bandwidth: The speed at which message passing takes place on the chip.
- 8 GB/s off-chip bandwidth: The performance of communicating off chip.
- 1.5ns network per-hop latency.
- <2 Watt maximum chip power consumption: The power consumption of the Epiphany processor, combined with the rest of the board is 5 Watt.
- Two 1.4 GB E-link connectors for off-board Epiphany communication.
- 48 GPIO pins.

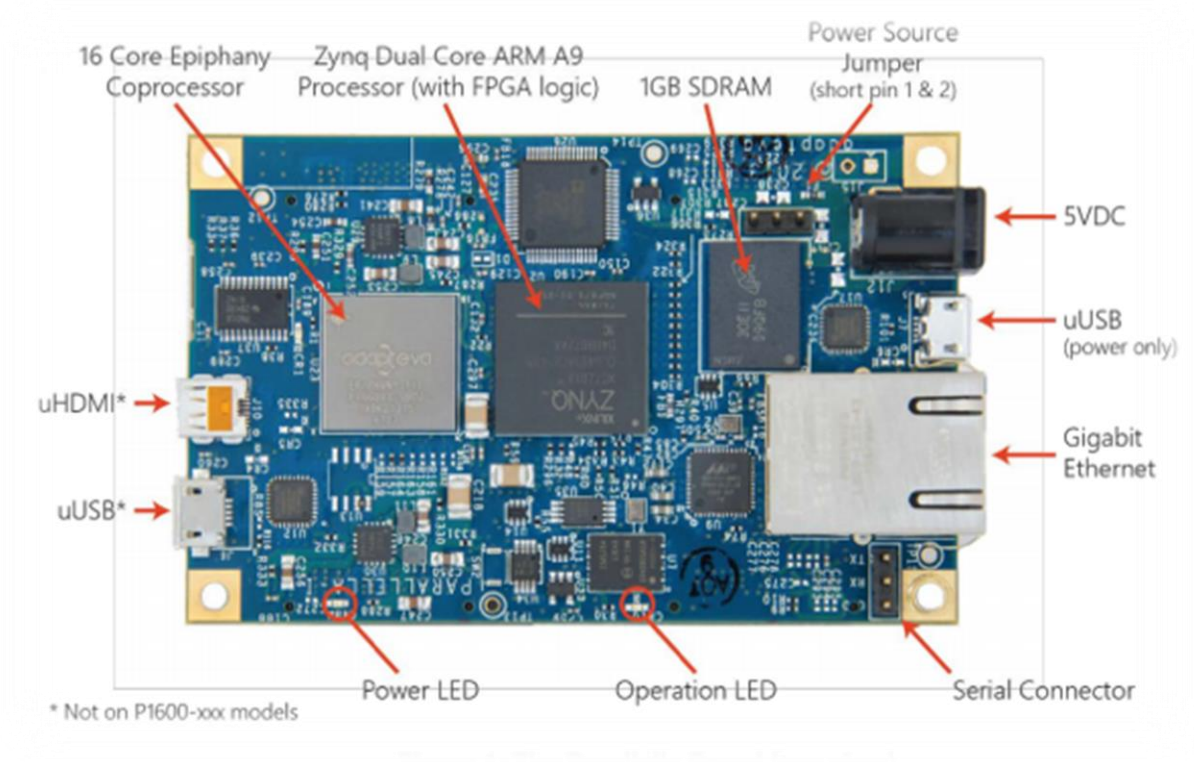

Figure 2 - Parallella top view [1]

Epiphany

The Epiphany architecture defines a multicore, scalable, shared memory, parallel computing fabric. It consists of a 2D array of compute nodes connected by a low latency mesh network on chip. Capabilities and important specs:

• A superscalar, floating point RISC CPU in each mesh node that can execute two floating point operations and a 64 bit memory load operation on every clock cycle.
• Local memory in each mesh node that provides 32 Bytes/cycle of sustained bandwidth and is part of a distributed, shared memory system.
• Multicore communication infrastructure in each node that includes a network interface, a multichannel DMA engine, multicore address decoder, and network monitor.
• A 2D mesh network that supports on-chip node-to-node communication latencies in nanoseconds, with zero startup overhead.

Epiphany cores address other cores by their addresses in the shared space as shown in the next image:

