Title: SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR SIMULTANEOUSLY GENERATING NETLISTS WITH MULTIPLE FORMATS

Abstract: A system, method and article of manufacture are provided for compiling a computer program for programming a hardware device. In general, a first net list is created with a first format based on a computer program and a second net list is created with a second format based on the computer program. The first net list and the second net list are created utilizing a single compiler.
For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
SYSTEM, METHOD AND ARTICLE OF MANUFACTURE FOR
SIMULTANEOUSLY GENERATING NETLISTS WITH MULTIPLE
FORMATS

FIELD OF THE INVENTION

The present invention relates to programmable hardware architectures and more particularly to programming field programmable gate arrays (FPGA’s).

BACKGROUND OF THE INVENTION

It is well known that software-controlled machines provide great flexibility in that they can be adapted to many different desired purposes by the use of suitable software. As well as being used in the familiar general purpose computers, software-controlled processors are now used in many products such as cars, telephones and other domestic products, where they are known as embedded systems.

However, for a given function, a software-controlled processor is usually slower than hardware dedicated to that function. A way of overcoming this problem is to use a special software-controlled processor such as a RISC processor which can be made to function more quickly for limited purposes by having its parameters (for instance size, instruction set etc.) tailored to the desired functionality.

Where hardware is used, though, although it increases the speed of operation, it lacks flexibility and, for instance, although it may be suitable for the task for which it was designed it may not be suitable for a modified version of that task which is
desired later. It is now possible to form the hardware on reconfigurable logic circuits, such as Field Programmable Gate Arrays (FPGA's) which are logic circuits which can be repeatedly reconfigured in different ways. Thus they provide the speed advantages of dedicated hardware, with some degree of flexibility for later updating or multiple functionality.

In general, though, it can be seen that designers face a problem in finding the right balance between speed and generality. They can build versatile chips which will be software controlled and thus perform many different functions relatively slowly, or they can devise application-specific chips that do only a limited set of tasks but do them much more quickly.
SUMMARY OF THE INVENTION

A system, method and article of manufacture are provided for compiling a computer program for programming a hardware device. In general, a first net list is created with a first format based on a computer program and a second net list is created with a second format based on the computer program. The first net list and the second net list are created utilizing a single compiler.

In an aspect of the present invention, the first format may include EDIF. As another aspect, the second format may include VDHL. In a further aspect, the computer program from which the first net list was created may be the same as the computer program from which the second net list was created. In another aspect, the computer program may be written in Handel-C.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be better understood when consideration is given to the following detailed description thereof. Such description makes reference to the annexed drawings wherein:

Figure 1 is a schematic diagram of a hardware implementation of one embodiment of the present invention;

Figure 2 illustrates a design flow overview, in accordance with one embodiment of the present invention;
Figure 3 illustrates a method for compiling a computer program for programming a hardware device;

Figures 4A and 4B illustrate a table showing various differences between Handel-C and the conventional C programming language, in accordance with one embodiment of the present invention;

Figure 5 illustrates an interface between Handel-C and VHDL for simulation, in accordance with one embodiment of the present invention; and

Figure 6 illustrates the use of various VHDL files, in accordance with one embodiment of the present invention.
DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment of a system in accordance with the present invention is preferably practiced in the context of a personal computer such as an IBM compatible personal computer, Apple Macintosh computer or UNIX based workstation. A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 110, such as a microprocessor, and a number of other units interconnected via a system bus 112.

The workstation shown in Figure 1 includes a Random Access Memory (RAM) 114, Read Only Memory (ROM) 116, an I/O adapter 118 for connecting peripheral devices such as disk storage units 120 to the bus 112, a user interface adapter 122 for connecting a keyboard 124, a mouse 126, a speaker 128, a microphone 132, and/or other user interface devices such as a touch screen (not shown) to the bus 112, communication adapter 134 for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter 136 for connecting the bus 112 to a display device 138.

