VariPipe: Low-overhead Variable-clock Synchronous Pipelines

Navid Toosizadeh, Safwat G. Zaky and Jianwen Zhu
Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada
navid.toosizadeh@utoronto.ca, safwat.zaky@utoronto.ca, jzhu@eecg.utoronto.ca

***Abstract*— Synchronous pipelines usually have a fixed clock frequency determined by the worst-case process-voltage-temperature (PVT) analysis of the most critical path. Higher operating frequencies are possible under typical PVT conditions, especially when the most critical path is not triggered. This paper introduces a design methodology that uses asynchronous design to generate the clock of a synchronous pipeline. The result is a variable clock period that changes cycle-by-cycle according to the current operations in the pipeline and the current PVT conditions. The paper also presents a simple design flow to implement variable-clock systems with standard cells using conventional synchronous design tools. The variable-clock pipeline technique has been tested on a 32-bit microprocessor in 90nm technology. Post-layout simulations with three sets of benchmarks demonstrate that the variable-clock processor has a two-fold performance advantage over its fixed-clock counterpart. The overhead of the added clock generation circuit is merely 2.6% in area and 3% in energy consumption, compared to an earlier proposal that costs 100% overhead.**

I. Introduction

The time required to complete an operation in any given stage of a pipelined system depends on the operation being performed. In a conventional synchronous system, the delay of the longest path of the pipeline under the worst process-voltage-temperature (PVT) corner is used to determine the clock frequency. However, the longest path of the system is not necessarily triggered in every cycle. Also, the system is normally operating under typical PVT conditions. Hence, there are many times when a frequency much higher than that derived under worst conditions is possible.

This paper introduces a variable-clock synchronous pipeline design (VariPipe), in which the clock period is adjusted in each clock cycle based on the operations taking place in the pipeline stages. An on-chip clock generation circuit dynamically matches the delay of the current operations of the pipeline in every cycle. At the same time, the clock period automatically adjusts to the current PVT conditions. The proposed approach achieves better performance than isochronous clocking, while retaining the simplicity of synchronous system design. Other advantages of variable-clock synchronous pipelines include a reduction in electromagnetic noise and suitability for voltage scaling techniques. These features make variable clocking appealing for many applications including embedded systems and portable devices.

Several studies have been published that address variable-speed pipelines, including Telescopic units [1] and a variable-clock pipeline processor introduced by Dean [2]. Asynchronous design methodologies such as desynchronization [3] and Mousetrap [4] have also been proposed to achieve average-case performance. The main advantage of VariPipe over previous work is its lower clock generation overhead. In the case study presented in this paper, the overhead of the added clock generation circuit is only 2.6% in area and 3% in energy consumption for a VariPipe DLX processor. By comparison, Dean's variable clocking approach uses duplicates of functional units, which double the area and energy consumption of the functional units. Both the VariPipe processor and Dean's achieve the same performance improvement of 2X over conventional isochronous design. The desynchronization method introduces a 13.5% area overhead in a DLX processor [3]; the processor does not adjust its speed based on the operations in the pipeline and therefore its performance gain is limited.

The VariPipe approach is based on a synchronous circuit implementation, and thus, many challenges in the design of asynchronous circuits are avoided. Also, it employs a simple design methodology using standard cells and conventional synchronous design tools, which allows designers to use the proposed approach in many applications. A comprehensive comparison to related work is presented in Section VIII.

Section II describes the basic idea of the VariPipe approach and Section III explains the methodology in more detail. Timing constraints are given in Section IV. A design flow to implement VariPipe application specific integrated circuits (ASICs) is presented in Section V, which is demonstrated and evaluated through a microprocessor case study in Sections VI and VII.

II. VariPipe: The idea

Consider a pipelined system consisting of several pipeline stages of combinational logic, separated by pipeline registers. Each pipeline stage may have different modes of operation that are exercised by different instructions as they flow through the pipeline stage. For example, the execution stage of a RISC processor may execute different operations such as addition, bitwise logical operations, etc. The key observation is that these operations activate different paths and thus, they have different delays. In the isochronous clocking scheme, employed in today's dominant EDA methodology, the clock period is constant, which means it must be longer than the delay of all possible operations in the pipeline at *all times*. VariPipe employs a clocking scheme in which the clock period continuously tracks the maximum delay under *current* PVT conditions for all operations currently being performed. As Fig. 1 shows, a variable delay is associated with each pipeline stage, and its delay is adjusted to match the delay of the current operation in the stage, as determined by the data in that stage's input registers. When the delays of all pipeline

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

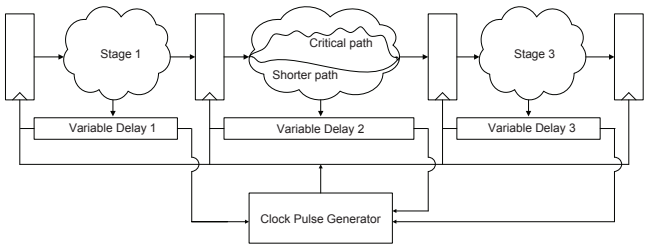


Fig. 1. VariPipe technique

stages have elapsed, the clock pulse generator creates a new clock pulse. As a result, some clock cycles are shortened and the overall speed is increased. The variable delay unit is placed close to the corresponding datapath to be subject to the same PVT conditions.

