Rapid Virtual Prototype Model development using UML and IP-XACT

SystemC Code Generation

Deepak S Kurapati, Intel Mobile Communication India Pvt. Ltd, Bengaluru, India
(deepak.s.kurapati@intel.com)

Aravinda Thimmapuram , Intel Mobile Communication India Pvt. Ltd, Bengaluru, India
(aravinda.thimmapuram@intel.com)

Ever increasing demands for reduced Time to Market and increasing complexity of SoCs, have led to the development of rapid prototyping methodologies like Field Programmable Gate Arrays (FPGA) and Virtual Prototypes (VPs). VPs are functional models of the physical hardware that can simulate the behavior of entire SoC using normal workstation without the using additional hardware components. High simulation speed, ability to provide deep visibility, identical development environment, and cost effectiveness makes VPs preferred prototyping environment for today's SoCs development lifecycle.

Rapid development of the functional models of the Intellectual Properties (IPs) is critical to the development and early availability of the VPs. This paper presents an approach towards rapid development of the control path oriented functional models for which functionality relies on state machines, where multiple inputs/ events can generate state transition events. IP functionality can be specified using graphical representation and the interface and register details can be captured into specific format for code generation. This approach has the advantages of ease of understanding and ease of modifying the state machine and ensures uniform code generation across multiple IPs.

This paper describes a methodology for SystemC model generation (Loosely Timed) based on UML representation of the structural and functional behavior. The paper concludes with analysis report and benefits of successful deployment of this approach for one of the SystemC IP model development.

Keywords — Virtual prototyping; Loosely Timed; TTM; SystemC; UML; XML; IP-XACT; FSM; Controlled path IPs; Python

I. INTRODUCTION

Ever increasing demands for reduced Time to Market and increasing complexity of SoCs, have led to the evolution of rapid prototype methodologies such as Virtual Prototype (now onward referred as VP)s and FPGAs. VP is a functional model for SoC, simulating the hardware behavior as standalone software without using actual targeted SoC hardware.

VPs are developed at various abstraction levels addressing different system level issues (e.g. architecture exploration, driver verification, co-simulation etc). The SoC prototyping consists of development and integration of various cores, connectivity blocks, system-bus/es, memories, control and data path IPs. These building blocks integrated to behave similar to physical SoC functionality. The abstraction level of these IPs defines the VP use-case. High simulation speed, ability to provide deep visibility, identical development environment, and cost effectiveness makes SoC VP preferred prototyping environment in today's SoCs development lifecycle and to gain 'Time to Market' (as shown in figure 1).

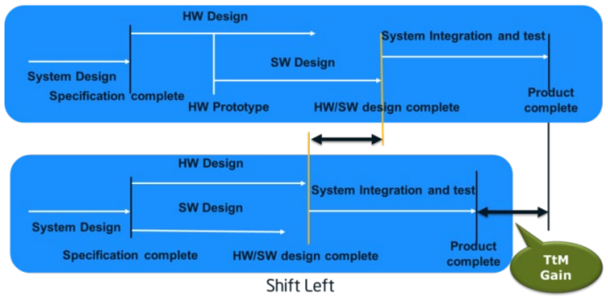

Figure 1. Time To Market achievement using VP

Control path IPs' (e.g. eMMC[1], I2C) functionality relies mainly on state machines. The inputs/ internal events of such an IP causes state transition and associated actions for each transition. This feature is used for rapid code generation. This paper highlights challenges in manual SystemC[2] IP development for virtual platform and proposes code generation for rapid development of control path IPs using UML and IP-XACT based code generation. The paper concludes with time gain in development effort and other benefits achieved during rapid SoC prototyping using proposed approach.

II. LIMITATATION WITH CURRENT METHODOLOGY

Current SystemC IP development depends upon manually coding and has following limitations:

- Requires longer development cycle and effort. Because of manual development, the IP development involves considerable time for designing, coding and testing.
- VPs are required to be available in early stages of product life-cycle hence the IP development should start during concept finalization phase. Since the design is not finalized, there may be chances to design change quite frequently (sometimes it may be multiple times within a week). The development flow should be dynamic enough to adopt these changes. In manual coding, this may not be possible.
- One SoC VP consists of multiple IPs (in few cases it may be 100+ IPs depending upon the SoC complexity). These IPs may be developed by various teams located at different geographical locations also. Every developer has his/ her own style of implementation (e.g. naming conventions, global vs local variables etc). There will be mammoth task involving huge effort for integration engineer to generate SoC.
- The IPs functionality may change till the product is developed; also there can be scope for enhancement or bug-fixes depending upon the end-product field experience. There will be considerable efforts required for doing maintenance and future enhancement.

In the market there are few tools which provides SystemC IP generations from existing code; but has their own limitations. E.g. SystemC code generated from RTL will be having less simulation speed because of low abstraction level implementation and doesn't suit for architecture exploration. SystemC code generated from C/C++ algorithm had to significantly modify to generate SystemC code and also requires post-generation modifications. So there is a need to have a methodology for rapid development and easy to maintenance.

