TCP Communication Implementation via zephyr OS by Quectel Modem

Contents

1. Introduction to Project

At first there was a C code based on FreeRTOS on Modem board, which could just send data through a TCP connection and couldn't receive any data. Modem was used to send mitter's data, which measures voltage, current and some parameters of power system components. This code wasn't complete and had some bugs. There was an OS called Zephyr, which was object-oriented based and taken from Linux OS. Embedded programming with Zephyr was much easier than FreeRTOS, so we decided to reprogram the Modem board with Zephyr OS to have much easier and simpler embedded code which can send and receive data through a TCP connection.

How Modem Works

In 3 days, I was reading the FreeRTOS code which was on ARM MCU to understand how a Modem works. Modem has 3 layers:

1. **Application Layer:** in this layer commands like send and receive are sent and any application can use these commands to send and receive data.
2. **Middle Layer:** this is the main layer which converts send and receive commands from application layer to specific AT commands. This layer includes sending number, string, character, connecting to TCP server and etc.
3. **Hardware Layer:** here packets are physically received and sent as a stream of bytes

Receiving data

When a packet is received in hardware layer, hardware layer calls back middle layer and parses received data to middle layer. Middle layer reads received data and according to what is received it calls related function.

Sending data

At first application layer calls send function, then middle layer translate send to related AT commands, at the end hardware layer sends binary data from DMA through physical medium.

ARM Boot and Modem Startup

At first ARM MCU boots and clock and reset is configured then kernel initials then sections are defined via linker at the end functions are defined through API function and then an infinite while loop starts, which Modem's main code will run in this while loop.

Side Items Learned

Beside the Modem, I learned some OS concepts such as masking interruption, constructors, tick less operation systems, semaphores, linker, rcc and etc. I learned some network concepts such as MQTT, TCP and PTP protocols, circular networks and linear network, stunnel, PLL and etc. I also learned about structure of stm32 ARM microcontroller like its MMU, sections like DSP and arithmetic.

2. Zephyr OS installation and startup

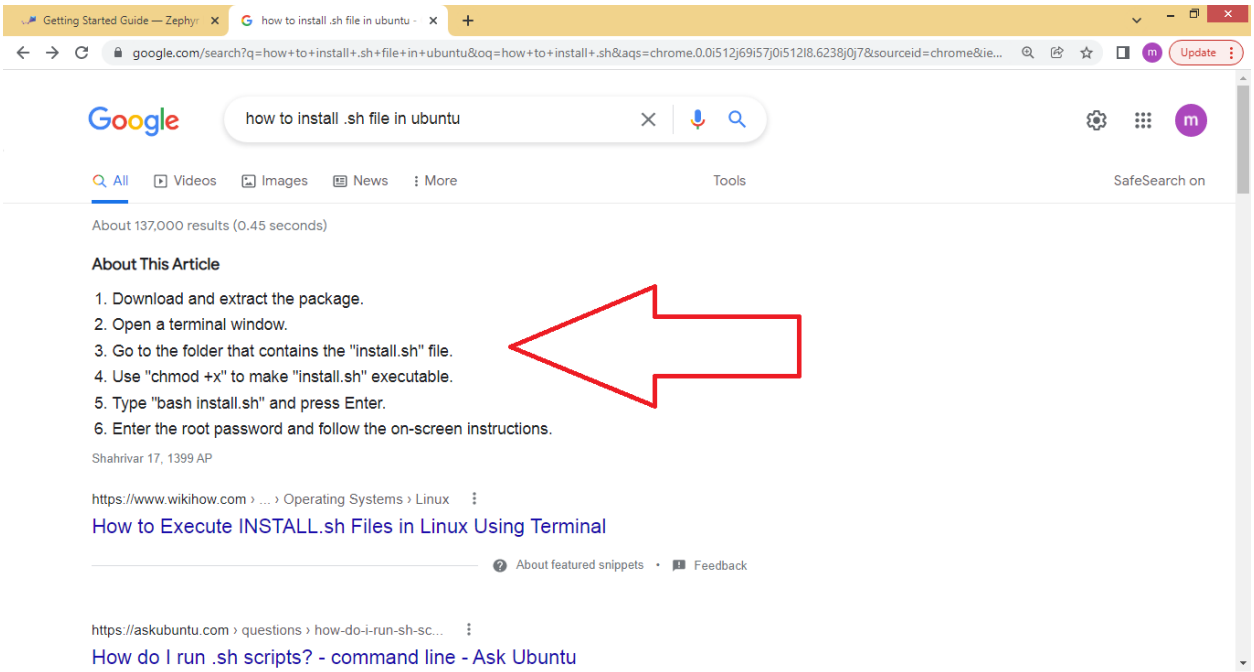
Installation

At first we installed Zephyr OS from below link:

"https://docs.zephyrproject.org/latest/develop/getting_started/index.html"

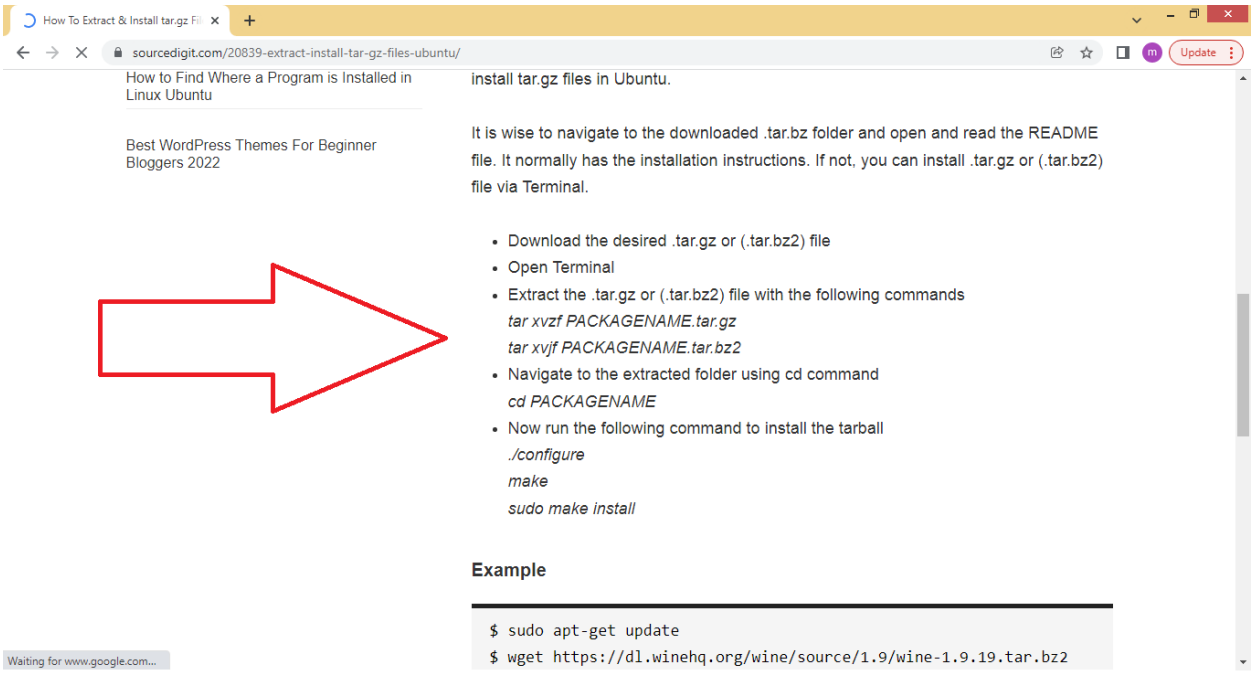
For installation Zephyr OS on Linux just follow the steps told in above link.

For installation .sh files follow steps in below image:



Or you can right click on .sh file, go to permissions tab, check "allow run as a program" and then simply open terminal in .sh file directory and then type "./program's name.sh" command. (sh stands for shell which is for installation)

For installation .tar files follow steps in below image:



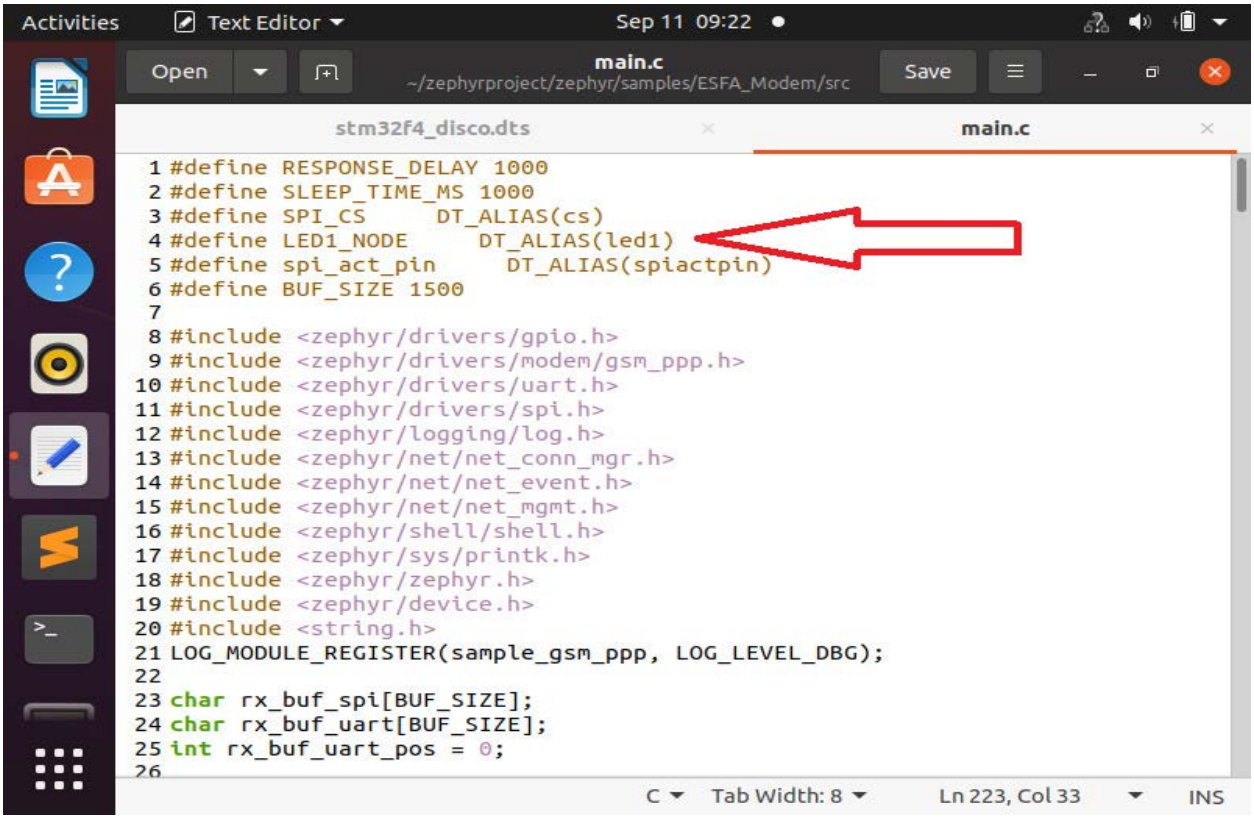
(tar stands for tarball which is for running multiple files)

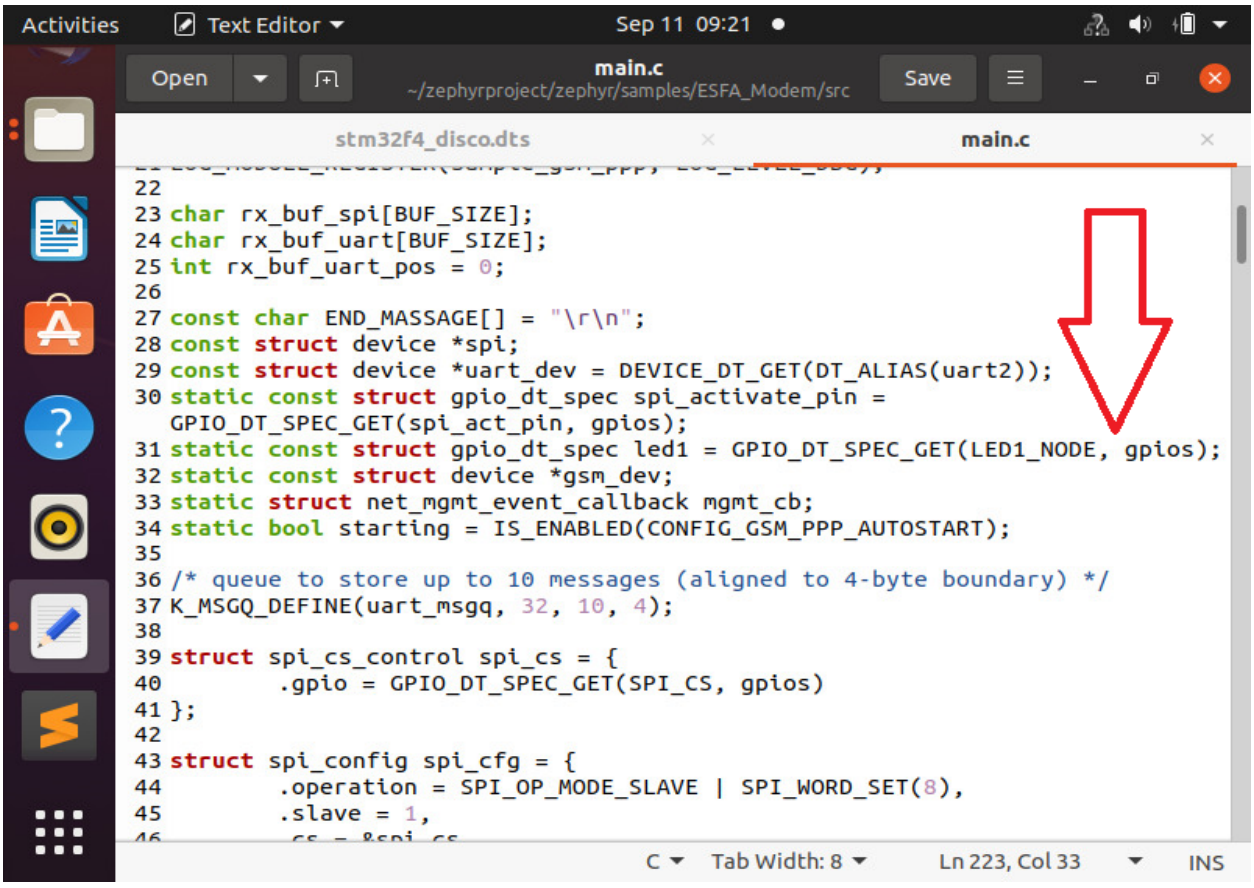
Startup

1. Making a device tree: Device tree is a text file which you define board's different units in there such as uarts, spi, can, leds, flash, sram and etc. To make a new device tree for your board, go to zephyrproject/zephyr/boards/arm, find the name of your board and make a copy of it. At the end rename the copied file. In our case, we found stm32f4_disco (disco stands for discovery board) and made a copy of it and renamed it to stm32f407vgt6.

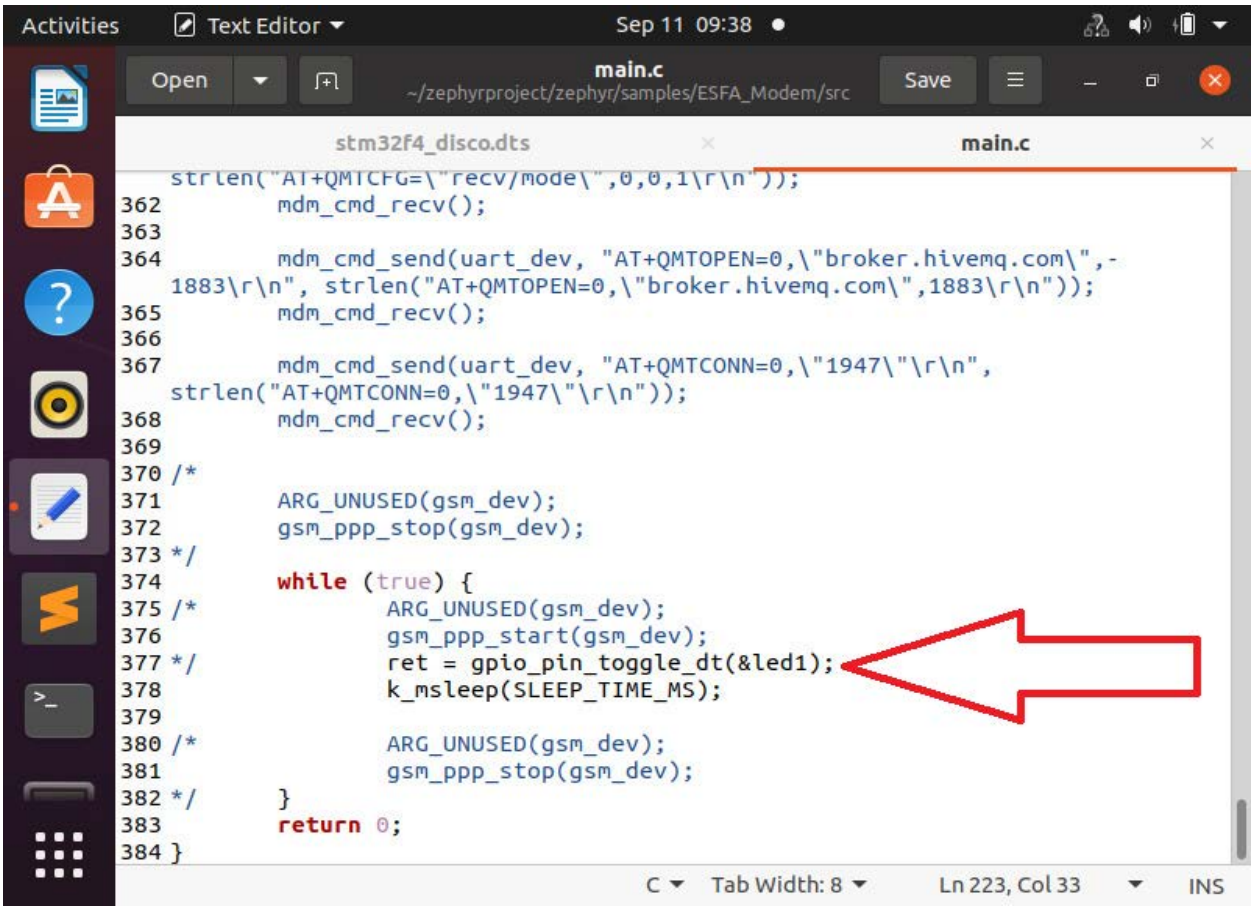
2. Changing the settings of a device tree

For adding a new led or change the associated pin of a led write a code similar to the below images:

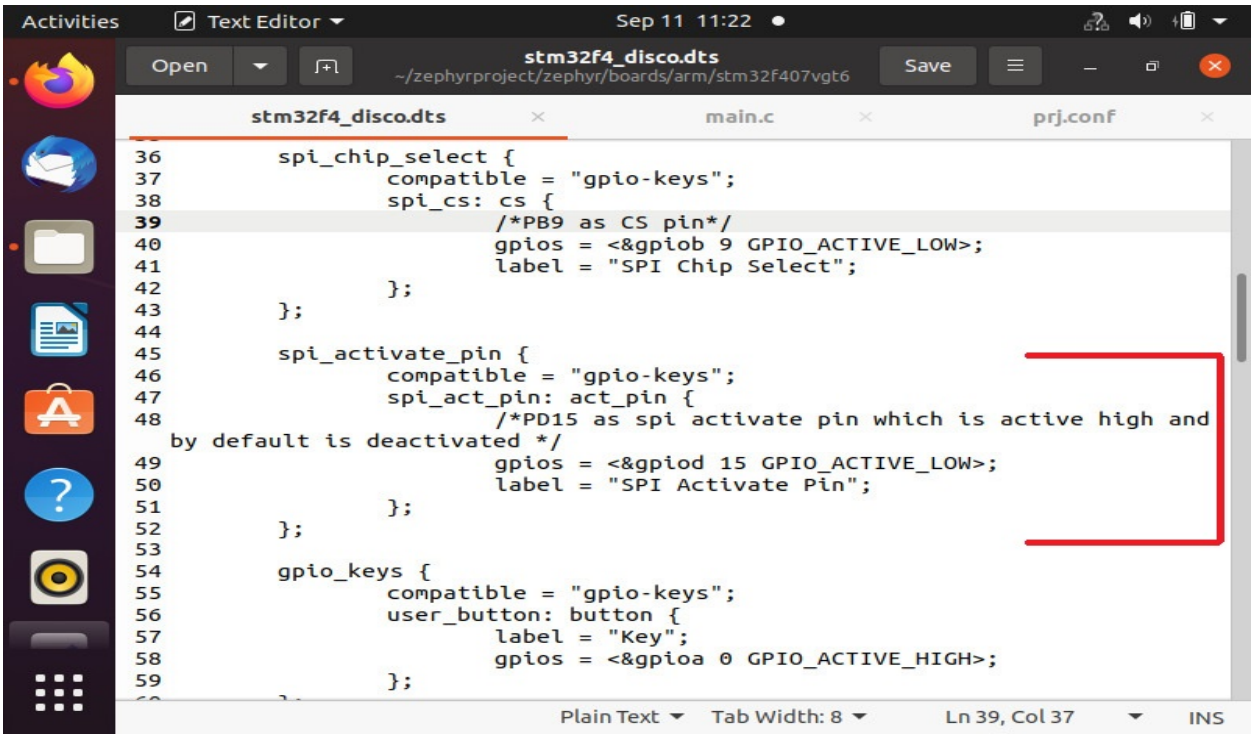

Activities
Text Editor
Sep 11 09:22
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
1 #define RESPONSE_DELAY 1000
2 #define SLEEP_TIME_MS 1000
3 #define SPI_CS DT_ALIAS(cs)
4 #define LED1_NODE DT_ALIAS(led1)
5 #define spi_act_pin DT_ALIAS(spiactpin)
6 #define BUF_SIZE 1500
7
8 #include <zephyr/drivers/gpio.h>
9 #include <zephyr/drivers/modem/gsm_ppp.h>
10 #include <zephyr/drivers/uart.h>
11 #include <zephyr/drivers/spi.h>
12 #include <zephyr/logging/log.h>
13 #include <zephyr/net/net_conn_mgr.h>
14 #include <zephyr/net/net_event.h>
15 #include <zephyr/net/net_mgmt.h>
16 #include <zephyr/shell/shell.h>
17 #include <zephyr/sys/printk.h>
18 #include <zephyr/zephyr.h>
19 #include <zephyr/device.h>
20 #include <string.h>
21 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
22
23 char rx_buf_spi[BUF_SIZE];
24 char rx_buf_uart[BUF_SIZE];
25 int rx_buf_uart_pos = 0;
26
C
Tab Width: 8
Ln 223, Col 33
INS

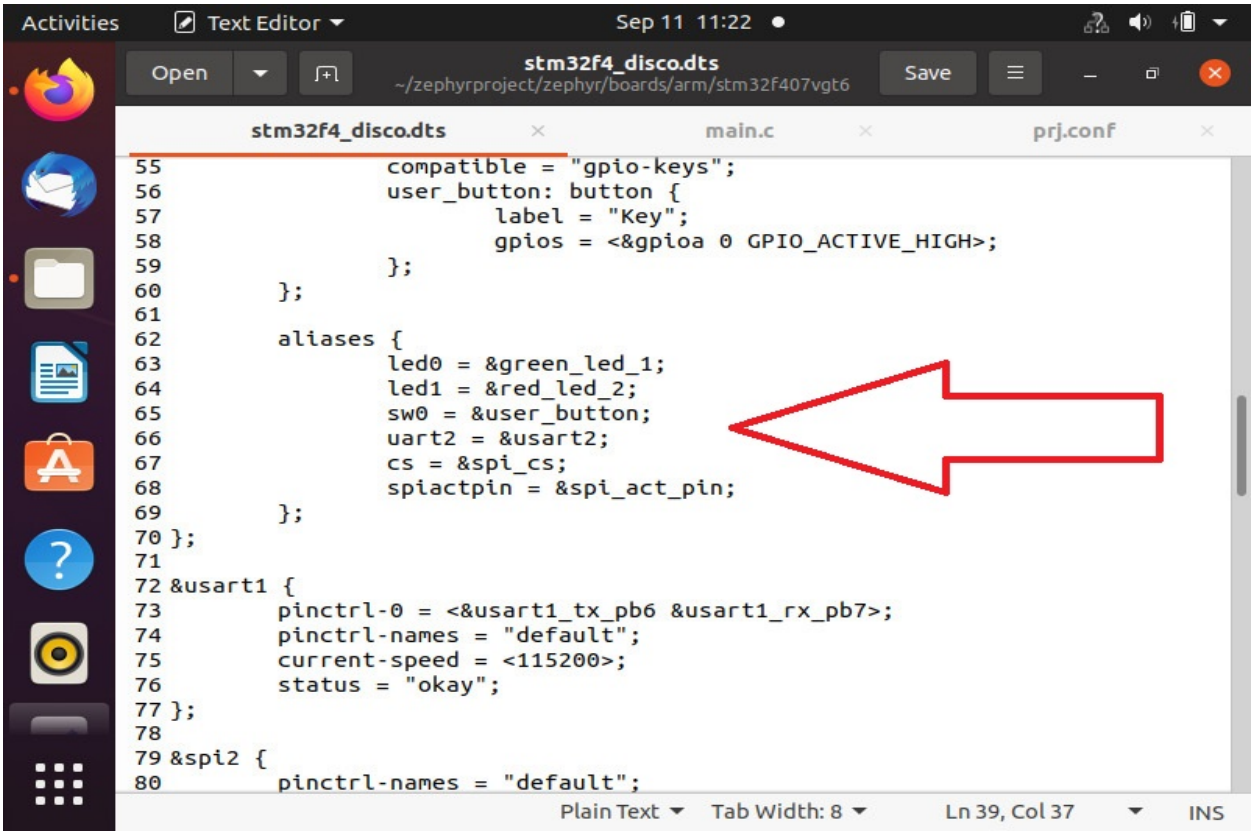

Activities
Text Editor
Sep 11 09:21
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
22
23 char rx_buf_spi[BUF_SIZE];
24 char rx_buf_uart[BUF_SIZE];
25 int rx_buf_uart_pos = 0;
26
27 const char END_MASSAGE[] = "\r\n";
28 const struct device *spi;
29 const struct device *uart_dev = DEVICE_DT_GET(DT_ALIAS(uart2));
30 static const struct gpio_dt_spec spi_activate_pin =
GPIO_DT_SPEC_GET(spi_act_pin, gpios);
31 static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
32 static const struct device *gsm_dev;
33 static struct net_mgmt_event_callback mgmt_cb;
34 static bool starting = IS_ENABLED(CONFIG_GSM_PPP_AUTOSTART);
35
36 /* queue to store up to 10 messages (aligned to 4-byte boundary) */
37 K_MSGQ_DEFINE(uart_msgq, 32, 10, 4);
38
39 struct spi_cs_control spi_cs = {
40 .gpio = GPIO_DT_SPEC_GET(SPI_CS, gpios)
41 };
42
43 struct spi_config spi_cfg = {
44 .operation = SPI_OP_MODE_SLAVE | SPI_WORD_SET(8),
45 .slave = 1,
C
Tab Width: 8
Ln 223, Col 33
INS

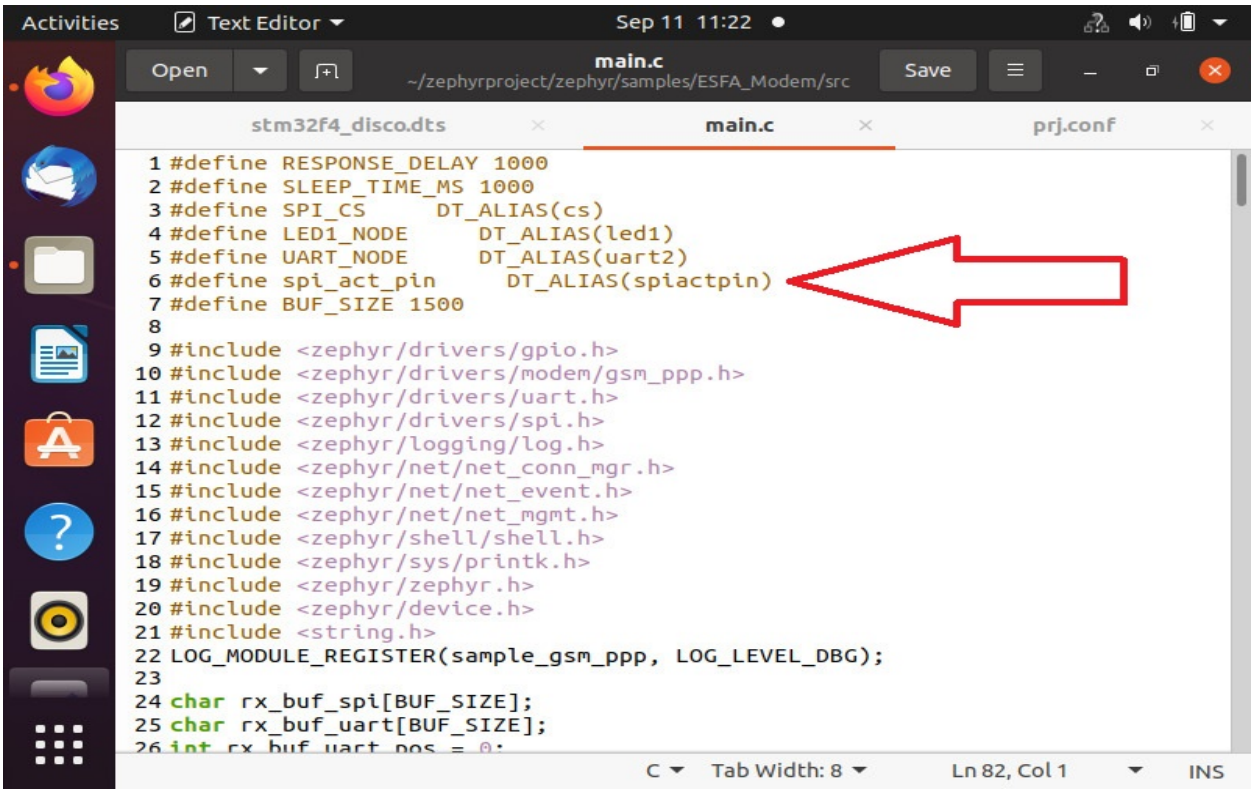

Activities
Text Editor
Sep 11 09:31
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
287
288 //starting spi communication between modem's MCU and main MCU:
289 spi_init();
290
291 //starting gsm-modem:
292 gsm_init();
293
294 gpio_pin_configure_dt(&spi_activate_pin, GPIO_OUTPUT_ACTIVE);
295 gpio_pin_set_dt(&spi_activate_pin, 1);
296 spi_receive();
297
298 int ret = gpio_pin_configure_dt(&led1, GPIO_OUTPUT_ACTIVE);
299
300 //tcp_init("postman-echo.com", "80");
301
302 //tcp_init("tcpbin.com", "4242");
303
304 //tcp_init("smtp.gmail.com", "465");
305
306 tcp_init("google.com", "80");
307
308 tcp_send("test1");
309 tcp_receive();
310 tcp_receive();
311
C
Tab Width: 8
Ln 223, Col 33
INS

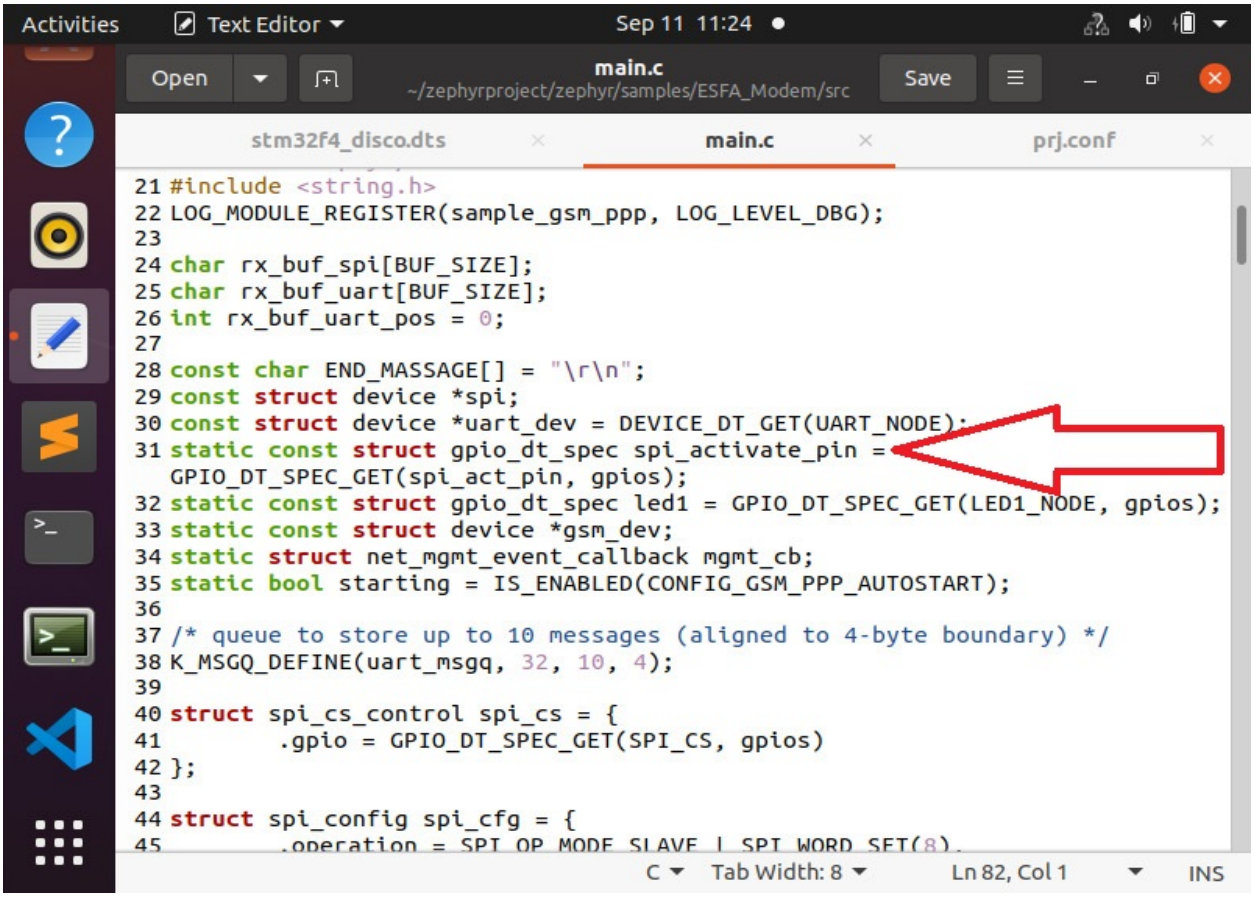

Activities
Text Editor
Sep 11 09:38
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
strlen("AT+QMTCFG=\"recv/mode\",0,0,1\r\n"));
362 mdm_cmd_recv();
363
364 mdm_cmd_send(uart_dev, "AT+QMTOPEN=0,\"broker.hivemq.com\",-
1883\r\n", strlen("AT+QMTOPEN=0,\"broker.hivemq.com\",1883\r\n"));
365 mdm_cmd_recv();
366
367 mdm_cmd_send(uart_dev, "AT+QMTCONN=0,\"1947\"\r\n",
strlen("AT+QMTCONN=0,\"1947\"\r\n"));
368 mdm_cmd_recv();
369
370 /*
371 ARG_UNUSED(gsm_dev);
372 gsm_ppp_stop(gsm_dev);
373 */
374 while (true) {
375 /* ARG_UNUSED(gsm_dev);
376 gsm_ppp_start(gsm_dev);
377 */ ret = gpio_pin_toggle_dt(&led1);
378 k_msleep(SLEEP_TIME_MS);
379
380 /* ARG_UNUSED(gsm_dev);
381 gsm_ppp_stop(gsm_dev);
382 */ }
383 return 0;
384 }
C
Tab Width: 8
Ln 223, Col 33
INS

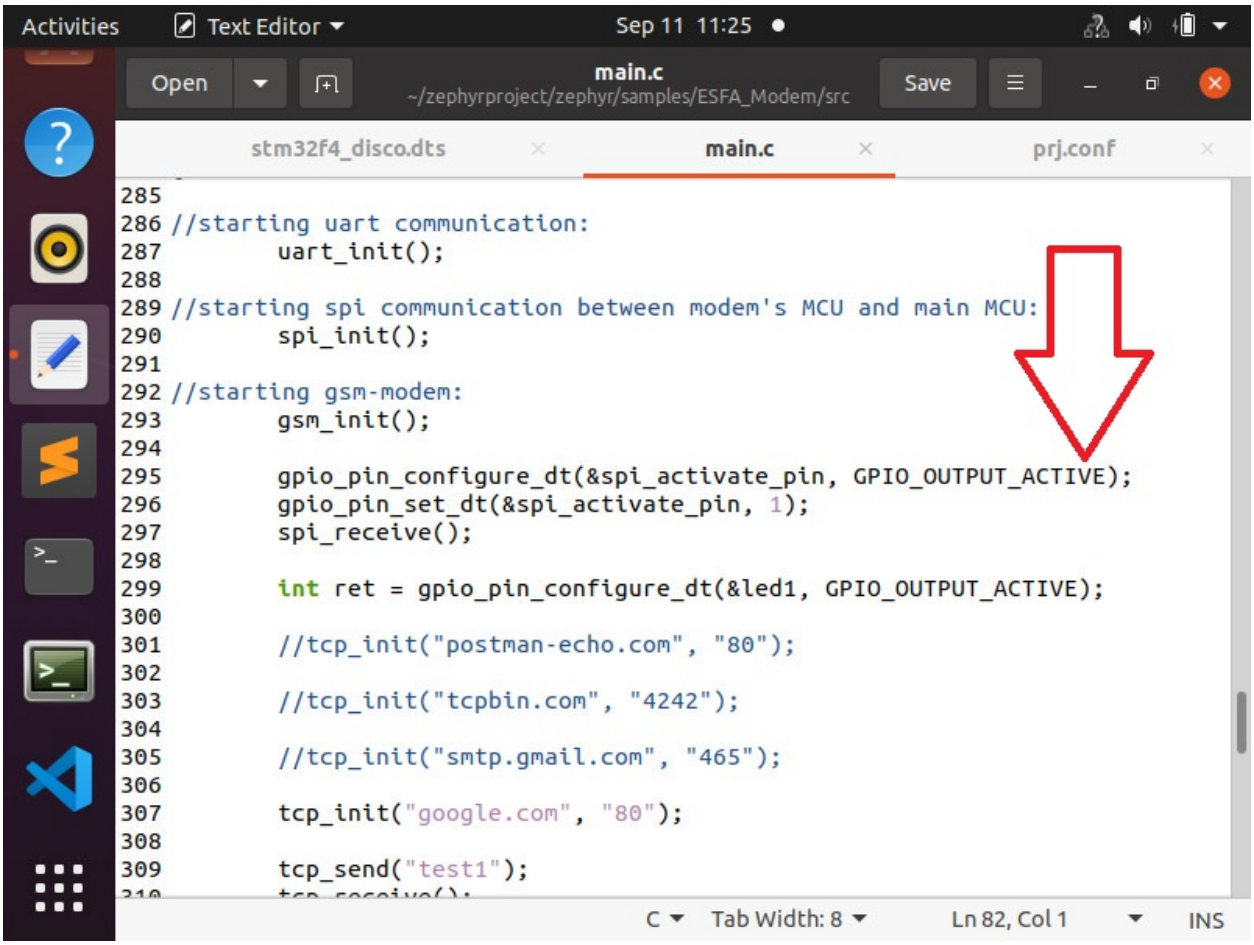
For pin configuration write a code similar to the below images:



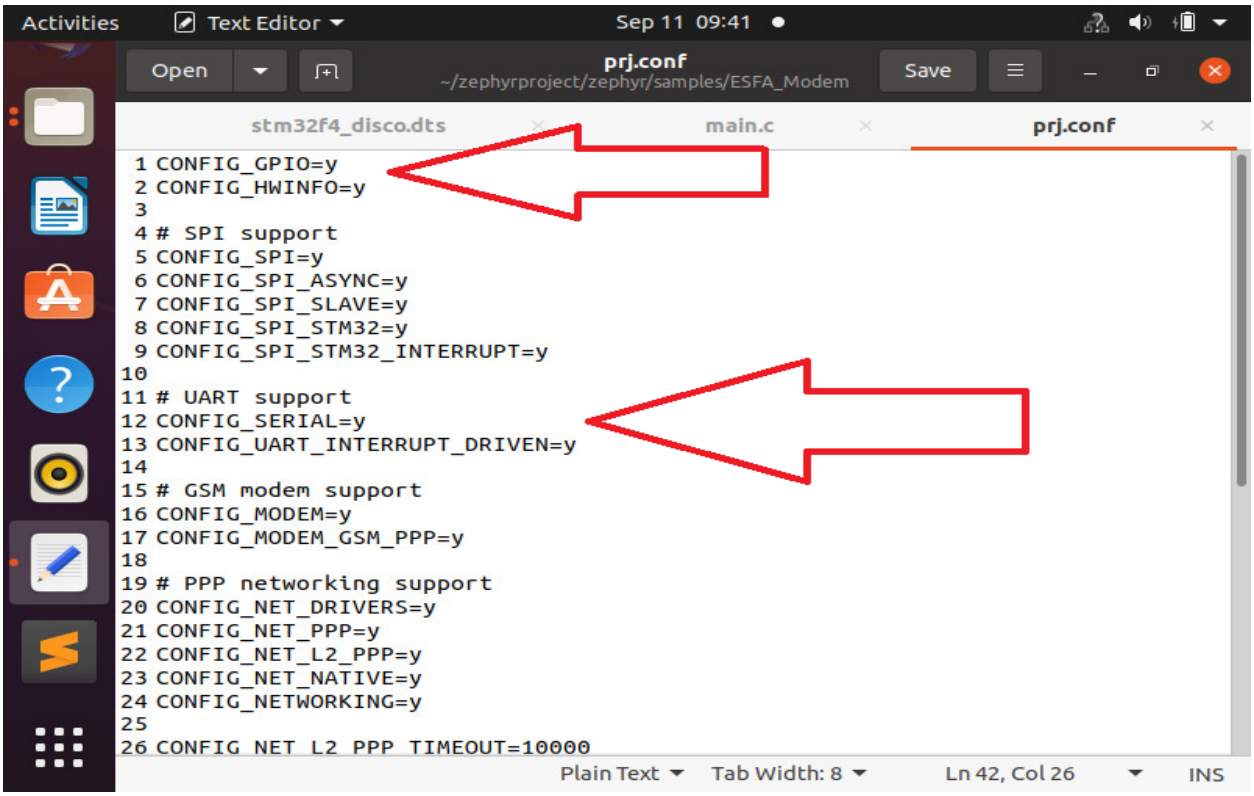


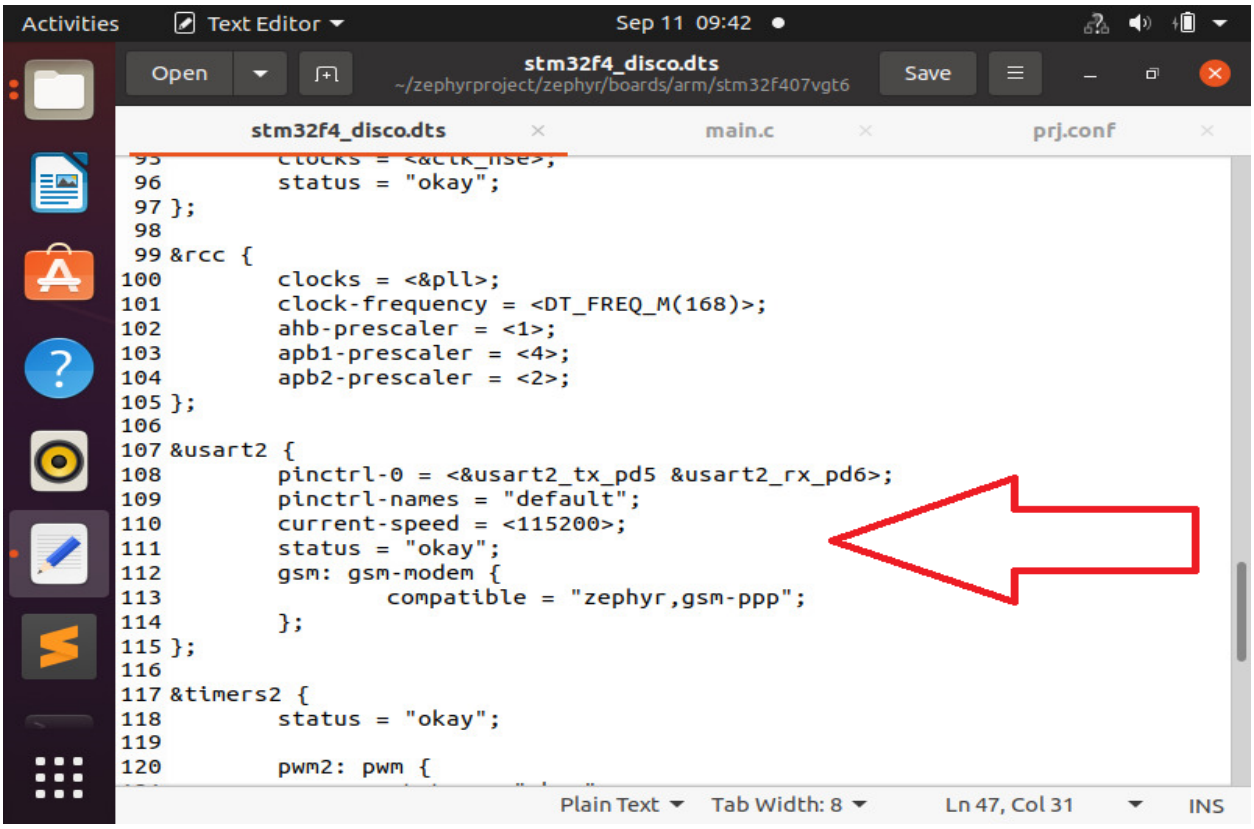

Activities
Text Editor
Sep 11 11:22
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
1 #define RESPONSE_DELAY 1000
2 #define SLEEP_TIME_MS 1000
3 #define SPI_CS DT_ALIAS(cs)
4 #define LED1_NODE DT_ALIAS(led1)
5 #define UART_NODE DT_ALIAS(uart2)
6 #define spi_act_pin DT_ALIAS(spiactpin)
7 #define BUF_SIZE 1500
8
9 #include <zephyr/drivers/gpio.h>
10 #include <zephyr/drivers/modem/gsm_ppp.h>
11 #include <zephyr/drivers/uart.h>
12 #include <zephyr/drivers/spi.h>
13 #include <zephyr/logging/log.h>
14 #include <zephyr/net/net_conn_mgr.h>
15 #include <zephyr/net/net_event.h>
16 #include <zephyr/net/net_mgmt.h>
17 #include <zephyr/shell/shell.h>
18 #include <zephyr/sys/printk.h>
19 #include <zephyr/zephyr.h>
20 #include <zephyr/device.h>
21 #include <string.h>
22 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
23
24 char rx_buf_spi[BUF_SIZE];
25 char rx_buf_uart[BUF_SIZE];
C
Tab Width: 8
Ln 82, Col 1
INS

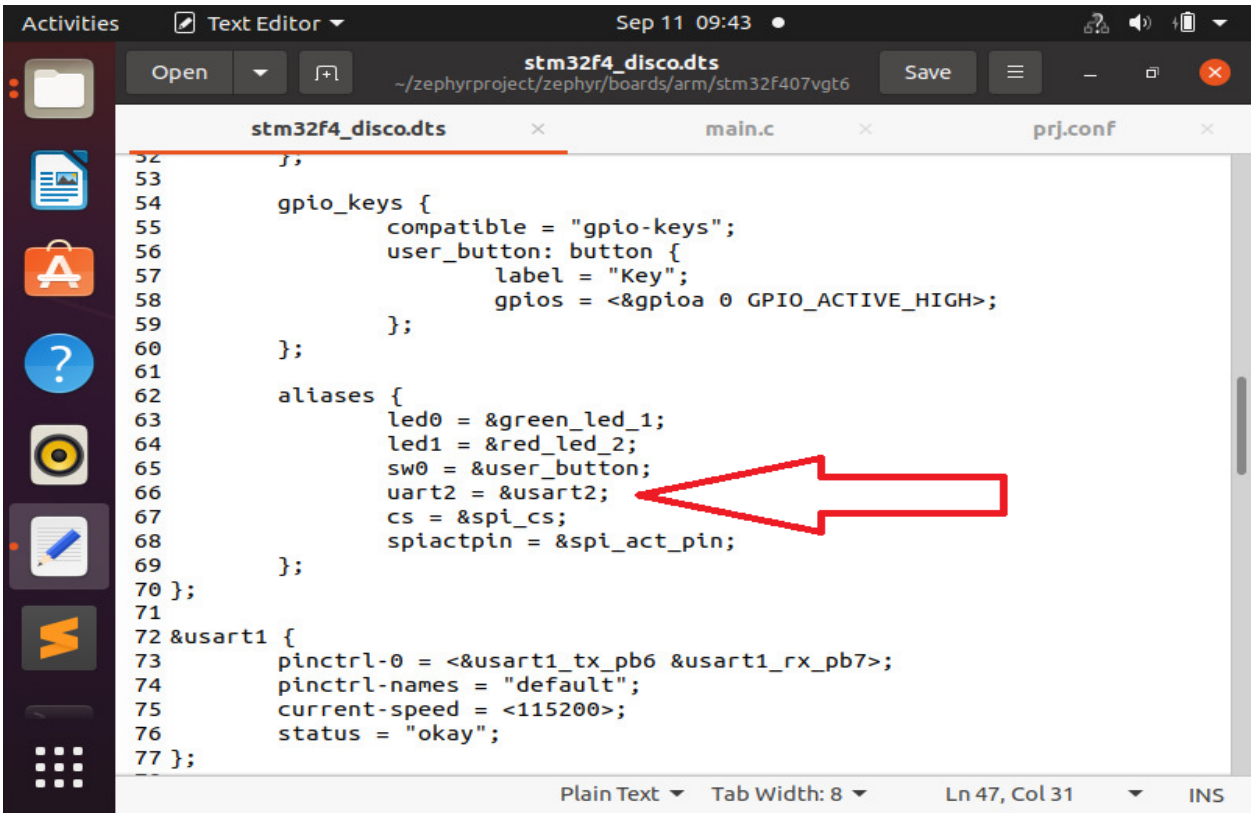

Activities
Text Editor
Sep 11 11:24
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
21 #include <string.h>
22 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
23
24 char rx_buf_spi[BUF_SIZE];
25 char rx_buf_uart[BUF_SIZE];
26 int rx_buf_uart_pos = 0;
27
28 const char END_MASSAGE[] = "\r\n";
29 const struct device *spi;
30 const struct device *uart_dev = DEVICE_DT_GET(UART_NODE);
31 static const struct gpio_dt_spec spi_activate_pin =
GPIO_DT_SPEC_GET(spi_act_pin, gpios);
32 static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
33 static const struct device *gsm_dev;
34 static struct net_mgmt_event_callback mgmt_cb;
35 static bool starting = IS_ENABLED(CONFIG_GSM_PPP_AUTOSTART);
36
37 /* queue to store up to 10 messages (aligned to 4-byte boundary) */
38 K_MSGQ_DEFINE(uart_msgq, 32, 10, 4);
39
40 struct spi_cs_control spi_cs = {
41 .gpio = GPIO_DT_SPEC_GET(SPI_CS, gpios)
42 };
43
44 struct spi_config spi_cfg = {
C
Tab Width: 8
Ln 82, Col 1
INS

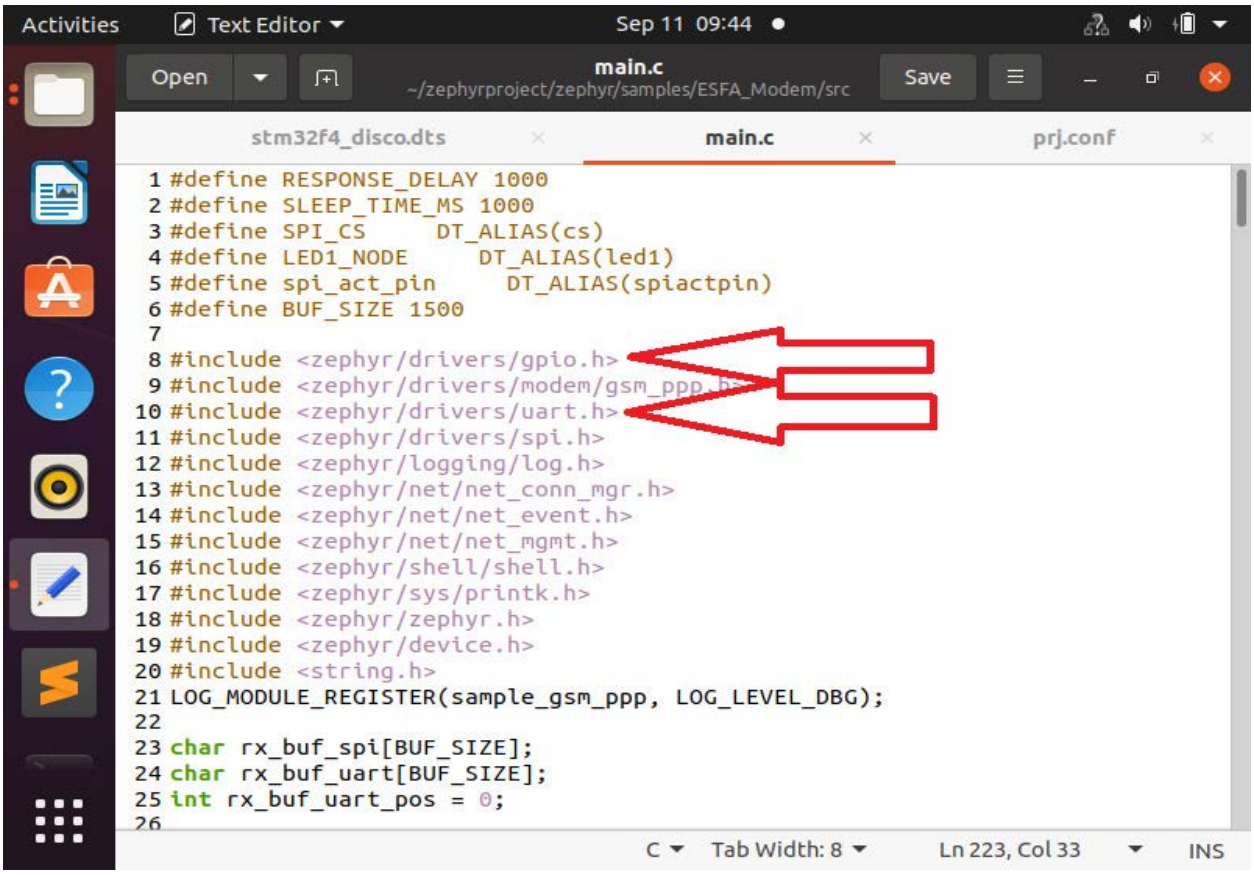

Activities
Text Editor
Sep 11 11:25
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
285
286 //starting uart communication:
287 uart_init();
288
289 //starting spi communication between modem's MCU and main MCU:
290 spi_init();
291
292 //starting gsm-modem:
293 gsm_init();
294
295 gpio_pin_configure_dt(&spi_activate_pin, GPIO_OUTPUT_ACTIVE);
296 gpio_pin_set_dt(&spi_activate_pin, 1);
297 spi_receive();
298
299 int ret = gpio_pin_configure_dt(&led1, GPIO_OUTPUT_ACTIVE);
300
301 //tcp_init("postman-echo.com", "80");
302
303 //tcp_init("tcpbin.com", "4242");
304
305 //tcp_init("smtp.gmail.com", "465");
306
307 tcp_init("google.com", "80");
308
309 tcp_send("test1");
C
Tab Width: 8
Ln 82, Col 1
INS

