International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

Design for Exterminating Synchronization Latency Using Sequenced Latching

M.Naveen kumar[*1]; N.Srikanth [*2]& J.E.N Abhilash[*3]
[#] Department of ECE, Swarnandhra College of Engg & Tech.,A.P,India
meesala434@gmail.com[*1] srikanth648@yahoo.co.in[*2] abhilash.jen@gmail.com[*3]

ABSTRACT:
In the present generation where the modern multi-core systems which have a large number of components that operate with different clock domains and communicating through asynchronous interfaces. And due to this interfaces synchronizer circuits are used, which guard against metastability failures but this introduces delay in processing the asynchronous input. Hence we propose a novel method that hides synchronization abeyancy by overlapping it with computation cycles. In this work a method that performs speculative computations during synchronization cycles and hence prevented synchronization time from incurring abeyance. Our methods relies on sequence the latching of data using synchronizer state during synchronization cycles and automatically re-latch if any corrupt data is latched. Synthesis results reveal that our approach achieves average savings of Area, costs and power costs compared to two similar speculative techniques. The Proposed design will be implemented using Verilog HDL.

Keywords: Duplication; latency; met stability; speculation; synchronization.

1.INTRODUCTION

For the longest part of the history of integrated circuits, synchronous operation has allowed designers to put together an ever-larger number of components without having to worry much about the complex timing issues of their interoperability. Clock domain interfacing and the problem of flip-flop metastability are among the challenges that must be addressed to facilitate this transition and support the creation of more powerful heterogeneous many-core systems. When components in different clock domains attempt to communicate, the receiver is always at a risk of failure due to the finite probability that sender's request arrives at a bad time. Such occurrences can Cause flip-flops on the receiving module to become "metastable" and take a theoretically unbounded time to decide whether to go logic high or low. And the resulting failures and the large numbers of failed attempts that have been made at avoiding it. Where these failures are impossible to eliminate completely, but can be minimized to an desired level by re-sampling the input signal by cascading the flip-flops, which is known as a synchronizer. However this introduces the latency and reduces the performance of the system. Therefore, a synchronizer design gives a reliability versus performance, where the mean time between failure(MTBF) of the design is traded for the time having for synchronization(settling time t_s). And relationship between the MTBF and *ts* is captured by the textbook formula.

$$\mathrm{MTBF} = \frac{e^{t_s/\tau}}{f_c \times f_d \times T_w}$$

Where t is the metastability regeneration time constant, f_c is the clock frequency, f_d is the data arrival rate, and T_w is a reference time window for the exponential relationship.

Here the MTBF of synchronization is adjusted by the designer's choice of t_s. And later it is chosen as an integer multiple of the clock period by presenting a flip flops in chain in order to perform synchronization. Two flip-flop synchronizers are the most common and are likely to yield a MTBF in the order of thousands of years in modern processes(t = 20 ps, taking $f_c = f_d = 1.0$ GHz and Tw = 1ns). Designs operating with lower

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

supply voltages, extreme temperatures, high clock frequencies exhibiting larger variations in design parameters requires four stages of synchronization to maintain similar MTBF.

Also latency by this synchronizer chains has impact on the performance of latency-sensitive applications and therefore alternative methods are used to reduce this latency and they are:

1.1 Circuit-level Designs

In this design where the synchronization time can be reduced by using faster flip-flops (lower t). In the early attempts where the jamb latch was used to meet the requirements of synchronization. While this type of latches provides the same level of reliability and also requiring lower synchronizing time.

1.2 Exploiting Known Timing relationships

In this case when the communicating clocks which share timing relationship that avoids synchronization latency. Whenever clock-data conflicts might occur and can be detected and avoid sampling the input in some critical intervals.

1.3 Using Pausible/Stretchable Clocks

When pausing the receiver's clock to an unbounded amount of time then the need for synchronization can be eliminated. This can be done by adding a mutual-exclusion element that arbitrates between the receiver's clock until any occurred metastable states are resolved.

