# A Methodology for Software-Hardware Codesign

**Jie Gong**
**Daniel D. Gajski**
**Allen C. Wu**

Dept. of Information and Computer Science
University of California, Irvine
October, 1992

**Technical Report 92-94**

## Abstract

In this report we present two approaches for synthesis of real-time systems with a minimal number of application specific integrated circuits (ASICs) while still meeting the required performance constraints. One approach starts with a single process description which can be easily compiled into a software implementation for any standard processor. If this software implementation does not satisfy the required performance, descriptions of the performance-critical parts are extracted out and implemented as ASICs. The other approach starts with a description written as multiple processes communicating through global signals. This description can be naturally mapped to a hardware-only implementation in which each process is implemented as one ASIC. In order to minimize number of ASICs, the processes are merged and split for mapping to a combination of standard processors and ASICs. The step-wise refinement process for both approaches is demonstrated on an example of a real-time system. Issues and tools regarding the automation of the proposed codesign methodology are also discussed.

# Contents



---

# List of Figures

# 1 Introduction

Many real-time applications need both hardware and software components. Software implementation is preferred because of lower cost, shorter design cycle and simpler upgrading while hardware implementation offers performance necessary for real-time systems. Traditionally, the partitioning of a design into software and hardware is made at very early stage and after which the hardware and software are developed separately. With high-level synthesis and software compilation techniques in a fairly mature state, it becomes possible to consider the software-hardware tradeoffs through the entire development cycle and to automate the codesign process by building tools for software-hardware codesign. Those tools allow a designer to trade software for hardware in order to achieve performance and hardware for software to minimize cost.

Software-hardware codesign involves many design aspects: system modeling, software-hardware partitioning, mixed module simulation and integration. Several researchers have described frameworks for modeling, simulation and generation of mixed types of software-hardware designs, time-discrete or time-continuous specialized hardware. Srivastava and Brodersen [SrBr92], using a library of parameterized hardware modules and existing synthesis tools, map the system description to a hierarchical four-level architecture template in which ASICs only appear at the lowest level of the hierarchy while software runs on either DSP units, off-shelf single board processor or host workstation. Their work describes a manual methodology for hardware and software prototyping. Another approach by Kalavade and Lee [KaLe92] suggests a unified framework for software-hardware codesign by using *Ptolemy*. *Ptolemy* is an environment for simulation and prototyping of heterogeneous systems containing components with different design style or implementation. They also point out some issues related to software-hardware partitioning, such as shifting dedicated heavy-computation function from software into ASICs to let processor have sufficient computation cycles for other computations. However the partitioning process is still manual and *Ptolemy* does not have tools to support partitioning. Falling in the same trend, the work by Buchenrieder and Veith [BuVe92] tries to integrate several available tools in an open architecture through a codesign manager to facilitate system specification, component design, as well as system simulation and integration. Basically, all the work described above focuses on modeling, simulation and generation of software-hardware system in a unified framework by using, extending, or integrating existing tools or systems.

On the other hand, only a few papers exist for software-hardware partitioning problem that is essential in the codesign process. The approach used by Ernst and Henkel [ErHe92] starts with complete software implementation and then extracts out for implementation in hardware only those code segments where timing constraints are violated. It suggests using extensive simulation to identify if and where timing constraints are violated. It points out that the overhead of hardware extraction includes communication overhead, interlocks, as well as compiler effects, and that cost function should consider area overhead and timing improvement caused by additional hardware. Although this paper proposes a system for software-hardware codesign, the criteria and the procedure for the hardware extraction

are not discussed. Gupta and De Micheli [GuDe92] propose a partitioning algorithm that starts with an initial partition where all operations, except for the unbounded delay data-dependence operations, are assigned to hardware. The rest is assigned to software running on a processor. If the partition is not feasible, then the algorithm fails, otherwise the partition is refined by migrating operations from hardware to software in the search for a lower cost feasible partition. This algorithm considers input/output data rates as a constraint. It does not consider another commonly used type of constraint such as timing constraint between operations. The algorithm does not describe the the procedure of determining the feasibility of each partition and the order of moving operations from hardware into software.

The work towards software-hardware codesign is in its initial stages. There are lots of problems which need to be identified and solved, e.g. the interface between software and hardware, the criteria and algorithm for the partition, the techniques for mapping software-hardware description to target architecture etc. To understand the codesign process and necessary tools to support this process, we have studied two contrasting software-hardware codesign methodologies. One approach starts with a cost-effective (software) design and the other starts with a performance-effective (hardware) design. In the first approach, we start with a description which is compiled into machine code for a selected processor. However such a pure software implementation may be too slow to meet the imposed performance constraints. Therefore we extract performance critical parts and implement them as ASICs. In the second approach, a hardware solution is assumed at the beginning. This description is modified and mapped onto a target architecture containing standard processor and one or more ASICs. Non-performance-critical parts are identified in the description and implemented by software running on the standard processor in order to reduce total design cost. The step-wise description refinement process for both approaches and the necessary CAD tools are discussed in this report.

The report is organized as follows. An example of a real-time system is described in section 2. The two contrasting codesign approaches are described in sections 3 and 4. Section 5 describes the tools needed to support such codesign process. The report concludes with section 6.

# 2 A Example Real-time Embedded System

Our codesign approaches will be demonstrated on a real-time medical system used to measure patient's urinary bladder volume [Wu85]. Typical characteristics of the system include real-time control, data acquisition, and complex computation. The system controls a transducer, which is attached to the motors, to scan the related abdominal area along a two-dimensional grids. At each scanning point, the transducer sends a ultrasonic wave directly into the anatomical region to be examined. When the ultrasonic wave strikes tissues of different acoustic impedance, an echo is reflected back to the transducer. Two major peaks will be generated by the echoes from the anterior and posterior walls of the bladder. Therefore the

distance between the anterior and the posterior walls of each section of the bladder can be determined and thus the volume of the bladder can be computed. Fig. 1 shows the volume measuring system and its interfaces with the environment.

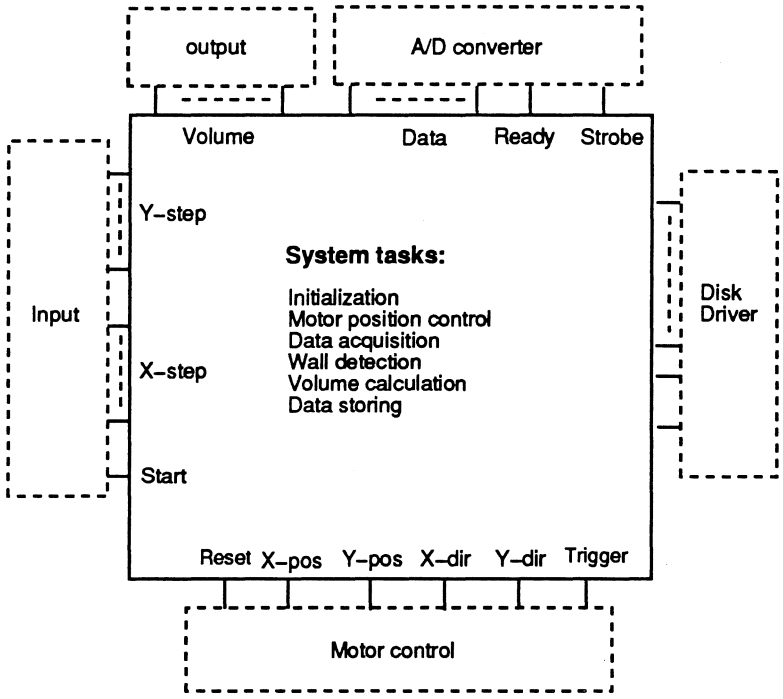


Figure 1: An example real-time system

Basically, the system consists of six tasks: (1) initialization; (2) motor position control; (3) data acquisition; (4) wall detection; (5) volume calculation; (6) data storing. In the initialization stage, the system loads the number of control steps along x direction and y direction and resets the motors to the initial scanning position. When the starting signal is set, the scanning process begins. The motor position controller sends out the control signals that drive the motors to move the transducer to the scanning position. Then the motor controller activates the transducer to send the ultrasonic wave to the examing point. The data acquisition module converts the ultrasonic echo into digital signals and store them temporarily. The wall detection and volume calculation modules determine the distance between the anterior and posterior walls of the bladder and the volume based on the fetched data. Finally the data storing module stores the fetched data into disk for later analysis. The process continues until the system covers the entire given scanning range.

