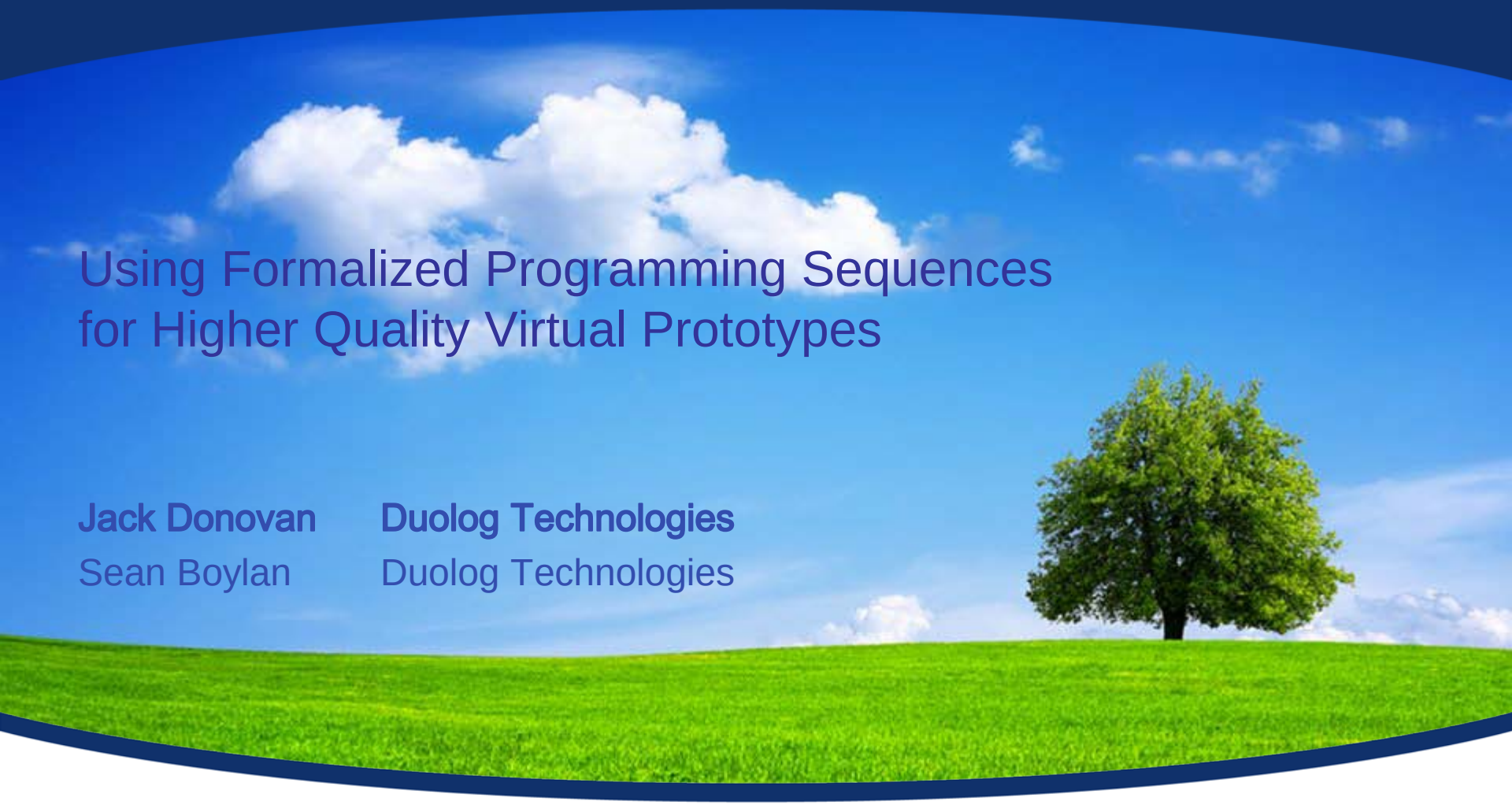
Using Formalized Programming Sequences for Higher Quality Virtual Prototypes



