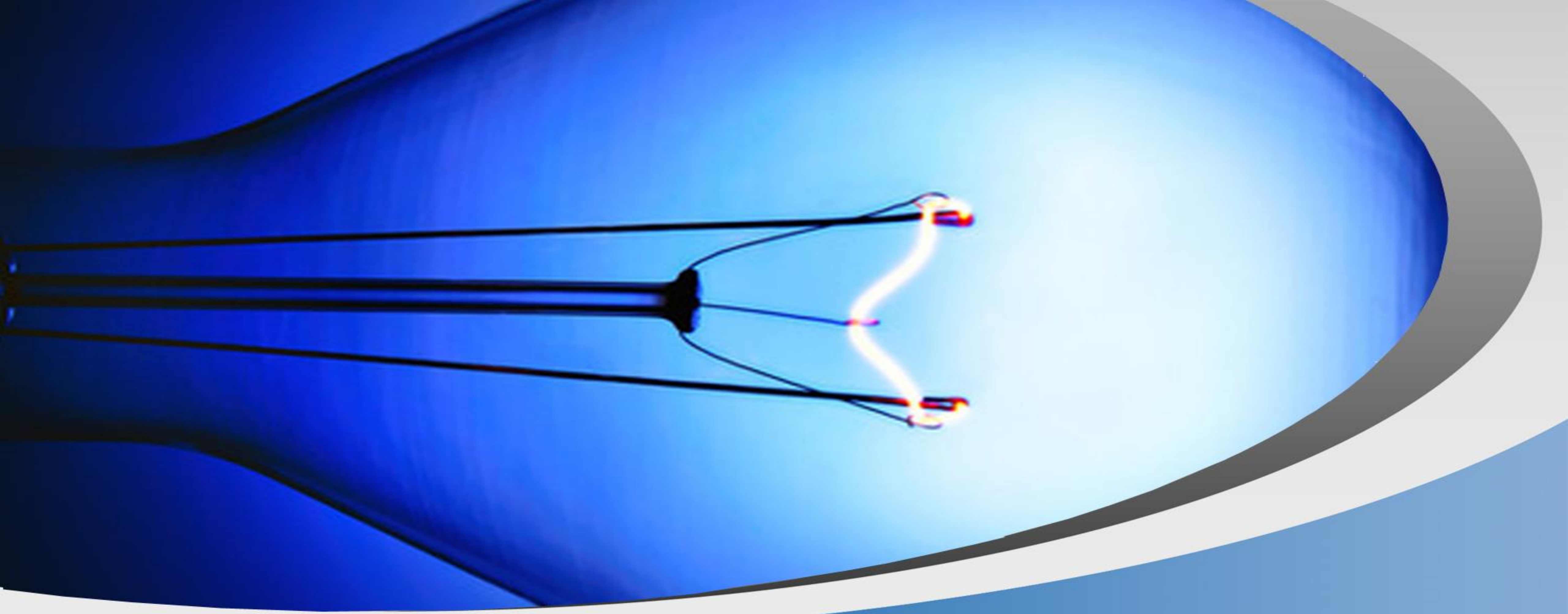


SystemC Synthesis Workgroup Update

Frederic Doucet,	*Qualcomm,*	*SWG chair*
Mike Meredith,	*Cadence Design Systems,*	*SWG member*
Stuart Swan,	*Siemens EDA*	*SWG member*

