\Orchestrating a brighter world NEC

C-based High Level Synthesis and Verification Tool Set for ASIC / FPGA

CyberWorkBench®

Pioneering C-based LSI Design

NEC

Behavioral C-Based Synthesis and Verification with CyberWorkBench

Taping out commercial chips since 1993

developed All-in-C

Technical Report

2025

Table of Contents

List of Figures

List of Abbreviations

ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
BIP	Behavioral Intellectual Property
CPU	Central Processing Unit
CWB	CyberWorkBench
DSP	Digital Signal Processing/Processor
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HLS	High-Level Synthesis
IDE	Integrated Design Environment
IP	Intellectual Property
LSI	Large Scale Integration
QoR	Quality of Results
RTL	Register Transfer Level
SoC	System-on-Chip
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VLSI	Very Larger Scale Integration

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.

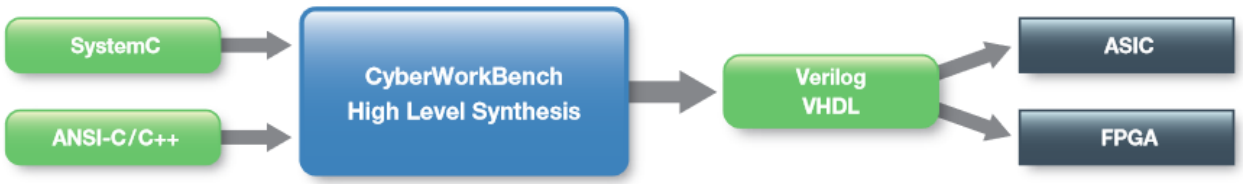

Figure 1 High-Level Synthesis with CyberWorkBench overview

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 - CyberWorkBench

NEC has been developing a C-based behavioral synthesis tool called CyberWorkBench (a.k.a Cyber or CWB) since the late 80's.

Initially this tool was developed as an in-house tool for the semiconductor division at NEC, later to become NEC electronics. This has led to building one of the most advanced HLS tool due to its actual use by VLSI design engineers "sitting" next to the CWB development team. Since 2005, CWB has been made available to outside customers around the world and is now one of the leading HLS tools.

CWB allows to fully design and verify complete heterogeneous System-on-Chips (SoCs) in C. We call this ***All-in-C***.
This includes being able to synthesize C, C++, C with hardware extensions called BDL and SystemC as well as verifying this through different types of simulation models (behavioral, cycle-accurate, and RTL) and also including C-based property checking.

The ***All-in-C*** philosophy is built around two principal ideas:

1. **All-in-C Synthesis:** This 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**: This 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.

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 on-chip 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 cycle-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

property/assertion model checker tool. Global properties and in-context (immediate) assertions are described for/in the C source code.

All-in-C Synthesis

The CWB design flow is shown in Figure 2. 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 model in C++ or SystemC. The behavioral synthesis can therefore be considered as a Verilog, VHDL, C, C++, and SystemC unification step.

