A Multiple Clocking Scheme for Low Power RTL Design[†]

Christos Papachristou Mark Spining Mehrdad Nourani

Department of Computer Engineering
Case Western Reserve University
Cleveland, OH 44106 USA

Abstract

This paper presents an effective multiple clocking scheme for lower power RTL circuit design. The basis is to partition a behavioral description of the circuit into m modules fed by n non-overalapping clocks. The idea is is to operate each module only during its corresponding duty cycle at a frequency f/n to reduce power. power. However, the overall effective frequency of the circuit remains f, the single clock frequency. The scheme uses a resource allocation algorithm to synthesize clock module partitions at the RTL. The allocation avoids combinational power consumption during the off duty cycles of each module.

The method has been implemented on Sparcstation machines using a commercial CAD tool which includes an activity-based power estimator. Experimental results and comparisons of the multiple clocking scheme to single clock (gated and ungated) designs are reported, entailing power savings up to 50%.

1 Introduction

The increasing demand for battery-operated devices, such as cellular phones and notebook computers, requires a special attention to *power* instead of *speed* which is a typical concern in conventional VLSI design. However, optimizing for power requires a second look at the entire VLSI design process, meaning technology, architectures and even algorithms.

There are two sources of power dissipation in CMOS [1]:
1) Static dissipation mainly due to leakage current.
2) Dynamic dissipation due to: a) switching transient current; b) 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 improvement. 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 cost 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 in Silicon. 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 three levels that power consumption methods have been investigated: 1) transistor , 2) logic ; and 3) register-transfer level or architecture. At the transistor level, the objective is to optimize the transistor size, by careful cell selection, thus reducing the parasitic capacitances of transistors and interconnect routings [3]. The work in [4] presents an LP-based algorithm to find the best cell combination to reduce power consumption. A pass transistor logic family was found to minimize the capacitance and introduced in [5] as an alternative to conventional CMOS logic family.

Much work for power optimization has focused at the logic level. Several logic synthesis optimization techniques have been proposed [6] to lower the power consumption. In [7] power is minimized by modifying the function of each node in the circuit. Re-encoding of a sequential circuit [8] and using gated clocks [9] are two techniques for power reduction in sequential circuits.

Reducing power at levels higher than logic has been recently attempted. A high-level synthesis system, HYPER-LP, presented in [10] uses a variety of architectural transformations to estimate and optimize the power dissipation.

In this paper we present an effective multiple clocking scheme for lower power RTL design. The main 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. The working frequency running each partition module is f/n, where f is the single clock frequency, however, the effective frequency of the entire circuit remains f. This means the inactive partitions are literally "turned off" during their off duty cycle to reduce power dissipation. Careful selection of partitions during RTL design assures achieving the same performance as in the single clock datapath. 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 an example illustrating the basic idea of our work. Section 3 describes RTL modeling and analysis of the multiple clocking scheme. Section 4 presents our synthesis approach to low power. Experimental results are in section 5, and the conclusion 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, *T1, T2, ..., T5*. In Figs 1(b) and (c) are two RTL circuit implementations of this behavior, Circuit 1, and Circuit 2, respectively.

[†]This work has been supported in part from contracts OAI 94-1-015 and SRC DJ-527.

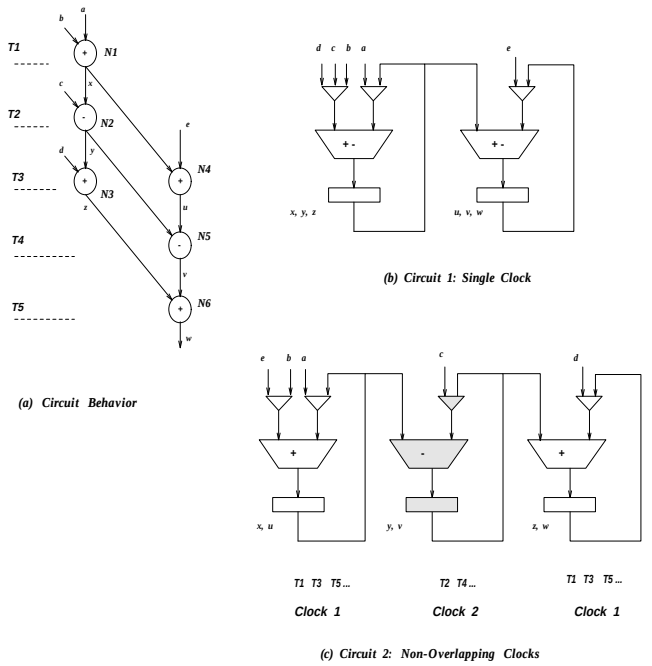

Figure 1: Two Circuit Designs: Minimal Component and Lower Power

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* and *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 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.

