Supported by:   

2009 MEMOCODE Co-Design Contest Rules

January 12, 2009

1. Contest Organization

The 2009 MEMOCODE Co-Design Contest is organized as a part of the 2009 MEMOCODE Conference (http://csg.csail.mit.edu/Memocode2009). This contest has been held twice previously in 2007 and 2008. The 2009 contest is organized by Forrest Brewer (UCSB) and James C. Hoe (CMU). For more information about this contest, please visit http://www.ece.cmu.edu/~jhoe/mc09.html.

The contest rules are subject to change (with notice) until March 1, 2009.

The 2009 contest is generously supported by IEEE CEDA, Bluespec and Xilinx.

2. The Design Problem

On March 1, 2009, a design challenge will be posted on the contest website (http://www.ece.cmu.edu/~jhoe/mc09.html). Each participating team has until March 31, 2009 to submit a working solution. The solution must work correctly to be considered for awards.

3. Tools and Platforms

You may use any HW and SW design methodologies at your disposal. Formal methods are encouraged but not required. You are allowed to make use of existing IPs available to you.

You may use any development platforms at your disposal. The platform can include any number of processors and FPGA devices. The processor and/or the FPGA should have access to at least 512 MBytes of memory.

The reference platform supported by the contest organizers is the Xilinx XUP development board (http://www.digilentinc.com/Products/Detail.cfm?Prod=XUPV2P&Nav1=Products&Nav2=Programmable) with 512MB of DRAM. For those of you using XUP, we provide a reference EDK project for the Xilinx XUP board that implements a basic interface library to enable communication between the software running on the 300MHz embedded PowerPC405 and the 100MHz FPGA fabric through the DSOCM interface. You may develop or acquire any other interfacing schemes you prefer.

4. Design Disclosure

To be considered for an award, you will need to submit your design to the judging panel for review. The judges will make a best effort to treat any source code in the submissions as confidential. We will not publish your source code without your permission.

A winning team is expected to make a presentation about their design and design methodology at the conference. A winning team is also expected to submit a 4-page short paper for publication with the conference's formal proceedings.

If your entry involves the use of confidential/proprietary IP, design methodology or platform, please consult with the contest organizers before you begin.

5. Contest Judging

A subjective element of judging is based on the elegance of the solutions as determined by a panel of judges. In addition, a design is evaluated both for absolute performance and normalized performance (efficiency) in the objective portion of the judging.

Separate awards are given in the two performance categories. The performance metric used is the geometric average speedup (over a prescribed set of test inputs) of your implementation relative to a reference software-only implementation. For the absolute performance category, speed up is computed using the wall-clock execution times. For the normalized performance category, speed up is computed using a normalized execution time as specified in **Appendix A**.

For each category of the performance competition, the entries are ranked overall by the value (*Rank_performance* + *Rank_elegance*). In the case of a tie, *Rank_performance* is the tie-breaker.

The winning design must work correctly. We rely on the honor systems for initially reporting the performance data. In the case you should be among the top finishers, we will arrange for verification of the correctness of your design and the performance data.

6. Awards

The top finisher in each of the two performance categories will be invited to contribute a 4-page short paper and to present their design and design methodology at the MEMOCODE 2009 conference. We will offer a $1000 cash prize for the winning team in each of the two performance categories. A team however is only eligible for winning the $1000 cash prize in one category. If the same team finishes first in both performance categories, the judges will select a runner-up team to receive a $500 cash prize.

In addition, every team that submits a complete and working entry will be invited to submit for review a 2-page abstract for the formal conference proceedings.

7. Eligibility and Entry Instructions

This contest is open to industry and academic institutions (faculty and/or students). A team may include both industry and academic members. There is no limit on the size of a team. There is no limit on the number of teams per institution. Each person can participate in only one team.

To enter the contest, the corresponding team member should email jhoe+mc09@ece.cmu.edu with the team name, and the names and affiliations of the team members. On March 1, 2009, we will send the register teams the SW-only reference design and test cases (in portable C) and the XUP HW/SW interface design (C and Verilog in an EDK project). There are no obligations associated with requesting the reference designs. A team can enter until the close of the contest on March 31, 2009.

