Evaluation of the throughput computed with a dataflow model - A case study

Arno Moonen[1], Marco Bekooij[2], Rene van den Berg[2], Jef van Meerbergen[1,3]

[1]University of Technology, Eindhoven, The Netherlands
[2]NXP Semiconductors, The Netherlands
[3]Philips Research, Eindhoven, The Netherlands

Evaluation of the throughput computed with a dataflow model - A case study

Arno Moonen[1], Marco Bekooij[2], Rene van den Berg[2], Jef van Meerbergen[1,3]
[1] University of Technology, Eindhoven, The Netherlands
[2] NXP Semiconductors, The Netherlands
[3] Philips Research, Eindhoven, The Netherlands
A.J.M.Moonen@tue.nl

Abstract

Providing real-time guarantees in complex, heterogeneous, and embedded multiprocessor systems is an important issue because they affect the perceived quality. Digital signal processing algorithms are often modeled with dataflow models. A guaranteed minimum throughput can be computed from such dataflow model. In this paper we analyze three causes for the difference between the computed and measured throughput. We measure the throughput with a cycle accurate simulation. For our channel equalizer application the measured throughput is 10.1% higher than the computed minimum throughput.

1 Introduction

Many consumer applications process a number of data streams and have throughput and latency requirements. Radios are often designed as hard real-time systems [9], because they suffer from steep quality degradation if the throughput and latency requirements are not met. Hard real-time systems require a guaranteed minimum throughput and bounded latency. Missing a deadline, e.g. at the digital to analog converter, can cause a click in the audio.

Reasoning about real-time guarantees in embedded heterogeneous multiprocessor systems is challenging. Concurrency, resource sharing and many operation modes complicate the analysis. Many recent studies have focused on real-time analysis, which includes checking whether the timing constraints are met, identification of possible bottlenecks, and estimation of resource utilization.

We identify two categories for existing analysis techniques, namely simulation and exhaustive analysis. Potential disadvantages of simulation are a high running time, an incomplete coverage and failure to identify possible bottlenecks. The simulation tools accompanying the modeling language SystemC and POOSL [4] are used to simulate transaction level models [3]. Transaction level models
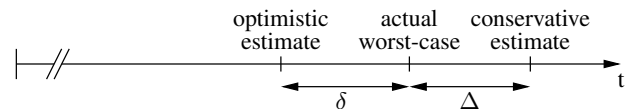

Figure 1. The worst-case arrival time of an audio sample

tradeoff accuracy for running time. An optimistic worst-case arrival time of e.g. audio samples can be determined via cycle-accurate simulation. Longer simulation runs can increase the accuracy and therefore decrease δ in Figure 1. However, from an optimistic estimate we are unable to guarantee that throughput and latency requirements are met.

Real-time calculus [12] falls in the category exhaustive analysis. The main drawback of real-time calculus is that it has difficulties in handling cyclic dependencies that influence temporal behavior.

Another exhaustive analysis model, which is a subclass of timed Petri nets, is called marked graph [2]. Single Rate Dataflow (SRDF) graphs [7] have the same expressibility as marked graphs. In a SRDF graph we can analytically derive the length of all the cycles in the graph. The cycle with the maximum length, which is called the Maximum Cycle Mean (MCM) [11], is related to the inverse of the throughput. More versatile than SRDF graphs are Multi Rate Dataflow (MRDF) graphs [7] and Cyclo-Static Dataflow (CSDF) graphs [1] [10]. MRDF and CSDF graphs can be unfolded into an equivalent SRDF graph of which the MCM can be calculated [11] [1]. Another analysis method for deriving the throughput is presented in [5], which is based on explicit state-space exploration of an MRDF graph.

Conservative worst-case arrival times of audio samples can be computed with a dataflow model. From a conservative estimate we are able to guarantee that throughput and latency requirements are met. However, it is usually not

```
t1()
{
    int x,y1,y2;
    if(i mod 8==0) {
        x=read(A);
        y1=func1(x);
        write(B,y1);
        y2=func2(x,y1);
        write(C,y2);
    }
    i=i+1;
}
```

Figure 2. Pseudocode of task *t1*, which has a cyclo static behavior

Figure 3. Actor *t1*, which is representing task *t1* in a CSDF model

known how far the distance between the conservative estimated and actual worst-case arrival time is. A large distance Δ can result in a significantly overdimensioned system with a higher cost than strictly needed.

