Design and Implementation of a Robot Control System Using a Unified Hardware-Software Rapid-Prototyping Framework

Mani B. Srivastava, Trevor I. Blumenau, and Robert W. Brodersen
EECS Department, University of California at Berkeley

Abstract

The increasing complexity of applications and the relative maturity of CAD tools for ASIC design have made design aids for the higher-level aspects of system design essential. We have developed a unified framework for the rapid-prototyping of hardware and software for application-specific systems. This framework is fully described in [1]. The application of the framework to the development of a real-time multi-sensory robot control system is described in this paper.

1.0 Introduction

The integration of heterogenous hardware, software, and electromechanical components to form a complete system presents a challenging design problem. Computer-aided design techniques are quite primitive at the system level when compared to those available at the chip level. Previous research in CAD has primarily concentrated on the design of individual application-specific ICs (ASICs) and has avoided taking a *systems* perspective of the design process.

Motivated by this we have been developing a CAD framework for the rapid-prototyping of application-specific systems that provides support for the design of not just chips, but boards and software as well. Details about the CAD framework can be found elsewhere [1].

In this paper we present the design of a real-time multi-sensory robot control system using our system rapid-prototyping framework. The key features of the computer-aided system design methodology offered by our framework are exemplified through this system. The system controls, in real-time, a six-degree of freedom articulated robot arm using position, force, and proximity sensing.

In addition to the novel system design methodology, another key aspect of the robot control system is the extensive use of special-purpose dedicated hardware, made feasible by our system-level rapid-prototyping framework. This provides much improved performance over commercial and other research robot control systems that are largely based around general-purpose computers.

2.0 Framework for rapid-prototyping of hardware and software

We have implemented a design framework [1] for dedicated board-level systems which uses a unified view for the hardware and software components that make up a system. As shown in Figure 1, and elaborated in the following subsections, a high-level specification of the system is implemented as a set of custom printed-circuit boards using a mix of software programmable and non-programmable hardware and ASICs, going through steps of architecture mapping, board-level hardware module and system software module generation.

2.1 System specification

The target system is viewed by the user as a parameterized, static, hierarchical network of sequential processes that operate concurrently, and interact using a well defined communication mechanism. Systems of interest to us, such as a robot controller or a speech recognizer, possess a large granularity parallelism, and are naturally expressed as a static set of processes communicating via message queues. This is also evident from the fact that the system software for such systems ends up looking like a custom distributed real-time OS. Communication is done using FIFO channels that connect input and output ports on the processes.

2.2 Architecture mapping

In architecture mapping each process in the original description is mapped either to a *software process* running on a programmable processor module, or to a *hardware process* running on a dedicated custom hardware module. A template mapping based approach is used in which the process network is partitioned to a parameterized architecture template. The template parameters and the partitioning are manually specified, although the h/w-s/w partitioning technique presented in [2] may be adaptable to automate part of this process.

The architecture template, shown in Figure 2, has a layered, distributed architecture with a hierarchical bus organization for increased bandwidth. There are four layers in the architecture. The bottom two layers of the hierarchy are spanned by custom boards. Each custom board has one or more programmable processor modules running a real-time OS kernel, and coordinating a number of application-specific slave modules with standard hardware and software interfaces. These slave modules form the lowest layer of the hierarchy, and can be either dedicated custom hardware modules or dedicated software programmable modules. The custom boards reside on a back-plane bus, and are slaves to another processor module running a real-time OS which forms the second layer of the hierarchy. The processing modules on the custom boards communicate and synchronize with the master layer 2 processing module using a standard hardware interface and software protocol. The topmost layer of the hierarchy is formed by a UNIX workstation that communicates with the layer 2 processor over a network using standard protocols. Computation power, the communication bandwidth, and the ability to

1063-6404/92 $3.00 © 1992 IEEE

meet real-time constraints increase as one goes down this layered architecture template.

