# System-Level Point-to-Point Communication Synthesis Using Floorplanning Information†

Jingcao Hu Yangdong Deng Radu Marculescu
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
*{jingcao,yangdon,radum}@ece.cmu.edu*

**Abstract:** *In this paper, we present a point-to-point (P2P) communication synthesis methodology for System-On-Chip (SOC) design. We consider real-time systems where IP selection, mapping and task scheduling are already fixed. Our algorithm takes the communication task graph (CTG) and IP sizes as inputs and automatically synthesizes a P2P communication network, which satisfies the specified deadlines of the application. As main contribution, we first formulate the problem of automatic bitwidth synthesis which minimizes total wirelength and then propose an efficient heuristic to solve it. A key element in our approach is a communication-driven floorplanner which considers the communication energy consumption in the objective function. Experimental results show that, compared to standard shared bus architecture, significant power savings can be achieved by using the P2P scheme and communication-driven floorplanning. For instance, for an H.263 encoder, we estimate 21.6% savings in energy and 15.1% in terms of wiring resources with an area overhead of only 4%.*

**Keywords**
System-leve design, Communication synthesis, Point-to-point communication, Floorplanning, Low-power

## 1. Introduction

With the advent of System-On-Chip (SOC) era, IP-based design is rapidly becoming the dominant design paradigm. Under this paradigm, a verified system description is first mapped to a certain set of pre-designed IPs. Then, the SOC designer's task is to construct a communication network and corresponding glue logic to interconnect these IPs.

To date, the shared-bus scheme (either single bus or multi-bus) has been the system communication architecture of choice [1]. However, there are several problems associated with the standard bus architectures. First, a global bus implies a large capacitive load for the bus drivers. In turn, this implies large delays and huge power consumption. Second, in ultra-deep sub-micron era, design of long, wide buses becomes a real challenge. While physical information is extremely important for successful bus design, the environment in which the bus is embedded is very hard to predict and characterize early in the design stages.

To address the above problems related to the bus architecture, other communication schemes are currently being explored [2][3]. Among them, the *Point-to-Point* (P2P) communication seems to be a promising solution. Under this scheme, between each pair of communicating IPs, there exists a dedicated communication link. From the perspective of physical design, these links will all be routed along the shortest Manhattan distances with a good router. Intuitively, the P2P scheme has the following advantages:

- Compared to a shared bus implementation, the capacitive loads of dedicated links tend to be smaller. Hence, the P2P scheme has potential for reduced latency.
- Concurrency is improved with the elimination of bus contention. This implies higher performance.
- During each communication transaction, only the wires of the active links are driven, while in the shared-bus case the *whole* bus tree has to be charged or discharged. Consequently, reduced energy consumption is expected.
- For physical design, each communication link can be *independently* optimized (e.g. wire sizing, buffer insertion, repeater sizing, etc.). On the other side, optimizing a shared bus with multiple loads can be very difficult.

Obviously, the P2P scheme has its own disadvantages. For instance, it needs more wires which may lead to routability problems[1]. Another shortcoming is the additional hardware for the extra ports on the IPs and associated design effort. However, the hardware overhead might be compensated by the savings from removing all bus arbiters. Meanwhile, the design overhead can be overcome by using parameterized IPs [5].

Although both shared bus and P2P schemes have their own pros and cons, to the best of our knowledge, there is no systematic comparison between these two schemes on real examples. In this paper, we provide a quantitative study and comparison to justify the choice of the P2P architecture. Our main contribution is to propose a methodology to automatically synthesize the P2P communication

†Research supported in part by NSF under grant CCR-00-93104, and in part by DARPA/MARCO Gigascale Silicon Research Center.

1. As far as we know, there is no quantitative analysis on this issue yet.

network for SOC design. A salient feature of our approach is a *bitwidth synthesis algorithm* which assigns proper bitwidth for each communication link under performance constraints, while minimizing the *total wirelength* of the communication network. A key element in our approach is an embedded energy-aware *communication-driven floorplanner* which is used to assign the IPs on the chip at the beginning stage of our synthesis flow. Experimental results show that, compared to standard shared bus architecture, significant power savings can be achieved by using the P2P scheme and communication-driven floorplanning.

