Asynchronous Router Design for Scalable GALS Networks on Chip

Ben Meakin
meakin@cs.utah.edu

Abstract

The aggressive scaling of multi-core designs in embedded and desktop domains has lead to the incorporation of scalable networks on chip (NoC) as the physical communication medium. There is a significant demand for low-latency and high-throughput communication via these networks. Heterogeneous designs in embedded applications have additional communication demands including lower power consumed by the network and bridging of different clock domains. Globally asynchronous- locally synchronous (GALS) networks provide an effective solution to these additional demands. This paper describes the design and implementation of an asynchronous on-chip router module; the fundamental building block of true GALS networks.

1 Introduction

Current computing trends are moving towards exploiting more application and thread level parallelism. Designing chips with multiple computing cores is critical to increasing parallelism. As designs incorporate more and more cores, communication becomes a major bottleneck. Efficient on-chip networks are needed to continue this trend. These types of chips do not just exist in the high performance computing domain, but in embedded devices as well. Therefore, power and area are critical design parameters in addition to performance.

These types of parallel chip architectures also may employ heterogeneous cores; particularly in the embedded domain. These chips may have several general purpose cores, as well as various application specific cores such as DSP, GPU, cryptographic, and various other hardware accelerators. In such heterogeneous designs, cores are very likely to be in different clock domains. Therefore, communication between these cores is most efficiently implemented as asynchronous logic. As a system this is referred to as globally asynchronous- locally synchronous (GALS).

GALS networks must meet the same demands of low-latency and high-throughput at a low hardware cost as synchronous networks. Design techniques that best meet these requirements have been extensively researched in previous work. Some of these state-of-the-art design methods have been incorporated in a synchronous wormhole router designed previously. The objective of this work is to desynchronize this design by replacing the clock with formal communication protocols to control the pipeline flow. Section 2 presents the router organization and implementation using the 130nm University of Washington library. The communication protocols that control the pipeline are described in section 3. The latency, throughput, power, and area metrics of the asynchronous design are presented and compared to the synchronous design in section 4. Much of the work presented here is based on work previously done by the author and is under review for publication to Memocode 2009 in [1].

2 Router Organization

The main hardware structure in an on-chip network is the router. The router performs packet switching, arbitration, and buffering. For flow control, modern on-chip routers use a wormhole routing scheme to minimize the required buffer space at each hop. In a wormhole scheme, packets are divided

into flits where only the header flit contains routing information. These flits traverse the network in a pipelined fashion. The flits of a single packet may be strung out across several network hops. Since only the header contains routing information, the body flits must follow immediately behind the header.

The organization of this router is a 5 by 5 switch with buffers at the input and output ports, permitting two flits to reside in the router along any given input/output path. The original design had two virtual channels per physical, but this added another layer of arbitration and increased the complexity of the pipeline control. Fair round-robin arbitration and deadlock free dimension order routing are utilized. Firgure 1 shows a block diagram of the router. Note that this diagram only shows logic for one input and one output physical channel. The actual design has 5 input and 5 output channels.