The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

In one embodiment, the hardware environment of Figure 1 may include, at least in part, a field programmable gate array (FPGA) device. For example, the central processing unit 110 may be replaced or supplemented with an FPGA. Use of such device provides flexibility in functionality, while maintaining high processing speeds.
Examples of such FPGA devices include the XC2000™ and XC3000™ families of FPGA devices introduced by Xilinx, Inc. of San Jose, Calif. The architectures of these devices are exemplified in U.S. Pat. Nos. 4,642,487; 4,706,216; 4,713,557; and 4,758,985; each of which is originally assigned to Xilinx, Inc. and which are herein incorporated by reference for all purposes. It should be noted, however, that FPGA’s of any type may be employed in the context of the present invention.

A preferred embodiment is written using Handel-C. Handel-C is a programming language marketed by Celoxica Limited. Handel-C is a programming language that enables a software or hardware engineer to target directly FPGAs (Field Programmable Gate Arrays) in a similar fashion to classical microprocessor cross-compiler development tools, without recourse to a Hardware Description Language. This allows the designer to directly realize the raw real-time computing capability of the FPGA.

Handel-C allows one to use a high-level language to program FPGAs. It makes it as easy to implement complex algorithms by using a software-based language rather than a hardware architecture-based language. One can use all the power of reconfigurable computing in FPGAs without needing to know the details of the FPGAs themselves. A program may be written in Handel-C to generate all required state machines, while one can specify storage requirements down to the bit level. A clock and clock speed may be assigned for working with the simple but explicit model of one clock cycle per assignment. A Handel-C macro library may be used for bit manipulation and arithmetic operations. The program may be compiled and then simulated and debugged on a PC similar to that in Figure 1. This may be done while stepping through single or multiple clock cycles.
When one has designed their chip, the code can be compiled directly to a netlist, ready to be used by manufacturers' place and route tools for a variety of different chips.

As such, one can design hardware quickly because he or she can write high-level code instead of using a hardware description language. Handel-C optimizes code, and uses efficient algorithms to generate the logic hardware from the program. Because of the speed of development and the ease of maintaining well-commented high-level code, it allows one to use reconfigurable computing easily and efficiently.

Handel-C has the tight relationship between code and hardware generation required by hardware engineers, with the advantages of high-level language abstraction. Further features include:

- C-like language allows one to program quickly
- Architecture specifiers allow one to define RAMs, ROMs, buses and interfaces.
- Parallelism allows one to optimize use of the FPGA
- Close correspondence between the program and the hardware
- Easy to understand timing model
- Full simulation of owner hardware on the PC
- Display the contents of registers every clock cycle during debug
- Rapid prototyping
- Convert existing C programs to hardware
- Works with manufacturers' existing tools
- Rapid reconfiguration
- Logic estimation tool highlights code inefficiencies in colored Web pages
- Device-independent programs
- Generates EDIFand XNF formats (and XBLOX macros)
Handel-C is thus designed to enable the compilation of programs into synchronous hardware; it is aimed at compiling high level algorithms directly into gate level hardware. The Handel-C syntax is based on that of conventional C so programmers familiar with conventional C will recognize almost all the constructs in the Handel-C language. Sequential programs can be written in Handel-C just as in conventional C but to gain the most benefit in performance from the target hardware its inherent parallelism must be exploited. Handel-C includes parallel constructs that provide the means for the programmer to exploit this benefit in his applications. The compiler compiles and optimizes Handel-C source code into a file suitable for simulation or a net list which can be placed and routed on a real FPGA.


Conventions

A number of conventions are used throughout this document. These conventions are detailed below. Hexadecimal numbers appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that one must type are given in typewriter font as follows:

“void main();”
Information about a type of object one must specify is given in italics as follows:

"copy SourceFileName DestinationFileName"

5 Menu items appear in narrow bold text as follows:

"insert Project into Workspace"

Elements within a menu are separated from the menu name by a > so

10 Edit>Find means the Find item in the Edit menu.

Introduction

Handel-C is a programming language designed to enable the compilation of programs into synchronous hardware. The Handel-C compiler and simulator will now be described. The Handel-C language will be described hereinafter in greater detail.

