IMPLEMENTATION OF NON-INTRUSIVE FAULT DETECTION IN EMBEDDED CONTROL SYSTEMS

Domen Verber, Matej Šprogar[1], Matjaž Colnarič

University of Maribor Faculty of Electrical Engineering and Computer Science, Maribor, Slovenia

Key words: embedded control systems, fault management, fault detection, monitoring cells, evolutionary computing.

Abstract: Paper presents fault detection in embedded control systems by the so-called monitoring cells. The basic idea is to monitor input/output variables and internal states of systems, processes or sub-processes by using acquired and built-in knowledge about the normal behavior in order to detect abnormalities. Paper gives the detailed architecture and the operation of the monitoring cells. The concept is applicable even if only a limited knowledge about the control system is available. In such cases the proposed automated learning of the monitoring function can be used. In the second part, two different implementations of the monitoring cell are presented. The first one uses discrete analogue devices and a field programmable gate array. The second is based on the programmable system-on-a-chip devices.

Izvedba neintruzivne detekcije napak v vgrajenih krmilnih sistemih

Kjučne besede: vgrajeni krmilni sistemi, ravnanje z napakami, detekcijanapak, nadzorne celice, evolucijsko učenje.

Izvleček: Članek predstavlja detekcijo napak v vgrajenih sistemih s tako imenovanimi nadzornimi celicami. Osnovna ideja je nadzor vhodno/izhodnih spremenljivk in notranjih stanj sistemov, procesov in podprocesov z uporabo pridobljenega in vgrajenega znanja o normalnem obnašanju in z namenom prepoznati nepravilnosti. Članek podrobno predstavlja arhitekturo in delovanje nadzornih celic. Koncept je uporaben tudi, kadar je na razpolago le omejeno poznavanje krmilnega sistema. V takšnih primerih se lahko uporabi predlagano avtomatsko učenje nadzorne funkcije. V drugem delu sta predstavljeni dve različni izvedbi nadzorne celice. Prva uporablja diskretne analogne enote in programirljiva FPGA vezja, druga pa temelji na programirljivih sistemih na čipu (pSoC).

Introduction

Embedded control systems are rapidly becoming the invisible mind behind most modern appliances. Their major spread could be observed even in safety critical environments, where failures can have serious or even fatal consequences. However, highly dependable programmable embedded systems for safety critical applications still lack proper scientific treatment. Controllers must be (besides being dependable) flexible in order to cut production costs. Flexibility (achieved mainly by programmability), however, is in conflict with dependability.

Faults in programmable control systems are unavoidable. Fault management as a discipline is embracing four types of techniques /7/: (a) fault avoidance is preventing faults in the design phase; 1(b) fault removal is attempting to find faults before the system enters service (testing); (c) fault detection is finding faults during system service and minimizing their effects, and (d) fault tolerance is allowing the system to operate correctly even in the presence of faults. A number of competent authors elaborated this area, for example /5, 1/.

Failure in a system can be handled for example by redundancy, diversity, reconfiguration etc. Firstly, however, it must be detected. For detection some sort of a dependable monitoring subsystem must be used that detects abnormalities and triggers appropriate corrective actions. Because of complexity, safety related issues of the system should be designed, evaluated and implemented independently and in parallel with the functional part; the same is with fault detection - to achieve it, a monitoring component called monitoring cell (MC) is introduced to supervise the control function. Early in the development of the system each MC is considered as an abstract object. Later on, the MCs are implemented as hardware and/or software components. Paper presents a MC concept for embedded control systems implemented in hardware either using discrete components with field programmable gate array (FPGA) or programmable system-on-chip (PSoC).

The MC must detect run-time faults, which are the most difficult to discover because they are a consequence of an unpredictable event or chain of events. One way to detect them is to observe whether the system's states are within reasonable limits at all times. To recognize what is

1 *Partially supported by Ministry of Higher Education, Science and Technology of Republic of Slovenia by the post-doctoral project grant no. Z2-6398-0796-04*

"normal" we propose to observe the system during normal operation by recording all inputs, outputs and internal states which are believed to affect system's future behavior. Based on these recordings a machine learning technique can be used to learn the normal behavior. Evolutionary algorithms (EA), for example, are one suitable paradigm for this task /3/.

Section 2 explains the concept of the fault detection by monitoring cells. The detailed description of their operation is given in the Section 3. A method for establishing the monitoring function using machine learning is described in Section 4. Finally, Section 5 gives two case-studies of MC implementation.

