ECE4530 Fall 2015:
The Codesign Challenge
I Am Seeing Circles

Assignment posted on Thursday 11 November 8AM
Solutions due on Thursday 3 December 8AM

The Codesign Challenge is the final assignment in ECE 4530. This project is an exercise in performance optimization: you will start from a given reference application on a Nios-II processor. You have to improve the performance of the reference application as much as possible, using the hardware/software codesign techniques covered in this course. Typically, you would design a hardware coprocessor. In addition, you would also optimize the driver software, and/or modify the system architecture.

The major constraints in the design are as follows.

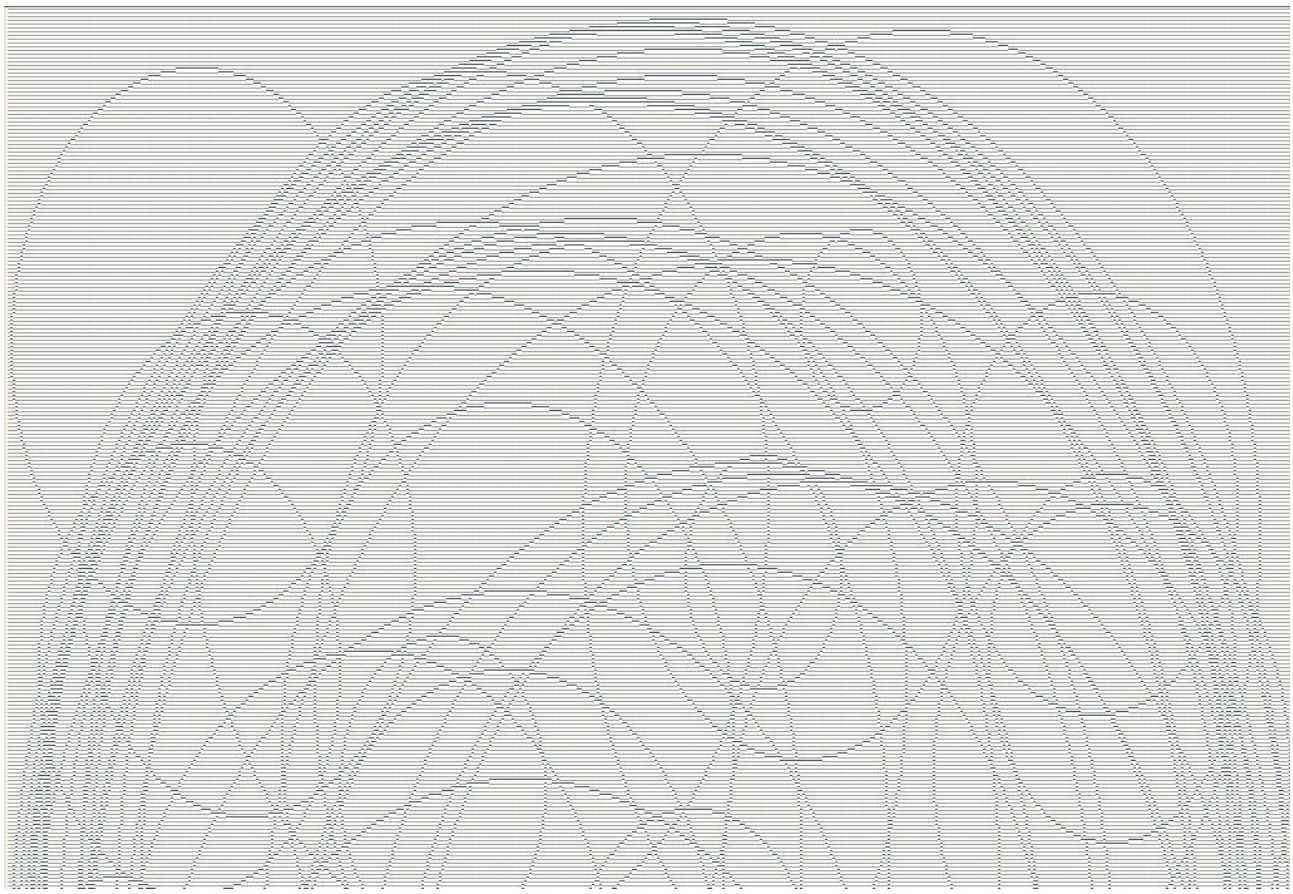
- The final result has to run on a DE2-115 board
- The final result has to be turned in before the submission deadline, 3 December 8AM
- The testbench needs to remain in C on the NIOSII core; the assignment precisely defines the API for the accelerated design

Application: Bresenham Circle Drawing

The assignment is an application of the Bresenham Circle Drawing Algorithm. This algorithm is used to draw circle shapes on a discrete grid of pixels. A nice feature of the algorithm is that it does not require floating point arithmetic. Circles can be drawn using integer arithmetic alone.

