Functional modeling techniques for efficient SW code generation of video codec applications

Sang-Il Han* ** Soo-Ik Chae* Ahmed. A. Jerraya**

*Department of Electrical Engineering,
Seoul National Univ., Seoul, Korea
{sihan,chae}@sdgroup.snu.ac.kr

**SLS Group, TIMA Laboratory
Grenoble, France
{sang-il.han,ahmed.jerraya}@imag.fr

Abstract–Architectures with multiple programmable cores are becoming more attractive for video codec applications because they can provide highly concurrent computation and support multiple video standards and a shorter time-to-market. To find an efficient SW code for the multiple core architecture for a video codec application, it is very important to easily explore the design space by generating a SW code automatically from its functional model.

We introduce Abstract Clock Synchronous Model (ACSM) for functional modeling of video codec applications. The ACSM can easily represent both parallelism and conditionals, which are common in video codec applications. By applying ACSM to an H.264 baseline decoder on single core architecture, we reduced the execution time and the number of external memory accesses by 32 % and 46 % respectively compared to traditional dataflow model.

I. Introduction

Current video codec applications require a higher performance architecture that can process more complex algorithms for higher resolution images. To find an architecture that satisfies this requirement with a limited resource, it is essential to use functional modeling that can exploit deep pipelines and parallelism and maximize re-use of the limited resource. Furthermore, it is important to use a programmable architecture that supports multiple video formats and newly emerging standards. To improve the design productivity, both SW and HW codes should be generated automatically if possible because the time-to-market is getting shorter. Therefore, automatic SW code generation from a functional model is one of the essential technologies in system design. In this paper we focus on a functional modeling method to generate an efficient SW code from the functional model for a video codec application.

Four properties of the video codec applications should be addressed in functional modeling [1].

1) **Computation intensive operations**, such as motion compensation, sub-pel interpolation, and DCT transform, which should be executed within timing constraints.
2) **Massive data transfer operations**, e.g. for motion estimation and compensation.
3) **Data dependent operations** according to various image modes and macroblock (MB) modes.
4) **Iterative execution** for sub-macroblock, macroblock, and frame levels, which requires large memory buffers.

To generate an efficient SW code for the video codec applications with these properties, functional model should support the following requirements.

1) Parallelism and pipeline should be exploited using specific parallel architectures in order to perform computation-intensive operations within timing constraints. Therefore, a functional model should enable to represent intra- and inter-iteration dependencies explicitly to exploit parallelism and pipeline respectively.
2) Communications should be expressed explicitly and the sizes of data transfers should be predictable in the functional model in order to efficiently use burst data transfers and data pre-fetches that are essential for the video applications.
3) A functional model should support conditionals such as if-then-else structure in order to efficiently represent conditional computation and communication of video codec applications.
4) A functional model should represent communication buffers explicitly in order to minimize a memory cost by allocating a minimal size of memories and reusing the buffers.

In this paper, we propose Abstract Clock Synchronous Model (ACSM), which is an extension of the Clocked Synchronous Model for RTL modeling [2]. The ACSM employs a coarser clock to compose functional blocks, which will be mapped onto HW blocks or SW functions on a specific CPU core. By using the coarser clock, the ACSM can represent parallelism and conditionals of video codec applications while the existing data-driven models and event-driven models have difficulty to express conditionals and parallelism respectively. First we explain the basics of ACSM for video applications. Then we compare ACSM with previous functional modeling methods in detail. We will show its efficiency by comparing it with conventional dataflow methods on a single core architecture in terms of performance, and communication bandwidth.

The rest of the paper is organized as follows. In Section II we explain ACSM, a proposed solution suitable for modeling video codec applications. In Section III ACSM is compared with other functional models in detail. In Section IV we present several experimental results to check the efficiency of ACSM, which is followed by the conclusions in Section V.

II. Abstract Clock Synchronous Model

A. Assumptions

Fig. 1 shows the overall steps of automatic SW code generation from functional model. Three key design steps are 1) building a functional model of a target application, 2) mapping of the functional model to a target architecture, and 3) generation SW code automatically from the mapping result. The mapping step is based on an evaluation function where the inputs are a functional model and a target architecture and the outputs are evaluation metrics such as performance, power and cost. Therefore, to generate efficient SW code from a functional model, the target architecture should be taken into consideration in building a functional model.

