February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

Using Advanced OOP Concepts To Integrate Templatized Algorithms for Standard Protocols With UVM

by

Anunay Bajaj

Gaurav Chugh

Agenda:

- Reusability: Leaps and Lapse
- Protocol Commonalities
- Hidden Time traps that we do not know
- UVM With a difference: Proposed packages
- Summary

Why Reusability : Recap

Reduce Repetition

- No need to reinvent the wheel

Clean and Compact look

- Saves compile time

Forward Compatibility

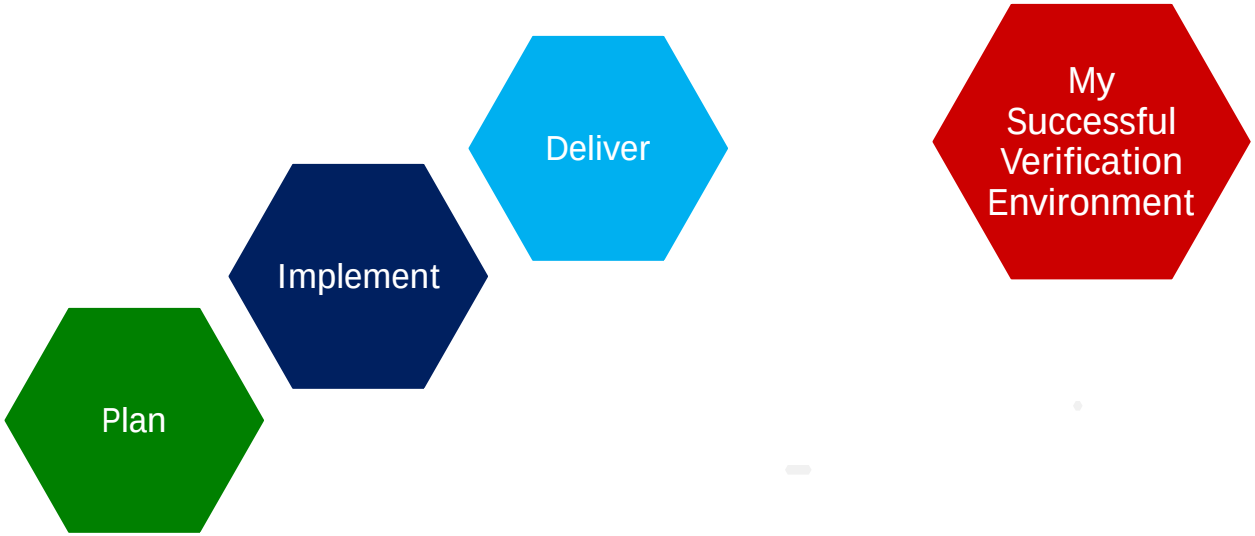
- The greatest gift

Easier to understand and debug

- Debug effort reduced

The PID cycle

- Plan: Covers all the stages of environment architecture
- Implement: Includes the code based on Plan
- Deliver: Successful Product delivery

