A Fail-Safe Interlocking System for Railways

VINOD CHANDRA
Indian Institute of Technology, New Dehli

M.R. VERMA
Indian Railway Service

The authors describe a new approach to the design of microprocessor-based fail-safe systems for railways. The method, which was used to design the FIRM architecture, involves assigning appropriate levels of safety to system functions, depending on how critical they are, instead of using the same safety standard for all functions. The FIRM (short for fail-safe interlocking system for railways using microprocessors) architecture uses a pair of processors that operate in a see-saw mode, with one or more pairs kept on standby. An engineering prototype of the architecture has been fabricated for Indian Railways.

Although existing design techniques to develop fault-tolerant and fail-safe systems vary, all of them design to a global safety level, regardless of the standard of safety needed for a particular function. With this approach, the most critical function of the system dictates the safety standard for the entire system.

Such a global safety standard wastes resources and increases hardware and software complexity, which leads to costly testing and validation. Consequently, it can actually reduce system reliability, since fewer errors are caught when testing and validation is that complex.

We think there is a better approach: Implement functions critical to system safety at a higher (different) safety standard than routine functions, which have little or no bearing on overall safety. We have used this approach to develop the FIRM architecture, a fail-safe interlocking system for railways using microprocessors.

The FIRM architecture consists of a pair of processor modules that operate in duplex mode, with one or more pairs kept as standby. A sequence-controller module, called SQC, supplies power to the active pair through a switch. When a fault is detected in the active pair, SQC switches off its power supply and activates a standby pair of modules.

FIRM also has a number of functional modules, which are connected to the processor-communication bus, or PC bus. These modules perform the following major tasks:

- conduct a fail-safe comparison of the output signals of the two processors
- check the signals that affect system safety
- store the states of the system and maintain the system outputs at the status quo level when there are no processor signals. Thus, new processors, which are activated during recovery, can update their memory from the functional modules and resume normal operations.

Levels of safety

The choice of design technique for safety systems depends on the safety standard required. As we pointed out earlier, a globally enhanced safety requirement not only increases hardware and software complexity, but also

0740-7475/91/0003-058/$01.00 © 1991 IEEE

makes the design's testing and validation more difficult, since processor-based systems are not fully testable for all possible failure modes. For a large complex system, software development and hardware maintenance would be prohibitive. A system of this type would require a large amount of test software as well as dedicated hardware for proving system safety.

If, on the other hand, we use appropriate design techniques for different functions depending on their safety requirements, we avoid this complexity. The level of safety of a function is correlated to its contribution to overall system behavior. If, for example, an error in function performance is likely to result in the issuing of unsafe commands, that function is designed to a greater level of safety than, say, a routine function, which causes only minor penalties when it performs incorrectly.

Kaul gives a detailed description of a route-interlocking system—a slow, real-time system used in safety applications—and its various functions.[1] The functions of this system, which is used to control the movement of trains in yards, fall into the following categories:

1. data logging and maintenance support
2. panel processing
3. route holding, point locking and route setting
4. signal clearance and route monitoring
5. output-data conditioner

Table 1 shows the consequences when these functions fail and gives their safety level.[2,3]

Selecting design techniques

The next step in the design process is to select appropriate design techniques for implementing these functions according to their assigned safety level. Table 2 shows how the different functions are divided into four categories, each with its own design techniques. The function at the lowest level of safety—data logging and maintenance support—is implemented by a single processor with routine software. The function at the next level of safety—panel processor—is implemented through software diversity (see box below). The vital functions—route holding, point locking, signal clearance, route monitoring, and route setting are implemented at safety level 3 using specially developed fail-safe hardware. Finally at safety level 4, the highest level, the output data conditioners are implemented through electromechanical relays. These relays are proven safety devices and are commonly used in the entire system.

The FIRM architecture

The FIRM architecture supports all the categories of functions just described. It has been developed for slow real-time systems used in safety applications, such as route-interlocking systems for railways. Various route-interlocking systems based on microprocessors exist, but as we mentioned earlier, these are based on hardware and software redundancy techniques and thus have two drawbacks. One is that their design is based on a uniform safety standard across functions. The other is that common-mode and time-correlated failures go undetected.