Detecting faults by monitoring cells

A typical present day control system consists of physical process components, sensors, actuators, distributed computers with communication networks, and a lot of software. Examples are control systems in industrial plants, nuclear reactors, avionics, etc. The size and complexity of such systems increases every year and so does the probability of a fault in one of the many components.

A control system can usually be divided into a set of well-defined sub-processes or tasks, running on a processing resource called *control cell*, each performing a specific *control function*. A task takes its inputs from sensors and/or from the results of other processes, produces results that can be used by other tasks, and controls the controlled system through actuators. Tasks are triggered by synchronous or asynchronous events.

Causes for faults in such systems are either hardware or software related; additionally the input and output signals may not comply with functional specifications of the system also causing faulty behavior. In hard real-time systems, improper temporal behavior is also considered a fault. Another, not so obvious reason for errors are temporal inconsistencies of the signals; e.g. the signals' changes can be too steep, frequency of events can be to high, etc.

The basic task of a monitoring cell is to monitor the validity of input $\boldsymbol{x}$ and output $\boldsymbol{y}$ values and (possibly) the internal states $\boldsymbol{m}$ of the control process. As a result, the correctness c is reported to a higher fault management layer that will handle the detected fault. If feasible, additional diagnostics parameter d can also be provided.

The control cell under surveillance must have physically accessible input and output signals. Since it can implement any (sub) process, it is important to identify control functions with clear and explicit relations between the input and output signals. The monitored control cell is considered a *gray* box with defined external functional behavior and with at least partly known internal structure, which is observable through its obtainable internal states. There are several reasons for that decision:

- The monitored component is not necessarily a black box and this knowledge can reveal additional and more accurate information about any faulty behavior.
- White box can be too complex. If chosen, it would be necessary to implement the monitor cell physically inside the original software and hardware to limit the communication problems. That, however, would be too intrusive and would increase the complexity and reduce the performance of the control part, what is counterproductive.
- White box implementation on the same resources would introduce another central point of failure.
- Clear separation between the control and monitoring function simplifies the design; both functions can be done separately and by different designers with diverse competence enhancing the dependability to some extent.

If the control function is a component with a well defined behavior but unknown internal structure, it can as well be taken as a black box. We propose to physically separate the control part and the monitoring cell, i.e. to employ separate hardware. While it may be more expensive, it provides much more competent implementation of the above guidelines. Also, complexity is kept lower by partitioning of the functions.

To allow for the gray box implementation, both input and output digital and analogue signals, as well as the internal states need to be observable. There is a number of possibilities for the latter: either they are made accessible via standard parallel or serial interfaces at the control cell, or another feature of contemporary processors is made use of, like JTAG boundary scan testing and in-system programming /4/ or a concept similar to "background debugging mode" high-speed clocked serial debugger interface by the CPU32 Motorola family.

Monitoring cell should be by orders of magnitude less complex than the control cell and should produce as little interference with the control environment as possible. It should be built from simple and robust components with low probability of failure. The consequence could be that the complexity of monitoring functions may be limited (e.g. no floating point arithmetic, etc.). This limitation, however, can be of advantage: because of the simplicity such a system could be formally verified and possibly certified by a certification authority.

When an abnormality is detected by a monitoring cell, the diagnostic mechanism will attempt to acquire as many details as possible. The error or a failure together with possible description will be coded in the diagnostic signal and mediated to the upper layers of fault management system where appropriate actions will be taken by, e.g., re-configuration manager. This, however, is beyond the scope of this paper.

Evaluation of the monitored signals

As shown in Figure 1, the control function F operates on input values ***x*** and accessible internal states ***m*** (if any) to produce output values ***y***. It is assumed that it operates in discrete time in a PLC-like fashion. It acquires its inputs at a specific point in time *t* and after some delay produces the results and updates the internal states. During this time no new inputs can be evaluated.

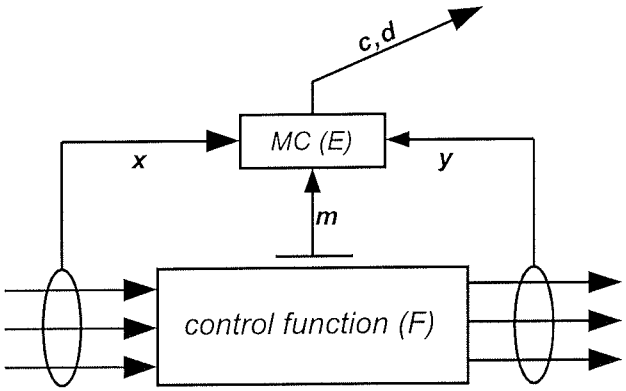

Fig. 1. Concept of the Monitoring cell.

