An Effective Power Management Scheme for RTL Design Based on Multiple Clocks[†]

C. Papachristou[‡] M. Spinning[§] M. Nourani[¶]

Abstract

This paper presents an effective technique of low power design for RTL circuits and microarchitectures. The basis of this technique is: a) to use a multiple clocking scheme of n non-overlapping clocks, by dividing the frequency f of a single clock into n cycles; b) to partition the circuit into disjoint modules and assign each module to a distinct clock with frequency f/n. However, the overall effective frequency of the circuit remains f the single clock frequency. The results show that our multiple clocking scheme provides more effective power management (power savings up to 50%) at the RTL in comparison to conventional power management techniques based on gated clocks.

1. Introduction

A major feature of currently proliferating portable applications is their requirement for low power consumption because they use battery voltage. Considering "power" rather than area or speed as the main optimization factor requires a second look at the entire VLSI design technologies, techniques, architectures and even algorithms. There are two major sources of power dissipation in a CMOS circuit[1]: 1) Static dissipation due to leakage current or other currents drawn continuously from the power supply; and 2) Dynamic dissipation due to: switching transient current (short circuit power dissipation) and charging and discharging of load capacitors.

Usually it is the dynamic switching components, resulting from the charging and discharging of capacitors, which dominate the total power consumption of CMOS circuits. The dynamic power dissipation for a CMOS gate with a load capacitor C_L is [1]: $P_d = C_L.V_{DD}^2.f$, where V_{DD} is the power supply voltage and f is the frequency of switching.

Reducing V_{DD} is an obvious way of power reduction. It was shown in [2] that 60% power reduction is possible for a 3.3 Volt system compared to the same circuit runing with a 5 Volt power supply. Unfortunately, this simple solution comes at a cost on the delay. We will pay a speed penalty for a V_{DD} reduction which after all may not be acceptable. Load capacitor is a technology-dependent factor and can be reduced by using special implementation of transistors on Silicon. The frequency of the switching signals (e.g. the clock frequency) is another important contributor to the dynamic power dissipation and roughly speaking is an indication of activities performed by a component.

• **Prior Work**: There are basically three levels in which power consumption methods have been investigated: 1) transistor level, 2) logic level; and 3) register-transfer level.

At the transistor level, the attention is on optimizing the transistor size and careful cell selection. By minimizing transistor size as much as possible we minimize the parasitic capacitances of transistors and interconnect routings [3]. The work explained in [4] presents an LP-based algorithm to find the best cell combination (when more than one cell for each logic gate is available) to reduce power consumption. A pass transistor logic family was found to minimize the capacitance[5] as an alternative to conventional CMOS logic family.

Much work for power optimization has focused at the logic level. There are various choices for implementing a boolean function. The power tradeoff between different types of adders and multipliers was investigated in [6]. To minimize the total switched capacitance in random logic modules, several logic synthesis optimization techniques have been proposed [7] to lower the power consumption. In [8] power is minimized by modifying the function of each node in the circuit. Re-encoding of a sequential circuit [9] and using gated clocks [10] are two techniques for power reduction in sequential circuits.

Considering power consumption at levels higher than logic has been recently attempted. A high-level synthesis system, HYPER-LP, presented in [11] uses a variety of architectural and computational transformations to estimate and optimize the power dissipation.

In this paper we present an effective power management scheme for RTL designs based on non-overlapping multiple clocks. The contribution of this work is twofold. First, we show that using non-overlapping multiple clocking to design a partitioned datapath, so that each module is assigned to a distinct clock, is an effective way of RTL datapath synthesis with minimum power consumption. Second, we present a multiple clock allocation algorithm for power reduction. A key feature of this scheme is holding the old input values as long as possible (in a register or through a MUX whose control line is latched) to reduce the transitions on the ALU input ports.

This paper is organized as follows: Section 2 presents a motivating example by which we show the basic idea of our work. Section 3 describes RTL structural model and analysis of the multi-phase clocking to save power dissipation. Section 4 presents our synthesis approach and the effect of allocation on power consumption. Experimental results are shown in section 5. Finally, concluding remarks are in section 6.

2. Motivating Example

The data flow graph of Fig. 1(a) shows the behavior of a simple circuit. This flow graph is scheduled in five time steps. Fig. 1(b) and (c) are two RTL circuit implementations of this behavior, Circuit 1, and Circuit 2, respectively. Circuit 1 has been implemented by minimal resource allocation using two *(+,-)* ALUs. Specifically, we allocated nodes *N1, N2, N3* and *N4, N5, N6* into the left and right side ALU of Circuit 1, respectively. Circuit 2 is generated by allocating *N1, N4, N2, N5, N3, N6* into the left, middle and right side ALU, respectively. The obvious difference between these circuits is that Circuit 1 requires less resources than 2. However, there is a more subtle difference. The two ALUs of Circuit 1 work concurrently during the behavior using a single clock . We notice that Circuit 2 is partitioned into two disjoint subcircuits, shown in Fig. 1(c) by the unshaded and shaded parts. The first subcircuit (unshaded) is active during the *odd* time

[†]This work was partially supported by the Semiconductor Research Corporation (SRC) under Contract No. DJ-527.

