E.E.C.S.

Efficient Elevator Control System

by

Kenneth Barr

K. Trisha Chang

Jennifer Hon

Brian Raf

Sabrina Shukri

Abstract

E.E.C.S. provides an efficient means to gather information from elevators and floors (hall calls), process this information, and use the results to choose the appropriate destination for each elevator. State-of-the-art elevator controllers churn through databases of time-stamped button-presses and rely on complex queueing algorithms, and while the Efficient Elevator Control System (E.E.C.S.) cannot attain this level of sophistication in its limited development cycle, the E.E.C.S. microcontroller contains several useful features that are not present in today's elevator control systems.

For purposes of this project, the controller is designed to handle 16 elevators and 64 floors. However, the concept is scalable since the elevator dispatching algorithms do not depend on the specific number of floors or elevators. By implementing a 16-bit RISC processor enhanced with interrupt capability, data storage, and specialized instructions, our microcontroller can obtain and act upon data provided by the elevators. The E.E.C.S receives interrupts from floors and elevators. Elevators generate interrupts when their status changes. Their status includes information about their current weight, location, direction of travel, and destination. An on-chip 512K word RAM stores the collected data from the building. The specialized instructions, SEND and RECV, allow the controller to easily communicate with the outside world without reducing the speed of the processor.

System Overview

Lifestyle in the United States is very fast-paced. It is a society of instant gratification where wait time should not exist. An elevator's speed, unfortunately, is limited by its mechanics, but improvements on elevator dispatching algorithms have made it possible to decrease effective wait time through software. The concept of E.E.C.S is to provide the hardware and software to gather and process data to minimize time spent waiting for an elevator.

E.E.C.S is relatively simple thanks to VLSI. The alternative would be to build the controller with existing standard parts on a large board, however, the implementation of specialized instructions cannot be realized in this fashion. Also with large, cumbersome boards, connections between parts may slow down processing time. VLSI design allows for the modification of a microprocessor, possible decrease in cost due to integration, and a smaller footprint. In addition, it will be more cost efficient to build considering today's efficient fabrication technology.

The E.E.C.S core consists of a modified RISC microprocessor and RAM. A standard ALU and shifter allow the microprocessor to complete all basic arithmetic, logic, and shift functions. The processor is also outfitted with interrupt capabilities to facilitate communication to elevators and floors. On-chip RAM holds the necessary working data for the algorithms.

Interrupts and Communication

Interrupts, specialized instructions, and testability options differentiate the E.E.C.S from a regular 16-bit microprocessor. Interrupts are implemented to inform the controller of elevator status and floor requests. New instructions are added to a baseline architecture to support interrupts. This includes commands to enable and disable interrupts and also to return from the interrupt service routines. The E.E.C.S uses two new instructions: SEND and RECV. SEND causes the controller to transmit the appropriate information to the elevator. The RECV instruction directs data coming into the controller to specific RAM locations. Since our algorithms are predominantly communication-based, it was efficient to create a dedicated RECV instruction that places elevator data directly in memory (whereas a memory-mapped method of I/O would require two instructions – a load and a store). The SEND instruction is the logical counterpart of the efficient RECV. Having these special instructions conveniently eliminates the need for the extra decoding logic that would be required in a memory-mapped scheme.

Design for Test

The SEND instruction is very versatile because it can be used to test the functionality of the chip. Using this instruction, data from registers or memory becomes accessible on the external pins. The contents of the program counter and the instruction register are available through external connections to the instruction ROM. Scan options are also available on the program counter and instruction register to force certain instructions for the purposes of testing.

Implementations and Engineering Considerations

The E.E.C.S implements a standard set of RISC operations (please refer to Table 1). In addition to these operations we include five new instructions. Three instructions are dedicated to operating the interrupts. The enable interrupt instruction (EI) enables interrupt capability. Once this instruction is executed, the controller can accept interrupts. The disable interrupt command (DI) sets the external signal, *busbusy*, high which informs the elevators and floors that interrupts will not be processed. E.E.C.S. uses a modified form of vectored interrupts. There are two interrupt signals, IRQ0 and IRQ1, generated by the elevators and hall calls respectively. The interrupt service routines for these interrupts are stored in the instruction ROM at FFE0 and FE00 respectively. In general, elevator interrupts takes precedence over hall interrupts. If two interrupts come at the same time, the *busbusy* signal will allow only one interrupt to be processed. The interrupt chosen is determined by which address is holding on the line at the moment the *busbusy* signal is set.

Hall call interrupts force a specific interrupt vector (one for each hall call button) into the program counter; elevator status interrupts are all handled in one ISR. When an interrupt occurs, the current value of the program counter (PC) and program status register (PSR) are stored for later retrieval. To return to the original program, the return instruction (RETX) is given. This instruction loads the program counter and PSR with the values they contained prior to the interrupt. The last two instructions are used to communicate with the elevators and floors. The SEND instruction takes data from the register file and sends it to an elevator uniquely addressed by concatenating the contents of a register and 0x1. The receive instruction (RECV) stores the data being transmitted from an elevator to the appropriate location in memory.

