2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

Pushbutton Complete IP Generation

Freddy Nunez

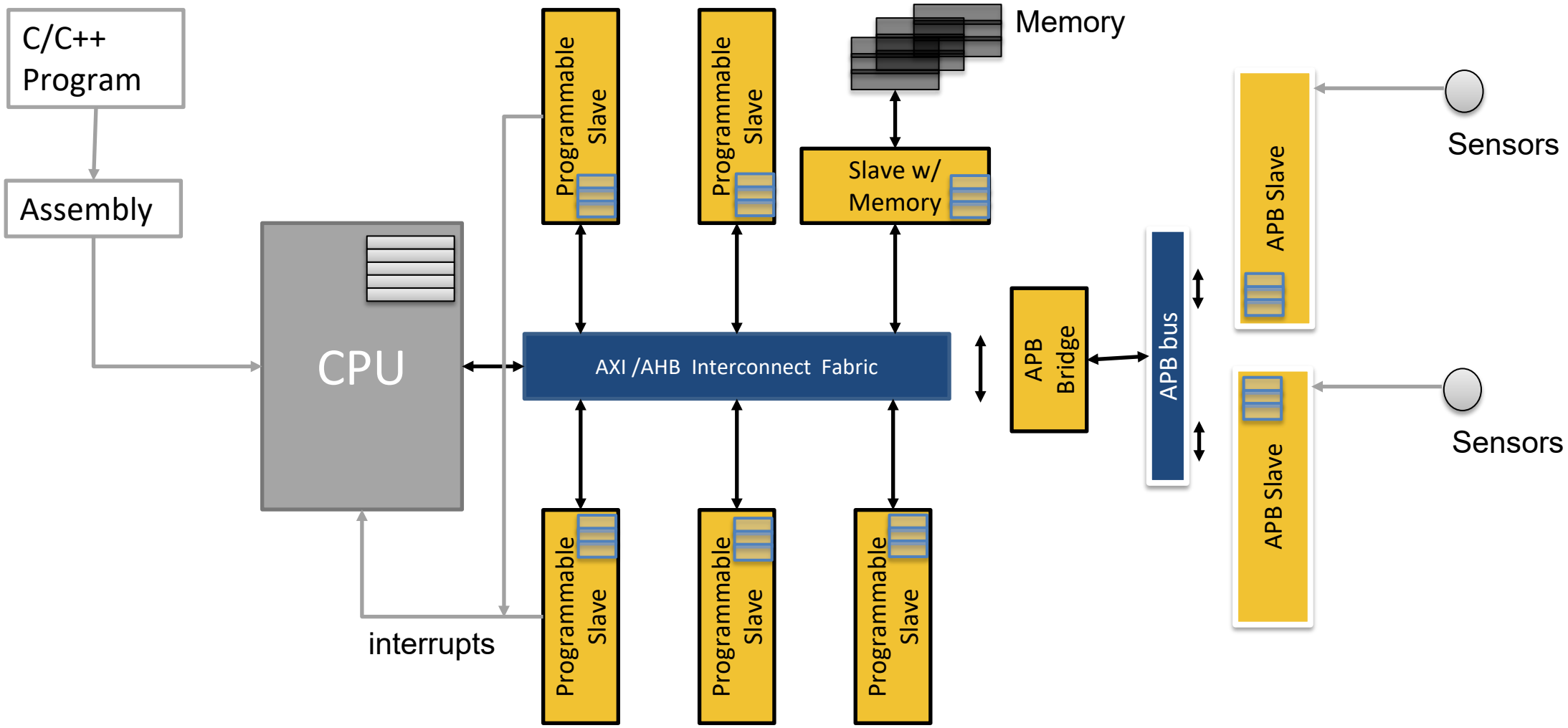
AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

accellera
SYSTEMS INITIATIVE

Typical Chip Design

- Hardware of the SoC is designed by HW team
  - But used by
    - Verification/Emulation team
    - Firmware team
    - Validation team
    - Software team
- How does the software interact with the IPs?
  - Through the Hardware Software Interface (HSI)
- Hardware is at the core and software API is around it
- Device drivers (part of the HSI) are tedious to create
  - They are written in C and Assembly

Introduction to a Typical SoC



Challenges Faced

- Design challenges
  - Too much data
  - Even small changes in data causes havoc
  - Significant source of bugs
  - Reusing IP

- Verification/Validation challenges
  - Duplication of work across teams
  - Rise in complexity of designs
  - Inability to create same debug environment for multiple platforms
  - Mismatch in specification and implementation

