Constant Rate Dataflow Model with Intermediate Ports for Efficient Code Synthesis with Top-down Design and Dynamic Behavior

Hyunok Oh
ARM Inc.
1 Jenner, Irvine, CA 92618, USA
hyunok.oh@arm.com

Abstract

This paper extends the existing synchronous dataflow (SDF) model to provide dynamic behavior and top down design with compile time deadlock detection and bounded buffer memory.

We propose a new dataflow model called constant rate dataflow with intermediate ports (CRDF-IP) in which a component (or actor) can send and receive data to/from another actor through intermediate port during its execution. Since an actor can call another actor multiple times per execution, dynamic behaviors are easily specified without introducing run-time scheduler. Moreover, top-down design can be achieved by extracting a sub-actor from the top actor.

This paper has proved that deadlock detection and buffer size computation can be performed at compile in CRDF-IP model. The proposed model has been implemented in a system level design platform in which H.263 video encoding algorithm is specified.

1. Introduction

Automatic code synthesis is emerging topic to provide fast development of a system in order to time-to-market and manage a system complexity by reusing the pre-designed code. Restricted rule based specification makes easy the analysis of the specified system and generates efficient codes close to the hand-optimized code. Dataflow based specification is attractive model of computation in embedded system design since DSP algorithms and multimedia algorithms can be naturally specified and synthesized efficiently.

Generally, dataflow model is specified with actors (or nodes) and edges. An actor with ports represents a function that transforms data and an edge represents the connection between actors through ports. The model can be specified in hierarchical manner so that an actor can contains another dataflow graph. The atomic actor which does not contain any dataflow graph is written in a high-level language such as C and VHDL. The code synthesis from dataflow specification, therefore, stitches atomic actors with proper ordering called scheduling and generates the high-level language code from the actors with wrapper code which manages data transferring between actors and executes the actor codes. Synchronous dataflow (SDF) [4] model has been used widely in which fixed number of data is produced and consumed per actor execution. Figure 1 (a) represents a simple SDF graph and its generated code.

Even though the dataflow specification is promising approach for rapid system development and system complexity management by reusing pre-developed actors, there are difficulties in adopting dataflow model in system development process.

First, dynamic behavior is not easy to specify since dataflow model does not represent control. Many models such boolean dataflow (BDF) [1] and dynamic dataflow (DDF) [3] have been proposed to provide if-then-else and for-loop structures. In those models, the specification of dynamic behavior should meet specific rules and actors for dynamic behavior are specially handled by scheduling and code generation algorithms. The special handling of dynamic behavior, generally, requires deep hierarchical representation or complex connection between the actors for dynamic behavior. The synthesized code is not efficient since it may contain run-time scheduler and/or buffer manager which degrade performance. Figure 1 (b) and (c) represent if-statement and do-while-statement in BDF and DDF model respectively.

Second, top down design of a system is hardly supported in most dataflow models. When designers de-
develop a system in top-down approach, they create a few actors to build a system based on which algorithms should be optimized for the system. When the system becomes matured, the actors are decomposed into more actors to improve performance by optimizing algorithms in actors or using hardware accelerators for actors. Therefore, an actor in a system should be able to become decomposed into multiple actors step by step during the development of the system.

Consider figure 2(a) representing H.263 encoding algorithm which is a video encoding standard. Since motion estimation algorithm dominates the encoding time, it needs to be extracted from the system as shown in figure 2(b). During optimizing motion estimation actor, we want to extract sum of absolute different(SAD) actor from ME actor since we can increase performance by optimizing SAD using some special instructions on a target hardware architecture.

2. Constant Rate Dataflow Model with Intermediate Ports

We propose constant rate dataflow with intermediate ports model (CRDF-IP) to allow dynamic behavioral specification and top-down design with static analysis of deadlock checking and buffer memory computation at compile time.

The CRDF-IP model has fixed rate or constant rate like SDF in which rates are not changing during execution. CRDF-IP model has not only normal ports that are equivalent to ports in SDF model but also intermediate ports which have not been available in the previous dataflow models. Through the intermediate ports, an actor can produce and consume data during its execution while data are consumed at the beginning and produced at the end of the execution through normal ports. The intermediate ports provide dynamic behavior and top-down design approach.

Since an actor can produce and consume data arbitrary number of times through the intermediate port, dynamic behaviors like if-then-else, for-loop and while-loop can be easily specified as shown in figure 3. Actor If in figure 3(a) contains two groups of intermediate ports: "true" and "false" groups. If input port of ctrl receives true value then the actor produces and consumes data through "true" group intermediate ports; otherwise it does through "false" group intermediate ports. Note that If actor is different from that in BDF model since it is not specially handled by scheduling and code generation algorithms. Similarly, For and DoWhile actors can be easily constructed by a user using intermediate ports. Moreover, a user can build a single actor containing both if and for simultaneously. The generation codes for If, For and DoWhile actors are obtained from the specification of the actors.

