February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

SYNOPSYS® Accelerating Innovation

An Approach for Faster Compilation of Complex Verification Environment: The USB3.0 Experience

By

Mahesha Shankarathota (maheshas@synopsys.com)

Vybhava S (vybhava@synopsys.com)

Indrajit Dutta (indrajit@synopsys.com)

Synopsys India Pvt Ltd. Bangalore-India

Overview

- 6X compilation time gain using VCS® (2011.12) Partition Compile Methodology
- Example of a complex USB3.0 Configurable VMM Verification Environment
- Motivation and Methodology
- Coding guidelines and good practices
- Summary of Results and Conclusions

```
program p;
virtual SBus VirDutBus = tb.DutBus;
import TransactorPkg::*;
SBusTransctor xactor = new(VirDutBus);


initial begin

fork

  begin
    xactor.request();
    xactor.wait_for_bus();
  end


  begin
    wait(tb.DUT.DutBus.req == 1);  //XMR
    //wait(VirDutBus.req == 1);
    wait(tb.DUT.DutBus.grant == 1);
    //wait(VirDutBus.grant == 1);   // XMR replaced by Virtual Interface
  end
join
$unit::cov1.sample();
end
endprogram
```

```
package TransactorPkg;
  class SBusTransctor;

    virtual SBus bus;

    function new( virtual SBus s);
      bus = s;
    endfunction

    task request();
     bus.req <= 1;
    endtask

    task wait_for_bus();
     @(posedge bus.grant);
    endtask

  endclass
endpackage
```

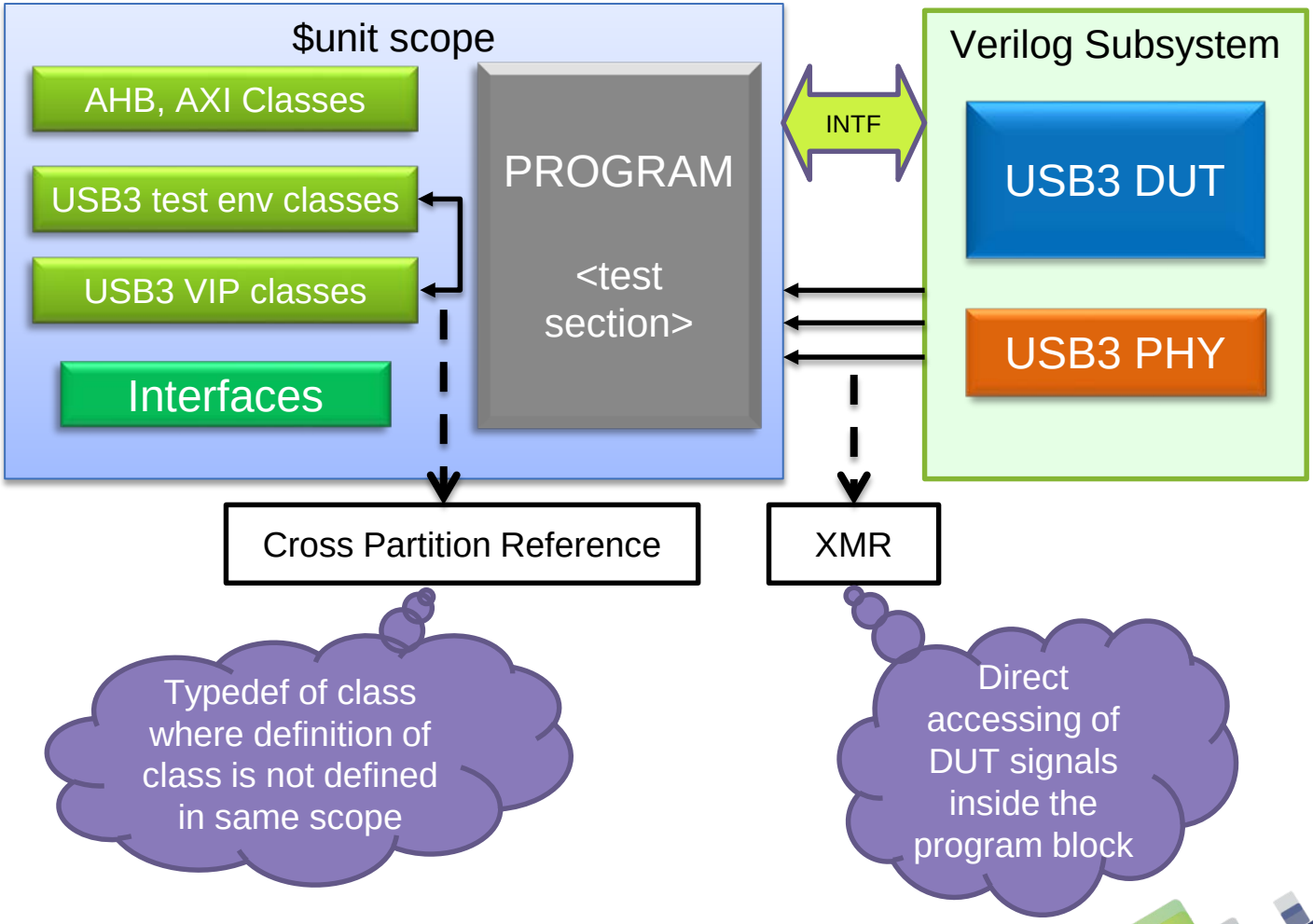
```
module tb;
  interface SBus;
    logic        req,grant;
    logic [7:0] addr,data;
  endinterface

  module dut(SBus DutBus);
  initial begin
    wait(DutBus.req == 1);
    dutBus.grant =1;
    dutBus.addr =0;
    dutBus.data = 15;
  end
  endmodule

  Sbus DutBus();
  dut DUT(DutBus);
endmodule
```

```
// Code in $unit scope
covergroup cov;
option.per_instance =1;
cp1: coverpoint tb.DutBus.req ;
cp2: coverpoint tb.DutBus.grant;
cp3: coverpoint tb.DutBus.addr;
cp4: coverpoint tb.DutBus.data;
endgroup
cov cov1 = new();
```