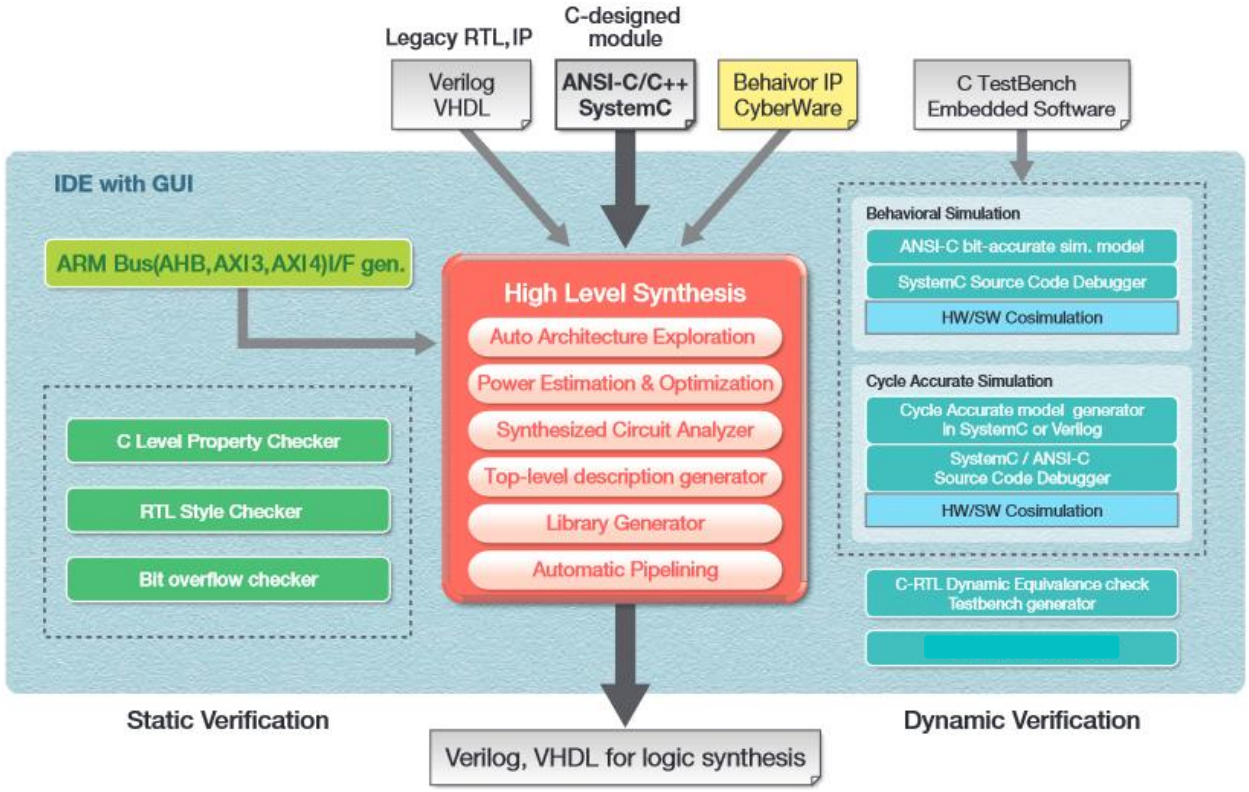

Figure 2 CyberWorkBench detailed overview

All-in-C Verification

CWB allows you to verify the behavioral description in different ways:

1. Simulation-based
2. Source code debugging

Simulation-based Verification

The functionality of the hardware described in C can be verified at different levels of abstraction. Figure 3 shows the different simulation models that CWB supports. In particular:

Behavioral untimed simulation: Pure C/C++ Simulation through gcc/g++ and bit-accurate simulation to simulate custom data types in Cyber-C or SystemC

Cycle-accurate simulation: Generates a cycle-accurate SystemC model of the synthesized circuit. This has several advantages over RTL simulations:

1. It is faster (10-100x)
2. It does not require an RTL license as it generates SystemC which then compiled with g++
3. The generated model can be integrated in any virtual platform environment for full chip simulation.

RTL simulation: CWB comes with an RTL testbench generator that generates the simulation scripts for most commercial and even open-source RTL simulators like modelsim, vcs, isim and iverilog.

