

# **Debugging of Embedded Systems**

Term paper submitted in partial fulfillment of the requirements  
for the degree of

Bachelor of Science in Engineering at the University of Applied  
Sciences

Technikum Wien - Degree Program Business & Electronics

By: Norbert Christof

Student Number: 1010255098

Supervisor : MSc Roman Beneder  
Wien, 30.05.2013

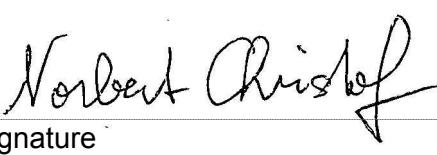


## Declaration

„I confirm that this paper is entirely my own work. All sources and quotations have been fully acknowledged in the appropriate places with adequate footnotes and citations. Quotations have been properly acknowledged and marked with appropriate punctuation. The works consulted are listed in the bibliography. This paper has not been submitted to another examination panel in the same or a similar form, and has not been published. I declare that the present paper is identical to the version uploaded.“

Dornbirn, 30.5.2013

Place, Date

  
Signature

# Kurzfassung

Die Anforderungen an Werkzeuge die die Fehlersuche im Embedded System Bereich unterstützen erhöhten sich drastisch. Analysen zeigen, dass wir alle fünf Jahre mit Prozessoren konfrontiert werden, die ihre Anschlussanzahl verdoppelt haben. Mit Blick auf die Anzahl der Transistoren pro Chip finden wir eine Verdoppelung innerhalb von zwei Jahren (Moor'sches Gesetz). Diese Entwicklung bringt uns nicht nur Produkte mit einer erhöhten Komplexität „Systems on Chip“ (SoC), sondern es werden damit auch die Chancen auf mögliche Fehler erhöht. Mehrkern-SoC Produkte und der Trend Hardware durch Software zu ersetzen, steigern diesen Effekt noch weiter.

Jeder Anbieter von Mikrocontroller oder SoC Produkten versucht eine Lösung für dieses Problem anzubieten. Sie stellen dabei meist eine eigene Entwicklungsumgebung (IDE) zur Verfügung und definieren eigene spezielle „Debug“-Schnittstellen. Der Anwender wird dabei durch Informationen über die technischen Vor-und Nachteile der einzelnen bevorzugten Systeme überflutet und in eine fast hilflose Lage versetzt. Die Situation wird durch die Produkte von Software und Hardware Anbietern, die auf Debugging-Tools spezialisiert sind, noch verstärkt.

Diese Arbeit stellt eine Recherche über aktuelle Hardware und Software „Debug“ Konzepte dar und zeigt ihre wichtigsten Funktionen bzw. ihr bevorzugtes Einsatzgebiet. Die Arbeit zeigt einen Weg durch die Vielfalt der Fehlersuchkonzepte und kann dazu beitragen, dass Designer ihre Systemauswahl sicherer treffen.

**Schlagwörter:** Debug, Bond-Out Chip, In-Circuit-Emulator, In-System-Debugger, JTAG, Nexus, DAP, BDM, Boundary Scan, Trace, OCP, SoC, Multicore

# Abstract

The requirements on tools to support the debugging work flow in embedded systems designs have increased drastically. Analysis shows that every five years we are faced with processors which have doubled their pin count. Looking at the count of transistors within a chip we find a two year doubling rate (Moore's Law). This evolution not only brings us Systems on Chip (SoC) devices with an increased complexity but also raises the chance of bugs. Multicore-SoC and the trend to replace hardware by software only serve to increase this phenomenon.

Every manufacturer of microcontrollers or SoC devices tries to find a solution to this problem with their own created Integrated Development Environment (IDE) and their specially defined debug interfaces. The user is put in an almost helpless situation, flooded by information on technical advantages and disadvantages of each privileged system. This situation is reinforced by products from software and hardware providers specialized on debugging tools.

This thesis researches the present hardware and software debug concepts, their key features and preferred areas of application. The work shows a path through the forest of debug concepts and may help designers to make their system decision more assured.

**Keywords:** Bond-Out Chip, In-Circuit-Emulator, In-System-Debugger, JTAG, Nexus, DAP, BDM, Boundary Scan, Trace, OCP, SoC, Multicore

## **Acknowledgements**

At first place I would like to thank my wife for her sympathy and support to “pull” me through all that work and study time. She motivated me many times not to give up. We passed a difficult stage of life and I hope, because I was not the youngest student, that the result now worth the effort. Thank you.

Second I would like to thank all the lecturers who taught me with a mountain of understanding and patience to a learning situation beside a fulltime job. Thanks.

A special thanks to my supervisor Beneder Roman, who was supporting me while doing this bachelor thesis, giving tips and standing by for help. Thank you.

It was great to get such interesting practical topic in the specialization course of embedded system.

Other special thanks to my customers for coming short in time and still trust me.

At Last I want to thank all student colleagues and many other people who supported me. It was an enrichment and pleasure to me to get in contact with you and had gone a short distance together with you, on the way of my life. Thanks.

# Table of Contents

1	Introduction .....	6
2	Motivation & Focus .....	7
3	Definitions .....	8
3.1	Debugging / Errors .....	8
3.2	Embedded System .....	9
4	Debugging of Hardware .....	12
4.1	Electrical Check of Circuit Boards .....	12
4.2	Manual Visual Inspection .....	13
4.3	Automatic Optical Inspection (AOI) .....	13
4.4	In Circuit Test (ICT) .....	13
4.5	Boundary Scan - Embedded System Access .....	13
5	Debugging of Software .....	15
5.1	Debugging Connections .....	15
5.2	Debugging Application Software without OS – System .....	19
5.3	Debugging of OS – System .....	20
6	Research on Special Debugging Features .....	21
6.1	Code Coverage .....	21
6.2	Event Trigger and Trigger Conditions .....	21
6.3	Execution Profiler .....	21
6.4	Fault-injection Management .....	21
6.5	Link of Internal Signals to Outside .....	22
6.6	Performance Analyzer .....	22
6.7	Streaming Trace Data .....	22
6.8	Time Stamp Ability of Trace Information .....	22
6.9	Trace Buffer Inside of the SoC .....	23
6.10	Data Monitor and Control Interface .....	23
7	Research on IDE and Tool Manufacturers .....	24
7.1	ARM Tools .....	24
7.2	Freescale CodeWarrior .....	24

7.3	Green Hills Multi - IDE .....	25
7.4	IAR-Systems Embedded Workbench .....	27
7.5	Intel Tools .....	27
7.6	iSystem Debug Tools.....	28
7.7	Keil MDK-ARM.....	29
7.8	Lauterbach's Contex Tracking System .....	29
7.9	Open Source Tools .....	30
7.10	SEGGER Development Tools.....	31
7.11	Symtavision.....	32
7.12	Wind River Tools.....	32
8	Conclusion .....	33
	Bibliography .....	34
	List of Figures .....	37
	List of Tables .....	38
	List of Abbreviations .....	39
	A: Source Links Of PCB Design software.....	41
	B: Source Links Embedded Tools .....	42

# 1 Introduction

Our modern life depends on status seeking of further, higher, better, faster, .. etc. Technical development is a boost factor on this process. Analysis [1] shows that every five years we are faced with processors which have doubled their pin count (see Figure 1). Looking at the count of transistors within a chip we find a two year doubling rate (Moore's Law: Figure 2). This evolution starts at simple digital logic circuits with few transistors and brings us to "Systems on Chip" (SoC) devices with up to a billion transistors on one chip. Packing density of printed circuits, multicore devices and the trend to replace hardware by software only serve to increase this phenomenon. High requirements on quality with nearly zero errors and operation safety in 24/7 application are challenges to pass.

Hardware designer has to overcome the problem to be able to differ between hardware errors based on production process or corrupt components and errors based on insufficient design. For that reason the requirements to those tools, to support the debugging work flow have increased drastically in the same way.

Software designers are faced with a well-nigh unmanageable count of CPU-registers and functions. The steady increase of code size leads to a time consuming debug process in "testing - searching - finding – correction – testing". Further faced with errors in maybe many software parts, highly unknown, but used in multitask operating systems.

Manufacturers of microcontrollers or SoC devices try to find a solution to this problem with their own created Integrated Development Environments (IDE) and their specially defined debug interfaces. Of course they try to bind customers to their product with special functions and incompatibility to other products.

The user is put in an almost helpless situation, flooded by information on technical advantages of each privileged system. This situation is reinforced by products from software and hardware providers specialized on tools for developing and debugging.

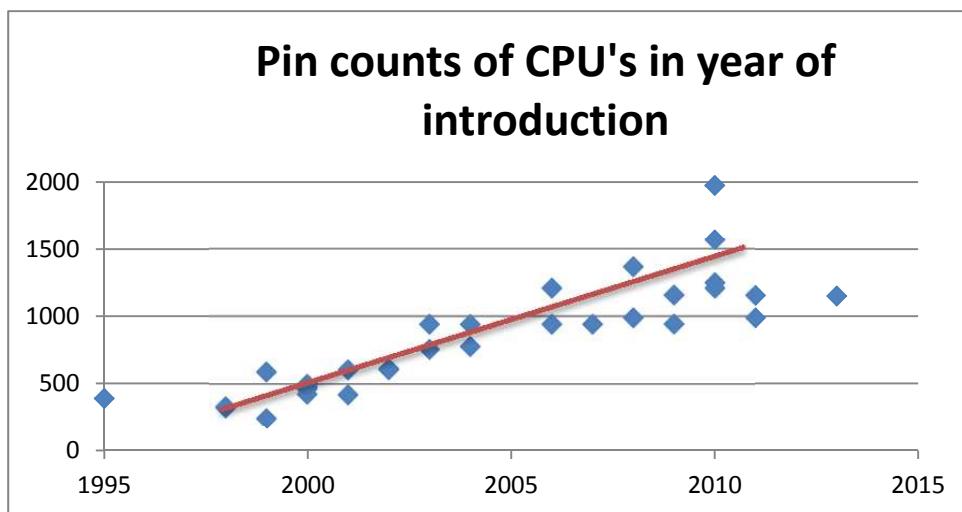


Figure 1 - Pin counts of CPU's

See below [2] as stated, the transistor count doubles nearly every two years.

