Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS

“CRC 32 and AES 128”

Practice 1

Student: **REYES OLVERA IVAN EDGAR**

Teacher: **NICOLÁS SANTANA SERGIO**

Tlaquepaque, Jalisco. February 29, 2022

Table of Contents

TABLE OF FIGURES

1. Introduction

1.1. CRC32

Cyclic Redundancy Check is a powerful algorithm (technique) used to verify integrity and detect changes between source and target digital data. The CRC generates an initial checksum of the data in memory that can be later compared with the calculated checksum for mismatch. A checksum mismatch can indicate an otherwise undetectable memory fault. The number after CRC is a reference of the number of bits to storage the calculated checksum. The checksum is the residue of the division (XOR bitwise operation) between the original data and a polynomial number (coefficients), this polynomial is usually 8 (CRC8), 16 (CRC32) or 32 (CRC32) bits.

1.2. AES128

Advanced Encryption Standard is a technique capable of using cryptographic keys of 128/192/256 bits to encrypt and decrypt data, based in Rijndael algorithm. An algorithm that uses a key is, in general, much more secure. If the algorithm itself is secure in its design, then the data is secure (as long as the key is secure) even if the encryption algorithm is known. The only way to decipher the message is through the use of the correct key. There are two basic types of keys: Symmetric and Public. In a symmetric key algorithm, both encryption and decryption processes use the same key, which must be kept secret. In a public key system, two keys are used: one public (used to cipher messages) and another, private and secret (used to decipher the message).

1.3. Requirements

1. CRC 32 bits

The new protocol layer must provide verification of the data integrity using a CRC32 checksum.

2. AES 128 bits

The message must be cypher with a 128 bits AES.

3. Ethernet and TCP/IP

The new layer must be implemented as a C language library and testing by using an Ethernet and TCP/IP Python script.

4. Nodes Functionalities

Every node must be capable of transmit and receive messages through the new layer

5. Testing

Implement an application to transmit 8 different (size and data) data packages in both directions.

2. Methodology

2.1 Requirement 1

In order add the CRC functionality in the base project, we need to add the driver files, because CRC is performed by hardware in K64F board. As Figure 1 shows, .c and .h files are already in the project and can be used to add CRC functionality by including the header file "fsl_crc.h" anywhere you needed. In this project I added the header file in aes.c file.

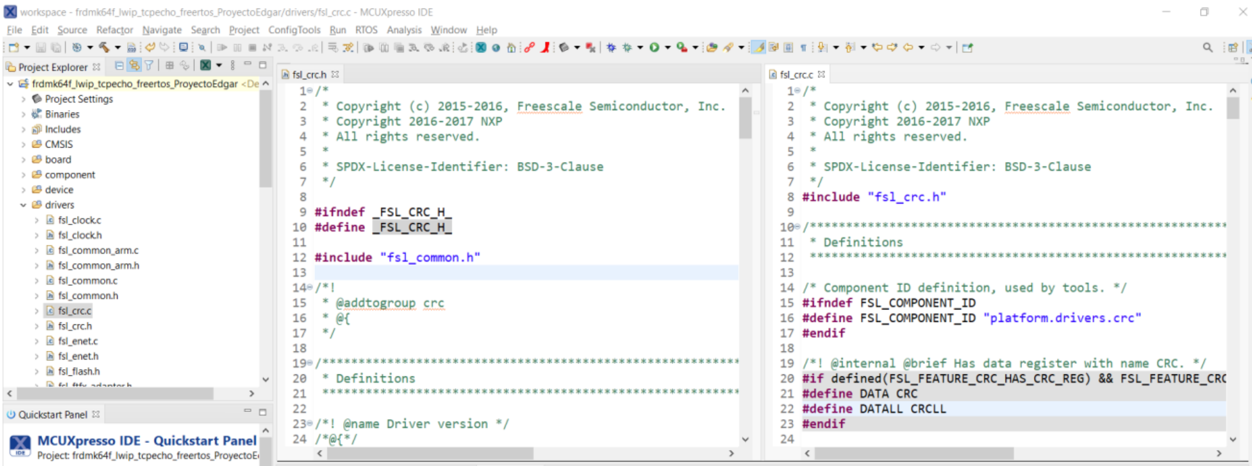


Figure 1. CRC driver files.

