Elastic Flow in an Application Specific Network-on-Chip

Daniel Gebhardt[1,2]

School of Computing
University of Utah
Salt Lake City, U.S.A.

Kenneth S. Stevens[3]

Electrical and Computer Engineering
University of Utah
Salt Lake City, U.S.A.

Abstract

A Network-on-Chip (NoC) is increasingly needed to interconnect the large number and variety of Intellectual Property (IP) cells that make up a System-on-Chip (SoC). The network must be able to communicate between cells in different clock domains, and do so with minimal space, power, and latency overhead. In this paper, we describe an asynchronous NoC using an elastic-flow protocol, and methods of automatically generating a topology and router placement. We use the communication profile of the SoC design to drive the binary-tree topology creation and the physical placement of routers, and a force-directed approach to determine router locations. The nature of elastic-flow removes the need for large router buffers, and thus we gain a significant power and space advantage compared to traditional NoCs. Additionally, our network is deadlock-free, and paths have bounded worst-case communication latencies.

Keywords: VLSI, GALS, Network-on-chip, Asynchronous

1 Introduction

As process scaling continues, more complex designs can be fit on a System-on-Chip (SoC). The design difficulty of SoCs is increasing with multiple clock domains and intellectual property (IP) components integrated in one design. A Network-on-Chip (NoC) is a solution for the increasing communication complexity, which has made traditional point-to-point and bus interconnects less feasible [2,5].

Globally asynchronous, locally synchronous (GALS) communication is an answer to the problem of maintaining a low-skew global clock signal across a large SoC.

[1] This work was supported in part by Semiconductor Research Corporation under Contract No. 2006-TJ-1424.

[2] Email: gebhardt@cs.utah.edu

[3] Email: kstevens@ece.utah.edu

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

In GALS, communication is asynchronous between isolated synchronous domains. The overarching NoC communication structure is a natural design space in which to include GALS principles, with recent implementations showing promise [3,13].

The design space of a NoC is very large, and includes topology choice (mesh, torus, star, etc.), circuit switched or packet switched, and other parameters (link widths, frequency, etc.). Because the traffic patterns of most SoCs can be known, a custom generated network topology and physical placement of components yields better performance and power than a regular-pattern network [10]. A NoC's buffers and links can consume near 75% of the total NoC power [15], thus there is significant benefit to optimizing buffer size, link length and bandwidth of a NoC design.

In this work, we present an asynchronous NoC using an elastic channel protocol, which offers a number of advantages over traditional NoC elements. We then present a workflow for automatic network topology generation and router placement. We use heuristic algorithms, but an optimal mixed integer linear programming method can be used [17], at the sacrifice of scalability. We use the communication characterization of a design to drive the topology generation, router placement and link bandwidth matching.

We use an example design to illustrate our methods throughout this paper. The example design consists of six soft-IP blocks and specified communication paths. We assume a characterization of these paths can be derived from the expected traffic patterns.

Our paper is organized as follows. Section 2 describes our novel network components and our elastic-channel protocol. Section 3 specifies the analysis needed to synthesize an efficient network. Sections 4 and 5 explain our algorithms for determining routing topology and network component placement. Finally, Section 6 presents our conclusion.

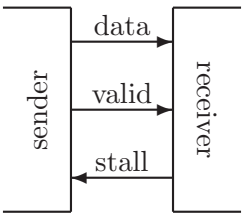

Fig. 1. Sender and receiver connected by Elastic Channel

2 Network Components

Our network fabric is based on latency insensitive protocols. *Latency Insensitive* or *elastic* system design are an adaption of asynchronous handshake protocols to the clocked domain. These protocols allow extra pipeline delays to be inserted into a datapath without changing the results of the computation. Elastic systems are just like clocked systems in that they consist of a collection of modules and channels. However, the elastic communication channels have two control wires, *valid* and *stall*, analogous to request and acknowledge, that implement a handshake between the sender and receiver. Figure 1 shows a clocked sender and receiver module communicating across an elastic channel. valid propagates in the same direction as

