The Family of 4-phase Latch Protocols

Graham Birtwistle
DCS, Sheffield
graham@dcs.shef.ac.uk

Kenneth S. Stevens
ECE, University of Utah
kstevens@ece.utah.edu

Abstract

A complete family of untimed asynchronous 4-phase pipeline protocols is derived and characterised. This family contains all untimed protocols where data becomes valid before the request signal rises. Starting with a specification of the most parallel such protocol, rules are provided for concurrency reduction to systematically generate the family of all 137 related protocols that can be pipelined. Graphical and textual nomenclatures are developed to represent protocol properties and behaviours. The protocols are categorised according to their behaviours when composed into linear and structured parallel pipelines. Six basic categories emerge, along with several properties such as a single state that determines whether a protocol is fully or half buffered. When equivalence classes are calculated for parallel pipeline behaviours they are dominated by 15 shapes (all of which are delay-insensitive) which are related by a simple lattice. Several published circuits are shown to map to 16 of our 137 family members. This work enhances the understanding of handshake protocols, their properties, and relationships between different implementations in terms of concurrency and behavioural properties.

1. Introduction

Asynchronous request acknowledge protocols have been employed for years. Yet it is surprising how little is understood of the fundamental behaviour of the protocols when they are composed into systems. This work formally and exhaustively investigates all possible untimed asynchronous latch controller protocols. The behaviour of each protocol is then investigated in linear and parallel configurations to study its concurrent behaviour. A number of properties emerge such as protocol equivalence classes, protocol compatibility sets, behavioural properties such as the ability to latch data in every latch, control the latch without extra state logic, and full lattice representation.

We have found Milner's CCS (Calculus of Communicating Systems) [10] to be very apt notation for studying protocol families in this way: it is expressive enough to model signal protocols; its semantics conveniently capture event orderings rather than specific timings; and it is compositional which makes it straightforward to model both linear and parallel pipelines. Further, latch protocols and pipeline structures can be compressed down to the minimal canonical state graph and property checked on CCS's supporting software, the public domain CWB (Concurrency Workbench) [11]. CCS has also been extended to directly support circuit realisations with speed-independent broadcast communication [12].

Our technique is quite straightforward: the most parallel behaviour of a 4-phase latch controller is first described in CCS and the CWB is used to generate its equivalent state graph (32 states). Using a few concurrency reduction rules, states are systematically cut-away on the incoming and outgoing channels to generate all less concurrent state graphs that will still obey some related latch controller protocol. The cut-aways are exhaustive: all possible protocols in the family are generated as minimised state graphs. Notice that this paper only describes concurrency reduction for untimed protocols. The rules for timed protocols (burst-mode or relative timed) will be presented elsewhere.

The CCS notation allows us to compose parallel specifications of the channels with their concurrency reducing synchronisation, and reduce these to canonical state-graph specifications. The composition of parallel protocols and their systematic reduction to a minimal canonical representation renders comparison between implementations trivial and assists in validating completeness. Such transformations are not readily possible with STG and Petri-net specifications.

1.1. Previous Work

Figure 1 shows *LC*, a 4-phase latch controller, and its associated latch where the data is stored. The input (upstream) channel handshakes with *lr* (the left request) and $\overline{la}$ (the left acknowledgment), and the output (downstream) channel with $\overline{rr}$ (the right request) and *ra* (the right acknowledgment). Each channel employs the simple protocol of interleaving request and acknowledgment signals. By convention we overline output signals but not input signals. Note that in this work we have abstracted out the data-path, and only model the proto-
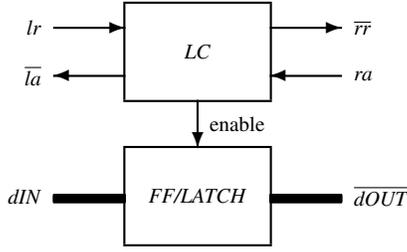
**Figure 1.** LC**: the generic latch protocol**
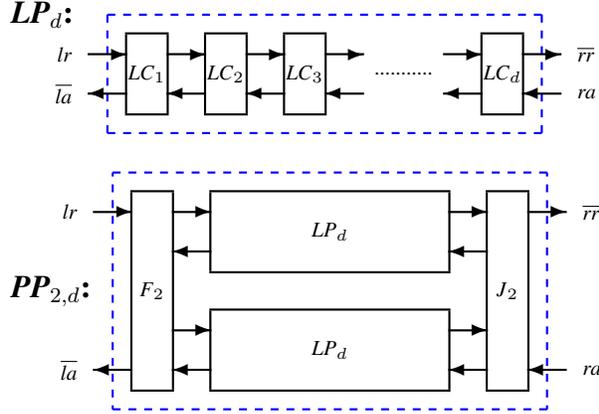
Figure 2. Linear and Parallel Pipelines

col on the handshake pins of the two channels. Work is in progress on modeling the latch *enable* signals for normally open and normally closed protocols and will be reported elsewhere.

We quickly review the relevant results presented in [1]. That work made no attempt to be complete and considered only four published 4-phase latch controllers and three idealised protocols. Besides modeling these seven latch controllers singly, it also considered their behaviours when composed into structured pipelines:

1. LP_d: a linear pipeline of latch protocols of depth d (see the top part of Figure 2).
2. $PP_{w,d}$: the structured composition of w d-deep pipelines running in parallel (see the lower part of Figure 2). The fork module F_2 broadcasts lr to both linear pipelines and waits until all have replied before responding with $\overline{la}$. The join module J_2 is the inverse of F_2.

Notice that the specifications of LC, LP_d and $PP_{w,d}$ are all in terms of lr, $\overline{la}$, $\overline{lr}$, ra and can thus be compared and contrasted directly.

The Manchester group has published several 4-phase latch controller circuits, some faster, some more power efficient. One source of variety is the amount of overlap permitted between the recovery phase on the $\overline{rr}\downarrow/ra\downarrow$

side and the notification of the arrival of the next data value ($lr\uparrow$). It is thus not surprising that the Manchester 4-phase latches studied vary in state size (from roughly 18 to 26 states). When combined into linear pipelines LP_d, their minimised behaviours settled into predictable pattern of state sizes from pipeline depth 2; whereas the parallel pipeline patterns $PP_{w,d}$ were regular from depth 1, but did not agree with the linear pipeline pattern ($PP_{w,d}$ was always more state rich). Three new mathematically inspired 4-phase protocols were also examined, two of which did exhibit stable behaviour in that $PP_{w,d} \equiv LP_d$ for positive w and d. One of these protocols has 32 states and is the most parallel 4-phase latch protocol achievable.

