Presented at the European Conference on Design Automation (EDAC), 1994

Synthesis of System-Level Bus Interfaces

Sanjiv Narayan and Daniel D. Gajski
Department of Information and Computer Science
University of California, Irvine, CA, 92717, USA

Abstract

Given a set of communication channels to be implemented as a single bus, we present a bus-generation algorithm which determines the width of a bus implementation. Tradeoffs between the width of the bus and the performance of the processes communicating over the bus are evaluated. The algorithm incorporates system level constraints such as data transfer rates and the number of pins and allows several channels that may be transferring different sizes of data to be implemented as a single bus. We demonstrate through a detailed example the usefulness of the algorithm in implementing system-level interfaces between modules.

1 Introduction

A system can be viewed as a set of *processes* which communicate with each other over abstract communication *channels*. System-level partitioning [1] groups processes and variables in the system specification into modules (representing chips and memories) and groups the channels to be implemented by buses. The set of tasks performed to implement communication between the modules in a system are collectively defined as *Interface Synthesis*.

In Figure 1, process A after system partitioning is mapped to two system modules. The variables MEM and $STATUS$ which were originally referenced within process A are now mapped to processes $A1$ and $A2$ respectively in a different module. Process A processes reads and writes data to the variable MEM over channels *ch1* and *ch2* respectively. $STATUS$ is accessed over channel $ch3$. Channels *ch1, ch2* and *ch3* have been grouped into bus B.

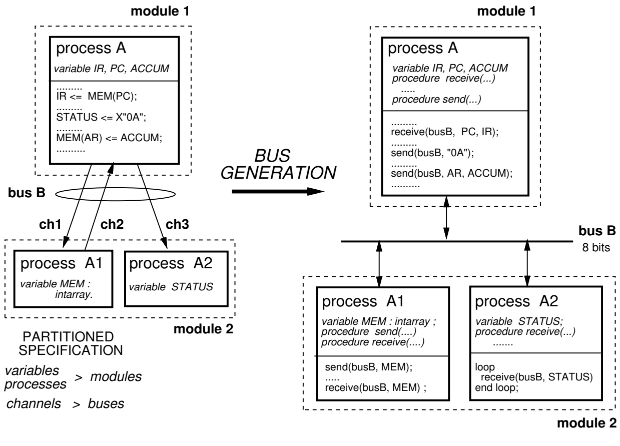

Figure 1: Bus generation in system design.

Bus generation is the interface synthesis task that determines the bus structure (i.e. number of data and control lines) for implementing a group of communication channels. Since it determines the buswidth, bus generation directly affects two critical design metrics: (1) **Performance** of the processes communicating over the bus, and (2) **Interconnect** cost of the modules, which is a measure of the number of wires or pins at the module boundary.

Most previous research efforts that have examined interfaces can be classified into two broad categories. In the first category are research efforts which have incorporated interface timing constraints into the scheduling of operations and events within the process during high-level synthesis. These include ISYN [2], CONSPEC [3], and Interface Matching [4]. The second set of approaches address issues relating to interfacing standard components which have incompatible protocols. Synthesis of transducers was presented in [5, 6] where, given timing diagrams of two incompatible interfaces, a template matching strategy is used to assign hardware to behavior. In [7], the two behaviors being interfaced were specified as Verilog based FSMs and a cross product of the two FSMs was optimized to obtain the description of the transducer.

Of the above approaches, only [4] addresses the issue of channel merging. The interface matching approach constructs and analyzes message dependency graphs with a view to reduce the synchronization overhead required for interprocess communication by minimizing the number of blocking messages between processes. Channel merging is implemented through rescheduling and serialization of communication. A process with branching or iteration constructs results in a hierarchical timing constraint graph. Since analysis of communication between two processes is restricted to a single graph at any level of the hierarchy in the processes, the number of cases to which the approach is applicable is limited. In addition, it is assumed that the size of data transferred over different channels is identical.

After system partitioning, we wish to synthesize a bus to implement communication between different processes communicating over abstract channels. Each channel may transfer data of different sizes. In addition, we seek to examine issues relating to design parameters external to the process (such as tradeoffs involving the number of pins or data transfer rates and the performance of processes). None of the above approaches address such issues.

