Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 50 (2004) 687–696
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A clocking technique for FPGA pipelined designs

Oswaldo Cadenas *, Graham Megson

University of Reading, School of Systems Engineering, P.O. Box 225, Whiteknights, Reading RG6 6AY, UK

Received 29 July 2003; received in revised form 12 March 2004; accepted 1 April 2004
Available online 2 July 2004

Abstract

This paper presents a clocking pipeline technique referred to as a single-pulse pipeline (PP-Pipeline) and applies it to the problem of mapping pipelined circuits to a Field Programmable Gate Array (FPGA). A PP-pipeline replicates the operation of asynchronous micropipelined control mechanisms using synchronous-orientated logic resources commonly found in FPGA devices. Consequently, circuits with an asynchronous-like pipeline operation can be efficiently synthesized using a synchronous design methodology. The technique can be extended to include data-completion circuitry to take advantage of variable data-completion processing time in synchronous pipelined designs. It is also shown that the PP-pipeline reduces the clock tree power consumption of pipelined circuits. These potential applications are demonstrated by post-synthesis simulation of FPGA circuits.

Keywords: Pipelines; Micropipeline; FPGAs

1. Introduction

Pipelining is a key implementation technique for achieving high throughput and parallelism [7]. Pipelined systems are composed of stages, typically separated by storage elements. Each stage takes in data processed by the previous stage, processes it, and delivers new data to a subsequent stage. Based on the control mechanisms for data interchange, pipelining is classified as synchronous or asynchronous [11]. Synchronous pipelining uses a simple control based on a global clock signal, where computation starts at all stages in the pipeline after a clock signal arrives. The corresponding results must be ready before a new clock signal is issued. The clock period is thus constrained by the longest delay in the pipeline plus timing specifications of the physical storage elements. This demands a careful balance of the combinational delay of the pipeline stages when building pipelined datapaths [12]. Usually a pipeline holds a fixed amount of data, that is, the input rate and the output rate of data is equal or inelastic and the output exhibits a delay known as latency. The insertion of dummy data or bubbles at some points in the pipeline, common in a RISC

* Corresponding author. Tel.: +44-118-3786358; fax: +44-118-3788583.
E-mail addresses: o.cadenas@reading.ac.uk (O. Cadenas), g.m.megson@reading.ac.uk (G. Megson).

1383-7621/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2004.04.001

datapath control, is an example of a simple elastic pipeline. Keeping the processing throughput in elastic pipelines requires extra control in synchronous pipelines, and is often application-specific. For instance, designing a FIFO. Sutherland in his classical work [16] inspires a persistent interest in asynchronous pipelining motivated by its original simplicity and regularity for implementing elastic pipelining.

FPGAs do not have the key elements used by today's asynchronous designs since they are orientated to implement circuits synchronously [6]. Indeed, asynchronous elements are costly to implement in FPGAs. For example, a Muller C-element, the most frequently used element for controlling micropipelining, takes 1 CLB (Xilinx configurable logic block). Consequently, special FPGA architectures for asynchronous design have been proposed, see [5] for example. More pragmatic efforts have concentrated on optimizing asynchronous elements commonly at the CMOS level [15].

We introduce, in Section 3, a new clocking technique referred to as PP-pipelining, that better exploits commercial FPGA resources for asynchronous-like computation. PP-pipelines use simple edge-triggered D-type flip-flops as registers and so suitable for efficient FPGA implementation. A PP-pipeline has a synchronous control mechanism run by a global clock to generate local clock signals in the form of pulses and hence the registers do not share a global clock. The basic control mechanism is built on a simple state machine accepting asynchronous status signals from neighboring stages. The pipeline as a whole runs at a cycle time which is a function of the global clock period and the longest delay in the pipeline. With PP-pipelines the portability from asynchronous to synchronous pipeline operation is straightforward at the design description level, thus facilitating reduced design time and effort. A methodology to convert existing pipelined designs to PP-pipelined designs is illustrated in Section 3. Section 4 discusses two potential applications for the PP-pipeline clocking technique. Section 2 briefly outlines two selected asynchronous pipeline schemes that are feasible for FPGA implementation for comparative purposes. Some concluding remarks are given in Section 5.

