Control and Communication Performance Analysis of Embedded DSP Systems in the MASIC Methodology

Abhijit K. Deb, Johnny Öberg, Axel Jantsch
Department of Microelectronics and Information Technology
Royal Institute of Technology
Electrum 229, 164 40 Kista, Sweden
Tel: +468 752 1115, +468 752 1305, +468 752 1221
Email: abhijit|johnny|axel@ele.kth.se

ABSTRACT
The time required by an embedded system to process information does not only depend on the amount of data; it also depends heavily on the synchronization overhead, bus protocol and communication architecture. This paper presents a technique to estimate the performance of the control and communication part of an embedded system modeled using the MASIC methodology. A key concept in MASIC is the strict separation of the computation part from the control and communication. Based on this clear separation of concerns the estimator analyzes the communication delay. Our method targets applications with intense, but regular data flow with a fair amount of complex control. Examples are base stations or mobile terminals. For these applications the method allows a cycle accurate estimation of the delay due to the communication. Hence, different architectures can be evaluated and e.g. the effect of different bus arbitration and different DMA block sizes can be assessed. Thus, the proposed method is an aid for the designer to explore the system with different system level decisions.

Keywords
Performance analysis, Grammar, Embedded DSP systems.

1. INTRODUCTION
As the modeling of an embedded DSP system is refined to an implementation phase, typically a bus-based architecture emerges where the system bus provides the communication between the DSP blocks and accesses the shared memory. Since the bus is a shared communication channel, it requires arbitration. Therefore, an embedded core interested in transmitting data first needs to wait to handshake with the arbiter which results in a synchronization overhead. Hence it is important for the performance analysis technique to consider this overhead [3]. The bus protocol and SoC communication architecture also affects the system performance. Effects of shared memory access and DMA block size on the performance of HW/SW systems, where system operation is dominated by memory accesses, have been reported in [5]. Ignoring these effects caused by control and communication would result in inaccurate performance estimates, which in turn would lead to non-optimal design decisions. Simulation of the RTL models of embedded core with the host environment gives accurate estimates. However, RT level is too detailed to be efficient enough during exploration of the design space. Moreover, detailed RTL hardware models of embedded processors are often not available.

The performance estimation technique presented in this paper considers the delay caused by the control and communication part of the MASIC model of an embedded system. Performance often refers to both the *numbers of clock cycles required for a particular implementation* and the *period of the clock cycle needed to meet the constraints*. Our work deals with the former aspect, while the later is to be dealt with at a lower level of abstraction. The MASIC, *Maths to ASIC*, methodology [8-10] models an embedded system as a set of communicating FSMs. The communication and synchronization among the cores give rise to a complex architecture and control scheme that we refer to as the GLObal Control Configuration and Timing (GLOCCT). The estimator takes the MASIC description of the communicating FSMs, replaces the nodes of the GLOCCT FSM by varying weighted arcs, and builds an internal representation of the system known as the Delay Graph (DG). To analyze the system performance, the DG is simulated with input events at different time instances. Though the data flow rate may vary among different parts of the system, the rate is fixed for each particular path. Hence the arrival of the events, that is, the set of input vectors to the DG can be deduced accurately. The estimator applies the input events on the DG and computes the following information:

- *Bus usage* – shows the amount of time each resource is occupying the bus.
- *Component execution trace* – includes the time a component needs to wait to get access to the shared resources and the time to finish its communication.
- *System performance* – gives the total time that the system needs to consume a certain amount of data using user given system attributes.

Before describing the proposed technique, we present few related works in the next section. Section 3 briefly describes the MASIC methodology of embedded DSP systems design. Next, we

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISSS '01, October 1-3, 2001, Montréal, Québec, Canada.
Copyright 2001 ACM 1-58113-418-5/01/00010...$5.00.

illustrate the proposed performance estimation technique in section 4. The following section shows the results computed from the LPC speech coding example. Finally, in section 6, we conclude the work presented in this paper.

