Modified from Dr. Brian Fraser's CMPT 433 Assignment by Dr. Matthew Stewart (September 2025)

# Assignment 1: Reaction Timer

Read the entire assignment before beginning!

- This assignment is to be done **individually**.
- Do not give your work to another student, do not copy code found online, and do not post questions about the assignment online other than the course forum. See website for guidance on using AI tools.
- If you have previously taken the course, you may *not* re-use your previous solution.
- Post questions to the course forum (see website); for questions which include more than 2 lines of your code, make your post private.

## 1. Reaction Timer

### 1.1 Hello World

- On the host, create a folder for sharing your compiled applications with the target via NFS:
  `(host)$ mkdir ~/ensc351/public/myApps/`
- Create a folder on your host anywhere you like for your assignment 1 work. Pushing your work to a remote git repository is highly recommended. Possible folder location:
  `(host)$ mkdir ~/ensc351/work/as1/`
- Use the sample CMake project as a base for your assignment.
- Create a `reaction_timer` C program that displays a message using `printf()`
  `"Hello embedded world, from <your name>!"`

 

## 1.2 Use LEDs & Read Joystick

Modify your `reaction_timer` program from above to also play the following game.

**Overview**

The program chooses up or down, turns on an LED to indicate this direction, and times the user's response time for pressing the joystick in the correct direction.

**Application pseudo-code**

Display the welcome message and directions on how to play.

Continuously loop through the following steps:

1. Print a "get ready" message, and do the following four (4) times to flash the BYAI's LEDs:
   a) Turn on the green LED for 250ms, then turn off.
   b) Turn on the red LED for 250ms, then turn off.
2. If the user is pressing the joystick up or down, tell them "Please let go of joystick" (just once) and then wait until the joystick is not pressing up or down.
3. Wait a random amount of time between 0.5s and 3.0s.
4. If, after the delay, the user is pressing the joystick up or down, tell them "too soon" and go back to step 1. No need to check for joystick state during the delay.
5. Picks a random direction (up or down)
   a) Print the direction name to the screen.
   b) Display on the BYAI's LED:
   If chose "Up", the turn on the green LED.
   If chose "Down" turn on the red LED.
6. Time how long it takes for the user to press the joystick in any direction.
   If timer > 5s, exit program right away with a message.
7. Process the user's joystick press:
   a) If the user pressed up or down *correctly*, then:
   - print a message,
   - if this was the fastest correct response time yet then display that in a message,
   - display this attempt's response time, and best response time so far (in ms),
   - flash the green LED on and off five times in one second.
   b) If the user pressed up or down *incorrectly*, print a message and flash the red LED on and off five times in one second.
   c) If the user pressed the joystick left or right, print a message and quit.

**Requirements**

- Design
  - You must use the sample CMake project on the course website which has the `hal/` and `app/` folders.

 

- You must have *at least two* HAL modules: one for LED, one for Joystick. You may have (many) more HAL modules if you like.
- You must have your game logic in the `app/` folder. OK to have all the app in one `.c` file, but you may choose to break it up.

- LEDs:
  - At shutdown, your C program must leave both BYAI LEDs off.
- Joystick:
  - Set your threshold to detect the joystick being pressed in a direction so that the user has to press the joystick most of the way in a direction to be triggered.
  - It is up to you to put together A/D circuit with the provided electronic components to be able to operate the joystick.
  - *Hint: Try making the joystick auto-calibrate. Start with some reasonable initial min/max values for each direction, and as the user moves the joystick around update the min/max range for your calibration.*

**Sample Output**

- Sample console output for a few passes of the game, followed by pressing right to exit. Your output does not have to match exactly, but needs to be at least as clear.

```
Hello embedded world, from Dr. Brian!

When the LEDs light up, press the joystick in that direction!
(Press left or right to exit)
Get ready...
Press UP now!
Correct!
New best time!
Your reaction time was  926ms; best so far in game is  926ms.
Get ready...
Press DOWN now!
Incorrect.
Get ready...
Press DOWN now!
Correct!
Your reaction time was 2289ms; best so far in game is  926ms.
Get ready...
Press UP now!
User selected to quit.
```

- This output shows one pass through the game, followed by no input for 5s.

 

```
Hello embedded world, from Dr. Brian!

When the LEDs light up, press the joystick in that direction!
(Press left or right to exit)
Get ready...
Press DOWN now!
Correct!
New best time!
Your reaction time was  946ms; best so far in game is  946ms.
Get ready...
Press UP now!
No input within 5000ms; quitting!
```

 

**Hints**

- *Hint: Good modular and function design will save you a lot of time throughout this course!*
  - Each module should have an initialization and cleanup function.
    - Make the initialization function set a flag (boolean variable) indicating it has initialized.
    - To catch bugs related to not initializing the module, make all other functions in that module assert that the module has been initialized.
  - For joystick, I suggest you create an enum to represent the four directions, plus a 'no direction'. Then create a function which returns which direction the joystick is pressed. Note this won't handle multiple directions at once, but that's OK for this application.

