Microprocessor and FPGA interfaces for in-system co-debugging in field programmable hybrid systems

M.A.Aguirre, J.N. Tombs, V.Baena-Lecuyer, J.L. Mora, J.M. Carrasco,
A. Torralba and L.G. Franquelo
Dpto. de Ingeniería Electrónica. Universidad de Sevilla (SPAIN)
{aguirrre,jon,baena,mora,carrasco,torralba,leopoldo}@gte.esi.us.es

Abstract.- New trends in technology require efficient control and processing platforms based on connected software-hardware subsystems. Due to their complexity and size, algorithms implemented on these platforms are difficult to test and verify. When these types of solution are being designed, it is necessary to provide information of the internal values of registers and memories of both the software and hardware during the execution of the complete system. The final architecture of the targeted design and its debugging capabilities strongly depends on how the hybrid system is connected and clocked. This article discusses different architectural strategies that have been adopted for a hybrid hardware-software platform, built ready for debug, and that uses components that can be easily found with a few special features. All the solutions have been implemented and evaluated using the UNSHADES-2 framework[1].

Keywords: Hybrid System, Co-design, Co-debug, Rapid prototyping.

I. Introduction

The requirements of many of the industrial applications, such us power electronic control systems, sensorless encoders, video processors, communications systems and others, soon discover that solutions based purely on microprocessors can't achieve the required specifications or require extremely complex programs with their associated long development times. Almost always the only solution involves the migration to a more powerful (and therefore more expensive) microprocessor or DSP.

From the application point of view, the solutions are usually based on a control scheme; the data coming from electronic sensors or being sent to driver controllers is intensively processed and then used in software loops that are dedicated to the high level decisions. The program becomes very complex because the time devoted to each task is uncertain and difficult to organise. Solutions based on application specific hardware and software machines have been extensively used in the past for this kind of system, but recently field programmable devices are substituting the custom circuits. The design of Application Specific Integrated Circuits (ASICs) has historically been a common solution, which now

[1] This work has been developed under the framework of the project "New intelligent actuators for electrical rotative and linear motors, using sensorless techniques in applications of vertical traffic" Spanish Science and Technology Ministry - DPI2001-3089.

becomes economically unfeasible for low cost systems except when the number of samples built per year is very high (typical number of over 100.000 units). The solution that provides the lowest costs per unit is that of the user customisable circuits or programmable logic devices. Today, we can find in the electronics market many Field Programmable Gate Arrays (FPGAs). These user programmable solutions are capable of performing the hardware part of a design for a significantly low price, and maintain many of advantages of the ASIC solutions. In this paper we explore a flexible platform based on two common interconnected devices that can be easily found at competitive prices. One device is a FPGA to implement hardware tasks and the other a DSP for software tasks. However, due to their increasing presence in the market, we should also mention solutions based on single programmable devices.

These devices have recently been introduced into the market under the name of Programmable Systems on a Chip (PSOCs). These devices have a digital programmable section, a microprocessor core, and a programmable analog tile all embedded into them, often together with other specialised modules for communications. The earliest example of such a solution was the FIPSOC [13][14] device. Another early example of this type of integrated circuits are the Triscend E5[23] devices (which include an 8 bit 8051). The most modern and largest in size and performance devices are the E7V CSoC from Triscend [24], Excalibur [12] from Altera, Virtex-II Pro [25] from Xilinx and FPSLIC from Atmel[5]. In these devices the embedded microprocessors are high performance CPU's (like the ARM7 for the Altera, PowerPC 445 for the Virtex and an AVR for the Atmel). Other devices from other vendors are continually arriving. All these new devices can be considered single chip implementations of the hybrid system, and therefore any conclusions and results generated could equally apply to these devices.

FPGAs are devices that can be configured with the functions and performance of almost any digital circuit. There are devices which can replace digital systems with as many as 10 million system gates (with 1 gate equivalent to a 2 input NAND), all this with a reasonably high performance level (e.g. 200MHz or higher). FPGA foundries offer many built-in circuit features, such as memories, multipliers, high speed communication links, and others to accelerate designs without inefficient use of programmable logic blocks. The reconfigurable nature of theses devices bring new system advantages to the manufacturer: FPGA designs can be field upgraded just like software, while ASICs are fixed solutions, there is no way to fix a mistake or improving the design once it has been made; faulty devices can only be replaced by new expensive devices. But probably the most interesting characteristic of FPGAs is the possibility of quickly obtaining a rapid and fully functional prototype that can emulate and verify solutions, or even be embedded into the final system.

The co-debugging of hybrid hardware-software systems consists in the resolution of two key problems; the **observability** and the **controllability,** during execution time. The efficiency of the debugging scheme depends on how well both tasks can be implemented. The debugging tasks are normally based on frameworks that provide comprehensible information about the internal processes of the design that is executing. In terms of software debugging, these frameworks make the values of randomly selected variables or expressions visible during the program execution. When the program is compiled for the same processor as the development framework, the processor executes directly every statement, either waits

or continues under user control. Typical debugging commands are “go until line x”, “run one step”, “stop when the condition x is satisfied”. Also, while the software is being debugged, it is possible to produce modifications in variables and carry the system to otherwise unfeasible situations. In terms of hardware the situation is not so clear and no single commercial platform exists that is dedicated to the hardware design process through real time debugging. To implement such a tool, a hardware platform and a link with a host computer are needed. In order to understand the hardware state a complete software toolbox is needed to present state information in an appropriate manner, usually by linking it with the signal names given during the high level design description phase before synthesis.

In this paper we explain how a hardware debugger can be made for a general-purpose hybrid system; the system consists of a software part running on a microprocessor or DSP, and a hardware part represented by a FPGA. Our approach is focused on common commercial devices with very few special features. We describe our experience obtained with the platform UNSHADES-2, this system has been designed with special purpose hardware resources to support the debugging task. In section II previous work on debugging over hybrid systems is addressed. In section III the essential debugging features is described. In section IV the system clock and the hardware-software interface is analysed due to the great impact on the platform behaviour. In section V the proposed design flow for the insertion of the debugging tasks is described. Finally section VI describes the platform UNSHADES-2, our hybrid hardware/software platform.

