UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

ANALYSIS OF ASYNCHRONOUS ROUTERS FOR NETWORK-ON-CHIP APPLICATIONS

A Bachelor's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Roshni Maniraj

In partial fulfilment

of the requirements for the degree of

BACHELOR IN ELECTRONICS AND COMMUNICATION ENGINEERING

Advisor: Dr. Sergi Abadal

Barcelona, July 2019

Title of the thesis: Analysis for Asynchronous Routers for Network-on-Chip Applications

Author: Roshni Maniraj

Advisor: Dr Sergi Abadal

Abstract

The asynchronous circuit design has been conventionally regarded as a valid alternative to synchronous logic due to its potential for low consumption of resources, power and delay. This includes areas such as the communication infrastructure of modern multi-core processors, the so-called Network-on-Chip (NoC) paradigm on which this thesis focuses on. In recent times, the transistor downscaling and the increasing clock frequencies have pushed synchronous design to high static power and delay. As a result, the interest for asynchronous integrated routers and links has re-emerged, especially in fields with ultra-low power requirements such as embedded systems. In this thesis, we construct an asynchronous router using Verilog code based on architectures found in the literature. We analyze the functionality of each of the building blocks and verify the operation of the implemented routing algorithm and an arbitration mechanism. In the future, the results obtained here are expected to enable a complete implementation of the router in Verilog and its posterior analysis of its scalability.

Acknowledgements

I would like to express my sincere gratitude to Dr Sergi Abadal who through the entire course of the project was extremely patient and helpful. His guidance was the most valuable and this project's success is all thanks to him.

I would also like to thank Amrita University to give me this opportunity to participate in the student abroad program that gave me a chance to take this project up.

I would like to thank UPC, ETSETB for hosting me and allowing me to do my bachelor thesis in their esteemed institution

Last but not least I would like to acknowledge and thank my family and their efforts. Their constant love and support was the reason I was able to do all this.

Revision history and approval record

Revision	Date	Purpose
0	27/06/2019	Document creation
1	03/07/2019	Document revision
2	04/07/2019	Final version

Written by: Roshni Maniraj		Reviewed and approved by:	
Date	04/07/2019	Date	05/07/2019
Name	Roshni Maniraj	Name	Dr Sergi Abadal
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

Asynchronous logic design has been around for quite a while and is considered a good alternative to its synchronous counterpart. This is because of its ability to perform tasks without the need for a global common clock. Asynchronous logic only consumes dynamic power when the value of signals change, which is opposed to the behaviour of synchronous logic that consumes power in each clock tick. Therefore, asynchronous logic lowers the use of system power and latency, leading to higher performance and efficiency. As the size of the transistor reduces and the number of transistors on the chip increase in accordance with Moore's law, the degradation of the synchronous design is evident with shot noise, charge sharing, thermal effects, supply voltage noise and process variations all making these advantages even more appealing.

The synchronous design always calculates the worst case values whereas the asynchronous designs provide us with the average values. This proves to be very helpful at 90nm technology and below. The asynchronous designs also take less area on the chip due to the absence of the bulky clock design which is an integral part of the synchronous design [1].

A particular area where the asynchronous design shines is Network-on-Chip (NoC). NoC has become the new normal for structured on-chip communication for low powered embedded chips as well as high-powered multiprocessors. It is slowly replacing traditional bus-based communication with packet switching integrated networks [21].

As technology advances, the limits of on-chip transistors are pushed, integrating up to a thousand cores on the same chip, making the design extremely energy limited. Thus the need for energy efficient designs. Also, the demands of the market for super fast and efficient communication leads us to asynchronous systems [23].

GALS network or *globally asynchronous locally synchronous* networks is a hybrid of synchronous components in asynchronous design. The synchronous components include the cores, memory units, accelerators, I/O units etc. GALS allows us to use the best of both designs. The elimination of global clock provides highly scalable, low power robust mechanism which proves useful for assembling complex systems [1,14].

1.1 Requirements and Specifications:

Asynchronous designs are claimed to be better than synchronous design in terms of cell area, latency and power. In this thesis, an asynchronous router is built using the structure from [1].

The goal is to have routers and links that can be integrated into many-core processors. Let us assume that a complete 100-core processor is implemented in a die of dimensions 20x20 mm^2, that are the usual values. Moreover, let us assume that the power consumption is 120 W, at maximum. Assuming equal area and power budget for the processing elements, the memory, and the network, then the network must occupy less than 133 mm^2 and use less than 40 W. In a 100-core case, assuming that each core contains a router, the calculations above lead to approximately 1.33 mm^2 and 400 mW per router as the crucial requirements for our design.

A simple and widely used NoC architecture is assumed. The network topology of the router is a 2D mesh with wormhole flow control. The routers have no virtual channels. The routing algorithm used is dimension-order (XY) routing where first the X moves are performed then the Y moves. The destination address is kept in the head flit payload. For all this, the router needs to have five ports (north, south, east, west, and local) and we confirm that each router is attached to a core.

From Imai et al [1] the comparison is done between synchronous routers with and without clock gating optimization and asynchronous routers linear and circular FIFO. This is done with two types of architectures i.e 1mm and 2mm routing.

Cell area comparison: we see that the area taken by asynchronous linear FIFO is less than the one with circular FIFO. The cell area of a synchronous router without clock gating optimization is 1.44 times an asynchronous linear FIFO and almost the same as circular FIFO router. On the other hand without the clock gating optimization shows us the cell area is 1.80 times that of linear FIFO router and 1.25 times the circular FIFO router. This is very helpful in proving that the asynchronous router takes less area in a circuit design. Similar results were observed with 2mm routing. From [1] the values of cell area of an asynchronous routers for are 675272 μm^2(linear) and 976179 μm^2(circular) for 1mm routing.

Latency comparison: the latency from the input port to the output port for a synchronous router and average latency for asynchronous router is listed. The values in table 2 illustrate the values showing a significant reduction in latencies of the circular FIFO routers when compared to the synchronous routers. The latency values of the asynchronous linear FIFO and synchronous router with clock gating optimization are comparable for 1mm place and route.

	Synchronous no clk gt	Synchronous with clk gt	Asynchronous linear FIFO	Asynchronous circular FIFO
1mm place and route	4.74 ns	4.86 ns	4.84 ns	4.22 ns
2mm place and route	5.48 ns	6.74 ns	5.13 ns	4.43 ns

Table 2: Latency values for synchronous and asynchronous routers from Imai et al [1]

Power Consumption: The comparison is done in three scenarios (i) zero load situation (ii) low packet injection and (iii) high packet injection. In the zero load situation, we observe that asynchronous router have almost no power consumption and the synchronous router consumes 211mW of power in 1mm place and route and 178.2mW power in 2mm place and route. During the high packet injection, the values of the routers are almost the same because they saturate. With these results in [1], we decided to implement their design of the asynchronous router.

1.2 Statement of purpose (Objective):

The main objective of this thesis is to construct an asynchronous router using Verilog code based on architectures found in the literature that comply with the specifications set above. We have confirmed that the design from [1] fulfils the specifications and we choose to implement it (the area is 0.97617mm^2 meeting our requirement of 1.33mm^2; and the power is approximately 100 mW at 20 mHz of packet injection rate, meeting our requirement of 400 mW). Moreover, the technology used in [1] is relatively old (130nm CMOS) and therefore the specifications can be even improved. We aim to analyze the functionality of each of the building blocks and verify the operation of the implemented routing algorithm and an arbitration mechanism. It is also our aim to build the router using Verilog and open-source EDA tools as much as possible. In the future, the results obtained here are expected to enable a complete implementation of the router in Verilog and its posterior analysis of its scalability.

1.3 Methods and procedures:

The code for the router modules was written in Verilog HDL. The applications used were all open source. Notepad++ was used to type the code out, iVerilog was used to compile the code and GTKWave was used to view the waveform. The router architecture is provided by [1].

1.4 Work plan:

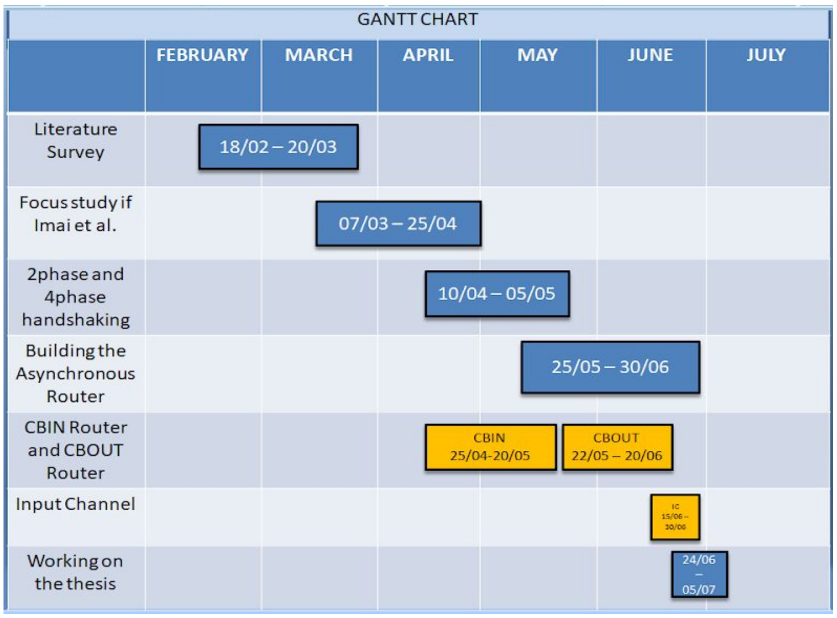


Figure 1: Gantt Chart

The Gantt chart displays the timeline of the project. The project started of literature survey for the first month from 18th February till 20th March. In that time we decided to focus on the asynchronous router architecture mentioned in Imai et al [1] and replicate that on Verilog. The month of April was dedicated to working on the handshaking protocols. After the execution of 2 and 4 phase protocol on Verilog, we started working on the different modules of the asynchronous router. This took two months to do. We worked on the code until the end of June. From 24th June to the 5th of July

1.5 Deviations from the initial plan:

In the beginning, during the project planning, we decided to build the complete asynchronous router and make it work as a standalone unit. Then chart its area power and latency using available CAD tools. We had to narrow the scope at the midterm status update because the time was too tight to implement each component of the asynchronous router. We underestimated the time required to successfully interconnect the different modules and work them as a standalone unit.