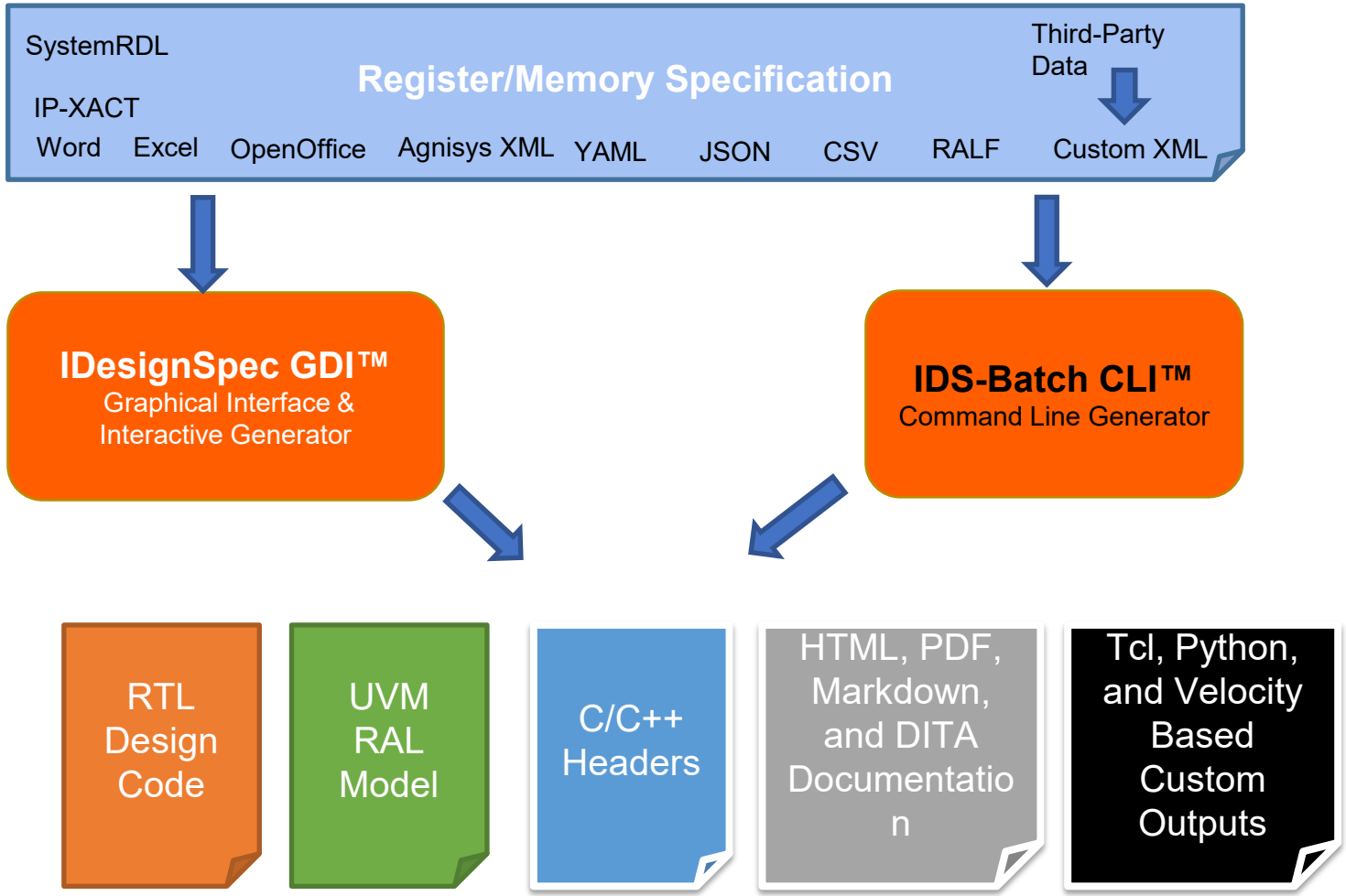
Challenges Faced - Cont'd

- SOC design companies
  - Increasing demands of design complexity and design performance
  - Combining automation with flexibility to accommodate changes in sub-systems across applications
  - Driving down the cost of design for a better ROI
  - Shrinking market windows
  - Boosting productivity of design teams to meet shorter market windows

An Ideal Solution

- Ease of generation
- Generated code should not be encrypted
- Should provide appropriate error messages
- Ability to reuse IPs
  - Customizing the designs
  - Configuring the designs
- Easy mechanism for generating IP blocks
- Ability to handle different bus protocols
- Handling metastability of multi clock domain designs
- Design must be functionally safe and secure

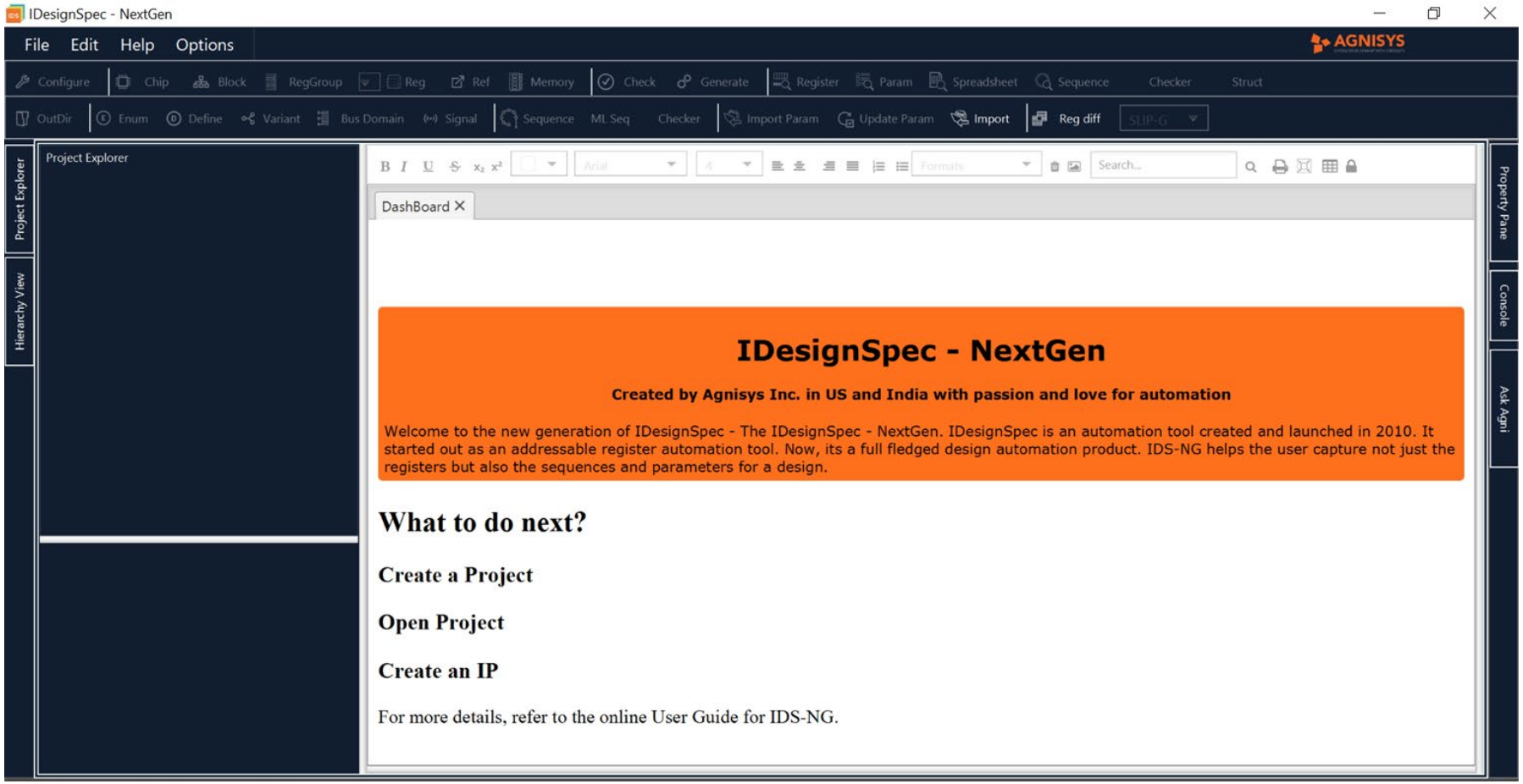
IDesignSpec ǂ GDI & CLI Data Flow



Generated Bus interfaces

AMBA APB
AMBA AHB
AMBA AHB-Lite
AMBA AXI-Lite
AMBA AXI4 Full
AMBA AXI5-Lite
TileLink
Avalon
Wishbone

Designing IPs



Designing IPs - Cont'd

- Addressable Register specification

Define user registers' settings

Register view:

IDS Register Specification

	reg_name		32	address default
Properties..				
Description..				

bits	name	s/w	h/w	default	description
31:0	pkt32	rw	ro		a 32 bit packet field.

Spreadsheet view:

IDS Register Specification

Open IDS Template

	A	B	C	D	E	F	G	H	I	J
1	chip	block	register	width	field	sw access	hw access	field default	bits	description
2			reg_name	32						
3					pkt32	rw	ro		31:0	a 32 bit packet field.

C Header,
Misra-C,
C Tests,
....
SystemRDL,
Register Sequences...

