An Approach to Mixed Systems Co-Synthesis

Thomas Benner and Rolf Ernst
Institute of Computer Engineering, Technical University of Braunschweig
P.O. Box 3329, D-38023 Braunschweig, Germany
{benner | ernst}@ida.ing.tu-bs.de

Abstract

The paper presents an extension of co-synthesis for data dominated applications to include reactive processes. The extension allows for rate constraints as used in data dominated applications as well as minimum and maximum time constraints for communication and I/O which is required to define reactive behavior of control tasks. A co-synthesis approach is proposed which differentiates global process and communication scheduling, which is non preemptive, and local scheduling which includes a restricted interrupt controlled process invocation to extend the design space. Several user parameters allow design space exploration. The approach includes buffering, process pipelining and parallelization for control as well as for data dominated tasks on different levels of granularity. It supports inter process time constraints which span processes with different periods. The target architectures are heterogeneous systems consisting of multiple processors, hardware components, memories and different types of communication media.

1. Introduction

Despite many improvements in co-synthesis during the past years, the range of application and the target architectures of the individual approaches are still very limited and prevent design space exploration for more complex architectures. There is work in reactive, control dominated systems, such as [1, 2, 3, 4] which covers single microcontroller-coprocessor architectures typically shared memory or point-to-point interconnect and work in data dominated or computation oriented systems, such as [5, 6] which use single processor-coprocessor architectures or relatively simple communication structures, such as a single bus. There are also less specific systems, such as SpecSyn [7], COSYMA [8] or VULCAN [9], even though they are certainly biased towards control [9] or data dominated applications [8].

Many applications, such as mobile communication, however, show a mixture of both control and data dominated applications on different levels of a design often included in a single task, such as a wireless communication channel. Data dominated applications, in particular periodic applications in signal processing and control engineering applications make extensive use of buffering to increase the design space and allow high performance pipelining of statements up to processes. A well developed mathematical background of signal flow graphs and synchronous data flow graphs (SDF) supports scheduling and buffer size optimization, at least for cyclo-static implementations. There is a host of work in this area and we just want to refer to [10] which can be seen as a basis for much of the later work. On the other hand, the current control dominated co-synthesis approaches which focus on relatively small systems with processor-coprocessor architectures make little or no use of buffering and pipelining. [11] covers more complex heterogeneous multiprocessor architectures. They have presented several heuristic allocation and static or dynamic scheduling algorithms for global system optimization. Again, the approaches do not include buffering. This paper presents a co-synthesis approach for mixed systems and heterogeneous architectures.

2. Process and timing model

We use a simple yet instructive example to explain the process and timing model as well as to demonstrate the problem of mixed application co-design and the impact on scheduling in the context of co-synthesis. The system in fig. 1 is a simplified model of a remote motor control. It receives packeted messages from a central controller over a specialized simple field bus. The messages are encoded and any error condition must be signaled on the bus after a maximum time of $t_{IO1,max}$ after the error has been detected. Process P_1 is responsible for interface control. When complete, P_2 will process the message and decide whether the motor control must be adjusted. P_1 and P_2 are control tasks reacting to messages which arrive asynchronously at some unknown time. So, a bus message causes a chain of events. But, as an additional constraint, both must be fast enough not to miss any message on the bus. So, there is a minimum process instantiation rate which can be represented as a maximum time $t_{1,min}$ from the occurrence of a bus event I_1 to the execution of the corresponding P_1. If $t_{P1ex} > t_{IO1,max}$, where t_{P1ex} is the execution time of P_1, then P_1 must be pipelined or mapped to different processing elements (PEs), which execute concurrently. P_2 must be fast enough to process P_1's messages. Since P_1 will only send

