# Implementation

The design requirement has four inputs signals which correspond to gas, brake, clutch and override, and two outputs signals, denoted as gas\_control and brake\_control. The inputs will come from three keys and switch on the DE2 board. The outputs are represented by a red LED and a green LED, which represent the brake control and gas control, respectively; when the control is ON, the corresponding LED lights up. We observe that three given conditions should be satisfied: (a) if brake is ON, then brake\_control is ON and gas\_control is OFF; (b) if clutch is ON, then gas\_control is OFF; (c) if override is ON, then gas\_control is OFF and brake\_control is ON. The Boolean functions were then easy to derive after writing out the relationships.

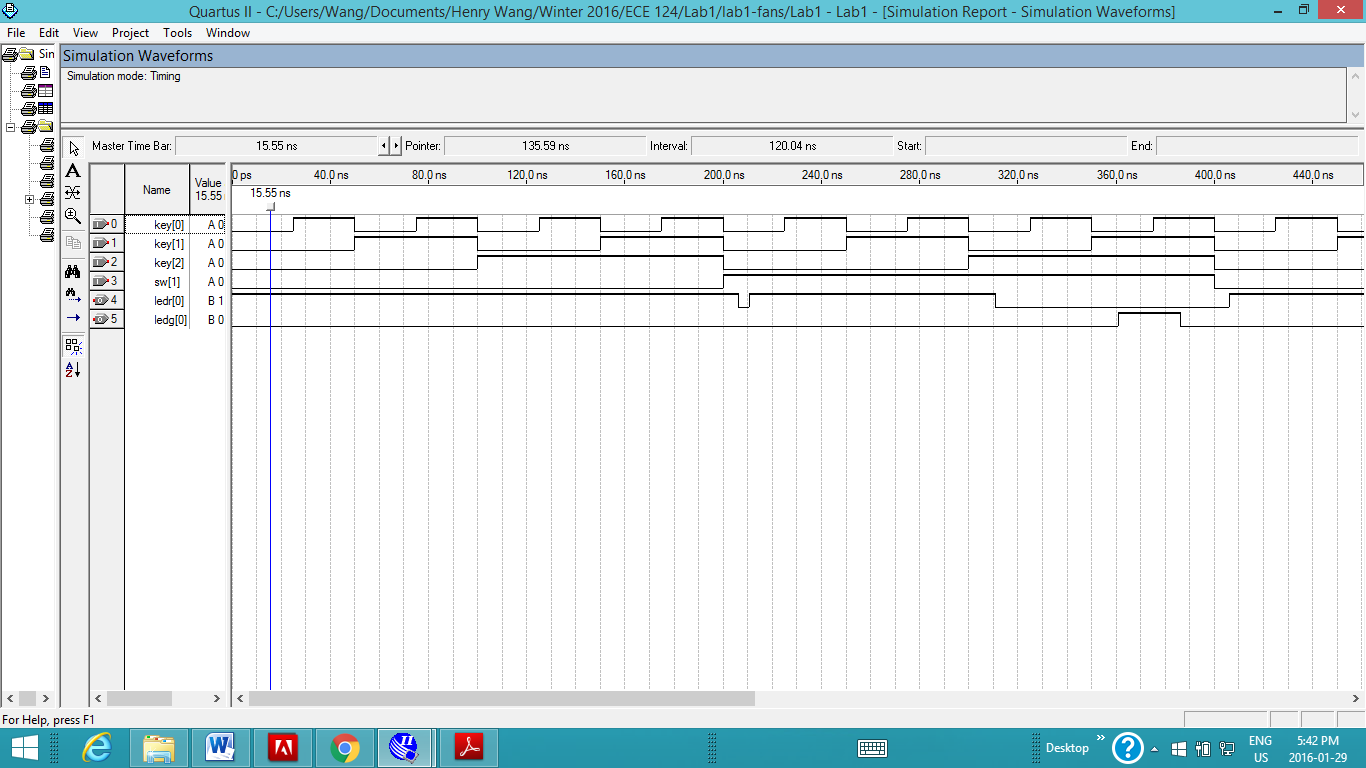
# Design decisions

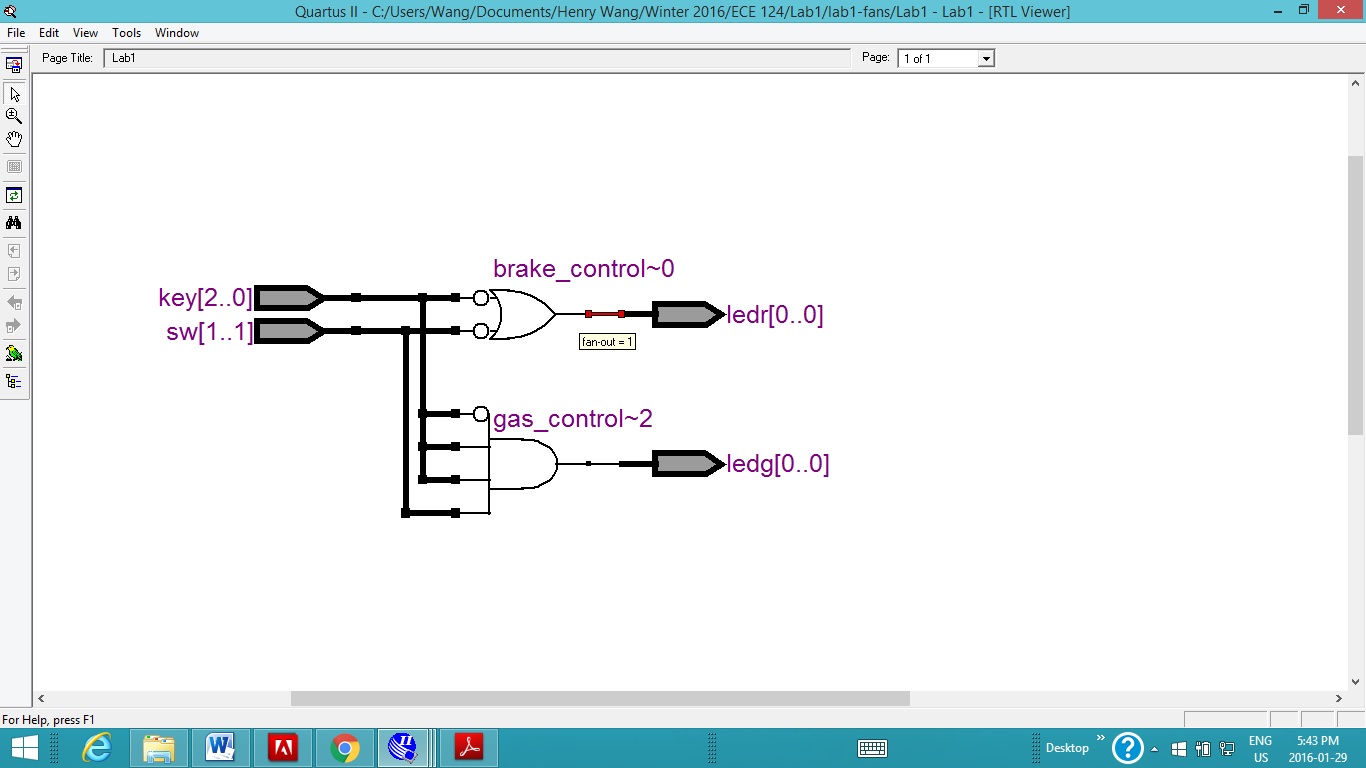
The first significant design decision was whether or not to invert the input signals for gas, brake and clutch. By nature, the DE2 boards have button keys that are inverted, such that the input is normally HIGH and becomes LOW when the button key is pressed. This is not conventional as normally people would expect pressing a button to enable a function. Therefore the input signals were inverted to create a more conventional user interface.

The other major design decision was whether or not to implement a multiplexer when computing the output signals. Following the examples provided, an attempt was made to implement the multiplexer, however because of the unfamiliarity with VHDL syntax it took more time to successfully implement. The end result was not as elegant as simply using two lines for the two Boolean functions implemented in the first place. In this case, the logic relating to the output isn’t complicated enough to merit to implementation of the multiplexer, therefore the decision was made to omit it.