TABLE I
COMPARISON OF SYNCHRONIZATION LATENCY MITIGATION METHODS

	Circuit Designs	Timing Relationships	Pausible Clocks
Standard-cell design flows	-	✓	✓
Eliminates latency completely	-	✓	✓
Supports asynchronous clocks	✓	-	✓
Supports external clocks	✓	✓	-
Scales well with technology	-	-	-

Despite the advantages in the above three contexts, their applications remains very limited. Solutions from first type requires full custom design flows, as less common in use than that of standard cell libraries. In the second type, clocks have dependable timing relationships. And in the last type a local generator to generate plausible clocks have poor stability. So these applications are limited for small scale applications and the differences are observed in table 1.

Hence to exterminate latency during synchronization a redundant hardware is used in order to perform speculation computations. Here the latency can be hidden by overlapping it with the number of computational cycles in equivalent that follows the data arrival. In computing it requires n synchronization cycles and m computational cycles, where for the processing time it takes of max (m , n) . The maximum of the cycles is the processing time in speculative computations, where in the conventional approach the processing time required is the addition ($m + n$). And this difference is seen in fig.1.

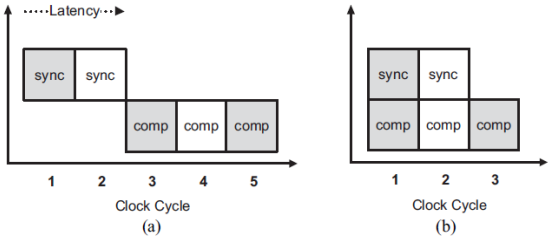

Fig-1: Hiding synchronization latency by speculative computations (sync = synchronization cycle, comp = computation cycle). (a) Conventional approach. (b) Speculative computations.

Over the two types speculation is preferred to the conventional type in reducing synchronizing latency. As speculation process does not aim for the synchronization and also it is architectural one. And also it does not requires faster flip-flops since it is not aiming for the synchronization. In the second context where reliability and the duplicated hardware will be an increasingly affordable in the future technologies that is because of continuous growth in the design area. But the performance of the flip-flops is becoming worse due to supply voltage scaling [5] and growing process variations [8] and also the clock relatively timing relationships, which are difficult to verify [11].

However it has advantages, speculation type needs large amount of duplicated hardware. As it requires much hardware the area and the power costs is also more. And now we introduced a new technique which is called sequenced latching that has better performance than the existing system in costs and latency improvements.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

2. BACKGROUND

This section describes the existing methods of speculative technique in reducing the synchronization latency. And there are two methods, first (Datapath unfolding) which is a general type hardware duplication solution that helps in eliminating the latency during synchronization. And secondly (speculative synchronization) which mainly used for the synchronization solution specifically.

2.1 Overview Of Speculation

Speculation which can be said as using either time or resource redundancy in order to perform certain useful work. In the modern digital systems where this speculation is used in different abstraction levels. Considering an example of memory management, where the pre-fetched data is stored in the caches, this increases the processing speed [17]. And also this increases the though put in the branch prediction and the execution of the instructions [18]

2.2 Datapath Unfolding [21]

In this there are two types of data path which are pipelined and non-pipelined. Where it is easy to obtain speculative calculations in pipelined system because restoring the pipeline state in the case of misspeculation is trivial. Considering an example, when any pipelined processor mispredicts a branch, instructions that are invalid in the fetch and decode stages can be eliminated. And this same cannot be done in the case of non-pipelined systems. Moreover the speculative computations cannot be reversed in the same straight forward manner. And this id due to having of loop dependencies (i.e., feedback paths).

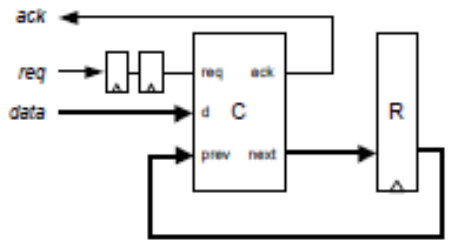

Figure-2: Asynchronous Receiver

Apart from this the arbitrary designs can use this speculation computations by unfolding this designs[22]. Moreover it produces the same output as the actual design after multiple number of cycles which is similar to that of actual pipeline depth.