By enumerating the time intervals, the notion of time can be reduced into discrete (integer) values as shown in (1), where F represents the control function.

$$(y(t+1), m(t+1)) = F(x(t), m(t)); t \in IN \qquad (1)$$

The MC acquires the current inputs at the beginning of the cycle at the instant *t*; at the same time the inputs x are read by the control function. At the end of the cycle (what is also the start of the next cycle) at instant *t + 1* (when outputs of the system are produced) the output ***y*** and internal state ***m*** of control function are read and the final evaluation of the subsystem is performed by MC. The MC effectively performs the function *E* defined by (2).

$$E(x(t), m(t), x(t+1), m(t+1), y(t+1)) \qquad (2)$$

Note that the function *E* can operate on both new and old instances of ***x*** and ***m***, what allows for early detection of discrepancies on the inputs and internal states at the instant *t + 1*.

MC operates on three groups of inputs – ***x***, ***m***, ***y***, the former two in instances *t* and *t + 1*. These are the values actually available to MC for evaluation. To simplify further discussion all inputs are called *signals* s_i; thus, function (2) can be rewritten as $E(\mathbf{s}(t))$.

To simplify the analysis and the implementation, the evaluation function *E* can be decomposed into a set of simpler evaluation tests E_{part} that evaluate parts of the system. At the end the partial evaluations are combined to produce the final result. If any of the sub-evaluations E_{part} indicates an illegal state of the system, the final result should also be noted as illegal.

A minimalist evaluation function E checks the integrity of individual signals. The information on their basic properties is acquired from the system specifications, technical documentation or similar sources. This way at least information on the data ranges – valid and invalid values of different signals – is extracted. In certain cases this is the only step that can be performed.

Using this information a simple classification of a signal into a legal or illegal class can be made. However, it is not always possible to find a sharp boundary between valid and invalid states. Additional buffer zones should be introduced where the validity of the signals can not be determined. This is the foundation of the partial evaluation function E_i that determines correctness c_i of the signal $\mathbf{s}_i$.

$$c_i = E_i(\mathbf{s}_i)$$

$$c_i \in C=\{\text{valid, invalid, undetermined}\}$$

This enables a simple validation of each signal. For example, any invalid input value can be detected (possibly from a faulty sensor) and/or illegal output can be detected and, consequently, prevented.

However, a more thorough validation of the system should be performed to make sure that outputs are consistent with the inputs. For this, the properties of transformation function *F* of the system must be known. If there is enough knowledge about the system behavior, analytical methods can be used.

Theoretically, it is possible to observe all possible combinations of signals, although such analysis is very complex. Instead, a partial signal dependency analysis can be performed, where the correlation between different pairs of signals $(\mathbf{s}_i, \mathbf{s}_j)$ is determined. For correlated pairs further analysis and simplification is plausible. Based on this, function E_{ij} can be constructed.

$$c_{ij} = E_{ij}(\mathbf{s}_i, \mathbf{s})_{,j} \; c_{ij} \in C$$

If the correlation of pairs of signals is unsatisfactory, a combination of three or more signals can be attempted.

Establishing the monitoring function

The monitoring function is sometimes impossible to create analytically (for example when the details of the control system's operation are unknown). One solution is to use machine learning (ML) to find it. Unlike analytical construction, ML is based on the recorded operation of the control cell – the control cell is observed during an interval of valid operation to produce a learning set *L* of instances (points in the search space *S*), every instance representing the input signals ***s*** to the MC. The decision models created by both ML and analytical approach are in turn used to determine the correctness *c* of any signal ***s***. MC must determine whether a currently occurring signal instance ***s*** belongs to the space of known valid instances in *L* or not – a classification / clustering problem. The MC's processing

limitations prevent the use of complex models to discover the overlapping of current instance with *L*.

The decision model must divide the search space *S* into disjoint sub-spaces – clusters. Cluster is a small group of signals of the same type. The created clusters should hold the samples from the learning set and represent the *valid* sub-space; the sub-space out of cluster(s) is holding *invalid* points. The main problem is that clusters are computationally difficult to create and use. Consider the learning set in Fig. 2: the clustering is easily done by a human. However, significant computing power is needed to create *and* use this oddly looking region as a classification model. Moreover, the number of clusters is also unknown making the clustering process even more difficult.