As we mentioned earlier, a uniform safety standard leads to a system design that is too costly and too difficult to validate. The other drawback of conventional fail-safe systems is that when errors such as hardware and software developmental errors, external interference, and errors from identical component failure go undetected, there is often a safety hazard.

We can solve the first problem by using an approach based on assigning different safety levels to different functions. A model for FIRM is based on this approach. The second problem stems from tightly synchronized architecture, which is due to the use of hardware redundancy. Since the processors' outputs have to agree in each machine cycle, we must use identical hardware and software resources, but this makes the system prone to common-mode and time-correlated errors.[4,5] Systems using software redundancy have different problems. Although some researchers claim that most hardware and software faults can be detected if diverse software is used,[4,6,7] fault coverage is difficult to estimate because software errors are inherently complex. Moreover, the two programs have to be somewhat correlated, so we cannot produce different system reactions for all possible hardware faults. Thus, we may get unsafe results when the hardware fails.

The FIRM architecture attempts to overcome this drawback and increase the overall safety level. The cost is

WHAT IS SOFTWARE DIVERSITY?

Software (or hardware) diversity is the addition of system elements in software (or hardware) to provide more than one functional channel.

Software *redundancy* is more commonly used to achieve reliability, but it may permit system operation in a permissive state despite partial failure. Software diversity, on the other hand, places system output in a restrictive state, checking multiple channels for failure and inconsistencies when such conditions occur.

The diverse elements that cooperate to form different channels must be assembled in a logical way, however.

Table 1. Assignment of safety levels to functions of the route interlocking system.

Function	Description	Consequences of Failure	Safety Level
F1	Panel processor handling,	Incorrect route setting delay and confusion in traffic but no safety hazard	2
F2	Route setting	Setting of conflicting routes, disturbance of route, change of point position under running train	3
F3	Route holding	Disturbance of route, may cause accident. Detected by point locking & route monitoring function	3
F4	Point locking	Changing of point under running train causing derailment. Checked by route holding process	3
F5	Signal clearance	Clearance of conflicting signals, serious accident. Can be corrected by output data conditioners if detected in time	4
F6	Route monitoring	—do—	4
F7	Output data conditioners	Clearance of conflicting signals, random-point operation, serious accident if not detected and corrected by any other process	5
F8	Data logging, maintenance support	Incorrect management information, loss of records, maintenance problems, but no operational difficulties	1

Table 2. Implementation of different functions.

Safety Level	Functions	Implementation
1	Data logging and maintenance support	Single processor; routine software
2	Panel processor	Software diversity
3	Route holding point locking signal clearance route monitoring route setting	Fail-safe electronic hardware and dual hardware/ software diversity
4	Output data conditioners	Electromechanical safety relays

slower operation and increased hardware resources, but system complexity decreases. The architecture consists of a pair of processor modules operating in duplex mode with one or more identical pairs kept as standby. Figure 1 illustrates. The processors operate in seesaw mode; that is, at any given time, one of the processors, say Processor A, is in full control of the communication bus and other shared resources. During this period, Processor A performs the I/O operation while Processor B performs the control tasks. In the next phase of the cycle, Processor B takes control of the communication bus and vice versa.

The two processors move from one phase to the other using a handshaking mode; thus, there is no need for synchronization. With asynchronous operation of the two processors, we can use both hardware and software diversity, eliminating the possibility of unsafe failure from common-mode errors.

We used many functional modules in the implementation of the FIRM model.[2,3] The first layer of the model is the panel processor, which is at safety level 2. It is implemented by software diversity through the processor modules.

The next layer is the vital logic processor, which consists of functions F2, F3, F4, F5, and F6 of the route-interlocking system. This layer, which is at safety levels 3 and 4, is implemented through specially developed hardware circuits housed in a number of functional modules. The functional modules cross check the vital signals and examine the safety aspect of the processors' outputs before delivering the final results.

