System Development for Verification of General Purpose Input Output

Rakshit Bhandare[1], Dr. M. S. Panse[2]

[1]M. Tech Scholar, [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 get 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:

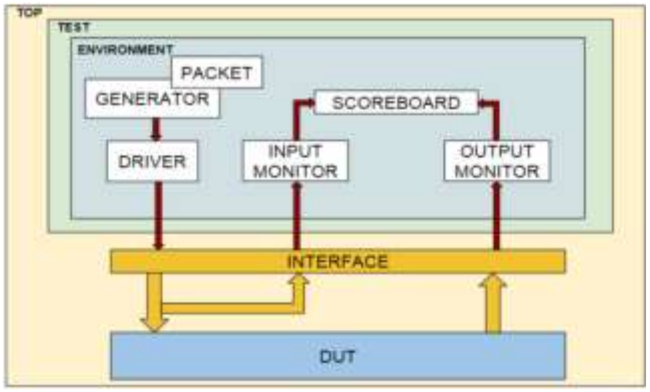


Fig. 2.1 Testbench Architecture.

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.

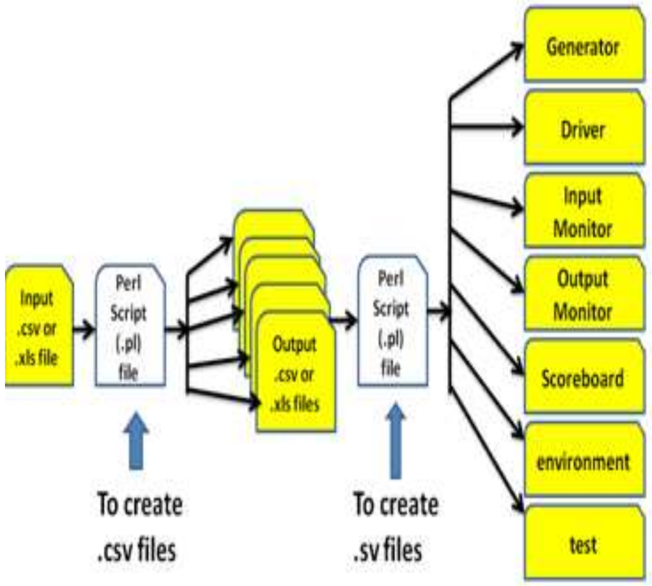


Fig. 2.2 Design flow of creation of system.

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.

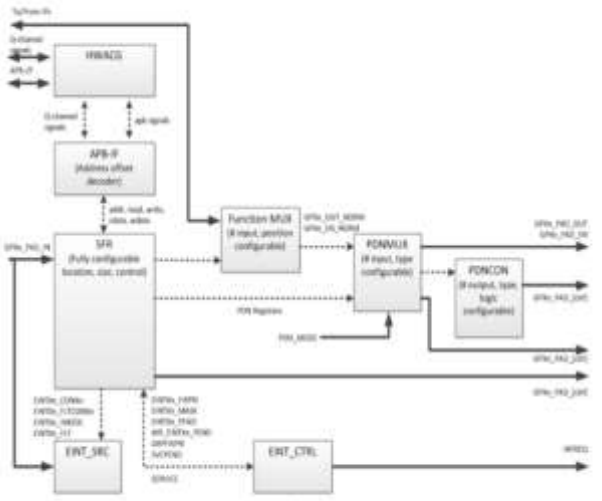

HWACG
APB-IF (Address offset decoder)
SFR (Fully configurable location, size, control)
Function MUX (# input, position configurable)
PDNMUX (# input, type configurable)
PDNCON (# output, type, logic configurable)
PDN_MODE
EINT_SRC
EINT_CTRL

Fig. 3.1 GPIO Block Diagram.

B. GPIO operation:

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.

