Pipelining on FPGAs: A Tutorial

Eduardo Boemo
Universidad Autónoma de Madrid
Madrid, Spain
eduardo.boemo@uam.es

***Abstract*— This tutorial reviews historical milestones and main concepts regarding the pipelining of electronic circuits. Although the technique emerged in the 1960s, it remains a direct way to simultaneously increase throughput and reduce power in FPGA-based systems. However, the efficacy of pipelining is limited by the dominance of register and routing delays. This work focuses on bit-level pipelining. It analyses by examples keys aspects such as construction hints, pipeline metrics, effects of registering, preferential pipeline directions, and synchronization failures. The text condenses the first section of the invited tutorial lecture at the 2019 Southern Conference on Programmable Logic (SPL). Whenever is possible, numeric examples are particularized to FPGA technology, but in some cases, cell-based ASICs data are deemed more convenient. The ideas would be useful for students of an advanced course on digital electronics, or PhD candidates interested in the details of the design of integrated circuits.**

Keywords—Pipeline, FPGAs, High-speed digital design, Henry Ford, Clock skew, Wave pipeline, Power consumption.

I. Introduction

Main concepts of pipelining originated on the assembly line of the T-Model in 1908. In that year, production in the Ford plant was reorganized by dividing the construction of the cars between groups of workers who specialized in only one part of the process. Each team was placed along lines, repeating the assigned task – always the same – on successive pieces situated on a conveyor belt (Fig.1). In this scheme, the maximum manufacturing speed is limited by the slowest task. As a consequence, these tasks must be planned to require the same time. The flow of parts is continuous, except for the inevitable time needed to fill or empty the conveyor belts. Henry Ford eliminated this inconvenience by introducing an 8-hour working day, with 3 shifts a day. Therefore, the steady-state was virtually infinite and all the workers operated in parallel making simultaneously the same part of the assembly, but on different cars. This type of process is now called temporal parallelism. Using all these concepts, Ford sped up chassis assembly from 12.5 to 1.5 hours making nearly 15 million cars between 1908 and 1927 [1]. The success of the process converted the T-model into a milestone of mass production history, as well as an agent of urban and social revolution.

Henry Ford´s ideas to improve production are also valid for digital circuits. Large combinational blocks are unable to process data at high-speed rates. Normally, new data cannot be imputed until the previous result is outputted. If this rule is violated, the fastest bits of the next data reach the slowest bits of the previous one. Each line of gates only works during a short fraction of the processing period [2]. Like in an unorganized car production, most of the time the gates are waiting for the arrival of new data.

Nevertheless, the main difference between car production and digital processing is the matter of the parts. Mechanical objects must be artificially moved while bits must be artificially stopped. On the contrary, they travel along the circuit until the electric potential equilibrium is reached. Thus, digital pipelines require the insertion of edge-triggered D-type flip-flops (FFs) in the datapath in order to align and stop the intermediate results. These FFs make it possible to synchronize the operations, but they do not contribute to any transformation of the data, just as conveyor belts do not assemble the cars. Like in the car production, in digital pipelines the slower stage fixes the system period. To maximize pipeline speed, the circuit partition must be balanced. A direct option is to make identical the processing elements. In the 1980s, this type of circuit was renamed a "systolic arrays" [3]. However, once again the beneficial effect of uniformity had already been discovered by Ford. The T-model was originally built in black, red, green or grey. But in the following years, the colour choice was reduced to only one. "*Any colour so long as it's black*", said Ford [4]. Fortunately, in electronics there is a myriad of digital circuits based on identical blocks: they were designed to be extensible and cascadable. Even today, these features facilitate circuit description, construction, width extension, and debugging.

Fig.1: Workers on a T-model moving assembly line with magnetos and flywheels in 1913. Reproduced from Wikicommons [5].

Probably the first published study about pipelining in digital circuits is the work of Leonard Cotten [6]. But even in the 1960s, the origin of pipelining was uncertain. Cotten writes: *"…The term pipeline has been used for over 5 years by designers to describe maximal rate processing …. However, the author has so far been unsuccessful in determining the origin of the term as used in this context…"*. A second paper of Cotten [7] studied the effect of clock skew (Section VII of this tutorial), and the wave pipeline alternative. The IBM 360 FPU was an early materialization of the technique [8]. At that time, the term was quite novel as they used "*pipelined*", in quotation marks [8],[9]. Pipeline adders and multipliers were explored in [10]. After that, notable pipeline implementations were [11]-[14].

This work has been supported by Comunidad Autónoma de Madrid (Spain) DIFRAGEOS Project S2013/ICE-3004.

978-1-7281-1363-0/19/$31.00 ©2019 European Union

II. A SIMPLE EXAMPLE: PIPELINING A RIPPLE-CARRY ADDER

The extremely well-known ripple-carry adder (RCA) is adequate for illustrating the effects of pipelining. This circuit suffers from a large serial delay caused by the propagation of the carries. Even so, nowadays it is one of the fastest options for addition in FPGA technology. The cause is the low fanout of its nets.

Fig.2 shows an 8-bit RCA composed of identical full-adders (FAs). Unusually in digital design, it is drawn with the input data traveling from bottom to top, and from right to the left. It reflects the way in which the calculation is done by hand.

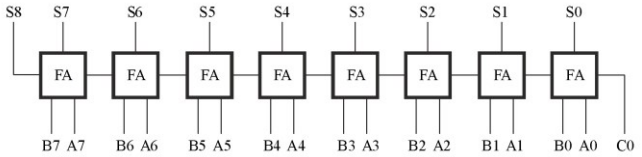

Fig.2: 8-bit ripple-carry adder (RCA).

A fine-grain pipeline version of an RCA is obtained by adding 117 FFs (Fig.3). As is traditional in pipeline schematics, an FF (and its associated clock and reset lines) is represented as a small square or dot [15], [16]. At the input, a triangular arrangement of FFs are utilized to delays each datum to synchronize it with the corresponding carry bits. At the output, another triangular arrangement of FFs delays each result to synchronize it with slower S8 and S7 bits. These triangles are named skewing and deskewing registers in [12].

