A Push-button Idea to GDS-II SoC Design Flow

Habiba Gamal[1], Amr Gouhar[2], and Mohamed Shalan[1]
[1]The American University in Cairo, New Cairo, EGYPT
[2]eFabless Corporation, San Jose, USA

***Abstract*—SoCGen is a system on chip (SoC) design automation tool that takes in a simple JavaScript Object Notation (JSON) description of the system's components, connections and structure. The tool then outputs the Verilog HDL for the SoC, the intermediate files of hardening and the final GDS-II. SoCGen utilizes OpenLANE, an automatic RTL to GDS-II physical design flow. SoCGen is tailored for SoCs intended for internet of things (IoT) and deep embedded applications.**

***Index Terms*—SoC, Design, Automation, System on Chip, EDA, AMBA, AHB, APB, Bus, Master, Slave, GDSII, ASIC, flow, RTL, IP**

I. Introduction and Problem Statement

System on chip (SoC) design and manufacturing has become one of the necessities of the modern Integrated Circuit (IC) design world. In this new world, time is critical. When the designer finds a bug after a long process of creating a GDS-II, the designer has to go through a lengthy process of reviewing Verilog codes, followed by re-hardening the chip. However, because of the systematic nature of the process, there has been a rising interest in developing open-source tools for carrying it out. In addition, there are similarities between different SoCs in their standard intellectual properties (IPs). Henceforth, came the idea of automating the Verilog RTL model generation of an SoC utilizing a verified open-source IPs library that is easily extensible by the users, followed by using OpenLANE for the GDS-II generation. Thus, automating the full flow from creating the hardware description language (HDL) code of the design to generating GDS-II.

SoCGen is an SoC design automation tool that takes a simple, schematic-like description of the SoC architecture and generates the final GDS-II without any human intervention. This tool is useful in many scenarios, two extremities of which will be discussed. The first scenario is enabling end-users who are not experienced in SoC design to easily tape-out chips with the desired system behavior. All the user has to do is describe the system architecture in a simple format, then start the flow. Firstly, the verified Verilog files are generated, followed by generating the GDS-II utilizing OpenLANE. The end-user can effortlessly make modifications to the system archirecure and generate the GDS-II again.

The second scenario is experienced SoC designers using SoCGen. In this case, the designer can review the generated files at each intermediate step of hardening then do the required modifications to address any concerns. In addition, experienced designers can benefit from the flexibility of the tool and the ease of reiterating the design through the flow. This being said, the designer can test different configurations or system architectures to achieve the required system behavior, while meeting certain constraints. Therefore, SoCGen will allow designers to focus more on the block system design, while leaving the hectic process details for SoCGen.

Therefore, the problem discussed in this paper is automating the whole process of designing an SoC. Nevertheless, there are many challenges when tackling such a universal problem.

1) Attaining a comprehensive description methodology to represent any potential IP adequately.
2) Circumscribing how the connections between the different IPs, buses, and masters are amply described.
3) Parsing the representation and generating the proper format and connections of all the Verilog modules that make up the SoC.
4) Adhering to an acceptable runtime period to fulfill the purposes of the automation process and render it attractive to designers. Thus, the efficiency of the process is critical, and its maintenance is inevitable to keep the project updated with the most modern technologies.
5) Usability and achieving the *no-man-in-the-loop* goal, while ensuring the accessibility of all the files and generated outputs of the intermediate stages of the process to the designers to discern and reform.
6) Avoiding huge compromises in the quality of the work produced, if compared with a fully manual design process. Hence, we articulate our project's objectives as versatility, usability, extensibility, reliability, competence, and efficiency.

Therefore, this project aims to create a universal description methodology that is comprehensive enough to include all potential IPs, masters, and SoC architectures intended for internet of things (IoT) and deep embedded applications. In addition, SoCGen seeks to automate the processes of signals connections, hardening macros, and creating the chip-level GDS-II. SoCGen intends to achieve all the previous objectives in an efficient and versatile manner while having a flexible infrastructure with a user-friendly interface.