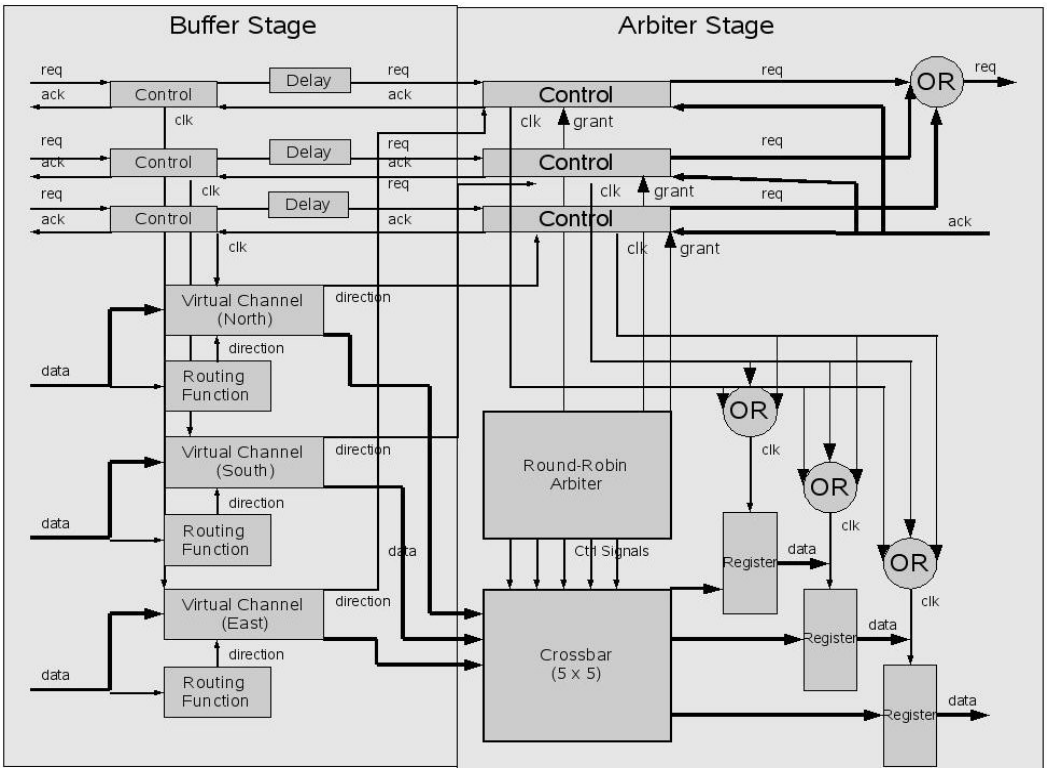

Fig. 1

The data flow is as follows: a flit enters the data-path and is buffered at the input port. If the flit is a header, it is used to look up the direction that the packet must go in a ROM contained in the routing function module. This direction is then stored in the virtual channel module so that subsequent body flits can be routed correctly. Without this subsequent flits would be unroutable since only the header contains the destination address. This direction, is sent as a request to the arbitration module. The arbiter then determines if this packet can have mutually exclusive access to the output port. This decision is based on a round-robin token. If the request is granted, then a signal is sent to the pipeline controllers to allow the flit to make forward progress. The arbiter does not release the mutex until the tail flit is received. This a a requirement for correct operation. However, it does mean that another packet waiting to use the same output port may have to wait for a while. This is a performance/cost tradeoff. Larger flits could contain routing information resulting in lower average case latency, but it would increase buffer size and result in poor network utilization. When the arbiter grants permissions it also sets the appropriate control signals to the crossbar so that the flit is routed to the intended output channel. The flit is then latched at the output port until it can traverse the link to the next router hop.

3 Asynchronous Pipeline

The router pipeline is two stages and is controlled by a 4-cycle 'return to zero' request/acknowledge handshake protocol. The controllers for each pipeline stage were specified in extended burst mode [2] and synthesized using the 3D asynchronous synthesis tool. The state machines used as input to the synthesis tool are given in figures 2 and 3.

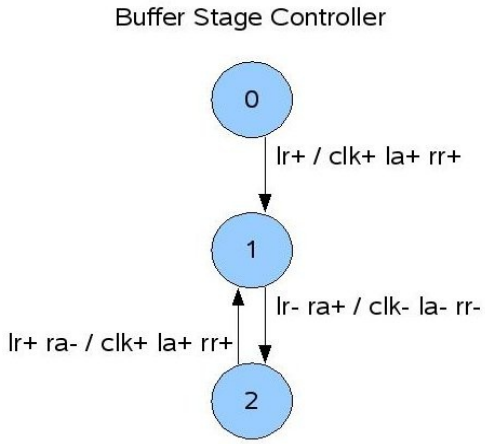

Fig. 2

There are 5 controllers for each pipeline stage, one for each input/output channel. Packets move through the pipeline independent of activity on other independent channels. Therefore, there are 5 concurrent router pipelines. The arbiter ensures that packet transitions from one pipeline to another (ie North input to South output) are mutually exclusive. Two packets cannot both transition to the same output channel at the same time; one must wait for the other. This decision is based on the value of a ring counter and priority logic contained in the arbiter module.

