Dataflow analysis for real-time embedded multiprocessor system design

Chapter 4

DATAFLOW ANALYSIS FOR REAL-TIME EMBEDDED MULTIPROCESSOR SYSTEM DESIGN

Marco Bekooij[1], Rob Hoes[2], Orlando Moreira[1], Peter Poplavko[2], Milan Pastrnak[2], Bart Mesman[1,2], Jan David Mol[3], Sander Stuijk[2], Valentin Gheorghita[2], and Jef van Meerbergen[1,2]

[1] *Philips Research Laboratories, Eindhoven, The Netherlands*

[2] *Eindhoven University of Technology, Eindhoven, The Netherlands*

[3] *Delft University of Technology, Delft, The Netherlands*

Marco.Bekooij@philips.com

Abstract Dataflow analysis techniques are key to reduce the number of design iterations and shorten the design time of real-time embedded network based multiprocessor systems that process data streams. With these analysis techniques the worst-case end-to-end temporal behavior of hard real-time applications can be derived from a dataflow model in which computation, communication and arbitration is modeled. For soft real-time applications these static dataflow analysis techniques are combined with simulation of the dataflow model to test statistical assertions about their temporal behavior. The simulation results in combination with properties of the dataflow model are used to derive the sensitivity of design parameters and to estimate parameters like the capacity of data buffers.

Keywords: real-time, dataflow analysis, multiprocessor system, predictable design, system-on-chip

1. INTRODUCTION

Consumers typically have high expectation about the quality delivered by multimedia devices like DVD-players, audio, and television sets. These devices process data streams and are often built using (weakly) programmable embedded multiprocessor systems for performance, cost, and power-efficiency reasons. The design and programming of these real-time multiprocessor systems should be such that the real-time constraints are met, and the desired

P. van der Stok (ed.),
Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices, 81-108.

audio and video quality is delivered. These multiprocessor systems should be suitable for the simultaneous execution of audio and channel decoders as well as video decoders. The audio and channel decoders have hard real-time constraints because a miss of a deadline results in a click in the audio or loss of data which is unacceptable for the end-user. The video decoders have soft real-time constraints because if a deadline is missed then the video quality is reduced which is not appreciated by the end user but is to some extent acceptable.

The current design practice is that timing constraints of hard real-time applications are guaranteed by making use of analytical techniques while the (temporal) behavior of soft real-time applications is measured. As will be explained in the next paragraphs, these measurements do not make all the characteristics of soft real-time applications explicit which are usefull during the design process. Therefore we are concerned in this chapter with the use of dataflow models for the validation of the (temporal) behavior of applications with *soft real-time* constraints. These dataflow models are also key to derive a proper dimensioning of the multiprocessors system and to derive a proper mapping of the application onto the multiprocessors system. We claim that the use of these dataflow models reduces the number of design iterations and shortens the design time. Also our network based embedded multiprocessor system is presented. This system is suitable for the derivation of the temporal behavior of the application with dataflow models.

The applications executed on our multiprocessor system consist of jobs (see Figure 4-1). A job is an entity that processes a data stream. It is started and stopped by the user. The hard real-time jobs are indicated in this figure by dotted circles while the soft real-time jobs are indicated by dashed circles. A job is described by a dataflow graph. Such a dataflow graph contains actors that represent software tasks, or computations performed by a hardware component. Actors are started after sufficient input data and output space is available, such that they can finish their execution without having to wait for additional input data or output space. The edges denote communication of data between actors via First-In-First-Out (FIFO) buffers.

For soft real-time jobs such as video decoders, a tradeoff is typically made between the amount of resources that are made available and the deadline miss rate. Less system resources result in less hardware and a reduction of the hardware cost, but also result in a higher deadline miss rate and a reduced quality of experience for the end user. It is therefore an objective of the system designer to dimension and program the multiprocessor system in such a way that the quality is minimally compromised for a given resource budget.

The current design practice of systems that execute soft real-time jobs can be schematically depicted with a Y-chart (Kock, Essink, Smits, Wolf, Brunel, Kruijtzer, Lieverse and Vissers, 2000), as is shown in Figure 4-2. The dashed arrows in this figure denote design iterations. During an iteration a multipro-

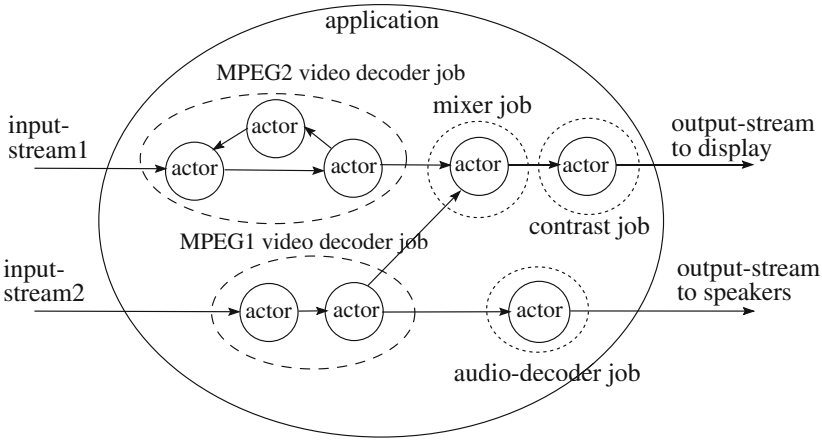

Figure 4-1. An application that consists of jobs. Jobs are started and stopped by the user. Jobs consist of actors that communicate via FIFOs. Hard real-time jobs are indicated by dotted circles while soft real-time jobs are indicated by dashed circles.

cessor instance is (re-)defined, programmed, and evaluated by means of simulating the target application in a cycle true simulator. From the simulation results, the system designer tries to derive clues on how he can improve the system or its programming such that all design constraints are satisfied, which is indicated by the light bulbs in the Y-chart figure. The current design practice is that the design constraints are verified after simulation but to a large extent ignored during mapping.

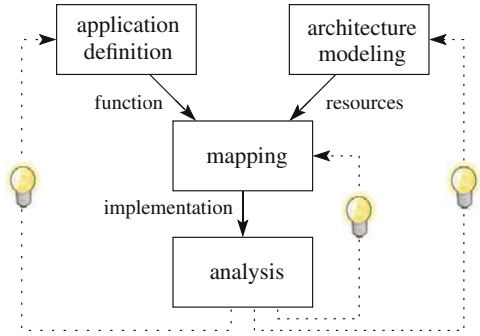

Figure 4-2. Y-chart programming paradigm.

Such a simulation based design process is cumbersome for modern applications and architectures due to the uncertainty in the amount of resources

demanded by the application at run-time and the uncertainty in the amount of resources supplied by the hardware. The resource demand fluctuates during execution because the amount of computation and communication performed by the application often depends on the content of the input stream. For example, the execution time of the actors depends usually on the values of the input data. This can also be the case for the amount of data communicated between the actors. Also, the amount of resources supplied by the hardware fluctuates due to arbitration of shared resources in the system. The term arb itration" is used in this chapter for the local scheduling of actors on processors, as well as for the policy used to resolve at run-time simultaneous requests for a shared resource, such as for example a communication bus or a memory port.

Another reason why this simulation based design process has become cumbersome is that the complexity of system-on-chip designs has grown much faster than the increase in speed of the simulators. This has resulted in slow design iterations in which usually only a small fraction of the system can be evaluated. It should also be noted that it can be very difficult to find performance critical corner cases in the design and generate the proper input stimuli to observe the system's behavior for these cases.

