Slack Allocation Based Co-Synthesis and Optimization of Bus and Memory Architectures for MPSoCs

Sujan Pandey*
NXP Semiconductors Research
Eindhoven, The Netherlands
sujan.pandey@nxp.com

Rolf Drechsler
Dept. of Computer Science
University of Bremen, Germany
drechsle@informatik.uni-bremen.de

Abstract

In this paper, we present a bus and memory architectures co-synthesis technique. The co-synthesis problem is formulated as an optimization problem, where scheduling, allocation, and binding of tasks are done simultaneously in order to optimize the bus widths, the number of buses, and the memory sizes. As a main contribution, bus and memory architectures are optimized simultaneously by allocating different amount of slacks to them during co-synthesis. The method finds a balance of slack allocation for both bus and memory optimization. While the previous co-synthesis approaches do not consider the slack allocation technique, the synthesized bus and memory architectures will not be optimal in terms of area and energy consumption. The experimental results carried out on real-life applications show 19% and 24% reduction in bus and memory area, respectively and 37% reduction in energy overhead due to a bridge in compared to the previous co-synthesis approach.

1 Introduction

In recent years, the Multiprocessor System-on-Chip (MPSoC) computing platform is becoming a solution to meet the strength performance requirements for the next generation multimedia and network applications. Besides of its flexibility and scalability, designers have to cope with several design challenges that become severe as the complexity of a system increases. Among them communication and memory become a major performance bottleneck as there is a need to transfer and load/store a huge amount of data specially for multimedia applications. Thus, it is essential to design and optimize both communication and memory architectures simultaneously.

Traditionally, bus and memory architectures were synthesized separately and the resulting solution could be sub-optimal in terms of chip size and performance. For instance, the first approach for synthesizing a single global bus was proposed in [2], which finds the minimum bus width in order to minimize the chip size. The recent approaches for multi-bus based hierarchical communication synthesis can be categorized into: synthesis oriented optimization techniques [17, 21, 8, 12, 14] and exploration based approaches [11, 18, 16]. The first approach is based on scheduling of tasks to synthesize a custom communication architecture. While, the second approach uses a simulation technique and maps a requirement to a set of bus architecture templates such as AMBA and CoreConnect. The major concern of the simulation based approaches are that the resulting bus architecture may be over or underutilized and makes it inefficient in terms of performance. All the above approaches deal with communication synthesis, however, they do not address the problem of synthesizing a memory architecture.

In a real-time embedded system, a memory architecture influences the performance in terms of power, delay, and chip size. Recently, it has been predicted that almost 70% of the chip area will be occupied by memories [1] and this trend is expected to grow in future as the complexity of system increases. The recent works on memory synthesis and optimization [20, 1] use a system level approach, which maps variables of a system behavior to a set of memories using variable lifetime analysis and task reordering. Our approach complements these works by analyzing lifetime and reordering of tasks for both buses and memories optimization, simultaneously. For this purpose, we allocate different amount of slacks to shorten the lifetime and reordering of tasks in order to maximize buses and memories sharing. In [3, 5] an exploration based approach to optimize bus and memory architectures was proposed. The approach gives a local optimal solution by mapping a design to a library of memories and buses. Recently, in [4] a small subset of the co-synthesis problem has been addressed. The approach synthesized a bus architecture and a set of buffers that stores copies of the frequently used data in the main memory. In [15] a co-synthesis of communication and memory architecture is studied. However, the approach is limited to a crossbar switch based point-to-point architecture. Further, the method relies on the simulation based technique, thus many design problems such as task reordering for bus sharing and variable lifetime shortening to optimize the memory size are missing. This eventually leads to a sub-optimal solution in terms of bus cost (bus widths and the number of buses) and memory size.

As main contributions compared to the recent approaches [4, 15, 5], this paper formulates the problem of co-synthesis of bus and memory architectures as a scheduling, allocation, and binding problem. These steps are performed simultaneously in order to minimize the bus cost, memory size, and the number of communications via a bridge

*This work was done, while the first author was with the Computer Architecture Group at University of Bremen, Germany.

