Clock Frequency Selection for Power-optimized Bus-based SoC Designs

Abstract

In this paper we propose a clock frequency selection algorithm that optimizes system-on-chip bus power consumption under latency and area constraints. Experimental results for the AMBA Advanced High-performance Bus Protocol (AHB) show an average of 35% power consumption reduction, compared to clocking the bus and processor at the same frequency.

1 Introduction

In recent years the advances in VLSI and materials technology have enabled the integration of several instruction processors and peripherals on a single chip. The Advanced Microcontroller Bus Architecture (AMBA) and, more specifically,its Advanced High-performance Bus protocol (AHB) [1] are widely adopted standards for SoC communication. The adoption of protocols such as AHB facilitates right-first-time development, encourages modular design, and makes the process of testing easier. AHB is a synchronous protocol that requires that the bus clock is routed to all interfaces connected to the bus.

As the SoC sizes increase, system clock distribution network wires increase in number and length, resulting in higher driven capacitance. Moreover, capacitance also increases in proportion to the number of interfaces connected to the system bus. The power consumption of the clock distribution network and system bus are increasing in proportion to this (often unnecessary) growth in driven capacitance and clock frequency.

In many cases the clock distribution network can be the largest contributor to total integrated circuit (IC) power consumption [2, 3, 15] . Many approaches aim at reducing clock power consumption by reducing voltage swing [2] or clock load [4]. The approach presented in this paper is orthogonal to, and can be used in conjuction with, these techniques.

In bus-based SoC designs, the system bus frequency is generally the same as the processor frequency [5]. This simplifies the interface between processor and bus. Moreover, no processor cycles are lost while transferring data between the processor and the system bus. However, other design techniques can result in superior power and energy consumption.

The processing elements included in SoCs are becoming more sophisticated; they now normally include a local (dedicated) level of memory hierarchy. CPU cores [5], for example, come with one or more caches and ASICs include buffers. Some cores include scratch pad memories, to which critical data or instructions are mapped for improved performance. This local memory allows computation to continue in parallel with AHB bus transactions. For example, a hardware unit that executes an algorithm containing a loop may be executing one iteration of the inner body of the loop while requesting the data for the next iteration from the bus. On programmable cores, the operating system can switch to a new execution thread while another thread is waiting for a long transaction on the system bus. The facts that computation and communication may proceed in parallel and processing elements need not stall while waiting for data decouples an increase in bus latency from system energy consumption, as long as latency constraints are honored.

SoC designers should use bus clock frequencies that minimize power consumption in the system clock network while preventing the bus clock frequency from causing violations of communication latency constraints or floorplan area constraints. In this paper, we propose a technique to select the bus frequency resulting in minimal power consumption under constraints on latency and floorplan area. Although our approach uses a specific type of bus protocol and specific circuits as interfaces (see Section 2) in order to present concrete experimental results, our analysis can be easily extended to other types of bus architectures and interfaces that use similar bus topologies and protocols.

2 Basic Parameters and Assumptions

In this section, we describe our system model. Note that we assume floorplanning of already existing devices is complete, i.e., the positions of the components, interfaces, and bus logic are known.

2.1 Components

Each component works at a predefined fixed clock frequency, f_c. This frequency is used for scheduling operations on the component. Each component is connected to one or more interfaces, each of which enables communication with the bus. The components may have multiplexed address and data busses, indicated by $m_c = 1$, or separate address and data busses, indicated by $m_c = 0$. Finally, we assume that the number of data bus lines Ndl_c, address bus lines, Nal_c, control lines, Ncl_c, and response lines, Nrl_c, are known. The control lines specify the type of transaction to perform. Response lines describe the outcome of a transaction. Static on-chip memories are a special component type. They do not work at a fixed clock frequency, but are characterized by their longest access times, i.e., the maximum time they require to perform a read or a write.

2.2 Interfaces

All interfaces are AHB-compatible and are connected to the AHB. Each interface can be a master or a slave. Components invoke communication operations (reads or writes) via their master interfaces. Slave interfaces receive communication operations and pass them to their components. After a component executes an operation, its slave interface responds to the master that invoked the operation.

The interfaces work at the bus clock frequency, f_{bus}. Each interface is synthesized based on the needs of the component it connects to the bus. The maximum unused area between a component, c, and one of its interfaces, i, can be used to place the synchronization logic for data exchange between c and i. This area is called AS_{ci} (area slack). The area slack will be modeled, at the system-level, as an integer indicating the number of minimum-size cells that can be placed between the component and one of its interfaces. The use of area slack allows the accurate prediction of wire lengths before clock selection.

