IP Generators - A Better Reuse Methodology

Amanjyot Kaur, Agnisys

Problem Statement

Creating reusable intellectual properties (IPs) is the dream of all designers. Who will not want to create a design once and have it used a multitude of times in the future? Each time the design is reused, the return on investment (ROI) continues to increase monotonically. Yet creating a perfectly usable IP design is challenging and resource intensive. In addition, the challenge of creating a reusable IP is not limited to just the register-transfer-level (RTL) design.

It impacts all aspects of system-on-chip (SoC) development, such as firmware, software, verification, and even documentation.

- Designers spend a huge amount of time creating standard IPs.
  - These IPs are very rigid and brittle at the same time.
- To control the flow and reuse of such IPs, engineers use a parameter-based flow,
  - but they generally break when used in a different environment because changes in a port list or customization of these IPs is difficult.
- Also, when the RTL changes, it becomes of utmost importance to vary the corresponding application programming interfaces (APIs) and sequences already created for it.

Proposed Methodology

We came up with a solution to improve the development process of IPs to create highly configurable and customizable IPs. Once the user selects the appropriate configuration settings, the IP, its verification environment, documentation and corresponding set of APIs can be generated with minimal time and effort. Instead of trying to create a reusable design, create an IP generator. We have deployed this approach for multiple SoC development teams with resounding success. While creating IP generators, there are the following focal points:

- Configuring the IPs using parameters:
  - Generate time
  - Instance/elaboration time
- Customizations required, such as additional fields or registers added to the register map (regmap) of the IP
- Creating interconnections between different IPs
- Automatic generation of configuration APIs in UVM and C format
- Creation of test sequences for different platforms like firmware, validation, verification, and Automatic Test Equipment (ATE)
- The choice of the bus used to access the IP, such as, AHB, APB, AXI, etc, is abstracted out, to avoid the design becoming too brittle

Implementation Details

We have worked to automatically generate the most common IPs used in SoCs today. These include, but are not limited to, the following:

- General Purpose Input/Output (GPIO)
- Advanced Encryption Standard (AES)
- Programmable Interrupt Controller (PIC)
- Serial Peripheral Interface (SPI)
- Pulse Width Modulation (PWM)
- Direct Memory Access (DMA)
- Inter-Integrated Circuit (I2C)
- Integrated Inter-IC Sound Bus (I2S)
- Universal Asynchronous Receiver/Transmitter (UART)
- Timer

	IP generator approach	Parameterizable IP
Flexibility	Since IPs are being created on the fly at generation time, there is additional flexibility in controlling how to create the IP	Limited flexibility since only the parameters can be used for customization
Ability to change ports	Yes	No, parameters or generics cannot change the port list in the RTL
Ability to add registers or fields to the IP's regmap	Yes	No
Impact on other aspects of IPs	Along with RTL, design verification environment, firmware, software, and documentation can also be generated	The impact of parameters is only on Verilog/SystemVerilog/VHDL code (no link to other aspects)
Development resources required	Lower	Higher