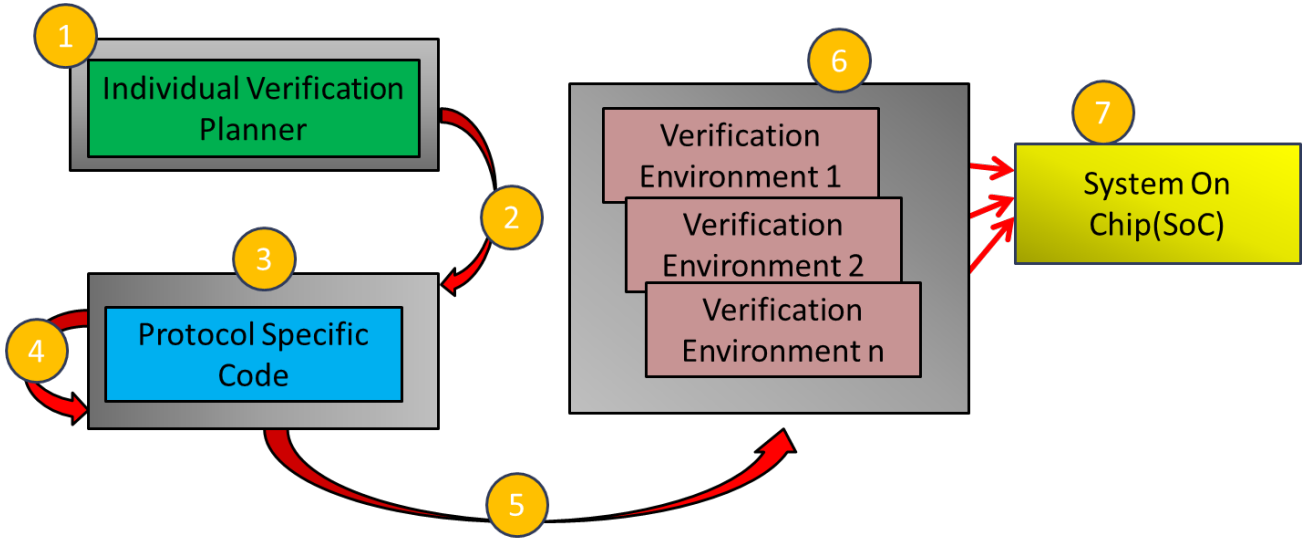


Protocol commonalities

- Encoding and Decoding
- Data Integrity
- Clock Generation and Recovery
- Data Sampling
- Digital Filters
- Running Disparity.....and many more

The Pitfall

- Code-Recode of similar logic or algorithms across multiple verification environments
  - -Hidden man-hours being wasted
  - -Need to verify the logic every time
  - -Varying coding standards



Questions popping up

Am I satisfied with my Reusability?

Can I still reduce my PID time?

Have I overcome Hidden Time traps?

Are the returns of UVM enough?

Do I see an Opportunity begging?

My UVM wants to be Intelligent

- Understand the need of the architecture (Plan)
- Select the appropriate enhancement package(Implement)
- Identify the algorithms as per the need
- Reuse the algorithms

Proposed Packages

Standard Template Algorithms Container(STAC) classes

- Template classes for standard encoding/decoding and data integrity schemes
- Linear encoding

Standard Template Math Container(STMC) classes

- Templates for DSP specific blocks requiring transforms and filters

Standard Template Utility Container(STUC) classes

- Templates for day to day utility blocks pertaining to clock, data sampling etc.

Classes in the packages

Sponsored By:

STAC

- uvm_stac_standard_encoder
- uvm_stac_data_integrity_encoder
- uvm_stac_linear_encoder