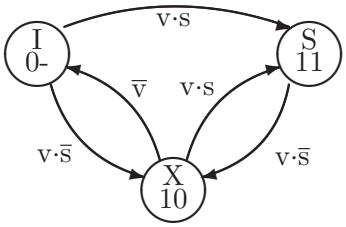

Fig. 2. Elastic Channel Protocol, state: {v s}

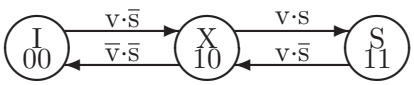

Fig. 3. Interleaved Elastic Channel Protocol, state: {v s}

the data, and stall travels in the opposite direction.

Design performance, complexity, and applications are determined by the channel protocol employed. We have chosen a protocol similar to the Synchronous Elastic Flow (SELF) channel protocol [4] shown in Figure 2. This synchronous protocol was targeted to add elasticity to the design with zero overhead to the clock frequency. We have completed the initial development of a complementary new *phase* channel protocol (pSELF) shown in Figure 3. Both protocols are idle in the 'I' state, actively transfer data each cycle in the 'X' state, are stalled in the 'S' state. Each of these states are labeled with the values of the valid and stall signals. This new protocol has similar performance and design benefits of the original protocol but better interoperability since it is compatible with both asynchronous handshake protocols and cycle based clocked designs. We have also targeted our phase based design to network topologies where logic repipelining is not needed. Therefore the two latches in the flop-flops of a traditional clocked design can be broken into two independent latches in the network topology separated by long communication links.

The network fabric in this work uses design targets that differ significantly from other Network on Chip (NoC) designs, including: a) A non-redundant network topology. b) No multi-segment packets – each transmission is a single data word containing all necessary routing information. c) Simple high throughput network routers and buffers. d) Both clocked and fully asynchronous realizations of the NoC fabric. This produces a fabric that has ultra low latency, high throughput, and a static worst case latency for all transmissions (assuming sufficient buffering exists in the network interfaces). Bidirectional network links consist of two elastic channels transmitting data in opposite directions.

Latency insensitive network fabrics can be implemented using two components: a phase elastic half buffer (pEHB) shown in Figure 4, and a binary routing buffer, or $\top$ element. The router consists of three pEHB's, three data muxes, and three data merge components. The pEHB's provide buffering on the three outgoing channels. The muxes steer data on incoming channels to one of the other two outgoing channels. The merge elements allow each of the outgoing channels to be shared the other two incoming arms in the $\top$. Fair arbitration protocols are used between the incoming channels for both the clocked and asynchronous designs. See [18] for more details on the design of the network hardware and a characterization of the network fabric performance.

The most important properties of the network fabric are bandwidth, latency, and power. These are all directly proportional to the topology and placement chosen for the fabric. Latency in a clocked network will be equivalent in cycles to the number of buffers on a path between the sender and receiver - be they routers or elastic half

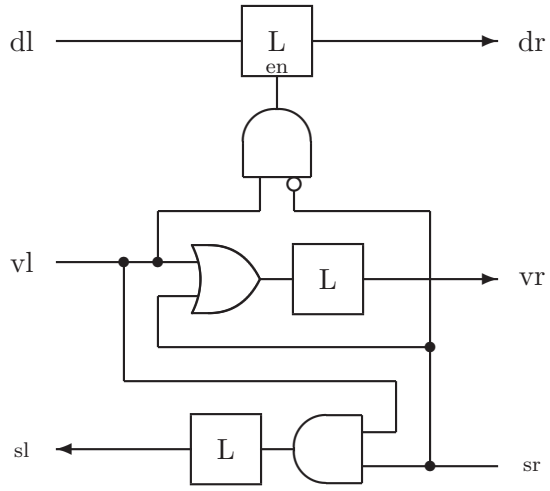

Fig. 4. A phase Elastic Half Buffer (pEHB)