[‡]Department of Computer Engineering, Case Western Reserve Univ., Cleveland, Ohio.

[§]Advanced Micro Devices (AMD), Sunnyvale, California.

[¶]Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran.

steps of the behavior, *T1, T3, T5, ...* whereas the second subcircuit (shaded) is active during *even* intervals. We achieved this partitioning by allocating behavioral nodes at even and odd time steps, into separate subcircuits. Since the activity of the two subcircuits occurs at non-overlapping time intervals, it would be possible to use two non-overlapping clocks for the corresponding subcircuits.

The primary motivation for multiple clocking scheme of course would be to reduce power consumption during inactive time slots. This is shown in Fig. 2 illustrating the two non-overlapping clocks, Clock 1 and 2, as they relate to the original Clock. Note that the frequency of the two clocks is *f/2* where *f* is the frequency of the original clock. However, the *effective* frequency of the entire Circuit 2 is the same as Circuit 1, i.e. f. In other words, although the disjoint components of Circuit 2 are clocked at half the frequency of Circuit 1, there is no loss of performance because the effective frequency is the same. At the same time, there is potential for power reduction on the disjoint subcircuits.

In what follows we will consider the potential for power reduction of Circuit 2 by comparison to Circuit 1 operating in two different modes, a) without power management, b) with conventional power management.

2.1 No power management

The potential power reduction can be argued as follows. Let us assume that the AC power consumption of Circuit 1 is given by $P_1 = C_1 V^2 f$, where C_1 is average load capacitance, V is the supply voltage, and f the clock frequency applied. For Circuit 2 we have $P_2 = (C_{21} + C_{22}) V^2 f/2$ where now C_{21} and C_{22} are the load capacitances of the corresponding disjoint subcircuits of Circuit 2. Then, to achieve power reduction in Circuit 2 we need $C_{21} + C_{22} < 2C_1$, which in the above example is quite feasible, confirmed by experimentation. Admittedly, this rationale is approximate because it does not consider a more accurate switching signal activity, nevertheless it provides the motivation for investigating lower power designs based on multiple clocking schemes. For our experimental results we take into account the effect of switching activities in our power estimation.

The idea of multiple clocks of course applies to more than two non overlapping clocks. Thus an implementation of the Fig. 1(a) behavior using three clocks requires *four* ALUs. The resulting circuit consists of three disjoint subcircuits. If the load capacitances are C_{31}, C_{32}, C_{33} then to achieve power reduction under the 3 clocking scheme we would need: $C_{31} + C_{32} + C_{33} < 3\ C_1$ and $2\ (C_{31} + C_{32} + C_{33}) < 3\ (C_{21} + C_{22})$. Of course, there are tradeoffs between power and circuit cost that may apply here.

We should remark at this place that our approach is considerably different from the "duplicating hardware" technique of [12]. The researchers there use frequency reduction together with hardware duplication to achieve power reduction by reducing voltage. In our approach, we use synthesis to avoid duplication when we affect frequency reduction. As shown in the examples of Fig. 1, synthesis has a "capacitive reduction" effect and although hardware is increased when frequency is reduced to $f/2$, the increase is far from duplication.

2.2 Conventional Power Management

The general idea of power management schemes is to turn off circuit units when they are not busy. This is achieved by using gated clocks, i.e. turning off the clock signal of storage elements during their idle cycles. Moreover, it is possible for some extra logic to isolate ALUs so that they will not consume useless combinational power in their off duty cycles.

To compare the power consumption of Circuit 1 and 2, we consider several consecutive computations of the behavior in Fig. 1 such that each computation begins at time slot T_{1+4k}, where $k \geq 0$. For efficiency, we overlap the beginning and end of two consecutive computations on both circuits. Thus, the second computation begins at time slot T_5 and ends at T_9, the 3rd begins at T_9, and so on. On component-by-component basis, Circuit 1 is busy on the average 75% of the time whereas Circuit 2 is busy 50% of the time. For example, the two ALUs of Circuit 1 during the first three computations operate at time slots $T_1\ T_2\ T_3\ T_5\ T_6\ T_7\ T_9\ T_{10}\ T_{11}$ and $T_3\ T_4\ T_5\ T_7\ T_8\ T_9\ T_{11}\ T_{12}\ T_{13}$, respectively. The subtracter of Circuit 2 operates at $T_2\ T_4\ T_6\ T_8\ T_{10}\ T_{12}$.

