Communication Synthesis for Distributed Embedded Systems*

Ti-Yen Yen
Quickturn Design Systems, Inc.
Mountain View, California

Wayne Wolf
Department of Electrical Engineering
Princeton University

Abstract

Communication synthesis is an essential step in hardware-software co-synthesis: many embedded systems use custom communication topologies and the communication links are often a significant part of the system cost. This paper describes new techniques for the analysis and synthesis of the communication requirements of embedded systems during co-synthesis. Our analysis algorithm derives delay bounds on communication in the system given an allocation of messages to links. This analysis algorithm is used by our synthesis algorithm to choose the required communication links in the system and assign interprocess communication to the links. Experimental results show that our algorithm finds good communication architectures in small amounts of CPU time.

1 Introduction

This paper describes new methodology to co-synthesize communication links for real-time distributed embedded systems. Distributed co-synthesis is important because many embedded systems are heterogeneous distributed machines. Communication is the bottleneck in many embedded systems, because communication links add both chip and board costs, and designers frequently underestimate peak load. Design decisions based on average communication requirements may lead to an infeasible design. The communication must be scheduled and allocated to determine feasibility, and communication synthesis interacts with process scheduling and hardware engine design. In this paper, we propose a bus model for communication in embedded systems with arbitrary topologies in which point-to-point communication is a special case. We extend previous work on delay estimation [19] to include communication delay. We then use the delay estimates to develop methods for synthesizing communication links, based on a previous work on co-synthesis of distributed systems [20]. Our communication-synthesis algorithm selects the number of buses, the type of each bus, the messages transferred on each bus, and schedule the bus communication.

*This work was supported in part by the NSF under grant MIP-9424410.

2 Previous Work

A great deal of literature on real-time distributed systems with interprocessor communication focus on a single non-periodic task graph model. Chu et al. [1] reviewed some of these works and categorizes them as graph-theoretic, integer 0-1 programming, and heuristic approaches. However, in most real-time embedded system, different tasks running in different rates mix together. Rate-monotonic scheduling [10] (RMS) gives a fixed-priority assignment for periodic independent processes on a single processor. Suppose $P_1, P_2, \ldots, P_n$ are n priority-ordered processes allocated on the same CPU, with P_1 being the process with the highest priority. When each deadline is smaller than or equal to the period, Lehoczky et al. [6] showed that the worst-case response time of P_i is the smallest positive root of the equation

$$x = g(x) = c_i + \sum_{j=1}^{i-1} c_j \cdot \lceil x/p_j \rceil \qquad (1)$$

where c_j and p_j are the computation time and the period respectively. Equation 1 will be used as groundwork for communication delay estimation in Section 4. The iteration technique to solve the nonlinear equation have been mentioned by Sha et al. [16], and restated as a *fixed-point iteration* technique [19]. RMS and its extensions can be applied to single CPU or single bus scheduling.

In spite of its wide application, RMS has seldom been extended to schedule distributed systems for periodic processes with data dependencies. The survey of the scheduling or allocation algorithms for periodic tasks in fixed-architecture real-time distributed systems can be found in [17] and [15]. Many algorithms [5, 14, 12] for periodic tasks in distributed systems form a big task with length of the least common multiple (LCM) of all the periods. The LCM method is not efficient and sometimes inaccurate [19]. Peng et

al. [12] and Hou et al. [5] assumed point-to-point communication with delay proportional to the volume of data. Ramamritham [14] used a single multiple-access network for interprocessor communication. Leinbaugh and Yamani [7] derived analytic bounds for response times of a set of tasks, given direct full duplex connection between every pair of PEs.

A great deal of recent work has studied hardware-software partitioning, which only targets a one-CPU-one-ASIC topology [4, 3, 18, 2]. Prakash and Parker [13] formulated distributed system co-synthesis as an integer linear program (ILP). However, their ILP formulation cannot handle periodic and preemptive scheduling of processes or communication in the RMS model. In their system model, the communication topology is restricted to either point-to-point interconnection or a single system bus.