Designing IPs in IDS-NG - Cont'd

- Sample specification

Sequence View

DashBoard | ethernet_ip_core.idsng* ✕

Sequences

sequence name	ip	description
seq_name1	ethernet_ip_core.idsng	

arguments
arg1

constants
const1

variables
var1

command
write
write

Param View

DashBoard | ethernet_ip_core.idsng ✕

Address	Register Name	bit 31	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit	bit
	MODER																															
	INT_SOURCE																															
0x8	INT_MASK																															
0xC	INT_FIFO																															
0x10	USER_Reg																															
0x14	REG_FILE																															
0x14	IPGT																															
0x18	IPGR1																															
0x1C	IPGR2																															
	end section																															
0x5C	REG_ARR																															
0x5C	PACKETLEN																															
	end section																															
0x1000	MEM																															
0x1000	COLLCONF																															
	end section																															
0x2000	TX_BD_NUM																															
0x2004	CTRLMODER																															
0x2008	MIIMODER																															

Spreadsheet View

DashBoard | ethernet_ip_core.idsng ✕

IDS Register Specification

Open IDS Template

	A	B	C	D	E	F	G	H	I	J
1	**enum name**	**enum value**	**enum desc**	**mnemonic name**						
2	INT_Source_enum	0x0	re							
3	INT_Source_enum	0x1	re							
4	INT_Source_enum	0x2	tra							
5	INT_Source_enum	0x3	tra							
6										
7	**define name**	**define value**	de de							
8	Enable1	1	En 1							
9	Disable1	0	Di 0							
10										
11	**chip**	**block**	re							
12	Ethernet									
13		Eth_block1								

Register View

DashBoard | ethernet_ip_core.idsng ✕

IDS Register Specification

1	Ethernet			address|0x0
define name	value	description		private
Enable1	1	Enable equal to 1		0
Disable1	0	Disable equal to 0		0

	INT_Source_enum	
mnemonic name	value	description
RXC_dis	0x0	receive enable
TXC_dis	0x1	receive disable
TXC_enb	0x2	transmit enable
RXC_enb	0x3	transmit disable

variant name	value	description
altera	1	For Altera target
xilinx	1	For Xilinx target

1.1	Eth_block1			address|0x0

1.1.1	MODER			32	address| 0x0 default | 0x0000A000
bits	name	s/w	h/w	default	description
16	RECSMALL	rw	ro	0	{0:ignore_small_pkt; 1:accept_small_pkt; } Receive Small Packets 0 = Packets smaller than MINFL are ignored. 1 = Packets smaller than MINFL are accepted. Enum defined for this Field in its description. Also , Enum name can be written in field's Default value.
15	PAD	rw	ro	1	{0:dont_pad_short_frame; 1:pad_short_frame; }Padding enabled 0 = Do not add pads to short frames. 1 = Add pads to short frames (until the minimum frame length is equal to MINFL).

Designing IPs Cont'd

- Addressable Register configuration

IDesignSpec - NextGen

Configuration Settings sig.idsng

General
Outputs
User-Defined Outputs
Settings
Variant
Formating
DataSheet
CustomCSV
Advance Verification
Advance Validation
Advance Design
Sequences
Outputs
UVM
C
CSV
System Verilog
Matlab
Checker
Settings

Outputs

Output Directory: idsng [Browse]

RTL: Verilog, 1995, 2001, RTL Wire, System C, alt1, alt2, VHDL, alt1, alt2, Multi Out File, System Verilog

Verification: UVM, Multi Out File, OVM, VMM, ARV-Sim, ARV-Formal, IVS-Excel, eRM

Headers: C, alt1, alt2, MISRAC, Multi Out File, SV Header, Perl, Python, C++, VHeader, CSharp, VHD Header

Documentation: HTML, alt1, alt2, alt3, 2D Reg, Mem Dump, YAML, CustomCSV, SVG, PDF, XML, Word, JSON

Standard: IP-XACT, V1.5(IEEE 1685-2009), V1.4, System RDL, alt1, alt2, CMSIS-SVD

Select All | Clear All

-slipg

Import | Export | OK | Cancel

IDesignSpec - NextGen

Configuration Settings sig.idsng

General
Outputs
User-Defined Outputs
Settings
Variant
Formating
DataSheet
CustomCSV
Advance Verification
Advance Validation
Advance Design
Sequences
Outputs
UVM
C
CSV
System Verilog
Matlab
Checker
Settings

Settings

Address Units (Bits): 8, 16, 32, 64, 128, 256

Reg Width: 8, 16, 32, 64, 128, 256

Bus Width: 8, 16, 32, 64, 128, 256

Bus: AMBA-AHB, AVALON, PROPRIETARY, AMBA-APB, AMBA3-AHB-lite, AMBA-AXI, AMBA-AXI4FULL, OCP, WISHBONE, SPI-beta, I2C-beta

Block Size

Chip Size

Board Size

C Type: BigEndian, LittleEndian

Template Directory: C:\Program Files (x86)\Agnisys\IDSNextgen\ids_templates\datasheet [Browse]

Memory dump file path [Browse]

Quality Checks

-slipg

Import | Export | OK | Cancel

Designing IPs - Cont'd

- Generated sample code

```
module sig_ids#(
    parameter bus_width = 32,
    parameter addr_width = 2,
    parameter block_size = 'h4,
    parameter [addr_width-1 : 0] block_offset = {(addr_width){1'b0}}
    )
    (
    output reg_name_enb,
    input [32- 1 : 0] reg_name_fld_in,
    input reg_name_fld_in_enb,
    output [31 : 0] reg_name_fld_r,
    input aclk,
    input aresetn,
    input [addr_width-1 : 0] awaddr,
    input awvalid,
    output awready,
    input [2 : 0] awprot,
    input [bus_width-1 : 0] wdata,
    input wvalid,
    output wready,
    input [bus_width/8 - 1 : 0] wstrb,
    output [1 : 0] bresp,
    input bready,
    output bvalid,
    input [addr_width-1 : 0] araddr,
    input arvalid,
    output arready,
    input [2 : 0] arprot,
    output [bus_width-1 : 0] rdata,
    output rvalid,
    input rready,
    output [1 : 0] rresp
    );
.   .   .
    axi_widget # (.addr_width(addr_width), .bus_width(bus_width) )axi (
        .   .   .
    );
    .   .   .
    assign wr_slave_select = ((slvwaddr[addr_width - 1 : 0]  >= block_offset) && (slvwaddr[addr_width - 1 : 0]
    <= block_offset + block_size -1)) ? 1'b1 : 1'b0;
    assign rd_slave_select = ((slvraddr[addr_width - 1 : 0]  >= block_offset) && (slvraddr[addr_width - 1 : 0]
    <= block_offset + block_size -1)) ? 1'b1 : 1'b0;
endmodule
```

