

Power-Aware Architecture Synthesis and Optimization for Mission-Critical Embedded Systems

Category: E3. Hardware/Software Codesign: specification languages, interfaces and integration, partitioning, synthesis

Abstract

A power-aware system architecture must provide all the necessary mechanisms to enable its application to manage power most effectively. Designers must explore system-level architectures without hardwiring high-level policies in low-level mechanisms. Unfortunately, without tool and methodology support, today's designers are unable to explore enough design points to make an effective power-aware system. This paper presents a design tool that bridges the policy-mechanism gap by automating the mapping from high-level power/performance constraints to the low-level mechanisms. Given a workload and power constraints, this tool will compute an optimal bus topology that will enable bus segmentation, voltage scaling, and many novel power-management techniques to be applied together, as driven by the application constraints. It also synthesizes a low-level power manager that implements high-level power/performance constraints in terms of architectural primitives. These include changing power modes of the components and configuring the bus topology. We demonstrate the effectiveness of this technique by mapping the Mars Pathfinder application onto the NASA X-2000 architecture. This work forms the foundation for an integrated design framework for supporting the exploration of power-aware systems.

1 Introduction

Low power design techniques previously aimed at power reduction. System-level behaviors in embedded system require power-aware features. To implement such systems, we must meet the gap between them. Power-aware synthesis would be a good methodology to enable the implementation, and fulfill the tasks in mission-critical systems.

1.1 Power aware vs. low power design

Low power designs have been studied for many years and many techniques have been developed to reduce power consumption at many levels as much as possible. Recent trends in power aware computing calls for methodology for designing systems that not only use power efficiently but also with greater flexibility, that is, to use power smartly. Power-aware system design calls for a system whose power/energy consumption adapts to constraints, internal or external changing conditions, or user requests. For many systems, end-users(not designers) want be able to make tradeoff between high performance and low power more freely or let the system itself react to the internal and/or external conditions in a power-aware way.

1.2 Architectural power-aware synthesis

Embedded system's behavior in critical environment might be constrained by various internal and external runtime conditions. Changing mission scenarios requires the system response adaptively in terms of functionality, performance and power consumptions. Previous work on low power design and power management mainly emphasis on power reduction. There is a gap between versatile system-level demands and appropriate lower level implementations.

At architectural level, system behaviors would be mapped into lower level implementations. Previously, it is very hard for the designers to implement a system that adapts to changing system behaviors and meet all system constraints at the same time. Either the designers have to hardwire some system assumptions into the implementation, or the system does not function well(or does not meet certain constraints) in some changed conditions.

Power-aware synthesis is to translate system specifications into implementation, in a way that allows the higher level behavioral variations to be mapped into different lower level configurations which correctly and effectively implement system as a whole. In other words, power-aware synthesis would be a bridge between behavioral level and lower level, not only identifying the system behavior at behavioral level, but implement it at lower levels.

The approach towards power-aware synthesis is by performing architectural level optimization and design space exploration, we translate system behaviors into architectural power management policies, and map the policies onto low level power modes configurations.

1.3 Application

NASA's X2000 project[10] for deep-space avionics system designs provide us a good platform for architectural design synthesis and optimizations. it basically provides hardware resource, like off-the-shelf component, bus architecture, manufacturing technologies. The missions are challenging and critical like sending scientific equipment to a variety of planets and send back data to the Earth. And of course there are uncertain conditions and harsh constraints out there in deep-space. Designers of the system really want design implementations that adapt to different missions and changing constraints.

With power-aware synthesis methodology, we expect system implementations that gracefully fulfill system tasks in a power-aware way.

The work presented in this paper represents one of the core tools in this larger design framework. System design work is done at behavioral level and architectural level. Our tool would perform architectural synthesis and optimization. Output from the power-aware scheduler(another tool in our framework), mainly behavioral schedule, communicating processes and constraints from system specifications, feeds into our tool. At architectural level, our tool explores the design space and find out the architecture and validate it that it meets all system constraints including runtime power constraints. The tool also generates power management schemes for the validated architecture and feed them back to behavioral level used as an estimate of software workload of communication processes.

This paper is organized as follows. Section 2 reviews backgrounds at architectural level and related work in low power and power-aware design. Section 3 describes the problem statement. We describe the architectural modeling in Section 4. Then we present the problem formulation in Section 5 and a constraint-driven design space exploration algorithm in Section 6. We discuss experimental results in Section 7 followed by our concluding remarks and future work.

2 Backgrounds and Related Work

Researchers in low power design fields has been working hard for about a decade and developed many useful techniques. There are techniques in almost all low levels that help realize power(energy) reduction. At circuit and gate levels, voltage scaling [5], clock scaling [9], gating sizing [8] are all effective techniques. At macro and component level, memory and cache have been studied [2, 3], mainly to minimize power consumption by reducing memory/cache accesses. At architectural level, bus/memory segmentation [12, 6, 4] were the efforts to achieve power reduction at higher levels.

At architectural level, system interconnect is an important issue for the reasons of both performance and power. Mission-critical embedded system composed of off-the-shelf components requires architectural support to accomplish heterogeneous communications and condition-critical tasks. System bus architecture becomes a main concern.

2.1 Bus segmentation

We intend to incorporate some of the low power design techniques into our tools. The techniques we first address here, is bus segmentation. It works at the right level for architectural design optimization. The main idea of bus segmentation is that by partitioning the entire bus into several segments or clusters, we reduce the bus length and capacitance on the bus, which helps reduce power consumption on the bus. Since the bus is shorter, the latency to transfer data over bus is shortened, which helps improve communication performance of the system. Moreover by dividing the bus into several clusters, we obtain parallel bandwidth over these clusters. In other words, we can perform data transferring on more bus segments at the same time, these would greatly improve system's communication throughput.

2.2 Gate/circuit level techniques

At gate level or lower level, clock scaling and voltage scaling are appropriate for certain kinds of components. With clock scaling, we can have high system performance by increasing the clock rate, with more power, or we can decrease the clock rate when we do not require high performance, thus achieve power reduction without lose system performance. Similarly, voltage scaling makes same tradeoff between high performance and low power consumption.