The power of Circuit 1 under conventional management is $P_1 = 3/4\ C_1\ V^2\ f$. For Circuit 2 we have $P_2 = 1/2\ (C_{21} + C_{22})\ V^2\ f$. Then to achieve power reduction for the 2-clock scheme over the conventional power management technique we need $C_{21} + C_{22} < 3/2\ C_1$. Note it is still feasible to satisfy the above relation, depending on how good is the allocation synthesis for multiple clocking. Actually, a crude power analysis on Circuits 1 and 2 can show that indeed the above condition holds. Ignoring the multiplexer power, and assuming in both circuits ALU capacitance C and register capacitance C_R, then the power difference between Circuit 1 and Circuit 2 is $P_1 - P_2 \approx 3/4\ C_R\ V^2\ f$. So the multiple clocking scheme can indeed improve the power consumption in comparison to the conventional power management technique, and our experimental results confirm this crude analysis.

Another advantage of our scheme is that due to the non-overlapping activity of clock partitions it is possible to use latches instead of registers, which has significant impact on reducing power. In general, it is very difficult to use latches in circuits with conventional power management because of overlapping READs and WRITES, which we avoid.

3. Multiple Clock Scheme Architecture

We now describe our basic RTL architecture model for a multiple clocking scheme. Our model focuses on the datapath part of the RTL structure, however, we also discuss the datapath interactions with the controller concerning important timing issues. This analysis will be the basis of our synthesis method for low power RTL structures using multiple clocks.

3.1 RTL Structural Model

In our model, the basic datapath unit is the *Functional Block (FB)*. As shown in Fig. 3(a), FB consists of 3 layers of components, i.e, 2 Muxes – 1st layer – connected to the two ports of an ALU – second layer – which is connected to one or more memory elements (ME) – 3rd layer. The memory elements could be registers or latches. The control points of the Muxes and the ALU (select points) and the ME (load point) are also shown. Shown also in Fig. 3(a) is the clock line driving the FB registers. The Mux ports and the MEs serve as the data input and output ports of the FB, respectively.

A *Datapath Module (DPM)*, shown in Fig. 3(b), is composed by connecting together a number of FBs using bus lines from the output of one to the input of another. Specifically, every output port of FB$_i$ to one or more input ports of other FBs, possibly FB$_i$ itself. The *external* input and output ports of a DPM are merely some of the input and output ports of its constituents FBs. Moreover, the DPM has a number of control lines each connected to a number of internal control points of the DPM.

From our viewpoint, the most important characteristic of the DPM is that a single clock is used for its internal memory elements (MEs). In our model, a datapath structure consists of a number of interconnected and disjoint DPMs, DPM_1, DPM_2, DPM_3, $\cdots$ driven by the non-overlapping clocks, respectively, CLK_1, CLK_2, CLK_3, $\cdots$

3.2 Timing Relationships and Power

Consider two datapath modules, DPM$_1$ and DPM$_2$, Fig. 4(a). Assuming that their memory elements are latches, the timing relationship of the stored signal values R_1 and R_2 is depicted in Fig. 4(b). Signal R_1 switches only at clock period and it is stable elsewhere. Since the input signals of DPM$_1$, X_1, Y_1 etc, are fed by DPM outputs they follow the same timing pattern as R_1 and R_2. Note that the ALU output F_1 must be stable right before Clock 1, at the latest, for R_1 to make transition correctly. This is illustrated in Fig. 4(b). The combinational delay of the Mux and the ALU should satisfy this timing.

The basic requirements of our multiclock scheme with respect to power consumption on module DPM$_1$ are: a) no storage power during $\tau_2(k)$, and b) no combinational power during $\tau_{12}(k)$. Our design clearly satisfies requirement a). However, for b) to be satisfied there should be no value changes at the combinational inputs of DPM$_1$ during $\tau_{12}(k)$. However, it is not always possible to allocate variables so that all inputs of each DPM will be fed by outputs of the *other* DPMs and not by itself. Nonetheless, we suggest: 1) to extend the life span of a type signal by one cycle; this may require an additional storage element to save the old value; and 2) to make sure that the Mux control input remains unchanged between adjacent clock 2 pulses, specifically during $\tau_2(k) + \tau_2(k+1) = 2/f$ where f is the system clock frequency. This can be achieved by *latching* the control lines coming out of the controller for the above time period, i.e. during adjacent pulses of CLK_1.

4. Synthesis with Power Consideration

We present two allocation techniques, assuming in both that the data flow graph (DFG) schedule has been determined earlier by any scheduling methodology such as in [15]. The first technique is a "split-allocation" approach, i.e uses DFG partitioning based on clock assignments and then proceeds to synthesize each DFG part separately, connecting them at the end. The second method performs allocation in an integrated way taking into account the clock assignment of DFG nodes.

The advantage of the split allocator is that it can be easily adapted by any existing allocator to generate partial datapaths with less power requirements. Of course, the designer needs a clean-up phase to glue independently generated partitions. We will show that this clean-up phase is quite straight-forward and can be done manually or automatically. The second approach has the potential to effect better resource sharing because it performs an integrated allocation of the partitions.

4.1 Split Allocation Method

A designer may prefer his/her own allocation tool. Our split allocation approach can be adapted by any allocator to generate individual partitions. However, there is need for a clean-up phase to interconnect the partitions and remove redundancies.

