Last updated: Thursday, 30 May 2024 at 7:09 PM

**Instructions**

For instructional video see: https://www.youtube.com/watch?v=v3C9WHLpoeY&t=425s

1. Open special ESP-IDF 5.2 PowerShell window 1 for OpenOCD the Open On-Chip Debugger
2. Create copy of blink example
   1. Create folder: "C:\ESP-IDF\esp-idf-v5.2.1\mycode"
   2. Copy from: "C:\ESP-IDF\esp-idf-v5.2.1\examples\get-started\blink"
   3. Paste into: "C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink"
   4. Edit main.c to match the code listed in Appendix A, or just copy & paste
   5. Create "C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink\gdbinit" with following code
      1. set remotetimeout 100
      2. target extended-remote :3333
      3. set remote hardware-watchpoint-limit 2
      4. mon reset halt
      5. maintenance flush register-cachethb app\_main
      6. c
3. PS C:\ESP-IDF\esp-idf-v5.2.1> cd mycode
4. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode> cd blink
5. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink> idf.py set-target esp32
6. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink> idf.py build
7. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink> idf.py -p COM3 flash
8. JTAG upload can be used as an alternative to serial via a COM port using the following:
   1. openocd -s scripts -f interface/ftdi/esp32\_devkitj\_v1.cfg -f board/esp32-wrover-kit-3.3v.cfg -c "program\_esp C:/ESP-IDF/esp-idf-v5.2.1/mycode/blink/build/blink.bin 0x10000 verify reset exit"
9. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink> openocd -f board/*esp32-wrover-kit-3.3v.cfg*
10. Config file located at: C:\Espressif\tools\openocd-esp32\v0.12.0-esp32-20230921\openocd-esp32\share\openocd\scripts\board\*esp32-wrover-kit-3.3v.cfg*
11. Open special ESP-IDF 5.2 PowerShell window 2 for GDB, the GNU Project debugger
12. PS C:\ESP-IDF\esp-idf-v5.2.1> cd mycode
13. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode> cd blink
14. PS C:\ESP-IDF\esp-idf-v5.2.1\mycode\blink> *xtensa-esp32-elf-gdb* build/blink.elf -x gdbinit
15. Executable app located at: C:\Espressif\tools\xtensa-esp-elf-gdb\12.1\_20231023\xtensa-esp-elf-gdb\bin\*xtensa-esp32-elf-gdb.exe*
    1. Thread 2 "main" hit Temporary breakpoint 1, app\_main () at C:/ESP-IDF/esp-idf-v5.2.1/mycode/blink/main/blink\_example\_main.c:31
    2. 31 configure\_led();
    3. (gdb) next
    4. Note: automatically using hardware breakpoints for read-only addresses.
    5. [esp32.cpu0] Target halted, PC=0x400D53A6, debug\_reason=00000001
    6. Set GDB target to 'esp32.cpu0'
    7. [esp32.cpu1] Target halted, PC=0x400845E2, debug\_reason=00000000
    8. 34 blink\_led();
16. The gdb command “next” or just “n” steps through the source code, there are many others, see: https://www.tutorialspoint.com/gnu\_debugger/gdb\_commands.htm