(*cuts*) [13] among the modules that are mapped to separate buses. These three variables of an objective function are solved together with a set of constraints using an optimization tool that finds a global optimal solution. Further in the formulation, the timing slack of each task is exploited for a variable lifetime optimization and tasks reordering to share buses and memories among the tasks. The tasks are scheduled to minimize the number of task's timing overlaps for bus sharing and to minimize the lifetime of variables that are mapped to a memory. Intuitively, the shorter the lifetime of a variable, the more the chances of variable overwriting, which, in turn, results in the minimum memory size. Further, at each step for allocation and binding of tasks to a set of buses and memories, the number of cuts among the tasks that are mapped to separate buses, is evaluated and a solution is chosen that gives the minimum number of cuts. The simultaneous scheduling, allocation, and binding of tasks based on slack allocation technique gives an optimal solution in terms of bus widths, number of buses, memory sizes, and energy consumption compared to the state-of-the-art co-synthesis techniques.

2 Preliminaries

We consider an embedded system which is realized as an MPSoC. Such a system consists of several on-chip processing modules, like general-purpose processors, application specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). These on-chip modules communicate with each other by transferring data through a shared bus. We assume that a system has been partitioned into HW/SW and mapped efficiently onto the appropriate modules of an SoC. All communications $c \in C$ that take place among the on-chip modules using an on-chip bus are captured by communication tasks c_i as shown in Fig. 1(a). Each communication task in the figure takes certain time duration to transfer data. This duration is called a *communication lifetime interval* (CLTI), which is a function of data size, bus width, and voltage. An edge between two nodes c_i and c_j weighted with w is the data processing time, which gives an early start time constraint for a successor c_j to transfer data using a bus. The data transfer delay of each communication task c is modeled as

$$CLTI_{c,r} = \left\lceil \frac{NB_c}{b_r} \right\rceil \cdot T_d, \tag{1}$$

$$T_d = K \cdot \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}}, \tag{2}$$

where NB_c (number of bit) is a size of data to be transferred by a task c with bus width b_r and T_d is a gate delay, which is a function of voltage and technology dependent parameters [19]. In Eq. (2), V_{dd} and V_{th} are the supply and the threshold voltage, respectively. Further, notations K and α are the technology dependent parameters.

3 Motivational example

In this section we give a motivation for simultaneous scheduling, allocation, and binding of tasks for both bus and memory optimization, where the slack is exploited to maximize buses and memories sharing among the tasks. As a result of this, the synthesized bus and memory architectures are optimal in terms of 1.) bus widths and the number of buses and 2.) memory sizes, respectively. We consider a partitioned and mapped HW/SW system. Based on the partitioned and mapped system, a communication task graph $G_C(C, \Pi)$ with nine tasks and their data dependencies is extracted as shown in Fig. 1(a), where a task with dotted circle is for memory write and a task with a solid circle is for memory read. In the figure, tasks $\{c_4, c_5, c_7\}$, $\{c_8, c_9\}$, $\{c_1, c_2, c_3\}$, and $\{c_6\}$ are initiated by on-chip modules M1, M2, M3, and M4, respectively. Fig. 1(b) shows a schedule of tasks with their ASAP and ALAP time. A white rectangle is a slack of a task so that a task can be scheduled in an interval of ASAP and ALAP with an aim to minimize the number of overlaps among the tasks. This results in the minimum number of buses. For a given schedule of tasks, Fig. 1(c) shows the memory lifetime interval of tasks[1] (i.e., data associated with a memory write task) c_1, c_4, and c_8. Since the possible memory write time of task c_4 overlaps with the memory lifetime of task c_1, task c_4 can not overwrite the memory space of task c_1. Thus a separate memory space needs to be allocated for a memory write task c_4. In the figure, it can be seen that the length of memory lifetime depends on the slack and when a task is scheduled. For example, if tasks c_2 and c_3 are scheduled at their ASAP time then the memory lifetime of c_1 will be shortened and task c_4 can overwrite the memory space of task c_1. Thus a single memory space can be shared by all three tasks c_1, c_4, and c_8.

