

Communication Architecture Tuners: A Methodology for the Design of High-Performance Communication Architectures for System-on-Chips^{*}

†Kanishka Lahiri ‡Anand Raghunathan ‡Ganesh Lakshminarayana †Sujit Dey
†Dept. of Electrical and Computer Engg., University of California, San Diego, CA
‡C & C Research Labs, NEC USA, Princeton, NJ

Abstract

In this paper, we present a general methodology for the design of custom system-on-chip communication architectures. Our technique is based on the addition of a layer of circuitry, called the Communication Architecture Tuner (CAT), around any existing communication architecture topology. The added layer enhances the ability of the system to adapt to changing communication needs of its constituent components. For example, more critical data may be handled differently, leading to lower communication latencies. The CAT monitors the internal state and communication transactions of each component, and “predicts” the relative importance of each communication transaction in terms of its potential impact on different system-level performance metrics. It then configures the protocol parameters of the underlying communication architecture (*e.g.*, priorities, DMA modes, *etc.*) to best suit the system’s changing communication needs.

We illustrate issues and tradeoffs involved in the design of CAT-based communication architectures, and present algorithms to automate the key steps. Experimental results indicate that performance metrics (*e.g.* number of missed deadlines, average processing time) for systems with CAT-based communication architectures are significantly (sometimes, over an order of magnitude) better than those with conventional communication architectures.

1 Introduction

The evolution of the System-on-Chip (SOC) paradigm in electronic system design has the potential to offer the designer several benefits, including improvements in system cost, size, performance, power dissipation, and design turn-around-time. The ability to realize this potential depends on how well the designer exploits the customizability offered by the system-on-chip approach. While one dimension of this customizability is manifested in the diversity and configurability of the components that are used to compose the system (*e.g.*, processor and domain-specific cores, peripherals, *etc.*), another, equally important aspect, is the customizability of the system communication architecture. In order to support the increasing diversity and volume of on-chip communication requirements, while meeting stringent performance constraints and power budgets, communication architectures need to be customized to the target system or application domain in which they are used.

1.1 Paper Overview and Contributions

In this paper, we demonstrate the need for the design of flexible communication architectures by analyzing example systems and scenarios in which no static customization of the protocols can completely satisfy the system’s time-varying communication requirements. The presented technique can be used to optimize any underlying communication architecture topology by enhancing it with Communication Architecture Tuners (CATs) which make it capable of adapting to the changing communication needs of the components connected to it. In this paper we illustrate the issues and tradeoffs involved in the design of CAT-based communication architectures,

present a methodology, and describe algorithms for their design. Experimental results for several example systems, including an ATM switch port scheduler and a TCP/IP Network Interface Card subsystem, indicate that performance (metrics such as number of missed deadlines, average or aggregate processing time, *etc.*) of systems with CAT-based communication architectures can be significantly better than corresponding systems with well-optimized conventional communication architectures. In the rest of the paper we show that:

- CAT-based communication architectures can *extend the power of* any underlying communication architecture. The timing behavior of a CAT-based communication architecture is *better customized* to the needs of each component that is connected to it.
- The presented CAT design methodology trades off sophistication of the communication architecture protocol with the complexity of (and hence, overhead incurred by) the added hardware.
- In several cases, the use of CAT-based communication architectures can result in systems that significantly outperform those based on any static customization of the protocol parameters.

1.2 Related Work

There is a large body of work on system-level synthesis of application-specific architectures, through HW/SW partitioning, and through mapping of the application tasks onto pre-designed cores and application-specific hardware [1, 2, 3, 4, 5, 6, 7, 8]. Those that do not ignore communication altogether, either assume a fixed communication protocol (*e.g.*, PCI-based buses), or select from a “communication library” of a few alternative protocols. Research on system-level synthesis of communication architectures [9, 10, 11, 12] mostly deals with synthesis of the communication architecture topology. While topology selection is a critical step in communication architecture design, equally important is the design of the protocols (*e.g.*, time-slice based [13], static priority based [14], *etc.*) used by the channels/buses in the selected topology. The VSI Alliance on-chip bus working group [14] has recognized that a multitude of bus protocols will be needed in order to serve the wide range of SOC communication requirements. Finally, there is a body of work on interface synthesis [15, 16, 17, 18, 19, 20, 21, 22], which deals with automatically generating efficient hardware implementations for component-to-bus or component-to-component interfaces. These techniques address issues in the implementation of specified protocols, and not in the customization of the protocols themselves.

In summary, we believe that previous work in the field of system-level design and HW/SW co-design does not adequately address the problem of customizing the protocols used in SOC communication architectures to the needs of the application. Another characteristic of previous research is that the design of the communication architecture is performed *statically* using information about the application and its environment. In several applications, the communication bandwidth required by each component and the relative “importance” of each communication request, may be subject to significant dynamic variations. As shown later in this paper, in such situations, protocols used in conventional communication architectures may result in inadequate or significantly sub-optimal performance.

2 Communication Architecture Tuners: Design Issues

In this section, we first motivate the need for CAT-based communication architectures by showing how the limited flexibility of conventional communication architectures can lead to significant deterioration in the system’s performance. We then introduce CAT-based communication architectures and show how they address the above mentioned drawbacks. Finally, we discuss the key issues and tradeoffs involved in a CAT-based communication architecture design methodology.

