A Mixed-Signal System Design Methodology using SystemC-AMS for Automotive Audio Power Amplifiers

Skule Pramm, Joen Westendorp, Quino Sandifort
NXP Semiconductors, Nijmegen, the Netherlands
skule.pramm@nxp.com
joen.westendorp@nxp.com
quino.sandifort@nxp.com

Abstract— **This paper presents the first System Design Methodology based on SystemC-AMS for the development of Heterogenous Mixed-Signal Automotive Audio Power Amplifier ICs. The complexity of Automotive Audio Power Amplifier ICs is increasing steadily due to the increasing number of features/speakers and the introduction of functional safety. To cope with the higher application complexity, embedded software and microcontrollers are introduced to increase flexibility. The software is usually developed post-silicon, leading to additional throughput time and a sub-optimal system design due to compromises in the hardware-software architecture and partitioning. To improve system quality and shorten the development cycle, the hardware and software should be developed concurrently and the full system be verified before design freeze. Our proposed methodology has been applied successfully to control the signal-path of a Mixed-Signal Audio Power Amplifier.**

Keywords—SystemC-AMS; Mixed-Signal; Hardware; Embedded; Software; Co-development; System-Level; Design; Verification; Methodology; Flow; Automotive; Audio; Power; Amplifier; Requirements; Model-Based Design; Modelling; SCML; TLM; Use-Case; Functional Safety; Heterogenous; ESL; ASIC; RTI; SoC

I. INTRODUCTION

The new generation Automotive Audio Power Amplifiers ICs are fully **heterogeneous** systems, implementing both low-power digital and high-power analog hardware, controlled by software/firmware running on an embedded microcontroller. The **complexity** of these systems is steadily increasing [6][8][9] due to the increasing number of features, like e.g. more speakers/channels, increasing output power, advanced feedback, diagnostics, protections, microcontrollers, embedded software and functional safety. Existing languages and tools such as Mathematica, Matlab/Simulink and Verilog-AMS do not offer a single, consistent framework in which complex heterogeneous systems can be designed.

The overall chip development is multi-disciplinary, and traditionally different disciplines use **different languages, tools** and **environments**. The lack of a common modelling language leads to inconsistent functionality descriptions (e.g. models) at different disciplines, making it difficult to reach the more demanding automotive quality standards. System and IC architects typically use a graphical block diagram to draw the System and HW architecture, but this block diagram is completely disconnected from the design environment where the models and design are created, verified and integrated. Multiple analog models of the same IP are created in different modelling languages in different environments, making it difficult to keep them consistent. For example, a Simulink and VerilogAMS model cannot be simulated using a single simulation kernel. Co-simulation is possible, but complex, inconvenient and slow.

Therefore the **goal** of the new System Design Methodology is to improve the system design, product quality and reduce the time to market by using a Model-Based Design (MBD) [5] philosophy, where one System Model is used as the main communication medium between the different disciplines in the design team. The System Model also enables early System Verification, leading to less tapeouts and hardware debugging and therefore to a reduced time-to-market.

This paper is structured as follows. Section II introduces related work. Section III presents the methodology requirements. Section IV presents the new System Design Methodology including the Design Flow. Section V presents a simplified System-Level Model of an Automotive Audio Power Amplifier developed using our proposed Methodology and shows some preliminary simulation results. Finally, Section VI presents the conclusions and opportunities for future work.

II. Related Work

Our proposed System Design Methodology is based on [7] that presents an ESL top-down design methodology for mixed-signal systems using SystemC-AMS, but extends it by adding software, a microcontroller and a testbench environment, and applies it to a different domain, namely the Automotive Audio Power Amplifier domain.

III. Requirements

To reach the goal described in the introduction, the Top-Down System Design Methodology should cover the following use-cases:

- **Full System Verification**
  - Improves the product **quality**, because all sub-systems are verified together pre-silicon.
  - Reduces the **Time-to-Market**, because unnecessary spins are avoided.
- **Concurrent Hardware/Software Development**
  - Improves the **system design**, because hardware/software architecture and partitioning choices are made upfront before the implementation phase is entered.
  - Reduces the **Time-to-Market**, because activities run in parallel.
- **Executable Specification**
  - Improves the **system design**, because the full system can be simulated at any point in time during the development cycle.
  - Improves the **product quality**, because different disciplines share their understanding of the system in the system model. This avoids misinterpretation of text documents.

IV. Methodology

The proposed System Design Methodology consists of a model architecture, a modelling language, a test infrastructure, a modelling library and a design flow. These are all described in the coming paragraphs.

A. System Model Architecture

The architecture of a typical state-of-the-art Audio Power Amplifier consists of three domains: Software&Control, Digital Hardware and Analog Hardware, as illustrated in Figure 1. The **Software&Control** domain controls and receives status and diagnostic information from the digital and analog signal-path, and typically consists of a microcontroller, memory and a bus. The **Digital Hardware** domain does digital signal processing, like e.g. TDM/I2S reception, filtering, interpolation, amplification and digital PWM-modulation. The Analog Hardware domain does analog signal processing, like e.g. power amplification, protection and filtering.

