CY3120/CY3125/CY3120J

Warp2® VHDL Compiler for CPLDs

Features

- **VHDL (IEEE 1076 and 1164) high-level language compiler**
  - **— Facilitates device-independent design**
  - **— 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**
  - **— Supports functions and libraries facilitating modular design methodology**
- ***Warp2*® provides synthesis of IEEE Standard 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**
  - **— Aldec Active-HDL™ FSM graphical Finite State Machine editor (PC only)**
  - **— 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 and Fitting 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**
  - **— Ultra37000™ CPLDs (now with FBGA support)**
  - **— FLASH370i™ CPLDs**
  - **— MAX340™ CPLDs**
  - **— Industry standard PLDs (16V8, 20V8, 22V10)**
- **VHDL and Verilog timing model output for use with third-party simulators**
- **Timing simulation provided with Active-HDL™ Sim Release 3.3 from Aldec (PC only)**
  - **— Graphical waveform simulator**
  - **— Entry and modification of on-screen waveforms**
  - **— 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**
- **Year 2000 Compliant**
- **PC Support (Windows 95™, Windows 98™ and Windows NT™ 4.0)**
- **Workstation Support including Sun Solaris™ and HP-UX™**
- **On-line documentation and help**

Functional Description

Warp2 is a state-of-the-art HDL compiler for designing with Cypress’s Complex Programmable Logic Devices (CPLDs). *Warp2* utilizes a subset of IEEE 1076 and 1164 VHDL as its Hardware Description Language (HDL) for design entry. *Warp2* accepts VHDL, synthesizes and optimizes the entered design, and outputs a JEDEC file for the desired PLD or CPLD (see *Figure 1*). Furthermore, *Warp2* accepts VHDL produced by the Active-HDL FSM graphical Finite State Machine editor (PC only). For simulation, *Warp2* provides a timing simulator (PC only), as well as VHDL and Verilog timing models for use with third party simulators.

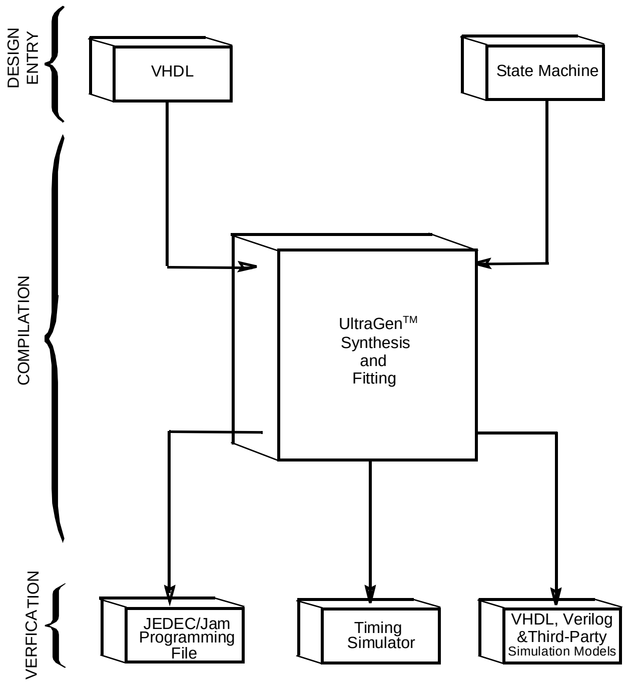

DESIGN ENTRY
VHDL
State Machine
COMPILATION
UltraGen™ Synthesis and Fitting
VERFICATION
JEDEC/Jam Programming File
Timing Simulator
VHDL, Verilog &Third-Party Simulation Models

Figure 1. *Warp2* VHDL Design Flow.

VHDL Compiler

VHDL (VHSIC [very high speed integrated circuit] Hardware Description Language) is a powerful language that is a standard for behavioral design entry and simulation, and is supported by every major vendor of EDA 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 up the design process.

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 easily migrate to high volume technologies. The wide availability of VHDL tools provides complete vendor independence as well. Designers can begin their project using *Warp2* for Cypress CPLDs 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 easy to describe a complex hardware system. In addition, models created in VHDL can readily be transported to other EDA Environments. *Warp2* supports IEEE 1076 and 1164 VHDL including loops, for º generate statements, full hierarchical designs with packages, as well as synthesis for enumerated types and integers.