^{*}To appear at the 37th Design Automation Conference (DAC), Los Angeles, May 2000.

Example 1: Consider the example system shown in Figure 1 that represents part of the TCP/IP communications protocol used in a network interface card. The system performs checksum-based encoding (for outgoing packets) and error detection (for incoming packets), and interfaces with the Ethernet controller peripheral (which implements the physical and link layer network protocols). Since packets in the TCP protocol do not contain any notion of quality of service (QoS) [23], we have enhanced the packet data structure to contain a field in the header that indicates a deadline for the packet to be processed. We assume that the objective during the implementation of the system is to minimize the number of packets with missed deadlines.

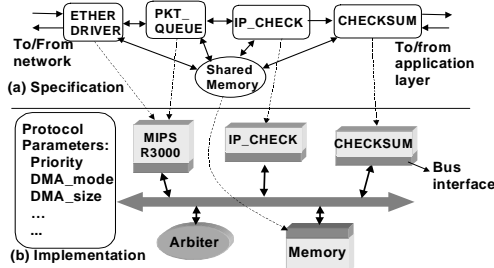


Figure 1: TCP system from a network interface card: (a) Specification and (b) Implementation utilizing a conventional bus-based communication architecture

Figure 1(a) shows the behavior of the TCP system as a set of concurrent communicating processes. Figure 1(b) shows the system architecture used to implement the TCP system. The `ether_driver` and `pkt_queue` processes are mapped to embedded software running on a MIPS R3000 processor, while the `ip_check` and `checksum` processes are implemented using dedicated hardware. All communications between the system components are implemented using a shared bus. The protocol used in the shared bus supports static priority based arbitration and DMA-mode transfer. The arbiter and the bus interfaces of the components together implement the bus protocol, which allows the system designer to specify values for various parameters such as bus priorities, DMA block size etc.

We analyzed the performance of the TCP system of Figure 1 for several distinct values of the bus protocol parameters. For ease of explanation, we vary only the bus priority values for each component, with fixed values for the remaining protocol parameters. An abstract view of the execution of the TCP system processing four packets (numbered $i, i+1, j, j+1$) is shown in Figure 2. The figure indicates the times at which each packet arrives from the network, and the deadline by which it needs to be processed. Note that while the arrival times of the packets are in the order $i, i+1, j, j+1$, the deadlines are in a different order $i+1, i, j, j+1$.

Consider the first waveform in Figure 2, which represents the execution of the system when the bus priority assignment $checksum > ip_check > ether_driver$ is used. After the completion of the `ether_driver` process for packet i , the arbiter receives two conflicting bus access requests: `process ip_check` requests bus access to process packet i , while `ether_driver` requests bus access to process packet $i+1$ (since packet $i+1$ has already arrived from the network). Based on the priority assignment, the arbiter gives bus access to process `ip_check`. This effectively delays the processing of packet $i+1$ until `ip_check` and `checksum` have completed processing packet i . This leads to packet $i+1$ missing its deadline. Packets j and $j+1$ do meet their deadlines.

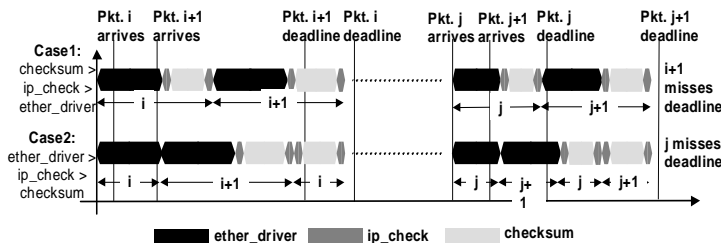


Figure 2: Execution of the TCP system for various bus priority assignments

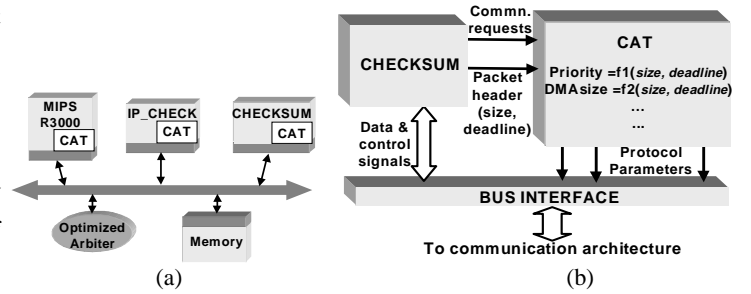


Figure 3: CAT-based architecture for the TCP example

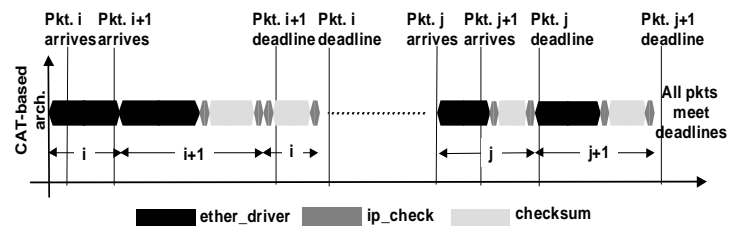


Figure 4: Execution of the optimized TCP system with a CAT-based architecture

We attempted to eliminate the problem mentioned above by using a different priority assignment ($ether_driver > ip_check > checksum$) for the bus protocol. The execution of the system under the new priority assignment is depicted in the second waveform of Figure 2. As a result of the new priority assignment, packets i and $i+1$ meet their deadlines, but packet j misses it, due to contention with a request from `ether_driver` for processing packet $j+1$.

