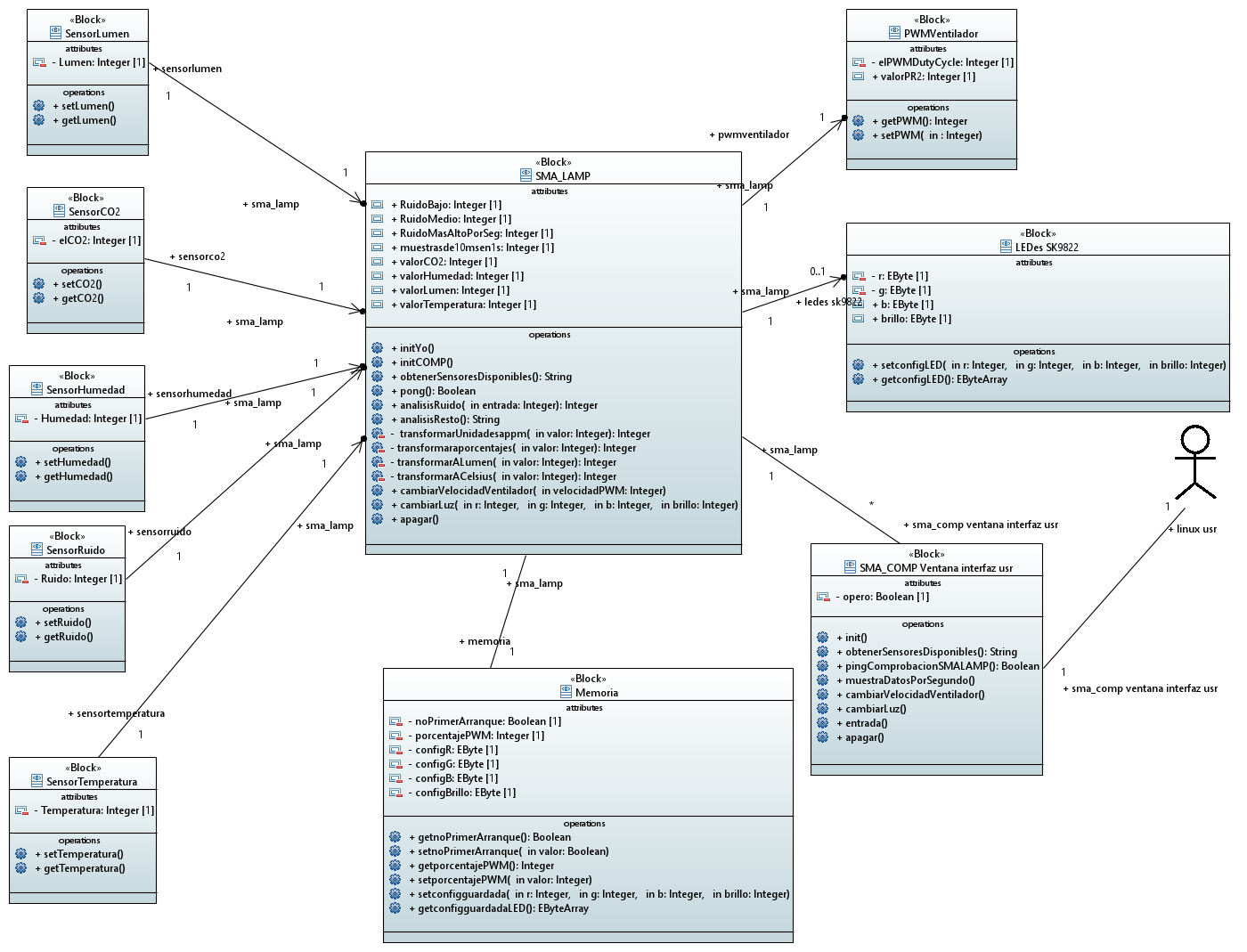
A partir de la documentación generada en el apartado anterior, los alumnos deberán preparar un documento de diseño hardware y software del SE.

1. Diseño software de alto nivel. Se planteará el diseño software de alto nivel (solo grandes bloques del sistema) utilizando para ello un diagrama de bloques (SysML).

Nos hemos basado en una estructura de arquitectura sensores-controlador-actuadores, en las que los sensores entregan los datos a la unidad SMA\_LAMP de control que será la que actúe sobre los LEDes y motor PWM. Nótese que se incluyen dos bloques adicionales, uno que permite la comunicación en doble sentido con un usuario Linux, y otro que almacenará las características de los actuadores PWM y LEDes en memoria flash cada vez que se modifiquen.