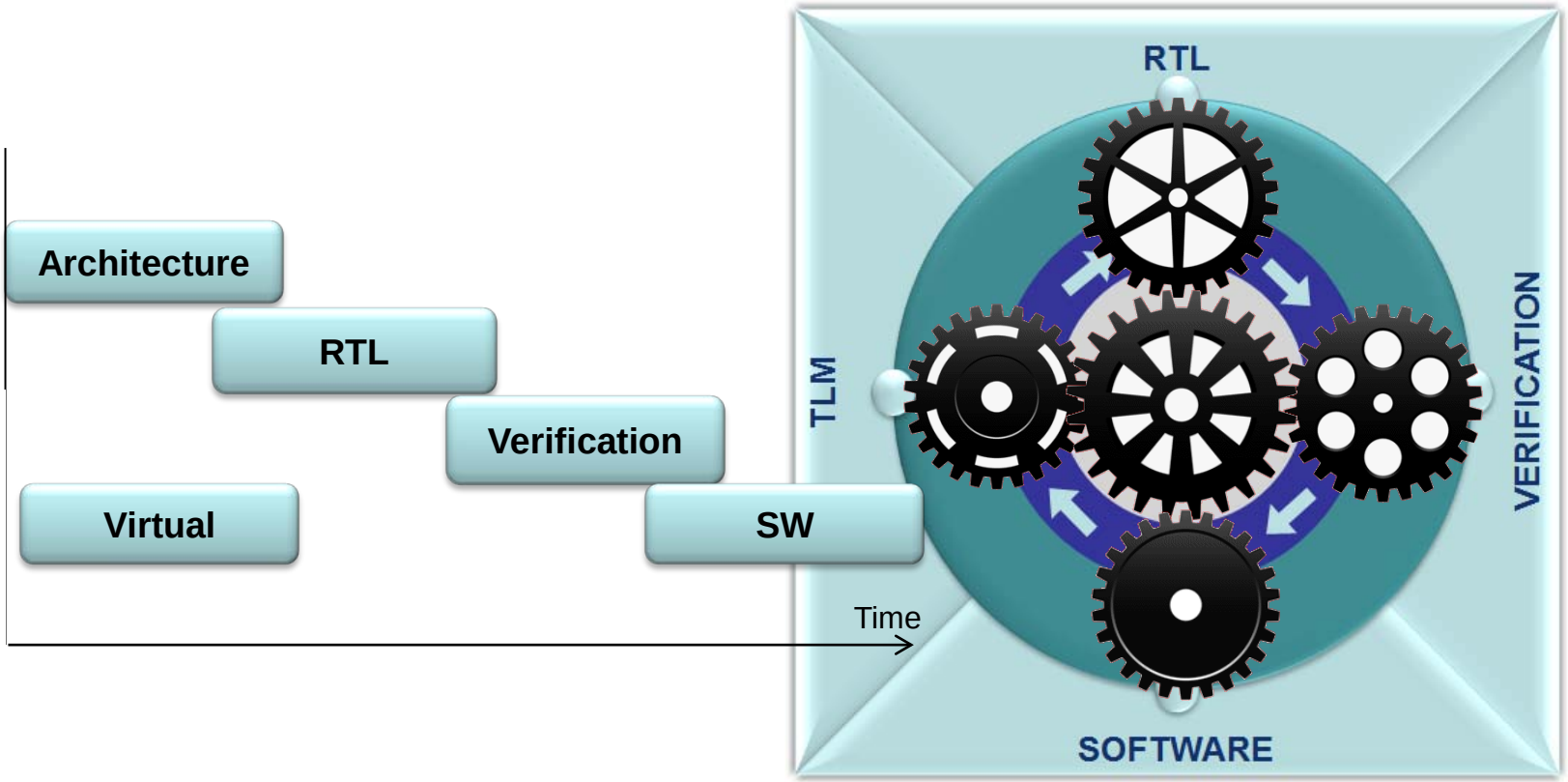
Jack Donovan **Duolog Technologies**
Sean Boylan Duolog Technologies

duolog technologies

Outline

- Motivation - VSP Quality
- Programming Sequences
- Applying Sequences
- Tools for Sequences
- Conclusions

SoC Design – The New Reality



New Reality

SoC design is becoming highly concurrent and requires precise synchronization between teams

duolog technologies

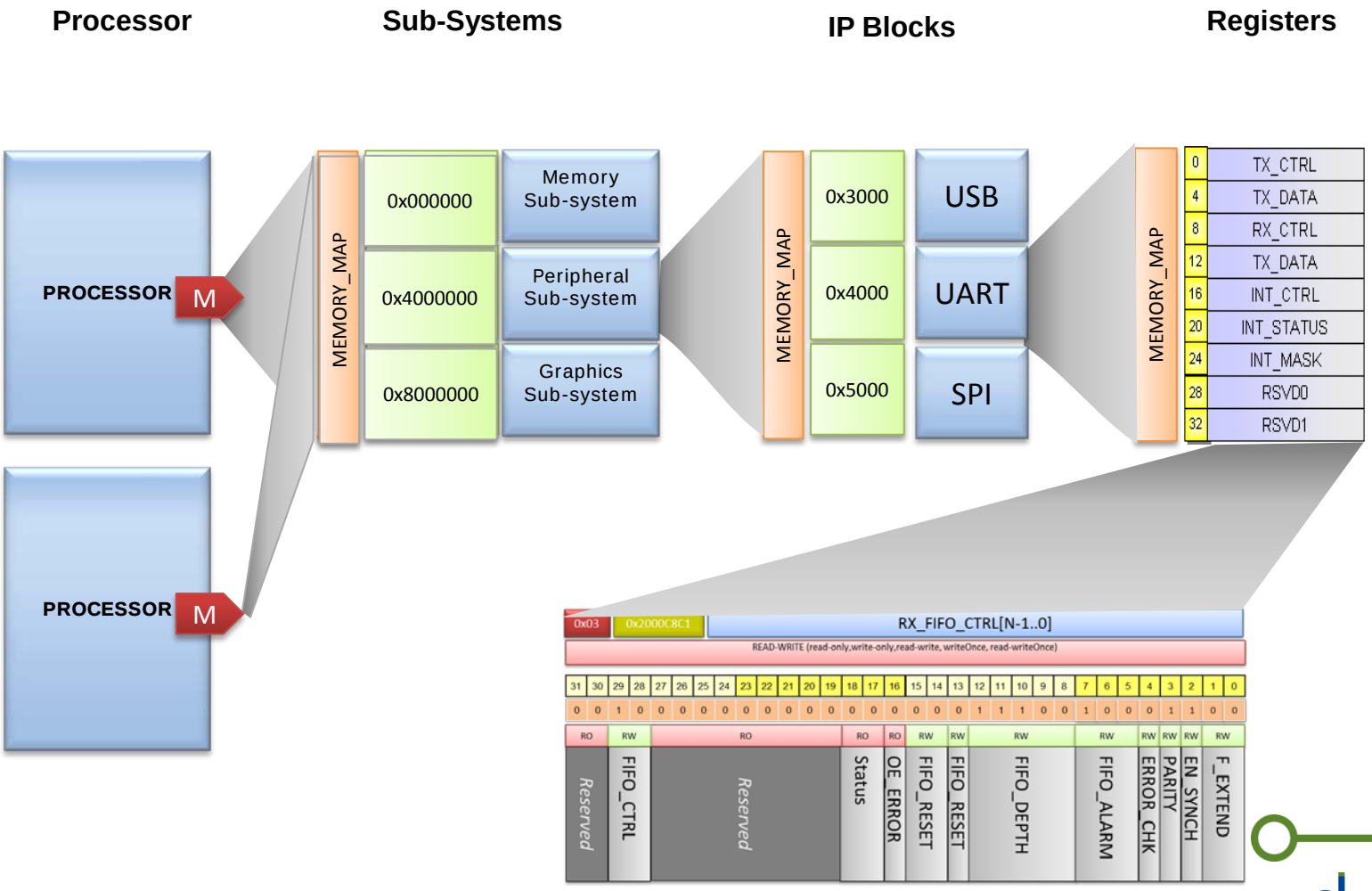
ESL Motivation

- Virtual Prototypes are now “main stream”
  - No Longer just used by “early adopters”
  - Part of Project Plan … Main Stream
- Main Stream -> need for higher quality
  - Debug software not Virtual Prototype integration issues
  - Busted Virtual Prototypes can block whole departments of SW engineers
- Virtual Prototypes are more complex
  - 1000’s of IP’s
  - Power sub-system, clocks sub-system, SoC modes