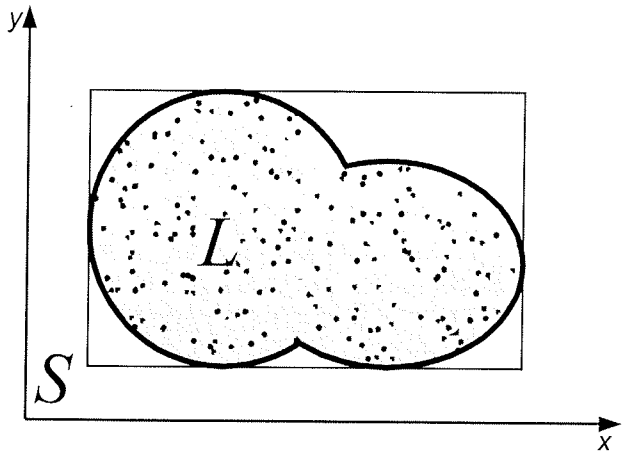

Fig. 2. Learning set enclosed in a human-drawn clustering region together with a bounding hyper-cube.

Hyper-cubes

The simplest and fastest monitoring function for the MC implemented in primitive hardware uses clusters in the shape of orthogonal hyper-cubes; a hyper-cube is limited by two opposing hyper-planes in every dimension. To avoid the troublesome creation and optimization of *n*-hyper-cubes a simple two-value (s_i, s_j) analysis is preferred. Since each component pair (s_i, s_j) is evaluated separately, a separate discretization of s_i is possible in the context of s_j and vice-versa. Such partitioning exploits the correlations between the parameters. The bounding hyper-cube (dash-dot rectangle in Fig. 2) is defined by the ranges of the values – the lower and upper limit are the simplest validation standards.

A completed model partitions the search space into disjoint sub-spaces with an outer bounding hyper-cube. The configuration of hyper-cubes is done off-line prior to employing the MC because sufficient processing power must be available to determine their size and position. The maximum number of hyper-cubes available for testing is defined by the MC's hardware. The whole process is an optimization task of maximizing the hyper-cubes while keeping the error to a minimum. Figure 3 shows one possible partitioning of the learning samples using three hyper-cubes $H_{1\text{-}3}$ covering *all* learning samples in *L*; the dashed regions inside *Hi* yet outside *L* are this model's error.

Constructing the hyper-cubes

The construction of hyper-cubes is based solely on the learning set. Unfortunately the learning set does not include all possible signals of the control cell. Even more, it does not contain a single *invalid* signal. It is impossible to collect all valid/invalid signals in advance or else the fault-detection itself would be unnecessary.

Invalid points are the only measure of error when creating hyper-cubes inside the bounding hyper-cube. If the algorithm is not given a human-drawn clustering region, it has to establish the cluster(s) by itself. The trivial hyper-cube which includes all points in *L* is the bounding hyper-cube itself. If the criteria are advanced to exclude invalid space, this space must first be recognized. In general it is impossible to assert whether a point in space is valid or not; if it is "close enough" to any existing valid point it could be valid, too. The question is how far is close enough.

First solution is to optimize for the *smallest* sub-space which includes whole *L*. Effectively, this method will create several *positive* hyper-cubes that together constitute a positive sub-space $V^+ = \cap H_i$ inside the bounding cube (see Fig. 3).

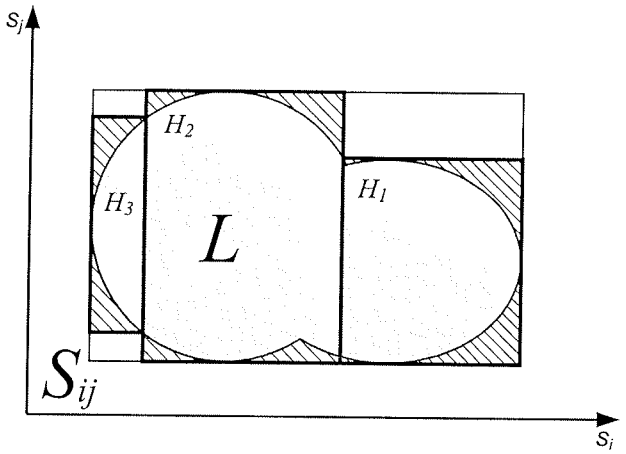

Fig. 3. One possible partitioning of the search space.

Second solution is the inverse of the first: *negative* hyper-cubes inside the bounding hyper-cube yet without common points with *L* are created. This scenario is shown in Fig. 4, where the three hyper-cubes X_{123} are positioned at the corners of the bounding hyper-cube to constitute the negative hyper-space *V*.

Any signal instance out of the bounding cube is always classified invalid. Also invalid are all points inside *V*, all other points *can* be valid. The optimization goal of both V^+ and *V* is to minimize the errors made by the clustering model.

