Título:	Pulsioxímetro

Ciclo Lectivo **2024**	Curso:	**R2004**	Grupo:	**5**

Integrantes	Apellido Nombres	Legajo	Calificación individual	Fecha
Costarelli, Facundo		1762916		
Lagache, Ezequiel		2086669		

Calificación grupal:		Fecha:	12/12/2024

Profesor:	Ing. Prieto Canalejo, Mariana Andrea
Auxiliar/es Docente:	Ing. Escola, Jorge Ignacio

Observaciones primera entrega	
Observaciones segunda entrega	

ÍNDICE

INTRODUCCIÓN.....3
OBJETIVOS..... 3
DIAGRAMAS EN BLOQUES.....4
DESARROLLO.....5
DESCRIPCIÓN DEL HARDWARE Y SOFTWARE..... 5
Diagramas de Bloques.....5
Herramientas y equipos.....6
Especificaciones y Limitaciones del Proyecto.....6
Circuito y PCB..... 7
Programa principal y drivers.....8
Aplicación QT.....13
Hojas de datos..... 15
PROBLEMAS ENCONTRADOS EN EL DESARROLLO DEL PROYECTO.....16
BENEFICIOS ENCONTRADOS EN EL DESARROLLO DEL PROYECTO..... 18
CONCLUSIÓN.....19
ANEXO..... 20

INTRODUCCIÓN

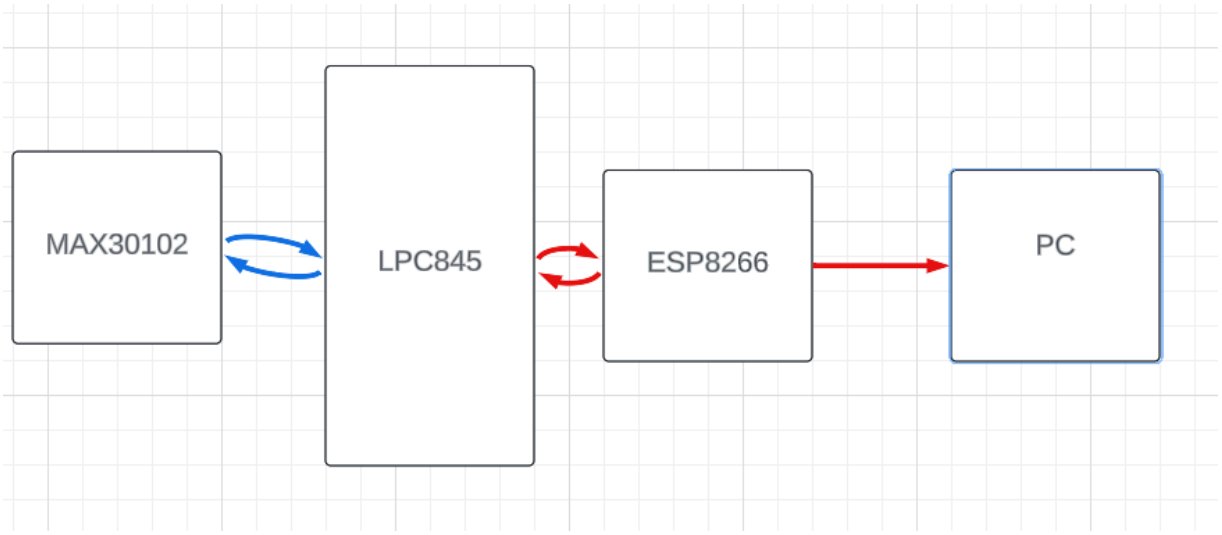
En este proyecto se busca desarrollar un dispositivo pulsioxímetro prototipo en términos de software y un hardware básico y sencillo para aplicar el software. Para lo primero, se realizará una aplicación con sus drivers, de manera tal que mediante una interfaz gráfica se pueda visualizar en tiempo real los valores medidos de saturación de oxígeno en sangre (SPO2) y de frecuencia cardíaca (Heart Rate), obtenidos mediante un módulo sensor MAX 30102. La comunicación entre el sensor y la interfaz se realiza a través de un módulo ESP8266, utilizando el protocolo TCP de WiFi para transmitir los datos hacia una PC portable. El control de los módulos es dirigido por el microcontrolador LCP 845. Para lo segundo, se pretende combinar módulos programables en un único PCB.

Este proyecto integra múltiples conceptos de programación orientada a objetos, diseño de interfaces gráficas, comunicación inalámbrica, manejo de protocolos de comunicación intermodulares como el I2C, desarrollo esquemático con PCB y conexionado de módulos y componentes necesarios de hardware. El proyecto está orientado a aplicaciones biomédicas accesibles y eficientes.

OBJETIVOS

- Estudio y aplicación sobre conocimientos del stick LPC845.
- Incorporar y afianzar la teoría de C++, conceptos de OOP, Sistemas Embebidos y Análisis de Señales y Sistemas.
- Unir conceptos de aplicación con los conceptos de drivers, ambos desarrollados para el microcontrolador y módulos.
- Establecer comunicación entre módulos programables y el microcontrolador con protocolo I2C, así como comunicación serie.
- Establecer conexión inalámbrica TCP/IP entre una aplicación de interfaz de QT y la aplicación del microcontrolador junto a sus drivers.
- Incorporar conceptos de Internet Of Things.
- Implementar conceptos de desarrollo de hardware de forma básica utilizando KiCad.
- Introducir y estudiar conceptos de bioingeniería básicos como la medición de SPO2 y HeartBeat.

DIAGRAMAS EN BLOQUES

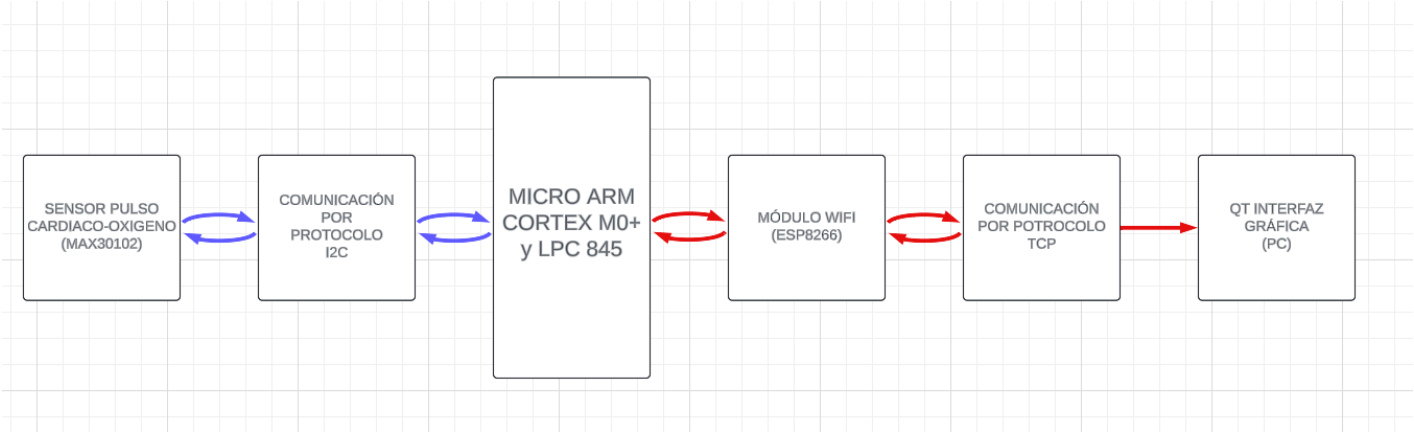


Con este sencillo diagrama podemos ver que el microcontrolador LPC845 se comunica con el sensor MAX 30102 para pedir los datos de SPO2 y Heart Rate censados. Estos datos son procesados con un algoritmo y enviados posteriormente hacia el ESP8266 quien se encarga de transmitir la información por WiFi hacia la PC. Dicho dispositivo los recibe y los procesa nuevamente de tal forma de poder mostrarlos en una interfaz gráfica con animaciones sencillas.