Note that although the proposed architecture benefits from its asynchronous nature, the use of asynchronous design is limited to the clock generation circuit, leaving the rest of the system still a synchronous circuit that can be designed, synthesized, and laid out using a traditional design flow.

III. Design Methodology

In this section, the methodology for the design and implementation of VariPipe systems is described in detail. The design process starts with a high-level hardware description of the system and its implementation in the target technology. Adding the VariPipe facilities involves three steps: creating delay profiles, simplifying the delay profiles and implementing the clock generation circuit.

A. Creating Delay Profiles

Different operations in any stage of the pipeline can be identified from the high-level hardware description of the system. Each operation takes the values in the input registers and saves its result in the output registers. The result of an operation may not be needed in every cycle, as determined by the *selection signals* for that operation. The different operations that can be performed in any pipeline stage and the conditions under which the results of those operations are selected are recorded in an *operation selection table*. The case study below shows that operation selection tables are simply constructed using a high-level description of the system without reference to the low-level implementation details.

The maximum delay of any operation of the pipeline stage is determined based on its implementation in the target technology. There are two methods to find the delays: I) Dynamic timing analysis (DTA), which finds the delays using test vectors. II) Static timing analysis (STA), which is used in this paper. For each pipeline stage, the delays of all the operations are found, and a *delay profile* is created by grouping the operations of the pipeline stage according to their delay values. The case study below presents a simple and automated approach to construct delay profiles.

B. Simplifying Delay Profiles

Each pipeline stage has a *minimum delay* that can be identified from the delay profile of that stage. The path having the largest minimum delay of all pipeline stages is the *shortest inevitable path* of the pipeline. To reduce the number of delay values needed, delay values less than the delay of the shortest inevitable path in each profile are grouped and rounded up to the maximum value of the group. This simplifies the delay profile and the implementation of the clock generation circuit.

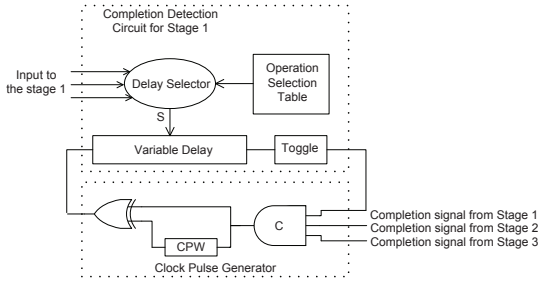


Fig. 2. Clock generation circuit

C. Implementing the Clock Generation Circuit

Fig. 2 shows the clock generation circuit, which is composed of two parts: the completion detection circuits and the clock pulse generator. The design of the clock generation circuit is based on the two-phase asynchronous design style [5] and thus inherits many properties from asynchronous systems. The completion detection circuit for each stage is composed of a variable delay, a toggle, an operation selection table and a delay selector. The delay selector reads appropriate signals from the inputs to the pipeline stage. It uses the operation selection table, which is ordered according to the delay values, to generate a one-hot delay selection signal (S) to select the appropriate delay value. If the inputs to the pipeline stage activate more than one operation (e.g., in a complex multi-task stage), the delay corresponding to the operation with the longest delay is selected. When the clock pulse emerges from the variable delay element, it is converted to a level by the toggle before being sent to the C-element. Initially, all toggle elements are reset and so is the output of the C-element. After the reset is removed, all toggle elements change state causing the C-element to toggle its output, thus creating a clock pulse of width CPW at the output of the XOR. The clock pulse loads new values into the input registers of each pipeline stage.

Note that the delay through the delay elements must be at least long enough for the corresponding delay-selection signals to become valid. After a delay matching the operation with the longest delay currently in the stage, the toggle changes state. When all stages have switched to the new state, the C-element toggles creating a new clock pulse.

D. Variable Delay Implementation

Consider a pipeline stage whose delay profile has three values, $d1$, $d2$, and $d3$ ($d3 > d2 > d1$), to be selected by signals $S1$ and $S2$. The design of the variable delay and the output toggle for that stage are illustrated in Fig. 3. The values of the three delay elements $k1$, $k2$ and $k3$ are selected such that the total delay around the clock loop in Fig. 2 matches the stage's delay profile $d1$, $d2$ and $d3$.

