Power Minimization Using Control Generated Clocks

Srikanth Rao M. and S. K. Nandy
msrao@rishi.serc.iisc.ernet.in, nandy@serc.iisc.ernet.in
Supercomputer Education and Research Center
Indian Institute of Science
Bangalore, India 560012

Abstract

In this paper we describe an area efficient power minimization scheme "Control Generated Clocking" that saves significant amounts of power in datapath registers and clock drivers of sequential circuits. Power savings are achieved by making simple changes to the state machines controlling the datapath. These changes enable the control signals from the state machines themselves to be used as clocks for the datapath registers. Use of these control generated clocks makes the static timing analysis of designs implementing this scheme simpler when compared to techniques such as clock gating. This scheme preserves the cycle boundaries on which registers load data, thereby allowing reuse of functional test cases developed for the original circuit. In this paper we also describe timing requirements of a design in which this scheme has been implemented, cost-benefit aspects of this scheme and an algorithm for the automatic synthesis of control generated clocks. Results from application of this technique on a complex design are then discussed.

1. Introduction

High levels of integration, high operational frequencies and the proliferation of battery operated applications has rendered power dissipation a key parameter in the design of present day VLSI circuits, since it affects reliability, performance and cost of the circuit [1][8]. In high frequency CMOS digital circuits, the dynamic power resulting from charging and discharging of parasitic capacitances dominates over the power dissipated due to leakage currents [1][8]. The focus of most minimization techiniques is therefore dynamic power. In a sequential circuit, the sources of dynamic power dissipation and commonly used techniques for power minimization are listed in Table 1.

Clock gating [1][6][8] is an effective technique for power minimization in circuits that are idle for long periods of time. Clock gating in its most general form has some practical difficulties, viz.

Source	*Minimization techniques*
Clock drivers	Clock gating
Registers	Clock gating
Combinational logic	Precomputation and guarded evaluation[9][12], Retiming[5], State assignment in FSMs [2] [7]

Table 1. Sources of power dissipation in sequential circuits and associated minimization schemes.

1. Possibility of glitches on the gated clock signal when the gating signal arrives later than the clock.
2. Difficulties in using static timing analysis effectively on the design when the gating signal is data dependent, which is usually the case.

In thus paper, we describe a robust new technique called Control Generated Clocking (CGC) that can be applied to minimize dynamic power in a synchronous sequential circuit. This scheme achieves higher power savings compared to clock gating while overcoming the drawbacks of clock gating listed above. This scheme minimizes the power dissipation in the datapath registers and in the clock drivers, without affecting the cycle boundaries on which registers load data.

The organization of the rest of the paper is as follows. Section 2 of the paper provides an overview of the proposed scheme. Following this, we describe the static timing analysis of a design with CGC in section 3. Section 4 describes an algorithm for automatic synthesis of CGC and we discuss the cost-benefit aspects of CGC in section 5. We present results obtained after implementing CGC in a RISC processor in section 6 and finally conclude in section 7.

2. Control Generated Clocking (CGC)

Consider a general edge-triggered synchronous sequential circuit shown in figure 1(a), in which multiple state machines are shown, sequencing operations in the datapath. Control signals from the state machines enable writes to registers in the datapath. The master clock MCLK synchronizes all activity in the datapath registers as well as in the controlling state machines. Note that when writes to certain datapath registers are infrequent, a lot of power is unnecessarily dissipated in the drivers of the clock MCLK and in flip-flops of the datapath registers.

Figure 1(b) shows the timing diagram for a write to a single flip-flop in any datapath register in such circuits. The figure shows the control signal WRITE, which is synchronous to the clock MCLK. The flip-flop loads the data on the next rising edge of clock MCLK. Observe that for a single write operation, this also happens to be the falling edge of the control signal. Our scheme

essentially uses this control signal as the clock for the flip-flop. The flip-flop must now respond to the falling edge of the WRITE signal. This simple arrangement does not work when there are back-to-back writes to the flip-flop. In such a scenario, the **WRITE** signal changes only at the end of the last write to the flip-flop. In the CGC scheme described below, this problem is alleviated using RZ pulse generator circuits.