buffers. Since buffers must all be evenly spaced on the network fabric in a clocked design the total wire distance will determine the number of hops in the network - not number of routing decisions required. Bandwidth is independent of the number of cycles when passing through elastic buffers, but becomes restricted when two high traffic paths share the same communication link through a router. Thus bandwidth is primarily improved by limiting the number of routers high bandwidth data passes through, and latency is improved by reducing the length of the wires. Power is a combination of latency and bandwidth: it is proportional to the quantity of traffic multiplied by the distance traveled. The network fabric synthesis algorithms in this work minimize the number of shared links, or routers, that high bandwidth data traverses, and reduce the total distance between low latency networks. There is of course some competition between latency, power, and bandwidth since high bandwidth links will also increase the latency of signals sharing the link.

3 System Characterization

System characterization is the process of discovering the communication properties of paths between blocks of an SoC design. Unlike chip multiprocessors which run arbitrary applications, SoCs typically have a well-known traffic model of communication between blocks. Such SoCs include small designs (MP3 decoder) to large designs (Philips Nexperia [6]). Other research has developed methods to perform this characterization [9,11], and it has been used in related automatic NoC synthesis [10,17].

Our algorithms require the system characterization to generate a number of weighted-edge graphs, called communication trace graphs (CTG). A CTG consists of vertices which are the IP blocks in the design, edges showing communication between IP blocks, and edge weights representing the relative criticality of the paths between IP blocks. This criticality is determined by the bandwidth and latency requirements of a path, and is explained further below. A CTG for our example is shown in Figure 5.

Our methodology uses two CTGs, one for topology generation and one for router

placement. Note that in our example design, we reference Figure 5 as an example for both. The CTG weights used for topology generation are a function mostly of the bandwidth requirements between blocks. A path requiring high bandwidth should receive a large weight to minimize network congestion. The topology generation process tries to minimize the number of routers on highly weighted paths to reduce the number of paths contending for high-traffic routers. Additionally, high bandwidth paths should have few routers to reduce power [14]. The latency requirement for a path may also play a role in topology generation, but to a lesser degree. However, a path requiring low latency could also receive a high weight. Although the physical distance of a path will dictate latency more than its number of routers, the more routers a path has, the greater the probablility of congestion at one of those routers which creates excess delay.

The CTG used in router placement might be weighted differently than that used in topology generation. The actual weighting factors are determined by the target application and design goals, most notably, performance or power consumption. Performance is best optimized by factoring in only latency requirements. A high-bandwidth link with a lax latency requirement should not be heavily weighted, as the available bandwidth of a link is not affected by its total length. However, when optimizing power, high bandwidth links should be a first-priority [14], because long, heavily trafficked paths use significant power.

The system characterization process must also build a table of required bandwidths for each path, and derive an ideal packet width in bits. This information is used for specifying link widths and repeater locations.

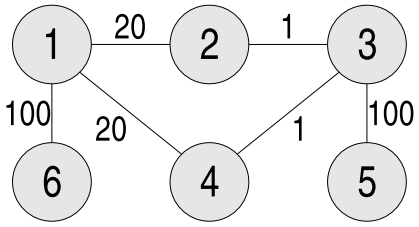

Fig. 5. CTG of example design.

Floorplanning

Floorplanning determines the location on the chip of each IP block based on block geometry and some minimization function (such as wire length). Our methodology at this point does not integrate a custom floorplanner, but we can use an existing floorplanner modified to incorporate our characterized communication requirements, as provided in a CTG. Other work using this method is in [10], which uses the Parquet floorplanner [1]. Custom SoC/NoC floorplanning and router placement algorithms are used in [16].

4 Routing Topology Generation

The topology generation algorithm uses a CTG of a design, as specified by the system characterization process, to determine which IP blocks connect to which

routers of the binary-tree. The process is iterative, where each iteration connects two unconnected groups with a new router, forming a new group. Each group consists of one or more IP block(s) and their previously connected routers. The end result is a binary tree, with the maximum number of routers in a path on the order of $\mathcal{O}(log_2\ n)$, where n is the number of IP blocks. Others have proposed a similar method [8] that may yield shorter paths from some CTG graphs, but its worst case path distance is $\mathcal{O}(n)$, and thus can require more address bits transmitted with each packet.