Table 1. Comparison of IP generator approach and parameterizable approach

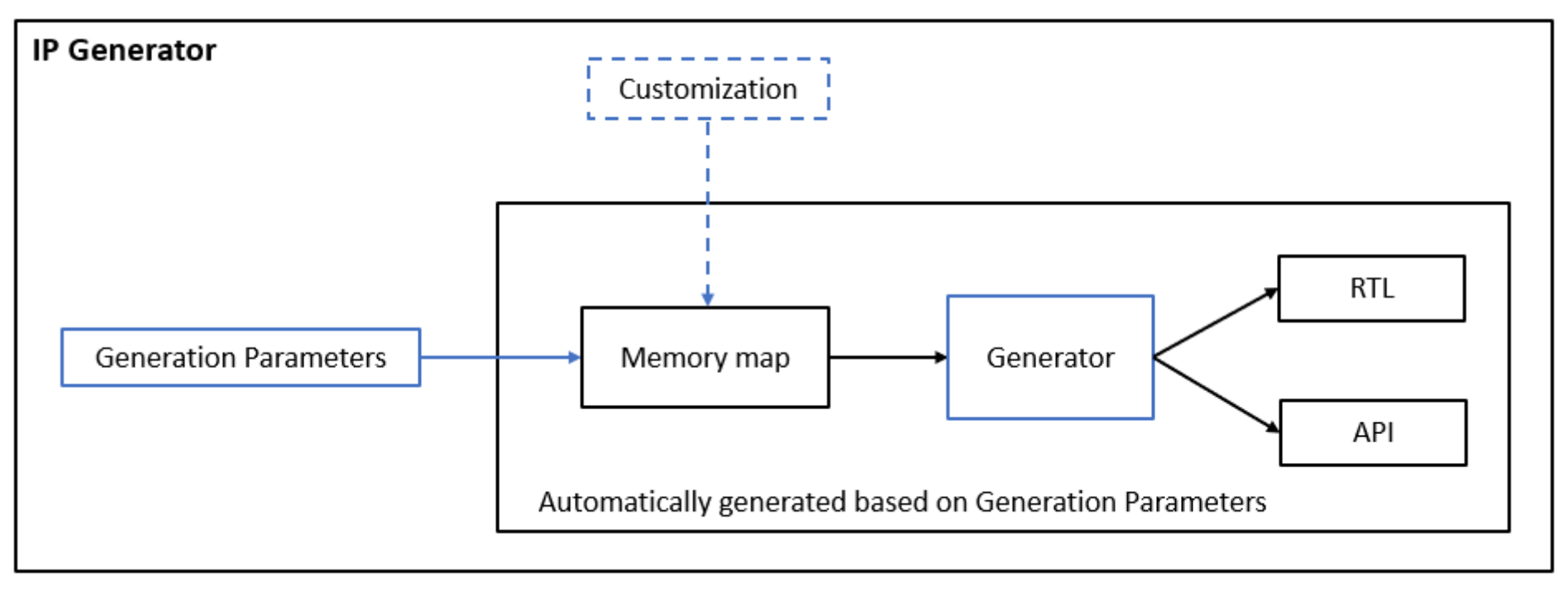

Figure 1. Block Diagram of IP Generators

NOTE: The mentioned IPs are completely verified and validated. IPs are validated on ZedBoard (ZYNC 7000 series) using Vivado v2018.3 (64-bit) and Xilinx Software Development Kit (Release Version: 2018.3).

Application - Trigger Word Detector

We have created a real-life example, Trigger Word Detector (TWD), which uses DMA, I2S, and GPIO IP. The design implements a machine learning trigger word detector algorithm to generate a prediction, based on the input spectrogram values of a voice sample that is fetched using the I2S interface. On each run, the design can detect one of the four unique words in the audio sample and then drive the appropriate LED, LED0-LED3, using the GPIO interface. If none of the words is detected, LED4 glows.Each set of weights is trained to detect four unique words. mem1_csr block is connected to a dual port memory. The memory on the other end is connected to mem2_csr by a read-only interface. On pressing the button[0] the first set of weights is loaded by the DMA into the sample regmap from the mem2_csr. The neural net logic uses these weights and input audio sample spectrogram values to generate an output which in turn drives the LEDs. Similarly, on pressing button[1] the second set of weights is loaded.