2. Asynchronous pipelining feasible on FPGAs

In asynchronous pipelining the storage elements do not use a global clock. Instead a local handshake protocol based on request and acknowledge signals between neighboring stages is used for coherent communication of the data. Asynchronous systems separate signals into: control signals and data signals. There are schemes to generate and manage the control signals and ways to encode the data signals and relations between them [3,13]. Commonly, data can be encoded as dual-rail and as bundle-data. In dual-rail a pair of signals encode a bit of data and its own request. With bundle-data, data is presented by the sender and after it becomes valid then and only then a request signal is asserted, called the bundling constraint. The request/acknowledge handshake can follow a two-phase or a four-phase signalling mechanism. Micropipelining [16] is a very simple and modular solution for asynchronous pipelining using bundle-data and a two-phase protocol. Control in micropipelining is based on Muller C-elements and the storage on a special capture-pass latch design. In [20] two-phase micropipelines are explored using double edge-triggered D-type flip-flops instead of capture-pass latches. A four-phase micropipeline variation which uses edge-triggered D-type flip-flops is presented in [13]. A related scheme using a four-phase protocol communication with operation similar to synchronous pipelines is presented in [10]. The scheme is described as a Double-Latched Asynchronous Pipeline (DLAP). FPGAs are populated with edge-triggered flip-flops; double edge-triggered flip-flops can be synthesized by employing two edge-triggered flip-flops and multiplexers. Consequently all these schemes based on micropipelining are feasible for FPGA implementation.

2.1. Two-phase micropipeline

Micropipelining is based on a very simple state rule between adjacent stages consisting of two conditions. At stage i new data is latched only after it is available from stage $i-1$ (condition I) and data processed at stage i has been consumed by the stage $i+1$ (condition II). The basic control

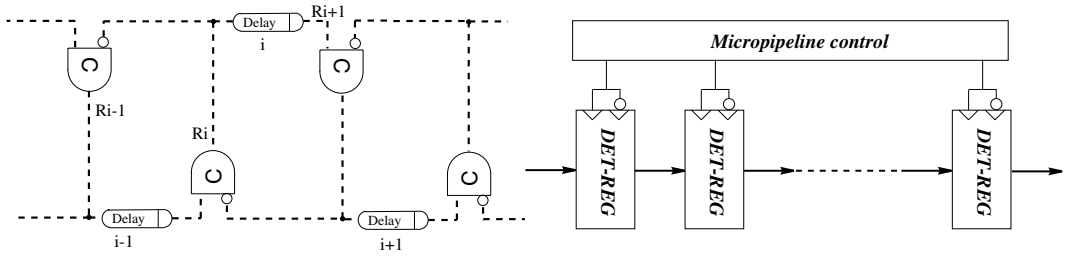


Fig. 1. Left: Micropipeline control circuit. Right: A micropipeline controlling double edge-triggered registers (Det-Reg) with no processing logic by a two-phase mechanism.

circuit for micropipelining is shown on the left of Fig. 1. When control R_{i+1} activates (after $delay_i$) stage i indicates data ready to be consumed by the stage $i+1$. Assuming that stage $i+2$ has accepted the previous data, then after a Muller's C-element propagation delay the control signal to latch the new value by stage $i+1$ is asserted. This signal events back to stage i indicating that the data has been taken. The mechanism repeats for signal R_i after $delay_{i-1}$ for stage i to latch the data delivered by stage $i-1$. A two-phase micropipeline as proposed in [20] is shown on the right of Fig. 1. It uses pipeline registers based on double edge-triggered D-type flip-flops clocked by the local pulses generated at all the R_k nodes of the micropipeline circuit on the left of Fig. 1.

2.2. Four-phase micropipeline and edge-triggered DLAP

A four-phase bundle-data pipeline is shown in Fig. 2 [13]. The larger Muller C-elements in the figure generate the local clocking signals when both a request signal from a previous stage (data ready) and the acknowledge from the next stage (data taken) are valid. The delay elements ensure that a request to the next stage is asserted only when data is valid. The smaller Muller C-elements hold the return-to-zero transitions since the four-phase protocol acts on positive-going transitions allowing the use of edge-triggered storage elements. Before any request can be asserted, the bubble input of the Muller C-elements have to be zero which is accomplished with a master reset. The delays can be implemented by an inverter chain, to model a matched delay for the worst case in the associated logic of each pipeline stage. This is the simplest completion detection circuit for asynchronous designs with bundle-data protocol [2].

In [10] a dual latch asynchronous pipeline (DLAP) is proposed. Each stage is composed of a pair of storage elements; a master and a slave. New values can be put into masters while slaves retain previous values. The idea is shown on the right of Fig. 2. The controller circuit for each stage is composed of two Muller C-elements. Each master register is activated by the rising edge of L_m when a new value is ready (R_i is set), and the previous value has been moved to the slave. The rising edge of L_s activates the slave register when a new value is ready at the master and the previous value has been consumed by the following pipeline stage.