20 Overview

Design flow overview

Figure 2 illustrates a design flow overview 200, in accordance with one embodiment of the present invention. The dotted lines 202 show the extra steps 204 required if one wishes to integrate Handel-C with VHDL.

30 COMPILER
Figure 3 illustrates a method **3050** for compiling a computer program for programming a hardware device. In general, in operation **3052**, a first net list is created with a first format based on a computer program. Further, in operation **3054**, a second net list is created with a second format based on the computer program. The first net list and the second net list are created utilizing a single compiler. Note operation **3056**.

In an aspect of the present invention, the first format may include EDIF. As another aspect, the second format may include VDHL. In a further aspect, the computer program from which the first net list was created may be the same as the computer program form which the second net list was created. In another aspect, the computer program may be written in Handel-C.

**HARDWARE EMBODIMENTS**

If one is approaching Handel-C from a hardware background, one should be aware of these points:

- Handel-C is halfway between RTL and a behavioral HDL. It is a high-level language that requires one to think in algorithms rather than circuits.
- Handel-C uses a zero-delay model and a synchronous design style.
- Handel-C is implicitly sequential. Parallel processes must be specified.
- All code in Handel-C (apart from the simulator **chanin** and **chanout** commands) can be synthesized, so one must ensure that he or she disables debug code when he or she compiles to target real hardware.
- Signals in Handel-C are different from signals in VHDL; they are assigned to immediately, and only hold their value for one clock cycle.
- Handel-C has abstract high-level concepts such as pointers.
Points of difference

Figures 4A and 4B illustrate a table showing various differences 4000 between Handel-C and the conventional C programming language, in accordance with one embodiment of the present invention.

TARGETING HARDWARE

Targeting hardware via VHDL

If one is integrating Handel-C code with raw VHDL code, one would compile the Handel-C for debug, and use a simulator such as ModelSim to compile the VHDL for simulation. One could then compile the Handel-C to VHDL and use a VHDL synthesis tool such as Simplify LeonardoSpectrum or FPGA Express to synthesize the code. One would then use Xilinx or Altera tools to place and route it.

Linking to the Handel-C VHDL library

Celoxica supplies the HandelC.vhdl file which supports all Handel-C VHDL files. To use Handel-C VHDL, one must compile the HandelC.vhdl file into a library called HandelC. (Consult the documentation for the synthesis or simulation tool on compiling library files.) A person also needs to compile the supplied file ROC.vhdl into the work library for simulation.

Connecting Handel-C EDIF to VHDL

If one compiles a Handel-C file to EDIF and wish to connect it to a VHDL, he or she must be aware that the ports in EDIF and VHDL are different. EDIF ports consist of a collection of single wires. VHDL ports are normally described as n-bit wide
cables. To ensure that the generated EDIF can connect to the VHDL, the VHDL ports must be listed as single-bit wires.

**VHDL component within Handel-C project**

5

**Handel-C code**

```
set clock = external "D17";
unsigned 4 x;
10    interface vhdl_component(unsigned 4 return_val)
    vhdl_component_instance(unsigned 1 clk = __clock,
                            unsigned 4 sent_value = x);
    etc...
    unsigned 4 y;
    y = vhdl_component_instance; // Read from VHDL component
15    x = y; // Write to VHDL component
```

**VHDL code**

20 The VHDL entity may need an interface like this to be compatible with the Handel-C.

```
entity vhdl_component is
    port (  
    clk : in std_logic;
    sent_value_0 : in std_logic;
    sent_value_1 : in std_logic;
    sent_value_2 : in std_logic;
    sent_value_3 : in std_logic;
    return_val_0 : out std_logic;
    return_val_1 : out std_logic;
    return_val_2 : out std_logic;
    return_val_3 : out std_logic
    );
25    end;
```
Note that all the ports are 1-bit wide, `standard_logic` types. This is because when the Handel-C is compiled to EDIF, this is how the expanded interface appears. (EDIF cannot represent n-bit wide cables, only single wires).