Another disadvantage of a simulation based design process is that it can be difficult to draw conclusions from the simulation results how to adapt the multiprocessor system's hardware or its programming. It can be difficult to draw conclusions because these multiprocessor systems can exhibit a highly non-linear behavior.

Finally, we would like to mention that it is difficult to reproduce the same temporal behavior with such a simulation based design process. The reason is that the initial state of the arbiters (e.g. Time Division Multiple Access (TDMA) arbiters) in the system is unknown at the moment that the job is started. Therefore, the order in which access to a shared resource will be granted by an arbiter is not known at compile time. A different order in which requests are granted can result in a completely different temporal behavior of a job in the case that the same job is started at a different point in time. This will make it for example impossible to reproduce the same temporal behavior with an (Field Programmable Gate Array (FPGA)) prototype of the system, which is currently often used to speed up the performance evaluation and debugging process.

In this chapter, we propose a multiprocessor system in which the uncertainty in the resource supply is bounded by enforcing resource budgets. A resource budget is for example a guaranteed amount of time to use a resource such as a bus or processor during a predefined period. These enforced resource budgets will make it possible to share resources, such as a port to background memory, between hard real-time and soft real-time jobs. These budgets also drastically reduce the effort to verify the temporal behavior of soft real-time jobs. The

reason is that given enforced resource budgets, the temporal behavior of one job cannot affect the temporal behavior of another job. This gives a job the illusion that it executes on its own private hardware, so it can be evaluated in isolation.

Given that resource budgets are enforced and guaranteed, then dataflow models and their corresponding analysis techniques can be applied to guarantee that hard real-time jobs will meet their deadlines. However these techniques are not directly applicable for soft real-time jobs because they require that a schedule can be derived offline. Such a schedule cannot be constructed for soft real-time jobs because the amount of resources that is provided for soft real-time applications is typically *less* than the worst-case amount of resources that are needed to meet all deadlines.

In this chapter we advocate the use of a mix of simulation and model based analysis techniques for the derivation of the temporal behavior of the soft real-time jobs. We show that dataflow models can be applied by demonstrating that if resource budgets are enforced that then the effect on the temporal behavior of run-time arbitration can be modeled in a dataflow model. These dataflow models can be used for soft real-time jobs to derive *conservative* arrival times of the data in the system by simulation of this dataflow model. During simulation the response times of the actors are used instead of the worst-case response times. The response time of an actor depends on the value of the input data of the actor. The arrival times of the data observed during simulation is *conservative* because data will not arrive earlier in the simulator than in the real system. There is no need to derive a schedule in advance because the execution order of actors is determined at run-time by the local schedulers/arbiters. The same dataflow models can be *analyzed* at compile-time to derive estimates of the effects on the throughput and latency of a job when a resource budget is adapted by the designer at compile time. An example of a resource budget is the capacity of a buffer.

2. RELATED WORK

In this work, dataflow models are used to derive the end-to-end temporal behavior of jobs. The focus is on synchronous dataflow (SDF) models (Lee and Messerschmitt, 1987), because it is currently the most popular and widely studied dataflow model for streaming applications with well defined semantics.

A similarity between SDF models and Kahn process networks (Kahn, 1974) is that they can be used to describe streaming applications. However SDF models are suitable for static analysis while Kahn process networks are unsuitable. Kahn process networks are unsuitable for static analysis because a Kahn process network is Turing complete. Therefore, questions of termination and bounded buffering are undecidable. That is, no finite time algorithm can

decide these questions for all Kahn process networks. This is illustrated with the Kahn process network example in Figure 4-3. In this example we assume that the behavior of process P1 depends on the values of the input data and is therefore unknown at compile time. We assumed also that the values of the input data are at run-time such that this process P1 will write one data word in FIFO1 after it has written 11 data words in FIFO2. We also assume in this example that process P2 reads first one data word from FIFO1 before it reads data from FIFO2. Deadlock of this process network occurs because process P1 cannot finish its writing of data in FIFO2 because the capacity of FIFO2 in the implementation is only 10 data words. Therefore, processes P1 will never be able to write data in FIFO1 such that process P2 can first read data farom FIFO1 and then from FIFO2. It should be noted that FIFOs with a finite capacity should be represented in a Kahn process network as two FIFOs with an infinite capacity. The data producing process stores tokens filled with data in one FIFO while the data consuming process stores tokens which indicate space in the other FIFO.

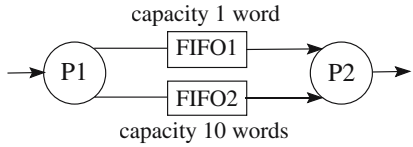

Figure 4-3. Example of Kahn process network which deadlocks due to insufficient FIFO capacity.

Another reason why Kahn process networks are unsuitable for static analysis is that a Kahn process blocks after it did a read attempt on an empty FIFO. A Kahn process that blocks must be preempted such that other processes on the same processor can continue their execution and produce the required input data. The number of times that a process blocks, depends on the run-time schedule and can strongly fluctuate. Therefore it is usually not possible to derive a tight bound on the preemption overhead at compile time. However a tight bound on the overhead due to preemption can be derived for SDF actors. The reason is that an SDF actor does not start its execution before all input data is present to finish its execution. Therefore SDF actors never block during their execution.

The SDF graphs are used in this chapter as a short hand notation of event graphs which are a special case of Petri nets (Petri, 1962). The temporal behavior of event graphs can be derived with MaxPlus Linear System Theory (Bacelli, Cohen, Olsder and Quadrat, 1992). SDF models are in (Sriram and Bhattacharyya, 2000) applied for hard real-time jobs that do not share resources with other jobs. The execution order of actors on the same processor

is derived from an offline computed schedule. Similar techniques are applied in (Poplavko, Basten, Bekooij, Meerbergen and Mesman, 2003) for soft real-time jobs. Good results could only be obtained by taking measures to limit the difference between the typical and the worst-case response times of the actors. The reason for these good results is that if the difference in response time is small, then the offline computed execution order is close to the optimal execution order. In this chapter we assume that the processors support preemption and that the execution order of the actors is determined at run-time. This makes it possible to cope with large variations in the execution time of the actors, and will allow sharing of resources by actors of different jobs.

In this chapter we advocate the use of a mix of simulation and model based analysis techniques for the derivation of the temporal behavior of the soft real-time jobs. The analysis of the temporal behavior of soft real-time jobs is different from the analysis of hard real-time jobs. The objective of the analysis for hard real-time jobs is to derive the worst-case temporal behavior of the system, while for soft real-time jobs the objective of the analysis is to derive the typical temporal behavior. The typical temporal behavior of jobs depends on the values of the input data which are unknown at compile time. Therefore, purely model based analysis techniques for hard real-time jobs, such as the techniques in (Kopetz, 1997; Pop, Eles and Peng, 2002; Richter, Jersak and Ernst, 2003), are not directly applicable for the analysis of soft real-time jobs. The reason is that the actual values of the input data can be ignored during analysis of hard real-time jobs because the objective is to derive the worst-case temporal behavior for any possible input data stream. For soft real-time jobs the values of the input data cannot be ignored during analysis because the objective is to derive the typical temporal behavior for a representative input stimuli set. The use of probabilistic models, such as Markovian and Poisson models, for the derivation of the typical temporal behavior of soft real-time jobs is either too simple to characterize the important properties of the source and the system, or too complex for tractable analysis (Zhang, 1995; Sriram and Bhattacharyya, 2000). Therefore, simulation is used by us to estimate parameters such as the execution times of the actors and to test statistical assertions about the temporal behavior of a job that is executed on the system, in a similar way as done in (Hee, 1994).

