A Real-time debugger with Bitstream Configurator & ‘C’ Language Design Control for FPGAs

Steve Casselman, John Schewel, Frank Wartel
sc@vcc.com, jas@vcc.com, fw@vcc.com
Virtual Computer Corporation
6925 Canby Ave #103
Reseda, California, USA 91335

ABSTRACT
At the boundary between hardware and software, where FPGAs with 2,000,000 gates is just the beginning, we’ve found that the tools you have in your toolbox make all the difference. With larger and more feature rich programmable devices such as the Virtex? Platform FPGA [1,2,3], even minor changes in the design can require hours of compile time. The combination of design complexity and component size is taxing current design entry and implementation tools, making the design cycle longer. The simulation-verification cycle doesn’t mean the design will work in the final product. The engineer needs more than ever, to debug designs within the target hardware in real-time. We have built a series of integrated tools aimed at enhancing productivity at the last stages of product design, the final ten percent of the design that takes ninety percent of the time. The tools shown are not meant to replace current tools. These are advanced tools for the FPGA power user. Our goal is creating a set of tools specifically designed to provide better design control, dramatically reduce iteration times and enable real-time In-Circuit debugged in hardware. We have organized this series of tools as The Technology Stack?.

Keywords: Bitstream, Emulator, Reconfigurable Computers, FPGA, Hardware Software Co-Design, H.O.T., C to Hardware

1. INTRODUCTION
The Technology Stack (Figure 1) is a highly integrated family of productivity tools designed to dramatically reduce iteration times through direct bitstream manipulation, add behavioral control to your design using ‘C’ language code and provide an easy to use, real-time, in-circuit debugging environment that plugs into your hardware. At the top of The Technology Stack, H.O.T. ICE? The In-Circuit Enabler provides an easy to use environment for the editing, debugging and validating your design. Unlike other emulators, H.O.T. ICE plugs directly into the target hardware using the SelectMap port of the Virtex. H.O.T. ICE uses other members of The Technology Stack to add functionality. The H.O.T. Browser? is the GUI window to the design; this Design Editor provides a familiar, easy to use environment for inspecting, organizing, editing and running the design changes through the Enabler. The H.O.T. Browser features include chip to logic views, editing windows for changing logic, formulas, and RAM or BlockRam contents of the Virtex. The H.O.T. State Machine Compiler? is a ‘C’ to Hardware compiler with direct BitStream generation capabilities. It enables you to use the BlockRam for logic control. BitMan? is a BitStream Configuration generator.

The Technology Stack is a continuing application of VCC’s Hardware Object Technology. VCC’s Hardware Object Technology enables the designer to use digital designs with standard ‘C’ language programs [5]. Your digital design is downloaded from within an application program (as a ‘C’ language function). VCC’s unique implementation of reconfigurability and ease-of-use allows Real-Time debugging of digital designs. We will start at the base of The Technology Stack, with direct bitstream manipulation for better control and reduced iteration times.
2. DIRECT BITSTREAM MANIPULATION

Direct manipulation of the BitStream can deliver faster iteration times for a number of changes to your design. Our Bitstream Manipulator is called BitMan, a command line editor for manipulating the Virtex configuration bitstream.

Features:
1. Ability to change contents of LUT and BlockRam
2. BitMan by-passes standard Map, Place and Route tools
3. Outputs complete configuration bitstream or partial configuration bitstream
4. Command line entry and scripting capabilities

The Virtex configuration space can be represented as an array of bits (fig 2.1), grouped into one bit-wide vertical frames. A frame is the smallest unit of the reconfiguration data that can be read or write. To read or write a bit it is necessary to transfer a whole frame. Frames themselves are grouped into different type of bigger entities called columns. There are five different types of columns whose sizes and numbers depend on the size of the chip itself: a center column for the global clocks, CLBs columns, IOBs columns, Block SelectRAM interconnect columns, and finally Block SelectRAM Content columns.

This memory structure is addressed using three variables: the block type, the Major Address which is the address of a column, and a Minor address which is the address of a frame inside a column. For further detailed information regarding the configuration array, please refer to Xilinx Application Note XAPP151 [4].

A good way to approach the structure of the configuration bitstream is to think of it as a packet oriented architecture. A packet is made of a command header followed by the actual data (fig 3). The command header section defines the type of transfer such as the targeted CLBs or RAM Blocks, the location of the frame to modify, the direction, and its length. Then the actual data is either read or write to the Virtex. The data consists of 32 bits wide words.

