

CYBERWORKBENCH

NEC's High Level Synthesis Solution



SEPTEMBER 01, 2016 NEC INDIA PVT LTD [Company address]

Contents

Introduction	2
C-Based Design Flow	3
Basic Concept of CyberWorkBench	3
Design Tool Overview	3
Synthesis Flow	4
Verification Flow	5
Behavioral Synthesis	6
Behavioral Synthesis Advantages over Conventional Flows	8
Shorter Design Period and Less Design Cost	8
Source Code Reusability and Behavioral IPs	9
Configurable Processor Synthesis	10
Automatic Architecture Exploration	10
System VLSI Design Example Using C-Based Behavioral Synthesis	11
Summary and Conclusion	12
Detailed Specification of CWB	13
Behavioral Synthesis	13
Input supported	13
Output Languages	15
Target Device Supported	16
Input Constraints	17
Behavioral Synthesis modes or Scheduling options (C to RTL conversion modes)	19
Other Features	19
Verification	33
Integration with Third Party Tools	37

Introduction

The design productivity gap problem is becoming more and more serious as VLSI systems become larger. In the mid-1980s, gate-level design shifted to register transfer level (RTL) design for designs that typically exceeded 100K gates.

Currently, several million gates circuits are commonly used just for random logic parts of a design, which equate to more than several hundred thousand lines of RTL code. It is therefore needed to move the design abstraction one more level in order to cope with this increasing complexity. Behavioral synthesis is a logical way to go as it allows "less detailed design description" and "higher reusability".

A higher level of abstraction description requires smaller code and provides faster simulation time. For example a one million gates circuit requires about 300K lines of RTL (Verilog or VHDL) code, but only around 40K lines of C code. The RTL simulation of 300K lines, is on an average takes 10 to 100 times than the 40K lines of equivalent behavioral code.

The benefits of behavioral synthesis are palpable through multiple commercial chip successes, thus Behavior Synthesis, or High Level Synthesis, is gaining acceptance within the design community. Various commercial chips for printers, mobile phones, set-top boxes and digital cameras are being designed using behavioral synthesis these days. ANSI-C is the preferred programming language for behavioral synthesis because embedded software is often described in C and design tools like compilers, debuggers, libraries and editors are easily available and there is a big amount of legacy code.

Presenting here an overview of C-based design flow where the efficiency and simulation performance is compared against pure RTL with co-simulating it with embedded software. C-based behavioral IPs are advantageous over RTL IPs and the application specific processors can be benefited from it. The hardware architecture explorer at the behavioral level allowing a fast and easy way to study the area, performance and power trade-offs of different designs automatically.

This document covers how behavioral synthesis can be used for any hardware module (data and control intensive).

C-Based Design Flow

NEC have been developing C-based behavioral synthesis called "Cyber" since the late 80's and developing C-based verification tools such as formal verification and simulation around Cyber during the last 20 years All these tools are integrated into an IDE, where designers execute these tools upon the C-source code. The name of this IDE tool suite "CyberWorkBench"(CWB).

Basic Concept of CyberWorkBench

The main idea behind CyberWorkBench is an "all-in-C" approach. This is built around two principal ideas:

- "All-in-C Synthesis": means that all modules in a VLSI design, including control intensive circuits and data dominant circuits, should be described in behavioral C language. The system supports legacy RTL or gate net list blocks as black boxes, which are called as C functions. At the same time it allows designers to create all new parts in C.
- 2. "All-in-C Verification": means that Verification (including debugging) tasks should be done at the C source code. In behavioral synthesis, a designer should not have to debug the generated RTL code. The CWB environment allows a designer to debug the original C source code and the CWB model checker allows designer to write properties or assertions directly on the C source code.

Design Tool Overview

CWB targets general LSI systems which normally contain several CPUs or DSPs, dedicated hardware modules and some pre-designed or fixed RTL-or gate level IP modules, which are directly connected or through buses.

Initially, each dedicated hardware module is described in behavioral C. Once its functionality is verified using the C simulator and debugger, the hardware module is synthesized with the behavioral synthesizer. Configurable processors are also synthesized from the C description in CWB environment. Legend RTL modules are described as function, and handled as a black box.

The CPU bus and bus interface circuits are automatically generated using a CPU bus library. After synthesizing and verifying each hardware module, the design environment allows designers to create a cycles-accurate simulation model for the entire system including CPUs, DSPs and custom hardware modules. With this simulation model, designers can verify both functionality and performance of their hardware design as well as the embedded software run on the CPU, DSP and/or generated configurable processors. The behavioral C source code can also be debugged with the formal verification, property/assertion model checker tool. Global properties and in-context (immediate) assertions are described for/in the C source code. The equivalence between behavioral C and generated RTL can be verified both in dynamic and static way.

Synthesis Flow

The CWB design flow is shown in Figure 1-1. A hardware design in extended ANSI-C (called "BDL", or "Cyber-C"), or SystemC is synthesized into synthesizable RTL with the "Cyber" behavioral synthesizer with a set of design constraints such as clock frequencies, number and kind of functional units and memories. Usually RTL is handles as a black box, but if necessary, the RTL can also be fed to behavioral synthesis. The behavioral synthesizer can insert extra registers to speed up the original RTL and generate new RTL of smaller delay. It also generates a cycle accurate simulation models in C++ or SystemC. The behavioral synthesis can therefore be considered as a Verilog, VHDL, C, C++, and SystemC unification step.

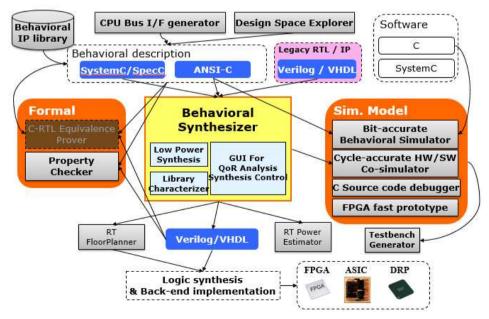


Fig 1-1. CyberWorkBenchTM design flow

The "RTL Floor Planner" takes the RTL modules generated by the behavioral synthesizer with estimated area. Accurate timing information is extracted from the floor planner and fed back to the behavioral synthesizer. The behavioral synthesizer reads the timing information and reschedules the C code considering this.

Verification Flow

The functionality of the hardware described in C can be verified at the behavioral level, while performance and timing are verified at the cycle-accurate level (or RTL) through simulation. CWB has a behavioral C source code debugger linked to the cycle-accurate simulation and FPGA emulation tool. After verifying each hardware module, the entire SoC is simulated in order to analyze the performance and/or to find inter-modules problems such as low performance through bus collision, or inconsistent bit orders between modules. Since such entire chip performance simulation is extremely slow in RTL-based HW-SW co-simulation. CWB generates cycle accurate C++ simulation models which can run up to hundred times faster than RTL model. The HW-SW co-simulator, uses the generated cycle-accurate model for this purpose. The simulator allows designers to simulate and debug both hardware and software at the C source code level at the same time. If any performance problems are found, designers can change the hardware-software partitioning or algorithm directly at the C level, and can then repeat the entire chip simulation. This flow implies a much smaller and therefore faster redesign cycle than in a conventional RTL methodology. This entire chip simulation can be further accelerated using an FPGA emulation board. A "Testbench Generator" helps designers to run an RTL simulation with test patterns for behavioral C simulation faster and easier. Its inputs are test patterns for the C simulation and outputs a Verilog and/or VHDL testbench, which generates stimulus for the RTL simulation. It also creates a script to run commercial simulators to feed the behavioral test patterns and check the equivalence of outputs patterns between the behavioral and RTL simulation.

Another important feature of CWB is the formal verification tools, which is tightly linked to the behavioral synthesizer. With the behavioral synthesis information the formal verification tools can handle larger circuits than usual RTL tools and have C-source level debugging capability even though the model checker works on the generated RTL model. "C-RTL equivalence prover" checks the functional equivalence between a behavioral (un-timed or timed) C description and the generated RTL, using information on of the optimizations performed such as

loop unrolling, loop merge and array expansion performed by the behavioral synthesis. Without such information, the equivalence check is almost impossible for a large circuit.

Designers can specify assertions or properties at the behavioral C level, similar to the cycle accurate simulator. Such behavioral level properties/assertions are converted into RTL ones automatically, and are passed to our RTL model checker.

CWB generates a power enhanced RTL model which estimates the power consumed by the design. A set of power libraries for different technology is provided and used with the generated RTL estimates that power for the selected technology.

