Bus Architecture Synthesis for Hardware-Software Co-Design of Deep Submicron Systems on Chip

Nattawut Thepayasuwan, Vaishali Damle, Alex Doboli
Department of Electrical and Computer Engineering
State University of New York at Stony Brook
Stony Brook, NY, 11794-2350
{nattawut, vaishali, adoboli}@ece.sunysb.edu

Abstract

System level design always has a disadvantage of not possessing detailed knowledge of the communication sub-system. This is a crucial issue for System-on-Chip design, where uncertainty in communication by very deep submicron effects cannot be neglected. This paper presents a bus architecture (BA) synthesis algorithm for designing the communication sub-system of an SoC. The algorithm is part of a hardware-software co-design methodology for resource constrained embedded applications. BA synthesis includes finding the bus topology, and routing the individual buses so that various constraints, like bus length, topology complexity, potential for communication conflicts over time, are addressed. The paper presents BA synthesis results for a network processor, and a JPEG SoC.

1 Introduction

Systems on chip (SoC) include multiple IP cores connected through complex data, address and control buses. The variety of IP cores is large. For example, the H.263 encoder includes digital DSPs and microprocessor cores, memories, RF and mixed-signal blocks (analog-to-digital and digital-to-analog converters, filters, and PLL circuits). Over the next 4-7 years, it is foreseen that the number of SoC cores will steadily increase, while clock frequencies will range around 10-15 GHz [4]. For very deep submicron designs (VDSM), physical-level attributes such as interconnect parasitics, substrate coupling, and substrate noise significantly influence system performance, e.g. bus communication speed, system latency, power consumption, and signal integrity [5] [21]. The performance of analog cores, such as the dynamic range of analog-to-digital converters, is notably affected by the digital switching noise originated at the digital cores. Hence, the usual design paradigm of abstracting physical details during system level design is of limited use for SoC implemented in a VDSM process. To successfully address the myriad of emerging challenges, research must focus on incorporating detailed layout level elements into system-level synthesis, including hardware/software co-design.

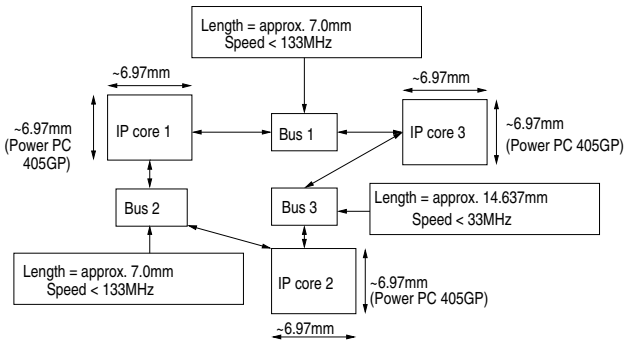


Figure 1. Impact of layout design on bus speed

The motivating example illustrates the impact of layout design on data communication speed. Figure 1 shows a set of three Power PC 405GP cores [2], which exchange data as part of their functionality. The physical dimensions of the cores are presented in the figure. Without considering layout information, the co-design step decides to allocate a single system bus of speed 266MHz for all core communications. All timing constraints were met in this case. However, given the physical dimensions of the cores, it is difficult to implement a bus with the requested speed. The same latency performance can be obtained with three buses of lower speed, as shown in the figure. The bus speeds of 133MHz, 133MHz and 33 MHz were found based on the physical locations of cores, and the RLC effects of the routed buses [21]. This example argues that the communication sub-system of an SoC needs to be designed while contemplating design feasibility criteria (e.g., achievable bus speeds). Task and communication scheduling must be integrated with core placement, bus speed allocation, bus topology design and routing to guarantee that identified bus speeds are possible to be realized in a real SoC.

This paper presents a bus architecture (BA) synthesis algorithm for designing the communication sub-system of an SoC implemented using a VDSM process. This is important, because it is difficult to postulate a unique BA as optimal for various applications and performance requirements. Instead, BA

must be customized depending on the application specifics and design needs. The proposed algorithm is part of a hardware-software co-design methodology for resource constrained embedded applications, like PDA, or mobile sensor-based systems. BA synthesis includes finding the bus topology, and routing the individual buses. The BA synthesis algorithm identifies the set of possible building blocks (using the proposed PBS bitwise generation algorithm), and assembles them into a topology using simulated annealing (SA) [18] as an exploration engine. The cost function models bus length, bus topology complexity, potential for communication conflicts over time, and amount of unnecessary core connectivities present in a topology (unnecessary connectivity needlessly increases bus length and decrease bus speed). We propose a special table structure (called bus architecture synthesis table) and select-eliminate method to prune solutions, which are dominated by others. For example, buses with complex and redundant connectivity are removed. This effectively reduces the exponential number of possible bus architectures. The paper offers results of BA synthesis for a network processor using CoreConnect bus architecture [3], and for a JPEG SoC design.