The following items define datastructures and terminology used in our algorithms.

- Group: a single IP block, or two groups joined by a router. We use this recursive concept to hierarchically explain our method.
- Topology Graph $T(V_t, E_t)$, where each $v_i \in V_t$ is a group and each $e_k = \{v_i, v_j\} \in E_t$ is a physical network link between groups. Note that as this graph is being built, it may not be a connected graph.
- CTG Graph $C(V_c, E_c)$, where each $v_i \in V_c$ is initially an IP block and each $e_k = \{v_i, v_j\} \in E_c$ is a communication trace between v_i and v_j. As the algorithm progresses, two vertices combine to form a new vertex in C and a corresponding group in T containing the same IP blocks.
- Map of edge weights, $W[e \rightarrow w]$. For every $e_k \in E_c$, w_k is the path criticality weight.

Algorithm 1: Connects IP blocks and routers to generate a routing graph.
TopoGen()
  Initialize T with vertices of V_c and no edges.
  while V_c contains two or more vertices (groups)
    Unmark all $v_i \in V_c$ to indicate *ungrouped*.
    while two or more $v_i \in V_c$ marked *grouped*
      Find $(v_i, v_j) \in V_c$ connected with the highest weighted edge, e_{max}, that are not marked. An edge of weight 0 is implied between a vertex pair with no incident edge.
      Create a new router v_{tnew} in T. Connect v_{tnew} to vertices in V_t corresponding to v_i and v_j in V_c.
      Group v_i and v_j of e_{max} to form new v_c.
      Mark v_c to *grouped*.
      Combine edges incident to both v_c and any one $v_i \in V_c$ by making a single edge from v_c to v_i with weight equal to the sum of the separate edge weights.
  Remove unneeded "root" router from T, connecting its children groups directly.
  T contains the generated topology.
return

Algorithm 1 generates the topology for our example as shown in Figure 6, using the CTG in Figure 5. The first iteration groups blocks 1 and 6 through router A, blocks 3 and 5 through router B, and blocks 2 and 4 through router C. The next

iteration groups router A and C through router D because the total wieght of CTG paths between A and C is 40 compared to 2 between B and C. Finally, D and B are paired. This final pairing does not require a new router, but the figure shows a "virtual root" node to keep the familiar binary-tree form.

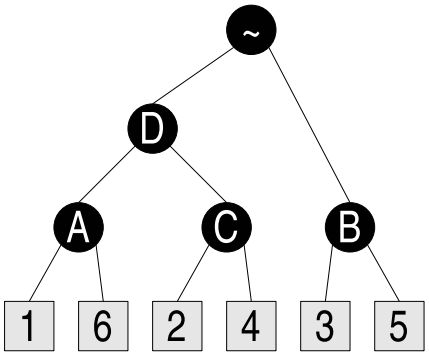

Fig. 6. Generated routing topology.

5 Network Component Placement

This section describes our methodology for placing routers, link repeaters, and specifying other network parameters.

Router Placement

The physical placement of network routers is an important step to minimize latency on critical communication paths. This is especially true when the network is asynchronous because physical distance between endpoints (routers or cells) directly determines latency. In a synchronous network this is also true, but the number of routers through which a message must travel has a greater effect.

We use a force-directed method [12,7] modified for our router placement problem to determine router locations. The underlying theory has been in literature for decades, but the details of this implementation and its application are novel. The input to this algorithm is a CTG, network topology, and IP block layout. The CTG is generated during system characterization, the network topology is the binary routing tree built in Section 4, and the block layout is generated by the floorplanner. The key concept is that each path in the CTG may assert a force on a router along the path to attempt to move it such that the physical path length is shortened. A force is only asserted on a router by a particular path if that router is considered *critical*. A router that is not critical for a given path and axis simply means that moving it along that axis does not yield a shorter path. This is explained in depth in Definition 5.1.