0-8186-7895-X/97 $10.00

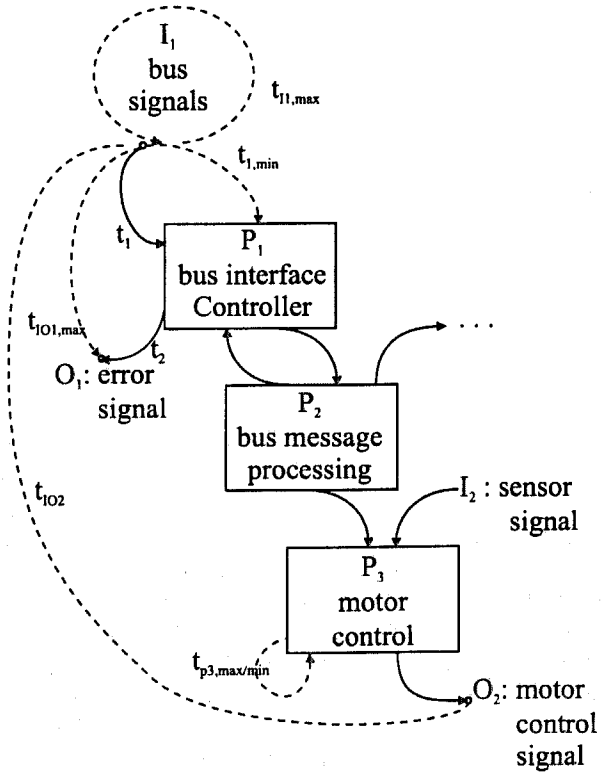

Figure 1. A remote motor control.

complete messages while receiving individual bus signals, its instantiation interval is smaller. The motor controller P_3 uses a complex control algorithm which must be iterated periodically within bounds $t_{rate3,min/max}$. It samples sensor signals and reads messages from P_2 and outputs control signals. P_3 is a data dominated application with a static data flow, except for the adjustment of control parameters in reaction to P_2 messages, which would lead to a conditional control flow in a part of P_3.

Finally, A maximum latency time t_{I1O2} defines a new reactive chain P_1, P_2, P_3. Now, P_3 must react to events as well as execute a data dominated task. Moreover, the chain is conditional. P_1 only passes a message to I_1 if the message is correct and addressed to this system, P_2 only communicates to P_3 if the control parameters shall change.

In a standard single processor architecture, we would probably use an operating system kernel with preemptive scheduling and input and output buffers to assure the required input and output sampling. As soon as the timing becomes too tight, we might decide that P_1 must be moved to hardware or try to use a specialized controller.

But, the general co-design space is much larger. In the example, we could avoid using specialized hardware by a pipelined TPU-type peripheral processor or a second parallel processor, but then we must regard P_1's internal states, which leads to feedback loops. In this case, process level allocation is inappropriate and the process itself must be analyzed and split according to the feedback. Scheduling for (cyclo-)static data flow systems [10] includes mapping of consecutive process iterations, or, on the lower level of granularity, of consecutive loop iterations to different PEs. We do not want to discuss the associated problems e.g. of loop parallelization [12], but want to support scheduling in this context. Whether we need to pipeline or parallelize a process or even a process chain very much depends on the wide variation of PEs and communication structures and timing requirements. This makes scheduling in heterogeneous hardware/software architectures harder than in real-time programming applications where hardware/software partitioning has already been done upfront thus limiting the design space and the feasible timing requirements.

Under these extended co-design parameters, global preemptive scheduling becomes a difficult task due to buffering and process pipelining. Also, preemptive scheduling is only appropriate for certain architectures as it involves control and timing overhead.

Therefore, we propose a non preemptive global scheduling approach which is described in the next sections. We start with a problem formulation for the new approach. Then, we explain the solution which is a combined partitioning and scheduling approach. Several parameters can be used for design space exploration.

3. Local scheduling

First, we would like to classify the PEs and local scheduling strategies with respect to global I/O constraints and process characteristics. The first characteristic refers to single or multiple processes mapped to a PE.