Designing with *Warp2*

Design Entry

Warp2 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 *Warp2* 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 *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 <= 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. *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:

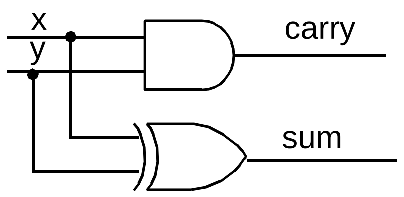


Figure 2. One-Bit Half Adder.

```
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 (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;
```

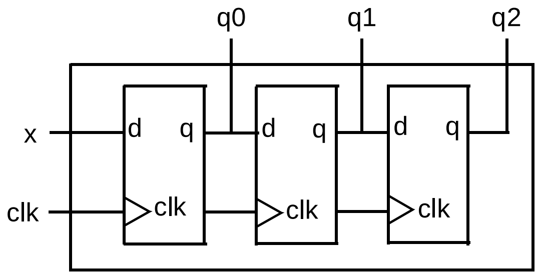


Figure 3. Three-Bit Shift Register Circuit Design.

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 *Warp2* to describe designs using whatever method is appropriate for their particular design.

Finite State Machine Editor (PC only)

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.

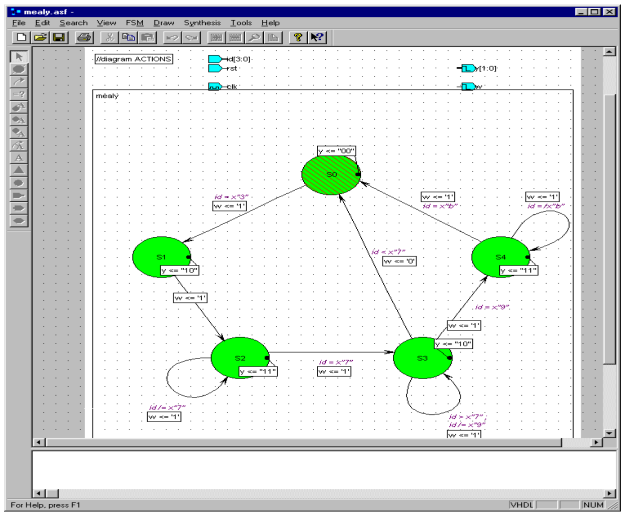


Figure 4. Active HDL FSM Finite State Machine Entry.

Compilation

Once the VHDL description of the design is complete, it is compiled using *Warp2*. 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 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, multipliers, 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 iterative 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 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, *Warp2* creates a JEDEC file for the specified PLD or CPLD.

To facilitate design conversion, a JEDEC translator is provided to convert FLASH370i JEDEC files to target Ultra37000 devices.

Simulation (PC only)

Warp2 includes a post-synthesis timing simulator called Active-HDL Sim. Active-HDL Sim features a graphical waveform simulator that can be used to simulate PLD/CPLD designs generated in *Warp2*. The simulator provides timing simulation for PLDs/CPLDs and features interactive waveform editing and viewing. The simulator also provides the ability to probe internal nodes, and automatically generate clocks and pulses. (Source 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 file that implements the input design in the targeted device. Using this file, Cypress devices can be programmed on any qualified third-party programmer.

Cypress's FLASH370i and Ultra37000 In-System Reprogrammable™ (ISR™) devices can also be programmed on board with an ISR programmer. The JEDEC files are used to program FLASH370i CPLDs or are converted to Jam files by Cypress's ISR software for use with Ultra37000 CPLDs. Once in Jam format, Ultra37000 CPLDs may be programmed using the Jam player with Cypress's ISR software and ISR cable. For more information on Cypress's ISR software see the Ultra37000 ISR Programming Kit (CY3700i) or the FLASH370i ISR Programming Kit (CY3600i) data sheet.

