ECE 414 – Embedded Systems
Implementation Plan for Lab 05 – LCD Display & Touchscreen

GuoYuan Li (Scribe)
Harry Zhu

Oct 1, 2020

1. Introduction

The objective of this lab is to gain familiarity with the Adafruit 2.4” LCD display by learning the SPI interface between the display and PIC 32 and its hardware connections. In addition, this lab also aims to learn to create a touchscreen driver that interpolates the TFT coordinates to LCD coordinates. Finally, this lab has the objective of using the LCD and touchscreen to build a simple four-function calculator.

2. Requirements

1. Create a new schematic symbol for the LCD display that you will use in diagrams in later labs and the project.
   1.1. Team shall create a new schematic symbol in KiCad named LCD_2_4 that shows the power and I/O connections of the AdaFruit 2.4” LCD display and touchscreen including power and ground. This symbol shall be saved in the team’s Git repository for future use.
   1.2. Team shall create a schematic diagram showing the wiring connections needed to interface a Microstick II with the Adafruit 2.4” LCD display. This diagram shall be saved in the Git repository for future use and included in the lab report.
2. Create a touchscreen driver that translates the raw data input by the initial touchscreen driver into LCD screen coordinates.
   2.1. The touchscreen driver shall be named “`ts_lcd`” and include a header file `ts_lcd.h` and code file `ts_lcd.c`. Additional code modules may be added as necessary to support this driver.
   2.2. The touchscreen driver shall include a function read the status of the touchscreen as follows: `uint8_t ts_lcd_get_ts(uint16_t *x, uint16_t *y);` This function shall return TRUE when a finger or stylus has been placed on the display; the two pointer parameters shall be assigned the current position of a finger or stylus on the display in LCD coordinates.
   2.3. The touchscreen driver shall include an initialization function: void `ts_lcd_init();` This function will perform any initialization and configuration required before the touchscreen is read.
   2.4. The touchscreen driver module shall include a unit-test module that includes the following functions:
      2.4.1. When the touchscreen is pressed, display the numeric screen coordinates of the button press (see the touchscreen example to figure out how to do this).
      2.4.2. When the touchscreen is pressed, draw a 10 pixel X 10 pixel “crosshair” centered at the current screen coordinates of the press.

2.4.3. When the touchscreen is not pressed, display coordinates and crosshair corresponding to the last location where the screen was pressed.

3. Create a four-function calculator using the touchscreen driver and other provided modules. The calculator should perform operations on signed 32-bit integer numbers entered via the touchscreen and displays the result.

3.1. The calculator display shall include a numeric display window at the top of the display.

3.2. The display shall include “buttons” for digits 0-9 and the four arithmetic functions + (add), - (subtract), * (multiply), and / (divide) along with an = (equals) and “CLR” (clear).

3.3. Each button shall respond to a “press” exactly once.

3.4. The user shall enter operands one digit at a time; each time a digit is pressed it shall be appended to the existing entered value multiplied by 10. Entering an operand ends when an operator button is pressed.

3.5. After one operand is entered, the user shall select an operator (+, -, *, /, =) by pressing the appropriate button. All other button presses shall be ignored at this point.

3.6. After the user has selected an operator, the user shall enter a second number following the same procedure as in requirement 3.4. The entry of the second operand shall end when the user presses the equals button or another operator button.

3.7. If the user presses the equals button to end entry of the second operand, the program shall calculate the result of the entered operator on the two operands and display the result.

3.8. If the user presses an operator button to end entry of the second operand, the program shall calculate the result of the previously entered operand. This result shall be treated as a first operand for the new calculation specified by the latest operator button press.

3.9. If at any time the user presses the “CLR” button, the program shall display a result of zero and go back to waiting for entry of a first operand.

3.10. All numbers shall be displayed right-justified with no leading zeros; negative numbers shall be displayed starting with a “-“ (minus sign).

3.11. If the user enters a number or performs a calculation with a result that is too large to store in 32 bits, the program shall display the error string “ERROR” until the “CLR” button is pressed.

3.12. If the user attempts to divide any number by zero, the display shall display the error string “DIV0” until the “CLR” button is pressed.

3. Architectural Description
This section describes the planned architecture for a design, generally partitioned into a hardware architecture and a software architecture.

3.1 Hardware Implementation
The hardwares for this lab includes the PIC 32 microcontroller and the Adafruit LCD touchscreen. These two hardware modules communicate using the standard SPI protocol that the PIC 32 is the master and the LCD touchscreen is the slave. The IM1-3 pins on the touchscreen are connected to 3Vo to achieve this SPI protocol. Other pins connected for communication on the touchscreen are CS, D/C, and MOSI, which is for taking in the output from the PIC 32. Specific hardware connections are shown below in figure 1.

