UCD School of Electrical and Electronic Engineering

EEEN30190 Digital System Design

Calculator Design Report

Name: Oluwademilade Oke **Student Number: 16363763**

Working with: Daniel McManus **Team: 10**

I certify that ALL of the following are true:

1. I have read the *UCD Plagiarism Policy* and the *College of Engineering and Architecture Plagiarism Protocol*. (These documents are available on Blackboard, under Assessment.)
2. I understand fully the definition of plagiarism and the consequences of plagiarism as discussed in the documents above.
3. I recognise that any work that has been plagiarised (in whole or in part) may be subject to penalties, as outlined in the documents above.
4. I have not previously submitted this work, or any version of it, for assessment in any other module in this University, or any other institution.
5. I have not plagiarised any part of this report. The work described was done by the team, but this report is all my own original work, except where otherwise acknowledged in the report.

Signed: _____Demi Oke_____ **Date:** 3 December 2018

Introduction

The aim of this assignment was to design hardware that would function as a simple calculator. This was done by determining the calculator functionality and developing appropriate RTL diagrams to implement it. The design was then described using Verilog. Verification of the design was also written in Verilog. The design was then implemented on the Digilent Nexys-4 circuit board. This assignment was carried out over 4 laboratory sessions.

Calculator Functionality

We designed a simple algebraic calculator where the calculation is entered in the order that one would write it. The calculator remembers only two numbers; the last number entered and the result of the previous calculation.

In order to simplify the design, the calculator uses integers only, working in hexadecimal. We used a fixed 4-digit display to display either the number that is being entered or the result of the previous calculation.

Our design displayed negative numbers using 2's complement. We designed an LED to turn on when a negative number was in the display.

The calculator implements 5 arithmetic operations: addition $(+)$, subtraction $(-)$, multiplication $(\times)$, square (n^2) and change sign $(+/-)$.

Note that the last two operations; square and change sign are *unary operations* and so involve only one operand compared to the *binary operations*; addition, subtraction and multiplication, which involve two. The change sign operation also allows negative operands to be entered into the calculator.

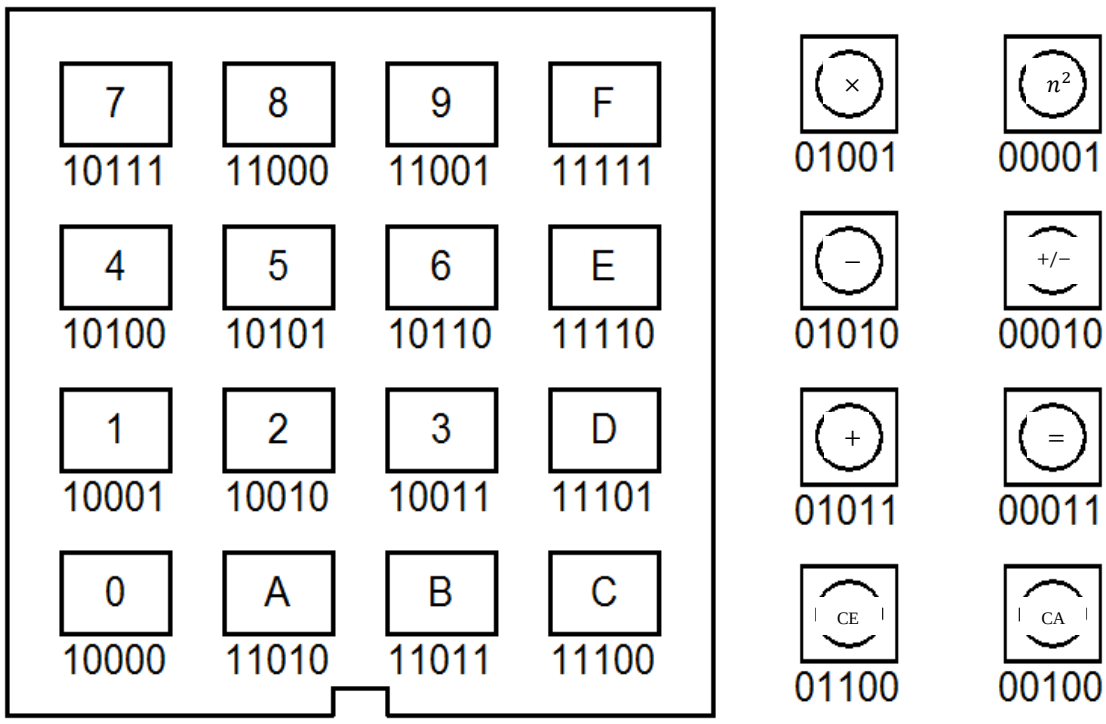
The result of calculations using binary operations can be displayed by pressing the equals $(=)$ key. However, there is no need to press the equals key to display the result of the unary operations as it is shown on the display immediately.