2. Diseño lógico hardware. Debe elaborarse el diseño lógico hardware (o diseño esquemático) necesario para atender los requisitos planteados para el sistema (ver herramientas obligatorias).

Según las indicaciones de la documentación del Microchip PIC 16F886, el circuito necesita de alimentación y señal de reloj para funcionar. Según el manual, entre Vpp y Vdd debe haber una resistencia mayor de 1kΩ pero menor de 10kΩ, y opcionalmente un condensador que conecte Vpp y GND. Nosotros hemos elegido aplicar 5kΩ y no emplear condensador. Además ambas entradas Vss deben estar conectadas, en nuestro caso a tierra. Queremos que la señal de reloj sea de 20MHz, así que a CLKOUT y CLKIN debemos conectar un cristal de dicha frecuencia, que según las indicaciones del manual debe ir acompañado de dos condensadores de 22 pF conectados cada uno a un lado del cristal, y entre sí a tierra.

Además hemos tenido en cuenta el regulador lineal A7800 para una salida fija y eliminación de ruido, por el cuál debíamos asignar al regulador dos condensadores polares en las patas Vi y Vo de 0.33 y 0.1 µF, respectivamente.

Ya con las partes más obvias para el proyecto, necesitamos de una conexión a nuestro ordenador para programar la placa (con PICKIT3) y las entradas y salidas pedidas en el enunciado. Así, hemos conectado el PICKIT3 según las especificaciones (PIN1 al MCLR, PIN2 a Vdd, PIN3 a GND, PIN4 a RB7 y PIN5 a RB6, dejando PGM sin implementar); creado un Conector 01x03 para la USART (PIN3 es TX, PIN2 ES RX y PIN1 es GND) y colocado un conector 01x08 en el que los pines 1, 4 y 5 son de entradas analógicas de ruido, humedad relativa y temperatura y conectan con los pines analógicos AN0, AN1 y AN2, los pines 2 y 3 con SCL y SDA de I2C y el PIN6 para el motor con PWM al módulo CCP1. Desafortunadamente para nosotros, en nuestro chip el bus I2C y el bus SPI están conectados en mismos pines, por lo que como alternativa más sencilla deberemos simular uno de ellos (específicamente el SDI sobre el I2C), lo que supone que el pin 7 se conecta a SCK/SDL y será muy probable que aunque el pin 8 se conecte al SDO del SPI, luego acabe sin usarse.

Los sensores de CO2 y luminosidad se comunicarán por I2C

Diagrama, Esquemático

Descripción generada automáticamente

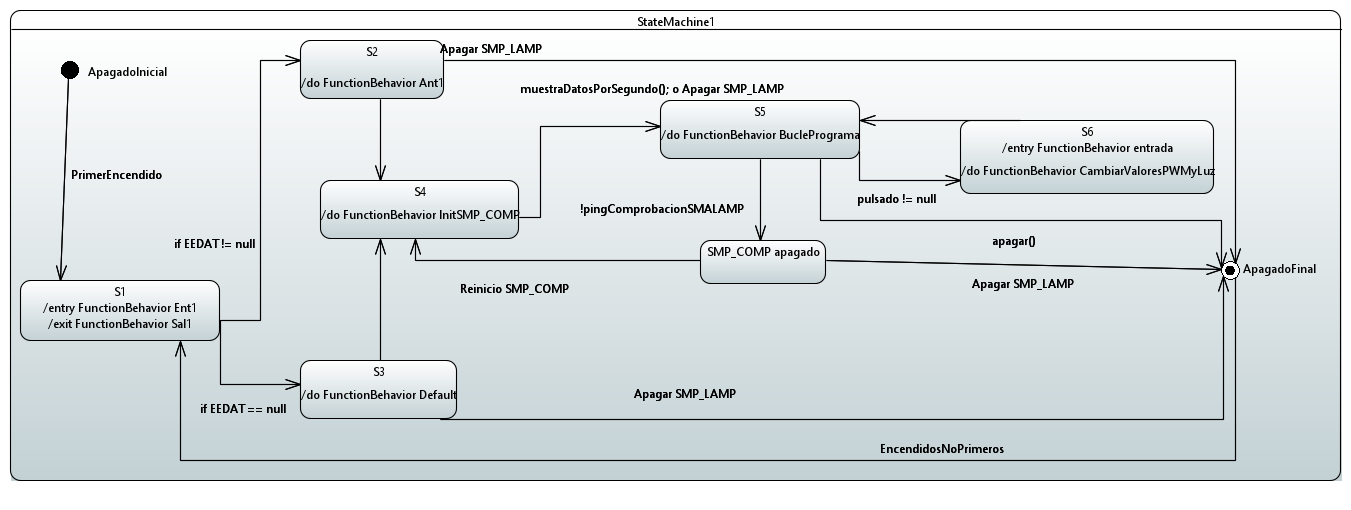
Sin embargo, al final por limitaciones a nivel físico (específicamente asegurarse de que todos los cables pudieran pasar sin cruzarse) realizamos unos cambios al diseño lógico, en el que el Connector de 8 lo dividimos en un Connector de 3 (para las entradas de Ruido, Humedad y Temperatura, pines 1,2, 3) y otro de 5 (1 y 2 para SCL y SDA, 3 para CCP1 y 4 y 5 para SCK y SDO)