With a little modeling and analysis on the CWB, it was easy to show the following results for the latches considered:

1. LC, a single latch protocol, may have between 16..32 states.
2. LP_d and $PP_{w,d}$ usually have O(16d) states, but this will be O(8d) if values are latched in alternate stages.
3. for the 4 Manchester designs, $LP_d \not\equiv PP_{w,d}$.

These 4-phase results were checked by running CCS models on the CWB for w,d = 1..8. In addition several equivalent 2-phase results were given formal proofs. None of these are deep, rather they are case rich, shallow and tedious. Preliminary work on the 4-phase proofs shows them to be similarly structured and yet more case rich and tedious. It would be nice to get the proofs mechanised and verified with a proof checker such as HOL.

1.2. Structure of the Paper

The structure of the rest this paper is as follows. In Section 2, we present our specification notation CCS and construct a specification of the most concurrent 4-phase latch protocol LC_{max} which has 32 states when expressed in our *normal form* as a minimised state graph. In Section 3 we show how the whole family of less state rich (less parallel) 4-phase latch protocols can be derived from LC_{max} through concurrency reduction. Each sub-behaviour is expressed as a state graph and given a unique characterisation. We also tabulate the behaviours when pipelined singly and in parallel. In Section 3.3 we discuss six protocol categories that emerge, including the 15 protocols that are **stable**: for them $LP_d \equiv PP_{w,d}$. An important side effect for designs with this behaviour is that we can replace quite complicated formal models of parallel $PP_{w,d}$ datapaths by the much simpler LP_d model when reasoning about concurrent pipelined designs such as a microprocessor. In Section 4 we partition the state space into protocol equivalence classes when pipelined in linear and

parallel configurations. Section 5 presents the stable circuits (and hence their equivalence classes) into a lattice based on concurrency. Section 6 ties in related work and Section 7 lists some published designs and places them in our family lattice. Finally we summarise the work done so far and some future directions.

2. LC_{max}: The Maximal 4-phase Protocol

In this section, we model the behaviour of LC_{max}, the 4-phase latch protocol of maximal concurrency, and display its regular behaviour when composed into pipelines. In CCS, our first step in specifying LC_{max} is to describe as the composition of L which deals with the incoming channel and R which deals with the outgoing channel.

$$
\begin{aligned}
L &= lr\uparrow \ . \ \overline{la}\uparrow \ . \ lr\downarrow \ . \ \overline{la}\downarrow \ . \ L \\
R &= \overline{rr}\uparrow \ . \ ra\uparrow \ . \ \overline{rr}\downarrow \ . \ \overline{ra}\downarrow \ . \ R \\
LC_{con} &= (\ L \mid R \ )
\end{aligned}
$$

The definition of L simply spells out the order of one cycle of input signals and then repeats forever. The signals are separated by '.' which we may interpret informally as signal precedence. CCS specifies the order of events, but they occur with arbitrary delays rather than strict timings, resulting in all possible concurrent signal interleavings. The protocol it describes may accordingly take an arbitrary time between these signals. The definition of R follows the same pattern. The above specification minimises to a 4×4 block of states which (with loop back) and via the semantics of CCS, covers every possible interleaving of the 8 signals, from one extreme just L running and to the other just R running and all intermediate possible interleavings.

However L and R can not run untrammeled. We now add synchronisations that will (1) ■ stop L from accepting fresh data when the previous data value has not been accepted downstream, and (2) ● stop R from emitting an $\overline{rr}\uparrow$ until a fresh value has been latched. The second version of the specification of LC_{max} indicates how to handle these interplays between L and R:

$$
\begin{aligned}
L &= lr\uparrow \ . \ \blacksquare \ . \ \bullet \ . \ \overline{la}\uparrow \qquad . \ lr\downarrow \ . \ \overline{la}\downarrow \ . \ L \\
R &= \bullet \ . \ \overline{rr}\uparrow \qquad . \ ra\uparrow \ . \ \blacksquare \ . \ \overline{rr}\downarrow \ . \ \overline{ra}\downarrow \ . \ R \\
LC_{max} &= (\ L \mid R \ )
\end{aligned}
$$

1. ■ : after R has received a signal $ra\uparrow$, it is sure that the current data value has been captured downstream. R will now unblock L (if it were blocked). Both L and R may continue on.
2. ● : with space assured, (the unblocked) L is free to capture next fresh data value and then unblock R (if it were blocked). Both L and R may continue on.

Notice the ordering $lr\uparrow.\blacksquare.\bullet...$ in process L. Channel L must have an empty latch (make sure that R has received $ra\uparrow$) before it stores the next value on dIN in the latch. For channel R the conditions are reversed.

We may remark that whereas the placing of the receiving ■ in L and ● in R are crucial, it is quite in order to shuffle the awakening ● to the right in L and the awakening ■ to the right in R. All such shufflings are captured by our cut-away method described in Section 3.
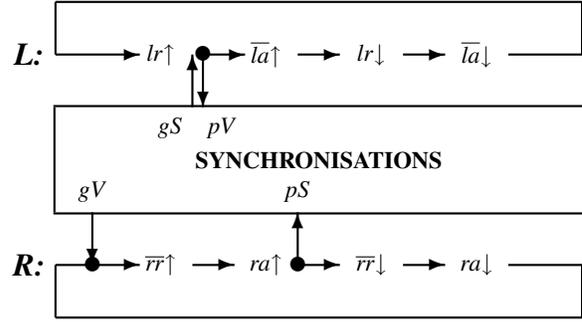

Figure 3. Constraints on LC_{max}

These two synchronisations may be modeled in several ways. One reusable style[1] to model (see Figure 3) is define two tokens one for each of the synchronisations:

1. ■ by S, the space token $\qquad\qquad S = \overline{gS}.\overline{pS}.S$
2. ● by V, the value token, $\qquad\qquad V = \overline{gV}.\overline{pV}.V$

each of which is taken (by a **g**et handshake, gV or gS) and replaced (by a **p**ut handshake, pV or pS). Importantly, puts never delay the sender; but gets will block a requester until permission is granted. This leads to the final form of our specification:

$$
\begin{aligned}
L &= lr\uparrow.gS.pV.\overline{la}\uparrow.lr\downarrow.\overline{la}\downarrow.L \\
R &= gV.\overline{rr}\uparrow.ra\uparrow.pS.\overline{rr}\downarrow.\overline{ra}\downarrow.R \\
S &= \overline{gS}.\overline{pS}.S \qquad\qquad V = \overline{pV}.\overline{gV}.V \\
LC_{max} &= (\ L \mid S \mid V \mid R \ ) \setminus \{gV,pV,gS,pS\}
\end{aligned}
$$

The last line of the specification defines the behaviour of LC_{max} as the composition of the upstream channel process L; the downstream channel process R; and the synchronisation between the two channels with space and value tokens S and V. The handshakes between L, R and S and V are made private (hidden) with $\setminus \{gV,pV,gS,pS\}$ so that no other process can tamper with them.

