CONTROL FLOW MONITORING FOR A TIME-TRIGGERED COMMUNICATION CONTROLLER

Thomas M. Galla[1], Michael Sprachmann[2], Andreas Steininger[1] and Christopher Temple[1]

Abstract

A novel control flow monitoring scheme is presented that has been tailored to the architecture of the protocol control unit of a time-triggered communication system. Within the approach a signature derived on-line for a sequence of instructions ("block") is checked against an embedded reference signature. The execution of the signature checks is enforced by a timeout mechanism that is based on the instruction count between two successive checks.

The approach provides a derivable Hamming distance for the signatures and a bounded error detection latency while being transparent for the protocol execution. The overhead in terms of performance penalty, memory overhead and extra hardware is low.

1. Introduction

Efficient error detection is of fundamental importance in dependable computing systems. Among the numerous approaches for error detection control flow checking is a well established cost-efficient technique. Various types of control flow monitors based on watchdog processors have been devised to ensure the integrity of control flow. [1] provides a survey of the basic approaches. Within control flow monitoring the instruction stream is partitioned into *basic blocks*. A basic block consists of a sequence of consecutive instructions with an unique entry point and an unique exit point. In most approaches each block is assigned an identifier that is determined by some of its properties, like start address and block size as in [2], or a signature of the instruction sequence within the block [8]. A watchdog processor not only observes the correct signature (or size, respectively) of the block, but most importantly it checks the sequence in which different blocks are executed. For this purpose information describing allowed control flow paths must be provided. In most approaches this information is embedded in the instruction stream, there are, however, various strategies to minimize the resulting memory and performance overhead [5, 7].

2. System Model

In the course of the Esprit OMI Project "Time-Triggered Architecture TTA" [4] a single chip communication controller has been developed for TTA. The time-triggered architecture is de-

[1]Institut für Technische Informatik, Technische Universität Wien, Treitlstr. 3/182/1, A-1040 Wien, Austria
[2]Dependable Computer Systems KEG, Mariahilferstr. 117/1/14, A-1060 Wien, Austria

signed to support a wide range of fault-tolerant distributed real-time systems for use in safety critical dependable applications, especially within the fields of automotive, aviation and railway electronics. Thus the demand for efficient error detection capabilities arises. Within the time-triggered architecture each node of a distributed system is composed of a host computer that executes the assigned share of a distributed application and of a communication controller that is responsible for the autonomous exchange of messages among the different nodes. In order to accomplish this task the communication controller executes a round based communication protocol.

The *communication controller* is a key component of the time-triggered architecture. It interacts with the host computer via a shared memory holding messages and status/control information. Apart from the shared memory the communication controller contains the *protocol control unit* (PCU) and a set of low-level functional units. The protocol control unit executes the communication protocol while the functional units provide means for executing performance critical protocol services, like frame transmission and reception, CRC calculation, and clock synchronization. The protocol control unit is implemented as a 16-bit application-specific instruction set processor.

For the proposed technique we assume a set of faults, both transient and permanent, consisting of storage faults within the instruction memory, faults in the address path from the decoder to the instruction memory and faults in the data path from the instruction memory to the decoder.

The *fault model* imposes two basic requirements on the control flow monitoring. On the one hand the *integrity of the basic blocks* must be established and on the other hand the *integrity of the execution flow* must be checked. Apart from these two basic requirements an additional set of application specific requirements must be considered. The execution time penalty caused by control flow checking must be minimal and known at design time since the system is intended to serve hard real-time applications. Due to the limited size of the instruction memory the code size overhead must be kept low. To meet the requirements of fault-tolerance it is important to have a low bound on the error detection latency, in particular the case that a fault masks all embedded check instructions must be covered. To meet the demands of safety critical applications the code used to ensure the integrity of the blocks should have a derivable hamming distance.

3. Principle of Operation

The presented approach is based on integrating a signature calculation circuit into the processor pipeline of the protocol control unit to test the *integrity of the basic blocks* and to *monitor the control flow* between basic blocks.