Two clear keys were also designed; clear entry (CE) and clear all (CA). The clear entry key is used to clear the number on the display while the clear all key is used to clear both the number on the display, the result of the previous calculation and all other codes stored in the calculator.

Either of the clear keys is to be pressed between each calculation before entering a new calculation. However, when doing continuous operations, a clear key need not be pressed, rather the calculation can be entered as follows: (2 + 3 = + 4 =). Noting that an equals key needs to be pressed for each operation.

Also, every time a binary operation key is pressed the calculator display clears to allow the second operand to be entered.

Key Assignment



Multiplication ($\times$):	01001	Square (n^2):	00001
Subtraction ($-$):	01010	Change Sign ($+/-$):	00010
Addition ($+$):	01011	Equals ($=$):	00011
Clear Entry (CE):	01100	Clear All (CA):	00100

The key assignment for multiplication, subtraction and addition were chosen so that the first three bits of their codes were the same i.e. 010 and the last two were different. This simplified hardware description slightly.

The LED's used to indicate the occurrence of an overflow and a negative number are LED0 and LED1 respectively.

Calculator Hardware

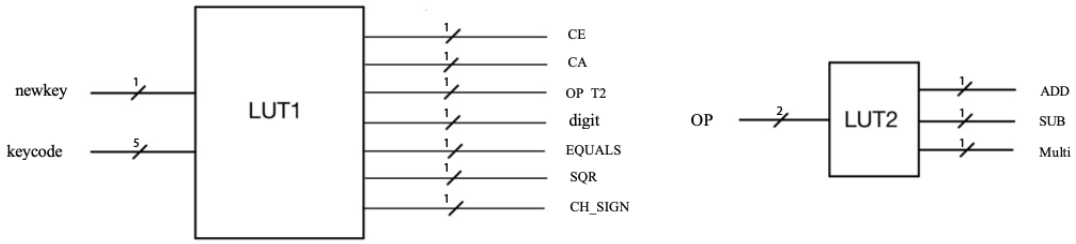
Designing an algebraic calculator required the use of the following registers:

- X: This register is used to store the number being entered. It is also connected to the display interface so whatever is stored here is displayed on the 4-digit display.
- Y: This register is used to store the first operand when a binary (requiring 2 operands) operation key is pressed.
- OP: This register stores binary operations pressed.

Other signals were also created to design the calculator and our explained in detail using RTL Diagrams below.

RTL Diagrams

Control Signals

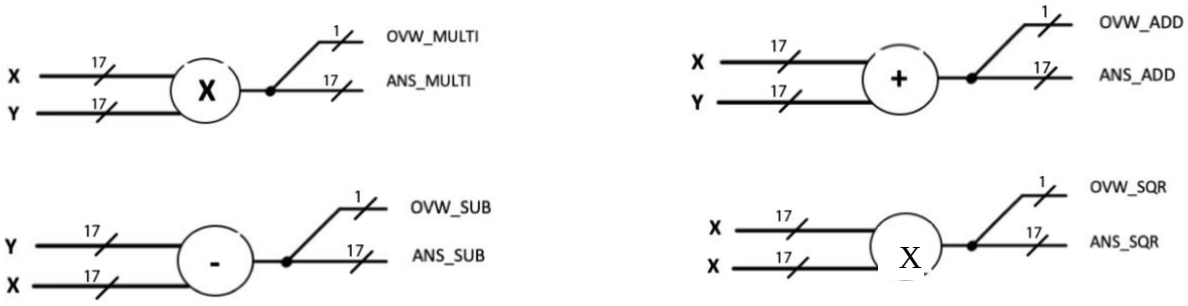


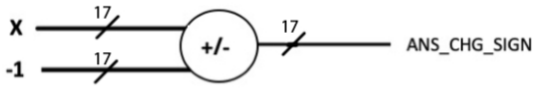
Control signals were used to enable different parts of the calculator with respect to the inputs obtained from the keypad interface and the OP register. The LUT's shown above are described in the submitted Verilog.