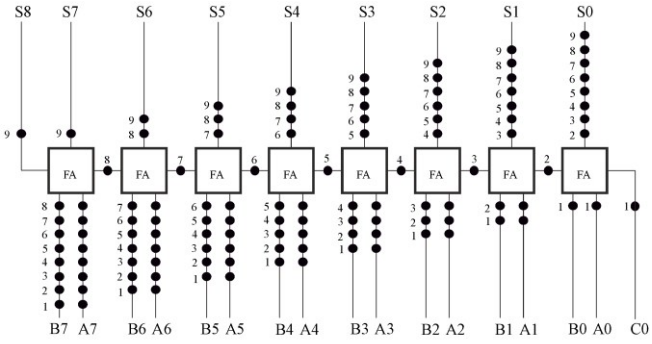

Fig.3: 8-bit RCA pipeline version.

Fig.3 illustrates an obvious property of pipelines: the number of FFs along any I/O path is a constant number (9 in this example, including the I/O registering). A path with more (or less) FFs cannot exist as this would lead to the mixing of bits of different data.

The flow of bits after each clock edge is shown in Fig.4. Only the first clock cycles are shown. The second index indicates the position of successive data (A02 is the bit A0 of the second data, and so on). Initially, the pipeline outputs spurious additions. For example, the first bit S01 (of the first addition) is obtained after the 2nd clock cycle, but the corresponding last bit S81 of the same result is outputted after the 9th cycle. So, it is necessary to delay the least significant bits to maintain the parallel format. But unlike the combinational version, after filling the pipeline all the gates operate completely in parallel.

III. EVIDENCING DEPENDENCIES

In order to pipeline a circuit it is better to redraw it to evidence the dependencies between processors (B depends on A, if B needs the results of A to begin its calculations).

A single processor does the same job in different data over time (e.g. S0 calculation)

Clock	FA1	FA2	FA3	FA4	FA5	FA6	FA7	FA8
1	A01 B01 C01	X	X	X	X	X	X	X
2	A02 B02 C02	A11 B11 C11	X	X	X	X	X	X
3	A03 B03 C03	A12 B12 C12	A21 B21 C21	X	X	X	X	X
4	A04 B04 C04	A13 B13 C13	A22 B22 C22	A31 B31 C21	X	X	X	X
5	A05 B05 C05	A14 B14 C14	A23 B23 C13	A32 B32 C22	A41 B41 C41	X	X	X
6	A06 B06 C06	A15 B15 C05	A24 B24 C24	A33 B33 C13	A42 B42 C42	A51 B51 C51	X	X
7	A07 B07 C07	A16 B16 C16	A25 B25 C25	A34 B34 C34	A43 B43 C43	A52 B52 C52	A61 B61 C61	X
8	A08 B08 C08	A17 B17 C17	A26 B26 C26	A35 B35 C35	A44 B44 C44	A53 B53 C53	A62 B62 C62	A71 B71 C71

After filling the pipeline, all the processors work in parallel.

Fig.4: Details of the computation at each RCA pipeline stage.

The procedure for evidencing dependences is simple (Fig.5): the FA that first starts to operate is positioned at the bottom of the circuit, then the second, and so on.

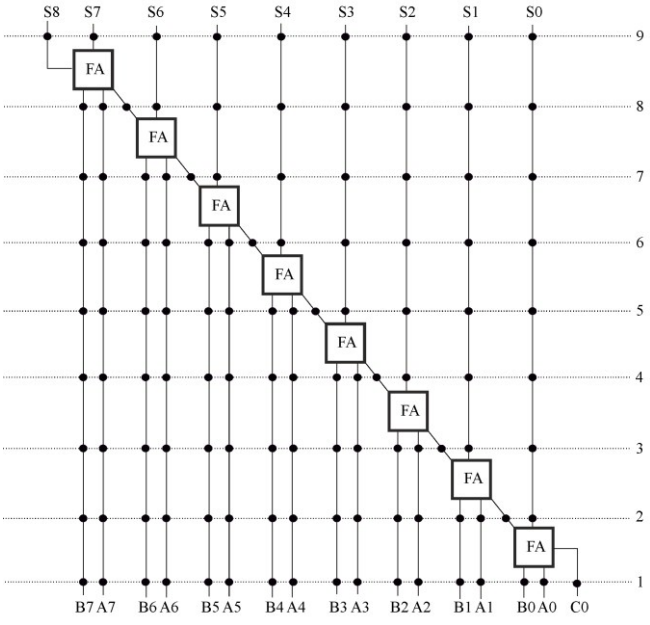

Fig.5: RCA topology evidencing dependencies.

Naturally, the resulting structure is identical to the represented by the Fig.2, but now pipelining is straightforward. The intersection between the horizontal lines and the wires indicates the places where the synchronization FFs must be inserted (supposing that all the FA blocks and wiring have the same delay).

Fig.5 confirms the first drawback of pipelining: the area overhead. For example, in standard cell technology [17] both full-adder and D-type FF (with reset) have a minimum of 28 transistors each. So, during the pipeline of the 8-bit RCA of Fig.4, the total number of transistors passes from 224 to 3500. Moreover, as the number of FF grows as $NFF(n) = 1.5\ n^2 + 2.5\ n + 1$, (for a input data width n), a hypothetical 1024-bit fine-grain pipeline RCA would surpass 1.5 times the maximum 1,095,200 available FFs in a state-of-the-art Virtex chip [18]. These numbers explain why FPGAs architects reuse the FFs of the configuration path to get additional chains of registers, called SRL16, 32, etc. [19]. In any case, such a huge pipeline would require tens of amperes to raise the clock edge along the circuit in few nanoseconds.

