
TITLE:

**Co-simulation Environment for an Advanced
Current Measurement Application**

AUTHOURS: Stefaan Kerckenaere
 Hans Manhaeve

REVISION: A

DATE: 28-09-2009



Table of contents

1	Abstract	2
2	Work plan	2
3	Results	3
4	Conclusions.....	4

1 Abstract

The objective is the creation of a co-simulation environment for testing an advanced current measurement instrument that makes use of an embedded of-the-shelf MCU. The co-simulation environment is setup using a VHDL IDE and the IDE software of the MCU vendor. The advanced current measurement system is successfully co-simulated with this flow and will be used in product maintenance and customization actions as it offers added value to the traditional MCU design flow in the form of increased product reliability and a systematic work flow for developing customer tailored firmware. The framework significantly decreases firmware design time and significantly improves test quality and product reliability. The simulation environment can also assist in creation and validation of higher level MCU models. The choice of VHDL to create this framework is not a limiting factor as higher level system language support (SystemC, ADA, ...) could also be added to this work flow.

2 Work plan

The instrument consists of an analog front-end, an ADC and an MCU. The MCU is an Atmel AVR RISC 8-bit MCU and comes with the Integrated Development Environment (IDE) AVR Studio. AVR Studio will be used for the MCU design entry.

It is not the intention to simulate the analog front-end behavior but only to co-simulate the ADC's digital back-end and the MCU. The objective is to use the ModelSim VHDL design environment to accomplish that task. VHDL is also selected as the target description language as this is the best suited language to model HW. No matter at what high level a system is described, eventually for synthesis, VHDL is what it is all converted to. VHDL allows to handle concurrent events and processed in a very easy and native way and does not require the multi-threading overhead imposed by standard SW programming languages. As such, in addition to co-simulate system HW, it is also very easy to co-simulate HW components external to the MCU, of which a model is available, that are part of a DFT or serve a debugging and verification strategy but that not necessary are part of the final application. Such external HW parts can be part of the automatic response verification functionality as well.

Figure 1 shows a first potential implementation of such a co-simulation platform. The MCU code is designed and compiled by the MCU vendor's Integrated Development Environment (IDE). The resulting disassembled code is fed to a VHDL model of the MCU core. A full system description is created and appropriate stimuli are written to test the system. While the simulation is running, independent processes check the system behavior and compare it to the expected system behavior.

The advantage of such a setup is that the system is tested in a systematic and repeatable way which in se contributes to more reliable system design. A traditional ISS does not offer this kind of features

and do not offer a waveform viewer tool like in a VHDL IDE. By moving to the VHDL IDE, the viewing option offers an alternative tool to the designer to debug his system.

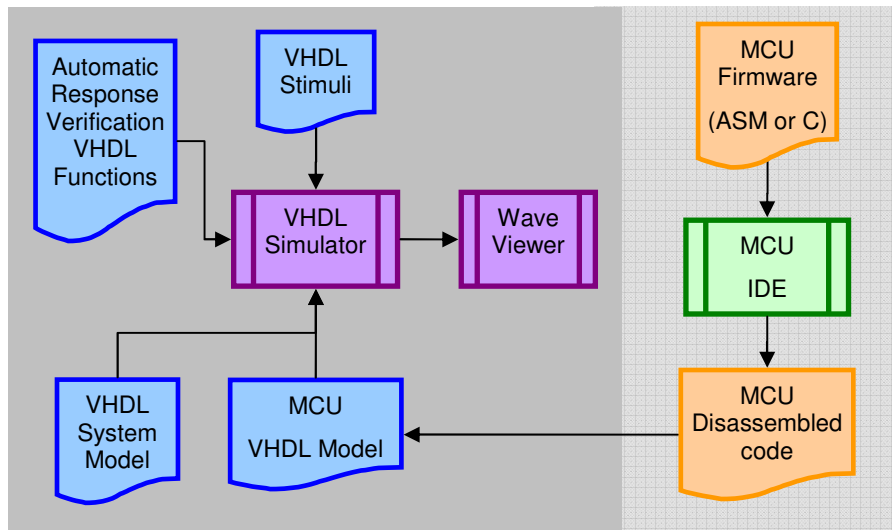


Figure 1. Design flow using a VHDL model

3 Results

The VHDL tool chain is well known and offers sufficient means to create all necessary input files for the VHDL simulator except for the MCU VHDL model. The MCU vendor does not provide VHDL models as by doing so they would reveal too much confidential information. Creating an in-house model introduces potential error sources in the design flow. A lot of effort must be spent in creating such model as well. To avoid this error source, an alternative path is outlined and shown in figure 2. The difference between the flows depicted in figures 1 and 2 is the MCU VHDL description versus the MCU model. In the flow depicted in figure 2, it is not a model that can interpret disassembled code generated by the MCU IDE, it is a description of the MCU response on the system test stimuli generated by the ISS. Through ISS log information of I/O and internal registry, the MCU response is logged and converted to a VHDL description and synchronized with the system's response in the VHDL simulator time domain. The stimuli created in VHDL are also converted to the MCU ISS stimuli format and fed to the ISS. The domain convertors are written in Python script files.

