2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

From Device Trees to Virtual Prototypes

Sakshi Arora, Verification Group, Synopsys India Pvt. Ltd, Noida, India
(*Sakshi.Arora@synopsys.com*)

Vikrant Kamboj, Verification Group, Synopsys India Pvt. Ltd, Noida, India
(Vikrant.Kamboj@synopsys.com)

Preeti Sharma, Verification Group, Synopsys India Pvt. Ltd, Noida, India
(*Preeti.Sharma@synopsys.com*)

***Abstract*—Embedded platforms and associated products are getting smarter day-by-day, thanks to their rapidly increasing complexity. Since software content of these systems is supporting complex features, the overall product development becomes highly complicated and time-consuming. The answer to keeping pace with this innovation is developing Virtual Prototypes (VP). With increasing SoC complexities in embedded-systems, the corresponding pressure to accelerate VP model/platform creation and reduce overall development-time is also increasing. In this context, generation of structural VPs for various IPs and their assembly, plays a key-role in improving overall iteration turnaround-time of generating VP.**

Keywords—device trees; memory map; human error; complex SoC

I. INTRODUCTION

Embedded platforms and associated products are getting smarter day-by-day, thanks to their rapidly increasing complexity. Since software content of these systems is supporting complex features, the overall product development becomes highly complicated and time-consuming. The answer to keeping pace with this innovation is developing Virtual Prototypes (VP). With increasing SoC complexities in embedded-systems, the corresponding pressure to accelerate VP model/platform creation and reduce overall development-time is also increasing. In this context, generation of structural VPs for various IPs and their assembly, plays a key-role in improving overall iteration turnaround-time of generating VP.

Current process of developing a VP requires SoC specifications availability. Initially, complete specification is generally not available to VP developers. Even if available, manual reading-parsing of information takes significant time and effort, increasing human-error probability. Furthermore, if specification is revised by SoC provider, updating VP involves repetitive manual iterations that considerably adds to overall platform development-time.

The proposed solution addresses these challenges and provides a systematic and innovative way to create structural VPs, using information available in operating system (OS) software (Device Tree, abbreviated as DT from here onwards), along with taking limited inputs from vendor provided IP/platform specification document(s). Figure 1 provides an overview of proposed flow, and how it wins over the traditional manual flow which crunches on huge manual input data. The flow enables taking minimal user inputs, facilitated by automation-framework, hence significantly reducing efforts in creating complex VP systems with 100's to 1000's of models and connections.

In this paper, we will present details of how complex platforms have been created with our structured and systematic proposed solution, presenting significant gains achieved in overall turnaround-time.