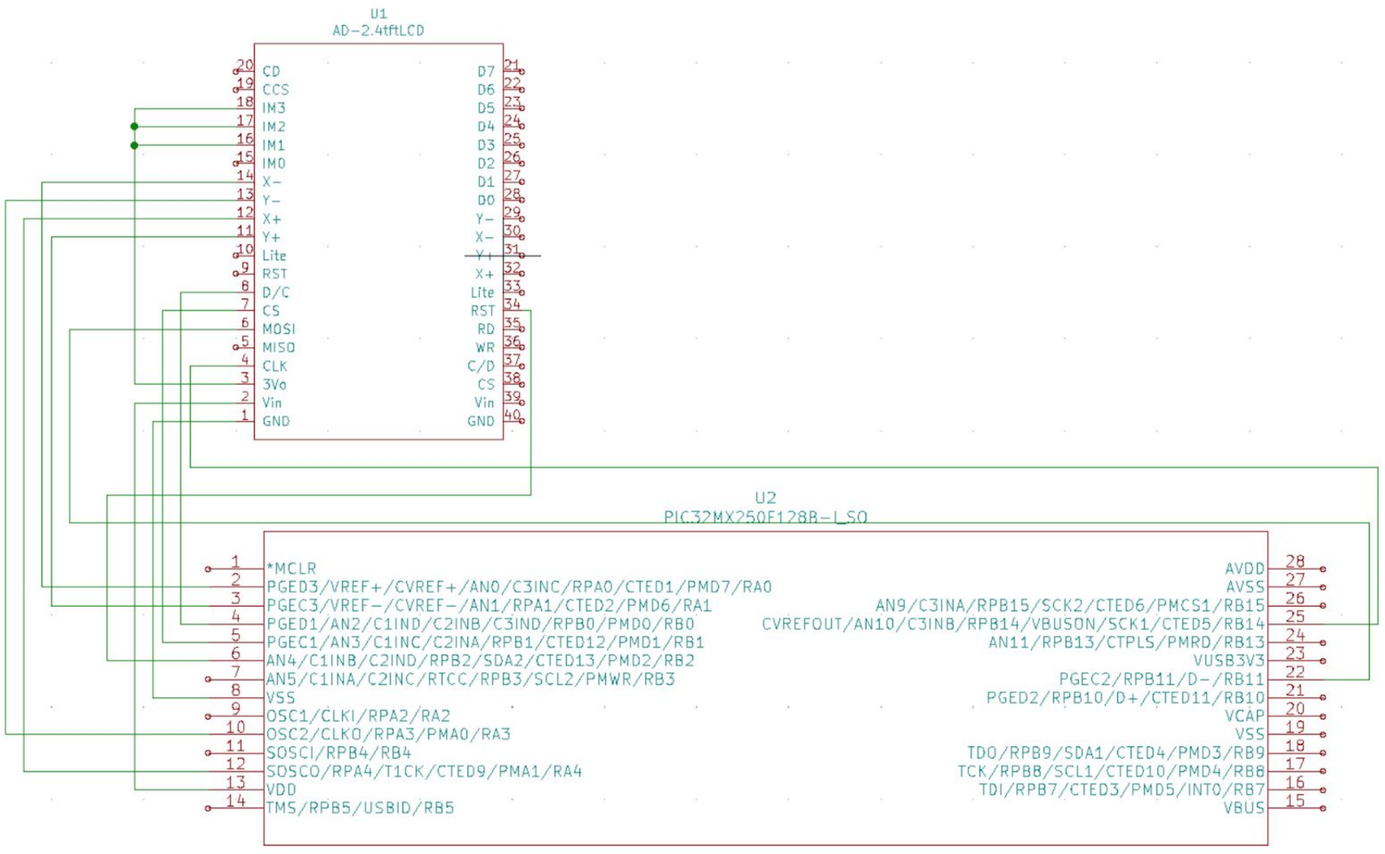
U1
AD-2.4tftLCD
CD
CCS
IM3
IM2
IM1
IM0
X-
Y-
X+
Y+
Lite
RST
D/C
CS
MOSI
MISO
CLK
3Vo
Vin
GND
D7
D6
D5
D4
D3
D2
D1
D0
Y-
X-
Y+
X+
Lite
RST
RD
WR
C/D
CS
Vin
GND
U2
PIC32MX250F128B-I_SO
*MCLR
PGED3/VREF+/CVREF+/AN0/C3INC/RPA0/CTED1/PMD7/RA0
PGEC3/VREF-/CVREF-/AN1/RPA1/CTED2/PMD6/RA1
PGED1/AN2/C1IND/C2INB/C3IND/RPB0/PMD0/RB0
PGEC1/AN3/C1INC/C2INA/RPB1/CTED12/PMD1/RB1
AN4/C1INB/C2IND/RPB2/SDA2/CTED13/PMD2/RB2
AN5/C1INA/C2INC/RTCC/RPB3/SCL2/PMWR/RB3
VSS
OSC1/CLKI/RPA2/RA2
OSC2/CLKO/RPA3/PMA0/RA3
SOSCI/RPB4/RB4
SOSCO/RPA4/T1CK/CTED9/PMA1/RA4
VDD
TMS/RPB5/USBID/RB5
AVDD
AVSS
AN9/C3INA/RPB15/SCK2/CTED6/PMCS1/RB15
CVREFOUT/AN10/C3INB/RPB14/VBUSON/SCK1/CTED5/RB14
AN11/RPB13/CTPLS/PMRD/RB13
VUSB3V3
PGEC2/RPB11/D-/RB11
PGED2/RPB10/D+/CTED11/RB10
VCAP
VSS
TDO/RPB9/SDA1/CTED4/PMD3/RB9
TCK/RPB8/SCL1/CTED10/PMD4/RB8
TDI/RPB7/CTED3/PMD5/INT0/RB7
VBUS

