Asynchronous Interaction between FSM and Dataflow Models

Dohyung Kim, Soonhoi Ha
Department of Computer Engineering, Seoul National University
Seoul, Korea 151-742
{dhkim, sha}@iris.snu.ac.kr

Abstract

In this paper, we focus on communication protocols which integrate control modules and function modules. Among others, asynchronous interaction enables FSM model to control the scheduling of dataflow model and to change the parameters in the dataflow modules. Compared with previous approaches, the proposed technique supports formal specification for each component and provides protocols which integrate the two models in a flexible way.

1 Introduction

As the system complexity of electronic digital systems grows, a new systematic design methodology, called codesign, has been sought for exploring optimal architectures [1]. The specification languages used in the codesign researches so far depend on the application areas they target for. For computation-oriented applications such as DSP applications, dataflow models gain popularity. On the other hand, for control-oriented applications, the Finite State Machine (FSM) or its variants are favored. Recently there emerges a new class of applications that contain significant amounts of computation modules and control modules simultaneously.

We propose a cospecification method that specifies control modules and function modules with FSM and dataflow respectively, and integrates them via communication protocols on the codesign backplane [2]. This paper focuses on the details of communication protocols between FSM and dataflow models.

To define the communication requirements between dataflow and FSM models, we examine the interaction patterns between control modules and function modules in typical embedded systems. Regardless of implementations, interaction patterns between control modules and function modules can be divided into two categories: one is *synchronous interaction* and the other is *asynchronous interaction*.

In a synchronous interaction, a control module and a function module communicate with each other by exchanging data samples. The function module may send a sample to set a flag of a control register and the controller becomes active on the arrival of the flag. Or, the control module may send a data sample which activates a function module to process the sample.

Using asynchronous interactions, the control module plays the role of supervisory module to manage the state of the dataflow module in figure 1(i). We define three states of a dataflow module by its current activity: *active*, *suspend* and *stop*. If there is no synchronous input interaction from the FSM module to a dataflow module, the dataflow module goes into the active state from the start by default. When the control module enters into a certain state, it may want to suspend the dataflow module and resume it later. When the dataflow module goes into the suspend state, it stops its execution and just discards the incoming samples from the outside blocks.

Another situation of asynchronous interaction occurs like figure 1(ii) when the control module wants to change some parameters of dataflow nodes asynchronously. Suppose a dataflow module decodes and plays an encoded audio. If the user lowers the volume, that action is delivered to the control module and finally to the dataflow module by changing the "gain" parameter of the dataflow node that amplifies the sound samples.

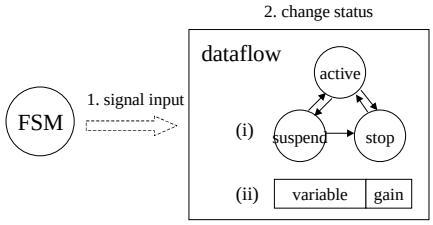

Figure 1: Transmission of asynchronous control signal

In the next section, we introduce previous researches and compare them with our approach. Section 3 introduces the codesign backplane and shows implementation details of asynchronous interaction. We will show the practicality of our work through an example and conclude the paper in sections 4 and 5 respectively.

2 Previous Work

The Ptolemy software environment [3] supports multiple models of computation including dataflow and FSM models. It enables a model of computation to contain another models of computation inside as illustrated in figure 2.

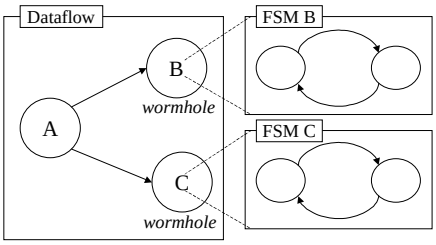

Figure 2: Interaction between FSM and dataflow in Ptolemy

Interactions between different models of computation are achieved by the wormhole mechanism. If the block "B" gets a data from the block "A", the wormhole delivers the data to internal FSM B, which defines a synchronous interaction. Ptolemy does not support asynchronous interaction directly. If an FSM node contains a dataflow module, the dataflow module is executed while the FSM node is active. If FSM nodes contain different dataflow modules with different parameter values, it can express parameter update though it is not an intuitive expression. Other asynchronous interactions can not be specified easily in Ptolemy.

