Network-on-Chip Partitioning and Implementation on Multi-FPGA system

Saurabh Agrawal (123076007)

Supervisor: Prof. Sachin B. Patkar,
Dept. of EE, IIT Bombay

June 26, 2015

Acknowledgement

- **Prof. Sachin B. Patkar**, for giving me opportunity to present this idea and his consistent suggestion.
- **Vinay B.Y. Kumar**, PhD student Electrical Engineering working under Prof. Sachin Patkar for his continuous guidance and support throughout my M-Tech Project.
- **Pinal J. Engineer**, for his valuable suggestions for the presentation.
- **Yatish Turakhia**, the core idea has been inspired from his DDP thesis work.
- **Shaishav Shah**, for providing me his application design for testing my work.
- HPC lab, WEL and VLSI lab for letting me use their resources.

Aim

- Investigation of communication interface links in Multi FPGA network of chips.
- Development of Quasi-SERDES physical communication link module compatible with CONNECT NoC.
- Automated Network-on-Chip (NoC) partitioning.
- Investigation and development of stand-alone system for FPGA / CPLD programming and support for hardware / software co-design application using Raspberry Pi.

Introduction

- Physical interconnect network is a bottleneck when mapping applications for Multi-FPGA Network-of-Chips
- Applications for Multi-FPGA system needs isolation from the communication network
- NoC partitioning and implementation using multiple FPGAs will help designing larger systems
- Challenge is the automation of NoC partitioning
- Interconnect between FPGA has to be high speed and physically realizable
- Implementation of DUT PG-LDPC (7,3) over partitioned and un-partitioned CONNECT generated 4×4 Mesh NoC
- Utilizing Raspberry Pi with DE0-Nano for hardware-software co-design

Need of NoC

- Isolation of processing nodes and communication network
- NoC utilizes the computer networking fundamentals and provides a communication infrastructure for designing System-on-Chip (SoC)
- Trade-off between
  1. Point to point transmission - high speed and complex
  2. Shared bus data transmission - Low speed and simpler
- Ease of verification and logic error determination
- By connecting computing nodes to PE of NoC, routing congestion is taken care by NoC
- Useful to design flexible and scalable architecture implementation

Need of NoC Partitioning

- With the increase in number of processing cores, on-chip block RAMs, specialized IP cores etc.., packing all these onto single die using traditional interconnection techniques will no longer be feasible
- For multi FPGA network-of-chips interconnect is the bottleneck
- Partitioning utilizes networking fundamental and isolates the application layers and physical link layer by introducing partitioning cuts at data link layer
- Now all the modifications and partitioning take place in data link layer making the application design and mapping independent of data exchange methods

NoC Building Blocks

- **Links**
  The elements which physically connect the nodes and actually implement the communication are called as Links.
- **Router**
  The decentralized logic which implements the communication protocol is called Router.
- **Network Adapter (NA) or Network Interface (NI)**
  The block which makes the logic connection between IP cores and network is called as Network Interface or Network Adapter.

Router and Flit

Routers

- The critical part of NoC.
- Receives the data packet from the adjacent router or from the attached Processing Element.
- Redirects packet to the Processing Element or the next adjacent router.

Flit

- The physical unit, which is the minimum amount of data that is transmitted in one link transaction.
- Atomic unit that forms *packets* and *streams*.

Flow Control: Characterizes the packet movement across NoC channel.

- *Centralized Control:* Routing decisions made globally and applied to all nodes.
- *Distributed Control:* Each router makes decision locally.

Routing Algorithm: Logic that selects one output port to forward a packet that arrives at the router input.

- *Deterministic Routing:* A packet always uses the same path between any set of nodes.
- *Adaptive Routing:* Alternate path between two nodes may be used if the original path or local link is congested.
- *Static Routing:* Paths between cores are defined at compilation time.
- *Dynamic Routing:* Paths between cores are defined during run-time.

Arbitration: Logic that selects one input port, when multiple packets arrive at the router simultaneously requesting the same output port.

- *static* (fixed), or *dynamic* (variable)
- *distributed* (one per port) or *centralized* (one per router).

Buffering: The strategy used to store information in the router during congestion in the network, and the packet cannot be forwarded immediately to the correct port **Routing Algorithm:** Logic that selects one output port to forward a packet that arrives at the router input.

- *Single Buffer per Router*
- *Single Buffer per port*

- **Deadlock:** The condition when network resources are fully occupied and waiting for each other to be released for communication.
- **Virtual Channel (VC):** VC uses the concept of multiplexing a single physical channel over numbers of logically separate channels with individual and independent buffer queues.
- The NoC used involves *Distributed Flow Control* approach and *Virtual Channels*. NoC uses *XY routing*, which is *deterministic* and *static* routing, *Round Robin* arbitration, which is *distributed* and *dynamic* arbitration approach, and multiple buffers (typically 8) at each input port

