Compiling Video Algorithms into Hardware

Ian Page
Hardware Compilation Research Group Leader
Oxford University Computing Laboratory
(DRAFT - No References)

July 1997

Abstract

Our major research goal is to make hardware/software co-design into just another smart compilation technology. Our current approach is to provide compilation and evaluation tools which enable programmers to describe the mix of hardware and software they require.

Handel-C is a programming language which we have designed for compiling programs into hardware implementations. A compilation and simulation system has been built which maps Handel-C programs into descriptions of hardware as gate-level netlists. We then use commercial software to map these netlists onto FPGAs (Field-Programmable Gate Arrays).

A number of reconfigurable computers have been designed and built as platforms for our hardware-software codesign work. Some of these boards are commercially available through our industrial partners and in particular Embedded Solutions Limited.

We describe some applications of our hardware compilation technology to the area of real-time video processing. A simple video generator program is shown in detail. We describe more briefly other applications which we have implemented such as computer games and object tracking from video camera images. Some of these applications run on a single stand-alone FPGA, other more complex ones use a single FPGA, a conventional microprocessor and video input and output modules.

1 Background

We believe that, at some point in the future, designers will routinely be using programs as the major and perhaps only description of complete systems. In this vision of the future, many hardware-software systems will be created by programmers rather than mixed teams composed of programmers and electronic engineers. The work of our group is focussed on building the tools and methodologies to make this dream a practical reality.

The primary reason for implementing some system functions with hardware rather than with software is because of gains in speed of execution. Thus, as hardware is only fast because of its inherent parallelism, it is clear that the final hardware descriptions must exhibit the appropriate levels of parallelism. If the behavioural description does not contain, or strongly hint at, the appropriate parallelism, then it must be extracted automatically. Our experience is that this very difficult task is not, in general, yet within the grasp of automatic tools. However we have found that with appropriate languages

and tools it is in fact quite reasonable to ask that programmers provide this level of description themselves.

In summary, our commitment is to undertake research into the building of tools that will allow real-time, parallel programmers to design and build industrially relevant systems by means of programs written within a single computational framework.

In this paper we briefly review our work in hardware compilation and then describe some applications of our technology to building applications which deal with real-time video signals. Some of these applications have real-time video input and output such as a moving object tracker, others are video generators such as games and animated logos.

2 The Handel-C and Rapid Prototyping

The Handel-C language has been developed at Oxford for describing programs which are compiled into hardware. The language is necessarily different from C itself because of the different nature of hardware and software implementations. Handel-C is aimed at hardware implementations and has some features which support that mapping. In particular, the C language itself is not adequate to our task because it is inherently a sequential language and it is crucial to exploit parallelism in hardware implementations.

Handel-C is closely related to a simple subset of C, extended with the parallelism and communication constructs (from Hoare's CSP) and arbitrary width variables and expressions. Programs, written in Handel-C can fully exploit the explicit parallelism of the programs themselves and serial execution is only introduced where explicitly required. We have successfully used the system to compile and execute applications in graphics and machine vision, video compression and decompression, cellular automata, neural networks, database servers, and a number of other applications as well as the ones reported here.

New hardware paradigms have emerged in recent years which support the construction of hardware at millisecond speeds. A key element has been re-configurable hardware in the form of FPGAs (Field-Programmable Gate Arrays). Using FPGAs enables us to compile algorithms directly into hardware, and avoid the need for a computer (i.e. an interpreter of stored instructions) at all. As this is often the bottleneck in a computational process, speed increases can be gained by getting rid of the computer!

Although not near as fast or as dense as application-specific silicon, FPGAs can be programmed by purely software means to implement hardware systems of reasonable size and speed. The tremendous advantage of using hardware compilation together with FPGAs is that it enables us to construct realistic, working hardware-software systems in hours, or even minutes. The video games which we detail later in the paper can typically be turned around in about ten minutes.

3 A Simple Video 'Game'

The program to be described might be the core of a video 'bat and ball' game. The screen consists of a background colour, surrounded by a border and a moving square (the 'ball') which moves in straight lines and bounces off the border. A very simple application has been chosen so that the whole of the program can be described here (although some liberties have been taken with the ordering of the program fragments). The major features of the language and many of our techniques for dealing with real-time video will become apparent even from a small example.

It is hoped that a programmer looking at this code will be able to gain confidence that hardware applications like this can be written fairly easily by someone who pos-

sesses programming skills rather than electronic design skills. It is also hoped that an electronics designer looking at this program will recognise a simpler way of designing than is provided by typical hardware description languages.

