Opensource SoC Workflow in CloudV

Mohamed Shalan
Department of Computer Science and Engineering
The American University in Cairo *(AUC)*
Cairo, Egypt
mshalan@aucegypt.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI
Sherief_Reda@brown.edu

***Abstract*—Designing a System-on-a-Chip (SoC) for embedded systems involves many challenges such as IP integration and system verification. IP integration is increasingly seen as a key challenge in SoC development. Issues such as increased system complexity, IP reuse, and IP configurability render traditional flows obsolete. Moreover, SoC verification involves co-verifying the SoC hardware along with the software running on top of it. Hardware/software co-verification addresses one of the most critical steps in the embedded system design process, the integration of hardware and software. In this paper, we outline a workflow based on open-source components (SoC Editor, Beekeeper, and Pollen) that enables SoC workflow from IP integration to system verification as part of a cloud-based platform.**

Keywords—SoC, Opensource, co-verification, Embedded Systems, CloudV, Cloud

I. Introduction

The Electronic Design Automation (EDA) scene is known to be particularly encumbered with proprietary toolchains that not only alienate those who are unable to afford the extremely expensive tools typically priced for multi-billion-dollar design houses but are also very difficult to license and administer for academic, low budget and non-profit projects. The goal of CloudV [1][2] is to address these problems. CloudV is aiming at integrating open source EDA tools and make them available through seamless familiar web interfaces. CloudV's goal is to reduce design costs by relying on cloud infrastructure and on collaborative design. Using a web browser, designers can use CloudV anywhere using any connected computing device.

A main component of the project is a fully-featured SoC design workflow. This workflow incorporates an SoC designer (SoC Editor), an SoC compiler (Pollen), and a Co-simulation framework (Beekeeper). This workflow was introduced to address the challenges of SoC design, specifically SoC assembly and Verification (through hardware/software co-simulation). Also, the workflow provides basic library of hardware components (hard IPs) to build the SoC. Currently, the workflow supports SoC designs based on RISC-V CPU utilizing AMBA AXI4-Lite, AHB-Lite and APB buses [3]. In near future, we plan to support SoC design based on ARM Cortex-M01/3/4 will be supported. The following sections describes the work flow and each of its components.

II. SoC Workflow

As shown by Figure 1, the SoC design workflow in CloudV provides means for the HW IP designers, software developers, SoC architects and SoC verification engineers to collaborate on designing a new SoC. The HW designer may use CloudV IDE to enter a new HW IP RTL, and simulate it to verify its functionality. Also, using CloudV, the HW designer can get estimates of the area, clock speed and power. Once verified, the IP can be packaged and placed in a common repository to be used by the SoC Editor. The SoC architect can use the SoC editor to build the SoC out of components provided by designers collaborating with her/him or provided by CloudV itself. Through the SoC editor, the architect can configure the bus structure, configure the memory and attach and configure I/O components to the buses. Once done, an SoC translator called *Pollen* generates and configures device drivers for the SoC I/O devices and files needed to compile the software application (linker script, bootloader, …) as well as generating the HDL files for the designed SoC. Using CloudV IDE, the SW engineer can develop the software application which is compiled for the designed SoC using the GNU C/C++ toolchain. The SoC verification engineer, uses *Beekeeper* to co-simulate the SoC hardware with the software

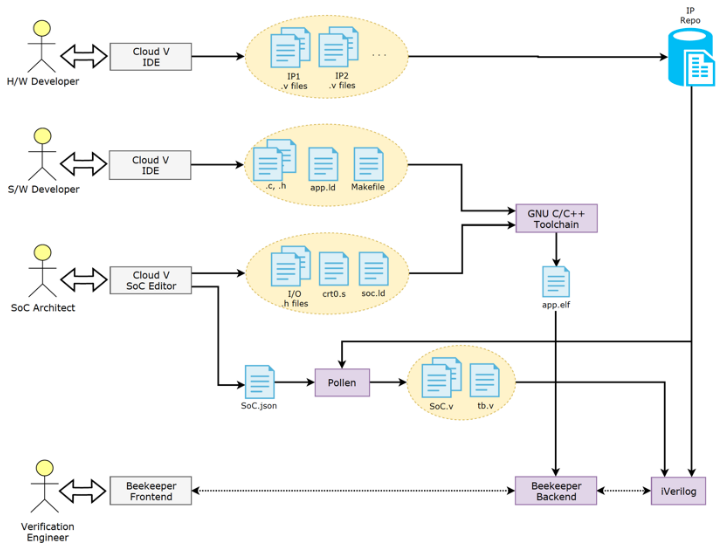
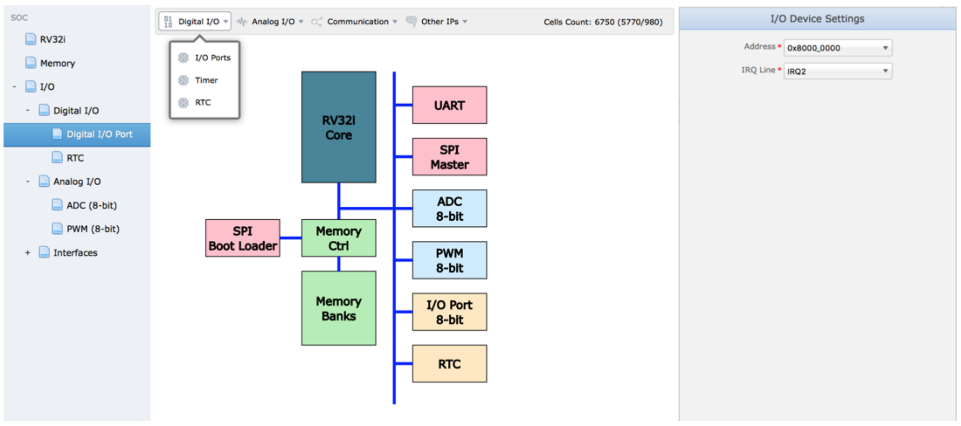