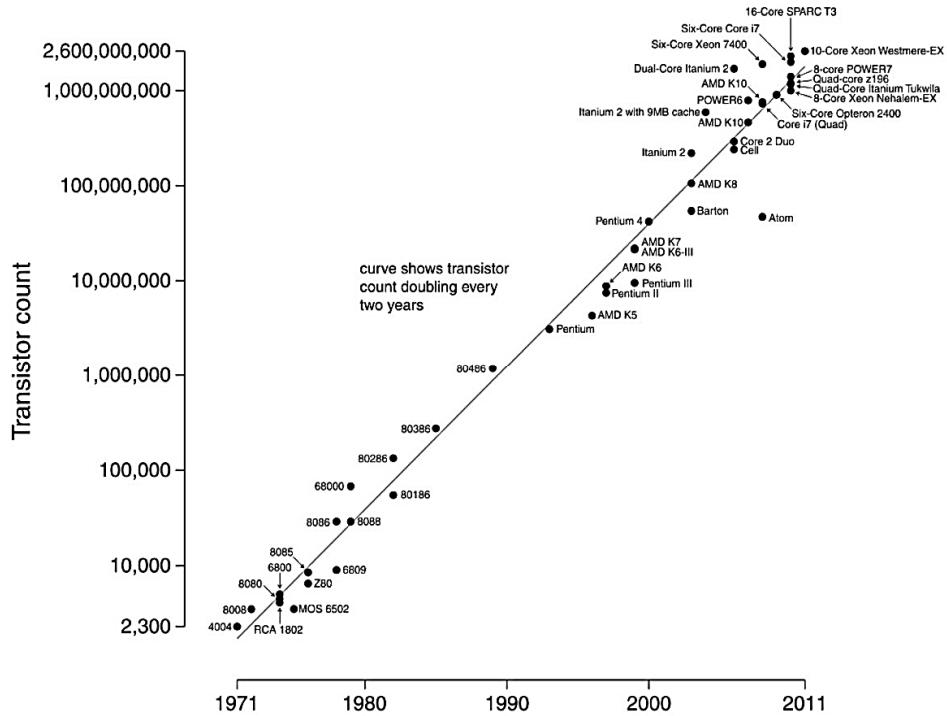


Figure 2 - Microprocessor Transistor Counts

## 2 Motivation & Focus

Depending on my own personal experience, when I had to choose a system platform for an embedded system project, I am able to state that there is lack of information on decision making in manner of debugging an embedded system. Seen from the view of life time of such a system/product, the cost driver of debugging in the extensions and maintenance phase could be very high. It is possible, as in any project, that with the change of knowledge carriers in the development team knowledge is lost and danger of bugs arises. Driven by demands in user friendly applications the complexity of such systems increases in the same way. In- and outputs for every imaginable function and requirements in high reaction or execution speed, pushes the developments to multi core and multithreading concepts.

But such systems need other solutions for debugging as developer are struggling with the problem to identify a special condition in a high density complex hardware, “filled” with complex and high functional software.

This thesis researches the present hardware and software debug concepts, their key features and preferred areas of application. The work shows a path through the forest of debug concepts and may help designers to make their system decision more assured.

## 3 Definitions

In the first step it is needed to define some terms, to bring clarity to the topic area and bounds of the research.

### 3.1 Debugging / Errors

Debugging is a work flow to find a special condition which is stated as error condition, with the aim to eliminate it or examine it in more detail. In this context the word bug is a synonym for error condition.

The cause of the condition could be based on many reasons and the common ones are listed below.

#### 3.1.1 Hardware Errors

The slighter defects are errors in the circuit design, like wrong connection or kind of trace:

- Signal errors which depends on wrong connections
- Signal errors which depends on wrong signal feed
- Function and signal errors which depends on wrong selection of component
- Errors based on critical design geometry of component pads (to solder)

These errors are countable to the group of design errors.

More Hardware bugs are based on production faults like:

- Errors in print circuit (at via connections or narrow printing wires, ...)
- Errors based on wrong placed components
- Errors based on unassembled components (missing)
- Errors based on defect components
- Errors based on placement of pirate copies<sup>1</sup> of components
- Errors based on the wrong temperature used in the soldering process  
-> bad conduction or defect components
- Errors based on tombstone<sup>1</sup> effect -> bad conduction
- Errors based on the amount of solder -> bad conduction or short circuits

These errors are countable to the group of production errors.

---

<sup>1</sup> See list of abbreviations

### 3.1.2 Software Errors

Software bugs could be divided in three main groups:

- Wrong behaviour of the software based on software developer faults
- Code syntax error in context with the programming speech -> not part of this work
- Wrong operation based on inaccurate guidelines -> not part of this work

As analysis shows [3] that 85% of embedded projects are programmed in “C” or “C++”, I will reduce my research on that area.

The second group split in more detail [4] see below:

- Error based on code syntax -> using “=” instead of “==”
- Error based on arithmetic calculations -> division by zero,..
- Error based on logic -> infinite loops<sup>2</sup>,..
- Error based on illegal use of resources -> access errors, buffer overflow<sup>2</sup>,..
- Error based on misuse of library functions
- Error based on incorrect access to hardware
- Error based on performance problems -> too high code complexity to fulfil job
- Error based on multi-threading -> race condition<sup>2</sup>, deadlocks<sup>2</sup>, TOCTOU<sup>2</sup>,..
- Error based on multi-core -> access race condition
- Error based on incorrect comments or out of date comments

## 3.2 Embedded System

An embedded system is defined as a processor which has specific tasks to do. It is embedded into a device or technical system [5], see also Figure 3. It is not limited to a special type of processor like a micro controller, PLC<sup>2</sup>, DSP<sup>2</sup> or FPGA<sup>2</sup>. Complexity and operational electrical power of the system is scaled to the needed calculating power, special function in application and on costs. The range of connections lasts from few inputs and outputs up to several complex communication lines to other devices and other embedded systems. A visual user interface is feasible, but not obligatory. Embedded systems tend to be more robust, resist to vibrations and have a greater span in operational temperature.

Depending on their area of application the program architecture depends on a single task system with some interrupt handling or depends, caused by high requirements, on a multi-tasking operating system (OS<sup>2</sup>) like Linux or Mobile Windows, as used in smart phone. In some applications there is real time response of the system (RTOS<sup>2</sup>) needed and other applications need a maximum safety of power, as they are battery powered and/or implemented in a human body.

---

<sup>2</sup> See list of abbreviations

Most embedded systems are created to run 24 hours a day – seven days a week without a personal intervention, they use a watch dog timer (WDT<sup>3</sup>) and a power fail or loss detection circuit as safety elements, to get after a fault condition back on the “feet”.

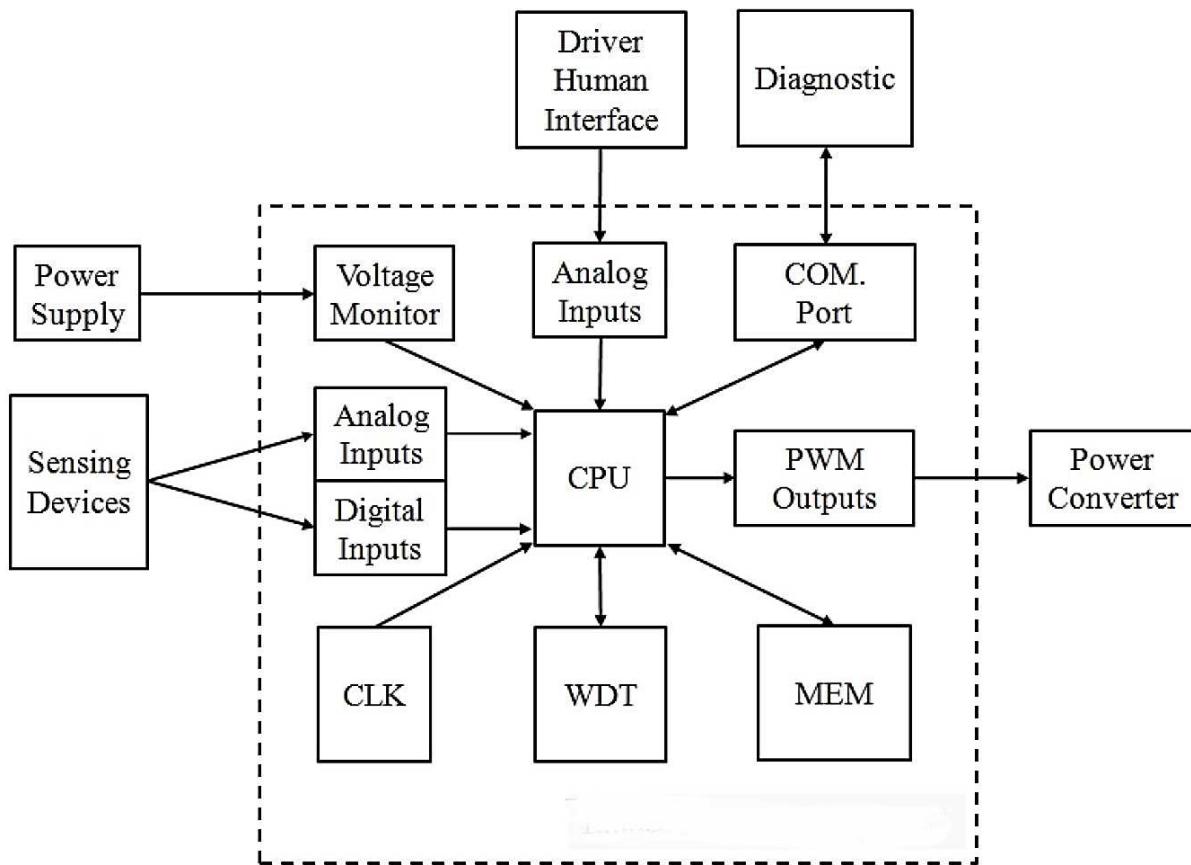


Figure 3 - Typical embedded system architecture [6]

The analysis of a present email research done by the Wilson Research Group [3] based on 2098 valid responds shows that:

- 85% of embedded projects are programmed in “C” or “C++”
- Java lies below 4%
- 60% of embedded projects contain a 32 bit processor
- 70% use an operating system or real time operating system
- 24% use an in-house/custom written OS
- 16% use Android, 13% Ubuntu, 13% FreeRTOS, 7% MS Embedded Windows
- 76% of designs are based on a single or dual processor system (24%)
- 70% of the current designs don't contain a FPGA

---

<sup>3</sup> See list of abbreviations

- Influence on selection of system-level tools depends with 27% on software engineering management; 20% selects “hardware architecture” as main influence point
- 82 % use Subversion<sup>4</sup>, Git<sup>4</sup> or CVS<sup>4</sup> as version control system
- 26 up to 29% use manufacturers like Texas Instruments, MicroChip or Freescale (multi-selection was possible)
- 80% of participants are not using and not planning to use virtualization/hypervisors<sup>5</sup>

---

<sup>4</sup> See list of abbreviations

<sup>5</sup> A piece of computer software, firmware or hardware that creates and runs virtual machines

## 4 Debugging of Hardware

On the first sight this sounds easy, but these types of errors could be a challenge to find. The use of high quality circuit design software could eliminate most of the design bugs in the first step. Messages are raised when connections are missed or connection between wires and components pins are not allowed.