III. PROPOSED SOLUTION

Control path IPs functionality can be precisely described in simple state- diagram using Unified Modeling Language (UML). These state diagrams are used in code generator which is an integrated environment consisting of multiple tools (third party and internally developed) & python scripts. Using these tools, IP developer can describe the IP state machines, the events causing the state transitions and the effect of these state transitions. The described state diagram can be exported to generate intermediate XML (eXtensible Markup Language) files using these tools. Interface details (e.g. signals, transactional ports and their types) are captured in another XML file. Register details are captured in the IP-XACT[3] XML; which is IEEE standard 1685-2009. These intermediate files are used by python scripts for generating SystemC code. It can be achieved as described below (also shown in Figure 2):

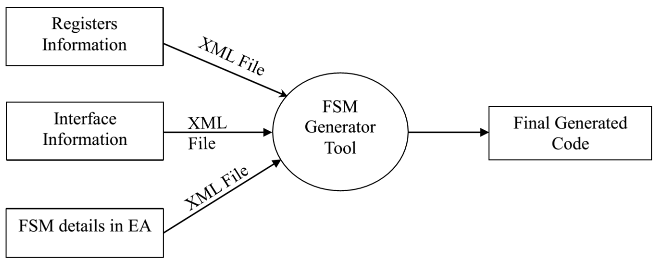

Figure 2. Code Generation Flow for SystemC IP generation

1. Register generation from IP-XACT[2] XML: The IP-XACT XML file consists of register width and bit-field details, software and hardware access permissions, offset addresses, reset values for each register.
   As a part of code generator, the internally developed XML parser scripts are used to generate SystemC source files. In the source file, each register is implemented as class with register reset value, accessors and manipulators for register bit-fields. The register classes are instantiated in SystemC class as private members. In class constructor the registers objects are registered for offset address mentioned in IP-XACT file. Also the constructor specifies the hardware and software read/write access permissions. This SystemC class will be used base class for IP to be implemented in next stages.
2. Depending upon the use-case. IP developer need to identify the ports and type of the each port. If possible, the developer can define TLM sockets (combining address, data, read/write, enable, byte enable ports present in hardware), as this improves the simulation performance. This information should be exported in XML file

format to be used by code generator. The development team can define naming guidelines to differentiate the type and direction of the port/ socket to ease integration process.

As a part of code generator, the internal developed GUI tool is used to describe the port details. The tool supports SystemC ports (in, out), TLM sockets (initiator and target). If there are multiple ports of the same type the array of ports can be defined (e.g. more than one interrupt lines). The tool is also capable exporting this information in intermediate XML files which are used in the later state of code generator. This information should be exported into XML format, which will be used during code generation. This tool generates two xml files (one for interface details, one for component description which instantiates the IP interface object).

3. Describing functional behavior of IP using UML. The UML diagram should capture the various stable states of the IP, the cause of the state transition and the effect of the state transition (e.g. modification of any internal data members/ register/ toggling the ports etc). Also, the UML diagram should capture any functionality associated the state (i.e. any functionality while entering into the state, exiting from the state). The tool should have capability to export this information in XML file format which can be used by code generator for SystemC IP generation. Depending upon the use-case and the level of required abstraction, developer can implement intermediate states, combine the states etc. To make the design easier, the developer can map the state diagram available in IP specification; which is general case.

   As a part of code generator, a third party tool has been used who supports above mentioned capabilities. The tool is having the capability to integrate internally developed configuration library (e.g. stereotypes for declaring timed event notification, interface and register class name, constructor and destructor functionality, class member variables and member functions) and used for code generation. The functionality supported in the library is state transitions on input ports, delayed transitions, actions/ effects on state transition (e.g. output port value change, member variable updates etc.), variables and function declarations etc.

