Project Overview: External Debug System (EDS)

“Give me a place to stand on, and I will move the earth.”
Image from: http://www.math.nyu.edu/~crorres/Archimedes/Lever/leverBig.gif

“The new platform is the place to stand, the new debugger is the lever, it is up to software teams to move the earth.” – author of this paper
1 SCOPE ................................................................................................................................................... 3
2 ABSTRACT............................................................................................................................................. 3
3 DEFINITION OF TERMS/ABBREVIATIONS.......................................................................................... 3
4 PRE-CONCEPT PHASE .......................................................................................................................... 4
5 PROJECT LIFECYCLE ............................................................................................................................... 5
  5.1 Concept Phase .......................................................................................................................... 5
  5.2 Development Phase .................................................................................................................. 9
    5.2.1 Project Planning........................................................................................................ 10
    5.2.2 Vendor Selection Process .......................................................................................... 12
    5.2.3 SE Requirements........................................................................................................ 13
    5.2.4 SE Architecture ......................................................................................................... 13
    5.2.5 Basic Operation .......................................................................................................... 17
    5.2.6 Verification and Validation .......................................................................................... 19
  5.3 Operations Phase & Maintenance Phase ............................................................................... 19
  5.4 Retirement Phase .................................................................................................................... 19
6 PROJECT RETROSPECTIVE .................................................................................................................. 20
7 ROLE OF SYSTEM ENGINEERING ................................................................................................... 20
8 CONCLUSION AND PARTING WORDS ............................................................................................. 21
1 SCOPE
This document is a project summary for the creation of a semi-custom software debug system or External Debug System (EDS).

2 ABSTRACT
With the development of a new product platform designed to enable new product features at higher performance it was also important to consider the needs of internal customers. This paper discusses the development of semi-custom software debug system (Emulator), called the External Debug System, designed to work with the new platform to create a commercial quality development environment. The work was done as a sub-project of the main platform project and is an excellent example of the application of system engineering.

3 DEFINITION OF TERMS/ABBREVIATIONS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakpoints</td>
<td>Places in the code where the user tells a software debugger they wish the system to stop. A “standard” breakpoint would be identified by selecting a specific instruction or line of code as the place to stop.</td>
</tr>
<tr>
<td>Canned Device</td>
<td>Production products are sealed in titanium can. Hence the term “canned device”. Once sealed, IO is limited.</td>
</tr>
<tr>
<td>Complex Breakpoints</td>
<td>In addition to standard breakpoints, many tools allow complex breakpoints in various forms. Options appearing in commercial tools include: A standard breakpoint that only occurs when a Boolean condition is satisfied or when a variable takes on a certain value. Breaking on the stack pointer hitting a certain value. Breaking on a write to a range of memories. Breaking on a read from a range of memories. State machine controlled breakpoints where multiple steps arm a breakpoint, while certain events disarm it. I.e. Break on a call to the move_mem routine but only when it is called by the event holter module and not by the communication code.</td>
</tr>
<tr>
<td>Deep Trace</td>
<td>Trace typically refers to recording a sequence of instructions to memory. This is standard feature of modern emulators but typically the trace depth is limited to 1 to 16 Megabyte. Recent products (last five years), have introduced the concept of deep trace where trace is stored directly to a hard drive allowing gigabytes of information to be analyzed.</td>
</tr>
<tr>
<td>Emulator</td>
<td>In this context, emulator is short for in-circuit-emulator or ICE. ICE is a tool that allows software developers to debug code on real hardware. Often this is done by replacing the CPU with an external emulated CPU, but it can be done in other ways without the user being able to tell the difference. In many modern architectures, CPU replacement is impossible due to systems-on-chip constraints so solutions such as on-chip-debug, software monitors and bus-observation are employed to accomplish the desired functionality.</td>
</tr>
<tr>
<td>Host Computer</td>
<td>Typically emulator systems consist of a host PC/computer in conjunction with a custom piece of hardware that connects to the target. In some systems, the custom hardware is merely a JTAG connector but in others it is a more sophisticated piece of equipment.</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>Hybrid</td>
<td>A small circuit board with IC and passive components.</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>In-Circuit Emulator</td>
<td>Another term for emulator.  See emulator.</td>
</tr>
<tr>
<td>Module</td>
<td>Another term for Hybrid used in production.</td>
</tr>
<tr>
<td>Patch Registers</td>
<td>Hardware support for patching ROM. CPU operation is monitored, when the CPU attempts to fetch from a “patched location”, it is re-vectored to a new location. Usually only a small number of Patch Registers are provided to allow ROM to be modified in the field. In addition, such a feature could be...</td>
</tr>
</tbody>
</table>
used as the basis for implementing a SW monitoring programmer for a SW based debugger.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>The term “register” without modifiers refers to CPU registers such as the Stack Pointer, Program Counter, Index registers and accumulator. See special function registers.</td>
</tr>
<tr>
<td>Single Stepping</td>
<td>A logical extension of the ability to apply breakpoints is to execute code one line at a time in the debugger. This is referred to as single stepping and allows the software developer to inspect the firmware condition at every point along the execution path.</td>
</tr>
<tr>
<td>Special Function</td>
<td>The term special function register refers to a set of special purpose registers that control the system operation beyond the CPU registers. In the case of the Unified Platform, the registers that control the General Purpose CPU Timers are an example.</td>
</tr>
</tbody>
</table>