Figure 1. SoC Workflow in CloudV



Cells Count: 6750 (5770/980)

Address * 0x8000_0000

IRQ Line * IRQ2

Figure 2. The SoC Editor

application. In the following sections, we will discuss the SoC Editor, Pollen and Beekeeper in great details.

III. The SoC Editor

The SoC editor, shown on Figure 2, is a rule-based editor that enables designers to architect an SoC around RISC-V CPU using a library of open-source hardware IPs as well as IPs created by the users collaborating on the project. The architect should not worry about the details of how the IPs will be wired as it is handled behind the scene by the SoC editor. The SoC editor generates set of outputs that help hardware and software development:

- The top-level Verilog netlist needed for simulation and SoC Assembly
- Intermediate files to communicate with other CloudV modules (e.g. memory map, linker script, device driver's configurations).

```
[{
    "name": "Example",
    "bus": "AXI4LITE",
    "ips":
    [
        {
            "id": "RISCVCore",
            "owner": "Cloud-V",
            "verilog": "MultiBee",
            "bfm": true
        },
        {
            "id": "Main",
            "owner": "Cloud-V",
            "verilog": "InstantRAM",
            "bfm": true,
            "programHere": true,
            "start": 0,
            "length": 4096
        },
        {
            "id": "GPIO_0",
            "owner": "Cloud-V",
            "verilog": "GPIOSet",
            "start": 4096,
            "length": 8
        }
    ]
}]
```

Figure 3. JSON description of a simple SoC

The SoC Editor supports different configurations of the AMBA AHB-Lite, AXI4-Lite and APB buses. The architect can configure the SoC components if it is configurable. For example, the base address of each component, component IRQ line number, the type of each memory module and its base address, the needed ISA extensions for the CPU. The SoC Editor saves the SoC design in JavaScript Object Notation (JSON) data format. Figure 3 shows a simple SoC described in JSON. The SoC consists of a CPU core (RISCVCore), 4KB RAM and A GPIO peripheral. Pollen SoC Compiler

IV. Pollen SoC Compiler

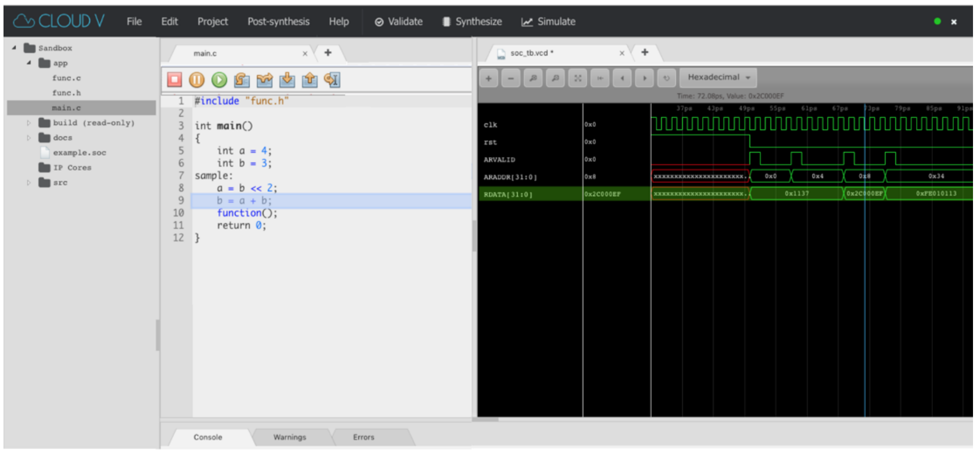
As an SoC compiler, Pollen takes an SoC as defined in a more abstract JSON format (similar to that of Figure 3) and converts it to HDL code. Out of the SoC Editor, SoCs are represented as a hierarchy of buses and components. A top-level bus, for example, AMBA AXI4-Lite, is considered the root bus, and every component attached to it would be a leaf unless the component happens to be a bridge that connects to another bus, which can also connect to multiple components under it. This tree-like structure, helps with processing, validation and generation of an SoC top level module.

