Concurrency Reduction of Untimed Latch Protocols – Theory and Practice

Santosh N. Varanasi Kenneth S. Stevens
Electrical and Computer Engineering
University of Utah

Graham Birtwistle
Department of Computer Science
The University of Sheffield

***Abstract*—A systematic investigation into concurrency reduction of untimed asynchronous 4-phase latch controllers is reported. Starting with a state graph that exhibits maximal concurrency, rules are provided for systematically reducing its states and thereby curtailing its behaviors. The rules predict liveness and occupancy, as well as the regularity and behavior of their pipelines. The rules also reveal the precise extent of the design space and thus provide a secure platform on which to study the implications of concurrency reduction on power, performance and area by implementing and evaluating the complete set of abstracted controllers. This complete characterization enhances the understanding and usage of concurrency and its reduction in handshake protocols. Trade-offs have been observed and reported which will aid designers in trying to find the best protocols for a required specification. Finally, the best synthesized protocols in this class have been identified.**

I. Introduction

This study is motivated by the desire to gain a better understanding of concurrency and synchronization, examine the impact composition has on protocols, how to verify such systems, and to validate some assumptions of concurrency reduction on synthesized hardware implementations. It is an extension of previous work which derived a family of untimed 4-phase latch controller protocols where data is bundled and valid before the request signal rises [2]. In this paper we give this family a simpler structural categorization, use it to characterize some behavioral properties, and present experimental results about the VLSI implementation of the complete family.

The origins for this study lie in work carried out designing, specifying and verifying control signal structures over a range of asynchronous microprocessors culminating in variants (one shown in Fig. 1) on the Manchester AMULET3 [11], [1]. When experimenting with architectural changes to data paths by varying pipeline depths and widths, we noticed that upon the minimization of our models down to the smallest equivalent state graph, pipeline width had no impact. Each variation would minimize down to be equivalent to some single pipeline, but rarely one with the our initial building blocks.

Each asynchronous pipeline stage consists of control and a data path. When the data path is abstracted out, only the controller with its handshake signals remains. Such a controller is represented as a single stage controller, *ST*, in Fig. 2. Such single stage controllers are composed in parallel to create series pipelines SP_d of arbitrary depth d. Series pipelines are often connected in parallel with fork and join components

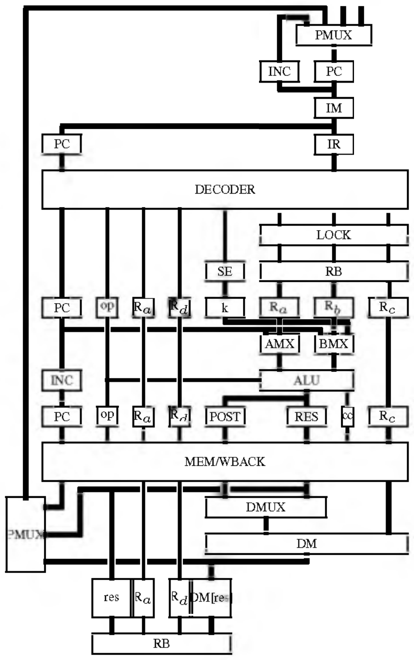


Fig. 1. Abstracted control structure of a simplified AMULET3

to form a parallel pipeline $PP_{w,d}$ of width w and depth d. Such structured pipeline segments can be observed in the microprocessor in Fig. 1.

The results in this paper address the evaluation of data abstracted linear pipelines where each stage contains the same protocol. Discussing pipelines with feedback and addressing the rich set of timed protocols (e.g. burst-mode and relative timed) are not herein addressed.

A. What has been done

The key phases in this study have been:

1. Incubation: Some 40 published latch controller designs (usually specified as STGs) were surveyed and translated to a generalized state graph notation in which internal state variables and latch control signals were hidden whilst retaining the constraints they imposed on how the external pipelining signals interleave. We then ran experiments for each protocol when composed into single pipelines SP_d of depths 1..8 and parallel pipelines $PP_{w,d}$ of widths and depths 1..8 (Fig. 2).

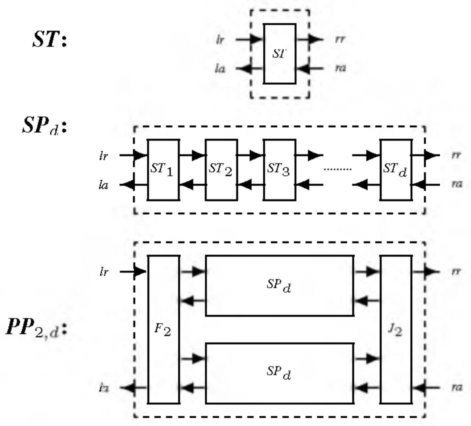


Fig. 2. Single (*ST*), Series (*SP*), and Parallel (*PP*) Pipeline Control Graphs

2. Generalization: All designs and their pipelines were now expressed solely in terms of the same external pipeline control signals and could thus be compared. The most concurrent protocol was identified, called *max*, into which each abstracted published design could be completely embedded. A systematic method of reducing concurrency by cutting away states from *max* was developed to generate the complete family of protocols. Upon examination of the 40 published STG's, we noted that the standard way of restricting behavior was to constrain upstream pipeline signals and downstream pipeline signals separately. Each constraint would engender a characteristic pattern of states being removed, or cut away, from *max*. These we generalized and call the set of all upstream cut patterns $\mathcal{R}$ and the set of all downstream cut patterns $\mathcal{L}$.

3. Search for structure:
The family design space is formed applying all pair combinations of cuts, (one from $\mathcal{L}$ and one from $\mathcal{R}$) on *max*. The orthogonal cut sets $\mathcal{L}$ and $\mathcal{R}$ have lattice structures, and can be used to specify, relate and order all family members [5], [12]. The cut pairs for a specific abstracted design calculate its liveness, behavior, and capacity when pipelined.

4. Implementation: The effect of concurrency reduction was studied by synthesizing the complete set of pipelined controllers and evaluating them for throughput, latency, and power.

II. Modeling 4-Phase Pipeline Stages

We use Milner's Calculus of Communicating Systems (CCS) to model and reason about protocols [18]. CCS has a number of pertinent attributes that make it attractive for this work. It is straightforward to capture signal level behavior. It has a simple formal semantics to support reasoning about designs. Flow graph structure and hierarchy are part of the language, including semantics for how internal hidden behavior, represented as τ, affects externally observable signals. This allows us to formally reason about the signal hiding techniques we used for protocol abstraction. It has reliable public domain tool support, the Concurrency Workbench (CWB) [19].

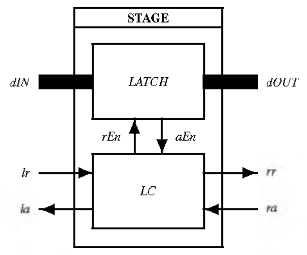


Fig. 3. Bundled Data Controller and Data Latch

Finally, the CWB implements the very powerful modal-μ property checking calculus [21].

Along with these positive points, CCS suffers from the usual state explosion problems. In practice this means that CCS is perhaps best suited to exploring abstract views and control properties of systems, rather than data path logic.

A. The MAX Protocol Abstraction

Fig. 3 shows a bundled data pipeline stage. The latch is responsible for holding the current data value captured from input bus *dIN*. The latch controller (LC) is responsible for synchronizing the input and output channels with the data stream. Since the latch controller protocol works the same for all bus values, *dIN*/*dOUT* can be abstracted by tokens indicating when data can change and become stable.

The model development used in this paper is an abstraction of the 4-phase bundled data protocol using a normally closed (opaque) latch (the approach is equally valid for normally open latches). The following five constraints specify the safety properties of the protocol: **s0:** Liveness: there is a unique quiescent state which can carry out only one action $lr\uparrow$ and which is reachable from all other states. **s1:** A new data value must be stable on *dIN* before the input channel request $lr\uparrow$ is asserted. **s2:** The data is captured in the latch before the input channel acknowledgment $la\uparrow$ occurs. **s3:** The data must be passed through the latch before the output channel request $rr\uparrow$. **s4:** The latch must remain closed, keeping *dOUT* stable, until the output channel acknowledge signal $ra\uparrow$ is asserted.