In this paper we assume that a target architecture is composed of processor (and HW) subsystems for image processing, global memory subsystems for image store, and an interconnection for communication between subsystems, as shown in Fig. 2. A processor (or HW) subsystem is composed of a processor (or HW IP), a local memory, an interconnection interface, and a local bus.

0-7803-9451-8/06/$20.00 ©2006 IEEE.

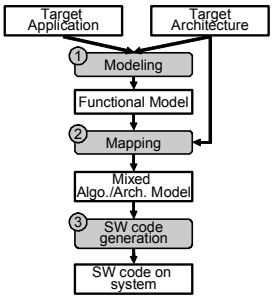

Fig. 1. The overall steps of automatic SW code generation.

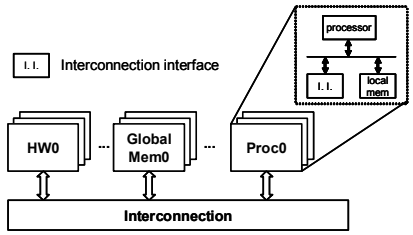

Fig. 2. Generic target architecture.

Basically, the target architecture is a loosely coupled architecture, so that a processor (and HW) subsystem should copy necessary image data from its global memory subsystems to its local memory before processing it.

We limit the target applications of ACSM to macroblock-based video codecs, such as, MPEG-2, H.263, MPEG-4, and H.264 [1]. These video codecs combine inter-picture prediction to exploit temporal redundancy with transform-based codec of the prediction errors to exploit spatial redundancy. Fig. 3 shows a block diagram of an H.264 decoder that receives an encoded video bit stream from a network or a storage device and produces a frame sequence. Each frame is reconstructed by iterative executions of macroblock-level functions such as entropy decoding, inverse zigzag scan, inverse quantization, inverse transform, motion compensation, and deblocking filter.

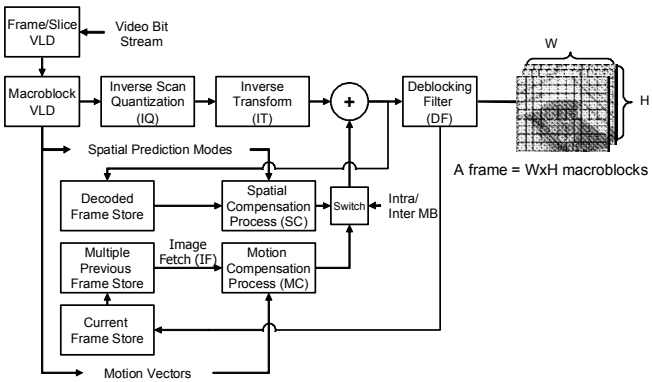

Fig. 3. Target application example: H.264 decoder block diagram.

In this paper, we focus on the functional modeling step for efficient SW code generation of video codec application.

B. *Abstract clock synchronous model*

The Clocked Synchronous model (CSM), which has been used in designing the hardware for clocked synchronous circuits, is based on the **clock synchrony hypothesis** [2]: There is a global clock signal controlling the start of each computation in the system, and communication takes no time, and computation takes one clock cycle. This assumption makes it possible to describe the functionality of a circuit deterministically independent of the detailed timing of the gates in the circuit by separating each combinational logic block from others with clocked registers. In other words, the CSM is used to exploit the orthogonalization between functionality and timing in the synchronous design methodology [4]. In this paper we extend the CSM to the ACSM by using an abstract clock of larger granularity that is suitable for system-level design.

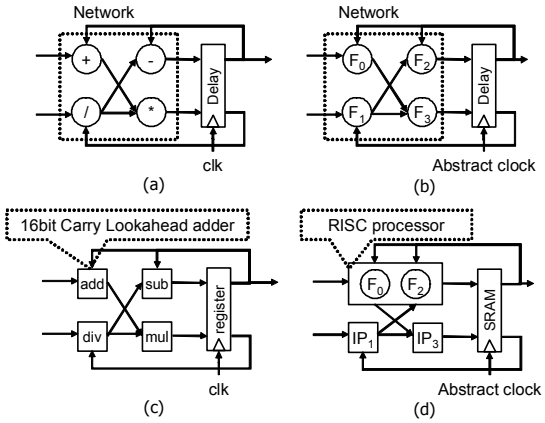

Fig. 4. Examples of (a) clocked synchronous model, (b) abstract clock synchronous model and their implementations (c-d).