2. Background

2.1 Synchronous logic design

2.1.1 Synchronous designs: What are synchronous designs

Commercial digital systems usually use Synchronous design because of its simplicity and stability. The communicating systems in a synchronous design usually operate with a global clock. All the computations follow the clock for timing in the system, this greatly simplifies the computations. All input and output signals and internal nodes are stabilized in the high or low state on the active edge of the clock. Between the fall and rise of the clock, the signals and nodes are allowed to change and may take any intermediate state. The behaviour of a synchronous network is predictable and is programmed not to fail due to hazards or glitches introduced by irregularities of the real circuit.

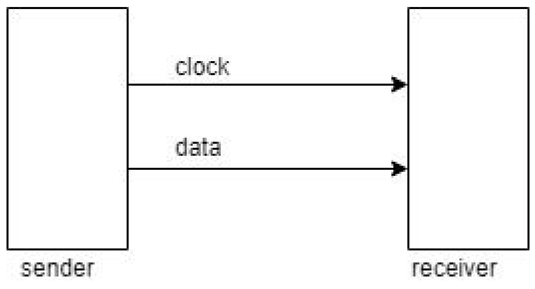


Figure 2: Synchronous Communication

2.1.2 Synchronous designs: Their challenges.

The global clock mechanism simplifies the circuit design, but it also has its own share of challenges. For these circuits to perform correctly, a great deal of care is needed in the design of the clock distribution networks. Static timing analysis is most often used to decide upon the maximum safe operating speed. Moreover, the clock distribution network can consume a significant amount of power in manycore NoCs [23].

Synchronous systems often slow down their circuits to accommodate the clock skew. As the feature constantly reduces in size, clock skew becomes a topic of greater concern.

Standard synchronous circuits have to toggle clock lines, and possibly pre-charge and discharge signals, in portions of a circuit unused in the current computation.

For example, even though a floating point unit on a processor might not be used in a given instruction stream, the unit must still be operated by the clock.

Synchronous circuits yield worst-case performance because they must wait until all possible computations in the module have completed before latching the results.

In systems such as a synchronous microprocessor, the system clock, and system performance is dictated by the slowest (critical) path. This affects the system performance because it has to slow down and wait thus consuming more power and increasing the latency of the system.

Integrated circuits will often be implemented in several different technologies during their lifetime. Early systems may be implemented with gate arrays, while the later may migrate to semi-custom or custom ICs. Better performance for synchronous systems can often only be achieved by migrating all system components to a new technology since the overall system performance is based on its longest path. The delay through a circuit fluctuates with changes in fabrication, power-supply voltage and temperature variations. Synchronous circuits always assume the worst case of factors and clock the system accordingly.

Synchronous circuits require all its elements to exhibit bounded response time. There are some chances that mutual exclusion circuits will fail in a synchronous system.

2.2 Asynchronous logic design

2.2.1 Asynchronous designs: What are asynchronous designs

Asynchronous designs are clock-less systems where the different modules do not synchronise its computations according to a single global clock. Asynchronous systems do not depend on strict arrival times of signals or messages for reliable operation. As a result, the different processing elements in the system are free to operate at different speeds. In an asynchronous design, the elements communicate with each other using local handshaking techniques, resulting in systems being faster and power efficient. That’s why asynchronous designs are so alluring to researchers, but with the changes in technology we are faced with its own set of challenges.

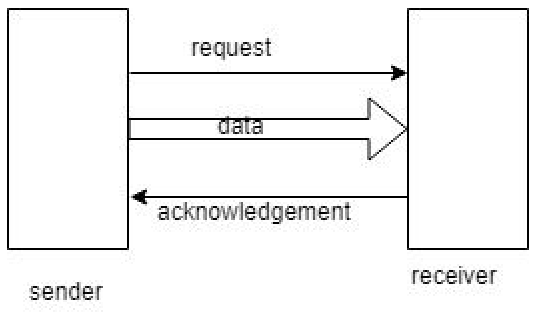


Figure 3: Asynchronous Communication

2.2.2 Asynchronous Designs: History

Asynchronous design is not a new technology, it has been around for a long time. Around the 1950s to the early 1970s, the early years for the asynchronous design included the development of the classical theory (Huffman, Unger, McCluskey, Muller), and the commercial use of asynchronous processors. The middle years, from the mid-1970s to early 1980s, was a time of reduced activity, corresponding to the advent of the synchronous VLSI era. The mid-1980s to late 1990s was when the asynchronous designs were coming back with the beginning of modern methodologies for the asynchronous controller and pipeline design, initial computer-aided design tools and optimization techniques. This includes the first academic microprocessors (Caltech, University of Manchester, Tokyo Institute of Technology), low-power commercial products (Philips Semiconductors) and high-performance interconnection networks (Myricom). From the 2000s to the present deemed as the modern era that includes modernization of design approaches, CAD tool development and systematic optimization techniques, migration into on-chip interconnection networks, several large-scale demonstrations of cost benefits, industrial uptake at leading companies (IBM, Intel) as well as startups, and application to emerging technologies (sub-/near threshold circuits, sensor networks, energy harvesting, cellular automata) [6].

2.2.3 Asynchronous designs: How they help to solve the challenges of synchronous designs

The synchronous design is a system powered by a global clock which makes the system stable. The challenges of the synchronous designs as listed above are tackled by the asynchronous clock-less designs.

The reason for lower power consumption by the asynchronous designs compared to its synchronous counterpart is due to the absence of a global clock. Although asynchronous circuits often require more transitions on the computation path than synchronous circuits, they generally have transitions only in areas involved in the current computation.

The average-case is taken into account, unlike the synchronous designs which take the worst–case performance. Many asynchronous systems sense when the computation has completed, allowing them to exhibit average-case performance. For circuits such as ripple-carry adders where the worst-case delay is significantly worse than the average-case delay, this can result in substantial savings.

Since many asynchronous systems operate at the speed of the circuit path currently in operation, rarely used portions of the circuit can be left un-optimized without adversely affecting system performance thus easing the global timing issues [27].

In most of the asynchronous systems, because performance is dependent on only the currently active path migration of only the most critical system components can improve system performance on an average. Also, since many asynchronous systems sense computation completion, components with different delays may often be substituted into a system without altering other elements or structures.

Asynchronous systems can wait for an arbitrary amount of time for an element in the system to complete, allowing robust mutual exclusion. Also, since there is no clock with which signals have to be synchronized, asynchronous circuits better accommodate inputs from the outside world, which are by nature asynchronous.

2.2.4 Asynchronous designs: Commercial applications and Industrial Experiments

Philips Semiconductors (now NXP) used asynchronous microcontroller 805C1 initially as a pager chipset to lower the electromagnetic noise emission so that it could operate with Radio Frequency (RF) data without the use of shielding. This resulted in eliminating the need for a fixed function circuit and encoded the RF data in software. This also showed a significant decrease in power usage. Later this microcontroller started being used in smart cards for public transport. Now the new improved microcontroller (SmartMX) is used in biometric systems and ID's in more than 75 countries including the European Union and the United States [14].

Intel acquired Fulcrum Microsystems, a startup working on asynchronous high-speed networking chips. Intel's FM5000/FM6000 are a family of switch chips which support the 40 gigabit Ethernet that includes a fully asynchronous high-speed crossbar switch which provides low latency, high energy efficiency and bandwidth, flexible link topologies support. The high speed is achieved by fine-grain asynchronous pipelining at individual stages. When operated at a below peak throughput they are highly energy- efficient [3,14].

Achronix semiconductor's Speedster 22i is a family of FPGAs that can operate at a speed of 1.5GHz. They claim to be the world's fastest FPGA yet cost a lot less than their synchronous counterparts in terms of operating energy and design. They achieve such fast operation by asynchronous fine-grain bit-level pipelines hence overcoming the need for global synchronization [3].

Several industrial experiments with asynchronous design have been done successfully, but never implemented commercially such as Intel RAPPID and IBM FIR Filter. There are multiple areas of application for an asynchronous design still being researched like large scale heterogeneous system integration, energy harvesting and Ultra-low-energy systems, handling the extreme environment and alternate computing approaches. There is a lot of effort being put into asynchronous and mixed synchronous-asynchronous systems like "GALS system" and "Networks-on-Chip". The emerging current technologies such as nano-arrays and nano-magnetics, with highly robust asynchronous designs, prove to be vital to improve the timing irregularities [3,14].

2.2.5 Asynchronous Design: Links and Handshaking

The communication channel of an asynchronous router involves a request (*req*) wire and an acknowledge (*ack*) wire. The *req* wire shows us when the data sent by the sender is valid, and the *ack* wire indicates that the receiver successfully received the data. The two most common handshaking protocols used for the communication channel. The first is a 4-phase (return-to-zero (RZ) protocol), and the second is a 2-phase (non-return-to zero (NRZ) protocol) [3]. Section 5.2 provides more details on the different signalling mechanisms and encoding.

2.2.6 Asynchronous Design: Challenges of Asynchronous Design

There has been a large amount of work in addressing various challenges of asynchronous design since the asynchronous circuits are much more difficult to design than synchronous circuits.

In a synchronous system, a designer has to merely outline the combinational logic necessary to compute the given function and surround it with latches. By setting the clock rate for a longer period, all worries about hazards and the dynamic state of the circuit are removed. On the contrary, designers of asynchronous systems must focus on the dynamic state of the circuit. To avoid incorrect results, hazards must also be physically removed from the circuit, or not introduced in the first place. The order of operations, which was fixed by the placement of latches in a synchronous system, must be ensured by the designer of the asynchronous control logic. For the present ultra-complex systems, these issues become extremely difficult to handle manually.

Asynchronous circuits, unfortunately, cannot use the existing CAD tools that are specially designed to be used for the synthesis of synchronous circuits. For example, some asynchronous methodologies allow only algebraic manipulations (associative, commutative, and De-Morgan's Law) for logic decomposition. Placement, routing, partitioning, logic synthesis, and most other CAD tools either need modifications for designing asynchronous circuits, or they are not applicable at all.

The comparison between asynchronous and synchronous circuits started off with the claim that asynchronous have faster computation speed compared to synchronous systems, but this hasn’t been proven yet. Asynchronous circuits generally require extra time due to their signalling policies, thus increasing the average-case delay.

Testing of asynchronous circuits faces challenges compared to synchronous designs. A typical testing procedure for synchronous designs is called single-stepped approach which involves pausing or slowing down the system, and checking the internal states. However, this testing approach is not possible for asynchronous designs due to the absence of a global clock. The testing tools for asynchronous designs should not only check for functional correctness, it should also check for hazards, which adds to the complications.