May 30th 2024

accellera
SYSTEMS INITIATIVE

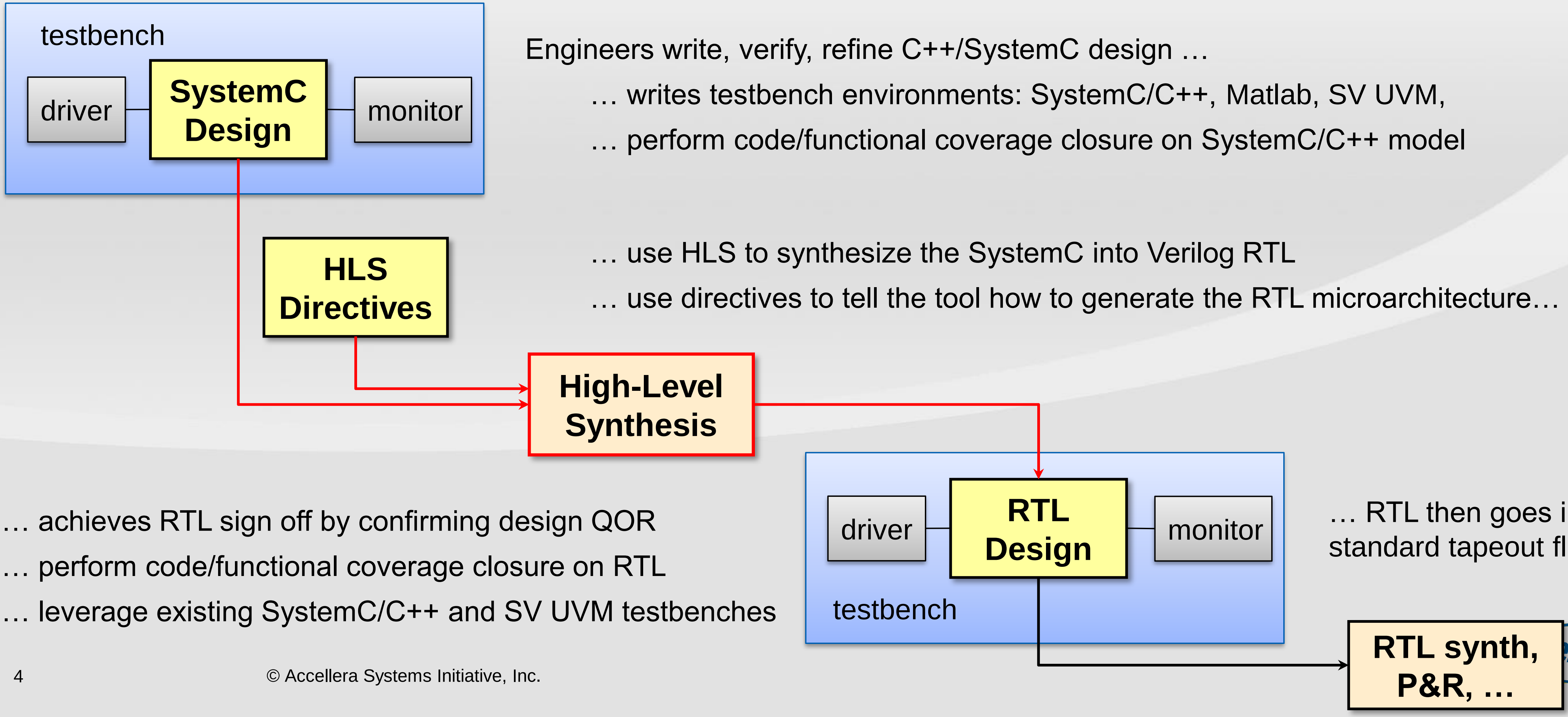
Agenda

▪ Workgroup introduction and overview of SystemC-based HLS	(Fred)
▪ Content of Current SWG Standard 1.4.7	(Mike)
▪ Overview of important items for standardization	(Stuart)
▪ Summary and Call for participation	(Fred)

SystemC Synthesis Workgroup

- **Workgroup objectives:**
  1. Define synthesizable subset and consistent modeling style for SystemC HLS
  2. Streamline SystemC HLS syntax and semantics to
     a) achieve interoperability between HLS/HLV tool,
     b) help standardize HLS/HLV design flows and trainings to be tool independent, and
- **Current public standard: 1.4.7**
  - Workgroup re-starting with focus on new items
- **Standardization process**
  1. Members list, discuss and agree on candidate items for standardization
  2. Design Objective Document (DOD) documents work items for standardization
  3. Open call for contributions from workgroup members for DOD work items
  4. WG members to submit contributions for specific item
     a) WG discusses contribution, works with contributor on any requested changes, and
     b) votes for inclusion in standards document and collateral (PoC / test code etc)