Bus design and routing are critical research problems for SoC design. Early work on bus and communication synthesis [8] [13] [17] [22] focuses on multiprocessor embedded systems. Research addresses interface design [8] [17], communication packeting, mapping and scheduling [22]. However, this work does not tackle details of hardware implementation and layout design. In many approaches, bus topology is assumed as given [6] [13] [14] [15]. Finding a very good bus architecture is a tedious task for SoC with large number of cores, and involves extensive designer knowledge. Seceleanu *et al* [19] discuss segmented bus architectures for improving speed and power consumption. Bus segments are dynamically linked through inter-segment bridges (glue logic + tri state buffers) under the control of a Central Arbiter. Hu *et al* [14] introduce point-to-point (P2P) communication synthesis to optimize energy consumption and area. Their work concentrates on buswidth synthesis to meet timing constraints on the communication links, and floorplanning to minimize energy consumption and total SoC area.

Dick *et al* [9] discuss bus synthesis as a part of the core-based synthesis tool, MOCSYN. Floorplanning and placement are iteratively conducted inside a feedback loop to obtain point-to-point delay estimation. The delay information is used for recalculating link priorities of a core graph. Our methodology differs since it does the floorplanning and placement independent of the feedback loop. Therefore, our methodology is considerably faster. Also our algorithm does not require the number of buses as an input constraint [9], since it can be only determined after the optimized bus structure is available.

More recently, Drinic *et al* [11] [16] present a method for SoC bus network design to maximize overall processing throughput. The communication architecture includes shared buses connected through bridges. The design flow includes two steps: one produces the communication topology, and the other finds the core floorplanning. Our work partially resembles the methodology of Drinic *et al* [11]. However, it differs in considering more layout knowledge for BA synthesis, and suggesting a new pruning method to remove poor quality BA. This is important, because the BA with the highest speed (thus, best data communication speed) critically depends on IP core placement. Also, the solution space is difficult to traverse without good pruning.

The paper is organized as follows. Section 2 presents the layout-aware co-synthesis flow. Section 3 discusses the modeling for BA synthesis. Section 4 presents BA synthesis algorithms. Experimental results are given in Section 6. Finally, we put forth our conclusions.

2 Layout-aware Codesign Methodology

Figure 2 presents our layout aware hardware-software co-synthesis methodology for SoC. Goal is to find hardware-software implementations that minimize overall system latency. Systems are described as hierarchical data and control graphs (HDCG) [10]. HDCG include cluster nodes for expressing software tasks and procedures, operation nodes for hardware operations, and communication cluster nodes for data communications. All available hardware resources are known, including general-purpose processors (GPP) for executing software, and functional units - FUs (like adders, multipliers, shifters, etc) for hardware. The communication architecture of an SoC is found as part of the co-design process. The proposed methodology is novel in that it offers superior usage of hardware resources by allowing resource sharing across different tasks. Also, the methodology accounts for critical physical aspects, like core placement, bus routing, and interconnect parasitic, when identifying the bus speed requirements and data communication times for a design. This improves SoC design feasibility.

The co-design methodology includes three subsequent steps:

- The first step performs combined task partitioning to GPP, operation binding to FU, and task and communication scheduling. It also identifies minimum speed constraints for each data communication, if a dedicated point-to-point channel is used between any pair of communicating GPPs or ASIC. Combined exploration uses a simulated annealing algorithm (SA). Exploration is guided by Performance Models (PM) [10]. PMs are graphs that capture the relationship between latency, graph characteristics (like data and control dependencies), and design decisions, such as binding and scheduling. Experiments showed that PM are more efficient than

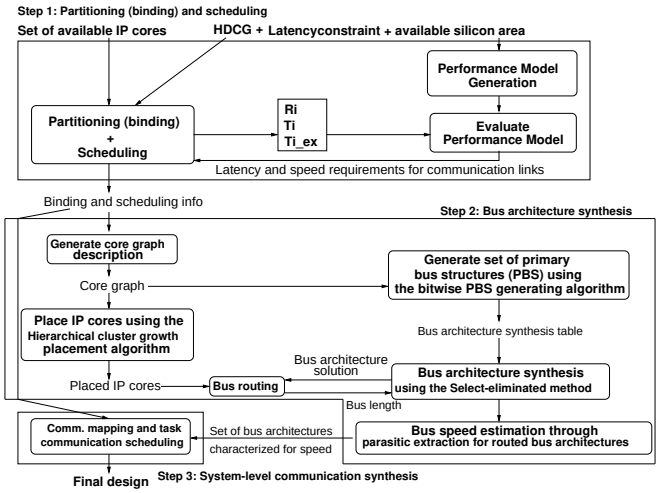


Figure 2. Layout-aware co-synthesis

other metrics, like task priorities, forces, concurrency degree, utilization rate [12]. PM are flexible, and can be easily updated to incorporate new design aspects, without requiring cumbersome validation. The first step ends with generating a Core Graph structure (CG) (defined in Section 3) that expresses the volume and timing for data communications on each link.

- The second step uses CG to synthesize and route bus architectures for an SoC. IP cores are placed using a hierarchical cluster growth algorithm [20], which places highly communicating cores close to each other. Bus architecture synthesis first identifies a set of possible building blocks (using the PBS bitwise generation algorithm), and then assembles them together using SA. Dominated solutions are pruned using bus architecture synthesis tables and select-eliminate method. Each bus architecture is routed, and individual buses are pre-characterized for their speed after parasitic extraction.
- The third step identifies the best bus architecture to be used, binds individual data communications to buses, and re-schedules tasks, operations and communications to minimize system latency. In this step, tasks are not re-partitioned, and operations are not re-bound. Bus speeds are described by values found at Step 2.