Looking at Figure 1, the signal-path starts from the Stimuli Generator ① (an audio source on the same or a different IC) that generates a digital audio stream. The audio signal then passes through the digital signal processing ② and analog signal processing ③ in the Device-Under-Test (DUT) ⑨ and finally ends in the loudspeaker ④. The digital and analog signal processing is controlled from embedded control **software** ⑤ that is connected to a hardware-software interface ⑦ via a bus ⑥. The hardware-software interface ⑦ communicates with the digital and analog signal processing circuitry via a control path ⑧ implemented as a bus or a dedicated interface. Finally, the three domains are instantiated in a Testbench ⑩.

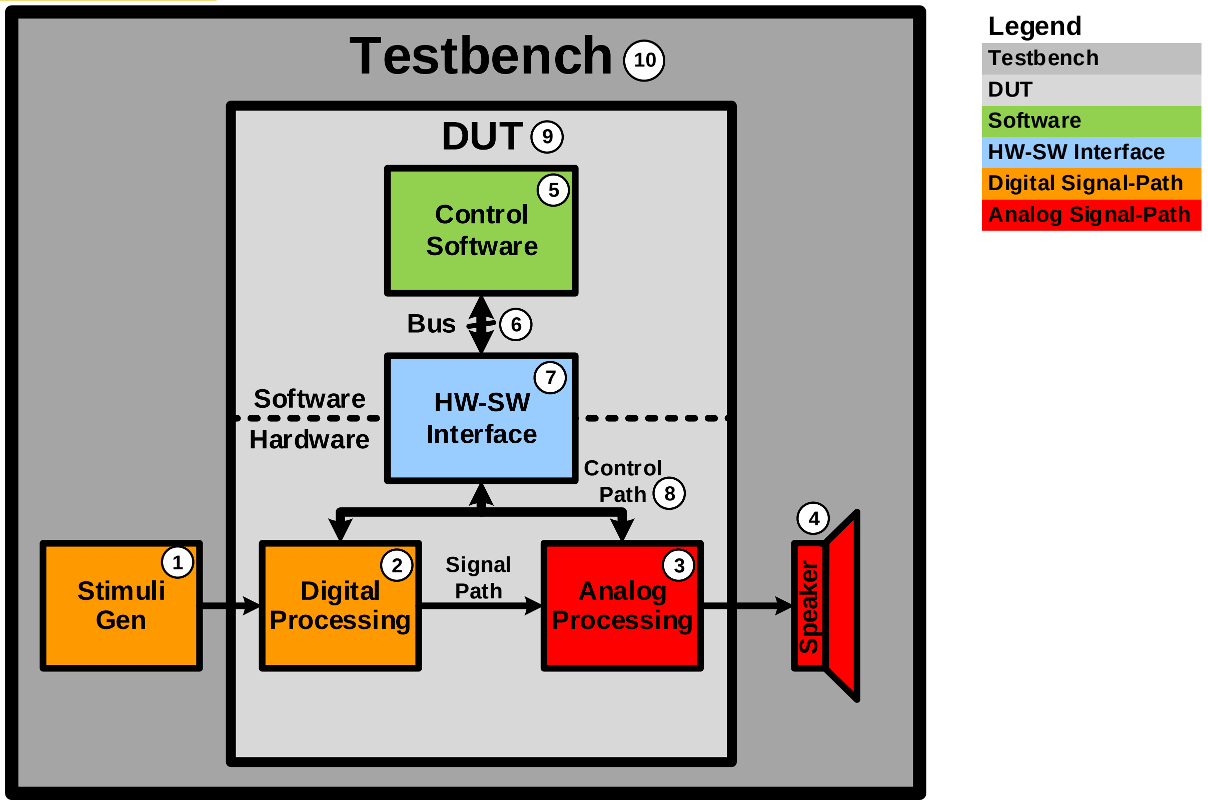


Figure 1. System Model Architecture with the signal-path at the bottom from left to right, and control-path from the top to the bottom.

B. *SystemC & SystemC-AMS*

The SystemC [1] and SystemC-AMS [2][3] languages are used to model the mixed-signal signal-path, due to the following reasons:

1) **Use-cases:** SystemC and SystemC-AMS are the only languages that can cover all the use-cases in one model.

2) **C-based:** SystemC and SystemC-AMS are both based on C, and therefore the full system including software and analog- and digital hardware can be described in only one language. This enables quick **Full System Verification** using only one simulator and it therefore avoids slow and complex co-simulation. The high simulation speed means that the full system can be verified over a lot of user scenarios within a reasonable period of time. The hardware model (Virtual Prototype) in SystemC and SystemC-AMS can be used by the Software team to verify if their software is working correctly with the hardware, while the software in C/C++ can be used by the Hardware team to verify if their hardware is working correctly with the software. This enables **Concurrent Hardware/Software Development**. The SystemC-AMS model represents the system requirements specification and is used as a golden reference for model refinement down-to implementation level. The SystemC-AMS model is an **Executable Specification**, because it can be simulated (in contrast to a text-based specification).

