CYPRESS

CY3130

Warp Enterprise™ VHDL CPLD Software

Features

- **VHDL (IEEE 1076 and 1164) high-level language compilers with the following features:**
  - **Designs are portable across multiple devices and/or EDA environments**
  - **Facilitates the use of industry-standard simulation and synthesis tools for board- and system-level design**
  - **Support for functions and libraries facilitating modular design methodology**
  - **Support for enumerated types, operator overloading, For... Generate statements and Integers**
- **Several design entry methods support high-level and low-level design descriptions:**
  - **Graphical HDL Block Diagram editor with a library of blocks and a text-to-block conversion utility from Aldec**
  - **Aldec Active-HDL™ FSM graphical Finite State Machine editor**
  - **Behavioral VHDL (IF...THEN...ELSE; CASE...)**
  - **Boolean**
  - **Structural VHDL**
  - **Designs can include multiple entry methods (but only one HDL) in a single design.**
- **Language Assistant library of VHDL templates**
- **Flow Manager Interface to keep track of complex projects**
- **UltraGen™ Synthesis and Fitting Technology:**
  - **Infers "modules" such as adders, comparators, etc., from behavioral descriptions and replaces them with circuits pre-optimized for the target device.**
  - **User-selectable speed and/or area optimization on a block-by-block basis**
  - **Perfectly integrated synthesis and fitting**
  - **Automatic selection of optimal flip-flop type (D type/T type)**
  - **Automatic pin assignment**
- **Support for all Cypress Programmable Logic Devices**
  - **PSI™ (Programmable Serial Interface™)**
  - **Delta39K™ CPLDs**
  - **Quantum38K™ CPLDs**
  - **Ultra37000™ CPLDs**
  - **FLASH370i™ CPLDs**
  - **MAX340™ CPLDs**
  - **Industry standard PLDs (16V8, 20V8, 22V10)**
- **VHDL or Verilog timing model output for use with third-party simulators**
- **Timing simulation provided by Active-HDL™ Sim Release 4.1 from Aldec**
  - **Graphical waveform simulator**
  - **Graphical entry and modification of all waveforms**
  - **Ability to compare waveforms and highlight differences before and after a design change**
  - **Ability to probe internal nodes**
  - **Display of inputs, outputs, and high-impedance (Z) signals in different colors**
  - **Automatic clock and pulse creation**
  - **Support for buses**
  - **Unlimited simulation time**
- **Architecture Explorer and Dynamic Timing Simulator for PSI, Delta39K and Quantum38K devices:**
  - **Graphical representation of exactly how your design will be implemented on your specific target device**
  - **Zoom from the device level down to the macrocell level**
  - **Determine the timing for any path and view that path on a graphical representation of the chip**
- **Static Timing Report for all devices**
- **Source-Level Behavioral Simulation and Debugger from Aldec**
- **Testbench Generation**
- **UltraISR Programming Cable**
- **Delta39K\Ultra37000 prototype board with a CY37256V 160-pin TQFP device and a CY39100V 208-pin PQFP device[1]**
- **On-line documentation and help**

Functional Description

Warp Enterprise™ is an integration of the *Warp* Professional™ CPLD Development package with additional sophisticated EDA software features from Aldec. In addition to accepting IEEE 1076/1164 VHDL text and graphical finite state machines for design entry, *Warp* Enterprise VHDL provides a graphical HDL block diagram editor with a library of graphical HDL blocks pre-optimized for Cypress devices. Plus, it provides a utility to convert HDL text into graphical HDL blocks. *Warp* Enterprise synthesizes and optimizes the entered design, and outputs a JEDEC or Intel hex file for the desired PLD or CPLD (see *Figure 1*). For simulation, *Warp* Enterprise provides a timing simulator, a source-level behavioral simulator, as well as VHDL and Verilog timing models for use with third party simulators. *Warp* Enterprise also provides the designer with important productivity tools such as a testbench generation wizard and the Architecture Explorer graphical analysis tool.

