EECS 395/495: Engineering System Design 1

Homework 4

Due: Feb 9, 2017

Please complete this assignment with your group. To get credit for your work, please upload your zipped project folder to Canvas, and come to either Ilya's or Sid's office hours and show them your programmed circuit. The project folder should contain the .atsln file, and a folder containing all the source and debug files.

For this assignment, you will program your MCU and configure the WiFi module. The end goal is to have your system take a photo and store it as a file on the AMW004.
Note: you do not have to implement every one of these functions by hand. Many of them will be directly copy-and-pasted from sample projects. Others will require only slight modifications. Only a few have to written from start to finish.

1. PROGRAMMING THE MCU
   The files you should modify (or create, shown in **bold**) are:
   - main.c
   - **wifi.c**
   - **wifi.h**
   - **camera.c**
   - **camera.h**
   - conf_board.h
   - conf_clock.h
   - init.c

   Below are the descriptions of what you should implement.
   - **wifi.h**: WiFi pin definitions, WiFi UART parameters, WiFi function and variable declarations.
   - **wifi.c**
     - WiFi variable initializations.
     - **void WIFI_USART_HANDLER(void)**: Handler for incoming data from the WiFi. Should call **processIncomingByte_wifi** when a new byte arrives.
     - **void processIncomingByte_wifi(uint8_t inByte)**: Stores every incoming byte (**inByte**) from the AMW004 in a buffer.
     - **void wifi_command_response_handler(uint32_t ul_id, uint32_t ul_mask)**: Handler for "command complete" rising-edge interrupt from AMW004. When this is triggered, it is time to process the response of the AMW004.
     - **void process_data_wifi (void)**: Processes the response of the AMW004, which should be stored in the buffer filled by **processIncomingByte_wifi**.
     - **void wifi_web_setup_handler(uint32_t ul_id, uint32_t ul_mask)**: Handler for button to initiate web setup of AMW004. Should set a flag indicating a request to initiate web setup.
     - **void configure_usart_wifi(void)**: Configuration of USART port used to communicate with the AMW004.
     - **void configure_wifi_comm_pin(void)**: Configuration of "command complete" rising-edge interrupt.
     - **void configure_wifi_web_setup_pin(void)**: Configuration of button to initiate web setup.
     - **void write_wifi_command(char* comm, uint8_t cnt)**: Writes a command (**comm**) to the AMW004, and waits either for an acknowledgment or a timeout. The timeout can be created by setting the global variable **counts** to zero, which will automatically increment every second, and waiting while **counts** $<$ **cnt**.

  - **void write_image_to_file(void)**: Writes an image from the SAM4S8B to the AMW004. If the length of the image is zero (i.e. the image is not valid), return. Otherwise, delete the previous stored image. Then, create the new image.
- **camera.h**: Camera pin definitions, Camera TWI parameters, Camera function and variable declarations.
- **camera.c**
  - Camera variable initializations.
  - **void vsync_handler(uint32_t ul_id, uint32_t ul_mask)**: Handler for rising-edge of VSYNC signal. Should set a flag indicating a rising edge of VSYNC.
  - **void init_vsync_interrupts(void)**: Configuration of VSYNC interrupt.
  - **void configure_twi(void)**: Configuration of TWI (two wire interface).
  - **void pio_capture_init(Pio *p_pio, uint32_t ul_id)**: Configuration and initialization of parallel capture.
  - **uint8_t pio_capture_to_buffer(Pio *p_pio, uint8_t *uc_buf, uint32_t ul_size)**: Uses parallel capture and PDC to store image in buffer.
  - **void init_camera(void)**: Configuration of camera pins, camera clock (XCLK), and TWI port.
  - **void configure_camera(void)**: Configuration of OV2640 registers for desired operation.
  - **uint8_t start_capture(void)**: Captures an image after a rising edge of VSYNC, gets image length. Returns 1 on success (i.e. a nonzero image length), 0 on error.
  - **uint8_t find_image_len(void)**: Finds image length based on JPEG protocol. Returns 1 on success (i.e. able to find "end of image" marker), 0 on error.
- **conf_board.h**: Pin definitions for general board.
- **conf_clock.h**: Clock definitions for general board.
- **init.c**: Pin initializations.
- **main.c**: General operation. First, run all initializations. Then, loop through a set of commands. More specifically:
  - Initialization
    * Initialize clock and board definitions.
    * Configure the WiFi USART, as well as the Command pin and Web Setup pin.
    * Reset the WiFi and wait for it to connect to a network. While waiting, make sure to listen for the Web Setup pin.
    * Initialize and configure the camera.
    * Configure and start the Timer.
    * Tell the WiFi to turn off the command prompt and command echo.
  - Loop
    * Check for Web Setup request.
    * If network is available, query available websocket connections.
    * If no connections available, delay 1s and start over.
    * If connections available, take picture.
    * If picture taken successfully, transfer it to a WiFi file.
    * Check which streams are available, and send a message to each one to update their image.