Typical SystemC High-level Synthesis Flow



Engineers write, verify, refine C++/SystemC design …

… writes testbench environments: SystemC/C++, Matlab, SV UVM,

… perform code/functional coverage closure on SystemC/C++ model

… use HLS to synthesize the SystemC into Verilog RTL

… use directives to tell the tool how to generate the RTL microarchitecture…

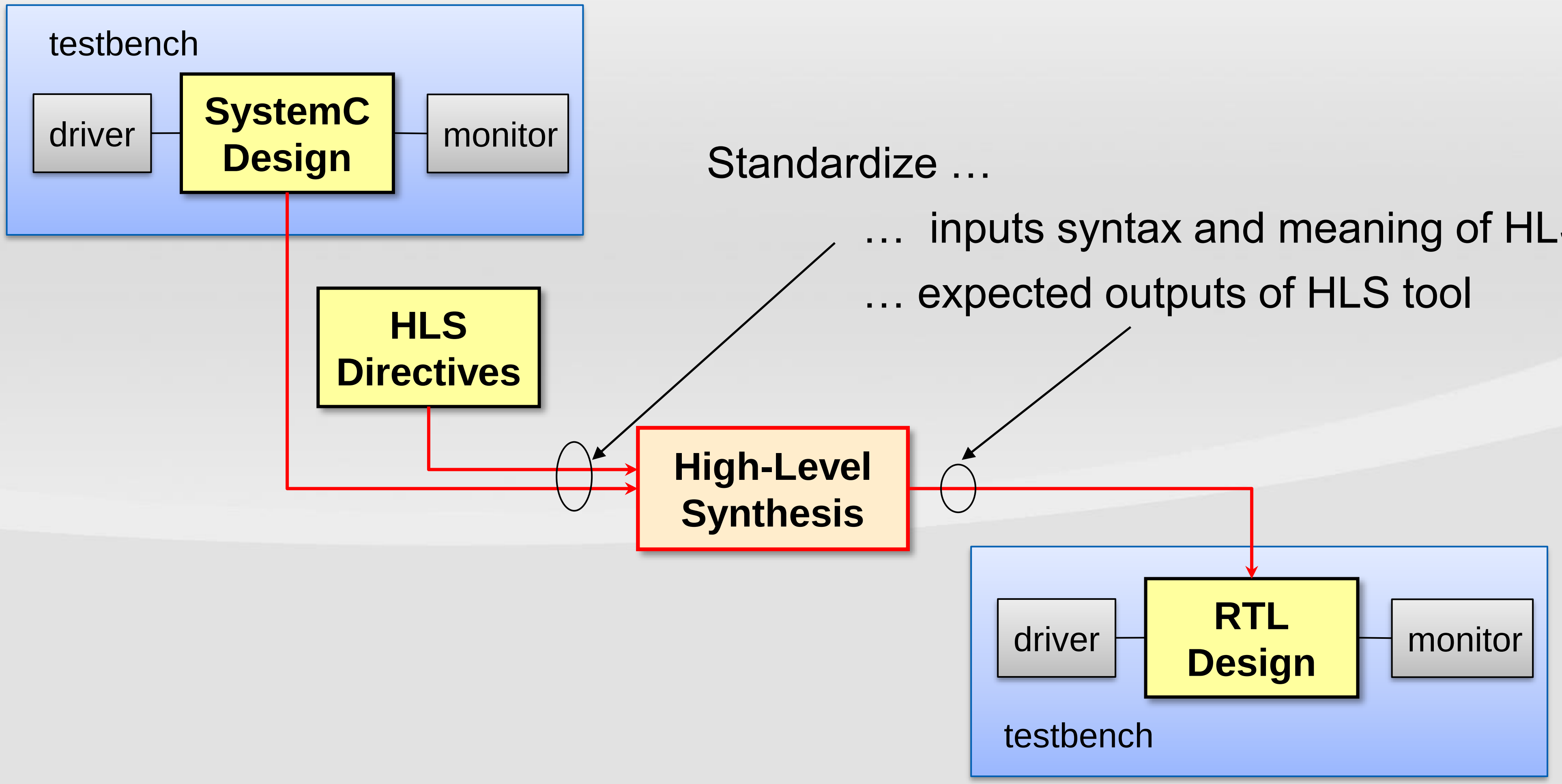
… achieves RTL sign off by confirming design QOR

… perform code/functional coverage closure on RTL

… leverage existing SystemC/C++ and SV UVM testbenches

… RTL then goes into standard tapeout flows

Focus of SystemC Synthesis Standard Workgroup



Standardize …

… inputs syntax and meaning of HLS tool and

… expected outputs of HLS tool

Agenda

- Workgroup introduction and overview of SystemC-based HLS (Fred)
- ➢ ***Content of Current SWG Standard 1.4.7*** ***(Mike)***
- Overview of important items for standardization (Stuart)
- Sumary and Call for participation (Fred)

Scope of Current SystemC Synthesizable Subset Standard

- **Current standard is version 1.4.7**
- **Objectives for the 1.4.7 standard**
  - Define a meaningful minimum subset
    - Establish a baseline for transportability of code between HLS tools
    - Leave open the option for vendors to implement larger subsets and still be compliant
    - Include useful C++ semantics, if they can be known statically – eg templates
- **Covers behavioral model in SystemC for synthesis**
- **Covers RTL model in SystemC for synthesis**