Note:

1. Cypress reserves the right to substitute prototype boards based on product availability.

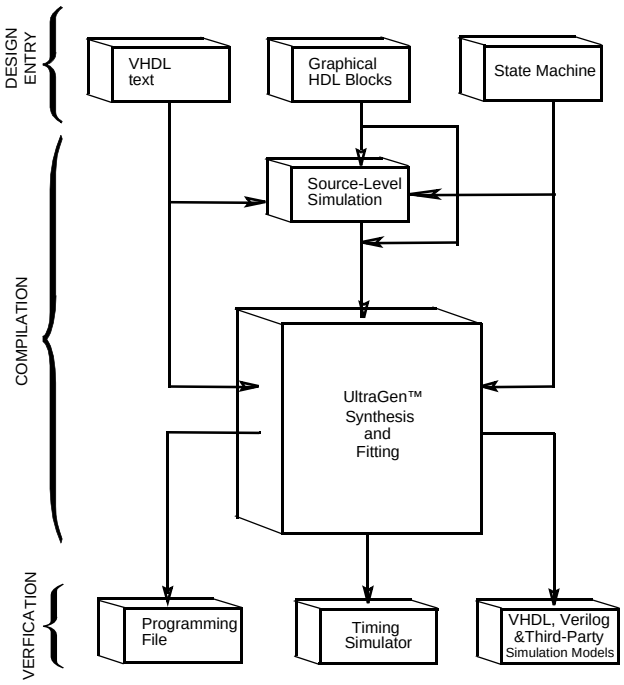

Figure 1. *Warp*® Design Flow

VHDL Compiler

VHDL is a powerful, industry-standard language for behavioral design entry and simulation, and is supported by all major vendors of EDA tools. It allows designers to learn a single language that is useful for all facets of the design process.

VHDL offers designers the ability to describe designs at many different levels. At the highest level, designs can be entered as a description of their behavior. This behavioral description is not tied to any specific target device. As a result, simulation can be done very early in the design to verify correct functionality, which significantly speeds the design process.

The *Warp* syntax for VHDL includes support for intermediate level entry modes such as state tables and Boolean entry. At the lowest level, designs can be described using gate-level descriptions. *Warp* Enterprise gives the designer the flexibility to intermix all of these entry modes.

In addition, VHDL allows you to design hierarchically, building up entities in terms of other entities. This feature allows you to work either “top-down” (designing the highest levels of the system and its interfaces first, then progressing to greater and greater detail) or “bottom-up” (designing elementary building blocks of the system, then combining these to build larger and larger parts) with equal ease.

Because this language is an IEEE standard, multiple vendors offer tools for design entry and simulation at both high and low levels and synthesis of designs to different silicon targets. The use of device-independent behavioral design entry gives users the freedom to easily migrate to high volume technologies. The wide availability of VHDL tools provides complete vendor independence as well. Designers can begin their project using *Warp* Enterprise for Cypress CPLDs and convert to high-volume ASICs using the same VHDL behavioral description with industry-standard synthesis tools.

The VHDL language also allows users to define their own functions. User-defined functions allow users to extend the capabilities of the language and build reusable files of tested routines. VHDL provides control over the timing of events or processes. It has constructs that identify processes as either sequential, concurrent, or a combination of both. This feature is essential when describing the interaction of complex state machines.

VHDL is a rich programming language. Its flexibility reflects the nature of modern digital systems and allows designers to create accurate models of digital designs. Because it is not a verbose language it is easy to learn and compile. In addition, models created in VHDL can readily be transported to other EDA Environments. *Warp* Enterprise VHDL supports IEEE 1076/1164 VHDL including loops, for/generate statements, full hierarchical designs with packages, enumerated types, and integers.

A VHDL Design Example

Design Entry

Warp Enterprise descriptions specify:

- The behavior or structure of a design, and
- The mapping of signals in a design to the pins of a PLD/CPLD (optional)

The part of a *Warp* Enterprise description that specifies the behavior or structure of the design is called an entity/architecture pair. Entity/architecture pairs, as their name implies, are divided into two parts: an entity declaration, which declares the design’s interface signals (i.e., defines what external signals the design has, and what their directions and types are), and a design architecture, which describes the design’s behavior or structure.

The entity portion of a design file is a declaration of what a design presents to the outside world (the interface). For each external signal, the entity declaration specifies a signal name, a direction and a data type. In addition, the entity declaration specifies a name by which the entity can be referenced in a design architecture. This section shows code segments from five sample design files. The top portion of each example features the entity declaration.