In this paper we identify and quantify the causes that are responsible for the difference between the conservative estimated throughput and measured throughput. This difference is determined for a channel equalizer application. The conservative estimated throughput is derived with a CSDF model of the channel equalizer. The measured throughput is determined with a cycle accurate simulator of the complete multiprocessor system.

The outline of this paper is as follows. We introduce the CSDF model in Section 2. In Section 3 we describe the multiprocessor architecture on which the channel equalizer is executed. In Section 4 we describe the CSDF model of the channel equalizer application. We describe the mapping of the channel equalizer application onto the multiprocessor system in Section 5. Each mapping decision is modeled with additional constraints in the CSDF model. We use the CSDF model to compute a conservative estimate of the throughput. In Section 6 we compare the computed and the measured throughput. We conclude in Section 7.

2 The CSDF model

In this section we present the CSDF model. We use CSDF model because it can model cyclic dependencies and the channel equalizer application can intuitively be modeled with a CSDF. In this section, we also identify three causes for the difference between the computed and measured throughput, which we will further investigate in later sections.

A CSDF model is represented as a directed graph. The tasks in the implementation are represented by nodes, which we call actors. A communication channel between two tasks is represented by an edge. Tasks and actors transform input streams into output streams. An example of a task is shown in Figure 2. This task reads from channel A
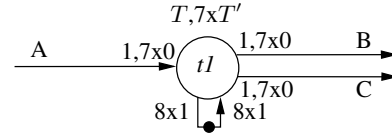
and writes to channel B and C. Variable `i` is a global variable of the type integer and is initialized to zero. After each execution, `i` is increased by one. The condition of the if-statement is `true` if `i` is a multiple of eight, therefore, task *t1* has a cyclically changing but predefined behavior. We call this behavior a cyclo static behavior.

Task *t1* is represented by actor *t1* depicted in Figure 3. An actor is executed as soon as the firing rule is evaluated as `true`. A firing rule is a boolean expression in the number of tokens present at the inputs. A token is a data element that is transferred over a communication channel, e.g. an audio sample. When an actor executes, it consumes a certain number of tokens from its inputs, and it produces a number of tokens to its output. The number of tokens consumed during one execution is called the consumption rate and the number of tokens produced during one execution is called the production rate. The consumption rate of actor *t1* at channel A is one if `i` is a multiple of eight and zero in the remaining executions. This cyclo static behavior is represented by 1,7x0 which is equivalent to 1,0,0,0,0,0,0,0. The production rate at channel B and C is 1,7x0. The execution time of actor *t1* is T if `i` is a multiple of eight and T' in the remaining executions, which is represented by $T,7\mathrm{x}T'$.

A task can start its next execution after its previous execution is finished. We model this with a self-edge, which is an edge where the source and destination actors are the same, with one initial token. An initial token is depicted as a black dot. In order to keep the figures, which represent a CSDF model, simple we do don't draw the self-edges of a task and don't show the production and consumption rates equal to one.

A first-in-first-out (FIFO) queue in the implementation can be modeled with a forward and backward edge. The capacity of the queue is indicated with initial tokens on the backward edge.

The difference between a task in the implementation and an actor in the CSDF model is the following. A task is enabled by a scheduler while an actor is enabled by the firing rule. The execution time of a task can vary but should be bounded. The execution time of an actor is cyclo static. A task consumes its input data and produces its output data somewhere during its execution. An actor consumes its input and produces its output at the end of its execution. A

2

task is blocked if no input data or output space is available. An actor is non-blocking because the firing rule is only enabled if input data and output space, in the case of a FIFO, is available.

A CSDF model can be unfolded into an equivalent SRDF model from which the MCM can be computed. The throughput, e.g. number of tokens produced per second, is related to the inverse of the MCM. The throughput measured with a cycle accurate simulator is higher than the throughput computed with the CSDF model. There are three causes for the difference between the measured and computed throughput.