The concept of reservation based resource allocation has been introduced by the real-time community in order to eliminate interference between the software tasks of soft real-time multimedia jobs that are executed on a single processor system. The enforcement of resource budgets is a service provided by the operating system kernel (Rajkumar, Juwa, Moleno and Oikawa, 1998). The size of the resource budget is determined during a (re)negotiation phase between the job and the operating system. In this work, we address multiprocessor systems in which the resource budgets enforcement is not centralized

but distributed. Resource budgets are reserved to eliminate interference between jobs such that it is possible to share resources between hard real-time and soft real-time jobs, as well as to obtain a so-called monotonic system (see Section 4). An important property of a monotonic system is that an increase of a resource budget of a job cannot result in a reduction of the throughput of this job.

3. OUTLINE OF THIS CHAPTER

The organization of this chapter is as follows. The properties of the synchronous dataflow (SDF) model are recapitulated in Section 4. Then in Section 5 a multiprocessor architecture is presented that is suitable for the derivation of the temporal behavior of jobs with an SDF model. It is shown in Section 6 that the effects on the temporal behavior of a job, due to TDMA arbitration, can be expressed in the SDF model. By simulating this SDF model, conservative and accurate arrival times of tokens can be derived. The same SDF model is analyzed in Section 7 in order to derive at compile time the sensitivity for variations in the execution time of actors on the throughput of the system. We show that adaptation of the capacities of the FIFO buffers can reduce the sensitivity for fluctuations in the execution times of the actors on the end-to-end temporal behavior of a job. To obtain tight bounds on the arrival times of data it may be necessary to make the conditional execution of actors explicit in the dataflow model. Section 8 introduces conditional constructs in the dataflow model that guarantee mutual exclusive execution of actors. The dataflow graph that is obtained is an analyzable version of a Boolean Data Flow (BDF) graph (Buck, 1993). It is shown that these BDF graphs can be analyzed with the SDF analysis discussed in Section 4. These conditional constructs are applied in Section 9 to make explicit that different actors are executed during I-frame and P-frame decoding in an H263 video decoder. Finally, we state the conclusions in Section 10.

4. DATAFLOW ANALYSIS

In this section we define the SDF model and recapitulate its properties. This SDF model is used in successive sections for the derivation of the temporal behavior of jobs that are executed on multiprocessor systems with similar characteristics as the multiprocessor system that is presented in Section 5.

Before the properties of an SDF model are stated, we first define an SDF graph as follows:

Definition 1 (Synchronous Data Flow Graph.) *The tuple* (V, E, d, P, O, I) *defines a Synchronous Data Flow (SDF) graph, where*

- *V is the set of nodes (actors),*
- *$E \subseteq V \times V$ is the set of directed edges,*
- *$d : E \rightarrow \mathbb{N}$ is a function describing the number of initial tokens on an edge $(u, v) \in E$,*
- *$P : V \rightarrow \mathbb{R}^+$ is a function describing the worst-case response time of actor $v \in V$,*
- *$O : E \rightarrow \mathbb{N}$ is a function describing the number of tokens produced on edge $(u, v) \in E$ by actor u for each execution,*
- *$I : E \rightarrow \mathbb{N}$ is a function describing the number of tokens consumed from edge $(u, v) \in E$ by actor v for each execution.*

An arbitrary SDF graph is depicted in Figure 4-4. The nodes in an SDF graph are called actors. Actors have a well defined input/output behavior and a worst-case response time. Actors produce and consume tokens. The edges represent dependencies. A token is a container in which a fixed amount of data can be stored and is depicted in Figure 4-4 as a black dot. If more than one token is (initially) present on an edge then the number of tokens (d) is specified next to the dot. The tokens are consumed in the same order as they are produced. However random access of the data inside a token is allowed. Each actor in Figure 4-4 is annotated with its worst-case response time. An actor is enabled after a predefined number of tokens is available on every input of the actor. An actor can fire (starts its execution) after it is enabled. The number of tokens that must be available is specified next to the head of the data edges. The specified number of tokens is consumed from the input edges of the actor before the execution of an actor finishes, that is, within the response time of the actor. The number at the tail of an edge denotes the number of tokens an actor produces before the execution of the actor finishes. A self-edge of an actor is used to model that the previous execution must be finished before the next execution can start. This self edge is given one initial token such that the next execution cannot start before the previous execution of the actor is finished.

An SDF graph can be transformed into a Homogeneous Synchronous Data Flow (HSDF) graph (see Figure 4-5) on which we perform the analysis. An algorithm that transforms any SDF graph into an HSDF graph is described in (Sriram and Bhattacharyya, 2000). An HSDF graph is a special case of an SDF graph, in which the execution of an actor results in the consumption of one token from every incoming edge of the actor and the production of one token on every outgoing edge of the actor.

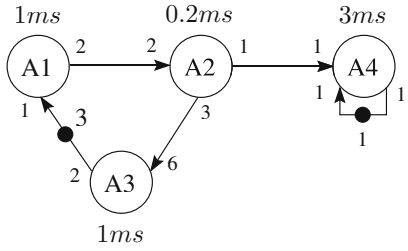

Figure 4-4. A Synchronous Data Flow (SDF) graph example.

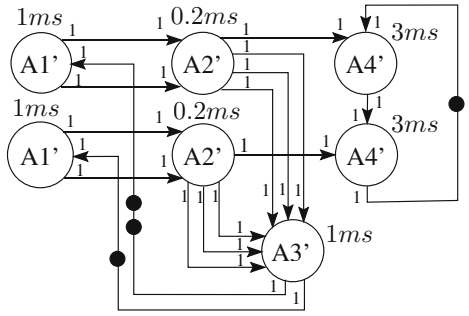

Figure 4-5. The Homogenous Synchronous Data Flow (HSDF) graph obtained after transformation of the SDF in Figure 4-4.

An HSDF graph can be executed in a self-timed manner, which is defined as a sequence of firings of HSDF actors in which the actors start immediately when there is at least one token on each input of the actor. In the case that the HSDF graph is a strongly connected graph and a FIFO ordering for the tokens is maintained between executions of the actors, then the self-timed execution of the HSDF graph has some important properties. A FIFO ordering is maintained if the completion events of firings of a specific actor occurs in the same order as the corresponding start-events. This is the case if an actor has a constant response time or belongs to a cycle in the HSDF graph with only one token. In (Bacelli et al., 1992) are the properties of the self-timed execution of such HSDF graphs derived with MaxPlus algebra.

First of all, the most important property of the self-timed execution of an HSDF graph is, that it is deadlock-free if there is on every cycle in the HSDF graph at least one initial token. Secondly, the execution of the HSDF graph (and also an SDF graph) is *monotonic*, i.e. decreasing actor response times result in non-increasing actor start times. The reason is that an earlier arrival time of a token cannot result in a later start of the actor that consumes this token.

Third, an HSDF graph will always enter a periodic regime. More precisely, a $K \in \mathbb{N}$, an $N \in \mathbb{N}$ and a $\lambda \in \mathbb{R}$, such that for all $v \in V$, $k > K$ the start time $s(v, k + N)$ of actor v in iteration $k + N$ is described by:

$$s(v, k + N) = s(v, k) + \lambda \cdot N \tag{4-1}$$

Equation 4-1 states that the execution enters a periodic regime after K executions of an actor in the HSDF graph. The time one period spans is $\lambda \cdot N$. The number of firings of an actor v in one period is denoted by N. Thus, λ is equal to the inverse of the average throughput measured over one period.

