Multi-domain Communication Scheduling For FPGA-based Logic Emulation

Murali Kudlugi
Emulation Systems Group
IKOS Systems Inc.
Waltham, MA
murali@ma.ikos.com

Russell Tessier
Dept. of Electrical and Computer Engg.
University of Massachusetts
Amherst, MA
tessier@ecs.umass.edu

Abstract

Communication scheduling is a technique used by many parallel verification systems to pipeline data signals across shared physical wires. This scheduling approach makes it possible to multiplex processing component pins across numerous logical signals, effectively overcoming pin limitations. While it is generally straightforward to derive a fixed relationship between the verification system clock which controls communication and the numerous phase-related clocks of a design under test, mapping complications arise if a user design contains multiple clocks that operate asynchronously to each other. Specifically, multi-clock domain behavior makes it difficult to ensure that reconvergent multi-FPGA paths that are sourced and sampled by multiple asynchronous design clocks can be evaluated accurately. In this paper, we describe scheduling and synthesis techniques that address the reconvergent fanout problem for designs with multiple asynchronous clock domain signals. It is shown that when our approach is applied to an FPGA-based logic emulator, evaluation fidelity is maintained and increased design evaluation performance can be achieved for large benchmark designs with multiple asynchronous clock domains.

1. Introduction

Recently, parallel verification platforms, which can perform millions of logic evaluations concurrently, have received widespread acceptance. These systems, such as logic emulators [1,2,3] and parallel simulation engines, generally are constructed from specialized components, such as special-purpose logic processors or FPGAs. During design verification, logic is evaluated and results are communicated using a high-speed system clock. For combinational paths, multiple logic evaluations are completed during each user design cycle, necessitating a fixed relationship between the behavior of system and design clocks. This relationship can then be used to schedule the evaluation of logic functions and to communicate the results between processors. Multiple design clocks with known phase relationships can also easily be handled by deriving a base frequency which can be used for logic and communication scheduling.

Early FPGA-based verification systems dedicated user design signals to inter-FPGA routing resources, leading to device pin limitation issues [4]. As a result of limited pin availability, the amount of logic assigned to each FPGA was constrained to a fraction of the available device logic. Modern FPGA-based systems use pin multiplexing to assign multiple emulated signals to a specific physical wire over time [5]. These virtual connections are pipelined at the maximum clocking frequency allowed by the FPGA technology [6]. Although communication scheduling eases pin limitations and allows for full FPGA logic utilization, its synchronous nature can introduce constraints for user designs containing multiple asynchronous clock domains.

For multi-domain circuits mapped to pin-multiplexed logic emulators, inter-FPGA signal transport is complicated by design signals that both transition and are sampled on multiple domain edges. This multi-domain behavior is limiting since pin-multiplexed systems often require logical signals that are assigned to a given physical inter-FPGA wire be driven in the same domain. As a result, the transport of multi-domain signals requires that each signal be logically split into constituent single-domain values before inter-FPGA transport. These single-domain values must then be combined at the destination to support multi-domain behavior. Causality is an issue in such systems since actual routing delays can vary across inter-FPGA domain paths. System scheduling algorithms must ensure that a regenerated multi-domain value is consistent with the pre-transport value created at the source FPGA. In the past [4,6], the multi-domain transport problem has been addressed through special compilation and/or manual steps that isolate individual asynchronous domains in hardware. These approaches directly map communication paths to special system hardware at the expense of performance and mapping flexibility. Not only has this approach been difficult and time-consuming, but often results are unpredictable, leading to verification flaws.

In this paper we present a scheduling approach that provides causally-correct transport of multi-domain signals. The basis of this approach is the formulation of a set of constraints which can be integrated into system-wide scheduling. It will be shown that this approach can be scaled to handle an unlimited number of asynchronous domains and can achieve provable modeling fidelity. After formulating the scheduling problem and describing our general approach, a discussion of the integration of our algorithms with a commercial FPGA-based logic emulation system from Ikos Systems is provided [2]. It is shown that the new constraints and algorithms achieve modeling fidelity and overall system performance improvement versus a previous "hard-wired" approach for two large commercial ASIC designs that contain multiple asynchronous clock domains.