First, The execution time of an actor is a compile time estimate of the Worst Case Execution Time (WCET). Conservative estimates on the WCET have been actively investigated in the real-time system design community. The variation on the execution time is for example a consequence of conditional branches, data dependent loops and the behavior of caches.

Second, inter-task communication results in dependencies between tasks, e.g. one task can start consuming data after another task has finished producing its data. In the implementation a task consumes its input data and produces its output data somewhere during its execution. An actor consumes its input and produces its output at the end of its execution. In a CSDF model a tradeoff can be made between accuracy and the number of phases in the cyclo static behavior. A higher number of phases results in a larger equivalent SRDF model.

Third, even when we run our system with identical input data multiple times, the temporal behavior of each run can vary, because an arbiter has to grant permission to access a shared resource and the behavior of this arbiter is not completely known at design time. In our multiprocessor architecture, which is introduced in the next section, the only shared resource is the network. We model the worst-case temporal behavior of the arbiter in the latency of the network.

3   Architecture

In this section we describe the architecture of our heterogeneous multiprocessor system on which the channel equalizer is executed.

Our multiprocessor system is composable. A system is composable if the behavior of application A can not influence the temporal behavior of application B.

The architecture consists of tiles that communicate via a network, as depicted in Figure 4. The network allows to set up a point-to-point connection that supports Guaranteed Throughput (GT) service [6]. The bandwidth of a GT connection is configured before starting the application. The latency of a GT connection is bounded. Tiles can contain a
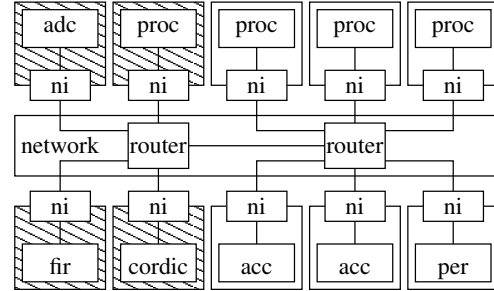

Figure 4. Heterogeneous multiprocessor system

programmable core (proc), a hardware accelerator (acc) or a peripheral (per). Each tile is connected to a network interface (ni). The network interface is connected via a link to a router. In each network interface there are a number of FIFOs. The number of FIFOs and the capacity of each FIFO is selected at design time. The FIFO capacity should be at least the size of a token, such that a task can store its input and output data. In our multiprocessor system the capacity of each FIFO is 32 words, which is sufficiently large. Clock domain crossings are implemented in the network interface such that each tile can have its own clock frequency.

When a processor transfers data over the network, it writes the data into a FIFO in the network interface. The processor is blocked if the FIFO is full and it continues again if there is space available. When the processor reads data from the network, it will read the data from a FIFO in the network interface. The processor is blocked if there is no data available and it continues again if there is data.

For our case study we are interested in the throughput of the channel equalizer. The channel equalizer is mapped to the cross-hatched tiles in Figure 4. The analog to digital converter is the input of our channel equalizer and is located in the adc tile. We make use of a CORDIC (COordinate Rotation DIgital Computer) and FIR (Finite Impulse Response) hardware accelerator for performance and cost reasons. These accelerators are located in, respectively, the cordic and fir tile. Furthermore, we make use of one processor tile. The output of the channel equalizer is send to the input of the FM radio receiver, which is mapped on the remaining tiles.

We can derive the channel equalizers throughput independent from the radio receiver because the architecture is composable. The temporal behavior of the channel equalizer is not effected by the FM radio receiver due to the GT connections in the network. Furthermore, we make use of distributed memories, which are local to the processor, such that we can derive tight estimates of the WCET of tasks.
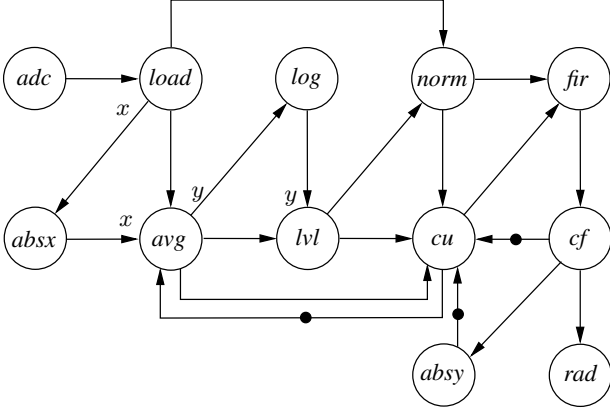
Figure 5. CSDF model of the channel equalizer

4 Application

In this Section we describe our channel equalizer algorithm and present a CSDF model of this application.

