System Development for Verification of General Purpose Input Output

Rakshit Bhandare\textsuperscript{1}, Dr. M. S. Panse\textsuperscript{2}

\textsuperscript{1}M. Tech Scholar, \textsuperscript{2}Associate Professor
Electrical Engineering Dept., VJTI, Mumbai, India

Abstract—In SoC no. of IP block inside it depends upon specific application, increase in the IP block increases no. of digital control lines causes increase in the size of the chip. GPIO helps internal IP blocks to share digital control lines using MUX and avoids additional circuitry. Since design productivity cannot follow the pace of nanoelectronics technology innovation, it has been required to develop various design methodologies to overcome this gap. In system level design, various design methodologies such as IP reuse, automation of platform integration and verification process have been proposed. GPIO configuration register decides in which mode system has to work GPIO has four modes i.e input, output, functional, interrupt. As per operation particular mode is selected and the operation gets performed. Devices with pin scarcity like integrated circuits such as system-on-a-chip, embedded and custom hardware, and programmable logic devices cannot compromise with size can perform well without additional digital control line circuitry.

Index Terms—GPIO, SoC, Verification Plan, APB, UVM.

I. INTRODUCTION

A. Blood The verification of any design of size is a daunting task that requires successful forethought in the form of formulating, architecting, strategizing and documenting an overall verification blueprint. The value of creating such a blueprint at the start of a project has been proven out through gathered metrics of successful projects. The goal of verification planning and management is to architect an overall verification approach, and then to document that approach in a family of useful, easily extracted, maintainable verification documents that will strategically guide the overall verification effort so that the most amount of verification is accomplished in the allotted time. The aim of this course is to define terms, logically divide up the verification effort, and lay the foundation for actual verification planning and management on a real project.

B. A system on a chip or system on chip (SoC or SOC) is an integrated circuit (IC) that integrates all components of a computer or other electronic system into a single chip. It may contain digital, analog, mixed-signal, and often radio-frequency functions which all on a single chip substrate. The term system on a chip is hyperbole, indicating technical direction more than reality: a high degree of chip integration, leading toward reduced manufacturing costs, and the production of smaller systems. Many systems are too complex to fit on just one chip built with a processor optimized for just one of the system's tasks.

C. General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior including whether it is an input or output pin is controllable by the user at run time. GPIO pins have no predefined purpose, and go unused by default. The idea is that sometimes a system integrator who is building a full system might need a handful of additional digital control lines—and having these available from a chip avoids having to arrange additional circuitry to provide them.

D. The goal of hardware design is to create a device that performs a particular task, such as a DVD player, network router, or radar signal processor, based on a design specification. Your purpose as a verification engineer is to make sure the device can accomplish that task successfully—that is, the design is an accurate representation of the specification. Bugs are what you get when there is a discrepancy. The process of verification parallels the design creation process. A designer reads the hardware specification for a block, interprets the human language description, and creates the corresponding logic in a machine-readable form, usually RTL code. To do this, he or she needs to understand the input format, the transformation function, and the format of the output. There is always ambiguity in this interpretation, perhaps because of ambiguities in the original document, missing details, or conflicting descriptions. As a verification engineer, you must also read the hardware specification, create the verification plan, and then follow it to build tests showing the RTL code correctly implements the features. The verification plan is closely tied to the hardware specification and contains a description of what features need to be exercised and the techniques to be used. These steps may include directed or random testing, assertions, HW/SW co-verification, emulation, formal proofs, and use of verification IP.

E. The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol. The APB protocol
relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow. Every transfer takes at least two cycles. The APB can interface with:

- AMBA Advanced High-performance Bus (AHB)
- AMBA Advanced High-performance Bus Lite (AHB-Lite)
- AMBA Advanced Extensible Interface (AXI)
- AMBA Advanced Extensible Interface Lite (AXI4-Lite)

You can use it to access the programmable control registers of peripheral devices.