3. A single-pulse pipeline clocking technique: PP-pipelining

A PP-pipeline operation is analogous to the behavior of micropipelining. However, control signals are not based on Muller C-elements, but on simple synchronous state machines. Neighboring state machines cooperate to generate local clock signals to clock registers based on edge-triggered D-type flip-flops. Each state machine reacts to signals associated with neighboring stages, which could be asynchronous in nature, in order to generate the corresponding local clock control signals. The state machines are modelled on a common circuit for handling asynchronous events in synchronous designs known as a single pulser [18].

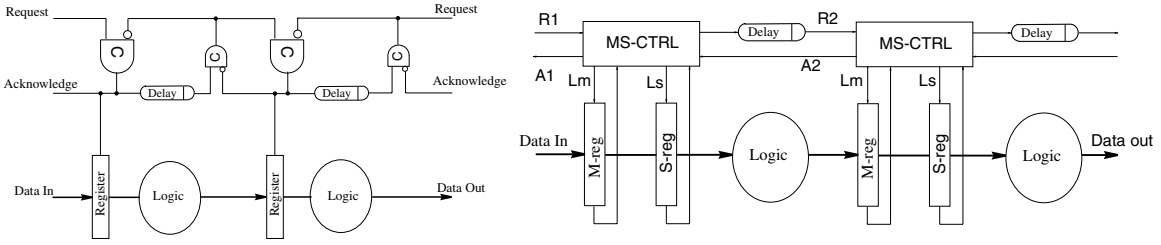


Fig. 2. Left: A two-stages four-phase micropipeline. Right: DLAP pipelining for two-stages.

3.1. General scheme of PP-pipeline

The PP-pipeline is basically a representation of the micropipelinig state rule specified as a state machine. Fig. 3 shows a PP-pipeline control for three stages, where the U_i (or PP-modules) correspond to state machines controlled by a global clock signal *clk*. The processing time associated with pipeline stages is modelled by delay lines; these are shown on top of each PP-module. Registers of a pipelined datapath are clocked by local pulses p_i, generated by individual PP-modules. At stage i, module U_i generates synchronously p_i only if both go_i and $done_i$ are asserted. After processing at stage $i-1$ (modelled by $delay_{i-1}$) $done_i$ will be active (asynchronously) indicating that data is ready to be delivered to stage i. Similarly go_i when asserted means that stage $i+1$ has accepted the data from stage i. U_i synchronizes these two events, at the first clock-edge after both signals are asserted U_i will generate a clock pulse on p_i. All the stages operate in parallel, accordingly the organization of interconnected PP-modules is referred to as a PP-controller. In synchronizing signals *done* and *go*, the PP-controller can be susceptible to metastable behavior [17]. We address this shortly but first consider the circuit operation.

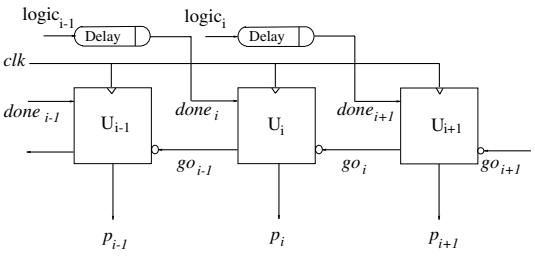


Fig. 3. PP-pipeline control for three stages pipeline.

Circuit operation: An Algorithmic State Machine Chart (ASM chart) description and a circuit for an individual PP-module is shown in Fig. 4. External events *done* and *go* are synchronized into *done.sync* and $go \cdot sync$, respectively. The state machine has two states: FIND and WAIT. FIND checks the condition $capt = done_i \cdot sync$ AND

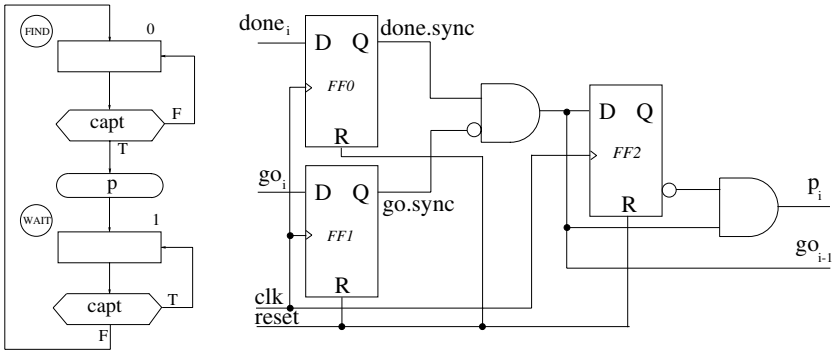


Fig. 4. An individual PP-module. Left: ASM chart specification. Right: A circuit implementation.