Linked with a version control software it is possible to build up an almost error free component database, so that only with creation of new components and first time use, bugs are possible.

Today, almost every variant of print circuit design software meet the above requirements. Same applies to the ability to trace high speed signal as differential wires (same length of wire to prevent signal run time differences). Few manufacturers are able to trace the lines with guidelines to impedance values, to fulfil requirements on signal form conditions or to match connection impedance. Some producer are able to simulate the circuit design in part, up to full function of it and are able to visualize the print in 3D to check mechanical collisions with other parts of the system. The saw "What you pay is what you get." counts here. Various manufacturers names are presented in Table 1 of Appendix "A". The above mentioned tools help to prevent bugs at the design stage.

### 4.1 Electrical Check of Circuit Boards

Today this is done fully automated by the producer of the board. Test data are generated out of the connection list, taken from production data. The board is put on an unique nail bed (Figure 4) where connections are checked, to identify and reject boards with open or shorted wire conditions [7]. This could be supported by x-ray inspection to get a view on the inner layers.

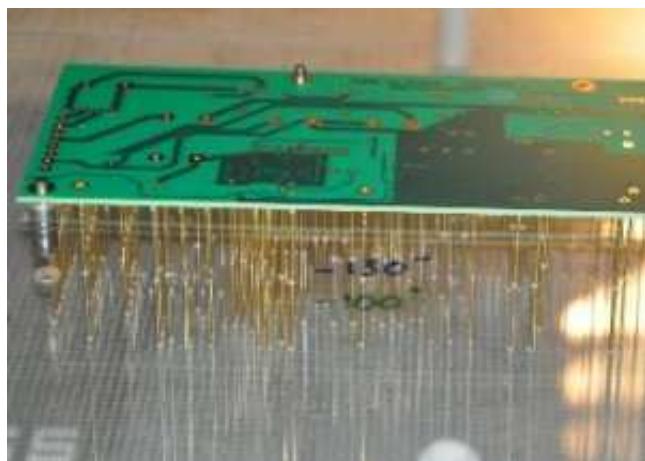


Figure 4 - unique nail bed [7]

## 4.2 Manual Visual Inspection

Manual visual inspection systems are mainly used in low volume productions after the assembling and soldering process. They are flexible and highly accurate for low to medium count of components. System variants are supported by magnifier, automated light pointer and automated transport systems. With rising component density, count of components and higher production volume this inspection type is not accurate – work flow tends to get tedious to the working persons, with the consequent of raising error rate in finding bugs.

## 4.3 Automatic Optical Inspection (AOI)

The investment and running costs in such inspection systems are high, so they are useful when manual inspection is unsuitable. In the first run one or more cameras take pictures of the board as reference. In the next step the production run is compared to the reference. Divergent boards are sorted out for visual inspection and repair. To identify defects at BGA solder points, X-ray tools are used [8].

## 4.4 In Circuit Test (ICT)

The board is put on a nail bed (Figure 4) to establish connection to power supply, programming devices and measurement tools. Possible tests vary from just checking some voltages and current consumption, up to a full in system check with an observation of power up condition, injections of special signals or preparation of special error conditions. Limitation bounds are set by mechanical density of test points and possible electrical connection to the point of interest which should be checked and of course, the cost factor.

A special version of an ICT is the “flying probe<sup>6</sup>” system, but it is limited by the mechanical precision to reach test points at a given size. The advantage is to save the relatively high expense of the nail bed and be able to change test points at low costs. This is purchased by high investment and running costs. In the end it depends on the place of action if suitable to use it.

## 4.5 Boundary Scan - Embedded System Access

The Joint Test Action Group established in 1990 test possibilities, moulded in the IEEE standard 1149.1 [9], with the development of the JTAG- connection. Meanwhile improvements are undergone to the original version to reduce the needed pin count and to support improved debugging [10], see also Figure 5.

The used processor type has to support this type of test connection. With use of this debug connection, it is now possible to get direct access to input and output pins with influence on their state.

---

<sup>6</sup> Special version of an in circuit tester, based on robot technic with limited connections.

In the following test process it is possible to check static signal behaviour and states, and to switch on/off enable signals of dedicated circuit parts. Combined with in circuit tests it is possible to test basic hardware conditions without writing end user software.

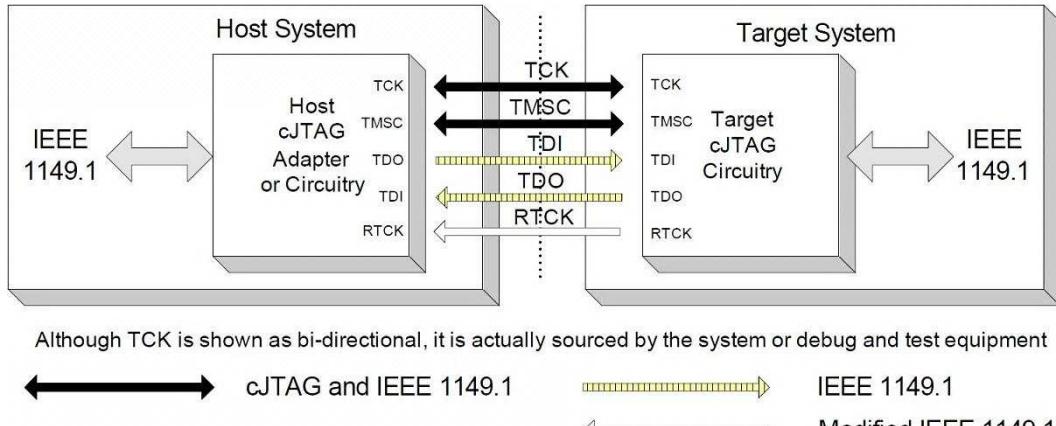


Figure 5 – Signals of Standard IEEE 1149.1 and compact JTAG (cJTAG<sup>7</sup>) connection [10]

Limitation of use brings the transfer speed of the connection, since all bits of each pin has to be shifted in serial mode (see Figure 6), each time of bit change, even only one bit has to be changed. So it is not accurate to program internal or external memory and operate communications interfaces bound to the processor. High speed state changes are not possible to record.

To overcome this limitation some manufacturers implement a so called “micro code”, a special written test software with basic function. Now it is possible to access and test internal or external modules in a high efficient way.

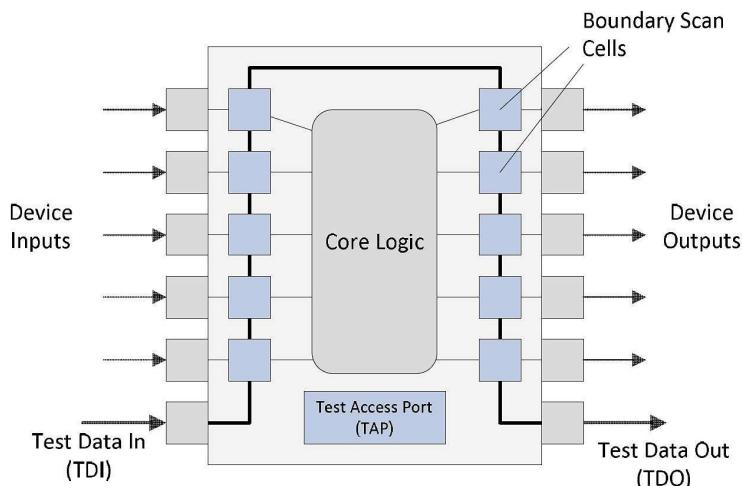


Figure 6 - Boundary scan cell [11]

<sup>7</sup> Compact JTAG IEEE 1149.7 standard

## 5 Debugging of Software

Debugging of software is maybe the most demanding and time consuming working process in a software development work flow. To support this with the best possible solution manufacturers have to fight with the increase of memory size, operation speed and complexity of implemented functions. The efforts rise and fall with the intelligence, speed, cost factor and usability of it.

As mentioned nearly each manufacturer presents his own integrated development environment. Most time in a basic version free of use.

The IDE software is able to

- manage software projects (including library support)
- handle properties for compiling/building
- compile code
- provide a code editor
- download software in the processor
- and run the debug session.

An easy, fast and common approach is to (cmp. to [12] p. 28 - 34):

- Use a port line to signal a special condition -> very fast, with oscilloscope to check
- Print characters over a communication line -> in some (slow) cases accurate, but not always possible -> influence the execution speed
- Use of a digital or mixed signal oscilloscope with storage memory -> some have direct support to communication protocols standards like: SPI<sup>8</sup>, I2C<sup>8</sup>, RS232<sup>8</sup>, CAN<sup>8</sup>, WLAN<sup>8</sup> ,..

### 5.1 Debugging Connections

Nowadays behaviour on debugging connection should be:

- Easy to implement in hardware
- Cheap to implement
- Low costs for the needed interface, to connect to the debugging system
- Usability, easy to handle
- High speed to handle present amount of data in an accurate time
- Multi-useable for setting up application, debugging hard- and software,

Below there is a list of present implemented interface types. Most of them count on implemented debug hardware (ICD<sup>9</sup>) and on "In circuit Emulation" (ICE) solution.

---

<sup>8</sup> See list of abbreviations

### 5.1.1 In Circuit Emulation

Because of the high costs and complex procedure, it is a solution for special cases. Each manufacturer has his own standard. Change of producer or sometimes even a change of processor family leads to repeated expenses -> so bind customer to him.

The original chip is replaced by a bond-out chip<sup>10</sup> version with debug options or a FPGA chip which simulates the function. The advantage is to have the full control, at full speed.

“MicroChip” put their standard serial interface extended with additional debug lines and put some trace memory and break point memory into their “Real ICE” product. But compared with the above explanations, I would say it is not a real ICE.

### 5.1.2 ICD over Serial Connection

Below some debug technology based on serial connection. In common they need an external specialized, but cheap interface.

- Background Debug Mode (BDM) used by “Motorola” (now Renesas) and “Freescale”
- DebugWire invented by “Atmel”

Both are single wire debug concepts with limited debug capability like “run”, “stop”, “single step” and setting some break points. At a break point event it is possible to read and write internal memory registers. “MicroChip” uses a similar interface with three wires, called “In circuit serial programming” (ICSP).

