

TCP Communication Implementation via zephyr OS

by Quectel Modem

Contents

1. Introduction to Project.....	1
2. Zephyr OS Installation and startup.....	2
3. Connecting J-Link Debugger to Modem Board.....	2
4. Register Modem in Local Network via gsm.....	3
5. UART Communication Implementation between Modem and ARM MCU	4
6. TCP Communication Implementation.....	6

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

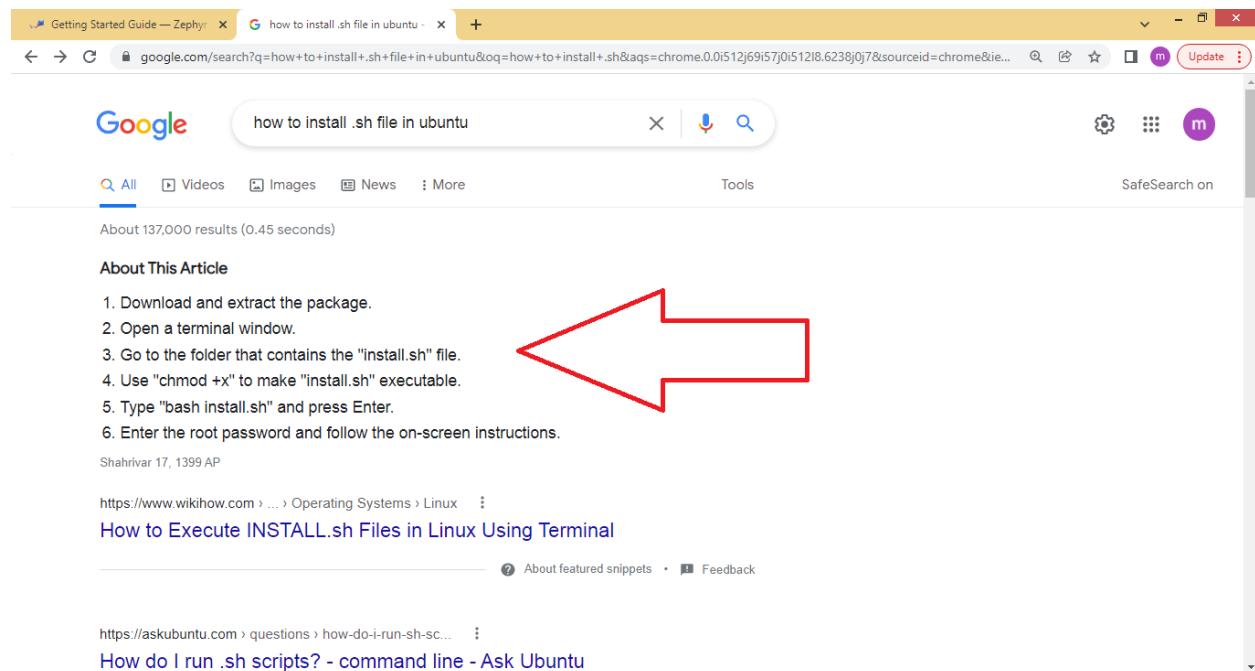
Installation

At first we installed Zephyr OS from below link:

[“https://docs.zephyrproject.org/latest/develop/getting_started/index.html”](https://docs.zephyrproject.org/latest/develop/getting_started/index.html)

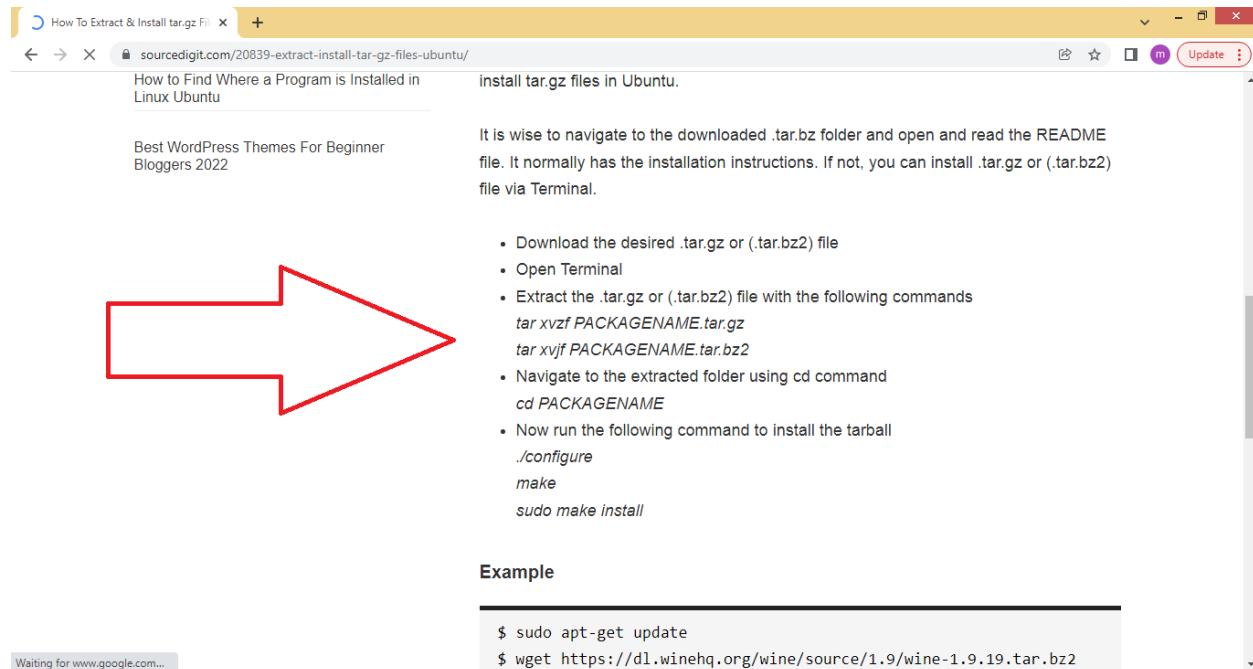
For installation Zephyr OS on Linux just follow the steps told in above link.

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 09:02 ●
Open stm32f4_disco.dts ~zephyrproject/zephyr/boards/arm/stm32f407vgt6 Save
- □ ×
19     zephyr,flash = &flash0;
20     zephyr,ccm = &ccm0;
21     zephyr,canbus = &can2;
22 };
23
24 leds {
25     compatible = "gpio-leds";
26     green_led_1: led_1 {
27         gpios = <&gpiod 9 GPIO_ACTIVE_HIGH>;
28         label = "User LD1";
29     };
30     red_led_2: led_2 {
31         gpios = <&gpiod 10 GPIO_ACTIVE_HIGH>;
32         label = "User LD2";
33     };
34 };
35
36 spi_chip_select {
37     compatible = "gpio-keys";
38     spi_cs: cs {
39         /*PB9 as CS pin*/
40         gpios = <&gpiob 9 GPIO_ACTIVE_LOW>;
41         label = "SPI Chip Select";
42     };
43 };
44
45 spi_activate_pin {
46     compatible = "gpio-keys";
47 }
```



```
Activities Text Editor Sep 11 09:16 ●
Open stm32f4_disco.dts ~zephyrproject/zephyr/boards/arm/stm32f407vgt6 Save
- □ ×
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 };
78
79 &spi2 {
80     pinctrl-names = "default";
81 }
```

Activities Text Editor Sep 11 09:22

Open main.c ~ /zephyrproject/zephyr/samples/ESFA_Modem/src

main.c

stm32f4_disco.dts

```
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

main.c

stm32f4_disco.dts

```
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_MESSAGE[] = "\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
47 }
```