There are two important remarks that we should make.
1) The frequency of the two clocks is *f/2* where f is the frequency of the original clock.
2) 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.

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 sub-circuits 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 experimetation (see Results section). 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 results we take into account the effect of switching activities in our power estimation.

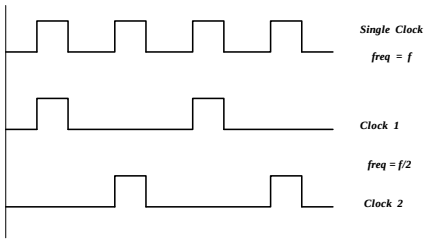

Figure 2: Multiple Clocking Scheme

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: $C_{31}+C_{32}+C_{33} < 3C_1$ 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.

3 Analysis of Multiple Clock Scheme

We now describe our basic RTL architecture model for a multiple clocking scheme. We focus on the datapath part of the RTL structure, however, we also discuss the datapath interactions with the controller [20] concerning important timing issues. The analysis will be the basis of our synthesis method.

• **RTL Structural 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 of 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 MEs.

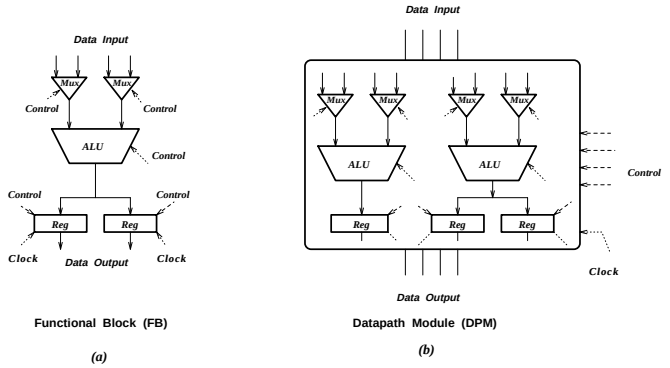

Figure 3: Functional Block and Datapath Module

A datapath structure consists of a number of interconnected and disjoint DPMs, $DPM_1, DPM_2, DPM_3, ...$ driven by the non-overlapping clocks, respectively, $CLK_1, CLK_2, CLK_3, ...$ This architecture model is shown in Fig. 4 for a 3-clock scheme. The basic premise of this scheme is to consume power only in one DPM at a time during its corresponding clock period, while the other DPMs are essentially "turned off".

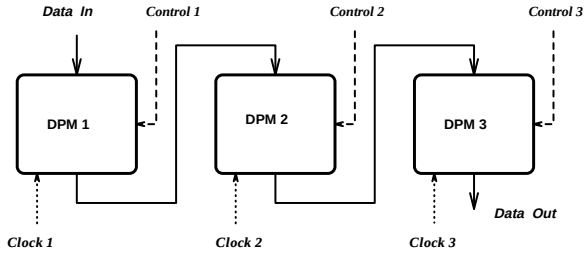

Figure 4: Multiple Clock data path Module architecture

• **Timing Relationships and Power**. Consider two datapath modules, DPM_1 and DPM_2, Fig. 5(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. 5(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. 5(b). The combinational delay of the Mux and the ALU should satisfy this timing.

Let $\tau_{12}(k)$ denote the time interval between pulse k of CLK_1 and k of CLK_2 ; let $\tau_{21}(k)$ be the time interval between pulse $k-1$ of CLK_2 and pulse k of CLK_1. Also, $\tau_1(k)$ denotes the k-th pulse of CLK_1, and $\tau_{11}(k) = \tau_{12}(k) + \tau_2(k) + \tau_{21}(k)$ the time interval between the pulses k and $k+1$ of CLK_1. This notation is shown in Fig. 5(b).

