M. TECH. STAGE - I PROJECT REPORT

(EE797)

Title: Investigation of serial data communication in Multi FPGA network of chips

Submitted in partial fulfillment of the requirements of

The degree of Master of Technology

by

Saurabh Agrawal

123076007

Guide: Prof. Sachin Patkar

Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY, BOMBAY

2014-2015

Abstract:

The emerging new generation of high speed computing / Application Specific Integrated Chip design / System-on-Chip designs consists of devices such as micro-controllers, memories, microprocessors, Digital Signal Processors (DSPs), signal conditioning units, etc. These devices are capable of processing at a very high speed which requires inter and intra chip very high speed communication links. With increasing complexity of device designs reconfigurable hardware designs fall short in resources and such architectures have to utilize multi FPGA implementation for realization. High speed communication within a single chip is realizable with not much timing management but, the same architecture when divided to be realized in more than one FPGA the interconnect delays slows the performance and also increases the number IO pins. These issue need to be addressed so that these drawbacks can be handled. This can be done by implementation of high speed serial communication protocol with clock recovery from the serial data received using PLL. One such protocol used in this project Aurora 8B10B v5.3 a Xilinx IP core. Another challenge faced in multi-chip application partitioning. The solution for which is implementation of Network-on-Chip which has been now widely adopted by the ASIC designers community to overcome communication bottlenecks and application independent application. In this project thesis, CONfigurable NETwork Creation Tool (CONNECT [1]) is used to create a fully synthesizable network of desired topology. The generated network is then partitioned either manually or by using a python script with all required initialization of routing tables. The Script written will initialize all the required modules along with the high speed Multi-gigabit aurora transceiver IP core provided as a Xilinx IP core. The system tested through a test bench which initializes the partitioned networks and also connects them via serial wire links. This test bench sends and receives data from each port of each router. Number of clocks required is counted and we can observe the difference in the clock cost requirements.

List of Figures:

Chapter I. Introduction

1.1 Objective: Investigation of serial data communication in Multi FPGA network of chips. This project targets to achieve the following

- Implementation of Xilinx high speed serial IP core Aurora 8B10B as a replacement of parallel bus interconnect between two router nodes of any network topology

- Performance evaluation of High Speed serial communication in Multi FPGA network of chips compared to parallel and simple UART based serial communication

- Exploration algorithms and automated partitioning methods of NoC for Multi-FPGA hardware acceleration

- Implementation of Boolean Matrix Vector Multiplication (BMVM) using Network-on-Chip for communication and also performance evaluation of Multi-FPGA hardware acceleration with respect to single FPGA

1.2 Report Organization:

- Chapter II the motivation behind the use of High-speed communication. This chapter also gives brief idea to understand Aurora 8B10B Xilinx IP core, along with the description of user interface module
- Chapter III gives an overview of Network-on-Chip and describes the methods of NoC generations and also explains the working of a simple NoC
- Chapter IV gives an overview of related work done in this direction of network partitioning
- Chapter V explains our approach and experimental setup for proof of concept
- Chapter VI describes the approach toward automated network partitioning motivated from previous works
- Chapter VII this chapter gives the conclusions for the work done till today and talks about the next stage work and targets that are expected to be achieved
- Appendix I give details about customizing Aurora Core
- Appendix II explains NoC usage and exposed user pins
- Appendix III gives the current directory structure of the project for recreating the simulation project
- References.

Chapter II. High - speed inter chip serial communication

2.1 Overview

As the communication speed increases to multi gigabit, IO performance decays and presents a bottleneck for successful functioning of the hardware. High speed parallel communication has many drawbacks like increase in number of required IO on chip, increase in the hardware resource requirements, specialized IO port design to support multi gigabit communication, crosstalk etc. Parallel communication also needs additional control lines to give meaning to the data for example enable, multiplexing data/control. On the other hand serial communication help overcome some of the issues like reduced hardware resource requirement, reducing inter wire crosstalk etc. and in serial communication flags and markers are created to differential data and control signals.

The main components of for any serial communication are serializer, deseriallizer, and data/clock recovery blocks [2 and 3]. These serialization and deserialization blocks are realized by a few D flip-flops whereas the clock/data recovery block utilizes PLLs. The clock recovery process does not provide a common clock or send the clock with the data. Instead, a phased locked loop (PLL) is used to synthesize a clock that matches the frequency of the clock that generates the incoming serial data stream.

2.2 Our Approach

In our design we have utilized high speed serial IO IP core "Xilinx Aurora 8B10B [4]". The Aurora core is a high-speed serial communication application developed based on the Aurora protocol, Virtex-5 FPGA GTP/GTX transceivers, Virtex-6 FPGA GTX transceivers, and Spartan-6 FPGA GTP transceivers. The core is an open-source code and supports both Verilog and VHDL design environments. This core can be generated by using Xilinx core generator. This generated core comes with an example design that consists of a frame generator module and a frame check module. Frame generator module generates 16-bit data implemented using Linear Feedback Shift Register (LFSR) and which can be regenerated in frame check module for comparison. Aurora core has a local link layer block (explains in detail below) which is interfaced from these frame generator and frame check module. A detailed user guide provided by Xilinx explains the customization, generation and usage of the core. Figure I shows a top level architecture of the aurora core. To use this core we need to understand the framing interface as shown in the Figure II gives pin description.

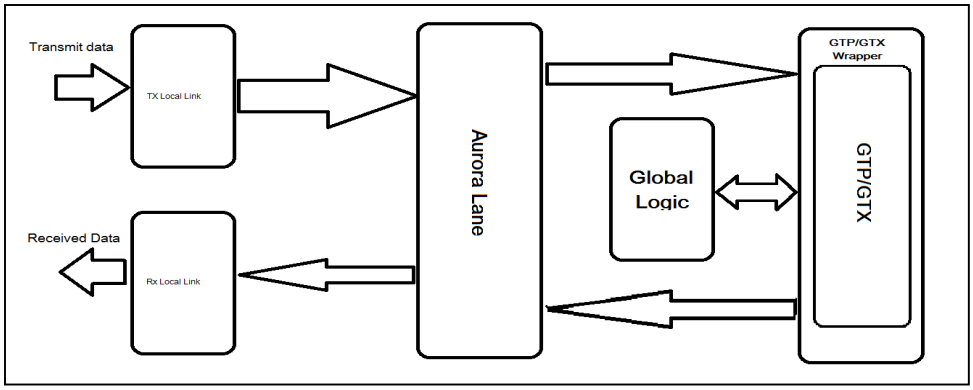

FIGURE I: Aurora Top level Module Block Schematic.