The paper is organized as follows: Section 2 summarizes the related research in communication synthesis. An overview of our approach is given in Section 3. Sections 4 and 5 discuss our communication-driven floorplanning and bitwidth synthesis algorithms, respectively. Experimental results are shown in Section 6. Finally, section 7 summarizes our contribution and outlines some directions for future work.

## 2. Related work

There is already a significant body of research on the synthesis of shared-bus communication architectures. In their paper, Lahiri et al. [6] take communication traces and a given bus topology as inputs and then assign IPs to different buses according to their communication patterns to achieve the best communication performance. They assume that the buses with calculated parameters and the protocols can be successfully built at the physical design stage, which may not be always the case. Once such kind of violation is detected, one has to go back to the higher level of abstraction, adding some constraints, and re-invoke the design process.

The authors of [7] propose a latency-guided on-chip network design methodology, which generates bus topology and maps IPs to buses according to their communication pattern. An approximate floorplanner is included to ensure that any bus length will not exceed the user specified constraints. Simulated annealing is then employed to derive both bus topology and floorplan, which makes it unaffordable in terms of CPU time for large systems.

We also mention that neither of the above solutions directly takes energy consumption into consideration during the synthesis step.

## 3. Overview of our approach

### 3.1 Input information

The systems that we consider are *real-time systems*. We assume that the designer has already selected the set of IP cores, mapped the computational tasks to different IPs and scheduled their execution. The following three types of input information should be provided.

a) *Communication Task Graph* (CTG)

CTG is provided by the designer as result of a profiling step. A typical CTG example is shown in Figure 1. In this representation, every link denotes a dependency, which can be a *temporary dependency* (TD), a *functional dependency* (FD), or an *internal dependency* (ID) [8]. TD connects two tasks that do not communicate but are hosted on the same IP. FD represents data transfers between tasks. Finally, ID connects two communicating tasks on the same IP.

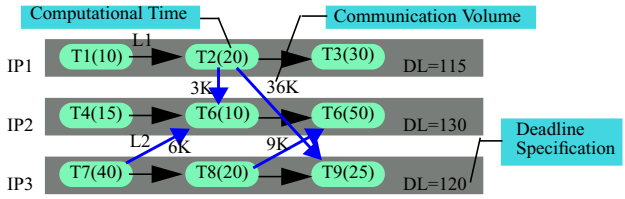



**Figure 1. A CTG example**

We assume that ID and TD are executed instantaneously without using any communication link resources. Thus, for our communication synthesis, we don't distinguish between them. For example, in Figure 1, L1 could be either an ID or a TD, while L2 is a FD. Each FD is tagged with an integer representing the *volume* of data that has to be transferred between the tasks. Each task is also given a *computational time*.

All tasks work sequentially which means that communication and computation of any task do not overlap. Communication is carried out when the sending task has finished its execution and the receiving task is ready to execute. In Figure 1, for example, if *T2* finishes execution when *T8* is still running, it will stall there until *T8* finishes execution and sends out data to *T6*. As soon as *T2* sends out all the data to *T9*, *T3* starts executing on *IP1*.

b) *Size of IPs*

For hard IP cores, the vendor will provide the size information. For soft IP cores, a fast synthesis of the RTL code or an empirical estimation based on the gate count of that IP gives the size of the IP [9][10].

c) *Deadlines*

The deadlines are specified by designers to serve as performance constraints for the entire design. For instance, in Figure 1, deadline of task *T3* is 115 units. The solution generated by our algorithm must meet all these deadlines.

### 3.2 Platform overview

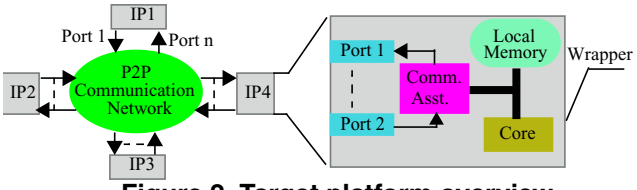



**Figure 2. Target platform overview**

Figure 2 illustrates the target platform. We assume that every IP core is encapsulated in a wrapper which implements the communication interface. Each port in the wrapper is connected to a link in the P2P network which is exclusively used to communicate with one and only one

other specific IP. Good candidates of wrappers should be designed under the guide of parameterized system design methodology [5], which offers the flexibility to easily adapt its configurations, such as number of ports, width of each port, etc. to its specific application. The functionality of communication assistant (*Comm. Asst.* box in Figure 2) is to transfer data from ports to local memory. As shown in Figure 2, incoming data is temporarily stored in the local memory before being processed by the core.

