An Intra-Task Dynamic Voltage Scaling Method for SoC Design with Hierarchical FSM and Synchronous Dataflow Model

Sunghyun Lee
Design Automation Lab.
EECS, Seoul National Univ.
Seoul 151-742, Korea
shlee@mithra.snu.ac.kr

Sungjoo Yoo
SLS Group, TIMA Lab.
46 Avenue Felix Viallet
38031 Grenoble, France
Sungjoo.Yoo@imag.fr

Kiyoung Choi
Design Automation Lab.
EECS, Seoul National Univ.
Seoul 151-742, Korea
kchoi@azalea.snu.ac.kr

ABSTRACT

This paper presents a method of intra-task dynamic voltage scaling (DVS) for SoC design with hierarchical FSM and synchronous dataflow model (in short, HFSM-SDF model). To have an optimal intra-task DVS, exact execution paths need to be determined in compile time or runtime. In general programs, since determining exact execution paths in compile time or runtime is not possible, existing methods assume worst/average-case execution paths and take static voltage scaling approaches. In our work, we exploit a property of HFSM-SDF model to calculate exact execution paths in runtime. With the information of exact execution paths, our DVS method can calculate exact remaining workload. The exact workload enables to calculate optimal voltage level which gives optimal energy consumption while satisfying the given timing constraint. Experiments show the effectiveness of the presented method in low-power design of an MPEG4 decoder system.

Categories & Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer Aided Design (CAD)

General Terms: Design, Performance

Keywords: Low power, dynamic voltage scaling, variable supply voltage, formal model, finite state machine, synchronous dataflow

1. INTRODUCTION

In recent SoC (System-on-Chip) design, low-power or power-efficient design and usage of formal models of computation are among the most important issues. Low-power consumption is needed for long battery lifetime, reduction in system maintenance cost (e.g. cooling fan), etc. For such a power-conscious design, recently, significant research efforts have been made on dynamic voltage scaling (DVS) exploiting the quadratic scale of energy consumption to the power supply voltage ($E \propto V^2$). In this paper, we investigate an application of DVS to power-conscious SoC design.

In terms of design productivity, to master the ever growing complexity of SoC design, formal models of computation are becoming more and more important since they enable shorter design cycle by formal analysis (e.g. analysis of liveness, deadlock, maximum memory usage, etc.) as well as systematic reuse. In commercial SoC design tools, several formal models of computation are supported: CFSM in Cadence VCC [1], hierarchical FSM with dataflow in Synopsys CoCentric System Studio [2][3], etc.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

There have been presented many low-power design methods which exploit the formality of computation models to reduce the power consumption. For instance, many DVS methods have been presented for real-time task models [4][5][6][7][8]. These methods identify slack times of task execution (in the initiation or termination of task execution) and scale supply voltage to reduce energy consumption in executing real-time tasks on the processor. FSM models have been used to model systems with power states (e.g. idle, active, sleep, etc.) [9].

In our work, we investigate the application of DVS to SoC design with a popular formal model of computation, hierarchical FSM (HFSM) and synchronous dataflow (SDF) model, in short HFSM-SFD model. The HFSM-SDF model is well suited to designing both complex control (by HFSM) and dataflow computation (by SDF). They also enable useful formal analysis including state reachability test, deadlock analysis with bounded memory, etc. Currently, commercial tools such as Synopsys CoCentric System Studio [2][3] and academic tools such as PtolemyII [10] support the HFSM-SDF model.

Although the HFSM-SDF model is well suited to SoC design, there has been little research on low power design methods, especially, DVS methods for the HFSM-SDF model. In this paper, we first show that the HFSM-SDF has a property well suited to DVS and present a method of intra-task DVS exploiting the property.

2. Hierarchical FSM and Synchronous Dataflow Model

Figure 1 shows an example of HFSM-SDF model. In the figure circles and squares represent states of FSM and SDF actors, respectively. A state (or SDF actor) can be refined to an FSM or an SDF graph. In the figure, at the top of the model, we have an SDF graph consisting of two actors A_1 and A_2. Actor A_1 is refined to an FSM consisting of two states, S_1 and S_2. State S_1 is refined to an SDF graph consisting of two actors A_3 and A_4. In the HFSM-SDF model, FSM and SDF can be nested arbitrarily as in PtolemyII and CoCentric System Studio.