2.3 Board-level hardware modules

The framework provides several board-level module generation aids to the user. We use the same database and design management tools as we use for ASICs, so that the various pre-existing ASIC generation tools are naturally available as special case board-level module generators. This includes a silicon assembler (LAGER [3]), a silicon compiler (C-to-Silicon [4]), and a behavioral-synthesis system (HYPER [5]).

A key component of our board-level hardware module generation strategy is a very extensive library of reusable parameterized board level sub-system modules. This includes a variety of complete processor modules, memory subsystems, data acquisition subsystems, bus interface modules, fiber optic communication modules, testing modules etc. For example, a TMS320C30 processor module in the library provides a complete microcomputer based around a powerful DSP whose configuration in terms of memory organization, I/O devices, and host interface is parameterized.

This library enables sub-system level reusability, and is essential in supporting the architecture template strategy described earlier. The system architecture template is essentially a parameterized network of library modules.

To facilitate the design of board level modules, a structural description language with lisp-based parameterization facilities, tools to do board level automatic/interactive/user-specified/tiling-based placement and routing, and a tool to map a behavioral description to a netlist of programmable devices, such as FPGAs, are also provided. An interface synthesis system ALOHA [6] has been integrated to provide the ability to synthesize interfaces between various modules such as a cpu-memory interface or a VME bus interface.

2.4 System software modules

Many processes in the original process network description of the system are implemented as software processes. Several of these software processes can map to a processor module.

To allow a uniform mapping of these software processes to the various programmable processor modules in the library, we have standardized on a set of efficient, real-time multi-tasking kernels for the processor modules. All software blocks in the system description are mapped as processes on one of these real-time kernels which also provide libraries implementing the channel based communication protocol. Each hardware process also has a wrapper software process running on the programmable processor to which it is attached. This results in a uniform interface to each process, whether it be hardware or software, and facilitates easy migration of functions between hardware and software.

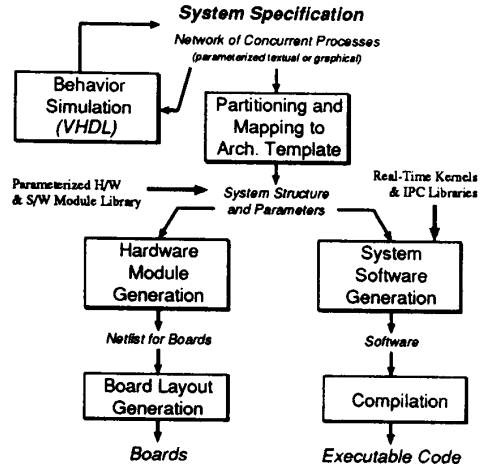

FIGURE 1: Overview of the CAD Framework for System-Level Design.

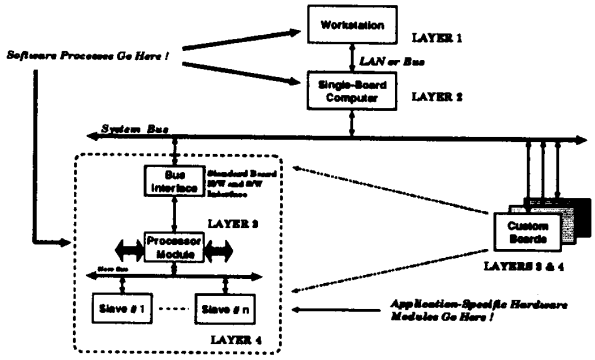

FIGURE 2: Layered Architecture Template.

Special software processes are run on the various processor modules to provide a consistent run-time environment for user-interface, file I/O, program loading etc. This is a trivial by-product of our use of multi-tasking kernels.

At present the framework supports processor modules based around TMS320C30, MC96002, DSP32C, MC68020 and SPARC, together with corresponding kernels. For example, we use SPOX real-time kernel for the TMS320C30 modules, and VDI [7], a simple home-grown kernel, for the DSP32C module.