II Previous Work

Very little work can be found in the literature on how to orientate the methods and requirements for the debugging of a hybrid hardware-software system. The solution for debugging for the software part is old and well known, whereas the hardware debugging has never been so deeply studied. In this section the problem of hardware debugging is formalised in terms of the possibilities currently offered by the state of the art.

In general, hardware debugging is a task that has only been reported using ad hoc techniques. Special extra logic and special debugging modules are required. The presence of these additions degrades system performance and is usually referred to as the *overhead*. The main goal for a hardware debugger is to obtain an approach that minimizes the debugging overhead of the complete system.

In [22] the authors address the problem of co-debugging in hybrid systems. For their approach the CPU is merged with the hardware part into a single netlist, and the complete system emulated by a single FPGA. Finally the system can be observed and controlled using their custom tool named JHDL [17][6], a new special design language has been developed together with an interactive debug framework for configurable hardware. The approach has been developed on the assumption that the microprocessor is a soft IP core, and the internal structure and its registers are known, an HDL description of the processor is required.

PSOCs devices have built-in debugging schemes that treat the problem using separated probe circuits. A very important feature for the implementation is the clocking scheme of the

complete system. In the case study [22] both parts work synchronously with the same clock signal on the same device and the behaviour of both parts is therefore completely deterministic.

Some hardware debugging approaches can be found in the literature and commercial tools can be found, offered by the FPGA vendors. There are two basic solutions available for solving this problem, the logic analyser approach and the capture method. The former consists of including with the design a logic analyser module, this module records some selected signals, during a defined amount of clock cycles. The recording process is triggered when a run-time condition is satisfied, thus this condition has to be foreseen before the synthesis phase. The main advantage of this approach is that many clock cycles can be recorded and observed, however the recording memory has to be taken from the unused resources of the FPGA and its sizing depends on how many signals that are to be recorded and on the number of samples required. Moreover, the presence of the memory embedded with the circuit in the same device implies that the total performance of the system will be decreased when compared with the original circuit implementation. This approach is supported by the commercial package Chipscope from Xilinx[8] and SignalTapII and SignalProbe from Altera [10].

The other approach for the observation mechanism is described in more detail in the next section. Special tools have been developed that exploit the Capture and Readback feature of the Virtex, Spartan-II, Virtex-II and Spartan-III families from Xilinx devices. In [18] the XHWIF system is an interface tool between the board and the JBits CAD suite. JBits is a tool that manages Java classes and permits the user manipulation at binary *bitstream* level of the FPGA configuration information using dynamic modifications. By using this toolbox it is possible to implement partial reconfiguration of the FPGA device during runtime. Another approach of this class is UNSHADES-1 platform [1]. This platform adds to the previously described features, a framework that allows the user to interactively perform read-modify-write of the captured state.

III Co-debugging in hybrid systems

3.1 Hardware Debugging

Debugging of hardware structures is mainly an observability problem, the goal is to know the state of the processing tasks at a particular instant of time. In the previous section we mentioned two approaches, one based on the insertion of a custom logic analyser core and the other based on an internal state capture feature.

The state capture is based on a special feature that is particularly found in the advanced Xilinx devices. This feature is called *Capture and Readback* and is more deeply described in [26]. It consists of the presence of a half memory cell associated to every single flip-flop of the device. The structure includes a mechanism that can simultaneously transfer the contents of all the device flip-flops into that memory during a single clock edge. The transfer is launched whenever a run-time condition is satisfied. In other words, we can obtain a snapshot of the device state on a particular clock cycle, and after this transfer, the recorded state can be read through the dedicated device configuration circuit. The main advantage of

using this mechanism is that it is completely non-intrusive, all of the capture and read-back circuit is already built inside the device, and there's no impact on the circuit performance. Secondly the signal selection can be made randomly, the reading process is made through the device configuration structure and is completely independent from the synthesis stage. The main drawback is that we only can record the state of a single clock cycle, but it is not difficult to establish a procedure for running a single clock cycle between read backs of the system state. A typical work plan is to run at normal speed until the debug condition is satisfied and, after this moment, it is possible to run step by step recording the desired signal values in an asynchronous way. The Xilinx design flow gives enough information for de-mapping (back annotation) of the recorded bits into their names given during HDL design phase.

A comparison with the techniques based on the implementation of internal logic analysers [8][10] against Capture and Readback approach can be found in [15][16]. With the logic analyser implementation internal signals have to be selected previously to synthesis. Secondly, it is an intrusive technique as some resources modify their behaviour due to the presence of this debug core. The signals are recorded during execution, but normally the system speed has to be reduced due to the debug overhead. On the other hand, the capture and readback approach has two drawbacks, first only a single snapshot of the complete system can be obtained from a run time design without affecting the system performance. Secondly, if more recording cycles are required, some control techniques have to be implemented in the circuit, as a finite time is needed after every snapshot in order to transfer the hardware state out of the emulator and into the PC debugger. If the platform runs inside a target system, the synchronisation will be lost during this process. However this situation is the desired solution for hardware debugger based on commercial devices, as the software processor will also need to be stopped if the internal state is to be observed. Additionally, and not less important, the internal state can be modified. In [1], a tool that implements a technique for producing these state modifications using the proprietary patents [2][3] is presented. This feature increases the user controllability over the hybrid system and allows more advanced debugging control, for example, a *break point* condition can be modified on-the-fly without repeating the synthesis stage of the design.

3.2 Software processor Debuggers

The debugging task for software processes is a mature field and it is very well developed and supported by means of advanced user friendly tools. We are only interested in those aspects that are useful for co-debugging in a hybrid system. Modern processors have an in-built non intrusive control port based on JTAG Boundary Scan (IEEE 1149.1). Accessing to the internals of the processor through this serial port is a relatively simple task. Every register value (program counter, accumulators, status and control registers,… etc) is observable, and in the same way controllable. Foundries extended the standard protocol of the JTAG port with new commands that allow observability and control of the processor execution. When an access is required the processor state has to be frozen either by executing a special instruction or external handling the clock signal.