2. RELATED WORK

A number of system performance analysis techniques were proposed in the early 90s that targeted the behavioral specification containing mutually exclusive paths due to the presence of conditionals. It was assumed that the branches of an if-then-else have an equal probability of execution. Bhattacharya et. al. have proposed a Markov chain based technique that takes care of the fact that, the branch probability may be unequal [1]. They have derived the branch probability and the state transition probability to calculate the expected number of clock cycles needed for a schedule. A Control Flow Graph (CFG) model based on a homogeneous Markov chain has been reported in [2]. They have used a probabilistic finite state machine to model the schedule and evaluate the effectiveness. Since scheduling of the CFG is mainly a path based approach, they have used scheduled paths from the CFG and constructed an FSM corresponding to the resulting schedule.

However, the approaches described above do not take the synchronization overhead into account. In many real life systems like telecommunication, networking applications, embedded control applications, multimedia applications, etc. the system is described as a set of concurrent communicating processes. Though the clock period estimation of a system is not affected whether the system is described and implemented as a single process or as a set of communicating processes, the number of required clock cycle estimation is affected. Hence a synchronization graph based approach is reported where the synchronization overhead is considered using a static performance analysis technique [3]. They begin by constructing a synchronization graph corresponding to a set of communicating processes. Later on, the loops in the graph are divided into different communication layers. Analysis of a single communication layer results in the estimate of a system with multi layer communication. The technique statically calculates the worst case performance of a system of communicating processes.

The Bus and Synchronization Event (BSE) graph was introduced to evaluate the performance of a bus-based SoC architecture [4]. The authors showed the effects of bus architecture and bus protocol on the system performance. They used a dynamic estimation approach, which is split into two phases. Initially, the system specification is partitioned into HW and SW, and a cosimulation is performed where the communication between components is modeled in an abstract manner. Secondly, from the initial cosimulation, they extracted a set of computation and communication traces for each of the components that was known as the BSE graph. Our proposed performance analysis technique does not require an initial cosimulation. The estimator takes the grammar based MASIC notation of the system to evaluate the performance. Before we describe the proposed technique, the following section gives a brief overview of the MASIC methodology.

3. THE MASIC METHODOLOGY

Even if the functionality of a DSP system remains the same, the architecture may need to be amended to be able to deal with different product families or HW/SW combination. Furthermore, it requires being configurable to be able to work with next generations of products. All these requirements suggest that the functionality be kept separated from the architecture and communication. As such, design methodologies have been proposed to separate these two aspects [6,7]. The MASIC methodology uses grammar notations to capture a design at a higher level of abstraction and separates the functionality from the architecture and communication, which represents the GLOCCT of the system [8,9]. The MASIC design flow is shown in Figure 1. Modeling in MASIC begins at the functional level where the algorithm development and verification is concerned with making sure that the specified signal processing figures of merit are met. The output of this phase is a set of DSP functions in C, which represents the data flow part of the system model. The GLOOCT is expressed in the MASIC grammar notations in which the grammar rules describe the communication and synchronization, and the constraints to the rules manifest the architectural decisions. The architectural decisions characterize the chip level architecture that includes processor cores, busses, memories, external interfaces and sharing of resources. The MASIC compiler reads the system model written using the MASIC notations and generates a VHDL description that links to the C functions.

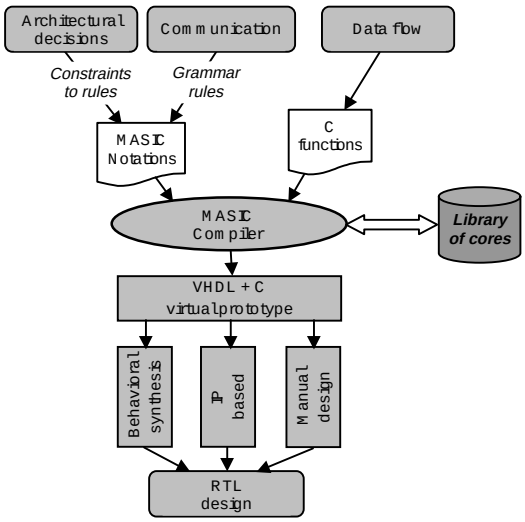

Figure 1: MASIC design flow

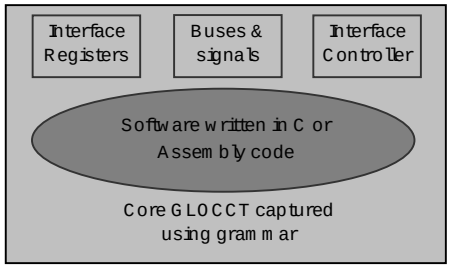

Figure 2: MASIC model of the core

Embedded systems are often modeled using a set of communicating FSMs [10,11]. MASIC uses grammar notations to describe the communicating FSMs and employs a library to manipulate the cores. An embedded core is viewed at two different layers as shown in Figure 2. The primary intention of the

