Codesign Methodology for ATM Application Domains

Jon Kleinsmith
Tatsuya Umezaki†
Zhuzhen Kang
Daniel D. Gajski

Technical Report ICS-97-14
January, 1997

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA
(714) 824-8059

† Hitachi Ltd., Yokohama, Japan

Abstract

With the recent explosion of ATM technologies, continuing increases in design complexity and shortened time-to-market demands, new design methodologies coupled with powerful CAD tools are becoming necessary to keep abreast with consumer need in this high-performance/memory intensive application area. Using a system-level approach to design, we demonstrate that a significantly shorter design cycle may be achieved. This methodology is explained on the example of an ATM filter.

Contents

List of Figures

1 Introduction

The expanding base of ATM products and services has largely depended upon product designs that are reliable, cost effective and flexible enough to serve a diverse customer base. In particular, ATM devices are typified by high-performance, memory intensive designs. In order to meet time-to-market demands and increasing application complexity, evolution and customization, designers are forced to start their designs at higher-levels of abstraction using powerful CAD tools to assist in the design process.

In order to drastically reduce the design cycle time with proper design trade-offs, our research has focused on a system-level design methodology employing a single design language and using system-level CAD tools, partition the design into applicable hardware, software and interface compontents to be handed off to appropriate design tools for simulation, synthesis and modeling.

For the remainder of this paper, we will demonstrate a new methodology on an example of an ATM filter and the details of the tasks at each step in the design process. We then illustrate the power and speed of our methodology by evaluating the design steps on this a general design model.

2 Example Specification Description

In order to evaluate the design process and illustrate the needed methodology and tools, we selected a general design model of an ATM product. The particular application that we used was the High-Level Data Link Controller (HDLC), an ATM filter. Our goal was to specify the design at the highest abstraction level, evaluate the effectiveness of our methodology and compare it to manual design. Another important goal was to gauge the approximate time required for completion of the design from conceptualization to implementation.

In this section we provide an overview of the HDLC filter. We then discuss the ATM design experiment and the tradeoffs that were achieved in each stage of the design process.

2.1 HDLC filter

The HDLC filter is a generalized design model of ATM products, designed to accept as input HDLC Frames, a LAN digital packet format, and decompose this input into ATM packets. The model is also capable of performing the reverse process of concatenating ATM standard packets into HDLC Frames. In the remainder of this paper, we will refer the HDLC Frames as *frames* and ATM packets as *packets* . The flow-graph of the HDLC filter can be seen in Figure 1

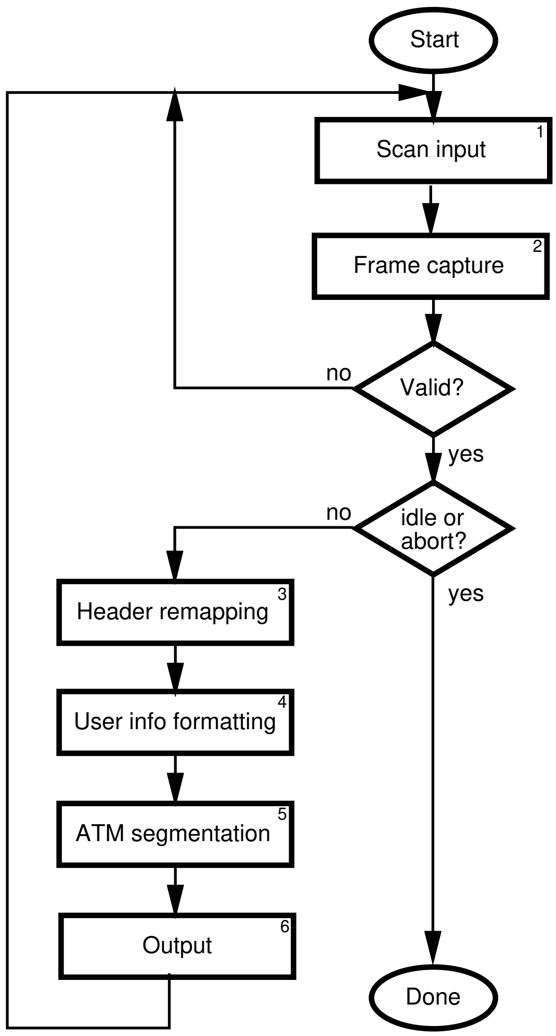

Figure 1: Flowgraph of the HDLC filter

We can view the HDLC filter as a series of tasks that execute sequentially. In order to perform ATM conversion, the HDLC filter must scan a stream of data, capturing all valid *frames* through the recognition of flag data. This is accompished in box 1 of Figure 1. The filter then recognize various functional flags within the body of a frame. After a valid *frame* is captured, a series of checks are performed to ensure *frame* integrity. This is accomplished in box 2 in the figure. If an abort or idle flag is not encountered, and the *frame* is valid, then the *frames* are decomposed into ATM *packets* . In box 3, header information from the HDLC *frame* is translated into the ATM header format. Box 4, represents the task of splitting the information portion of the frame into equal length segments to be placed into a sequence of *packets* . Finally, in boxes 5 and 6, header information is duplicated across all *packets* and these are then output.