5 **Handel-C component within VHDL project**

The Handel-C needs to have ports to its top level, so that the VHDL can connect to them.

```vhdl
10   unsigned 4 x;
    interface port_in(unsigned 1 clk) ClockPort();
    interface port_in(unsigned 4 sent_value) InPort();
    interface port_out() OutPort(unsigned 4 return_value = x);
15   set clock = internal ClockPort.clk;
    etc...
    unsigned 4 y;
    y = InPort.sent_value;  // Read from top-level VHDL
    x = y;  // Write to top-level VHDL
20   VHDL code
The top level VHDL may need to instantiate the Handel-C like this:
    component handelc_component
    port (  
    clk : out std_logic;
    sent_value_0 : out std_logic;
    sent_value_1 : out std_logic;
    sent_value_2 : out std_logic;
    sent_value_3 : out std_logic;
    30   return_val_0 : in std_logic;
    return_val_1 : in std_logic;
    return_val_2 : in std_logic;
    return_val_3 : in std_logic  
    );
35   end component;.
```
Targeting hardware via EDIF

To target hardware via EDIF, one may set up the project to target EDIF using the Build>Set Active Configuration command. This compiles directly to an .edf file which can be passed to the place and route tools.

Port renaming for debug

To aid in debugging the generated EDIF, one can rename the EDIF nets within the net list such that the Handel-C declaration name appears before the EDIF unique identifier.

CONNECTING TO VHDL BLOCKS

Requirements

If one wishes to connect Handel-C code to VHDL blocks and simulate the results, one may require the following objects:

- A VHDL simulator (such as ModelSim)
- The cosimulator plugin (e.g. PlugInModelSim.dll) to allow the VHDL simulator to work in parallel with the Handel-C simulator. This file is provided with the copy of Handel-C
- The file plugin.vhdl to connect the VHDL to the cosimulator plugin. This file is included with the copy of Handel-C
- A VHDL wrapper file to connect the VHDL entity ports to the Handel-C simulator and to VHDL dummy signals. (One must write this)
- The VHDL entity and architecture files (one must provide or write these)
- A Handel-C code file that includes an interface definition in the Handel-C code to connect it to the VHDL code. (One must write this.)
Simulation requirements

Before one can simulate the code he or she must:

5 1. Set up ModelSim so that the work library refers to the library containing this wrapper component.

2. Check that the plugin has been installed in the same place as the other Handel-C components. If one has moved it, he or she must ensure that its new location is on the PATH.

3. Compile the VHDL model to be integrated with Handel-C into the VHDL simulator.


5. Compile the wrapper.

15 6. Compile the Handel-C code and run the Handel-C simulator. This may invoke any VHDL simulations required.

Batch files

20 Sample batch files that carry out these tasks have been supplied with the examples:

Figure 5 illustrates an interface 5000 between Handel-C and VHDL for simulation, in accordance with one embodiment of the present invention.

Place and route requirements

If one wishes to compile the Handel-C code and VHDL blocks and place and route the results, he or she may need to:

30

• Compile the Handel-C code to VHDL.
Pass the compiled Handel-C and the VHDL model to an RTL synthesis tool (such as FPGAEExpress).

- Run the place and route.

5 Writing Handel-C to communicate with VHDL

The code needed in the Handel-C program is in two parts. First, one needs an interface declaration. This prototypes the interface sort and is of the format:

10 interface

    VHDL_entity_sort (VHDL_to_HC_port
    {,VHDL_to_HC_port })
    (VHDL_from_HC_port
    {, VHDL_from_HC_port});

where:

VHDL_entity_sort is the name of the VHDL entity. This name must be used as the interface sort.

VHDL_to_HC_port is the type and name of a port bringing data to the Handel-C code (output from VHDL) precisely as specified in the unwrapped VHDL entity

VHDL_from_HC_port is the type and name of a port sending data from the Handel-C code (input to VHDL) precisely as specified in the unwrapped VHDL entity.

