

Design and Verification of FPGA Applications

Giuseppe Ridinò Paola Vallauri MathWorks giuseppe.ridino@mathworks.it paola.vallauri@mathworks.it

Torino, 19 Maggio 2016, INAF





Agenda

- Model-Based Design for FPGA
- Generating HDL Code from MATLAB and Simulink
 - For prototyping and production
 - Optimizing code for efficiency
- Verifying HDL Designs with MATLAB and Simulink
 - Co-simulation with HDL simulators
 - FPGA-in-the-Loop verification
- Verifying HDL Designs outside MATLAB and Simulink
 - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C



Model-Based Design for FPGA

RESEARCH	REQUIREMENTS	
ALGO	RITHM DESIGN	≥
Envi	ronment Models	-GO
Digital Models A	nalog Models RF Models	
Timing	and Control Logic	
	Algorithms	EST
	•	 ≪ <
ALGORITH	M IMPLEMENTATION	RIP
C/C++	HDL RF & Analog	
MCU DSP FI	PGA ASIC Transistor	ALGORITHM TEST & VERIFICATION
IN	TEGRATION	

- Model multi-domain systems
- Explore and optimize system behavior
- Collaborate across multidisciplinary teams
- Generate bit-accurate models
- Explore and optimize implementation tradeoffs
- Generate efficient code
- Verify designs to detect errors earlier in development
- Reuse testbenches
- Automate regression testing



It's about Collaboration

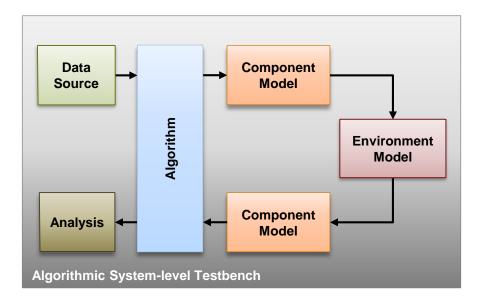
• Usually, many engineers get involved in different parts of the design flow:



- Each brings valuable expertise from their discipline
- Model-Based Design aids collaboration across the project
 - integrating the workflow
 - providing the backbone of a common modelling environment



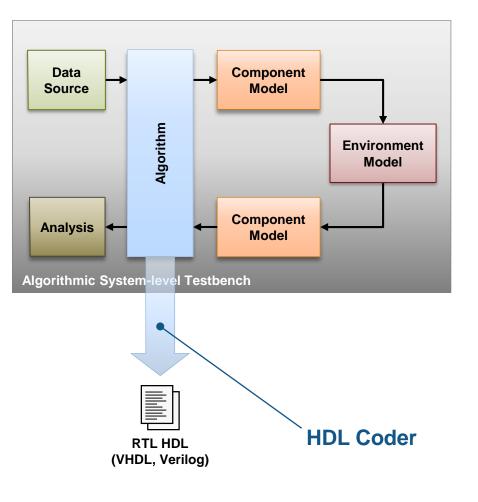
A Typical Model Structure



- Algorithm interacts with outside environment through other components.
- Algorithm is stimulated with data
- Algorithm performance is analysed.



Algorithm Development Generation of HDL Source Code



- HDL Coder
 - Generation of synthesible RTL HDL (VHDL or Verilog)
- Support for
 - MATLAB
 - Simulink
 - Stateflow
- Workflow Advisor
 - Guides through process
 - Preparing model for generation of HDL
 - Configuring HDL Generation options
 - Integrated with FPGA synthesis tools for timing annotation on model
 - Configurations for turnkey FPGA targets and IP Core generation