In addition, the HDLC filter is also capable of converting ATM packets into HDLC frames via a process similar to the one outlined above. ATM packets are captured and checked for validity. Common header information is grouped and user information in the body of the *packets* is concatenated to form the HDLC frame. The constructed *frame* is then output.

2.2 Design Specification

In order to reduce design time, the design is specified at the highest possible level of abstraction using a model that closely resembles the way designers would conceptualize a design as presented in 2.

Capturing the functionality of the HDLC filter at the system level was made possible through the use of the SpecCharts language [1]. Using a model called **Program-state Machines (PSM)**, which is based on a combination of the hierarchical finite-state machine and programming language paradigms [2], a system can be specified as a hierarchy of program-states. These program-states may either be composite, consisting of lower-level states, or they may be bottom level or leaf states. Further, composite program-states will contain a set of concurrent program-states (all of them active) or sequential program states (only one of them active). At the lowest level, the leaf program-states have a sequence of computations expressed using programming language statements.

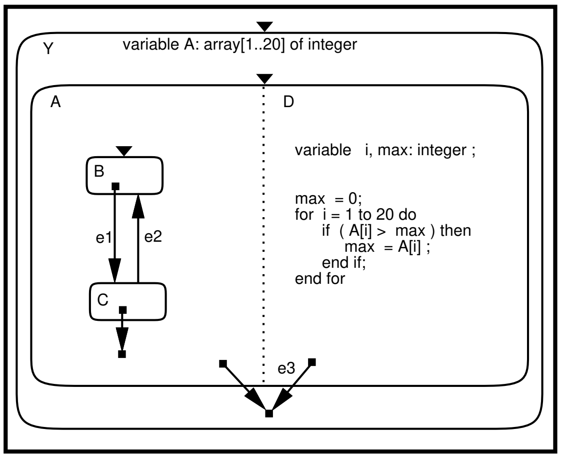

Figure 2: Program State Machine

The computations in the leaf program-states are expressed with VHDL programming statements. They contain three object classes: **variables** that store data, **behavior** that represents an algorithmic level computation, and **channels** that move data between behaviors. Each of these subbehaviors can be modeled using a finite-state machine with datapath (FSMD). The FSMD paradigm allows the designer to capture the behavior in a fashion that is understandable and easily realized in hardware. Essentially, the FSMD is divided into two parts, a controller realized by the FSM and a group of functional unit performing computations.

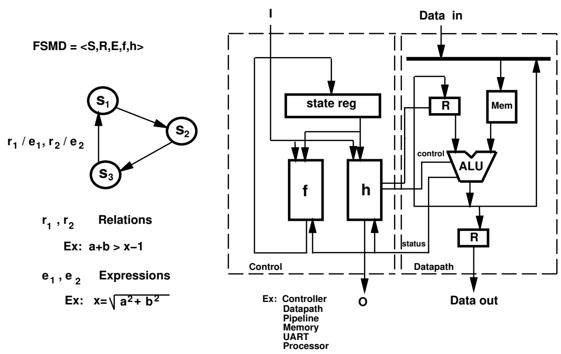

Figure 3: Definition of the FSM with Datapth

In Figure 3 we define the FSM in terms of a set of states, S_i, these states are connected via relations r_i and expressions e_i. In the FSMD model, the relations realized through a status signal generated by computations on the datapath. These relations, coupled with the current state of the FSM, drive expressions, which are realized by control signals from the FSM to the datapath driving the associated computation. The set of possible control values on this datapath can be viewed as the instruction set of the FSMD. The power of this model is seen in its ability to represent not only a hardware implementation, but as software running upon a general processor.

Based on the PSM model, the composite program states are constructed by combining two or more such VHDL based leaf program-states such that they execute sequentially or in parallel. The resulting architectural model can then be seen as a hierarchical concurrent FSMD (HCFSMD) [3].

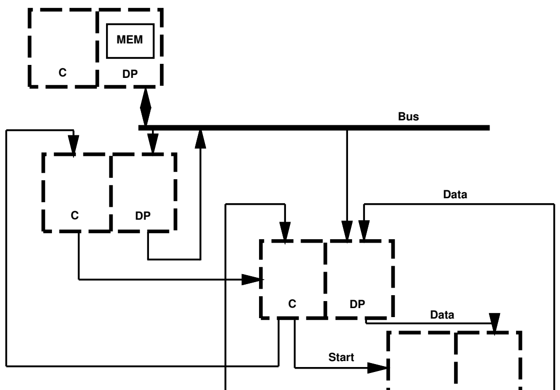

Figure 4: HCFSMD Example

The model given in Figure 4 shows a set of interconnected FSMDs. Each block represents a single FSMD, divided to indicate the control **C** and datapath **DP** connectivity in a hierarchical fashion. The system shown in this figure illustrates both concurrent and sequentially running processors. The FSMD above the databus is operating independently and concurrently with the other FSMDs in the system. Below the databus, the series of FSMDs are connected through control and data can model a variety of sequential operations, including function or subroutine calls and co-processing.