2.2 Requirement 2

This functionality is hardware implement as well, the base project has already the files to perform AES encryption, when a message was received the first step is to calculate the CRC to verify the data integrity, if the calculated CRC is equal to the added CRC then the program decrypt the message and erase the added CRC from the message, otherwise a message is printed in screen warning the user about data corruption and delete the received message. When a message will be sent, the first step is to encrypt the message, after this the CRC must be calculated and added to the message.

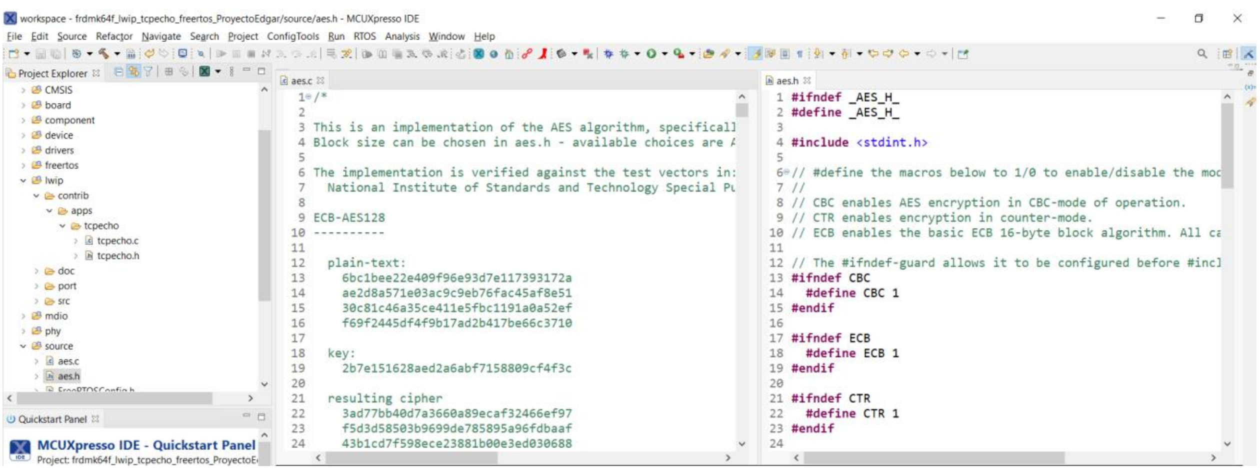


Figure 2. AES files.

2.3 Requirement 3

The CRC and AES functionalities were added with the files “aes.c” and “fsl_crc.c” and the header files “aes.h” and “fsl_crc.h”, by adding these files in any project users must be able to use CRC and AES using the API functions created (aes_send_task and aes_recv_task)and located in “aes.c” file.

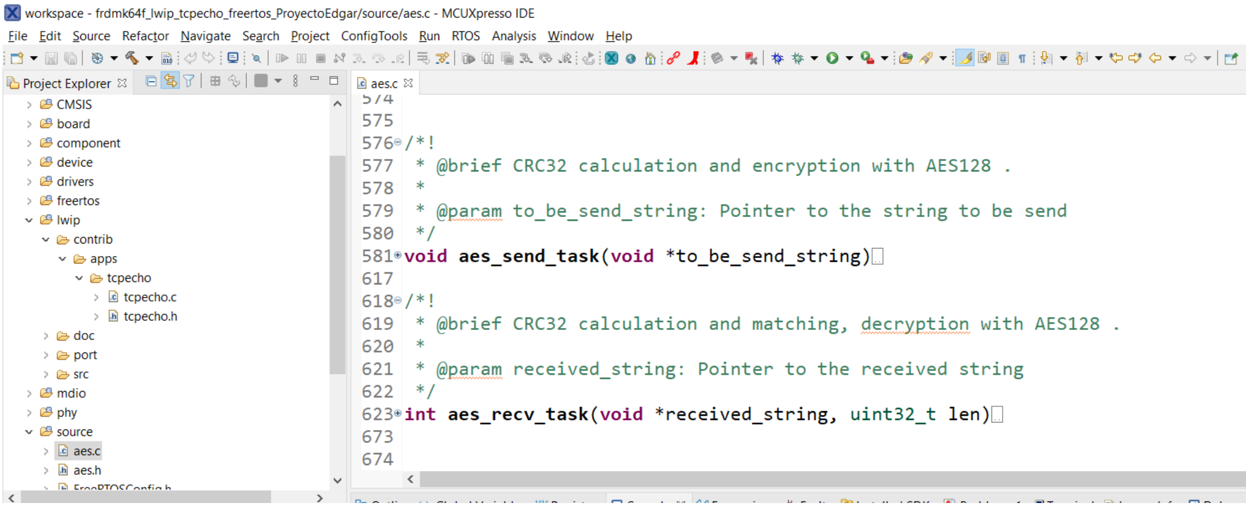


Figure 3. AES created API´S.