The final layer of the model—output data conditioner— is at safety level 5 and is implemented through proven signaling relays. Outputs of the functional modules drive these relays.

Thus, in the FIRM architecture, safety does not depend solely on the processor's hardware and software. Instead, a number of distributed functional modules provide the safety assur-

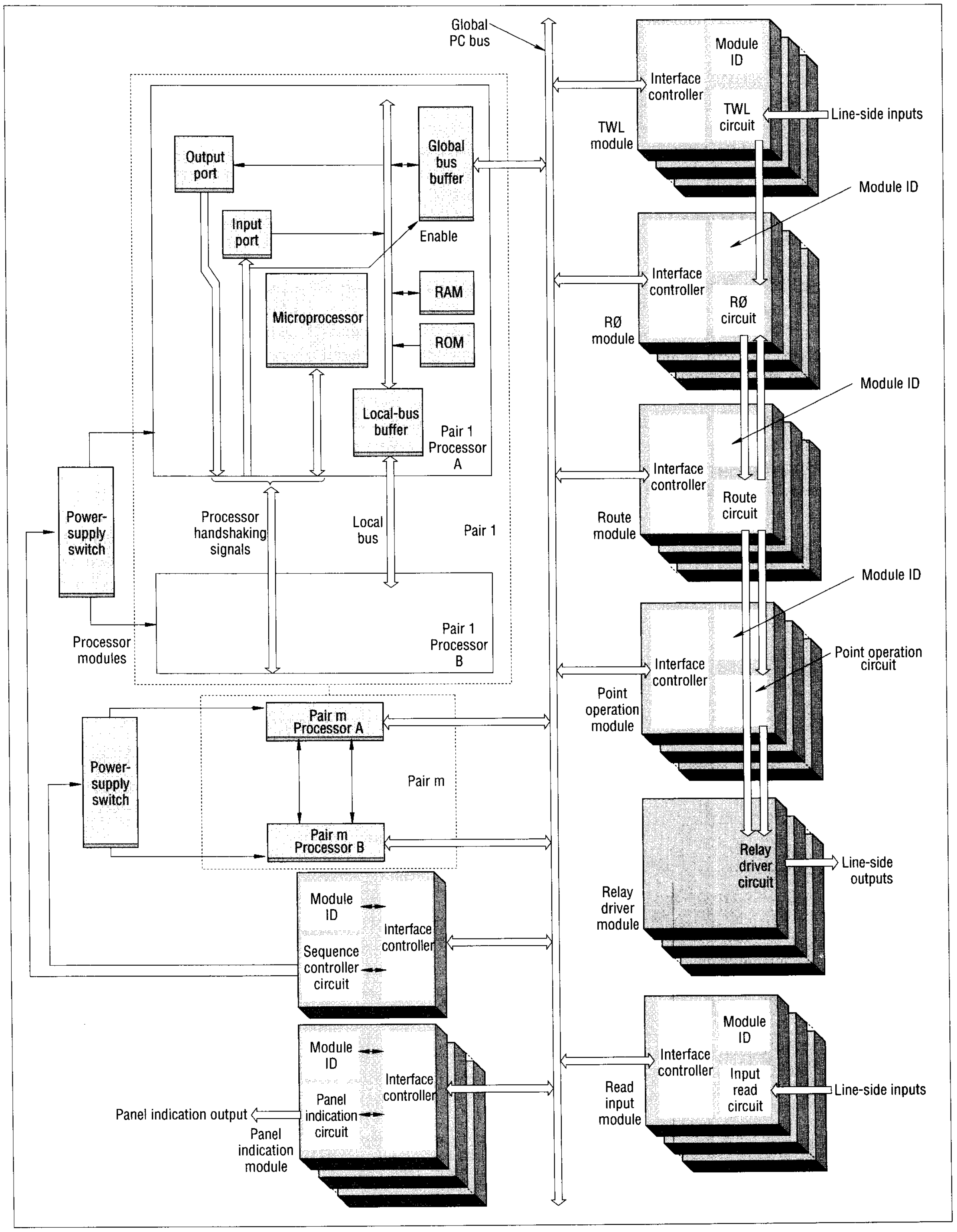


Figure 1. The FIRM architecture.

ance. Since each functional module performs a specific job, we can design, validate, and test each module independently. This approach simplifies design validation with regard to system safety as compared with existing approaches, which require the validation of all processor hardware and software.

