



Using Formalized Programming Sequences for Higher Quality Virtual Prototypes

Jack Donovan
Sean Boylan

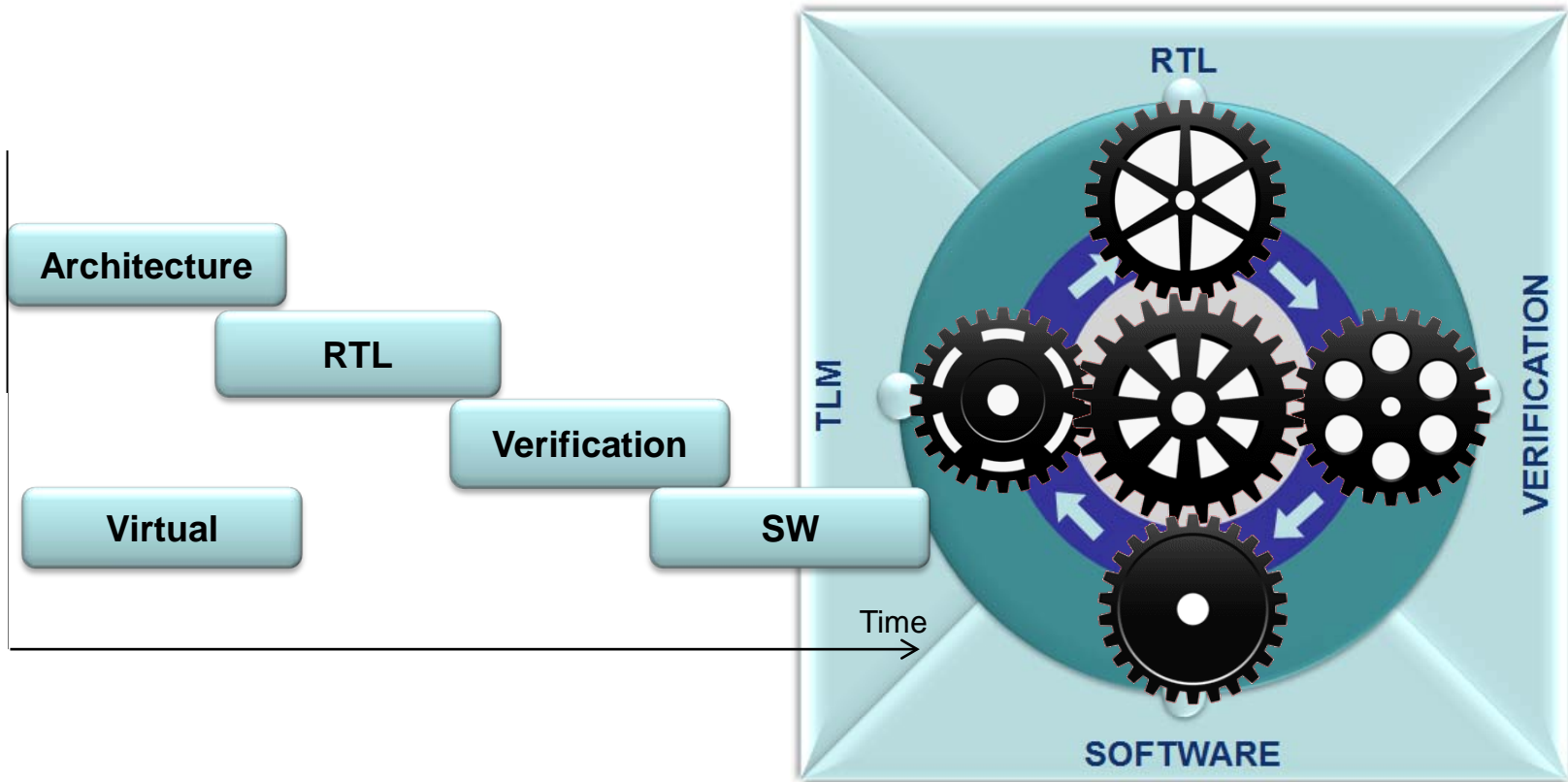
Duolog Technologies
Duolog Technologies



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



ESL Motivation

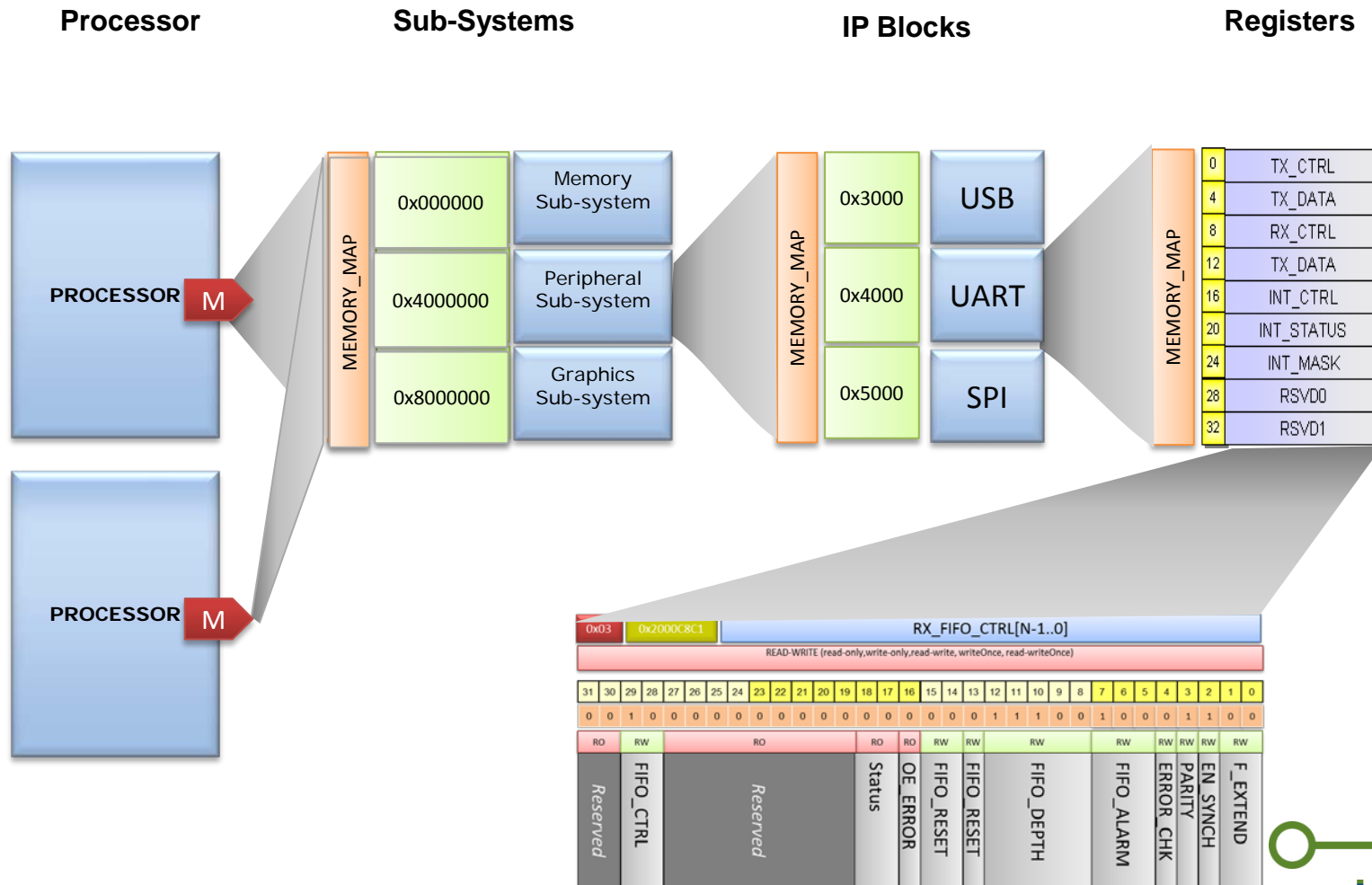
- Virtual Prototypes are now “main stream”
 - No Longer just used by “early adopters”
 - Part of Project Plan ... Main Stream
- Main Stream -> need for higher quality
 - Debug software not Virtual Prototype integration issues
 - Busted Virtual Prototypes can block whole departments of SW engineers
- 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