Finally, Fig.6 shows another example of pipelining. Drawing the "cubic" processors (above) according to its data dependencies (below) makes the solution direct: a 36-register and five-stage pipeline. Processing elements number 2, 3 and 4 do not exchange data between each other; so, they can operate in parallel. The same situation occurs with processing elements 6 and 7.

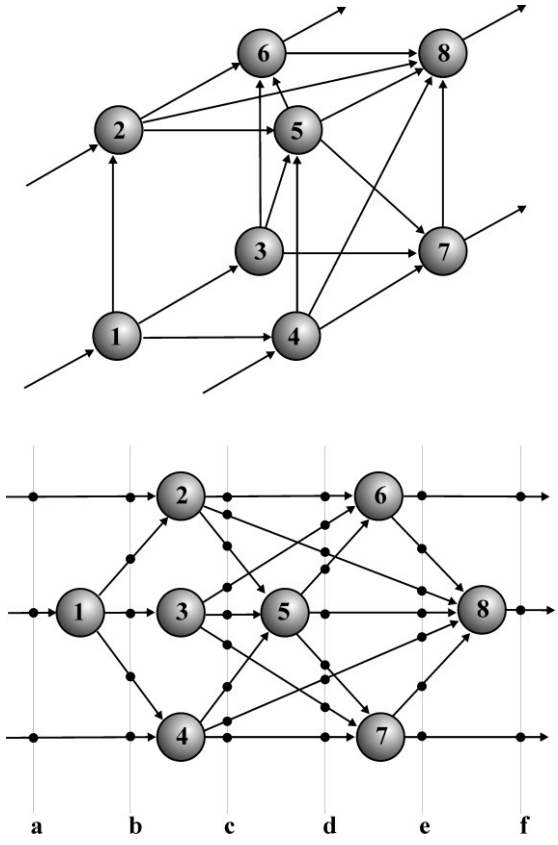

Fig.6: Redrawing a circuit (above) to evidence its data dependencies (below).

The visualization of the dependencies also facilitates the trimming of the pipeline. In those cases where the delays of the processing elements are different, some lines can be eliminated, without affecting the speed of the circuit. For example, if the delay of PE1 is 200 ns while the other PEs has a delay of 100 ns, the *c* and *e* lines can be eliminated.

IV. Pipelining in numbers

The result of pipelining can be described by several numbers: throughput, latency, speed-up, area penalty, pipeline granularity, and logic depth among others. Pipelining does not reduce the time required to obtain an individual result; but increases the number of obtained results per second. R. F. Lyon describes a pipeline as a circuit *"which has an operation period less than its operation delay"* [20]. This phrase condenses the two main numbers of a pipeline: throughput and latency.

The Oxford dictionary states throughput as *"the amount of material or items passing through a system or process"*. In electronic pipelines, it can be adapted as the number of results per second [21], or simply the processing rate. Other key terms are bandwidth or production, as well as its inverse magnitude, the pipeline period. The annual production of the T-model reached in 1914 the number of 260,720 units [22]. That is, a throughput of nearly a car every 2 minutes.

The latency is the time necessary to process a single piece of data. It also called the delay or response time. Latency is masterfully defined by Peter Cappello as "the amount of time between the first-bit-of-the-first-data entrance and the last-bit-of-the-last-data output, for a single (just one) computation" [23]. However, sometimes the latency of a block is specified as the time between first input bit and first output bit of a single piece of data.

The Boing Company is a useful illustration of the above concepts, even considering that apply both spatial parallelism (assembly lines in parallel) and temporal parallelism (pipelines). Nowadays, Boeing produces a 777-model plane every 3 days (throughput = 0.33 planes/day). Naturally, a single plane cannot be assembled in 3 days. It is composed of more than 3 million parts, and has approximately 60,000 rivets [24]. To determine the latency of the process it would only be necessary to time any of those rivets from when it enters the factory to when it leaves as part of the plane at the other end of the factory.

The effectiveness of pipelining in terms of time is measured using the speedup figure. This is the throughput of the pipelined version divided by the throughput of the original circuit. In return, this extra speed increases the circuit cost. The area penalty is the pipeline area divided by the original circuit area.

A more useful concept is the pipeline granularity β [25]. This is the maximum number of processors operating in series between successive lines of FFs. In a regular pipeline, granularity is the key to tuning the result, trading speed for extra FFs. For example, if lines 2, 4, 6, and 8 are removed in the circuit of Fig.4, the granularity is β=2, the FF count passes from 117 to 65 but the minimum clock period is greater than the delay of 2 FAs. Now it is necessary to wait for the results of two FAs. Another area-time pair is obtained using β=4, by removing lines 2, 3, 4, 6, 7, and 8 Fig.4. Now, the number of FFs is 38 and the minimum clock period must be greater than the delay of 4 FAs. The idea is illustrated in Fig.7.

Granularity is a topological parameter. At silicon level, its equivalence (or consequence) is the logic depth. In FPGA technology, logic depth can be considered as the maximum number of LUTs in series between successive lines of FFs. A fine-grain pipelined RCA has a granularity β=1 but the logic depth can be 1, 2 or more LUTs, depending on diverse parameters such as the LUT size, dedicated XOR gates and carry-chain lines, routing congestion criteria, or the ability of the synthesis tool. In his remarkable book, H. B. Bakoglu includes the gates of the input FF in the calculus of the logic depth, for masked integrated circuits [26].