The formulation of the bus generation problem and an algorithm to determine the buswidth are presented in the next two sections. Results of our experiments with bus generation are presented in Section 4. Finally, we present our conclusions and plans for future work.

2 Problem Formulation

2.1 Computing Bus and Channel Rates

Let $Width(B)$ refer to the number of data lines in bus B. Let $Delay(B)$ be the total delay associated with the protocol used by a process to transfer data over the bus. It includes all delays involved with implementing the handshaking between the processes and any access times needed for array variables implemented as memories. **Bus Rate** is defined as the maximum rate at which data can be transferred across the bus. The bus rate and the buswidth are related to each other as given by:

$$BusRate(B) \ = \ Width(B) \ / \ Delay(B) \tag{1}$$

The total execution time for any process consists of two components: (1) Computation Time, $CompTime(P)$, which is defined as the average start to finish execution time required by the process to perform all its internal computations. These computations represent almost any statement in the process: assignments, loops, conditional statements etc. (2) Communication Time, $CommTime(P)$, which is defined as the time spent by the process accessing data external to the process (i.e. variables which are in another process/module as a result of system partitioning).

Let $Access(P,C)$ represent the number of times process P will transfer data over channel C over its lifetime. Let $Bits(C)$ represent the number of bits received/sent over channel C in a single transfer. Any address bits required for accessing array variables over a channel are also included in $Bits(C)$.

$CompTime(P)$ can be determined by scheduling the basic blocks in the processes. Given the probability of branching at each `if, case, while` and `for loop`, we can apply flow analysis techniques to determine the average start to finish execution time for the process. $Access(P,C)$ is determined similarly by counting the number of times the variable accessed over channel C, is referenced within process P in the original description. $Bits(C)$, is inferred from the type of variable being accessed over channel C. Estimators for these quantities are presented in [8, 9].

For simplicity of explanation, assume that process P has exactly one channel C over which it transfers data. The communication time for a process is calculated as:

$$CommTime(P) = Access(P,C) \times \lceil \frac{Bits(C)}{Width(B)} \rceil \times Delay(B) \tag{2}$$

In case the buswidth is less than the number of bits in the data sent over the channel, several transfers (the term $\lceil \frac{Bits(C)}{Width(B)} \rceil$ in the above equation) may be required to implement the communication.

Channel Average Rate, $AveRate(P,C)$, is defined as the rate at which data is sent over channel C over the lifetime of process P communicating over it. In Figure 2, channel A sends 8 bits of data twice over the 4 second period. Thus, the average bitrate of channel A of 4 bits/second. The average data transfer rate over channel C, $AveRate(P,C)$ can be determined as:

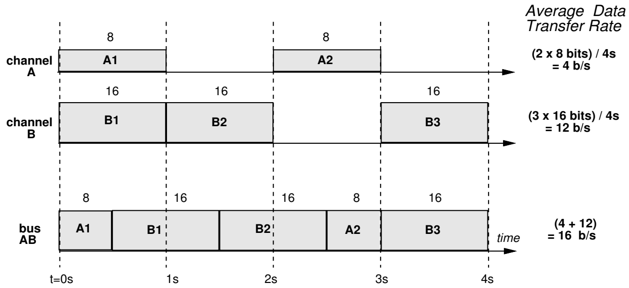

Figure 2: Merging channels A and B into bus AB.

$$AveRate(P,C) = \frac{Access(P,C) \times Bits(C)}{CompTime(P) + CommTime(P)} \tag{3}$$

Channel Peak Rate is defined as the rate at which a *single* data transfer takes place over the channel. In Figure 2, channel B sends 16 bits of data in one second in each transfer and thus has a peak rate of 16 bits/second.

2.2 Relating Bus and Channel Rates

In the most efficient implementation in terms of the minimum number of pins, *the bus is never idle* and has a 100% utilization. Consider two channels A and B shown in Figure 2. Assume that the 4 second time interval shown in the figure is representative of data transfer over the lifetimes of the processes which communicate over channels A and B. Channels A and B have average rates of 4 and 12 bits/second respectively. If channels A and B are implemented as a single bus AB, then the bus AB needs to send data at the rate of 16 bits/second to be able to satisfy the data transfer requirements of the two original channels A and B. While individual data transfers may be delayed due to bus access conflicts ($B2$ is delayed by 0.5 seconds in bus AB) the amount of data transferred over the individual channels before channel merging is *still* sent over the shared bus *in the same amount of time*.