In summary, each of the two bus priority assignments considered for the TCP system led to missed deadlines. Further, the arguments presented in the previous two paragraphs can be applied to show that for every possible priority assignment, either packet $i+1$ or packet j will miss its deadline. ■

The example shows a situation where the relative importance of the communication transactions generated by the various system components varies depending on the deadlines of the packets they are processing. It demonstrates that conventional communication architectures (i) provide too limited a degree of customizability for systems with stringent performance requirements, and (ii) are typically not capable of sensing and adapting to the varying communication needs of the system and the varying nature of the data being communicated.

Example 2: A CAT-based communication architecture for the TCP system is shown in Figure 3(a). CATs are added to each bus master and the bus control logic (arbiter and component bus interfaces) is enhanced to facilitate their operation. A more detailed view of a component with a CAT is shown in Figure 3(b). The component notifies the CAT when it generates communication requests. The CAT also observes selected details about the data being communicated and the component's internal state.

In this example, the CAT observes the packet size and deadline fields from the header of the packet currently being processed by the component and performs the following functions: (i) it groups communication events based on the size and deadline of the packet currently being processed, and (ii) for events from each group, it determines an appropriate assignment of values to the various protocol parameters. The rationale behind using the deadline is that packets with closer deadlines need to be given higher importance. In cases when all the packets in the system have roughly equal deadlines, it is advantageous to favor the completion of packets which are smaller, since they have a better chance of meeting the deadline. For example, the priority is computed using the formula $s * (t_d - t_a)$ where s , t_d and t_a represent the packet size, deadline, and arrival time, respectively.

The packet sequence of Example 1 was used to drive execution of the optimized system of Figure 3. Under a CAT-based architecture, the system meets the deadlines for all the packets. When packet $i+1$ (which has a tight deadline) arrives, the CAT assigns to the communication requests generated by `ether_driver` a priority higher than those assigned to requests from `ip_check` and `checksum`, which are still processing packet i . This leads to packet $i+1$ meeting its deadline. When

packet $j + 1$ arrives, however, the communication requests generated by *ether_driver* are assigned a lower priority, allowing *ip_check* and *checksum* to process packet j to completion in order to meet its tight deadline.

An effective realization of a CAT-based communication architecture hinges on judiciously performing the following steps:

- Identifying performance-critical communication events from a performance analysis of the system.
- Detecting the occurrence of these communication events in hardware while the system is executing.
- Assigning appropriate values for communication protocol parameters (such as priorities and DMA sizes) to the critical events, and translating these results into a high-performance implementation.

In our work, we use an *analysis of the system execution traces* as a basis for identifying critical communication events. An advantage of using execution traces is that they can be derived for any system for which a system-level simulation model exists. The generated traces can be analyzed to examine the impact of individual (or groups of) communication events on the system’s performance and identify critical events.

In addition to identifying critical communication events, we need to correlate their occurrence to other *easily detectable properties* of the system state and data it is processing. If an analysis of the simulation trace reveals that the occurrence of a critical data-transfer is highly correlated to a specific branch in the behavior of the component executing the transfer, the occurrence of the branch might be used as a predictor for the criticality of the data transfers generated by the component. The following example examines some tradeoffs in designing these predictors.

Example 3: Consider the system shown in Figure 5, which is used to encrypt data for security before transmission onto a communications network. *Component1* processes the data, determines the coding and encryption scheme to be used, and sends the data to *Component2*, which encodes and encrypts the data before sending it through the shared bus to the peripheral that transmits it onto the network. Figure 6 shows the data transfers occurring on the system bus. The shaded ellipses, marked y_i ($i = 1 \dots n$), represent data transfers from *Component2* to the network peripheral. Let us suppose that *Component2* should transfer data at a fixed rate, and each data transfer should occur before a deadline (indicated in Figure 6 by a dotted line). The analysis of the execution trace indicates that deadlines are frequently not met and identifies those communication events that did not meet their deadlines, e.g., y_1 and y_2 . In addition, it also identifies *critical communication events*, i.e., those which when sped up, could potentially improve system performance. If x_i denotes a critical communication event (Figure 6), and S denotes the set of all x_i ’s, the performance of the system can improve if the execution of events in S improve.

Next, we correlate the occurrence of critical communication events with information about the system state and data it is processing. We define a *control-flow event* as a Boolean variable which assumes a value of 1 when a component executes a specific operation. For example, the behavior of *Component1* shown in Figure 5 is annotated with control-flow events e_1, e_2, e_3 , and e_4 . In general, if e_1, e_2, \dots, e_n are the control-flow events which are used to determine whether or not a communication request is critical, we can define a Boolean function $f_{critical} = f(e_1, e_2, \dots, e_n)$ whose on-set denotes the set of communication events classified as critical.

