Handel-C for Rapid Prototyping of VLSI Coprocessors for Real Time Systems

S. M. Loo, B. Earl Wells, N. Freije, J. Kulick
Department of Electrical and Computer Engineering
The University of Alabama in Huntsville
Huntsville, Alabama 35899

Key Words: Real Time Systems, Handel-C, FPGA, Reconfigurable Computing

Abstract - The current maturity of modern reconfigurable hardware elements such as field programable gate arrays now makes it possible to utilize application-specific reconfigurable co processor logic as part of real time system design. Such logic has great potential to improve both the level of performance and run time determinism of the system. It also gives the real time system designer the capability of performing nonintrusive high speed monitoring operations such as missed deadline detection and external bus/IO activity analysis that can be directly utilized by the system scheduler to dynamically adapt to changing process load conditions. In most cases, though, designers of real time systems are software development practitioners, not hardware developers. They know much more about traditional software high level programming languages such as C and C++ than hardware description languages such as VHDL and Verilog. New hardware description languages such as Handel-C are now becoming available to make the hardware design process more accessible to these software developers. In this paper, we investigate the effectiveness of using Handel-C in an academic setting, to develop real time embedded systems in environments that incorporates reconfigurable FPGA based co-processor logic.

I. INTRODUCTION

In real time systems the timing requirements are essential parts of its specification. Reconfigurable logic can aid directly in real-time processing by allowing highly repetitive operations to be off-loaded to a high speed, rapidly prototyped coprocessor whose features were created specifically to handle the particular operations associated with the application. Thus efficient dedicated hardware can now be created at the same time that the system software is created without encountering the high non-recurrent cost associated with IC and PC board fabrication. This also means that in some cases the nondeterministic timing effects associated with conventional embedded processing environments, such as cache memory, DMA access, and unpredictable process interruption due to sporadic external interrupts can be eliminated for certain operations if they are implemented in the reconfigurable hardware.

Another important use of reconfigurable coprocessor logic in embedded real time systems is in the area of adaptive performance modeling of system parameters. Reconfigurable hardware allows the direct monitoring of such parameters as missed deadlines, maximum lateness, effects of external I/O and bus activities. The on board asynchronous event driven scheduler of the real time software environment can then make use of this information to dynamically predict and recover from transient process overload conditions and other anomalies that affect hard and soft real time systems. The reconfigurable co-processors could also be used to dynamically and nonintrusively record the behavior of the real system for later postmortem analysis designed to improve future representations of the system.

One of the main problems with implementing such combined reconfigurable hardware/software real time systems is the fact that most real time system practitioners are software oriented. Most hardware description languages are inherently concurrent in nature whereas most high level software languages are highly sequential. One must change their basic thought pattern when one uses an HDL - - parallelism is the default, sequentiality must be explicitly expressed. This means learning hardware description languages such as VHDL is not trivial and often will not fit within the scope of the project. In addition many of the constructs of these HDLs are not synthesizable. This is because many hardware description languages were developed with the idea of documenting and simulating a design not actually implementing it. This means that many of the most powerful constructs will go unused and actually serve as an impediment to the novice user since these users often inadvertently use these constructs in their designs not knowing that they are not synthesizable.

The idea expressed by many, is to use a hybrid language to describe the hardware portion of the design that is in some way similar to existing software high-level languages. Thus traditional real-time system practitioners could themselves rapidly prototype reconfigurable coprocessors hardware technology without climbing a huge learning curve. One such language is Handel-C [1].

II. Handel-C

A. Overview

Handel-C is in effect a parallel synchronous programming language where the notion of time is fundamental to its specification. In this language all events occur relative to a global clock that runs continuously. Information is encoded behaviorally level in a manner that is common to most high level languages. Handel-C is based on subset of
data sharing, timing, and directly access block RAM in
features are mainly to support parallelism, communication,
flow control, and interprocess communication. These new
variables (variables that are mapped to registers), logical
nous named channel point-to-point communication mecha-
which allows parallel execution of blocks of code. There is
present in Handel C beginning with the
characteristics of the hardware structures that are created
tement designer better predict and understand the execution
confidence that a design that has been entered in Handel-C
are synthesizable into hardware. This greatly increases the
basic constructs (except the
channel and chanout I/O calls) are synthesizeable into hardware. This greatly increases the
confidence that a design that has been entered in Handel-C can actually be created and will actually perform in the
manner that the designers intended. The synchronous
nature of Handel-C also allows designers to closely esti-
mate the number of clock cycles a block of code will take
to execute. This features is also applicable when communic-
ion between parallel block of code is incorporated in the
design. There exists a simple execution model where every
assignment and delay takes a single clock cycle. This
means Handel-C has the potential to help the real-time sys-
tem designer better predict and understand the execution
characteristics of the hardware structures that are created
during synthesis.

