
Manual de configuración del entorno de construcción de MicroPython en Windows 10

Versión 1.0

**Carlos Daniel Hernández Miranda
Jonatan Emanuel Salinas Ávila
Ana María Ruiz Fernández**

23 de Febrero, 2022

Tabla de contenido

1. Introducción	2
2. Herramientas a utilizar	2
3. Instalación de herramientas para la compilación	2
3.1 <i>Instalación de MinGW W64</i>	2
3.2 <i>Instalación de Cygwin</i>	6
4. Compilación de MicroPython para la tarjeta Ophyra	8
4.1 <i>Obteniendo el código fuente de MicroPython</i>	8
4.2 <i>Configuraciones previas para compilar</i>	8
4.3 <i>Proceso de compilación</i>	9
5. Puesta a prueba del firmware en la tarjeta Ophyra	11
6. Inserción de las librerías no nativas en MicroPython	12
Anexo 1. Modificación de los archivos de configuración de la Ophyra para compilar el firmware de MicroPython 1.14	15
Anexo 2. Compatibilidad con los archivos de compilación del STM32F405 en la versión 1.15	19
Anexo 3. Modificación de archivos para la compilación de firmware en la versión 1.18 de Micropython	23

1. Introducción

En este manual se describen los pasos a seguir para configurar un entorno de construcción y compilación de MicroPython en una computadora con sistema operativo Windows 10. Esto con el objetivo de generar el firmware necesario (los archivos .hex* y .dfu*) para una tarjeta de desarrollo con un microcontrolador capaz de ejecutar el intérprete de MicroPython. En el presente documento se toma como referencia la tarjeta de desarrollo Ophyra, fabricada y ensamblada por Intesc Electrónica y Embebidos. A la última fecha de modificación de este reporte, la última versión de MicroPython es la 1.14.

2. Herramientas a utilizar

A continuación, se presenta una lista de las herramientas que serán instaladas en las siguientes secciones de este documento, con el fin de implementar el entorno de compilación para MicroPython en Windows 10:

- MinGW64
 - GNU Compiler Collection (para C y C++)
 - gcc-arm-none-eabi
 - GDB para C y C++
 - Python
- Cygwin
 - gcc-g++
 - make

3. Instalación de herramientas para la compilación

3.1 Instalación de MinGW W64

En principio, es necesario instalar la implementación del compilador GCC para Windows, llamado MinGW64. Para ello, se ingresa a la página: <http://mingw-w64.org/doku.php/download> y se da clic en “Msys2” para descargar MSYS2. El link redirigirá a otra página, en donde será posible descargar el instalador de este programa.





 MacPorts	Rolling	macOS	8.2.0/5.0.4
MingW-W64-builds	Rolling	Windows 	7.2.0/5.0.3
 Msys2	Rolling	Windows 	9.2.0/trunk
		12.04 Precise Pangolin	4.6.3/2.0.1
		14.04 Trusty Tahr	4.8.2/3.1.0

Imagen 1. Link que redirige a la página de descarga de Msys2.

Installation

- Download the installer: [msys2-x86_64-20210105.exe](#)
 Verify with SHA256 checksum `c6cbaad7f3e939ec9024ed6a6a36b0271`
 or GPG signature by `0xf7a49b0ec`.
- Run the installer. MSYS2 requires 64 bit Windows 7 or newer.
- Enter your desired **Installation Folder** (short ASCII-only path, no accented characters or network drives).

Imagen 2. Link para descargar el instalador de Msys2.

Después de descargar, ejecutar el archivo **.exe** y terminar el proceso de instalación. Al dar clic en el botón “Finish”, se abrirá la terminal MSYS2. Ahí, ejecutar el comando: **pacman -Syu** para actualizar los paquetes. Una vez actualizados, la terminal se cerrará. Ahora, es necesario correr “MSYS2 MSYS” desde el menú principal de Windows.

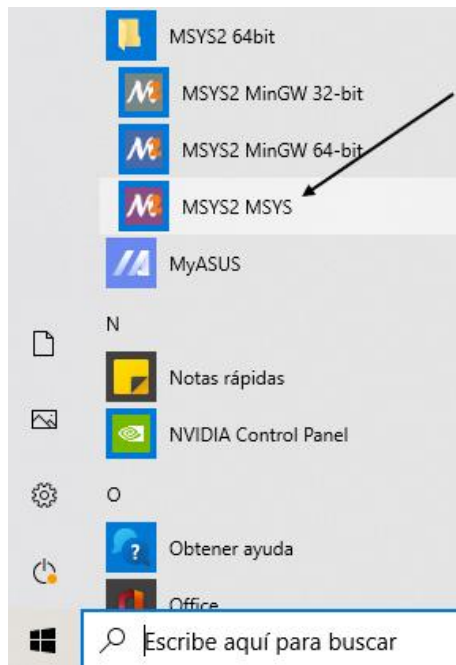


Imagen 3. MSYS2 MSYS en el menú principal de Windows.

Cuando haya abierto la terminal, correr el comando: **pacman -Su** para terminar de actualizar los paquetes. Una vez terminado el proceso, cerrar la terminal y abrir “MSYS2 MinGW” (seleccionar la opción de 32-bits o la de 64-bits dependiendo del sistema operativo que esté corriendo en la computadora).

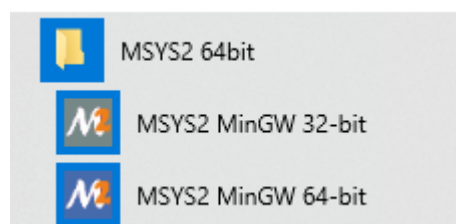


Imagen 4. MSYS2 MinGW en el menú principal de Windows.