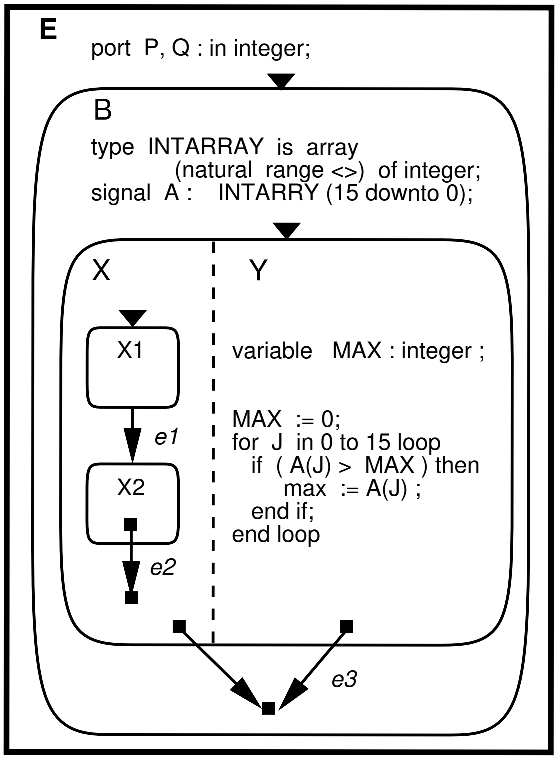

Figure 5: SpecChart Example

As seen in Figure 5, SpecCharts closely follows the PSM model. SpecCharts allows high level specification, completely describing the behavior of a system with few implementation details since the description consists of pure behavior. This simplifies the specification enabling us to achieve a greatly reduce design cycle. SpecCharts was designed to serve as a front end for VHDL[4]. As such, the syntax and semantics of the language are well defined, reducing the number of possible errors caused by language ambiguities. Translators have also been developed convert SpecCharts into VHDL, allowing for easy simulation of the specification using various commercially aviailable tools.

Our specification of the ATM filter consisted of eight behaviors expressed using VHDL process statements. Four of these processes model the ascent portion of the HDLC conversion, while the remaining four represent the descent portion of the design. The system-level design in SpecCharts is represented graphically in Figure 6.

This SpecChart can be seen as very closely representing the behavior given in Figure 1. The main entity has been labeled HDLC/ATM Filter. This entity is composed of two sequential subbehaviors, *PowerOn* and *PowerOff*. The behavior *PowerOff* represents a null state when the unit is not functional or is receiving no power. The transition labeled with the relation $power = 1$ represents "turning on" the unit.

The *PowerOn* behavior is comprised of two sequential behaviors, *Initialize/Idle* and *HDLC*. The initialize behavior is responsible for setting the initial state of the system. When finished, the *HDLC* behavior is entered. This behavior consists of two concurrent subbehaviors *HDLC Ascent* and *HDLC descent*, representing the HDLC to ATM and ATM to HDLC conversions, respectively. In turn, these concurrent behaviors are comprised of sequentially running subbehaviors which are leaf behaviors in this case. As shown, the *HDLC Ascent* behavior will first capture a *frame* . The headers are remapped in the next leaf behavior and upon completion, the user information is formatted for ATM distribution. Finally, the ATM packets are output from the filter before the process is repeated. Conversely, the *HDLC Descent* behavior, running in parallel with *HDLC Ascent*, is responsible for capturing ATM packets and after formatting, outputting HDLC frames.

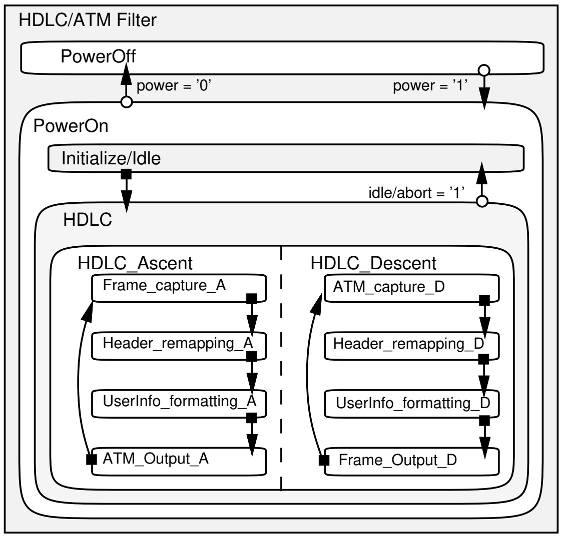

Figure 6: SpecCharts Capture of HDLC Filter

2.3 System Design Tasks

The actual design steps consist of three sets of tasks, which include, (a) **System design tasks** that deal with mapping the specified functionality into multiple chips. (b) **Architectural design tasks** that deal with designing the architecture (i.e., control and datapath) for each of the chips, (c) **Technology adaptation tasks** that deal with mapping the components in the chip to actual physical implementations[5].