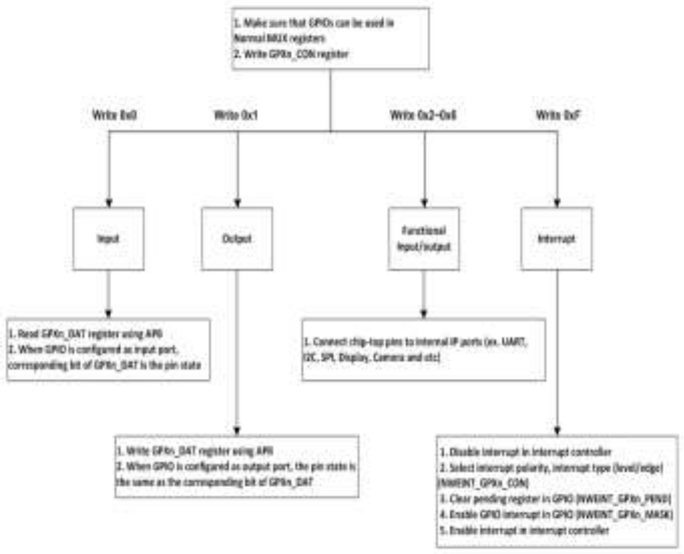
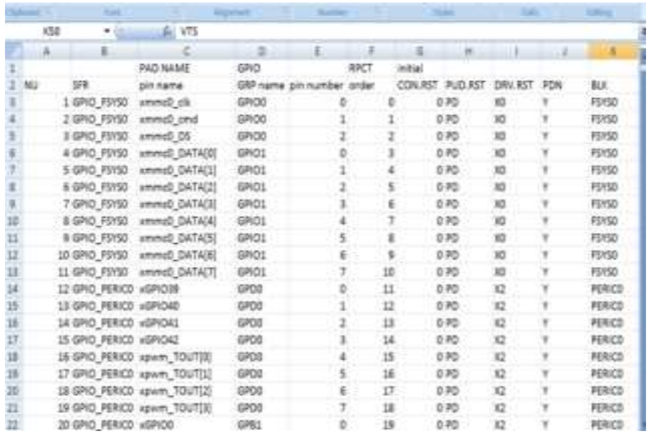

1. Make sure that GPIOs can be used in Normal MUX registers
2. Write GPXn_CON register
Write 0x0
Write 0x1
Write 0x2~0x8
Write 0xF
Input
Output
Functional Input/output
Interrupt
1. Read GPXn_DAT register using API
2. When GPIO is configured as input port, corresponding bit of GPXn_DAT is the pin state
1. Connect chip-top pins to internal IP ports (ex. UART, I2C, SPI, Display, Camera and etc)
1. Write GPXn_DAT register using API
2. When GPIO is configured as output port, the pin state is the same as the corresponding bit of GPXn_DAT
1. Disable interrupt in interrupt controller
2. Select interrupt polarity, interrupt type (level/edge) (NWEINT_GPXn_CON)
3. Clear pending register in GPIO (NWEINT_GPXn_PEND)
4. Enable GPIO interrupt in GPIO (NWEINT_GPXn_MASK)
5. Enable interrupt in interrupt controller

Fig. 3.2 GPIO Operation.

IV. DATA COLLECTION

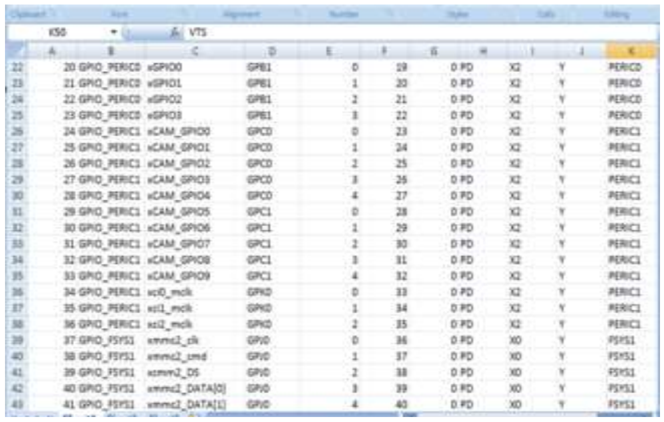
Data is taken from the input datasheet which contains pin names, SFRs, Group name of the GPIO and the condition on which functional mode get selected for required operation.