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

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
312     tcp_init("postman-echo.com", "80");
313
314     //tcp_init("tcpbin.com", "4242");
315
316     //tcp_init("smtp.gmail.com", "465");
317
318     tcp_send("test1");
319     tcp_receive();
320
321     //tcp_init("postman-echo.com", "80");
322
323     //tcp_init("tcpbin.com", "4242");
324
325     //tcp_init("smtp.gmail.com", "465");
326
327     tcp_send("test1");
328     tcp_receive();
329
330     //tcp_init("postman-echo.com", "80");
331
332     //tcp_init("tcpbin.com", "4242");
333
334     //tcp_init("smtp.gmail.com", "465");
335
336     tcp_send("test1");
337     tcp_receive();
338
339     //tcp_init("postman-echo.com", "80");
340
341     //tcp_init("tcpbin.com", "4242");
342
343     //tcp_init("smtp.gmail.com", "465");
344
345     tcp_send("test1");
346     tcp_receive();
347
348     //tcp_init("postman-echo.com", "80");
349
350     //tcp_init("tcpbin.com", "4242");
351
352     //tcp_init("smtp.gmail.com", "465");
353
354     tcp_send("test1");
355     tcp_receive();
356
357     //tcp_init("postman-echo.com", "80");
358
359     //tcp_init("tcpbin.com", "4242");
360
361     //tcp_init("smtp.gmail.com", "465");
362
363     mdm_cmd_recv();
364
365     mdm_cmd_send(uart_dev, "AT+QMTOOPEN=0,\"broker.hivemq.com\",-1883\r\n", strlen("AT+QMTOOPEN=0,\"broker.hivemq.com\",1883\r\n"));
366     mdm_cmd_recv();
367
368     mdm_cmd_send(uart_dev, "AT+QMTCONN=0,\"1947\"\r\n", strlen("AT+QMTCONN=0,\"1947\"\r\n"));
369     mdm_cmd_recv();
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
384 }
```

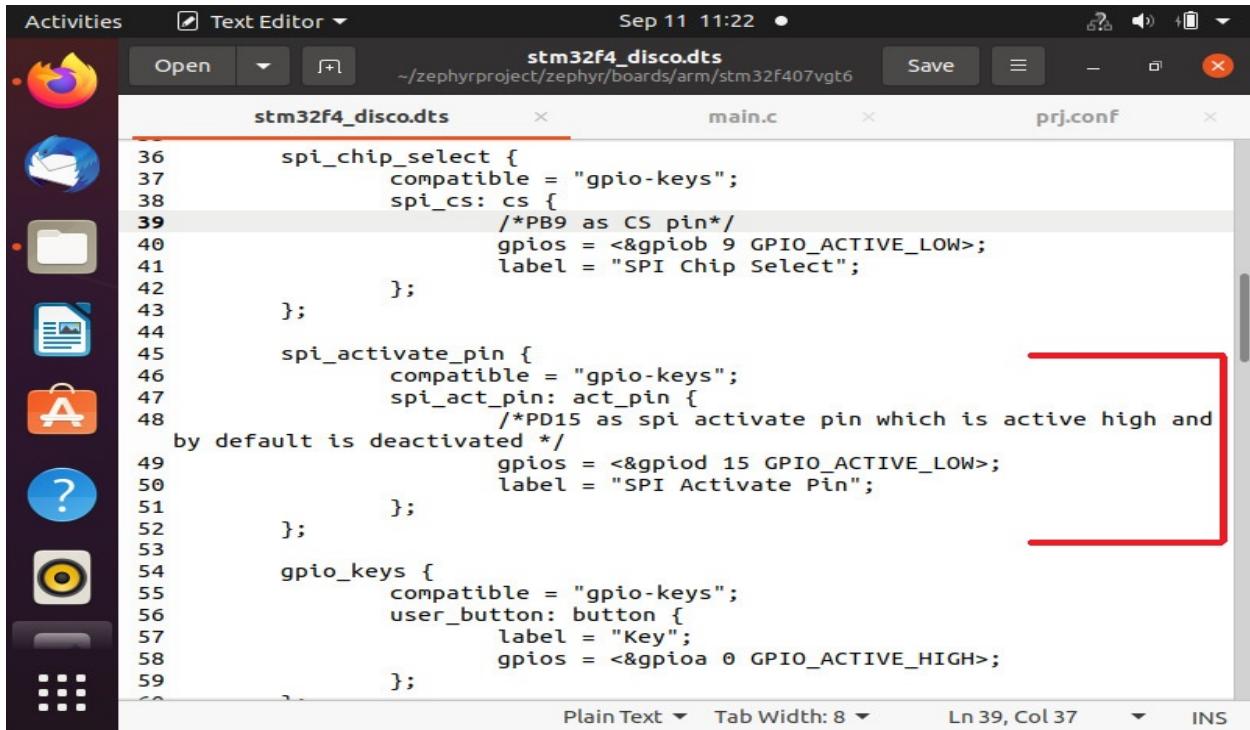
Activities Text Editor Sep 11 09:38

Open main.c ~zephyrproject/zephyr/samples/ESFA_Modem/src

stm32f4_disco.dts main.c

```
362     mdm_cmd_recv();
363
364     mdm_cmd_send(uart_dev, "AT+QMTCFG=\"recv	mode\",0,0,1\r\n", strlen("AT+QMTCFG=\"recv	mode\",0,0,1\r\n"));
365     mdm_cmd_recv();
366
367     mdm_cmd_send(uart_dev, "AT+QMTOOPEN=0,\"broker.hivemq.com\",-1883\r\n", strlen("AT+QMTOOPEN=0,\"broker.hivemq.com\",1883\r\n"));
368     mdm_cmd_recv();
369
370     mdm_cmd_send(uart_dev, "AT+QMTCONN=0,\"1947\"\r\n", strlen("AT+QMTCONN=0,\"1947\"\r\n"));
371     mdm_cmd_recv();
372
373     /* ARG_UNUSED(gsm_dev); */
374     gsm_ppp_stop(gsm_dev);
375
376     while (true) {
377         /* ARG_UNUSED(gsm_dev); */
378         gsm_ppp_start(gsm_dev);
379         ret = gpio_pin_toggle_dt(&led1);
380         k_msleep(SLEEP_TIME_MS);
381
382         /* ARG_UNUSED(gsm_dev); */
383         gsm_ppp_stop(gsm_dev);
384     }
385
386 }
```

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

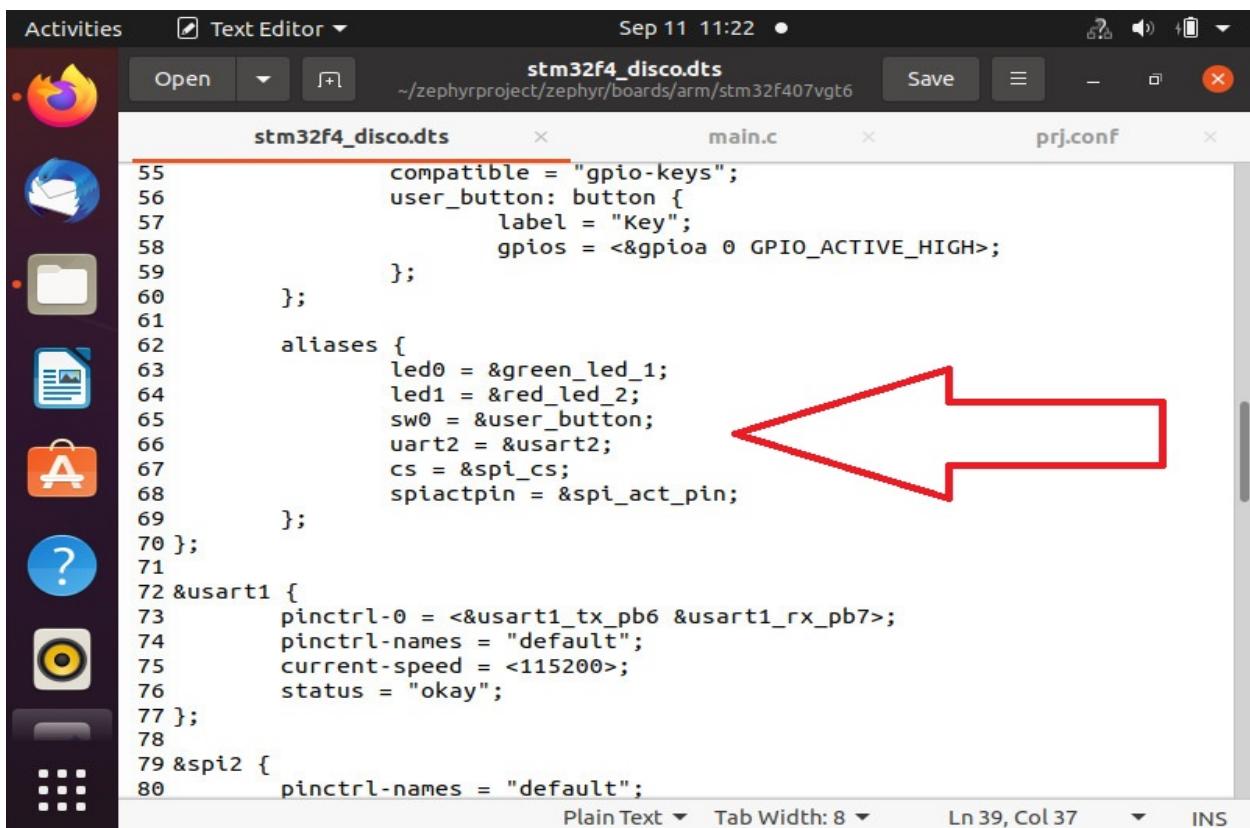


Activities Text Editor Sep 11 11:22

stm32f4_disco.dts main.c prj.conf

```
36     spi_chip_select {
37         compatible = "gpio-keys";
38         spi_cs: cs {
39             /*PB9 as CS pin*/
40             gpios = <&gpiob 9 GPIO_ACTIVE_LOW>;
41             label = "SPI Chip Select";
42         };
43     };
44
45     spi_activate_pin {
46         compatible = "gpio-keys";
47         spi_act_pin: act_pin {
48             /*PD15 as spi activate pin which is active high and
49             by default is deactivated */
50             gpios = <&gpiod 15 GPIO_ACTIVE_LOW>;
51             label = "SPI Activate Pin";
52         };
53     };
54
55     gpio_keys {
56         compatible = "gpio-keys";
57         user_button: button {
58             label = "Key";
59             gpios = <&gpioa 0 GPIO_ACTIVE_HIGH>;
60         };
61     };
62
63     aliases {
64         led0 = &green_led_1;
65         led1 = &red_led_2;
66         sw0 = &user_button;
67         uart2 = &usart2;
68         cs = &spi_cs;
69         spiactpin = &spi_act_pin;
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 };
78
79 &spi2 {
80     pinctrl-names = "default";
```

Plain Text Tab Width: 8 Ln 39, Col 37 INS



Activities Text Editor Sep 11 11:22

stm32f4_disco.dts main.c prj.conf

```
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 };
78
79 &spi2 {
80     pinctrl-names = "default";
```

Plain Text Tab Width: 8 Ln 39, Col 37 INS

Activities Text Editor Sep 11 11:22

Open main.c ~zephyrproject/zephyr/samples/ESFA_Modem/src Save main.c prj.conf

stm32f4_disco.dts x main.c x prj.conf x

```
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];
26 int rx_buf_uart_pos = 0.
```

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 main.c prj.conf

stm32f4_disco.dts x main.c x prj.conf x

```
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_MESSAGE[] = "\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 = {
45     .operation = SPT_OP_MODE_SLAVE | SPT_WORD_SET(8).
```

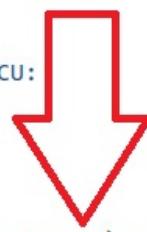
C Tab Width: 8 Ln 82, Col 1 INS

Activities Text Editor Sep 11 11:25

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");
310     tcp_receive();
```