model is to focus on the interface between the host and the core. The outer layer provides the interface of the core to the system GLOCCT. This layer includes peripheral registers, buses and the associated control logic, which is referred to as core level GLOCCT. The inner layer contains the software in C, which is a DSP function developed in the functional modeling phase of the design. The GLOCCT provides the mean to transfer data among the cores while the C function performs the DSP computation when the data is transferred [10]. For a future implementation phase, the C-function is compiled for the target processor and elaborated over system clock cycles.

4. PERFORMANCE ANALYSIS TECHNIQUE

4.1 A Pair of Communicating FSMs

Let us consider a core wishing to read a memory using a bus. The system is built using the asynchronous bus protocol described in the Figure 3. The choice of asynchronous protocol stems from the fact that the target system will have a number of cores connected to the system bus and they might run on different clock speeds and have different response times. The operation sequence is initiated with the arrival of an IO read signal, which creates the arc between states *C1* and *C2*. Here, we use the symbol *C1→C2* to represent the arc between nodes *C1* and *C2*. With this arc the core sends `rdReq` signal to the memory. At the same time, it places the memory address on the multiplexed bus. However, the estimator does not account for data assignment statements, like putting address or data on the bus. It only considers the synchronization signals, which generate events and synchronize the system. Hence the estimator does not need to perform any exhaustive computation to find the performance figures.

The memory reacts to the `rdReq` signal by asserting the `Ack` and reads the address from the bus, which is shown in `M1→M2`. The core sees the acknowledgement and drops the `rdReq` and relinquishes the bus, in *C2→C3*. With the remaining steps, the word is read from the memory and the cycle is completed when the core finally drops the `Ack` signal. The MASIC notations are used to describe the control steps of the above model. The total time needed to complete the whole operation can be estimated from the time required in each of these steps.

The total time needed to complete the cycle is the summation of time elapsed in the nodes and time elapsed in the arcs. We target the Mealy type implementation with a clocked output stage. Hence all the arcs of the FSM will take one clock cycle, if not annotated with a memory access time information as discussed below. The time taken by a node depends on a signal from the communicated process. If the desired signal is already present, the machine does not wait in the node and the wait time in the node is zero. Otherwise, it waits in the node until it receives the signal. Depending on the reaction time of the communicated process, we replace a node with a weighted arc, which gives the wait time in the node.

Let us apply these observations on the model of the Figure 3. If we do not consider the initial wait time, the core FSM starts with the *C1→C2*, which requires one cycle. The wait time in the node *C2* depends on *M1→M2*, which is another cycle. After one more cycle it crosses the *C2→C3* and waits in node the *C3* for *M2→M3* be completed. Since this arc is related to memory access we cannot apply the general rule of one cycle for an arc. In this

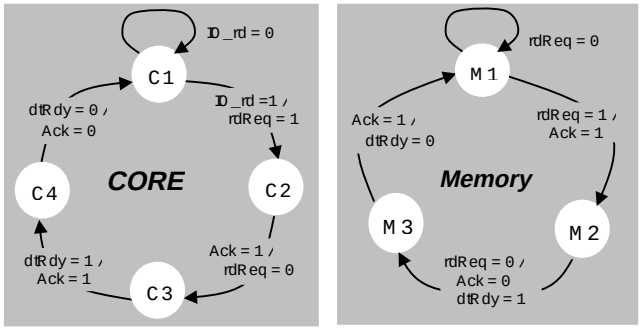

Figure 3: Communicating FSMs in MASIC

sequence of operations, the memory receives the address during *M1→M2*. Therefore, the wait time in the node *C3* can be estimated as: ***max (roof(memory_access_time/clock_period), 2)***. The MASIC description does not contain any information about memory access time. This information is given to the estimator. The rest of the arcs take one cycle each. Thus the core will take seven cycles to perform the operation, if memory access time is less than or equal to two clock cycles.

