UTN FRA - Ramón Franco 5050 (B1874ABY)
Villa Dominico, Buenos Aires, Argentina
Tel. +54 11 4353 0220

PID CCRENAV0004791: "DISEÑO DE NUEVO MODELO DE ESTACION AGROMETEOROLÓGICA AUTOMÁTICA PORTATIL "NIMBUS III INTA" BASADO EN SISTEMAS EMBEBIDOS. PARA EL ESTUDIO DEL MICROCLIMA EN PARCELAS DE ENSAYO".

LIBRERÍA PARA LPC4337 MULTICORE

Especificación:	**UTNF-INTA-NIM3-D000-1A**	**/ USO INTERNO**
Versión y fecha:	Rev1A 04/06/2019	
Generado por:	**Nahuel Coco**	
Revisado por:	**Marcelo Estevez, Ariel Marín**	
Aprobado por:	**Marcelo Belloni**	
Ingeniería de Proyecto:	**Marcelo Belloni, Mario D´Indio, Marcelo Estevez. Ariel Marín, Gerardo Battaglia, Martin Nápoli, Guillermo Ciampone, Edgardo Porral, Nahuel Coco, Fabian Massotto.**	
Resumen:	*Documento explicativo y demostrativo sobre el uso de la librería y sus alcances.*	
Palabras Clave:	*EMA, Estaciones, Clima, Sistemas Embedidos, CORTEX*	

Tabla de Contenidos

Introducción

Con la presente documentación se buscará explicar y guiar a todo aquel que desee utilizar la API. Esto se hará de manera que todos puedan utilizarla y su modo de uso sea lo más ameno posible.

Debe conocer que la documentación de esta API la podrá encontrar actualizada en su última versión desde mi GitHub https://github.com/NahuelCoco/lpcAPI. Con este proyecto se busca que la comunidad pueda contribuir y hacer sus aportes para que se logre una librería completa en todos sus aspectos siendo la finalidad principal hacer que un proyecto nacional como es la CIAA pueda ser de fácil utilización y permita, en el caso de la EDU CIAA un aprendizaje exitoso y un mayor abanico de posibilidades en todos los aspectos.

En el GitHub se podrá encontrar con tres carpetas.

- Coco_api: Librería para el Cortex M4
- Coco_api_M0: Librería para el Cortex M0
- Examples: Ejemplos de programas para cada módulo.

Microcontrolador LPC4337

Este se trata de un microcontrolador de doble núcleo asimétrico. Hay que saber que el LPC4337 está basado en un Cortex M4 con FPU (Unidad de Punto Flotante) de 32 bits, un Cortex M0 de también 32 bits como co-procesador que tendrá hasta 1 MB de memoria flash y 136 KB de SRAM, 16 KB de EEPROM, 2 controladores USB high-speed, Ethernet, LCD, un controlador de memoria externa, módulo SPIFI, SCT (State Configurable Timer) con el cual podremos ejecutar PWM, tendremos además GPIOs y múltiples periféricos analógicos y digitales.

La frecuencia máxima a la que puede operar el LPC4337 es de 204 MHz aunque se recomienda disminuirla si así el proyecto lo permite para evitar temperaturas elevadas de trabajo.
Para más información se recomienda descargar el Datasheet de este microcontrolador. Si bien durante este documento tomaremos referencias de dicho material, será de gran ayuda tenerlo, más aún si alguna explicación de las que se encuentra aquí no se logra comprender.

EDU-CIAA

En Argentina, uno de los proyectos nacionales es el de la CIAA teniendo como plataforma educativa la EDU-CIAA. Este proyecto, tiene una página oficial donde podrá encontrar infinidad de información e incluso un foro de debate donde los expertos contestan consultas e inquietudes. Como aporte a la comunidad, surgió como idea frente a las dificultades de iniciarse en la programación de estos microcontroladores, encontrar una forma más sencilla de programarlos, pero también manteniendo el Hardware ya diseñado.

Entonces, teníamos la siguiente inquietud: Generar un código que permita a todos los estudiantes, técnicos e ingenieros involucrarse y utilizar la EDU-CIAA de manera sencilla y que incluso, una vez terminado el prototipo con dicha plataforma, poder independizarse de ella y poder utilizar ese

código hecho para el LPC4337 – microcontrolador que posee la EDU-CIAA – en un hardware más o menos pequeño pero adaptable a lo necesario.

Analizando, llegamos a la sencilla conclusión de que hoy en día, la plataforma más sencilla de utilizar es la de Arduino, como contrapartida, la forma en la que está creada no apunta a un ámbito de Ingeniería. Por lo que, pensamos en generar una API a las que todos puedan acceder, pero utilizando la plataforma oficial del microcontrolador, el MCUXpresso.

Por lo cual, usted podrá descargar la API desde el GitHub, añadirla al MCUXpresso y utilizarla para el proyecto que desee. ¿Cuál es la ventaja? Las funciones están creadas con el mismo nombre que en Arduino. Entonces, al saber programar Arduino, sabemos programar la EDU-CIAA. Otra de las ventajas es que están generadas de la manera más sencilla con C y C++, sin buscar ninguna complicación al que quiera agregar o modificar algo a la librería.

Si bien este es un proyecto de código abierto, también tiene la finalidad que la comunidad busque hacer aportes que puedan hacer aún más completa esta API, por lo que puede sentirse libre de hacerlo.

Instalación y uso de la API

En principio, debe saber que esta guía está hecha para una EDU-CIAA utilizada con un programador externo. Esto se hizo así pensando en la independización de la EDU-CIAA luego de utilizarla para prototipo, ya que sería el caso en donde muchos se podrán encontrar frente a la necesidad de no querer una plataforma con leds, microprocesadores y demás componentes que generan más consumo y ocupación física de espacio para su proyecto.

La plataforma tiene los 10 pines de JTAG facilitados que nos permitirán hacer esto. En nuestro caso, utilizamos el LPC Link 2 – OM13054 como programador. Este último es una plataforma que puede ser usada para desarrollo o tiene firmwares desarrollados que se los pueden cargar para que actúen como programadores en herramientas o IDEs donde se soporte protocolos SEGGER J-Link y/o CMSIS-DAP.

Para este caso, el firmware que se le cargará será el SEGGER J-Link. Para hacerlo, deberá descargar el LPCScrypt y el firmware para J-Link desde la página de NXP. Es muy posible que deba hacerse una cuenta en NXP que luego también la necesitará para descargar el MCUXpresso.

Una vez descargado tanto el LPCScrypt como el Firmware – el cual generalmente se descarga en conjunto con el LPCScrypt -, abriremos el primero y luego, asegurándonos que no esté el jumper en JP1 en el LPC-Link2, ejecutaremos “program_JLINK.cmd” desde la CLI y seguiremos las instrucciones.