The outputs of the LUT1 indicate what type of key was obtained from the keypad interface i.e. OP_T2, CA, CE, digit, EQUALS, SQR or CH_SIGN key only one of which can be 1 while the others are 0. Note that OP_T2 indicates that a binary operator was obtained from the keypad.

The outputs of LUT2 indicate what type of binary operation is stored in the OP register i.e. ADD, SUB, or MULTI.

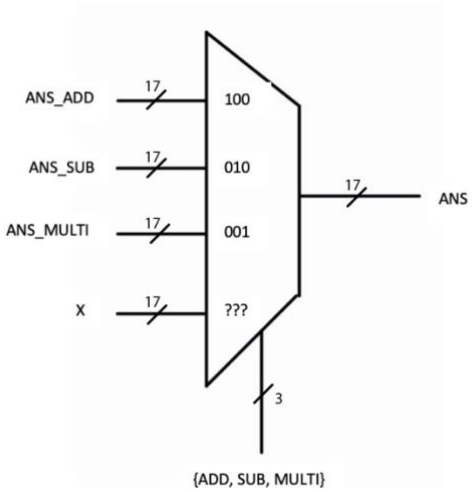
ANS Signals





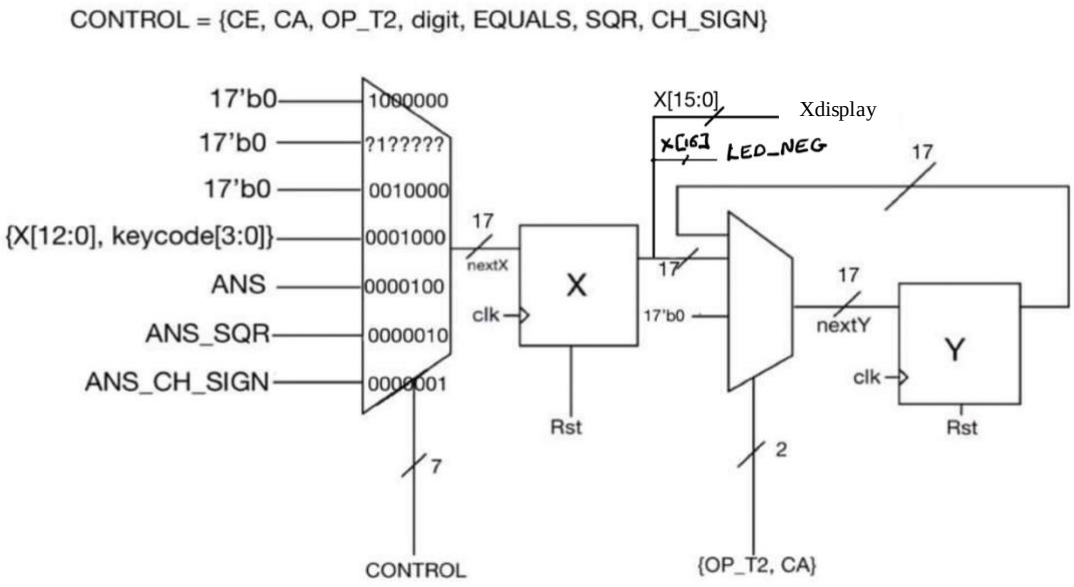
This diagram shows the results of the 5 operations used by the calculator. Note that for the addition, subtraction, multiplication and square calculations the MSB is removed to give the respective answer. In this case the MSB is used to detect overflow. However, this was not done for the result of the change sign operation as an overflow would never occur.

ANS Multiplexer



This multiplexer was used to obtain the correct answer to display respective to which binary operation was entered. It uses the ADD, SUB and MULTI control signals to determine this as shown in the diagram. Once a binary operation key is pressed at least one of these control signals should 1 and the others 0. This allows for the answer of continuous operations to be displayed.