RTL

```
/*-----------------------------------------------------------------------
Class       : ethernet_ip_block
DESCRIPTION:-
-----------------------------------------------------------------------*/
`ifndef CLASS_ethernet_ip_block
`define CLASS_ethernet_ip_block
class ethernet_ip_block extends uvm_reg_block;
    `uvm_object_utils(ethernet_ip_block)

    rand ethernet_ip_tx_pkt tx_pkt;
    rand ethernet_ip_rx_pkt rx_pkt;

    // Function : new
    function new(string name = "ethernet_ip_block");
        super.new(name, UVM_NO_COVERAGE);
    endfunction

    // Function : build
    virtual function void build();
        //define default map and add reg/regfiles
        default_map= create_map("default_map", 'h0, 4, UVM_BIG_ENDIAN, 1);

        //TX_PKT
        tx_pkt = ethernet_ip_tx_pkt::type_id::create("tx_pkt");
        tx_pkt.configure(this, null, "tx_pkt");
        tx_pkt.build();
        default_map.add_reg( tx_pkt, 'h0, "RW");

        //RX_PKT
        rx_pkt = ethernet_ip_rx_pkt::type_id::create("rx_pkt");
        rx_pkt.configure(this, null, "rx_pkt");
        rx_pkt.build();
        default_map.add_reg( rx_pkt, 'h4, "RW");

        lock_model();
    endfunction

endclass
`endif
```

UVM

Designing IPs - Cont'd

- Generated sample code

```c
#ifndef _ETHERNET_IP_REGS_H_
#define _ETHERNET_IP_REGS_H_
typedef union {
    struct {
        hwint start : 1;          /* 31 SW=rw HW=rw 0x0 */
        hwint pkt : 29;          /* 30:2 SW=rw HW=rw 0x0 */
        hwint parity : 2;          /* 1:0 SW=rw HW=rw 0x0 */
    } bf;
    hwint  dw;
} ethernet_ip_tx_pkt;
typedef union {
    struct {
        hwint start : 1;          /* 31 SW=rw HW=rw 0x0 */
        hwint pkt : 29;          /* 30:2 SW=rw HW=rw 0x0 */
        hwint parity : 2;          /* 1:0 SW=rw HW=rw 0x0 */
    } bf;
    hwint  dw;
} ethernet_ip_rx_pkt;
typedef struct {
    ethernet_ip_tx_pkt  tx_pkt;
    ethernet_ip_rx_pkt  rx_pkt;
} ethernet_ip_s;
#define ethernet_ip_s_SIZE 0x8
#define ethernet_ip_tx_pkt_SIZE 0x4
#define ethernet_ip_rx_pkt_SIZE 0x4
#define ethernet_ip_s_OFFSET 0x0
#define ethernet_ip_tx_pkt_OFFSET 0x0
#define ethernet_ip_rx_pkt_OFFSET 0x4
#define ethernet_ip_s_ADDRESS 0x0
#define ethernet_ip_tx_pkt_ADDRESS 0x0
#define ethernet_ip_rx_pkt_ADDRESS 0x4
#define ETHERNET_IP_TX_PKT_START_OFFSET 31
#define ETHERNET_IP_TX_PKT_START_MASK 0x80000000
#define ETHERNET_IP_TX_PKT_START_INV_MASK 0x7FFFFFFF
#define ETHERNET_IP_TX_PKT_START_VALUE_MASK 0x40000000
#define ETHERNET_IP_TX_PKT_START_INV_VALUE_MASK 0xBFFFFFFF
#define ETHERNET_IP_TX_PKT_START_SIZE 1
#define ETHERNET_IP_TX_PKT_START_DEFAULT 0
.    .    .
#endif /* _ETHERNET_IP_REGS_H_ */
/* end */
```

CHeader

Block : ethernet_ip

Table of Content			
S.No.	Names	Default	Address
1.1	reg : tx_pkt	0x00000000	0x0
1.2	reg : rx_pkt	0x00000000	0x4

1 : Block : ethernet_ip 0x0

Description:

Reg :
tx_pkt
rx_pkt

1.1 : Reg : tx_pkt 0x0

Description:

1.1 : Reg : tx_pkt																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

[31] start
[1:0] parity
[30:2] pkt

Bits	Field name	sw	hw	default	Description
1:0	parity	rw	rw	0x0	
30:2	pkt	rw	rw	0x0	
31	start	rw	rw	0x0	

1.2 : Reg : rx_pkt 0x4

Description:

1.2 : Reg : rx_pkt																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

[31] start
[1:0] parity
[30:2] pkt

Bits	Field name	sw	hw	default	Description
1:0	parity	rw	rw	0x0	
30:2	pkt	rw	rw	0x0	
31	start	rw	rw	0x0	

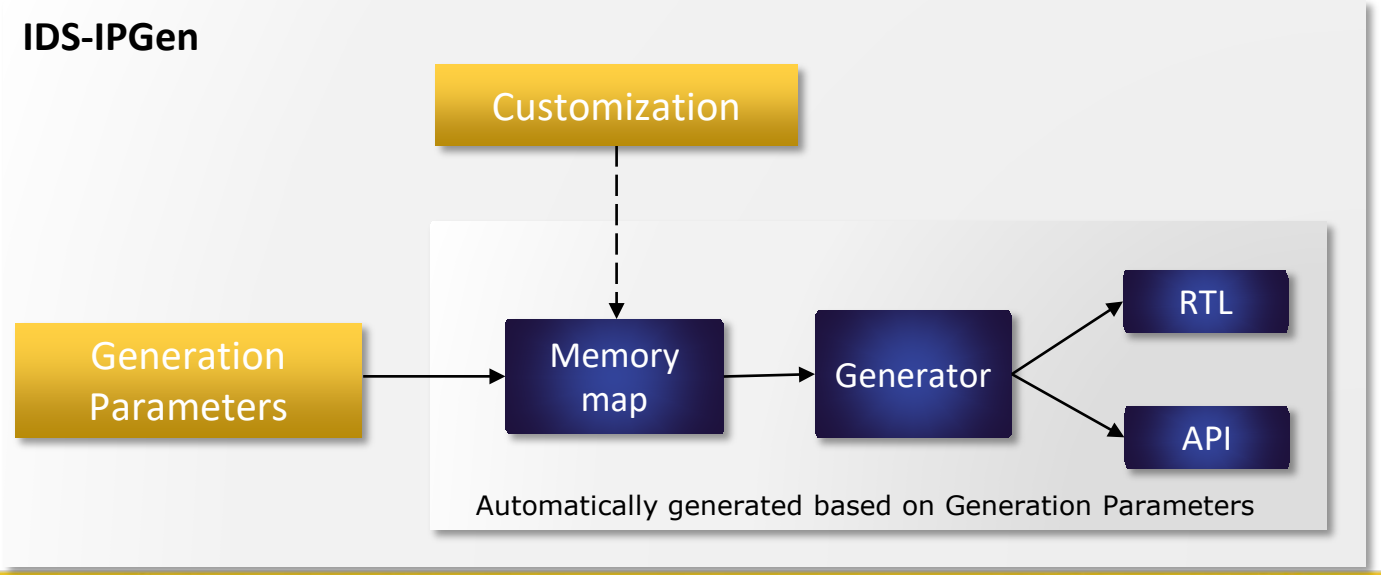
(top of block)