3.0 Application to robot control system

In sharp contrast to architectures based around a general-purpose homogeneous shared-memory MIMD multi-processor with off-shelf I/O boards that are typical of most state-of-the-art robot control systems [8][9], our custom architecture approach offers much improved performance in a more compact package. The tasks in a robot control system not only have high real-time computation requirements, but also need extensive specialized I/O capabilities. This restricts the controllers based around general purpose machines with limited I/O capabilities to simple control mechanisms, or to non-real-time algorithm test benches at best.

Robotic control algorithms have become far more complicated than the simple PID joint controllers of the past. State of the art systems incorporate both force and position controllers (sometimes both running at the same time [10]). Position control is generally used when the robot does not touch anything. However, for tasks in which the robot touches an object, force control is almost a necessity [11]. For example, the robotic task of scraping paint from a surface cannot be easily performed using a position controller - any error in the system can be catastrophic. The ability to specify as a control input the desired force against the surface solves the problem.

We use a form of force control called impedance control [12], in which the force applied by the robot end-effector is proportional to the displacement from its goal position. This system is similar to a spring,

where F = K•ΔX. This type of controller is well suited to tasks such as peg-in-hole insertions using a remote center of compliance as proposed by [13]. Figure 3 shows an extremely simplified process network view of the system.

Linearization of the robot control algorithm is based on the calculation of complicated inertial, Coriolis, and gravitational terms, and these terms have to be updated at a reasonably high frequency. There are also numerous frame transformations and unit conversions that must be performed inside the control loop. At a higher level, a trajectory control process continuously monitors the proximity sensor, and updates the controller inputs. Certainly, the hardware architecture must be designed to facilitate all these high bandwidth tasks.

3.1 Architecture of the system

The resulting architecture of the robot control system is shown in Figure 4. It is an instantiation of the layered architecture template described in section 2.2, and contains two custom robotic boards. The first is a controller board which in turn communicates with a custom peripheral board using a fiber-optic link. This peripheral board interfaces with the joint motors and current sensors in the robot arm. The controller board also communicates with force, position and other sensors.

3.1.1 Robot controller board

This is a custom board to which the processes related to the control law, position, velocity, force, and current sensing, and motor drive are mapped. It spans layers 3 and 4 of the architecture template discussed in section 2.2. There are two processor modules based around the 33 MHz TMS320C30, a powerful floating-point signal processor. The two TMS processing modules are distinguished by the kind of slave modules that they have. These slave modules reflect the specific requirements of the robot control tasks.

TMS processor module #1 has three slaves. First is a powerful programmable processor module based around a 50 MHz DSP32C which is dedicated to the trajectory calculation process. The second slave is a fiber-optic based communication link to connect the robot peripheral board at the other end, and serves as an interface to the robot motors. It is used to apply specific voltages or torques to the motor, to sense the motor currents, and to apply brakes to the robot. The ASIC that implements the protocol processing employs an asynchronous design methodology, and

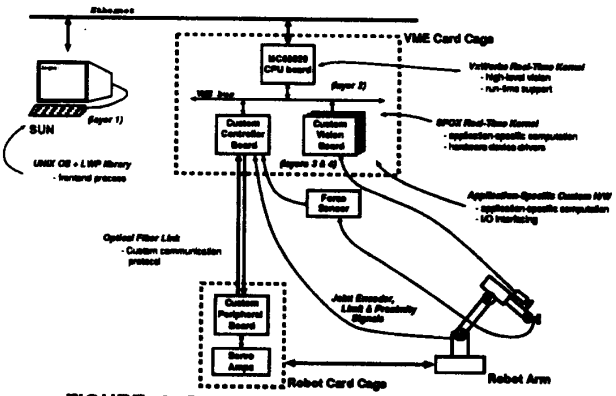

FIGURE 4: Implementation of the Robot Control System.

was synthesized using the interface synthesis tools. The third slave attached to TMS processor module #1 is a position-velocity sensing module. The ASICs for this module were automatically generated from a parameterized structural description.