### 3.3 Assumptions

We assume that a *Globally-Asynchronous Locally-Synchronous* (GALS) [11][12] design methodology is used. This enables each P2P communication link to work at a different speed, either specified by the designer or calculated from its electrical parameters after floorplanning. An *upper bound* for the speed of the link between $IP_i$ and $IP_j$ could be calculated as:

$$F_{i,j}^{max} = \frac{k_1}{C_{i,j}} = \frac{k_1}{k_2 \times D_{i,j}} = \frac{k_3}{D_{i,j}} \quad (1)$$

where $D_{i,j}$ the Manhattan distance between $IP_i$ and $IP_j$, and $k_1$, $k_2$, $k_3$ are constants determined by electrical and physical parameters.

We use the result calculated in (1) as our default speed for links. However, for small area chips, maximum working speed derived from eq. (1) may not be feasible. In such cases, we allow the user to specify a working speed for the link, taking into account such factors as frequencies of the two IPs attached to that link, buffer speed, etc.

The wires of a communication link could roughly be divided into *data wires* and *control wires*. The number of control wires is usually much smaller than that of data wires. Additionally, since they are transferring more correlated information, control wires consume much less power. Therefore, we omit the energy consumption of these wires from our analysis.

Let $Vol_{i,j}$ be the total amount of data to be sent from $IP_i$ to $IP_j$, and $w_{i,j}$ be the bitwidth of the link from $IP_i$ to $IP_j$. Similar to [13], we also assume that communicating data is random, so we can estimate energy consumed by the link in sending the volume $Vol_{i,j}$ of data as:

$$E_{i,j} = C_{i,j} \times Vdd^2 \times S_f \times w_{i,j} \times \left(\frac{Vol_{i,j}}{w_{i,j}}\right)$$

$$= S_f \times C_{i,j} \times Vdd^2 \times Vol_{i,j} \quad (2)$$

where $S_f$ is the switching factor and $((Vol_{i,j})/w_{i,j})$ is proportional with the number of packets transferred over the link. Since $C_{i,j}$ is proportional to $D_{i,j}$, we have:

$$E_{i,j} = \alpha \times Vdd^2 \times Vol_{i,j} \times D_{i,j}, \text{ where } \alpha = k_2 \times S_f \quad (3)$$

In (2) and (3), we assume a constant switching factor ($S_f$) no matter how many bits a link has. To justify this assumption, we conducted several experiments on an H.263 encoder (see Section 6.2) using real video clips. We counted the number of switching bits over a few links and observed the changes under different bitwidth (word size) values. Figure 3 shows two typical plots for the *Average Hamming Distance* (AHD) vs. word size.

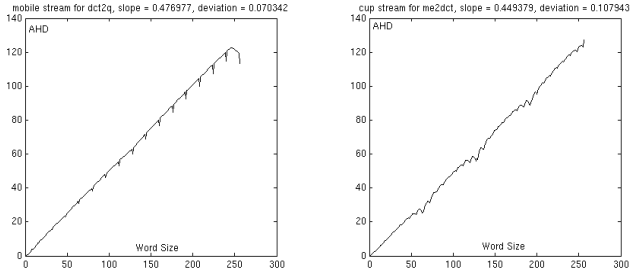



**Figure 3. A typical plots of AHD vs. word size**

As we can see, when the word size of a given link becomes larger than 10 (which is almost always true in real implementations), the AHD of consecutive words is *linearly* proportional to the word size (max deviation from the line with a slope of 0.5 is below 20% for all the traces that we simulated.) This justifies that we can use a constant switching factor $S_f = 0.5$ in our energy estimations.

### 3.4 Design flow overview

Given the CTG and size of IP cores, our floorplanner places IP cores to the appropriate locations on the chip (according to their communication needs) and then our algorithm synthesizes a P2P communication network (Figure 4). The network satisfies the performance constraints by meeting *all* the deadlines that the designer specified for the system. Since the topology is fixed for the P2P network, our toolset is used to generate the IP placement and bitwidth assignment of different P2P communication links.