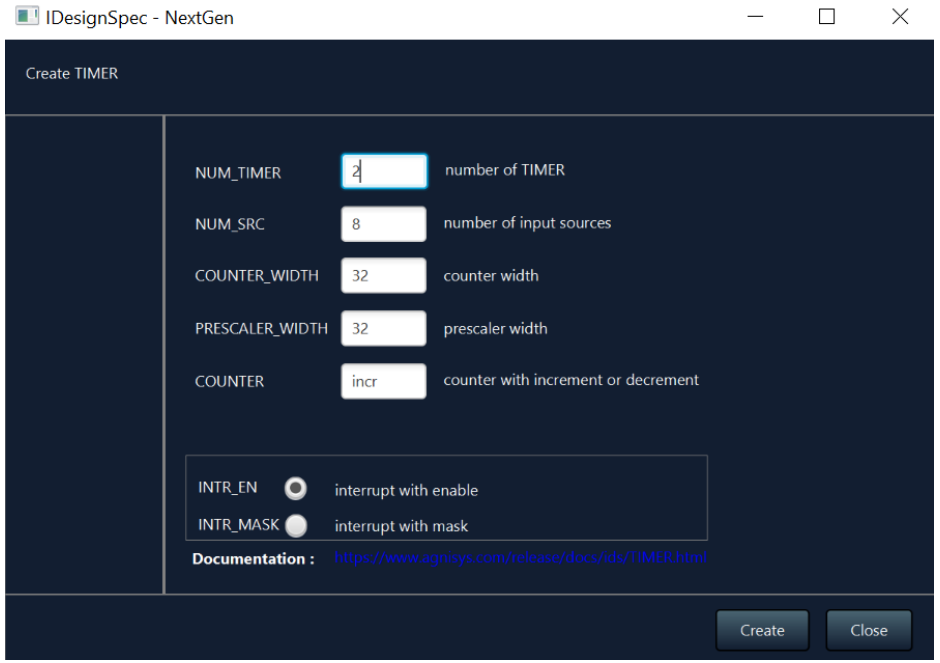
Agnisys, In ***** Copyright 2018 All Rights Reserved. *****

HTML

Auto Generating Standard IPs

- IDS-IPGen can also be used to automatically generate standard IPs (fully verified and validated) and their APIs, also provides add-in functionality of configurability and customizability





Auto Generating Standard IPs - Cont'd

- Register specification - Automatically generated by setting generation parameters

timer		address

intr.irq_per_channel=true;

Description..

define name	value	description	private
$NUM_TIMER	2		1
$NUM_SRC	8		1
$COUNTER_WIDTH	32		1
$PRESCALER_WIDTH	32		1

timer_signals

Properties..

Description..

name	port type	description
status_reset[1:0]	input	
result_reset[1:0]	input	
counter_reset[1:0]	input	
counter_start[1:0]	input	

status — 32 — address / default

rtl.reg_enb=false;resetsignal=status_reset%d:0:sync:low;count=2;

bits	name	s/w	h/w	default	description
0	overflow	r/w1c	wo	0	Interrupt status for overflow{intr.status=overflow;}
1	run_intr	r/w1c	wo	0	Interrupt status for running mode{intr.status=running;}
2	period_intr	r/w1c	wo	0	Interrupt status for periodic mode{intr.status=periodic;}
3	cfg_ch	r/w1c	wo	0	Interrupt status for configuration change{intr.status=cfg;}

control — 32 — address / default

count=2;

Description..

bits	name	s/w	h/w	default	description
0	en	rw	ro	0	1:- Enables the timer block or calculations 0:- Disables the timer block
2:1	mode	rw	ro	0	00: running mode 01: Periodic mode with source enable 00: periodic mode without source 11: reserved
5:3	event_sel	rw	ro	0	Legal values for running mode 000: high level 001: low level 010: between two high edges 011: between two low edges Legal values for the Periodic Mode 100: Posedges 101: negedges 110: both edges
8:6	Src_sel	rw	ro	0	To select the source on which measurement will be performed. These act as a counter enable signals for a given timer. If no source width is defined, then the timer will count the clock edges depending upon the prescaling value.

period — 32 — address / default

count=2;

Description..

bits	name	s/w	h/w	default	description
31:0	F1	rw	ro	0	Provides the threshold value

enable — 32 — address / default

rtl.reg_enb=false;count=2;

bits	name	s/w	h/w	default	description
0	Over_intr_en	rw	na	0	Interrupt enable for overflow{intr.enable=overflow;}
1	Run_intr_en	rw	na	0	Interrupt enable for running mode{intr.enable=running;}
2	Period_intr_en	rw	na	0	Interrupt enable for periodic mode{intr.enable=periodic;}
3	cfg	rw	na	0	Interrupt enable for configuration change{intr.enable=cfg;}

result — 32 — address / default

rtl.reg_enb=false;resetsignal=result_reset%d:0:sync:low;count=2;

bits	name	s/w	h/w	default	description
31:0	data	ro	wo	0	Stores the counting information depending upon the selected modes.

prescaler — 32 — address / default

count=2;

Description..

bits	name	s/w	h/w	default	description
31:0	F1	rw	ro	0	Defines the prescaling values

counter — 32 — address / default

rtl.reg_enb=false;count=2;

bits	name	s/w	h/w	default	description
31:0	F1	rw	ro	0	{counter=incr;resetsignal=counter_reset%d:0:sync:high,counter_start:1:sync:high;}

Auto Generating Standard IPs - Cont'd

- Generated sample code

```
module  timer_top #(

    parameter bus_width = 32,
    parameter addr_width = 6,
    parameter timer_offset = 'h0,
    parameter timer_count = 2
    )
    (
    input  clk,
    input  reset,
    input  [7:0]src,
    output  [timer_count-1:0]irq_tmr,
    .   .
    );
    .   .   .
    reg  [timer_count-1:0] cfg_f;
    wire  [timer_count-1:0] irq_wire;
    timer_ids #(.bus_width(bus_width),.addr_width(addr_width),.block_offset(timer_offset))
    regmap(
    .   .   .
    .status_reset(control_en_r),
    .result_reset(control_en_r)
    );
    generate
        genvar tmr_cnt;
        for(tmr_cnt =0; tmr_cnt < timer_count; tmr_cnt = tmr_cnt + 1) begin  : timer_cnt
        timer_core #(.bus_width(bus_width),.addr_width(addr_width)) core(
        .clk(clk),
        .src(src),
        .reset(reset),
        .timer_enb(control_en_r[tmr_cnt]),
        .   .   .
        .pre_clk(clk_en[tmr_cnt])
        );
        .   .   .
        assign  irq_tmr[tmr_cnt] = irq_wire[tmr_cnt];
    end
endgenerate
endmodule
```