NOT($go_i \cdot sync$). The WAIT state loops until *capt* is false and a single pulse (of a clock period duration) is generated at p_i. A PP-controller of any number of stages requires initial conditions to the leftmost *done* and the rightmost *go* signals. These initial states are set to high and low, respectively.

As an illustration, consider the case in Fig. 3 when $done_{i-1}$ and go_{i+1} are forced high and low, respectively. From analysis of the circuit in Fig. 4, after a master reset, all the *go* signals go low and after the first clock edge a single pulse is generated only at p_{i-1}. The low-asserted level for *go* signals was chosen to make this operation simpler after a master reset. After the processing time at stage U_{i-1} ($delay_{i-1}$), $done_i$ goes high and it is synchronized by U_i (one clock edge). At the next clock edge (second one), a single pulse p_i is generated (for stage i to capture data from stage $i-1$). After go_{i-1} is synchronized and returns to low, stage U_{i-1} could capture new data. Taking *go* signals from *capt* instead of from *p* is like predicting the next-stage capturing and go_{i-1} occurs simultaneously with p_i thus saving one clock cycle in the PP-controller operation. A p_i event, in turn, triggers the delay element to generate $done_{i+1}$ and the operation is repeated. Different timing behaviors can result based on the relatives values of $delay_{i-1}$ and $delay_i$. The worst case is when $delay_{i-1} > delay_i$, and the pipelining runs at a period $T_{pipe} = (\lfloor delay_{i-1}/T_{clk} \rfloor + 3)T_{clk}$ where T_{clk} is the clock period of the frequency f at which the state machine runs. This is illustrated in Fig. 5 for the case of three interconnected PP-modules. The waveforms are generated when $delay_{i-1} = 23$ ns and $delay_i = 7$ ns, respectively. It is observed that the pipeline runs at a period of $T_{pipe} = 5T_{clk}$ as predicted by the formula given above.

Returning to possible metastable problems might arise when synchronizing *done* and *go* signals, observe that these can be avoided by eliminating the flip-flop to synchronize *go* (flip-flop FF1 in Fig. 4) and driving go_i directly from go_{i-1} of the subsequent PP-module. In fact, the risk of synchronizing *go* is essentially the risk of synchronizing *done* since go_{i-1} is a function of *done.sync*. The risk associated to *done* synchronization can be closely studied based on design and technology parameters of reliable synchronizer designs [17]. For practical cases, the delay lines can be safely replaced with a synchronous counter. This is suitable for one of the applications described in Section 4.

3.2. PP-pipeline with no delay lines

PP-modules can be further simplified by removing the delay lines shown in Fig. 3. This simplified version also eliminates the flip-flop to synchronize *go*. Essentially $done_i$ is directly taken from p_{i-1} and the PP-controller is acting as a gated clock [1]. The global clock is distributed to pipeline

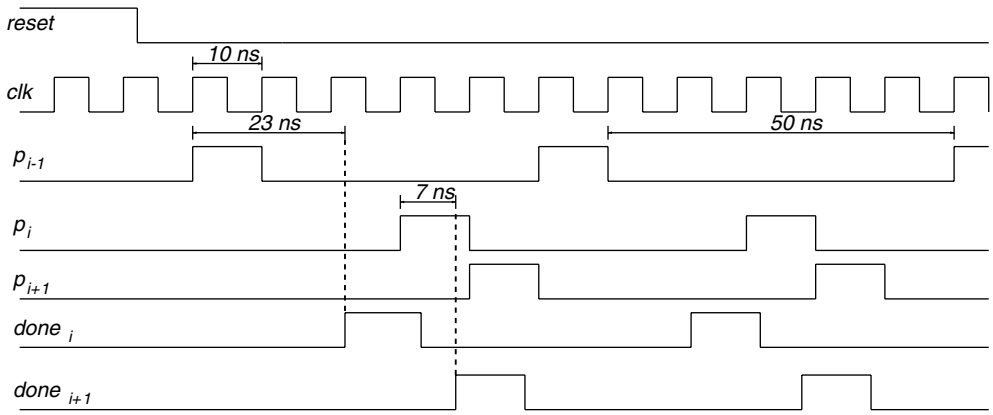


Fig. 5. Simulation waveforms with no associated processing logic when $delay_{i-1} = 23$ ns and $delay_i = 7$ ns for the case of three interconnected PP-modules.

stages only when there is new input data available. Additionally, the global clock load of gated clocks is distributed to the local PP-units and consequently the global clock to the PP-controller reduces the clock buffer demand.