Delay $k1$ in Fig. 3 consists of a long chain of gates which change state twice with every input pulse. To reduce power consumption, the delay architecture shown in Fig. 4

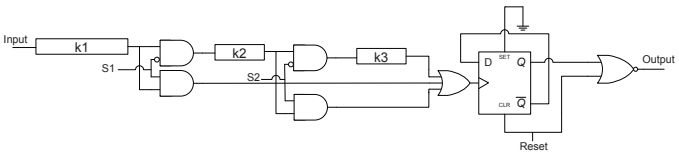


Fig. 3. Variable delay and toggle

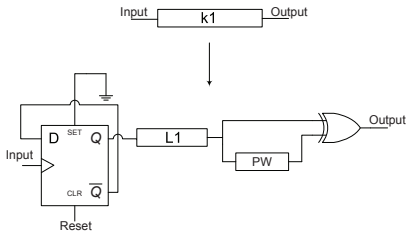


Fig. 4. Reducing the switching power of delay element

is proposed. The input pulse to the delay chain is converted to a level, then at the end of the chain, converted back to a pulse. As a result, the gates composing delay $L1$ switch only once with each input pulse. Delay $L1$ should be tuned such that the minimum delay of the path between the input and the output matches the desired delay and delay element PW should be adjusted to generate a suitable pulse width. Simulations showed that the power saving achieved by this technique is close to 50% for long delay chains.

IV. Timing Constraints

Fig. 5 shows a simplified model of the clock generation circuit with three completion detection circuits. The timing constraints on the design of the clock generation circuit may be summarized as follows:

- The reset signal to the system must be long enough to ensure that the delay elements get successfully reset and all the gates and flip-flops become stable.
- Each loop in Fig. 5 has different rise and fall times and thus, the minimum delay of the loop should be used for delay tuning.
- Each completion detection circuit must be placed within the corresponding stage to ensure that it matches the datapaths' delays under the prevailing PVT conditions in that stage. When adjusting the delay elements, appropriate margins should be used because factors such as crosstalk, IR drops, noise, inductance, etc. may affect the datapath and the completion detection circuit differently.
- Part of the delay of the loops in Fig 5 is the clock pulse generator and the clock tree delay. The clock pulse generator and the root cells of the clock tree are not necessarily close to the pipeline stage and their PVT conditions may be different. Therefore, the clock pulse generator and clock tree delays must be used with appropriate margins when tuning delays. In the case study presented below, only 90% of the clock tree and the clock generation circuit delay is taken into account, thus ensuring that the total delay around each of the clock loops is slightly larger than the required delay.
- The delays of the clock generation loops should be tested under all PVT corners to ensure that the delay elements inside the loops are sufficiently large.

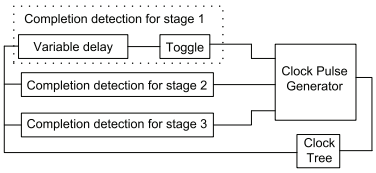


Fig. 5. A simplified model of the clock generation circuit

- Delay selection signals Si in Fig. 2 and Fig. 3 must become valid before the input clock pulse emerges from the first delay element ($k1$) and thus, delay $k1$ must be sufficiently long.
- The clock pulse width determined by CPW in Fig. 2 and the pulse width determined by PW in Fig. 4 are tested under all PVT conditions to ensure that the pulse width requirements of sequential elements are not violated.
- Communication of a variable-clock system with its environment needs special attention to ensure correct data transfers. The problem of transferring data between unsynchronized clock domains already exists in many high-speed systems. As such, many approaches are in use to minimize metastability and data loss when different clock domains are connected. They include multi-flop synchronizers, multiplexer recirculation techniques, use of first-in-first-out buffers between different clock domains and handshake techniques [6], [7]. Similar synchronization techniques may be applied for inter-chip and intra-chip data transfers between a VariPipe system and its environment.

V. Design Flow

Fig 6 shows the proposed design flow to implement VariPipe systems using standard cells. The design flow is explained in detail in the following case study.

VI. Case Study: VariPipe DLX Microprocessor

To test the performance of a variable-clock synchronous pipeline, a VariPipe version of Hennessey and Patterson's 32-bit DLX pipeline microprocessor [8] was implemented in 90nm technology. The Verilog code of the processor was downloaded from opencores.org [9]. The DLX core is a RISC microprocessor with five pipeline stages: instruction fetch, instruction decoder, instruction execution, memory access and write back. To implement the processor, the design flow of Fig. 6 was realized using the toolset shown in Table I.

The main synchronous core was constrained to a clock period of 8.73 ns to accommodate the worst PVT corner. Then, two versions of the processor were generated: one version equipped with the VariPipe technique and the other a conventional synchronous circuit (fixed-clock). Both designs were optimized for minimum power and area.