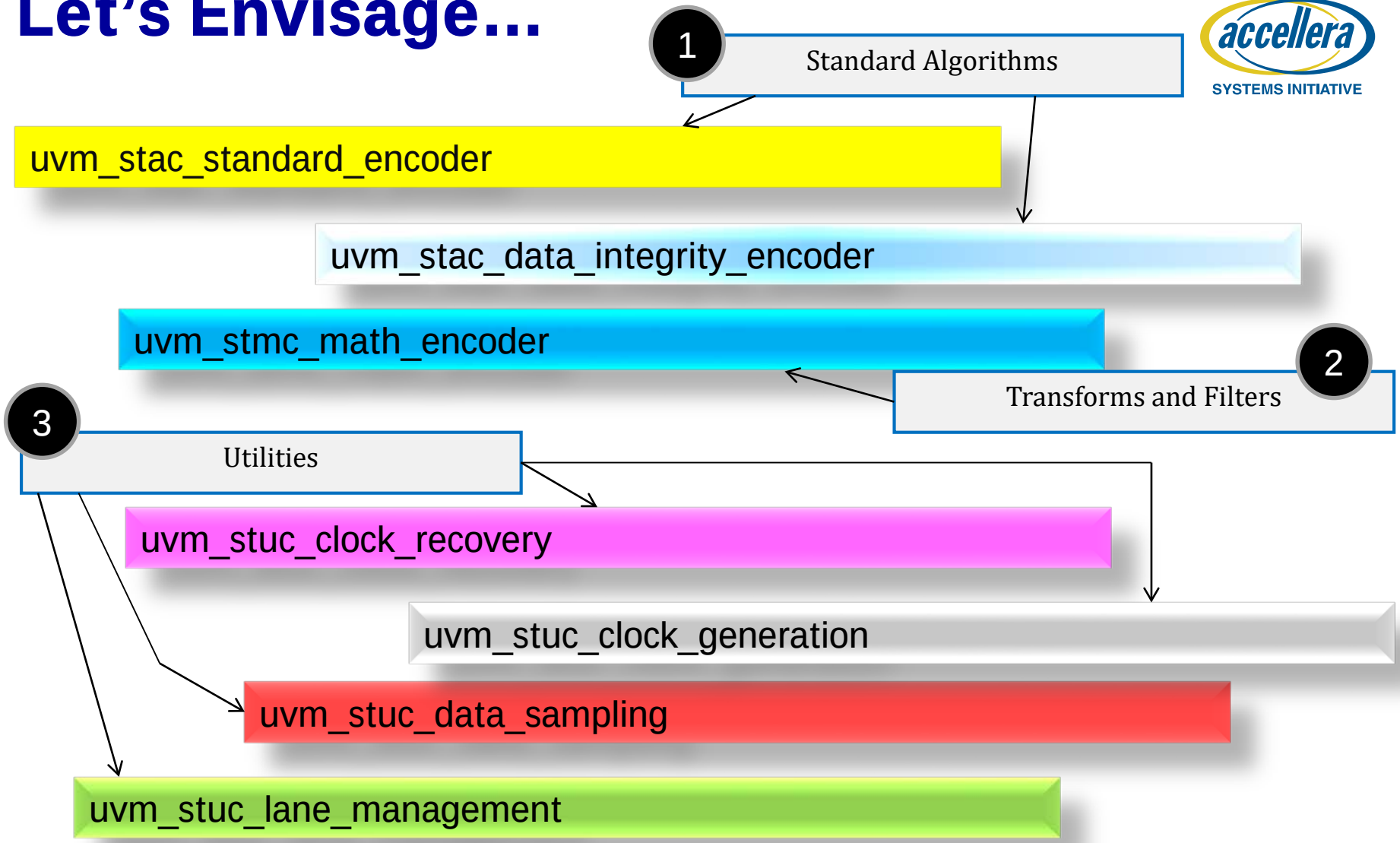
STUC

- uvm_stuc_clock_recovery
- uvm_stuc_clock_generation
- uvm_stuc_calc_running_disparity
- uvm_stuc_data_sampling
- uvm_stuc_lane_management

STMC

- uvm_stmc_math_encoder

Let's Envisage...



Container classes code structure

```
package <package_name>;
```

1

Three prime components

```
class container_name extends uvm_object
#(int     ENCODING_TYPE=0,
    type  INSTREAM=bit/int,
    type  OUTSTREAM=int/int,


  ) extends uvm_object;
```

ENCODING_TYPE in UVM document

2

More container related declarations

3

```
virtual function bit/void <algorithm_specific>
        (input INSTREAM _in_data,
         output OUTSTREAM _out_data
         );

endfunction
```

4

Algorithm selection machine calls desired conversion scheme

5

```
endpackage
```

Encoding Schemes Tabulated in UVM User Manual

S No.	Coding Scheme	Parameter
1.	8b/10b	`8B10B
2.	N-bit CRC Check	`NBITCRC
3.	Manchester Coding	`MANCHESTER
4.	Digital Butterworth Filter	`FILTER

1. This list will be exhaustive
2. Shall be updated from time to time

Steps for the user

- Specialize the desired template class with appropriate values
- Create the object of the specialized class
- Call the *algorithm_specific* function and provide input *_in_data*
- Obtain Encoded/Decoded *_out_data* from the same function