Elements of the 1.4.7 standard

SystemC Elements

- Modules
- Processes
  - SC_CTHREAD
  - SC_THREAD
  - SC_METHOD
- Reset
- Signals, ports, exports
- SystemC datatypes

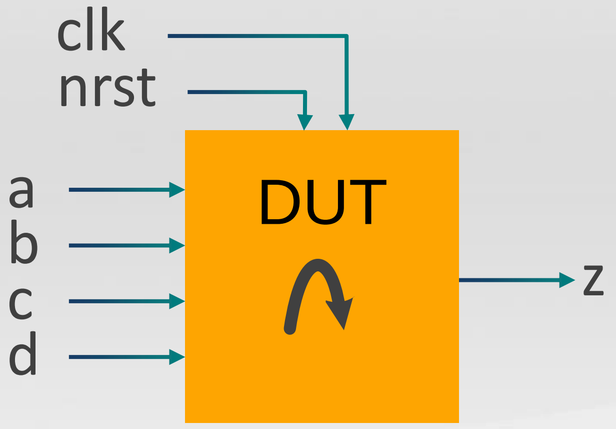
C++ Elements

- C++ datatypes
- Expressions
- Functions
- Statements
- Namespaces
- Classes
- Overloading
- Templates

SystemC Synthesizable Subset Standard 1.4.7 Hardware structure

=> Syntax to provide hardware-meaning to C++

- Modules
- Signal ports
- Signal connections
- Processes
- Wait statements

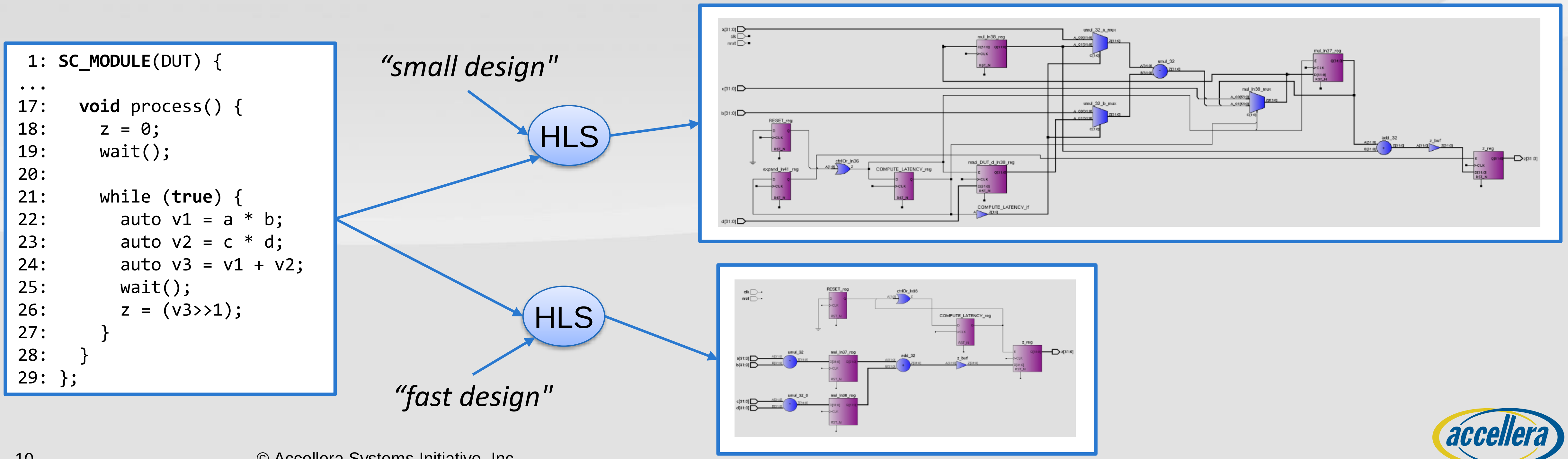


HLS tool strictly follows the syntax to transform SystemC into Verilog

```
SC_MODULE(DUT) {
  sc_in <bool> SC_NAMED(clk);
  sc_in <bool> SC_NAMED(nrst);

  sc_in <sc_int<12>>  SC_NAMED(a);
  sc_in <sc_int<12>>  SC_NAMED(b);
  sc_in <sc_int<12>>  SC_NAMED(c);
  sc_in <sc_int<12>>  SC_NAMED(d);
  sc_out<sc_int<24>>  SC_NAMED(z);

  SC_CTOR(DUT) {
    SC_THREAD(proc);
    sensitive << clk.pos();
    reset_signal_is(rst_n, false);
  }

  void process() {
    z = 0;
    wait();

    while (true) {
      auto v1 = a * b;
      auto v2 = c * d;
      auto v3 = v1 + v2;
      wait();
      z = (v3>>1);
    }
  }
};
```