RTL

```
#ifndef IDS_TIMER
    #define IDS_TIMER
    #include "sig.h"
#endif

#include "timer_reset_mode_iss.h"

int timer_reset_mode( int timer_sel) {
unsigned int consolidated_temp_value = 0;
static const int reset_val = 0 ;
int dim_wr;
dim_wr = (timer_control_OFFSET + (timer_control_PER_INSTANCE_SIZE * (
timer_sel))) + (timer_s_OFFSET);
REG_WRITE(dim_wr,reset_val);
dim_wr = 0;
dim_wr = (timer_period_OFFSET + (timer_period_PER_INSTANCE_SIZE * (timer_sel
))) + (timer_s_OFFSET);
REG_WRITE(dim_wr,reset_val);
dim_wr = 0;
dim_wr = (timer_prescaler_OFFSET + (timer_prescaler_PER_INSTANCE_SIZE * (
timer_sel))) + (timer_s_OFFSET);
REG_WRITE(dim_wr,reset_val);
dim_wr = 0;
dim_wr = (timer_counter_OFFSET + (timer_counter_PER_INSTANCE_SIZE * (
timer_sel))) + (timer_s_OFFSET);
REG_WRITE(dim_wr,reset_val);
dim_wr = 0;
return 0;
}
```

C sequence

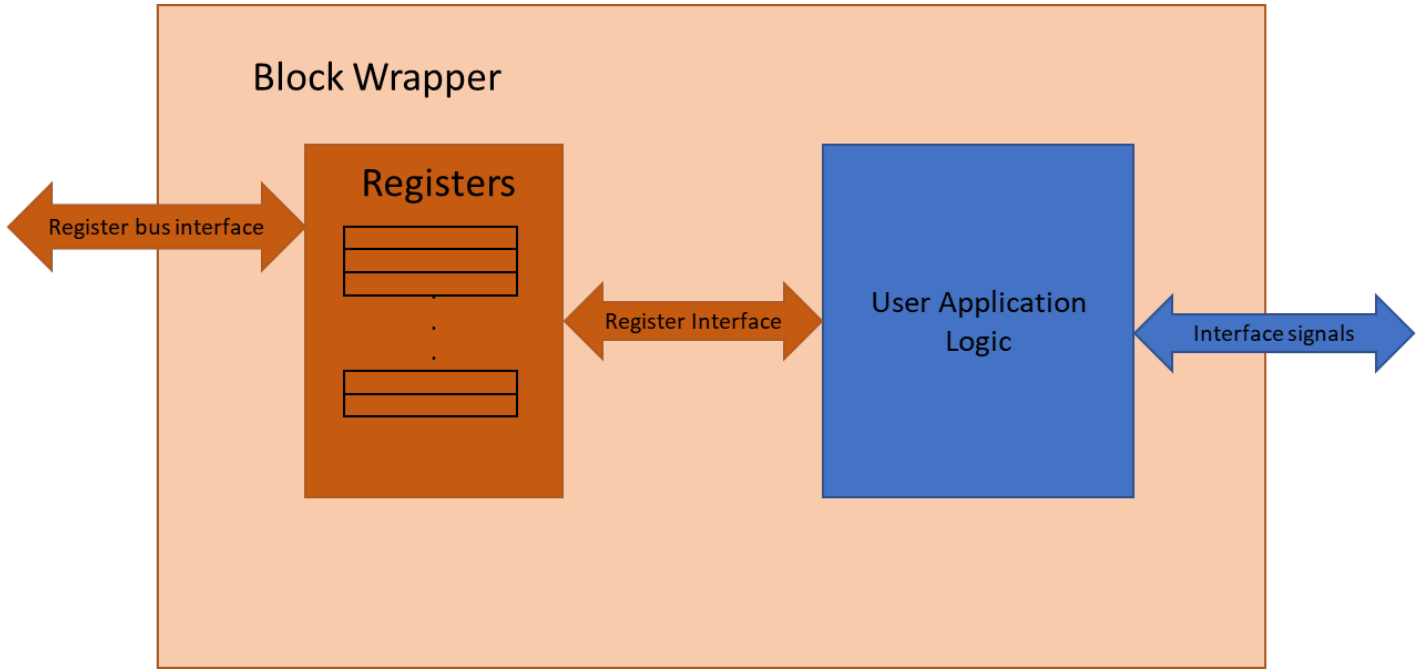
What's Complete-IP ?

- Generating the RTL for the register IP/specification for the addressable registers

- Once a register specification is captured and its RTL generated -
  - A synthesizable application logic layer is required to interact with the addressable registers
  - The intended functionality and configuring of the RTL registers in done using this user logic

- The pushbutton Complete-IP helps in capturing this design functionality (User Application Logic) by using simple templates, which will help in the overall "completeness" of an IP

Complete-IP Overview Diagram



- *The orange box depicts the addressable registers for the IP*
- *The blue box depicts the user application logic which the user creates manually. The aim is to automate this application logic creation with the help of predefined templates*

User Application Logic Constructs

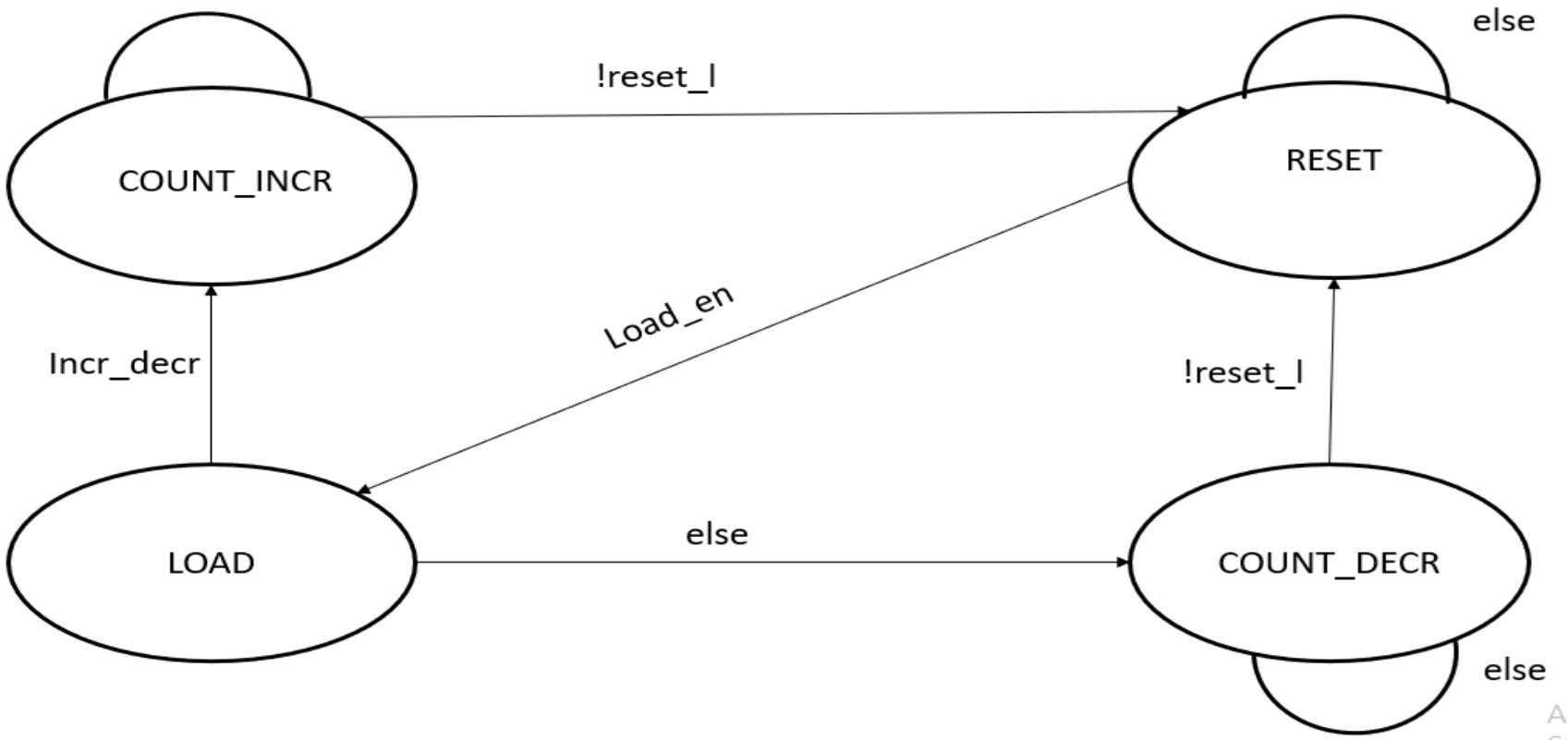
Over the year on looking at different sort of register IP and its user hardware logic, it can be said that the user logic mainly constitutes of the following constructs -