Symbol	Definition
l	latency of a com-op
d	relative deadline for the com-op
λ	time that the com-op occupies the bus
t_{exe}	time for the component to perform the read or write operation
r	binary value that indicates whether the operation is a read (1) or write (0)
Nb	number of useful data bits transferred during the com-op
β	number of beats of the com-op
ρ	number of invocations of the com-op per sec

Table 1. Com-op Parameters

2.3 Communication Operations

Com-ops are read or write operations invoked by a specific master interface for a specific slave. Each com-op, k, has a latency, l_k, and a deadline, d_k, measured in seconds. For the SoC architecture to be valid, all com-ops must meet their deadlines, i.e.,$\forall k \in$ com-ops : $l_k < d_k$. Furthermore, each com-op imposes an execution delay, t_{exe}, on the specific component that is involved for its execution. Therefore, if the component is a memory and the com-op is a read operation, t_{exe} is the duration, starting from the time at which control lines are valid, required by the memory to send the data. t_{exe} is also measured in seconds and does not include the time to transmit the address and the data via the AHB bus. The time a com-op occupies the bus is represented by λ. In this work, we do not consider split transactions. Therefore, λ is always greater than t_{exe}. The number of useful data bits transferred during one com-op is represented by Nb, e.g., if the com-op writes an ASCII character, then $Nb = 7$. Each com-op has an average invocation rate, ρ_k, the number of invocations per second. The number of beats for a com-op is denoted by β. For a single-beat transaction, $\beta = 1$; for a burst, $\beta > 1$. Finally, a binary value, r, indicates the type of com-op; $r = 1$ for read operations and $r = 0$ for write operations.

2.4 Clock Domains

SoCs may contain multiple clock domains. The global system clock is the AHB clock routed as an H-tree network. This clock signal is distributed using a multiple driver scheme [10]. Each component, except memories, may have a different clock frequency and, therefore, a different clock domain. We assume that phase-locked loops (PLLs) for the major components are already present in the floorplan in order to reduce clock skew between the component's and the AHB clock, and to generate the component's local clock signal, if necessary [10].

2.5 Data Transfers between Clock Domains

The process of transferring data between interfaces and components depends on the bus clock. If f_{bus} is a multiple or divisor of f_c, where c is a component of the system, synchronization logic need not be added between these domains. Propagation and setup times provide upper bounds on f_{bus}. Moreover, setup and hold times of f_{bus} must be less than all T_cs of the system.

If f_{bus} is not a multiple or a divisor of f_c, a bridge is necessary to ensure safe data transfer between different clock domains. In this work, we use rational clocking circuits [9] as bridges because they allow bounds on transfer latency. Note that other non-uniform clock frequency communication schemes that support bounded latency are also appropriate [16].

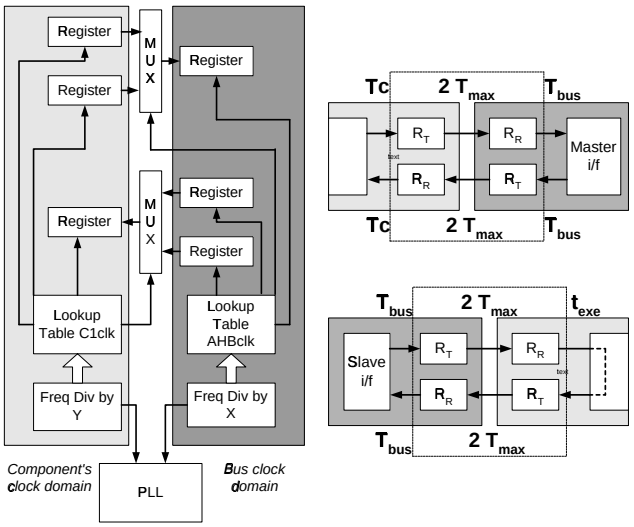

Figure 1. Rational Clocking

2.6 Bridges

If the frequencies are related as follows: $\frac{f_{bus}}{f_c} = \frac{X}{Y}$, the coincidence period $T_{XY} = Y \cdot T_c = X \cdot T_{bus}$ where X and Y are integers.