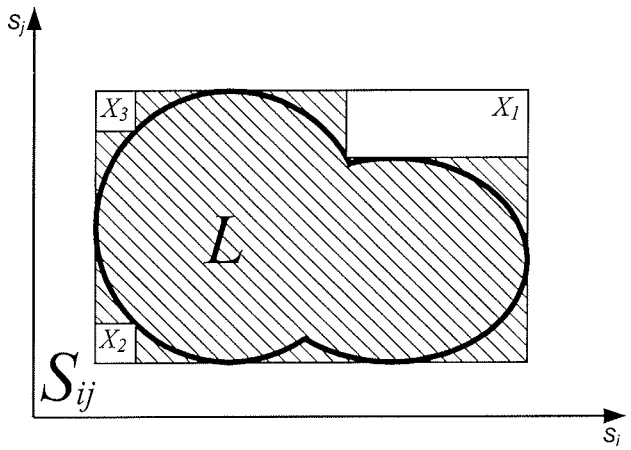

Fig. 4. Learning set fully outside of the negative hyper-space V⁻=)"Xi.

This single criterion optimization can be solved using a number of ML techniques, for example with evolutionary algorithms.

Evolutionary algorithms

The process of biological evolution by natural selection can be viewed as procedure for finding better solutions to some externally imposed problem of fitness[2]. Given a set of solutions (the initial population of individuals), selection reduces that set according to fitness, so that solutions with higher fitness are over-represented. A new population of solutions is then generated based on variations (mutation) and combinations (recombination) of the reduced population. Sometimes the new population will contain better solutions than the original. When this sequence of evaluation, selection, and recombination is repeated many times, the set of solutions (the population) will generally evolve toward greater fitness /2/.

Usually the evolutionary algorithms can be outperformed (EAs are rather slow) by the field-specific algorithms. For the purpose of optimization, however, EAs are excellent. Similar EA techniques differ in the implementation details and the nature of the particular problem. To optimize the MC's hyper-cubes all EA techniques are applicable; the most straightforward are genetic algorithms (GA, /3/) and differential evolution (DE, /6/).

For the negative model V^-, EAs need a fitness function that favors larger hyper-spaces V^- without valid signals. The simplest raw fitness function (5) divides the size of the hyper-space $|V^-|$ with the error $\epsilon(V^-)$, which is simply the count[3] of signals inside V^-.

$$f(V^-) = \frac{|V^-|}{\varepsilon(V^-)+1} \qquad (5)$$

For the positive model V^+, however, the fitness function must favor smaller V^+ and penalize any (valid) signals outside V^+. Again, higher values are better.

$$f(V^+) = \frac{|S|-|V^+|}{|L|-\varepsilon(V^+)+1} \qquad (6)$$

Example

An example result is shown in figure 5. For the simplicity only two variables (an input and an output) are used. The observed control cell has non-linear logarithmic characteristic. The learning algorithm was differential evolution (DE) employing positive fitness model V^+ (6), 30.000 iterations and default DE values for other settings. DE was used off-line to produce four hyper-cubes that included 42 distinct two-dimensional signals with a 12 bit resolution. In this example the depicted positive model V^+ is better than the V^- model because the points are relatively well aligned.

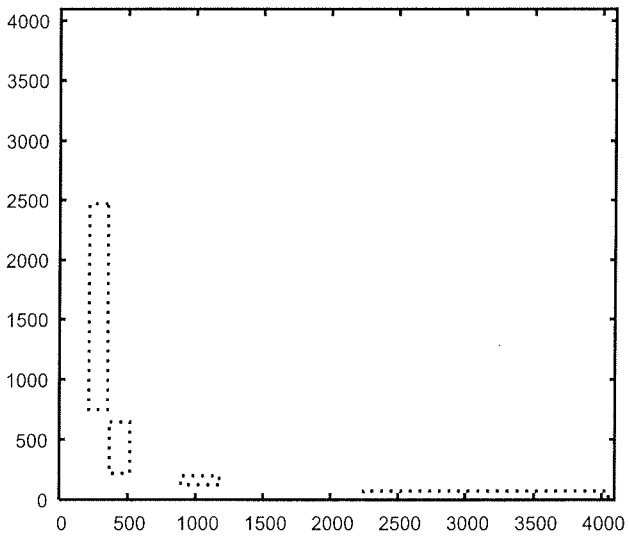

Fig. 5. A positive model of four hyper-cubes for 2 non-linear signals.

Implementations of the MC

MC can be implemented in several ways. If original application code is available and can be modified, the MC can be implemented as a set of monitoring routines that run together with the control software on the same processing resources. After the input is acquired, the pre-evaluation routine is called. It evaluates the individual signals, quantifies them and stores them for later assessment. Similarly, post-evaluation routine is called before output is produced.