$unit is the name of the scope that encompasses a compilation unit. Compilation unit refers to module, interface, package, and program blocks.

USB 3.0 verification environment

This environment uses VCS single compile flow for simulation.



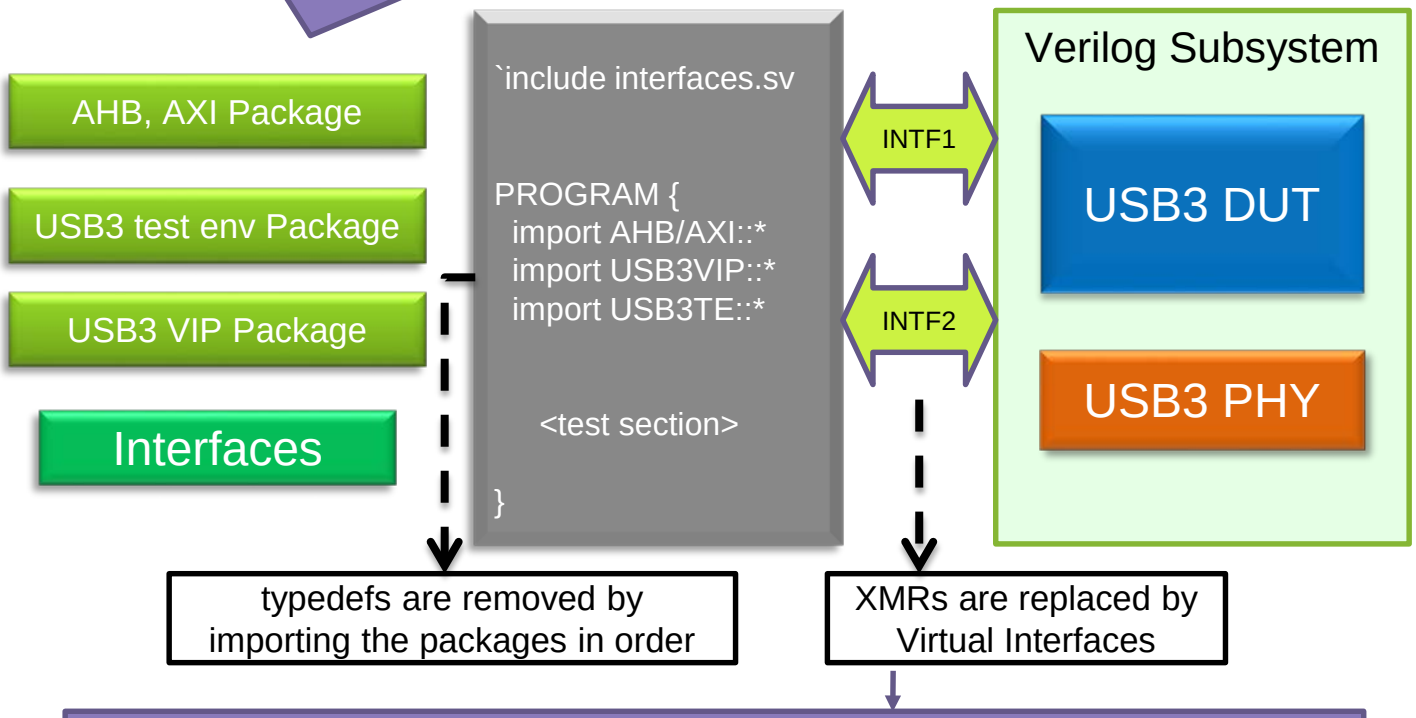
Adopting Partition compile Flow

- Partitions need to be made as packages/modules/programs.
- Avoid code in $unit (code-changes in $unit scope trigger recompilation of the entire design).
- Identify the Partitions - USB3 TE partitioned well into AHB/AXI, USB3 VIP, DUT+PHY, COM, TST partitions.
- SV code modifications to remove XMRs(Cross module references) and Cross Partition References.
- The biggest challenge for us was converting all the XMRs to interface signals.
- Update of makefiles and scripts.

SystemVerilog packages provide an additional mechanism for sharing parameters, data, type, task, function, sequence and property declarations. Packages can be imported or referenced in the SystemVerilog module, interface, and program blocks.

