Concepts of high level synthesis using SystemC

K. Pakalniškis, E. Kazanavičius

Kaunas University of Technology, Digital Signal Processing Laboratory, Department of Computer Engineering, Studentu 50, LT 3031 Kaunas, Lithuania, {ekaza,pakal}@dsplab.ktu.lt

Introduction

Real time applications impact the design of embedded systems with many different types of constraints, including timing, size, weight, power consumption, reliability, accuracy and cost.

Current methods for designing embedded systems require to specify and to design hardware (HW) and software (SW) separately. A specification is often incomplete and written in different languages are send to the hardware and software engineers. Partition of SW / HW is designed *a priori*. Generally software is used for features and flexibility and hardware is used for performance. Any changes in this partition may extend a redesign cycle and time to market.

Digital Signal Processing (DSP) [1] is one of the fastest growing fields in modern electronics used in us any areas. Application areas include: image/video processing, speech-audio processing, telecommunications, biomedical applications and others.

Functional specification for adaptive systems is mostly written in languages like C or C++. This software model has no timing, size and cost constraints information. The systems implementations models mostly are written in hardware description languages like VHDL, Verilog.

In Fig.1 non unified system design flow is shown. The redesign gap between functional verification to architectural verification is covered by high level synthesis. Present high level synthesis systems inputs C/C++, HDL behavioural and generates register transfer – RT level HDL description. The system is non unified.

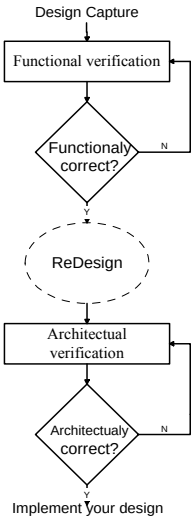

Fig. 1. **Non unified design flow**

For system level specification on 1999 leading EDA, IP, semiconductor systems and embedded software companies announced the "Open SystemC Initiative" of a C++ modeling platform called SystemC [7].

SystemC is the standard design and verification language built in C++ that spans from concept to implementation in hardware and software. Designers design and verify using SystemC and standard ANSI C++.

In this paper we describe the concepts of a high level synthesis using adaptive system models designed with the SystemC. This concept makes the unified system design methodology starting from functional verification to synthesis and architectural verification.

Design models of the system

Each system model could be described in different type of description. Depending on the model description may be impractical or meaningless. One of the system exploration charts (Y-chart) is shown in Fig.2. The Y-chart has three domains of design description: behavioural, structural and physical. Each domain has many levels of abstraction. The design process is represented step by step refinement in all the three domains from the outer level to the center.

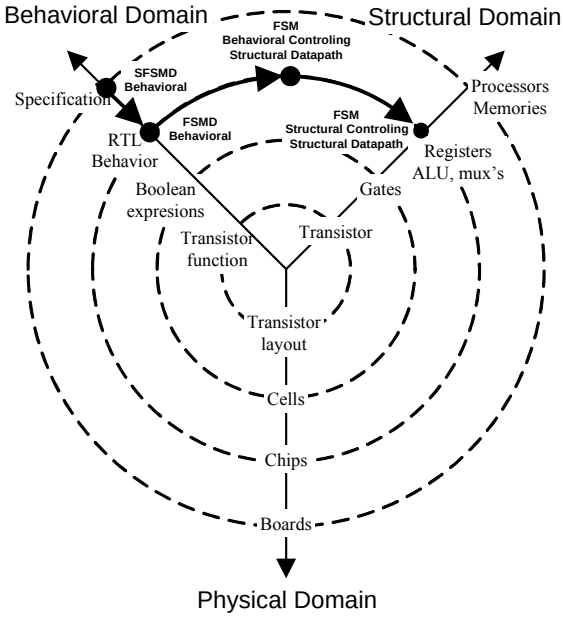

Fig. 2. **System exploration Y-Chart**

The Super Finite State Machine with Datapath (SFSMD) is the most abstract level of description. The whole algorithm can be considered to be the SFSMD with one superstate. But generally, the algorithm is divided into any number of parts of any size. These parts are the

superstates. The basic difference between the two models is, that SFSMD does not restrict the size of the algorithm parts assigned to a state, whereas FSMD does. This is, because SFSMD does not correspond to hardware at all, whereas the states of FSMD correspond to clock cycles.