X and Y Registers



The input of the X register (nextX) is determined by a multiplexer whose selector is a 7-bit control signal composed of the outputs of LUT1 concatenated together as shown on the diagram.

$$CONTROL = \{CE, CA, OP\_T2\ , digit, EQUALS, SQR, CH\_SIGN\}$$

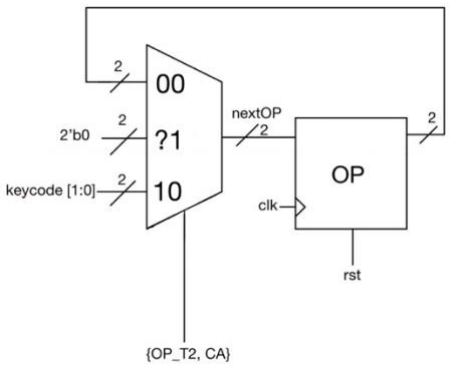
As these control signals are obtained from a single keycode, only one of them should be on at a time.

- When CE or CA is 1, the X register is loaded with 17'b0, as in both cases the X register is to be cleared.
- When OP_T2 is 1 (a binary operation key was pressed), the X register is also cleared in the same way. This is to allow for the next number to be entered into the X register.
- When digit is 1, the digit obtained from the keypad is loaded into the X register. Note that each new digit entered is pushing the previous digit to the left, increasing their value. This is done by concatenating the 13 leftmost bits with the digit obtained from the keycode.
- When EQUALS is 1, result of the ANS multiplexer is loaded into The X register so the answer of the calculation can be seen on the display.
- When SQR or CH_SIGN is 1, ANS_SQR or ANS_CH_SIGN is loaded into the X register respectively.

The MSB of X is used to indicate a negative number by assigning LED_neg = X[16]. The last 16 bits of X are sent to the 4-digit display interface.

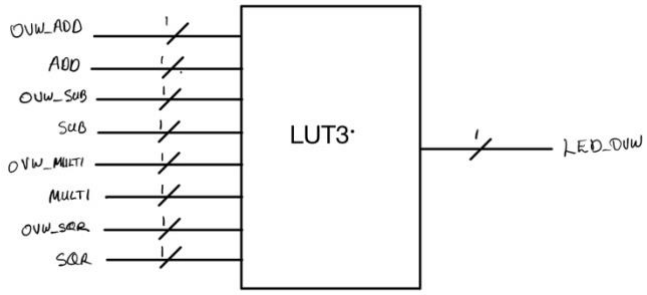
The input of the Y register (nextY) is determined by a multiplexer controlled by the control signals: OP_T2 and CA concatenated together. This is because X is only loaded into the Y register if a binary operator key is pressed (OP_T2 = 1). If a clear all key is pressed (CA = 1), the Y register needs to be cleared. If OP_T2 = 0 AND CA = 0, Y remains the same.

OP Register



The input to the OP register is determined in the same manner as that of the Y register as seen on the diagram. However, in this case, when OP_T2 = 1, the binary operation is obtained from the keycode. Note that the diagram states that the operation can be determined by the last 2 bits of the keycode signal. This is due to the strategic key assignment of the binary operation keys. They were chosen so that their last 2 digits were different.

LED_OVF



The signal that indicates overflow (LED_OVW) obtained using a LUT which was described in Verilog.

Verilog

The Verilog used to describe the calculator logic was uploaded to blackboard. However, extracts describing the LUT's used in the RTL diagrams are included below.

Local parameters were used in the Verilog to give names to the constants making the code easier to understand.

```
//MAPPING KEYS
    localparam[4:0] KEY_CE = 5'b01100,
                    KEY_CA = 5'b00100,
                    KEY_MULTI = 5'b01001,
                    KEY_SUB = 5'b01010,
                    KEY_ADD = 5'b01011,
                    KEY_SQR = 5'b00001,
                    KEY_CH_SIGN = 5'b00010,
                    KEY_EQUALS = 5'b00011;
```

The first 3 bits of the binary operation buttons; KEY_MULTI, KEY_SUB and KEY_ADD are 010 and their last 2 digits are different from each other. This is used to simplify the OP_T2 signal described in LUT1 and the signals from LUT2.

All digits coming from the keypad interface begin with 1 as shown on the key assignment diagram in the Calculator Functionality section. This also simplifies the description of the digit signal in LUT1.

