Interactive System Design Flow

Technical Report CECS-02-15
April 15, 2002

Junyu Peng, Lukai Cai, Andreas Gerstlauer, Daniel D. Gajski

Center for Embedded Computer Systems
University of California
Irvine, CA 92697, USA

{pengj, lcai, gerstl, gajski}@cecs.uci.edu

Contents

List of Figures

Interactive System Design Flow

Junyu Peng, Lukai Cai, Andreas Gerstlauer, Daniel Gajski
Center for Embedded Computer Systems
University of California
Irvine, CA 92697, USA
{pengj, lcai, gerstl, gajski} @cecs.uci.edu

Abstract

This report presents an interactive system design flow. The design tasks and scenarios are defined. Data displays and the design rules to use displays are discussed. The interactive design flow enables fast and extensive design exploration with minimal effort from designer's side.

1 Introduction

In order to handle the ever increasing complexity and time-to-market pressures in the design of system-on-chip (SOCs) or embedded systems, design abstraction has been raised to system level to increase productivity [1]. At the system level, designers deal with system components which include microprocessors, special-purpose hardware units, memories and busses.

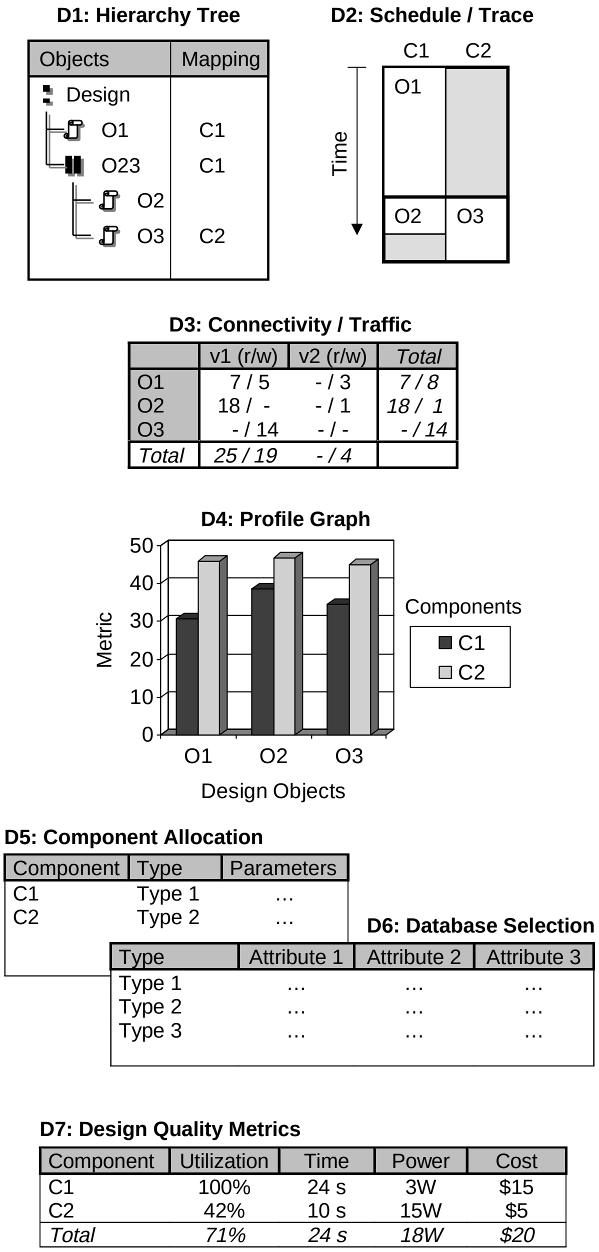
In general, the process of system level design can be divided into two major steps: **architecture exploration** and **communication exploration**.([2]) During architecture exploration, designers map the computation (behaviors) in the specification onto components of a system architecture pulled out of a component library. The design tasks in architecture exploration include allocation of system components, mapping of behaviors onto components, ordering of behaviors on each component and mapping of variables into memories. During communication exploration, designers implement the abstract communication (channels) over the actual wires of system busses based on bus protocols selected out of a protocol library. The design tasks in communication synthesis include allocation of busses, mapping of channels onto busses and insertion of transducers between busses when communication goes across different busses.

In order to be able to evaluate the quality of the design at any stage of the process, executable models describing the design at different stages should be generated. These models can be analyzed, for example through simulation, estimation, or profiling, to obtain important metrics to check the quality of design decisions made for each task. The starting model is a **specification model**, which describes the desired system functionality without any implementation detail. The model coming out at the end of system level design is an **architecture model** which describes the system architecture of the design. The system architecture consists of a network of system components connected by system busses. Between them, there are a spectrum of intermediate models generated after one or more tasks are performed.

In order to make appropriate design decisions at each task, designers need to look at all kinds of information, including characteristics profiled on the specification model, performance metrics estimated on intermediate models (and architecture models) and design database. Without an effective visualization of the data, it would be extremely difficult for designers to comprehend the needed information manually. In general, the information has to be organized and visualized graphically to help designers make quick and good decisions. For instance, a bar chart can be used to display the numbers of operations in all behaviors, which helps designers feel the computation complexity of the behaviors. Different types of information may need to be presented with different kinds of looks. For instance, the hierarchy of the specification can be better displayed in the form of tree graph than other alternatives. In the report, the graphical form of visualizing a specific kind of information is called a **display**.

1.1 Displays

We define seven basic types of displays needed for system design (1). Different sets of displays will be used at the different stages of the design process. At each step, specific instances of the general displays shown here will be used by applying them to different objects or different metrics. In the following we will briefly introduce each display and provide an overview of their capabilities. Specific usage of the displays will be shown later during the discussion of the design flow.