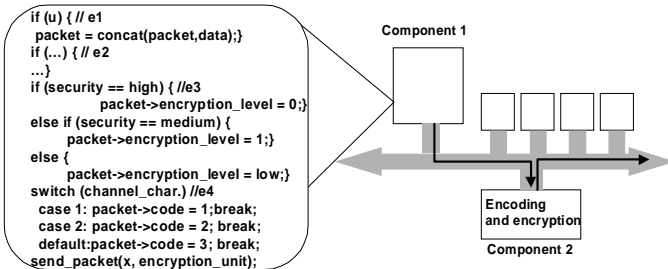


Figure 5: A data encryption system that illustrates tradeoffs in the identification of critical communication events

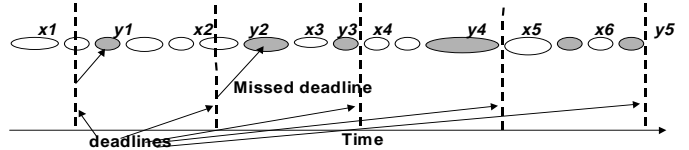


Figure 6: A trace of bus activity for the system shown in Figure 5

The number of control-flow variables used for this classification has a profound impact on the classification of communication events. A good classification should have the properties of a one-to-one map, i.e., every event classified as critical should indeed be critical, and every critical event should be detected by the classification. Suppose, in this example, we are allowed to use only one variable for classification, say, e_3 . We find that, in all the cases where deadlines are missed, event e_3 occurs. Based on this insight, we may choose $f_{critical} = e_3$. However, e_3 often occurs along with non-critical communication events as well. If e_3 is used as a classifier, only 16% of the communication events classified to be critical are indeed critical. Therefore, e_3 could erroneously classify several communication events, and incorrectly increase their priorities, causing system performance to suffer.

Figures 7 (a) and (b) plot the percentage of critical communication events in $f_{critical}$, and the percentage of S covered by $f_{critical}$, versus the number of variables that perform the classification, respectively. The x axis shows the number of variables used to perform the classification. For example, the best classifier that uses two variables captures 100% of critical communication events, while only 50% of the communication events classified as “critical” by it are actually critical. Note that, in this example, as the number of variables increases, the percentage of critical communication events in $f_{critical}$ increases. This is because, as the number of variables increases, the classification criterion becomes more stringent, and non-critical events are less likely to pass the test. However, simultaneously, critical events could be missed, as shown in Figure 7(b) (note the decrease in the percentage of S covered as the number of variables used increases). Therefore, one needs to judiciously choose the right number of variables, and the right classification functions in order to maximally improve system performance. In this example, optimal results are obtained by using three variables (e_1, e_2 , and e_3) and a classification function $f_{critical} = e_1 \cdot e_2 \cdot e_3$. This identifies most of the critical events, and very few non-critical ones.

3 Overall Methodology for the Design of CATs

In this section, we present a structured methodology and outline the different steps involved in the design of CAT-based communication architectures.

Our algorithm takes as inputs a simulateable partitioned/mapped system description, the selected communication architecture topology, typical environment stimulus or input traces, and objectives and/or constraints on performance metrics. The performance metrics could be specified in terms of the amount of time taken to complete a specific amount of work (e.g., a weighted or uniform average of processing times) or in terms of the number of output deadlines met or missed for applications with real-time constraints. The output of the algorithm is a set of optimized communication protocols for the target system. From a hardware point of view, the system is enhanced through the addition of Communication Architecture Tuners wherever necessary, and through the modification of the

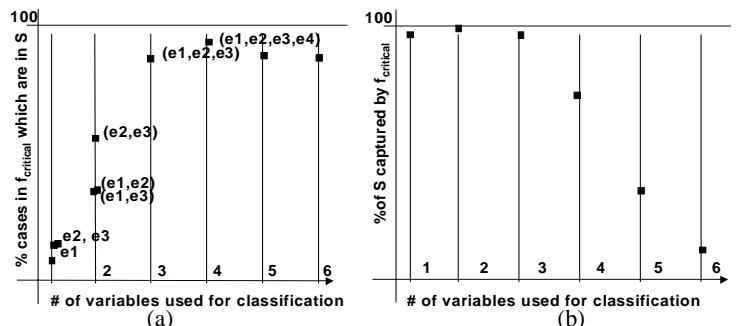


Figure 7: A plot of different classification metrics with respect to the number of variables used for the classification

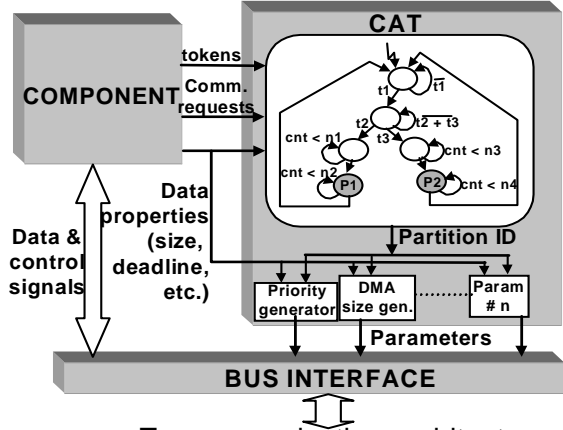


Figure 8: Detailed view of a component with a CAT

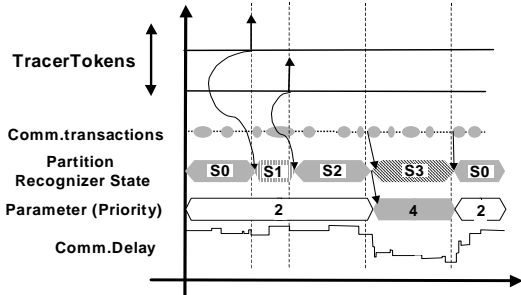


Figure 9: Symbolic illustration of CAT-optimized communication architecture execution

controllers/arbiters for the various channels in the communication architecture.