An arc between states represents a state transition. A state transition arc is tagged with a guard/action. An arc between SDF actors is tagged with the numbers of tokens to be consumed (for input) and produced (for output). By default, arcs without numbers have single token production/consumption. In the figure, the rightmost SDF graph has an arc tagged with 2 and 1. In this case, actor A_9 produces two tokens for each firing of the actor and actor A_{10} consumes one token when fired. Thus, to balance the number of produced tokens and that of consumed tokens, the SDF graph has a schedule of actor firing, in short schedule.

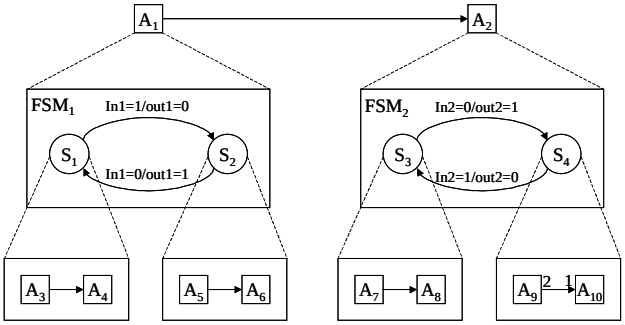


Figure 1: An example of HFSM-SDF model.

There can be several candidates for the schedule. In the above case, we can have $A_9 2 A_{10}$, $2 A_9 4 A_{10}$, etc. For each SDF graph, the designer sets one of candidate schedules as the schedule of the SDF graph.

Details of HFSM-SDF model can be found in [11]. In this paper, to give a brief explanation of HFSM-SDF model execution, we summarize it with three rules, one property, and some terminology as follows.

Rule 2.1 *Corresponding to a state transition of parent FSM, a child sub-FSM makes only one state transition.*

This is a basic rule of hierarchical construction of FSMs.

Rule 2.2 *Whenever a (self or outgoing) state transition is made from a state, if the state is refined to an SDF graph, the schedule of the SDF graph is always executed once.*

This rule is needed to conform to Rule 2.1 when a state is refined to an SDF graph. For instance, in Figure 1, when a (self or outgoing) state transition is made from state S_1, the schedule of SDF graph refining the state, i.e. the schedule of A_3A_4 is always executed once. This rule is necessary to conform to Rule 2.1, especially when a state is refined to an SDF graph and the SDF graphs has an SDF actor which is refined to an FSM. In this case, for a state transition of upper level FSM, the low level FSM should also make a state transition. To do that, the schedule of the intervening SDF graph needs to be executed once.

Property 2.1 *If all the current FSM states are known, the schedule of all the SDF actor firings can be identified.*

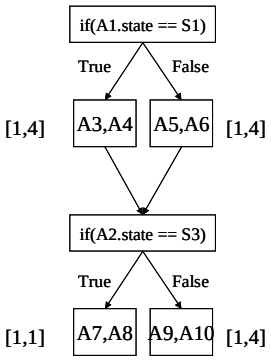
To exemplify Property 2.1, in Figure 1, we assume that the current states of two FSMs, FSM_1 and FSM_2 are S_1 and S_4, respectively. When the top SDF graph executes its schedule, i.e. A_1A_2, the firing of actor A_1 yields a (self or outgoing) state transition from S_1. According to Rule 2.2, the state transition executes the schedule of the SDF graph which refines state S_1, i.e. the schedule of A_3A_4. In the same manner, the firing of actor A_2 yields the schedule of $A_9 2 A_{10}$ since the current state of FSM_2 is S_4. Thus, the total order of actor firing is $A_3A_4A_9 2 A_{10}$.

Since all the current states of FSMs can be identified just before the schedule of top SDF graph starts to be executed, we can obtain the information of SDF actors to be fired. The information is a total order of actor firing.

In terms of execution paths, the total order of actor firing corresponds to an execution path in the execution of HFSM-SDF model. Thus, in the HFSM-SDF model, before executing an execution path, the exact execution path can be identified.

```
1    if(A1.state==S1) {
2        A3();
3        A4(); st1();
4    }
5    else {
6        A5();
7        A6(); st2();
8    }
9    if(A2.state==S3) {
10       A7();
11       A8(); st3();
12   }
13   else {
14       A9();
15       A10(); A10(); st4();
16   }
```

(a)



(b)

Figure 2: A code section of implemented HFSM code.

Based on this information, we can obtain exact workload to be executed, and optimal energy consumption can be achieved in intra-task DVS. Details will be given in Section 3.