Based on the coincidence period, a schedule of safe and unsafe cycles for transfers can be determined [9]. Here we assume that double buffering approach is used (see Figure 1). Moreover, f_c is already fixed and f_{bus} is chosen by the described procedure before synthesis of the rational clocking circuit. Therefore, off-line scheduling can be used with a look-up table of predefined frequencies. This look-up table can be implemented by a NOR-based serial-accessed ROM [11]. Decoding signals are generated by a circular shift-register. The number of ROM cells needed is $4 \cdot X$ in the bus's clock domain for a coincidence period T_{XY} defined as before. The shift-register will be of X flip-flops. The length of each bit-line will be $X \cdot L_{cell}$, where L_{cell} is the corresponding dimension of the ROM cell. The same equations, this time based on Y, can be used to find the characteristics of the look-up table in the components domain.

Two transmit registers and one receive register are used in each domain. If the interface is a master, then the component transmits address, control, and data signals. Therefore, the two component-side transmit registers and the one interface-side register need be no larger than $\max(Ndl_c, Nal_c + Ncl_c)$ bits. The master-to-component registers will have $Ndl + Nrl$ flip-flops. Slave interface and component registers are equal in number and size. Only the directions of the registers change as the address, control, and write data bits are transferred from the interface to the component. The component then transmits the data and response bits. In each case, multiplexor inputs have the same sizes as the registers.

The frequency dividers are counters of size $\lceil \log X \rceil + 1$ and $\lceil \log Y \rceil + 1$. We assume that half of the ROM cells contain minimum-size NMOS transistors. Therefore, there are $2 \cdot X$ $(2 \cdot Y)$ transistors in the bus's (component's) look-up table. In addition, there are 4 PMOS transistors sized to precharge the bit lines.

The binary variable *rat* is used to indicate whether rational clocking is used for an interface ($rat = 1$) or not ($rat = 0$).

Symbol	Definition
f_c	operating clock frequency for component c
m_c	binary variable indicating whether ($m_c = 1$) or not ($m_c = 0$) component c multiplexes address and data busses
AS	area slack between a component and one of its interfaces
rat	binary variable indicating whether rational clocking is used for an interface ($rat = 1$) or not ($rat = 0$)

Table 2. Component and Interface Parameters

3 Clock Frequency Selection

In the following sections, we describe several SoC parameters that depend on the bus clock frequency.

3.1 Hardware Logic - Area

The area slack for each interface is indicated by an integer number of reference cells that may be placed between the interface and the component. We estimate the area cost of the synchronization hardware by counting the number of required flip-flop (N_{ff}), ROM (N_{rom}), and multiplexor cells (N_{mux}). Each type of cell will be assigned a weight w denoting the number of (minimal-size) reference cells it occupies. The weights also incorporate wiring overhead costs. Therefore, the area of the added logic for an interface i is be expressed as

$$AA_i = w_{ff} \cdot N_{ff} + w_{rom} \cdot N_{rom} + w_{mux} \cdot N_{mux}$$

In order for the solution to be valid: $\forall i \in \text{interfaces} : AA_i < AS_i$. After finding the ratios X and Y, for which $X \cdot T_{bus} = Y \cdot T_c$, the number of cells of each type can be computed in constant time by using the formulas in Section 2.6.

3.2 Effect on Latency

The worst case latency for a single-beat com-op, k, follows:

$$l_k = t_{CM} + t_{arb} + \lambda_k \quad (1)$$

where λ_k is the time k occupies the bus:

$$\lambda_k = t_{transfer} + \left\lceil \frac{t_{SC} + t_{exe}}{T_{bus}} \right\rceil T_{bus}$$

t_{CM} is the time that is lost in the exchange of control, address, and data between the component and the master interface; t_{arb} is the time that the master interface waits to be granted the bus; $t_{transfer}$ is the time to transfer control, address, and data between the master and the slave interfaces using AHB; t_{SC} is the time lost in the exchange of control, address, and data between the slave interface and the component attached to it; and t_{exe} is the time it takes for the slave to execute the operation. For convenience we will denote the time spent on the slave side in bus cycles as t_s. Therefore,

$$t_s = \left\lceil \frac{t_{SC} + t_{exe}}{T_{bus}} \right\rceil T_{bus} \quad (2)$$

3.2.1 Component-Master Communication

t_{CM} depends on the number of required transfers between the interface and the component as well as the latency of each transfer. The number of transfers varies depending on com-op and protocol. Many solutions have been proposed for interface synthesis between different protocols [6, 7]. We assume that interfaces between different protocols consist of registers and straight-forward combinational logic for translation of control and handshaking signals. If more complicated interfaces are desired, the analysis presented in this paper can be easily extended. For a single-beat operation we assume the number of transfers to be given by:

$$TR_{MC} = \left\lceil \frac{Nb_k}{Ndl_C} \right\rceil + \overline{r_k} \cdot m_C + 1 \quad (3)$$

The first part operand is the number of required transfers in case the component's bus lines Ndl_C are less than the useful bits Nb_k transferred during the com-op k. If the operation is a write ($\overline{r_k} = 1$) then one more data exchange is needed if the data and address busses are multiplexed ($m_C = 1$). The response of the slave will need one more transfer from the interface to the component.