LUT 1 described the control signals as shown:

```
  wire OP_T2 = (newkey && (keycode[4:2] == 3'b010) );
    wire CA = (newkey && (keycode == KEY_CA) );
    wire CE = (newkey && (keycode == KEY_CE) );
    wire digit = (newkey && (keycode[4] == 1'b1) );
    wire EQUALS = (newkey && (keycode == KEY_EQUALS) );
    wire SQR = (newkey && (keycode == KEY_SQR) );
    wire CH_SIGN = (newkey && (keycode == KEY_CH_SIGN) );
```

These control signals require the newkey signal from the keypad interface to be high in order to be activated.

LUT2 described the operation control signals as shown:

```
  wire ADD = (OP == KEY_ADD [1:0]);
    wire SUB = (OP == KEY_SUB [1:0]);
    wire MULTI = (OP == KEY_MULTI [1:0]);
```

LUT3 described the operation control signals as shown:

```
assign LED_OVW = ((OVW_ADD && ADD)||(OVW_SUB && SUB)||(OVW_MULTI && MULTI)||(OVW_SQR && SQR));
```

Verification

Verification Plan

Our verification plan consisted of verifying that the correct behaviour occurred when certain inputs were entered. It consisted of the following:

Behaviour to be Verified	Inputs Required	Expected Outputs
1. Entering a multi-digit number	More than 2 consecutive digits from the keypad. E.g.: 1, 2, 3, 4	Each new digit should push the previous digits to the left, increasing their value on the display. E.g.: 1234
2. Use of binary operations	A number followed by an operation then another number. The equals key should be pressed to display answer. E.g.: 5 + 8 =	The correct answer should be displayed in hexadecimal. E.g.: D
3. Use of continuous binary operations.	This is an extension of verifying the use of binary operations, so inputs should be entered as previously described. Then after the equals key is pressed, another operation key followed by a number and an equals key is pressed. E.g.: 5 + 8 = -3 =	The correct answer should be displayed after each time the equals key is pressed. E.g.: d then a.
4. Use of unary operations	A number followed by the desired unary operation. Note that for the change sign operation the order of the input values does not matter, however this is not the case for the square operation. E.g.: 12 [+/-] 12 [n^2]	The answer of the unary number should be displayed without needing the equals key to be pressed. E.g.: negative LED lights, 12 144 *Negative LED lights to indicate negative number
5. **Displaying negative numbers	An arithmetic sequence that causes a negative number: E.g.: 12 [+/-]	Magnitude of the answer should be displayed, and LED should light indicate negative number: E.g.: negative LED lights, 12
6. Use of clear keys	After entering an arithmetic sequence either one of the clear keys should be pressed. E.g.: 5, CE 5 + 8 = -3 =, CA	If the clear entry key is pressed, the display should go to zero. If the clear all key is pressed, the display should also go to zero and all registers in the calculator should clear.
7. **Overflow	An arithmetic sequence that results in an overflow of the 17-bit X register. E.g.: ffff + ffff =	The display should show the last 2 bytes of the answer in hexadecimal form and the overflow warning LED should light up. E.g.: LED lights, fffe on display

Note that there was no exact specification for the use of unary operators. We determined that the expected outputs described in the verification plan were reasonable behaviours.

***Regarding the displaying of negative numbers and overflows, the design of the LED's did not work as expected (as shown in verification) and we were unable to produce the magnitude of the negative numbers. Thus, decided to just display negative numbers in 2's complement with the LED.*

Note that the specific timing requirements for the verification were described in the Verilog testbench file.

Testbench

The Verilog testbench file used to verify the behaviour of the calculator was uploaded to Blackboard. It made use of two provided tasks that were used in order to make the file more legible.

- **PRESS()**: Simulates an input from the keypad. As the keypad hardware has outputs that change just after the rising edge of the clock, this task changes the inputs to the calculator just after the rising edge of the clock. It also generates a pulse on newkey for one clock cycle. The key obtained from the keycode is held for 2 clock cycles in an attempt to shorten the timing diagrams. This task also writes which key was pressed to a log file.
- **Check()**: Checks that the output of the calculator matches the expected value which is given as an argument. The output is checked just at the falling edge of the clock when it should be stable. The outputs and any errors are written onto a log file.