3) **Models of Computation (MoC):** SystemC and SystemC-AMS provides a wide range of Models of Computation, like Discrete Event (DE), Timed Data Flow (TDF), Linear Signal Flow (LSF) and Electrical Linear Network (ELN). The different MoC makes it easy to describe functionality on different abstraction levels in the **Top-Down Design Flow**, from fast virtual prototypes for early software development and algorithm design, via architecture-level and RTL-level (e.g. VHDL and SystemVerilog), downto transistor-level. Using different MoC for different sub-systems, enables an optimal trade-off between simulation speed and accuracy.

4) **Industry:** SystemC and SystemC-AMS are heavily used in the industry, making it possible to learn from others, reuse IP, share knowledge and cooperate.

5) **IEEE:** SystemC and SystemC-AMS are standard open-source IEEE languages, that are supported by many EDA tools and makes refinement downto implementation-level easier.

C. *Reusable Test Infrastructure (RTI)*

The Reusable Test Infrastructure (RTI) [10] is a software-configurable Testbench based on SystemC that can generate analog interfaces (e.g. signal generators, loads), monitors, checkers, microcontrollers (e.g. ARM Fastmodel), memories (e.g. RAM/ROM), timers, busses (e.g. Transaction Level Modelling (TLM)) and HW-SW Interfaces (e.g. SCML registers) to control the DUT. Verification or Application Software is compiled into the memories and executed by the microcontroller on the DUT without any knowledge about the Testbench itself.

D. *Design Flow*

The Design Flow is illustrated in Figure 2 and it consist of the following (concurrent) steps. **Step 1** is to create a **SW Specification** ① and **SW-code** ②. **Step 2** is to create a **HW-SW Interface Specification** ③ which is used to generate the SW header file ④ and HW-SW interface ⑤. **Step 3** is to create a **Model Specification** ⑥ and create SystemC-AMS models ⑦ of the mixed-signal **Signal-Path**. **Step 4** is to integrate the HW-SW interface and SystemC-AMS models together in the HW toplevel ⑧. The DUT instantiates the software and hardware ⑨. A Testbench Specification ⑩ is created to generate the Controller ⑪ which is instantiated with the DUT in the Testbench ⑫. Finally the Testbench is simulated, analyzed and debugged ⑬.

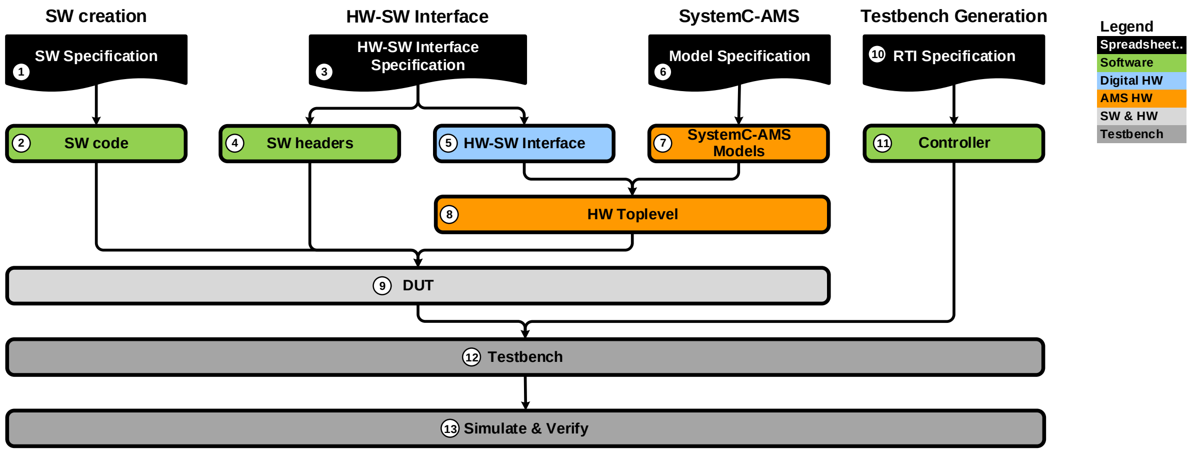


Figure 2. Design Flow

E. *SystemC Modeling Library (SCML)*

The SystemC Modeling Library (SCML) [4] is a library used to describe HW-SW interfaces both on a high abstraction-level (e.g. SystemC TLM) without fixing the physical interface, and low abstraction-level (e.g. SystemC registers). SCML integrates easily with our SystemC-AMS model, because it is also based on SystemC.

V. AUDIO POWER AMPLIFIER IMPLEMENTATION EXAMPLE

To demonstrate the feasibility of our proposed methodology, we have built a System Model of a simplified Automotive Audio Power Amplifier using the ingredients from the previous chapter. Our goal is to use software to control the gain of the signal-path.

A. *Audio Power Amplifier Architecture*

The Audio Power Amplifier architecture is shown in Figure 3 and it is based on the System Model Architecture shown in Figure 1.