COSY [4] environment provides a unified specification of control and functional modules for IP-based real-time design methodology. COSY uses an extension of Kahn process network, called Y-chart API (YAPI) as the specification language. YAPI model consists of a network of processes connected by FIFOs.

A process has *read*, *write* and *select* port operations. Read and write operations provide synchronous interaction between components as in the Kahn process network. Select operation is added to express data dependent behavior of a process. It gets two input ports as arguments and chooses one of the input non-deterministically. Figure 3 shows an usage of select operation for asynchronous interaction in COSY. If there is no data in control path, the block "MPEG Decoder" processes data and sends output. But if a control data arrives at a control port, the block accepts the control data through select operation and processes it, which may be scheduling control or internal parameter update.

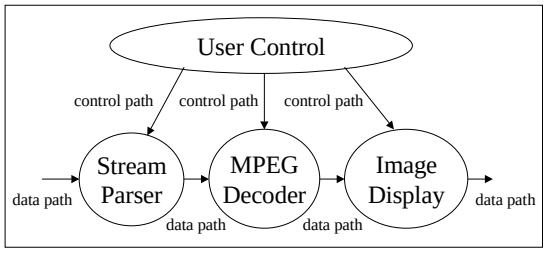

Figure 3: Interaction between components of an MPEG player in COSY

COSY uses a unified specification of control modules and function modules that are assumed to be predefined IPs. But predefined IPs must have a control module inside to process a control data from the select port to combine control path and data path.

STATEMATE of i-Logix inc. [5] has been developed for complex control-oriented reactive applications. It consists of three charts: statechart [6], activity chart, and module chart. The module chart represents the system architecture. The statechart, an innovative extension of FSM, is the major module that depicts reactive behavior over time. The activity chart, corresponding to the codesign backplane in our context that will be discussed in the next section, describes the functional decomposition and the information flow of the system.

In the activity chart of figure 4, real lines show synchronous interactions between the statechart and function modules, dotted lines indicate asynchronous interactions which are emitted by a state transition in the statechart.

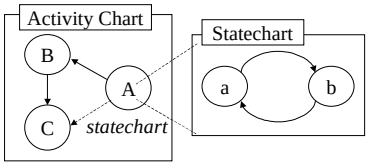

Figure 4: Interaction between statechart and function modules in the activity chart of STATEMATE

Statemate provides very flexible interaction mechanism. But because it focuses on control-oriented reactive systems, function modules in the activity chart are dependent on the statechart. And both activity chart and statechart do not have formality.

In short, no previous approach satisfies all requirements of formal composition of dataflow and control. The proposed technique is to use the codesign backplane to provide formal specification for control modules and function modules like Ptolemy, and support both synchronous and asynchronous interactions like Statemate.

3 Codesign Backplane

We implement the proposed technique in our codesign environment PeaCE(Ptolemy extension as a Codesign Environment) [7] that is based on the Ptolemy software environment. The codesign backplane manages all interactions between different models of computation. We adopt the discrete-event model of computation [8] for the codesign backplane. The interactions between dataflow modules and FSM modules should go through the top-level co-design backplane. Therefore the codesign backplane can implement proposed synchronous and asynchronous communications.

In a synchronous interaction, a control module and a function module communicate with each other by exchanging data samples. Since the receiving module waits for the arrival of samples at a certain state, the interac-

tion is synchronized with sample exchange. Though there are many possible methods to describe asynchronous interactions, we choose to use a "*script*" language inside a state in the control FSM. Table 1 shows the syntax of the scripts. The first three manage the states of the dataflow modules and the last script describes the asynchronous parameter updates of dataflow modules.

How to accomplish an asynchronous interaction is shown in figure 5. If the FSM make a transition to a new state, the codesign backplane receives a script command from the FSM. The backplane interprets the command and sends a special control signal to the specified dataflow block whose node_name property is equal to the *n_node* field of the script. However, this control signal path is not drawn explicitly in the graphic editor. There is a hidden connection between the backplane and all blocks in the graph. Each dataflow module checks the arrival of the special control tokens from the backplane regularly. If a special token is received, it changes the graph activity as indicated. Otherwise, it continues to execute the graph.