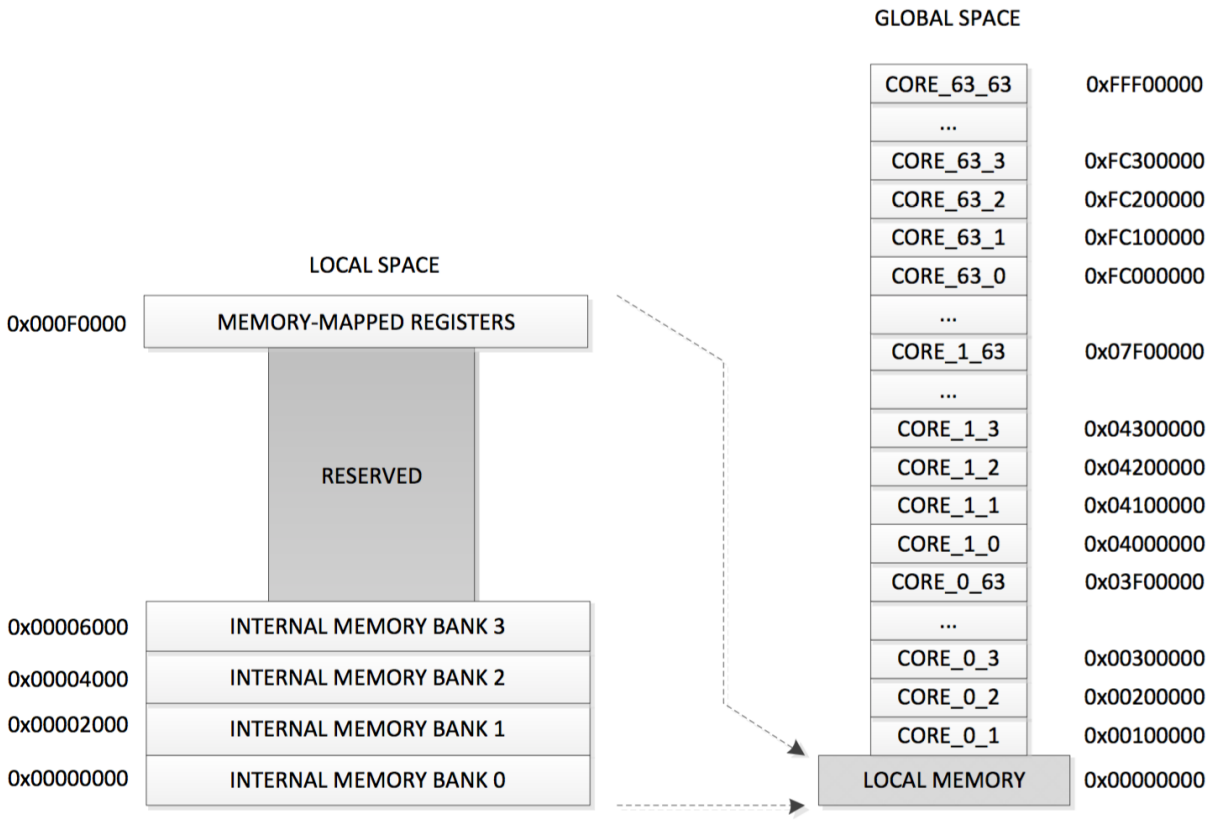

Figure 3- Epiphany global memory map [2]

Note that each of the e-cores address themselves as core (0,0), and therefore the maximum amount of cores possible in a cluster theoretically is 4095.

Epiphany routing and communication:

Every e-core has a routing interface as described below:

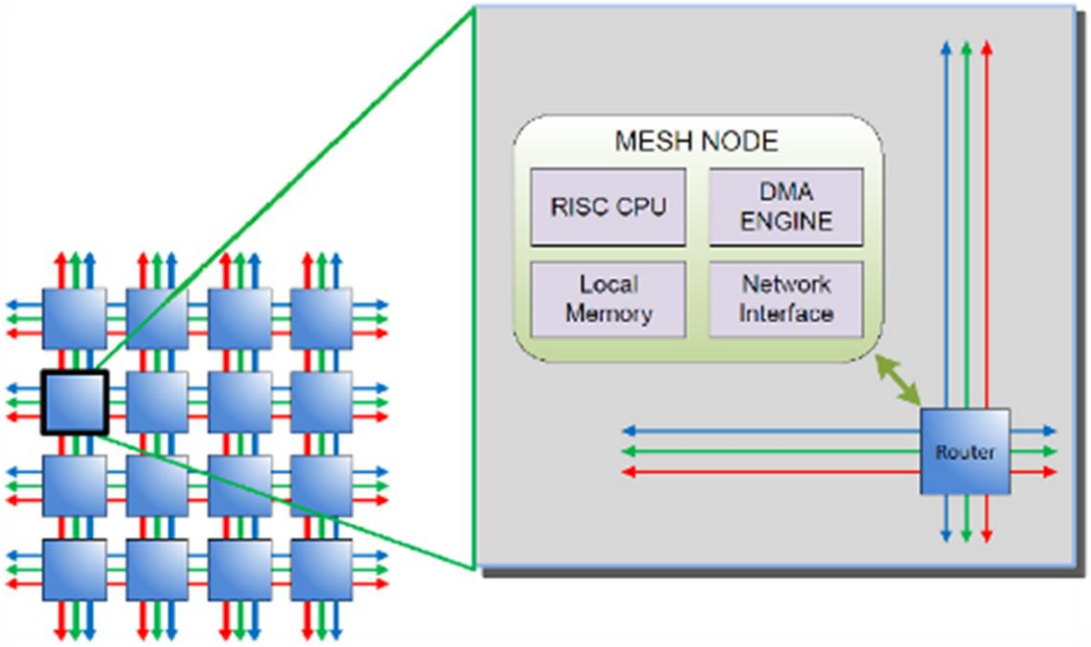

Figure 4- Epiphany routing interface [2]

The routing occurs over three different grids:

- cMesh - Write on-chip.
- xMesh - Write off-chip.
- rMesh - Read.