Outline

- Motivation - VSP Quality
- **Programming Sequences**
- Applying Sequences
- Tool Requirements for Sequences
- Conclusions

HW/SW Interface



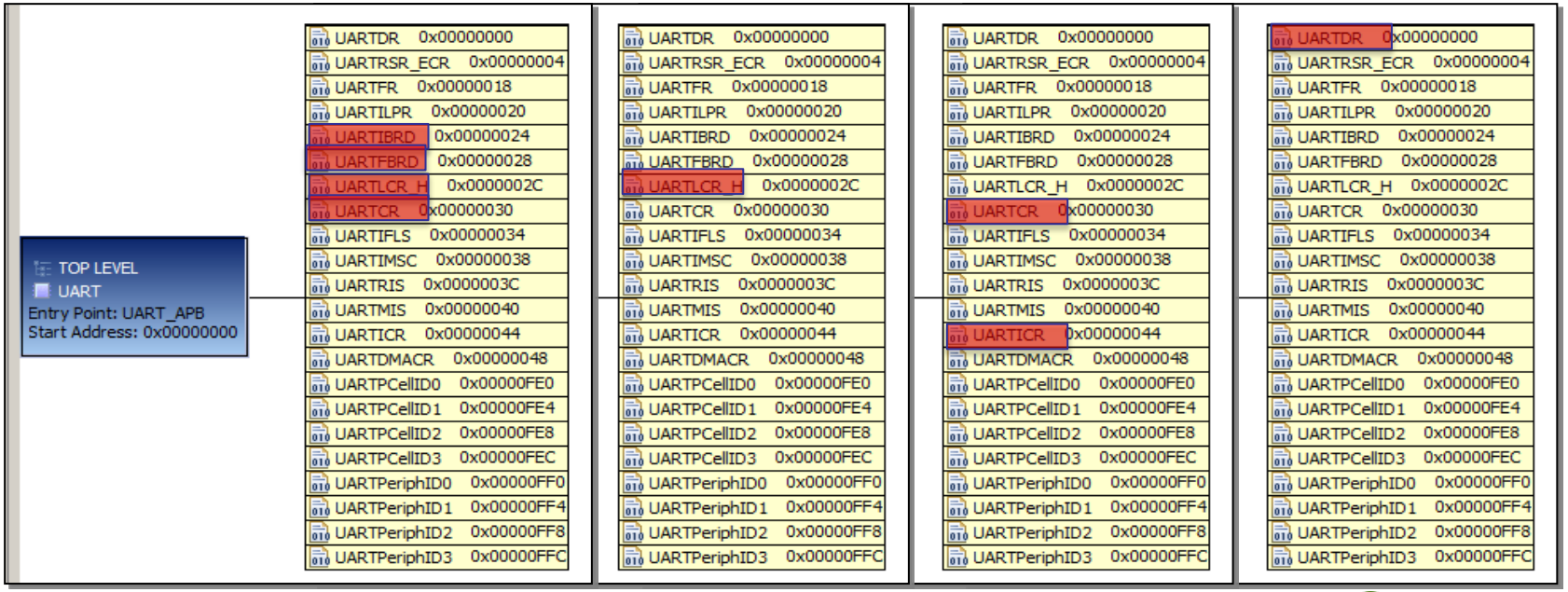
duolog technologies

UART - Programming Sequences

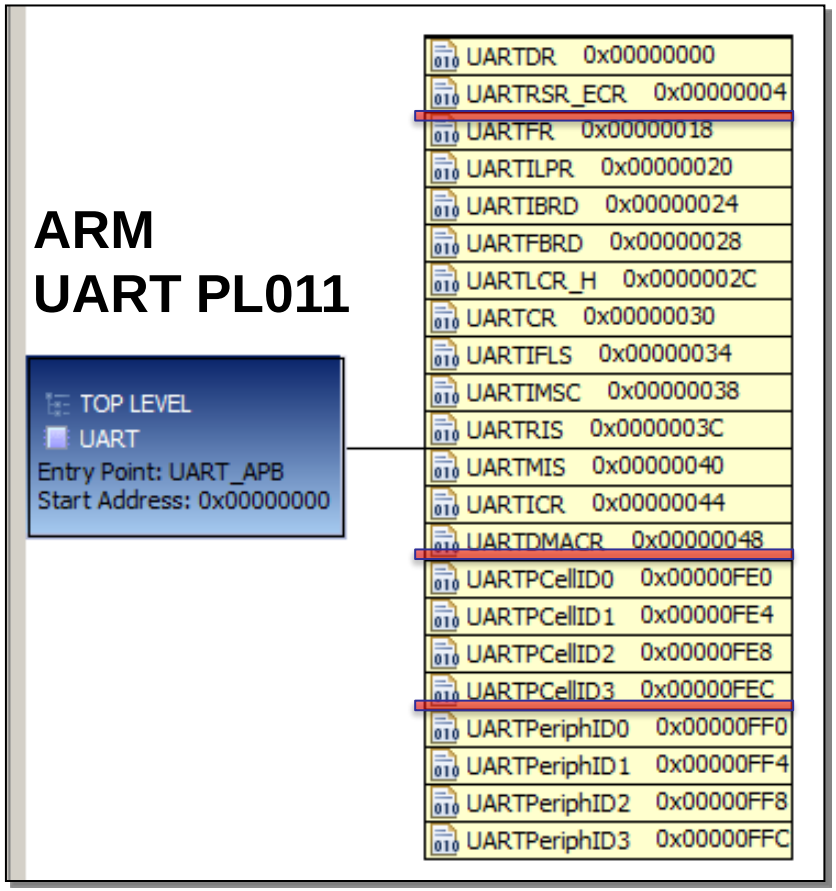
ARM UART PL011

HW/SW spec defines how to program, verify and integrate IP Block

Set the Baud Rate | **Set up FIFO** | **Enable Transmit** | **Write Data**



Constraints



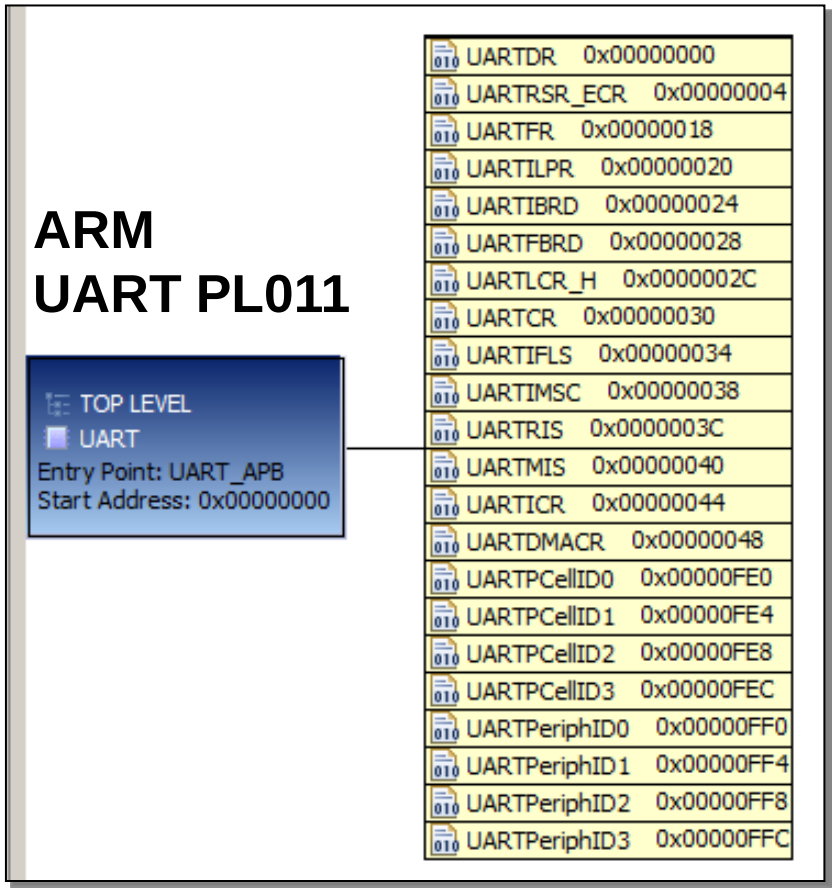
Global Constraints

Reserved bits – must not be used