DESARROLLO

DESCRIPCIÓN DEL HARDWARE Y SOFTWARE

Diagramas de Bloques



El sensor MAX 30102 emite ondas electromagnéticas con longitudes de onda tal que una de ellas es visible de color rojo mientras que la otra no es visible y es de carácter infrarrojo. Dichas ondas chocan contra las moléculas presentes en la sangre de un dedo, reflejándose de forma tal que son capturadas por un fotodiodo. Los cambios producidos en las ondas emitidas respecto de las reflejadas dependen del ritmo cardíaco y de la oxigenación presente en la sangre. Para obtener los valores de SPO2 y Heart Rate, se realizan cálculos, procesos de filtrado de señal y conversión analógica digital ADC, donde todo es realizado por el hardware propio del módulo del sensor permitiendo que a la salida del mismo podamos leer un tren de pulsos.

La comunicación con dicho sensor es por protocolo I2C tal que el tren de pulsos se corresponde con esta codificación. La comunicación es establecida entre el microcontrolador en modo Maestro mientras que el sensor funciona como Esclavo. Se realiza desde el microcontrolador una configuración inicial del sensor a través de modificar sus registros para lograr las funcionalidades mencionadas. Se implementan también métodos de lectura de datos del sensor.

Por su parte, los datos crudos de SPO2 Y Heart Rate son procesados por el microcontrolador LPC845 a través de un algoritmo sugerido por el fabricante del sensor Maxim. Con los datos obtenidos, se realiza un segundo filtrado mediante un promedio ponderado, a fin de lograr mayor fiabilidad en la medición. Los datos procesados son entonces enviados hacia el módulo WiFi ESP8266.

El módulo ESP8266 es configurado inicialmente a través de comandos AT sin entrar estrictamente en la modificación de sus registros. Luego es establecida una comunicación con la PC por protocolo TCP

utilizando comandos AT, mediante la zona WiFi que emite el dispositivo móvil. Por otro lado, la PC actúa como el servidor, y el módulo ESP8266 como cliente. Con la comunicación establecida, se envían los datos de SPO2 y Heart Rate hacia la PC, la cual los recibe y procesa con una interfaz gráfica realizada previamente en QT.

Herramientas y equipos

- Sensor pulso cardíaco y saturación de oxígeno MAX 30102.
- Módulo WiFi ESP8266.
- Microcontrolador LPC845.
- Módulo DC-DC Step Down Mini-360 Convertidor Fuente 3A 0,8V - 20V
- Placa PCB 10cm largo y 3cm ancho, 1 diodo 1n4148, 1 capacitor electrolítico 100uF 50V, tira de pines hembra y gabinete de 12x7x5,5 cm.
- Fuente de alimentación de pila de 9V DC.
- PC o Notebook para interfaz gráfica de QT.

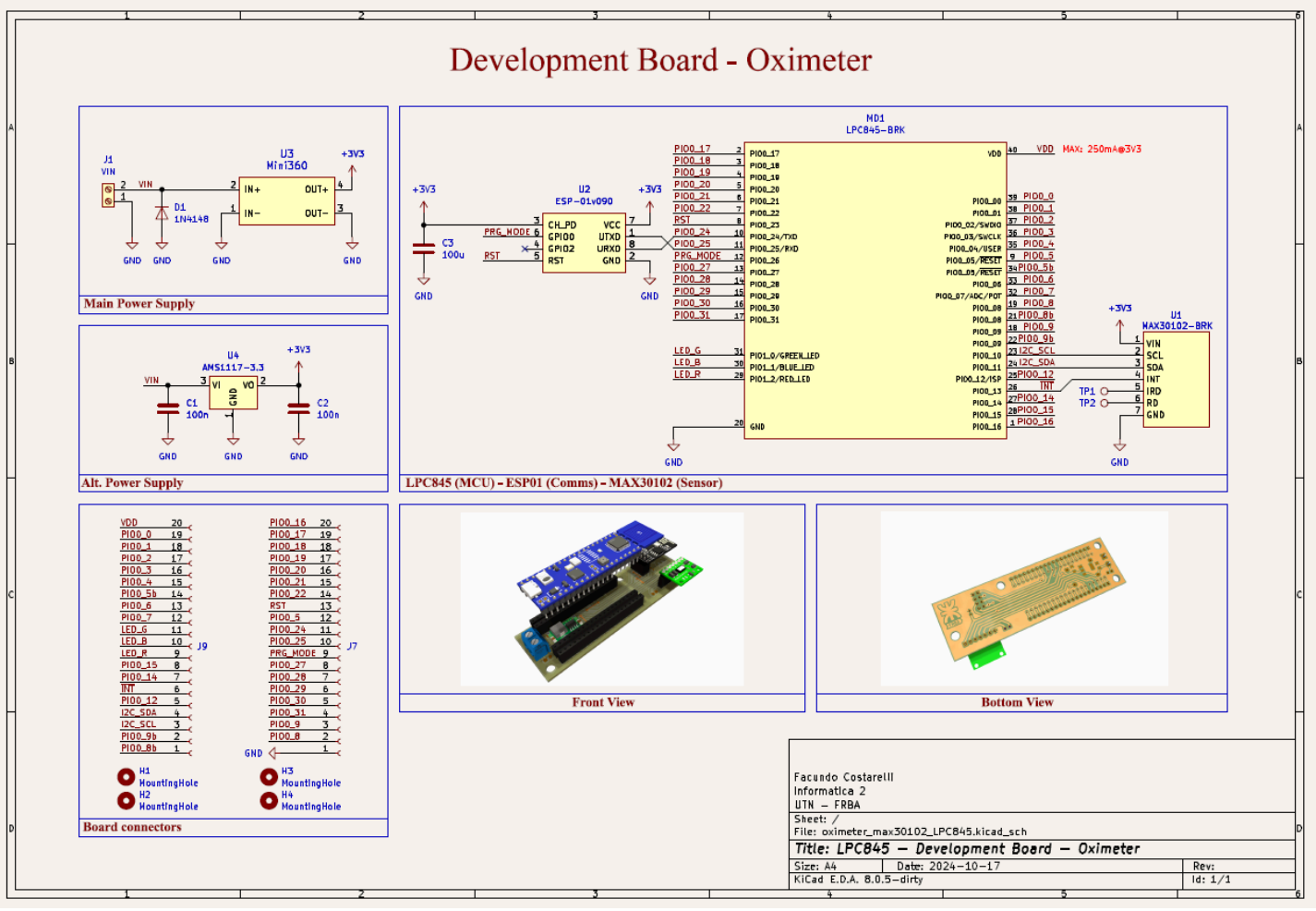
Especificaciones y Limitaciones del Proyecto

- Alimentación de entrada de todos los módulos: 3,3V.
- Consumo habitual de corriente del proyecto: 150-200mA.
- La eficiencia del regulador DC-DC es de ~75%
- Duración máxima estimada con batería de 9V: ~2-3 horas bajo carga típica.
- Alimentación separada para los módulos con regulador y para el LPC845 por puerto USB.
- Diseño modular con conexión entre MAX 30102, ESP8266 y LPC845.
- Alimentación principal basada en regulador AMS1117 y Mini360.
- La aparición de picos de corrientes combinados del ESP8266 y el sensor pueden aumentar la temperatura en el regulador y del microcontrolador. No se puede conectar el pin de VCC del LPC845 con el pin de VCC del ESP8266 dado que este último consume típicamente más de 200 mA, superando la disponibilidad de corriente que puede ofrecer el microcontrolador. Las alimentaciones deben ser separadas.
- Protocolo de comunicación I2C: Inestabilidad y ruido frente al cableado para la comunicación. Es necesario diseñar pistas de cobre cortas y/o blindaje adecuado para minimizar interferencias. Velocidad máxima de 400 KHz de transferencia de datos. Se requiere de un software robusto para el manejo correcto de todo.
- Protocolo de comunicación UART: Baud rate recomendado 9600 o 115200 bps para comunicación confiable. El uso del regulador DC-DC puede introducir ruido que puede interferir en RX/TX. Los

buffers son pequeños por lo que requieren ser vaciados o refrescados rápidamente. Por ser un protocolo serial, tiene una latencia intrínseca debido a la transmisión secuencial de bits. Solo permite comunicación directa entre dos nodos. Para múltiples dispositivos, se necesitan configuraciones adicionales como UART-to-Multiplexers o conversores. Se requiere de un software robusto para el manejo correcto de todo.