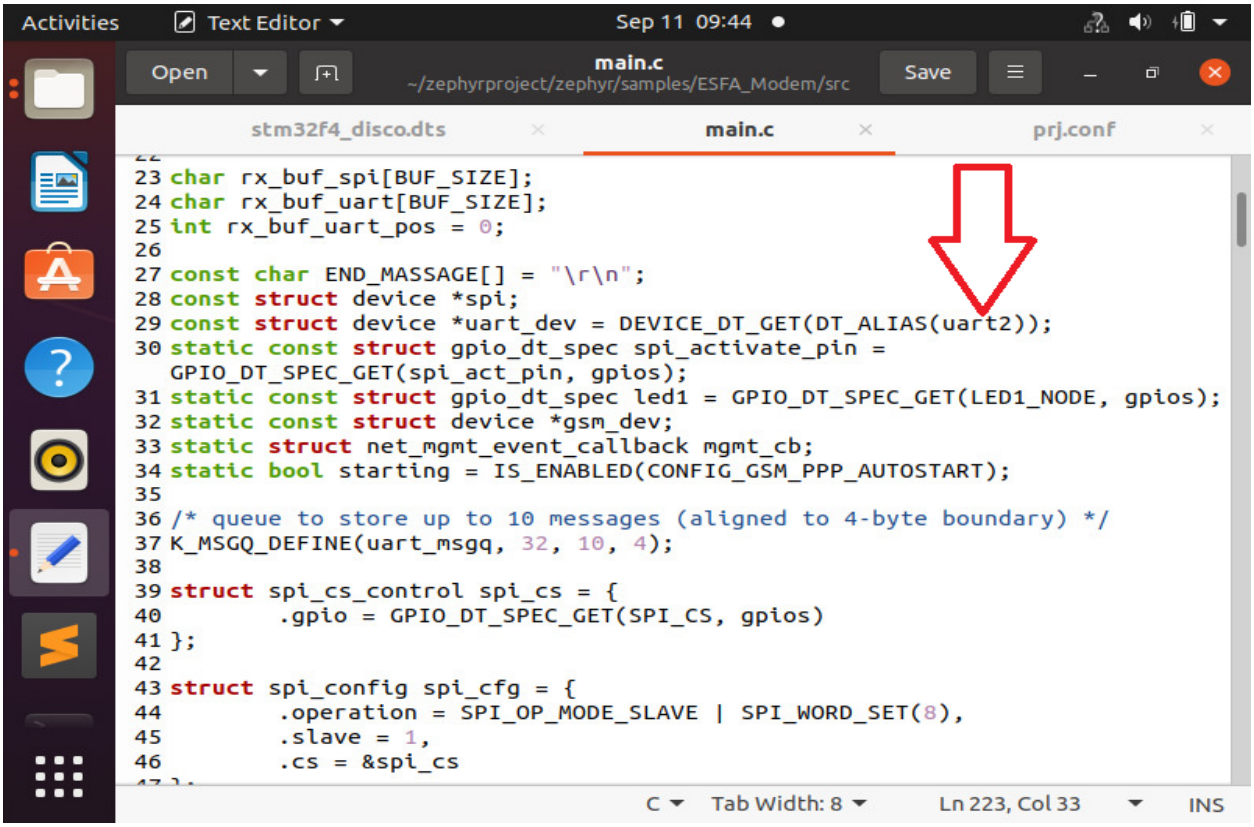
For using uart write a code similar to the below images:

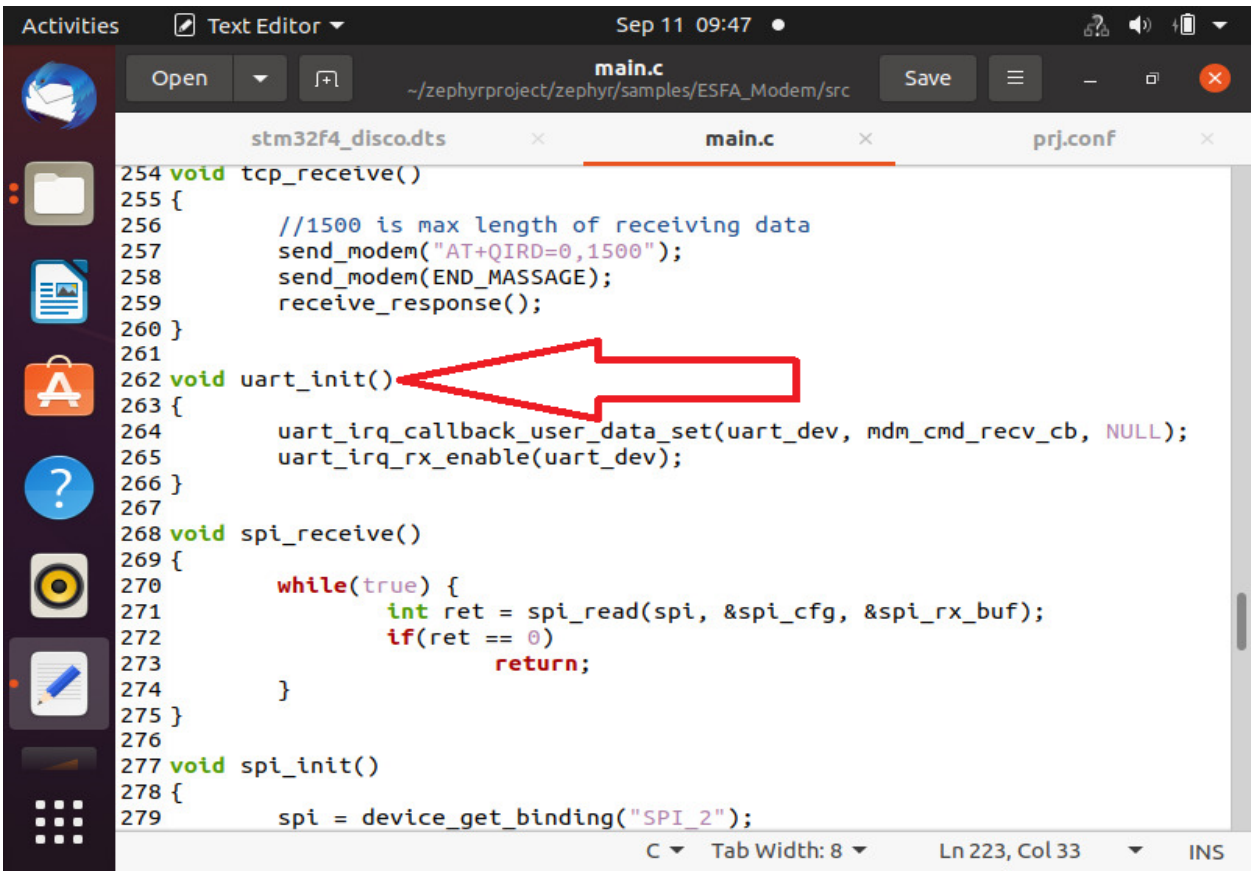





Activities
Text Editor
Sep 11 09:43
Open
stm32f4_disco.dts
~/zephyrproject/zephyr/boards/arm/stm32f407vgt6
Save
stm32f4_disco.dts
main.c
prj.conf
53
54 gpio_keys {
55 compatible = "gpio-keys";
56 user_button: button {
57 label = "Key";
58 gpios = <&gpioa 0 GPIO_ACTIVE_HIGH>;
59 };
60 };
61
62 aliases {
63 led0 = &green_led_1;
64 led1 = &red_led_2;
65 sw0 = &user_button;
66 uart2 = &usart2;
67 cs = &spi_cs;
68 spiactpin = &spi_act_pin;
69 };
70 };
71
72 &usart1 {
73 pinctrl-0 = <&usart1_tx_pb6 &usart1_rx_pb7>;
74 pinctrl-names = "default";
75 current-speed = <115200>;
76 status = "okay";
77 };
Plain Text
Tab Width: 8
Ln 47, Col 31
INS


Activities
Text Editor
Sep 11 09:44
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
1 #define RESPONSE_DELAY 1000
2 #define SLEEP_TIME_MS 1000
3 #define SPI_CS DT_ALIAS(cs)
4 #define LED1_NODE DT_ALIAS(led1)
5 #define spi_act_pin DT_ALIAS(spiactpin)
6 #define BUF_SIZE 1500
7
8 #include <zephyr/drivers/gpio.h>
9 #include <zephyr/drivers/modem/gsm_ppp.h>
10 #include <zephyr/drivers/uart.h>
11 #include <zephyr/drivers/spi.h>
12 #include <zephyr/logging/log.h>
13 #include <zephyr/net/net_conn_mgr.h>
14 #include <zephyr/net/net_event.h>
15 #include <zephyr/net/net_mgmt.h>
16 #include <zephyr/shell/shell.h>
17 #include <zephyr/sys/printk.h>
18 #include <zephyr/zephyr.h>
19 #include <zephyr/device.h>
20 #include <string.h>
21 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
22
23 char rx_buf_spi[BUF_SIZE];
24 char rx_buf_uart[BUF_SIZE];
25 int rx_buf_uart_pos = 0;
26
C
Tab Width: 8
Ln 223, Col 33
INS

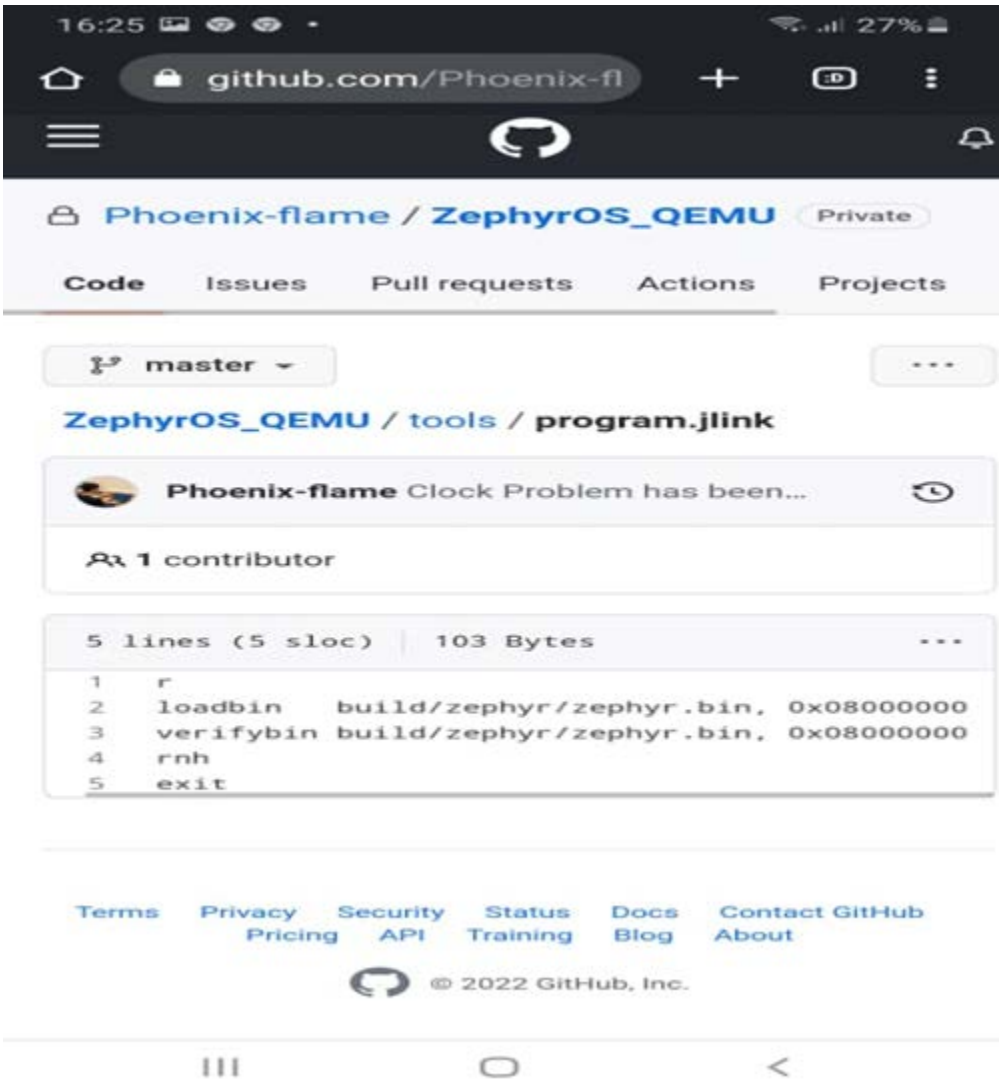

Activities
Text Editor
Sep 11 09:44
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
23 char rx_buf_spi[BUF_SIZE];
24 char rx_buf_uart[BUF_SIZE];
25 int rx_buf_uart_pos = 0;
26
27 const char END_MASSAGE[] = "\r\n";
28 const struct device *spi;
29 const struct device *uart_dev = DEVICE_DT_GET(DT_ALIAS(uart2));
30 static const struct gpio_dt_spec spi_activate_pin =
GPIO_DT_SPEC_GET(spi_act_pin, gpios);
31 static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
32 static const struct device *gsm_dev;
33 static struct net_mgmt_event_callback mgmt_cb;
34 static bool starting = IS_ENABLED(CONFIG_GSM_PPP_AUTOSTART);
35
36 /* queue to store up to 10 messages (aligned to 4-byte boundary) */
37 K_MSGQ_DEFINE(uart_msgq, 32, 10, 4);
38
39 struct spi_cs_control spi_cs = {
40 .gpio = GPIO_DT_SPEC_GET(SPI_CS, gpios)
41 };
42
43 struct spi_config spi_cfg = {
44 .operation = SPI_OP_MODE_SLAVE | SPI_WORD_SET(8),
45 .slave = 1,
46 .cs = &spi_cs
C
Tab Width: 8
Ln 223, Col 33
INS

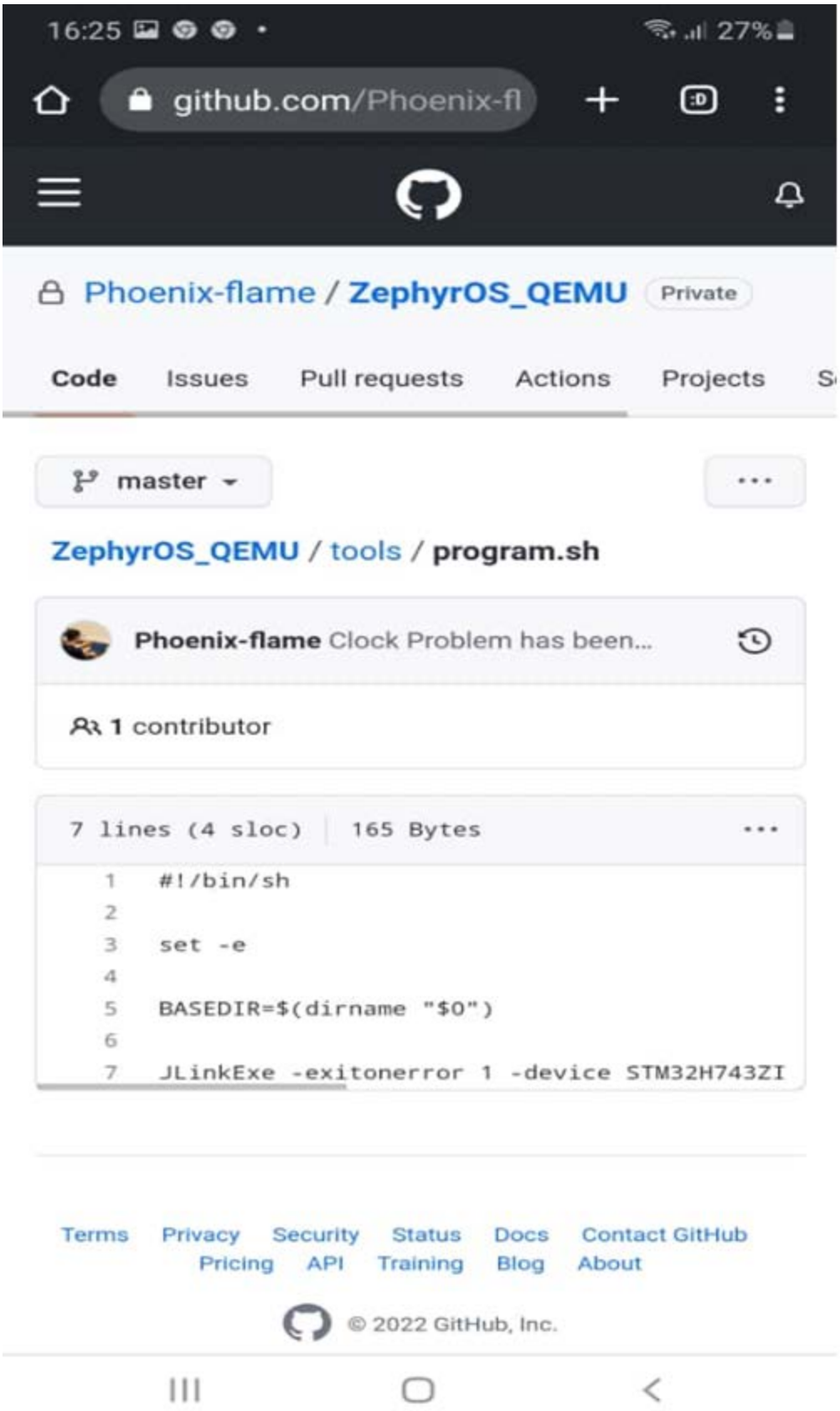

Activities
Text Editor
Sep 11 09:47
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
stm32f4_disco.dts
main.c
prj.conf
254 void tcp_receive()
255 {
256 //1500 is max length of receiving data
257 send_modem("AT+QIRD=0,1500");
258 send_modem(END_MASSAGE);
259 receive_response();
260 }
261
262 void uart_init()
263 {
264 uart_irq_callback_user_data_set(uart_dev, mdm_cmd_recv_cb, NULL);
265 uart_irq_rx_enable(uart_dev);
266 }
267
268 void spi_receive()
269 {
270 while(true) {
271 int ret = spi_read(spi, &spi_cfg, &spi_rx_buf);
272 if(ret == 0)
273 return;
274 }
275 }
276
277 void spi_init()
278 {
279 spi = device_get_binding("SPI_2");
C
Tab Width: 8
Ln 223, Col 33
INS

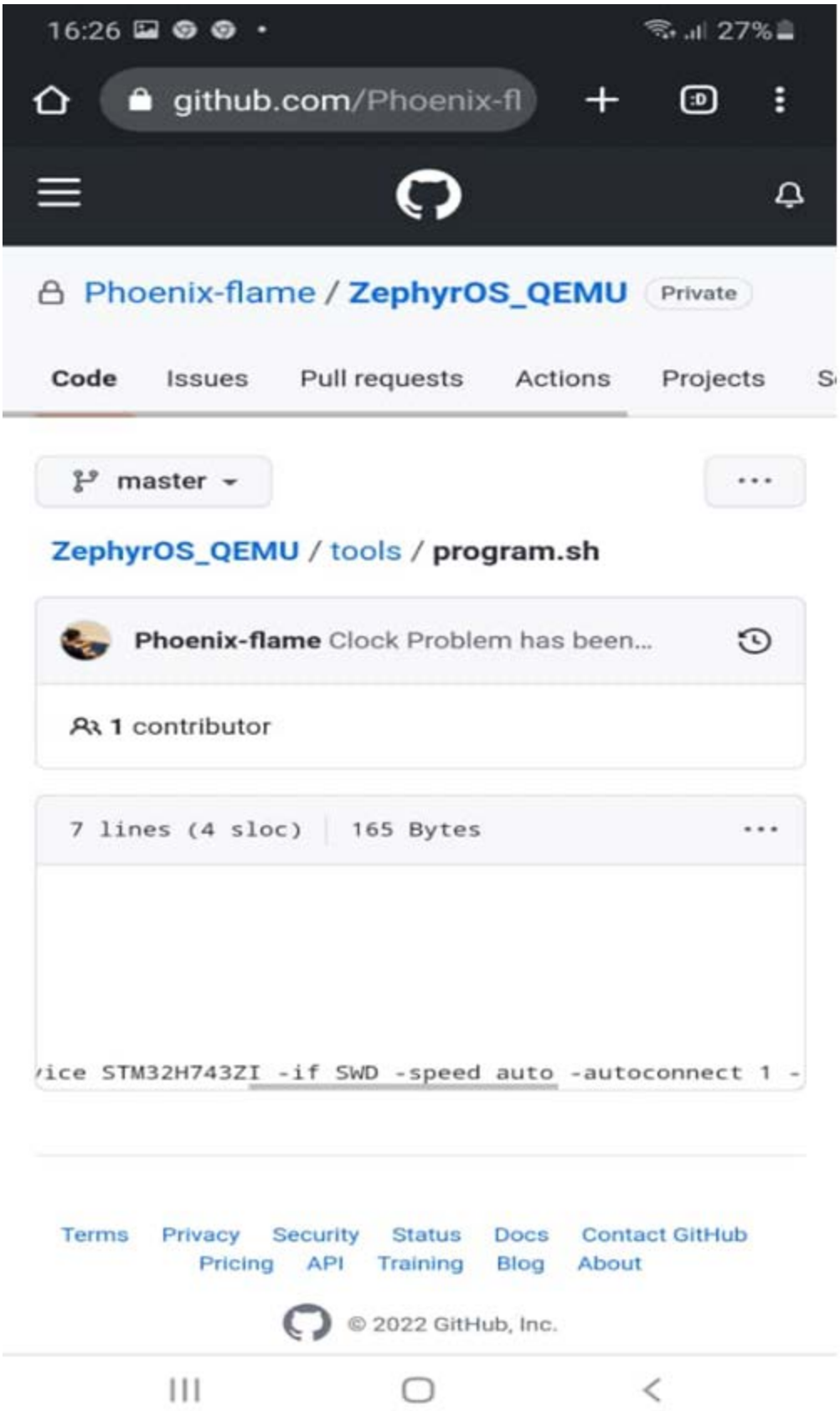
Be careful you must know which ports on board are associated with UART unless you will get error.