So far, the wait time in a node corresponds to an arc in the communicated FSM. The scenario changes when several of them communicates with each other. We model such a situation by creating a DG of the system.

4.2 Systems with Multiple FSMs

To be able to connect multiple cores to the system bus we need to introduce a bus arbiter, which ensures that only one core has control over the bus at a particular time. Thus a core wishing to transfer data using the bus needs to wait to handshake with the arbiter. The GLOCCT provides the bus arbitration, which could be of any kind. As an example, let us describe a round robin arbitration scheme where the GLOCCT sequentially reads the request from each core to grant the bus and moves to the next one as shown in the Figure 4. The read operation still works in the same way as described in Figure 3, with the exception that it incorporates a wait state labeled as *C2* to synchronize with the GLOCCT. The time spent in this state shows the wait time in the performance analysis. Along with the read cycle, now the core FSM also includes the memory write cycle. With the arrival of an `IO_wr`, instead of generating a memory `wrReq`, the core generates a request to the GLOCCT and waits in node *C6*. To get an estimate of this system we need to consider the GLOCCT input events associated with a time instance and generate the resulting system response.

Let us consider that the arbiter begins at time `t` to react to the input events shown in Figure 5. At time `t`, there is no request from core-1 though core-2 wishes to write to the memory. Hence, the GLOCCT does not wait in node *G1* and the estimator replaces node *G1* with an arc of weight zero to build the DG. It would, however, need a clock for the *G1→G3*. A request from core-2 has been waiting since time `t` and this request brings the system from *G3* at `t+1` to *G4* at `t+2`. At this time `Grant2` signal also appears. It communicates with two different nodes of the core-2 as shown by the dotted arrows in Figure 4. Since Core-2 was waiting in node *C6* the path *C6→C7* is chosen that in turn communicates with the memory. From this point the core and the memory perform a set of operations, similar to the previously described read cycle. These are shown by the dotted arrows in the Figure 4. The