Fig. 4 (a) shows an example of CSM for RTL modeling with a clock and Fig. 4 (b) shows an example of ACSM for functional modeling with an abstract clock. A CSM is composed of a network of combinational gates and delays. It is implemented by low level hardware as shown in Fig. 4 (c). For example, an addition and a delay in the CSM can be implemented by a 16bit carry lookahead adder and a register respectively. However, an ACSM is composed of a network of state-less functions and delays. It may be implemented by a combination of hardware and software as shown in Fig. 4 (d). For example, a function and a delay in the ACSM can be implemented by a SW code on RISC processor and an SRAM respectively. The major difference between the two models is the granularity of the clock and the components. Note that cyclic paths must contain at least one delay in both models.

C. *Tagged signal model of ACSM*

In order to describe ACSM and compare it with the existing functional modeling methods, we follow the tagged-signal model introduced in [3]. Given a set of *values* V and a set of *tags* T, an *event* e has a tag t and a value v, i.e. $e = (t, v) \in T \times V$. A *signal* s is a set of events. The tags are used to model time, precedence relationships, and synchronization points. The values represent the operands and results of computation.

In the ACSM, it is necessary to represent intra- and inter-iteration dependencies explicitly in order to exploit parallelism and pipeline as above mentioned. To do this, we use a set of tags $T \in \omega \times \omega$, where ω is the set of nonnegative integers with the usual numerical order. The set of the first components of all events is totally ordered. The first component is used to model data precedence between operations across an abstract clock

boundary, i.e. inter-iteration dependencies. The set of the second components of all events is partially ordered. The second component is used to model data precedence between functions within an abstract clock interval. The set of all event tags is partially ordered because of the second component.

Let $e_{i,j}$ denote an event where the tag t is (i, j) and the value v is $e_{i,j}$. Given two events $e_{i,j}$ and $e_{n,m}$, $e_{i,j} < e_{n,m}$ if (i, j) < (n, m). Fig. 5 shows an example of precedence relationships between events in an ACSM. F_1 imposes a precedence constraint such that $e_{i,j1} < e_{i,j3}$. Delay imposes a precedence constraint such that $e_{i,j5} < e_{i+1,j2}$. But there is no precedence relationship between $e_{i,j3}$ and $e_{i,j4}$, and $e_{i,j1}$ and $e_{i+1,j0}$. The partially ordered events give rise to parallel and pipelined execution of functions in an ACSM

A data type D can be extended into a data type $D_\perp$ by adding the special value $\perp$ to model the absence of a value at a certain tag. Absent events are used to model the outputs of unselected operations in if-then-else structures in the ACSM.

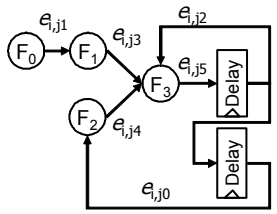

Fig. 5. An example of ACSM with events.

D. Abstract clock for video codec application

In the ACSM, it is important to select an abstract clock suitable for explicitly representing parallelism, communication, and communication buffers for efficient design space exploration. Video decoding (and encoding) processes are basically composed of four hierarchical iterations: sub-macroblock level, macroblock level, slice level, and image level. Each of these iteration indices can be a candidate for the abstract clock. Two iteration indices of slice and image levels are too coarse to represent parallelism, and communication explicitly between the essential functions of video codec applications such as motion estimation, motion compensation and inverse transform. Using the iteration index of 4x4 sub-macroblock level requires representing irregular delays due to the data dependencies among 4x4 blocks. For example, for a QCIF image as shown in Fig. 6, the delay between block 5 and its upper block is 166 delay units, but that between block 13 and its upper block 7 is 6 delay units. Therefore, we use the macroblock index as an abstract clock in an ACSM because the granularity of the macroblocks is good enough to represent parallelism, communication, and communication buffer explicitly and the decoding order of macroblocks as shown in Fig. 6 (a) is regular so that we can represent easily the data dependency.