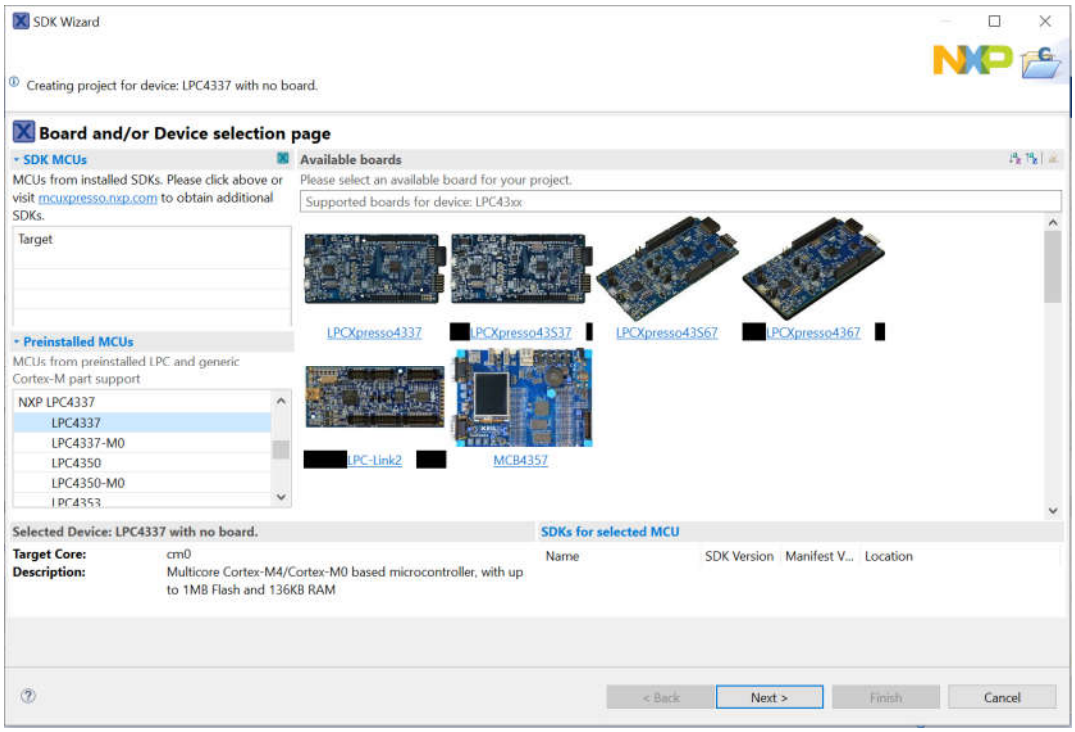
```
C:\nxp\LPCScrypt_2.1.1_15\scripts>ls
LPCScrypt_CLI.cmd  ListLPCComPorts.cmd  boot_lpcscrypt.cmd       encrypt_and_program.scy  program_JLINK.cmd
ListComPorts.cmd   aeskey.cmd           encrypt_and_program.cmd  program_CMSIS.cmd

C:\nxp\LPCScrypt_2.1.1_15\scripts>program_JLINK.cmd
```

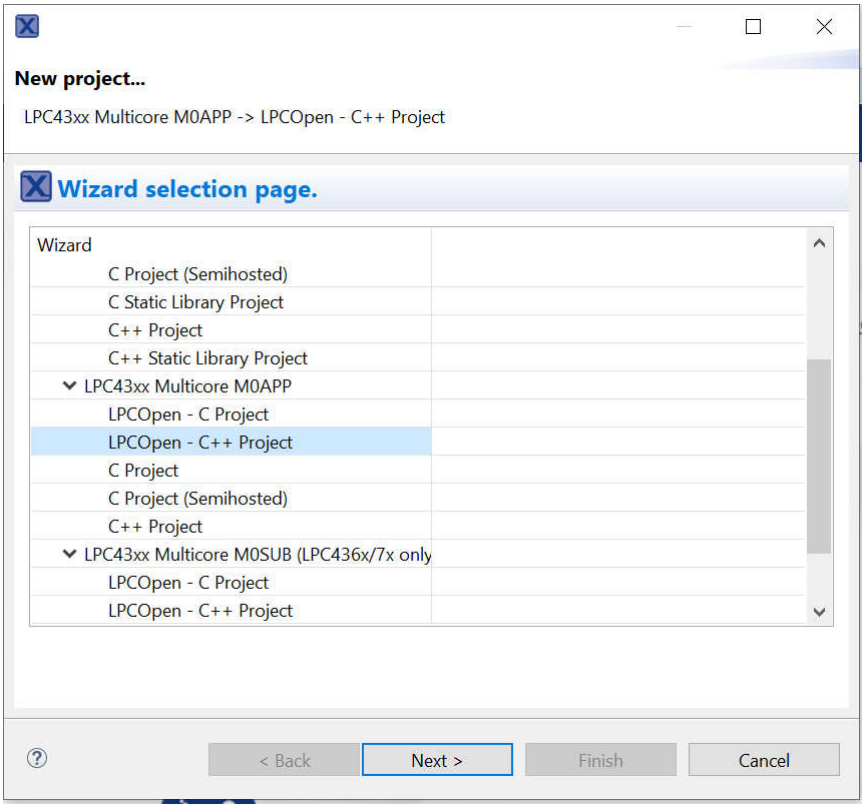
Si usted utilizará otro programador, la explicación de arriba quizás no le sea de utilidad. Una vez que ya tengamos el programador listo y compatible con el MCUXpresso, podemos descargar e instalar este último software.

Luego que esté instalado, tendremos que descargar desde el repositorio de GitHub, coco_api y coco_api_M0.

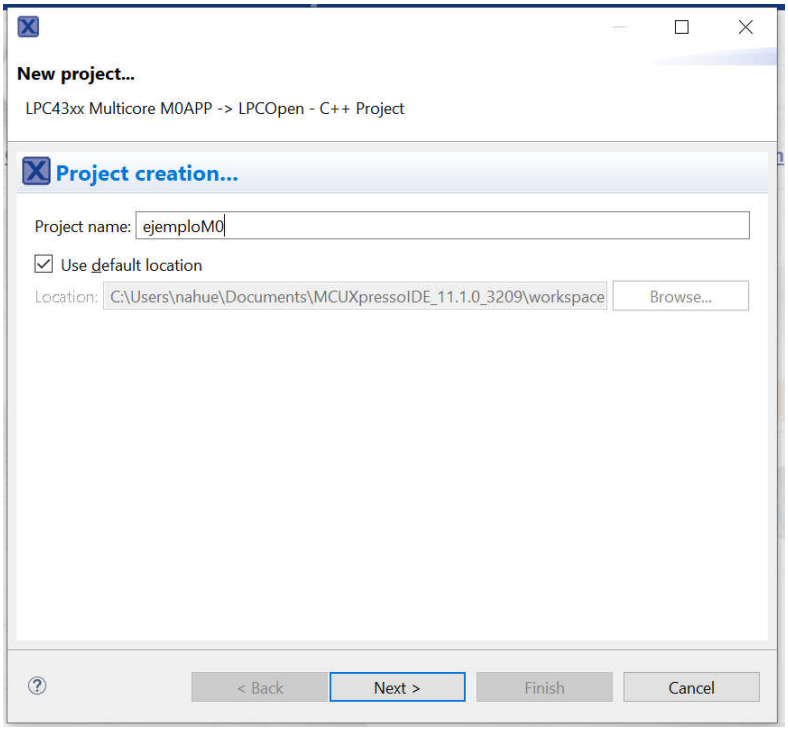
Comencemos en principio a instalar una librería de la que se basa la API, esta será LPCOpen y en particular instalaremos la correspondiente al LPC4337. Entonces, abrimos el MCUXpresso y creamos un nuevo proyecto de ejemplo. Esto lo haremos desde File > New > New C/C++ Project.
Aquí, seleccionaremos el LPC4337 – M0 ya que haremos un proyecto para el Cortex M0 de este microcontrolador.



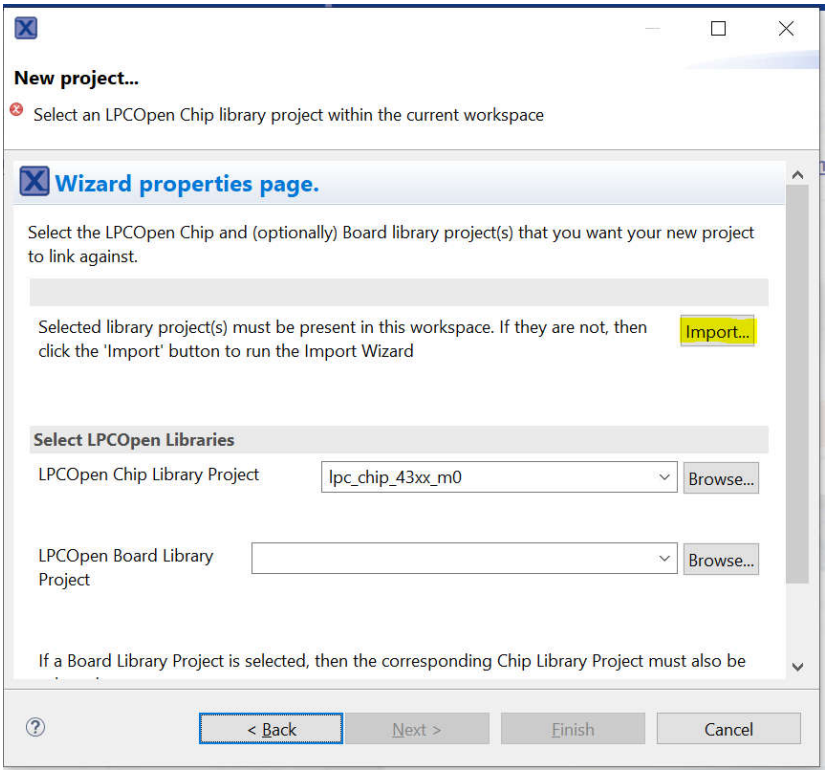
Luego, presionamos “Next” y en la siguiente ventana seleccionaremos como tipo de proyecto “LPCOpen – C++ Project” pero tenga en cuenta de seleccionar el que está dentro de LPC43xx Multicore M0APP ya que lo usaremos en un futuro como programa del Cortex M0 en Multicore.



