[1.Task 1](#_Toc1675896817)

[2.Implementation 3](#_Toc442180164)

[2.1 Main Window 3](#_Toc1936479271)

[2.2 Start / Stop Button 3](#_Toc327987228)

[2.2.1 Start 3](#_Toc1492257018)

[2.2.2 Stop 3](#_Toc1190308711)

[2.3 Idle / Busy Indicator 4](#_Toc290102985)

[2.4 Display Indicators 4](#_Toc1326850801)

[2.5 Time Input / Indicator 5](#_Toc115948970)

[2.6 Graph 5](#_Toc1880637262)

# Task

As the advancement in Embedded systems has progressed, more and more complex task is achieved by the embedded systems. As a result, GUI in Embedded systems has more meaning than ever. One of the possible ways to design a GUI of Embedded systems is using the FLTK libraries for C++. This development is possible in Windows and Linux, the system used by me is Linux. I prefer Linux, because the terminal functions can be easily executed and installation of the libraries is quite easy compared to windows system.

The functional requirements for the GUI are as follow,

* Embedded GUI design must be applicable to touchscreen: big buttons, no Sub windows, no menus, no sliders
* Embedded GUI Buttons must be responsive during SPS measurement: asynchronous programming
* Embedded GUI should detect SPS state change from Response-Simulation e.g., Busy to Idle and update Buttons accordingly
* **Start / Stop Button**:
  + Start = Send SPS Set to set measurement time and then send SPS Busy to start measurement.
  + Stop = Send SPS Idle to stop measurement before completion
* **Idle / Busy Indicator**
* **Display Indicator**: Show SPS measurement data 1 and 2 current values
* **Time Button / Indicator**:
  + Idle = Change measurement time
  + Busy = Show remaining time
* **Graph**: Scale to set measurement time, Plot SPS measurement data 1 and 2, indicate elapsed time

# Implementation

This chapter will discuss in details about the implementation of the GUI for all the components in the project in its own section.

## 2.1 Main Window

As per the requirement there is only one window which consists of all the components mentioned in the Functional requirements. The size of the main window is fixed but can be changed by just two global defines WINDOW\_X and WINDOW\_Y. The window components dynamically adapt to the change in the Window. The majority of the part of the window is the Graph, all the controls and the indicators are present at the bottom of the window as shown in the Figure\_.

## 2.2 Start / Stop Button

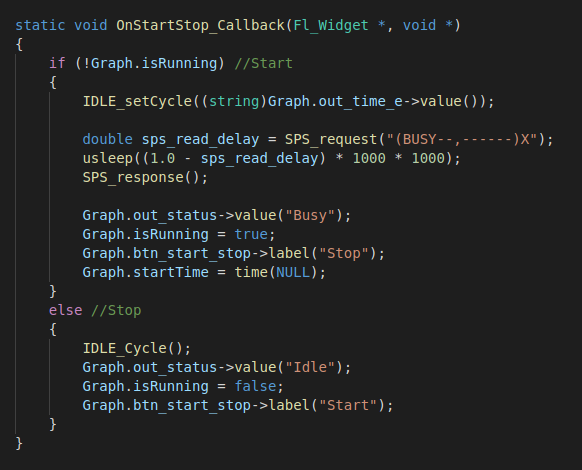
Start/Stop button does exactly what it says, it starts and stop the measurement reading from the SPS. This button also changes some of the indicators such as the Idle/Busy and starts/stops the time indicators.

### 2.2.1 Start

On Start, it takes the input of measurement time and converts it into set value which is then set in the SPS using the method *IDLE\_setCycle()*. Then it is time to set it to busy state and takes a response from the SPS. The status of the SPS is changed on the GUI and the status of the reading can be set to true. The Start button is than converted into Stop button and the Elapsed time counting can be started.

