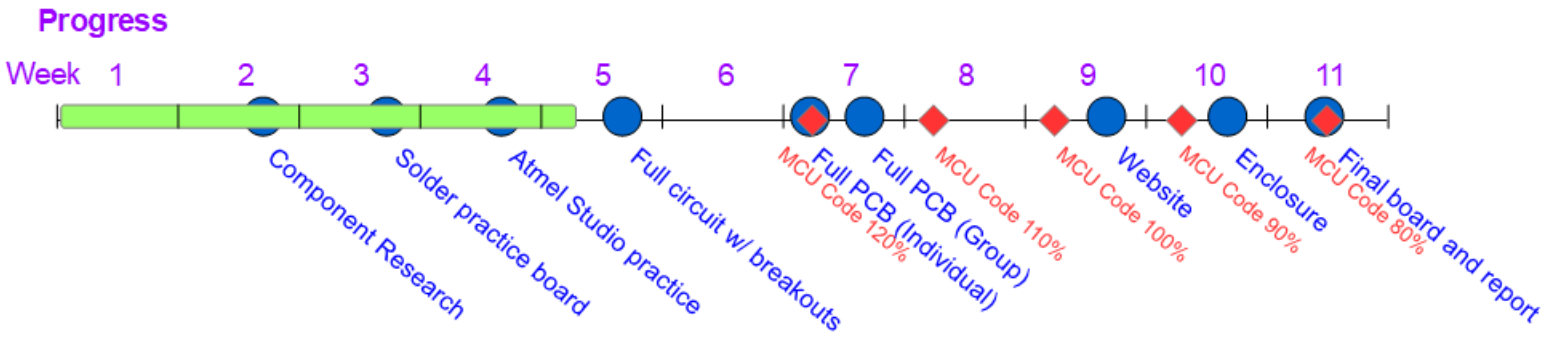
EE 326: Electronic System Design 1

Firmware Design

Due: Feb 18, 2020 (120%)
Feb 25, 2020 (110%)
Mar 03, 2020 (100%)
Mar 10, 2020 (90%)
Mar 18, 2020 (80%)



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 Roberto'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 send it to the AMW136.
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. I recommend using the following sample projects: common ioport service example, ov7740 imagesensor capture example, and usart hardware handshaking example.

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 **process_incoming_byte_wifi** when a new byte arrives.
  - **void process_incoming_byte_wifi(uint8_t in_byte)**: Stores every incoming byte (**in_byte**) from the AMW136 in a buffer.

  - **void wifi_command_response_handler(uint32_t ul_id, uint32_t ul_mask)**: Handler for "command complete" rising-edge interrupt from AMW136. When this is triggered, it is time to process the response of the AMW136.
  - **void process_data_wifi (void)**: Processes the response of the AMW136, which should be stored in the buffer filled by **process_incoming_byte_wifi**. This processing should be looking for certain responses that the AMW136 should give, such as "start transfer" when it is ready to receive the image.
  - **void wifi_web_setup_handler(uint32_t ul_id, uint32_t ul_mask)**: Handler for button to initiate web setup of AMW136. 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 AMW136.
  - **void configure_wifi_comm_pin(void)**: Configuration of "command complete" rising-edge interrupt.
  - **void configure_wifi_web_setup_pin(void)**: Configuration of button interrupt to initiate web setup.
  - **void write_wifi_command(char* comm, uint8_t cnt)**: Writes a command (**comm**) to the AMW136, 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 AMW136. If the length of the image is zero (i.e. the image is not valid), return. Otherwise, follow this protocol (illustrated in Appendix B):
    1. Issue the command "image_transfer xxxx", where xxxx is replaced by the length of the image you want to transfer.
    2. After the AMW136 acknowledges that it received your command, start streaming the image.
    3. After the image is done sending, the AMW136 should say "Complete". However, the "command complete" pin will not have a rising edge, so it will be hard to sense. You can still try to sense it before moving on, or simply insert a slight delay.
- **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 calling the **configure_twi** function.
  - **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, and 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" and "start of image" markers), 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. A flow chart of the whole program is shown in Appendix A. First, run all initializations. Then, loop through a set of commands. More specifically:
  - Initialization
    * Initialize clock and board definitions.
    * Configure and start the Timer. (Look in the "timer_interface" functions.)
    * 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.
    * 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 it is not, reset the module.
    * If no connections available, delay 1s and start over.
    * If connections available, take picture.
    * If picture taken successfully, transfer it to the AMW136.