We designed the FIRM architecture to ensure that a single fault does not lead to an unsafe system failure. The fault is detected and recovery action is initiated quickly enough to avoid the occurrence of double faults.

In the following sections, we provide only a brief description of the components in the FIRM architecture. A more in-depth description is available elsewhere.[2,3,8,9]

The sequence controller

The sequence controller, or SQC, controls system configuration and supplies power to the active processors. Each processor of the active pair has enough power to perform a self-check and cross-check, and each is required to periodically set a flag in the SQC. If a processor fails to set the flag within the specified time, the SQC kills that processor's power supply and initiates fault recovery by activating a spare pair of processors.

During recovery, which briefly interrupts system operation, the functional modules maintain system outputs at the status quo level. When a new pair of processors is activated, the processors perform a power-up self-check before starting normal operation. Since the configuration is meant for slow, real-time processes, the brief interruption during recovery should be tolerable.

Processor modules

Figure 2 shows the architecture of the processor modules. Each module consists of a microprocessor and its memory, which are connected to the global PC bus through bidirectional buffers that the companion processor controls. A local communication bus provides for the exchange of data between the two processors of the active pair during cross-check. This exchange happens when important results of the previous computation are exchanged in DMA mode and verified. Each processor has an input and an output port for generating interrupts and exchanging handshaking signals with the companion processor.

See-saw mode

The processors can operate in one of two modes: the computation mode and the communication mode. The two processors of the active pair see-saw be-

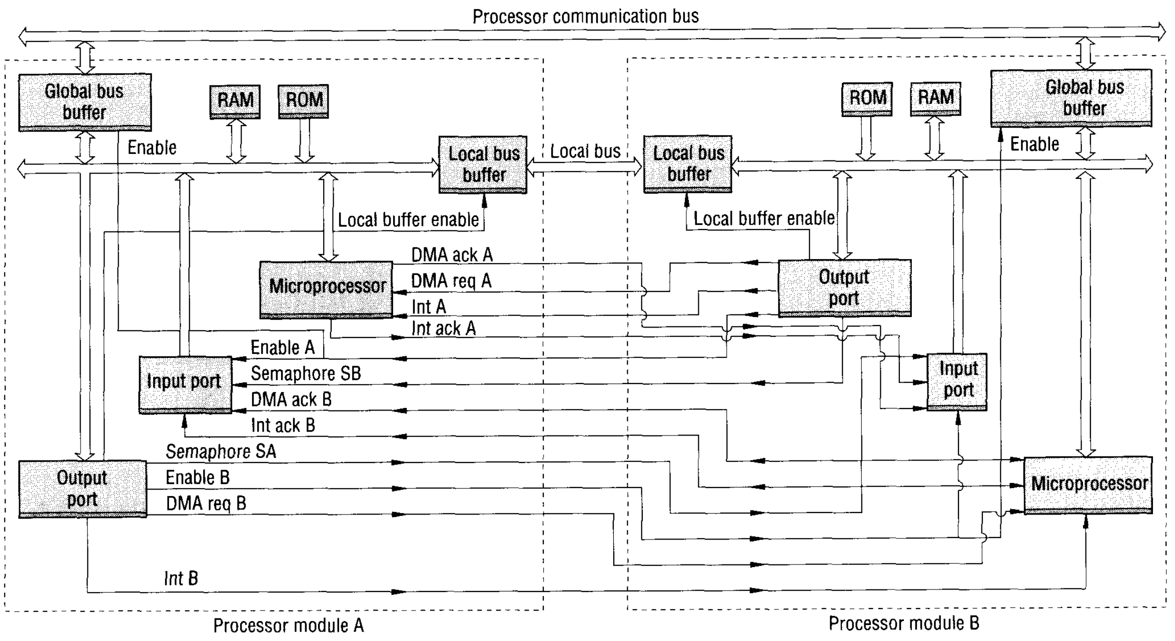


Figure 2. Architecture of the processor module.

tween the two modes. That is, at any given time, one processor is in computation mode, and its companion is in communication mode. In next phase of the cycle, they change modes by handshaking.

