EE 337: Microprocessors Laboratory (Spring 2024)

Indian Institute of Technology Bombay

ENDSEM: 35 points (Duration: 3 hours) Date: April 6, 2024

Instructions:

- You are **ONLY** allowed to use your codes from previous labs. Sharing your previous codes and resources to other students during exam is strictly not permitted.
- Use of internet during the course of this examination will be considered as copying and strict action will be taken.
  - **You must put your laptops in airplane mode.**
  - **No internet browser should be opened during the exam**
  - **Install a non-browser pdf reader to read pdf files during the exam.** Examples are Acrobat Reader, Nitro PDF Reader, Sumatra PDF Reader.
- Any discussion during exam is not permitted.
- Lock your screen at the end of 3 hours at 5:00 PM sharp. Any student found writing code after the exam duration will face penalty.
- At the completion of **every question**, **show your solution to an evaluating TA** for evaluation.
- Check the rubrics and show the outputs as and when completed in-order to get partial markings.
- Create separate projects for each question. For submission, create a single zip file consisting of all your projects and name it as `rollnumber_name.zip`.

Introduction:
In this exam you will be performing the following tasks-

✓. Receive inputs into Pt-51 from the external world (which will be your PC/laptop) using UART.
2. Conduct neural network inference on the Pt-51 board utilizing the received input data.
3. Present the outcome of the neural network inference on the LCD screen.
4. Use timer 0 as the "Watchdog timer" (for more details refer part 4) to prevent your application from becoming unresponsive in any situation.

Questions:

1. [5 points] The neural network has 2 inputs, so you'll need to use UART to send them serially from your PC to the Pt-51 board. Use 4800 baud rate. The inputs are positive integers ranging from 0 to 9 only. Write a function to perform this and name it as `task1`. Use on-board LEDs to display the inputs, so as to verify this task.

2. [15 points] The neural network you will be implementing is a multi-class classification model. The architecture is as follows: 1 input layer consisting of 2 input neurons, 1 hidden layer consisting of 3 neurons and 1 output layer consisting of 3 output neurons. Refer to Figure 1 for illustration.

   Hidden layer computations-

   $$h_j = \text{ReLu}\left(\sum_{i=0}^{1} w_{1ij}x_i + b_{1j}\right); \qquad j = 0, 1, 2$$

   Output layer computations-

   $$y_j = \left(\sum_{i=0}^{2} w_{2ij}h_i + b_{2j}\right); \qquad j = 0, 1, 2$$

   Final class prediction-

   $$k = \text{Argmax}\ (\ [y_0,\ y_1,\ y_2]\ );$$

   Note that the neurons in the hidden layers use the Rectified Linear Unit (ReLu) as the activation function. The prediction from output neurons is obtained by using the Argmax function. So you will need to implement both of them.

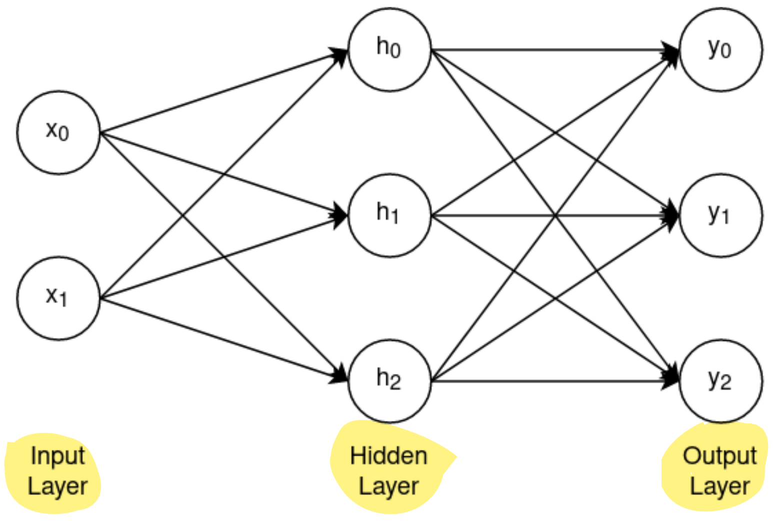

Figure 1: Neural Network Architecture

ReLu Activation function:

$$\mathrm{ReLu}(z_i) = \begin{cases} z_i & : \quad z_i \geq 0 \\ 0 & : \quad z_i < 0 \end{cases}$$

Argmax function returns the index of the maximum value in the array.
Implement both Argmax and Max functions, so that you can display both the index of the maximum number i.e the predicted class and the maximum number itself.