Additionally to single wire debugging capability the debugging interface of “Analog Devices” used in the ADuC-family and based on the Intel 8052 core, is able to be switch over to the real RS232 port. In this case it is possible to debug and download program code without a special connector and external Interface.

### 5.1.3 Debugging with JTAG Connection

The IEEE standard 1149.1 and its variation 1149.6 with its support to differential signals and 1149.7 as compact JTAG with two wire connection (see Figure 5) makes it possible to use an inexpensive interface for debugging. Because it relays on an industry standard and is broadly used by many manufacturers there exist a great number of commercial and free open source tools. A great advantage is the possibility to use it also as hardware test connection. The new 1149.7 extends the standard by support of power management of the processor (power down, sleep,...) and possible star topology of multiple TAPs<sup>11</sup>. “IPextreme”

---

<sup>9</sup> In circuit debugger - Implemented hardware to support in system debugging capability

<sup>10</sup> Special version of chip with debug capability

<sup>11</sup> Test Access Point

offers a ready to use IP-core for FPGA implementation of the IEEE standard 1149.7 [13]. Texas Instruments is one of the first manufacturers who has implemented it in his products.

### 5.1.4 Debugging over Nexus 5001 Connection

Nexus 5001 standard extends the JTAG standard with additional control lines and a high speed full duplex auxiliary connection. The high speed connection is based on packet-based messaging to support some real time capabilities like logic analyser and data acquisition (see Figure 7).

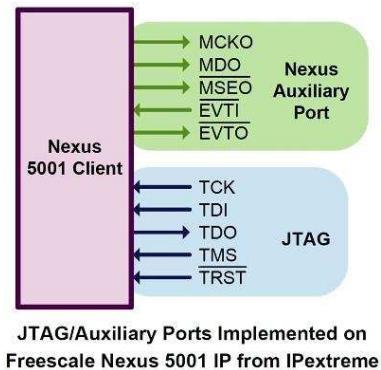


Figure 7 - JTAG/Auxiliary Port allocation [14]

Additionally feature classes were defined to support different (cheaper) demands on the interfaces (see Figure 8). For example class 1 is realized through the standard JTAG connection. Class 2 up to 4 needs the new extended lines. Time critical applications will benefit from these new features.

This standard is supported by “Motorola” (Renesas), “ST Microelectronics”, “Hitachi Semi.”, “Infineon” and “Mitsubishi Electronic Corp.”

Additional products are available from “Freescale” and “National Semiconductor”.

Debug Feature	Class 1	Class 2	Class 3	Class 4
Read and write user registers in debug mode	✓	✓	✓	✓
Read and write user memory in debug mode	✓	✓	✓	✓
Enter debug mode from reset	✓	✓	✓	✓
Enter debug mode from user mode	✓	✓	✓	✓
Exit debug mode to user mode	✓	✓	✓	✓
Single-step instruction in user mode; re-enter debug mode	✓	✓	✓	✓
Stop execution on instruction/data breakpoint; enter debug mode	✓	✓	✓	✓
Set breakpoint or watchpoint	✓	✓	✓	✓
Device identification	✓	✓	✓	✓
Notify of watchpoint match	✓	✓	✓	✓
Monitor process ownership in real time (ownership trace)		✓	✓	✓
Monitor program flow in real time (program trace)		✓	✓	✓
Monitor data writes in real time (data trace)			✓	✓
Monitor data reads in real time (optional for Class 3 and 4)			✓	✓
Read and write memory in real time			✓	✓
Start ownership, program, or data trace on watchpoint				✓
Control processor to avoid trace overruns				✓

Figure 8 - Support Classes of Nexus 5001 [14]

### 5.1.5 Debugging with ARM CoreSight

It is also a possible IP-core implementation which is especially developed by the ARM Corporation to use it as debug core in combination with their offered CPU-cores. As ARM is no processor producer, the implementation depends on processor manufacturer.

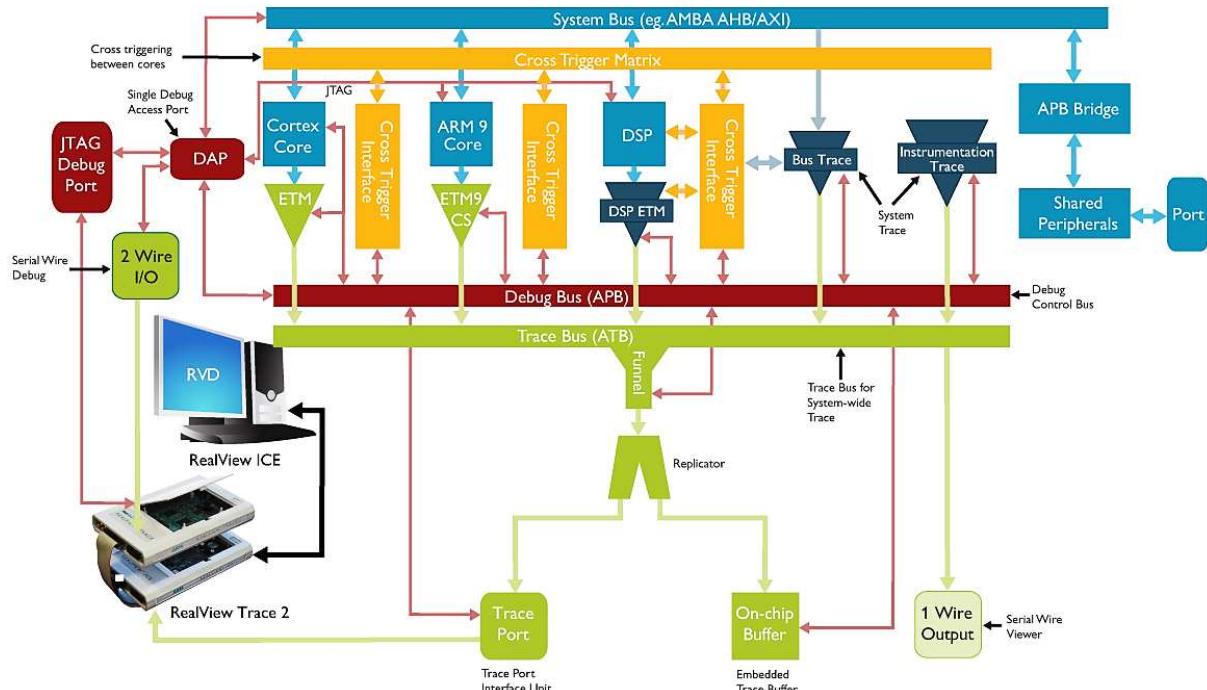


Figure 9 - Block image of CoreSight - ARM Debug & Trace IP [15]

Connection compared to IEEE 1149.1 is extended by a trace port Interface unit. Scalability in pin count, cost rate and function are offered by use of a JTAG connection or a two-wire serial port. Full functional driven, it supports in multi –core environment time stamp ability of trace information, access to system memory and peripheral or debug registers.

## 5.2 Debugging Application Software without OS – System

Software most time offered by the processors manufacturers are able to support this application in a basic way.

Basic debug features are supported as:

- Loading program
- Reset processor / restart
- Run code
- Stop code execution
- Step though source code
- Run to cursor in source code
- Set / clear breakpoint
- Watch symbol values
- Watch local variable values
- Watch/Modify memory - view it in some number representation
- Watch heap/stack

Additional advanced tasks are maybe supported by the processor and debug software:

- Stopwatch - taking time between two breakpoints
- View a call graph – views who is calling the function
- Software call stack with function arguments
- High speed data trace – only with special interfaces possible
- Running several instances of the debug software, to be able to debug multi-processor systems or to debug two embedded systems with one computer.

Events which may occur in a debugging session, but easy to handle and to be kept in mind:

- Interrupts “run” in parallel with the main code. If code execution stopped, they are not serviced any longer.
- Watchdog timer function has to be stopped / switched off – otherwise this will lead to a watchdog error event during the debug session.
- Timing at communication ports is maybe no longer correct. It depends at which execution step in the protocol frame the code execution was stopped.

## 5.3 Debugging of OS – System

As in systems code size and instruction speed increases, it is now possible to put complex multi-tasking system in an embedded system. The use of an OS-System software brings higher structure in code and brings support or make it even possible to separate device drivers from normal application code.

The higher flexibility is bought with very high complexity:

- Multi-tasking overhead
- Memory of different threads should be protected to one another
- A Special debugger for each OS-System
  - > RTOS aware like: RTOS-Greenhill, Tasking, Keil RTX and furthermore Debian Linux , MS-Mobile Window, and many more variants
- Only special OS versions offer real time ability
- Processor must support a special memory concept to run the OS
- Higher requirements on system memory size
- An execution stop of the application should not stop the OS or the opposite

The above “problem to solve list” shows that debugging inside an OS – System raises conditions and creates demands which are essential. So the debug environment gets maybe more expensive.

## 6 Research on Special Debugging Features

The following subitems show an over view of possible features as state of the art picture additionally to the mentioned points under 5.2 and 5.3. The features are not represented by only one software/hardware product, but it should give the reader an idea what is possible today.

### 6.1 Code Coverage

Code coverage marks code that has been executed, helping to ensure you have thoroughly tested your application. It helps to observe which part of code has not been tested yet. So give a direction, which test falls are missed to test the rest of the code [16].

### 6.2 Event Trigger and Trigger Conditions

Event Triggering by the on-chip trigger unit, by software breakpoints or by “Break Input” pins [17]. To be able to set complex trigger conditions in or without symbolic conditions for enhanced definitions [17].

### 6.3 Execution Profiler

The Execution Profiler records execution time and counts for each assembler instruction and high-level statement in your program.