The only synchronising constraint on the input channel of LC_{max} is that L must wait until a slot is free (gS) to accept the value on dIN; and the only constraint on the output channel is that fresh data must be latched (gV)

---

[1]In particular it handles the shuffles alluded to above and implementing the cut-aways of Section 3.
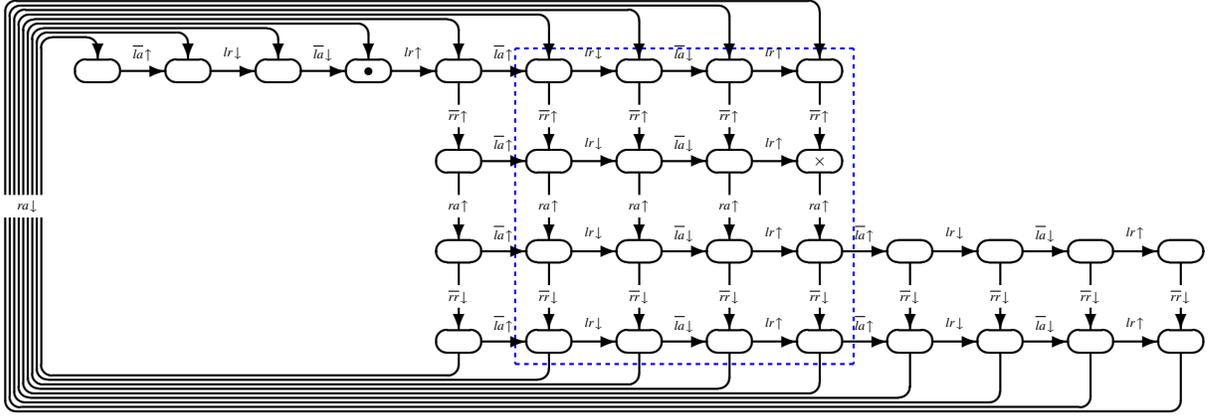
Figure 4. (Minimised) states of the LC_{max} latch protocol

before the $\overline{rr}\uparrow$ signal can be sent downstream. Both L and R are allowed to proceed at their earliest possible opportunity when blocked.

> *LC_{max} as defined is the most concurrent protocol possible for a latch protocol where data is valid before the rising request on the left channel.*

This protocol has 32 states, and d-deep pipelines and parallel pipelines have $16d + 16$ states. Figure 4 depicts the minimised state graph of LC_{max}. A middle 4×4 block of states can be iterated for deeper pipelines to give rise to minimised versions of LP_d. Runs on the CWB confirm that $PP_{w,d} \equiv LP_d$ for $w, d = 1..8$. Thus LC_{max} exhibits stable behaviour. By inductive argument, we can reason about the overall control signal behaviours of structured widening and thinning parallel pipelines as though they were linear pipelines of the same depth — a much simpler model to grasp.

3. The Family Derived from LC_{max}

The possible design space for 4-phase protocols is bounded above by LC_{max} which exhibits the largest possible parallelism. A formal method is developed to derive all less concurrent protocols from LC_{max}. This is achieved by creating and applying rules which systematically reduce concurrency by minimal increments. The behaviour of all protocols when pipelined is then tabulated.

A convenient, more a compact notation of the minimised state graph of the most concurrent protocol LC_{max} of Figure 4 has been developed. Since all the transitions follow simple patterns, we choose to present our ideas using what we call a **shape**:

```
R1:   o  o  o  +  o  o  o  o  o
R2:            o  o  o  o  o
R3:            o  o  o  o  o  o  o  o  o
R4:            o  o  o  o  o  o  o  o  o
```
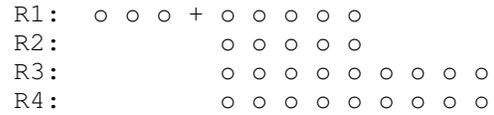
The initial state is marked '**+**'; other reachable states by '**o**', and unreachable states by '**.**'. Each shape is a graphical representation of a specific handshake protocol, fully specifies its behaviour, and differentiates it from all other protocols. The graphical representation provides intuition about the concurrency and specific behavioural and pipeline properties of all protocols.

3.1. Concurrency Reduction Rules

The rules for generating members of the untimed[2] 4-phase family are:

1. The initial idle state must be reachable from all states in the graph. This has the following consequences:

   (a) This will restrict the number of states that can systematically be removed from the "left" and "right" side of the state graph. For example, the following is the maximum left cut-away that preserves reachability of the initial state:

   ```
   R1:   .  .  .  +  o  o  o  o  o
   R2:            .  .  .  o  o
   R3:            .  .  .  o  o  o  o  o  o
   R4:            .  .  .  o  o  o  o  o  o
   ```

---
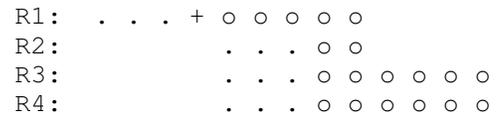
[2]Rules 3 and 4 may change for other timing disciplines such as burst-mode and relative timing, but the overall approach remains the same.

(b) Each row in the graph must contain at least one state, otherwise the graph will deadlock (represented as $\mathcal{D}$).

2. Internal holes in the state space are disallowed. Thus the following state graph is deemed illegal:

```
R1:   o   o   o   +   o   o   o   o   o
R2:                   o   .   o   o   o
R3:                   o   .   o   o   o   o   o   o   o
R4:                   o   o   o   o   o   o   o   o   o
```

Such graphs are found to generate very irregular behaviour when pipelined. This rule cuts the search space from over 400,000 protocols to 250.