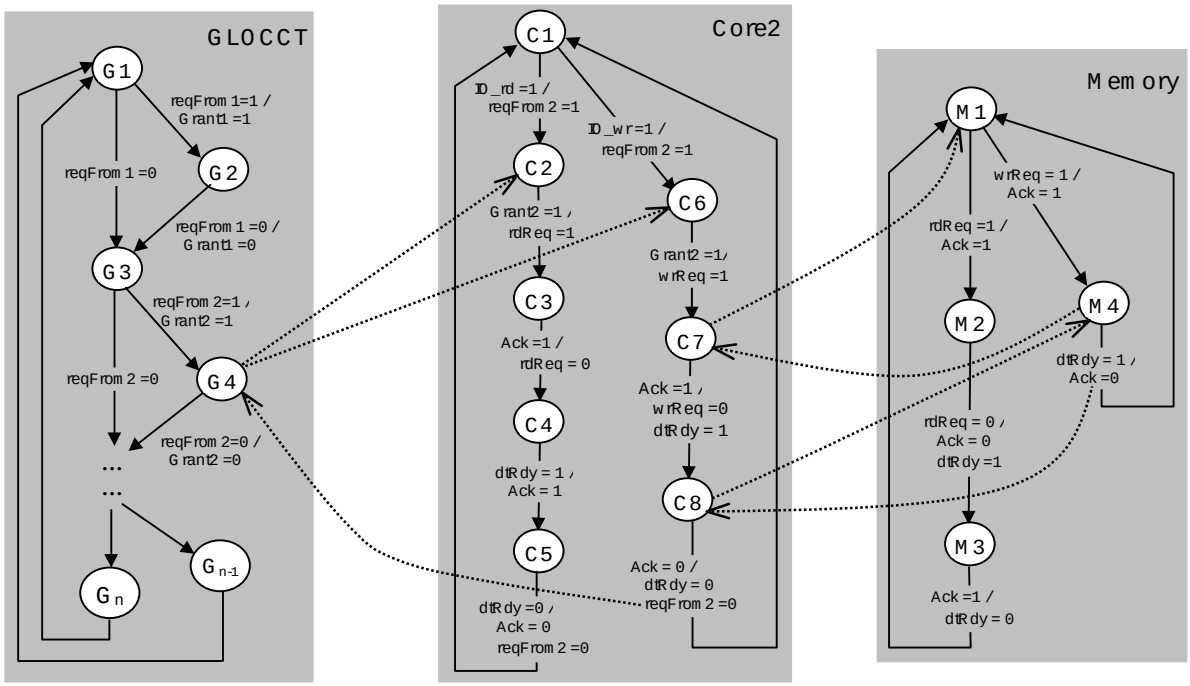

Figure 4: Effect of a set of communicating FSMs

operation finishes when the core2 reaches *C1* after five clock cycles. A read cycle, alternatively, would require seven cycles to traverse from node *C2* to *C1*. Hence, a node might have different wait time depending on the operation. To build the DG, we replace node *G4* with an arc of weight five for write cycle and weight seven for read cycle. On the other hand, a node might have a weight of zero, as was the case with node *G1*. Depending on the input event one of the weighted arcs is chosen, which gives the time elapsed in the node. The arcs of the FSM are not replaced in the DG and they correspond to a time of one clock period, unless annotated with memory access information as explained in the previous sub section. In this way, the effects of all the communicating FSMs are mapped together in the DG. The DG is simulated with the input events at different time instances to get the performance figures.

4.3 Tracing the Resource Execution

The DG of the system is the basis of the estimator. A start up sequence gives the initial state of the system. From that point the estimator simulates the DG of the system with the input events and fills up the rows of the Execution Trace Table (ETT) with the output response as shown in the Figure 5.

Let us assume that the GLOCCT in Figure 4 begins at time *t* with state *G1*. It receives input events *ReqFrom1* and *ReqFrom2* at time *t* as shown in the Figure 5. Since *ReqFrom1* is low, the FSM moves through *G1→G3*. Hence, we put state *G3* at the time *t+1* and keep the input event *ReqFrom2* in input event register. The length of the input event register is one. The sender can rewrite the register. For instance, in the event of a timed out request the sender broadcasts a zero, which changes the content of the input event register.

The *ReqFrom2* is read from the register at time *t+1* that causes the machine to move to G4. Hence, we put core-2 in *G4* at time *t+2*. In addition we also generate the *Grant2,* as it is a synchronizing signal. For a write cycle, we know from the discussion in the previous sub section that the arbiter stays in *G4* for five cycles after the grant is made. Hence we put the GLOCCT in *G4* from time *t+3* to *t+7*, at the end of which we know that the *ReqFrom2* will be dropped low. Therefore, the input event register is made zero at *t+7*, which causes the machine to move to *G5* and drop the *Grant2* signal at time *t+8*. The ETT shows that the event *ReqFrom2* waits for one cycle and takes seven cycles to be processed.

The ETT grows with time as it receives more input events and generates corresponding system responses. If the same pattern of communication and data transfer are repeated over and over then it is enough to simulate the DG with input events corresponding to

Time	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8
State of GLOCCT	G1	G3	G4	G4	G4	G4	G4	G4	G5
Grant1		0	0	0	0	0	0	0	0
Grant2		0	1	1	1	1	1	1	0
ReqFrom1	0	0	0	0	0	0	0	0	0
ReqFrom2	1	1	1	1	1	1	1	0	0

Figure 5: Execution trace table

one frame. Though the technique is explained using a round robin arbiter, it is not restricted to any particular type of implementation. To change the arbitration scheme to a priority arbiter, for instance, we need to change the MASIC description of the system. In that case the arbiter would read all the requests from the cores at a time and assign the system bus to a core using a preset priority.

Until now, we have only explained a single word of data transfer over the bus. It will be efficient to use a DMA mode transfer to shift a block of data, because we do not need to send the address with all the data words. To model a block data transfer the core and memory FSMs include DMA mood transfer. In that case, the DG is built by replacing the nodes of the GLOCCT FSM by a varying weighted arc that depends on the size of the DMA transfer. For a system with multiple clocks we consider the system clock, which is typically a multiple of the slowest clock in the system. When the GLOCCT communicates with a slower process the estimator updates the value of the weighted arcs using the ratio between the clock speeds. Bus width also affects the system performance. If, for example, we use a 16 bit bus to transfer a 32 bit word then we change the core and memory FSM to include the extra steps needed. To show the effect of the architectural change on system performance, the estimator builds a DG from the modified MASIC model and the ETT shows a longer response time for an input event.