Not in current Standard: HLS Directives

- **Instructs the HLS tool how to generate the micro-architecture**
  - Impacts latency, resources, registers, multiplexers, FSM, etc.
  - They are as important to the design as the source code
- **Subject of standardization activities**



```
SC_MODULE(DUT) {
...
  void process() {
    z = 0;
    wait();

    while (true) {
      auto v1 = a * b;
      auto v2 = c * d;
      auto v3 = v1 + v2;
      wait();
      z = (v3>>1);
    }
  }
};
```

Agenda

▪ Workgroup introduction and overview of SystemC-based HLS	(Fred)
▪ Content of Current SWG Standard 1.4.7	(Mike)
➢ ***Overview of important items for standardization***	***(Stuart)***
▪ Summary and Call for participation	(Fred)

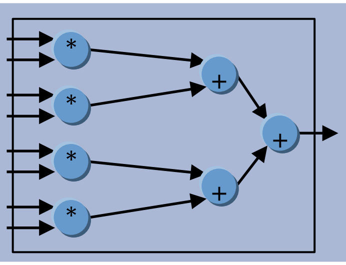
Key HLS Directives

Some C++ constructs can interpreted in a variety of ways when synthesizing SystemC into RTL

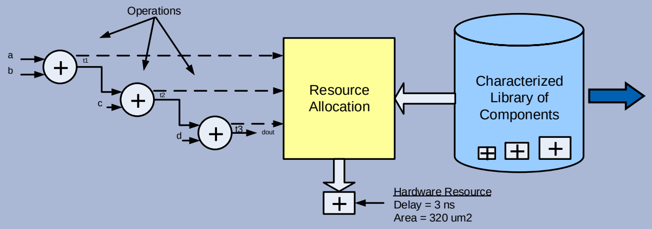
- **Array implementation**
  - Memory, register file, registers
- **Loop implementation**
  - Unrolling, pipelining, merging, etc.

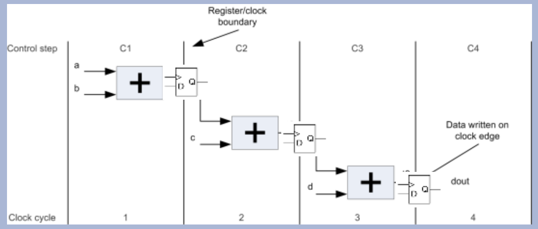
```
sc_int<12> data[N];
sc_int<12> coeffs[N];
...
MAC_H: for (int i=0;i<N;i++) {
   acc += data[i] * coeffs[i];
}
```

- **Custom resources**
  - Larger resources that can be used by the scheduler
    - Arithmetic optimizations cluster for coarser grain resource sharing
    - Multicycle operations



- **Scheduling**
  - Latency/throughput constraints:
    - where/when to add states
  - Protocol constraints :
    - Where/how to access I/O





Channels and Memory Architecture

- Channel-based I/O can be implemented as SystemC IP
  - Can implement TLM and PIN-level semantics
  - Presents TLM API to the design
- Memories can be modeled as
  - SystemC IP
    - Shown here
  - C++ Arrays
- Standardization of channels and memory libraries and IP is a topic of planned work
  - Exact syntax specifics will be discussed

```
SC_MODULE(MatrixMultiply) {
  sc_in<bool> SC_NAMED(clk) ;
  sc_in<bool> SC_NAMED(rstn);

  Connections::In <sc_int<8>>     SC_NAMED(A);
  Connections::In <sc_int<8>>     SC_NAMED(B);
  Connections::Out<sc_int<8+8+3>> SC_NAMED(C);
  Connections::SyncChannel        SC_NAMED(sync);

  shared_bank_array<sc_int<8>, 8, 8*2>             SC_NAMED(B_transpose);
  Connections::Combinational<array_t<sc_int<8>,8>> SC_NAMED(A_row);

  SC_CTOR(matrixMultiply) {
    SC_THREAD(pack_A);
    sensitive << clk.pos();
    async_reset_signal_is(rstn, false);

    SC_THREAD(transpose);
    sensitive << clk.pos();
    async_reset_signal_is(rstn, false);

    SC_THREAD(mac);
    sensitive << clk.pos ();
    async_reset_signal_is(rstn, false);
  }
...
```

