Chapter 4. Introduction to the Models of Computation

Author: Soonhoi Ha

Block diagram specification enables us to reuse pre-defined blocks and to understand the overall structure of the system behavior in a hierarchical fashion. But, there are numerous ways of interpreting the block diagram unless we do not specify the execution rule. For example, see Figure 4-1 and guess what the meaning of this block diagram is.

![Figure 4-1 A simple block diagram](image)

One possible interpretation is that block C is executed as soon as it gets an input sample either from block A or block B. Another is that block C becomes executable only after it gets input samples from both block A and B. When block A is executed twice to produce two output samples to block C, either two samples are queued on the arc or the later sample overwrites the former sample.

Therefore, it is necessary to assign a specific execution rule to a block diagram for unambiguous interpretation by defining a “formal” model of computation. Therefore using formal models of computation is recently advocated because there are several benefits: (1) Static analysis of the specification allows one to verify some correctness measures of the functional specification. (2) The specification model can be refined to an implementation to ease the system validation by the principle of "correct-by-construction". (3) A formal specification model is not biased to any specific implementation method, so allows one to explore the wider design space. (4) It represents the system behavior unambiguously so that collaboration and design maintenance can be accomplished easily. In this chapter, we overview four basic models of computation used in PeaCE while each model will be explained in the following chapters in great details.

In PeaCE, a block diagram is assigned its domain and target. The **domain** of a block diagram implies the model of computation and its purpose. For example the CGC domain in PeaCE implies that the block diagram follows the SPDF model and its purpose is to generate C codes from the graph. The **target** of a block diagram indicates the target architecture or environment we want to run the graph. A domain usually has various targets associated. For example, we can generate different C codes from the same graph depending on whether the target architecture is the host machine for simulation, a prototype hardware with a single processor, a multiple processor SoC, and so on.
4.1 Dataflow Model: SPDF (Synchronous Piggybacked Dataflow Model)

In a dataflow model, a block is executable only if there exists a specified number of input samples on each input arc. The number of samples produced (or consumed) per block execution is called the output (or the input) sample rate of the block. In case all sample rates are statically determined and fixed at runtime, and can be any integer, the block diagram is called a synchronous dataflow (SDF) graph. By default, the sample rate of an input or output port is unity. And, there is assumed an unbounded FIFO queue on each arc.

Based on this model, examine the execution scenario of the block diagram in Figure 4-1. Block C is executable only after block A and block B are executed to produce at least one sample on each input arc of block C. It means that a block execution sequence of \{ABC\} is valid while \{ACB\} is not. When block A is executed twice in sequence, two samples are accumulated on arc AC.

Dataflow graph has become a successful representation for DSP algorithms since dataflow semantics is well matched with algorithmic function flow in DSP applications. You may think that an input port of a block represents an input argument for a function that corresponds to the body of the block. A function can be called only after all input arguments are provided. An output port corresponds to a return value of the function. Then, a block diagram with dataflow model looks like a function call graph. In fact, a dataflow program graph is used to specify a digital signal processing (DSP) task such as H.263 encoder, MP3 decoder, and so on.

Even though the SDF model and its related models are widely adopted for algorithm specification in numerous DSP design environments, it has a couple of serious limitations to represent multimedia applications: First, the SDF model cannot express conditional execution of a block. Second, the model cannot use the shared data structure between blocks and pointer operations so that the efficiency of the synthesized code is noticeably worse than the hand-optimized code in terms of memory requirements. Therefore, we make two extensions to the SDF model to overcome these problems, which are fractional rate dataflow (FRDF) and synchronous piggybacked dataflow (SPDF), which will be discussed in the next chapter. (@ SPDF)

In PeaCE, we distinguish two different domains associated with the SPDF model: CGC and VHDL. These domains generate a C code and a VHDL code from a SPDF graph respectively.

4.2 DE Model: DE (Discrete Event Model)

In a Discrete Event (DE) model, a sample is associated with a time stamp indicating when it is generated. And a block processes input events in the order of arrival times. In Figure 4-1, block C is executed as soon as it receives an input event from any input port. In the DE model there is a run-time scheduler that collects and sorts all events in a queue, and delivers the earliest event to the destination block. The scheduler manages the global clock. When a block generates an event with a smaller time stamp than the global clock, the scheduler signals a "causality error" and aborts the execution of the block diagram.

Unlike the SDF model, there is no guarantee that an output event is produced from an output port after the block is executed. In Figure 4-1, block A may never produce any event to block C though it is executed multiple times. Therefore, the execution sequence of blocks cannot be determined a priori before run time.
The DE model is widely used for dynamic system behavior as a function of time: for example, queuing systems, communication networks, and hardware simulations. The DE model in PeaCE is inherited from the Ptolemy classic. While the DE model is provided in PeaCE, note that it is not integrated in the HW/SW codesign flow.

Even though a block in the DE model is event-driven, or triggered by an event arrival, it can be modeled as time-driven if we attach a self-loop to the block. Through the self-loop, the block may trigger itself at the predetermined future time. See chapter 7 for more explanation.

4.3 FSM Model: fFSM (flexible Finite State Machine Model)

An FSM graph is a directed connected graph composed of states and transition arcs. A state describes a specific status of the system and a transition arc is associated with a condition that must be fulfilled to makes state transition from a source state to a destination state. Finite state machine (FSM) is the most popular model to describe the control module of a system. Even though FSM is simple to use, its unstructuredness and state explosion problem due to system concurrency and memory prohibit FSM model from practical representation of complex control behavior. So many extensions have been proposed to overcome these problems.

To describe complex control modules, the extensions should support various kinds of concurrency. Another desired feature is compositionality whether the complex module can be represented as a composition of simpler modules. Then, modules can be easily reused to construct a large system. Moreover, because of their complexity, it is desired to have a static analysis method to check ambiguous behavior. Finally for fast prototyping, automatic hardware (or software) synthesis is needed from the extended model. To meet these requirements, PeaCE devises a new extension of FSM model, called fFSM Model. It extends the expression capabilities by concurrency, hierarchy, and state variable. See chapter 8 for detailed explanation.

4.4 Task Model: BP(Backplane)

The SPDF model and fFSM model is used to describe the internal behavior of a computation and a control task respectively. We use another model, called Task Model, to describe the system behavior at the task level. Thus we are able to represent multi-tasking applications. In PeaCE, we can define a super block that encapsulates a block diagram with a different model of computation to make a hierarchical block diagram. At the top-level, the Task model represents a composition of multiple tasks: each task is modeled as a SPDF model or fFSM model. Since we use the Task model in this fashion, we call it a BP (Backplane) model.

The Backplane model defines additional features for inter-task interaction and task management: when a task is triggered to execution or suspended, and how tasks can communicate with each other, and so on. See chapter 9 for more explanation.