- single process per PE - hardware
  for purely control dominated systems without complex data operations, RT-level FSMs are often sufficient to implement processes. In this case, the processes can react instantaneously: $t_{pi,min} = t_{pv,min} < t_{cyc}$, where t_{cyc} is the FSM cycle time, t_{pi} is the process instantiation time, and t_{pv} is the process instantiation interval. This model is the basis for most co-synthesis approaches in the control domain, where hardware processes are mapped to individual FSMs [1, 4, 3]. If there are more complex operations on data, which take several clock cycles, operation scheduling is needed and the FSM transition time increases while instantiation is still spontaneous as long as the FSM is not busy in transition. This leads to an increasing process instantiation interval: $t_{pv,min} = n * t_{cyc}$. We can use pipelining to reduce the instantiation interval, even in case of purely control dominated functions (e.g. [13, 14]). In both cases, the target model of computation is event driven. In contrast, a data flow application would be executed periodically. Here, it is required that $t_{pe,max} < t_{pv,min}$,

where t_{pe} is the period.

- single process per PE - software:
  if only a single reactive process is running on a processor, we can either poll or use an interrupt. Here, the target model of computation is event driven like in hardware above and the same arguments hold, except that t_{pi} and t_{pv} will typically be larger. Whether interrupt or polling leads to a shorter t_{pi} depends on the application and is architecture dependent.

- multiple processes per PE:
  If we assume that a PE has a single control flow, local scheduling is required for both hardware and software. Different strategies may be used for the various PEs depending on the processes mapped to the PE. While periodic scheduling is well suited to data flow tasks with (cyclo-)static data flow such as the motor control in fig. 1, it can be less efficient for reactive processes. In fig. 1 this is obvious, when $t_{I1O1,max} < t_{I1,max}$. To support such cases, we use a static scheduling approach, but reserve part of the PE cycles for preemptions of a single user defined process, if this PE is a processor. Implementation details are given below. With this exception, local hardware and software scheduling can use the same approach.

The second characteristic distinguishes processes with static and with dynamic control and data flow. As an example, P_3 in fig. 1 must react to sporadic control information from P_2. This would be implemented with conditional control and data flow in P_3.

If the branches of a conditional construct in a process are split for fine grain partioning optimization, then scheduling should make use of the mutual exclusion of branches to avoid redundant assignment of execution time. If during optimization, the branches are mapped to different PEs, (additional) conditional data flow between PEs will result. In the current approach, we avoid this situation by not splitting the branches of a conditional construct, except for loops with fixed bounds. This, however, is a limitation of the current scheduling approach, which does not yet handle mutual exclusion, rather than a fundamental limitation of static non-preemptive scheduling.

4. Timing constraint transformation

The scheduling model requires timing constraint transformation. First, we must derive process periods for the reactive processes. These periods induce other periods along reactive process chains. The trouble in co-design is that the execution times and, as a consequence, the required period depends on the PE characteristics as we will see in the sequel.

For global static scheduling, we first replace the external event periods $t_{I,max}$ by rate constraints for the reactive processes. The P_1 period must at most be $t_{I1,min}$. If we can buffer a single event, it is sufficient to require $t_{pe1} < t_{Pr1,max} < t_{I1,min}$ to react to all input events. Since, now, P_1 reads the input events only in intervals of t_{p1} instead of instantaneously, latency increases with the resulting requirement $t_{Pr1,max} < t_{I1O1,max} - t_{exeP1,max}$.

In general, this holds for all input events I_j and interface processes P_i of a system:

$$\forall_{i,j,k} : t_{rPi,max} < Min(t_{Ij,min}, t_{IjOk,min} - t_{exePik}),$$

where t_{exePik} is the execution time from reading input I_1 to sending output O_k, where O_k is an I/O event.

Internal process periods can be derived from I/O constraints and the rate of other processes. In case of a single rate of all processes, it is an optimization problem to distribute the different I/O constraints over several inter process constraints [19] of a reactive process chain (e.g. P_1, P_2, P_3) since only the sum of all constraints must match the I/O constraint (e.g. $t_{I1O2,max}$).