### II. METHODOLOGY

#### A. Testbench Architecture:

![Fig. 2.1 Testbench Architecture.](image)

**Packet:**

Defines the pin level activity generated by agent (to drive to DUT through the driver) or the activity has to be observed by agent (Placeholder for the activity monitored by monitor on DUT signals).

**Generator:**

Generates the stimulus (create and randomize the transaction class) and send it to Driver.

**Driver:**

Receives the stimulus (transaction) from generator and drives the packet level data inside transaction into pin level (to DUT).

**Monitor:**

Observes pin level activity on interface signals and converts into packet level which is sent to the components such as scoreboard.

**Scoreboard:**

Receives data items from monitors and compares with expected values.

**Expected values can be either golden reference values or generated from reference model.**

### Environment:

The environment is a container class for grouping higher level components like agent's and scoreboard.

**Test:**

Test is responsible for,

- Configuring the testbench.
- Initiate the testbench components construction process.
- Initiate the stimulus driving.

**Testbench_Top:**

This is the top most file, which connects the DUT and TestBench. It consists of DUT, Test and interface instances, interface connects the DUT and TestBench.

Creation of internal blocks (generator, driver, input monitor, output monitor, scoreboard) of test-bench architecture by using scripting language. The required information (pin_name, group_name, Blk_name, SFR_name) will be extracted by perl script language. create bock wise output which are .csv files which further helps to create .sv files (generator, driver, input monitor, output monitor, scoreboard) of internal blocks of test-bench. Combined .sv files created using perl scripting are used by test-case file which runs on cadence software and result will be displayed on verdi simulator.

### III. GENERAL PURPOSE INPUT OUTPUT (GPIO)

#### A. General-purpose input/output (GPIO):

It is a generic pin on an integrated circuit or computer board whose behavior including whether it is an input or output pin is controllable by the user at run time. GPIO pins have no predefined purpose, and go unused by default. The idea is that sometimes a system integrator who is building a
full system might need a handful of additional digital control lines—and having these available from a chip avoids having to arrange additional circuitry to provide them.

GPIO capabilities may include:

- GPIO pins can be configured to be input or output.
- GPIO pins can be enabled/disabled.
- Input values are readable (typically high=1, low=0).
- Output values are writable/readable.
- Input values can often be used as IRQs (typically for wakeup events).

GPIO peripherals vary widely. In some cases, they are simple a group of pins that can switch as a group to either input or output. In others, each pin can be set up to accept or source different logic voltages, with configurable drive strengths and pull ups/downs. Input and output voltages are typically though not always limited to the supply voltage of the device with the GPIOs, and may be damaged by greater voltages.

Figure 4.2 shows the operating scenario of GPIO. To use chip-top pins as general input/output or functional input/output, GPIO mux control registers should be configured at first. If GPIOs are used to general input, corresponding bis of GPXn_DAT is the pin state. Also, when the GPIOs are configured as general output, the pin state is the same as the corresponding bit of GPXn_DAT. If GPIOs are used to functional input or output, it connect chip-top pins to internal IP ports. For example, GPXn_CON register has configured to I2C, two chip-top pins are connected to SCL and SDA of internal I2C controller ports. If GPIOs are used to get interrupts, the configurations for the interrupt polarity and the sampling scheme should be selected. Interrupt mask (NWEINT_GPXn_MASK) for each GPIO can be programmable. The interrupt status can be checked by reading the interrupt pending registers.

Data is taken from the input datasheet which contains pin names, SFRs, Group name of the GPIO and the condition on which functional mode get selected for required operation.
V. TOOLS

Verification environment is needed for a new design, or for a design revision with significant changes, it is important to objectively look at the shortcomings of the existing verification environment and expected productivity gain with the new methodology and determine the best solution.

In our case we need to find an optimum balance between re-usability of our legacy Verilog environment and the resource utilization along with limited timelines in adopting the new methodology. This can be accomplished by reusing the knowledge /legacy code from an earlier project along with an upgrade to a new methodology provided with the verification language, that is

A. SystemVerilog.

The feature-set of SystemVerilog can be divided into two distinct roles:

1. SystemVerilog for RTL design is an extension of Verilog-2005; all features of that language are available in SystemVerilog.