2.3 Network-on-Chip (NoC)

2.3.1 NoC: What are NoCs

Network-on-Chip is a network-based communications subsystem on an integrated circuit, most typically between modules in a system on a chip (SoC)[6]. The modules on the IC are typically semiconductor IP cores schematizing various functions of the computer system and are designed to be modular in the sense of network science.

NoC technology applies the theory and methods of computer networking to on-chip communication and contributes to improvements over conventional bus and crossbar communication architectures. Networks-on-chip improves the scalability of systems-on-chip and the power efficiency of complex SoCs. A very common NoC used in contemporary personal computers is a graphics processing unit (GPU), which is used in computer graphics, video gaming and accelerating artificial intelligence.

Over the span of the last decade, Networks-on-Chip (NoCs) have become the standard approach for a structured on-chip communication, for low-power embedded systems as well as high-performance chip multi-processors.

These on-chip networks typically replace traditional bus-based communication with packet switching and can be targeted to a variety of cost functions (fault-tolerance, power, latency, saturation throughput, quality-of-service [QoS]) and parameters (network topology, channel width, routing strategies).

2.3.2 NoC: Motivation behind using an NoC.

On-chip interconnects have become the limiting factor to achieve high performance and low power for the current multi-core technology mainly due to two reasons: (i) the system cores operate at different clock frequencies, thus the need for reliable and efficient interconnects is crucial to maintain error-free interactions between the different timing domains (ii) technology scaling has made computational elements and memories faster and more energy efficient, but the interconnects used to have the same performance and power, and not been changed with the change in technology. These issues prove the need for NoCs in the current technology standpoint.

NoC provides a distributed communication infrastructure, consisting of switches and channels. Each processing element is connected to the switch through a network interface, and the switches are in turn connected to each other using channels or links. The switches and channels are organized in a fixed structure called a topology, which can be of different types, e.g. mesh, ring, etc

2.3.3 NoC: Advantages of NoC

NoCs support modularity by separating communication from the computation. They help in decreasing design efforts by facilitating design reuse thus allowing faster testing and validation resulting in improvement in the overall design cycle.

NoCs allow sharing the communication infrastructure facilitating parallel and distributed traffic flow. This leads to faster performance, without the need for extra wiring resources for dedicated pathways, decreasing the area used and power overheads.

The bandwidth in traditional global buses is limited, which is shared by all the attached units and suffers when the number of units increases. The bandwidth of the NoC depends on the scaling of the network. NoCs have regular architectures, with short wires that have controlled and predictable electrical properties, leading to a more reliable operation compared to global long wires.

2.3.4 NoC: The advances of synchronous NoCs

The synchronous design is the most common style of NoC design. Since synchronous NoCs have been around from the early 2000s, there have been significant advances have been made in this area.

Many different topologies have been proposed and used in the NoCs. The most common of the topologies are Mesh, Ring, Torus and Trees. For high-performance computing, high radix topologies such as Dragon-fly is proposed.

Routing Algorithms for NoCs are divided into two categories:(i) deterministic routing: where the path is fixed for the packet (ii) adaptive routing: where the best path is dynamically selected depending on the congestion in the network.

Many power and performance optimizations have been introduced in the NoCs. To minimize power, router techniques such as dynamic voltage and frequency scaling (DVFS) have been used. To improve performance, optimization techniques such as speculation [20], prediction [21] and bypassing or lookahead [22] have been used within routers [3].

Several approaches have been proposed to support patterns involving multicast and aggregation while simultaneously achieving high performance with low overheads. There has also been significant research to support communication patterns common in parallel computing applications, such as cache coherency, and emerging areas of deep neural network architectures.

2.3.5 NoC: Advances in asynchronous NoC design

The NoC approach separates the communication infrastructure and timing, from its processing elements, it seems like a natural match with the asynchronous paradigm. Asynchronous interconnect eliminates the need for global clock management across a large network structure. Power and performance benefits of asynchronous NoCs have been demonstrated for high-performance shared-memory chip multi-processors [11] and Ethernet switch chips [12] as well as their facilitation of extreme fine-grain power management and flexible integration of many-core GALS architectures. The end-to-end latency benefits of asynchronous NoCs over synchronous NoCs have also been demonstrated

[8,9,11,12] due to the low forward latency of individual asynchronous router nodes, and the ability of packets to advance without a clock. As a recent example, an asynchronous NoC switch architecture [9] using single-rail bundled data and two-phase communication, obtained a significant reduction in average energy-per-packet and area compared to a highly-optimized synchronous single-cycle NoC switch in the same 40nm technology.

3. State of the art

Multiple research challenges for asynchronous NoCs have been targeted. To achieve the quality of service (QoS), asynchronous NoCs have been proposed that provide guaranteed service and multiple levels of services, in addition to best effort traffic. There has been important research on improving fault-tolerance and reliability of asynchronous NoCs with some works focusing on developing efficient asynchronous NoCs that mitigate the effects of process variation [3].

Earliest established works on asynchronous routing for NoC dates back to the mid-2000s[16]. Since then, improvements in this field have constantly been made allowing designs of asynchronous routers with functionalities similar to their sophisticated synchronous counterparts. For instance, Horak *et al.* set the foundation for two-phase, single-rail bundled-data routers with wormhole routing capabilities and evaluated the critical components separately [17]. After that, Jiang *et al*. proposed and evaluated a router with virtual channel capabilities, which are achieved by a replication of the internal switch rather than the input buffers [18].

Numerous researchers have implemented and made a comparison between their asynchronous router design with existing equivalent synchronous alternatives. Table 1 summarizes their main characteristics and the results of the cost and performance comparison that includes area, power, throughput and latency.

The four analyzed works evaluate place-and-routed designs at very different technology points, namely, from 14nm to 130nm, yet the results are consistent throughout. Comparing the costs, asynchronous routers are much more lightweight than the synchronous ones. Cell area is reduced due to the simplicity of the router architecture and the use of latches instead of flip-flops in all the buffers, whereas lower power usage is due to the virtual lack of static power consumed by the clock.

Reference	Technology	Design	Latency	Throughput	Area	Power
[83]	65 nm	4P, BD, 3 ports, 1 VC	**-5%**	N/A	**-80%**	**-44%**
[1]	130 nm	2P, BD, 5 ports, 1 VC	**-12%**	-20%	**-45%**	**-24%**
[82]	14 nm	2P, BD, 5-7 ports, 2-8 VC	**-28%**	N/A	**-55%**	**-58%**
[9]	40 nm	2P, BD, 5 ports, 1 VC	**-10%**	Similar	**-71%**	**-45%**

Table 2: Asynchronous on-chip routers in the literature. Performance and cost numbers are compared to synchronous counterparts. 2P and 4P mean 2-phase and 4-phase handshaking. BD means bundled-data

4. Methodology/Project Development

For building an asynchronous router, I used simple Verilog code. The compiler is an open-source software called iVerilog and the code was written on Notepad++. The waveforms for the output were generated on Gtkwave which is also an open source software.

Verilog is a hardware description language used to model electronic systems. It supports design, testing and implementation of digital and mixed-signal circuits at different levels of abstraction. It is similar to C programming with a few crucial differences such as timing and hierarchical execution of models.

The code is typed out on Notepad++ which is a free open-source code editor. It is a notepad replacement and supports several languages. Notepad++ 's build ensures a higher execution speed and compact program size [25].

Iverilog or Icarus Verilog is an open source Verilog simulation and synthesis tool that operates as a compiler that compiles the Verilog code into some target format. For synthesis, the compiler generates the netlists as desired [24].

GTKWave is an open source fully featured GTK+ based wave viewer which reads standard Verilog .vcd/.evcd files and allows viewing the waves [26].

The Verilog code is written on Notepad++ and then the file is saved as a ".vl" file, recognized by the iverilog compiler. Once the file is saved in the iverilog folder, shift-right-click and choose "Windows PowerShell". Instruction "iverilog -o filename filename.vl is given to compile the code in the given file. If there are errors/ bugs in the code, they are listed in the window alongside the line which has the error which makes debugging easier. If there are no errors then "vvp filename" is used to give the output for the given code.

If the output of the code is in waveforms then the file has the following code:

"$dumpfile("filename.vcd");"
"$dumpvars;"

The above line of code creates a .vcd file which is the GTKWave format to view the output waveforms. If we want to view the output of the code in text form and not waveform then the following code is used

"$monitor($time," parameters = %b", parameters);"

“#time $finish;”

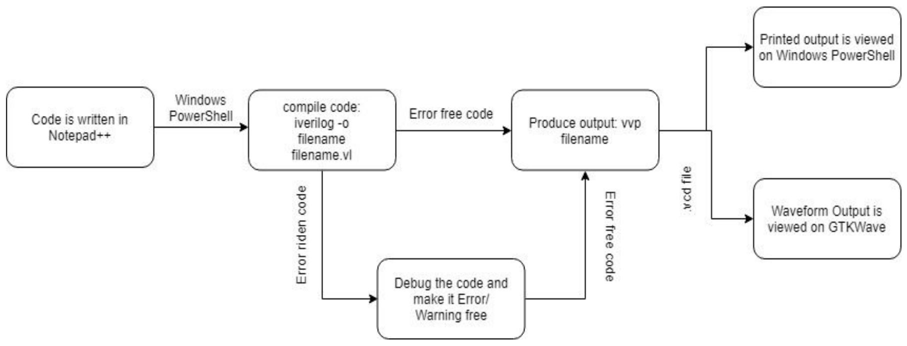

Code is written in Notepad++
Windows PowerShell
compile code: iverilog -o filename filename.vl
Error free code
Produce output: vvp filename
Printed output is viewed on Windows PowerShell
vcd file
Waveform Output is viewed on GTKWave
Error riden code
Debug the code and make it Error/ Warning free
Error free code

Figure 4: Flowchart of executing Verilog code and observing the outputs

5. Architecture:

5.1 Network

The NoC is made up of Routers, network interface (NI), IPs and links. These are the main elements of NoC architecture. For the same example, let us assume that the components are connected in (4×4) Mesh topology shown in Figure 2. The most important features of NoC architecture are routing algorithm, network topology, and switching techniques. As for the network itself; the router is the most important element in SoC based on NoC architecture. Network Interfaces connect the IP cores to the on-chip router network. Network Interface in an NoC is the medium between the computational part and communication infrastructure. Network Interfaces exchange the data generated by the IP blocks into data packets and place extra routing information based on the underlying NoC network.