More aggressive and interesting power trade-off can be achieved by combining both bus segmentation and clock scaling techniques. With segmented bus, we can choose either to gain higher system performance from parallel bandwidth available on bus segments, or to maintain the system performance and lower clock rate or supply voltage to obtain power reduction. With this combined techniques, we have more freedom to system power management.

2.3 Bus architecture

From view point of power, bus architecture is an important issue because in most of the systems buses draw a considerable amount of total system power. And different bus architectures influence system power consumption and system performance in different ways. For example, parallel bus like PCI, typically has good throughput because the parallel bits of data can be transferred at the same time. But power cost is high because multiple wires need to power up and be switched up and down simultaneously. The capacitance on each wire and between the wires would cause significant power consumptions. On the other hand, serial bus, such as I2C, has relatively low power consumption because fewer wires need to be power up. But performance, or throughput, can be compromised because data have to be transferred bit by bit.

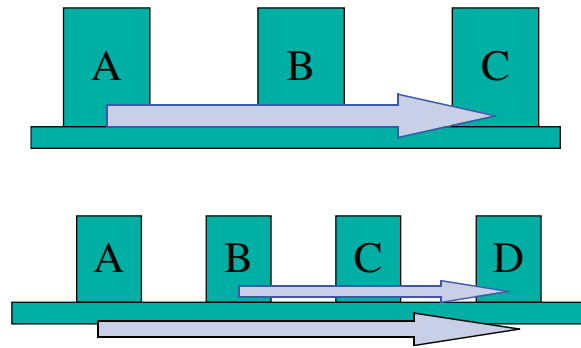


Figure 1: Inter-node constraints

Different bus topologies affect system power consumption differently. If the communications on the bus often make global travel, the bus traffic is aggravated and power consumption tends to increase. There are benefits to localize the workload over the bus. Different bus topologies have various accommodation for communication locality. Exploration of bus topology to minimize global communications would benefit systems power consumption.

2.4 FireWire bus architecture

FireWire bus architecture[1] is designed as a low power, low cost, serial bus. It gains popularity in more industries including embedded system fields. It allows 63 nodes to connect into a bus, and up to 1024 buses can be interconnected. With the increasing number of nodes connected into the architecture in a system, communications between the nodes are become eminent and complicated behaviors. In lower part of Figure 1, suppose node A sends data to node D at a transfer rate of 150mbps, at the same time node B sends data to node D at a rate of 80mbps, and suppose there is no other traffic going through node C. Thus there would be 230Mbps workload going through node C. Therefore the communication bandwidth for nodes C should be no less than 230Mbps. For a current Firewire 1394 bus, the standard transfer rates are 100, 200 and 400Mbps. The node C should then have 400Mbps transfer capability and transfer rate is set at 400Mbps when working at this scenario. Since component's power consumption is related to the working configurations, we can see that different bus topology with unbalanced distribution of communication workload on bus lead to different system performance and power consumption.

For mission critical systems, there often are restricted system resources and harsh internal and external working conditions. System design and optimization is often constraint-driven. At architectural level, we need to identify the correlations between the operations, communications and constraints to find proper power management schemes for the system.

In this paper, our emphasis is not on power management policies themselves, but rather the architectural support that enables those policies. By exploration of bus topology and adoption of low power design techniques, we expect to obtain bus topology that support both low power and power-ware system management policies.

3 Problem statement

In this section, we first define the problem, then specify inputs and outputs of the problem.

3.1 Define the problem

We study the problem of optimizing bus topology to meet all system power constraints as well as communications bandwidth constraints

to achieve better system-level performance and power consumption and enable power aware features.

Our problem is basically constraint-driven. Given system constraints of power consumption and communication bandwidth, and system behavioral schedule from higher level, we find out a feasible bus topology that accomplish the system-level tasks and meets all the system constraints. We use FireWire bus an illustrating example and map X2000 application on the FireWire bus architecture. Our approach is to design a tool that allows the designers to explore the bus topology space and find power management schemes that support low power optimization and power-aware design.

3.2 Problem inputs

The inputs to our problem are the schedules from behavioral level, a graph that represents the communication workload over the bus (we name it workload graph), and system constraints. Also we'll use the information in component library as input, too, when we need to find power management schemes and evaluate system power consumption. The outputs we expect are validated topology (if any) that meets all system constraints and power management schemes that go with the topology in each scenario of during the task lifetime.

Behavioral scheduling and architectural optimization and validation are two parts in a close-loop system designs. One can start from a available architecture and find a valid schedule or the other way round. The ultimate goal of our tools would be enable this loop-level design space exploration and optimization. To make things start, we break the loop by giving the predefined schedules to the architectural exploration problem. Unlike some other schedules which describe system or subsystem's periodical behavior like 'heartbeats', our behavioral schedule captures the system's long-term behavior at the task or mission level.

Workload graph identifies the communications of the tasks on the bus. Basically we characterize the communication path and required communication bandwidth into the graph. Behavioral schedule specifies the tasks that need communication bandwidth and there are communications constraints from higher level. Further we will show how to combine the schedule and communication information together into schedule-communication table (SCT), and use the table to derive the constraint set to evaluate the different architectures. Workload graph gives task-level knowledge of communication and workflow, and some topological information as well, but it does not give us any timing information.

System constraints come from both internally and externally. Internal constraints include those imposed by bus protocols and components. FireWire bus requires to be a tree and there should be no circle in the topology. Constraints from components involve component availability, maximum communication rate, driving ability (maximum fanout). For example, current FireWire 1394 bus transceiver can work at any of the 100, 200 and 400Mbps bit-rate. And for isochronous transfers, at most 80% of the total bandwidth is allowed to use. External constraints can be power and performance constraints from the higher level. One of the important features of our tools to support power aware design is that it accommodates many types of power constraints. Users are able to set either a maximum constant power number for the system, or a power function of time, which means power constraints can vary over time for the same architecture, or power range with a maximum power number and a minimum power number, for the system. With this flexibility, users gain much freedom in behavioral-level optimization and design space exploration.