Both subsystems are separately controllable and observable. The next step is to join the approaches in order to obtain a coordinated and predictable behaviour of the hybrid system. From the debugging point of view, and due to the fact that the debugging of the processor is based on obtaining a single value of the internal registers, a strategy based on snapshots is adequate for a hybrid co-debugging system analysis.

IV Interfacing and clocking

Due to the systemic nature of the co-design methodology [19], an essential set of decisions in a co-design system are the clocking scheme and the interface between the programmable logic section and the software processor. This classification was initially made by [9], but it wasn't made from the co-debugging point of view. **Figure 1** shows examples of the interface model that can be summarized as follows:

- Embedded in memory model (figure 1-a). This model is introduced in certain PSOCs, where the configuration memory cells and the flip-flop values of the hardware section are mapped into the processor memory scheme. The internal values can be inspected and controlled as a part of a CPU program, and sent to the host computer through serial port devices. Devices using this scheme are FIPSOC and Virtex-II Pro (in this device, the processor has an internal access to the configuration scheme called ICAP).
- Embedded in routing model (figure 1-b). This model is introduced in all PSOCs and a device from Xilinx XC6200. In this scheme the routing resources of the hardware section are mapped into the address-data ports of the processor. The main difference between this and the previous model is that as there is no direct path to the configuration circuit to be found, and the connection between processor and programmable hardware is not limited by the pinout between two devices. Unless the programmable hardware section has another inner inspection scheme, debugging using this scheme has to be done by means of dedicated resources for state inspection that has to be foreseen before hardware synthesis.
- Device mapped model (figure 1-c). In this case the FPGA is connected to the Address-Data and control lines bus. Typically, this model is used in co-design systems based on two separate devices and is limited by the number of pins dedicated to connect the interface. Debugging task strongly depends on the clocking strategy implemented for the system.
- Memory cache model (figure 1-d). In this model, a dual port memory device is used to interchange data between processor and FPGA, where data is written and read in random way. In such systems the debugging task for each part has to be done independently, because there's no synchronous functional relationship between them.

Clocking schemes are another essential decision in hybrid systems, because the co-design tools are strongly conditioned by how they are clocked. Two basic schemes are possible: synchronised or independent.

- In the synchronised model a single unique clock source is used in the system to feed both parts. Its main advantage is that the behaviour of the complete system is deterministic and predictable. From co-debugging point of view, this scenario is the desirable situation. In this model, the number of clock cycles needed for a

hardware task is well known by the software programmer and the number of code instructions between task launching and finishing of the task. Using this scheme, it is possible to stop the clock line, run a single step or clock cycle and obtain coherent information of the complete (soft and hard) system state.

- The independent model needs techniques for communications between software and hardware components such as polling data or interrupt lines, as it has no synchronisation between them. From debugging point of view, the clock phase between both parts is continually changing, thus making global clock control almost impossible.

V Design flow for co-debugging tasks

The design flow for co-design systems is a very complex problem that is currently still under research. Many subproblems have been addressed, from the definition of a unified language that would describe the behaviour of the system, to a set of approaches that would allocate the different tasks of each part in an optimal way [27][11][7]. Another problem is the partition and scheduling of tasks between the hardware and software processors [20].

Co-verification or co-simulation is also a complex task that has been traditionally solved by means of separate treatment of each part. However, new co-verification software tools for embedded co-design systems use a hardware model of the processor that is joined with the hardware section so that a conventional hardware simulator can be used for the complete hybrid system. In [4] a survey of the co-simulation environments are exposed and benchmarked. In [13][14] another approach using reconfiguration techniques was proposed for the co-verification problem: the simulator tool was replaced by the target device and the stimuli were injected to device inputs and the HW/SW processor behaviour recorded. In this case, the reconfigurable device works as a prototype of the system and acts as a simulator kernel.

5.1 Co-debugging flow

The first step is to decide which interface model is desired. This decision strongly conditions the implementation model for each task. The hardware subsystem always follows the scheme described in [1] that is summarized in the following paragraphs.

The approach is based on a strict control of the clock line. Before the implementation phase the high level code has to be modified in order to instantiate the clock enable signal or to implement a gated-clock of every flip-flop in the design that is going to be controlled. It is not recommended to handle the clock line directly, as it is often generated from DLL blocks that need phase control. New devices such as the Xilinx Virtex-II family have special clock buffers that can implement the gated clock. Due to the same reasons, this management can only be done in the software processor, if the clock is not internally used in a PLL.

In **Figure 2** all the essential blocks for the hardware debugger are depicted. Virtex flip-flops that belong to the design under debug can be controlled by means of their '*clock*

enable' input. Using this pin the system evolution can be totally or partially frozen without any risk of malfunctioning due to glitches or clock skew problems. If the *'clock enable'* input is only asserted during a single clock cycle, the system will perform a 'single step' evolution. In the UNSHADES system family an external IO line, called *'debug clock'* is used to allow the software to perform this single stepping of the S-FPGA (see figure 2). For this option, a small circuit that detects changes on this control line must be included in the S-FPGA design. About 147 equivalent system gates for this circuit is the overhead of the system. Using a second line, called the *'resume'* line, normal execution can be restored until the next run-time condition.

The system evolution can be frozen when a run-time event is satisfied. Run-time stop conditions sources have to be foreseen at design time. For example, they can refer to register contents that have enough interest that should be studied when the condition is satisfied. Usually they're comparisons with bus values or bits. More complex conditions can be introduced that are combined with time conditions.

After each rising edge of the *'debug clock'* line the flip-flop '*clock enable*' input is asserted during a single clock cycle, and a *'capture'* is launched. After this, the software uploads the information to be represented in the user interface. UNSHADES can launch a sequence for a configurable number of steps and record the information to be displayed in a classical waveform viewer.