C Tab Width: 8 Ln 82, Col 1 INS

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

The screenshot shows a terminal window with three tabs open: `stm32f4_disco.dts`, `main.c`, and `prj.conf`. Red arrows point from the `stm32f4_disco.dts` and `prj.conf` tabs to the corresponding code snippets below.

```
1 CONFIG_GPIO=y
2 CONFIG_HWINFO=y
3
4 # SPI support
5 CONFIG_SPI=y
6 CONFIG_SPI_ASYNC=y
7 CONFIG_SPI_SLAVE=y
8 CONFIG_SPI_STM32=y
9 CONFIG_SPI_STM32_INTERRUPT=y
10
11 # UART support
12 CONFIG_SERIAL=y
13 CONFIG_UART_INTERRUPT_DRIVEN=y
14
15 # GSM modem support
16 CONFIG_MODEM=y
17 CONFIG_MODEM_GSM_PPP=y
18
19 # PPP networking support
20 CONFIG_NET_DRIVERS=y
21 CONFIG_NET_PPP=y
22 CONFIG_NET_L2_PPP=y
23 CONFIG_NET_NATIVE=y
24 CONFIG_NETWORKING=y
25
26 CONFIG_NET_L2_PPP_TIMEOUT=10000
```

The screenshot shows a terminal window with three tabs open: `stm32f4_disco.dts`, `main.c`, and `prj.conf`. A red arrow points from the `stm32f4_disco.dts` tab to the code snippet below.

```
95     clocks = <&clk_lise>;
96     status = "okay";
97 };
98
99 &rcc {
100     clocks = <&pll>;
101     clock-frequency = <DT_FREQ_M(168)>;
102     ahb-prescaler = <1>;
103     apb1-prescaler = <4>;
104     apb2-prescaler = <2>;
105 };
106
107 &uart2 {
108     pinctrl-0 = <&uart2_tx_pd5 &uart2_rx_pd6>;
109     pinctrl-names = "default";
110     current-speed = <115200>;
111     status = "okay";
112     gsm: gsm-modem {
113         compatible = "zephyr,gsm-ppp";
114     };
115 };
116
117 &timers2 {
118     status = "okay";
119
120     pwm2: pwm {
```

Activities Text Editor Sep 11 09:43

Open stm32f4_disco.dts ~/zephyrproject/zephyr/boards/arm/stm32f407vgt6 Save main.c prj.conf

```
52     },
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 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;
```

C Tab Width: 8 Ln 223, Col 33 INS

Activities Text Editor Sep 11 09:44

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

Save prj.conf

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_MESSAGE[] = "\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
47 };
```

C Tab Width: 8 Ln 223, Col 33 INS

Activities Text Editor Sep 11 09:47

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

Save prj.conf

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_MESSAGE);
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 }
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.

Programming

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:

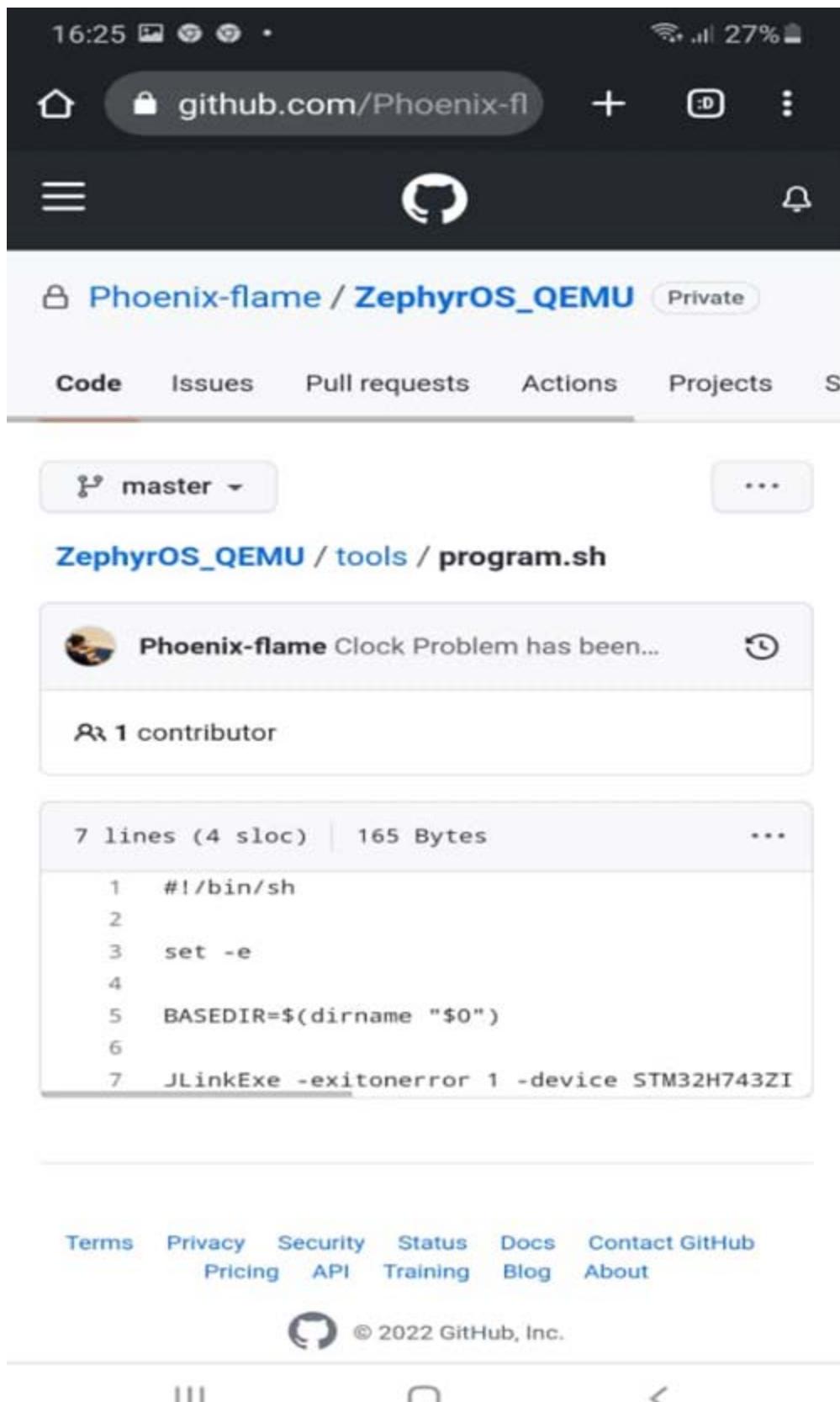
The screenshot shows a GitHub mobile interface. At the top, the URL 'github.com/Phoenix-flame' is visible. Below it, the repository 'Phoenix-flame / ZephyrOS_QEMU' is shown as private. The 'Code' tab is selected. Under the code editor, the file path 'ZephyrOS_QEMU / tools / program.jlink' is displayed. The content of the file is a shell script:

```
1 r
2 loadbin build/zephyr/zephyr.bin, 0x08000000
3 verifybin build/zephyr/zephyr.bin, 0x08000000
4 rnh
5 exit
```

At the bottom of the screen, there are navigation icons for back, forward, and search.

GitHub footer links include: Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, About.

© 2022 GitHub, Inc.



16:26 27%

github.com/Phoenix-flame

☰ GitHub 🔒

Phoenix-flame / ZephyrOS_QEMU Private

Code Issues Pull requests Actions Projects Settings

master

ZephyrOS_QEMU / tools / program.sh

Phoenix-flame Clock Problem has been...

1 contributor

7 lines (4 sloc) | 165 Bytes

```
rice 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 Settings

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.

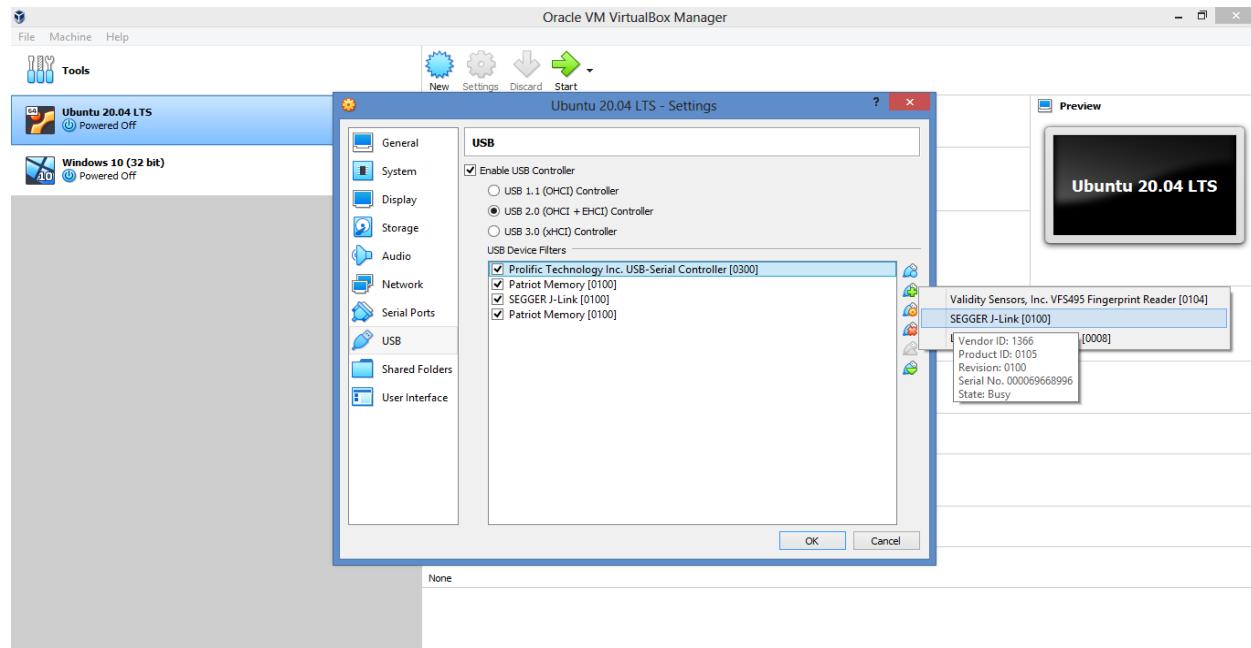
3. Connecting J-Link Debugger to Modem Board