3 Modeling for Bus Architecture Synthesis

Definition: A *core graph* (CG) is a graph (V,E), where $v_i \in$ V represents the i^{th} core in the architecture, and $e_{ij} \in$ E is the communication link between core *i* and core *j*. This concept has been illustrated in Figure 3(a). The weight w_{ij} is the communication load between core *i* and core *j*. The communication load (Cl) expresses the amount of data exchanged between the two cores. The core size h_i x w_i is described along with node v_i.

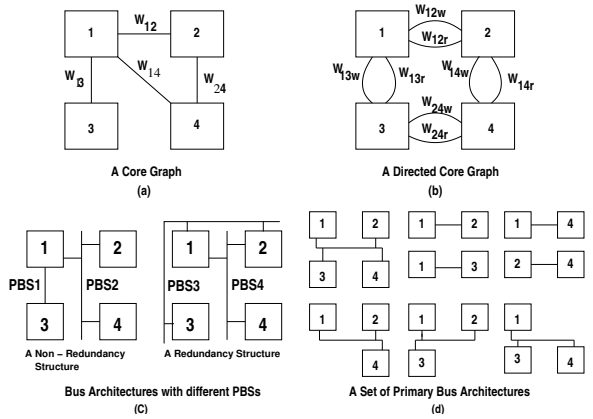


Figure 3. Core Graph and PBS examples

The core graph representation of a system architecture is used for BA synthesis. For simplicity of modeling, CG do not distinguish between unidirectional and bi-directional dataflow. Communication direction depends on whether an operation is a read or a write, and it isn't specified directly in a CG. However, the core graph can be modified in order to address the direction of data. Figure 3(b) presents the core graph for bi-directional communications. In case there is more than one communication channel between two cores, then the communication load is split across the channels. This type of CG will have multiple channels between cores, and channels might get synthesized as separate buses in an architecture.

Industrial bus standards can be expressed using the CG formalism. The superposition principle can be applied, if an industrial bus standard is used at the transaction level (post-topology synthesis). This can be done by classifying links according to their bandwidth (low, medium, and high), and generating core graphs for each category. This idea is well defined because most of the industry standards for SoC, i.e., AMBA [1] and IBM CoreConnect [3], have different buses to support data communication at different bandwidth. After bus architecture synthesis, sub-bus architectures associated with its core graph are combined together to get a finalized optimal solution.

Definition: Primary bus structure (PBS) is defined as a *potential* cluster of connected cores. PBS are the building blocks for bus architecture synthesis. A PBS is *valid*, if its node connectivities exist in the original CG. Otherwise, it is *invalid*.

Figures 3(c) and 3(d) show eight PBS for the CG in Figure 3(a). PBS in Figure 3(c) are valid. PBS are characterized by following physical and topological properties:

- *PBS utilization percentage*: Utilization is defined as the communication spread in a structure. For example, a PBS

```
Let L = {e_{m-2}, e_{m-1}, ..., e_2, e_1, e_0 }
for (i = 0 ; i < 2^{DIM(L)} -1; i++)
    {
     Output basis set = decoder(i);
     PBS = OR(Output basis set);
     if (validate(PBS)==1){
       result = COMPARATOR (PBS , PBS Storage)
       if (result == 0) table update (PBS);
       }
    }
```

Bitwise PBS Generating Algorithm

Figure 4. Bitwise decoder and bitwise PBS generating algorithms

corresponds to two links in the CG, i.e., l_{12} and l_{13}. This PBS can also contain l_{23}, the connection between core 2 and core 3. There might, however, be no communication between these cores. Therefore the PBS under-uses its structure. We can also consider the unused element l_{23} as a redundant link of the PBS. The PBS utilization percentage, P_u, is defined as $P_u = \frac{2N_b}{n(n-1)} \times 100\%$, where N_b is number of links in a PBS, and *n* is number of associated cores in a PBS. The maximum PBS utilization occurs when all associated cores communicate between each other, and the PBS corresponds to a clique in the CG.

- *Communication conflict:* A PBS is implemented as a shared bus in the system architecture. Performance of a bus architecture can be evaluated by its contention. For a static time scheduling of tasks, it is important to evaluate if there is a communication conflict in a PBS. Communication conflict of a PBS, $C_{conflict}$, is defined by the amount of time overlap between communications mapped to the same link.
- *PBS bus length*: PBS bus length is a vital attribute for evaluating the bus speed in the presence of very deep submicron effects. Longer buses require more silicon area and additional circuitry like bus drivers [5] [21]. Also, larger cross coupling and parasitic capacitances of longer buses increase interconnect latency [21]. Larger power dissipation for interconnect and drivers can be caused by longer buses. It is, however, difficult at the architecture level to estimate PBS bus length with high accuracy without contemplating the SoC layout. As explained in Section 2, hierarchical cluster growth placement is used for placing IP cores, and then estimating PBS bus lengths.