The latch behavior is specified as follows. Enable request and acknowledgment signals *rEn* and *aEn* control the opening and closing of the latch. As the *dIN* and *dOUT* actions of Fig. 3 are not modeled, markers open and closed are inserted to show the state of the latch. We assume that if a new data value is valid on the input *dIN* when the latch is opened, it will have time to be stored in the latch and propagate to the output *dOUT* before the latch is closed. The separating dot **.** between actions in a CCS definition may be read as *and some time later*.

$$LATCH \;=\; rEn\uparrow.\mathsf{open}.\overline{aEn}\uparrow.rEn\downarrow.\mathsf{closed}.\overline{aEn}\downarrow.LATCH$$

The most concurrent protocol *max* is obtained by delaying handshake signals in the specification only to prevent a safety violation. A CCS specification that results in the most concurrent protocol is shown in Eqn. 1.

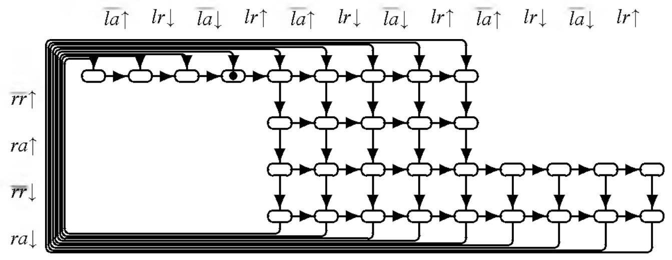

$\overline{la}\uparrow$ $lr\downarrow$ $\overline{la}\downarrow$ $lr\uparrow$ $\overline{la}\uparrow$ $lr\downarrow$ $\overline{la}\downarrow$ $lr\uparrow$ $\overline{la}\uparrow$ $lr\downarrow$ $\overline{la}\downarrow$ $lr\uparrow$
$\overline{rr}\uparrow$
$ra\uparrow$
$\overline{rr}\downarrow$
$ra\downarrow$

Fig. 4. Minimized state graph of *max*, configured as a shape

$$\begin{aligned}
L &= lr\uparrow.gS.\overline{rEn}\uparrow.aEn\uparrow.\overline{rEn}\downarrow.aEn\downarrow.pV.\overline{la}\uparrow.lr\downarrow.\overline{la}\downarrow.L\\
R &= gV.\overline{rr}\uparrow.ra\uparrow.pS.\overline{rr}\downarrow.ra\downarrow.R\\
S &= \overline{gS}.\overline{pS}.S \qquad\qquad V=\overline{pV}.\overline{gV}.V\\
LC &= (L \mid R \mid S \mid V)\setminus\{gS,pS,gV,pV\}\\
LATCH &= rEn\uparrow.\mathsf{open}.\overline{aEn}\uparrow.rEn\downarrow.\mathsf{closed}.\overline{aEn}\downarrow.LATCH\\
max &= (LC \mid LATCH)\setminus\{rEn,aEn\} \qquad (1)
\end{aligned}$$

The trace variables open and closed have done their job and are now omitted. All the handshakes between L and *LATCH* are treated as CCS τ moves (silent internal actions of arbitrary duration). Thus the interplay between the *LATCH* and L is equivalent to:

$$\begin{aligned}
L &= lr\uparrow.gS.\tau.\tau.\tau.\tau.pV.\overline{la}\uparrow.lr\downarrow.\overline{la}\downarrow.L\\
LATCH &= \tau.\tau.\tau.\tau.LATCH
\end{aligned}$$

where the four handshakes in L between gS and pV are all reduced to τ. As each of these τ signals represent nothing more than an arbitrary delay, their net effect is that of a single **.** in CCS. In addition the contribution from the (normally closed) *LATCH* is completely silent. The same argument would apply had we used a normally open (transparent latch). For example, Efthymiou and Garside [7] give normally open/normally closed variations on four different latches. Each pair has the same minimized state graph after hiding.

Process S is a token that ensures property s4 holds and new data is not written into the latch until the previous data has been consumed and the latch has space for the token. Process V is a token that ensures that property s3 holds and data has been stored in the latch before the downstream request is asserted. Process L internally ensures property s2 holds by completing the handshake with the latch before the input channel acknowledgment can assert. Safety property s1, that the data arrives before $lr\uparrow$, is assumed to hold by correct system timing when s3 holds in the upstream controller. Maximal concurrency is obtained by releasing the tokens as early as possible. S is released allowing latch storage upon $ra\uparrow$. V is released as soon as data is stored in the latch.

B. Shape Representation

Fig. 4 displays the minimized state graph of the concurrent protocol *max*. We call this specific state and signal configuration a **shape**. Horizontally the labels show the input channel signals; vertically and wrapped around are the output channel handshake signals. The initial state is marked with the circle ●. The 4×2 block of states on the right of the shape are reached when the input channel gets ahead of the output channel. The leftmost three states are reached when the converse is true and the input channel must catch up with the output channel. The neck on the right is where the device is ensuring that the current value held is not overwritten until it has been passed downstream.

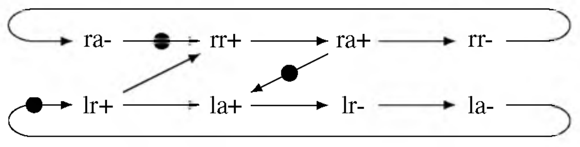


Fig. 5. STG for the abstracted *max* protocol

CCS tracks all possible interleavings. Notice that after an initial $lr\uparrow$ action, *max* permits (i) L to complete the action sequence $\overline{la}\uparrow.lr\downarrow.la\downarrow.lr\uparrow$ before R carries out its $\overline{rr}\uparrow$, and (ii) R to complete the action sequence $\overline{rr}\uparrow.ra\uparrow.\overline{rr}\downarrow.ra\downarrow$ before L carries out its $\overline{la}\uparrow$ action. An equivalent STG of the *max* shape is shown in Fig. 5.

A less cluttered shape for *max* is derived by removing the arcs as shown below. This is the shape notation we prefer to use.

```
o o o ● o o o o o o
        o o o o o
        o o o o o o o o o
        o o o o o o o o o
```

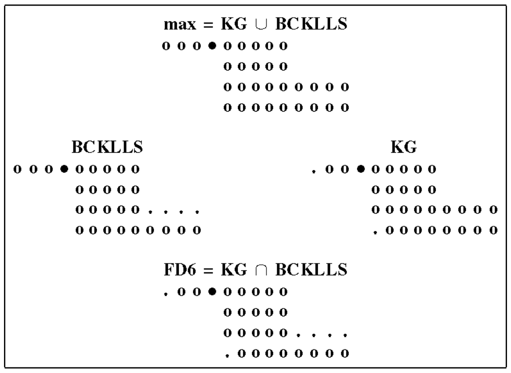
III. Concurrency Reduction on MAX with Cuts

Once each protocol has been modeled as a shape, it is clear that less concurrent abstracted protocols contain fewer states. This can be represented in our compact representation of a shape by replacing the ○ with a **.** if the state is unreachable (cutaway) in a particular protocol.

Table I pictures the shapes of *max* and three published designs: KG due to Kol and Ginosar [13]; BCKLLS due to Blunno *et. al.* [3]; and FD6 due to Furber and Day [10]. Note that *max* is the union of two highly concurrent published protocols, KG and BCKLLS; the third, FD6 is their union. Our experiments confirmed that the *max* shape is the union of all 40 published designs we investigated in the incubation phase.

Less concurrent shapes can be generated by systematically removing (or cutting away) states from *max*, just as the *max* shape can be formed by the union of other shapes. Observation of the shapes of published designs indicated that concurrency reduction removed states from the left and right sides of *max*. Taking our cue from Table I, we decided to partition our concurrency reduction rules into two sets: $\mathcal{L}$ on the left and $\mathcal{R}$ on the right. We call our systematic concurrency reduction rules **left cuts** $\mathcal{L}$ and **right cuts** $\mathcal{R}$ from the *max* shape.

TABLE I
THE RELATIONSHIP OF THE *max* SHAPE AND THREE PUBLISHED DESIGNS



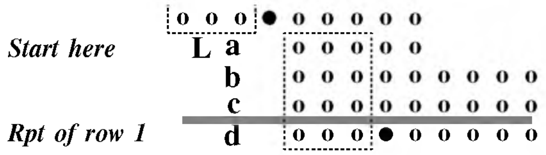


Fig. 8. Left cut $\mathcal{L}$ denotation and range. The top row is duplicated at the bottom of the shape to more easily show the Left cut ordering.