Una vez abierta la terminal de MSYS2 MinGW, ahora es necesario instalar la colección de compiladores de GNU para C y C++, las herramientas de GNU para procesadores embebidos ARM, y el depurador gdb para C y C++. Para esto, se ejecutarán los siguientes comandos en la terminal, uno por uno:

Si el SO es de 64 bits:

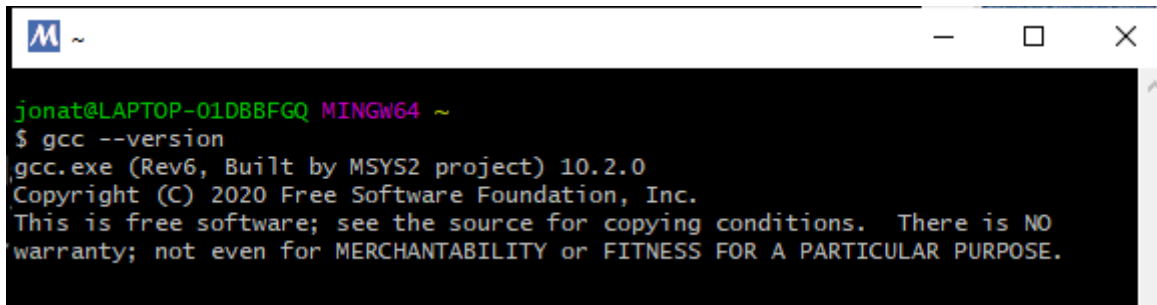
- Colección de compiladores de GNU para C y C++ :
pacman -S mingw-w64-x86_64-gcc
- Herramientas de GNU para procesadores embebidos ARM:

```
pacman -S mingw-w64-x86_64-arm-none-eabi-gcc
```

- Depurador gdb para C y C++:

```
pacman -S mingw-w64-x86_64-gdb
```

Para verificar que la instalación se ha realizado con éxito, es posible ejecutar el comando: **gcc --version**, para ver la versión instalada de gcc:



```
jonat@LAPTOP-01DBBFGQ MINGW64 ~  
$ gcc --version  
gcc.exe (Rev6, Built by MSYS2 project) 10.2.0  
Copyright (C) 2020 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Imagen 5. Se verifica la instalación de gcc con “gcc --version”.

Ahora, es necesario verificar que se reconozca el comando “gcc” en cmd.exe de nuestro sistema Windows. Para ello, abrir cmd.exe y ejecutar el comando: **gcc --version**. Si el comando no es reconocido, es necesario agregar el PATH donde se encuentra gcc.exe a las variables del entorno del sistema. La localización de la carpeta depende de dónde se instaló MSYS, pero el PATH puede ser parecido al siguiente: **C:\msys64\mingw64\bin**. Una vez copiado el PATH, buscar en el menú de Windows: “Editar las variables de entorno del sistema”, y dar clic. En “Opciones Avanzadas”, dar clic en “Variables de entorno”, en “Variables del sistema” buscar “Path” y dar clic en “Editar”:

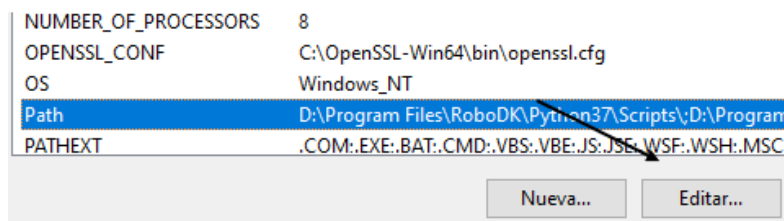


Imagen 6. Es necesario editar las variables de entorno.

Dar clic en “Nuevo”, y pegar el PATH copiado anteriormente. Finalmente, dar clic en “Aceptar” para aplicar los cambios en todas las pestañas. Para verificar que los cambios hayan actuado de forma correcta, ejecutar en cmd.exe el comando: **gcc --version**:

```

Microsoft Windows [Versión 10.0.18363.1377]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\jonat>gcc --version
gcc (Rev6, Built by MSYS2 project) 10.2.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