Before 11:59:59PM US Eastern Time on March 31, 2009, submit your design in a tar or zip file via http://www.ece.cmu.edu/tools/upload/. Use jhoe+mc09@ece.cmu.edu as the recipient address. After your submission, you will receive a confirmation email within 24 hours. If you do not receive this confirmation email after 24 hours, please contact jhoe+mc09@ece.cmu.edu.

Your submission should contain the following items in a single zip or tar file.

1. Design Source Files
2. Design Performance Metric as obtained using the testbench specified with the design challenge
3. Design Documentation describing your FPGA platform, your design methodology, the organization of your design and its theory of operation, and a brief analysis of its performance and bottlenecks. Documentations in the form of PowerPoint slides are perfectly acceptable. Keep in mind, the judges' subjective sense of elegance is likely to be influenced by the quality and conciseness of the documentation.

The judges will make a best effort to treat Item 1 (Design Source Files) as confidential. The information in Items 2 and 3 are considered public.

Appendix A. Platform Normalization

For the normalized performance (efficiency) competition, speed up is computed using the normalized execution time

$$Time_{normalized} = Time_{measured} \times NormalizationFactor$$.

If you use an XUP board (with a -6 XC2VP30) to enter the contest, your performance normalization factor is 1. If you use a more "capable" platform, your normalization factor will be greater than 1. If you use a less capable platform, however, your normalization factor cannot go below 1.

If you use a platform different from XUP, you are responsible to obtain the normalization factor for your platform as given by the following formulas.

$$NormalizationFactor = f_{FPGA-performance} \times f_{CPU-performance}$$

with

$$f_{FPGA-performance} = \max\left\{1, \sum_{All\_FPGA} \frac{\#ofLogicCells_{your\_FPGA_i}}{\#ofLogicCells_{XC2VP30-6C}} \times \frac{FreqCalibration_{yourFPGA_i}}{FreqCalibration_{XC2VP30-6C}}\right\}$$

and

$$f_{CPU-performance} = \max\left\{1, \sum_{All\_Hard\_CPU} \frac{TimeCalibration_{300MHz\_PPC\_in\_XUP}}{TimeCalibration_{your\_CPU_i}}\right\}$$.

Use the following constants in the above formulas.

$\#ofLogicCells_{XC2VP30-6C}$ =30,816 Logic Cells

$FreqCalibration_{XC2VP30-6C}$ =67.9 MHz (ISE 9.1, post place-and-route)

$TimeCalibration_{300MHz\_PPC\_in\_XUP}$ =0.1573 sec

For the purpose of normalization, we treat a Xilinx Logic Cell and an Altera Logic Element as equivalent (comprising 1 LUT, 1 flop, and carry logic). Use vendor-published Xilinx LC or Altera LE count to determine for each of your FPGAs the value of $\#ofLogicCells_{your\_FPGA_i}$.

Follow the instructions below to build the benchmarking hardware and software modules to determine the values of $FreqCalibration_{your\_FPGA_i}$ and $TimeCalibration_{your\_CPU_i}$ for each of the FPGAs and hard CPUs on your platform. (Soft CPUs are accounted for in the FPGA logic

resources.) The calibration modules and instructions are identical to the 2008 contest. The modules were developed by Patrick Schaumont (Virginia Tech) for the 2008 contest.

Appendix A.1 Hardware

The factor $FreqCalibration_{your\_FPGA_i}$ for your FPGA is determined by synthesizing the following module on your FPGA, and finding the reciprocal of the critical path of this module after place and route. You will need this number for each FPGA that your system uses.

```
module test(
    input [31:0] A,
    output reg [4:0] LZ,
    input clk
);

reg [31:0] LA;
reg [4:0] Z;
integer       i;

always @(posedge clk) begin
    LA<=A;
    LZ<=Z;
end

always @(*) begin
    Z=0;
    for(i=0;i<=31;i=i+1) begin
        if (LA[i]) begin
            Z=Z+1;
        end
    end
end

endmodule
```

Appendix A.2 Software

The factor $TimeCalibration_{your\_CPU_i}$ is found by compiling the following test program on your CPU and determine the execution time. While compiling and running this program, keep the following constraints in mind.