The other benefit from intermediate ports is to support top-down design or progressive algorithm refinement. Since the intermediate port allows an actor to call another actor during its execution, a designer can extract an algorithm from an actor and build a callee actor without decomposing the caller actor into two actors which are fired before and after a callee actor executes. Hence, one large actor is constructed to represent whole algorithm initially and then some algorithms become independent actors from the large actor progressively. Designers refine the callee actors to improve algorithm performance where the actors are normally dominant in the algorithm.

3. Static Analysis

In the existing dataflow models like SDF, a valid schedule of actor execution can be constructed at compile time if the graph is consistent [4] which indicates that the repetition of execution according to the schedule does not overflow and underflow buffers. In SDF
connected with actor consistent.

bounded memory if a CRDF-IP graph is correct and consistent.

\[ \forall g \in D(G) \text{, } g \text{ is consistent where } D(G) \text{ indicates decomposed graphs from graph } G. \]

Definition 2. \( G \) is correct iff every intermediate port connected with actor \( a \in g \in D(G) \) has a single group name.

Theorem 1. A valid schedule can be constructed with bounded memory if a CRDF-IP graph is correct and consistent.

Proof. Assume that the repetition of execution according to a valid schedule requires unbounded memory on edge \( e \). Since the graph is correct, a decomposed graph is called atomically. Therefore we can uniquely determine graph \( g \in D(G) \) where \( g \) contains \( e \). \( g \) is consistent by definition 1. By [4], a valid schedule for a consistent graph does not introduce unbounded memory. Therefore there is no edge introducing unbounded memory.

4. Scheduling and Code Generation

Since a CRDF-IP graph can be decomposed into independent SDF graphs if it is correct, scheduling algorithms developed for SDF models are also applicable for CRDF-IP. However, the decomposed graph should be revised before the existing scheduling algorithms are applied since the decomposed graph includes cyclic loop.

Consider figure 4(a) where actor B has intermediate ports connected with actor D and E. The graph can be decomposed as shown in figure 4(b). No actor is executable in decomposed graph of "B2, D, E" since actors construct a cycle and no input port has enough data while actor B2 (or B) produces data first and then it consumes data in CRDF-IP semantics. To eliminate the difference between the original CRDF-IP graph and decomposed graphs, imaginary initial delays are appended for input port of B2 where those delays are not used in code generation but only for scheduling. The delay size is equivalent to the consuming rate of B2 since if there are consuming rate number of data on the port then actor B2 can execute. Therefore, figure 4(c) which contains initial delays is constructed for scheduling purpose. Finally, we can obtain two schedules for the decomposed graphs, which are "A B1 C" and "B2 D E".
for intermediate ports in an actor has two informations: group name, and whether it produces or consumes data. In our implementation, we propose two special keywords: $\text{Send}(\text{groupName})$ and $\text{Receive}(\text{groupName})$. $\text{Send}(\text{groupName})$ keyword notifies that the actor produces data while $\text{Receive}(\text{groupName})$ informs data for input are available.

Figure 5 represents a CRDF-IP code generation example where $bo$ and $bi$ are intermediate ports. First, codes are generated according to the schedule of "ABC" for the top graph. When the reserved keywords of $\text{Send}(\text{"x"})$ and $\text{Receive}(\text{"x"})$ are met during generation of actor B, code generation for "BDE" is called in which actor B is removed from the code generation. Hence, we can obtain the final code as shown in figure 5(c).

Figure 5. (a) CRDF-IP graph, (b) specification of B in C and (c) generated code in C

5. Experiment

We have implemented the proposed dataflow model in system level design platform of PeaCE (Ptolemy extension as a Codesign Environment) [2] which includes system design process from specification to code synthesis in software and hardware. Figure 6(a) denotes the specification of H.263 encoding algorithm in CRDF-IP model while figure 6(b) presents it in SDF model. The code size and performance of generated codes are quite similar. For instance, the code size is 16.7KB in CRDF-IP while it is 17.5KB in SDF using ARM compiler.

6. Conclusion

In this paper, we propose a new dataflow model called constant rate dataflow with intermediate ports which extends an existent dataflow model like SDF to allow dynamic behavior and top-down design. The proposed model can specify intermediate ports as well as normal ports which is equivalent to ports in SDF model while intermediate port can send and receive data many times during actor execution. The intermediate port easily allows dynamic behavior specifications such as if-then-else, for-loop and while-statements. Moreover, since the intermediate port provides function call style representation in dataflow model, top down design and progressive algorithm improvement are easily achieved. This paper proves that CRDF-IP graphs can be statically analyzed to find valid schedules with bounded memory. Since a CRDF-IP graph can be decomposed into independent SDF graphs, existing scheduling and code generation algorithms can be applied. We have implemented the propose dataflow into system design environment, designed H.263 encoding algorithm using CRDF-IP model and generated the C code.

References