Simulink Library Support for HDL Generation

HDL Supported Blocks

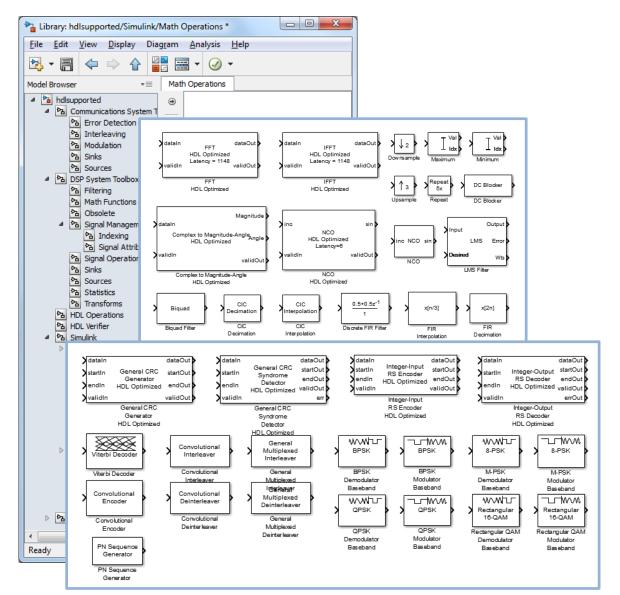
- ~180 blocks supported
- Core Simulink
 - Basic and Array Arithmetic, Look-Up Tables, Signal Routing (Mux / Demux, Delays, Selectors), Logic & Bit Operations, Dual and single port RAMs, FIFOs, CORDICs, Busses

Digital Signal Processing

 NCOs, FFTs, Digital Filters (FIR, IIR, Multirate, Adaptive, Multi-channel), Rate Changes (Up & Down Sample), Statistics (Min / Max)

Communications

 Pseudo-random Sequence Generators, Modulators / Demodulators, Interleavers / Deinterleavers, Viterbi Decoders, Reed Solomon Encoders / Decoders, CRC Generator / Detector



MATLAB & Stateflow for HDL Generation

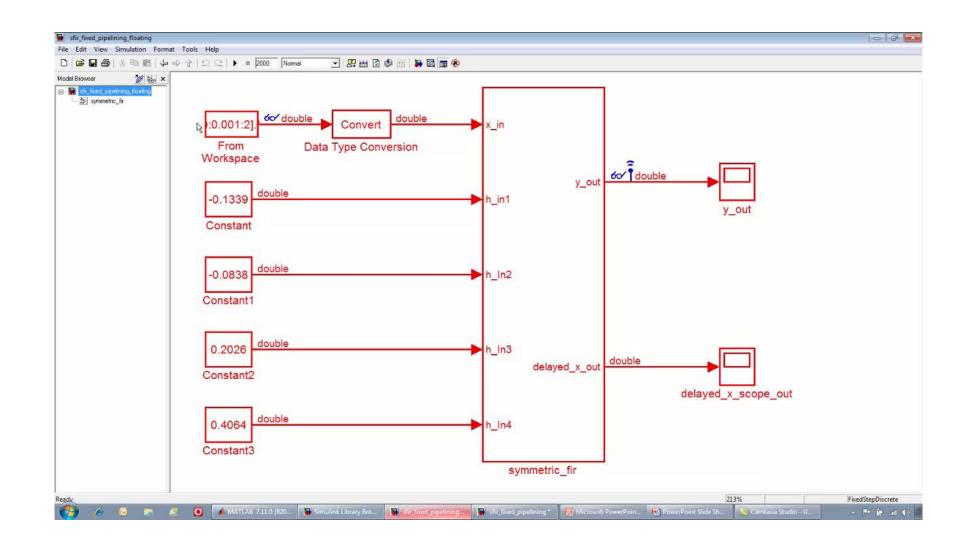
HDL Supported Blocks

- MATLAB
 - Relevant subset of the MATLAB language for modeling and generating HDL implementations
 - Useful MATLAB Function Block
 Design Patterns for HDL
- Stateflow
 - Modeling FSMs (Mealy, Moore)
 - Different modeling paradigms (Graphical Methods, State Transition Tables, Truth Tables)
 - Integrate MATLAB code