2 *An equivalent term utility can be used in place of fitness.*

3 *∈ (X) is the cardinal number |F| of set F, where F={s; s ∈ L ∧ s ∈ X}*

Additionally it also checks for proper mappings between inputs and outputs. The main disadvantage of this approach is that modifying the original code may distort the temporal characteristic of the system. Also changing of the original application is rarely possible. Because of this, this approach was not taken in the research.

Higher degree of dependability and agility can be achieved by using dedicated hardware solutions. Continuous reduction of prices for the hardware makes this approach economically feasible. The monitoring device should run in two different modes of operation. In the first mode it measures and records the input and output signals. These measurements are consequently used as a learning base for the off-line construction of the monitoring function. In the second mode of operation it actually monitors the control system. By using the same device for both sampling and evaluation any differences between measurement and monitoring hardware can be eliminated.

The conceptual diagram of MC monitoring hardware implementation is shown in Figure 6. Signals from the control system ***s*** are connected to a set of registers (implemented as two-stage FIFO buffers) that hold their current value ***s(t)*** and the previous values ***s(t-1)***. Analogue and numeric values are processed by the quantization blocks (labeled Q in the Figure 6). Each quantization block transforms the input into corresponding discrete value in order to simplify the evaluation. These transformations are determined by the off-line learning phase (see 4). Signals ***s*** may also contain some internal states ***m*** of the control system. For the hardware implementation of MC, however, special access points must provide current values of internal states.

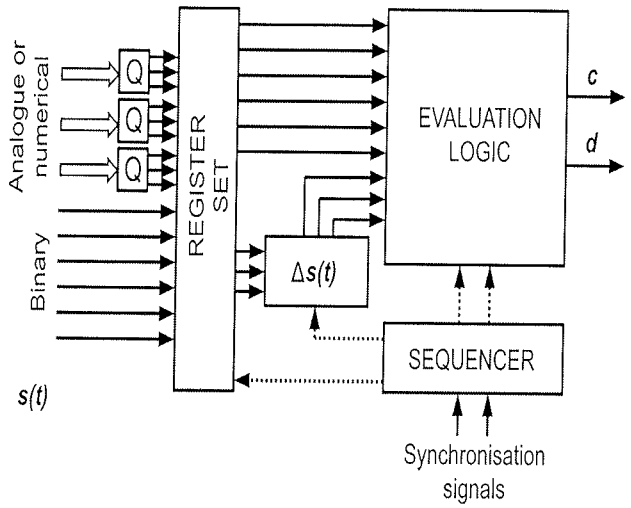

Fig. 6. Implementation of the monitoring cell.

To observe the dynamics of the signal, the block $\Delta s_i(t)$ compares new and previous values and calculates the difference producing additional information for the evaluation.

Parallel to the execution of the monitoring function certain physical characteristics of the control cell hardware can be observed allowing for detection of any abnormalities (e.g. increased temperature or current consumption). It is also possible to validate temporal properties of the observed process using simple watchdog timers.

Components of the MC are synchronized with a simple sequencer according to external synchronization signals that indicate the beginning of each execution cycle. If no such explicit signal exists, it can be derived from other signals (e.g., from control lines of I/O devices). Signals like reset, power-save, etc. should also be considered to determine the system's current mode of operation – the monitoring logic is not applicable in all modes.

The outputs *c* and *d* are generated by the evaluation logic. Mostly, *c* is just a status that alerts for the (potential) fault in the system. In most cases *c* can be determined by using pure digital logic. For diagnostic output *d*, which gives more detailed information about the cause of the fault, more complex logic may be required.

Implementation of MC with discrete components

In the fist case study, a solution with discrete analogue devices, FPGA chip and a simple microcontroller is explained. For the A/D conversion 12 bit A/D converters ADS7841 with serial communication interface were used. The FPGA was Spartan-IIe with 3072 programmable slices and 8 KB of dedicated memory blocks. This hardware was used in other experiments /8/.

The monitoring logic for one of the analogue signals is shown in Figure 7.

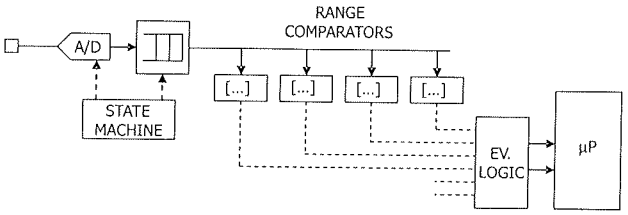

Fig. 7. Discrete monitoring logic for a single analogue signal.