2. Background

The target system for this paper is an Ikos VirtuaLogic emulation system that contains 384 Xilinx XC4062XL FPGAs. This system contains six boards of 64 FPGAs each. All intra-board connections in the system are point-to-point between FPGAs, while inter-board connections are supported via a passive backplane. Inter-FPGA communication in VirtuaLogic systems is based on Virtual Wires technology, an inter-FPGA communication scheduling technique. This approach pipelines multiple logical signals called Virtual wires across single inter-FPGA wires to overcome FPGA pin limi-

tations [6,7]. Logic designs are mapped to multi-FPGA VirtuaLogic systems through a series of compilation steps. These steps include design partitioning into logic blocks small enough to fit within FPGAs, placement of logic blocks onto specific FPGAs, and scheduling of both intra-FPGA logic evaluation and inter-FPGA communication. Both logic evaluation and signal communication are controlled by a high-speed clock called a Virtual Clock which serves as a discrete timebase, providing a reliable mechanism for controlling the order of events at a fine granularity. Since many combinational evaluations and signal transfers may occur in a single design clock cycle, the virtual clock by necessity runs at a much higher frequency than the design clock. For logic emulation systems, inter-FPGA (processor) communication scheduling is based on the virtual clock.

This work builds upon previous scheduling approaches that address multi-domain signal behavior in parallel verification systems. The basic approach of scheduling asynchronous domain data transfer for logic emulation systems has previously been applied to latches[8]. The main focus of this previous work was to address hold-time constraints for latches driven by signals sourced by multiple clock domains.

3. Transporting Multi Domain Values

*Functional Axiom 1***: Causality**
The occurrence times of combinational logic form a partial order based on causality. If part A feeds part B, events on A must have occurred before events on B.

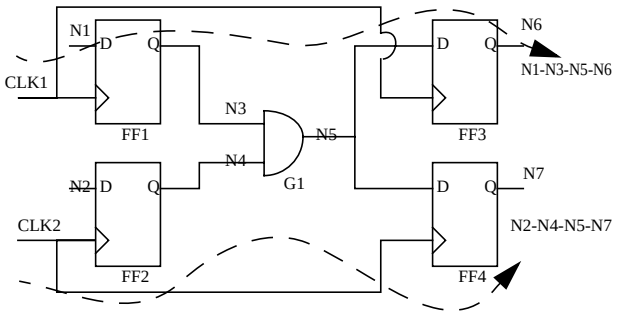

Figure 1: A Multi Domain Circuit Example

Consider the circuitry shown in Figure 1 where two asynchronous clocks CLK1 and CLK2 drive state elements (FF1,FF3) and (FF2, FF4) respectively. This circuit contains two same domain paths, FF1.Q-N3-N5-FF3.D in the domain of CLK1 and FF2.Q-N4-N5-FF4.D in the domain of CLK2. Note that the net N5 transitions and is sampled in both clock domains. It is called a ***MTSD*** *(Multi Transition and Sample Domain)* net.

Consider a situation where the circuit in Figure 1 is partitioned such that the multi domain value N5 that needs to be transported from FPGA1 to FPGA4 as shown in Figure 2. The physical wires that connect FPGAs are grouped into uni-directional *channels* where each physical wire is capable of carrying a single domain value in each Virtual clock cycle. Pin multiplexing makes it possible to reuse physical wires to support numerous logical wires. To complete signal transport, the communication scheduler determines a path from a source FPGA to a destination FPGA and identifies