(a) Disallowing holes in shapes has the consequence that we can generate all possible sub-behaviours by listing all viable ways of cutting states away on the left; similarly on the right; and then mechanically generating all combinations of cut-aways.

3. In untimed protocols, inputs *lr* and *ra* must always be accepted.

4. The protocol can restrict when outputs $\overline{rr}$ and $\overline{la}$ are possible.

(a) The Speed-independent set of protocols is a concurrency reduction of the delay-insensitive set after employing *output ordering*.

3.2. The Cut-Away Notation

The following notation is adopted for cut-aways:

1. *Labcd* means from LC_{max} remove the leftmost a live states (circles) from R1; the leftmost b live states (circles) from R2; etc. Thus cut-away *L2112* from LC_{max} results in the shape:

```
R1:   .   .   o   +   o   o   o   o
R2:           .   o   o   o   o
R3:           .   o   o   o   o   o   o   o
R4:           .   .   o   o   o   o   o   o
```

in which each cut-away state is denoted by '.'. Since this shape has 7 reachable states in row 1, 4 in row 2, 8 in row 3, and 7 in row 4, we use the short hand 7487 where it suits (the notation is occasionally ambiguous, whereas the cut-away notation is not).

2. Similarly *Rabcd* cuts away from the right hand end of LC_{max}. Cut-away *R2222* on LC_{max} results in the following shape or shorthand 7377:

```
R1:   o   o   o   +   o   o   .   .
R2:                   o   o   o   .   .
R3:                   o   o   o   o   o   o   o   .   .
R4:                   o   o   o   o   o   o   o   .   .
```

3. Applying both the cut-aways $L_{2112} \circ R_{2222}$ to LC_{max} returns the shape 5265:

```
R1:   .   .   o   +   o   o   o   .   .
R2:                   .   o   o   .   .
R3:                   .   o   o   o   o   o   o   .   .
R4:                   .   .   o   o   o   o   .   .
```

The following cut-away patterns emerge from the rules:

1. **LEFT**: L0000, L1001, L1111, L2002, L2112, L3003, L3113, L2222, L3223, L3333.

There are 10 in all. Any cut-aways of depth 4 would make the initial state an orphan and are rejected. Cut-aways consisting entirely of even numbers (L0000, L2002, L2222) are of the *delay-insensitive* (DI) class. The set with odd numbers (L1001, etc.) are the *speed-independent* class and employ output ordering.

2. **RIGHT**: R0000, R0020, R0040, R0022, R0042, R2022, R2042, R2222, R2242, R2262, R0044, R2044, R4044, R2244, R2264, R4244, 4264.

There are 25 in all but after experimentation only the 17 listed here turn out to yield protocols that implement pipelining. The *delay-insensitive* class of right cut-aways exist when both the first two numbers agree and the last two numbers agree (R0000, R0022, R2222, R0044, R2244). The others are of the *speed-independent* class.

3.3. Protocol Categories

These cut-aways allow us to classify pipeline protocols into three families. The delay-insensitive family consists of both left and right DI cut-aways. The speed-independent family consists of protocols where the left or right cut-away employs output ordering. The timed family (not included in this paper) consist of cut-aways that restrict the arrival of inputs *lr* or *ra* based on local timing assumptions.

We have mechanised the task of generating all possible delay-insensitive and speed-independent pipelined protocols. All 250 have been evaluated on the CWB by running them in linear pipelines of depth 1..8 and parallel pipelines of depth 1..8 and width 1..8.

When the 250 protocols were examined, 6 categories emerged:

1. **deadlock**: The protocol deadlocks because the L and R cut-aways meet or overlap.   **92**

2. **constant**: Protocols that only hold one data item per linear pipeline.   **21**

3. **O(8)**: A special class of protocols that can only hold a data item in every other pipeline stage. Their state sizes increase by 8 not 16 as the pipelined grow deeper. These are only found when applying the R2244, R2264, R4244 and R4264 right cut-aways. Notice that 4 of these shapes are stable even though O(8).   **22**

4. **semi-regular O(16)**: These protocols do not maintain their native shape when composed into either linear pipelines LP_d or parallel pipelines $PP_{w,d}$. Some of the concurrency removed in the protocols is regained in their parallel compositions.   **43**

5. **regular O(16)**: These protocols retain their shapes predictably when composed in a linear pipeline LP_d, and increment by 16 states with each increase of pipeline depth. However, parallel pipelines $PP_{w,d}$ do not maintain their native shape.   **60**

6. **stable O(16)**: These protocols retain their shapes in linear and parallel pipelines of all depths. This only occurs for delay-insensitive protocols.   **12**

The category of constant protocols are all concurrency reduced versions of the DI protocol $L_{0000} \circ R_{2266}$. This consists of cutting off the right six columns of the LC_{max} shape of Figure 4. Thus, at least one of the states in R2266 are required for pipelining.

Certain protocols can only store data in *every other* latch when the pipeline is stalled[3], called half-buffering [8]. Any protocol that does *not* contain the state marked with $\times$ in Figure 4 (or that remove *any* states in R2) cannot store data in every latch when stalled. Thus, we define this state as the **pipeline state**. The O(8) category is a subset of this set since these states only occur with R2244, R2264,R4244 and R4264 cut-aways. However, note that even certain delay-insensitive protocols, such as $L_{0000} \circ R_{2222}$, cannot store data in all latches. Protocols that do not include the pipeline state are not useful for certain implementations such as FIFOs.

Protocols in the stable category retain their native shapes when composed in parallel. Further, the linear and parallel pipelines are equivalent: $LP_d \equiv PP_{w,d}$ $\forall w, d > 0$. This means that a linear portion of such a pipeline may be replaced by a parallel pipe of the same length; and vice versa. Thus such structured pipelines may be thinned or fattened with no effect visible to the external observer of their control signals. This is a very useful guarantee and a handy simplification when

---

[3]Assuming pulse latch clocking is not employed.

reasoning about parallel pipelines. Stable protocols can only occur when both the left and right cut-aways are delay-insensitive. For each additional parallel stage, 16 states are added. Thus, the state space of protocol $L_{0000} \circ R_{0000}$ grows as 32, 48, 64, 80, ... as pipeline depth increases. The 16 additional states per pipeline stage correspond to the 4×4 block in the hashed box of Figure 4. Therefore, the native interface protocol of a d-deep pipeline is the resultant shape calculated by removing the $d \times 16$ states in the center of the shape.