Identifying the set of valid PBS has an exponential complexity, if a brute-force algorithm is applied. The upper bound of the total number of PBS is $P_m = 2^l - 1$, where l and P_m represent the number of links in a CG and the maximum possible number of PBSs, respectively. We suggest a more efficient, bitwise algorithm to generate the set of valid PBSs. The algorithm is presented in Figure 4. First, using the bitwise decoder algorithm, each link label is translated into binary, and stored as a set of *basis elements* (a basis element is a link in the CG). Then, in a loop, the bitwise PBS generating algorithm performs a bitwise OR operation on the basis elements to generate new PBS structures. A produced PBS is valid, if and only if all its basis elements are connected. Otherwise, the PBS has redundant links. If the PBS is valid, the PBS storage is checked to avoid duplications of the same PBS.

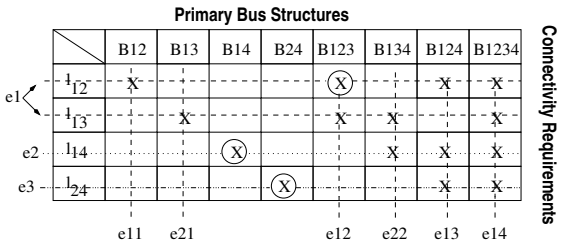
Example: The core graph in Figure 3(a) has 4 cores. Binary numbers are used to represent links e_{ij},i.e., e_{12} is described as "0011". The number of bits is equal to number of cores (the first core has the right most digit, while the last core has the left most digit). In this case, there are 4 basis elements in the PBS set, namely, e_{12}, e_{13}, e_{14}, and e_{24} labeled in order. Therefore the basis set is B = $\{$(1,"0011"),(2,"0101"),(3,"1001"),(4,"1010")$\}$, where the first coordinate is the label of the basis element. Bitwise PBS generating algorithm starts with index 0 and empty PBS storage. All basis elements are added as separate PBS into the PBS storage. Considering index 3, this is decoded into "0011". Therefore, the PBS has two basis elements, namely, (1,"0011") and (2,"0101"). This PBS is valid because all the basis elements are connected. The PBS is then validated with the PBS storage. Storage is updated if there is no such PBS.

Definition: *Bus architecture synthesis table* is a table describing the relationship between a set of PBS and connectivity requirements in a CG. Number of rows is similar to number of basis link elements in the CG. Number of columns is the dimension of the PBS set. An entry in the table has value "X", if the PBS corresponding to the column includes the basis link element specific to the row.

Figure 5 presents a bus architecture synthesis table.

4 Bus Architecture Synthesis

Depending on their structure, bus architectures can be either *non-redundant* or *redundant* structures, and *flat* or *hierarchical*. The core connectivity offered by a non-redundant bus tightly matches the nature of the communication links in the CG of the application. Also, there is a single connection through a bus for any CG link. There are no core connections, which do not correspond to a link. Non-redundant structures have the benefits of using minimal resources for offering the needed core connectivity. They require no additional control circuitry (like for segmented buses), because a single channel is used to communicate between any two IP cores. The structure is simple (thus simplifies the bus routing step), but lacks the concurrency advantage. In contrast, redundant structures have superior concurrency, and thus decrease communication conflicts. However, expensive control logic is required to intelligently drive the shared bus. In flat (non-hierarchical) bus




```
L = number of basis link elements;
i = 1;
do  selecting a structure to satisfy connectivity i
      eliminate candidate structures;
      pre-eliminate other candidates of other connectivities;
   i = i + 1;
 until   i > L
```

Figure 5. Select-eliminate algorithm

architectures there are no bus-to-bus communications, as buses link only cores. Hierarchical bus architectures include bus-to-bus communications through bridge circuits [3]. Examples of non-redundant, non-hierarchical (NRNH) and redundant, non-hierarchical (RNH) bus architectures are given in Figure 3(c). The left NRNH bus architecture in Figure 3(c) uses the shared bus B_{124} to serve communications links l_{12},l_{14},l_{24}, and the point-to-point bus B_{13} for the link l_{13}. The right part of Figure 3(c) shows a RNH bus architecture, in which two buses can be used to implement the link l_{12}. For the NRNH structure, given a destination address, bus selection is statically assigned by the core-bus interface controller. Its routing area is less compared to the redundant structure. The NRNH structure leaves concurrency exploration to task re-scheduling at the system level (similar to Step 3 in the methodology in Figure 2).

We consider only non-redundant, non-hierarchical (NRNH) bus architectures. This is motivated by our goal of designing resource constrained SoC with minimal architectures (thus minimal bus architecture) and software support. The modeling for bus architecture synthesis (including CG, PBS and BA synthesis tables) supports the other three bus architecture classes. We propose the select-eliminate (SE) algorithm to generate NRNH bus architectures based on the satisfaction of the core connectivity requirements. SE algorithm is represented in Figure 5. To illustrate the algorithm, we use the simple BA synthesis table in Figure 5. For example, to satisfy the l_{12} connectivity, one of the four PBS $\{B_{12}, B_{123}, B_{124}, B_{1234}\}$ has to be chosen. Suppose PBS B_{123} is chosen, the rest of the candidates must be eliminated, so that there is no redundancy in the final structure. The horizontal dash line S_1 represents eliminated structures. Once a structure is eliminated, it automatically voids the whole column. Vertical dash lines e_{11},e_{12},e_{13},e_{14} show the eliminated column. PBS B_{123} does not satisfy only the l_{12} and l_{13} connectivity. Therefore, another horizontal line S_2 is created with vertical lines e_{21} and