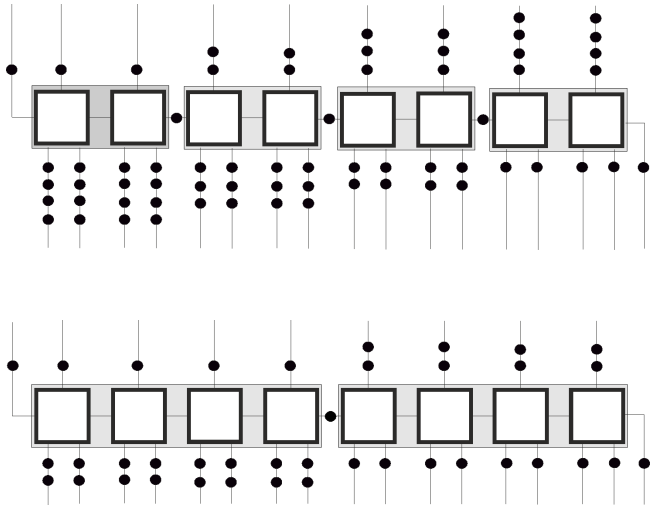

Fig.7: RCA pipelined with β=2 (above) and β=4 (below).

V. The Effect of Registering and Extra Wiring

The followers of Henry Ford in the field of electronics shared his frustration about pipelining. The N-fold gain in speed is a myth; it is only possible if the wiring and FF delays are insignificant in comparison with logic delays. In other words, fine-grain pipelines have a relatively slow processing speed limit.

The organization of a pipelined production requires several blocks receiving each datum at the right time. Even though the nature of bits and car parts is different, the consequences of introducing mechanisms to synchronize them lead to the same result: loss of time. Ford divided the motor assembly into 48 operations (N=48). This arrangement should lead in the abstract to a speedup to nearly 48. However, he merely obtained a speedup of 3. In the same way, the construction of the magneto was split into 29 parts, allowing the time to be reduced only from 20' to 13'10" (speedup=1.5). Finally, the overall car assembly evolved from 12'30 hours to only 1'33 (speedup=8.3) [27].

In current integrated circuits, the effect of transporting (wiring) and synchronizing (FFs) the parts (bits) is expensive in term of time. FFs and wiring delays are larger than combinatorial logic delays. This fact destroys the magic effect of pipelining: dividing the task in N concurrent blocks never produces the theoretical speedup of N. For example, in Fig.7 a circuit is shown with a total combinational delay of value Δ_{COMB}. The propagation delay and setup of the FFs are labelled $\Delta_{CK\text{-}OUT}$ and Δ_{SETUP} respectively. The pipelining of the block in N stages, effectively diminish the combinational delay of each stage by N, but the delays associated to the FF remain constant. Additionally, a wiring delay ΔW must be also computed. Thus, the pipeline period is:

$$T \geq \Delta_{CK\text{-}OUT} + \Delta_{COMB} / N + \Delta_{SETUP} + \Delta_W + SKEW \qquad (1)$$

The effect of the clock skew in Eq.1 is analysed in Section VIII. In any case, for a large number N of stages, the clock period will remain dominated by FF and wiring delays.

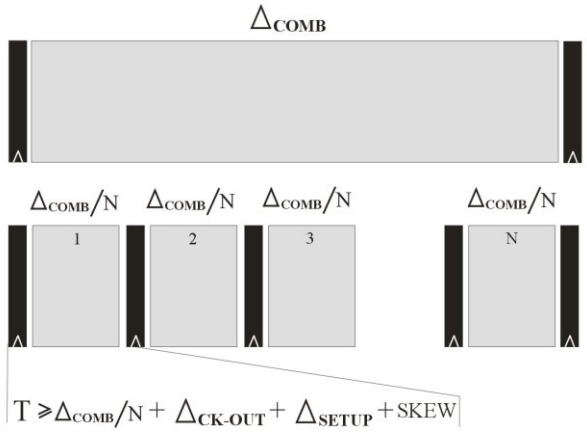

Fig.8: The expansion of the stage delay.

Fig.9 shows the relationship between FF delays ($\Delta_{CK\text{-}OUT}$ + Δ_{SETUP}) and minimum Δ_{COMB} for ten different technologies [28]-[38] ranging from discrete logic to FPGAs. The T_{ILO} parameter of Xilinx datasheets was taken as the minimum combinational delay for LUT-based circuits, while the two-input NAND was selected for discrete logic. In any case, the period of a maximum fine grain-pipeline would be practically fixed by FFs and wires. Even in high-performance computers, the logic depth cannot be just one gate or LUT. For example, the CRAY models 1, 2 and 3 had a logic depth of 8, 4 and 6 levels respectively [26].

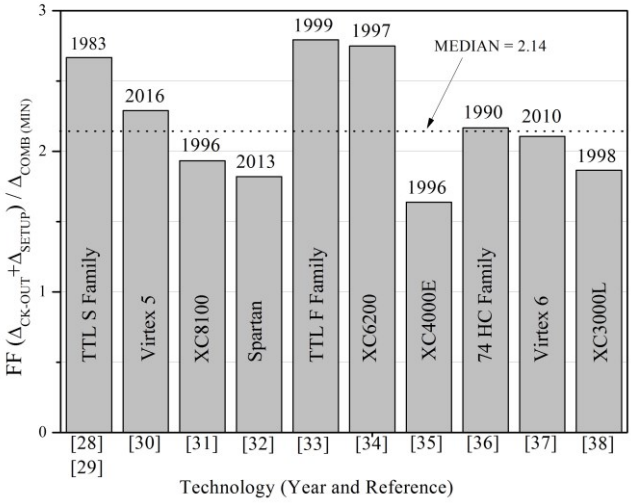

Fig.9: Ratio between the total FF delay (propagation + setup) and minimum combinational delay for different technologies.

As result, pipelining leads to a paradox. It is applied to avoid inactive gates while waiting for new data (as it happens in a combinational circuit). But for fine-grain pipelining, only 20-30 % of the period is involved in the data

processing. The rest of the time is necessary to synchronize and transport the bits. After all, the gates still remain inactive most of the time. But at least, they work in parallel.

