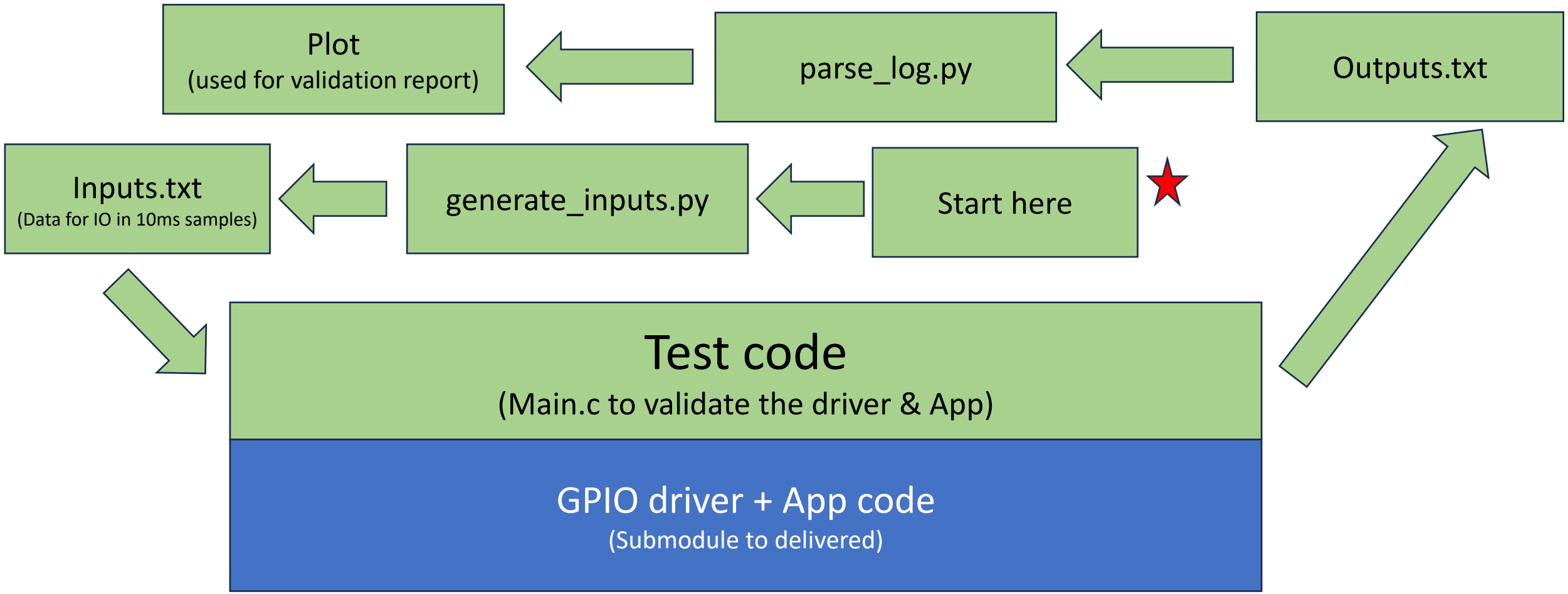
GPIO driver validation report

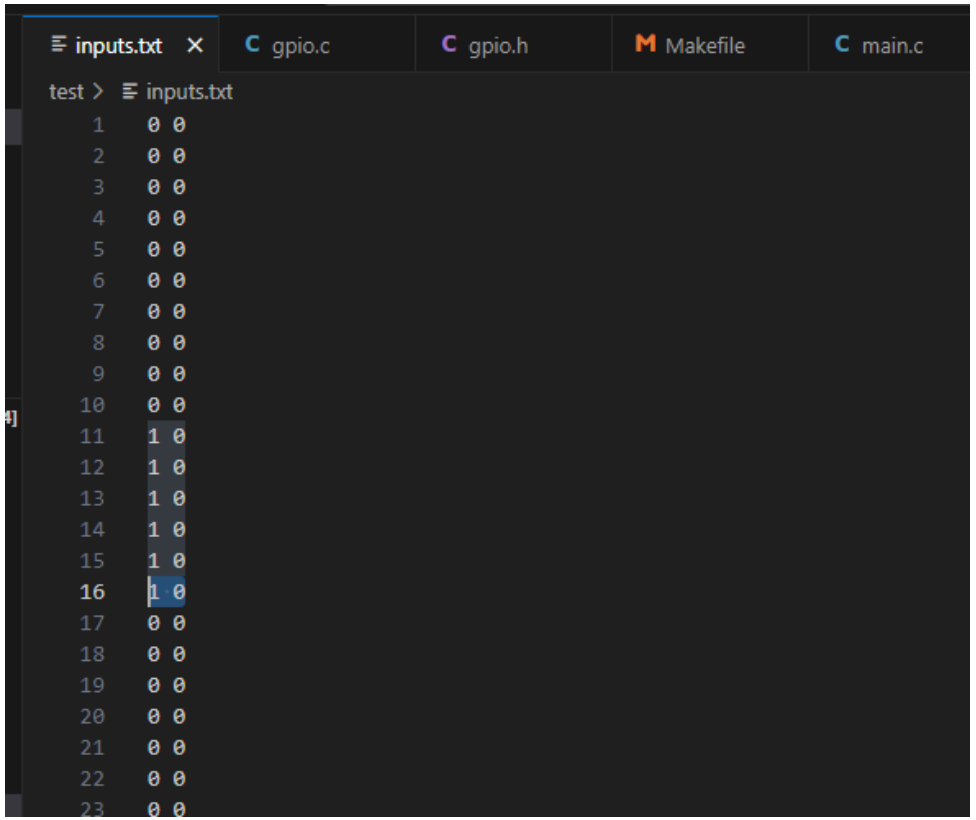
Baranidharan KARUPPUSAMY

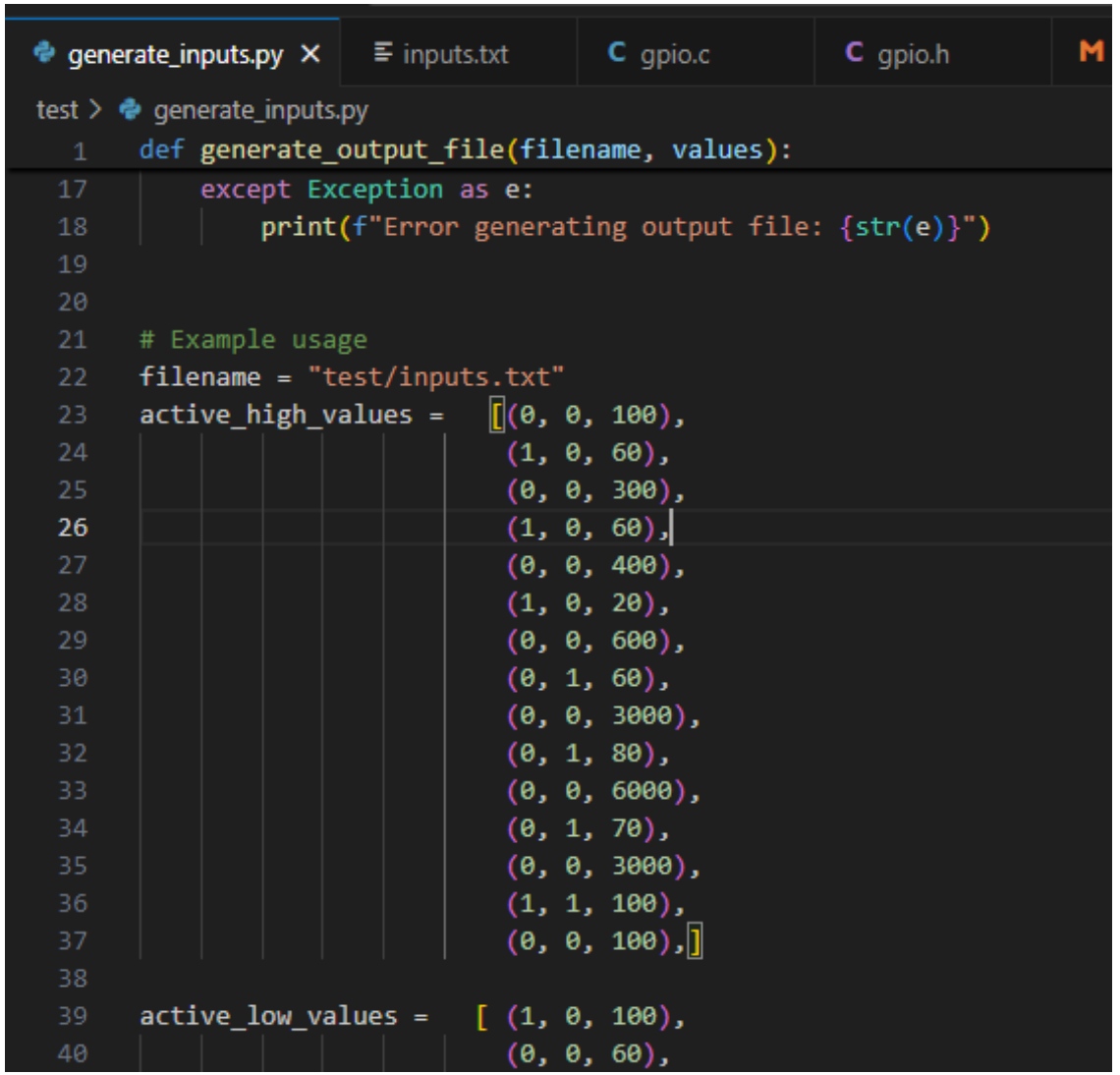
Driver development using BDD



Sample input data

- Data for IO in 10ms samples
- 1 for HIGH
- 0 for LOW
- First data PB1
- Second data PB2
- Generated programmatically





Parsing output data

- Output data parse to extract input, output & timing information to dataframe
- Dataframe plotted for inference

main.c U | parse_log.py U | gpio.h U | gpio.c U

parse_log.py

```
5   def parse_log(log_string):
17      for line in log_lines:
21          if match:
23              time = int(match.group(1))
24              input0 = 0 if match.group(2) == '_' else 1
25              input1 = 0 if match.group(3) == '_' else 1
26              green_led = 1 if match.group(4) == 'O' else 0
27              red_led = 1 if match.group(5) == 'O' else 0
28
29              # Store extracted values in dictionary
30              log_data = {
31                  "time": time,
32                  "pb_1": input0,
33                  "pb_2": input1,
34                  "green_led": green_led,
35                  "red_led": red_led
36              }
37              parsed_log.append(log_data)
38      return parsed_log
```

main.c U | parse_log.py U | gpio.h U | gpio.c U | led.c U | led.h U | push_button.