In synthesizing the bus AB in Figure 2, we attempt to utilize the idle time slots of one channel for data transfers of other channels by synthesizing a bus over which data is *always* being transferred at a constant rate. If the channels before merging were transferring data at a certain average rate, they should be able to transfer the data over the bus at the *same average rate*. This can be achieved only if:

$$\boxed{BusRate(B) \ \ge \ \sum_{C \in B} AveRate(C)} \tag{4}$$

2.3 Constraints for Bus Generation

For a bus implementation, the designer can specify constraints and relative weights for the following parameters: (1) **Bus Width**: *Maximum/minimum* buswidth constraints that may be derived from the overall pin constraints specified for the modules (2) **Channel Average Rate**: Given performance constraints for a process communicating over a channel, the designer may budget the amount of spent for communication over the various channels, from which a *minimum* channel average rate constraint can be derived. This will ensure that the processes are not slowed down due to communication delays. A *maximum* channel average rate may be specified in cases when one of the processes communicating over the channel represents a slow device that is not capable of sending or receiving data faster than a certain rate. (3) **Channel Peak Rate**: To ensure that transfer of a single data item over the bus does not take an unnecessarily long time, a *minimum* peak channel rate constraint may be specified. Then,

$$\boxed{BusRate(B) > PeakRate_{min}(C), \quad \forall\ C \in B} \tag{5}$$

3 Bus Generation Algorithm

The Bus Generation algorithm determines the width of the bus required to implement a group of channels. The bus data and control lines are assumed to be disjoint. At any given instant, only one channel can transfer data over the bus. If the width of the bus is greater than the address and data bitwidths, then the address and data bits are sent simultaneously over the bus, otherwise they are sent separately in two distinct transfers. In the latter case, the address bits have to be latched in the receiving process. If the buswidth is smaller than the size of the data being sent over it, the data is sent in multiple transfers. All the processes which communicate over one of the channels in the bus are assumed to be have a synchronous implementation. Variables mapped to another module are modeled by a separate process which sends/receives variable values over the channel in response to requests from a process. Thus, in computing the performance of the main process, we assume that there are no synchronization delays involved in accessing these variables i.e. communication is non-blocking.

Intuitively, the algorithm examines a range of possible buswidths. For each buswidth, the bus rate and the channel average rates are computed. If the bus rate is greater than the sum of the channel average rates (as required by Equation 4), then we have a feasible bus implementation. From the set of feasible bus implementations corresponding to different buswidths, we select the one which has the least cost. In case no constraints are specified, a buswidth of 1 corresponding to a serial data transfer is selected.

The Bus Generation algorithm is summarized in Figure 3. Variables $MinBW$ and $MaxBW$ represent the range of bitwidths that are examined by the algorithm. $MinCost$ represents the minimum cost computed for all feasible implementations of the bus and $MinCostBW$ represents buswidth which corresponds to the minimum cost. The variable $CurrBW$ represents the current buswidth that is being evaluated by the algorithm. $AveRateSum$ is the sum of the channel average rates for a specific value of $CurrBW$. The algorithm has five basic steps:

```
Algorithm GENERATE BUS
begin
   if no constraints specified then return (1); endif
                                                  /* Step 1 */
   Compute MinBW and MaxBW (Equation 6)

   MinCost = ∞; MinCostBW = ∞;
   for CurrBW = MinBW to MaxBW do
                                                  /* Step 2 */
      Compute BusRate(B) for CurrBW (Equation 7)
                                                  /* Step 3 */
      AveRateSum = 0;
      for all channels C ∈ B do
         Compute AveRate(C) (Equation 3)
         AveRateSum = AveRateSum + AveRate(C);
      endfor
                                                  /* Step 4 */
      if (BusRate(B) > AveRateSum) then
         /* feasible solution */
         Compute Cost(CurrBW)
         if (Cost(CurrBW) < MinCost) then
            MinCost = Cost(CurrBW);
            MinCostBW = CurrBW;
         endif
      endif
   endfor

   if (MinCost = ∞)
      then No feasible solution. return(FAILURE).
      else return(MinCostBW);
   endif
end procedure
```

Figure 3: Bus Generation Algorithm.