There are two main imposed performance constraints for this system: (1) data acquisition and conversion of 1000 data points is 1 ms; (2) maximum time between two scans, including time for motor control, data acquisition, wall detection, volume calculation and data storing, is two seconds.

# 3 Hardware Extraction Approach

This approach starts with a complete software implementation from which the critical parts are extracted and implemented as ASICs.

## 3.1 The Design Process

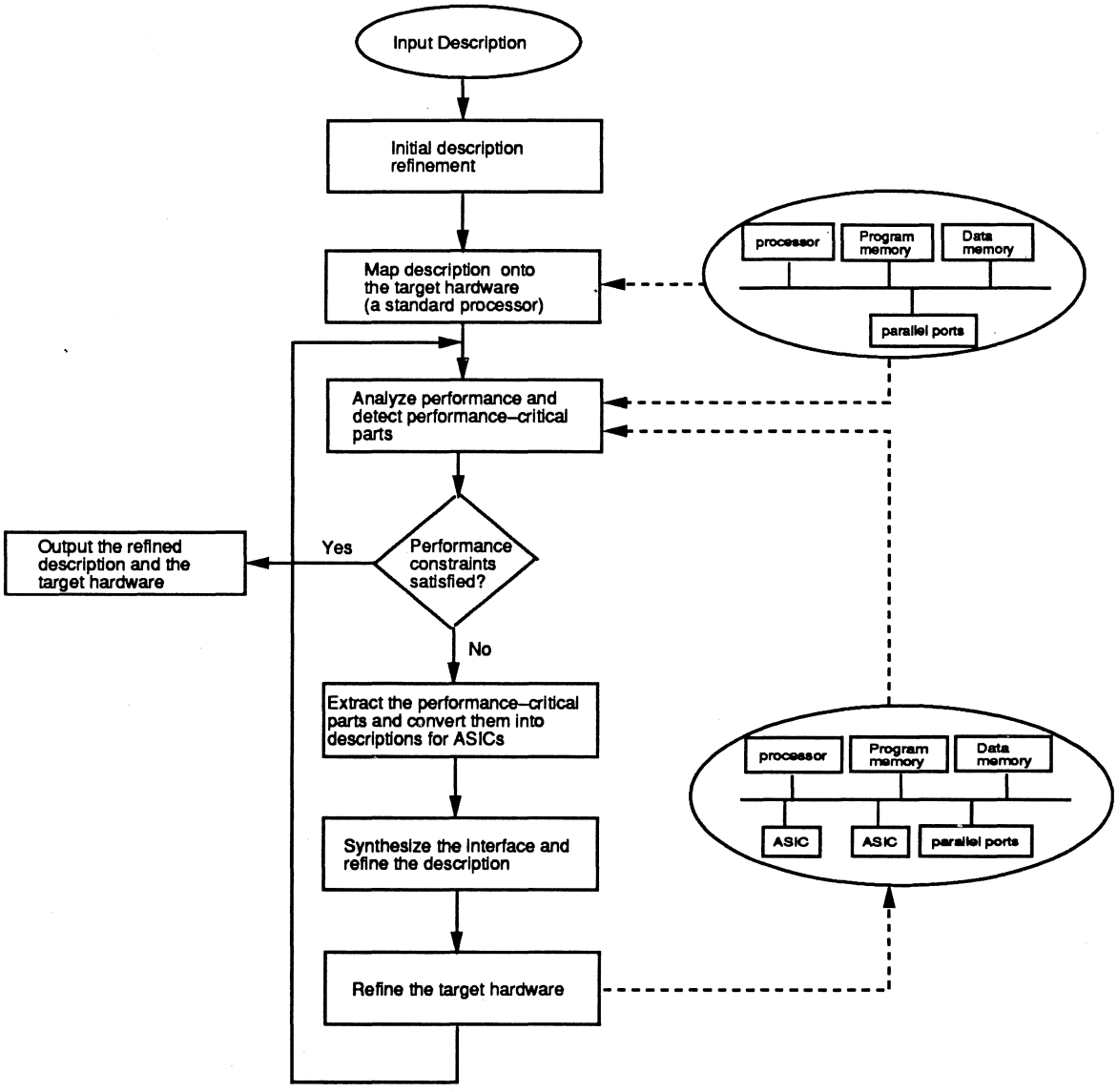


Figure 2: Design process in hardware extraction strategy

The design process, shown in Fig. 2, starts with a VHDL description of the system.

'Initial description refinement' task connects the input description into one process. In the beginning the input description is assumed to be implemented completely by software running on a standard processor. The associated target hardware (shown in the ovals) is a bus-oriented uni-processor architecture in which the memory modules, I/O ports or ASICs are connected to the processor bus. Although the target model can be extended to multi-bus, multi-processor architecture [SrBr91], we believe that most of the real-time applications, including our example, can be accommodated by this uni-bus uni-processor model. Initially no ASICs are connected to the target hardware since the description is implemented completely in software. The mapping of description onto the target hardware can be achieved by compiling the description into the processor instruction set. The designed system communicates with its environment through memory mapped parallel I/O ports.

To see whether the software implementation satisfies the performance constraints or not, performance estimation is carried out and performance-critical parts are detected in this step. Performance-critical parts (PCPs) are referred to those description segments whose estimated performance do not satisfy the imposed constraints. Here control flow graph and inter-basic block data flow graph are built from the description. The nodes are basic blocks, and the edges in the control flow graph represent sequencing while the edges in the data flow graph represent data dependence. The control flow graph is used for finding PCPs. The data flow graph is used to find the potential concurrency among basic blocks. If the performance constraints are satisfied, then the description refinement process ends, otherwise the codesign process proceeds.

After detection, performance-critical parts are extracted and converted into separate ASIC descriptions. The task of detecting and extracting of PCPs is a problem by itself which will be discussed in a subsequent reports. After the extraction, the description is further decomposed into several processes since the extracted PCPs form new processes which are going to be implemented in ASICs. The concurrency identified from the data flow graph may be exploited by inserting PCPs into different processes. We call these new formed processes ASIC processes. The original process, from which the PCPs are extracted, is called the software process. Thus, software process is fragmented and refined to a new software process. All these processes communicate through global signals realized by the processor built-in bus. Protocols used for the communication between software process and ASIC processes are generated and inserted in appropriately. If more than two AISC processes may access global signals simultaneously, necessary arbitration process is generated to resolve the competition. The arbitration process will be implemented as an interface circuit.

Along with the description refinement, the target hardware is also modified by adding ASIC modules to the processor bus. If there is an arbitration process, an arbitration module will be added in the target hardware also. Basically, the codesign system will refine either description or target hardware in order to achieve one-to-one mapping between them. The codesign process ends whenever the performance constraints are satisfied. After the constraints are met, we must (1) generate machine code for the chosen processor from the software process; and (2) synthesize ASICs from the ASIC processes.

## 3.2 Design Steps for the Example

We have chosen the 8086 microprocessor to implement our example. The hardware extraction design process follows the strategy described in Fig. 2.

The initial description is mapped onto its corresponding target hardware as shown in Fig. 3. The resulting machine code (called software) will be stored in the program memory. Variables in the description will be stored in data memory while global signals through which the processor communicates with its environment will be mapped to the parallel IO ports. The processor bus is invisible at this level, where the description is implemented completely by the machine code.