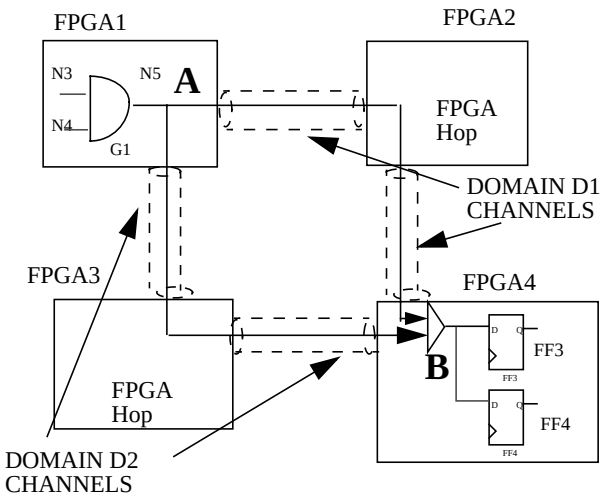

Figure 2: Transporting Multi Domain Values

schedule time slots for the communication to take place. Signal routing may include several intermediate FPGA hops.

A key verification issue involves the transport of multi value signals such as N5 in Figure 1 in a system where inter-FPGA communication needs to be synchronous to a system clock (virtual clock) over a shared physical resource (channel wire). Previous work suggests that we either avoid such a situation by limiting the size of asynchronous-domain logic to one FPGA or dedicate special inter-FPGA wires to transport the values (hard-wiring) [6]. Since hard-wired signals cannot be multiplexed to carry non-MTSD nets, pin limitation problems [5] can result leading to reduced system performance. To avoid this problem, it is desirable to split multi domain values into constituent domain values and to route (schedule) them in respective domains and recover the multi-domain value at the destination FPGA. This solution poses another problem because of unpredictable route timing that is inherent in statically routed systems.

In Figure 2, the circuit in Figure 1 has been partitioned onto a multi-FPGA topology such that the multi-domain value N5 needs to be transported from FPGA1 to FPGA4. As shown in the figure, communication for each asynchronous clock domain takes place over a different set of inter-FPGA channels. In the case of N5, paths

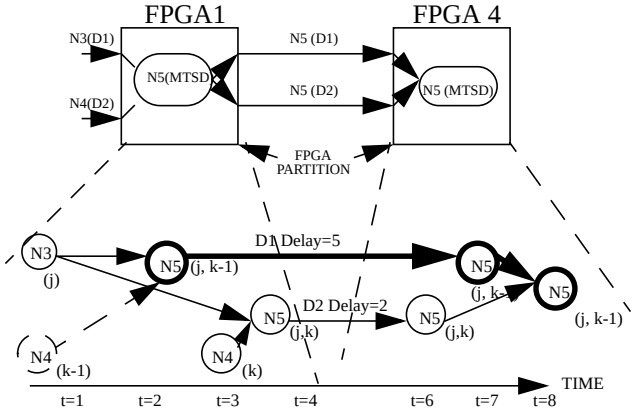

Figure 3: Multi Domain Causality Problem

using both domain1 (D1) and domain2 (D2) channels are needed to transport N5 between FPGA1 and FPGA4. The disjoint nature of multiple routing paths for the same logical signal can lead to causality concerns at the destination FPGA. As a result of unpredictable routing delays due to routing congestion, it is possible for the domain1 (D1) value of N5 to start from the source FPGA sooner than the domain2 (D2) value but still arrive after the D2 value reaches its destination. This can break the causality principle and cause the clobbering of the D2 value. Figure 3 illustrates such a case where a D1 version of signal N5 departs from Point A at t=2 while the D2 version departs at t=3,after a new value of N4 has been created. Due to route congestion, the D1 value reaches destination B after the D2 version. Using combinational rules, when multiple versions of a signal in asynchronous domains are merged at a destination, the most-recently arriving version is used in subsequent calculation. As a result, the late arriving older D1 value at t=7 will be the final value of the signal at B and the newer value that arrived at t=6 will be completely lost. A requirement in transporting multi-domain signals is to ensure that causality of events is guaranteed within each of the constituent domains irrespective of routing delays.

4. Definitions

MTSD Net. A net which transitions (changes value) and is sampled (read) by more than one clock domain. In Figure 1, net N5 is a MTSD net.