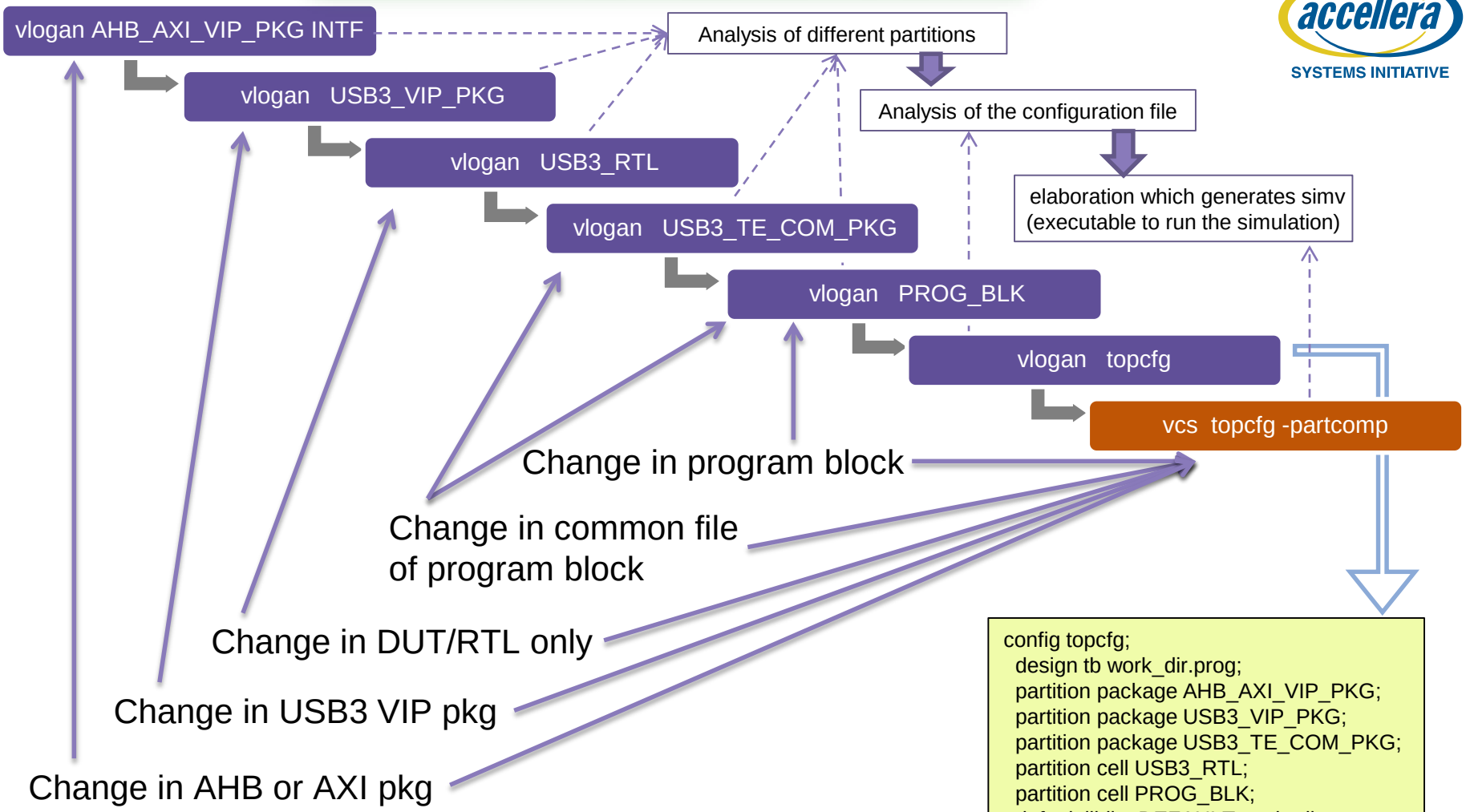
USB 3.0 verification environment

This environment uses VCS partition compile flow for simulation.



A virtual interface allows the same subprogram to operate on different portions of a design and to dynamically control the set of signals associated with the subprogram. Instead of referring to the actual set of signals directly, users are able to manipulate a set of virtual signals.

PARTITION COMPILE FLOW STEPS




```
config topcfg;
  design tb work_dir.prog;
  partition package AHB_AXI_VIP_PKG;
  partition package USB3_VIP_PKG;
  partition package USB3_TE_COM_PKG;
  partition cell USB3_RTL;
  partition cell PROG_BLK;
  default liblist DEFAULT work_dir;
endconfig
```

For compilation from scratch, run all steps

Partition Compile Coding Guidelines

- Avoid code in $unit scope – It triggers re-compilation
- No forward references to different compilation-unit scope
- No direct access of DUT signals in program scope
- All DUT signal accesses should be through virtual interface
- No force, release constructs of the signals

No direct access of signals from TE

- All signal accesses should be done through interface signals.