**SW Software**

**Target System**

**Triggered Trace**

Trace that is triggered on some pre-condition taking place. On commercial systems, it is often the case that the same conditions that can trigger a breakpoint, can also trigger a trace to stop or start.

**VLSI**

Very Large Scale Integration. Used in this document to refer to IC design teams.

4 **PRE-CONCEPT PHASE**

As part of our normal product development cycle, the R&D teams began planning for the next generation platform that would enable the development of new product features at lower costs and higher performance. Another clear goal of the platform was to better meet the needs of internal customers in Marketing, Clinical, R&D, Production and Field Support. In particular, software development costs were rising and software was increasingly dominating the critical path to project completion.

To handle the needs of software development, a series of workshops took place with the various software development groups to understand the tools they needed to do their work and the impact of those needs on the development of the new platform project. This was done very early in the platform definition process so that the results could drive some of the platform decisions. Some of the results included:

- Early development of product software needed to happen before platform ICs would be available in order to hit market timelines. Thus a system simulator was critical to starting software development before hardware was available.

- Once HW was available, it needed to be used as a primary software development target to maximize the benefits of concurrent engineering. However, past use was hampered by poor reliability and availability of debug solutions. Different projects also used different solutions including: printf style debugging, Off-the-Shelf Emulators, Logic Analyzers, custom deep trace tools, simulators, etc.

- There were also issues discovered with the early development boards which caused corruption of low-level measurements. This was because the development boards included much longer traces and much higher capacitance than the final product due to debug support and technology differences (development boards are 1 square feet vs less than 4 square inches). This hampered the development of algorithms which relied on low noise measurements, as they were forced to use product hardware with limited debug visibility. It was questioned as to what extend the noise issues could be addressed or whether more debug support could be put into the final platform. If the solution required more debug support in the final platform, then it was clear that there would be limits as to what could be provided due to connectivity issues caused by the small final footprint.

- The platform would be supported company wide across different continents and thus any debug solution would need to take support and training into account.

Based on these inputs and others we started researching available technologies:
• We talked to some external emulator vendors about what solutions they had and what they felt could be done given our limitations and at what cost.

• We researched a debug standard coming out of the automotive industry which gave ideas as to what was possible and done in other industries.

• We talked to a company specializing in On-Chip-Instrumentation. This is a solution where small amounts of debug circuitry were put on-chip and are usually accessed by a special bus. Several levels of debug support were possible and this company had worked with a competitor of ours on a similar project. Many mainstream systems-on-chip products use this approach.

• We researched past solutions to the same problem done on other projects across the company. The company had created multiple debug systems with different feature sets which gave us a pretty good idea about what was possible and how much it would cost in terms of resources. Support, maintenance and ability deploy in sufficient volumes were all problems with past solutions.

• We worked with the groups working on the main platform components and the follow-on product development projects to understand their timelines and plans.

• We worked with the groups defining the system architecture to understand what blocks from past projects would be reused and the concept for the CPU core and external bus.

At this point it became clear that we wanted to take a step forward and that we wanted to do so as a series of sub-projects of the platform project in addition to making “software friendly” design decisions for the platform sub-systems\(^1\). One sub-project would create a software simulator using existing in-house technology. The other sub-project would create the External Debug System\(^2\) (EDS) capable of running on real hardware which is the main focus of this paper.

5 PROJECT LIFECYCLE

The following sections discuss important parts of the project within the context of the System Engineering Life-cycle as advocated in the INCOSE system engineering handbook version 3.1.

<table>
<thead>
<tr>
<th>Concept Stage</th>
<th>Development Stage</th>
<th>Production Stage</th>
<th>Utilization Stage</th>
<th>Retirement Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.1 Concept Phase

\(^1\) Perhaps the simplest way to make a system software friendly is to include adequate amounts of system memory. Other basic concepts include consistent paradigms for HW access, processor choice that is high-level language friendly, moving highly time critical functions into hardware, etc.

