System-on-a-Chip (SoC) Design

EE382V, Fall 2014

Homework #1

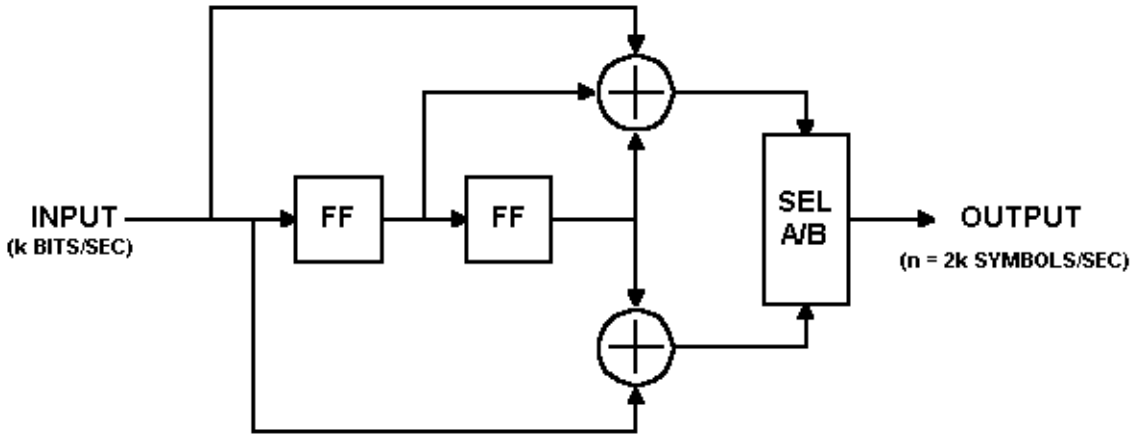
Assigned: September 4, 2014
Due: September 21, 2014

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: Viterbi Decoder (50 points)

The following circuit is used to convolutionally encode a bit stream:



Determine the most likely input stream given the following received stream:

```
[11, 10, 10, 00, 11, 11, 01, 11]
```

NOTE: Last two symbols (01, 11) are the flushing bits

Show all your work and include (1) the trellis diagram, (2) the accumulated error metric table, (3) the state history table, and (4) the survivor state table. Also, (5) indicate which accumulated error metrics and states you have selected during the decoding process and (6) point out received errors on the input stream.

The following document will help understand what you need to do:
http://www.ece.utexas.edu/~gerstl/ee382v_f14/soc/drm/Viterbi.pdf
This is an abbreviated version of the full Viterbi tutorial found here:
http://home.netcom.com/~chip.f/viterbi/algrthms.html

Problem 2: SystemC (50 points)

Make sure to use one of the newer 64-bit LRC machines (available machines are listed at http://www.ece.utexas.edu/it/remote-linux) and setup the SystemC environment as follows:

- [t]csh:
```
setenv SYSTEMC /usr/local/packages/systemc-2.3.1
setenv LD_LIBRARY_PATH $SYSTEMC/lib-linux
```
- [ba]sh:
```
export SYSTEMC=/usr/local/packages/systemc-2.3.1
export LD_LIBRARY_PATH=$SYSTEMC/lib-linux
```

You can then access the SystemC installation by referring to the '$SYSTEMC' variable.

a) Get the attached *Hello* example running: Unpack the archive, change into the `Hello-1` subdirectory, compile the example by running '`make`' and using your favorite debugger (`gdb` and `ddd` are good; for the latter you need to first run '`module load gnutools`' on the LRC machines), walk through the behavior of the example.

b) Create a for-loop in the process to output the "Hello" message 10 times in bursts with a random delay between messages evenly distributed from 50 to 90 ns.

c) Create two sub-modules, *Generate* and *Monitor*, connected by a channel *ch1*. Create two variants of the design where the sub-modules are connected by a `sc_fifo<string>` or a `sc_signal<char>`. You will need an output port and an input port on each sub-module. Instantiate them inside *Hello*. Move the loop into the *Generate* module, but have it write to the output port. Have the *Monitor* display values that show up on the input port.

Sources for *Hello* example are available at
http://www.ece.utexas.edu/~gerstl/ee382v_f14/hw/hw1.zip

Hello.h

```
#ifndef Hello_h
#define Hello_h
#include <systemc>
SC_MODULE(Hello) {
  SC_CTOR(Hello);
  void end_of_elaboration(void);
  void Hello_thread(void);
  ~Hello(void);
};
#endif
```

main.h

```
#include "Hello.h"
#include <iostream>
using namespace std;
using namespace sc_core;
int sc_main(void) {
  Hello top_i("top_i");
  cout << "Starting" << endl;
  sc_start();
  cout << "Exiting" << endl;
  return 0;
}
```

Hello.cp

```
#include "Hello.h"
#include <iostream>
using namespace std;
using namespace sc_core;
void Hello::Hello(sc_module_name nm)
: sc_module(nm) {
  cout << "Constructing "
       << name() << endl;
  SC_HAS_PROCESS(Hello);
  SC_THREAD(Hello_thread);
}
void Hello::end_of_elaboration(void) {
  cout << "End of elaboration" <<
endl;
}
void Hello::Hello_thread(void) {
  cout << "Hello World!" << endl;
}
Hello::~Hello(void) {
  cout << "Destroy " << name() <<
endl;
}
```