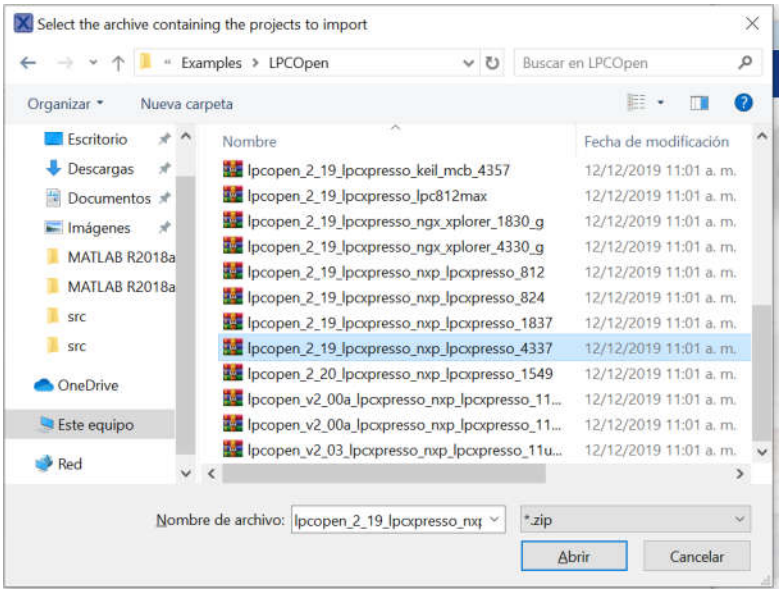
Al presionar en “Next” nos aparecerá una ventana donde colocaremos el nombre al proyecto llamándolo “ejemploM0” y creándolo en el Workspace por defecto.



En la ventana siguiente, tendremos que agregar por única vez las librerias del LPCOpen para el LPC4337. Esto lo haremos presionando en Import.

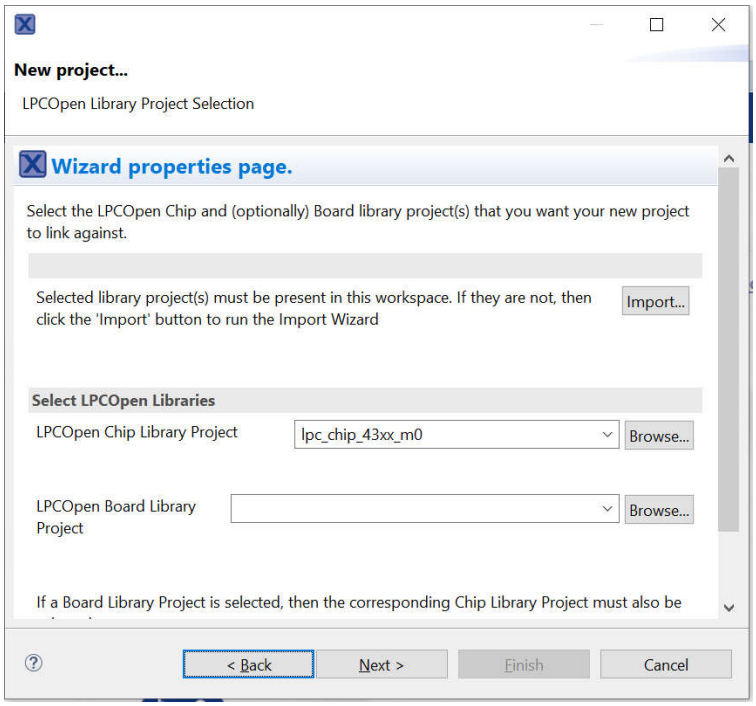


Luego presionamos en Browse en el campo de “Project archive (zip)” y buscamos en la carpeta de LPCOpen la librería correspondiente al LPC4337.

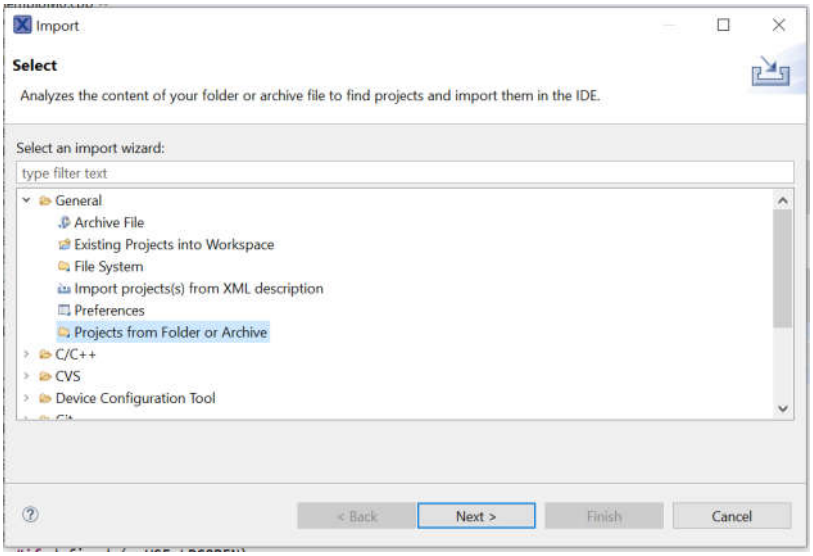


Presionamos siguiente y con todo seleccionado, presionamos en “Finish” esperando que termine la instalación.

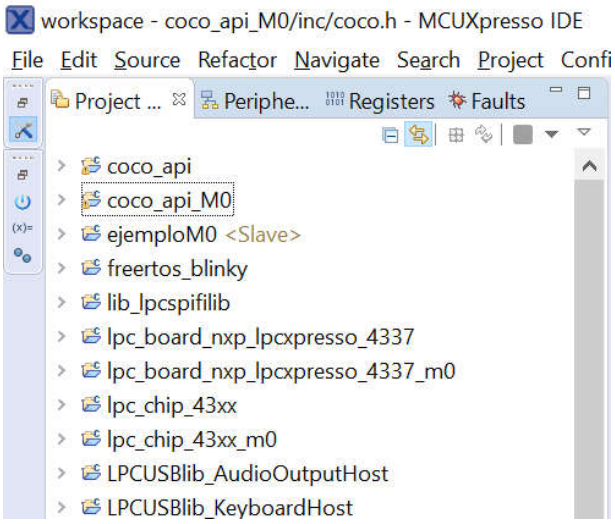
Finalmente, cuando todo haya terminado de instalar, en lugar de seguir con la creación del proyecto, lo cancelaremos y cerraremos el MCUXpresso para luego abrirlo nuevamente. Esto hará que se actualice el Workspace. Ahora, hacemos nuevamente todos los pasos, pero ya podremos seleccionar la librería del LPCOpen y quedará disponible para el Workspace en el que estamos trabajando, **por lo que no hará falta repetir estos pasos en un futuro.**



