

### D1.a) Relevant configuration data for FreeRTOS

FreeRTOS general configuration is default from the project template.

Custom and changed parameters:

FreeRTOSConfig.h:

```
#define configNETWORK_INTERFACE_TO_USE 1
```

main.c:

```
// PC IP4 address
#define TCP_SERVER_ADDR1 192
#define TCP_SERVER_ADDR2 168
#define TCP_SERVER_ADDR3 100
#define TCP_SERVER_ADDR4 11
// VirtualBox IP4 address
// #define TCP_SERVER_ADDR1 10
// #define TCP_SERVER_ADDR2 0
// #define TCP_SERVER_ADDR3 2
// #define TCP_SERVER_ADDR4 15

#define TCP_SERVER_PORT 4040
```

### D1.b) Report about create of tasks and the parameters used, attach c-code

```
xTaskCreate(prvClientTask, "ClientTask", mainTCP_SERVER_STACK_SIZE, NULL, mainTCP_SERVER_TASK_PRIORITY + 1, &xTCPClientTaskHandle);
xTaskCreate(prvCannyEdgeTask, "CannyEdgeTask", mainTCP_SERVER_STACK_SIZE, NULL, mainTCP_SERVER_TASK_PRIORITY, &xCannyTaskHandle);
xTaskCreate(prvEncryptionTask, "EncryptionTask", mainTCP_SERVER_STACK_SIZE, NULL, mainTCP_SERVER_TASK_PRIORITY, &xEncryptionTaskHandle);
```

### D2.a) Relevant c-code for edge detection setup and screen shot of running algorithm. Provide result of picture with edge detection before/after

static void prvCannyEdgeTask( void \*pvParameters )

```
{
```

```
(void) pvParameters;
```

```
message_t plainBuf;
```

```
while(1)
```

```

{
    static bitmap_info_header_t ih;
    const pixel_t* in_bitmap_data = load_bmp("lenna.bmp", &ih);

    if (in_bitmap_data == NULL)
    {
        FreeRTOS_printf(("BMP image is not loaded.\n"));
        break;
    }

    FreeRTOS_printf(("BMP: width=%d height=%d bitspp=%d\n", ih.width, ih.height,
ih.bitspp));

    TickType_t startTick = xTaskGetTickCount();

    const pixel_t* out_bitmap_data = canny_edge_detection(in_bitmap_data, &ih, 40, 80, 1.0);

    if (out_bitmap_data == NULL)
    {
        FreeRTOS_printf(("failed canny_edge_detection.\n"));
        free((pixel_t*)in_bitmap_data);
        break;
    }

    free((pixel_t*)in_bitmap_data);

    if (save_bmp("lenna_ce.bmp", &ih, out_bitmap_data)) {
        FreeRTOS_printf(("BMP image is not saved.\n"));
    }

    if(!pack_pixels(ih.width, ih.height, out_bitmap_data, &plainBuf.pData, &plainBuf.len))
    {
        FreeRTOS_printf(("failed to pack pixels.\n"));
        free((pixel_t*)out_bitmap_data);
        break;
    }

    free((pixel_t*)out_bitmap_data);

    TickType_t endTick = xTaskGetTickCount();
    FreeRTOS_printf(("duration in ticks: %d\n", endTick - startTick));

    if (xQueueSend(xEncryptQueue, &plainBuf, pdMS_TO_TICKS(1000)) != pdPASS)
    {

```

```

        free((uint8_t *)plainBuf.pData);
    }
    else
    {
        // the encryption task releases the buffer
    }
}

vTaskDelete(NULL);
}

```



D2.b) Relevant c-code for integrated encryption task running system. Provide output for some encrypted data

```

// p = 61, q = 53
// n = p * q
// de = 1 mod (p - 1)(q - 1)
// de = 413 * 17 = 1 mod 780
#define N 3233
#define E 17
#define D 413

static void prvEncryptionTask( void *pvParameters )
{
    (void) pvParameters;

    message_t plainBuf;
    message_t encBuf;

    while(1)
    {
        if(xQueueReceive(xEncryptQueue, &plainBuf, pdMS_TO_TICKS(1000)) == pdPASS)
        {
            if(plainBuf.pData != NULL)
            {
                TickType_t startTick = xTaskGetTickCount();

                FreeRTOS_printf(("encryption\n"));
            }
        }
    }
}