A "QoR" synthesis report file of the generated circuit shows a quick overview of the design quality. The report file includes area, number of states, critical path delay, number of wires and routability. This information is used for quick micro-architectural exploration as well as system architectural exploration. The system architecture explorer automatically generates different hardware architectures based on the preferences and constraints entered by the user (area, latency, power) at the C level. The designer can analyze the different generated architectures and finally choose the one that meets the design constraints at the smallest cost.

Behavioral Synthesis

To support the "all-in-C" paradigm presented before, the behavioral synthesizer must cope with three types of circuits: (i) data-dominated, (ii) control-dominated, and (iii) control-flow intensive (CFI) ones. The three types of synthesis engines in order to support these varieties of circuit types: (i) automatic scheduling for CFI and data-flow circuits, (ii) fixed scheduling for control-dominated circuits, and (iii) pipeline scheduling for automatic pipelining or loop folding. Figure 1-2 shows a block diagram of CWB's synthesizer engines.

Control dominated circuits such as PCI I/F, DMA controller, DRAM controller, Bus Bridge, etc., require cycle-by-cycle behavior description, which is fit for timing chart. The extended C language BDL can describe clock boundary in a behavioral description, and is able to express very complex timing behavior concisely. Such description is synthesized with "fixed scheduling" engine. For that circuits, which require fixed sequential communication protocols but all other computations can be freely scheduled, automatic scheduling engine is used for synthesis.

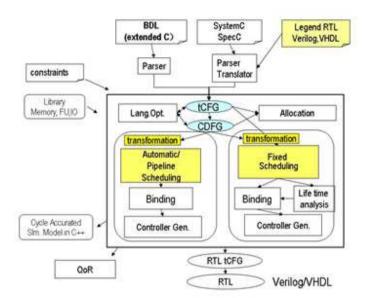


Fig 1-2. Configuration of Cyber Behavior Synthesis

For CFI circuit synthesis, scheduling and allocation techniques play a major role. The quality of synthesis is affected by the control flow structure. A smart scheduling algorithm must be designed to overcome the effects of the programming style. The scheduler will have to modify the control logic in order to obtain circuits with less latency while maintaining the data-flow intact.

Merging two branches into a single using CDFG transformation is not as effective because the procedure is complex and the merging does not always lead to better results. Thus, this approach uses a systematic scheduling algorithm without CDFG transformations. In other words, the scheduler schedules all operations in several basic blocks and several branches at the same time in a unique way, as if they were all operations in a single basic block. The approach handles many other types of speculations, global parallelization with a method called "Generalized Condition Vector", which is extended version of "Condition Vector"

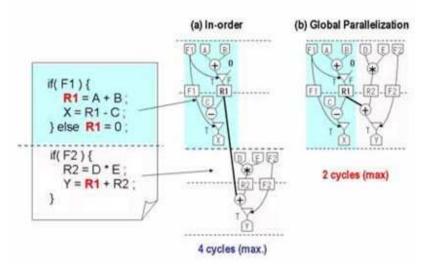


Fig 1-3. Parallelization of multiple branches for control-flow intensive applications (CFI)

Behavioral Synthesis Advantages over Conventional Flows

The next subsections describes in detail some of the advantages of behavioral synthesis over conventional RTL methodologies like hardware-software co-design, source code re-usability, application specific processor optimizations and automatic architecture exploration.

Shorter Design Period and Less Design Cost

Since C-based behavioral synthesis automates the functional design of hardware, it shortens the design cycle and at the same time shortens the design time of embedded software. Figure 1-4 shows the design cycle of two designs. The first uses the traditional RTL-based design flow and the second the proposed C-based design flow. The total design period and design menmonth for the RTL-based design is larger than the C-based one, even though the gate size for RTL design (200K) is one third of that for the C-based (600K) one. The hardware design period of the C-based design is 1.5 months, much shorter than the RTL-based design takes 7 months. It needs to be stressed that the software design in the C-based design takes only 2 months while it takes 6 months for the RTL-based. This is due to the fact that the embedded software can be debugged before the IC fabrication using the hardware-software co-simulator. In RTL design, the software is usually verified on the evaluation board since RTL co-simulation is too slow even for this size of circuits. Lastly, C-based design allows very quick generation of simulation models for embedded software at a very early stage, allowing hardware and software to be concurrently designed both in C.

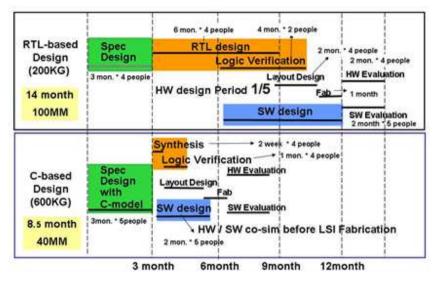


Fig 1-4. Comparison of Design Periods with C-based and RTL- based Design

Source Code Reusability and Behavioral IPs

Another important aspect of CWB is the high-reusability of behavioral models, we call this as "behavioral IP" or "Cyberware". An RT level reusable module, called "RTL-IP", can be successfully used for circuits of fixed performance such as bus interface circuits. However, RTL-IPs for general functional circuits such as encryption can only use for a specific technology, since the RTL-IP's "performance" is hard to adapt for newer technologies. On the contrary, a behavioral IP is more flexible and more reusable than RTL-IPs, since it can change its structure and behavior allowing the synthesis tool can generate circuits of different performances by simply changing high level synthesis constraints such as number of functional units and clock frequencies. Table-1-1 shows how various circuits of different "clock-frequency" can be generated from a single behavioral IP. This IP is a BS broadcast descramblers (Multi2). All generated circuits satisfy the required performance (more than 80Mbps) at various frequencies. Note that the highest clock circuit (108 MHz) uses less number of gates than the slow circuit (33MHz). This never happens in RTL-IPs, which follow the area-delay tradeoff relation of logic synthesis. A behavioral synthesizer generates a smaller circuit of higher clock frequency for the same performance, since less parallel operations are necessary to achieve the same performance at higher clock frequency.

Clock Frequency	Generated Gate size	Generated RTL size	Performance
33MHz	57KG	7.0KL	80Mbps
54MHz	42KG	5.9KL	80Mbps
108MHz	26KG	2.5KL	80Mbps

Table 1-1. BS broadcast descrambler behavioral IP comparison

Another important aspect is behavioral IPs are much easier to modify their "functionality" and "interface" than in RTL-IPs.

The behavioral IPs sometimes generates smaller circuits than RTL IPs as behavioral synthesis share registers and functional units for sequential algorithms, but recent RTL designers do not usually share registers since such time multiplexed sharing makes RTL simulation and debug very difficult.

Configurable Processor Synthesis

Since chip fabrication cost have raised considerably, SoC are becoming as flexible as possible. For this purpose, recent SoC usually have several configurable processors besides a main CPU. These configurable processors should be small, have a high performance and low power consumption for a specific application. Such a configurable processor is also called Application Specific Instruction set Processor (ASIP). ASIPs employ custom instruction-sets to accelerate some applications. The CWB provides ASIP's base processor and supplementary instructions that are described fully in behavioral C, which are behavioral synthesized. This allows the baseprocessors and the addition of instructions to share functional units. This sharing leads to much smaller circuits than the conventional RTL-based ASIPs. C-based ASIPs are more flexible than RTL-based ones in terms of public register number, pipeline stages or interrupt policy.

Automatic Architecture Exploration

CWB allows the creation of multitude hardware architecture for a unique C design. The user can specify a set of constraints which all architectures have to meet (e.g. area, latency, power) and a set of different architectures that meets those constraints will automatically be generated. The area-performance-power trade- offs can be easily analyzed and the architecture that meets the constraints with the lowest cost can be chosen by the designer.

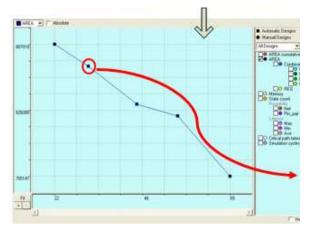


Fig 1-5. Automatic Architectures Exploration

System VLSI Design Example Using C-Based Behavioral Synthesis

Fig.1-6 shows a design example of a real complex SoC used by a cell phones generated with the behavioral synthesizer. This SoC is called MP211, which has three ARM cores, one DSP, several dedicated hardware engines and various applications of mobile phone such as audio and video processing, voice recognition, encryption, Java and so on.

Wide ranges of circuits including control dominated circuits and data-intensive circuits were successfully implemented. The grey boxes (including bus) indicate modules that have been synthesized from C descriptions with the proposed behavioral synthesizer, while the white boxes are IP cores given in RTL format (some are legacy RTL components. All newly developed modules are designed with our C-based design flow. This example clearly illustrates that our C-based environment is able to design entire SoC designs, and not only algorithmic modules.