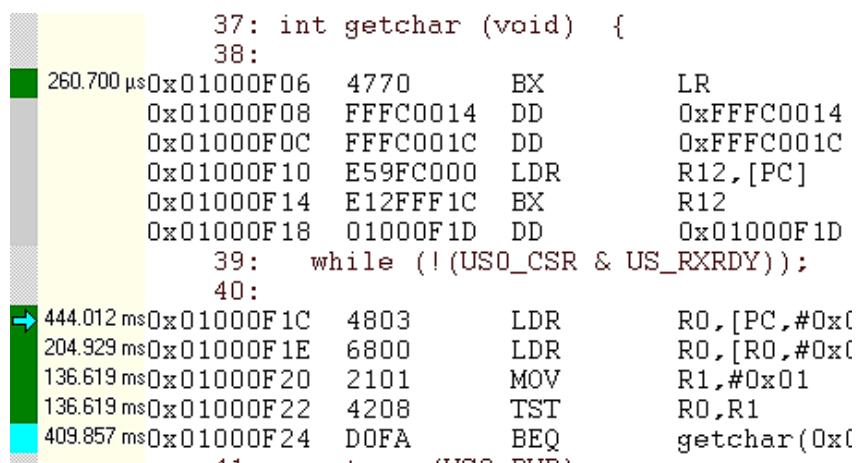


Figure 10 - View example of recorded execution time [16]

### 6.4 Fault-injection Management

The tool supports a system check, how it reacts on known faults condition [18]. After changes were made to the system, it is not suitable to make all possible tests (maybe no time to make it or not possible). The tool leads to a case sensitive library of test pattern.

## 6.5 Link of Internal Signals to Outside

Some SoCs link internal signals, like DMA request lines, interrupt request, wakeup events or power domain status to an outside port pin. In Linux this characteristic could be found under the term of “linux-omap master branch” or OMAP3.

## 6.6 Performance Analyzer

The performance analyzer records and displays execution times for selected functions and program blocks. Bar graphs display the time spent in each part of your program [16].

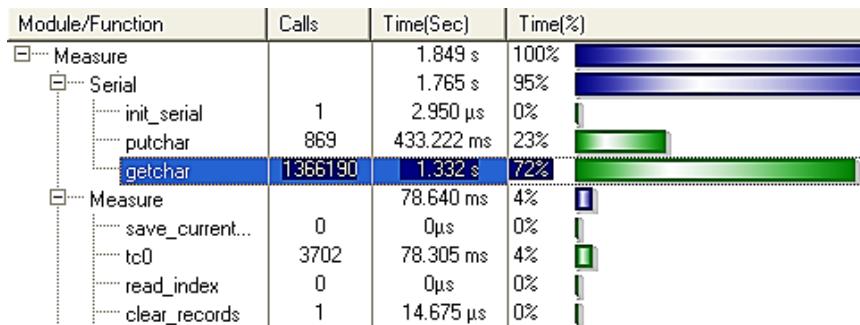


Figure 11 - Graph example of a performance analyzer [16]

## 6.7 Streaming Trace Data

Debug method of streaming trace data to an external trace receiver is called ETM [15]. It is supported by the JTAG interface and of course a receiver is needed. Afterwards it is possible to run a re-debugging session of a traced program section and allows forward and backward debugging capabilities.

## 6.8 Time Stamp Ability of Trace Information

Especially useful in multi-core environments, this helps to get the time context of various events. Work flow is supported by timeline charts [15].

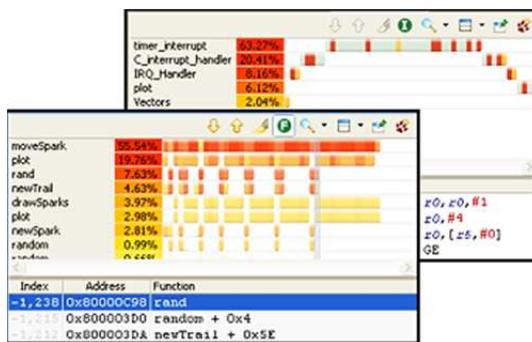


Figure 12 - Time Stamp trace information [15]

## 6.9 Trace Buffer Inside of the SoC

Memory for buffering executed code ETB<sup>12</sup> [15]. Based on cost it is usually small -> start at 2kByte or 8kByte. Supported by JTAG.

E.g.: ARM processors with ETB up to 16 Kbytes or Infineons TC27xED family with up to 1 Mbytes size.

## 6.10 Data Monitor and Control Interface

The Data Monitor and Control Interface (DMCI) IDE plugin module [19] provide the possibility to change values of variables during runtime by graphical instruments (slider) and visual the value of the influenced variables as plotted graph. The main application is seen in area of motor control. It is possible to see the influence of changed values in a direct visual form.

---

<sup>12</sup> Embedded Trace Buffer

## 7 Research on IDE and Tool Manufacturers

The research should give a cross section of common development tools. A compare between the tools is renounced not to prefer one without testing it in praxis. Some options look basic and not dramatic, but may have powerful strength in the work flow. So this was not realized to prevent prejudice on tools and of course some genius tools will have their prize.

### 7.1 ARM Tools

The product line for easy-to-use shows us the Keil-IDE, see below at section 7.7.

The product line for “more” professionals is the Development Studio 5 (DS-5) [15].

Based on Eclipse IDE, it offers following features:

- Debug support for bare-metal, RTOS and Linux and Android platforms
- Non-intrusive cycle-accurate ETM and PTM<sup>13</sup> instruction trace
- System performance analysis for Linux and Android systems
- CPU usage statistics per process, thread, function and source code
- Event-based sampling allowing to assign PMU<sup>14</sup> counters, such as cache misses, to source code
- Automated debug sessions for faster debug cycles
- ARM Mali™ GPU graphics performance analysis
- Probe to acquire and correlate actual power data with system performance
- Data collection over network - No debug adapter required
- A Real-Time Simulator

### 7.2 Freescale CodeWarrior

The standard Development Studio from Freescale [20] is based on Eclipse with industry-standard plug-ins.

The development Suites are scalable in function and price with following properties:

- Support multi-thread debugging and profile analysis
- Prepared for debugging in Linux and common RTOSs
- Integrated support for scripts written in Python language to collect and export trace data automatically

---

<sup>13</sup> Program Trace Macrocell

<sup>14</sup> Performance Monitoring Unit

The analysis is displayed in a basic form.

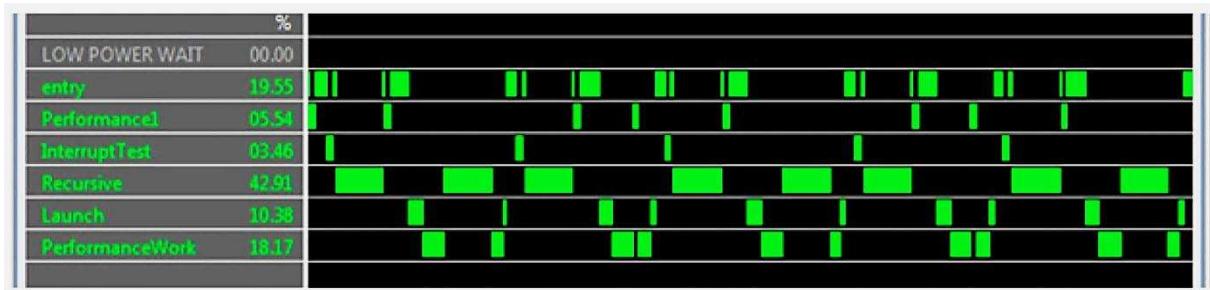


Figure 13 - Timeline Result of HW Trace Points [20]

Data Analysis is presented in the following form:

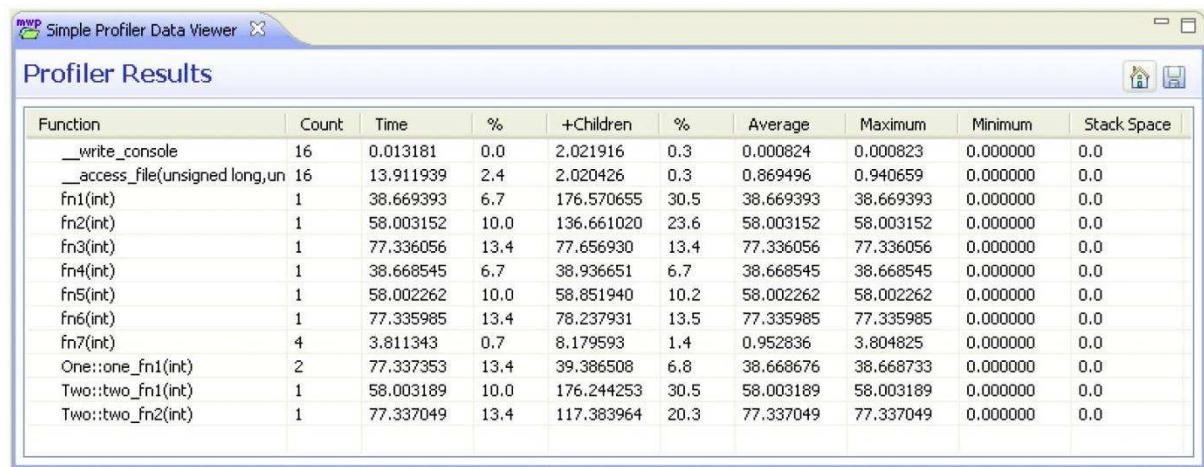


Figure 14 - Simple Profiler Data Viewer - Tree View [20]

### 7.3 Green Hills Multi - IDE

Green Hills Multi Integrated Development Environment [21] offers special support for their multi-tasking System “RTOS” and offers Tools like:

An “OSA Explorer” [21], which provides comprehensive views into the status of kernel objects, tasks and resources.

An “EventAnalyzer” [21], which helps to tackle high-level performance problems such as incorrect task priorities, excessive interrupt level processing, too many calls to the operating system, and unexpected task context switches.