In all grids routing of requests is done column-wise (within a row) and then row-wise (within a column) according to the core-id of the destination core (first 6 bits from the MSB).

Parallella Memory Spaces

Parallella uses memory mapped IO for communication between different components on the board. The ZYNQ routes requests to the local Epiphany board, the FPGA logic, the SDRAM and the flash drive according to a 4GB (32 bits) memory map, illustrated in the next image.

Address Start	Address End	Size	Function	Note
0x0010_0000	0x3FFF_FFFF	1GB	DRAM	Accessible to all interconnect masters
0x4000_0000	0x7FFF_FFFF	1GB	PL	Custom logic address range
0x8000_0000	0xBFFF_FFFF	1GB	PL	Epiphany address range
0xFC00_0000	0xFCFF_FFFF	16MB	FLASH	Quad-SPI linear address for linear mode
0xFFFC_0000	0xFFFF_FFFF	252KB	OCM	OCM upper address range

Figure 5 - Parallella memory map [1]

Epiphany, on the other hand, has its own 32 bits virtual memory space, comprised of a large mesh of Epiphany boards (from its point of view). Parallella uses a slice of that space as shared RAM space from Epiphany point of view. Epiphany read/write requests to these specific addresses are routed east out of Epiphany and then redirected to the matching actual RAM addresses in the ARM point of view. The image below demonstrates the Parallella memory map:

Figure 6- Parallella memory map. The black square indicates the physical Epiphany cores on the board.

Note that Epiphany is only connected to the ARM from the east connector. Adapteva announced that Epiphany 3 will have inner routing configuration registers that will allow custom routing manipulations to support use of other parts of the Epiphany memory space, but as of today shared RAM can only be placed east of the Epiphany. This disadvantage also stands in the way of scaling of Epiphany clusters as will be elaborated on later in this document.

Parallella programming

While Parallella supports a variety of programming languages (C, C++, OpenCL), this project only covered C applications. The code is written in C and uses additional library functions supplied by Adapteva is their ESDK library. These library functions support the required Epiphany interface both from the ZYNQ point of view and from within the e-cores. The code is written in two different sections - the code that runs on the ZYNQ and the code that is loaded and runs on the Epiphany platform. E-hal and E-lib libraries support interface operations from the ZYNQ and Epiphany specific operations from within the e-cores.

The E-cores code is compiled twice: first to .elf format and then to .srec format (the format that is compatible with the e-cores). The code is loaded individually to the different cores in run time by the program running on the ZYNQ.

The Epiphany Cluster Dream

Motivation

Adapteva's Parallella Kickstarter campaign carries the title "Parallella: A Supercomputer for Everyone" [6]. This idea is indeed one of the most appealing parts of the Parallella platform.
The platform is available for only $99 at the time of writing, so it is affordable for many. But is it a supercomputer? That question is difficult to answer as a supercomputer is not very well defined, but it is safe to say that in a few years 16 cores might not be a significant computational power. Therefore, the true appeal of Parallella for many users is its scalability. The large, mostly unused memory space it possesses and the fast E-link and Ethernet connectors make it a good candidate for a large scalable cluster. So far many Parallella clusters have been built using communication via Ethernet, but this kind of application does not make full use of Parallella's capabilities.
This raises a natural question - Why not make use of the fast E-links and read-write communication between the boards?
This could theoretically allow an actual workgroup of as many as 4095 cores (!) with read-write communication between them. Appealing as this may sound, a record of a cluster of that sort is nowhere to be found. This project is a first step in achieving this goal.

Designing the Connector

The first step in building an Epiphany cluster is figuring out how to physically connect the Parallella boards. Parallella has two E-link connectors that are routed to the Epiphany south and north ports. It is possible to connect these ports using the matching Porcupine [7] board and additional jumpers or flat cables. However, using Porcupine is slow (due to use of long, noisy jumper cables) and clumsy (uncomfortable layout, noisy connections). Instead, it is possible to use the next PCB connector, designed in the Technion Parallel Systems Laboratory. The idea of the design is to connect a south E-link connector to a north E-link connector.