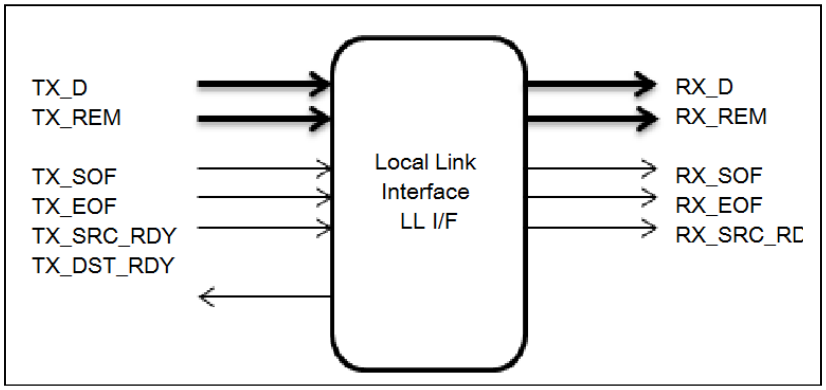

FIGURE II: Aurora Local Link Interface Pin Diagram

To transmit data, the user manipulates control signals to cause the core to do the following:

- Take data from the user on the TX_D bus
- Encapsulate and stripe the data across lanes in the Aurora 8B/10B channel (TX_SOF_N, TX_EOF_N)
- Pause data (that is, insert idles) (TX_SRC_RDY_N)

When the core receives data, it does the following:

- Detects and discards control bytes (idles, clock compensation, SCP, ECP)
- Asserts framing signals (RX_SOF_N, RX_EOF_N)
- Recovers data from the lanes
- Assembles data for presentation to the user on the RX_D bus

Latency through an Aurora core is caused by pipeline delays through the protocol engine (PE) and through the GTP/GTX transceivers. The PE pipeline delay increases as the LocalLink interface width increases. The GTP/GTX transceivers delays are fixed per the features of the GTP/GTX transceivers. This section outlines expected latency for the Aurora 8B/10B core's LocalLink user interface in terms of USER_CLK cycles for 2-byte-per-

lane and 4-byte-per-lane designs. For the purposes of illustrating latency, the Aurora 8B/10B modules partitioned into GTP/GTX transceivers logic and protocol engine (PE) logic implemented in the FPGA fabric. This latency Figures do not take into account the length of the serial interface. Maximum latency for a 2-byte design from TX_SOF_N to RX_SOF_N is approximately 52 [5].

In addition to operations described above, Aurora core interface contains special components such as the elastic buffers, whose role is to avoid buffer overflow/underflow which may occurs due to the frequency difference between sender and receiver, as serializer (transmitter) and deseriallizer (receiver) are located on different boards and use different clock sources. Depending on the transceiver configuration and the clocking scheme used, RIO bit rate varies between 600Mb/s and 3.123 Gb/s. However these values are theoretical, and represent only the serial line bitrates, and not the end-to-end bitrate.

Chapter III. Network on Chip

This chapter would provide the readers a basic understanding of the Network-on-Chip (NoC) design methodology. For those conversant with this topic may skip to Section 2.2 to understand the fundamentals of CONNECT, the NoC generator tool used in our work.

3.1 Overview

Network on Chip or a NoC is a communication module which utilizes networking fundamental and implemented on an integrated circuit called a chip here. NoC technology leverages the networking fundamentals for on-chip and/or inter-chip communication and presents tremendous improvements over conventional bus and crossbar interconnections. Apart from these improvements, NoC implementation improves the scalability and repeatability if ASIC/SoC designs.

3.2 Introduction

Many core chip multi-processors are increasingly being seen as the new incarnation of high performance computing. Next generation of multi-core processors may contain hundreds of processors on a single chip. As the number of cores keeps rising, the on-chip communication infrastructure needs to revamp from the traditional bus architecture, which is the major performance bottleneck. Network-on-chip (NoC) is increasingly being seen as a possible alternative as it achieves the necessary trade-off between the cost-efficient but performance-limited bus architectures and performance efficient but cost/area inefficient point-to-point architectures. It can achieve a high degree of parallelism; however, since its design space is large, the designer is required to make intelligent design decisions to meet the Quality of service (QoS) requirements.

At a high level, a Network-on-chip (NoC) design methodology consists of a number of computational resources communicating to each other over a network infrastructure. The computational resources may include general purpose processing cores, DSP cores, specialized H/W modules, memory units such as RAMs, or even a heterogeneous combination of the above. The communication infrastructure consists of a set of interconnected routers (sometimes also referred to as "switches") which relay the messages from the sending computational element (also referred to as processing element (PE)), in the form of smaller "packets" and "flits", to the destination computational element. In the NoC nomenclature, a "packet" is the smallest unit used for routing and sequencing and may be composed of one or more "flits". A flit (flow control digit) is the unit of bandwidth and storage, and is often also the smallest unit that can be transferred over a channel from one router to the other in a single cycle. It must be noted that a flit may be required to traverse over a number of network routers (which are also responsible for the routing decisions) before reaching the final destination. In effect, the

routing of messages, in the form of smaller packets and flits, takes place on a network infrastructure, consisting of routers and interconnects between them, while the computational elements (connected to the network infrastructure through their associated routers) are only responsible for performing the required computations based on these messages.

Figure III shows an example of a set of 16 routers connected in a Mesh topology, in which a flit sent by router 0 traverses through three other routers (1, 5 and 9) before reaching its final destination, namely router 10.

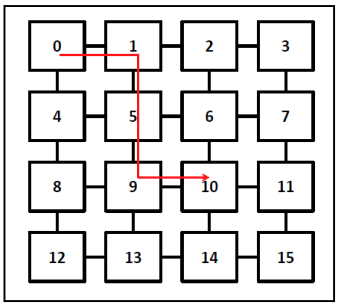

FIGURE III: Example of routing on NoC

The NoC methodology is largely inspired by the established “layered" approach in computer networks and effectively involves separation of application development from the communication infrastructure, thereby ameliorating design productivity. The pioneering paper of Dally et. al [12] originally proposed the use of an on-chip network instead of global wiring for achieving communication between modules. Kumar et. al [13] further dwelled on this idea and identified the four layers in the NoC design methodology:

1. **The physical layer:** This layer specifies the width of the router buffers and channels. It also determines the flit-width. This layer is technology-dependent.
2. **The data link layer:** This layer determines the protocol (for example single cycle or pipelined data transfer) between two interconnected routers. This is again technology-dependent. This layer may also incorporate error-checking.
3. **The network layer:** This layer specifies the routing and forwarding mechanism of packets from the sender to receiver node. The processing nodes are indexed by the address bits (encoded within packets). This is again technology-dependent.
4. **The transport layer:** This layer is responsible for bundling of messages into smaller "packets" and "flits" and also the appropriate addressing of source and destination nodes. This is often handled by the interfacing modules.

This layer is technology-independent. As we enter the billion-gate era [11], the NoC design methodology significantly helps to improve the productivity, design cost and time-to-market in the following ways:

- The dissociation of communication and application layers and restricting the communication to standard interface enforces modularity in design, which also aids

reuse of modules. It enables communication between heterogeneous modules and also imposes information hiding, since details of one module need not be known to the other for communicating messages. This has been explained in [9].

- The isolation of the network interface further reduces the test and verification costs.
- The NoC can be easily customized for a particular application to meet the timing and performance requirements.
- This is a scalable approach and can be easily extended to hundreds of communicating processing elements.