The testing set was performed with the K64F as a server and executing the python script as a client, but the inverse functionality is implemented as well by setting the next Macro definitions to 1 and executing the server python script, located in "tcpecho.c" file.

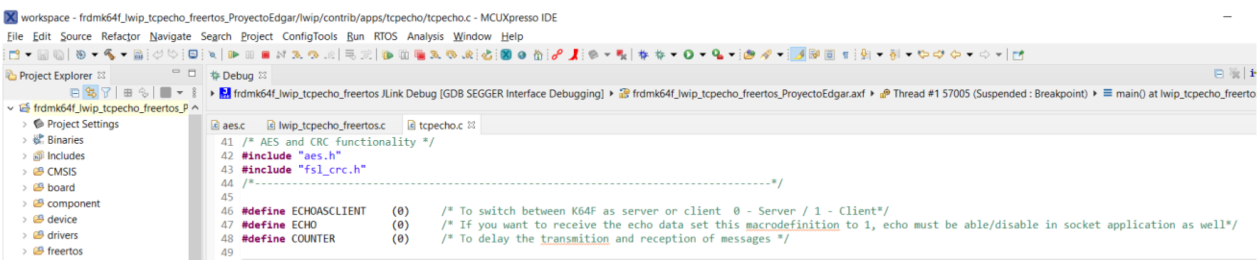
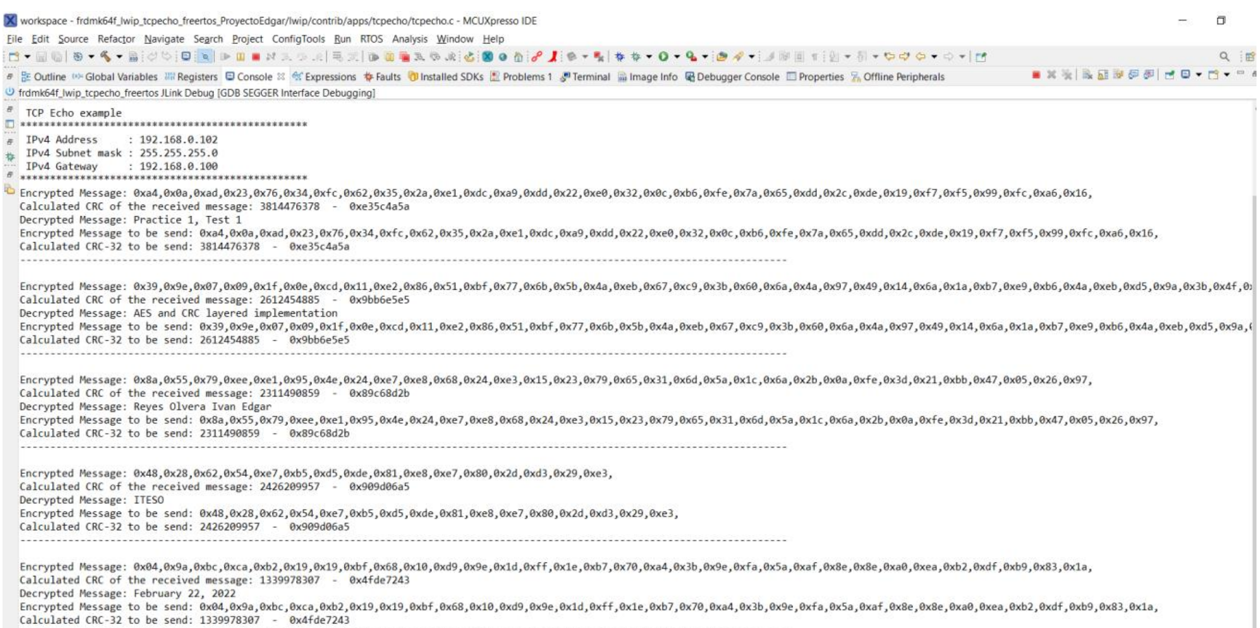


Figure 4. Macro-definitions to switch between server and client.