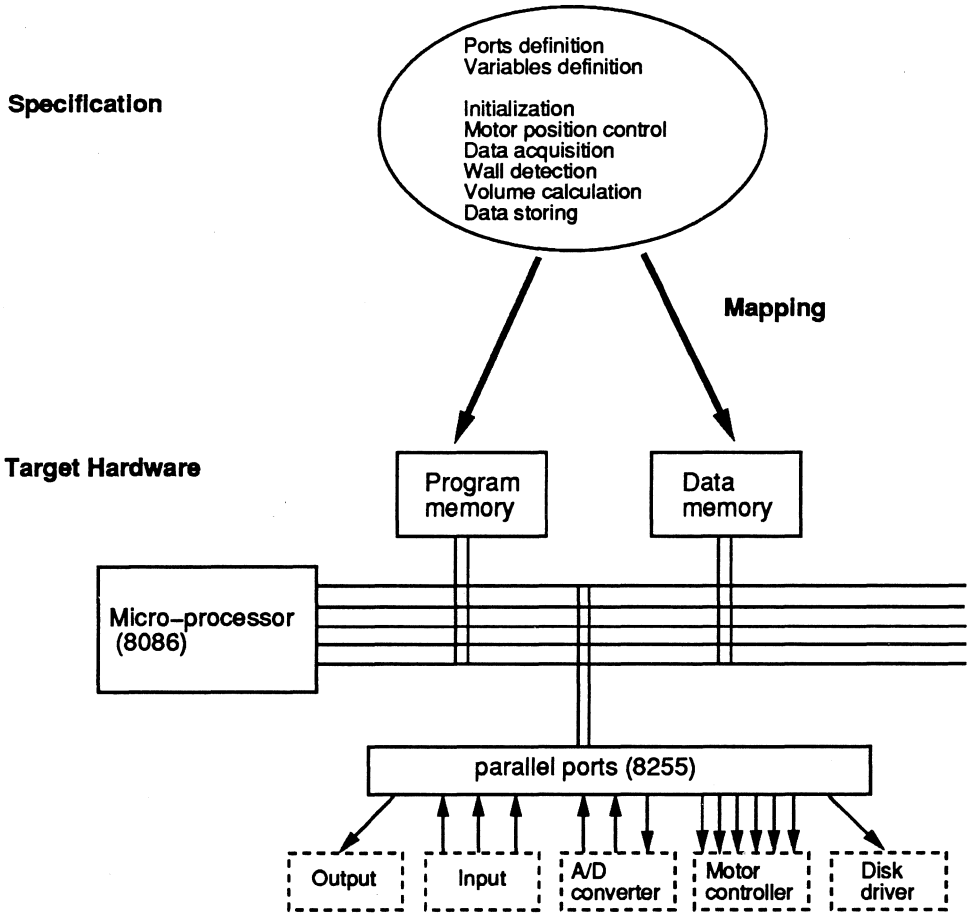


Figure 3: Initial description

Since the processor bus is used for communication with ASICs, it needs to be specified explicitly as global signals. Therefore the original single process description is decomposed into two processes communicating through those global signals supported by processor built-

in bus (Fig. 4). One process (left oval) is the original main process with some of its variables such as arrays allocated to the data memory locations. Variables which can be allocated into processor registers still remain in the main process. The other process is a newly-formed memory process which communicates with the main process using conventional read/write protocols. Thus the memory process will be synthesized into the data memory module while the main process will be compiled into the processor instruction set. Basically, this step refines our description to reflect the connections between standard components.

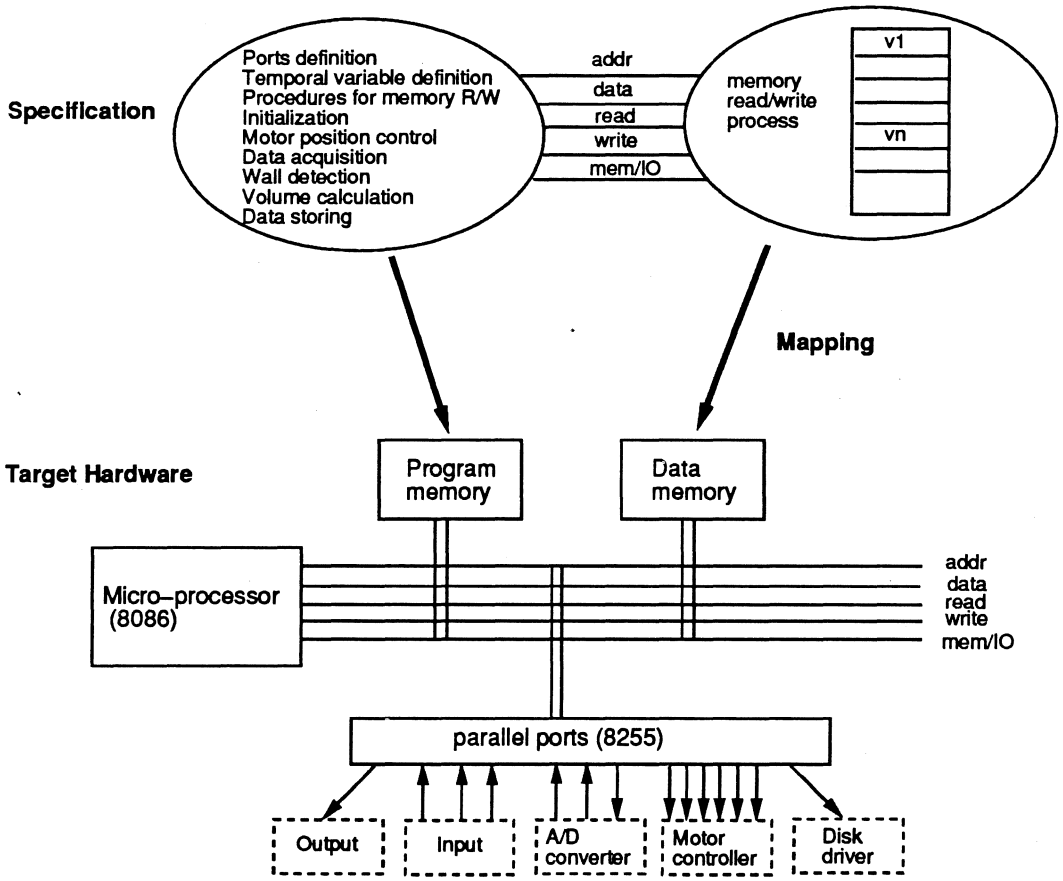


Figure 4: Refined description after memory process is introduced

To see whether the previous implementation satisfies system requirements or not, performance of the system is estimated and performance-critical parts are detected. Fig. 5(a) and (b) demonstrate the control flow graph for the whole description and the data flow graph for the loop body consisting of blocks 1 through 7 in Fig. 5(a). Time constraint 1 and 2 are associated with block 2 and the loop body. First we do software estimation for the 'data acquisition' block. It is compiled into instruction set. The clock cycles needed to execute it can be estimated from the number of instructions and the clock cycles used for each instruction. If the estimated performance does not satisfy constraint 1, then 'data acquisition' is a

PCP and will be moved to hardware.

