www.ietdl.org

Published in IET Software
Received on 26th November 2011
Revised on 27th August 2012
Accepted on 5th October 2012
doi: 10.1049/iet-sen.2011.0212

ISSN 1751-8806

On-chip support for software verification and debug in multi-core embedded systems

Padraig Fogarty, Ciaran MacNamee, Donal Heffernan

Department of Electronic and Computer Engineering, University of Limerick, Limerick, Ireland
E-mail: padraig.fogarty@ul.ie

Abstract: The challenges in silicon testing and debug of complex integrated circuits are well understood. Where these circuits include multiple processor cores there is also a dramatic increase in the complexity of verifying and debugging the associated software; with much of this complexity being because of the inherent lack of visibility over internal signals which integration brings. The trend to-date has been to rely upon silicon test interfaces to provide access to internal signals required for software verification and debug. However, it is questionable whether this is sufficient for real-time systems or future designs with increasing processor cores. This study examines the on-chip technology supporting software verification and debug in current designs and proposes enhancements in this area. As much of this technology is primarily intended for silicon test it is lacking in terms of I/O bandwidth, which is a significant limitation for software verification and debug. The authors propose their alternative approach of using an on-chip coprocessor and debug circuitry to address this principal limitation; and describe an embedded application where this approach was successfully applied to monitor timing requirements and detect failures. The authors also outline how this approach could be applied as an architectural solution for formal runtime verification.

1 Introduction

The dramatic increase in system integration and complexity has compounded debugging and testing problems. Problems in manufacturing test are well-documented [1–3] and are being addressed through design-for-test techniques and system-on-chip (SoC) test disciplines, such as IEEE 1500 [4], IEEE 1149.7 [5] and IEEE P1687 [6]. Progress is being made in coordination among the various groups working on standardisation of on-chip and external test interfaces [7].

Post-silicon validation is performed exhaustively on 'first silicon' parts to ensure that the integrated circuit (IC) meets its design requirements. The operation of the IC is checked by capturing data from a physical device over a range of operating conditions, and comparing against simulation data gathered during earlier design phases. Once validation is completed the associated hardware may be removed to reduce costs. The distinction with software verification is that software can be updated, requiring further verification; therefore the final packaged device must include the necessary interfaces and instruments.

Every IC produced is tested to identify those with manufacturing defects. This involves stimulating the device with test patterns and checking the results against expected values. As silicon test must be carried out on each device, the on-chip test hardware and interfaces are generally accessible on the packaged device. This makes reuse of the test interface an attractive proposition, but it does not offer the bandwidth required for runtime software verification. Hopkins and McDonald-Maier [8] show how interfaces primarily designed for silicon test and verification have been used for software purposes, but also highlight the limitations of this approach when considering real-time systems.

When available, simulation environments are useful for initial software verification and debugging; where visibility of the internal state of the simulation models is a significant benefit. When cycle accurate simulation is too slow, field programmable gate array (FPGA)-based prototypes or emulation systems can accelerate the process. Chuang *et al.* [9] describe seeding a simulation environment with snapshot data acquired from an FPGA platform, thereby improving both speed and visibility. Unfortunately, many 'bugs' do not appear in simulation; this applies in particular to timing related issues such as race conditions. The inherent complexity of real-time systems and expansion of multi-core designs further compounds verification difficulties [10].

1.1 Problem background and definition

The software engineering research community offers formal theories, based in sound semantic models that lead to verifiable designs or correct-by-construction designs. Program verification determines whether a program satisfies a specification. However, program verification is unsolvable in general, and theorem prover solutions do not in practice scale to large embedded software systems. In spite of shortcomings, formal methods research has huge potential and continues to receive much attention with an aim to prove a design to be correct; so that, verifiable commercial products might be built without the need for complex development and exhaustive testing. With increasing complexities new approaches are needed. Burns and Hayes [11] state that new scientific foundations are now required for specifying, designing and implementing complex real-time systems.

For this field, with the exception of a few well-publicised examples, formal methods have not yet met expectations for the development of commercial products. Some major research initiatives are showing potential; for example, DEPLOY, a European Commission FP7 project, aims to make major advances in engineering methods for dependable systems. The DEPLOY Project has several industrial partners using Event-B and Rodin [12] on deployment projects for real embedded products. Commercial development tools are already emerging. ADVANCE is another FP7 project where the objective is to develop a unified tool-based framework for automated formal verification and simulation-based validation.

Although such developments are encouraging, research towards solutions for correct software designs need to be complemented with solutions that can monitor the exact program behaviour, non-intrusively, for highly integrated and multi-core embedded systems. Even if a complex software system can be 'believed' to be correct, confidence in the total hardware/software product combination is paramount; therefore accurate monitoring and testing of its behaviour in its target environment is still necessary. The execution of real-time application software can be affected by system issues such as unpredicted hardware performance, real-time operating system (RTOS) performance, I/O interrupt behaviour and interference from other software applications on the system. The authors propose that improved in-situ on-chip development instrumentation is needed to assess proper program behaviour.

To perform in-situ software verification, validation, development and debug, it is imperative that engineers have adequate tools. There is significant research into on-chip instrumentation (OCI) architectures. Solutions exist for on-chip trigger and trace infrastructures, using high-bandwidth interfaces and on-chip buffers to capture data at very high rates. However, current and emerging solutions have limitations because of available bandwidths and available on-chip hardware, which restrict their capabilities for software monitoring and debug.

According to Min-Allah *et al.* [13] further complications arise in the development of real-time systems, where conflicting issues arise in the design process; the most fundamental issue being the adjustment of task parameters for scheduling that directly influence the system feasibility, as task computation times can fluctuate. The authors' proposal to embed runtime reprogrammable instrumentation on to the chip has the potential to monitor and even correct such behaviour.