After synthesizing a bus and memory architectures, it is often impossible to avoid dependencies among the tasks that are mapped to a set of memory blocks. So, there will be a frequent communication among the modules, which are connected to two different buses. Thus in an MPSoC architecture, bridges are used to connect multiple buses so that a module that is connected to a bus can communicate with a module connected to another bus. The recent effort on power estimation of communication architecture [6] shows that a bridge contributes a significant amount of power due to its communication overhead. Thus, while synthesizing the memory architecture, it is equally essential to have an efficient partitioning and mapping of data associated to tasks onto a set of memory blocks. This results in the minimum number of cuts among the modules.

Fig. 2(a) depicts a cluster of tasks with their data dependencies for a given schedule. The main aim is to cluster the data associated with tasks, which have data dependencies and map each cluster to a memory. From the given cluster of tasks as shown in Fig. 2(a), tasks are clustered further in order to find the minimum number of memories unless there is a memory access conflict (when two tasks access a memory at a same time). Fig. 2(b) and (c) show the synthesized memories and the cuts. In the first figure, data of tasks, which are initiated by modules M1 and M2 are mapped to MEM1. While data of tasks initiated by mod-

[1] Throughout this paper, we use a term memory lifetime interval (MLTI) of a task instead of data associated to a task.

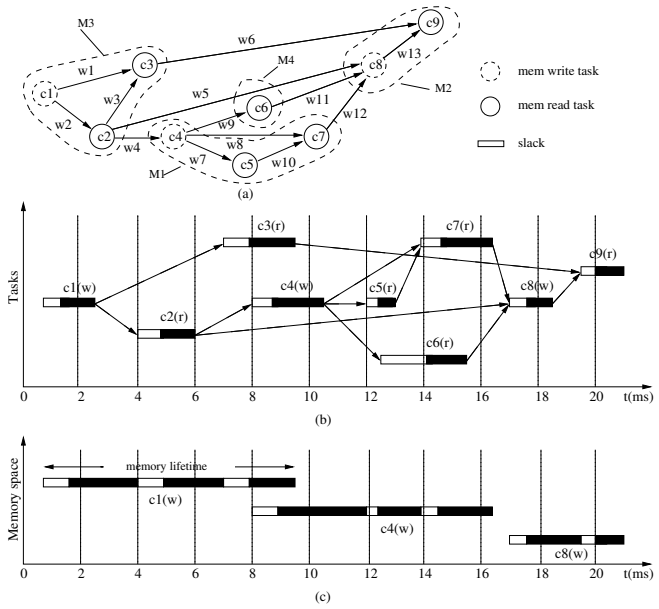

Figure 1. Communication tasks and their schedule. (a) Example communication tasks. (b) A schedule of tasks with their ASAP and ALAP time. (c) Memory lifetime interval of tasks

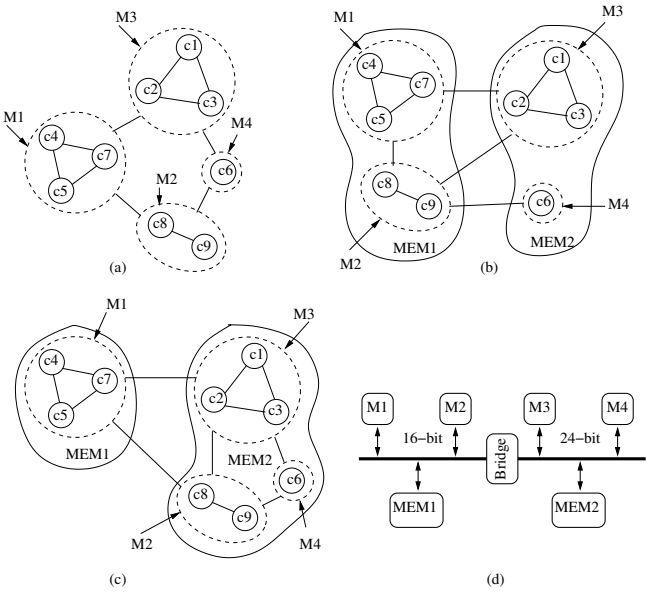

Figure 2. Co-synthesis of on-chip buses and memories. (a) Clique of data dependency tasks and their dependencies. (b) Synthesized memories with number of cuts = 3. (c) Synthesized memories with number of cuts = 2. (d) Synthesized bus architecture and memories with interconnection of on-chip modules and bridge.