In addition to the above, NoCs also find applications in cache coherency [5], GALS architectures [14] and secure architectures [15]. We now discuss the four major considerations governing the design of a Network-on-Chip, as identified in [16], namely the NoC topology, routing, flow control and router microarchitecture:

1. **Topology:** The topology of a network describes how the routers and the processing elements are connected in the network infrastructure. Most commonly explored NoC topologies in the literature include Ring, Mesh, Torus, Tree and butterfly. Topology can have a significant impact on the overall performance, as topology, together with routing strategy, determines the number of hops for message transmission. A topology often determines the cost of the NoC as it decides the number of PEs used and layout complexities involved. It may also have a significant bearing on the NoC power consumption, which is increasingly becoming a serious concern.
2. **Routing:** The routing strategy determines how the packet is routed from its source to the final destination. The routing algorithm maybe (i) deterministic, in which the path from source to destination is always fixed (e.g. X-Y routing in Mesh), (ii) oblivious, in which routing decisions maybe different for the same source and destination, however, the routing decision does not consider the state of the network (e.g minimal oblivious routing) or (iii) adaptive, which uses local or global information about the network state to make decisions (e.g. PQ-routing, negative first routing). The goal of the routing strategy is often to balance the load in a network, however, several routing strategies suffer from deadlock and the designer needs to be careful in choosing the right routing strategy.
3. **Flow Control:** The role of flow control is to regulate and manage the shared resources, such as buffer space, in the network. Since, unlike computer networks, the NoC is required to be lossless, the flow control protocol must also ensure that packets are not dropped and reliably delivered. Based on the granularity of resource allocation and bandwidth utilization, the flow control mechanisms are broadly classified as store-and forward, virtual-cut through, wormhole and virtual channel flow control. Flow control mechanism has a direct impact on the packet latency (and therefore Quality-of-Service or QoS), link utilization and complexity of router micro-architecture. The most commonly used flow control mechanisms for guaranteeing lossless behavior of network are
   a. Credit-based flow control and

b. Peek flow control (also referred to as on/off signaling).

In credit-based flow control, the sending routers maintain a credit count for each downstream router, which indicates the amount of available buffer space in the downstream routers.

In peek flow control, the downstream routers directly signal the buffer availability and the upstream routers are required to check this signal before forwarding flits. Peek flow control obviates the need for maintaining credit counters for available buffer space and also reduces the number of flow control signals.

4. **Router micro-architecture:** The router micro-architecture deals with the actual circuit implementation of the router that helps in realizing the routing and flow control strategies. A typical router consists of a set of input ports connected to a routing logic, which determines the output port along which the flit/packet has to be forwarded. The flits are stored in buffers before being scheduled by the arbitration logic to the switching network, after considering the buffer space availability in the downstream buffers. The switch finally helps the packet depart the router through the selected output port. Additionally, each port (input and output) has flow control signals associated to it. Figure IV show a typical router micro-architecture. Sometimes, the router data path is pipelined to improve the clock speed. Router complexity is influenced by the number of incoming/outgoing ports and the flow control and routing logic.

The routers at each level share a similar functionality i.e. routing and handling packets protocol. They contain up to 5 ports. The port indicates a direction to/from the group, the cluster or the system root. This port can be connected only to the router of the same level or the gateway to the higher level of the hierarchy. If this port is not connected, the router acts as a system root and discards packets of unknown destination. That means that the input port can be connected to the output port of another router at the same level or to the group gateway. Additionally, in case of a local network, the input port may be connected to the application end point.

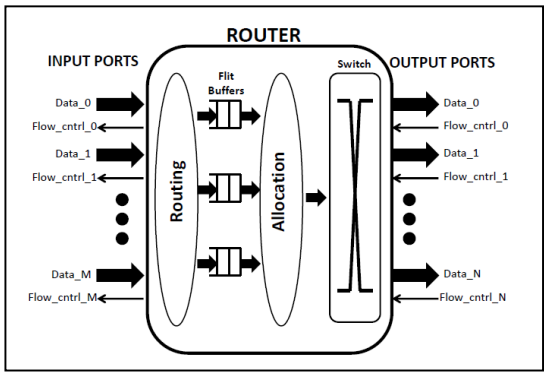

FIGURE IV: A typical router micro-architecture.

3.3 CONNECT for network generation

CONfigurable NETwork Creation Tool [1] created by Michael Papamichael of Computer Science Department, Carnegie Mellon University, provides an FPGA friendly fully synthesizable NoC. This RTL design of a NoC can be automatically generated using the web based interface for any desired topology and node count. CONNECT supports a variety of common network topologies, as well as custom user-conFigured networks, that can be either specified through special configuration files or created through the CONNECT Network Editor. A variety of network and router parameters, such as the router type, pipelining options, the number of virtual channels or allocator type, allow the user to further customize the network and trade-off between resource usage and performance to satisfy application-specific requirements. Refer to readme document for detailed user guide of the tool. Figure V is a network generated by CONNECT.

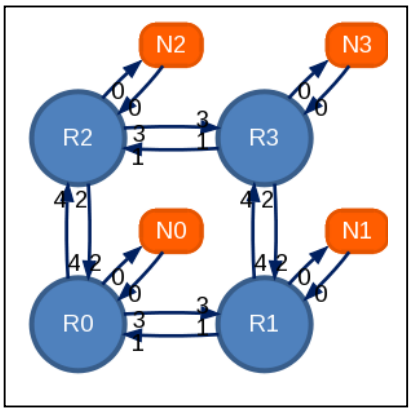

FIGURE V: 2X2 Mesh topology network generated using CONNECT

Each of these nodes is associated with a routing table which directs the incoming data know as a flit to a particular output port based on the input address flagged in the flit packet. The flit format is as shown below

<valid bit>	<is_tail>	<destination>	<virtual channel>	<data>

<Valid bit>: Indicates a valid flit. Set to 1 when sending a flit.

<Tail bit>: Indicates this is a tail flit, i.e. the last flit of a packet. Set high for single-flit packet as they are also tail flits.

<Destination>: Specifies the destination of the current flit. Populate with a valid binary encoded router ID that corresponds to one of the routers in the network.

<Virtual channel>: For VC-based networks this field specifies the virtual channel to use and should be populated with a valid VC, depending on the number of available virtual channels. Lower VCs are prioritized over higher VCs. Networks that do not employ virtual channels (Virtual-Output-Queued and Input-Queued) essentially behave as if they only had a single virtual channel. As a result, for non-VC networks this field is 1-bit wide and should be set low.

<Data>: User data field. Populate with data to be transferred.

The table below gives the routing table preconfigured for this network topology.

Destination	**Router 0**	**Router 1**	**Router 2**	**Router 3**
0	Port 0	Port 1	Port 2	Port 2
1	Port 3	Port 0	Port 2	Port 2
2	Port 4	Port 1	Port 0	Port 1
3	Port 4	Port4	Port 3	Port 0

Let the input flit at router 0 port 0 be

<1/Valid/, 0/Tail/, **11/Destination**/, 0/VC/, 1010101111/Data/>

To understand the transfers we just need to keep a note of the destination flags in the flit format. Router 0 looks up in the routing table and puts the flit to the appropriate port. In this case port 4. Now the flit is received at the port 4 of router 2 which is again directed to port 3, which is received at port 3 of router 3. Finally looked up in the routing table and is presented in port 0 of router 3. The transfer of the flit from source to destination is now complete. The cost of the transfer in terms of number of clocks required is equal to the number of routers flit passes through to before it reaches the destination. In this case no. of clock cycles required by the flit from source to destination is 3 clock cycles.

Chapter IV. Related Work

The proposed [5 and 6] design methodology in Figure VI tries to build a customized-accurate emulation platform for a given system, by choosing the right cores mapping and system partitioning; both steps are aimed at minimizing off-chip activity and causing a minimum accuracy alteration. The paper also suggests a formula for network partitioning partition. With N reconfigurable devices available, the NoC being emulated needs to be partitioned into K (K = N) subsets (cluster) and each one assigned to a reconfigurable device, with respect to the following constraints:

1. Total intra-cluster links must be minimized.
2. A cluster must fit on one FPGA chip.

NoC for multi-chip systems, should exhibit the following features [6]:

1. Scalability, i.e. extension of a current structure with new network modules cannot require changes in an existing part of the network,
2. Multicast support
3. Synthesizability for FPGA.