D1: Hierarchy Tree

Objects	Mapping
Design	
O1	C1
O23	C1
O2	
O3	C2

D2: Schedule / Trace

D3: Connectivity / Traffic

	v1 (r/w)	v2 (r/w)	*Total*
O1	7 / 5	- / 3	*7 / 8*
O2	18 / -	- / 1	*18 / 1*
O3	- / 14	- / -	*- / 14*
Total	*25 / 19*	*- / 4*	

D4: Profile Graph

D5: Component Allocation

Component	Type	Parameters
C1	Type 1	…
C2	Type 2	…

D6: Database Selection

Type	Attribute 1	Attribute 2	Attribute 3
Type 1	…	…	…
Type 2	…	…	…
Type 3	…	…	…

D7: Design Quality Metrics

Component	Utilization	Time	Power	Cost
C1	100%	24 s	3W	$15
C2	42%	10 s	15W	$5
Total	*71%*	*24 s*	*18W*	*$20*

Figure 1. System Design Displays.

The **Hierarchy Tree (D1)** displays a tree representation of the hierarchy of design objects. As such, it gives and overview of the design's composition and allows for easy navigation and selection. In addition, the Hierarchy Tree provides columns for mapping design objects to physical components.

The **Schedule or Trace (D2)** display shows the composition of the design objects over time. It provides immediate insight into the execution semantics of the design and gives feedback about timing or utilization, for example.

The **Connectivity or Traffic (D3)** display is related to the communication between different objects in the design. It is arranged as a matrix of computational objects (behaviors) over communication objects (variables or channels). The matrix can be filled with either simple connectivity information or with traffic results from design profiling or estimation.

The **Profile Graphs (D4)** chart analysis data in the form of bar graphs. Given a set of design objects, the graphs show the profile of the objects' metrics on the set of possible (allocated) components. Hence, the graphs facilitate comparison between objects and their implementation on different components.

The **Component Allocation (D5)** and **Database Selection (D6)** dialogs are forms through which the user enters decisions about the set of allocated components. The Database Selection browser is the interface to the component databases. It lets the user browse and filter the list of components and their attributes. Out of this list, the user selects components to allocate. The Component Allocation dialog then shows all currently allocated components. The user can add or delete components, and the display allows for modification of the allocated component's names and other parameters.

Finally, the **Quality Metrics (D7)** display provides feedback about the effects of the user's decisions on the design quality. After design decisions have been made, the Quality Metrics display shows the top-level results from analysis of the refined design, both in total and split into the contributions of each component.

1.2 Design Flow

Based on the displays, the corresponding design flow becomes interactive. At each task, designers are given relevant displays. Based on the displays, design decisions are made and new models reflecting the decisions are generated. After analysis of the newly generated models, resulting metrics are shown to designers to evaluate the design quality. If they are satisfactory, designers move on to the next task; otherwise, design decisions are adjusted to start another iteration.

To describe this interactive design flow, we will divide it into four scenarios (Figure 2). **Specification tuning** changes

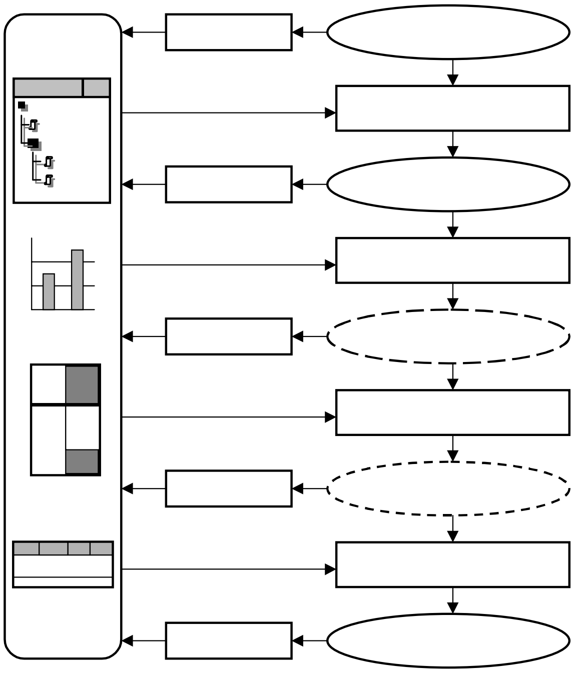


Figure 2. Interactive Design Flow.

the specification model to expose maximal behavior level parallelism and to reduce the specification complexity. **Behavior mapping** selects system components and maps behaviors onto components. **Variable mapping** select memory components and maps variables into memories. **Channel mapping** determines system busses and maps channels onto busses.

This report describes displays needed for each scenario. Some design rules on how to use the displays are presented and illustrated with examples.

2 Specification Tuning

A specification model can be developed based on the original specification written in English or in high level programming languages. The specification model is a hierarchical network describing the desired system functionality. Because the specification model is purely functional, it can be written in many different ways, all functionally equivalent. However, the quality of the specification model in terms of parallelism and complexity has immediate influence on design decision making, which is essential to the quality of the final design.

In order to improve the quality of the specification model, two kinds of optimizations can be performed. One is called **parallelization optimization**, which aims at exposing maximal parallelism at behavior level. The other is called **hierarchy optimization**, which reduces specification complexity in terms of number of behaviors and the depth of the behavior hierarchy.