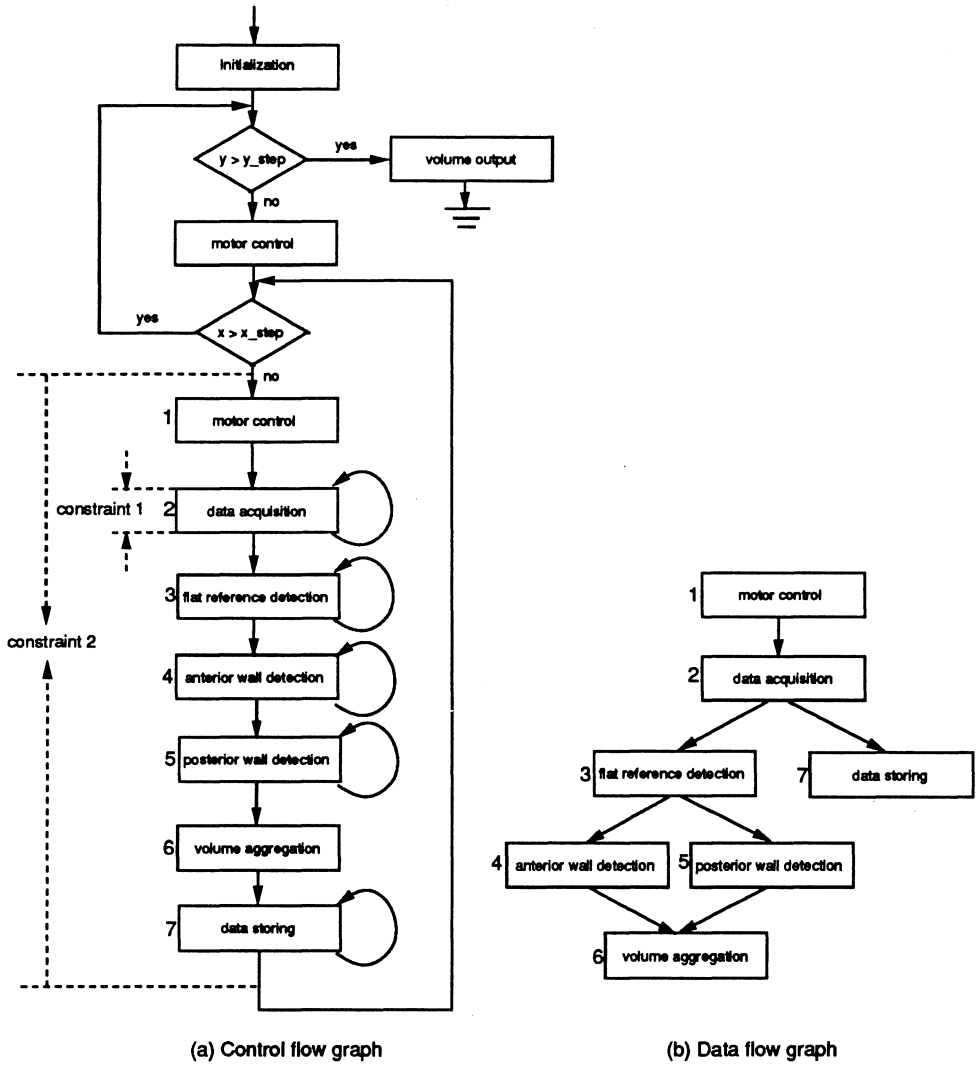


Figure 5: Control and data flow graph for the description

After detection, we extract the PCP 'data acquisition' out to form a new process. Three global signals 'bus_request', 'bus_granted' and 'interrupt', which are supported by the processor bus, are added to the description to facilitate the communication between the main process and the newly-formed data acquisition process. Communication protocols for the bus access are generated and inserted in the main process and the data acquisition process. Besides, conventional read/write protocol is inserted in the data acquisition process. Fig. 6 shows the refined description consisting of three processes and its corresponding target.

After constraint 1 is satisfied, we try to deal with constraint 2 which is associated with the loop body shown in Fig. 5 (a). We do software estimation for 'motor control'(block 1),

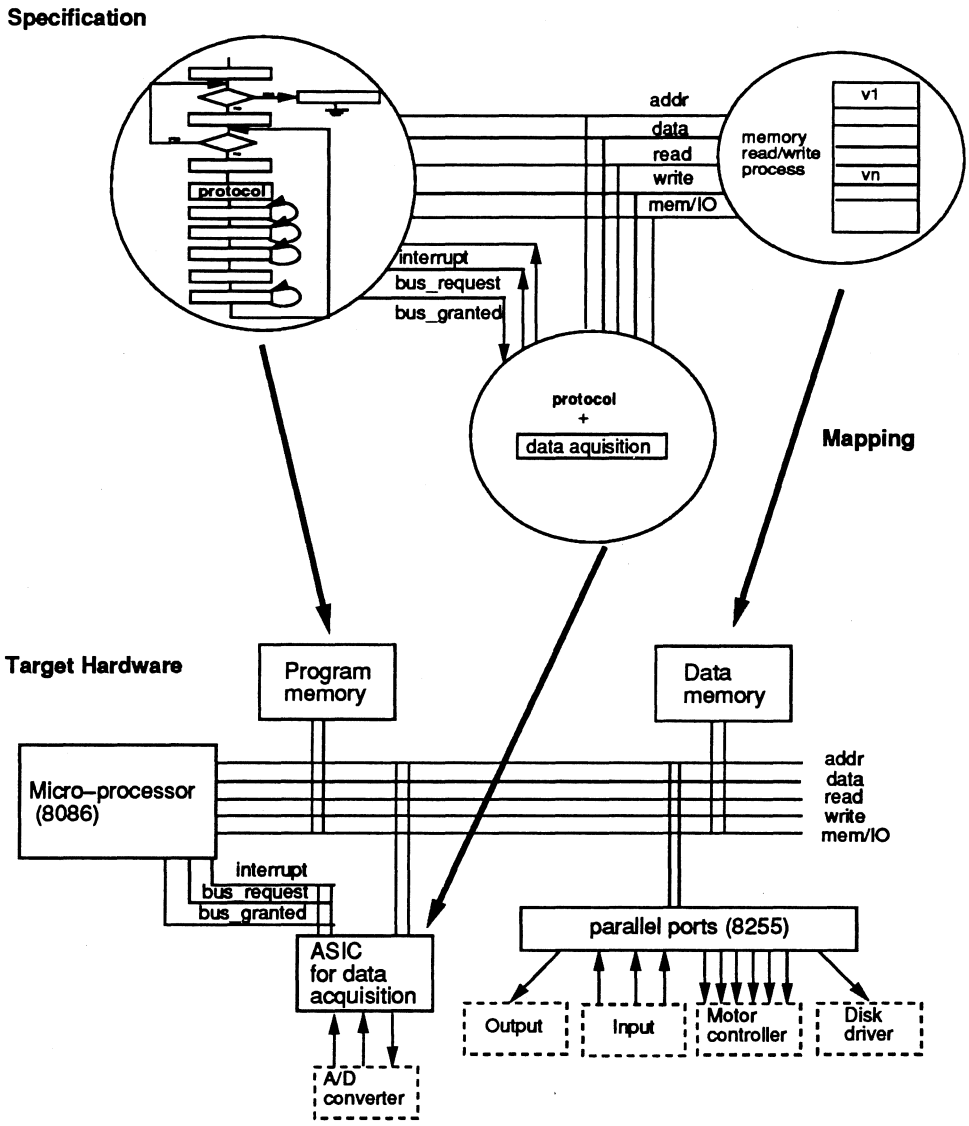


Figure 6: The refined three-process description

'computation'(block 3 to block 6), and 'data storing'(block 7). And we also do hardware estimation for 'data acquisition' and its related protocols. Both software and hardware estimation will be discussed in subsequent report.

If the loop body does not satisfy constraint 2, we must extract some parts or the whole loop body and implement them as ASICs. Among many alternatives, here we describe one way for the extraction. Since 'motor control' includes only several simple output statements, we can leave it in software. Since 'data acquisition' (block 2) is already an ASIC process, we can add the rest of the description (block 3 to block 7) to the same process. However 'computation' and 'data storing' can be executed concurrently shown in the data flow graph in Fig. 5(b). Therefore 'computation' and 'data storing' could be in different processes to preserve the concurrency. Since 'computation' itself is complicated enough, we put 'data storing' in the same process with 'data acquisition' and have 'computation' in a new process.

Fig. 7 demonstrates the further decomposition of the example system. The previous description are refined to five-process description. Those processes communicate through the global signals supported by the processor built-in bus. Those five processes include one software process, one memory process, two ASIC processes and one arbitration process. Protocols for the bus access are generated and inserted in the software process and the two ASIC processes. Besides, conventional read/write protocols are inserted in both ASIC processes. Since the two ASIC processes may request bus at the same time, an arbitration process is generated to resolve bus contention.