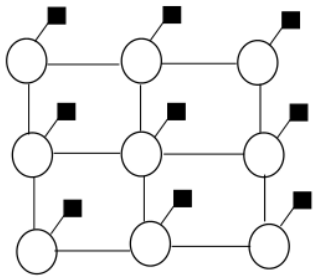
NoC Topologies and Routing strategies

NoC Topologies

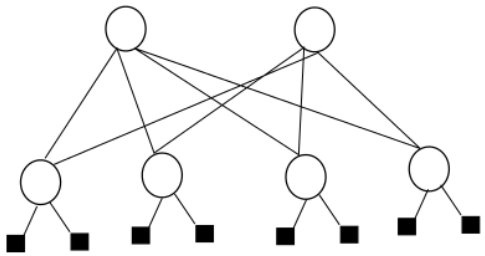
- Mesh
- Fat-Tree
- Ring, Bus etc

NoC Routing Strategies

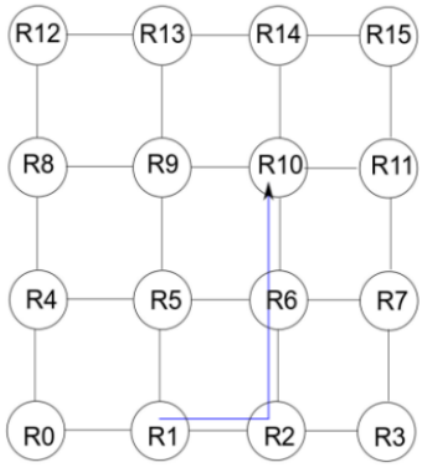
- XY Routing



Mesh Noc 3x3



Fat Tree Noc

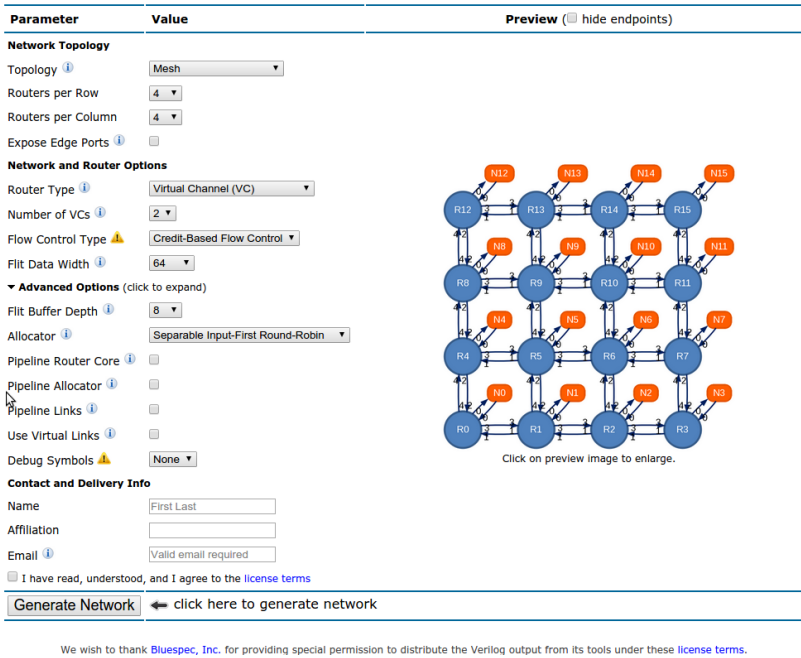


XY Routing

*CON*figurable *NE*twork *C*reation *T*ool [1]

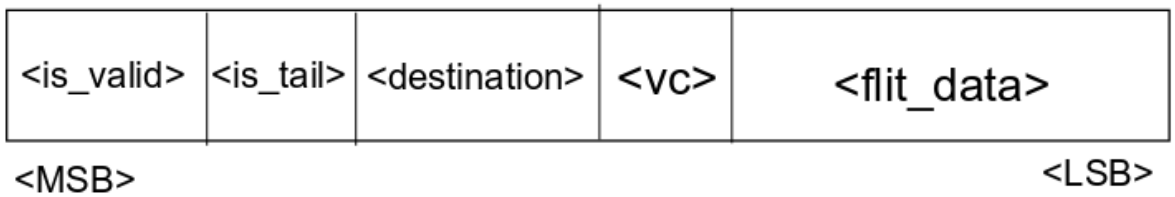
CONNECT NoC

- Developed by Michael K. Papamichael, Carnegie Mellon University
- Generates fully synthesizable Verilog code.
- Latency between two adjacent Processing Element under no traffic is 2 cycle and +1 cycle per additional router distance.
- Provision of custom parameter selection i.e. Data width, number of I/O buffers, Routing strategies.



Developed by Michael Papamichael. Fast scalable FPGA-based network-on-chip simulation models.