Circuito y PCB

A partir de la siguiente imágen vemos el plano del circuito esquemático del dispositivo así como su modelado 3D y su placa PCB con los módulos:



Para el desarrollo del dispositivo utilizamos el software KiCad, con la intención de eliminar los problemas de falso contacto en pines de conexión que teníamos en un protoboard, y reducir el uso de cables innecesarios. De esta forma, todos los módulos están fijos y colocados en pines ofreciendo la posibilidad de reemplazar por módulos iguales de repuesto en caso de que se rompa alguno. Además las pistas son lo más

cortas posibles en anchura y longitud lo que principalmente permite reducir significativamente el ruido que se introduce en la comunicación I2C, TCP y UART.

La entrada de alimentación está dada por la bornera azul permitiendo una tensión DC de entrada de hasta 20V. Ya que esta se conecta a un regulador DC-DC Step Down que se encuentra justo debajo del LPC845, podemos tener portabilidad de hardware, permitiendo que la entrada se alimente con baterías de distintas tensiones, como por ejemplo de 9V. La salida del regulador se ajusta con un tornillo ubicado en la placa del regulador. Lo dejamos configurado para que tenga 3,3 V de salida. Su corriente de salida puede ser de hasta 3A según la demanda de los dispositivos.

Es importante destacar que en un dispositivo Step Down, la entrada debe ser de tensión superior a la prevista en la salida. Por lo que, en este caso, no podríamos ingresar con una tensión menor a 3,3V. Las pruebas fueron realizadas con baterías de 9V de entrada y fuentes de laboratorio a 5V de entrada donde se evaluaron resultados positivos.

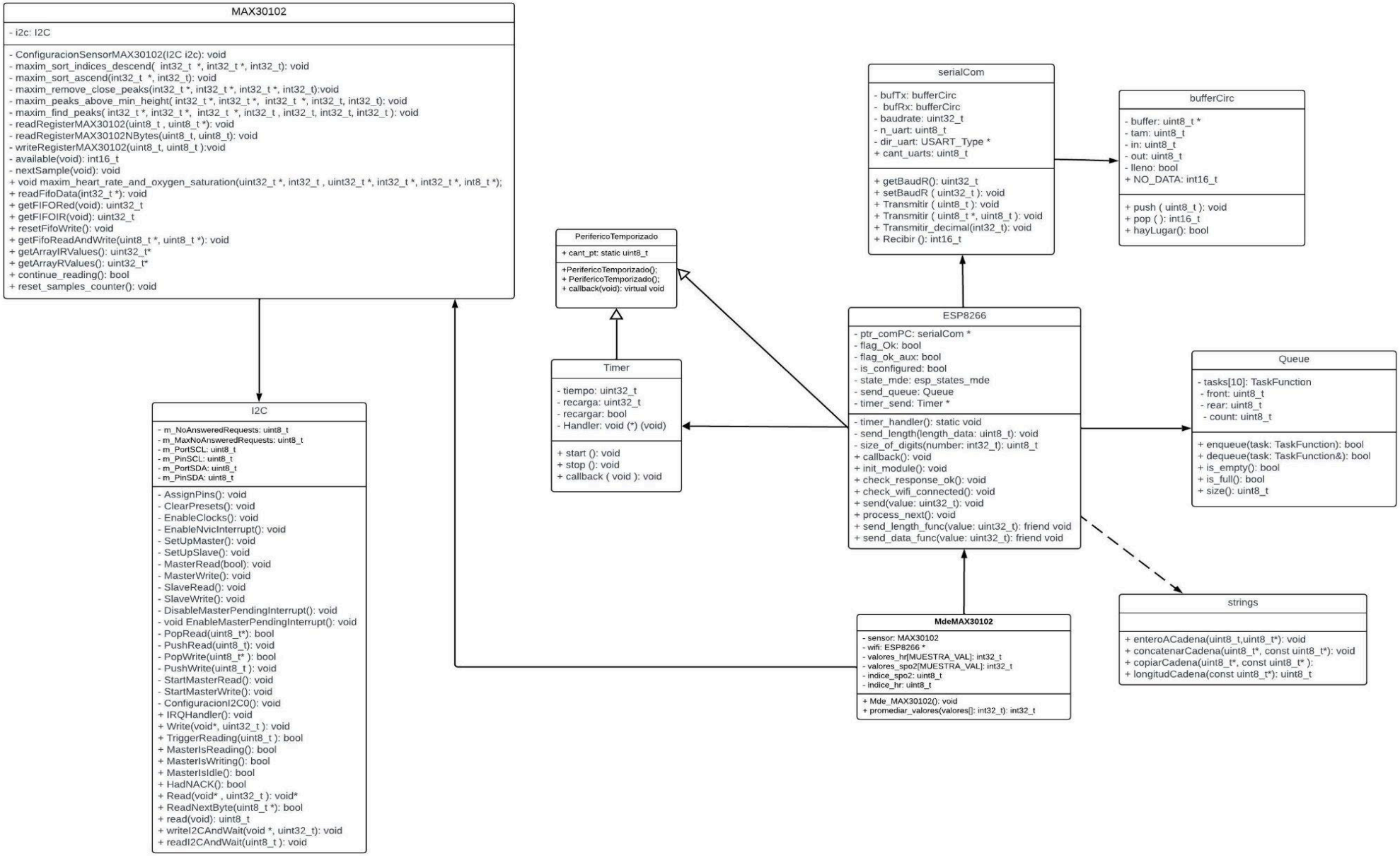
Por otro lado, destacamos que en este diseño la tensión de salida del regulador afecta solamente al módulo de WiFi y el sensor, mientras que para alimentar el módulo del microcontrolador es necesario conectarlo a una PC con una cable mini usb-usb C. La necesidad de tener alimentaciones separadas se debe a que si todos los módulos quedan conectados entre sí a través de sus pines VCC a 3,3V, unidos dichos pines a su vez con la Vout del regulador, ocurre que el módulo WiFi demanda mucha corriente durante su funcionamiento, exigiendo al microcontrolador cada vez más corriente que pueden superar los 200 mA máximos que pueden ofrecer el microcontrolador. Esto se traduce en sobrecalentamiento del módulo WiFi y del módulo microcontrolador hasta romperse. Para detectar esta situación hemos colocado post-diseño y post-construcción de la placa, un par de pines como “jumper” que unifica los VCC entre sí con el pin de Vout del regulador. En la sección de Anexo se mencionan posibles mejoras al respecto.