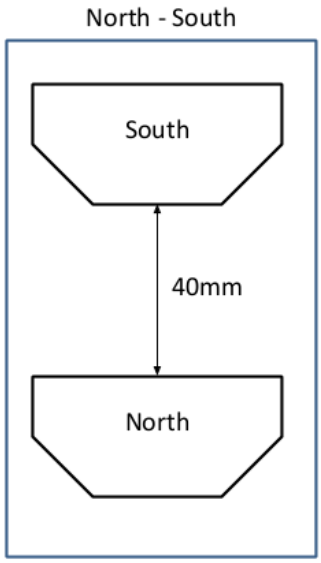

Figure 7 - Connector general layout

Main Design Issues:

- Orientation - as is shown in figure 7, the connectors on the Parallella are placed identically, rather than in a mirrored layout. Therefore, some of the wires cover greater distances on the PCB than others. The design tries to overcome this problem with minimal cross wiring.
- Noise robustness - the design uses a five layers PCB with two GND surfaces to reduce noise issues. Differential lines are wired together for the same reason. The clock wires are connected in the shortest routes possible.
- Biasing - the cluster is set up with separate power sources for the Parallella boards. Therefore, the biasing pins were wired to infinite resistors. If one wishes to use a single source in future applications he must simply solder a bypass over the infinite resistors on the PCB.
- Core-id - note that this design does not solve the current core-id setting problem. This problem is elaborated on and solved in the 'second attempt' section.

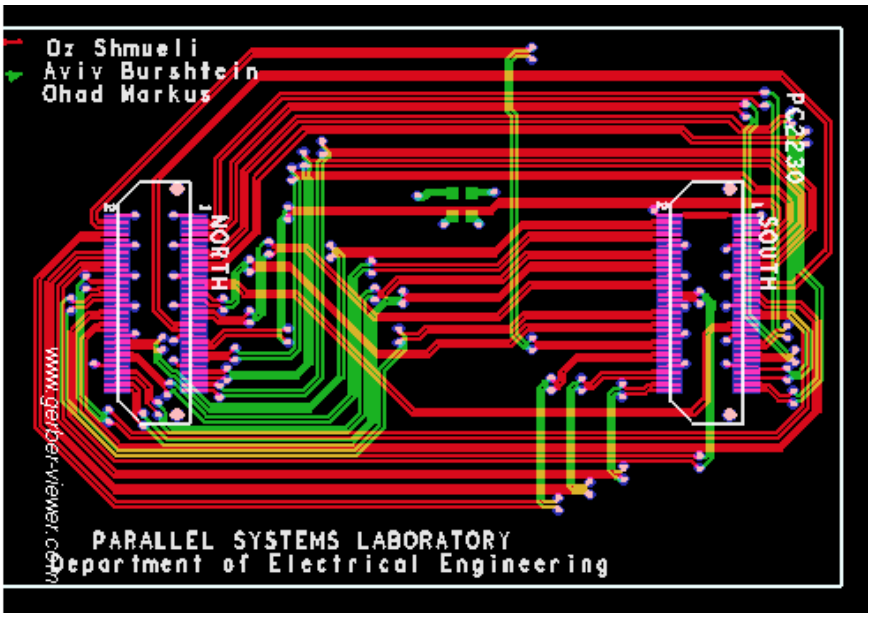

Figure 8 - Design layout

Figure 9 - Printed connector

Cluster Attempts

The process of setting up the cluster was long and fraught with obstacles. Many of which were due to misunderstanding the documentation, incorrect documentation, or lack of sufficient documentation. Here are a few of the attempts that were made, and the reasons why they failed or succeeded.

First Attempt – East and West Connections

Looking at the Parallella memory map (figure 5), it seems almost natural to set up a cluster of Epiphanies that would fill the width and height of the logical space that is reserved for Epiphany. This is definitely not as simple as it sounds.

As mentioned previously, Parallella only has north and south E-link connectors for direct Epiphany to Epiphany communication. The west bank of the Epiphany unit is disconnected, while the east bank is connected to the ZYNQ. So how do you connect boards in four directions using three connectors?

The answer is – you don't.

Theoretically, one could use GPIO pins and reroute appropriate requests from the east bank of one Epiphany, through FPGA logic, GPIO connector, second board's FPGA logic and into the east bank of another Epiphany. If that were possible, one could use two rows of Epiphany boards – a maximum of 16 boards. This creates a new set of problems – inner routing and edge cases. It means that each core must decide if a request it needs to route is bound off-board or on-board, and respectively direct it east or west.

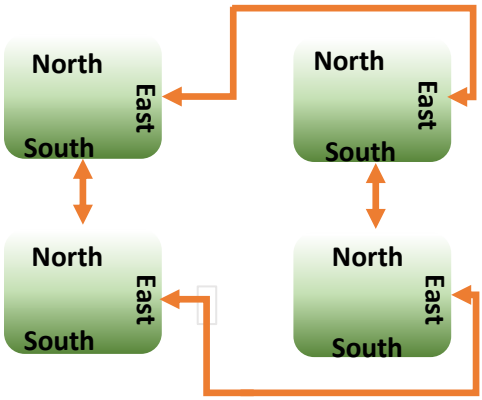

Figure 10- Theoretical two rows Parallella cluster

Theoretically, this would be solved if the routing algorithm could have been altered by the user. Note that despite the fact that [2] mentions this feature, it is only supported by Epiphany IV, while Parallella uses Epiphany III. Perhaps future Parallella might allow such a connection, but for the time being this has proven itself to be impossible.