Multipath distortion in an FM signal can be reduced with a channel equalizer. FM signals are reflected by houses, cars, hills, etc. and these reflections cause variations in the magnitude and phase of the FM signal. Multipath can be described as a complex digital transversal filter $C(Z)$ because there is a delay between different paths and each path has a different phase and magnitude.

$$C(Z) = a + b \cdot Z^{-\Delta_1} + c \cdot Z^{-\Delta_2} + d \cdot Z^{-\Delta_3} + ... \quad (1)$$

To correct for multipath distortion, a complex digital filter $H(Z)$ can be made which approximates the inverse of the multipath filter $C(Z)$. The desired FM signal is obtained if $H(Z) \cdot C(Z) = 1$. The channel equalizer should be adaptive in portable systems because the channel characteristics vary over time.

The sequential C-code of the channel equalizer application was manually rewritten, such that task level parallelism is explicit. The rewritten C-code is represented as a CSDF model, which is shown in Figure 5. Task *adc* is modeling the strict periodic AD convertor. Task *rad* is modeling the strict periodic input of the FM radio receiver. The tasks *absx*, *log*, *fir*, and *absy* can be mapped on a hardware accelerator to offload the processor. The production and consumption rates denoted by x and y are 1,7x0. Therefore, the number of executions of task *absx* and *log* is eight times lower than the number of executions of the other tasks.

5 Mapping

In this Section we map the channel equalizer application onto our multiprocessor system. First, we determine the binding of tasks to tiles. Secondly, we fix the bandwidth of the point-to-point connections in the network and calculate an upper bound on the latency. Finally, we determine the execution order of the tasks that are mapped to the same tile. After each mapping decision we add constraints to the CSDF model and compute a throughput estimate with MCM analysis.

Our application consists of twelve tasks. The *adc* task represents the incoming data from the adc tile. The *rad* task represents the input of the FM radio receiver. The tasks *adc* and *rad* are strict periodic with the period $1/f_s$, with f_s the sample frequency. We offload the processor by binding the *fir* task to the fir tile and the *absx*, *absy*, and *log* tasks to the cordic tile. The WCET of these tasks is 144ns, as shown in Table 1. The remaining tasks are executed on the proc tile. The WCETs of the tasks executed on the proc tile are computed with a WCET analysis tool. The computed WCET is an upper bound on the execution time of the task.

The MCM in the CSDF graph is 49248ns. During this MCM period, 8 samples are read from *adc* and 8 samples are written to *rad*. Therefore, an estimated throughput is $f_s = 8/(49248 \cdot 10^{-9}) = 162$KHz. This estimate does not include network communication latency and fixed order execution of tasks.

The next step is to set up point-to-point connections in the network. This application requires five connections in the network, namely from adc to proc, from proc to fir, from fir to proc, from proc to cordic, from cordic to proc, and from proc to rad. The tool that comes with the network is able to generate a configuration in such a way that all connections have a guaranteed throughput service. We compute for each point-to-point connection an upper bound on the latency. The upper bounds, which are shown in Table 2, are computed with a CSDF model of the network [8]. Notice that multiple communication channels use one single network connection. For example the channel *norm→fir* and *cu→fir* are both mapped on the connection from the proc to the fir tile. Sharing a network connection can result in a higher worst-case latency. In our implementation we know that their communication is mutually exclusive because tasks have a static order execution.

Communication latency is modeled with actor c1 through c11 in Figure 6. The arrows with an open head are a shorthand notation for a forward and backward edge with a certain number of initial tokens on the backward edge. The number of initial tokens represents a FIFO capacity of 32 words, which is equal to the capacity of the FIFOs in the network interface.