Briefly speaking, for a given schedule we want to use an existing allocator to generate multiple clock datapath partitions. As explained earlier in detail in section 3 such datapath has much less power consumption than a conventional single-clock datapath while running with the same speed. The basic idea is as follows. Suppose we have n non-overlapping clocks, $CLK_1, CLK_2, \cdots, CLK_n$. Consider a scheduled DFG which we partition into n disjoint sub-DFGs, based on the schedule, $P_1, P_2, \cdots, P_n$ clocked by $CLK_1, CLK_2, \cdots, CLK_n$, respectively. Clearly, the nodes of P_k are all nodes of the original DFG located in steps clocked by the k-th clock. More specifically, the nodes scheduled in time steps t $(1 \leq t \leq T)$ belong to P_k $(1 \leq k \leq n-1)$ if $t \bmod n = k$. Partition P_n contains the nodes of time steps t where $t \bmod n = 0$.

Moreover, we preserve all scheduling information associated with the sub-DFG P_k nodes that they have in the original DFG. This means there will be "internal input" edges coming into P_k from the other partitions, but for the purpose of the split allocation, these edges will be treated as input edges of P_k. Similarly, there are "output" edges of P_k that connect to the nodes of the other partitions. Some edges of P_k may well be primary inputs or primary outputs but it is possible that P_k may not have any primary edges at all.

For the purpose of our method, we map the scheduling steps of the original DFG into *local steps* of the partitions. Thus, node N scheduled in global time step t_{glb} will be mapped to partition $k = t_{glb} \bmod n$, and into local time step $t_{loc} = \lceil t_{glb}/n \rceil$. Conversely, given local time step t_{loc} and partition k then the corresponding global step is $t_{glb} = (t_{loc} - 1)n + k$.

Now, that we have generated all n scheduled sub-DFG's, all we need to do is to "feed" each one to some allocator of our choice. The results will be n datapath modules, DPM_1, $DPM_2, \cdots, DPM_n$ clocked by $CLK_1, CLK_2, \cdots, CLK_n$.

We propose the following process to generate multiple clock datapath partitions. For simplicity, we will describe our method using a non-overlapping two-clock scheme.

- *Step 1* (Partition the Schedule):
  This partitioning is simply done based on odd and even time steps in this example. Fig. 5 (a) and (b) show the original schedule and two partitions based on odd and even time steps. Note that the time steps $1', 2', 3'$ and $1'', 2''$ are *local* time steps in each partition to show the local sequencing. Also, we consider primary inputs and outputs for those edges cut based on partitioning boundaries and their life span in the original schedule.
- *Step 2* (Run Allocator on Partitions):
  In this step the designer runs an allocation method of his/her choice on the two schedules, independently. The allocator considers the local time steps as real ones and uses the library and its internal optimization methods (e.g. for ALU selection, REG minimization and MUX/BUS collapsing)[15]. The output of this step will be two datapath partitions, shown in Fig. 5(c), realizing two schedules.
- *Step 3* (Remove Redundancies):
  To merge two datapath partitions to have a single datapath with multiple clocking, we need to remove the redundancies and also establish necessary but missing connections. We look at the primary inputs of the original schedule. If some of them are used in both partitions, we have a register or a port in both partitions which can be merged. Also, those variables which were introduced as primary inputs/outputs in the partitions but were not really primary inputs/outputs (e.g. u or v) do not necessarily need a register. So, the register should be removed and a connection should be establishes instead. Finally, our clean-up phase also involves splitting those variables, merged in the same memory element, having READ and WRITE conflicts so that they would be re-allocated into different *latches*. Fig. 5(d) shows the final result at the end of step 3.

4.2 Integrated Allocation Method

In this allocation method, our approach is to use the registers to control transitions on the other components of the partition like the alus and muxes. In other words, the transitions on the registers cause transitions on the Muxes and ALUs, but if no transitions occur on the registers then no transitions occur throughout the partition. Because of this restriction, we must trace through the schedule and make sure each operand of all operations are in the same partition. If one operand is in one partition say β and the other is in partition α, then we must transfer the value from partition β to partition α by using a temporary variable to hold that value in α. This can be seen in Fig. 6(a) where the the addition operation has inputs E and X. The variable X is written in time step 1 which is partition β and the variable E is written in time step 2 which is partition α. As you can see in the figure we create a temporary variable T which saves the value that was in X at time step 2 which forces the operands of the subtraction to both be from partition α. It is basically just a transfer between partitions.

The allocation of registers is mainly affected by the new scheme in that only variables which are placed in the same partition may be merged into the same register. With the alternating clocking scheme, the variables are placed in the partition(p) such that $p = k \bmod n$, where k is the time step the variable is written. In Fig.6(a), since we deleted the READ for X in time step 3 we can merge variables U and X since they were both written in partition β and their lifespans are disjoint. Also, since we are using latches in our implementation, only variables with completely disjoint life spans (non overlapping READs and WRITEs) may be merged. Both of these considerations can be seen in the lifetime analysis of the variables, then using the left edge algorithm, these variables are merged to specific registers.