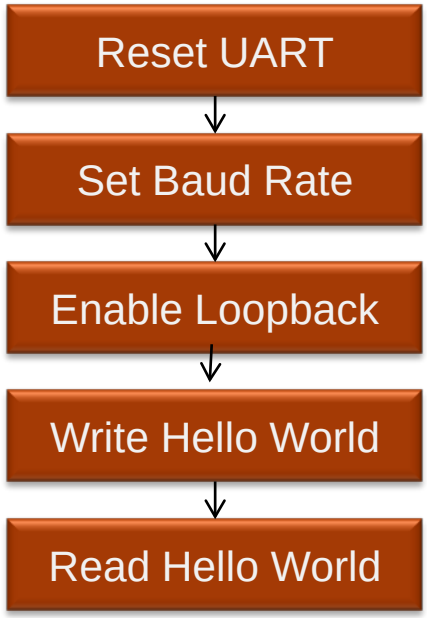
- locations at offsets 0x008 through 0x014, 0x01C are reserved and must not be
- accessed
- locations at offsets 0x04C through 0x07C are reserved for possible future extensions
- locations at offsets 0x080 through 0x08C are reserved for test purposes
- locations at offsets 0x90 through 0xFCC are reserved for future test purposes
- location at offsets 0xFD0 through 0xFDC are used for future identification registers
- location at offsets 0xFE0 through 0xFFC are used for identification registers