Second Attempt

The cluster is narrowed down to a column of up to 8 Epiphany boards (128 cores).
The main disadvantage of this sort of layout (aside from the limited scalability) is the non-uniform memory access latency. This aspect should definitely be tested for future applications and taken into consideration when writing a parallel program for the cluster.
In order to set each one of the Epiphany boards in its correct location in the memory space, the core-id must be changed respectively.

The core-id problem

The core-id must be configured in two places for each board. The first is the EPIPHANY_HDF file. This file defines the interface to the Epiphany from the ARM point of view. The default configuration is set to core coordinate (32,8), which translates to oX808. As more boards are added to the system these files must be updated respectively.

```
parallella@parallella:~/Aviv_Ohad/interrupts_connected/master$ cat $EPIPHANY_HDF
// Platform description for the
// Parallella/1GB/E16G3
PLATFORM_VERSION   PARALLELLA1601
ESYS_REGS_BASE         0x70000000

NUM_CHIPS                     2
CHIP                    E16G301
CHIP_ROW                     32
CHIP_COL                      8
CHIP                    E16G301
CHIP_ROW                     36
CHIP_COL                      8

NUM_EXT_MEMS                  1
EMEM                   ext-DRAM
EMEM_BASE_ADDRESS      0x3e000000
EMEM_EPI_BASE          0x8e000000
EMEM_SIZE              0x02000000
EMEM_TYPE                    RDWR
```

Figure 11 - HDF file for the master board in a two boards configuration

This is further explained in the user guide section of this document.

The second configuration must be the actual epiphany address location from Epiphany's point of view. After reading the documentation and going through related forum discussions it is rather unclear how this is to be done. Adapteva has set up some FPGA registers and Epiphany configuration registers for this, but as of January 2016 they haven't implemented any of these in the design [8].
The only way to do this at the moment seems to be hardwiring the core-id coordinates through the PEC-POWER connector. The way the correct coordinates are worked out is specified in the user manual section of this document. It is possible to make a small jumpers interface to allow simple, flexible configuration. This is also elaborated on in the user manual section of this paper.

Final Solution

The column solution still does not allow read/write communication between the ZYNQ of the first Parallella board (the 'master' board) and distant Epiphanies. This is due to the way Epiphany routes requests. The next image illustrates an example of a read request that fails.

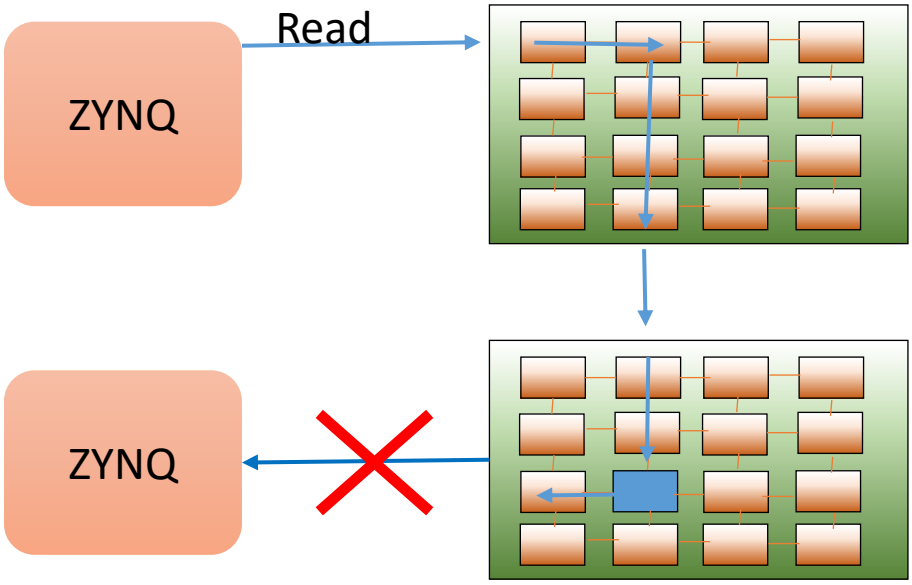

Figure 12 - Read request from master ZYNQ is routed row-wise first upon return, and stops when routed to the wrong ZYNQ.

Mailboxes

The cluster is therefore left with read/write communication capabilities only between Epiphany cores. Communication between a remote Epiphany to the master ZYNQ can be implemented using 'mailboxes' in the master Epiphany's local memory. Communication between the e-cores themselves can be done using read-writes to shared memory, and using user interrupts.

Reset_Connected_Systems

Apparently in resetting the Epiphany, the reset_system call shuts all communication through the E-links in order to save power and avoid over-heating. A separate reset function has been written to allow setting up the Epiphany environment without clock gating. This function is denoted reset_connected_system, and is located in this repository along with the rest of the example code.

In the repository you can find a code package that uses user interrupts and mailboxes to prove communication between up to 8 Parallella boards. Further explanation about setting up the cluster is available under the user manual section of this document.