Finalmente, podemos ver un diodo rápido 1N4148 colocado a la entrada con la bornera justo antes de llegar al regulador. Esto es para protección contra corto circuito provocado por una posible inversión de los cables de alimentación en la bornera. También, dado que el módulo WiFi ESP8266 produce picos de corriente durante su funcionamiento, hemos colocado un capacitor electrolítico de 100uF de 50V como protección para absorber dichas corrientes y evitar problemas en el resto del circuito.

Programa principal y drivers

El programa principal se compone de un grupo de líneas de código que ejecutan la inicialización del módulo LPC845, módulo UART de serialCom, módulo ESP8266 y el módulo del sensor; y de un while(1) que ejecuta repetitivamente la Máquina de Estados (MDE) de la aplicación. Esto involucra la interacción de todas los módulos. El código se puede observar en el proyecto.

A continuación se presenta un diagrama de clases que relaciona todos los drivers de los módulos así como las utilidades implementadas para lograr el control de todo los módulos. Se pretende observar sin demasiado detalle cual es la estructura general de nuestro sistema. El link se adjunta en la sección de Anexo.


MAX30102
- i2c: I2C
- ConfiguracionSensorMAX30102(I2C i2c): void
- maxim_sort_indices_descend(int32_t *, int32_t *, int32_t): void
- maxim_sort_ascend(int32_t *, int32_t): void
- maxim_remove_close_peaks(int32_t *, int32_t *, int32_t *, int32_t):void
- maxim_peaks_above_min_height(int32_t *, int32_t *, int32_t *, int32_t, int32_t): void
- maxim_find_peaks(int32_t *, int32_t *, int32_t *, int32_t , int32_t, int32_t, int32_t): void
- readRegisterMAX30102(uint8_t , uint8_t *): void
- readRegisterMAX30102NBytes(uint8_t, uint8_t): void
- writeRegisterMAX30102(uint8_t, uint8_t):void
- available(void): int16_t
- nextSample(void): void
+ void maxim_heart_rate_and_oxygen_saturation(uint32_t *, int32_t , uint32_t *, int32_t *, int32_t *, int8_t *);
+ readFifoData(int32_t *): void
+ getFIFORed(void): uint32_t
+ getFIFOIR(void): uint32_t
+ resetFifoWrite(): void
+ getFifoReadAndWrite(uint8_t *, uint8_t *): void
+ getArrayIRValues(): uint32_t*
+ getArrayRValues(): uint32_t*
+ continue_reading(): bool
+ reset_samples_counter(): void
serialCom
- bufTx: bufferCirc
- bufRx: bufferCirc
- baudrate: uint32_t
- n_uart: uint8_t
- dir_uart: USART_Type *
+ cant_uarts: uint8_t
+ getBaudR(): uint32_t
+ setBaudR (uint32_t): void
+ Transmitir (uint8_t): void
+ Transmitir (uint8_t *, uint8_t): void
+ Transmitir_decimal(int32_t): void
+ Recibir (): int16_t
bufferCirc
- buffer: uint8_t *
- tam: uint8_t
- in: uint8_t
- out: uint8_t
- lleno: bool
+ NO_DATA: int16_t
+ push (uint8_t): void
+ pop ( ): int16_t
+ hayLugar(): bool
PerifericoTemporizado
+ cant_pt: static uint8_t
+PerifericoTemporizado();
+ PerifericoTemporizado();
+ callback(void): virtual void
Timer
- tiempo: uint32_t
- recarga: uint32_t
- recargar: bool
- Handler: void (*) (void)
+ start (): void
+ stop (): void
+ callback (void): void
ESP8266
- ptr_comPC: serialCom *
- flag_Ok: bool
- flag_ok_aux: bool
- is_configured: bool
- state_mde: esp_states_mde
- send_queue: Queue
- timer_send: Timer *
- timer_handler(): static void
- send_length(length_data: uint8_t): void
- size_of_digits(number: int32_t): uint8_t
+ callback(): void
+ init_module(): void
+ check_response_ok(): void
+ check_wifi_connected(): void
+ send(value: uint32_t): void
+ process_next(): void
+ send_length_func(value: uint32_t): friend void
+ send_data_func(value: uint32_t): friend void
Queue
- tasks[10]: TaskFunction
- front: uint8_t
- rear: uint8_t
- count: uint8_t
+ enqueue(task: TaskFunction): bool
+ dequeue(task: TaskFunction&): bool
+ is_empty(): bool
+ is_full(): bool
+ size(): uint8_t
I2C
- m_NoAnsweredRequests: uint8_t
- m_MaxNoAnsweredRequests: uint8_t
- m_PortSCL: uint8_t
- m_PinSCL: uint8_t
- m_PortSDA: uint8_t
- m_PinSDA: uint8_t
- AssignPins(): void
- ClearPresets(): void
- EnableClocks(): void
- EnableNvicInterrupt(): void
- SetUpMaster(): void
- SetUpSlave(): void
- MasterRead(bool): void
- MasterWrite(): void
- SlaveRead(): void
- SlaveWrite(): void
- DisableMasterPendingInterrupt(): void
- void EnableMasterPendingInterrupt(): void
- PopRead(uint8_t*): bool
- PushRead(uint8_t): void
- PopWrite(uint8_t*): bool
- PushWrite(uint8_t): void
- StartMasterRead(): void
- StartMasterWrite(): void
- ConfiguracionI2C0(): void
+ IRQHandler(): void
+ Write(void*, uint32_t): void
+ TriggerReading(uint8_t): bool
+ MasterIsReading(): bool
+ MasterIsWriting(): bool
+ MasterIsIdle(): bool
+ HadNACK(): bool
+ Read(void* , uint32_t): void*
+ ReadNextByte(uint8_t *): bool
+ read(void): uint8_t
+ writeI2CAndWait(void *, uint32_t): void
+ readI2CAndWait(uint8_t): void
MdeMAX30102
- sensor: MAX30102
- wifi: ESP8266 *
- valores_hr[MUESTRA_VAL]: int32_t
- valores_spo2[MUESTRA_VAL]: int32_t
- indice_spo2: uint8_t
- indice_hr: uint8_t
+ Mde_MAX30102(): void
+ promediar_valores(valores[]: int32_t): int32_t
strings
+ enteroACadena(uint8_t,uint8_t*): void
+ concatenarCadena(uint8_t*, const uint8_t*): void
+ copiarCadena(uint8_t*, const uint8_t*):
+ longitudCadena(const uint8_t*): uint8_t

Explicamos el funcionamiento del sistema, el diagrama de clases y de los protocolos de comunicación utilizados: I2C, TCP, UART.

El sistema está diseñado de manera tal que primero se realiza la inicialización del módulo microcontrolador mediante la función InicializarPLL(), luego se instancia e inicializa un objeto tipo serialCom y uno tipo ESP8266 para el módulo WiFi. Posteriormente se realiza la configuración del módulo WiFi a partir de un ciclo while. Este while es bloqueante pero no afecta negativamente en el proyecto dado que forma parte del bloque de inicialización del sistema. Inmediatamente después se instancia e inicializa el módulo del sensor y finalmente se ejecuta un while(1) con una máquina de estado que controla la aplicación. Interesa estudiar en detalle cómo es que se configuran y se utilizan los módulos del sensor y WiFi en la aplicación.