	EC	DITOR		VIEW											
	42		E	🔄 Find Fi	les				\$₽ \$≥		0 0 i ₁ 1 ₁			++++ (*****)	
	New	Open	Save	Compa	re 🔻	Comment g	6 🛣	<i>2</i> ,7	斜 Go To	• F	Breakpoints	Run	Stop	_	
	-	•	-	🚍 Print	•	Indent			Q Find		-	Model	Model	•	
			FILE			E	EDIT		NAVIGATE	BF	REAKPOINTS		RUN		
				oel_edge_er											
	16			******		*********	*****		********		********	******	*****	******	
	17 18		uncti	on xr_o	ut =	f12_xf(u,:	xaı, xo	12,10.	2, zai, za	()					
	19 -	f	m = h	dlfimat	h;										
	20 -	c	2 = f	i(2, 0,	2, 0	, fm);									
	21 -			-		9, 0, fm		. .							
	22 -					d2, 0, 10 9, 0, fm		cm);							
	24 -					zd2, 0, 1		, fm)	;						
	25 -					2, 1, 11,									
	26		nd												
	27			******** ute y g			*****	*****	********	****	*******	******	*****	******	
	20					110 8888888888	*****	*****	********	****	********	******	*****	******	
	30	Ģf	uncti	on yf_o	ut =	f14_yf(xd	2,u,yc	d2,1b	1,zd2,1b2	2)					
	31														
	32 -	f	m = h	dlfimat	h;										
	TATES	T	RANSITIC		on / Ac	tion / Desti	· 1);		T				1	~	parse_ir() }
-				if		else-if(1)			Į.		(mar	_assign_pc	0:1 2	26	2
en: y =			x > 0				- .	res		dle			Inst	_fetch ead_mem = 1;	do_one_byte en:mar_assign_pc(); [,
			VEXT				n);	0	pc = 0,		_mem=0; e_mem=0;	(de	write_mem=0;	
• PARK			ift==1		shift	==2				7 7	11 Birl71==	1&&ir[6]+0	188/151-	1[grant == 1 01 {addr = ma	
						-						sign_mar()		20.	read_mem=0;
		N	UTRAL		GEAR		-						1	-01-	pc_increment();}
NEUTR	AL		ift==2		shift	==0							(do_1	two_byte	
					İ				/			-	-4		
		GI	SAR		tion Table Description	Condition	01	1 02 03	D4 D5 D6 I	07 08					0
GEAR		st	ift==1	1	Description	coeff_addr==0	T		04 05 00 1				00		
en: y	, = 20;			1		coeff addr==1	1						-	4 3	4 2 4
		N	SUTRAL	2		coeri_addr==r	-	т -						[ir[7] ==188	ir[6]==08.8ir[5]==1]
						coeff_addr==2								{write_mem	
				3			-	- T							{mar_assign_data(); read_mem=0;}
				4		coeff_addr==3	-		т			11		² Vorant=11	in the second
				Action	Table	l	_							"Bigur-1	
					Description	A1:c0=data_in;	,	Action							
				1	-	wi.cu=data_in;									
					2	A2:cl=data_in;									
				2						1					8