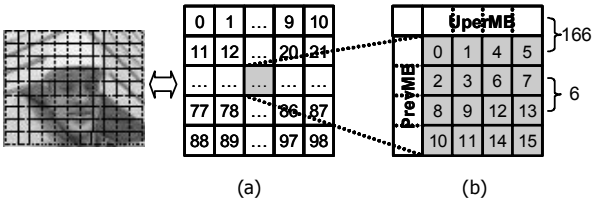

Fig. 6. Decoding orders of macroblock within an image (a), and 4x4 blocks within a macroblock (b).

E. Basic components and firing rules of ACSM

We decided to use Simulink [5] for a video codec application because it provides simulation and modeling environment of discrete-time systems that are sufficient to build an ACSM. It also includes Real-Time Workshop (RTW) [6], which generates a C code automatically from a Simulink model.

Fig. 7 shows basic components of ACSM that are expressed easily in Simulink.

- **Block** (process): A block, as shown in Fig. 7 (a), maps n input events on m output events: $(o_1, \ldots, o_m) = F_0(i_1, \ldots, i_n)$. It corresponds to S-function or pre-defined block with inherent sample rate in Simulink.
- **Delay**: A delay, as shown in Fig. 7 (b), represents that its output is delayed from its input by k abstract clock cycles. It corresponds to discrete delay in Simulink.
- **Arc** (edge): An arc carries events from one output port of a block or a delay to one or more input ports of one or more blocks and/or delays as shown in Fig. 7 (c). It corresponds to connecting line in Simulink.

We also defined two kinds of subsystems that are composed of blocks, delays, arcs, and other subsystems.

- **If-action subsystem (IAS)**: An IAS, as shown in Fig. 7 (d), represents an if-then-else structure. An IAS is enabled when its control input port, which is connected to an *if/else* block, has a present event, i.e. not absent. If an IAS is not enabled, its output ports have absent events. All output ports must be connected to a *merge* block and only one of them can have a present event at a time. It corresponds to "If-action subsystem" in Simulink.
- **For-iterator subsystem (FIS)**: A FIS, as shown in Fig. 7 (e), represents a for-loop structure. It is used to describe sequential or parallel repeated executions of blocks where the number of repetitions is known. It corresponds to "For-iterator subsystem" in Simulink. A FIS usually includes *Demux*s and *Mux*s. A *Demux* divides an event into several (sub-)events. A *Mux* integrates several (sub-)events into an event.

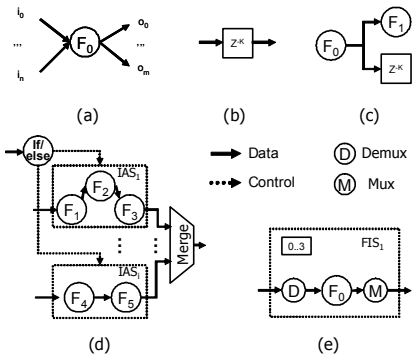

Fig. 7. Basic components in ACSM.

A block consumes one event from each input port and produces one event to each output port. This action is called *firing* and takes place under certain conditions called *firing rules*. A block except *merge* blocks in Fig. 7 (d) is fired when all input events are present. A *merge* block is fired when one of input events are present. A block except *if/else* blocks in Fig. 7 (d) produces one present event to each output port when fired. A *if/else* block produces one present event to one of output ports when fired.

F. An example of ACSM

Fig. 8 shows a simplified ACSM in Simulink for an H.264

baseline profile decoder. It includes two paths. A path consists of macroblock VLD (MB VLD), 8x8 sub-macroblock inverse quantization (8x8 IQ), and inverse quantization (8x8 IT) to compute a residual image from a video bit stream. The other path consists of MB VLD, spatial compensation (SC) or motion compensation (MC) from the current frame or previous frames. If four neighbor 4x4 sub-macroblocks, e.g. 0, 1, 2, 3 blocks in Fig. 6(b), have the same motion vector, it is possible to fetch less image data from previous frames (8x8 IF in IAS_2) and manipulate the image data more efficiently (8x8 MC in IAS_2) compared to 4x4 sub- macroblock based motion compensation (4x4 IF and 4x4 MC in IAS_3) by the elimination of common computation and communication. The ACSM consists of 83 S-functions, 286 arcs, 21 *if/else* blocks, 43 IASs, 5 FISs, and 24 delay blocks.