The regular and semi-regular protocols behave regularly for $d = 2, 3, 4 \ldots$. The shape for $d > 1$ is not equivalent to the native shape for the protocol. All shapes consisting of two or more stages in parallel ($d \geq 2$) converge on a specific concurrent protocol with more concurrency. Thereafter the behaviour maintains the same native protocol shape. Much of the concurrency that is regained in these categories is the return of concurrency that was removed through output ordering.

For example, consider the semi-regular speed-independent protocol $L_{1001} \circ R_{0000}$. It contains 30 states and implements output ordering where $\overline{la}\uparrow$ precedes $\overline{rr}\downarrow$. The protocol interface becomes identical to LC_{max} when composed in linear pipelines of depth 2 or more. However, it is identical to LC_{max} in *all* parallel pipelines. This is shown in the following table that gives the number of states in various parallel configurations:

	$d = 1$	$d = 2$	$d = 3$	$d = 4$
LP_d	30	48	64	80
$PP_{w,d}$	32	48	64	80

This shows that some of the concurrency removed from a protocol is recovered in a regular way when protocols are placed in parallel configurations.

Thus a protocol may behave identically to a more concurrent protocol when placed in parallel configurations. This implies that protocol equivalence classes could emerge, as is shown to be true in Sections 4.1 and 4.2. This also implies that inside a protocol equivalence class, certain concurrency reductions might result in more efficient implementations than others. Our results in this area will be reported in future publications.

The family of untimed protocols is rather large. Removing the deadlock and constant categories as being uninteresting for implementing pipelines leaves a family of 137 distinct and useful protocols. This family is tabulated in Table 1. Only categories 3–6 are recorded for brevity. The 12 stable shapes are represented as category 6, the 60 regular shapes with 5, the 43 semi-regular shapes with 4, and the 22 O(8) shapes with 3. Deadlocking states, such as $L_{3333} \circ R_{4044}$, are marked $\mathcal{D}$.

L0000	L1001	L1111	L2002	L2112	L3003	L3113	L2222	L3223	L3333	L ∘ R
6	4	5	6	5	5	5	6	4	5	R0000
4	4	4	4	4	4	4	4	5	4	R0020
5	4	5	5	5	5	5	5	4	5	R0040
6	4	5	6	5	5	5	6	4	5	R0022
5	4	5	5	5	5	5	5	4	5	R0042
5	4	5	5	5	5	5	5	4	$\mathcal{D}$	R2022
5	4	5	5	5	5	5	5	4	$\mathcal{D}$	R2042
6	4	5	6	5	5	5	6	4	$\mathcal{D}$	R2222
4	5	4	4	4	4	4	4	4	$\mathcal{D}$	R2242
5	4	5	5	5	$\mathcal{D}$	$\mathcal{D}$	5	$\mathcal{D}$	$\mathcal{D}$	R2262
6	4	5	6	5	5	5	6	4	5	R0044
4	5	4	4	4	4	4	4	4	$\mathcal{D}$	R2044
5	4	$\mathcal{D}$	5	$\mathcal{D}$	5	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4044
3	3	3	3	3	3	3	3	3	$\mathcal{D}$	R2244
3	3	3	3	3	$\mathcal{D}$	$\mathcal{D}$	3	$\mathcal{D}$	$\mathcal{D}$	R2264
3	3	$\mathcal{D}$	3	$\mathcal{D}$	3	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4244
3	3	$\mathcal{D}$	3	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4264

Table 1. Categorisation of the family of 4-phase latches

3.4. Additional Properties

Additional important distinguishing properties of the protocols can be graphically represented on Figure 4 and using the cut-away notation:

- Only protocols that contain the state marked with × in Figure 4 will latch data in every pipeline stage when using 4-cycle protocols. The two-phase and O(8) protocols can latch every stage if using a pulsed clock or handshaking through the register.

- The states in which the latch must be transparent and opaque can be represented by a coloring. Based on these colorings, it can easily be shown that certain states require a state variable to control the latch due to the state spaces. Some protocols don't cover the states that require a state marking, and thus result in simpler latch control logic that can be encoded directly from a combinational function of the handshake signals, and even the $\overline{rr}$ and $\overline{la}$ signals.

4. Parallel Protocol Equivalence Classes

Huygens invented the pendulum in 1658. In 1665 he noticed that if he put two of his clocks side by side then their pendulums would always synchronise within 30 minutes whatever their out-of-phase initial settings. We have an analogous convergence between different protocols when placed in parallel configurations.

The parallel behaviour of the family of protocols configured in parallel pipelines is represented in Table 2. Linear pipelines are presented in Table 3. These tables are divided by three vertical and five horizontal blocks. The top left of each block is a stable state that is the result of composing two delay-insensitive cut-aways. In Table 1 no particular pattern emerges if we examine by rows or by columns; there is no predictable pattern of row or columns of just 4's or 5's. This indicates that neither the L or R cut-aways are a dominant factor in the pipelined behaviour of our protocols.

Examining block-by-block, the best behaved is the center block with stable shape $L_{2002} \circ R_{0022}$. This shape is very symmetric in its left and right cut-aways, as a pair of $\overline{rr}\downarrow$ transitions are pruned by the left cut-away and a pair of $\overline{la}\downarrow$ transitions by the right cut-away.

4.1. Parallel Pipelines

Table 2 displays the behaviours of the parallel pipelines $PP_{w,d}$. Models were run for $w = 1..8$ and $d = 1..8$ for all 137 category 3–6 protocols.