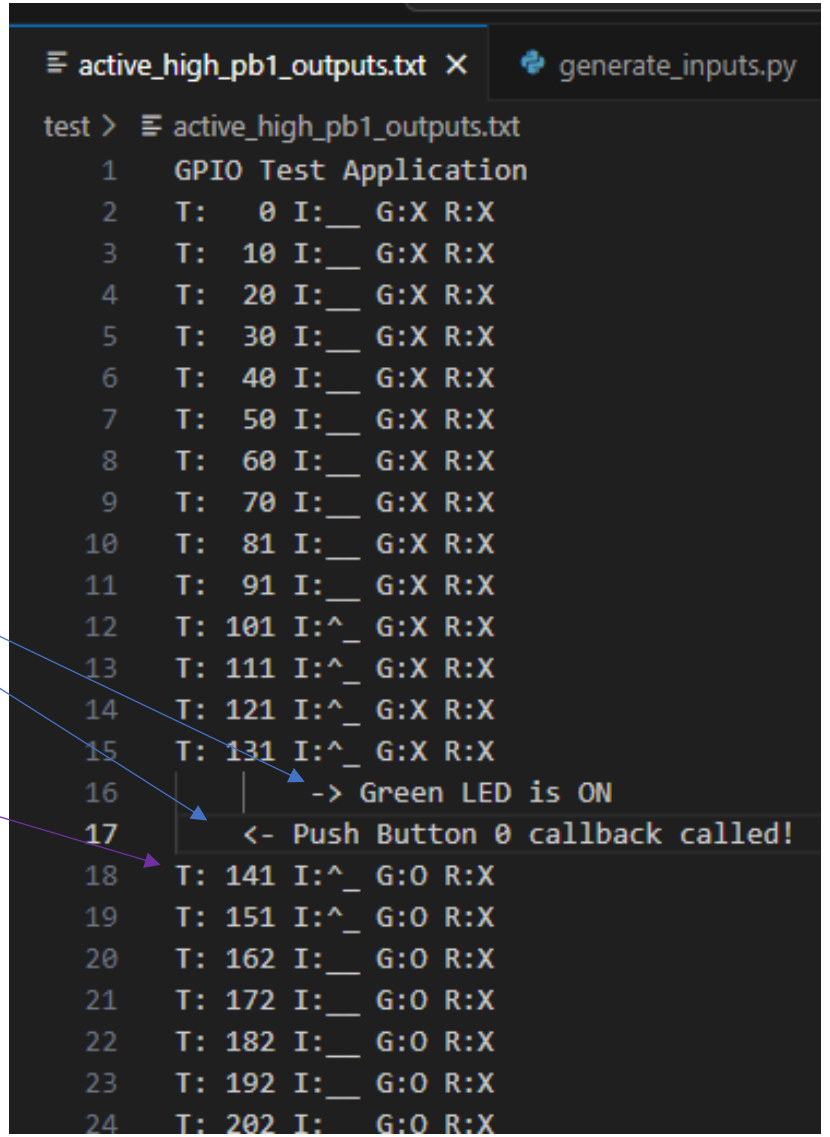
parse_log.py

```
50  def main():
54      if log_string:
59          print(df)
60
61          # Plot DataFrame
62          fig, axs = plt.subplots(4, 1, figsize=(10, 12))
63          axs[0].plot(df['time'], df['pb_1'], label='pb_1', marker='o')
64          axs[0].set_title('pb_1')
65          axs[0].set_ylabel('State')
66
67          axs[1].plot(df['time'], df['green_led'], label='green_led', marker='o', color = 'g')
68          axs[1].set_title('green_led')
69          axs[1].set_ylabel('State')
70
71          axs[2].plot(df['time'], df['pb_2'], label='pb_2', marker='o')
72          axs[2].set_title('pb_2')
73          axs[2].set_ylabel('State')
74
75          axs[3].plot(df['time'], df['red_led'], label='red_led', marker='o', color = 'r')
76          axs[3].set_title('red_led')
77          axs[3].set_ylabel('State')
78          plt.show()
79
```