System Design involves mapping the functionality, as captured in the SpecCharts specification to a set of chips, such that all design constraints are met. The tasks at the system level deal with a very coarse granularity of objects (i.e., variables, behaviors and channels in the specification). Our approach to system design consists of three well-defined tasks on these objects. (a) allocation of chips, to satisfy the overall cost constraints, (b) partitioning of objects in the specification to satisfy performance constraints, and (c) refinement of the specification into an architectural model for determining communication and performance.

Figure 7 represents the target architecture for these system design tasks. In this figure, the system can be seen as collection of processing entities PE_i, interfaces IF_i, and memories connected via buses. The processing entities can be processors, ASCIs or FPGAs, etc. with associated local memories representing local variables. Communication is conducted through local buses without arbitration. Global variables, buffers and the like are partitioned into the global memories and communication between processes is along the buses through interfaces linking potentially different protocols associated with these components. Arbiters may be needed to resolve contention between processes and processes and memory.

The SpecSyn [6] [7]environment allows the designer to rapidly explore a variety of designs, maintain-

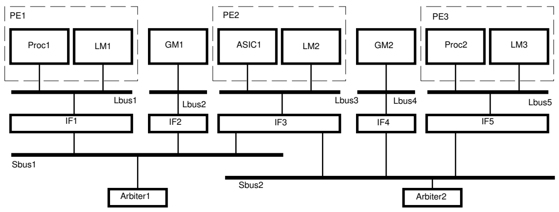

Figure 7: Target Architecture

ing control over any particular allocation, partition or transformation at the system level of abstraction. The designer may select system-level component types such as different processors, ASICs, FPGAs and others. The design can then be partitioned either manually or automatically in a matter of seconds based upon constraints placed on the system by the designer.

In our example system, as with many ATM applications, throughput is the most critical constraint. An typical throughput requirement was used for the processing of a group of *frames* . The chart below lists the constraints upon the HDLC unit which we set ourselves.

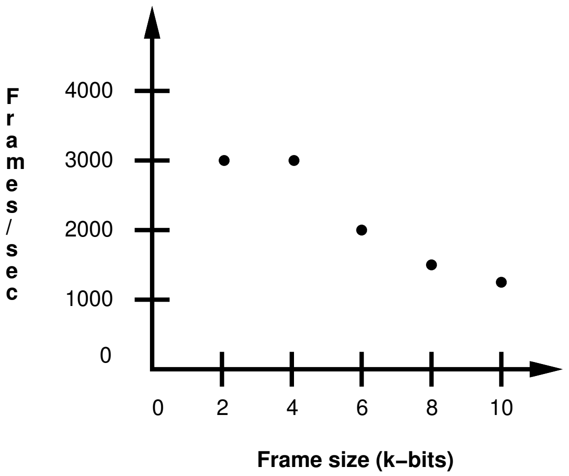

Figure 8: Performance Constraints upon the HDLC Filter

Using SpecSyn, we initially select an all software implementation, assigning all behaviors in the HDLC filter to a single processor. SpecSyn calculates the processing time of the processor by converting the behavioral description into three-address machine instructions, and based upon the instruction set and processing cycle of the processor, determines the number of clock cycles necessary to perform the associated behavior. This partition revealed that a single processor would not be sufficient for our design due to violation of performance constraints. Allocating additional hardware, such as an ASIC or an additional processor was attempted, migrating computationally intensive behavior to these high-speed units. yielding designs that satisfied our constraints. In this fashion, we were able to rapidly explore various software/hardware configurations to determine viable solutions, finally opting for an implementation which partitioned the *Frame Capture A* and *ATM Output A* processes to individual ASICs while the remaining behaviors were assigned to software and a single processor.

SpecSyn can generate VHDL code to represent the partitioned design in order to perform simulation or verification and provide the basis for further design synthesis efforts at lower levels of abstraction.

For example, as stated in the previous paragraphs, the *Frame Capture A* behavior was partitioned onto an ASIC from a generic library. In order to fabricate the HDLC filter system, this ASIC will in turn need to be designed and implemented. As an RT level component, the ASIC may be modeled using the FSMD paradigm using a behavioral compiler such as BdA (Behavioral design Assistant) of ISE (Interactive Synthesis Environment). BdA allows a designer to perform the design tasks mentioned in our methodology, but at a lower level of abstraction with higher granularity. The calculations within *Frame Capture A* can be can be allocated to RT-level functional units such as adders, subtractors, ALUs and the like, memory components such as RAMs, ROMs and registers. Control of this datapath is modeled as a finite-state machine constructed of combinatorial and sequential logic to drive execution across the datapath.

At this time, however, Speccharts and SpecSyn omit a few key design conciderations that would be useful both at this level and at lower levels of design. Foremost, timing and execution constraints are not taken into account or modeled at the system level, rather, rough estimates are made on process execution after component allocation has been attempted. For example, upon allocation of a custom ASIC, a timing and area estimate is made based upon likely functional units, varaible and functional lifetimes. Contraints upon the various behaviors of a system may be applied in a global sense, however, constraints upon individual processes or procedures is not supported in the SpecCharts language.