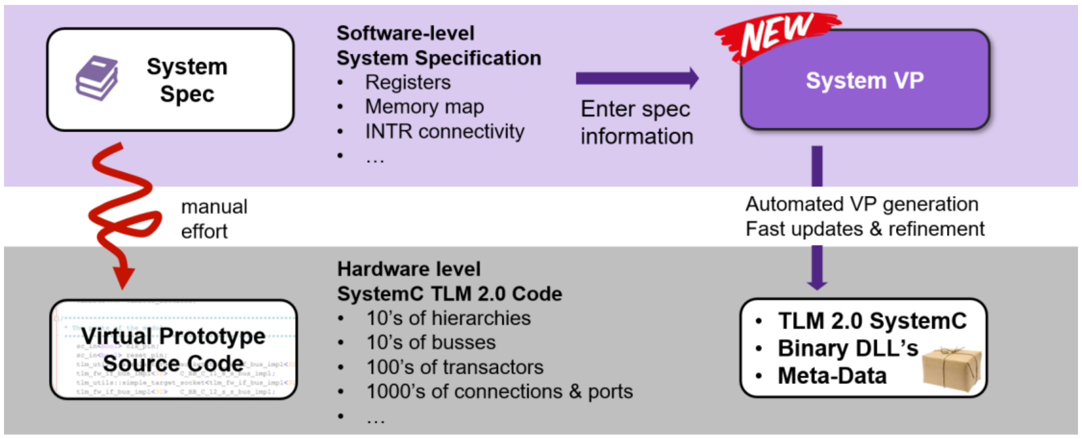


Figure 1. Impact of the proposed flow

II. RELATED WORK

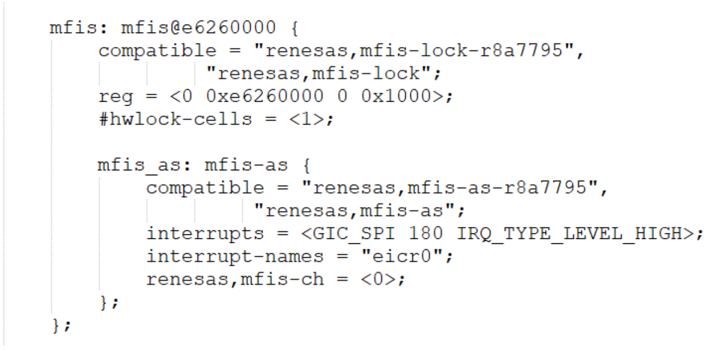
Current primary information-source for describing hardware platform from an embedded software developer viewpoint is Technical Reference Manual (TRM) [1]. Typically, developing a VP platform is manual-process, involving huge efforts and time bandwidth. There have been multiple attempts to automate creation of VPs using UML/MARTE profiles [2], SysML [3] and IP-XACT [4]. To use these, the hardware information needs to be available and encoded in respective formats: UML, SysML diagrams, IP-XACT etc. Making these formats available and decoding is again time-consuming and human error-prone. The proposed flow presents an alternative, independent and efficient way to derive most of platform assembly information from DT in absence of a detailed TRM.

III. PROPOSED FLOW

The proposed flow consists of two levels of inputs and stages as described below.

A. Level 1

First, the flow takes Device Tree files as input, reads all nodes in tree and generates a basic platform-skeleton with all required components. Device Tree is an essential component of OS kernel. It refers to a data-structure describing hardware components of SoC. These components include CPUs, memories, buses, peripherals and are managed by OS kernel.



```
mfis: mfis@e6260000 {
    compatible = "renesas,mfis-lock-r8a7795",
                 "renesas,mfis-lock";
    reg = <0 0xe6260000 0 0x1000>;
    #hwlock-cells = <1>;

    mfis_as: mfis-as {
        compatible = "renesas,mfis-as-r8a7795",
                     "renesas,mfis-as";
        interrupts = <GIC_SPI 180 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "eicr0";
        renesas,mfis-ch = <0>;
    };
};
```

Figure 2 Device tree nodes containing "reg" property

Platform's memory-map/connectivity information is further extracted from Device Tree. In Device Tree, every memory-mapped peripheral contains an entry of "reg" to specify address and size of memory-mapped registers [5] that are used for peripheral itself (Figure 2). Once memory-map information is extracted, a basic VP framework is created with various peripherals as placeholders at respective memory-locations.

B. Level 2

Using 2nd level of automation-framework execution, these IP placeholders are replaced with structural models. Here, IP specific data provided by user is parsed to add correct register-sets and interfaces to peripherals, creating a shell model known as structural Virtual Prototype. This data is categorized in three formats each for the registers (Table 1), bitfields for each register (Table 2) and interfaces (Table 3). This data is extracted from SoC specification(s) in CSV formats, using third-party tools. This structural VP can now be further developed with specific behaviors.

Register Name	R/W	Address	Initial Value	Size
RegisterA	R/W	H'E615 0004	H'0032 0235	32
RegisterB	R/W	H'E615 00E0	H'0000 0000	32
RegisterC	R	H'E615 00D0	H'0000 1F1F	32
RegisterD	W	H'E615 0078	H'0000 0209	32

Table 1 Format of the registers CSV file

Register Name	Bit	Bit Name	Initial Value	R/W
RegisterA	31 to 16	BitA1	All 0	R
	15 to 1	BitA2	All 0	R
	0	BitA3	1	R/W
RegisterB	31 to 0	BitB	All 0	R/W
RegisterC	31	C	0	R/W
	30	—	0	R
	29	—	0	R
	28 to 24	—	All 0	R
	23 to 20	C1	11	R/W
	19 to 0	C2	10	R/W

Table 2 Format of the bitfields CSV file

Interface Name	Quantity	Protocol	Role
p_reg_in	1	Memory	Input
P_clk_in	1	Clock	Input
P_rst_in	1	Reset	Input
P_mem_in	2	Memory	Input
P_mem_out	2	Memory	Output
P_irq_out	1	Interrupt	Output

Table 3 Format of the interfaces CSV file

The two-levels of proposed automation flow are depicted in Figure 3.