\(^2\) One of the early roadblocks we had was the literal meaning of the word emulator as opposed to its use in the software debug industry. When HW engineers heard emulator they were very concerned about the work involved in “emulating” the entire system behavior. The name External Debug System was used to try to get around this problem.
As the platform project moved forward and some of the major architectural decisions were made, the External Debug System sub-project was started. System Engineering moved forward and created a series of presentations designed to trigger discussions on the fundamental requirements and how different solutions might work at the high-level. These eventually served the purpose of an informal “White-Box” Concept of Operations that tried to balance the functionality provided with something that could be achieved with an acceptable level of effort. The primary purpose of the new development was to support product features and it was very difficult to come up with budgets for levels of effort/cost that could be devoted to development needs. Metrics on the costs due to inefficient tools did not exist, and thus, many of the decisions had to be made on a qualitative basis. We proceeded with a set of iterative discussions between System Engineering and each of the following stake – holders:

- Management – Very concerned with costs. They also had a strong bias towards marketing the importance of the platform so were very interested in debug features could be turned into selling points.
- VLSI Development – Responsible for creating the IC that had to support the debug interfaces. They had existing IP and ideas on what the IC should be like, so they had strong opinions about what they would and would not support. The IC would be developed on an independent sub-project of the main platform effort.
- Software Development – These would be the final users of the debug system. The software team members came from diverse backgrounds and generally preferred to work with the tools they used last. Some thought features like breakpoints were useless, others thought they were essential, and others had never used the concept before and did not know what it was. There was no single voice for the group and they generally did not have the same set of values as the IC developers with respect to what would be an acceptable cost to support a given piece of debug functionality and would not.

At the highest level, the following features were desired:
- RAM/ROM Emulation
- Ability to watch memory locations
- Support for Customizations in the CPU Core and Bus System
- Ability to do single stepping and breakpoints in C and/or ASM
- Support for our chosen Compiler
- Support for Logic Analyzer style Deep Traces of system behavior
- Support for Profiling
- Freezing system on breakpoints
- Needed commercial quality GUI to debug with
- External automation support
- Possibility to connect the GUI to a simulation some day

The following compromises were acceptable to the software teams:
- Breadboard would be the primary target and we needed to optimize debug support for this. To deal with the issue of noise:

---

3 This feature was a very hard sell to the VLSI teams but very important to SW debug of real-time systems. It was very difficult for VLSI to understand the benefit of stopping certain digital parts of the system and not others.
4 Often during the project, HW design engineers would indicate certain features could be accomplished with an FPGA and a minimal amount of effort but would neglect to consider the work for creating the GUI and control software for the HW.
Hardware development would put a higher focus on achieving a low noise development board. Minimal Hybrid/Can debug support would be available based on the use of a downloadable software monitor program. The HW support for this would be available with first HW but the SW support would be deferred unless noise measurements indicated it was needed.

- Whatever support we provided for a hybrid/module needed to match the support we had in the canned device. This ruled out the addition of high-speed debug output that did not use the existing I/O mechanisms accessible in the final product.

- Whatever tools supported debugging on real hardware, would be different from the tools that supported debugging on simulation. Long-term there was a desire to have a consistent interface to debugging both targets, but resource limitations would not make that possible for this effort.

Figure 1 below shows a slide used in one of the final trade-off discussions with the Software teams. It shows four debug targets: Simulation, Breadboard, Hybrid, Canned Device and some of the major features and their applicability across debug targets. It is shown as an example of how the various trade-offs were worked through and how final buy-in from the software team was achieved.

Figure 1  Debug targets and supported features

Understanding the complete scope of what was agreed to is beyond the scope of this paper, but here are a few of the points that the slide (and the discussion around it) conveyed.
• Simulation (developed as a parallel sub-project to this) had the most powerful support. In particular, on a break point, all simulated inputs and chip behavior would stop. (called Freeze World in the slide) It also allowed real-time visibility of internal hardware and software events at many levels (Real-Time Watch).

• Breadboard was the next most powerful solution. Any system event visible on the external bus could be captured by the debug system and displayed to the user (Real-Time Watch Bus Only). Freezing was supported but only the digital features of the chips would freeze. This meant that users could single step through small portions of code but without interrupts piling up but eventually the software would get out of sync with the external system stimulus’s.

• Hybrid and Canned Device targets are shown with the same feature sets. Real-time watches are no longer possible due to connectivity issues and the main possibility to debug the system is by using breakpoints and single stepping. While stopped you can examine any memory location. As previously stated, this was to be possible, but would only be implemented if it truly needed based on the noise characteristics of the breadboard target.