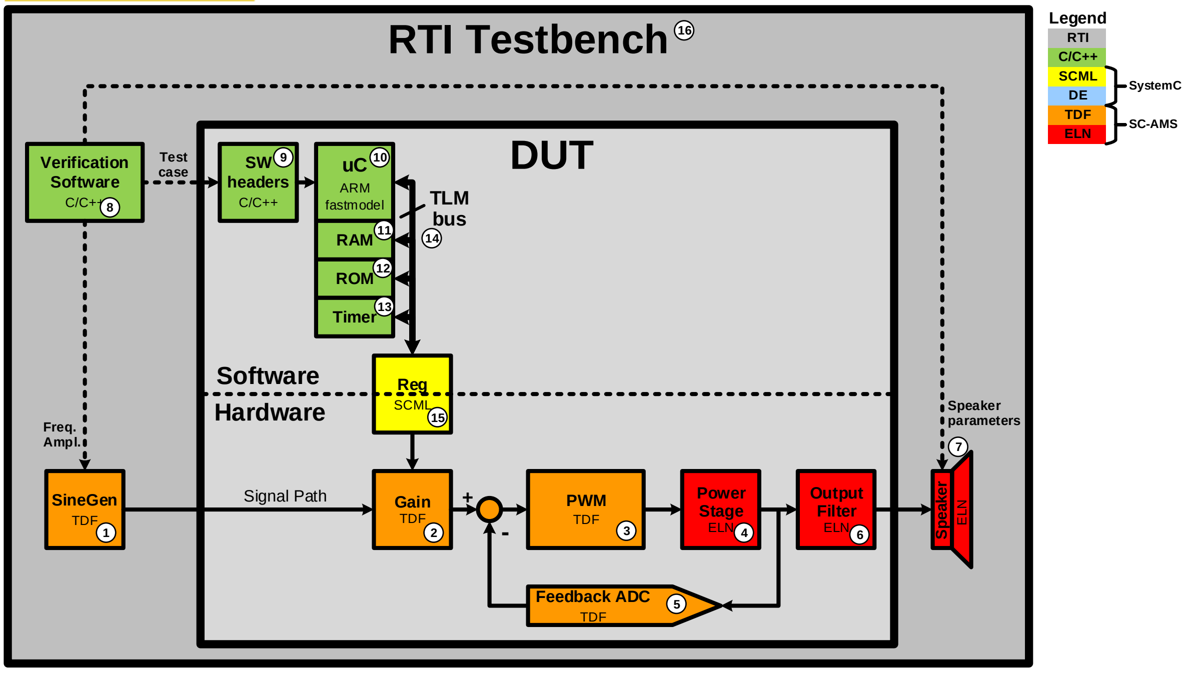
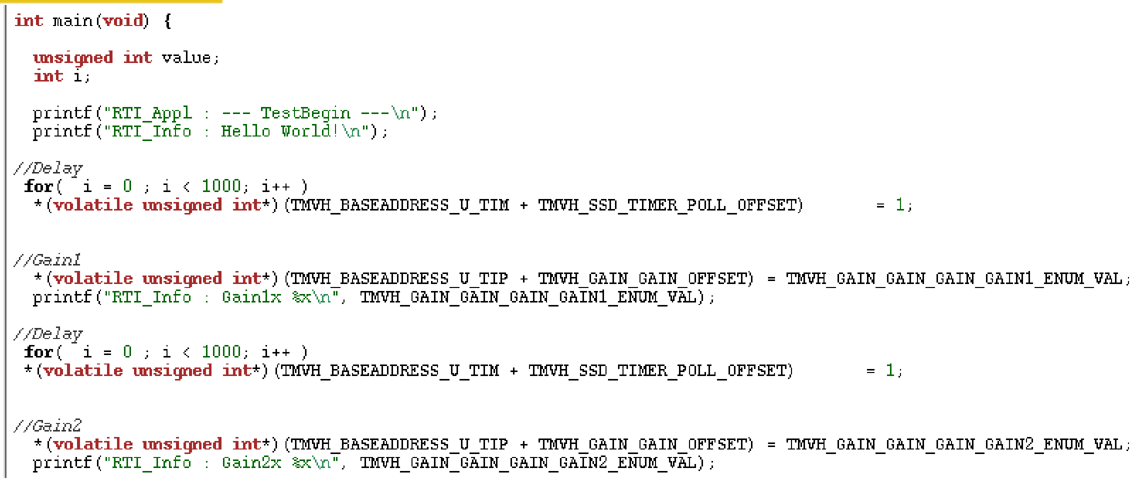


Figure 3. Audio Power Amplifier Architecture including Testbench (grey), Control-Software domain (green), Signal-Path domain (orange/red) and a register (yellow) on the Software-Hardware interface. The Signal-Path goes from the stimuli generator bottom left (orange), via the controlled gain-block, PWM-modulator, powerstage and output filter, to the speaker load at the bottom right. The Feedback ADC closes the controlloop.