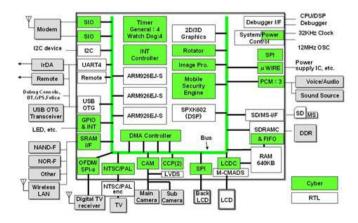


Figure 1-6. Behavioral design flow design example used in a cell phone SoC (Green boxes design using Cyber)

Summary and Conclusion

The advantages of CyberWorkBench includes faster development time, hardware-software cosimulation and development, easier and faster verification as well as automatic system exploration are some of these.

CWB tool is as mature as logic synthesis in the late 80's, when designers started to use them widely RTL level design flows. These days' designers adopt this new design paradigm shifting from RTL "structural" domain thinking to "behavioral" domain thinking.

Detailed Specification of CWB

Behavioral Synthesis

Automatic conversion of C based code to HDL (Hardware Description Language)

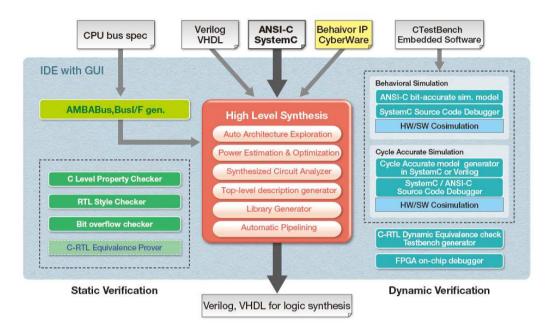


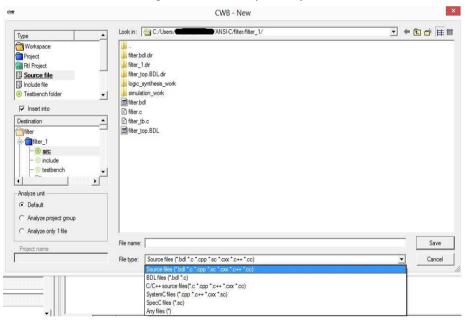
Figure 2-1. Block diagram of CyberWorkBench

Input supported

- a. ANSI-C
- b. System C/ SpecC
- c. VHDL (Legacy RTL with C-based design)
- d. Verilog (Legacy RTL with C-based design)

General Key Editor	Menu Display Analysis/Synthesis Simulation Tool path
1 1 1	
- Analysis	
.cpp files	C (scpars) C Analyze as BDL (bdlpars)
1911	
.sc files	C (scpars) C Analyze as SpecC (SpecC2BDL)
· Analyze as System	c (schais) (Milalyze as Specc (specczbbc)
- Synthesis	
Update infomation file of s	sub module if changed
C Update always	
 Ask me every time 	
C Never update	
History of Synthesis	
Save synthesis results	s(IFF files)
Save synthesis enviror	ment for each surtherin
Je Save synthesis environ	The it for each synthesis
-User define synthesis optio	an file

Figure 2-2. Different ways of analysis





3	rtlpars	×
C/Program Files (x86)/cyber/5.55	Add -> Remove	
include ibi osci osci-dbg packages	Library.	
Venlog file (*.v, *.venlog)	<u> </u>	Options
Verlog file (*.v. *.verlog) VHDL file (*.vhd, *.vhdl)		OK Cancel

Figure 2-4. Input Supported as VHDL/Verilog

Output Languages

The output generated from CWB is VHDL/ Verilog.



Figure 2-5. Generated output as VHDL/Verilog

Target	C Simulation		Logic Synthesis	C Desig	n Compiler
	C Synplify/Certify	C	Leonardo Exemplar		
Multiplexer output format			default C case	C casex	C AND-OR C if-else
default clause in the case statement			append if necessary		ge last item to default
Value to be assigned at default clause in the car	se statement		Undefined value X	C 0	
Reset mode of FF with initial value			Default	C Async reset with set po	t C Async reset without set port
Output line number			Yes	C No	
Dutput tool options as comments		ſ	Yes	C No	
Output empty lower hierarchy module			' Yes	C No	
FU output file		ſ	Same file	C Anot	herfile (OP.v)
Create separate register clock pin synchronous with	h reverse phase				
Add pragma about clock and reset infomation					
Multiple internal memory of same type			 Same module 	C Sep	parate module
Same instance module from each module					
NOT operation on clock for synchronous memory			 current module 	C und	ler layer module
 Output memory model if it is synthesized as memory 	by logic synthesis tool				
 Output memory simulation description 					
Expand multi-bit port of memory into multiple one-bit					
Specify module instance name in false path script f					
Multi bit pin name format at design compiler false pa	ath script				
					Exec

Figure 2-6(a). Different options available for selected output as Verilog

CWB - Venlogg	gen Option - filter(/filter/filter_1)		
eneral Hierarchy Format			
Herarchize module	1	22	
Multiplexer, register Hierarchize multiplexer		 Do not hierarchize 	
Expand to same hierarchy			
C Hierarchize and set to lower hierarchy module			
Specify hierarchy multiplexer More than 4 bits, and More than	8 branches		
Change top module name			
Top module name :			
Add prefix for all lower modules			
 Use top module name as prefix 			
C Use the specified strings as prefix			
Add process name as prefix for functional units			
Add process name as prefix for memory			
Add process name as prefix for decoders			
Add pragma for cyber to module definition			
			Execu
Default Additional Options - Load Save			
		·	
		OK Ca	ancel Ap

Figure 2-6(b). Different options available for selected output as Verilog

neral Hierarchy Format			
Style of register 📀 d	lock enable style		
Suffix of register name			
Dump attribute 🙃 E	nable	C Disable	
	stbench.INST_0 C Specify		
	lexadecimal C Decimal C C		
	nplement with simple algorithm	 Output with operator code 	
	lse unsigned functional unit and sign conve		
		ine synopsis Design Ware 🔿 NEC Design Ware	
addsub type	C select	(share	
Pipeline functional unit(s)		nodel C Simulation model only for unsupported functional unit(s)	
FSM state number	 Parameter variable (with: 		
	Parameter variable (with		
Assignment code with combinational circuit alv		C a<=b	
Assignment code within LATCH circuit always		(• a<=b	
Insert absolute delay to register substitution	OFF	CON	
Parameter name			
Delay value	1		
'define identifier			
"define file name		Browse	
Generate 'define file	C OFF	CON	
Insert delay to async reset	C OFF	CON	
Decoder output format	Index format	C for loop format C case format	
** **	18.95 J. 1.		
			_
			Execu

Figure 2-6(c). Different options available for selected output as Verilog

Target Device Supported

Full ASIC/ FPGA support

Support all FPGA families from Xilinx and Altera

The Generated RTL is optimized for the specified technology.

1.Basic Settings Clock period	1000 1/100ns 💌 =	10 ns (Frequency: 100 MHz)	
Clock uncertainty	200.00 (1/100ns) =	2ns	
C Absolute valu	e of delay (1/100ns) Percentage of the clo	ck period
Effective clock period	i 800.00 (1/100ns) =	9 co	
		0115	
Library	ry C Select Existing Library		
-2 Device	y v Sciect Existing Ebility		
FPGA	C ASIC		
family name	cycloneV 💌		
device name	virtex 🔺		
package name	virtex2		
- 10 - 10 - 10 - 10 - 10 - 10 - 10 - 10	virtex2p		
speed grade	virtex4		
	virtex5		
Logic synthesis tool	virtex6	Follow the logic synthesi	is tool in ba
	virtex6-lower-power virtex7		
3.Basic Library	virtexu		
1	zyng		
Setting	Stop		
\${CYBER_PATH}/	packages/fpga_cv.BLIB		
Delete			
Delete			
	al Unit Library		
-4.Standard Function			
-4.Standard Function Setting	Stop		
Setting			
Setting	Stop packages/fpga_cv.FLIB		
Setting			
Setting			
Setting		T	Modify

Figure 2-7. Different options for selecting families of Xilinx and Altera

Input Constraints

Ability to specify the clock frequency

Ability to specify reset signal and its type.