The second processor module, TMS processor module #2, has three slaves. The first slave is a DSP32C based processor module, identical to the one attached to TMS processor module #1. This, however, is dedicated to the process calculating the Forward Kinematics and the Jacobian of the robot arm. The second slave is the force sensor module that interacts with a strain-gauge based force-torque sensing wrist. The third module is the position-velocity sensing module, which is in fact shared with the TMS processor module #1.

The board was fabricated as a 9U VME slave board, and is fully functional. Primarily as a result of the module-generators available, and the sub-system level library, this complex 500+ component 12" x 14" board had a design cycle of less than two months. Further, since it follows the architecture template, the system software was configured for it in very little time.

3.1.2 Robot peripheral board

This board is really part of the robot in that it provides interface to the robot joint motors and brakes. Its task is to receive voltage or current values from the controller board and apply them to the robot motors, to

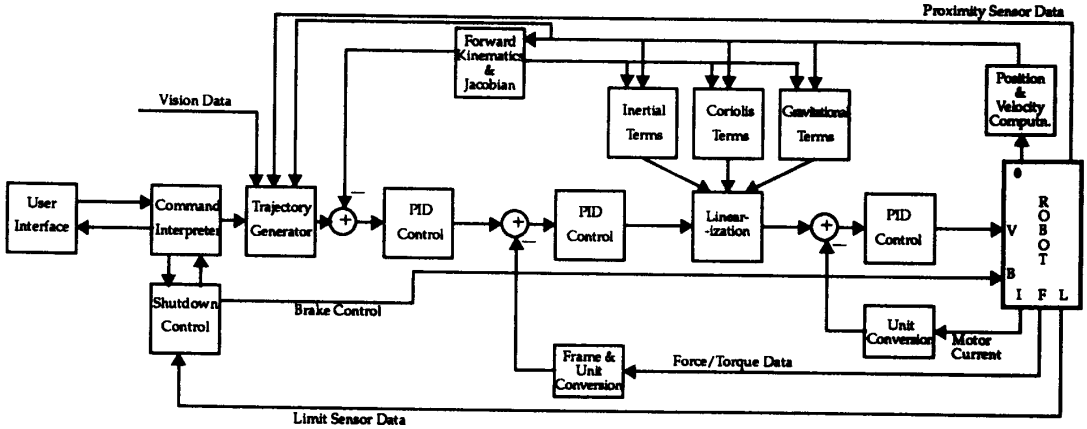

FIGURE 3: Simplified View of the Robot Control System

sense the motor current, and to apply the brakes in case of stalls or when commanded.

The board uses A/D, D/A, and optical communication modules from the sub-system module library. The protocol processors are synthesized using the ALOHA tool, while the digital pulse-width modulator is implemented with the FPGA module-generator. Only a small analog portion of the board (op-amp based filters) had to be custom designed for this board - the rest was either automatically generated or instantiated from the reusable parameterized subsystem library. As a result of this level of automation the entire development cycle from input description to the working board was again less than two months.

3.2 Organization of software processes

Due to the inherent large-grained parallelism present in the robot control system, it is easily decomposed into a network of processes. Some of these processes are *hardware processes* in that they run on their own dedicated hardware. Others however are implemented as *software processes*, and are mapped to one of the several software-programmable processors available in the top three layers of the architecture, where they run under the control of a kernel.

The SUN workstation in the top layer runs a user-interface process that provides an interactive environment based on top of a C interpreter. A library of robot specific C routines provides an interface to the lower-levels of the control system, and allow the user to switch between impedance control and position control modes, calibrate the robot, read sensor data, update the goal state of the robot (position, force, and gripper status), specify motion trajectory, etc.

The MC68020 based processor in the second layer is responsible for calculating position values from the data obtained from the vision board, as well as acting as a gateway between the SUN and the custom boards.