TLM Channels

Memory Models

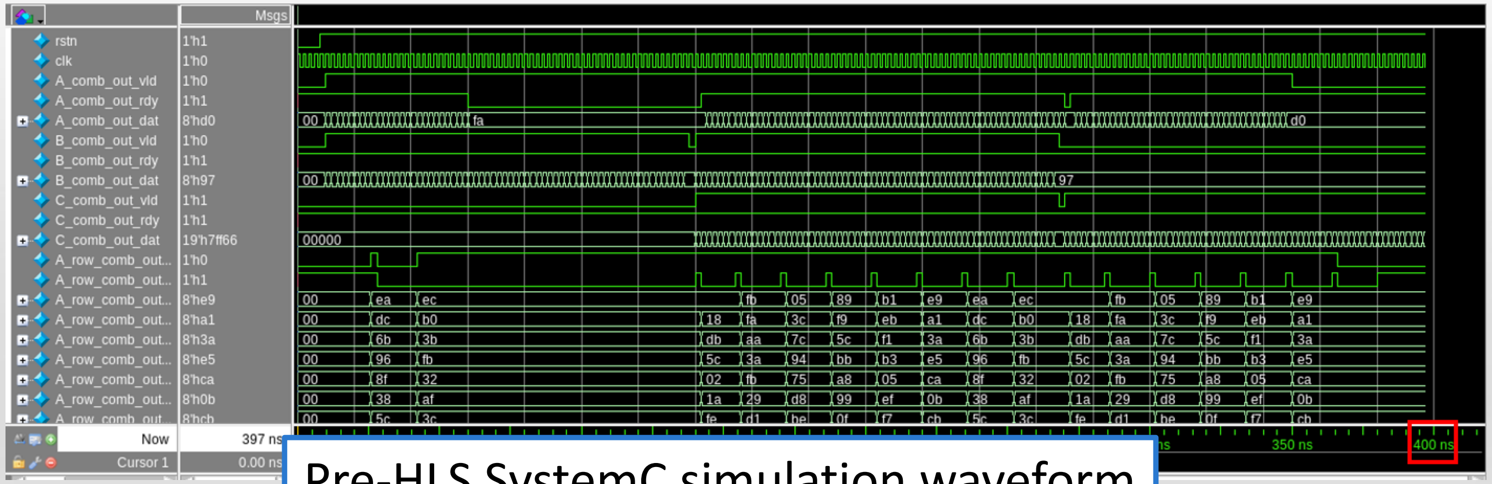
Directives for Loop Unrolling and Pipelining

```
void mac() {
  C.Reset();
  A_row.ResetRead();
  sync.reset_sync_in();
  bool ping_pong = false;
  wait();

  while (1) {
    sc_int<8+8+3> acc = 0;
    sync.sync_in();
    #pragma hls_pipeline_init_interval 1
    #pragma pipeline_stall_mode flush
    ROW:for(int i = 0; i < 8; i++) {
      auto A_dat = A_row.Pop();
      COL:for(int j = 0; j < 8; j++) {
        acc = 0;
        #pragma hls unroll yes
        MAC:for(int k = 0; k < 8; k++) {
          auto B_dat = B_transpose[K][j + 8*ping_pong];
          acc += A_dat.data[k] * B_dat;
          C.Push(acc);
          ping_pong = !ping_pong;
        }
      }
    }
  }
}
```