The remainder of this paper goes as follows: in the next section, we outline the current features of SoCGen and the implementation details of some of its components. Section 3 describes the integration with the OpenLANE flow to fulfill the goal of automatic GDS-II generation. In Section 4, we describe the experiments tried by SoCGen. Finally, we conclude and explore viable future work.

II. Features and Implementation Details

SoCGen supports AMBA Advanced High Performance Bus (AHB) and AMBA Advanced Peripheral Bus (APB). Masters can only be placed on AHB's. For details about AMBA AHB and APB, refer to chapter 3 in [1]. However, SoCGen is flexible when it comes to the number of masters that can exist on the same bus. Furthermore, a single master can be connected to multiple AHB's. As for the APB sub-systems, they can only be placed under AHB's. For the communication between AHB and APB sub-systems, a bridge is generated. The aim behind the AHB-APB bridge is to convert the AHB signals into APB signals and vice versa. For instance HWRITE, HADDR and HWDATA are changed into PWRITE, PADDR and PWDATA to be used by the APB peripherals. The bridge also converts the APB signals into AHB signals. For instance, PREADY and PRDATA are changed into HREADY and HRDATA. In a nutshell, the AHB-APB bridge enables the communication between AHB and APB by making the APB sub-system act like a slave on the AHB, and making the AHB act like a master on the APB. An arbitrary number of APB subsystems can exist on the same AHB.

Regarding the hierarchy of the generated system, at the top level exists the board which is equivalent to the testbench generated in the HDL of the SoC. This includes the SoC chip, or an instance of the SoC in the generated HDL. The SoC includes the SoC core which encompasses the masters and the AHBs. The AHB instantiates the APB sub-systems. The hierarchy is shown in Fig. 1.

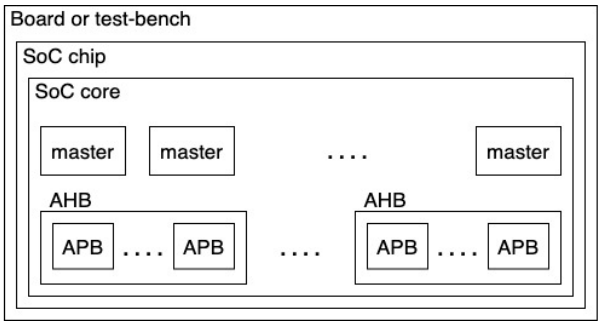

Fig. 1. System hierarchy

As for the peripherals, SoCGen supports both bus-specific peripherals and generic peripherals. Generic peripherals are components that are not compatible to a specific bus. SoCGen supports the use of generic peripherals through automatically generating bus-specific wrappers. The aim of said wrappers is to map the ports and the registers of the peripheral into bus signals, through assigning each peripheral a base address, and each register an offset address. Therefore, each register becomes addressable by the master on the bus. The number of address bits, as well as, the base address of the peripherals and the APB sub-systems, in addition to the register offsets, can all be specified by the designer in the JSON.

The peripherals can be soft modules or hard macros. This characteristic can be specified in the JSON description of the IP. For generic soft modules and hard macros, their wrappers are placed on the bus. However, soft modules are instantiated and directly connected to the wrappers at the bus level. In contrast, for hard macros, the wrapper signals are propagated to the SoC core where hard macros are placed and connected to said wrapper signals. Regarding bus-specific soft modules, these are placed directly on the bus. On the other hand, for bus-specific hard macros, the bus signals are propagated to the SoC core where the hard macros are placed and connected to said bus signals.

The library of IPs encompasses description of different IPs. The IP description includes, but is not limited to, its external ports, registers, connection to other modules, connection to input/output (I/O) pads and bus type. The library of IPs currently includes a set of verified open-source components including: UART, timer, PWM, watchdog, SPI master, I2C master, GPIO, QSPI flash controller and SRAM controller. Some of these IPs are generic, while others are bus-specific. Some of the IPs like the I2C master, SPI master, timer, PWM and GPIO have c files and header files that are used during testing with real masters. In addition, SoCGen has open-source verification IPs for PWM, GPIO, the I2C master and the SPI master. The IPs library is easily extensible by end-users. The IPs library facilitates reusing IP blocks that are commonly similar in different SoC designs.