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

TOP LEVEL

UART

Entry Point: UART_APB

Start Address: 0x00000000

010	UARTDR	0x00000000
010	UARTSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

010	UARTDR	0x00000000
010	UARTSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

010	UARTDR	0x00000000
010	UARTSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

010	UARTDR	0x00000000
010	UARTSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

Constraints

ARM UART PL011

TOP LEVEL
UART
Entry Point: UART_APB
Start Address: 0x00000000

010	UARTDR	0x00000000
010	UARTSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

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

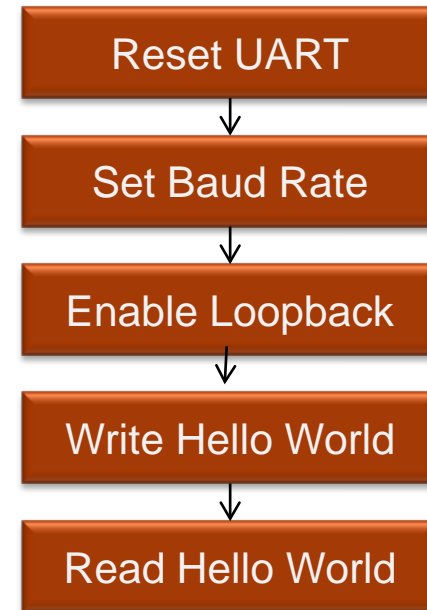
Hierarchical Sequence : Test UART

ARM UART PL011

TOP LEVEL
UART
Entry Point: UART_APB
Start Address: 0x00000000

010	UARTDR	0x00000000
010	UARTRSR_ECR	0x00000004
010	UARTFR	0x00000018
010	UARTILPR	0x00000020
010	UARTIBRD	0x00000024
010	UARTFBRD	0x00000028
010	UARTLCR_H	0x0000002C
010	UARTCR	0x00000030
010	UARTIFLS	0x00000034
010	UARTIMSC	0x00000038
010	UARTTRIS	0x0000003C
010	UARTMIS	0x00000040
010	UARTICR	0x00000044
010	UARTDMACR	0x00000048
010	UARTPCellID0	0x00000FE0
010	UARTPCellID1	0x00000FE4
010	UARTPCellID2	0x00000FE8
010	UARTPCellID3	0x00000FEC
010	UARTPeriphID0	0x00000FF0
010	UARTPeriphID1	0x00000FF4
010	UARTPeriphID2	0x00000FF8
010	UARTPeriphID3	0x00000FFC

Test UART





Outline

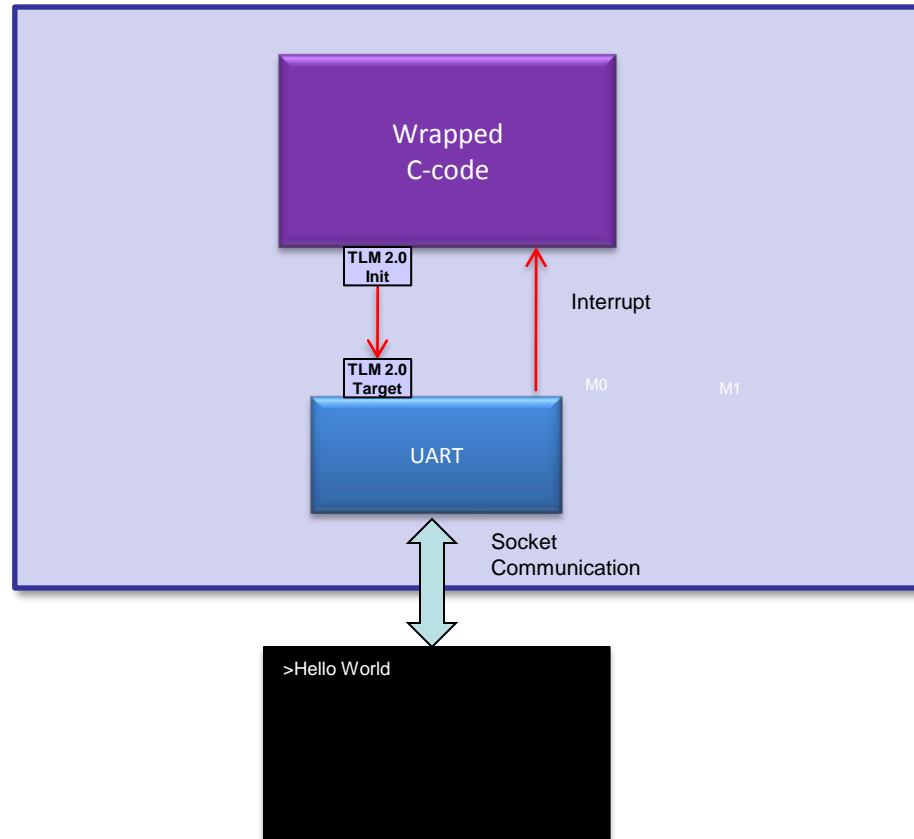
- Motivation - VSP Quality
- Programming Sequences
- **Applying Sequences**
- Tool Requirements for Sequences
- Conclusions