# Problems and Bugs

As stated above the debugging process was lengthy because of the nuances of the VDHL syntax for using multiplexers. Although through trial and error, the proper syntax format was eventually discovered. There were no other issues with the software or hardware.





# Car Controller Code

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.ALL;**

**use** ieee**.**numeric\_std**.ALL;**

-- The above libaries lines must be included in every VHDL file, before EVERY ENTITY!

--

-- Main circuit Entity: connects all wires to the FPGA IO pins.

-- PORT mapping - declare all wire connections to INput or OUTput pins.

-- Note that all signal names here are fixed by the "DE2\_pins.csv" file which you must use for every lab

--

**entity** Lab1 **is** **port(**

key **:** **in** std\_logic\_vector**(**2 **downto** 0**);** -- 3 push buttons on the board - HIGH when not pressed

sw **:** **in** std\_logic\_vector**(**1 **downto** 1**);** -- use 1 out of 18 switches on the board LOW when down towards edge of board

ledr **:** **out** std\_logic\_vector**(**0 **downto** 0**);** -- 1 red LED, if lit, indicates brake control is on

ledg **:** **out** std\_logic\_vector**(**0 **downto** 0**)** -- 1 green LED, if lit, indicates gas control is on

**);**

**end** Lab1**;**

**architecture** CarSimulator **of** Lab1 **is**

--

-- Create the temporary variables reprensting our input signals

--

-- Signals are either a vector or not. A vector is a group of two or more signals

--

-- Note that there are two basic types and we nearly always use std\_logic:

-- UNSIGNED is a signal which can be used to perform math operations such as +, -, \*