In the third layer, the TMS processor module #1 has the following software processes mapped to it:

- server process that interprets commands from the user-interface process on layer 1.
- process for computing the desired end-effector force to apply, by comparing the tool frame of the robot arm with the goal frame ($F = K \cdot \Delta X$).
- process adjusting the amount of voltage applied to the robot motors using the current sensor data obtained from the robot peripheral board.

The DSP32C slave module attached to this processor runs the trajectory calculation process.

The TMS processor module #2 has the following processes mapped to it:

- process to find the current tool-frame through forward kinematics.
- process to compute the error in applied force/torque using the data from the sensor.
- process to compute the desired output joint torque, the inertial, Coriolis, and gravitational terms, as well as the Jacobian transform.

The DSP32C slave module attached to this processor runs the Forward Kinematics and the Jacobian calculation process.

In addition to the user processes mentioned above, all the processing modules also run system processes in the background to provide run-time I/O services, and to handle data-routing.

4.0 Conclusions

We presented a case study of the design of a robot control system that employs extensive use of custom hardware at both chip and board levels to do simultaneous force and position control of a robot arm using data from a variety of sensors. In the design of a system of this complexity it is necessary that the chips, boards and software be *co-designed*, and this was made possible by our CAD framework. The heterogeneity inherent in such systems – both in terms of computation model and physical implementation – requires that a system-level CAD environment be much more than a mere scaling up of a chip level behavioral synthesis system to do hardware-software partitioning. Further, the flexibility and rapid-turnaround time offered by the CAD framework makes custom hardware attractive at all levels of the system. This is amply demonstrated when the architecture of our robot control system is contrasted with conventional controllers that have lower performance while using large general-purpose multi-processors.

Acknowledgments

This project was funded by DARPA. The second author was also partially funded by a NSF Graduate Fellowship.

References

[1] M. B. Srivastava, and R. W. Brodersen. *Rapid-Prototyping of Hardware and Software in a Unified Framework*. Proceedings of ICCAD, November 1991.

[2] R. K. Gupta and G. De Micheli. *System-level Synthesis using Re-programmable Components*. EDAC, March 1992.

[3] C. S. Shung, *et al. An Integrated CAD System for Algorithm-Specific IC Design*. IEEE Transactions on CAD of ICs and Systems, April 1991.

[4] L. Thon and R. W. Brodersen. *From C to Silicon*. Custom Integrated Circuit Conference, May 1992.

[5] C. M. Chu, M. Potkonjak, M. Thaler, and J. Rabaey. *HYPER: An Interactive Synthesis Environment for Real-Time Applications*. Proceedings of ICCD, October 1989.

[6] J. S. Sun and R. W. Brodersen. *System Module Interface Design in SIERA*. Under preparation.

[7] Manish Arya. *A Standard Software Platform for Shared Memory Multiprocessor Signal Processing Systems*. M. S. Report, EECS Department, U. C. Berkeley.

[8] J. B. Chen, *et. al. NYMPH: A Multiprocessor for Manipulator Applications*. IEEE International Conference on Robotics and Automation, April 1986.

[9] S. Narasimhan, D. Siegel, and J. M. Hollerbach. *Condor: A Revised Architecture for Controlling the Utah-MIT Hand*. IEEE International Conference on Robotics and Automation, April 1988.

[10] J. Craig and M. Raibert. *A Systematic Method for Hybrid Position/Force Control of a Manipulator*. IEEE Computer Software Application Conference, November 1979.

[11] D. E. Whitney. *Historical Perspective and State of the Art in Robot Force Control*. International Journal of Robotics Research, 6(1).

[12] N. Hogan. *Impedance Control: An Approach to Manipulation: Parts I, II & III*. ASME Journal of Dynamic Systems, Measurement and Control, vol 107:1-24.

[13] D. E. Whitney. *Quasi-Static Assembly of Compliant Supported Rigid Parts*. ASME Journal of Dynamic Systems, Measurement and Control, vol 104:65-77, 1982.