Flit-Data structure and NoC Parameters



Topology	Mesh
Routers per Row	4
Routers per Column	4
Router Type	Virtual Channel (VC)
Number of VCs	2
Flow Control Type	Credit-Based Flow Control
Flit Buffer Depth	8
Allocator	Separate-Input First Round-Robin

Router Ports

Each router in this generated network consist of 5 ports each port has the following interfacing pins:

Table: Router Pins

PIN	DESCRIPTION
in_ports_<PORT>_putRoutedFlit_flit_in	Input data to router
EN_in_ports_<PORT>_putRoutedFlit	Enable to receive from the input port
EN_in_ports_<PORT>_getNonFullVCs	Enable for virtual channel for this port
in_ports_<PORT>_getNonFullVCs	Input information of non-full VC
EN_out_ports_<PORT>_getFlit	Output port enable
out_ports_<PORT>_getFlit	Output data from router
out_ports_<PORT>_putNonFullVCs_nonFullVCs	VC information for output port
EN_out_ports_<PORT>_putNonFullVCs	Enable for out port VC

Partitioning Links Between Two Adjacent Routers

- These links shown in table are cut and exposed at the top level for interfacing with inter-chip communication module

Table: NoC Partitioning Links

PIN	DESCRIPTION
in_ports_<PORT>_putRoutedFlit_flit_in	Input data to router Exposed
out_ports_<PORT>_getFlit	Output data from router Exposed
out_ports_<PORT>_putNonFullVCs_nonFullVCs	VC information exposed for interface

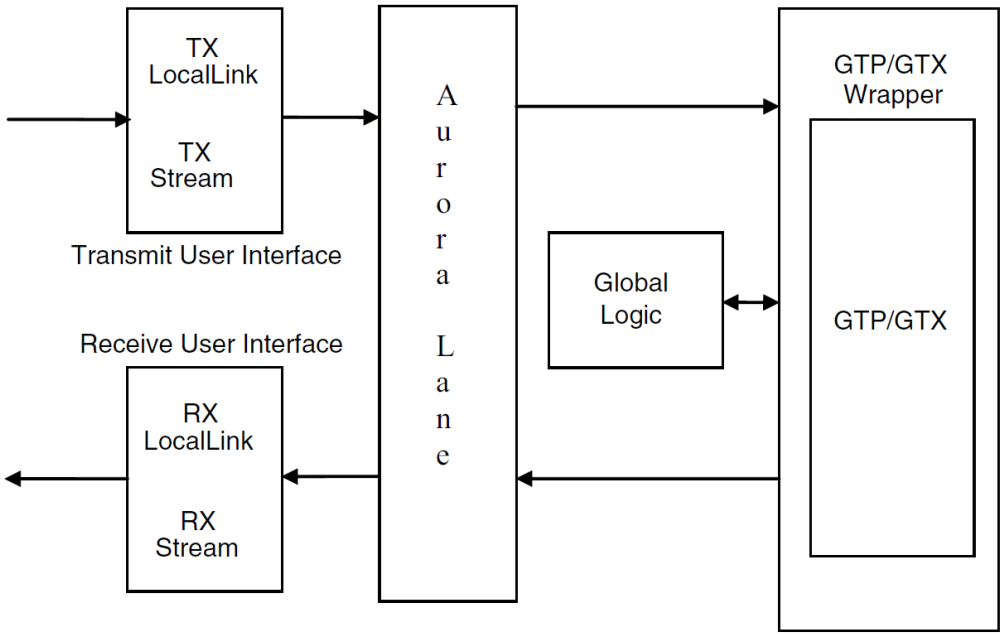
Automated Partitioning Script Parameters

Original Script developed by Yatish Turakhia

```
########## Python script Parameters for automatic partitioning ##############
import re
routers = [0, 1, 2, 3]  # Total No. of router in partitioned network
part_0 = [0, 2]         # Number of router in partition 0
part_1 = [1, 3]         # Number of router in partition 1
part_no_0 = 0           # Partition part Number 0.
part_no_0 = 0           # Partition part Number 1.
ports_per_router = 5    # Number of ports in each router
flit_data_width = 16    # Data width
vc_bits = 1
dest_bits = 2           # Number of destination nodes
data_width = 2
hex_filename = <Routing_table_<filename. Hex>
outside_in =  [[0  ,3],[1  ,3]] # Define the input  port cuts
outside_out = [[0  ,1],[1  ,1]] # Define the output port cuts
########## This script partition's the given network and instantiates #######
########### all the necessary inter-chip link interfaces using wire ########
############################ at specified router position ###################
```