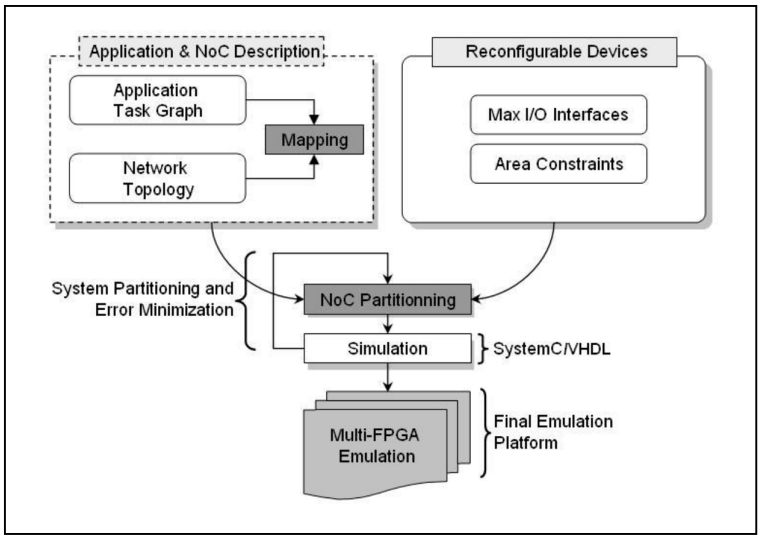

FIGURE VI: Proposed flow for Multi FPGA NoC emulation in [4 and 5]

Previous design [3] for implementation of FPGA-based hardware acceleration of Boolean Matrix Vector Multiplication used the same concept with a UART transmitter receiver pair. This was a proof of concept test for the current architecture. The Figure VII below shows the schematic architecture of UART interconnect design. A functional python

script developed for the automated partitioning of a network partitions a given network and initialized UART transceiver pair.

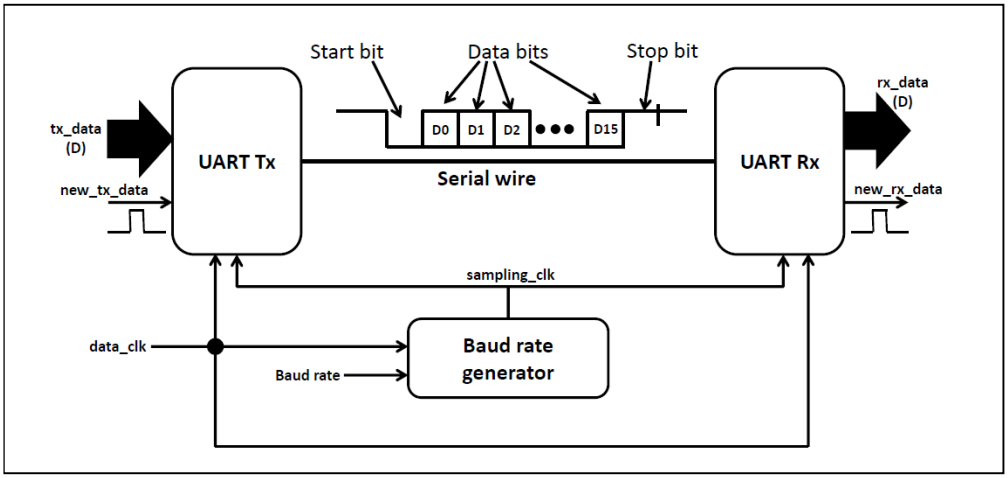

FIGURE VII: Partition networks with UART links

In the paper [7], a clear description of all the important focus points are illustrated in regards with a communication network with all the protocol specific details. A details study and description of Network-on-Multi-Chip (NoMC) is described. A NoMC architecture which is the hierarchical NoC designed for multichip systems. The proposed interconnect system consists of the three levels of the hierarchy, each separated with a dedicated device, referenced as the gateway. Its main task is data formatting to suite respective level requirements. It is a flexible and scalable approach that allows separating the design of each level. The addressing scheme reflects a division of the NoMC into the three levels. Therefore, the address consists of a three fields, each regarding the respective hierarchy level.

Chapter V. Partitioning the Network and Instantiation of High speed Links

Partitioning of network is a very useful technique which provides opportunities to test large ASIC and SoC which are not able to fit in a single chip. The challenges of partitioning a network for inter chip communication is large number of IO requirement and IO need to be designed to support high speed switching. To overcome these challenges we introduce high speed serial links in the partitioned instances. Figure VIII shows 2X2 mesh partitioned into 2 instances and also the high speed serial links between them.

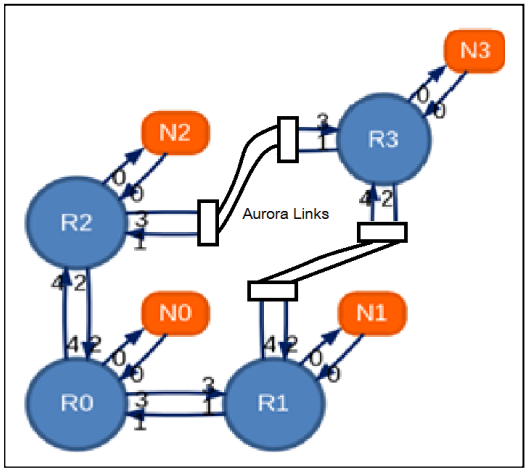

FIGURE VIII: Partitioned Network with High Speed serial Aurora Links

Let us now exercise the above example again in this setup

`<1/Valid/, 0/Tail/, **11/Destination**/, 0/VC/, 1010101111/Data/>`

The flit traverses in exactly the same manner till the router 2 port 3. When the flit arrives at port 3 of aurora module it undergoes encoding in based on Aurora 8B10B protocol, parallel to serial conversion and then transmitted. This serial data is then transmitted from specialized high speed multi-gigabit transceiver (MGT) IOs which is received on the similar MGT on the counter part in the second partition. This received serial data is the converted into parallel data and decoded to form the original flit. And finally when it reaches the destination router the flit is pushed out from port 0 of router 3.

This partitioned network with high speed serial links introduces a latency of 36 clock cycles which is trade of against the reduction of number of required IOs. The results as compared to earlier UART links are much better. Simulation waveforms are shown in

Figure IX of Aurora. This is a general architecture of a partitioned network of any topology when divided into 2 partitions. Generated simulation waveforms are shown in Figure X and Figure XI followed by console input output strings.