K58 VTS

	A	B	C	D	E	F	G	H	I	J	K
1			PAD NAME	GPIO		RPCT	initial				
2	NU	SFR	pin name	GRP name	pin number	order	CON.RST	PUD.RST	DRV.RST	PDN	BLK
3	1	GPIO_FSYS0	xmmc0_clk	GPIO0	0	0	0	PD	X0	Y	FSYS0
4	2	GPIO_FSYS0	xmmc0_cmd	GPIO0	1	1	0	PD	X0	Y	FSYS0
5	3	GPIO_FSYS0	xmmc0_DS	GPIO0	2	2	0	PD	X0	Y	FSYS0
6	4	GPIO_FSYS0	xmmc0_DATA[0]	GPIO1	0	3	0	PD	X0	Y	FSYS0
7	5	GPIO_FSYS0	xmmc0_DATA[1]	GPIO1	1	4	0	PD	X0	Y	FSYS0
8	6	GPIO_FSYS0	xmmc0_DATA[2]	GPIO1	2	5	0	PD	X0	Y	FSYS0
9	7	GPIO_FSYS0	xmmc0_DATA[3]	GPIO1	3	6	0	PD	X0	Y	FSYS0
10	8	GPIO_FSYS0	xmmc0_DATA[4]	GPIO1	4	7	0	PD	X0	Y	FSYS0
11	9	GPIO_FSYS0	xmmc0_DATA[5]	GPIO1	5	8	0	PD	X0	Y	FSYS0
12	10	GPIO_FSYS0	xmmc0_DATA[6]	GPIO1	6	9	0	PD	X0	Y	FSYS0
13	11	GPIO_FSYS0	xmmc0_DATA[7]	GPIO1	7	10	0	PD	X0	Y	FSYS0
14	12	GPIO_PERIC0	xGPIO39	GPD0	0	11	0	PD	X2	Y	PERIC0
15	13	GPIO_PERIC0	xGPIO40	GPD0	1	12	0	PD	X2	Y	PERIC0
16	14	GPIO_PERIC0	xGPIO41	GPD0	2	13	0	PD	X2	Y	PERIC0
17	15	GPIO_PERIC0	xGPIO42	GPD0	3	14	0	PD	X2	Y	PERIC0
18	16	GPIO_PERIC0	xpwm_TOUT[0]	GPD0	4	15	0	PD	X2	Y	PERIC0
19	17	GPIO_PERIC0	xpwm_TOUT[1]	GPD0	5	16	0	PD	X2	Y	PERIC0
20	18	GPIO_PERIC0	xpwm_TOUT[2]	GPD0	6	17	0	PD	X2	Y	PERIC0
21	19	GPIO_PERIC0	xpwm_TOUT[3]	GPD0	7	18	0	PD	X2	Y	PERIC0
22	20	GPIO_PERIC0	xGPIO0	GPB1	0	19	0	PD	X2	Y	PERIC0

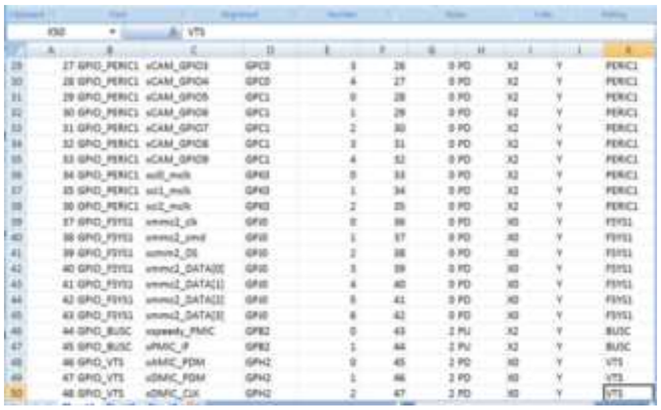
Fig. 4.1 Input Datasheet.