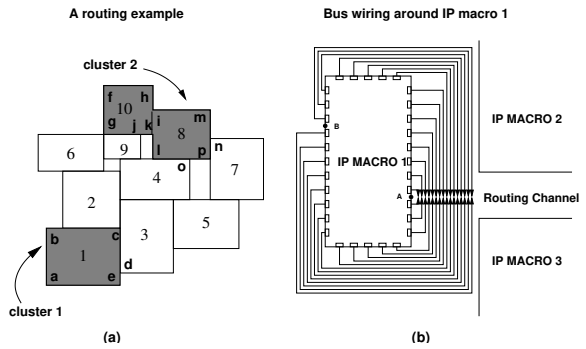


Figure 6. Routing example

e_{22}. Connectivity l_{14} is considered next. There is a candidate left, namely, PBS B_{14}. Once PBS B_{14} is chosen, we have only one candidate, PBS B_{24}, left to satisfy l_{24} connectivity. It is noticed that no more horizontal lines or vertical lines are created because there is no structure existing in a table. The generated NRNH bus architecture is composed of PBS B_{124}, PBS B_{14}, and PBS B_{24}. Circled structures in Figure 5 show the final bus architecture.

The size of a synthesis table grows depending on the number of cores and the amount of inter-core communication. If number of cores and interconnects between them is small, the SE algorithm contemplates all possible coverings of the CG links using the available PBS structures. However, if a system consists of more than 20 cores intensively tied up together, the exhaustive SE algorithm becomes infeasible. To allow SE algorithm to explore the PBS candidate space efficiently, we employed a simulated annealing algorithm to search different candidate PBSs while satisfying connectivity requirements. The algorithm randomly chooses a PBS from each requirement row, and combines it into a bus architecture. The total cost function that guides simulated annealing is given by the formula

$$\textit{Total cost} = w_l L_t + w_n N_b + w_c C_c + w_{ml} M_l - w_u C_u,$$

where L_t is the total bus length, N_b is the number of shared buses with coefficient, C_c is the communication conflict, M_l is the maximum loss, and C_u is the total bus utilization. w_l, w_n, w_c, w_{ml}, w_u are weight factors. The cost function parameters are defined in Section 3. Maximum loss reflects the maximum data loss in a bus architecture, if there is a conflict in a particular PBS. The algorithm objective is to minimize this cost function.

Buses are routed based on the IP core placement. The layout is described as an intersection graph [20] to diminish the size of the routing space. Depending on the IP vendor, a number of metal layers is available. To easy the analysis of deep submicron effects, inter-macro communication uses only routing channels along the macro borders. To calculate the bus length of a PBS, inter-core routing path and bus wiring on a macro are

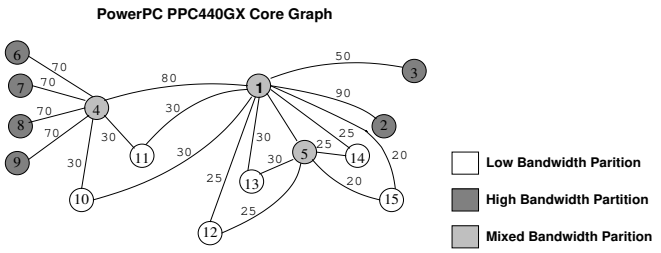


Figure 7. Core graph for the network processor

taken into account. Inter-core routing algorithm starts by finding the shortest path of the link with the lowest communication load. This strategy is motivated by the placement method always placing two highly communicating cores close together. This implies that the longest path between two cores is at the link with the lowest communication load. Therefore, this link influences the total bus length of a PBS. We thus give it the highest priority to find the shortest path. Pin assignment along a core is considered for routing.

The shortest path between two cores is defined as the shortest path between any pair of intersection nodes around the border of two clusters. An intersection graph is illustrated in Figure 6(a). Cluster 1 contains core 1, with the interconnect set {a,b,c,d,e}, and cluster 2 contains cores 8 and 10 with the interconnect set {f,g,h,i,j,k,l,m,n,o,p}. The shortest path between clusters 1 and 2 is the shortest path from any two nodes of the two interconnect sets. If an intersection point of a core is chosen, all the wires which connect to every pin have to route around the core from the pin to a chosen intersection point. As seen in the bus routing diagram in Figure 6(b), the bus length is approximately half of the core perimeter. Therefore, if there is also another intersection point at point B, which is the longest distance from point A, all the wires have to route against the original direction. That is the total bus length is approximately a perimeter. This is considered to be the worst case, however, it is a good approximation if a macro has several intersection points required for inter-core routing. We use this worst case approximation in our bus length calculation. Lets l_{PBSi} be the bus length of the i^th PBS, l_{inter} be the inter-core bus length of the i^{th} PBS, and l_{macros} be the length of the bus-wiring around the cores associated with i_{th} PBS, the PBS bus length is defined as $l_{PBSi} = l_{inter} + l_{macros}$.