Por un lado, se establece una comunicación entre el módulo del microcontrolador LPC845 y el módulo WiFi ESP8266 a partir de utilizar comandos AT. En concreto, para configurar el WiFi se ejecuta una pequeña máquina de estados dentro de un ciclo while, dado como while(!wifi.check_wifi_connected()){ wifi.init_module();}. Una vez establecida la conexión, se utiliza un grupo de métodos para lograr la transmisión de los datos obtenidos por el sensor hacia una PC, a través de un sistema de encolamiento tipo FIFO. No estudiamos la recepción de datos provenientes desde la PC en este proyecto dado que no es necesario. Es importante notar que la comunicación es por protocolo TCP, por lo tanto, primero debemos enviar el tamaño del mensaje o dato que queremos transmitir expresado en cantidad de bytes, y luego debemos enviar el mensaje propiamente dicho. Por cada comando enviado al módulo de WiFi, éste responde según corresponda con diferentes strings que poseen la palabra clave “OK\r\n”. Es importante poder recibir dicho “OK\r\n” por cada comando AT ejecutado dado que es la forma de saber en qué etapa se encuentra el módulo WiFi y si está funcionando todo correctamente. Para lograr la recepción correcta, usamos una máquina de estado que utiliza puerto serie y protocolo UART, que luego será explicada.

Por otro lado, se establece una comunicación entre el módulo del microcontrolador LPC845 y el módulo del sensor MAX 30102 a partir de utilizar protocolo I2C. En particular, para configurar el sensor se deben enviar comandos formados por la dirección de un registro interno a modificar y del dato a cargar en ese registro, ambos en formato hexadecimal. Primero se debe enviar la dirección y luego el dato pensado como un grupo de bits que deben ser colocados en ese registro con operadores de nivel de bit. En este modo, el microcontrolador se encuentra en modo Maestro, mientras que el sensor en modo Esclavo. Esta configuración define la resolución y frecuencia del muestreo, qué datos vamos a sensar y, en base a eso, qué leds encender, entre otras cosas. Una vez realizada la configuración, podremos ver el encendido de una luz led roja e infrarroja para medir el nivel de oxigenación en la sangre (SPO2) y la frecuencia cardíaca (Heart Beat). Los datos analógicos son capturados por el sensor por fotopleigtismo, filtrados y convertidos con ADC, y guardados como muestras en espacios de memoria administrados por punteros denominados FIFO write and FIFO read. Todo esto último mencionado lo hace el hardware propio del módulo del sensor. El algoritmo de procesamiento de los datos crudos es el utilizado por el fabricante Maxim, no es tan preciso y exacto pero arroja mediciones interesantes.

Finalmente, la aplicación en sí misma es controlada por una máquina de estados, basada en las hojas de datos del sensor donde se establecen dos transiciones. En la primera se le pide al sensor indicar el número

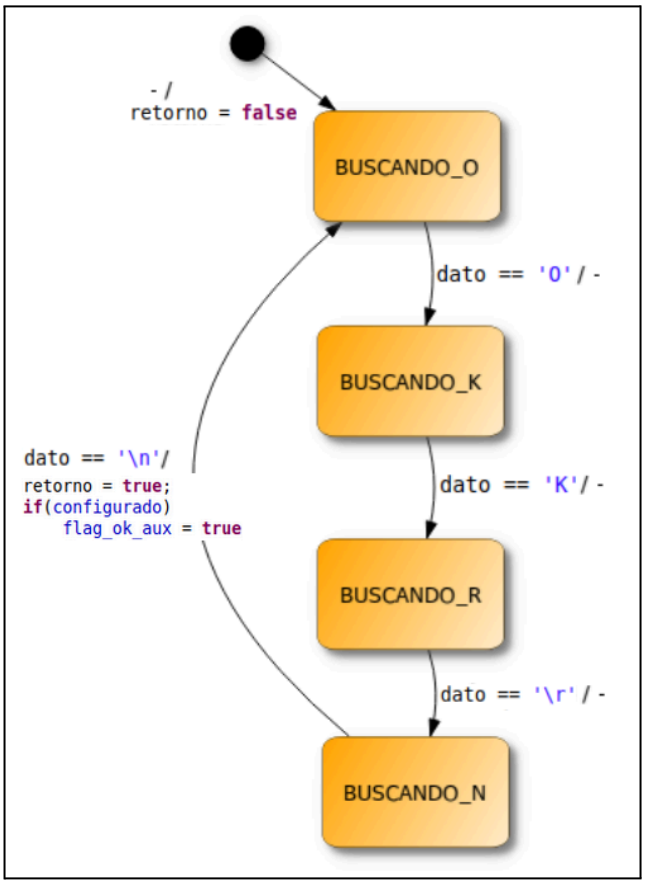
de muestras disponibles según la dirección a donde apuntan los punteros FIFO write and read; y en la segunda se leen las muestras y se les aplica un algoritmo de procesamiento de datos para obtener un valor de SPO2 y de Heart Rate para así transmitirlos por WiFi, ser recibidos en la PC y procesados nuevamente por el programa de la interfaz gráfica del QT.

Analizamos ahora las distintas máquinas de estado de nuestro proyecto.

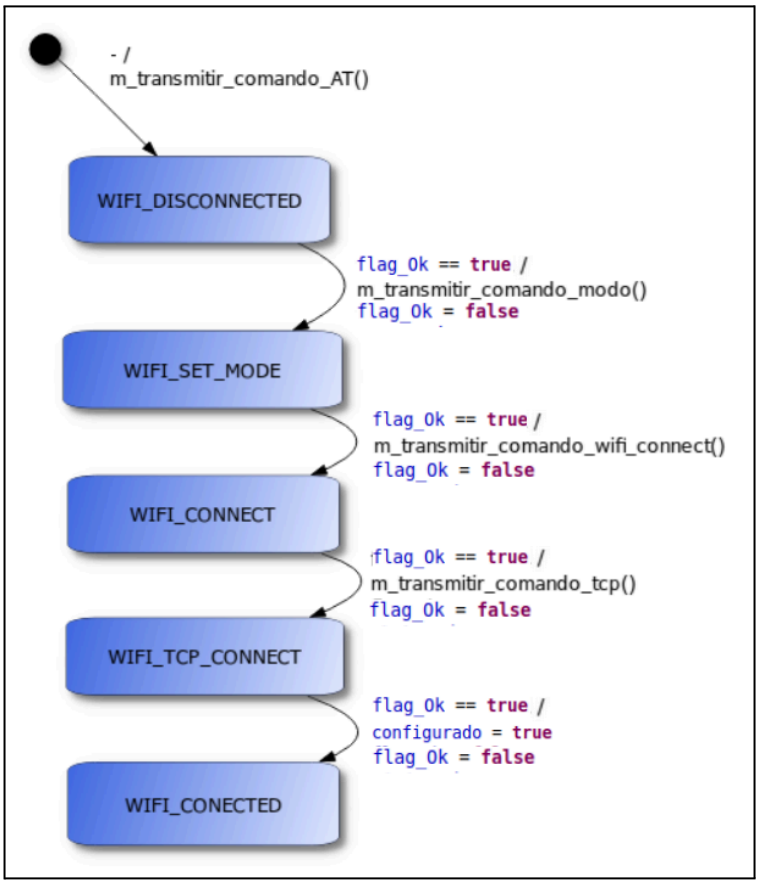
En la primera máquina de estados, en color naranja, podemos observar la lógica para controlar la llegada del string "OK\r\n" que es enviado por el módulo de WiFi ESP8266 en cada una de las etapas de funcionamiento. Esto es tanto en la configuración como en la transmisión y recepción. La máquina de estados recibe un dato y pregunta por su valor en código ASCII. Cuando el dato recibido es igual a 'O', 'K', o '\r', pasa al siguiente estado. Si, por el contrario, el dato recibido es igual a '\n', lo que significa que llegaron todos los caracteres, devuelve un booleano true, coloca en true un flag auxiliar que usaremos para la transmisión y se resetea a su estado inicial, es decir, al estado donde espera la 'O'. En cualquier otro caso se retorna false.