In order to accelerate the recording process partial access techniques are employed. The bit allocation file provides the information that allows the calculations for accessing the Xilinx FPGA with the minimum information unit possible called a *frame*. A single frame contains the information of a single column of configurable logic units. The UNSHADES software exploits this property to access the minimum number of frames in order to access the selected signal set. These accesses to the S-FPGA are the bottleneck of the system and limit the effective clock speed possible when single stepping. In order to improve these tasks the link between the PC and the C-FPGA should be accelerated. The current version achieves 1.6MB/s using the parallel port in its EPP1.9 mode and 1.4MB/s using a V1.2 USB port.

VI Unshades-2

UNSHADES-2 is a case study that consists of a complete hardware and software co-design and co-debug system, ready to benchmark almost all the approaches exposed in section III for co-debugging, it uses two separate discrete devices. UNSHADES stands for UNiversity of Seville HArdware DEbugging System and is the second member of a family of boards dedicated to exploit the hardware debugging concept. It is an example that supports the concepts presented in [22]. UNSHADES-2 introduces enough flexibility to define almost all architectures for the connection and clocking of the hybrid system. Two Xilinx FPGAs, a DSP and a flash memory compose the bulk of the board. The first FPGA is dedicated to host the hardware section and forms part of the hybrid system (it will be referred to as the S-FPGA). The second FPGA, known as the C-FPGA, controls all the communications with the

computer and many board configuration options, including the type of clock line control. The advantage of using this second FPGA is that the delay between FPGA clock line and DSP can be compensated using internal phase handling modules such as DLL. The C-FPGA generates the clock for the S-FPGA and DSP. The selected devices for the board are:

- S-FPGA. A Xilinx Virtex-E XCV1000E-PQ240. This device is FPGA footprint compatible with many smaller devices and has a large amount of internal SRAM memory.
- C-FPGA. A cheap Xilinx Spartan2 device XC2S30-PQ144.
- DSP. A Texas Instrument TMS320VQ33. This device has floating point instructions in order to complement the hardware processing if needed.
- Flash memory. This acts as a ROM memory for the processor or a fixed memory for the FPGA data testing, self configuration and others tasks.

All the system is connected as described in **Figure 3 and a photograph of the system is in figure 4**. While the DSP is working, the S-FPGA can monitor the memory accesses, current instruction and addresses. At the same time DSP can host in its memory map some resources of the S-FPGA like memory blocks, registers, complex datapaths,..etc, like a dedicated coprocessor. The interconnect model is completed with an interrupt line for asynchronous communications between both devices.

The clocking scheme is supported by C-FPGA. The user can select the clock for both subsystems, its frequency and phase, it is based on a 80Mhz crystal oscillator that exploits the benefits of the internal DLL block that can be found in C-FPGA. Using a combination of DLLs the clock can be doubled, divided, and phase managed (for example, the clock line can work at 160MHz, but FPGA clock and DSP clock can be delayed 90º, 180º or 270º).

Models Supported by UNSHADES-2

From the system architecture point of view, the models supported are:

- Embedded in memory model. This model is partially supported in the way described earlier due to the banked access to the FPGA internals. It is used when the DSP needs to access data taken from the run time capture or directly manipulate the configuration bitstreams of the Virtex Configuration. By means of the general purpose I/O lines between S-FPGA and C-FPGA it is possible to form a path from the DSP to the *SelectMap* configuration port of S-FPGA in such a way that it can be read or written.
- Device mapped model. Using the S-FPGA internal resources, decoders for hardware tasks can be built and mapped directly into the memory.
- Memory cache model. S-FPGA has on-chip dual port SRAM memories. By using them, a data interchange can be done as required by this model.

Both clocking models has been implemented in UNSHADES-2 and tested in order to verify all the debugging schemes.

Results using UNSHADES-2

Benchmark examples have been implemented using, both Xilinx design flow and Code Composer tools for its implementation in several forms. Between all of them, the most successful design is found to be using the same clock line for both devices – with the clock

line of S-FPGA delayed 180 degrees for device synchronisation. In the non synchronised model, the values found in the hardware part when the DSP stops the system are often unexpected. The same case is found if the hardware part stops the microprocessor, where it has been found difficult to detect the current instruction code. In this case, the user should need control both clock lines by separately, which is not trivial. From the data interfacing model, the use of models based on registers are easier to debug because the internal memories of the FPGA have a very complex de-mapping process and no approaches has yet been reported to do this, in spite of the fact that Xilinx has given sufficient information to build this tools.

A real world test case

The UNSHADES-2 capabilities as a hybrid co-design hardware debugger were put to the test with a study of a previously designed ASIC known as SLESS[2]. This ASIC was designed as a low-cost substitute for an optical encoder in induction motors speed control drives [21]. By measuring the currents flowing in the motor phases, the ASIC estimates the mechanical speed of the rotor and generates the bi-phasic encoder identical to that of a mechanical encoder unit. The heart of SLESS is a datapath structure that performs the differential model equations for mechanical speed estimation. A block diagram overview is depicted in **Figure 5**. The datapath structure includes several registers, an 22+22 bits adder an a 14x10 bits multiplier. The test case focused on a series of improvements to the design which required modifications of the complex datapath and control structure.

Using UNSHADES-2 the effort of the design process of a highly complex ASIC such as SLESS can be extremely reduced. Moreover, it is possible to assure the overall functionality and capabilities of the ASIC by an extensively debugging process carried out using UNSHADES-2 platform. An example of the debugging framework is found in **figure 6**.

The design procedure of the improved SLESS design was done in the following steps:

- The architectural design based on ´C´ simulation results. The SLESS architecture, the speed control loop algorithm and the induction motor model was coded in the 'C' programming language. These simulations allow a first approximation for register, adder and multiplier dimensions.
- Architecture codification and VHDL simulation. The architecture was coded in VHDL and several simulations was carried out to assure a correct codification.
- Using UNSHADES-2 debugging. The S-FPGA was configured with the ASIC architecture. A motor model together with the speed control loop algorithm was compiled for loading into the DSP. Running together, the real ASIC capabilities can be explored intensively without the expense or risk of the use of a real induction motor.
- The clocking scheme was the synchronised model. Datapath of the SLESS controller produces new actions to the processor after a predictable number of clock cycles. The processor computes output, again after a predictable number of clock cycles. This scheme allows an an easy control of input-output synchronisation useful for the debugging. Nevertheless

[2] The ASIC was originally developed under the framework of the FUSE project nº 25828 titled "ASIC for Sensorless Speed Control of Induction Motors: SLESS".