Higher energy consumption and associated heat issues are an emerging problem as real-time systems become more computationally powerful to cope with processor intensive applications. Proposed solutions involve adjusting the system speed on-the-fly, so that application deadlines are respected. However, the multi-core front is relatively unexplored from the perspective of task scheduling [14]. An on-chip instrument could potentially monitor such complex on-the-fly changes in processor operations.

1.2 Research question

The fundamental research question is to ask is there a better architectural solution for on-chip software monitoring and debug instrumentation. The purpose of this paper is to review some of the more important emerging solutions for on-chip monitoring and debug instrumentation that support real-time, embedded software development, and to propose an enhanced OCI solution that is based on the use of a coprocessor that can be configured at run-time, and has the ability to analyse, on-chip, the trace or other captured data and to optimise the use of on-chip resources in a non-intrusive or in a minimally invasive fashion.

Section 2 provides a general overview to embedded software debugging and some challenges when considering real-time and multi-core devices. Sections 3–5 provide a more detailed review of the technologies currently employed. In Section 6, the authors propose and demonstrate the use of a coprocessor to assist verification and debug in a multi-core system.

2 Embedded software debugging

Software verification is the process of ensuring that the software meets its design requirements; or as pointed out by Myers [15], it might be better considered as the task of finding and removing bugs. Debugging is not a well-defined discipline commanding its own literature; despite often taking more than 50% of a project time-scale. Even authoritative publications tend to be sets of guidelines and check-lists [15, 16]. Vermeulen [17] provides a comprehensive overview of recent 'functional' debug techniques. This paper identifies the primary challenges in designing on-chip debug support, not least the fact that it is impractical to extract the volume of data required to observe all on-chip activity in real time. Nonetheless, in real-time systems that cannot be halted to facilitate checking of the internal state, the engineer must attempt to find or reconstruct the problem from information captured while a system is running. This in turn means that huge quantities of data must be captured at full execution speed and analysed off-line.

Transferring large volumes of data off-chip for analysis creates a bandwidth problem. The industry-standard means of addressing this has been to reduce the volume of data to be captured and sent off-chip [18, 19]. Efforts to further reduce the volumes of data transmitted off-chip have led to the use of data compression techniques [20–23]. Taken along with the Nexus, ARM and related techniques from other manufacturers, these methods represent the state of the art in system tracing today. As Vermeulen [17] indicates, there is a need to better define the criteria which provide the best debug support for future architectures.

Multi-core designs bring increasing volumes of data to analyse, and new questions which need to be addressed, such as: if one core is halted, should others be halted? How to correlate data from multiple cores? How to preserve the relative timings of traced data? How to best accomplish data tracing across multiple clock domains? These questions, and more, greatly complicate the landscape for developing today's debugging tools.

3 Standard interfaces

The I/O interface, as the first layer between the external development environment and on-chip signals, is a critical link in the debug chain; and several standard I/O interfaces exist. The IEEE 1149.1 joint test action group (JTAG) standard [24] is the principal method for board-level interconnection testing of complex digital circuits. Although a board-level test standard, IEEE 1149.1 has been habitually modified to include debug capabilities. One significant limitation of IEEE 1149.1 is that it envisages only a single test access port (TAP) per IC. Vermeulen *et al.* [25] review several approaches to accommodate multiple cores and propose an architecture, which maintains compliance with the standard; however, they do not address the issue of limited I/O bandwidth.

The more recent IEEE 1149.7 standard [5] aims to address the limitations of 1149.1, while maintaining compatibility with it. In addition to allowing serial connection of multiple cores, the standard introduces two-wire and four-wire star configurations. As described by Ley [26] this standard is of particular benefit for SoC or system-in-package designs. Enhancements such as: reducing I/O to just two pins, using a star topology, the concept of 1149.7 being an 'adaptor' for existing 1149.1 TAPs, and the introduction of hierarchy, ease integration challenges. There are some improvements for debug (simplification of the architecture, reduced scan-chain lengths and allowing data transmission during idle states), but the critical limiting factor is still I/O bandwidth, which may be exacerbated by reduced pin count and interleaving of control and data signals onto pins.

IEEE-ISTO 5001-2003 (Nexus) [18] is the only standard developed to address the challenges in debugging real-time embedded control applications. The standard defines a development interface, a messaging protocol and a comprehensive set of features many of which are optional. The optional auxiliary port is the primary differentiator in Nexus as it directly addresses I/O bandwidth limitations; but this requires additional pins which are a precious resource in IC designs. Consequently, despite being developed by many leading silicon and tool vendors, Nexus has not been widely adopted. However, planned updates to include 1149.7 and high-speed serial ports may provide the stimulus for wider use.

For SoC designs the on-chip core wrappers and interfaces between cores are also critical for test and debug purposes. The preliminary IEEE P1687 standard eases access to the instruments which accompany many cores within SoC designs. Utilising IEEE 1149.1 to access the on-chip instruments, this standard aims to address the challenges of test interface efficiency and adaptability while maintaining compatibility with existing tools. The standard proposes the addition of gateway logic between the 1149.1 and P1687 zones, and the hierarchical connection of cores within the P1687 zone [6]. This hierarchical arrangement addresses the inefficiencies with existing daisy-chained or multiplexed 1149.1 access methods. However, initial analysis on a range of benchmark designs suggests test application time overhead of between 9 and 24% [27]. These results exclude cores with built-in self-test (BIST) capability; nonetheless they do highlight the overhead associated with P1687.

The IEEE 1500 standard [4] facilitates testing of independent cores within SoC designs. The standard defines core wrapper architecture, which is analogous to the use of IEEE 1149.1 for boundary scan testing of ICs. Higgins *et al.* [28] use this wrapper architecture to simultaneously test multiple cores; their paper outlines the use of on-chip test controllers and reuse of the on-chip bus for test purposes. Prior work of others such as Lee *et al.* [29], Huang *et al.* [30] and Krstic *et al.* [31], also highlight the benefits of using on-chip resources to implement test. These approaches are relevant for software verification and debug, as avoidance of the I/O bottleneck is a shared objective.