Ability to control resource allocation: CWB allow user to specify any resource constraints that is desired during implementation of the specific module.

ibrary type GE	NERATED					Library name	ilter					
Library file for an	thmetic operat	ors			•	Unit 1/10	▼ ps ▼					
ibrary file for ba Ibrary file for art	Inmetic operati	018										
FPGA Settings Family cyclor			Device			Package	[Speed	1		
unction unit	Add	Delete	Copy									
Operator name	Kind	Sign	Bit width	Delay	Delay(Input-Rej	Delay(Reg-Out;	Area(Combinatio	Area(REG)	MACRO_BLOC	DSP implement	NETLIST	NET
		1	*							•	2	-
c]												
C + Kind C Funct	ion operator n	ame	< >> < >>		**							
Kind C Funct C Arithm C Arithm		ame	St		Unit 1/"							<u>1</u>
Kind C Funct C Anthro C Al C Sp	netic operator i osolute delay pecify cycle pecify pipeline	ame	St	Cycle (A sma age count slay(Input-Reg)	Unit 1/"							<u>1</u>

Figure 2-8. Function unit libraries for input constraints

e.	CV	VB - bdltran Option - filter(filter/filter_1)				
					☐ Show	v advanced o	ptio
asic setting Mode Clock/Reset	-Macro options						
Specify clock_constraint	Priority for execution time						
brary	1	1					
FU Libraries	NO 1	2	3	4		5	
Memory Libraries	Optimize			Not optimize			
Other Libraries	Long synthesis time	<>		Short synthesis time			
nthesize planning	Other option have priority over Macro			Short aynu load tine	-		
Macro	Contra option nave privily over made						
Error	Priority for circuit area						
stail setting							
Circuit specification1	1						
Language-level optimization	NO 1	2	3	4		5	
• Other optimizations	Large area			Small area			
Scheduling	Fast execution	ç		Slow execution			
FU sharing	Other option have priority over Macro				-		
- Array/Memory					-		
Register FSM	Priority for cycle		Priority for delay				
··· Loop/Function							
- Pipeline	1. C. 1.	2 3		1 2		3	
ol output	NO 1		NO	1 4		3	
·· Circuit specification2	Short cycle	Long cycle	Large delay	<>	Small delay		
- Output file	Large area	Small area	Small area		Large area		
- Information	Other option have priority over Macro	•	Other option have pri	ionty over Macro		-	
	- Other Options						
						-	
MA1 -c1000 -s -Zresource_font=0 [CYBER_PATH]/packages/fpga_	SENERATE -Zresource_mont=GENERATE -Zinter _cv.FLIB	nal_valid_sig_gen -Zdup_reset=YES	EE Ib \$(CYBER_PATH)/p	ackages/fpga_cv.BLIB -ffl		Execu	,t
						Stop	p
Default Additional Options	▼ Load Save						
				ОК	Cancel	1	
				UK	Cancel	Арр	iy

Figure 2-9. Different options for synthesize planning to control resource allocation.

-	Add	Delete		бору					
Operator name	Max. FU count	Max. FU count for folding loop	Alias	Kind	Sign	Bit width	Delay	Delay (Input-Reg)	Delay (Reg-Outpu
4									<u>,</u>
Maximum FU	count + -	1	For	folding loop .		1			
		J							
	ect Add	Delete							
Specify chain ef	ea Aub			Operator name		Specify chain de	stination delay		
Specify chain ef		Chain destination		porotor manie		-			
		Chain destination	(poracol marine					
		Chain destination		perecer menne					

Figure 2-10. Different options for function constraint to control resource allocation.

Main Analysis	Error Display
-1.Basic Settings Clock period	1000 1/100ns 💌 = 10 ns (Frequency: 100 MHz)
Clock uncertainty	200.00 (1/100ns) = 2ns

Figure 2-11. Options for specifying clock frequency

		Show advanced op
basic setting	Clock/Reset port	
"Mode "Glock/Rest "Glock/Rest "Glock/Rest "Glock/Rest "FU Ubraites "FU Ubraites "Memory Libraries "Other Ubraites "Other Ubraites "The Ubraites	Cubch Needs point Clock Name CLOCK Reining edge C Falling edge When not specified C Do not create when not required C Create always G Generate clock signal dedicated for inversed edge triggered registers Clock signal name C Clock name + "_FRUV" C Specify Reset Name RESET C Active high C Active low C Synchronous C Asynchronously The number of registers inserted at reset input F Make a duplicate of reset signal in each clock domain When not specified C D on to create when not required C Create always	
Circuit specification2 Output file Information	Initial values at variable declaration C Assign at the reset state C Assign at the first state	
and the second	Reset state creation	
-c1000 -s -Zresource_fcnt=GENE \${CYBER_PATH}/packages/fpga		Execut Stop

Figure 2-12. Settings for frequency reset

Behavioral Synthesis modes or Scheduling options (C to RTL conversion modes)

- Automatic scheduling
- Manual Scheduling
- Mixed Scheduling (user specific)

1	CWB - bdltran Option - filter/filter/filter_1)	
	☐ Show ad	vanced option:
seic setting - Mode - Occk/Reset - Specfy clock_constraint bray - FU Librates - Memory Librates - Circuit specification 1 - Languagetivel optimizations - Schedung - FU banng - Array, Memory - Register - FSM - Loop/Function - Peline o louput	Synthesis Mode Manual scheduling Automatic scheduling Coste in text constraints automatically Memory constraints generate/use mode Generate memory constraints automatically Memory lated in MCNT file Nove hierarchical modules Memory lated in MCNT file Nove hierarchical modules Memory lated of lower hierarchical modules Memory constraints agines of lower hierarchical modules Memory constraints automatically Memory lated in MCNT file Nove hierarchical modules Memory lated in MCNT file Nove hierarchical module	
Output file Information Informati		Execute Stop Apply

Figure 2-13. Behavioral synthesis mode (Scheduling options)

Other Features

Hardware architecture or design level optimizations at each source code level as well as at module level and both by user specifications and automatically by tool such as:

Loop merging, optimization, unrolling, pipelining, false loop detection and loop parallelization.

Automatic bit-width optimization, Automatic priority logic generation for shared memories and registers, Speculations and suggestions for optimizations, Operator level optimizations, Array Overflow checks etc., Register sharing and exclusivity, Reset State Behavior defining, Interfaces, variables, arrays initializations, Selecting memory and register types (ports of memories and registers clock style or feedback style). Selecting memories as synchronous/ asynchronous and port access mechanisms.

Ability to define various types of interfaces for behavioral designs such as pipelined, buffered, serial or parallel. Ability to define scheduling of behavioral operations to be done in each cycle at source code level. Micro-architectural Design Exploration: Ability to generate multiple RTL designs from one behavior code under given or changing physical constraints. Generate Trade-off chart between area/resource, latency and timing and provide choices to user for best design

as per the requirement. Automatic top module generator to integrate different modules of (cdesigned module or existing RTL designs modules) and able to define top module in C as per the user requirement. On-chip Bus Interface Generator to automatically generate bus interfaces for AMBA AHB, AXI, APB and bridges. On-chip Bus Generator to automatically generate onchip buses (AMBA based). Ability to handle clocks domain crossing (Supports asynchronous and synchronous interface generation between two modules at different clocks) while generating top module and integration of modules. Ability to support Clock Gating. Detailed QoR report generation for the design, having information about area, latency and resources utilized. Data path or RTL schematic Viewer capability. Dataflow diagram viewer and Signal table to

Synthesis options for generating kind of FSM encoding required. Synthesis options for kind of logic to be generated for constant array, variables and globals Ability to maintain history of changes made in the C code and behavioral synthesis in case user wishes to compare or move to previous design or next design. RTL generation options: RTL code generation as per the target (Logic or FPGA Synthesis, DC Compiler, Simulation etc.)

show operations occurring in each state of FSM generated.

to-generated Property Bug Detection Desir	Constraints Options Version	Info. Outputs	
Checker type	Meaning	Pi	Check all
			Uncheck all

Figure 2-14. Array overflow check

-Bitwidth Optimization

Optimization level options

- Optimize bitwidth of all variables
- C Optimize bitwidth of logic type (char, short, int, long, or long long) variables and internally generated variables
- C Optimize only bitwidth of internally generated variables
- C No bitwidth optimization performed, including internally generated variables

Oefault

- C Optimize bitwidth of array bound to memory
- C Do not optimize bitwidth of array bound to memory

Figure 2-15. Automatic bit width optimization

Shared array

Create read enable for multiplexor(s) for address port of memory without read enable and chip select

Create read enable for multiplexor(s) for address port of shared register array(LUTRAM style)