Thanks to this last design step, several implementation mistakes could be corrected. It must be noticed that SLESS performs an speed estimation based in a model reference adaptive system, so several differential equations must be resolved in fixed point codification.. In these cases is very important to keep the propagation of integration errors bounded. Using UNSHADES-2 the precision of the speed estimation algorithm could be evaluated and optimised for particular motor constants. For different motor constants, several registers where redimensioned as unexpected overflows in arithmetic operations where detected during debug. The simulations where repeated for several induction motor models and in one case an incorrectly dimensioned motor constant register was detected and corrected. **Figure 6** shows a screenshot of part of the user interface showing some of the important datapath registers in that instance.

It is important to realise that UNSHADES-2 platform allowed the debugging of SLESS in real-time, and in a environment very close to its real application. If UNSHADES-2 was not to be used, the SLESS architecture must be debugged by VHDL netlist simulations (together with a motor model). These simulations precise a very high computational effort leading to a an increase in design time and cost. The capability of the UNSHADES-2 system to freeze both DSP and S-FPGA simultaneously allows inspection of both the ASIC data path and C motor model without loosing the synchronisation between the two, something completely impossible with a real motor and ASIC prototype.

VII Conclusions.

A new task in the codesign design flow has been addressed for in-system and execution time verification. The interfacing requirements have been classified from the debugging point of view. A new co-design platform has been designed and built using commercial devices, where various different interfacing models can be tested.

By sharing a common clock source, it is possible to make meaningful debugging measurements from a hybrid system. This can lead to a deeper understanding of the system being debugged, and opens the possibility to perform intensive post-failure studies of the system when an unexpected result is produced.

From the application point of view, UNSHADES-2 is a powerful platform that can produce sufficient information about the execution system even without stopping the system by using the models explained in [1]. In this way a system can be monitored while working a full system speed, thus allowing optimisations of certain design characteristics in real world situations.

VIII References

[1] Aguirre M A, Tombs J N, Torralba A and Franquelo L G, UNSHADES-1: An advanced tool for In-System Run-Time Hardware Debugging. Lecture Notes in Computer Science. N 2778. Ed Springer-Verlag. ISSN 0302-9743. pp 1170-1173. 2003.

[2] Aguirre M A, Tombs J N, Torralba A and Franquelo L G, Method for the functional test of large digital circuits using hardware emulators. International Patent Number: W ES-02/00571

[3] Aguirre M A, Tombs J N, Torralba A and Franquelo L G, Procedure for the induction of register values in an emulated circuit using integrated hardware emulation, Patent Pending.

[4] Albretch T W, Notbauer J, Rohringer S, HW/SW CoVerification Performance Estimation & Benchmark for a 24 Embedded RISC core design. Proc. Of the Design Automation Conference San Francisco 1998.

[5] AT94K Series Field Programmable System Level Integrated Circuits. Atmel June 2002.

[6] Bellows P and Hutchings B, JHDL- An HDL for reconfigurable systems. Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, K.L. Pocek and J.M. Arnold, Eds. Napa CA, April 1998, IEEE Computer Society, pp 175-184.

[7] Bolsens I, De Man H, Lin B, Van Rompaey K, Vercauteren S and Verkest D Hardware-Software co-design of digital telecommunication systems. Proceedings of the IEEE vol 85 pp. 381-418, Mar 1997.

[8] ChipScope Pro Software and Cores user manual. Xilinx. UG029. May 2003.

[9] Compton K C, Hauck S, Reconfigurable Computing: A Survey of Systems and Software, ACM Computing Surveys, vol 34, n 2, pp 171-210. June 2002

[10] Design Verification Using the SignalTap II Embedded Logic Analyzer. Application note 280. Altera. June 2003.

[11] Ernst R, Henkel and Benner T, Hardware/Software Cosynthesis for microcontrollers. IEEE Design & Test of Computers vol 10, pp 64-75, Dec. 1993.

[12] Excalibur Device Overview. Altera. May 2002. ver 2.0.

[13] Faura J, Horton C, Van Duong P, Madrenas J, Aguirre M A and Insenser J M, A novel mixed signal Programable device with On-chip microprocessor. Proc. Of the Custom Integrated Circuits Conference 1997. CICC'97.

[14] Faura J, Moreno J M, Horton C, Van Duong P, Aguirre M A and Insenser J M Multicontext Dynamic Reconfiguration and Real Time Probing on a Novel Mixed Signal Programmable Device with On-chip Microprocessor. Lecture Notes in Computer Science. N 1304. Ed Springer-Verlag. ISSN 0302-9743. pp 1-10. 1997.

[15] Graham P, Nelson B, Hutchings B, Instrumenting Bitstreams for Debugging FPGA circuits. 2001 IEEE Symposium on Field Programmable Custom Computing Machines, California April 2001.

[16] Graham P Logical Hardware Debuggers for FPGA-Based Systems, PHD Dissertation. Bringham Young University. 2001

[17] Hutchings B, Bellows P, Hawkins J, Hemmert S, Nelson B and Rytting M, A CAD suite for high performance FPGA design. Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, K.L. Pocek and J.M. Arnold, Eds. Napa CA, April 1999, IEEE Computer Society, pp 12-24.

[18] JBits tutorial. JBits v2.7. Xilinx. 2002.

[19] Koch A, A comprehensive Prototyping Platform for Hardware-Software Codesign, Proc. Of IEEE International Workshop on Rapid System Prototyping Paris, France, June 2000, pp 78-82.