The more general case is a multi-rate system. Communication between processes of different rates has not been included in previous work on scheduling in co-synthesis [11]. Again, fig. 1 shows the problem. Because I_1 and the messages on the bus appear asynchronously, we have no information on the time when a message is complete and will be sent to P_2. In other words, P_1 to P_2 communication is asynchronous, again. Like in case of t1, latency between P_1 and P_2 can increase up to a period t_{p2}, if $t_{p1}\ mod\ t_{p2} \neq 0$. If $t_{p2} = t_{p3}$, then P_2 and P_3 can be synchronized such that there is no additional increase in latency.

In general, each change in the process period between two processes will increase the latency of a reactive chain by the period of the receiving system. If a process can react spontaneously (see above) then there will be no increase in the latency time for this process (except for its execution time). On the other hand, the process period determines the system load. So, process rate selection would be another optimization problem. We still have to include buffering. To simplify latency analysis, we require unconditional read operations.

Now, we can sum up the overall latency time and conclude:

$$\begin{aligned}\forall_{i,k} t_{IiOk,max} > &\sum(\text{execution times}) + \sum(\text{buffer times}) \\ &+ \sum(\text{communication times}) \\ &+ \sum(\text{period adjustment times}).\end{aligned}$$

This also includes the single rate systems, where period adjustments are not necessary. This formula represents the period assignment problem as an optimization task. For the moment, the user is asked to assign the periods and the will then optimize schedule and hardware/software partitioning.

5. The co-synthesis environment

The co-synthesis environment imposes few constraints on partitioning and scheduling. The input language used to describe the system functionality is C^x, an extension of C with

a process construct. Communication between processes in the input description is based on logical channels which are objects accessed by C functions [16]. The basic concept is comparable to the approach in COSMOS [17] or in CoWare [18], except for the buffering description and the distinction between blocking and non-blocking communication. Time constraints may be rate constraints for processes or minimum and maximum time constraints between operations in the same process (intra process constraints) or in different processes (inter process constraints, see [19, 16]). Compared to [16], the set of time constraints is slightly extended to include I/O operations in order to cover the timing model explained above. The different types of timing constraints may be arbitrarily combined for any process and I/O operation.

The target architecture template which is provided by the user describes the physical communication channel type between the components and their maximum channel buffer capacity (if any), the component types, i.e. the memories, the processor components and their types, and the technology of application specific hardware components and their respective maximum number of function units. The latter is used to control hardware synthesis as in earlier versions of COSYMA [8]. The set of components and communication channels which is actually available for the co-synthesis process depends on the module library. Currently, this includes non buffered busses, point-to-point interconnect, FIFO buffers, and shared memory accessible over busses or double buffers.

6. Co-synthesis with combined partitioning and scheduling

The optimization goal in scheduling and partitioning is to implement the system with minimal application specific hardware cost, regarding all time constraints. The target architecture template is used as a constraint during co-synthesis and can be adapted by the user for design space exploration. The application specific hardware costs are estimated on a scheduling block basis with an approach that is based on earlier work in [20].

As mentioned, partitioning supports different levels of granularity. There are two reasons not to fix granularity. First, and already explained, finer granularity offers a higher optimization potential. Second, static non-preemptive scheduling can support shorter process periods and process instantiation times in case of process pipelining. It can also better utilize the PEs with finer granularity of the scheduling blocks provided the improved scheduling density compensates the increased context switch overhead. Here, we define a scheduling block to be the atomic unit for scheduling. On the other hand, finer granularity leads to an increased design space which makes optimization more complicated and possibly less efficient. So, until we have better knowledge on the appropriate selection of granularity, the user is asked to control granularity by either constraining the maximum input code size of a scheduling block or the maximum scheduling block execution time on a reference processor (determined with COSYMA timing analysis functions [15]).