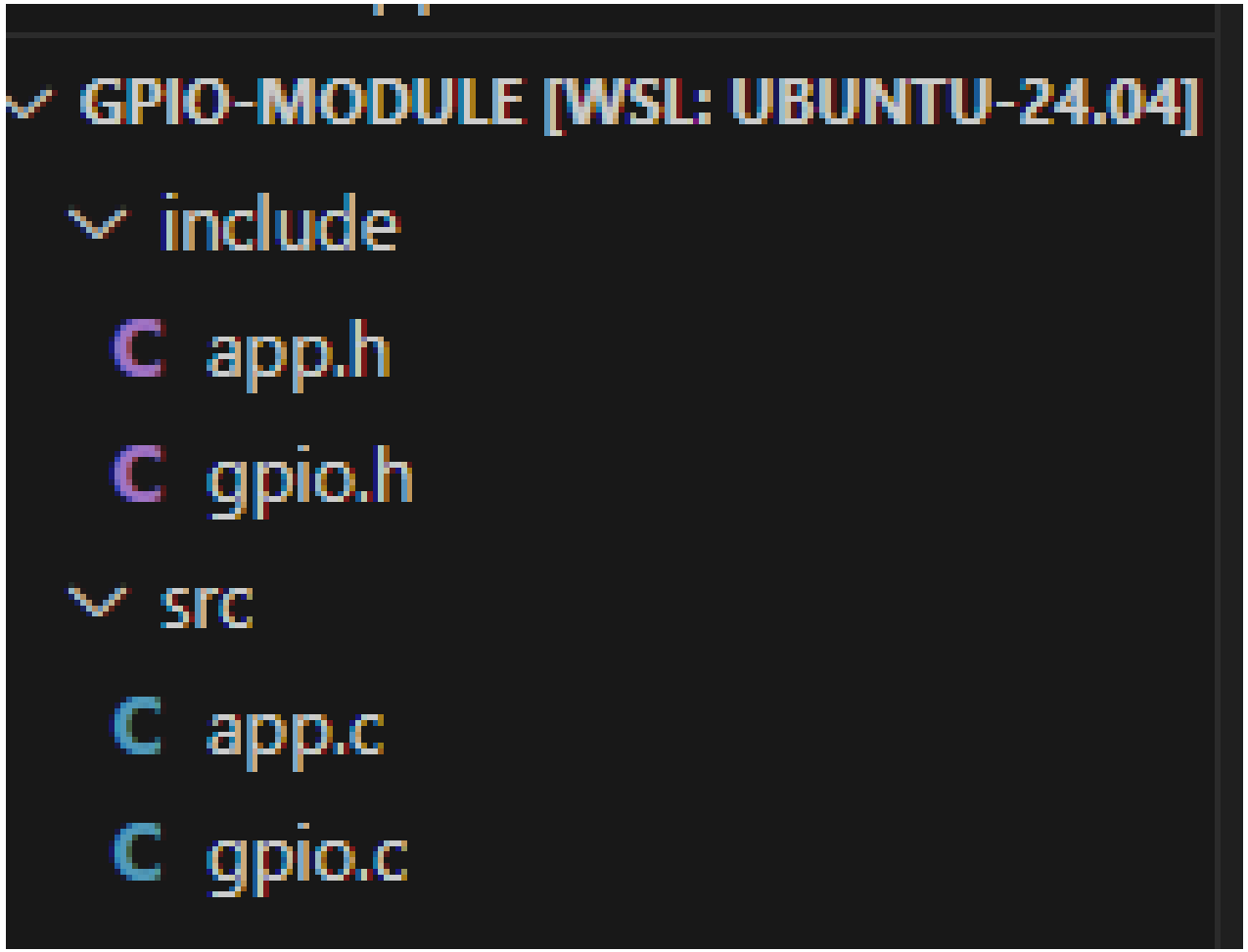
Sample output data

- Output by test code in 10 ms task
- T -> time in milliseconds since program start
- I -> GPIO input status
  - _ indicates LOW and ^ indicates HIGH state
- G & R are GPIO output status for GREEN and RED LESs
  - O indicates ON and X indicates OFF state
- Also shows code execution flow as nested log