5. RESULTS

In this section we apply the described performance analysis technique on the Linear Predictive Coding (LPC) speech processing system. We shall model the system with different system level decisions and investigate their effects on the system performance. A 20 ms frame of speech data is buffered at a sampling rate of 8 kHz. The system takes a frame of 160 samples, which corresponds to a 20 ms frame and performs the Hamming windowing operation on the samples of the speech frame. The result is stored in the memory. Next the autocorrelation block reads the 160 samples stored by the Hamming block and computes 50 autocorrelation lags. Finally, the LPC block takes these values and compute 10 coefficients and a pitch period, which is stored back in the memory. We decided to perform these operations on three different cores, which are connected to the shared memory by a multiplexed bus. The speech processing system works in a globally pipelined manner where each of the blocks works on the set of data computed by the previous block in

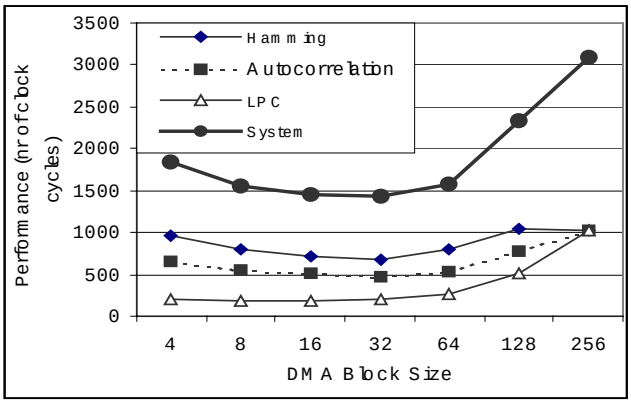

Figure 6: Effect of DMA size on performance

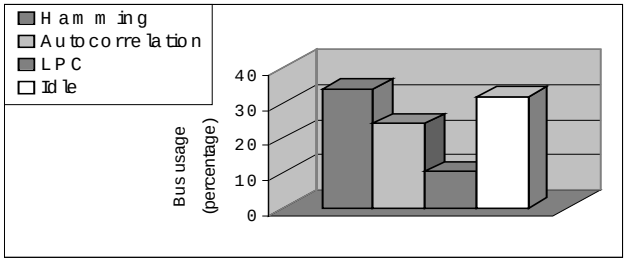

Figure 7: Bus usage statistics

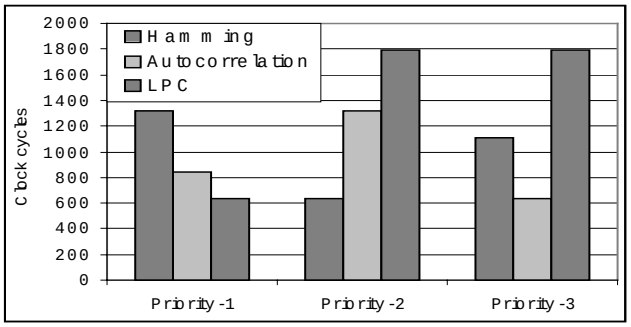

Figure 8 : Computation time available under different priority schemes

the earlier frame. The interface of the core is captured in the MASIC notation and the computation is performed using C function, as described in section 3. The GLOOCT provides the interface of the system, and communication and synchronization between the cores and the memory.

The estimator takes the MASIC description and builds the DG of the system. Since the same pattern of data communication is performed over every 20 ms frame, we simulated the DG with the input events corresponding to a single speech frame. First we studied the effects of different realistic DMA block sizes on the system performance as shown in the Figure 6. We considered different sizes starting from 4 to 256. For each of the cases, same input events are applied on the DG to construct the ETT, which shows total cycles needed to consume the input events. As expected, the performance increases with the increase in block size. However, the performance of the LPC block starts to deteriorate after 16 because it outputs only 11 words. On the other hand, the Hamming block deteriorates after 64, because it operates on 160 words and a bigger block causes some non-meaningful words in the last block. Figure 6 shows that the overall system performance is optimum with a DMA size of 32 words. Since we know the optimum size of the blocks, we use this size to check the bus usage figures. Figure 7 shows the percentage of bus usage by different blocks that include both reading data from the memory and writing the results back to the memory. The idle time shown in the figure is calculated assuming a budget of 2000 cycles.