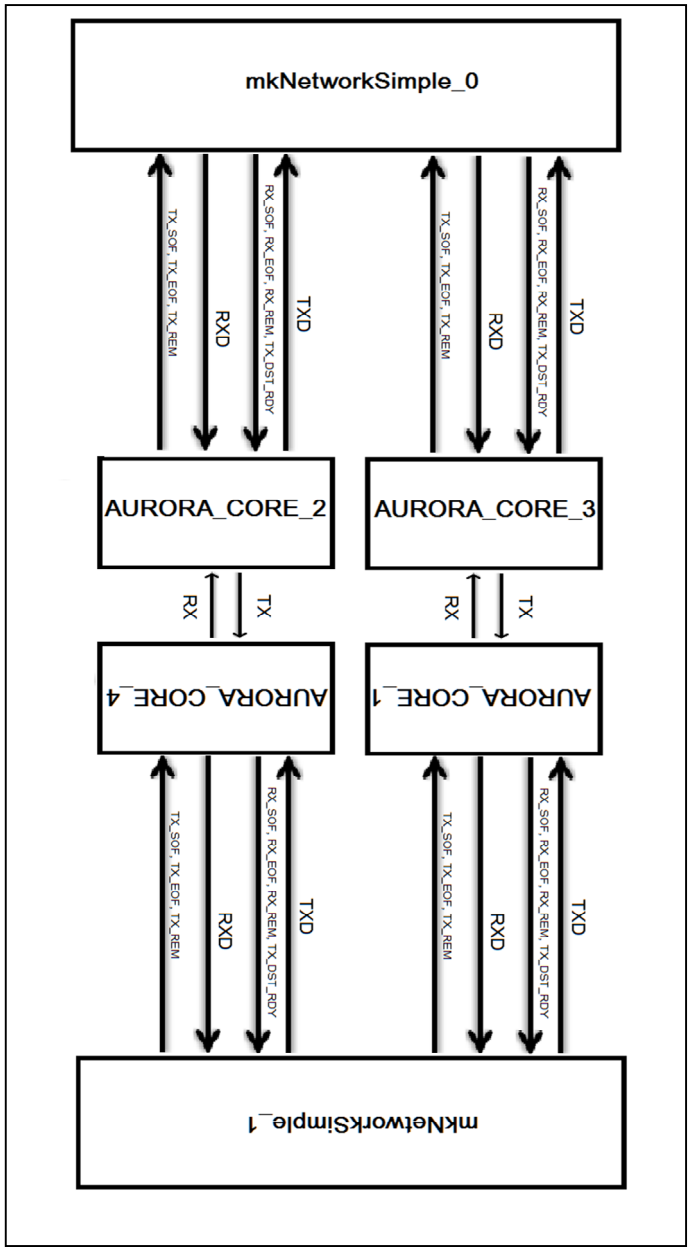

FIGURE IX: Inter connected partitioned network architecture

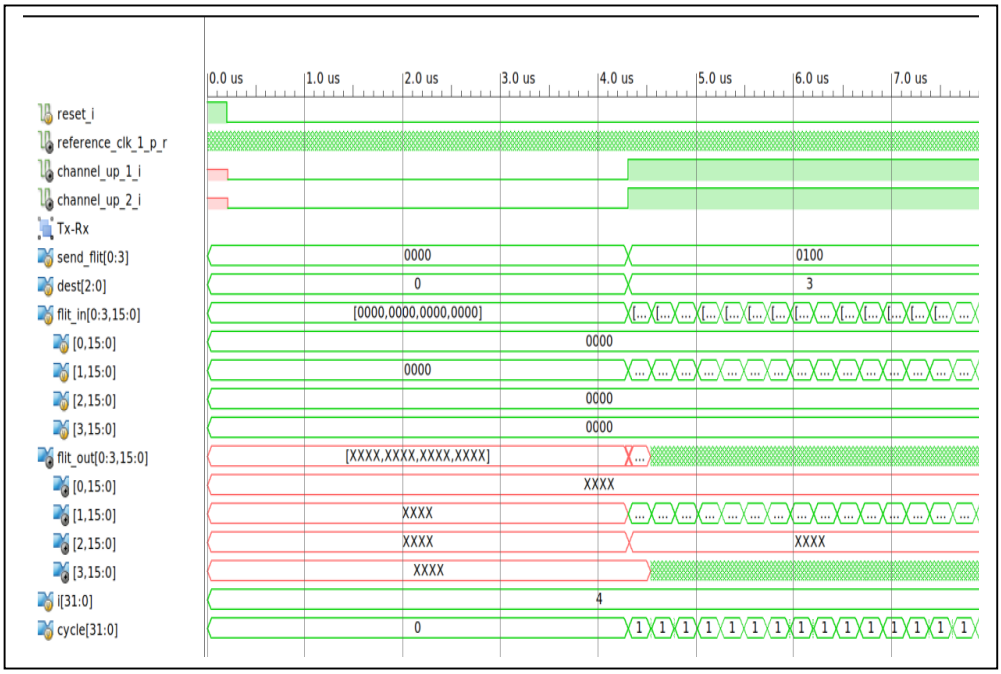

FIGURE X: Simulation output waveform part1

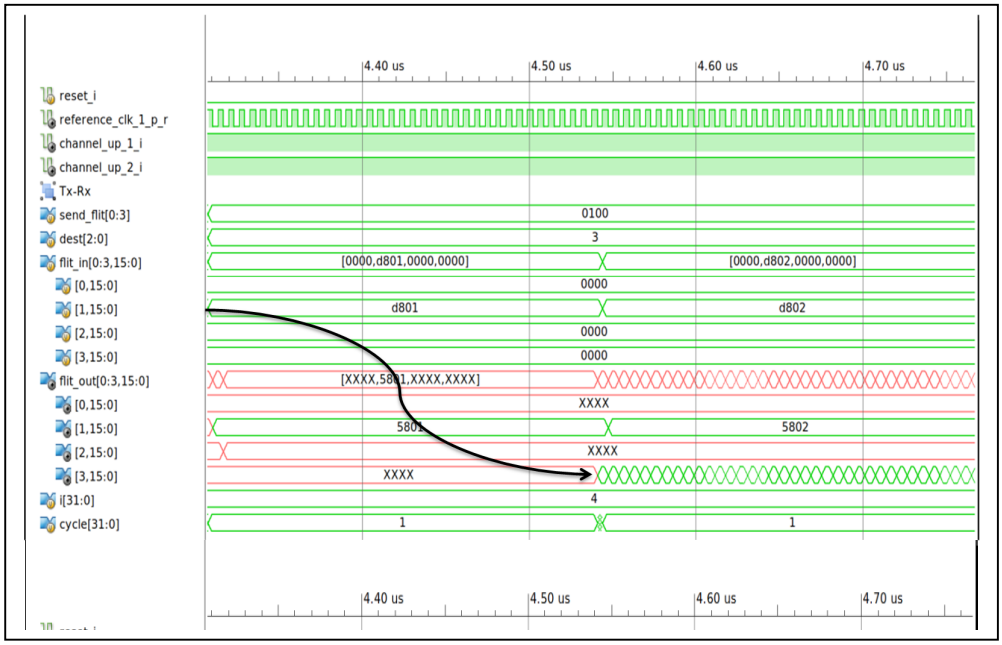

FIGURE XI: Simulation output waveform part2

Simulation result:

@1: Sending flit 001 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 001 at receive port of router = 3
@1: Sending flit 002 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 002 at receive port of router = 3
@1: Sending flit 003 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 003 at receive port of router = 3
@1: Sending flit 004 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 004 at receive port of router = 3
@1: Sending flit 005 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 005 at receive port of router = 3
@1: Sending flit 006 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 006 at receive port of router = 3
@1: Sending flit 007 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 007 at receive port of router = 3
@1: Sending flit 008 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 008 at receive port of router = 3
@1: Sending flit 009 into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 009 at receive port of router = 3
@1: Sending flit 00a into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00a at receive port of router = 3
@1: Sending flit 00b into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00b at receive port of router = 3
@1: Sending flit 00c into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00c at receive port of router = 3
@1: Sending flit 00d into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00d at receive port of router = 3
@1: Sending flit 00e into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00e at receive port of router = 3
@1: Sending flit 00f into send port 0 from Router = 1 to Router = 3
@0: Receiving flit 00f at receive port of router = 3
@1: Sending flit 010 into send port 0 from Router = 1 to Router = 3