Another indispensable input of the tools, if it is not tools itself, is the component library. Component library captures the component properties that would be used in the tools.

We capture power modes into our library. Power modes are different states a component is working at, with different power con-

sumption levels. For example, a FireWire bus transceiver can work at full-on mode, or physical-layer-only mode or suspend mode. At different modes, component perform different functionality, and consume different level of power. There are some rules applied to power modes. First, power modes must be software or hardware controllable. If a power state can only be accessed indeterminately, it can't be a power mode. Of course occasionally access to certain power modes are refrained by the system constraints or runtime conditions, but they should still be accessible given the right conditions. Second, power modes are switchable. If a component works at a state but is not able to switched into other states in terms of power levels (unless the system power is shut down), then the working state can't be called a power mode. And component at a power mode should be able to switched into another power mode if possible and necessary. One important assumption we made here is that there's an ordering between power modes. We say power mode A is lower than power mode B, meaning that in mode A, component consume less power in mode B. And we assume power modes for the same component has the ordering relations pair-wise.

Other properties we need to include into the component library are communication bandwidth each component can be working at, number of communication ports, data processing capability for processors, whether a component can act as a bus controller, maximum fanout of a component's communication port. Generally speaking, component library provides useful information for the tools to evaluate and validate the architecture.

3.3 Problem outputs

The output from our tool is a validated architecture (bus topology) that meets all the system constraints. We obtain this by enumerating trees from input information and check for every schedule intervals whether the topology is a legal one. Finally we select the one that satisfies all the conditions as our solution.

Another output from our tools is the power management schemes. We are able to find out a topology that satisfies both of two scenarios of high performance and low power with two different power management schemes. It is useful to track down the power management schemes over schedule intervals and output to higher level in order to: first, show how each component is managed, thus gives an adequate estimate of software workload of configuration program; second, help to identify which component or which interval is 'hot' and allows the users to make further optimization of the scheduling.

4 Architectural Modeling

Here we mainly model the bus architecture at system level that is composed of off-the-shelf components. We model bus architecture from the view point of power. We create our bus model and node model based on the concept of 'power mode'. We then identify the relationship between bus model and system power constraints and communications constraints. In next section, we'll use the bus model to formulate our design space exploration problem for FireWire bus architectures.

4.1 Bus model

Our bus model is a layered stack, in a sense of both services and power modes (in Figure 2). Nodes on the bus are functional components connected with bus by bus drivers. Physical layer (PHY) is responsible for data transfer, arbitration and bus configuration. Data packets can transparently pass through the node if the node neither send nor receive the packets. PHY is the interface between the cable and link layer (LNK), and PHY-LNK interface can be disabled if there are only passing packets going through the node. LNK provides safe data link and interface between PHY and transaction

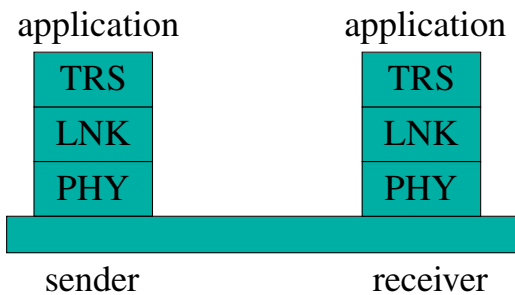


Figure 2: Bus model

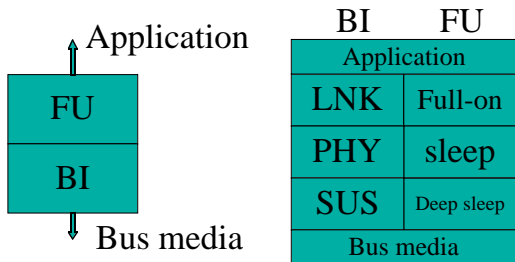


Figure 3: Node model

layer(TRS) or higher layers. LNK is closely related with transactions and applications. Usually if the LNK layer is power up, the application would be running and communicating with other nodes through all layers of bus interfaces. When a sender transfer data to a receiver, transfer rate and transfer mode of the two sides must be the same. By transfer mode, we mean whether the transfer is synchronous, asynchronous or isochronous and whether the receiver sends back acknowledge packets, depending on the transfer modes it is.

4.2 Node Model

A node is a component that has bus interface with bus. Node model consists of functional unit(FU) and bus interface(BI)(in Figure 3). Functionally, the component is at higher levels above the bus interface levels. component goes up to application levels and bus interface goes down to bus media level. From the view point of power, however, there is a correlation of power modes between components and bus interface. If the the component works at full-on mode and needs to communicate with other nodes, the bus interface has no choice but working at full-on mode, because it needs to transfer or receive data. On the other hand, if the bus interface is suspended, the component can be set into either low power mode when it does not work or operation modes if it works but does not communicate with other nodes. In systems where computing and communication are highly interleaved, it is difficult for us to completely separate the computing time period and communication time period. It is important to find right granularity to identify the separate time period of computing and communication.

We will use our bus model and node model to help identify the legal combinations of power modes and communication bandwidth levels in the system. When components are working together, there are constraints between the components. A component may not be able to go to any power modes because of the nearby component have influence on the component. For example, in upper part of Figure 1, when both node A and node C are working, and there's communication between them, node B can't be in suspend mode, because node B at least need to passing packets from node A to

node C. Thus for the neighborhood and communication reasons, there's power mode constraints imposed upon node B even if, by itself, node B can working at any power mode. The property of inter-node constraints have influence on the system power consumption. We can envision that component in different topologies may have different inter-node constraints, which lead to different system performance and power consumptions.

5 Problem Formulation

In this section we will formulate the input of our tools. Then we give a small application example of X2000 project that is mapped to FireWire bus architecture.

We consider the system architecture as components connected into a certain bus topology. The components are working at certain power modes. There are communications between the nodes via the bus, and the behaviors of the communication is given by behavioral schedule. Our tools enumerate a variety of different topologies and find out a feasible one that meet all system constraints. The inputs to our tools are the schedules from behavioral level, workload graph that represents the communications over the bus, and system constraints. Component library can also be considered an input. The outputs We expect are validated topology(if any) and power management schemes that go with the topology. We describe them in detail in the following.