Meanwhile the delays of a FF are well-characterized numbers; the nature of wiring delay is more complicated. The wiring distribution depends on the number of stages N. Pipelining expands the number of wires, changes the fanout of the nets, and modifies the wiring distribution delay itself. Some aspects that pipeline designers must take into account about $\Delta_W(N)$ are:

- Some pipeline directions change the nature of input data wiring, passing from heavily loaded global or broadcasted lines to lines with a minimum fanout. This point is explained in the next section.
- Pipelining usually improves the circuit routability and reduces wiring congestion in FPGAs. In a pipeline circuit, FFs that drive another FF are the most common structure of the circuit. So that, from all the pins associated to a FPGA logic elements (a k-LUT plus the associated FF), only two are utilized: the input and output pins of the FF.
- In any well-routed circuit, the wiring histogram follows a Pareto-Levi distribution [39]. That is, there are lots of wires with low delays and few with the highest delay values. In a combinational circuit, the worst wire not always is part of the worst (critical) path. But in a balanced pipeline, all paths are equally critical for the minimum clock period. Normally, the worse wire nests within the worse stage.

VI. Global Lines and Directions of Pipelining

The word pipeline evokes water and tubes. If friction is neglected, the speed of the water is independent of the pipe length [15]. However, on digital pipelines the circuit size imposes a speed limit. Clock skew and heavily loaded global lines increase the pipeline size. In this section, the second effect is illustrated.

In [40], Jump and Ahuja analysed the different directions of pipelining for array multipliers. The idea is illustrated in Figs.10 and 11. The array of Fig.10 has a typical structure of communication. The horizontal data enter in just one processor. There are local communications, exhibiting a low-fanout that is almost independent of the array size. In contrast, the vertical data are broadcasted: each one requires a global line to reach a complete column of processors. The fanout of the vertical global wires is a function of the array size. This type of mixed communication is common in binary multipliers where a column (or a row) of AND gates concurrently calculates, for example, the partial products *A0B0*, *A0B1*, *A0B2* …, *A0Bn*. The signal *A0* requires a global line to reach all the ANDs, meanwhile any *Bi* is local [41].

Fig.10 shows other valid directions of pipelining (there are other possible "angles" but they do not maximize speed). From a topological point of view, neglecting wiring delays, both pipeline directions have a granularity of one processor. So, in the abstract they should reach the same speed. The "vertical" pipeline option sounds better because it exhibits a less latency and smaller number of FFs. But the situation is different in actual integrated circuits were wiring delay is dominant: for large array sizes, these global lines will exhibit high delays.

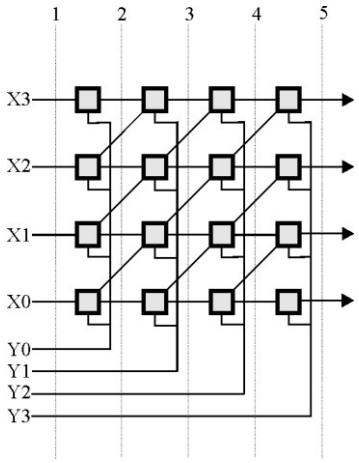

Fig.10: Generic pipeline with global lines inside each stage.

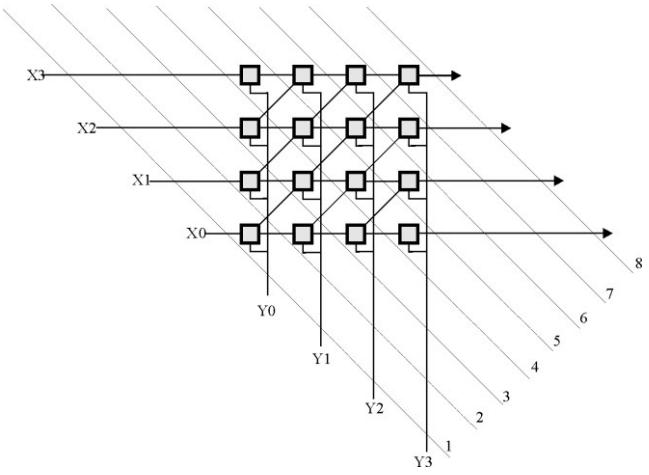

Fig.11: Pipeline direction disrupting global lines.

The vertical line pipeline confines each of these global lines inside a pipeline stage. So, the total delays of these heavily loaded interconnections are a part of the clock period. In contrast, the "45 degrees" pipeline breaks the global lines, transforming them into almost local wires. The effect is detailed in Fig.12 for Y0 signal as example.

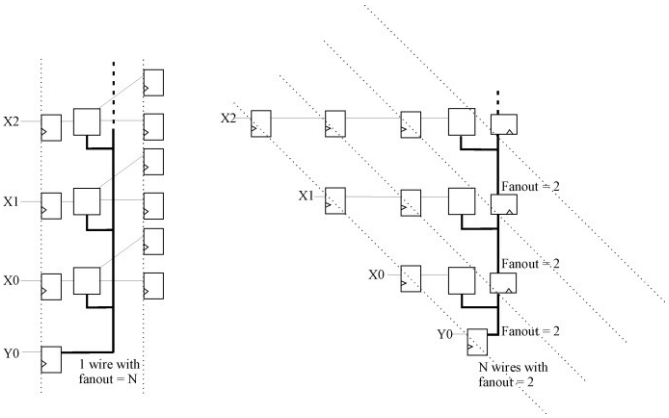

Fig.12: Elimination of global lines by pipelining (detail).

Work [42] carries out a case-study of the transformation of global lines by the direction of pipelining. Target technology was 1µ CMOS Standard Cells from the former ES2 foundry [43], and Xilinx FPGAs. Two 16-bit arrays multipliers were compared: the Hatamian-Cash [12] and the McCanny-McWhirter [44]. Both pipeline circuits share the same topology but the first maintain global interconnection inside each stage, meanwhile the second transforms these interconnections into a set of local wires with fanout equal to 2.

Fig.13 shows the histogram of wiring capacitance for each version: Hatamian-Cash (above) and McCanny-McWhirter (below). In both graphs there are two similar groups of wires. On the left, there are a high number of local interconnections loaded with small capacitances. On the right side, a set of heavily loaded lines corresponding to the clock and global reset signals. The delay of theses clock branches does not directly affect the period. Its maximum difference (skew) only increases the pipeline period (Section VII).