2.1 Data Displays

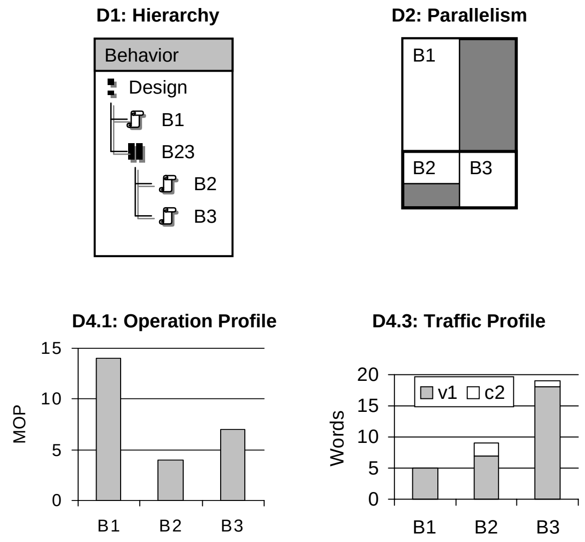


Figure 3. Specification Tuning Displays.

The data displays needed for specification tuning are shown in Figure 3. **Hierarchy (D1)** is a tree representation of the specification models. The nodes are behaviors and the edges represent parent-child relationships between behaviors. The types of nodes can be parallel, sequential, finite-state-machine, pipeline, or leaf. The example here shows the top level behavior Design is a sequential decomposition of leaf behavior B1 and B23, which is a parallel decomposition of leaf behaviors B2 and B3. The number of nodes is 5 and the depth is 2. **Traffic Profile (D4.3)** displays, for each behavior, the amount of communication traffic (product of size and access frequencies in and out of a behavior) required for variable and channel access. The example figure shows that the traffic generated by accessing variable v1 are 5, 7, 17 Words for behaviors B1, B2 and B3, respectively. The traffic generated by accessing channel c2 are 2 and 1 Words for behavior B2 and B3, respectively. **Operation Profile (D4.1)** displays the total number of operations of each behavior. **Parallelism (D2)** displays the available behavior level parallelism in a SpecC model. Each behavior is represented by a rectangle. The length of each rectangle is proportional to the number of operations of the behavior it represents. The number of columns shows the maximal amount of available parallelism. The example shows behavior B2 and B3 can be executed in parallel. Hierarchy display visualizes behavior hierarchy better than parallelism display does. However, Parallelism display shows the size for each behavior in terms of number of operations, which is not seen in the Hierarchy display.

2.2 Parallelization Optimization

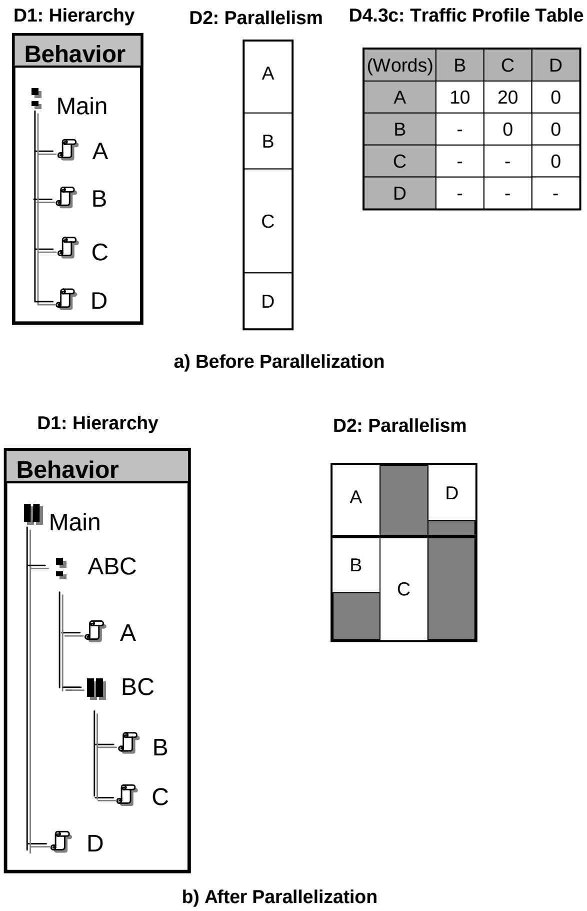
It is straightforward to transform a C description into a system specification model. For example, the functions in the C description can be easily encapsulated into SpecC behaviors by choosing the appropriate behavior granularity (size of leaf behaviors). Because C language lacks the constructs for specifying concurrent execution, the C description is purely sequential at both statement level and function level. Most system level languages provide concurrent constructs to specify behavior level parallelism, which can be exploited to improve performance in the design process. In general, it is very difficult for designers to identify all available parallelism in the C description since a thorough data dependency analysis would be needed. In a interactive design flow, with the help of displays of profiled data, the task of finding all potential parallelism becomes much easier.

2.2.1 Rules

Rule 1. Parallelize two sequential behaviors if there is no dependency between them. Since Traffic Profile displays data dependency information between behaviors, we can find out whether two sequential behaviors have dependency. If they are not dependent, they must be able to be executed in parallel with each other.

2.2.2 Example

Figure 4 gives an example of parallelization optimization. Part a) of Figure 4 shows the Hierarchy, Parallelism and Traffic Profile of the specification model before parallelization. Hierarchy and parallelism graph indicate that behaviors A, B, C and D are specified to be executed sequentially. However, the Traffic Profile shows that there are no dependency between behavior B and behavior C. Therefore, behavior B can be executed in parallel with behavior C. Similarly, since there is no dependency between behaviors D and all other behaviors A, B and C, they can also be parallelized. Part b) of Figure 4 shows the Hierarchy and Parallelism after parallelization optimization. As we can see, the pure sequential specification was transformed into a three-way parallel-execution through this optimization.



(Words)	B	C	D
A	10	20	0
B	-	0	0
C	-	-	0
D	-	-	-

Figure 4. Parallelization Optimization Example.

2.3 Hierarchy Optimization

In the specification model of a normal size design, there could be very large number of behaviors and the behavior hierarchy could be very deep. It is obvious that the design effort is proportional to the complexity at the specification level. In order to reduce the complexity, hierarchy optimization can be performed. Hierarchy optimization reduces the depth of the behavior hierarchy as well as the number of behaviors in the specification.