- State machines
- DQ tables
- Assign statements

State Machine Design Template

- A construct which makes transitions through a series of states based on external inputs and the current state of the machine
- Used for designing more complex hardware
- The hardware functionality can be broken down into a collection of states which determines when the system transitions from one state to another
- Uses can capture these states and align then on different external inputs the next state as described. Also, outputs at different states can be specified.
- User can also capture the state transitions from the current state to the next state and all the outputs associated with it

Sample State Machine Design



A sample template for capturing FSM

DQ Table Design Template

- A flip-flop is the basic storage element in a sequential logic, which is a fundamental building block of any electronic system
- Thus it becomes more important to give designers some way to make flops to capture sequential logic
- A template (DQ Table) can be used for the above purpose, where the user can specify the outputs in one column and specify the assignments on the other column

DQ Table Design Template Cont.

	DQ Name
description	

Q name	D Value

counter_reg[15:0]	if (!reset_l){
	counter_reg = 0
	}
	else{
	if (load_en){
	counter_reg = block1.load.load
	} else if (incr_decr){
	if((incr_decr_sat_val -counter_reg) <= incr_decr_val){
	counter_reg = incr_decr_sat_val
	} else {
	counter_reg = counter_reg + incr_decr_val
	}
	} else{
	if((counter_reg -incr_decr_sat_val) <= incr_decr_val) {
	counter_reg = incr_decr_sat_val
	} else {
	counter_reg = counter_reg - incr_decr_val
	}
	}
	}

Assign Table Template

- For designing circuits often dataflow modelling is used. Dataflow modelling describes the flow of data from input to output in hardware.

- The continuous assignment statement is the main construct of dataflow modelling and can be used to drive(assign) values.

- The introduction of the assign table as below can be used to capture dataflow modelling.

	assign Name	
description		

assign	value
block1.status.overflow	((counter_reg >= incr_decr_sat_val) && incr_decr) ? 1'b1 : 1'b0;
block1.status.underflow	((counter_reg <= incr_decr_sat_val) && !incr_decr) ? 1'b1 : 1'b0;

An assign table template

Non Addressable Registers Template

- Creation of application logic requires registers to store certain results, etc. A table can be incorporated which captures the register names and the width of the register.

- For making the non addressable registers as output, then a register can be specified in the table below with direction as out.

	register Name	
description		

register name	default
counter_reg[15:0]	0x0

A sample template to add non-addressable registers

Generated User RTL

- By using the mentioned templates, the user can generate the RTL as well as the UVM prediction model for their user logic interaction with the RTL of the addressable registers

- And along with it, the UVM prediction model will be hooked in the environment automatically for the same

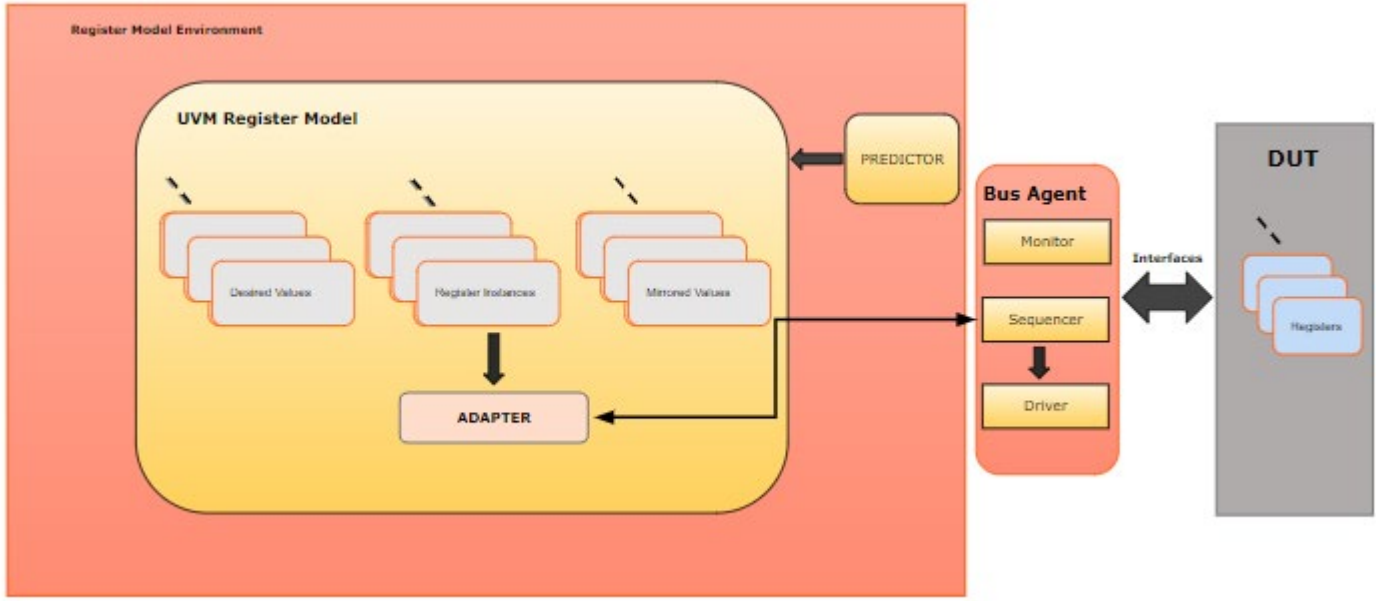
```
// FSM1: Registered outputs, sequential always block
always @(posedge clk) begin
    case(FSM1_next_state)
        RESET: begin
            counter_reg <= 0;
        end
        LOAD: begin
            counter_reg <= load_load_q;
        end
        COUNT_INCR: begin
        if((incr_decr_sat_val - counter_reg) <= incr_decr_val)
            begin
                counter_reg <= incr_decr_sat_val;
            end

        else
            begin
                counter_reg <= counter_reg + incr_decr_val;
            end
        end
        COUNT_DECR: begin
        if((counter_reg - incr_decr_sat_val) <= incr_decr_val)
            begin
                counter_reg <= incr_decr_sat_val;
            end

        else
            begin
                counter_reg <= counter_reg - incr_decr_val;
            end
        end

    endcase
```

