CS220 Lab#5
Iterative Computation

Mainak Chaudhuri
Indian Institute of Technology Kanpur

Sketch

- Assignment#1: 32-bit Fibonacci number generator
  - Carries two marks
- Assignment#2: Integer part of square root of a 32-bit non-negative number
  - New learning: nested iterative computation
  - Carries three marks
- Assignment#3: Primality testing
  - New learning: nested iterative computation
  - Carries three marks

Assignment#1

- 32-bit Fibonacci number generator
  - Given n as input, design a hardware to generate the nth Fibonacci number
    - With 32 bits, the largest n is 47 (so n can have 6 bits)
  - Add F_k and F_{k-1} to generate F_{k+1} in one clock cycle using a 32-bit ripple-carry adder
    - $F_{k+1} = F_k + F_{k-1}$
    - $F_k <= F_{k+1}$
    - $F_{k-1} <= F_k$
  - Note that $F_1 = F_2 = 1$
  - The top module should take three inputs (n, *reset*, *clock*) and produce two outputs (nth Fibonacci number, *done*)

Assignment#1

- 32-bit Fibonacci number generator
  - Do behavioral simulation to test functional correctness
  - Create AXI4 IP, synthesize, implement, generate bit stream, export hardware as done in previous assignments
  - Show that your design works by writing a C program in Vitis IDE
    - The relevant C code segment is shown in the next slide
    - You can verify your output using the list in the following page: https://r-knott.surrey.ac.uk/fibonacci/fibtable.html

Assignment#1

- 32-bit Fibonacci number generator

```
xil_printf("\n\rList of Fibonacci numbers:\n\r");
unsigned n, reset, fib, done;
for (n=1; n<=47; n++) {
   reset=1;
   Xil_Out32(base_addr+1, reset);
   reset=0;
   Xil_Out32(base_addr, n);
   Xil_Out32(base_addr+1, reset);
   done=0;
   while(!done) done = Xil_In32(base_addr+3);
   fib = Xil_In32(base_addr+2);
   xil_printf("%u ", fib);
}
xil_printf("\n\r");
```

Assignment#2

- Integer part of square root
  - Given a 32-bit non-negative integer *n*, compute the integer part of its square root
  - Each iteration of the algorithm involves squaring an integer *m* and checking whether $m^2 \geq n$; if not, increment *m* by one and repeat the step
    - The integer part of square root of *n* is the largest *m* such that $m^2 \leq n$
  - Use the 32-bit multiplier developed in Lab4 to compute m^2
  - The top module takes three inputs (*n*, *reset*, *clock*) and produces two outputs (integer part of the square root of *n*, *done*)

Assignment#2

- Integer part of square root
  - Nested iterative nature of the computation arises from the following facts
    - A multiplication may take a certain number cycles depending on the multiplier
      - These cycles constitute the inner iterations
    - An outer iteration always starts with resetting the multiplier and inputting a new value of *m*
    - The inner iterations for a certain value of *m* conclude when the multiplier sets its *done* output
      - At this point m^2 should be compared against *n* and either the computation terminates or a new outer iteration starts with multiplier and multiplicand both set to $m+1$

Assignment#2

- Integer part of square root
  - Do behavioral simulation to test functional correctness
    - The default maximum simulation duration is 1000 time units
    - To extend it, right-click *SIMULATION* in the *Flow Navigator* menu on the left, then select *Simulation Settings*, then click the *Simulation* tab in the lower right panel, and increase *xsim.simulate.runtime* to a suitably large value
    - Click *Apply* and then click *OK*

Assignment#2

- Integer part of square root
  - Create AXI4 IP, synthesize, implement, generate bit stream, export hardware as done in previous assignments
  - Show that your design works by writing a C program in Vitis IDE
    - A sample C code segment is shown in the next slide

Assignment#2

- Integer part of square root

```
xil_printf("\n\r");
unsigned i, n, reset, root, done;
for (i=0; i<=32; i++) {
   reset=1;
   Xil_Out32(base_addr+1, reset);
   reset=0;
   n = (1ULL << i) – 1;
   Xil_Out32(base_addr, n);
   Xil_Out32(base_addr+1, reset);
   done=0;
   while(!done) done = Xil_In32(base_addr+3);
   root = Xil_In32(base_addr+2);
   xil_printf("%u: n=%u, square root=%u\n\r", i, n, root);
}
```

Assignment#3

- Primality testing
  - Given a 32-bit non-negative integer *n*, determine whether *n* is a prime
  - Step#1: Find the integer part of square root of *n*
  - Step#2: Divide *n* by each integer in the range 2 to integer part of square root of *n* until a case is found where the remainder of the division is zero
    - If no such case is found, *n* is prime
  - Use the square root module designed in Assignment#2 to do Step#1
  - Use the 32-bit divider developed in Lab4 to do each iteration of Step#2

Assignment#3

- Primality testing
  - The top module takes three inputs (*n*, *reset*, *clock*) and produces two outputs (*is_prime*, *done*) where both outputs are single-bit
  - Nested iterative nature of the computation arises from the fact that a division may take multiple cycles to complete
    - These cycles constitute the inner iterations
    - An outer iteration always starts with resetting the divider and inputting a new divisor
  - The number of outer iterations depends on the output of the square root module
    - The nested computation phase cannot start until the square root module sets its *done* output

Assignment#3

- Primality testing
  - Do behavioral simulation to test functional correctness
    - You may have to increase the default maximum simulation duration
  - Create AXI4 IP, synthesize, implement, generate bit stream, export hardware as done in previous assignments
  - Show that your design works by writing a C program in Vitis IDE to count prime numbers
    - A sample C code segment is shown in the next slide
    - Verify your output using the data in the following page: https://t5k.org/howmany.html#table

Assignment#3

- Primality testing

```
unsigned x=100000, n, reset, is_prime, done, count=0;
xil_printf(“\n\rList of primes up to %u:\n\r”, x);
for (n=0; n<=x; n++) {
   reset=1;
   Xil_Out32(base_addr+1, reset);
   reset=0;
   Xil_Out32(base_addr, n);
   Xil_Out32(base_addr+1, reset);
   done=0;
   while(!done) done = Xil_In32(base_addr+3);
   is_prime = Xil_In32(base_addr+2);
   if (is_prime) { count++; xil_printf(“%u ”, n); }
}
xil_printf(“\n\rTotal number of primes: %u\n\r”, count);
```