How to use IDesignSpec with UVM?

This document discusses the use of IDesignSpec: Automatic Register Model Generator to generate a Register Model for an IP of SoC.

Agnisys, Inc.
1255 Middlesex St. Unit I Lowell, MA - 01851, USA.

Introduction

A register model can be written by hand. But, if the number of registers increases, this can become a big task and is always a potential source for errors. So it is better to get an automatic Register Model generator like IDesignSpec, as this may be helpful in number of ways:

- The register model can be re-generated whenever there is a change in the register definition.
- The register model can be generated efficiently without errors.
- It allows a common register specification to be used by the hardware, software and verification engineering teams.

IDesignSpec™ (IDS) generates all downstream code and derived documentation for the addressable registers in the design from the single source document. One of the outputs generated from the IDS is register model for the Universal Verification Methodology (UVM) register package. IDS automatically generate the UVM Register Model for the Registers defined in the IDS document (Word/Excel/Open-Office/Frame).

This document shows how one can easily specify registers and generate a UVM register model from it.
Consider the following example of a register defined inside the block, in IDS Word (the Word version of IDS).

IDS_blk		IDS_blk		Block	0x00001
offset	01	External		Size	

IDS_reg_A		IDS_reg_A		Reg.	0x00011
offset	'h10	External			

Sample Register of Block IDS_blk

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

bits	name	s/w	h/w	default	Description
31:0	Fld_1	Rw	Ro	'hA	Field for the IDS_reg_A register with reset value equal to 'hA

This will create a Register Model containing 'IDS_reg_A' register inside the 'IDS_blk' block. IDS_reg_A register class will have the Field named 'Fld_1' with configurations taken from the above template.

```
Fld_1.configure(this, 32,  0, "RW", 0, 'd10, 1, 0, 0);
```

Field name — Field size *(in bits)* — Field LSF — Software Access — Reset Value

Likewise, IDS can generate Register Arrays, Memories, Indirect Access Registers, Fifo Registers and Coverage Models that are supported by the UVM register model in its UVM output. Each one of these is discussed later.

Usage of Register model

Once the Register Model is generated from the Register Specifications of the DUV (design-under-verification), it is to be tested in the UVM Environment. User can also use RTL (Verilog/VHDL) generated by IDS in the UVM Environment as a DUV. Design and BUS Interface is created where bus (controlling the traffic of the transactions) signals are connected to the design. This Interface is the Top Level HDL Path. See the example below in which the 'apb' bus signals are connected to the RTL design (generated by IDS).

```
reg [bus_width-1:0] IDS_reg_A;                    // Creating Register with same name as in
```

```
                                                    UVM Register Model
  assign IDS_reg_A = BLK_DUV.IDS_reg_A_rd_data;     // Connecting Register with the Register
                                                       inside the Design
  IDS_blk_ids #(bus_width) BLK_DUV(                 // Instantiation of DUV(IDS_blk_ids)
    .clk(apb.pclk,                                     connected with Interface
    .reset_l(~rst),
    .address(apb.paddr),                            //apb bus connected to the DUV
    .wr_data(apb.pwdata),
    .rd_data(apb.prdata),
             . . . . .   );
```

Once, the bus signals are connected with the design, the TOP level DUV is created, where the bus interface and the DUV interface are instantiated.

```
   apb_if apb0(clk);                                //Instantiation of Bus Interface
   DUV_if DUV(apb0, rst);                           //Instantiation of DUV Interface
```

This completes the HDL part for the UVM Environment. For the support of the UVM Environment, Register Sequence classes need to be developed for the Read/Write operations in the Registers. User can also use the UVM Bulit-In Register Sequences "uvm_reg_mem_built_in_seq". Or can create own sequence classes using "uvm_reg_sequence" as its base class. This class provides base functionality for both user-defined RegModel test sequences and "register translation sequences". Here, uvm_ids_reg_seq class is created with uvm_reg_sequence as its base class.

```
    class uvm_ids_reg_seq extends uvm_reg_sequence #(uvm_sequence #(uvm_reg_item));
```

Similarly, build the Sequences for Memories defined in the Register model. After, the sequencers are created for the Registers and Memories, the testbench environment class needs to be created. The testbench environment class with uvm_component as its base class, is architected to provide a flexible and extendable verification component. The main function of this class is to model behavior by generating constrained random traffic, monitoring DUV responses and checking the validity of the protocol activity.

```
class tb_env extends uvm_component;
 . . . . .
```

This class consists of Register Model, Bus agent and Sequence class. In this class, HDL root path is defined for the back-door accessibility. Here, the bus adapter and the run task is implemented.

```
   IDS_blk_block regmodel;           // IDS Generated UVM Register model
   apb_agent    apb;                 // APB BUS (UVM already supports this bus interface)
   uvm_reg_sequence seq;             // UVM Sequence class
     . . . . . .
        string hdl_root = "top.DUV";                // Top level HDL Path
        . . . . .
        regmodel.set_hdl_path_root(hdl_root);       // Defining HDL Path to Rg-Model for Back-
                                                       door access
      end
     . . . .
         regmodel.default_map.set_sequencer(apb.sqr,reg2apb);      //Connecting the Bus Adapter
    . . . . . .
   virtual task run_phase();                        // run Task
    . . . . . ;
```