2.3.1 Rules

Rule 2. Combine parent and child behaviors with same execution types. As we mentioned earlier, each node (behavior) in the hierarchy tree has an associated execution type. For each edge of the hierarchy tree, if the parent node and the child node have the same type of execution, then the child node is removed and all its children become new children of the parent node. A top-down or bottom-up traversal of the tree can be performed to check for each edge.

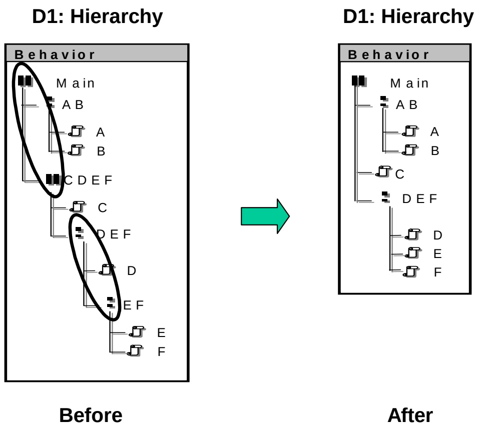


Figure 5. Hierarchy Optimization Example.

2.3.2 Examples

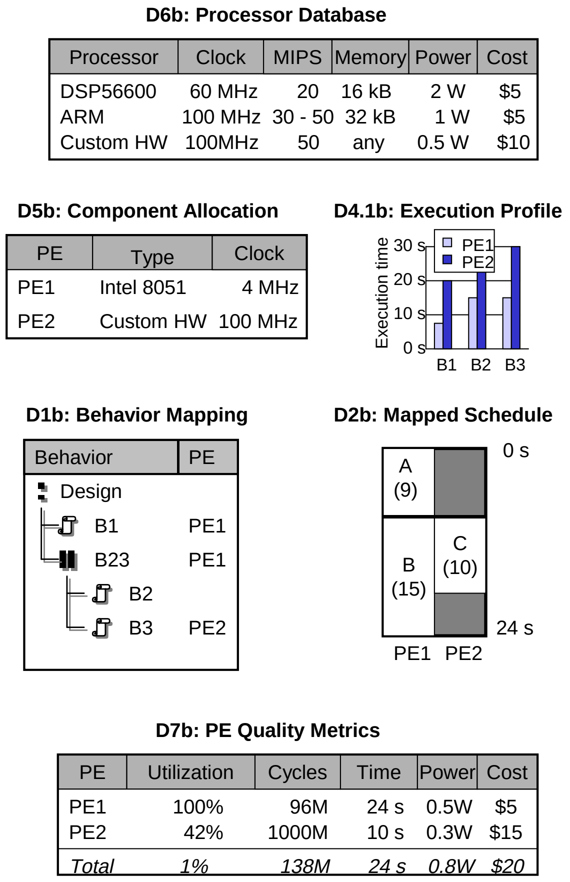
In the example shown in Figure 5, both parent behavior *DEF* and child behavior *EF* are of SEQ type. Therefore, *EF* is removed and its children *E*, *F* are promoted to be the children of *DEF*. Similarly, because both behavior *Main* and *CDEF* are of PAR type, *CDEF* is removed and *C*, *DEF* become the children of *Main*. Compared to 11 behaviors and a hierarchy of depth 5 in the old specification, the new specification has only 9 behaviors and a hierarchy of depth 3.

3 Behavior Mapping

After the aforementioned optimizations on the specification model, system components will be selected and the behaviors will be mapped onto the selected components. Parallel behaviors on each component will also be serialized because of single thread execution inside each component. We will describe component selection and mapping together because they are closely coupled.

3.1 Data Displays

The data displays needed for behavior mapping are shown in figure 6. **Processor Database (D6b)** lists all available processors with clock, memory, power and cost attributes. **Component Allocation (D5b)** is used by designers to input component allocation decisions. **Execution Profile (D4.1b)** gives the running time of each behavior on different PEs. **Behavior Mapping (D1b)**, which resembles the Hierarchy display, is used by designers to input behavior mapping decisions, i.e., which behavior is mapped to which PE. **Mapped Schedule (D2b)** displays a default execution order on each PE after behaviors are mapped to PEs. **PE Quality Metrics (D7b)** gives feedback on the quality of behavior mapping by displaying utilization, execution time information.



D6b: Processor Database

Processor	Clock	MIPS	Memory	Power	Cost
DSP56600	60 MHz	20	16 kB	2 W	$5
ARM	100 MHz	30 - 50	32 kB	1 W	$5
Custom HW	100MHz	50	any	0.5 W	$10

D5b: Component Allocation

PE	Type	Clock
PE1	Intel 8051	4 MHz
PE2	Custom HW	100 MHz

D4.1b: Execution Profile

D1b: Behavior Mapping

Behavior	PE
Design	
B1	PE1
B23	PE1
B2	
B3	PE2

D2b: Mapped Schedule

D7b: PE Quality Metrics

PE	Utilization	Cycles	Time	Power	Cost
PE1	100%	96M	24 s	0.5W	$5
PE2	42%	1000M	10 s	0.3W	$15
Total	*1%*	*138M*	*24 s*	*0.8W*	*$20*

Figure 6. Behavior Mapping Displays.

3.2 Component Selection and Mapping

The goal of component selection and behavior mapping is to ensure the design meet the given constraints in terms of time, cost, or power. The basic idea is to group behaviors into a number of groups and select appropriate component for each behavior group. There are a number of ways of grouping behaviors by considering parallelism, hierarchy and computation complexity. In this report, we will focus on the exploitation of the behavior level parallelism.