Figure 1 lists many of these extra features that are
present in Handel C beginning with the par statement
which allows parallel execution of blocks of code. There is
also a channel statement which supports the CSP syn-
chronous named channel point-to-point communication mecha-
nism. Other features include the use of special data path
variables (variables that are mapped to registers), logical
operators, arithmetic operators, relational operators, delay
construct, assignment, sequential and parallel composition,
flow control, and interprocess communication. These new
features are mainly to support parallelism, communication,
data sharing, timing, and directly access block RAM in
modern FPGAs.

An interesting feature provided by Handel-C is the
capability to specify the number of bits required for each
variable. This allows hardware to be created that does not
waste resources, i.e. it doesn’t make sense to have a 32-bit
adder when the maximum number to add is only 15! In
addition, Handel-C allows for the use of pointers.

It should be noted that Handel-C does not directly sup-
sport the ANSI C defined constructs of double, float, sizeof
and union. (Handel-C does support a special floating point
library which can be used to implement floating point in a
hardware design.) Unlike traditional C, Handel-C is a
strongly typed language, casting is allowed but casting
cannot change the width of an item. Recursion is not sup-
ported and multiple assignments in the same statement is
not allowed.

B. An Example

To illustrate the use of Handel-C, let us consider an
implementation of a four-bit binary counter that is modeled
as a classical producer/consumer relationship. The purpose
of this example is not to illustrate how best to implement
this particular application in Handel-C but rather to present
a simple example that will show some of the major con-
structs and features of the language. This example repre-
sents a simple 4 bit counter that is implemented on a
FPGA-based protoboard which is clocked by a 30 MHz
clock. This counter is to continuously drive four FPGA
pins on the protoboard with a sequential binary count value
which changes approximately every 2.2 seconds. The Han-
del-C code is shown in Figure 2. It consists of initialization
code (lines 2-9 -- declaration of FPGA pin assignments for
clocks and counter outputs and the interprocess commu-
nication channel to be used), delay_time function (lines 11-
18), producer function (lines 20-28), consumer function
(lines 30-43), and the main function (lines 46-56). The pro-
ducer and consumer functions operate in parallel as
directed by the par construct in the main function (line 50).
The producer and consumer functions communication
through the common channel they share which is named
report_count. This is where the 4-bit count value is passed
from one parallel function to the other (line 26, and line
40). Once the value has been accepted by consumer, the
producer increments it’s counter value and then exits back
to the main function (where it is re-invoked). It should be noted, that the counter value, cnt_proc, in the producer is incremented at the same time other operations are being performed by the consumer but in this model the producer function spends most of its time being stuck at line 26 waiting for the consumer to accept the new count value. The consumer function firsts accepts the count value, places it on the 4 FPGA output pins and then executes the delay_time function. The delay_time function is simply a nested loop which executes 67088385 times which consumes that many clock cycles. With a 30 MHz system clock this means that it takes ~2.2 seconds to complete this sequential function call. When this time has passed, the consumer function exists and reenters through the main function where it accepts the next value from the producer (lines 26 and 40) and this process repeats indefinitely. Lines 58 and 59 show the commented versions of the de-

```c
//delay_time function
while (1)
{
    delay_time();                                 //wait ~2.2 seconds
    output = cnt_monitor;                   //output to FPGA pins
}
```

The example can be modified to work with the simulator in Handel-C simulator instead of the pins on the FPGA.

```c
//OutToSim ! cnt_monitor;    //simulator
```

III. Handel-C DESIGN ENVIRONMENT

Handel-C is packaged as part of a design environment called DK1 which incorporates a modified version of the GNU preprocessor [3]. It is a standard Windows development environment with dockable windows and customizable tool bars. The environment consists of four main parts: workspace window, code editor window, output window, and debug windows. This facilitates the interactive debugging of Handel-C design which occurs in a pre-synthesis manner. Thus the design does not have to go through synthesis, place and route until after functional simulation is fully completed. The turn around time during the simulation process is relatively fast because no second source software needs to be invoked. Currently, Handel-C can be compiled directly into Electronic Design Interchange Format (EDIF) or translated into VHDL. The EDIF/VHDL designs are then processed by commercially available synthesis, place, and route tools which generate designs for a variety of target FPGAs.