Central notes about the code

- The code is based on the system interrupts test code by Yaniv Sapir (yaniv@adapteva.com).
- The master ARM code reads the mailbox from the local Epiphany, and addresses the mail box by implicit address. If the code is changed then this implicit address should also change. To retrieve the correct address after initial compilation, type:
  >> e-objdump -x ./Debug/e-int-test.master.elf | grep mailbox
- The idea of the code is that all cores in the work_group are loaded with interrupt handlers, and upon receiving a message interrupt they write their id to a mailbox located in the (0,0) core memory space. The first core in the group (0,0) then gets a timer interrupt and handles it by sending messages to all the others. The rest of the cores handle it as specified. The master ARM program then reads the mailbox and writes it to the screen.

Run example

These are screen shots of the output of the master side of the program for a two board configuration:

```
parallella@parallella:~/Aviv_Ohad/interrupts_connected/master$ ./run.sh
finished init
finished reset
platform info:  num_chips =0x2 ,emems = 0x1
hdf_ platform info:  core (0,0) id =808
 platform info:  rows =0x8 ,cols = 0x4
starting e_open
finished e_open
starting e_load
starting e_load
```

```
3. started:

msgs from(0) :
808 says:E_TIMER0_INT

msgs from(1) :
809 says:E_MESSAGE_INT

msgs from(2) :
80a says:E_MESSAGE_INT

msgs from(3) :
80b says:E_MESSAGE_INT

msgs from(4) :
848 says:E_MESSAGE_INT

msgs from(5) :
849 says:E_MESSAGE_INT

msgs from(6) :
84a says:E_MESSAGE_INT

msgs from(7) :
84b says:E_MESSAGE_INT

msgs from(8) :
888 says:E_MESSAGE_INT

msgs from(9) :
889 says:E_MESSAGE_INT

msgs from(10) :
88a says:E_MESSAGE_INT

msgs from(11) :
88b says:E_MESSAGE_INT

msgs from(12) :
8c8 says:E_MESSAGE_INT

msgs from(13) :
8c9 says:E_MESSAGE_INT

msgs from(14) :
8ca says:E_MESSAGE_INT

msgs from(15) :
8cb says:E_MESSAGE_INT

msgs from(16) :
908 says:E_MESSAGE_INT

msgs from(17) :
909 says:E_MESSAGE_INT

msgs from(18) :
90a says:E_MESSAGE_INT

msgs from(19) :
90b says:E_MESSAGE_INT
```

```
msgs from(19) :
90b says:E_MESSAGE_INT

msgs from(20) :
948 says:E_MESSAGE_INT

msgs from(21) :
949 says:E_MESSAGE_INT

msgs from(22) :
94a says:E_MESSAGE_INT

msgs from(23) :
94b says:E_MESSAGE_INT

msgs from(24) :
988 says:E_TIMER0_INT

msgs from(25) :
989 says:E_MESSAGE_INT

msgs from(26) :
98a says:E_MESSAGE_INT

msgs from(27) :
98b says:E_MESSAGE_INT

msgs from(28) :
9c8 says:E_MESSAGE_INT

msgs from(29) :
9c9 says:E_MESSAGE_INT

msgs from(30) :
9ca says:E_MESSAGE_INT

msgs from(31) :
9cb says:E_MESSAGE_INT
```

Figure 13 - Screen shot of a parallel program run on a two Parallella Epiphany cluster

Results

The cluster constructed in this project holds significant computational power, alongside some major drawbacks. Some of the features and abilities of this system are clear from the tests that have been conducted, while others still need to be measured and verified.

The system is a high speed, scalable, multicore processing unit, with fast communication and good computational capabilities. The fantastic e-core computational capabilities spectrum makes this system extremely efficient for highly parallel computational programs. As long as the majority of the software running time is spent processing separate parallel data, this system is highly efficient. When passing information between cores, the system also operates quite well thanks to the strong Epiphany NOC. However, access latency may vary significantly between distant and neighboring cores. As off-chip communication is clearly slower, the system functions better for programs with high spatial locality and minimal off-chip communication. The non-uniform access time certainly requires further testing and measuring for better understanding of the system capabilities.

Communication with the ARM cores is significantly less efficient and less user friendly. The use of mailboxes simplifies this procedure and perhaps should be expanded in the future with library functions. Note that this feature might make programming on the platform easier, but would not change the high latency overhead and synchronization problems for ARM to distant Epiphany communication.

Overall the current cluster configuration does not support ARM off chip communication well and is much more effective for Epiphany-centric computational schemes.