Figure 2-16. Automatic priority logic generation

		Show advanced o
sic setting Mode Clock/Reset Specify clock_constraint orary	Image redundant logic operations Ratem multi-verim hulpscherin Image redundant logic operations Ratem multi-verim hulpscherin Image dapath Image redundant logic operations	
- FU Libraries - Memory Libraries - Other Libraries thesize planning - Macro	Build in Self Test	
Error all setting Circuit specification 1 Language-level optimization Other optimizations Scheduling FU sharing Array/Memory	Pot	
Register FSM Loop/Function Pipeline	Generate multiflipflop synchronizer at input ports. Number of registers for synchronizer at each port Generate mux synchronizer:	
output Circuit specification2	Others	
• Output file	Optimization in identical conditional branches	
c1000 -s -Zresource <u>f</u> ont=GEN er-auto FLIB +#6 filter-amacro-au	ERATE -Zresource_mont-GENERATE -Zintemal_valid_sig_gen -Zdup_reset=YES -EE-lb s(CYBER_PATH)/packages/fpga_cv BLIB-lfl s(CYBER_PATH)/packages/fpga_cv FLIB +lfl to FLIB #c filter=auto FCNT +dc filter=auto MLIB +mc filter=auto MCNT	Exect
Factor and mining remactored		Stop
Default Additional Option	is ▼ Load Save	1

Figure 2-17. Clock domain crossing

5 <u>7</u>	CWB - bdltran Option - filter/filter/filter_1)	<u>×</u>
	☐ Show advanced of	options
basic setting - Mode - Cock/Reset - Specify clock_constraint Ubrary - FU Ubranes - Memory Ubranes - Other Ubranes - Other Ubranes - Memory Ubranes - Memory Ubranes	for all registers assigned to variables Maximum length of register name 32 include function name in variable name Al variable names and function names first	
Error detal setting - Circuit specification 1 - Language-level optimization - Other optimizations - Scheduling - Scheduling - Full sharing - Array/Memory Register	gated dock ✓ Synthesize register with gated clock Synthesize register withose bitwidth is or more than with gated clock Plegister coptimize ← The register only in initialization with substitution is optimized. ← Do not optimize register feedback loop to retain register value.	
- FSM - Loop/Function - peline tool output - Circuit specification2 - Output file - Information	Default value for write data port of shared register (not including arrays) O Don't Care O 0	
-c1000 -s -Zresource_font=GENE +fl filter-auto.FLIB +fl filter-amacro-	RATE -Zesource_mont-GENERATE -Zinternal_valid_aig_gen -Zdup_reset=YES -Zgated_clock -EE -Ib \$(CYBER_PATH)/packages/fpga_ov BUB #I \$(CYBER_PATH)/packages/fpga_ov FUB auto FUB #Ic filter-auto.PCNT +If	
Default Additional Options		<u> </u>

Figure 2-18. Clock gating

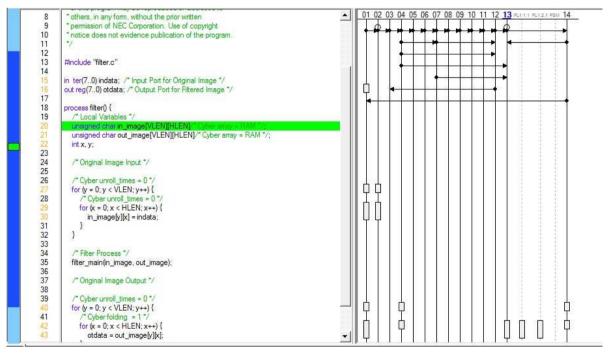


Figure 2-19. Dataflow viewer

Ort Stage Stage 2 FSM indata in ter(70) indata; otdata out reg(70) otdata; RG_x int x, y, kx, ky; RG_x_011 int x, y; RG_y_11 int x, y; RG_y_11 int x, y; RG_y_11 int x, y; Memory w Image(MEMB8W62208) unsigned char in_image[VLEN][HLEN] Image(MEMB8W62208, RA1 MEMB8W62208, RD1 MEMB8W62208, WA2 MEMB8W62208, WA2 MEMB8W62208, WC2 MEMB8W62208, WC2 MEMB8W62208, WC2 W W W W W W W W W W W W W W W W W W W W W W W W W W W W W W W W<	Port Reg	Mem Fu			Others												
Inidata in ter(70) indata; inidata out reg(70) otdata; Register	Signal Name	C/bdl	01	02	03	04	05	06	07	08	09	10	11	12		1	14
- otdata out reg(7.0) otdata: w - r RG_x int x, y, kx, ky; - RG_y int x, y, kx, ky; w RG - RG_y' int x, y; w RG RG - RG_y' int x, y; w RG RG - RG_y' int x, y; w w RG - RG_y_1 int x, y; w w RG RG - out_mage(MEMB8W62208, unsigned char out_image[VLEN][HLEN] w w w r MEMB8W62208, WC14 w w w w w w MEMB8W62208, WC12 w w w w w w			-											-			
egister		in ter(70) indata;		R						-							
- rddata out reg(7.0) otdata: w w w w - RG_x int x, y, kx, ky; w w RG RG RW RW R W w	otdata	out reg(70) otdata;														w	
W w r R R R R W Image: Constraint of the state of											a				a		
														-		w	
W W R R W R W RG_y intx,y, kx, ky; intx,y; w w w R		int x, y, kx, ky;	W	rw	r	r	R			R	RW		R				
With Section With Section With Section With Section With Section With Section Rest of the section Image MEMB8W62208 unsigned charin_image [VLEN][HLEN] With Section Image Membra Section		int x, y;				w	-		_		-			and the second second	rw		W
Image		int x, y, kx, ky;	W	rw	B	r			RW		R					_	_
M r R		int x, y;				w	1		w					R	R		RW
Bit With With With With With With With Wi						_				_		_			41 1	10	
MEMB8W62208.RA1 w w w MEMB8W62208.RCLK1 -				W		r	R	R	r	R	R	R	8	r			
MEMB8W62208.RCLK1		unsigned char out_image[VLEN][HLEN				w	-				-		_	W		r i	-
MEMB8W62208.RD1 r MEMB8W62208.WA2 w						-				-					w		
MEMB8W62208.WA2 MEMB8W62208.WCLK2 MEMB8W62208.WCLK2			_							-							-
-MEMB8W62208.WCLK2			_							-						r.	-
MEMB8W62208.WD2						w				-				W	1		-
MEMB8W62208.WE2		2								-				_			
						w	2			-				W	2		-
VERY CONTRACTOR OF A						W								W	1		
	unction Unit				_				_	12						00	
-add12s_11@1 use use USE USE USE					use	use			USE	_		USE					
				USE		use	USE	USE		USE	USE	-	USE		USE	_	
									Use						1		

Figure 2-20. Dataflow viewer signal table

Timed C Coding (Specify clock for C language coding)

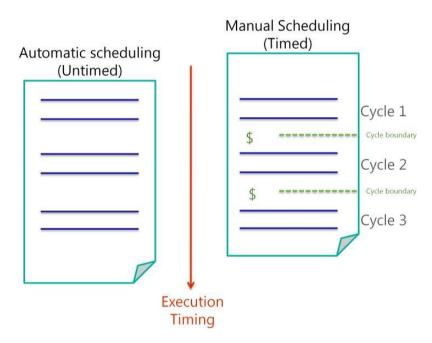


Figure 2-21. Different scheduling behavioral operations to be done each cycle

Cyber Synthesis Report Summary

	Module Name		Basic Library Name							
	filter			BLIB						
FPGA Fami	ly FP	GA Device	FPG	A Package	FPGA Speed					
cycloneV		12 A		2	-					
	Resource	<u>Utilization</u>								
ALUTs ¹	UTs ¹¹ Registers Block M Bi		DS	iPs						
498	189	995,328	()						
atency Index	Clock Period	Net	<u>P</u>	ort						
746,929	10ns	617		18						
Total States	<u>Critical Path</u> <u>Delay</u>	<u>Pin Pair</u>	In	Out						
14	11.4213ns	2,144	10	8						

Resource Utilization

Module Name	Count	ALUTs 11	Registers	Block Memory Bits	DSPs
Total	5-	498	189	995,328	0

Functional Unit

FU Name	Kind	Sign	Bit Width	Area	Reg	Delay (ns)	Count
add12s_11	+	signed	(12,9) 11	13	0	0.99	1
add12u_11	+	unsigned	(11,9) 11	13	0	0.99	1
add8s	+	signed	(8,6,1) 8	9	0	0.86	1