C:\Users\jonat>

```

Imagen 7. Verificación de la instalación de gcc en Windows.

Al dar clic “Enter”, debería aparecer la versión instalada de gcc para Windows. Es absolutamente relevante mencionar que para generar la compilación del firmware de MicroPython, es importante tener Python en el ordenador. Afortunadamente, MinGW nos provee de Python 3.8, el cual se encuentra también en el PATH mencionado anteriormente. Es así, que al realizar el paso anterior de agregar el PATH a las Variables de entorno, ya no es necesario tener una instalación de Python externa en la computadora.

	Nombre	Fecha de modificación	Tipo	Tamaño
do	python	18/01/2021 03:09 p. m.	Aplicación	96 KB
	python3.8	18/01/2021 03:09 p. m.	Aplicación	96 KB
	python3.8-config	18/01/2021 03:09 p. m.	Archivo 8-CONFIG	4 KB
os	python3	18/01/2021 03:09 p. m.	Aplicación	96 KB
	python3-config	18/01/2021 03:09 p. m.	Archivo	4 KB
os a man	python3w	18/01/2021 03:09 p. m.	Aplicación	94 KB
ummer Ru	python-config	18/01/2021 03:09 p. m.	Archivo	4 KB
	pythonw	18/01/2021 03:09 p. m.	Aplicación	94 KB
	pzstd	11/01/2021 01:12 p. m.	Aplicación	134 KB
	ranlib	01/02/2021 06:52 p. m.	Aplicación	2,247 KB
	readelf	01/02/2021 06:52 p. m.	Aplicación	2,062 KB

Imagen 8. MingGW provee Python 3.8, necesario para la compilación del firmware.

El siguiente tutorial muestra los pasos anteriormente descritos para la instalación de MinGW: <https://www.youtube.com/watch?v=0HD0pqVtsmw&t=625s>

3.2 Instalación de Cygwin

Ahora bien, es necesario instalar la terminal Cygwin64, que es un programa que emula un ambiente Unix en Windows. Permite usar funcionalidades comunes de los sistemas Unix/Linux

en entornos Windows. Para ello, es necesario ingresar a: <https://www.cygwin.com/> , y descargar y correr el instalador de este programa:

Installing Cygwin

Install Cygwin by running [setup-x86_64.exe](#)

Imagen 8. Link de descarga del instalador de Cygwin.

Dar clic en “Siguiente” para continuar con la instalación, hasta llegar a la siguiente ventana. Al llegar a la sección de Instalación de paquetes, buscar: **gcc-g++**, y **make** ; y seleccionar una versión de éstos para instalar, como se muestra en la siguiente imagen:

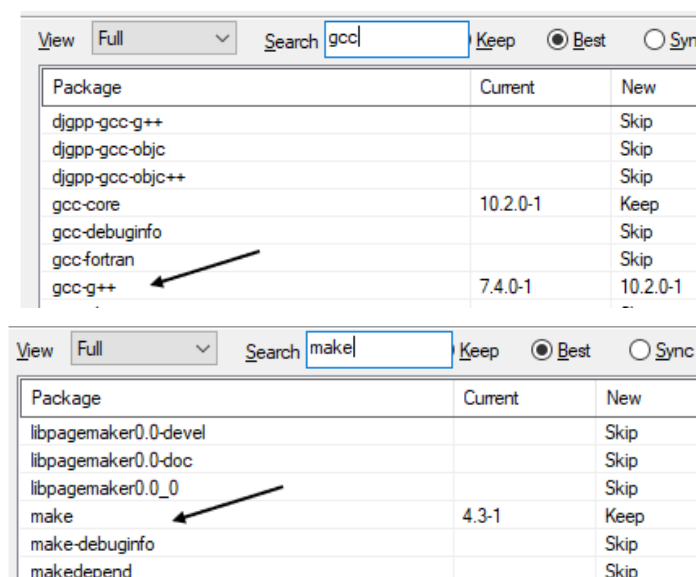


Imagen 9. Selección de paquetes útiles para la instalación con Cygwin.

Dar clic en “Siguiente”, para continuar con la instalación. Finalmente, dar clic en “Finish”, para finalizar. Cygwin estará ahora instalado en el sistema.

4. Compilación de MicroPython para la tarjeta Ophyra

Una vez hecho todos los pasos anteriores comenzaremos el proceso de compilación para obtener los archivos **firmware.hex** y **firmware.dfu** que son los archivos necesarios para poder ejecutar MicroPython en la tarjeta Ophyra.

4.1 Obteniendo el código fuente de MicroPython

Existen múltiples formas de obtener el código fuente de MicroPython, desde la página oficial: <https://micropython.org/download/>, o desde el repositorio de la plataforma Github en el siguiente enlace: <https://github.com/micropython/micropython/tags>

MicroPython downloads

MicroPython is developed using git for source code management, and the master repository can be found on GitHub at github.com/micropython/micropython.

The full source-code distribution of the latest version is available for download here:

- [micropython-1.14.tar.xz](#) (54MiB)
- [micropython-1.14.zip](#) (104MiB)

Daily snapshots of the GitHub repository (not including submodules) are available from this server:

- [micropython-master.zip](#)
- [pyboard-master.zip](#)

Firmware for various microcontroller ports and boards are built automatically on a daily basis and can be found in the pictured sections below. Alternatively, a list of all available firmware is [here](#).

Imagen 10. Página oficial de descarga del código fuente de Micro Python.

Una vez descargado el archivo “**micropython-1.14.zip**” lo que haremos por último será descomprimirlo.

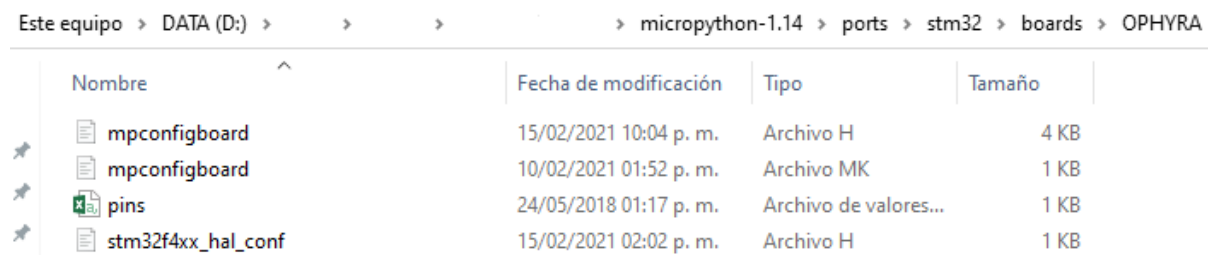
4.2 Configuraciones previas para compilar

Para compilar MicroPython para la tarjeta Ophyra, es necesaria la carpeta de los archivos de configuración de esta tarjeta de desarrollo. Esta carpeta, la cual debe llevar el nombre de “OPHYRA”, debe poseer cuatro archivos de configuración, los cuales son los siguientes:

- mpconfigboard.h
- mpconfigboard.mk

- pins.csv
- stm32f4xx_hal_conf.h

Esta carpeta OPHYRA con estos archivos de configuración, debe ser puesta dentro de la carpeta del código fuente de MicroPython descargada anteriormente, específicamente en el Path: `/ports/stm32/boards/`.



Nombre	Fecha de modificación	Tipo	Tamaño
mpconfigboard	15/02/2021 10:04 p. m.	Archivo H	4 KB
mpconfigboard	10/02/2021 01:52 p. m.	Archivo MK	1 KB
pins	24/05/2018 01:17 p. m.	Archivo de valores...	1 KB
stm32f4xx_hal_conf	15/02/2021 02:02 p. m.	Archivo H	1 KB

Imagen 11. Contenido de la carpeta OPHYRA dentro de la carpeta boards.

El Anexo 1 de este documento especifica ciertos cambios que tuvieron que ser hechos en los archivos de configuración de la tarjeta Ophyra, cuando se trató de actualizar el firmware que tenía la tarjeta Ophyra (MicroPython 1.11) a la versión más reciente (MicroPython 1.14).

4.3 Proceso de compilación

Para comenzar con el proceso de compilación previamente es necesario compilar el contenido de la carpeta `/mpy-cross`. Para ello, a través de Cygwin se navega hasta dicho folder, y ahí se debe ejecutar el siguiente comando:

```
make STRIP=echo SIZE=echo
```

Al ejecutar este comando es importante mencionar que es posible que surja un error, debido a que quizá no se encuentre el compilador gcc para hacer la compilación. En caso de que el comando de arriba arroje dicho error, es posible usar el comando siguiente:

```
make STRIP=echo SIZE=echo CROSS_COMPILE=x86_64-w64-mingw32-
```

Terminado el proceso, una vez compilado `/mpy-cross` necesitamos posicionarnos a través de Cygwin en la carpeta `/ports/stm32/` (observemos el árbol de directorio de la carpeta MicroPython):

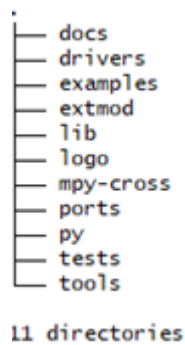


Imagen 12. Árbol del directorio del código fuente de MicroPython

Y empezaremos el proceso de compilación del firmware de MicroPython para la tarjeta Ophyra, ejecutando el siguiente comando:

```
make BOARD=OPHYRA -j4
```

El comando **-j4** hace alusión al número de tareas en paralelo que necesitamos ejecutar.

Ahora simplemente esperaremos que se compilen todos los archivos y posterior enlazamiento de los archivos de compilación para generar el **.hex** y el **.dfu** se tendrá como resultado:

```
LINK build-OPHYRA/firmware.elf
   text  data   bss   dec    hex filename
 333428   16  27112 360556  5806c build-OPHYRA/firmware.elf
GEN build-OPHYRA/firmware0.bin
GEN build-OPHYRA/firmware1.bin
GEN build-OPHYRA/firmware.hex
GEN build-OPHYRA/firmware.dfu
```

Con estos archivos generados, hemos tenido una compilación exitosa del software MicroPython, y es posible ahora programar el firmware generado en la tarjeta Ophyra.

5. Puesta a prueba del firmware en la tarjeta Ophyra

Para probar el nuevo firmware en la tarjeta Ophyra es necesario tener el programa Stm32CubeProgrammer, por lo que primero debemos de poner la tarjeta de modo bootloader para poder leer la memoria flash de la misma, una vez hecho esto en la tarjeta Ophyra procederemos a conectar y grabar el **firmware.hex** en la misma y ver si el firmware funciona correctamente.

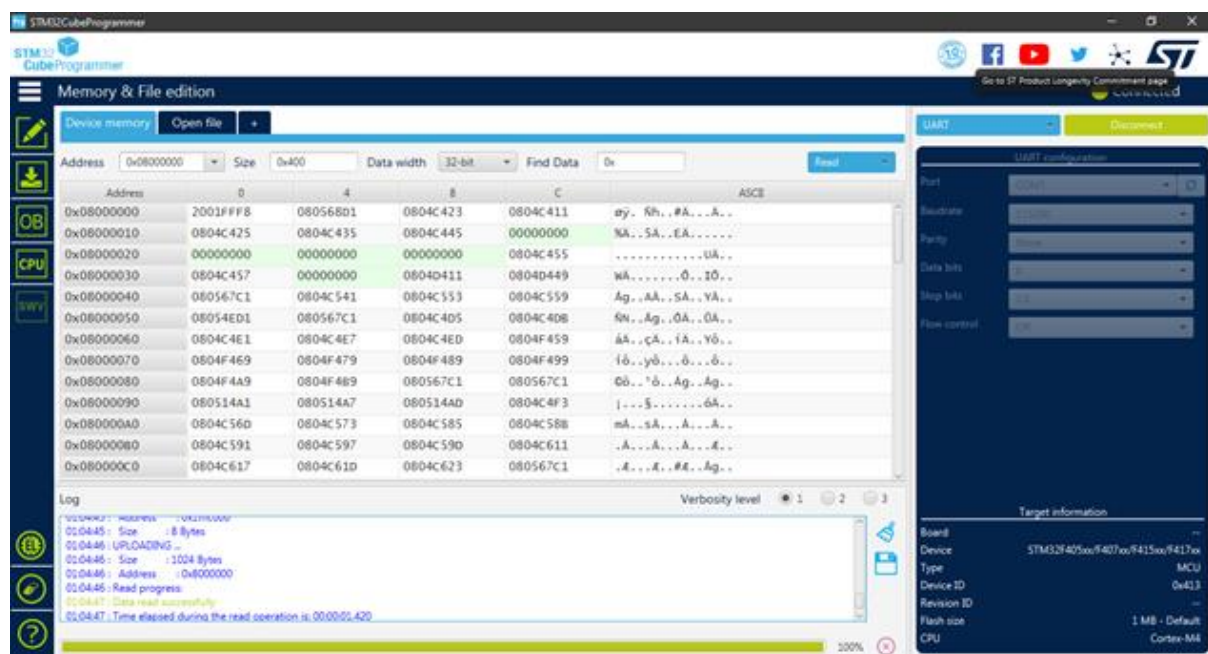


Imagen 13. Reescritura del Firmware en la tarjeta Ophyra a través de Stm32CubeProgrammer.

Ahora para verificar en la tarjeta a través de la conexión USB/OTG si MicroPython funciona correctamente, haremos uso de un emulador de consola llamado PuTTY para poder conectarnos con la tarjeta Ophyra, en este caso tenemos que verificar el puerto COM (Nombre del puerto serial) de programación de la tarjeta; por defecto en cada computadora se crea un identificador de puerto diferente, en este caso es el COM7 y nos conectaremos a una velocidad de 115200 baudios. Podemos observar en la siguiente imagen que la tarjeta funciona correctamente con MicroPython versión 1.14:

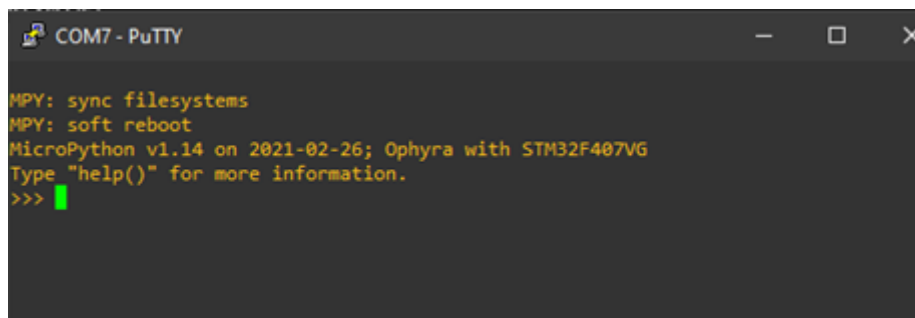


Imagen 14: MicroPython ejecutándose satisfactoriamente en Ophyra.

Ahora solo ejecutaremos algunas funciones básicas para ver si en la tarjeta tienen el funcionamiento correcto de manera que tengamos las primera impresiones del funcionamiento del firmware.

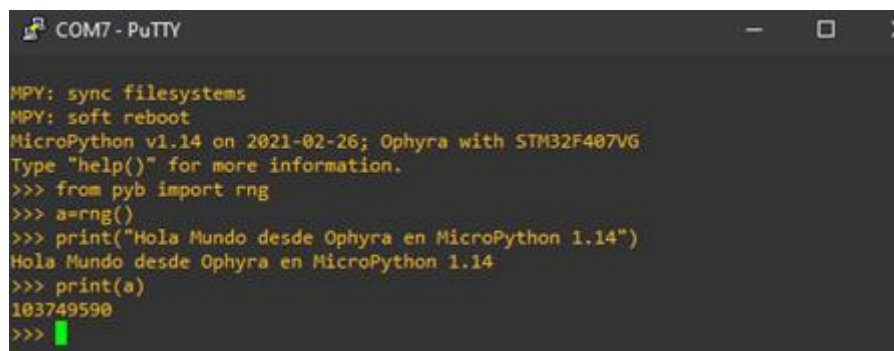


Imagen 15. Probando funciones y librerías nativas en MicroPython.

Observamos que en el REPL¹ de MicroPython funciona correctamente algunas librerías como la librería de números aleatorios y funciones como la impresión de texto en pantalla, por lo que tenemos las primeras impresiones aceptables.

6. Inserción de las librerías no nativas en MicroPython

Como hemos visto anteriormente, la compilación del archivo **.hex** se ha obtenido con éxito de tal forma que el siguiente paso es añadir las librerías escritas en lenguaje Python para que se puedan ejecutar en la tarjeta Ophyra. Estas librerías sirven para que el microcontrolador de la tarjeta pueda interactuar con otros elementos de esta, como la memoria EEPROM integrada, el sensor MPU6050, los botones de propósito general, etc. Estas estarán almacenadas en la

¹ Read Event Print Loop – Bucle de Lectura, Evaluación e Impresión

memoria RAM del microcontrolador a través de un código intermedio llamado bytecode que se incluye en el archivo **firmware.hex**.

Ahora, para incluir las librerías en Python de acuerdo con el archivo Makefile de compilación para la familia de tarjetas con microcontroladores Stm32:

```
FROZEN_MANIFEST ?= boards/manifest.py
```

Para agregar las librerías en la compilación lo primero que haremos será situarnos dentro de `ports/stm32/` y crearemos la carpeta “modules” que contendrá todas las librerías en Python que deseemos agregar:

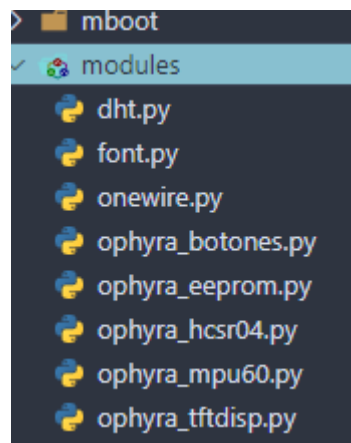


Imagen 16. Directorio `ports/stm32/modules` con librerías escritas en Python.

En `ports/stm32/boards` se encuentra el archivo `manifest.py` que enlaza la dirección de las librerías escritas en Python y genera el bytecode en la compilación del firmware de MicroPython. En tal archivo anexamos las librerías de la siguiente manera:

```
freeze("${MPY_DIR}/ports/stm32/modules", "font.py")
freeze("${MPY_DIR}/ports/stm32/modules", "onewire.py")
freeze("${MPY_DIR}/ports/stm32/modules", "ophyra_botones.py")
freeze("${MPY_DIR}/ports/stm32/modules", "ophyra_eeprom.py")
freeze("${MPY_DIR}/ports/stm32/modules", "ophyra_mpu60.py")
freeze("${MPY_DIR}/ports/stm32/modules", "ophyra_tftdisp.py")
freeze("${MPY_DIR}/ports/stm32/modules", "ophyra_hcsr04.py")
```

- **freeze():** Es la función que anexa las librerías a la compilación en la que previamente crea un bytecode que utiliza Gcc para poder anexar el código Python al firmware.
- **MPY_DIR:** Es la variable que sabe la ubicación del directorio MicroPython en este caso `c:/micropython-1.14`

`freeze(dir, file)` : esta función sólo recibe dos parámetros: la dirección donde están alojados los módulos y el nombre del módulo que se anexara como se ve en el ejemplo de código de la parte superior.

Para generar el firmware con las librerías escritas en Python, primero antes que nada se tiene que limpiar el objetivo de compilación porque previamente compilamos una vez, esto se hace con el siguiente comando en `ports/stm32/`:

```
make clean BOARD=OPHYRA
```

Al ingresar este comando en Cygwin, se arroja el siguiente resultado:

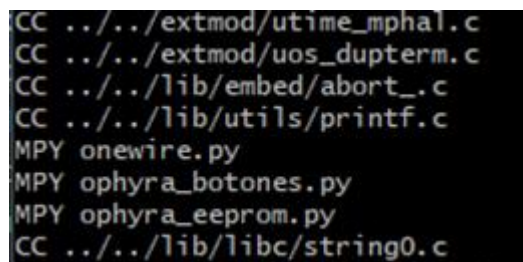
```
Use make V=1 or set BUILD_VERBOSE in your environment to increase
build verbosity.
```

```
rm -rf build-OPHYRA
```

Ahora una vez limpiado el objetivo ejecutamos:

```
make BOARD=OPHYRA -j8
```

En la siguiente imagen, se puede apreciar que las librerías indicadas en el archivo `manifest.py` son incluidas en el firmware que se está construyendo:



```
CC ../../extmod/utime_mphal.c
CC ../../extmod/uos_dupterm.c
CC ../../lib/embed/abort.c
CC ../../lib/utis/printf.c
MPY onewire.py
MPY ophyra_botones.py
MPY ophyra_eeprom.py
CC ../../lib/libc/string0.c
```

Imagen 17. Librerías en Python agregadas correctamente al proceso de compilación.

Ahora solo esperamos a que se termine el proceso de compilación para probar las librerías en la tarjeta Ophyra.