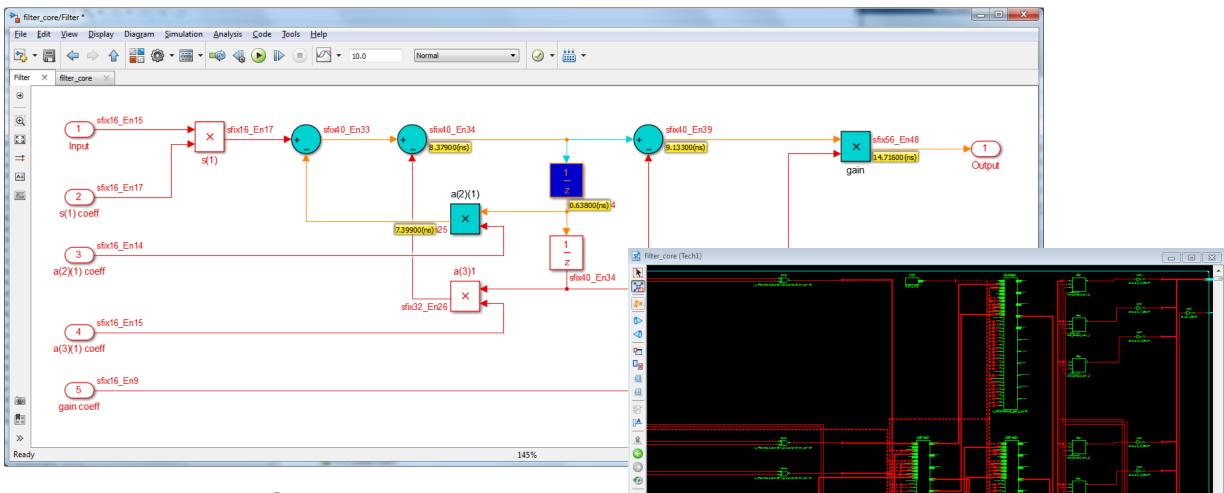


HDL code generation





Critical Path Highlighting and Design Review

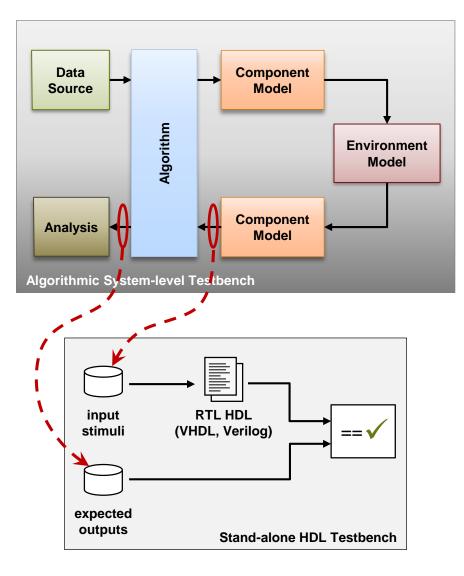


- Feedback in Simulink
- Review results in synthesis tools



Algorithm Verification

Data-driven Verification of HDL Source Code

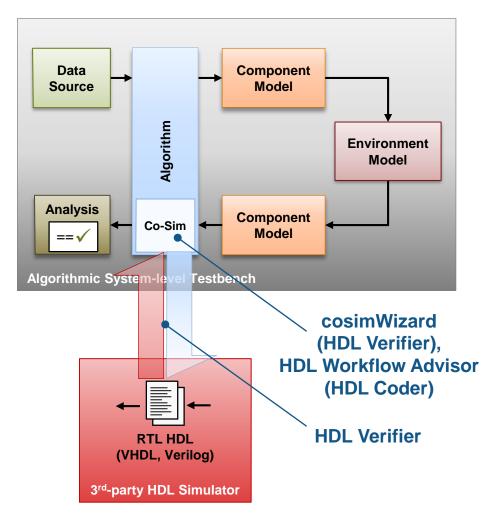


- Stand-alone HDL testbench
 - Stand-alone
 - Executable in any 3rd-party HDL simulator
 - Self-contained
 - Instantiated algorithmic RTL HDL (DUT)
 - Input stimuli stream at DUT top-level interface
 - Expected output stream at DUT top-level interface
 - Self-testing
 - Checks on bit and cycle accuracy
- Handwritten or generated code
 - With HDL Coder, RTL HDL and standalone testbenches are created automatically