The processor in communication mode gets control of the global PC bus to communicate with the functional modules and the local bus to access the vital results of the companion processor, which it needs to perform a cross-check. Communication mode includes the following activities:

- reading the vital results of the companion processor under DMA
- performing I/O operation on the functional modules by polling
- executing the self-check routine, which continues until an interrupt is received from the companion processor

When the processor receives an interrupt from its companion, it shifts to computation mode and performs the following functions:

- releases its bus for DMA operation by the companion unit and waits until DMA is complete
- performs control tasks on the data read from the functional modules during the last I/O operation and stores the outputs in RAM
- compares the vital results of the computation with those of the companion processor (which were read during the communication mode). When a repeated discrepancy is detected, the SQC initiates the recovery process.
- checks the bus buffers. This activity is important, since a fault in these units may lead to bus contention. Both the global and local bus buffers of the companion processor are checked in this mode.

The operating cycle is the same for both processors.

The see-saw operating mode allows complete asynchronous operation of the two processors. This type of operation has a number of advantages over the use of synchronized architectures for safety systems. First, we can easily implement hardware and software diversity. We also avoid deadlocks in the access of shared resources. Because processors operate at different time intervals, we avoid time-correlated errors. Finally, we do not have to consider the problem of how to synchronize processors.

The recovery process

As mentioned earlier, the SQC controls the system's configuration. Whenever an error is detected in any processor module of the active pair, the SQC switches off that processor's power supply and activates a spare pair of processors by switching on its supply and providing a reset signal. Each processor of the active pair must periodically set an OK flag in the SQC. Whenever a processor detects a fault as a result of either a self-check

EXISTING FAIL-SAFE DESIGN TECHNIQUES

The following references comprise a good cross-section of work on the design of fail-safe systems based on a global safety standard (unlike the approach described here, which is based on the assignment of safety levels to functions according to how critical the function is to overall system safety).

1. P. Neumann, "On Hierarchial Design of Computer Systems for Critical Applications," *IEEE Trans. Software Engineering*, Vol. SE-12, No. 9, Sept. 1986, pp. 905-920.
2. M. Subramaniam, M. Verma, and V. Chandra, *Use of Electronics in Railway Signaling,'* tech. rpt. Care/85-2, IIT Delhi, Mar. 1985.
3. H. Lohmann, "Fail-Safe Data Processing in Railway Signaling Systems," *Siemens Forsch- u. Entwickl-Ber*, Vol. 7, No. 6, 1978.
4. H. Lohmann and A. Zillmer, "Safety Principle and Fail-Safe Analysis of Electronic Interlocking Devices and Practical Realization of Electronic Interlocking," *Proc. Int'l Conf. on Railway Safety and Automation*, 1984, pp. 41-48.
5. K. Wobig, "Fail-Safe Microcomputer Systems for Railway Signaling—Problems and Possibilities," *Proc. Conf. on Railways in the Electronic Age*, IEE No. 203, pp. 164-168.
6. I. Okumura et al., "The Development of an Electronic Interlocking System," *JNR Quarterly Report*, Vol.22, No.4, 1981, pp. 161-167.
7. M. Kaul, *Modern Railway Signaling (Principles and Practice)*, Bahri Bros, Delhi, 1983.
8. A. Cribbens, "The Solid State Interlocking System," *Proc. Int'l Conf. Railway Safety and Automation*, Inst. Railway Signaling Engineers, London, Sept. 1984, pp. 24-29.
9. A. Cribbens, M. Furniss, and H. Ryland, "The Solid State Interlocking Project," *Proc. Conf. Railways in the Electronics Age*, No. 203, IEE, London, pp. 1-5.
10. A. Cribbens, M. Furniss, and H. Ryland, "An Experimental Application of Microprocessors to Railway Signaling," *Electronics and Power*, Mar. 1978.
11. R. Mathson, "Computers Replacing Safety Relays in Railway Signaling," in *Digital Computer Application to Process Control*, V. Lemke, ed., North-Holland Pub., Amsterdam, 1977, pp. 321-328.
12. D. Disk, "A Unique Application of a Microprocessor to Vital Controls," *Proc. Int'l Conf. Railway Safety Control and Automation*, 1984, pp. 97-104.
13. D. Rutherford, "Fail-Safe Microprocessor Interlocking with Safety Assurance Logic: Establishing a Vital Benchmark," *Proc. Int'l Conf. Railway Safety Control and Automation*, 1984, pp. 1-5.
14. O. Von Linde, "Computers Can Now Perform Vital Functions Safely," *Railway Gazette Int'l*, Nov. 1979, pp. 1004-1007.