In your file structure, make sure to also include the files ov2640.c, ov2640.h, ov2640_table_registers.c, timer_interface.c, and timer_interface.h.

Notes:

- Don't forget to include <asf.h> in every .h file.
- Don't forget to include .h files in .c files.

- Don't forget to declare variables as "volatile" if they are changed in interrupt service routines.
- It may be helpful to turn off compiler optimization for debugging.

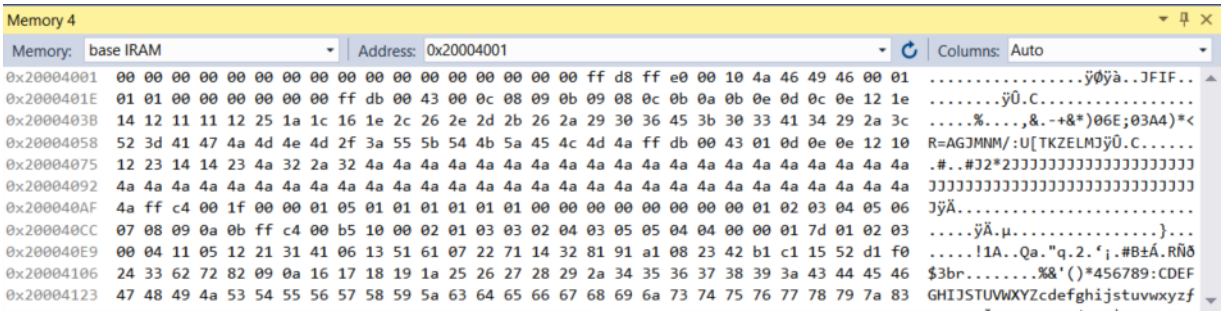
2. AMW004 CONFIGURATION

   We also have to configure our WiFi module to properly interact with the MCU. These commands can be issued using a terminal emulator, or as code at the beginning of your MCU program. If you want to interact with the WiFi module using the terminal, make sure you have not yet configured the USART module on the MCU. If you have already written your program and are running it, you can unplug the UART wires from the WiFi module. Alternatively, you can use the debugger to pause your program before it initializes the USART pins.
   - Turn on flow control on UART 1.
   - Set the size of the UART RX buffer to 5000.
   - Set GPIO 20 as the WLAN indicator.
   - Set GPIO 21 as the NETWORK indicator.
   - Set GPIO 22 as the SOFTAP indicator.
   - Set one of GPIO 1-4 as the "command complete" signal. Make sure to configure the corresponding pin on the MCU appropriately.
   - Set one of GPIO 1-4 as the "network status" signal, which is used to indicate when the module is connected to the internet. Make sure to configure the corresponding pin on the MCU appropriately.

   After performing all of these configurations, make sure to save and reboot the module to make the changes effective and permanent.