Furthermore, scheduling blocks are delimited by blocking communication. This is a necessity for non-preemptive scheduling. When partitioning below the level of processes or functions, additional communication must be introduced which is not part of the input description. So, unlike approaches for interface generation as in [17, 18], the algorithm must be able to derive logic communication channels and communication operations with a data flow analysis when the system is partitioned. This problem is known from small system co-synthesis using basic block level partitioning e.g. [9, 8], except that, here, we have to cope with different physical communication media at the same time. We used the partitioning function which was already available in the COSYMA system and extended it to cover FIFOs and buses connecting multiple components.

Loops with fixed bounds are split into loops with smaller bounds to match the required maximum schedule block time or length [22]. This approach reduces the resulting code size compared to loop unrolling. Each process is divided into a set of scheduling blocks. For each of these scheduling blocks, a lower and an upper bound execution time bound is determined either via symbolic RT-level simulation [15] or by a path based estimator [21]. This is done for each component of the target architecture template.

The scheduling blocks of a process access common data, or, more precisely, intra process communication is implicit via shared memory. If, during scheduling, part of the scheduling blocks are moved to a different PE, explicit communication may become necessary. The same happens in case of process pipelining which requires multiple sets of process variables and explicit communication between pipeline stages with buffers. In such cases, communication processes (CP) are inserted which are generated from templates depending on the physical communication channel. CPs allocate two or three devices at a time, the sending PE, the communication device (CD) and the receiving PE. In case of a non blocking communication, the CP is allocated only to the CD and to the sending or the receiving PE. We must define a period in which the static schedule is repeated. For synchronization with the process periods, it is sufficient to choose the LCM of all process periods.

Partitioning and scheduling is based on simulated annealing (SA) and is now executed in the following steps:

1. All processes are allocated to a single processor preserving all precedences, i.e. without pipelining, but disregarding timing constraints.

2. Iterate with SA over the following steps

   - randomly select a process P to be moved
   - randomly select a processor PR to be moved to

- calculate the mobility interval $[M_s(P), M_e(P)]$: in which the P can be placed without precedence violations.
- randomly select a available time slot on PR within the mobility interval
- update communication
- for each redundant communication: delete communication processes.
- for each new communication: random selection of available communication slots on the required CD and instantiation of communication processes CP.
- accept or reject move based on cost function C.

It should be noted that communication slot assignment is required when busses are shared for different logic communication channels between processes. This is the case in all experiments presented later. Simulated annealing has been used before for a combined scheduling and partitioning in co-synthesis [6]. There, the optimization goal was acceleration rather than meeting a complex set of interrelated time constraints between processes with reactive and periodic models of computation. Also, the target architecture was constrained to a single processor and several coprocessors communicating over a single bus and pipelining or buffering was not included. Nevertheless, it gave an incentive to try the combination on this more complex problem [23].

The cost function definition is a major problem in the approach since there are many conflicting constraints. The basic approach is an extension of [8], i.e. constraint violations are mapped to cost components with an exponential weight, such that we can start with a solution where all processes are allocated to a single PE, which would otherwise be invalid, and such that there is a high probability to accept moves which reduce constraint violations and eventually lead to a valid solution of the original problem.

7. Results

In order to evaluate the approach, we investigated variants of two different examples. The first one is a model train control. The train is controlled by a personal computer, from which messages are transferred over the rails. The model train has a motion speed regulation that moves the train close to original large trains. A decoder scans the pulse duration on the rails. A low pass filter and a 3 bit burst error correction code are used. The message is passed to a motor controller which samples the speed, motor voltage and current and controls the motor via a peripheral PWM circuit. This motor controller must be executed with a fixed period. Even though this sounds like a simple application, it has all the characteristics of a mixed system and is the basis for the example in fig. 1, where part of the processes are shown. We tried two variants with different transfer rates, each of them consists of

	P	G	8051	Sparc	Gates	t(s)
Train A	n	max		1	26200	28
Train A	y	max		1	27212	36
Train A	y	9us		1	22279	392
Train A	n	max	1		43629	30
Train A	y	max	1		43629	34
Train A	y	9us	1		38677	664
Train B	y	max		1	4623	40
Train B	n	9us		1	7128	88
Train B	y	9us	1		40876	11
Train B	n	9us	1		40101	519

Table 1. The train examples.