```

```

int32_t nx = *((int32_t*)plainBuf.pData);
int32_t ny = ((int32_t*)plainBuf.pData)[1];
pixel_t* pixels = &(pixel_t)((int32_t*)plainBuf.pData)[2];

pixel_t* enc = encrypt_image(pixels, nx, ny, N, E);
if (enc == NULL) {
    FreeRTOS_printf(("failed to allocate an encryption
buffer.\n"));
    free((uint8_t*)plainBuf.pData);
}
else
{
    free((uint8_t*)plainBuf.pData);

    if (!pack_pixels(nx, ny, enc, &encBuf.pData, &encBuf.len))
    {
        FreeRTOS_printf(("failed to pack pixels.\n"));
        free((pixel_t*)enc);
    }
    else
    {
        free((pixel_t*)enc);

        TickType_t endTick = xTaskGetTickCount();
        FreeRTOS_printf(("duration in ticks: %d\n", endTick
- startTick));

        if (xQueueSend(xTCPClientQueue, &encBuf,
pdMS_TO_TICKS(1000)) != pdPASS)
        {
            free((uint8_t*)encBuf.pData);
        }
    }
}
}
}

vTaskDelete(NULL);
}

```

D2.c) Relevant c-code TCP client tasks running system. Provide output transmitted packets

```
static void prvClientTask( void *pvParameters )
```

```

{
    (void)pvParameters;

    message_t uploadBuf;
    int frame = 1;

    ulTaskNotifyTake(pdTRUE, portMAX_DELAY); // wait for IP link is UP

    while(1)
    {
        if (xQueueReceive(xTCPCClientQueue, &uploadBuf, pdMS_TO_TICKS(1000)) ==
pdPASS)
        {
            if (uploadBuf.pData != NULL)
            {
                TickType_t startTick = xTaskGetTickCount();

                FreeRTOS_printf(("socket transmission\n"));

                socket_transmission(uploadBuf.pData, uploadBuf.len, frame++);

                free((uint8_t*)uploadBuf.pData);

                TickType_t endTick = xTaskGetTickCount();
                FreeRTOS_printf(("duration in ticks: %d\n", endTick - startTick));
            }
        }
    }

    vTaskDelete(NULL);
}

```

D3.a) Relevant c-code for setting up communications channels between FreeRTOS task

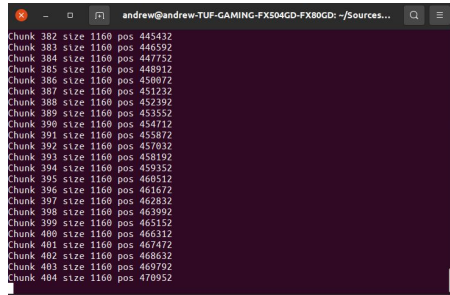
```

if (xQueueSend(xTCPCClientQueue, &encBuf, pdMS_TO_TICKS(1000)) != pdPASS)
{
    free((uint8_t*)encBuf.pData);
}

if (xQueueReceive(xTCPCClientQueue, &uploadBuf, pdMS_TO_TICKS(1000)) == pdPASS)
{
}

```

D4.a) Report and screen shot for TCP server executing on cloud system and accessing transmitted packets. Provide output of received packets

A screenshot of a terminal window with a dark purple background. The window title is "andrew@andrew-TUP-GAMING-FX504GD-FX80GD: ~/Sources...". The terminal displays a list of received TCP packets, each on a new line. Each line follows the format "Chunk [number] size [size] pos: [position]". The packets are numbered from 382 to 484, with a size of 1160 and positions ranging from 445432 to 478952.

```
Chunk 382 size 1160 pos: 445432
Chunk 383 size 1160 pos: 446592
Chunk 384 size 1160 pos: 447752
Chunk 385 size 1160 pos: 448912
Chunk 386 size 1160 pos: 450072
Chunk 387 size 1160 pos: 451232
Chunk 388 size 1160 pos: 452392
Chunk 389 size 1160 pos: 453552
Chunk 390 size 1160 pos: 454712
Chunk 391 size 1160 pos: 455872
Chunk 392 size 1160 pos: 457032
Chunk 393 size 1160 pos: 458192
Chunk 394 size 1160 pos: 459352
Chunk 395 size 1160 pos: 460512
Chunk 396 size 1160 pos: 461672
Chunk 397 size 1160 pos: 462832
Chunk 398 size 1160 pos: 463992
Chunk 399 size 1160 pos: 465152
Chunk 400 size 1160 pos: 466312
Chunk 401 size 1160 pos: 467472
Chunk 402 size 1160 pos: 468632
Chunk 403 size 1160 pos: 469792
Chunk 404 size 1160 pos: 470952
```

D5.c) Technique and hardware for interconnecting the camera onto the real platform resolution/picture quality, energy consumption, speed of communications, reliability, financial estimation

Picture quality: 512 x 512 pixels

Energy consumption: maximum 1W

Speed of Communications: 25 fps, 6MB/s

Financial estimation: maximum \$20

D6.c) Report including references to relevant hardware OS chosen as most significant for creating the

real hardware platform:

OS: FreeRTOS

Hardware: ESP32 (\$4-10)