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.
- Don't forget that the RTS pin of the MCU does not function properly. Therefore, configure it as an output and drive it low (to always allow communication from the WiFi module to the MCU).

AMW136 CONFIGURATION

First, we must load a custom firmware for the AMW136 that will allow it to act as a webcam. (*Note: you can actually create a webcam without this firmware. For every new photo, you can save it to a file on the AMW136. Then, you can tell the webpage that a new photo is available, and the image will be refreshed. This is not the optimal way, because writing to the Flash memory of the WiFi chip is not efficient. Instead, our custom firmware allows us to write the photo to the WiFi chip's RAM, and to read it from there on our website. There are no native API commands to allow for this, which is why we have to use custom firmware.*) All commands can be issued using a terminal emulator. 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.

1. Connect to the internet:
   (a) If you are at Northwestern, you must connect to the "Device-Northwestern" network. However, you must first register your device with Northwestern.
      i. In the terminal, type "get wl m". This will give you your WiFi chip's MAC address.
      ii. Go to https://device.wireless.northwestern.edu, and log in with your NetID and password.
      iii. Enter your WiFi chip's MAC address, and give your device a name. Enable device sharing, with "Others and Me" under Sharing Options. Finally, click "Register Device".
   (b) In the terminal, type "setup web". The WiFi chip will now become a hotspot. It will tell you its name in the terminal.

(c) On your phone or laptop, connect to the AMW136. If it asks for a password, use "password".
(d) Open a browser, and go to "setup.com".
(e) In the interface that opens, select your WiFi network, or "Device-Northwestern" if you are at Northwestern.
(f) Make sure to select the "reconnect to device" option.

2. Load the custom firmware:
   (a) Register for Zentri DMS (https://dms.zentri.com/signup).
      - Make sure to use your Northwestern email to register.
      - Make sure to enter "Northwestern University Student" under Company Name.
   (b) Claim your device and update:
      i. In the terminal, type "dms claim <dms_username> <dms_password>".
      ii. Type "ota".
   (c) Type "dms activate TTIzQkhHbFRlUktzcVljcEN2RHZVbXll".
   (d) Type "ota".

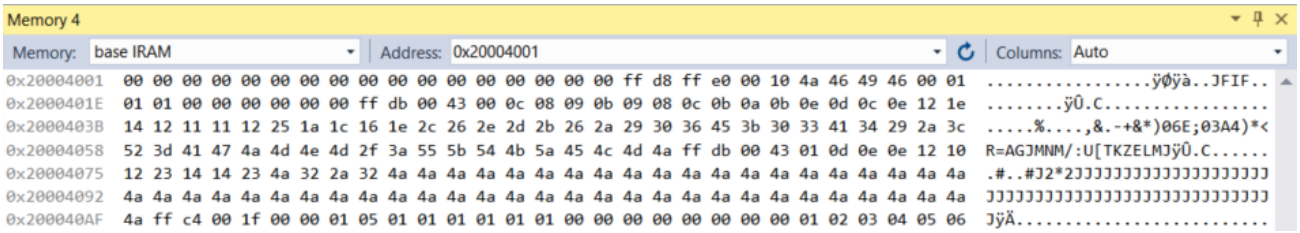
We also have to configure our WiFi module to properly interact with the MCU. These commands can be issued using a terminal emulator, or as code at the beginning of your MCU program. The commands and descriptions can be found at https://docs.zentri.com/zentrios/wz/latest/.

- Turn on flow control on UART 0.
- Set the size of the UART RX buffer to 5000.
- Set GPIO 20 as the WLAN indicator.
- Set GPIO 18 as the NETWORK indicator.
- Set GPIO 21 as the SOFTAP indicator.
- Set one of GPIO 13-16 as the "command complete" signal. Make sure to configure the corresponding pin on the MCU appropriately.
- Set one of GPIO 13-16 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 in your MCU code, make sure to save and reboot the module to make the changes effective and permanent (also in code).