The MCM in the CSDF model in which the WCET

Task	Tile	WCET [ns]
adc	adc	$1/f_s$
load	proc	8x224
absx	cordic	144
avg	proc	704, 7x416
log	cordic	144
lvl	proc	440, 7x24
norm	proc	328
cu	proc	4944, 7648, 6x4944
fir	fir	144
cf	proc	328
absy	cordic	144
rad	-	$1/f_s$

Table 1. The binding of tasks to tiles and their corresponding cyclo static WCET.

actor	channel	token size [words]	latency [ns]
c1	*adc→load*	2	66
c2	*load→absx*	4	114
c3	*absx→avg*	4	114
c4	*avg→log*	4	114
c5	*log→lvl*	4	114
c6	*cf→absy*	4	114
c7	*absy→cu*	4	114
c8	*norm→fir*	3	66
c9	*cu→fir*	17	162
c10	*fir→cf*	4	114
c11	*cf→rad*	2	66

Table 2. Per communication channel the token size and bounded latency.

Figure 6. CSDF model in which WCET of tasks, communication latency and fixed order execution of tasks is modeled

of tasks and worst-case communication latency is modeled is 51940ns. In this case the estimated throughput is $f_s = 8/(51940 \cdot 10^{-9}) = 154$KHz.

Six tasks are executed on the proc tile and three tasks are executed on the cordic tile. Therefore, we derive a fixed order execution for these tasks. We choose the ordering in such a way that the processor utilization is optimized. The fixed order of execution for the tasks executed on the processor is *cu*, *load*, *avg*, *lvl*, *cf*, and *norm*. Dependency edges are added in the CSDF graph to model this fixed order of execution, as shown in Figure 6. The preamble to this fixed order of execution is the execution of tasks *load*, *absx*, *avg*, *log*, *lvl*, and *norm* during the initialization. The preamble is modeled by a different placement of the initial tokens in the CSDF graph. Or in other words, the tasks are executed once in the initialization such that the initial token on the channel
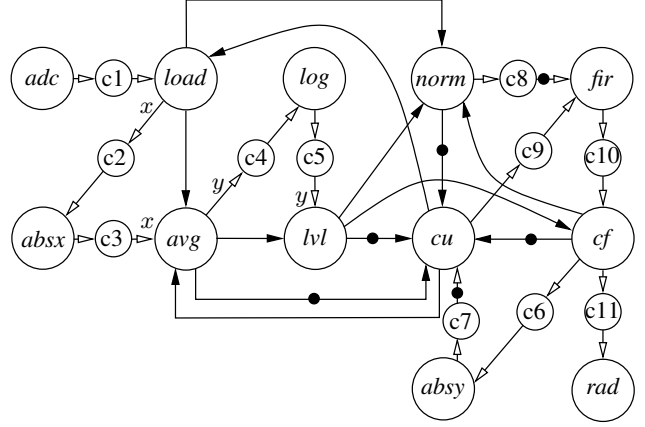
cu→avg in Figure 5 is removed and there are initial tokens generated on the channels *avg→cu*, *lvl→cu*, *norm→cu*, and *norm→fir* in Figure 6.

The MCM in the CSDF model in which the WCET of tasks, communication latency and fixed order execution of tasks is modeled, which is shown in Figure 6, is 54616ns. Therefore, the throughput estimate is $f_s = 8/(54616 \cdot 10^{-9}) = 146.4$KHz. This throughput is the guaranteed minimum throughput of the channel equalizer, because the CSDF model is conservative compared to the implementation.

6 Experiments

In this section we measure the throughput and analyze the impact of the three causes that are introduced in Section 2. Furthermore, we measure the difference between the computed and measured latency.