Mesh network topology consists of n rows and m columns. Each router in a mesh topology is connected to the adjacent router through the interconnection of wires. This is the simplest topology to implement. The faults and can easily be detected and the faulty nodes can be avoided while routing the packets to its destination.

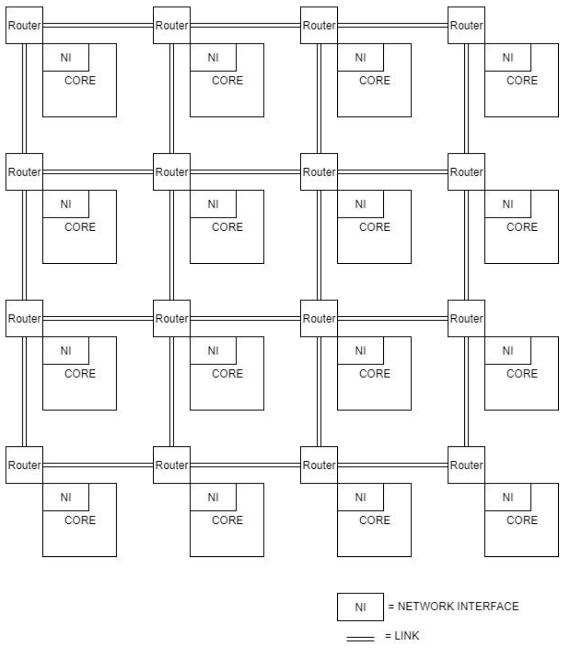


Figure 5: Network Architecture - Mesh Topology

The following paragraphs describe the asynchronous links and Handshaking protocols adopted by the asynchronous router. The router architecture is discussed and the router modules are dealt with in great detail.

5.2 Links and Handshaking

Asynchronous circuits rely on a certain request-acknowledgement-data transmission strategy since it cannot assume the availability of a channel beforehand. Asynchronous schemes greatly depend on well-constructed handshaking protocols and designs [14].

Handshaking circuits can be modelled using popular CAD tools such as Cadence suits. The handshaking protocol can easily be defined by simple VHDL/Verilog code and then the delay and cost overheads can be obtained by instantiating the code in a circuit simulator environment. For high-level behavioural models, tools such as DSENT can be used.

Setting up a link:

The asynchronous design employs two types of handshaking protocols (i) two-phase protocol (ii) four-phase protocol. Both of these protocols require a request for transmission (req) and acknowledging the request(ack). The handshaking protocol is repeated for every flit of the transmission.

The 4 phase protocol or return-to-zero (RZ), the req and ack signal start at 0. The transmitter sends the data and sets the req signal to high. The receiver then absorbs the data and sets the *ack* to high when done. When the transmitter sees the *ack* signal, the *req* signal is set to 0 (when the data's validity is no longer guaranteed). The receiver then sets the *ack* to zero, thus returning to the initial state. After this, a new connection can be formed for another cycle.

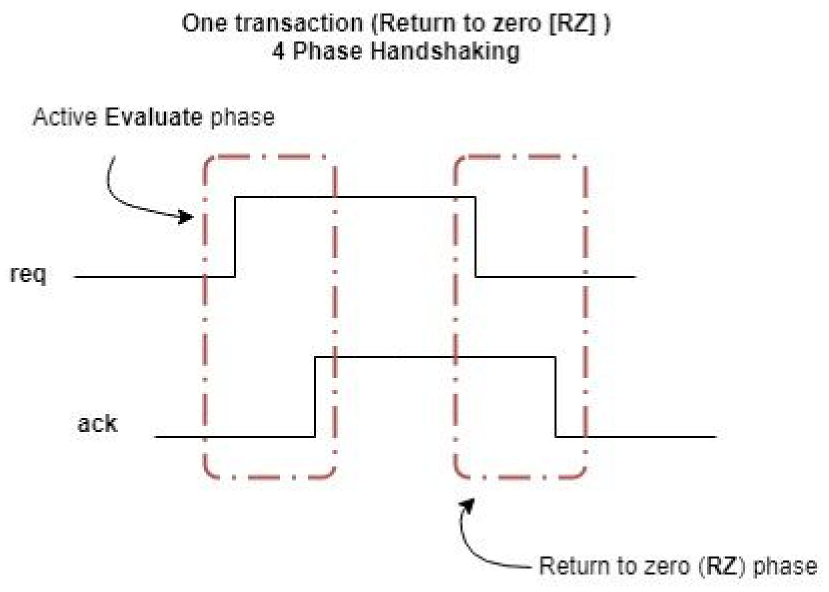

One transaction (Return to zero [RZ])
4 Phase Handshaking
Active Evaluate phase
req
ack
Return to zero (RZ) phase

Figure 6: four-phase handshaking protocol

The 2-phase protocol or non-return-to-zero (NRZ) does not return to the initial state. Both the transmitter and receiver leave the signals *req* and *ack* unchanged after the transmission is completed. The signal for the next transmission is toggling the *req* signal (0 to 1 or from 1 to 0) and waiting for the receiver to do the same with the *ack*.

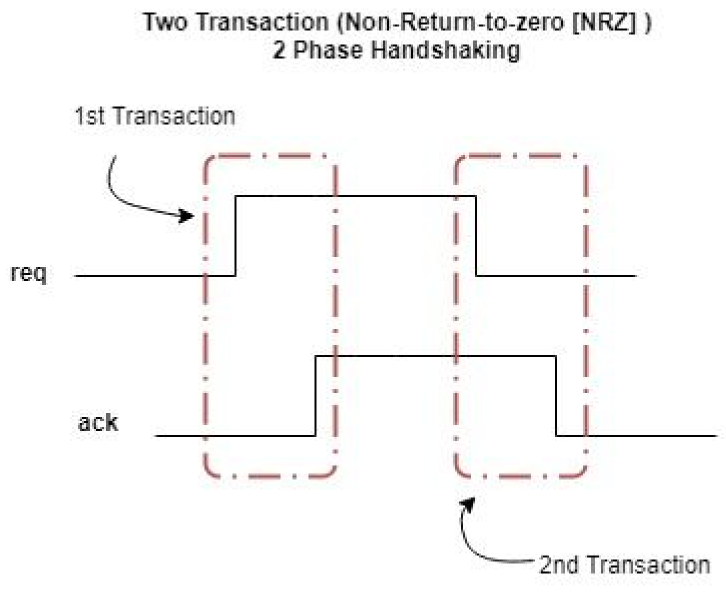
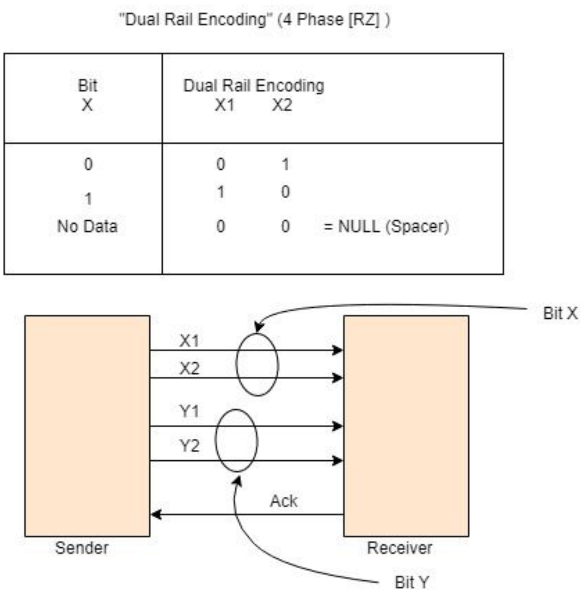


Figure 7: Two-phase Handshaking protocol

Data encoding and transmission:

Data transmission occurs after handshaking completes. There are two known types of data encoding: dual rail or single-rail bundled encoding, both are illustrated in Figures 8,9.

Dual rail encoding: Two wires are required per transmitted bit. Zero in both wires indicate that no data is present, whereas if the first or second wire is high means transmitting a ‘0’ or ‘1’. This transition serves as *req* signal and ack is sent using a dedicated wire. Dual rail encoding improves the robustness to the different delays that are found between the transmitter and the receiver. This robust nature is at the cost of considerable area and power overhead (presence of two wires for transmission). Efficient codes have been developed to reduce the cost [14].


"Dual Rail Encoding" (4 Phase [RZ])

Bit X	Dual Rail Encoding X1	X2	
0	0	1	
1	1	0	
No Data	0	0	= NULL (Spacer)

Figure 8: Dual Rail encoding of data

Single rail bundled encoding: The data transmission resembles that of synchronous systems. Req and ack have their own dedicated lines and each bit of data is transported through a single wire. Whenever the request has been acknowledged, data is transmitted through each data wire. Data should be stable from the time before the *req* signal is received to after the *ack* bit is set. Also, the delay of the *req* line has to be longer than any of the data ones.

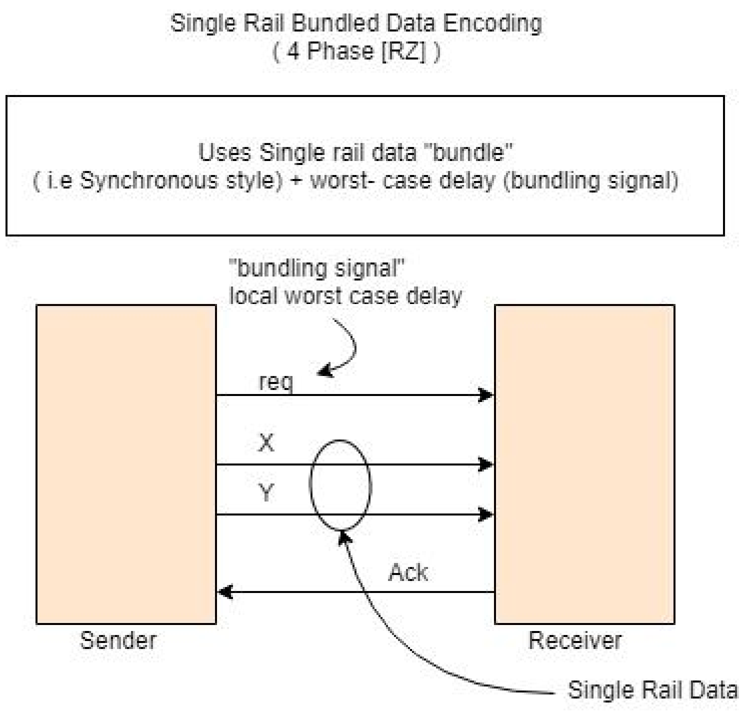

Single Rail Bundled Data Encoding
(4 Phase [RZ])
Uses Single rail data "bundle"
(i.e Synchronous style) + worst- case delay (bundling signal)
"bundling signal"
local worst case delay
req
X
Y
Ack
Sender
Receiver
Single Rail Data

Figure 9: Single Rail bundled data encoding

5.3 Router Micro-Architecture

Asynchronous and synchronous routers have a common principle of operation and structure. However, within the router, the nature of communication between the various components changes the design flow significantly that plays a major role in the performance and cost.