The allocation of ALUs is done by using the schedule and number of partitions to determine which partitions the ALUs are allocated to. We merge the operations according to the partition(p) in which they are placed; where $p = l \bmod n$ and l is the time step of the operation. As in Fig. 6(b) you can see that only the additions can be merged since these two are in the same partition β. Whereas, the subtraction can not be merged with either of the additions, because it is in partition α. Because we want the transitions of the ALU to occur during a certain time period, we must consider the registers which feed the ALUs or Muxes feeding the ALU. Therefore, all operands of the ALUs must come from the same partition; otherwise, we must use the control lines of the muxes to make the transition occur during the correct clock period or add extra registers to hold the operands.

The allocation algorithm can be summed up in these steps:

- *Step 1:* Trace through the schedule and create temporary variables for operations which have operands in different partitions.
- *Step 2:* Merge variables of the same partition into registers using the left edge algorithm.
- *Step 3:* Use an iteratively greedy method to merge operations according to their partition. If the operands of the ALUs are not in the same partition, then either create registers to transfer values into or use the control on the Muxes to force transitions to occur during the correct time period.
- *Step 4:* Create the Muxes necessary to complete the data path decided by the register and ALU allocation.

For illustration, the result of this algorithm is shown in Fig. 7. Note that in this case we did not latch the control inputs (for example for MUXes), as we discussed earlier, forwarding a result within one partition creates redundant power consumption. For example, the subtracter consumes power at time step 3 because we change its input data. To save power, we could have forced the inputs to the subtracter (y) to change its value at the end of time step 3 (similar to u). To assure this, we remove internal partition transfers by forwarding a register to another register controlled by the second clock.

5. Experimentation and Results

The integrated allocation algorithms described in the previous sections has been implemented in C on a SUN SPARC-IPC workstation. Unfortunately, most of the work on power optimization and estimation is at the logic level and we can not compare our RTL results to them. However, to show the effectiveness of using multi-clock datapath synthesis we have implemented some of high-level synthesis benchmarks using COMPASS CAD system [19]. In this section, we first explain how we obtained the datapaths and used COMPASS CAD toolset to estimate the power and then show the actual data for high-level synthesis benchmarks.

We knew our analysis were sound and that if the additional components did not cause too much extra capacitance we could achieve at the best 50% power by adding two clocks and at best 33% power by adding three clocks. Although we knew that these reductions would be too much to expect, we did achieve substantial reductions of power at the expense of area.

	Power [mW]	Area [λ^2]	ALUs	Mem. Cells	Mux In's
Conven. Alloc. (Non-Gated Clock)	9.85	2680425	1(*+),1(&+), 1(-|),1(/)	8	10
Conven. Alloc. (Gated Clock)	6.92	2383553	1(*+),1(&+), 1(-|),1(/)	8	10
1 Clock	7.39	2668365	1(+-&),1(*+|), 1(/)	10	12
2 Clocks	6.41	2552425	2(+), 1(/), 1(-|),1(*&)	10	12
3 Clocks	3.52	2484873	1(+&),1(-),1(*) 2(+),1(/),1(|)	14	4

Table 1: Multiple Clocks with Latches for the FACET

5.1 Experimental Setup

To validate our idea we used our allocation procedure to produce VHDL code representing the partitioned circuit. The circuits were synthesized using the COMPASS ASIC Synthesizer based on 0.8 micron CMOS library [18] on a SPARC-IPC workstation with 32M RAM. After our 4-bit circuits were synthesized we needed to compute the power consumed by the circuit. This was accomplished by simulating the circuit in the COMPASS simulator with the power option enabled. The power option works by counting the transitions on each node and computing an average frequency f over a given amount of time T. Using the formula $P = fC_LV^2$ with V being 4.65V for all experiments and C_L being the loading capacitance on the particular node, the tool finds the power for each node. Then, summing over all nodes in the circuit after time T the simulator reports the average power consumed by the circuit.

We computed the power for a circuit by simulating the circuit with a large number of random inputs, and having the tool report the power after all patterns were executed. In other words, we allowed the simulator to find the average transitions on nodes over a long period of randomly inputed data. The area was simply computed by allowing COMPASS to layout the circuit and report the size. Keep in mind that the area for the conventional allocation models uses D-flip flops for registers as opposed to our method which uses latches.

5.2 Experimental Results

In this section we present our results for four example benchmarks generated by our allocation scheme as compared to the conventional allocation scheme. The conventional datapaths were generated by SYNTEST [15]. The results are tabulated in Tables 1 to 4. Each table reports on five different designs of the same example: conventional allocation using non-gated single clock, conventional allocation using gated single clock, commonly used in industries [10]; and three datapaths based on latch implementation[*].