Compliance is tested automatically, as Pollen incorporates a Verilog port parser that checks for the existence of all ports required by a specific bus on said module, including filtering out extra ports to expose as SoC ports. An SoC is then generated in HDL, with rigid internal naming conventions to assure lack of conflicts, and the SoC's top level module is exposed as a black box to the user as part of a test bench, complete with extra ports for IO elements and other such components.

V. Beekeeper Co-simulation Framework

CloudV has been extended to support C/C++ development using the GNU toolchain, so developers may create software projects for a given SoC design, edit C/C++ files, then simulate the compiled applications on the designed SoCs. Unfortunately, hardware/software co-simulation is very weak with standard tools, as there are no software debugging capabilities.

Development thus commenced on a framework that can be used to create CPU simulators that interact with a Verilog simulation using the Verilog Procedural Interface (VPI) [5]. Simple modifications to the SoC HDL description allow a specially built cycle-accurate CPU simulator to effectively replace the CPU during simulation and external hooks allow the user to control the simulation in a manner familiar to software debuggers, the popular lldb debugger having been used as the model for the interface. Upon integration with a



```c
#include "func.h"

int main()
{
    int a = 4;
    int b = 3;
sample:
    a = b << 2;
    b = a + b;
    function();
    return 0;
}
```

```
Time: 72.08ps, Value: 0x2C000EF
37ps 43ps 49ps 55ps 61ps 67ps 73ps 79ps 85ps 91ps
clk          0x0
rst          0x0
ARVALID      0x0
ARADDR[31:0] 0x8        xxxxxxxxxxxxxxxxxxxxx.  0x0  0x4  0x8  0x34
RDATA[31:0]  0x2C000EF  xxxxxxxxxxxxxxxxxxxxx.  0x1137  0x2C000EF  0xFE010113
```

Figure 4. Co-simulation Session using Beekeeper FE

development environment, a complete system based on an SoC, i.e. the SoC itself and its software, can be fully simulated.

As a proof of concept, a version of the framework has been implemented based on early design with a custom multi-cycle RISC-V CPU. The early implementation fully supports the co-simulation aspect and allows for limited debugging capabilities such as breakpoints, stepping over a line and stepping over an instruction. Future designs will see Beekeeper evolving to be akin to more sophisticated debugging tools such as gdb and lldb, i.e. hosting a debugging server for easy integration while still doing full hardware simulation under-the hood.

Simulation on CloudV is done using the Icarus Verilog open source simulator [4], which Beekeeper integrates with. There are plans to fork Icarus Verilog (iVerilog) to provide a socket-based interface as the GNU readline-based interface the tool adds a considerable overhead to CloudV, and it does not integrate meaningfully for error reporting.

CONCLUSION

In this paper, we outlined the CloudV SoC workflow. It relies on three open-source components: SoC Editor, Beekeeper, and Pollen. Integrated into CloudV, they enable SoC workflow from IP integration to system. The cloud-based system enables system integration with open-source design and tools and simplifies the costs for infrastructure setup. CloudV is available to try at http://cloudv.io.

ACKNOWLEDGMENT

This work supported by the JESOR grant from ASRT. We thank Ahmed Agiza and Mohamed Gaber for their great contributions to the project.

The source code of CloudV is accessible at https://github.com/Cloud-V.

REFERENCES

[1] M. Shalan and S. Reda, “CloudV: A Cloud-Based Educational Digital Design Environment," IEEE International Conference on Microelectronic Systems Education (MSE), pp. 39- 42, 2017.

[2] M. Shalan and S. Reda, Poster: CloudV, SIGDA University Booth, The Design Automation Conference (DAC), 2017.

[3] ARM, “AMBA AXI and ACE Protocol Specification”, 2011

[4] Icarus Verilog, http://iverilog.icarus.com/

[5] Stuart Sutherland, “The Verilog PLI Handbook: A User's Guide and Comprehensive Reference on the Verilog Programming Language Interface,” Springer, 2013