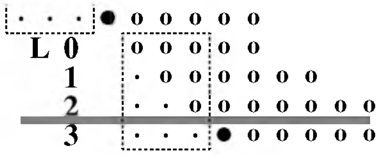


Fig. 9. The shape (above the duplicated line) resulting from cutaway L0123

A. Concurrency Reduction from Right Cuts $\mathcal{R}$

The states removed in a right cut are denoted as **Rabcd** as shown in Fig. 6. Rabcd denotes the removal from *max* of a states from the right end of row 1, b from row 2, c from row 3, and d states from the right end of row 4. The maximal cutaway per row is 4 for rows 1 and 2; and 8 for rows 3 and 4, as shown by the dashed box. If we cut away more states liveness constraints will be violated.

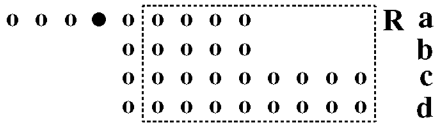


Fig. 6. Right cut $\mathcal{R}$ denotation and range

The result of cut R2152 is depicted in Fig. 7.

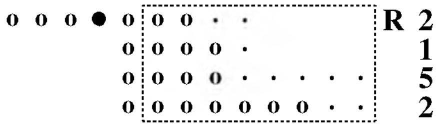


Fig. 7. The shape resulting from cutaway R2152

The family of all $\mathcal{R}$ cuts is generated by the constraints:

$$0 \le a, b \le 4 \qquad 0 \le c, d \le 8$$
$$a \ge b \;\wedge\; b + 4 \ge c \;\wedge\; c \ge d \;\wedge\; d \ge a \tag{2}$$

B. Concurrency Reduction from Left Cuts $\mathcal{L}$

Left cuts, denoted **Labcd**, remove from *max* a states from the left of row 2, b from row 3, c from row 4, and d from the left of row 1. The potential candidates for a left cut now lie in a 3×4 block of states in the dashed boxed of Fig. 8. The liveness rule that requires the initial state to be present and reachable from any state is violated if any more than these states are removed. To emphasize the ordering of left cuts, Fig. 8 temporarily employs a new representation by duplicating the top row after the last row of the shape.

The result of cut L0123 is depicted in Fig. 9.
The family of all $\mathcal{L}$ cuts is generated by the constraints:

$$0 \le a, b, c, d \le 3$$
$$a \le b \;\wedge\; b \le c \;\wedge\; c \le d \tag{3}$$

C. Using cuts to generate the family and to define liveness

The complete family of protocol shapes is generated by applying all pair combinations of left cuts and right cuts to *max*. Not all these shapes will be valid: for example the shape L2222∘R4444 is not live since it deletes all the states from row 2 of the shape. Using an obvious cut indexing, shape Labcd∘Rabcd is live iff Eqn. 4 holds. Thus the liveness of a shape can be calculated directly from its cuts from *max*.

$$La + Rb < 5 \;\wedge\; Lb + Rc < 9 \;\wedge\; Lc + Rd < 9 \;\wedge$$
$$La + Ra < 5 \;\wedge\; Lb + Rb < 5 \;\wedge\; Lc + Rc < 9 \;\wedge\; Ld + Rd < 9 \tag{4}$$

D. The Untimed Family

The right and left cut constraints in Eqn. 2 and 3 express all cuts, including the burst-mode and relative timed protocols. Timed protocols occur when the arrival of an input from the environment can be delayed based on another protocol input or output signal. We restrict the evaluation in this paper to untimed (delay insensitive (DI) and speed independent (SI)) protocols. Constraint rule R1 must additionally hold for all untimed protocols. Delay insensitive protocols must also obey constraint R2.

1) **R1:** input signals *lr* and *ra* must always be accepted
2) **R2:** output signals may be delayed only by inputs

The SI family: When rule R1 is added to the cut and liveness constraints of Eqn. 2, 3 and 4 we obtain the speed independent family of cuts. States must be removed in 1×2 pairs by left cuts and 2×1 pairs by right cuts. For example, referring to Fig. 4, one cannot remove just the rightmost state in any row (e.g. cut R0011) or the input $lr\!\uparrow$ is delayed (resulting in a timed design). The state to left must also be removed (giving cut R0022). R1 is enforced by the following cut equation.

$$\begin{aligned} \mathcal{R}: &\quad a, b, c, d \text{ are even} \\ \mathcal{L}: &\quad a = b \ \wedge \ c = d \end{aligned} \tag{5}$$

The DI family: Adding rules R1 and R2 to our base cut constraints creates the delay insensitive protocols. This is more restrictive than the SI cuts, requiring states to be removed in 2×2 blocks. This removes "output ordering" in the protocol. For example, referring to Fig. 4, if the rightmost two states are cut in the top row, output $\overline{la}\downarrow$ is delayed, producing cut R2022. (The bottom right four states must also be removed for liveness by Eqn. 2). To obey R2 and prevent output $\overline{la}\downarrow$ being delayed by output $\overline{rr}\uparrow$, the right two states must also be removed in the second row. This produces the DI cut R2222. R1 and R2 are enforced by the following equation on both $\mathcal{L}$ and $\mathcal{R}$ cuts.

$$\mathcal{R}, \mathcal{L}: \quad a, b, c, d \text{ are even} \ \wedge \ a = b \ \wedge \ c = d \tag{6}$$

E. $\mathcal{L}$ and $\mathcal{R}$ Cut Lattices

The DI definition is a subset of the SI family. Rather than subtracting them out, we prefer to keep them all and refer to it as the DI/SI family. This results in 10 left cuts and 25 right cuts. Composing members of the $\mathcal{L}$ and $\mathcal{R}$ cuts to create protocol shapes gives 250 possible protocols, where 91 are not live as their cuts violate the liveness constraint Eqn. 4.

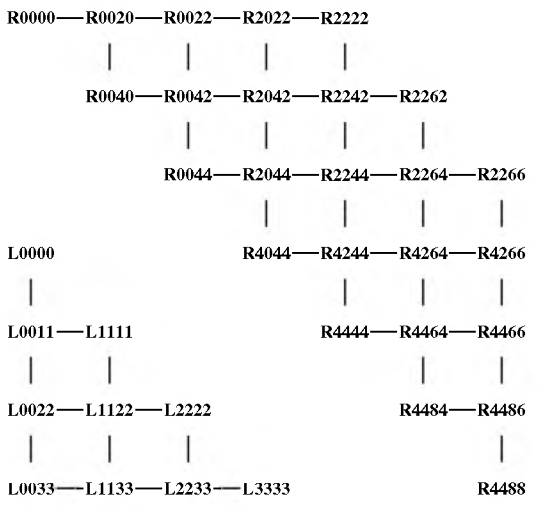


Fig. 10. The Symmetric Lattices of Untimed DI/SI Left and Right Cuts

Both sets of cuts form symmetric lattices and each cut has a complement. Both lattices are coherent and complete. They are both shown in Fig. 10. The lattice of left cuts has 10 members and is symmetric about an axis through cuts L0033 and L1122. Each cut Labcd has a complement given by L(3-d)(3-c)(3-b)(3-a). The cuts on the axis are self-complementary. The lattice of right cuts has 25 members and is symmetric about an axis through R2262, R2244, R4044 (which are self-complementary). Each cut Rabcd has a complement given by R(4-b)(4-a)(8-d)(8-c).

IV. Experimental DI/SI facts and patterns

This paper reports on homogeneous linear pipelines without feedback. Three important behaviors were revealed by experiments on these pipelines.

1) $PP_{w,d}$ (see Fig. 2) is independent of w. Seen from the outside, each structured parallel pipeline behaves like a single pipeline SP_d of the same depth. The behavior of these pipelines always emulates a DI protocol and can be predicted from the cuts of the shape.
2) There are only 23 possible structured parallel $PP_{w,d}$ behaviors and they are the 23 live DI shapes.
3) Some single pipeline behaviors can change shape when in single pipelines of depths 2 or more. Interestingly they may gain or lose states.