1.a Standard Algorithms

1

```
class uvm_stac_data_integrity_encoder
      #(int  ENCODING_TYPE=0,
        type INSTREAM=bit,
        type OUTSTREAM=bit,
        int  ENCODE_POLYNOMIAL=1
       )extends uvm_object;

  int _e_type = ENCODING_TYPE;
  int _e_poly =ENCODE_POLYNOMIAL;
```

2

```
virtual function void encode_data_integrity
        (input  INSTREAM _in_data,
         outputOUTSTREAM _out_data
        );
       case(_e_type)
         `NBITCRC:  f_n_bit_crc
            (_e_poly,_in_data,_out_data);

         `ODDPARITY:  f_odd_parity
            (_in_data ,_out_data);

         `EVENPARITY: f_even_parity
            (_in_data,_out_data);

         `USER_DEFINED_CODE:
               f_user_coding_scheme
           (_e_poly,_in_data,_out_data);
          //More Coding Algorithms
      endcase
    endfunction
```

3

```
virtual function void f_n_bit_crc
      (input ENCODE_POLYNOMIAL _e_poly,
             INSTREAM _in_data,
            output OUTSTREAM _out_data
      );
             // Input_1  :  Polynomial
            // Input_2  : Input Data
           // Output   : N-Bit CRC value
       endfunction
```

1.b Standard Algorithms

1

```
class uvm_stac_standard_encoder
      #( int ENCODING_TYPE=0,
         type INSTREAM=bit,
         type OUTSTREAM=int
       )extends uvm_object;

int _e_type = ENCODING_TYPE;
```

2

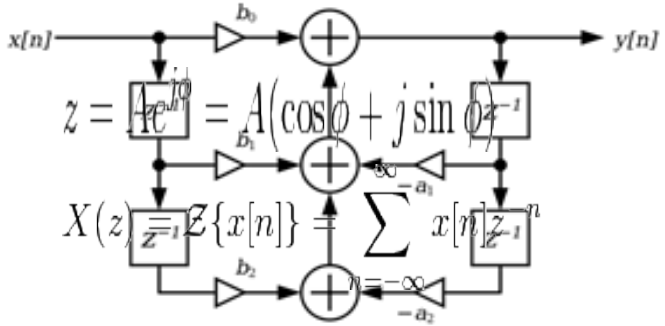
```
virtual function bit standard_encode_data
      (input INSTREAM _in_data,
       output  OUTSTREAM _out_data
       );
    case(_e_type)
      `8B10B_CODE:
        f_8b_10b(_in_data,_out_data);

      `128B130B_CODE:
         f_128b_130b(_in_data,_out_data);
          /* -------------------------
          More Standard Encoding/Decoding
          Schemes in the code
          -------------------------*/
       `USER_DEFINED_CODE:
f_user_coding_scheme(_in_data,_out_data);
       endcase
    endfunction
```

3

```
/* -------------------------
Generic Conversion Functions
Definitions for all the algorithms
 ------------------------- */
virtual function void f_8b_10b
   (input INSTREAM _in_data,
    output OUTSTREAM _out_data
    );
     // Input : 8-bit data
    // Output : 10-bit encoded data
endfunction
```

2. Transforms and filters

1



2

```
class uvm_stmc_math_encoder
 #( type  INSTREAM=bit,
    int   ENCODING_TYPE=0,
    int    ORDER=0,
    int    PHASE=0,
    int    POLES=0,
    int    ZEROS=0,
    int    TIME_LOWER_LIMIT=0,
    int    TIME_UPPER_LIMIT=0,
    type  OUTSTREAM=bit
  ) extends uvm_object;
```

3

```
 virtual function bit math_encoder
(input ENCODING_TYPE _e_type,
 ORDER _order,
 PHASE _phase,
 POLES _poles,
 ZEROS _zeros,
 TIME_LOWER_LIMIT _t_ll,
 TIME_UPPER_LIMIT _t_ul,
 INSTREAM _in_data,
 output _out_data
);
 case(_e_type)
   `Z_TRANSFORM:
      out_data=
        f_z_transform(_phase,_t_ll,_t_ul,_in_data);

   `FILTER:
      out_data=f_filter(_order,_poles,_zeros);
            // …….. Other cases here
 endcase
endfunction
```