The Maximum Cycle Mean (MCM) (Sriram and Bhattacharyya, 2000) of an HSDF, which is equal to λ, is given by (4-2). The MCM of an HSDF graph is also called in literature the maximal cost to time ratio (Lawler, 1976). The Cycle Mean (CM) of a simple cycle c in the HSDF graph G is given by (4-3). In this equation denotes $d(c)$ the number of tokens on the edges in a cycle c. The Worst Case Response Time (WCRT) of actor v is denoted by WCRT(v). The MCM of an HSDF graph can be derived with a pseudopolynomial algorithm (Cochet-Terrasson, Cohen, Gaubert, McGettrick and Quadrat, 1998) with complexity $O(m|E|)$ with m the product of the out-degrees of all nodes.

$$\text{MCM}(G) = \max_{c \in C_G} \text{CM}(c) \tag{4-2}$$

$$\text{CM}(c) = \sum_{v \ on \ c} \text{WCRT}(v)/d(c) \tag{4-3}$$

The worst-case start-times of the actors during the transition state as well as the steady state can be observed during self-timed execution of an SDF graph in a simulator. During this simulation, all actors must have a response time equal to their worst-case response time. The start-times observed during this simulation are equal to the worst-case start times of the actors due to the monotonicity of the SDF graph. From (4-1) it follows that a periodic regime will be entered and therefore simulation can be stopped after the first period of the periodic regime. The SDF will enter a periodic regime because the HSDF graph that is obtained after transformation will enter a periodic regime. The SDF enters a periodic regime because the i-th start of an actor A1 in the SDF graph is as soon as all input tokens have arrived for this actor. All input tokens have arrived as soon as there is one token on each input of an actor A1' in the HSDF such that an actor A1' is started for the i-th time.

Actors in an SDF graph produce their output tokens exactly the WCRT after the actor is started. The input tokens are consumed and removed from the

input exactly the WCRT after the actor is started. Code segments in the implementation can be represented by an SDF actor in the model. Code segments produce the output tokens and consume the input tokens within the WCRT of the actor. The arrival times of tokens during selftimed execution of the SDF graph is not earlier then in the implementation due to the monotonic behavior of a selftimed executed SDF graph. Therefore an upper bound on the arrival time of tokens is observed during selftimed execution of the SDF graph.

We refer in this chapter to a code segment as an actor in the implementation. The actors in the implementation have a response time as well as an Execution Time (ET). The ET of an actor in the implementation is defined as the interval of time it takes to execute the corresponding code segment on a processor without that its execution is preempted. The execution time depends often on the values of the input data. The Worst Case Execution Time (WCET) is an upper bound on the execution time of an actor in the implementation and is derived with static program analysis techniques (Li and Malik, 1999).

It should be noted that the token arrival times during selftimed execution in the SDF simulator remain conservative if the Response Times (RTs) of the actors are used instead of the worst-case response times of the actors. The RT of an actor is an upperbound on the time interval between the point in time that the actor is enabled and that the point in time that the actor finishes its execution. The response time of an actor can depend on the values of the input data that are consumed during that execution. The token arrival times during selftimed execution in the SDF simulator are conservative because the selftimed execution of the SDF graph is monotonic. The use of the RTs of actors allows us to derive an upperbound on the token arrival times for soft real-time jobs given a specific input stimuli set for that job.

In Section 7 we will use Predicted Response Times (PRTs) of the actors to derive at compile time the resource budgets of soft real-time jobs. The PRT of an actor is the measured average response time of this actor on a processor given a specific input stimuli set for that actor. Given the PRT of an actor we will predict the resource budget for a soft real-time job.

5. MULTIPROCESSOR SYSTEM TEMPLATE

This section describes a network based multiprocessor system. This multiprocessor system is defined in such a way that a tight bound on the temporal behavior of jobs can be derived at compile time with dataflow analysis techniques. These analysis techniques are described in the previous section and are extended in Section 6.

Figure 4-6 shows the architecture template of this multiprocessor system. The processors in this template are, together with their local data memory, connected to the Network Interface (NI) of a packet switched Network on Chip

(NoC) (Rijpkema, Goossens, Rădulescu, Dielissen, Meerbergen, Wielage and Waterlander, 2003). The transfer of data between a local memory and a network interface is performed by a Communication Assist (CA). A processor together with its local instruction and data memory, communication assist, and network interfaces is grouped into a leaf. The leafs are connected to the routers of our network. Network links connect the routers in the desired network topology.

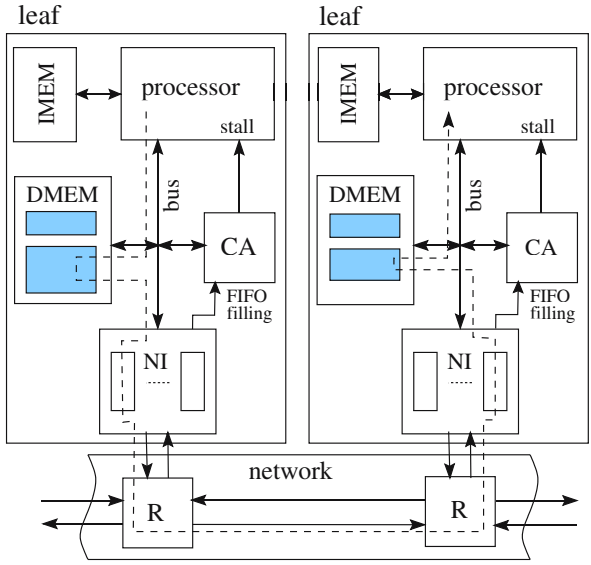

Figure 4-6. Multiprocessor template.

A processor in a leaf has a separate Instruction Memory (I-mem) and Data Memory (D-mem), such that instruction fetches and data load and store operations do not cause contention on the same memory port. An unbounded range of memory access time variations due to contention on a memory port is intolerable, as this would result in an unpredictable execution time of the instructions of the processor. This is also the reason why we consider in this paper only the case that the processors access only their local data memory. Given a 1 cycle access time of a local memory there is no reason to introduce caches.

Communication between actors on different processors takes place via a virtual point to point connection of the NoC. The result of the producing actor is written in a logical FIFO in the local memory of the processor. Such a logical FIFO can be implemented with the C-HEAP (Gangwal, Nieuwland and Lippens, 2001) communication protocol, without use of semaphores. The

communication assist polls at regular intervals whether there is data in this FIFO. As soon as the CA detects that there is data available, it copies the data into a FIFO of the NI. There is one private FIFO per connection in the NI. Subsequently the data is transported over the network to the NI of the receiving processor. As soon as the data arrives in this NI, it is copied by the CA into a logical FIFO in the memory of the processor that executes the consuming actor. The data is read from this FIFO after the consuming actor has detected that there is sufficient data in the FIFO. Flow control between the producing and consuming actor is achieved by making sure that data is not written into a FIFO before it is checked that there is space available in this FIFO.

Data is stored in the local memory of the processor before it is transferred across the network. This is done for a number of reasons. First of all, the bandwidth of a connection is set by configuring tables in the network for a longer period of time. The bandwidth reserved for a connection will typically be less than the peak data rate generated by the producing actor. Therefore a buffer is needed between the processor and the network to average out the peak data rate such that the bandwidth provided by the network is well utilized. Also the memory in the leaf which receives the data can typically not accommodate the peak bandwidth because another processor can access this memory at the same time. Another reason is that without such a buffer the execution time and the response time of the actors is dependent on the allocated bandwidth in the network. This dependency will complicate the analysis of the temporal behavior.

The size of the buffer in which data is stored before it is transferred across the network is significant, given the assumption that the actors produce large chunks of data at large intervals. On the other hand, the network will transfer very small chunks of data (3 words of 32 bits) at very small intervals ($\sim$2 ns). Given that large memories are inherently slow, it is desirable to split the large logical FIFO between the processor and the network, in a small ($\sim$32 word) dedicated FIFO per connection in the network interface, and a large logical FIFO in the local memory of the processor. The task of the CA is to copy the data between FIFOs in the NI and FIFOs in local memory of the processor.