Module DPM_1 consumes power on its memory element for WRITE, $\tau_1(k)$ and on its combinational part, Muxes and ALU. The power cycle of DPM_1 is as follows. Combinational power begins consumption after pulse 1 of Clock 2 until F_1 stabilizes, Storage power is consumed during pulse 1 of Clock 1. 5(c). In general, combinational power is consumed during $\tau_{21}(k-1)$ and storage power during $\tau_1(k)$, $k \geq 1$.

The basic requirements of our 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 $\tau_{12}(k)$. This may or may not occur for DPM_1 depending on the type of input signals, type (I) or (II) Fig. 5(d). Note type (I) signals occur when the input is fed by different DPM modules (different clocks). For type (II) the corresponding input is fed by outputs of the same DPM. Although type (I) signal always satisfies our requirement, type (II) is not always unsatisfactory; only whenever changes of value at Clock 1, $\tau_1(k)$, will cause combinational power consumption during $\tau_{12}(k)$. Our scheduling of the computational data flow provides this information of value changes concerning every variable and this is used by our allocation method (next section). 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 the following design rules which make sure that type (II) signals do not change value at clock 1, $\tau_1(k)$.
Rule 1. Extend the life span of a type (II) signal one cycle; this may require an extra storage element to save the old value.
Rule 2. Ensure 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, 5(c).

• **Latches versus Registers**. The memory elements mentioned above pertain to registers or latches. The advantage of using latches is that they consume significantly less power than registers and they are faster. However, in single clock systems, registers have significant advantages over latches, regarding timing issues [11]. At the RTL, registers (edge-triggered or master-slave) allow concurrent READs and WRITEs and this property significantly improves resource sharing at the RTL. In non overlapping multiple clocking schemes it is quite possible to use latches provided the following basis requirement is satisfied [9]: $\Delta T_{Comb} \leq 2/f$ where f is the system clock frequency (single clock).

4 RTL Synthesis with Low Power

The idea behind the lower power allocation is to force components in the same partition to make transitions only during the clock period dedicated to the particular partition. There are two factors which force transitions, one is the control signals and the other is the data being fed to the partition. Therefore, by taking advantage of both of these considerations we can allocate our circuit into n partitions. We have shown that by a careful partitioning of components and forcing the registers feeding the components to hold their old value as long as possible we can reduce the power consumption effectively. In this methodology, we use n non overlapping clocks, each assigned to a distinct partition. To apply such a methodology for RTL synthesis we must use special RTL synthesis techniques because conventional RTL synthesis (scheduling and allocation) does not consider savings in input transition (and thus power)

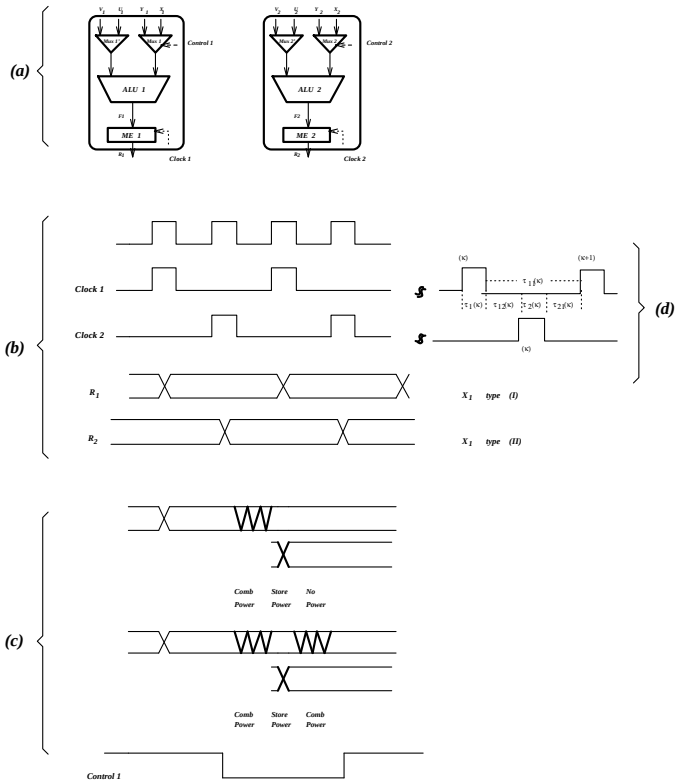

Figure 5: Timing Relationships of Data Path Modules in multiple clocking scheme

of components during clock cycles they do not operate.

