Addressing HW/SW Interface Quality through Standards

David Murray
Duolog Technologies
Mervue Technology Park
Galway, Ireland
david.murray@duolog.com

Sean Boyan
Duolog Technologies
Mervue Technology Park
Galway, Ireland
sean.boylan@duolog.com

***Abstract*— As software in an increasingly important aspect of system development, product schedules are mandating the earlier development of software concurrently with hardware. The Hardware/Software (HW/SW) interface is a critical development artifact that plays a key role in efficient system realization. This white paper gives an overview of the HW/SW interface and discusses the typical complexities encountered when designing the SW/HW interactions. The HW/SW interface flow is analyzed to show how insidious bugs are introduced into this domain. A compelling HW/SW interface solution is presented that combines best-practice design, formal specifications, the leveraging of different industry standards and register management solutions. This paper discusses each solution and how the emergence of standards such as IP-XACT (IEEE1685) and UVM help to eliminate many of these bugs and vastly improve the quality of the HW/SW interface. This white paper also discusses areas of improvement and possible standardization going forward.**

Keywords- ; HW/SW interface; IP-XACT; UVM; IP quality; Register Management.

I. Background

Software development has become a dominant factor in the realization of complex systems and the overall success of related products is increasingly dependent on software oriented features. For complex systems, software can consume more than 50% of the development cost. The integration of software with complex hardware platforms can take over 50% of the product development time [1] and with software firmly on the critical path, the development of software earlier and concurrently with the hardware is of crucial importance to time-to-market. It is therefore, extremely important to ensure efficient software development, hardware/software integration and concurrent hardware/software development flows. One key area that responds well to overall productivity and quality improvements is the low-level hardware/software (HW/SW) interface.

II. The HW/SW Interface

The main software perspective of the hardware can be defined as the view that a processor has of a system within its accessible address space. This address space is defined in a modular manner and is generally an ordered layout or memory map of the hardware. A hardware or IP block such as a UART module which is part of a peripheral sub-system would appear within the peripheral sub-system memory map as indicated in Figure 1 below:

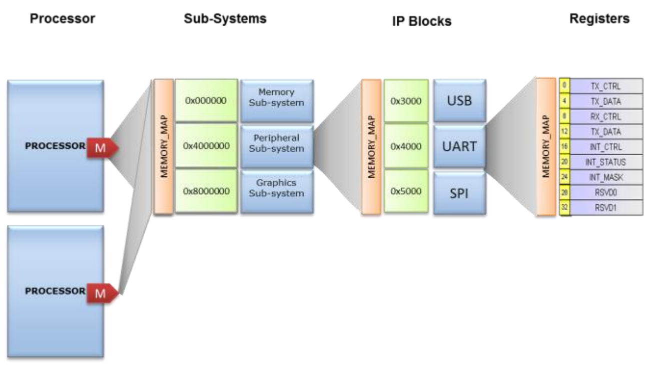


Figure 1 : Processor views of a system

The memory map can be considered hierarchical as it fragments into IP blocks. The hardware implementation of memory maps typically consists of bus interconnect fabrics, bridges and decoders. Within the IP blocks the lowest level HW/SW interface primitive is found in the form of SW programmable registers.

There are other vital aspects of the HW/SW interface, such as interrupts but these are not covered within the scope of this paper. Figure 2 shows the software interface of an IP block as an address-mapped bus (from processor) being mapped into registers that provide the configuration, control and status interface of the hardware logic.

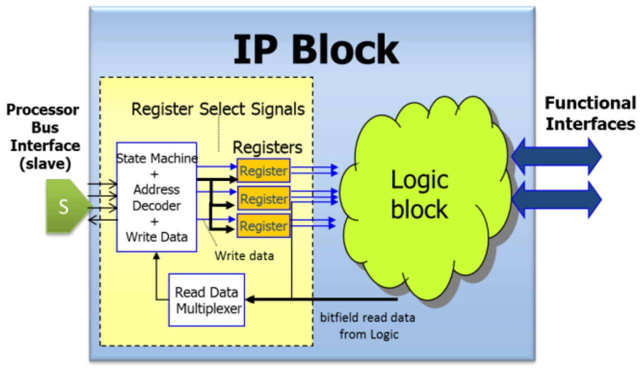


Figure 2 : SW interfacing with HW within an IP block

The software typically has a variety of different interfacing mechanisms to these registers e.g. read-only, read-write, write-only etc. These registers typically contain an aggregated set of different bitfields, each of which can have their own access characteristics. From a software perspective a typical register definition would contain the information shown in Figure 3.