Single Compile	Partition Compile
tp1926.sv begin **wait (** **tb.U_DWC_usb3_subsys.ulpi_tx_data[7:0] ==** **8'h00);** end	**DWC_usb3_if.sv** **interface DWC_usb3_tb_signals_if ();** **logic [7:0] ulpi_tx_data;** **endinterface //DWC_usb3_tb_signals_if** **ec_hst.sv** **gbl.tb_signals_if=** **`TB_INST_NAME.tb_signals_if;** **gbl.clk_rst_if = `TB_INST_NAME.clk_rst_if;** **gbl.misc_if = `TB_INST_NAME.misc_if;** **tp1926.sv** begin **wait (gbl.tb_signals_if.ulpi_tx_data[7:0] ==** **8'h00);** end

Cross Partition Reference

Single Compile Flow

```
typedef bit[3:0] TYPE;

typedef class operator_class;

class update_vecs_class;

  TYPE n1;
  TYPE n2[4:0];
  TYPE n3[3:0][3:0];
  TYPE n4;
  operator_class op;

  task put(TYPE in[3:0]);
    op = new;
    n1 = in[0];
    n2[3:0] = in;
    n3 =  '{4{in[3:0]}} ;
    n4 = op.invert(n1);
  endtask;

  function TYPE get(int j);
    return n2[j];
  endfunction

endclass
```

$unit scope

```
class operator_class;

TYPE x1;

function TYPE invert(TYPE in);
x1 =  ~in;
invert = x1;
endfunction

endclass
```

Partition Compile Flow

```
package p1;
typedef bit[3:0] TYPE;

 class operator_class;

 TYPE x1;

 function TYPE invert(TYPE in);
   x1 =  ~in;
   invert = x1;
 endfunction

 endclass

endpackage
```

```
import p1::*;

class update_vecs_class ;

 TYPE n1;
 TYPE n2[4:0];
 TYPE n3[3:0][3:0];
 TYPE n4;
 operator_class op;

  task put(TYPE in[3:0]);
    op = new;
    n1 = in[0];
    n2[3:0] = in;
    n3 =  '{4{in[3:0]}};
    n4 = op.invert(n1);
 endtask;

 function TYPE get(int j);
    return n2[j];
 endfunction

endclass
```

```
// Common Program in both flows
program p;
  update_vecs_class T1 = new;
  initial begin
    TYPE in[3:0],in0,in1, in2, in3;
    in[0] = 10;
    T1.put(in);
    in0 = T1.get(2);
    $display( "in data = %d get data = %d shift_data =
%d\n", in[0], in0, T1.n4);
  end
endprogram
```

Removal of XMRs – Example

- The RAM "XX" initialization logic from the program scope is moved to Verilog Subsystem