TABLE I
Toolset

Objective	Tool	Version
Synthesis	Design Compiler	Y-2006.06-SP5
Timing and power analysis	PrimeTime-PX	Y-2006.06-SP3-1
Physical design	SoC Encounter	5.2
Simulation	ModelSim	6.3c

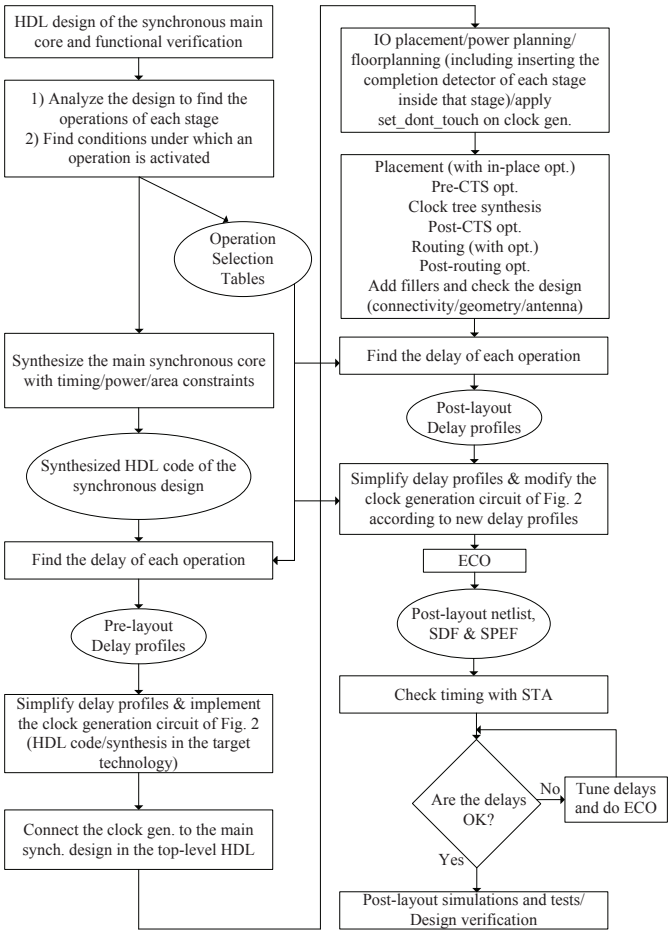


Fig. 6. Proposed VariPipe design flow, HDL ≡ Hardware Description Language, DRC ≡ Design Rule Check, STA ≡ Static Timing Analysis, ECO ≡ Engineering Change Order, SDF ≡ Standard Delay Format, SPEF ≡ Standard Parasitic Exchange Format, CTS ≡ Clock Tree Synthesis

A. Implementing the VariPipe DLX processor

According to the design flow of Fig. 6, the first step after obtaining the behavioral HDL of the DLX processor is to analyze the design to identify the operations of each pipeline stage and the conditions under which each operation is selected. The execution unit and the decoder are given here as examples.

Execution unit: Part of the behavioral Verilog code of the execution unit is shown in Fig. 7. The execution unit performs a range of tasks including logical and arithmetic operations on input registers *A* and *B* and places the result into the ALU result register. The results of these operations are available on the intermediate signals *ADD_result*, *AND_result* and *SUB_result*. One of these signals is selected as the output on *ALU_result* based on the instruction opcode field and instruction function field, which are available in the input registers of the execution unit. Thus, the operation selection table of the execution unit can be derived as in Table II.

```
`define ADD 6b`100000
`define SUB 6b`100010
`define AND 6b`100100
...
assign ADD_result = reg_A + reg_B;
assign SUB_result = reg_A - reg_B;
assign AND_result = reg_A & reg_B;
...
if (IR_opcode_field == 0) //R-type format inst. or NOP
    case(IR_function_field)
            `ADD: ALU_result <= ADD_result;
            `SUB: ALU_result <= SUB_result;
            `AND: ALU_result <= AND_result;
...
```

Fig. 7. Verilog code of the Execution unit

TABLE II
OPERATION SELECTION TABLE OF EXECUTION UNIT

Operation	Selection signals (Si)	
	IR_opcode_field	IR_function_field
ADD	0	6'b100000
SUB	0	6'b100010
AND	0	6'b100100
...	...	...

Decoder: The decoder is responsible for generating the branch signal, which declares that a branch has to be taken in the next cycle. The decoder also computes the branch address and sends it to the fetch unit. The result of this computation is needed only if the branch is to be taken. Therefore, when the branch signal becomes valid and if it is equal to zero, there is no need to wait for the computation of the branch address to be completed. The operation selection table of the decoder is shown in Table III. After synthesizing the main core with the design constraints, pre-layout delay profiles of the pipeline stages are extracted using the STA tool and operation selection tables. This and simplifying delay

TABLE III

OPERATION SELECTION TABLE OF DECODER

Operation	Selection signals (Si)
Branch_signal	1 (always computed)
Branch_address	Branch_signal
...	...

profiles are explained later for post-layout delay profiles as the process is the same.

To implement the clock generation of Fig. 2, the operation selection table of each pipeline stage is used to create the delay selection logic. Delay elements are not fine-tuned at this stage as the delays will change during layout. Initially, delay elements are selected to be around 30% larger than needed.

To simplify the implementation of the clock generation circuit, a library of delay elements in the target technology is created. A delay element is implemented as a chain of $2n$ inverters, where $n = 1, 2, ..., N$. Then, the delay of each delay element is estimated using the STA tool. The result is a table of several delay elements and their corresponding delay values. The clock generation circuit is completed using these delay elements.

The clock generation circuit is then connected to the main synchronous core in the top-level HDL used for the layout flow. Most layout steps are similar to the conventional synchronous design flow [10]. The main difference is that the completion detection circuit of each stage is constrained to be placed inside that stage.