Circuit operation: With this simplification, the generation of p_{i-1} will occur on the following edge of the clock after p_i will be generated, for all successive stages. Hence, the PP-controller operation replicates a totally synchronous pipelined design. The period for the pipeline frequency is $T_{pipe} = 2T_{clk}$.

Timing analysis: From timing analysis on the interconnection of PP-module circuits, it is easily shown that the clock period at which the controller can be clocked is $T_{clk} > 2t_{pd} + t_{co} + t_{su}$ where t_{pd} is the propagation delay of the AND gates while t_{co} and t_{su} is the clock-to-output delay and setup times of the flip-flops, respectively. This constraint assumes clock skew $\delta = 0$ and neglects the delays associated with the local interconnection wires in the figure. Very high clock frequencies can be obtained in FPGA technology and the time constraint does not appear to limit practical applications of pipelining when stage processing time is greater than $2t_{pd}$.

Clock skew effect: Clock skew may affect the relative generation of the clocking pulses for pipelining operation. Consider the two cases in Fig. 6 as an illustration. For the case of negative skew, shown on the left of Fig. 6, Regions 1 and 2 are identified for the relative generation of consecutive pulses p_{i-1} and p_i. Region 1 can vary from $2t_{pd}$ when $\delta > t_{pd}$ to $3t_{pd} - \delta$ when $\delta < t_{pd}$. Region 2 stretches from t_{pd} when $\delta > t_{pd}$ to $2t_{pd} - \delta$ when $\delta < t_{pd}$.

For positive skew, shown on the right of Fig. 6, Region 1 is fixed at $2t_{pd}$ while Region 2 can be extended by δ for $\delta < t_{co} + 2t_{pd}$. When $\delta = 0$ both regions are fixed at $2t_{pd}$. In both cases it is observed that clock skew will always generate a p_i event followed by a p_{i+1} event as required.

3.3. The PP-pipeline methodology

Next, a methodology for converting existing pipelined designs into equivalent PP-pipelined designs based on the PP-pipeline clocking technique is proposed. Assume a pipeline of n individual PP-modules (left of Fig. 7) are interconnected as shown on the right of Fig. 7 to form a PP-controller and all the n PP-modules are clocked from a

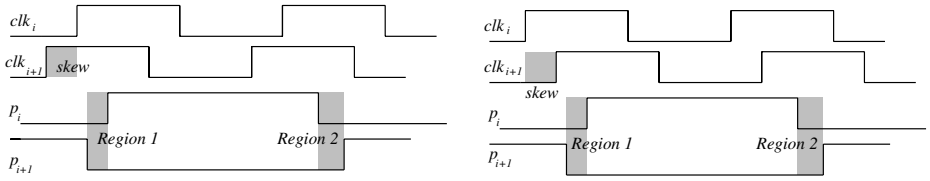


Fig. 6. Skew for two consecutive pulses generation. Left: Negative skew. Right: Positive skew.

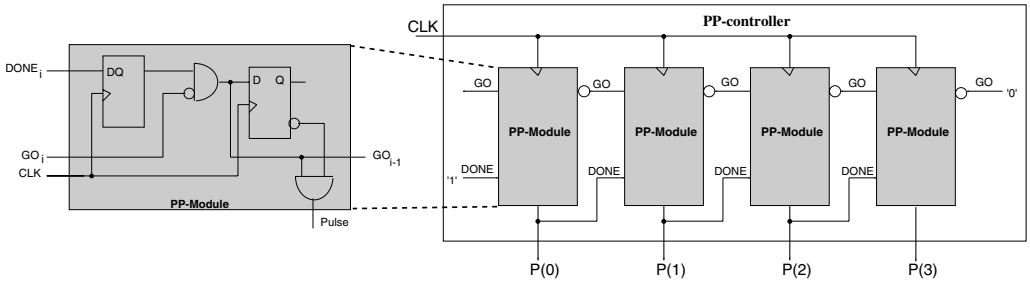


Fig. 7. Left: An individual PP-module. Right: A PP-controller. For simplicity a reset signal has been omitted.