For the communication between the SoC core and the outside world, I/O pads are needed. SoCGen by default has an I/O pads library which describes the ports of the I/O pads. The library of I/O pads includes four different types of general I/O pads that are not process specific. These are: analog, digital input/output, digital input and digital output. The designer can easily add to this library process-specific I/Os, or his/her own set of general I/O pad definitions. Moreover, the designer can specify the placement level of components that are not on the bus, as well as, the connections of the external ports of bus peripherals to components that are not on the bus. For instance, an SRAM controller would be placed on AHB and its external ports would be connected to the SRAM which is placed inside the SoC core. This can be specified by the JSON description of the SoC. Furthermore, a flash controller would be placed on AHB while its external ports would be connected to the flash that is placed in the testbench. In different terms, the flash is placed outside the chip of the SoC, on board. In addition, the external ports of peripherals like the SPI master, I2C master and GPIO, which can be placed on APB, may be connected to verification IPs (VIPs) which simulate connections to components that are placed outside the SoC chip, on board. All these different connectivity examples can be specified in the JSON for the SoC.

As for testing, in case of the presence of a real master (e.g. ARM Cortex M0), SoCGen has a debug peripheral for self-checking testbench generation that the user can place in a design. The debug peripheral is simply a debug register used during testing. A specific pattern is used to indicate a passed test, while another pattern is used to indicate a failed test. Furthermore, SoCGen automatically generates a dummy mas-

ter in case of the absence of a real master. The dummy master performs read and write transactions with every register in the system to verify the expected system communication requirements. SoCGen also outputs hierarchical testbenches. In other words, SoCGen outputs a testbench for each APB-subsystem, a testbench for each AHB system and a top-level testbench for the whole SoC. The specifications of the top-level testbench can be speficied in the JSON for automatic generation. For instance, the number of ticks can be specified and the path of the file that will be loaded into the flash.

SoCGen also has a masters library that includes definitions for ARM Cortex M0, ARM Cortex M3 and N5, an open-source core. Currently, the master has to be compatible with AHB. The masters library includes description for the connections of the master's bus interface to the bus signals, as well as, the default connections of the master ports and the interrupt lines. The masters library is easily extensible by the designer. In case of the absence of a real master, if a user only wants to test the system structure without the real master, a dummy master is automatically generated by SoCGen as described before.

An arbiter is generated in case of the presence of multiple masters on the same bus. The role of the arbiter is to grant the bus to only one master requesting the bus based on the priority of the master. In AMBA terms, the arbiter takes the HBUSREQ signal from each master and generates the HGRANT signal. The master that is granted the bus starts the bus transaction. The bus signals generated by said master that are sent to AHB are selected by multiplexer, as shown in Fig. 2. The mentioned arbiter and multiplexer are taken from GEN_AMBA [7]. GEN_AMBA does not generate complete SoCs, but rather it generates bus skeletons for AMBA AHB, APB and AXI. Therefore, it does not connect subsystems, generate bus wrappers for generic peripherals, or connect the masters and peripherals to the bus ports. In addition, GEN_AMBA offers less flexibility than SoCGen since it does not allow the user to specify the addresses of peripherals or buses.

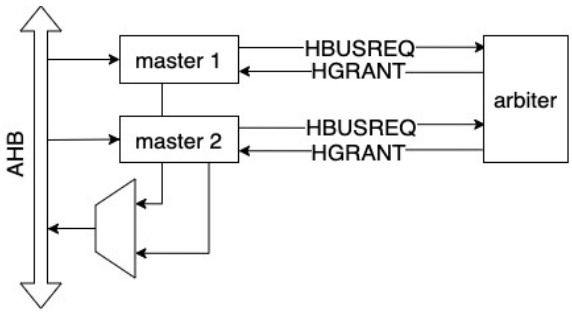

Fig. 2. Multiple masters on the same AHB

In the case of having the same master present on multiple buses, a multiplexer is needed to select between the return signals of each bus based on the value of the address signal (HADDR). In a particular transaction, HADDR would be meaningful to a certain bus while it would be in the invalid range for the other buses. The multiplexer will filter the return signals of each bus, using HADDR of the transaction as the select line, to select the corresponding bus signals. The block diagram is shown in Fig. 3.