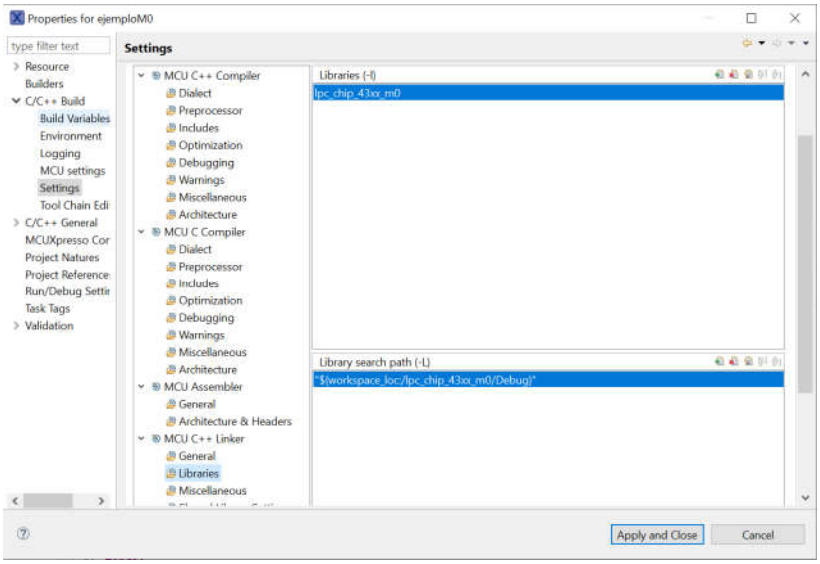
Para todo lo demás presionaremos “Next” y llegado el momento, “Finish”. En esta etapa no será necesario hacer el mapeo de memoria ya que luego se puede modificar.
Ya con el proyecto creado y la librería descargada. Presionaremos en File > Import. Allí dentro agregaremos coco_api_M0.



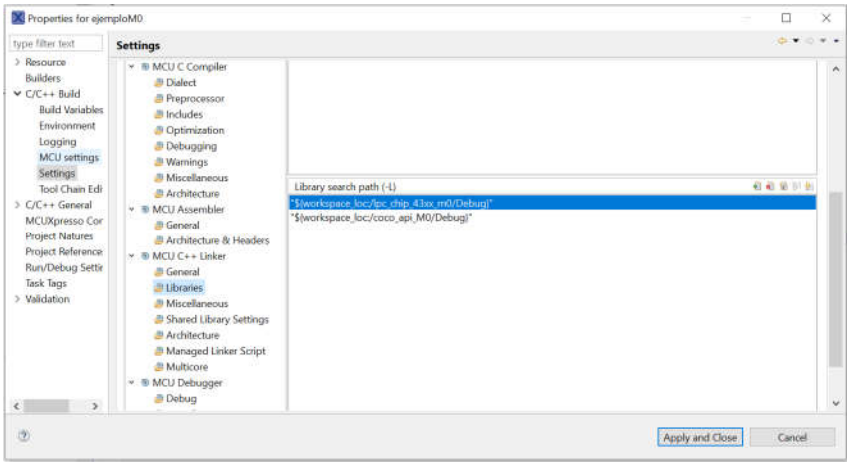
Una vez seleccionada la carpeta coco_api_M0 presionando en Directory dentro del campo Import source, debería aparecernos dentro de los proyectos del Workspace. Esto mismo deberemos hacer con coco_api para el Cortex M4.



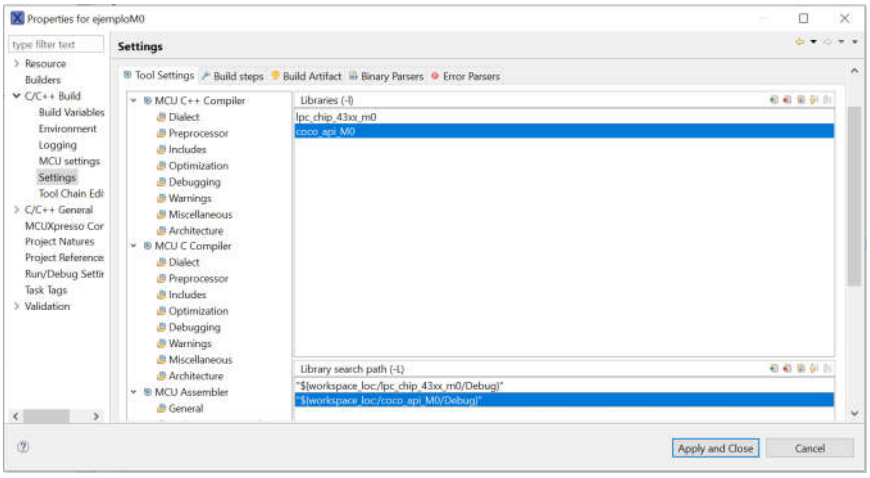
Ahora, esta API podrá ser usado en cada proyecto que usted quiera y no será necesario repetir estos pasos nuevamente. Ahora, enlacemos nuestro proyecto ejemploM0 con la librería coco_api_M0 para poder ser utilizada.
Para ello presionamos botón derecho sobre el proyecto, y luego en Properties. Allí expandimos C/C++ Build y apretamos en Settings. Dentro de ahí y en Tool Settings, presionaremos en Libraries del módulo MCU C++ Linker.



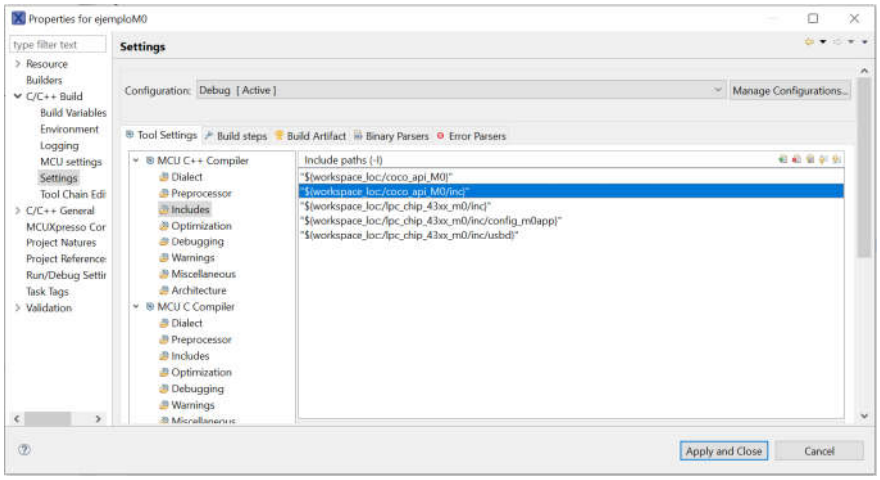
Luego, apretemos en el símbolo de Add de la sección “Library search path (-L)” y al hacerlo aparecerá una ventana donde presionaremos en Workspace para buscar coco_api_M0/Debug.



Ahora en la sección Libraries, agregaremos una nueva con el nombre coco_api_M0 escribiéndolo.

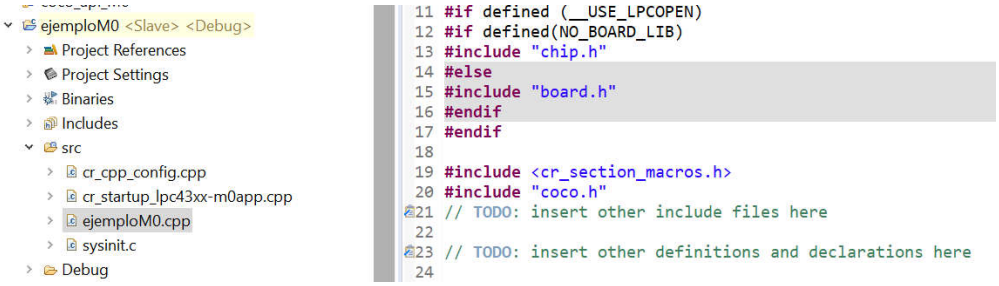


Finalmente, en MCU C++ Compiler > Includes, agregaremos la carpeta inc de la librería.