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

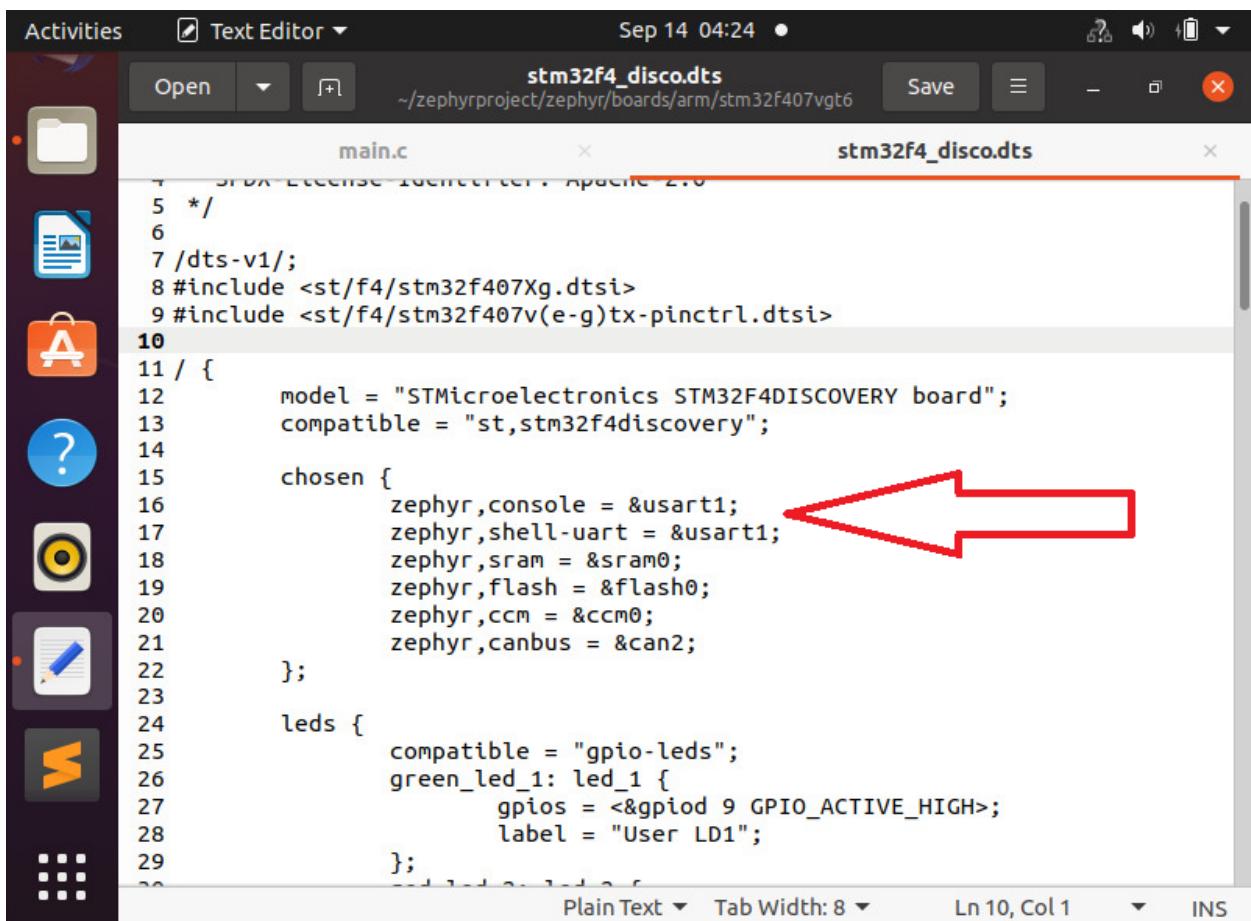
- 1.** There is a SWO viewer software which shows transferring data on serial ports.
- 2.** For programming with STM32CubeMX do the followings:
 1. Select the board which you want to program
 2. Pin configuration: declare output and input pins
 3. Project manager: define the properties of your project 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:



The screenshot shows a terminal window titled "Text Editor" with the file "stm32f4_disco.dts" open. The code is a Device Tree Source (DTS) file. A red arrow points to the "chosen" block, specifically to the line "zephyr,console = &usart1;".

```
SPDX-License-Identifier: Apache-2.0
main.c          stm32f4_disco.dts
...
5 */
6
7 /dts-v1;
8 #include <st/f4/stm32f407Xg.dtsi>
9 #include <st/f4/stm32f407v(e-g)tx-pinctrl.dtsi>
10
11 / {
12     model = "STMicroelectronics STM32F4DISCOVERY board";
13     compatible = "st,stm32f4discovery";
14
15     chosen {
16         zephyr,console = &usart1;
17         zephyr,shell-uart = &usart1;
18         zephyr,sram = &sram0;
19         zephyr,flash = &flash0;
20         zephyr,ccm = &ccm0;
21         zephyr,canbus = &can2;
22     };
23
24     leds {
25         compatible = "gpio-leds";
26         green_led_1: led_1 {
27             gpios = <&gpiod 9 GPIO_ACTIVE_HIGH>;
28             label = "User LD1";
29         };
30     };
31 }
```

Activities Text Editor Sep 14 04:24

Open 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";
125         pinctrl-0 = <&tim2_ch1_pa0>;

```

Plain Text Tab Width: 8 Ln 47, Col 31 INS

Activities Text Editor Sep 14 04:24

Open 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 };
79
```

Plain Text Tab Width: 8 Ln 47, Col 31 INS

Activities Text Editor Sep 14 04:22

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

```
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;
```

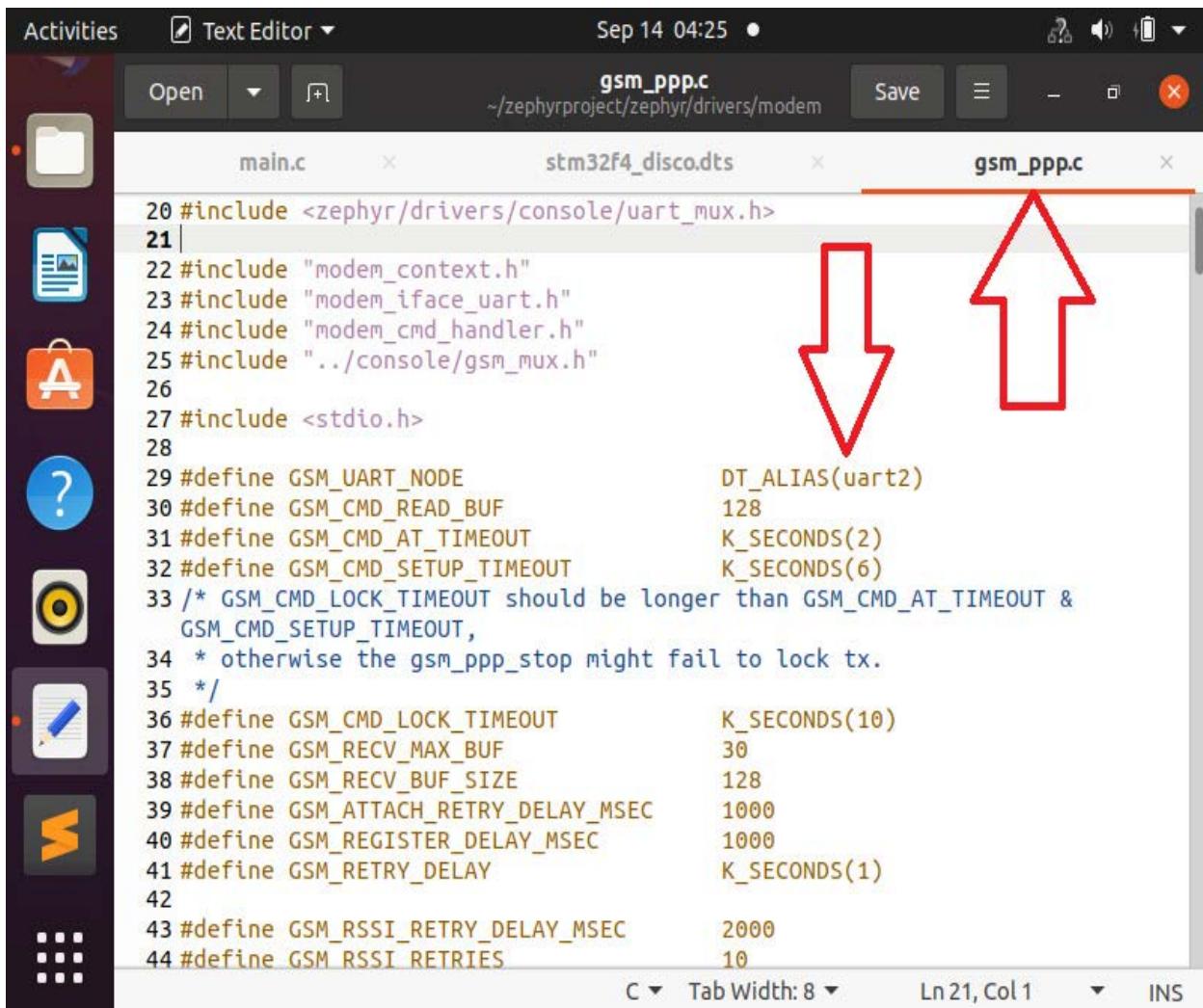
C Tab Width: 8 Ln 24, Col 1 INS

Activities Text Editor Sep 14 04:23

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

```
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_MESSAGE[] = "\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