The algorithm starts by placing all routers in their initial positions. A reasonable initial placement is as follows. For each pair of IP blocks, place their shared router at the midpoint between their centers. This procedure is repeated for the next level of routers, using the midpoint between the previously placed routers, and continues until all routers are placed. The initial router placement of our example is shown

in Figure 7. The routers are shown as black circles, IP blocks as grey rectagles, and cell centerpoints as Xs.

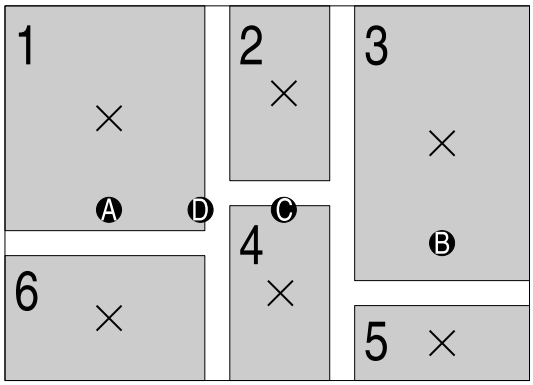

Fig. 7. Initial router placement.

We next iterate the force-directed algorithm by repeated calls to the iteration step described in Algorithm 2, which moves the routers until a stopping condition is met. A simple stopping condition is when no router moves more than Δd. Δd, and the constant c in Algorithm 2, are dependent on the desired convergence time and accuracy as set by the algorithm implementation.

Algorithm 2: The procedure steps repeated during router placement.
IterationStep()
  foreach edge $e_k = (v_i, v_j)$ in the CTG, $C(V_c, E_c)$
    foreach router R along the path v_i to v_j in the topologic graph $T(V_t, E_t)$
      AssignForce(R,e_k)
    foreach router
      Sum all force vectors on router, yielding $\boldsymbol{F_R}$. Move router in the direction of $\boldsymbol{F_R}$, and distance proportional to the length of $\boldsymbol{F_R}$ and a constant c.
    End placement if no router moves more than Δd.
return

Definition 5.1 A *critical router* for an edge in E_c on either the x or y axis is a router on the path from v_{ci} to v_{cj} in the topology graph that has the following property: its incident edges lead to two vertices on the path with distance vectors pointing in the same direction. In other words, given a router R with coordinates (x_R, y_R), edges to vertices $v_{t1} = (x_{v1}, y_{v1})$ and $v_{t2} = (x_{v2}, y_{v2})$ on the path, and axis $\boldsymbol{a}$ (either x or y): R is a critical router if d_1 and d_2 do not have opposite signs, where: $d_1 = \boldsymbol{a}_R - \boldsymbol{a}_{v1}$ and $d_2 = \boldsymbol{a}_R - \boldsymbol{a}_{v2}$.

The force equation in Algorithm 3 determines the length of a router's force vector and is proportional to two factors: the path CTG weight and the ratio of its shortest path distance on axis a to the total distance on both axes. Highly weighted paths obviously will get proportionally higher forces. We include the distance factor as a ratio in order to reduce forces on highly weighted paths that would not greatly benefit from a decreased length along a, or to increase the force on a lesser weighted path if it will greatly benefit.

Algorithm 3: Calculates a force on a router based on communication path needs.
AssignForce(R,e_c)

foreach axis $\boldsymbol{a}$ in $\{\boldsymbol{x}, \boldsymbol{y}\}$

if R is a *critical router*

Assign a force vector $\boldsymbol{F_R}$ to R on axis $\boldsymbol{a}$ as follows:

Find the least path distance, $d_{\boldsymbol{a}}$, on $\boldsymbol{a}$ between:

R to incident v_{t1} added to v_{t1} to v_{c1} and

R to incident v_{t2} added to v_{t2} to v_{c2}

where v_{c1} and v_{c2} are endpoints of e_c and v_{t1} and v_{t2} in V_t are incident to R along paths to v_{c1} and v_{c2}.