Note that ports are seen from the VHDL side, so port names may be confusing. In Handel-C, the ports that input data TO the Handel-C must be specified first.

30
One then needs an interface definition. This creates an instance of that interface sort, and defines the data that may be transmitted. This is of the format:

```vhdl
interface

5

VHDL_entity_sort (VHDL_to_HC_port [with portSpec]
{" VHDL_to_HC_port [with portSpec]})
interface_Name (VHDL_from_HC_data = from_HC_data
[with portSpec]
10
{" VHDL_from_HC_data = from_HC_data
[with portSpec]])
with {extlib="PluginModelSim.dll",
extinst="instanceName; model=entity_wrapper;
clock=clockName:period; delay=units"};
```

where:

- `VHDL_entity_sort` is the interface sort that one previously declared.
- `VHDL_to_HC_port` is the type and name of a port bringing data to the
  Handel-C code (output from VHDL). This may have the same type as
  defined in the interface declaration
- `interface_Name` is the name for this instance of the interface.
- `VHDL_from_HC_port` is the type and name of a port sending data from
  the Handel-C code (input to VHDL). This may have the same type as
  defined in the interface declaration
- `VHDL_from_HC_data` is an expression that is output from the Handel-C
to the VHDL.

**Interfacing the VHDL with the Handel-C simulator**

30
Figure 6 illustrates the use of various VHDL files 6000, in accordance with one embodiment of the present invention. One needs to provide a wrapper file 6002 for VHDL code 6004. The wrapper file wraps the VHDL code, connecting the entity ports to dummy signals and provides the interface to the Handel-C simulator plugin 6006. The wrapper code is only required in the simulation phase, not in the synthesis phase. The following information assumes that one has two VHDL files, the object code for the architecture file (entity_architecture.vhdl) and the source code for the interface to the behavior file (entity.vhdl).

One needs to examine the ports defined in the entity file, and ensure that each port is connected to a signal in a wrapper file.

Note that a limited number of port types are supported:

- 1-bit types in Handel-C must be implemented by std_logic
- n-bit unsigned and signed types in Handel-C must be implemented by std_logic_arith.unSigned

No other types may be used. If the circuit uses other types one may need to create another VHDL wrapper containing type conversions to these three types between the plugin wrapper and the circuit to be integrated.