After the place and route steps are completed, post-layout delay profiles are created. The list of operations for which delay values are needed is readily available from the operation selection tables. The delays of various operations are found using static timing analysis (STA). The compiler's STA facility enables the designer to obtain the longest delay in any pipeline stage. However, constructing the delay profiles requires information about the longest path for each of the operations in the operation selection table. The required information can be obtained using the STA facility as follows. To find the delay of a given operation of a pipeline stage, the corresponding selection fields in the input registers of the stage are set to the values that select that operation, using *assign* statements in the HDL netlist. These values will in turn, set the corresponding selection signals for that operation and the delay reported by the STA tool will be the delay of the desired operation. This process can be automated using an appropriate script.

As an example, operations *ADD*, *SUB* and *AND* of the execution stage save their results in the *ALU_result* register. To find the delay of the *ADD* operation, selection signals *IR_opcode_field* and *IR_function_field* are set to 0 and 6'b100000 in the post-layout netlist, as per the information in Table II. As a result, the delay from the input registers to the *ALU_result* register reported by the STA tool is the desired delay.

The delay profiles of the execution unit and the decoder under the worst PVT corner are given in Table IV, with a 10% margin. These are the values to be matched by the delay elements. The critical path delay is augmented from 8.73 ns to 9.6 ns, using the 10% margin.

TABLE IV

POST-LAYOUT DELAY PROFILES OF DECODER AND EXECUTION UNIT

Decoder	
Operation	Delay + 10% margin (ns)
Branch_address	9.06
Slot_number	6.35
Branch_signal	6.07
...	...
WriteBack_index	0.59
Execution Unit	
Operation	Delay + 10% margin (ns)
SUBI	9.60
...	...
SLT	7.60
SRLI	6.91
...	...
NOP	2.27

TABLE V

SIMPLIFIED DELAY PROFILES

Decoder unit	
Operation	Delay (ns)
Branch_address	9.06
All others	7.18

Execution unit	
Operation	Delay (ns)
SUBI	9.60
...	...
SLT	7.60
SRLI, and all others	6.91

The next step is to simplify the delay profiles. The longest delay of the decoder is the branch address calculation. The branch signal determines if this calculation is needed. Therefore, the branch signal computation is an unavoidable operation and its corresponding path is the shortest inevitable path of the decoder, which is 6.07 ns (with the 10% margin). The simplified view of the clock generation circuit previously given in Fig. 5 may be used for the clock period calculation. To calculate the clock period corresponding to the shortest inevitable path of the decoder, the delay of the path is augmented by the delay of the toggle (0.35 ns), the clock pulse generator (0.45 ns) and the clock tree (0.31 ns). As a result, the shortest possible clock period is 7.18 ns. Since this is more than the delay of the other operations of the decoder (except *Branch_address* computation), the decoder's delay profile can be simplified to two delays, as shown in Table V.

The minimum delay of the decoder in Table V is larger than the minimum delay of all other stages, making it the shortest inevitable path of the pipeline. Hence, delays less than 7.18 ns in the delay profile of each other unit were grouped and rounded up to the maximum of the group. In the case of the execution unit, all delays equal to or less than 6.91 ns were grouped together, as shown in Table V. The clock generation circuit is modified according to the new delay profiles, followed by an engineering change order (ECO) pass to update the layout.

The next step is to fine-tune the delay elements. The physical design tool is used to write the post-layout HDL netlist along with the standard delay format (SDF) file and the standard parasitic exchange format (SPEF) file for post-layout STA. The long delay chains used during synthesis are adjusted by replacing them with other delay elements from the delay library, then tested by the STA tool to check if they are of appropriate length. This process is repeated until

appropriate delay values are achieved. An ECO pass is done to update the layout with the modified delays.

The longest delay for the *Si* signals of each pipeline stage was compared against the $k1$ delay (see Fig. 3) to ensure that the delay selection signals are settled before the input pulse emerges out of the $k1$ delay element. In our experiments, delay $k1$ was sufficiently larger than the delay of *Si* signals.

The PVT corners for the 90nm technology used in the experiments are shown in Table VI. Delays were tuned under the worst PVT corner with the 10% margin. They were also examined under the typical and best PVT corners to ensure they are sufficiently large. As explained in the next section, the VariPipe processor was simulated under all PVT corners to verify correct functionality.

TABLE VI

PVT CORNERS

PVT corner	Process	Voltage	Temperature
Best	Fast	1.1	-40°C
Typical	Typical	1.0	25°C
Worst	Slow	0.9	125°C

VII. EVALUATION

In this section, different characteristics of VariPipe and fixed-clock processors are compared. Also, the area and energy overhead of the clock generation circuit are quantified. Functionality, performance and energy consumption of the VariPipe DLX processor and its fixed-clock counterpart were analyzed using the three benchmark suites shown in Table VII, which were compiled by DLX GCC [11]. Post-layout simulations of the circuits were performed for each benchmark and switching activities were recorded in the switching activity interchange format (SAIF). These, together with parasitic data (SPEF files), were used by PrimeTime-PX for simulation-based power analysis.