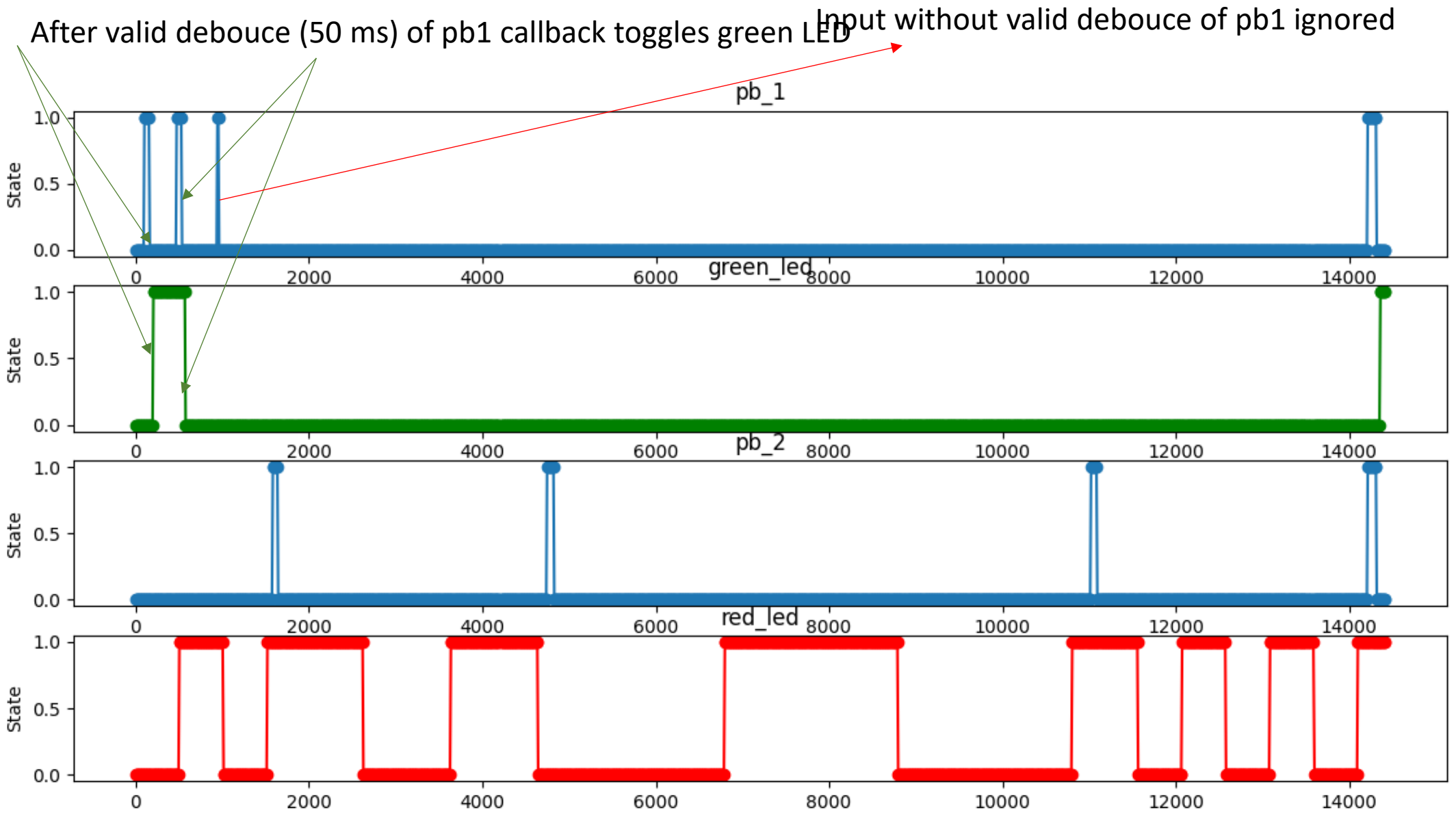
- 50ms debounce, log entry is output after the actual process of inputs