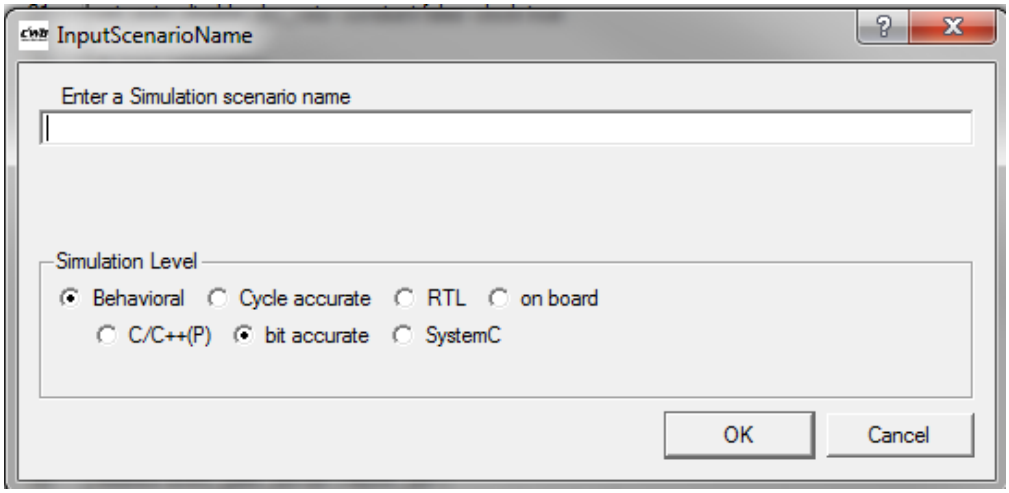

Figure 3 Different simulation models generated by CWB

One of the main advantages of this integrated verification flow is that every simulation model from untimed behavioral to RTL allows to fully re-use the untimed test vectors from the original behavioral description.

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.

Source Code debugger

One of the very unique features of CWB is that its IDE allow to debug the C source code like any traditional software debugger. Figure 4 shows the in-built debugger.

The main difference between this debugger and a SW debugger is that the user looks at the untimed C code, but CWB runs at the background the already synthesized code. Hence, when a breakpoint is reached, the debugger stops and highlights at the debugger all the lines of C code being “executed” in the clock cycle. The debugger basically steps cycle-by-cycle showing which part of the description are being executed.

As shown in the figure below, the debugger also outputs a cycle-accurate waveform that gets updated every clock cycle while the debugger advances.

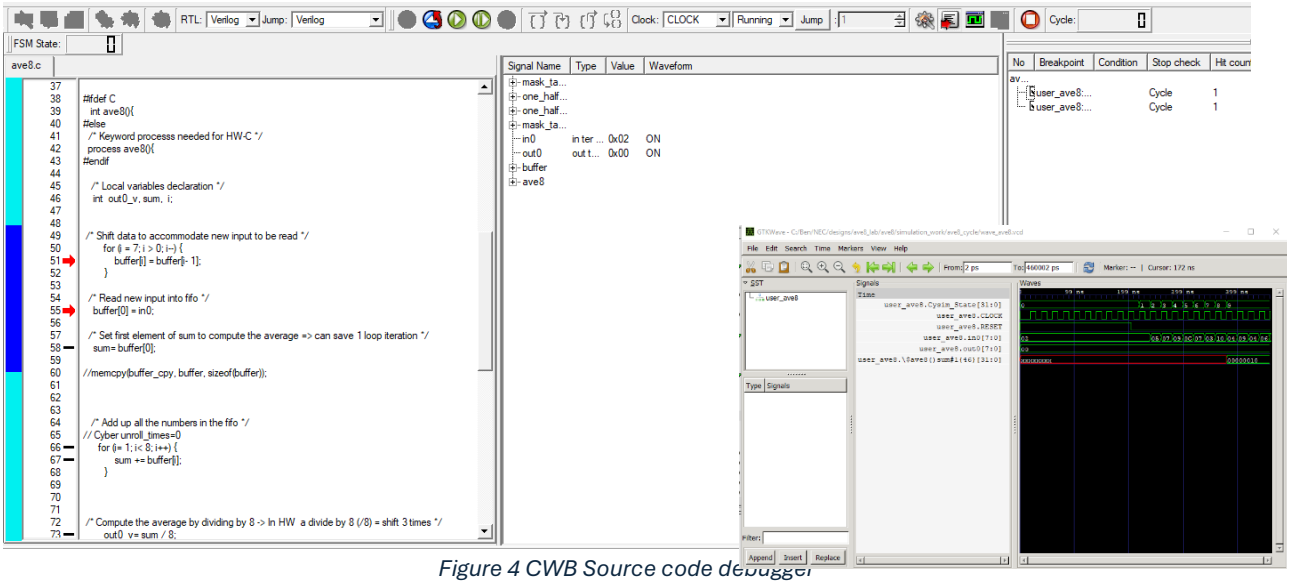

Figure 4 CWB Source code debugger

Advantages of Behavioral Synthesis

The next subsections describe 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.