$\boldsymbol{F_R}length = \dfrac{d_{\boldsymbol{a}}}{d_{\boldsymbol{a}} + d_{\boldsymbol{a'}}} * w_{ij}$

where w_{ij} is the weight of the CTG edge $e_{ij} = (v_c1, v_c1)$, and $d_{\boldsymbol{a'}}$ is the distance between v_{c1} and v_{c2} along the opposite axis a'.

$\boldsymbol{F_R}dir$ is positive if the location of v_{t1} or v_{t2} is greater on $\boldsymbol{a}$ than R. Otherwise, $\boldsymbol{F_R}dir$ is negative.

return

We have a choice of what specific location on a block to use to represent a connection to that block during router placement calculations. Possible locations include the geometric center of the block, the specified location of the network adapter in hard IP blocks, or the nearest point on a block to some target point. This target point can be the block's parent router, or a path destination block. For soft IP or fully custom blocks, the nearest border point should be used because that is where the block's network adapter should ideally be placed. The center of a block can also be used effectively when the area of the block is small compared to the total floorplan area.

We now use the example in Figure 8 to explain the router force-assignment process. For this example, we assume a soft-IP design and thus use the block edge rather than the block center coordinates for force calculations (d_1 and d_2). Consider the communication path of block 1 to 2. From the topology in Figure 6, we see that a packet must go through routers A, D, and C. This example shows the algorithm in the state of assigning force in the x-direction to node A with respect to the aforementioned path. A is a critical router because both its link-distance vectors, d_1 and d_2, point in the same direction. The endpoint of d_1 is the right edge of block 1 because it is the closest x-coordinate of block 1 to the path destination (block 2). The endpoint of d_2 is the left edge of block 2 because it is block 2's closest x-coordinate to the path destination. Force F is calculated based on the shorter of these two distances, d_1. We use the shortest distance because the path length does not decrease by moving A more than d_1. Note, if the centers of blocks were used, no force would be applied because A lies between the centers of 1 and 2 and hence would not be critical.

The router placement may locate routers within the border of a block. This is very likely on a dense floorplan with minimum space between blocks. If hard-IP block are used, these overlapping routers must be moved outside the block in order to form a valid placement. Finding an optimal solution requires calculating the equilibrium of the system for every combination of side for every overlapping router,

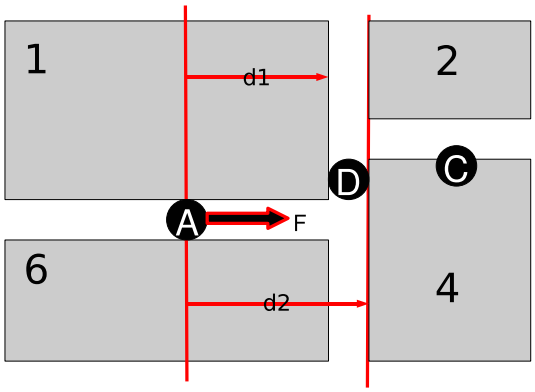

Fig. 8. Force on router A in the x-direction due to path 1-2.

which is computationally infeasible. Thus, we use an approximation method as described in Algorithm 4, containing the following datastructures and terminology.

- A router placement $\boldsymbol{R}$
  $\boldsymbol{R} = \{r_1, r_2, ..., r_n\}$, $r_k = (x, y)_k$
  $\boldsymbol{R}$ is sorted in descending order by block's area of the block that r_i overlaps. A large block has the potential to move the overlapping router more than a small block, and thus we want to consider the effect that re-placement of this router will have on the other routers overlapping a block.
- Block coodinates (lower right corner) and dimentions of overlapping routers for each r_i:
  $\boldsymbol{B} = \{b_1, b_2, ..., b_n\}$, $b_k = (x, y, d_x, d_y)_k$
  Note, that a b_i is null if r_i is not within a block.