A simple combinatorial circuit example

The VHDL code

The VHDL code for the combinatorial circuit is in the file ttl7446.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity TTL7446 is
port (ltn : in std_logic;
```
rbin : in std_logic;
digit : in unsigned(3 downto 0);
bin : in std_logic;
segments : out unsigned(0 to 6);
rbon : out std_logic);
end;
architecture behavior of TTL7446 is
begin
process(ltn, rbin, bin, digit)
begin
rbon <= '1';
if bin = '0' then
segments <= "1111111";
elsif ltn = '0' then
segments <= "1000000";
else
case digit is
when "0000" => segments <= "0000001";
if rbin = '0' then
segments <= "1111111";
rbon <= '0';
end if;
when "0001" => segments <= "1001111";
when "0010" => segments <= "0010010";
when "0011" => segments <= "0000110";
when "0100" => segments <= "1001100";
when "0101" => segments <= "0100100";
when "0110" => segments <= "1100000";
when "0111" => segments <= "0001111";
when "1000" => segments <= "0000000";
when "1001" => segments <= "0001100";
when "1010" => segments <= "1100100";
when "1011" => segments <= "1101100";
when "1100" => segments <= "1011100";
when "1101" => segments <= "0110100";
when "1110" => segments <= "1110000";
when "1111" => segments <= "1111111";
when others => segments <= "XXXXXXXX";
end case;
end if;

A sample wrapper for the combinatorial circuit

entity TTL7446_wrapper is
end;
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
architecture HandelC of TTL7446_wrapper is
signal ltn : std_logic;
signal rbin : std_logic;
signal digit : unsigned(3 downto 0);
signal bin : std_logic;
signal segments : unsigned(0 to 6);
signal rbon : std_logic;
begin
plugin1: entity work.plugin;
ttl: entity work.TTL7446 port map (ltn, rbin, digit, bin, segments, rbon);
end

This shows the two instances. It also shows each port of the circuit to be integrated connected to a signal which is not connected to anything else. This is not a requirement of the plugin, but a requirement of VHDL. Note that VHDL'93 features have been used to create direct instantiations of the components.

Example Handel-C using the combinatorial circuit

    // Set chip details
    set clock = external "D17";
    set part = "V1000BG560-4";

    // Interface declaration
interface TTL7446(unsigned 7 segments, unsigned 1 rbon)
(unsigned 1 ltn, unsigned 1 rbin, unsigned 4 digit,
unsigned 1 bin);
// Main program

void main(void)
{
    unsigned 1 ltnVal;
    unsigned 1 rbinVal;
    unsigned 1 binVal;
    unsigned 4 digitVal; // Connecting to VHDL blocks
    unsigned 1 rbonVal;
    unsigned 20 Delay;
    interface TTL7446(unsigned 7 segments, unsigned 1 rbon)
    decode(unsigned 1 ltn=ltnVal, unsigned 1 rbin=rbinVal,
    unsigned 4 digit=digitVal, unsigned 1 bin=binVal)
    with {extlib="PluginModelSim.dll",
    extinst="decode; model=TTL7446_wrapper; delay=1"};
    interface bus_out() display(unsigned display =
    ~decode.segments)
    with {extlib="?segment.dll", extinst="0",
    data={"AN28", "AK25", "AL26", "AJ24", "AM27", "AM26",
    "AK24"})};
    par
    {
        ltnVal = 0;
        rbinVal = 0;
        binVal = 0;
        digitVal = 0;
    }

    while(1)
    {
        binVal = 1;
        ltnVal = 1;
        do
        {
            do
            {
                rbonVal = decode.rbon;

    21
digitVal++;  
#ifdef SIMULATE  
do { Delay++; } while(Delay!=0);  
#endif  
} while (digitVal != 0);  
rbinVal++;  
} while (rbinVal != 0);  
}

10 Interface code

One must declare an interface sort that has port names of the same name and type as the VHDL signals in the circuit to be integrated. The interface sort must be the same as the VHDL model's name.

interface TTL7446(unsigned 7 segments, unsigned 1 rbon) (unsigned 1 ltn, unsigned 1 rbin, unsigned 4 digit, unsigned 1 bin);

An instance of this component is then created one or more times in the

Handel-C code. An example of an instantiation is:

interface TTL7446(unsigned 7 segments, unsigned 1 rbon)  
decode(unsigned 1 ltn=ltnVal, unsigned 1 rbin=rbinVal, unsigned 4 digit=digitVal, unsigned 1 bin=binVal) with  
{extlib="PluginModelSim.dll",  
extinst="decode;  
model=ttl7446_wrapper; delay=1"};.Connecting to VHDL blocks
CLAIMS

What is claimed is:

1. A method for compiling a computer program for programming a hardware
   device, comprising the steps of:
   (a) creating a first net list with a first format based on a computer program; and
   (b) creating a second net list with a second format based on the computer
        program;
   (c) wherein the first net list and the second net list are created utilizing a single
       compiler.

2. A method as recited in claim 1, wherein the first format includes EDIF.

3. A method as recited in claim 1, wherein the second format includes VDHL.

4. A method as recited in claim 1, wherein the computer program from which
   the first net list was created is the same as the computer program from which the