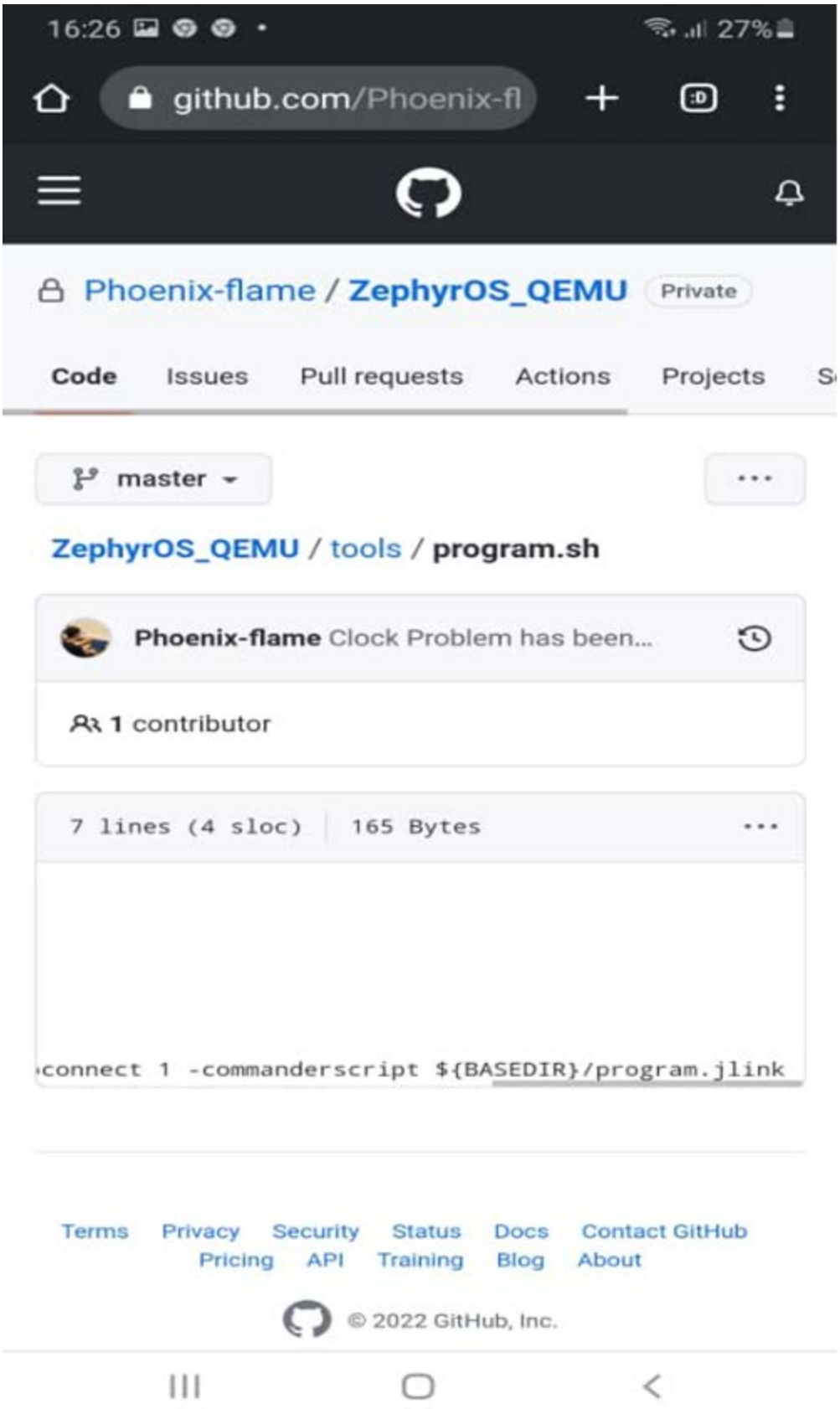
Programing

To Program the board with your Zephyr code, just open the directory of your project where you can see src folder. Then make 2 files called program.sh and program.jlink. program.sh calls JFlashExe which is for programming the board (**Be careful you must install version 6.98 of JLink-Segger software unless you may not able to connect the board**). And program.jlink calls zephyr.bin which is final executable binary file. Images below are contents of these 2 files:




16:25
27%
github.com/Phoenix-fl
Phoenix-flame / ZephyrOS_QEMU
Private
Code
Issues
Pull requests
Actions
Projects
master
ZephyrOS_QEMU / tools / program.sh
Phoenix-flame Clock Problem has been...
1 contributor
7 lines (4 sloc) | 165 Bytes
1 #!/bin/sh
2
3 set -e
4
5 BASEDIR=$(dirname "$0")
6
7 JLinkExe -exitonerror 1 -device STM32H743ZI
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
© 2022 GitHub, Inc.


16:26
27%
github.com/Phoenix-fl
Phoenix-flame / ZephyrOS_QEMU
Private
Code
Issues
Pull requests
Actions
Projects
master
ZephyrOS_QEMU / tools / program.sh
Phoenix-flame Clock Problem has been...
1 contributor
7 lines (4 sloc) | 165 Bytes
vice STM32H743ZI -if SWD -speed auto -autoconnect 1 -
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
© 2022 GitHub, Inc.

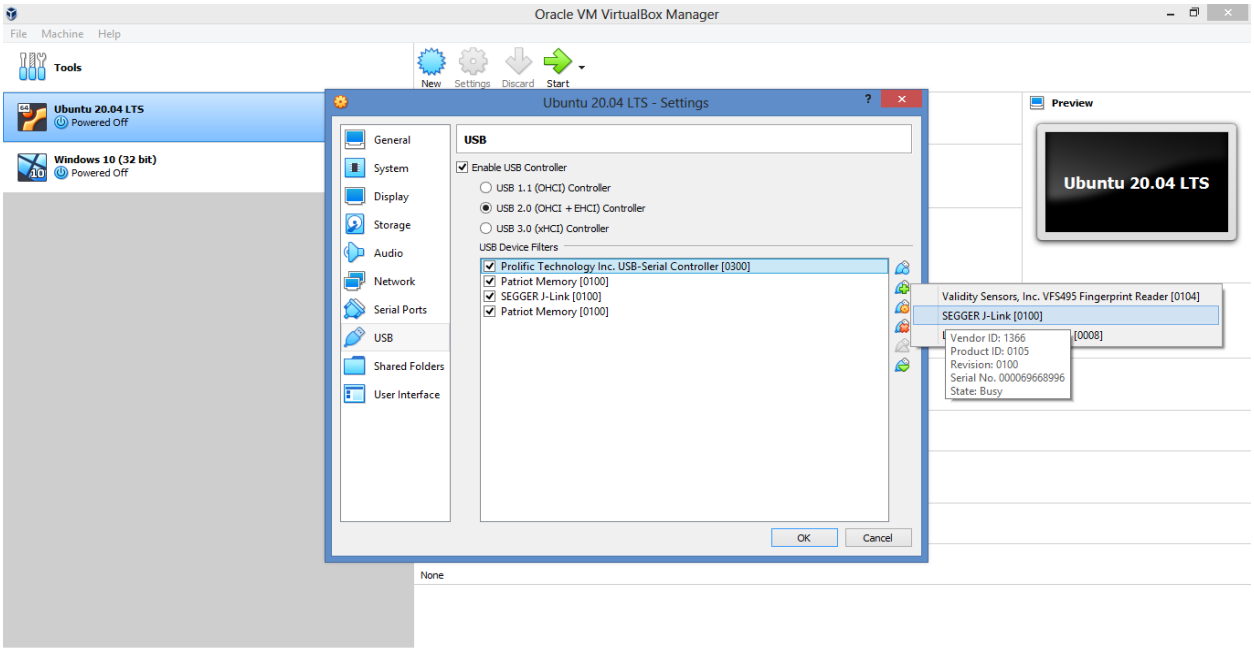

16:26
27%
github.com/Phoenix-fl
Phoenix-flame / ZephyrOS_QEMU
Private
Code
Issues
Pull requests
Actions
Projects
master
ZephyrOS_QEMU / tools / program.sh
Phoenix-flame Clock Problem has been...
1 contributor
7 lines (4 sloc)
165 Bytes
connect 1 -commanderscript ${BASEDIR}/program.jlink
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About
© 2022 GitHub, Inc.

3. Connecting J-Link Debugger to Modem Board

Recognize and access USB ports by Oracle VM

To recognize and access USB ports by Oracle VM, at first install new extension of Oracle available on https://www.oracle.com/fr/virtualization/technologies/vm/downloads/virtualbox-downloads.html#extpack

Then open Oracle VM, go to tools, go to Extensions tab and add downloaded extension. Go to settings, go to USB tab and select USB 2.0. Now plug in your device and add it to Oracle's USB filters like below image.



At the end unplug your device and plug it again. Now you can see your device in Oracle VM.

Choosing Debugger Tool

At first we used ST-Link for debugging, the problem with ST-Link was that there was a broken resistor and caused ST-Link not working all the time(ST wasn't recognized by computer). It just works once and then not working for several times. So we decided to change our debugger tool to J-Link. So we installed JFlashExe software, set the frequency at 200 kHz and connect J-Link to the Modem board through a Xh cable. Notice that there is "st-util" command for checking ST-Link connection status and st-gui software for connecting to ST-Link easier.

Set up the VScode debugger

At first open VScode software, go to extensions tab at right side of showing window, then write "cortex-debug" and install the debugger tool.

After that we create files called launch.json and tasks.json and filled them with launch.json and tasks.json on https://github.com/Phoenix-flame/ZephyrOS_QEMU/tree/master/.vscode. **Be careful we must change file paths to our own file paths which must be a full address not relative.** Launch.json demonstrate debugger properties, device properties like stm32's version and the executable binary file(.elf) which board is programmed with. Tasks.json demonstrate tasks that can be executed by debugger like makefile task, run task and etc. at run time launch.json is run and tasks.json won't run and is just for declaring tasks. **Be careful J-Link works at 200 kHz so statement "serverArgs = ["-speed", "200"] " is added to launch.json**. At the end click on play icon which is for debugging and wait to get at first of your main code. **If debugger paused on "compiler_barrier" function, just comment that line.**

Be careful you must install gdb-multiarch with sudo apt-get install command first, then make a shortcut of it and name it arm-none-eabi-gdb.

Side Items Learned

1. There is a SWO viewer software which shows transferring data on serial ports.

2. For programming with STM32CubeMX do the followings:

1. Select the board which you want to program

2. Pin configuration: declare output and input pins

3. Project manager: define the properties of your project such as project name and etc. Be careful you must set Toolchain / IDE to MDK-ARM to use keil debugger and STM32cubeIDE to use TrueStudio.

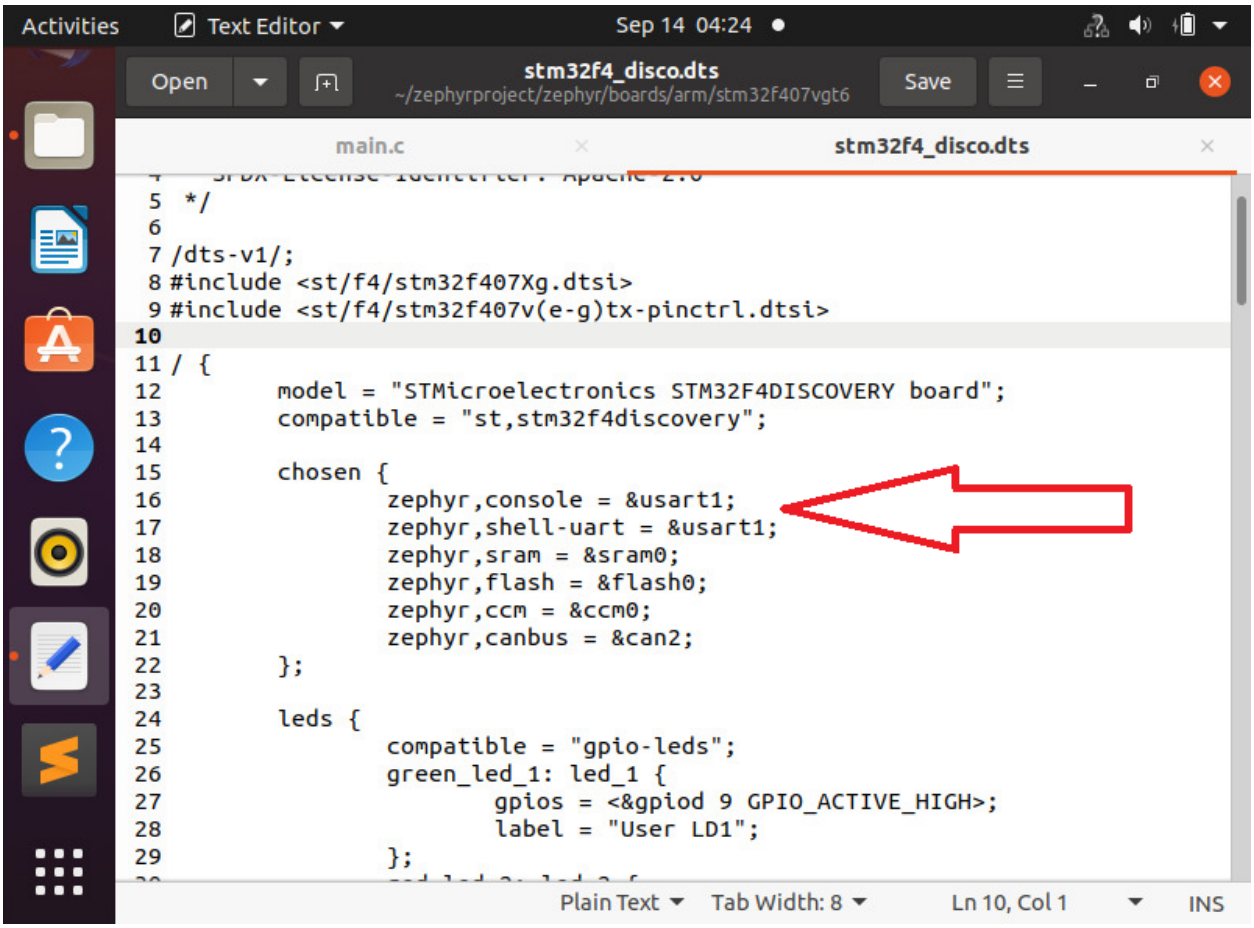
4.click on Generate Code icon and then open the debugger. Be careful you must set compiler and set the connect under reset option after debugger opens. Then build the project and at the end flash it to the board. Notice that for implementing UART we used “CuteCom” application to monitor serial ports.

There is blink sample at D:\STM32\STM32\Projects.

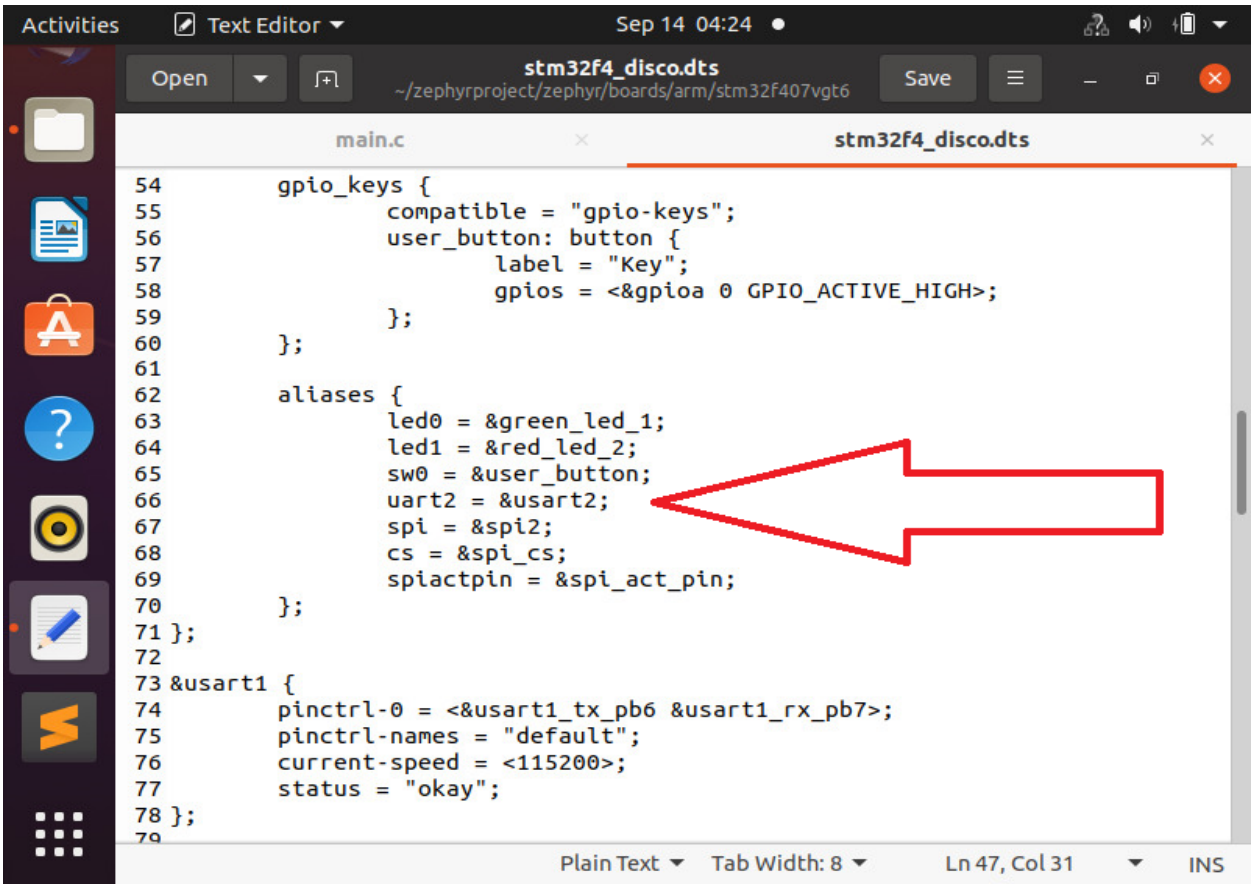
3. There are commands called “echo” and “cat” for sending and receiving data through UART serial ports.