The buffer stage controller is a simple handshake protocol which controls the clock to the virtual channel modules. Initially, a request from the previous network router must occur. This results in a positive clock edge which latches the flit on the appropriate virtual channel. After this initial event, acknowledgements must be received from the arbitration stage to continue to move flits through the pipeline.

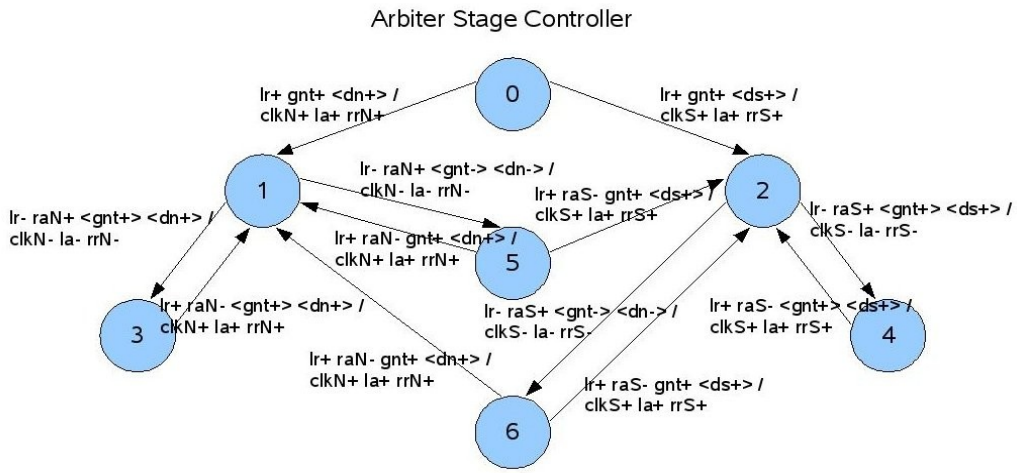

Fig. 3

The asynchronous finite state machine given in figure 3 is considerably more complicated. This is because this controller must manage the transition of a packet from one input channel to any of the five output channels. This requires the packet directions and a grant signal from the arbiter, indicating if the input channel has mutually exclusive access to the desired output channel. The state machine given in figure 3 includes only the north and south output channels, but it will be shown how this diagram is expanded to all five output channels.

The extended burst-mode specification used by 3D permits the use of conditional, or level-sensitive, signals in addition to signal transitions. Conditional signals are used to indicate the packet direction (dn, ds, de, etc.) as well as the grant signal. In figure 3, the states on the perimeter (0->1->3 or 0->2->4) are similar to state machine for the buffer stage in figure 2 and they control the flow of flits through the pipeline once the direction has been determined and a mutex has been obtained. The states in the center are decision states where the appropriate actions are taken for a flits intended direction. The extension needed to accommodate all output channels is to have five state similar to states 5 and 6, one for each direction. All of those states would be able to transition to five different states similar to states 1 and 2, one for each direction. This would create 5 states where a decision is made about what direction a packet is going (ie 5 and 6) and 5 states for doing the handshake for each possible output channel (ie 1 and 2).

On the positive edge of the clock from the arbitration stage controller the flit from the input buffer is latched at the appropriate output port. To avoid a race condition and to ensure that the correct value is at the output port at the time the clock arrives, the request signal from the buffer stage to the arbiter stage is delayed. This is indicated by the delay module in the block diagram which is simply a string of inverters.

4 Results

At the time of this writing much of the testing to obtain accurate measurments for power and performance are a work in progress. However, synthesis, placement and routing of the design have been completed. The UofU digital library was used in the 500nm process technology. The complete chip layout is given in figure 4. Floorplanning was difficult, and due to poor pin placement of the sub modules only 65% of the chip area could be utilized. It measures approximatly 1545 by 2080 microns. Design rule checking completed with zero errors. The log is given in the appendix. Due to an unknown error related to possible missing cell views in sub-modules of the arbiter, LVS could not be run on the whole circuit. However, LVS was performed and was successful for all modules except the arbiter. All of the verilog source code, synthesis results, and AFSM specifications are given in the appendix and give approximate numbers for the delay of each module (pre place and route). DEF, LEF, and post place and route netlists are also given.