System Requirements

For PCs

- IBM PC or equivalent (Pentium® class recommended)
- 32 Mbytes of RAM (64 Mbytes recommended)
- 110 Mbytes Disk Space
- CD-ROM drive
- Windows (including Japanese) 95, Windows 98, or Windows NT 4.0

For Sun Workstations

- SPARC™ CPU
- Solaris™ 2.6
- 32 Mbytes of RAM (64 Mbytes recommended)
- CD-ROM drive

For HP 9000™ Workstations

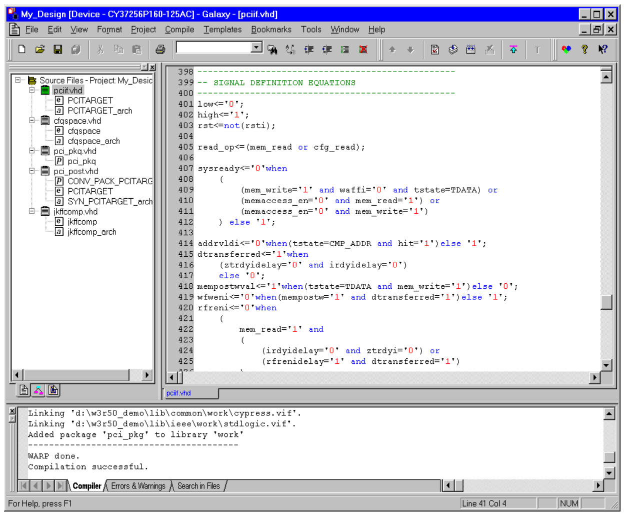
- HP-UX™ 10.20
- 32 Mbytes of RAM (64 Mbytes recommended)
- CD-ROM drive

Product Ordering Information

Product Code	Description
CY3120R52	*Warp2* VHDL development system for PCs
CY3125R52	*Warp2* VHDL development system for Sun Solaris Workstations
CY3110JR52	*Warp2* VHDL Japanese development system for PCs

Warp2 VHDL includes:

- CD-ROM with *Warp2*, Aldec Active-HDL Sim & FSM, and on-line documentation (Getting Started Manual, User's Guide, HDL Reference Manual)
- *VHDL for Programmable Logic* Textbook
- Registration Card
- Release Notes
- Document #: 38-00218-K



```
------------------------------------------------
-- SIGNAL DEFINITION EQUATIONS
------------------------------------------------
low<='0';
high<='1';
rst<=not(rsti);

read_op<=(mem_read or cfg_read);

sysready<='0'when
    (
        (mem_write='1' and waffi='0' and tstate=TDATA) or
        (memaccess_en='0' and mem_read='1') or
        (memaccess_en='0' and mem_write='1')
    ) else '1';

addrvldi<='0'when(tstate=CMP_ADDR and hit='1')else '1';
dtransferred<='1'when
    (ztrdyidelay='0' and irdyidelay='0')
    else '0';
mempostwval<='1'when(tstate=TDATA and mem_write='1')else '0';
wfweni<='0'when(mempostw='1' and dtransferred='1')else '1';
rfreni<='0'when
    (
        mem_read='1' and
        (
            (irdyidelay='0' and ztrdyi='0') or
            (rfrenidelay='1' and dtransferred='1')
```

```
Linking 'd:\w3r50_demo\lib\common\work\cypress.vif'.
Linking 'd:\w3r50_demo\lib\ieee\work\stdlogic.vif'.
Added package 'pci_pkg' to library 'work'
----------------------------------------
WARP done.
Compilation successful.
```

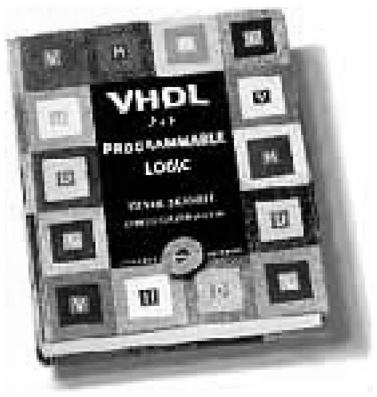


Figure 5. *Warp2* Graphical User Interface and VHDL Book.

Warp2 and *Warp3* are registered trademarks and UltraGen, Ultra37000, MAX340, *Impulse3*, ISR, In-System Reprogrammable, and FLASH370i are trademarks of Cypress Semiconductor Corporation.
Pentium is a registered trademark of Intel Corporation.
Windows 95, Windows 98 and Windows NT are trademarks of Microsoft Corporation.
Solaris is a trademark of Sun Microsystems Corporation.
Active-HDL is a trademark of Aldec Incorporated.