MTSD Gate. Any combinational gate whose output is connected to an MTSD net. In Figure 1, gate G1 is a MTSD gate.

MTSD Block. The MTSD logic is partitioned into chunks of size that are small enough to fit into a FPGA. It is at the block boundary all the inter-FPGA communication (routing) takes place.

5. The Approach

Observation 1:

For any signal path relationship Ri(A, B) in a multi domain circuit containing domains A and B, it is sufficient to satisfy Ri(A) and Ri(B) for correct functional verification.

For example, in the circuit showing Figure 1, we only need to satisfy the causality property for the same domain paths FF1.Q-N3-N5-FF3.D and FF2.Q-N4-N5-FF4.D but not for the cross domain paths FF1.Q-N3-N5-FF4.D or FF2.Q-N4-N5-FF3.D.

Inter-FPGA data transport of an MTSD net can be decomposed into the independent transport of signal components from each domain. These components are then causally merged at the destination. We represent these flows by adding FORK/MERGE operator pairs at FPGA boundaries as shown in Figure 4, resulting in a set of same domain signals on FPGA boundaries. From Observation 1, notice that flow and dependence relationships on intra-FPGA paths only need to consider combinationally connected signals from the same domain. Causal merging can be accomplished by dynamically selecting an appropriate single domain signal at a MERGE point.

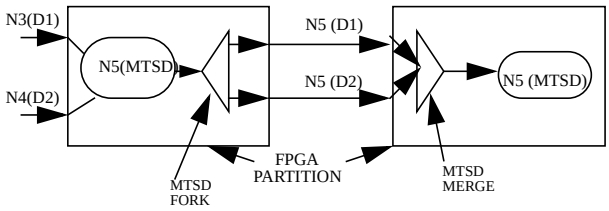

Figure 4: Splitting MTSD paths

To facilitate causal merging, a new type of signal dependency is introduced that takes multi-domain behavior into account. Dependency analysis is performed across multiple domains and is used to compute path depths that are normalized for multi-domain paths. These paths are ordered based on multi-domain depths and are subsequently routed to support causal behavior at destination FPGAs. Our scheduler and synthesizer ensure that the total transport delay in each of the independent domain paths are equal so that the value arriving at the destination is guaranteed to be causally correct.

6. Static Scheduling

We have used a modified TIERS scheduling algorithm to route communication paths between blocks[7]. This is a reverse scheduling algorithm in that it routes paths starting from primary outputs to primary inputs. Note that the techniques explained are also applicable to forward routing. In this section we describe the basic steps involved in static routing. In the next section we describe in detail the specific steps involved in scheduling MTSD paths.

A *route-link* (Pi, Pj) represents a logical connection from block output terminal Pi to block input terminal Pj located on a different FPGA. A route-link often has to cross multiple FPGAs before reaching its destination. We calculate link depths that represent the longest time required to propagate through the network from the source FPGA to the destination FPGA. We create a partial order by sorting route-links by depth to ensure that all the route-links upon which a given route-link depends are scheduled before the route-link itself. The core scheduling algorithm involves the following steps for each route-link from Pi to Pj.

Algorithm 1: Route (route-link)

1. Find the latest time, called *ReadyTime* at which a value must arrive at its destination for further evaluation. For Pj terminating at design primary output k, ReadyTime is Delay(Pj to k).

2. Find the shortest path ‘sp’ from Pi to Pj such that data arrives by ReadyTime(Pj). We use a modified Dijkstra’s algorithm[9]. This takes into account the domain, channel width, channel availability and direction.

3. Reserve wiring resources along the path sp.

4. Compute *DepartureTime*(Pi) at the source Pi:
   DepartureTime(Pi) = ReadyTime(Pj) - PathLength(sp)

5. Update input ReadyTimes at the block,
   for each terminal Pk in Parent(Pi)
   ReadyTime(Pk) = DepartureTime(Pi) - Delay(Pk to Pi)

7. MTSD Scheduling

This section describes how MTSD paths are scheduled so that causality requirements are satisfied at destination FPGAs.