duolog technologies

Hierarchical Sequence : Test UART



Test UART



duolog technologies

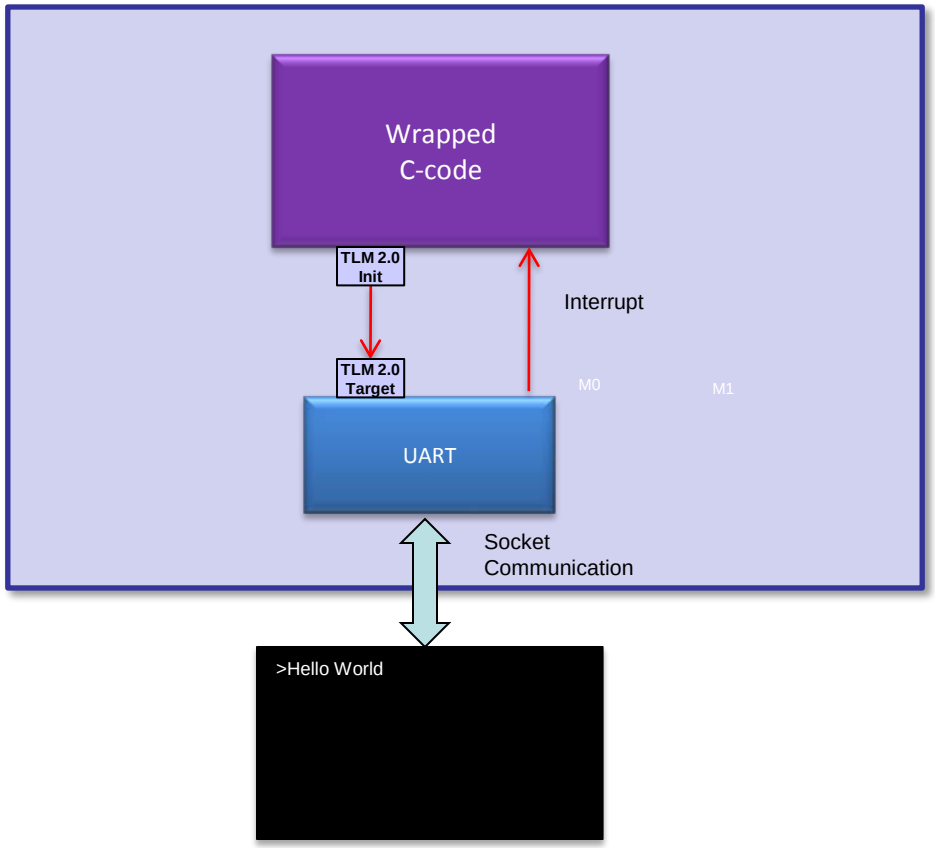
Outline

- Motivation - VSP Quality
- Programming Sequences
- **Applying Sequences**
- Tool Requirements for Sequences
- Conclusions

Virtual Prototype Qualification

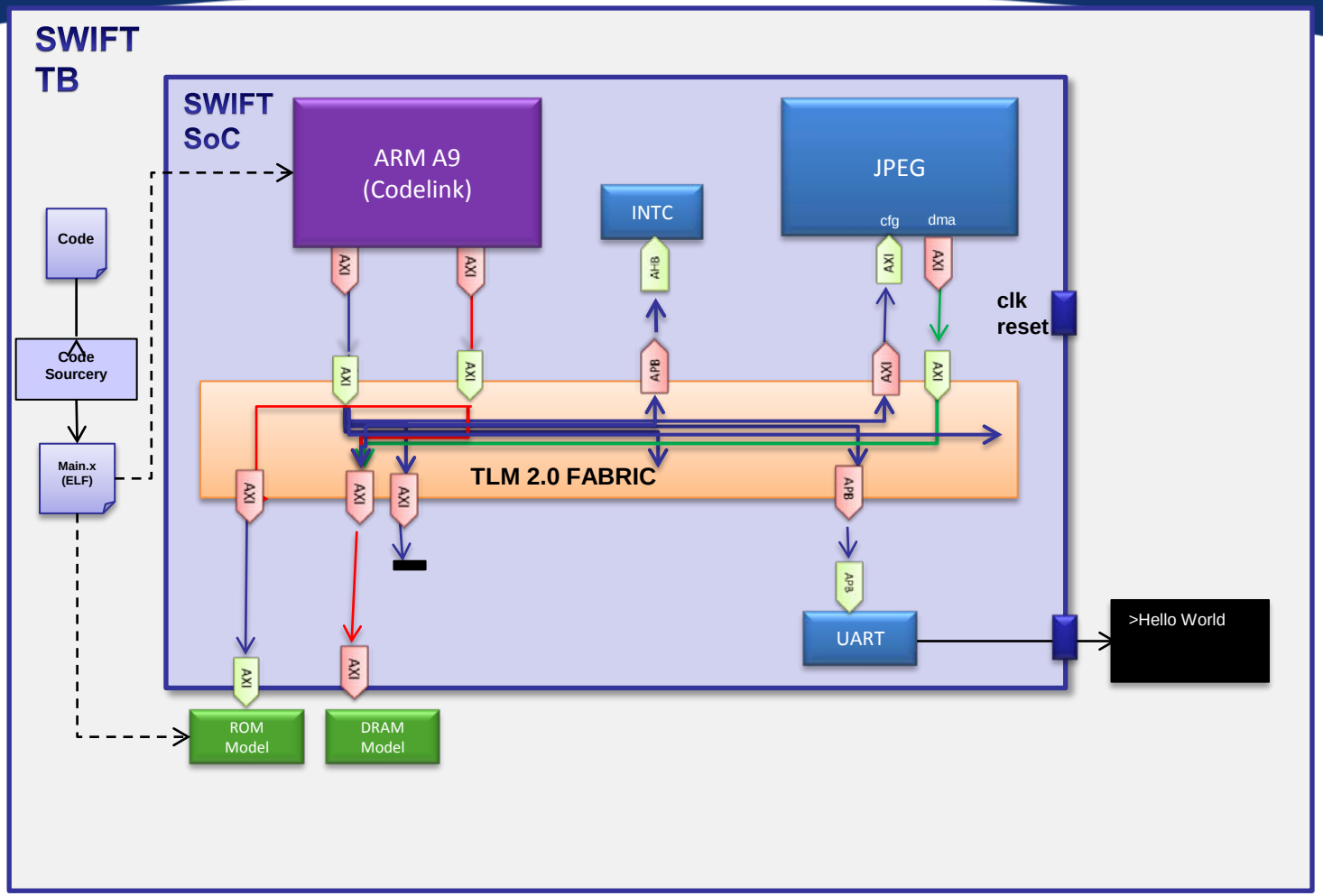
- Goal: Qualify/Assure that basic functionality is working after incrementally adding functionality
- In reality this is complex
  - Power Control
  - Clock Control
  - Mode Control – IO Mux'ing
  - Functional Modes
- Requires multi-block/function sequences