• Two features were shown and have crosses through them. These were the last debug extensions that were considered possible, and were being shown to the software teams as a last chance to provide input as to whether this was acceptable or not. In the end, they were determined to be lower priority improvements for the software team and not cost effective to implement at this point.

From the VLSI perspective, the end-result of all the architecture discussions was that they needed to support the following features beyond their original proposal:5

• Adding patch registers. Patch registers was a mechanism that allowed CPU operation to be redirected on a patch hit, effectively allowing ROM to be “patched” in late in development should the need arise. The same mechanism can be used to implement breakpoints in a SW monitor program. 2 to 4 would be added.

• Freeze support on NMI assertion or patch register hit. NMI allowed an external system to trigger the freezing (an emulator).

In addition to high-level agreements on what needed to be supported, there was significant effort put into selling the concept of working with an outside vendor to purchase the system. Our company had developed custom debug systems several times over the years with some success but had little experience working with outside entities on such a project. Figures 2 and 3 below show some of the slides used to promote acquiring a solution rather than building it.6

5 Their original proposal included supporting a series of test modes that allowed the processor to be switched from using internal resources for RAM/ROM to using some combination of using internal resources and external resources. It also supported mirroring internal bus operations onto the external bus during these test modes. Both features are critical for RAM/ROM emulation and use of the system with a Logic Analyzer or Emulator.

6 It was important to sell the concept not only to management but to engineering. This way management would get the same input not only from the new platform project team but from the potential users of the system.
This theme was continually pushed right up to the point when the contract was signed.\textsuperscript{7}

5.2 Development Phase

\textsuperscript{7}As late as a week before the contract was signed, there was an engineer suggesting we build our own solution using Eclipse and a FPGA.
5.2.1 Project Planning

As this was not a product development effort, existing process documents had little guidance on what sort of processes/documentation was required. Thus, the process tailoring was used to determine what was needed.

The high-level process followed was:

- Vendor Selection
  - Results and reasoning presented to management.
- Contract Negotiation with selected vendor
  - Management sign-off was required at relatively high-level in order to proceed.
- System Development
  - Requirements Development
  - Technical Workshop for System Design and Knowledge Transfer
  - One or more Prototype Deliveries of the System
  - Acceptance Testing of Prototype Deliveries
  - Final Acceptance By Customer
- Delivery of Production Systems

The planned documentation was to include the following:

<table>
<thead>
<tr>
<th>Document</th>
<th>Responsible Party</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q&amp;A Document from Each Vendor</td>
<td>Software Development</td>
<td>Formal Q &amp; A were asked of both vendors and recorded for future reference in the contract.</td>
</tr>
<tr>
<td>Technical Report Comparing Selected Vendors</td>
<td>Software Development</td>
<td>As the users, Software Development engineers took the lead on comparing the products to come up with a preference. However, it was made very clear that the final solution would be chosen based on Cost, Features and Intangibles. Discussed further later.</td>
</tr>
<tr>
<td>Development Contract</td>
<td>Supply Chain</td>
<td>Supply chain was responsible for authoring the contract with input from system engineering and management.</td>
</tr>
<tr>
<td>Requirements Document</td>
<td>System Engineering</td>
<td>A high-level requirements document was written as a basis for performing acceptance testing.</td>
</tr>
<tr>
<td>Design Document</td>
<td>System Engineering</td>
<td>This document served several purposes: - document the design of the system to support trouble-shooting, debug and recording of design decisions - a single point of reference for the emulator vendor on our chip operations</td>
</tr>
<tr>
<td>Verification Plan</td>
<td>System Engineering</td>
<td>Documented acceptance testing for the emulator.</td>
</tr>
<tr>
<td>User Manuals</td>
<td>Vendor</td>
<td>Customized versions of the user manual.</td>
</tr>
<tr>
<td>Verification Summary</td>
<td>Vendor</td>
<td>Summary of all tests performed on</td>
</tr>
</tbody>
</table>
the emulator by the vendor.

<table>
<thead>
<tr>
<th>Product Description</th>
<th>Vendor</th>
<th>High-level product description.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installation Guide/Tips/Help Information</td>
<td>System Engineering</td>
<td>To simplify setup, a web page was created on the corporate intranet which detailed how to setup the emulator. It also had pointers to other documentation and an Emulator FAQ.</td>
</tr>
<tr>
<td>Emulator Training</td>
<td>System Engineering</td>
<td>Originally it was decided to rely on the vendor for training but many of the questions boiled down to understanding the combined system of the emulator with the breadboard target. Thus internal training was created.</td>
</tr>
</tbody>
</table>

However, the real challenge of the project with respect to planning, was managing the schedule because of the tight dependency to work/deliverables associated with other platform sub-projects. The following table discusses several schedule/project risks and how they were managed. The comment column discusses what happened on the project with respect to that risk.