Rule 2.3 *When an SDF actor is refined to a sub-FSM, the sub-FSM makes a state transition at the last firing of the SDF actor in the schedule of the parent SDF graph.*

This rule is needed to conform to Rule 2.1 when an SDF actor is refined to a sub-FSM. Rule 2.3 means that there are two types of actor firing for the SDF actor refined to a sub-FSM: one (called TypeA firing in [11]) that does not have the sub-FSM make a state transition and the other (called TypeB firing) that enables the state transition.

For more details of HFSM-SDF model, more generally, HFSM with concurrency models, refer to [11].

Figure 2 (a) shows a code section that implements the HFSM-SDF model in Figure 1 (Figure 2 (b) shows the corresponding control structure, and worst-case execution time of each basic block.). The schedule of top SDF graph is assumed to be A_1A_2. In the figure, line 1-8 corresponds to the execution of actor A_1 and line 9-16 that of actor A_2.

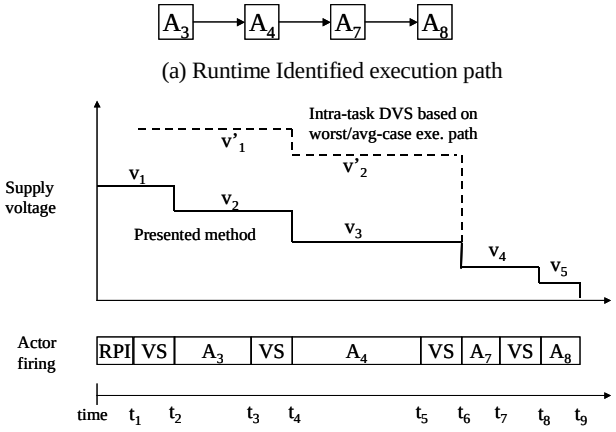
If the current state of the FSM refining the actor A_1 is S_1, then according to Rule 2.2 and 2.3, the refining SDF graph of state S_1 executes once a schedule of SDF actors, i.e. function A3() and A4() (line 2-3). Function st1() represents the guard evaluation of state transition and the change of states (line 3). If the current state of the FSM refining actor A_1 is S_2, then the refining SDF graph of state S_2 executes function A5() and A6() (line 6-7).

When actor A_2 is fired, if the current state of the FSM refining actor A_2 is S_3, then the refining SDF graph of state S_3 executes a schedule of SDF actors, i.e. function A7() and A8() (line 10-11). If not, it executes function A9() and two times of function A10() (line 14-15).

3. Proposed Method

3.1 Interleaving SDF Actor Firing and Voltage Scaling

Figure 3 exemplifies the execution of HFSM-SDF model and DVS. We assume that a deadline constraint D is given to the execution of the HFSM-SDF model. Each schedule of top SDF graph should meet the deadline constraint D, e.g. D=1/25 second for a single frame operation of an MPEG4 decoder system.



(a) Runtime Identified execution path

(b) Intra-task DVS based on exact execution paths

Figure 3: Intra-task dynamic voltage scaling.

We scale the supply voltage on an actor basis. That is, at the beginning of each actor execution, the supply voltage is scaled.

Before each schedule of top SDF graph, the information of actors to be fired is obtained. In Figure 3 (a), we assume that the example code of Figure 2 is executed and the current execution path includes two basic blocks, A3, A4 and A7, A8. The worst-case execution path is when basic block A9 and A10 are executed.

In the example of Figure 3, the total order of actor firing is A_3, A_4, A_7 and A_8. In the figure, arrows represent execution order. For instance, A_3 should be executed before A_4.

Actor firing is interleaved with voltage scaling as exemplified in Figure 3 (b). At the beginning of schedule of top SDF graph, a function called runtime execution path identification (RPI) gives information of SDF actors to be fired for the current schedule of top SDF graph.

Using the information, a voltage scaler (VS) calculates the current workload by summing all WCETs (worst-case execution times) of actors to be fired. Then, it calculates an initial level of supply voltage (i.e. speed ratio, more exactly) using the current workload.

At time t_2, the supply voltage is scaled to the initial voltage level (V_2 in the figure) by the voltage scaler. After the voltage scaling, the first SDF actor A_3 executes until time t_3.

After the termination of actor A_3, the voltage scaler calculates a new level of supply voltage (i.e. new speed ratio) to exploit a time slack obtained after the execution of actor A_3 (due to the variable execution time). The new voltage level is calculated considering the time slack, the current workload, and the runtime overhead of voltage scaling. Details will be given in Section 3.3. Then, the voltage scaler sets the supply voltage to a new level (V_3 in the figure). The actor firing and voltage scaling continue in this way.