Figure 1. KidCad Hardware Schematic Connection

3.2 Software Implementation
3.2.1 Task 2 Software Implementation
The main module for task 2 is the “ts_lcd” touchscreen driver. Three existing modules `tft_master, tft_gfx, and TouchScreen` provide additional support for this module. `Tft_master` provides initialization functions to initialize the screen. `TouchScreen` outputs a x, y, z raw data from the analog reader. It reads the raw data output of x and y directions from the analog reader on the touchscreen and converts the reading into actual touch screen

coordinates. Since the raw data consists of x, y, and z datas, the conversion can be achieved by creating a structure called point where the point's x and y coordinates are the x and y data read by the analog reader after interpolation and z should be the pressure reading. Specifically, the linear interpolation converts the raw data of x (153-896) and y (110-996) to the actual coordinate of x (0-240) and y (0-320). The press detector outputs 1 when the raw data z is within the range of 200 to 600, which represents a factitious touch. This driver module has a test function which displays the crosshair at the position that is being pressed with the actual x and y coordinates. Finally, the `ts_lcd` driver outputs the x and y coordinate for drawing the crosshair and the `tft_gfx` module provides necessary functions to actually display the crosshair on the screen.

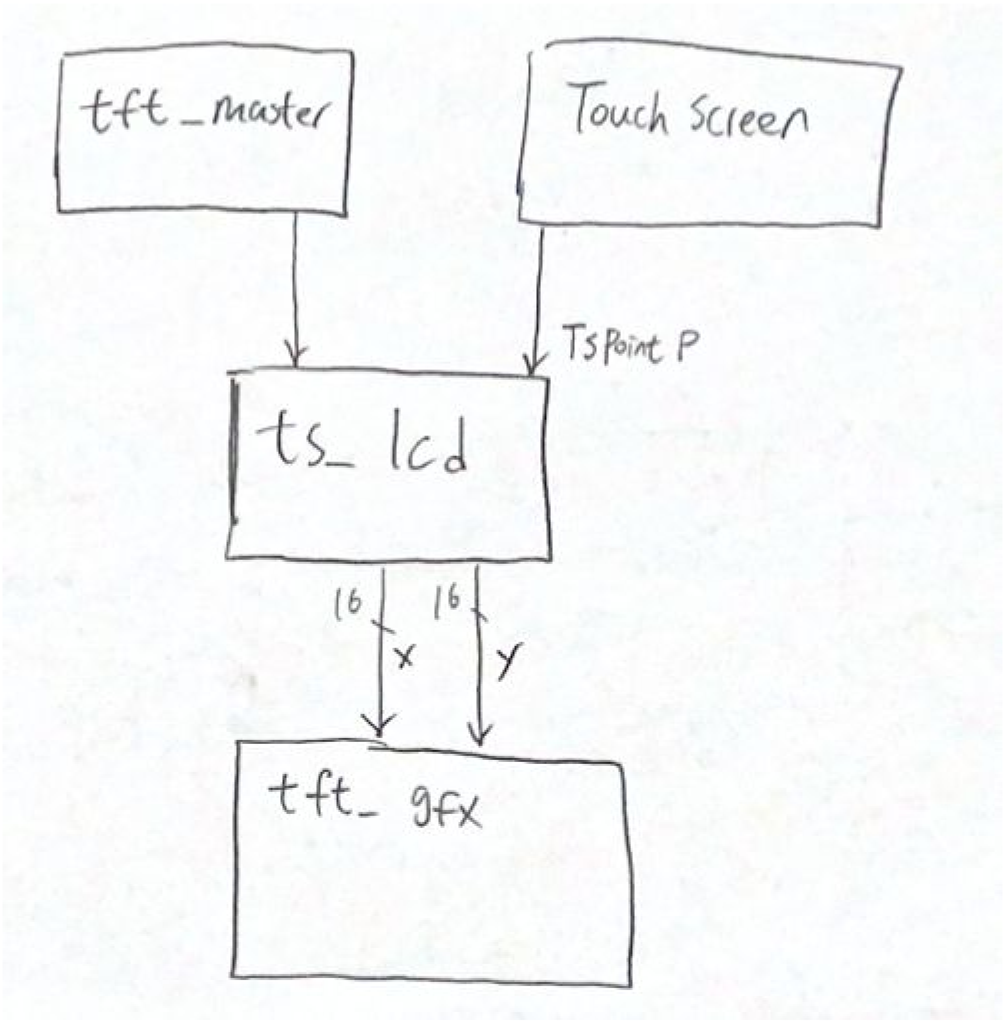

Figure 2. Touch Screen Driver High Level Block Diagram

3.2.2 Task 3 Software Implementation

The touchscreen calculator consists of 2 main synchronous state-machine modules and 2 helper modules: the `calculator_FSM`, the `debouncer_FSM`, a coordinate getter module `getter`, and the calculator display module `ts_DisplayCalculator`. Finally, the `ts_gfx` module helps to display the strings.