TABLE II
Pipeline Protocol Behaviors

	Basic shape	**SP_d shape**	**$PP_{w,d}$ shape**
DI	o o o ● o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o	o o o ● o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o	o o o ● o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o
REGULAR	. o o ● o o o o o o . o o o o . o o o o o o o o o . o o o o o o o o o	. o o ● o o o o o o . o o o o . o o o o o o o o o . o o o o o o o o o	o o o ● o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o o
2REGULAR	. . . ● o o o o o o . . o o o . . o o o . . . . o o o o o o o	. . . ● o o o o o o . . . o o . . . o o . . . . o o o o o o o	. . o ● o o o o o o . . o o o . . o o o o o o o o . . o o o o o o o o

Table II indicates the three possible categories of valid pipelined behavior that arise in DI/SI shapes:

1) DI: here *max* = L0000 ∘ R0000, retain their left and right profiles when pipelined singly or in parallel.
2) REGULAR: here L1111 ∘ R0000, retain their left and right profiles when singly piped, but gain state to a DI shape (in this case *max*) when piped in parallel.
3) 2REGULAR: here L2233 ∘ R0040, changes shape when singly pipelined (here it loses two left states and acts as L3333 ∘ R0040 from depths two onwards), and, when run in parallel, changes shape again to a DI protocol (to L2222 ∘ R0000, gaining two states on the left and four on the right).

A. Tableau of DI/SI experiments

The results of the experiments which cover the whole design space are displayed in a $\mathcal{L}$ by $\mathcal{R}$ tableau in Table III.

1) ●: a shape is DI if and only if both its cuts are DI.

TABLE III
TABLEAU OVER ALL DI/SI CUTS

L0000	L0011	L1111	L0022	L1122	L0033	L1133	L2222	L2233	L3333	L o R
●	□	Δ	●	Δ	Δ	Δ	●	□	Δ	R0000
□	□	□	□	□	□	□	□	□	□	R0020
Δ	□	Δ	Δ	Δ	Δ	Δ	Δ	□	Δ	R0040
●	□	Δ	●	Δ	Δ	Δ	●	□	Δ	R0022
Δ	□	Δ	Δ	Δ	Δ	Δ	Δ	□	Δ	R0042
Δ	□	Δ	Δ	Δ	Δ	Δ	Δ	□	.	R2022
Δ	□	Δ	Δ	Δ	Δ	Δ	Δ	□	.	R2042
●	□	Δ	●	Δ	Δ	Δ	●	□	Δ	R0044
□	□	□	□	□	□	□	□	□	.	R2044
Δ	□	.	Δ	.	Δ	.	.	.	.	R4044
●	□	Δ	●	Δ	Δ	Δ	●	□	.	R2222
□	□	□	□	□	□	□	□	□	.	R2242
Δ	□	Δ	Δ	Δ	.	.	Δ	.	.	R2262
●	□	Δ	●	Δ	Δ	Δ	●	□	.	R2244
Δ	□	Δ	Δ	Δ	.	.	Δ	.	.	R2264
Δ	□	.	Δ	.	Δ	.	.	.	.	R4244
Δ	□	.	Δ	.	.	.	.	.	.	R4264
●	□	Δ	●	Δ	.	.	●	.	.	R2266
□	□	.	□	.	.	.	.	.	.	R4266
●	□	.	●	.	Δ	.	.	.	.	R4444
□	□	.	□	.	.	.	.	.	.	R4464
Δ	.	.	.	.	.	.	.	.	.	R4484
●	□	.	●	.	.	.	.	.	.	R4466
Δ	.	.	.	.	.	.	.	.	.	R4486
●	.	.	.	.	.	.	.	.	.	R4488
●: 23			Δ: 76		□: 60		.: 91			/250

2) Δ: a shape is regular if and only if both its cuts are regular or just one is DI.
3) □: a shape is 2regular if one or both of its cuts is/are 2regular.
4) .: shows a non-live protocol.

One striking result is that the $\mathcal{L}$ and $\mathcal{R}$ cuts have orthogonal and persistent behavior. For example, L0011 cuts 2regularly (in the same way) whichever $\mathcal{R}$ cut it is composed with. Similarly, R0040 cuts regularly whichever $\mathcal{L}$ cut it is composed with.

The tableau is divided into blocks with a DI shape ● in its top-left corner. This is the most state rich shape in that block. The least state rich shape sits in its bottom right corner (only six of these are live). All shapes in a specific block have the same parallel pipelining behavior. Mathematicians may prefer DI shapes because they retain that shape when pipelined; engineers may prefer others if they give rise to faster or lower power implementations for the same pipelining behavior. For example, the designs `KG`, `BCKLLS` and `FD6` by Kol and Ginosar, Blunno *et al*, and Furber and Day respectively all have *max*'s behavior when composed into parallel pipelines. The doubly latched `KG` even has *max*'s behavior when composed into a serial pipeline of depth two or more. Within each block, any 2regular shape will change shape to a regular or DI shape from depth two when singly pipelined. Any regular or 2regular shape will change to the DI shape of its own block when pipelined in parallel, even from depth one. So the 23 viable DI blocks have a significant underlying structural significance which would not have been revealed had we not experimented with parallel pipelines.

Finally the tableau splits into 3 levels. All shapes with $\mathcal{R}$ cuts between R0000..R2262 can achieve full occupancy, and between R2244..R4264 half occupancy. The rest are unpipelined.

V. CONCURRENCY REDUCTION

The remainder of the paper discusses a study to determine the impact of concurrency reduction on a protocol family. All protocols are derived from the single most concurrent protocol shape based on systematic concurrency reduction rules of the left and right cuts. All of the pipelined protocols in this untimed family were synthesized, place and routed, and their physical designs were characterized in order to perform this evaluation.

The theoretical part of the paper defines a complete protocol family consisting of 137 different specifications. This provides perhaps the first opportunity to perform a large scale systematic study of the effect of regular concurrency reduction upon a complete class of protocols. The theory identifies the fact that many specifications are indistinguishable when placed in pipelines that are common design topologies. It also shows that many properties of the protocols are persistent. This opens up a choice space for a designer to pick amongst various designs in order to match a particular requirement and optimize the metrics most important for the design while meeting specific protocol requirements.

A tractable design space is presented by abstracting out the data path logic. However, this also results in substantial inaccuracy in the reported results if the goal is to build controllers that include a data path. We expect there to be a substantially larger penalty in implementing the latch clocking signals for the more concurrent protocols than for the protocols of lesser concurrency.

Concurrency reduction results in a complex interplay between logic level optimization and system level interaction across the handshake channels. In general, concurrency reduction tends to reduce the complexity of the logic which can speed up the response time of the controller. However, it can also result in system level performance degradation by delaying output signals on a channel when they otherwise would be able to proceed in a more concurrent protocol. Thus, some amount of concurrency reduction produces an improved design, but too much can degrade performance.

Our initial hope was to find that concurrency reduction produced a convex function that placed the optimal design somewhere in the middle of the concurrency reduction spectrum. We hoped this would be true both globally across the

TABLE IV
NUMBER OF STATE VARIABLES GENERATED BY PETRIFY

L0000	L0011	L1111	L0022	L1122	L0033	L1133	L2222	L2233	L3333	L ∘ R
-	-	2	4	-	-	2	2	2	2	R0000
3	-	2	2	2	2	1	2	1	1	R0020
2	2	2	2	2	2	1	2	1	1	R0040
3	3	2	1	2	2	1	2	1	1	R0022
2	2	2	-	2	2	1	2	1	1	R0042
2	2	2	2	2	2	1	2	1	D	R2022
2	1	1	1	1	1	0	1	0	D	R2042
3	2	2	2	2	2	1	2	1	1	R0044
2	2	1	1	1	1	0	1	0	D	R2044
2	1	D	1	D	1	D	D	D	D	R4044
2	2	2	2	2	2	1	2	1	D	R2222
2	1	1	2	1	1	0	1	0	D	R2242
2	1	1	1	1	D	D	1	D	D	R2262
2	1	1	1	1	1	0	1	0	D	R2244
2	1	1	1	0	D	D	1	D	D	R2264
2	1	D	1	D	1	D	D	D	D	R4244
1	0	D	0	D	D	D	D	D	D	R4264