Actual driver code to be used in complex app



Active high PB1



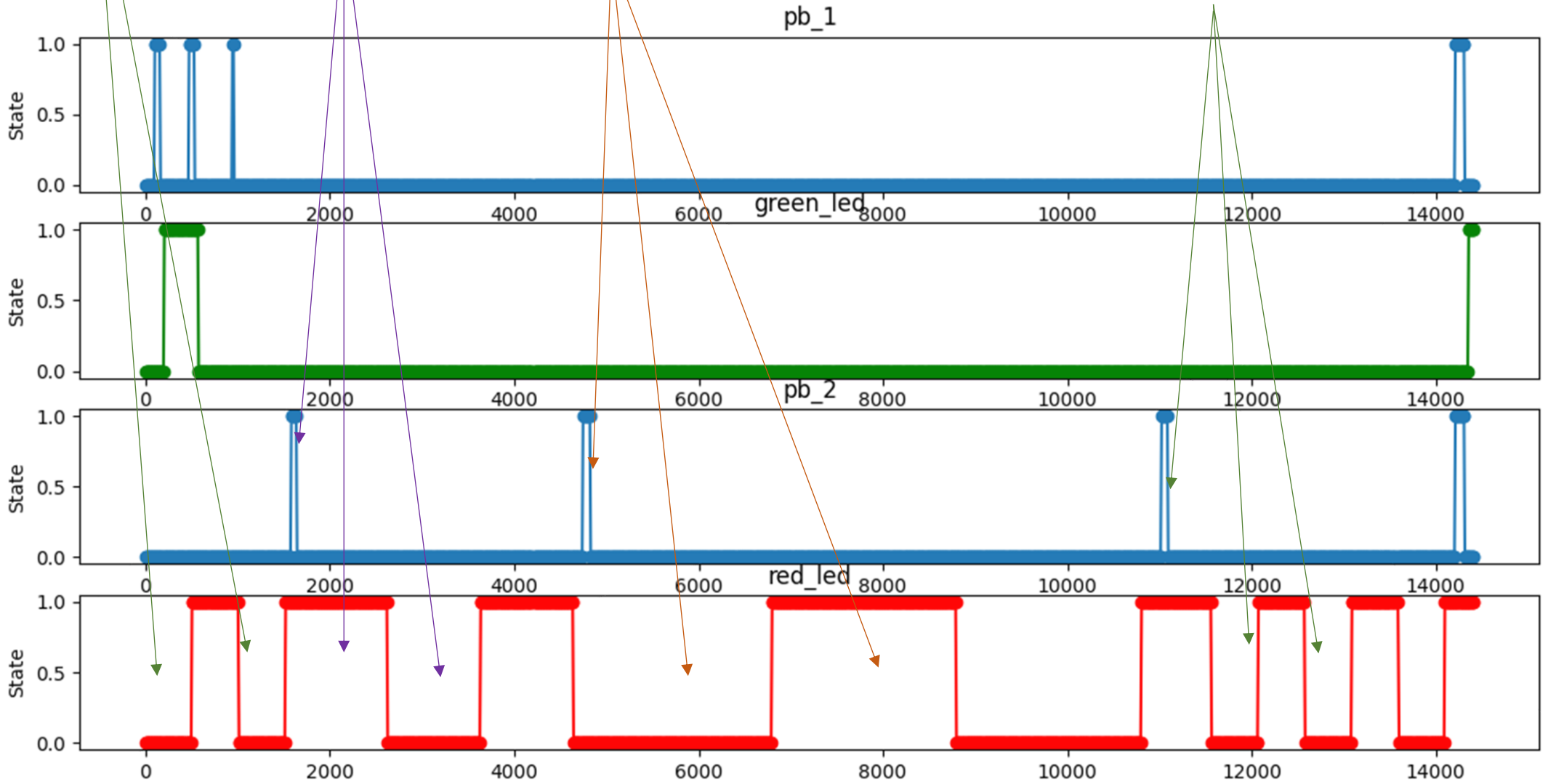
PB2

Before pb2 activation red LED toggles at 500ms interval

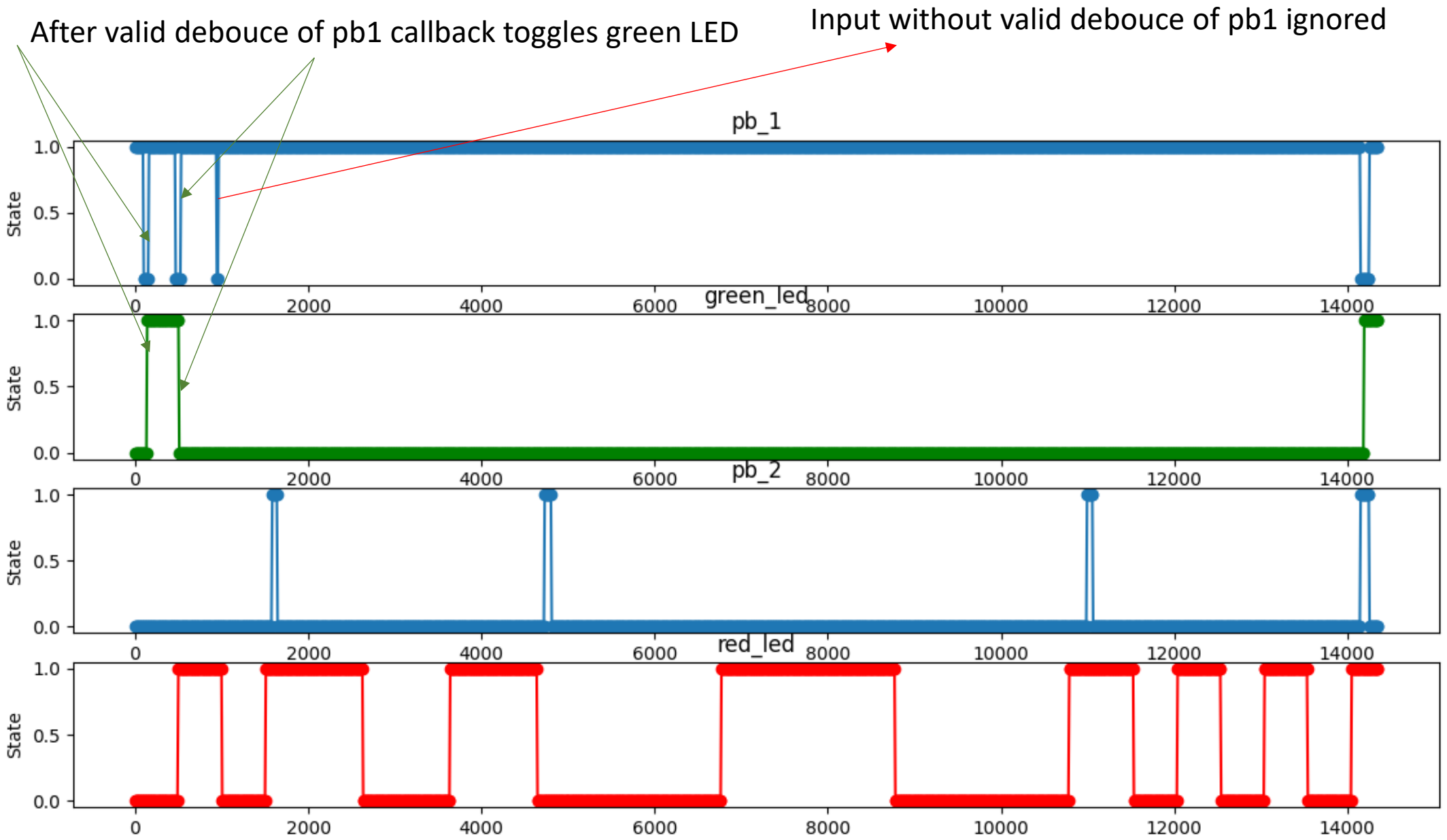
After first pb2 activation red LED toggles at 1000ms interval

After second pb2 activation red LED toggles at 2000ms interval

After third pb2 activation red LED toggles rollback at 500ms interval



Active low PB1



Future improvements

- Currently input are generated programmatically, this can be further enhanced using python behave that would make defining dynamic behaviour much easier
- Output is validated by manual inspection with behave can be validated systematically and validate against system requirement.
- Then this can be integrated to build process (CICD) validating against spec each time code being modified.
- This approach will enhance software system functional reliability considerably, next time no need to debug entire system for a trivial software bug which can be easily capture in the CICD build process itself.