Scripts	Actions
Run *n_name*	Resume the *n_name* block
Suspend *n_name*	Suspend the *n_name* block
Stop *n_name*	Stop the *n_name* block.
Set *n_name parameter value*	Update the *parameter* with *value* in the *n_name* block

Table 1. Scripts for asynchronous interaction

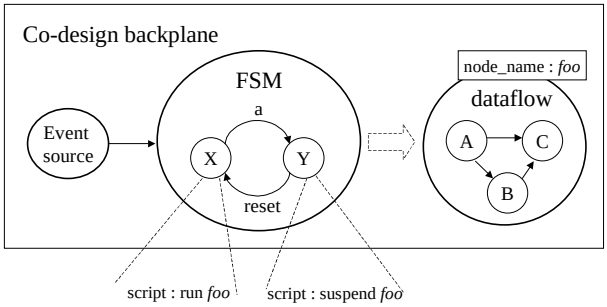

Figure 5: Asynchronous interaction with a normal dataflow module

If a dataflow module is synthesized as a C code, a connection is established via TCP/IP to the codesign backplane. We divide it into two situations for effective simulation. If there is any synchronous connection between FSM and the dataflow module, the C process is synchronized with data from FSM. But if there is no synchronous connection, the C process can concurrently be executed with the codesign backplane.

The situation when only asynchronous interactions exist is illustrated in figure 6 which shows the synthesized dataflow module of figure 5. Because there is no data dependency between the codesign backplane and the synthesized C process, two processes are executed simultaneously. Since it does not need to wait for a control packet from FSM, we use non-blocking read operation while the C process runs. But if FSM changes the state of the synthesized dataflow module to the "suspend" or "stop" state, the process blocks and waits for a control packet which make it active again, using blocking read.

```
C code process
  init_variable:
  [ variable initialization ]

  while (true) {
    if (status==RUN) {
      non-blocking read for a control signal
      if (control packet==SUSPEND) {
        status=SUSPEND;
        continue;
      } else if (control packet==STOP) {
        status=STOP;
        goto init_variable;
      } else process parameter update
    } else {   // status==SUSPEND || STOP
      blocking read for control a signal
      if (control packet==RUN) status=RUN;
      else if (control packet==STOP) {
        if (status==STOP) continue;
        status=STOP;
        goto init_variable;
      } else process parameter update
    }
    [ synthesized C code for a dataflow module ]

  }
```

Figure 6: Pseudo code for a synthesized dataflow module with asynchronous interaction only

In figure 6, the C process uses non-blocking read at the "run" state and continues to execute synthesized codes until a control packet arrives. If a control packet arrives, the process reads the packet and processes it, which sets the scheduling status or changes the parameter value. But if the state is changed to "suspend" state, the process uses blocking read until a control packet arrives which changes the scheduling status to "run" state.

There are two other asynchronous signals: *stop* and *variable update*. The stop signal makes a dataflow module stop and change all internal states to initial values. And the variable update signal changes the target variable to a delivered value.

Figure 7 shows asynchronous interactions with a synchronous connection between FSM and synthesized dataflow modules. Every time a packet arrives at the C process, the process tests whether it is a data packet or the control packet from the packet header information. If it is a data packet, the process consumes the data and produce outputs if any. If a control packet is arrived, it performs the scheduling control or updates the target parameter with the control value. In this type of execu-

tion, it reads a packet from the backplane using blocking read operation, processes the packet, and blocks again at the next read operation.