ules M3 and M4 are mapped to MEM2. In the figure, there are three cuts, which mean that either modules M1 and M2 or M3 and M4 access memory MEM2 or MEM1 for three times using a bridge. Similarly, Fig. 2(c) depicts synthesized memory sizes, number of memories, and the number of cuts. In the figure, the number of cuts is less than the synthesized memory of Fig. 2(b). This, in turn, results in less power and delay overhead due to a communication via a bridge. Thus, the memory partitioning and mapping of Fig. 2(c) give the optimal number of memories with the minimum number of bridge accesses. The synthesized on-chip buses and memories with their interconnection are shown in Fig. 2(d).

4 Problem formulation

As discussed in the motivational example, in this section, we formulate the problem of simultaneous scheduling, allocation, and binding of tasks with an objective function and a set of constraints.

Definition 1 *Let R be a set of bus widths, let Mem be a set of byte memories, and let cuts be the number of dependencies among a set of tasks that are mapped to separate buses. Further, let a set* $Depn \subseteq C \times C$ *be a set of tasks that have data dependency, then the cost function can be written as*:

$$BMCost = \sum_{r \in R} \beta \cdot b_r + \sum_{s \in Mem} \gamma \cdot m_s + \sum_{\substack{\forall (c_i, c_j) \in Depn \\ \wedge (c_i \wedge c_j) \not\rightarrow b_r}} \theta \cdot cuts, \tag{3}$$

where the terms β, γ, and θ are constants. The notations b and m are indexes to a bus and a memory block, respectively. While r and s are bus width and memory size. In Eq. (3) the first summation is for a given set of bus widths, the second summation is over a given set of memories, and the third summation is for tasks that have dependencies and are not mapped to the same bus b_r (bus index b and width r).

Definition 2 *Let* $X_{c,t,r,s}$ *be a binary variable that gives a binding constraint such that* $X_{c,t,r,s}$ *= 1 when a task* c *is scheduled at a unique time* t*, bus width* r*, and an allocated memory size* s*. Otherwise, it is zero*.

$$\sum_{r \in R} \sum_{s \in Mem} \sum_{t = ASAP_c'}^{ALAP_c} X_{c,t,r,s} = 1 \quad : \forall c \in C \tag{4}$$

In Eq. (4), the first and second summations are over bus widths and memories, respectively. While the third summation is over a possible task schedule time that ranges from its $\text{ASAP}_c^{'}$ to ALAP_c such that the timing slack can be exploited for both bus and memory sharing as discussed in Section 3. The term $\text{ASAP}_c^{'}$ can be defined as

$$ASAP_c' = ASAP_c + (1 - \%BusSlack_c) \cdot (ALAP_c - ASAP_c). \tag{5}$$

In Eq (5) for all tasks c, if $x\%$ of the total slack is allocated for bus sharing then $(1-x)$ % of the slack is used for memory size optimization. A new ALAP time of a task c for memory optimization is $\text{ASAP}_c^{'}$. So, depending on the amount of slack that is allocated for bus optimization, it influences the synthesized memory size. Thus, the aim is to find a balance of slack allocation for both bus and memory optimization.

Definition 3 *Let $\Omega = \bigcup_{c \in C}\{ASAP_c^{'}, \cdots, ALAP_c\}$ be a time window such that the tasks that are scheduled within this interval could overlap. If the timing of a task overlaps with another task then the task is assigned to a separate bus with index b and width r. The bus constraint can be described as [12]:*

$$\forall t \in \Omega, \forall r \in R, \forall s \in Mem,$$
$$\sum_{c \in C} \sum_{\substack{(t' \in \{t, \cdots, t+CLTI_{c,r}-1\} \\ \cap \{ASAP_c^{'}, \cdots, ALAP_c - CLTI_{c,r}\})}} X_{c,t',r,s} \leq b_r, \quad (6)$$

where the first summation is over a set of tasks and the second summation is over a time window covering all schedule times $t^{'}$ for which tasks could overlap.