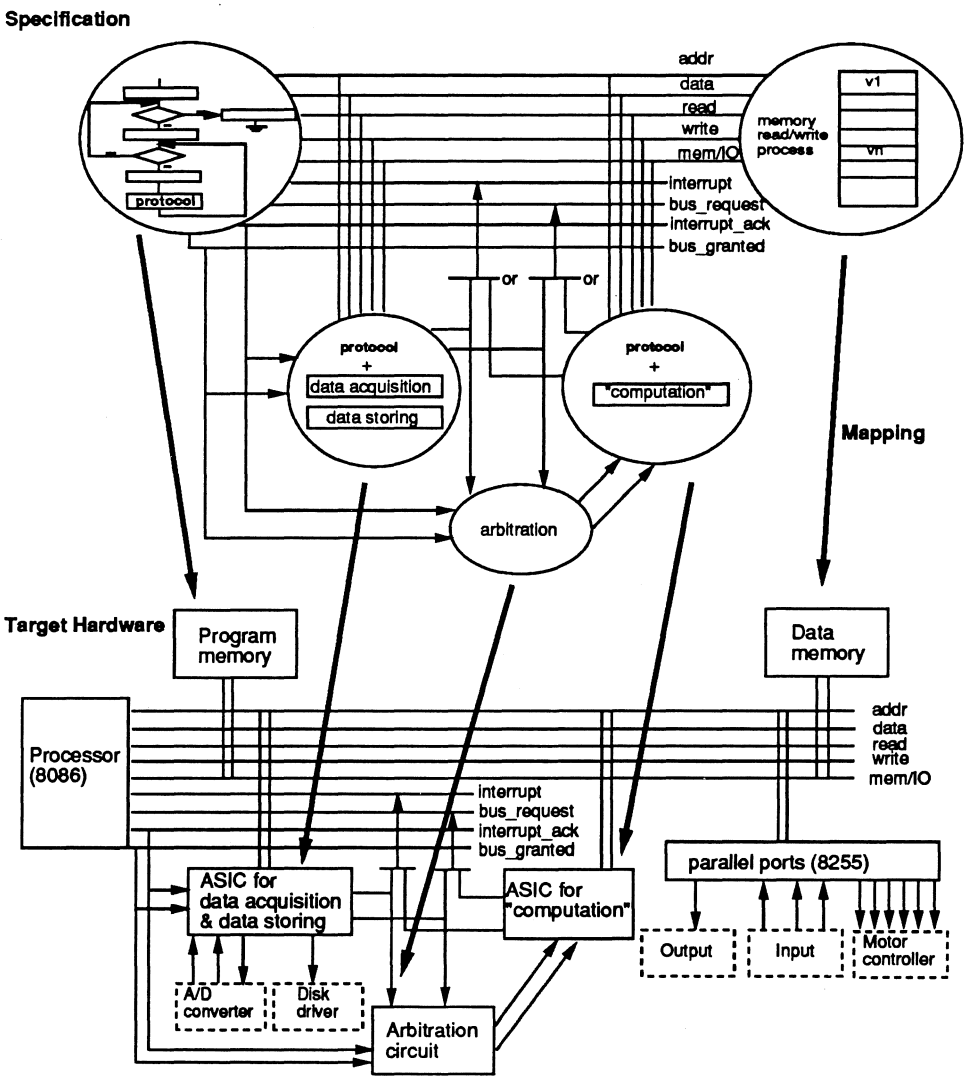


Figure 7: The refined five-process description

# 4 Software Selection Approach

This approach starts with a hardware solution in which all concurrent activities are implemented as ASICs. Some ASICs are merged and others are selected for implementation in software.

## 4.1 The Design Process

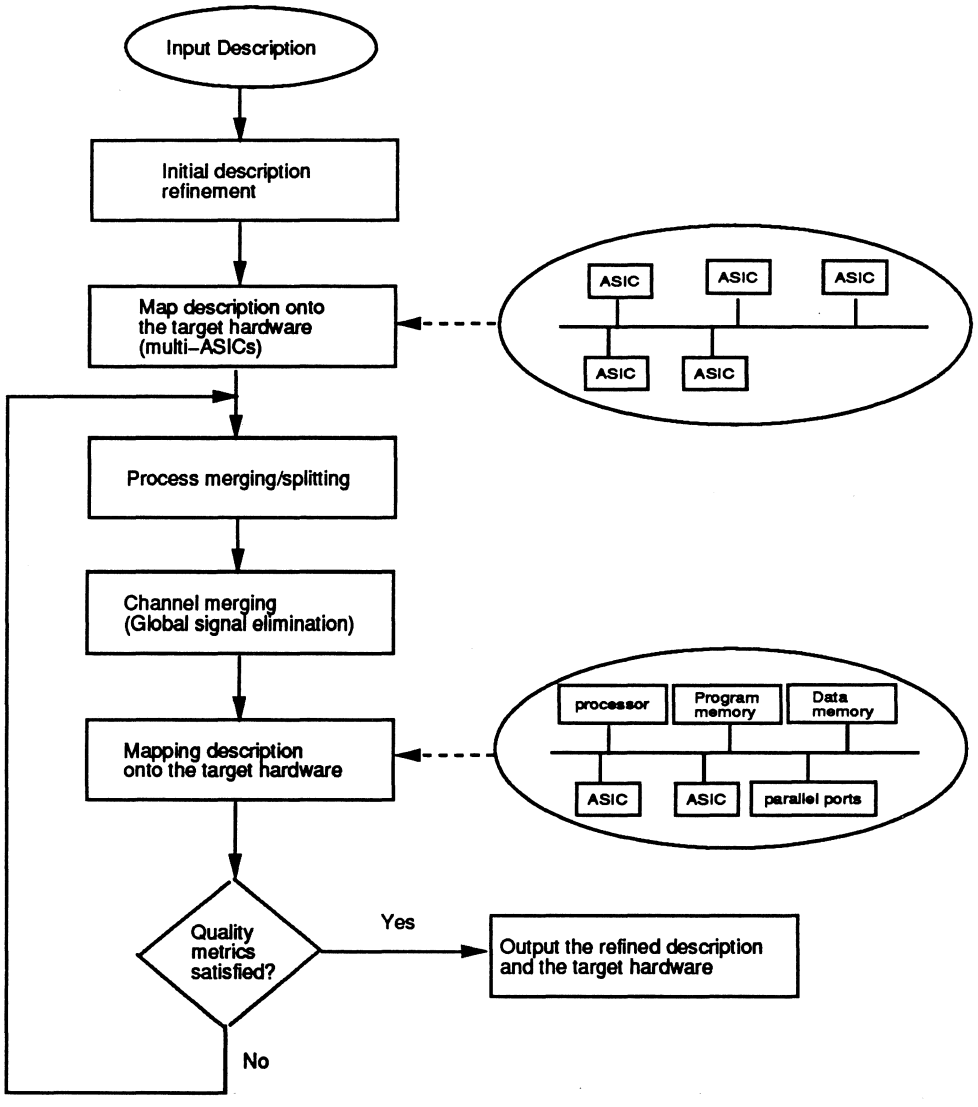


Figure 8: Design process in software selection strategy

The design (Fig. 8) starts with a VHDL multi-process description of the system. These

multiple processes communicate with one another through global signals. This model can be naturally mapped into the target hardware in which each process is realized by an ASIC and the global signals are realized as buses. This hardware implementation however may be too costly since it uses hardware for non-performance-critical parts. It may be also redundant since certain functions may be available in the standard processor selected for software implementation. Thirdly, the design has too many busses since each process communicates directly with other processes. In order to reduce system cost and design time, it is necessary to map the description to an architecture which consists of a standard processor and some inevitably needed ASICs for performance-critical parts. The target architecture we choose is still the single-bus uni-processor architecture used in the previous section.

To map the given description to the target architecture, we need to merge processes. By merging two processes, we can eliminate global signals used for communication between them and convert them into local variables of the newly-merged process. The order in which processes are merged, depends on the data dependences or communication rates among processes. On the other hand we may need to split some process to facilitate mapping to standard components. For example the arbiter for the bus belongs to the memory process in the description. But in the target hardware, the memory process only reads/writes and does not resolve the bus contention. Therefore the arbiter description needs to be moved away from the memory process.

Besides process merging or splitting, channel merging is also needed because processor bus can only accommodate certain number of global signals. Those global signals which can not be realized by the processor bus should be mapped into memory locations accessible through the bus lines. We call this procedure channel merging. Channel merging is considered as one type of global signal elimination. Basically there are two types of global signal elimination. One is through process merging. The other is through channel merging.