TABLE VII

BENCHMARKS

Source	Benchmark
MiBench [12]	adpcm_coder adpcm_decoder crc32 dijkstra qsort
PowerStone [13]	bcnt blit compress ucbqsort
Applications from [14], [15]	Bubble Sort JPEG-DCT MP3-DCT32 MPEG2-Bdist

A. Performance analysis

Fig. 8 shows execution times under best, typical and worst PVT conditions. The same 10% margin was used for the fixed-clock processor. The performance of the fixed-clock system is the same under all conditions, but the performance of the VariPipe system varies with PVT conditions as shown.

Table VIII shows the execution time reduction percentage obtained using VariPipe. Under the worst-case conditions, the execution times of benchmarks are 13% shorter, on average, for the VariPipe design, because the VariPipe system adjusts the clock period in each cycle to match the current

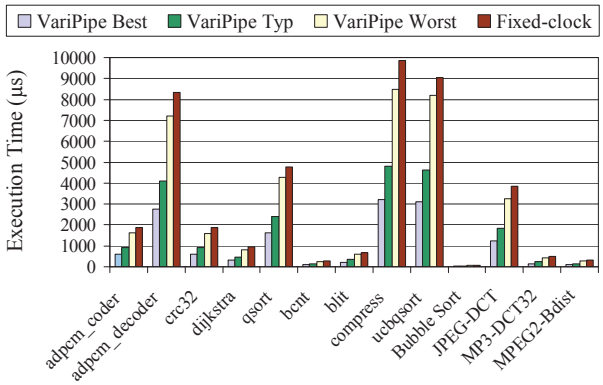


Fig. 8. Performance of VariPipe and fixed-clock DLX processors under all PVT corners

TABLE VIII

EXECUTION TIME REDUCTION PERCENTAGE USING VARIPIPE

Program	Reduction % under worst	Reduction % under typ
adpcm_coder	14.0	51.3
adpcm_decoder	13.4	51.0
crc32	14.5	51.7
dijkstra	12.8	50.6
qsort	10.6	49.4
bcnt	12.4	50.4
blit	11.4	50.0
compress	14.1	51.4
ucbqsort	9.4	48.7
Bubble Sort	11.0	49.6
JPEG-DCT	15.9	52.4
MP3-DCT32	15.0	51.9
MPEG2-Bdist	15.4	52.1
Average	12.7	50.6

operations. The reduction in execution time varies with the frequency of occurrence of the instructions and their sequence in the program. The VariPipe system is twice as fast as the fixed-clock counterpart under typical conditions. The average percentages in the table are calculated by summing the execution times of all programs.

B. Energy consumption analysis

Energy consumption of the two processors under typical PVT conditions is compared in Table IX. Energy values in the table do not include memory and IO. The core energy is the energy consumption of the processor excluding the clock tree. The VariPipe system consumes only 3% more energy than the fixed-clock period system.

C. Area and energy overhead

The clock generation circuit takes up only 2.6% of the total area of the VariPipe processor. The area of the clock generation circuit is mainly taken by the delay elements. As previously mentioned, the energy overhead of using VariPipe is merely 3%. The clock generation circuit is a very small portion of the total area and is implemented using low-leakage cells and thus, its leakage power is negligible compared to that of the processor.

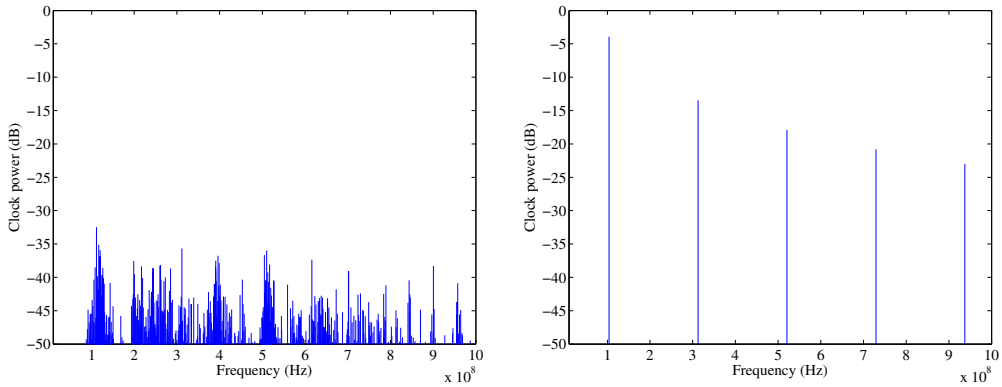
D. Resilience to PVT variations

The VariPipe system automatically adjusts to inter-chip and intra-chip PVT variations to deliver the best-possible performance. Fig. 8 represents the inter-chip PVT variation analysis of the VariPipe DLX processor. It works correctly