Behavioural description

For behavioural description, the algorithm, which is chosen to be implemented in hardware must be correct and executable on any platform, but no restrictions concerning a special structure apply. The whole algorithm could be assigned to one single big superstate, but in most cases several superstates are recommended. When describing the algorithm in a high level programming language, it is typical to map the procedures to superstates. Fig.3 shows an example with several superstates.

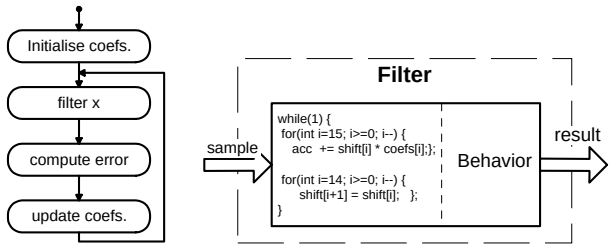

Fig.3. **SFSMD of adaptive algorithm (left - representation, right - description)**

Structural description

A structural description is not used for this model. Hardware allocation and scheduling are not done at this time. Therefore a structural description just could be something like a general purpose computer which is capable of executing the appropriate program which is stored in some memory (Y- Chart). This is not helpful for implementing a adaptive algorithm.

Finite state machine with datapath - FSMD

In the model Finite State Machine with Datapath (FSMD) scheduling is performed. That means the algorithm is divided into small pieces which are assigned to cycles using cycle based states [6]. FSMD- cycles will be mapped to the cycles of the hardware clock. However hardware cycle time is not finally set yet. In each state two things are done:

- The operation scheduled to this state is executed within one clock cycle.
- The next state for the next clock cycle is determined.

Behavioural description

The FSMD Behavioural description is shown in figure 4. The Behavioural description is preferred for this model, because of better readability. It clearly illustrates how the code pieces are scheduled in the states. It is located at the RTL Behaviour point of the Y-Chart. The term "RTL" (Register Transfer Level) indicates the cycle accuracy of this description and the corresponding register transfers of the final hardware.

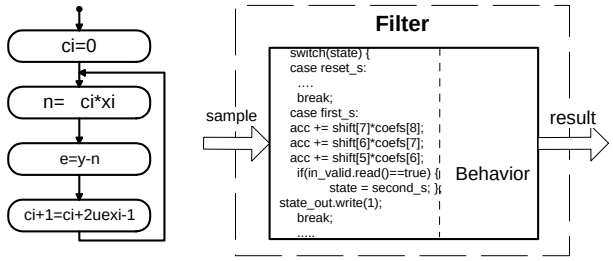

Fig.4**. FSMD of adaptive algorithm (left - representation, right - description)**

The variables of this description will be assigned to final hardware (register, memory, input, output, wires). A structural description is not used for this model.

FSM controlling data path

The system is described on the same refinement level as the previous model. The FSMD is splitted into a control part, described by a Finite State Machine (FSM) without datapath and a separate datapath. The control block controls the execution of the operations while the datapath actually performs them.

Depending on the current state, the control block sets control signals for the datapath. These signals tell the datapath, how operations to be processed. The datapath can inform the control block about special results. Depending on the input signals and the current state, the control block determines the next state, which becomes valid in the next clock cycle. The separation into these two blocks offers new possibilities - each can be independently described behaviourally or structurally.

Behavioural description

A pure behavioural description of this model is used only for simulation, not usable for synthesis. When trying to describe data path behaviorally this would generate a second state machine for the data path (see Fig. 5). The model FSMD is used instead.

Behavioural - structural description

The big advantage of the split model is the possibility of still using a behavioral description for the control, while describing the datapath structurally (Fig. 6). The first block illustrates the states in accordance with the Register Transfer Level while the second contains the data structure: memory, registers, ALU and etc.

This description is less comprehensive than a behavioural description of the model FSMD, but this approach to the final architecture has to be used. Physical hardware finally is obtained by implementing a structural description. Therefore we finally need a pure structural description. The data path, which is by far the largest part of the design, is structural already.