Additionally, dataflow is absent from the SpecCharts language. SpecCharts allows control-flow specification between variouos behaviors of a system, but does not model the passage of data between behaviors, rather, the language and design environment assign shared data to global variables and as the design is partitioned, these global variables are mapped to shared memory. This model is overly simplified and does not depict complex encoding and communication between real components.

BdA will accept a VHDL process specification and, following manual allocation of functional units and memories, will schedule and bind the specification to the functional units and generate appropriate control. The FSMD netlist for the ASIC could also be derived manually by simply enumerating the operations performed within the behavior and allocating corresponding functional units. Scheduling and binding can then be performed by assigning particular operations to particular functional units. The use of a State-Action Table to define the time-steps and actions to be performed is especially useful and can be developed from an flowgraph of the algorithm wherein each state consists of operations that may be performed simultaneously given adequate resources. The state-action table for the *Frame Capture A* behavior is given below.

State Action Table for Frame Capture A

	Next-state		Datapath actions
State	Condition	S	operation
0	On = 1 On ≠ 1	1 0	Length ← 0 Counter ← 0
1	Frame(7..0) = Flag Frame(7..0) ≠ Flag	2 1	Frame ←Frame ≪ IData Length ← Length+1
2	Frame(7..0) = Flag Frame(7..0) ≠ Flag	4 3	Frame ← Frame ≪ IData Length ←Length+1
3	Length> MAX Length≤ MAX	5 0	
4	Length≥ MIN Length< MIN	0 6	
5	Frame(7..0) = Flag Frame(7..0) ≠ Flag	0 2	
6		7	Count ←Length−15
7	Frame(15..0) = Idle Frame(15..0) ≠ Idle	9 8	
8	Count = 0 Count ≠ 0	10 7	Frame ← Frame ≪ 1 Count ← Count −1
9		0	CheckIdleFlag ← 1
10		11	Count ← Length−22
11	Frame(5..0) = Zero Frame(5..0) ≠ Zero	13 12	
12	Count = 0 Count ≠ 0	14 13	Count ←Count −1 Frame ← Frame ≪ 1
13		11	Frame(5..0) ←Frame(5..0) ≪ 1
14		15	Count←Length −16
15	Frame(MSB) = 0 Frame(MSB) ≠ 0	16 17	
16		17	Frame(16..0) ←Frame(16..0) *XOR* CRC16Equation
17	ShiftCount = 16 ShiftCount ≠ 16	18 15	Frame ←Frame≪ +1 Count ←Count−1
18	Frame(16..0) = 0 Frame(16..0) ≠ 0	20 19	Count ←Count+1
19	FlagCount > 6 FlagCount ≤ 6	1 21	
20		0	OUT ← Frame
21	Frame(5..0) = Flag Frame(5..0) ≠ Flag	0 21	Frame ← Frame≪ IData

The corresponding RT-level block digram can be seen in Figure 9.

3 Design Methodology

As stated earlier, specification at the highest level of abstract allows the designer to use a model that closely resembles the conceptualization of the design as presented in 2. Our example and experimentation suggests a methodology based upon this high level of abstraction that includes design tasks that occur at lower level with a higher granularity.

ourpsfigexact

In Figure 10 we present an outline of the design flow within our proposed methodology.

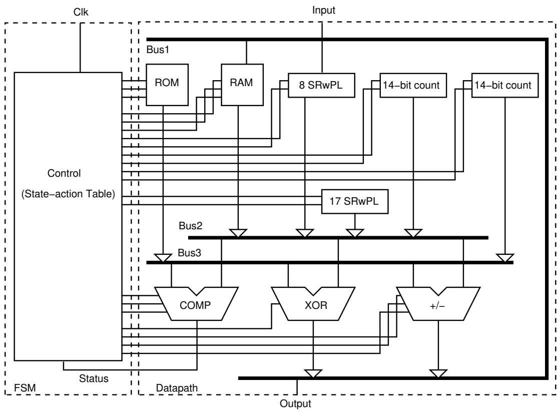

Figure 9: FSMD implementation of the HDLC Frame Capture Behavior

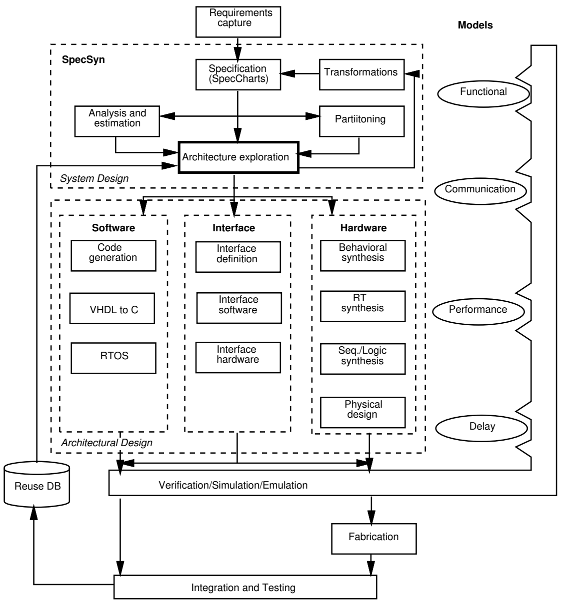

Figure 10: System Design Flow

3.1 System Design Tasks

The purpose of system design is to take a high-level specification and given a set of constraints, partition the behavior onto appropriate system-level components.