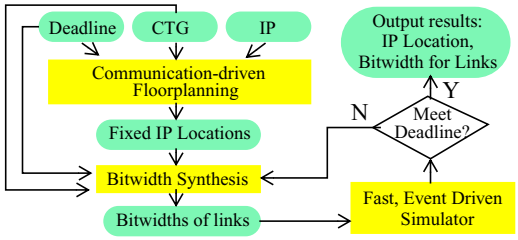



**Figure 4. The design flow**

Two cost functions are minimized during the synthesis algorithm. The first (and the most important one) is the *energy consumption* of the communication network. The second is the *total wirelength* which is a good measure of the routability of the design.

Using equation (3), the total communication energy consumption can be calculated as:

$$E = \sum_i \sum_j E_{i,j} = \alpha \times Vdd^2 \times \sum_i \sum_j Vol_{i,j} \times D_{i,j} \quad (4)$$

Equation (4) shows that the total energy consumption is proportional to $\sum\sum Vol_{i,j} \times D_{i,j}$ and is *not* related to the actual bitwidth that we use for each link.

Since the communication energy consumption is the primary goal of our optimization (and changing bitwidth of any connections will not change the total energy consumption for communication once the locations of all the IPs are fixed), we put our *Communication-Driven Floorplanning* as the first stage of our synthesizing process (see Figure 4). The second step uses a greedy communication synthesis algorithm to calculate the optimal bitwidths for different connections to optimize the routability of our system. Finally, we make sure that our solution satisfies performance requirement by simulating the solution using a fast event-driven simulator that we developed. In what follows, we describe in detail the main steps of our approach.

## 4. Communication-driven floorplanning

The floorplanning problem, also called *building block placement* problem, can be formulated as follows: Given a set of arbitrary shaped but usually rectangular modules and interconnection information among modules, find a minimum area placement with shortest wirelength. The modules may be hard or soft. The size of hard modules is fixed during floorplanning process. On the other hand, soft modules have fixed area but changeable aspect ratio. For timing driven floorplanning problem, the objective is minimum delay. There is already a large body of literature on this problem and many techniques have been proposed to date [14]. The basic idea is that highly connected modules should be placed near each other.

In our case, during the system-level design, interconnection information is not available. After hardware/software partitioning, scheduling and mapping, the design is composed of a set of IPs, volume of transactions among IPs, and the schedules of transactions.

The volume and scheduling of transactions define the communication patterns among IPs, which have important implications on module placement and link bitwidth. We try to use new closeness measures instead of physical interconnection to guide the floorplanning process. We call this approach *system-level communication-driven floorplanning*. In our implementation, we use a BSG data-structure [15] to manipulate the geometrical objects under a optimization framework of simulated annealing.

To achieve lower communication energy consumption, we suggest the following metric $M_{i,j}$ for each link:

$$M_{i,j} = Vol_{i,j} \times D_{i,j} \quad (5)$$

$M_{i,j}$ is a good objective to minimize during floorplanning because it is directly related to minimum energy consumption for the communication network. In other words, we try to optimize the wirelength weighted by the communication volume instead of pure wirelength. Meanwhile, since the square chip layout is preferred in most practical cases, we also need to consider chip area in the cost function. Therefore, assuming that the total number of IPs is $n$, the final objective function is as follows:

$$Cost = A_{chip} + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} M_{i,j} \quad (6)$$

In equation (6), the first term is the chip area and second term is the total weighted distance among all IPs. $\lambda$ is a weight introduced to control the optimization effort distribution to both terms. We choose to make these two components roughly equal.

There is an additional advantage when using cost function in (6). Guided by this cost function, IP cores that communicate frequently are very likely to be placed nearby in the final floorplan. Since $D_{i,j}$ is a good measure of communication delay between $IP_i$ and $IP_j$, minimizing the second item in equation (6) actually minimizes indirectly the total communication delay of the system. This means that the solution provided by our floorplanner also favors communication performance.

## 5. Bitwidth synthesis algorithm

### 5.1 Problem formulation

The placement of IPs is already known at this stage. Based on this information, our bitwidth synthesis algorithm assigns a bitwidth to each connection with the overall goal of keeping the total wirelength minimal to achieve better routability.