TABLE V
CONTROLLER AREA IN μM^2

L0000	L0011	L1111	L0022	L1122	L0033	L1133	L2222	L2233	L3333	L ∘ R
-	-	271.6	387.7	-	-	244.3	350.2	203.2	196.4	R0000
404.8	-	285.3	288.7	240.9	302.3	217.0	223.7	237.4	189.6	R0020
261.3	339.9	292.1	264.7	339.9	309.1	244.3	206.6	213.5	240.9	R0040
346.7	428.7	203.2	213.5	333.1	210.1	206.6	175.9	155.4	162.2	R0022
418.5	268.2	257.9	-	281.8	331.1	206.6	251.1	189.6	172.5	R0042
319.4	350.2	333.1	206.6	285.3	298.9	196.4	179.3	179.3	D	R2022
398.0	268.2	213.5	196.4	203.2	206.6	169.1	175.9	162.2	D	R2042
247.6	196.4	237.4	162.2	199.9	227.1	172.5	206.6	152.0	155.4	R0044
206.6	210.1	186.1	189.6	186.1	189.6	162.2	124.7	155.4	D	R2044
278.4	193.0	D	155.4	D	165.6	D	D	D	D	R4044
350.2	213.5	179.3	165.6	172.5	162.2	152.0	162.2	124.8	D	R2222
288.7	223.7	186.1	199.8	172.5	186.1	138.3	131.5	131.5	D	R2242
220.3	206.6	175.9	165.6	162.2	D	D	155.4	D	D	R2262
179.3	162.2	128.1	117.8	134.9	128.1	104.2	97.3	83.7	D	R2244
196.4	182.7	152.0	128.1	134.9	D	D	107.9	D	D	R2264
189.6	169.1	D	134.9	D	121.2	D	D	D	D	R4244
145.2	148.6	D	104.2	D	D	D	D	D	D	R4264

full design space as well as locally inside protocol equivalence classes. While this trend does generally hold, there is substantial noise with significant exceptions to the norm.

In the end we are interested in implementing efficient systems and are searching to find more efficient designs than have heretofore been discovered. The exhaustive nature of this evaluation points to locations in the design space to search for design optimization. Our literature search uncovered published implementations for only 40 of the 137 untimed shapes [3], [6], [8], [9], [10], [13], [14], [15], [16], [22], [25], [26]. None of those had the same protocol as the best-in-class in the study performed here.

VI. CIRCUIT CHARACTERIZATION

The complete set of abstracted controllers were synthesized, placed and routed, and characterized using post layout extraction employing the static Artisan 12T library on IBM's 65nm 10sf process [23]. Each of the controllers has been characterized for cycle time, forward and backward latency and area. Power results are not reported here. Simulation is performed in ModelSim on the post layout design with delays extracted from SoC encounter based on the layout and parasitics of the design. Most of the flow has been fully automated since we started with 137 different pipelined controllers.

The characterization starts with the controller behavior specified as a state graph "shape" in CCS. The specification is then synthesized and technology mapped to the Artisan library. Petrify is used to synthesize the design and tech map to the Artisan library [4]. We created a technology mapping file in the genlib format that was used by Petrify and applied the exhaustive decomposition algorithm. Of the 137 controllers,

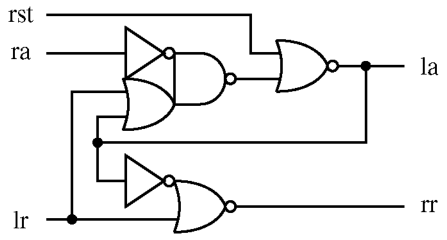


Fig. 11. Controller with smallest area and backward latency: L2233 ∘ R2244

Petrify could not find a valid state variable assignment for six controllers, including *max*.

Petrify has the capability of adding reset to a design. However those results are inconsistent, and so we opted to manually add reset to the controllers. This process was aided by `Verilog2CCS` software that we wrote to map a Verilog module to a formal CCS specification. Verilog2CCS forces the inputs low and simulates the Verilog to determine the logic value of each net. If any nets are undefined, reset is added by modifying one or more gates to drive the net to the proper state. An attempt was made to have reset create as small an impact on area and power as possible.

Characterization is performed by placing each structural Verilog controller into a 4-deep linear pipeline. This pipeline is sufficient to evaluate our design metrics. Simple behavioral interfaces are added to the left and right side of the pipeline that enable simulation control of the external handshake channels. The left interface implements `assign ack = go_l & ~req` and the right interface `assign ack = go_r & req`. Latency through the left and right interfaces is an AND or NOR gate. This must be no

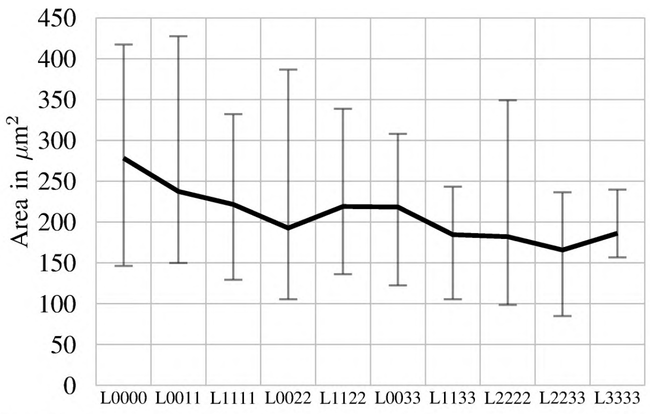


Fig. 12. Area averaged across left cuts. Interval shows largest and smallest values for the cut, line passes through the mean.

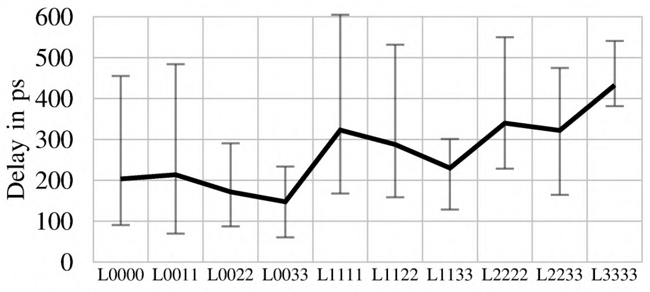


Fig. 13. Forward latency based concurrency reduction of the output channel

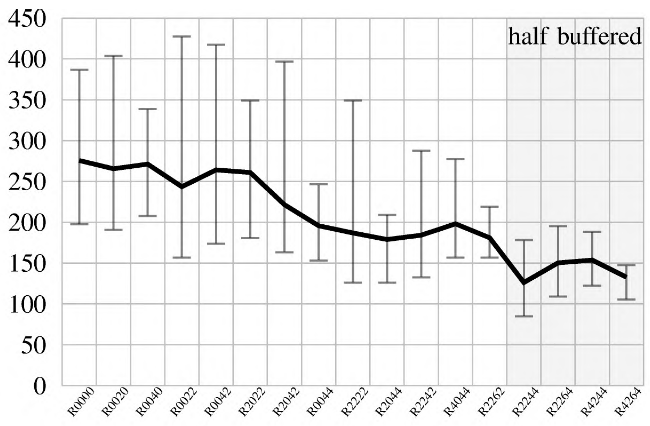


Fig. 15. Area averaged across right cuts

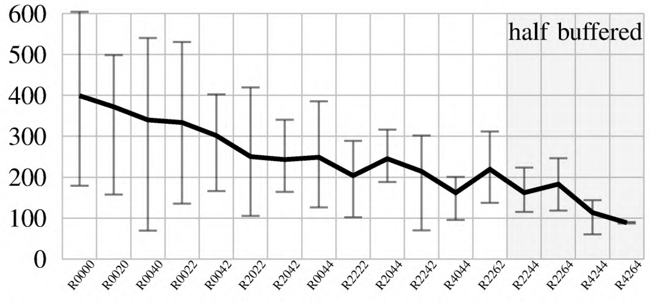


Fig. 16. Forward latency based on input channel concurrency reduction

greater than the delay through the controllers to avoid limiting the cycle time of the controller in the characterization step.

Each design is synthesized with design compiler (DC). Modification of logic and sizing of the gates was prevented by adding the `set_dont_touch` constraint. The structural controllers used the same "X2" drive strength for all library cells. Therefore, the designs were not optimized for power and performance as our full relative timing flow is not yet automated [20]. Physical place and route using SoC Encounter was performed to obtain the physical design. SoC Encounter is also used to produce accurate power numbers from the activity factors from simulation using extracted parasitics.

VII. Results

Results are only reported for the subset of protocols that can be pipelined. This eliminates the 22 valid unpipelined protocols found in the bottom box of Table III.