Diagrama, Esquemático

Descripción generada automáticamente

3. Diseño software detallado. Por medio de diagramas de estado, diagramas de secuencia u otros recursos que se consideren apropiados, debe plantearse el diseño detallado del componente software del sistema.

A partir del diseño SW de alto nivel hemos elaborado un diagrama de estados en el que partiendo de un estado de apagado inicial se inicia la placa SMA\_LAMP y los periféricos, y al acceder a la memoria o bien se pone la configuración anterior, o si no existe la configuración por defecto de luz blanca y motor apagado. Una vez hecho eso se arranca el SMA\_COMP, que iniciará sus ventanas, y se quedará en un bucle infinito que comprobará que se puede conectar al SMA\_LAMP. Si no puede se cierra y deberá reiniciarse para volver a intentarlo. Si se conecta con éxito procederá a mostrar los datos de los sensores en un periodo de tiempo preestablecido (1 o 5 segundos) y dará la opción de aceptar por entrada de terminal o botón el permitir modificar la luz y la velocidad del ventilador. Una vez la configuración se ha realizado el SMA\_LAMP procederá a escribir los datos en la memoria para un posible acceso futuro tras apagar.



Más tarde tras recibir información adicional realizamos un segundo diagrama de Estados que indicaba de forma básica en qué momento el SMA\_LAMP se encontraba en el programa main o en la rutina de tratamiento de interrupción. Este diagrama se centraba más en la lámpara y nos permitió definir con mucho más detalle como finalmente funcionaría la lámpara: el main iniciaría el sistema, comprobaría la memoria y según los datos obtenidos, actualizará los valores de la tira de LEDes por el bus SPI y del ventilador por señal PWM con el CCP1 y la nueva configuración se guardará en la memoria; tras hacer esto se quedaría en un bucle infinito en el que se irían haciendo algunas de las funciones pedidas, como actualizar el valor que luego se enviaría a SMA\_COMP. La rutina de tratamiento de interrupción se encarga de un gran número de funciones importantes, regulación de los Timers 0 (cuenta procesos generales del main, cada segundo y cada 5 s), 1 (regulación del tiempo sample and hold del ADC, monitoriza el nivel de ruido cada 10 ms) y 2 (para generación de la señal PWM para nuestro ventilador), los flags de escritura completa en memoria, el de I2C para los sensores de luminosidad y CO2 y por último flags de la USART, tanto de TR como RX, para poder ejecutar diferentes opciones según una cadena enviada o recibida:

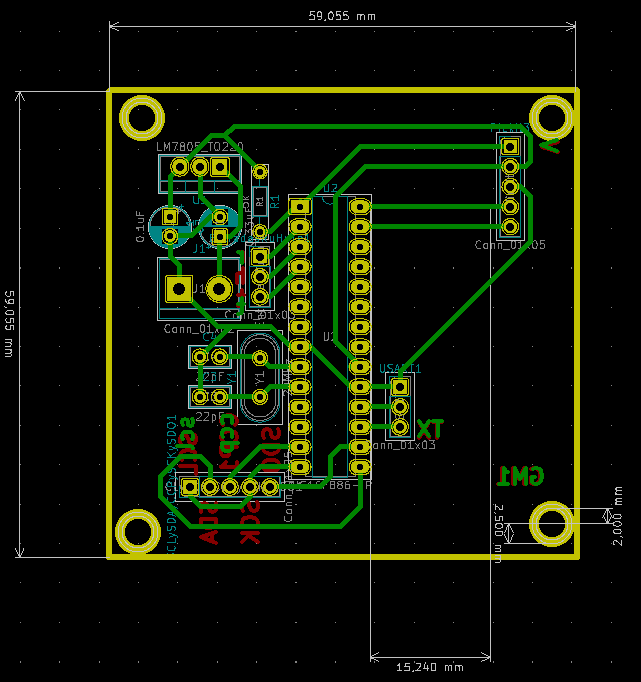
* "a" sería devolver los valores de sensores al SMA\_COMP: TXREG = 'A' + valoresSensores(); valoresSensores devolvería una cadena de caracteres indicando el estado de las variables asociadas a los sensores indicados, según unos valores de variables globales manejados por funciones que informarían de si los valores son basura o no.
* "b" es hacer ping: TXREG = 'B'; (nos bastaría con esto para que el SMA\_COMP supiera que ahí estamos).
* "c" es cambiar el PWM, setPWM(valor); donde valor sería el segundo char recibido de RCREG.
* "d" es cambiar LED, setLED(red, green, blue, luminosidad); siendo los parámetros el segundo, tercero, cuarto y quinto char recibido de RCREG
* "e" sería apagar el SMA\_LAMP, se podría hacer con una señal de reset específica, la que mejor nos convenga a bajo nivel, y una variable int deboContinuar (inicialmente a 1) que se pusiera a 0 y así rompiera el bucle del programa principal.

Diagrama

Descripción generada automáticamente

4. Diseño físico hardware. A partir de la utilización de alguna herramienta de diseño, debe obtenerse el diseño físico hardware, respetando para ello el diseño lógico hardware previamente planteado (ver herramientas obligatorias).

Acá se eligió nombrar algunos de los pines como TX, el pin 1 de pickit3, los pines de ruido (r) y temperatura (t), el SCL y el CCCP1 (ccp) pero no el resto por falta de espacio. También se decidió conectar el LM7805 con el pin 3 del pickit3 en vez de con el de 5V del microchip porque a pesar de que la ruta era más corta hubiera supuesto pasar entre varios pines del microchip y eso acarrearía problemas a la hora de atacar la placa.



5. En el apartado de diseño deben plantearse las siguientes trazas:

1. Traza de componentes del diseño software de alto nivel contra requisitos.
2. Traza de diseño físico (HW) contra diseño lógico (HW) haciendo las siguientes comprobaciones.
3. Todos los componentes del diseño lógico aparecen en el diseño físico.
4. Todas las conexiones establecidas en el diseño lógico aparecen en el diseño físico.

Al comprobar el diseño físico no se ha visto que falten conexiones ni componentes del diseño lógico.

|  |  |  |
| --- | --- | --- |
| **Requisito** | **Componente HW** | **Componente SW** |
| ENC-10 | SMA-LAMP, ordenador Linux | SMA-LAMP, SMA-COMP |
| ENC-20 | Sensor Temperatura, Humedad, Ruido, CO2 y Luminosidad, SMA-LAMP, USART y ordenador Linux. | Sensor Temperatura, Humedad, Ruido, CO2 y Luminosidad; SMA-LAMP y SMA-COMP |
| ENC-30 | EEPROM/Flash | Memoria |
| AP-10 | SMA-LAMP (registros/mem volátil) | SMA-LAMP |
| AP-20 | Ordenador Linux/USART | SMA-COMP |
| AP-30 | Ordenador Linux/USART | SMA-COMP |
| MO-10 | CAD | Sensor Ruido |
| MO-20 | Timer | Sensor Ruido |
| MO-30 | Timer | Sensor Temperatura, Humedad, Ruido, CO2 y Luminosidad |
| MO-40 | I2C (C02 y Luminosidad), ADC (el resto) | Sensor Temperatura, Humedad, Ruido, CO2 y Luminosidad |
| AC-10 | I2C, USART | Memoria |
| AC-20 | PWM, USART | PWMVentilador |

1. Comprobaciones añadidas del diseño físico (HW).  
   1) No hay soldaduras con imposibilidad de ser realizadas.

2) El espacio reservado para el conector de programación permite insertar la sonda sin chocar con ningún componente (Pickit3).

4cm x 1.5 cm alrededor del Pickit3

3) Las líneas de conexión entre componentes tienen el ancho adecuado y separación suficiente para el proceso de fabricación a utilizar.  
4) Todos los componentes están correctamente alimentados.

5) Se han introducido marcas de serigrafiado adecuadas para identificar correctamente el circuito y su interfaz.

6) Se incluyen las marcas para el mecanizado de sujeción. Por defecto taladros de métrica 4 (deja espacio adicional).

7) No hay pistas que hagan contacto con las partes metálicas del mecanizado de sujeción.