Figure 2-22. Detailed QOR report

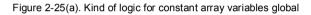
n an		CWB - bdltra	n Option - filter(/filter/filter_1)		×
					Show advanced options
basic setting - Mode - Mode - Clock/Reset - Specify clock_constraint Library - FU Ubranes - Memory Ubranes - Other Libranes - Other Libranes - Where Ubranes - Other Libranes - Synthesize planning - Error detail setting - Crout specification 1 - Unarguage-level optimization - Other optimizations - Scheduling - FU sharing - Aray/Memory - Register - FisiM - Loop/Function - Register - Dolut specification 2 - Output file - Unarguage-level optimization - Color Junt - Const specification 2 - Output file - Unarguage-level optimization - Dolut specification 2 - Output file - Unarguage-level optimization - Dolut specification 2 - Output file - Information	FSM state encoding multiple FSM © Default © Generate FSM w © Single FSM State encoding © Binary FSM FSM © Transit from illegal sta	C FSM partitioning Large <control delay=""> Small Small < Circuit area> Large C Grey code</control>	State/FSM	C Onehot	
-c1000 -s -Zresource_fcnt=GEN filter-auto.FLIB +fil filter-amacro-a	IERATE -Zresource_mont=GEI .to.FLIB -lfc filter-auto.FCNT +lf	NERATE -Zintemal_valid_sig_gen -Zdup_reset=Y fc filter-amacro-auto.FCNT -Iml filter-auto.MLIB -Im	ES -EE -lb \${CYBER_PATH}/packages/fpga_cv c filter-auto.MCNT	r.BLIB -lfl \${CYBER_PATH}/packages/fpga_cv.FLIB +lfl	Execute Stop
Default Additional Optio	ns 🕶 🛛 Load 🛛 Sa	we			

Figure 2-23. FMS encoding

Instance count	AREA total	State count	FU	REG	MUX	DEC	MISC	Memor	Pin_pa	Net	Latenc	Critical	BlockM	DSP	
1	498(100.0%)	14	140	189	264	14	80	5	2144	617	746929	1.4213r	995328	0	
														J	
	498		140	189	264	14	80	c:	2144	617	746929	1.4213r			
	and the second sec	1 498(100.0%)	1 458(100.0%) 14	1 498(100.0%) 14 140	3 498(100 <i>0%)</i> 14 140 189	1 498(100.0%) 14 140 189 264	1 438(100.0%) 14 140 183 284 14	1 498(1000%) 14 140 189 264 14 80	1 458(100.0%) 14 140 189 264 14 80 -	1 498(100.0%) 14 140 189 264 14 80 - 2144	1 498(100.0%) 14 140 189 264 14 80 - 2144 617	1 498(100,0%) 14 140 189 264 14 90 - 2144 617 746929	1 458(1000%) 14 140 189 264 14 80 - 2144 617 7469291.4213~	1 458(100.0%) 14 140 189 264 14 80 - 2144 617 7469291.4213r 995328	1 498(100.0%) 14 140 189 264 14 80 - 2144 617 74692914213+995328 0

Figure 2-24. History of synthesis

-Register array	
	In Default style C LUTRAM style C Decoder style







str.	CWB - bdltran Option - filter(/filter_1)	
	☐ Show adv	anced options
basic setting Mode Cock/Reset	Initial values at variable declaration • Assign at the reset state • Assign at the first state	-
Specify clock_constraint Library FU Libraries Memory Libraries	Peset state creation C Decided automatically C Create C Do not create	
Other Libraries	- Reset for register -	
Macro Error detail setting	Do not reset registers without nitial value Reset registers connected to adput signal Reset registers connected to valid as atomis	
Crout specification 1	T Initialize shared registers in current process	
Other optimizations Scheduling FU sharing Array/Memory	Initialize uninitialized global signal(s). I∽ Initialize uninitialized global signal(s). (except for exclust signal(s) and global register array(s)).	
Register FSM Loop/Function	Initialize uninitialized output signal(s). Initialize uninitialized global register array(s).	
L- Pipeline tool output Circuit specification2	Occk/Reset for shared registers Cock/Reset for shared registers Cock/Reset for shared registers as that of lower module which does write operation C Set current module of clock/reset signal as input of shared registers	
- Output file - Information	Clock/Reset for shared memories Cock/reset signal of shared memories as that of lower module which does write operation C Set ourer module of clock/reset signal of shared memories	
	Section submodule Section submodule Section submodule	
-c1000 -s -Zresource_font=GENI filter-auto.FLIB +ill filter-amacro-aut	RATE "Zresource_mort-GEDRATE" Zritemal valid_sig_gen "Zdup_reset+YES EE do \$(CYBER_PATH)/packages/pga_cv & BLB #I \$(CYBER_PATH)/packages/pga_cv FLB #I ROLB # of iter-auto FCNT #of iter-auto MLUB #mc Titer-auto MCNT	Execute
Default Additional Option	_	Stop Apply

Figure 2-26. Different options for Interface variables arrays initializations

	☐ Show	advanced optio
eic esting	Loop unrolling Cop ond unroll of for loops G Specify the condition of unrolling G Unrol for loop(s) (f the number of statements after unrolling is less than or equal to C Unrol for loop(s) (f the number of testion is less than or equal to C Unrol for loop(s) Copy enterset Copy enterset Converting functions C Function call C Default C Inline Many ← Cycle court → Fev Small ← Circuit area → Large Initialize uninitialized local variables with 0	
c 1000 e -Zmeouron font-GEN		
CYBER_PATH)/packages/fpg	на импенающие уполначите на кото на кото је удени за орудени се оконо се по еконо се при кото и уделеко удеј си ВЕВ 41 да јех FLIB	Execute

Figure 2-27(a). Options for Loop merging, optimization, unrolling, parallelization

For binding algorithm 2 Image: Constraint of the second second

Figure 2-27(b). Options for Loop merging, optimization, unrolling, parallelization

Micro-architectural Design Exploration: Ability to generate multiple RTL designs from one behavior code under given or changing physical constraints. Generate Trade-off chart between area/resource, latency and timing and provide choices to user for best design as per the requirement.

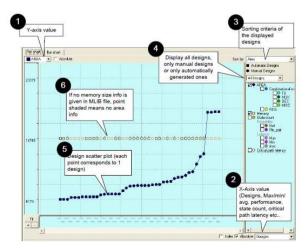


Figure 2-28(a). Micro-architectural design exploration

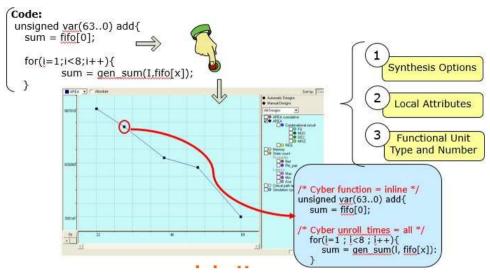
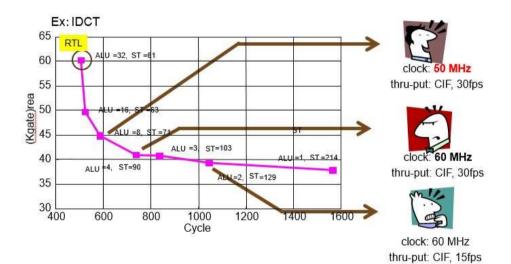


Figure 2-28(b). Micro-architectural design exploration





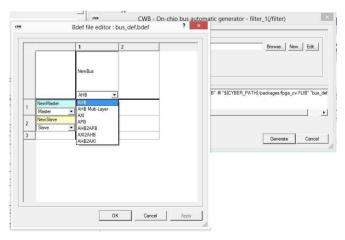
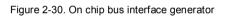


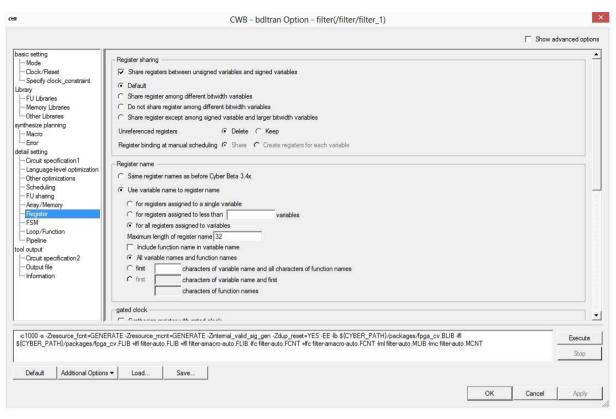
Figure 2-29. On chip bus generator

Input file	P	Browse New Edit
Initial value macro file	1	Browse New Edit
Address macro file	1	Browse New Edit
Output format	Separate process format C Include format	
Mode format	Default C APB slave C AHB slave	(
	Create register to input address pin	
Error output	Standard output	
		Browse
Specify output file prefix	¢ [
Polarity of reset pin	Active High Active Low	
Declare reset pin	Declare O not declare	
Declare clock pin	Declare O not declare	
Port for debugging	C Create C Do not create	