As described by Stollon *et al.* [32] the open core protocol (OCP) international partnership debug working group aims to define and standardise debug interface requirements for OCP compliant cores and proposes optional debug interface sockets for cores which require debug. Although their paper outlines desirable debug capabilities such as run control, trace and trigger, it explains that the implementation details for debug within each core is specific to that IP block and not mandated by OCP. Instead, OCP provides a framework for interconnection between cores, so that IP from various sources can be integrated. Similarly, the chip-level pin interface is not within the scope of OCP, instead the objective is to define a solution that is compatible with existing interfaces.

4 Proprietary OCI

OCI refers to integrated circuit blocks which provide the features required by modern debug environments. Stollon and Leatherman [33] categorise typical OCI under five headings: (i) logic analysis, (ii) run control and programme trace, (iii) embedded bus trace (iv) performance analysis and (v) system-level monitoring and synchronisation, which encompasses most verification and debug tool features. In SoC designs this OCI may be provided with each core or a customised chip-level OCI block may be developed.

ARM offers a suite of modules to provide debug and trace functionality for SoC designs. Orme [34] gives an overview of these 'CoreSight' modules and the key design decisions required. The 'debug access port' (DAP) and 'trace port interface unit' (TPIU) modules connect to external pins and thus impact upon bandwidth. The DAP provides 'access ports' to on-chip resources such as buses and JTAG scan chains, plus 'debug port' access to external tools via a two-pin serial or four-pin JTAG interface; if both interfaces are required these pins may be shared. The TPIU formats trace data into packets, and provides an interface for external tools which is scalable from 1 to 32 pins. Since these modules use I/O interfaces which are similar to other debug solutions so too are their bandwidth capabilities.

The ARM suite also includes modules to assist software-based instrumentation. By providing a memory mapped hardware module to which software can write instrumentation data, the latency and code-space problems associated with traditional 'printf' software instrumentation can be avoided. Nonetheless, the data captured must still be offloaded to external tools for analysis, which for real-time applications necessitates the use of a high-speed port.

MIPS enhanced JTAG (EJTAG) [35] provides on-chip debug capabilities for SoCs using MIPS processor cores. EJTAG augments, the IEEE 1149.1 standard to provide access to a range of on-chip features including: single-step execution, breakpoints, memory substitution and program-counter sampling. Debug of multi-threaded applications is supported within the specification, but multi-core debug requires duplication of the hardware, with the option to share a single interface.

Trace information from a MIPS processor core can be obtained by connecting a trace control block (TCB) to the processor's PDtrace interface [36]. The TCB compresses the data before storage in on-chip or off-chip memory. External tools can then access this trace data using the EJTAG interface. An optional parallel interface (probe IF) providing higher-bandwidth is also defined, but requires additional I/O pins. Therefore despite supporting multi-core SoCs this debug architecture provides no unique mechanism to deal with the accompanying volume of data.

Infineon promotes a slightly different approach to cater for OCI. Rather than adding instrumentation and memory within the SoC core area, these circuits are instead placed on the device perimeter or on separate die to which the SoC is bonded [37]. The key benefit of this approach is that removing these circuits at a later stage has no impact upon SoC behaviour; therefore silicon verification need not be repeated!

Mayer *et al.* [38] describe how the Infineon multi-core debug solution addresses the challenge of limited I/O bandwidth by focusing on trigger and trace qualification. By providing sophisticated data capture capabilities only a small amount of relevant data need be stored in on-chip memory. This smaller amount of data can then be transferred off-chip using a conventional JTAG or other low-bandwidth interface.

Leatherman and Stollon [39] propose a multi-core embedded debug (MED) architecture to address specific multi-core debug challenges. MED adds on-chip capabilities to deal with issues such as debug synchronisation, but the solution for limited I/O resources is to multiplex the data streams from various cores, using a scheme called HyperJTAG. Although this may simplify the I/O interface it does not address the limited bandwidth problem.

Bernardi *et al.* [40] proposed the addition of an infrastructure intellectual property (I-IP) block to assist in silicon debug and testing of SoCs. This I-IP block uses the SoC host processor to run software-based self-test (SBST) routines to test and debug the silicon. By restricting external data transfers to test commands and compressed test results, the bandwidth limitations of the interface can be overcome. However, executing SBST in the manner proposed takes over the host processor, which is not acceptable for real-time software verification.

A wide variety of proprietary OCI blocks are available, each offering its own particular features and benefits, but in general the problem of limited I/O bandwidth is addressed by: limiting the data captured, using faster I/O interfaces or using parallel interfaces requiring additional I/O pins.

5 Runtime monitors

To ensure the correct runtime operation of high-reliability systems, hardware or software monitors, which share many requirements with verification and debug instrumentation, are often added. Scottow *et al.* [41] demonstrate how an invasive software monitor can, in certain situations, extract performance related data with acceptable overhead. However, their paper also highlights the problems associated with software monitors, particularly in real-time systems where the overhead can impact upon determinism.

Watterson and Heffernan [42] demonstrate the use of the Java-Mac runtime verification system to provide a minimally invasive method of verifying a Java optimised processor embedded within an FPGA. This framework demonstrates the complexity in extracting the required data from an embedded system at runtime. It points towards a technique which could be applied to multi-core devices whereby the core under examination could be instrumented to a minimal level, thus maintaining determinism, while a secondary heavily instrumented core could replicate the execution behaviour.

