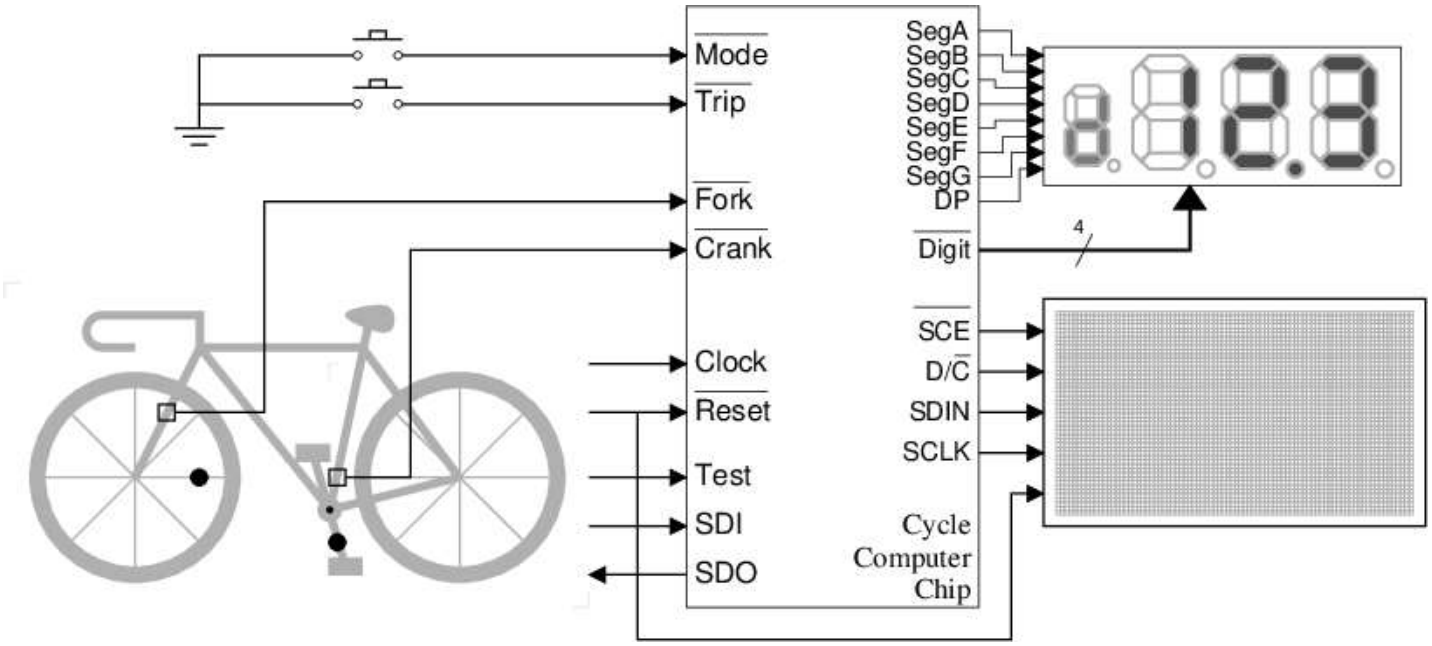
Simulation Environment

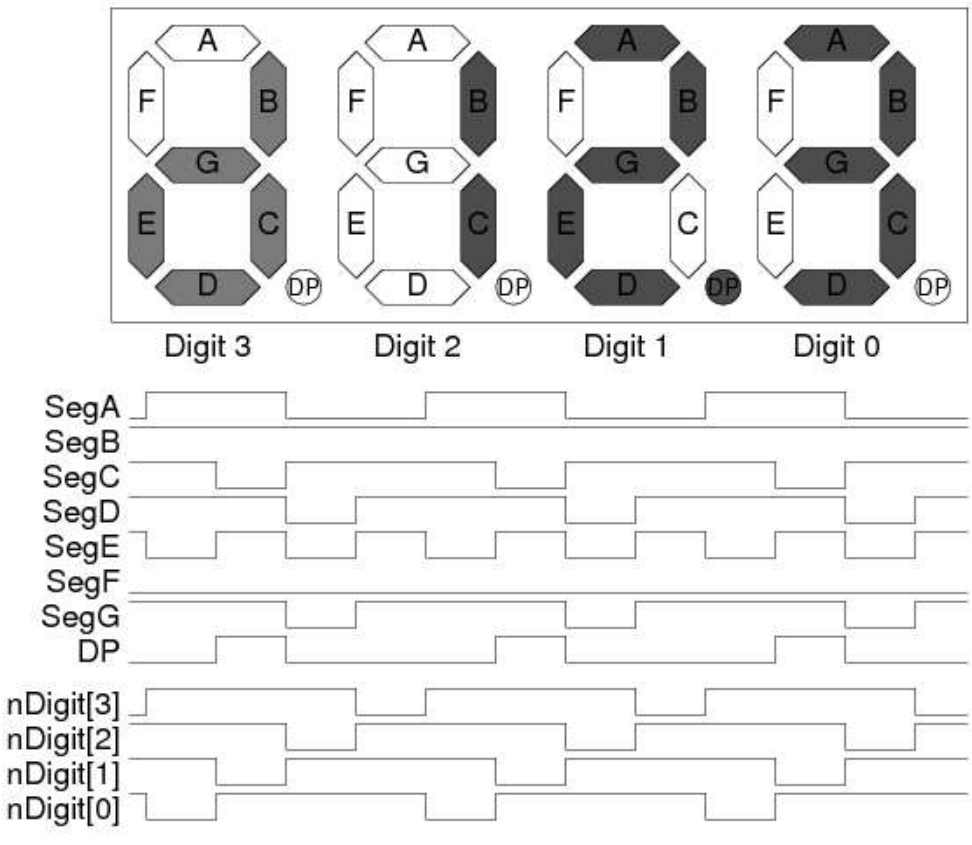
Overview

The simulation environment for the VLSI Design Project consists of a number of SystemVerilog files which model a basic system