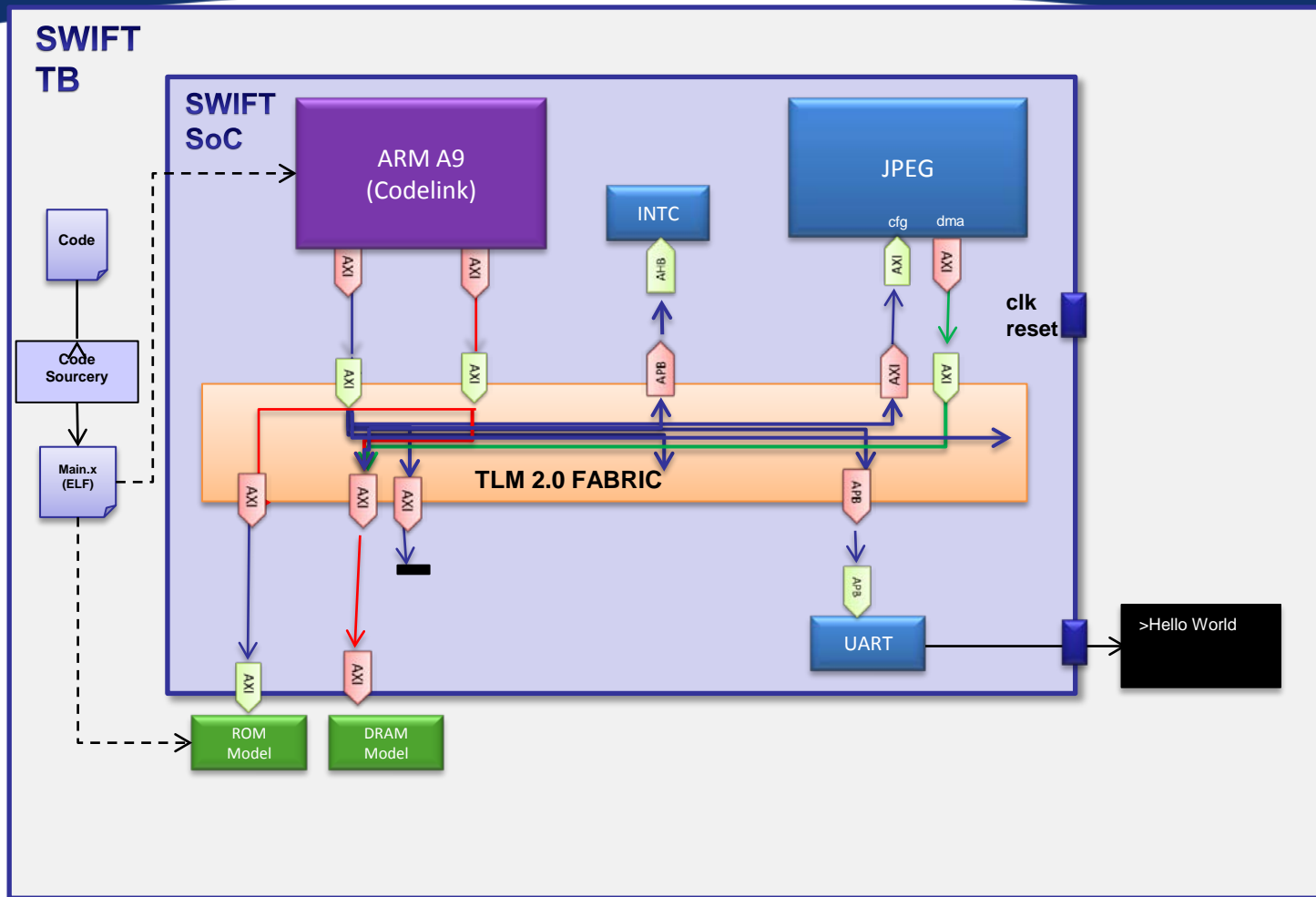
Virtual Prototype Qualification

- 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

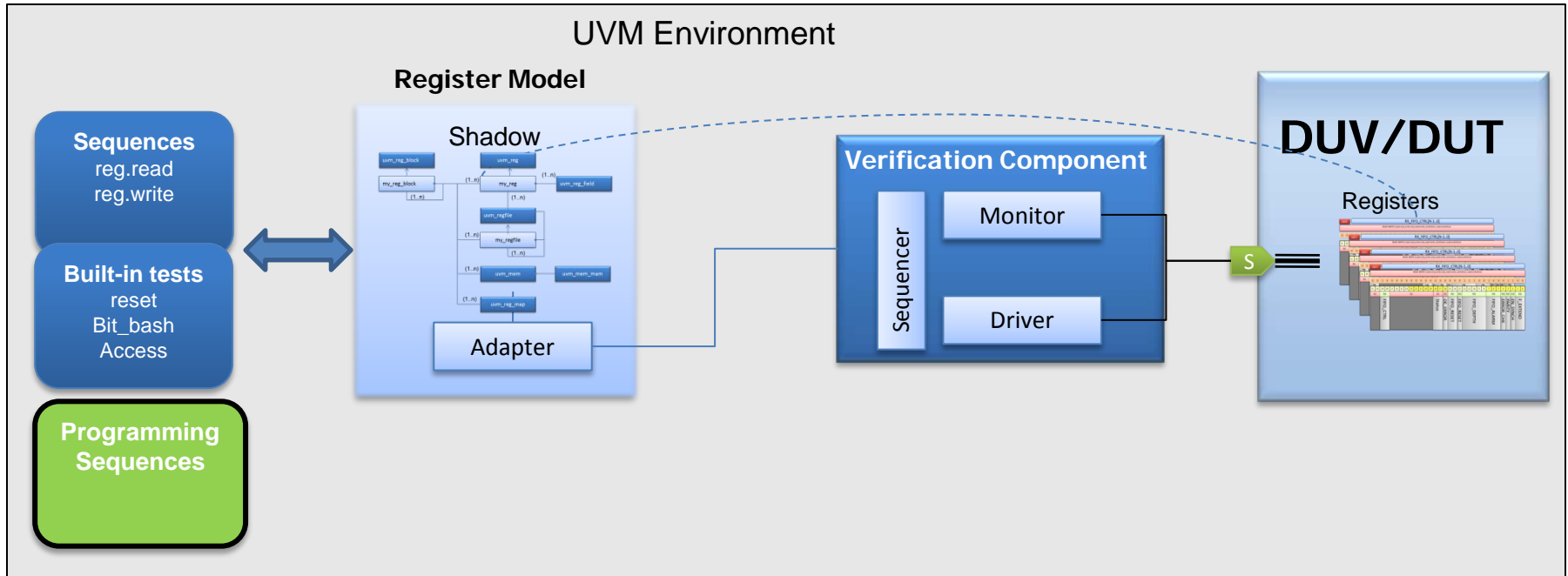
Unit Test



Virtual Proto Incremental Changes



RTL/TLM Unit Test – UVM Environment





