Punto 1.7

7. Documentar mediante tablas c/texto e imágenes la secuencia de comandos: Clean firmware_v2 ->Build firmware_v2 ->Debug firmware_v2 ->Ejecutar gpio_02_blinky (ejemplo de aplicación)

a. Completo (Resume), detener (Suspend) y resetear (Restart)

b. Por etapas colocando breakpoints (Resume)

c. Por línea de código (Step Into, Step Over, Step Return)

d. Recuerde siempre abandonar Debug (Terminate) antes de Editar o Compilar algún archivo, o Abandonar el IDE (Exit)

---

7) Para poder ejecutar y depurar el código se debe seleccionar el proyecto (figura 1)

```
#-------------------------------------------------------------------------------
# current project
#-------------------------------------------------------------------------------

#------------ examples ---------------------------------------

#PROJECT = examples/blinky
#PROJECT = examples/blinky_rit
#PROJECT = examples/adc_fir_dac
#PROJECT = examples/freertos_blinky
#PROJECT = examples/statechart

#------------ sapi_examples ------------------------------------

#------------ Bare-metal examples ------------

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/gpio/gpio_01_switches_leds
PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/gpio/gpio_02_blinky
#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/gpio/gpio_03_blinky_switch
#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/gpio/gpio_04_led_sequences

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/keypad_7segment_01
#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/lcd_01

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/uart/uart_01_echo
#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/uart/uart_02_receive_string_blocking
#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/uart/uart_03_receive_string

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/stdio_01_printf_sprintf

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/adc_dac_01

#PROJECT = sapi_examples/edu-ciaa-nxp/bare_metal/cycles_counter_01
```

Figura 1

Se procede a limpiar todos los archivos fuente compilados seleccionando *clean project* (figura 2). Debería verse un mensaje como el de la figura 3.