K58 VTS

	A	B	C	D	E	F	G	H	I	J	K
22	20	GPIO_PERIC0	xGPIO0	GPB1	0	19	0	PD	X2	Y	PERIC0
23	21	GPIO_PERIC0	xGPIO1	GPB1	1	20	0	PD	X2	Y	PERIC0
24	22	GPIO_PERIC0	xGPIO2	GPB1	2	21	0	PD	X2	Y	PERIC0
25	23	GPIO_PERIC0	xGPIO3	GPB1	3	22	0	PD	X2	Y	PERIC0
26	24	GPIO_PERIC1	xCAM_GPIO0	GPC0	0	23	0	PD	X2	Y	PERIC1
27	25	GPIO_PERIC1	xCAM_GPIO1	GPC0	1	24	0	PD	X2	Y	PERIC1
28	26	GPIO_PERIC1	xCAM_GPIO2	GPC0	2	25	0	PD	X2	Y	PERIC1
29	27	GPIO_PERIC1	xCAM_GPIO3	GPC0	3	26	0	PD	X2	Y	PERIC1
30	28	GPIO_PERIC1	xCAM_GPIO4	GPC0	4	27	0	PD	X2	Y	PERIC1
31	29	GPIO_PERIC1	xCAM_GPIO5	GPC1	0	28	0	PD	X2	Y	PERIC1
32	30	GPIO_PERIC1	xCAM_GPIO6	GPC1	1	29	0	PD	X2	Y	PERIC1
33	31	GPIO_PERIC1	xCAM_GPIO7	GPC1	2	30	0	PD	X2	Y	PERIC1
34	32	GPIO_PERIC1	xCAM_GPIO8	GPC1	3	31	0	PD	X2	Y	PERIC1
35	33	GPIO_PERIC1	xCAM_GPIO9	GPC1	4	32	0	PD	X2	Y	PERIC1
36	34	GPIO_PERIC1	xci0_mclk	GPK0	0	33	0	PD	X2	Y	PERIC1
37	35	GPIO_PERIC1	xci1_mclk	GPK0	1	34	0	PD	X2	Y	PERIC1
38	36	GPIO_PERIC1	xci2_mclk	GPK0	2	35	0	PD	X2	Y	PERIC1
39	37	GPIO_FSYS1	xmmc2_clk	GPI0	0	36	0	PD	X0	Y	FSYS1
40	38	GPIO_FSYS1	xmmc2_cmd	GPI0	1	37	0	PD	X0	Y	FSYS1
41	39	GPIO_FSYS1	xmmc2_DS	GPI0	2	38	0	PD	X0	Y	FSYS1
42	40	GPIO_FSYS1	xmmc2_DATA[0]	GPI0	3	39	0	PD	X0	Y	FSYS1
43	41	GPIO_FSYS1	xmmc2_DATA[1]	GPI0	4	40	0	PD	X0	Y	FSYS1

Fig. 4.2 Input Datasheet.



K50 VTS

	A	B	C	D	E	F	G	H	I	J	K
29	27	GPIO_PERIC1	xCAM_GPIO3	GPC0	3	26	0	PD	X2	Y	PERIC1
30	28	GPIO_PERIC1	xCAM_GPIO4	GPC0	4	27	0	PD	X2	Y	PERIC1
31	29	GPIO_PERIC1	xCAM_GPIO5	GPC1	0	28	0	PD	X2	Y	PERIC1
32	30	GPIO_PERIC1	xCAM_GPIO6	GPC1	1	29	0	PD	X2	Y	PERIC1
33	31	GPIO_PERIC1	xCAM_GPIO7	GPC1	2	30	0	PD	X2	Y	PERIC1
34	32	GPIO_PERIC1	xCAM_GPIO8	GPC1	3	31	0	PD	X2	Y	PERIC1
35	33	GPIO_PERIC1	xCAM_GPIO9	GPC1	4	32	0	PD	X2	Y	PERIC1
36	34	GPIO_PERIC1	scl0_mclk	GPK0	0	33	0	PD	X2	Y	PERIC1
37	35	GPIO_PERIC1	scl1_mclk	GPK0	1	34	0	PD	X2	Y	PERIC1
38	36	GPIO_PERIC1	scl2_mclk	GPK0	2	35	0	PD	X2	Y	PERIC1
39	37	GPIO_FSYS1	xmmc2_clk	GPI0	0	36	0	PD	X0	Y	FSYS1
40	38	GPIO_FSYS1	xmmc2_cmd	GPI0	1	37	0	PD	X0	Y	FSYS1
41	39	GPIO_FSYS1	xmmc2_DS	GPI0	2	38	0	PD	X0	Y	FSYS1
42	40	GPIO_FSYS1	xmmc2_DATA[0]	GPI0	3	39	0	PD	X0	Y	FSYS1
43	41	GPIO_FSYS1	xmmc2_DATA[1]	GPI0	4	40	0	PD	X0	Y	FSYS1
44	42	GPIO_FSYS1	xmmc2_DATA[2]	GPI0	5	41	0	PD	X0	Y	FSYS1
45	43	GPIO_FSYS1	xmmc2_DATA[3]	GPI0	6	42	0	PD	X0	Y	FSYS1
46	44	GPIO_BUSC	xspeedy_PMIC	GPB2	0	43	2	PU	X2	Y	BUSC
47	45	GPIO_BUSC	xPMIC_IF	GPB2	1	44	2	PU	X2	Y	BUSC
48	46	GPIO_VTS	xAMIC_PDM	GPH2	0	45	2	PD	X0	Y	VTS
49	47	GPIO_VTS	xDMIC_PDM	GPH2	1	46	2	PD	X0	Y	VTS
50	48	GPIO_VTS	xDMIC_CLK	GPH2	2	47	2	PD	X0	Y	VTS