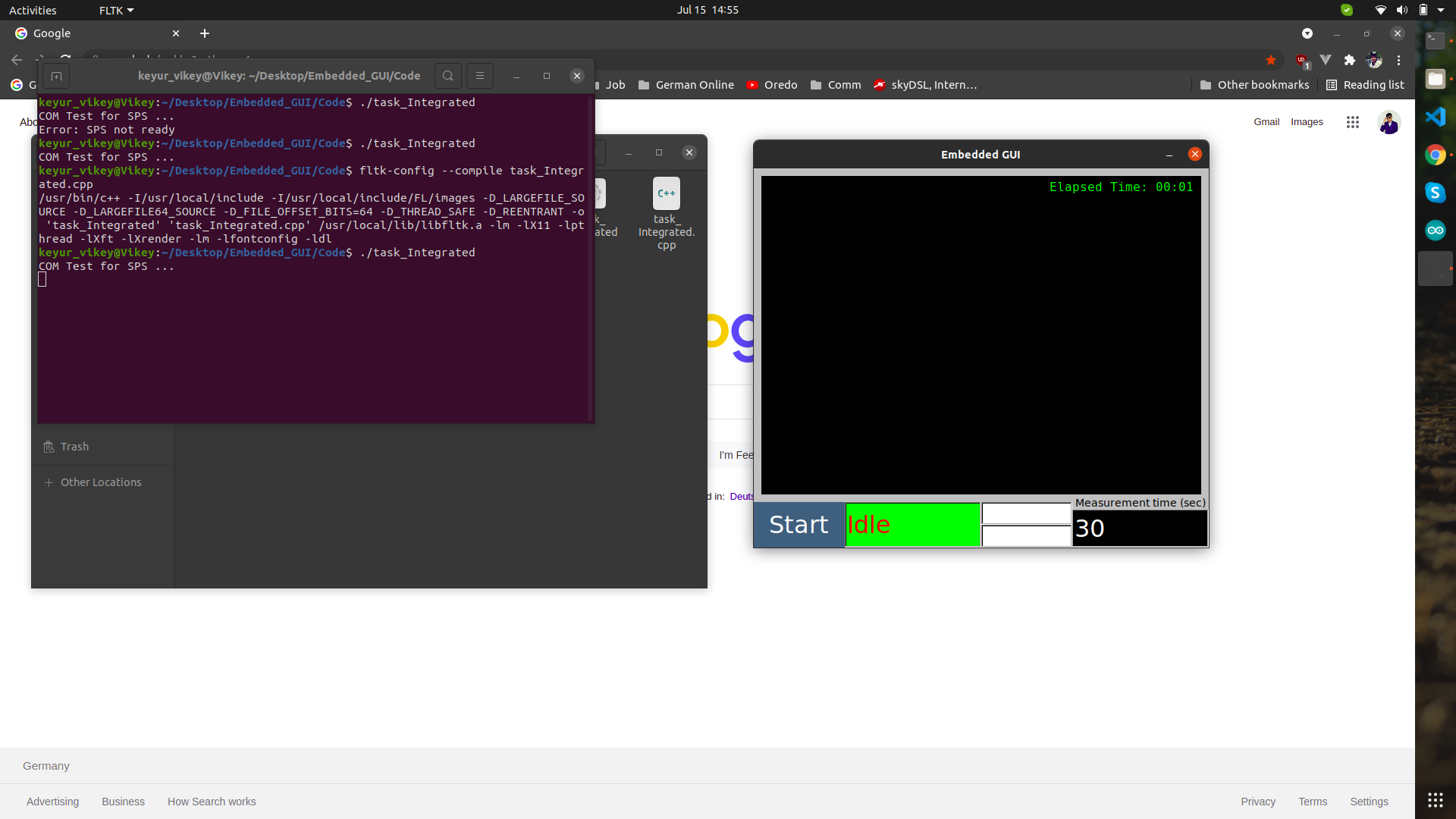