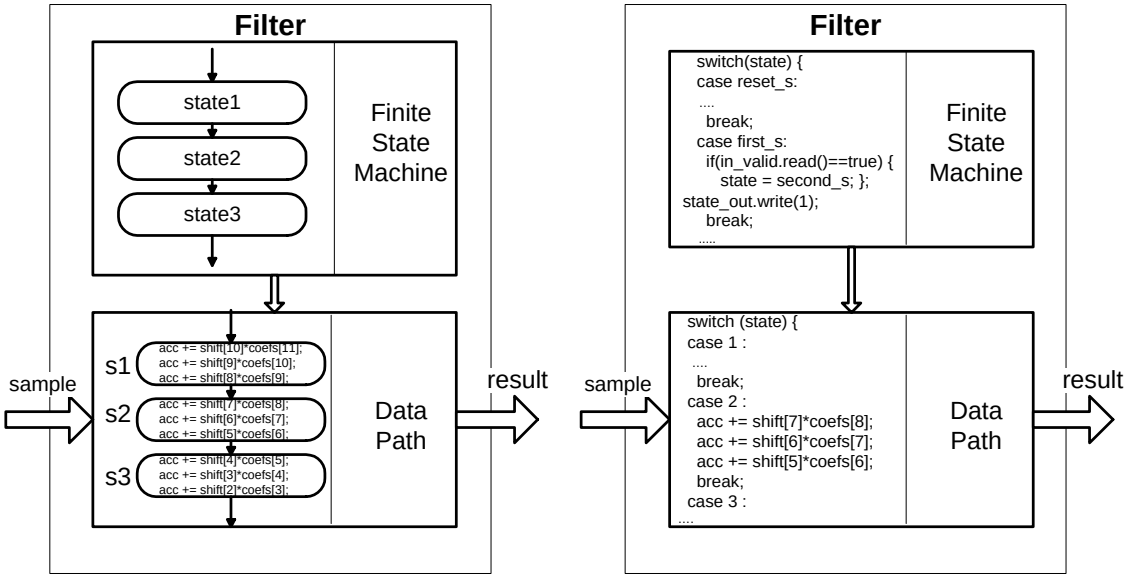

Fig. 5. **FSM Controlling Data path. Behavioural. (left - representation, right - description)**

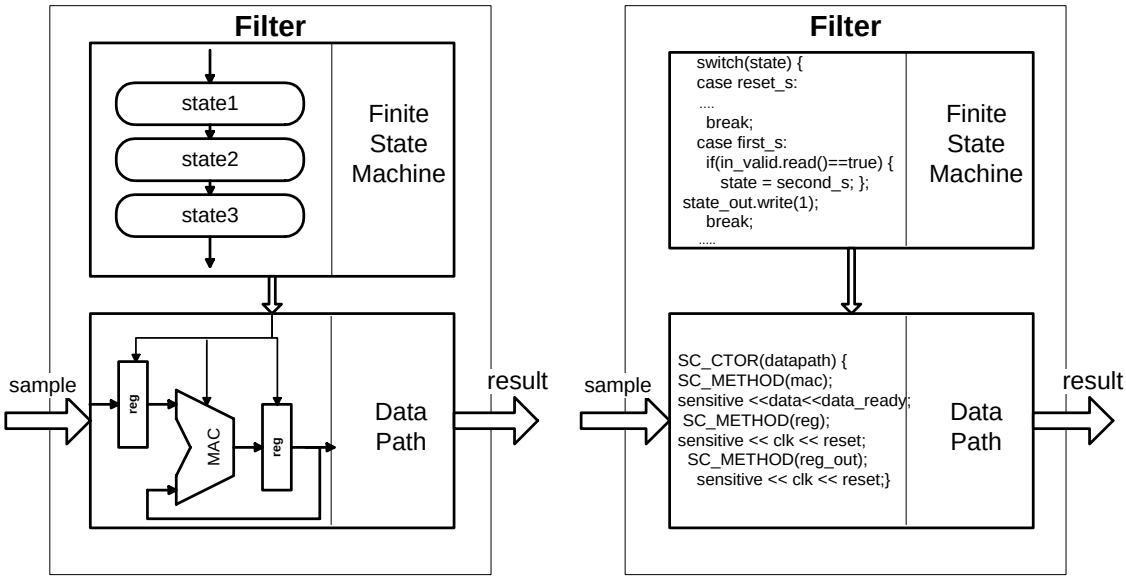

Fig. 6. **FSM Controlling Data Path. Behavioural-Structural. (left - representation, right - description)**