Fig. 4.3 Input Datasheet.

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 using familiar symbols Intuitive state transition diagram that can grasp the operation of the state machine

VI. RESULTS

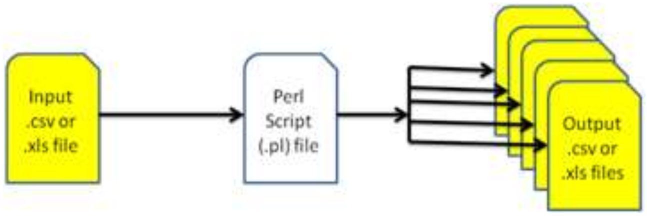
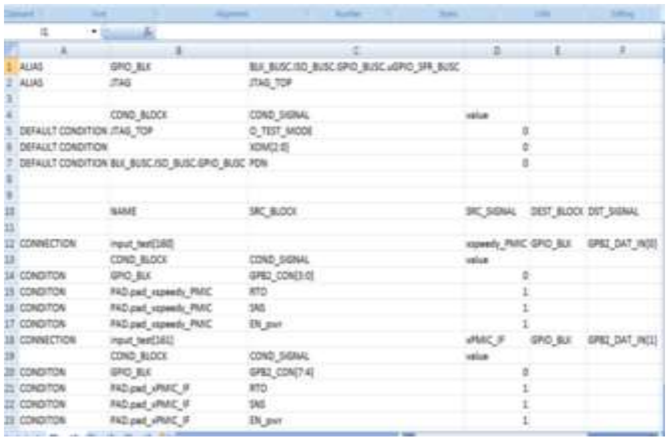


Fig. 6.1 Implementation of scripting.

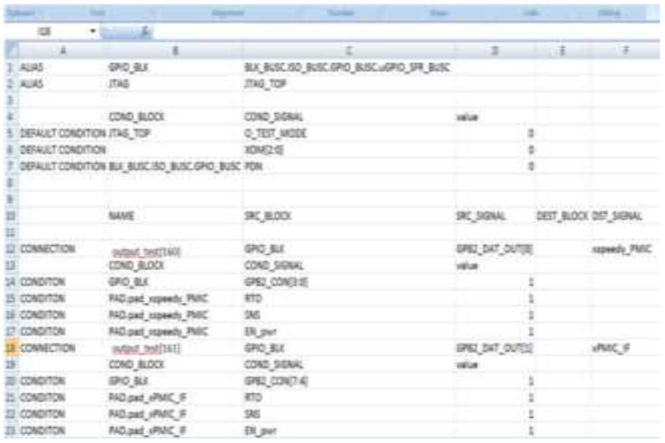
Mapping of required source and destination from the respective functional mode from the input Datasheet is done using Scripting Language i.e. PERL. The script extracts required source and destination for respective peripherals which automates the result.