Para finalizar, entraremos en C/C++ General > Paths and Symbols > References y seleccionaremos coco_api_M0.

Al presionar Apply and Close, ya podremos usar la librería con total libertad en el proyecto ejemploM0. **Estos pasos se deberán repetir cada vez que se cree un nuevo proyecto.**
Hagamos una prueba rápida que nos quedará programada para la prueba en doble núcleo. En principio debemos añadir un include para la librería. Solo hará falta incluir coco.h.



Este será un programa sencillo en donde con el Cortex M0 blinkearemos el Led1 de la EDU-CIAA. Entonces el código quedará:

```
49
50      // TODO: insert code here
51      _coco();
52      pinMode(39, OUTPUT);
53
54      while(1) {
55
56          digitalToggle(39);
57          delay(1000);
58          __asm volatile ("nop");
59      }
```

Si compilamos no deberíamos obtener ningún error. Todavía no lo subiremos al microcontrolador ya que lo haremos en la prueba completa con el Cortex M4 ejecutando un escenario Multicore.

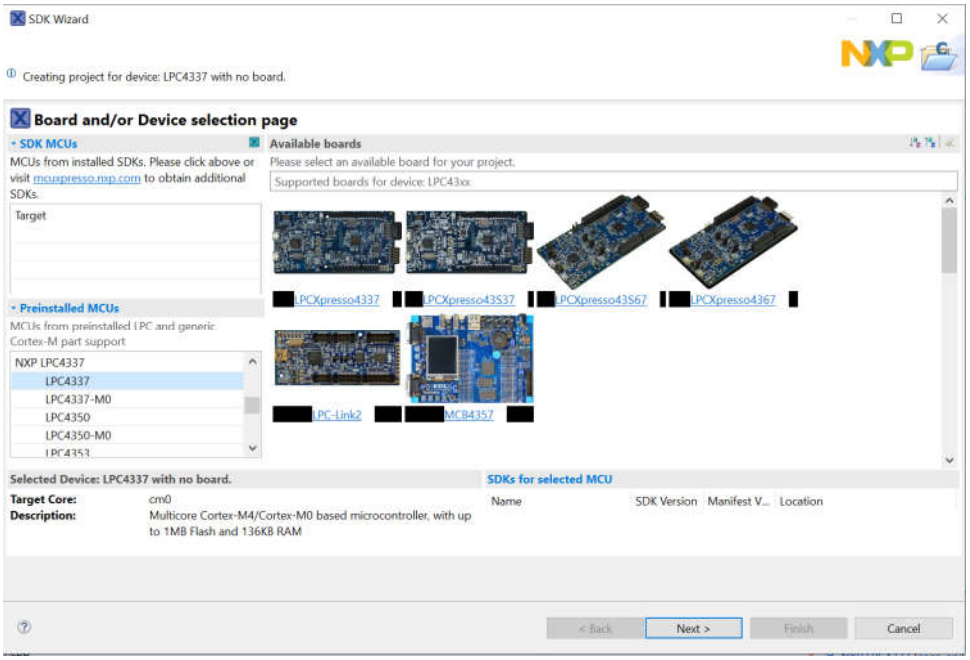
Escenario multicore

Una de las principales ventajas que tiene el LPC4337 es que posee un Cortex M4 y un Cortex M0 en el mismo encapsulado. Cabe destacar que el primero será siempre el maestro y el encargado de habilitar al Cortex M0, siendo este último un co-procesador.

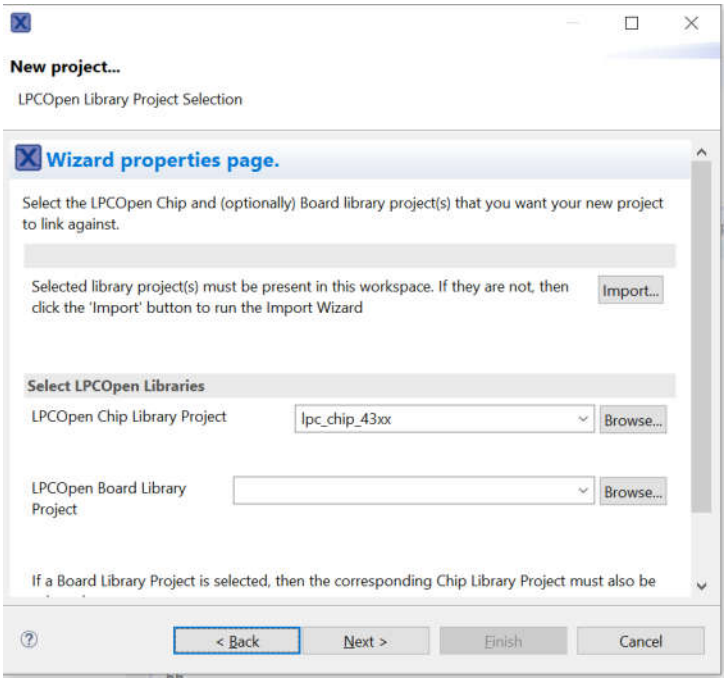
Para poder programar un escenario multicore, debemos tener en cuenta que vamos a tener tantos proyectos creados como núcleos se deseen programar. Con esta librería tenemos la libertad de programar tanto el Cortex M4 como el Cortex M0 con las mismas funciones, sin embargo, al momento de generar el proyecto o mismo ya generado, debemos mapear la memoria donde se ejecutarán sus respectivos programas.

Entonces, en principio tenemos que crear primero los proyectos de los núcleos esclavos y por último el del maestro, razón por la cual el ejemplo del módulo anterior fue hecho con el Cortex M0. Luego una práctica recomendable es eliminar toda memoria flash de los esclavos para que su programa se ejecute sobre la RAM, posición de memoria que deberá compartir con el maestro.

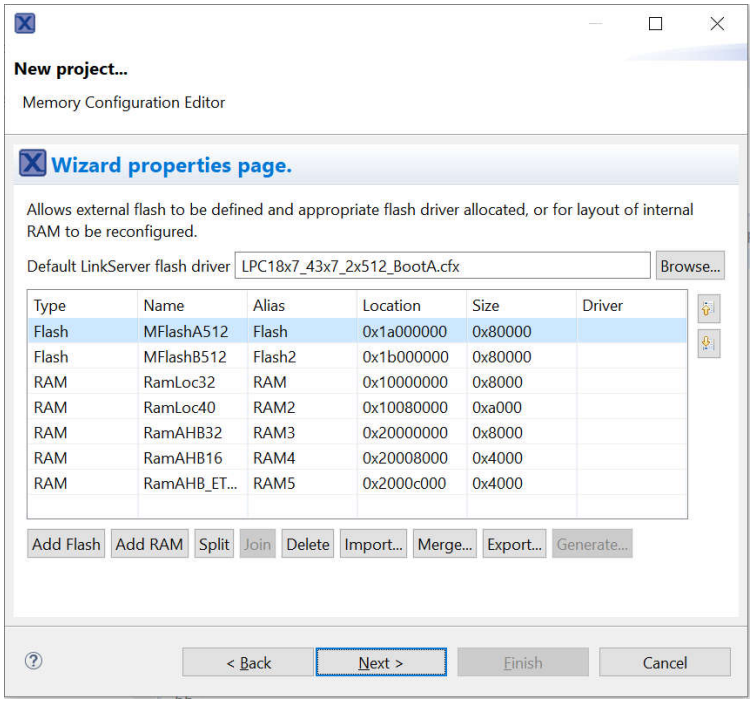
Explicado esto, procederé a crear un proyecto para el Cortex M4 que se llamará ejemploM4 de la misma forma que lo hicimos con el M0 pero acá seleccionaremos LPC4337.



