A Doubly-Latched Asynchronous Pipeline

Rakefet Kol and Ran Ginosar
VLSI Systems Research Center, Electrical Engineering Department
Technion - Israel Institute of Technology, Haifa 32000, Israel
rakefet@techunix.technion.ac.il

Abstract

DLAP, an asynchronous pipeline with master-slave (dual) registers, offers improved performance. It is most suitable for converting synchronous circuits into asynchronous ones. DLAP is capable of truly decoupled operation: All pipeline stages can shift data simultaneously, and execution is faster than previous designs when variable delays are encountered. Implementations based on both edge triggered registers and transparent latches are shown. STG and verified controllers are presented and simulated.

1. Introduction

Asynchronous logic does not use a clock. Logic elements are *self-timed*: they detect and announce when the computation is complete and the outputs are ready [6, 11]. They also wait for inputs to be announced before starting the computation. Registers load their inputs under local control, rather than on a global clock edge. Asynchronous logic trades time for discrete events. Actual delays are hidden (abstracted), and only sequences of events (as depicted by transitions) matter. Thus, the correctness of computation is made independent of delays. Another advantage is that events can be treated hierarchically and local details can be abstracted, similar to hierarchical logic design, whereas the design of continuous timing is global and 'flat'. Instead of (metric) timing waveforms, discrete event precedence graphs, such as STG [1], are employed to describe circuit operation.

Asynchronous circuits are expected to achieve lower power consumption and/or higher performance, by eliminating the driving clock. In addition, computational delays can be data dependent. While synchronous pipelines are timed according to the worst case delay over all stages, asynchronous logic can be designed to determine and signal its own completion time, typically saving time and power. On average, self-timed operation with completion detection results in about 2× speedup of the individual units; power savings depend on the particular application, but can reach as high as 80% [14].

We have developed algorithms for converting synchronous circuits into asynchronous ones, thus exploiting some advantages of asynchronous circuits while retaining investments in synchronous designs and tools [7]. We consider synchronous logic specified in VHDL. The VHDL code is synthesized into netlists according to the common architecture of 'register-and-cloud' pipelines [10], where 'clouds' of combinational logic are separated by clocked registers. We would like to take advantage of the general pipeline structure and of the combinational logic clouds, but we need to get rid of the clocked registers, thus converting a synchronous circuit into an asynchronous one. To that end, we must identify the best target asynchronous pipeline.

Asynchronous micropipelines were first introduced in [13]. They were based on a 2-phase communication protocol. Four-phase handshake protocol pipelines are presented in [8, 9], where edge triggered registers are employed. Various similar control structures were proposed in order to enhance the performance of the asynchronous pipeline [3, 4, 5, 15], based on either edge triggered registers or level sensitive latches. However, all those asynchronous pipeline designs suffer from one of the following drawbacks: They either achieve only 50% utilization of the pipeline stages (i.e., only every other stage is active at any one time), or (in some cases) incur a long backwards propagation of the acknowledge signals. The backwards latching scheme cannot be avoided when only a single register is used in each stage, since a storage element cannot release its value until the following stage has signaled that it is ready for another value. This might result in a major performance problem for deeply pipelined circuits, e.g., rings or linear pipes with data dependent stage delays. In [12] it was shown that pipeline performance depends on the number of bubbles, namely registers ready to accept new values. When only a single bubble exists in the above mentioned designs, their performance is limited by the lack of bubbles. Synchronous pipelines, on the other hand, are not limited: If each register is master-slave, then a bubble is always available, and all values can propagate simultaneously.

In this paper we present the Doubly-Latched

Asynchronous Pipeline (DLAP), which operates similarly to synchronous pipelines. We show that not only is it most suitable for synchronous-to-asynchronous conversion, but in certain important cases it outperforms previous designs. Section 2 describes the Doubly Latched Asynchronous Pipeline. Edge triggered and latched DLAP are discussed in Sections 3 and 4, respectively. A comparative analysis is presented is Section 5, and Section 6 discusses non-linear DLAPs.