In a nutshell, the application works as follows. The reference design contains a video pixel memory of 32 Kbytes, which is used to store the pixel of a 512 by 512 pixel display. Pixels are binary values, and can be on or off. The testbench will prepare a set of 50 circle parameters, each defined by a centerpoint (x, y) and the a radius r. The testbench ensures that the circles will always fit completely on the display, and that there are no clipping effects near the edges. Starting from the list of 50 randomly generated circle parameters, you accelerator needs to fill the pixel memory as fast as possible with a drawing for each circle. A circle is drawn by turning on all pixels on the edge of the circle. The performance of your design is measured by the number of clock cycles required to draw all circles in the pixel memory. The correctness of your solution is verified by computing a checksum over the pixel memory, and by comparing that checksum to the checksum of the same testbench over the reference implementation in software.

There is no visualization hardware included in the reference design, but you can print the content of the pixel memory on a terminal display (using a tiny font). A part of the resulting image is shown below. The objective of this assignment, though, is performance.



The materials from the assignment

The following materials are included in the assignment.

- challenge.pdf, the specifications.
- challenge.qar, a quartus archive with the reference implementation.
- bcircle.pdf, a document by J. Kennedy (Math Dept, SMC) to explain the basic operation of the Bresenham circle drawing algorithm.
- circle.c, a reference implementation of the testbench which you can run directly on a PC

The reference implementation

The reference implementation in challenge.qar contains the following components.

- A Nios-II/e core

- A 32KByte on-chip memory for data and program storage
- A 32KByte Pixel Memory, implemented as on-chip RAM attached to the Avalon Memory Bus
- A timer for performance evaluation
- 3 PIO ports to drive the HEX display and LEDs (these are not use by the testbench, but you may use them for your accelerator).
- A JTAG UART

The pixel memory shall use the following convention. Within a byte, pixels are stored from LSB to MSB. A byte contains pixels with a single row (same y-coordinate) but different columns (different x-coordinate). Pixel (X, Y) is store in pixel byte memory location (y *512 + x) / 8, and occupies bit x mod 8 within the byte. Your design needs to respect this storage order so that the checksum is computed correctly.

In order to compile and run the testbench, you need to prepare a board support package as follows. It needs to map all sections from the program into the on-chip memory (NOT the pixel memory). The pixel memory is only used to store pixel data and nothing else. The access of the pixel memory is done through read and write operations from absolute memory addresses/

```
#include "system.h"
unsigned char *screen = (unsigned char *) PIXELMEMORY_BASE;
```

Reference Testbench

Please consult the program `main.c` in the software subdirectory. The following documentation is only a quick walkthrough of the testbench operation.

The main system constants are defined early in the file. There are 50 circles to be drawn on a 512-by-512 pixel memory.

```
#define SCREENSIZE 512
#define SCREENBYTES (SCREENSIZE * SCREENSIZE / 8)
#define NUMCIRCLES 50
```

The testbench sequence is found in the main program. Your acceleration needs to come in the function `plotallcircles_hw`. You need to keep the testbench intact.

```
int main() {
  unsigned n1;

  alt_timestamp_start();
```

```
  // generate test code
  driverdata();

  // reference case
  clearscreen();

  n1 = alt_timestamp();
  plotallcircles_ref();
  n1 = alt_timestamp()-n1;

  printf("Reference Plotting time is  %d\n", (unsigned) n1);
  printf("Reference Checksum is       %d\n", checkscreen());

  // accelerated case
  clearscreen();

  n1 = alt_timestamp();
  plotallcircles_hw();
  n1 = alt_timestamp()-n1;

  printf("Accelerated Plotting time is  %d\n", (unsigned) n1);
  printf("Accelerated Checksum is       %d\n", checkscreen());

  return 0;
}
```

The circle specifications are created by a function `driverdata` and they are stored in three arrays. The output of the drawing routine is to be stored in the pixel memory. Checksums are computed directly over the pixel memory by the NIOS-II.

```
// input (in onchip memory)
unsigned dx[NUMCIRCLES];
unsigned dy[NUMCIRCLES];
unsigned dr[NUMCIRCLES];

// output (in pixel memory)
unsigned char *screen = (unsigned char *) PIXELMEMORY_BASE;
```

If you configure and run the program as it comes with the assignment, you will see the following output. Minor variations are possible.

```
Reference Plotting time is 14892988
```

```
Reference Checksum is      1067143
Accelerated Plotting time is 14893068
Accelerated Checksum is      1067143
```

Your speedup will be computed as (14892988 / accelerated plotting time). The value 14892988 can be considered absolute and will not be affected when you use software optimization compiler flags. Obviously, your checksum value must match that of the reference testbench.

Ranking Criteria

All designs will be strictly ranked from best to worst. This section defines what 'best' means. We will use the following metrics to evaluate the rank of your design.