In Table 1 we show our results for the FACET example[14]. As expected, there is a reduction of power as the number of clocks increases, from 6.92 mW (for the conventional gated clocks) to 3.52 mW (for 3 clock method) which is 49%. Also, notice that for our scheme with one clock the power has increased as compared to the Conventional with Gated Clocks. This is because of the increase of area with no reduction of the frequency. Also, interesting about this example is the decrease in area from 1 to 2 and 2 to 3 clocks. This is because the multifunction ALUs being used for the different clocking scheme. The COMPASS synthesizer does not reduce logic as well for most multifunction ALUs, as opposed to the (+-) which reduces very well. The schedule can also help the area reduction. The 3 clock scheme suits the particular schedule better than the 2 clock scheme because of ALU utilization.

In Table 2 we show our results for the HAL example[13]. There is a continual decrease in power and an increase in area.

[*]Note that the difference between the conventional gated clock technique and the 1-clock implementation of our scheme is in that we use the same allocation technique that avoids concurrent READs and WRITEs, without using clock partitions.

	Power [mW]	Area [λ^2]	ALUs	Mem. Cells	Mux In's
Conven. Alloc. (Non-Gated Clock)	12.48	3080133	1(+),1(*-), 1(*+),1(*->)	8	10
Conven. Alloc. (Gated Clock)	8.12	2819025	1(+),1(*-), 1(*+),1(*->)	8	10
1 Clock	5.61	2627484	1(*>-),2(*), 1(+-)	12	20
2 Clocks	4.98	2901501	3(*),1(+-), 1(->),1(*+)	14	20
3 Clocks	3.73	2954465	1(*-),5(*), 1(+),1(>)	17	8

Table 2: Multiple Clocks with Latches for the HAL

	Power [mW]	Area [λ^2]	ALUs	Mem. Cells	Mux In's
Conven. Alloc. (Non-Gated Clock)	18.65	5118795	3(*+),1(*-), 1(*+-)	18	35
Conven. Alloc. (Gated Clock)	11.49	4826283	3(*+),1(*-), 1(*+-)	18	35
1 Clock	11.31	5126718	3(*+),1(*+-), 1(*-)	20	47
2 Clocks	9.24	5194451	4(*+),1(*), 1(-),1(*-)	20	56
3 Clocks	7.19	5327823	4(*+),1(*), 3(+-),1(-),1(+)	26	45

Table 3: Multiple Clocks with Latches for the Biquad Filter

The exception is from the Conventional Gated to the 1 Clock where there is a decrease in power and area. This can be explained by remembering that we are using latches in our scheme and by looking at the ALU allocation. The ALU allocation for our one clock scheme is better because one of the multifunction ALUs is the (+-) which synthesizes well, compared to other multifunction ALUs. The important results here are the 8.12 mW (for the conventional gated clocks) to 3.73 mW (for our 3 clock method) or 54% reduction in power versus 5% increase in area.

In the remaining Tables we examine our results for the Biquad Filter example and the Band Pass Filter example[16][17]. They are both similar to the previous results in terms of justification. The Biquad Filter goes from 11.49 mW (for the conventional gated clocks) to 7.19 mW (for our 3 clock method), or 37% power reduction versus 9% increase in area. The Band Pass Filter goes from 8.87 mW (for the conventional gated clocks) to 5.78 mW (for our 3 clock method), or 35% power reduction versus 12% increase in area.

There is an obvious trade-off between the amount of power reduction and the amount of area increase. As you can see the area is increased as the number of clocks increases, but the power is still being reduced. Although you can not see in our results it is obvious that you can not keep adding clocks and expect power reduction. There are diminishing returns in this scheme, because as the area grows the capacitance will grow to offset the reduction of frequency accomplished by our method.

	Power [mW]	Area [λ^2]	ALUs	Mem. Cells	Mux In's
Conven. Alloc. (Non-Gated Clock)	18.01	5588975	2(+-),1(*)	23	39
Conven. Alloc. (Gated Clock)	8.87	4181238	2(+-),1(*)	23	39
1 Clock	7.39	3049956	1(*),1(-), 1(+)	15	50
2 Clocks	6.15	3729654	2(*),1(+-), 1(-),2(+)	19	57
3 Clocks	5.78	4728731	3(*), 3(-), 3(+)	25	66

Table 4: Multiple Clocks with Latches for the Band Pass Filter

6. Summary

We have presented an effective technique of low power design for RTL circuits and microarchitectures. The basic idea is to operate each module only during its corresponding duty cycle, thus clocking each module by a frequency f/n to reduce power. However, the overall effective frequency of the circuit remains f the single clock frequency. We have observed very encouraging results of power savings up to 50%.

References

[1] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1993.

[2] D. Dahl, "Designing High Performance Systems to Run from 3.3 V or Lower Resources," Silicon Valley Personal Computer Conf., 1991.

[3] A. Chandrakasan, S. Sheng and R. Brodersen, "Low Power CMOS Digital Design," in *IEEE Journal of Solid State Circuits.*, April 1992.