or a cross-check, it moves into a safety trap and stops the setting of the flag. This causes the SQC to kill its power supply and activate a spare pair.

During a cross-check, however, the outputs of the two processors are allowed to disagree for a fixed number of cycles without recovery action. If at the end of these cycles, the outputs match, the SQC does not initiate recovery. The reason is that the processors read the functional modules and the feedback outputs of the line-side equipment independently. Thus, outputs may disagree for a period, depending on the time constants of the functional modules and the interface circuits.

When the SQC activates a new pair of processors, each processor performs a brief power-up self-check and then updates its memory according to the state of the system stored on the functional modules. After initialization, one processor branches to the communication mode, while the other branches to the computation mode, and they continue see-sawing between modes.

Thus, recovery has little or no impact on the system environment. Although the recovery process includes a brief interruption in system operation as a faulty pair is switched off and a new pair activated and initialized, this interruption is tolerable in slow, real-time systems. Moreover, the functional modules store the status of the system and maintain system outputs, so the process can continue from the point at which recovery action began.

A typical functional module

Functional modules perform two major tasks. First, they generate system outputs after doing a fail-safe comparison of the processor output and checking vital signals in the hardware. Second, they store the states of the system for smooth recovery if there are processor faults. The modules are connected to the global PC bus through which the processors communicate with the modules by polling.

Figure 3 shows the architecture of a typical functional module, which consists of

- *an interface controller*, which controls the processors' I/O operations. The interface controller in Figure 3 has three buffers: a data buffer, an address buffer, and a control buffer. These buffers isolate the module from the PC bus. The processor selects the module through a page-select circuit. The 8-bit address space is divided into 32 pages, and a unique page number is assigned to each module.

- *a module ID*, which is the individual identification of the module. The module ID is hard-wired as an 8-bit word and read by the processors through the ID port. The page address allotted to the module is also hard-wired as a 5-bit word. The processors verify the identity of each module during self-check to ensure that the module is connected to the PC bus, and that its page select and address decoding circuits are functioning. This check acts as a security kernel around the module.

- *a data-loop test*, which is performed on each functional module as part of the self-check to verify the health of the data bus. The test is done from the processor's internal bus to the module interface controller. The test checks the processor's global bus buffer, the PC bus, and the data buffer on the module. Four-bit test data is written on the test port through data lines D4–D7 and the same is read back through lines D0–D3. Test data is changed every time to detect all possible stuck-at and coupled faults on the data bus. To prove the integrity of the individual control lines of the two pro-