Heffernan *et al.* [43] describe an on-chip SoC monitor which can be used for runtime verification. Two control applications demonstrate how the monitor can ensure that the application state transitions occur within prescribed time limits. For the solution proposed, the constraints to be verified are established prior to SoC synthesis and the appropriate circuitry is automatically generated. This results in a monitor with low software and hardware overheads, but it is specific to one set of constraints which limits its applicability to more general verification situations.

Larsson *et al.* [44] describe on-chip monitor hardware which is capable of detecting the occurrence of an incorrect read/write sequence, because of the presence of a race condition, when two CPUs access shared memories. The monitor described involves a bus interface, address/data matching circuits and a state machine, which can be reprogrammed in-circuit to monitor different potential race conditions.

For hybrid SoC designs (containing both hard and reconfigurable elements) Hopkins and McDonald-Maier [45] highlight how traditional techniques to monitor registers within the reconfigurable elements can impact upon the system behaviour, and propose enhancements to the reconfigurable circuitry fabric to enable the registers to be memory mapped.

6 Proposed alternative approach

In addition to challenges, multiple processors within a single package also offer opportunities. In silicon test, data intensive operations have migrated on-chip using techniques such SBST, BIST and dedicated test controllers. In the same way, the authors believe that an on-chip coprocessor could perform many software verification and debug tasks. This arrangement, as illustrated in Fig. 1, does require additional on-chip logic, but the size and cost of core digital cells continues to shrink. By using a coprocessor, the execution of the host processor is not impeded; alternatively, spare processing capacity in an existing core could be used. This would be particularly attractive in multi-processor designs if these tasks could be performed by a dedicated processor.

Many existing solutions already provide the necessary on-chip trigger and trace infrastructure with high-bandwidth interfaces to on-chip buffers to capture data at the fastest possible rate. Therefore what is required is a coprocessor capable of analysing this data, on-chip. This reduces I/O bandwidth requirements, with a reduced volume of data needing to be sent off-chip for analysis.

Conventional hardware instrumentation solutions limit the number of monitored events to minimise I/O, logic and memory requirements. Using a coprocessor the number of potential signals can be greatly increased, with a subset of these being selected at runtime for capture and analysis. The ability to reprogram at runtime, to monitor different

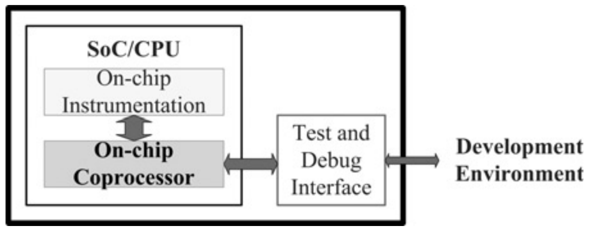


Fig. 1 *Development platform with on-chip coprocessor*

events or properties, eliminates the need to define a fixed subset at configuration or design time; giving more flexibility and minimising the risk of omitting data needed for verification. A coprocessor could also adjust trigger conditions at runtime; thereby creating sophisticated trigger conditions more efficiently than existing OCI, which requires additional hardware as the trigger complexity increases.

Analysing trace or other captured data on-chip allows the data to be filtered with redundant data being discarded, thus optimising the use of on-chip buffer memory. Where data must be sent off-chip, compression could be performed by the coprocessor; potentially eliminating dedicated hardware compression modules.

6.1 Exemplar platform

The platform chosen to examine the feasibility of the proposed approach is illustrated in Fig. 2. This is representative of many simple embedded control applications; whereby a real-time signal is computed in software, fed into a pulse-width modulation (PWM) module, and outputted using a digital pin. This PWM signal is then filtered using a simple resistor-capacitor (RC) network to produce an analogue waveform. In this case the aim is to generate is a simple triangle waveform, the amplitude of which can be increased or decreased using two input buttons. As shown, the platform uses the Freescale MC9S12XE microcontroller; which, in addition to a 16-bit CPU12X and peripheral modules also includes an on-chip debug module (S12XDBG) and XGATE coprocessor [46]. The software on the platform consists of three central functions: ReadKeys(), ComputeOutput() and OutputPWM (); which are called in a continuous loop using a simple cyclic scheduling scheme. The analogue waveform generated is dependent upon the software meeting the expected timing for the loop, which in this case is 50 µs.

Checking that execution proceeds correctly, and within defined time-limits, often poses a challenge on such platforms. By placing breakpoints in the application the execution can be tracked and verified or errant behaviour identified. However for real-time systems, such as this, halting at breakpoints is not practical. Other verification approaches using the MC9S12XE include: adding instrumentation code to generate status output, capturing and outputting trace data or using the S12XDBG state sequencer to combine several triggers before a breakpoint is generated; all of which are limited by the available hardware resources.

The alternative proposed is to use the S12XDBG module to trigger the XGATE coprocessor, which can determine if execution is proceeding correctly or an error occurs. This is a non-intrusive approach, as it does not require instrumentation or modification of the application. Although the XGATE is primarily an I/O coprocessor for real-time tasks such as interrupt handling, it can be used for any purpose including verification and debug support. The MC9S12XE allows shared access to internal memory and peripherals, which is supported by a memory protection unit, hardware semaphores and associated instructions. However, in this instance the XGATE has exclusive access to the S12XDBG module, which it can reprogramme to trigger on new events and thus overcome the limitation posed by a finite number of triggers/breakpoints.

This approach was used to debug and verify the timing for the exemplar platform described earlier. The XGATE coprocessor configured the on-chip debug module to generate an interrupt when the first function (ReadKeys) was called. When this XGATE interrupt handler was activated, the debug module was updated with conditions for the next trigger event and so on. The XGATE software also included a timer interrupt handler, which was configured to trigger at 1 µs intervals. Using this timer the XGATE could measure the time taken to execute each function and determine if it was within the expected limits. Table 1 shows the timing values which were outputted after each 100,000 iterations of the application loop. As shown, the XGATE software stored the minimum and maximum duration for each function and checked the execution time against the deadline provided. The ReadKeys() function was written with an inherent flaw whereby each key press was processed independently, introducing a delay of 10 µs for each key. Therefore when both keys were pressed this function exceed its deadline. Table 2 shows the results obtained after both keys were pressed; showing that the function executed for 20 µs longer, and that the XGATE detected the deadline violation.