After each refinement step, estimation is used to see if the design satisfies the quality metrics such as cost and performance. If not, the codesign process continues. Otherwise the refined description and it corresponding target hardware are generated. From here, the software and hardware can be compiled or synthesized.

## 4.2 Design Steps for the Example

The multi-process description for the example real-time system is shown in Fig. 9. There are six processes communicating with one another through 19 global signals. Memory is accessed by data and address buses. The arbiter in the memory process resolves bus contentions coming from 'data acquisition', 'disk storage' and 'wall detection' processes. Since there is data dependence between 'wall detection' and 'volume aggregation' processes, they can be merged.

Fig. 10 shows the description after the 'wall detection' process and the 'volume aggrega-

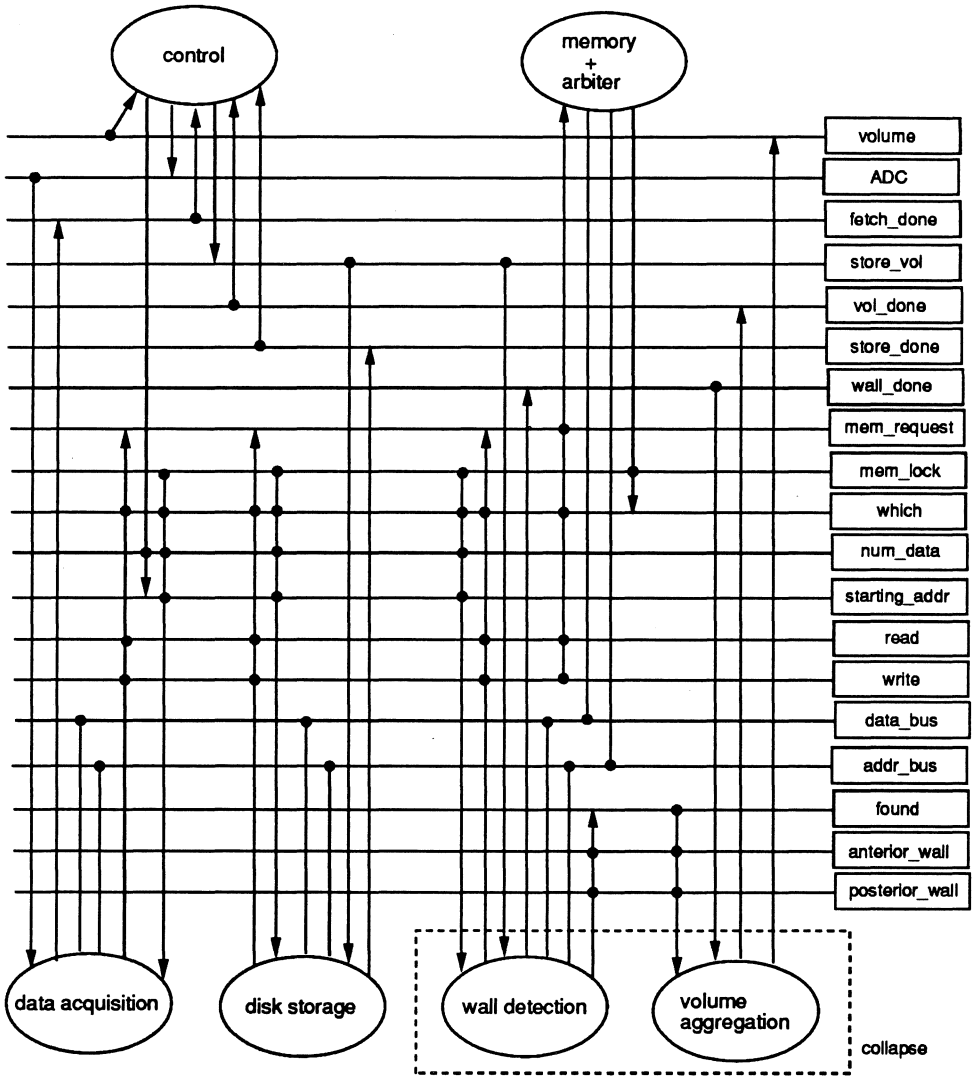


Figure 9: Initial description

tion' process are collapsed together. Previous global signals such as 'found', 'anterior_wall', 'posterior_wall' and 'wall_done' are eliminated and mapped to the local variables of the new 'computation' process.

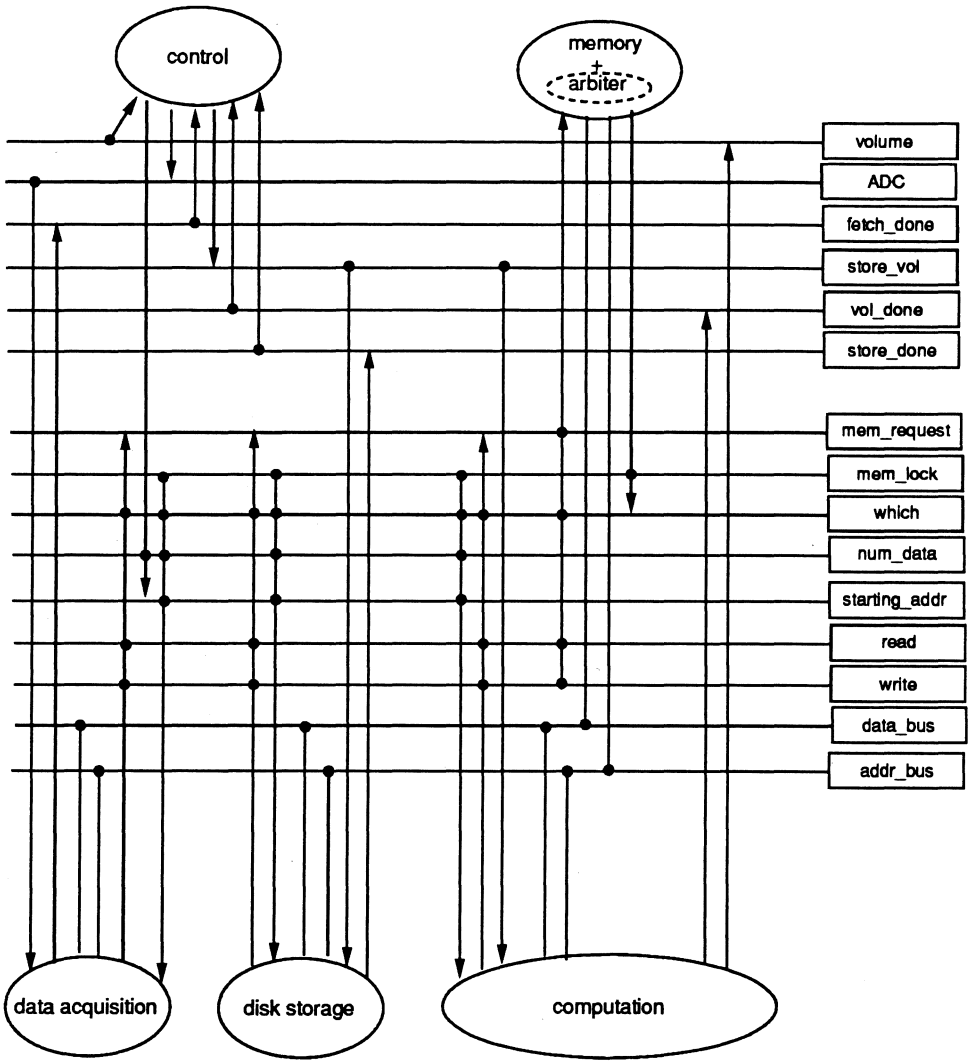


Figure 10: Refined description after collapsing 'wall detection' and 'volume aggregation'

To make mapping to real hardware possible, the arbiter description is removed from the 'memory' process and merged with the 'control' process (Fig. 11). Notice that a new global sigal, 'mem_IO', must be introduced for the 'control' to distinguish memory from other components.