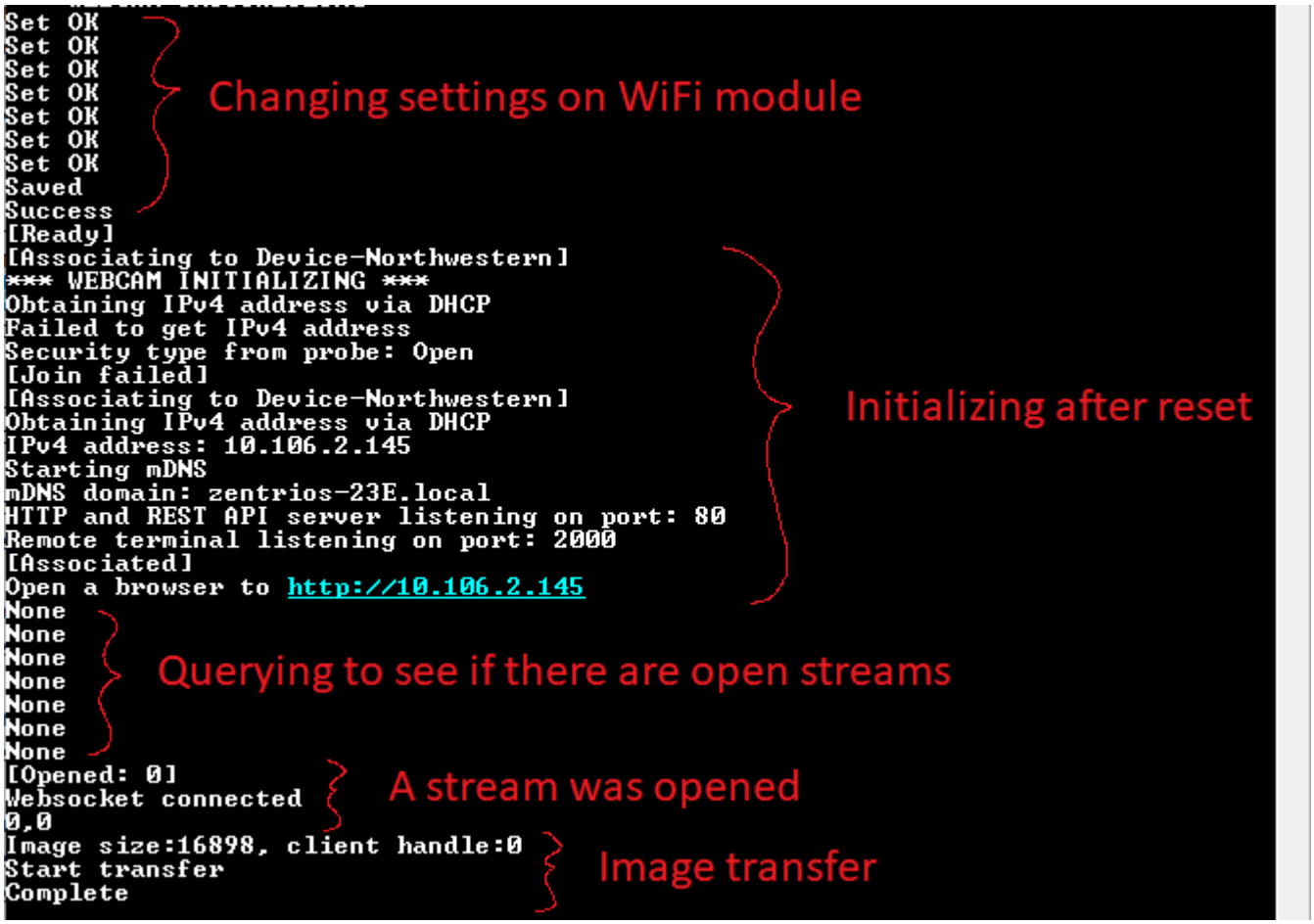
While we have not yet made a website to display our image, we can still check to see that everything is working. The easiest check is to open a browser to the IP address of your AMW136, and, using the file system, navigate to "webcam/webcam.html". It should immediately start streaming images. If that 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).