Unit Test



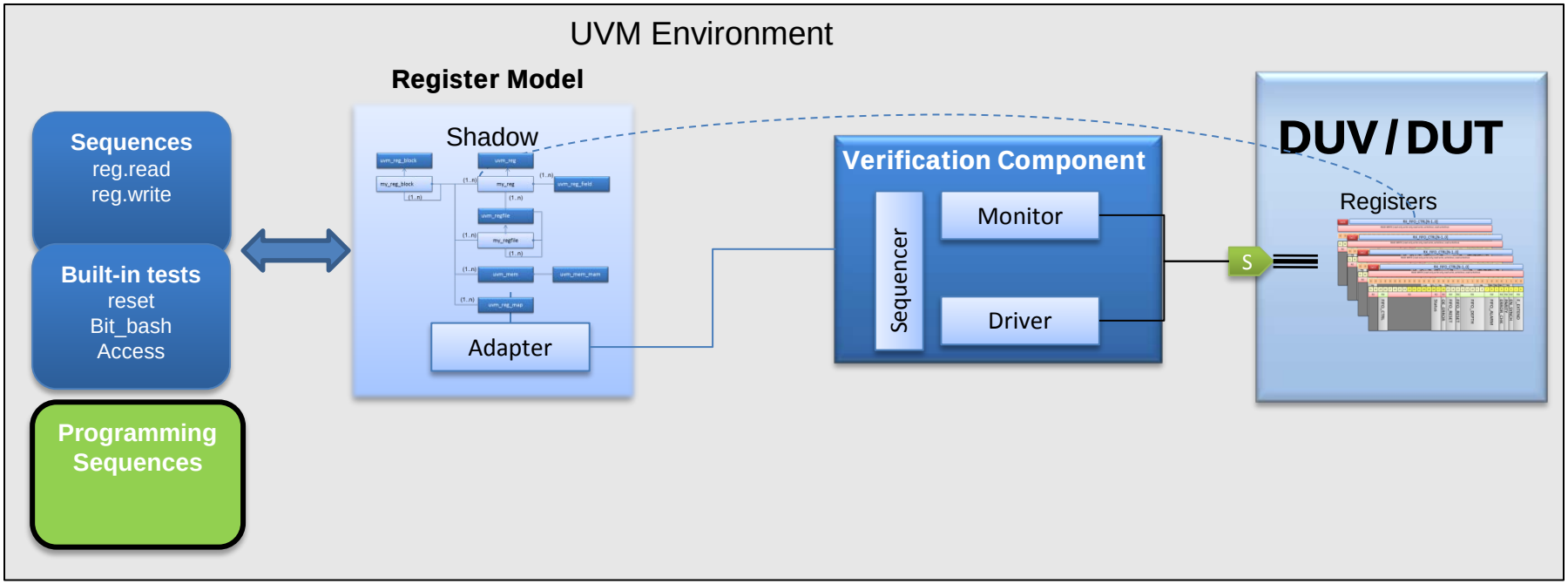
duolog
technologies

Virtual Proto Incremental Changes



duolog technologies

RTL/TLM Unit Test – UVM Environment



duolog technologies

Outline

- Motivation - VSP Quality
- Programming Sequences
- Applying Sequences
- **Tool Requirements for Sequences**
- Conclusions

Sequence Capture

- Captured via a Dedicated Editor
  - Support for auto-completion, syntax checking, code templates and a range of markers
- Sequence Language Easy to Learn and Understand
  - A subset of ‘C’ with some sequence specific extensions
  - Constraint syntax closely aligned to SystemVerilog
- Leverages Existing Formats for Register Descriptions
  - Compatible with Bitwise format, IEEE1685 (IP-XACT), others