```
LINK build-OPHYRA/firmware.elf
text      data      bss      dec      hex filename
577944    4544      27100   609588   94d34 build-OPHYRA/firmware.elf
GEN build-OPHYRA/firmware.dfu
GEN build-OPHYRA/firmware.hex
```

Imagen 18. Generación del firmware.hex con las librerías en Python.

El archivo `firmware.hex` ahora puede ser programado en la tarjeta Ophyra, para ejecutar MicroPython con las librerías en Python ya integradas.

Anexo 1. Modificación de los archivos de configuración de la Ophyra para compilar el firmware de MicroPython 1.14

En este anexo se describen los pasos de las modificaciones que se llevaron a cabo en los archivos de configuración cuando se quiso actualizar el firmware de la tarjeta Ophyra de la antigua versión de MicroPython (1.11) a la más reciente a la fecha de creación de este reporte (1.14).

Hubo que crear un nuevo archivo `stm32f4xx_hal_conf.h` en la carpeta OPHYRA basándonos en el archivo que se aloja en la dirección `/ports/stm32/boards/PYBV10/` debido a que el procesador utilizado en la tarjeta Ophyra es muy similar al de PYBV10. El contenido del archivo `stm32f4xx_hal_conf.h` que debe ir dentro de la carpeta OPHYRA se muestra a continuación:

```
#ifndef MICROPY_INCLUDED_STM32F4XX_HAL_CONF_H
#define MICROPY_INCLUDED_STM32F4XX_HAL_CONF_H

#include "boards/stm32f4xx_hal_conf_base.h"

// Oscillator values in Hz
#define HSE_VALUE (8000000)
#define LSE_VALUE (32768)
#define EXTERNAL_CLOCK_VALUE (12288000)

// Oscillator timeouts in ms
#define HSE_STARTUP_TIMEOUT (100)
#define LSE_STARTUP_TIMEOUT (5000)

#endif // MICROPY_INCLUDED_STM32F4XX_HAL_CONF_H
```

Viendo el código de la parte superior es como quedaría la configuración de la tarjeta Ophyra en el archivo `stm32f4xx_hal_conf.h`.

Ahora en el archivo "boards/stm32f4xx_hal_conf_base.h" deshabilitamos algunas macros que no funcionan en Ophyra para que el compilador pueda enlazar la configuración correctamente y obtener el **.hex**

```
// Enable various HAL modules
#define HAL_ADC_MODULE_ENABLED
#define HAL_CAN_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
//#define HAL_CRC_MODULE_ENABLED
#define HAL_DAC_MODULE_ENABLED
#define HAL_DCMI_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
//#define HAL_ETH_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
//#define HAL_HASH_MODULE_ENABLED
//#define HAL_HCD_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
//#define HAL_I2S_MODULE_ENABLED
//#define HAL_IWDG_MODULE_ENABLED
#define HAL_PCD_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_RTC_MODULE_ENABLED
#define HAL_SD_MODULE_ENABLED
//#define HAL_SDRAM_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
#define HAL_TIM_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED
//#define HAL_USART_MODULE_ENABLED
//#define HAL_WWDG_MODULE_ENABLED
```

Las macros comentadas con “//” son las que no necesitamos para compilar correctamente Ophyra. Por último copiaremos el contenido del archivo **stm32f405.ld** y lo alojaremos en un nuevo archivo llamado **stm32f407.ld**, el contenido del archivo es el siguiente:

```
/*
GNU linker script for STM32F405
*/
/* Specify the memory areas */
MEMORY
{
    FLASH (rx)      : ORIGIN = 0x08000000, LENGTH = 1024K /* entire flash */
}
```

```

FLASH_ISR (rx) : ORIGIN = 0x08000000, LENGTH = 16K /* sector 0 */
FLASH_FS (rx)  : ORIGIN = 0x08004000, LENGTH = 112K /* sectors 1,2,3,4 are for filesystem
*/
FLASH_TEXT (rx) : ORIGIN = 0x08020000, LENGTH = 896K /* sectors 5,6,7,8,9,10,11 */
CCMRAM (xrw)   : ORIGIN = 0x10000000, LENGTH = 64K
RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 128K
}

/* produce a link error if there is not this amount of RAM for these sections */
_minimum_stack_size = 2K;
_minimum_heap_size = 16K;

/* Define the stack. The stack is full descending so begins just above last byte
of RAM. Note that EABI requires the stack to be 8-byte aligned for a call. */
_estack = ORIGIN(RAM) + LENGTH(RAM) - _estack_reserve;
_sstack = _estack - 16K; /* tunable */

/* RAM extents for the garbage collector */
_ram_start = ORIGIN(RAM);
_ram_end = ORIGIN(RAM) + LENGTH(RAM);
_heap_start = _ebss; /* heap starts just after statically allocated memory */
_heap_end = _sstack;

```

Recordemos también que necesitamos del archivo **stm32f407_af.csv**. Estos dos archivos, **stm32f407.ld** y **stm32f407_af.csv**, tienen que estar alojados en la carpeta `/ports/stm32/boards/`. Es también importante recalcar cierta modificación que se realizó en el archivo **stm32f407_af.csv**, ya que al intentar compilar el firmware de MicroPython 1.14, es común que se presentará el siguiente error:

```

In file included from pin_static_af.h:31,
                 from qspi.c:33:
build-OPHYRA/genhdr/pins_af_defs.h:405:9: error: ISO C99 requires whitespace after the macro
name [-Werror]
  405 | #define STATIC_AF_JTMS-SWDIO_NULL(pin_obj) ( \
      |           ^~~~~~
build-OPHYRA/genhdr/pins_af_defs.h:409:9: error: ISO C99 requires whitespace after the macro
name [-Werror]
  409 | #define STATIC_AF_JTCK-SWCLK_NULL(pin_obj) ( \
      |           ^~~~~~
cc1.exe: all warnings being treated as errors
In file included from pin_static_af.h:31,

```

Para solucionar este error se corrigió el archivo de configuración de la tarjeta **stm32f407_af.csv** en las líneas 16 y 17 se tuvieron que corregir las configuraciones del modo depuración de la tarjeta quedando de la forma siguiente:

```
PortA,PA13,JTMS/SWDIO,,,,,,,,,,,,EVENTOUT,
```

```
PortA,PA14,JTCK/SWCLK,,,,,,,,,,,,EVENTOUT,
```

Debido a que el proceso de compilación se detuvo debido a este error volveremos a iniciarlo con el comando:

```
make BOARD=OPHYRA -j4
```

Otro cambio que fue necesario realizar fue en el archivo `mpconfigboard.h` dentro de la carpeta OPHYRA, en donde se sustituyó el macro:

```
#define MICROPY_HW_HAS_SDCARD      (1)
```

Por este otro:

```
#define MICROPY_HW_ENABLE_SDCARD   (1)
```

Este cambio fue necesario ya que la tarjeta Ophyra no detectaba la tarjeta SD a pesar de que sí estaba insertada. Al realizar este cambio se corrigió el problema.

Es importante mencionar también que al realizar la revisión del nuevo firmware 1.14 con las prácticas entregables del curso de Intesc “Programación de microcontroladores ARM en lenguaje MicroPython”, salieron a la luz ciertos defectos que se corrigieron con algunos de los cambios mencionados anteriormente (particularmente el del archivo `stm32f4xx_hal_conf.h` y el del archivo `mpconfigboard.h`)

Anexo 2. Compatibilidad con los archivos de compilación del STM32F405 en la versión 1.15

Por otro lado el MCU de la tarjeta OPHYRA el STM32F407VG es idéntico a nivel de hardware con el STM32F405 es por eso que los archivos de configuración de pines y de memoria pueden ser usados para compilar MicroPython de forma nativa sin la necesidad de crear nuevos archivos de configuración.

Para eso se tienen que hacer cambios en el archivo `mpconfigboard.h` quedando de la siguiente forma:

```
#define MICROPY_HW_BOARD_NAME      "Ophyra"
#define MICROPY_HW_MCU_NAME        "STM32F407VG"

#define MICROPY_HW_HAS_SWITCH      (1)
#define MICROPY_HW_HAS_FLASH      (1)
#define MICROPY_HW_HAS_MMA7660    (0)
#define MICROPY_HW_HAS_LIS3DSH    (0)
#define MICROPY_HW_HAS_LCD        (0)
#define MICROPY_HW_ENABLE_RNG      (1)
#define MICROPY_HW_ENABLE_RTC      (1)
#define MICROPY_HW_ENABLE_SERVO    (1)
#define MICROPY_HW_ENABLE_DAC      (1)
#define MICROPY_HW_ENABLE_USB      (1)
#define MICROPY_HW_ENABLE_SDCARD   (1)
#define MODULE_OPHYRA_MPU607_ENABLED (1)
#define MODULE_OPHYRA_EEPROM_ENABLED (1)
#define MODULE_OPHYRA_BOTONES_ENABLED (1)
#define MODULE_OPHYRA_HCSR04_ENABLED (1)
#define MODULE_OPHYRA_TFTDISP_ENABLED (1)
// HSE is 8MHz
#define MICROPY_HW_CLK_PLLM (8)
#define MICROPY_HW_CLK_PLLN (336)
#define MICROPY_HW_CLK_PLLP (RCC_PLLP_DIV2)
#define MICROPY_HW_CLK_PLLQ (7)

// UART config
// A9 is used for USB VBUS detect, and A10 is used for USB_FS_ID.
// UART1 is also on PB6/7 but PB6 is tied to the Audio SCL line.
// Without board modifications, this makes UART1 unusable on this board.
#define MICROPY_HW_UART1_TX        (pin_A9)
#define MICROPY_HW_UART1_RX        (pin_A10)
#define MICROPY_HW_UART2_TX        (pin_A2)
#define MICROPY_HW_UART2_RX        (pin_A3)
#define MICROPY_HW_UART2_RTS       (pin_A1)
#define MICROPY_HW_UART2_CTS       (pin_A0)
#define MICROPY_HW_UART3_TX        (pin_B10)
#define MICROPY_HW_UART3_RX        (pin_B11)
#define MICROPY_HW_UART3_RTS       (pin_B13)
#define MICROPY_HW_UART3_CTS       (pin_B12)
#if MICROPY_HW_HAS_SWITCH == 0
```

```

#define MICROPY_HW_UART4_TX      (pin_A0)
#define MICROPY_HW_UART4_RX      (pin_A1)
#endif

// I2C busses
#define MICROPY_HW_I2C1_SCL (pin_B6)
#define MICROPY_HW_I2C1_SDA (pin_B7)
#define MICROPY_HW_I2C2_SCL (pin_B10)
#define MICROPY_HW_I2C2_SDA (pin_B11)

// SPI-1 bus
#define MICROPY_HW_SPI1_NSS (pin_A15) //Puerto SPI1 exclusivo para pantalla TFT
#define MICROPY_HW_SPI1_SCK (pin_B3)
#define MICROPY_HW_SPI1_MISO (pin_B4)
#define MICROPY_HW_SPI1_MOSI (pin_B5)
// SPI-2 bus
#define MICROPY_HW_SPI2_NSS (pin_B12)
#define MICROPY_HW_SPI2_SCK (pin_B13)
#define MICROPY_HW_SPI2_MISO (pin_B14)
#define MICROPY_HW_SPI2_MOSI (pin_B15)
// CAN busses
// #define MICROPY_HW_CAN1_NAME "YA"
#define MICROPY_HW_CAN1_TX (pin_D1) // Y4
#define MICROPY_HW_CAN1_RX (pin_D0) // Y3

// USRSW is pulled low. Pressing the button makes the input go high.

#define MICROPY_HW_USRSW_PIN (pin_C2)
#define MICROPY_HW_USRSW_PULL (GPIO_PULLUP)
#define MICROPY_HW_USRSW_EXTI_MODE (GPIO_MODE_IT_FALLING)
#define MICROPY_HW_USRSW_PRESSED (0)

// LEDs
#define MICROPY_HW_LED1 (pin_E0) // red
#define MICROPY_HW_LED2 (pin_E1) // green
#define MICROPY_HW_LED3 (pin_E2) // blue
#define MICROPY_HW_LED_ON(pin) (mp_hal_pin_low(pin))
#define MICROPY_HW_LED_OFF(pin) (mp_hal_pin_high(pin))

// USB config
#define MICROPY_HW_USB_FS (1)
#define MICROPY_HW_USB_VBUS_DETECT_PIN (pin_A9)
#define MICROPY_HW_USB_OTG_ID_PIN (pin_A10)

#define MICROPY_HW_SDCARD_DETECT_PIN (pin_C6)
#define MICROPY_HW_SDCARD_DETECT_PULL (GPIO_PULLUP)
#define MICROPY_HW_SDCARD_DETECT_PRESENT (GPIO_PIN_RESET)

```

También se necesitará añadir una nueva librería al archivo `stm32f4xx_hal_conf_base.h` y descomentar las demás librerías del archivo que habíamos comentado en el Anexo 1 quedando de la siguiente forma:

```

#include "stm32f4xx_hal_dma.h"
#include "stm32f4xx_hal_adc.h"
#include "stm32f4xx_hal_can.h"
#include "stm32f4xx_hal_cortex.h"
#include "stm32f4xx_hal_crc.h"
#include "stm32f4xx_hal_dac.h"
#include "stm32f4xx_hal_dcmi.h"
#include "stm32f4xx_hal_eth.h"
#include "stm32f4xx_hal_flash.h"
#include "stm32f4xx_hal_gpio.h"
#include "stm32f4xx_hal_hash.h"
#include "stm32f4xx_hal_hcd.h"
#include "stm32f4xx_hal_i2c.h"
#include "stm32f4xx_hal_i2s.h"
#include "stm32f4xx_hal_iwdg.h"
#include "stm32f4xx_hal_pcd.h"
#include "stm32f4xx_hal_pwr.h"
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal_rtc.h"
#include "stm32f4xx_hal_sd.h"
#include "stm32f4xx_hal_sdram.h"
#include "stm32f4xx_hal_spi.h"
#include "stm32f4xx_hal_tim.h"
#include "stm32f4xx_hal_uart.h"
#include "stm32f4xx_hal_usart.h"
#include "stm32f4xx_hal_wwdg.h"
#include "stm32f4xx_ll_adc.h"
#include "stm32f4xx_ll_pwr.h"
#include "stm32f4xx_ll_rtc.h"
#include "stm32f4xx_ll_usart.h"

```

```

// Enable various HAL modules
#define HAL_ADC_MODULE_ENABLED
#define HAL_CAN_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
#define HAL_CRC_MODULE_ENABLED
#define HAL_DAC_MODULE_ENABLED
#define HAL_DCMI_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_ETH_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_HASH_MODULE_ENABLED
#define HAL_HCD_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
#define HAL_I2S_MODULE_ENABLED
#define HAL_IWDG_MODULE_ENABLED
#define HAL_PCD_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_RTC_MODULE_ENABLED
#define HAL_SD_MODULE_ENABLED
#define HAL_SDRAM_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
#define HAL_TIM_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED

```

```
#define HAL_USART_MODULE_ENABLED
#define HAL_WWDG_MODULE_ENABLED
```

Por último, el archivo `mpconfigboard.mk` quedara de la siguiente manera para que use los archivos de STM32F405:

```
MCU_SERIES = f4
CMSIS_MCU = STM32F407xx
AF_FILE = boards/stm32f405_af.csv
LD_FILES = boards/stm32f405.ld boards/common_ifs.ld
TEXT0_ADDR = 0x08000000
TEXT1_ADDR = 0x08020000
```

Teniendo todos estos cambios seremos capaces de compilar sin ningún problema con los archivos de memoria y pines del STM43F405.

Anexo 3. Modificación de archivos para la compilación de firmware en la versión 1.18 de MicroPython

Al revisar los archivos de la última versión de MicroPython, se observó que se mantiene la compatibilidad a nivel de hardware con el STM32F405. Por lo tanto, los archivos de configuración de OPHYRA no sufrieron ningún cambio en este proceso.

Por otro lado, se realizaron algunos cambios en el archivo `stm32f4xx_hal_conf_base.h`, que se encuentra en la carpeta `ports/stm32/boards/boards` de MicroPython. En la versión 18 en este archivo incluye un módulo HAL para el controlador MMC. Debido a que OPHYRA no cuenta con este controlador, es necesario desactivarlo para evitar errores de configuración en la tarjeta.

Así pues, se comentaron las líneas 45 y 78 de este archivo. Así pues, la sección de inclusión y activación de módulos de este archivo—líneas 30 a 89—, queda de la siguiente forma:

```
// Include various HAL modules for convenience
#include "stm32f4xx_hal_dma.h"
#include "stm32f4xx_hal_adc.h"
#include "stm32f4xx_hal_can.h"
#include "stm32f4xx_hal_cortex.h"
#include "stm32f4xx_hal_crc.h"
#include "stm32f4xx_hal_dac.h"
#include "stm32f4xx_hal_dcmi.h"
#include "stm32f4xx_hal_eth.h"
#include "stm32f4xx_hal_flash.h"
#include "stm32f4xx_hal_gpio.h"
#include "stm32f4xx_hal_hash.h"
#include "stm32f4xx_hal_hcd.h"
#include "stm32f4xx_hal_i2c.h"
#include "stm32f4xx_hal_i2s.h"
#include "stm32f4xx_hal_iwdg.h"
// #include "stm32f4xx_hal_mmc.h"
#include "stm32f4xx_hal_pcd.h"
#include "stm32f4xx_hal_pwr.h"
#include "stm32f4xx_hal_rcc.h"
#include "stm32f4xx_hal_rtc.h"
#include "stm32f4xx_hal_sd.h"
#include "stm32f4xx_hal_sdram.h"
#include "stm32f4xx_hal_spi.h"
#include "stm32f4xx_hal_tim.h"
#include "stm32f4xx_hal_uart.h"
#include "stm32f4xx_hal_usart.h"
#include "stm32f4xx_hal_wwdg.h"
#include "stm32f4xx_ll_adc.h"
#include "stm32f4xx_ll_pwr.h"
#include "stm32f4xx_ll_rtc.h"
#include "stm32f4xx_ll_usart.h"
```



```
// Enable various HAL modules
#define HAL_ADC_MODULE_ENABLED
#define HAL_CAN_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
#define HAL_CRC_MODULE_ENABLED
#define HAL_DAC_MODULE_ENABLED
#define HAL_DCMI_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
#define HAL_ETH_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_HASH_MODULE_ENABLED
#define HAL_HCD_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
#define HAL_I2S_MODULE_ENABLED
#define HAL_IWDG_MODULE_ENABLED
// #define HAL_MMC_MODULE_ENABLED
#define HAL_PCD_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_RTC_MODULE_ENABLED
#define HAL_SD_MODULE_ENABLED
#define HAL_SDRAM_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
#define HAL_TIM_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED
#define HAL_USART_MODULE_ENABLED
#define HAL_WWDG_MODULE_ENABLED
```