3 Problem Formulation

3.1 Task Graphs

Our task model is similar to those used in distributed system scheduling and allocation problems [5, 14, 12, 7]. A **process** is a single thread of execution, characterized by an **computation time**, which is a function of PE type to which it is allocated. A **task** is a partially-ordered set of processes, which may be represented as an acyclic directed graph known as a **task graph**, in which a directed edge represents a data dependency. A weight on a process denote the volume of output data for communication. A problem specification may contain several concurrently running tasks. Each task is given a **period** (sometimes referred to as a **rate constraint**), which defines the time between two consecutive initiations, a **hard deadline**, which defines the maximum time allowed from initiation to termination of the task and must be satisfied, and a **soft deadline**, which describes the optimization goal of the task delay but does not have to be satisfied. The computation time of a process or the period of a task can be a constant or an interval specified by a lower bound and an upper bound. Release times (i.e., delayed initiation of a process) and multiple deadlines can be modeled by inserting dummy processes—processes with delay but not allocated on any physical PE—in the task graph. An **inter-task communication** is a data transfer between processes in two different tasks. Because tasks run in different rates, such communication does not have to be synchronized. However, the cost and delay for inter-task communication cannot be ignored.

Co-synthesis produces an embedded system architecture. The hardware engine architecture is a labeled graph whose nodes represent **processing elements (PEs)** and whose edges represent **communication links**. When a data dependency crosses PE boundary, define the block of data for transfer as as a **message**. The **allocation** is given by a mapping of processes onto PEs, and a mapping of messages onto communication links. The system **schedule** is an assignment of priorities to processes and an assignment of priorities to messages. The CPU always executes the highest-priority ready process to completion, and a communication link always grant the request of the highest-priority message ready to transfer. There is a cost (component price, etc.) associated with each PE type. There is a hard constraint and a soft constraint on the total system cost.

3.2 The Bus Model

We assume each CPU has a local memory where the program code and local data are stored, so that instruction and local data fetching do not affect interprocess communication. The data transfer between two processes allocated on the same PE causes no extra delay or cost. On the other hand, when two processes are allocated on different PEs, the communication between them need go through a bus and can introduce a delay in addition to the execution of the processes.

For each message, we create a **sending process** right after the process P_1 to send the output data of P_1 to the shared memory of a bus, and a **receiving process** right before the process P_2 to receive the data from the shared memory. A **communication process** is either a sending process or a receiving process. We call a normal process in the original task specification an **application process**, whose existence is independent of the system architecture. Unlike an application process, a communication process needs to be allocated on not only a PE but also on a bus. The following properties should be identified for a communication process or a communication link:

- The **communication time** of a communication process is the time spent on finishing an uninterrupted data transfer. It is proportional to the size of the message, the PE speed, and the bus speed.
- Whether the communication will interfere with the computation depends on the type of PE. Note that a DMA controller without on-chip memory may not really separate computation and communication, because a DMA transfer holds the local bus and prevents the CPU from fetching instructions.
- If there is a dual-port buffer between a PE and a bus, the sending or receiving process on the PE can be spared. Another PE can directly drop its data on the buffer instead of shared memory.
- Whenever a PE is connected to a bus, an associated cost is added to the total system cost. The cost includes bus interface logic and extra bus length.

In bus communication, at most one master (PE) can utilize a bus at a time. When more than one PE want to send or receive a message through a bus, it is necessary to schedule the communication on the bus. A common implementation scheme may assign a fixed priority for each PE on a bus. If a PE has higher priority than the others, all the sending and receiving processes will have higher priority than the communication processes on the other PEs. A more complex scheme, as used in FutureBus and NuBus, let a PE select its own priority during bus arbitration. The second scheme requires more interface logic and more bus arbitration delay, and are seldom used in embedded systems.

A communication process actually uses two resources: a bus and a PE. In addition to scheduling the communication on the bus, we also need to schedule the communication processes on the PE, especially when the PE cannot perform communication and computation in parallel. In this case, the priorities are ordered for all the processes, including both application processes and communication processes. However, we assume that a communication process is non-preemptive on a CPU for several reasons: CPU preemption is usually implemented with interrupts, and interrupts are not sampled in the middle of a bus cycle; and DMA operations are usually not preemptable. Furthermore, preempting a communication causes swapping of processes on the bus and both the sending and receiving PEs; the large amount of overhead occurs suggests that preemptive communication on a PE is not a practical approach.

4 Communication Delay Estimation

Performance analysis is essential to synthesis. In this section, we extend the delay estimation techniques developed previously [19] to handle communication. Suppose the allocation and scheduling have been given. For each process P_i, c_i is the computation time or communication time, and P_i is allocated on PE_i with a priority $Pprt_i$. If P_i is a communication process, it is also allocated on BUS_i with a priority $Bprt_i$. The occurring period of P_i is p_i.