2. The Doubly-Latched Asynchronous Pipeline

DLAP (Doubly-Latched Asynchronous Pipeline) is shown in Fig. 1. It is designed for a single rail, 4-phase communication protocol between the stages. DLAP imitates the operation of a synchronous master-slave pipeline, by decoupling the pipeline stages. If the pipeline is balanced, DLAP operates the same as a synchronous pipeline: since all pipeline stages finish their computation at the same time, they can all latch the values concurrently into the master part of the registers, while the slave parts retain the previous values. Subsequently, all values latched into the masters are simultaneously transferred to the slaves. DLAP takes advantage of variable delays, as other asynchronous pipelines do. However, unlike other implementations, DLAP is truly decoupled: Thanks to double latching, a stage that has completed early can start processing the next data even if the following stage is still occupied.

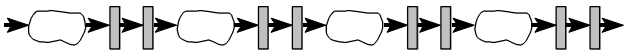

Fig. 1 – A Doubly-Latched Asynchronous Pipeline (DLAP)

A single stage controller for DLAP is shown in Fig. 2. The controller communicates with neighbor stages by *Ready* (Ri, Ro) and *Acknowledge* (Ai, Ao) lines. The latching of data into the master and slave registers is controlled by appropriate signals (Lm, Ls). The Done lines (Dm, Ds) signal when latching has occurred.

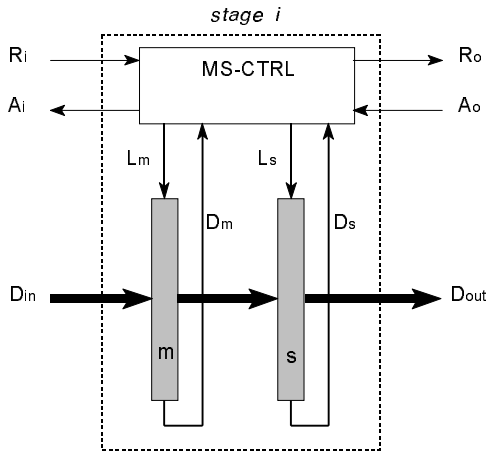

Fig. 2 – A DLAP stage structure.

An example DLAP test circuit is shown in Fig. 3. The *ReadyOut* signal emerging from stage *i* is delayed before entering stage *i*+1. That delay matches the computation delay of the combinational logic between the two stages. We employ an asymmetrical delay [11] in order to make the reset phase as short as possible. If the logic generates a completion signal (e.g., DCVSL [8] or dynamic logic [5]), there is no need to add a special matched delay in the control circuit: *ReadyOut* feeds the combinational logic and the completion signal serves as *ReadyIn* for the following controller.

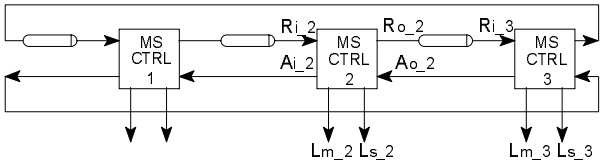

Fig. 3 – The DLAP test circuit.

DLAP can be implemented with either edge-triggered registers or transparent latches, as discussed in the following two sections. In the former case the control is simpler, while in the latter case simpler registers may be employed.

3. Edge-Triggered DLAP

The behavior of a controller for an edge triggered register based DLAP stage is defined by the STG [1] of Fig. 4. STG nodes represent signal transitions (underlined signals are inputs), directed edges are precedence relations, and the black dots are tokens, shown at the initial marking. The graph is 'executed' by moving tokens around. A transition is enabled by the presence of tokens on all edges leading to it. The transition removes those tokens and places new ones on

all edges emanating from it (thus, the number of tokens may change). As described in Fig. 4, the master register is activated by the rising edge of *Lm* when a new value is ready (*Ri* is set), and the previous value has been moved to the slave (as marked by the internal signal *B*, for 'bubble'). Similarly, the slave register is activated by the rising edge of *Ls*, when a new value is ready at the master, and the previous value has been consumed by the following pipeline stage. *Petrify* [2] has been employed to ensure that the STG is safe, persistent, and has a complete state coding so it can be implemented as a speed independent circuit with no hazards. The control circuit implementation synthesized by *Petrify* is depicted in Fig. 5.