The time for each transfer depends on the system and component clock frequencies. If $rat_c = 0$, then it takes one clock period of the slowest-clock domain for each transfer $T_{max} = \max(T_c + T_{bus})$. However, in the case of rational clocking, $T_c + T_{bus} + 2T_{max}$ time is needed for each transfer, as shown in Figure 1. Each transfer through the bridge can be bounded by $2T_{max}$ [9]. Therefore, the maximum time needed is:

$$t_{CM} = \overline{rat_c} TR_{MC} T_{max} + rat_c TR_{MC} (T_c + T_{bus} + 2T_{max})$$

3.2.2 Arbitration

In this work, the arbiter is assumed to implement a FIFO protocol. The time t_{arb} spends waiting to grant the bus depends on the number of masters connected to the bus and upon the longest-running bus com-op of each master. The worst-case time, measured in bus clock cycles, for a com-op initiated by master i is

$$t_{arb} = \sum_{m=1, m \neq i}^{M} \lambda_{max m} - (M-1) T_{bus} + T_{bus} \quad (4)$$

This is the case when all other $M-1$ masters have already issued their highest-latency com-ops. The total latency is the sum of worst-case latencies minus $M-1$ cycles (to account for address and data pipelining). After a request from master i is issued, the arbiter samples it at the beginning of the next cycle. Therefore, at most, one cycle is added to t_{arb}. The worst case arbitration time is the same for all com-ops of the same master.

3.2.3 Transfer Time through the AHB

The AHB protocol requires one clock cycle to transfer the control and address signals from master to slave. In the event of a write operation, data is transferred during the subsequent clock cycle. After the operation is executed by the component attached to the slave interface, one more cycle is required to transfer the response from the slave [1]. Therefore, the total time spent transferring data through the AHB is

$$t_{transfer} = (2 + \overline{r}) T_{bus}$$

3.2.4 Com-Op Execution

This term depends on the com-op and the component on which the com-op is executed, does not vary with f_{bus}, and is considered known (see Section 2.3).

3.2.5 Component-Slave Communication

t_{SC}, like t_{CM}, depends on the number of required transfers between interface and component as well as the latency of each transfer. For a single-beat operation the number of transfers is given by Equation 3.

Memory Memory slave interfaces have level-sensitive latches instead of edge-triggered flip-flops for the *read data* and *response* signals. By requiring the memory access execution time, t_{exe}, and the propagation delay, t_p, through the slave latch, the AHB wires, and AHB logic to be less than T_{bus}, the slave bus time, t_s, is zero. The delay is included in one of the transfer cycles. To ensure the zero overhead in this memory access case, the longest memory access time added to the propagation delay is included in the constraints for the maximum frequency of the bus. In addition we require $Nb_k < Nl_C$.

Non-memory, Non-rational Communication By definition, t_{exe} accounts for two transfers; it is the duration from component input to response. Therefore, the total time in this case is

$$t_s = \left\lceil \frac{t_{exe} + (TR_{SC} - 2) \cdot T_{max}}{T_{bus}} \right\rceil T_{bus}$$

As seen from the above equation, if more than two transfers are required, their delay is equal to the longest clock cycle: $T_{max} = \max(T_{bus}, T_c)$.

Rational Clocking Transfers When rational clocking logic is needed, the situation is similar to communication between the component and the master interfaces. In this case, as seen in Figure 1, the total delay is

$$t_s = \left\lceil \frac{TR_{SC}(T_{bus} + 2T_{max}) + t_{exe}}{T_{bus}} \right\rceil T_{bus}$$

3.2.6 Burst Transactions

The above discussion was limited to single-beat transactions. For an β-beat burst transaction the latency of a com-op k on the bus follows:

$$\lambda_k = (\beta - 1)t_m + \beta(t_{transfer} + t_s) \quad (5)$$

where t_m is the time spent at master-component communication in bus cycles. For simplicity, we will assume that the component, c, attached to the master interface will always produce the values of the signals for one beat of the transaction one cycle T_c after receiving the response from the previous beat. Thus, $t_m = \lceil t_{CM}/T_{bus} \rceil T_{bus}$. The total latency of the com-op is given by Equation 1. We assume that each beat of the burst starts only after the previous response has reached the component attached to the master. This trades off interface simplicity for com-op latency.