In this section we describe our RTL allocation scheme which not only implements partitions but also minimizes the input transitions reducing power. We assume that the data flow graph (DFG) schedule has been determined earlier by any scheduling methodology such as in [14]. Our 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 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 is quite straight-forward and can be done manually or automatically.

• **Split Allocation for Multiple Clocking**. There are many conventional datapath allocation methods in the CAD community with their own pluses and minuses. Most likely a designer would 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.

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 $i \bmod k = j$. 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 will be mapped to partition $k = t \bmod n$, and into local time step $j = \lceil t/n \rceil$. Conversely, given local time step j and partition k then the corresponding global step is $t = (j-1)n + k$.

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, $\mathrm{DPM}_1, \mathrm{DPM}_2, \cdots, \mathrm{DPM}_n$ clocked by $CLK_1, CLK_2, \cdots, CLK_n$.

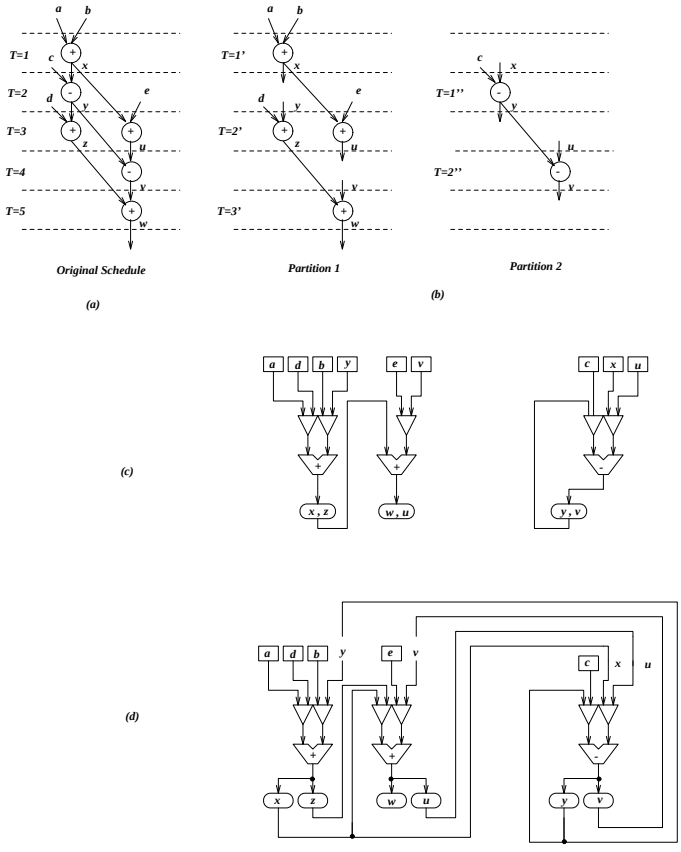

Figure 6: Example of Split Allocation for Multiple Clock Datapath Partitions

We describe our multiple clock datapath partitioning process using a non-overlapping 2-clock example.

Step 1 (Partition the Schedule):
This partitioning is simply done based on odd and even time

steps in this example. Fig. 6 (a) and (b) show the original schedule and two partitions based on odd and even time steps. Note the following observations for the partitions:
– Time steps $1', 2', 3'$ and $1'', 2''$ are *local* time steps in each partition to show the local sequencing.
– Consider primary inputs and outputs for those edges cut based on partitioning boundaries and their life span in the original schedule. For example, in partition 1 y is considered to be a primary input. x does not need to be introduced as primary output in partition 1 because there is no operation in even time steps (second partition) which consumes x *after* the last time that it is used in odd time steps (i.e. by u in time step 3). These assumptions will create additional registers which will be removed in the clean-up phase (i.e. Step 3).