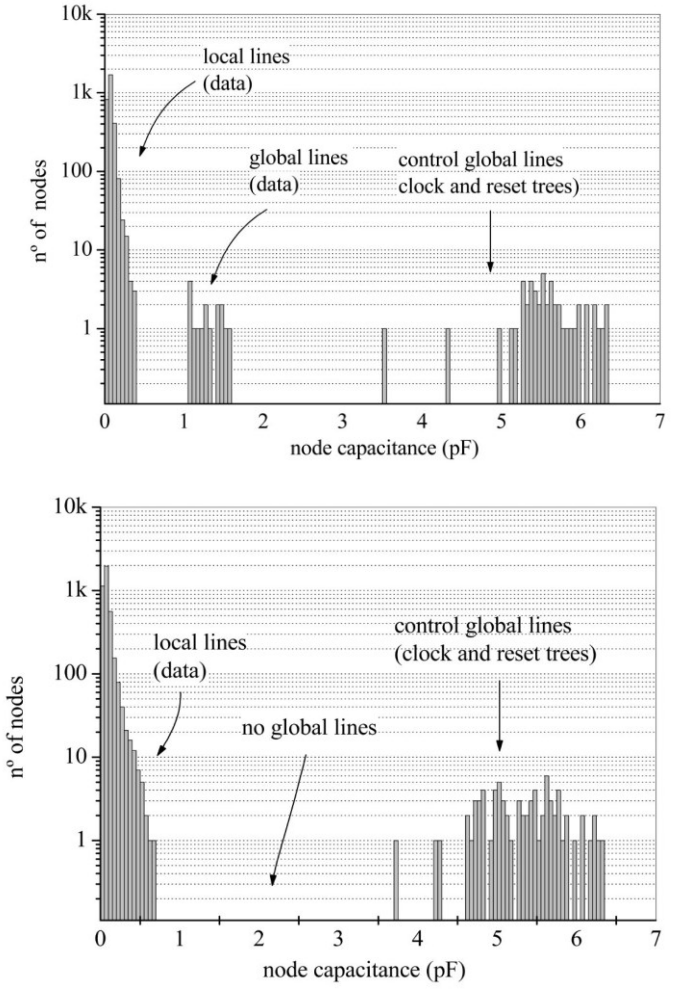

Fig.13: Histograms of wiring capacitance.

The Hatamian-Cash pipeline exhibits less datapath nodes (3115 versus 3986). But there is in the middle a group of global lines of data that affect the clock period. These lines do not exist in the fully local-line pipeline of McCanny-McWhirter. As result, its throughput for typical delays is higher (154 versus 117 MOPS).

VII. Synchronization Failures in Pipelines

In the previous sections, it was shown that fine grain pipelining easily increases the total number of FFs from a few units to tens of thousands. A large number of FFs makes a clock distribution tree indispensable to drive them synchronously. And clock trees generate latency and skew. The first effect is mitigated in FPGA technology by adding digital PLLs to align external and internal clock edges. More problematic is the clock skew: almost all the pipeline´s paths are vulnerable to it.

The clock skew is the maximum difference in time between the same clock edge at two points of the die. The factors that contribute to clock skew even in balanced trees are differences in parameters like wiring length, distributed RC, local temperature and voltage, FF trigger thresholds, and finally buffer and FFs propagation delays. The mixture of all these components makes skew inevitable.

Fig.13 clarifies the effect of clock skew. In a single-phase clock scheme, each clock edge initiates a race between the data in D1 and D2. To work properly, the data in D2 must be captured by the FF2 before being replaced by the data that travels from D1. Moreover, after the arrival of the edge to FF2, the previous data in D2 must still remain stable during the hold time of the FF.

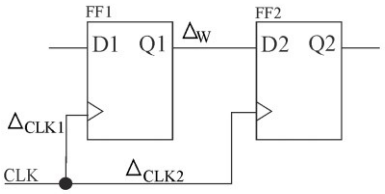

Fig.14: FFs vulnerable to clock skew in single-phase clocking.

As a consequence, for the worst case (wiring delay zero), the maximum admissible clock skew is:

$$SKEW = \Delta_{CLK2} - \Delta_{CLK1} < \Delta_{CK\text{-}OUT} - HOLD \qquad (2)$$

As a rule-of-thumb, chip designers know that clock skew must always be lower than the FF propagation delay. This problem was called double-clocking by Fishburn [45] because one bit passes through two FFs in one clock edge. It is also known as a short-path fault. That is, not only the longest path generates problems in digital systems. An important fact is that the clock period is not present in (2). If a circuit suffers clock skew, the circuit will never work at any frequency. Another aspect to consider is the sign of the skew. If it has an opposite sign, the clock edge arrives first at FF2, and the risk of double-clocking is reduced. Eq. 2 is completely applicable to pipeline circuits where FF chains are very common. For example, in the 8-bit fine grain pipeline RCA of Fig.4, the 77 % of the FFs only drive another FF.

The second source of synchronization failure is more evident. It is called a long-path fault and states that the minimum clock period must be larger than the delay of worst pipeline stage. In [46] was proposed to calculate the clock period using a circular pipeline, in order to include the I/O pin delays in the computation of the period. The condition to avoid long path fault is indicated in (1). The worst effect of the skew occurs if the slower clock line triggers the input FF

and the faster clock line triggers the output FF. In such a particular combination, the value of the skew is added to the clock period.

VIII. Conclusions

This tutorial reviewed the main aspects of the pipelining of digital circuits. Students interested in integrated circuits can discover several points of research interest. However, some important related issues like wave pipelining, the relationship between pipeline and power consumption, and the self-timed synchronization exceed the length available for this work.