3.2.7 Lower and Upper Bound for the Bus Frequency

In order for the system to meet its latency constraints, the following inequality must hold for every com-op k: $d_k - l_k > 0$. The com-op with minimum d represents the tightest deadline for the system, independent of the bus frequency.

From the previous sections it can be shown that, in the best case, $\lambda = 2T_{bus}$ and $t_m = 2T_{bus}$. The deadline must be met even if all other masters have already requested the bus and will execute their transactions first. The best-case arbitration time t_{arb} is therefore MT_{bus}. Since

$$l = t_m + t_{arb} + \lambda = (M + 4)T_{bus}$$

in the best case,

$$\min d - (M + 4)T_{bus} > 0 \Rightarrow f_{bus} > \frac{M + 4}{\min d}$$

Any value of f_{bus} that does not satisfy this inequality will lead to at least one com-op that will not meet its deadline under some ordering of operations on the bus.

The upper bound of the clock frequency can be derived after timing analysis based on the SoC floorplan. The bus clock frequency must satisfy the constraints imposed by propagation delays, set-up times, hold times, and longest memory access times, allowing a memory access to complete in one bus cycle.

3.3 Power

We will first analyze the reduction in power consumption for the clock network already present in the SoC. We will then estimate the power consumption of the additional logic required by the proposed technique.

3.3.1 Power Consumption on the Clock Network

A clock network's power consumption is the sum of its static and dynamic power $P_{hclk} = P_{dyn} + P_{stat}$. The dynamic power consumed on the clock network is given by the following equation: $P_{dyn} = V_{dd}^2 \cdot C_{hclk} \cdot f_{bus}$ In this equation, V_{dd} is already fixed for a specific technology. In order to compute the effect of changing f_{bus}, an estimate of C_{hclk}, the effective capacitance of the bus clock network, is necessary. Our method of estimating clock network capacitance is based on the literature [10, 12], i.e.,

$$C_{hclk} = C_{Drivers} + C_{Wiring} + C_{PLLs} + C_{Registers/Latches}$$

Drivers Power Estimation The contribution of the drivers of the clock network to the power consumption will be estimated using the models and formulas presented in the literature [12]. In total there is one driver for each main branch of the H-tree network. Moreover, there is one driver for each interface and one for the arbiter. The driver of an interface will also be utilized by the bus clock domain registers of the bridge, if any, between the interface and the corresponding component. Since a choice of f_{bus} can add a bridge and increase the load for the drivers, it may increase the number of the stages needed and by that the static power on the drivers. Therefore, the static power for the drivers will be computed for every new frequency.

Wiring Power Estimation The total clock tree wire capacitance is the sum of the capacitance in the H-tree. The wire length L for the main branches of the network can be found from the floorplan of the system. The contribution of each wire to the total capacitance, C_{wire}, is $C_{int/unit-length} \cdot L$, where $C_{int/unit-length}$ is found by summing the wire parallel-plate capacitance per unit length and the interwire capacitance estimates [15].

PLLs Power Estimation With the exception of the phase-frequency detector, and potentially the frequency divider before it, all other elements of the PLLs belong to the clock domains of individual components. Analytical formulas can be used to compute the bus clock load on the phase detector and the frequency divider [12]. There will be one PLL for each clock domain. Therefore, the effective capacitances of all PLLs must be added to obtain C_{PLLs}. The leakage power for the PLLs is relatively small [12] and will be ignored.

Registers and Level-Sensitive Buffers Power Estimation The capacitance of the arbiters and interface registers may be estimated using analytical formulas [12]. The slave interfaces attached to memory components contain level-sensitive buffers for the read data and response signals. Each latch of the buffer will be considered to produce half the load of an flip-flop. The sum over all registers and level-sensitive buffers in the bus clock domain yields $C_{Registers/Latches}$.

In addition to increasing the clock network's capacitive load, the registers consume leakage power, transition power,

and data power. This power does not depend on the selection of the clock frequency for the registers in the interfaces. Therefore, it will not be considered in the estimation of the clock network power.