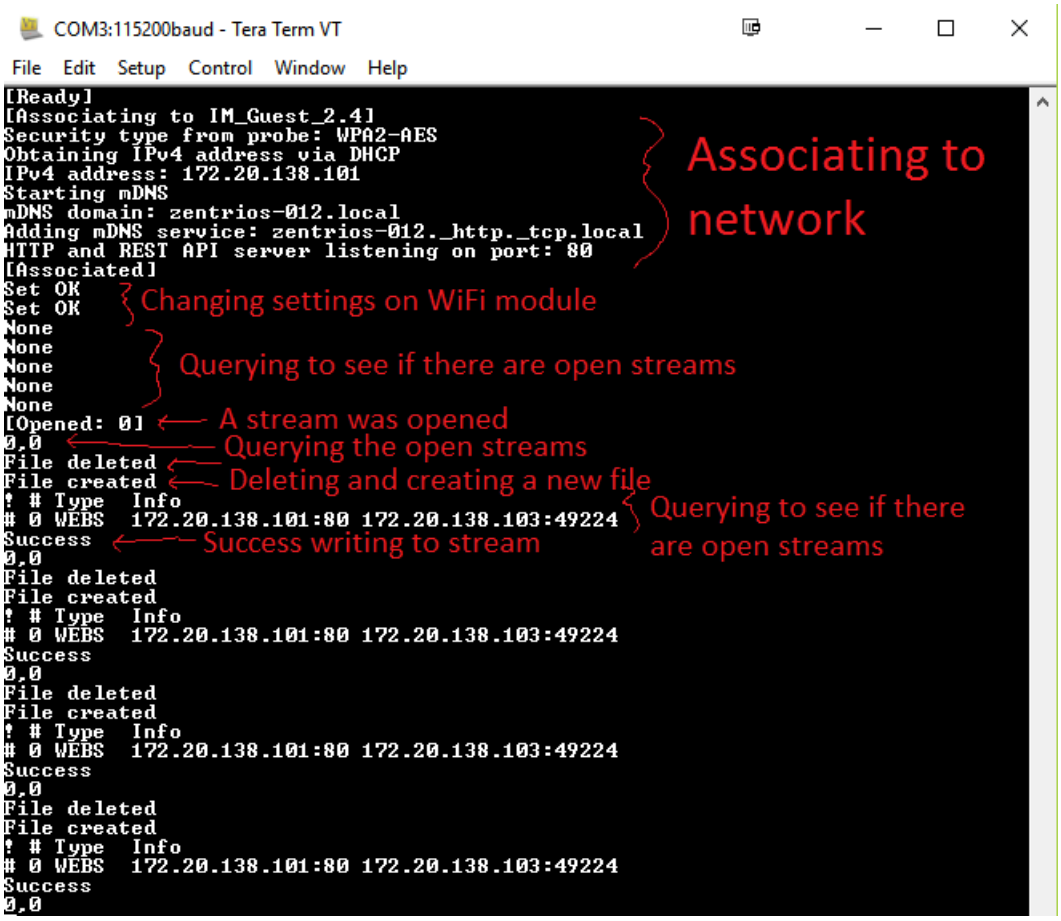
While we have not yet made a website to display our image, we can still check to see that everything is working. The easiest check is if a valid image is stored in the WiFi module. You can check this using its online file management system. Just click on the image you stored, and it should open in a new window. If that does not work, we have to break it down into steps.

Assuming your circuit passed the tests of Homework 3, most of your connections should be OK (with the possible exception of the connections that were not fully specified and therefore not tested, such as "free GPIOs"). Try placing a breakpoint right after your image is captured. When execution stops there, open the "memory" interface in the debugger and go to "Base IRAM" (internal RAM). If you scroll through it a little, you should be able to find your image. Based on the JPEG protocol, you should see a structure similar to the one below, though probably at a different address than mine (depending on your program).



If you see this, then the image was probably captured successfully. Make sure your **find_image_len** function finds a non-zero image length.

To debug your WiFi interface, keep your terminal application open to be able to see the interactions of the MCU with the WiFi module. The WiFi module should respond with a "Set OK" response after each setting you change in the initialization. When you are transferring the image, you should also see appropriate responses. A sample output is shown below.



This output is actually a bit more than what you would expect, since we have not yet written the website code. In the final application, you will only delete and write a new file if there is a streaming connection. However, for debugging, you should bypass this check, as there will not be a connection until we make a website later.