5.1 Workload graph

Workload graph characterizes the communications in the system. In a workload graph $G(V, E)$, the vertices V represent nodes on the bus, and the edges $E \subseteq V \times V$ represent communication workload between nodes. Weights on the edges represents required maximum bandwidth for the communications. Edges in the graph represent only direct communications. indirect communications do not included in the graph. Labels on the edges are marks that identify the edges themselves. In next section, we use the label to create connection between the workload graph and behavioral schedule. Figure 8 shows a workload graph. We represent the workload (either data flow or control flow) in the workload graph as tuples. The first item in the tuple is the edge in the graph, representing point-to-point communication path. The second item is a label, specifying a workload in the graph. The label will be matched up with the same label in the schedule graph. The third item is communication bandwidth, which is the required bandwidth to accomplish the specified communication.

$$\begin{aligned}
 wl_graph &= (tuple1, tuple2, \dots) \\
 tuple1 &= ((cpu1, cam), 1, 20) \\
 tuple2 &= ((mc1, cpu2), 3, 50), \dots
 \end{aligned}$$

Workload graph gives task-level knowledge of communications, and some topological information as well, but it does not give us any timing information.

5.2 Behavioral schedules

A typical behavioral schedule is shown in Figure 9. The X axis represents the time. The Y-axis is components. The bars in the graph represents the active periods of each component. The numbers inside the bars are the labels indicating different workflow in communication graph.

For the schedule graph, we represent it in the following way:

$$\begin{aligned}
 sched_graph &= (tuple1, tuple2, \dots) \\
 tuple1 &= cpu1 : ((0, 20), 1), ((30, 50)), 3) \\
 tuple2 &= cpu2 : ((20, 40), 1), ((30, 40), 3)
 \end{aligned}$$

We list tuple(s) for each component, the first item of the tuple is the time interval when the component is active, the second item is the label that is correspondent to the one in the workload graph. The schedule graph gives timing and scheduling information of the component activity and communications. By the label connection, we know exactly, for a workflow, where(in workload graph) it happens, and when(in schedule graph).

5.3 Constraint set

The fact that there exist illegal combinations of power modes necessitate construction of constraint set. A constraint set is basically a group of constraints that specify which power modes are illegal. Our approach is to identify those illegal combinations and rule them out by imposing constraint set upon each component. Under those constraints, we have the freedom, for each component, to set those power modes not included in the constraint set.

Note that given a topology, we build constraint set for each component, and for each time interval, because during each time interval, the active components are different, and there may be different constraints between them.

```
cst_set = (tuple1, tuple2, ...)
tuple = [componentname] : [ON|WL] : [value]
```

for example,

```
cam : ON : LNK
cam : WL : 120
```

The component name is 'cam', 'ON' indicates power mode, 'LNK' means link layer. The tuple as a whole regulates that the component 'cam' must be working at a power modes higher than or equal to link-layer-on (thus bus transceiver must be working at full-on mode) The second tuple represents workload, meaning that the minimum bandwidth for the component 'cam' is 120Mbps.

5.4 Microrover example

Microrover is a scientific robot whose job is to collect data and image and do scientific experiment on Mars, after Mars Pathfinder, a space vehicle, send it to the surface of the planet. Technically, Microrover is an embedded system working in environment with constraints. There is only limited solar power that can be used by Microrover, and it can encounter indeterminate situations when it goes on the surface of Mars.

Here we model its digital system as in Figure 8. A camera(CAM) captures pictures, send image data to CPU. After the data are compressed in CPU, it is send to radio frequency modem(RF) and send out. A scientific equipment(SC) performs certain experiments and send data to microcontroller(MC). MC processes the data and stores in a hard drive(HD). At last the data stored in HD will be delivered to RF and sent out. Numbers on the edges indicate the workload(in term of bandwidth) of the communications between two nodes. Circled numbers are labels that distinguish communications between nodes with each other. Figure 9 is the schedule for this scenario.

Figure 10 show a topology that generated when the power constraint is 15W. It looks not like the original workload graph, but it is a solution that meets system constraints. When, for instance, CAM sends data to CPU, in generated topology, it must go through RF and HD. Then these nodes can not be put into suspended mode(SUS).

We write SUS modes into HD and RF's constraint set, indicating that these two nodes can not be suspended. Under the constraints like this for all component, we go over all the legal constraints and see whether the generated topology satisfies the system's power and communication bandwidth constraints.

```
Read in workload graph, behavioral schedule
Creating Communication-Scheduling Table
Enumerating topology, building topology library TL
For Ti in TL :
  For interval in schedule :
    Building Constraint Set
    Traverse power management schemes PMSi;
    Run power_calculator to find power number P for PMSi
    If p satisfy power_constraint :
      print find a feasible solution, Ti, PMSi
    Stop
Print cant find a feasible solution
```

Figure 4: Top level algorithm

6 Algorithm

Our algorithm is completely constraint-driven, meaning that we don't seek best or optimal solutions, but only find one solution that meets all system constraints. In that section, we first define feasibility function, which qualifies whether a generated topology is a feasible solution to our problem. Then we introduce the top level algorithm, followed by the outer loop and inner loop algorithms. We also incorporate low power design techniques into our power management schemes.

6.1 Feasibility function

We use feasibility function to validate a solution. We define feasibility function as follows:

$$P(t) = \sum_i (P_{b_i}(t) + P_{c_i}(t))$$

Where P_{b_i} is the power consumption of bus driver, and P_{c_i} is the power consumption of component. i traverses all the nodes on the bus. If $P(t) < P_{max}(t)$, then the topology is a feasible solution to the problem. Here we chiefly consider the power consumption on the bus driver and component. Power function is a function of time because power constraints may be changed over time.

With feasibility function, we are able to check whether a topology in certain power modes meets the power constraints.