Let $w_{i,j}$ be the bitwidth of link from $IP_i$ to $IP_j$. The problem can be formulated as:

$$min \sum_i \sum_j w_{i,j} D_{i,j} \quad (7)$$

such that, for each path $i$ in the CTG,

$$\sum_k \left\lceil \frac{Vol(t_{ik}, t_{i(k+1)})}{w_{m,n} \times f_{m,n}} + comp(t_{ik}) + wait[t_{ik}] \right\rceil \le DL_i \quad (8)$$

$t_{ik} \in IP_m$ , $t_{i(k+1)} \in IP_n$

where $t_{ik}$ is the $k$-th task along the path $i$, $t_{ik} \in IP_i$ means that task $t_{ik}$ is hosted on $IP_i$, $Vol(t_{ik}, t_{i(k+1)})$ is the transaction volume between $t_{ik}$ and $t_{i(k+1)}$, and $f_{m,n}$ is the speed of the link from $IP_m$ to $IP_n$. Also $(Vol(t_{ik}, t_{i(k+1)}))/(w_{m,n} \times f_{m,n})$ is the time taken to send the data over the link, $comp(t_{ik})$ and $wait(t_{ik})$ represent the

computational time and waiting time of $t_{ik}$, respectively.

Because of the nonlinear constraints in the above formulation, we cannot use directly the classic ILP (Integer Linear Programming [16]) technique to solve this problem. Instead, we propose a greedy algorithm that is described in the next sub-section.

### 5.2 Implementation

We propose a greedy algorithm which, when integrated with the fast event-driven simulator that we developed, can be used to assign the bitwidths to all links automatically. The pseudo-code of the algorithm is given in Figure 5 and its step-by-step details are presented in what follows.

**Step1**: For each task:
  Calculate *ASAP* start time
  Calculate *ALAP* start time
  For all *i*,*j*:
  $w_{ij}$=$W_{ij}$=lower bound of bitwidth for link from $IP_i$ to $IP_j$
**Step2**: Fast simulation
  If (meet all deadlines) Goto step4
**Step3**: Select one link to increase bitwidth according to *SUP*
  Select links to decrease bitwidth, if any.
  Goto step2.
**Step4**: Refine network by shrinking link widths
  Output $W_{ij}$

**Figure 5. The bitwidth synthesis algorithm**

As **step1**, for each task in the CTG, the *As-Soon-As-Possible* (*ASAP*) and *As-Last-As-Possible* (*ALAP*) times are calculated[1]. Let $f_{i,j}$ be the link speed between $IP_i$ and $IP_j$. Then, the lower bound of their link width can be calculated as:

$$W_{i,j} = max\left\{\frac{Vol(t_m, t_n)}{ALAP(t_n) - ASAP(t_m) - comp(t_m)}\right\} / f_{i,j}$$
$$\forall t_m \in IP_i, t_n \in IP_j, t_m \to t_n \qquad (9)$$

where $t_m \to t_n$ indicates the direction of the communication, and $comp(t_m)$ is the computational time of $t_m$.

Equation (9) provides the *lower bound* of the link bitwidth. This can be explained as follows: In order to meet timing constraints, the receiving task $t_n$ must start no later than $ALAP(t_n)$. While the earliest time when the sending task $t_m$ finishes execution and is ready to send data to $t_n$ is $ASAP(t_m) + comp(t_m)$, the maximum slack for transferring the $Vol(t_m, t_n)$ of data is $ALAP(t_n) - ASAP(t_m) - comp(t_m)$.

The lower bound bitwidth of a link means that if the link is assigned a bitwidth below that bound, no solution that satisfies the timing constraints can be found, no matter what bitwidths we assign to all other links. Lower bounds are used to prune solution space to accelerate bitwidth synthesis.

Referring to **Step 2** in Figure 5, the bitwidth of each link is first set to its lower bound. Then a refining process is iteratively conducted based on simulation results for current bitwidth combination. A fast event-driven simulator, developed in-house for this project, is used for the simulation.

For each iteration, one or both of the following actions are carried out according to the simulation trace (**Step3** in Figure 5).

a) *DecreaseBitWidth*: This action is used to avoid unnecessary usage of wide links. It is carried out to any links that satisfy the following requirements:

1) Does not belong to any critical path.

2) Cannot be the link whose bitwidth has just been increased in the previous iteration. (This is necessary to avoid oscillations.)

3) Its current bitwidth is larger than its lower bound.