Note that all inputs were entered as hexadecimals, including the operation keys which were given the following local parameters for legibility of code:

```
// Define names for the non-digit keys
    // Note the leftmost bit of the keycode is inverted - see later
    localparam [4:0] PLUS = 5'h1B,
                     MINUS = 5'h1A,
                     MULTI = 5'h19,
                     SQR = 5'h11,
                     CH_SIGN = 5'h12,
                     EQUALS = 5'h13,
                     CA = 5'h14,
                     CE = 5'h1C;
```

The testbench file verified multiple behaviours in grouped arithmetic sequences as described below.

- <u>Testing multi-digit input:</u>
  Entering the number 1234 and then clearing it using clear entry.

- <u>Testing addition, continuous addition, change sign and clear all:</u>
  - Entering the number 12.
  - Changing the sign twice.
  - Adding the number to 5 to give 17, which was added to 1111 to give 1128.
  - Then changing the sign to give eed8 and finally clearing all to give 0.

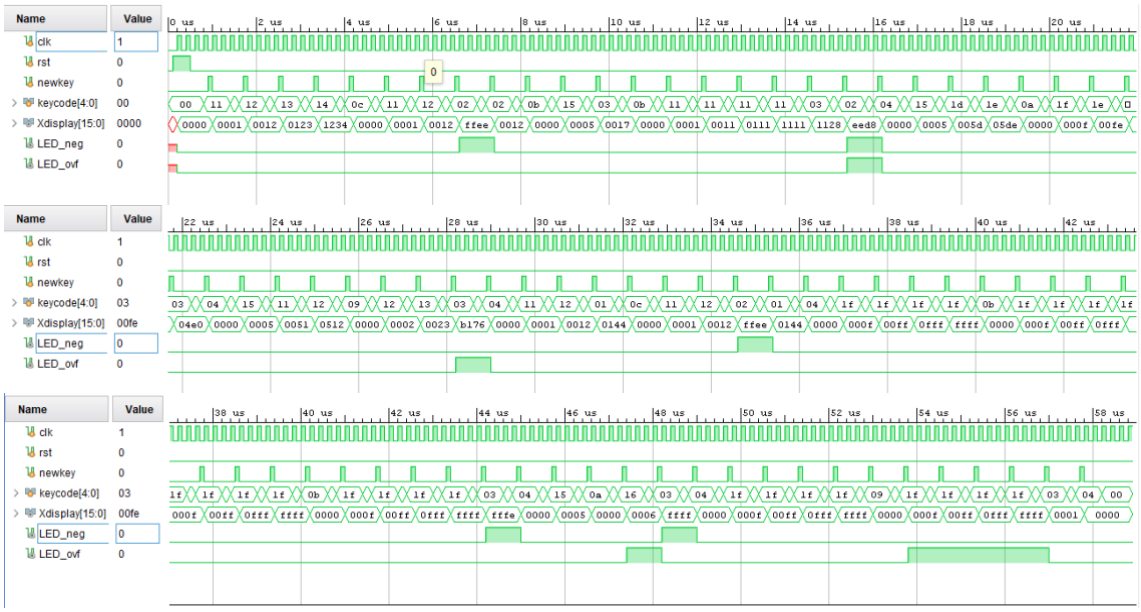
- <u>Testing other operations:</u>

- Subtracting fe from 5de to give 4e0 and clearing all to give 0.
- Multiplying 512 by 23 to give b176 and clearing all to give 0.
- Squaring 12 and -12 (12 and change sign) to give 144 in both cases and clearing each to give 0.

- <u>Testing overflow:</u>
  Adding ffff to ffff to give fffe then clearing all to give 0.
  Subtracting 6 from 5 to give ffff then clearing all to give 0.
  Multiplying ffff by ffff to give 0001 then clearing all to give 0.

The keys pressed, and any errors in the outputs were logged to a separate file. Note that where overflows and negative numbers occurred, they were determined by looking at the timing diagram. The log file contained no errors.

Timing Diagrams



The timing diagrams obtained as a result of the testbench shows that the calculator is displaying the expected values.

The LED_neg signal was high when a negative number was displayed. i.e. when changing the sign 12 to give ffee and changing the sign of 1128 to give eed8. It was also high when 5 was subtracted from 6 to give ffff.