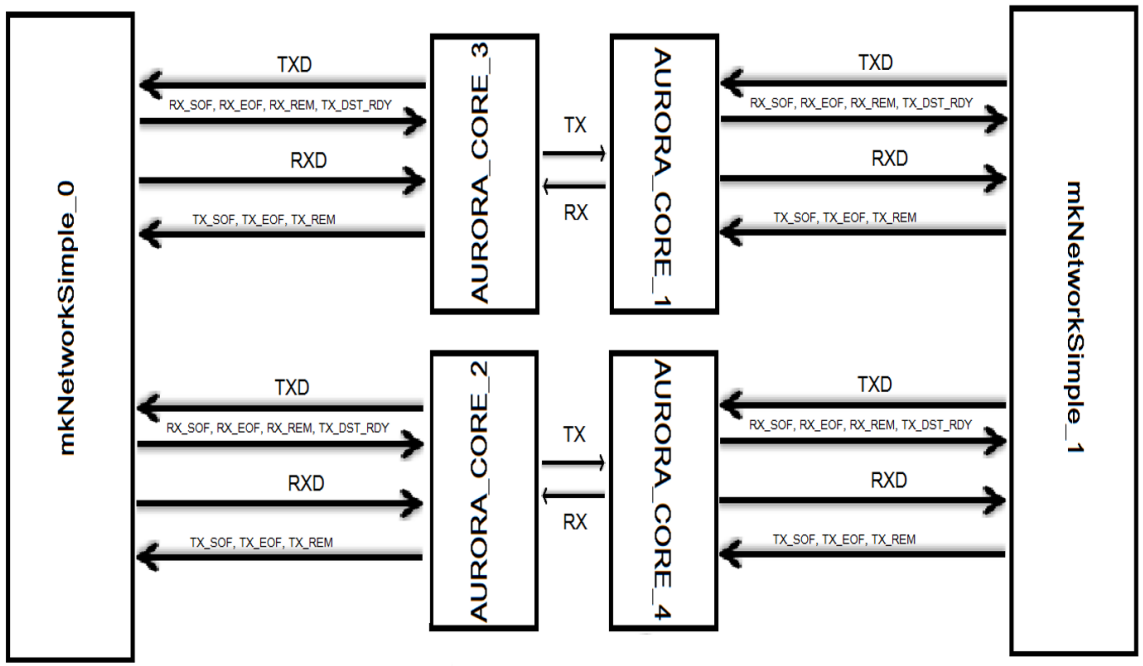
Aurora 8B10B: Block Schematic

This shows a simplified block schematic of Aurora IP Core generated using Xilinx CoreGen.



Experimental Setup

The figure shown below is and experimental setup of partitioned NoC integrated with Aurora core.

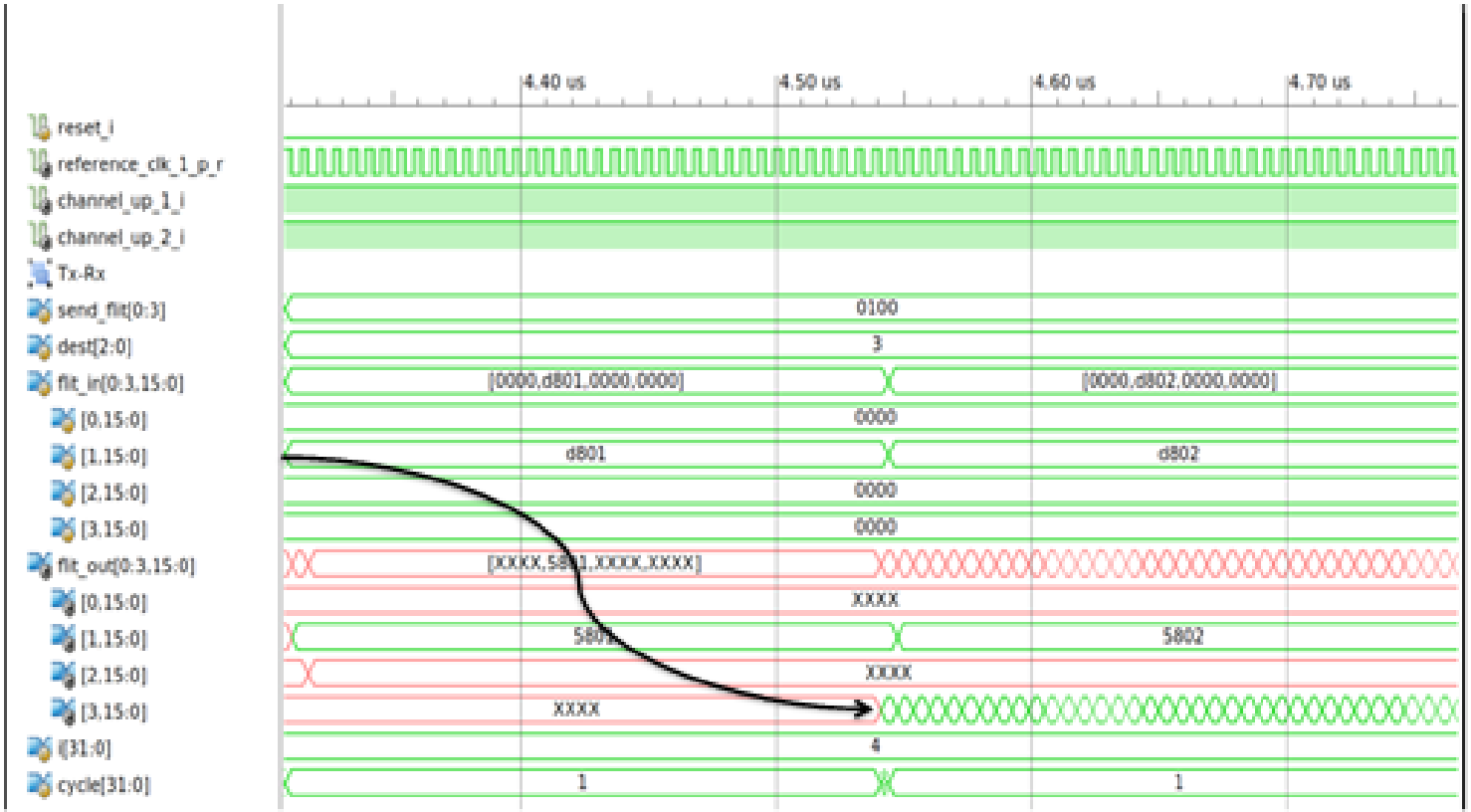


Simulation Result

- Simulation results shows a latency of 36 clock cycles
- Here the system clock and Aurora clocks are different
- System is running at 156.25 MHz
- The data rates obtained is 3.125Gbps
  i.e. each 16 bit data (encoded to 20 bits using 8B10B encoding) is transferred is single system clock
  Therefore, effective giving data rate of

$$156.25 \text{ MHz} \times 16 = 2.5\text{Gbps or}$$
$$156.25 \text{ MHz} \times 20 = 3.125 \text{ Gbps}$$