<u>Step 1: Determine the buswidths to be examined:</u> The smallest and largest buswidths, $MinBW$ and $MaxBW$, examined by the algorithm are computed as:

$$MinBW = 1, \quad MaxBW = max(Bits(C)) \tag{6}$$

For each bitwidth, $CurrBW$, in the range determined above, repeat steps 2 and 3.

<u>Step 2: Compute bus rate corresponding to $CurrBW$:</u> The bus rate, $BusRate(B)$, can be calculated from Equation 1:

$$BusRate(B) \ = \ CurrBW \ / \ Delay(B) \tag{7}$$

<u>Step 3: Determine the channel average rates:</u> From Equations 2 and 3, compute the average rate of each channel:

If the bus rate, $BusRate(B)$, is greater than the sum of the average rates of all the channels (as required by Equation 4), then we have a **feasible implementation** for the bus. Go to step 4. Otherwise, the bus rate will not be able to satisfy the performance requirements of the channels. Go to Step 2 and try with the next buswidth in the range $< MinBW, MaxBW >$.

Step 4: Determine cost for $CurrBW$: The cost of a bus implementation is calculated as the sum of the squares of violations of each of the constraints (weighted by the relative weights specified for them).

Step 5: Select the buswidth: If there are more than one feasible solutions that were determined at the end of Step 3, select the buswidth corresponding to the one with the least cost. If there were no feasible solutions at the end of step 3 for all the buswidths examined, then an implementation for the group of channels is not possible. Any implementation for such a group of channels would progressively delay the processes communicating over the bus. Such a situation can arise when several channels that have very high average rate requirements are grouped together to be implemented as a single bus. One solution to this problem would be to split the group of channels further to be implemented by more than one bus. Alternatively, the lowest cost infeasible buswidth may be selected.

Channel C	Process P	Variable accessed	Bits(C)	Access(P,C)	Computation Time(P) (clocks)
ch1	EVAL_R3	trRu0	16 data + 7 addr = 23 bits	128	515
ch2	CONV_R2	trRu2	16 data + 7 addr = 23 bits	128	129

Figure 4: Channels in the Fuzzy Logic Controller

3.1 Example

We shall illustrate the Bus Generation algorithm with the example of a Fuzzy Logic Controller (FLC). Processes $EVAL\_R3$ and $CONV\_R2$ of the FLC access the array variables $trRu0$ and $trRu2$ respectively over communication channels $ch1$ and $ch2$ that have been merged to be implemented as a single bus. Figure 4 lists the data sizes, computation times and channel accesses for the two channels of the FLC. For the sake of simplicity, all execution times and data transfer rates are expressed in terms of *clocks*. For the bus implementation, assume that a minimum peak rate constraint of 10 bits/clock for channel $ch2$ with a relative weight of 10 has been specified.

Step 1: Both channels access array variables which have 128 words (7 address bits) of 16 bits each. From Equation 6: $MinBW = 1$ and $MaxBW = 23$. We will compute bus and channel rates for $CurrBW = 18$.

Step 2: For a handshake protocol, $Delay(B) = 2$ $clocks$. From Equation 7:

$$BusRate(B) = 18 \div (2) = 9\ bits/clock$$

Step 3: From Equations 2 and 3, we get:

$$AveRate(ch1) = (128 \times 23) \div [515 + (128 \times \lceil \tfrac{23}{18} \rceil \times 2)] = 2.86\ bits/clock$$

$$AveRate(ch2) = (128 \times 23) \div [129 + (128 \times \lceil \tfrac{23}{18} \rceil \times 2)] = 4.59\ bits/clock$$

Step 4: As $BusRate(B) > AveRate(ch1) + AveRate(ch2)$ we have a feasible solution for $CurrBW = 18$. We calculate the cost associated with this buswidth:

$$Cost = PeakRateCost(ch2)^2 = (10 \times (10 - 9))^2 = 100$$

Step 5: On repeating the algorithm with other values of the buswidth in the range $< 1, 23 >$, we get a graph of cost vs. buswidth as shown in Figure 5. From the graph, the minimum value of the cost function is 0 which occurs at a buswidth of 20. This is the buswidth that is selected for implementing the bus consisting of channels $ch1$ and $ch2$.