- *Hint: For timers*
  - To start a timer, record the current time as the start-time. Then compare the current time to the start-time as your program runs.
  - Work with time in milliseconds.
  - Time often means elapsed time since ~1970, so with milliseconds we should use the `long long` data type to ensure we don't overflow a 32-bit int.
  - To print a `long long`, use:
    ```
    long long x = 123456789LL;
    printf("Big number is %lld right!", x);
    ```
  - You can get the current time with:

```
static long long getTimeInMs(void)
{
    struct timespec spec;
    clock_gettime(CLOCK_REALTIME, &spec);
    long long seconds = spec.tv_sec;
    long long nanoSeconds = spec.tv_nsec;
    long long milliSeconds = seconds * 1000
                + nanoSeconds / 1000000;
    return milliSeconds;
}
```

  - You can wait a number of milliseconds with:

```
static void sleepForMs(long long delayInMs)
{
    const long long NS_PER_MS = 1000 * 1000;
    const long long NS_PER_SECOND = 1000000000;

    long long delayNs = delayInMs * NS_PER_MS;
    int seconds = delayNs / NS_PER_SECOND;
    int nanoseconds = delayNs % NS_PER_SECOND;

    struct timespec reqDelay = {seconds, nanoseconds};
    nanosleep(&reqDelay, (struct timespec *) NULL);
}
```

 

## 2. RFS Customization and output capture

- The following guides are posted on the course website. These guides tell you how to do things, the assignment tells you want must be done for marks:
  - Quick-Start guide
  - Networking guide
  - NFS guide
  - RFS Customization guide

### 2.1 Customize the BYAI's RFS

On the target, use the following guide to make the listed changes.

- Networking guide
  - Create `internetToTarget.sh` scripts on both the host and the target to allow the target to access the internet.
- NFS Guide
  - On target, create `mountNFS.sh` script to mount the NFS folder from the host
- RFS Customization Guide
  - Change the target's `hostname`
  - Change the target's message displayed after logging in via SSH
  - Make the target execute your assignment solution automatically by editing your `.profile` file
    - Hint: After this assignment, disable this "feature"!

### 2.2 Capture output using SSH

Perform the following commands on the **host** in a terminal. If you make a typo or get an error, correct the error and continue; you do *not* need to edit the capture to remove the error.

1. Display the host's network configuration (`ip addr`)
2. Ping the target and let in ping at least once; then press Control C to stop it
3. SSH to the target
4. Play a couple rounds of your game:
   - Play at least one round where you do the correct action
   - Play at least one round where you do the incorrect action
   - Exit the game using the joystick left or right
5. Connect the target to the internet
6. Display the target's network configuration
7. Ping the host
8. Ping `google.ca`
9. Display the contents of the following files (use the `cat` command)
   - `/boot/firmware/ID.txt`
   - `/proc/version`
   - `/proc/uptime`
10. Mount your NFS shared directory using your script.
11. Use `ls` to display the contents of the NFS shared directory
12. Exit the SSH session (`exit`)

Copy the text of your session to a new file named `as1-capture.txt`
*Hint: Select the text in the terminal and copy-and-paste it into a text file using* `gedit` *or the like.*

 

# 3. Deliverables

Submit the items listed below to the CourSys: https://coursys.sfu.ca

1. `as1-reaction_timer.tar.gz`
   Compressed copy of your project. Delete the `build/` folder first so it's much smaller.

   *Hint: Compress the `as1/` directory with the command like:*
   ```
   (host)$ tar cvzf as1-helloWorld.tar.gz as1
   ```

2. `as1-capture.txt`

Please remember that all submissions will be compared for unexplainable similarities. If you have taken the course before, you may **not** use your work from previous semesters as any part of your solution. You are likely to find other student's previous submissions online. Copying from another source is academic dishonesty. Anyone found to have copied work from a prohibited source will be give a 0 and a report on file with the university.

## 3.1 Informal Milestones

Here are some steps and milestones that may help you work through this assignment.

- How to start
  - Follow the quick start, NFS, networking, LED and GPIO guides to get started.
  - Build and run the sample CMake project on the target.
  - Do a little design: What HAL modules will you need for this assignment? Design their methods (.h file) that you want your main app to call. Writing those modules first is a reasonable start.
- Half done
  - Fully working HAL modules to control the joystick and LEDs needed for this assignment.
- Final checks
  - Double check that your solution performs the steps correctly as described in the application pseudo-code. This is what is really being observed for marking.
  - Review the learning objectives for this assignment (see webpage).

## 3.2 Revision History

- V1: initial post

 