

## Warp2™ VHDL Compiler for PLDs, CPLDs, and FPGAs

### Features

- VHDL (IEEE 1076 and 1164) high-level language compiler
  - Facilitates device independent design
  - Designs are portable across multiple devices and/or CAE environments
  - Facilitates the use of industry-standard simulation and synthesis tools for board and system-level design
  - Supports functions and libraries facilitating modular design methodology
- Warp2™ provides synthesis of IEEE standards 1076 and 1164 VHDL including:
  - enumerated types
  - operator overloading
  - for ... generate statements
  - integers
- Several design entry methods support high and low-level design descriptions:
  - Behavioral VHDL (IF...THEN...ELSE; CASE...)
  - Boolean
  - Structural VHDL (RTL)
  - Designs can include multiple VHDL entry methods in a single design
- State-of-the-art optimizations and reduction algorithms
  - Automatic selection of optimal flip-flop type (D type/T type)
  - Automatic pin assignment
- UltraGen™ Synthesis Technology
  - Infers “modules” like adders, comparators, etc., from behavioral descriptions
  - Replaces operator internally with an architecture specific circuit based on the target device
  - User selectable speed and/or area optimization on a block-by-block basis
- Supports all Cypress Programmable Logic Devices
  - Industry standard PLDs (16V8, 20V8, 22V10)
  - FLASH370i, and MAX340 CPLDs
  - pASIC380 FPGAs
- VHDL and Verilog timing model output for use with third-party simulators
- Functional simulation provided with Cypress NOVA simulator:
  - Graphical waveform simulator
  - Entry and modification of on-screen waveforms
  - Ability to probe internal nodes
  - Display of inputs, outputs, and High Z signals in different colors
  - Automatic clock and pulse creation
  - Waveform to JEDEC test vector conversion utility
  - Support for buses
- PC, Sun (SunOS™, Sun Solaris™), and HP platforms
- Windows 3.1x, Windows 95, and Windows NT

- Motif on Unix workstations
- On-line documentation and help

### Functional Description

Warp2 is a state-of-the-art VHDL compiler for designing with Cypress Programmable Logic Devices. Warp2 utilizes a subset of IEEE 1076 and 1164 VHDL as its Hardware Description Language (HDL) for design entry. Warp2 accepts VHDL input, synthesizes and optimizes the entered design, and outputs a JEDEC map for the desired PLD or CPLD, or a LOF file for the desired FPGA. (see Figure 1). For simulation, Warp2 provides a graphical waveform simulator called NOVA, as well as VHDL and Verilog models for use with third party simulators.

### VHDL Compiler

VHDL (VHSIC Hardware Description Language) is a powerful, non-proprietary language that is a standard for behavioral design entry and simulation, and is supported by every major vendor of CAE tools. VHDL 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.

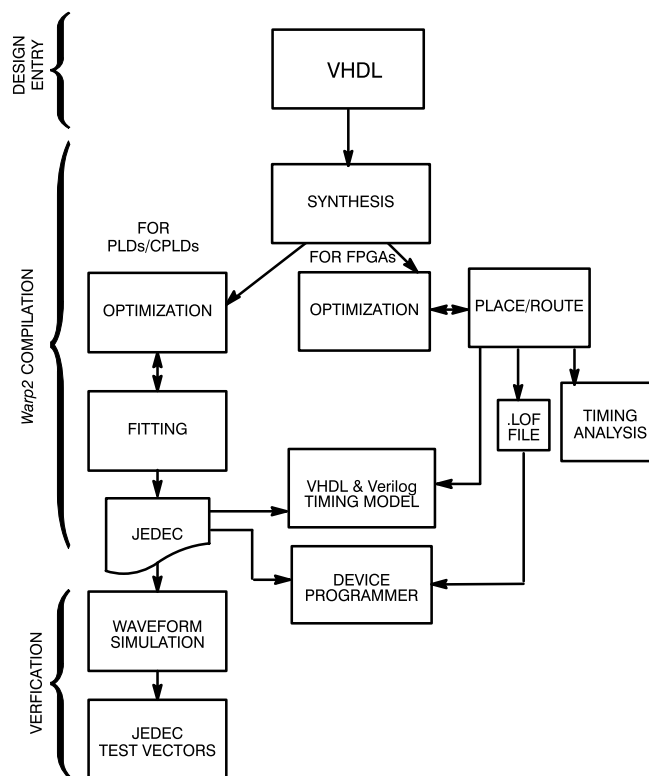


Figure 1. Warp2 Design Flow

*Warp2*'s VHDL syntax also 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 RTL (Register Transfer Language) descriptions. *Warp2* 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 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 VHDL 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 re-target designs to different devices. The wide availability of VHDL tools provides complete vendor independence as well. Designers can begin their project using *Warp2* for Cypress CPLDs and FPGAs and convert to high volume gate arrays using the same VHDL behavioral description with industry-standard synthesis tools.

The VHDL language allows users to define their own functions. User-defined functions allow users to extend the capabilities of the language and build reusable libraries of tested routines. As a result the user can produce complex designs faster than with ordinary "flat" languages. VHDL also provides control over the timing of events or processes. VHDL has constructs that identify processes as either sequential, concurrent, or a combination of both. This 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 of its depth and completeness, it is easier to describe a complex hardware system accurately in VHDL than in any other hardware description language. In addition, models created in VHDL can readily be transported to other CAE Environments. *Warp2* supports IEEE 1076 and 1164 VHDL including loops, for ... generate statements, full hierarchical designs with packages, as well as synthesis for enumerated types and integers.