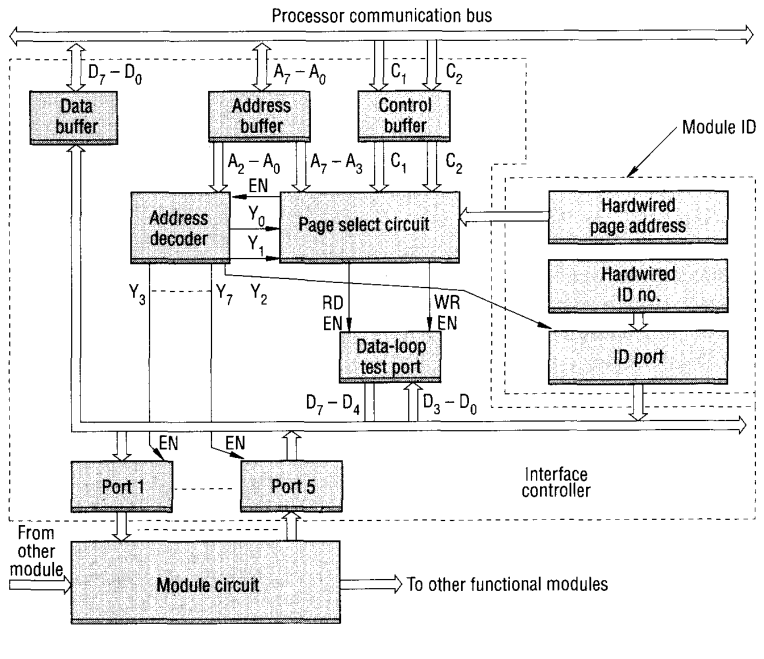


Figure 3. Architecture of a typical functional module.

Table 3. Requirements of modules for the prototype.

Safety Level	Module	No. of Modules	Remarks
1	Processor	4	One active pair and standby pair
2	SQC	1	Controls two pairs of processors
3	Route	10	Each module caters to two routes
4	TWL	5	Each module can process three to five routes
5	RQ	1	Each module controls about 20 routes
6	Input read	3	Each module can read 48 inputs
7	Relay driver	8	Each module can drive four relays
8	Point operation	2	Each module can control four points or level-crossing gates
9	Panel indication	1	The module can provide 40 indications

cessors, the processors have different addresses for accessing the test port.

The interface controller and module ID are identical for all the modules, but the module circuit differs with the type of module, depending on the required functions.

An engineering prototype

As a first step toward making an engineering prototype of the FIRM architecture, we used the route-interlocking simulator[7] to develop and validate the design of FIRM subsystems, including processor modules, the PC bus, and functional modules. We then built the FIRM prototype for a station of India's Northern Railway, called Brar Square. This station has been earmarked for the experimental installation of FIRM by Indian Railways. Brar Square is located on the "New Delhi Avoiding Line," through which long-distance goods traffic is diverted without affecting the New Delhi main station. It also handles the passenger trains of New Delhi local traffic. About 80 to 85 trains per day are handled at this station.

The station now has a route-interlocking system that uses conventional electromechanical relays. The plan is to replace this conventional system with the FIRM architecture in several phases.

The prototype consists of the processor modules and functional modules described earlier. Table 3 gives the number of functional modules of each type required for the station.

These modules are plugged into a mother board, which interconnects the modules and the communication buses. Modules of like type are interchangeable and their design is independent of how the yard is laid out. Specific requirements of the yard are incorporated in the mother board, which varies from yard to yard.

Installation

The experimental installation of the system at Brar Square is being done in three phases. In Phase 1, which was commissioned in May 1988, we connected inputs from the line-side equipment and the control panel to the system and used a data logger to monitor the outputs. The data logger also compares the outputs with those of the existing installation.

In Phase 2, which we are in now, we plan to connect vital outputs of both the existing system and FIRM architecture in series to drive the line-side equipment. For example, we will issue the final signal clearance command after we give both systems their respective commands. We will use proven signaling relays to perform the series function. Again, the data logger will monitor the outputs to detect and locate faults. We have also provided the operator with the ability to transfer full control to the existing system if repeated failures occur.

In Phase 3, we will disconnect the experimental installation and transfer full control to the FIRM system.

We are now monitoring the route-interlocking system with a data logger and comparing the results with the performance of the existing, relay-based system. Unfortunately, there is a time lag before the two system outputs agree, since the relay-based system is slow, and the delay in relays is not precise. Consequently, we can compare only the final, stable outputs and have no meaningful comparison of intermediate steps. System performance from the standpoint of operational and safety requirements has been satisfactory.