MTSD Dependency and Depth

Combinational dependency analysis is performed on the placed but unrouted design to support the creation of an ordered set of route links for routing. As shown in Figure 4, MTSD paths are given special treatment; they are split into a group of route links that belong to different domains. These links collectively transport domain versions of the MTSD value across FPGAs. One way to guarantee causality of the transported value is to require that all versions of the MTSD signals require the same number of virtual clocks for transport. If the route scheduler can ensure that these route-links are scheduled such that they all take an equal number of virtual clocks to propagate the value, the causally correct value can be easily obtained at the destination.

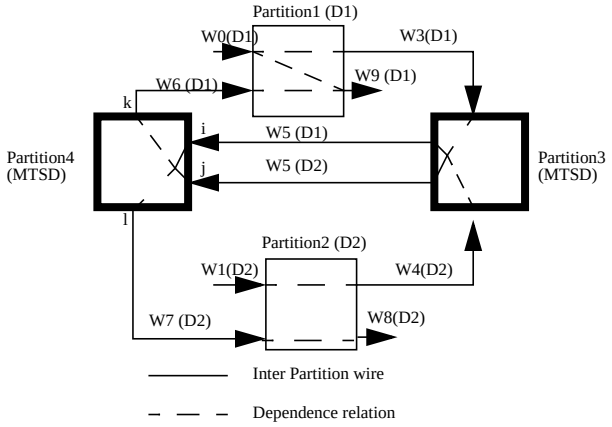

Figure 5: MTSD Dependency and Depth Computation

To support scheduling two types of dependency are computed. *Same-Domain dependency* tracks link dependencies within a single domain and *MtsdDependency* tracks link dependencies across all domains including cross domain paths

Dependency determination is a multi-step process. Initially, link-to-link dependencies are determined.

For each block input terminal i we calculate,

Parent [i] = {Set of block inputs in the same domain that combinationally reach block output i}

and

MtsdParent [i] = {Set of all block inputs that combinationally reach block output i}

Similarly we calculate Child[j] and MtsdChild[j] sets to hold inverse relationships.

Figure 5 shows an example design with four partitions. Partition1 contains a block with strictly domain D1 logic, Partition2 contains a block with strictly domain D2 logic and Partition3 and Partition4 contain MTSD blocks. The partitions are interconnected with a set of wires labeled W0 through W9. Each wire is associated with a specific domain of transition. MTSD nets are split into component domain signals. For example, wires W5(D1) and W5(D2) carry D1 and D2 versions of a MTSD wire W5 from Partition3 to Partition4. For Partition4, which contains an MTSD block, the following domain relationships hold:

- *Child[i] = {k}*
- *MtsdChild[i] = {k, l}*
- *Parent[l] = {j}*
- *MtsdParent[l] = {i, j}*

Following the evaluation of parent and child relationships, the Same-Domain and MTSD depths of each inter-partition wire are determined recursively from the wire dependent sets such that for each wire i:

$$Depth[i] = \begin{cases} 0 (If \ldots Parent[i] = \varnothing) \\ 1 + MaxDepth(Parent[i]) \end{cases}$$

Similarly we compute MtsdDepth by:

$$MtsdDepth[i] = \begin{cases} 0 (If \ldots MtsdParent[i]) = \varnothing \\ 1 + MaxDepth(MtsdParent[i]) \end{cases}$$

Similarly MtsdDepth is computed using MtsdParent relationships.