The CA is also responsible for the arbitration of the data memory bus. The applied arbitration scheme is such that a low worst-case latency of memory store and load operations is obtained and that a minimal throughput and maximal latency per connection is guaranteed. A more detailed description can be found in (Bekooij, Moreira, Poplavko, Mesman, Pastrnak and van Meerbergen, 2004).

In the proposed architecture, the communication between actors that run on different processors has a guaranteed minimal throughput and a maximal latency. Given these characteristics, the communication can be modeled as if it takes place through completely independent virtual point-to-point connections.

These connections can be modeled together with the actors of a job in one SDF graph (Poplavko et al., 2003). Given this SDF graph, the guaranteed minimal throughput of a hard real-time job can be determined.

6. RESOURCE ARBITRATION

In this section, we show that the resource conflicts that are resolved at run-time by TDMA arbiters, can be taken into account in an SDF model. Conservative token arrival times are observed during self-timed execution of this SDF model in a simulator. The same SDF model is analyzed in Section 7 to obtain the sensitivity for fluctuations in the response times of actors on the temporal behavior of a job.

Resource conflicts can occur if multiple actors execute on one processor. These resource conflicts can be resolved at compile time or at run time. The resource conflicts can be resolved at compile time by computing offline a valid schedule for the SDF graph under consideration. In the static order scheduling approach (Sriram and Bhattacharyya, 2000), the execution order of the actors in this offline computed schedule is enforced at run time. If the static order scheduling approach is applied, then a decrease in response time of actors can only result in an earlier arrival of tokens and an increase in throughput of the system. The reason is that there is a one to one correspondence between actors in the system and the actors in the SDF model and that it is known that the self-timed execution of the SDF model is monotonic (see Section 4).

An important disadvantage of the static order scheduling approach is that it cannot be applied if the execution of actors is conditional, as is the case in the H263 video decoder example in Section 9. The execution of actors is conditional if a value of a token determines whether an actor will be executed or not. In a static order schedule it can occur that if, for example, actor A is not executed, then another actor B will wait forever for a token produced by actor A. Other actors on the same processor as actor B will not be executed as long as actor B waits for the token because the execution order of actors on the same processor is predefined and fixed.

Resource conflicts can also be resolved at run time by an arbiter (local scheduler). In the case that arbitration is performed at run time, the arrival of tokens determines whether an actor will be executed or not, and what the execution order of the actors on a processor will be. In the case that, for example, TDMA arbitration is applied, then the effects of the TDMA arbitration on the arrival time of the tokens can be taken into account in the response times of the actors in the SDF model. A proof that TDMA arbitration can be modeled implicitly in the response time of an actor is presented in the next paragraphs. This proof demonstrates with mathematical induction that tokens will not arrive later in the implementation than during selftimed execution of an

HSDF model. It is sufficient to prove for one actor executed during a TDMA time slice on a processor that the actor will not produce tokens later in the implementation than in the HSDF model because the selftimed execution of an HSDF model is monotonic.

In this proof, an abstract representation of a processor is used which executes actor A1 during interval T_1 in a period T. This representation is shown in Figure 4-9. Actor A1 starts its execution during interval T_1, as soon as the previous execution of actor A1 has finished and an input token has arrived. If actor A1 did not finish its execution at the end of the interval T_1 then this actor will be preempted and it will continue its execution in the next period. The additional time due to context switches can be included in the (worst-case) response time of an actor, because the maximum number of context switches that can happen during the execution of an actor is known at compile time. The time $p(j)$ denotes the execution time of the j-th execution of actor A1 when it executes on the processor without being preempted.

Two cases should be distinguished to determine the response time of an actor. An actor can start at the begin of the interval T_1 or during the interval T_1. If the actor starts at the begin of an interval and $p(j) = 2.5\ T_1$ then this actor will be preempted twice, as is shown in Figure 4-7. Given that the actor is preempted twice then the actor will finish its execution $p(j) + 2(T - T_1)$ after it is started. In other words the Interruption time (I1) of the actor is in this case according to (4-4).

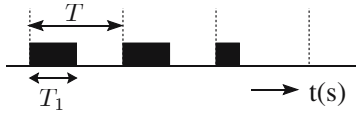

Figure 4-7. Stretch of the response time of an actor due to preemption in the case that the actor starts at the begin of interval T_1.

$$I1(j) = (T - T_1)(\left\lceil \frac{p(j)}{T_1} \right\rceil - 1) \tag{4-4}$$

On the other hand, if the execution of an actor starts during the time slice T_1, as is shown in Figure 4-8 then this actor will be preempted 3 times. Given that the actor is preempted 3 times, then the actor will finish its execution $p(j) + 3(T - T_1)$ after it is started. In other words, the interruption time (I2) of the actor is in this case according to (4-5).

$$I2(j) = (T - T_1)(\left\lceil \frac{p(j)}{T_1} \right\rceil) \tag{4-5}$$

Figure 4-8. Stretch of the response time of an actor due to preemption in the case that the actor can start at any point in time during interval T_1.

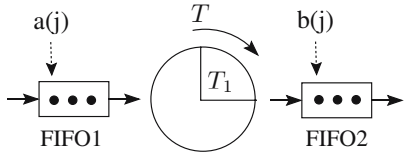

Figure 4-9. Abstract representation of the time wheel of a TDMA arbiter. The time wheel rotates every period T. Actor A1 can execute during slot 1 with duriation T_1.

The arrival time of the j-th token in FIFO1 and FIFO2 is denoted in Figure 4-9 by $a(j)$ and $b(j)$ respectively. The moment in time that the j-th execution of actor A1 finishes, is denoted by $f(j)$. During the j-th execution of actor A1, the j-th token is consumed from FIFO1 and the j-th token is produced in FIFO2. Therefore is $a(j) \leq f(j)$ and $b(j) \leq f(j)$. It will be proven for the HSDF model in Figure 4-10 that if (4-6) holds that then also (4-7) holds, where $\hat{a}(j)$ and $\hat{b}(j)$ denote the arrival time of tokens in the SDF model. The position of the initial token at time t=0 is as shown in Figure 4-10. However the position of the time-wheel in the implementation at time t=0 is unknown. Given this, it will be proven with mathematical induction for $j \geq 0$ that if (4-6) holds then also (4-7) holds.