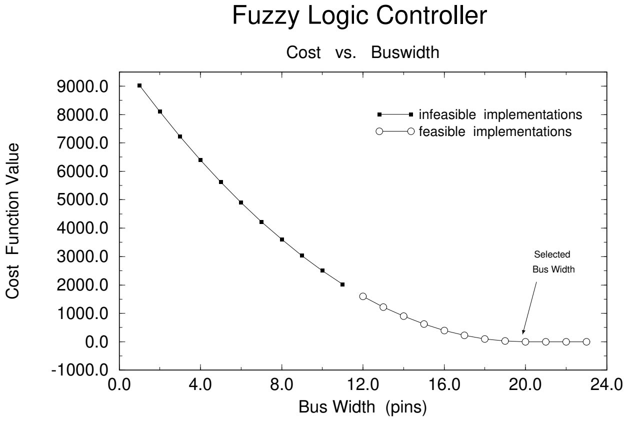

Figure 5: Determining a minimum cost buswidth.

4 Experiments and Results

Bus generation presented in this paper has been implemented and integrated with the SpecSyn system-level partitioner [1]. For a selected buswidth, VHDL *send/receive* procedures defining the data transfer over the bus, are generated for each channel. Accesses to a variable in the processes are replaced by the appropriate send/receive procedure call. For more details on protocol generation, refer to [10]. We performed several experiments involving the application of the bus generation algorithm on an Ethernet network coprocessor, a bladder volume controller, and the fuzzy logic controller. We will present experiments for the fuzzy logic controller (FLC) in greater detail.

The FLC consists of two inputs which sense the temperature and humidity in a room and evaluates four rules to control the operation of an air-conditioning system. System partitioning mapped the processes and the array variables which store the membership functions and fuzzy logic rules to different modules, thus creating several channels between the two modules. We restrict our experiments to bus B, formed by merging the channels in Figure 4.

Figure 6 shows how the performance of the two processes transferring data over bus B is affected by the various buswidths that can be used to implement the bus. For each buswidth, we generated the protocol required for all the channels in the bus. A performance estimator [9] was then used to obtain the execution times of the processes. Clearly, as the buswidth increases, the execution time for the processes decreases. Given a performance constraints for these processes, the designer can select an appropriate width for implementing the bus. For example, if process $CONV\_R2$ has a maximum execution time constraint of

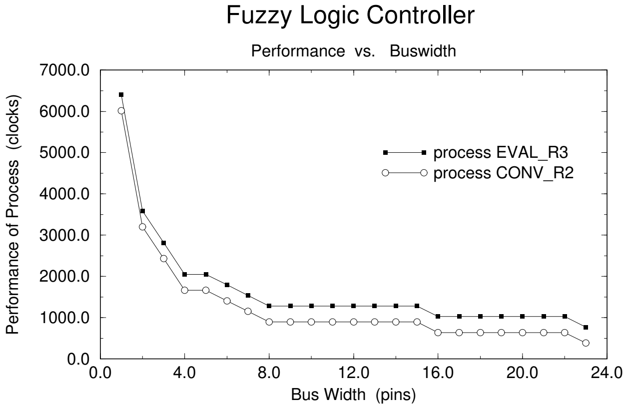

Figure 6: Effect of buswidth on process execution times.

2000 clocks, then only buswidths greater than 4 bits need be considered for implementing the bus.

To demonstrate how the designer can exercise control over bus generation by specifying appropriate constraints and weighing them accordingly, we ran the Bus Generation algorithm for three sets of constraints. Figure 7 shows the bus constraints, selected buswidths, corresponding bus rates for the three bus designs A, B and C.