3.1.1 Specification

Specification at the highest abstraction level is necessary in order to quickly capture the behavior of a complex system. SpecCharts, following a program-state machine model, allows the designer to specify the behavior of the design, capturing both hierarchy and concurrency at a conceptual level closely matching that of the design at this high level enabling understandability[8]. Timing and constraint specification are currently left for lower level models, however, inclusion of this information at high levels of design would allow the designer to avoid potentially infeasible designs that might be entertained with the current specification language.

Using standard languages like VHDL, (or front-ends based on VHDL) we exploit the vast support available for them. Further, designers may avoid a large number of specification errors, since the syntax and semantics of the front-end language closely match design characteristics. Manual refinement and transformation is still a problem, in that, the designer must take information supplied be the sythesis tools and apply transforamtions to the specification by hand in an attempt to meet constraints. This process can be time consuming and error prone. Future work may attempt to automate this process, allowing the designer to more efficiently refine the model towards a final manufacturable product.

3.1.2 Partitioning

After architecture selection and manually allocating the chips in the system, through library selection, the behaviors in the specification have to be partitioned and mapped onto these chips. Thus **partitioning** forms the next important task at the system level. During the partitioning process, variables are mapped to memories, behaviors are mapped to standard/custom processors, and channels are mapped to global variables residing in shared memories.

The current partitioning method and objects modeled are fairly simplistic. The shared memory model does not coincide with real-world communication schemes and do not capture the encoding of data from one behavior or component to another.

3.1.3 Exploration

Generally, there are a large number of partitioning solutions for a given allocation. However in order to achieve our rapid design turnaround goal, we need fast partitioning algorithms that can rapidly explore this large solution space. Clustering based partitioning is an example of a fast partitioning algorithm. Use of such algorithms in the partitioning process, can result in a large number of solutions being explored quickly. Fast estimates are required to evaluate the quality metrics for each partitioning tried by the algorithm.

The partitioning criteria varies for different types of allocation. Partitioning a system onto a set of ASICs requires a different cost metric compared to partitioning a system onto processors. In order to enable the partitioner to explore different types of allocation, the closeness criteria for the partitioning algorithms can be selected by the designer.

3.1.4 Analysis and Estimation

Verification of functional correctness and an estimation of design quality with respect to applied constraints is required at this high level of abstraction in order to rule out incorrect or infeasible designs. Since actual physical qualities such as cost and performance cannot be determined precisely at this stage, estimates of these qualities must be derived. For hardware partitions, area, pin, cost, speed and power can be estimated through libraries of standard ASIC components. Once a specific library of functional units is decided upon, estimates can be generated based upon their particular attributes. However, current libraries and estimation techniques do not take into consideration a number of important design characteristics, namely, pipeling and communication. As noted before, communication is handled in a very simplistic manner and does not consider the delay of data encoding across communications channels. Additionally, buffering or queueing of data is not modeled or accounted for in performance estimates. Software estimation is also limited. For software systems, instruction estimation, branch prediction and memory size estimation is necessary in order to get an accurate view of processor qualities. These static techniques do not take into account input dependence nor potential compiler optimizations.

3.1.5 Transformations

Transformation of the specification may be necessary in order to improve performance, reduce area or satisfy any number of constraints. The designer must use information gained through observing constraint violations and alter the specification to improve the quality of the design. Currently this is a manual process and is fraught with errors and is very time consumming as trnsformations may be required a number of times throughout the design cycle and repreated iteratively until the design meets all constraints.

3.1.6 Model Refinement

After performing these tasks, a specification must be refined to reflect the allocation and partitioning decisions. The various parts of the specification are moved to their corresponding components and communication is established between these components. This refined specification consists of an executable specification; one for each system component. Each component spec contains VHDL process level statements. The channels between two chips form the input and output ports on the chips. The details of this design process is given in [6].

3.2 Architectural Tasks

Once the specification has been partitioned, allocated and refined into a potentially viable design, the specification must then be further developed or synthesized in order to be fabricated.

The tasks in the architectural design phase are very similar to the tasks in the system design phase, except for the level of abstraction. While system design tasks generates system structure, the architectural design tasks generate individual chip structure and program code.