6.2 Top level algorithm

In order to explore the design space of two dimensions of both topologies and power management schemes, we let the algorithm run with two nested loops. The outer loop enumerates different bus topologies; while the inner loop tries different power management schemes on each topology, walks all the step in schedule, and check whether it meets system constraints. The outer loop generates a variety of topologies by extracting a valid bus topology from a redundant graph, which is obtained from a transformation of input workload graph. The inner loop tries different power management schemes, from full-on mode for all components, to switching a number of components into lower power modes. If system constraints still can't be met after we try all the combinations, we try bus segmentation and clock scaling techniques. Here in Figure 4 is the top level pseudo code of our algorithm(pseudo code are written in PYTHON format, indentation indicates hierarchy).

6.3 Enumerate bus topologies

Our exploration space consists of different bus topologies combined with different power management schemes. if there are n nodes in the system and each node has m power modes, the complexity of exhaustively enumerating all the design points would be $\frac{n^2!m^n}{n!(n^2-n)}$,

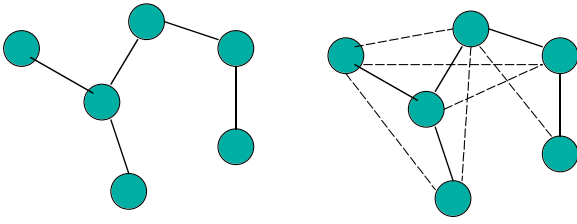


Figure 5: Construction of redundant topology

when $m = 3$ and $n = 5$, it would be greater than 10^8 . To enable design space exploration, we should be able to generate reasonable number of topology candidates. Our approach is to enumerate all the valid topologies from a redundant topology derived from input workload graph.

The way we diversify the original workload graph is to: first we add some redundant edges to the original topology, forming a redundant graph. From it we extract required topology(for FireWire 1394 bus, it is a tree without any circles). We repeat this procedure to enumerate all the possible topology.

The edges we add are those unconnected directly but connected by two edges in between. In other words, we add edges with one other nodes in between. (Figure 5) We try all power management scheme over all the topologies we enumerated. If a feasible solution is not found, we then try adding edges of up to two edges in between. If failed again, we added edges of up to three edges in between, and so on. For N edges in between. Theoretically, the upper bound of N is $\lfloor \frac{N}{2} \rfloor - 1$, the larger the number of N , the closer we build a graph to complete graph. Heuristically, we can set N to a number between 1 and $\lfloor \frac{N}{2} \rfloor - 1$. Practically, $N = 1$ would be all right for most of cases.

The complexity of enumerating topologies largely reflects complexity of our algorithm as a whole. It depends on the topology of the original graph. In an extreme case, if the original graph is the linear topology, the complexity is proportional to the number of nodes in the redundant graph. In the other extreme, if the workload graph is star-like, unfortunately, we get a complete graph. This case in reality is rare because few bus architecture totally depends on one node to enable communications between all nodes in the system. Complexity of other topologies falls in between these two extreme scenarios.

Specifically for Firewire bus architecture, it requires a tree topology. In order to build a tree of N nodes, we abstract $n - 1$ edge from a redundant graph and run the following two steps to make sure it is a tree.

- 1) first check whether it covers N nodes
- 2) whether the topology is connected

We use depth-first-search algorithm [7] to check connectivity of the topology.

6.4 Build constraint set

We build constraint set not only of power modes, but also of communication bandwidth, because during each time intervals, communications between components are different and there are different bandwidth demands for completion of all the traffic passing through the component.

Figure 6 is the algorithm to build constraint set for a component during a time interval:

```
def buildConstraintSet(interval, topo, wLGraph):
    constraintSet =
    for each wl in wLGraph:
        if wl is in interval:
            constraintSet[ON.wl's end nodes] = 'LNK'
            constraintSet[WL.wl's end nodes] += trafficOnNode
    find passing nodes into passNodeSet
    for each node in passNodeSet:
        if constraintSet[ON.node] != 'LNK':
            constraintSet[ON.node] = 'PHY'
            constraintSet[WL.node] += trafficOnNode
    return constraintSet
```

Figure 6: Build constraint set

6.5 Traverse power management schemes

Once we generate a topology, we try different kinds of power management schemes on it, for each schedule interval. Then we use our component library, constraint set for each component, and power calculator to check whether it meets all system constraints. if it does, then it is a solution.

It is straightforward to imagine trying all possible combinations of power modes for every components. But complexity of do that would be m^N where m is maximum number of power modes and N is the number of components in the topology. What we do is first clustering the components into several groups, and traversing only different combination of groups instead of individual components. We cluster the component into three groups: the components that communicate with each other(full-on), the ones that pass workload of other components(passing), and the ones neither transferring data nor passing data through itself(idle). Then we traverse power modes of only combinations of groups, thus we reduce the complexity to 3^N . With constraint set, we can rules out certain combinations and further reduce the complexity to a remarkable low level.

We also incorporate low power design techniques into our traversal schemes. When we reach the boundary of constraint set, we can't go further because the rest of power mode combinations are illegal. Then we try bus segmentation and clock scaling techniques. We actually do not use the technique themselves, but the power modes(and combinations) that are generated from them. By applying bus segmentation, multiple communication paths are allowed at the same time, or more component in unused segments can be put to suspend mode. This leads to either higher communication bandwidth or power reduction. By applying clock scaling, component can run at lower clock rate and consume less power, thus possibly be set to lower power modes. Then we have more freedom to find legal mode combinations to explore.

We paid attention to the ordering of the traversal. During a time interval, we set all components' working modes exactly same as the ones in the previous interval, and check whether it meet system constraints. If not, we cluster the components and form power modes groups, and traverse the power modes combinations as usual. For the first time interval, we set all components in full-on modes. The benefit of doing this is that potentially we may reduce power mode switching between schedule steps, which means lower switching overhead for both hardware operation and software controls, and it may help alleviate power surge as well.

Figure 7 is the top level algorithm for power management.