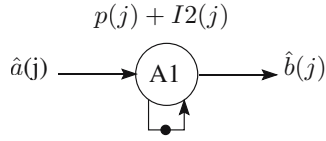

Figure 4-10. SDF model of an actor executed during a time slice on a processor.

$$a(j) \leq \hat{a}(j) \tag{4-6}$$

$$b(j) \leq \hat{b}(j) \tag{4-7}$$

Given that the position of the time-wheel of the implementation is unknown and that initially actor A1 does not execute then $f(0)$ is:

$$f(0) \leq a(0) + p(0) + \max(I1(0), I2(0)) \leq a(0) + p(0) + I2(0) \quad \text{(4-8)}$$

For the arrival time of the first output token in the HSDF model it holds that:

$$\hat{b}(0) = \hat{a}(0) + p(0) + I2 \quad \text{(4-9)}$$

From (4-6), (4-9) and (4-8) it follows that:

$$f(0) \leq \hat{b}(0) \quad \text{(4-10)}$$

Now we want to establish our inductive step by showing how the truth of our induction hypothesis in (4-11) forces us to accept that $f(j+1) \leq \hat{b}(j+1)$.

$$f(j) \leq \hat{b}(j) \quad \text{(4-11)}$$

For the implementation and $j \geq 0$ the following equations hold in which the intermediate variables tx and ty are defined:

$$tx = a(j+1) + p(j+1) + \max(T - T_1 + I1(j+1), I2(j+1)) \quad \text{(4-12)}$$

$$ty = f(j) + p(j+1) + \max(T - T_1 + I1(j+1), I2(j+1)) \quad \text{(4-13)}$$

$$f(j+1) \leq \max(tx, ty) \quad \text{(4-14)}$$

Equation 4-14 can be rewritten as:

$$f(j+1) \leq \max(f(j), a(j+1)) + p(j+1) + I2(j+1) \quad \text{(4-15)}$$

Equation 4-14 holds because: if $a(j+1) > f(j)$ then token $j+1$ has arrived after the j-th execution of actor A1 has finished (see Figure 4-11). After arrival

of token $j+1$ it will take maximally $p(j+1) + \max(T - T_1 + I1(j+1), I2(j+1))$ before the $(j+1)$-th execution of actor A1 finishes. It takes $p(j+1) + T - T_1 + I1(j+1)$ before $(j+1)$-th execution of actor A1 finishes, if the position of the time wheel is such that token $j+1$ arrives during interval $T - T1$, otherwise it takes $p(j+1) + I2(j+1)$.

If $a(j+1) \leq f(j)$ then token $j+1$ has arrived in FIFO1 before the j-th execution of actor A1 has finished (see Figure 4-12). After the j-th execution of actor A1 has finished it takes maximally $p(j+1) + \max(T - T_1 + I1(j+1), I2(j+1))$ before the $(j+1)$-th execution of actor A1 has finished. It takes $p(j+1) + T - T_1 + I1(j+1)$ before the $(j+1)$-th execution of actor A1 finishes, if the position of the time wheel is such that the j-th execution of actor A1 finishes at the end of interval T_1 otherwise it takes $p(j+1) + I2(j+1)$.

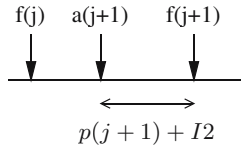

Figure 4-11. Arrival of token j+1 in FIFO1 after the j-th execution of actor A1 has finished.

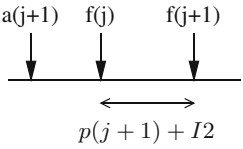

Figure 4-12. Arrival of token j+1 in FIFO1 before the j-th execution of actor A1 has finished.

For the SDF model and $j \geq 0$ the following equations hold in which the intermediate variables tp and tq are defined

$$\hat{b}(j+1) = \max(tp, tq) \tag{4-16}$$

with

$$tp = \hat{a}(j+1) + p(j+1) + I2(j+1) \tag{4-17}$$

$$tq = \hat{b}(j) + p(j+1) + I2(j+1) \tag{4-18}$$

It follows from (4-12), (4-17) and (4-6) that $tx \leq tp$. From (4-13), (4-18), and our induction hypothesis in (4-11) it follows that $ty \leq tq$. This results in the conclusion that $f(j) \leq \hat{b}(j)$ for $j \geq 0$ because:

$$tx \leq tp \wedge ty \leq tq \Rightarrow max(tx, ty) \leq max(tp, tq) \tag{4-19}$$

Given that $b(j) \leq f(j)$ we arrive at the conclusion that (4-7) holds for $j \geq 0$. □

In the proof an HSDF actor was considered with only one input and one output and FIFO buffers with an infinite capacity were assumed. However the proof also holds for an SDF actor with multiple inputs and outputs and buffers with a finite capacity. The reason is that the event $a(j)$ which denotes the arrival of a token in FIFO1 in Figure 4-9 is equivalent to the event which denotes that sufficient tokens are available on each input of an SDF actor. The proof also holds for SDF actors with multiple outputs because all output tokens are produced before the SDF actor finishes its execution. The availability of space in a finite FIFO buffer can be modeled as the presence of a space token on an input of the SDF actor.

The use of TDMA arbitration can be taken into account in the SDF model of hard real-time jobs by setting the WCRT of the actor Ax according to (4-20), in which P denotes the WCET of actor Ax if it would be executed on the processor without being preempted.

$$\text{WCRT}_{Ax}(j) = P + (T - T_1) \left\lceil \frac{P}{T_1} \right\rceil \tag{4-20}$$

Given these worst-case response times of actors, the worst-case arrival times of the tokens in the system can be derived from a self-timed execution of the SDF model. Also the minimal throughput of the system that will obtained equals 1/MCM of this SDF model. This SDF model can also be used with response times instead of worst-case response times. An upperbound on the RT of the j-th execution of actor Ax in the SDF model is equal to:

$$\text{RT}_{Ax}(j) = p(j) + (T - T_1) \left\lceil \frac{p(j)}{T_1} \right\rceil \tag{4-21}$$

Conservative token arrival times are then observed during self-timed execution of the SDF model due to monotonicity of the system. Conservative token arrival times are observed because an earlier arrival of a token can only result in an earlier start of an actor in the SDF model and an earlier production of a result. Therefore conservative arrival times are observed if the i-th response time of actor A_x in the SDF model is not shorter then the i-th response time of

actor A_x in the implementation. This is the case if response times according to (4-21) are used in the SDF model. An important advantage of the use of an SDF model instead of cycle true simulation model of the system is that the arrival time of tokens during self-timed execution is conservative while this is not the case for a cycle true simulation model. The reason is that the initial position of the time wheels in the system at time t=0 is not known. Another advantage is that execution of the SDF model will be much faster than simulation of a cycle true model because an SDF model is a more abstract model.

7. SENSITIVITY ANALYSIS & REDUCTION

This section describes dataflow analysis techniques that are used to determine the FIFOs of which the capacity should be increased, in order to reduce the sensitivity of a soft real-time job, for fluctuations in the response times of the actors. A lower sensitivity of a job will reduce the deadline miss rate which enhances the quality of experience of the user.

For soft real-time jobs a predicted MCM can be calculated with (4-2) given the resource budgets of a job and by using the predicted response times of the actors instead of the WCRT of the actors. Here it is assumed that the PRT of an actor is equal to the average response time of this actor on a processor with TDMA arbitration and representative input data.

It is obvious that this predicted MCM is not smaller than the actual MCM if the response times of the actors is smaller than the PRT of the actors. The predicted MCM is also not smaller than the actual MCM if the cycle mean of a cycle in the SDF graph, to which the actors belong that have an RT larger than their PRT, does not exceed the predicted MCM. The Cycle Mean (CM) is defined in (4-3). In other words the temporal behavior of a job is more sensitive for deviations in the response times of actors which belong to cycles of which the CM is likely to be larger than the predicted MCM. By increasing the FIFOs capacity, the CM of these cycles can be decreased such that the job becomes less sensitive.

That the sensitivity of a job can be reduced by increasing the FIFO capacities can be seen as follows. Assume that the job is described by the SDF graph in Figure 4-13. The PRT of actor A1 in this job is chosen to be equal to the average response time measured over 3 successive executions of this actor. If the desired MCM is 2T then the FIFO capacity should be at least 2 tokens given the PRT of the actors. However, it is likely that the actual MCM is larger than the desired MCM because the RT of actor A1 can be larger then its PRT which results in a CM larger than the predicted MCM.