Behavioral Description

The architecture portion of a design file specifies the function of the design. As shown in *Figure 1*, multiple design-entry methods are supported in *Warp* Enterprise. A behavioral description in VHDL often includes well known constructs such as If...Then...Else, and Case statements. Here is a code segment from a simple state machine design (soda vending machine) that uses behavioral VHDL to implement the design:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY drink IS
 PORT (nickel,dime,quarter,clock:#in
std_logic;
   returnDime,returnNickel,giveDrink:out

std_logic);
END drink;
```

```
ARCHITECTURE fsm OF drink IS

TYPE drinkState IS (zero,five,ten,fifteen,
twenty,twentyfive,owedime);
SIGNAL drinkstatus:drinkState;

BEGIN

PROCESS BEGIN

 WAIT UNTIL clock = '1';
  giveDrink <= '0';
  returnDime <= '0';
  returnNickel <= '0';

  CASE drinkStatus IS

  WHEN zero =>
    IF (nickel = '1') THEN
      drinkStatus <= five;
    ELSIF (dime = '1') THEN
      drinkStatus <= Ten;
    ELSIF (quarter = '1') THEN
      drinkStatus <= twentyfive;
    END IF;
  WHEN five =>
    IF (nickel = '1') THEN
      drinkStatus <= ten;
    ELSIF (dime = '1') THEN
      drinkStatus <= fifteen;
    ELSIF (quarter = '1') THEN
      giveDrink <= '1';
      drinkStatus <= zero
    END IF;

  -- Several states are omitted in this
  -- example. The omitted states are ten,
  -- fifteen, twenty, and twentyfive.

  WHEN owedime =>
    returnDime <= '1';
    drinkStatus <= zero;

  when others =>
  -- This makes sure that the state
  -- machine resets itself if
  -- it somehow gets into an undefined state.
    drinkStatus <= zero;
  END CASE;
  END PROCESS;

END FSM;
```

VHDL is a strongly typed language. It comes with several predefined operators, such as + and /= (add, not-equal-to). VHDL offers the capability of defining multiple meanings for operators (such as +), which results in simplification of the code written. For example, the following code segment shows that “count <= count +1” can be written such that count is a std_logic_vector, and 1 is an integer.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;

ENTITY sequence IS
  port (clk: in std_logic;
    s : inout std_logic);
end sequence;

ARCHITECTURE fsm OF sequence IS

SIGNAL count: std_logic_vector(3 downto 0);

BEGIN

PROCESS BEGIN

  WAIT UNTIL clk = '1';
    CASE count IS

      WHEN x"0" | x"1" | x"2" | x"3" =>
        s <= '1';
        count <= count + 1;
      WHEN x"4" | x"5" | x"6" | x"7" =>
        s <= '0';
        count <= count + 1;
      WHEN x"8" | x"9" =>
        s <= '1';
        count <= count + 1;
      WHEN others =>
        s <= '0';
        count <= (others => '0');
    END CASE;

END PROCESS;

END FSM;
```

In this example, the + operator is overloaded to accept both integer and std_logic arguments. *Warp* Enterprise supports overloading of operators.

Functions

A major advantage of VHDL is the ability to implement functions. The support of functions allows designs to be reused by simply specifying a function and passing the appropriate parameters. *Warp* Enterprise features some built-in functions such as ttf (truth-table function). The ttf function is particularly useful for state machine or look-up table designs. The following code describes a seven-segment display decoder implemented with the ttf function:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.table_std.all;

ENTITY seg7 IS
  PORT(
    inputs: IN STD_LOGIC_VECTOR (0 to 3)
    outputs: OUT STD_LOGIC_VECTOR (0 to 6)
  );
END SEG7;

ARCHITECTURE mixed OF seg7 IS

CONSTANT truthTable:
  ttf_table (0 to 11, 0 to 10) := (
-- input&    output
-- ----------------------
  "0000"&  "0111111",
```

```
  "0001"&  "0000110",
  "0010"&  "1011011",
  "0011"&  "1001111",
  "0100"&  "1100110",
  "0101"&  "1101101",
  "0110"&  "1111101",
  "0111"&  "0000111",
  "1000"&  "1111111",
  "1001"&  "1101111",
  "101-"&  "1111100", --creates E pattern
  "111-"&  "1111100"
  );

BEGIN

    outputs <= ttf(truthTable,inputs);

END mixed;
```

Boolean Equations

A third design-entry method available to *Warp* Enterprise users is Boolean equations. *Figure 2* displays a schematic of a simple one-bit half adder. The following code describes how this one-bit half adder can be implemented in *Warp* Enterprise with Boolean equations:

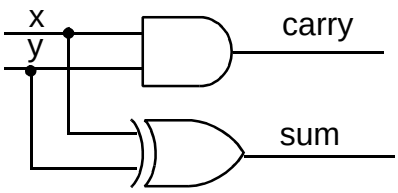

Figure 2. One-Bit Half Adder

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

--entity declaration
ENTITY half_adder IS
  PORT (x, y: IN std_logic;
    sum, carry : OUT std_logic);
END half_adder;
--architecture body
ARCHITECTURE behave OF half_adder IS
BEGIN
  sum <= x XOR y;
  carry <= x AND y;
END behave;
```