Asynchronous routers:

1. since they are a clockless design, they require handshaking protocols to transfer data.
2. require a different set of buffers and arbiters, as well as safe pipeline designs to avoid glitches and other errors
3. since they use power only during data transfer, they would theoretically have lower energy requirements
4. as data transitions do not occur exactly at the same instant marked by the edge of the clock, they have much lower interferences caused by electromagnetic emissions (this is especially useful in the metasurface context)
5. because the stages do not have to adhere to the clock edges, it might result in lower latency

Figure 7 shows the typical architecture of an asynchronous router. The router consists of Input Channels (ICs), crossbar inputs (CBINs), and crossbar outputs (CBOUTs). IC stores packets in an internal queue and passes them to the CBIN. This communication happens using a handshake protocol with req/ack signals. CBIN then performs the routing computation(RC) and accordingly sends a request to the required CBOUT. The CBOUT has an asynchronous arbiter, which performs switch arbitration by deciding which request to respond to. When data comes from the granted CBIN, a muxer is driven to pass the data through the CBOUT towards the link in a process that acts as switch traversal.

The steps mentioned confirm that the structural design of the routers is conserved: both require buffer write (BW), route computation (RC), switch arbitration (SA), and switch traversal (ST). This allows for a fair comparison between synchronous and asynchronous designs. It is worth noting that the unavailability of tools for asynchronous routers prevents from making a broad design space exploration without an actual gate-level simulation of the router.

To achieve higher performance with lower power dissipation, the asynchronous router is designed using the bundled data method with transition signalling protocol.

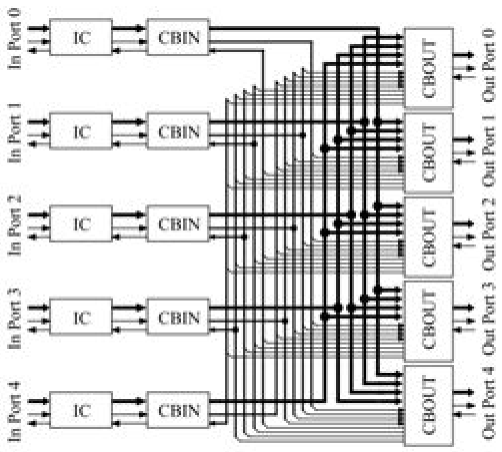


Figure 10: The router architecture from Imai et al [1]

5.3.1 Input Channel (IC)

The IC gets the flit sent to its input port which it then puts into a queue. This queue is just a simple asynchronous FIFO. Asynchronous FIFO is a FIFO design where data is written to the buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain, both these clocks are asynchronous to each other.

In a synchronous FIFO design, the status of the FIFO (full or empty) is determined by a count register. This increments and decrements on the same clock cycle. This is not possible on an asynchronous system, therefore two different asynchronous clock domains are used. An additional bit is used to detect the status of the buffer in an asynchronous buffer. The full and empty is determined by comparing the read and write pointers. The write pointer always points at the next position that is to be written and the read pointer points at the current entry to be read. When the FIFO is empty both the pointers point at zero which is also the reset position. When the FIFO is full, all the bits except the MSB are equal. This FIFO pointer convention contributes to the low access latency.

For simplifying the synchronisation of the pointers across the clock domains we use Gray code. Gray code is the preferred method of encoding the pointers because the Gray code changes by 1 bit each time. This eliminates the problem of synchronizing the read and write pointers.

5.3.2 Crossbar Inputs (CBIN)

Figure 8 shows the crossbar input (CBIN) in detail. When the CBIN receives a head flit from its IC, it goes to the RC Comp. block where it performs the routing computation (RC) and sends a request (arb req) to the arbiter in the CBOUT decided by the algorithm performed. The output is gated by an enable input en1, which is obtained by delaying “head(idata)”, where the head(d) becomes 1 only when signal d carries a head flit.

The XY routing algorithm is simple to implement and the most common routing algorithm that is proposed by Wang Zhang and Ligang Hou used in NoC design [5]. This routing technique comes under distributed deterministic routing algorithm. The advantage of the XY routing is that it never runs into a deadlock or livelock. The XY routing algorithm follows the shortest path and the only one determined path for the packet. This algorithm is suitable for regular and irregular network topologies.

“RC Comp. unit” has another enable input en2 that is activated only by valid head flits. The value of en2 is defined by the value of ireq xor oack. The head flit should be held in the CBIN until the grant “arb_grant” is given from the arbiter in the CBOUT. This is done by gating the request signal ireq using a TR-gate, which is implemented by a D-latch. When the grant is obtained, the TR-gate is opened by the positive edge of the arb_done, and ireq d is forwarded to the corresponding CBOUT through a TR-DEMUX which is a simple demux where the value of arb_grant acts as its selection lines. The functionality of the control circuits in the TR-gate is shown in the red box in Fig. 8. The data-path block is implemented using normal D-latches. When an acknowledgement is sent from the CBOUT, it is forwarded to the IC as oack. TR-MRG is just an exclusive-OR and is used to merge transition signalling inputs that are the value of acknowledgement from the CBOUTs. When oack of the head flit is asserted, the request (arb_req) to the arbiter is released, but arb grant is kept by a latch in CBOUT. The following data and tail flits are just passed from the IC to the CBOUT because the TR-gate is kept open. When ireq_d of a tail flit is sent to the CBOUT through TR-DEMUX, the TR-gate is closed at ire_d when the tail flit becomes 1, for the preparation of a head flit of next packet [1].

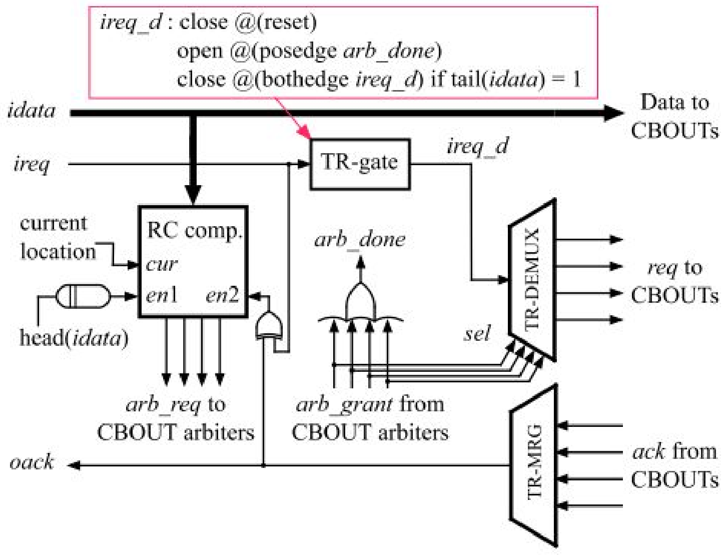


Figure 11: From the literature, Imai et al[1].
The architecture of CBIN in an asynchronous Router

5.3.3 Crossbar Outputs (CBOUT)

CBOUT has a four-input asynchronous arbiter. The grant outputs of the arbiter are latched by a LevelLT which are normal D Latches, and the latched grant outputs arb_grant is sent back to CBINs. The CBOUT has a multiplexer (MUX) in the data-path (its output is mdata) and arb_grant acts as the selection lines for it, i.e., the data from the CBIN that has a grant is selected. This forms a cross-bar.

The arbiter latch is initially open and is closed when a head flit arrives at mdata. This is because arb_req is released after the head flit goes through as mentioned. It is opened again for the next packet when the tail flit in mdata is acknowledged. CBOUT also has one pipeline stage connected to the output port. The ireq signal is obtained by merging the request signals from four CBINs using TR-MRG, and its acknowledgement output is sent to the four CBINs through TR-DEMUX based on arb grant [1].

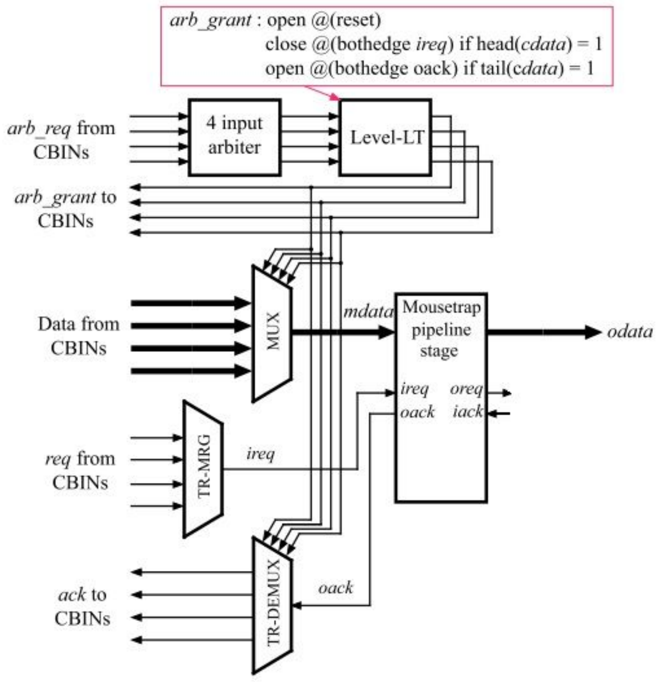


Figure 12: CBOUT Architecture from Imai et al [1]

6. Results

This section lists the output waveforms of all the modules for the asynchronous router and the two different types of handshaking protocols.