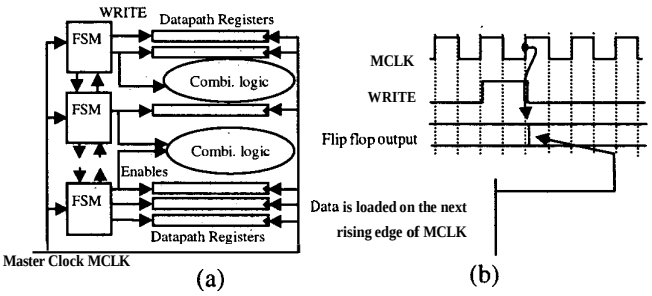

Figure 1. (a) A general synchronous sequential circuit. (b) Timing diagram of a write operation.

In CGC, the general system shown in figure 1(a) is transformed to a system shown in figure 2(a). This scheme introduces certain changes to the original circuit listed below:

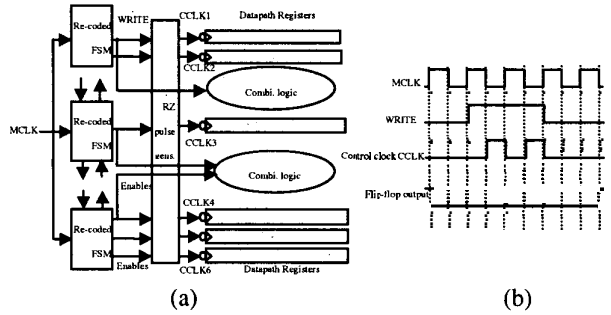

Figure 2. (a) Circuit with CGC (b) Timing diagram of a circuit with CGC.

(a) Changes to the datapath registers: All positive edge triggered flip-flops in the datapath registers are replaced by negative edge triggered ones. This is needed in order to preserve timing relationships between signals of the original circuit, because writes to registers in the original circuit complete on the falling edges of the control signals.

(b) Changes to the clocks: Clocks to all datapath registers in the target logic are replaced by control generated clocks, indicated as CCLKs in the figure. The control generated clocks are control signals that have been conditioned by the RZ pulse generators. The RZ pulse generator circuit is described in [10][11]. Step (a) above can be eliminated if active low control clocks are used.

(c) Changes to the controlling state machines: The states of the controlling state machines are recoded to ensure that each of the control clocks is glitch free and has minimum skew with respect to the master clock MCLK. If a control signal is used for any other purpose, besides enabling the write to a register, two versions of the control signal are created. One is used for the generation of the control clock, and the other is used to feed any other logic as required. The latter is not fed to the RZ pulse generator. Additionally, control signals for registers that are written to by more than one state of the controller may need shadow flip-flops and retiming. This step is described in detail in [10][11].

Figure 2(b) illustrates the timing relationships between signals in an implementation of this scheme. The figure indicates two back-to-back writes taking place to a flip-flop. Note that the WRITE signal has only one falling edge in this case. If the WRITE signal were directly fed to the clock input of the flip-flop, only one write would take place. The RZ pulse generator toggles the control clock once every cycle, allowing back to back writes to take place. The RZ pulse generation circuit is not necessary for those flip-flops that never get updated in a back-to-back fashion. Note from the timing diagram that data gets loaded into the flip-flops at the same cycle boundaries as in the original circuit. This is desirable, because test cases written for the verification of the original circuit can now be reused without any changes.

3. Timing Analysis

In this section, we describe the timing constraints to successfully implement CGC in a circuit. In a circuit with CGC, in addition to the original clock domain of MCLK, new clock domains of the control clocks CCLKs are introduced. The constraints to be satisfied for ensuring reliable transfers between registers in different clock domains are based on the source and the target clock domains. In CGC, the control clocks are derived from flip-flop outputs, and will have deterministic delays with respect to the master clock. To derive the constraints for reliable transfers between any two flip-flops in a design with control generated clocking, we assume a model shown in figure 3. For simplicity we assume that RZ pulse generators generate all control clocks.

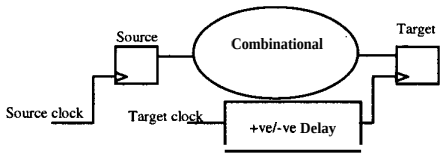

Figure 3. Model used for timing calculations.