4. Register Modem in Local Network via gsm

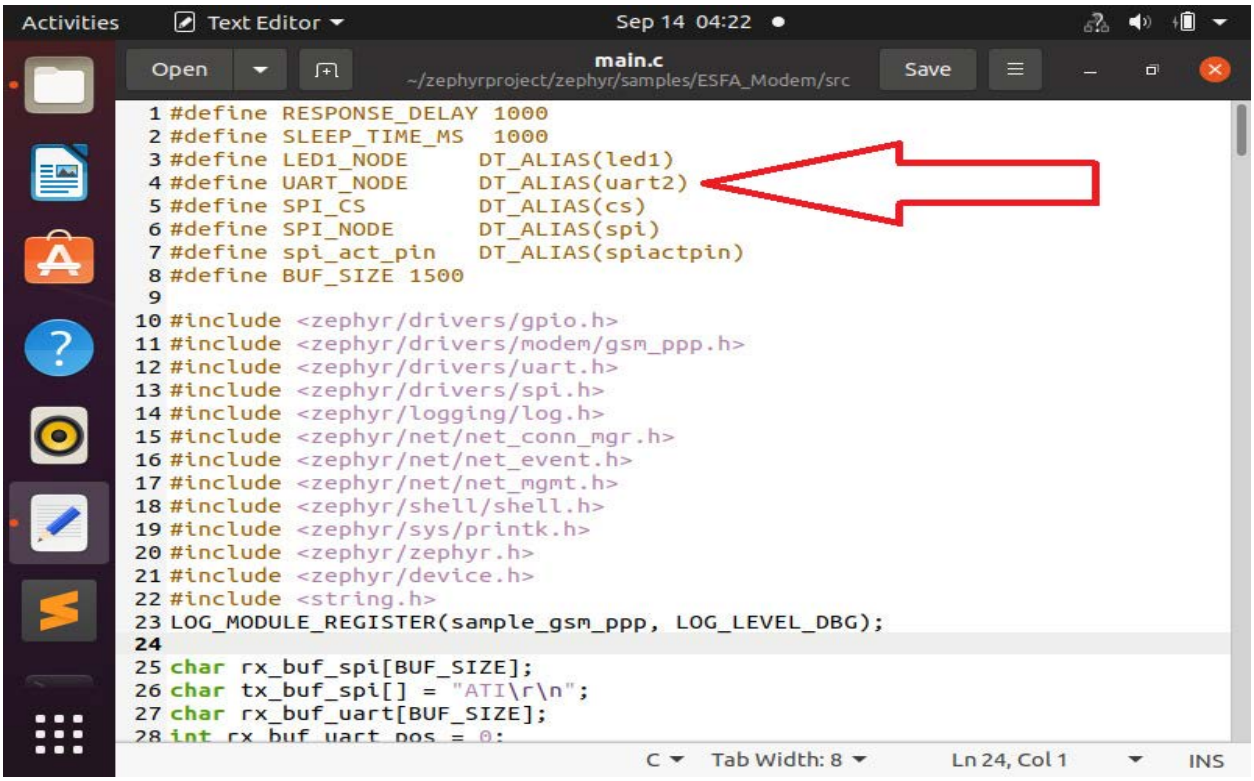
At First we found a gsm sample at zephyrproject/zephyr/samples/net and copied its src code into our main code. **Be careful for copying we must copy src code, prj.conf and device tree. For device tree we must get uart2 from shell and console, so we give uart1 to shell and console.** there is the code we wrote in stm32f407vgt6 device tree:

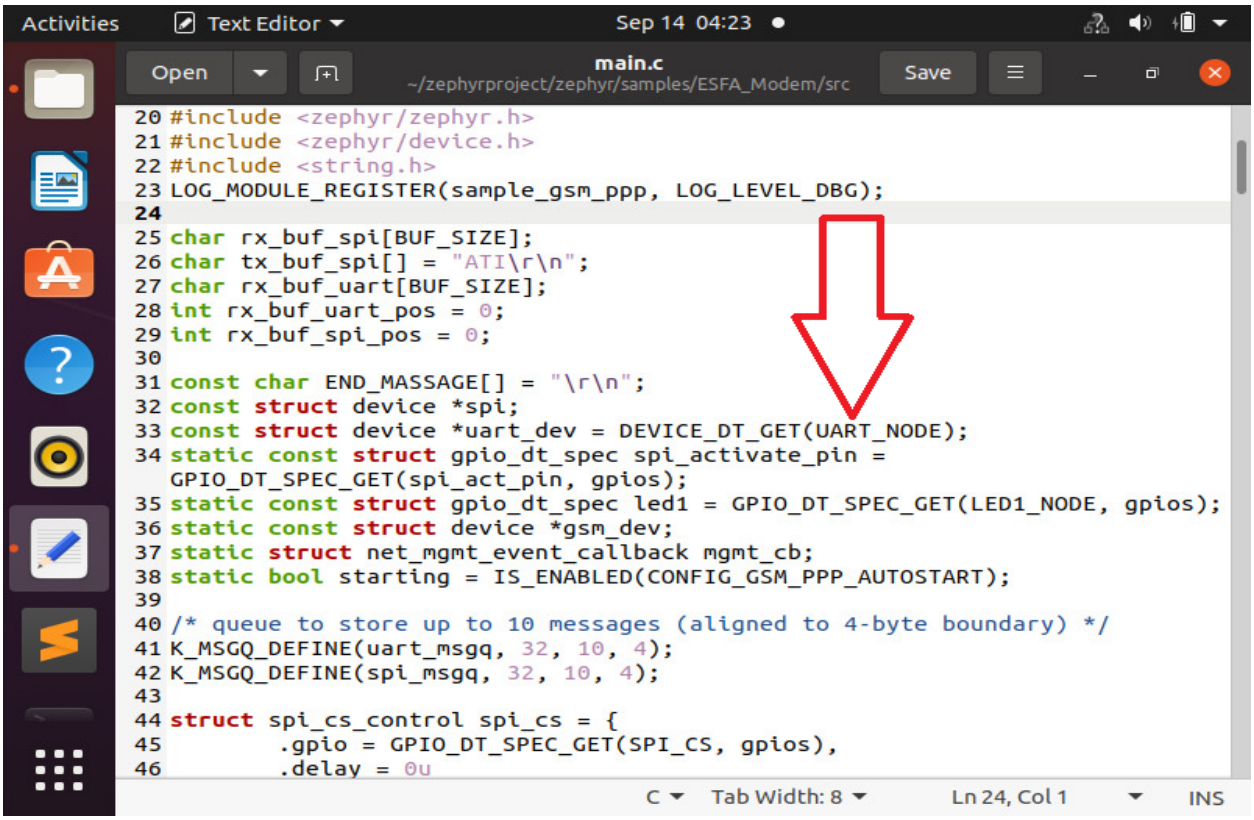


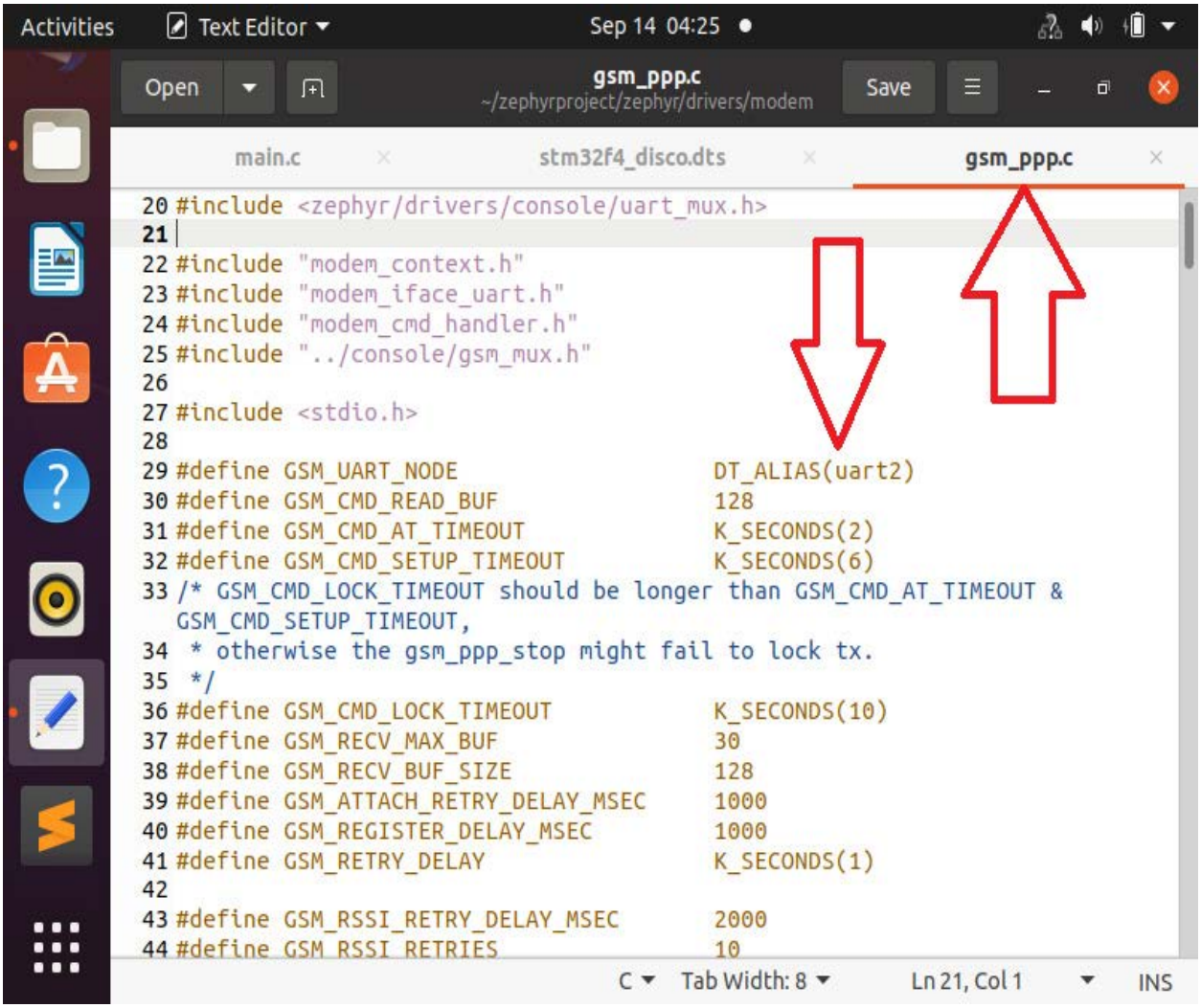

Activities
Text Editor
Sep 14 04:24
Open
stm32f4_disco.dts
~/zephyrproject/zephyr/boards/arm/stm32f407vgt6
Save
main.c
stm32f4_disco.dts
100 };
101
102 &rcc {
103 clocks = <&pll>;
104 clock-frequency = <DT_FREQ_M(168)>;
105 ahb-prescaler = <1>;
106 apb1-prescaler = <4>;
107 apb2-prescaler = <2>;
108 };
109
110 &usart2 {
111 pinctrl-0 = <&usart2_tx_pd5 &usart2_rx_pd6>;
112 pinctrl-names = "default";
113 current-speed = <115200>;
114 status = "okay";
115 gsm: gsm-modem {
116 compatible = "zephyr,gsm-ppp";
117 };
118 };
119
120 &timers2 {
121 status = "okay";
122
123 pwm2: pwm {
124 status = "okay";
Plain Text
Tab Width: 8
Ln 47, Col 31
INS


Activities
Text Editor
Sep 14 04:24
Open
stm32f4_disco.dts
~/zephyrproject/zephyr/boards/arm/stm32f407vgt6
Save
main.c
stm32f4_disco.dts
54 gpio_keys {
55 compatible = "gpio-keys";
56 user_button: button {
57 label = "Key";
58 gpios = <&gpioa 0 GPIO_ACTIVE_HIGH>;
59 };
60 };
61
62 aliases {
63 led0 = &green_led_1;
64 led1 = &red_led_2;
65 sw0 = &user_button;
66 uart2 = &usart2;
67 spi = &spi2;
68 cs = &spi_cs;
69 spiactpin = &spi_act_pin;
70 };
71 };
72
73 &usart1 {
74 pinctrl-0 = <&usart1_tx_pb6 &usart1_rx_pb7>;
75 pinctrl-names = "default";
76 current-speed = <115200>;
77 status = "okay";
78 };
Plain Text
Tab Width: 8
Ln 47, Col 31
INS

`


Activities
Text Editor
Sep 14 04:22
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
1 #define RESPONSE_DELAY 1000
2 #define SLEEP_TIME_MS 1000
3 #define LED1_NODE DT_ALIAS(led1)
4 #define UART_NODE DT_ALIAS(uart2)
5 #define SPI_CS DT_ALIAS(cs)
6 #define SPI_NODE DT_ALIAS(spi)
7 #define spi_act_pin DT_ALIAS(spiactpin)
8 #define BUF_SIZE 1500
9
10 #include <zephyr/drivers/gpio.h>
11 #include <zephyr/drivers/modem/gsm_ppp.h>
12 #include <zephyr/drivers/uart.h>
13 #include <zephyr/drivers/spi.h>
14 #include <zephyr/logging/log.h>
15 #include <zephyr/net/net_conn_mgr.h>
16 #include <zephyr/net/net_event.h>
17 #include <zephyr/net/net_mgmt.h>
18 #include <zephyr/shell/shell.h>
19 #include <zephyr/sys/printk.h>
20 #include <zephyr/zephyr.h>
21 #include <zephyr/device.h>
22 #include <string.h>
23 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
24
25 char rx_buf_spi[BUF_SIZE];
26 char tx_buf_spi[] = "ATI\r\n";
27 char rx_buf_uart[BUF_SIZE];
28 int rx_buf_uart_pos = 0;
C
Tab Width: 8
Ln 24, Col 1
INS


Activities
Text Editor
Sep 14 04:23
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
20 #include <zephyr/zephyr.h>
21 #include <zephyr/device.h>
22 #include <string.h>
23 LOG_MODULE_REGISTER(sample_gsm_ppp, LOG_LEVEL_DBG);
24
25 char rx_buf_spi[BUF_SIZE];
26 char tx_buf_spi[] = "ATI\r\n";
27 char rx_buf_uart[BUF_SIZE];
28 int rx_buf_uart_pos = 0;
29 int rx_buf_spi_pos = 0;
30
31 const char END_MASSAGE[] = "\r\n";
32 const struct device *spi;
33 const struct device *uart_dev = DEVICE_DT_GET(UART_NODE);
34 static const struct gpio_dt_spec spi_activate_pin =
GPIO_DT_SPEC_GET(spi_act_pin, gpios);
35 static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
36 static const struct device *gsm_dev;
37 static struct net_mgmt_event_callback mgmt_cb;
38 static bool starting = IS_ENABLED(CONFIG_GSM_PPP_AUTOSTART);
39
40 /* queue to store up to 10 messages (aligned to 4-byte boundary) */
41 K_MSGQ_DEFINE(uart_msgq, 32, 10, 4);
42 K_MSGQ_DEFINE(spi_msgq, 32, 10, 4);
43
44 struct spi_cs_control spi_cs = {
45 .gpio = GPIO_DT_SPEC_GET(SPI_CS, gpios),
46 .delay = 0u
C
Tab Width: 8
Ln 24, Col 1
INS



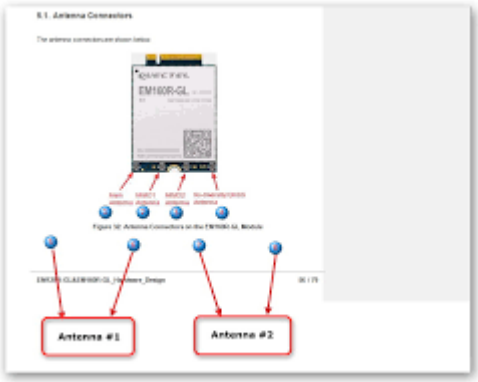
If you got error, open terminal, write “west build –t menuconfig”, go to “sub additional“ option and check random generator option.

Then we wrote a while true which started gsm_ppp and then stopped gsm_ppp by gsm_start and gsm_stop functions.

Here a problem occurred which caused code got fatal error. The reason was that when ARM MCU boots, gsm_init is called and gsm is started. By calling gsm_start again in our written code, gsm_start is called while gsm is already started and cause fatal error.

So we wrote a gsm_stop before the while true to stop initialized gsm and then we wrote gsm_start and gsm_stop in while true.

Then we connect antenna to “main” port of Quectel modem:



To make sure that Modem is registered in local network, we went to gsm_ppp.c, “registering: “ part, added gsm struct to watch in VScode and saw attribute “gsm_net_status”. If “gsm_net_status” is “HOME_NETWORK”, Modem has registered in local network. Another way to make sure that Modem is registered in local network is blink of green led of Quectel Modem module. If green led is on at 80% of time, then Modem has registered in local network, else it will be on at 20% of time.