In **design A** of Figure 7, a minimum peak rate for channel $ch2$ of 10 bits/clock was specified. The range of possible buswidths and the cost associated with each buswidth is shown in Figure 5. Feasible bus implementations are those which have a bus rate greater than the sum of the individual channel average rates. The minimum cost function value corresponds to a buswidth of 20 which can then be used to implement the bus. Going back to Equation 7, we can see that a buswidth of 20 pins indeed corresponds to a bus rate of 10 bits/clock. The reduction in the number of data lines compared to the case when the two channels would have been implemented as *separate* buses is 56%. This reduction was achieved *without sacrificing any performance of the processes.* In **design B** of Figure 7, minimum and maximum buswidth constraints with a relative weight half as that for the peak rate constrain for $ch2$ have been specified. A minimal cost function value occurs at a buswidth of 18. Finally, consider **design C** in Figure 7. Let's assume that the designer wishes to implement the channels $ch1$ and $ch2$ using a 16 bit bus. To achieve this, *both* the minimum and maximum buswidth constraints have been specified to be 16 bits and given very high relative weights. As shown in Figure 7, the

Design	Bus Constraints (relative weight)	Total Bitwidth of channels (pins)	Selected Bus Rate (bits/clock)	Selected Buswidth (pins)	Interconnect reduction (%)
A	Min PeakRate (ch2) = 10 bits/clock (10)	46	10	20	56 %
B	Min PeakRate (ch2) = 10 bits/clock (2) Min BusWidth (B) = 14 bits (1) Max BusWidth (B) = 16 bits (1)	46	9	18	61 %
C	Min PeakRate (ch2) = 10 bits/clock (1) Min BusWidth (B) = 16 bits (5) Max BusWidth (B) = 16 bits (5)	46	8	16	66 %

Figure 7: Effect of constraints on Fuzzy Logic Controller bus.

Bus Generation algorithm selects 16 bits as the buswidth since it has the minimum cost function value of all feasible implementations.

5 Conclusions and Future Work

In this paper we have presented a Bus Generation algorithm which determines the width of the bus required for implementing a group of communication channels while minimizing performance degradation of the processes in the system. The algorithm incorporates system level constraints such as the data transfer rates of the individual channels and the number of pins available for implementing the bus. The algorithm allows the designer to explore tradeoffs between the buswidth and performance of the processes communicating over the bus.

The work presented in this paper can be extended in several ways. Since simultaneous data transfer over two channels mapped to the same bus will result in bus access conflicts, an arbiter scheme needs to be generated. Work is needed to examine the effect of bus arbitration delays on the performance of processes. We are also studying generation of communication protocols other than the full-handshake that is currently implemented.

6 Acknowledgements

We would like to thank Frank Vahid for his valuable assistance in defining system design issues and integration of the SpecSyn system partitioner with the interface synthesis tools. We would also like to acknowledge the support of the Semiconductor Research Corporation (grant #92-DJ-146) and the helpful suggestions of the reviewers.

References

[1] F. Vahid and D. Gajski, "Specification Partitioning for System Design," in *DAC*, 1992.

[2] J. Nestor, *Specification and Synthesis of Digital System with Interfaces.* PhD thesis, Carnegie Mellon Unversity., April 1987.

[3] S. Hayati and A. Parker, "Automatic Production of Controller Specifications from Control and Timing Behavioral Descriptions," in *DAC*, 1989.

[4] D. Filo, D. Ku, C. Coelho, and G. DeMicheli, "Interface Optimization for Concurrent Systems under Timing Constraints," in *IEEE Transactions on Very Large Scale Integration Systems*, pp. 268–281, September 1993.

[5] G. Borriello and R. Katz, "Synthesis and Optimization of Interface Transducer Logic," in *ICCAD*, 1987.

[6] G. Borriello, *A New Interface Specification Methodology and its Applications to Transducer Synthesis.* PhD thesis, University of California, Berkeley, May 1988.

[7] J. Akella, *Input/Output Performance Modeling and Interface Synthesis in Concurrently Communicating Systems.* PhD thesis, Carnegie Mellon Unversity., November 1991.

[8] S. Narayan and D. Gajski, "System Clock Estimation based on Clock Slack Minimization," in *EuroDAC*, 1992.

[9] S. Narayan and D. Gajski, "Area and Performance Estimation from System-Level Specifications." UCI Tech. Report ICS-92-16,1992.

[10] S. Narayan and D. Gajski, "Synthesis of System-Level Bus Interfaces." UCI Tech. Report ICS-93-28,1993.