CSCE 313: Embedded Systems

Performance Counters

Instructor: Jason D. Bakos
Performance Counters

- Performance counters are hardware counters that increment under certain conditions

- Significant because they allow cycle-level precision

- Allows for high resolution hardware instrumentation:
  - Events, i.e. cache misses, instruction commits, exceptions
  - Timing, use counter to count cycles to execute a segment of code

- In lab 3, we will use them to time sections of code
  - Begin counter immediately prior to executing a section of code
  - Execute section of code
  - Stop counter immediately after executing a section of code
Performance Counters in NIOS SOPC

• To use, add in SOPC Builder:
  – Performance Counter Unit
  – Name: “performance_counter_0”
  – Must specify number of counters
    • This number must be one greater than the number of counters you want to measure
  – Make sure you put it on sys_clk and (automatically) assign base addresses
Performance Counters in NIOS SOPC

- To use:
  - FIRST, delete timer_0 from your system design, then include file:
    #include <altera_avalon_performance_counter.h>

- To reset counters:
  
  PERF_RESET(PERFORMANCE_COUNTER_0_BASE);

- To start/stop measuring (you must stop before you can read the counters):
  
  PERF_START_MEASURING(PERFORMANCE_COUNTER_0_BASE);
  PERF_STOP_MEASURING(PERFORMANCE_COUNTER_0_BASE);
  (place at begin and end of main())

- To measure a section of code (accumulating count):
  
  PERF_BEGIN (PERFORMANCE_COUNTER_0_BASE, <counter num.--start at 1>);
  <code to time>
  PERF_END (PERFORMANCE_COUNTER_0_BASE, <counter num.>);

- The global counter, 0, measures from the START to STOP
Performance Counters in NIOS SOPC

- To read global count:
  \[\text{time} = \text{perf\_get\_total\_time}\ ((\text{void}*)\text{PERFORMANCE\_COUNTER\_0\_BASE});\]

- To read section count:
  \[\text{time} = \text{perf\_get\_section\_time}\ ((\text{void}*)\text{PERFORMANCE\_COUNTER\_0\_BASE}, \text{<counter num.}>);\]

- To read number of occurrences between start and stop:
  \[\text{occurences} = \text{perf\_get\_num\_starts}\ ((\text{void}*)\text{PERFORMANCE\_COUNTER\_0\_BASE}, \text{<counter num.}>);\]

- To convert clock cycles to time:
  \[\frac{\text{time}}{\text{ALT\_CPU\_FREQ}}\]
Fixed-Point

• Need a way to represent fractional numbers in binary

• \((N,M)\) Fixed-point representation
  – Assume a decimal point at some location in a value:
  – Example \((6,4)\)-fixed format:

    \[
    \begin{array}{cccc}
    1 & 0 & .1 & 1 & 0 & 1 \\
    \end{array}
    \]

    \[
    \begin{align*}
    &= 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4} \\
    &= 2 + 0.5 + 0.25 + 0 + 0.125 + 0.0625 \\
    &= 2.9375
    \end{align*}
    \]

• For signed, use two’s compliment

  – Range = \([-2^{N-1}/2^M, 2^{N-1}/2^M - 1/2^M]\)
  – For example above, \([-2^5/2^4, 2^5/2^4-1/2^4] \Rightarrow [-2,2-(1/16)]\)
Lab 3 Fixed-Point

• C doesn’t offer a dedicated fixed-point type

• We don’t have fixed-point sin and cos, so use floating-point for this, but convert to (32,4) fixed point by:
  – Multiply result by 16.0
  – Typecast to int

• Multiply:
  – cos/sin, which is (32,4) fixed-point value with
  – row/col, which is a (32,0) fixed-point number,
  – gives a (32,4) value, so keep this in mind!

• Adding two (32,4) values gives a (32,4) value
Lab 3 Fixed-Point

• Round when converting (32,4) fixed-point back to integer:
  - Check bit 3 to see if you should round up, i.e. val & 8
  - Shift fixed-point value 4 bits to right
  - If original bit 3==1, then add one to converted integer
Lab 3

- Split into two parts:
  - Change CPU to NIOS II/f
  - Enable hardware multiply
Lab 3

- Enable hardware floating point
Lab 3

- Use performance counters to find average time, in cycles, to transform each individual pixel when using different cache sizes:
  - 4K instruction cache, no data cache
  - 4K instruction cache, 4K data cache
  - 4K instruction cache, 16K data cache
Lab 3

- After testing floating point with various cache sizes, repeat the same tests with fixed-point

- Determine the performance impact of each

- Write a short report detailing your results and conclusions