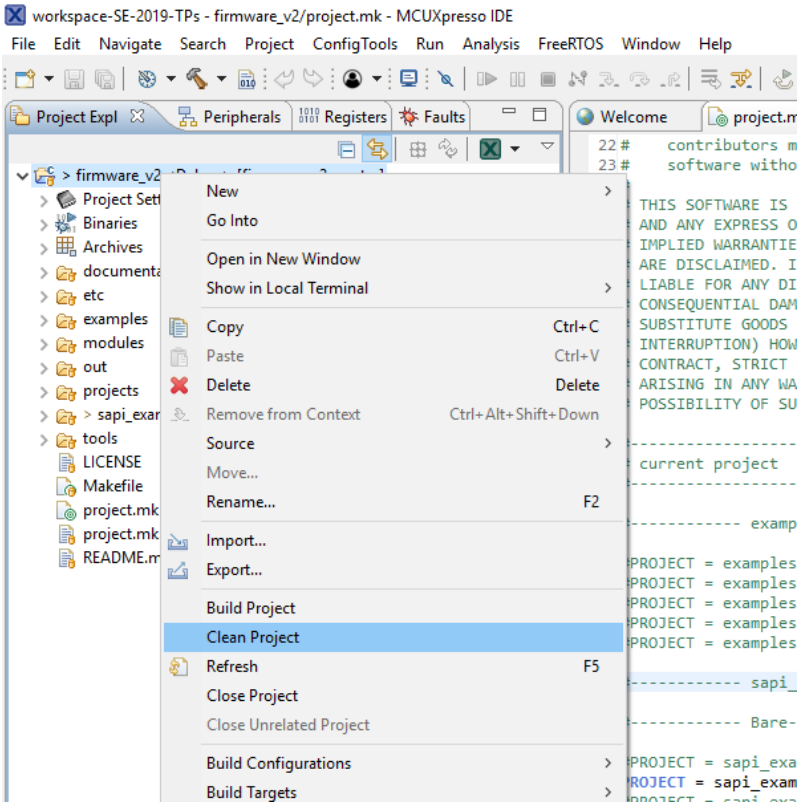

Figura 2

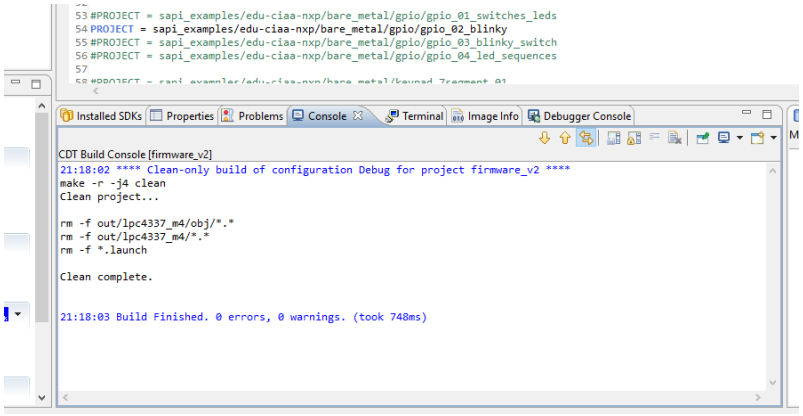

Figura 3

Para compilar todos los archivos del proyecto se usa *build project* (figura 4). Si todo salió bien debería verse lo de la figura 5.

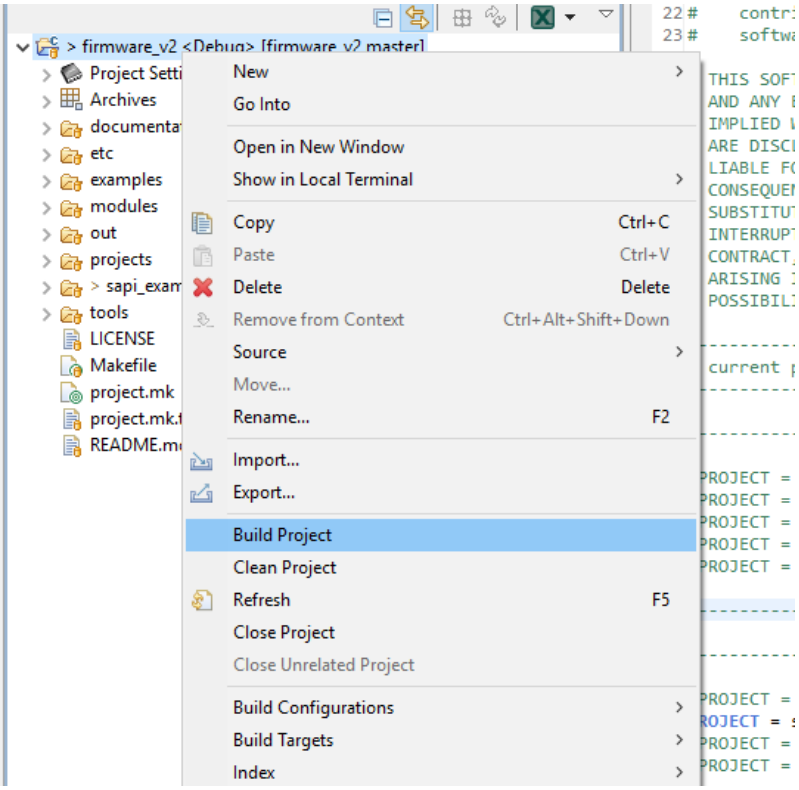

Figura 4

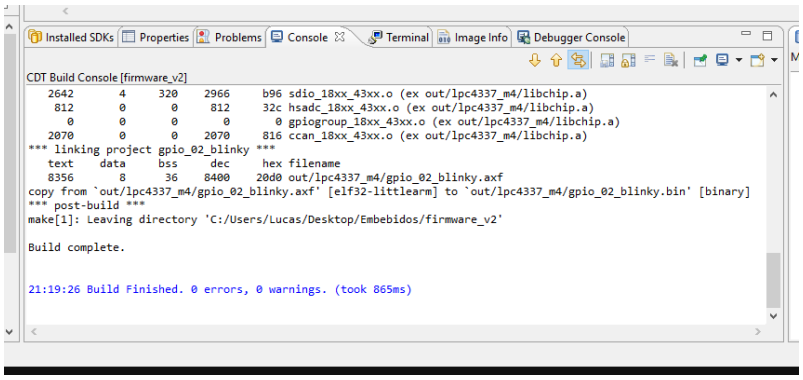

Figura 5

Para subir el código y empezar la depuración se debe asegurar que la aplicación seleccionada es la correcta. Para esto se debe ir a *debug configuration* (figura 6) y en *C/C++ application* se debe elegir la que corresponde al proyecto usado (figura 7).