The figure shows a combinational circuit path originating at a flip-flop output and ending in flip-flop input, possibly in another clock domain. The flip-flop at the target of the combinational path sees a skewed clock with respect to the source. Note that because of the RZ pulse generators, the control clocks will have the same period T_{clk} as the master clock when they are turned on. We assume that the maximum possible value of the clock skew on the master clock in the design is known and its value is denoted by $\pm T_{skew}$.

Table 2 lists all the timing paths in a design with CGC and the corresponding allowable maximum and minimum delay. In the table.

$$\alpha_1 = T_{clk} - T_{clk-q} - T_{su} - T_{skew} \text{ and}$$
$$\alpha_2 = T_{skew} + T_{hold} - T_{clk-q}$$

where T_{clk-q} denotes the output delay of the flip-flop at the source relative to its clock, T_{su} denotes the setup time of the flip-flop in the target, T_{hold} its hold time, and T_{rz} the propagation delay through the RZ pulse generator.

For transfers between an MCLK domain and a CCLK domain, the: delay introduced by the RZ pulse generator manifests itself **as** additional clock skew. For transfers between two different CCLK domains, this method can expose false timing paths in the original design. In table ***2, n*** indicates the minimum path length between set of states that write to the source and the set of states that write to the target, in the controller state transition diagram.

In sum, timing analysis of a circuit with control generated clocking is simpler than that of a circuit implementing clock gating. This is due to the fact that this scheme introduces deterministic skews on the control clocks.

Path originating in domain	Path ending in domain	Max. Delay	Min. Delay
MCLK	*MCLK*	α_1	α_2
MCLK	*CCLK*	$\alpha_1 + T_{rz}$	$\alpha_2 + T_{rz}$
$CCLK_i$	*CCLK;*	α_1	α_2
CCLK,	$CCLK_j$	α_1 , if $n=0$ $\alpha_1 + (n-1)T_{clk}$, if $n>0$	α_2 , if $n=0$ 0, if $n>0$
CCLK	*MCLK*	$\alpha_1 - T_{rz}$	$\alpha_2 - T_{rz}$

Table 2. Timing paths in a circuit with CGC and their bounds.

4. Automation

This section describes a procedure to automate the insertion of CGC in a circuit. It takes **as** input, a description of state machines controlling the registers in the datapath along with a set of target registers and outputs a re-coded state machine suitable for CGC. This procedure also marks those registers that require shadow flip-flops and can be updated in a back-to-back manner. Each state in the state machine description has an attribute ***code*** that represents its encoding. The set of registers in the datapath that are to be targeted by this scheme is specified in the set *T*. Each register $r_k \in T$ has an attribute ***CS,*** called the control set, which is the set of all states which cause a write to r_k . The control set of each register is constructed from the input state machine description. Every target register has two other boolean attributes namely, ***shadow*** and ***bb*** which respectively indicate whether the register needs a shadow flip-flop for its control signal and whether it can get updated in a back to back fashion. The attribute ***bb*** is used to insert an RZ pulse generator for the control signal. The algorithm for inserting CGC is described below.

```
//Main body of the algorithm
FSM f;
set_of_registers T;
main() {
  // Read the input and construct the FSM f
  read_input();
  // Convert the state machine into Moore if required Procedure
  // is described in Ref [3]
  if ( isMealy() )
     Mealy2Moore();
  // Identify all those states which write to a register. Construct
  //'controlset for each register
  genControlSet();
  // Encode the state machine for CGC
  Encode();
  // Identify those registers which get updated in a back to back
  //fashion, i.e., find if states in the control set of any register
  //have a path length of exactly 1.
  MarkBB();
  // Identify all the control signals which need a shadow flip-flop
  // These are control signals for registers with more than one
  //state in their control set
  MarkShadow();
  // Write the encoded state machine and its attributes to the
  //output.
  write_output();
J
```

The above algorithm is implemented **as** a tool TICTOC. This tool accepts a state machine description and outputs synthesizable VHDL code after inserting CGC. The tool has been implemented in Perl and has been written in approximately 1200 lines of code.