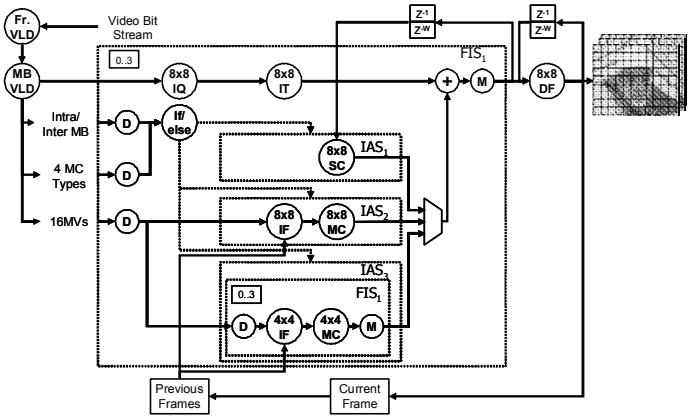

Fig. 8. A simplified ACSM of H.264 decoder in Simulink.

IV. Comparison with previous modeling styles

In order to obtain an efficient SW code from a functional model, it is important to select an appropriate functional modeling style according to the property of an application domain. In the following subsections, we will compare ACSM with the most popular previous modeling styles in respect of building functional models of video codec applications. The capabilities and the weakness of these models will be analyzed with regards to the requirements stated in section I.

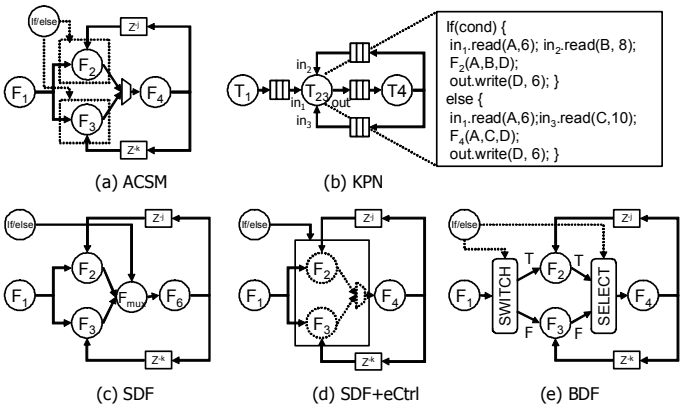

Fig. 9. Functional model examples of different styles.

A. Kahn Process networks

In the Kahn process networks [7], concurrent processes communicate through one-way FIFO channels with unbounded capacity. This means that writes to the channel always succeed immediately, while reads block until there is sufficient data in the channel to satisfy them. In particular, a process cannot test an input channel for the availability of data and then branch conditionally.

Fig. 9 (b) shows a KPN example corresponding to an ACSM model as shown in Fig. 9 (b). A process in a KPN model has a computation code mixed with a communication code. Therefore, The KPN doesn't support explicit and predictable communication that is required to efficiently use burst data transfers and data pre-fetchs. It also requires context switching to deal with blocked processes. To reduce the overhead of context switching, it is necessary to increase the granularity of process.

B. Synchronous Dataflow

In the Synchronous Dataflow (SDF) model [8], a process (actor) is fired when it has sufficient tokens (events) on its input ports. When an actor executes, it consumes a positive fixed number of data tokens from each input port, and produces a positive fixed number of tokens to each output port. If a SDF model is consistent, it is possible to execute the SDF model in bounded memory without context switching.

SDF cannot represent explicit conditional such as if-then-else structure, which is required for video codecs, because it doesn't allow absent token. To express an if-then-else structure, redundant computation and communication may be required as shown in Fig. 9 (c). According to the result of "if/else" actor, either F_2 or F_3 does not need to be executed because just only one of the outputs is used. Similarly, either communication between F_2 and Z^{-j} or that between F_3 and Z^{-k} is redundant.