- Clock/Reset - Specify clock_constraint Library -FU Libraries -FU Libraries -FU Libraries -Wemory Libraries -Wemory Libraries -Wemory Libraries -Wemory Libraries -Grout specification1 - C In	Termet operations Termet operations until implementation automatically Jse customized IP (Synopsys DesignWare, Xilinx CORE generator, Altera Megafunctions) mplement operations as single cycle functional units see combinational functional unit ment multi-cycle operations, multi-cycle memory form ulti-cycle operations nput from same register	Show advanced optic
Mode Mode Clock/Reset Specify clock_constraint Cu ubranes Chu Ubranes	Determine functional unit implementation automatically Jse customized IP (Synopsys DesignWare, Xilinx CORE generator, Altera Megafunctions) mplement operations as single cycle functional units Jse combinational functional unit ment multi-cycle operations, multi-cycle memory- for multi-cycle operations	
Other optimizations input: Scheduling input: FU sharing input: Array/Memory C in Array/Memory C in FSM C in Loop/Function Multip Peeline Oduput Octput file C Array Information schedi	Input from multiplexers with latches Input from multiplexers with inverse phase clock registers Inp	
-c1000 -s -Zresource_fcnt=GENERATE -Zre \${CYBER_PATH}/packages/fpga_cv.FLIB -	resource_mont=GENERATE-Zintemal_valid_sig_gen-Zdup_reset=YES-EE +lb \$(CYBER_PATH)/packages/fpga_cv.BLB +f1 +f1 filter-auto.FLIB +f1 filter-amacro-auto.FLIB +fc filter-auto.FCNT +fc filter-amacro-auto.FCNT +m1 filter-auto.MLIB +mc filter-auto.MCNT	Stop

Figure 2-31. Operator Level Optimization

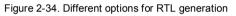




	CWB - bdltran Option - filter//filter/filter_1)	
	T Show a	dvanced optio
oasic setting Mode	Initial values at variable declaration	
Gock/Reset Specify clock_constraint Library FU Libraries	Reset state creation	
- Memory Libraries - Other Libraries	Reset for register	
synthesize planning Macro Error	Image: Construct Project Proje	
detail setting Circuit specification 1	Initialize shared registers in current process	
Language-level optimization Other optimizations Scheduling FU sharing Array/Memory Register FSM	Initialize uninitialized global signal(s). (except for output signal(s) and global register array(s)) (or Initialize uninitialized output signal(s). Initialize uninitialized output signal(s). Initialize uninitialized global register array(s).	
Loop/Function Pipeline cool output Circuit specification2	Clock/Reset for shared registers Clock/Reset for shared registers as that of lower module which does write operation C Set current module of clock/reset signal as input of shared registers	
- Output file Information	Clock/Reset for shared memories Ocok/Reset for shared memories Set the same clock/reset signal of shared memories as that of lower module which does write operation Set current module of clock/reset signal as input of shared memories	
	Reset to submodule Specify reset signal to be connected to module with only one reset port Reset signal name	
	RATE -Zresource_mont-GENERATE -Zntemal_valid_sig_gen -Zdup_reset=YES -EE +b \$(CYBER_PATH)/packages/fpga_cv BLIB +fi \$(CYBER_PATH)/packages/fpga_cv FLIB +fi .FLIB #C filter-auto.FCNT +fic filter-amacro-auto.FCNT +mi filter-auto.MLIB +mc filter-auto.MCNT	Execute
Default Additional Options	Load Save	
	OK Cancel	Apply

Figure 2-33. Options for reset state behavior defining

	CWB - Ve	eriloggen Option - filter(/filter/filter_2)	
eneral Hierarchy Format			
Target	C Simulation	C Logic Synthesis	C Design Compiler
	C Synplify/Certify	C Leonardo Exemplar	
Multiplexer output format		Interfault C case	C casex C AND-OR C if-else
default clause in the case statement		append if necessary	C change last item to default
Value to be assigned at default clau	se in the case statement	Undefined value X	C 0
Reset mode of FF with initial value		Default	C Async reset with set port C Async reset without set port
Output line number		(Yes	C No
Output tool options as comments		Yes	C No
Output empty lower hierarchy module		Yes	C No
FU output file		Same file	C Another file (OP.v)
Create separate register clock pin sync	hronous with reverse phase		
Add pragma about clock and reset info	mation		
Multiple internal memory of same type		Same module	C Separate module
	h module		
NOT operation on clock for synchronous m	emory	 current module 	under layer module
 Output memory model if it is synthesized 	d as memory by logic synthesis tool		
Cutput memory simulation description			
Expand multi-bit port of memory into mu	Itiple one-bit ports		
Specify module instance name in false	path script for Design Compiler		
Multi bit pin name format at design com	piler false path script		
			Exe
2 A A A A A A A A A A A A A A A A A A A			
Default Additional Options - Loa	id Save		
			OK Cancel Ac



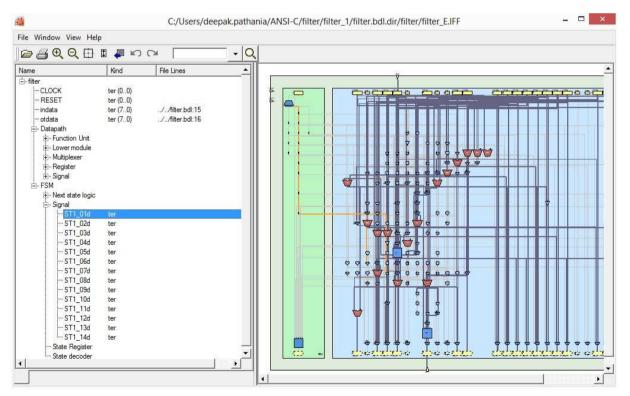
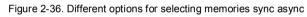


Figure 2-35. RTL Schematic Viewer

CWB - bdltran Option - filter//filter_1)	2
[Show	w advanced optio
Memory KND DATA WIDTH R1 Memory accese Potaut RW1 Allocate to RW2 chronous read access and synchronous wite access (pipelined memory) Allocate to RW2 chronous read access and synchronous wite access (pipelined memory) Allocate to Revize Imaginary to memory with byte-enable pott The register is inserted before and after the access of the pipeline memory. Imput side Output side Position of register before and after shared pipeline memory. C The register is inserted in the module under the synthesis. C Its synthesizes it assuming that the register is on the outside. Pregister array Register array Register array Cetate read enable for multiplexor(s) for address pot of memory without read enable and chip select Cate read enable for multiplexor(s) for address pot of shared register array(LUTRAM style)	
RATE -Zresource mont=GENERATE -Zintemal valid sig gen -Zdup reset=YES -EE +b \${CYBER PATH}/packages/fpga cv.BLIB +fl \${CYBER PATH}/packages/fpga cv.BLIB +fl	Execute
	Memory KIND DATA WIDTH R1 Memory accele R2 Image: Construction of the second second second synchronous write access (pipelined memory) Allocate to RW1 Map array to memory with byte-enable port The register is inserted before and after the access of the pipeline memory. Image: Inserted before and after the access of the pipeline memory. Image: Inserted before and after the register is inserted in the module under the synthesis. Image: Register array Register array Register array Register array Register array Shared array Image: Context of the number of the memory without read enable and chip select.



<u>cinz</u>	CWB - bdltran Option - filter/filter_1)	×
	T Show ad	vanced options
basic setting 	Memory KIND DATA WIDTH R1 Memory acces RW1 Memory acces RW1 Allocate to Register before and after shared synchronous wite access Postion of register before and after shared pipeline memory. C The register is inserted in the module under the synthesis. Image: Register array Register array Register array Register array Register array	
-c1000 -s -Zresource_fort=GENE filter-auto_FLIB +HT filter-amacro-aut Default Additional Option	ERATE -Zresource_monti-GENERATE -Zintemal_valid_pig_gen -Zdup_reset=YES -EE 4b \$(CYBER_PATH)/packages/fpga_ov BLIB 4I \$(CYBER_PATH)/packages/fpga_ov FLIB +H to FLIB 4c filter-auto FCNT +fc filter-auto ALIB -inc filter-auto MCNT 	Execute Stop Apply

Figure 2-37. Different options for selecting memory register types

No	Елгог Туре	Detail	<u> </u>
007	I_BT4607	for loop is unrolled [Source Lines] 40(/./filter.c): for (ky = 0; ky < KSIZE; ky++) { 41(/.filter.c): for (kx = 0; kx < KSIZE; kx++) {	
800	I_BT4466	The sequence of the operation in unrolled loop is optimized, and number of stages of operation is reduced. [Source Lines] 43(/,filter.c): sum += in image[v+kv-1][x+koc-1] * coeff][kv][kx];	
011	I_BT4619	Since the number 4896 of statements after unrolling is large(> 512), loop cannot be unrolled. [Action] In the case of unrolling, specify attribute, or the option (-U# or -UN). [Source Lines] [S0(/.filter.c); for (x = 0; x < HLEN; x++) {	
012	I_BT4619	Since the number 4320 of statements after unrolling is large(> 512), loop cannot be unrolled. [Action] In the case of unrolling, specify attribute, or the option (-U# or -UN). [Source Lines] 29(,filter.c): for (y = 0; y < VLEN; y++) {	
012	L DT4442	Array signal coeff is expanded at dimension #2 from right.	•

Figure 2-38. Speculations suggestions for optimizations

Automatic top module generator to integrate different modules of (c-designed module or existing RTL designs modules) and able to define top module in C as per the user requirement.