At the top-level the program consists of the parallel composition of three processes as shown below.

```
unsigned int 10 vx, vy;  // current pixel address

void main () {
  par {
    sync (vx, vy);
    display ();
    per_frame_update ();
  }
}
```

This fragment already highlights a major difference between Handel-C and conventional sequential languages like C. This program calls for three separately executing processes to run concurrently. The first one is responsible for producing (approximately) standard VGA video sync signals and for updating the `vx, vy` variables which track the current pixel position both during display and during the hidden parts of the video signal, namely the horizontal and vertical retrace intervals. The other two processes synchronise themselves to this one by inspecting the values of these two variables. The `display` process is responsible for producing the video data signal and the `per_frame_update` process changes the variables used by the `display` process to advance the state of the 'game'.

Another difference with C is that variables can be typed with an arbitrary bitwidth; here the pixel coordinates are represented in 10 bits each. Width inferencing and typing systems ensure that the programmer can declare widths only where necessary and the compiler will usually be able to work out the widths of objects that the programmer has not explicitly specified.

The following fragment is the declaration of a video output register, the (initialised) ball position and 'velocity' variables and the process which creates the video signal. This program was made to run on the demonstration board supplied by Xilinx which has a 4010 part on it. We made up a simple D/A converter by wiring binary-weighted resistors to 15 of the pins of this chip to give three analogue outputs for the red, green, and blue signals each of 5-bits resolution. These signals together with the horizontal and vertical sync signals are simply wired to a standard VGA monitor to observe the output of the program. The `interface` structure declaration defines the mapping of the `video` register onto the FPGA pins.

```
unsigned int 15 video;  // the video output register - 3 x 5-bit RGB
unsigned int 10 bx = XLNG/2, by = YLNG/2;  /* ball position  */
unsigned int 10 dx = 2, dy = 2;            /* ball increment */


interface bus_out () video_out (video)  // map it onto these pins
  with { data = { "P10", "P9" , "P8" , "P7" , "P6" ,
                  "P5" , "P4" , "P3" , "P84", "P83",
                  "P82", "P81", "P80", "P79", "P78"}};

/* some colour definitions for 15-bit video output */
```

```
macro expr black  =  0x0000;
macro expr white  =  0x7FFF;
macro expr red    =  0x7C00;
macro expr green  =  0x03E0;
macro expr blue   =  0x001F;

macro proc display () {
  while (1)
    /* this inner loop runs once every pixel time (20MHz clock) */
    if (visible (vx, vy))

      /* put a blue square at position (vx, vy) */
      if (vx >= bx  &&  vx <= bx+BALL_SIZE  &&  vy >= by  &&  vy <= by+BALL_SIZE)
        video = blue;

      /* put a border around the visible region */
      else if (vx <= BORDER-1  ||  vx >= XLNG-BORDER ||
               vy <= BORDER-1  ||  vy >= YLNG-BORDER)
        video = red;

      /* elsewhere put in a background colour */
      else video = blue | green;  // i.e. cyan

    else video = black;  // suppress video outside visible scan
}
```

The display process runs continuously in a 'forever' `while` loop. If the current pixel position `(vx, vy)` is visible then the nested `if` statement branches one of three ways depending on whether the current pixel is with a border region, or with the ball, or is simply part of the background. A different pixel colour is assigned to the video output register for each of these situations and the video signal is additionally forced to black outside the visible region.

The Handel-C language defines that an assignment takes exactly one clock cycle. It can be seen that the most that happens in one invocation of the body of the `while` loop is a single assignment; there is no sequential composition anywhere in the body. Thus we know immediately that this inner loop will run in exactly one clock cycle and that the video register will be updated at the full 20MHz video clock speed.

The frame rate update process runs continuously and waits for the first non-visible pixel. At this point it calculates the new ball direction and updates the ball position. There are thousands of clock cycles for this process to execute so there is no particular advantage to having it run in parallel. However there is no particular advantage to having it run sequentially either. The saving in hardware from any attempt to share the adders sequentially is likely to be more than outweighed by the addition of multiplexors to achieve this sharing.

```
macro expr BALL_SIZE = 16;
macro expr BORDER    =  4;

macro proc per_frame_update () {
  while (1) {
    /* runs just after last pixel is drawn */
    if (vx == XLNG  &&  vy == YLNG-1) {
```

```
      par {
        /* reverse ball direction on hit with border */
        if (bx >= XLNG-BALL_SIZE-BORDER-dx) dx = -2;
          else if (bx <= BORDER-1+dx) dx =  2;
        if (by >= YLNG-BALL_SIZE-BORDER-dy) dy = -2;
          else if (by <= BORDER-1+dy) dy =  2;
      }
      /* update ball position */
      par { bx += dx;  by += dy; }
    }
  }
}
```

This final fragment shows the compiler directives defining mainly the target FPGA family and the definition of a number of expressions representing various combinations of the x, y position variables. These definitions would normally be part of a library which is why they are parametrised on the name of the x, y pixel counter variables.

