Aerospace & Defence | Lighting | Power Management | Internet of Things | Transportation

Components – EMEA

Engineering DACH

PTC and TrustZone on SAML11

Quang Hai Nguyen

Revision 1.0, 05.06.2019

Hands-on

Five Years Out

Revision History

Revision, Date	Editor	Subject (major changes)
Revision 0.1, 03.06.2019	Quang Hai Nguyen	Preliminary
Revision 1.0, 05.06.2019	Quang Hai Nguyen	Release

Table of Contents

List of Abbreviations

List of Figures

List of Icon Identifiers

Table 1: Icon Identifiers List

	Extra information about a topic
	Task need to be done
	Important information or a warning
	Result expected to see

Overview

Introduction

This hands-on will show you how to create an access control application based on SAML11 microcontroller with TrustZone capability, and QT3 Xplained keypad.

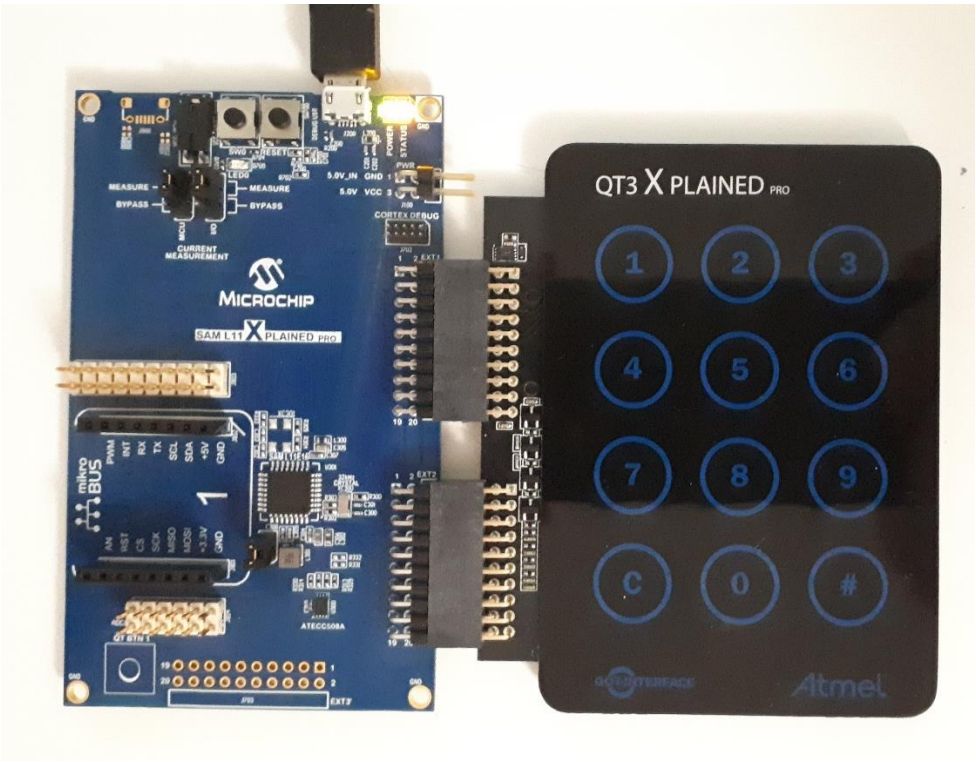


Figure 1: Hands-on Setup

Description

The secure application located in TZ initializes the PTC and process the input from the keypad. The user can enter the passcode through the keypad. Whenever user presses a key, the pressed key is displayed on the console terminal. The user can enter number from 0 – 9 as passcode, clear the previous value, and start authenticating.