- Functional Correctness: This requirement is mandatory for all designs. Your design has to work, in order to be considered for ranking. If it does not work, you will be automatically ranked last. Working means: your design computes the correct checksum.
- Metric 1: The speedup of your design following the formula given above. Higher is better.
- Metric 2: The area efficiency of the resulting design, expressed in LE. An LE is a Logic Element, a unit of hardware in a Cyclone IV FPGA. A lower LE cell count corresponds to a smaller design.
- Metric 3: The turn-in time of your design and report, as measured by the turn-in time on Scholar. Turning in the solution earlier is better. Note that, if you turn in the design multiple times, only the latest turn-in time will be used.

Two designs will be compared as follows, to determine their ranking order. If a design is functionally not correct (i.e. fails the testbench), it will automatically moved to the last rank. If multiple designs are not operational, they will all share the same lowest rank. All functionally correct designs will get a better and unique rank.

- First, the speedup (Metric 1) will be compared. If there is a difference of more than 5% between them (with the fastest design considered 100%), the fastest design will get a better rank. If, one the other hand, the difference is smaller than 5%, Metric 2 will be used as tie-breaker.
- Metric 2 will be used in a similar way to compare two designs. If the difference in area efficiency between two designs is larger than 5%, then the smallest design gets the better rank. Otherwise, if they are separated less than 5%, Metric 3 will be used as tie-breaker.

- Metric 3 will be used as a final metric in case the ranking decision cannot be made using Metric 1 and Metric 2 alone. In this case, the design turned-in earlier wins.

The grade for your project is determined as follows.

- 70 points of the grade are determined by the ranking of your project as described above. The best design gets 70 (out of 70) points, the worst design gets 30 (out of 70) points, and all other designs are linearly distributed between 30 and 70 points.
- 30 points of the grade are determined by the quality of the written documentation you provide with the solution. The requirements are listed below.

Accelerating the Bresenham Algorithm

The main objective of this assignment is to make the testbench run as fast as possible, using hardware software codesign. There are several layers to this problem and it is worthwhile to carefully think them through.

- First, there is the Bresenham circle drawing algorithm itself. You can study the code of the reference implementation and observe that this is a simple while loop that sequentially generates the pixels from an octant. Each iteration, 8 pixels are then plotted to obtain a full circle.
- Next, there is a sequence of 50 Bresenham circles, each with a different centerpoint and radius. You cannot predict these specifications; your accelerator will need to be able to accept flexible input.
- The access to the pixel memory is sequential, since the memory is attached as a slave to the Avalon Memory Bus. Hence, building an accelerator needs to consider not only the computation tasks, but also the memory access bottleneck.
- The reference testbench takes over 14 million clock cycles to generate just 50 circles. That is more than a quarter of a million cycles per circle. This already indicates that the reference implementation is inherently inefficient.

Given these layers, there are several optimization mechanisms to consider.

- Avoid using a random 'let's try this and see' strategy. Very likely this will make you forget something, or it will lead you into a local optimum while loosing sight of the big picture. Always keep an objective in mind (eg. a desired acceleration factor), and measure your progress in how close you are to your objective. Such a final objective should be derived through a reasonable back-of-the-envelope calculation. Always look for the bottleneck in the *overall* system.

- The Bresenham circle drawing algorithm can be quite easily converted into a custom instruction or an FSMD. You can also evaluate the possibility of generating more than one step per cycle (ie. 16, 24, .. pixels per cycle), or the possibility of creating multiple concurrent circle generator modules.

- The pixel memory itself can be customized as well. While you cannot change the requirement that it must provide an Avalon Memory Mapped Slave Port with 32 Kilobyte of data, nothing prevents you from adding additional ports, replacing the memory module with multiple memory modules, changing its architecture from SRAM to registers, and so forth. Think carefully about this aspect, and how it relates to the design of the pixel drawing engines.

- The key to succeed in this assignment is NOT in a monster session of back-to-back all-nighters a few days before the deadline. The key is to work on this design in little bits over a longer period. Design is a creative process, and that takes time. Ideas take time to develop. Think about your strategy and discuss with your colleagues. Since this is an open-ended assignment, and since you are competitively graded, it's not in your advantage to disclose all your ideas. However, a brainstorm session or two with your peers may do wonders to sharpen your insight in this assignment.

What to turn in

You have to deliver the following items for the project result:

- A qar file of your final design. It must contain a Nios processor that is capable to execute the evaluation testbench. Make sure it is a complete system, and, pay particular attention to the following items.

  - Include the driver software for the resulting system as well. By default, the software is not included as part of a qar file - you have to include it by hand.
  - Include the bitstream (sof file) for your design. This can be selected as an option during qar file generation.

- Any Verilog code that you've developed.

- A PDF document that describes your resulting design. Please explain your design strategy, the architecture of your hardware/software solution, and overall observations on the design.

Good Luck!!