<table>
<thead>
<tr>
<th>Risk</th>
<th>Mitigation</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Initial schedule highly dependent on the platform project schedule for delivery of first silicon. | Minimized the dependency on Platform ICs to a single IC.  
Contract was written such that the bulk of the cost of the project would not be paid until final acceptance and delivery of production systems.  
Choose vendor who demonstrated flexibility and a strong desire for our business. | Biggest issue turned out to be getting final buy-in from management in order to sign the contract. This delayed signing the contract until silicon timeline was less risky. |
| Show-stopper in first silicon meant it was unusable.                | Minimized the dependency on Platform ICs to a single IC.  
Contract was written such that the bulk of the cost of the project would not be paid until final acceptance and delivery of production systems.  
Choose vendor who demonstrated flexibility and a strong desire for our business. | A show-stopper bug was found in first silicon by the emulator vendor. Vendor was very flexible on agreeing to implement a work-around that allowed the emulator acceptance testing to proceed despite the problem. |
| Schedule slips by the emulator company.                             | Contract was written such that the bulk of the cost of the project would not be paid until final acceptance and delivery of production systems.  
Choose vendor who demonstrated flexibility and a strong desire for our business. | Emulator company was very responsible but tended to ship features before they were ready without indicating which features were not ready. This made us move to a model were updates were screened before they were adopted by most users. |
| Unforeseen technical issues in the IC designers involved in the early profiling was one of the last emulator | IC designers involved in the early profiling was one of the last emulator | |

Page 11 of 21
interface between the emulator and product IC

system discussions.
Early testing on first silicon when changes were possible.
Contract was written such that the bulk of the cost of the project would not be paid until final acceptance and delivery of production systems.
Choose vendor who demonstrated flexibility and a strong desire for our business.

features to be available and it did run into interface issues that required small but late IC changes. The decision was made to purchase all remaining systems and make final payments. This meant getting the last emulator updates were based on good-will and the possibility of us ordering future systems.

5.2.2 Vendor Selection Process
A key decision was to go with a company which manufactured emulators for processors as similar to ours as possible. While there were only a relatively small number of companies which met this criteria, it was decided to focus the bulk of the analysis on two companies. This balanced appropriate due diligence with the desire to be able to move forward quickly.

The vendor selection process was based on three major components:
- User Analysis of Feature Set based on Vendor Q&A and evaluation of full systems
- Quoted NRE and per Unit costs
- Business related intangibles such as support during the evaluation process and perceived importance of our business

The results were mixed as shown on the following table:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Company A</th>
<th>Company B</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features/usability</td>
<td>Stronger</td>
<td>Acceptable</td>
<td>Company A was the clear favorite for the software engineer who performed the evaluation. Company B was acceptable but was missing some of the higher-end features.</td>
</tr>
<tr>
<td>Cost</td>
<td>No NRE. Fixed Cost per unit with a minimum number of units.</td>
<td>Very flexible on options and on pricing.</td>
<td>Company B was better based on the flexibility.</td>
</tr>
<tr>
<td>Intangibles</td>
<td>Limited contact. Slow response.</td>
<td>Very pro-active. Definitely appeared to want our business.</td>
<td>It was determined that this was the most important criteria.</td>
</tr>
</tbody>
</table>

Based on the above, the decision was made to go with company B contingent on them bringing their costs down to make them a clear winner on two counts.  

---

8 This turned out to be a very good decision as the flexibility and desire for our business were important parts of what made the project a success.
5.2.3 SE Requirements

As we targeted companies selling commercial emulator systems on similar processors, we could look at their commercial products to understand how the system would operate from a user's perspective, and what the major features were. It was also pretty clear that the company we choose was used to less formal development practices and certainly was not used to formal requirements specifications. This motivated us to keep the requirements at a relatively high-level dealing with the core features we needed supported and covered by the contract. We then treated early emulator releases as prototypes and worked with the vendor to make changes as appropriate.

In particular, the system was capable of monitoring memory contents using either real-time DMA or something called shadow memory. Real-time memory was powerful as it could monitor memory addresses that were never read or written to (a HW count down timer) but also resulted in cycle stealing from the CPU. Shadow memory captured the value written to or read from the memory it shadowed, and was probably ideal for most normal memory. As all this was platform specific, the original proposed method of configuring the system was to use a configuration file. This seemed very confusing so we agreed up on the following rules.