Definition 4 *Let C_w be a set of tasks that writes data to a memory. A successor task $c_j \in C_w$ can overwrite a predecessor task $c_i \in C_w$ only if its memory lifetime does not overlap with the lifetime of c_i. Otherwise, a separate memory space $NB_c \mapsto s$ is allocated to c_j. This constraint is given as:*

$$\forall t \in \Omega, \forall r \in R, \forall s \in Mem,$$
$$\sum_{c \in C} \sum_{\substack{(t'' \in \{t, \cdots, t+MLTI_{c,r}-1\} \\ \cap \{ASAP_c, \cdots, ASAP_c^{'} - MLTI_{c,r}\})}} X_{c,t'',r,s} \leq m_s, \quad (7)$$

where the second summation is over a time window that covers memory lifetime of tasks, which could overlap.

Definition 5 *If tasks $(c_i, c_j) \in Depn$ and tasks c_i and c_j are scheduled at time t_i and t_j on two different buses, respectively, then there is a cut (communication via a bridge). Thus, those tasks are scheduled with an aim to cluster a set of tasks and map them onto a module with the minimum number of cuts.*

$$\forall (c_i, c_j) \in Depn \wedge (c_i \wedge c_j) \not\mapsto b_r$$
$$\sum_{c \in C} \sum_{t=ASAP_c^{'}}^{ALAP} X_{c,t,r,s} \leq cuts \quad : \forall r \in R, s \in Mem \quad (8)$$

Definition 6 *Let $c^{'}$ and c be predecessor and successor tasks, respectively. Their dependency constraint can be expressed as [12]:*

$$\forall (c^{'}, c) \in Depn,$$
$$\sum_{r \in R} \sum_{\substack{t= \\ ASAP_c^{'}}}^{ALAP_c} \sum_{s \in Mem} t \cdot X_{c,t,r,s} \geq \sum_{r \in R} \sum_{\substack{t= \\ ASAP_{c^{'}}^{'}}}^{ALAP_{c^{'}}} \sum_{s \in Mem} \quad (9)$$
$$(t + CLTI_{c^{'},r} + w) \cdot X_{c^{'},t,r,s}$$

where the right hand side of the equation expresses that a successor task c should be executed only after the execution of the predecessor task $c^{'}$. The term w is the data processing delay.

5 Co-synthesis algorithm

The objective of the bus and memory co-synthesis problem is to minimize the cost function, which includes bus, memory, and the number of cuts. This can be expressed as:

Minimize: BMCost (see Eq. 3)
subject to, Eqs. (4), (6), (7), (8), and (9)

The data transfer delay $CLTI$ for each task c with bus width r is evaluated using Eq. (1). Algorithm 1, shows a method to compute the memory lifetime interval of a task c (memory write task), which is scheduled at time $stime$ with a bus width r. At line 1, a loop is for tasks that write data to a memory. At line 3, conditions are checked for all c_i, which write data first to a memory before the current task c. If the conditions are satisfied, then line 5 finds a successor (memory read task) of c_i to compute the last read time from the memory. This is shown at line 6, where the $MLTI$ of a task c_i is the summation of the successor's last read time ($z.stime$) and its data transfer delay $CLTI_{z,r}$ with a bus width r minus write time $c_i.stime$ of task c_i. As a task c_i can have more than one successor (memory read tasks), at line 7, its $MLTIs$ considering all successors are stored. Line 10 provides the maximum $MLTI$ considering the last memory read time. At line 11, $MaxMLTI$ is returned to generate the memory constraint as shown in Eq. (7). From the algorithm, it can be seen that the $MLTI$ of tasks depends on the time when a task is scheduled (i.e., $stime$) and a bus width r. Thus, our optimization algorithm finds an optimal schedule with a selection of bus width that minimizes the length of MLTIs.

Similarly, Algorithm 2 finds the number of cuts for a set of tasks that are scheduled and mapped onto a set of buses. At line 1 a loop starts for tasks c_i and c_j. If the conditions are fulfilled at line 3 then the number of cuts are evaluated for each pair of tasks that are mapped onto separate buses at line 5. The number of cuts is returned at line 7 to generate the cut constraint as shown in Eq. (8).