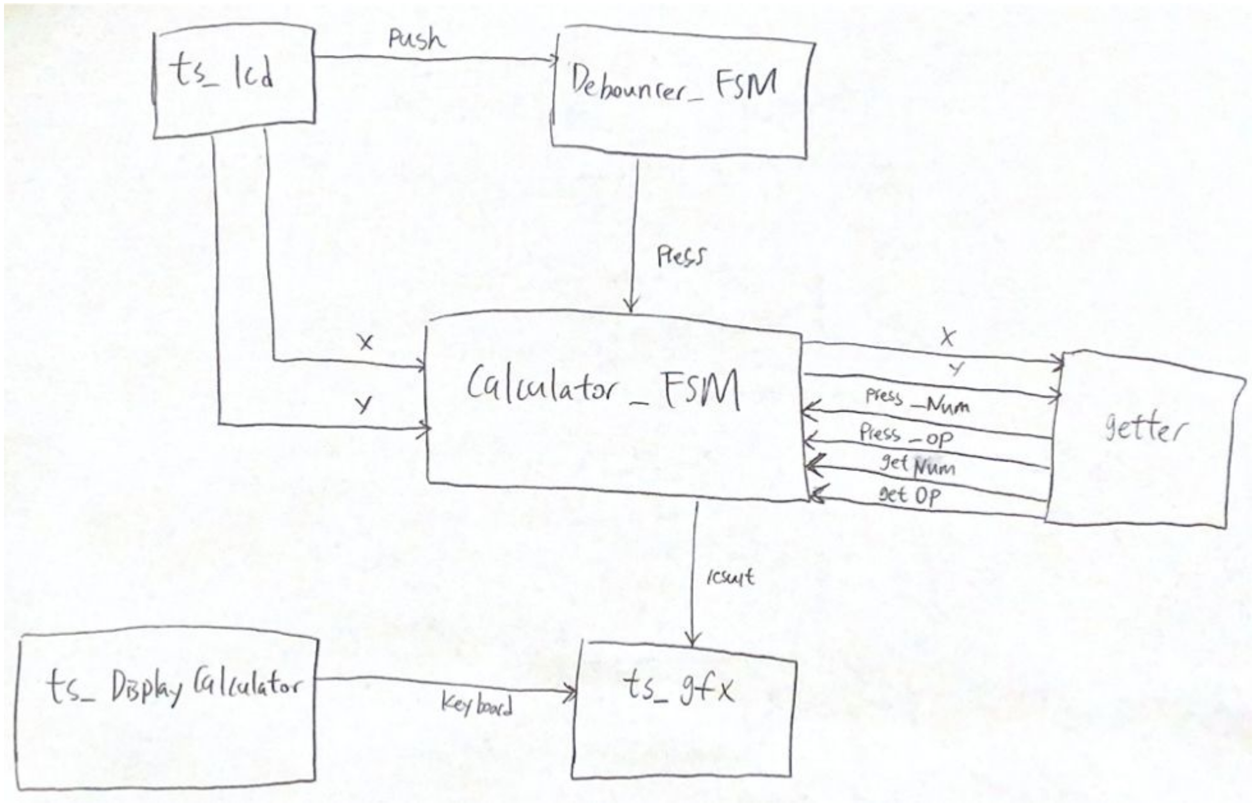

Figure 3. Touch Screen Calculator High Level Block Diagram

ts_DisplayCalculator:

This module displays the keyboard and the top window for results for the touchscreen calculator. It uses a for loop to draw the rectangles which represent each individual key and strings which represent any numbers or operators on each key. Each key has a width of height of 50 and length of 80. There are horizontal and vertical intervals to separate each key. The horizontal interval `intervalY` is 5 and the vertical interval `intervalX` is 3. The left side of the screen is the number region and the right side is the operator region.

Getter:

This module is a helper module that takes in a (x, y) coordinates and outputs if any number of operators is pressed based on this coordinate. First, it checks if a number or operator is pressed. Additionally, it takes the coordinate (x,y) and checks if it is within a specific rectangle region, which represents a key. If the coordinate is within a rectangle region, the content of the key will be returned, either a number outputted by getNum(x,y) or an operator outputted by getOp(x,y).

Calculator_FSM:
The calculator_FSM is the main state machine that controls the steps of executions of the calculator. For example, it oversees the entering of operands and operators, calculation, and displaying the results. The variables for this FSM are 3 32-bit variables, A, B, C, for recording the operands and calculating the result, 2 error variables, div_0 and overflow, indicating errors, and finish and reenter for keep tracking if a state is re-entered or should be finished. It has 7 states: start, opA, operator, opB, Equals, DIV0, and Overflow. The first state is the start where all the variables are initialized. When a number is pressed, it moves to the opA state where it reads the first operand A that it multiplies itself by 10 and adds the new entered digit. It goes back to itself if no operator is entered. If an operator is entered, the FSM goes to the operator state and acquires the operator. Then, it calculates the result and displays it(the first time is just A because B = 0). When another number is entered, it goes to opB, which same thing happens to B as happens to A. After entering operand B, the user can either press another operator or press ‘equal’. If another operator is pressed, reenter is 1 now so that the FSM will first calculate the result using the old operator entered before and record the newly pressed operator for next calculation. If the ‘enter’ key is pressed, the FSM will calculate the result using the operator and displays it. When B is 0 and division operation is executed, div_0 is raised to 1 and the FSM will enter DIV0 state where the error message “DIV0 ERROR” will be displayed. In both opA, opB, operator, and Equals states, the FSM checks if overflow occurs where the number entered or calculated is too large to store in 32 bits and raises overflow to 1 when that happens. Then, the FSM will enter Overflow state where the error message “OVERFLOW” is displayed. Finally, the FSM will return to the start state whenever the “clear” button is pressed such that everything is re-initialized to 0. The detailed state diagram is included below.