Fig. 9 (d) shows an alternative to express an if-then-else structure with embedded controls in a SDF model, where blocks F_2, F_3, and an if-then-else structure are merged into a single block. It can remove unnecessary execution of F_2 or F_3, but redundant communication of F_2 with Z^{-j} or F_3 with Z^{-k} is still required. Furthermore, mapping F_2 and F_3 onto two different processors or HWs is not possible so that its design space is restricted. Therefore, it does not support explicit conditional well.

C. Boolean Dataflow

Boolean Dataflow (BDF) [9] is an extension of SDF to support conditionals. In the BDF model, an if-then-else structure is modeled with two actors, SWITCH and SELECT. The SWITCH actor reads one token from the control input port, and depending on whether the value of the control token is true or false, routes the input either to the output port marked T, or to the output marked F. It also produces an absent token to the other output port.

However, it is not guaranteed whether the execution of a BDF model is completed in a finite time or whether it requires a bounded memory [9]. In Fig. 9 (e), the tokens on the arc between F_2 and Z^{-j} and those on the arc between F_3 and Z^{-k} are accumulated because the execution ratios of F_2 and F_3 to F_4 are different depending on the control token produced by the if/else actor. To solve this problem, it is necessary to insert additional SWITCH actors both between F_2 and Z^{-j} and between F_3 and Z^{-k}. Building a BDF model of a video codec application at fine granularity will require many SWITCH actors, which is more error prone.

D. Synchronous model

Synchronous model (SM) [10] used in Esterel [11] and Lustre [12] is based on the perfect synchrony hypothesis assuming that the reaction to each set of the inputs is considered to be instantaneous. In the SM, a process is fired when it has at least one event on its input ports. When a process is fired, it can test absent events on its input ports and produce absent events on its output ports.

The synchronous assumption simplifies system specification and verification. However, difficulties arise especially for video codec applications if the target architecture is a distributed multi-core system, because it is very expensive to maintain a global clock for testing and producing absent events over a distributed system. The fully synchronous implementation based on time-triggered architecture [13] must be conservative, forcing the global clock to run as slow as the slowest computation and communication process. Therefore, SM is not suitable for video codec applications, which have large variations in computation and communication.

E. Analysis: comparing with ACSM

The ACSM overcomes all the restriction of the above mentioned models while providing the same scheduling facilities.

In data-driven models of KPN and SDF, the absent token is not defined. Therefore they have some difficulties in expressing an explicit if-then-else structure. In the BDF model the token overflow problem comes from the definition of the absent token without a global clock. However, the ACSM uses an abstract clock for the well-defined absent token. In the ACSM, an event on an arc is updated at every abstract clock tick, so there is no token overflow problem.

Although the dataflow model has difficulties in representing conditionals, it can express several valid schedules of a multi-rate algorithm with only a model. In a SDF model as shown in Fig. 10 (a), a schedule that requires minimal buffers is $F_0F_0F_1F_2F_2F_0F_1F_2F_2$ and another schedule with loop structures is $(3F_0)(2F_1(2F_2))$. Contrary to the dataflow model, the ACSM can express only one schedule with a model. Fig. 10 (b) shows an ACSM example that expresses a schedule that is $(3F_0)(2F_1(2F_2))$. However, it is possible to find other schedules by loop transformation techniques such as loop unrolling and loop split.

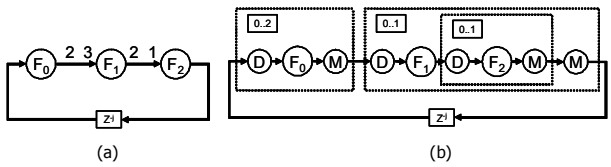

Fig. 10. Multi-rate modeling with SDF (a) and ACSM (b).

In the SM, there is no restriction on producing and testing absent events. It can cause a causality problem and requires a global clock to solve the problem. However, the ACSM allows only restricted absent events to express if-then-else structures. In other words, it is sufficient for a distributed multi-core system to replace absent events of unselected IAS outputs with present events only if an if-then-else structure is implemented over different processors.