The first interesting fact to emerge is that $PP_{w,d} \equiv PP_{1,d}$ for $w = 2, 3, .., 8$. Therefore when reasoning

L0000	L1001	L1111	L2002	L2112	L3003	L3113	L2222	L3223	L3333	L o R
9599	9599	9599	7597	9597	9597	9597	7377	7377	7377	R0000
9599	9599	9599	9597	9597	9597	9597	7377	7377	7377	R0020
9599	9599	9599	9597	9597	9597	9597	7377	7377	7377	R0040
9577	9577	9577	7575	7575	7575	7575	7355	7355	7355	R0022
9577	9577	9577	7575	7575	7575	7575	7355	7355	7355	R0042
9577	9577	9577	7575	7575	7575	7575	7355	7355	$\mathcal{D}$	R2022
9577	9577	9577	7575	7575	7575	7575	7355	7355	$\mathcal{D}$	R2042
7377	7377	7377	5375	5375	5375	5375	5155	5155	$\mathcal{D}$	R2222
7377	7377	7377	5375	5375	5375	5375	5155	5155	$\mathcal{D}$	R2242
7377	7377	7377	5375	5375	$\mathcal{D}$	$\mathcal{D}$	5155	$\mathcal{D}$	$\mathcal{D}$	R2262
9555	9555	9555	7553	7553	7553	7553	7333	7333	7333	R0044
9555	9555	9555	7553	7553	7553	7553	7333	7333	$\mathcal{D}$	R2044
9555	9555	$\mathcal{D}$	7553	$\mathcal{D}$	7553	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4044
7355	7355	7355	5353	5353	5353	5353	5133	5133	$\mathcal{D}$	R2244
7355	7355	7355	5353	5353	5353	5353	5133	$\mathcal{D}$	$\mathcal{D}$	R2264
7355	7355	$\mathcal{D}$	5353	$\mathcal{D}$	5353	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4244
7355	7355	$\mathcal{D}$	5353	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4264

Table 2. Parallel Pipeline Protocols $PP_{w,d}$

about structured parallel pipelines, one can always use the simpler representation $PP_{1,d}$.

The second interesting fact is that within each of the 15 blocks in Table 2, all structured parallel pipelines result in the equivalent behaviour of the most parallel shape. Thus in a parallel pipeline, if a less concurrent protocol is implemented, it is indistinguishable from the most parallel delay-insensitive protocol. This implies that any of the protocols that apply concurrency reduction might result in a more efficient implementation that results in the same delay insensitive behaviour.

4.2. Linear Pipelines

LP_d were evaluated for $d = 1..8$ over all 137 category 3–6 protocols. All single pipeline protocols showed predictable growth and shape for pipelines of depth 2 and deeper. Thus Table 3 shows state sizes for depth 2, and group together equivalent protocols.

Three different equivalence sets emerge:

1. There are four 2×2 groups (in red) of adjacent cut-aways which have identical protocols for LP_d where $d \geq 2$. In each of the four cases, these protocols converge to the most parallel protocol, that in the top left position of the group. These sets consist of the four shapes that converge to protocols $L_{0000}{\circ}R_{0000}$, $L_{0000}{\circ}R_{2242}$, and $L_{0000}{\circ}R_{2044}$ in the first column and $L_{3223}{\circ}R_{0000}$ in the ninth.

2. There are 12 vertically arranged pairs of shapes (in blue) that exhibit unique *LC* behaviours but are equivalent when pipelined at depths 2 or greater. In each case they converge to the most state rich shape, the higher of the two. These pairs consist of the protocols in the first and second rows, ninth and tenth rows, and 12th and 13th rows in columns three through eight. Notice, for example, that in rows one and two of Table 3 there are two distinct pairings of 44 states, 42 states, and 40 states. All other equivalent state pairs do not have equivalent shapes. For example, even though both $L_{2002}{\circ}R_{0042}$ and $L_{2002}{\circ}R_{2022}$ have 38 states, they do not have equivalent shapes.

3. There are 13 horizontally arranged pairs of shapes (in green) that result in identical protocols. These are the pair with 24 states in the last row, 26 states in rows 15 and 16, 28 states in row 14, 30 states in row five, those with 32 states in rows three and four, 40 states in rows seven, eight, and eleven, 42 states in rows five and six, and 44 states in row four. All other protocols are unique, even when they consist of the same number of states. For example, protocols $L_{0000}{\circ}R_{0040}$ and $L_{1001}{\circ}R_{0040}$ both have 44 states but they are different protocols.

L0000	L1001	L1111	L2002	L2112	L3003	L3113	L2222	L3223	L3333	L o R
48	48	44	44	42	42	40	40	36	36	R0000
48	48	44	44	42	42	40	40	36	36	R0020
44	44	40	40	38	38	36	36	32	32	R0040
44	44	40	40	38	38	36	36	32	32	R0022
42	42	38	38	36	36	34	34	30	30	R0042
42	42	38	38	36	36	34	34	30	$\mathcal{D}$	R2022
40	40	36	36	34	34	32	32	28	$\mathcal{D}$	R2042
40	40	36	36	34	34	32	32	28	$\mathcal{D}$	R2222
36	36	32	32	30	30	28	28	24	$\mathcal{D}$	R2242
36	36	32	32	30	$\mathcal{D}$	$\mathcal{D}$	28	$\mathcal{D}$	$\mathcal{D}$	R2262
40	40	36	36	34	34	32	32	28	28	R0044
36	36	32	32	30	30	28	28	24	$\mathcal{D}$	R2044
36	36	$\mathcal{D}$	32	$\mathcal{D}$	30	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4044
28	28	24	24	22	22	20	20	16	$\mathcal{D}$	R2244
26	26	22	22	20	$\mathcal{D}$	$\mathcal{D}$	18	$\mathcal{D}$	$\mathcal{D}$	R2264
26	26	$\mathcal{D}$	22	$\mathcal{D}$	20	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4244
24	24	$\mathcal{D}$	$\mathcal{D}$	20	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{D}$	R4264

Table 3. Linear Pipeline Protocols LP_2

5. The Family Hierarchy

The cut-away representation $L{\circ}R$ of the protocol family provides a direct method of ordering the entire family into a lattice based on protocol concurrency. The protocols are ordered based on state richness: protocol $X \leq$ protocol Y iff every state in shape X is also a state in shape Y. The easiest way of carrying this out is simply to compare the cut-away definitions of X and Y.

Let $L_{abcd} \leq L_{a'b'c'd'}$ iff $a \geq a$' **and** $b \geq b$' **and** $c \geq c$' **and** $d \geq d$'. That is L_{abcd} cuts away more or the same as $L_{a'b'c'd'}$ for each row of a shape.

Similarly for the class of right cut-aways. Then protocol $L_{abcd}{\circ}R_{efgh}$ is a proper sub-protocol of shape $L_{a'b'c'd'}{\circ}R_{e'f'g'h'}$ iff $L_{abcd} \leq L_{a'b'c'd'}$ and $R_{efgh} \leq R_{e'f'g'h'}$. Otherwise they are not comparable.

The process is very simple to mechanise without the need to generate and compare the minimised state graph shapes.