Chapter VI. Automated NoC Partitioning

6.1 Overview

FPGAs have been extensively used for full system prototyping and verification of large ASIC designs [8, 9 and 10]. Lowering costs, as a consequence of advanced manufacturing processes and increasing competition, coupled with rising operating speeds, have rendered FPGA prototyping a considerable advantage over other known alternatives. These include RTL simulation and virtual prototyping, which are known to be slow and expensive in comparison. FPGAs make good use of available fine-grained parallelism and find applications in High Performance Computing (HPC) and hardware acceleration. A number of hardware/software co-simulation tools, such as ProtoFlex , have been developed which rely on FPGA for fast hardware emulation of complex designs and also provide testing and debugging support. Modern day customized chips, on the other hand, have also seen a remarkable rise in design complexity with advanced ASICs consisting of over a billion gates. Although FPGAs have also witnessed a rapid increase in integration costs, resource constraints in terms of limited gate capacity, DSP units, I/O pins, memory arrays and clock speeds render FPGAs almost impractical for prototyping of very large designs using a single chip. To mitigate this issue, the large ASIC designs are often partitioned across multiple FPGAs such that the major functional blocks reside over different FPGAs, which are further provisioned with an appropriate communication interface. Multi-FPGA partitioning is still considered to be one of the toughest challenges in FPGA prototyping. It is cumbersome and error-prone as it requires considerable manual intervention and knowledge of partitioned modules. To this end, we believe that a standardized and simplified communication interface based on Network-on-chip (NoC), with automated partitioning across multiple FPGAs, could greatly allay the designers from the tedious partitioning process. The partitioned NoC should also be flexible in terms of topology and routing configurations. Since several modern day ASICs and MPSoCs also employ NoCs for communication between modules, our partitioned NoC would also be effective for prototyping and debugging of these designs on multiple FPGAs, which are otherwise limited by the resource constraints of a single FPGA. This approach is also handy for designing multiple FPGA-based hardware accelerators, using the underlying NoC as the communication interface.

6.2 NoC partitioning

Parameters:

- Routers: List containing all router id's in the NoC to be partitioned.
- Part: List containing the router id's in the partition.
- Part_no: Partition number.
- Ports_per_router: Number of ports in the router module.

- Flit_data_width: Flit data width in the CONNECT generated NoC (can also be looked up in connect_paramters.v).
- vc_bits: Number of bits for virtual channels (Should be 1 for non-VC networks. Can be looked up in connect_paramters.v).
- dest_bits: Bits required for specifying port address (logarithm of number of end/receive ports in the Network).
- hex_filename: Prefix of the .hex files generated for routing tables by the CONNECT tool.

Usage: Python script. Place the input NoC file mkNetworkSimple.v in the same directory.

Command: python mkNetworkScript.py

Output: Output partitioned network file mkNetworkSimple_(part_no).v.

```
########## Python script Parameters for automatic partitioning ###############
import re
routers = [0, 1, 2, 3]                  # Total No. of router in partitioned network
part = [0, 1, 2]                        # No. of router in this partitioned network
part_no = 0                             # Partition part No.
ports_per_router = 3                    # No. of ports in each router of partitioned network
flit_data_width = 16                    # Data width
vc_bits = 1
dest_bits = 2                           # destination width depending on no. of ports
data_width = 2
hex_filename = <Routing_table_<filename.hex>

  ##This script is supposed to partition the given network and instantiate all the necessary ##
  ######Serial Aurora module and interfaces using wire to appropriate counterpart########
```

Chapter VII. Conclusion and Future Work

7.1 Conclusions

This experimental setup presents suitable and scalable high speed serial link integration between partitioned networks. The project uses a custom networks generated separately using CONNECT custom network generator. These networks generated are designed, when joined together using direct parallel interface resemble an un-partitioned mesh topology. For the experimental setup these networks were connected via high speed serial links. Simulation tests the hardware by providing inputs to all ports and reports the arrival, when the defined destination receives the data. Simulation waveform shows maximum latency of 36 clock cycles. This seems to be an acceptable trade-off in terms of hardware requirement.

7.1.1 Advantages of Automated NoC Partitioning for Multiple FPGAs

In addition to some of the advantages of automated NoC partitioning for multiple FPGAs listed earlier, we enumerate a few more advantages of our approach:

- This approach scales seamlessly to a large number of modules and FPGA partitions. The application developer is not required to be aware of the details of the serialization/de-serialization modules used by the NoC. The user modules are also oblivious to the NoC partitions and may continue to use the NoC in the same manner as without the partition.
- The partitioning produces a synthesizable Network-on-Chip spread across multiple FPGAs and is flexible in terms of network topology and routing. Since the network is generated using CONNECT modules, it is also optimized for FPGA fabric.
- This NoC-based partitioning approach allows several modules operating at different clock speeds to effortlessly communicate with each other. This is because; the partitioned network uses asynchronous serial links, instead of synchronous parallel wires, to communicate with adjacent routers. In fact, our approach could be used in prototyping voltage-frequency islands for GALS-based networks-on-chip onto FPGAs, as described in [11]. The authors have demonstrated 40% energy savings using this approach on real video applications.
- Since the routers use serial links to communicate across different FPGA partitions, this approach minimizes the requirement of sparsely available I/O pins on the FPGA. Use of peek flow control, instead of credit-based flow control, further helps in this regard.
- This approach enforces modularity and information hiding in a multi-FPGA design, further improving design productivity and development time of multi-FPGA designs.

- Our approach also allows for design-space exploration of NoCs over multiple-FPGAs. A network topology could have significant bearing on the overall performance and design space exploration would help strike the right balance between cost and performance.

7.1 Future Work

The next part of the projects involves automated partitioning of generated network. The challenge here is to partition the desired network topology, instantiate high speed aurora modules at appropriate positions and connect all the data generation hardware units at appropriate input interface nodes to produces and quantify the concept of multi FPGA network partitioning. After a successful creation of partitioned networks with exposed high speed serial links, hardware testing with a high speed computing application like matrix multiplication will substantiate the simulation results.

Appendix I. Customizing the Aurora Core

Aurora core can be customized according the application. An example settings screen shot is shown below. Refer [4] for details.