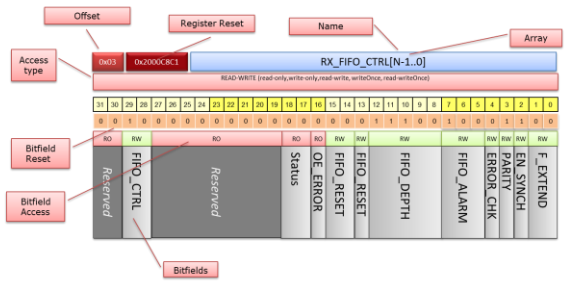


Figure 3 : Example of a SW-programmable register

This information typically includes access types, reset values, offsets, width, names, at register and bitfield levels. Some added complexities include additional behavior on bitfields/registers when they are accessed such as when a read access clears bits (read-to-clear). Overall the HW/SW interface structure can be quite complex as some systems can have 10s of 1000s of registers.

Given the current software development dilemma, discussed at the start of this white paper, it is important to have high levels of efficiency, quality and timeliness to reach product delivery goals. This mandates high levels of involvement from different teams all focused on a single domain (HW/SW interface) but who have diverse goals, expertise and perspectives. The HW/SW interface is the ultimate gathering place for different electronic system engineering disciplines. It is the meeting point of Design ⇔ Verification, HW ⇔SW, IP-level ⇔ Chip Level, Virtual ⇔ Real, RTL ⇔ TLM and Specification ⇔ Implementation. The necessity of concurrent HW/SW design across these domains leads to interesting dynamics and problems. The convergence of the different teams to this singular domain creates a key prerequisite to achieving system design goals – good communication.

III. Problems on the HW/SW Interface

Problems in the HW/SW interface originate from factors such as complexity, concurrency and team misalignment

Complexity: The HW/SW can have a high level of complexity. SW programmable registers can contain aggregated functions of bitfields with different access characteristics and sideband behavior. Complex IP can have 100s and sometimes 1000s of registers all of which need to be implemented across the different design domains. Multicore designs can add a further level of complexity as memory maps and registers are shared across the different processors.

Concurrency: Supporting a concurrent design flow moves design-flow methodology away from traditional waterfall-based processes to more incremental and iterative-based ones. This means that, for a large part of the design flow, the whole system can be in flux and unstable. IP registers and memory maps can change, IPs may be moved from one sub-system to another and sub-system/top-level memory maps may need realignment. Quick turn-around times will be needed to reflect these changes in all of the different design views. Keeping a coherent processor view of the system can be quite a challenge.

Team Misalignment: The number of teams involved adds to the complexity associated with addressing problems in the HW/SW interface. Typically hardware IP design, IP verification and firmware development are specification driven. With increasingly iterative design flows the specifications themselves can be considered unstable and for this reason it can be very difficult to keep teams aligned. If the quality and stability of the specification are compromised during implementation it leads to a HW/SW interface engineering gap as shown in Figure 4.