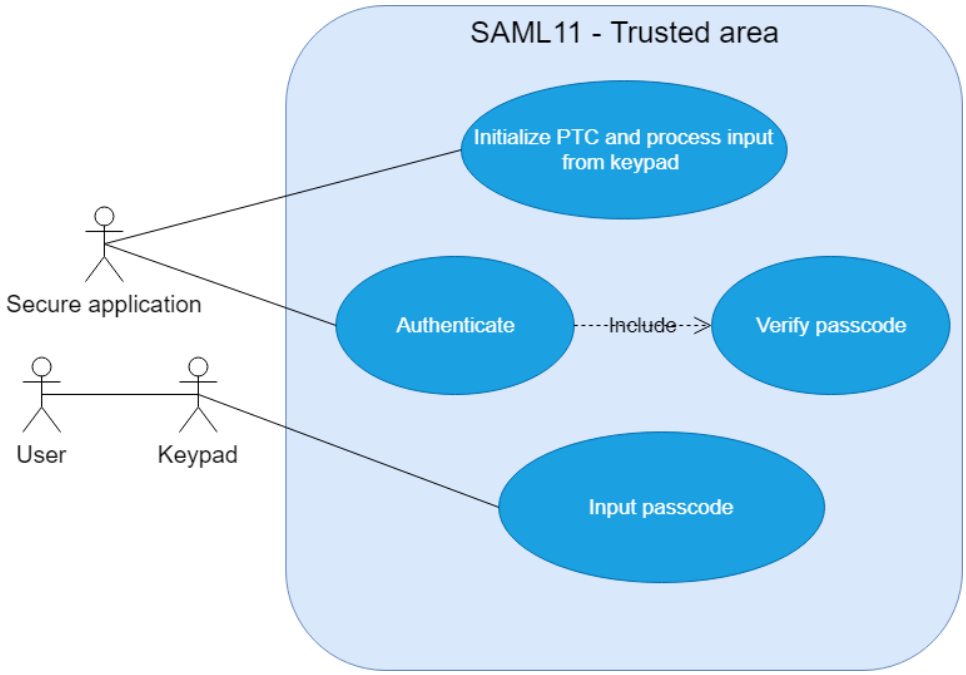


Figure 2: Use case diagram

When the authentication process has been started, the secure application verifies if the passcode entered by user is correct or not and prints the result on the console terminal.

Assignments

- Assignment 1: import the project and explore the project configuration
- Assignment 2: Adding the code

Goal

After this hands-on, you will know the benefits of using TZ and PTC for access control application. In addition, you will have the confidence to demonstrate the hands-on to customers.

Requirement

Hardware

- SAML11 Xplained
- QT3 Xplained
- Type A-to-micro USB cable

Software

- Atmel Studio version 7
- TeraTerm

The software is already installed if you are using the virtual machine. Otherwise please refer to the Installation Guide for more information on how to get these programs.

Assignments

Assignment 1: import the project and explore the project configuration

Open the project with AtmelStudio

Navigate to the project folder and double click on the .atsln file. After clicking AtmelStart starts automatically.