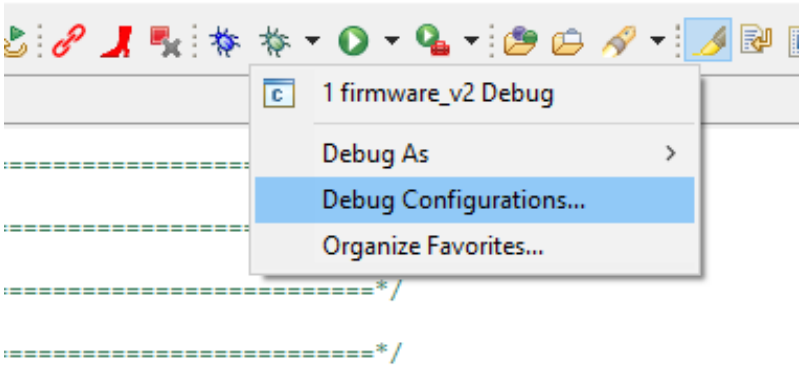

Figura 6

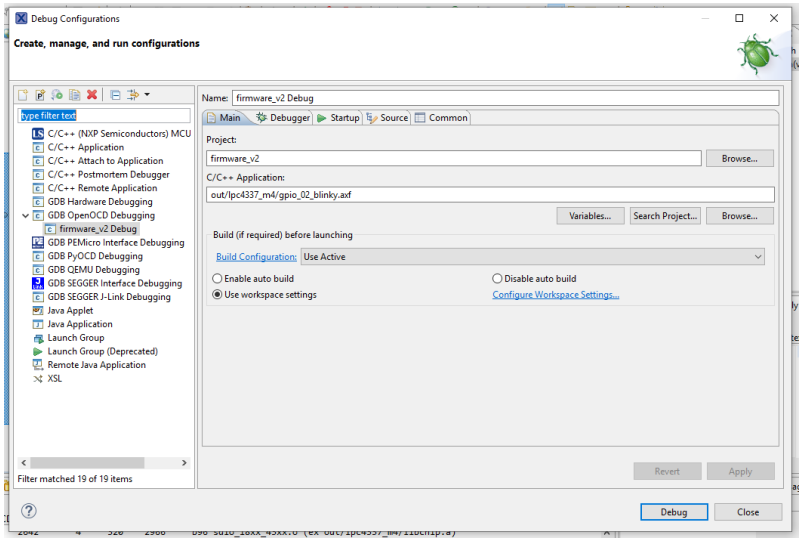

Figura 7

Se debe asegurar que la configuración del *debugger* es la correcta para la EDU-CIAA (figura 8).