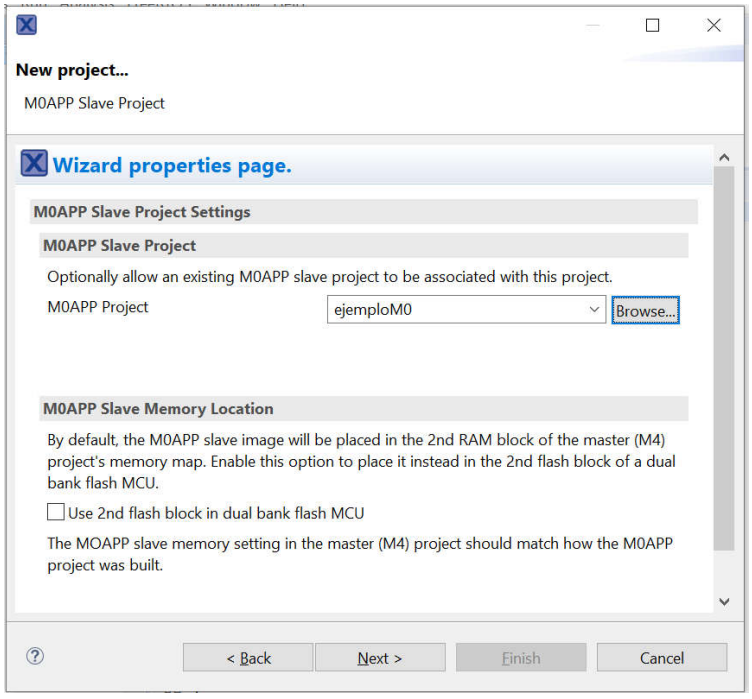
Y luego, dentro de LPC43xx Multicore M4 seleccionaremos LPCOpen – C ++ Project. Cuando haya que seleccionar la librería LPCOpen ya debería aparecer por defecto la del LPC4337.



El mapeo de memoria quedará el que aparece por defecto.

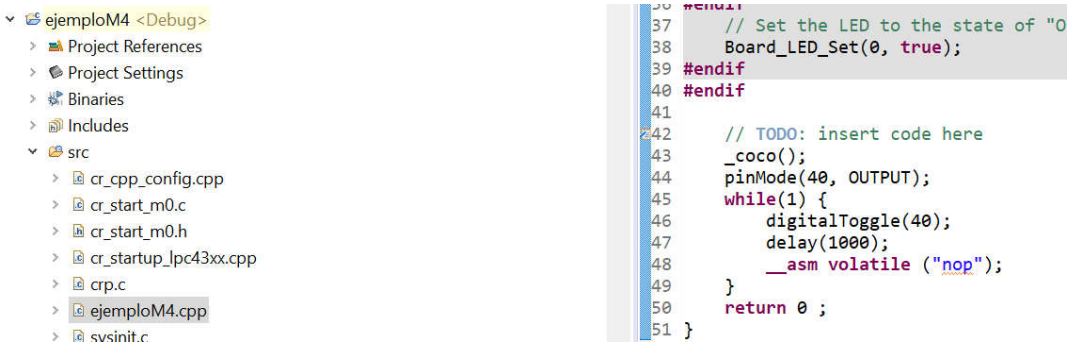


Luego se nos pedirá asociar con un proyecto para el Cortex M0, aquí seleccionaremos el ejemploM0.

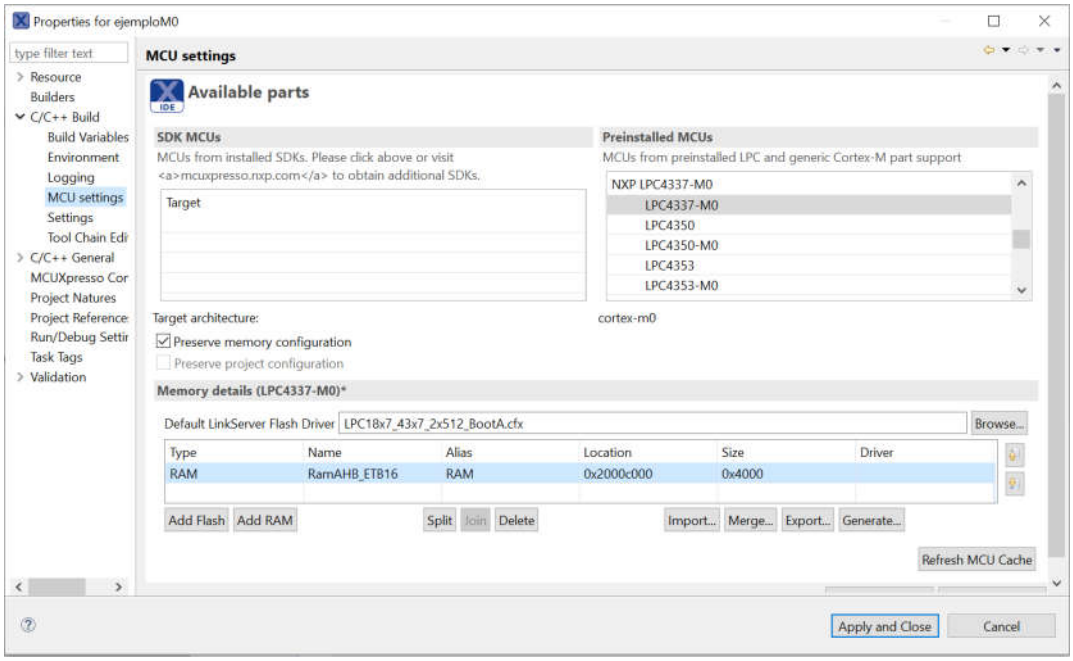


Ya con el proyecto creado generaremos el enlace con la librería coco_api como se explicó en el módulo anterior dentro de las Properties del ejemploM4. Recuerde de hacerlo con la correspondiente al Cortex M4.

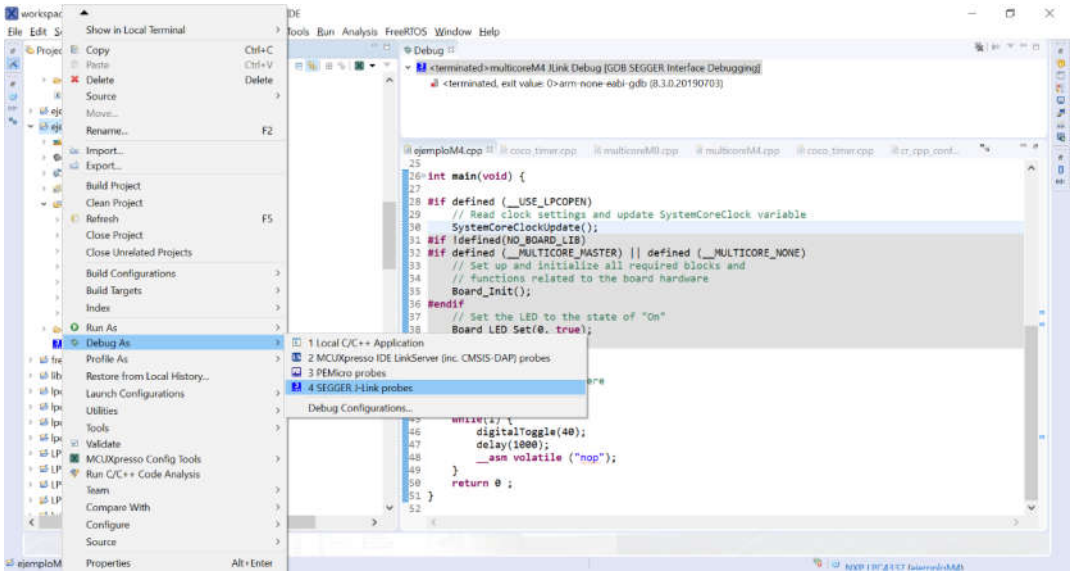
Por otro lado, al finalizarlo restará incluir coco.h en el programa principal y escribir el siguiente código para hacer un blink con Led2 de la EDU-CIAA.