built around the cycle computer under test:



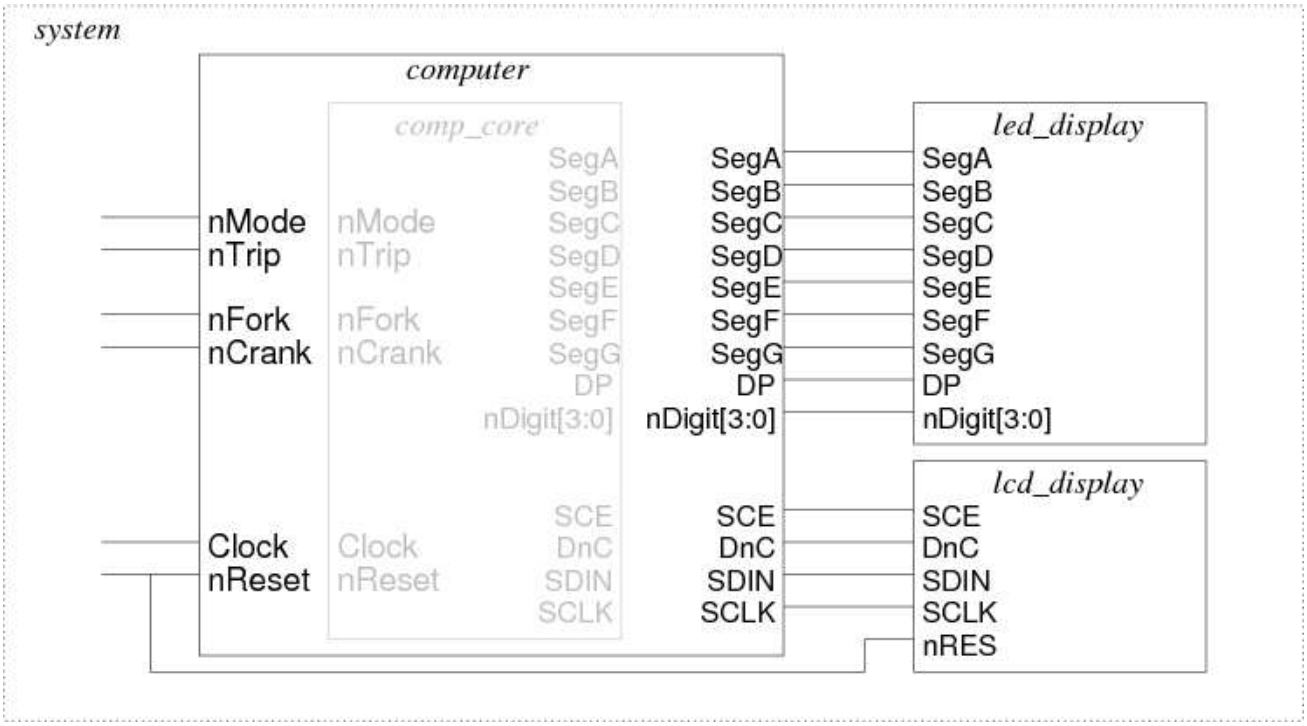
The system includes inputs from Hall effect sensors (Fork, Crank) and inputs from buttons (Mode, Trip) plus either one or two display outputs (LED, LCD).