7 Experimental Results

We implemented our algorithm in PYTHON language, about 500 lines of code in length. The platform is Apple Mac dual-G4 450Mhz, 128MB RAM. We ran several experiments, and results

```

def find_PMScheme(topo, CST):
  for i in interval_set:
    (full-on, passing, idle) = categorize(constraint_set, nodes)
    bw = interval_bandwidth(CST, i)
    if bw > maxBW:
      try bscs # bus segmentation and clock scaling
    for n in nodes:
      try modes in (on, passing, idle)
      if powerCalculator(modes) == 'fail':
        try bscs

```

Figure 7: Topology enumeration

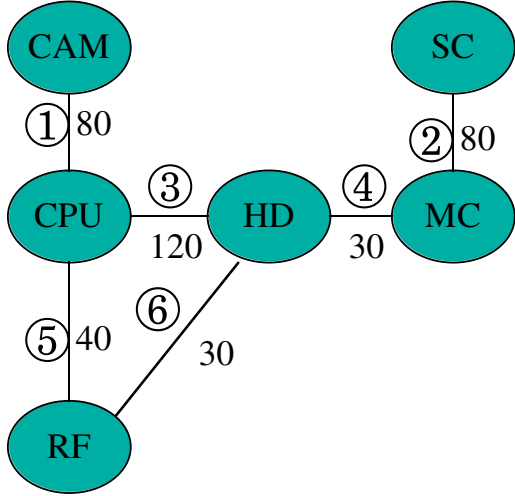


Figure 8: Workload graph for Example 1

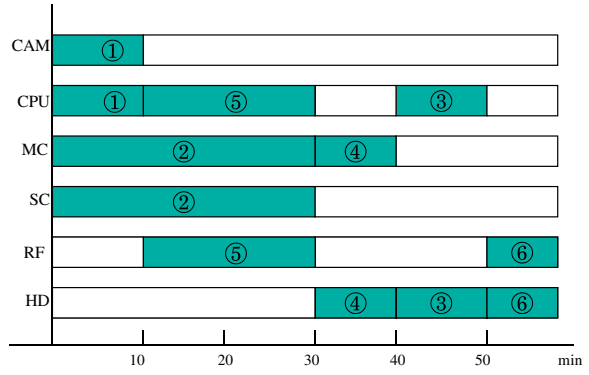
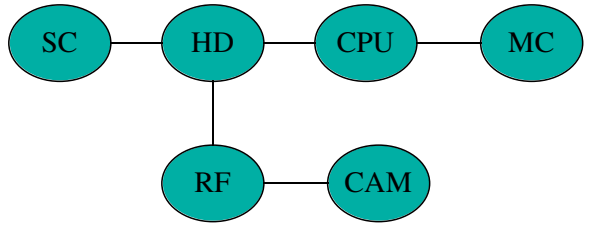


Figure 9: Schedule for the Example 1

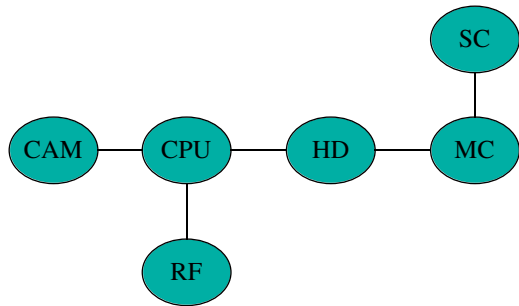


MAX_POWER constraint = 15.0W
Actual MAX_POWER = 14.9W

Figure 10: Solution with power constraint = 15W

show our algorithms are effective and efficient to find low power solutions and support power aware features as well. We developed the testbench of our own, mainly from applications of Microover. We tried different scenarios and different power constraints. Our algorithms successfully generate topologies that meet all the constraints.

1) Our first experiment is a simplified application from Microover robot. The working scenario is like this: in robot system, the camera(CAM) captures the pictures and send to a processor (CPU). After the image data are compressed, they are sent to radio frequency modem(RF). Another scientific equipment(SC) performs certain operation and send scientific data the microcontroller(MC), after the data are being processed, they are sent to non-volatile memory or a hard drive(HD) and then sent to the RF. The mission as a whole takes about one hour to finish. The workload graph and schedule graph are shown in Figure 8 and Figure 9, respectively. In workload graph, there are six nodes and six edges. We need to generate tree topology with six nodes and five edges. as we set power constraint to 15W, we get bus topology in Figure 10. all the power constraints are communication constraints are met. one interesting thing is that the edges with higher workload aggregate towards the center part of the topology. As we change the power constraints to 14W and maintain the schedule, our algorithm generated the topology in Figure 11. Note that in the latter case, bus segmentation and clock scaling techniques are used. During interval A, CAM and CPU form a cluster while SC and MC form the other one. Without bus segmentation, it requires 160Mbps bandwidth for the communication, thus every component involved in interval A would transfer data at 200Mbps. With bus segmentation and clock



MAX_POWER constraint = 14.0W
Actual MAX_POWER = 13.94W

Figure 11: Solution with power constraint = 14W

Interval	0-10	10-30	30-40	40-50	50-60
Without BS/CS	14.9W	14.0W	9.7W	12.7W	7.9W
With BS/CS	13.94W	13.04W	9.04W	11.98W	7.18W

Figure 12: Power numbers for each time interval

Interval Component	0-10	10-30	30-40	40-50	50-60
CAM	LNK200	PHY	PHY	PHY	PHY
CPU	LNK200	LNK200	PHY	LNK200	LNK100
MC	LNK200	LNK200	LNK100	PHY	PHY
SC	LNK200	LNK200	PHY	PHY	PHY
RF	PHY	LNK200	PHY	PHY	LNK100
HD	PHY	PHY	LNK100	LNK200	PHY

Figure 13: PMS table for higher max power

Interval Component	0-10	10-30	30-40	40-50	50-60
CAM	LNK100	SUS	SUS	SUS	SUS
CPU	LNK100	LNK100	SUS	LNK200	LNK100
MC	LNK100	LNK100	LNK100	SUS	SUS
SC	LNK100	LNK100	SUS	SUS	SUS
RF	SUS	LNK100	SUS	SUS	LNK100
HD	SUS	SUS	LNK100	LNK200	SUS

Figure 14: PMS table for lower max power

scaling, same amount of work is done, but transfer rates reduce to 100Mbps since each transaction requires only 80Mbps bandwidth, and power consumption is reduced from 14.90 to 13.94, due to the reduced data transfer rate on the bus and reduced clock rate for the bus transceivers. Figure 12 is the power number for each time interval for different power constraints. Figure 13 and Figure 14 are the PMS tables with higher maximum power constraints and lower maximum power constraints, respectively. The execution time for this example is less than one second.