Antes de programar la EDU-CIAA habrá que configurar el mapeo de memoria del proyecto del Cortex M0. Para hacerlo presionaremos botón derecho sobre ejemploM0 e ingresaremos en Properties. Dentro de allí, nos dirigiremos a la siguiente ventana y eliminaremos todas las memorias excepto una RAM – verificando que soporte el tamaño del código que deseamos cargar -. En mi caso deje la de menor tamaño, pero siempre deberá coincidir la posición inicial con una de las del maestro.



Finalmente, para programar la EDU-CIAA, presionaremos botón derecho sobre el proyecto ejemploM4, luego en Debug As > SEGGER J-Link y el mismo se encargará de programar ambos núcleos.



Al terminar la programación no olvide dirigirse a Run > Resume para ejecutar el programa. Al hacerlo, deberán estar parpadeando tanto LED1 como LED2, pero como nosotros sabemos, al primero lo estará haciendo modificar su estado el Cortex M0 y al segundo el M4.

API COCO

Como vimos anteriormente y se explicó, es una librería que está enfocada a poder programar la EDU-CIAA con las mismas funciones que en Arduino, es por ello por lo que se programaron un abanico de funciones para cada módulo que permitirán ejecutar diferentes acciones en ambos Cores con el mismo nombre de función tanto para uno como para el otro. Es decir, ahora con el código de Arduino, lo puedo copiar y pegarlo tanto para un núcleo como para el otro.

A continuación, se hará una explicación de cada función, aunque también están comentadas en el código fuente y desde el Github se podrá encontrar con programas ejemplo para ejecutarlos. Al hacerlo no olvide enlazar la librería desde las Properties del proyecto.

Módulo GPIO

Este módulo incluido en “coco_gpio” nos será de utilidad para inicializar, configurar y establecer pines digitales INPUT/OUTPUT.

Aquí se asociaron los pines de la EDU-CIAA con su propia etiqueta a un número de pin para ser utilizado con la librería. En la tabla siguiente se puede observar y asociar el número de pin con la etiqueta correspondiente. Además, con el símbolo ~ se indicará cuáles de los pines se podrán usar para PWM en su correspondiente librería.

Número puerto	Número pin	Número GPIO	Número pin GPIO	Pin digital API	Etiqueta EDU-CIAA
1	15	0	2	0	RXD0
3	7	5	10	1	SPI_MOSI
4	6	2	6	2	LCD_EN
6	1	3	0	3	GPIO0
6	5	3	4	4~	GPIO2
6	8	5	16	5	GPIO4
6	10	3	6	6	GPIO6
6	12	2	8	7~	GPIO8
6	11	3	7	8	GPIO7
6	9	3	5	9	GPIO5
6	7	5	15	10	GPIO3
6	4	3	3	11	GPIO1
4	4	2	4	12	LCD1
4	5	2	5	13	LCD2
4	6	2	6	14	LCD3
4	8	5	12	15	LCD_RS
4	10	5	14	16	LCD4
1	3	0	10	17	SPI_MISO
1	20	0	15	18	TXD1
1	18	0	13	19	TXD0
1	17	0	12	20	MDIO
1	16	0	3	21	CRS_DV
7	7	3	15	22	MDC
0	1	0	1	23	TX_EN
0	0	0	0	24	RXD1
2	4	5	4	25	232_RX
2	3	5	3	26	232_TX
3	1	5	8	27	CAN_RD
3	2	5	9	28	CAN_TD
7	4	3	12	29~	T_COL1
4	0	2	0	30	T_FIL0
4	3	2	3	31~	T_FIL3
4	2	2	2	32~	T_FIL2
1	5	1	8	33~	T_COL0
4	1	2	1	34~	T_F1
7	5	3	13	35~	T_C2
2	0	5	0	36	LED_R
2	1	5	1	37	LED_G
2	2	5	2	38	LED_B
2	10	0	14	39~	LED1
2	11	1	11	40~	LED2
2	12	1	12	41~	LED3

Dentro de esta librería tendremos las siguientes funciones:

- pinMode: Esta función nos permitirá configurar un pin digital para poder utilizado como Input o como Output. Si deseamos colocar el pin 30 como entrada, la función será llamada

de la siguiente manera.

```
pinMode(30, INPUT)
```

- digitalWrite: Esta función nos permitirá configurar un pin digital para poder establecerlo en estado 1 o 0 digital. Si deseamos colocar el pin 20 en 1, la función será llamada de la siguiente manera.

```
digitalWrite(20, HIGH)
```

- digitalRead: Esta función nos permitirá conocer el estado de un pin digital. Si deseamos saber el estado del pin 12 y guardarlo en una variable bool, la función será llamada de la siguiente manera.

```
bool variable = digitalRead(12)
```

- digitalToggle: Esta función nos permitirá invertir el estado de un pin digital. Si deseamos invertir el estado del pin 8, la función será llamada de la siguiente manera.

```
digitalToggle(8)
```

Módulo ADC

En este microcontrolador LPC4337 contamos con dos ADC de 10 bits de aproximación sucesiva con posibilidad de lectura de 0v hasta VDDA que no puede superar los 3,3V por seguridad. La frecuencia de muestreo puede ser de hasta 400 KMuestras/segundo.

Este módulo incluido en “coco_adc” nos será de utilidad para iniciar, configurar y establecer el módulo ADC y sus canales. En particular se han definido 8 canales que fueron etiquetados desde A0 hasta A7.

En la siguiente tabla podremos observar la definición en esta librería y a que canal está asociado tanto para el microcontrolador como para la EDU-CIAA. Hay que aclarar que si bien fueron definidos en la librería, los canales del 4 al 7 no fueron facilitados en la plataforma EDU-CIAA, por lo cual usted podrá ver solamente desde el canal 0 hasta el 3 en dicha placa.

Si al canal elegido, a la hora de utilizar la función le sumamos 8, podremos trabajar con el ADC1. Por default, las funciones están programadas para trabajar con el ADC0.

Número de PIN en LPC4337 QFP144	Definición en API	Etiqueta EDU-CIAA
6	A0	DAC
2	A1	ADC0_1
143	A2	ADC0_2
139	A3	ADC0_3
138	A4	-
144	A5	-
142	A6	-
136	A7	-

Aquí la única función con la que nos encontraremos será:

- analogRead: Esta función nos permitirá conocer el valor de lectura de un cierto canal del ADC0 o 1. Cabe destacar que al llamar la función _coco() solo se inicializa el ADC0, por lo que de necesitar ADC1 lo deberá inicializar y sumarle 8 al canal deseado. Veamos un ejemplo para cada ADC suponiendo que los dos están inicializados.

  Para el ADC0 leamos el canal 2

  ```
  analogRead(A2)
  ```

  Para el ADC1 leamos el canal 3:

  ```
  analogRead(A3 + 8)
  ```

Módulo SCT

Este módulo lo utilizaremos para generar PWM. Para la EDU-CIAA tendremos 11 pines accesibles para generarlo. Estos están marcados en la tabla de la sección “Módulo GPIO”.

Al llamar a la función _coco() ya se inicializará el módulo con una frecuencia de 50Hz. Fue creada una función para modificarla, pero si se desea puede hacerlo desde coco_pwm.h y cambiar el valor de frec_init a otro en la unidad Hz.

La única función que tendremos disponible aquí será:

- analogWrite: Esta función nos permitirá variar el dutty de un pin de PWM entre un valor de 0 a 255. Por lo que, si queremos un dutty del 50% en el pin 4 de la API, llamaremos a la función de la siguiente manera.

  ```
  analogWrite(4, 128)
  ```

Módulo TIMER

Este módulo será utilizado para generar retardos y tomar tiempos en diferentes órdenes de magnitud.

En principio, hay que conocer que se programó un delay que utilizará el Timer 0 para el Cortex M4 y el Timer 1 para el Cortex M0 por lo que quedarán inhabilitados para otras funciones.

Las funciones disponibles en este módulo coco_timer serán:

- delay: Generará un retardo en milisegundos según el parámetro que se indique. Para llamar a esta función y generar un retardo de 500 mS, se deberá hacer de la siguiente manera.

  ```
  delay(500)
  ```

Módulo UART

Este módulo será utilizado para la comunicación UART. En particular, el LPC4337 posee tres USART y cada uno de estos ya tiene una función asignada en la EDU-CIAA pero esto puede ser modificado si asi se requiere.