[20] Mei B, Schaumont P, Vernalde S, A Hardware-Software Partitioning and Scheduling Algorithm for Dinamically reconfigurable Embedded Systems. Proc of the ProRISC/IEEE Workshop. Nov 2000. pp 405-411

[21] Mora J L, Tombs J N, Pachon R, Torralba A, Barranco M, Franquelo L G, ASIC-based tachometer without mechanical transducer for induction machines, IECON '99 Vol 3 , pp 1039-10044, Nov. 1999

[22] Roesler E, Nelson B, Debug methods for Hybrid CPU/FPGA Systems. Proceedings of the First IEEE International Conference on Field Programmable Technology (FTP), pages 243-251, December 2002.

[23] Triscend A7S Configurable System On Chip Platform. August 2002.

[24] Triscend E5 Customizable Microcontroller Platform. March 2003.

[25] Virtex-II Pro Platform: Introduction and Overview. Xilinx. August 2003. ver 2.4.2

[26] Xilinx application notes 138 and 151.

[27] XC6200 family datasheet. Xilinx. 1996.

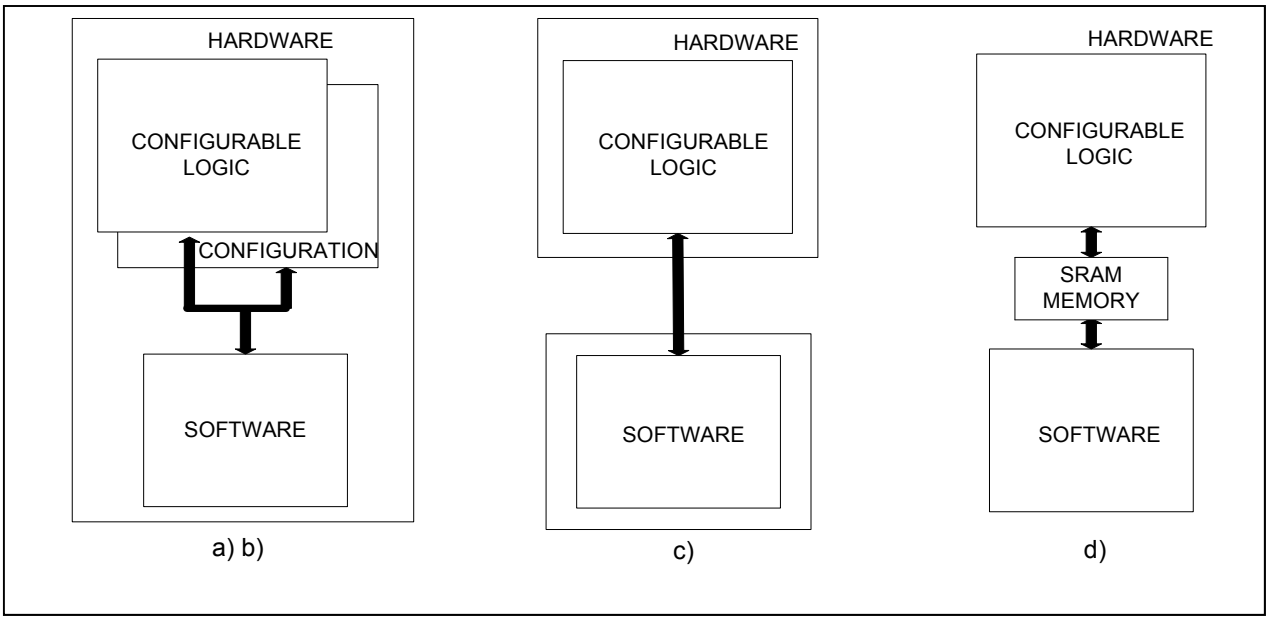