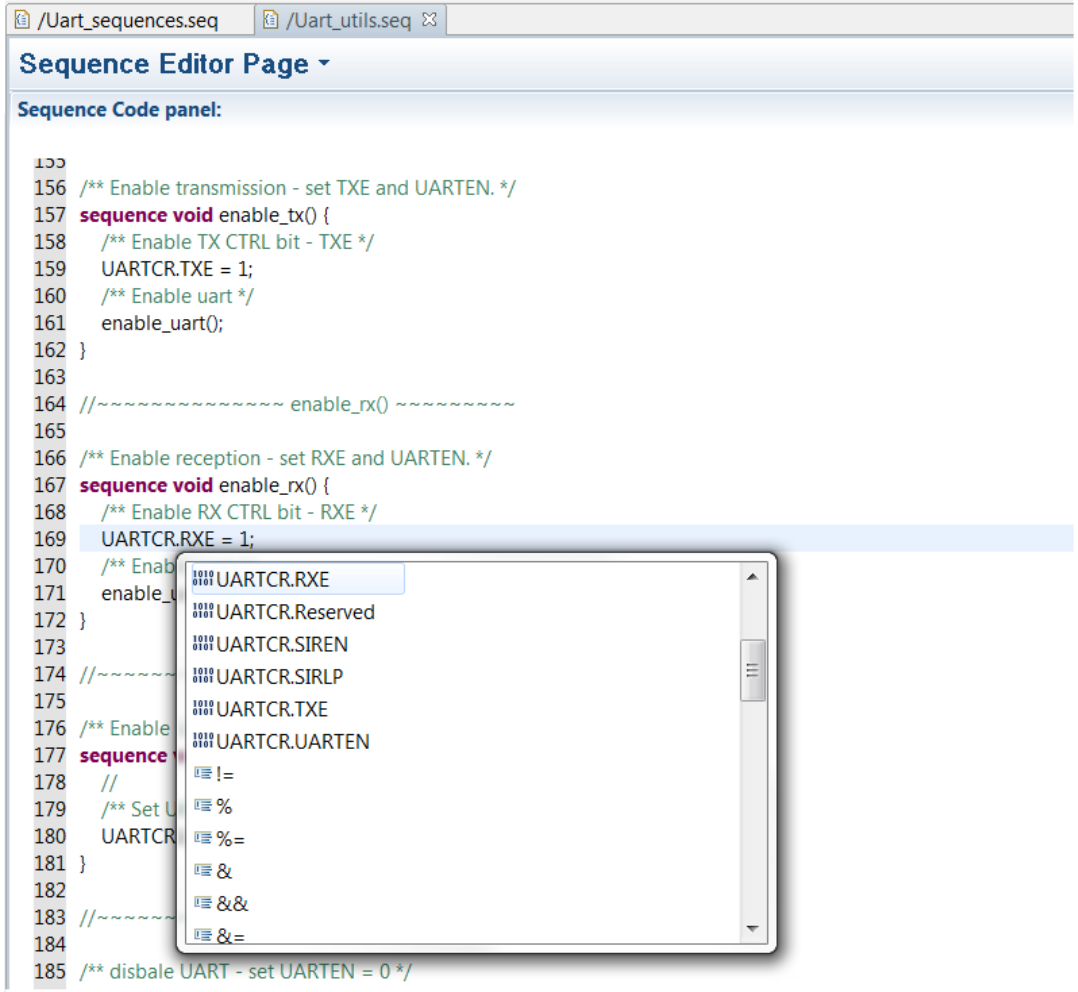
bitwise SOCRATES

IP-XACT™

- Coherency Checking
  - Assures compatibility of the sequence and the Register Descriptions for all blocks in the sequence

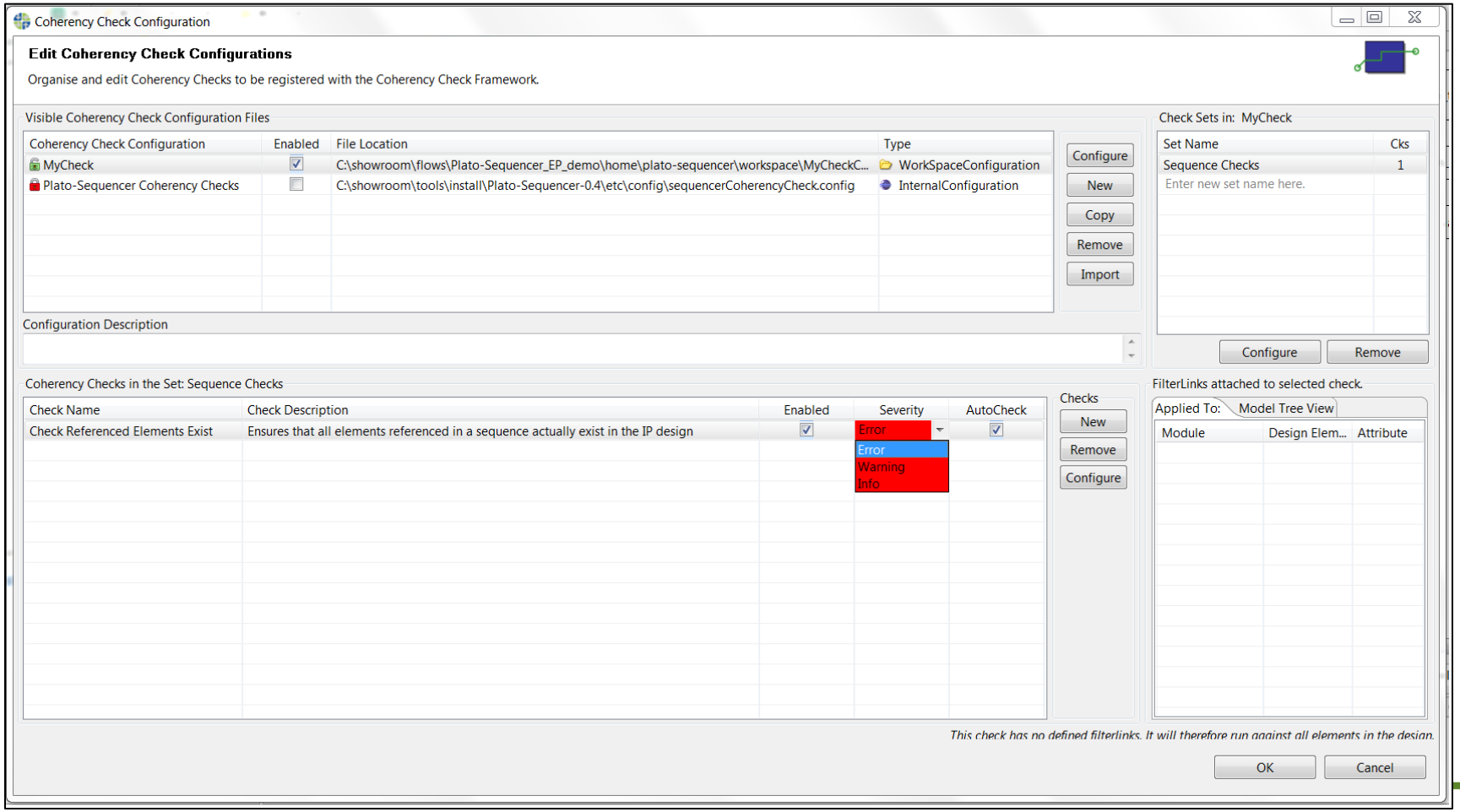
duolog technologies

Language and an Editor



duolog technologies

Coherency Checker



duolog technologies

2.1.2 Pseudocode

2.1.2.1. sequence void uart_hello_world()