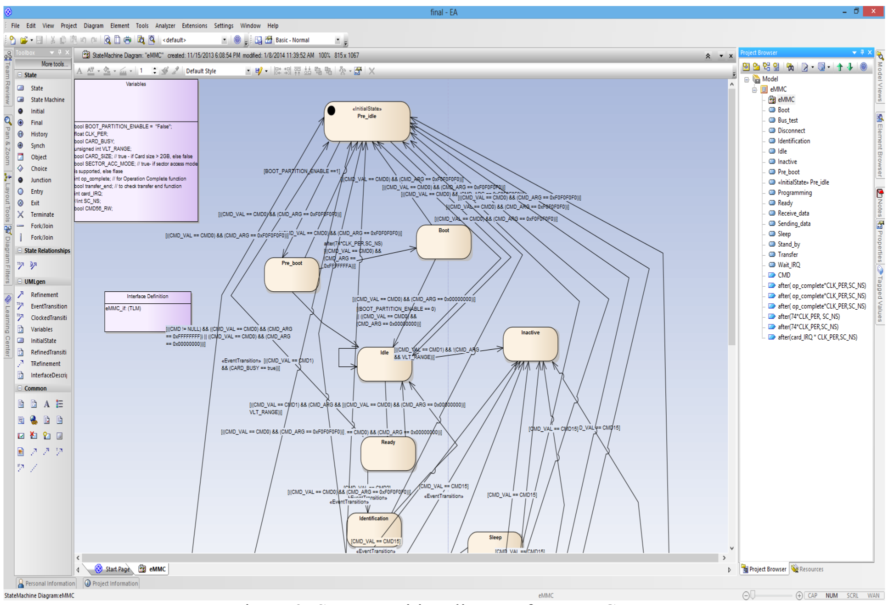

Figure 3. State transition diagram for eMMC

4. Execute code generator (consists of python scripts) generates SystemC code using intermediate XML files (generated in step (2) and (3)) and stub files. The scripts parse the configuration file to check code generation for:
   - Generic blocks (e.g. FIFOs, interrupt control logic, clock control logic)
   - Bus interfaces (TLM 2.0 sockets or internally enhanced transaction sockets).
   - Timing abstraction (e.g. LT, AT etc).

   The parser generates a header file and a source file:
   - Header file generation:
     - From register class name (defined in step [1]), the IP class is inherited. Also the corresponding header file is included.
     - The standard file header and include files (e.g. systemc.h etc) are dumped into the header file. This will be standard template used by code generator.
     - From component XML file (generated in step [2]):
       - The file and IP class name is obtained. The appropriate namespace name is derived.

- From port details corresponding SystemC ports are derived and dumped into class. From socket details and configuration parameter, either TLM 2.0 sockets or user defined transaction ports names are derived and dumped into class. Also corresponding events are identified and dumped into class declaration.
- For reset and clock ports callback function signatures are dumped into class declaration. For other input ports dummy callback function signatures are declared. In a manual coded file, the associated functionality should be written by IP developer.
- Define generic modules to instantiated (identified through configuration file parsing).

    - From XML file (generated in step [3]):
        - State names are parsed and enum members are derived.
        - Declare event descriptor (event_desc) variable to keep the track of event execution for the current simulation time. The flag will be used to avoid repeated calls within same state during current simulation time.
        - Declare state machine signature which is sensitive for any input port change event.
        - Declare function declaration for doEnter and doExit. These function consists of functionality to be executed when state machine enter/exits into/from any state. This functionality can be updating class members, register manipulation, output port toggling, timed transition to new state etc.
        - From member function stereotype, member function signatures are dumped into class declaration. These are the functions which user can coded in C++ file manually. This is provided for the developer to extend the functionality not covered through UML diagram.
- Source file generation:
    - From component XML file (generated in step [2]):
        - Dump header file inclusion.
        - In class constructor, declare SystemC methods (callback functions) and static sensitivity of respective input ports.
    - From XML file (generated in step [3]):
        - The constructor and destructor functionality is dumped from constructor and destructor stereotype.
        - Generic functions for functionality execution (such as while state machine entering into the state, exiting from state) will be dumped in doEnter and doExit functions.
        - Dump IP state machine functionality implementation (as explained in 'FSM Functionality section).
- FSM Functionality:
    - Whenever any input is changed corresponding callback function is called.
    - This will call state machine function with event_desc after checking if the event descriptor is valid to call state machine.
    - In the state machine:
        - If reset is active, no functionality is executed and returned.
        - From the current state and event, next valid state is evaluated.
        - If there is any valid state change, doExit function is called for current state exiting functionality exiting. Then the state machine enters to new valid state and executes doEnter functionality for new state.
        - If new state is an intermediate state (i.e. moving to next state after specified time as mentioned in XML file), notify the event to execute state machine functionality.

5. The maintenance/ future enhancement of the functionality can be done modifying in UML/ interface/ register description and doesn't involve any manual code modifications.

The generated IP model conforms to standard SystemC and can be used in any VP use-cases depending upon the abstraction details provided in UML diagram in step [3].

SUMMARY WITH RESULTS

Code generator has been piloted for couple of control path IP (eMMC and I2C) state machine implementation and found maximum of 80%* reduction in development effort compared to manual implementation.
The other benefits achieved using code generator are:

- Ease of maintenance and enhancements as this can be achieved by modifying graphical representation of the IP functionality.
- Uniformity across all control IP development

- Doxygen[4] compliant comments for documentation purpose.

* - This is only IP development effort and doesn’t include study and IP verification effort.

FUTURE ENHANCMENTS

- In the proposed solution, the SystemC code is generated by in step [4], from UML description captured in XML file. This is independent of any programming language. By changing the code generators, HDL/ C/ C++ code can be generated for various puspose (HDL for RTL generation, C/ C++ for any fast functional model etc). Using the same inputs same IP is generated in different languages, this eases maintaining the code.
- Test bench framework with dummy functions can be generated from XML specifying interface details.

REFERENCES

[1] eMMC is JEDEC standard (http://www.jedec.org/)

[2] IEEE 1666-2011: SystemC LRM

[3] IEEE 1685-2009: IP-XACT

[4] http://www.stack.nl/~dimitri/doxygen/

[5] Single-Source Hardware Modeling of Different Abstraction Levels with State Charts by Rainer Findenig.