Testing

We have conducted several kinds of tests on different modules of the prototype. We did functional testing to ensure that the module performed the intended functions for different input patterns. We also tested the fail-safe feature under different fault conditions, such as shorted adjacent IC pins and an open circuit in the resistor. We then verified that modules fail on the safe side. Except for the analog signals, functional testing of the modules is done automatically by processors using test programs.

We have proposed a new approach to designing fail-safe systems that differs from an implementation in which all system functions are designed to the same standard of safety. By designing levels to their required standard of safety, we avoid the complex software and hardware and thus simplify testing, which in turn increases the overall safety level we can achieve.

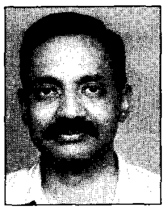
We used our approach in the design of the FIRM architecture, in which we identified five levels of safety. We partitioned the design into these safety levels and selected appropriate techniques for their implementation. An engineering prototype of FIRM has been built and is currently under field test by Indian Railways.

Acknowledgments

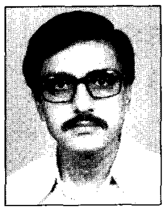
This work was supported by a grant from the Department of Electronics, Government of India in collaboration with Indian Railways.

References

1. M. Kaul, *Modern Railway Signaling (Principles and Practice)*, Bahri Bros, Delhi, 1983.
2. V. Chandra and M. Verma, "A New Approach to Design of a Processor-Based Route Interlocking System," *Proc. Int'l Seminar and Exhibition on Railway Electrification*, No. RE-85, Indian Railways, New Dehli, 1985, pp. 169-176.
3. M. Verma and V. Chandra, "The Design and Development of a Fail-Safe Interlocking System Using Microprocessors for Indian Railways," *Proc. Region Ten IEEE Conf.*, IEEE Press, New York, 1989, pp. 511-514.
4. R. Mathson, "Computers Replacing Safety Relays in Railway Signaling," in *Digital Computer Application to Process Control*, V. Lemke, ed., North-Holland Pub., Amsterdam, 1977, pp. 321-325.
5. P. Neumann, "On Hierarchial Design of Computer Systems for Critical Applications," *IEEE Trans. Software Engineering*, Vol. SE-12, No. 9, Sept. 1986, pp. 905-920.
6. K. Wobig, "Fail-Safe Microcomputer Systems for Railway Signaling—Problems and Possibilities," *Proc. Conf. on Railways in the Electronic Age*, IEE No. 203, pp. 164-168.
7. I. Okumura et al., "The Development of an Electronic Interlocking System," *JNR Quarterly Report*, Vol. 22, No. 4, 1981, pp. 161-167.
8. M. Verma and V. Chandra, "A Software Package for Simulation and Validation of New Designs of Route Interlocking System for Railways," *Proc. Soc. for Computer Simulation Eastern Simulation Conf.*, University of Ghent, Belgium, 1988, pp. 9-12.
9. V. Chandra and M. Verma, "A Microprocessor Based Route Interlocking System for Control of Trains in Railway Yards," *Proc. Vehicular Technology Conf.*, IEEE Press, New York, 1988, pp. 680-681.



Vinod Chandra is a professor in the Department of Electrical Engineering at the Indian Institute of Technology in New Dehli, where his research interests include communication systems, fault-tolerant and fail-safe systems, railway signaling systems, and microprocessor applications. Previously, he was a visiting professor at Florida Atlantic University, Boca Raton. He holds a BTech and a PhD in electrical engineering from IIT and received the National Inventions Award in 1978 for his work on automatic signaling for Indian Railways.



M.R. Verma has been with the Indian Railway Service of Signal Engineers since 1983, where he is now pursuing research and development work on fail-safe systems. He holds an ME (master of engineering) with distinction in automation from the Indian Institute of Science, Bangalore, and a PhD in electrical engineering from the Indian Institute of Technology, New Dehli.

Send comments or questions on this article to V. Chandra, Dept. of Electrical Eng., Indian Institute of Technology, New Delhi 110016, India.