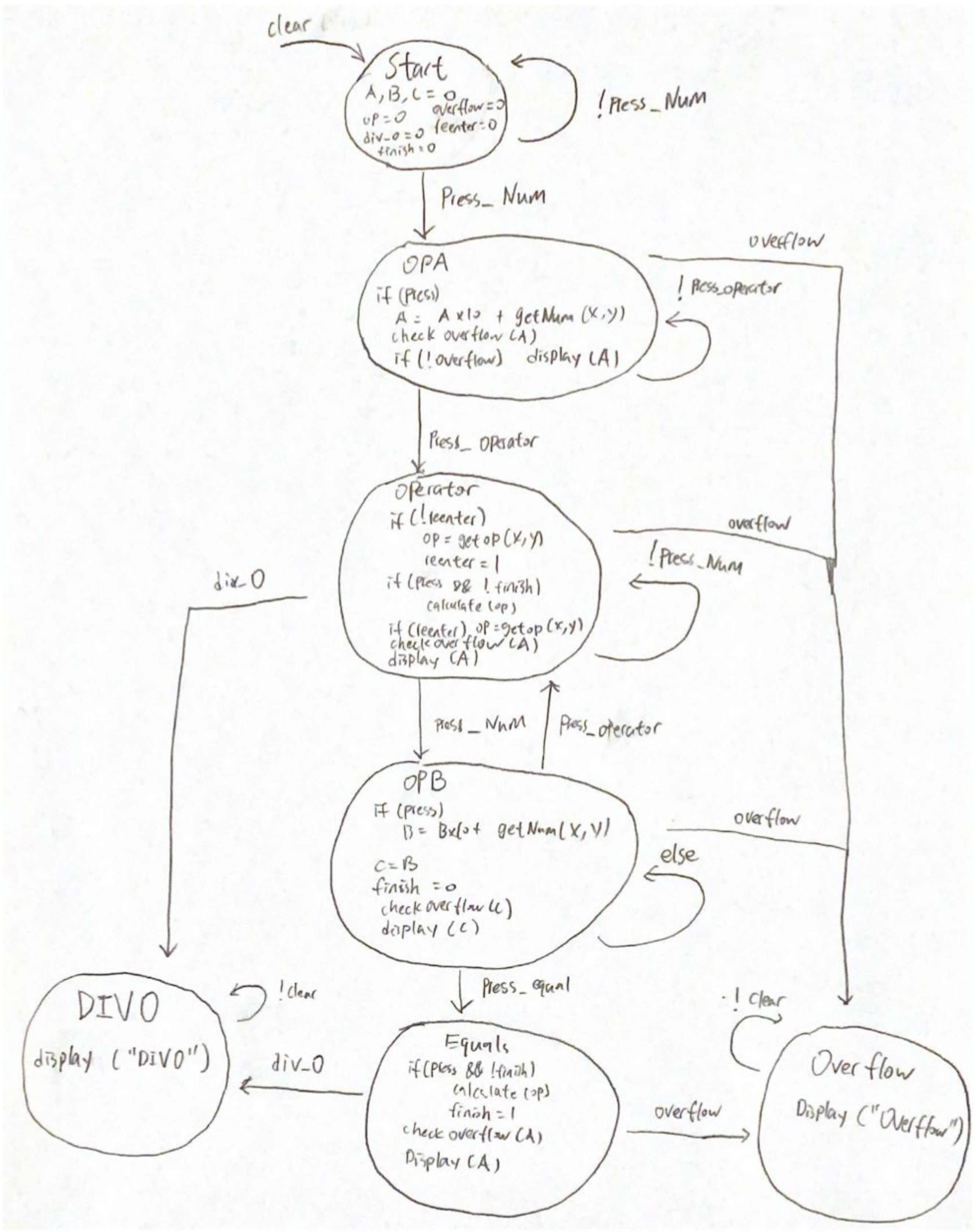

Figure 4. Calculator_FSM State Diagram

Debouncer_FSM:
The debouncer_FSM makes sure that the button is only pressed once such that the number recorded by the calculator_FSM matches the actual number entered by the user. It outputs a variable called out, representing the screen is being pressed. It has 3 states: init, pressed, and debounce. Init state initializes the out to 0. When the signal ‘pushed’ outputted by the ts_lcd module, it will enter the pressed state. The output out is raised to 1. Then, it directly goes into the debounce state where the out is 0, representing a single pulse. If ‘push’ is 0, meaning the screen is no longer pressed, it returns to the init state.