2) Next experiment has a more complicated scenario. MC1 and MC2 are microcontrollers which control the steering motors and driving motors of Microover, they also sense and control a lot of sensors and actuators. CPU2 controls and coordinates the behaviors of the two microcontrollers and process the data from them. SC1 and SC2 are scientific equipment which collect data and store data in HD. CAM, CPU1 and RF do the similar job as in the above examples. The behavioral schedule shows the working scenario: CPU2, MC1 and MC2 control the Microover move a certain distance, and it takes some pictures and send to RF, then it goes to another place and do some scientific experiments and send data to CPU1, after data have been processed by CPU1, they would be stored in HD. In workload graph(Figure 15), there are nine nodes and eight edges. We can see at CPU1 there are a large fanout, which may not supported by bus interface(say, the bus interface support

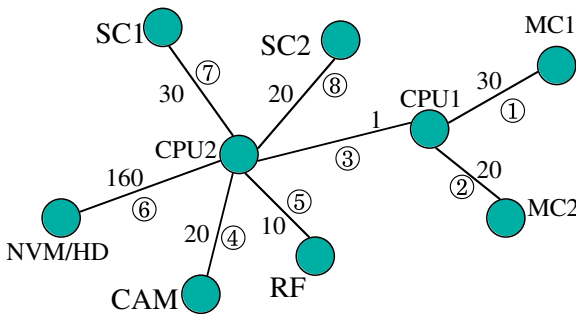


Figure 15: Workload graph for Example 2

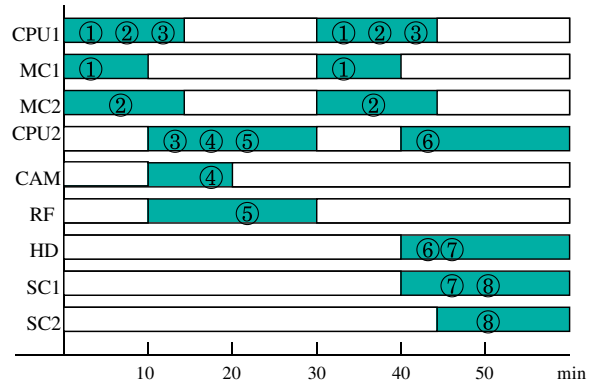


Figure 16: Behavioral schedule for Example 2

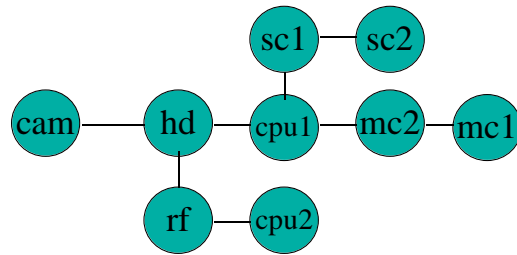


Figure 17: Solution with the power constraint = 14W

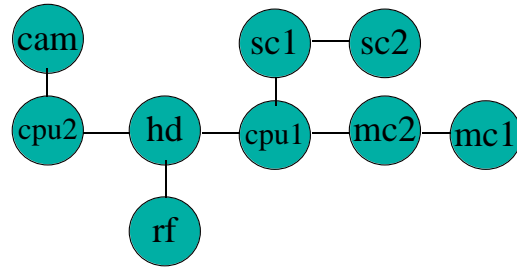


Figure 18: Solution that meets power constraints in two scenarios

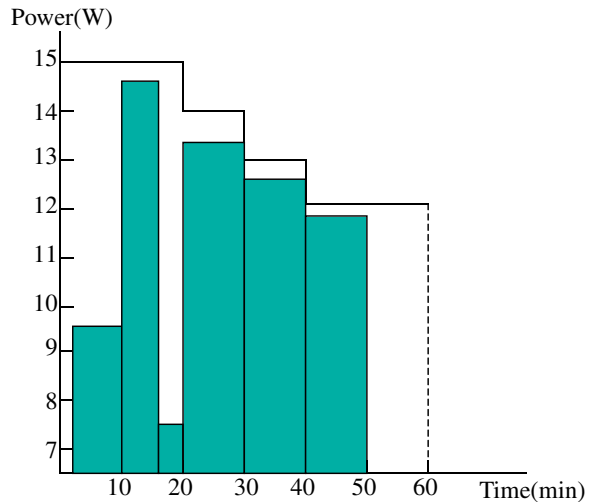


Figure 19: Power constraints: a function of time

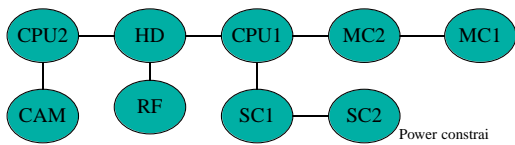


Figure 20: Solution that meets changing power constraints

maximum fanout of 3).

2.1) We give a power constraint of a maximum constant: 14W and behavioral schedule (Figure 16). And we get the following in Figure 17.

2.2) We give two different behavioral schedules and two different power constraints. The first schedule is Figure 16. The tasks are fairly parallelized, and a power constraint of 14W is given. The tasks in second schedule is full serialized, and we expect less power consumption, thus a lower power constraint is of 11W is set. Then we run our algorithm and find topology Figure 18 satisfying both schedules.

2.3) We set our power constraints as a function of time, see Figure 19, with behavioral schedule in Figure 16, we get the following topology. This constraint makes sense in our application because if the Micro rover start working at Mars noontime, the available solar power would not be a constant, but a function of time like in . We find a solution in Figure 20. Our experiment shows that we are able to explore the topology space and find the one that meets changing power constraints over time.