The actual MCM would not be larger than the desired MCM if a FIFO capacity of 6 instead of 2 tokens was applied. That this is the case can be intuitively seen as follows. Assume the an actor A1' in Figure 4-14 requires 3

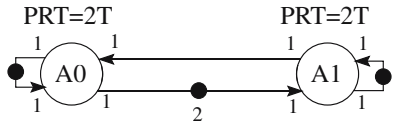

Figure 4-13. SDF with a predicted MCM of 2T.

tokens instead of 1 token on its input before it fires and that after firing it executes internally 3 times the same code segment. Actor A1' would have in this case a PRT equal to the maximum response time of 3 successive executions. In Figure 4-14 we assumed that out of the 3 successive executions of actor A1, 2 executions have a response time smaller than T and one a response time smaller than 4T. In this case is the PRT of actor A1' equal to 6T. Given the PRT there is a FIFO capacity needed of 6 tokens for a desired MCM of 2T. This MCM can be obtained with (4-2) after the SDF in Figure 4-14 is transformed in an HSDF with the algorithm described on page 40 in (Sriram and Bhattacharyya, 2000). The longest path in this HSDF contains 3 times actor A0 and once actor A1' and is 12 T long. Therefore, there must be 6 tokens on this path for an MCM of 2T. Given these 6 tokens an MCM of 2T will be obtained if actor A1 in the implementation fires as soon as there is one input token available. The reason is that starting of the actor with only 1 instead of 3 tokens can only result in an earlier production of tokens.

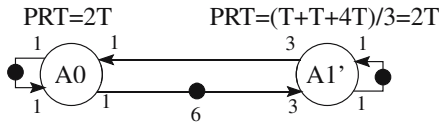

Figure 4-14. SDF with a predicted MCM of 2T.

8. PREDICTABLE DYNAMIC DATA FLOW

In this section a so-called Predictable Dynamic Data-Flow (PDDF) graph is introduced in which the conditional executions of actors can be expressed as well as a variable but bounded number of executions of actors can be expressed. An important property of a PDDF is that it can be analysed with the in Section 4 and Section 7 described analysis techniques.

An H263 video decoder is an example in which it depends on the values of the input data which actors will be executed and which not. In this decoder different actors are executed in the case that an I-frame or a P-frame is decoded. The conditional execution of actors cannot be made explicit in SDF

graphs but can be made explicit in Boolean Data-Flow (BDF) graphs (Buck, 1993). However, the use of a BDF graph is undesirable because the detection of deadlock is undecidable for an arbitrary BDF graph. By restricting, with construction rules, the type of BDF graphs that can be expressed so-called well-behaved dataflow graphs (Gao, Govindarajan and Panangaden, 1992) are obtained. These well-behaved dataflow graphs are per construction deadlock free. However, to derive a tight lower bound on the throughput of an application with MCM analysis it is also necessary that the actors that are conditionally executed do not share resources or are executed mutual exclusive. Mutual exclusive execution is typically desirable because sharing of resources reduces the resource requirements. Mutual exclusive execution can be guaranteed by extending the well-behaved dataflow graph with a so-called mode manager actor M, as is done in Figure 4-15. This mode manager actor provides N times a control token with the same boolean value for the switch and select actor and then waits till the select actor has been executed N times before it produces a control token with possibly a different boolean value. It is required that the select actor produces a token at the end of its execution. The construct in Figure 4-15 guarantees that there is no input token available for actor A0 and A1 at the same time and that therefore the execution of these actors is mutual exclusive. That N times the same control token is produced by actor M is made explicit in Figure 4-15 with the N[T/F] annotation. The name Predictable Dynamic Data-Flow (PDDF) graphs has been given to dataflow graphs in which the construct in Figure 4-15 is used to express conditional execution of actors.

The minimal throughput of a PDDF graph can be determined by calculating the MCM of the PDDF graph with (4-2) which is the same equation as is used for the calculation of the MCM of an SDF graph. The same equation can be used because the PDDF graph in Figure 4-15 has an equivalent worst-case temporal behavior as the SDF graph in Figure 4-16. This is the case because the PDDF graph in Figure 4-15 is per construction deadlock free. Also, the execution of the actors A0 and A1 is by construction mutually exclusive. Therefore, it can be assumed during MCM analysis that the actors A0 and A1 are both executed for each input token of the select actor but that each of these actors is executed on its own private processor. If actors A0 and A1 are executed for each input token then the switch and select actors should behave like ordinary actors which consume tokens from all inputs and produce tokens on all their outputs and have a zero WCRT. The value of the control token provided by the mode manager actor M to the switch (SW) actor is ignored because the data token must be duplicated by the switch actor to both outputs. The select (SE) actor should consume a token produced by actor A0 as well as A1 and copy one of these tokens to its output. Because the value of the control token is irrelevant for the worst-case temporal behavior there is no need to make explicit that the same control token is sent to the switch as well as the select actor.

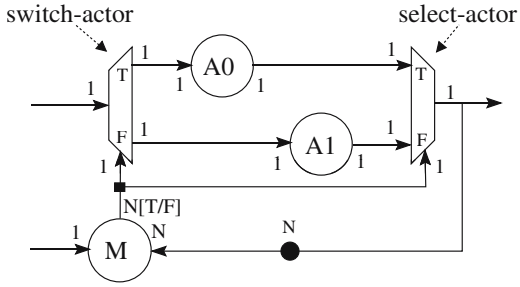

Figure 4-15. Predictable Dynamic DataFlow (PDDF) graph with mutual exclusive execution of actors in the True and False branch.

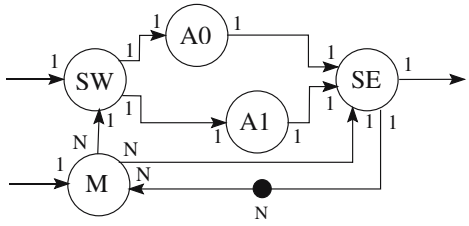

Figure 4-16. SDF graph with the same worst-case temporal behavior as the PDDF graph in Figure 4-15.

A PDDF construct in which actor A2 is executed p times is shown in Figure 4-17. This construct executes in bounded memory because actor A1 informs actor A3 about the number of tokens it must consume. Actor A1 informs actor A3 by sending one token with value p to A3. This is indicated in Figure 4-17 with the notation 1[p]. During calculation of the PDDF graph's MCM the maximum value of p must be used because a larger value p will result in more executions of actor A2 and a later start of actor A3.

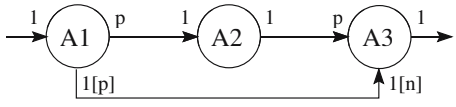

Figure 4-17. PDDF graph in which actor A2 is executed p times.

9. DATAFLOW MODEL OF AN H263 VIDEO DECODER

A dataflow model of an H263 video decoder is presented in this section which illustrates the use of the modeling techniques that were introduced in the previous sections. This H263 video decoder is a soft real-time job of which the values of the input data determine whether some of the actors will be executed or not. Also the number of tokens produced and consumed by the actors can be data dependent. Despite the dynamic behavior of this job, it remains possible to derive the minimal capacity of the FIFOs as well as conservative arrival times of tokens with a dataflow model.