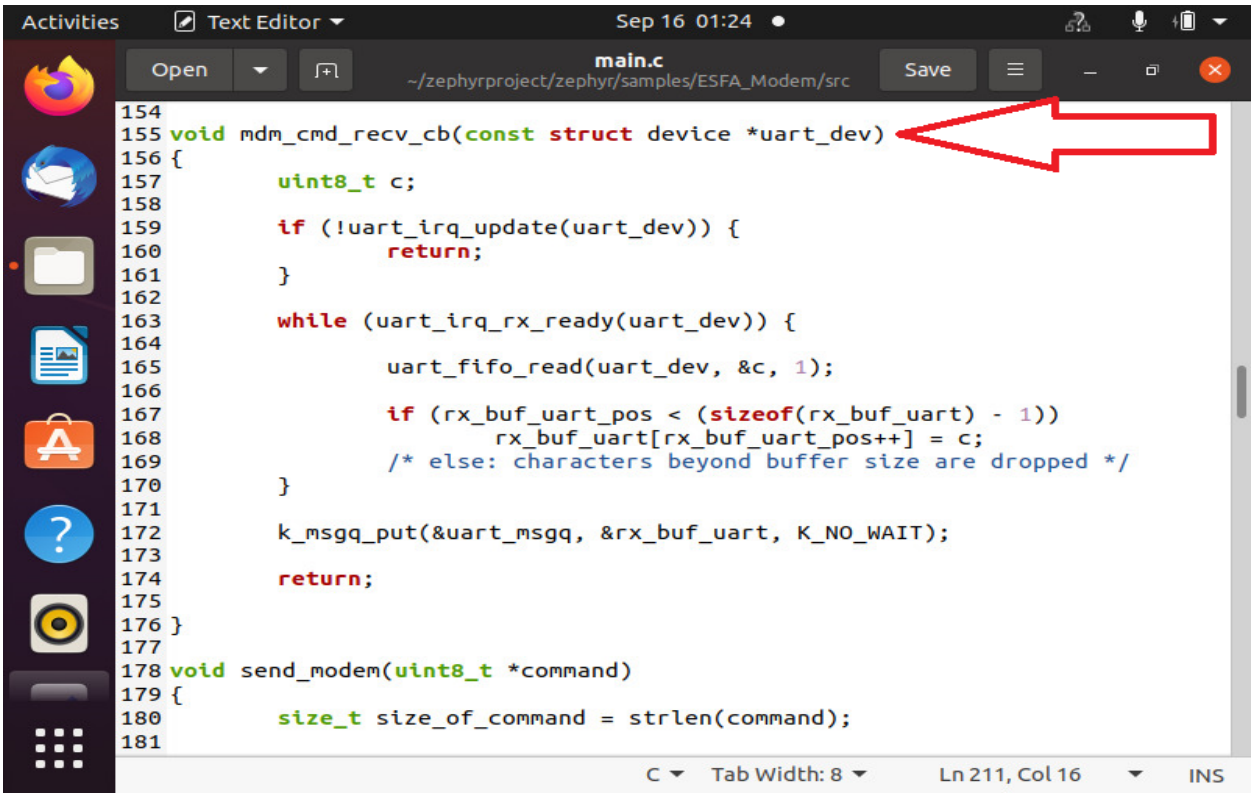
5. UART Communication Implementation between Modem and ARM MCU

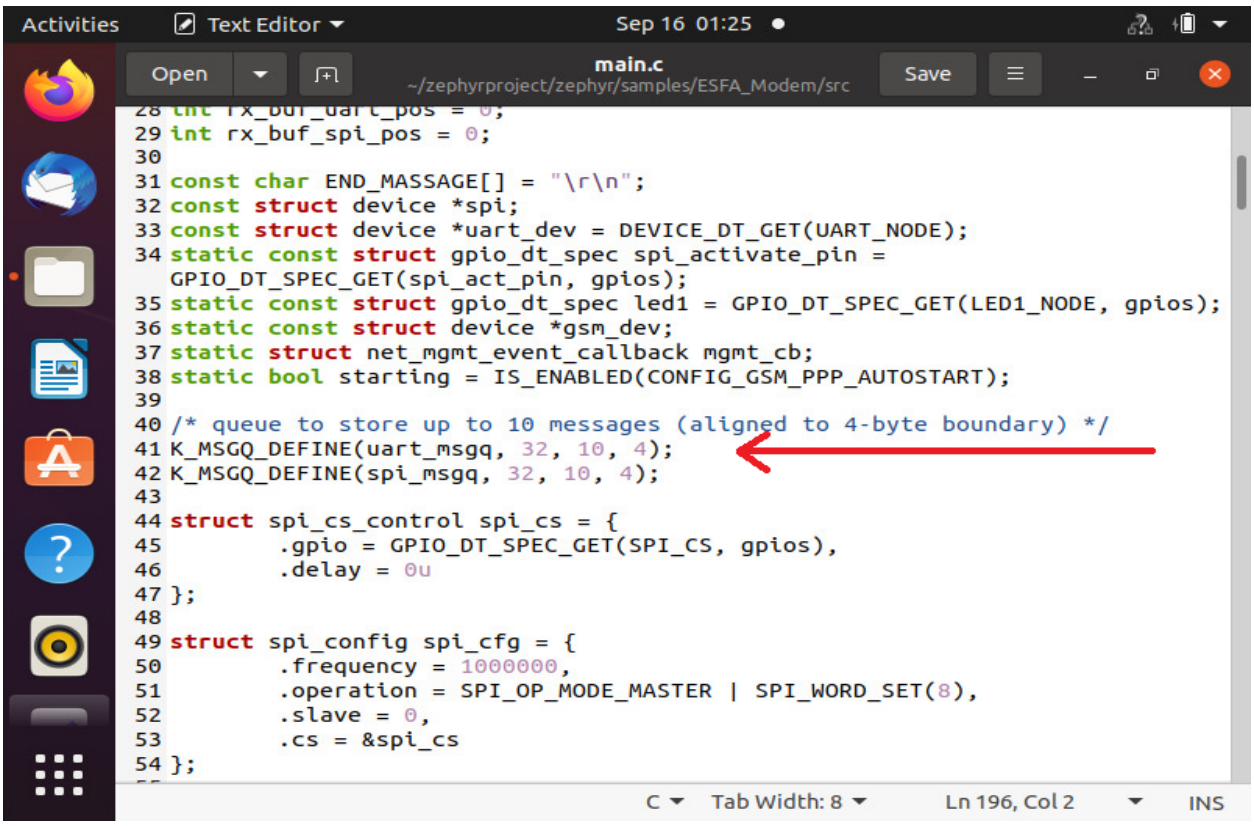
UART declaration in device tree

At first we declared Uart2 in device tree and get its device from device tree in src/main.c which is told on pages 22 till 25.

Callback function for Modem's Response

Usually ARM MCU sends AT command to Modem and waits for Modem's response. So we need to enable irq(interrupt request) for uart2 and define a callback function that whenever interrupt occurs, this function be called. So we write a uart_init and mdm_cmd_recv_cb function like below images which reads from UART till end of massage and put read massage to UART massage queue:

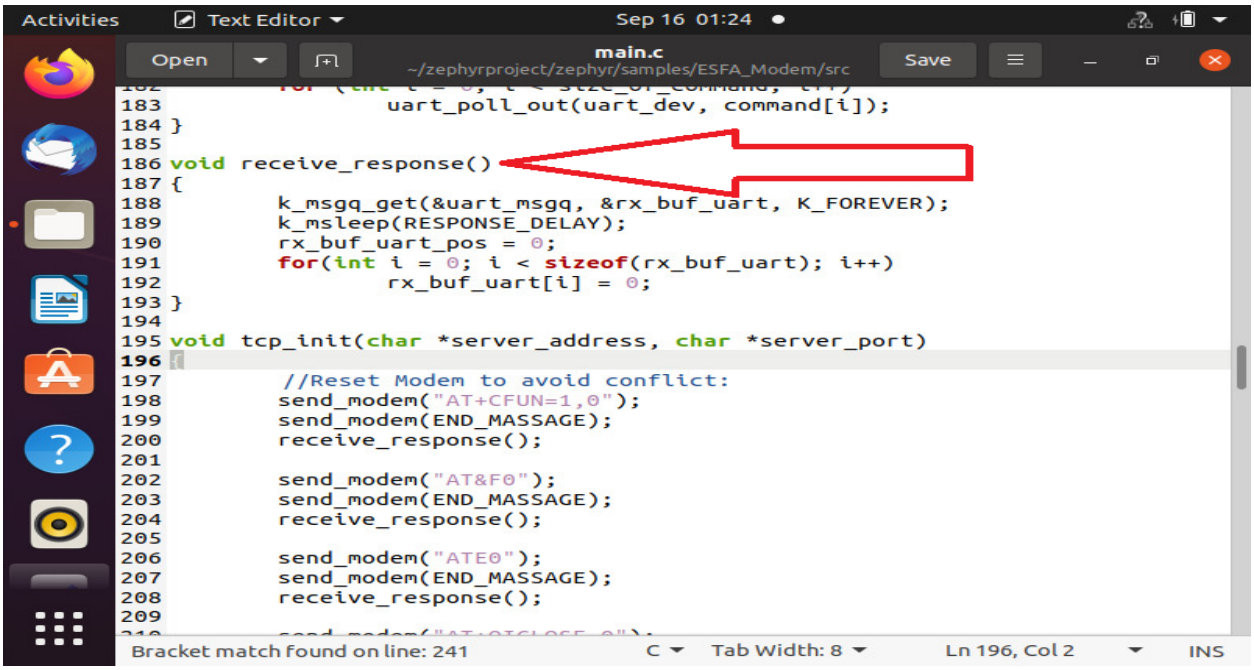


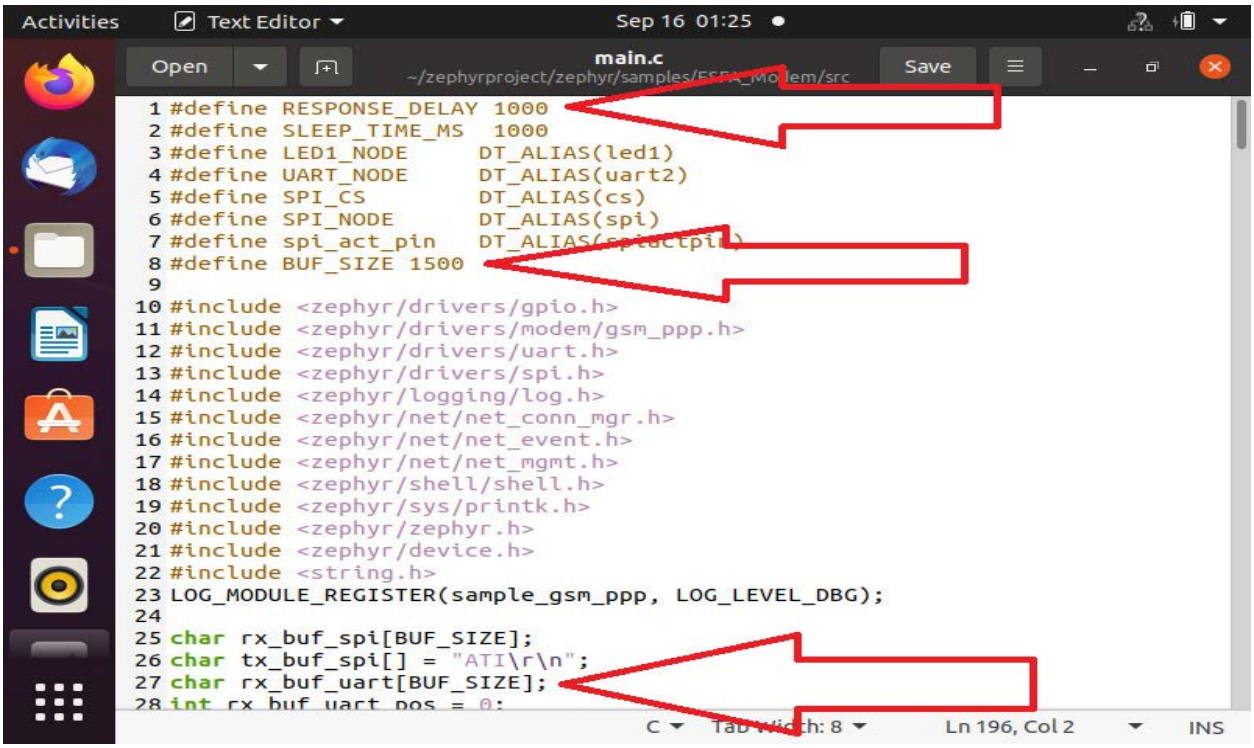


Be careful callback function is called for each receive character.

Receiving Modem's response from UART

For receiving Modem's response we just get massage from UART massage queue then we empty the UART rx buffer:

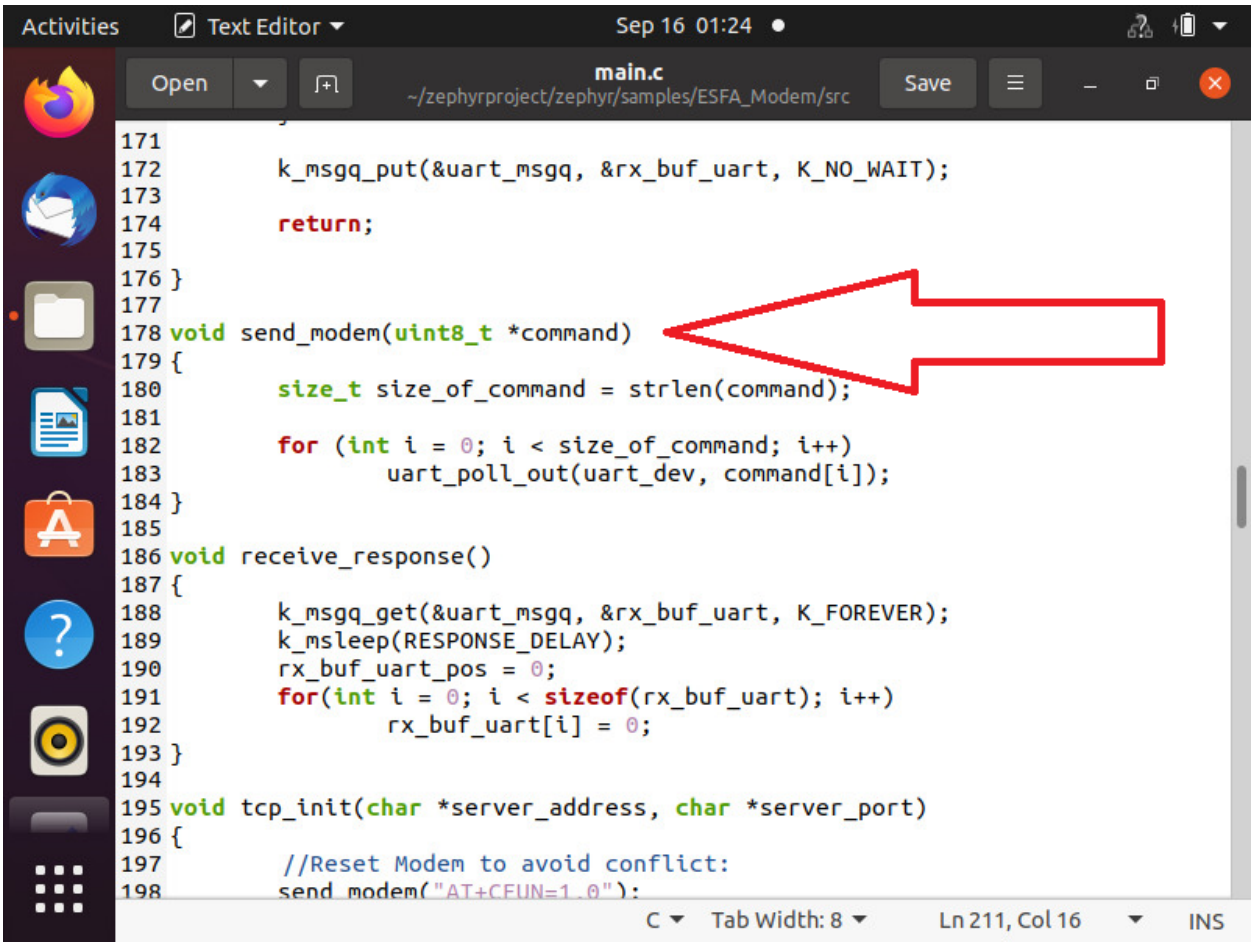




Be careful we must give some delay to receive all Modem's response at UART.

Sending AT command to Modem through uart

For sending AT command to Modem we just write each character of the command to UART by uart_poll_out function:



Side Items Learned

1. For checking if Modem is registered in local network, you can send AT command “AT+CREG?” to Modem and see its response.

2. There is a command “lsdev” which lists available devices.

3. Beside how Modem works and its UART communication there are another things like configuring Modem’s settings, settting the fire wall (for example if received packet is UDP, it will be rejected), port forwarding (for example there is a root Modem that forwards received packet to one of Modems that are linked with root Modem), transparency (Modem just send whatever receives) and etc.

6. TCP Communication Implementation

In general to establish a TCP connection, send and receive a massage through a TCP connection, we send their relative AT commands to the Modem and get Modem's response.

TCP connection initialization

To establish a TCP connection do the followings: (**see Quectel Modem's manual**)

1. Reset functionality level of Modem by "AT+CFUN=1,0"

2. Reset Modem's configurations by "AT&F0"

3. Disable echo in Modem's response by "ATE0"

4. Close any opened TCP connections by "AT+QICLOSE=1"

5. Deactivate the TCP connection by "AT+QIDEACT=1"

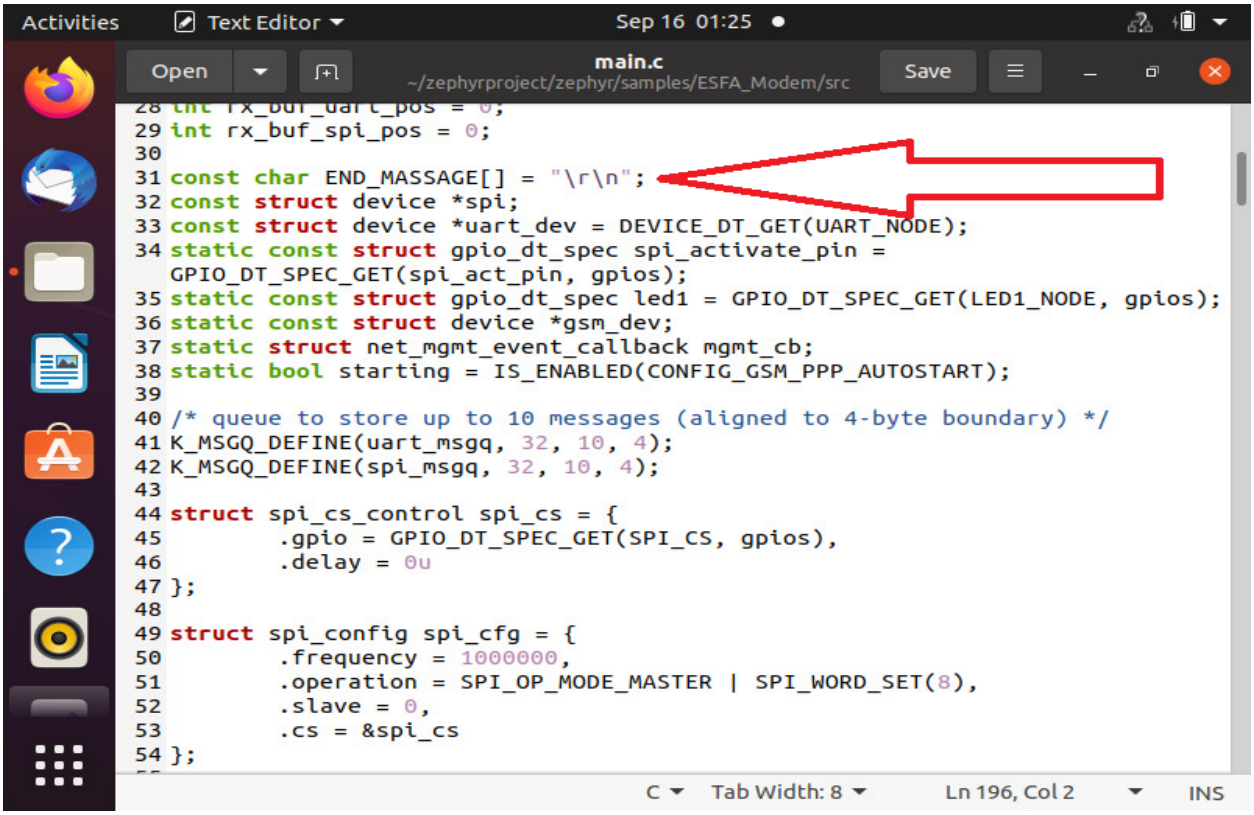
6. Set the APN for your TCP connection by "AT+QICSGP=1,1 … "