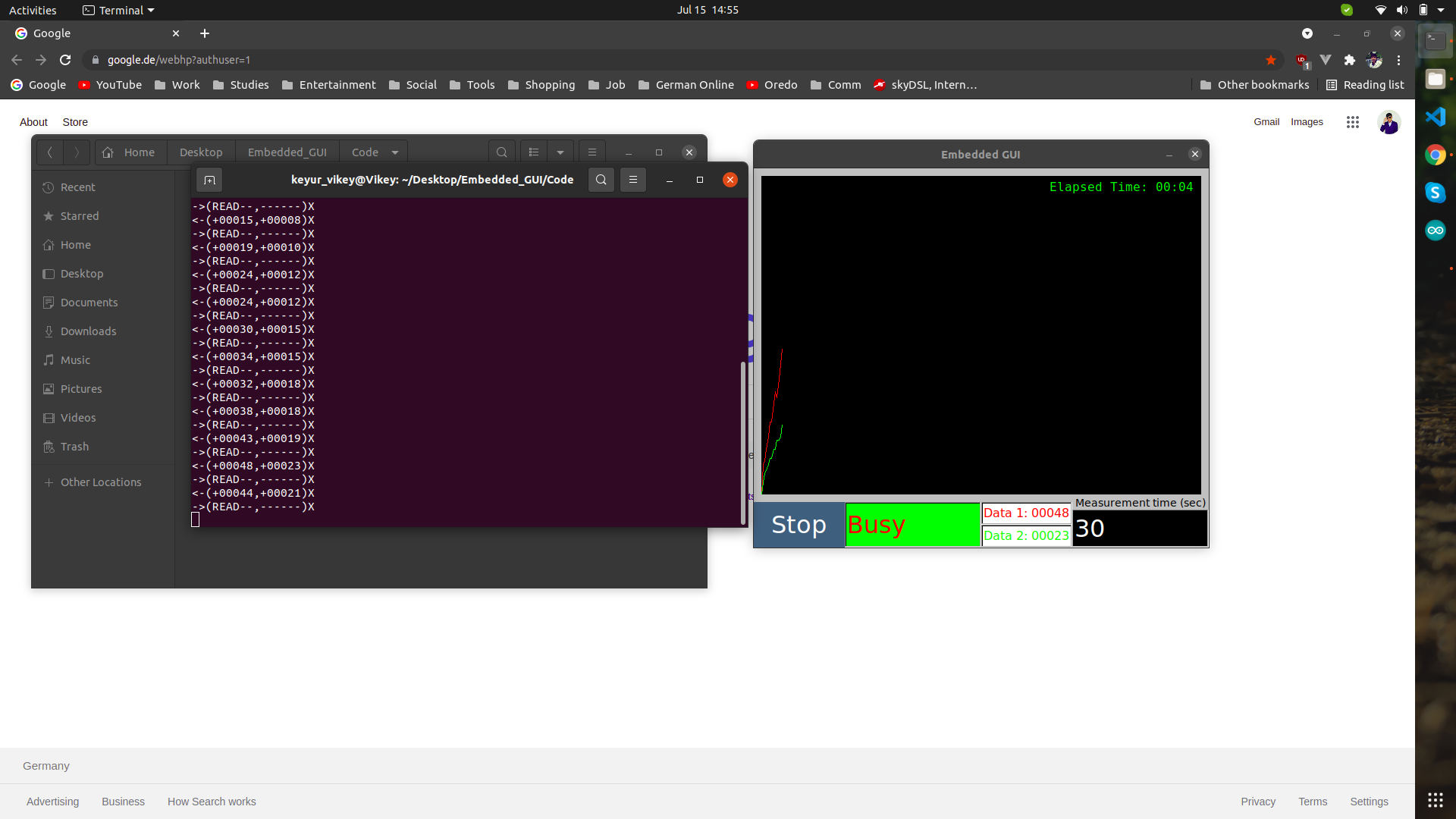
### 2.2.2 Stop