**Explanation**

1. Special PowerShell shortcuts are created when ESP-IDF the IoT Development Framework for ESP32 is installed from: <https://dl.espressif.com/dl/esp-idf> Opening a special PowerShell Window from the shortcut sets up all the Windows paths for correct operation of idf.py command-line tool, in this instance for OpenOCD the Open On-Chip Debugger
2. Make a copy of the BLINK example supplied with the ESP-IDF and add to a new folder mycode. Edit the source code to make simpler and create an initialisation file for GDB, the GNU Project debugger. Note the port used for listening by OpenOCD is port 3333.
3. Change directory to mycode
4. Change directory to blink
5. Run the command to set the target hardware to ESP32
6. Run the command to build the source code to an ESP32 binary image
7. Flash, or upload, the built image to the target hardware, COM port could be different
8. Two upload methods are possible using an ESP-PROG JTAG debugger board. One is to use the virtual COM port e.g. COM3, which I believe uses Espressif code running on the FreeRTOS integrated in all ESP MCUs; the command for this is shown in point 7. The other method is to use the JTAG part of the ESP-PROG, which I believe uses dedicated JTAG boundary scan hardware on the ESP32 to read and write binary data, from and to the onboard flash memory. JTAG boundary scan hardware is essentially a large complex arrangement of shift registers that can move data between parallel connected internal devices and external serial connected debugger boards. OpenOCD commands for JTAG uploads are fiendishly complex and difficult to find and assemble, so here is my attempt at an explanation:
   1. When using file search tools, it is important to note that the delimiter for Windows paths is the back-slash \ and for UNIX and Linux it is a forward-slash /, using the wrong one can cause a search to fail!
   2. ***-s scripts*** is used to specify the search path for scripts located at C:\Espressif\tools\openocd-esp32\v0.12.0-esp32-20230921\openocd-esp32\share\openocd\scripts
   3. ***-f interface/ftdi/esp32\_devkitj\_v1.cfg*** specifies the configuration file for the debug interface, located at C:\Espressif\tools\openocd-esp32\v0.12.0-esp32-20230921\openocd-esp32\share\openocd\scripts\interface\ftdi
   4. ***-f board/esp32-wrover-kit-3.3v.cfg*** specifies the configuration file for the target device, located at C:\Espressif\tools\openocd-esp32\v0.12.0-esp32-20230921\openocd-esp32\share\openocd\scripts\board
   5. ***-c "program\_esp C:/ESP-IDF/esp-idf-v5.2.1/mycode/blink/build/blink.bin 0x10000 verify reset exit"*** passes commands directly to the OpenOCD command interpreter, in this example the function program\_esp (should be program\_esp32 but due to a bug in OpenOCD program\_esp32 does not work). The function program\_esp takes the file blink.bin and uploads it to the target ESP32 hardware starting at hex address 0x10000 i.e. where binary images should be placed. The verify and exit commands do what their name suggests and the reset command resets the ESP32 target hardware, if one omits this command, one has the press the reset button on the target hardware to start execution.
      1. Note that the image is uploaded to 0x10000, known as the *factory or app* location, which is a pseudo location that is mapped by the configurable *partition table* to the physical address 0x3F40 0000 in the ESP32’s memory space, see Appendix B. To see the partition mapping run *idf.py partition-table* from the IDF PowerShell window and you will see something like this:
         * \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
         * # ESP-IDF Partition Table
         * # Name Type SubType Offset Size
         * nvs data nvs 0x9000 24K
         * phy\_init data phy 0xf000 4K
         * factory app factory 0x10000 1M
         * \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
      2. The default pseudo location for the partition table itself is: 0x8000.
      3. Unless you need to know, best not to ask!
9. Start OpenOCD using the hardware configuration file for the esp32-wrover
10. This is where the configuration file is located along with other targets
11. Open a second instance of the special ESP-IDF PowerShell window
12. Change directory to mycode
13. Change directory to blink
14. Start the GDB debugger
15. Location of gdb executables for various target hardware and example output from GDB
16. Other GDB commands are listed at: https://www.tutorialspoint.com/gnu\_debugger/gdb\_commands.htm

**Appendix A - Code in main.c**

/\* Blink Example

\*/

#include <stdio.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "sdkconfig.h"

int delay\_mS = 1000;

#define BLINK\_GPIO 2 // CONFIG\_BLINK\_GPIO

static uint8\_t s\_led\_state = 0;

static void blink\_led(void)

{

/\* Set the GPIO level according to the state (LOW or HIGH)\*/

gpio\_set\_level(BLINK\_GPIO, s\_led\_state);

}

static void configure\_led(void)

{

gpio\_reset\_pin(BLINK\_GPIO);

/\* Set the GPIO as a push/pull output \*/

gpio\_set\_direction(BLINK\_GPIO, GPIO\_MODE\_OUTPUT);

}

void app\_main(void)

{

configure\_led();

while (1) {

blink\_led();

/\* Toggle the LED state \*/

s\_led\_state = !s\_led\_state;

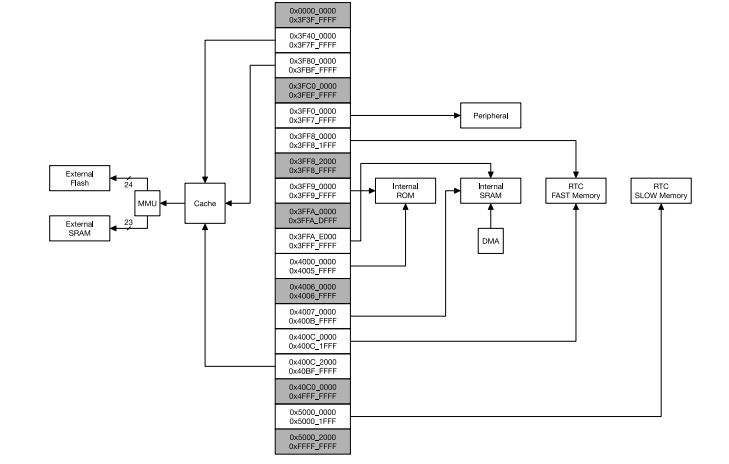
vTaskDelay(delay\_mS / portTICK\_PERIOD\_MS); //Delay 200mS

}

}

**Appendix B – ESP32 Memory Mapping**

Look at the memory map below and you can see that physical address 0x3F40 0000 passes through the cache then the MMU (Memory Management Unit) and then to the external flash chip. It is this starting address that is mapped to pseudo address 0x10000 by the MMU.



**Useful web resources**

1. <https://openocd.org/>
2. <https://openocd.org/doc/pdf/openocd.pdf>
3. <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32/versions.html>
4. <https://docs.espressif.com/projects/esp-idf/en/v5.0/esp32/api-guides/tools/idf-py.html>
5. <https://www.tutorialspoint.com/gnu_debugger/gdb_commands.htm>
6. <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>