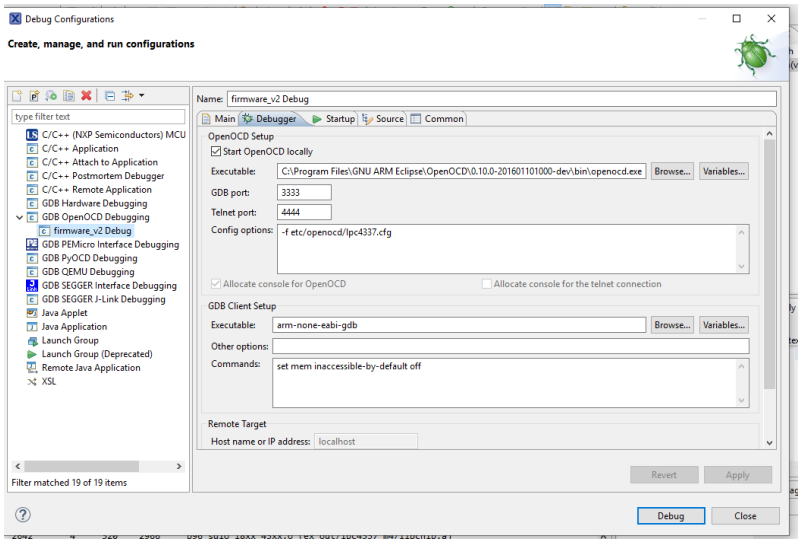

Figura 8

Una vez que está lista la configuración se puede empezar a depurar. Para ello seleccionar en el ícono del insecto la opción *Debug As: ->1 Local C/C++ Application* (figura 9). Los mensajes por consola del proceso de depuración deberían ser simiares a los de las figuras 10 y 11 y comienza así el proceso de depuración.