4.1 Communication Modeling

Figure 1 describes how a sending process and a receiving process may be inserted into an edge in a task graph for various cases. If two processes connected by an edge are allocated on the same PE, or either of the processes is a dummy process, no communication process needs to be created for this arc. When two processes connected by an edge are allocated on different PEs, at least one communication process needs to be created for the corresponding message. If there is a dual-port buffer, either sending or receiving process can be deleted. The existence of a communication process depends on the allocation of other processes.

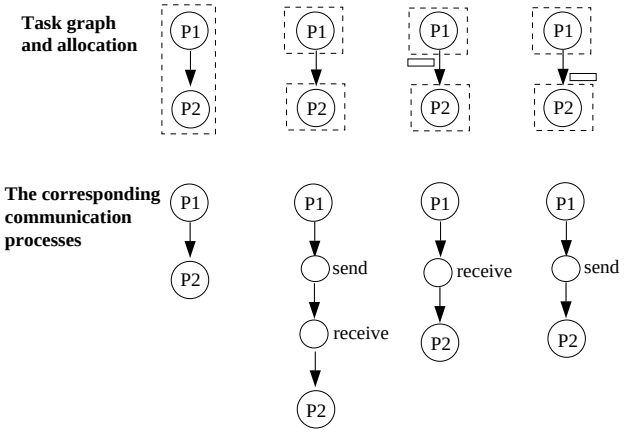


Figure 1: The creation of communication processes for various situations. Dash boxes represent PEs. A small solid box stands for a dual-port buffer for a PE.

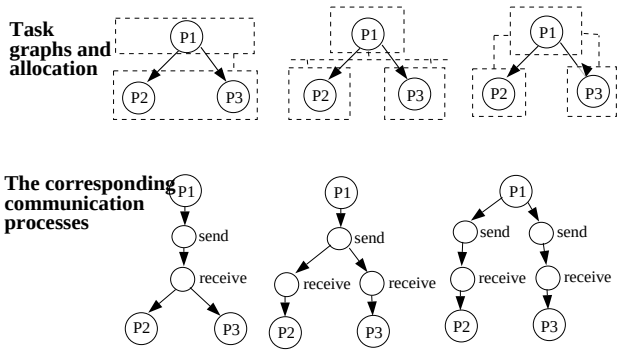


Figure 2: The sharing of communication processes for various situations. Dash boxes and dash line represent PEs and communication links in the current allocation, respectively.

When there is more than one edge leaving a process in a task graph, and no dual-port buffer is available, a communication process may be shared by different edges, as shown in Figure 2. If the destinations of two edges leaving the same source process are allocated on the same PE, both the sending process and the receiving process can be shared, because once the same message is read into the local memory for the first process, the second process can also read it without accessing the bus again. If the destinations of two edges are on two different PEs which use the same bus used by the PE containing the source process, the sending process can be shared, although the receiving processes must be separated. The saving of sending processes implied that when a process wants to broadcast data to more than one process, a bus might provide better performance than point-to-point communication links. By eliminating unnecessary communication processes, we can reduce overhead in delay.

4.2 Communication Delay

Our previous algorithm [19] for estimating response time of a task graph ignored communication delay. In the following, we discuss how much should be added to the total task delay when a communication process is visited. We will use a pure RMS model without data dependencies. The method in this section can be extended by techniques such as *separation analysis* and *phase adjustment* [19] to deal with data dependencies between processes.

Given any communication process P_i, define the set of processes which share the same bus as P_i, and have higher priority than P_i on the bus, but are not allocated on the same PE as P_i:

$$\mathcal{B}_i = \{P_j | BUS_j = BUS_i, PE_i \neq PE_j, Bprt_j > Bprt_i\}$$

Let the worst-case *bus response time* b_i be the longest time from the instant P_i request the bus to the instant P_i finish all its data transfer. Similar to equation (1), b_i is the smallest positive root of the equation

$$x = c_i + \sum_{P_j \in \mathcal{B}_i} c_j \cdot \lceil x/p_j \rceil \tag{2}$$

where c_j is the communication time for each communication process. The worst-case *total response time* d_c^C due to a communication process P_c is the longest time from the request of P_c to the finish of P_c. The request of a sending process occurs when the process generating the data completes its computation. The request of a receiving process occurs when the corresponding sending process finishes sending the data. The delay d_c^C is divided into two components: $d_c^C = d_c^P + d_c^B$, where d_c^B is the time spent on the bus, and d_c^P is the scheduling delay to wait for some other processes to finish on the PE before starting to use the bus. Apparently, $d_c^B = b_c$ where b_c is calculated by equation (2).