As figure 5 show, we tried several messages and every time the program performed the expected functionality.



```
myssn_client.py - Shortcut
myssn INFO: connecting to 192.168.0.102 port 7
$ Practice 1, Test 1
myssn INFO: sending data: b'Practice 1, Test 1'
myssn INFO: encrypted message: b'\xa4\n\xad#v4\xfcb5*\xe1\xdc\xa9\xdd"\xe02\x0c\xb6\xfeze\xdd,\xde\x19\xf7\xf5\x99\xfc\xa6\x16'
myssn INFO: tx crc32 = 3814476378, crc bytes = b'ZJ\\\xe3'
myssn DATA: b'Practice 1, Test 1'
DATA: b'Practice 1, Test 1'
----------------------------------------------------------------------------------------------------------------------------

$ AES and CRC layered implementation
myssn INFO: sending data: b'AES and CRC layered implementation'
myssn INFO: encrypted message: b'9\x9e\x07\t\x1f\x0e\xcd\x11\xe2\x86Q\xbfwk[J\xebg\xc9;`jJ\x97I\x14j\x1a\xb7\xe9\xb6J\xeb\xd5\x9a;O\x8b\x1f\xc2\xcf\xd3\x927\xb8\x89\xfb&'
myssn INFO: tx crc32 = 2612454885, crc bytes = b'\xe5\xe5\xb6\x9b'
myssn DATA: b'AES and CRC layered implementation'
DATA: b'AES and CRC layered implementation'
----------------------------------------------------------------------------------------------------------------------------

$ Reyes Olvera Ivan Edgar
myssn INFO: sending data: b'Reyes Olvera Ivan Edgar'
myssn INFO: encrypted message: b'\x8aUy\xee\xe1\x95N$\xe7\xe8h$\xe3\x15#ye1mZ\x1cj+\n\xfe=!\xbbG\x05&\x97'
myssn INFO: tx crc32 = 2311490859, crc bytes = b'+\x8d\xc6\x89'
myssn DATA: b'Reyes Olvera Ivan Edgar'
DATA: b'Reyes Olvera Ivan Edgar'
----------------------------------------------------------------------------------------------------------------------------

$ ITESO
myssn INFO: sending data: b'ITESO'
myssn INFO: encrypted message: b'H(bT\xe7\xb5\xd5\xde\x81\xe8\xe7\x80-\xd3)\xe3'
myssn INFO: tx crc32 = 2426209957, crc bytes = b'\xa5\x06\x9d\x90'
myssn DATA: b'ITESO'
DATA: b'ITESO'
----------------------------------------------------------------------------------------------------------------------------

$ February 22, 2022
myssn INFO: sending data: b'February 22, 2022'
myssn INFO: encrypted message: b'\x04\x9a\xbc\xca\xb2\x19\x19\xbfh\x10\xd9\x9e\x1d\xff\x1e\xb7p\xa4;\x9e\xfaZ\xaf\x8e\x8e\xa0\xea\xb2\xdf\xb9\x83\x1a'
myssn INFO: tx crc32 = 1339978307, crc bytes = b'Cr\xdeO'
myssn DATA: b'February 22, 2022'
DATA: b'February 22, 2022'
----------------------------------------------------------------------------------------------------------------------------
```

Figure 5. Testing set.