HLS Directives specified using inlined pragmas

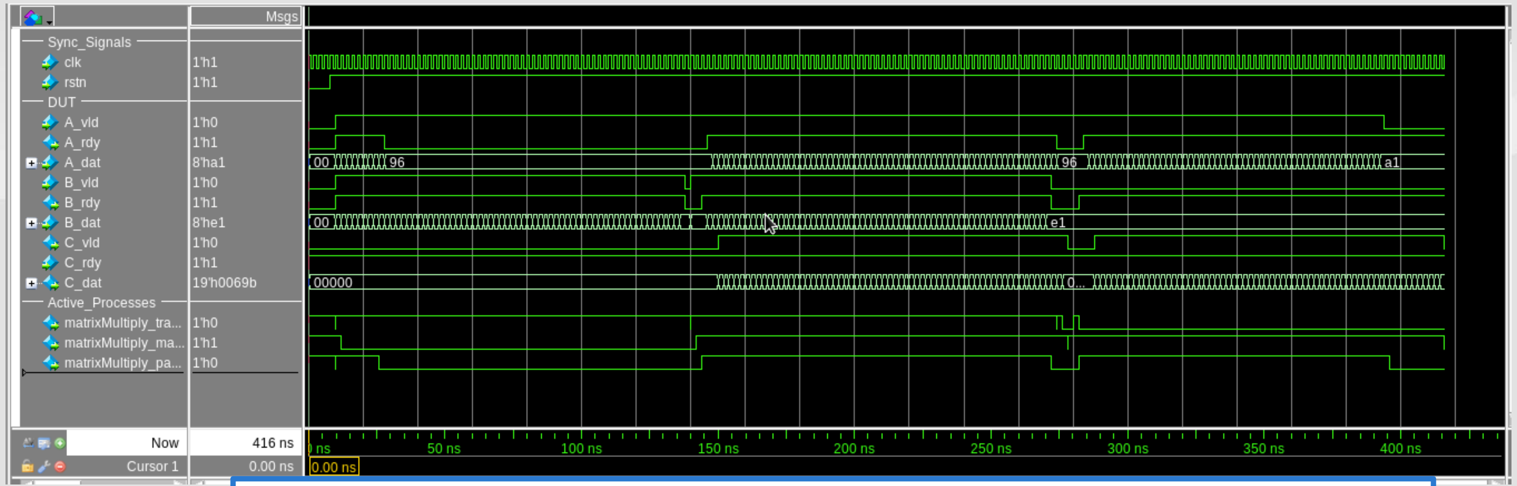
- **Many forms of directives can be considered**
  - Pragmas or other annotations in the source code
    - Shown here
  - External, e.g. Tcl control
- **Standardizing one or more approaches is a topic of planned work**

Loop Pipelining: Throughput and Latency



	Msgs
rstn	1'h1
clk	1'h0
A_comb_out_vld	1'h0
A_comb_out_rdy	1'h1
A_comb_out_dat	8'hd0
B_comb_out_vld	1'h0
B_comb_out_rdy	1'h1
B_comb_out_dat	8'h97
C_comb_out_vld	1'h1
C_comb_out_rdy	1'h1
C_comb_out_dat	19'h7ff66
A_row_comb_out...	1'h0
A_row_comb_out...	1'h1
A_row_comb_out...	8'he9
A_row_comb_out...	8'ha1
A_row_comb_out...	8'h3a
A_row_comb_out...	8'he5
A_row_comb_out...	8'hca
A_row_comb_out...	8'h0b
A_row_comb_out...	8'hcb
Now	397 ns
Cursor 1	0.00 ns

Pre-HLS SystemC simulation waveform



	Msgs
Sync_Signals	
clk	1'h1
rstn	1'h1
DUT	
A_vld	1'h0
A_rdy	1'h1
A_dat	8'ha1
B_vld	1'h0
B_rdy	1'h1
B_dat	8'he1
C_vld	1'h0
C_rdy	1'h1
C_dat	19'h0069b
Active_Processes	
matrixMultiply_tra...	1'h0
matrixMultiply_ma...	1'h1
matrixMultiply_pa...	1'h0
Now	416 ns
Cursor 1	0.00 ns

Generated RTL simulation waveform w/ pipeline

- **HLS users need the ability to do accurate performance analysis on the pre-HLS model**
  - Much better than on RTL
- **HLS tool will increase design latency when pipelining a loop, for a specified throughput**
  - … to meet timing or reduce area
- **Use of channels with handshake makes design robust to such changes**
  - *These channels need to be standardized*

accellera SYSTEMS INITIATIVE

Agenda

▪ Workgroup introduction and overview of SystemC-based HLS	(Fred)
▪ Content of Current SWG Standard 1.4.7	(Mike)
▪ Overview of important items for standardization	(Stuart)
➢ ***Summary and Call for participation***	***(Fred)***