Algorithm Verification

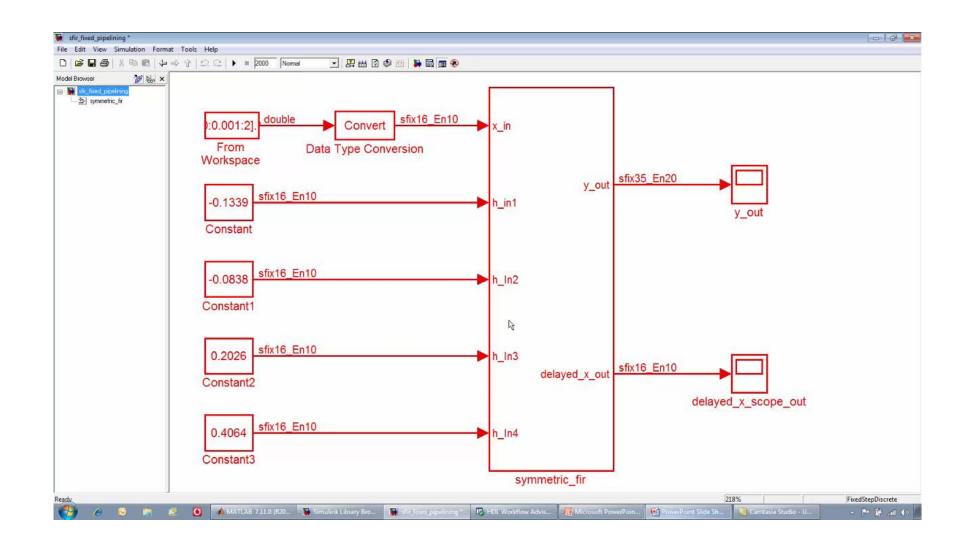
Co-simulation for Verification of HDL Source Code



- Co-simulation with 3rd-party HDL simulator
 - Reuse of existing testbench in MATLAB/Simulink
 - HDL code execution in 3rd-party HDL simulator
 - Flexible HDL sources
 - Handwritten or generated code
 - Automated generation of co-simulation infrastructure
 - Automatic handshaking
 - Combined analysis and debugging in both simulators



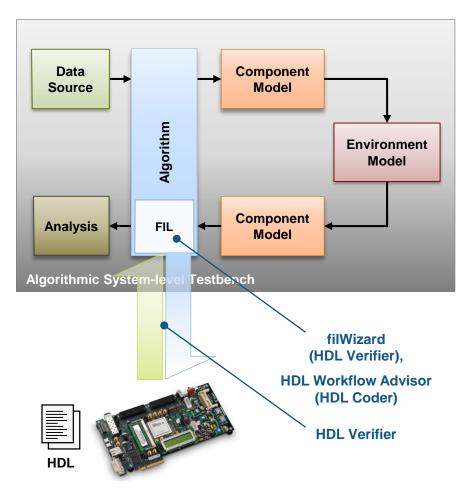
Co-simulation





Algorithm Verification

FPGA-in-the-Loop Verification of HDL Source Code



Prototype your algorithm in hardware connected to the system-level test environment

- FIL simulation with FPGA development board
 - Reuse existing testbench
 - HDL code execution on FPGA
 - Handwritten or generated HDL code
 - Automated generation of co-simulation infrastructure
 - Encapsulation of algorithm within GBit Ethernet MAC, or JTAG
 - Automatic handshaking