Some waveforms obtained from the simulation of a DLAP test circuit (Fig. 3) based on edge triggered registers are presented in Fig. 6. We have designed the two types of DLAP (edge triggered and latched), and have simulated them with SPICE for a 0.8µ, 5V, typical CMOS process. Transistor sizing are optimized for speed and symmetric transitions. We have loaded the latch control signals (*Lm*,*Ls*) to simulate the drive of 32 bit registers. The simulated register driving delay time is 0.9nS. Note that this delay is included in the cycle time (to ensure the correct operation of the control circuit). The relative timings are summarized in section 5. Observe that since the pipeline is balanced, all *Ri* lines are set simultaneously. Consequently, all masters are triggered simultaneously (*Lm* lines). After completing the handshake on the *Ai*/*Ao* lines, all the slaves are triggered (*Ls* lines), and *Ro* signals are set. Following the computational delays, the *Ri*s are set again. The cycle time (from Ri+ to the following Ri+) is 10.9nS, which includes a combinational logic delay of 5.03nS, a logic reset delay of 1nS, and four times 0.9nS for driving the two registers. In other words, the control overhead is only 1.27nS. The response time (passing the data through the double registers, i.e., Ri+ to Ro+) is 2.9nS (which also includes twice 0.9nS register driving delay).

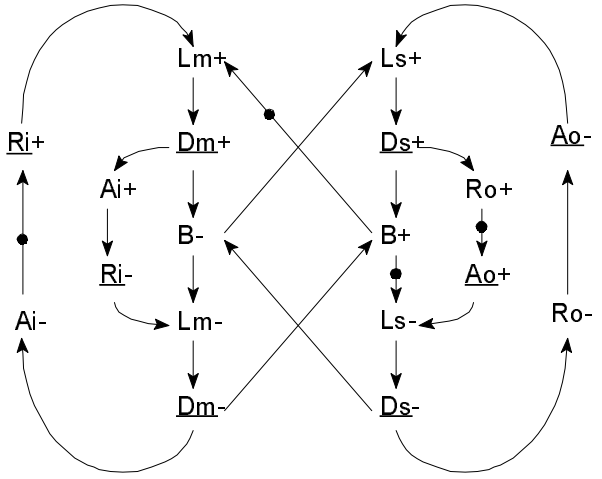

Fig. 4 - STG for a Master-Slave Edge Triggered stage controller.

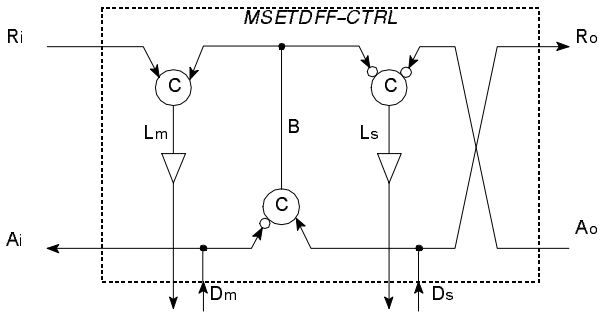

Fig. 5 - A Master-Slave Edge Triggered stage controller circuit implementation.

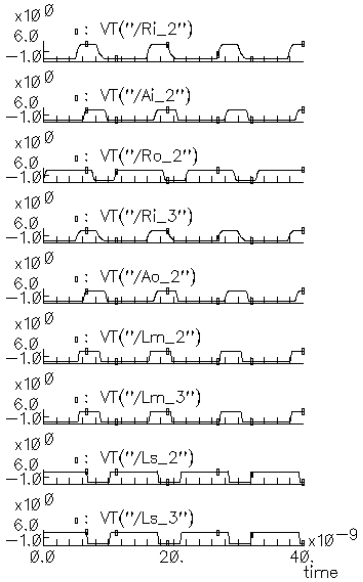

Fig. 6 - Waveforms of Master-Slave Edge Triggered DLAP test circuit.