Global signals are also eliminated by channel merging. Fig. 12 shows the refined description after some global signals (shown in rectangle boxes) are mapped into the locations in

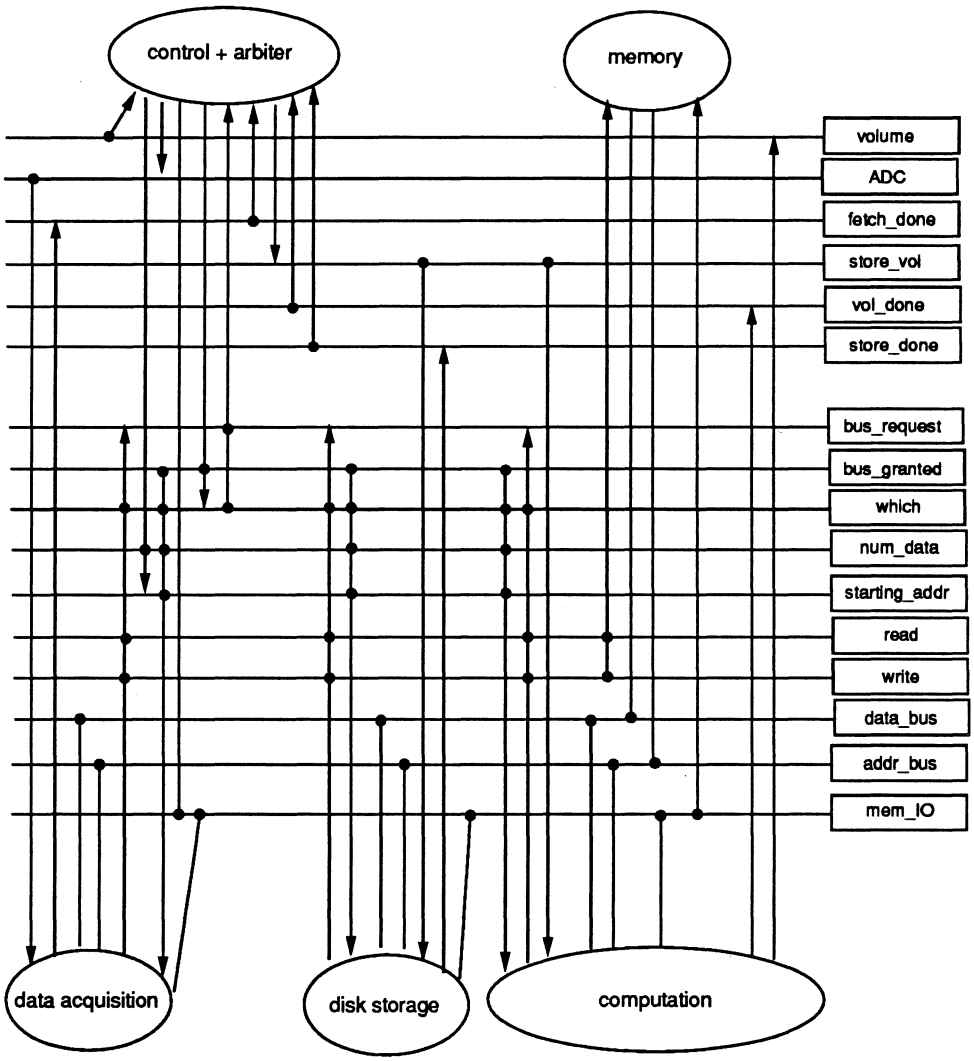


Figure 11: Refined description after migration of 'arbiter'

the memory addressing space. The processes in the description communicate through the remaining global signals.

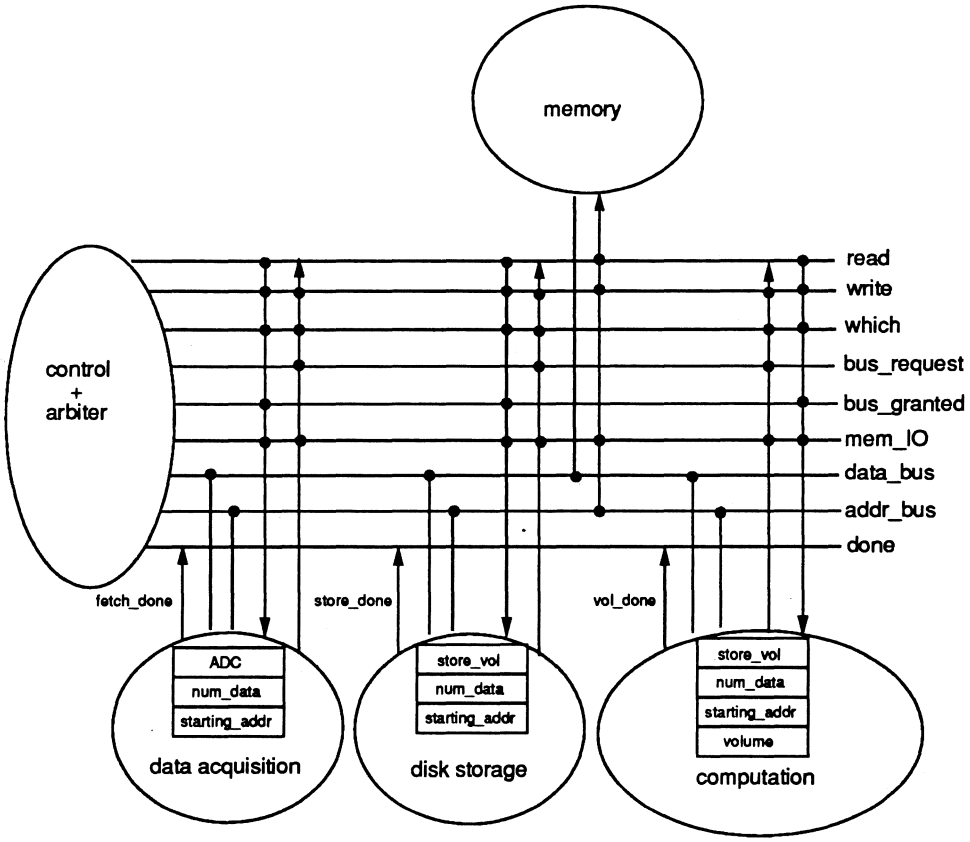


Figure 12: Refined description after channel merging

Fig. 13 shows the final description in which the arbiter is moved out from the 'control' process and described separately. Global signal 'done' is mapped to bus line 'interrupt' and global signal 'which' is mapped to 'interrupt_ack' to accommodate daisy chain in the arbitration process. At the same time 'data acquisition' and 'disk storage' are merged into one process because of the data dependence between them.