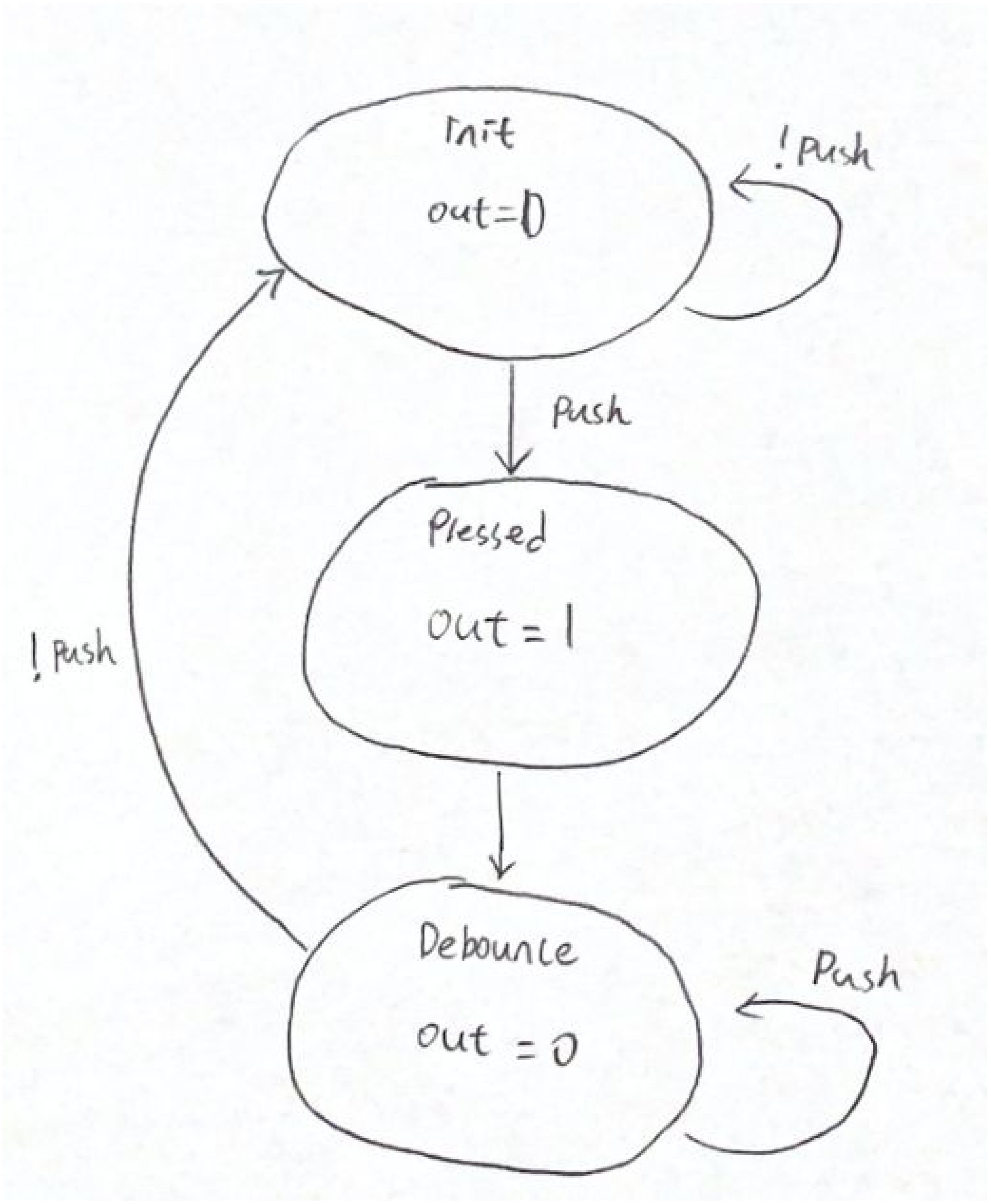

Figure 5. Debounce_FSM state diagram

4. Test Plan

4.1.1 Unit Test for Task 2 - TouchScreen driver ts_lcd

- Press on the touch screen.
    - **Pass**: numeric screen coordinates of the button press is displayed, a 10 pixel X 10 pixel “crosshair” centered at the current coordinates of press is displayed
    - **Fail**: No numeric coordinates are shown or “crosshair” not shown at the currently pressed position.
- Leave the screen unpressed
    - **Pass**: the coordinate corresponds to the last location pressed is displayed and the crosshair stays at the same location.
    - **Fail**: coordinate changes or crosshair changes its location

4.1.2 Acceptance Test for Task 2 - TouchScreen driver ts_lcd

- **T1. Press the top left corner of the screen**
    - **Pass**: (x,y) coordinate displayed should be (0,0) and a crosshair centered at there is displayed
    - **Fail**: (x,y) coordinate displayed is not (0,0) or a crosshair centered at the origin is not shown.
- **T2. Press the center of the screen**
    - **Pass**: (x,y) coordinate displayed should be (120,160) and a crosshair centered at there is displayed
    - **Fail**: (x,y) coordinate displayed is not (120,160) or a crosshair centered at the origin is not shown.
- **T3. Remove the finger originally pressed at the center**
    - **Pass**: (x,y) coordinate displayed should be (120,160) and a crosshair centered at there is displayed
    - **Fail**: (x,y) coordinate displayed is not (120,160) or a crosshair centered at the origin is not shown.
- **T4. Press the finger on the screen and move it around**
    - **Pass**: (x,y) coordinates are changing as the finger moves. Coordinates decrease as finger moves to top-left regions and increase as finger moves to bottom-right regions. Crosshair also follows the finger and the correct numeric display.
    - **Fail**: (x,y) coordinates are not changing as the finger moves or decreasing or increasing directions are wrong. Crosshair is not following the finger.