HLS Standardization Topics

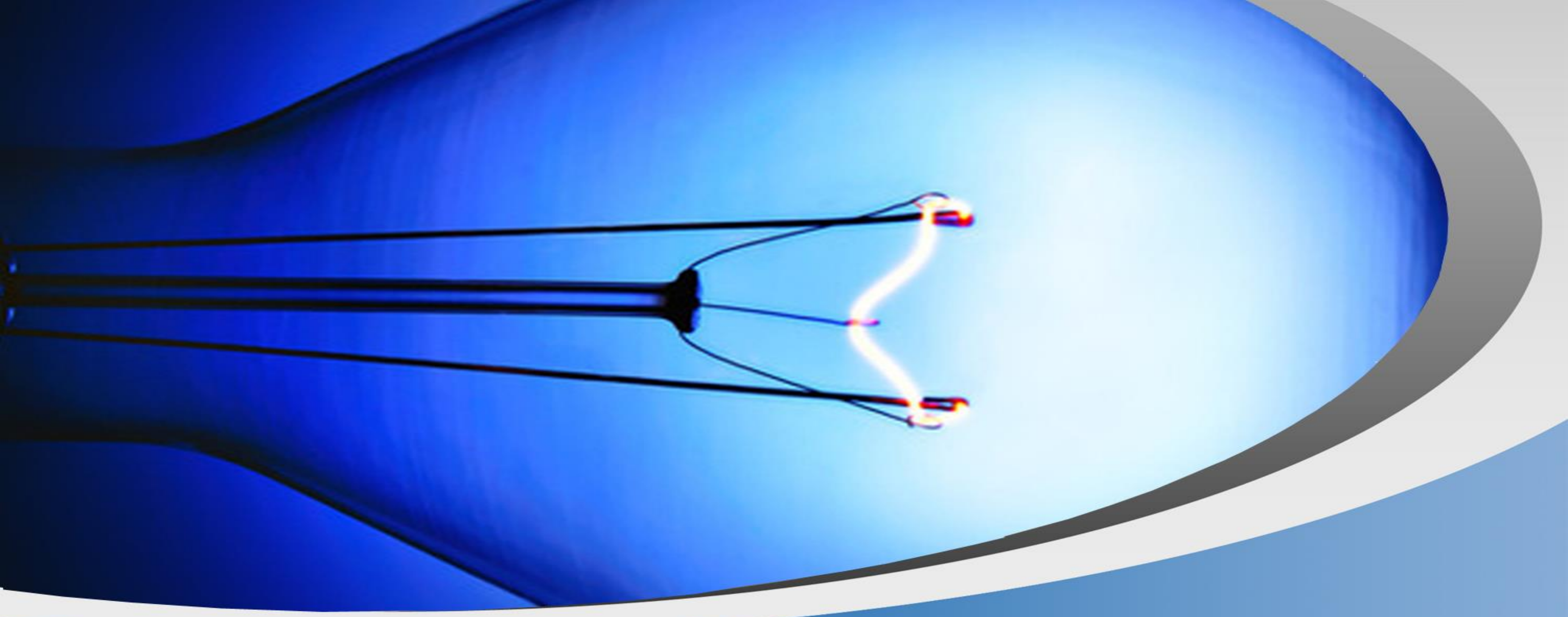
1. **Syntax and semantic interpretation**
   - Unambiguous interpretation SystemC syntax for synthesis
     - Structure and Interfaces: modules, ports
     - Behavior descriptions: methods, threads and wait()
     - Scheduling rules (where states are added, how are I/O kept together etc.)
   - Expected syntax for HLS-generated output
2. **HLS Directives**
   - loops (pipelining, unrolling etc.), arrays and memories, input and output scheduling, specification of latency and throughput constraints, etc
3. **Communication interfaces**
   - Channels (point-to-point, fifos, message passing) and memory-like constructs (banks, sram)

HLS Standardization Topics

4. **Modern language constructs**
   - Which C++17/C++20 language constructs to be supported for HLS
   - Which standard library classes (std::*) to be supported for HLS
5. **Synthesizable Data types**
   - Float, complex, arrays, fixed-point, composite data types support (arrays, structs, complex data type)
     - How to we synthesize arrays, structs and other abstract collections to different storage types
   - Standalone header files for all synthesizable datatypes.
6. **SOC/infrastructure libraries and common design basic blocks**
7. **Proof-of-concept implementation and examples**
   - To drive the success, discuss a code tarball documenting the standard with:
     - Regression suite : qualify the syntax works the way as expected, and
     - Set of examples: more educational, demonstrating the standard modeling style

Call for participation

- **Join the workgroup !**
  - https://www.accellera.org/activities/working-groups/systemc-synthesis
  - https://workspace.accellera.org/wg/SWG/dashboard
- **Meeting schedule:**
  - On the 2nd Wednesday of every month
  - 8:00 AM - 9:30 AM PDT / 17:00 - 18:30 CET



Thank you for listening!