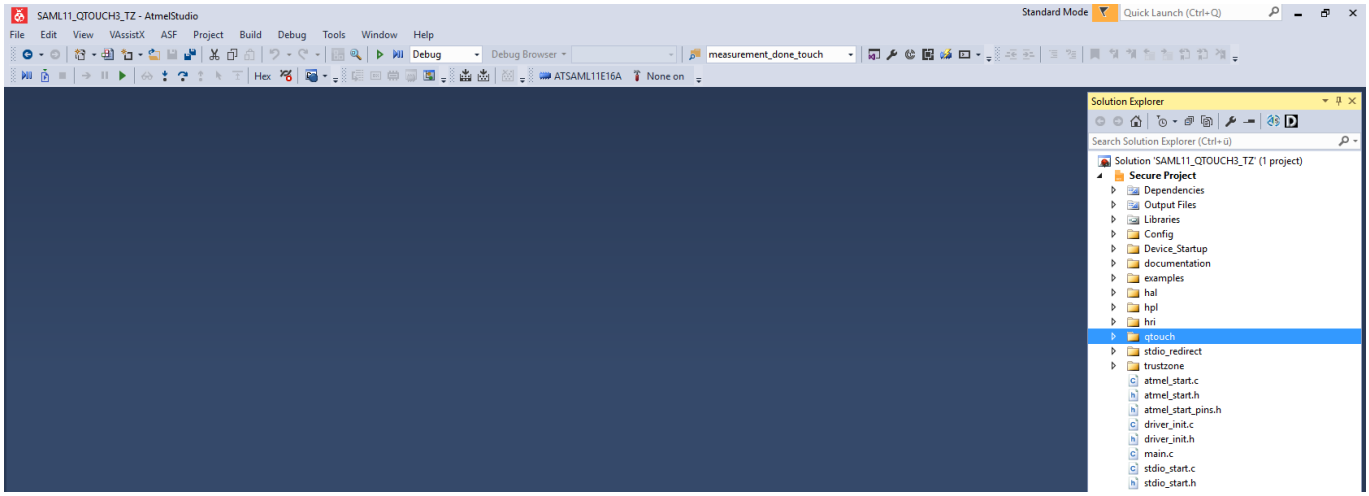


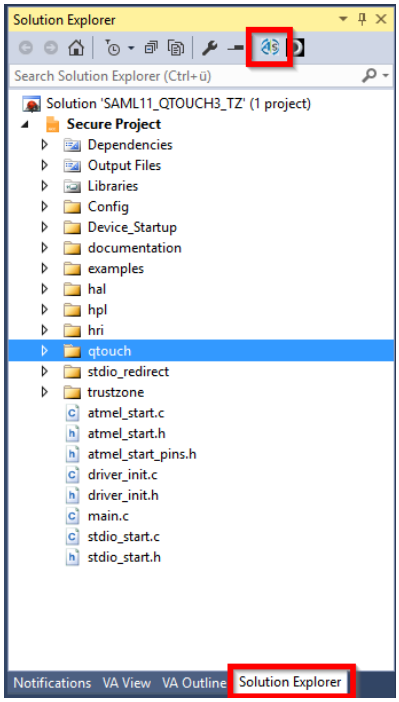
Figure 3: AtmelStudio

Alternatively, you can start AtmelStudio first, then choose File → Open → Project/Solution → point to .atsln file

Exploring the project configuration with AtmelStart

Click on the Solution Explorer tab and choose AtmelStart icon

AtmelStart is an online tool so please make sure that you have the internet connection



If AtmelStart started correctly, the project configuration should be as following:

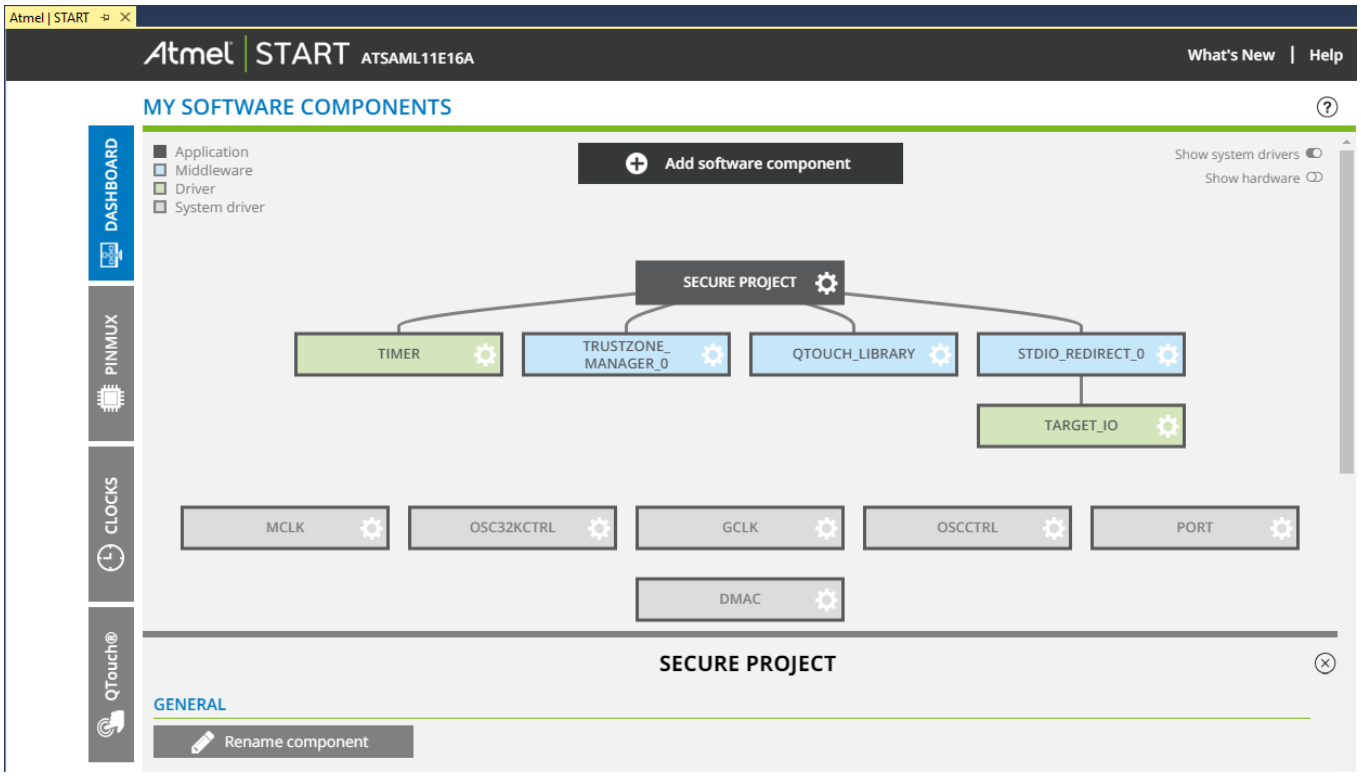
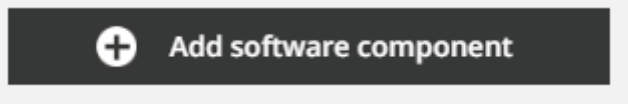


Figure 4: Project configuration

The project includes four components Timer, TrustZone manager, Qtouch library, and stdio redirect. Timer and Qtouch library take care of the interfacing between QT3 Xplained and SAML11 Xplained boards. TrustZone manager initializes the memory zone of the microcontroller and Stdio redirect allow routing debug information to the terminal on the PC.

Those components can be added by click at Add software components button



You are more than welcome to explore the detail of each component by clicking on it.

Please do not modify the configuration

Compile the project and see task list

To compile the project, please click Build → Build Solution or simply press F7

After compiling the project, click View → Task List to see the tasks need to be done for this hands-on. Those Tasks will be discussed in detail in the next assignment

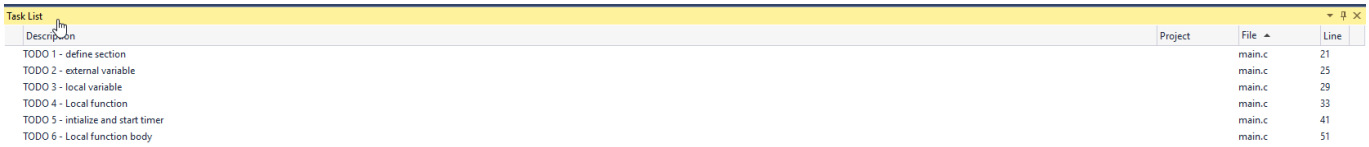


Figure 5: Task list of the hands-on

Assignment 2: Adding the code

If you get stuck at any point, there is a solution available. But please try it on your own before using the solution. Learning by doing.

Please refer the text file for easy copy paste the code

We start to fill the code into our project

TODO 1 – define section

Double click on TODO 1 to jump to the code section.

Fill in the code below right after the TODO line.

```
#define TOUCH_TIMER_MILI            5
#define KEY_PRESS_THRESHOLD         25
#define NUM_OF_PASSCODE_CHAR        5
```

TODO 2 – external variable

Double click on TODO 2 to jump to the code section.