The timing results obtained from the coprocessor were verified by measurement with a calibrated Agilent MSO6054A Oscilloscope. In addition, software on the XGATE coprocessor toggled an output pin to indicate execution of each function. In this way, the execution time of each function could be measured by attaching a probe to the corresponding output pin. As shown in Fig. 3, the timing values obtained are very close to the expected values, considering that the software timing values have inherent discretisation errors.

Of course the MC9S12XE was not designed for this role, and so this architecture does have limitations when used in this way. Most noteworthy, the S12XDBG module generates a CPU12X interrupt and not an XGATE interrupt, therefore a simple interrupt handler (two lines of C code) is required to trigger the XGATE. The windowing of debug module registers and trace buffer into the memory map imposes some latency and limitations. Plus the S12XDBG module uses global address locations rather than logical

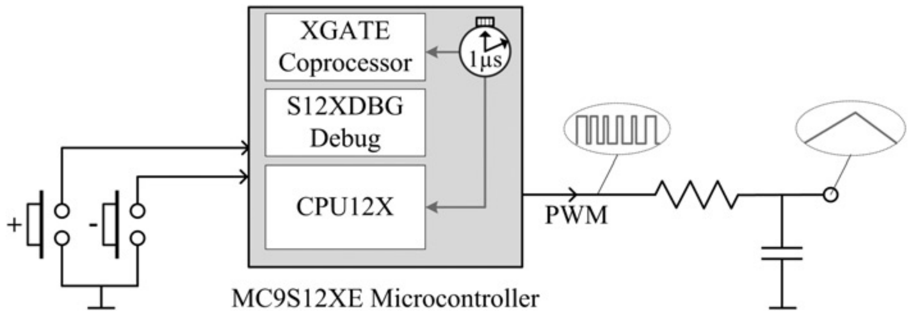


Fig. 2 *Exemplar target platform*

Table 1 Execution times for functions when operating correctly, and no keys pressed

Function	Execution time, μs		Deadline	
	Min	Max	Time, μs	Exceeded
ReadKeys	7	9	20	N
ComputeOutput	10	12	15	N
OutputPWM	9	11	15	N

Table 2 Execution times indicating missed deadline when both keys pressed

Function	Execution time, μs		Deadline	
	Min	Max	Time, μs	Exceeded
ReadKeys	7	29	20	Y
ComputeOutput	10	12	15	N
OutputPWM	9	11	15	N

paged addresses as used by the CPU, which necessitates a simple address translation.

Nonetheless this platform serves as a useful example of how such an arrangement of: CPU, debug module and coprocessor, can support verification and debug. The example application described is limited to three functions to ease explanation; however, the technique outlined could be scaled to applications of any size. A key benefit of this approach is that there is no need to capture, export and analyse vast amounts of redundant trace data; instead, the coprocessor can be programmed to output simple status data, for example, execution is as expected or an error has occurred. With infrequent or intermittent errors this can save considerable time which would otherwise be spent analysing trace data from execution runs without errors.

6.2 Architectural solution for formal runtime verification

The simple exemplar platform above is intended to demonstrate and explain the principles of the proposed approach; however, more complex solutions can also be supported. We now consider an architectural scheme for an SoC, which can support instrumented solutions for resident software testing and formal verification.

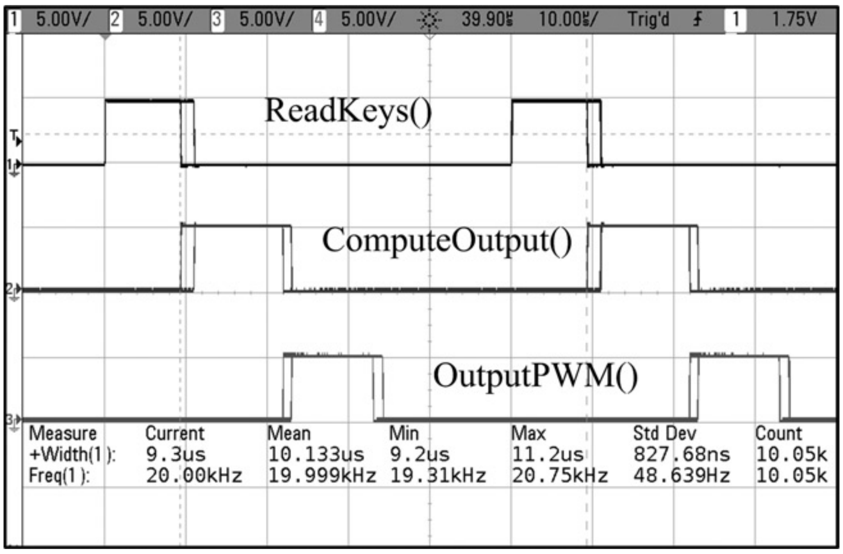
The monitoring and checking (MaC) framework architecture [47] (and the associated JavaMaC [48]) is based on formal specifications and provides a scheme to automatically link low-level observations of program execution, which can be emitted as events from a suitably instrumented target application program, to the relevant monitored properties. Such properties can then be formally verified at runtime.

A simplified representation of the overall MaC framework architecture is shown in Fig. 4. The architectures include a static phase (before program execution), and a runtime phase (during the execution). In the static phase, a mapping is established, from a formal requirements specification, between the following two entities: (i) high-level events, from the high-level requirements specification; and (ii) low-level state information, to be extracted from the instrumented target program during execution. During the runtime phase, the instrumented program is monitored and checked with respect to its requirements specification.