The `XLNG, YLNG` constants represent the size of the visible region for this particular video signal. The `hblank, vblank` expressions calculate the horizontal and vertical blanking periods, i.e the non-visible periods. They are generated by logical expressions involving the current pixel position (the subscripts are bit field ranges). The sync signals for the monitor are generated similarly and were designed by inspecting the timing definition of a VGA video signal.

```
/* define FPGA details for technology mapping phase */
set family = Xilinx4000E;
set part   = "4010EPC84-3";
set clock  = external "P13";

/* Definitions of raster scan constants */
macro expr XLNG          =  512;
macro expr YLNG          =  480;
macro expr hblank(x)     =  x[9];
macro expr vblank(y)     =  y[9] | (y[8] & y[7] & y[6] & y[5]);
macro expr hsync(x)      =  hblank(x) & ((x<-7) >= 12) & ((x<-7) <= 87);
macro expr vsync(y)      =  (y & 0x3FE) == 490;
macro expr end_scan(y)   =  y[9] && y[3] && y[2];
macro expr end_line(x)   =  x[9] && x[6] && x[5] && x[4] && x[3] && x[1];
macro expr visible(x,y)  =  !hblank(x) & !vblank(y);  // ie. not blanking

/* The sync generator */
macro proc sync (x, y) {
  interface bus_out() hsync_out (hsync(x)) with {data ={"P77"}};
  interface bus_out() vsync_out (vsync(y)) with {data ={"P70"}};
  while (1)
     par { if (end_line(x)) y = (end_scan(y) ? 0 : y+1);
           x = (end_line(x) ? 0 : x+1); }
}
```

Fig. 1 shows how the various sync signals relate to the visible portion of the video signal. It has proven a relatively simple matter to change the video generation process to support different screen resolutions, clock rates and refresh rates.

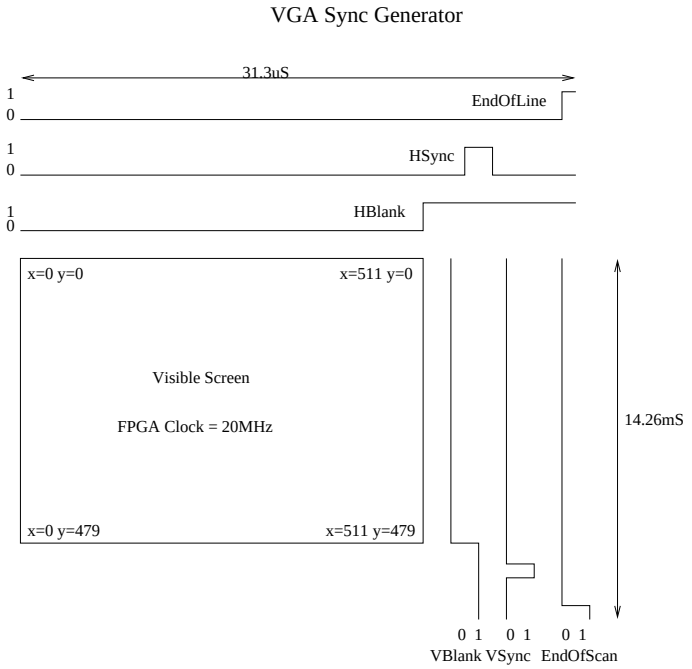

Fig. 1. Diagram of sync signals

4 Video Motion Tracker

An application is described to demonstrate the use of hardware compilation in a challenging application area. The task is to extract from a real-time video source a frame-by-frame list of the bounding boxes of objects which are moving. The context of this application is an automatic security camera system which will usually relay a wide-angle view of the entire visual field, but which will pan and zoom the camera onto any moving objects within the scene. The system could be used to record full-frame images of all moving objects onto videotape or to alert a human operator that a particular sort of moving object had been detected, perhaps an intruder. It is implemented using a Harp board, together with frame grabber and video card. The Harp reconfigurable computer, designed by our group, is a combination of a transputer and an FPGA and is described in more detail later in the paper.

The transputer-based frame-grabber digitises the full video image and then subsamples it to produce a 128x128x8-bit grey-scale image stream along its output link. This video stream is taken into the HARP module via one of the transputer's links and is immediately passed to the FPGA using the transputer bus.