```
Activities Text Editor Sep 14 04:25
Open ~/zephyrproject/zephyr/drivers/modem gsm_ppp.c Save
main.c stm32f4_disco.dts gsm_ppp.c
20 #include <zephyr/drivers/console/uart_mux.h>
21 |
22 #include "modem_context.h"
23 #include "modem_iface_uart.h"
24 #include "modem_cmd_handler.h"
25 #include "../console/gsm_mux.h"
26
27 #include <stdio.h>
28
29 #define GSM_UART_NODE DT_ALIAS(uart2)
30 #define GSM_CMD_READ_BUF 128
31 #define GSM_CMD_AT_TIMEOUT K_SECONDS(2)
32 #define GSM_CMD_SETUP_TIMEOUT K_SECONDS(6)
33 /* GSM_CMD_LOCK_TIMEOUT should be longer than GSM_CMD_AT_TIMEOUT &
   GSM_CMD_SETUP_TIMEOUT,
   * otherwise the gsm_ppp_stop might fail to lock tx.
34 */
35
36 #define GSM_CMD_LOCK_TIMEOUT K_SECONDS(10)
37 #define GSM_RECV_MAX_BUF 30
38 #define GSM_RECV_BUF_SIZE 128
39 #define GSM_ATTACH_RETRY_DELAY_MSEC 1000
40 #define GSM_REGISTER_DELAY_MSEC 1000
41 #define GSM_RETRY_DELAY K_SECONDS(1)
42
43 #define GSM_RSSI_RETRY_DELAY_MSEC 2000
44 #define GSM_RSST_RETRIES 10
```

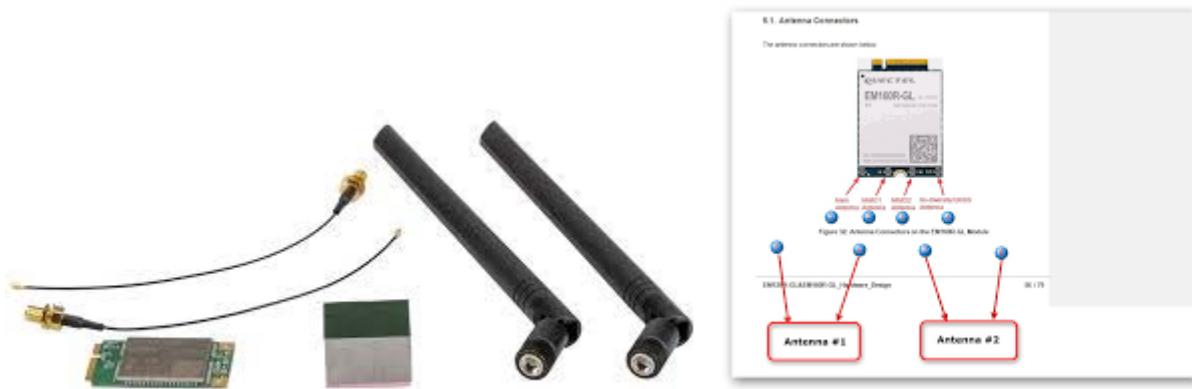
If you got error, open terminal, write “west build –t menuconfig”, go to “sub additional” option and check random generator option.

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.

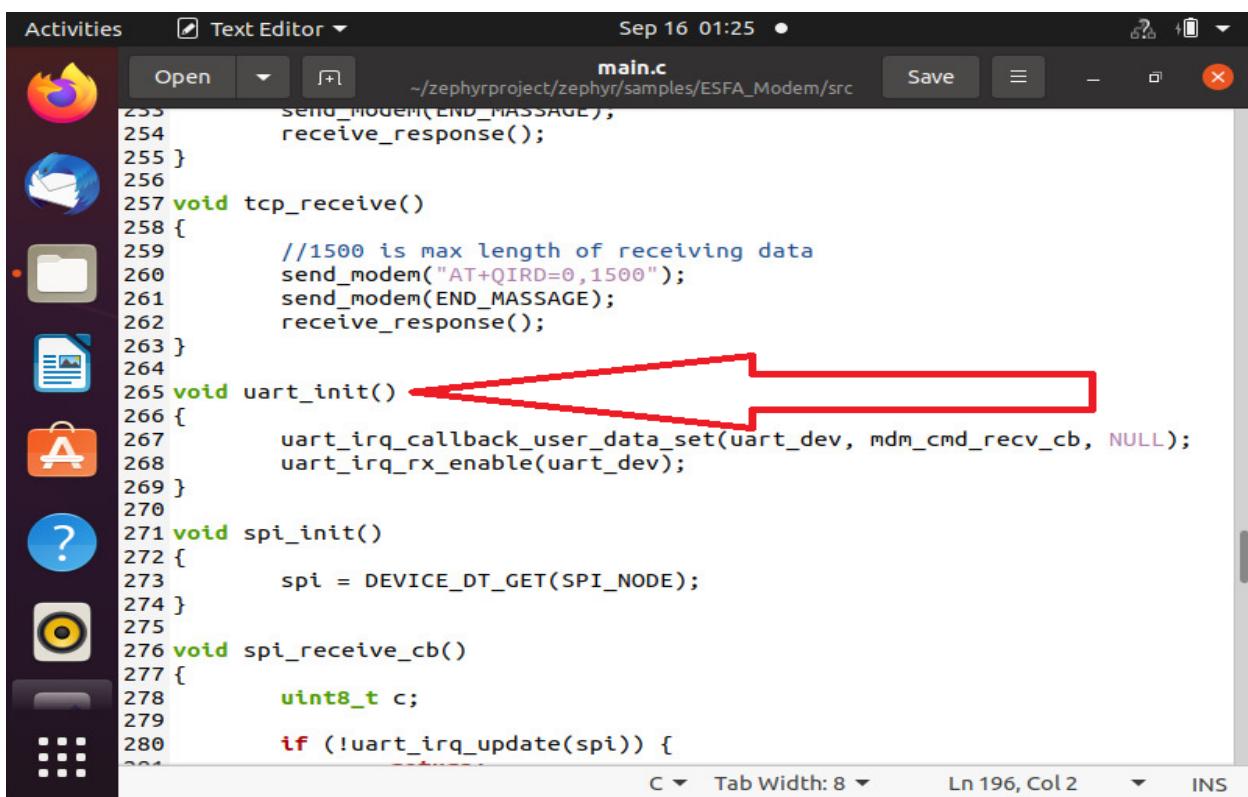
5. UART Communication Implementation between Modem and ARM MCU

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.

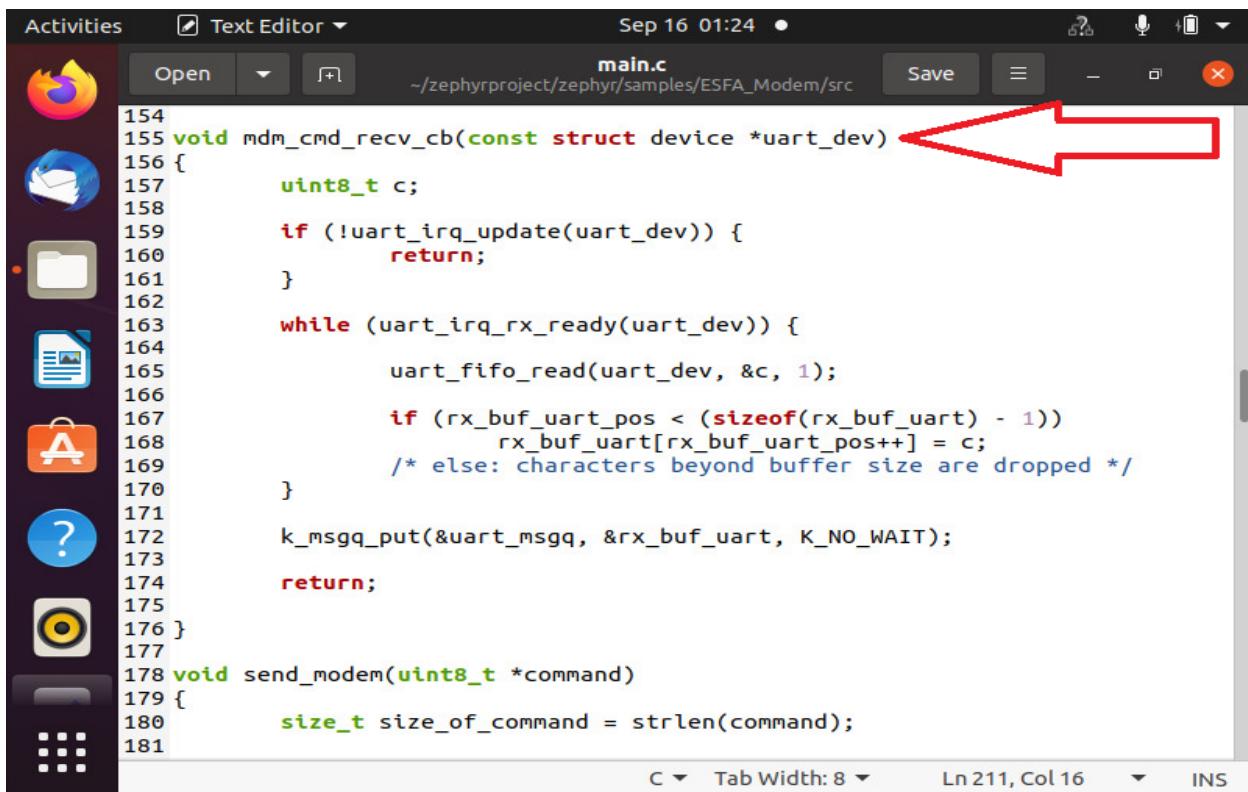
Callback function for Modem's Response

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 message and put read message to UART message queue:



```
Activities Text Editor Sep 16 01:25
main.c ~zephyrproject/zephyr/samples/ESFA_Modem/src
Open Save
253     send_modem(END_MESSAGE),
254     receive_response();
255 }
256
257 void tcp_receive()
258 {
259     //1500 is max length of receiving data
260     send_modem("AT+QIRD=0,1500");
261     send_modem(END_MESSAGE);
262     receive_response();
263 }
264
265 void uart_init() ->
266 {
267     uart_irq_callback_user_data_set(uart_dev, mdm_cmd_recv_cb, NULL);
268     uart_irq_rx_enable(uart_dev);
269 }
270
271 void spi_init()
272 {
273     spi = DEVICE_DT_GET(SPI_NODE);
274 }
275
276 void spi_receive_cb()
277 {
278     uint8_t c;
279
280     if (!uart_irq_update(spi)) {
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301 }
```

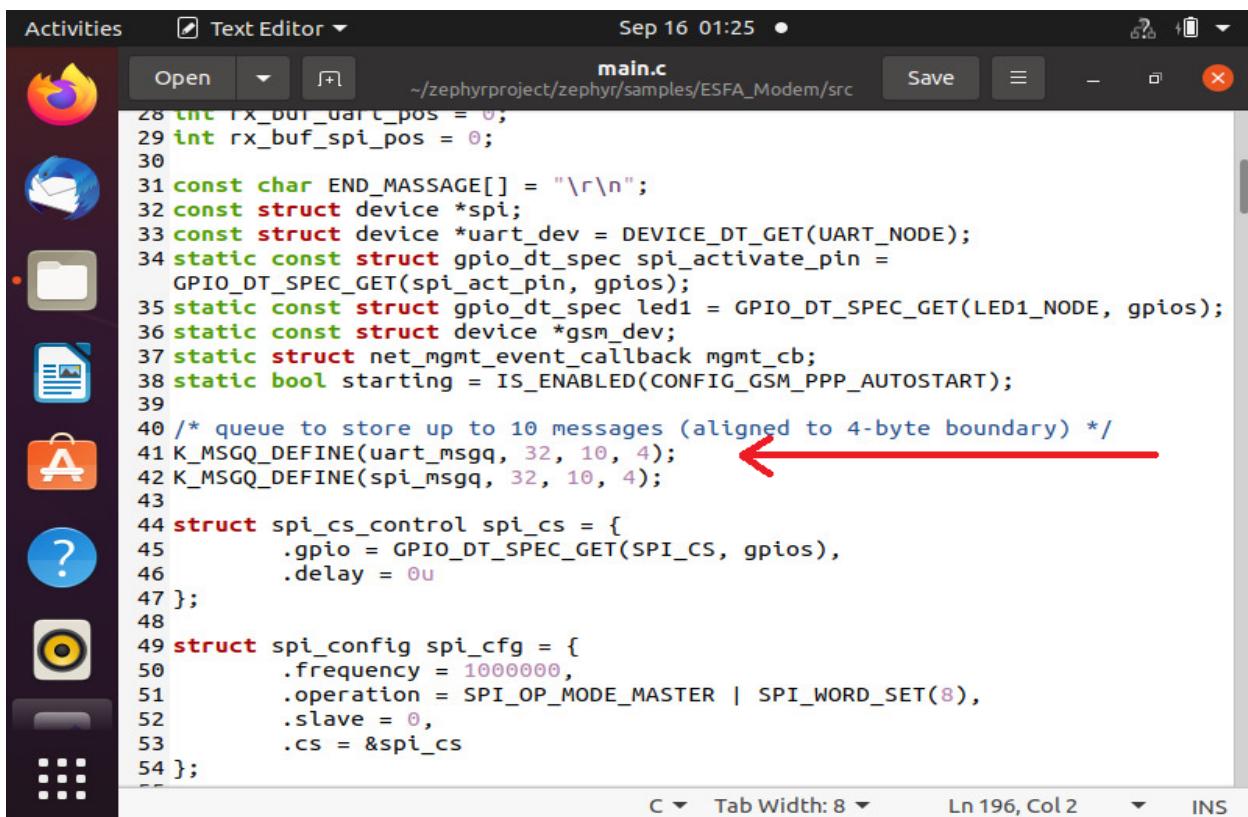
Activities Text Editor Sep 16 01:24



```
main.c
154 void mdm_cmd_recv_cb(const struct device *uart_dev)
155 {
156     uint8_t c;
157
158     if (!uart_irq_update(uart_dev)) {
159         return;
160     }
161
162     while (uart_irq_rx_ready(uart_dev)) {
163
164         uart_fifo_read(uart_dev, &c, 1);
165
166         if (rx_buf_uart_pos < (sizeof(rx_buf_uart) - 1))
167             rx_buf_uart[rx_buf_uart_pos++] = c;
168         /* else: characters beyond buffer size are dropped */
169     }
170
171     k_msgq_put(&uart_msgq, &rx_buf_uart, K_NO_WAIT);
172
173     return;
174 }
175
176 void send_modem(uint8_t *command)
177 {
178     size_t size_of_command = strlen(command);
179 }
```

C Tab Width: 8 Ln 211, Col 16 INS

Activities Text Editor Sep 16 01:25



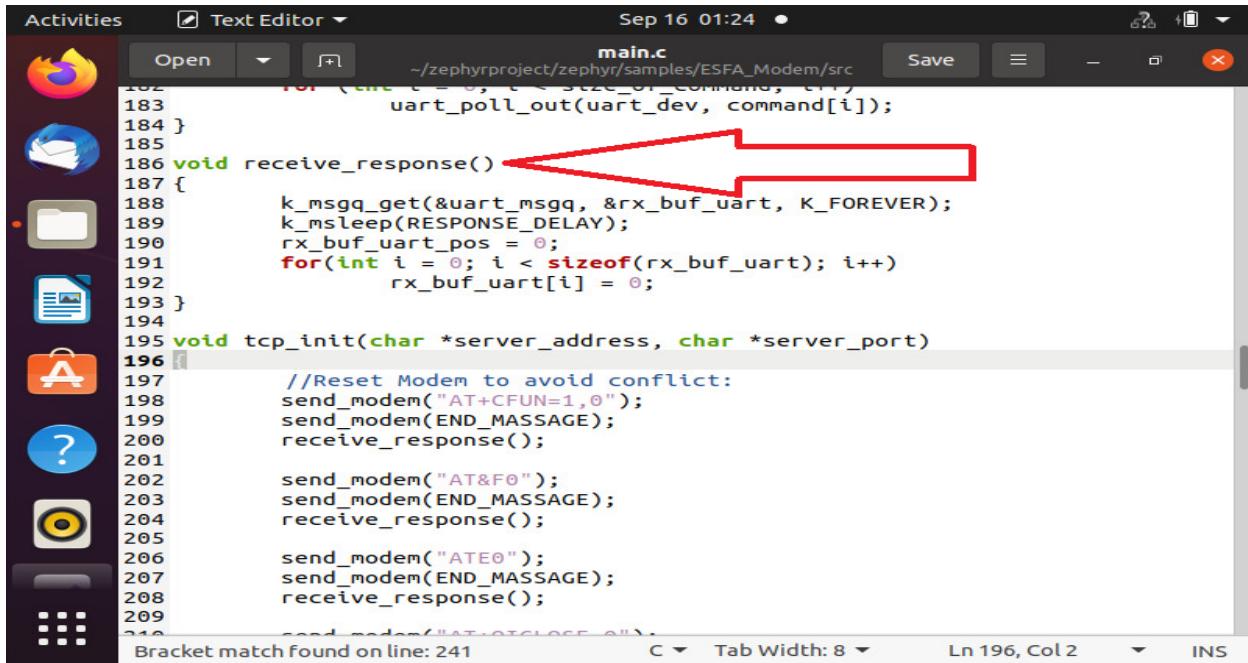
```
main.c
28 int tx_buf_uart_pos = 0;
29 int rx_buf_spi_pos = 0;
30
31 const char END_MESSAGE[] = "\r\n";
32 const struct device *spi;
33 const struct device *uart_dev = DEVICE_DT_GET(UART_NODE);
34 static const struct gpio_dt_spec spi_activate_pin =
    GPIO_DT_SPEC_GET(spi_act_pin, gpios);
35 static 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
47 };
48
49 struct spi_config spi_cfg = {
50     .frequency = 1000000,
51     .operation = SPI_OP_MODE_MASTER | SPI_WORD_SET(8),
52     .slave = 0,
53     .cs = &spi_cs
54 };
```

C Tab Width: 8 Ln 196, Col 2 INS

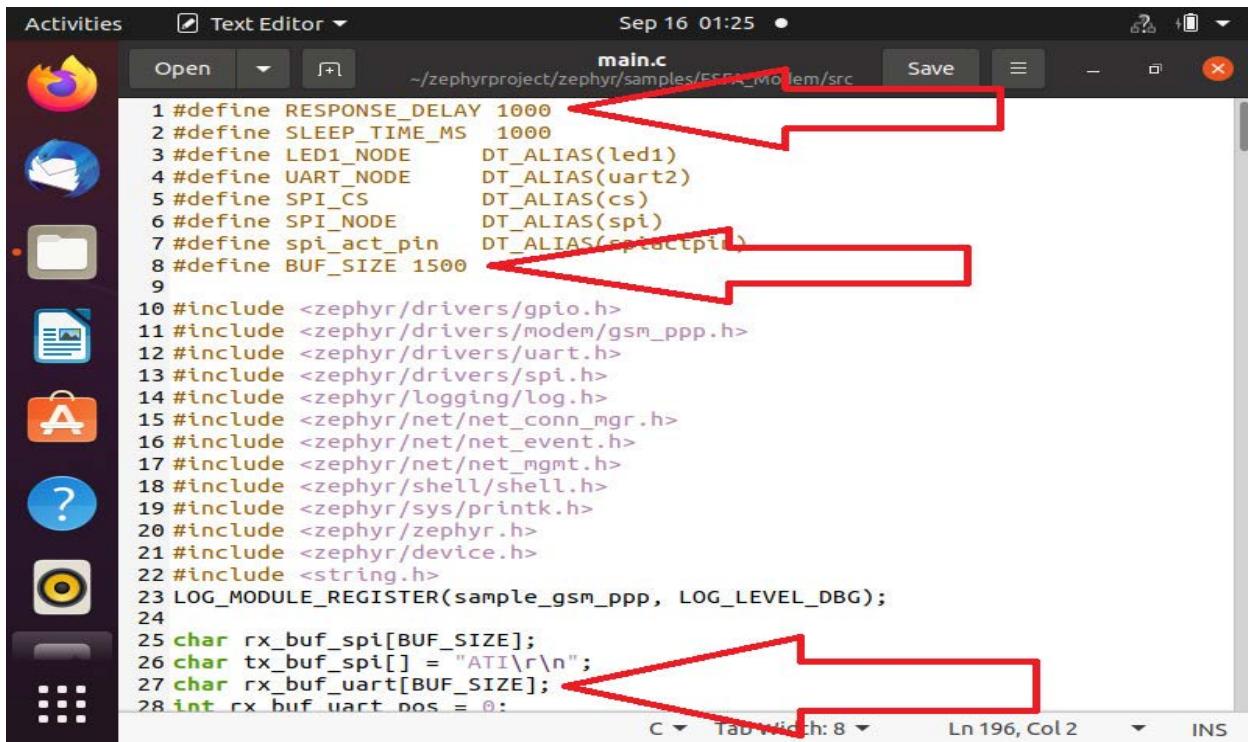
Be careful callback function is called for each receive character.

Receiving Modem's response from UART

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