[4] Y. Tamiya, Y. Matsunaga and M. Fujita, "LP based Cell Selection with Constraints on Timing, Area and Power Consumption," in *Proc. ICCAD Conf.*, Nov. 1994.

[5] K. Yano et al., "A 3.8 ns CMOS 16 multiplier using Complimentary Pass Transistor Logic," in *IEEE Journal of Solid State Circuits.*, April 1990.

[6] T. Callaway and E. Swartzlander, "Optimizing Arithmetic Elements for Signal Processing," *IEEE VLSI Signal Processing Workshop*, Oct. 1992.

[7] A. Shen, A. Ghosh, S. Devedas and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," in *Proc. ICCAD Conf.*, June 1992.

[8] S. Iman and M. Pedram, "Multi-Level Network Optimization for Low Power," in *Proc. ICCAD Conf.*, Nov. 1994.

[9] G. Hachtel, M. Hermida, A. Pardo, M. Pocino and F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation," in *Proc. ICCAD Conf.*, Nov. 1994.

[10] L. Benini, P. Siegel and G. De Micheli, "Saving Power by Synthesizing Gated Clocks for Sequential Circuits," in *IEEE J. of Design and Test of Computers*, 1994.

[11] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey and R. Brodersen, "Optimizing Power Using Transformations," in *IEEE Trans. on CAD*, Jan. 1995.

[12] C. Piguet, J. Masgonty, V. Kaenel and T. Schneider, "Logic Design for Low-Voltage/Low-Power CMOS Circuits," in *Proc. ISLPD'95 Symposium*, April 1995.

[13] P. G. Paulin and J. P. Knight, "Forced-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. Computer-Aided Design*, vol. 8, No. 6, June 1989.

[14] C. Tseng and D. P. Siewiorek, "FACET: A Procedure for the Automated Synthesis of Digital Systems," in *Proc. 20th design Automation Conf.*, June 1983, pp. 566-572.

[15] H. Harmanani, C. Papachristou, S. Chiu and M. Nourani, "SYNTEST: An Environment for System-Level Design for Test," *European Design Automation Conference (EURO-DAC)*, (Hamburg, Germany), Sept. 1992.

[16] B. D. Green and L. E. Turner, "New Limit Cycle Bounds for Digital Filters," *IEEE Trans. Circuits and Systems*, April 1988, pp. 365-374.

[17] S. Y. Kung, H. J. Whitehouse and T. Kailath, *VLSI and Modern Signal Processing*, Prentice-Hall, 1985.

[18] VLSI Technology, "0.8-Micron CMOS VSC450 Portable Library," VLSI Technology, Inc., 1993.

[19] Compass Design Automation, "User Manuals for COMPASS VLSI V8R4.7," Compass Design Automation, Inc., 1995.

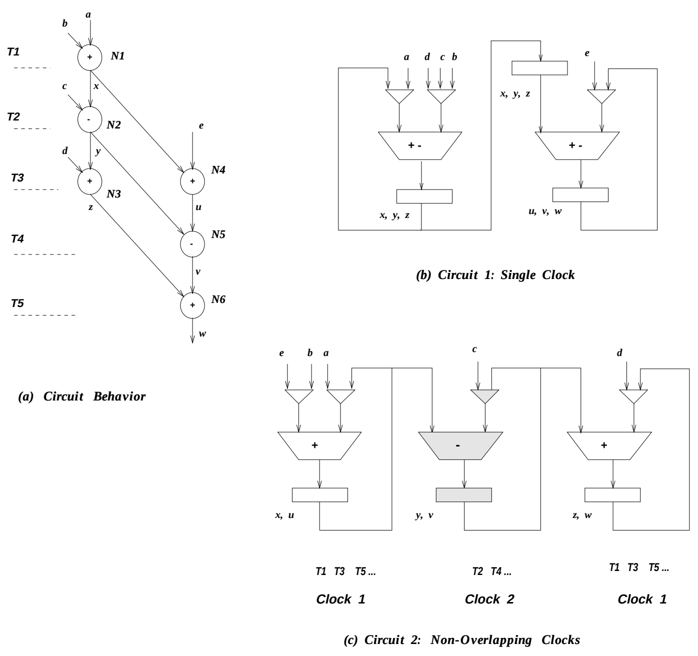

(a) Circuit Behavior
(b) Circuit 1: Single Clock
(c) Circuit 2: Non-Overlapping Clocks
Clock 1
Clock 2
Clock 1