However, it failed later in the verification as it was high for the addition of ffff and ffff to give fffe which should not have been negative.

The LED_ovf signal was incorrect at all times as it was high when 512 was multiplied by 23 to give b176 which should not have caused an over flow. For the last calculation (ffff * ffff) the signal was high too early.

The correct answers to the calculations are displayed as stated in the testbench.
An attempt to rectify these errors was made and is discussed in the conclusion.

Testing on Hardware

The same tests done in the testbench were also done on the implemented hardware. The same results were obtained.

Synthesis and Implementation

Warning in Synthesis

When the design was synthesised, four warnings were obtained:

```
[Synth 8-3917] design calculator_top has port digit[7] driven by constant 1
[Synth 8-3917] design calculator_top has port digit[6] driven by constant 1
[Synth 8-3917] design calculator_top has port digit[5] driven by constant 1
[Synth 8-3917] design calculator_top has port digit[4] driven by constant 1
```

These warnings were to be expected as they the calculator only used 4 out of the eight displays available.

Resources used on the FPGA

Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	224	0	63400	0.35
LUT as Logic	224	0	63400	0.35
LUT as Memory	0	0	19000	0.00
Slice Registers	86	0	126800	0.07
Register as Flip Flop	86	0	126800	0.07
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

All LUT's used were for combinatorial logic and no memory LUT's were used. Out of the 86 registers used, it was found that 72 were clock enabled asynchronous reset and 14 were clock enabled synchronous reset. This made sense considering we implemented the registers to be asynchronous reset. The synchronous registers were probably used to implement clock hardware.

DSP's

Site Type	Used	Fixed	Available	Util%
DSPs	2	0	240	0.83
DSP48E1 only	2			

Digital Signal Processing blocks are used as arithmetic blocks in the design

IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	30	0	210	14.29
Bonded IPADs	0	0	2	0.00
PHY_CONTROL	0	0	6	0.00
PHASER_REF	0	0	6	0.00
OUT_FIFO	0	0	24	0.00
IN_FIFO	0	0	24	0.00
IDELAYCTRL	0	0	6	0.00
IBUFDS	0	0	202	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	24	0.00
PHASER_IN/PHASER_IN_PHY	0	0	24	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	300	0.00
ILOGIC	0	0	210	0.00
OLOGIC	0	0	210	0.00

Only 30 out of the 210 Bonded IOB's are used meaning that our design can be implemented on the selected FPGA package.

Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	24	0.00
MMCME2_ADV	1	0	6	16.67
PLLE2_ADV	0	0	6	0.00
BUFMRCE	0	0	12	0.00
BUFHCE	0	0	96	0.00
BUFR	0	0	24	0.00

The BUFGCTRL clock buffer used, drives the routing and distribution resources across the entire device.

The MMCME2_ADV supports clock network deskew, frequency synthesis and jitter reduction for the clock used in the design.

In total, our design used up a small percentage of the FPGA hardware allowing the FPGA to easily implement it

Results of Timing Analysis

Setup:

- Worst Negative Slack (WNS): 191.362ns
- Total Negative Slack (TNS): 0.000ns
- Number of Failing Endpoints: 0
- Total Number of Endpoints: 215

Hold:

- Worst Hold Slack (WHS): 191.362ns
- Total Hold Slack (THS): 0.000ns
- Number of Failing Endpoints: 0
- Total Number of Endpoints: 215