3.2.1 Software

Partitioning may result in various behaviors being mapped to software. As such, the behavioral specification must be transformed into a compilable format or language for the target architecture. Again, the use of standard languages may be of aid to the designer. Translation from VHDL to C would allow the use of high-performance compilation and wide-spread industry support in addition to compatibility with existing code and systems. Another possibility is to specify the system in C initially and then later translate the hardware portions of the system to VHDL for synthesis.

Further, the system may require dynamic scheduling of events, particularly in the case of reactive systems. A real-time operating system (RTOS) may need to be designed or generated in order to handle the scheduling of tasks or events within a given processor or group of processors.

3.2.2 Hardware

In Figure 10, the hardware design tasks are divided into levels of increasing granularity. At the highest level, *behavioral synthesis* abstract behaviors are mapped to allocated ASICs, FPGAs, or perhaps DSP cores. These components may or may not be off-the-shelf.

At the register transfer (RT) level, we begin by mapping each chip description to a set of microarchitectural components, such that the design constraints for the chip are met. We use the **Finite State Machine with Datapath (FSMD)** model [9] to represent the design at the RT level. This model consists of: (a) a datapath to represent operations, and (b) a finite state machine to represent the sequencing of the operations. A design based on the FSMD contains (a) a datapath that contains functional units such as ALUs, storage units, and interconnect units, and (b) a controller specified in terms of state transitions and the value of the control lines in each state, as explained in Section 2.

At the lowest levels, gate or transistor netlists are generated in order to facilitate manufacture and more precisely depict the performance of the system.

3.2.3 Interface

The coupling of the various objects within the system may require complex communication. The interface between hardware components and software programs, as well as external objects must be specified, captured and designed. Standard components and processors may have fixed communication protocols that require interfaces to exchange data across communication channels.

Also to take into consideration is buffering or temporary storage of data across a communication channel, particularly in the case of ATM applications that require relatively large packet of data to be streamed through a number of processing units. Inconsistent transfer rates will require some buffering of data to be performed. The size or depth of this buffering can be calculated in order to ensure throughput constraints are met.

3.3 Verification, Simulation and Emulation

Throughout the design process, at each level of abstraction, the design must be verified to ensure equivalence across transformations of the behavior. At each step in the design process, the design is realized by a particular model.

- **Functional** At the time of specification, this model is purely *functional*, illustrating the behavior of the system. No allusions to implementation is made.

- **Communication** After partitioning, a *communication model* defines the connectivity between partitioned behaviors assigned to components in the architecture.

- **Performance** As components are synthesized and refined, a cycle accurate *performance model* depicts the size, cost and timing of the design.
- **Delay** Finally, at the lowest level of design, the *delay model* gives a clock period the design.

A means by which to compare each model with another is needed to ensure functional equivalence. This task can be facilitate through automated CAD tools that control the refinement process reducing designer error, and simulation tools to allow comparison of functional output. Further, each model should be in a form that allows the use of simulation or emulation tools. Again, the use of standard languages such as VHDL or C will allow the use of currently available tools and reduce time and effort often associated with new or novel languages and tools.

4 Present Tools

At any given level of abstraction, powerful design tools and environments are needed to aid the designer, due to increasing design complexity and time-to-market constraints. In particular, a design environment is needed that will allow the designer to smoothly model the design at various abstraction levels, maintaining functional integrity and a conceptual representation that facilitates the various design tasks

Our experiments have targeted the SpecSyn environment, as well as the SpecCharts language, BdA and Synopsys simulation tools.

4.1 System Design Tools

In order to facilitate system design, the designer needs a specification language and design environment that affords rapid design capture and exploration of various design tradeoffs at a high-level of abstraction. We believe that SpecCharts and SpecSyn offer such facility.

4.1.1 SpecCharts

SpecCharts allows the designer to rapidly capture the behavior of a given system. Conceptually, the specification closely resembles the designer's view of the system, offering both hierarchy and concurrency in a readily understandable form. Further, SpecCharts leaf code consists of a subset of standard VHDL, making the SpecCharts language easy to learn for designers familiar with VHDL. However, SpecCharts does not facilitate the use or reuse of previously specified components due to its pure behavior approach. In order to allow the use of pre-defined components, SpecCharts must include structural constructs. Currently, should designers wish to utilize off-the-shelf system-level components such a DSP filters, the designer must model and define the requisite behavior and then assign that behavior to a general ASIC that will later be replaced by the actual unit. A more intuitive approach would be to allow the designer to specify that a particular behavior passes data to a previously designed filter from a component library, which in turn may be communicating with other behaviors in the system. Later, during synthesis, the structural abstraction of the filter may be replaced with an appropriate description stored in a library.

4.1.2 SpecSyn

The SpecSyn environment allows the designer to quickly explore a variety of system configurations and determine feasible designs automatically handling partitioning, estimation and refinement. However, we found that several key features are lacking to handle our ATM design methodology. As stated earlier, ATM is very memory intensive, and as such, memory allocation, modeling and communication between behaviors is critical. The concept of a communications channel as a complex, heirarchical component consisting of timing and protocol or encoding information is needed in order to model the communciation between processes in the system. Accurate estimation of bus bandwidth, memory access and process-memory handshaking is necessary in order to accurately determine constraint violation and make informed decisions about design transformations. SpecSyn offers a very simplistic view of memory allocation, ie. shared variables stored in global memories, and does not currently offer specialized memory paradigms such as data cashe, queues or FIFO units.