The bitwidth of the qualified links will be decreased by one and then the system simulated again to see whether this action has deteriorated the performance of the whole system. This is measured by the number of misses and the amount of missing on all critical paths. The action is rolled back if the performance is deteriorated.

b) *IncreaseBitWidth*: Every time one and only one link will be selected and its bitwidth will be increased by one. The candidates are composed of links belonging to at least one critical path. Among these candidates, the one with the *highest Speed-Up-Potential (SUP)* is selected.

The derivation of the highest *SUP* is based on following observation. For a link $w_{ij}$, the latency of sending the $Vol_{ij}$ of data over it is proportional to $Vol_{ij}/w_{ij}$. If the link is increased by one, then the relative speed up becomes $Vol_{ij}/w_{ij} - Vol_{ij}/(w_{ij}+1)$. Moreover, to minimize the total wirelength, we prefer to assign relative larger bitwidth to links spanning shorter distance. Hence, we use the spanning distance of a link to weight the speed up. In other words, we try to assign large bitwidths to short links. Consequently, *SUP* is the combination of these two factors:

$$sup_{ij} = \left[\frac{Vol_{i,j}}{w_{i,j}} - \frac{Vol_{i,j}}{(w_{i,j}+1)}\right] / D_{i,j} \approx \frac{Vol}{(w_{i,j})^2 \times D_{i,j}} \qquad (10)$$

The above approximation holds when $w_{i,j}$ is much larger than one.

After *IncreaseBitWidth*, the simulator runs again to check whether the system meets the deadline or not. If not, a new iteration is started. Otherwise, it jumps to Step4.

When the algorithm arrives at **Step 4**, it means that a solution which satisfies the timing constraints of the sys-

1. Infinite communication speed is assumed when calculating ASAP and ALAP times.

tem has been found. Because of the greedy nature of algorithm, there is still space left that can be explored to further lower down the routing demands. The algorithm tries to *decrease* the bitwidth of qualified links in order to reduce the total wirelength. For each round, a link with the least *SUP* is picked and its bitwidth is decreased by one. Then the simulator again checks to see whether the combination could still meet the deadline. If not, the action is rolled back and the current bitwidth combination is output as the final solution. Otherwise, it tries to find the next link and repeat above operation.

## 6. Experimental results

### 6.1Area/energy evaluation experiments

In a first set of experiments, we use TGFF [17] to generate the target task graphs. The output graph of TGFF is randomly mapped to a given number of IPs Then the computational times and communication volumes are generated according to a specified distribution.

Our tool is used to preprocess and annotate these task graphs and build the CTGs for our evaluation experiments. We generate three categories of benchmark CTGs: Category I contains randomly generated CTG with 12 IPs. Categories II and III have 30 and 50 IPs, respectively. Each category contains 10 benchmarks and every benchmark in any category has around 50 computational tasks.

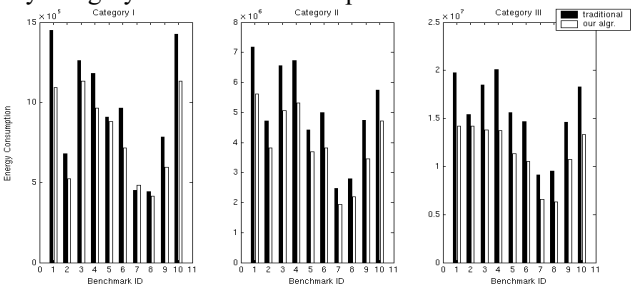



**Figure 6. Energy consumption for communication**

The experimental results show that our algorithm causes only a small area penalty; that is, for all three categories, the average overhead is within 5% of the whole chip area. Figure 6 shows the communication energy consumption comparisons for P2P (light-color bar) and traditional implementations (dark-color bar). The average energy savings for categories I, II and III are 16.89%, 21.33% and 26.34%, respectively.

As there is currently no comparable algorithm in the literature for bitwidth generation, we used the simulated annealing as reference to compare the effectiveness and efficiency of our algorithm. Results are shown in Table 1 .