Simulation Result

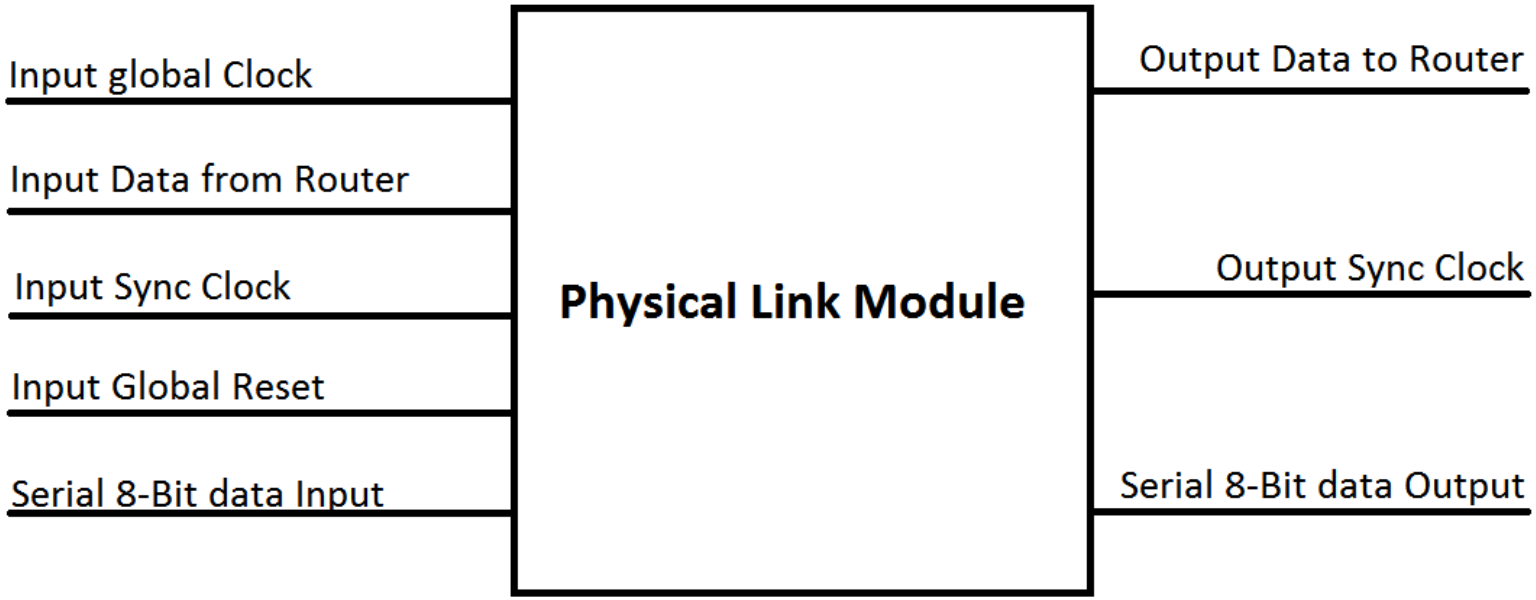


Advantages and Disadvantages

- Easy Core generation and available example design helps integrating user application
- User flow control (UFC) allows applications to send a brief, high-priority message through the Aurora 8B/10B channel.
- Very high data rates can be obtained ideally 3.125Gbps
- Aurora core uses Gigabyte Transceiver Pair (GTP) for high speed serial interface between partitioned NoC which can be a limitation
- This core is device and vendor specific which limits the user

Quasi-SERDES Physical Link Module: Block Schematic

This Quasi-SERDES Physical link module is a partial serializer / de-serializer block which can be used as and interface between two routers.



Quasi-SERDES Physical Link Module Input/Output Port

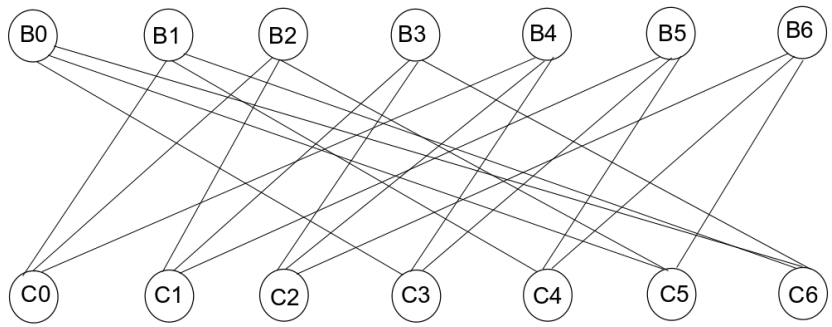
PIN	DESCRIPTION
CLK	Input Clock
rst_n	Global Reset Input(Active low Reset)
input_data_from_router[17:0]	18 bit input Flit
serial_data_in[7:0]	8 bit data input (MSB first)
serial_data_out[7:0]	8 bit data output (MSB first)
output_data_to_router[17:0]	Reconstructed Flit
sync_clk_in	input synchronization clock
sync_clk_out	output synchronization clock