En la segunda máquina de estados, en color azul, observamos la lógica que permite la configuración del módulo WiFi y su conexión con la PC, que como se mencionó anteriormente, actuará de servidor. Consta de cinco estados y, al pasar de uno a otro, el módulo transmite los distintos comandos necesarios para establecer la conexión. La condición para pasar de un estado a otro es, justamente, que el booleano que indica si se recibió el string "OK\r\n" esté en true. Cuando llega al estado WIFI_CONNECTED la conexión está establecida, por lo que no tendrá que volver a ningún otro estado.

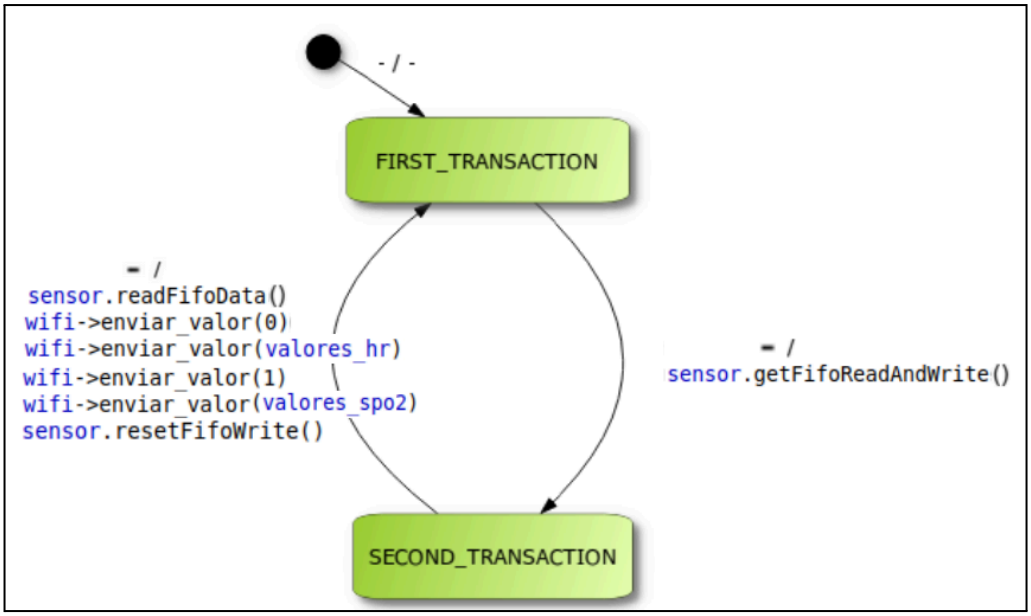
La tercera máquina de estados del proyecto, en color verde, es la máquina de estados que controla la aplicación. Consta de dos estados, el primero llamado FIRST_TRANSACTION, en donde se procesan y almacenan las distintas mediciones del sensor, y el segundo, SECOND_TRANSACTION, que lee la información recolectada y la envía a la PC. Antes de enviar la información, realiza un promedio ponderado, de manera que ésta sea más confiable y no se vea tan afectada por basura o errores en alguna medición. Para enviar la información, primero envía un header (0 o 1) para que la interfaz de Qt sepa de qué medición se trata (si es la frecuencia cardíaca o en nivel de oxigenación en sangre), y a continuación envía el valor de la medición. Luego de enviar las dos mediciones, limpia el sensor y lo deja preparado para que vuelva a reunir mediciones en el estado FIRST_TRANSACTION.



Máquina de Estados I



Máquina de Estados II



Máquina de Estados III

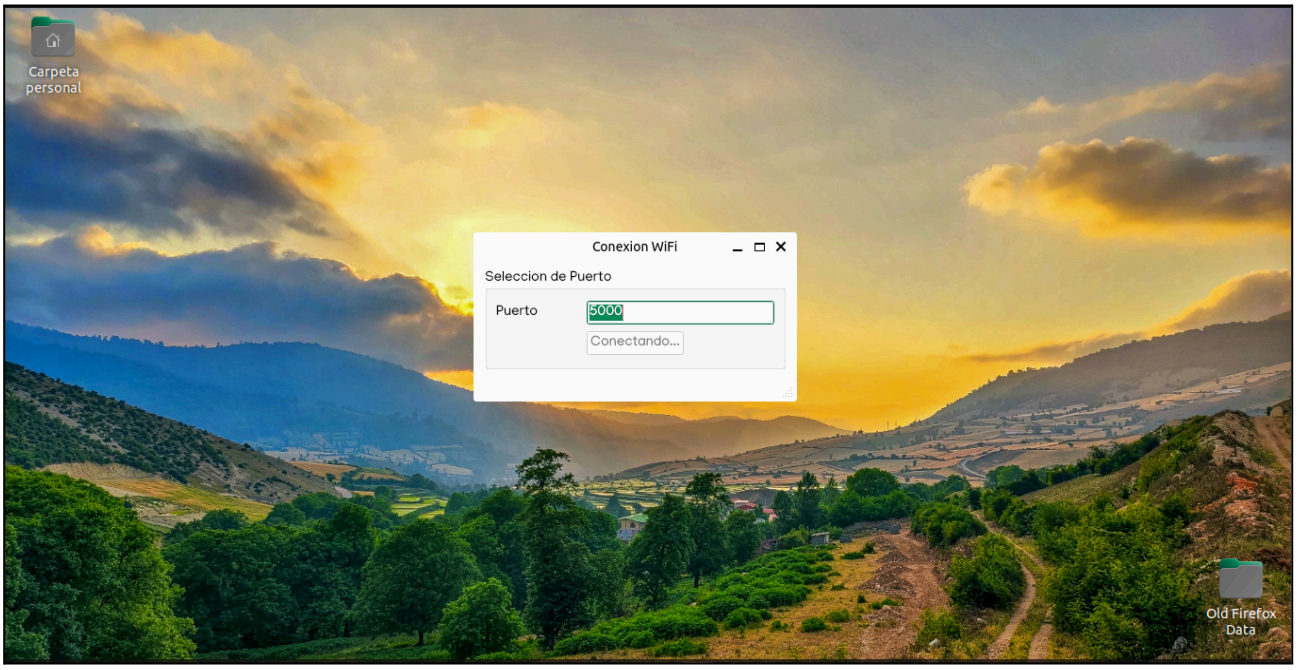
Aplicación QT

Para la presentación de la información enviada por el sensor, desarrollamos una interfaz mediante el entorno de desarrollo Qt. La misma consta de cuatro ventanas interactivas, que permiten hacer distintas cosas según la necesidad del usuario.

Apenas iniciar, se despliega la ventana de conexión WiFi, que permite seleccionar el puerto en el que vamos a escuchar las conexiones de los clientes. Una vez que el módulo WiFi se conecta, se despliega el menú, que nos ofrece dos opciones: podemos desplegar una nueva ventana que nos mostrará las mediciones que envió el módulo WiFi, o bien, podemos abrir un historial con todas las mediciones realizadas y que fueron guardadas anteriormente.