In the above bus and memory co-synthesis and optimization problem, the optimization variables b_r, m_s, and $cuts$ are integer and linear to the objective function MBCost. While the variables $CLTIs$ and $MLTIs$ are real values, thus, the optimization problem is a *mixed integer linear programming* (MILP) problem, which can be solved using an optimization tool.

6 Case studies

We validate the effectiveness of the proposed technique using real-life multimedia applications, i.e., an audio decoder and a speech recognition system. The audio decoder includes four main decoding steps, which are inverse quantization, channel decoupling, reconstruct curve, and IMDCT. Similarly, the second speech recognition system consists of three main components: front end, decoder, and linguist. The front end includes series of data processing tasks such as pre-emphasis, hamming window, FFT (fast Fourier transformation), mel frequency filter, IFFT, cepstral mean nor-

```
COMPUTEPREDSMLTI(c, stime, r)
 1  for (c_i ∈ C_w)
 2  do
 3     if (c_i.stime < c.stime)
 4       then
 5            z = c_i.SUCCESSORS(c_i) ∧ z ∉ C_w;
 6            MLTI = z.stime + CLTI_{z,r} − c_i.stime;
 7            vector<int*> MLTIList.push_back(MLTI);
 8            endif;
 9     endfor;
10  MaxMLTI = MAX(MLTIList);
11  return MaxMLTI;
```

Algorithm 1: Compute memory lifetime interval (MLTI).

```
COMPUTECUTSNUM()
1  for (c_i ∈ C) ∧ (c_j ∈ C)
2  do
3     if (c_i, c_j ∈ Depn) ∧ (c_i ↦ b'_{r1}) ∧ (c_j ↦ b''_{r2})
4       then
5            cuts = cuts + 1;
6     endfor;
7  return cuts;
```

Algorithm 2: Compute the number of cuts.

malization, and feature extraction to generate the features from the speech. After manually partitioning and mapping of the above applications, Table 1 depicts a set of on-chip modules for the audio decoder and the speech recognition system. The modules are two PowerPC processors, inverse modified discrete cosine transformation (IMDCT), compact flash interface, speech processor, fast Fourier transformation (FFT), and an audio buffer for streaming. A set of communication tasks is extracted [9] after profiling an application using the GNU *gprof*.

The on-chip buses are given as a library of buses with different bus widths, which range from 16 to 128 bit wide. The bus synthesis algorithm was implemented in C as a preprocessing model to interface with a solver of MOSEK. Table 2 depicts the results of synthesized bus width, number of buses, memory size, and the number of memories for different percentage of slack (BusSlack) that is allocated for bus optimization. In the table, the column entitled *BusSlack* gives the percentage of slack that is allocated for bus optimization. For instance, column *BusSlack* 100% means that all the slacks are allocated for bus optimization (i.e., for bus sharing), while no slack is allocated for memory optimization (i.e., increment in overlaps among MLTIs). Similarly, 40% in column *BusSlack* means that 40% slack is allocated for bus optimization and 60% slack is allocated for memory size optimization. The column entitled *Buses (b_r)* gives the synthesized bus widths and the number of buses for different percentage of slacks. In the column entitled *Mems (m_s)* shows a synthesized memory block for each bus as presented in the same row in the table. The co-synthesis results show that the bus width increases with decreasing percentage of slack that is allocated for bus optimization. However, in column *Mems (m_s)* the synthesized memory size decreases with decreasing BusSlack. As discussed in Section 3, if a small amount of slacks is allocated for bus optimization and the rest of the slacks is used for memory optimization then this results in the minimum number of overlaps among the memory lifetimes. Thus a set of tasks can overwrite the memory space that was used previously. Further, the column entitled *cuts* gives the number of cuts among the tasks that are mapped to separate buses. The columns entitled *MILP* shows the run time of the MILP based co-synthesis approach. The run time for solving the MILP based co-synthesis problem increases as the number of variables increases, which is due to the memory resource constraint set by variable BusSlack.