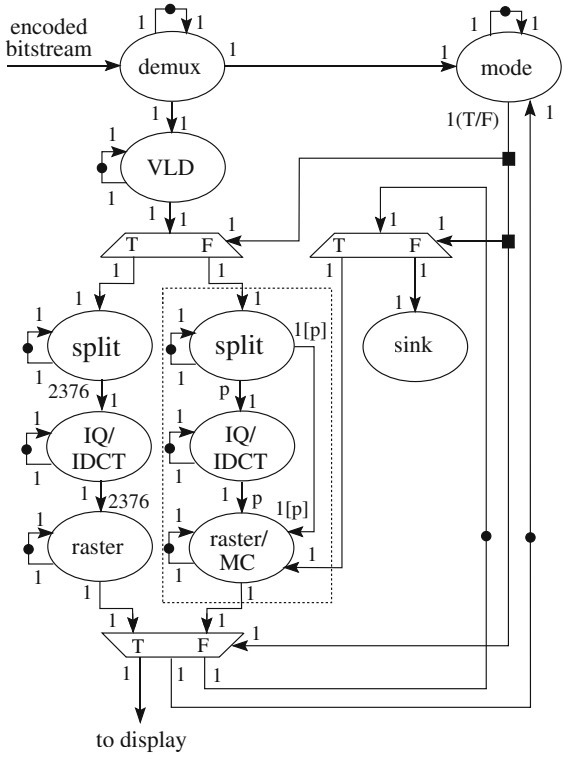

Figure 4-18. Predicatable dynamic dataflow model of an H263 video decoder.

The predictable dynamic dataflow model of an H263 decoder is shown in Figure 4-18. This decoder receives a bit stream which is split by the demultiplexer (demux) actor in a token for the mode manager (mode) actor and a token for variable length decoder (VLD) actor. The token for the mode actor

indicates whether the next frame to decode is an Intra (I) or a Predicted (P) frame. The token for the VLD actor contains one encoded frame.

The token produced by the VLD actor in case of an I-frame in CIF resolution (352×288 pixels) is split in 2376 tokens of which each token contains an encoded block. These 2376 tokens are processed by the combined Inverse Quantization (IQ) and Inverse Discrete Cosine Transform (IDCT) actor and a rasterization (raster) actor. The result of the rasterization actor is one decoded frame which can be displayed. That a complete frame has been decoded and that the next frame can be decoded is indicated by sending a token to the mode manager.

The token produced by the VLD actor in case of a P-frame is split in p tokens of which each token contains an encoded macro block. The split actor also notifies the rasterization/Motion Compensation (raster/MC) actor that it should consume p tokens. The tokens produced by the split actor are processed by an IQ/IDCT actor and the raster/MC actor. The raster/MC actor receives also a token which contains the previous frame.

The production of a variable number of tokens by the split actor is allowed in this dataflow graph because a maximum number of tokens ($p \leq 2376$) is known at compile time. Given this maximum number of tokens, the minimum FIFO capacity between the split actor and the IQ/IDCT actor can be derived, as well as the minimum FIFO capacity between the IQ/IDCT actor and the raster/MC actor. Another important property is that conceptually one actor could be introduced, which is indicated in Figure 4-18 by the dashed box, in which the production and consumption of a variable number of tokens is hidden.

Conservative arrival times of tokens can be observed during simulation of the dataflow model of the H263 decoder, given that the response times of the actors are according to (4-21).

10. CONCLUSION

Embedded multiprocessor systems in consumer products execute a combination of soft real-time and hard real-time jobs that process data streams. Dataflow models in which computation, communication and arbitration is modeled can be used to derive the minimal throughput of the hard real-time jobs, using MCM analysis. For soft real-time jobs, simulation of these dataflow models are used to test statistical assertions given representative input streams. The simulation effort is reduced and the confidence of the simulation results is improved by making only use of predictable arbitration policies (e.g. TDMA) in the proposed network based multiprocessor system. The simulation effort is reduced because the use of predictable arbitration policies eliminates the interference between jobs, and guarantees that conservative arrival times of to-

kens are observed during simulation of the dataflow model of a job. Dataflow analysis techniques are used to estimate the resource budgets of soft real-time jobs. With these analysis techniques the buffers are derived which should be increased to reduce the sensitivity for fluctuations in the response time of actors on the temporal behavior of a job. A predictable dynamic dataflow model of an H263 video decoder job is presented in which conditional construct determine which actors are executed during the decoding of I-, and P-frames. The temporal behavior of such a job can be analyzed with the presented analysis and simulation techniques.

References

Bacelli, F., Cohen, G., Olsder, G. and Quadrat, J.-P., 1992, *Synchronization and Linearity*, John Wiley & Sons, Inc.

Bekooij, M., Moreira, O., Poplavko, P., Mesman, B., Pastrnak, M. and van Meerbergen, J., 2004, Predictable embedded multiprocessor system design, *Proc. Intl Wor kshop on Software and Compilers for Embedded Systems (SCOPES)*, LNCS 3199, Springer.

Buck, J., 1993, *Scheduling dynamic dataflow graphs with bounded memory using the token flow model*, PhD thesis, Univ. of California, Berkeley.

Cochet-Terrasson, J., Cohen, G., Gaubert, S., McGettrick, M. and Quadrat, J.-P., 1998, Numerical computation of spectral elements in max-plus algebra, *Proc. IFAC Conf. on Syst. Structure and Control*.

Gangwal, O., Nieuwland, A. and Lippens, P., 2001, A scalable and flexible data synchronization scheme for embedded hw-sw shared-memory systems, *Int'l Symposium on System Synthesis (ISSS)*, ACM, pp. 1–6.

Gao, G., Govindarajan, R. and Panangaden, P., 1992, Well-behaved dataflow programs for DSP computation, *International Conference of Acoustics, Speech and Signal processing*.

Hee, K. v., 1994, *Information System Engineering*, Cambridge University Press.

Kahn, G., 1974, The semantics of a simple language for parallel programming, *Proceedings IFIP Congress*, pp. 471–475.

Kock, E., Essink, G., Smits, W., Wolf, P. v. d., Brunel, J.-Y., Kruijtzer, W., Lieverse, P. and Vissers, K., 2000, Yapi: Application modeling for signal processing systems., *In Proceedings of 37th Design Automation Conference (DAC00)*, Los Angeles, pp. 402–405.

Kopetz, 1997, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer.

Lawler, E., 1976, *Combinatorial optimization: Networks and Matroids*, Holt, Reinhart, and Winston, New York, NY, USA.

Lee, E. and Messerschmitt, D., 1987, Synchronous data flow, *Proceedings of the IEEE*.

Li, Y.-T. S. and Malik, S., 1999, *Performance analysis of real-time embedded software*, ISBN 0-7923-8382-6, Kluwer academic publishers.

Petri, C., 1962, *Kommunikation mit Automaten*, PhD thesis, Institut für Instrumentelle Mathematik, Bonn, Germany.

Pop, T., Eles, P. and Peng, Z., 2002, Holistic scheduling of mixed time/event-triggered distributed embedded systems, *Proc. Int'l Symposium on Hardware/Software Codesign (CODES)*, pp. 187–192.

Poplavko, P., Basten, T., Bekooij, M., Meerbergen, J. v. and Mesman, B., 2003, Task-level timing models for guaranteed performance in multiprocessor networks-on-chip, *Proc. Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pp. 63–72.

Rajkumar, R., Juwa, K., Moleno, A. and Oikawa, S., 1998, Resource kernels: A resource-centric approach to real-time and multimedia system, *SPIE/ACM Conference on Multimedia Computing and Networking*.

Richter, K., Jersak, M. and Ernst, R., 2003, A formal approach to MpSoC performance verification, *IEEE computer* **36**(4), 60–67.

Rijpkema, E., Goossens, K., Rădulescu, A., Dielissen, J., Meerbergen, J. v., Wielage, P. and Waterlander, E., 2003, Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip, *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 350–355.

Sriram, S. and Bhattacharyya, S., 2000, *Embedded Multiprocessors: Scheduling and Synchronization*, Marcel Dekker, Inc.

Zhang, H., 1995, Service disciplines for guaranteed performance services in packet-switching networks, *Proceedings of the IEEE* **83**(10), 1374–96.