On Stop it stop the reading of the measurement values from the SPS, sets the SPS in Idle state and converts the button in to Start for user to start the measurement again.



## 2.3 Idle / Busy Indicator

This indicator indicates the state of the SPS and its value is changed by the button Start/Stop as mentioned in the chapter 2.3.

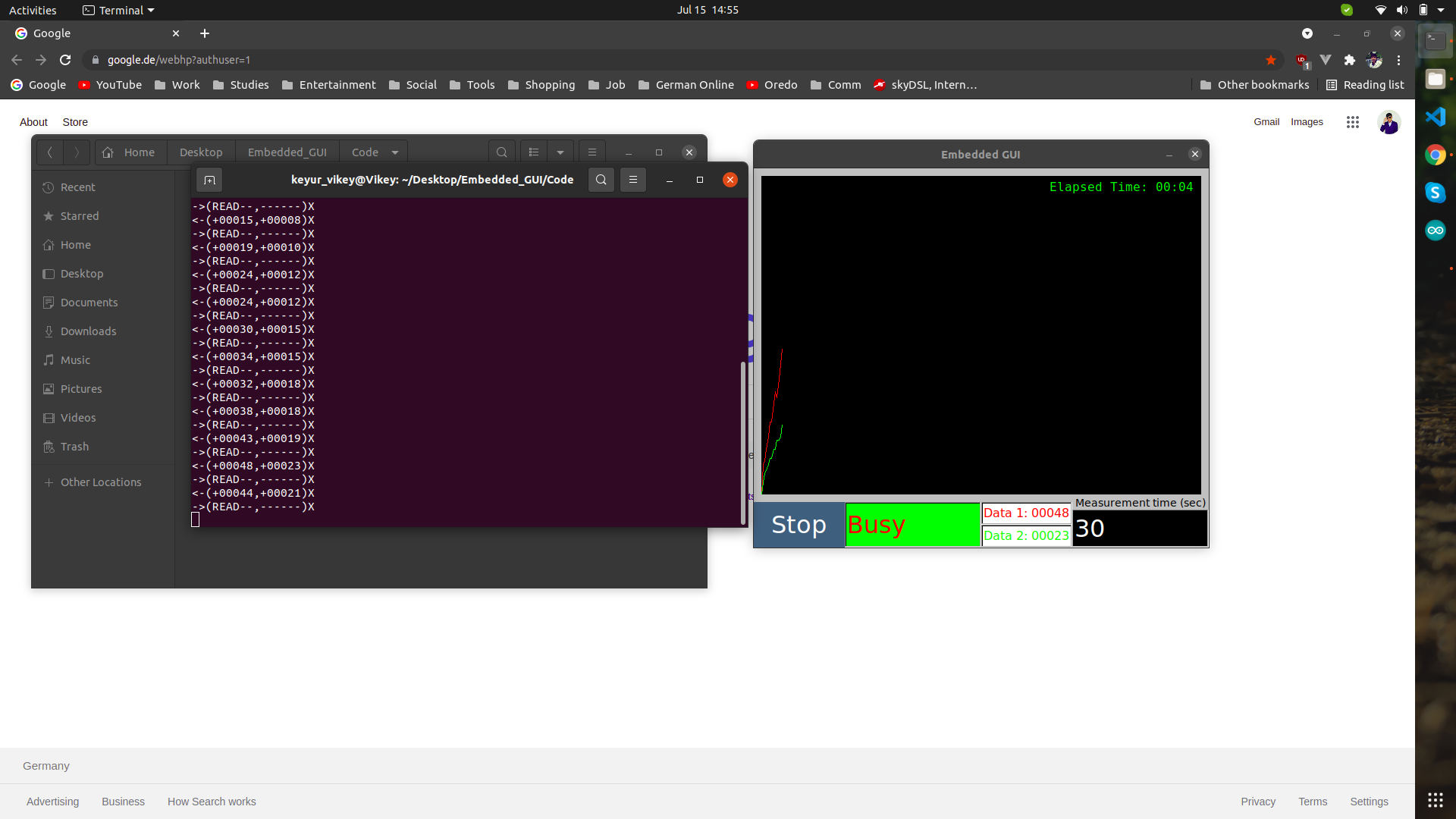
 

## 2.4 Display Indicators

Display indicator show the current values of the data read from the SPS. This data changes with every time the graph is updated after reading the data from the SPS. As show in the Figure\_, to make it more readable, this data is color coded corresponding to the graph.

## 2.5 Time Input / Indicator

Input field is present at the end of the Controller bar, this takes input in sec and later on is converted in appropriate set value for the SPS. This value is not 100% accurate in conversion and has the tolerance of about ±3s as seen in the Elapsed time. The Elapsed time indicator is present at the top right corner of the graph. This time calculation starts on start button as mentioned in the Chapter 2.3.1.



## 2.6 Graph

The Graph is the most essential part of this whole Task. This graph is drawn using two Arrays that stores the values read from the SPS, shown in Figure\_. This drawing processes is accomplished by timeout from FLTK, the two methods *Fl::add\_timeout()* and *Fl::repeat\_timeout()* represents a loop in which the following Prosses takes place,

1. Draws the Graph Box
2. Adds the timeout
3. Read values from the SPS
4. Checks if the SPS is in IDLE state and converts the values in Int
5. Stores the values in the corresponding Array
6. Displays the current values in GUI
7. Redraws the graph
8. Repeat timeout to step 3