Where unfolding is widely used by the compilers[23]-[25], and schedulers[26],[27] in order to increase the throughput of the execution. And in the present technology it is mainly used in the digital signal processors [28], [29], and from here it is said to be as datapath unfolding.

Considering an asynchronous receiver shown in fig 2 And the receiver is taken as the moore machine which consists of a combinational block C and and the stste register R. and moreover the asynchronous data which has to be transferred is controlled by the handshake signals which are termed as REQ and ACK. IN this case for illustration purpose four-phase handshake protocol is used by the receiver. Hence the data made available on the bus by the sender and then asserts REQ. And then the data on the bus is latched by the bus and few data dependent computations are done and asserts ACK to acknowledge that the consumption of data is completed. After this process REQ is de-asserted by the sender and the receiver de-asserts ACK.

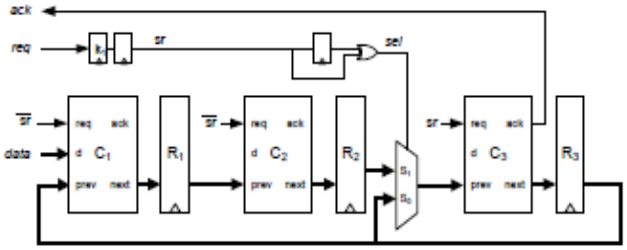

Figure-3: Unfolded Asynchronous Receiver

In order to maintain reliability, two flip-flops are used at the request to synchronize. But due this flip-flops there is a delay of two subsequent cycles. However the latency caused by this can be eliminated by using two additional datapath instances and this arrangement is seen in fig 3. In this it is clear that the datapath of the receiver has been unfolded into three stages of cyclic pipeline. In the first two stages where the speculative computations are done holding the valid data by the bus. And considering the third stage it is the same copy of the datapath. When original REQ which appears at the synchronizer output, and the state stored in the register R_2 is transferred to C_3 in order to compute the third stage. And here the actual datapath (R_3, C_3) resumes its computation and after its completion sends acknowledgement to the sender.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

And this process requires isochronic principle where the REQ must arrive after a required large delay of the data [30]. In this case even if there is any metastable state is even latched by R_1 and passing to the nest state R_2.

2.3 Speculative Synchronization [31]

In this type where a single flip-flop is used in order to synchronize the handshake signals and also to detect n cycles after, if any metastable state occurred due to any synchronizing flip-flop. As for *n*-cycles it requires n datapath state register duplicates but has no combinational logic. However any metastable state is identified and state is restored, it has to be stated from the beginning and requires additional delay of *n* cycles. However such type of cases rarely and also has minimum impact on overall latency and it provides a 1-2 cycles of latency which is opposed in the case of datapath unfolding .

3. SEQUENCED LATCHING

This is a novel technique to latch data during the synchronization cycles. And the synchronizers can be used as a state machine to sequence the series of latching operations. Moreover the synchronizer is fixed such that its state does not change when the latching operation fails. Hence any failed latching operations are retried in the coming cycles. This technique is known as sequenced latching. And this analysis uses assumptions about the behavior of metastability of the flip-flops.

Here the metastable state of the flip-flop can cause an increase in the clock-to-q delay, but this does not change the monotonic nature or the rise/fall time of its output. Although toggling outputs and long transition times due to metastability reported in [2], and these occurs under circuit-level conditions which occurs rarely in practice. The output may be non-monotonic of a metastable flip-flop when the threshold voltage (V_{th}) lies in between the metastable voltages of its master and slave latches. However supply voltage noise may disturb the metastable voltage of a latch, and it cannot push latch which has diverged by a enough large voltage V_m back into metastable state. Hence a flip-flop whose V_{th} is far from its metastability voltage of its latches by atleast V_m will necessarily have a monotonic output.

3.1 Isochronicity

Similar to the datapath unfolding technique, present method is based on isochronic handshakes[30]. Isochronicity states the data and the request signals travel separately and the data which is available is indicated by the request signal that has to be arrive a sufficient time later. Another technique to this approach is to encode valid information within the data itself. This one eliminates the requirement of isochronic timing constraint but requires more complex data encoding and decoding circuits and number of signal lines.