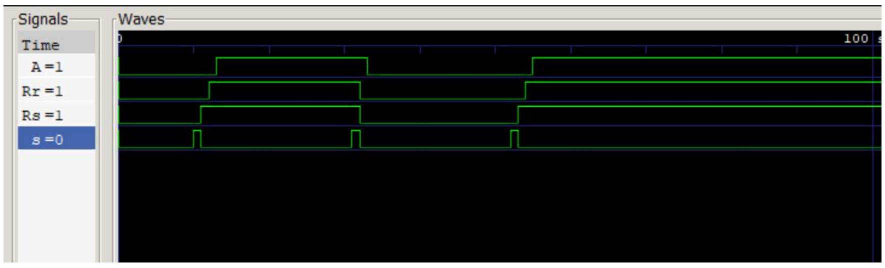


Figure 13: Results for the 2-phase handshaking

Figure 13 shows us the results of a module demonstrating 2 phase handshaking. *Rs* is the request signal from the sender module and *Rr* shows the request signal in the receiver module. Once the stimulus is given to the sender module, a request *Rs* is generated and sent to the receiver module. When the receiver module gets the request signal it generates the acknowledgement signal *A*. We observe that this continues till another stimulus signal is given, at which these signals reset to low and wait for the next stimulus to start another handshake. This shows the non-return-to-zero protocol.

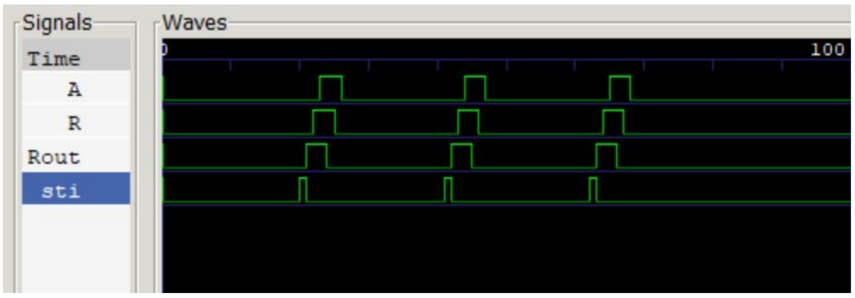


Figure 14: Results for 4-phase handshaking

Figure 14 illustrates the waveform of the code written for a 4-phase handshaking protocol. *Rout* - represents the request signal from the sender. *R* is the signal that is received by the receiver model after some time. *A* - the acknowledgement sent by the receiver module to the sender. The request signal is sent by the sender module after the stimulus. The handshaking is performed between the two modules and then it waits for the next stimulus signal. This shows us the return-to-zero protocol.

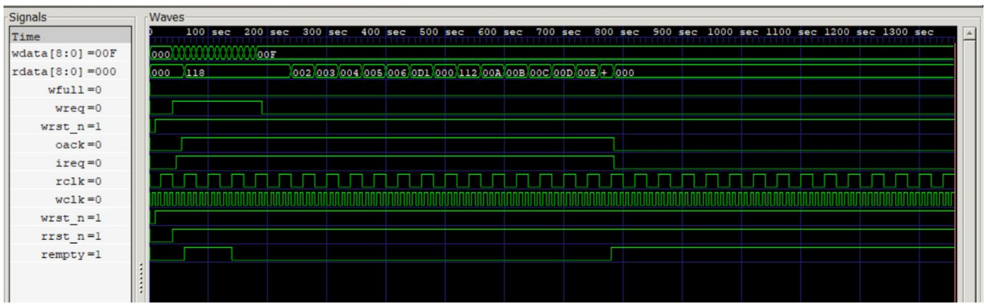


Figure 15: Results for the Input Channel - Asynchronous FIFO

Figure 15 shows the output waveforms of the signals in the Input channel. The asynchronous FIFO has two clocks *rclk* and *wclk* that are auxiliary internal to the FIFO asynchronous to each other. These clocks are for the write and read pointers in the FIFO. *wdata* represents the input data to the FIFO memory. The *rdata* signal represents the data to be read from the FIFO memory when it receives an *ireq* from the CBIN. The *wfull* represents the signal which indicates that the FIFO memory is full and *rempty* represents that it is empty. The *wreq* signal is a request signal for writing the data into memory. While the *wreq* is high and the *wfull* is low, the data written in the memory.

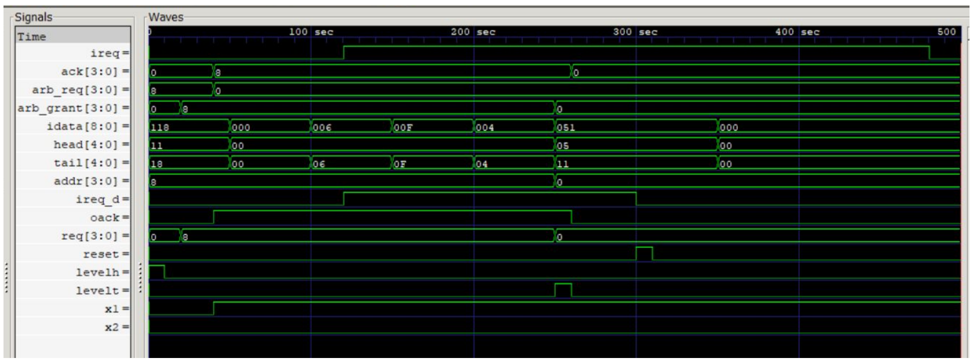


Figure 16: Results for Crossbar input - CBIN of the Asynchronous Router

Initially, the CBIN receives a request from the IC. When the head flit arrives at the RC Comp.unit, the module performs the routing algorithm that decides where the data gets routed. The RC Comp. output is arb_req which is sent to the CBOUT. The *arb_grant* signal from the CBOUT is used as selection lines by the TRDEMUX and also used as *arb_done* in TRGATE. The TRDEMUX sends a *req* signal to the CBOUT. The CBOUT reciprocates with an *ack* signal. This is received by the TRMRG which sends an *oack* signal to the IC. The TRMRG gets req signals from the CBINS and the output of this module is the input to the mousetrap pipeline.

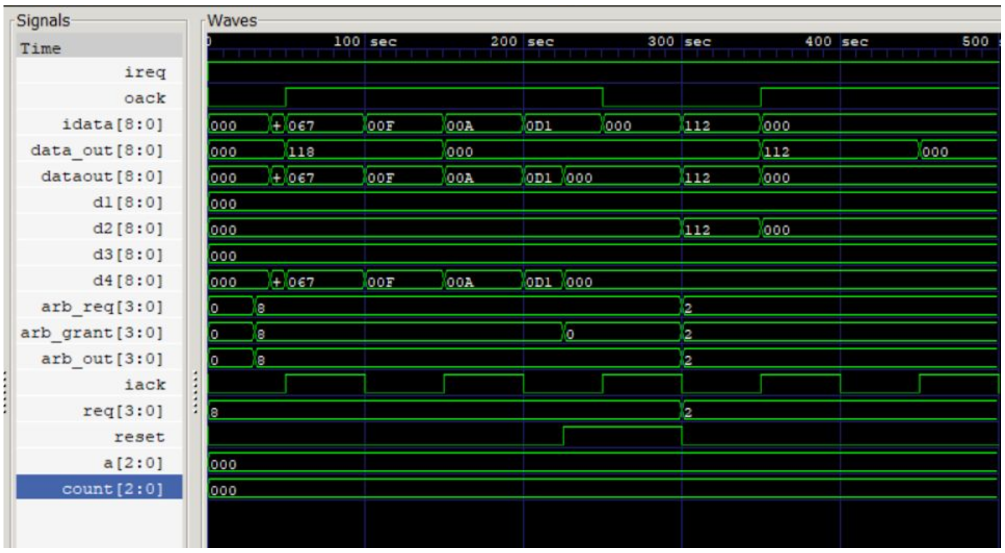


Figure 17: Results for the Crossbar output- CBOUT of the asynchronous Router

The CBOUT receives arb_req from CBIN to its 4 input arbiter. After the arbitration is done the output is gated by a LevelLT module which sends the arb_grant signal to the CBIN. Arb_grant is used as selection lines in the CBOUT module by the MUX and TRDEMUX. The mousetrap pipeline has ireq, iack and mdata as inputs and sends the output as oack, oreq and data_out to the other routers. The MUX gets data inputs (d1,d2,d3,d4) and the arb_grant selects the output of the MUX.

7. Budget

	Time Taken	Cost/Hr	Total Cost
Freelance	4hrs/day x 5months (working days)	€ 30/ person	95 days x 4 x 30 = € 11,400
Startup Company	4 hrs/day x 5months (working days)	€100(Research)	95 x 4 x 100 = € 38,000
	Technology	Minimum Cost	Maximum Cost
Fabricating the Asynchronous Design on a Chip	Advanced Nanometer technology (<90nm)	€ 200k (non-commercial purpose)	€ 1mil (commercial Purposes)

Table 3: Table illustrating the budget for this project

The budget tabulated in Table 3 mentions cost incurred by a freelancer/student working on this particular topic. Time put into this project to build the modules for an asynchronous code is taken into account and the cost for it for 5 months is calculated.

If a company was to take up this project and work on this code, the time for the research is taken into account. The cost incurred by the company is calculated.
For the design to be built on a chip is calculated. Since the asynchronous design is a technology that works under 90nm the cost of producing a chip for non-commercial purposes come to around €200k and if a well-established organisation were to build an asynchronous chip it would cost more than a million euros.

8. Conclusions and future development

We have designed asynchronous Router modules, like the input Channel(IC), Crossbar input(CBIN) and Crossbar output(CBOUT), to work as Standalone units. Also designed programs illustrating the 2-phase and 4-phase handshaking protocols. The routing algorithm used is a simple XY dimension routing algorithm. The packets move in four direction-north, south, east and west. A one stage mousetrap pipeline is used in the CBOUT.

Future Development: The different modules should be connected and made to work as an Asynchronous Router. The area power and latency of this router can be compared with the synchronous router using CAD tools. The asynchronous router that we have constructed is a 4 port module, the scope is to construct multiple port router and have a suitable routing computation algorithm. This will give us a deeper insight into how the router performs compared to synchronous routers.

Bibliography

[1] M. Imai, T. V. Chu, K. Kise and T. Yoneda, "The synchronous vs. asynchronous NoC routers: an apple-to-apple comparison between synchronous and transition signalling asynchronous designs," 2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS), Nara, 2016, pp. 1-8.

[2] M. Krstic et al., “Globally asynchronous, locally synchronous circuits: Overview and Outlook,” IEEE Des. Test, vol. 24, no. 5, pp. 430–441, 2007.