First, the analogue signal is acquired by A/D converter and captured using simple state machine automaton. The state machine periodically generates control signals that trigger A/D conversion and, after conversion is done, reads the result. By utilizing FPGA device, it was possible to implement a number of state machines that are working in parallel, each serving a single A/D interface. To limit the number of wires needed for the communication, the A/D converters with serial communication protocol were used. From the 12 bits provided by the A/D device only the upper six were used. Similarly, the discrete digital signals are acquired periodically by other state machines.

To compensate for the delays between input and outputs, and to evaluate the dynamic characteristic of the signals, each sample goes through a simple two-stage FIFO buffer. Later, both stages are available for the evaluation.

The basic evaluation is performed by a set of range comparators, each testing if the input value is within the predetermined range. The results from this and other evaluation channels are then combined with simple Boolean evaluation logic, which is implemented as a truth table inside the dedicated memory blocks of the FPGA – the status signals are interpreted as a memory address containing the appropriate output. The output *c* states if all signals are in the valid ranges; output *d* is a vector designating validity of individual inputs. All constants for the range comparators and the content of the memory blocks are generated off-line and may be changed only by the full reprogramming of the FPGA device. For the implementation of evaluation logic for a single evaluation channel with a four range comparators approximately 150 slices are used.

For the communication with the fault management system, a simple 8 bit microcontroller is used. It is also needed for the initialization of the MC at start-up, for initial data acquisition, and later for the debugging and diagnostic. It is possible to transfer the microcontroller into the FPGA; however, some functionality (e.g. enhanced debugging) is lost.

Implementation of MC with programmable SoC

The above solution requires separate analogue circuits for the analogue signal manipulation. Nowadays, more compact and low cost novel technology with programmable system-on-chip is available. Those chips integrate a microcontroller and configurable blocks of analogue and digital logic. In the case study the PSoC CY8C29466 Mixed Signal Array is used. PSoC is a trademark for a family of programmable SoC devices from Cypres. PSoC devices include configurable blocks of analogue and digital logic, as well as programmable interconnects. Additionally, a fast CPU, Flash program memory, SRAM data memory, and configurable I/O ports are included. The analogue part is composed of dozen configurable blocks, each allowing the creation of complex analogue functions like A/D and D/A converters, comparators, filters, amplifiers, etc. More complex functions are implemented by combining several primitive cells. The digital part is composed of several digital blocks. Each block is an 8-bit resource that can be used alone or combined with other blocks to form 8, 16, 24, and 32-bit peripherals. The capabilities of those blocks are greater than its counterpart in typical FPGA device and may be configured as counters and timers, PWMs, different serial communication interfaces etc. CPU has full control over the configuration of the analogue and digital blocks.

The conceptual diagram of the MC evaluation logic with PSoC is shown in Figure 8. Each channel consists of a Programmable Gain Amplifier (PGA) and a six bit Successive Approximation Register (SAR) A/D converter. The PGA allows for adapting to different signal levels. This way low-voltage signals can be observed. To observe quantities larger than 5V, an off-chip voltage divider is required. By using the six bit A/D converter, the quantification was included in the conversion. Other possible implementations of A/D conversion with the PSoC device exists, however, they are either slower or consume more analogue cells. The hardware provides four analogue data acquisition channels that can be expanded to eight by using the also provided two-way analogue multiplexers. The digital part of the device allows up to sixteen bit of discrete data acquisition. Although it was possible to implement buffering and preliminary signal evaluation with digital cells on the device, they were implemented in software because of various limitations of the digital part.

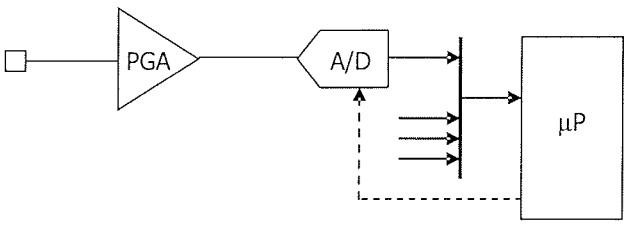

Fig. 8. Logical organization of a single analogue acquisition channel with PSoC.

The evaluation logic is executed by the microprocessor. It is implemented as a series of tests that check the inclusion of a variable in hyper-cubes. The hyper-cubes are prepared off-line and are loaded to the device during the initialization phase. The solution used by FPGA (i.e., the implementation with truth tables in memory blocks) would use more memory than is available.