A simple system with a CAT-based communication architecture generated using our techniques was shown in Figure 3(a). A more detailed view of a component with a CAT is shown in Figure 8. The CAT consists of a “partition detector” circuit, and parameter generation circuits that generate values for the various communication architecture protocol parameters during system execution.

Partition detector: A *communication partition* is a subset of the communication transactions generated by the component during system execution. The partition detector circuit monitors and analyzes the following information generated by the component:

- *Tracer tokens* generated by the component to indicate that it is executing specific operations. The component is enhanced to generate these tokens purely for the purpose of the CAT.
- The communication transaction initiation requests that are generated by the component.
- Any other application-specific properties of the communication data being generated by the component (e.g., fields in the data which indicate its relative importance).

The partition detector uses this information to identify the start and end of a sequence of consecutive communication transactions that belong to a partition. In Section 4.3, we present general techniques to automatically compute the start and end conditions for each partition.

Parameter generation circuits: These circuits compute values for communication protocol parameters (e.g., priorities, DMA block sizes, etc.) based on the partition ID generated by the partition detector circuit, and other application-specific data properties specified by the system designer. The values of these parameters are sent to the arbiters and controllers in the communication architecture, resulting in a change in the characteristics of the communication architecture. Automatic techniques to design the parameter generation circuits are presented in Section 4.2.

The functioning of a CAT-based communication architecture is illustrated using symbolic waveforms in Figure 9. The first two waveforms represent tracer tokens generated by the component while the third waveform represents the communication transactions generated

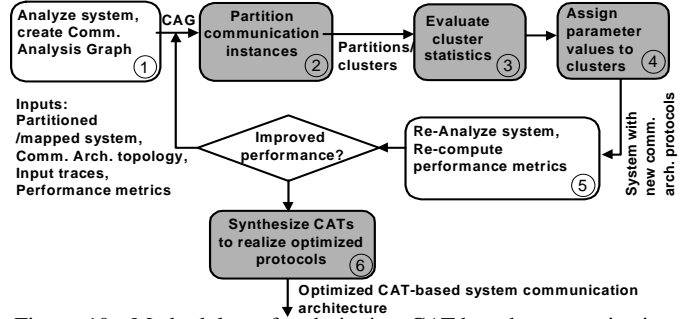


Figure 10: Methodology for designing CAT-based communication architectures

by the component. The fourth waveform shows the state of the partition detector circuit which changes in response to the first three. All communication transactions that occur when the partition detector FSM is in state $S3$ are classified as belonging to partition CP_1 . The fifth waveform shows the output of the priority generation circuit. Here it assigns a priority level of 4 to all transactions that belong to partition CP_1 . This increase in priority leads to a decrease in the delay associated with the communication transactions that belong to partition CP_1 , as shown in the last waveform of Figure 9.

The overall methodology for designing CAT-based communication architectures is shown in Figure 10. In step 1, performance analysis is applied to the partitioned/mapped system description in order to derive the information and statistics used in the later steps. In our work, we use the performance analysis technique presented in [24], which is comparable in accuracy to complete system simulation, while being much more efficient to employ in an iterative manner. The output of this analysis is a *Communication Analysis Graph*, (CAG) which is a highly compact representation of the system’s execution under the given input traces. The vertices in the graph represent clusters of computations and abstract communications performed by the various components during the system execution. The edges in the graph represent the dependencies between the various computations and communications. The CAG is constructed from a detailed system execution trace and can be easily analyzed to determine various performance statistics.

In step 2, we group the communication vertices in the CAG into a number of partitions such that each partition contains a set of events that have similar communication requirements, while different partitions represent distinct requirements. Note that in the extreme case, each communication vertex in the communication analysis graph can be assigned to a distinct partition. However, this can cause the area and delay overhead incurred in the CAT to become prohibitive. Step 3 evaluates various statistics for each communication partition, based on which, step 4 determines an assignment of protocol parameter values for each partition. The output of step 4 is a set of candidate protocols for the system communication architecture. Step 5 re-evaluates the system performance for the optimized protocols derived in step 4. If a performance improvement results, steps 1 to 5 are repeated until no further performance improvement is obtained. Step 6 deals with synthesis of hardware (CATs) to implement the optimized protocols that were determined in step 4.

4 Algorithms for the Design of CATs

In this section we describe the shaded steps of Figure 10 in more detail. We present techniques to obtain partitions of the communication event instances, discuss how to select an optimal set of protocol parameter values and how to synthesize CAT hardware for classifying communication event instances into partitions.