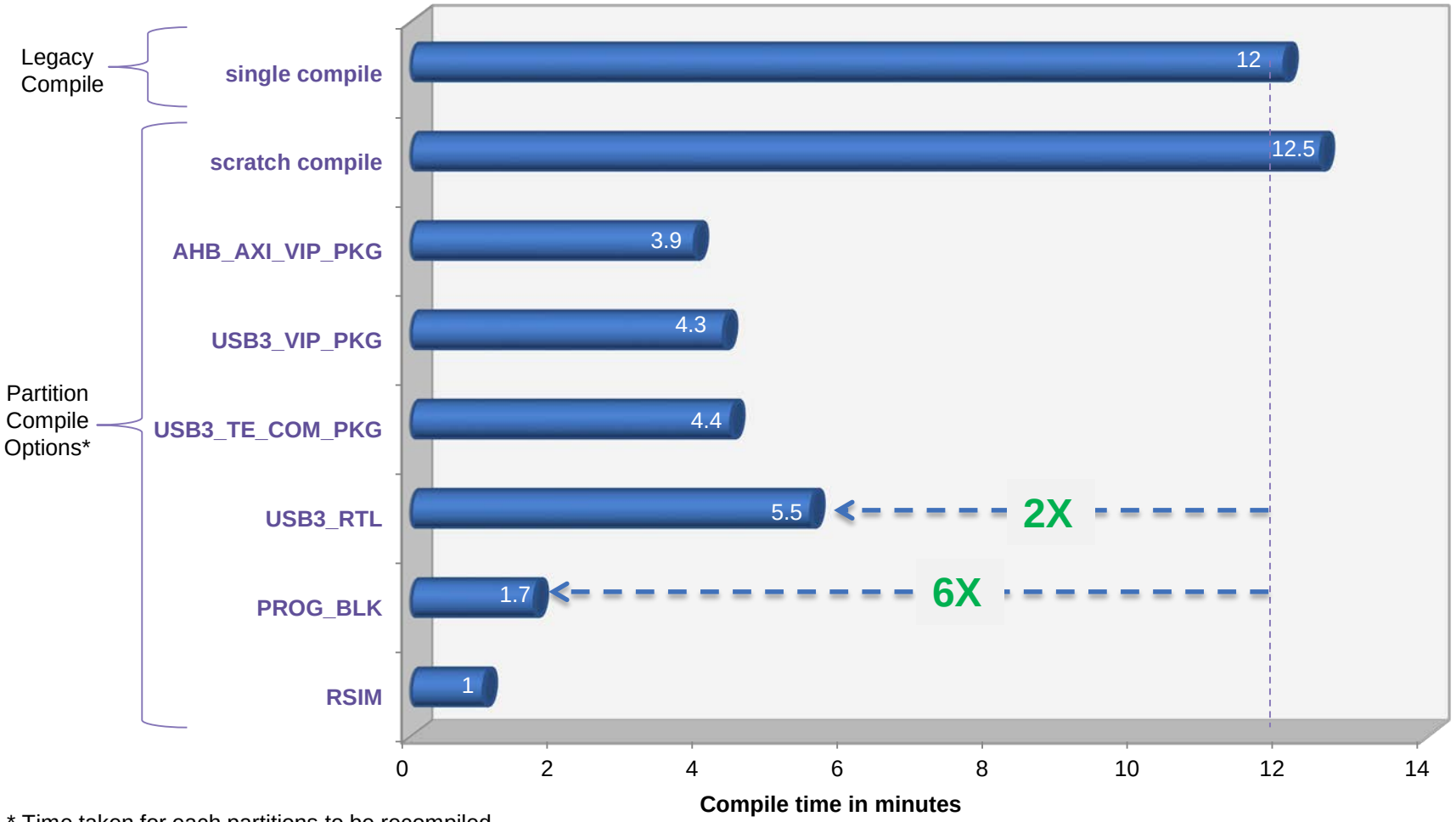
ad_xactor.sv

```
`ifdef DWC_USB3_DPRAM_PORT_EN
  `PRINT_NORMAL($psprintf("%0s: Re-init
DPRAM0 with X ",prefix));
  `ifdef DWC_USB3_RAM0_PORT1_EN
    gbl.misc_if.dpram_initx[0] = 1'b1;
    wait(gbl.misc_if.dpram_initx_done[0] == 1'b1);
    gbl.misc_if.dpram_initx[0] = 1'b0;
    `PRINT_NORMAL($psprintf("%0s: Done Re-init
of DPRAM0 with X ",prefix));
  `endif
`endif
```

DWC_usb3_subsys.sv

```
`ifdef DWC_USB3_DPRAM_PORT_EN
  `ifdef DWC_USB3_RAM0_PORT1_EN
    always
    begin
      #1;
      @(posedge misc_if.dpram_initx[0]);
      for (int i = 0; i < U_RAM0_dpram.DEPTH; i= i+1)
      begin
        $display("DWC_usb3_xmrs:Done Re-init of
DPRAM0 i=%d",i);
        U_RAM0_dpram.mem_array[i] =
        {U_RAM0_dpram.DATA_WIDTH{1'bx}};
      end
      misc_if.dpram_initx_done[0] = 1'b1;
    end
  `endif
`endif
```

Summary of Results



* Time taken for each partitions to be recompiled

Conclusions

- The legacy single compile flow takes about 12 minutes of compile time each time. The typical usage is change in the test (PROG_BLK), which takes about 2 minutes to be compiled. This provides a 6X gain compilation time.
- When the simulation run-time options change (RSIM), there is no compile to be done and we see maximum improvement.
- Adhering to good coding practices makes migration easier.
- Migration has overhead of maintaining both the flows in sync until partition compile flow is stable.