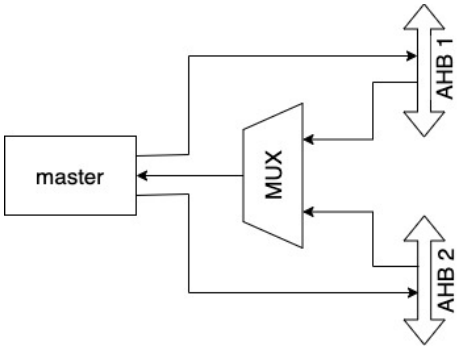

Fig. 3. One master on multiple AHB's

III. Integration With OpenLANE

Once SocGen generates the RTL, OpenLANE handles the remaining of the process [6]. OpenLANE is an automated RTL to GDS-II flow that utilizes many open-source tools, including Yosys, OpenROAD, Magic, Netgen, and its custom scripts [2] [3] [4] [5].

As for how SoCGen uses OpenLANE, firstly, hard IPs are hardened. If there are no hard IPs in the design, this step is skipped. Then the SoC is flattened with the soft IPs and hardened as a core without the pad frame. The hard IPs are included as macros and are placed as-is inside the core. Finally, the chip is hardened with the pad frame.

Currently, the integration of SoCGen with OpenLANE is not fully automated. In other terms, OpenLANE was manually run utilizing the RTL output of SoCGen. However, in order to achieve this simple integration, SoCGen needs to automatically call OpenLANE after the HDL of the SoC is generated. In addition, each of the hard IPs in the IPs library should have a configuration file included in its library description. The configuration files should be optimized before adding them to the library, and thus the IPs could be optimally hardened without the need for any input from the designer; however, the designer has full control to enforce any modifications. SoCGen will generate default configurations for the SoC that would allow it to run through the OpenLANE flow. Alternatively, the user could provide these configurations. In both cases, the user could utilize the exploration scripts of OpenLANE or the analysis of the generated results to optimize those configurations to obtain a DRC and LVS clean GDS-II or achieve better results based on a user-specific metric. After this step, the generated core is treated as a macro on the chip and is hardened along with the pad frame to generate the GDS-II for the whole chip. The configurations of this step should also be automatically generated by SoCGen or optionally provided by the user. In either case, the user will have the ability to modify these configurations to achieve the design goals. The final results of the generation would be the GDS-II of the

chip, the LEF/DEF views, and all the intermediate files and logs generated by OpenLANE. OpenLANE is integrated with the SKY130 Open PDK, including different libraries for the standard cells [8]. However, the user could use any other PDK as long as the user configures OpenLANE to use it.

IV. Experimentation

SoCGen was used to create different SoC designs. One of them is a system with a single master placed on AHB. A flash controller and SRAM controller are also placed on AHB, along with a GPIO with 16 I/O ports. There is an APB subsystem on said AHB that includes I2C master, SPI master, timer and PWM. The I2C master, SPI master, PWM and GPIO are connected to verification IPs that exist in the testbench, or in other words on board outside the SoC chip. The flash controller is also connected to the flash that is placed on board and not on the SoC chip. The SRAM controller is connected to the SRAM that is placed inside the SoC core. The external ports of the I2C master, GPIO, SPI master, flash controller and PWM are connected to I/O pads that are placed on the SoC chip outside the SoC core. This system structure can all be specified in the JSON. The described SoC is shown in Fig. 4.