4.1 Profiling and partitioning communication event instances

In step 2 of Figure 10, we perform an analysis of the CAG generated in step 1 to measure the impact of individual communication instance delays on the system performance. Instances which have a similar impact on the system performance are grouped into the same partition. The performance impact of an instance is measured by a parameter called *sensitivity* that captures the change in system performance when the communication delay of the instance changes. The following example illustrates our partitioning procedure.

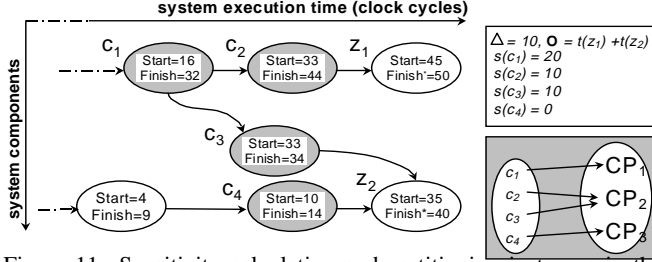


Figure 11: Sensitivity calculation and partitioning instances in the CAG

Figure 11 shows a section of a CAG generated from a representative execution of an example system. Shaded vertices c_1 through c_4 represent instances of communication events. Vertices z_1 and z_2 represent the final outputs of the system. The objective function to be minimized is the quantity $t(z_1) + t(z_2)$, where $t(v)$ is the finish time of a vertex v in the CAG.

To measure the *sensitivity* of the system performance to communication instance c_1 , the existing delay of c_1 is perturbed by a value Δ , and a traversal of the transitive fanout of c_1 in the CAG is used to re-compute the start and finish times of the affected vertices. The updated finish times of the vertices are used to calculate the change in the system performance metric. In this example, perturbing the delay of c_1 by 10 units delays the finish of both z_1 and z_2 by 10 units each, while perturbing the delay of c_2 delays z_1 alone.

In the manner described above, we calculate a sensitivity $s(c_i)$ for each instance c_i , which measures the change in the value of the objective function O after perturbing the delay of c_i by Δ . Next, we group communication event instances that have similar sensitivity values to form a partition. In this example, c_1 is assigned to partition CP_1 , c_2 and c_3 are assigned to CP_2 , and c_4 is assigned to CP_3 .

4.2 Modifying Protocol Parameters

In this section we describe steps 3 and 4 of the overall flow, *i.e.*, how to examine each partition and then assign optimized protocol parameter values to them. While our discussion is confined to determining the priority that should be assigned to each partition, it could be extended to include other protocol parameters such as whether burst mode should be supported or not, and if so what the correct DMA size should be.

The *sensitivity* of a *partition* indicates the impact its events have on the performance of the system. However assigning priorities based on the sensitivity alone may not lead to the best assignment. This is because sensitivity does not capture the indirect effects of a communication event (or set of events) on the delays of other concurrent communication events. We account for this by deriving a metric that penalizes partitions which are likely to negatively impact the execution of communication events in other partitions. In order to obtain this information, we analyze the CAG and evaluate, for each pair of partitions CP_i, CP_j , the amount of time for which communication events that belong to CP_i are delayed due to events from CP_j . Table 1 shows example data for a system with three partitions. Column 2 gives the sensitivity of each partition. Columns 3, 4 and 5 gives the total time (w_{ij}) that instances in partition CP_1, CP_2 and CP_3 wait for instances in each of the other partitions. For example, instances in CP_1 induce a total waiting time of 100 cycles for instances of CP_2 to finish. Column 6 gives the sum of columns 3, 4 and 5 to indicate the total waiting time (W_i) events in partition CP_i have introduced in other partitions.

To assign a priority to partition CP_i , we use a heuristic metric P_i which is calculated (using the notation of Table 1), as

$$P_i = s(i) - \sum_{j=1}^n \frac{s(j)w_{ij}}{W_i}$$

In this formula, the first term accounts for the sensitivity of the partition CP_i , while the summation penalizes the partition for inducing

Table 1: Statistics of the Partition

Partn.	$s(c_i)$	w_{i1} (cycles)	w_{i2} (cycles)	w_{i3} (cycles)	W_i (cycles)	Partition Rank
CP_1	100	0	100	3	103	17.18 => 2
CP_2	85	4	0	3	7	23.57 => 1
CP_3	10	0	7	0	7	-75.0 => 3

waiting time in other partitions. Once this metric has been evaluated for each partition, the partitions are ranked in descending order of P_i . From this ordered list, the first partition is assigned the highest priority, the second partition the second highest priority and so on. Table 1 shows the ranking of partitions for the given values.

4.3 Synthesis of an optimal communication protocol

Figure 12(a) shows an extract of a CAG after the sensitivity-based partitioning step has been performed. For simplicity, only communication vertices are shown in the figure. All highlighted vertices in component C_1 's trace belong to partition CP_1 . In step 6 our goal is to generate a Boolean formula which evaluates to 1 when a vertex belongs to CP_1 and to 0 at other times. The next three waveforms in Figure 12(a) show three cases where different events, t_1, t_2, t_3 , have been chosen to act as *tracers* for component C_1 . For a given *tracer* t_i , a *distance* d is calculated for each communication instance c , equal to the number of communication instances separating c from the previous instance of t_i .