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



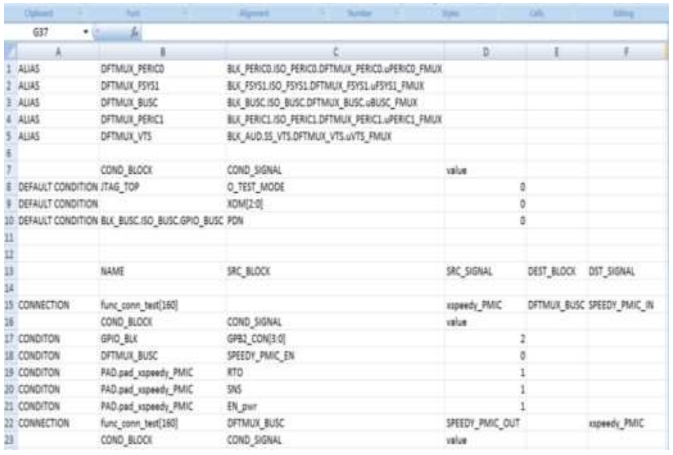
	A	B	C	D	E	F
1	ALIAS	GPIO_BLK	BLK_BUSC.ISO_BUSC.GPIO_BUSC.uGPIO_SFR_BUSC			
2	ALIAS	JTAG	JTAG_TOP			
3						
4		COND_BLOCK	COND_SIGNAL	value		
5	DEFAULT CONDITION	JTAG_TOP	O_TEST_MODE	0		
6	DEFAULT CONDITION		XOM[2:0]	0		
7	DEFAULT CONDITION	BLK_BUSC.ISO_BUSC.GPIO_BUSC	PDN	0		
8						
9						
10		NAME	SRC_BLOCK	SRC_SIGNAL	DEST_BLOCK	DST_SIGNAL
11						
12	CONNECTION	input_test[160]		xspeedy_PMIC	GPIO_BLK	GPB2_DAT_IN[0]
13		COND_BLOCK	COND_SIGNAL	value		
14	CONDITON	GPIO_BLK	GPB2_CON[3:0]	0		
15	CONDITON	PAD.pad_xspeedy_PMIC	RTO	1		
16	CONDITON	PAD.pad_xspeedy_PMIC	SNS	1		
17	CONDITON	PAD.pad_xspeedy_PMIC	EN_pwr	1		
18	CONNECTION	input_test[161]		xPMIC_IF	GPIO_BLK	GPB2_DAT_IN[1]
19		COND_BLOCK	COND_SIGNAL	value		
20	CONDITON	GPIO_BLK	GPB2_CON[7:4]	0		
21	CONDITON	PAD.pad_xPMIC_IF	RTO	1		
22	CONDITON	PAD.pad_xPMIC_IF	SNS	1		
23	CONDITON	PAD.pad_xPMIC_IF	EN_pwr	1		

Fig. 6.2 Result of Input Mode.



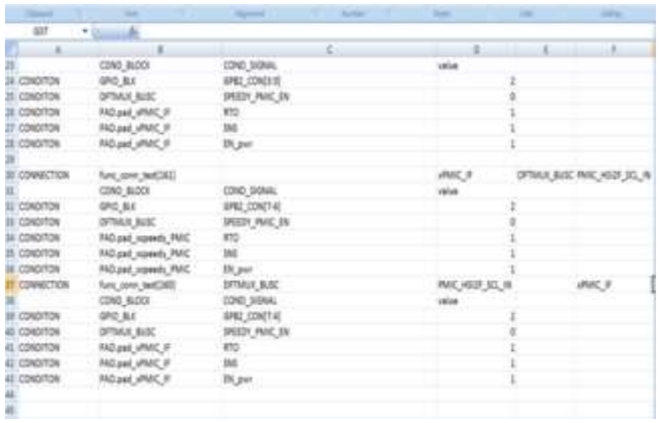
	A	B	C	D	E	F
1	ALIAS	GPIO_BLK	BLK_BUSC.ISO_BUSC.GPIO_BUSC.uGPIO_SFR_BUSC			
2	ALIAS	JTAG	JTAG_TOP			
3						
4		COND_BLOCK	COND_SIGNAL	value		
5	DEFAULT CONDITION	JTAG_TOP	O_TEST_MODE	0		
6	DEFAULT CONDITION		XOM[2:0]	0		
7	DEFAULT CONDITION	BLK_BUSC.ISO_BUSC.GPIO_BUSC	PDN	0		
8						
9						
10		NAME	SRC_BLOCK	SRC_SIGNAL	DEST_BLOCK	DST_SIGNAL
11						
12	CONNECTION	output_test[160]	GPIO_BLK	GPB2_DAT_OUT[0]		xspeedy_PMIC
13		COND_BLOCK	COND_SIGNAL	value		
14	CONDITON	GPIO_BLK	GPB2_CON[3:0]	1		
15	CONDITON	PAD.pad_xspeedy_PMIC	RTO	1		
16	CONDITON	PAD.pad_xspeedy_PMIC	SNS	1		
17	CONDITON	PAD.pad_xspeedy_PMIC	EN_pwr	1		
18	CONNECTION	output_test[161]	GPIO_BLK	GPB2_DAT_OUT[1]		xPMIC_IF
19		COND_BLOCK	COND_SIGNAL	value		
20	CONDITON	GPIO_BLK	GPB2_CON[7:4]	1		
21	CONDITON	PAD.pad_xPMIC_IF	RTO	1		
22	CONDITON	PAD.pad_xPMIC_IF	SNS	1		
23	CONDITON	PAD.pad_xPMIC_IF	EN_pwr	1		