Fill in the code below right after TODO line

```
extern volatile uint8_t measurement_done_touch;
```

TODO 3 – local variable

Double click on TODO 3 to jump to the code section.

Fill in the code below right after TODO line

```
typedef enum KEY_PAD
{
      KEY_NULL = -1, //No key is stored so far
      KEY_1,
      KEY_2,
      KEY_3,
      KEY_4,
      KEY_5,
      KEY_6,
      KEY_7,
      KEY_8,
      KEY_9,
      KEY_C,
      KEY_0,
      KEY_POUND,
      NUM_OF_KEY,
}KeyPad;

static KeyPad input_code[NUM_OF_PASSCODE_CHAR] ={KEY_NULL, KEY_NULL,
                                                 KEY_NULL, KEY_NULL, KEY_NULL};

// password is 63263
static KeyPad pass_code[NUM_OF_PASSCODE_CHAR] = {KEY_6, KEY_3,
                                                 KEY_2, KEY_6, KEY_3};

/* pointer to current input character */
static uint8_t input_code_index = 0;

static uint8_t key_pressed = 0;
static uint8_t key_press_count[NUM_OF_KEY];

static struct timer_task touch_timer;
```

TOTO 4 – Local function

Double click on TODO 4 to jump to the code section.

Fill in the code below right after TODO line

```
/**
 *  @brief  Timer callback
 *
 *    @param      timer_task  timer invokes the callback
 *
 *    @return     NULL  always return
 *
 *    Calling touch_process() and handle the imput
 *    if touch measurement is finished
 *
 *    @date 01.06.2019 - initial
 *
 *  @bug No known bugs.
 */
static void TouchTimer_CB(const struct timer_task *const timer_task);

/**
 *  @brief  Print pass code on the screen
 *
 *    @param      NULL  no params
 *
 *    @return     NULL  always return
 *
 *    @date 01.06.2019 - initial
 *
 *  @bug No known bugs.
 */
static void PrintPassCode(void);

/**
 *  @brief  Verify the input from the keypad
 *
 *    @param      NULL  no params
 *
 *    @return     true  if the input and password are matching
 *
 *    @date 01.06.2019 - initial
 *
 *  @bug No known bugs.
 */
static bool VerifyPassCode(void);

/**
 *  @brief  Process the input key from the keypad
 *
 *    @param      NULL  no params
 *
 *    @return     NULL  always return
 *
 *    Check if the input is a number, the pound sign,
```

```
 *    #: verify the input
 *    C: clear the previous input
 *    other: next character of the pass code
 *
 *    @date 01.06.2019 - initial
 *
 *  @bug No known bugs.
 */
static void ProcessKey(KeyPad key);




 /**
  *  @brief  Handle the input from the keypad
  *
  *    @param      NULL  no params
  *
  *    @return     NULL  always return
  *
  *    Check which key is pressed and if the pressing action
  *    is over the threshold or not
  *    which indicates a successful press action
  *
  *    @date 01.06.2019 - initial
  *
  *  @bug No known bugs.
 */
 static void HandleKeypadInput(void);
```

Those functions take care of the processing the input from user and print the result on the terminal.

Please refer the function header to know in detail how each function works.

TODO 5 – initialize and start timer

Double click on TODO 5 to jump to the code section.

Fill in the code below right after TODO line. The snippet initializes the timer, which is in charge of polling for the measurement of the touch controller and handles the output of the touch controller.

```
/* Timer initialization */
touch_timer.interval = TOUCH_TIMER_MILI;
touch_timer.cb       = TouchTimer_CB;
touch_timer.mode     = TIMER_TASK_REPEAT;


timer_add_task(&Timer, &touch_timer);
timer_start(&Timer);
```

```
printf("\x1b[2J");
printf("Enter Password:\r\n");
printf("_ _ _ _ _\r\n");
```

TODO 6 – Local function body

Double click on TODO 6 to jump to the code section.