4. Latched DLAP

Transparent latches are simpler than edge triggered registers. To save power, the latches are used according to the 'blocking latch' scheme [15], i.e., they are kept closed at all times except when data must be latched. Power is saved since hazards are blocked. Note that since the latches are transparent, master and slave cannot be both open at the same time. Consequently, the controller is a bit more complex than for edge triggered DLAP. An extra internal signal is needed to mark which of the two latches has been opened last, and to ensure that the STG has a complete state coding. The proper STG is presented in Fig. 7, and the implementation (synthesized by *Petrify* [2]) is presented in Fig. 8.

The waveforms obtained from the simulation of a DLAP test circuit (Fig. 3) based on transparent latches are presented in Fig. 9, and the relative timings are summarized in section 5 below. Comparing this to the waveforms of the edge triggered DLAP, one can see that the pipeline is balanced and all the masters are enabled at the same time (*Lm* signals), followed by a simultaneous transfer of the data through the slaves (by activating the *Ls* signals). Note also that the *Lm* and *Ls* signals are mutually exclusive. The cycle time (from Ri+ to the following Ri+) is 12.06nS (including the combinational logic delay of 5.03nS, the logic reset delay of 1nS, and 3.6nS for driving the two registers; thus, the control overhead is only 2.43nS). The response time (Ri+ to Ro+) is 6.9nS, including four times the 0.9nS for driving the registers.

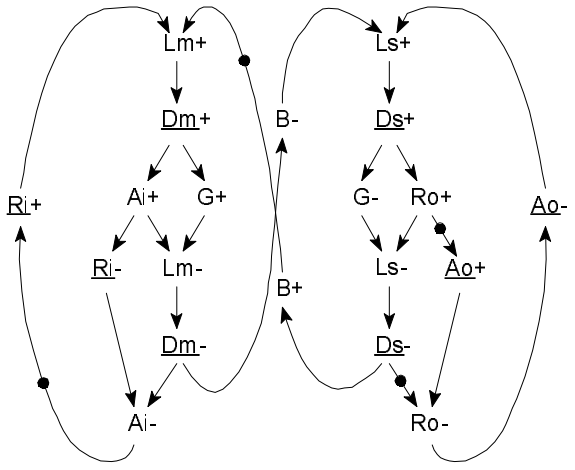

Fig. 7 – STG for a Master-Slave Latch stage controller.

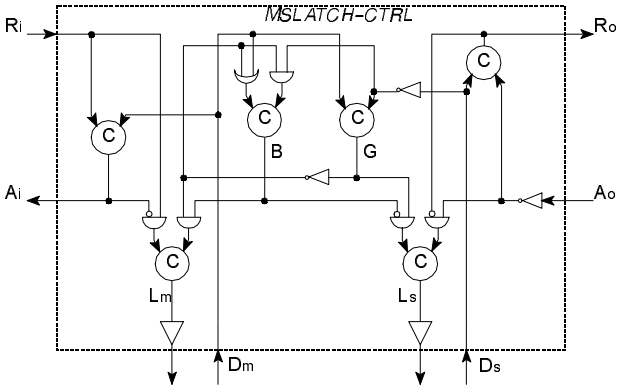

Fig. 8 – A Master-Slave Latch stage controller circuit implementation.

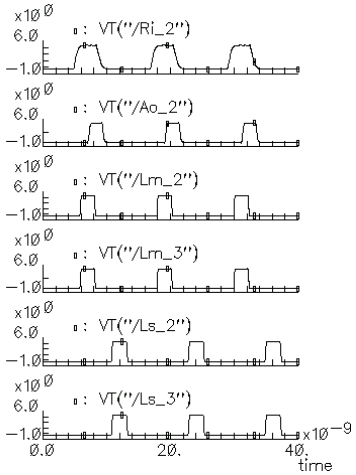

Fig. 9 – Waveforms of Master-Slave Latch DLAP test circuit.