5. Cost benefit analysis

In this section we derive analytical expressions for the power saved in a circuit with CGC, and the overheads associated with it. With CGC, power is saved in the registers and in the clock drivers. In subsection 5.1, we derive an expression for register power saved. Following this, in subsection 5.2, ***an*** expression **for** the power saved in the clock drivers is derived. In subsection **5.3,** we estimate the overheads in CGC.

5.1 Power saved in registers

Power dissipated in the target logic without CGC is first computed. We assume that each target flip-flop in the datapath consumes the same energy E_{clk} per clock, and the same energy E_{write} per write. The total dynamic power consumed per flip-flop, without CGC is given by

$$P_{flip-flop} = E_{clk} f_{clk} + E_{write} f_{write-i}$$

where f_{clk} is the frequency of the clock, and $f_{write-i}$ is the average frequency of writes to register i . The total dynamic power consumed by register i is then given by

$$P_{reg}(i) = (E_{clk} f_{clk} + E_{write} f_{write-i}) width_i$$

where $width_i$ represents the number of flip-flops in register i . Hence the total dynamic power dissipation P of the target circuit is expressed as $P = \sum_{i=1}^{N_{reg}} P_{reg}(i)$.

In a circuit with CGC, the number of clocks to the target flip-flops is the same as the number of writes. The power dissipated by register i is now given by

$$P_{reg-opt}(i) = (E_{clk} f_{write-i} + E_{write} f_{write-i}) width_i$$

The dynamic power dissipation P_{opt} of the target logic is now expressed as $P_{opt} = \sum_{i=1}^{N_{reg}} P_{reg-opt}(i)$

The power savings in the registers can be computed by substituting for P and P_{opt} in the following expression.

$$Saving = \frac{P - P_{opt}}{P} \qquad (1)$$

to yield

$$\text{Register Power Saving} = \frac{1-x}{1+kx} \qquad (2)$$

where $x = \sum_{i=1}^{N_{reg}} \frac{f_{write-i} width_i}{f_{clk} N_{flip-flops}}$, $0 \le x \le 1$ and $k = \frac{E_{write}}{E_{clk}}$, and $N_{flip-flops} = \sum_{i=1}^{N_{reg}} width_i$ is the number of flip-flops in the target logic. Here, k is a parameter that is dependent on the circuit design of the flip-flop, and x is a parameter that captures activity in the registers. Figure 4 shows a plot for this function for various values of k. The maximum savings are obtained when $x = 0$, or when the circuit is shut off. When the circuit is operational, the savings depend on the value of k and the value of x. For a fixed value of x, a larger value of k results in lower power savings. The effect of k on power saved is low when the value of x is close to 0 or close to 1. The ratio $(f_{write-i}/f_{clk})$ is a measure of the activity in the registers. This ratio may be computed as the sum of probabilities of the state machine being in those states that write to register i as follows

$$(f_{write-i}/f_{clk}) = \sum_{s} p(s),\ s \in CS(i)$$

where $CS(i)$ is the control set of register i, and $p(s)$ is the static probability of the controller being in state s. This expression allows us to estimate the savings early in the design process from profile information.

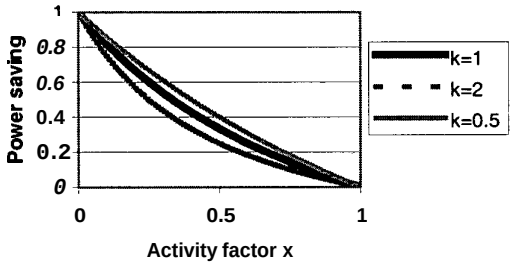

Fig 4. Variation of power saved with activity and flip-flop parameters

5.2 Power saved in clock drivers

We assume that every flip-flop presents a capacitive load $C_{flip-flop}$ on the clock line. In a circuit without CGC, the total capacitance load on the clock line is given by

$$C_{clk} = C_{other} + N_{flip-flops} C_{flip-flop}$$

where C_{other} represents the load on the clock line because of logic that is not part of the target logic. With CGC, a fixed overhead $C_{overhead}$ is added per control clock to the master clock. This makes the load on the clock line

$$C_{clk-opt} = C_{other} + N_{reg} C_{overhead}$$

where N_{reg} is the number of registers targeted by this scheme. At the same time, the registers targeted by CGC load the control clocks. The load on the control clock line i is given by

$$C_{cclk}(\text{i}) = width_i C_{flip-flop}$$

Power dissipated in the clock drivers of the circuit without CGC is expressed as $P = C_{clk} V^2 f_{clk}$, and power dissipated in the clock drivers of the circuit with CGC is expressed as

$$P_{opt} = C_{clk-opt} V^2 f_{clk} + V^2 \sum_{i=1}^{N_{reg}} C_{cclk}(i) f_{write-i}$$