The bus usage figures show the amount of time required by each of the DSP blocks within each speech frame. These figures help the designer to assign the priority to different blocks. The block with a higher priority receives its data first and gets more time to finish its task. Hence, we experimented with different priority given to different blocks. Figure 8 shows the computation time available to each of the blocks in three different priority schemes assuming a budget of 2000 cycles. These figures help the designer

to assign priority to individual blocks so that the computation timing requirements are met.

6. CONCLUSION

This paper presents a performance estimation technique that has been integrated in the MASIC environment. The technique targets applications with predictable data rates. The estimator works on the higher abstraction level of MASIC models. It does not perform any computationally exhaustive method as would have been needed at a lower level. Despite working on the higher level model, it gives an accurate estimate, which considers the synchronization overhead, effects of shared memory access and memory access time. Experimental results have been presented, which show the effects of different system level decisions on system performance. Thus it helps the designer to probe the model from a higher level of design abstraction by assisting the designer to optimize the model with different system level decisions. For instance, making a dedicated link between the two most frequently communicating components would lessen the traffic on system bus. Changing the arbitration scheme or DMA block size might result in less wait time for a component.

7. REFERENCES

[1] S. Bhattacharya, S. Dey, and F. Brglez, "Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications". *In 31st Design Automation Conf*, pp. 491-496, June 1994.

[2] M. Rahmouni and A.A. Jerraya, "Formulation and evaluation of scheduling techniques for control flow graphs". *In the Proc. of European Design Automation Conference*, pp. 386-391, Sep. 1995.

[3] S. Dey and S. Bommu, "Performance analysis of a system of communicating processes". *In the IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 590-597, 1997.

[4] K. Lahiri, A. Raghunathan and S. Dey, "Fast performance analysis of bus-based system-on-chip communication architectures". *In IEEE/ACM Int. Conf. on Computer-Aided Design*, pp. 566-572, 1999.

[5] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, A. Sangiovanni-Vincentelli, "A Case Study on Modeling Shared Memory Access Effects during Performance Analysis of HW/SW Systems". *In Proc. of Int. Workshop on Hardware-Software Codesign*, pp. 117-121, Seattle, USA, March 1998.

[6] J.A. Rowson and A. Sangiovanni-Vincentelli, "Interface Based Design". *In the Proc. of Design Automation Conf.*, pp. 178-183, June 1997.

[7] C.K. Lennard, P. Schaumont, G. Jong, A. Haverinen and P. Hardee, "Standards for system-level design: practical reality or solution in search of a question?" *In Proc. Design, Automation and Test in Europe Conf.* pp. 576–583, Paris, France. 2000.

[8] A.Hemani, Abhijit. K. Deb, J.Öberg, A.Postula, D.Lindqvist and B. Fjellborg, "System Level Virtual Prototyping of DSP SoCs Using Grammar Based Approach". *Design Automation for Embedded Systems*, Vol.5, No.3, August 2000, pp.295-311.

[9] Abhijit K. Deb, A. Hemani, and J.Öberg, "A Heterogeneous Modeling Environment of DSP Systems Using Grammar Based Approach". In the Proc. of 3rd Forum on Design Languages, pp. 365-370, Tubingen, Germany, Sep. 2000.

[10] Abhijit K. Deb, A. Hemani, J. Öberg, A. Postula, and D. Lindqvist, "Hardware software codesign of DSP systems using grammar based approach". *In Proc. of the 14th Int. Conf. on VLSI Design*, pp.42-47, Bangalore, India, January 2001.

[11] M. Chiodo, P. Giusto, A. Jurecska, H.C. Hsieh, A. Sangiovanni-Vincentelli, L. Lavagno, "Hardware-software codesign of embedded systems". *IEEE Micro*, vol. 14 4, pp. 26-36, Aug. 1994.