MaC automatically generates the runtime components: including a filter and event recogniser, generated from the low-level specification; and a runtime checker, generated from the high-level specification. The filter sends relevant state information to the event recogniser. The event recogniser detects an event from the state information received from the filter, according to a low-level specification. The recognised events are sent to the runtime checker. The runtime checker verifies, during execution, that the current execution history satisfies a high-level requirement specification.

The runtime architecture for the MaC framework is shown in Fig. 5*a*. The filter is implemented as a thread within the target application program, and communicates with a separate host computer using a dedicated communications link. The host computer executes the event recogniser and the runtime checker.

The authors' research group has devised various implementation architectures for JavaMaC and other runtime verification solutions [42, 49]. The research



```
Measure      Current    Mean       Min      Max      Std Dev   Count
+Width(1 ):  9.3us      10.133us   9.2us    11.2us   827.68ns  10.05k
Freq(1 ):    20.00kHz   19.999kHz  19.31kHz 20.75kHz 48.639Hz  10.05k
```

Fig. 3 *Oscilloscope measurement of software-loop frequency and duration of ReadKeys function*

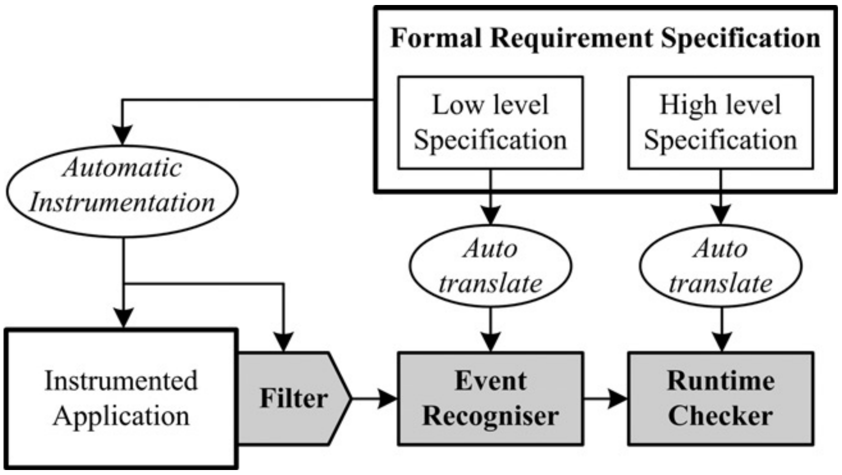


Fig. 4 *MaC architectural framework*

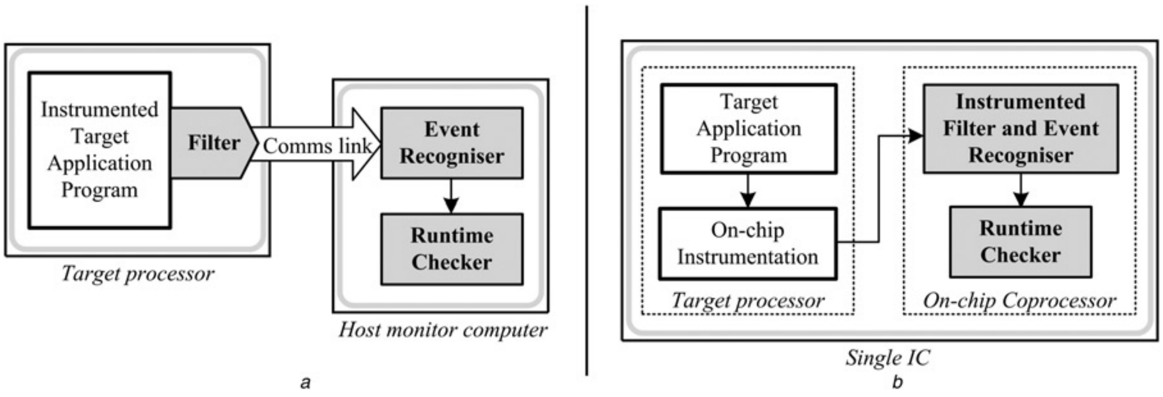


Fig. 5 *Runtime architecture for the MaC framework*

a Classical architecture for a MaC runtime verification system
b Solution for an instrumented runtime verification architecture

summarises the main disadvantages of the classical implementation architectures as follows:

- adding a thread to the target program is invasive and can make the program less deterministic,
- the communication link needs to strictly guarantee the timeliness and ordering of events,
- time synchronisation across host and target processors needs to be carefully managed,
- use of a separate host computer is cumbersome and expensive.

The solution proposed in this paper offers an architectural solution which overcomes these drawbacks. Fig. 5*b* illustrates such a solution with the following key features:

- the filter is implemented in the coprocessor and thus does not burden the target application,
- communications link is implicit in the architecture,
- synchronised timing is easily realised in the architecture,
- the single chip solution is streamlined and cost effective,
- the solution will scale to multi-processor architecture,
- a single toolset can be devised for end-to-end automated development.

7 Conclusions

Software verification and debugging on modern SoC devices requires access to the signals and data which have become deeply embedded within the silicon. IC design engineers use test interfaces aimed at minimising the I/O pins, silicon area and costs. Although the reuse of these interfaces for software purposes may reduce costs, the differing requirements between silicon and software verification leaves software developers with a sub-optimal solution. These interfaces are not designed for software verification and debug of real-time systems, and they do not provide sufficient bandwidth.

Proposed improvements to increase I/O bandwidth using high-speed or optical [22] interfaces are unlikely to resolve the problem in either the short or long term. Such interfaces require specific process capabilities, limiting usage to a subset of applications. Improving bandwidth by adding I/O pins is often not cost-effective. Despite the considerable integration possible with modern processes, I/O pads and buffers consume much larger die area as compared with core digital cells. In I/O-bound designs as the pad area increases so does the core area. Therefore minimising the number of I/O pins will continue to be a key objective in IC design. Plus, additional pins may not be practical where a rugged interface is required for harsh application environments.