	P	8051	Sparc	Gates	t(s)
Bridge A	n	1		74195	42
Bridge A	y	4		0	133
Bridge A	y		1	0	40
Bridge A	y	1		65658	123
Bridge B	y		1	20626	47
Bridge B	n	1		76450	91
Bridge B	n	1	1	19861	44
Bridge B	y	1	1	9204	52

Table 2. The bridge examples.

5 processes with the cycle times shown in Table 1. The target architecture consists of a 8051 or a SPARC processor and an application specific coprocessor. The processor core and the coprocessor are running in parallel with a buffered point-to-point communication, which is scheduled by the optimization tool. In Table 1 we compare different implementation alternatives, which differ in its granularity (G), the use of process pipelining (P) and the target architecture. Column 2 specifies whether processes are pipelined. The parameter G gives the maximal execution time of a scheduling block, whereas the value max means, that a scheduling block is split only by communication and labels, which refer to timing constraints. The parameter G influences the number of nodes in the task graph. The columns 4-6 specify the amount of hardware resources. The type and number of processor cores and their communication is provided by the user, while the algorithm minimizes the application specific hardware (16 bit coprocessor). The execution time of the partitioning tool on an UltraSparc Workstation is given in the last column.

Code generation and context switching using a micro kernel is implemented with the approach described in [23] and the GNU C-compiler for the SPARC and a commercial C-compiler for the 8051. Since the approach is not yet integrated into the COSYMA system, we cannot provide the final costs as a result of high-level synthesis. Instead, the hardware costs are estimated with a path based estimation tool [24].

The effect of the granularity on the design quality can be investigated e.g. by alternatives in line 2 and 3. In both alternatives processes are pipelined. But the comparison with the design in line 1 shows an advantage of pipelining only for one with finer granularity. Here smaller scheduling blocks are mapped to hardware. The lower transfer rate of example Train B compared to Train A reduces the hardware overhead drastically for the SPARC core implementation, whereas it has a small effect on the 8051 solution, where most parts of the decoder and receiver task are mapped to the coprocessor, anyway.

The bridge (table 2) description consists of seven processes. In Bridge A the fastest process is executed with a cycle time of 50us, in Bridge B its cycle time is 100us. The A version can be mapped on a single SPARC processor as well as on four 8051 cores without additional hardware. For all mixed HW/SW solutions with an 8051 holds that most of the design is mapped to the coprocessor, whereas the coprocessor replaces up to three 8051 cores. The last variant is mapped to a SPARC, an 8051 and coprocessor. Significant advantage compared to alternative in line 4 can be reached if pipelining is used.

8. Conclusion

We have presented an approach to combined scheduling and hardware/software partitioning of mixed reactive and periodic systems. The results demonstrate viability and demonstrate the effects of design space exploration.

References

[1] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli. *Hardware-Software Codesign of Embedded Systems,* IEEE Micro, August 1994, pp.26-36.

[2] Pai Chou, Ross B. Ortega, Gaetano Borriello. *The Chinook Hardware/Software Co-Synthesis System,* in International Symposium on System Synthesis, Cannes, France, September 13-15, 1995.

[3] C. Carreras et. al., *A Co-Design Methodology Based on formal Specification and High-Level Estimation,* IEEE Proc. of Fourth International Workshop on Hardware/Software Codesign, Pittburg, Pensylvania, 1996.

[4] A. Balboni, W. Fornaciari, D. Sciuto, *Cosynthesis and Co-Simulation of Control-Dominated Embedded Systems,* Design Automation for Embedded Systems, Vol. 1, No. 3, Kluwer, 1996.