The FPGA is used to difference successive digitised images and threshold the result to determine which areas of the image are moving. Figure 2 shows the data flow through the application. The same FPGA then uses a region-filling style of algorithm to explore the changing regions of the image in order to determine the boundaries of the moving objects. Regions which fall below a given area threshold are then discarded and the list of remaining bounding boxes is returned to the transputer for further processing. While the differencing and region-filling algorithms are not ideal, they are computationally cheap and produce hardware which is small enough to fit into the single Xilinx 3195 FPGA available.

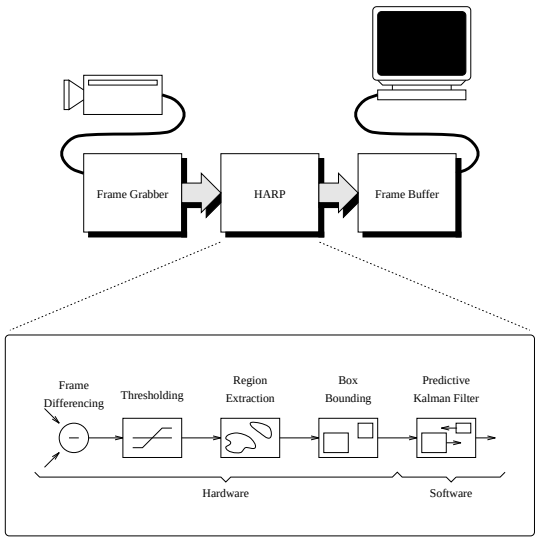

Fig. 2. Tracking Algorithm Flow

To improve the noise-immunity of the system, the transputer will first merge overlapping bounding boxes. These normally result from the FPGA algorithm splitting a single moving object into multiple objects because of the crudeness of the image differencing algorithm. The floating point unit of the transputer is then used to implement a multiple object predictive Kalman filter to track the moving objects over time. The filter uses the position, size and first derivatives of these quantities to model and predict the temporal behaviour of the bounding boxes. It maintains a 'confidence level' for each of the rectangles which models the likelihood that each rectangle corresponds to a moving object in the scene. An object is deleted when the filter's confidence level for that object drops below a given threshold. Similarly, the filter creates an object description when a new moving object is detected.

The final output of the system is a stream of video images passed from the HARP board to a transputer-based framebuffer for viewing on a monitor. The output video images can be presented in a number of ways, one of these being with the imposition of the bounding boxes as coloured overlays. The system performance that this application achieves is between 12-23 frames/sec, depending on what is happening in the scene. When the same application is run on a transputer alone, its performance drops by a factor of six. This ratio could be even greater if the hardware platform allowed video input/output into the memory directly accessible to the FPGA rather than being routed via the transputer as is necessary on the HARP board.

The performance of the hardware based tracker is impressive compared to the standard processor solution, especially considering that it is obtained from two general-purpose chips, and has been programmed in a general-purpose language, with absolutely no intervention being made by the programmer below the level of the programs he wrote. Even more impressive is the development time: Handel-C enabled the system to be written by an undergraduate programmer with no knowledge of hardware, as

part of an 8 week project. The hardware compiler and reconfigurable computer enabled dozens of implementations to be evaluated in this time.

5 Other Examples

We have constructed a number of other video processing systems.

5.1 Video compression/decompression

We have built the following video compression and decompression applications:

- Hadamard (compression + decompression)
- Wavelet (compression + decompression)
- Replay (an Acorn standard; decompression only)

These video compression/decompression applications all run on our Harp reconfigurable computer which consists of a T805 transputer, a Xilinx 3195 FPGA and 4 Mbytes of DRAM connected by a bus. A 2-90MHz frequency synthesiser provides the clock signal for the FPGA. In addition the FPGA has up to 128K of SRAM as a fast local cache. These boards comply with the TRAM standard, and we generally host them on a TRAM motherboard which is in turn hosted by a PC. Commercially available TRAM boards, such as frame grabbers and video cards, can be added to the motherboard to produce more sophisticated systems. A motherboard can host more than one Harp, giving a system of multiple interconnected processors and FPGAs.

5.2 Video processing

We have built the following processing applications:

- Edge detectors
- Threshold
- Colour extraction (highlight all 'red' objects)
- Mapping a video sequence onto a sphere

These all run on the RC1000-II board from Embedded Solutions Ltd. It is a standard half size ISA bus card equipped with a Xilinx XC4000E series FPGA in PLCC-84 format. This is socketed, and may be replaced by any pin-compatible FPGA from the Xilinx XC4003E to the XC4010E device range. It is programmable with the Handel and Xilinx Xactstep tools, and a PC provides server microprocessor support. An on-board serial PROM can be used to configure the FPGA for dedicated run time applications. Otherwise the FPGA can be configured over the ISA bus using the PC-based interface programs provided.