[3] Bhardwaj Kshitij.,” On Multicast in Asynchronous Networks-on-Chip: Techniques, Architectures, and FPGA Implementation.” PhD Diss. Columbia University, 2018

[4] R. Dobkin, R. Ginosar, and I. Cidon. QNoC asynchronous router with dynamic virtual channel allocation. In International Symposium on Networks-on-Chips (NOCS), page 218, 2007

[5] Wang Zhang, Ligang Hou, Jinhui Wang, Shuqin Geng, and Wuchen Wu. Comparison research between xy and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip. In Intelligent Systems, 2009. GCIS’09. WRI Global Congress on, volume 3, pages 329–333. IEEE, 2009.

[6] Benini, Luca, and Giovanni De Micheli. "Networks on chips: A new SoC paradigm." computer 35.1 (2002): 70-78.

[7] Bohnenstiehl, Brent, et al. "KiloCore: A 32-nm 1000-processor computational array." IEEE Journal of Solid-State Circuits 52.4 (2017): 891-902.

[8] Y. Thonnart, E. Beigne, and P. Vivet, “A Pseudo-synchronous implementation flow for WCHB QDI asynchronous circuits,” in Proc. 18th IEEE Int. Symp. ASYNC, 2012, pp. 73–80.

[9] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, “A transition signaling bundled data NoC switch architecture for cost-effective GALS multicore systems,” in Proc. ACM/IEEE DATE, 2013, pp. 332–337.

[10] E. Kasapaki and J. Sparsø, “Argo: A time-elastic time-division-multiplexed noC using asynchronous routers,” in Proc.20th IEEE Int. Symp. ASYNC, 2014, pp. 45–52.

[11] M. N. Horak et al. , “A low-overhead asynchronous interconnection network for GALS chip multiprocessors,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 30, no. 4, pp. 494–507, 2011.

[12] A. Lines, “Asynchronous interconnect for synchronous SoC design,” IEEE Micro, vol. 24, no. 1, pp. 32–41, 2004.

[13] W. Jiang et al., “A lightweight early arbitration method for low latency asynchronous 2D-mesh NoC’s,” in Proc. ACM/IEEE DAC, 2015.

[14] S. M. Nowick and M. Singh, “Asynchronous DesignPart 1: Overview and Recent Advances,” IEEE Design & Test, vol. 32, no. 3, pp. 5–18, 2015.

[15] Jerger, Natalie Enright, and Li-Shiuan Peh. "On-chip networks." Synthesis Lectures on Computer Architecture 4.1 (2009): 1-141.

[16] T. Bjerregaard and J. Sparso, “A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip,” in Proceedings of the DATE ’05, 2005, pp. 1226–1231.

[17] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, “A low-overhead asynchronous interconnection network for GALS chip multiprocessors,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 4, pp. 494–507, 2011.

[18] W. Jiang, D. Bertozzi, G. Miorandi, S. M. Nowick, W. Burleson, and G. Sadowski, “An Asynchronous NoC Router in a 14nm FinFET Library: Comparison to an Industrial Synchronous Counterpart,” in Proceedings of the DATE ’17, 2017, pp. 732–733.

[19] D. Gebhardt, J. You, and K. S. Stevens, “Design of an energy-efficient asynchronous NoC and its optimization tools for heterogeneous SoCs,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 9, pp. 1387–1399, 2011.

[20] L.S. Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In International Symposium on High-Performance Computer Architecture (HPCA), pages 255–266, 2001.

[21] H. Matsutani, M. Koibuchi, H. Amano, and T. Yoshinaga. Prediction router: Yet another low latency on-chip router architecture. In International Conference on High-Performance Computer Architecture (HPCA), pages 367–378, 2009.

[22] S. Park, T. Krishna, C.H. Owen Chen, B.K. Daya, A. Chandrakasan, and L.S. Peh. Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45 nm SOI. In Design Automation Conference (DAC), pages 398–405, 2012.

[23] Sun, Chen, et al. "DSENT-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling." 2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip. IEEE, 2012.

[24] Retrieved from iverilog.icarus.com

[25] Retrieved from https://notepad-plus-plus.org

[26] Retrieved from gtkwave.sourceforge.net/

[27] Retrieved from http://www.csun.edu/edaasic/roosta/Syn_Asyn_Design.pdf

Appendices

1. Code for 2-Phase Handshaking

```
// sender module
module sendermod(reqse, ackse, sti);
output reqse;
input ackse, sti;
reg reqse;
initial
                reqse = 0;
always @ *
                begin
                 if(sti)
                  #1 reqse = !reqse;
                 else
                  reqse = reqse;
                end
endmodule

//receiver module
module receivermod(ackre,reqre);
output ackre;
input reqre;
reg ackre;
initial
                ackre = 0;
always @ *
                begin
                 if(reqre)
                  #1 ackre = 1;
                 else
                   #1 ackre = 0;
                end
endmodule
```

2. Code for 4-Phase Handshaking

```
//sender module
module sender(reqse,ackse,S);
input ackse,S;
output reqse;
reg reqse;
initial
                begin
                 reqse=0;
                end
always @*
                begin
                  if(S)
                  #1 reqse <= 1;
                  else
                   if(ackse)
                   #1 reqse <=0;
    end

endmodule

//receiver module
module receiver(ackre,reqre);
output ackre;
input reqre;
reg ackre;
```

```
initial
                begin
                 ackre=0;
                end
always @*
                begin
                 if(reqre)
                  #1 ackre=1;
                 else if(!reqre)
                  #1 ackre=0;
                end
endmodule
```

3. Code for Input Channel (IC) - Asynchronous FIFO of an Asynchronous Router

```
module async_fifo #(
    parameter DSIZE = 9,
    parameter ASIZE = 4
) (
    input   wreq, wclk, wrst_n,
    input   rreq, rclk, rrst_n,
    input   [DSIZE-1:0] wdata,
    output  [DSIZE-1:0] rdata,
    output  reg wfull,
    output  reg rempty,
                input oack,
                output reg ireq

);
reg [DSIZE-1:0]rdata;

reg     [ASIZE:0]   wq2_rptr, wq1_rptr, rptr;
reg     [ASIZE:0]   rq2_wptr, rq1_wptr, wptr;
wire    rempty_val;
wire    [ASIZE : 0] rptr_nxt;
wire    [ASIZE-1:0] raddr;
reg     [ASIZE:0] rbin;
wire    [ASIZE:0] rbin_nxt;
wire    [ASIZE-1:0] waddr;
reg     [ASIZE:0] wbin;
wire    [ASIZE:0] wbin_nxt;
wire    [ASIZE : 0] wptr_nxt;

// synchronizing rptr to wclk
always @(posedge wclk or negedge wrst_n) begin
    if(!wrst_n)
        {wq2_rptr, wq1_rptr} <= 2'b0;
    else
        {wq2_rptr, wq1_rptr} <= {wq1_rptr, rptr};
end

// synchronizing wptr to rclk
always @(posedge rclk or negedge rrst_n) begin
    if(!rrst_n)
        {rq2_wptr, rq1_wptr} <= 2'b0;
    else
        {rq2_wptr, rq1_wptr} <= {rq1_wptr, wptr};
end

// generating rempty condition
assign  rempty_val = (rptr_nxt == rq2_wptr);

always @(posedge rclk or negedge rrst_n) begin
    if(!rrst_n)
        rempty <= 1'b0;
    else
    rempty <= rempty_val;
end

// generating read address for fifomem
assign rbin_nxt = rbin + (rreq & ~rempty);

always @ (posedge rclk or negedge rrst_n)
  if (!rrst_n)
    rbin <= 0;
  else
    rbin <= rbin_nxt;
assign raddr = rbin[ASIZE-1:0];

// generating rptr to send to wclk domain
// convert from binary to gray
assign rptr_nxt = rbin_nxt ^ (rbin_nxt>>1);

always @ (posedge rclk or negedge rrst_n)
  if (!rrst_n)
    rptr <= 0;
  else
    rptr <= rptr_nxt;

// generating write address for fifomem
assign wbin_nxt = wbin + (wreq & !wfull);

always @ (posedge wclk or negedge wrst_n)
  if(!wrst_n)
    wbin <= 0;
  else
    wbin <= wbin_nxt;

assign waddr = wbin [ASIZE-1:0];

// generating wptr to send to rclk domain
// convert from binary to gray
assign wptr_nxt = (wbin_nxt>>1) ^ wbin_nxt;

always @ (posedge wclk or negedge wrst_n)
  if(!wrst_n)
    wptr <= 0;
  else
    wptr <= wptr_nxt;

// generate wfull condition
wire wfull_val;
assign wfull_val = (wq2_rptr == {~wptr[ASIZE : ASIZE-1],wptr[ASIZE-2 : 0]});

always @ (posedge wclk or negedge wrst_n)
  if (!wrst_n)
    wfull <= 0;
  else
    wfull <= wfull_val;

// fifomem
// Using Verilog memory model
//rdata is the input to CBIN
//the ireq is sent to CBIN when there is a data written in the mem
// oack is the ack from the CBIN to send the data
localparam DEPTH = (1 << (ASIZE));
reg [DSIZE-1 : 0] mem [0: DEPTH -1];

initial begin
        rdata = 'b0;
end

always @(rclk && oack) begin

                if(oack)begin
                        rdata <= mem[raddr] ;
                        end
                        else if(rempty)
                         rdata = 9'b0;
            end

initial
ireq = 0;

always @ (posedge wclk)begin
   if (wreq & !wfull) begin
                mem[waddr] <= wdata;
                        ireq = 1;
                end
            else if (rempty) begin
                         ireq = 0;
                        end
            end

endmodule
```