The LED display is a seven-segment, common-cathode, multiplexed type. Thus a segment is lit if and only if the relevant segment line is taken high and the relevant digit line is taken low. The following figure shows the waveforms required to display "d12.3" (indicating a distance of 12.3km) on the seven segment display.



Model Hierarchy

The diagram below shows the hierarchy of the SystemVerilog models that make up the system:



Note that there are no DFT signals (Test, SDI, SDO) in the diagram. These signals may optionally be supported by the environment but they will not be used in the initial simulations and so are not shown here.

The hierarchy of the design is defined within the top-level module and testbench file which instances the cycle computer and display modules:

`system/system.sv`

Display Models

- **LED Seven Segment Display Model (led_display)**

  A minimal behavioural model (not synthesizable) of the Seven Segment Display module can be found in the SystemVerilog file:

  `system/led_display.sv`

- **LCD Model (lcd_display)**

  A minimal behavioural model (not synthesizable) of the LCD module can be found in the SystemVerilog file:

  `system/lcd_display.sv`

Pad Ring Behavioural Model

A behavioural model (not synthesizable) of the cycle computer pad ring can be found in the SystemVerilog file:

`system/computer.sv`

This model is merely a shell which simulates the pad ring and which instances the core cycle computer model (`comp_core`). This pad ring file is only used in initial behavioural simulations when no structural model of the pad ring exist.

Simulation

- Initialise your environment using the following command:

  ```
  init_chip_example
  ```

  this command will
  - create a working directory ~/design/chip/
  - copy the required files to that directory
    - system files will be put in the system sub-directory
    - example files will be put in the example sub-directory
    - a simulate executable shell script will also be copied.
  - open a new xterm window for the simulation set to the correct directory

- Using the following command, you can investigate the operation of the cycle computer:

  ```
  ./simulate  <model_dir>  <simulation_time>
  ```

- To run a simulation of a model in the **behavioural** directory for **100 seconds** you would type:

  ```
  ./simulate  behavioural  100s
  ```

Simulation Options

A number of options may be set to control the simulation:

- Clock Period

  `+define+clock_period=`*t*

  This option allows the period of the system clock to be customised if the design to be simulated will not work with the default 12.8kHz clock.

  You can use this or other options by adding them on to the end of the simulate command line.
  e.g. to run a 1000 second behavioural simulation with a 500kHz clock (period = 2µs) you might type:

  ```
  ./simulate behavioural  1000s  +define+clock_period=2us
  ```

  Note that setting a fast clock on a system-on-chip design can lead to excessively long simulation times especially for post-layout simulations. The simulation above with an apparently modest clock speed will require the simulation of 500 million clock cycles and will take approximately 40 times longer to simulate than a similar design which supports the default clock period. Keeping simulation times short encourages good testing practice and results in fewer mistakes in fabricated chips.

- Supported Modes

`+define+num_modes=`*n*

`+define+Mode`*n*`=`*mode_name*

The **num_modes** option tells the simulator how many modes can be reached by pressing the mode button. The **Mode***n* options describe the order and names of the reachable modes. For example, the following options define 2 modes named Distance and Duration:

```
+define+num_modes=2
+define+Mode0=Distance
+define+Mode1=Duration
```

- LCD Support

  `+define+include_lcd`

  This option indicates that the model being simulated supports an LCD display.

- Scan Path Support

  `+define+no_scan_signals`

  This option indicates that the model being simulated does not support scan path signals (Test, SDI, SDO).

- Stimulus

  `+define+special_stimulus`

  This option indicates that the simulation should use the advanced stimulus information found in a stimulus.sv file in the model directory.

- Monitoring

  `+define+special_monitor`

  This option indicates that the simulation should use the advanced monitoring information found in a monitor.sv file in the model directory.

In addition, an options.sv file can be used to set specify default options for the the model. The example options.sv file specifies that no scan path is supported in the example model. In this case appending `+define+no_scan_signals` to the simulate command will have no effect.

(note that an uncustomised template for this file can be found in the system directory: system/options.sv)

HDL Design

For your own HDL design, copy the example files to a new directory:

```
cp -r example behavioural
```

Then edit the files in the new directory to reflect your design. You can simulate your design using a command such as:

```
./simulate behavioural  20s  +define+special_stimulus  +define+special_monitor
```

You can also customise the display modules by placing copies in a **system2** directory:

```
mkdir system2
cp system/led_display.sv system2/
nedit system2/led_display.sv &
```

Note that you should not need to edit any files within the system directory. Making any changes in these files is likely to result in simulation failures during automated post-submission testing.

Iain McNally

28-2-2020