3.2.1 Rules

Rule 3. If no PE can execute the critical path in the original parallelism graph within time constraint, no solution is possible. At first, the feasibility to implement the design is checked. Assuming there are unlimited resource (components) available, the time to execute the critical path (height of parallelism graph) is the lower bound of the execution time of the design. If running the critical path on the fastest available component can not meet the time constraint, there is no solution to the design. A new specification needs to be developed or faster components must be introduced to the component library.

Rule 4. Determine the number of PEs by examining the critical path and the total amount of computation. The number of PEs can be determined by designers based on their experience. For example, if a specification has a critical path that is close to the total amount of operations, two PEs including a software processor and a custom hardware component may be needed. The number of PEs can also be estimated by simply dividing the total number of operations with the critical path length. For example, if the critical path length is 50 MOPS and the total number of operations is 120 MOPS, 3 PEs are needed. By slightly increasing the critical path length, 2 PEs may also be enough.

Rule 5. Group behaviors by evenly distributing parallel behaviors among different groups and making group 1 as full as possible. After the number of groups is determined, behaviors are assigned to groups. Parallel behaviors are evenly distributed into different groups to take advantage of parallelism. This load-balancing heuristic can achieve high PE utilization and minimal critical path. The way to fill up groups will guarantee the critical path always lies in the first group.

Rule 6. Select the lowest cost PE for each group while timing constraints are satisfied. After behaviors are grouped, designers select components out of component database for each group. There can be a variety of different algorithms for component selection under different design constraints. A useful rule here is to select the lowest cost PE that can satisfy the time constraint for all groups.

3.2.2 Examples

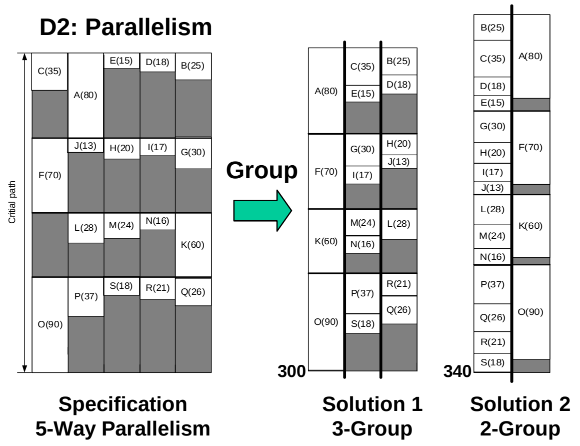


Figure 7. Two Grouping Solutions.

The example specification shown in Figure 7 has 5-way parallelism in the specification. The time constraint for the design is 10 ms. The component database has three components *PE1, PE2* and *PE3*. For the feasibility check, let's assume the critical path (A-F-K-O) can be executed on *PE3* within 10 ms time constraint. Now we need to find out the number of groups for grouping behaviors. The total number of operations can be found to be 640 MOPS, while it is 300 MOPS on the critical path. Therefore, we conclude we need 640/300 (= 3) groups. However, using only 2 groups will not have noticeable increase of critical path length. Therefore, we can try two different candidates. Using Rule 5, we can come up with 2 grouping solutions with numbers of groups being 2 and 3, respectively (Figure 7).

D6b: Processor Database

Proc.	Clock	Power	Cost
PE1	60 MHz	0.5W	$2
PE2	100 MHz	0.7W	$5
PE3	133 MHz	1.0W	$10

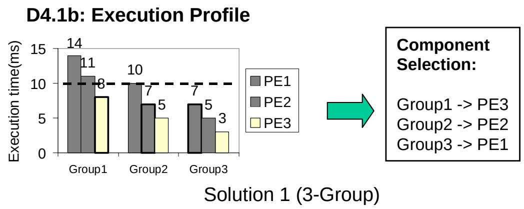


Figure 8. Component Selection for Solution 1.

For each solution, a component is selected for each group. Solution 1 is used here for illustration. Behavior Profile displays the execution time of each group on all components for solution 3. As we can see, group 1 must be executed on the fastest component *PE3* to meet 10 ms constraint. Therefore *PE3* is selected for group1. Since both *PE2* and *PE3* can finish group 2 on time, we select *PE2* because of it is less expensive. Similarly, *PE1* is selected for group 3 (Figure 8). Then we can calculate the execution time, power consumption and cost for both solutions and choose a better one (Figure 9).

(Time constraint = 10ms)

	# of PEs	Execution time	Total power	Total cost
Solution1	3	8ms	2.2w	$17
Solution2	2	9ms	2.0w	$20

Figure 9. Comparison of 2 Solutions.

3.3 Behavior Ordering

After component selection and behavior mapping, designers need to order the behaviors on each PE. A default execution order derived from the original specification can be used as a starting point by designer for further ordering in order to improve performance.

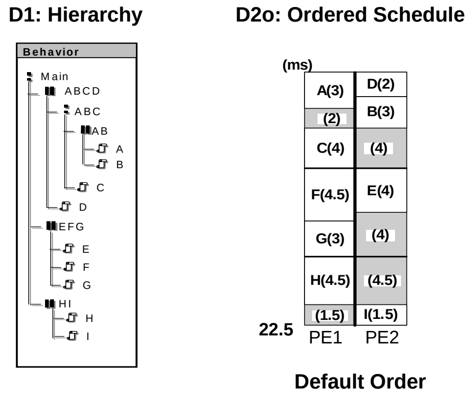


Figure 10. Default Order of Behaviors.

3.3.1 Rules

We suggest three important rules for behavior ordering.

Rule 7. Among all parallel behaviors, the one with most amount of output traffic executes first. Parallel behaviors can be executed in any order. However it may influence the starting times of other behaviors as well as the overall execution time of the design. By executing behaviors with heavy output traffic first, we can reduce the pressure from their depending behaviors thus produce a better schedule. Output Traffic Display (D4.3) shows the output traffic for each behavior.