Define the set of processes which are allocated on the same PE as P_c, but cannot run in parallel with P_c, and have higher priority than P_c:

$$\mathcal{P}_c = \{P_j | PE_j = PE_c, Pprt_j > Pprt_c\} \tag{3}$$

The set $\mathcal{P}_c$ includes both application processes and communication processes when the PE spends CPU time on communication, but includes only communication processes if computation and communication can be done independently.

Based on equation (1), we derive that the value of d_c^P is the smallest positive root of the equation

$$x = d_c^N + \sum_{P_j \in \mathcal{P}_c} c_j \cdot \lceil x/p_j \rceil + d_c^H(x) \tag{4}$$

The term d_c^N is the worst-case of the delay caused by a communication process with a lower priority than P_c on PE_c. Because a communication process is assumed to be non-preemptable, if it starts immediately (1 time unit) before the request of P_c, it will continue until it is finished even though it has lower priority. Define the set of communication processes with lower priority than P_c on the same PE:

$$\mathcal{N}_c = \{P_j | PE_j = PE_c, Pprt_j < Pprt_c, P_j \text{ is a communication process.}\}$$

If $\mathcal{N}_c = \phi$, $d_c^N = 0$. Otherwise,

$$d_c^N = \max_{P_j \in \mathcal{N}_c} (b_j - 1) \tag{5}$$

The function $d_c^H(x)$ in equation (4) represents the time interrupted through buses by some other communication processes from other PEs. Define the set of processes which use a bus connected to the PE for P_c and may affect the total response time of P_c:

$$\mathcal{H}_c = \{P_j | PE_j \neq PE_c, \exists P_k \in \mathcal{P}_c \text{ such that } BUS_j = BUS_k \text{ and } Bprt_j > Bprt_k\}$$

For each process P_i in $\mathcal{H}_c$, define the set of processes which can be preempted by P_i on the bus, and belong to $\mathcal{P}_c$.

$$\mathcal{E}_i = \{P_j | P_i \in \mathcal{H}_c, P_j \in \mathcal{P}_c, BUS_j = BUS_i, Bprt_j < Bprt_i\}$$

The function $d_c^H(x)$ in equation 4 is formulated as follows:

$$d_c^H(x) = \sum_{P_i \in \mathcal{H}_c} \min\{ \sum_{P_j \in \mathcal{E}_i} \lceil x/p_j \rceil \cdot \lceil b_j/p_i \rceil, \lceil (x - d_c^N)/p_i \rceil \} \cdot c_i \tag{6}$$

The formulation is special because the preemptive relationship is *not transitive* when two resources—PE and bus—are involved. In fixed-priority scheduling of a single CPU, if process P_1 can preempt P_2, and P_2 can preempt P_3, P_1 is allowed to preempt P_3 too. Suppose P_1 and P_2 are communication processes and use the same bus but different PEs, while P_2 and P_3 use the same CPU but P_3 does not use buses. If P_1 can preempt P_2 during bus transactions, and P_2 can preempt P_3 on the CPU, P_1 may not preempt P_3 because they use different resources. Figure 3 demonstrates such situations. For a process $P_i \in \mathcal{H}_c$, it can preempt a process P_j in $\mathcal{E}_i$ at most $\lceil b_j/p_i \rceil$ times, where b_j is the longest time P_j stays on the bus and can be solved from equation (2). The process P_j can occur during the time interval d_c^P at most $\lceil x/p_j \rceil$ times. Therefore, P_i can affect d_c^P at most $\lceil x/p_j \rceil \cdot \lceil b_j/p_i \rceil$ times. On the other hand, in addition to the occurrences during the interval d_c^N, the number of occurrences for P_i cannot exceed $\lceil (x - d_c^N)/p_i \rceil$. Consequently, we use a min function for these two formulas. Note that a min function is a non-decreasing function, so fixed-point iterations will still converge for equation (4).