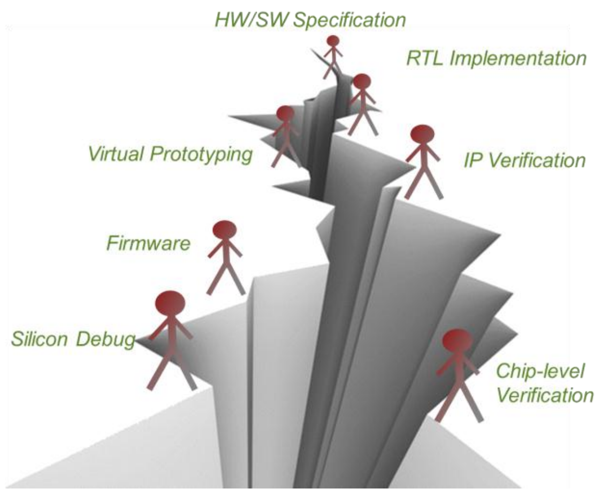


Figure 4 : The HW/SW interface engineering gap

This HW/SW engineering gap exists when a design flow produces different HW/SW interface implementations for the different teams. This engineering gap contributes to very long HW/SW integration cycles as issue resolution across multi-domains and multi-disciplines is very cumbersome. For example, if a firmware developer misinterprets a specification and implements a bitfield with some subtle difference from the original specification, it may be very difficult to find and isolate this during HW/SW integration or even to debug in a lab on the real device. Compelling solutions need to eliminate these possibilities.

While these are the main challenges, it is useful to show how these manifest themselves as real life issues and bugs in the design flow. The next section of this white paper will address this.

IV. HW/SW Bug Analysis

Bugs can be introduced into the design process at a very early stage, such as HW/SW interface definition. These bugs can be extremely subtle such as missing information in the specification. They may only materialize much later in the downstream process with dire consequences. This makes the bugs insidious in nature and this is where we introduce 'SID' the insidious HW/SW interface bug.

Figure 5 : SID- The 'insidious' HW/SW interface bug

'SID' and his prolific family can make his way into the design process in a number of ways.

A. *Specification Bugs.*

Specifications are a considerable vulnerability that allows insidious bugs to enter the process. They can either be blatant errors e.g. miscalculation of bitfield offsets or reset values or they may be more non-deterministic and subtle bugs such as incorrect descriptions of behavior, or missing information. Typical specification errors include:

- Incorrect and inconsistent bitfield and register reset values
- Overlapping register/bitfield offsets
- Incorrect and inconsistent bitfield and register access types
- Missing bitfield behavior
- Inconsistent naming conventions

B. *Interpretation bugs*

These bugs follow the last category very closely. A specification may have certain ambiguities that can be interpreted differently or information might be missing. This can result in different implementations. Some examples of ambiguities that cause these bugs could be;

- A reset value for an 8-bit wide bit-field is defined as 10. This could be interpreted as 0x10, 10 decimal or 10 binary
- A register access value of 'R'. This can be interpreted as Read-Only, or maybe it could be Read-Write
- Unspecified bitfield behavior - this could be open to implementation – e.g. what value should get read back. Is it deterministic?

C. *Transformation bugs*

These bugs occur when a specification is being transformed (or translated) into a different format e.g. from a specification into an RTL design or a verification test-bench. These bugs are particularly acute when the transformation is done manually and is exacerbated by unstable specifications requiring repeated transformations. Examples of these are:

- Standard typos in names, offsets and access values
- Incorrect behavior implemented e.g. missing write behavior to a register
- Copy-Paste-Forget errors

D. *Team Synchronization bugs*

Team synchronization is causing the most recent *outbreak* of HW/SW interface bugs. These bugs are typically caused when different implementations interact incorrectly and result in bugs being generated in the design flow. One of the frustrations here is that each implementation could be correctly adhering to a specification but just not a mutually common one. Examples of where these mismatches can occur are:

- IP testbench ⇔ IP design mismatches
- SW/Firmware ⇔ TLM model mismatches

In summary Figure 6 shows where the bugs can appear during HW/SW interface development.