Fig. 6.3 Result of Output Mode.



	A	B	C	D	E	F
1	ALIAS	DFTMUX_PERIC0	BLK_PERIC0.ISO_PERIC0.DFTMUX_PERIC0.uPERIC0_FMUX			
2	ALIAS	DFTMUX_FSYS1	BLK_FSYS1.ISO_FSYS1.DFTMUX_FSYS1.uFSYS1_FMUX			
3	ALIAS	DFTMUX_BUSC	BLK_BUSC.ISO_BUSC.DFTMUX_BUSC.uBUSC_FMUX			
4	ALIAS	DFTMUX_PERIC1	BLK_PERIC1.ISO_PERIC1.DFTMUX_PERIC1.uPERIC1_FMUX			
5	ALIAS	DFTMUX_VTS	BLK_AUD.SS_VTS.DFTMUX_VTS.uVTS_FMUX			
6						
7		COND_BLOCK	COND_SIGNAL	value		
8	DEFAULT CONDITION	JTAG_TOP	O_TEST_MODE	0		
9	DEFAULT CONDITION		XOM[2:0]	0		
10	DEFAULT CONDITION	BLK_BUSC.ISO_BUSC.GPIO_BUSC	PDN	0		
11						
12						
13		NAME	SRC_BLOCK	SRC_SIGNAL	DEST_BLOCK	DST_SIGNAL
14						
15	CONNECTION	func_conn_test[160]		xspeedy_PMIC	DFTMUX_BUSC	SPEEDY_PMIC_IN
16		COND_BLOCK	COND_SIGNAL	value		
17	CONDITON	GPIO_BLK	GPB2_CON[3:0]	2		
18	CONDITON	DFTMUX_BUSC	SPEEDY_PMIC_EN	0		
19	CONDITON	PAD.pad_xspeedy_PMIC	RTO	1		
20	CONDITON	PAD.pad_xspeedy_PMIC	SNS	1		
21	CONDITON	PAD.pad_xspeedy_PMIC	EN_pwr	1		
22	CONNECTION	func_conn_test[160]	DFTMUX_BUSC	SPEEDY_PMIC_OUT		xspeedy_PMIC
23		COND_BLOCK	COND_SIGNAL	value		

Fig. 6.4 Result of Functional Mode.