3.1. Integrity Checking and Control Flow Monitoring

The integrity of the basic blocks is checked by assigning signatures to each instruction. In general the signature for an instruction is generated by feeding the instruction through the signature calculation circuit where it is processed along with the signature value from the previous instruction to obtain the new signature. In this way the signature of a block is an

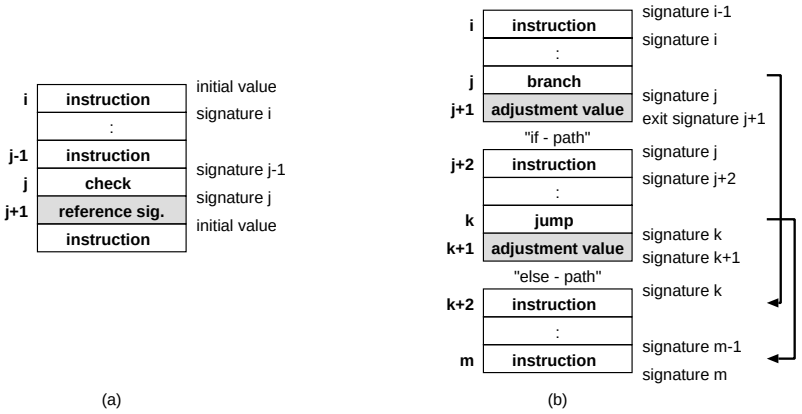

Figure 1. *Signature Checking and Adjustment*

incrementally generated value that protects all instructions within the block. The signatures are calculated by the assembler at compile time and selected signatures are then stored within the code. At run-time the the signatures are recalculated as the instructions are fed through the protocol control unit. A special *check-instruction* triggers checking the signature. The code word subsequent to the check-instruction contains the reference signature calculated by the assembler for the preceding block (Figure 1a). The check-instruction resets the value of the signature to its initial value.

In order to handle *branches* it is necessary to modify the signature according to the branch destination. This modification is performed based on an additional *adjustment-value* included in the program code immediately after the branch instruction. This value is used to modify the signature so that it matches the entry signature at the destination instruction. The adjustment value is only considered if the branch is taken, if the branch is not taken the adjustment value is skipped. Figure 1b depicts the signature adjustment for an if-then-else construct. The adjustment value at address j+1 is set to adjust the signature to equal signature k. This procedure is repeated for the adjustment value at address k+1. In this case the signature is adjusted to equal signature m-1.

For subroutines the signature of the first instruction is set to a specific *entry signature* of the subroutine and at the end of a subroutine the signature is adjusted to a specific *exit signature*. Each subroutine is assigned an unique entry signature in order to establish whether the program flow has continued with the correct subroutine. Similar to the assignment of entry signatures it makes sense to chose a unique exit signature for each subroutine in order to verify the correct return from the subroutine to the calling code. Figure 2 illustrates the use of the entry and

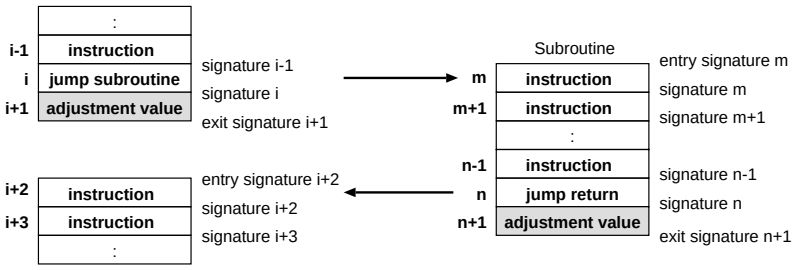

Figure 2. *Subroutine Call with Entry and Exit Signatures*

exit signatures for subroutines. As in the previous example the adjustment value at address i+1 causes the exit signature i+1 to equal the entry signature m of the subroutine. The same procedure is used at the end of the subroutine to match the exit signature n+1 to the entry signature i+2.