V. Experiments

We performed several experiments to check the efficiency of the ACSM compared to the dataflow model, assuming that target architecture consists of a processor subsystem for image processing and an external global memory for image storage, as shown in Fig. 11, where a processor subsystem consists of a Tensilica Xtensa processor [14] with a default configuration, a local memory, and a DMA with a memory interface. We assumed that image fetch processes are executed in the DMA module.

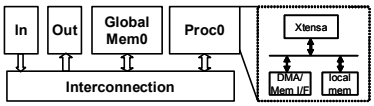

Fig. 11. Evaluation architecture.

The target application is an H.264 baseline decoder and its input bitstream is the Foreman sequence of QCIF format, which was encoded with QP=28 and IntraPeriod=5 when all MC block sizes were enabled.

Fig. 12 shows our experimental procedure. In the modeling step, we made three Simulink models from H.264 decoder reference C code as shown in Table I. In the Simulink model equivalent to a SDF model, an if-then-else structure is expressed by using a MUX block as shown in Fig. 9 (c) and the processing block size is 4x4. In Simulink model equivalent to an ACSM, an if-then-else structure is expressed by an *if/else* block and IASs and the processing block size is 4x4. In the optimized ACSM, the processing block sizes are either 4x4 or 8x8. For 8x8 block, an additional IAS is necessary as shown in Fig. 8. We limited the block size of the SDF model to 4x4 because the additional IAS causes redundant computation as explained in IV-B.

Then, we used Real-Time Workshop (RTW) to generate SW codes from the three different Simulink models. In the final step, we used Xtensa gdb and gprof to measure the execution cycles and the external memory access of the SW codes. We also added the reference C code to the test sets.

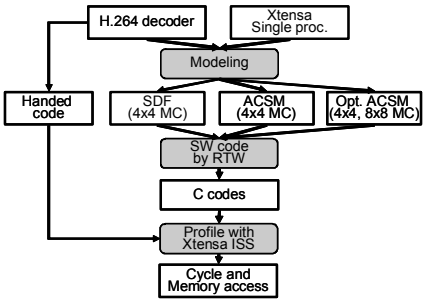

Fig. 12. Experimental procedure.

TABLE I four test sets with different configurations

Name	description
SDF	SDF model as shown in Fig. 9 (c)
ACSM	Proposed ACSM model as shown in Fig. 9 (a)
Optimized ACSM	Proposed ACSM model with 8x8 block size as shown in Fig. 8
Handed code	Reference C code

Fig. 13 shows the execution time and the number of external memory accesses for the four different configurations. According to the experimental results, the ACSM reduces both the execution time and the number of external memory accesses by 29 % and 22 % respectively compared to those of the SDF model. Furthermore, the optimized ACSM reduces them by 32 % and 46 % respectively compared to those of the SDF model.

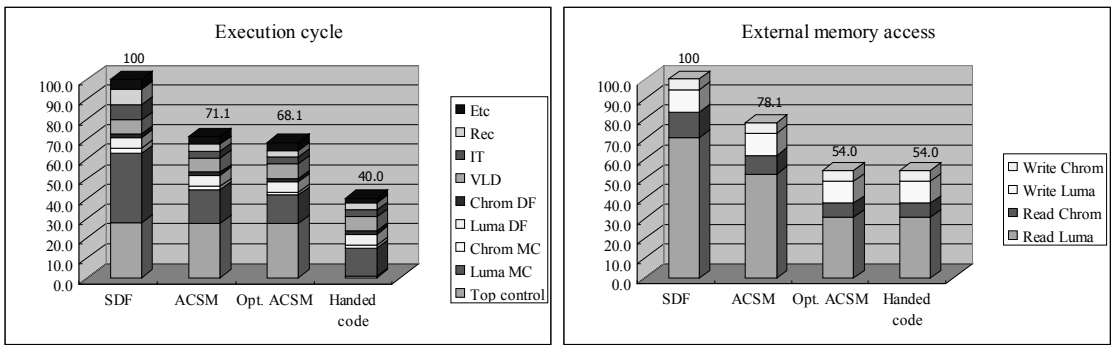

Fig. 13. Relative execution times and external memory accesses of different configurations.