From the analysis in Section 5 we know that the MCM is 54616ns, therefore, the Cycle Mean (CM) in the implementation is lower than or equal to 54616ns. During one MCM period 8 samples are read from *adc* and 8 samples are written to *rad*. Let $\alpha(i)$ and $\beta(i)$ be the arrival time of token i at the input of *rad*, respectively, in the implementation and in the CSDF model. The CM in the implementation is defined as $CM = \alpha(i) - \alpha(i-8)$. With cycle accurate simulation we measure a maximum CM of 49080ns, an average CM of 48609ns and a minimum CM of 48366ns, after assuring that the tasks *adc* and *rad* not determine the throughput. The difference between the measured maximum CM and the computed MCM is $54616 - 49080 = 5536ns$, which is 10.1% compared to the MCM. We don't know the ac-
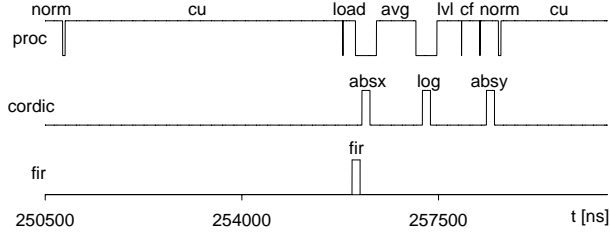
Figure 7. A trace computed from the CSDF model
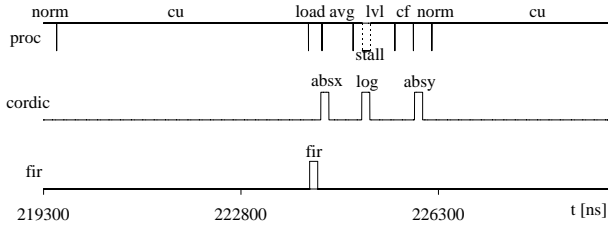

Figure 8. A trace measured with the cycle accurate simulator

Figure 9. Difference between the arrival time of tokens in the dataflow and cycle accurate simulator

tual minimum throughput of the channel equalizer, but it is sure that the difference between the guaranteed minimum throughput and the actual minimum throughput is less than 10.1%.

In Section 2 we introduced three causes for the difference between the computed and measured throughput, namely: variation in the execution time, dependencies between tasks and different degrees of contention. A trace of the schedule gives a good impression about the impact of these causes. A $10\mu s$ trace from the proc, cordic and fir tile is shown in Figure 7 and Figure 8 for, respectively, the CSDF model and implementation. All tasks executed on the processor are part of the critical cycle, therefore, variation in these execution times is effecting the throughput linearly with the slope one. The sum of these computed WCETs is 53520ns in one MCM period. The sum of these measured execution times is 48700ns in one CM period. The difference between the sum of computed WCETs and measured execution times is $53520 - 48700 = 4820ns$, which is 8.8% of the MCM period. For our case study this cause has the biggest impact on the difference between the computed and measured throughput.

The second cause is the variation in the moment when a token is produced or consumed. The processor, in our case, is stalled when it waits for the data coming from the cordic tile. The processor stall time, due to this dependency, is at most 994ns in one MCM period. This 944ns is 1.8% of
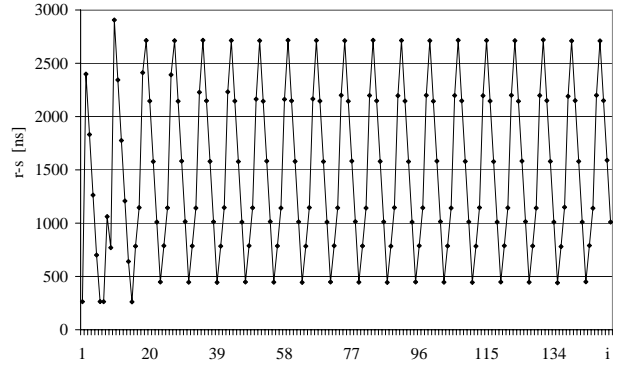
the MCM, therefore, this cause has a small impact on the throughput.

The third cause is the different degree of contention in the network. The CSDF model from Section 5 takes worst-case contention in the network into account. We can build a similar CSDF model for the situation that there is no contention in the network. The difference in MCM for these two models would be 544ns, which is 1% of the MCM from our original CSDF model. Therefore, different degree of contention in the network has a very small impact on the throughput.