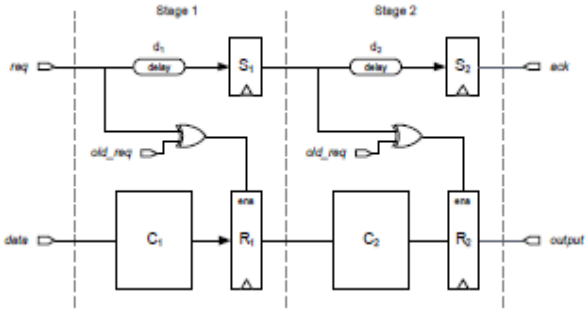

Figure-4: Two Stage Pipeline

3.2 Technical Overview

Our technique is illustrated in the fig 4. In the design where the two flipflops (s_2 and s_3) synchronize an asynchronous handshake request (REQ) and it acts as a state machine to control the passing of the data through pipeline.

Once the input of the pipeline is held stable by the sender, it does not change until the data item propagates through the pipeline is acknowledged. However when the stable data is held on DATA, the transition of REQ propagates through the synchronizer chain enabling the register R_1 in succession. However, problems might arise as REQ is asynchronous to receive and may cause S_2(and possibly S3 afterwards) to become metastable. When any flip-flop experiences prolonged clock to q transition due to metastability, the setup condition of the path (S_i to R_i) which is not satisfied and R_i may latch corrupt values.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

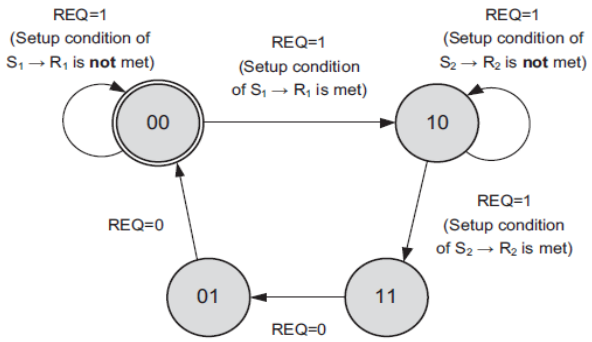

Figure-5: Synchronizer State Machine

And the main idea of our technique is to eliminate latching of corrupt values from the previous stages of the pipeline by preventing late transitions of any synchronizer flip-flop from being captured by its successor. And this can be done by inserting delay elements between the synchronizer flip-flops shown in fig.4. With this enough delay the late transitions of any flip-flop, which may not meet the setup condition of the register R_i wil necessary fail to be captured by S_{i+1}. Therefore if the setup condition is not met then this stalls the pipeline for a cycle and latches the corret data in the next cycle.

4. IMPLEMENTATION

4.1 Operating Principle

In this section, the speculative computations are performed using two datapath instances. Moreover this is connected in a cyclic pipeline and sequenced latching is used to enable them alternately. And the basic application of the sequenced latching is seen in the fig.6. Two datapath instances are connected in cyclic pipeline and these are enabled alternately using ODD and EVEN, which are combinationally derived from the synchronizer nodes and asserted during the odd and even cycles of synchronization.

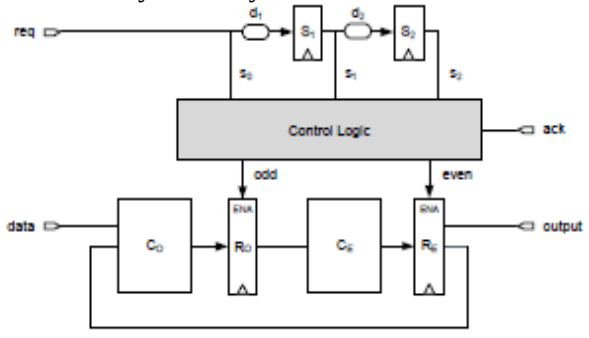

Figure-6: Cyclic Pipeline Using Sequenced Latching

When REQ transitions indicates the validity of data, and if none of the synchronizer flip-flops become metastable, then the signals ODD and EVEN will be asserted in the alternating cycles. With this the state registers (R_O and R_E) will be enabled alternately and also the datapath state will alternate between them. And if any of the flip-flop becomes metastable then is behaves as in the section 3.2. Therefore any corrupt data is latched by the registers then the state will not change ,ODD and EVEN will not toggle in the following cycle and the corrupt state is re-latched.