2. SystemVerilog for verification uses extensive object-oriented programming techniques and is more closely related to Java than Verilog.

B. Universal Verification Methodology (UVM)

It is a standardized methodology for verifying integrated circuit designs. UVM is derived mainly from the OVM (Open Verification Methodology) which was, to a large part, based on the eRM (e Reuse Methodology) for the e Verification Language developed by Verisity Design in 2001. The UVM class library brings much automation to the SystemVerilog language such as sequences and data automation features (packing, copy, compare) etc., and unlike the previous methodologies developed independently by the simulator vendors, is an Accellera standard with support from multiple vendors: Aldec, Cadence, Mentor, and Synopsys.

C. scripting

script language is a programming language that supports scripts, programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator.[1] Scripting languages are often interpreted (rather than compiled). Primitives are usually the elementary tasks or API calls, and the language allows them to be combined into more complex programs. Environments that can be automated through scripting include software applications, web pages within a web browser, the shells of operating systems (OS), embedded systems, as well as numerous games. A scripting language can be viewed as a domain-specific language for a particular environment; in the case of scripting an application, this is also known as an extension language. Scripting languages are also sometimes referred to as very high-level programming languages, as they operate at a high level of abstraction, or as control languages, particularly for job control languages on mainframes.

The term "scripting language" is also used loosely to refer to dynamic high-level general-purpose languages, such as Perl, Tcl, and Python, with the term "script" often used for small programs (up to a few thousand lines of code) in such languages, or in domain-specific languages such as the text-processing languages sed and AWK. Some of these languages were originally developed for use within a particular environment, and later developed into portable domain-specific or general-purpose languages. Conversely, many general-purpose languages have dialects that are used as scripting languages. This article discusses scripting languages in the narrow sense of languages for a specific environment.

Embedded software plays a critical role in driving today’s SoC verification at various stages. With growing SoC complexity and evolving processor architecture, it has become increasingly difficult to perform SoC debug activities. What is needed is a simultaneous view of both hardware and software to help designers efficiently and effectively advance their debug productivity. A sophisticated waveform viewer that can display and analyze activities in time series Fast Signal Database (FSDB) A high-performance waveform comparison engine that identifies discrepancies between files Source code / browser that facilitates coordination between source code and hierarchical browser Flexible circuit diagram and block diagram that can display logic and connection relationship Intuitive state transition diagram that can grasp the operation of the state machine

VI. RESULTS
Mapping of required source and destination from the respective functional mode from the input datasheet is done using Scripting Language i.e. PERL. The script extracts required source and destination for respective peripherals which automates the result.

Input to the script can be .csv file or .xls file and generates the results also in .csv or xls format.

VII. CONCLUSION

The System-on-Chip (SoC) industry has developed rapidly over the last fifteen years from producing VLSI devices that integrated a processor and a few memory and peripheral components onto a single chip to today’s high-performance SoCs that incorporate hundreds of IP blocks. This progress is a consequence of Moore’s Law (which enables ever-higher levels of integration) and of market economics (where consumers demand ever-more functionality in smaller, lower-cost products with better battery life).

System on a Chip (SoC) integrated circuits make extensive use of general-purpose I/O (GPIO) pins. GPIO controllers provide a variety of functions for peripheral devices, including interrupts, input signaling, and output signaling. A feature of some GPIO controllers is the ability to generate interrupts on both edges of a signal (rising, or Active-High edges, and falling, or Active-Low edges). GPIO helps internal IP blocks to share digital control lines using MUX and avoids additional circuitry. Devices with pin scarcity like integrated circuits such as system-on-a-chip, embedded and custom hardware, and programmable logic devices cannot compromise with size can perform well without additional digital control line circuitry.

REFERENCES