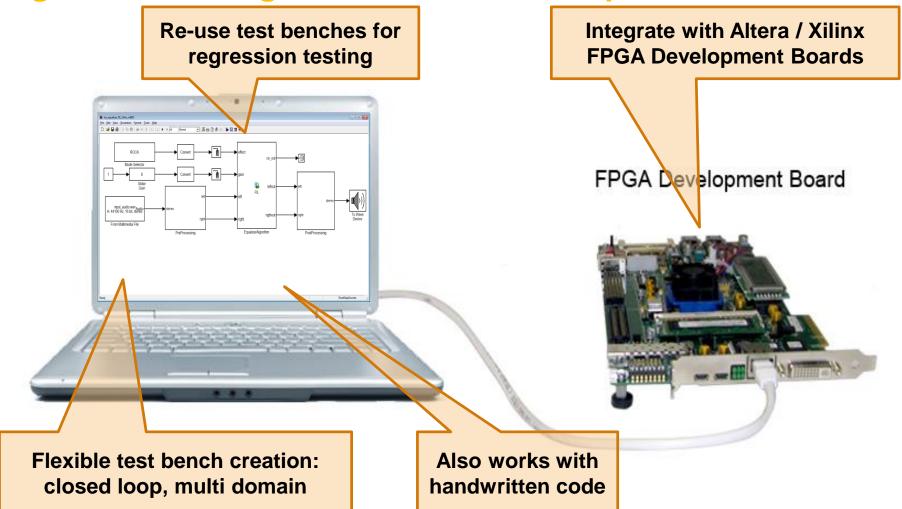
FPGA-in-the-Loop Target Device

ile Edit Run Settings Help								
ind: • 🗇 🌳								
 BHDL Workflow Advisor 1. Set Target ^1.1. Set Target Device and Synthesis Tool 2. Prepare Model For HDL Code Generation 2.1. Check Global Settings ^2.2. Check Algebraic Loops ^2.3. Check Block Compatibility 	1.1. Set Target Device and Synthesis Tool Analysis (^Triggers Update Diagram) Set Target Device and Synthesis Tool for HDL code generation Input Parameters Target workflow: FPGA-in-the-Loop Target workflow: FPGA-in-the-Loop Target North Parameters							
 ^2.4. Check Sample Times ^2.5. Check FPGA-in-the-Loop Compatibility 3. HDL Code Generation 3.1. Set Code Generation Options 3.1.1. Set Basic Options 3.1.2. Set Advanced Options 3.1.3. Set Testbench Options ^3.2. Generate RTL Code and Testbench 4. FPGA-in-the-Loop Implementation 4.1. Set FPGA-in-the-Loop Options 4.2. Build FPGA-in-the-Loop 	Target platform Choose a platform Synthesis tool: Altera Arria II GX FPGA development kit Altera Arria V SoC development board - Rev.C Family: Altera Arria V starter kit Package: Altera Cyclone III FPGA development kit Project folder: Altera Cyclone IV GX FPGA development kit Project folder: Altera Cyclone V GX FPGA development kit Altera Cyclone V GX FPGA development kit Altera Cyclone V GX FPGA development kit Altera Cyclone V SoC development kit Altera Cyclone V SoC development kit - Rev.C Altera DSP development kit, Stratix V edition Altera DSP development kit, Stratix V edition Altera Nios II Embedded Evaluation Kit, Cyclone III Edition Altera Stratix IV GX FPGA development kit Result: Not Rum		eunch Board Manager Refresh Browse					