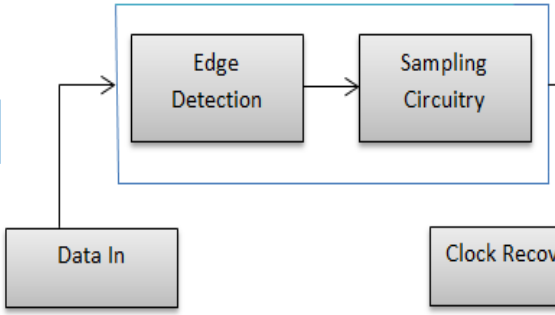
3.a Utilities :Clock Recovery


```
class uvm_stuc_clock_recovery extends uvm_object;
        virtual task uvm_clock_rec
            (ref bit _in_data,
                        _clk,
             ref _out_data
            );
          t_edge_detection_and_sampling_circuitry
                                ( _in_data, _clk, _out_data);
         endtask
endclass //uvm_stuc_clock_recovery
```

User Input data

User specified Ref Clock

Output Clock



Basic Clock Recovery Block Diagram

1. The task wraps Edge detection and Sampling Circuitry into a single task
2. Very Simple in nature
3. More complex blocks can be evolved

3.b Utilities :Clock Generation

The most common block

```
class uvm_stuc_clock_generator extends uvm_object;
    virtual task uvm_clock_gen
        (ref clk,
         ref freq,
         ref duty_cycle
        );
          //Clock generation code here
    endtask
endclass //uvm_stuc_clock_generator
```

User specifies values by over-riding this task

Sample Top Level Module of user Testbench

```
//Top level module
module top;
  bit clk;
  real freq;
  real duty_cycle;

  uvm_clock_generator clk_gen_object;
  clk_gen_object=new();
   initial begin
    clk_gen_object.uvm_stuc_clock_gen (clk,freq,duty_cycle);
     //Other user code here
    end
 endmodule
```

Allocate memory

Call the task with user specified inputs

3.c Utilities :Data Sampling


```
class uvm_stuc_data_sampling
    #( int SAMPLING_TYPE=0
     ) extends uvm_object;

int _s_type=SAMPLING_TYPE;
```

SAMPLING_TYPE in UVM document

```
virtual task data_sampling
      (ref bit _in_data,
       ref bit _reference_clk=0,
        ref bit _out_data
       );
       case(_s_type)
       `NRZ:
          t_nrz_code(_in_data, ,_out_data);
       `MANCHESTER:
          t_manchester_code (_in_data,
                    _reference_clk, _out_data);
        `PSK:
          t_psk_code(_in_data,out_data);
        //More sampling algorithms
       endcase
endtask
```

```
virtual task t_nrz_code(input _in_data,
                        output _out_data);
          //NRZ Code conversion here
endtask

virtual task t_manchester_code(input  _in_data,
                               _reference_clk,
                              output _out_data);
//Manchester Code conversion here
endtask
```

3d. Lane Management

Sponsored By:

```
class uvm_stuc_lane_management
          #(type INSTREAM=bit,
                int WIDTH=0,
                int LANES=0,
             )extends uvm_object;

            int _active_lanes;
             bit [WIDTH-1:0] _lane_count [LANES][$];
          virtual function void lane_mgmt
                (input INSTREAM _indata,
                       WIDTH _width  );
              case(_active_lanes)
                 //Function calling based on the number of
                 //active lanes
              endcase
         endfunction
endclass //uvm_stuc_lane_management
```

Summary

- Need to harness Protocol commonalities
- Need to make a standard verification environment across the industry
- Full usage of Reusability
- Reduced Repetition using templatized approach
- Verification developers get a head start: No need to code from scratch
- Consistency across all the verification blocks in SoC