5. Comparative Analysis

We employ a scheduling notation (Fig. 10) to compare the latency incurred by four kinds of two stage pipelines, namely a synchronous pipe, a 'semi-decoupled' ('half handshake') asynchronous pipeline, a 'fully-decoupled' ('full handshake') asynchronous pipeline, and DLAP. Three tasks (*i, j, k*) are to be processed, and the computational delays of each task per each pipeline stage are listed in Table 1. A synchronous design requires the clock cycle time to accommodate the worst case of all calculations over all stages, namely 2 time units, thus requiring eight time units to complete the computation (Fig. 10(a)). The semi-decoupled (or half handshake) pipeline [3, 4, 8] achieves only 50% utilization. Since the pipeline contains only two stages, and they must operate alternatively, the computation takes eight time units (Fig. 10(b)). In a fully decoupled ('full handshake') asynchronous pipeline [4, 8] task *k* cannot start execution at stage A, since task *j* is stalled there until task *i*

frees stage B. Thus the computation requires six time units to complete (Fig. 10(c)). The DLAP completes the computation in only five time units (Fig. 10(d)), since stages A and B are decoupled by the double latches between them, and task *k* is not stalled.

	Task i	*Task j*	*Task k*
Stage A	1	1	2
Stage B	2	1	1

Table 1 - Processing times (in relative time units).

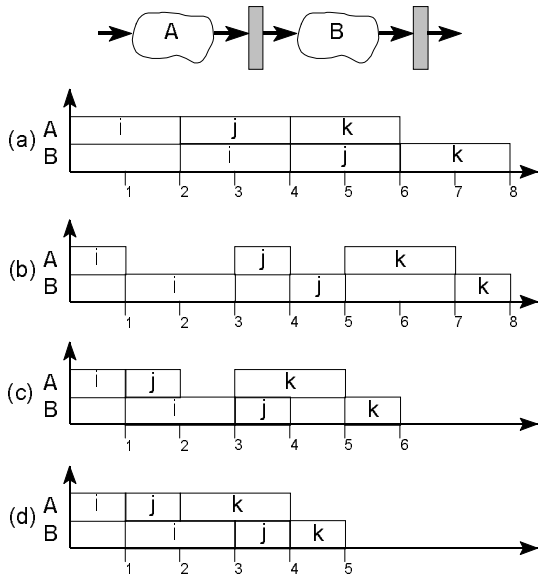

Fig. 10 – Scheduling comparison of alternative pipelines: (a) synchronous, (b) semi-decoupled asynchronous, (c) fully-decoupled asynchronous, and (d) DLAP.

As explained above, we have designed the two types of DLAP (edge triggered and latched), and have simulated them with SPICE. The basic test circuit with several stages and the resulting waveforms are presented in Figs. 3, 6, and 9, respectively. The measured times are summarized in Table 2. Note that the registers driving delays are included in the cycle time.

Relative to synchronous pipelines, DLAP requires about twice as many registers and a small control circuit per stage. However, when replacing each edge-triggered FF with double latches, the area overhead is kept to a minimum. The timing overhead required (for edge-triggered DLAP) is one more register loading delay. The return to zero path of the handshake protocol is kept to a minimum. These times are typically negligible compared to the logic computational time.

		Edge Triggered DLAP [nS]	Latch DLAP [nS]
1	Ri+ → Ai+	1.10	2.06
2	Ai+ → Ri- [a]	2.24	1.62
3	Ri- → Ai-	1.13	0.60
4	Ai- → Ri+ [b]	6.43	7.78
5	Cycle time [c] (Ri+ → Ri+)	10.9	12.06
6	Response Time [d] (Ri+ → Ro+)	2.9	6.9

[a] Including logic reset delay of 1nS
[b] Including computational delay of 5.03nS
[c] Sum of lines 1-4 (include the delay set and reset times)
[d] Measured on a single empty pipe stage (i.e., the time to pass data through the master and slave)

Table 2 - SPICE simulation results

The latched DLAP (relative to the edge triggered DLAP) incurs slightly (about 1nS) longer cycle time, due to the need to precisely sequence more transitions. The total overhead is still negligible compared to typical computational logic delays.