3.3.2 Power Consumption on the Bridges

In the following sections we will analyze the power consumption overhead resulting from the insertion of additional bridges.

Static Power For the static power we will use the model described in the literature [14] $P_{static} = V_{dd} N_t k_{design} I_{leak}$. Section 2.6 describes how to compute the number of flip-flops and multiplexor inputs for a specific f_{bus}. We will assume V_{dd} fixed for a specific technology. N_t, k_{design}, and I_{leak} can be found in the literature [14]. For finding the static power of ROM transistors the formulas in the BSIM3v3 manual [17] are used with the proper W for each transistor and assuming temperature of $100°C$.

Component and Bus Clock Power The equations in Section 2.6 are used to estimate the number of flip-flops and in the transmit, receive, and shift registers. Their power consumption, and that of each component's precharge transistors, are used to determine component clock power consumption. Register and precharge transistor power consumption is also considered when computing the bus clock power consumption.

Transition and Data Power This section describes the estimation of flip-flop power consumption [13].

Double buffering at the transmitter of the bridge results in transition and data power consumption during each bit transition. The receiver uses only one register. Therefore, only its transition power is considered.

The registers used for transfers between domains change values based on com-op and transfer type. When flip-flops are used for address, read, or write data bits, their switching activities are high compared to the flip-flops used for control and response transfers. Therefore, the contribution of the latter flip-flops may be neglected. For a β beat com-op, k, the average number of transitions on the data lines is $btpi_k^D = \beta \frac{1}{2} N b_k$.

We assume that the addresses of different communication operations are not correlated. Therefore, during the first beat of a com-op, half of the address bits typically change values. We assume that $\log(\beta)$ transitions take place during remaining com-ops. The bit transitions during an address transfer between the invoking component $c1$ and its corresponding master interface is $btpi_k^{Ac1} = \frac{1}{2} N al_{c1} + (\beta - 1)\log\beta$.

The bit transitions on the inserted logic will occur only if rational clocking is used between a component and its interface. Since two components are involved for each com-op the total bit transitions per invocation will be

$$btpi_k = rat_{c1}(btpi_k^{Ac1} + btpi_k^D) + rat_{c2}(btpi_k^{Ac2} + btpi_k^D)$$

The bit transitions per cycle, α_k, for a specific clock domain with period T and a com-op k with average invocation rate of ρ_k is $\alpha_k = btpi_k \rho_k T$. Thus, the power resulting from k in that domain is

$$P_k = \frac{1}{2} V_{dd}^2 C \alpha_k f \Rightarrow P_k = \frac{1}{2} V_{dd}^2 C btpi_k \rho_k \not{T} \not{f} \Rightarrow$$

$$\Rightarrow P_k = \frac{1}{2} V_{dd}^2 C btpi_k \rho_k \quad (6)$$

Note that P_k is independent of the frequency of the specific domain. In this equation, the capacitance, C, is given by the transition and data capacitance of the flip-flops. The sum of the resulting power consumption, for all com-ops, gives power overhead due to additional bridges.

Additional Dynamic Power on Bridges We model the additional dynamic power consumed by the ROM components as follows. In every cycle, it is assumed that half of the bit lines change values. This implies that two bit lines in each ROM are precharged and two are discharged in the same cycle. In each cycle, one word line is deactivated and another is activated. Finally, two transitions occur every cycle for each shift-register.

4 Proposed Algorithm

Figure 2 illustrates the proposed algorithm. After choosing a value for f_{bus}, it is necessary need to establish the values of *rat* for each component as well as the ratios X and Y (see Section 2.6). Using these values, the algorithm checks area and latency constraints. We assume that the f_{bus} and f_c of each component are integers. Euclid's algorithm [8] is used to find the greatest common divisor (GCD) of f_{bus} and f_c for each component. If the GCD is equal to f_{bus} or f_c then no bridge is needed. Otherwise, the ratios X and Y will be found by dividing f_{bus} and f_c, respectively, by the GCD. Note that the optimization algorithm considers the area and power costs of the look-up tables used for communication between different clocking regions. Euclid's algorithm completes after $O(\lg(\min(f_{bus}, f_c)))$ and can safely bounded by $O(\lg f_{bus}^{max})$, where f_{bus}^{max} is the upper bound for the bus frequency as described in Section 3.2.7. Euclid's algorithm is executed N_c times for each frequency f_{bus}, where N_c is the number of components.