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)[14]. The output of this step will be two datapath partitions, shown in Fig. 6(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 establish necessary but missing connections as follows:
– 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. For example if a is used in both partitions and allocator created one register in each partion for it we can remove one of them and connect the other one by a wire instead.
– Those variables which were introduced as primary inputs or outputs in the partitions but were not really primary (e.g. u or v) do not necessarily need a register. So, the register should be removed and a connection should be establishes instead.
– 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*. For example, in Fig. 6 (c), variables x and z have been merged into the same memory element by the allocator. However, x and z have READ and WRITE conflict thus they can not be stored into a single latch, resulting in a two latch implementation shown in Fig. 6 (d). Fig. 6(d) shows the final result at the end of step 3.

5 Experimentation and Results

The allocation algorithm has been implemented in C on a SUN SPARC-IPC workstation. To show the effectiveness of using multi-clock datapath synthesis we have implemented some of high-level synthesis benchmarks using COMPASS CAD system [18]. In this section, we first explain our experimental setup for datapath design and power estimation, and then show the actual data for high-level synthesis benchmarks.

• **Experimental Setup**. To synthesize the circuits we used first SYNTEST [14], a high level synthesis system producing RTL datapaths which schedules an input circuit description in behavioral VHDL. Then we applied our allocation technique to synthesize multiclock circuit partitions in RTL. Finally we synthesized the RTL circuit using a commercial tool, the COMPASS ASIC Synthesizer, based on 0.8 micron CMOS library [17]. We computed the circuit power using the power option of COMPASS which 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 overall average power consumed by the circuit. We simulated 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 random patterns were only fed to the data lines, since we wanted to know the power consumed while under operating conditions; as opposed to the testing method where you feed random patterns to all lines looking for fault coverage. Note that the ordering of inputs is importantwhen considering transitions. However, we verified experimentallly that if the number of patterns is large any set of random patterns will converge to the same result.

• **Experimental Results**. We present our results for four example benchmarks generated by our allocation scheme as compared to a conventional allocation scheme, without power consideration. The results are tabulated in Tables 2 to 5. 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 [9]; and three datapaths based on latches.

In Table 2 we show our results for the FACET example[13]. 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, 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, it is interesting the decrease in area from 1 to 2 and 2 to 3 clocks. This is best explained because of the multifunction ALUs being used for the different clocking scheme. The COMPASS tool does not reduce logic as well for most multifunction ALUs, as opposed to the (+-) which reduces very well. The role of the schedule can also help explain the area reduction. The 3 clock scheme suits the particular schedule better than the 2 clock scheme because of ALU utilization.

In Table 3 we show our results for the HAL example[12]. The results are mostly typical in that there is a continual decrease in power and an increase in area. 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 – again the (+-) ALU 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 the 5% increase in area.

In the remaining Tables we examine our results for the Biquad Filter example and the Band Pass Filter example[15][16]. 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),

	P (mW)	Area	ALUs	R	M
Non Gated	9.85	$2680425\lambda^2$	1(-|), 1(*+), 1(&+), 1(/)	8	10
Gated	6.92	$2383553\lambda^2$	1(*+), 1(&+), 1(*+), 1(-|), 1(/)	8	10
1 CLK	7.39	$2668365\lambda^2$	1(*+|), 1(+-&), 1(/)	10	12
2 CLK	6.41	$2552425\lambda^2$	1(-|), 2(+) 1(/), 1(*&)	10	12
3 CLK	3.52	$2484873\lambda^2$	1(+&), 1(-), 1(*), 2(+), 1(/), 1(|)	14	4

Table 1: Multiple Clocks with Latches for the FACET example

	Power	Area	ALUs	Mem	Mux
Non Gated	12.48mW	$3080133\lambda^2$	1(+), 1(*-), 1(*+), 1(*->)	8	10
Gated	8.12mW	$2819025\lambda^2$	1(+), 1(*-), 1(*+), 1(*->)	8	10
1 CLK	5.61mW	$2627484\lambda^2$	2(*), 1(+-) 1(*>-)	12	20
2 CLK	4.98mW	$2901501\lambda^2$	3(*), 1(+-), 1(->), 1(*+)	14	20
3 CLK	3.73mW	$2954465\lambda^2$	1(*-), 5(*), 1(+), 1(>)	17	8

Table 2: Multiple Clocks with Latches for the HAL example

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.