Algorithm 4: Moves routers that overlap IP blocks.
De-Overlap()
  foreach r_i
    if b_i is not NULL
      foreach *Side* of b_i
        Make new router placement $rnew_i$ by changing one coordinate of r_i such that $rnew_i$ falls on the *Side* of b_i.
        Run the *IterationStep* algorithm with only R_i moveable, allowing only movement parallel to *Side*.
      Set r_i to the $rnew_i$. This results in the minimum force placed on R_i among each *Side* evaluation.
  while *IterationStep* has not converged
    Run *IterationStep*. A router is not allowed to enter a block's area, and instead stop at the edge.
return

If the design uses soft-IP or custom blocks, we need to place the network adapters for each block. We simply use the point on the block nearest to its attached router.

The final router placement for our example is shown in Figure 9. The dashed arcs show the logical connectivity of the network topology for convenience. We see from the CTG and topology in Figures 5 and 6 that blocks 1 and 3 communicate

to block 4 through router C. Router C has forces excerted on it due to the path from node 3 to 4 (P_{34}) and path P_{14}. Since the weight on P_{14} is so much larger than P_{34}, it has much more effect over the placement of router C (and D). Also notice that even though blocks 2 and 4 are connected by a single router, they do not communicate and thus there is no force on router C bringing it closer to the midpoint of its connected blocks, unlike routers A and B and their connected blocks.

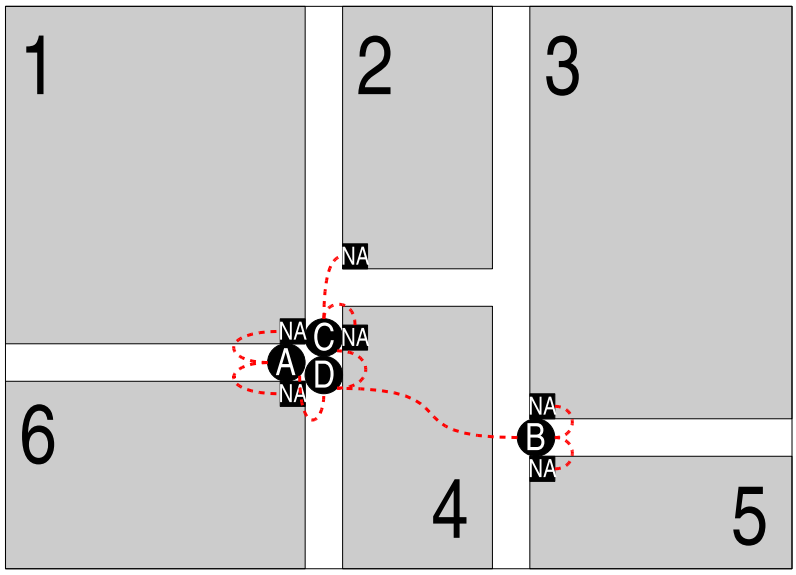

Fig. 9. Final router placement.

We have two choices for routing network wires: channel space using lower metal layers, or entire reserved upper layers. In future work we will investigate the tradeoffs involved with each of these methods, including wire routing difficulty, repeater placement, and scalability issues.

Guaranteed Link Bandwidth

A well designed network will have the bandwidth capacity to prevent network congestion under normal operating conditions. Our elastic NoC framework allows us to specify the bandwidth capacity of any particular link by varying the physical spacing of the asynchronous pEHBs placed on a link between endpoints. This spacing effectively controls the link's "pipeline" depth. We can also have differing link widths on those paths that require them. Finally, we note that bandwidth into and out of a block can be asymmetric. This would be a common case for memory block transfers, with one address packet in one direction, and many data packets in the other direction.

We find the total capacity requirement for a link by summing the individual available bandwidth needs for each path using that link. The system characterization process determines each path requirement, and the topology graph describes the set of links each path uses.