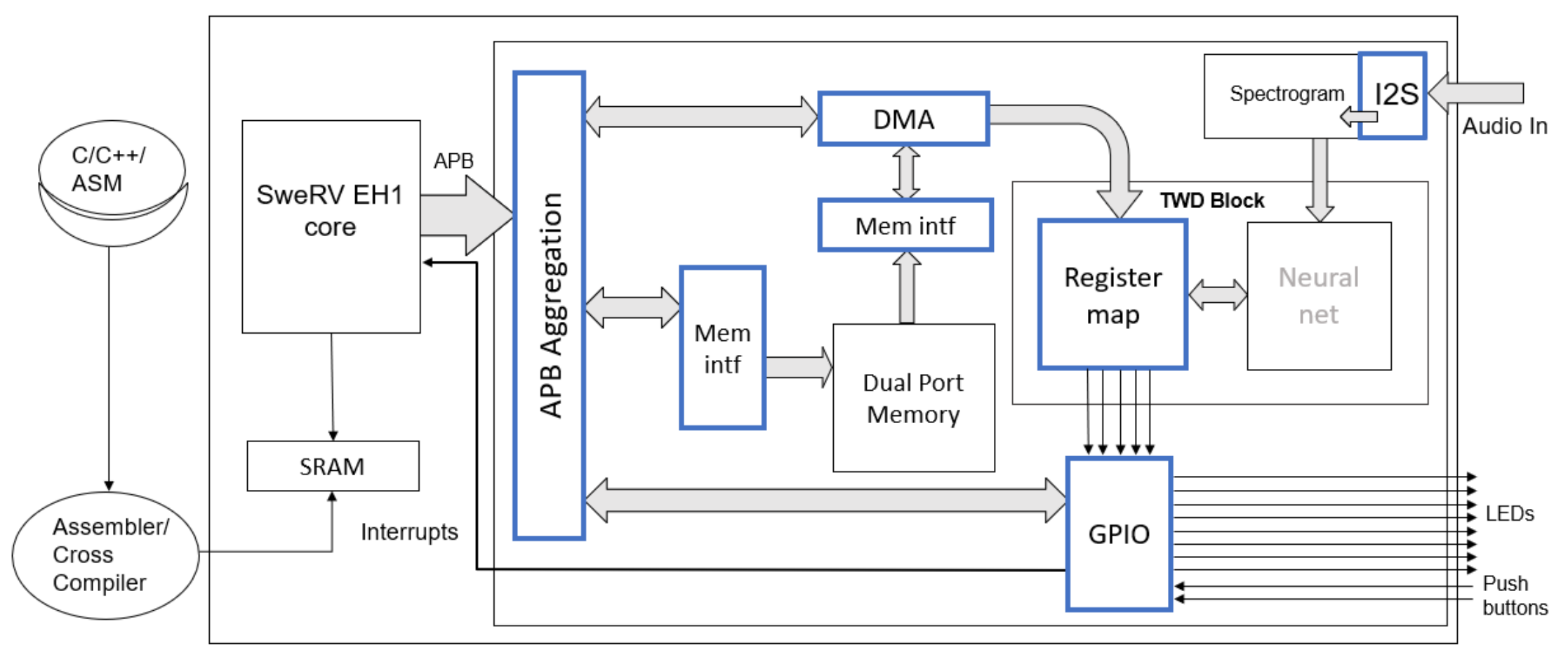

Figure 2. TWD Design Using DMA, I2S and GPIO IP

Results Table

This section shows the automatically generated aggregation logic of the top TWD block, with code for RTL, UVM, and HTML (*Figure 3,4*). It also contains sample code (*Figure 5*) for initialization configuration API of GPIO in C and UVM.

S.No.	File Name	Line of code
Configuration Bus File		
1	apb_widget.v	80
GPIO IP (Config bus : APB, Number of sources : 2, Number of outputs : 5)		
2	sync_ff.v	24
3	edge_detect.v	26
4	gpio_top.v	252
5	gpio.v	745
DMA IP (Config bus : APB, Bus type : APB, Number of channels : 2)		
6	dma_regmap_arbiter.v	227
7	dma_apb_master.v	132
8	dma_regmap_core.v	407
9	dma_regmap_txn.v	379
10	fifo.v	102
11	dma.v	742
I2S IP (Config Bus: APB, Interrupt generation with enable)		
12	i2s_master.v	310
13	prescaler.v	82
14	i2s_csr_block.v	970
15	txn_fifo.v	116
16	i2s_core.v	521