Structural VHDL

While all of the design methodologies described thus far are high-level entry methods, structural VHDL provides a method for designing at a very low level. In structural descriptions, the designer simply lists the components that make up the design and specifies how the components are wired together.
Figure 3 displays the schematic of a simple 3-bit shift register and the following code shows how this design can be described in *Warp* Enterprise using structural VHDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.rtlpkg.all;

ENTITY shifter3 IS port (
    clk : IN STD_LOGIC;
    x : IN STD_LOGIC;
    q0 : OUT STD_LOGIC;
    q1 : OUT STD_LOGIC;
    q2 : OUT STD_LOGIC);
  END shifter3;

ARCHITECTURE struct OF shifter3 IS
  SIGNAL q0_temp, q1_temp, q2_temp : STD_LOGIC;
  BEGIN
    d1 : DFF PORT MAP(x,clk,q0_temp);
    d2 : DFF PORT MAP(q0_temp,clk,q1_temp);
    d3 : DFF PORT MAP(q1_temp,clk,q2_temp);
    q0 <= q0_temp;
    q1 <= q1_temp;
    q2 <= q2_temp;
  END struct;
```

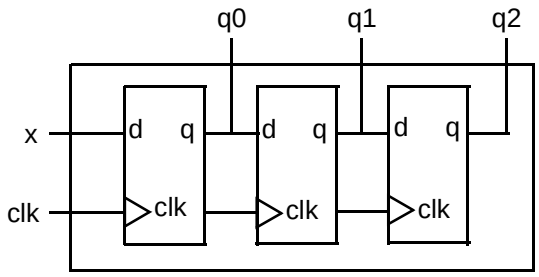

Figure 3. Three-Bit Shift Register Circuit Design

All of the design-entry methods described can be mixed as desired. VHDL has the ability to combine both high- and low-level entry methods in a single file. The flexibility and power of VHDL allows users of *Warp* Enterprise to describe designs using whatever method is appropriate for their particular design.

Finite State Machine Editor

Aldec's Active-HDL FSM finite state machine editor, allows graphic design entry through the use of graphical state diagrams. A design may be represented graphically using state diagrams and data flow logic. This tool will automatically generate the HDL code of the design.

HDL Block Diagram Editor

The HDL block diagram editor lets you represent portions of your code with graphical symbols. This representation allows you to view the high-level structure of your complex designs and lets you copy and paste entire modules of your design within or between designs. The editor comes with a library of HDL blocks optimized for Cypress devices. *Warp* Enterprise comes with a utility that converts HDL text into these blocks.

Language Assistant

The language assistant is a library of language templates that you can browse and automatically insert into your HDL text. They provide syntax and structure and give examples to aid users who are new using a particular HDL.

Flow Manager

The flow manager is a special interface that helps you keep track of your complex projects. It arranges the tools as part of

the logical flow the designer takes through a project and remembers what steps have been completed on which designs.

Source-Level Simulation

Warp Enterprise's source-level behavioral simulator helps you catch problems with your code early in the design process by letting you simulate a design before synthesis. The tool lets you graphically watch inputs and outputs, gives you timing information and allows you to step through your code line by line.

Compilation

Once the VHDL description of the design is complete, it is compiled using *Warp* Enterprise. Although implementation is with a single command, compilation is actually a multistep process, as shown in *Figure 1*.

The first part of the compilation process is the same for all devices. The input description is synthesized to a logical representation of the design. *Warp* synthesis is unique in that the input languages support device-independent design descriptions. Competing programmable logic compilers require very specific and device-dependent information in the design description. *Warp* synthesis is based on UltraGen technology that allows *Warp* Enterprise to infer adders, subtractors, multipliers, comparators, counters and shifters from the behavioral descriptions. *Warp* Enterprise then replaces these operators internally with an architecture-specific circuit. This circuit or "module" is also pre-optimized for either area or speed. *Warp* Enterprise uses the appropriate implementation based on user directives.

The second step of compilation is an iterative process of optimizing the design and fitting the logic into the targeted device. Logical optimization in *Warp* Enterprise is accomplished using Espresso algorithms. The optimized design is automatically fed to the *Warp* Enterprise fitter for targeting a PLD or CPLD. This fitter supports the automatic or manual placement of pin assignments as well as automatic selection of D or T flip-flops. After optimization and fitting, *Warp* Enterprise creates a JEDEC or Intel hex file for the specified PLD or CPLD.

Automatic Error Tracking