Note that depths are computed in reverse fashion from path sinks to sources and may cross a number of partition boundaries. Consider depth evaluation for the partitioned example shown in Figure 5. In this example it is known that initially the Same-Domain and Mtsd Depths of wire W8=4 and W9=1 due to downstream paths that are not shown in the figure. From these initial conditions the remaining depths can be determined as follows:

Nets	Depth	MtsdDepth
W9	1	1
W6	2	2
W8	4	4
W7	5	5
W5(D1)	3	6
W5(D2)	6	6
W4	7	7
W3	4	7
W1	8	8
W0	5	8

In the above table, related MTSD links W5(D1) and W5(D2) have different Same-Domain depths but equal MtsdDepths. During routing, same-domain paths are typically scheduled independently to promote optimal scheduling [7]. For MTSD scheduling, a cross-domain restriction is added for MTSD nets. To support causal transport, it is necessary to process all related MTSD route-links together so that they can be routed after their same-domain child

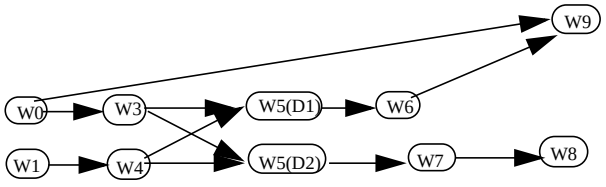

Figure 6: A partial order of Route-links sorted based on MtsdDepth

route-links have been scheduled. As shown in Figure 6, MtsdDepth is used to sort all route links in all domains to produce a partial order that is consistent across all domains.

Routing MTSD Paths

The group of all related route-links of a MTSD net n is referred to as MTSDLinks(n). All route-links in MTSDLinks(n) must be processed at the same time since scheduling one path may affect another. The end result of MTSDLinks routing is a schedule for every route-link in MTSDLinks that is equivalent in length in terms of virtual clock cycles. Note that this distance must be at least the length of the longest FPGA path from a source of net n to the destination of net n. This distance is referred to as the target distance.

In determining routing for each MTSD net, four variables are used:

- *DTrequired* (RequiredDepartureTime), the time at which the signal must depart a source block terminal to satisfy target distance requirements.
- *DTactual*, the time at which a signal departs the source block.
- *ATrequired* (RequiredArrivalTime), the time at which a signal should arrive at a destination block to satisfy the target distance. This is the ready-time computed during the scheduling of dependent route-links, as explained in Algorithm 1.
- *ATactual* (ActualArrivalTime), the time at which a signal is scheduled to arrive at the input of the destination block.

Algorithm 2: MTSDLinks Routing

The basic algorithm is as follows (assuming *reverse* routing):

1. Compute *TargetDistance* for the MTSD route-links. Note that even if source and destination FPGAs for an MTSD net are the same, the path lengths of different domain paths may vary based on channel domain designation.

$$TargetDistance = \underset{Ri \in MTSDLinks(n)}{MAX(Min(Distance(Ri)))}$$

2.For each route-link in MTSDLinks compute DTrequired, the time by which a signal must leave a source block.

DTrequired(Ri) = ATrequired + TargetDistance

3.For each route-link in MTSDLinks find a schedule using Algorithm 1 such that the value arrives at the destination at or before *ATrequired* and has a length less than or equal to the *TargetDistance*.

4. If Step 3 is successful for all MTSD route-links in MTSDLinks, go to step 6.

5. If Step 3 failed because such a feasible schedule can not be found
- delete all the schedules for links already routed and
- increment TargetDistance by 1
- Goto Step 3

6. For each route-link in MTSDLinks(n), reserve the selected wiring resources.

Delay Compensation Synthesis

At the completion of Algorithm 2, it is known that all route-links of a MTSD net have been routed successfully. However, each route-link in MTSDLinks(n) may require a different number of virtual clocks to complete source-destination paths. In order to equalize the delay for all route-links of MTSDLinks(n), it is necessary to add extra virtual clocks to all paths that initially require fewer virtual clocks than the longest route-link path. This can be accomplished by adding delay compensating flip flops under one of two cases:

- Source Compensation: Virtual clock triggered flip flops can be added in the source FPGA at the FPGA boundary such that *SourceCompensation = DTrequired - DTactual*. This is to prevent a signal from being sampled in any domain before it is ready.
- Destination Compensation: Virtual clock triggered flip flops can be added in the destination FPGA at the FPGA input boundary such that *DestCompensation=ATactual- ATrequired*. This approach ensures that the domain data reaches the destination so that causality is preserved.