In order to perform partition assignment, the CAT hardware must detect a *tracer*, count and ignore θ_1 communication instances, and then start assigning the next θ_2 instances to a particular partition. For example, for *Formula*₁, t_1 is the *tracer*, $\theta_1 = 4$, and $\theta_2 = 3$. Figure 12(a) shows the actual classification of communication instances that results from each of the three formulae. Figure 12(b) shows the prediction accuracy of each formula. It turns out that *Formula*₁ performs the best, predicting correctly with a probability of 0.9 whether or not a given instance belongs to *Partition*₁. Figure 12(c) shows an FSM that implements *Formula*₁.

In general, choosing the appropriate tracer tokens and appropriate values for θ_1 and θ_2 may not be a trivial task. We formulate the problem in terms of a well-known problem from regression theory, and use known statistical techniques to solve it.

A data set is constructed from the CAG for each examined *tracer* consisting of *distances* d_1, d_2, \dots, d_n , and a 0 or 1 value for each d_i (derived from the partitioned CAG) indicating whether or not the communication instance at distance d_i from the tracer token belongs to a partition CP_i . The regression function f is defined as follows :

$$f(d, \theta) = \begin{cases} 1 & : \theta_1 < d < \theta_1 + \theta_2 \\ 0 & : \text{elsewhere} \end{cases}$$

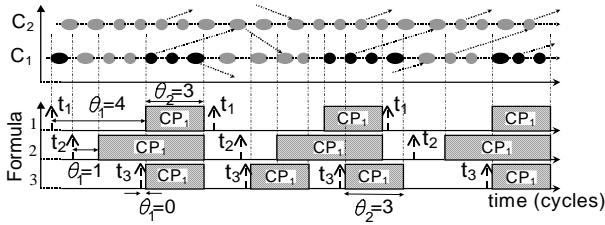
When f is 1, it indicates the instance at a distance d belongs to CP_i . An assignment $\hat{\theta} = \{\theta_1, \theta_2\}$ is required that causes the least mean square error $\sum_{i=1}^n |y - f(d, \theta)|^2$, where y is the value from the data set, and $f(d, \theta)$ is the prediction, to be minimized. Since the regression function is non-linear in $\hat{\theta}$, no universal technique is known to compute an explicit solution. However, several heuristics and iterative procedures may be used [25].

5 Experimental Results

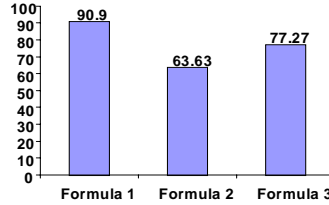
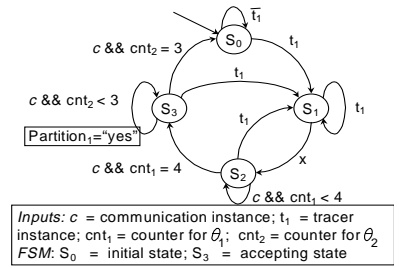
In this section, we present results of the application of our techniques to several example systems. We present performance results based on system-level co-simulation for each example.

The first example is the TCP system described in Section 2. The second example is a packet forwarding unit of an output queued ATM switch. The next example, SYS, is a four component system, where each component issues independent concurrent requests for access to a shared memory through a shared bus. BRDG is a system consisting of four components, two memories and two buses connected by a bridge. Details of both these systems may be found in Figure 2 of [26].

Table 2 demonstrates the performance benefits of using CAT-based communication architectures over a static priority based communication protocol. Each row in the table represents one of the example systems described earlier. For each system, column 2 defines a performance metric. For the TCP, SYS and ATM systems, the performance objective is to minimize the number of missed deadlines. In the case of BRDG, each data transaction is assigned a weight. The performance of the system is expressed as a weighted mean of the execution times of all bus transactions. The objective in this case is to minimize this weighted average processing time. The static communication protocol consists of a fixed DMA size for each communication request and a static priority based bus arbitration scheme. For these examples, the CAT-based architecture uses application specific



(a) Alternative predictor strategies

(b) Performance of predictors for partition CP_1 (c) FSM implementation of $Formula_1$ Figure 12: Design of a predictor function for partition CP_1 of component C_1

information such as the weights on each request and deadlines, as described in Section 4, to provide for a more flexible communication protocol.

Table 2: Performance of systems using CAT-based architectures

Example System	Metric	Input Trace	Static Protocol	CATs based architecture	Improvement
TCP/IP	missed deadlines	20 packets	10	0	-
SYS	missed deadlines	573 trans.	413	17	24.3
ATM	missed deadlines	169 packets	40	16	2.5
BRDG	avg. no. of cycles	10,000 cycles	304.72	254.1	1.2

Table 3: Immunity of CAT-based architectures to variation in inputs

Inputs to the SYS example	Input Trace Information	Static Protocol	CATs based architecture	Performance improvement
Trace 1	848 transactions	318	161	1.98
Trace 2	573 transactions	413	17	24.3
Trace 3	1070 transactions	316	38	8.37

For each system, column 4 reports performance results obtained using a static communication protocol, while column 5 reports results generated by simulating a CAT-based architecture. Speed-ups are reported in column 6. The results indicate that significant benefits in performance can be obtained by using a CAT-based architecture over a protocol using fixed parameter values. In the case of TCP, the number of missed deadlines was reduced to zero, while in the case of SYS, a 24X performance improvement (reduction in the number of missed deadlines) was observed.