   second net list was created.

5. A method as recited in claim 1, wherein the computer program is written in
   Handel-C.

6. A computer program product for compiling a computer program for
   programming a hardware device, comprising:
   (a) computer code for creating a first net list with a first format based on a
       computer program;
   (b) computer code for creating a second net list with a second format based on
       the computer program;
   (c) wherein the first net list and the second net list are created utilizing a single
       compiler.
7. A computer program product as recited in claim 6, wherein the first format includes EDIF.

8. A computer program product as recited in claim 6, wherein the second format includes VDHL.

9. A computer program product as recited in claim 6, wherein the computer program from which the first net list was created is the same as the computer program from which the second net list was created.

10. A computer program product as recited in claim 6, wherein the computer program is written in Handel-C.

11. A system for compiling a computer program for programming a hardware device, comprising:

   (a) logic for creating a first net list with a first format based on a computer program; and

   (b) logic for creating a second net list with a second format based on the computer program;

   (c) wherein the first net list and the second net list are created utilizing a single compiler.

12. A system as recited in claim 11, wherein the first format includes EDIF.

13. A system as recited in claim 11, wherein the second format includes VDHL.

14. A system as recited in claim 11, wherein the computer program from which the first net list was created is the same as the computer program from which the second net list was created.
15. A system as recited in claim 11, wherein the computer program is written in Handel-C.
Fig. 1.
Fig. 2.

1. Port algorithm to Handel-C

2. Compile program for debug

3. Use simulator to evaluate and debug

4. Add interfaces to external hardware

5. Compile Handel-C to target hardware netlist

6. Use FPGA tools to place and route netlist

7. Program FPGA with result of place and route

8. Handel-C simulator invokes VHDL simulator if required

9. VHDL

10. Synthesise compiled Handel-C and VHDL

200 → 202 → 204
Fig. 3.

3050 Creating a first net list with a first format based on a computer program

3052

Creating a second net list with a second format based on the computer program

3054

Wherein the first net list and the second net list are created utilizing a single compiler

3056

---

Fig. 4B.

4000

<table>
<thead>
<tr>
<th>Constrained functions</th>
<th>You cannot have empty loops in Handel-C.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Functions may not be recursive.</td>
</tr>
<tr>
<td></td>
<td>Variable length parameter lists are not supported.</td>
</tr>
<tr>
<td></td>
<td>Old-style function declarations are not supported.</td>
</tr>
<tr>
<td>Strong typing</td>
<td>Handel-C has variables which can be defined to be of any width.</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Casting can't change width.</td>
</tr>
<tr>
<td></td>
<td>There are no automatic conversions between signed and unsigned values. Instead, values must be 'cast' between types to ensure that the programmer is aware that a conversion is occurring that may alter the meaning of a value.</td>
</tr>
<tr>
<td></td>
<td>Pointers can only be cast to void and back, between signed and unsigned and between similar structs. You cannot cast pointers to any other type.</td>
</tr>
<tr>
<td>True parallelism</td>
<td>You can have multiple main functions in a project. Each Handel-C main function must be associated with a clock.</td>
</tr>
<tr>
<td></td>
<td>Although implicit sequential, Handel-C has parallel constructs which allow you to speed up your code.</td>
</tr>
<tr>
<td>Width of variables</td>
<td>Handel-C has variables which can be defined to be of any width.</td>
</tr>
<tr>
<td></td>
<td>In ISO-C, bit fields are made up of words, and only the specified bits are accessed, the rest are padded. Since there are no words in Handel-C, no form of packing can be assumed.</td>
</tr>
<tr>
<td></td>
<td>If you have an array[4] and you use its index as a counter, the index width will be assumed by the Handel-C compiler to be two bits wide (to hold the values 0-3). It will not be able to hold the value 4.</td>
</tr>
<tr>
<td>No side-effects allowed</td>
<td>Instead of writing complex single statements, it is more efficient in Handel-C to write multiple single statements and run them in parallel.</td>
</tr>
<tr>
<td></td>
<td>You cannot perform two assignments in one statement.</td>
</tr>
<tr>
<td></td>
<td>Auto variables cannot be initialised, as that means that hidden clock cycles are required, instead, they must be explicitly assigned to in a separate statement.</td>
</tr>
</tbody>
</table>