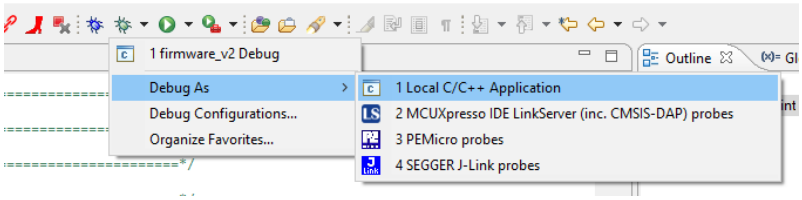

Figura 9

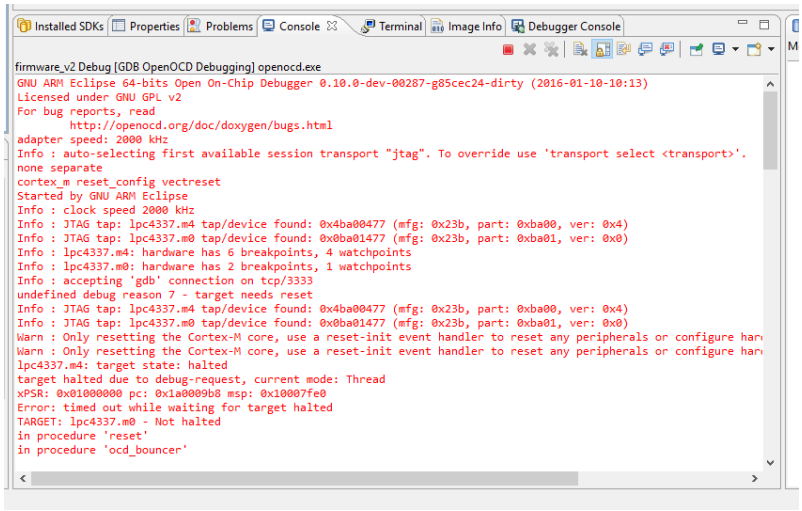

Figura 10

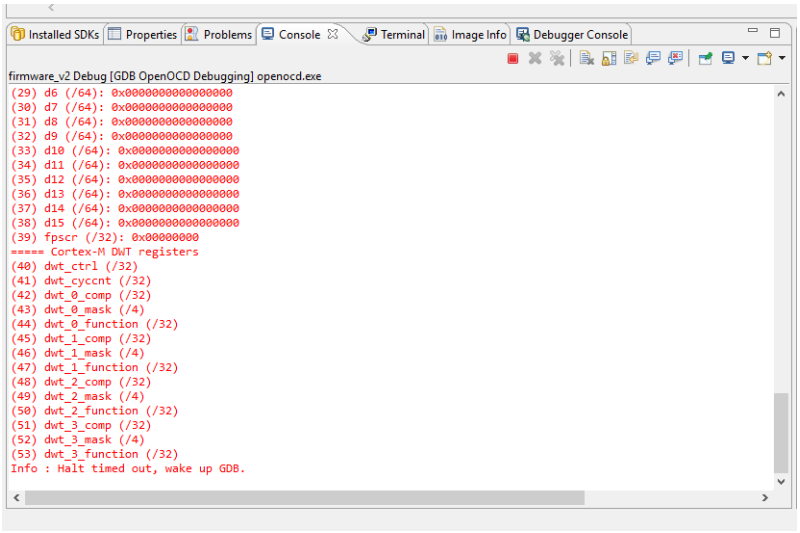

Figura 11

Opciones de *debugging*

Una vez que se subió correctamente el código se puede controlar el flujo del programa mediante las herramientas de depuración. La primera de ellas es **resume**. Esta opción hace que el programa corra desde el inicio hasta el primer *breakpoint*, el cual es un indicador del código para especificar al programa que debe detenerse allí. El primer *breakpoint* se encuentra en la función *boardConfig()* y el segundo *breakpoint* en *gpioWrite(LEDB, LOW)*. Al usar la función *resume* se pudo ver como el programa se ejecuta en la EDU-CIAA y se detiene en los *breakpoints* especificados (figuras 12 y 13).

```
51 /*==================[external data definition]=============================*/
52
53 /*==================[internal functions definition]========================*/
54
55 /*==================[external functions definition]========================*/
56
57 /* FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA LUEGO DE RESET. */
58 int main(void){
59
60    /* ------------- INICIALIZACIONES ------------- */
61
62    /* Inicializar la placa */
63    boardConfig();
64
65    /* ------------- REPETIR POR SIEMPRE ------------- */
66    while(1) {
67
68       /* Prendo el led azul */
69       gpioWrite( LEDB, ON );
70       delay(500);
71
72       /* Apago el led azul */
73       gpioWrite( LEDB, OFF );
74       delay(500);
75
76    }
77
78    /* NO DEBE LLEGAR NUNCA AQUI, debido a que a este programa no es llamado
79       por ningun S.O. */
80    return 0 ;
81 }
82
83 /*==================[end of file]==========================================*/
84
```

Figura 12

```
53 /*==================[internal functions definition
54
55 /*==================[external functions definition
56
57 /* FUNCION PRINCIPAL, PUNTO DE ENTRADA AL PROGRAMA
58 int main(void){
59
60    /* ------------ INICIALIZACIONES ------------
61
62    /* Inicializar la placa */
63    boardConfig();
64
65    /* ------------ REPETIR POR SIEMPRE ----------
66    while(1) {
67
68       /* Prendo el led azul */
69       gpioWrite( LEDB, ON );
70       delay(500);
71
72       /* Apago el led azul */
73       gpioWrite( LEDB, OFF );
74       delay(500);
75
76    }
77
```

Figura 13

La opción de *suspend* pausa el código sea cual sea la instrucción de *assembly* que está ejecutando en el momento de presionar *suspend* por lo que generalmente no se puede señalar en el código en C ya que no sería preciso. Esto se puede ver en la figura 14.

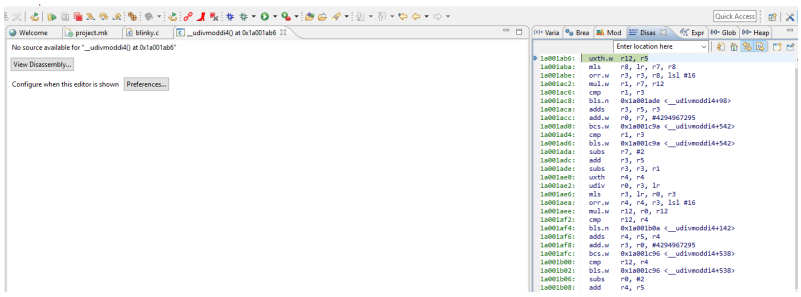

Figura 14

La opción de *restart* hace lo esperado y reinicia el código, el cual parará hasta el primer *breakpoint*. Debería verse lo de la figura 12.