Quasi-SERDES Physical Link Module: Features

- Clock synchronization and reset synchronization is handled in the module
- Any bit to 8-bit serialization/de-serialization
- Data reconstruction with validity check
- Flow control to ensure lossless data exchange
- Simple GPIO to GPIO full duplex communication interface
- $2 \times 2 \times$ n bit (data) / 8 - gives the number of clock cycles for data transfer. Therefore for 24 bit input data we need 12 clock cycles, so effectively 2 bits/clock
- Therefore, if clock is 50MHz (tested on DE0-Nano), Data-rate is 100Mbps
- Buffer-full indication to routers to stop it from sending flits

Tanner Graph for PG-LDPC (7,3)

- PG-LDPC (7,3) code which is obtained by Projective Geometry over $GF(2^s),\ s = 1$.
- The degree of total bit-node and check-node is $2^s + 1 = 3$.
- In this design, 5 bit precision :1 bit for sign, 3 bit of integer precision and 1 bit of fraction precision



Bn: Bit node n

Cn: Check Node n

How LDPC works?

1. Initialize all bit nodes by Log Likelihood Ratio.
2. LLR propagates from bit nodes to check nodes via edges.
3. Check node performs computation.
4. Check node sends the computed LLR back to Bit node via edges.
5. Bit node checks whether the minimum required error is achieved or not.
6. If yes, sends the decoded data out, or else go to step(2).

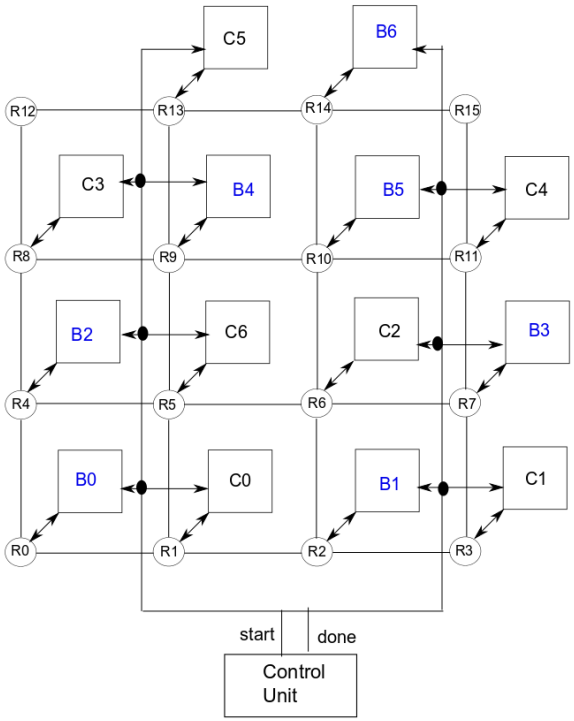
The Algorithm

A very simple explanation of bit-flipping algorithm is as follows

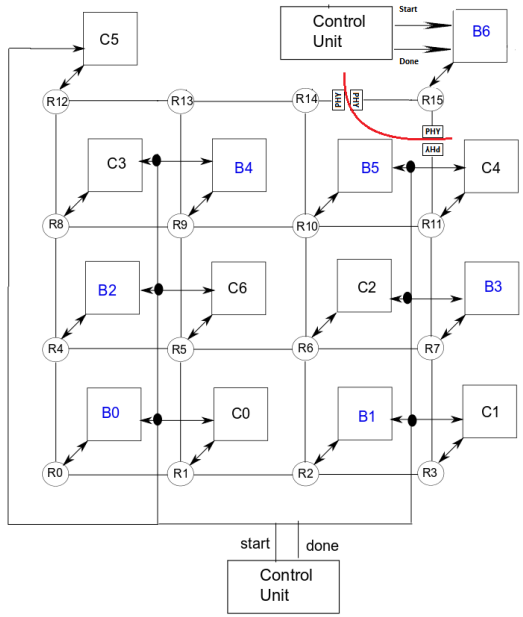
- Check node bits are *satisfied* if the sum of adjacent variable nodes is zero, *unsatisfied* otherwise.
- Bit nodes flip the received value if the number of unsatisfied neighbours is larger than the number of satisfied neighbours.