3.2. Protocol Control Unit

The protocol control unit is a pipelined instruction set processor with an accumulator based arithmetic logic unit. Instructions are 16 bit wide and are executed in a 3-stage pipeline. The protocol software is located in the instruction memory from where the instructions are fetched by the "Instruction Fetch"-stage. The "Decode/Move/Branch and Check"-stage decodes the

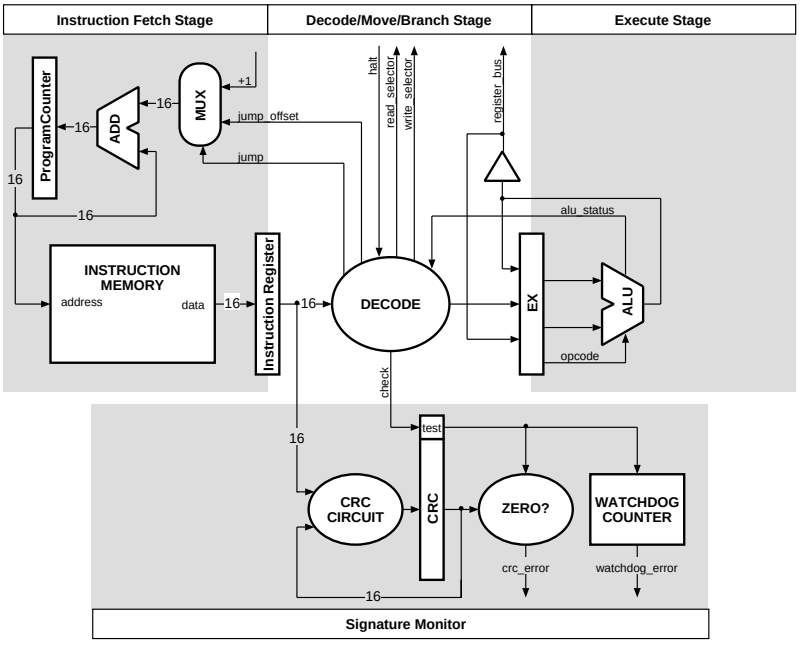

Figure 3. *Protocol Control Unit with Signature Monitor*

instructions, issues register transfers on the internal register bus, and resolves branches. According to the opcode the "Execute"-stage then executes ALU instructions. Due to the architecture of the pipeline, no data hazards can occur. The PCU is master of the synchronous register bus, which interconnects the PCU with the low-level functional units. The processor supports three

types of instructions: ALU instructions issue ALU operations either on registers or immediate values, move instructions issue register transfers on the register bus, and branch and jump instructions control the program flow. To provide subroutine calls, the program counter can be stored and reloaded by issuing appropriate move instructions.

3.2.1. Signature Calculation

Control flow monitoring is facilitated by adding a signature calculation circuit to the PCU (Figure 3). In contrast to MISR based approaches a CRC polynomial is used here to perform signature calculation, which enables easier determination of the code properties [3]. The size of each block is limited by the *hamming distance* of the CRC polynomial and the desired *error detection latency*.

The hardware implementation supports both, signature adjustment and signature check (implying signature reset). The block signature detects errors in the instruction memory and the related address and data paths from and to the instruction register. Using signature adjustment for branch and jump instructions extends the effectiveness of the signatures beyond the block borders and thus allows to detect errors in the address calculation logic (MUX and ADD in Figure 3) as well. To support transparent and continuous operation of the pipeline the CRC calculation has to be performed within one instruction cycle which equals one clock cycle. Traditionally CRC calculators are based on the linear feedback shift register (LFSR) approach which is well suited for sequential bit-stream processing. Within the PCU a parallel CRC calculation technique is employed that is capable of processing 16 bits per clock cycle ([6]). Apart from the CRC calculation circuit the signature monitor contains a register for storing the intermediate signature values and a circuit for comparing the signature to the initial value, which is chosen as zero in the current implementation.