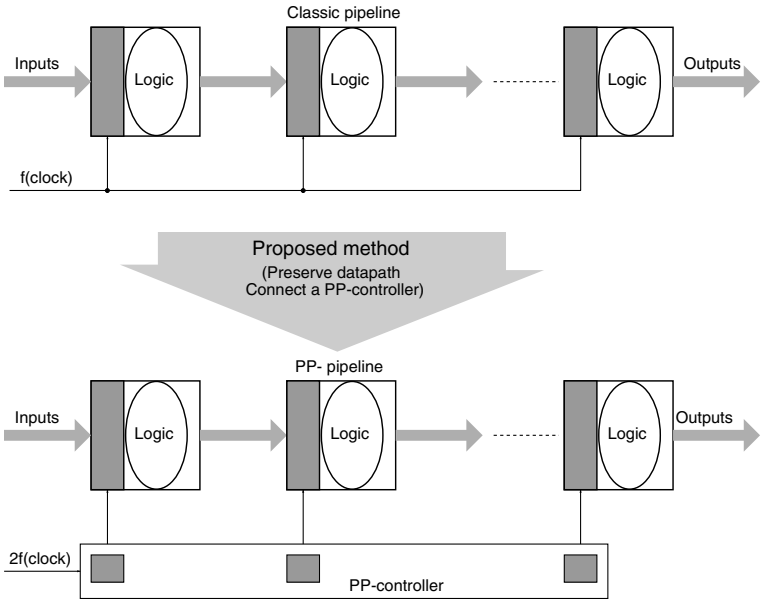


Fig. 8. A methodology to convert classic synchronous pipelined designs to a PP-pipelined design.

global clock f_{clk}. The PP-controller generates n local pulses which can be used to clock a n-stage pipeline. No modification is needed to the main existing pipelined datapath at the description level. Simply replace the feed into the global clock connection. The clocking mechanism being replaced can be either an asynchronous micropipelined such as the one shown on the right of Fig. 1 or a global clocked synchronous pipeline. For the latter case PP-controller pulses can be either directly clock pipeline registers based on edge-triggered flip-flops or be used as register enable signals. For an existing global clocked pipeline the frequency to the PP-controller is doubled to preserve the same data rate. As shown in Fig. 8 the conversion methodology is straightforward. For a micropipelined design the bundle constraint is $2T_{clk}$ and therefore f_{clk} can be adjusted accordingly to preserve the same pipeline data rate.

4. Potential applications

Since the proposed PP-pipeline technique and methodology is general it can be used to replace existing pipelined datapath either from an asynchronous nature such as a micropipeline or a globally synchronous one. All the registers in the PP-technique are synthesized with edge-triggered flip-flops common in FPGA devices. To illustrate the method consider the following applications. The first from a synchronous nature that can be further exploited in VLSI design. The second that can be used for asynchronous-like computation in FPGAs.

4.1. Clock-tree power reduction

Recent studies indicate that power consumption in the clock distribution tree of digital computers may account for up to 45% of the total integrated circuit power [9]. Consequently, the reduction of clock tree power is becoming crucial both in VLSI and FPGAs designs [14]. We have investigated the power consumption of the main clock tree of a Cordic Core pipelined design in a Xilinx Virtex2 FPGA using the PP-pipeline technique. A description of the study is as follows.

The Cordic Core: A 15-stage global clocked pipelined Cordic Core obtained from [8] computing

Sine and Cosine functions for input angles with 16 bit of precision.

Obtaining a PP-pipelined Cordic Core: The methodology presented in Section 3.3 was applied to translate the 15-stage global clocked pipelined Cordic Core. First, VHDL code was written for the PP-modules, and then captured as a parameterized structural object to act as a PP-controller. An equivalent PP-pipelined VHDL specification is straightforward after interconnecting the PP-controller to the original global clocked Cordic Core specification.

Power estimation process: The Xilinx XPower tool [19] was used to measure power for the global clocked and the PP-pipelined Cordic Core. Xpower computes power based on information of node switching rate activity of circuits. Switching rate activity was obtained from post place-and-route timing simulation data obtained from the simulator ModelSim XE 5.5b. Previously, both circuits were placed and routed with Xilinx ISE 5.1. Synthesis for the designs was obtained using the product Synplify Pro 7.2 targeting Virtex II devices. All the design entry specification was written in VHDL code.