System Implementation of PG-LDPC (7,3) using NoC

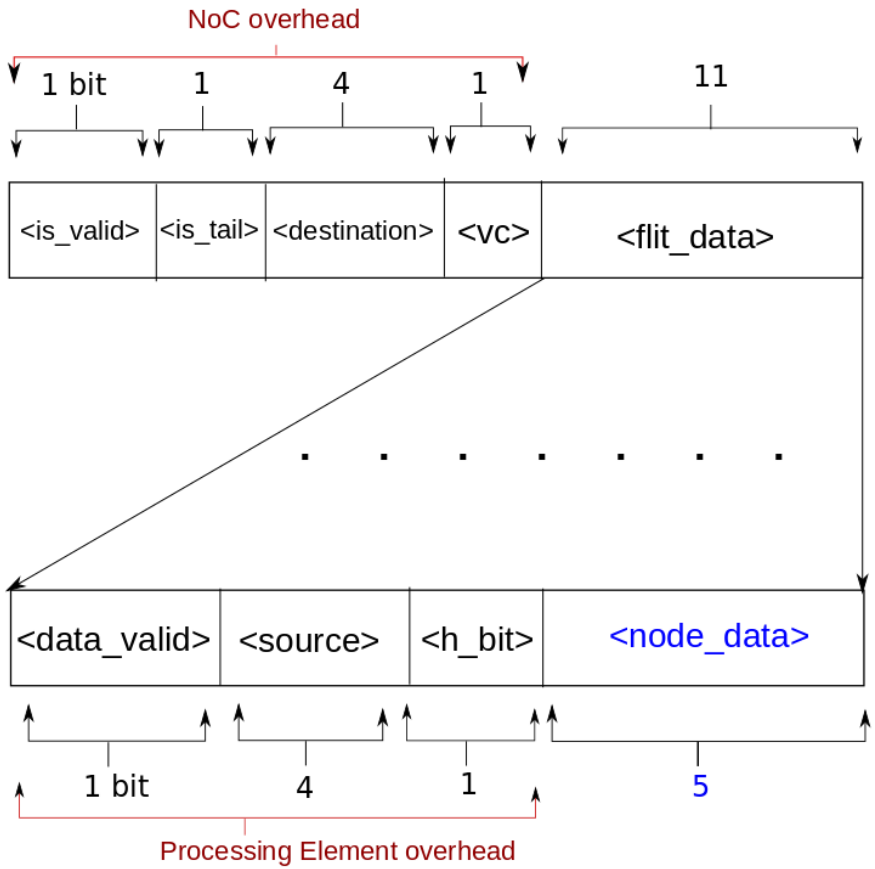
Implementation on Mesh 4x4 un-partitioned NoC



Implementation on Mesh 4x4 partitioned NoC



LDPC over NoC Flit Structure



Result comparison of Implementation of PG-LDPC(7,3) on un-partitioned NoC and partitioned NoC

Parameters	LDPC on 4×4 Mesh un-partitioned NoC	LDPC on 4×4 Mesh partitioned NoC
Clock-cycle (1 iteration)	18	18+12[1]
Clock-cycle (n iteration)	18	18+12[2]
Total cycle for n complete iteration	18 + (n-1)×12	(18 + (n-1)×12) +12

$$DataRate = \frac{Maximum\ \ frequency \times Data-width}{Nos\ \ of\ \ Iteration \times Nos\ \ of\ \ clock\ \ cycles}.$$

1,2:Whenever any data sent or received to or from partitioned node in this case for flit sent or received from bit node 15, additional 12 clock cycles is needed for the expected data to be received.

Performance Comparison and Resource Utilization for Altera :EP4CE22F17C6

	Available Resources	LDPC 7 × 7 Un-Partitioned Mesh 4 × 4 NoC	LDPC 7 × 7 Partitioned Mesh 4 × 4 NoC
Number of Logic Elements	22,320	9,840 (44%)	15,771 (71%) (Part1) 2,291 (10%) (Part2)
Number of Logic Registers	22,320	6,651 (30%)	11,113 (50%) (Part1) 1,787 (8 %) (Part2)
Maximum Operating Frequency	-	36.27 MHz	23.99 MHz (Part1) 85.49 MHz (Part2)
Number of clock cycles	-	18	$18 + 12(\text{Part1})^1$ $18 + 12(\text{Part2})^2$
Data Rate	-	36.27 Mbps	14.5 Mbps

1,2:Whenever any data sent or received to or from partitioned node in this case for flit sent or received from bit node 15, additional 12 clock cycles is needed for the expected data to be received.

Raspberry Pi Features

- Its is an ARM based CPU capable of running real time operating systems
- Sufficiently large number of GPIO pins available for external world application interfacing
- Ethernet port on-board makes it easily accessible remotely
- Easy to use USB port and Ethernet port helps configuring FPGA other development platform easy
- On-broad JTAG port shall prove of great usage for co-design application debugging.