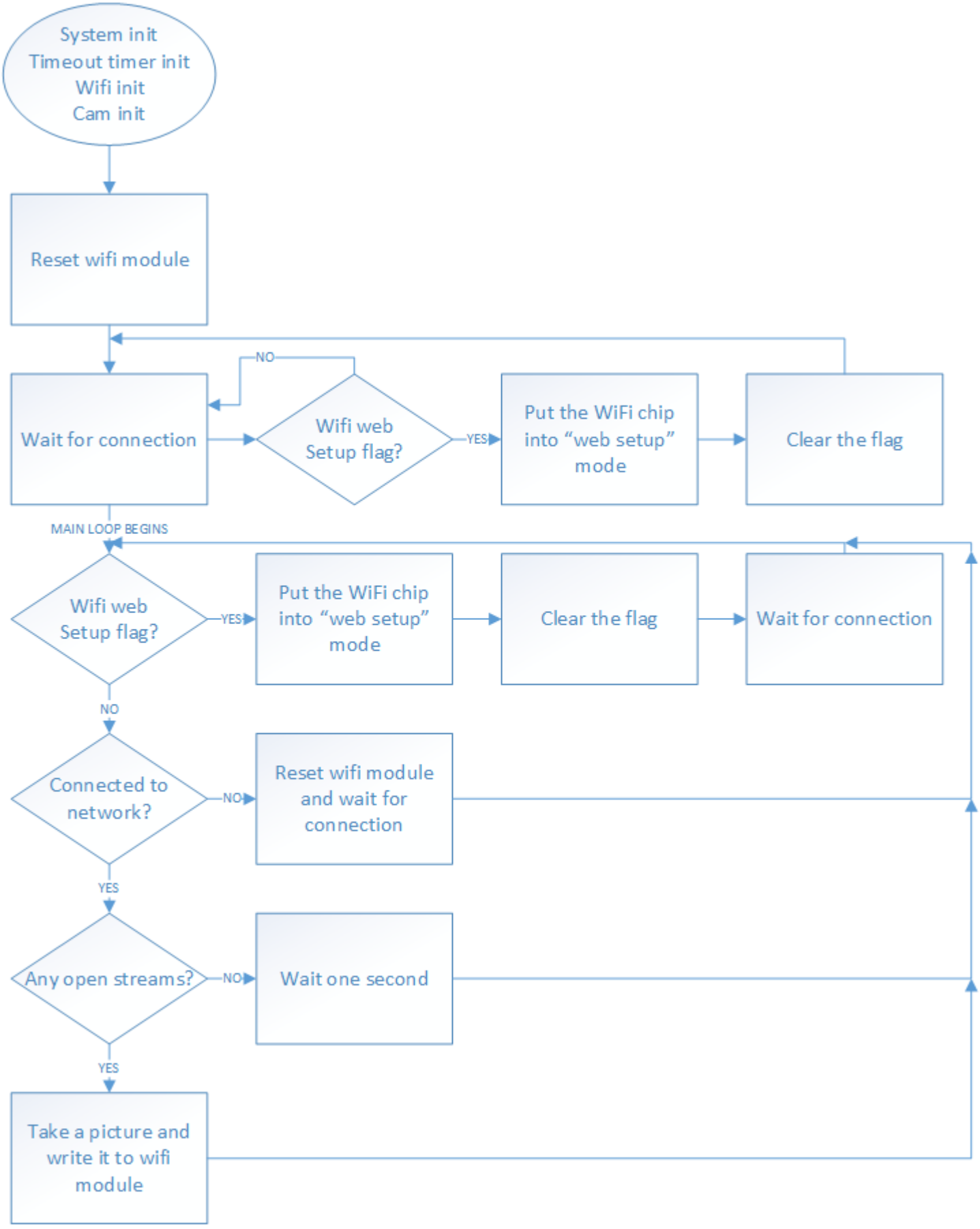


If you see this, then the image was probably captured successfully. Make sure your **find_image_len** function finds a non-zero image length.

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. If everything is working as expected, the AMW136 will prompt an image transfer with "Start transfer", and once the image is successfully transmitted, it will respond with "Complete". A more detailed view of this, using the Saleae logic analyzer, can be seen in Appendix B.