**Table 1. Comparison between our algorithm and simulated annealing**

| | Category I | Category II | Category III |
|---|---|---|---|
| Speed Up | 123.7 | 28.9 | 134.9 |
| Wirelength | 1.07:1 | 1.11:1 | 0.98:1 |

Compared with the results gathered from simulated annealing, our algorithm is, on average, 123.7, 28.9 and 134.9 times faster for categories I, II and III, respectively. Also, on average, the total wirelength achieved by our algorithm is only 7% and 11% longer for category I and II, respectively. For category III, the average wirelength is even shorter than the result derived by simulated annealing, which suggests either a slower cooling procedure or more movements per temperature step are needed but, of course, this implies even a higher speed up ratio for our algorithm. We also note that for 50 IPs, most benchmarks will need more than 18 hours to synthesize when using simulated annealing[1]. This is clearly not acceptable in practice.

### 6.2 Real application

To evaluate the potential of our approach for real applications, we use an H.263 [18] encoder as a real example.

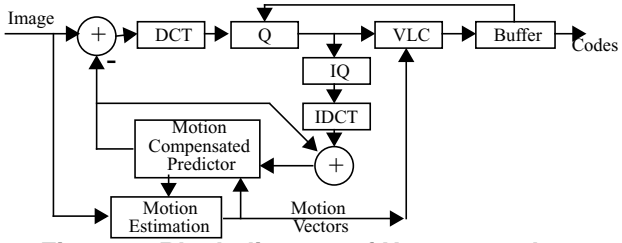



**Figure 7. Block diagram of H.263 encoder**

Figure 7 shows the block diagram of an H.263 encoder. Our target platform is composed of one AD converter, one ASIC for motion estimation (ME), three DSPs, one CPU and on-chip SRAM. Most of the above IP cores are soft IP cores and their sizes can be estimated from the IP providers such as Mentor Graphics [19]. The communication patterns among IPs are retrieved by profiling of the modified encoder code. Figure 8 shows the CTG graph for this application.

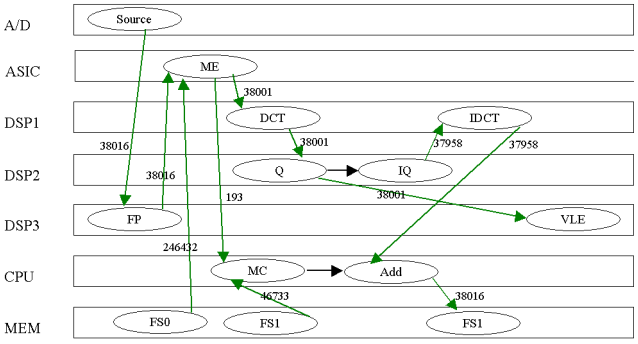
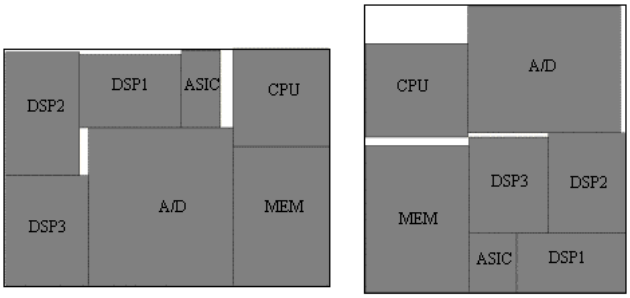



**Figure 8. CTG of H.263 encoder**

We first compare the communication synthesis results produced by our flow with and without considering the communication information at the floorplanning stage. The floorplans in this example are shown in Figure 9 and the corresponding results in Table 2. On average, we achieved

1. All experiments are conducted on a Sun Ultra SPARC 10 computer.




a. Traditional floorplan b. Communication-driven floorplan

**Figure 9. Floorplans for the H.263 encoder**

about 21.6% energy savings with an area overhead of only 4%. We should point out that an increase in area does not necessarily imply an increase in energy consumption. This is because, in a communication-driven floorplan, there may be more wasted area among IPs but this may not consume any power; also, not all the links are used for communication all the time. As for the total wirelength, our experiments show that the solution derived by communication-driven floorplan is 15.1% shorter than the one derived by considering the chip area only.

**Table 2.Energy consumption between two floorplanning solutions**

| | traditional | communication-driven | savings |
|---|---|---|---|
| Chip area($mm^2$) | 26.6805 | 27.772 | -4% |
| Wirelength(υm) | 275980 | 234200 | 15.1% |
| Energy(υJ/frame) | 3.17 | 2.49 | 21.6% |

We also compared the communication energy consumption between our P2P implementation and an ad-hoc shared-bus implementation. We simulated four video clips for both implementations and the results are shown in Table 3 (See columns labeled *Akiyo*, *Toy Box*, *Cup*, and *Color Hand*). As we can see, the energy savings are significant, that is, more than 75%, on average, in favor of P2P scheme for all four video streams.