5 Experimental Results

The first experiment presents the BA synthesis results for a network processor [7]. The processor receives Internet packets, reroutes them, and sends them out. A packet arrives through an Ethernet media access controller interface core (EMAC), and is sent to the multi-channel memory access layer core (MCMAL), a specialized DMA controller. MCMAL stores the packet in a buffer, and then transmits the buffer descriptor to the processor. The processor calculates the new destination

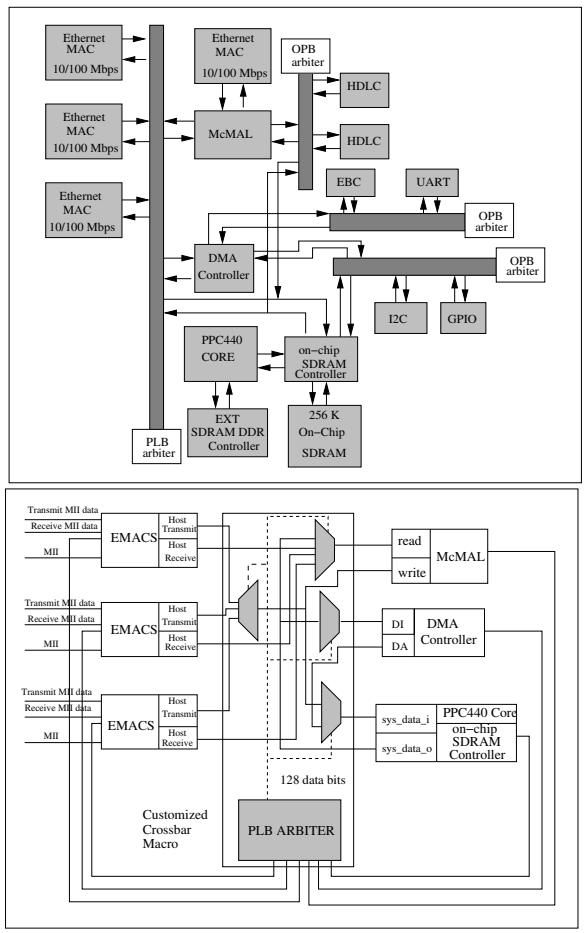


Figure 8. Bus architecture for network processor

address for the packet. MCMAL and one of the EMAC will send the packet out. Figure 7 shows the core graph for the network processor. Node 1 corresponds to core for the Power PC, on-chip SRAM, and SRAM controller. Node 2 is the DDR-SRAM controller. Node 3 is the PCI-X core, node 4 is the MCMAL core, and node 5 describes the direct memory access (DMA) core. Nodes 6-9 represent EMAC cores. Nodes 10 and 11 are the high-level data link controller (HDLC) core. Node 12 is the inter-IC (I^2C), node 13 describes the universal asynchronous receiver/transmitter (UART) core, node 14 the general purpose input/output (GPIO) core, and node 15 is the external bus controller (EBC) core.

Table 1 summarizes the BA synthesis results. Columns 2-5 present the weight factors for bus length, number of buses in the architecture, communication conflicts, and bus redundancies. Different design goals were modeled using the four weight factors. Column 6 shows the resulting bus length. Number of buses in an architecture is indicated in Column 7. Column 8 presents the resulting redundancy. The amount of communication conflicts for a BA is given in Column 9. Fi-

	w_l	w_n	w_c	w_r	Length	# buses	Redun.	Conflicts	loss		w_l	w_n	w_c	w_r	Length	# buses	Redun.	Conflicts	loss
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
1	0.1	0.7	0.1	0.1	0.84	8	0.159	0.13	0.3	13	0.1	0.1	0.7	0.1	0.94	16	0.08	0.0	0.0
2	0.5	0.7	0.1	0.1	0.86	9	0.104	0.11	0.3	14	0.5	0.1	0.7	0.1	0.94	18	0.03	0.0	0.0
3	1.0	0.7	0.1	0.1	0.83	7	0.128	0.43	0.2	15	0.1	0.1	1.0	0.1	0.95	15	0.08	0.0	0.0
4	0.1	1.0	0.1	0.1	0.83	7	0.104	0.22	0.3	16	0.5	0.1	1.0	0.1	0.91	16	0.07	0.0	0.0
5	0.5	1.0	0.1	0.1	0.85	8	0.140	0.25	0.45	17	0.1	0.1	1.0	0.5	0.95	19	0.01	0.0	0.0
6	1.0	1.0	0.1	0.1	0.84	9	0.138	0.14	0.31	18	0.5	0.1	1.0	0.5	0.94	18	0.03	0.0	0.0
7	0.1	0.7	0.1	0.5	0.83	9	0.114	0.29	0.28	19	0.1	0.5	1.0	0.1	0.96	14	0.19	0.0	0.0
8	0.5	0.7	0.1	0.5	0.86	8	0.183	0.38	0.32	20	0.5	0.5	1.0	0.1	0.99	14	0.17	0.0	0.0
9	1.0	0.7	0.1	0.5	0.84	7	0.132	0.11	0.26	21	0.1	0.5	0.7	0.5	0.96	15	0.07	0.0	0.0
10	0.1	1.0	0.1	0.5	0.87	8	0.154	0.13	0.28	22	0.5	0.5	0.7	0.5	0.93	15	0.12	0.0	0.0
11	0.5	1.0	0.1	0.5	0.84	7	0.166	0.43	0.37	23	0.1	0.5	1.0	0.5	0.93	15	0.11	0.0	0.0
12	1.0	1.0	0.1	0.5	0.86	7	0.154	0.29	0.37	24	0.5	0.5	1.0	0.5	0.95	14	0.16	0.0	0.0

Table 1. Results for bus architecture synthesis for the network processor example

nally, the maximum data loss is shown in Column 10.