User Manual

Setting up the environment

- Download this disk image : https://github.com/parallella/pubuntu/releases/tag/pubuntu-14.04-esdk.2015.1-20150130
  Make sure you choose the file that matches your board's ZYNQ version (7010 or 7020).
- Write the image onto a micro-USB card. You may use this software to do this: https://sourceforge.net/projects/win32diskimage/
- Connect to your Parallella using SSH. You may follow this Parallella tutorial (page 3): http://www.adapteva.com/wp-content/uploads/Parallella-Quick-Start-Guide_rev3.pdf
  I recommend using this software for your SSH connection: http://mobaxterm.mobatek.net/download-home-edition.html

Connecting multiple Parallella Boards

You may connect two or more boards via their north and south e-link connectors in one of two ways, both described below. The main advantage of the Porcupine connection is how comfortable and configurable it is, while the main disadvantage is it using large, slow and unreliable jumper cables. The main advantage of using my design is how small and fast it is, while the main disadvantage is the necessity of independently manufactured PCB.

1. Use Porcupine boards, and make your own connectors:

- Order a porcupine board for each Parallella: http://www.digikey.com/product-detail/en/ACC1600-01/1554-1003-ND/5048176
- Order flat cables and connectors: http://www.digikey.ca/product-search/en?keywords=MC30L-5-ND
  and get some jumper cables from the nearest electronics shop.
- Order flat cable connectors, two for each board you wish to connect: http://www.digikey.ca/product-search/en?keywords=WM14330-ND

- Connect both ends of the cable to the connectors you purchased:

Figure 14 – Flat cable connector

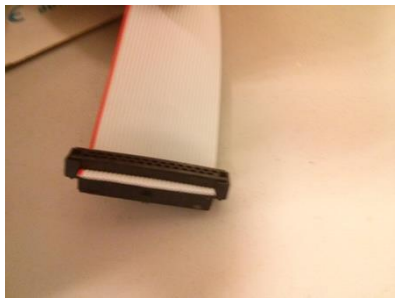

Figure 15 – Flat cable connector

You must make two of these for every pair of Parallella boards.

- Connect Parallella boards as follows:

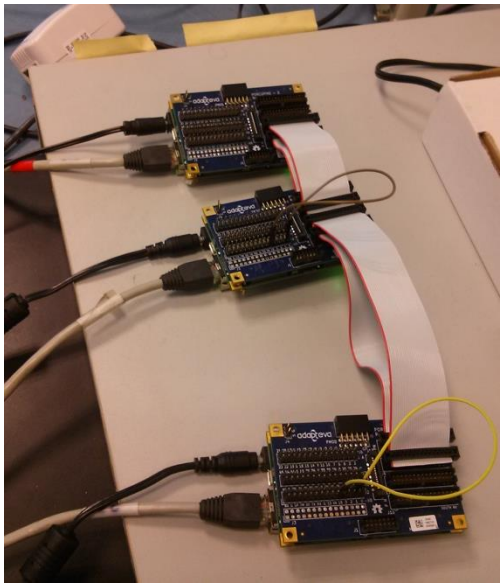

Figure 16 –Three boards flat cable connection

- Hardwire core-id:
  The core-id can be reconfigured by hardwiring the PEC_POWER connector.
  The pins representing the core-id are indicated in this image:

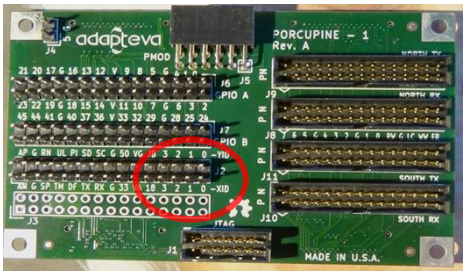

Figure 17 – core-id pins in Porcupine board

The core-id vector is 6 bits for Yid (columns) and 6 bits for Xid (rows). The last two bits of both are always 00 for the upper left core in each board (row and column are multiples of 4) and therefore 8 bits are enough to represent the address. The default core-id coordinates are (32,8), which translate to 808 in hexadecimal representation, and are represented as follows:

Yid 3	Yid 2	Yid 1	Yid 0	Xid 3	Xid 2	Xid 1	Xid 0
0	1	0	0	0	1	0	0

Figure 18 – default core-id vector

In order to set the core-id, jumpers must be used to connect GND or 1.8 Volts outputs to the correct coordinates. For example, coordinates (36,8) would translate to 908 in hexadecimal representation, and would be wired like this:

Figure 19 – Hardwiring (36,8) core-id coordinates

2. Produce PCB connectors and jumper sockets using my design:

- Download the connector design files from here.
- Send to reproduction in a fab. I recommend: https://www.itead.cc/
- Order connectors: http://www.toby.co.uk/content/catalogue/products.aspx?series=BTH-xxx-01-x-D-A-xx
  Make sure you order some spares for hardwiring core-ids.
- Solder the connectors to the PCB:

Figure 20 – Custom Parallella PCB connector

- Connect as follows, preferably under static pressure or bolted to a surface.

Figure 21 – A 3-board cluster with PCB connectors

- Hardwire PCB: It is possible to solder the connectors themselves to a constant core-id, and simply place the hardwired connector in the PEC_POWER socket. There is a preferable solution that is more flexible, use this design and connect the relevant pins in the PEC_POWER socket to jumper pins.