The 15 combinations of delay-insensitive cut-away classes that produce stable shapes are displayed in a lattice in Figure 5. A shorthand notation is used in the lattice to represent the protocols by listing the number of states in each row of the shape. The top of the lattice is 9599 (LC_{max}) with 32 states, and the least concurrent stable protocol is 5133 with 12 states. Notice, however, that this notation is not unique as two different protocols in the lattice share the shorthand notation of 7377 and

7355. The unambiguous $L{\circ}R$ notation can be derived from the figure to identify the protocol shape.

6. Related Work

Asynchronous designers are well aware of concurrency reduction as a means of modifying protocols to generate more efficient implementations. Some concurrency reduction algorithms have been automated and implemented in CAD tools [2]. The formalisation of a set of concurrency reducing transformations and rules have been previously published. Lines started with a concurrent handshake expansion in CSP, and then applied four *reshuffling* rules to the handshake signals to reduce concurrency [8]. This produced nine valid protocols, eight being reshufflings of the most concurrent MSFB protocol. McGee and Nowick developed a graphical framework based on signal transition graphs [9]. They formalised three correct-by-construction arc transformation constraints to reduce concurrency, and produced a lattice of protocols.

One significant difference to previous work is the completeness and coverage of the protocol space. The previous work implements subsets of the work presented here. Our formal process based transformations are complete and exhaustive. All protocols, starting with the most concurrent LC_{max}, are part of our set. The

L0000   L2002   L2222   L o R

| 9599 | 7597 | 7377 | R0000 |
9577	7575	7355	R0022
7377	5375	5155	R2222
9555	7553	7333	R0044
7355	5353	5133	R2244
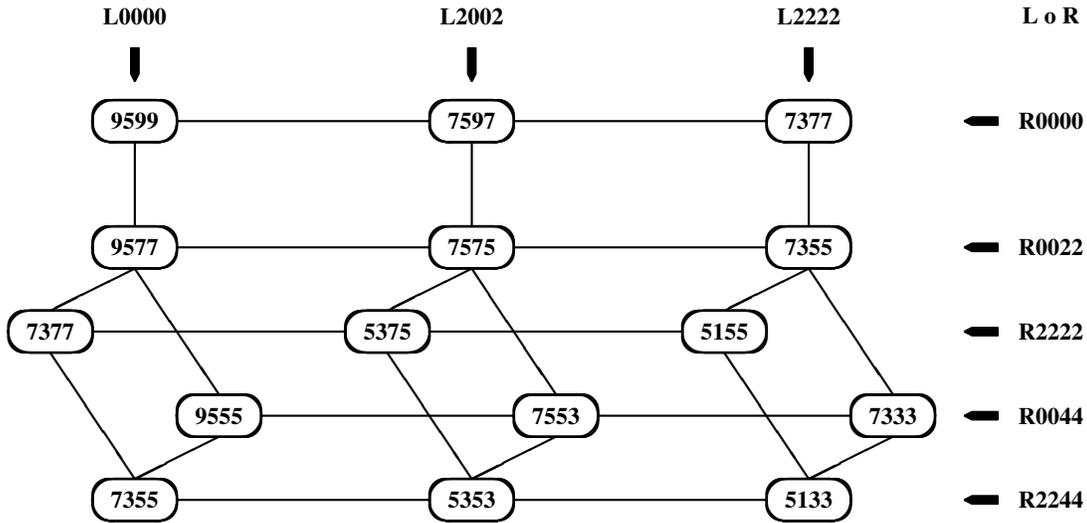
Figure 5. Lattice of Stable Protocols

most concurrent protocol in these publications is in the $L_{0000} \circ R_{0044}$ protocol equivalence class. This covers only the bottom six protocol equivalence classes in our lattice; the nine more concurrent protocol equivalence classes are not included. Additionally, our work is completely general. We don't impose any constraints on the implementation, and even abstract out the latch control signals. McGee's work focused on characterising a particular implementation style based on dynamic gates and relied upon internal signals such as reset, precharge, and evaluate for their model.

This work also derives many characteristics of pipelined protocols that were previously unknown or not clarified elsewhere. For example, Lines characterises protocols in terms of their ability to store data in each latch; the half buffered protocols (such as PCHB) can only store data in every other latch whereas the fully buffered protocols (such as PCFB) store data in all latches upon a pipeline stall [8]. However, no specific property was defined that results in this characteristic behaviour. Section 3.3 defines this property as being directly dependent on the *pipeline state* in row R2 of right cut-aways. The PCFB protocol $L_{1001} \circ R_{4044}$ is fully buffered since no states are removed in R2 of its cut-away R4004; the PCHB $L_{1001} \circ R_{4264}$ is half buffered because two states are removed from row two of its right cut-away. Given the pipeline state property we have defined one can observe the shape of any protocol and immediately determine if the protocol is half or fully buffered. Thus one can quickly prove that Sutherland's Micropipeline [13] is a half buffered protocol, and should not be used in a FIFO. Many other protocols and properties not previously known are presented here, such as the 15 equivalence classes that result when protocols are placed in parallel configurations.

7. Published Circuits

A selection of published circuits have been examined as shown in Table 4. Of the 28 listed there are only 16 distinct protocols implemented.

In the protocol family investigated in this paper, all but the control handshake signals $lr, \overline{la}$ and $\overline{rr}, ra$ are formally hidden from the protocol behaviour. This work does not consider power, area, speed, or whether the latch is normally open or closed. Thus each protocol has a multitude of possible implementations. What the protocol does tell you is how every corresponding implementation will behave at the interface when composed together in a single or parallel pipeline. These circuits can also be placed into the lattice and tables to determine properties of the protocol and study alternate implementations which may be improvements over the current version.

8. Contributions

In this paper we have presented the family of 4-phase latch protocols with data valid before rising request:

Name	Protocol	Reference
MSFB	$L_{0000} \circ R_{4044}$	[8]
KG	$L_{1001} \circ R_{0000}$	[7]
FD6	$L_{1001} \circ R_{0040}$	[5]
FD7	$L_{1001} \circ R_{0044}$	[5]
PCFB	$L_{1001} \circ R_{4044}$	[8]
PCHB	$L_{1001} \circ R_{4264}$	[8]
FL2,FL3	$\mathbf{L_{2002} \circ R_{0000}}$	[6]
BNC1	$L_{2002} \circ R_{0042}$	[4]
BNO1,EGc,EGd,FL1	$\mathbf{L_{2002} \circ R_{0044}}$	[3, 6]
BNO2,EGa,EGb,BNC2,LH2	$L_{2002} \circ R_{2042}$	[3, 4]
BRF1,LH1	$L_{2002} \circ R_{2044}$	[4]
MP	$L_{2002} \circ R_{2244}$	[13]
FD4,WCHB	$L_{2002} \circ R_{4264}$	[5, 8]
BAF1	$L_{2112} \circ R_{2042}$	[4]
FD5,ERS1,ERT1	$L_{2222} \circ R_{2022}$	[4, 5]
YBA	$\mathbf{L_{2222} \circ R_{2222}}$	[14]

Table 4. Published circuits, (stable in bold)

their control signal properties and behaviours, and how they compose into homogeneous structured linear and parallel pipelines.