-- std\_logic\_vector is a signal which can be used for logic operations such as OR, AND, NOT, XOR

--

**signal** gas**,** clutch**,** brake**,** override**:** std\_logic**;** -- four signals for inputs

**signal** gas\_control**,** brake\_control**:** std\_logic**;** -- two signals for LED outputs (one green and one red)

-- The function of CarSimulator entity is defined here

**begin**

-- Associate the input signals with the corresponding engine function

gas **<=** not key**(**0**);**

clutch **<=** not key**(**1**);**

brake **<=** not key**(**2**);**

override **<=** not sw**(**1**);**

-- The outputs of gas\_control and brake\_control are defined with the following boolean functions

gas\_control **<=** **(**not override**)** and **(**not brake**)** and **(**not clutch**)** and gas**;**

brake\_control **<=** override or brake**;**

-- assign intermediate signals to output ports

ledg**(**0**)** **<=** gas\_control**;**

ledr**(**0**)** **<=** brake\_control**;**

**end** CarSimulator**;**

# Seven Segments Code

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

--

-- 7-segment display driver. It displays a 4-bit number on a 7-segment LED display

-- This is created as an entity so that it can be reused many times easily

-- Apparently entities can contain other entities as components

-- General declaration of an entity has the OP port declarations and

-- the architecture which describes the behaviour of the entity (eg. outputs)

**entity** SevenSegment **is** **port** **(**

dataIn **:** **in** std\_logic\_vector**(**3 **downto** 0**);** -- The 4 bit data to be displayed

blanking **:** **in** std\_logic**;** -- This bit turns off all segments

segmentsOut **:** **out** std\_logic\_vector**(**6 **downto** 0**)** -- 7-bit outputs to a 7-segment

**);**

**end** SevenSegment**;**

**architecture** Behavioral **of** SevenSegment **is**

--

-- The following statements convert a 4-bit input, called dataIn to a pattern of 7 bits

-- The segment turns on when it is '0' otherwise '1'

-- The blanking input is added to turns off the all segments

--

**begin**

-- Assigning values to a logic vector requires that the left hand side be in double quotes

-- Where as a variable of type std\_logic requires the left hand side to be in single quotes

-- the blanking bit is concatenated with the input data, but it's always 0

**with** blanking **&** dataIn **select** -- gfedcba b3210 -- D7S

segmentsOut**(**6 **downto** 0**)** **<=** "1000000" **when** "00000"**,** -- [0]

"1111001" **when** "00001"**,** -- [1]

"0100100" **when** "00010"**,** -- [2] +---- a ----+

"0110000" **when** "00011"**,** -- [3] | |

"0011001" **when** "00100"**,** -- [4] | |

"0010010" **when** "00101"**,** -- [5] f b

"0000010" **when** "00110"**,** -- [6] | |

"1111000" **when** "00111"**,** -- [7] | |

"0000000" **when** "01000"**,** -- [8] +---- g ----+

"0010000" **when** "01001"**,** -- [9] | |

"0001000" **when** "01010"**,** -- [A] | |

"0000011" **when** "01011"**,** -- [b] e c

"1000110" **when** "01100"**,** -- [c] | |

"0100001" **when** "01101"**,** -- [d] | |

"0000110" **when** "01110"**,** -- [E] +---- d ----+

"0001110" **when** "01111"**,** -- [F]

"1111111" **when** **others;** -- [ ]

**end** **architecture** Behavioral**;**

--------------------------------------------------------------------------------

-- Main entity

--------------------------------------------------------------------------------

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**entity** Seven\_Segment **is** **port** **(**

sw **:** **in** std\_logic\_vector**(**17 **downto** 0**);** -- 18 dip switches

ledr **:** **out** std\_logic\_vector**(**17 **downto** 0**);** -- 18 red LEDs

hex0 **:** **out** std\_logic\_vector**(** 6 **downto** 0**)** -- 7-segment displays

**);**

**end** **entity** Seven\_Segment**;**

**architecture** SimpleCircuit **of** Seven\_Segment **is**

-- Component declaration of the SevenSegment entity is neccesary

-- in order for it to be instantiated

-- The ports in the component declaration have to match the entity port declaration

**component** SevenSegment **port** **(**

dataIn **:** **in** std\_logic\_vector**(**3 **downto** 0**);**

blanking **:** **in** std\_logic**;**

segmentsOut **:** **out** std\_logic\_vector**(**6 **downto** 0**)**

**);**

**end** **component;**

**begin**

ledr **<=** sw**;** -- the LEDs will be lit if the switch below it is at the position closer to it

SS\_0**:** SevenSegment **port** **map(**sw**(**3 **downto** 0**),** '0'**,** hex0 **);** -- Four bits of SW(3 downto 0) displayed on HEX0, blanking is disabled

-- SS\_0 is just a label for the instantiation of a SevenSegment component

-- The port map keywords mean that the logic vectors input are mapped to the corresponding vectors declared in the entity

**end** SimpleCircuit**;**