4.2 Proposed implementation

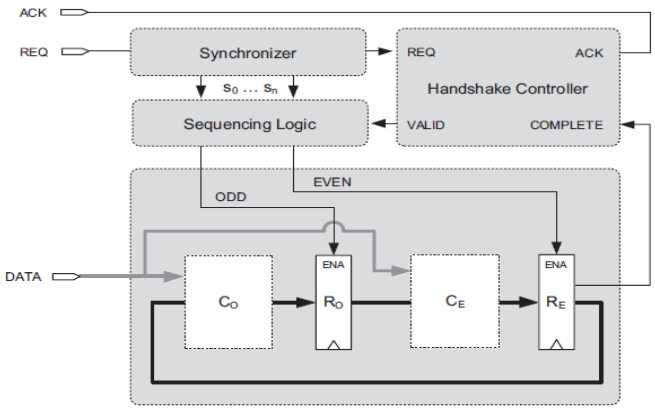

Figure-7: Asynchronous Controller Using Sequenced Latching

Fig-7 shows the proposed implementation block diagram. In this the handshake controller sits outside the datapath and communicates using the signals VALID and COMPLETE. After the synchronization then the controller asserts VALID signal. Where COMPLETE is asserted only after the end of the computations by the datapath. Moreover the ODD and EVEN signals are generated by the sequencing logic block to complete the number of datapath state transitions equal to the number of the synchronization cycles. In case of the datapath computations cycles (m) are more than the synchronization cycles (n) then the alternate behavior of the ODD and EVEN has to be maintained after synchronization. And this can be done by adding a toggle flip-flop in the sequencing logic block.

5. Simulation Results:

With the existing system using datapath path unfolding where the simulation results can be obtained in the fig.8. Since in the case of this technique where it uses the two handshake protocol which has the req and the ack in order to transmit the data. Initially the reset is set to high for one

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 10, October 2015
Available at http://internationaljournalofresearch.org

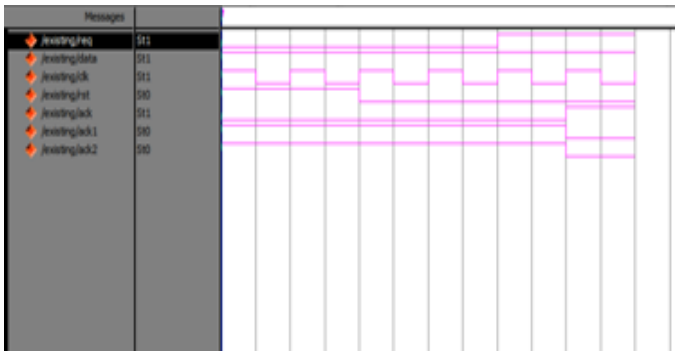

Figure- 8: Simulation results for reducing Latency using Datapath Unfolding

clock cycle and from the next cycle when the request is set to high i.e, (Req=1) then after two clock cycles where the data is completed and then acknowledges to the sender that is (ACK=1) after two clock cycles. But here both acknowledges at a single clock cycle has both has to travel for each clock cycle. Due to this the synchronization is affected . Hence this problem is not seen in the sequenced technique.

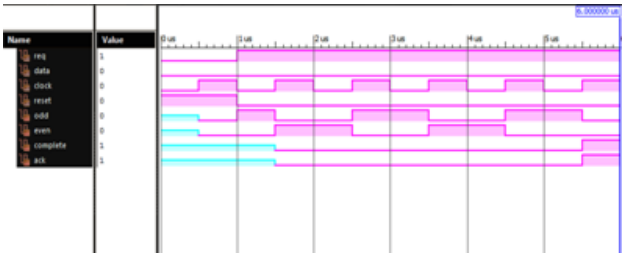

Figure-9: Simulation results for reducing latency using sequenced latching

In the figure9. the simulation results of implementing the sequenced latency for reducing latency is observed. Initially the reset is set to high for one clock cycle then in the next cycle request is set to high i.e., REQ=1 then the ODD and EVEN signals will toggle for four cycles and on the fifth cycle where the complete and ACK goes high. As this works on the principle of four handshake protocol then after four cycles as the data consumption is over then ACK goes high.