The **signal-path** starts with a Sinewave Generator ① that generates a fixed frequency/amplitude sinewave that is amplified in the Variable Gain block ②. The amplified sine is sent to a PWM-modulator ③ which converts the sine into a PWM signal for the Powerstage ④. The Powerstage amplifies the PWM and sends it through the Output Filter ⑥ to filter out the PWM carrier before going to the speaker ⑦. To improve audio quality, the Powerstage output is fed back to the PWM-modulator via an ADC ⑤. The low-power digital signal-path is modelled in SystemC-AMS TDF iso. SystemC DE, due to real-time requirements for the feedback loop and for high simulation speed. TDF simulates the loop quicker, because it calculates the loop schedule before starting the simulator. This in contrast to DE, which calculates the schedule during simulation, resulting in multiple iterations at the same time point causing delta-cycles. The high-power analog signal-path on the other hand, is modelled in ELN for high accuracy analog performance (analog limits the audio performance) and a close link to the analog implementation and analog design team.

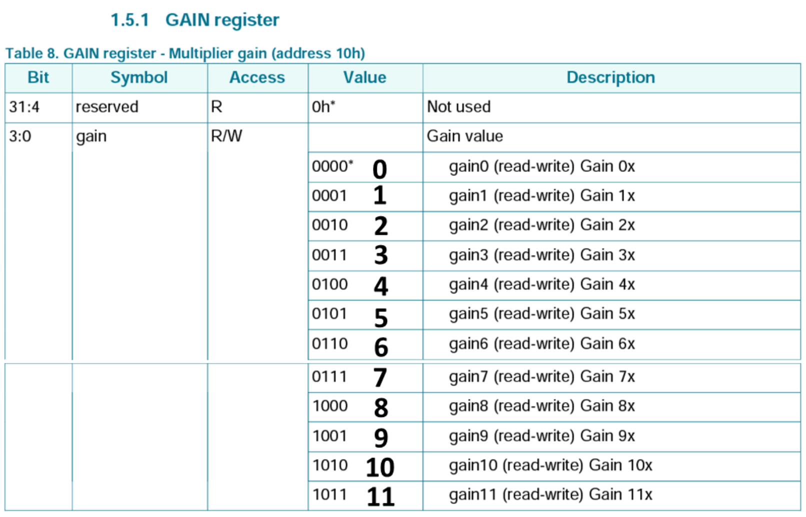
The **control-path** starts with **Verification Software** ⑧ written in C/C++ that communicates with the **Microcontroller** ⑩ via the **Software Headers** ⑨. The verification software controls the gain of the signal-path and a code snippet is shown in Figure 4.



```
int main(void) {

  unsigned int value;
  int i;

  printf("RTI_Appl : --- TestBegin ---\n");
  printf("RTI_Info : Hello World!\n");

//Delay
 for(  i = 0 ; i < 1000; i++ )
  *(volatile unsigned int*)(TMVH_BASEADDRESS_U_TIM + TMVH_SSD_TIMER_POLL_OFFSET)        = 1;


//Gain1
  *(volatile unsigned int*)(TMVH_BASEADDRESS_U_TIP + TMVH_GAIN_GAIN_OFFSET) = TMVH_GAIN_GAIN_GAIN_GAIN1_ENUM_VAL;
  printf("RTI_Info : Gain1x %x\n", TMVH_GAIN_GAIN_GAIN_GAIN1_ENUM_VAL);

//Delay
 for(  i = 0 ; i < 1000; i++ )
 *(volatile unsigned int*)(TMVH_BASEADDRESS_U_TIM + TMVH_SSD_TIMER_POLL_OFFSET)        = 1;


//Gain2
  *(volatile unsigned int*)(TMVH_BASEADDRESS_U_TIP + TMVH_GAIN_GAIN_OFFSET) = TMVH_GAIN_GAIN_GAIN_GAIN2_ENUM_VAL;
  printf("RTI_Info : Gain2x %x\n", TMVH_GAIN_GAIN_GAIN_GAIN2_ENUM_VAL);
```

Figure 4. Verification software code snippet

The verification software may also control e.g. the frequency and amplitude of the stimuli generator, the speaker parameters and include monitors and checkers to enable system-level verification. The RTI is used to generate the Testbench, Software Header file, Microcontroller, RAM, ROM, Timer, TLM-bus and Register. The **Microcontroller** ⑩ uses an ARM Fastmodel, while the **RAM** ⑪ and **ROM** ⑫ use generic **TLM memory** models. The **Timer** ⑬ is used by software to control the delay between commands. The **bus** ⑭ is implemented in TLM on a high abstraction level to make it flexible. The **Hardware-Software Interface Specification** is used to generate the **Software Header file** ⑨ and a **register** ⑮ in **SCML** using RTI [4]. The Hardware-Software Interface Specification is shown in Figure 5.