Power saving in the clock drivers is calculated by substituting for P and P_{opt} in Eqn (1), to yield

$$\text{Clock Power Saving} = \frac{1 - k_1 - x}{1 + k_2} \qquad (3)$$

where $k_1 = \frac{N_{reg} C_{overhead}}{N_{flip-fops} C_{flip-flop}}$ is a parameter that indicates the change in master clock net capacitance after implementing CGC, and $k_2 = \frac{C_{other}}{N_{flip-fops} C_{flip-flop}}$ is a parameter that indicates the ratio of the loads due to the untargeted logic to the targeted logic. Note that k_1 and k_2 are constants for a given design, but are not independent of one another. Unlike the previous expression for power saving in registers, the expression for power saving in clock power can take on negative values. When x has its maximum possible value of 1, the savings in clock power has a value $-\frac{k_1}{1+k_2}$. This means that the circuit will consume more clock power after optimization. By keeping the overhead capacitances low, k_1 can be minimized sufficiently to keep the additional power minimal.

5.3 Overheads in CGC

The overheads in CGC are in re-coding the state machines, the RZ pulse generators, and in the shadow flip-flops. As an estimate for the overhead, we use the number of additional flip-flops that have to be added to the design for CGC. The overhead depends on the nature of the state machines in the design before implementing this scheme. If the initial design had a Moore state machine with N_{states} states as the controller, and n out of these states write to at least one register, then after re-coding the state

machines, the number of flip-flops needed to implement the state machines is given by

$$N_{ff-fsm} = \begin{cases} N_{states}, \text{if } n = N_{states} \\ n + \lceil \log_2 (N_{states} - n) \rceil \text{ if } n < N_{states} \end{cases}$$

For each state that can write to at least one register, we would need an RZ pulse generator and a shadow flip-flop in the worst case. The total number of flip-flops $N_{ff-design-opt}$ in the design after implementing this scheme is

$$N_{ff-design-opt} = N_{ff-fsm} + N_{flip-flops}$$

$$N_{ff-design-opt} = \begin{cases} 3N_{states} + N_{flip-flops} \\ 3n + \lceil \log_2 (N_{states} - n) \rceil + N_{flip-flops} \end{cases}$$

where $N_{flip-flops}$ is the number of flip-flops in the target logic. In the original design the total number of flip-flops assuming binary encoding for the states is given by

$$N_{ff-design} = N_{flip-flops} + \lceil \log_2 N_{states} \rceil$$

The overhead as a fraction of the flip-flops in the original design is given by

$$Overhead = \frac{ff-design-opr - N_{ff-design}}{N_{ff-design}}$$

$$= \frac{3n + \lceil \log_2 (N_{states} - n) \rceil - \lceil \log_2 N_{states} \rceil}{N_{flip-flops} + \lceil \log_2 N_{states} \rceil} \quad (4)$$

As indicated by this expression, the overhead increases roughly linearly with the number of states that write to at least one register. This expression also indicates that if this scheme targets a large number of registers in the circuit, $N_{flip-flops}$ in the denominator would be large and the overheads can be kept low. This also has the positive effect of increasing the power savings.

When the state machines in the original circuit are Mealy State machines, the first step in order to implement CGC is to convert it into a Moore State machine. This step can introduce extra overhead. In general, a Mealy State machine with N_{states} states and qoutput symbols can be converted into a Moore State machine with $N_{states} q + 1$ states [2]. The overhead in this can be computed as earlier. The expression for the overhead in this case is given by

$$Overhead = \frac{3n + \lceil \log_2 (N_{states} q + 1 - n) \rceil - \lceil \log_2 (N_{states}) \rceil}{N_{flip-flops} + \lceil \log_2 (N_{states}) \rceil} \quad (5)$$

In addition to the area overhead derived above, this scheme incurs a small timing penalty on account of skew introduced on each control clock, as described in section 3.