4.3 The Delay of an Application Process

If the computation and communication cannot be executed in parallel on a PE, the communication processes can cause extra delay in the response time of an application process when the delay estimation algorithm visits it. The worst-case response time d_a^A for an application process P_a is the smallest positive root

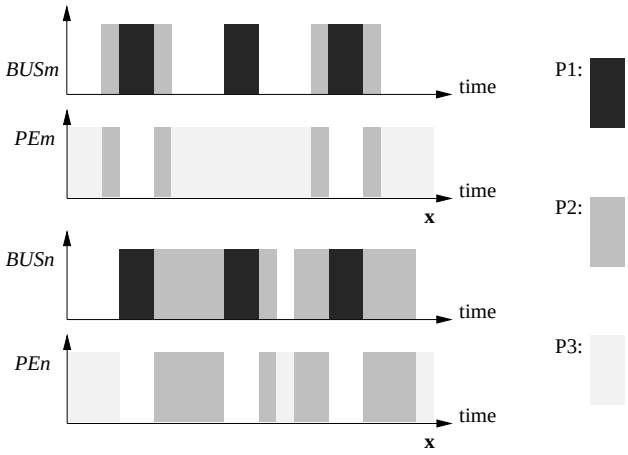


Figure 3: The execution of processes on a bus connected with a PE in two cases. Suppose P_2 has higher priority than P_3 on a PE, and P_1 has higher priority than P_2 on a bus. Let x be the total response time of P_3. For PEm and $BUSm$, the number of times P_1 affects the response time of P_3 is $\lceil x/p_2 \rceil \cdot \lceil b_2/p_1 \rceil$. For PEn and BUSn, the number is $\lceil x/p_1 \rceil$.

of the equation

$$x = d_a^N + c_a + \sum_{P_j \in \mathcal{P}_a} c_j \cdot \lceil x/p_j \rceil + d_a^H(x) \qquad (7)$$

The definition of the set $\mathcal{P}_a$ is the same as that of $\mathcal{P}_c$ in (3). Equation (7) is similar to equation (4), but the computation time c_a is included. The calculation of d_a^N is similar to that of d_c^N in equation (5), and the formulation of $d_a^H(x)$ is similar to that in equation (6).

5 Sensitivity-Driven Synthesis

Our co-synthesis algorithm uses an iterative improvement strategy. At each step, the algorithm may reallocate one process from one PE to another or creates a new PE for the application processes [20]; the algorithm may also reallocate one message from one bus to another or create a new bus. As in most gradient-search methods, we compute a local **sensitivity**: given the current design, we estimate how much the system performance and cost will change when a single process is reallocated. By the delay estimation algorithm in Section 4, we can get the value of a task delay in the current solution and the delay value after a reallocation to evaluate the reallocation.

5.1 Bus Scheduling

In our case, the deadline may not be equal to the period, so the rate-monotonic priority assignment [10] of priorities is not optimal. If the deadline is smaller than or equal to the period for a process, the inverse-deadline priority assignment [8] is optimal for one processor. However, in our model the deadline is specified end-to-end for a whole task, not for individual processes. Based on the *priority prediction* technique [20], we develop a heuristic to use the inverse-deadline priority assignment for bus scheduling.

We define the **fractional deadline** of a communication process—the portion of the task deadline which a particular communication process must meet—as follows. The performance analysis algorithm for the worst-case task delay [19] calculates the **latest initiation time** and **latest termination time** relative to the start of a task for each process. Assign each process a weight equal to its latest termination time minus its latest initiation time. For each bus B, temporarily assign weight zero for the set of processes $\mathcal{J}^B$ allocated on B in the task. Then apply the longest-path algorithm backwards from the end of the task. The **latest required time** of each process in $\mathcal{J}^B$ is the hard deadline the task minus the longest path weight of the process. The calculation of latest required times is similar to the technique in as-late-as-possible (ALAP) scheduling of high-level synthesis [11]. The latest required time is the time before which the process must finish in order not to violate the task deadline when the processes allocated on other PEs run in their worst case.

If the bus arbitration scheme allows the assignment of a priority for each message, the fractional deadline d_i of each process $P_i \in \mathcal{J}^R$ is its latest required time minus its latest initiation time. We can then order the priority by d_i—the shorter the fractional deadline is, the higher the priority is. When the bus arbitration scheme assigns priorities to PEs only, and all the communication processes on the same PE have the same priority on the bus, the PEs on a bus are scheduled as follows. For each PE R on the bus B and for each task, define $\mathcal{J}_R^B$ as the subset of $\mathcal{J}^B$ such that $\mathcal{J}_R^B$ contains only those processes which are allocated on R. Let the largest of the latest required times for processes in $\mathcal{J}_R^B$ be the fractional deadline of all the communication processes in $\mathcal{J}_R^B$. Choose the tightest (smallest) fractional deadline among the fractional deadlines computed in all the task graphs as the deadline of the PE. Schedule the PEs on the bus by assigning a higher priority to a PE with smaller deadline.