Four phase handshake protocol pipelines with edge triggered registers are also used in [8, 9], where two types of control circuits are presented: 'Half handshake' utilizes only 50% of the pipe, as only every other stage operates at a time. 'Full handshake' is more efficient, and acknowledge signals propagating backwards sequentially, can sometimes overlap stage operation. Four phase pipelines with transparent latches are presented in [3, 4, 5] (the latter employs dynamic logic). However, contrary to DLAP design, the latches are left open most of the time, resulting in possibly higher power dissipation due to data hazards. Their 'semi-decoupled' and 'fully-decoupled' schemes are similar to the 'half-handshake' and 'full-handshake' of [8]. A 2-phase protocol micropipeline using double edge triggered registers is presented in [15], as well as a 4-phase protocol micropipeline using latches and 'blocking latch' scheme. The design is reportedly faster than [3], but it is still a semi-decoupled circuit, limited to 50% pipeline utilization.

The cycle time of a semi-decoupled pipeline include approximately twice the processing delay of the combinational logic because of its 50% duty-cycle operation (i.e., a stage must wait for the following stage to clear before initiating its own next calculation). In a fully decoupled pipeline, even if all stages finish evaluation at almost the same time, a stage cannot latch the result in its output

register until the following stage has. When the pipe is full, operation is limited by a single 'bubble' flowing backwards and the latency overhead is relative to the length of the pipe. The master/slave action of the storage in a DLAP serves the purpose of interleaved bubbles in the pipeline, and relaxes the coupling between the stages. Thus, DLAP is more tolerable to changes in the output rate from the pipe than the other asynchronous pipelines.

We implemented a fully-decoupled pipeline [4] and a full-handshake pipeline [8,9], and ran SPICE simulations to compare the performance. Results show that when the computational load per stage is small (relative to the control overhead), DLAP cycle time is slightly slower due to the extra registers, but it is best for data-dependent delays.

Newer versions of *Petrify* and other synthesis tools may be applied to the STGs presented in this paper, to synthesize simpler control circuits. The circuits generated can be implemented with SR-FF instead of C-elements for standard cell implementation. Faster DLAP controllers might be implemented based on generalized C-elements, or designed to operate according to non-blocking scheme, as in [4]. The control circuits we used were designed to be delay insensitive. However, simple engineering optimization techniques can be applied for lower latency overhead, e.g., overlap control circuit timing with latch operation.

6. Non-Linear DLAPs

Non-linear DLAP data paths can also be created by using *Fork* and *Join* interconnection circuits. A *Fork* is basically a two output pipeline stage. Fig. 11 shows the implementation of a *Fork* for the case of edge triggered DLAP. Note that both following stages share the same *Ro* line, while their *Ao* lines are combined by the C-Element. Similarly, a *Join* interconnection is basically a two input pipeline stage, as presented in Fig. 12. The *Ri* signals are combined by the C-Element.

Many synthesized circuits have complex structures that contain loops (a.k.a. rings). A DLAP ring can be constructed by employing *Join* and *Fork* circuits as shown in Fig. 13.

A ring structure based on fully-decoupled pipeline scheme must contain an extra register to prevent a deadlock [9]. The single bubble going backwards around the loop might limit and slow down the ring operation. DLAP ring is faster than a fully-decoupled one when stage delay is shorter than the acknowledge round trip delay. Using semi-decoupled pipeline to construct a feedback loop yields a ring with only half stages full, since only alternate blocks can store valid values [9]. DLAP rings have enough bubbles and can operate the same way as a synchronous ring.

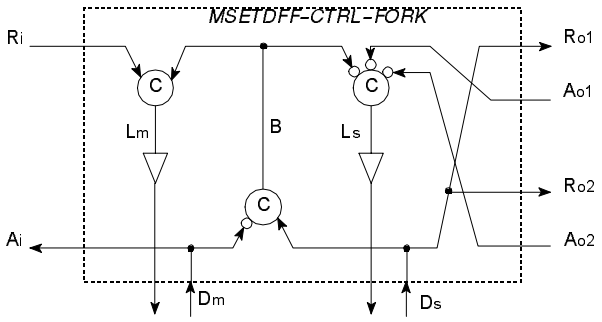

Fig. 11 – A Fork stage implementation, two output pipeline interconnection circuit.