For interrupts to function properly, precise timing is crucial. The controller's clock is phase-delayed by 15 nanoseconds to insure that the current instruction has been delivered to the controller before any control signals are generated. To correctly support branch instructions, the program counter's clock is delayed by five nanoseconds. This insures that the ALU calculates a branch address based on the current PC rather than the next PC. Delays were built into the controller via the EPOCH *delayps* part.

The scenario created for the E.E.C.S set many assumptions for its design and testing. The scenario uses the E.E.C.S in a 64-floor building that contains 16 elevators. The E.E.C.S operates with a five-volt power supply due to fabrication process (1.2u2m1pHP) use in its design. The environmental conditions in the building are assumed to be very consistent and extreme temperature conditions are not considered in our testing. Our measurements are based on typical operations of the E.E.C.S.

The E.E.C.S is separated into three different sectors: datapath, control unit, and RAM. The datapath consists of the program counter, instruction register, register file, arithmetic logic unit and shifter. All these components are full-custom layout for the E.E.C.S. To save layout time, the control unit, which contains the PSR and interrupt registers, is generated from hand-written verilog code and synthesized by EPOCH. This resulted in a design larger than would have been realized by a full-custom design, but allowed for quick layout and easy modification. The RAM is a part from EPOCH's standard cell library. EPOCH was also used as an automatic place-and-route and floorplanning tool.

The positive edge-triggered program counter and instruction register are designed for testability. By setting the scan mode on the chip, the E.E.C.S controller can be forced to a new address or to execute a specific instruction. Further verification of the contents of these registers can be accomplished via the pins of the E.E.C.S. chip (Figure 1)

The E.E.C.S uses a 16 bit by 8-word negative-edge triggered register file. Without the need for complex manipulation, the E.E.C.S controller can operate efficiently with only eight registers. Register contents can be tested by executing the SEND command. With the SEND command, the register values are sent to the external *elevdata* pins and can be probed.

With the convenience of on-chip RAM, accesses to memory will not take very long. However, the data from the external devices (elevators and hall buttons) will arrive very slowly compared to the operating frequency of the E.E.C.S. controller. Thus, the E.E.C.S. can afford its relatively slow speed (compared to modern VLSI circuits) in order to decrease chip size. Since speed is not an issue, we could afford to implement a simple ripple-carry adder with small NMOS pass gates. In fact, the ripple carry adder creates our critical path. Accusim simulations on a one-bit slice of the ALU revealed that a rippling subtraction operation requires approximately 40 nanoseconds to complete. The same size-over-speed tradeoff is made in the design of the logarithmic shifter. Size is reduced almost by half using a series of "flips" to complete a right shift. A right shift can be executed by flipping the value once (bit 15 becomes bit 0, bit 14 becomes bit 1, etc.), shifting it left by the appropriate amount, and flipping it once more. The original design of the logarithmic shifter required a separate set of shifts to implement the right shift. Now right shifts use the same shifters as the left shift function.

Testing of the entire datapath and individual components is completed through Mentor Graphics' Quicksim and Accusim simulation programs. Functionality for all instructions is tested. The complete core of the E.E.C.S. is tested through Signalscan and Quicksim. However, the chip will not be fabricated and further testing is not arranged.

Summary

The E.E.C.S is a fully functional microprocessor, but its specialized architecture is suited for the proposed elevator scenario. Size and functionality are the goals of the E.E.C.S and have been executed successfully. Considerations for testability are met with the available scan functions and pin interfaces. The concepts of the E.E.C.S are easily realizable and are valid improvements to consider for future designs of elevator controllers.

References

AIA/ACSA Research Council student design competition. Bristol: United Technologies Otis Elevator, 1986-1990.

http://www.elevator-world.com

http://www.otis.com

Strakosch, George R. Vertical transportation: elevators and escalators. New York: John Wiley and Sons, 1967.

Strakosch, George R., Ed. The Vertical Transportation Handbook, Third Edition. New York: John Wiley and Sons, 1998.

TABLE 1 – Instruction Set

ADD, SUB, CMP, AND, OR, XOR, LSH	Arithmetic, Logic, and Shift functions with register values
ADDI, SUBI, CMPI, ANDI, ORI, XORI, LSHI	Arithmetic, Logic, and Shift functions with immediate values
MOV, LUI, LD, ST, Bcond, Jcond, JAL	Various Data Loading and Storing functions. Also standard Branch and Jump commands
DI, EI, RETX	Interrupt commands
SEND, RECV	Elevator communication commands

TABLE 2 – Chip Statistics (before addition of padframe)

Die Size	3.5mm x 3.3mm
Total Power	Unknown
Number of Transistors	71266
Density of Layout	6251 / mm^2
Maximum Clock Speed	3.8 Mhz

FIGURE 1 – Block Diagram of E.E.C.S.