During all our experiments, we set a constant in topology enumerator maximum fanout = 3, if we set this constant to other numbers, we are able to give trees without different maximum fanout. Power management scheme tables for Experiment 2 are not shown due to the limitation of paper length. The execution time for this example is 20-22 seconds. Our experiments show our bus topology exploration methodology is effective. A variety of power constraints can be met.

8 Summary and Future Work

In this paper we address the issue of power aware computing at system level. Our approach to solve the problem of constraint-driven bus topology optimization and power management is to enumerate highly possible (but not all) topology candidates and find out ones that meet all system constraints. Our experiments show our algorithm support power aware features that we can give different kinds of power constraints such as maximum constraint of constant, maximum constraint of a function of time, a constraint of power range, and different schedules for one topology. We also incorporate low power design techniques into our algorithm and it obviously give more freedoms to explore the topology spaces. We generate power management schemes, which works as a roadmap how the components are configured cooperatively to achieve power aware features. Power management schemes would also be used as feed back to behavioral level and as an estimate of system software workload. This would help better interaction between behavioral level and architectural level and enable larger loop of optimization of better system performance and power consumption.

As writing the paper, our work is going on, mainly to improve the characterization of components and power management flexibility. Different components have different complexity of behaviors. A processor's behavior is much more complex than a bus transceiver. How to properly characterize each kinds of component is crucial to the quality of our tools. Some power management schemes and low power design techniques incurs either system performance degradation or hardware/software implementation overhead. In some timing-critical systems, switching between power modes lead to

large latency. To identify those tradeoff and make wise power management at right time, to right components, we need further study.

Our long term goals are to extend current work to: 1) enable behavioral-architectural loop iteration and exploration of design space that meet both behavioral and architectural level constraints. 2) put the tools into a dynamic environment. Bus topology may be dynamically reconfigured at runtime, due to the system behavioral needs or internal or external runtime changing conditions. And accordingly, power management schemes may also be changed at runtime, causing the behavioral level to find new schedules based on new constraints.

References

- [1] D. Anderson. In *FireWire System Architecture, 2nd Edition IEEE 1394a*, Addison-Wesley, 1998.
- [2] L. Benini, A. Macii, E. Macii, and M. Poncino. Selective instruction compression for memory energy reduction in embedded system. In *Proc. 1999 International Symposium on Low Power Electronics and Design*, pages 206–211, August 1999.
- [3] Y. Chang, B. Park, and C. Kyung. Conforming inverted data store for low power memory. In *Proc. 1999 International Symposium on Low Power Electronics and Design*, pages 91–93, August 1999.
- [4] J. Chen, W. Jone, J. Wang, H. Lu, and T. Chen. Segmented bus design for low-power systems. *IEEE Trans. on Very Large Scale Integration Systems*, 7(1):25–29, March 1999.
- [5] K. U. et al. Design methodology of ultra low-power mpeg4 codec core exploiting voltage scaling techniques. In *Proc. 1998 Design and Automation Conference*, pages 483–488, June 1998.
- [6] A. Farrahi, G. Tellez, and M. Sarrafzadeh. Memory segmentation to exploit sleep mode operation. In *Proc. 1995 Design Automation Conference*, pages 36–41, June 1995.
- [7] R. Gould. In *Graph Theory, The Benjamin/Cummings Publishing Company*, 1988.
- [8] M. Hashimoto, H. Mnodera, and K. Tamaru. A practical gate resizing technique considering glitch reduction for low power design. In *Proc. 1999 Design Automation Conference*, pages 446–451, June 1999.
- [9] C. Papachristou, M. Nourani, and M. Spining. A multiple clocking scheme for low-power rtl design. *IEEE Trans. on Very Large Scale Integration Systems*, 7(2):266–276, June 1999.
- [10] H. Stone. Experiences with operations and autonomy of the mars pathfinder micro rover. In *1998 IEEE Aerospace Conference Proceedings*, pages 337–351, March 1998.
- [11] R. Tai, L. Alkalai, and S. Chau. On-board preventive maintenance for lon-life deep-space missions: A model-based analysis. In *Proc. 1998 Design, International Computer Performance and Dependability Symposium*, pages 1–10, September 1998.
- [12] Y. Zhang, W. Ye, and M. Irwin. An alternative architecture for on-chip global interconnect: segmented bus power modeling. In *Thirty-Second Asilomar Conference on Signals, Systems and Computers*, pages 1062–1065, November 1998.

Power-Aware Architecture Synthesis and Optimization for Mission-Critical Embedded Systems

Dexin Li, Pai H. Chou, Nader Bagherzadeh, and Fadi Kurdahi
Dept. of Electrical & Computer Engineering
University of California
Irvine, CA 92697-2625 USA
{dli,chou}@ece.uci.edu

Dept. of Electrical & Computer Engineering
University of California at Irvine

Category: E3. Hardware/Software Codesign: specification languages, interfaces and integration, partitioning, synthesis

Format: Conference paper

Contact: Dexin Li
305 Engineering Tower
Irvine, CA 92697-2625
phone: (949) 824-1421
fax: (949) 824-3203
email: dliece.uci.edu

Estimated # of pages: 9

Keywords: power-aware design, bus topology optimization, embedded systems architecture, system-level design

Power-Aware Architecture Synthesis and Optimization for Mission-Critical Embedded Systems

A power-aware system architecture must provide all the necessary mechanisms to enable its application to manage power most effectively. Designers must explore system-level architectures without hardwiring high-level policies in low-level mechanisms. Unfortunately, without tool and methodology support, today's designers are unable to explore enough design points to make an effective power-aware system. This paper presents a design tool that bridges the policy-mechanism gap by automating the mapping from high-level power/performance constraints to the low-level mechanisms. Given a workload and power constraints, this tool will compute an optimal bus topology that will enable bus segmentation, voltage scaling, and many novel power-management techniques to be applied together, as driven by the application constraints. It also synthesizes a low-level power manager that implements high-level power/performance constraints in terms of architectural primitives. These include changing power modes of the components and configuring the bus topology. We demonstrate the effectiveness of this technique by mapping the Mars Pathfinder application onto the NASA X-2000 architecture. This work forms the foundation for an integrated design framework for supporting the exploration of power-aware systems.