Step Into

Para esta sección es conveniente tener una idea de la estructura de funciones de este proyecto:

```
1  blinky.c:
2
3  ─boardConfig()
4  ────SystemCoreClockUpdate()
5  ──────────Chip_Clock_GetRate(CLK_MX_MXCORE)
6  ──────────────(. . .mas funciones subyacentes)
7
8  ────cyclesCounterConfig()
9  ────Board_Init()
10
11
```

```
12  while(1)
13  —gpioWrite(LEDB, HIGH)
14  —delay(500)
15  (. . .continuaci n de c digo)
```

La opción de *Step Into* lleva a ejecutar y detenerse en la instrucción siguiente (del código en lenguaje C) a la instrucción actual pero teniendo en cuenta las invocaciones a funciones subyacentes. Es decir, si el cursor de función actual está en *boardConfig()* (figura 15), realizar un *Step Into* nos llevará a *SysteCoreClocKUpdate()* (figura 16). Tener en cuenta la estructura de funciones al principio de este inciso.

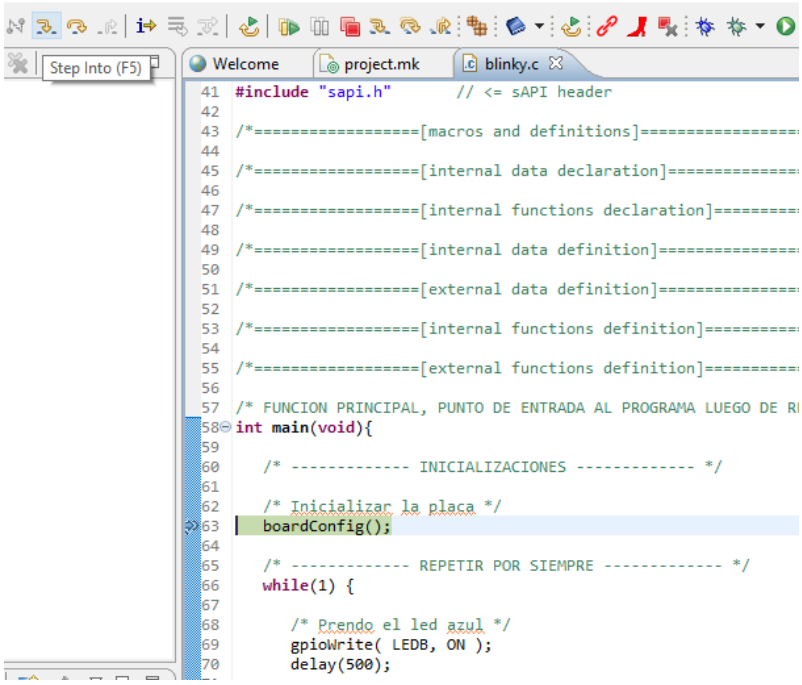

Figura 15

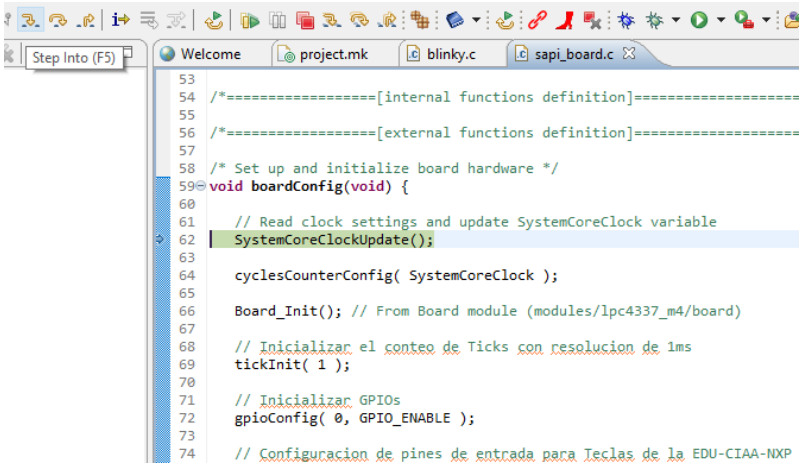

Figura 16

A su vez, otro *Step Into* nos lleva a la sigiuente función subyacente a *SystemCoreClockUpdate()* como se ve en la figura 17.