The design of an efficient CAT-based communication architecture depends on the selection of a good representative trace when performing the various steps of the algorithm of Figure 10. However, our algorithms attempt to generate communication architectures that are not specific to the exact input traces used to design them, but display improved performance over a wide range of communication traces. In order to analyze the input trace sensitivity of the performance improvements obtained through CAT-based communication architectures, we performed an additional experiment. For the SYS example, we simulated the system with CAT-based and conventional communication architectures for three different input traces that had widely varying characteristics. Table 3 presents the results of our experiments. The parameters of the input traces were chosen at random to simulate run-time unpredictability. In all the cases, the system with a CAT-based communication architecture demonstrated a consistent and significant improvement over the system based on a conventional communication architecture. This demonstrates that the performance of CAT-based architectures are not overly sensitive to variations in the input stimuli.

6 Conclusions

This paper showed how the judicious addition of a layer of circuitry, called the Communication Architecture Tuner (CAT), around any existing communication architecture topology can enhance a system's capability of adapting to the changing communication needs

of its constituent components. We illustrated issues and tradeoffs involved in the design of CAT-based communication architectures, and presented algorithms to automate the process. From conducted experiment we conclude that performance metrics (e.g. number of missed deadlines, average processing time) for systems with CAT-based communication architectures are significantly (sometimes, over an order of magnitude) better than those with conventional communication architectures.

References

- D. D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*. Prentice Hall, 1994.
- R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test Magazine*, pp. 64–75, Dec. 1993.
- T. B. Ismail, M. Abid, and M. Jerraya, "COSMOS: A codesign approach for a communicating system," in *Proc. IEEE International Workshop on Hardware/Software Codesign*, pp. 17–24, 1994.
- A. Kalavade and E. Lee, "A globally critical/locally phase driven algorithm for the constrained hardware software partitioning problem," in *Proc. IEEE International Workshop on Hardware/Software Codesign*, pp. 42–48, 1994.
- P. H. Chou, R. B. Ortega, and G. B. Borriello, "The CHINOOK hardware/software cosynthesis system," in *Proc. Int. Symp. System Level Synthesis*, pp. 22–27, 1995.
- B. Lin, "A system design methodology for software/hardware codevelopment of telecommunication network applications," in *Proc. Design Automation Conf.*, pp. 672–677, 1996.
- B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: hardware-software cosynthesis of embedded systems," in *Proc. Design Automation Conf.*, pp. 703–708, 1997.
- P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign," in *Proc. Int. Symp. System Level Synthesis*, pp. 111–116, Dec. 1998.
- T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, pp. 288–294, Nov. 1995.
- J. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of system-level communication by an allocation based approach," in *Proc. Int. Symp. System Level Synthesis*, pp. 150–155, Sept. 1995.
- M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," in *ACM Trans. Design Automation Electronic Systems*, pp. 1–11, Jan. 1999.
- R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, pp. 437–444, Nov. 1998.
- "Sonics Integration Architecture, Sonics Inc. (<http://www.sonicsinc.com/>)."
- "VSI Alliance on-chip bus DWG. "On chip bus attributes specification" version v1.1.0 (<http://www.vsi.org/library/specs.htm>)."
- G. Borriello and R. H. Katz, "Synthesis and optimization of interface transducer logic," in *Proc. Int. Conf. Computer Design*, Nov. 1987.
- J. S. Sun and R. W. Brodersen, "Design of system interface modules," in *Proc. Int. Conf. Computer-Aided Design*, pp. 478–481, Nov. 1992.
- P. Gutberlet and W. Rosenstiel, "Specification of interface components for synchronous data paths," in *Proc. Int. Symp. System Level Synthesis*, pp. 134–139, 1994.
- S. Narayanan and D. D. Gajski, "Interfacing incompatible protocols using interface process generation," in *Proc. Design Automation Conf.*, pp. 468–473, June 1995.
- P. H. Chou, R. B. Ortega, and G. B. Borriello, "Interface co-synthesis techniques for embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, pp. 280–287, Nov. 1995.
- J. Oberg, A. Kumar, and A. Hemani, "Grammar-based hardware synthesis of data communication protocols," in *Proc. Int. Symp. System Level Synthesis*, pp. 14–19, 1996.
- R. Passerone, J. A. Rowson, and A. Sangiovanni-Vincentelli, "Automatic synthesis of interfaces between incompatible protocols," in *Proc. Design Automation Conf.*, pp. 8–13, June 1998.
- J. Smith and G. De Micheli, "Automated composition of hardware components," in *Proc. Design Automation Conf.*, pp. 14–19, June 1998.
- A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, N.J.: Prentice Hall, 1989.
- K. Lahiri, A. Raghunathan, and S. Dey, "Fast performance analysis of bus-based system-on-chip communication architectures," in *Proc. Int. Conf. Computer-Aided Design*, pp. 566–572, Nov. 1999.
- G.A.F. Seber, C.J. Wild, *Non-linear Regression*. Wiley, New York, 1989.
- K. Lahiri, A. Raghunathan, and S. Dey, "Performance analysis of systems with multi-channel communication architectures," in *vlsi*, pp. 530–537, january 2000.