DLDCA Lab 1: Verilog
Week 2

August 6, 2025

Introduction

In this section you will be combining concepts from combinational and sequential programming.

Necessary tools

We shall be using `icarus-verilog` alias `iverilog` inorder to compile verilog HDL. To visualise waveforms we shall use `gtkwave`.

Before starting the lab, ensure that both tools are installed in your system. This can be verified by running

```
$ iverilog
iverilog: no source files.
Usage: iverilog [-EiRSuvV] [-B base] [-c cmdfile|-f cmdfile]
           [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir] [-L moduledir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
$ gtkwave --version
GTKWave Analyzer v3.3.116 (w)1999-2023 BSI

GTKWAVE | Use the -h, --help command line flags to display help.
```

```
WM Destroy
```

Structure of submission/templates

```
your-roll-no/
|- outlab/
    |- module.v (TODO)
    |- testbench.v
```

The folder structure of the expected submission for this lab is given above. We expect this folder to be compressed into a tarball with the naming convention of your-roll-no.tar.gz. The command given below can be used to do the same

```
$ tar -cvzf your-roll-no.tar.gz your-roll-no/
```

In this task, you will build a modular Verilog design to cycle through a specific sequence of RGB colors. The final design will consist of a counter that cycles through 8 states, and a mapping from those states to specific 3-bit RGB output values.

The target RGB sequence is given in the table below:

Cycle Number	Counter Value (3-bit)	RGB Output
0	000	111 (White)
1	001	010 (Blue)
2	010	011 (Cyan)
3	011	100 (Red)
4	100	110 (Purple)
5	101	101 (Yellow)
6	110	001 (Green)
7	111	000 (Black)

These are take home tasks that will help you finish the counter module. These are to be submitted by the end of this week.

Task 1: Next-State Logic Module

Design a purely combinational module called `getNextState` that takes the current 3-bit counter value and produces the next value in binary sequence.

```
module getNextState (
    input  [2:0] currentState,
    output [2:0] nextState
);
```

This module implements the next state logic for a 3-bit synchronous counter. Use toggle logic similar to T flip-flops, where each bit toggles based on the state of lower-order bits.

Task 2: 3-bit Counter Module

Using the above next-state logic, implement a sequential module called `threeBitCounter` that updates the counter on each rising edge of the clock. Include a synchronous `reset` input that resets the counter to 000.

```
module threeBitCounter (
    input clk,
    input reset,
    output reg [2:0] count
);
```

You must use the `getNextState` module to compute the next value of the counter, and assign it synchronously on the rising edge of the clock.

Task 3: Mapping Counter to RGB Lights

Design a combinational module called `counterToLights` that maps the binary counter value to a corresponding 3-bit RGB value according to the table above. Use only logic expressions. No arithmetic or control flow is allowed.

```
module counterToLights (
    input  [2:0] count,
    output [2:0] rgb
);
```

Use sum-of-products or direct logic expressions to define each RGB bit as a function of the 3-bit counter value.

For example on input count '000' the module should output rgb must be '111' according to the table.

Task 4: Top-Level Module

Create a top-level module called `rgbLighter` that connects all components. It should instantiate the counter and the color mapping module, and provide the final 3-bit RGB output.

```
module rgbLighter (
    input clk,
    input reset,
    output [2:0] rgb
);
```

You are expected to fill in the template code for all of the tasks and submit the same in the given format.