To assist the rapid prototyping process, Handel-C supports the use of the Celoxica RC1000 PCI based FPGA board [4,5]. The RC1000 is a PCI bus plug-in card for PC. It has one large Xilinx FPGA (varies BG560 Xilinx FPGA) with four banks of memory for data processing operations, and two PCI Mezzanine Cards (PMC) for input/output with the outside world. Accompanied the RC1000 prototyping board is the RC1000 support software library [5,6]. The RC1000 support software provides host libraries to simplify the processes of initializing and communicating the hardware. The software provides a number of groups of host functions including functions for initialization, handling of FPGA configuration files, control of the RC1000 built-in programmable clock, seamless transfer data to and from the RC1000 FPGA, and error checking and debugging. The library hides the details of pin assignment as the host and board communication is through the mean of PCI bus. The library provides three methods of communicating between RC1000 board and host computer through DMA operation, two unidirectional 8-bit port, and two unidirectional 1-bit port.

IV. INITIAL CLASS ROOM EXPERIENCE

The original plan for investigating Handel-C and an FPGA co-processor boards were to use them in real time and embedded systems courses. This would allow computer engineering students the opportunity to develop a real time test-bed hardware/software environment for real time applications. One of the most difficult problem in real time systems is the verification of non-functional issues including timing. Usually an expensive and complex test system is used to generate input signatures for timing verification. One goal of this project was to use the FPGA, programmed in Handel-C, to generate test signatures and verify functional and temporal correctness in real time. The use of

Figure 2. Producer/Consumer 4 bit Counter Handel-C Example.

The original plan for investigating Handel-C and an FPGA co-processor boards were to use them in real time and embedded systems courses. This would allow computer engineering students the opportunity to develop a real time test-bed hardware/software environment for real time applications. One of the most difficult problem in real time systems is the verification of non-functional issues including timing. Usually an expensive and complex test system is used to generate input signatures for timing verification. One goal of this project was to use the FPGA, programmed in Handel-C, to generate test signatures and verify functional and temporal correctness in real time. The use of
Handel-C in this manner would be designed to allow the development of these test beds by undergraduate students who are familiar with C rather than those that have spent the time to learn VHDL which is an entire course by itself.

This remains an important goal for future offerings of this course, but for the first time we decided to implement more modest goals that would allow us to evaluate the usefulness of Handel C in this environment. It was hoped that any show stoppers would be identified but first having students, some of which had previous experience with other hardware description languages implement modest sized hardware designs using a stand alone FPGA based protoboard (Spartan II XC2S100 by Insight Electronics). Projects included a digital filter, video display, and an advanced implementation of the solution to NP completeness.

Armed with the idea that Handel-C would be beneficial, these students set out to use Handel-C in their project with a mixture of results. For the students that had no hardware description languages background, they were able to learn the basics of the language within a few weeks. One of this group of students designed a parallel filter and found a 30 MHz design can be easily keep up with a 1 GHz Pentium. Another student was able to complete a design to display initial data to a CRT monitor within just a couple of days. His design was comparable in size and speed to that of a VHDL based implementation he had made previously. This student though was an experienced C programmer who had just learned VHDL during this semester so he could only be considered to be a novice VHDL programmer. Another student who was a very experienced VHDL programmer had a very hard time understanding the concepts of Handel-C. He was unable to complete the project successfully. Obviously, these sets of projects are too few in number for us to draw some general conclusions, but there were no obvious show stoppers that became apparent.

The next step that is planned is to perform some additional experiments and analysis to determine if the benefits of Handel-C in this environment are as claimed. The formal specification of the 2-D DCT operation is as follows,

\[
F_{uv} = \frac{c(m)c(n)}{4} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \left[ f_{mn} \cos \left( \frac{(2m+1)\pi u}{2N} \right) \cos \left( \frac{(2n+1)v\pi}{2N} \right) \right]
\]

where,

- \(u, v\) = discrete frequency variables
- \(f_{mn}\) = gray level of pixel at \((m,n)\)
- \(F_{uv}\) = coefficient of point \((u,v)\) in spatial frequency

For an image subdivided into 8x8 blocks of pixels, \(N\) is equal to 8. An important property of the cosine transform is that the two summations are separable. Thus, the DCT can be implemented with one transform and two 8x8 matrix multiplications [10].