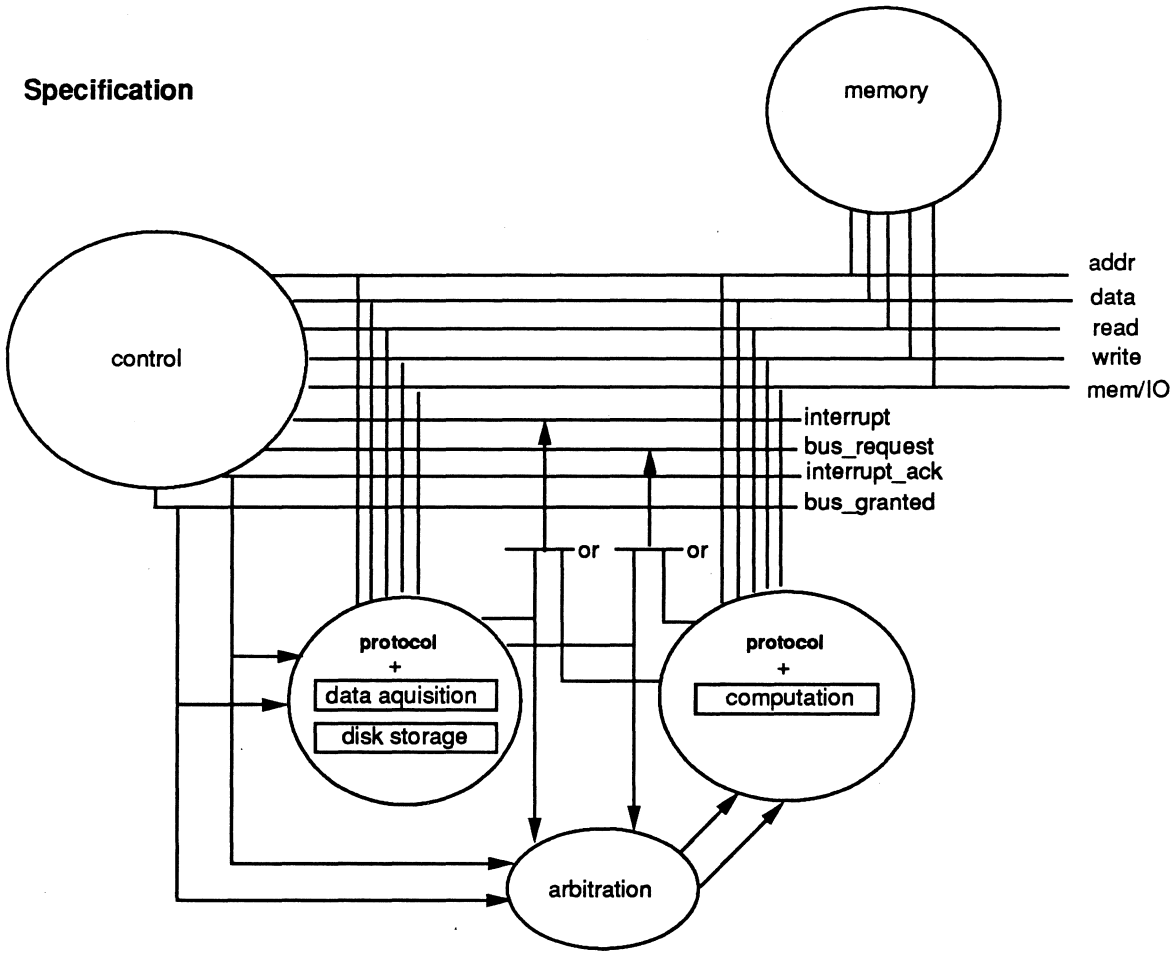


Figure 13: Refined description after moving arbiter out

# 5 System for Software-Hardware Codesign

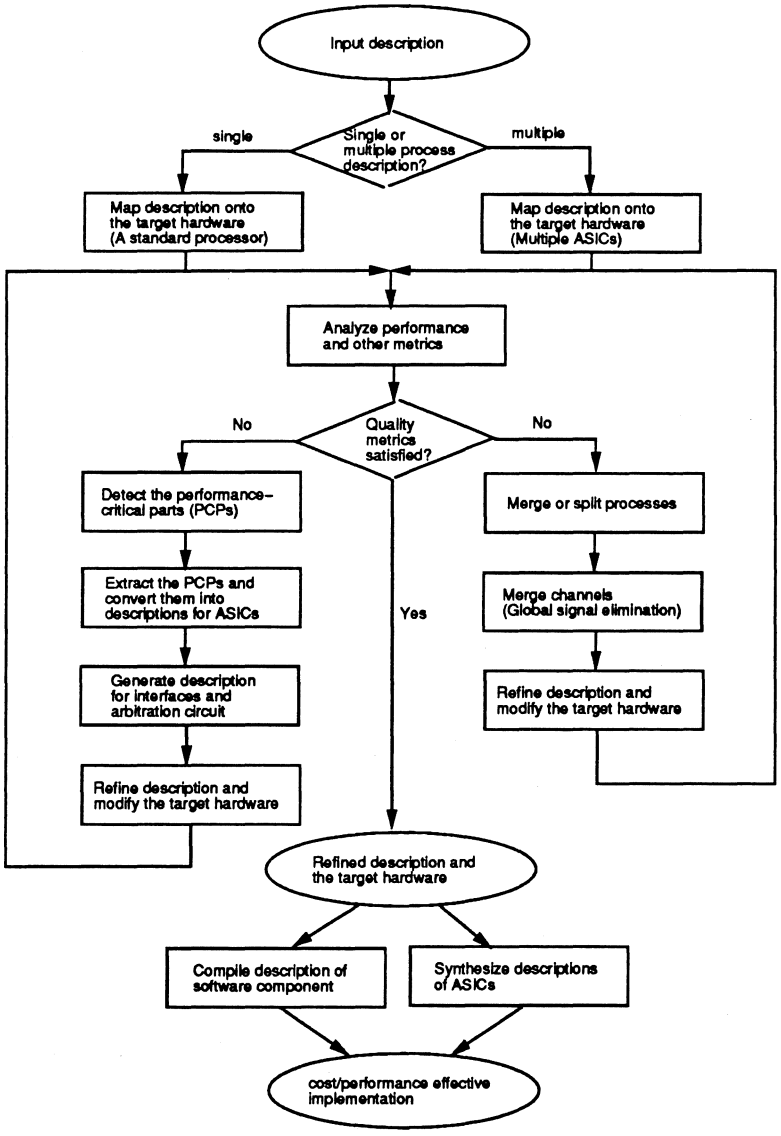


Figure 14: The system for software-hardware codesign

The block diagram of the system supporting the software-hardware codesign process is shown in Fig. 14. The following tools are needed to support it.

1. *Control and data flow graph generator*: It takes a single-process behavioral description as input and generates a control and data flow graphs.

2. *Quality-metrics estimator*: It does software and hardware performance estimations from the input description. Software estimates are obtained by compilation into in-

struction set. Hardware estimates are obtained by scheduling for the target architecture.

3. *Performance-critical parts(PCPs) detector*: It finds the parts where the estimated performances do not meet the imposed performance constraints.

4. *Performance-critical parts extractor*: It extracts the PCPs and converts them into ASIC descriptions.

5. *Interface and arbitration generator*: It generates the interface (protocol) description for the communicating processes. If needed, it also generates the arbitration scheme for resolving contentions for the shared buses.

6. *Description and hardware refinement*: It supports merging or decomposing processes in the description and correspondingly modifies the target hardware.

7. *Global signal elimination*: It maps global signals to local variable of new processes or maps global signals to memory locations.

8. *Software compiler*: It generates machine code for selected processor from the given description.

9. *Hardware synthesis*: It produces ASICs from the given description.

# 6 Conclusion

To understand the software-hardware codesign process and identify tools needed for automation of such a process, we have studied two contrasting approaches for software-hardware codesign. One starts with single process description which can be naturally mapped to a software implementation supported by a processor. The other approach starts with description of multiple processes communicating through global signals, which can be naturally mapped to a hardware implementation consisting of ASICs. The step-wise description refinement process for both approaches has been demonstrated through an example of real-time system.

# 7 Acknowledgements

We are grateful for the support from the Semiconductor Research Corporation (grant #92-DJ-146).

# 8 References


[BuVe92] K. Buchenrieder, C. Veith, " CODES: A practical concurrent design environment", the International Workshop on Hardware/Software Codesign, Sept. 1992, Estes Park, Colorado, USA.

[ErHe92] R. Ernst, J. Henkel, "Hardware-software codesign of embedded controllers based on hardware extraction", the International Workshop on Hardware/Software Codesign, Sept. 1992, Estes Park, Colorado, USA.

[GuDe92] R.K. Gupta, G. De Micheli, "System-level synthesis using re-programmable components", EDAC'92, Sept. 1992.

[GuDe92] R.K. Gupta, G. De Micheli, "Program implementation schemes for hardware-software systems", the International Workshop on Hardware/Software Codesign, Sept. 1992, Estes Park, Colorado, USA.

[KaLe92] A. Kalavade, E.A. Lee, " Hardware/software co-design using ptolemy – a case study", the International Workshop on Hardware/Software Codesign, Sept. 1992, Estes Park, Colorado, USA.

[SrBr91] M.B. Srivastava, R. W. Brodersen, "Rapid-prototyping of hardware and software in a unified framework", ICCAD, Nov. 1991.

[Wu85] A. Wu, "A Microprocessor–based ultrasonic system for measuring bladder volumes", Master Thesis in Electrical and Computer Engineering at University of Arizona, Tucson, 1985.