```
Activities Text Editor Sep 16 01:24 main.c
Open + ~zephyrproject/zephyr/samples/ESFA_Modem/src Save
182     for (int i = 0; i < sizeof(command); i++)
183         uart_poll_out(uart_dev, command[i]);
184 }
185
186 void receive_response()
187 {
188     k_msgq_get(&uart_msgq, &rx_buf_uart, K_FOREVER);
189     k_msleep(RESPONSE_DELAY);
190     rx_buf_uart_pos = 0;
191     for(int i = 0; i < sizeof(rx_buf_uart); i++)
192         rx_buf_uart[i] = 0;
193 }
194
195 void tcp_init(char *server_address, char *server_port)
196 {
197     //Reset Modem to avoid conflict:
198     send_modem("AT+CFUN=1,0");
199     send_modem(END_MESSAGE);
200     receive_response();
201
202     send_modem("AT&F0");
203     send_modem(END_MESSAGE);
204     receive_response();
205
206     send_modem("ATE0");
207     send_modem(END_MESSAGE);
208     receive_response();
209
210     send_modem("AT+ATCCLK=0");
211 }
```

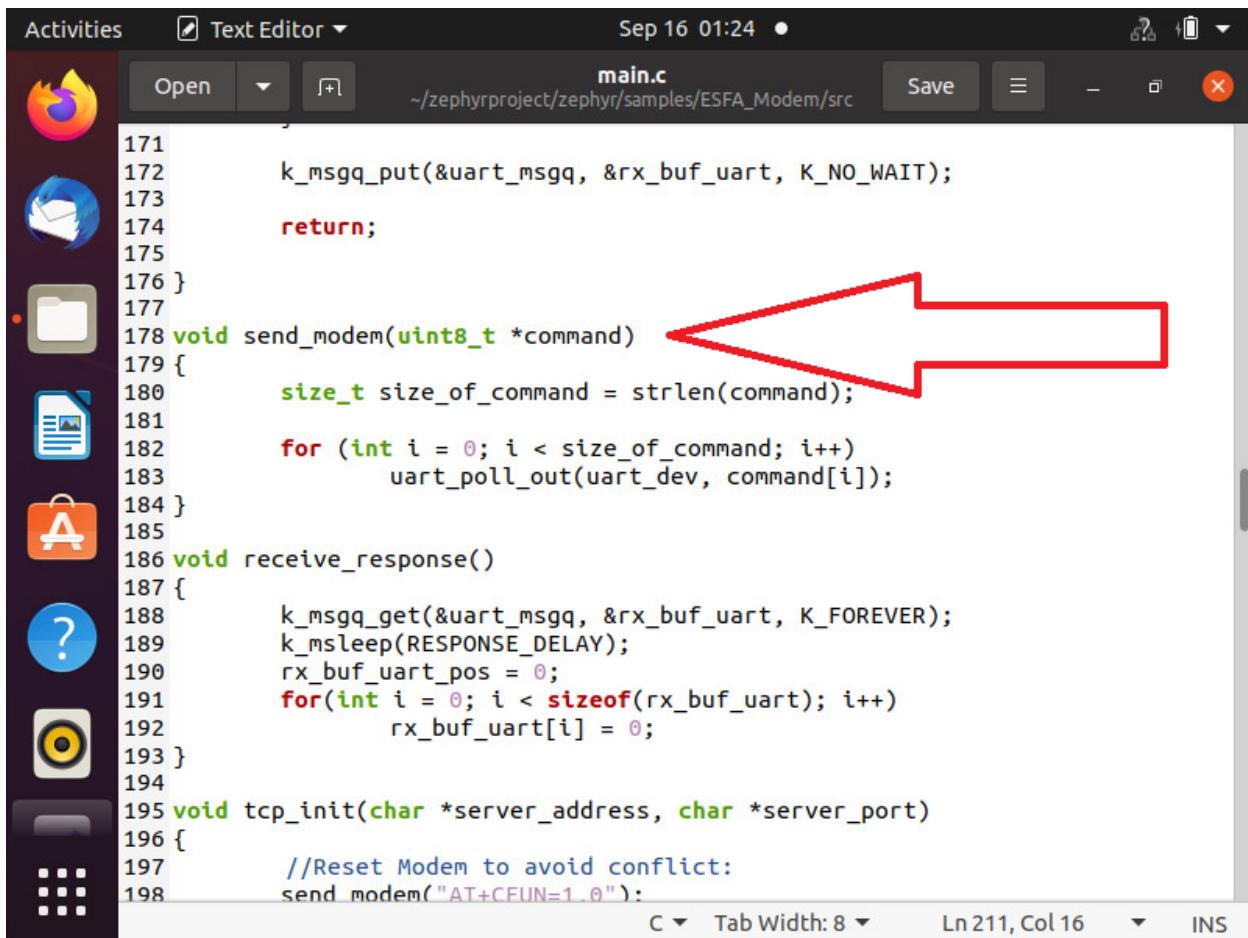


```
Activities Text Editor Sep 16 01:25 main.c
Open + ~zephyrproject/zephyr/samples/ESFA_Modem/src Save
1 #define RESPONSE_DELAY 1000
2 #define SLEEP_TIME_MS 1000
3 #define LED1_NODE DT_ALIAS(led1)
4 #define UART_NODE DT_ALIAS(uart2)
5 #define SPI_CS DT_ALIAS(cs)
6 #define SPI_NODE DT_ALIAS(spi)
7 #define spi_act_pin DT_ALIAS(spi_actpin)
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;
```

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:



The screenshot shows a terminal window titled "Text Editor" with the file "main.c" open. The code is written in C and includes functions for sending commands to a modem via UART. A red arrow points to the "send_modem" function, which takes a pointer to a uint8_t array and iterates through its elements, calling "uart_poll_out" for each character. The code also includes functions for receiving responses and initializing TCP connections.

```
171     k_msgq_put(&uart_msqq, &rx_buf_uart, K_NO_WAIT);
172
173     return;
174 }
175
176 }
177
178 void send_modem(uint8_t *command)
179 {
180     size_t size_of_command = strlen(command);
181
182     for (int i = 0; i < size_of_command; i++)
183         uart_poll_out(uart_dev, command[i]);
184 }
185
186 void receive_response()
187 {
188     k_msgq_get(&uart_msqq, &rx_buf_uart, K_FOREVER);
189     k_msleep(RESPONSE_DELAY);
190     rx_buf_uart_pos = 0;
191     for(int i = 0; i < sizeof(rx_buf_uart); i++)
192         rx_buf_uart[i] = 0;
193 }
194
195 void tcp_init(char *server_address, char *server_port)
196 {
197     //Reset Modem to avoid conflict:
198     send_modem("AT+CFUN=1,0");
```

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, setting 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 message 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

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

```
28 int rx_buf_uart_pos = 0;
29 int rx_buf_spi_pos = 0;
30
31 const char END_MESSAGE[] = "\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
47 };
48
49 struct spi_config spi_cfg = {
50     .frequency = 1000000,
51     .operation = SPI_OP_MODE_MASTER | SPI_WORD_SET(8),
52     .slave = 0,
53     .cs = &spi_cs
54 };
--
```

C Tab Width: 8 Ln 196, Col 2 INS

Activities Text Editor Sep 20 15:13

main.c ~/zephyrproject/zephyr/samples/ESFA_Modem/src

```
260 }
261
262 void tcp_init(char *server_address, char *server_port) {
263     //Reset Modem to avoid conflict:
264     send_modem("AT+CFUN=1,0");
265     send_modem(END_MESSAGE);
266     receive_response();
267
268     send_modem("AT&F0");
269     send_modem(END_MESSAGE);
270     receive_response();
271
272     send_modem("ATE0");
273     send_modem(END_MESSAGE);
274     receive_response();
275
276     send_modem("AT+QICLOSE=0");
277     send_modem(END_MESSAGE);
278     receive_response();
279
280     send_modem("AT+QIDEACT=1");
281     send_modem(END_MESSAGE);
282     receive_response();
283
284     send_modem("AT+QICSGP=1,1,\"mtnirancell\",\\\"\\\",\\\"\\\",0");
285     send_modem(END_MESSAGE);
286     receive_response();
287 }
```

C Tab Width: 8 Ln 219, Col 24 INS

Activities Text Editor Sep 20 15:13

main.c