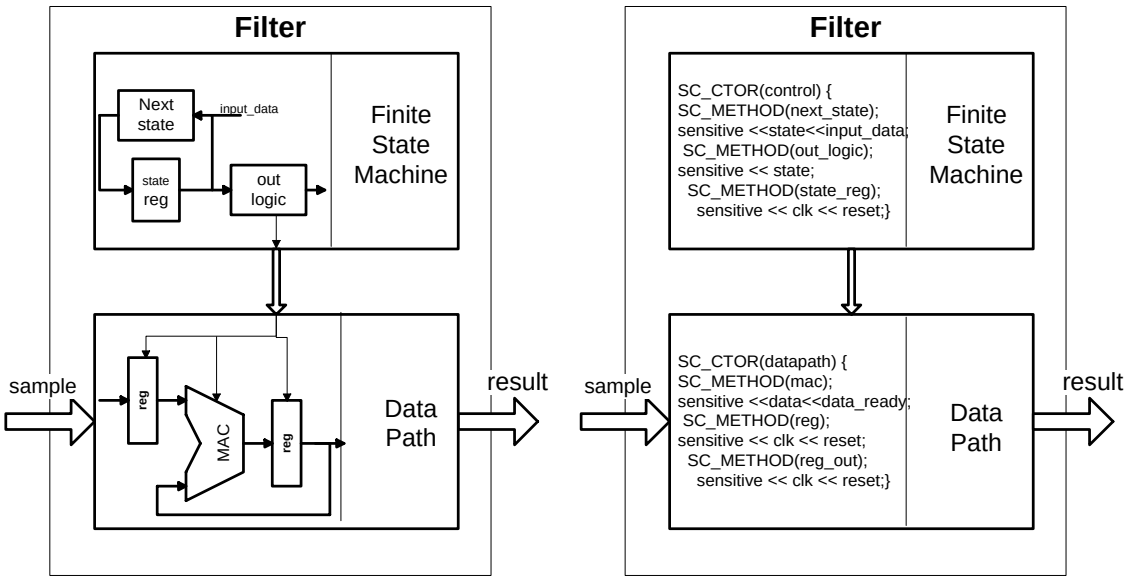

Fig. 7. **FSM Controlling Data Path. Structural-Structural. (left - representation, right - description)**

The behavioural control description is simple and can be converted to a structural description by a simple synthesis tool. The split model with behavioural control and structural datapath is a good basis for synthesis tools which have some intelligence about how to convert behavioural description into a structure.

Structural description

Synthesis tools with no intelligence may need a pure structural description. In Fig.7, even the control part is structural. The control block shown here implements the control FSM too.

A pure structural description should be used only if the synthesis tool or the input language does not permit a behavioural description of the control block.

Synthesis flow

The high level synthesis is transformation from a high level description of the system into a lower level called the Register Transfer - RT level [9]. The RT components are interconnected and satisfy the functionality and provides some constraints such as quantity of computational resources and timing information, etc., number of cycles to perform the algorithm.

The Y-chart (see Fig.1) dotted line circles are the abstraction levels of the system design. Outer level is most abstract level and down to center is more granular system design. The system design chart has three domains.

Specifying definition of high level synthesis we assume, it is transformation from Behavioural domain most abstract level (SFSMD – behavioural description) system description into Structural domain lower abstract level – RT level (FSM controlling datapath – structural description) system description.

High level synthesis consists of three major parts:

- Scheduling. Scheduling partitions the behavioural description into time divided algorithm pieces.
- Allocation. Allocation determines the number of the resources (memory, functional units- FU, buses).
- Binding assigns the variables to memory units, operations to FU and interconnections to buses.