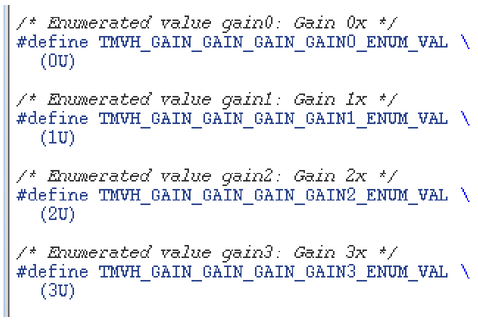
1.5.1 GAIN register

Table 8. GAIN register - Multiplier gain (address 10h)

Bit	Symbol	Access	Value		Description
31:4	reserved	R	0h*		Not used
3:0	gain	R/W			Gain value
			0000*	0	gain0 (read-write) Gain 0x
			0001	1	gain1 (read-write) Gain 1x
			0010	2	gain2 (read-write) Gain 2x
			0011	3	gain3 (read-write) Gain 3x
			0100	4	gain4 (read-write) Gain 4x
			0101	5	gain5 (read-write) Gain 5x
			0110	6	gain6 (read-write) Gain 6x
			0111	7	gain7 (read-write) Gain 7x
			1000	8	gain8 (read-write) Gain 8x
			1001	9	gain9 (read-write) Gain 9x
			1010	10	gain10 (read-write) Gain 10x
			1011	11	gain11 (read-write) Gain 11x

Figure 5. Hardware-Software Interface Specification for the Gain block

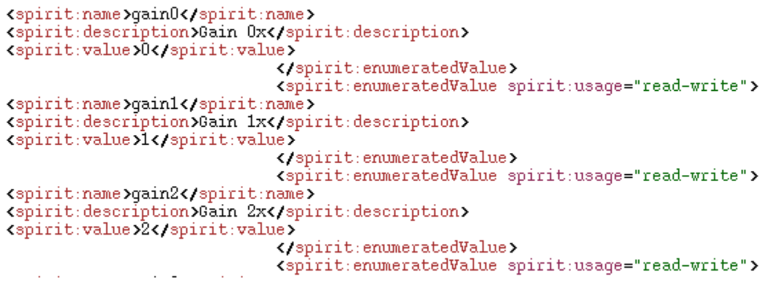
The Software header file is generated from the Hardware-Software Interface Specification, and a code snippet is shown in Figure 6.



```
/* Enumerated value gain0: Gain 0x */
#define TMVH_GAIN_GAIN_GAIN_GAIN0_ENUM_VAL \
   (0U)

/* Enumerated value gain1: Gain 1x */
#define TMVH_GAIN_GAIN_GAIN_GAIN1_ENUM_VAL \
   (1U)

/* Enumerated value gain2: Gain 2x */
#define TMVH_GAIN_GAIN_GAIN_GAIN2_ENUM_VAL \
   (2U)

/* Enumerated value gain3: Gain 3x */
#define TMVH_GAIN_GAIN_GAIN_GAIN3_ENUM_VAL \
   (3U)
```

Figure 6. Software header file code snippet

The register model is generated in SCML based on the Hardware-Software Interface Specification. A code snippet of the xml-description is shown in Figure 7.



```
<spirit:name>gain0</spirit:name>
<spirit:description>Gain 0x</spirit:description>
<spirit:value>0</spirit:value>
                            </spirit:enumeratedValue>
                            <spirit:enumeratedValue spirit:usage="read-write">
<spirit:name>gain1</spirit:name>
<spirit:description>Gain 1x</spirit:description>
<spirit:value>1</spirit:value>
                            </spirit:enumeratedValue>
                            <spirit:enumeratedValue spirit:usage="read-write">
<spirit:name>gain2</spirit:name>
<spirit:description>Gain 2x</spirit:description>
<spirit:value>2</spirit:value>
                            </spirit:enumeratedValue>
                            <spirit:enumeratedValue spirit:usage="read-write">
```

Figure 7. Register model code snippet (xml)

To simulate, first the Verification Software is compiled into the RAM/ROM, then the Microcontroller sets the register via the TLM-bus, and finally the register controls the Gain block via a dedicated interface.

B. *Design Flow*

The Design Flow is illustrated in Figure 8 and it is based on the Design Flow in Figure 2. The flow consist of the following steps: **Step 1** is to create the SW Specification ① which is used to create SW code ②. **Step 2** is to create the Hardware-Software Interface Specification ③ in a spreadsheet and use it to generate the software header file ④ and register (SCML) ⑤. **Step 3** is to create a **Model Specification** ⑥ and create SystemC-AMS models ⑦ of the mixed-signal **Signal-Path**. **Step 4** is to integrate the SCML register and SystemC-AMS models together in the HW toplevel ⑧. **Step 5** is to create an RTI Specification ⑨ which is used to create the Microcontroller ⑩, RAM/ROM ⑪, Timer ⑫, the DUT ⑬ and the Testbench ⑭. Finally the Testbench is simulated, analyzed and debugged ⑮.