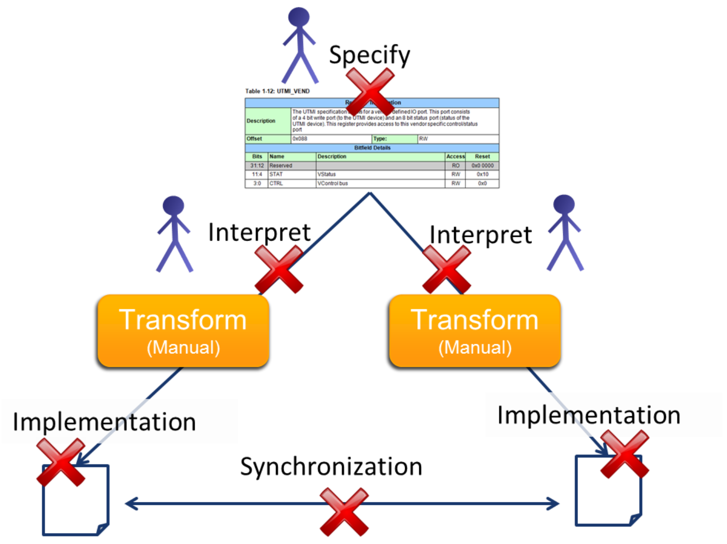


Table 1-12: UTMI_VEND

[illegible]				
Description	The UTMI specification [illegible]s for a ve[illegible] defined IO port. This port consists of a 4 bit write port (to the UTMI device) and an 8 bit status port (status of the UTMI device). This register provides access to this vendor specific control/status port			
Offset	0x088	Type:	RW	
Bitfield Details				
Bits	Name	Description	Access	Reset
31:12	Reserved		RO	0x0 0000
11:4	STAT	VStatus	RW	0x10
3:0	CTRL	VControl bus	RW	0x0

Figure 6 : Where are the HW/SW interface bugs?

V. Solutions on the HW/SW interface

The previous section summarized the main types of bugs in the HW/SW interface which included specification bugs, interpretation bugs, transformation bugs and synchronization bugs. The solutions to eliminating these bugs fall into the following categories:

- Better design practices
- Use of formal specifications
- Leveraging of industry standards
- Automated flows

It can be demonstrated that using these main strategies, and in particular by the leveraging of standards, the quality of the HW/SW interface can be *dramatically improved.*

A. Better Design Practice

Better design-practice aims to achieve a more consistent and standard design of the HW/SW interface. This is essentially a design-for-integration type of methodology. Some examples of good design-for-integration guidelines are detailed in a book by Gary Stringham [5] including:

- [ID-8.2.7] Design registers should return zeros for reads from unused bit positions
- [ID-8.2.11] Avoid write-only bits whenever possible
- [ID-8.4.6] Provide block-level ID and version registers for each block on the chip
- [ID-8.5.7] Registers should always return valid, accurate and documented values whether the block is idle or active

In general design-for-integration means the design focus encompasses the entire HW/SW domain scope. It may make sense for a hardware design engineer to implement a very 'alternative' access mechanism (e.g.write-1-twice-to-toggle) but it will not be easy to implement and verify in other domains such as verification, virtual modeling or firmware development.

B. Use of Formal Specifications

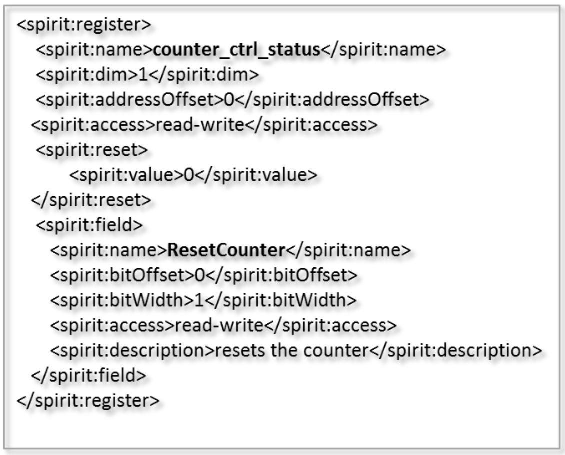
While the HW/SW interface has been traditionally described using a natural language specification (e.g. Word/Framemaker document) the problems associated with it have driven engineers to seek out more formal specification solutions. Formalizing the HW/SW interface definition requires adherence to well-defined and well-understood semantics. In natural language we can define register and bit-field accesses as read-write, read/write, rw, r-w, r/w or other deviations. In a formal description there would be only one way of writing this, for instance 'read-write'. There are many benefits of having formalized HW/SW interface specifications:

- A lot of ambiguity is removed
- Formal descriptions are easier to automate (also known as Machine-Readable)

The EDA industry is now providing an open and standard formal schema for HW/SW interface specification through IP-XACT.