Application	Modules
Audio decoder	PowerPC1
	IMDCT
	CF-interface
	Audio buffer
Speech recognition	PowerPC2
	FFT
	Speech processor

Table 1. A set of modules after HW/SW partitioning and mapping of real-life applications

Fig. 3 depicts a synthesized bus and memory architectures for BusSlack = 70%. The synthesized buses are 24, 32, and 48 bit wide and corresponding synthesized memories are 1.3KB, 1.7KB, and 2.6KB respectively. Table 3 compares the results of bus area, memory area, and energy overhead with the previous approach [5] for a synthesized bus architecture as shown in Fig. 3 with BusSlack = 70%. The bus and memory areas estimation models are based on the approaches proposed in [10, 7]. The values are estimated for 70nm technology with the maximum wiring pitch 300nm (ITRS 2006). The columns entitled *Mem-area* and *Bus-area* give estimated memory and bus area for both approaches, respectively. With our co-synthesis approach, there are 19.28% and 24.55% reductions in area for bus and memory architectures, respectively. The column entitled *Energy-Ov* gives an energy overhead for a bridge with an average wait cycles 17 to get a bus granted. Further, the column entitled *cuts* compares the number of cuts among the tasks that are mapped to separate buses. As our co-synthesis approach incorporates this cost during scheduling, allocation, and binding, while the previous approach does not, the number of cuts is less compared to the previous technique [5]. The results show 37.45% reduction in energy overhead after incorporating the *cuts* during co-synthesis.

7 Conclusion

The previous bus and memory architectures co-synthesis approaches focused mainly on solving a small subset of the co-synthesis problem such as combined bus and buffer synthesis, crossbar based bus and memory synthesis without considering the variable lifetime optimization and tasks ordering techniques. Thus, the resulting solution will be suboptimal in terms of bus widths, number of buses, memory sizes, and energy consumption. In this work, the co-synthesis problem is formulated as simultaneous schedul-

BusSlack (%)	Buses (b_r)	Mems m_s (KB)	No. of cuts	MILP (sec.)
100	b_1 = 24 b_2 = 32 b_3 = 48	m_1 = 1.8 m_2 = 2.4 m_3 = 3.7	23	41.3
90	b_1 = 24 b_2 = 32 b_3 = 48	m_1 = 1.6 m_2 = 2.1 m_3 = 3.3	24	54.7
80	b_1 = 24 b_2 = 32 b_3 = 48	m_1 = 1.6 m_2 = 2.1 m_3 = 3.3	24	69.2
70	b_1 = 24 b_2 = 32 b_3 = 48	m_1 = 1.3 m_2 = 1.7 m_3 = 2.6	24	83.3
60	b_1 = 32 b_2 = 32 b_3 = 48	m_1 = 1.3 m_2 = 1.7 m_3 = 2.6	26	91.6
50	b_1 = 32 b_2 = 32 b_3 = 64	m_1 = 0.9 m_2 = 1.3 m_3 = 2.0	26	103.4
40	b_1 = 32 b_2 = 32 b_3 = 64	m_1 = 0.9 m_2 = 1.3 m_3 = 2.0	26	113.1

Table 2. Bus and memory co-synthesis for different percentages of slack

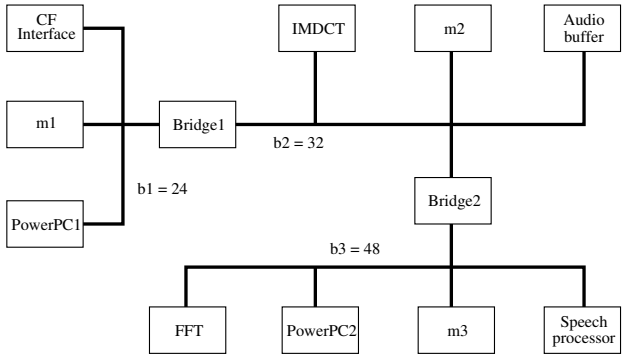

Figure 3. Synthesized bus and memory architectures with three buses and memory blocks for BusSlack = 70%.