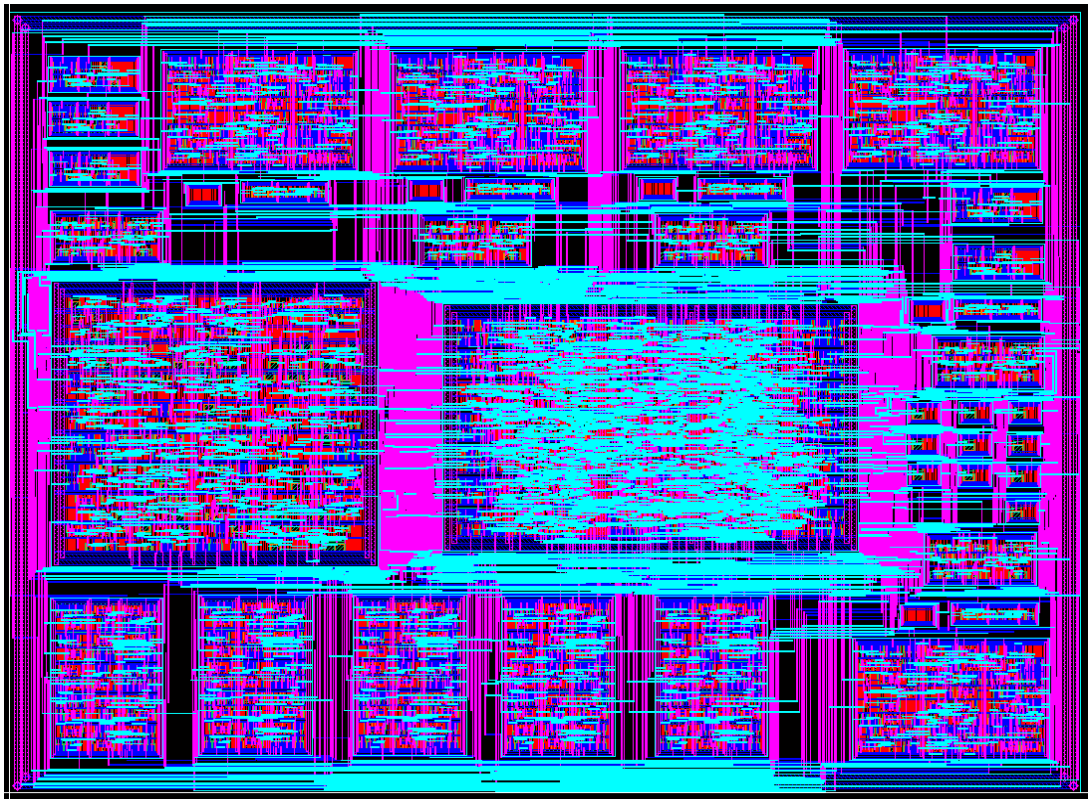

Fig. 4

5 Conclusion

The design of an asynchronous on-chip router has been presented. The design presented enables globally asynchronous on-chip communication. At this time it has not been shown that the resulting implementation achieves lower latency and power than its synchronous counterpart. Aside from the obvious benefits of an asynchronous router, the benefits of this design have not yet been proven. However, this work has presented a useful case study of asynchronous circuit design in an area that could greatly benefit from an asynchronous implementation. It seems obvious that embedded heterogeneous chips need a communication mechanism that supports modularity and different clock domains. The actual benefits of this design will be made more clear in future work.

6 References

[1] Ben Meakin and Ganesh Gopalakrishnan, “Hardware Design, Synthesis, and Verification of a Multi-core Communication API,” 2009.

[2] Kenneth Yun and David Dill, “Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementations),” IEEE Transactions on CAD of IC and Systems '98.