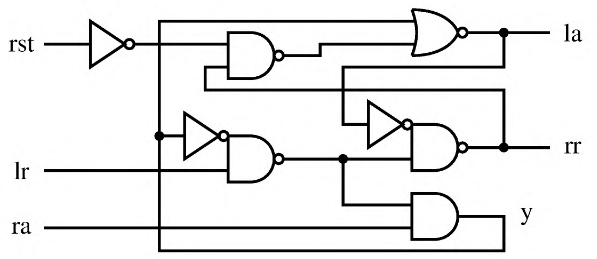


Fig. 14. Half buffered L0033 ∘ R4244 circuit gives smallest forward latency

A. State Variables and Controller Area

A reasonable measure of the controller complexity is the number of state variables that are used in the implementation. At least 16 states must be removed to avoid any explicit state variables. We consider a controller to have a state variable for each independent feedback variable required. An output that feeds back to other output logic cones is not counted.

The number of the state variables added by Petrify is reported in Table IV and the area of the implementations in Table V. Concurrency reduction tends to decrease the area and the number of state variables. These two values are are correlated, as average area scales as 408.2, 313.0, 243.8, 181.0, 135.8 for implementations using from four to zero state variables. Note that the right-most and bottom-most row in each DI protocol equivalence class box contains the fewest number of state variables. One might expect this to correlate with the best performance for the class, which will be validated in Section VII-D. Fig. 11 is the design with the smallest area.

Figures 12 and 15 graph the area of the left and right cuts. The intervals show the maximum and minimum values for each cut and the line crosses through the average. Area generally decreases with increasing concurrency reduction both globally and inside DI equivalent class sets (the boxes in the tables).

B. Forward Latency

Forward latency is measured as the delay in propagating a handshake from the input to output channel of an idle

TABLE VI
FORWARD LATENCY IN PS. OPTIMAL AVERAGE CUT IN **bold**.

L0000	L0011	L1111	L0022	L1122	**L0033**	L1133	L2222	L2233	L3333	L ∘ R
-	-	607	177	-	**-**	257	552	402	401	R0000
457	-	497	200	318	**155**	269	478	474	501	R0020
339	367	387	150	534	**234**	303	366	477	543	R0040
133	486	325	293	533	**236**	273	362	317	379	R0022
186	309	291	-	331	**164**	265	405	383	382	R0042
103	210	422	203	291	**107**	253	295	370	D	R2022
289	343	293	206	234	**188**	206	267	162	D	R2042
124	199	301	157	275	**130**	240	375	299	388	R0044
262	220	255	225	273	**186**	214	319	253	D	R2044
93	203	D	180	D	**172**	D	D	D	D	R4044
291	204	198	180	182	**100**	194	242	243	D	R2222
244	206	271	174	227	**68**	158	304	277	D	R2242
135	214	304	152	198	**D**	D	314	D	D	R2262
195	126	165	113	186	**118**	126	226	205	D	R2244
198	177	202	116	156	**D**	D	249	D	D	R2264
115	146	D	135	D	**58**	D	D	D	D	R4244
88	93	D	85	D	**D**	D	D	D	D	R4264

TABLE VII
BACKWARD LATENCY IN PS, OPTIMAL FULL BUFFERED CUT IN **bold**

L0000	L0011	L1111	L0022	L1122	L0033	L1133	L2222	L2233	L3333	L ∘ R
-	-	178	579	-	-	302	249	195	165	R0000
299	-	188	492	237	239	233	228	235	174	R0020
278	647	308	446	484	300	248	193	274	283	R0040
292	117	232	274	306	290	257	154	196	180	R0022
416	378	261	-	330	368	285	221	216	203	R0042
286	**168**	**189**	**184**	**178**	**221**	**174**	**155**	**161**	**D**	**R2022**
370	468	274	225	228	263	221	192	173	D	R2042
499	439	292	332	357	445	327	219	263	339	R0044
465	438	385	324	340	375	338	355	293	D	R2044
658	458	D	392	D	419	D	D	D	D	R4044
538	400	376	320	346	274	288	275	183	D	R2222
357	402	333	286	300	303	259	270	219	D	R2242
341	477	463	315	339	D	D	342	D	D	R2262
175	124	120	131	115	135	97	108	78	D	R2244
177	219	195	157	127	D	D	138	D	D	R2264
153	185	D	141	D	121	D	D	D	D	R4244
147	142	D	140	D	D	D	D	D	D	R4264

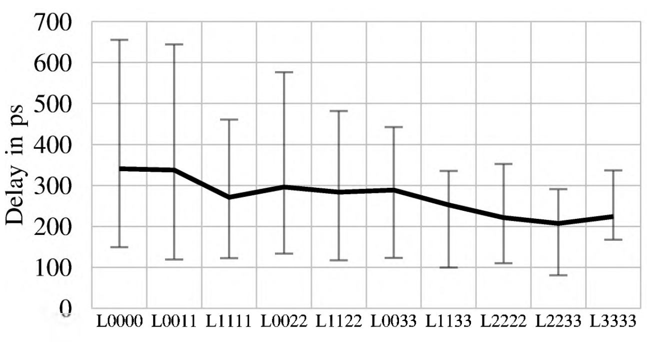


Fig. 17. Backward latency averaged across left cuts

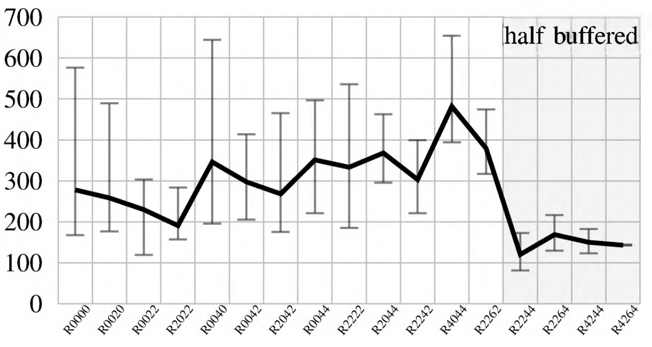


Fig. 18. Backward latency averaged across right cuts

controller. Specifically, this is the delay from $lr\uparrow$ to $rr\uparrow$ in an idle controller. Our simulations measured the delay across the four pipeline stages in the design and then divide this delay by four to get the latency per controller. Table VI shows forward latency in picoseconds.

Concurrency reduction on the incoming channel generally reduces the latency as shown in Fig. 16. This improvement is directly related to reduction in the complexity of the design as concurrency is reduced. This effect of concurrency reduction is similar to that of area and state variable reduction.

Concurrency reduction on the outgoing channel displays an interesting competition between concurrency reduction that increases protocol latency and decreases controller latency (Fig. 13). Left cuts that delay $rr\uparrow$ will substantially retard forward latency due to the reduction in protocol concurrency. This occurs when the first components (La and Lb) of the left cut increase. Thus as concurrency is reduced from L00xx to L11xx and so forth, the delay of $rr\uparrow$ increases substantially. However, the Lc and Ld components in the left cut generally decrease forward latency through logic simplification.

The fully buffered circuit synthesized by Petrify with the smallest forward latency is shown in Fig. 14. This design, L0033 ∘ R4244, contains the maximal right cut and maximal left cut where the La and Lb cuts are zero. The smallest forward latency in a fully buffered design is L0033 ∘ R2242, just 10ps slower. Note that for the same amount of buffering in an application such as a FIFO, a half buffered protocol needs to pass through twice as many controllers. Therefore the fastest forward latency for the same amount of buffering is 116ps for the best half buffered protocol versus 68ps for the best fully buffered protocol.