Rule 8. Move independent parallel behaviors into un-utilized time slots in the same PE.

Rule 9. Move independent parallel behaviors into un-utilized time slots in other PE.

3.3.2 Examples

We give an example to illustrate above three rules. In Figure 10, the default order is shown in Ordered Schedule and the original hierarchy is shown in Hierarchy. By comparing the output traffic of parallel behaviors *D* and *B* displayed in Traffic Profile, we conclude that *B* should be executed before *D* because *B* has greater output traffic, according to Rule 7. Then, according to Rule 8, we can move behavior *I* to an un-utilized time slot in *PE2*. Finally, we use Rule 9 to move the behavior *G* from *PE1* to *PE2* based on the display Behavior Profile. After behavior ordering, Ordered Schedule is displayed in the right-bottom corner. The execution time saved in the improved schedule is 5.5ms (25%).

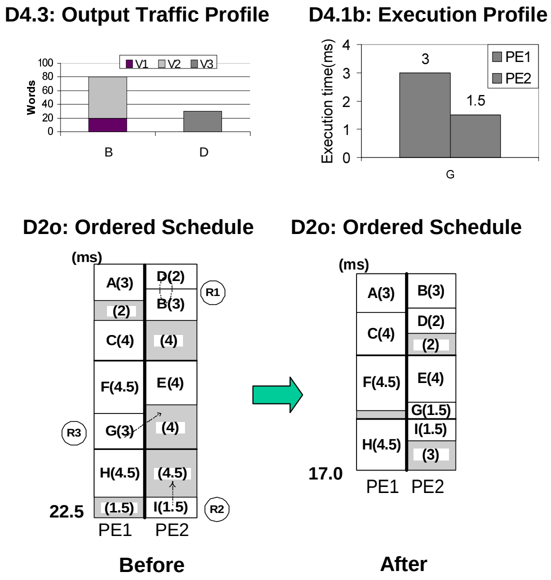


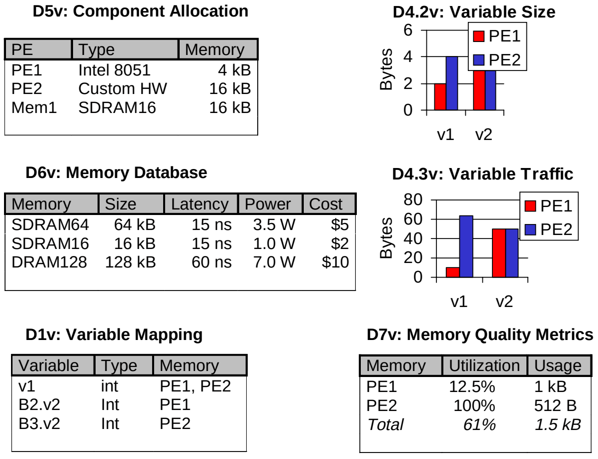
Figure 11. Improved Order of Behaviors.

4 Variable Mapping

After behaviors are mapped to PEs, variables in the specification need to be assigned to memories. Usually PEs have their own local memories and variables can be mapped to these local memories if space allows. If local memory space is not sufficient, a dedicated memory component needs to be allocated to store variables. In general, the introduction of memory component will increase cost and area. In addition, accessing variables in the shared memory tends to be slower than in PE's local memory. Therefore, it is wise to use the local memories of allocated PEs as much as possible and only to allocate memory component if needed.

4.1 Data Displays

The displays needed for variable mapping are shown in figure 12. **Memory Database (D6v)** lists all available memory components with their size, latency, power and cost information. **Component Allocation (D5v)** displays the local memory sizes of allocated PEs. It is also used by designers to input memory allocation decisions. **Variable Size (D4.2v)** gives the storage requirement for each variable. The sizes of the same variable can be different when stored on different PEs. **Variable Traffic (D4.3v)** displays the potential traffic (product of variable size and access frequencies in and out of PEs) generated by each variable, if



D5v: Component Allocation

PE	Type	Memory
PE1	Intel 8051	4 kB
PE2	Custom HW	16 kB
Mem1	SDRAM16	16 kB

D4.2v: Variable Size

D6v: Memory Database

Memory	Size	Latency	Power	Cost
SDRAM64	64 kB	15 ns	3.5 W	$5
SDRAM16	16 kB	15 ns	1.0 W	$2
DRAM128	128 kB	60 ns	7.0 W	$10

D4.3v: Variable Traffic

D1v: Variable Mapping

Variable	Type	Memory
v1	int	PE1, PE2
B2.v2	Int	PE1
B3.v2	Int	PE2

D7v: Memory Quality Metrics

Memory	Utilization	Usage
PE1	12.5%	1 kB
PE2	100%	512 B
Total	*61%*	*1.5 kB*

Figure 12. Variable Mapping Displays.

not mapped locally. **Variable Mapping (D1v)** is used for designers to input variable mapping decisions, i.e., which variable goes to which memory. **Memory Quality Metrics (D7v)** gives feedback on variable mapping in terms of memory utilization.

4.2 Rules

As we mentioned above, PEs' local memories are used as much as possible when variables are mapped. There are two kinds of variables, local variables and global variables, accessed by each PE. Local variables are accessed internally by a single PE while global variables are accessed by multiple PEs. Each variable will potentially generate some traffic (defined as product of number of accesses and variable size) if the variable is not mapped to the PE that accesses it. In order to minimize the traffic among PEs, we will give higher priority to variables with more traffic when assign them to PE's local memories.