Design	Delay (ns)	Power Delay Product (mw x ns)
Eliminating Synchronization Latency Using Datapath Unfolding	1.343	68.493
Eliminating Synchronization Latency Using Sequenced Latching	1.319	65.95

Figure-10: Comparison table of performance parameters for datapath Unfolding and Sequenced Latching

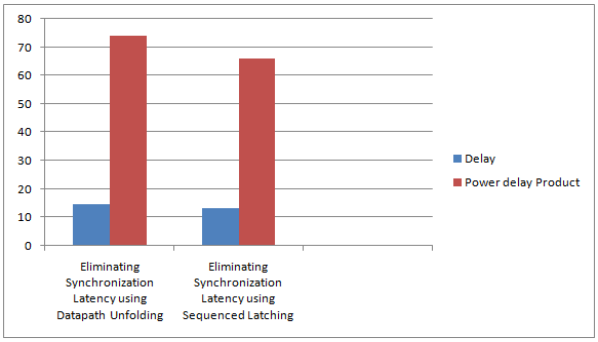

Figure-11: Comparison chart of performance parameters for Datapath Unfolding and Sequenced Latching

In performing both the techniques where the proposed sequenced latching has better performance than the datapath unfolding in terms of power and the delay. And also there is a reduction of 0.024ns from the existing model and in the case of power delay product there is a reduction of 2.543mw. This can be clearly seen in the fig-10

6. CONCLUSION

Here we presented a speculative technique which is sequenced latching which performs speculative computations during synchronization cycles and hence prevented synchronization time from incurring latency. Our method relies on using the synchronizer state to sequence the latching of data during synchronization cycles and automatically re-latch any corrupt data. Which is the overcome of the Datapath Unfolding which relies on loop unrolling to create duplicate state machines whose function is to compute speculative data-dependent states. These states are not used by the original machine until synchronization is complete and the validity of data is confirmed. As this value remains same even during the synchronization cycle which is not practical. It is shown that this approach is functionality correct and that it does not violate any of the principle tenets of the metastability problem.

7. REFERENCES

[1] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.*, vol. 22, no. 4, pp. 421–422, Apr. 1973.

[2] I. W. Jones, S. Yang, and M. Greenstreet, "Synchronizer behavior and analysis," in *Proc. 15th*

IEEE Symp. Asynchron. Circuits Syst., May 2009, pp. 117–126.

[3] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design Test Comput.*, vol. 28, no. 5, pp. 23–35, Sep.–Oct. 2011.

[4] D. Kinniment, A. Bystrov, and A. Yakovlev, "Synchronization circuit performance," *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 202–209, Feb. 2002.

[5] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "A robust synchronizer," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerg. VLSI Technol. Archit.*, Mar. 2006, pp. 442–443.

[6] S. Yang, I. Jones, and M. Greenstreet, "Synchronizer performance in deep sub-micron technology," in *Proc. 17th IEEE Int. Symp. Asynchron.Circuits Syst.*, Apr. 2011, pp. 33–42.

[7] J. Zhou, M. Ashouei, D. Kinniment, J. Huisken, and G. Russell, "Extending synchronization from super-threshold to sub-threshold region," in *Proc. IEEE Symp. Asynchron. Circuits Syst.*, May 2010, pp. 85–93.

[8] J. Zhou, D. Kinniment, G. Russell, and A. Yakovlev, "Adapting synchronizers to the effects of on chip variability," in *Proc. 14th IEEE Int. Symp. Asynchron. Circuits Syst.*, Apr. 2008, pp. 39–47.

[9] M. Baghini and M. Desai, "Impact of technology scaling on metastability performance of cmos synchronizing latches," in *Proc. 7th Asia South Pacific, 15th Int. Conf. VLSI Design. Proc., Design Autom. Conf.*, Jan. 2002, pp. 317–322.