C. Leveraging of industry standards

IP-XACT (IEEE-1685) is an open standard that defines a meta-data description of an IP block in the form of an XML schema [2]. This provides a common and language-neutral way to describe IP that is compatible with automated integration techniques and IP-XACT enabled tools. Many aspects of the HW/SW interface can be defined in IP-XACT and thus it can be used as a formal specification. For example, the following IP-XACT specifies a read-write register named 'counter_ctrl_status' at an address 0, with a single read-write bit-field 'ResetCounter'.



```
<spirit:register>
  <spirit:name>counter_ctrl_status</spirit:name>
  <spirit:dim>1</spirit:dim>
  <spirit:addressOffset>0</spirit:addressOffset>
 <spirit:access>read-write</spirit:access>
  <spirit:reset>
       <spirit:value>0</spirit:value>
 </spirit:reset>
  <spirit:field>
    <spirit:name>ResetCounter</spirit:name>
    <spirit:bitOffset>0</spirit:bitOffset>
    <spirit:bitWidth>1</spirit:bitWidth>
    <spirit:access>read-write</spirit:access>
    <spirit:description>resets the counter</spirit:description>
 </spirit:field>
</spirit:register>
```

Figure 7 : IP-XACT XML example

By having this formal specification as an industry-wide standard, the EDA industry provides highly automated solutions to ensure improved quality and efficiency. This will be covered in more detail in the Automation section of this white paper.

The **Universal Verification Methodology** (UVM) standard is a methodology to improve design and verification efficiency [3]. It enables verification data portability and interoperability between tools and verification IP (VIP). This methodology provides advanced verification capabilities and it encompasses specific applications including solutions centered on the area of HW/SW interface verification. For instance, UVM defines a class to describe registers and memory maps as well as providing access mechanisms to this class that offer verification engineers an intuitive API. In addition to this, UVM also provides a set of built-in test sequences that can be instantly used to check if the device under test's (DUT) registers conforms to this pre-defined register specification. Examples of these built-in test sequences are found in Figure 8:

Predefined Test Sequence	Description
uvm_reg_hw_reset_seq	Reads all the register in a block and check their value is the specified reset value.
uvm_reg_single_bit_bash_seq	Sequentially writes 1's and 0's in each bit of the register, checking it is appropriately set or cleared, based on the field access policy specified for the field containing the target bit.
uvm_reg_bit_bash_seq	Executes the uvm_reg_single_bit_bash_seq sequence for all registers in a block and sub-blocks.
uvm_reg_single_access_seq	For each address map in which the register is accessible, writes the register then confirms the value was written using the back-door. Subsequently writes a value via the backdoor and checks the corresponding value can be read through the address map.
uvm_reg_shared_access_seq	Requires the register be mapped in multiple address maps. For each address map in which the register is accessible, writes the register via one map then confirms the value was written by reading it from all other address maps.

Figure 8 : UVM built-in register test sequences

The provision of these built-in sequences by UVM means that verification engineers no longer need to write the usual register access and bit-bash tests. They can focus on defining register behavior and handover this typically monotonous work to the verification environment. This is a good example of how UVM provides improvements in verification efficiency. However, while verification efficiency is increased there are still some areas that can cause concern. For instance, how is the UVM register definition captured? Is it correct? The main quality gap is left open if the UVM register packages are created as part of a disconnected or manual process. This is where automation has a big impact.

D. Automation

While each of the previous solutions has their own benefit, automation is key to bringing these together to provide a more holistic and comprehensive solution to address problems in the HW/SW interface. If a formal specification is adopted, then the first application of automation can be focused on the specification itself. The formal specification can be quality checked to ensure there are no bugs or downstream issues. Automation can check the following:

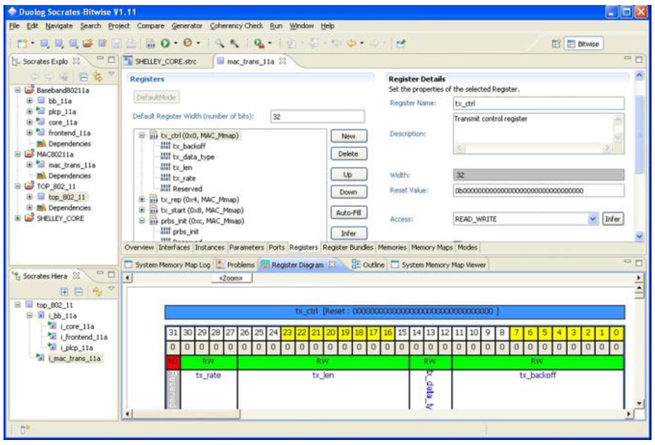
- The HW/SW interface specification adheres to the correct schema
- There are no overlapping bitfields, registers etc.
- There is consistency between registers and bitfield attributes
- There is no required information missing
- IPs fit within sub-system memory maps
- Sub-system memory maps fit within the full system memory map