4. Code for Crossbar input-CBIN for an Asynchronous Router

```
//The head flit should be held in the CBIN until the grant arb grant is given from the arbiter.
//This is done by gating the request signal ireq using a TR-gate, which can be implemented by a D-latch.
module trgate(ireq_d,ireq,reset,arb_grant,idata);

output ireq_d;
input reset,ireq;
input [3:0]arb_grant;
input [8:0]idata;

reg ireq_d;
wire arb_done;
reg [4:0]tail;

assign arb_done = arb_grant[0] || arb_grant[1] || arb_grant[2] || arb_grant[3];

initial
            begin
            tail = idata[8:4];
            end

always @(reset or ireq or arb_grant or idata) begin

            if(reset)
                        ireq_d = 0;
            else if(arb_done && ireq)
                        ireq_d <= ireq;
            else if(tail == 5'b10001)
                        ireq_d = 0;
            else
                        ireq_d = 0;
end

endmodule

//When the grant is obtained, the TR-gate is opened by "@posedge(arb_done)"
// and ireq_d signal is forwarded to the corresponding CBOUT through a TRDEMUX.
//The data-path block is implemented using the normal D-latches.
//When an acknowledgement is sent from the CBOUT, it is forwarded to the IC as oack
//A TR-DEMUX is a DEMUX for the transition signalling protocol,
// and is implemented by both-edge sensitive toggle FFs with enable inputs.

module tr_demux(req,ireq_d,arb_grant);

output reg [3:0]req;
input ireq_d;
input [3:0]arb_grant;
```

```
always @(ireq_d or arb_grant)
		begin
				case(arb_grant)
				4'b0001,4'b0010,4'b0100,4'b1000: req <= arb_grant;
				default: req = 4'b0000;
				endcase
		end

endmodule

module tr_mrg(oack,x1,x2,ack);

output oack;
output reg x1,x2;
input [3:0]ack;
reg oack;

always @(ack)begin
		if(ack)begin
				fork
						x1 = ack[3] ^ ack[2];
						x2 = ack[1] ^ ack[0];
				join
						oack = x1 ^ x2;
		end
		else
				oack = 0;
		end

endmodule

//RC unit is a combinational circuit which does the routing computation when a request comes from the IC to the CBIN. \\

module rcunit1 #(parameter cl = 4'b1010) (arb_req,idata,en1,en2,head,tail,addr,ireq,oack,levelt,levelh);

input [8:0]idata;
output levelt,levelh;
output reg en1;
output wire [4:0]head;
output wire [4:0]tail;
output reg [3:0]addr;
input ireq,oack;
output en2;
output [3:0]arb_req;
reg [3:0]arb_req;
wire [1:0]ax,ay,mx,my;

reg levelh,levelt;

initial begin
		levelh = 0;
		levelt = 0;
		addr = 4'b0;
		arb_req = 4'b0;
		en1 = 0;
		end

 assign head = idata[8:4];
 assign tail = idata[4:0];
 //assign addr = idata[3:0];

 always @(idata)
 begin
		if (head == 5'b10001)begin
				addr = idata[3:0];
				en1 = 1;
				levelh = 1;
				#10 levelh = 0;
```

```
                end
                else if(tail == 5'b10001)begin
                                en1 = 0;
                                addr = 4'b0000;
                                levelt = 1;
                                #10 levelt = 0;
                end
end

assign  mx = cl[3:2];
assign  my = cl[1:0];
assign  ax = addr[3:2];
assign  ay = addr[1:0];

xor(en2,ireq,oack);


 always @(en1 && en2 && addr) begin
                if(addr == 4'b0)
                                arb_req = 4'b0;
                else if((ax != mx)&&levelh)
                                                begin
                                                 if(ax > mx)
                                                                arb_req = 4'b0010; //go east
                                                 else if(ax < mx)
                                                                arb_req = 4'b1000; // go west
                                    end
                else if((ax == mx)&&(ay != my)&& levelh)
                                                begin
                                                 if(ay > my)
                                                                 arb_req = 4'b0001; // go north
                                                 else if(ay < my)
                                                                 arb_req = 4'b0100; // go south


                                                end
                else
                                arb_req = 4'b0;
                end
 always @ (*)begin
                if(addr == 4'b0)
                arb_req = 4'b0;
 end

endmodule
```

5. Code for Crossbar output - CBOUT of the Asynchronous Router

```
//trmrg module is an input signal merging module
module trmrg(ireq,req);

output ireq;
input [3:0]req;

reg ireq;
initial
ireq = 0;

always @ (req)
                begin
                                case(req)
                                4'b0001,4'b0010,4'b0100,4'b1000: ireq <= 1;
                                default: ireq <= 0;
                                endcase
                end

endmodule
```

```
//data mux module
// the mux module is a switch module for the input data into the CBOUT from the CBIN after giving the arb_grant to CBIN
module mux(dataout,d1,d2,d3,d4,arb_grant,idata);

input [3:0]arb_grant;
output reg [8:0]d1,d2,d3,d4;
output [8:0]dataout;
input [8:0]idata;
reg [8:0]dataout;

always @(*)begin
        case(arb_grant)
        4'b0001: begin
                        d1 = idata;
                        {d2,d3,d4} = 9'b0;
                        end
        4'b0010: begin
                            d2 = idata;
                            {d1,d3,d4} = 9'b0;
                       end
        4'b0100: begin
                            d3 = idata;
                            {d1,d2,d4} = 9'b0;
                       end
        4'b1000: begin
                            d4 = idata;
                            {d1,d2,d3} = 9'b0;
                       end
        default: {d1,d2,d3,d4} = 9'b0;
        endcase
end

always @(*)
        begin
                case(arb_grant)
                 4'b0001: dataout = d1;
                 4'b0010: dataout = d2;
                 4'b0100: dataout = d3;
                 4'b1000: dataout = d4;
                 default: dataout = 9'b0;
                endcase
        end
endmodule


//demux module
module trdemux(ack,oack,arb_grant);

output [3:0]ack;
input oack;
input [3:0]arb_grant;

reg [3:0]ack;


always @ (oack)
        begin
                case(arb_grant)
                4'b0001:ack <= arb_grant;
                4'b0010:ack <= arb_grant;
                4'b0100:ack <= arb_grant;
                4'b1000:ack <= arb_grant;
                4'b0000:ack <= arb_grant;
                default:ack <= 4'b0;
                endcase
        end

endmodule
```

```
//4 input arbiter with a round robin algorithm, which is connected to the level Lt.this arbiter gives out arb_out and not
//arb_grant. the level LT is the latch that gives out  arb_grant to the CBIN based on other signal inputs.
module arbiter(arb_out,a,count,arb_req);

input [3:0]arb_req;
output reg [2:0]a;
output reg [2:0]count;
output [3:0]arb_out;
reg [3:0]arb_out;

initial begin
		arb_out = 4'b0;
		count = 3'b0;
		a = 3'b0;
end

always @(arb_req) begin
a = arb_req[3] + arb_req[2] + arb_req[1] + arb_req[0];
end

always @ (a)begin
		if(a == 3'b001)begin

			count <= 3'b0;
		end
		else begin

			count <= a;
		end
		end

reg [3:0]s1,s2,s3,s4;

always @((a && arb_req)||(count))begin

 if(a == 3'b001)begin
		 arb_out <= arb_req;
		 //#5 $display($time,"arb_out = %b,arb_req = %b,count = %b\n",arb_out,arb_req,count);
		 a = 3'b0;
 end


 else if(a > 3'b001)begin
		s1= 4'b0001 & arb_req;
		s2= 4'b0010 & arb_req;
		s3= 4'b0100 & arb_req;
		s4= 4'b1000 & arb_req;

		while (count > 3'b0)
		begin
			if( s1 == 4'b0001)begin
				 arb_out = 4'b001;
				count <= count - 3'b001;

				s1 = 4'b0;
				//		#5 $display($time,"arb_out = %b,arb_req = %b,count =
%b\n",arb_out,arb_req,count);
			end

			else if( s2 == 4'b0010) begin
				arb_out = 4'b0010;
				count <= count - 3'b001;
				s2 = 4'b0;
				//#5 $display ($time,"arb_out = %b,arb_req = %b,count = %b\n",arb_out,arb_req,count);
			end

			else if( s3 == 4'b0100)begin
				arb_out = 4'b0100;
				count <= count - 3'b001;
				s3 = 4'b0;
```

```
                    //#5 $display ($time,"arb_out = %b,arb_req = %b,count = %b\n",arb_out,arb_req,count);
                end

                else if( s4 == 4'b1000) begin
                    arb_out = 4'b1000;
                    count <= count - 3'b001;
                    s4 = 4'b0;
                    //#5 $display ($time,"arb_out = %b,arb_req = %b,count = %b\n",arb_out,arb_req,count);
                end

                else begin
                    arb_out = 4'b0000;
                    //#5 $display ($time,"arb_out = %b,arb_req = %b,count = %b\n",arb_out,arb_req,count);
                end

        end

end
end
endmodule

//level lt module is where the signal of oack and ireq are checked before giving arbiter grant to the CBIN
// level lt gets its input from the arbiter after the round robin algorithm.

module level (arb_grant,arb_out,oack, ireq,idata,reset);

output [3:0]arb_grant;
input [8:0]idata;
input [3:0]arb_out;
input oack, ireq;
reg levelt, levelh;
output reg reset;

reg [4:0]head, tail;

//assign head = idata[8:4];
//assign tail = idata[4:0];

initial begin
        levelt = 0;
        levelh = 0;
        reset = 0;
end

always @ (levelt)begin
 #5 levelh= 0;
end

always @ (*) begin
        if( levelt && !levelh )begin
                #20 reset = 1;
                end
        else if( levelh)
          reset <= 0;
end

always @ (idata)begin
 head = idata[8:4];
 tail = idata[4:0];
        if(head == 5'b10001)
                levelh<= 1;
        else if(tail == 5'b10001)
                levelt <= 1;
        else if (head == 5'b0 && tail == 5'b0)begin
        levelh= 0;
        levelt = 0;
        end
end

reg [3:0]arb_grant;
```

```
initial
 arb_grant = 4'b0;

always @(reset or arb_out or idata) begin
		if(reset)begin
				arb_grant = 4'b0;
				levelt = 0;

				end
		else if( levelh || !levelt )
						arb_grant <= arb_out;
		else if(levelt)
				 arb_grant <= 4'b0;
		else
				arb_grant = 4'b0;
end

endmodule

//this is a 1 stage mousetrap pipeline.
module mousetrap(data_out,oack,en, ireq, iack,reset, datain);

output [8:0]data_out;
output oack;
output en;
input ireq, iack;
input [8:0] datain;
input reset;

reg clk = 0;
always #50 clk = ~clk;

reg [8:0]data_out;
reg oack;
reg en;

initial begin
		oack = 0;
		data_out = 9'b0;
		en = 0;
end

always @ (iack or oack)begin
		en = ~( iack ^ oack);
end

always @(posedge clk)  //(reset or ireq or datain or en)
		begin
				if(reset)
				fork
						data_out <= 9'b0;
						oack = 0;
						en = 0;
						//$display ($time,"data_out = %b,en = %b,oack = %b",data_out,en,oack);
				join

				else if(ireq)begin
						if(en)
								fork
										data_out <=datain;
										oack = 1;
										#5 en = 0;
								join
				   else
								data_out = 9'b0;
		   end
		 // data_out = 9'b0;
		end

endmodule;
```