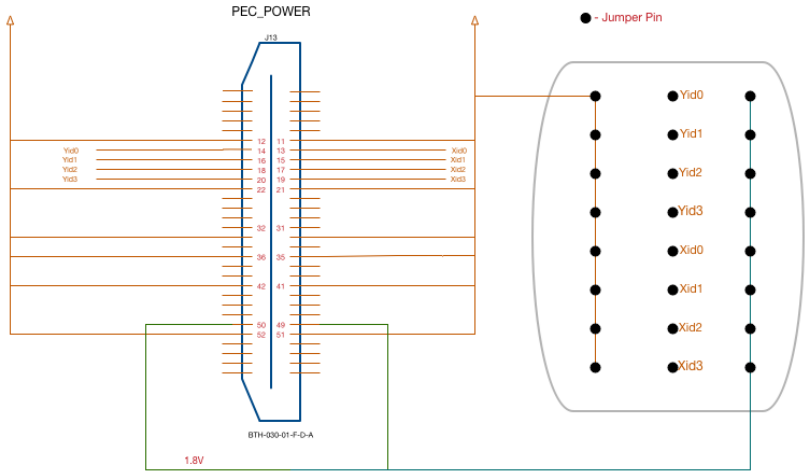

Figure 22 – core-id PEC-POWER configuration design

Here are two examples:

Figure 23

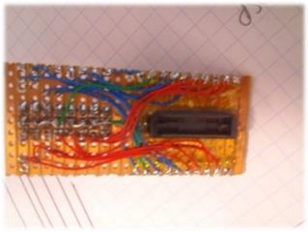

Figure 24

- If anyone could pick this up and add a PCB design for this I would gladly add it to the files here!

- When connecting the boards connect the correct pins for the core-id according to the explanation in the previous section.

Set up and run example code

The code uses the 2015 ESDK interrupts library to prove basic communication between the different Parallella boards. The communication is in the Epiphany level, meaning there is no read-

write connection from a ZYNQ to a distant Epiphany, and communication between OS and distant Epiphany boards is done via mailboxes in the local Epiphany board.

- Download the code folder from this link and unzip it.
- Copy the folder to your Parallella machines.
- Set up your HDF file using the edit_hdf script:
  From the first board (with a single slave board):
  Sudo ./edit_hdf.bash master 1
  Here the second argument represents the number of slave boards in the system.
  From a single slave board:
  Sudo ./edit_hdf.bash slave 1
  In this example the second argument represents the current slave board number.
  Note that if you want to use more than two boards you must change the *maxcorenum* macro in the files *int-test.c* and *e-int-test.master.c* to match the amount of cores you need, and use edit_hdf accordingly.
  For example, if you wish to use three boards in total (1 master and 2 slaves), you should set *maxcorenum* to 48, type "**Sudo ./edit_hdf.bash slave 1**" in the first slave Parallella, and type "**Sudo ./edit_hdf.bash slave 2**" in the second slave Parallella. In the master Parallella type "**Sudo ./edit_hdf.bash master 2**".
- Run ./build.sh from the slave directory on all slave boards and run it from the master directory on the master board.
- Run ./run.sh from the slave directory on all slave boards and then run it from the master directory on the master board.
  The output should be the text in each board's space in the mailbox (within the first core's memory space).
  Note that it might be wise to set up each core on its own at first and run "hello world" on it with the new core-id configuration.

Future Development Ideas

- Library patch for connected boards:
  It is possible to patch up the ESDK library files (epiphany_hal.c mainly) to support system reset for a connected system. This can be copied from my_h.c code in this repository.

- Parallella level cluster of column meshes, using Ethernet:
  Connecting multiple Epiphany clusters as presented in this paper to make a fully scalable cluster of 8 Parallella clusters, connected via Ethernet.

- Change FPGA to support core-id changes and make more space for Epiphany column clusters:
  Edit the routing in the FPGA build files to enable more than 8 Epiphany boards in a column. Also support configuring PEC_POWER output and FPGA registers to allow core-id configuration.

- Automate mailbox system as library function:
  Support read/write communication to remote Epiphany boards from master ARM using an encapsulated library function that operates using a mailbox scheme.

- Add FPGA support of read/write routing through GPIO connector:
  Close the routing loop through the Ethernet connection back to the master board.

References

1. Parallella Reference Manual
2. Epiphany Architecture Reference Manual
3. Epiphany SDK Reference Manual
4. Epiphany-III Datasheet
5. Epiphany-IV Datasheet:
6. Parallella Kickstarter project
7. Porcupine board
8. Forum Discussion

Software Repositories:

9. Parallella Hardware and Software Repository
10. Epiphany SDK Software Repository
11. SD Card Images

Credits and thanks

- Ela Gluzman (ela@ee.technion.ac.il), for her wonderful assistance in designing the PCB.
- Bruria Zohar (bruria@ee.technion.ac.il), for her superb soldering and improvisations.
- Peter (Parallella forum profile), for great guidance and advice, despite not knowing me.
- Yaniv Sapir (yaniv@adapteva.com), for his great interrupts test code.