Checking the latency constraints for all com-ops in the system can be done in $O(C)$ time, where C is the total number of com-ops. The algorithm first finds the λ value for all com-ops. Then, based on the known λ values, the worst case arbitration latency and worst-case latency are computed. The latencies are then compared to the deadlines of the communication operations. The area constraints can be checked in $O(I)$ time, where I is the number of interfaces in the system.

Having established the *rat* values of the components, power consumption estimation for a specific f_{bus} can be done in $O(I+C)$ time. For the clock network, we consider all three levels of the H-tree (seven branches). The number of PLLs is bounded by N_c, and the number of interfaces, I, is greater than or equal to N_c. The overhead power resulting from the bridges can be found in $O(I+C)$ time, since the transition power is found in $O(C)$ and the other power consumption values are found in $O(I)$ time. Therefore, the power computation can be computed in $O(I+C)$ time for a specific f_{bus}. The total run time for checking area and latency constraints and power computation is $O(N_c \lg f_{bus}^{max} + I + C)$

The proposed algorithm variation of the binary search. The search is initialized with the processor's frequency [5]. In each step of the algorithm a new frequency, f_{bus}^{cur}, is obtained. For this frequency, and a set of neighbor frequencies, the latency and area constraints are checked and the system power consumption is estimated. If the latency constraints are met for f_{bus}^{cur} or one of the neighbor values then $f_{bus}^{max} = f_{bus}^{cur}$. Otherwise, $f_{bus}^{min} = f_{bus}^{cur}$. The next frequency visited is $f_{bus}^{cur} = \frac{f_{bus}^{max} + f_{bus}^{min}}{2}$ with the updated values. The neighbor frequencies for f_{bus}^{cur} are the closest values to f_{bus}^{cur} for which at least one of the rat_c values is zero. For each

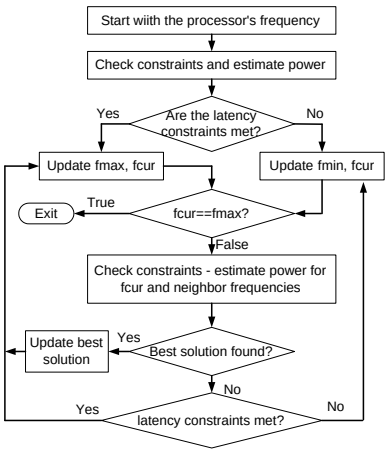

Figure 2. Block Diagram of the Algorithm

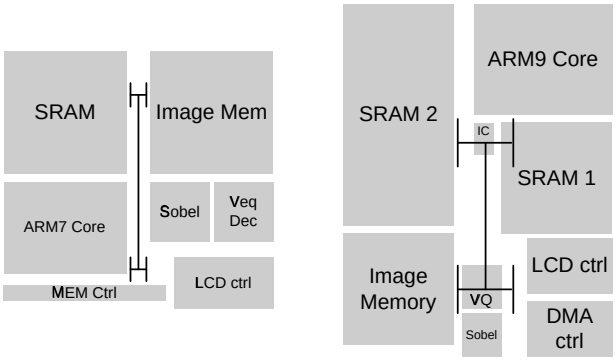

Figure 3. Floorplans for the 0.25μm and 0.18μm benchmarks

component c, these values are given by

$$f^{l}_{neighC} = \left\lfloor \frac{f^{cur}_{bus}}{f_c} \right\rfloor f_c, \quad f^{u}_{neighC} = \left\lceil \frac{f^{cur}_{bus}}{f_c} \right\rceil f_c$$

For these values, we expect local minimum for the power as the inserted logic for component c is eliminated. The algorithm keeps track of the frequency that produces the minimum power and satisfies all constraints, and terminates when a precision threshold for the frequency value is reached, i.e., $f^{cur}_{bus} = f^{max}_{bus}$.

The number of iterations of the binary search approach is bounded by $\log f^{max}_{bus}$. The number of frequencies visited in each iteration is $2N_c + 1$. Therefore, the total running time for this algorithm is: $O\left(N_c^2 \lg^2 f^{max}_{bus} + N_c \lg f^{max}_{bus}\{I + C\}\right)$.