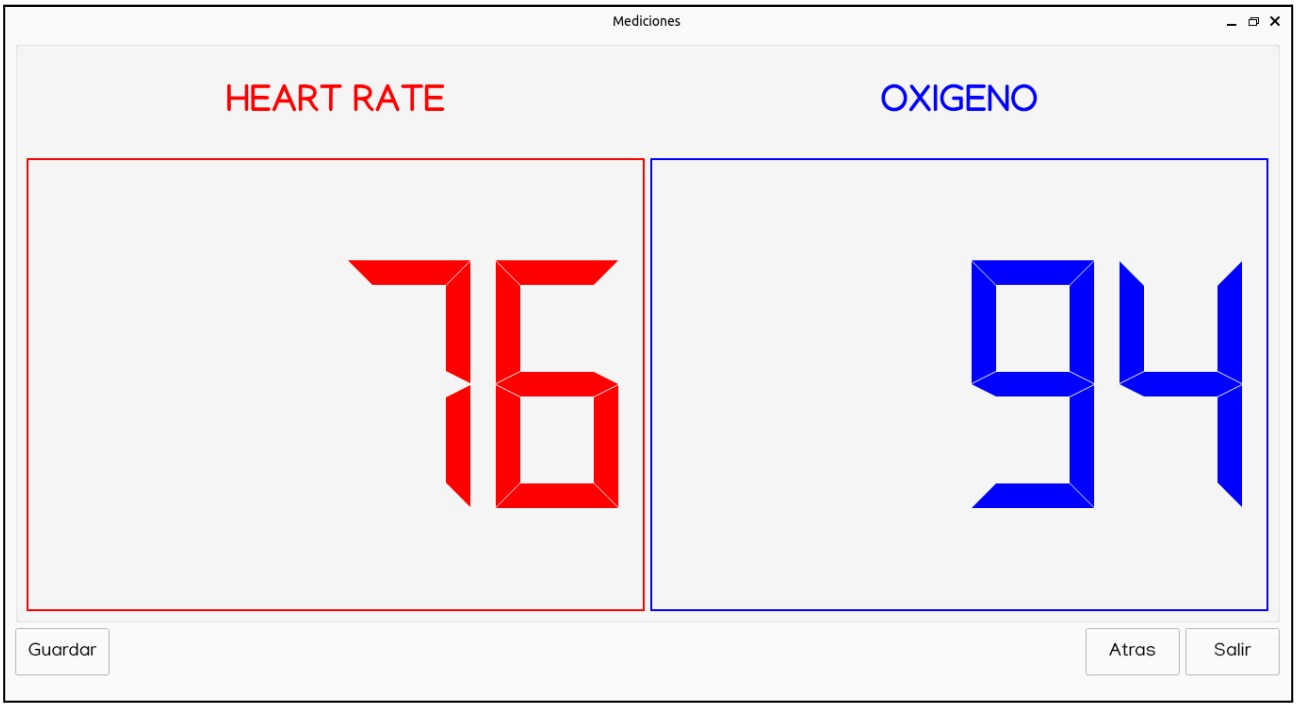
Si el usuario elige desplegar la ventana de mediciones, además de poder ver los datos medidos, tendrá la posibilidad de guardarlos en un archivo, presionando el botón “Guardar”. Al hacerlo, se despliega un dialog que les permitirá ingresar su información, tal como nombre, apellido o DNI, entre otros.

Si, por el contrario, el usuario decide abrir el historial, se abrirá y se mostrará (de existir) el archivo con toda la información guardada. En caso de que no exista tal archivo, saltará un aviso. Además, se le ofrece la posibilidad de editar el historial, o borrarlo, al presionar los botones correspondientes.

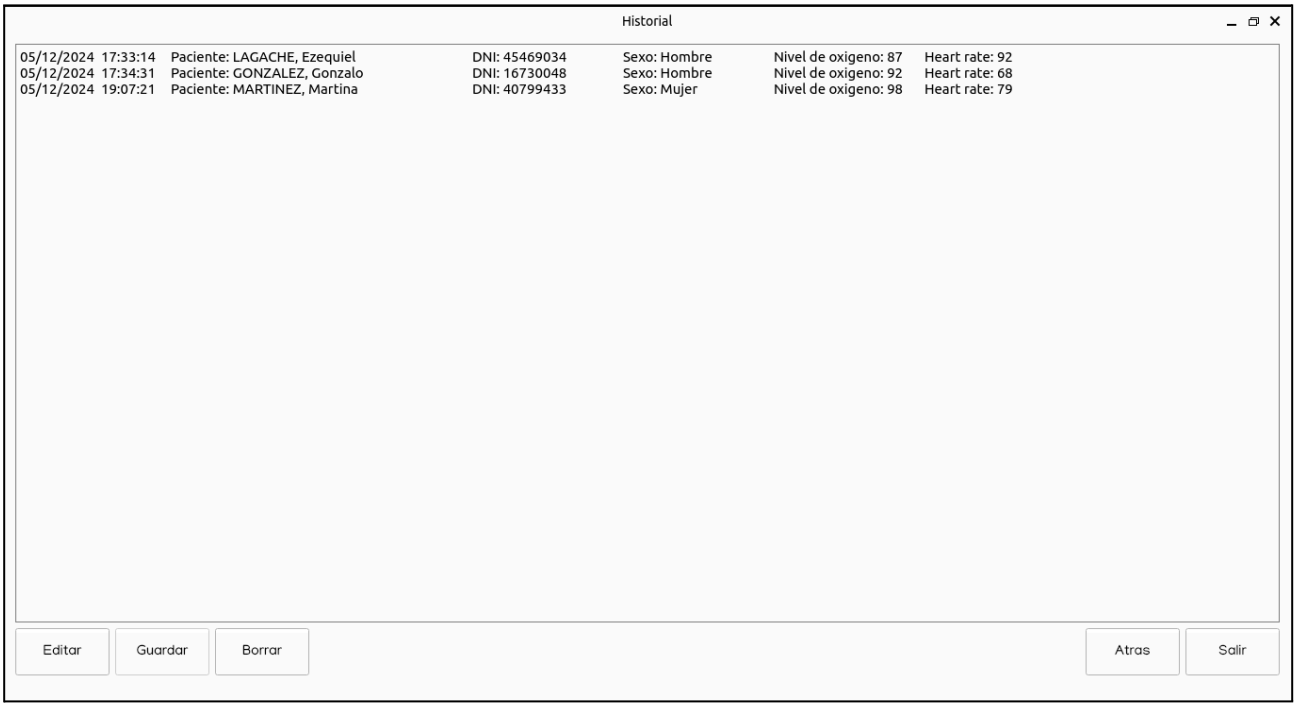


Ventana de conexión WiFi

Menú



Ventana de mediciones



Historial

Hojas de datos

- Módulo sensor MAX 30102
  - MAX30102 Datasheet and Product Info | Analog Devices
  - MAX30102--High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health
- Módulo WiFi ESP8266 y comandos AT
  - ESP8266EX
  - ESP8266 Technical Reference
  - ESP8266 AT Instruction Set
  - Basic AT Commands — ESP-AT User Guide documentation
- Módulo microcontrolador LPC845
  - LPC845 Breakout Board for LPC84x family MCUs | NXP Semiconductors
  - LPC84x Data Sheet
  - UM10601 LPC800 User manual
- Módulo regulador DC-DC Mini360
  - MP2307
- Diodo 1N4148
  - 1N4148 Datasheet(PDF) - NXP Semiconductors

PROBLEMAS ENCONTRADOS EN EL DESARROLLO DEL PROYECTO

Durante la realización del proyecto nos fuimos encontrando con diversas problemáticas tanto desde la etapa de investigación como en las etapas de desarrollo de hardware y software. Mencionamos los problemas más relevantes así como las soluciones que encontramos.