We have fully specified *every* protocol that exists in the family of 4-phase pipeline controllers where data is valid before the rising edge of request. The most concurrent protocol LC_{max} is specified, from which all less concurrent untimed protocols are derived.

A canonical state graph representation for protocols is presented and called a **shape**. This easily allows us to demonstrate properties of handshake protocols and the result of formal concurrency reduction transformations.

The behaviour of all 250 possible protocols was characterised in linear and parallel pipelines. Six fundamentally different categories emerged. We labeled these as stable (12 of O(16)), regular (60), semi-regular (43), regular 2-phase (22) of which 4 are stable, constant (22), and deadlock (91). Stable behaviours have shapes that are not modified in linear and parallel configurations. This set has an interesting property of defining protocol equivalence classes as noted below, and these protocols were used to define the protocol lattice. Regular protocols are not modified when placed in linear pipelines, but their behaviour is more concurrent when placed in parallel pipelines. Semi-regular protocols exhibit increased concurrency in both linear and parallel pipelines. For all protocols, their maximum concurrency is reached after only two pipeline stages.

Additional properties are derived and mapped to our protocol shapes. We defined the condition that must hold for a controller to be pipelined. This condition is dependent on right cut-away R2266 which overly restricts responses on the upstream channel.

While the interaction between the protocol and the latches was not explicitly modeled, two additional key properties were defined in this work that relate to the latching behaviour of the protocol.

First, an important pipeline property in the presence of stalls is the ability to store data in every latch. This work defined one specific state, the *pipeline state*, that must exist in any 4-phase protocol in this family to allow it to store data in all latches when stalled. Thus any fully-buffered protocol will contain this state in the shape, whereas half-buffered protocols will not.

Second, this work classifies protocols into two sets: those that require a state variable to control the latch and those that can control the latch using a function on the input and output signals of the gate (possibly using only one handshake signal). This can be a complexity parameter for implementations, as well as provide a reduction in protocol delays or timing requirements in circuit implementations. A coloring on the shape can be derived indicating the states that require an additional latch control state variable for either normally open or normally closed control. Details of these colorings are not presented here due to space limitations.

The protocol shapes were placed into equivalence classes based on the protocol behaviour presented at the interfaces. We found that for parallel pipelines there are 15 equivalence classes of up to 16 different protocols, each dominated by one of the stable protocols. Thus stable protocols have a central role in pipelining. Since linear, independent latches rarely occur, designs that use concurrency reduction techniques to improve performance and power, yet map to a pipeline equivalence class might result in very productive optimisation techniques. Our results in this area will be presented later.

Linear pipelines were also evaluated and placed into equivalence classes. These configurations showed a much finer granularity in equivalence classes, as the largest sets contained only four protocols.

A definition for categorising protocols into a lattice was defined, and the 15 parallel protocol equivalence classes were placed into a lattice. The lattice, nomenclature, and shape models presented in this paper provide several different methods to compare and contrast protocols and their realisations as circuits. The unique textual representation of the protocols encodes restrictions on the left and right channels and also encodes timing assumptions built in the circuit including delay-insensitive and speed-independent protocols, those with output ordering, and protocols that have inherent timing in the protocol. The stable protocols, which serve as basins of attraction to the other protocols, is also derived from this naming convention.

A large set of published circuits were then mapped to our protocol family. These circuits include designs using combinational logic, dynamic logic, and C-elements.

The evaluation of tradeoffs between concurrency reduction, energy, and performance across an entire protocol family can now be made. This tradeoff largely occurs due to circuit improvements based on circuit timing, such as output ordering, against the reduced system level concurrency that occurs based on the concurrency reduction. The small number of stable configurations (15) that serve as basins of attraction allow the choice of fixed design zones. There are also subsets of the family that present particularly interesting tradeoffs. Namely, all R0044 and larger cut sets retain full forward concurrency but result in substantially simplified protocols by removing concurrency when recovering from a stall. From a system level perspective this type of concurrency reduction can be extremely beneficial especially if stalls are rare, such as in a data-path. However, this optimisation may perhaps not provide the best protocol when designing FIFO buffers.

The completeness of this work provides information to help designers build circuits that meet their power, performance, and storage needs. This also provides a uniform representation for comparing various implementations of equivalent and similar protocols. This work defines the protocol used for current published circuit implementations.

There is still much to be done in furthering the understanding of asynchronous handshake protocols. Space precludes us from mentioning work completed or underway on mathematical proofs of our results and mathematical transformations that result in the cut-aways, 2-phase latch controllers, rules for timed protocols such as burst-mode and relative timed, and the efficiency of circuits synthesized for a variety of protocols for which there are no known published implementations.

References

[1] G. Birtwistle. Control states in asynchronous pipelines. In A. Yakovlev and R. Nouta, editors, *Asynchronous Interfaces: Tools, Techniques, and Implementations"*, pages 45–55, July 2000.

[2] J. Cortadella, M. Kishinevsky, S. M. Burns, A. Kondratyev, L. Lavagno, K. S. Stevens, A. Taubin, and A. Yakovlev. Lazy transition systems and asynchronous circuit synthesis with relative timing assumptions. *IEEE Transactions on Computer-Aided Design*, 21(2):109–130, Feb 2002.

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

[4] S. 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.

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

[6] 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.

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

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

[9] P. B. McGee and S. 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.

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

[11] F. G. Moller and P. Stevens. *The Edinburgh Concurrency Workbench (Version 7)*. University of Edinburgh, October 1992.

[12] K. S. Stevens. *Practical Verification and Synthesis of Low Latency Asynchronous Systems*. PhD thesis, University of Calgary, Calgary, Alberta, September 1994.

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

[14] K. Y. Yun, P. A. Beerel, and J. 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.