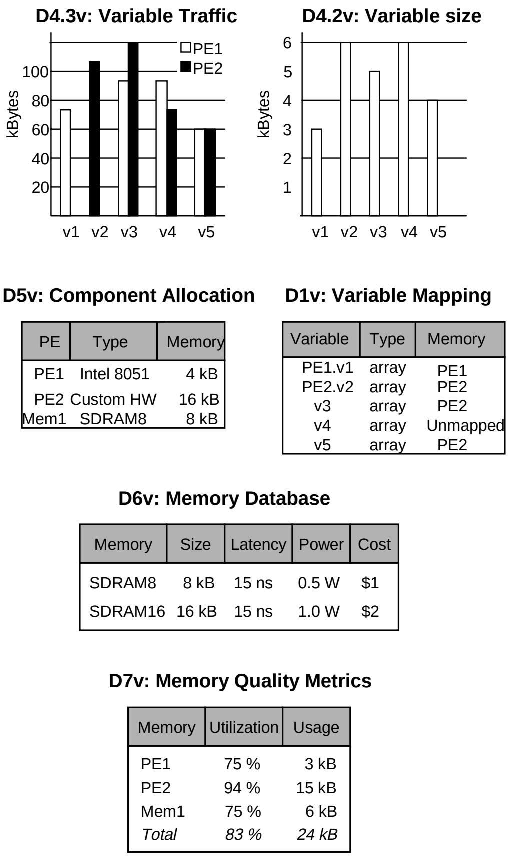
Rule 11. PEs' local memories are utilized before memory components are. We want to utilize PEs' local memory as much as possible. Variables are assigned to the allocated global memory component only if the available local memory space is not sufficient.

Rule 12. Local variables are assigned before global variables. Local variables should be assigned to their accessing PE as much as possible to reduce access time.

Rule 13. Variables are assigned in the descending order of potential traffics. The variables that potentially generate most traffic are assigned to local memories to reduce access time and traffic over busses.

Rule 14. Allocate the lowest cost memory from memory component database to hold all unmapped variables. At stated in Rule 11, some variables may not be mapped to PE's local memories due to memory space limit. In this case, a global memory component is needed for storing these unmapped variables. Usually, the lowest cost memory is selected while size requirement is satisfied.

4.3 Examples



D4.3v: Variable Traffic

D4.2v: Variable size

D5v: Component Allocation

PE	Type	Memory
PE1	Intel 8051	4 kB
PE2	Custom HW	16 kB
Mem1	SDRAM8	8 kB

D1v: Variable Mapping

Variable	Type	Memory
PE1.v1	array	PE1
PE2.v2	array	PE2
v3	array	PE2
v4	array	Unmapped
v5	array	PE2

D6v: Memory Database

Memory	Size	Latency	Power	Cost
SDRAM8	8 kB	15 ns	0.5 W	$1
SDRAM16	16 kB	15 ns	1.0 W	$2

D7v: Memory Quality Metrics

Memory	Utilization	Usage
PE1	75 %	3 kB
PE2	94 %	15 kB
Mem1	75 %	6 kB
Total	*83 %*	*24 kB*

Figure 13. Variable Mapping Example.

An example is shown in Figure 13. First, according to Rule 11 and Rule 12, we map PE1 and PE2's local variables (*v1*, *v2*)to their local memories respectively. After the mapping, *PE1* has 1kB local memory space left and *PE2* has 10kB local memory space left. Three global variables, *v3*, *v4* and *v5*, are accessed by both component *PE1* and *PE2*. We start to assign them in the order of their potential traffics shown in Variable Traffic display. *v3* is assigned to *PE2* only, because *PE1* has only 1kB free space left, which is not big enough for *v3*. *v4* can not be assigned to either *PE1* or *PE2* because of its size. However, *v5* can be barely assigned to *PE2*.

Now, *v4* with size 6kB is not mapped to any local memory. According to Rule 14, a SDRAM8 memory component *Mem1* is selected to hold *v4*. Finally, the Memory Quality

Metrics shows the utilization information of all three memories.

5 Channel Mapping

At this stage, the communication between PEs is through variable channels. Since in the architecture model, the connection among PEs are busses, we need to select busses to connect PEs and map variable channels to the selected busses. For a design consisted of a number of PEs, usually more than one bus is needed. Transducers are then needed to interface between different bus protocols. Our goal here is to minimize the use of transducers and the communication over transducers. The basic approach to bus allocation problem is to cluster PEs and select bus for each PE cluster. Then the mapping of variable channels to busses becomes automatic.

5.1 Data Displays

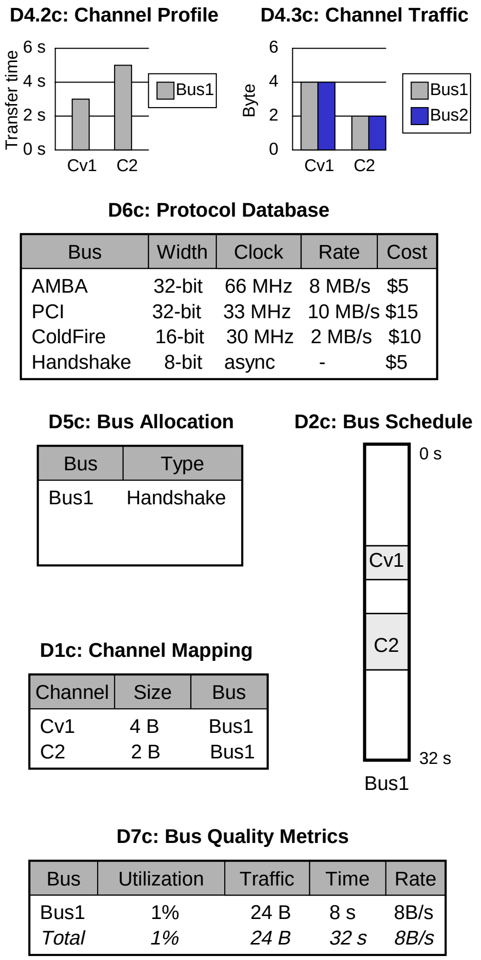
The displays needed for channel mapping are shown in figure 14. **Channel Profile (D4.2c)** gives the transfer time needed for the selected busses to send a channel message. **Channel Traffic (D4.3)** presents traffic (product of size and number of messages in and out PEs) generated by each channel. **Protocol Database (D6c)** lists all available bus protocols with bus width, clock, and cost attributes. **Bus Schedule (D2c)** shows the starting time and duration of each channel message on each bus. **Bus Allocation (D5c)** is used by designers to input bus allocation decisions. **Channel Mapping (D1c)** is used by designers to input channel mapping decision, i.e., which channel is mapped to which bus(ses). **Bus Quality Metrics (D7c)** shows feedback on channel mapping in terms of utilization, traffic, and so on.