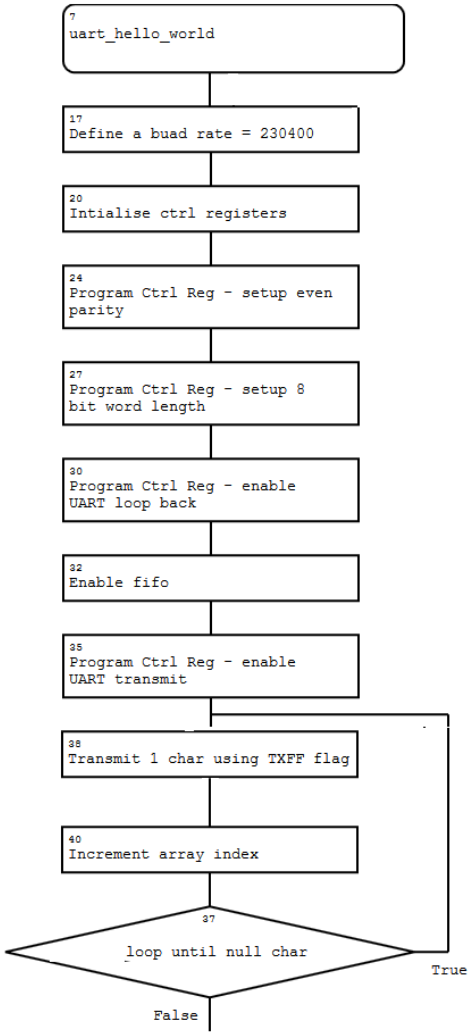
```
int i = 0
int static br = 230400
unsigned int n = 18
char data_buffer[n] = hello world

        setup_baud_rate(br)

        intialise_ctrl_registers_setup()

        setup_fifo_parity_even()

        setup_word_len(wlen8)

        setup_uart_loop_back()

        enable_fifo()

        enable_tx()
```

2.1.2.2. sequence void print_uart_id()

```
int static br = 230400

int part_no

int designer

int rev_no

int configuration

        part_no = uart_periph_id[11:0]

        designer = uart_periph_id[19:12]

        rev_no = uart_periph_id[23:20]

        configuration = uart_periph_id[31:21]

print "UART Peripheral Info: "Part No",  "Designer", "Revision No", "Configuration"
```

2.2 Sequence Flowchart

2.2.1. uart_hello_world()



```
void setup_baud_rate(int br) {

   /** Disable UART1*/
   disable_uart();
   /** Calculate the baud rate divisor */
   float brd = uartclk / 16.0 * br;
   /** Calculate the integer_baud_rate_divisor */
   int brdi = (int)brd;
   /** Calculate the fractional part of the

   baud_rate_divisor */
   float brdf = brd - brdi;
   /** Allow for rounding errors - add 0.5 */
   int brdf_err = (int)brdf * 64.0 + 0.5;
   /** Calculate generated_baud_rate_divisor */
   float gbrd = brdi + brdf_err / 64.0;
   /** Calculate generated_baud_rate */
   int gbr = (int)uartclk / 16.0 * gbrd;
   /** Calculate buad_rate_error */
   float err = br - gbr / gbr * 100;
   /** Add constaints for the integer and fractional baud rate registers */
   {
      void seq_rsa[] = { *UARTIBRD, *UARTFBRD,  NULL };
      seq_verify seq_v = { verify_setup_baud_rate_UARTIBRD_do, UARTIBRD.vft.verify, seq_rsa };
      UARTIBRD.vft.verify = &seq_v;

      {
         void seq_rsa[] = { *UARTIBRD, *UARTFBRD,  NULL };
         seq_verify seq_v = { verify_setup_baud_rate_UARTIBRD_do, UARTFBRD.vft.verify, seq_rsa };
         UARTFBRD.vft.verify = &seq_v;
         {

            write_UARTIBRD_BAUDDIVINT(base + 0x100, brdi);
            write_UARTFBRD_BAUDDIVFRAC(base + 0x200, brdf_err);
         }
         UARTFBRD.vft.verify = UARTFBRD.vft.verify->next;

      }
```

duolog technologies

```
task setup_baud_rate(input int br);

  real brd      = (uartclk)/(16.0 * br);
   int  brdi     = int'(brd);                    // check: casting
   real brdf     = brd - brdi;
   int  brdf_err = int'(brdf * 64 + 0.5);
   real gbrd     = brdi + brdf_err/64;
   int  gbr      = int'((uartclk)/(16 * gbrd));
   real err      = (br - gbr)/gbr * 100;

   disable_uart();

   reg_model.uartIBRD.uartIBRD.write(status, brdi);
   reg_model.uartFBRD.uartFBRD.write(status, brdf_err); // verifiy is missing
endtask : setup_baud_rate

task receive_uart_data(output byte data);  // check parameter
   reg_model.uartDR.DATA.read(status, data);       // verify statement is missing
endtask : receive_uart_data

task transmit_uart_data_txff(input byte data);
   int uartfr_txff;
   reg_model.uartFR.TXFF.read(status, uartfr_txff);
   if(uartfr_txff == 0) begin
      reg_model.uartDR.DATA.write(status, data);
   end
endtask : transmit_uart_data_txff

task transmit_uart_data_busy(input byte data[]);
   // finish this
endtask : transmit_uart_data_busy

task setup_uart_loop_back();
   reg_model.uartCR.SIREN.write(status, 0);
   //reg_model.uartCR.SIRTEST.write(status, 0);
   reg_model.uartCR.LBE.write(status, 1);
endtask : setup_uart_loop_back
```

duolog technologies

Outline

- Motivation - VSP Quality
- Programming Sequences
- Applying Sequences
- Tools Requirements for Sequences
- **Conclusions**

Conclusions

- A Need for Higher Quality Virtual Prototypes – Out of the Box
  - Main Stream implies a need for a higher quality level
- Programming Sequences can be used to assure higher Quality
  - “baseline of tests” that can be applied across the development spectrum
    - TLM unit test
    - Virtual Prototype Integration
    - …
    - RTL Unit Test
    - SoC RTL Integration
    - …
    - Lab Bring up
    - ….
    - Device Driver Development??

duolog technologies

Conclusion


duolog
technologies
•Faster IP integration
•Faster Testbench development
•Faster Software bring up
•Faster debug
•….. faster Time-to-Market
Better Products