Outline

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

Sequence Capture

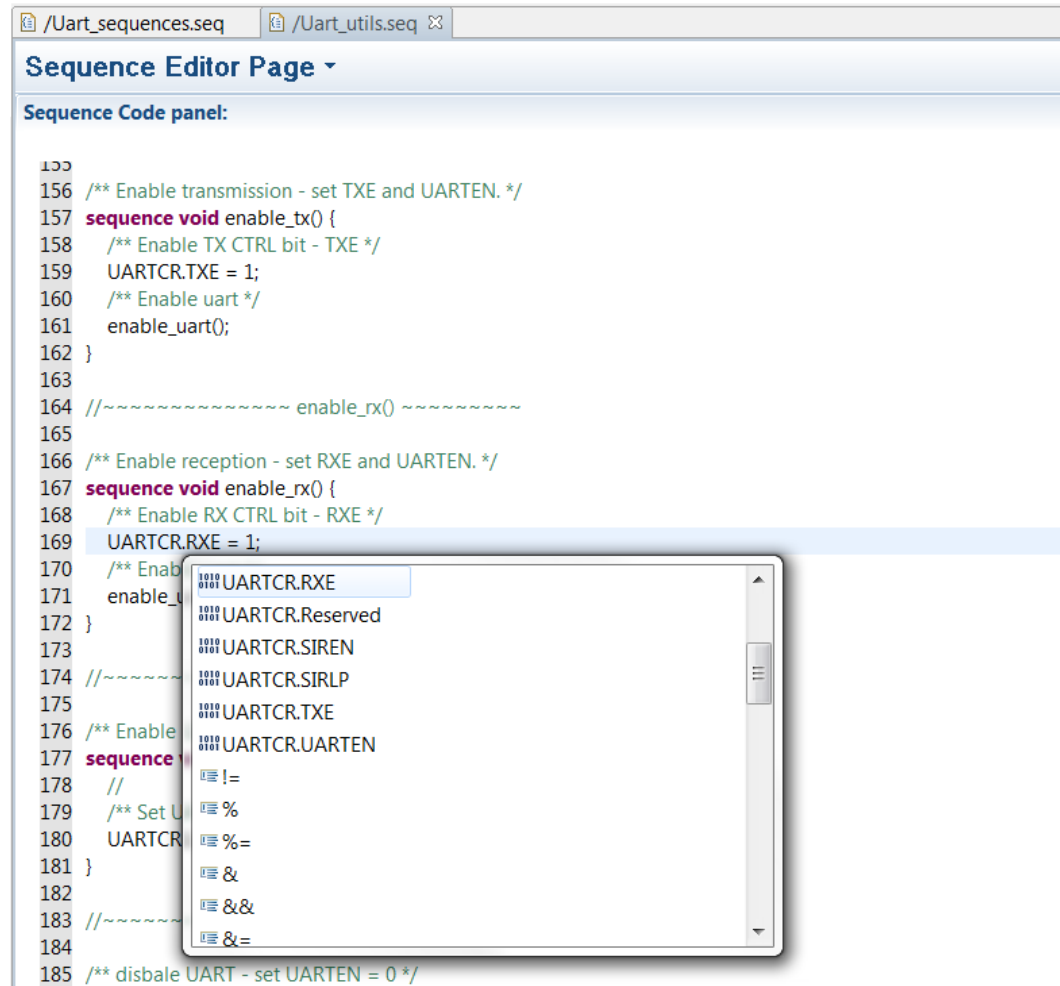
- Captured via a Dedicated Editor
 - Support for auto-completion, syntax checking, code templates and a range of markers
- Sequence Language Easy to Learn and Understand
 - A subset of 'C' with some sequence specific extensions
 - Constraint syntax closely aligned to SystemVerilog
- Leverages Existing 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