[5] A.Kavalade, E. Lee, *A Global Criticality/Local Phase Driven Algorithm for the Constrained Hardware/Software Partitioning Problem,* Proc. of 3rd Int'l Workshop on Hardware/Software Codesign, Grenoble, France, Sept. 22-24, 1994.

[6] J.K. Adams, D.E. Thomas, *Multiple-Process Behavioral Synthesis for Mixed Hardware/Software Systems,*, Proc. of 8th International Symposium on System Synthesis, Cannes, France, Sept. 13-15, 1995.

[7] D.D. Gajski and F. Vahid and S. Narayan,*A System-Design Methodology: Executable-Specification Refinement,* IEEE Proc. of EDAC, 1994

[8] R. Ernst, J.'Henkel et al, *The COSYMA environment for hardware/software cosynthesis of small embedded systems,*, Microprocessors and Microsystems 20 (1996), Elsevier Science B.V., 1996

[9] R.K. Gupta, C.N. Coelho, G.D. Micheli, *Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components,* Proc. of the DAC 92, 1992.

[10] E.A. Lee, G.G. Messerschmitt, *Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing,* IEEE, Trans. on Computers, Jan 1987.

[11] T.-Y. Yen, W. Wolf, *Sensity-driven co-synthesis of distributed embedded systems,* IEEE Proc. of the ISSS, 1995.

[12] U. Banerjee, *Loop Parallelization,* Kluwer, 1994.

[13] Motorola: "TPU - Time Processing Unit Reference Manual", 1990.

[14] A. Takach, W. Wolf, *Scheduling constraint generation for communicating processes,* IEEE Transaction on VLSI Systems, Vol. 3, No. 2, June 1995, pp. 215-230.

[15] W. Ye, R. Ernst, *Embedded program timing analysis based on program and architecture classification,* Technical Report CY-96-3, Institut für DV-Anlagen, Technische Universität Braunschweig, Oct. June 1996.

[16] R. Ernst, Th. Benner, *Communication, Constraints and User Directives in COSYMA,* Technical Report CY-94-2, Institut für DV-Anlagen, Technische Universität Braunschweig, June 1994.

[17] T.B. Ismail, M. Abid, A. Jerraya, *COSMOS: A CoDesign Approach for Communication Systems,* Third Int'l Workshop on Hardware/Software Codesign, Grenoble, Sep. 22-24, 1994.

[18] St. Vercauteren, B. Lin, H. De Man, *Constructing Application Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications,* IEEE Proc. of the DAC 96, pp. 521-526.

[19] P. Chou, G. Borriello, *Software Scheduling in the Co-Synthesis of Reactive Real-Time Systems,* 31st Design Automation Conference, San Diego, CA, June 1994.

[20] F. Vahid, D.D. Gajski, *Incremental Hardware Estimation during Hardware/Software Functional Partitioning,* IEEE Trans. on VLSI Systems, Vol. 3, No. 3, S 459-464, Sept. 1995.

[21] J. Henkel, R. Ernst, *The Interplay of Run-Time Estimation and Granularity in HW/SW Partitioning,* 4th. Int'l Workshop on Hardware/Software Codesign, Pittsburgh, 1996.

[22] Th. Benner, A. Österling, R. Ernst, *Comparison of Context Switching Methods for Fine Grain Process Scheduling,* Technical Report CY-96-1, Institut für DV-Anlagen, Technische Universität Braunschweig, 1996.

[23] Th. Benner, R. Ernst, *A combined Partitioning and Scheduling Algorithm for heterogeneous Multiprocessor Systems* Technical Report CY-96-2, Institut für DV-Anlagen, Technische Universität Braunschweig, 1996.

[24] J. Henkel, R. Ernst, *A Path-Based Estimation Technique for Estimating Hardware Runtime in HW/SW-Cosynthesis,* IEEE Proc. of 8th. Int'l Symp. on System Level Synthesis, pp. 116-121, Cannes, Sep. 1995.