Nexus and proprietary technologies face the same bandwidth limitations, the solution to which is usually a combination of: increasing the number of I/O pins, compressing the data and/or limiting the verification capabilities to those which can be supported by the interface. Efficient compression techniques can help, but increases complexity for external instrumentation. The increased silicon area and in particular the increased I/O required by these solutions makes these relatively expensive when compared with silicon test solutions.

Customised instrumentation added to hardware or software is useful where specific items need to be monitored, but excessive software instrumentation can impact upon normal execution and hardware instrumentation is not easily changed. However, such monitors can be relatively inexpensive, particularly if the solution devised can utilise an existing I/O interface and limited throughput is required.

The emphasis in all existing debug solutions is on control of the processor and data acquisition; no existing standard or industrial solution includes on-chip analysis capability. The authors outline the alternative approach of adding an on-chip coprocessor or using spare processing capacity to carry out on-chip analysis of the data captured. This solution avoids the I/O bottleneck, and the area required for a coprocessor continues to shrink as cell density increases. Plus, the ability to process data in real-time while the application is executing, serves software verification and debug needs better than existing solutions which are targeted at off-line test needs. In the authors' opinion, on-chip data analysis represents an effective way of addressing the I/O bandwidth limitations of SoC designs and enabling a new-generation of software verification and debugging tools.

8 Acknowledgments

The authors acknowledge the Irish Research Council for Science, Engineering and Technology (IRCSET) International Centre for Graduate Education in Micro and Nano Engineering (ICGEE) for its generous financial support for this research work.

9 References

1 Zorian, Y., Marinissen, E.J., Dey, S.: 'Testing embedded-core-based system chips', *IEEE Comput.*, 1999, **32**, (6), pp. 52–60

2 Foster, T.J., Lastor, D.L., Singh, P.: 'First silicon functional validation and debug of multicore microprocessors', *IEEE Trans. VLSI Syst.*, 2007, **15**, (5), pp. 495–504

3 Posse, K., Crouch, A., Rearick, J., *et al.*: 'IEEE P1687: toward standardized access of embedded instrumentation'. Proc. IEEE Int. Test Conf., Santa Clara, CA, October 2006, pp. 1–8

4 IEEE Std 1500-2005: 'IEEE standard testability method for embedded core-based integrated circuits', 2005

5 IEEE Std 1149.7-2009: 'IEEE standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture', 2009

6 Crouch, A.L.: 'IJTAG: the path to organized instrument connectivity'. Proc. IEEE Int. Test Conf., Santa Clara, CA, October 2007, pp. 1–10

7 Vermeulen, B., Stollon, N., Kuhnis, R., Swoboda, G., Rearick, J.: 'Overview of debug standardization activities', *IEEE Des. Test. Comput.*, 2008, **25**, (3), pp. 258–267

8 Hopkins, A.B.T., McDonald-Maier, K.D.: 'Debug support for complex systems on-chip: a review', *IEE Proc., Comput. Digit. Tech.*, 2006, **153**, (4), pp. 197–207

9 Chuang, C.L., Cheng, W.H., Liu, C.N.J., Lu, D.J.: 'Hybrid approach to faster functional verification with full visibility', *IEEE Des. Test. Comput.*, 2007, **24**, (2), pp. 154–162

10 Engblom, J.: 'Debugging real-time multiprocessor systems'. Proc. Embedded Systems Conf. (ESC), San Jose, CA, April 2007, pp. 1–16

11 Burns, A., Hayes, I.J.: 'A timeband framework for modelling real-time systems', *Real-Time Syst.*, 2010, **45**, pp. 106–142

12 Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T., Mehta, F., Voisin, L.: 'Rodin: an open toolset for modelling and reasoning in event-B', *Int. J. Softw. Tools Technol. Transfer*, 2010, **12**, pp. 447–466

13 Min-Allah, N., Khan, S., Yongji, W.: 'Optimal task execution times for periodic tasks using nonlinear constrained optimization', *J. Supercomput.*, 2010, **59**, pp. 1120–1138

14 Min-Allah, N., Hussain, H., Khan, S.U., Zomaya, A.Y.: 'Power efficient rate monotonic scheduling for multi-core systems', *J. Parallel Distrib. Comput.*, 2012, **72**, (1), pp. 48–57

15 Myers, G.J.: 'The art of software testing' (John Wiley and Sons Inc., New York, USA, 1979, 1st edn.)

16 Grötker, T., Holtmann, U., Keding, H., Wloka, M.: 'The developer's guide to debugging' (Springer, Heidelberg, 2008, 1st edn.)

17 Vermeulen, B.: 'Functional debug techniques for embedded systems', *IEEE Des. Test. Comput.*, 2008, **25**, (3), pp. 208–215

18 IEEE-ISTO 5001-2003: 'Nexus 5001 standard for a global embedded processor debug interface', 2003

19 ARM limited: 'embedded trace Macrocell ETMv1.0 to ETMv3.4 architecture specification', 2007

20 Jun, P., Lei, S., Siliang, H., Zhang, A.T., Hou, A.C.: 'Reconfigurable on-chip debugger with a real-time tracer'. Proc. Int. Symp. on Communications and Information Technologies (ISCIT), Sydney, NSW, 2007, pp. 158–162

21 Hopkins, A.B.T., McDonald-Maier, K.D.: 'Debug support strategy for systems-on-chips with multiple processor cores', *IEEE Trans. Comput.*, 2006, **55**, (2), pp. 174–184