2.4 Requirement 4

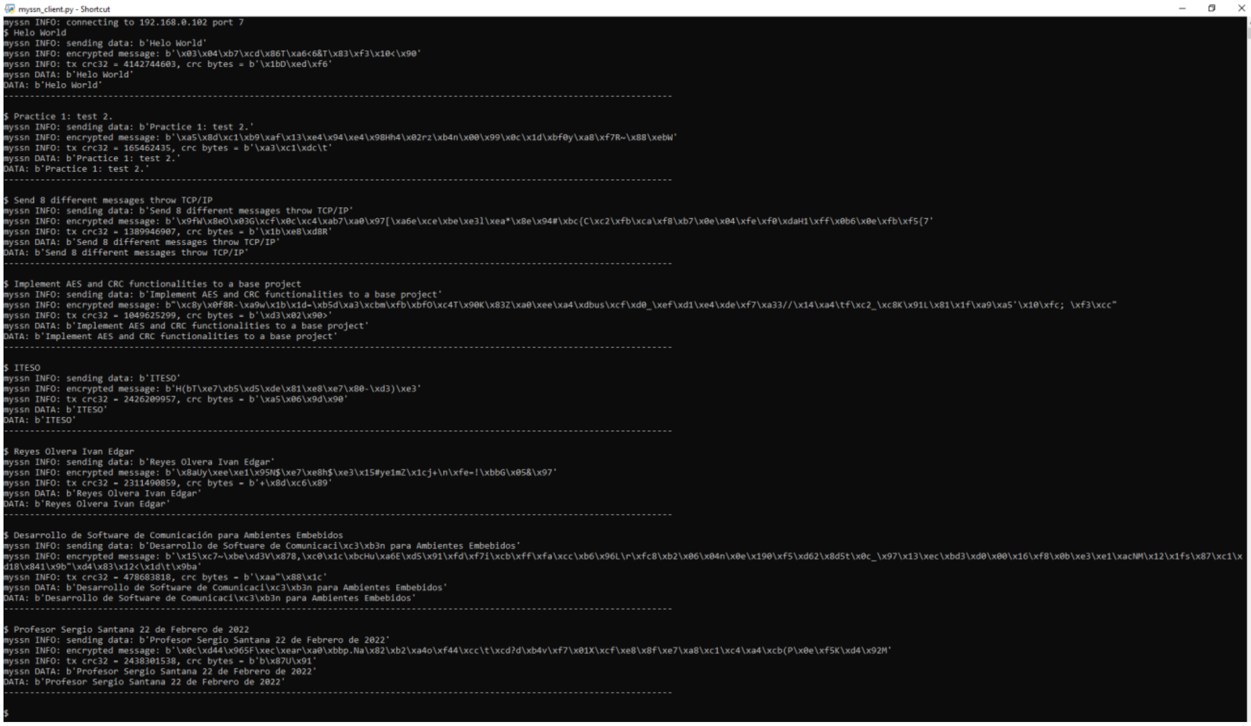
This requirement Will be show in the Test section.

3. TESTS

This test could be done easily by programming the K64F board as a client and sending in a for cycle 8-character arrays, but the python server script does not work, I tried several IP addresses, but it always showed the same error as in figure 6, when it did not show the error, it simply did not connect to the board. So I ran the test as the requirement 3, the figure 7 show the results.

```
Exception has occurred: OSError ×
[WinError 10049] The requested address is not valid in its context
  File "C:\Users\MCRO\Desktop\ESE ITESO\Desarrollo de Software de Comunicación para Ambientes Embebidos\Practica 1\mySafeAndSecureNetwork\pythonScripts\myssn.py", line 59, in server_create
    sock.bind(server_address)
  File "C:\Users\MCRO\Desktop\ESE ITESO\Desarrollo de Software de Comunicación para Ambientes Embebidos\Practica 1\mySafeAndSecureNetwork\pythonScripts\myssn_server.py", line 8, in <module>
    server = myssn.server_create(SERVER_ADDRESS)
```

Figure 6. Python Server script error.