**Table 3.Energy consumption per frame between bus-based and P2P implementation**

| | *Akiyo* | *Toy Box* | *Cup* | *Color Hand* |
|---|---|---|---|---|
| Bus (υJ) | 11.21 | 8.65 | 6.26 | 5.73 |
| P2P (υJ) | 2.49 | 1.81 | 1.32 | 1.09 |
| savings | 77.8% | 79.1% | 78.9% | 80.1% |

## 7. Conclusion and future work

In this paper, we proposed a new methodology which can be used to generate low-energy, high performance P2P communication networks for SOC design. A communication-driven floorplanner is used to generate a placement of IP cores on the chip, which minimizes communication energy consumption. Then, based on information from floorplanning, a greedy algorithm for bitwidth generation was proposed to derive the appropriate bitwidths for all the P2P links.

Our experimental results show that, compared to the shared bus architecture, P2P scheme is very efficient in reducing energy consumption. Another advantage of our methodology is its short CPU time. For typical applications, the P2P communication network can be synthesized in just seconds.

We plan to further this research in several directions. One interesting direction is to study the routability problem in the P2P communication network. Recently, many researchers have addressed the routing congestion problem at the placement stage [4]. We plan to explore the possibility of integrating and using these techniques at system-level. Another possible direction is to combine the mapping and scheduling into our framework in order to explore a larger space for the P2P communication scheme.

## References

[1] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *"Surviving the SOC Revolution,"* Kluwer Academic Publishers, 1999.

[2] L. P. Carloni, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," *IEEE/ACM ICCAD*, 1999, pp. 309-315.

[3] K. Van Rompaey, D. Verkest, I. Bolsens, H. De Man, "CoWare-a design environment for heterogeneous hardware/software systems," *IEEE/ACM DAC*, 1996, pp. 252-257.

[4] X. Yang, R. Kastner, and M. Sarrafzadeh, "Congestion estimation during top-down placement," *IEEE ISPD*, 2001.

[5] T. D. Givargis, F. Vahid, "Parameterized system design," *IEEE/ACM International Workshop on Hardware/Software Codesign (CODES)*, pp. 98-102, May 2000.

[6] K. Lahiri, A. Raghunathan, S. Dey, "Efficient exploration of the SoC communication architecture design space," *IEEE/ACM ICCAD*, 2000, pp. 424 -430.

[7] M. Drinic, D. Kirovski, S. Meguerdichian; M. Potkonjak, "Latency-guided on-chip bus network design," *IEEE/ACM ICCAD,* 2000, pp. 420-423.

[8] G. Gogniat, M. Auguin, L. Bianco and A. Pegatoquet, "A codesign back-end approach for embedded system design," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, No. 3, July 2000, pp. 492-509.

[9] D. Sylvester and K. Keutzer, "Getting to the bottom of deep submicron," *IEEE/ACM ICCAD*, pp. 203-211, 1998.

[10] http://www-device.eecs.berkeley.edu/~dennis/bacpac/index.html

[11] A. Hemani *et al*, "Lowering power consumption in clock by using globally asynchronous locally synchronous design style," *IEEE/ACM DAC.* 1999, pp. 873-878.

[12] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D thesis, Stanford University, Oct. 1984.

[13] T. Givargis, F. Vahid, "Interface exploration for reduced power in core-based systems," *Intl. Symposium on System Synthesis,* 1998, pp. 117-122.

[14] N. Sherwani, "*Algorithms for VLSI physical design automation,*" Kluwer Academic Publishers, 1999.

[15] S. Nakatake *et al*, "Module placement on BSG-structure and IC layout applications," *Proc. ACM/IEEE ICCAD*, Nov. 1996, pp.484-491.

[16] M. Garey, D. Johnson, "*Computers and intractability: a guide to the theory of NP-completeness,*" W. H. Freeman, San Francisco, 1979.

[17] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," *Proc. Of the 6th International Workshop on Hardware/software codesign*, 1998.

[18] T.Sikora, "MPEG digital video coding standards," *IEEE Signal Processing Magazine,* Sep., 1997

[19] http://www.mentor.com/inventra/cores/catalog/index.html