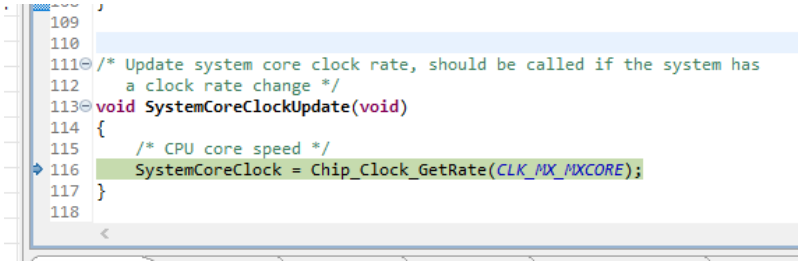

Figura 17

Se puede ver en la estructura que

```
1 —boardConfig()                % <— antes de los dos step_into estaba aqui
2 ———SystemCoreClockUpdate()            %  <— el primer step_into nos trae aqui
3 ——————Chip_Clock_GetRate(CLK_MX_MXCORE)  % <— El programa queda pausado aca
4 ————————(...)
5 ———cyclesCounterConfig()
6 ———Board_Init()
```

Step Over y *Step Return*

Step Return ejecuta todas las instrucciones subyacentes a la actual y deja el cursor en la función siguiente de la jerarquía anterior. O sea, siguiendo al ejemplo de *Step Into* que dejó el cursor en *Chip_ Clock_ GetRate(CLK_ MX_ MXCORE)*, al hacer *Step Return* no se siguió metiendo dentro de *Chip_ Clock_ GetRate()*, sino que se ejecutó todo y se dejó el cursor en la función siguiente a la que invocó *Chip_ Clock_ GetRate()*. El cursor queda entonces en *cyclesCounterConfig()* (figura 18).

En la estructura de funciones se tiene que

```
1 —boardConfig()
2 ———SystemCoreClockUpdate()
3 ——————Chip_Clock_GetRate(CLK_MX_MXCORE)  % <— estaba aca antes del step_return
4 ————————(...)
5 ———cyclesCounterConfig()  % <— El programa queda aca
6 ———Board_Init()
```

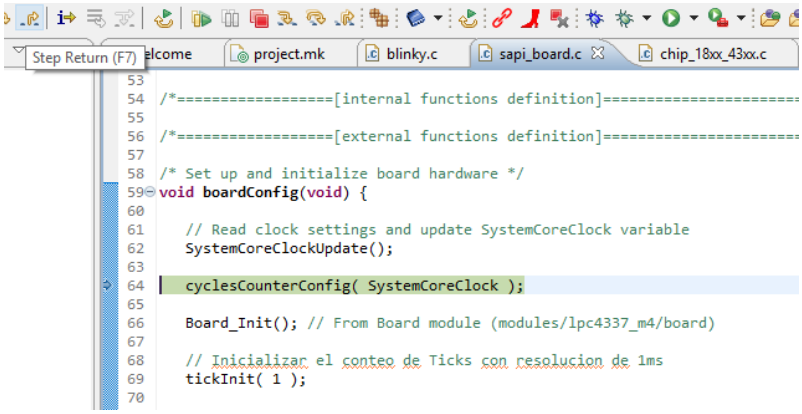

Figura 18

La función *Step Over* no se mete dentro de la función actual como *Step Into* sino que ejecuta la instrucción actual sobre la cuál está el cursor y deja este último en la siguiente función de la misma jerarquía. Siguiendo con el mismo ejemplo se pasa de *cyclesCounterConfig()* a *Board_ Init()* (figura 19).

```
1  ─boardConfig()
2  ────SystemCoreClockUpdate()
3  ──────────Chip_Clock_GetRate(CLK_MX_MXCORE)
4  ──────────────(...)
5  ────cyclesCounterConfig()   % <─ estaba aca antes del step_over
6  ────Board_Init()             % <─ El programa queda aca
```