5.2 Rules

The following are the most important rules for channel mapping.

Rule 14. Cluster components by closeness and compatibility. The closeness can be defined as either the number of channels or the total amount of traffic between PEs. The compatibility indicates whether two PEs can communicate using a common bus protocol. While custom PEs can be connected to any protocol (thus are compatible with all other PEs), IP or microprocessors usually can only accept their own fixed bus protocols.

Rule 15. Select bus protocol for each cluster. After clustering, the PEs in each cluster are allocated a bus to connect among them. If the cluster includes a PE with a fixed protocol, this protocol is then selected to connect all PEs in the cluster. If all PEs are synthesizable, we can select a bus protocol from the protocol database by consideration the tradeoff between cost and performance.



D6c: Protocol Database

Bus	Width	Clock	Rate	Cost
AMBA	32-bit	66 MHz	8 MB/s	$5
PCI	32-bit	33 MHz	10 MB/s	$15
ColdFire	16-bit	30 MHz	2 MB/s	$10
Handshake	8-bit	async	-	$5

D5c: Bus Allocation

Bus	Type
Bus1	Handshake

D1c: Channel Mapping

Channel	Size	Bus
Cv1	4 B	Bus1
C2	2 B	Bus1

D7c: Bus Quality Metrics

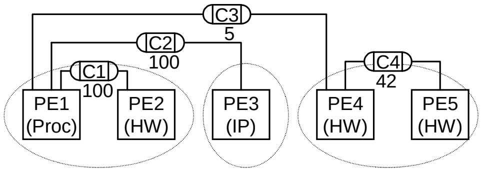
Bus	Utilization	Traffic	Time	Rate
Bus1	1%	24 B	8 s	8B/s
Total	*1%*	*24 B*	*32 s*	*8B/s*

Figure 14. Channel Mapping Displays.

Rule 16. Introduce transducers between different protocols. With busses selected, variable channels are then mapped to allocated busses. For channels that connect PEs both in the same cluster, they are mapped to the corresponding bus directly. For channels connecting PEs in two different clusters, they are mapped to both busses of the two clusters and a transducer is inserted between these two busses.

5.3 Examples

In the example shown in Figure 15, 5 PEs are connected with 4 channels. The closeness, defined as the amount of traffic, and compatibility (shown with asterisk) are tabulated. We start to cluster PEs by looking at closeness and compatibility. For example, *PE1* and *PE2* are clustered together because they are closest and compatible. In the same way, *PE4* and *PE5* form a cluster and *PE3* itself becomes a



D7c. Closeness and Compatibility Matrix

	PE1	PE2	PE3	PE4	PE5
PE1					
PE2	100 *				
PE3	100	0			
PE4	5 *	0	0		
PE5	0	0	0	42 *	

Figure 15. Channel Mapping Example.

cluster. Since *PE1* has its own bus protocol, that protocol (Bus1) is used between *PE1* and *PE2*. *PE4* and *PE5* are all custom component, thus a flexible decision can be made to select the desired bus (Bus3). For IP component *PE3*, its own bus protocol (Bus2) is used. Channel *C1* is mapped to *Bus1* because its connect PEs in the same cluster. By the same reason, channel *C4* is mapped to *Bus3*. On the other hand, *C2* is mapped to both *Bus1* and *Bus2* with transducer *IF2* in between because *PE1* and *PE4* are in different clusters. Similarly, *C3* is mapped to both *Bus1* and *Bus3* with transducer *IF1* in between. The final channel mapping result is shown in Figure 16.

Clustering
PE1, PE2 -> Bus1
PE3 -> Bus2
PE4, PE5 -> Bus3

Channel Mapping
C1 -> Bus1
C2 -> Bus1, Bus2
C3 -> Bus1, Bus3
C4 -> Bus3

Final Architecture

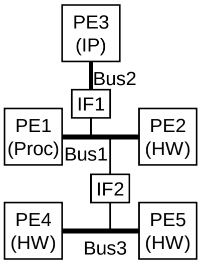


Figure 16. Channel Mapping Result.

6 Conclusions

In this report, we presented the tasks of system level design. An interactive design flow can help designers perform these tasks. The displays and suggested rules to use the displays at each design step were described and illustrated with examples. Designers are not limited to the aforementioned rules. New rules can be added to utilize the displays for better design results. As we can seen, the flow enables extensive architecture exploration with minimum design effort from designers.

7 Acknowledgments

The authors would like to thank Semiconductor Research Corporation for its support on this project.

References

[1] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers, March, 2000.

[2] A. Gerstlauer, R. Dömer, J. Peng, D. Gajski, *System Design: A Practical Guide with SpecC*, Kluwer Academic Publishers, May, 2001.