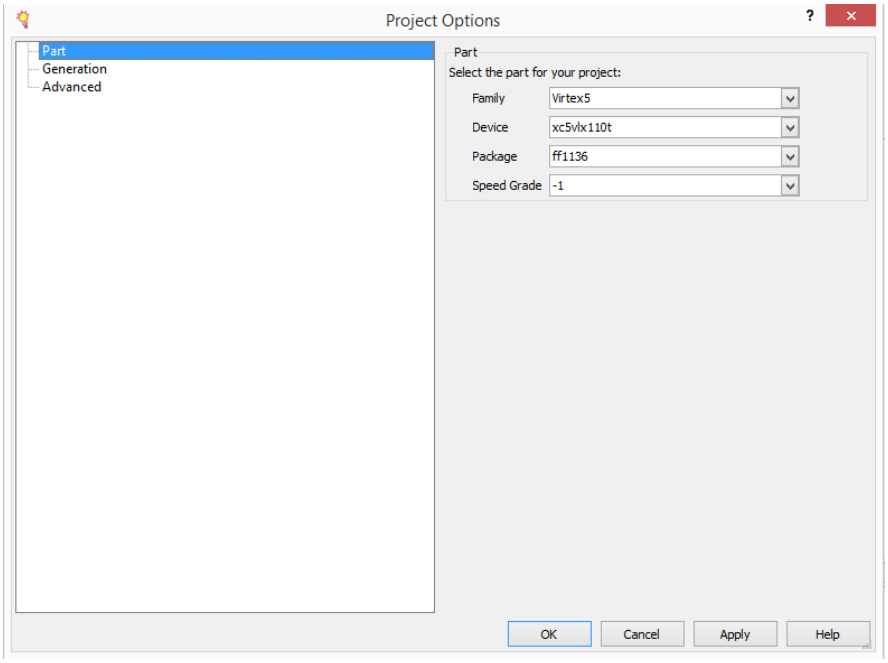

FIGURE XII: Part selection for core generation

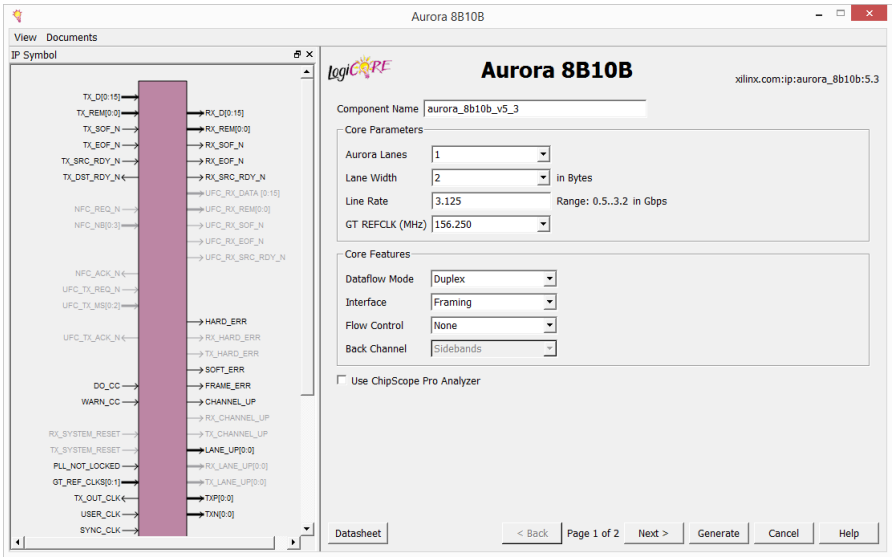

FIGURE XIII: Customizing Aurora core page 1

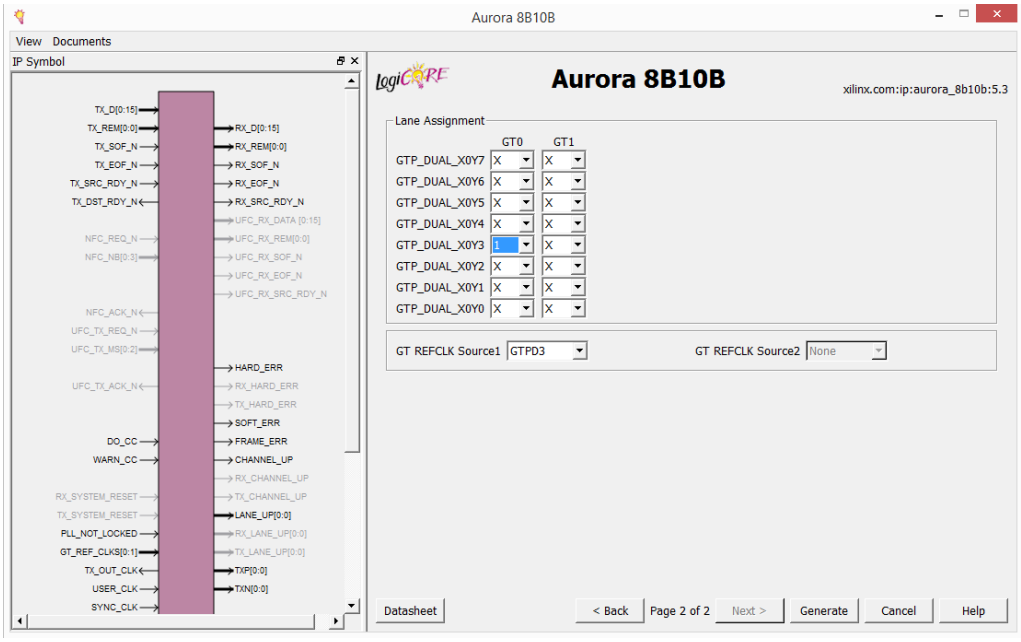

FIGURE XIV: Customizing Aurora core page 2

1. Open Xilinx core generator.
2. Create new project.
3. Make the setting as shown in Figure XII
4. Double click Communication and Networking > Serial Interface > Aurora 8B10B
5. Make settings according to Figure XIII and Figure XIV
6. Click generate.
7. Open the generated core in Xilinx ISE. This generated design has an example test bench. This example design has a frame generator and a frame check module.

Appendix II. CONNECT NoC generator

CONNECT is a flexible RTL generator for fast, FPGA-friendly Networks-on-Chip. This website serves as a front-end to CONNECT's network generation engine, which is primarily based on Bluespec System Verilog. All networks generated through CONNECT consist of fully synthesizable Verilog. CONNECT supports a variety of common network topologies, as well as custom user-specified networks. A variety of network and router parameters, such as router type, pipelining options, the number of virtual channels or allocator type, allow the user to further customize the network and trade-off between resource usage and performance. For license information please see the LICENSE file.

All necessary files to implement a working network are located in this directory. For networks with credit-based flow control the top module is called "mkNetwork" and is located in the mkNetwork.v file. For networks with peek flow control the top module is called mkNetworkSimple and is located in the mkNetworkSimple.v file. Both the mkNetwork and mkNetworkSimple module interfaces consist of a set of ports used for sending and receiving flits and flow control information. The credit-based and peek-based networks share the same interface for sending and receiving flits, but offer different flow-control interfaces, which are detailed below.

- EN_send_ports_<P>_putFlit input wire: Assert when sending a flit to indicate send_ports_P_putFlit_flit_in carries valid data.
- send_ports_<P>_putFlit_flit_in input bus
- EN_send_ports_<P>_getNonFullVCs input wire: indicates sender is ready to receive a credit.
- send_ports_<P>_getNonFullVCs, output bus, that carries a bitmask indicating which VCs have available buffers space (i.e. are non-Full). MSB corresponds to the highest VC and LSB corresponds to the lowest VC. For non-VC networks this signal is a single bit.
- EN_recv_ports_P_getFlit input wire: Assert when ready to receive an incoming flit
- Recv_ports_P_getFlit output bus

CONNECT-generated networks that use Peek flow control offer a simpler interface. Instead of maintain and exchanging credits, network clients simply need to check if the buffer they intend to inject into is not full by checking the send_ports_<P>_getNonFullVCs signal. In the case of VC-based networks they need to check the specific bit that corresponds to the VC that the flit belongs to.