Figure 1. Interfacing models for a Codesign system

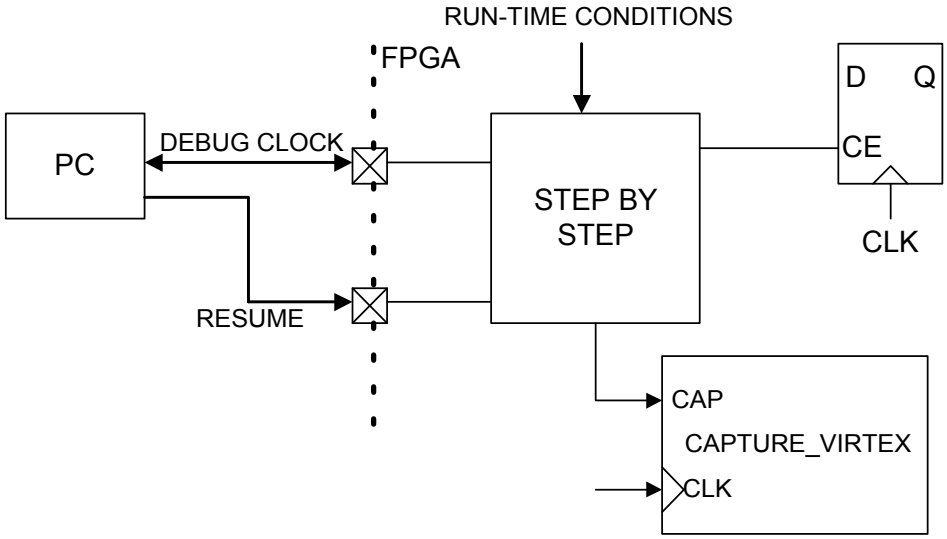

Figure 2. Debugger circuit block diagram

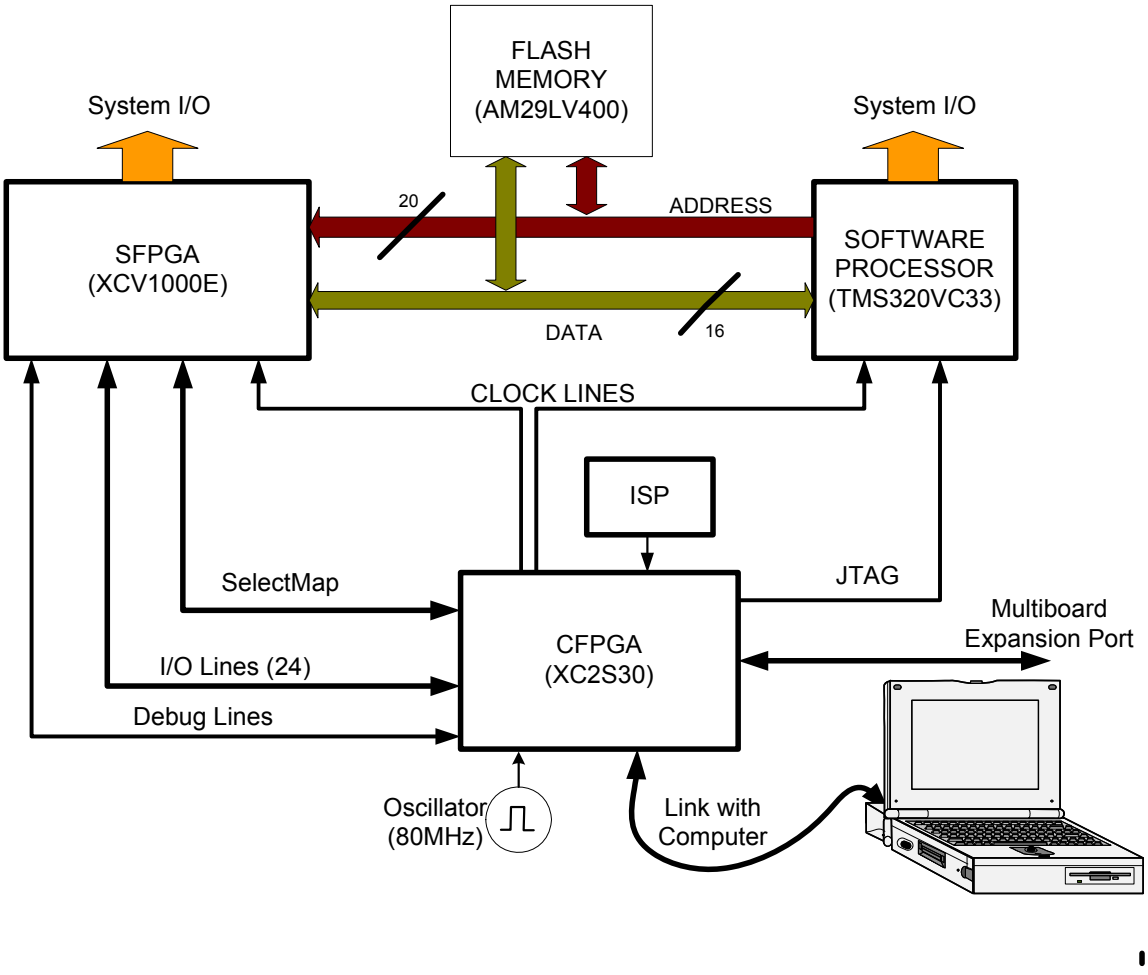

Figure 3. UNSHADES-2 system diagram

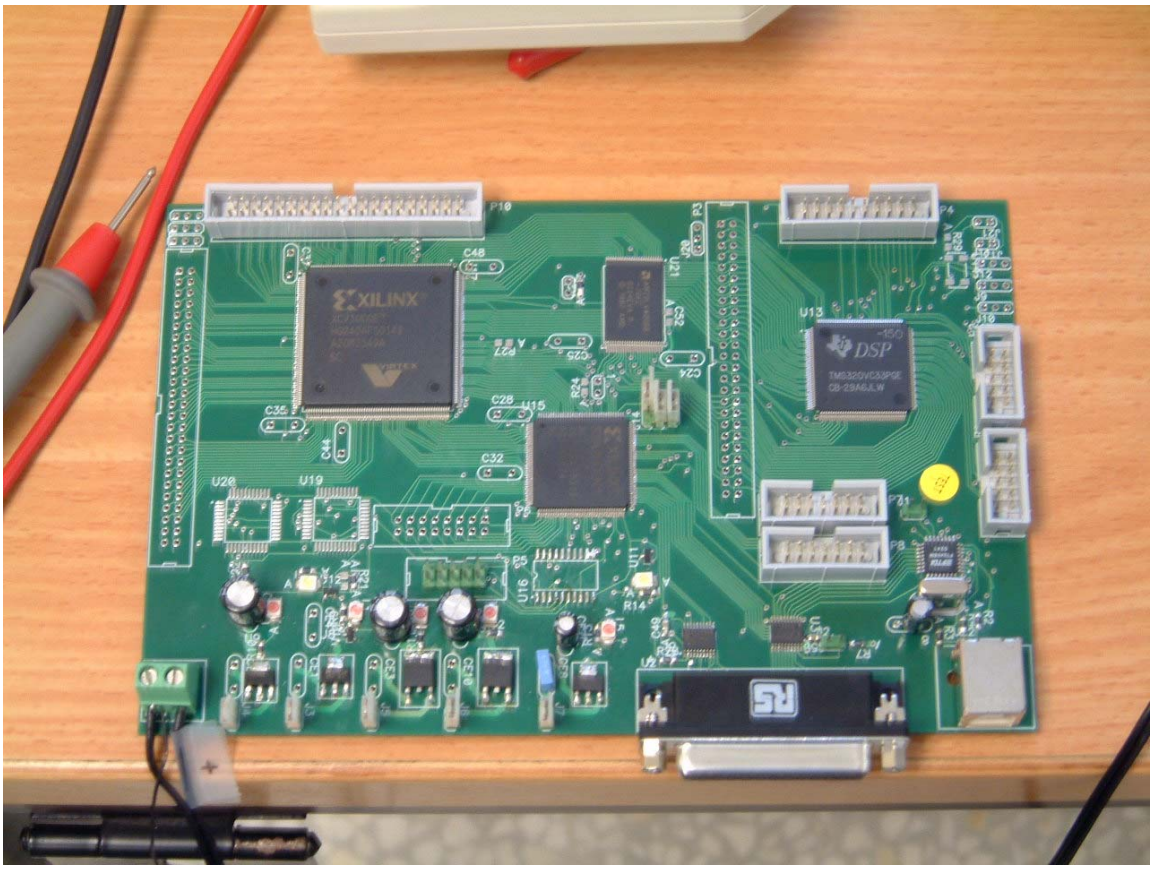

Figure 4 UNSHADES-2 platform board

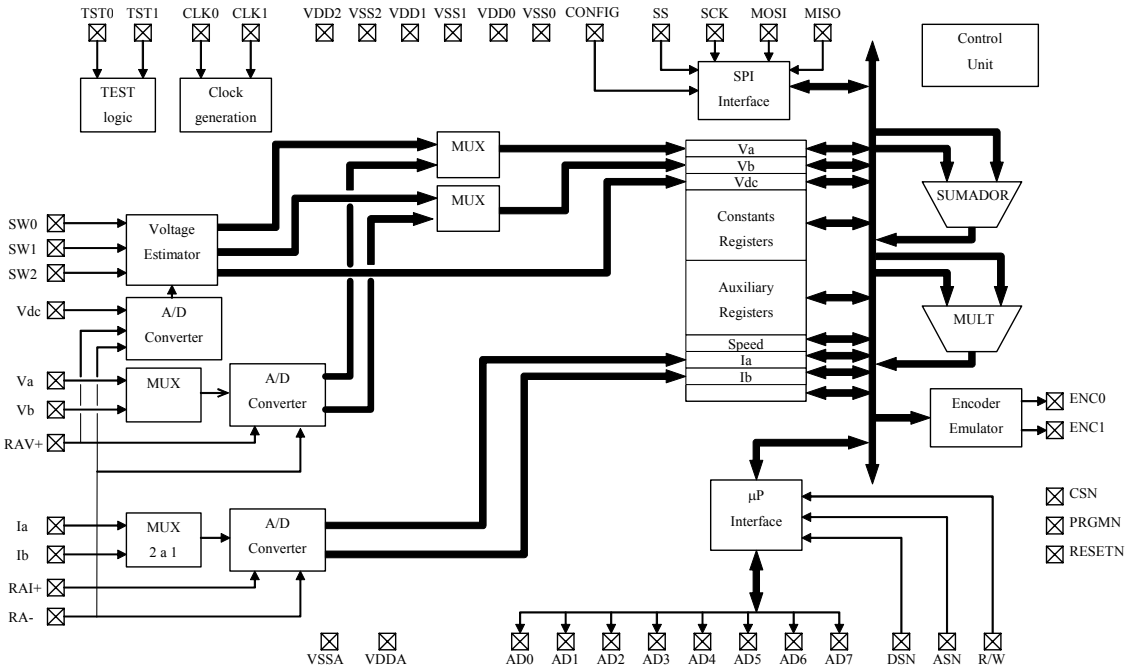

Figure 5. Block Diagram of SLESS ASIC

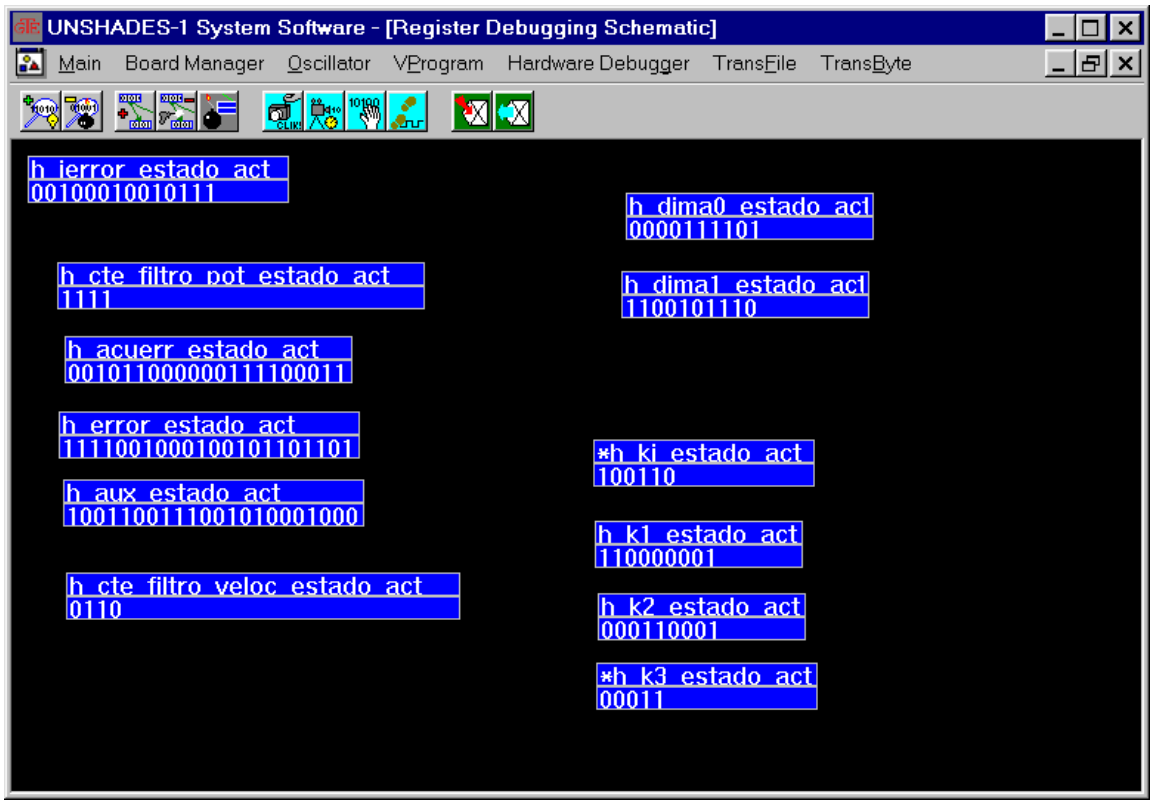

Figure 6. Screenshot of the UNSHADES-2 debugger interface showing part of the SLESS datapath