Area and time results: The global clocked pipelined Cordic Core runs at 205 MHz taking 475 slices in a Virtex2 XC2V250. An equivalent PP-pipelined Cordic Core runs at 155 MHz and takes 488 slices in the same device. The area overhead of the PP-technique (mainly the PP-controller) for this simple design is less than 5%. For reference, post place-and-route timing simulations showed that a PP-controller of 16 outputs can run up to 324 MHz in a Virtex2 XC2V250 device taking 20 slices. This means the PP-technique allows a micropipelined design mapped into a FPGA synchronous design that would runs at over 160 MHz.

Clock tree power: A Cordic Core global clocked circuit running at 10 MHz reported a clock tree power consumption of 2.24 mW. An equivalent PP-pipelined Cordic Core circuit showed a global clock tree power consumption of 0.66 mW. This represents an overall reduction in dynamic power consumption of around 30% for the whole design. However, due to the limited number of dedicated clock lines of FPGAs it is expected that the power reduction capabilities of the PP-technique would be immediately better suited for VLSI designs.

Discussion: For the case when all delays are known as in the synthesis of global clocked pipeline designs it seems that the PP-pipeline technique has a potential for reducing clock tree power which would be beneficial in VLSI designs. For PP-pipelined designs converted from global clocked designs, the pulse signals can be used as register enable signals simplifying the exploitation of clock gating at the level of language description of existing designs.

4.2. Data-driven frequency modulation

A PP-controller generates a pulse signal at a stage p_{i+1} only after processing at stage i has completed (condition I) and no further p_i is generated until a pulse signal p_{i+1} has been generated (condition II). This ensures the correct forward operation of a pipeline flow. A PP-module circuit can accept as an input a signal indicating the completion of processing at a particular stage. This is modelled by means of a delay line. This delay line is placed between the connection from p_{i-1} to $done_i$. For this case, the pipeline runs at a period of $T_{pipe} = \max(\lceil delay_i/T_{clk} \rceil), T_{clk}$, where $delay_i$ is the completion time associated to any stage i and T_{clk} is the period of the global clock to the PP-controller. In circuit realization a synchronous reset counter can be incorporated to each pipeline stage that can be triggered by each pulse p to count processing time in integer intervals of T_{clk}. More elaborate data completion circuitry are available such as variable processing time computation driven by data values as in [4]. In either case, a PP-controller can manage instantaneous periods for moving data across the pipeline stages in discrete steps from $2T_{clk}$ to $\lceil delay_i/T_{clk} \rceil$. Consequently computation will be performed at a variable instantaneous frequency modulated by variable processing time of data at time of execution. A simulation to illustrate this behavior was carried out for a three stage pipeline. Stages two and three were stripped of processing logic while stage one was simulated to have a variable processing completion time using the Verilog construct:

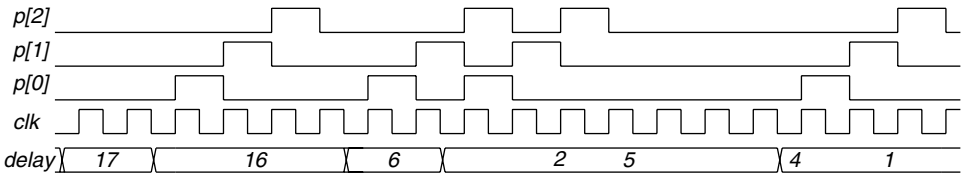


Fig. 9. Simulated waveforms of the PP-technique operation showing data completion at stage one and no processing at stages two and three. The local clocks are $p[0]$, $p[1]$ and $p[2]$, respectively while processing time *delay* of stage one is related to $p[0]$.

```
always @ (posedge p[0])
begin
delay = $dist_normal (seed, normal, sd);
t = 1'b0; #(delay) t = 1'b1;
end
```

Completion detection for stage one, `t`, is activated according to a delay time with normal distribution `delay` with standard deviation `sd`. The activation occurs after the rising edge of the clock pulse to stage one. Simulated waveforms are shown in Fig. 9 for $T_{clk} = 4$ ns. Note that when $delay = 25$ ns $(> 2T_{clk})$, $p[0]$ is generated with a period of $7T_{clk}$ complying with the given formula. It is also seen that the minimum period for $p[0]$ is $2T_{clk}$. If data completion circuitry with average-completion times implemented in FPGA hardware running around 50 MHz, frequency modulation can be obtained in discrete steps of less than 10%. The realization of these implementations is currently being investigated.