BitMan does not require a complete understanding of the configuration bitstream structure by the end user, as the tool will seek into the bitstream, modify it if necessary and return the complete or partial bitstream according to the modifications applied.

3. ‘C’ PROGRAMMABLE STATE MACHINE CONTROLLER
The H.O.T. State Machine Compiler allows an engineer to place logic in the BlockRam of the Virtex for the purpose of controlling the behavior of the design with ‘C’ programming language. The H.O.T. State Machine Compiler has three components: 1) State Machine, 2) ‘C’ Compiler & 3) BitMan. The HOT State Machine (HSM) and C compiler (hsmcc) are designed to control the algorithmic behavior of target architecture. The purpose of the controller is to control an external architecture. Its output bits should be attached to clock enables of flip flop or to the control bits of muxes or control inputs of some other IP. The inputs look at conditions such as the terminal count of a counter or the full and empty flags of a FIFO. The machine is synchronous and has only one clock.

Features:

1) 8 programmable outputs
2) 2 programmable inputs
3) Programmed in a subset of C
4) Parallelism expressed by the “,” operator
5) Once design is compiled design iterations take seconds
6) Uses all C preprocessor commands
7) Cycle by Cycle control of state machine behavior using C
8) GCC used to debug code
9) 140 MHz in XCV300E-8
10) 175 MHz in Virtex II XCV1000
11) Small size 20 CLBs and 1 Block Ram.
12) Only 128 micro-code words

C constructs allowed:     If, else, goto, do and while loops, for loops and subroutines. [6,7]
Operators =, ==, !, ||, &&
C constructs not used:  case, break or continue statements, typedef or structures or enumerated types pointers or other memory references, arrays, *; +, -, or / or arithmetic operators of any kind

Variables:

There are three types of variables: output, input and loop. Input and output variables must be declared “bool” which means that they may only be assigned 0 or 1.

The HSM8-Virtex1 has only one loop variable “i”. It has a range of 0 to 127.

Output Variables:
Output variables can have any name except for standard “C” language reserved words. Each output variable is global in scope and must be declared with the keyword “bool.” Any signal can be switched in parallel by using the list notation. The output variables in the hardware environment are called s[7:0], with first variable declared corresponding to s0 and the second to s1 … Before the first clock (and after any global reset) all outputs are zero (0).

Some Examples:

```c
bool var1=1,var2=0,var3=1,var4=0,var5=0; // declare some variables
var1=1,var2=0,var3=1,var4=1,var5=1; /* all switch at same time */
var1=0; // takes one clock
var0=1; // takes one clock
; // NO OP for one clock
```

Input Variables:
Input variables can have any name except for standard “C” language reserved words. Each input variable is global in scope and must be declared with the keyword “bool.” There are 2 input variables. Input signals can be used in control test expressions and are evaluated in a single clock. Control expressions are: if(<expression>), while(<expression>), do{} while(<expression>). The input variables in the hardware environment are call var[1:0] with first variable declared corresponding to var0 and the second to var1.
While and Do while loops:
While loops are of the form
Example: while(<expression == true>) {<Statements>}
Or do {<Statements>} while (expression == true)

The while loop takes one clock at the end of the statements to check the termination conditional. The expression is only tested once every loop. A while loop with the null statement will jump to its own address each clock until the expression is false. This is the same as a “wait until” operation. Example: while(<expression == true>);

Design Flow:

**First Time:**
Instantiate Controller in ‘VHDL’
VHDL CODE
Compile Design with Standard Map, Place & Route Tools
Configuration BitStream

**Saving Time:**
Change ‘C’ control Code
‘C’ CODE
Compile with VCC Compiler
Insert Changes Instantly Into Configuration Bitstream Using VCC’s BitMan™

Figure 4
Total time = 20 minutes => 20 hours

Figure 5
Total time = Less than 1 sec.

The combination of easy of use, ‘C’ Language for behavioral control of the architecture of your design and Direct Bitstream manipulation dramatically improve iteration times for design changes. The Data Path of your design is written in HDL. The control of the architecture is written in ‘C’. Therefore changes can be implemented without effecting timing restraints or routing. The design can be handed over to software engineers for testing.