According to the experimental results, it is necessary to express conditionals such as if-then-else structures explicitly for building a functional model of a video codec application. The optimized ACSM code from RTW requires 41% more execution time compared to the handed code because it includes many redundant copy operations.

Even if a SDF model with embedded controls can remove redundant computation, it cannot remove the external memory access as explained in section IV. It also limits the design space exploration due to increasing the granularity of blocks.

For the BDF model of the H.264 baseline decoder, 138 SWITCHes were required to resolve the token overflow problem mentioned in section IV. The number of the SWITCHes in the BDF model is larger than that of S-functions in the ACSM model, which means that although the BDF supports conditionals, it is not suitable in building a functional model for a video codec application because the BDF functional models get too complex.

VI. Conlusions

In this paper, we explained a functional modeling method for generating an efficient SW code from a functional model for video codec applications. It is based on ACSM, which is an extension of the clocked synchronous model by employing the macroblock index as an abstract clock. The ACSM can express conditionals easily by allowing absent events with the global abstract clock. It can also express parallelism and pipeline easily by partially ordered intra- and inter-dependencies. Therefore, the ACSM is suitable for functional modeling of video codec applications that require both parallelism and conditionals.

Experimental results with Simulink and RTW showed that the SW code generated from an ACSM of H.264 decoder is improved by up to 32% and 46 % compared to its SDF model in terms of the number of execution cycle and the number of external memory access respectively. We found that a BDF model of H.264 decoder requires many additional SWITCHes to express if-then-else structures. So the BDF model gets more complex. Therefore, the ACSM is more effective in building functional model for video codec applications because emerging standards such as H.264 require complex data-dependent operations.

For a H.264 baseline decoder, the SW code generated with the ACSM by RTW requires longer execution time and larger buffer memories compared to those of the handed code. We are in the process of developing a more efficient SW generation tool. Furthermore, we will extend this methodology for multi-processor systems because the current version of RTW can generate only a SW code for single-processor systems.

Acknowledgements

The authors are grateful to Xavier Guerin, Paul Amblard, Frédéric Pétrot and Nacer-Eddine Zergainoh from TIMA laboratory for their inputs on this paper. This work was supported by ITSoC Project and Brain Korea 21 Program.

References

[1]“Advanced video coding for generic audiovisual services,” Int. Telecommum. Union-Telecommum. (ITU-T) and Int. Standards Org./Int. Electrotech. Comm. (ISO/IEC) JTC 1, Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4) AVC, 2003

[2]Axel Jantcsh, “Modeling Embedded Systems and SoCs – Concurrency and Time in Models of Computation,” Morgan Kaufmann, 2001.

[3]E. A. Lee and A. Sangiovanni-Vincentelli, “A Framework for Comparing Models of Computation,” IEEE Trans. On CAD of Integrated Circuits and Systems. pp 1217-1229, December 1998.

[4]K. Keutzer et al, “System-level design: Orthogonalization of concerns and platform-based design,” IEEE Trans. On CAD of Integrated Circuits and Systems.

[5]Simulink,http://www.mathworks.com/

[6]Real-Time Workshop, http://www.mathworks.com/

[7]G. Kahn and D.B. MacQueen, “Coroutines and Networks of Parallel Processes,” In B. Gilchrist, editor, Information Processing 77, Proceedings, pp 993-998, Toronto, Canada.

[8]Lee, E. A., Parks, T. M. (1995), “Dataflow process networks,” Proceedings of the IEEE83(5), 773-801

[9]J.T. Buck, “Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model,” PhD thesis, University of California, EECS Dept., Berkeley, CA, 1993. Technical Memorandum UCB/ERL M93/69

[10] Benveniste, A. et al, “The synchronous languages 12 years later,” Proc. of the IEEE , Volume: 91 Issue: 1 , Jan 2003

[11] Gerard Berry, “The Foundations of Esterel,” Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tolfte, editors, MIT press, 1998

[12] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud. “The synchronous dataflow programming language Lustre,”. Proc. of the IEEE, vol. 79, nr. 9. September 1991.

[13] H. Kopetz, “The time-triggered architecture,” in Proc. First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’98), Kyoto, Japan, 1998.

[14] Tensilica Xtensa V, http://www.tensilica.com/html/xtensa_v.html