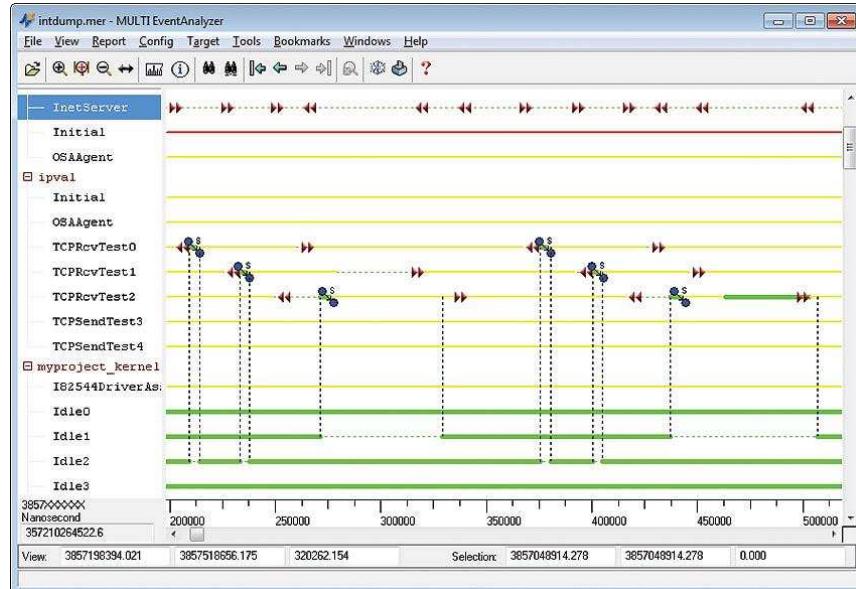


Figure 15 - Example view of "EventAnalyzer" [21]

A “TimeMachine” tool suite which displays complex system execution flow in an easy-to-understand way, see Figure 16:

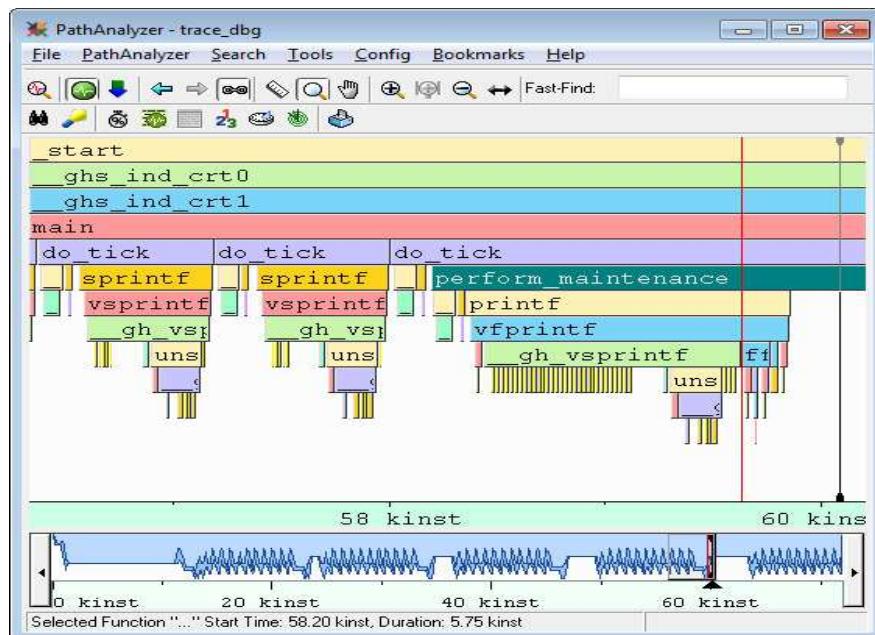


Figure 16 - Example view of "TimeMachine" tool [21]

## 7.4 IAR-Systems Embedded Workbench

“IAR-System” offers a JTAG interface for “in circuit debugging” scalable for different needs. Some have the ability to measure current and voltage values.

JTAG Tools offers:

- A 20-pin or 10-pin connector
- Download speed range up to 30MByte/s
- Up to 18MByts trace memory
- 56 bit Time stamp values
- All ETM<sup>15</sup> modes, triggers and filters

The Workbench supports the following features:

- Integrated development environment with project management tools and editor
- Automatic checking of MISRA C <sup>16</sup>rules (MISRA C:2004)
- Extensive hardware target system support
- Power debugging to visualize power consumption in correlation with source code
- Relocating ARM assembler
- C-SPY® debugger with ARM simulator, JTAG support and support for RTOS-aware debugging on hardware
- RTOS plugins available from IAR Systems and RTOS vendors

A cut out of an example view is shown below:

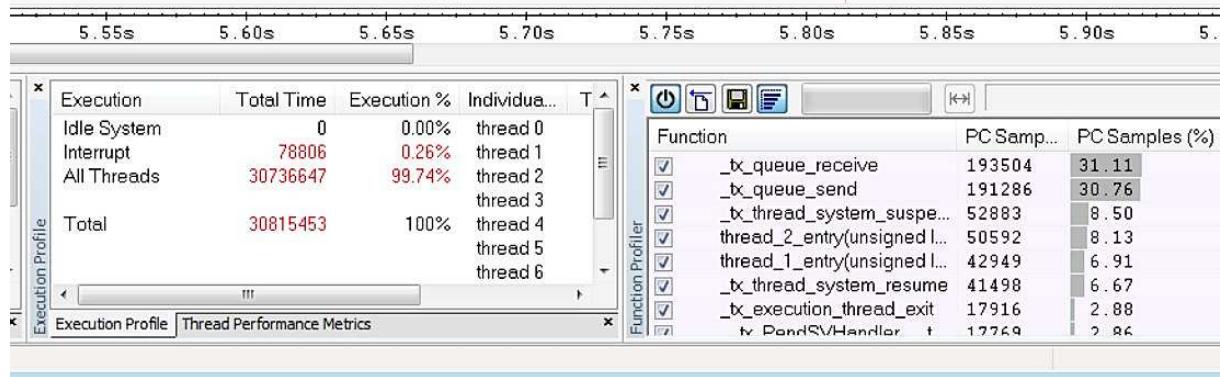


Figure 17 - Example view of execution profile [22]

## 7.5 Intel Tools

Intel refers to the embedded development tools from “Wind River”, see below at 7.12.

<sup>15</sup> ARM Embedded Trace Macrocell

<sup>16</sup> C programming language developed by MISRA

For tracing and to find bottlenecks and traffic peaks Intel has developed software like “Trace Analyzer” and “Collector suite”.

## 7.6 iSystem Debug Tools

“iSystem” offers hardware for “in circuit emulation” with specific processor replacement. Furthermore the product “iTag” which supports JTAG debugging over USB or Ethernet connection. In combination with the software product “winIDEA” it is possible to:

- run Python scripting
- be OS awareness (FreeRTOS, RTX, rcX, CMX, µC/OS-II,...)
- Realtime (unit) testing with testIDEA™
- allowing 3rd party tools accessing all winIDEA™ functionalities
- Multi-core debugging
- Real-time analyzer (Trace, Profiler, Execution Coverage), see Figure 19

Example view of Code coverage:

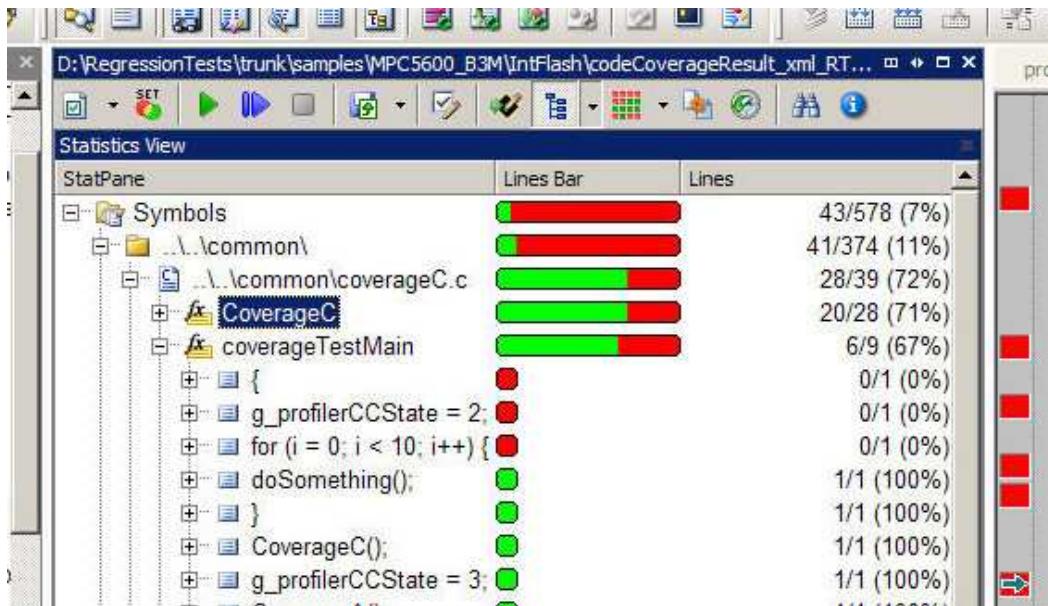


Figure 18 - Example view of code coverage [23]

Example view of code statistics:

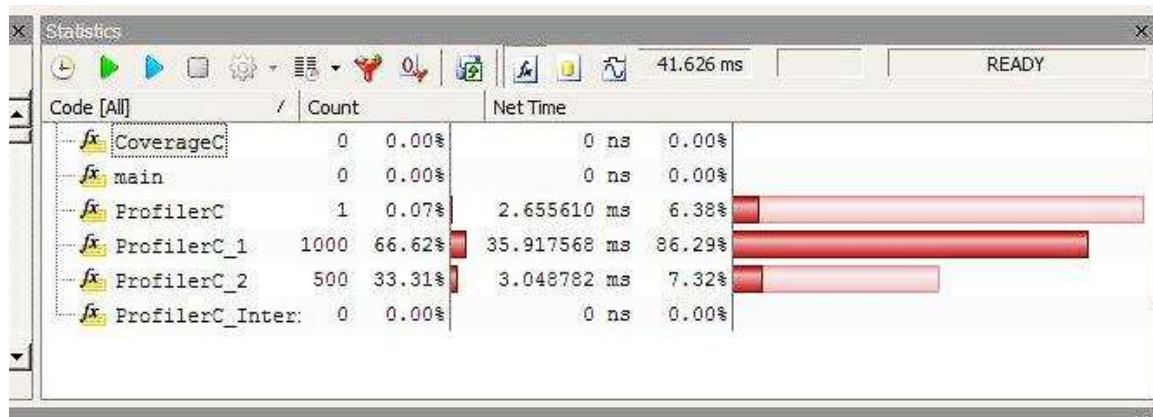


Figure 19 - winIDE profiler [23]

## 7.7 Keil MDK-ARM

The “MDK-ARM” is a specialized IDE for ARM processor-based devices. IDE supports the in-house small footprint real-time operating system “RTX”.

Debugger supports:

- Code Coverage
- Execution Profiler
- Performance Analyzer

For example views see Figure 10 and Figure 11.