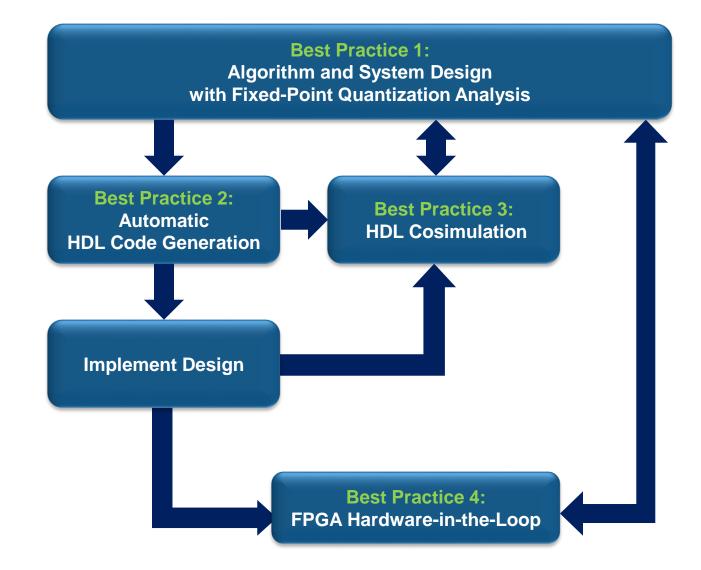
FPGA-in-the-Loop

Enable regression testing with FPGA-in-the-loop simulation





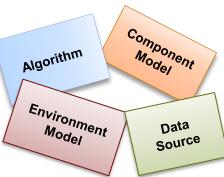
Key Takeaway





Integrating with other Verification Activities

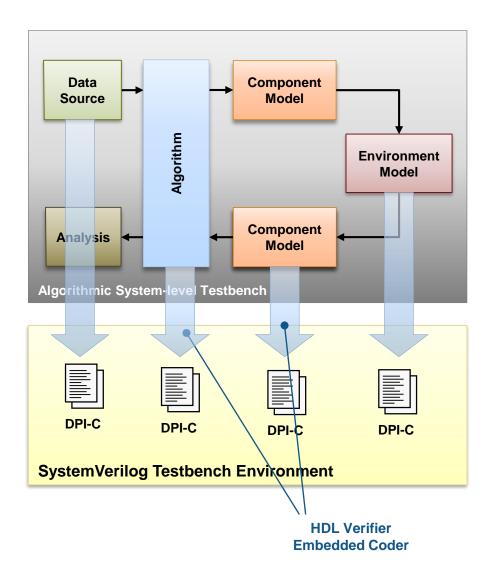
- Verification is the single biggest cost in hardware design
 - Investment in developing simulations for verification
 - SystemVerilog and UVM test frameworks
 - SystemC/TLM virtual platforms
 - Shift towards 'model-based' verification
 - Enabling techniques like Constrained Random testing
- Rather than recreate a behavioural model, we can reuse the assets developed in the system models in MATLAB & Simulink
 - Maintains connection with earlier part of the flow
 - Removes risk of manual error in test framework
 - Avoids duplicating effort





System Verification

Reuse of models in SystemVerilog Testbench



- Code generation translates models to other languages (e.g. C, HDL)
 - Implementation code
 - Verification models
- For verification, C code generation is convenient
 - analog and digital models
 - Wider block and langauge support for C generation
- HDL Verifier extends code generation tools to provide wrappers for
 - SystemVerilog DPI-C
 - SystemC TLM



Integrating DPI-C/SV into Existing Testbench

 Using a public SystemVerilog Testbench example*, adapted to execute the DPI-C as a golden reference:

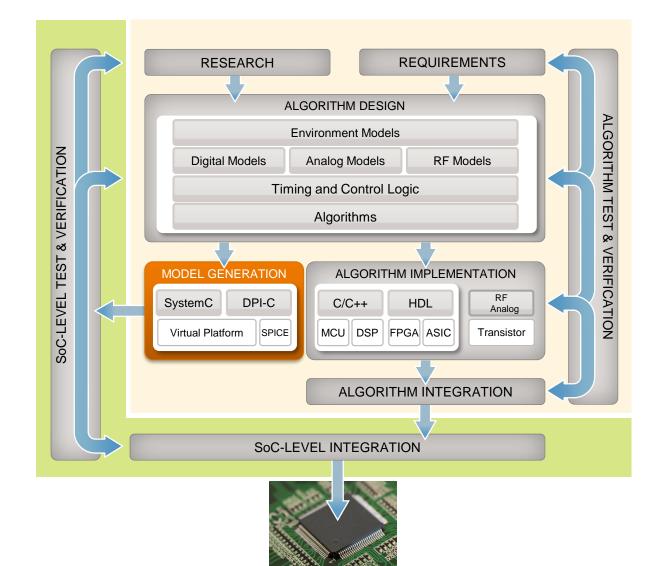
<u></u>	pile <u>S</u> imulate A <u>d</u> d T	<u>r</u> anscript T <u>o</u> ols	Layo <u>u</u> t Boo <u>k</u> marks <u>W</u>	indow <u>H</u> e	lp				
🛿 sim - Default 💷 💴		💊 cts ::::: 🛨 🖻 🗙	📰 Wave - Default 🚃				×		- + 7
Instance	Design unit 🔺	888986 ns ≯ 🕨	💫 🗸		Msgs				
→ top → testb_intf → out_intf → golden_intf → tc → filt → dpi_filt → DPI_sl_filter.	out_interface(fast) golden_interface(fa testcase(fast) sl_filter(fast) sl_filter_dpi(fast)	clk filter_out read read read read		er 8'hf6	890000 ns	(8°h00	<u>(3'h00</u>		
			🔒 🖉 😑 Cur	sor 1	888986 ns	15	200000 Hs	400000 hs	_
Library 🛛 🎬 Project	× 🛺 sim × 🔹	• •	•			•			Ð
690.00us : Score 890.00us : Envir	board : byte 000100 board : Byte from G onment : end of wa: ****** Final Report mpleted Successfull:	olden Ref match it_for_end() me ********	es byte received fr thod	om DUT					-
Total Simulatio	n time: 890.00us	-	*****						