The voltage scaling methods based on worst/average case execution paths is also exemplified as dashed lines in Figure 3 (b). These methods assume a worst-case execution path (when basic block A9, A10 are executed) at the beginning of HFSM-SDF model execution. Thus, the initial voltage level V_1' for the execution of Actor A_3 is higher than that of our method V_2. For the level of supply voltage for the execution of actor A_4, the methods based on worst/average-case execution paths give higher voltage level (V_2') than our method

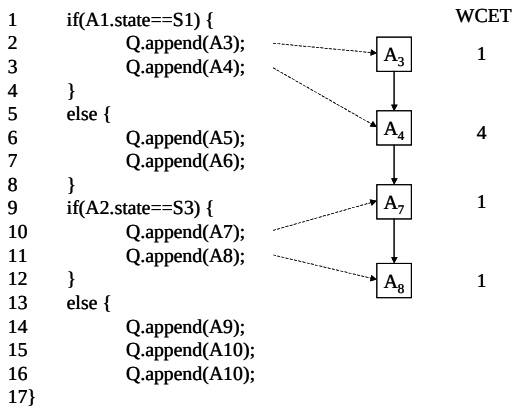


Figure 4: An example of RPI function.

(V_3) since it is not determined yet whether basic block A7, A8 or basic block A9, A10 will be executed, and the worst-case execution path (where basic block A9, A10 is assumed to be executed) is assumed yielding a higher voltage level.

In the next two subsections, we will explain the details of runtime execution path identification and dynamic voltage scaling.

3.2 Runtime Execution Path Identification

Figure 4 shows a pseudo code of RPI function for the HFSM-SDF model of Figure 1. The pseudo code has the same control structure as the code of HFSM-SDF example given in Figure 2. From the original code of HFSM-SDF model, the construction of RPI function is straightforward. It is constructed by extracting the hierarchy of HFSM (control structure of the original code), and state variables (to read them) as exemplified in Figure 4.

Before each schedule of top SDF graph, since we can evaluate all the current states of FSMs in the HFSM-SDF model, we can build the information of actors to be fired as a queue whose element contains information of an SDF actor, i.e. actor id and worst-case execution time (WCET) of the actor. In the case of Figure 6, assuming that the states of FSMs are S_1 and S_3, a queue of four actors is obtained as shown on the right-hand side of the figure.

Dashed arrows represents a correspondence between a line in the RPI function and an element of the queue obtained by executing the line. For instance, Q.append(A3) in the code of RPI function appends to the queue, Q an element containing the id of actor A_3 and its WCET (1 in the figure).

Note that the RPI function is executed once before the iteration of each schedule of top SDF graph since it is needed to identify all the actors to be fired before the iteration. The queue obtained by the RPI function execution is used by the voltage scaler to calculate the current workload.

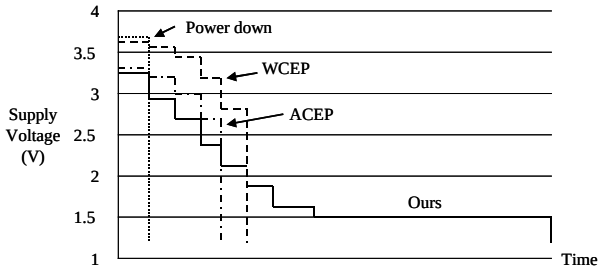
3.3 Dynamic Voltage Scaling

By examining the queue obtained by the RPI function, the voltage scaler calculates current workload (W_C) by simply summing up the worst-case execution time of actors to be fired. With the given deadline (D) of the iteration of top SDF schedule, the voltage scaler determines the speed ratio (R) by the following simple heuristic equation (which is shown to be safe with real-time in [5]).

$$R = W_C \Big/ (D - k \cdot T_{RPI} - n \cdot T_{VS})$$

Power down	WCEP	ACEP	Ours
214 mJ	174 mJ	158 mJ	138 mJ

(a) Comparison of energy consumption



(b) Intra-task DVS based on exact execution paths

Figure 5: Intra-task dynamic voltage scaling.

where,
W_C: sum of execution time of remaining actors to be executed.
T_{RPI}: worst-case execution time of RPI function.
T_{VS}: worst-case execution time of voltage scaler (VS).
k: 1 if this is the first invocation of voltage scaler, 0 otherwise.
n: the number of actors remaining to be executed.