1. By default memory would be shadowed.
2. On option would be available for using DMA for reading any memory location that was not shadowed. By default this would be off.
3. The user would have the ability to exclude certain memory locations from being shadowed. If the box above was checked, they would then be read by DMA.

This solution simplified the vendors work and allowed us to put all the UI controls configuring this behavior on a single dialog box.

5.2.4 SE Architecture

The following chart shows the context of the External Debug System with regard to the larger system in which it is enclosed. The purpose is to establish context for subsequent discussions.
The following sub-sections give an overview of the major components.

5.2.4.1 Platform Breadboard
The following is a simplified model of a platform system as setup on a breadboard.

Note: The focus is on features related to the external debug system rather than product features.
The system consists of the following:

- **High Frequency Oscillator** - The on-chip high frequency CPU clock generator (CCG) offers an adjustable frequency up to 3 MHz. It is started and stopped under control of the CPU and other bus masters.

- **Clock Generation** - A four-phase bus system has been implemented. The bus clocks are generated either using the high frequency oscillator or the 32kHz system clock, depending if the CPU is running, or if another bus master requests the higher clock speed.

- **CPU and Support Units** - A micro controller derived from a commercial processor but with some custom extensions was created.

- **Other Bus Masters** - The CPU and the Multipurpose DMA are examples of bus masters which can request the bus from the system and write to any memory mapped address via its 24 bit address. Some bus masters can also request the Burst Clock to run in a similar manner to the CPU.

- **Passive Peripherals** – Most peripherals in the system are passive in the sense that they do not make bus requests but are accessed via the bus. In the case of some units (counter/timers) they can change asynchronous to the CPU control flow, in other cases they must be read or written to change value. For the purposes of this model, this block includes on-chip RAM and ROM.

- **External Test/Bus Interface (ET)** is designed for system enhancements by connecting additional external devices like an analog IC or an external RAM as well as for test purposes. It features a 4 phase external bus transfer for high reliability. Several different configurations allow support of standard mode, test mode, software debug mode with external ROM or RAM, trace mode. Internal and external bus are decoupled for low power consumption. The ET is used as debugging interface for connecting to the EDS.
• Other IC’s – The system is designed to have multiple IC’s which use memory mapped registers and specialized I/O lines for inter-IC communication. Access to an other IC via the memory bus can be triggered by SW in the form of a CPU read/write (like most systems) but it can also be triggered by hardware bus master which requests the bus to signal an event to the other IC. For the purpose of the EDS system, it can be considered a block configured by a set of memory mapped registers providing at least these services:
  • Voltage and current references
  • Power supply and low-voltage reset detection
  • Interrupt source and reset source
  • Source of 32 KHz system clock

5.2.4.2 External Debug System
The external debug system also consists of multiple components that work together to form a complete system.
  • GUI – This is the GUI interface that runs on a windows PC that controls the debug system hardware according to the users needs by presenting a high-level interface that is consistent across debugging targets. The GUI is target aware in the sense that it needs to be modified inorder to support our unique HC11. It is also a crucial part of the system responsible for performing all tasks related to taking raw trace/memory contents and relating it intelligently to the target source code.
  • IC3000 Active Emulator/iCARD/HC11 ActivePRO POD – The debug hardware is based on a modular design concept enabling reuse at multiple levels. Their systems consist of a general purpose communication system that ships data/commands to/from the emulation hardware to the host pc and the GUI. This has a card slot in it that houses a target specific connection hardware for the “system under debug”.
  • iCONNECT- External automation support for the system. This would allow COM – aware clients to externally control winIDEA and thus the operation of the External Debug System.
  • iOPEN – This is a more complicated interface that allows clients to plug into the debug system software infrastructure. This would be what would be used if we were to interface the debug system to a simulation.
5.2.4.3 Aux. Test Equipment

It is important to note, that the emulator will not be the only piece of equipment attached to the target. Typically development configurations require the use of various stimulus devices that mimic real-world inputs in order to sequence the device through its various states and software execution paths.

It should be possible to connect a Logic Analyzer to the system while the debug system is attached. This is critical to find problems that are related to debug functionality or timing problems that appear to only occur while the breadboard is attached.

5.2.4.4 External Control Software

More and more of our tests are planned to be automated which means that the larger system includes test applications that control the simulators and possibility the debug system in order to accomplish testing objectives. Controlling the debug system would be done via the iConnect interface.