This is one of the most crucial automation activities because it ensures that insidious bugs are not entering the process. Automation can also provide the transformation process from specification to implementation, eliminating transformation errors. For example, automation can provide a mechanism to generate the UVM register package from a formal HW/SW interface specification. This is currently possible with IP-XACT and UVM which are essentially two standard, but different register models. The fact that these are industry standards means that this automation is quite deterministic and is solved by the EDA industry and is known as is known as register management [6] [7] [8]. It is interesting to note that while IP-XACT provides a formalization of the HW/SW interface, automation leverages the real value from this formalization. It is possible, through automation to enhance IP-XACT compliancy and check for inconsistencies and missing information. Automation of the main implementation formats, including documentation, also eliminates the synchronization bugs presented earlier.

Register Management is a well-recognized solution within the EDA industry [6] [7] [8] and is probably better described as HW/SW interface management. This solution typically has the following features:

- Formal HW/SW interface specification
- GUI for capturing registers, bitfields, memory maps at IP, sub-system and chip level
- Full Coherency checks including all register attributes and full memory map validation
- Import of different formats e.g. from excel, XML
- IP-XACT import/export
- Generation (Transformation) of a wide range of formats, including documentation, RTL, Verification, SystemC and firmware code

The following diagram shows the GUI for a register management solution called Socrates Bitwise described in [6] [7] [8]



Default Register Width (number of bits): 32

Set the properties of the selected Register.

Register Name: tx_ctrl

Description: Transmit control register

Width: 32

Reset Value: 0b00000000000000000000000000000000

Access: READ_WRITE

Figure 9 : Bitwise GUI

A good register management solution can completely eliminate insidious bugs from the HW/SW interface flow. This bugs include specification bugs, transformation bugs and team synchronization bugs.

VI. Future

Standards provide a clear benefit for automation of the HW/SW interface. The EDA industry has endorsed the importance of the HW/SW interface with focused solutions such as IP-XACT as well as advanced verification methodologies such as UVM. Clearly, there is a need to refine the current standards and expand the standardization into firmware and virtual prototyping development. On the firmware side there is an emerging standard based on the ARM processor called CMSIS (ARM Cortex Microcontroller Software Interface Standard) [4]. This is the type of direction the EDA industry needs to move towards in order to fully align all the teams that utilize this critical design domain. The provision of a SystemC register package would boost transaction level modeling (TLM) automation. Finally, future quality enhancements can be provided by raising the level of abstraction used in describing the HW/SW interface even more through the formal specification of the HW/SW programming sequences.

VII. Conclusion

With software being a vital part of System Realization any quality issues on the HW/SW interface have a direct impact on cost and time-to-market. As the HW/SW interface is a facet that is shared across many different design teams it is also the breeding ground for many bugs such as specification bugs, interpretation bugs, transformation bugs and synchronization bugs. Many disparate methodologies and standards on their own can incrementally improve on quality and efficiency in HW/SW interface design and integration. However only fully automated flows can aggregate and multiply these benefits to fully eliminate HW/SW interface bugs and streamline HW/SW design.

VIII. References

[1] Concurrent Hardware/Software Development Platforms Speed System Integration and Bring-Up : Avinun : 2011

[2] http://www.accellera.org/activities/committees/ip-xact

[3] www.uvmworld.org

[4] http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

[5] Hardware/Firmware Interface Design. : Stringham, 2010

[6] Register Management of Complex SoCs : Murray, Clinton, Sugar, Olaszi 2008.

[7] http://www.duolog.com/products/bitwise/

[8] ESL Models and their Applications: 2009 Martin-Bailey.