Approach	Mem-area (mm^2)	Bus-area (mm^2)	Energy-Ov (μJ)	cuts
Our-App	4.70e-04	1.24e-07	54.6	24
Pre-App [5]	6.23e-04	1.54e-07	87.3	38

Table 3. Comparison of memory area, bus area, and energy overhead for a synthesized buses and memories with BusSlack = 70%

ing, allocation, and binding of tasks. The slack is exploited for both bus and memory architecture optimization. A case study carried out on real-life multimedia applications shows that the slack allocation technique for co-synthesis finds a solution with a balance slack allocation to both bus and memory architectures optimization. Further, while binding a set of tasks on buses, their cuts are incorporated in an objective function to reduce the energy overhead due to communication via a bridge.

References

[1] L. Cai, H. Yu, and D. Gajski. A novel memory size model for variable mapping in system level design. In *proc. of ASPDAC*, pages 813–818, 2004.

[2] M. Gasteier et al. Bus-based communication synthesis on system level. *In ACM Tran. of design automation electronic systems*, pages 1–11, 1999.

[3] P. Grun, N. Dutt, and A. Nicolau. Memory system connectvity exploration. In *proc. of DATE*, pages 894–901, 2002.

[4] I. Issenin and N. Dutt. Data reuse driven energy aware mpsoc co-synthesis of memory and communication architecture for streaming applications. In *proc. of CODES*, pages 294–299, 2006.

[5] S. Kim, C. Im, and S. Ha. Efficient exploration of on-chip bus architectures and memory allocation. In *proc. of CODES*, pages 248–253, 2004.

[6] K. Lahiri and A. Raghunathan. Power analysis of system-level on-chip communication architectures. In *proc. of CODES*, pages 236–241, 2004.

[7] D. Langen et al. High level estimation of the area and power consumption of on-chip interconnects. In *proc. of Int. Conf. on ASIC/SoC*, pages 297–301, 2000.

[8] D. Lyonnard et al. Automatic generation of application specific architectures for heterogeneous mpsoc. In *proc. of DAC*, pages 518–523, 2001.

[9] Z. Ming. Architecture exploration for speech-feature-extraction acceleration. *Bachelor thesis, Darmstadt University of Technology, Germany*, September 2005.

[10] J. M. Mulder et al. An area model for on-chip memories and its application. *IEEE Tran. on CAD of Integrated Circuits and Systems*, Vol. 26(No. 2):98–106, 1991.

[11] S. Murali and G. D. Micheli. An application specific design methodology for STbus crossbar generation. In *proc. of DATE*, pages 1176 – 1181, 2005.

[12] S. Pandey et al. Statistical on-chip communication bus synthesis and voltage scaling under timing yield constraint. In *proc. of DAC*, pages 663–668, 2006.

[13] S. Pandey et al. Co-synthesis of custom on-chip bus and memory for MPSoC architectures. In *proc. of Int. Conf. on VLSISoC*, pages 304–307, 2007.

[14] S. Pandey and M. Glesner. Simultaneous on-chip bus synthesis and voltage scaling under random on-chip data traffic. *In IEEE Trans. VLSI Systems*, 15(10):1111–1124, 2007.

[15] S. Pasricha and N. Dutt. COSMECA: Application specific co-synthesis of memory and communication architectures for mpsoc. In *proc. of DATE*, pages 700–705, 2006.

[16] S. Pasricha et al. FABSYN:Floorplan-aware bus architecture synthesis. *In IEEE Trans. VLSI Systems*, Vol. 14(No. 3):241–253, 2006.

[17] A. Pinto, L. P. Carloni, and A. V. Singiovanni. Constraint driven communication synthesis. In *proc. of DAC*, pages 783–788, June 2002.

[18] K. K. Rye et al. Automated bus generation for multiprocessor soc design. *IEEE Tran. on CAD of Integrated Circuits and Systems*, Vol. 23(No. 11):1531–1549, Nov. 2004.

[19] N. H. E. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*. Addison wesley, 1994.

[20] S. Wuytack et al. Minimizing the required memory bandwidth in vlsi system realizations. *IEEE Trans. on VLSI Systems*, 7(4):433–441, 1999.

[21] T. Y. Yen and W. Wolf. Communication synthesis for distributed embedded systems. In *proc. of ICCAD*, pages 288–294, 1995.