## 7.8 Lauterbach’s Contex Tracking System

Lauterbach Trace systems hardware and software tools offer very special, not to say luxurious, trace features like:

- Trace32 tool memory starting from 128 MByte up to 4 GByte!
- PowerTrace-II module supports up to 16 GBit/s total trace port bandwidth
- Trace allows streaming to a file
- Real-time processing of streamed trace information
- Sample possibility on a specified event with switching on/off trace buffer at specified event
- Supports of multi-core debugging
- Supports all common on-chip trace buffers
- Allows re-debugging of a traced program section
- Forward and backward debugging capabilities!
- Cache Analysis for finding transfer bottlenecks
- Verify effects of different cache sizes
- Verify effects of code optimisation
- Detailed analysis of task run-times and states

- Graphical analysis of variable values over the time
- Detailed analysis of function run-time
- Long-Time Hardware Coverage Analysis for Emulator and ETM
- Trace Based Coverage Analysis for ICD and Emulator
- Real-time trigger on current, voltage and power, with optional analog module
- Real-time measurement of 3 current and 4 voltage lines, with optional analog module
- Energy statistics on function and task level

Example view of thread time line:

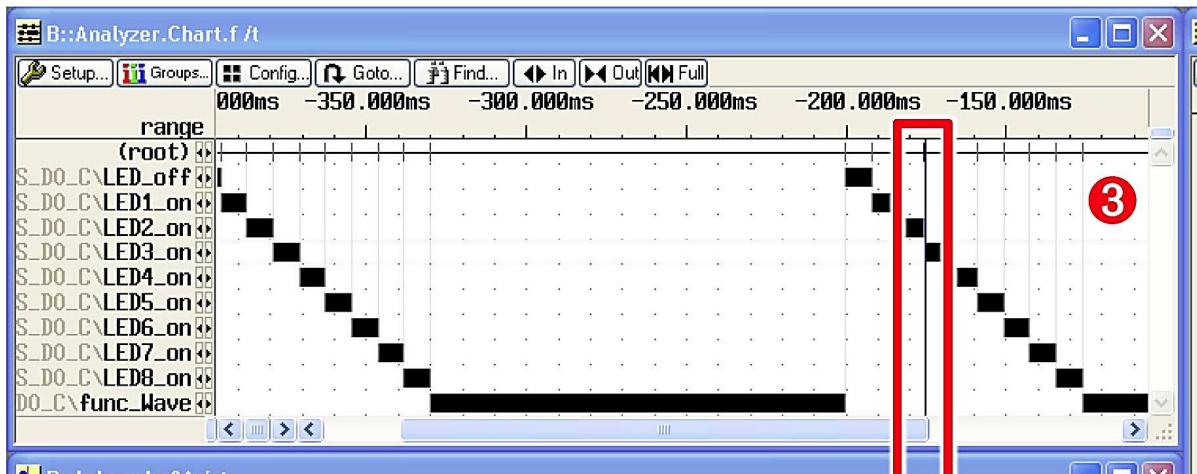


Figure 20 - Example view of thread time line [24]

Example view of power statistics:

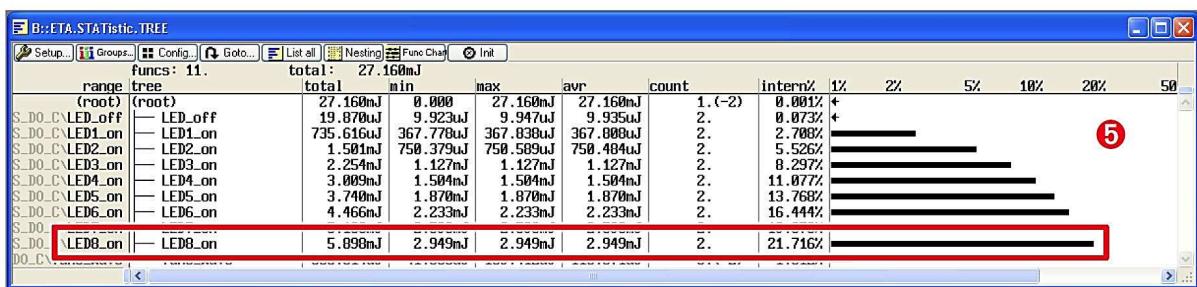


Figure 21 - Example view of power statistics [24]

## 7.9 Open Source Tools

For Unix like system as Linux, Ubuntu, Android, there are many open source tools available.

Below a list as cut out of them:

- OpenOCD – Open Source Debugger project based on On-chip Debugging [25]
- Common Trace Format (CTF) -> Industry standard for trace data
- BableTrace -> converter between CTF and other trace formats

- DTrace -> dynamic tracing framework to troubleshoot the kernel at real time called by command line, output through the standard output.
- FTrace -> internal tracer to find out what is going on inside the kernel called by command line, output through the standard output.
- SystemTap -> is a software infrastructure to simplify collection of information about the running Linux system
- Strace -> debug utility to monitor system calls, operating through command line

## 7.10 SEGGER Development Tools

Seeger offer a multi-task OS “embOS” and uses “embOSView” as Debug environment [26].

Especially usable for microcontroller with limited resources.

Analyzer functions with embOSView over serial UART interface.

Their Hardware product series “J-Link” (=JTAG Interface) supports common IDEs like Atmel Studio, Eclipse, Keil, Mentor Embedded or Freescale CodeWarrior” with listed features see below:

- Version with Ethernet interface
- Serial Wire Debug supported
- JTAG speed up to 60MHz
- JTAG signals could be monitored
- Target voltage can be measured
- Download speed up to 3MByte/s
- Fully plug and play

## 7.11 Symtavision

The company provides a model based solution for timing design, performance optimization, finding timing bounds and creation of scenario analysis. The trace analyser tool imports trace logs and visualize them in an easy-to-understand form, see Figure 22.



Figure 22 - Gantt-chart of tasks execution [27]

## 7.12 Wind River Tools

As others tools manufacturers Wind River offer an in-house hardware product line for debugging based on the JTAG debug standard, with Ethernet interface for remote debugging and a software product line for embedded development.

The work bench [28] offers support to common operating systems and the whole product life cycle. There exists a special Linux version.

More feature see below:

Development Phase	Hardware Bring-Up	Kernel/OS, Driver & BSP	Application Software Design & Debug	Code QA & Test	System-Level Testing	Deployment & Management
Workbench On-Chip Debugging						
Application Developer Edition, Linux Platforms						
Platform Developer Edition, Linux Platforms						
Platform Developer Ed., VxWorks Platforms						
Workbench Unit Tester						

**TARGET ARCHITECTURES:** ARM · Intel Architecture · MIPS · PowerPC · SuperH · XScale · ColdFire

**TARGET OPERATING SYSTEMS:** VxWorks 5.5.x · VxWorks 6.x · VxWorks 653 · Wind River Linux

**HOST OPERATING SYSTEMS:** Linux · Solaris · Windows

Figure 23 - Workbench for all phases of the development life cycle [28]

## 8 Conclusion

The research shows that time and demands create changes.

Hardware “debugging” has come to a high level standard. Improvements could be seen by simulation of the circuit design and with more intensive information exchange about problems in the production process, between designer and producer.

Software debugging starts from the point that nearly each manufacturer has his preferred in-house debugging and connection concept, a trend has started to come to less standards – or do you have thought some years ago that hardware and software producers are pulling together in the same direction (e.g. boundary scan / JTAG)?

Mostly driven by the needed amount of resource or development time (e.g. Intel bets on “Wind River” tools) and the time to put it on market as product, escorted by price pressure, more and more producers are setting on already established standards and try to influence them as work group member. As it is a global industrial driven market and therefore follows its legality, only really big players or innovative solutions are able to take influence on already implemented debugging concepts.

The customer decides if the money which he has to dispense for the working tools he gets, worth it. It is the question of economic efficiency to purchase a cheap and basic instrument or to buy the most expensive environment what is possible to acquire for money.

In my opinion it is, to keep in mind that *debugging an embedded system* is not only a question of high quality hardware or debugging tools. Many other aspects have to be taken into account also.

Question about the use of tools like:

- Project management tools
- Bug reporting tools
- Issues organizing / tracking tools
- Collaboration and Content Sharing tools
- Version control systems

should be discussed open or taken obviously as state of the art.

Finally it has to be said, the best tools do not worth the money paid, if the user is not able to get the best out of it. So training and education is the best investment on such a human “debugging system”.

## Bibliography

- [1] S. Physics, “*Progression of CPU pin counts*,” [Online]. Available: <http://leepavelich.wordpress.com/2011/09/19/progression-of-cpu-pin-counts/>. [Accessed 11 05 2013].
- [2] Wikipedia, “*Moore's Law*,” [Online]. Available: [http://en.wikipedia.org/wiki/Moore's\\_law](http://en.wikipedia.org/wiki/Moore's_law). [Accessed 18 05 2013].
- [3] R. G. Wilson, “*Embedded Market Study*,” UBM Tech, by email invitation, 2013.
- [4] Wikipedia, “*Software bug*,” [Online]. Available: [http://en.wikipedia.org/wiki/Software\\_bug#Mistake\\_metamorphism](http://en.wikipedia.org/wiki/Software_bug#Mistake_metamorphism). [Accessed 18 05 2013].
- [5] Schröder, Dillmann and Gockel, *Embedded Linux - Praxis Buch*, Berlin Heidelberg: Springer, 2009.
- [6] F. Alexandru, “*Embedded Control System Design*,” Berlin Heidelberg, Springer, 2013, p. 6.
- [7] C. I. Bay Area, “*Bare Printed Circuit Board Electrical Test*,” [Online]. Available: <http://bayareacircuits.com/bare-printed-circuit-board-electrical-test/>. [Accessed 18 05 2013].
- [8] T. I. Blackfox, “*Courseware store*,” [Online]. Available: [http://www.blackfoxonline.net/store/media/Blackfox\\_courseware\\_sample.pdf](http://www.blackfoxonline.net/store/media/Blackfox_courseware_sample.pdf). [Accessed 19 05 2013].
- [9] Vermeulen, Kühnis, Rearick, Stollon and Swoboda, “*Overview of Debug Standardization Activities*,” IEEE - 04534167, *IEEE Design & Test of Computers*, 2008.
- [10] Work Group, of Mobile Industry Processor Interface, “*Compact JTAG*,” MIPI - Alliance, 2005. [Online]. Available: [http://www.mipi.org/sites/default/files/mipi-wp-cjtag-v1\\_42.pdf](http://www.mipi.org/sites/default/files/mipi-wp-cjtag-v1_42.pdf). [Accessed 13 05 2015].
- [11] Cunha, Barbosa and Rodrigues, “*On the use of Boundary Scan for Code Coverage of Critical Embedded Software*,” IEEE - 06405382, *IEEE Computer Society*, 2012.
- [12] “*eLinux - Debugging Embedded Linux Systems*,” Texas Instruments, [Online]. Available: [http://elinux.org/images/d/d7/EIc2011\\_gadiyar.pdf](http://elinux.org/images/d/d7/EIc2011_gadiyar.pdf). [Accessed 21 05 2013].