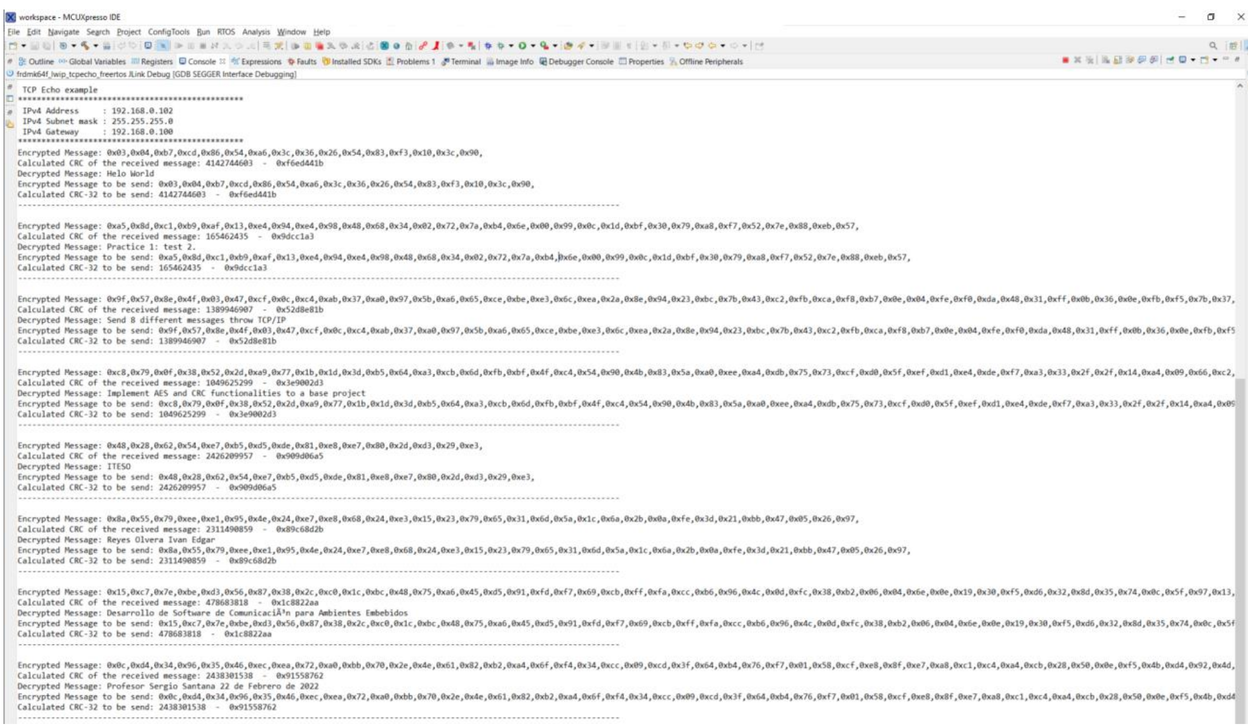


Figure 7. 8 different messages test.

A video with the test of the functionality could be found in the next link: https://www.youtube.com/watch?v=r88tbtATsCY

4. Functions explanation

In this project we implement the follow macro definition to choose between the K64F as a server or a client:

```
/* K64F Client or Server */
#define SERVER          (1)         /* By seting this Macro definition to 1 the K64F board will be a server, otherwise will be a client */
```

Then, depending on the last decition, we created a new thread in the system to execute the tcpecho_server_thread or the tcpecho_client_thread:

```
void tcpecho_server_init(void)
{
  sys_thread_new("tcpecho_server_thread", tcpecho_server_thread, NULL, DEFAULT_THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);
}

void tcpecho_client_init(void)
{
  sys_thread_new("tcpecho_client_thread", tcpecho_client_thread, NULL, DEFAULT_THREAD_STACKSIZE, DEFAULT_THREAD_PRIO);
}
```

Inside the thread, we use the tcp\ip functions from the tcpipecho example project to open a socket and follow the steps to stay listen as a server or sending data as a client.

```
static void tcpecho_server_thread(void *arg)
{
  struct netconn *conn, *newconn;
  err_t err;
  LWIP_UNUSED_ARG(arg);
  uint32_t len_to_be_send = 0;

  /* Create a new connection identifier. */
  /* Bind connection to well known port number 7. */
#if LWIP_IPV6
  conn = netconn_new(NETCONN_TCP_IPV6);
  netconn_bind(conn, IP6_ADDR_ANY, 7);
#else  /* LWIP_IPV6 */
  conn = netconn_new(NETCONN_TCP);
  netconn_bind(conn, IP_ADDR_ANY, 7);
#endif /* LWIP_IPV6 */
  LWIP_ERROR("tcpecho: invalid conn", (conn != NULL), return;);

  /* Tell connection to go into listening mode. */
  netconn_listen(conn);

/* Grab new connection. */
err = netconn_accept(conn, &newconn);


while ((err = netconn_recv(newconn, &buf)) == ERR_OK)
err = netconn_write(newconn, data, len, NETCONN_COPY);

/* Close connection and discard connection identifier. */
netconn_close(newconn);
netconn_delete(newconn);
```

```
static void tcpecho_client_thread(void *arg)
{
  struct netconn *conn, *newconn;
  err_t err;
  LWIP_UNUSED_ARG(arg);
  ip_addr_t IpAdd;
  u32_t counter = 50000000;

  /* Create a new connection identifier. */
  conn = netconn_new(NETCONN_TCP);
  IP4_ADDR(&IpAdd, 192, 168, 0, 101);
  LWIP_ERROR("tcpecho: invalid conn", (conn != NULL), return;);
  err = netconn_connect(conn, &IpAdd, 7);

  err = netconn_write(conn, data, send_len, NETCONN_COPY);
  err = netconn_recv(conn, &buf);
```