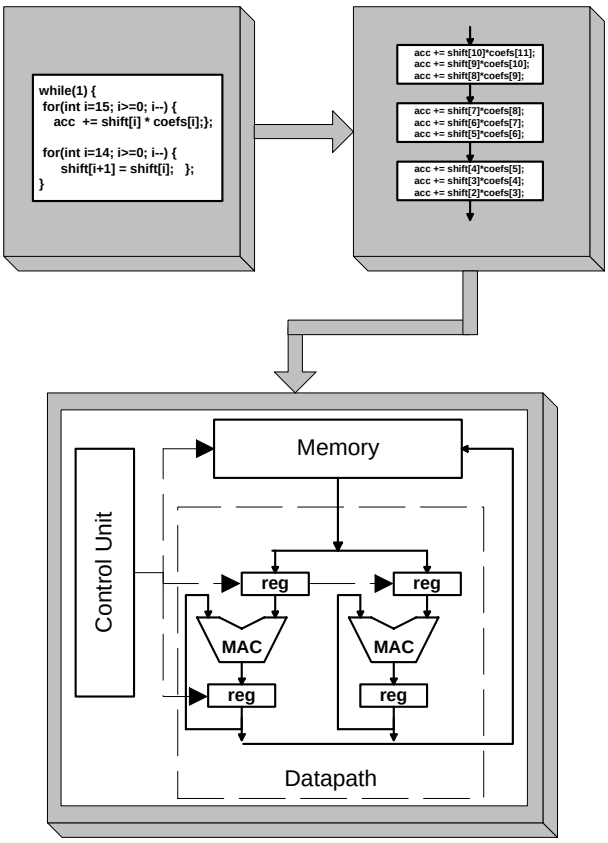

Fig. 8. **Basic steps of high level synthesis**

Abstract steps of synthesis methodology are shown in Fig. 8. The first step is input of the system described as SFSMD (behavioural). The next step is scheduling. The algorithm is partitioned into pieces which are assigned to cycles using cycle based states. FSMD – cycles will be mapped to the cycles of hardware. The last step is allocation and binding. Now is deciding the number of resources necessary to fulfil timing of the previous model (FSMD behavioural description). The resources have to be banded. The variables, operations, interconnections of partitioned pieces of the algorithm assigns to storage units, FU and buses.

Discussion and future challenges

Various system models described using SystemC are presented in the chapter 2. The models SFSMD, FSMD and FSM Controlling Data Path presents different kind of information.

The three basic steps of synthesis strategy are presented in the chapter 3.

The concepts presented in the chapters shows possibility to use the SystemC, the classes of C++ to represent all needed steps in a high level synthesis. Using the unified environment for system specification, verification and synthesis eliminates gap between the designers and tools.

These challenges to explore and create the real high level synthesis system, are very useful for an adaptive computing.

Conclusions

The paper has shown the concepts of high level synthesis using SystemC. This concepts allows use the same environment for all steps in the system design, beginning from functional verification to architectural exploration. The high level synthesis system is based on the same environment as other system design steps, eliminates transformations from one description languages to another and simplifies the synthesis process.

References

1. **Ifeachor E. C., Jervis B. W.** Digital signal processing: a practical approach. Addison Wesley. 1993. P.541-576.
2. **Semeria L., Ghosh A.** Methodology for Hardware/Software coverification in C/C++. Proceedings of Asia and South pacific design automation conference. Yokohama, January 2000. P.405-408.
3. **Economakos G., Oikonomakos P., Panagopoulos I.** Behavioral synthesis with SystemC. Proceedings design, automation and test in Europe conference 2000. Munich, Germany. March 13-16, 2001. P.21-25.
4. **Gerlach J., Rosenstiel W.** System level design using the SystemC modeling platform. Workshop on system design automation. Rathen, Germany. March 2000. P.185-189.
5. **Mueller W., Ruf J., Hoffmann D.** The simulation of SystemC. In Proceedings of design automation and test in Europe (DATE), IEEE Computer Society Press, Los Alamitos. Munich, March 2001.
6. **Zhao Sh.** RTL Modeling in C++ , UC Irvine, Technical Report ICS-01-18 April 30, 2001.
7. Functional Specificatio for SystemC. Synopsys, Inc., CoWare, Inc., Frontier Design, Inc. and others. Final Version2.0-M January 17, 2001.
8. Describming Synthesisible RTL in SystemC. Synopsys, Inc, Version 1.0 May, 2001.
9. **Gajski D., Dutt N., Lin S., and Wu A.** High level synthesis: Introduction to chip and System design. Kluwer Academic Publishers, 1992.

K. Pakalniškis, E. Kazanavičius

***System*C taikymo aukšto lygio sintezei koncepcija**

Reziumė

Darbe pateikta aukšto lygio sintezės, atliekamos naudojant aparatūros aprašymo kalbą SystemC, koncepcija. Pateiktos projektavimo metodologijos skaitmeninio filtravimo uždavinių klasei.

Pateikta spaudai 2002 07 5

DOI: 10.5755/j01.u.43.2.8121