Delay compensation is implemented by synthesizing virtual clock triggered flip-flops in each single domain path as shown in the Figure 7.a for the example in Figure 3. Figure 7.b shows one possible way to correctly transport multi-domain net N5 to the destination so that causality is preserved. Note that three flip flops are inserted in the D2 path to compensate for the difference in D1 versus D2 routing delay (5 versus 2).

8. Experimental Results

We have implemented the algorithms described in this paper and integrated them into the Ikos VirtualLogic Compiler[3] for the VStation-5M Emulator. We have taken two industrial designs containing asynchronous domains and compiled them using the Virtua-

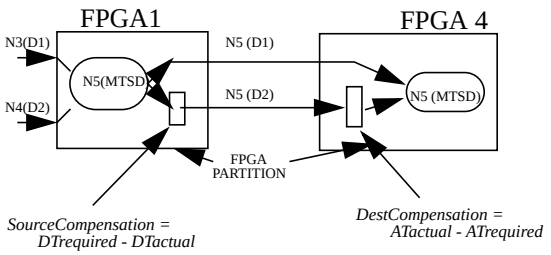

Figure 7.a: Delay Compensation

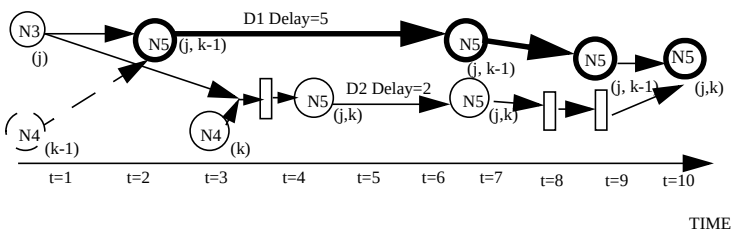

Figure 7.b: Causally correct merge at destination

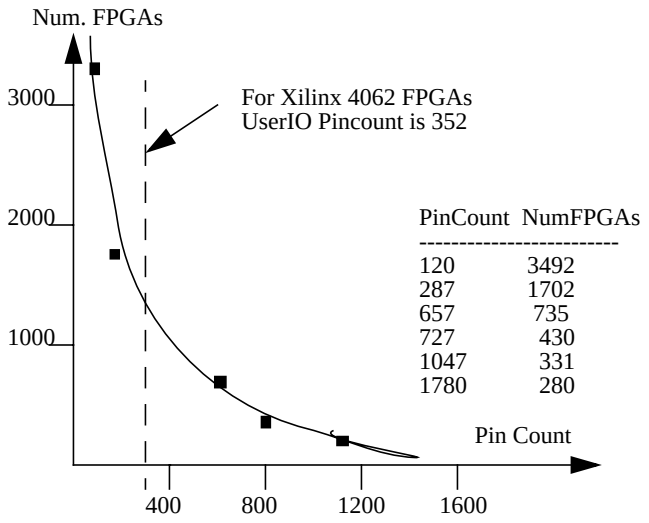

Figure 8: Pin requirements Vs. FPGA requirements

Logic compiler. Design1 has a smaller percentage of MTSD logic when compared to Design2 and has fewer memory modules. Table 1 compares the results of scheduled MTSD routing to hardwire routing. Note that MTSD routed wires and pins are multiplexed to achieve better FPGA pin utilization while hard routed wires require dedicated physical wires and pins. To determine the results for hard routing experiments we ran a pre-routing step which reserved physical pins between source and destination FPGAs for each MTSD wire and removed those pins from consideration during virtual routing of non-MTSD wires. Note that the number of Virtual clocks in the critical path for Design2 is much higher than Design1. This is because experiments for Design2 were dominated by memory transactions. It can be seen that the MTSD routing results in a slightly smaller number of Virtual clocks (hence faster execution) as compared to the hard wired approach. This is because if some physical wires are removed, the remaining wires have to carry a greater load of non-MTSD communication.