5.2 Communication Synthesis

We proposed an algorithm to schedule and allocate processes on PEs in [20], where we did not handle the synthesis of communication links. Several modifications are required to incorporate communication costs into that co-synthesis algorithm. First, in the initial solution, implement a bus for each message. Because we allocate a different PE for each process in the initial solution, there is a bus for each edge in a task graph. When computing sensitivities, include communication delays computed by the method in Section 4, and the

bus cost mentioned in Section 3. In addition to considering possible reallocation of a process to another PE, consider also possible reallocation of a pair of PEs to another bus. Choose either a process reallocation or a communication reallocation according to sensitivity analysis during each iteration. When no reallocation remains feasible, try to create a bus, in addition to a new PE. Compute the sensitivities for each possible bus type and each message. The reallocation with highest sensitivity will be chosen. Make such an reallocation, delete and regenerate communication processes according to the new allocation. Reschedule the PEs and buses. These steps are repeated until no reallocation is possible.

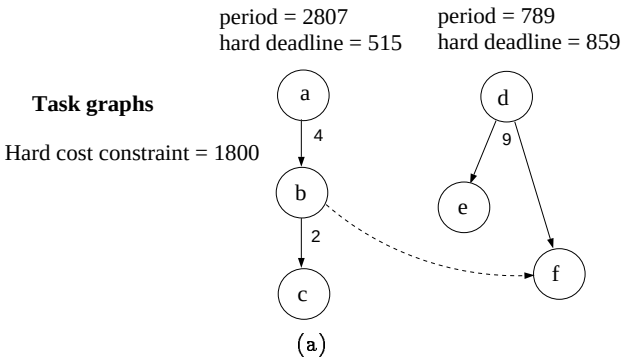
6 Experimental Results

We implemented our algorithm in C++ and performed experiments on several examples. All experiments were performed on a Sun Sparcstation SS20. The results of all our experiments are summarized in Table 6.

The first example, *ex1* is a small example shown in Figure 6. We assume the designer wants to reduce the delay and the cost as much as possible, so the soft constraints and soft deadlines are set to zero to encourage optimization whenever it is possible. The communication time is assumed to be proportional to the size of data. Communication and computation cannot run in parallel on all types of PEs. The embedded system architecture, total cost and the satisfaction of real-time deadlines after each iterative step are given in Table 6.

Our second and third examples are the second example of Prakash and Parker [13]; their algorithm works on a single task graph. We use their assumption about communication cost and delay and that each PE can perform computation and communication in parallel. However, we do not restrict the architecture to use point-to-point communication links. In *prakash-parker-1*, the hard cost constraint is 15; our result for the task delay is 6 which is worse than, but similar to 5 in their result. In *prakash-parker-2*, the hard cost constraint is 12; our result for the task delay is 6 which is the same as theirs. Their integer linear programming approach is optimal but takes hours on these examples, while ours takes only a minute.

We also combine Prakash and Parker's two examples together and assign the periods as well as the deadlines of 7 and 15 to the two tasks. The results are given under *prakash-parker-2*. This example demonstrates how our algorithm can co-synthesize from multiple disjoint task graphs, which ILP formulation cannot handle.



(a)

PE type	Cost	Computation time					
		a	b	c	d	e	f
X	\$800	179	95	100	213	367	75
Y	\$500	204	124	173	372	394	84
Z	\$400	210	130	193	399	494	91

(b)

Bus type	communication time per data unit	Bus interface cost		
		X	Y	Z
B1	2	36	19	30
B2	1	20	10	15

Figure 4: A small example *ex1*. (a) The task graphs for two tasks, their periods and deadlines. The number below a process is the volume of data for transfer. The dash line from b to f is an inter-task communication. (b) The computation time of each process and the cost on each type of PE. (c) The communication time per data unit and the bus interface cost on each type of bus.

7 Conclusions

Communication is critical for the performance and cost of distributed embedded systems. We have presented a new communication synthesis algorithm for heterogeneous distributed systems of arbitrary topology. In the future, we plan to consider more communication protocols in our bus model. Algorithms which analyze caching effects [9] can be used to adjust available bus bandwidth. We believe that algorithms such as this are an important tool for the practicing embedded system designer.