General Behavio	orlevel			É
Target project	/filter/filter_1			
Output file name	[.BDL	
Output directory				
Top module name	[
Description level fo	rtop module 🛛 🙃 I	pehavior C st	ructure	
🖵 Do SADL trans	form that negede FF clo	ock separation		
Specify monitor tap	definition file			
Create monitor	, tap pin for unused shar	ed register		
Create FCTS o				
- Module List				
				1.1
Module Name	Input Level	Generate Port		
Module Name	Input Level	Generate Port		
Module Name				
[] filter	r Behavior r Str	ucture F Clock F Reset		
[] filter	r Behavior r Str	ucture F Clock F Reset		-
[] filter	r Behavior r Str	ucture F Clock F Reset		-
[] filter	r Behavior r Str	ucture F Clock F Reset		
[[filter]	r Behavior r Str	ucture F Clock F Reset		
	r Behavior r Str	ucture F Clock F Reset		
Select all	r Behavior r Str	ucture F Clock F Reset		

Figure 2-39. Top Module Generator

Verification

Automated Test bench generator for simulation and synthesis and ability to generate test bench at behavioral level and automatic conversion of those test vector for RTL. Automated SystemC cycle accurate model generation for testing module level as well as system-level (top level).

	Testbench Simulation
_ Te	stbench
¢	Auto
	 Compare output value with the expected scenario
	The name of scenario: bsim
	C Write input/output value to the file
	When to write: ፍ Every cycle 🛛 🧲 Every valid cycle
Bui	ild/Execution
(•	Generate simulation script(3)
	C ModelSim SE C ModelSim C VCS C Riviera C NC-Verilog C Verilog-XL @ Icarus Verilog C Vivado Simula
	🔽 Run simulator in 64-bit mode
0	Specify simulation script(4)
-	
- 00	omment
1	
<u> </u>	
1	
1	

Figure 2-40. Verification Automated script generation

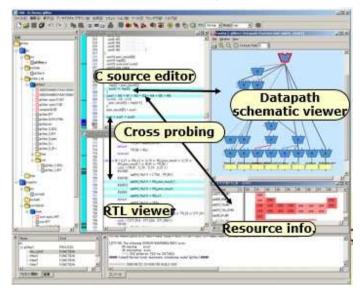
	InputScenario	UNAITIE	
Enter a Simulation so ter	cenario name for the 'filter'		
-			
Simulation Level		a	
	Cycle accurate RTL C on 	board	
	Alexandra and a second and a second sec		
	C SystemC co-simulation		
	SystemC co-simulation		

Figure 2-41(a). Verification Automated TestBench Generation

Main Model generate Testbench Build Simulation Debugger		
Input Data		
How to provide data to the input data		
Random data From pattern file From user function		
When to provide data		
Every cycle C Every valid cycle		
Create separate test bench file :	.cpp	
Do not overwrite separate test bench file		
Data Radix of Pattern File		
Hexadecimal C Decimal		
Compare (backwards-compatible option)		
🔽 It follows the settings of the expected scenario		
When to compare the value of output		
Every cycle C Every valid cycle		
		1
ut=random:cycle -org_type=pin -enum_int=NO -out_dir=.	Generate	m
	Testbe	
Default Additional Options - Load Save		
Conduct in addition of the conduct		

Figure 2-41(b). Verification Automated TestBench Generation

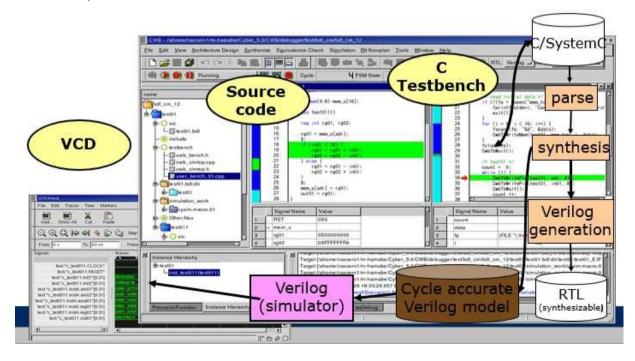
Automated verifications or comparisons of results of C behavioral source code results with Cycle accurate or RTL simulations results and display of results. Automated script generation for third party RTL simulators for performing RTL simulations of the designs. Cycle-Accurate SW/HW testing and co-simulation C source code Debugging of the RTL code, Cross Probing between C code and RTL code, C source code testing and verification.



Main Model	generate Testbench Build Simulation Debugger	
-Generate Det	bug Model	
	C Yes	
- Testbench -		
C Specify		
S Specity	J. Ailter_tb.c	
1.1	La .7mile _to.c	
Sele		
	e input/output value to the file	
	en to write: 🌀 Every cycle 🦵 Every valid cycle	
Auto		
	pare output value with the expected scenario	
	name of scenario:	
	e input/output valu	
Whe	en to write: F Event	
	rtlsim	4
Build	filter	
	arate Makefile(1) Specify command / Script(2)	
	Build	
	Clean	
-Execution		31
100000000000000000000000000000000000000	in the second (2)	
	e simulation script(3)	
C Specify s	simulation script(4)	
1		

Figure 2-43. Different options for automated verification comparison of results of C

Ability to use Transaction Level (TLM) Test Vectors for cycle accurate simulations as well as RTL level simulations (Same test vectors for software can be re-used for cycle accurate simulations).



Main Model generate Testbench Build Simulation	
_ Testbench	
C Specify	
File name filter_behav.cpp	Add
¹ □ □	
Select all All non-select Auto	
The name of scenario:	
Write input/output value to the file	
When to write: C Every cycle	cle
Build	
Auto genarate Makefile(1) C Specify command / Script(2))
Build	
- Execution	
 Generate simulation script(3) 	
C Specify simulation script(4)	
1	
d from Default Save to Default	

Figure 2-44. Verification C Source Code Debugger

Figure 2-45. Different options for C Source Code Testing Verification

Main Model generate Test	pench Build Simulation Debugger	
Generate Debug Model		1
- Testbench		
C Specify		
i 🛄 🔛 , //filter_tb.c		
Select all All non	rselect	
	ero menie ery cycle C Every valid cycle	
Auto	ciy cycle 🕐 Every valu cycle	
Compare output value v	with the expected scenario	
The name of scenario:	· · · · ·	
Write input/output valu		
When to write: 🕫 Ev	bam	
	csim rtlsim	
Build	filter	
	C Specify command / Script(2)	
Build		
Clean		
Execution		10
Generate simulation script(3)	
C Specify simulation script(4)		
 opecity antiduon actipt(4) 		
1		
		- AN

Figure 2-46. Cycle Accurate Software-Hardware Testing Co-simulation

Integration with Third Party Tools

Output generated should be compatible with other synthesis tools like Design Compiler, ISE, Vivado, Synplify, Quartus etc. Ability to generate scripts for automatically invoking tools both for Command Line User Interface (CUI) and Graphical User Interface (GUI). Provide waveform viewer such as GTKWave integrated in the tool (With no additional license requirement). OSCI (Open SystemC Initiative) simulator for cycle-accurate simulations integrated in the tool.

-Wave view GUI/debug		gtkwave(CWB)	•		
Property ch	5	gtkwave(CWB)	-		
Install path					
gtkwave	*				
nWave se	m Files (x86)\cyber\5.50/ verport 1025	025-65534)			
-Command o Symplify	firectory of logic synthesis	s tool		/synplify(_pro)	- F
ISE	D:/Xilinx_ISE/14.7/ISE	DC/ICE Ave ArtCA		/ise	
	D.774110_13E7 14.7713E				
Vivado				/vivado	
Quartus II				/quartus	

Figure 2-47. Different options for GTKWave Integration

Devic F famih devic pack	The second se	Settings
Top proce	ess filter	
Logic syn	synthesis tool script G Generate C Specify synthesis tool script file dd timing exceptions file	

Figure 2-49. Different options for Output generated compatibility