Comparison of different implementations of MC

The solution with discrete analogue devices is somewhat more flexible and robust. It can use different kinds of A/D converters to accommodate various kinds of signals. In addition, the external A/D converters are usually much less sensitive to the voltage overloads. The main benefit of using this approach is the speed because multiple monitoring channels and evaluation logic in the same device can be constructed. The sampling, quantization and evaluation for different signals occur in parallel. If the evaluation logic is simple enough, the execution cycle can be in range of several micro-seconds. This is much shorter than the time needed for the A/D conversion. Therefore, it is possible to evaluate the signals from one execution cycle during the acquisition of the signals from the next one. The drawback of this solution is the price. The estimated price for the parts used in the experiments is more than 30 euros and it increases with each additional A/D converter. In contrast the programmable SoC solution can cost (for up to the 4 analogue channels) less than 5 euros.

The solution with programmable SoC devices is much more compact and requires less supporting components than the previous one. However, apart from the A/D conversion, all of the processing of data is performed with the microprocessor. This impacts the execution time because all evaluations must be done sequentially. For example, for the configuration where two analogue signals were observed and evaluated with four regions (like in the figure 6), the execution time of the evaluation was 83 microseconds (with 24MHz system clock). On the other hand the typical A/D conversion takes only 25 microseconds. Therefore the evaluation can not be performed in parallel with the conversion. If more dimensions are needed, the execution time increases accordingly. This is in contrast to the first approach where additional variable have almost no impact to the execution time due the parallel nature of the execution.

The case studies use only simple evaluation functions although more complex regions than the hyper-cubes could be used that require higher mathematical operations. The FPGA is unsuitable for such calculations, the PSoC however, is powerful enough to implement them (e.g., it has two dedicated fast 8 by 8 bit multipliers).

Conclusions and future work

In the paper a concept of monitoring cell for supervision of the plausibility of a control function is presented. Knowing (or having learned) the normal behavior of the control function, one can detect abnormal behavior of input and output signals, internal states, dynamics, and coarse behavior of the output function with respect to the inputs in the previous time instance. For situations where the control function is not known in details, the automatic process based on the machine-learning principles is proposed. The samples of signals acquired from the longer period of time are analyzed and categorized into a set of multi dimensional regions. Two different solutions for MC implementation are described. The proposed devices can be relatively easily applied in variety of existing control systems. The first solution is appropriate when short response times are required. Due to the multiprocessing nature of evaluation logic used, the execution time is almost independent of the number of signals observed. However, the solution is relatively expensive, harder to implement and can be used only with simple evaluation functions. The approach with programmable SoC devices is more compact, simpler to construct and more appropriate for more sophisticated evaluation functions. However, because the main processing is done in software, the reaction times are much longer. The ideal solution would be the combination of both approaches: programmable mixed signal SoC device with large FPGA on a single chip. To our knowledge, there is no commercially available device of such kind on the market yet.

In the future work, other classification models will be considered, possibly more powerful yet of similar complexity as the hyper-cubes. Part of the research is focused on utilizing genetic programming (GP) because it is capable of directly producing the control function of the monitoring cell. The idea is to find appropriate set of functions that are easy enough to be implemented on simple hardware and yet good enough to appropriately describe the valid control signals. The other part of the research is focused on the implementation of those functions. The final goal is to include a machine-learning algorithm inside the monitoring device where evaluation function can be determined dynamically and might adapt to different operation modes of the system.

References

/1/ L.H. Chiang, E.L. Russell, and R.D.Braatz. Fault Detection and Diagnosis in Industrial Systems. Springer, 2001.

/2/ D.W. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. Physica D, 42:228-234, 1990.

/3/ J.H. Holland. Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor, MI, 1975.

/4/ 1149.1 IEEE. Test Access Port & Boundary Scan Architecture. IEEE, New York, 1990.

/5/ R. Isermann. Fault Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance. Springer, 2005.

/6/ K. Price and R. Storn. Differential evolution: A simple evolution strategy for fast optimization. Dr. Dobbs Journal of Software Tools, 22(4):18-24, 1997.

/7/ Neil Storey. Safety-Critical Computer Systems. Addison-Wesley Longman, 1996.

/8/ D. Verber, B. Lent, and W.A. Halang. Firmware support for disjunctive dataflow driven distributed control applications. In Z. Bradač, F. Zezulka, M. Polanski, and V. Jirsik, editors, Proceedings of IFAC workshop on programmable devices and embedded systems PDeS 2006, pages 84-89, 2006.

Domen Verber, Matej Šprogar, Matjaž Colnarič
University of Maribor Faculty of Electrical Engineering and Computer Science, Smetanova 17, 2000 Maribor, Slovenia
domen.verber@uni-mb.si

Prispelo (Arrived): 10.08.2006 *Sprejeto (Accepted):* 30.03.2007