First twelve rows in Table 1 correspond to the design scenario in which the bus complexity is minimized, while timing is less important. The weight factor for number of segments has high values. Weights for communication conflicts and redundancies are low. The weight for bus length is varied from small to large values. Bus complexity minimization will favor shared buses, and discourage usage of point-to-point communications. Simple bus architectures result, and the number of buses in an architecture is low, between 7 and 9 buses. For different weights, however, the bus structure involves different buses. 30 buses were used in the six cases. Only three buses are heavily re-used in 5 different architectures. Therefore, it is difficult to postulate that a unique bus will efficiently solve the communication needs for different cases. Complex buses connecting many cores are rarely used. It is mostly point-to-point links that are re-used. The total bus length is high, meaning that individual buses are long. This is reasonable as timing minimization was not a primary concern. If the bus length is important (w_l is high) then the method is able to produce architectures with a small total bus length (see rows 3, 6, 9 and 12). The average communication conflict is high, around 0.24. Thus, the low communication concurrency will result in poor timing. Redundancies are also high, meaning that the bus architectures offers some core connectivities that are not required. Thus, to obtain simple bus structures with a small total length, all weight factors w_l, w_r, and w_n must have large values. Rows 13-24 correspond to the second design scenario, in which communication overlaps must be avoided, while the other factors are less important. Also, this scenario considers that required bus speed is low, thus minimizing bus length is secondary. Note that there are no time conflicts and no data losses for the resulting BA. Bus architectures are more complex, and include more point-to-point links. Overall bus lengths are larger, which indicates that the individual buses will be slower. Figure 8 shows the synthesized bus topology for the design scenario in which bus length and complexity are very important ($w_n = w_l = w_r = 1.0$), and communication overlaps ($w_c = 0.1$) are secondary.

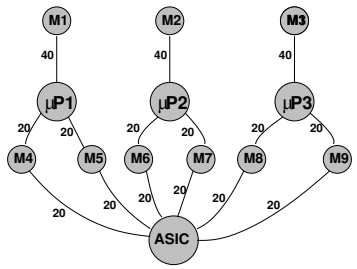


Figure 9. Core Graph for JPEG

BA synthesis was used to automatically produce optimized bus architectures for the SoC of the JPEG image compression encoder. The task graph for the JPEG encoder included three identical and parallel sequences of tasks. Each sequence processes a different color of an image (RGB), and includes five consecutive tasks: preprocessing, FDCT, quantization, zig-zag, and RLE & Huffman coding. The hardware-software co-design methodology in Figure 2 was performed on the JPEG task graph. For each sequence, tasks preprocessing, quantization, zig-zag, and RLE & Huffman coding were partitioned into software, and task FDCT into hardware. The architecture included three processor cores (a distinct core for each parallel sequence), an ASIC for the FDCT tasks, and memory modules for data communication. Each processor has its own local memory. Processors and ASIC communicate through shared memory. To decrease memory access times, the first architecture considered interleaved memory blocks M4-M9. Figure 10 shows the resulting Core Graph. The considered processing technology was 0.18μ TSMC. Microprocessor cores were of about $5 \times 5\ mm^2$, memory cores of about 25% of the area of processor cores, and ASIC were about 30% of processor core area.

Figure 10 shows bus architecture synthesis results for the CG in Figure 9. The hierarchical cluster growth algorithm generated the core placement shown at the top, and the bus architecture is presented at the bottom of Figure 10. The synthesis goal

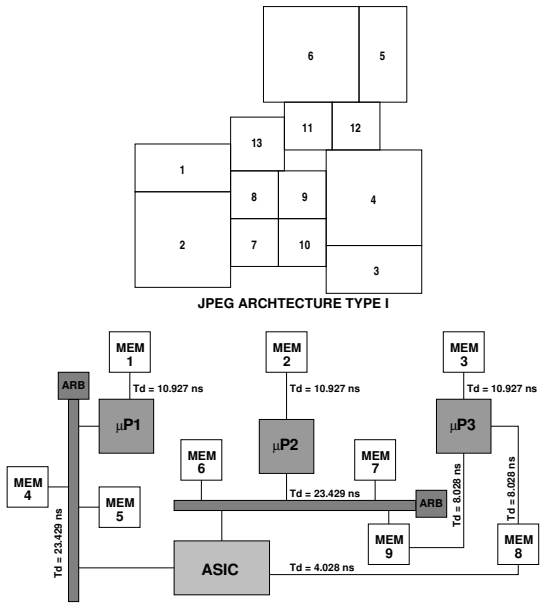


Figure 10. Bus architecture for JPEG SoC

was to generate a fast architecture. Bus architecture complexity was not a major concern, because the number of IP cores is reasonably high. Thus, the goal of BA synthesis was to minimize communication conflicts ($w_c = 1.0$), minimize the total bus length ($w_l = 1.0$), while disregarding the number of buses and redundant structures in a BA ($w_n = w_r = 0.1$). After bus architecture generation, each of the buses was routed, and the resulting delays are indicated in the figure. Note that the best BA is not perfectly regular, even though the CG is regular. Processor P1, and memory modules M4 and M5 are linked through a shared bus, similar to processor P2 and memory blocks M6 and M7. This happens because the placements of these blocks is similar. However, processor P3 and memories M8 and M9 are linked through a different structure, which improves the speed of the bus for the specific placement of these blocks. This explains that optimized BA do not depend only on architectural level elements (like the amount of exchanged data between cores), but on layout aspects, also.