Generated UVM Prediction Model

- The signal which corresponds to the non addressable registers can be monitored and can be compared with the UVM model, if the mismatch occurs then the model will give generate errors

- Call the predict method in UVM when there are assignments back to the IDS

  CSR registers (IDS CSR updated through applogic)



Generated UVM Prediction Model Example

```
task update();

    fork

    begin
        forever @(posedge hw_if.clk)
        begin
            if(!hw_if.re
                counter_
            end

            else begin
                if(hw_if
                    coun
                end
                else if(
                    if((

                    end

                    else

                    end
                end

                else beg
                    if((

                    end

                    else

                    end
                end
            end
        end
    end
```

```
    begin
        forever @(posedge hw_if.clk)
        begin
            overflow_status = ((counter_reg >= hw_if.incr_decr_sat_val) && hw_if.incr_decr) ? 1'b1 : 1'b0;
            underflow_status = ((counter_reg <= hw_if.incr_decr_sat_val) && !hw_if.incr_decr) ? 1'b1 : 1'b0;

            if(!rm.status.is_busy()) begin
                void'(rm.status.overflow.predict(overflow_status));
                void'(rm.status.underflow.predict(underflow_status));
            end
        end

    end

    join
```

Auto-Generated Tests

General algoritm for test generation

1. Create a list of all outputs from the app logic - app_logic_out_list
2. Create a list of all inputs to IDS generated register logic. Mark all the nets that are pairs (enable and input).
3. For each item in the above two lists, create a LOGIC CONE that drives the net
   a. The graph will have nodes and arrows going in (inputs) and coming out (outputs) of the node
   b. The node will be either combinational or sequential
   c. Combinational will simply have a combinational function based on the inputs
   d. Sequential will have a clock, reset and some properties that affect the clock
4. For each item, traverse the graph and find how to make the net transition to 1 and 0
5. There will be several solutions; store them in appropriate data structure

Auto-Generated Tests - Cont'd

```
rand int RV_16_0 ;
rand int RV_4_1 ;
constraint RV_16_0_constraint
{
    RV_16_0 <= RV_4_1;
}
constraint RV_4_1_constraint
{
    RV_4_1 >= RV_16_0;
}
task body;
    if(!uvm_config_db #(virtual block1_hw_if)::get(null,"*","block1_hif",block1_hwif)) begin
        `uvm_fatal("**CUSTOM_SEQ**", "cannot get block1_hwif from config_db")
    end

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : block1_block","cannot cast an object of type uvm_reg_sequence to rm");
    end

    if (rm == null)  begin
        `uvm_error("block1_block", "No register model specified to run sequence on, you should specify regmodel by using property 'uvm.regmodel' in the sequence")
        return;
    end
    #1000;
    block1_if.load_en = 'h0;

    block1_if.incr_decr = 'h1;

    void'(this.randomize());

    block1_if.incr_decr_val = RV_4_1;

    block1_if.incr_decr_sat_val = RV_16_0;

    while  (!block1_hwif.status_overflow_r)
    begin

        #1000;
    end
```

Auto-Generated Tests - Cont'd

```
function new(string name = "uvm_seq_1_seq") ;
    super.new(name);

endfunction

task body;

    if(!uvm_config_db #(virtual block1_hw_if)::get(null,"*","block1_hif",block1_hwif)) begin
        `uvm_fatal("**CUSTOM_SEQ**", "cannot get block1_hwif from config_db")
    end

    if(!$cast(rm, model)) begin
        `uvm_error("RegModel : block1_block","cannot cast an object of type uvm_reg_sequence to rm");
    end

    if (rm == null)  begin
        `uvm_error("block1_block", "No register model specified to run sequence on, you should specify regmodel by using
        property 'uvm.regmodel' in the sequence")
        return;
    end

    #1000;
    block1_hwif.valid = 'h1;

    block1_hwif.ff1 = 'h0;

    while  (!block1_hwif.reg1_f1_r)
    begin

        #1000;
    end
```

Coverage Report

- We can get code coverage of the design that is created to check whether all lines of code have been covered or not

Testplan | Design | DesUni

- top
  - block1_ambaapb
  - block1_hw
  - block1
- uvm_pkg (no coverage)
- block1_regmem_pkg
- config_pkg
- ambaapb_pkg
- arv_seq_pkg
- seq_pkg
- hw_pkg (no coverage)
- chip_name_iss_pkg (no
- test_pkg

Coverage Summary By Instance:

Scope	TOTAL	Statement	Branch	FEC Expression	Toggle
TOTAL	80.70	93.93	92.59	60.86	75.41
block1	83.97	91.30	90.47	80.00	74.11
apb	85.10	100.00	100.00	46.15	94.28

Local Instance Coverage Details:

Total Coverage:					74.86%	83.97%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	23	21	2	1	91.30%	91.30%
Branches	21	19	2	1	90.47%	90.47%
FEC Expressions	10	8	2	1	80.00%	80.00%
Toggles	1020	756	264	1	74.11%	74.11%

Recursive Hierarchical Coverage Details:

Total Coverage:					76.04%	80.7
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Cover
Statements	33	31	2	1	93.93%	93.9
Branches	27	25	2	1	92.59%	92.5
FEC Expressions	23	14	9	1	60.86%	60.8
Toggles	1090	822	268	1	75.41%	75.4

Benefits

- Fully configurable and customizable IPs/addressable registers that can be generated for varied set of needs
- All the generated files are available as plain text for easy debugging and use by downstream tools
- Generation of the synthesizable RTL for both the user logic and the addressable register logic
- The complete IP / entire design can further be extended for the verification of the design.
- Supports specification of finite state machines (FSMs), data paths, signals, and other parts of custom IP blocks in your application logic
- For both standard and customer blocks, IDS-IPGen generates RTL models, UVM verification models, and on the fly AI based tests that provide high functional and code coverage right out-of-the-box.

Conclusion

- Reduction in time and cost of development
- Focus more on creating the algorithm for your design, and let the tool handle and ensure the correctness
- Complexity can be handled by using abstraction
- One of the forms of abstraction is reuse
- Reuse is possible if the IPs are customizable and configurable
- Create the complete IP and validate the synthesizable design through auto-generated tests with optimum code coverage

Questions