4. Experiments

In our experiments, we build an HFSM-SDF model of an MPEG4 natural video decoder [12]. Our implementation of MPEG4 decoder consists of 9 hierarchical FSMs, 31 states, 44 state transitions, 89 SDF actors (10 hierarchical actors and 79 leaf actors). We run 10 frames of MPEG4 decoding with a reference motion picture.

As a target processor where the MPEG4 decoder runs, we use an ARM8 variable speed processor [13]. We set up an energy simulator with a commercial instruction set simulator, ARMulator and an energy model of the processor [14].

We set the deadline of MPEG4 decoder system to be the worst-case execution time of one frame decoding. The MPEG4 decoder system performs computation during voltage scaling. The rate of voltage scaling is 26.9us/V. We scale the supply voltage between 3.8V and 1.2V. When there is no computation to run, the processor power is down.

We compare our method with three other methods: power down only, WCEP (worst-case execution path)-based method, and ACEP (average-case execution path)-based method. Figure 9 (a) shows the comparison. Compared with the case of power-down only, our method gives 35.5% reduction in energy consumption. Compared with WCEP/ACEP-based methods, ours give 20.7% and 12.7% more reduction in energy consumption.

Figure 5 (b) shows the voltage scaling of four methods for an operation of intra-frame decoding of MPEG4 decoder system. As shown in the figure, our method gives a lower initial voltage than three other methods since it can utilize the exact execution path, i.e. the exact workload. In terms of operation frequency, our method gives 75MHz and 69MHz as the initial operation frequency values for inter-frame and intra-frame decoding, respectively, while the other three methods should set the maximum frequency value to 80MHz since the worst-case execution path is assumed at the beginning.

The RPI function and voltage scaler have also overhead in terms of runtime, code size and energy consumption. The code size overhead is negligible (about 4%). However, the runtime and energy consumption overhead is 21.5% and 20.4% respectively. To obtain further reduction in energy consumption, the overhead needs to be minimized.

5. Conclusion

We presented a method of intra-task voltage scaling for SoC design with a hierarchical FSM and synchronous dataflow (HFSM-SDF) model. The HFSM-SDF model gives a property which enables to identify an exact execution path in runtime before the execution of schedule of top SDF graph (or state transition of top FSM). Exploiting the property, optimal voltage scaling can be obtained. In dynamic voltage scaling, the overhead of voltage scaling is accounted for. Experimental results show that compared with existing two solutions, the presented method yields 20.7% and 12.7% more reduction in the energy consumption of an MPEG4 decoder system.

6. REFERENCES

[1] Cadence Design Systems, Inc., "Virtual Component Co-design (VCC)," available at http://www.cadence.com/products/vcc.html.

[2] Synopsys, Inc., "CoCentric System Studio," available at http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html.

[3] J. Buck and R. Vaidyanathan, "Heterogeneous Modeling and Simulation of Embedded Systems in El Greco," *Proc. Int'l Workshop on Hardware-Software Codesign*, pp. 142-146, May 2000.

[4] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power Optimization of Variable Voltage Core-Based Systems," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 12, pp. 1702-1713, Dec. 1999.

[5] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Design Automation Conf.*, pp. 134-139, 1999.

[6] G. Quan and X. Hu, "Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors," *Proc. Design Automation Conf.*, pp. 828-833, 2001.

[7] D. Shin, J. Kim, and S. Lee, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications," *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 20-30, Mar. 2001.

[8] C. Im, H. Kim, and S. Ha, "Dynamic Voltage Scheduling Technique for Low-Power Multimedia Applications Using Buffers," *Proc. Int'l Symposium on Low Power Electronics and Design*, pp. 34-39, Aug. 2001.

[9] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli, "Dynamic Voltage Scaling and Power Management for Portable Systems," *Proc. Design Automation Conf.*, pp. 524-529, 2001.

[10] "The Ptolemy Project," available at http://ptolemy.eecs.berkeley.edu/.

[11] B. Lee, "Specification and Design of Reactive Systems," *Ph.D thesis, Memorandum UCB/ERL M00/29, Electronics Research Laboratory, Univ. of California, Berkeley*, May 2000.

[12] "MPEG4 Industry Forum," available at http://www.m4if.org/.

[13] T. D. Burd and R. W. Brodersen, "Design Issues for Dynamic Voltage Scaling," *Proc. Int'l Symposium on Low Power Electronics and Design*, pp. 9-14, Aug. 2000

[14] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1571-1580, Nov. 2000.