Figure 5 summarizes these graphically in terms of code size reduction (increase in productivity), reduce verification time (simulation hours), while leading to high-quality RTL designs compared to hand coded RTL. Other advantages include code re-usability and the ability to generate effortless functional equivalent design variants with different architectural trade-offs.

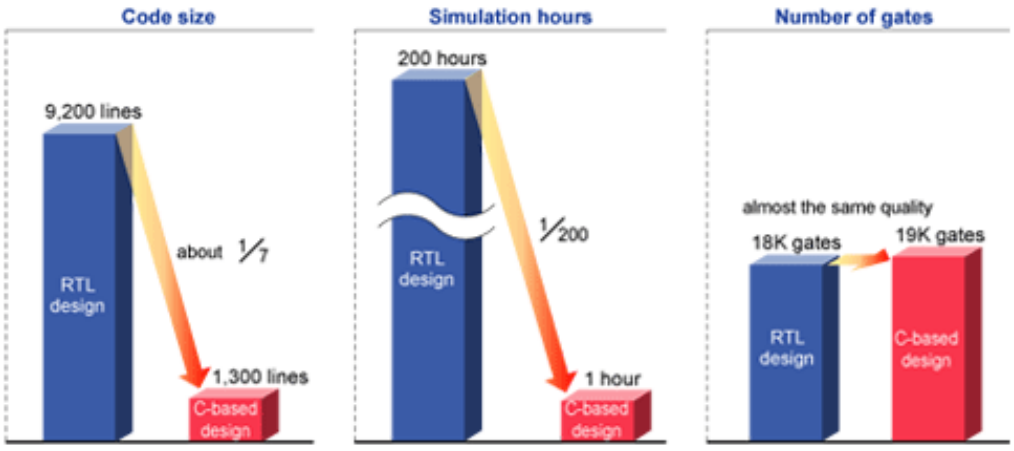

Figure 5 Advantages of raising the level of VLSI design abstraction from RTL to behavioral

Shorter Design Cycles

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 men- month 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 which 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.

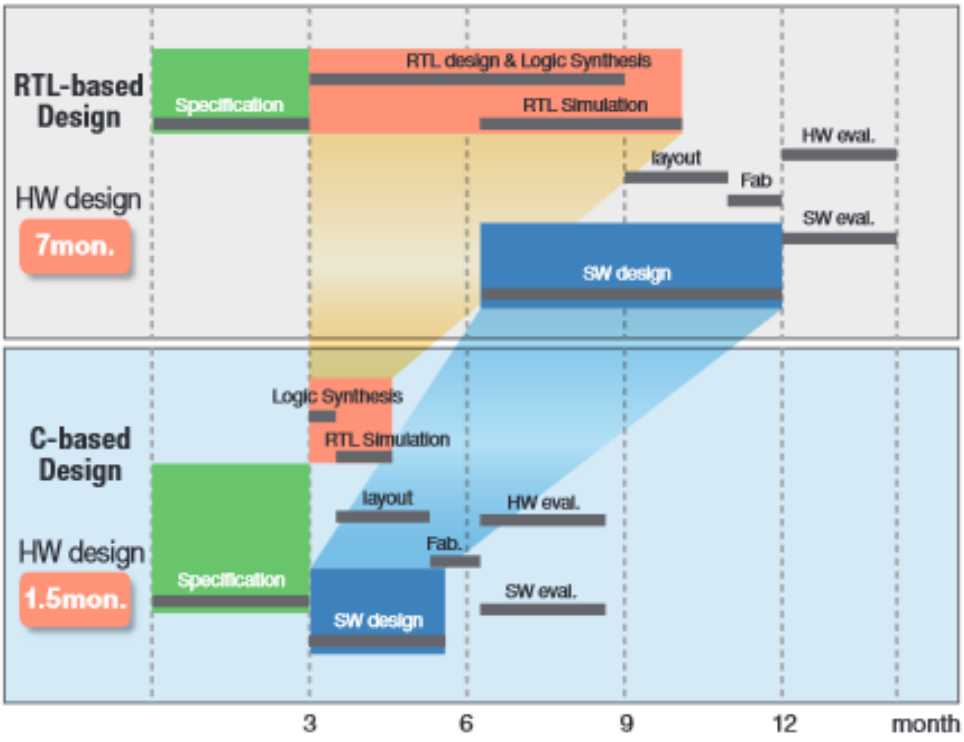

Figure 6 Shorter Design Cycles because of HLS

Source Code Reusability – 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 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.