TABLE IX

ENERGY CONSUMPTION UNDER THE TYPICAL PVT CORNER

Program	Fixed-clock processor				VariPipe processor			
	Energy (μJ)			Execution time	Energy (μJ)			Execution time
	Core	Clock tree	Total	(μs)	Core	Clock tree	Total	(μs)
adpcm_coder	2.464	4.741	7.205	1871.141	2.663	4.782	7.445	910.327
adpcm_decoder	12.309	21.151	33.460	8333.573	13.012	21.323	34.335	4084.067
crc32	2.450	4.748	7.198	1875.825	2.697	4.789	7.486	906.696
dijkstra	1.255	2.380	3.635	940.382	1.324	2.401	3.725	464.146
qsort	7.829	12.091	19.920	4767.912	8.075	12.195	20.270	2412.549
bcnt	0.419	0.732	1.151	289.032	0.441	0.738	1.179	143.215
blit	0.948	1.722	2.670	679.742	1.008	1.737	2.745	340.645
compress	12.340	25.006	37.346	9864.485	13.596	25.218	38.814	4792.448
ucbqsort	14.427	22.934	37.361	9039.672	15.224	23.404	38.628	4634.456
Bubble Sort	0.165	0.215	0.380	84.734	0.168	0.217	0.385	42.661
JPEG-DCT	5.830	9.769	15.599	3850.478	6.203	9.852	16.055	1831.509
MP3-DCT32	0.721	1.223	1.944	479.457	0.773	1.232	2.005	230.382
MPEG2-Bdist	0.461	0.844	1.305	330.715	0.505	0.846	1.351	158.338
Total energy, Total time	61.618	107.556	**169.174**	**42407.148**	65.689	108.734	**174.423**	**20951.439**



(a) VariPipe (b) Fixed-clock

Fig. 9. Comparison of the clock power spectra

under all PVT conditions. The execution times drop by as much as 62% from the worst to the best PVT conditions. The VariPipe system is also resilient to intra-chip PVT variations. To test this, starting from the typical-case SDF file, the delays of the execution unit and its completion detection circuit were augmented by 10% using the Design Compiler's derating commands. A new SDF file was generated to be used in simulations, which verified that the system executed all the benchmarks correctly.

E. Reduction in electromagnetic noise

The clock is often the main source of electromagnetic noise in a digital system, because it has a fixed frequency which is also the highest in the system [16]. Many circuits employ spread-spectrum oscillators to overcome this problem [17]. In VariPipe systems, the clock frequency varies within a range around an average value and thus, the clock power is spread over that range. The clock power spectrum of the VariPipe DLX processor under the worst PVT corner for one of the benchmarks is compared against its fixed-clock counterpart in Fig. 9. The maximum clock power of VariPipe is about 28 dB less than that of the fixed-clock design. With no central peak in the frequency spectrum, the VariPipe processor should generate less electromagnetic noise compared to its fixed-clock counterpart.

F. Suitability for voltage scaling

The 90nm technology used in this paper is characterized for two supply voltage levels: 1.0 V and 1.2 V. These characterizations were used to apply voltage scaling to the VariPipe processor. It was ensured that the pads were compatible with 1.2 V and no hold violation occurred. The system was simulated under typical PVT conditions for both supply voltages. The system automatically adjusts its speed to changes in supply voltage. The system was 1.2 times faster using the 1.2 V supply, compared to the 1.0 V supply. This shows that the VariPipe design is amenable to voltage scaling techniques.

VIII. RELATED WORK

Other studies have been published to design variable-speed pipelines and use typical PVT conditions. Variable clocking was addressed in Dean's PhD thesis [2]. Dean uses transistor-level duplicates of the functional units in each of pipeline stages to indicate the completion of an instruction. Duplicates introduce a substantial overhead because they double the required area for each functional unit, which in turn, increases the power consumption considerably. VariPipe uses variable delays, which have a significantly smaller overhead compared to the duplicates in Dean's method. The case study showed that the overhead of the added clock generation

circuit for a VariPipe DLX processor is 2.6% in area and 3% in energy consumption. The VariPipe DLX processor and Dean's both achieve 2X performance improvement over isochronous design. The design of duplicates in Dean's work is complicated as they are implemented at the transistor level. Loads on the transistors of the functional units are imitated using passive transistors. VariPipe employs *static timing analysis* in the design of matched delay elements for completion detection circuits. As such, the design of the completion detection circuit is independent of the function being matched. This allows the introduction of a simple design methodology that uses conventional design tools to implement variable-clock systems with standard cells. As a result, the proposed approach can be readily used in many applications.

Telescopic units are introduced in [1] to design variable-speed pipelines. A fixed clock period shorter than the delay of the critical path is applied to the pipeline. When the critical path is triggered, a hold signal is raised to show that another clock cycle is required for the instruction to complete. Since the critical path of each pipeline stage is not triggered in every cycle, an overall throughput improvement of 27% has been achieved. In comparison, VariPipe adjusts to the current instructions in the pipeline as well as present PVT conditions, and hence, achieves a better performance improvement.