6. Results

In this section we discuss results obtained after implementing CGC in an experimental processor. The processor has an instruction set based on the DLX described in [4]. The processor is coded in VHDL using RTL constructs. The processor design has been partitioned into the following blocks: Register File, ALU, Decode, Fetch and Control. The design is flip-flop based and strictly synchronous, with all flip-flops clocked by the same clock signal. The control state machine has six main states. A gate level netlist was synthesized with a commercial synthesis tool Design Compiler. The target standard cell library has a minimum feature size of 0.18μ. The cells in the library have been characterized for operating voltages in the range 1.65-1.95 Volts. Power analysis of this netlist was done using a commercial tool DesignPower with activity information obtained from netlist simulations. Simulations were carried out using a commercial tool Modelsim. The testcases chosen for simulation had been written so as to cause maximum or minimum activity on the program and data memory paths of the processor.

Several instances of the processor were then synthesized with CGC implemented at the RTL level in each block of the processor. For all instances of the processor, we used the same timing constraints for synthesis. The operating voltage was set to 1.95 Volts (to estimate worst case power) and all designs were clocked at 100Mhz for power analysis. The same test cases were executed on each instance of the processor. The average power consumed by each instance and the gate counts (in terms of equivalent 2-input NAND gates) of each instance are tabulated in figures 5 and 6 respectively. In these figures, DLX represents the baseline instance without any power minimization. DLXA is an instance with CGC implemented in the register file. In addition, DLXB implements CGC in its decode block. DLXC implements CGC in its Fetch and ALU blocks. DLXD implements CGC aggressively for its register file and DLXE implements this in the divider. DLXF is an instance obtained by aggressive use of Design Compiler's gated clock synthesis. For DLXF, we instructed Design Compiler to gate clocks for any register whose width exceeded 5. In DLXA-DLXC, a single control clock is used for the entire register file. In DLXD and DLXE, separate control clocks are used for each register in the register file

From figure 5, it is evident that with aggressive application of CGC, in DLXA-DLXE, significant amounts of power (upto 73.25% in this case) can be saved. One may note that the control power is higher in circuits in which CGC has been applied aggressively. This is due to the fact that the control block dissipates the clock power of datapath blocks with CGC. The higher control power is also due to the increased gate count of the control block. It is also observed that the clock power decreases when CGC is aggressively applied. This is because the datapath blocks derive their clocks from the control block, lowering the capacitance on the clock line. In DLXF, we observe that the Decode block power is nearly the same as in DLX. This is due to the fact that Design Compiler depends on the HDL description to identify idle conditions. The tool sometimes loses out on opportunities for minimization depending on the HDL coding style used. It is observed that Control block consumes very little power in DLXF. This is due to fact that Design Compiler can gate clocks in the control blocks as well. Internally, the tool does not distinguish between datapath and control. The tool uses the information from the HDL for the generation of the gated clock. Finally we observe that aggressive application of CGC can yield about 16% higher power savings than clock gating.

From figure 6 it is evident that CGC achieves a reduction in the gate count of blocks in which it is used. This is because because CGC eliminates the multiplexers used by datapath registers to recirculate data when idle. It is also observed that the gate count of the control block increases due to the overheads associated with CGC. Finally, Comparing DLXE (an aggressive implementation of CGC) against DLXF (an aggressive

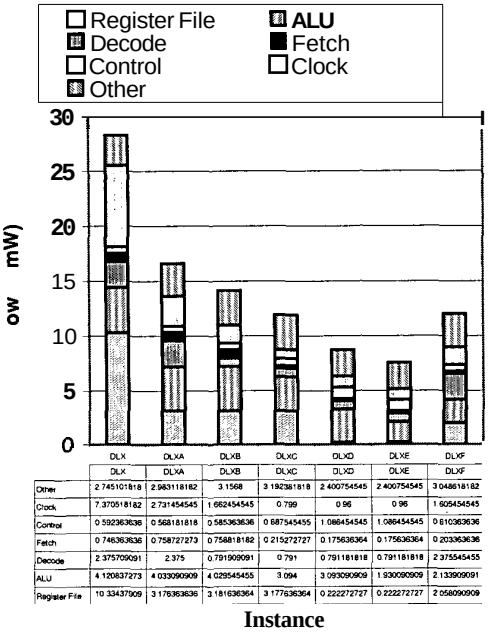

Fig 5. Power dissipation profiles of various DLX instances.