* Example from MicroElectronics Student Group at the University of Porto: <u>http://wiki.usgroup.eu/wiki/public/tutorials/svverification</u>.



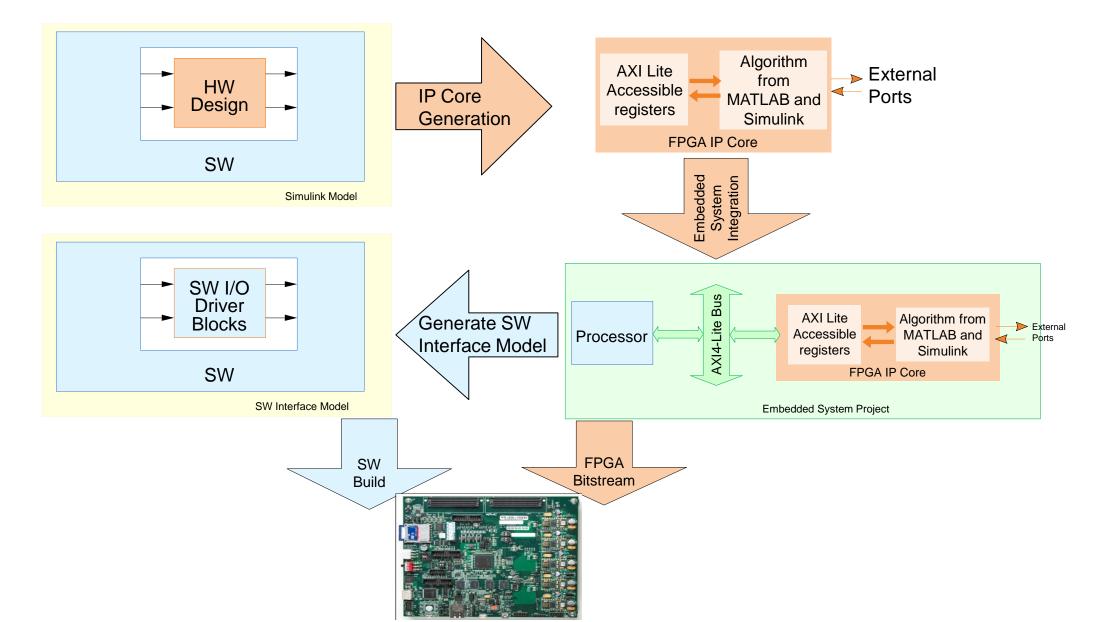
Integrated Verification

Model-Based Design and SystemVerilog/SystemC





Zynq HW/SW Co-design Workflow Summary





Summary

- Model-Based Design for FPGA
- Generating HDL Code from MATLAB and Simulink
 - For prototyping and production
 - Optimizing code for efficiency
- Verifying HDL Designs with MATLAB and Simulink
 - Co-simulation with HDL simulators
 - FPGA-in-the-Loop verification
- Verifying HDL Designs outside MATLAB and Simulink
 - Generating code for integration with SystemC/TLM and SystemVerilog/DPI-C