The board is equipped with an Industry Pack mezzanine daughter-board slot carrier directly interfaced to the FPGA. The Industry Pack interface is essentially a 16 bit, memory-mapped interface working at either 8 or 32 MHz. A choice of over 400 I/0 modules are commercially available for Industry Pack, covering the needs of most prototyping and low volume bespoke application needs. This mezzanine format is ideal for prototyping custom I/O or additional hardware assist circuitry for the FPGA, such as SRAM or graphics accelerators. In addition general-purpose input/output capability is

provided by a 50-pin SCSI-2 style panel connector connected to the FPGA. The board has a programmable clock which operates over the range 400KHz to 100MHz and can be configured from the host PC. In addition the 8.33MHz ISA clock, and the Industry Pack 8MHz and 32MHz clocks are available for clocking the FPGA.

5.3 Computer Games (with no computer!)

We have built the following 'computer games' on a single FPGA on a Xilinx demonstration board as described in the foregoing text. We have found that it is relatively straightforward to get a single FPGA to perform all the functions of these games, namely video generation, the game itself and the user interface as well. Not that this is without any special graphics hardware or even a screen bitmap - the FPGA and a crystal oscillator is all there is!

- Tetris
- 2-D racing car game
- Bouncing ball demos (one with a moving background)
- Breakout
- Pong (the original t.v. tennis video game)

5.4 Animated logo generators

These are a group of applications which run on various boards. The first two generate simple text from on-FPGA rom elements and map it dynamically onto the screen.

- Flight over logo plane
- Rotating, zooming, and 'wobbly' logos
- Rotating earth (using NASA imagery)

These applications all run on a single FPGA. The rotating earth is modelled on the BBC logo so familiar to television viewers. It runs in a single FPGA backed up by a single SRAM chip to hold the coloured, textured bitmap of the earth as derived by NASA from a series of satellite photographs. The application dynamically maps the bitmap onto a rotating sphere on the screen.

6 Conclusions and Future Work

Much exciting work remains to be done. For example the design of a single language for hardware-software co-design, the design and implementation of hardware and software compilers for the languages, optimisation of hardware implementations, asynchronous hardware implementations, automatic design and implementation of microprocessors, knowledge-driven and language-driven placement strategies for silicon implementations, new architectures for FPGAs and much more.

It seems likely to us that the combination of high-level, programming-based approaches to hardware-software codesign together with reconfigurable hardware will fundamentally change the ways in which the digital systems of the future are designed and built. If true, this has far-reaching implications for the necessary skills basis that

industry must adopt and nurture to be successful in the future. Hardware engineers, who already have an excellent understanding of the nature of parallelism, will have to become more programming-literate, and programmers will have to become more aware of parallelism and the time and space implications of the programs that they write.

We are continuing to develop all aspects of the work reported above, including languages, program transformations, compilers, hardware platforms, and actual applications. However, we have a particular focus for the next phase of work which is defined by the 'Aspire' Project. This is a partnership between Oxford University, ARM and Atmel-ES2. The goal of the project is to build a 'shrink-wrap' product for hardware-software co-designers which will allow them easily to design Asics for embedded computing systems based on ARM RISC processor technology. The major novelty of the project is that the ultimate target of the compilation process is an ASIC which has an ARM processor core and the designer's application-specific hardware on the same chip.

Acknowledgements.

I wish to thank all the members of the Hardware Compilation Group without whom much of the work reported here would not have been accomplished. I particularly want to thank Dominic Plunkett who added the D/A converter circuit to the Xilinx demonstration board and provided the Handel-C library routines to use it, Matthew Bowen who built the video tracker demonstration, and Jonathan Saul who provided some of the text describing this application.

Our work has been supported in part by the European Union (Esprit/OMI programme), the U.K. Engineering and Physical Sciences Research Council (EPSRC) and many industrial partners.

7 Contact Information

Ian Page, F.I.E.E.,
University Lecturer, Fellow of St. Hugh's College,
Leader, Hardware Compilation Research Group,
Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford OX1 3QD, U.K.
Tel: (+44/0) 1865 273853 direct, 273840 sec, 0378 058921 mobile, 273839 fax.
www: http://www.comlab.ox.ac.uk/oucl/people/ian.page.html