TEAtime [18] uses a replica of the critical path to track PVT variations in a DLX-style processor on FPGA and achieves a 34% speed improvement. The processor does not change its speed with instructions and it is not resilient to intra-chip PVT variations. In [19], the latencies of instructions in all pipeline stages are saved in memory to adjust the clock period at run time using a PLL and a clock synthesizer. A 17% speedup for one of the test programs has been achieved. This approach is limited by the number of phases a PLL can produce. Also, the clock does not adjust to PVT variations automatically.

The Razor project [20] shows the possibility of reducing the voltage margins used in worst-case analysis of synchronous circuits. This work reduces dynamic power by reducing the input voltage, but keeps the clock frequency intact. An error recovery circuit is added to cope with any timing errors due to the reduced voltage.

Asynchronous circuits can also be designed to achieve average-case performance and adjustability to PVT variations. Several asynchronous design styles exist, including desynchronization [3], by which a synchronous design is converted into an asynchronous one, and Mousetrap [4], which is a methodology to design high-speed pipelines taking advantage of PVT variability. In a design approach by Nowick, variable delays are used in the implementation of a speculative completion detection circuit for an asynchronous adder [21]. VariPipe is simpler than asynchronous design. It uses standard-cell design implementation and also does not require customized transistor-level circuits. It uses conventional synchronous design tools and thus, neither asynchronous design methods nor asynchronous design tools are required. Desynchronization introduces an area overhead of 13.9% in a DLX microprocessor [3]; the processor does not adjust its speed according to the operations in the pipeline.

IX. Conclusion

This paper proposes a new clocking scheme that can allow the clock period to track the delay of a pipeline on a cycle-by-cycle basis. A low-overhead clock generation circuit and a standard-cell design flow compatible with today's dominant EDA design methodology is demonstrated. This allows designers to use the proposed approach in many applications. A case study has been presented, which demonstrates that the VariPipe DLX processor has a two-fold performance advantage over its fixed-clock counterpart. The overhead of the added clock generation circuit is only 2.6% in area and 3% in energy consumption. Resilience to PVT variations, reduction in electromagnetic noise, and suitability for voltage scaling are among other advantages of VariPipe systems.

X. Acknowledgments

The authors gratefully acknowledge the financial support of Natural Sciences and Engineering Research Council of Canada and of the Government of Ontario.

References

[1] L. Benini, E. Macii, and M. Poncino, "Telescopic units: Increasing the average throughput pipelined designs by adaptive latency control," in *Design Automation Conf.*, June 1997, pp. 22–27.

[2] M. Dean, "STRiP: A self-timed RISC processor," Ph.D. dissertation, Stanford University, 1992.

[3] N. Andrikos, L. Lavango, D. Pandini, and C. Sotiriou, "A fully-automated desynchronization flow for synchronous circuits," in *Design Automation Conf.*, June 2007, pp. 982–985.

[4] M. Singh and S. Nowick, "Mousetrap: High-speed transition-signaling asynchronous pipelines," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 684–698, June 2007.

[5] I. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[6] "Clock domain crossing: Closing the loop on clock domain functional implementation problems." Cadence Design Systems, Inc., 2004.

[7] A. Lines, "Asynchronous interconnect for synchronous SoC design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, Feb. 2004.

[8] J. Hennessey and D. Patterson, *Computer Architecture: A Quantative Approach*. Morgan Kaufmann, 2007.

[9] "ASPIDA," in *http://www.opencores.org/projects.cgi/web/aspida*.

[10] "ASIC design flow," in *http://www.faraday-tech.com/html/products/asic/Design_Flow.html*.

[11] "DLX GCC," in *http://www2.ucsc.edu/courses/cmps111-elm/dlx*.

[12] "MiBench," in *http://www.eecs.umich.edu/mibench*.

[13] L. Lee, B. Moyer, and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 1999, pp. 267–269.

[14] B. Gorjiara and D. Gajski, "Automatic architecture refinement techniques for customizing processing elements," in *Proc. of the 45th ACM/IEEE Design Automation Conf.*, June 2008, pp. 379–384.

[15] "NISC technology website," in *http://www.cecs.uci.edu/∼nisc*.

[16] H. Ott, *Noise Reduction Techniques in Electronic Systems*, 2nd ed. John Wiley & Sons, 1988.

[17] DALLAS SEMICONDUCTOR, "App note 3512: Spread-spectrum clock oscillators lower EMI," 2005.

[18] A. Uht, "Uniprocessor performance enhancement through adaptive clock frequency control," *IEEE Trans. on Computer*, vol. 54, no. 2, pp. 132–140, Feb. 2005.

[19] H. Epassa *et al.*, "Implementation of a cycle by cycle variable speed processor," in *Proc. of IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2005, pp. 3335–3338.

[20] T. Austin, D. Blaauw, T. Mudge, and K. Flautner, "Making typical silicon matter with Razor," *Computer*, vol. 37, no. 3, pp. 57–65, 2004.

[21] S. Nowick, "Design of a low-latency asynchronous adder using speculative completion," in *IEE Proceedings – Computers and Digital Techniques*, Sept. 1996, pp. 301–307.