## Designing with *Warp2*

### Design Entry

*Warp2* descriptions specify

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

The part of a *Warp2* description that specifies the behavior or structure of the design is called an entity/architecture pair. Entity/architecture pairs, as their name implies, can be 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. In this

section are code segments from four 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 *Warp2*. 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 <= drinkStatus'SUCC
                    (drinkStatus);
                    -- goto 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 <= drinkStatus'PRED
                    (drinkStatus);
                    -- goto Zero
                END IF;

            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. *Warp2* 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.

*Warp2* 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

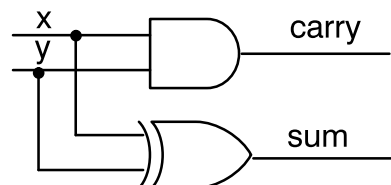
A third design-entry method available to *Warp2* users is Boolean equations. *Figure 2* displays a schematic of a simple one-bit half adder. The following code describes how this one-bit half adder can be implemented in *Warp2* with Boolean equations:

```

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;

```



**Figure 2. One-Bit Half Adder**

```
--architecture body
ARCHITECTURE behave OF half_adder IS
BEGIN
    sum <= x XOR y;
    carry <= x AND y;
END behave;
```

### Structural VHDL (RTL)

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 (also called RTL), 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 *Warp2* 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;
```

All of the design-entry methods described can be mixed as desired. The ability to combine both high- and low-level entry methods in a single file is unique to VHDL. The flexibility and power of VHDL allows users of *Warp2* to describe designs using whatever method is appropriate for their particular design.

### Compilation

Once the VHDL description of the design is complete, it is compiled using *Warp2*. Although implementation is with a single com-

mand, 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 VHDL description is synthesized to a logical representation of the design. *Warp2* synthesis is unique in that the input language (VHDL) supports device-independent design descriptions. Competing programmable logic compilers require very specific and device-dependent information in the design description.

*Warp2* Synthesis is based on UltraGen Technology. This technology allows *Warp2* to infer “modules” like adders, counters, comparators, etc., from behavioral descriptions. *Warp2* then replaces that operator internally with an architecture specific circuit based on the target device. This circuit or “module” is also pre-optimized for either area or speed and *Warp2* uses the appropriate implementation based on user directives.

The second step of compilation is an interactive process of optimizing the design and fitting the logic into the targeted device. Logical optimization in *Warp2* is accomplished using Espresso algorithms. The optimized design is automatically fed to the *Warp2* fitter if the user is 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 the optimization and fitting step is complete, *Warp2* creates a JEDEC file for the specified PLD or CPLD.

If the target device is an FPGA, *Warp2* outputs a QDIF netlist file after optimization that is read into the *Warp2* place and route software, SpDE. SpDE determines the placement of logic in the FPGA and routing of the interconnect that maximizes the speed of operation and minimizes the area utilization of the design. After the place and route is complete, the design timing can be checked by SpDE’s path analyzer, and a LOF file can be output for programming.

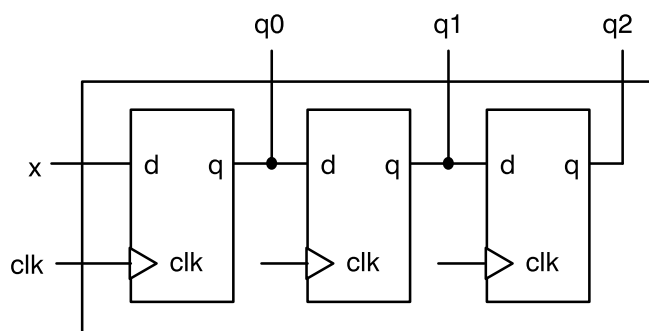
### Simulation

*Warp2* includes Cypress’s NOVA Simulator. NOVA features a graphical waveform simulator that can be used to simulate PLD/CPLD designs generated in *Warp2*. The NOVA simulator provides functional simulation for PLDs/CPLDs and features interactive waveform editing and viewing. The simulator also provides the ability to probe internal nodes, automatically generate clocks and pulses, and generate JEDEC test vectors from simulator waveforms. (Higher level simulation support is available with *Warp3*™ [CY3130].)

*Warp2* will also output standard VHDL and Verilog timing models. These models can be used with many third-party simulators to perform functional and timing verifications of the synthesized design.

### Programming

The result of *Warp2* compilation is a JEDEC or LOF file that implements the input design in the targeted device. Using this file, Cypress devices can be programmed on Cypress’s *Impulse3* programmer or on any qualified third-party programmer.



**Figure 3. Three-Bit Shift Register Circuit Design**



## System Requirements

### For PCs

IBM PC or equivalent (486 or higher recommended)  
16 Mbytes of RAM (32 Mbytes recommended)  
100-Mbyte hard disk space  
CD-ROM drive  
Two- or three-button mouse  
Windows 3.1x, Windows 95, or Windows NT

### For Sun Workstations

SPARC CPU  
SunOS 4.1.3 or Solaris 2.5  
16 Mbytes of RAM (32 Mbytes recommended)  
CD-ROM drive

### For HP 9000 workstation (700 series)

HP-UX™ 9.05  
16 Mbytes of RAM (32 Mbytes recommended)  
CD-ROM drive

## Product Ordering Information

Product Code	Description
CY3120	<i>Warp2</i> development system for PCs
CY3125	<i>Warp2</i> development system for workstations

### *Warp2* includes:

CD-ROM with *Warp2* and on-line documentation  
*VHDL for Programmable Logic* Textbook  
Registration Card  
Release Notes

Document #: 38-00218-D

*Warp2*, UltraGen, MAX340, pASIC380 and FLASH370 are a trademarks of Cypress Semiconductor Corporation.  
HP-UX is a trademark of Hewlett Packard Corporation.  
SunOS and Solaris are trademarks of Sun Microsystems Corporation.