Testcase	Design1	Design2
1. Num. Total Modules	543000	57000
2. Num. MTSD Modules	3100	7400
3. Num. Clock Domains	3	2
4. Num. MTSD Paths	173	213
5. Num. MTSD FPGAs	23	24
6. Clock Domains	D1 D2 D3	D1 D2
7. Num. Non MTSD FPGAs	11 43 180	4 7
8. Critical Path (Virtual-Clocks) MTSD Hard Routed	42 47 49	85 131
9. Critical Path (Virtual-Clocks)MTSD VirtualRouted	37 38 46	68 108
10. Est. Max Speed MTSD HardRouted	346 KHz	129 KHz
11. Est. Max Speed MTSD VirtualRouted	369 KHz	157 KHz

Table 1: MTSD Virtual Routing vs. Hard Routing

Maximum emulation clock speeds in rows 10 and 11 are estimated based on a 34 MHz Virtual clock on a VStation-5M Emulator. To further illustrate the usefulness of scheduled MTSD routing, we varied the partition sizes for Design1 and compared the resulting IO pincounts. Figure 8 shows the number of FPGAs needed vs. per-FPGA Pin-counts. Since there is a hard limit on the number of pins on a FPGA, unless time scheduled routing is used, it is necessary to reduce the partition size in order to keep the pincount below this hard limit. This results in a need for substantially more FPGAs to fit the same design for hard routing versus scheduled MTSD routing.

9. Conclusions

In this paper we have addressed the issue of transporting signals in parallel verification systems that are sourced and sampled by multiple asynchronous design clocks. A new scheduling algorithm has been developed that allows these signals to be split into several single-domain versions and transported across inter-FPGA channels dedicated to signals sourced by a single clock. These constituent signals are subsequently merged together at the routing destination to form a causally-correct result. A key aspect of the routing approach is a guarantee that all routing paths for signals created by the split require the same amount of communication time. The approach has been demonstrated on a VirtuaLogic emulation system for two large commercial benchmark designs. Experimental results show that the approach is scalable and provides good modeling fidelity. As a result of this scalability, an improvement in overall system performance was also obtained.

We plan to extend this approach to deal with memories under test and hard-wired cores. The heterogeneous nature of these blocks presents special considerations for scheduling and interfacing.

10. Acknowledgments

We thank Charles Selvidge, Ken Crouch and Matt Dahl for many insightful discussions on scheduling and routing.

11. References

[1] Shekhar Patkar and Pran Kurup, "ASIC Design Flow Scores on First Pass", Integrated Systems Design Magazine, Aug 1997.

[2] IKOS Systems Inc, Virtual Logic Datasheet, http://www.ikos.com/products/vsli/index.html

[3] Quickturn Design Systems, Cobalt Data Sheet, http://www.quickturn.com/products/cobalt.htm

[4] D.E. Van Den Bout, et al. “Any Board: An FPGA Based Reconfigurable System:, IEEE Computer, Sep. 1992, pp 21-30.

[5] J. Babb, R. Tessier et al. “Virtual Wires: Overcoming Pinlimitations in FPGA based logic emulators”. In Proceedings of IEEE Workshop on FPGA based Custom Computing Machines, pages 142-151, Napa, CA, April 1993.

[6] J. Babb, R. Tessier, et al. “Logic Emulation and Virtual Wires”. In IEEE Transactions on CAD, June 1997, Vol 16, No.6, Pages 609-626.

[7] C. Selvidge, et al. “TIERS: Topology Independent Pipelined Routing and Scheduling for VirtualWire Compilation”. In Proceedings of FPGA’95, pages 25-31, Berkeley, CA, Feb 1995.

[8] M. Kudlugi, C. Selvidge, R. Tessier, “Static Scheduling of Multiple Asynchronous Domains For Functional Verification”, To appear at DAC 2001, Las Vegas. Nevada.

[9] Corman et al. Introduction to Algorithms, MIT Press, 1992.