C. Backward Latency

There are various ways of measuring backward latency. Here we define backward latency as the delay in a stalled pipeline from the time that the output channel acknowledges

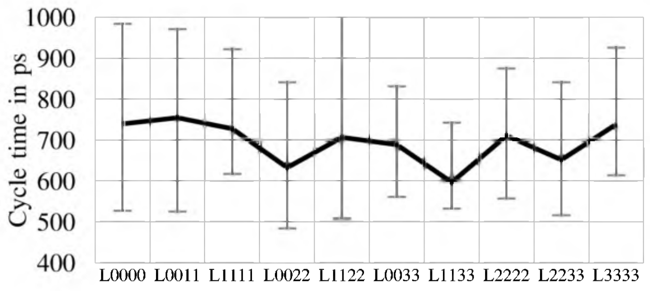


Fig. 19. Cycle time averaged across left cuts

TABLE VIII
CYCLE TIME IN PS

L0000	L0011	L1111	L0022	L1122	L0033	**L1133**	L2222	L2233	L3333	L ∘ R
-	-	833	844	-	-	**745**	848	672	633	R0000
861	-	751	841	606	834	**607**	804	808	754	R0020
669	974	726	749	1077	673	**601**	631	844	929	R0040
688	826	615	632	935	620	**563**	588	564	612	R0022
773	723	621	-	717	801	**592**	701	675	673	R0042
674	880	925	505	762	801	**562**	632	681	D	R2022
892	**873**	**632**	**482**	**507**	**592**	**530**	**555**	**514**	**D**	**R2042**
678	703	661	530	687	750	**636**	654	638	832	R0044
800	736	692	641	698	648	**611**	754	627	D	R2044
804	728	D	643	D	694	**D**	D	D	D	R4044
987	708	627	606	630	613	**611**	623	522	D	R2222
732	730	636	607	650	652	**542**	733	641	D	R2242
580	823	894	581	651	D	**D**	808	D	D	R2262
790	547	623	620	656	722	**566**	745	643	D	R2244
804	844	849	674	609	D	**D**	878	D	D	R2264
590	710	D	601	D	559	**D**	D	D	D	R4244
525	523	D	576	D	D	**D**	D	D	D	R4264

that data has been latched until the input channel begins the return-to-zero transitions. Specifically, this is the delay from $ra\uparrow$ to $la\uparrow$ in these controllers. Our simulations filled the four-deep pipeline with the maximum number of tokens and then measured this delay by allowing the output channel to accept the token, measuring the delay to $la\uparrow$ on the input channel. This value was divided by four to get the average per controller in the 4-deep pipeline. Backward latency is shown in Table VII. Figure 11 shows the controller with the smallest backward latency.

Concurrency reduction on the outbound channel generally reduces the latency as shown in Fig. 17. This improvement is a second order effect and is directly related to reduction in the complexity of the design as concurrency is reduced.

Concurrency reduction on the inbound channel shows some very interesting properties. Consider full buffered protocols. Performance initially improves and then dramatically decreases. This can be explained by referring to our shape in Fig. 4. The final state in the first two rows of the shape are reached in a fully stalled pipeline where the second data token is offered on the input channel. If the last state on the second row exists, then maximal progress has been made in a stalled pipeline. The tail on the right of the shape in the last two rows allows a protocol to quickly respond from a stalled condition. If this tail is removed, then backward latency will be significantly impacted due to a lack of protocol concurrency. The tail will be removed with Rc ≥ 4 (Rxx4x) cuts. R2022 has been observed as the optimal average right cut for backward latency in fully buffered protocols as shown in Fig. 18. This cut applies the the largest possible amount of concurrency reduction without removing the top first states of the “tail” in the third row of the shape that would negatively impact protocol concurrency.

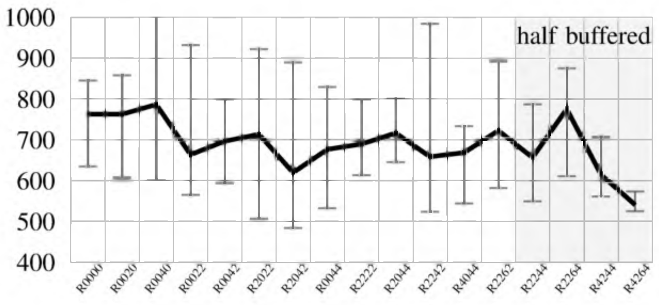


Fig. 20. Cycle time averaged across right cuts

Half buffered protocols show an interesting phenomenon with a substantial reduction in backwards latency. These controllers only store data in ever other latch when stalled. This results in an interesting artifact where every other controller stalls in a different location in the shape, one waiting for rising $ra\uparrow$, the other for falling $ra\downarrow$. This results in a very fast backward latency. However, note that since these protocols only store data in half the latches, they need to pass through twice as many controllers for equal storage as the full buffered protocols (or $2\times$ the latency shown). Taking that into account, they are slower than the best full buffered protocols.

D. Cycle Time

Cycle time provides information about the throughput of the pipeline. It is measured as the largest delay between the insertion of two tokens in the pipeline. Twelve tokens are inserted into an empty four-deep pipeline as fast as the pipeline will accept. All tokens are immediately consumed at the output channel. The slowest delay between the insertion of two adjacent tokens is recorded as the cycle time. Note that for this number to be correct, the left and right pipeline interfaces that control token and bubble insertion must have a cycle time less than the controller itself. The cycle time of all the controllers is listed in Table VIII.

Cycle time averaged across different left cuts is graphed in Figure 19. This graph shows that for left cuts the controller with the best average performance is near the middle of the concurrency reduction range. On average, cut L1133 results in the best throughput. Figure 20 graphs cycle time averaged across different right cuts. This graph also demonstrates a tradeoff between circuit simplicity and protocol handshake delays for full buffered protocols. The best delay lies in the

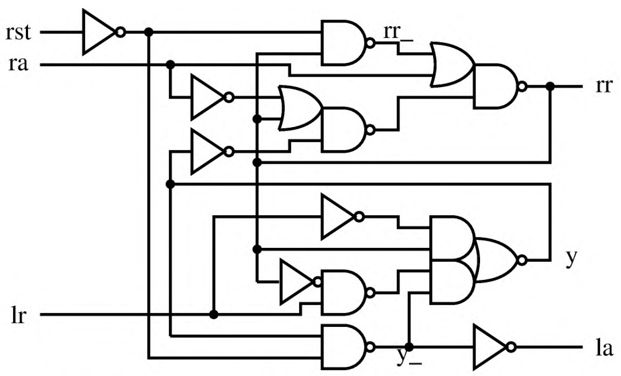


Fig. 21. Controller with the smallest cycle time: L0022 ◦ R2042

middle of the concurrency reduction spectrum, where right cut R2042 has the lowest average as well as the implementation with the highest throughput. However, the few live half buffer protocols with the largest concurrency reduction show surprisingly fast cycle times. Reducing concurrency beyond the half buffered to unpipelined protocols (which are not shown) will give dramatic increases in cycle times.

E. Performance Summary

The concurrency reduction tradeoff between circuit simplicity that speeds up a design and protocol penalty slowing it down is generally supported by the data that we have collected on the 131 synthesized controllers in this family. Our study shows the optimal choice lies somewhere in the center of the spectrum. The best full buffered design does not lie at either end of the concurrency reduction spectrum, and indeed lies somewhere near the middle. This generally holds for averages across cuts as well as the fastest designs. The tradeoffs seem to hold as the implementations that strike the right balance between protocol concurrency and logic simplicity should be in between the ends of concurrency reduction spectrum. Highly concurrent specifications result in a complex implementation and thus will have a higher cycle time. Likewise protocols with high amounts of concurrency reduction will have channels that are idle waiting for a response from the other side of the controller. The best average forward latency cut is at the middle of the concurrency reduction spectrum, at L0033. Backward latency still supports the observation for fully buffered designs, although the optimal cut of R2022 is skewed more toward the protocols of lesser concurrency reduction. Throughput also supports our claim as cut L1133 is optimal on average. This cut is slightly skewed toward higher concurrency reduction. We were surprised to find the half buffered protocols, which have the largest amount of concurrency reduction, demonstrate outstanding backward latency and cycle times. This does not directly support our expectations, and indicates that these protocols produce excellent designs if pipeline stalls will not occur. If a pipeline stalls, the backward delays for the same amount of storage would double over what is shown in the graphs.