C. Greatest Common Divisor

The Greatest Common Divisor determines the greatest common divisor of two numbers. This implementation is based on Euclid algorithm.

\[
gcd(x, y) = x, \quad \text{if } x = y
\]
\[
= gcd(x-y, y), \quad \text{if } x > y
\]
\[
= gcd(x, y-x), \quad \text{if } x < y
\]

V. DESIGN EXAMPLES

In addition to these initial antidotal experiences in the classroom environment the effectiveness of Handel C was compared to VHDL and ANSII C (GNU C) in a more detailed manner by implementing several moderately sized design examples which are described in this section. These Handel-C applications were implemented with the FPGA API library within DK1 environment for communicating with the host computer that were described in section 3. In addition, C/C++ programs were written and executed on the PC host which fully supported the host side programming and runtime communication with the FPGA. The ANSII C implementations were made by compiling generic C code using the gnu C compiler and executing it on a Sun Ultra 10 workstation with a 440 MHz processor.

The design examples include the Data Encryption Standard, Discrete Cosine Transform, and Greatest Common Divisor. Table 1 shows implementation details of three examples in C, Handel-C, and VHDL. No pipelining or other advanced optimization techniques were employed for any of these implementations. A complete description of these three applications now follow.

A. Data Encryption Standard

The Data Encryption Standard (DES) is the most well-known symmetric-key block cipher. DES is a Feistel cipher which processes plaintext blocks of 64 bits, producing 64-bit ciphertext blocks [7,8]. The effective size of the secret key \(K\) is 56 bits; more precisely, the input key \(K\) is specified as a 64-bit key, 8 bits of which (bits 8, 16,...,64) may be used as parity bits. Encryption/decryption proceeds in 16 stages or rounds. From the input key \(K\), sixteen 48-bit subkeys are generated, one for each round. Within each round, it involves a 6-to-4 bit substitution mappings (S-boxes). Each round is functionally equivalent, taking 32-bit inputs \(L_{i-1}\) and \(R_{i-1}\) from the previous round and producing 32-bit outputs \(L_i\) and \(R_i\).

B. Discrete Cosine Transform

The DCT is widely used in image compression application, especially in lossy image compression [9,10]. For example, the 2D DCT is used for JPEG still image compression. The formal specification of the 2-D DCT operation is as follows,

\[
F_{uv} = \frac{c(m)c(n)}{4} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \left[ f_{mn} \cos \left( \frac{(2m+1)\pi u}{2N} \right) \cos \left( \frac{(2n+1)v\pi}{2N} \right) \right]
\]
D. Observations

A total of eight implementations were made of these applications. A Handel-C, VHDL, and an ASCII C implementation was made for the DES, and DCT, applications. The GCD was implemented only in Handel-C and ANSII C. There are many observations which can be made, the first of which is that the number of Handel-C source lines of code lines is generally smaller or comparable to the corresponding VHDL or C implementations. This implies that Handel-C can be used to describe these algorithms in a manner that is at least as abstract and behavioral as VHDL or C. Handel-C implementations are comparable to VHDL in terms of speed (not much slower), but the amount of resources used for the implementation is 2 to 5 times larger than the VHDL implementation. This may be due to the fact that the API libraries associated with the prototype environment are automatically included as part of the design, even though significant portions are not being used and quality of translation from Handel-C to EDIF. Interesting results can also be observed when Handel-C implementation is compared to the C implementation. In the DES case, the C implementation on a SUN Ultra 10 with 440 MHz clock completed only four times faster than Handel-C FPGA implementation even though it had a clock speed that was over 10 times faster. In the case of GCD algorithm, the execution time for 1,000,000 loops of 1 GCD for the C SUN Ultra implementation is 6.08 seconds. Similar number of loops is time for the Handel-C with 19.17 seconds recorded by implementation of 23 MHz. The C implementation clock speed is almost 20 times faster, but the execution time is only 3.15 times faster. It should again be noted here that no pipelining was used in any of these implementations which would improve all hardware representations by a factor of four or more.

VI. CONCLUSION AND FUTURE RESEARCH

Much research needs to be done in determining if Handel-C is a good choice to replace traditional hardware descriptions languages in the real time systems arena. The experiences highlighted in this paper tend to confirm that Handel-C shows much promise. It appears that it can be used to create designs that are comparable to traditional hardware description languages in terms of speed. The size of the designs tend to be larger than their HDL counterparts. The reason for this and the manner in which this size discrepancy behoves for large size applications remains an area for future investigation. It also appears that Handel-C is easily learned by traditional software designers but there is some evidence that there is a high retraining costs for traditional hardware designers if they are to use this language. Future research into this hypothesis should be pursued as well as the possible use of other software oriented languages such as SpeC and SystemC.

VII. ACKNOWLEDGMENTS

This work is supported by NSF grant 0091853. Special thanks goes to James Higgins of Insight Electronics, Huntsville, Alabama. Without his help, implementing these example applications would not have been possible.

VIII. REFERENCES