Finally, we compare the end-to-end latency between our CSDF model and implementation. For this comparison we make the period of the strict periodic AD converter equal to $55000/8 = 6875ns$, such that the AD converter determines the throughput. We start the channel equalizer at the absolute time t_0, to make sure that the system is correctly initialized. The departure time of the token i at the AD converter in the cycle accurate simulation and the *adc* actor in the CSDF model is $t_i = t_0 + (i + 1) * 6875$. The measured arrival time of token i at the input of the FM radio receiver in the cycle accurate simulator is approximately:

$$\alpha(i) = t_0 + (i + 1) * 6875 + r[ns] \qquad (2)$$

With r the cyclo static variable 865, 1980, 865, 860, 855, 860, 865, 1790. The computed conservative arrival time of token i at the *rad* actor is:

$$\beta(i) = t_0 + (i + 1) * 6875 + s[ns] \qquad (3)$$

With s the cyclo static variable 2005, 4140, 3575, 3010, 2445, 1870, 1305, 2570. We define the latency of token i as the time between the departure from *adc* and the arrival at *rad*. The latency in the cycle accurate simulator is $\alpha(i) - t_i$ and the latency computed from the CSDF model is $\beta(i) - t_i$. The difference between the computed and measured latency

is:

$$(\alpha(i) - t_i) - (\beta(i) - t_i) = \alpha(i) - \beta(i) = r - s \quad (4)$$

With $r - s$ the cyclo static variable 1140, 2160, 2710, 2145, 1580, 1005, 440, 780. The value $r - s$ is measured over time and plotted in Figure 6. The departure of tokens from the AD converter is equal to the departure of tokens from the *adc* actor and $r - s \geq 0$, therefore, the CSDF model is conservative.

7 Conclusion

In this paper we analyze the causes for the difference between the throughput computed with a CSDF model and the throughput measured with a cycle accurate simulator of our multiprocessor system. This difference is quantified for our channel equalizer.

We described three causes for the deviation. First, the execution time of a CSDF actor is a compile time estimate of the WCET of a task. The difference between this estimated WCET and the measured execution time is 8.8%, which effects the throughput linearly with slope one. Second, the actor consumes the input and produces the output at the end of its execution, whereas, in the implementation this is done earlier. This leads to a throughput difference of at most 1.8% for our channel equalizer. As far as this contributor concerned, it is a trade-off between the number of actors and the accuracy in modeling the dependencies. Third, the CSDF model assumes the maximum degree of contention at shared resources, which depends on the arbitration. An arbiter need to support derivation of a tight conservative estimate of the response time. In our multiprocessor the network is a shared resource. Contention leads to a difference in throughput of at most 1%, for our channel equalizer.

References

[1] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on signal processing*, 44(2):397–408, February 1996.

[2] F. Commoner, A. Holt, S. Even, and A. Pnueli. Marked directed graphs. *Journal of Computer and System Sciences*, 1971.

[3] A. Donlin. Transaction level modeling: flows and use models. In *CODES+ISSS '04: Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 75–80, New York, NY, USA, 2004. ACM Press.

[4] M. Geilen et al. Object-oriented modelling and specification using she. *Journal of Computer Languages, special issue for VFM'99*, 27(1-3), April-October 2001.

[5] A. Ghamarian *et al*. Throughput analysis of synchronous data flow graphs. In *Sixth International Conference on Application of Concurrency to System Design (ACSD)*, 2006.

[6] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):21–31, Sept-Oct 2005.

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

[8] A. Moonen, M. Bekooij, and J. van Meerbergen. Timing analysis model for network based multiprocessor systems. In *ProRISC, 15th annual Workshop of Circuits, System and Signal Processing*, pages 91 – 99, Veldhoven, The Netherlands, November 2004.

[9] A. Moonen, R. v. d. Berg, M. Bekooij, H. Bhullar, and J. v. Meerbergen. A multi-core architecture for in-car digital entertainment. In *In proceedings of the GSPx Conference, Santa Clara, California USA*, October 24 - 27 2005.

[10] T. Parks, J. Pino, and E. Lee. A comparison of synchronous and cycle-static dataflow. *29th Asilomar Conference on Signals, Systems and Computers*, 1995.

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

[12] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva*, May 2000.