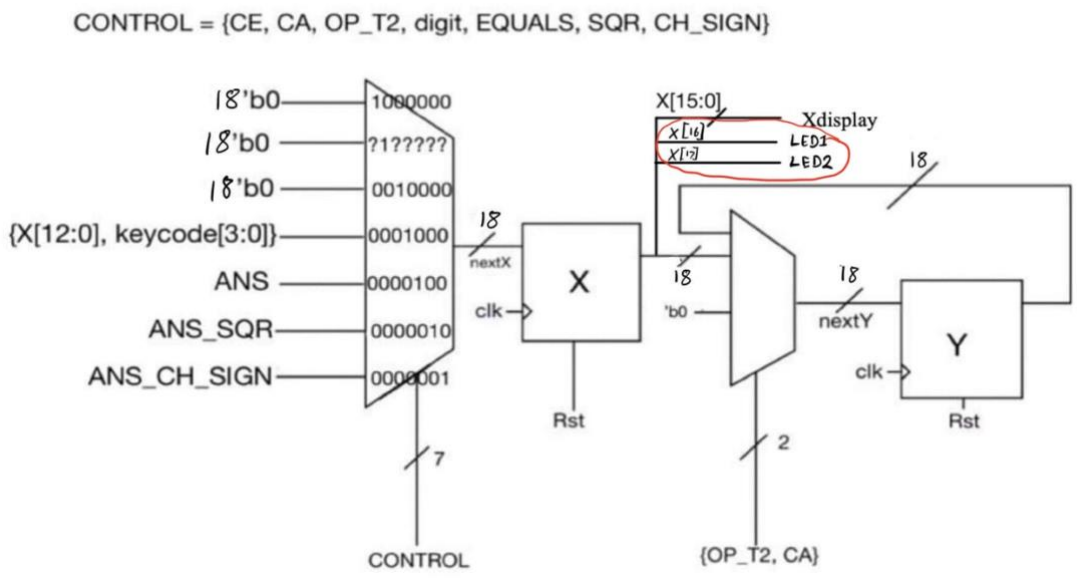
Pulse Width:

- Worst Pulse Width Slack (WPWS): 191.362ns
- Total Pulse Width Slack (TPWS): 0.000ns
- Number of Failing Endpoints: 0
- Total Number of Endpoints: 91

Conclusion

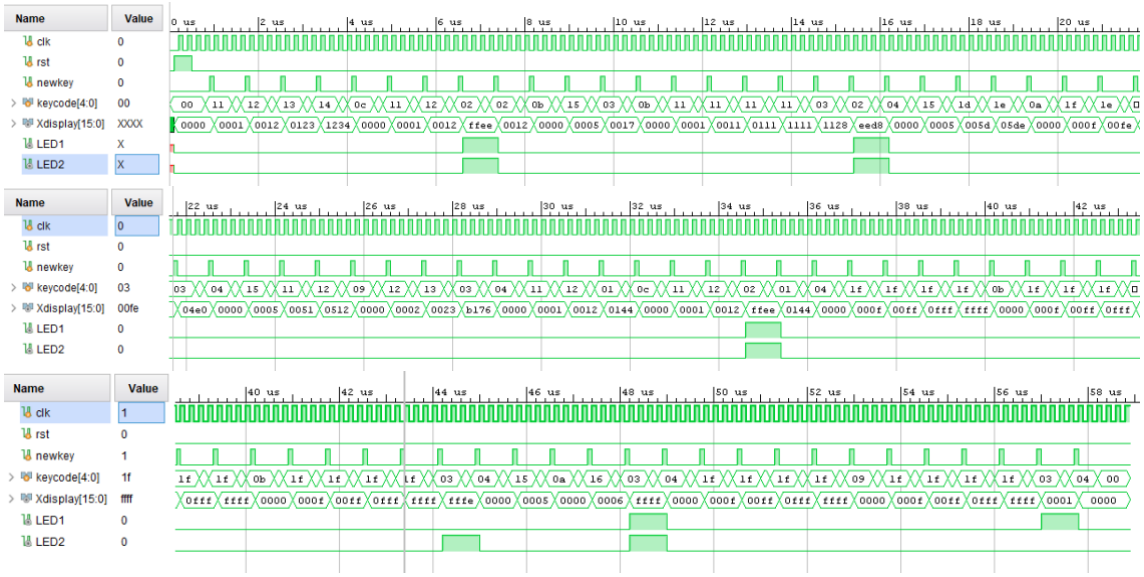
Our calculator worked as expected in all areas except correctly indicating when negative numbers and overflow occurred.

In an attempt to modify this, both LED's were removed and subsequently replaced by the following:



The X register is adjusted to 18 bits instead of 17. Two LED's are still used as follows: LED1 = X[16] and LED2 = X[17].

When we ran the verification with this modification, the following timing diagrams were obtained:



Now, when both LED's are on it indicates a negative number and when only one LED is on it indicates an overflow.

We know that this is a very crude solution to the problem and we would probably be able to come up with a better one if more time was allocated.