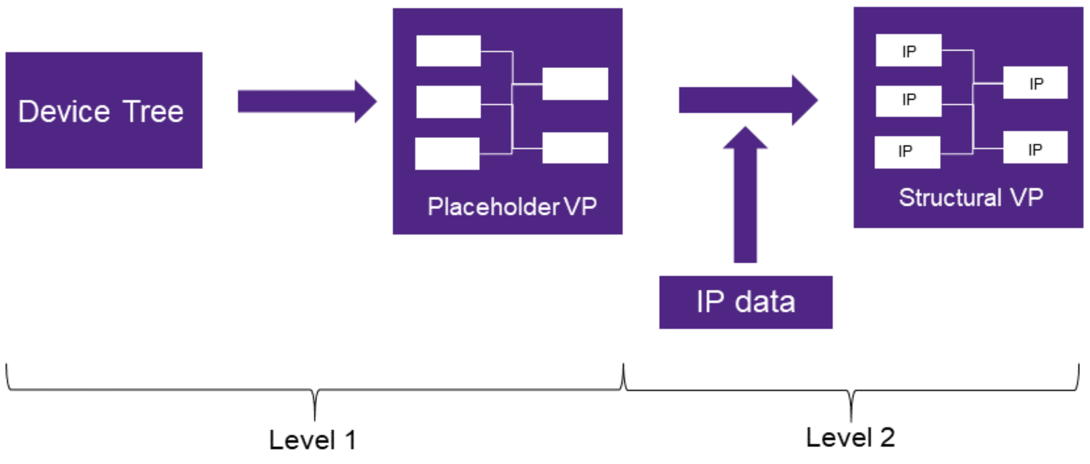


Figure 3 Proposed Flow for Creating Structural Virtual Prototypes using Device Trees

IV. Application

Main requirement of VPs is to execute the same software, that runs on actual hardware [1]. Therefore, creating VPs using Device trees eliminates various errors such as inconsistent information with respect to specification/software etc., during software bring-up.

The proposed flow caters to following use-cases:

- When an OS has already been ported for an SoC before availability of its specification to VP developers, DTs is available in open-source community, can be used to start developing VPs.
- Even if specification is available, the traditional manual approach of parsing it for each peripheral, and manually assembling the VP, is highly error-prone and time-consuming. Using DTs for memory-map/connectivity generation, significantly speeds up VP models/platform creation by systematically automating the process with minimal impact of human-errors and taking care of additional spec updates by vendor.
- This framework is very effective to quick-start next-generation (let say N+1th) platform development, whose memory-map specification is not yet available. Nth generation DTs can be used as foundation for platform development to reach a logically meaningful VP model.

The proposed flow reduces several steps exercised in the current process of creating VP for complex SoC (Figure 4). This structured and systematic flow can be effectively used to develop VPs for complex SoCs in automotive markets.