5 Experimental Results

The proposed approach has been applied to two SoCs. The first uses a 0.25μm process and the second one uses a 0.18μm process. The floorplans of the two benchmarks can be seen in Figure 3. The floorplan dimensions are 4.3mm$\times$4.17mm and 4.8mm$\times$6.3mm. In the first case, the initial frequency was set to the processor frequency, 66MHz [5]. All latency constraints are met and no bridges are needed in this control design because all frequencies are divisors of the processor's frequency. The proposed algorithm chose a 33MHz frequency for the 0.25μm SoC. At this frequency, two bridges are required, one for the Vector Quantization Decoder and one for Sobel hardware unit. All latecny constraints are satisfied and the power of the final design, which includes the power of the clock network and the overhead power of the inserted bridges, was reduced by 20.2%, as indicated in Table 3.

Benchmark	Power (mW)		Reduction in Power
	Initial Design	Final Design	
0.25μm	6.97	5.56	20.2%
0.18μm	8.01	3.94	50.8%

Table 3. Results for the two SoCs

For 0.18μm design, the initial bus frequency was 144MHz, equal again to the processor frequency. No bridges are needed for this frequency and all latency constraints are satisfied. The proposed algorithm selected a frequency of 48MHz. For this frequency, two bridges are required, one for the Sobel hardware unit and one for the Interrupt Controller. For this case, the power consumption was reduced by 50.8%, as indicated in Table 3. The improved power savings are the result of a larger on-chip memory reducing the number of time consuming accesses to off-chip memory. As a result, although deadlines have been scaled based on frequency changes and user requirements, the frequency can be reduced substantially without violating the latency deadlines. Moreover, as SoC size increased, the clock network wires became more important relative to the overhead resulting from additional bridges in the clock and the component domains. For both SoCs, the latency increase does not affect energy consumption, as computation occurs in parallel with communication.

6 Conclusions

This paper has presented an approach for selecting system bus frequency in order to optimize power consumption under constraints on area and latency. Compared to clocking bus and processor at the same frequency, this approach reduces power consumption by average of 35% for the AHB SoC designs for which it was used. In the future, we plan to extend this approach for use with additional bus protocols and for the multi-layer AHB architecture.

References

[1] AMBA Specification (rev2.0), http://www.arm.com
[2] F.H.A. Asgari, M. Sachdev; "A Low-Power Reduced Swing Global Clocking Methodology"; IEEE TVLSI, Vol 12, No. 5, May 2004
[3] D.W. Dobberpuhl et. al.; "A 200-MHz, 64-b dual-issue CMOS microprocessor", IEEE Journal of Solid-State Circuits, 27(11): 1555-1564, Nov 1992
[4] M. Conno, A. Ivaldi, L. Benini, E. Macii; "Clock-Tree Power Optimization based on RTL Clock-Gating"; DAC 03
[5] S. Furber; "ARM SoC Architecture"; Addison Wesley, 2nd edition.
[6] S. Narayan, D. D. Gajski; "Interfacing Incompatible Protocols using Interface Process Generation"; DAC 95
[7] R. Passerone, et. al. "Automatic Synthesis of Interfaces between Incompatible Protocols"; DAC 98
[8] T. H. Cormen, et. al.; "Introduction to Algorithms"; 2nd Edition, The MIT Press
[9] L. F. G. Sarmenta, et. al.; "Rational Clocking"; ICCD; Oct 95
[10] R.Y. Chen, et. al.; "Clock Power Issues in System-on-a-Chip Designs"; IEEE Workshop on VLSI, Apr 99
[11] J. Rabaey; "Digital Integrated Circuits: A Design Perspective"; Englewood Cliffs, NJ: Prentice-Hall Int., 1996
[12] D.E. Duarte, et. al.; "A Clock Power Model to Evaluate Impact of Architectural and Technology Optimizations"; IEEE TVLSI Systems, Vol. 10, No. 6, Dec 2002
[13] R. Ramanarayanan, N. Vijaykrishnan, M.J. Irwin; "Characterizing Dynamic and Leakage Behavior in Flip-Flops"; IEEE ASIC/SOCC, Sep 2002
[14] J.A. Butts, G.S. Sohi; "A Static Power Model for Architects"; International Symposium on MicroArchitecture, MICRO, Dec 2000
[15] D. Duarte, V. Narayanan, M.J. Irwin; "Impact of Technology Scaling in the Clock System Power"; ISVLSI 2002
[16] T. Chelcea, S. M. Nowick; "A Low-Latency FIFO for Mixed-Clock Systems"; IEEE Workshop on VLSI, Nov. 2000
[17] Y. Cheng, et. al.; "BSIM3V3 Manual"; http://www-device.eecs.berkley.edu/ bsim3