6 Conclusion

In this paper we have presented a novel asychronous NoC that operates using a new elastic-channel protocol, capable of operating with both clocked and asychronous IP blocks. We explained the benefits of moving the elastic components into the NoC domain, including a reduction in buffer sizes and boundable worst-case path latency and bandwidth. We then described a series of algorithms that automatically generate the topology and router placement based on a characterization of the system's communication requirements.

References

[1] Adya, S. and I. Markov, *Fixed-outline floorplanning: Enabling hierarchical design*, in: *IEEE Trans. on VLSI*, 2003.

[2] Benini, L. and G. D. Micheli, *Networks on chips: A new soc paradigm*, Computer **35** (2002), pp. 70–78.

[3] Bjerregaard, T. and J. Sparsø, *A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip*, in: *Proceedings of DATE*, 2005.

[4] Cortadella, J., M. Kishinevsky and B. Grundmann, *Synthesis of synchronous elastic architectures*, in: *Proc. ACM/IEEE Design Automation Conference*, 2006, pp. 657–662.

[5] Dally, W. J. and B. Towles, *Route packets, not wires: On-chip interconnection networks*, in: *Proc. ACM/IEEE Design Automation Conference*, 2001, pp. 684–689.

[6] Dutta, S., R. Jensen and A. Rieckmann, *Viper: A multiprocessor soc for advanced set-top box and digital tv systems*, IEEE Design and Test **18** (2001), pp. 21–31.

[7] Eisenmann, H. and F. M. Johannes, *Generic global placement and floorplanning*, in: *DAC '98: Proceedings of the 35th annual conference on Design Automation*, 1998, pp. 269–274.

[8] Jeang, Y.-L., W.-H. Huang and W.-F. Fang, *A binary tree architecture for application specific network on chip (ASNOC) design*, in: *Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems*, 2004.

[9] Lahiri, K., A. Raghunathan and S. Dey, *Design space exploration for optimizing on-chip communication architecture*, in: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004.

[10] Murali, S., P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli and L. Raffo, *Design of application-specific networks on chips with floorplan information*, in: *Proc. of the International Conf. on Computer-Aided Design*, 2006.

[11] Murali, S. and G. D. Micheli, *An application-specific design methodology for stbus crossbar generation*, in: *Proceedings of DATE*, 2005, pp. 1176–1181.

[12] Quinn, N. and M. Breuer, *A forced directed component placement procedure for printed circuit boards*, in: *IEEE Trans. CAS*, 6, 1979, pp. 377–388.

[13] Rostislav, D. R., V. Vishnyakov, E. Friedman and R. Ginosar, *An asynchronous router for multiple service levels networks on chip*, in: *Proc. of the International Symposium on Asynchronous Circuits and Systems*, Los Alamitos, CA, USA, 2005, pp. 44–53.

[14] Srinivasan, K. and K. S. Chatha, *A technique for low energy mapping and routing in network-on-chip architectures*, in: *Proc. of the International Symposium on Low Power Electronics and Design*, New York, NY, USA, 2005, pp. 387–392.

[15] Srinivasan, K. and K. S. Chatha, *A low complexity heuristic for design of custom network-on-chip architectures*, in: *Proceedings of DATE*, 2006.

[16] Srinivasan, K., K. S. Chatha and G. Konjevod, *An automated technique for topology and route generation of application specific on-chip interconnection networks*, in: *Proc. of the International Conf. on Computer-Aided Design*, 2005.

[17] Srinivasan, K., K. S. Chatha and G. Konjevod, *Linear-programming-based techniques for synthesis of network-on-chip architectures*, IEEE Trans. Very Large Scale Integr. Syst. **14** (2006), pp. 407–420.

[18] You, J., Y. Xu, H. Han and K. S. Stevens, *Performance Evaluation of Elastic GALS Interfaces and Network Fabric*, in: *Third International Workshop on Formal Methods in Globally Asynchronous Locally Synchronous Design (FMGALS 07)*, Elsevier Electronic Notes in Theoretical Computer Scinece, 2007.