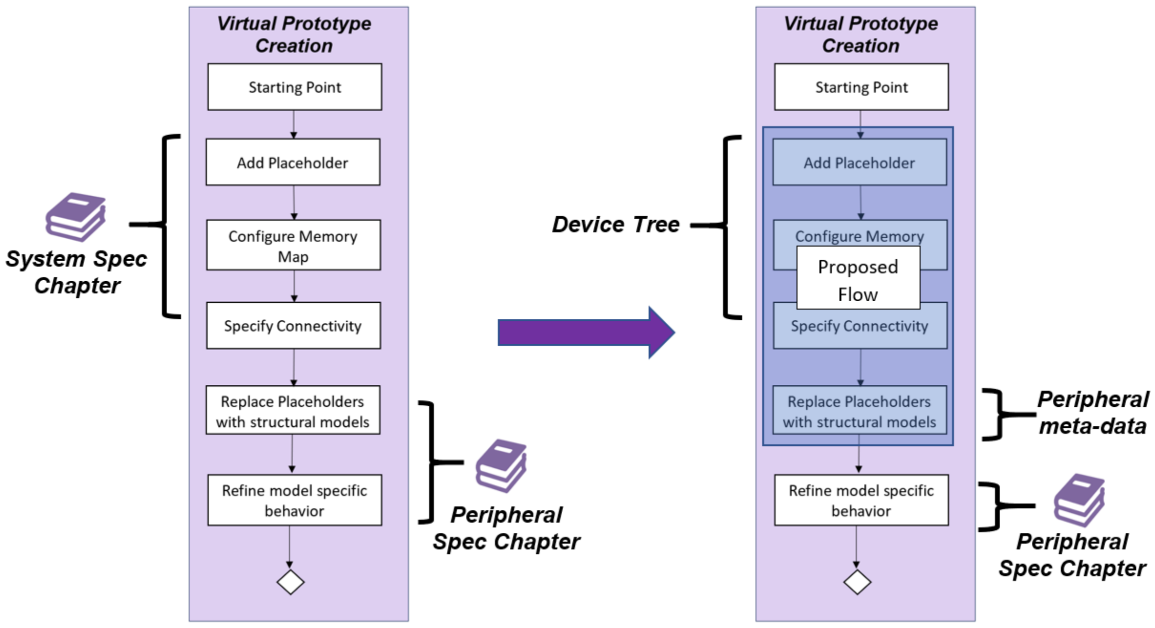


Figure 4 Existing manual flow (left) compared against proposed flow (right)

V. Results

We used the proposed solution to create VP platforms for complex SoCs for which initial-level specification didn't provide complete platform details, especially memory-map/connectivity. Device Tree for this platform was available because Linux had already been ported for the already available SoC hardware boards. Using Linux open-source code for DT in our systematic proposed flow, we unblocked ourselves (from waiting on specification 2nd draft), by extracting required memory-map information. This enabled us to quickly reach the stage of IP refinements and functionality addition. The total effort of VP creation got significantly reduced by 4X (months to days), as quantified in Table 4.

VP Platform	Number of IPs	Avg. no. of Registers per IP	Avg. no. of Interfaces per IP	Manual Effort (days)	Effort with Proposed Flow (days)	Improvement (x)
1	133	20	20	20	5	~4x
2	215	36	33	46	7	~7x
3	390	150	100	95	10	~9x
4	500	200	150	120	12	~10x
5	700	200	100	150	13	~11x
6	1000	250	150	270	14	~20x

Table 4 Effort reduction by using proposed flow

VI. Future Work

The proposed solution can be enhanced in the following ways:

- Extending the flow by extracting interrupt, clock and reset-tree details from the device tree.
- Enablement of automated software bring-up for OS boot (Linux) on VP platform, by further refining the meta-data extracted from Device trees, using standard OS-distribution source-repositories.

VII. Summary

The proposed flow very efficiently enables creation, development and integration of complex VP systems, by using DTs, and highly reduces dependency on vendor-specs and other mandatory information missing from specs. This flow tremendously reduces the probability of human-errors, and overall efforts and inefficiencies involved in developing complex SoC platforms, involving 100s of IPs, with 1000s of connections.

References

[1] Synopsys, Inc. ,"Using VDKs for Automotive Systems Development"

[2] F. Herrera, H. Posadas, E. Villar, and D. Calvo, "Enhanced IP-XACT Platform Descriptions for Automatic Generation from UML/MARTE of Fast Performance Models for DSE," in 2012 15th Euromicro Conference on Digital System Design, Cesme, Izmir, Turkey, 2012, pp. 692–699.

[3] D. Perillo and M. D. Natale, "Using MDA to Automate the Integration of Virtual Platforms for System-Level Simulation," in 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, 2017, pp. 268–277.

[4] G. Godet-Bar and J.-M. Fernandez, "A SystemC Extension for Enabling Tighter Integration of IP-XACT Platforms with Virtual Prototypes," p. 6.

[5] Booting the Linux/ppc kernel without Open Firmware

[6] SPIRIT Consortium, IP-XACT schema v1.4. http://www.spiritconsortium.org.