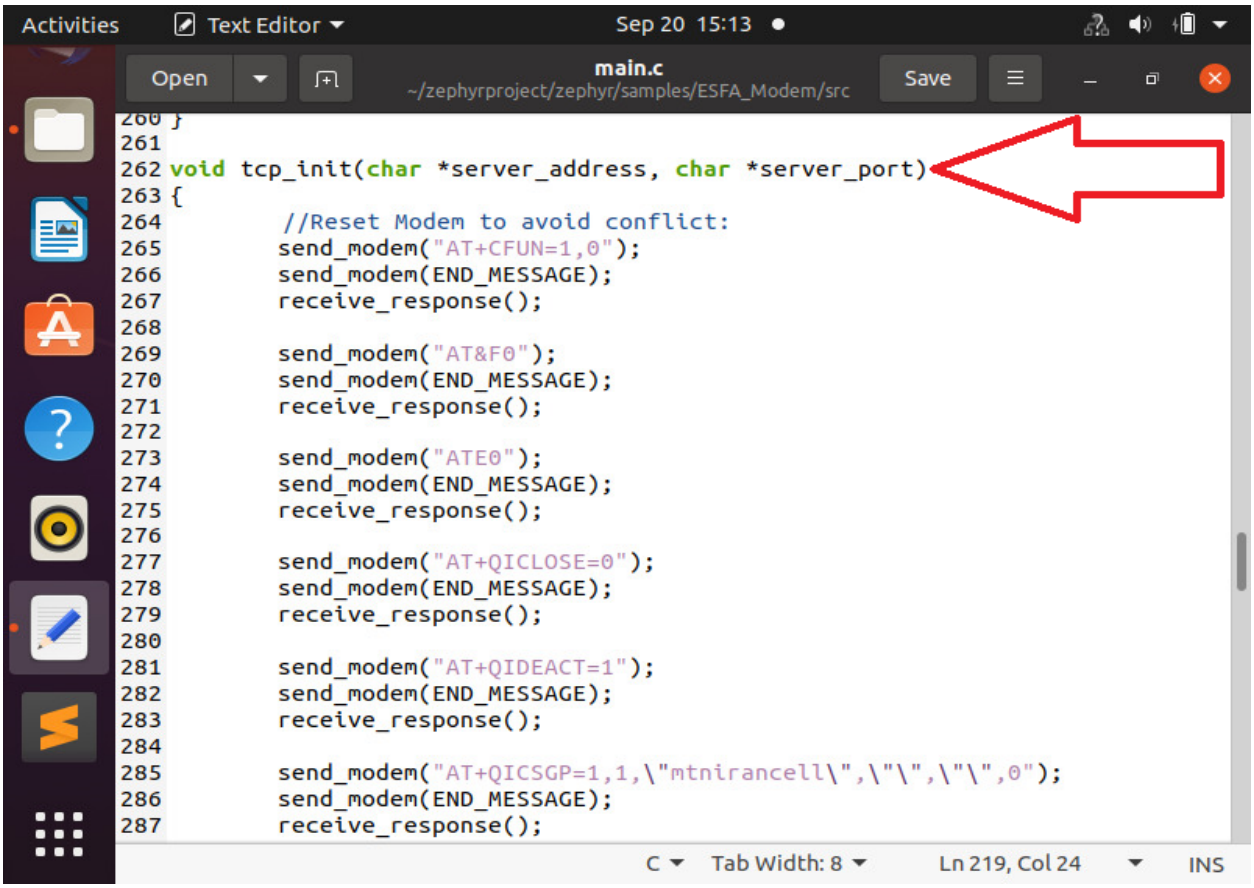
7. Activate the TCP connection by "AT+QIACT=1

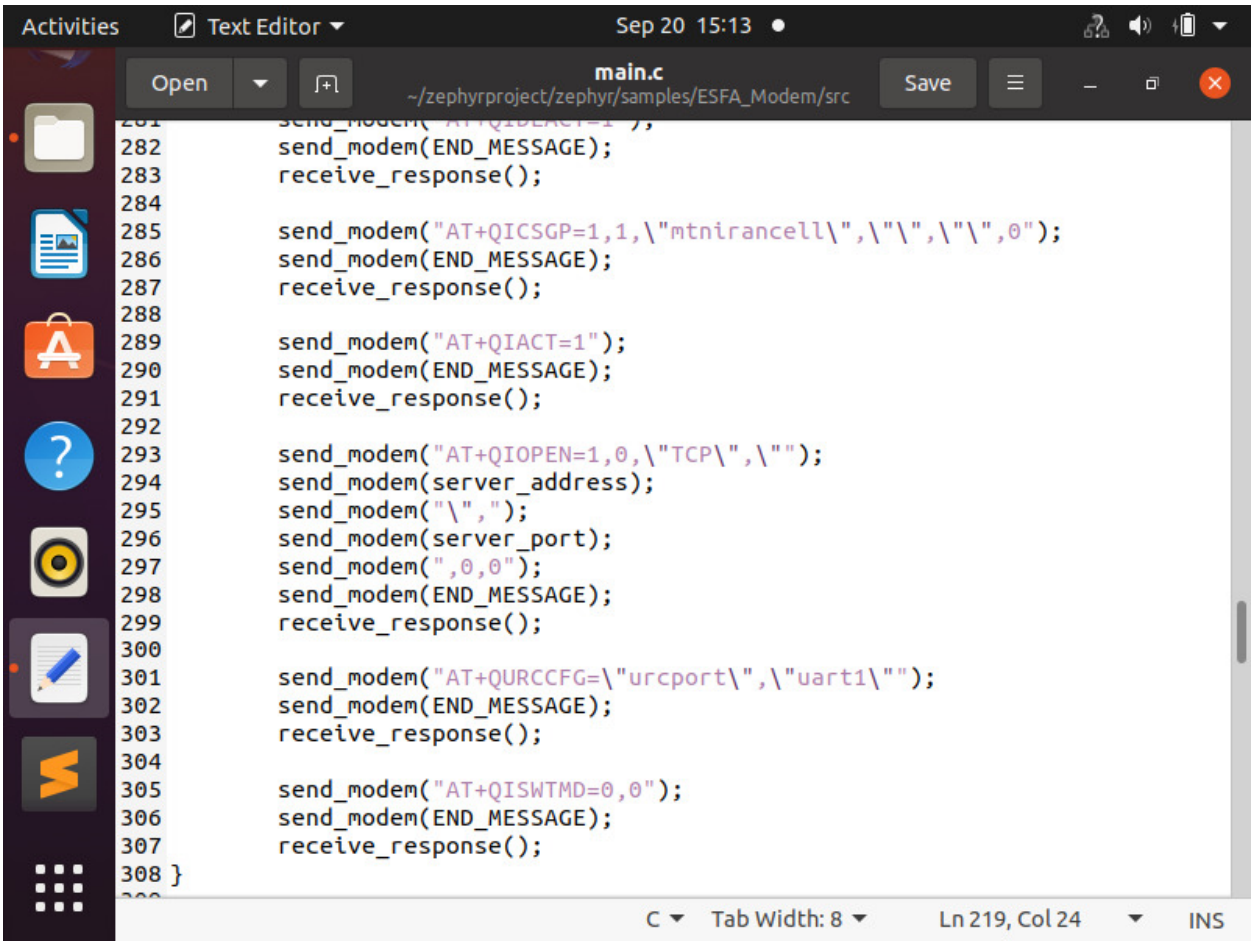
8. Open a new TCP connection by "AT+QIOPEN=1,0, …"

9. The Modem reports "QURC" whenever it receives a message, so set urcport to UART to see "QURC" on the UART connection between MCU and the Modem and know that Modem has received a message.

10. Switch Modem's mode to buffer access mode by "AT+QISWTMD=0,0"

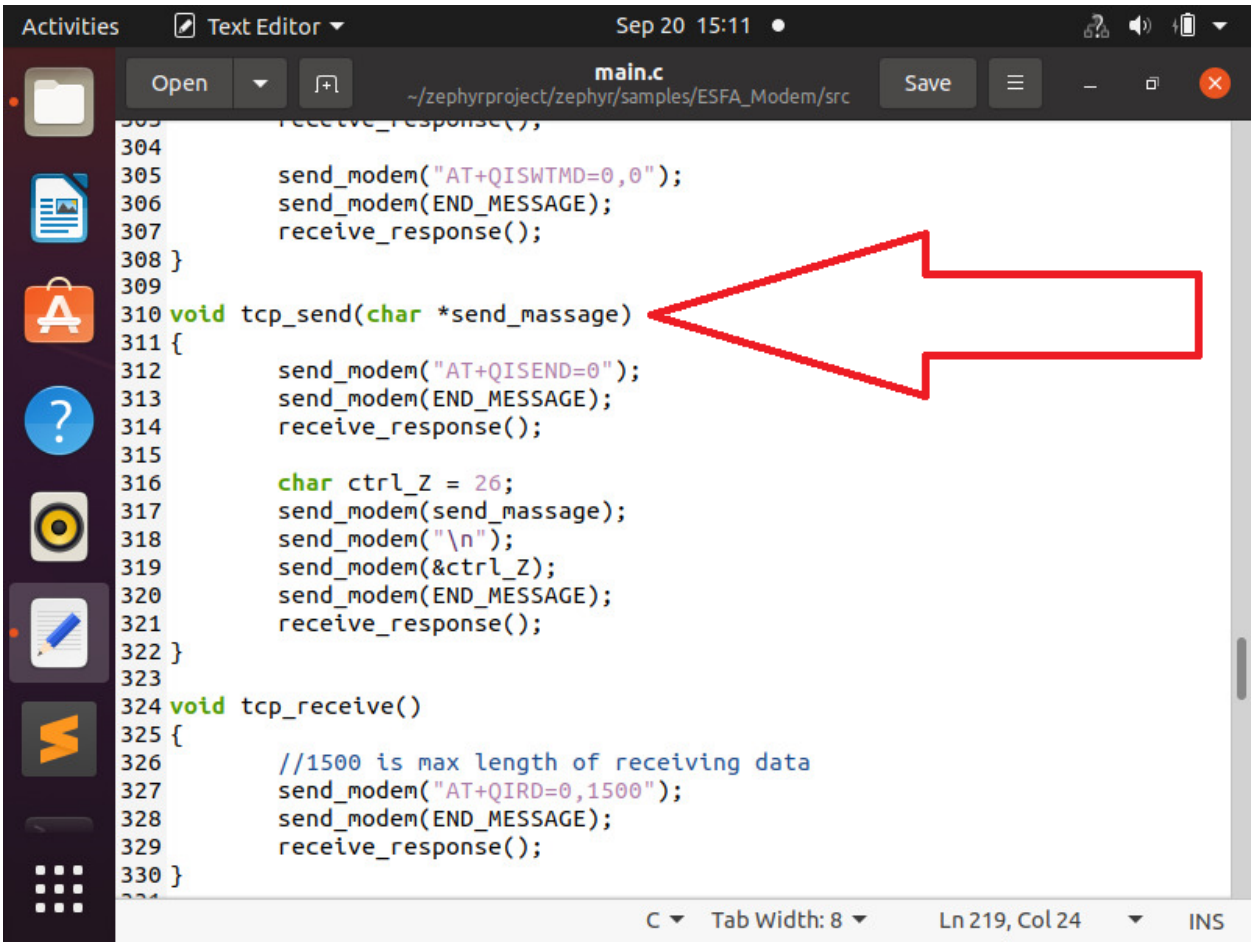

Activities
Text Editor
Sep 16 01:25
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
int rx_buf_uart_pos = 0;
int rx_buf_spi_pos = 0;
const char END_MASSAGE[] = "\r\n";
const struct device *spi;
const struct device *uart_dev = DEVICE_DT_GET(UART_NODE);
static const struct gpio_dt_spec spi_activate_pin =
GPIO_DT_SPEC_GET(spi_act_pin, gpios);
static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
static const struct device *gsm_dev;
static struct net_mgmt_event_callback mgmt_cb;
static bool starting = IS_ENABLED(CONFIG_GSM_PPP_AUTOSTART);
/* queue to store up to 10 messages (aligned to 4-byte boundary) */
K_MSGQ_DEFINE(uart_msgq, 32, 10, 4);
K_MSGQ_DEFINE(spi_msgq, 32, 10, 4);
struct spi_cs_control spi_cs = {
.gpio = GPIO_DT_SPEC_GET(SPI_CS, gpios),
.delay = 0u
};
struct spi_config spi_cfg = {
.frequency = 1000000,
.operation = SPI_OP_MODE_MASTER | SPI_WORD_SET(8),
.slave = 0,
.cs = &spi_cs
};
C
Tab Width: 8
Ln 196, Col 2
INS


Activities
Text Editor
Sep 20 15:13
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
}
void tcp_init(char *server_address, char *server_port)
{
//Reset Modem to avoid conflict:
send_modem("AT+CFUN=1,0");
send_modem(END_MESSAGE);
receive_response();
send_modem("AT&F0");
send_modem(END_MESSAGE);
receive_response();
send_modem("ATE0");
send_modem(END_MESSAGE);
receive_response();
send_modem("AT+QICLOSE=0");
send_modem(END_MESSAGE);
receive_response();
send_modem("AT+QIDEACT=1");
send_modem(END_MESSAGE);
receive_response();
send_modem("AT+QICSGP=1,1,\"mtnirancell\",\"\",\"\",0");
send_modem(END_MESSAGE);
receive_response();
C
Tab Width: 8
Ln 219, Col 24
INS


Activities
Text Editor
Sep 20 15:13
Open
main.c
~/zephyrproject/zephyr/samples/ESFA_Modem/src
Save
282 send_modem(END_MESSAGE);
283 receive_response();
284
285 send_modem("AT+QICSGP=1,1,\"mtnirancell\",\"\",\"\",0");
286 send_modem(END_MESSAGE);
287 receive_response();
288
289 send_modem("AT+QIACT=1");
290 send_modem(END_MESSAGE);
291 receive_response();
292
293 send_modem("AT+QIOPEN=1,0,\"TCP\",\"");
294 send_modem(server_address);
295 send_modem("\",");
296 send_modem(server_port);
297 send_modem(",0,0");
298 send_modem(END_MESSAGE);
299 receive_response();
300
301 send_modem("AT+QURCCFG=\"urcport\",\"uart1\"");
302 send_modem(END_MESSAGE);
303 receive_response();
304
305 send_modem("AT+QISWTMD=0,0");
306 send_modem(END_MESSAGE);
307 receive_response();
308 }
C
Tab Width: 8
Ln 219, Col 24
INS

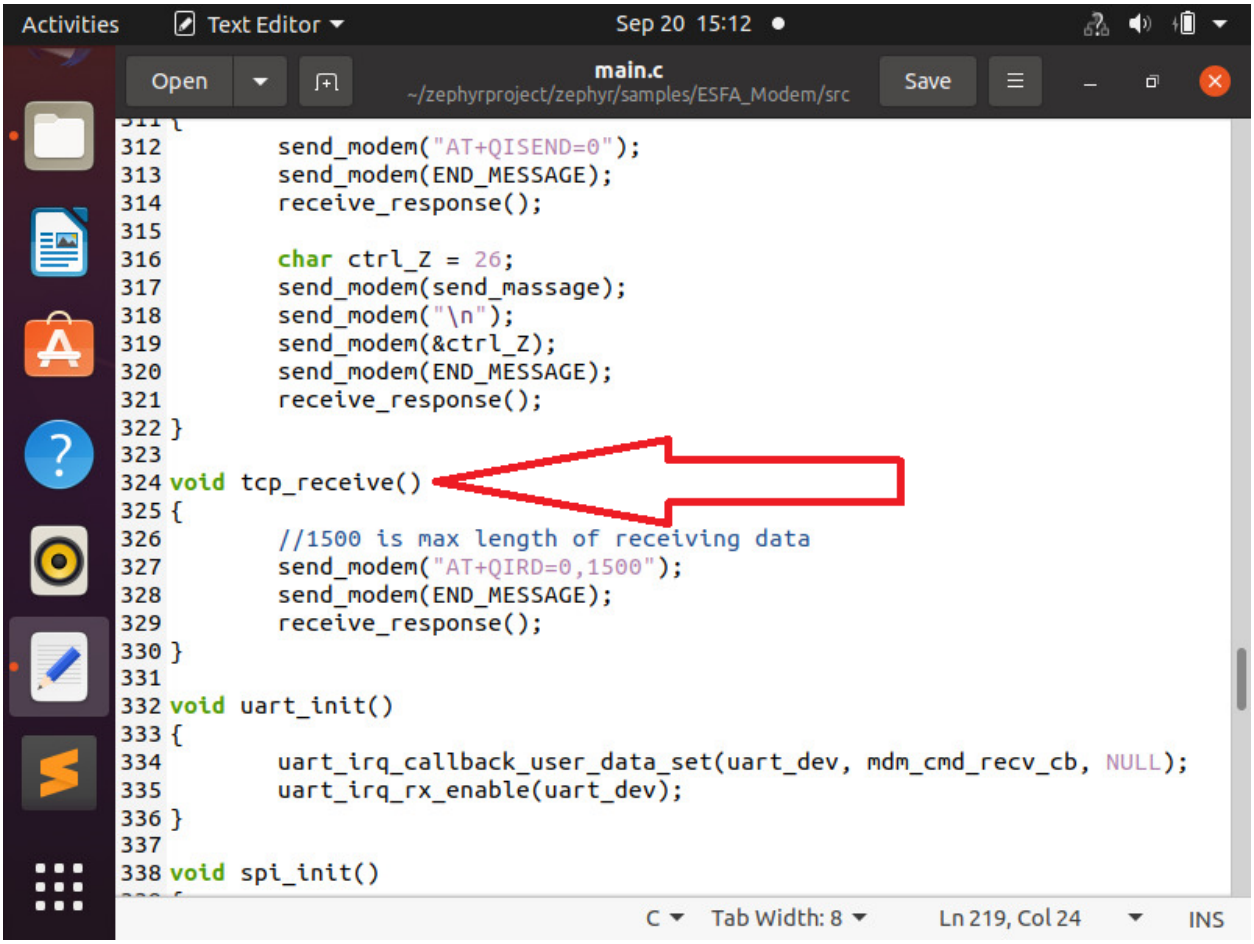
Send a massage through the TCP connection

To send a message through the TCP connection, at first send its relative command “AT+QISEND=0” where 0 is connect id (**see Quectel modem’s manual for more information**), then you should get “>” in Modem’s response which means “write your message”. Then send your message. Finally send ctrl_z (its asci is 26) to determine the end of your message. **Be careful you should get “SEND OK” in Modem’s response to confirm that message is sent successfully**.



Receive a massage from the TCP connection

To receive a message from a TCP connection, send its relative command “AT+QIRD=0,1500”, where 0 is connect id and 1500 is the maximum length of receiving data, to Modem and get its response.



Be careful if you get “SEND OK” from the Modem but nothing is received by “QIRD”, there is a problem with the server you are requesting. In our case the problem was that the timeout of “tcpbin.com” “port: 4242” was too low and the TCP connection was closing fast, so we used “google.com” “port: 80” which has no time out to test the TCP connection.

Side Items Learned

1. The Modem has 3 modes:

   1. Buffer access mode: In this mode, data is sent by “AT+QISEND” and whenever the Modem receives data from TCP server, it stores the received data in a buffer and with “AT+QIRD” the received data is sent to MCU.

   2. Direct push mode: In this mode, data is sent by “AT+QISEND” and whenever the Modem receives data, it sends the received data directly to MCU through the UART communication.

   3. Transparent access mode: In this mode, the Modem sends whatever it receives from MCU (even AT commands)to TCP server and sends whatever it receives from TCP server to MCU.

2. DNS stands for Domain Name Set, which is a long table that assigns each URL to its corresponding IP address.