5.2.5 Basic Operation

The basic principles being used by the debug system can explained by examining following major use cases:

- Breakpoints
- Trace
- Memory Monitoring

The following sections discuss each use case in turn.

5.2.5.1 Breakpoints

Breakpoints are supported by a combination of debug system and target breadboard features. To perform a breakpoint, the following actions occur:

- The debug system monitors the external bus for device operation
- After reset, the debug system uses the external bus to enable system freezing by direct memory write.
When a fetch from a breakpoint location is seen, the debug system asserts the NMI and waits for the appropriate point in the timing cycle to change the TMI pins to cause the system to run from breakpoint memory. The system runs from breakpoint memory allowing the debug system complete control of the CPU operation. The NMI will activate the freeze mechanism. By watching the stack on asserting the NMI, the debug system will know the contents of all registers.

While in the breakpoint:
- The breakpoint monitor program grabs data from memory inorder update the display of the device state.
- The breakpoint monitor program services the CPU watch dogs to prevent a CPU reset.
- The breakpoint monitor program disables memory write protection to allow the user to change memory during a breakpoint.

To exit a breakpoint:
- The debug monitor unfreezes the hardware by writing to a register
- The debug monitor executes an RTI routine and then switches the TMI pins in a controlled manner to allow the system to resume.

Implant software responsibilities:
- The stack must be correctly initialized by the ROM/RAM to allow breakpoints.
- To prevent reset loops, the debug system will wait until the LDS instruction is executed before allowing debug breakpoints

5.2.5.2 Bus Trace

Tracing the Bus for monitoring software execution or more general bus activity is accomplished as follows:
- The emulator monitors the address, data, BMACK and other control lines to catch data and store it to on debug system hardware memory along with appropriate timing information
- The PC and the debug system hardware then work together to ship this data up to the PC in real-time. As long as the rate of the data being collected does not exceed the communication link bandwidth, continuous traces of execution will then be stored to the hard-drive of the PC in a compressed format.
- The GUI then relates this raw trace info to the C code based on debug information exported by the compiler.

---

9 The concept of breakpoint memory is introduced to avoid confusion. The vendor would probably say the system switches to emulation memory BUT instead of running target code in that emulation memory, the system runs debug code swapped in by the emulator. The emulator is capable of switching the debug code in and out of emulation memory as it needs.
5.2.5.3 Memory Monitoring

During breakpoints:
- As the system is stopped, the debug monitor reads out memory on demand by the PC application. Shadow memory is not used.

In-real-time will executing:
- To minimize impact on the system, the debug system will contain a 8 Mbyte shadow memory that records what is read and written to any shadowed piece of RAM.
- The user will be able to remap this shadow memory as needed over configurable memory ranges should more than 8 MBytes be needed.
- By default, shadow memory will be used for all memory monitored regions, however, the user can override this by configuring the emulator to use DMA for regions not shadowed and excluding certain regions from shadowing.

5.2.6 Verification and Validation

The vendor was responsible for providing a working system which included performing their own testing on the product. Our company would perform acceptance testing to ensure that the system worked in real environments (Validation). Testing was done to the high-level requirements created early in the project but also focused a lot on expert ad hoc testing, as we methodically tried to use each feature in our environment.

Our testing was heavily influenced by the decision to only provide high-level requirements. This meant that the testing had to be done by engineers familiar with both our internal systems and the operation of the EDS system. The project heavily relied on the background of the Systems Engineer in embedded software development to go beyond the stated requirements and push into the details.

In addition to the vendor testing and the acceptance testing, product quality was further examined by early use of prototype emulators by the software development teams themselves. The releases they used were pre-screened but they did find some early reliability issues were found when tests were run overnight. In addition, the provided feedback on what level of training and documentation were needed to come up to speed with the tool.

5.3 Operations Phase & Maintenance Phase

The operations phase of the project is currently ongoing. The main SE relevant aspects are the following:

- A website was created on the corporate internet to help people install the product and point to the essential documentation.
- A training presentation was put in place that educated users on both the emulator and the IC to which it attaches to as an understanding of the complete system is necessary to be an effective user. The vendor also offered training on some of the advanced features but this has not been necessary.
- Settings and the version of the emulator software deployed were standardized across development. New releases are prescreened in order to not cause inefficiencies in deployment.
- The System Engineering responsible for the emulator development is currently doing emulator related trouble shooting and is the go to person for emulator related issues. Generally, an attempt is made to solve the problem locally before going to the vendor. Long-term this role will be transitioned to a member of the software engineering team.