En la EDU-CIAA cada USART fue asignado como lo indica la siguiente tabla.

USART	EDU-CIAA	Definición en API
0	RS485	Serial
2	USB	Serial2
3	RS232	Serial1

Para cada uno de estos tendremos las mismas funciones y estas serán:

- xxx.begin: Esta función nos será de utilidad para inicializar y configurar el USART deseado a una velocidad en baudios requerida. Cabe destacar que fue programada para establecer el USART en 8 bits con 1 bit de stop y sin paridad. De querer modificar esto último, podrá hacerlo desde “coco_uart.cpp”.
  Si deseamos inicializar el Serial2 en 9600 baudios la función deberá ser llamada de la siguiente manera.

```
Serial2.begin(9600)
```

- xxx.available: Esta función nos permitirá conocer si hay información disponible para ser leída en el buffer del USART en cuestión. Cabe destacar que esta función siempre deberá ser llamada antes de leer el dato.
  Esta devolverá un 1 un caso de haber información, por lo que por lo general será utilizada dentro de un condicional. Si deseamos guardar el estado del buffer del Serial en una variable podemos llamar a la función de la siguiente manera.

```
bool variable = Serial.available()
```

- xxx.write: Esta función nos permitirá enviar un byte o más por el USART indicado. Si deseamos enviar “Hola” por el Serial1 la función deberá ser llamada de la siguiente manera.

```
Serial.write(“Hola”)
```

- xxx.read: Esta función nos permitirá obtener un byte disponible en el buffer del USART indicado. Si deseamos leer un byte del buffer del Serial2 la función deberá ser llamada de la siguiente manera.

```
char c = Serial2.read()
```

- xxx.readString: Esta función nos permitirá obtener una cadena de caracteres disponible en el buffer del USART indicado. Recuerde que necesitará utilizar previamente la función available(). Si deseamos leer una cadena de caracteres del Serial la función deberá ser llamada de la siguiente manera:

```
String lectura = Serial.readString()
```

Módulo I2C

Este módulo nos permitirá una comunicación I2C con dispositivos externos mediante la librería coco_wire. Para este caso se usará el I2C0 que es el facilitado en la EDU-CIAA y las funciones disponibles serán las siguientes.

- Wire.begin: Inicializa el I2C0 a una velocidad de 400KHz para el Clock y con capacidad de modificarse desde “coco_wire.h” y con Pull-Up en SDA y SCL. Por otro lado, se deshabilitarán las interrupciones. Para inicializarlo la función deberá ser llamada de la siguiente manera.

```
Wire.begin()
```

- Wire.beginTransmission: Inicializa la comunicación con un esclavo indicando el Address del mismo. Por ejemplo, si deseamos comunicarnos con el MPU6050 que tiene Address 0x68 la función deberá ser llamada de la siguiente manera.

```
Wire.beginTransmission(0x68)
```

- Wire.endTransmission: Finaliza la comunicación I2C. Para ser utilizada la función deberá ser llamada de la siguiente manera.

```
Wire.endTransmission()
```

- Wire.write: Permitirá el envío de uno o más bytes por I2C con el esclavo con el cual se inició transmisión. Para enviar el número 30 por I2C la función deberá ser llamada de la siguiente manera.

```
Wire.write(30)
```

- Wire.requestFrom: Solicitará la cantidad de bytes indicadas al esclavo explicitado como parámetro. Para solicitarle 2 bytes al MPU6050 lo haremos llamando a la función de la siguiente manera.

```
Wire.requestFrom(0x68, 2)
```

- Wire.available: Nos hará conocer si existe información disponible en el buffer de lectura. Para conocer la cantidad de bytes en buffer y guardarlo en una variable la función deberá ser llamada de la siguiente manera.

```
uint8_t variable = Wire.available()
```

- Wire.read: Permitirá leer un byte del buffer de I2C. Para hacerlo antes no olvidemos llamar a requestFrom y luego la función read deberá ser llamada de la siguiente manera.

```
char c = Wire.read()
```

- Wire.setClock: Con esta función podremos configurar la velocidad del clock de I2c. Esta velocidad como indicamos anteriormente la inicializaremos en 400 KHz pero puede modificarse. Si se desea setearla en 100 KHz, la función deberá ser llamada de la siguiente manera.

```
Wire.setClock(100000)
```

Módulo SPI

Con este módulo podremos enviar y recibir información utilizando el protocolo SPI el cual además esta facilitado en la plataforma EDU-CIAA. La inicialización que se optó en esta librería fue con 8 bits pero se puede modificar desde "coco_spi.cpp".

Las funciones programadas para este módulo son las siguientes.

- SPI.begin: Inicializa los pines de SPI con Pull-Up y el SSP. Para hacerlo deberá llamar a la función de la siguiente manera.

```
SPI.begin()
```

- SPI.setDataMode: Esta función nos será de utilidad para setear el uso de 8 bits y uno de los cuatro modos disponibles los cuales se pueden ver en la siguiente tabla.

Definición en API	CPHA	CPOL
SPI_MODE0	0	0
SPI_MODE1	1	0
SPI_MODE2	0	1
SPI_MODE3	1	1

Para establecer el modo 0, lo deberá hacer de la siguiente manera.

```
SPI.setDataMode(SPI_MODE0)
```

- SPI.setClockDivider: Esta función nos será de utilidad para setear el divisor para la frecuencia del Clock. Tendremos 7 divisores que se pueden apreciar en la siguiente tabla.

Definición en API	Divisor
SPI_CLOCK_DIV2	8000000
SPI_CLOCK_DIV4	4000000
SPI_CLOCK_DIV8	2000000
SPI_CLOCK_DIV16	1000000
SPI_CLOCK_DIV32	500000
SPI_CLOCK_DIV64	250000
SPI_CLOCK_DIV128	125000

Cabe destacar que la máxima velocidad posible es de 25,5 Mb/s en full duplex.

UTN F r a

Secretaría de Ciencia, Tecnología y Posgrado

Secretaria Académica

BIENESTAR universitario

SEMANA DE LA INVESTIGACIÓN EN INGENIERÍA

AV. RAMÓN FRANCO 5050. VILLA DOMINICO
PATIO DE COMIDAS UTN FRA

NIMBUS III

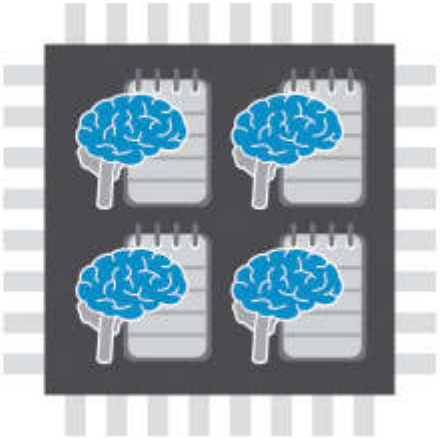
¿De qué se trata el PID?

"Diseño de nuevo modelo de estación agrometeorológica automática portátil NIMBUS III basado en sistemas embebidos para el estudio del microclima en parcelas de ensayo"

NIMBUS III

¿Innovación?

Estación agrometeorológica con aplicación/utilización de microcontroladores multi core en paralelo con sensores de precisión.



NIMBUS III

Proyecto CIAA

Comunidad Argentina colaborando en el desarrollo de este proyecto para la industria nacional.

Computadora
Industrial
Abierta
Argentina

EXPERIENCIA PERSONAL

COLABORACIÓN EN EL PID

Librerías para el uso de microcontroladores asimétricos multi core de la familia LPC con iguales funciones que Arduino

ARDUINO + CIAA Computadora Industrial Abierta Argentina

EXPERIENCIA PERSONAL

COLABORACIÓN EN UTN FRA

Librería y documentación que aportará a las cátedras TDII, TDIII, NIMBUSIII y al Laboratorio de Sistemas Embebidos