[10] M. Greenstreet, "Implementing a stari chip," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Process.*, Oct. 1995, pp. 38–43.

[11] A. Chakraborty and M. Greenstreet, "Efficient self-timed interfaces for crossing clock domains," in *Proc. 9th Int. Symp. Asynchron. Circuits Syst.*, May 2003, pp. 78–88.

[12] L. Sarmenta, G. Pratt, and S. Ward, "Rational clocking [digital systems design]," in *Proc. IEEE Int. Conf. Comput. Design, VLSI Comput. Process.*, Oct. 1995, pp. 271–278.

[13] W. Dally and S. Tell, "The even/odd synchronizer: A fast, all-digital, periodic synchronizer," in *Proc. IEEE Symp. Asynchron. Circuits Syst.*, May 2010, pp. 75–84.

[14] J. Kessels, A. Peeters, P. Wielage, and S.-J. Kim, "Clock synchronization through handshake signalling," in *Proc. 8th Int. Symp. Asynchron.*

[15] K. Yun and A. Dooply, "Pausible clocking-based heterogeneous systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 4, pp. 482–488, Dec. 1999.

[16] R. Dobkin, R. Ginosar, and C. Sotiriou, "High rate data synchronization in gals socs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 10, pp. 1063–1074, Oct. 2006.

[17] Z. Wang, T. O'Neil, and E.-M. Sha, "Optimal loop scheduling for hiding memory latency based on two-level partitioning and prefetching," *IEEE Trans. Signal Process.*, vol. 49, no. 11, pp. 2853–2864, Nov. 2001.

[18] M. Younis, T. Marlowe, A. Stoyen, and G. Tsai, "Statically safe speculative execution for real-time systems," *IEEE Trans. Softw. Eng.*, vol. 25, no. 5, pp. 701–721, Sep.–Oct. 1999.

[19] G. Lakshminarayana, A. Raghunathan, and N. Jha, "Incorporating speculative execution into scheduling of control-flow-intensive designs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 3, pp. 308–324, Mar. 2000.

[20] A. Bhowmik and M. Franklin, "A general compiler framework for speculative multithreaded processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 8, pp. 713–724, Aug. 2004.

[21] N. Park and A. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 7, no. 3, pp. 356–370, Mar. 1988.

[22] K. Parhi and D. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, no. 2, pp. 178–195, Feb. 1991.

[23] K. Ebcio˘glu, "A compilation technique for software pipelining of loops with conditional jumps," *ACM SIGMICRO Newslett.*, vol. 19, no. 3, pp. 36–41, Sep. 1988.

[24] A. Aiken and A. Nicolau, "Optimal loop parallelization," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement.*, 1988, pp. 308–317.

[25] M. Stoodley and C. Lee, "Software pipelining loops with conditional branches," in *Proc. 29th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 1996, pp. 262–273.

[26] L.-F. Chao and E. Hsing-Mean Sha, "Scheduling data-flow graphs via retiming and unfolding," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 12, pp. 1259–1267, Dec. 1997.

[27] L.-F. Chao, "Scheduling and behavioral transformation for parallel systems," Ph.D. dissertation, Dept. Comput. Sci., Princeton Univ., Princeton, NJ, USA, 1993.

[28] L. Lucke, A. Brown, and K. Parhi, "Unfolding and retiming for highlevel DSP synthesis," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 4. Jun. 1991, pp. 2351–2354.

[29] G. Goossens, J. Vandewalle, and H. De Man, "Loop optimization in register-transfer scheduling for DSP-systems," in *Proc. 26th Conf. Design Autom.*, Jun. 1989, pp. 826–831.

[30] A. Martin and M. Nystrom, "Asynchronous techniques for system-onchip design," *Proc. IEEE*, vol. 94, no. 6, pp. 1089–1120, Jun. 2006.

[31] D. J. Kinniment and A. Yakovlev, "Low latency synchronization through speculation." in *Proc. Power Timing Model. Optim. Simul. Conf.*, 2004, pp. 278–288.

[32] *Nangate 45nm Open Cell Library*. (2012) [Online]. Available:http://www.nangate.com