5.4 Retirement Phase

The system is planned to be used at least for the duration of the platform. No special retirement issues exist.
6  PROJECT RETROSPECTIVE

Stepping back, the project was a great success. 21 initial systems were ordered and 10 more systems are about to be ordered. NRE and system costs added up to approximately $500K making it a significant financial investment and this number excludes internal resources spent on development and management of the effort.

What went well:

- Executive sponsorship at the Director Level. The concept for the project came from a software/systems engineer but the director responsible for the platform liked the concept and helped support it to get buy in at the executive level. Since something like this had not been done before, it was very important to have management support through-out the project.

- The primary system engineer working on the project was also a software engineer and thus had a strong background in what was important for software engineers and how such systems could operate. He was also an electrical engineer and able to talk to the VLSI and Vendor Hardware engineers to suggest issues that needed to be dealt with.

- The VLSI engineers who designed the IC participated directly in many of the conversations with the vendor. Direct communication between the engineers who designed the IC and the debug system answered many questions quickly and effectively.

- Early in the pre-concept phase, some software engineering groups wanted to keep using an in-house solution. Ultimately their efforts were unsuccessful as they were unable to free up resources to deal with the support issues. They then became supporters of the project and helped get the backing from VLSI that was so successful above.

- The vendor started working with the IC very early and found a problem much more quickly than it would have otherwise been found. This allowed it to be fixed sooner than would have been otherwise possible helping keep the main project on-track.

- The vendor was very flexible and responsive to our needs.

What could have worked better:

- A dedicated technical resource to test the emulator. At times, the project was delayed unnecessarily because resource constraints meant we did not give feedback in a timely manner.

- Early emulator prototype releases had issues. Often a fix of one feature would break another feature. This is because we were using test releases that had limited testing before being past on to us. Here the vendors talent for fast turn-around of problems was also an occasional weakness.

- IC turn-around times for changes were greater than 4 months. Much faster turn-around times could have been achieved if the company had used FPGA's for prototyping. This was suggested but not feasible given resource constraints and tight VLSI schedules.

7  ROLE OF SYSTEM ENGINEERING

There are lots of definitions of “what is systems engineering” and attempts to define what system engineers do. This section makes some comments on what the system engineer did on this project and what they did not do.

Starting with the later, system engineering did not:

- Design the emulator. The emulator itself was treated as a black box.
- Design the IC or breadboard used to connect to the emulator.

System engineering did the following:

- Worked with users and came up with the vision of the debug system based on need and what was technical feasible.
• Marketed the vision to engineering and management to get the sub-project going. System Engineering texts often talk about the realities of politics for government projects. In my experience politics is a reality in any organization.

• Worked with all stake holders to find an appropriate concept of operations that had buy-in from all sides.

• Marketed and sold the vision to management and other users. This was an ongoing responsibility.

• Month to Month project management and controlling. A project manager existed for the platform project but was heavily involved in the more product oriented sub-systems. The project manager filed the status reports and kept the overall schedule documentation in order.

• Drove the technical part of the contract and some of the negotiations. All wording and legal aspects were handled by a Supply Chain Management group. Final price negotiations were also done by the Supply Chain Group.

• Participated and facilitated all technical discussions on the interface. Details of exact bus timings were handled by the IC design team. Final algorithms for the emulator were the responsibility of the vendor, but system engineering helped define the interface between the two systems and what they key responsibilities were.

• Authoring and maintaining project documentation.

• Main point of contact for all groups with regard to questions on the emulator.

• Reviewed platform development deliverables to ensure debug system support was appropriately considered.

• Acceptance testing on the emulator to ensure major features met requirements and needs.

• Training of users on the new system.

8 CONCLUSION AND PARTING WORDS

This was a very rewarding project to be involved with as it has been a great success with the users. I think it is a great example of systems engineering but it also is part of a larger systems of systems engineering vision.

• By interfacing this debug system to the custom simulation we would have a much more powerful simulation environment with a consistent interface between it and the hardware development environment. It would also simplify the maintenance effort for the simulation by removing support for duplicated features.

• The emulator has external automation capabilities which would allow it to be incorporated into our automatic test system. Automated tests could take advantage of:
  o DMA for rapid download of program code
  o Watches for real-time memory logging
  o Profiling for Performance Monitoring of software
  o Coverage analysis for analyzing test coverage analysis
  o etc

• Once the above features are in place, we could move to performing something called CPU-driven Testing to allow us to write tests for our simulation that could be re-run on HW using the emulator as front-end for test downloading and execution. In fact, many tests could probably be written without actually downloading code and instead using the DMA to manipulate hardware register contents directly.

Thus beyond this project, there are significant opportunities to further integrate this system into our other systems.