System-on-Chip (SoC) Design

ECE382M.20, Fall 2026

Homework #3

Assigned: October 15, 2026
Due: October 29, 2026

Instructions:

- Please submit your solutions via Canvas. Submissions should include a single PDF with the writeup and single Zip or Tar archive for source code.
- You may discuss the problems with your classmates but make sure to submit your own independent and individual solutions.

Problem 1: SystemC (50 points)

An installation of the SystemC class library is available for you to work with on the department's LRC Linux servers (see https://wikis.utexas.edu/display/eceit/ECE+Linux+Application+Servers). Alternatively, you can install SystemC yourself locally on your own machine (see http://www.accellera.org/downloads/standards/systemc). To work with SystemC on the LRC machines, first setup the SystemC environment by loading the corresponding module as follows:

```
% module load systemc/2.3.4
```

You can then access the SystemC installation via the '$SYSTEMC' environment variable.

a) The SystemC installation comes with a set of examples, available under $SYSTEMC/examples. Create a working directory and create a local copy of the included simple FIFO example:

```
% mkdir work
% cd work
% cp -r $SYSTEMC/examples/sysc/simple_fifo .
% cd simple_fifo
```

Edit the Makefile to point the include statements to the correct path and to provide the required compile and build settings. The Makefile should look like this:

```
include $(SYSTEMC)/examples/build-unix/Makefile.config

PROJECT := simple_fifo
SRCS    := $(wildcard *.cpp)
OBJS    := $(SRCS:.cpp=.o)

include $(SYSTEMC)/examples/build-unix/Makefile.rules
```

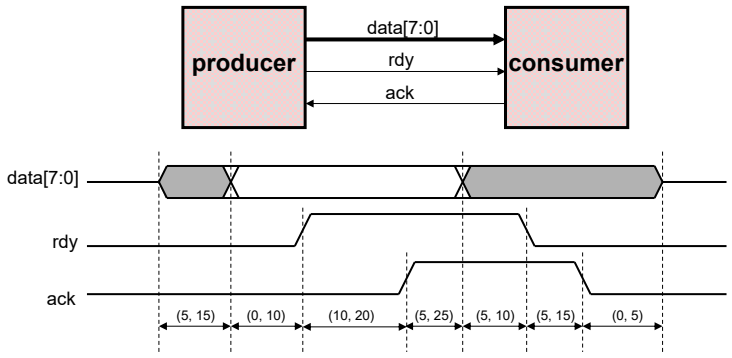
Finally, compile and run the example:

```
% make
% ./simple_fifo.x
```

Study the code and observe the program output. You can also use your favorite debugger (e.g., using `ddd` as a graphical frontend for `gdb`) to walk through the example. Briefly explain the program behavior and output, e.g. by drawing a trace of SystemC thread executions.

b) Modify the example to replace the custom FIFO channel with a corresponding `sc_fifo<char>` channel from the SystemC standard channel library. Compile and simulate the code to verify correctness. Vary the FIFO depth/size and observe the resulting behavior. Briefly explain any differences in behavior you see as a function of FIFO size/depth.

c) Now replace the `sc_fifo<char>` FIFO queue channel with `sc_signal<char>` and `sc_signal<bool>` channels that model the wires of a simple timed double handshake protocol connecting the *producer* and *consumer* modules, e.g. following this timing diagram (where tuples indicate minimum/maximum allowed times between events on the wires):



Insert code into the *consumer* that prints the current simulation time every time a character is received. Compile and simulate your design to verify its correctness and report on the simulation output.

d) Now go back to the original custom FIFO design from a). Can you modify this design by inserting time `wait()` statements into the custom queue channel to match the timing, behavior and output of the design from c)?

Problem 2: High-Level Synthesis (50 points)

Assume the following general matrix-matrix multiplication (GEMM) code written in Vitis High-Level-Synthesis (HLS) notation, where the *gemm_block* function multiplies 8x8 matrix blocks:

```
void gemm_block(unsigned int A[8][8], unsigned int B[8][8], unsigned int C[8][8]) {
    int i,j,k;
    loop1: for (i=0; i<8; i++){
        loop2: for (j=0; j<8; j++){
            loop3: for (k=0; k<8; k++){
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

(a) Draw the control data flow graph (CDFG) for this function.

(b) Assuming no pipelining or unrolling of loops, no resource constraints and a mapping of function inputs and outputs *A*, *B* and *C* into separate single-ported SRAMs each, what is the expected latency of *gemm_block()* if multiplications and additions have a non-pipelined delay of 2 and 1 clock cycles, respectively, and SRAM accesses take 1 cycle per read/write?

(c) Given the same constraints and interface definition as in (a), what is the expected latency if the innermost loop is fully unrolled, i.e. a `#pragma HLS unroll_factor=8` is applied to *loop3*? What limits the achievable speedup from unrolling? Without otherwise changing the loop or control structure (pipelining/unrolling/etc) itself, what is required to achieve the best latency of the unrolled version, and what is the ideal latency?

(d) What is the expected latency if *loop2* is pipelined in addition to fully unrolling *loop3* as in (c)? What limits the achievable initiation interval (II), and what is the II? What is required to achieve an II of 1, and what is the latency then?

(e) Assuming that a multiplication and an addition take 2 and 1 DSP slices to implement, respectively, how many DSPs do you expect to be utilized by a latency-optimal pipelined and unrolled implementation from (d)?

(f) Now assume that the optimized *gemm_block()* from (d) is part of a larger *gemm()* accelerator that repeatedly loads and stores 8x8 matrix blocks from/to external memory into/from local *A*, *B* and *C* block buffers in parallel with performing *gemm_block()* operations, i.e. where load/store and compute operations are overlapped. What is the minimal external bandwidth required to not be communication-bound?