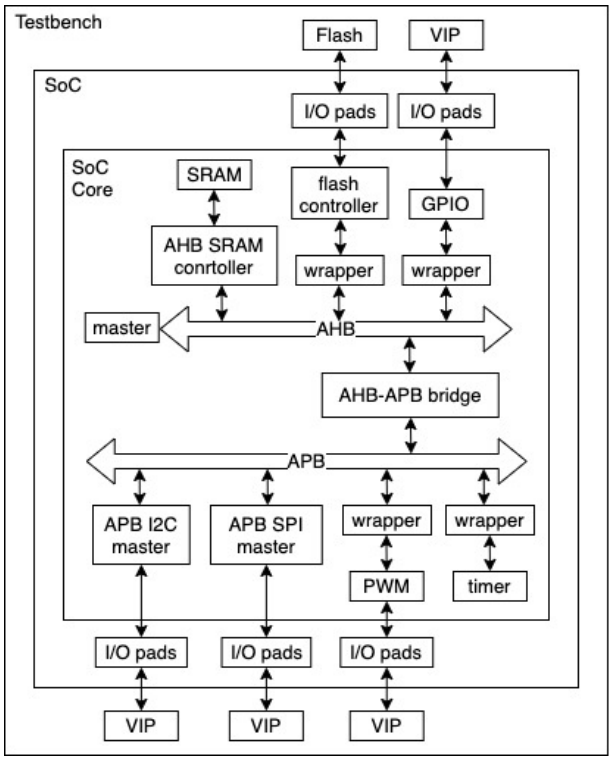

Fig. 4. A system on a chip architecture

Upon the press of a button, the Verilog files for this system are generated, along with a self-checking testbench since the debug peripheral was included in the design. In a couple of seconds, the whole SoC is generated and tested for verification of expected system behavior without any human intervention. The described system was generated and verified using three different masters: ARM Cortex M0, ARM Cortex M3, and N5, an open-source core.

The GDS-II for the SoC core in Fig. 4, using ARM Cortex M0 as the master, was generated using OpenLANE in less than 14 hours. The design is composed of about 121K cells. The die area is 3mm x 3mm. The design is DRC and LVS clean. The hardening was done using the high density standard cell library from the SKY130 Open PDK [8]. The layout of the automatically generated GDSI-II is shown in Fig. 5.

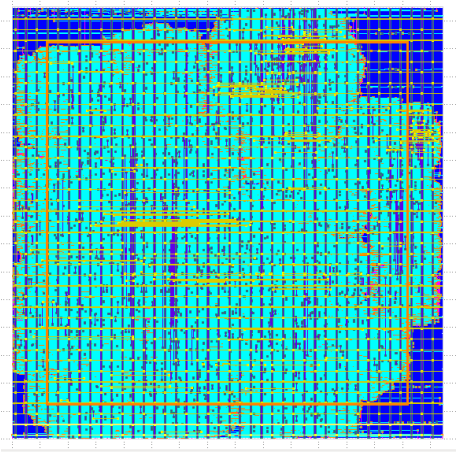

Fig. 5. Layout of the generated SoC

V. Conclusion

This paper presents an SoC design automation tool that starts with a simple description of the system architecture and automatically generates the SoC layout utilizing the SKY130 Open PDK. The tool has been tested by generating a verified SoC composed of open-source components and its hardening was DRC and LVS clean.

For the future work of this project, we aim to support more bus types such as the AMBA AIX-4 and the Wishbone buses. Also, we plan to add early stage estimators for area, power and clock frequency.

VI. Acknowledgment

SoCGen is a publicly available, open-source project that can be found at https://github.com/habibagamal/SoC_Automation. Taking the extra mile of generating GDS-II from RTL would not have been possible without the efforts of the OpenLANE development team, especially Karim Farid and Ahmed Ghazy.

References

[1] Joseph Yiu, "System-on-Chip Design with Arm Cortex-M Processors," Arm Education, 2019.
[2] Clifford Wolf and Johann Glaser, "Yosys - A Free Verilog Synthesis Suite," Proceedings of Austrochip, 2013.
[3] The OpenROAD Project, https://theopenroadproject.org/ https://github.com/The-OpenROAD-Project
[4] Magic, https://github.com/RTimothyEdwards/magic
[5] Netgen, https://github.com/RTimothyEdwards/netgen
[6] OpenLANE, https://github.com/efabless/openlane
[7] GEN_AMBA, https://github.com/adki/gen_amba
[8] SKY130 PDK, https://skywater-pdk.readthedocs.io/