Table 1 BS broadcast descrambler behavioral IP comparison

Clock Frequency	Generated Gate Size	Generated RTL size	Performance
33 MHZ	57 KGates	7.0 Klines	80 Mbps
54 MHz	42 KGates	5.9 Klines	80Mps

108 MHz	26 KGates	2.5 Klines	80 MPs

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

Behavioral IPs sometimes generate 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 Design

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 base-processors 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 Architectural Design Space 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 meet 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

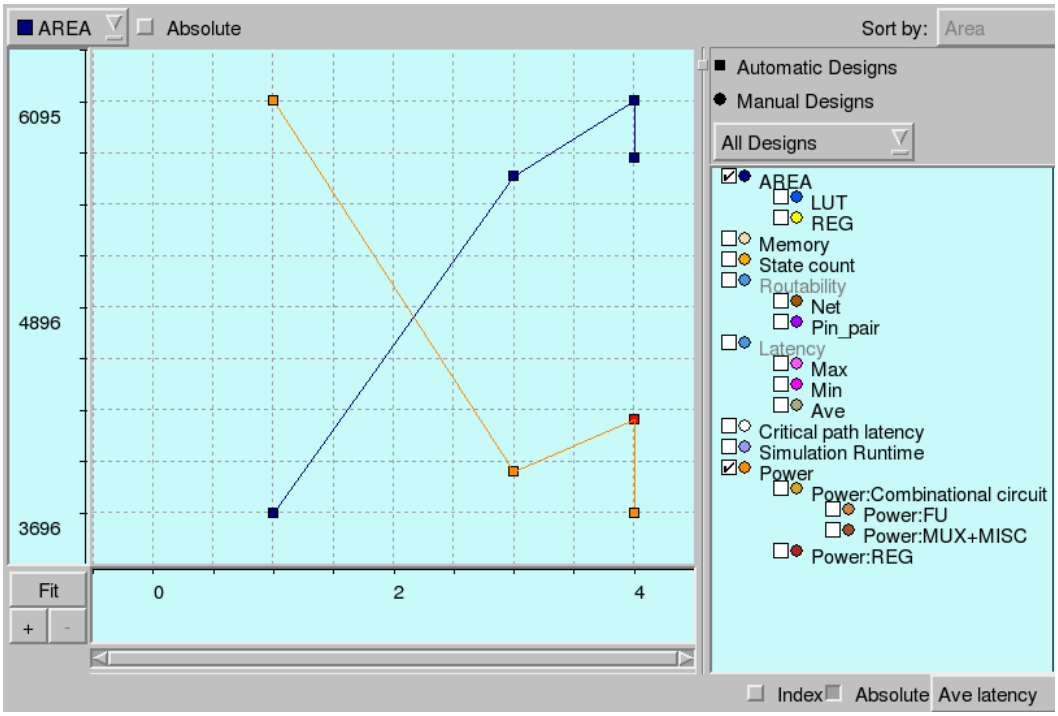

Figure 7 CyberWorkBench's design space exploration window

System VLSI Design Example using CyberWorkBench

Figure 8 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.