I decided to create 2 functions "aes_send_task" and "aes_recv_task" to encrypt\decrypt and calculate the CRC32 of every message, no matter if the boar was set as a server or client:

aes_send_task:

In order to send an encrypted message and add to it the calculated CRC, the function initialize the AES,

```
/* Init the AES context structure */
AES_init_ctx_iv(&ctx, key, iv);
```

then add zeros to a copy of the original message because the length must be a multiple of 16, and cypher the message

```
/* To encrypt an array its lenght must be a multiple of 16 so we add zeros */
to_be_send_string_len = strlen(to_be_send_string);
padded_len = to_be_send_string_len + (16 - (to_be_send_string_len%16) );
memcpy(padded_msg, to_be_send_string, to_be_send_string_len);
AES_CBC_encrypt_buffer(&ctx, padded_msg, (uint32_t)padded_len);
```

Print the encrypted message, initialize the CRC peripheral and calculate the CRC3, the print the CRC32 and finally add the calculated CRC3 to the encrypted message, after this the message is ready to be send and is already encrypted and with the CRC3 added.

```
/* Initialize CRC Peripheral */
InitCrc32(CRC0, 0xFFFFFFFFU);

/* Calculate the message CRC */
CRC_WriteData(base, padded_msg, (uint32_t)padded_len);

/* Add the CRC to the message to be send  */
for(i = 0; i < 4; i++)
{
    padded_msg[padded_len + i] = (checksum32 >> (8 * i)) & 0xFF;
}
padded_len += 4;
*len = padded_len;
for(i = 0; i < padded_len; i++)
{
    *(uint8_t *)(to_be_send_string + i) = padded_msg[i];
}
```

Note*: it is user responsibility assure that the original string has 4 extra bytes to storage the CRC32, because aes_send_task function writes the calculated CRC3 after the end of the string send as a parameter to the function.

aes_recv_task:

This function prints the received message, then stores the CRC32 received with the message and then deletes it (fills it with zeros). Initialize the CRC322 peripheral and calculate the CRC32 of the received message, without the received CRC32.

```
/* Storage the CRC received with the message */
checksum32 = *(uint32_t *)(received_string + len - 4);

/* Erase the received CRC to be updated before echo */
*(uint32_t *)(received_string + len - 4) = (uint32_t)0x00000000;

/* Initialize CRC Peripheral */
InitCrc32(CRC0, 0xFFFFFFFFU);

/* Calculate the message CRC */
CRC_WriteData(base, (uint8_t *)(received_string), (len - 4));
```

Then compare the calculated CRC vs the received one, if they are equal proceeds to decrypt the message, otherwise an Data corruption message will be print in the screen.

```
/* If the calculated CRC and received CRC are equal, then we decrypt the message */
if(CRC_Get32bitResult(base) == checksum32)
{
    /* Init the AES context structure */
    AES_init_ctx_iv(&ctx, key, iv);

    /* Decrypt the message */
    AES_CBC_decrypt_buffer(&ctx, (uint8_t *)(received_string), (len - 4));

    /* Printf the decrypted message */
    memcpy(padded_msg, (received_string), len - 4);
    PRINTF("Decrypted Message: %s\n", padded_msg);

    return 1;
}
/* Otherwise */
else
{
    PRINTF("Invalid CRC, data was corrupted\n");
    return 0;
}
```

By using these 2 functions the program is able to make an echo by receiving the messages, checking the CRC32, decrypting and then encrypting and adding the CRC32 to send back the received message.

5. Conclusions

The layered implementation has several advantages, by programming in modules portability between projects will be easier, the debugging is easier too, because you add a hole piece of software as a module then you know where to looking for any error if your program does not work after the module addition.

But if your program manages several functionalities as a module, then the interaction between the layers become complex and you could start to use functions from the upper layers in lower layers, and this is not allowed in layered programming.

The practice helped me to see this more clearly. The CRC and AES functionalities are essential nowadays and take some time to implement them and learn their structure and algorithm was a good exercise. For example, I learned that the CRC peripheral must be configure before be used every single time, this could be explained by the fact that the initialization configure the CRC Polynomial and erase the registers were the calculated CRC will be storage, ignore this step will make you to calculate a wrong CRC.

The source code (MCU expresso project) could be found in the next GIT repository:

https://github.com/iereyesfi/Practice-1-AES-and-CRC..git