Appendix III. Project Directory Structure

```
custom_TB.v
        | ---- Example_design_1_i.v
        |               |-------clock)module_i.v
        |               |-------standard_cc_module.v
        |               |-------reset_logic.v
        |               |-------aurora_module_i_3
        |               |               |-----aurora_lane_0_i.v
        |               |               |               |-----lane_init_sm_i.v
        |               |               |               |-----chbond_count_dec_i.v
        |               |               |               |-----sym_gen_i.v
        |               |               |               |-----sym_dec_i.v
        |               |               |               |-----err_detect_i.v
        |               |               |-----gtp_wrapper.v
        |               |               |               |-----GTP_TILE_INST.v
        |               |               |-----global _logic.v
        |               |               |               |-----channel_init_sm_i.v
        |               |               |               |-----idle_and_ver_gen_i.v
        |               |               |               |-----channel_err_detect_i.v
        |               |               |-----tx_ll_i.v
        |               |               |               |-----tx_ll_datapath_i
        |               |               |               |-----tx_ll_controlpath_i
        |               |               |-----rx_ll_i.v
        |               |               |               |-----rx_ll_pdu_datapath_i
        |               |-------aurora_module_i_4
        |               |               |-----aurora_lane_0_i.v
        |               |               |               |-----lane_init_sm_i.v
        |               |               |               |-----chbond_count_dec_i.v
        |               |               |               |-----sym_gen_i.v
        |               |               |               |-----sym_dec_i.v
        |               |               |               |-----err_detect_i.v
        |               |               |-----gtp_wrapper.v
        |               |               |               |-----GTP_TILE_INST.v
        |               |               |-----global _logic.v
        |               |               |               |-----channel_init_sm_i.v
        |               |               |               |-----idle_and_ver_gen_i.v
        |               |               |               |-----channel_err_detect_i.v
        |               |               |-----tx_ll_i.v
        |               |               |               |-----tx_ll_datapath_i.v
        |               |               |               |-----tx_ll_controlpath_i.v
        |               |               |-----rx_ll_i.v
        |               |               |               |-----rx_ll_pdu_datapath_i.v
        |               |------dut1-mkSimpleNetwork_1.v
        |               |
        |               |
```

```
|                |
|                | ---- Example_design_2_i.v
|                |-------clock)module_i.v
|                |-------standard_cc_module.v
|                |-------reset_logic.v
|                |-------aurora_module_i_2
|                |                |-----aurora_lane_0_i.v
|                |                |                |-----lane_init_sm_i.v
|                |                |                |-----chbond_count_dec_i.v
|                |                |                |-----sym_gen_i.v
|                |                |                |-----sym_dec_i.v
|                |                |                |-----err_detect_i.v
|                |                |-----gtp_wrapper.v
|                |                |                |-----GTP_TILE_INST.v
|                |                |-----global _logic.v
|                |                |                |-----channel_init_sm_i.v
|                |                |                |-----idle_and_ver_gen_i.v
|                |                |                |-----channel_err_detect_i.v
|                |                |-----tx_ll_i.v
|                |                |                |-----tx_ll_datapath_i
|                |                |                |-----tx_ll_controlpath_i
|                |                |-----rx_ll_i.v
|                |                |                |-----rx_ll_pdu_datapath_i
|                |-------aurora_module_i_1
|                |                |-----aurora_lane_0_i.v
|                |                |                |-----lane_init_sm_i.v
|                |                |                |-----chbond_count_dec_i.v
|                |                |                |-----sym_gen_i.v
|                |                |                |-----sym_dec_i.v
|                |                |                |-----err_detect_i.v
|                |                |-----gtp_wrapper.v
|                |                |                |-----GTP_TILE_INST.v
|                |                |-----global _logic.v
|                |                |                |-----channel_init_sm_i.v
|                |                |                |-----idle_and_ver_gen_i.v
|                |                |                |-----channel_err_detect_i.v
|                |                |-----tx_ll_i.v
|                |                |                |-----tx_ll_datapath_i.v
|                |                |                |-----tx_ll_controlpath_i.v
|                |                |-----rx_ll_i.v
|                |                |                |-----rx_ll_pdu_datapath_i.v
|                |------dut1-mkSimpleNetwork_2.v
|                |
|                |
```

References:

1. M. K. Papamichael and J. C. Hoe, “Connect: Re-examining conventional wisdom for designing NoCs in the context of FPGAs,” in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays. ACM, 2012, pp. 37–46.
2. Abhijit Athavale and Carl Christensen, "High-Speed Serial I/O Made Simple A Designers' Guide, with FPGA Applications", Connectivity Solutions Edition 1.0.
3. Yatish Turakhia, "Multi-FPGA Hardware Acceleration of Boolean Matrix Vector Multiplication using Network-on-Chip", Master Thesis report 2013-2014.
4. LogiCORE IP Aurora 8B/10B v5.3 User Guide UG353 January 18, 2012
5. Abdellah-Medjadji, Kouadri-Mostefaoui, Benaoumeur Senouci and Frederic Petrot Scalable, "Multi-FPGA Platform for Networks-On-Chip Emulation", TIMA Laboratory, System-Level Synthesis Group 46.
6. Abdellah-Medjadji, Kouadri-Mostefaoui, Benaoumeur Senouci and Frederic Petrot “Scalable, Large Scale On-Chip Networks: An Accurate Multi-FPGA Emulation Platform", TIMA Laboratory, System-Level Synthesis Group 46.
7. Marta Stepniewska, Adam Luczak and Jakub Siast, "Network-on-Multi-Chip (NoMC) for multi-FPGA multimedia systems" Chair of Multimedia Telecommunications and Microelectronics, Poznan University of Technology, 2010 13th Euro Micro Conference on Digital System Design: Architectures, Methods and Tools
8. M. Gschwind, V. Salapura, and D. Maurer, “FPGA prototyping of a RISC processor core for embedded applications,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 9, no. 2, pp. 241–250, 2001.
9. J. Ray and J. C. Hoe, “High-level modeling and FPGA prototyping of microprocessors,” in Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays. ACM, 2003, pp. 100–107.
10. R. E. Wunderlich and J. C. Hoe, “In-system FPGA prototyping of itanium microarchitecture,” in Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on. IEEE, 2004, pp. 288–294.
11. U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, “Voltage-frequency island partitioning for gals-based networks-on-chip,” in Design Automation Conference, 2007. DAC’07. 44th ACM/IEEE. IEEE, 2007, pp. 110–115.
12. W. J. Dally, C. Malachowsky, and S. W. Keckler, “21st century digital design tools,” in Proceedings of the 50th Annual Design Automation Conference. ACM, 2013, p. 94.
13. W. J. Dally and B. Towles, “Route packets, not wires: On-chip interconnection networks,” in Design Automation Conference, 2001. Proceedings. IEEE, 2001, pp. 684–689.
14. S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, “A network on chip architecture and design methodology,” in VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on. IEEE, 2002, pp. 105–112.
15. U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, “Voltage-frequency island partitioning for gals-based networks-on-chip,” in Design Automation Conference, 2007. DAC’07. 44th ACM/IEEE. IEEE, 2007, pp. 110–115.
16. J. P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, “Noc-centric security of reconfigurable soc,” in Proceedings of the First International Symposium on Networkson Chip. IEEE Computer Society, 2007, pp. 223–232.
17. L.-S. Peh, S. W. Keckler, and S. Vangal, “On-chip networks for multicore systems,” in Multicore Processors and Systems. Springer, 2009, pp. 35–71.