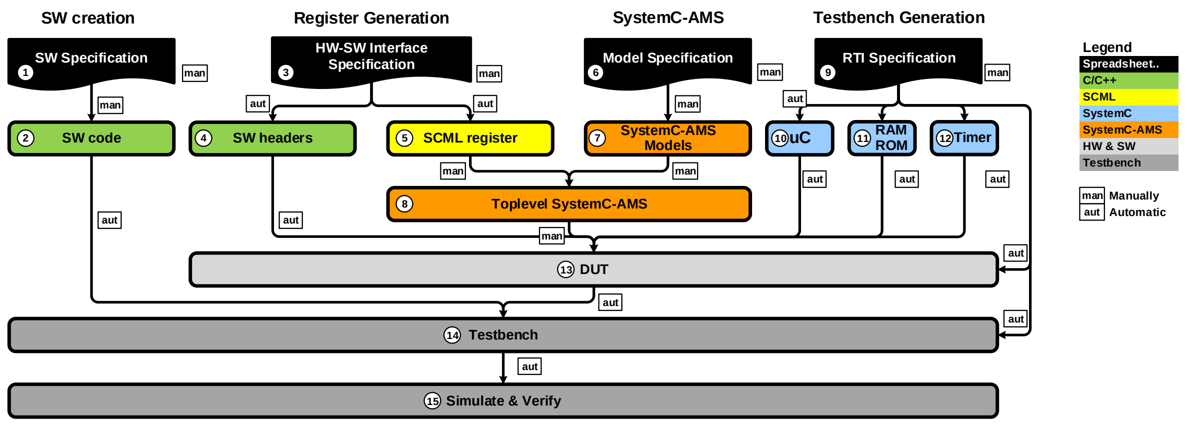


Figure 8. Design Flow for the Audio Power Amplifier

C. *Audio Power Amplifier Signal-Path schematic*

The schematic of the Audio Power Amplifier Signal-Path is shown in Figure 9. All blocks are modelled in SystemC-AMS except for the register which uses SCML. The signal-path goes from the signal source ① top left , via the gain block ② controlled by the register ③ and TLM bus ④ bottom left. The low-power signal then passes through the PWM-modulator ⑤ and power stages ⑥ to the LC-filter ⑦ and Speaker load ⑧ at the right side. The feedback ⑨ is found in the middle of the schematic.

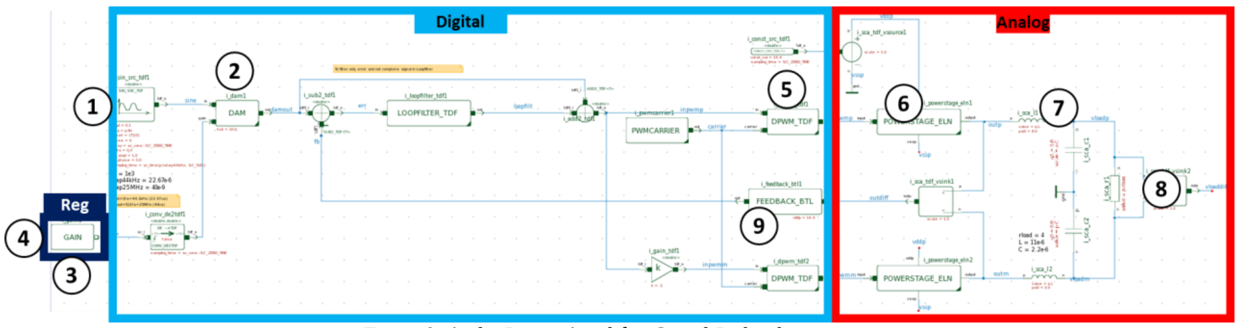


Figure 9. Audio Power Amplifier Signal-Path schematic

D. *Design Hierarchy*

The Design Hierarchy is shown in Figure 10 and Figure 11. The toplevel is called chip_core ⓪ and it consists of six Sub-Systems: ① top_eln is the SystemC-AMS model with the Signal-Path, ② u_bus is the TLM-bus, ③ u_cpu is the ARM fastmodel, ④ u_ram is the RAM, ⑤ u_rom is the ROM, and u_tim ⑥ is the timer

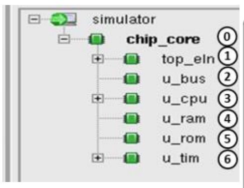


Figure 10. Design hierarchy

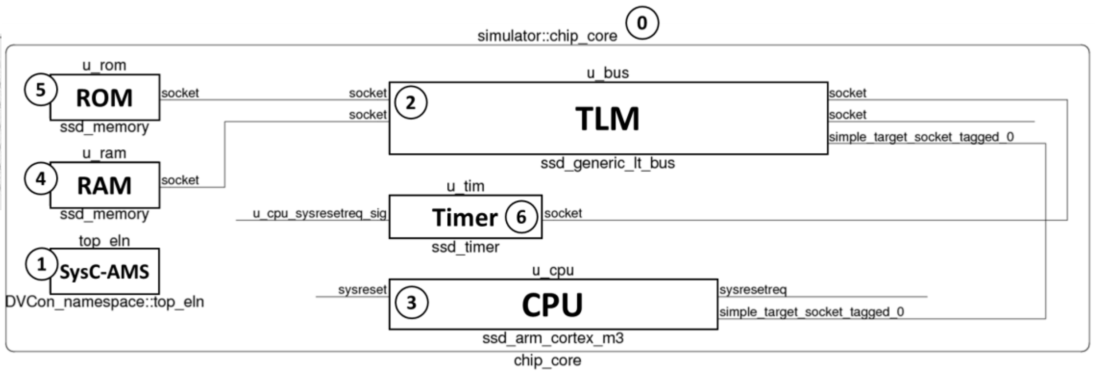


Figure 11. Toplevel schematic including SystemC-AMS model, CPU, TLM-bus, Timer, RAM and ROM.