Table 2. IP related generated files (RTL) with number of lines

Figure 3. Generated Verilog and UVM for regmap

Figure 4. HTML and Aggregation Logic of TWD

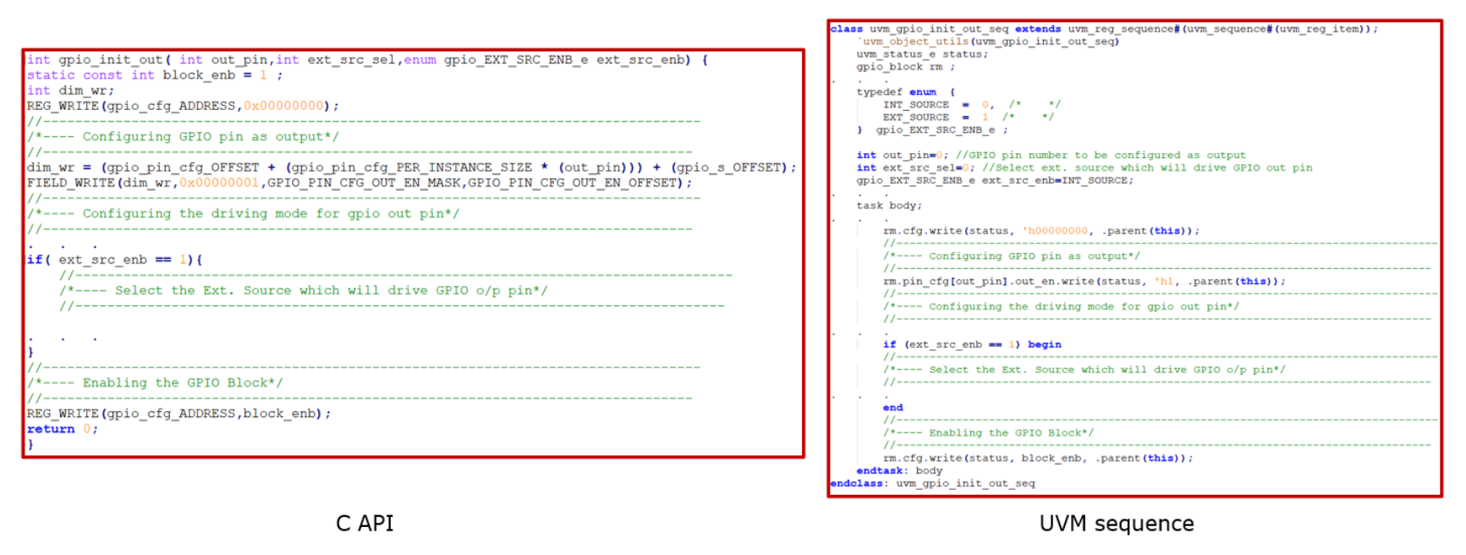

Figure 5. Sample of Configuration APIs of GPIO in C and UVM

Conclusion

In any SoC design there are certain standard IPs that are ubiquitous and are required and reused across various environments. A designer, generally, spends a huge amount of time creating such IPs from scratch. Such designs are very rigid and break when used in even slightly changed environments. The concept of generating reusable IPs and their APIs has been leveraged to help users create not just RTLs and their corresponding documentation and verification environment, but also target the firmware and software aspect of the development flow. Some of the benefits offered by approach discussed in the paper are:

- Fully configurable
- Easily customizable
- IPs are generated from the command line or on a click of a button
- Unencrypted code
- Ability to target multiple platforms (Design, Verification, Firmware, Software, and Documentation)

Using IP Generators, for about 10 different IPs, and about 50 different instantiations, users can easily generate 100,000 lines of RTL code, UVM, and C test environment with a click of a button as compared to manual efforts that would take several months.

REFERENCES

- *Philips Semiconductor:* https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf
- *Texas Instruments:* https://www.ti.com/lit/ug/sprufx4b/sprufx4b.pdf?ts=1621515564485&ref_url=https%253A%252F%252Fwww.google.com%252F
- *Intel:* https://www.intel.co.jp/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/qts_qii55003.pdf
- *Texas Instruments:* https://www.ti.com/lit/ug/spruem8a/spruem8a.pdf?ts=1628097487990&ref_url=https%253A%252F%252Fwww.google.com%252F