- Durante la etapa de investigación e idea fuerza o central del proyecto, nos encontramos primero con problemas para obtener dispositivos que fueran baratos y sencillos de programar, así como que tuvieran suficiente información en internet al respecto, discusiones en foros, soporte y demás. En este sentido, con la participación del GBIO de la universidad en el proyecto pudimos acceder fácilmente a estos dispositivos proveyéndonos del sensor MAX 30102 y el módulo WiFi ESP8266. Ambos utilizados por otros estudiantes miembros del laboratorio del GBIO con anterioridad y con experiencia.
- Al momento de realizar un prototipo simple y sencillo de hardware nos enfocamos en utilizar un protoboard, donde conectamos el sensor y el microcontrolador. Sin embargo, nos encontramos con ruido en la comunicación, producido por cables largos de protoboard y falso contacto; distancias largas entre pines de Tx y Rx entre el sensor y el módulo; y ruido introducido por fuentes DC de laboratorio. Esto se solucionó realizando una placa PCB soldando tiras de pines tal que se inserten los módulos allí quedando fijos y sin falso contacto con pistas cortas entre pines Tx y Rx. Se introdujo una batería como fuente de alimentación en vez de una fuente de laboratorio.
- Al obtener la placa PBC, nos encontramos con la disyuntiva de unificar o no las tensiones de alimentación entre todos los módulos. Para ello colocamos un jumper entre la VCC del módulo microcontrolador, de los módulos restantes y del Vout del regulador DC-DC. Sin embargo, esto introdujo un problema adicional lo que nos llevó a eliminar el uso del jumper. Dicho problema consistía en altas demandas de corriente desde el módulo WiFi hacia el microcontrolador y no hacia el regulador, resultando en sobrecalentamiento y destrucción del módulo WiFi o del microcontrolador. Optamos entonces por no utilizar el jumper, motivo por el cual las alimentaciones son separadas quedando la salida del regulador que alimenta al sensor MAX 30102 y el módulo WiFi ESP8266 mientras que un cable usb mini-usb C alimenta el microcontrolador.
- Al querer configurar el módulo WiFi, nos encontramos con diferentes alternativas, donde una de ellas era bastante usual y consistía en utilizar un módulo de adaptador serial TTL de nombre FTDI232, el cual en teoría facilita la visualización de los comandos transmitidos y recibidos entre el módulo y el LPC845. Dicho módulo no funcionó correctamente y no terminamos de comprender concretamente cómo utilizarlo. Como solución, optamos por utilizar la consola del IDE del MCUXpresso a través del protocolo UART y en concreto usando la UART0 del LPC845, ya que al usar otro periférico UART, no era posible visualizar en consola los comandos AT.

- Durante el desarrollo del software del driver del sensor, nos encontramos con problemas de manejo de I2C, donde no podíamos enviar correctamente en tiempo y forma la dirección del registro a modificar así como los bits a colocar en dichos registros. Dedujimos con múltiples pruebas y foros, que se debía enviar con cierto delay entre un mensaje y otro, primero la dirección del registro y luego el dato en hexadecimal asociado a los bits a colocar en ese registro. También tuvimos problemas conceptuales del protocolo. Como solución recurrimos nuevamente a foros de NXP, páginas de internet y proyectos de ejemplo del GBIO para guiarnos. Adicionalmente, utilizamos un analizador lógico de Saleae Logic Analyzer que nos permitió observar gráficamente qué ocurrirá durante la comunicación a partir de trenes de pulsos y códigos hexadecimales asociados a los pulsos.
- También, mientras desarrollamos el software del sensor, la hoja de datos usada no era muy clara respecto del orden de pasos a seguir en la configuración y también problemas con los punteros de fifo write y fifo read ya que no se mencionaba cómo utilizarlos adecuadamente. Por lo que tuvimos que realizar algunas pruebas donde enviamos ciertas configuraciones y luego leíamos los registros para verificar que se haya configurado correctamente hasta encontrar el orden correcto de configuración de registros. Las fotos y los códigos de estudiantes más avanzados del GBIO fueron de gran utilidad.
- Respecto al protocolo I2C, observamos falta de sincronización en la comunicación al enviar la dirección y el dato del registro hacia el sensor y problemas conceptuales del protocolo. Como solución recurrimos nuevamente a foros de NXP, páginas de internet y proyectos de ejemplo del GBIO para guiarnos.
- Al querer utilizar el módulo WiFi nos encontramos con que éste producía picos de corriente que superaban los 200 mA por lo que, como solución, la corriente demandada debía ser ofrecida por un regulador de tensión cuya salida fuera mucho mayor que los picos de corriente evaluados. También el colocar un capacitor de 100uF a 50V fue una solución efectiva para amortiguar los picos.
- Al programar el módulo WiFi nos encontramos con problemas de comprensión sobre el patrón de comunicación con el módulo WiFi, el manejo y orden de los comandos AT, uso de funciones bloqueantes para la recepción del “OK\r\n” y para la transmisión así como problemas para establecer delays adecuados entre un mensaje y otro. Para solucionarlo usamos una máquina de estado para manejar la recepción del “OK\r\n” y un sistema de encolamiento o Queue para transmitir el tamaño del mensaje y el mensaje en sí mismo.

BENEFICIOS ENCONTRADOS EN EL DESARROLLO DEL PROYECTO

Como beneficios obtenidos a partir del proyecto podemos identificar:

- Aprendizaje e incorporación de conocimientos de programación en C++ y OOP en sistemas embebidos para desarrollar un dispositivo prototipo y poder visualizar los efectos de nuestra programación.
- Entendimiento de cómo crear nuestras propias aplicaciones y drivers o librerías de código.
- Integración de contenidos interdisciplinarios de diferentes materias como ASyS, INFO1, INFO2 y Sistemas de Representación, Sistemas Embebidos, Diseño de Hardware y Análisis de Señales.
- Crecimiento académico y técnico.
- Entender la importancia de optimizar recursos en hardware y software en dispositivos biomédicos.
- Resolución de problemas complejos.
- Desarrollo de habilidades para enfrentar y resolver problemas en tiempo real, tanto en hardware como en software.
- Mejorar habilidades de trabajo grupal, comunicación y organización de tareas.
- Comprender los retos prácticos de los sistemas biomédicos, lo que amplía la visión hacia aplicaciones reales en IoT y bioingeniería.

CONCLUSIÓN

Podemos establecer las siguientes conclusiones del proyecto:

- El proyecto cumplió con los objetivos de desarrollar un prototipo funcional de pulsioxímetro.
- Integración exitosa de hardware, software y comunicación inalámbrica para la visualización en tiempo real de parámetros biomédicos.
- Superación de desafíos relacionados con la comunicación I2C y los picos de corriente en el módulo ESP8266.
- Implementación de soluciones robustas, como máquinas de estado y optimización del diseño PCB.
- Este proyecto sienta las bases para futuros desarrollos en dispositivos biomédicos accesibles y modulares.
- Potencial para continuar el trabajo en áreas como interfaces más avanzadas, mejora en la precisión de medición y optimización del consumo energético.

ANEXO

- Lucidchart
- NXP Forum

https://community.nxp.com/t5/forums/searchpage/tab/message?advanced=false&advanced=false&allow_punctuation=true&allow_punctuation=true&q=LPC845&q=LPC845

- Aula virtual INFO2

https://aulasvirtuales.frba.utn.edu.ar/course/view.php?id=15301

- I2C Guide

How to write a microcontroller driver for an I2C device? (MSP430 and VL6180X)
A Basic Guide to I2C
Understanding the I2C Bus
I2C Communication Protocol - GeeksforGeeks
I2C-bus specification and user manual
Understanding I2C
I2C For Beginners: Device Addresses from Data Sheets

- ESP8266 y TCP Guide

AT Command Set — ESP-AT User Guide documentation
ESP8266 AT Instruction Set
Protocolo TCP: definición y funcionamiento
What is TCP (Transmission Control Protocol)? - GeeksforGeeks
https://www.ionos.com/es-us/digitalguide/servidores/know-how/que-es-tcp-transport-control-protocol/?srsltid=AfmBOor_nT8KNcOoOwQFfQeYSDmm9i1DfOYVijV4Dj-I1e1-Apa7k2Ne