Every code word that passes to the decoder is fed into the signature monitor, i.e. the CRC calculation circuit. If the decoder encounters a check instruction it raises a check signal to the signature monitor. In addition the decoder ignores the next code word in the instruction register, which contains the reference signature, and generates a stall cycle. The signature monitor processes the reference signature and tests for zero. A similar procedure is followed for jump and branch instructions. The decoder always generates a stall cycle following a branch or jump instruction. Again this prevents processing of the adjustment value. For branches taken and for jumps the stall cycle is necessary, since the target address for these instructions is computed in stage two of the pipeline. Thus no run-time overhead is incurred as the adjustment value is fed to the signature monitor. If the branch is not taken then the stall cycle introduces a delay of one cycle (to skip the adjustment value). In this case the signature monitor ignores the code word.

Due to the nature of the time-triggered architecture the communication controller does not process any interrupts. Thus saving the signature within the signature monitor is not necessary, which simplifies the design of the watchdog processor. Adding the signature calculation circuit

to the PCU increases the silicon area on the chip die by approximately 5%. Signature calculation is performed within 5ns. Since the signature calculation circuit is not on the critical path, the self-checking hardware does not influence the processor cycle time.

3.2.2. Watchdog Counter

The CRC signatures provide a means for checking the code integrity and for monitoring the control flow. However a failure could mask the opcode of the check-instructions and thus prevent their execution. Hence a mechanism must ensure that the check-instructions are executed by the protocol control unit at run-time. For this purpose a watchdog counter is added to the protocol control unit. It is set to a predefined value[3] every time a check-instruction is executed and decremented by one whenever an instruction other than the check-instruction is carried out. When this counter reaches zero, it is assumed that a failure has occurred, and the processor pipeline is halted. Through this mechanism a bounded error detection latency is achieved even if the execution of the check-instructions is prevented.

4. Summary

The presented control flow monitoring scheme provides several vital properties for our fault-tolerant real-time application: The simple structure requires only low hardware overhead (only 5% of a small processor core) and results in an efficient implementation. An additional watchdog counter guarantees bounded error detection latency even if no more check instructions are executed. The use of a parallel CRC allows comparatively easy mathematical determination of the hamming distance. Runtime penalty is minimized by the placement of adjustment values into the idle slots after branch and jump instructions.

References

[1] A. Mahmood and E. J. McCluskey. Concurrent Error Detection Using Watchdog Processors – A Survey. *IEEE Transactions on Computers*, 37(2):160–174, Feb. 1988.

[2] G. Miremadi, J. Ohlsson, M. Rimen, and J. Karlsson. Use of Time and Address Signatures for Control Flow Checking. In *Dependable Computing and Fault-Tolerant Systems*, volume 10, Urbana Champaign, Illinois, USA, Sep. 1995.

[3] N. Saxena and E. J. McCluskey. Parallel Signature Analysis Design with Bounds on Aliasing. *IEEE Transactions on Computers*, 46(4):425–438, Apr. 1997.

[4] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple. Time-Triggered Architecture (TTA). In J.-Y. Roger, B. Stanford Smith, and P. T. Kidd, editors, *Proceedings of EMMSEC'97, Advances in Information Technologies: The Business Challenge*, pages 758–765. IOS Press, Nov. 1997.

[5] M. Schuette and J. P. Shen. Processor Control Flow Monitoring Using Signatured Instruction Streams. *IEEE Transactions on Computers*, 36(3):264–276, Mar. 1987.

[6] M. Sprachmann. Automatic Generation of Parallel CRC Circuits. *IEEE Design and Test*, submitted 1999.

[7] K. D. Wilken and T. Kong. Concurrent Detection of Software and Hardware Data-Access Faults. *IEEE Transactions on Computers*, 46(4):412–424, Apr. 1997.

[8] K. D. Wilken and J. P. Shen. Concurrent Error Detection using Signature Monitoring and Encryption. In *Dependable Computing and Fault-Tolerant Systems*, volume 4, pages 365–384, 1991.

[3] Similar to the insertion of the check-instruction this value depends on the hamming distance of the used CRC and on the desired error detection latency.