- Choose an accurate time measurement method. We prefer you use a hardware-based clock-cycle counting mechanism for maximal accuracy. The sample code below provides several examples.
- Compile your code to on-chip memories, and report the fastest possible execution time.
- Compile your code with gcc -O2 optimization.

```
//--------------------------------------------------------------------
// MEMOCODE Hardware/Software Codesign Contest 2009  (Same as 2008)
// CPU calibration routines.
//
// Instructions
// 1. Select one of the methods below for time measurement
//    (a) using a timer module (OPB Timer on EDK example given)
//    (b) using a CPU cycle counter (PPC and X86 examples given)
//    (c) least preferable - use software based on time.h
```

```
// 2. Execute the test sequence below and collect all output of
//    the program
// 3. Report the output of this program together along with the unit
//    of time
//--------------------------------------------------------------------

// uncomment only one of the following four macros

// #define USE_OPBTIMER_COUNTER
#define USE_PPC_COUNTER
///#define USE_X86_COUNTER
// #define USE_SW_COUNTER

//---------------- using an OPB timer on Xilinx EDK
#ifdef USE_OPBTIMER_COUNTER
#include "xparameters.h"
#include "xtmrctr.h"
XTmrCtr T;
void clockstart() {
  XTmrCtr_Initialize(&T, XPAR_OPB_TIMER_0_DEVICE_ID);
  XTmrCtr_Reset(&T,0);
  XTmrCtr_Start(&T,0);
}
void clockstop() {
  XTmrCtr_Stop(&T,0);
}
unsigned long clockread() {
  return XTmrCtr_GetValue(&T,0);
}
#endif

//---------------- using powerPC cycle counter
#ifdef USE_PPC_COUNTER
long long ts_get_ll() {
       unsigned int upper, upper2;
       unsigned int lower;
       unsigned long long ans;
       unsigned int x;
       // see PPC405 programmer's manual for explanation
       do {
               asm volatile ("mftbu %[arg]" : [arg] "=r" (upper) : );
               asm volatile ("mftb %[arg]" : [arg] "=r" (lower) : );
               asm volatile ("mftbu %[arg]" : [arg] "=r" (upper2): );
       } while (upper!=upper2);
       ans=upper;
       ans=ans<<32;
       ans|=lower;
       return ans;
}
long long t1, t2;
void clockreset() {}
void clockstart() {t1 = ts_get_ll();}
void clockstop()  {t2 = ts_get_ll();}
unsigned long clockread() {return (long) t2 - t1;}
#endif

//---------------- use on x86
```

```
#ifdef USE_X86_COUNTER
#include
#include
__inline__ uint64_t rdtsc() {
  uint32_t lo, hi;
  __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
  return (uint64_t)hi << 32 | lo;
}
uint64_t t1, t2;
void clockreset() {}
void clockstart() {t1 = rdtsc();}
void clockstop()  {t2 = rdtsc();}
unsigned long clockread() {return t2 - t1;}
#endif

//------ use for software-supported timekeeping (not recommended)
#ifdef USE_SW_COUNTER
#include
#include
void clockreset() {}
clock_t t1, t2;
void clockstart() {t1 = clock();}
void clockstop () {t2 = clock();}
unsigned long clockread() {return t2 - t1;}
#endif

//-----------------------------------------------------------------------

unsigned rnstate = 1;
unsigned accumulate() {
  unsigned a = 0;
  int i;
  for (i=0; i<(1 << 20); i++) {
    a += rnstate;
    rnstate = (rnstate>>1) ^ (-(signed int)(rnstate&1)&0xd0000001u);
  }
  return a;
}

int main() {
  int i;
  unsigned a;
  for (i=0; i<3; i++) {
    clockstart();
    a  = accumulate();
    clockstop();
    xil_printf("Round %3d Chk %8x Ticks Elapsed %d\n",
               i, a, clockread());
  }
  return 0;
}
```