	A	B	C	D	E	F
23		COND_BLOCK	COND_SIGNAL	value		
24	CONDITON	GPIO_BLK	GPB2_CON[3:0]	2		
25	CONDITON	DFTMUX_BUSC	SPEEDY_PMIC_EN	0		
26	CONDITON	PAD.pad_xPMIC_IF	RTO	1		
27	CONDITON	PAD.pad_xPMIC_IF	SNS	1		
28	CONDITON	PAD.pad_xPMIC_IF	EN_pwr	1		
29						
30	CONNECTION	func_conn_test[161]		xPMIC_IF	DFTMUX_BUSC	PMIC_HSI2F_SCL_IN
31		COND_BLOCK	COND_SIGNAL	value		
32	CONDITON	GPIO_BLK	GPB2_CON[7:4]	2		
33	CONDITON	DFTMUX_BUSC	SPEEDY_PMIC_EN	0		
34	CONDITON	PAD.pad_xspeedy_PMIC	RTO	1		
35	CONDITON	PAD.pad_xspeedy_PMIC	SNS	1		
36	CONDITON	PAD.pad_xspeedy_PMIC	EN_pwr	1		
37	CONNECTION	func_conn_test[160]	DFTMUX_BUSC	PMIC_HSI2F_SCL_IN		xPMIC_IF
38		COND_BLOCK	COND_SIGNAL	value		
39	CONDITON	GPIO_BLK	GPB2_CON[7:4]	2		
40	CONDITON	DFTMUX_BUSC	SPEEDY_PMIC_EN	0		
41	CONDITON	PAD.pad_xPMIC_IF	RTO	1		
42	CONDITON	PAD.pad_xPMIC_IF	SNS	1		
43	CONDITON	PAD.pad_xPMIC_IF	EN_pwr	1		
44						
45						

Fig. 6.5 Result of Functional Mode.

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

[1] Chris Spear, Greg Tumbush,"SystemVerilog for Verification: A Guide to Learning the Testbench Language Features",3rd edition,Springer Science & Business Media, 2012.

[2] Sik Kim, Kwang-Hyun Cho, Byeong Min "An efficient GPIO block design methodology using formalized SFR description", SoC Design Conference (ISOCC), 2011 International.

[3] D. Gajski, "Essential Issues for IP Reuse", Proceedings of ASP-DAC, pp. 37-42.

[4] C. K. Lennard, "Industrially Proving the SPIRIT Consortium Specifications for Design Chain Integration", Proceedings of DATE 2006, pp. 1-6.

[5] K. Cho, "Reusable Platform Design Methodology For SOC Integration And Verification", Proceedings of ISOCC 2008, pp. 1-78.

[6] 6W. Kruijtzer, "Industrial IP integration flows based on IP-XACT standards", Proceedings of DATE 2008, pp. 32-37.

[7] S. Sarkar, S. Chanclar G, S. Shinde “Effective IP reuse for high quality SOC design”, SOC Conference, 2005. Proceedings. IEEE International.

[8] M. Strik, "Subsystem Exchange in a Concurrent Design Process Environment", Proceedings of DATE 2008, pp. 953-958.

[9] V. Guide, "SystemVerilog TestBench", Verificationguide.com, 2016. [Online]. Available: http://www.verificationguide.com/p/writing-systemverilog-verification.html. [Accessed: 19- Sep- 2016].

[10] "Typical Verification Flow", Asic-world.com, 2016. [Online]. Available: http://www.asic-world.com/tidbits/typical_verification.html. [Accessed: 19- Sep- 2016].

[11] "Basic UVM | Universal Verification Methodology | Verification Academy", Verificationacademy.com, 2016. [Online]. Available: https://verificationacademy.com/courses/basic-uvm. [Accessed: 19- Sep- 2016].

[12] 12. "UVM (Universal Verification Methodology)", Accellera.org, 2016. [Online]. Available: http://www.accellera.org/activities/vip. [Accessed: 19- Sep- 2016].

[13] "Teach Yourself Perl 5 in 21 days - Table of Contents", Wwwacs.gantep.edu.tr, 2016. [Online]. Available: http://wwwacs.gantep.edu.tr/docs/perl-ebook/. [Accessed: 03- Dec- 2016].

[14] "Perl tutorial", Perl Maven, 2016. [Online]. Available: https://perlmaven.com/perl-tutorial. [Accessed: 03- Dec- 2016].

[15] R. Schwartz, T. Phoenix and B. Foy, “Learning Perl”, 1st ed. Sebastopol, CA: O'Reilly, 2005.

[16] "The Perl Programming Language - www.perl.org", Perl.org, 2016. [Online]. Available: https://www.perl.org/. [Accessed: 03- Dec- 2016].