**BioConnect Project – Report Form**

This form must be submitted via Moodle. The submission deadlines are published on Moodle.

**I. Project Work** *(max. 1 page)*

**Project Title**

A Modular Embedded System for Heart Rate and Blood Oxygen Monitoring

**Team Members:** *(min. 1, max 3. Default: 2)*

1. Yiming Liu

2. Yuchen Wang

**Team Goal:** *(Explain what you want to achieve with your project as a team.)*

We aim to develop a flexible embedded system capable of acquiring photoplethysmography (PPG) signals for both heart rate and blood oxygen measurements. Our design allows users to select between two operating modes of the PPG sensor, as well as various preprocessing and data analysis algorithms, thereby meeting different application requirements. Furthermore, we seek to ensure extensibility, so that new sensor control methods, preprocessing algorithms, and data analysis algorithms can be rapidly incorporated in response to evolving project demands.

**II. Your Contribution** *(max. 2-3 pages)*

Note: It is allowed to use content from the advanced exercise and to adapt it to the BioConnect project.

**Abstract:** *(Explain what you want to achieve with your project as a team.)*

My work involves hardware initialization and system architecture design. Specifically, I determined the physical connections between peripherals and the development board, implemented the peripheral configuration code, and designed the finite state machine, button logic, LED indication, GPIO control, ADC acquisition, and UART output modules. The goal is to provide a runtime environment and interface for both preprocessing and advanced processing algorithms, ensuring stable operation of the entire system while fulfilling expandability requirements for future needs.

**Method:** (*How did you solve it? Which methods did you apply? (Add diagrams, drawings, etc, if applicable)*

**1 Device Setup**

The sensor is connected to the development board in the same way as in the reference project.

**GPIO**

In the Pinout & Configuration page, apply the following settings:

- PA0, PA1, PA5

Mode: GPIO\_output

The initialization function generated in MX\_GPIO\_Init() calls HAL\_GPIO\_Init().

In the code, HAL\_GPIO\_WritePin() is used to control the red LED, infrared LED, and the onboard LED.

- PC13

Mode: External Interrupt

Pull-up

EXTI: Rising edge

NVIC: Enable EXTI line [15:10]

The program uses HAL\_GPIO\_EXTI\_Callback() to detect button inputs and relies on debouncing and time measurement to distinguish between “short press” and “long press.”

**ADC**

In Analog/ADC1, enable PA6 (IN11) in single-ended mode. STM32CubeMX generates the MX\_ADC1\_Init() function, which calls HAL\_ADC\_Init() and HAL\_ADC\_ConfigChannel(). The program uses the following interfaces to sample data:

HAL\_ADC\_Start()

HAL\_ADC\_PollForConversion()

HAL\_ADC\_GetValue()

In STATE\_RUNNING, it performs a one-time acquisition to convert the PPG sensor’s analog signal into a digital value.

**UART**

In Connectivity/USART2, set PA2 (TX) and PA3 (RX) to Asynchronous mode. STM32CubeMX generates the MX\_USART2\_UART\_Init() function, which calls HAL\_UART\_Init(). The program outputs information via these interfaces:

HAL\_UART\_Transmit()

sprintf() is used to assemble the strings to be sent.

The output includes the current state, sampled values, and algorithm processing results.

**Clock**

In Clock Configuration, use Resolve Clock Issues to set parameters automatically.

**2 Main Program and Functional Modules**

**Finite State Machine**

In the USER CODE BEGIN/END area, define six states (the FSM workflow is shown in Figure 1 and Table 1). Mode switching is performed based on button events and the current state. The main calls include:

HAL\_GPIO\_EXTI\_Callback() to set short-press/long-press flags

HAL\_GPIO\_WritePin() to control the output pins

A function pointer table to manage preprocessing and advanced processing algorithms

**LED Indication**

Use the updateLED(g\_systemState) function to configure the output on PA5, controlling the LED’s blinking rate (the blinking rate corresponding to each state is in Table 2). For example:

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_RESET)

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_5, GPIO\_PIN\_SET)

**ADC Acquisition and UART Output**

In STATE\_RUNNING, the program performs a single ADC polling and uses a preprocessing function (such as applySquaringFilter()), followed by an advanced processing function (such as computeHeartRate()), then outputs the result to the terminal using HAL\_UART\_Transmit().

**Error Handling**

If any peripheral initialization fails or if an exception occurs in the state machine, enterErrorState() is called to switch to STATE\_ERROR.

**Results:** *(Demonstrate the outcome of your contribution with screenshots, photos, etc.)*

A screenshot of a computer

Description automatically generated

**III. Appendix** *(copy&Paste of your code parts, electronic schematics, drawings, etc.)*

<code>

<schematics>

<drawings>