5. Conclusion

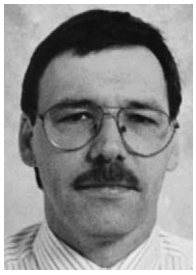
The PP-pipeline is introduced as a versatile pipeline clocking mechanism suitable for FPGA implementation. PP-pipeline can be used as a technique to migrate asynchronous pipelined designs such as micropipelining into FPGAs, and also as an alternative clocking mechanism to existing global synchronous pipelines. No redesign is needed to existing pipelined datapaths. PP-pipelined designs result in equivalent micropipelined designs with a fixed bundle constraint but using a synchronous methodology. A PP-pipeline synthesizes with circuit resources commonly available in commercial FPGAs. As alternatives to global clocked pipelined designs, equivalent PP-pipelined designs show lower power dissipation of the main clock tree and hence are also suitable for VLSI implementation. The technique can be extended to incorporate data-completion circuitry into a pipelined design using a synchronous approach. Simulations show that it is possible to handle variable data-completion times to modulate the instantaneous frequency in discrete time steps across the pipeline stages. The time of the discrete steps is in practice small compared to coarse-grain combinational logic of typical pipeline stages. These advantages are due to a regular PP-controller based on simple cooperating state machines.

References

[1] L. Benini, P. Siegel, G. DeMicheli, Designing for low power circuits: practical recipes, IEEE Circuits and Systems magazine 1 (1) (2001) 6–25.
[2] F.C. Cheng, Practical design and performance evaluation of completion detection circuits, in: Int. Conf. on Computer Design, ICCD, October 1998.
[3] A. Davis, S. Nowick, An introduction to asynchronous system design, University of Utah, Report N. UUCS-97-013, Salt Lake City, 1997.
[4] A.D. Gloria, M. Olivieri, Completion-detection carry select addition, IEE Proceedings—Computer Digital Techniques 147 (2) (2000) 93–100.
[5] S. Hauck, S. Burns, G. Borriello, C. Ebeling, Montage: an FPGA for synchronous and asynchronus circuits, in: 2nd Int. Workshop on Field-Programmable Gate Arrays, August 1992.
[6] S. Hauck, S. Burns, G. Borriello, C. Ebeling, An FPGA for implementing asynchronus systems, IEEE Design & Test of Computers 11 (3) (1994) 60–69.
[7] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, San Francisco, 1995.

[8] R. Herveille, Cordic core specification, rev. 04, www.opencores.com, December 2001.
[9] T.A. Johnson, I.S. Kourtev, A single latch, high speed double-edge triggered flip-flop (DETFF), in: Proc. of 8th IEEE International Conference on Electronics, Circuits and Systems, ICECS, June 2001.
[10] R. Kol, R. Ginosar, A doubly-latched asynchronous pipeline, in: Proc. of the Int. Conf. on Computer Design, ICCD'97, 1997.
[11] C.L. Seitz, System timing, in: C.A. Mead, L.A. Conway (Eds.), Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1980, Chapter 7.
[12] D.A. Patterson, J.L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, McGraw-Hill, New York, 1995.
[13] R. Payne, Self-timed FPGA systems, in: Proc. of the 5th Int. Workshop on Field Programmable Logic and Applications, LNCS 975, September 1995.
[14] M.P. Qing, X. Wu, A new design for double edge triggered flip-flops, in: Proc. of the Asia and South Pacific Design Automation Conference, Febuary 1998, pp. 417–421.
[15] M. Shams, J.C. Ebergen, Optimizing CMOS implementations of the C-element, in: Proc. of the Int. Conf. on Computer Design, ICCD'97, 1997.
[16] I.E. Sutherland, Micropipelines, Communications of the ACM 32 (6) (1989) 720–738.
[17] J.F. Wakerly, Digital Design: Principles and Practices, Prentice-Hall, New Jersey, USA, 2000.
[18] D. Winkel, F. Prosser, The Art of Digital Design: An Introduction to Top–Down Design, Prentice-Hall, Englewood Cliffs, 1980.
[19] Xilinx, XPower Tutorial, FPGA Design, Xilinx, Inc, San Jose, California, 2002.
[20] K.Y. Yun, P.A. Beerel, J. Arceo, High-performance asynchronous pipeline circuits, in: Proc. of the Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, March 1996, pp. 17–28.

Oswaldo Cadenas is a lecturer in the University of Reading's Electronic Engineering Department and Computer Science Department. His research interests include reconfigurable logic, hardware compilation and computer architecture. Cadenas received a PhD in Computer Science from the University of Reading.



Graham Megson is a professor in the Department of Computer Science in the University of Reading. Professor Megson has published over 160 papers at international conferences, in journals, including five books on algorithms/architectures and related topics.