6 Conclusions

This paper presents a BA synthesis algorithm for designing the communication sub-system of a SoC fabricated in a VDSM process. BA synthesis creates customized BA depending on the application specifics and performance requirements. BA synthesis includes finding the bus topology, and routing the individual buses. The BA synthesis algorithm identifies the set of possible building blocks, and assembles them into a topology using simulated annealing (SA) as an exploration engine. The algorithm also employs a bus architecture synthesis table and select-eliminate method to prune dominated solutions. This reduces the exponential number of potential bus architectures. The algorithm is capable of exploring in a very short time the large solution space for SoC with many cores. Also, it is sensitive to placement and routing information, which results in buses of higher speed.

7 Acknowledgements

This work was supported by a DAC graduate scholarship awarded to Nattawut Thepayasuwan, and IBM Faculty Partnership Award received by Dr. A. Doboli in 2001. The authors want to thank Dr. N. Dhanwada for his comments and suggestions.

References

[1] "DesignWare AMBA On-Chip Bus Solution", *www.convergencepromotions.com/ARM/catalog/156.html*.

[2] IBM Blue Logic Technology, *//http:www-3.ibm.com/chips/bluelogic/*.

[3] IBM CoreConnect Bus Architecture White Paper, *//http:www-3.ibm.com/chips/products/coreconnect/index.html*.

[4] SRC Research needs in Logic and Physical Level Design and Analysis, August 2002.

[5] T. R. Bednar, P. H. Buffet, R. J. Darden, S. W. Gould, P. S. Zuchowski, "Issues and Strategies for the Physical Design of System-on-Chip ASICs", *IBM J. of Research & Development*, Vol. 46, No. 6, Nov. 2002, pp. 661-673.

[6] R. Bergamaschi, W. Lee, "Designing Systems-on-Chip using Cores", *Proc. of the DAC*, 2000, pp. 420-425.

[7] J. Darringer, R. Bergamaschi, S. Battacharyya, D. Brand, A. Herkersdorf, J. Morell, I. Nair, P. Sagmeister, Y. Shin, "Early Analysis Tools for System-on-a-Chip Design", *IBM J. of Research & Development*, Vol. 46, No. 6, 2002, pp. 691-707.

[8] J. M. Daveau, G. F. Marchioro, T. Ben Ismail, A. A. Jerraya, "Protocol Selection and Interface Generation for HW-SW Codesign", *IEEE Trans. on VLSI Systems*, Vol. 5, No. 1, pp. 136-144, March 1997.

[9] R. P. Dick, N. K. Jha, "MOCSYN: Multiobjective core-based single-chip system synthesis," *Design Automation and Test in Europe Conf.*, pp.263-270, 1999.

[10] A. Doboli, "Integrated Hardware/Software Co-synthesis and High Level Synthesis for Design of Embedded Systems under Power and Latency Constraints", *Proc. of Design, Automation and Test in Europe Conference*, 2001.

[11] M. Drinic, D. Kirovski, S. Meguerdichian, M. Potkonjak, "Latency-Guided On-Chip Bus Network Design", *Proc. of the ICCAD*, 2000, pp. 420-423.

[12] D. Gajski, F. Vahid, "Specification and Design of Embedded Hardware/Software Systems", *IEEE Design & Test of Computers*, Spring 1995, pp. 53-67.

[13] M. Gasteier, M. Glesner, "Bus-Based Communication Synthesis on System Level", *ACM Transactions on DAES*, Vol. 4, No. 1, January 1999, pp. 1-11.

[14] J. Hu, Y. Deng, R. Marculescu, "System-level Point-to-Point Communication Synthesis Using Floorplanning Information", *Proc. of the International Conference on VLSI Design*, 2002.

[15] K. Lahiri, A. Raghunathan, S. Dey, "Efficient Exploration of the SoC Communication Architecture Design Space", *Proc. of the ICCAD*, 2000, pp. 424-430.

[16] S. Meguerdician, M. Drinic, D. Kirovski, "Latency-Driven Design of Multi-Purpose Systems-on-Chip", *Proc. of the DAC*, 2001.

[17] R. Ortega, G. Boriello, "Communication Synthesis for Distributed Embedded Systems", *Proc. of the ICCAD*, 1998, pp. 437-444.

[18] C. Reeves *et al*, "Modern Heuristic Techniques for Combinatorial Problems", *J. Wiley*, 1993.

[19] T. Seceleanu et al, "On-Chip Segmented Bus: A Self-Timed Approach", *Proc. of the IEEE ASIC/SOC Conference*, 2002, pp. 216-220.

[20] N. Sherwani, "Algorithms for VLSI Physical Design Automation", *Kluwer*, 1999.

[21] D. Sylvester, K. Keutzer, "A Global Wiring Paradigm for Deep Submicron Design", *IEEE Trans. on CADICS*, Vol. 19, No. 2, pp. 242-252, February 2000.

[22] T. Y. Yen, W. Wolf, "Communication Synthesis for Distributed Systems", *Proc. of the ICCAD*, 1995.