This approach avoids the need to write an MCU VHDL model and eliminates the potential error sources involved with this. Another advantage of this approach is that the MCU portfolio remains up to date in case a different MCU type would be required (e.g. obsolete parts over time). In case of a VHDL model, a maintenance program would need to be in place for potential bug fixes.

FW errors are identified in the VHDL IDE on cycle accurate basis. For further bug root causing, the MCU IDE is used to step through the code till the erroneous CPU cycle is reached triggering for further debugging efforts. After bug fixing, the ISS simulator is run again and the updated MCU behavior is fed to the VHDL simulator in a subsequent run for testing the system behavior according to the test benchmark.

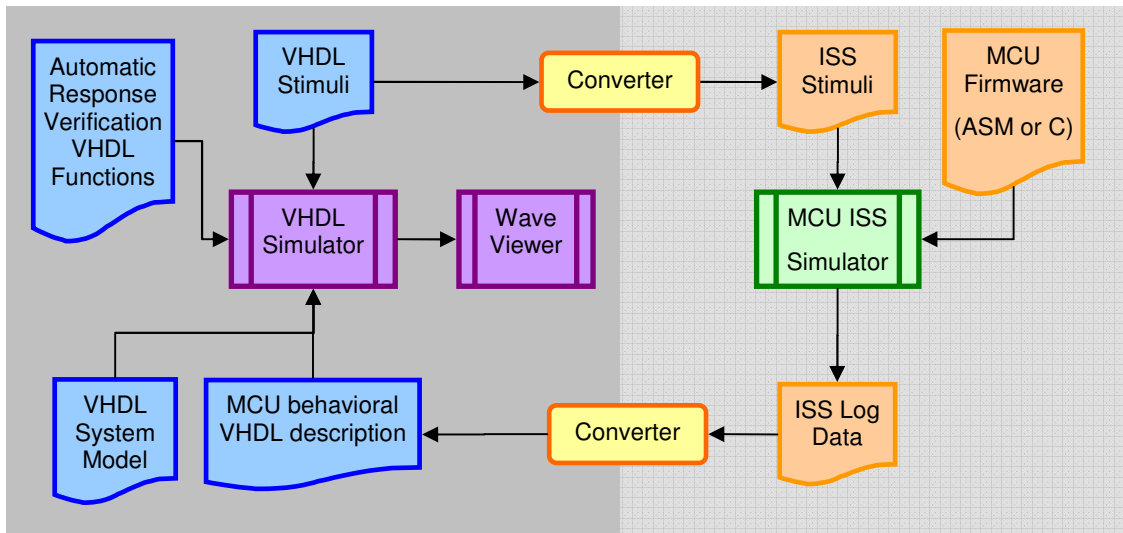


Figure 2. Design flow using the ISS

4 Conclusions

The advanced current measurement system was successfully co-simulated with this flow and will be used in product maintenance and customization actions as it provides a reliable, systematic flow for developing customer tailored measurement strategies. The framework significantly decreases firmware design time and significantly improves test quality and product reliability.

The drawback of this approach is that it does not allow real time co-simulation hereby limiting interactive simulation of other system parts. The AVR Studio simulator does not allow editing the stimuli file while the ISS is running, at least not at the time being, hereby limiting the interaction between both domains. It is unclear at this time if or when this feature will be supported in the near future. As such, the co-simulation model depicted in figure 1 is the way to go when real-time interaction is required to simulate the system. The creation of a MCU VHDL model is in that case unavoidable. However, the framework developed in this work can assist in checking the model by simulating the target code in both domains and using the MCU behavioral VHDL description in combination with a behavior difference tracking system to easily identify potential errors in the model. Without this framework this would be a tedious task.

Higher level system description language (SystemC, ADA, ...) support could also be added to this framework. Translators exist to generate VHDL code and MCU C or assembler code usable for the platform.