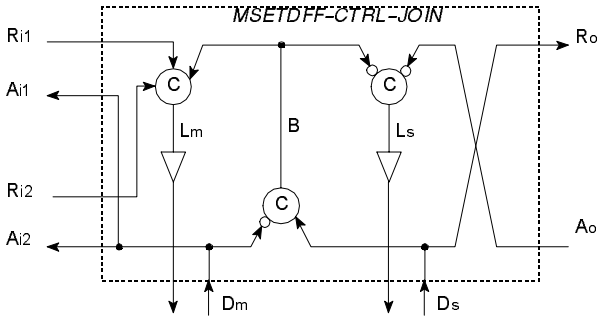

Fig. 12 – A Join stage implementation, two input pipeline interconnection circuit.

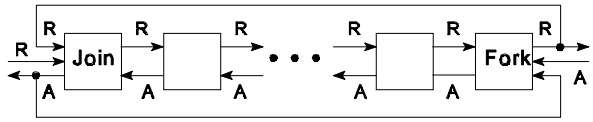

Fig. 13 - A ring DLAP.

7. Conclusion

In our quest for efficient conversion of synchronous circuits into asynchronous ones, we have examined various asynchronous pipeline schemes [8, 9, 3, 4, 5, 15, 12], and have found that none operates as efficiently as a balanced synchronous pipeline. Consequently, we have developed the doubly-latched asynchronous pipeline (DLAP) which employs master-slave registers. DLAP is capable of truly decoupled operation: In a balanced pipeline all stages can shift data simultaneously, and when variable (data dependent) delays are encountered, execution is faster than previous designs. DLAP architecture is most suitable for synchronous-like operation, and automatic conversion when we wish to eliminate the clock (for power saving and easier interface to other asynchronous units) without redesign. We have shown implementations based on either edge triggered registers or transparent latches. Both designs have been defined with STGs, verified, and fully simulated and compared with previous architectures.

References

[1] T.-A. Chu, *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*, PhD thesis, MIT Laboratory for Computer Science, Jun. 1987.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," Technical Report, Dept. d'Arquitectura de Computadors, Universitat Politecnica de Catalunya, Apr. 1996.

[3] P. Day and J. V. Woods, "Investigation into Micropipeline Latch Design Styles," *IEEE Trans. on VLSI Systems*, **3**(2), pp. 264-272, Jun. 1995.

[4] S. B. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Trans. on VLSI Systems*, **4**(2), pp. 247-253, Jun. 1996.

[5] S. B. Furber and J. Liu, "Dynamic Logic in Four-Phase Micropipelines," *2nd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems* (*Async'96*), Mar. 1996.

[6] S. Hauck, "Asynchronous Design Methodologies: An Overview," *Proc. IEEE*, **83**(1), pp. 69-93, Jan. 1995.

[7] R. Kol, R. Ginosar, and G. Samuel, "Statechart Methodology for the Design, Validation, and Synthesis of Large Scale Asynchronous Systems," *2nd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems* (*Async'96*), Mar. 1996.

[8] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specification," *IEEE Trans. on Computer-Aided Design*, **8**(11), pp. 1185-1205, Nov. 1989.

[9] T. H.-Y. Meng, R. W. Brodersen, ann D. G. Messerschmitt, "Asynchronous Design for Programmable Digital Signal Processors," *IEEE Trans. on Signal Processing*, **39**(4), pp. 939-952, Apr. 1991.

[10] D.L. Perry, *VHDL*, 2nd ed., McGraw-Hill, 1994, Ch. 9,10: Synthesis.

[11] C. L. Seitz, System timing, in Carver A. Mead and Lynn A. Conway, *Introduction to VLSI Systems*, chapter 7, Addison-Wesley, 1980.

[12] J. Sparso and J. Staunstrup, "Delay-insensitive multi-ring structures," *INTEGRATION, the VLSI journal*, **15**(3), pp. 313-340, 1993.

[13] I.E. Sutherland, "Micropipelines," *Commun. ACM*, **32**(6), pp. 720-738, Jun. 1989.

[14] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij, "Asynchronous Circuits for Low Power: a DCC Error Corrector," *IEEE Design & Test*, **11**(2), pp. 22-32, Jun. 1994.

[15] K. Y. Yun, P. A. Beerel, and J. Arceo, "High-Performance Asynchronous Pipeline Circuits," *2nd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems* (*Async'96*), Mar. 1996.