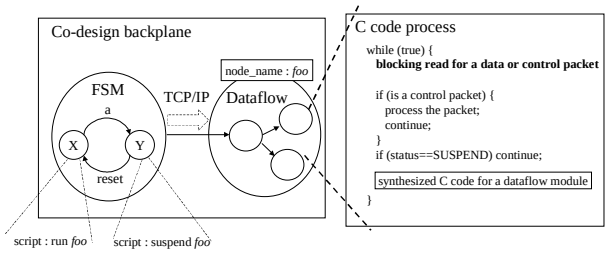

Figure 7: Synthesized dataflow module with both synchronous interaction and asynchronous interaction

4 Example : MPEG 1 Layer 3 Decoder

In this example, we implement an MPEG 1 Layer 3 audio player (MP3 player)[9] in which the decoding algorithm is depicted as a dataflow graph and synthesized into a C-code at compile-time (figure 8). The FSM control module resides in the "controlFSM" block and the TclScript block generates user control events to the FSM block. The MP3 decoder reads an MP3 file and decodes the encoded samples through a series of Huffman decoder, Dequantizer, and other function blocks. The FSM block has three operations. One is to control the execution of the MP3 decoder. Figure 8 shows that MP3 decoder has "start", "stop" and "suspend" execution states. Another is to control the "volume" value in the MP3 decoder asynchronously. Final operation is to display the current volume at the user interface.

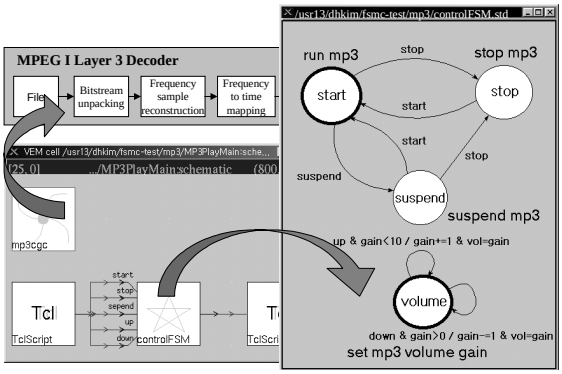

Figure 8: MPEG 1 Layer 3 Decoder example

An MP3 decoder is synthesized and compiled as a separate C process at compile-time, from the initial dataflow specification in the codesign framework. At runtime, the C process and the codesign framework communicate with each other via TCP/IP sockets. From this communication channel, the special control signal samples are delivered to the C process to suspend or resume the decoding. In this example, we play the MP3 file in the Sun Ultra1 machine.

5 Conclusion

This paper presents the proposed communication protocols between control modules and function modules. In the proposed technique, synchronous interaction acts like function calls and asynchronous interaction enables FSM to change the scheduling control and update parameters of dataflow modules, which provides the very flexible way to specify the system with both control modules and function modules. Through comparison with the related works and an example, the practicality is demonstrated.

References

[1] Chiodo, M.; Giusto, P.; Hsieh, H.; Jurecska, A.; Lavagno, L. Sangiovanni-Vincentelli, A.; *A formal methodology for hardware/software codesign of embedded systems.* IEEE Micro, August 1994.

[2] Kim, D.; Ha. S.; System level specification for multimedia applications. submitted to APCHDL, Fukuoka, Japan, Oct. 6-8, 1999

[3] Buck, J.; Ha, S.; Lee, E.; Messerschmitt, D.; *Ptolemy: a framework for simulating and prototyping heterogeneous systems*. International Journal of Computer Simulation, Vol. 4, pp. 155-182, 1994.

[4] Brunel, J-Y. et al.; *COSY: a methodology for system design based on reusable hardware & software IP's*, in J-Y. Roger (ed.), Technologies for the Information Society, pp. 709-716, 1998.

[5] Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A.; Politi, A.; Sherman, R.; Shtull-Trauring, A.; Trakhtenbrot, M.; *Statemate: a working environment for the development of complex reactive systems.* IEEE Trans. on Software Engineering, Vol. 16, No. 4, April 1990.

[6] Harel, D.; *Statecharts: a visual formalism for complex systems.* Sci. Comput. Program, Vol. 8, pp. 231-274, 1987.

[7] Sung, W.; Ha, S.; *Efficient and flexible cosimulation environment for DSP applications*. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Special Issue on VLSI Design and CAD algorithms, Vol.E81-A, No.12, pp. 2605-2611, December, 1998

[8] Cassandras, C.; *Discrete event systems, modeling and performance analysis.* Irwin, Homewood IL, 1993.

[9] Shlien, S.; *Guide to MPEG-1 Audio Standard*, IEEE Trans. on Broadcasting, Vol. 40, No. 4, pp. 206-218, December 1994.