DE0-Nano Features

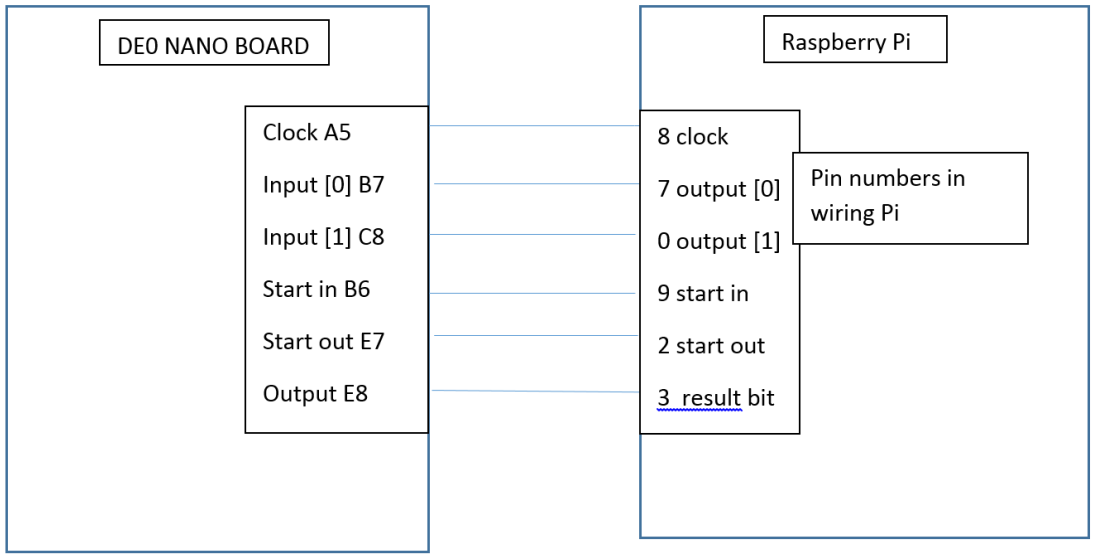
- Cyclone IV E family device EP4CE22F17C6 FPGA available on board having with 22,320 Logic Elements and 22,320 Logic Registers
- Access to 72 GPIO pins and on-board ADC
- Easy to configure and in system programming capability

FPGA/CPLD Configuration

- Raspberry Pi USB port and Ethernet port can be used for remote/hands-on FPGA/CPLD configuration
- Raspberry Pi has on board JTAG port which can be used for JTAG debugging (JTAG port details not shared in Raspberry Pi documentation)
- Easy OpenOCD installation help design and develop open-source hardware
- Remote hardware upgrade of user application useful for hardware software co-design

FPGA/CPLD Configuration

The Figure below show a test set-up of implementation of Bezier interpolation.



Preformed by Siddharth and Abhinav, summer interns 2015, IIT Bombay.

Summary and Conclusion of work done

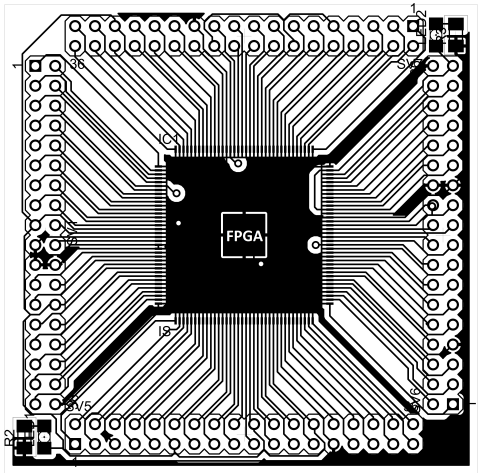
- NoC has been used as a data routing medium, which takes care of routing congestion
- NoC partitioning will help isolating application node from the communication infrastructure
- NoC integration with High speed serial Aurora Core very useful for Multi FPGA implementation
- Quasi-SERDES module between routers of partitioned NoC is a viable alternate
- Automation of NoC partitioning with inter-chip communication link integration using python scripts reduces the manual work tremendously

Summary and Conclusion of work done (contd.)

- LDPC has a complex interconnect network, partitioned NoC can be used for data routing in Multi FPGA implementation
- Bezier interpolation tested using DE0-Nano and Raspberry Pi working at 50 MHz converges in 0.161 seconds
- Raspberry Pi can be used as FPGA/CPLD JTAG configuration (tested) and debugging platform (not tested)
- FPGA/CPLD (Krypton board designed at WEL lab) can be configured using Raspberry Pi (tested), this is a suitable choice for remote lab set-up
- Raspberry Pi interfacing with FPGA presents a viable software hardware co-design platform

Future Work

- Design of a scalable and a repeatable block style design of FPGA PCB for Multi-FPGA NoC implementation
- A simple PCB design shown below which could be repeated to create n×n Mesh of FPGA Network-of-Chips shown in the next slide



1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3
1
2 FPGA 4
3

Reference


Yatish Turakhiya "Multi-FPGA Hardware Acceleration of Boolean Matrix Vector Multiplication using Network-on-Chip", *DDP Thesis-Dept. of Electrical Engineering, IIT Bombay*


Shaishav Shah, "LDPC Decoder Implementation using NoC", *EEE Transactions on Information Theory, pages 27112736, 2001*


Michael K. Papamichael, "Fast Scalable FPGA-Based Network-on-Chip Simulation Models" *Computer Science Department, Carnegie Mellon University*

Reference


Tanner, Robert Michael *EA recursive approach to low complexity codes.*


Kouadri-Mostefaoui, A.-M. and Senouci, B. and Petrot, F., "Scalable Multi-FPGA Platform for Networks-On-Chip Emulation", *Technical Report- Dept of Electrical Engineering, IIT Bombay*


Kumar, S. and Jantsch, A. and Soininen, J.-P. and Forsell, M. and Millberg, M. and Oberg, J. and Tiensyrja, K. and Hemani, A. *A network on chip architecture and design methodology.*


Dally, W.J. and Towles, B. "Design Automation Conference, 2001. Proceedings", *PhD thesis, IIT Bombay, India, 2012*

Thank You