- [13] I. IEEE 1149.7, “IPextrem,” [Online]. Available: <http://www.ip-extreme.com/IP/cJTAG.shtml>. [Accessed 19 05 2013].
- [14] I. Nexus, “IPextrem - Nexus,” [Online]. Available: [http://www.ip-extreme.com/IP/nexus\\_5001.shtml](http://www.ip-extreme.com/IP/nexus_5001.shtml). [Accessed 19 05 2013].
- [15] “ARM - Debug & Trace IP,” ARM, [Online]. Available: <http://www.arm.com/products/system-ip/debug-trace/index.php>. [Accessed 21 05 2013].
- [16] “Keil - Tools by ARM,” Keil, [Online]. Available: <http://www.keil.com/>. [Accessed 21 05 2013].
- [17] “PLS - Debugging and Emulation via JTAG,” PLS Programmierbare Logic & Systeme, [Online]. Available: <http://www.pls-mc.com/content/view/108/234/>. [Accessed 21 05 2013].
- [18] “Critical - csXCEPTION,” Critical, [Online]. Available: <http://asd.criticalsoftware.com/csxception/>. [Accessed 21 05 2013].
- [19] “MicroChip - Integrated Development Environment,” MicroChip, [Online]. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/51549f.pdf>. [Accessed 21 05 2013].
- [20] “Freescale - CodeWarrior Development Suite,” Freescale, [Online]. Available: [http://www.freescale.com/webapp/sps/site/prod\\_summary.jsp?code=CW-SUITE-PROFESSIONAL&fsp=1&tab=Documentation\\_Tab](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=CW-SUITE-PROFESSIONAL&fsp=1&tab=Documentation_Tab). [Accessed 21 05 2013].
- [21] “Green Hills - Multi IDE,” Green Hills, [Online]. Available: [http://www.ghs.com/products/MULTI\\_IDE.html#faster](http://www.ghs.com/products/MULTI_IDE.html#faster). [Accessed 21 05 2013].
- [22] “IAR - Embedded-Workbench,” IAR, [Online]. Available: <http://www.iar.com/en/Products/IAR-Embedded-Workbench/ARM/Product-packages/>. [Accessed 22 05 2013].
- [23] “iSystem winIDEA - Debug and Test Tools,” iSystem, [Online]. Available: <http://www.isystem.com/products/winidea>. [Accessed 21 05 2013].
- [24] “Lauterbach - Trace Features,” Lauterbach, [Online]. Available: <http://www.lauterbach.com/frames.html?home.html>. [Accessed 22 05 2013].
- [25] “OpenOCD - Open On-Chip Debugger,” Open source project, [Online]. Available: <http://openocd.sourceforge.net/>. [Accessed 21 05 2013].
- [26] “Segger Embedded Software Solutions,” Segger, [Online]. Available: <http://www.segger.com/cms/embos.html>. [Accessed 21 05 2013].

- [27] “*Syntavision - TraceAnalyzer*,” *Syntavision*, [Online]. Available: [http://www.syntavision.com/downloads/Flyer/Syntavision\\_TraceAnalyzer\\_33\\_overview\\_EN\\_1302.pdf](http://www.syntavision.com/downloads/Flyer/Syntavision_TraceAnalyzer_33_overview_EN_1302.pdf). [Accessed 21 05 2013].
- [28] “*Windriver - Work bench product overview*,” *Windriver*, [Online]. Available: [http://www.windriver.com/products/product-overviews/PO\\_WB\\_1110.pdf](http://www.windriver.com/products/product-overviews/PO_WB_1110.pdf). [Accessed 21 05 2013].
- [29] Beck, Lisboa and Carro, *Adaptable Embedded Systems*, New York: Springer, 2013.
- [30] S. Oresjo, “*Agilent Technologie - NEW TEST STRATEGY FOR COMPLEX PRINTED CIRCUIT BOARDS*,” *Agilent technologie*, [Online]. Available: [http://www.home.agilent.com/upload/cmc\\_upload/All/Nepcon99.pdf?&cc=AT&lc=ger](http://www.home.agilent.com/upload/cmc_upload/All/Nepcon99.pdf?&cc=AT&lc=ger). [Accessed 19 05 2013].

## List of Figures

Figure 1 - Pin counts of CPU's .....	6
Figure 2 - Microprocessor Transistor Counts .....	7
Figure 3 - Typical embedded system architecture [6] .....	10
Figure 4 - unique nail bed [7].....	12
Figure 5 – Signals of Standard IEEE 1149.1 and compact JTAG (cJTAG) connection [10] ..	14
Figure 6 - Boundary scan cell [11].....	14
Figure 7 - JTAG/Auxiliary Port allocation [14] .....	17
Figure 8 - Support Classes of Nexus 5001 [14] .....	18
Figure 9 - Block image of CoreSight - ARM Debug & Trace IP [15].....	18
Figure 10 - View example of recorded execution time [16] .....	21
Figure 11 - Graph example of a performance analyzer [16].....	22
Figure 12 - Time Stamp trace information [15].....	22
Figure 13 - Timeline Result of HW Trace Points [20] .....	25
Figure 14 - Simple Profiler Data Viewer - Tree View [20].....	25
Figure 15 - Example view of "EventAnalyzer" [21] .....	26
Figure 16 - Example view of "TimeMachine" tool [21].....	26
Figure 17 - Example view of execution profile [22].....	27
Figure 18 - Example view of code coverage [23] .....	28
Figure 19 - winIDE profiler [23].....	29
Figure 20 - Example view of thread time line [24] .....	30
Figure 21 - Example view of power statistics [24] .....	30
Figure 22 - Gantt-chart of tasks execution [27] .....	32
Figure 23 - Workbench for all phases of the development life cycle [28] .....	32

## List of Tables

Table 1 - Manufacturers register of printed circuit design software.....	41
Table 2 - Manufacturers register of embedded development tools .....	42

## List of Abbreviations

- **B** -

buffer overflow: Access out of bounds of the defined memory structure ..... 9

- **C** -

CAN: Controller Area Network - serial bus system ..... 15

CTF: • Common Trace Format..... 30

CVS: Aged open source version control system..... 11

- **D** -

deadlocks: Thread A waits on B to stop, but thread B waits onto A to finish too ..... 9

debugging: Work flow to find a bug in a design..... 2

DSP: Digital Signal Processor..... 9

- **E** -

ETM: ARM Embedded Trace Macrocell ..... 27

- **F** -

flying probe: Special version of an in circuit tester, based on robot technic with limited connections ..... 13

- **G** -

Git: latest open source version control system used by linux ..... 11

- **H** -

hypervisors: A piece of computer software, firmware or hardware that creates and runs virtual machines..... 11

- **I** -

I2C: Inter-Integrated Circuit, serial bus invented by Philips (NXP)..... 15

ICD: In Circuit Debugger - Implemented hardware to support in system debugging capability..... 15

ICE: In circuit Emulation ..... 15

ICT: In Circuit Tester..... 13

IDE: Integrated Development Environment..... 2

infinite loops: Loop until variable is zero, but variable is never decremented..... 9

- **M** -

Multicore-SoC: More than one system core on one chip ..... 2

- **O** -

OS: Operation System ..... 9

- **P** -

pirate copies: Illegal copy, most time with reduced quality and function..... 8

PLC: Programmable Logic Controller ..... 9

PMU: Performance Monitoring Unit..... 24

- **R** -

race condition: Order of access differ to the intend of programmer .....	9
RS232: Standard for serial communication -> TIA-232-F .....	15
RTL: Real Time Operation System - responds within the shortest needed time – time depends on application...	9

- **S** -

SoC: Systems on Chip .....	6
SPI: Serial Peripheral Interface, serial bus invented by Motorola.....	15
Subversion: Open source version control system.....	11

- **T** -

TOCTOU: Time to check differ to time of use .....	9
tombstone: Component rises in vertical position, due to physical effects during soldering process.....	8

- **V** -

version control software: Sotware to manage changes .....	12
---	----

- **W** -

WDT: Watch Dog Timer - resets in case of infinite loops.....	10
WLAN: Wireless Local Area Network .....	15

## A: Source Links Of PCB Design software

no	name	link <sup>17</sup>
1	Altium Designer	<a href="http://www.altium.com/">http://www.altium.com/</a>
2	Cadence Allegro	<a href="http://www.cadence.com/products/pcb">www.cadence.com/products/pcb</a>
3	Designspark	<a href="http://www.designspark.com">http://www.designspark.com</a>
4	DipTrace	<a href="http://www.diptrace.com/">http://www.diptrace.com/</a>
5	Eagle	<a href="http://www.cadsoft.de">www.cadsoft.de</a>
6	<i>Metor Graphics</i>	<a href="http://www.mentor.com/">http://www.mentor.com/</a>
7	Proteus	<a href="http://www.labcenter.com">http://www.labcenter.com</a>
8	Pulsonix	<a href="http://www.pulsonix.com/">http://www.pulsonix.com/</a>
9	Target 3001	<a href="http://server.ibfriedrich.com">http://server.ibfriedrich.com</a>

Table 1 - Manufacturers register of printed circuit design software

---

<sup>17</sup> Links are valid on date of paper creation.

## B: Source Links Embedded Tools

no	name	link <sup>18</sup>
1	ARM	<a href="http://www.arm.com/">http://www.arm.com/</a>
2	Freescale	<a href="http://www.freescale.com/">http://www.freescale.com/</a>
3	Green Hills	<a href="http://www.ghs.com">http://www.ghs.com</a>
4	IAR	<a href="http://www.iar.com/">http://www.iar.com/</a>
5	Intel	<a href="http://software.intel.com">http://software.intel.com</a>
6	iSystem	<a href="http://www.isystem.com/">http://www.isystem.com/</a>
7	Keil	<a href="http://www.keil.com">http://www.keil.com</a>
8	Lauterbach	<a href="http://www.lauterbach.com">http://www.lauterbach.com</a>
9	Segger	<a href="http://www.segger.com">http://www.segger.com</a>
10	Syntavision	<a href="http://www.syntavision.com/">http://www.syntavision.com/</a>
11	Wind River	<a href="http://www.windriver.com/">http://www.windriver.com/</a>
12		
13	atollic	<a href="http://www.atollic.com">http://www.atollic.com</a>
14	pls	<a href="http://www.pls-mc.com/">http://www.pls-mc.com/</a>

Table 2 - Manufacturers register of embedded development tools

---

<sup>18</sup> Links are valid on date of paper creation.