Our simulations did not directly measure the protocol penalty that exists when a channel is idle because an output

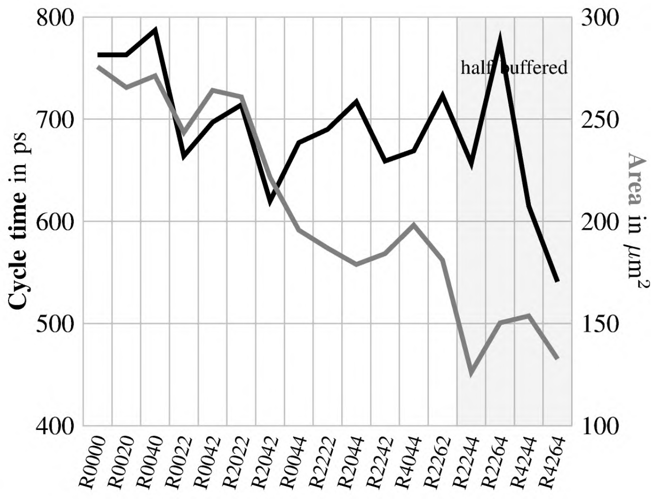


Fig. 22. **Area** and **cycle time** averaged across right cuts

is stalled in the controller or waiting unnecessarily on a signal from the other controller on the channel. However, we can estimate this penalty by comparing area and cycle time. Assume that performance increases linearly with area reduction and logic simplicity. This seems to roughly be the case for right cuts for forward latency, and somewhat for left cuts for backward latency. We can then plot area against cycle time as shown in Fig. 22. These graphs largely decrease in a similar manner until near the the optimal point of concurrency reduction, at which point the area continues to decrease but the cycle time increases. This divergence is likely based on idle time on channels due to too much concurrency reduction.

VIII. Validity of Results

We have formally verified that 72 of the 131 designs that Petrify could synthesize conform to the original specification. Only 6 of these do not require timing in order to function correctly. The other 66 have been push-button verified with automatic relative timing constraints generated by ARTIST [24]. We expect that the other 29 designs also conform to their specifications. This also demonstrates Petrify's impressive ability to faithfully implement each protocol.

IX. Conclusion

The results of a systematic investigation into the behaviors of asynchronous 4-phase latch controllers when composed into single and structured parallel pipelines is presented. Starting with the unique *shape* (minimized equivalent state graph) that exhibits maximal concurrency, rules are provided for systematically generating complete families of related but less state rich protocols by concurrency reduction rules that cut away selected states. The cutaways can be partitioned into left cuts $\mathcal{L}$ constraining outgoing channel signals and right cuts $\mathcal{R}$ constraining the incoming channel. Running though all combinations of $\mathcal{L}$ and $\mathcal{R}$ cuts generates families of untimed (and timed) protocols. Here we consider only untimed (delay insensitive and speed independent) protocols where data is bundled and valid before the incoming channel request ($lr\uparrow$) signal rises. The $\mathcal{L}$ and $\mathcal{R}$ cuts form separate symmetric lattices and give structure: they predict occupancy, the regularity of piped behaviors, and the behavior of non-homogeneous pipelines. The lattice product enables one to relate and compare protocols: it also reveals the design space.

The complete family of untimed four-phase asynchronous handshake controllers is characterized. This provides comparative data to help understand the effect of concurrency reduction on the area and performance (power was omitted for lack of space). The controllers are shown to be correct abstractions of controllers with full data path control. The logic is synthesized, placed and routed from formal specifications and then tech-mapped using the IBM 65nm 10sf Artisan Library. Reset was manually added to each controller.

We have shown that concurrency reduction generally increases the performance of designs up to a point, after which the designs begin to degrade in performance. We showed this is likely the case due to competing factors of overall faster designs as the circuits are simplified through concurrency reduction, versus larger protocol delays as certain handshake signals become stalled due to inefficiency in the protocols of highly reduced concurrency. Cycle time demonstrates this effect, as three of the four highest throughput designs are all full buffered near the middle of the concurrency reduction, with three being in the R2042 cut. There is a notable exception that half buffered protocols have some surprising efficiencies that make them more competitive than one might expect. This is particularly exaggerated with backward latency. The ultra inefficient unpipelined protocols, with high input channel concurrency reduction, were not included in the graphs.

A final contribution is the data that points engineers to designs that optimize each of the performance metrics. The best synthesized circuits are published for each metric as the complete design space was explored.

X. Acknowledgments

This work was supported in part through a gift by Sun Microsystems. We acknowledge the helpful suggestions of Jordi Cortadella and Luciano Lavagno in improving this document. Thanks are due to researchers who have explained or formally documented their circuits (usually as STG's or CSP) so they are clear to the community at large. This body of work enabled us to model real practical designs rather than experiment with a few idealized ones, kept us grounded, and was sufficiently large to help guide our research directions.

References

[1] Graham Birtwistle and Matthew Morley. Case Study: Specifying and Property Checking TK, and Asynchronous AMULET-like Microprocessor. In Alex Yakovlev and Reinder Nouta, editors, *Asynchronous Interfaces: Tools, Techniques, and Implementations"*, pages 13–22, July 2000.

[2] Graham Birtwistle and Kenneth S. Stevens. The family of 4-phase latch protocols. In *14th International Symposium on Asynchronous Circuits and Systems*, pages 71–82. IEEE, April 2008.

[3] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou. Handshake protocols for de-synchronization. In *International Symposium on Asynchronous Circuits and Systems*, pages 149–158. IEEE, Apr 2004.

[4] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.

[5] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, England, 1990.

[6] Paul Day and J. Viv Woods. Investigation into micropipeline latch design styles. *IEEE Transactions on VLSI Systems*, 3(2):264–272, June 1995.

[7] Aristides Efthymiou and Jim D. Garside. Adaptive pipeline structures for speculation control. In *Ninth International Symposium on Asynchronous Circuits and Systems*, pages 46–55. IEEE, May 2003.

[8] S. B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In *Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 11–16. IEEE Computer Society Press, March 1996.

[9] Stephen B. Furber. A small compendium of 4-phase macropipeline latch control circuits. Technical Report v0.3, 17/01/99, University of Manchester, Dept. of Computer Science, 1999.

[10] Stephen B. Furber and Paul Day. Four-phase micropipeline latch control circuits. *IEEE Transactions on VLSI Systems*, 4(2):247–253, June 1996.

[11] J. D. Garside, S. B. Furber, and S-H Chung. AMULET3 Revealed. In *5th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 51–59, April 1999.

[12] G. Graetzer. *Lattice Theory: First concepts and distributive lattices*. W. H. Freeman and Company, San Francisco, 1971.

[13] Rakefet Kol and Ran Ginosar. A doubly-latched asynchronous pipeline. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 706–711, Oct 1996.

[14] M. Lewis, J. D. Garside, and L. E. M. Brackenbury. Reconfigurable latch controllers for low power asynchronous circuits. In *International Symposium on Asynchronous Circuits and Systems*, pages 27–35, April 1999.

[15] Andrew M. Lines. Pipelined asynchronous circuits. Master's thesis, California Institute of Technology, Pasadena, CA, 1998.

[16] JianWei Liu. *Arithmetic and Control Componenets for an Asynchronous System*. PhD thesis, Department of Computer Science, University of Manchester, 1997.

[17] Peggy B. McGee and Steven M. Nowick. A Lattice-Based Framework for the Classification and Design of Asynchronous Pipelines. In *Proceedings of the Digital Automation Conference (DAC05)*, pages 491–496. IEEE/ACM, June 2005.

[18] Robin Milner. *Communication and Concurrency*. Computer Science. Prentice Hall International, London, 1989.

[19] Faron G. Moller and Perdita Stevens. *The Edinburgh Concurrency Workbench (Version 7)*. University of Edinburgh, October 1992.

[20] Kenneth S. Stevens, Yang Xu, and Vikas Vij. Characterization of Asynchronous Templates for Integration into Clocked CAD Flows. In *15th International Symposium on Asynchronous Circuits and Systems*, pages 151–161. IEEE, May 2009.

[21] Colin Stirling. An Introduction to Modal and Temporal Logics for CCS. In A. Yonezawa and T. Ito, editors, *Concurrency: Theory, Language, and Architecture*, number 491 in LNCS, pages 2–20. Springer-Verlag, 1991.

[22] Ivan E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989. Turing Award Paper.

[23] Santosh N. Varanasi. Performance Analysis of Four-Phase Untimed Asynchronous Handshake Protocols. Master's thesis, University of Utah, Salt Lake City, Utah, May 2009.

[24] Yang Xu and Kenneth S. Stevens. Automatic Synthesis of Computation Interference Constraints for Relative Timing. In *26th International Conference on Computer Design*, pages 16–22. IEEE, Oct. 2009.

[25] Eslam Yahya and Marc Renaudin. QDI Latches Characteristics and Asynchronous Linear-Pipeline Performance Analysis. In *Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation*, Lecture Notes in Computer Science, pages 583–592. Springer, 2006.

[26] Kenneth Y. Yun, Peter A. Beerel, and Julio Arceo. High-performance asynchronous pipeline circuits. In *Second International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 17–28. IEEE Computer Society Press, March 1996.