If pipelining is a masterpiece of the classical period of digital electronics, wave pipelining (WP) is an example of the baroque period. Both techniques were contemporary. WP was summarized in [8] in 1967: *"…If a section of combinatorial logic, such as the logic to execute an add, could be designed with equal delay in all parallel paths through the logic, the rate at which new inputs could enter this section of logic would be independent of the total delay through the logic…"*. Leonard Cotten describes the same concept with different words in 1969: *"…It is possible for max-rate pipeline machines to operate at high rates determined by path differences, rather than the conventional maximum delay..."* [7]. The WP technique speeds up a circuit without using intermediate FFs. In a WP all the paths are equalized; therefore, several "waves" of data can propagate along the circuit without interference between the fastest bit of a new data and the slowest bit of the previous one. The equalization must be immune to temperature and voltage variations. WP allows the designer to obtain a unique combination of fine-grain pipeline speed and the latency of the original combinational circuit. The technique was studied in detail in [47], [48]. An example of wave pipeline in an LUT-based FPGA is described in [49], [50].

Another important aspect of pipelining is its hidden relationship with power consumption. The fact that pipelining can reduce power defies common sense. This is especially true in FPGA technology, but negligible in standard cell devices [51]. In a pipeline, the intermediate lines of FFs prevent the propagation of glitches that, otherwise, would produce a snowball effect in the activity of large combinational circuits. If the synchronization power overhead caused by the extra FFs is less than the datapath glitch power reduction, pipelining saves power. The pipeline-power rule was first reported in [52], [53]. An early experimental verification for FPGA was performed in [54]. Since then, the effect has been verified in more than 34 experiments of 12 research groups in 8 countries using chips that cover 17 years of FPGA technology [55].

Other variation of the classical pipelining is the self-timed technique. In this case, the clock tree is replaced by a local handshake between processing elements. There is no global clock line; only low-fanout wires of *request-ack* signals. The origin of the technique is [56] but a solid line of research was led by Steve Furber [57], [58]. An interesting feature of self-timed circuits is their smooth requirement of power supply current.

Acknowledgment

The author would like to thank all these people who engineered electronic pipelines over the last 50 years.

References

[1] "Company Timeline", https://corporate.ford.com/history.html. Ford Motor Company. Retirved: 5/12/2018.

[2] J. Deverell, "Pipeline Iterative Arithmetic Arrays". IEEE Trans. on Computers, pp.317-322. March 1975.

[3] H.T. Kung, "Why Systolic Architectures", Computer, pp.37-46, Jan. 1982.

[4] N. Sherrin (Editor), "The Oxford Dictionary of Humorous Quotations", Oxford University Press, 1995.

[5] https://commons.wikimedia.org, File: Ford_assembly_line_-_1913.jpg. Retrieved: 3/1/2019.

[6] L. Cotten, "Circuit Implementation of High-Speed Pipeline Systems", Proc. Fall Joint Computer Conference, pp. 489-504, 1965.

[7] L. Cotten, "Maximum-rate pipeline systems", *Proc. Sprint Joint Computer Conference*, pp. 581-586, 1969.

[8] S. Anderson, J. Earle, R. Goldschmidt, and D. Powers, "The IBM system/360 model 91 floating point execution unit", IBM Journal Res. Development, Vol.11, pp. 34-53, Jan 1967.

[9] M. Flynn, "Very High-speed Computing Systems", Proceedings of the IEEE, Vol. 54, No. 12, December, 1966.

[10] T. Hallin and M. Flynn, "Pipeline of Arithmetic Functions". IEEE Trans. on Computer, pp.880-886. August 1972.

[11] D. Henlin, M. Fertsch, M. Mazin y E. Lewis. "A 16 bit x 16 bit Pipelined Multiplier Macrocell". IEEE Journal of Solid-State Circuits, Vol.SC-20, Nº2, pp.542-547. Abr. 1985.

[12] M. Hatamian and G.L.Cash. "A 70-MHz 8-bit x 8 bit Parallel Pipelined Multiplier in 2.5-um CMOS". IEEE Journal of Solid-State Circuits. August 1986.

[13] T. Noll, D. Schmitt-Landsiedel, H. Klar and G. Enders, "A Pipeline 330-MHz Multiplier", IEEE Journal of Solid-State Circuits, Vol. SC-21, pp. 411-416, Jun. 1986.

[14] M. Santoro and M. Horowitz, "A Pipelined 64x64-bit Iterative Multiplier", IEEE J.of Solid-State Circuits, VOL. 24, n°2, pp.487-493, Apr. 1989.

[15] H. V. Jagadish, R.G. Mathews, T. Kailath and J.A. Newkirk. "A Study of Pipelining in Computing Arrays". IEEE Transactions on Computers, vol. C35, No5 . May 1986.

[16] F. Lu, H. Samueli, J. Yuan and S. Svensson, "A 700-MHz 24-bit pipelined accumulator in 1.2 µm CMOS for Application on Numerically Controlled Oscilators", IEEE Journal of SolidState Circuits, Vol.28, N.8, pp.878-885, August 1993.

[17] Atmel Corp, "SClib ATMEL ATC18", Datasheet Version: 1.5.5-1.0.0, Jan 2002.

[18] Xilinx Inc., "All Programmable 7 Series Product Selection Guide", https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf. Retrieved:12-01-2018.

[19] Xilinx Inc., "Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Generation FPGAs", XAPP465 (v1.1), May 2005.

[20] R. Lyon, "Two's Complement Pipeline Multipliers", IEEE Transactions on Communications, pp. 418 - 425, Vol. 24 , Issue: 4 , Apr 1976.

[21] C. V. Ramamoorthy, "Pipeline Architecture", Computing Surveys, Vol.9, No.1, March 1977.

[22] D. Gross, "Greatest bussiness histories of all times", John Wiley & Sons, Inc. 1996.

[23] P. Cappello y K. Steiglitz. "A VLSI Layout for Pipelined Dadda Multiplier". ACM Trans. on Computer Systems, Vol.1, Nº2, May 1983.