22 Hopkins, A.B.T., McDonald-Maier, K.D.: 'Trace algorithms for deeply integrated complex and hybrid SoCs'. Proc. Second NASA/ESA Conf. on Adaptive Hardware and Systems (AHS), Edinburgh, Scotland, United Kingdom, August 2007, pp. 641–646

23 Daoud, E.A., Nicolici, N.: 'Real-time lossless compression for silicon debug', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2009, **28**, (9), pp. 1387–1400

24 IEEE Std 1149.1-2001: 'IEEE standard test access port and boundary-scan architecture', 2001

25 Vermeulen, B., Waayers, T., Bakker, S.: 'IEEE 1149.1-compliant access architecture for multiple core debug on digital system chips'. Proc. IEEE Int. Test Conf., Baltimore, MD, October 2002, pp. 55–63

26 Ley, A.W.: 'Doing more with less –an IEEE 1149.7 embedded tutorial: standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture'. Proc. IEEE Int. Test Conf., Austin, TX, November 2009, pp. 1–10

27 Zadegan, F.G., Ingelsson, U., Carlsson, G., Larsson, E.: 'Test time analysis for IEEE P1687'. Proc. 19th IEEE Asian Test Symp. (ATS), Shanghai, China, December 2010, pp. 455–460

28 Higgins, M., MacNamee, C., Mullane, B.: 'Design and implementation challenges for adoption of the IEEE 1500 standard', *IET Comput. Digit. Tech.*, 2010, **4**, (1), pp. 38–49

29 Lee, K.J., Hsieh, T.Y., Chang, C.Y., Hong, Y.T., Huang, W.C.: 'On-chip SOC test platform design based on IEEE 1500 standard', *IEEE Trans. VLSI Syst.*, 2009, **18**, (7), pp. 1134–1139

30 Huang, J.R., Iyer, M.K., Cheng, K.T.: 'A self-test methodology for IP cores in bus-based programmable SoCs'. Proc. 19th IEEE VLSI Test Symp. (VTS), Marina Del Rey, CA, April–May 2001, pp. 198–203

31 Krstic, A., Lai, W.C., Cheng, K.T., Chen, L., Dey, S.: 'Embedded software-based self-test for programmable core-based designs', *IEEE Des. Test. Comput.*, 2002, **19**, (4), pp. 18–27

32 Stollon, N., Uvacek, B., Laurenti, G.: 'Standard debug interface socket requirements for OCP-compliant SoC' (Whitepaper, OCP-IP Debug Working Group, 2007)

33 Stollon, N., Leatherman, R.: 'Integrating on chip debug instrumentation and EDA verification tools'. Proc. Design Con East, Worcester, MA, September 2005, pp. 1–16

34 Orme, W.: 'Debug and trace for multicore SoCs' (Whitepaper, ARM Limited, 2008)

35 MIPS Technologies, Inc.: 'EJTAG Specification', 2010

36 MIPS Technologies, Inc.: 'MIPS PDtrace Specification', 2009

37 Mayer, A., Siebert, H., McDonald-Maier, K.D.: 'Boosting debugging support for complex systems on chip', *IEEE Comput.*, 2007, **40**, (4), pp. 76–81

38 Mayer, A., Siebert, H., McDonald-Maier, K.D.: 'Debug support, calibration and emulation for multiple processor and powertrain control SoCs [automotive applications]'. Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE), 2005, pp. 148–152

39 Leatherman, R., Stollon, N.: 'An embedding debugging architecture for SOCs', *IEEE Potentials*, 2005, **24**, (1), pp. 12–16

40 Bernardi, P., Grosso, M., Rebaudengo, M., Reorda, M.S.: 'Exploiting an infrastructure-intellectual property for systems-on-chip test, diagnosis and silicon debug', *IET Comput. Digit. Tech.*, 2010, **4**, (2), pp. 104–113

41 Scottow, R.G., Hopkins, A.B.T., McDonald-Maier, K.D.: 'Instrumentation of real-time embedded systems for performance analysis'. Proc. IEEE Instrumentation and Measurement Technology Conf., Sorrento, Italy, April 2006, pp. 1307–1310

42 Watterson, C., Heffernan, D.: 'A runtime verification monitoring approach for embedded industrial controllers'. Proc. IEEE Int. Symp.

on Industrial Electronics (ISIE), Cambridge, UK, June–July 2008, pp. 2016–2021

43 Heffernan, D., Shaheen, S., Watterson, C.: 'Monitoring embedded software timing properties with an SoC-resident monitor', *IET Softw.*, 2009, **3**, (2), pp. 140–153

44 Larsson, E., Vermeulen, B., Goossens, K.: 'A distributed architecture to check global properties for post-silicon debug'. Proc. 15th IEEE European Test Symp. (ETS), Prague, Czech Republic, May 2010, pp. 182–187

45 Hopkins, A.B.T., McDonald-Maier, K.D.: 'Debug support for hybrid SoCs'. Proc. Second NASA/ESA Conf. on Adaptive Hardware and Systems (AHS), Edinburgh, Scotland, UK, August 2007, pp. 195–202

46 'MC9S12XE Family Reference Manual', Freescale Semiconductor, Inc., September 2010, Rev. 1.23, pp. 1–1326

47 Lee, I., Kannan, S., Kim, M., Sokolsky, O., Viswanathan, M.: 'Runtime assurance based on formal specifications'. Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Application (PDPTA), Las Vegas, Nevada, USA, June–July 1999, pp. 279–287

48 Kim, M., Viswanathan, M., Kannan, S., Lee, I., Sokolsky, O.: 'Java-MaC: a run-time assurance approach for Java programs', *Form. Methods Syst. Des.*, 2004, **24**, (2), pp. 129–155

49 Watterson, C.: 'A monitoring approach to facilitate run-time verification of software in deeply embedded systems'. *PhD thesis*. Library, University of Limerick, Ireland, 2010