```
281     send_modem(AT+QIDECT_1),
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();

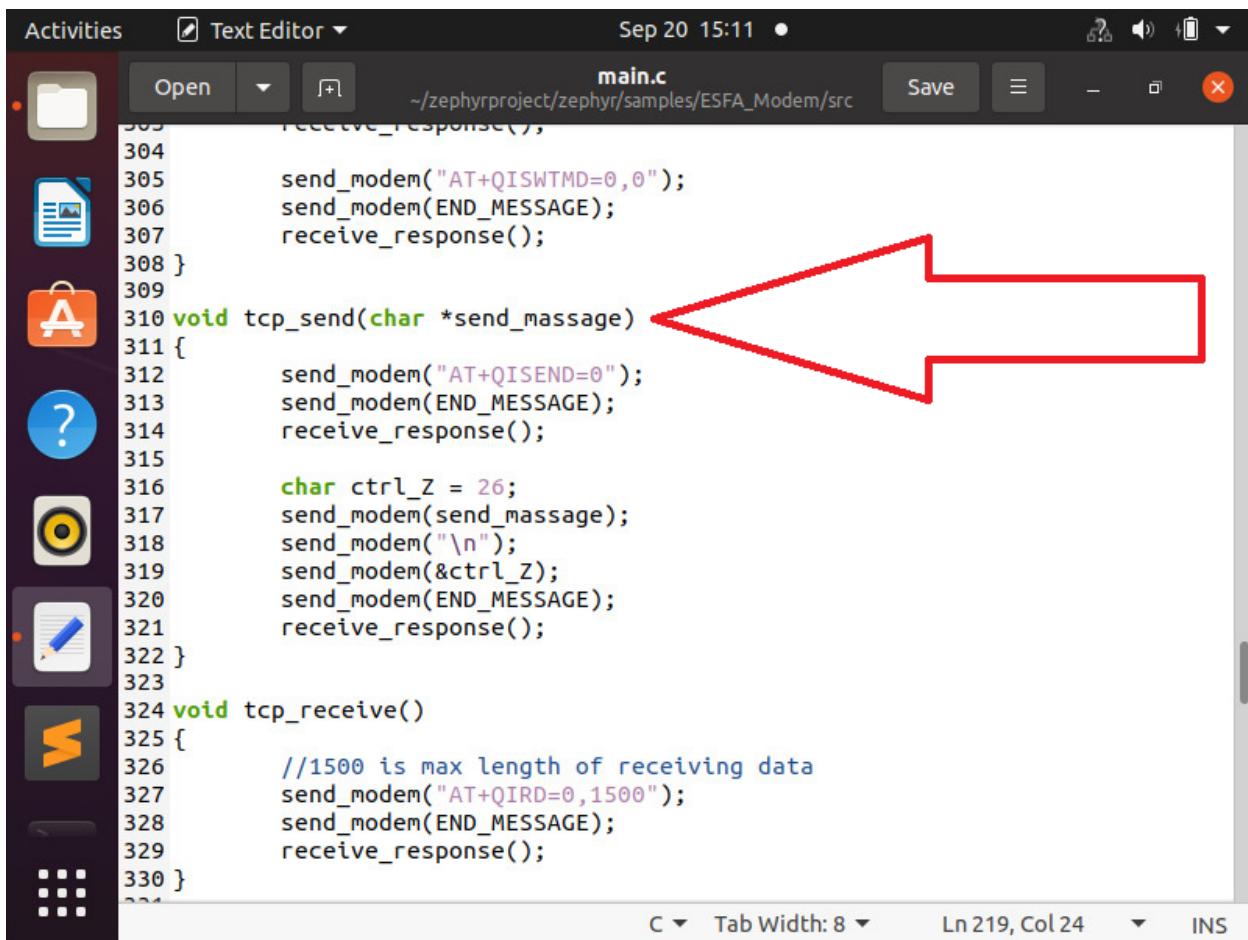
304
305     send_modem(AT+QISWTMD=0,0");
306     send_modem(END_MESSAGE);
307     receive_response();

308 }
```

C Tab Width: 8 Ln 219, Col 24 INS

Send a message 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.**



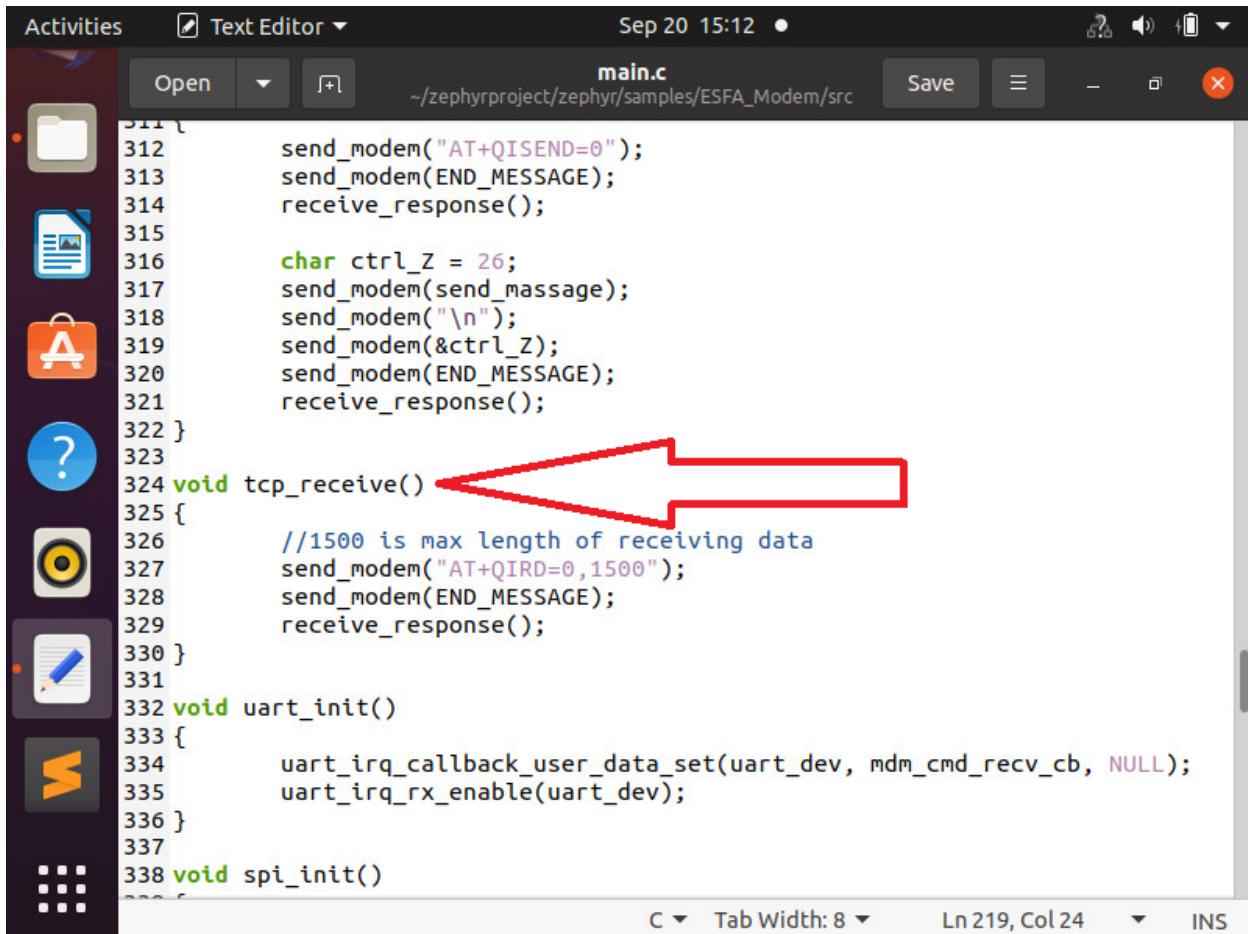
The screenshot shows a terminal window titled "main.c" in a "Text Editor". The file path is "/zephyrproject/zephyr/samples/ESFA_Modem/src". The code contains a function "tcp_send" which performs the following steps:

```
303     receive_response();
304
305     send_modem("AT+QISWTMD=0,0");
306     send_modem(END_MESSAGE);
307     receive_response();
308 }
309
310 void tcp_send(char *send_message)
311 {
312     send_modem("AT+QISEND=0");
313     send_modem(END_MESSAGE);
314     receive_response();
315
316     char ctrl_Z = 26;
317     send_modem(send_message);
318     send_modem("\n");
319     send_modem(&ctrl_Z);
320     send_modem(END_MESSAGE);
321     receive_response();
322 }
323
324 void tcp_receive()
325 {
326     //1500 is max length of receiving data
327     send_modem("AT+QIRD=0,1500");
328     send_modem(END_MESSAGE);
329     receive_response();
330 }
```

A large red arrow points from the text "Be careful you should get “SEND OK” in Modem’s response to confirm that message is sent successfully." to the line "send_modem("AT+QISEND=0");" in the code.

Receive a message 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.



```
Activities Text Editor Sep 20 15:12
main.c ~zephyrproject/zephyr/samples/ESFA_Modem/src Save
311
312     send_modem("AT+QISEND=0");
313     send_modem(END_MESSAGE);
314     receive_response();
315
316     char ctrl_Z = 26;
317     send_modem(send_message);
318     send_modem("\n");
319     send_modem(&ctrl_Z);
320     send_modem(END_MESSAGE);
321     receive_response();
322 }
323 void tcp_receive() {
324     //1500 is max length of receiving data
325     send_modem("AT+QIRD=0,1500");
326     send_modem(END_MESSAGE);
327     receive_response();
328 }
329
330 void uart_init()
331 {
332     uart_irq_callback_user_data_set(uart_dev, mdm_cmd_recv_cb, NULL);
333     uart_irq_rx_enable(uart_dev);
334 }
335
336 void spi_init()
337 }
```

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.