Figure 1: Two Circuit Designs: Minimal Component and Lower Power

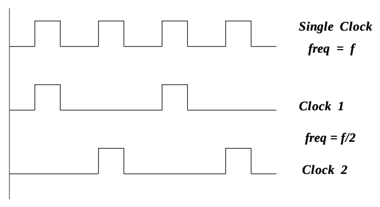


Figure 2: Multiple Clocking Scheme

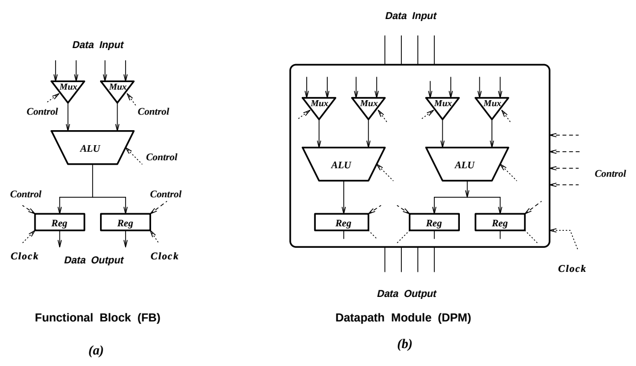


Figure 3: Functional Block and Datapath Module

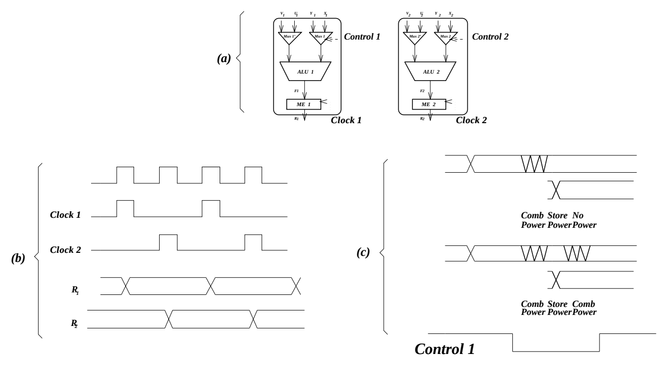


Figure 4: Timing Relationships of Data Path Modules in multiple clocking scheme

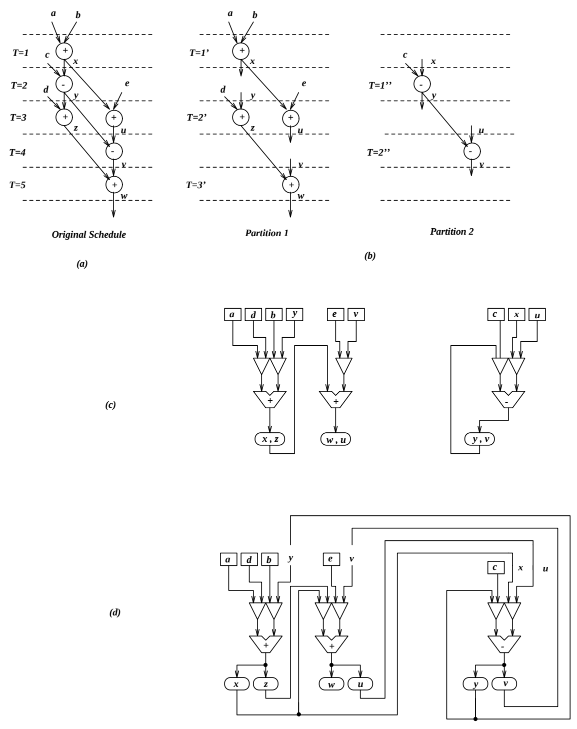
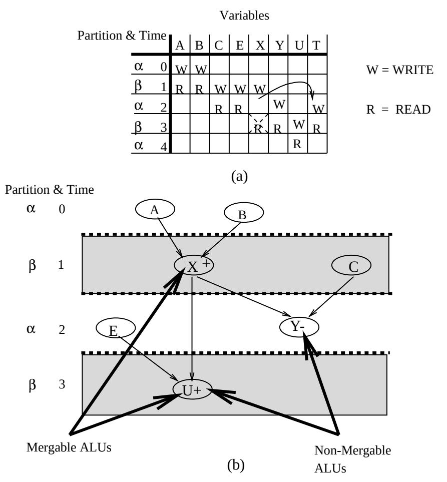


Figure 5: Example of Split Allocation for Multiple Clock Datapath Partitions



Variables

Partition & Time	A	B	C	E	X	Y	U	T
α 0	W	W						
β 1	R	R	W	W	W			
α 2			R	R		W		W
β 3					R	R	W	R
α 4							R	

W = WRITE

R = READ

(a)

Figure 6: Life time analysis of READs and WRITEs in allocation

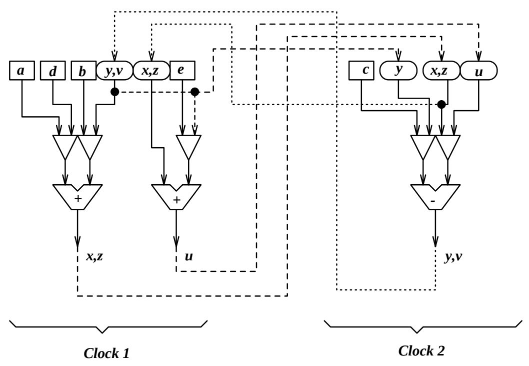


Figure 7: Example of the Integrated allocation