Language and an Editor



```
155
156 /** Enable transmission - set TXE and UARTEN. */
157 sequence void enable_tx() {
158     /** Enable TX CTRL bit - TXE */
159     UARTCR.TXE = 1;
160     /** Enable uart */
161     enable_uart();
162 }
163
164 //~~~~~ enable_rx() ~~~~~
165
166 /** Enable reception - set RXE and UARTEN. */
167 sequence void enable_rx() {
168     /** Enable RX CTRL bit - RXE */
169     UARTCR.RXE = 1;
170     /** Enable
171     enable_u
172 }
173
174 //~~~~~
175
176 /** Enable
177 sequence
178 //
179 /** Set U
180 UARTCR
181 }
182
183 //~~~~~
184
185 /** disable UART - set UARTEN = 0 */
```

Sequence Editor Page ▾

Sequence Code panel:

- UARTCR.RXE
- UARTCR.Reserved
- UARTCR.SIREN
- UARTCR.SIRLP
- UARTCR.TXE
- UARTCR.UARTEN
- !=
- %
- %=
- &
- &&
- &=

Coherency Checker

Coherency Check Configuration

Edit Coherency Check Configurations
Organise and edit Coherency Checks to be registered with the Coherency Check Framework.

Visible Coherency Check Configuration Files

Coherency Check Configuration	Enabled	File Location	Type
MyCheck	<input checked="" type="checkbox"/>	C:\showroom\flows\Plato-Sequencer_EP_demo\home\plato-sequencer\workspace\MyCheckC...	WorkSpaceConfiguration
Plato-Sequencer Coherency Checks	<input type="checkbox"/>	C:\showroom\tools\install\Plato-Sequencer-0.4\etc\config\sequencerCoherencyCheck.config	InternalConfiguration

Configuration Description

Coherency Checks in the Set: Sequence Checks

Check Name	Check Description	Enabled	Severity	AutoCheck
Check Referenced Elements Exist	Ensures that all elements referenced in a sequence actually exist in the IP design	<input checked="" type="checkbox"/>	Error	<input checked="" type="checkbox"/>

FilterLinks attached to selected check.

Applied To: Model Tree View

Module	Design Elem...	Attribute

This check has no defined filterlinks. It will therefore run against all elements in the design.

OK Cancel

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)

    initialise_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]

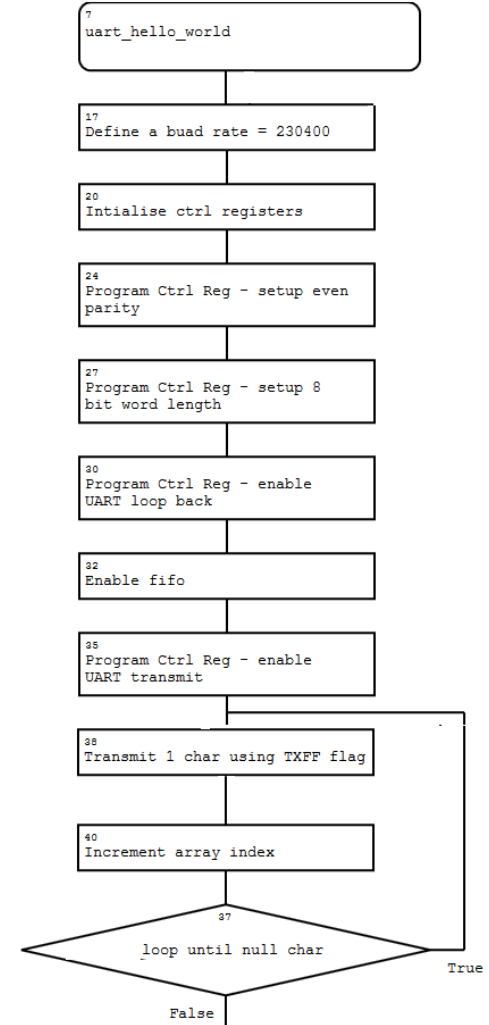
    rev_no = uart_periph_id[23:20]

    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;
        }
    }
}

```

```

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); // verify 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

```



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??

Conclusion

duolog
technologies

- Faster IP integration
- Faster Testbench development
- Faster Software bring up
- Faster debug
- faster Time-to-Market

Better Products

duolog
technologies