Current performance estimation for processors allocated by SpecSyn is based upon the translation of SpecCharts code into generic three-address instructions. The limitations to performance calculation include poor branch prediction and loop iteration calculations and predictable impact of compiler optimizations. Future work should include code generation targeted toward a common language such as C, in order to perform compilation and profiling to generate more accurate performance figures.

4.2 Architectural Tools

At the architectural level, BdA assists the designer in realizing a variety of chip layouts. However, BdA offers no support for procedure or function calls, records, constants, input ports, nor *for loops*. In addition, errors where found in the compilation of a behavioral specification into FSMD netlists. BdA offers no type checking between vectors of differing length and cannot handle partial vector assignment [10]. Further analysis of BdA could not be made since language constraints prohibited compilation. ISE appears to be addressing some of these problems and future work will include evaluating ISE under our current methodology.

5 Conclusions

In this paper we have discussed a design methodology aimed at producing quick design turnarounds. We have illustrated the power of this methodology with a model, that was (a) specified at the highest levels of abstractions using SpecCharts (b) synthesized at the system level with SpecSyn and (c) further synthesized at the RT level either manually of via RT-synthesis tools.

In this experiment the design quality is somewhat lower than a comparable manual design, but the design time is orders of magnitude shorter. Since the designer was dealing with higher levels of abstractions, a larger design space could be explored by quickly changing some of the exploration parameters. For example, numerous partitionings of the system level specification and numerous microarchitectures were quickly explored (in a few minutes). Providing hooks to VHDL, allowed verification of the designs at each of the abstraction levels.

As presented earlier, several improvements and additions are needed in order to heighten design quality and more effectively realize the high-performance, memory intensive applications typical of ATM products.

Particular to ATM is the need for buffering of data between components in the system in order to manage differing throughput rates. A typical ATM system consist of various hardware components operating at differing rates. Data producers may out-perform consumers with resulting data loss. SpecCharts and SpecSyn have been more successful with static, control oriented designs and do not adequately handle queueing and the concept of complex data channels between components. In order for SpecCharts and SpecSyn to be effective in ATM design synthesis several amendments should be made, including:

- The addition of the *channel concept* is needed to capture complex communication between components, including *cache, queueing* and *pipelining.* This would allow SpecSyn to model and measure traffic and delays between components in a system.
- Calculation or estimation of queue length and size for use in system throughput estimation.
- Addition of timing and constraints to the specification language in order to make design decisions based upon these characteristics early in the design cycle.
- Generation of interfaces between components in the system, from hardware to software, software

to software and hardware to hardware, including communication channels.

- Automation of transformations for constraint satisfaction, reducing the chance of errors induced through manual transformation.
- Automatic code generation from system-level specification and estimation methods that include profiling and compiler optimization information for software.
- A characterization of library components that more closely models the area an timing of predesigned hardware elements, perhaps automatically generated from timing diagrams supplied by manufacturers.

A single, uniform, high-level model, and an environment designed to bridge the conceptual gap between the various lower-level paradigms within a system is needed to allow the designer to quickly and effectively capture, simulate, refine and implement the whole of a given system.

We believe that the SpecCharts+ language and the SpecSyn environment can be upgraded for future development in this area.

6 References

[1] F. Vahid, S. Narayan, and D. Gajski, "SpecCharts: A Language for System Level Synthesis," in *Proc. of the International Symposium on Computer Hardware Description Languages and their Applications*, 1991.

[2] D. Gajski, F. Vahid, and S. Narayan, "SpecCharts: A VHDL Front-End for Embedded Systems." Technical Report93-31, 1993.

[3] D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems.* New Jersey: Prentice Hall, 1994.

[4] *IEEE Standard VHDL Language Reference Manual*, 1988.

[5] D. Gajski, L. Ramachandran, P.Fung, S. Narayan, and F. Vahid, "100 hr design methodology: A Test Case," in *Proc. EURO-DAC, Hamburg*, 1994.

[6] D. Gajski, F. Vahid, and S. Narayan, "A System-Design Methodology: Executable-Specification Refinement," in *Proceedings in European Conference on Design Automation*, 1994.

[7] D. Gajski, J. Gong, F. Vahid, and S. Narayan, "The SpecSyn Design Process and Human Interface." Technical Report93-3, 1993.

[8] D. Gajski and L. Ramachandran, "Introduction to High Level Synthesis," in *submitted to IEEE Design and Test*, 1994.

[9] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High Level Synthesis.* Boston, Massachusetts: Kluwer Academic Publishers, 1992.

[10] L. Ramachandran and D. Gajski, "Behavioral Design Assistant (BdA) User Manual," tech. rep., Dept. of Information and Computer Science, University of California, Irvine, CA 92717, 1994. Technical Report : 94-36.