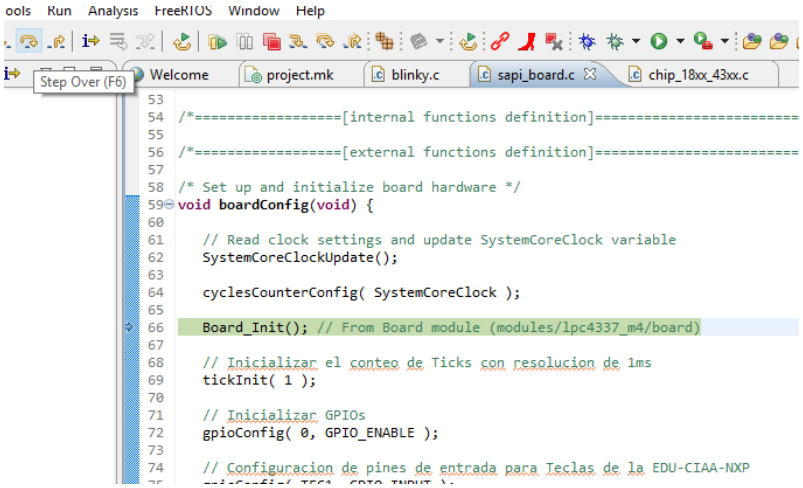

Figura 19

Terminar la depuración

Para terminar, se debe cerrar la depuración utilizando alguno de los botones de las figuras 20 y 21.

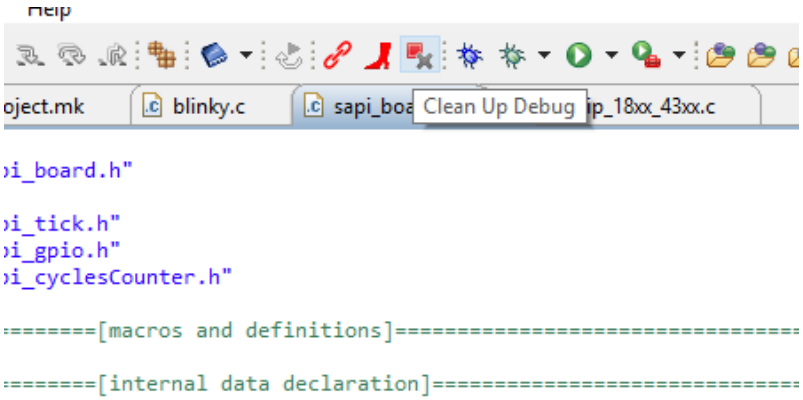

Figura 20

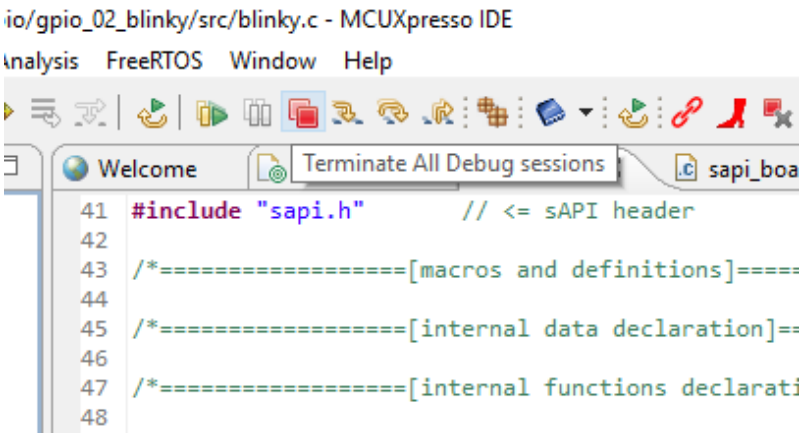
io/gpio_02_blinky/src/blinky.c - MCUXpresso IDE
Analysis FreeRTOS Window Help
Welcome
Terminate All Debug sessions
sapi_boa
41 #include "sapi.h" // <= sAPI header
42
43 /*==================[macros and definitions]=====
44
45 /*==================[internal data declaration]==
46
47 /*==================[internal functions declarati
48

Figura 21