After the testbench environment is setup, the testbench is created, where all the above classes are used. Testbench program collects all the files in the Environment.

```
`include "apb.sv"                           // Include APB Bus interface
`include "uvm_top_DUV.sv"                   // Include TOP level DUV
`include "uvm_seqlib.sv"                    // Include Register Sequence Classes
`include "ids/IDS_blk.regmem.sv"            // Include IDS generated Register Model
import IDS_blk_regmem_pkg::*;               // Import the Package containing Register Model
                                               Classes
program tb;                                 // testbench "tb" program
 . . . . .
 `include "uvm_env.sv"                      //Include the Testbench Environment Class
 . . . . . .
```

```
endprogram
```

With this the Environment is build up and is ready to be tested. On compiling the Environment only user have to do is, compile the testbench file and automatically all the Environment files will be compiled. And then run the test giving the sequence class name on the command line. Whichever sequences are given, that will be run in the UVM Environment. This is how; the IDS generated register model is used. There, are other ways of using the Register Model, depending on the User needs.

Creating Different types of Register Models using IDS

Register Array

The UVM Register Array is defined in IDS by creating one register in a register group with repeat count equals to the array size. IDS also support multi dimensional arrays, by mentioning RegGroup inside another RegGroup.

reg_grp	reg_grp	RegGrp	0x00015
offset	external	repeat 20	size

reg_array	reg_array	Reg.	0x00015
offset	external		

7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	Description
7:0	Fld_1	Rw	Ro	'h0	

End RegGroup

For this, 'reg_array[20]' array is created in UVM Register model.

Memory

In UVM Register model, memory can be created using IDS by the same procedure as it is done for Register Array. Only in this case, the RegGroup is marked as External equals to true. The size of the Memory will be calculated by the starting and end address of the RegGroup.

reg_ext	reg_ext	RegGrp	0x00029
offset	external true	Repeat 0x14	size

reg_mem	reg_mem	Reg.	0x00029
offset	external		

7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	Description
7:0	Fld_1	Rw	Ro	'h0	

End RegGroup

In the above sample memory will be created 'reg_mem' having size equal to 8(RegGroup size for one iteration) multiplied by 'h14 (Repeat count).

Register File

UVM supports register file, which contains more than one Register. To specify a register file in IDS, simply put one or more registers in a RegGroup, and then a UVM Register File is generated. Note that it is possible to have a RegGroup inside a RegGroup just like it is possible to have a Register File inside a Register File in UVM.

Coverage

In IDS, coverage code can be generated in the UVM Register model defined for Blocks, Registers, Memories and Fields. User can get the coverage for that element, by adding a property named "Coverage" with value "on". By default the coverage will be "off".

reg_cov				reg_cov				0x0003d	
Offset				external					
Coverage		On							

7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	Description
4:0	Fld_1	Ro	Ro	'h0	

In the above, coverage code will be generated for 'reg_cov' register.

Indirect Indexed Register

Some registers are not directly accessible via a dedicated address bus. Indirect access of an array of such registers is accomplished by first writing an "index" register with a value that specifies the array's offset, followed by a read or write of a "data" register to obtain or set the value for the register at that specified offset.

In IDS, Indirect Registers are supported by creating the following properties for the data register.
Type=indirect
index_reg=<name of Index Register from the Spec> - This is the corresponding index register in the IDS Spec.
Depth=<number> - This is the depth of the external register.

Data_reg			Data_reg			0x0003e
offset			external			
type	Indirect		Index_reg	reg_cov	Depth	256

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

bits	name	s/w	h/w	default	Description
15:0	Fld_1	Ro	Ro	'h0	

This generates, "Data_reg" as an indirect register, with its index register as "reg_cov" and an external Register will be named as 'data register_indirect' with depth equal to 256.

Fifo Register

There is a separate class in UVM Register Model for defining the Fifo Registers "`uvm_reg_fifo`". In IDS, Fifo Registers are supported by setting the property "type" with value "fifo".

List of all UVM Properties

This is a list of all properties for UVM Register Model supported by IDS. Note that the property names are case insensitive.

Property	Value	Description
Coverage	on/off	Include Coverage for that component and to its Lower Hierarchy.
Type	Fifo indirect	Register is Fifo Register. Register is Indirect Register.
index_reg **depth**	INDEX_reg1 256	Additional properties for indirect register. Specifies name of the index register and depth of the external register.
Variant	Value e.g.: V1, V2	The Register Model could be Generated based on the Variant mentioned.
Class **class_uvm**	Class name e.g: uvm_my_class	User can generate its own Class for any component in the Register Model.
Rand	On/off	Randomization of the component can be controlled by the user.
Enum	--	Special table for Enum in IDesignSpec.
Map	Text e.g.: AHB, APB	Address Mapping can be controlled as specified.
Hdl_path	Hierarchy name e.g. (top.dut.reg1)	Hdl path can be given for any component.

Table: List of all UVM Properties supported by IDS

Summary

This article has shown how convenient it is to specify registers or memories in IDesignSpec and create UVM code from it. So, it is better to use an automatic Register Model generator. As in IDS, once the Register description is complete, all user have to do is generate Register Model for UVM and include that in the UVM Environment as discussed above. Any changes in the Register description, with a small effort user can regenerate the Register Model again with zero margins for error.

IDS also supports advanced concepts supported by the UVM Register Class, like Address Mapping according to the user defined bus, Change in the class name for the register, block or register file as specified by the user, include hdl path for the component and all other things in the UVM Register Model, are generated by IDS. Thus, IDS supports complete UVM Register Model.