	Requirements					
Tests	**2.1**	**2.2**	**2.3**	**2.4.1**	**2.4.2**	**2.4.3**
T1.	**x**	**x**	**x**	**x**	**x**	
T2.	**x**	**x**	**x**	**x**	**x**	
T3.	**x**	**x**	**x**	**x**	**x**	**x**
T4.	**x**	**x**	**x**	**x**	**x**	**x**

Figure 6. Task 2 touch screen driver Traceability Matrix

4.2.1 Unit Test for Task 3 - Calculator

4.2.1.1 ts_Displaycalculator module unit test

- Turn on the touchscreen
    - **Pass**: The calculator is correctly displayed with the keyboard and the window
    - **Fail**: Nothing shows up or keys are missing, display window is not correct

4.2.1.2 getter unit test

- Enter the (x,y) coordinate for number “3”
    - **Pass**: “3” is returned. Press_num is outputted.
    - **Fail**: “3” is not returned. Press_num is not outputted.
- Enter the (x,y) coordinate for operator “+”
    - **Pass**: the op-code for “+” is 0. “0” is returned. Press_op is outputted.
    - **Fail**: “0” is not returned. Press_op is not outputted.

4.2.1.3 Calculator_FSM unit test

- Will be tested integrated with other modules in Acceptance Test since there it is very hard to test it alone

4.2.1.4 Debounce_FSM unit test

- Will be tested integrated with other modules in Acceptance Test since there it is very hard to test it alone

4.2.2 Acceptance Test for Task 3 - Calculator

- **T1. Turn on the touchscreen**
    - **Pass**: The calculator is correctly displayed with the keyboard and the window
    - **Fail**: Nothing shows up or keys are missing, display window is not correct
- **T2. Calculate by pressing buttons “73 + 49 = ” in order and press clear**
    - **Pass**: 73 and 49 are the first and second operands that show up on the window. Final result 122 appears on the window at the top of the calculator. Press clear to clear

    - **Fail**: nothing shows up or wrong number shows up
- **T3. Calculate by pressing buttons “851 - 987 =” in order and press clear**
    - Pass: 851 and 987 are the first and second operands that show up on the window. Final result -136 appears on the window at the top of the calculator
    - **Fail**: nothing shows up or wrong number shows up
- **T4. Calculate by pressing buttons ”991 - 891 * 2 / 100 + 5 = “ in order and press clear**
    - **Pass**: Operands show up on the window correctly. Results 100, 200, 2, 7 appear on the window in order (final result being 7).
    - **Fail**: the operands are not showing up correctly. Results are not showing after the operators are pressed. Final result is not correct.
- **T5. Calculate by pressing buttons “100 - 120 * 2 / 7 + 6 = “ in order and press clear**
    - **Pass**: Operands show up on the window correctly. Results -20, -40, -5, 1 appear on the window in order (final result being 5).
    - **Fail**: the operands are not showing up correctly. Results are not showing after the operators are pressed. Final result is not correct.
- **T6. Try to calculate “55 / 0 = “ and press clear**
    - Pass: Error message “DIV0 Error” is shown. Press “clear” to clear message
    - Fail: No error message is shown or message automatically erased.
- **T7. Try to calculate “2147483647 + 1 = “ and press clear**
    - **Pass**: Error message “OVERFLOW” is shown. Press “clear” to clear message
    - **Fail**: No error message is shown or message automatically erased.
- **T8. Try to enter “2147483648” as an operand and press clear**
    - **Pass**: Error message “OVERFLOW” is shown. Press “clear” to clear message
    - **Fail**: No error message is shown or message automatically erased.

	Requirements											
Tests	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8**	**3.9**	**3.10**	**3.11**	**3.12**
T1.	**x**	**x**	**x**									
T2.	**x**	**x**	**x**	**x**	**x**	**x**	**x**		**x**			
T3.	**x**	**x**	**x**	**x**	**x**	**x**	**x**		**x**	**x**		
T4.	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**			
T5.	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**	**x**		
T6.	**x**	**x**	**x**	**x**	**x**	**x**	**x**		**x**			**x**
T7.	**x**	**x**	**x**	**x**	**x**	**x**	**x**		**x**		**x**	
T8.	**x**	**x**	**x**	**x**	**x**				**x**		**x**	

Figure 7. Task 3 Calculator Traceability Matrix

5. Schedule.
Preliminary design is finished and corrected in actual implementation in the first weekend. Hardware implementation is also done in the first week. Guoyuan will be writing the software code for the calculator_FSM and debounce_FSM module Harry will write the code for getter and ts_DisplayCalculator modules. Both of us will be testing the calculator as a whole in the second week.