Warp Enterprise features automatic error location that allows problems to be diagnosed and corrected in seconds. Errors from compilation are displayed immediately in a window. If the user highlights a particular error, *Warp* Enterprise will automatically open the source code file and highlight the offending line in the entered design. If the device fitting process includes errors, a window will again describe them. A detailed report file is generated indicating the resources required to fit the input design and any problems that occurred in the process.

Timing Simulation

The Aldec Active-HDL Sim post-fitting timing simulator provides timing simulation for PLDs/CPLDs and features interactive waveform viewing as well as graphical creation and editing of all waveforms. The simulator also provides the ability to probe internal nodes, and automatically generate clocks and pulses. The version in *Warp* Enterprise has the ability to compare waveforms and highlight differences before and after a design change. *Warp* Enterprise has unlimited simulation time. To use the timing simulator in *Warp* Enterprise VHDL you must use a VHDL netlist.

Warp Enterprise VHDL will also output standard VHDL timing models. These models can be used with all third-party simulators to perform functional and timing verifications of the synthesized design.

Architecture Explorer

The Architecture Explorer graphically displays how the design will be implemented on the chip. It provides a view of the entire device to show what memory elements and logic clusters have been used for what part of the design. This gives the designer an idea of what resources are free. The Architecture Explorer allows you to zoom in multiple times. At maximum zoom it displays the logic gate implementation in each macrocell. The Architecture Explorer is available for PSI, Delta39K, and Quantum38K devices.

Timing Analyzer

The Timing Analyzer gives the time across any path as well as the breakdown of what steps are causing the timing delays. This tool does not simply display the general specification for the target device but a worst-case simulation of the actual path being taken through the device. When you highlight a path on the timing analyzer, the source and destination of that path are displayed on the Architecture Explorer. The timing analyzer graphical interface is available for PSI, Delta39K, and Quantum38K devices.

Programming

Cypress's FLASH370i, Ultra37000, Quantum38K and Delta39K In-System Reprogrammable™ (ISR™) devices can be programmed on board with an ISR programmer. For PSI, Delta39K, and Quantum38K devices, *Warp* Enterprise produces an Intel hex file. The ISR programmer converts this file into STAPL and programs the device. For Ultra37000 and FLASH370i devices, *Warp* Enterprise produces a JEDEC file. For Ultra37000, the ISR programmer converts this file into JAM/STAPL and programs the device. For FLASH370i, the JEDEC file is used directly to program the device.

Warp Enterprise comes with an UltraISR Programming Cable and a Delta39K\Ultra37000 prototype board with a CY37256V 160-pin TQFP device and a CY39100V 208-pin device.[1]

The JEDEC and Intel hex files produced by *Warp* Enterprise can also be used with any qualified third party programmer to program Cypress CPLDs.

For more information on Cypress's ISR software see the ISR Programming Kit (CY3900i) data sheet.

Warp Software System Requirements

- IBM PC or equivalent (Pentium® class recommended)
- 32 MB of RAM (64 Mbytes recommended)
- 110 MB Disk Space
- CD-ROM drive
- Windows 98, or Windows NT 4.0
- *Warp* Enterprise *for* VHDL Hardware Key

ISR Software PC System Requirements

- IBM PC or compatible running Windows 98, Windows 98 Second Edition, Windows ME, Windows NT 4.0 Service Pack 5 or later, or Windows 2000 Service Pack 1 or later
- One free parallel port
- Minimum of 32 MB of RAM
- Approximately 30 MB free hard disk space

Product Ordering Information

Product Code	Description
CY3130R62	*Warp* Enterprise VHDL CPLD software for PCs

Warp Enterprise includes:

- Cypress Lab CD-ROM with *Warp* Enterprise, ISR software, on-line documentation (Getting Started Manual, User's Guide, HDL Reference Manual, Data Book) and other Cypress software
- UltraISR Programming Cable
- Delta39K\Ultra37000 prototype board with a CY37256V 160-pin TQFP device and a CY39100V 208-pin device[1]
- *VHDL for Programmable Logic* textbook
- Registration Card
- Hardware License Key

Document Title: CY3130 *Warp* Enterprise™ VHDL CPLD Software Document Number: 38-03050				
REV.	**ECN NO.**	**Issue Date**	**Orig. of Change**	**Description of Change**
**	109969	10/24/01	SZV	Change from Spec number: 38-00242 to 38-03050
*A	111245	01/21/02	CNH	Update Product Code and remove references to Windows95