[24] Boeing,http://www.boeing.com/resources/boeingdotcom/history/pdf/Boeing_Chronology.pdf

[25] C. Hauck, C. Bamji and J. Allen, "The Systematic Exploration of Pipelined Array Multiplier Performance", Proc. ICASSP 85, pp.1461-1464. New York: IEEE Press, 1985.

[26] H. Bakoglu, "Circuits, Interconnections, and Packing for VLSI", Reading, Massachusset: Addison-Wesley Publishing Co. 1992.

[27] Burrel G. (Editor), *"Crónica de la Técnica",* Barcelona: Plaza & Janes Publishers, 1989, pp. 524-525.

[28] Texas Instruments, "SN74S74 Dual D-type positive-edge-triggered flip-flops with preset and clear", SDLS119 – December 1983 – revised March 1988.

[29] Texas Instruments, "SN74S00, Quadruple 2-input NAND" , December 1983.

[30] Xilinx Inc., "Virtex-5 FPGA Data Sheet: DC and Switching Characteristics", DS202 (v5.5) June 17, 2016.

[31] Xilinx Inc., "XC8100 FPGA family", Version 1.0, June 1, 1996.

[32] Xilinx Inc., "Spartan and Spartan-XL FPGA Families Data Sheet", DS060 (v2.0) March 1, 2013.

[33] Phillips, "Fast TTL Logic Series", Holland, 1999.

[34] Xilinx Inc. "XC6200 Field Programmable Gate Arrays", (Version 1.8) January 9, 1997.

[35] Xilinx Inc., "XC4000 Series Field Programmable Gate Arrays", Version 1.02, June 1, 1996.

[36] Motorola, "CMOS Logic Data", 1990.

[37] Xilinx Inc., "Virtex-6 FPGA Data Sheet: DC and Switching Characteristics, DS152 (v2.4)", May 11, 2010.

[38] Xilinx Inc, "XC3000 Series Field Programmable Gate Arrays (XC3000A/L, XC3100A/L)", Nov. 9, 1998.

[39] W. Donath, "Wire Length Distribution for Placemment of Computer Logic", IBM J. of Res. Development, vol.25, nº3, May 1981.

[40] R. Jump and S. Ahura. "Effective Pipeline of Digital Systems", IEEE Trans. on Computers, Vol. C-27, Nº9, pp.855-865, Sept. 1978.

[41] P. Song and G. de Micheli, "Circuit and Architecture Trade-offs for High-Speed Multiplication", IEEE Journal of Solid-State Circuits, Vol.26, No.9, Sept 1991.

[42] E. Boemo, S. Lopez-Buedo, N. Acosta, and E. Todorovich, "Local versus Global Interconnections in Pipelined Arrays: An Example of the Interaction between Architecture and Technology", Proc. XIV DCIS Conference, pp.181-186, November 1999.

[43] European Silicon Structures, "ES2 ECPD10 Library Databook", Doc. E01A09, 1993.

[44] J. McCanny and J. McWhirter, "Completely iterative, pipelined multiplier array suitable for VLSI", IEE Proc. pp.40-46. Vol.129, Part G, Nº2. April 1982.

[45] J. Fishburn, "Clock Skew Optimization", IEEE Trans. on Computers, Vol.39, Nº7, pp.945-951, July 1990.

[46] K. Sakallah, T. Mudge, T. Burks and E. Davidson, "Optimal Clocking of Circular Pipelines", Proceeding ICCD'91, pp.642-646. IEEE Press 1992.

[47] D. Wong, "Techniques for Designing High-Performance Digital Circuits Using Wave Pipelining", Tech.Rep. nº CLS-TR-92-508, Stanford Uiversity: Feb. 1992.

[48] C. Gray, W. Liu and R. Cavin, "Wave Pipelining: Theory and Implementation", Norwell, MA: Kluwer Academic Publishers. 1992.

[49] E. Boemo, S. López-Buedo and J. Meneses, "The Wave Pipeline Effect on LUT-Based FPGA Architectures" Proc. ACM FPGA 1996, Monterrey, Feb. 1996.

[50] E. Boemo, S. Lopez-Buedo, and J. Meneses, "Some Experiments about Wave Pipelining on FPGAs", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol.6, No.2, June 1998.

[51] E. Boemo, S. Lopez-Buedo, C. Santos, J. Jauregui and J. Meneses: "Logic Depth and Power Consumption: A Comparative Study between Standard Cells and FPGAs", Proc. Design of Circuit and Integrated Systems Conference (1998).

[52] J. Leijten, "Analysis of Transition Activity and Power Dissipation in Synchronous Logic Circuits". Nat. Lab. Technical Note, no. 339/93, Philips Electronics N.V. (1993).

[53] J. Leijten, J. van Meerbergen' and J. Jess, "Analysis and reduction of glitches in synchronous networks", Proc. European Design and Test Conference (1995).

[54] E. Boemo, G. Gonzalez de Rivera, S. Lopez-Buedo and J. Meneses: "Some Notes on Power Management on FPGAs". In: Field-Programmable Logic and Applications FPL'05, LNCS, vol. 975, pp.149-157, Springer-Verlag 1995.

[55] E. Boemo, J.P. Oliver, and G. Caffarena, "Tracking the Pipelining-Power Rule along the FPGA Technical Literature", Proc. ACM 2013 FPGA World, Stockholm, Sweden. ACM, Sept. 2013.

[56] I. Sutherland, "Micropipelines", Communication of the ACM, vol.22, nº6, pp.720-734. Jun. 1989.

[57] S Furber, "Computing without clocks: Micropipelining the ARM processor", in Asynchronous Digital Circuit Design, pp.211-262, Springer, London1995.

[58] J. Woods, P. Day, S. Furber, J.D. Garside, N. Paver, and S. Temple, "AMULET1: An Asynchronous ARM Microprocessor", IEEE Transactions on Computers, Vol. 46, No. 4, April 1997.