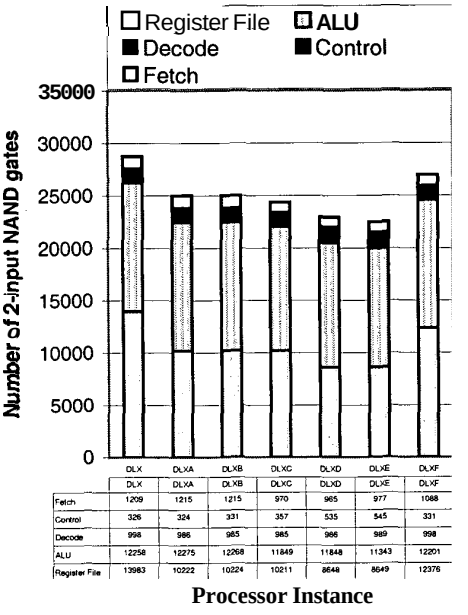

Fig 6. Gate counts of various DLX instances.

implementation of clock gating) reveals that DLXF requires a higher gate count.

7. Conclusions

In this paper, we describe an area efficient power minimization scheme "Control generated clocking" (CGC) for minimizing dynamic power dissipation of registers and clock drivers in a synchronous sequential circuit. CGC minimizes power by utilizing control signals generated by FSMs as clocks for the registers. CGC is a robust method in which power minimization is achieved without the possibility of introducing glitches on the clocks. Timing analysis of a circuit with CGC is simpler than in circuits with gated clocks. CGC preserves the cycle boundaries on which registers load data thereby allowing reuse of test cases developed for the original design without any modifications. CGC is a structured method and can be easily incorporated into a synthesis tool for automation. The utility of CGC has been demonstrated with a complex example.

References

1. Anantha P. Chandrakasan & Robert W. Broderson, Minimizing Power Consumption in Digital CMOS circuits, Proceedings of the IEEE, Vol 83, No. 4, Apr 1995.
2. C. Y. Tsui, M. Pedram & A. M. Despain, Low Power State Assignment targeting Two- and Multi-level Logic Implementation, Proceedings of the ICCAD 1994.
3. John E. Hopcroft & Jeffery D. Ullman, Introduction to Automata theory, Languages, and Computation, Addison-Wesley Publishers, 1979.
4. John Hennesey and David Patterson, Computer Architecture: A quantitative approach, 2nd ed., Morgan Kaufman **Publishers, 1996.**
5. J. Montiero, S. Devadas & A. Ghosh, Retiming Sequential Circuits for Low Power, Proceedings of the ICCAD 1993
6. L. Benini, P. Siegel, & G. De. Micheli, Automatic Synthesis of Gated Clocks for Power reduction in Sequential Circuits, IEEE Design and Test, winter 1994.
7. L. Benini & G. De. Micheli, State Assignment for Low Power Dissipation, IEEE JSSC, vol 11, No. 5, Mar. 1995.
8. Massoud Pedram, Power Minimization in IC design: Principles and Applications, ACM TODAES Vol. 1, No. 1, Jan 1996
9. M. Alidina, J. Montiero, S. Devadas & A. Ghosh, Precomputation based Sequential Logic Optimization for Low Power, IEEE transactions on VLSI systems, Dec. 1994
10. Srikanth Rao M and S. K. Nandy, Controller redesign based clock and register power minimization, Proceedings of the ISCAS (to appear), May 2000, Geneva, Switzerland.
11. Srikanth Rao M, Control generated clocking: A technique for minimizing power in sequential circuits, MSc(Engg) thesis, Research Report RR-CADL-99-08, Apr 2000, CAD lab, SERC, Indian Institute of Science.
12. V. Tiwari, S. Malik & P. Ashar, Guarded Evaluation: Pushing Power management to Logic Synthesis design, Proceedings of the International Symposium on Low Power Design, 1995.