E. *Simulation Results*

The full system is linked and simulated on the commandline. The simulation waveforms in Figure 12 show that the amplitude of the differential output voltage (green waveform third from the bottom) increases correctly with the increasing gain set by software. This 12 ms simulation takes approx. 12 seconds wall clock time to run, giving a simulation speed of 1ms/s. A dozen scenarios are foreseen for the full audio power amplifier chip to verify the different functions, and this simulation speed is sufficient for short scenarios (millisecond range) like e.g. boot, startup, modulation schemes, diagnostics, protections, shutdown. For long scenarios (seconds range) like e.g. a boot-startup-shutdown scenario or a scenario where a full song/speech-fragment is sent through the amplifier, further optimization of the models is anticipated.

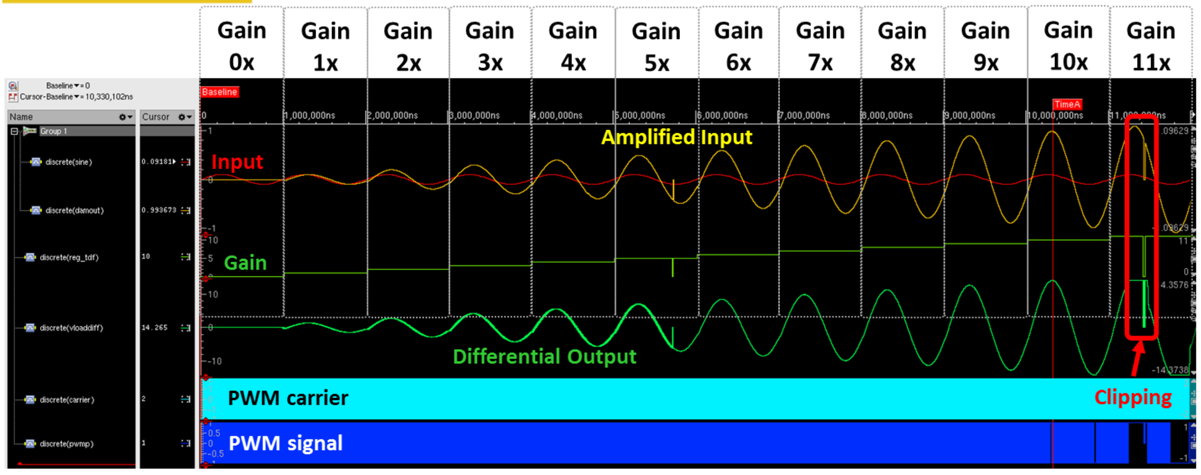


Figure 12. Simulation results showing from top to bottom the sine audio input (red), amplified sine (yellow), register settings (light green staircase), differential output load voltage (green), PWM carrier (turquoise) and PWM signal (blue).

VI. Conclusions

A new ESL Design Methodology based on a Model-Based Design Philosophy, SystemC-AMS and a standardized approach for register modeling and testing (RTI), has been presented. The requirements for the methodology were identified and all parts of the methodology described. The methodology has been applied successfully to the creation of a System Model of an Automotive Audio Power Amplifier, where software is used to control the gain of the signal-path. The current flow is partly automated, so full automation is a goal for future work.

References

[1] Accellera, "Standard SystemC Language Reference Manual", https://www.accellera.org/downloads/standards/systemc/

[2] Accellera, "Standard SystemC AMS extensions 2.0 Language Reference Manual", https://www.accellera.org/images/downloads/standards/systemc/SystemC_AMS_2_0_LRM.pdf

[3] Accellera Systems Initiative, "SystemC AMS 2.0 Extensions", https://www.accellera.org/community/systemc/about-systemc-ams

[4] Synopsys, "SystemC TLM Models", https://www.synopsys.com/verification/virtual-prototyping/virtual-prototyping-models/systemc-tlm-models.html (SCML)

[5] Roger Aarenstrup, "Model-Based Design", The MathWorks, 2015, https://nl.mathworks.com/campaigns/offers/managing-model-based-design.html.

[6] M. Barnasconi, M. Dietrich, K. Einwich, T. Vörtler, J-P. Chaput, M-M. Louërat, F. Pêcheux, Z. Wang, P. Cuenot, I. Neumann, T. Nguyen, R. Lucas and E. Vaumorin, "UVM-SystemC-AMS Framework for System-Level Verification and Validation of Automotive use Cases", IEEE Design & Test 2015, http://ieeexplore.ieee.org.

[7] M. Barnasconi and S. Adhikari, "ESL Design in SystemC AMS", DAC 2017.

[8] J.S. Barros, V.H. Schulz and D.V. Lettnin, "An Adaptive Closed-loop Verification Approach in UVM-SystemC for AMS Circuits", Symposium on Integrated Circuits and Systems Design (SBCCI) 2018, https://ieeexplore.ieee.org/document/8533229.

[9] T. Vörtler, K. Einwich, "Using Constraints for SystemC AMS Design and Verification", DVCon Europe 2018.

[10] E. de Kock, J. Verhaegh and S. Amougou, "A Configurable Test Infrastructure using a Mixed-Language and Mixed-Level IP Integration IP-XACT Flow", CODES+ISSS 2012.