6 Summary

We have presented an effective technique of low power design for RTL circuits. The basis of our technique is a) to use a multiple clocking scheme of n non-overlapping clocks, b) to partition the circuit into disjoint modules and assign each module to a distinct clock. The 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. Our technique generates the module partitioning by means of a resource allocation algorithm which uses behavioral and scheduling information. And also makes sure that there is no combinational power consumed by any module during its off-duty cycle time. Our method avoids READ-WRITE conflicts of behavioral variables during the allocation process, thus it is suitable for latch-based design, to further reduce power.

Our method has been implemented in C on Unix Sparcstations. For power estimation we perform simulation of circuit transitions at the logic level using a commercial power estimator from COMPASS. We have conducted extensive experimentation and comparisons of our scheme, specifically, implementations single clock circuits with ungated and with gated clocks, based on registers, to 1-clock, 2-clock and 3-clock implementations by our method, based on latches. We have reported power savings up to 50% of the multiple clocking circuits.

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," IEEE Journal of Solid State Circuits., April 1992.

	Power	Area	ALUs	Mem	Mux
Non Gated	18.65mW	$5118795\lambda^2$	3(*+), 1(*-), 1(*+-)	18	35
Gated	11.49mW	$4826283\lambda^2$	3(*+), 1(*-), 1(*+-)	18	35
1 CLK	11.31mW	$5126718\lambda^2$	3(*+), 1(*+-), 1(*-)	20	47
2 CLK	9.24mW	$5194451\lambda^2$	4(*+), 1(*), 1(-), 1(*-)	20	56
3 CLK	7.19mW	$5327823\lambda^2$	4(*+),3(+-), 1(*), 1(-), 1(+)	26	45

Table 3: Multiple Clocks with Latches for the Biquad Filter

	Power	Area	ALUs	Mem	Mux
Non Gated	18.01mW	$5588975\lambda^2$	2(+-), 1(*)	23	39
Gated	8.87mW	$4181238\lambda^2$	2(+-), 1(*)	23	39
1 CLK	7.39mW	$3049956\lambda^2$	1(*), 1(-), 1(+)	15	50
2 CLK	6.15mW	$3729654\lambda^2$	2(*), 1(+-), 1(-), 2(+)	19	57
3 CLK	5.78mW	$4728731\lambda^2$	3(*), 3(-), 3(+)	25	66

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

[4] Y. Tamiya, Y. Matsunaga and M. Fujita, "LP based Cell Selection with Constraints on Timing, Area and Power Consumption," ICCAD 94.

[5] K. Yano et al., "A 3.8 ns CMOS 16 multiplier using Complimentary Pass Transistor Logic," IEEE Journal of Solid State Circuits., April 1990. IEEE VLSI Signal Processing Workshop, Oct. 1992.

[6] A. Shen, A. Ghosh, S. Devedas and K. Keutzer, "On Average Power Dissipation and Random Pattern Testability of CMOS Combinational Logic Networks," ICCAD 92.

[7] S. Iman and M. Pedram, "Multi-Level Network Optimization for Low Power," ICCAD 94.

[8] G. Hachtel, M. Hermida, A. Pardo, M. Pocino and F. Somenzi, "Re-Encoding Sequential Circuits to Reduce Power Dissipation," ICCAD 94.

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

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

[11] E. J. McCluskey, Logic Design Principles, Prentice-Hall, 1986.

[12] P. G. Paulin and J. P. Knight, "Forced-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans. CAD vol. 8, No. 6, June 1989, pp. 661-679.

[13] C. Tseng and D. P. Siewiorek, "FACET: A Procedure for the Automated Synthesis of Digital Systems," DAC-83.

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

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

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

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

[18] Compass Design Automation, "User Manuals for COMPASS VLSI V8R4.6," Compass Design Automation, Inc., 1994.

[19] R. Brodersen, A. Chandrakasan and S. Sheng, "Low-Power Signal Processing Systems," IEEE VLSI Signal Processing Workshop, Oct. 1992.

[20] C. Su, C. Tsui, A. Despain, " Saving Power in the Control Path of Embedded Processors," IEEE Design and Test, pp. 24-30, 1994.