The weights (and biases) are provided to you in the `weights.txt` file.

The end application will require you to obtain inputs via UART, but in order to test the working of your neural network, you can initialize inputs as well in your C-code. To verify this task, you need to run debugger in Keil μVision and monitor the variables. If the expected values are obtained, you will get partial marks. Implement the neural network inside a function named `task2`.

3. [5 points] Display the predicted class and maximum value from `task2` on the LCD screen. The format should be as follows (if the prediction is class 1 with maximum value -6):

```
Class = 1
Max Val = -6
```

The display should be done inside a function named `task3`. Display your first name and roll number on the LCD to ensure it is working initially. You can verify this function to obtain partial marks.

4. [10 points] Before going into the task at hand in this part of the question, a brief introduction to watchdog timer is provided.

Overview:
A watchdog timer is a hardware/software mechanism designed to monitor the operation of an application. It acts as a fail-safe mechanism to ensure that if the system or application fails to respond within a predefined time-frame, it triggers a reset or other corrective action.

Motivation:
Recovery from a stuck program cannot be through software alone. If you write a recovery routine, it will not get executed because your program is stuck elsewhere!
One way out is to use an interrupt. When an interrupt occurs, the stuck program will be interrupted and the control can branch to the interrupt handler and perform corrective actions.

Detecting a stuck program:
Consider the following code snippet. The goal is to detect if the software is stuck and perform corrective action if it is.

```
while(1)
{
    task1();                    // Take inuts via UART
    task2();                    // Perform neural network inference
    task3();                    // Display result on LCD
    retriggWatchDogTimer();     // Retrigger/Restart the timer.
}
```

The solution is to detect the absence of some action. If the program is stuck, the action will not take place and if this can be detected, the problem can be solved.

(i) Use a timer, which is re-triggered or restarted periodically before it can time out.

(ii) Thus in normal operation, it never times out.

(iii) Once configured, the timer runs on its own and the only thing to be done in software (i.e. the main function) is to re-trigger/restart it.

(iv) If the timer exceeds a TIMEOUT, this means the software is stuck within a particular task.

(v) Upon detection of this condition, one can trigger a reset or other corrective action inside the interrupt handler of the timer, to resolve the software's unresponsive state.

```
void timer0_isr()
{
    // Timer 0 ISR
    ...
    watchdog_counter++; // Increment watchdog counter
    ...
    if (watchdog_counter >= TIMEOUT)
    {
        // If watchdog timeout reached, perform necessary actions
        ...
        ...
        watchdog_counter = 0;
    }
}
```

Therefore, in this part, you have to integrate "watchdog timer" in your program and write the code for its interrupt handler.

(A) In your program, if the user doesn't provide input, the `main` function will get stuck in `task1`, which is responsible for receiving inputs.

(B) If the user doesn't give input within 20 seconds, the watchdog timer should recognize this.

(C) The watchdog timer should then jump to `task3` and display the following message on the LCD screen.

```
Program stuck!
Restarting...
```

(D) Then, the program should return to `task1` to wait for input again.

If the user consistently provides input, your application should run smoothly, as if the watchdog timer doesn't exist.

In case your UART part is not complete but you want to verify watchdog timer, you can introduce a forever loop inside `task1` instead.

. . . .

RUBRICS:

Please show state-wise progress to your TA. This is to help us give you partial marks. You might have some part working and later your code may not work due to the changes you have made. In that case, you can get partial credit if you show the previous working part to your TA.

- **5 marks** for correct operation of `task1` i.e inputs are being received from PC to Pt-51 via UART.
- **15 marks** for correct implementation of `task2` i.e neural network inference.
  Partial marks:
  - 1 mark for correct implementation of ReLu function.
  - 1 mark for correct implementation of Max function.
  - 1 mark for correct implementation of Argmax function.
  - 4 marks for each test case. 3 test cases will be provided.
- **5 marks** for correct display on LCD.
  - 1 mark to test LCD working.
  - 4 marks for displaying result of neural network result in `task3`.
- **10 marks** for correct working of watchdog timer.

Note : If all the tasks together are working properly, full 35 marks will directly be awarded without the need to verify individual parts. Similarly if `task1`, `task2` and `task3` are together working properly, 25 marks will be awarded without the need to verify them individually.