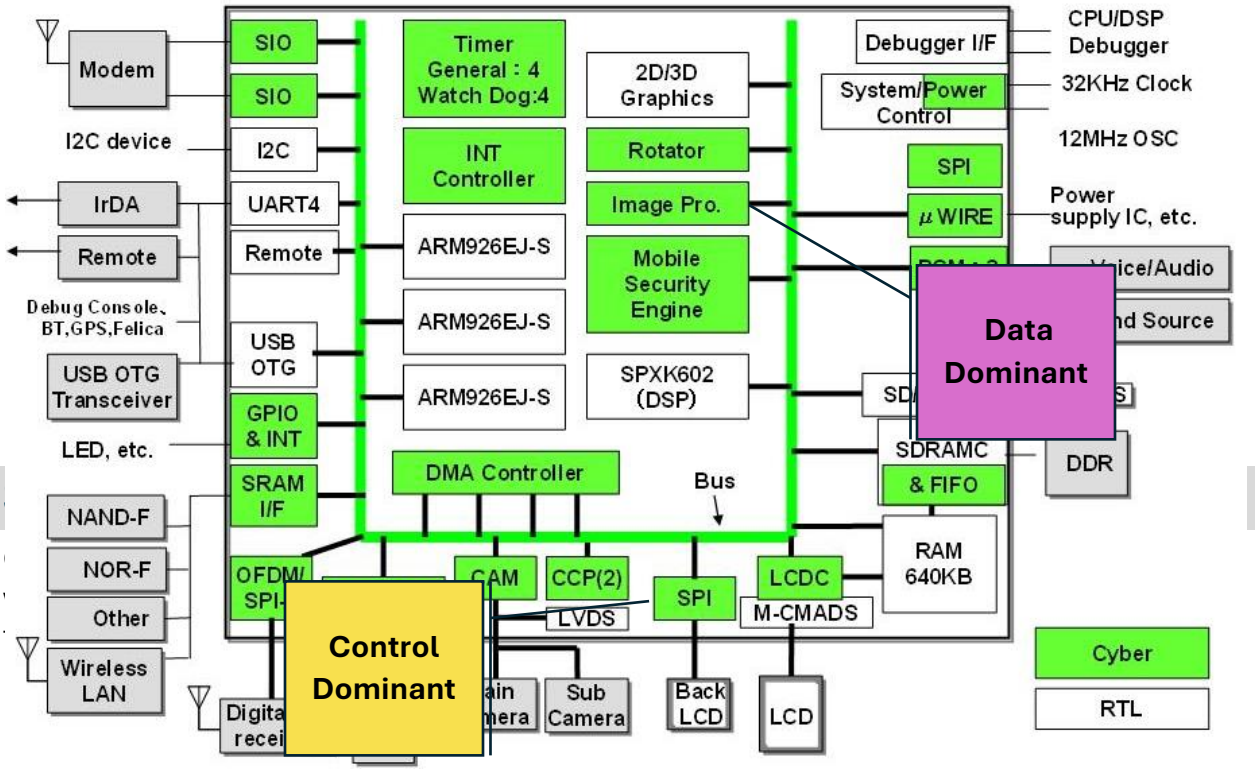

Figure 8 Example of Complete SoC designed and verified using CWB

CWB Targets ASICs and FPGAs

CWB targets any ASIC technology as well as any FPGA, as it comes with a library characterizer that allows to re-generate accurate area and timing information no matter what target platform and technology is being used.

Table 2 shows the product line up and the supported devices. One of the key differences between the ASIC and the FPGA versions is that the FPGA versions are tightly integrated with the FPGA vendors' synthesis tools including their IP generators. This allows to fully leverage their IPs from within CWB.

Table 2 CyberWorkBench product line up targeting ASICs and FPGAs

Product Name	CyberWorkBench®			
	Enterprise	Professional	Standard	Basic
Target Device	ASIC / FPGA	FPGA	FPGA	FPGA
Common Functionality	✔	✔	✔	✔
Option	✔	✔	✔	✔
Input Description Size	No Limit	No Limit	Limited	Limited
How to run				
GUI	✔	✔	✔	✔
Command Line	✔	✔		
Target Device				
ASIC	✔			
Intel Stratix	✔	✔		
XILINX Virtex	✔	✔		
Intel Arria	✔	✔	✔*1	
XILINX Kintex	✔	✔	✔*1	
Intel Cyclone	✔	✔	✔*1	✔*2
XILINX Artix (Spartan)	✔	✔	✔*1	✔*2

Summary and Conclusion

CWB's ***All-in-C*** design and verification platform has been extensively used in commercial chip design over the years. First within NEC and since early 2000's at other companies around the world.

Its uniqueness is that it does not only allow to synthesize any time of module, but also verify it using different verification techniques. The IDE integrates all of these in an intuitive and easy to use graphical environment that allows anyone to understand what the synthesizer has created and go back to the behavioral description to do any necessary changes such that the final RTL code meets the target constraints.