References

[1] W. W. Chu and L. M.-T. Lan. Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers*, C-36(6), June 1987.

Example	Problem size				The result			CPU
	#task	#process	#PE type	#bus type	#PE	#bus	cost	time
ex1	2	6	3	2	3	2	$1765	10.63s
prakash-parker-1	1	9	3	1	4	2	$14.5	59.15s
prakash-parker-2	1	9	3	1	4	1	$12.0	56.79s
prakash-parker-3	2	13	3	1	3	1	$11.5	193.3s

Table 2: The problem size, the final result, and the CPU time of running our algorithm for each example.

Step	embedded system architecture	cost
1	(Y: e) (Z: a) (Z: b) (Z: c) (Z: d) (Z: f) (B1: a→b) (B1: b→c) (B1: d→e) (B1: d→f) (B1: b→f)	$2645
2	(Y: e d) (Z: a) (Z: b) (Z: c) (Z: f) (B1: a→b) (B1: b→c) (B1: d→f) (B1: b→f)	$2215
3	(Y: e d) (Z: b a) (Z: c) (Z: f) (B1: b→c) (B1: d→f) (B1: b→f)	$1785
4	(Y: e d) (Z: c b a) (Z: f) (B1: d→f) (B1: b→f)	$1355
5	(X: c) (Y: e d) (Z: b a) (Z: f) (B1: b→c) (B1: d→f) (B1: b→f)	$2190
6	(X: c f) (Y: e d) (Z: b a) (B1: b→c b→f) (B1: d→f)	$1765
7	(X: c f b) (Y: e d) (Z: a) (B1: a→b) (B1: d→f)	$1765

Table 1: The iterative optimization for *ex1*. After each step, each PE or bus in the refined system architecture is shown by its type, followed by a colon, and then the processes or messages allocated on it are listed from the highest priority to the lowest according to the schedule.

[2] E.Barros, W. Rosenstiel, and X. Xiong. A method for partitioning UNITY Language in hardware and software. In *Proceedings, European Design Automation Conference*, 1994.

[3] R. Ernst, J. Henkel, and T. Benner. Hardware-software co-synthesis for microcontrollers. *IEEE Design & Test of Computers*, 10(4), December 1993.

[4] R. K. Gupta and G. D. Micheli. Hardware-software cosynthesis for digital systems. *IEEE Design & Test of Computers*, 10(3), September 1993.

[5] C. J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *Proceedings, Real-Time Systems Symposium*, 1982.

[6] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings, IEEE Real-Time Systems Symposium*, 1989.

[7] D. W. Leinbaugh and M.-R. Yamani. Guaranteed response times in a distributed hard-real-time environment. In *Proceedings, Real-Time Systems Symposium*, 1982.

[8] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2, 1982.

[9] Y.-T. S. Li and S. Malik. Performance estimation of embedded software with instruction cache modeling. In *Proceedings, IEEE International Conference on Computer-Aided Design*, 1995.

[10] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), Jan. 1973.

[11] M. C. McFarland, A. C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proceedings of the IEEE*, 78(2), February 1990.

[12] D.-T. Peng and K. G. Shin. Static allocation of periodic tasks with precedence constraints. In *Proceedings, International Conference on Distributed Computing Systems*, 1989.

[13] S. Prakash and A. C. Parker. SOS: synthesis of application-specific heterogeneous multiprocessor systems. *Journal of Parallel and Distributed Computing*, 16, 1992.

[14] K. Ramamritham. Allocation and scheduling of complex periodic tasks. In *Proceedings, International Conference on Distributed Computing Systems*, 1990.

[15] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1), January 1994.

[16] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1), January 1994.

[17] K. G. Shin and P. Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1), January 1994.

[18] F. Vahid, J. Gong, and D. D. Gajski. A binary-constraint search algorithm for minimizing hardware during hardware/software partitioning. In *Proceedings, European Design Automation Conference*, 1994.

[19] T.-Y. Yen and W. Wolf. Performance estimation for real-time distributed embedded systems. In *Proceedings, IEEE International Conference on Computer Design*, 1995.

[20] T.-Y. Yen and W. Wolf. Sensitivity-driven co-synthesis of distributed embedded systems. In *Proceedings, 8th International Symposium on System Synthesis*, 1995.