Example Control Code
// Uses all C pre processor commands
#define allOnes Led0 = 1, Led1 = 1, Led2 = 1, Led3 = 1, \ Led4 = 1, Led5 = 1, Led6 = 1, Led7 = 1
#define allZero Led0 = 0, Led1 = 0, Led2 = 0, Led3 = 0, \ Led4 = 0, Led5 = 0, Led6 = 0, Led7 = 0

// output variables
bool Led0=0; bool Led1=1; bool Led2=0; bool Led3=0; bool Led4=1; bool Led5=1; bool Led6=0; bool Led7=1;

// loop counter
int i;

// input variables
bool Button0; bool button1;
void blink1();
void main () {
while(1) {
allZero;
allOnes;
allZero;
Led0 = 1;
Led0 = 0,Led1 = 1; Led1 = 0,Led2 = 1; Led2 = 0,Led3 = 1; Led3 = 0,Led4 = 1;
Led4 = 0,Led5 = 1; Led5 = 0,Led6= 1; Led6 = 0,Led7= 1;
blink1();
}
}

void blink1(){
for(i=0;i<15;i++) {
if(Button0 == 1 && Button1 == 0) {
allZero;
}
Led0 = 1;
Led0 = 0,Led1 = 1;
while(Button0 == 0); // Jump to same address until true
Led1 = 0,Led2 = 1; Led2 = 0,Led3 = 1; Led3 = 0,Led4 = 1;
Led4 = 0,Led5 = 1; Led5 = 0,Led6= 1; Led6 = 0,Led7= 1;
Led7 = 0;
if(Button1 == 1) {
blink1(); // recursion allowed 16 deep stack
}
return;
}

4. AN FPGA BROWSER & DESIGN EDITOR

The H.O.T. Browser is a GUI Design Editor designed as a familiar, easy to use environment for advance design control using BitMan and the HOT State Machine Compiler. The H.O.T. Browser links chip and signal views to debugging and editing features. The main window has three components (Figure 6): The signal window, the world chip view, and the design view. You open an existing NCD file. All signals used in your design are displayed in table form. The world chip view displays the entire Virtex chip. The primitives used in your design are display by color at their respective Column, Row and Slice number. The World View window controls the Chip view window’s location on the Virtex. The chip view window allows you to zoom in and out of the entire design.

Figure 6
As you zoom in on a given group of primitives, either the name of the primitive or it’s value is displayed. You can zoom in to the fine grain level, which shows the detail of the Row and Column (Figure 7).

The H.O.T. Browser is integrated with BitMan and the H.O.T. State Machine Compiler. This allows the designer to edit values and contents of BlockRam or LUTs at the GUI level, while generating Configuration Bitstreams directly for simulation, verification or downloading.

5. REAL-TIME IN-CIRCUIT DEBUGGING

The H.O.T. ICE In-Circuit Enabler is a real-time In-Circuit Debugging and Validation Tool. The H.O.T. ICE allows the designer to plug their hardware into The Technology Stack. H.O.T. ICE is a workstation with single or multiple H.O.T. ICE communication cards that plug in to the H.O.T. ICE Breakout Box (Figure 8). Using flying leads or ribbon cable, you can connect directly to your hardware via the SelectMap port. The current version supports one target system per breakout box. We plan to extend that to four per break out box.

When using the H.O.T. Browser with H.O.T. ICE you get the added feature of direct downloading and polling of the target hardware. Choose the clock cycles in either your simulation or real hardware implementation you wish to run and read the outputs of critical functions. If you need a constant where a formula is, just change it, download it and run again in seconds.

6. CONCLUSIONS

We have presented a method whereby the behavior of a design can be completely changed without changing design itself. We have shown how to do this “in circuit” into a full configured and working FPGA. The ability to bypass all the place and route tools is a powerful method that can save time and money when integrating a new system. To do almost anything with the normal tool flow takes 5 to 10 minutes for the smallest changes in the smallest part. Changes at the bitstream level can be done in less than 1 second no matter what size part. The H.O.T. ICE In-Circuit Enabler is a productivity tool for advanced users of the Virtex Platform Family of FPGAs.
7. REFERENCES

2. *Virtex 2.5V Field Programmable Gate Arrays Advance Product Specifications* Xilinx, Xilinx Inc. 1999
3. *Virtex 1.8V Field Programmable Gate Arrays Advance Product Specifications* Xilinx, Xilinx Inc. 1999
5. S. Casselman
7. Stephen Kochan.

All trademarks belong to their respective companies.