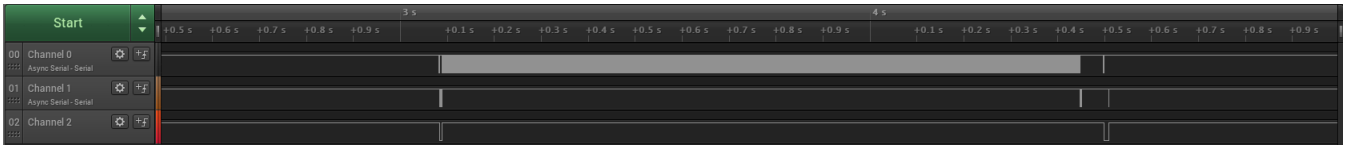
Appendix A: Program Flowchart



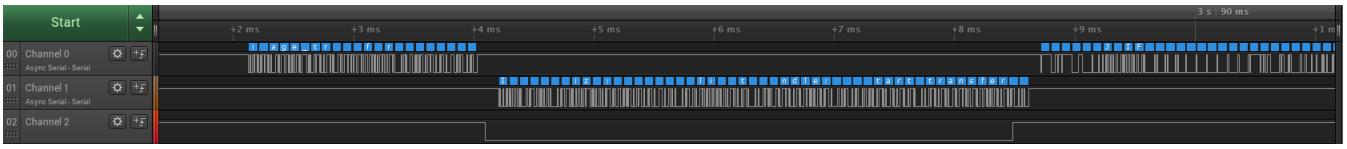
Appendix B: Communication with AMW136

The figures below show the AMW136's RX, TX, and "command complete" pins.

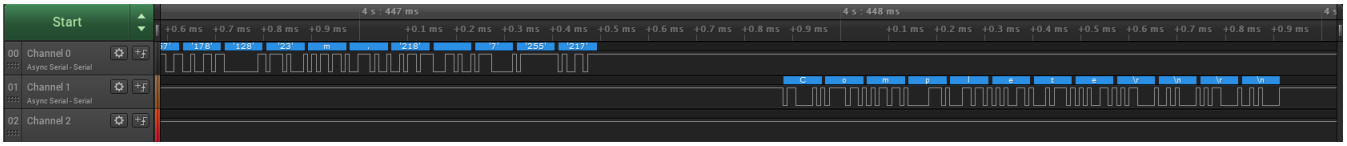
This figure shows the complete communication of one image. First, there is a "start transfer" command on the RX line, followed by a response on the TX line, as well as a rising edge on the "command complete" line after the AMW136 is done responding. Then, the image is transferred. Afterwards, the AMW136 acknowledges the image transfer. Finally, the MCU queries if there is still client connected, and starts taking a new photo (not depicted here).



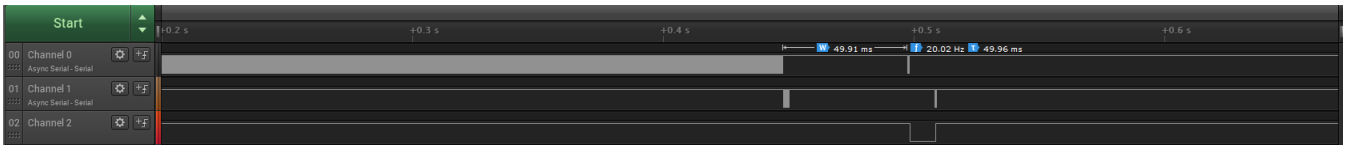
This figure shows a closeup of the communication that starts the image transfer.



This figure shows a closeup of the end of the image transfer. The important thing to note is that the "command complete" line does not have a rising edge after the acknowledgment.



This figure shows how I handled the lack of rising edge of the "command complete" line. I simply used a 50ms delay, which is plenty of time after the end of the image transfer.



FAQ

Note About Image Buffer

For your image buffer, use a preallocated array instead of a pointer (i.e. uint8_t im_buf[array_size], not uint8_t * im_buf).

Q. When it says "Reset the Wifi" in the main loop, is that a hard reset on the pin or just the command "reboot"?

A. That is a hard reset, since you have the WiFi pin connected to an MCU GPIO. I recommend pulling the pin low, waiting 100 ms, and then setting it high.

Q. Do we have to consider all the possible error code responses [for process_data_wifi()] or just Success/Command failed/Unknown command?

A. You do not have to consider every possible error code. Just the necessary subset that tells you if your commands succeeded or not. For example, "None" is a useful response to the "stream_poll" command, indicating whether people are connected to your webcam.

Q. What is meant by "configure camera pins" and "configure twi pins". Is this different from the rest of the initialization functions we are calling? If we are supposed to use the gpio_configure_pin function, what flags should we use?

A. Configuring the pins means that you make the pins act as their appropriate peripheral instead of a general-purpose pin. For the flags, please refer to the OV7740 sample project. They have all of the proper flags.

Q. Atmel indicates that the connected device is not a SAM4S8B, but it is.

A. This means that the soldering is not complete somewhere, or that you may have connected a high voltage to the MCU's 1.2V input. I saw this problem before when people accidentally shorted together the 3.0V and 1.2V input on the MCU.