Fill in the code below right after TODO line. The snippet is the body of the local functions.

```
static void HandleKeypadInput(void)
{
      for(KeyPad i = KEY_1; i < NUM_OF_KEY; i++){
            key_pressed = get_sensor_state(i) & KEY_TOUCHED_MASK;
            if (0u != key_pressed) {
                  ++key_press_count[i];

                  if(key_press_count[i] >= KEY_PRESS_THRESHOLD){
                        key_press_count[i] = 0;
                        ProcessKey(i);
                  }
                  else{
                  }
            }
      }
}

static void TouchTimer_CB(const struct timer_task *const timer_task)
{

      touch_process();
      if (measurement_done_touch == 1) {
            HandleKeypadInput();
      }
}

static void PrintPassCode(void)
{
      printf("\x1b[2J");
      printf("Enter Password:\r\n");
      for(uint8_t i = 0; i < NUM_OF_PASSCODE_CHAR; i++)
      {
            if(input_code[i] == KEY_NULL)
                  printf("_ ");
            else
                  printf("* ");
      }
      printf("\r\n");
}

static bool VerifyPassCode(void)
{
      bool result = false;
      if(!memcmp(input_code, pass_code,NUM_OF_PASSCODE_CHAR)){
            result = true;
```

```
            return result;
      }
      return result;
}

static void ProcessKey(KeyPad key)
{
      if(key == KEY_C){
            if(input_code_index  > 0 )
            {
                  --input_code_index;
                  input_code[input_code_index] = KEY_NULL;
            }
            PrintPassCode();
      }
      else if (key == KEY_POUND){
            if(true != VerifyPassCode())
            printf("wrong password! \r\n");
            else
            printf("password correct\r\n");
      }
      else{
            if(input_code_index < NUM_OF_PASSCODE_CHAR){
                  input_code[input_code_index] = key;
                  ++input_code_index;
            }
            PrintPassCode();
      }
}
```

If time allows, please exploring each function.

Compile the project again.

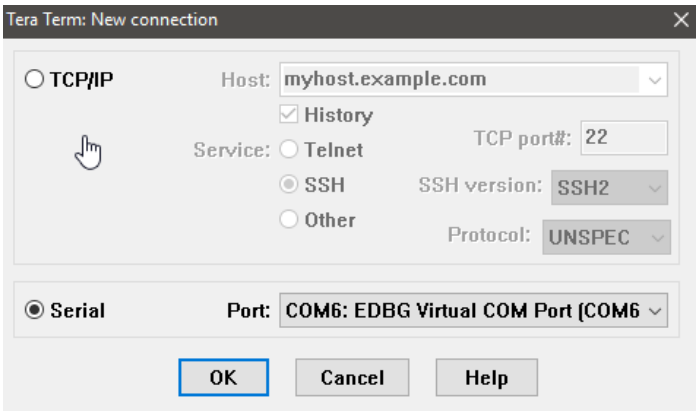
If there is no problem, the compilation output should be as following

```
Output
Show output from: Build
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "C:\Users\haing\Desktop\Project\SAML11_QTOUCH3_TZ\Secure Project\Secure Project.cproj" (entry point):
Done building target "Build" in project "Secure Project.cproj".
Done building project "Secure Project.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

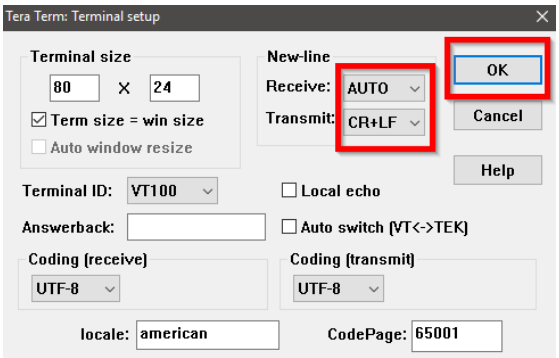
Setup TeraTerm

Before running the application, we need to setup TeraTerm.



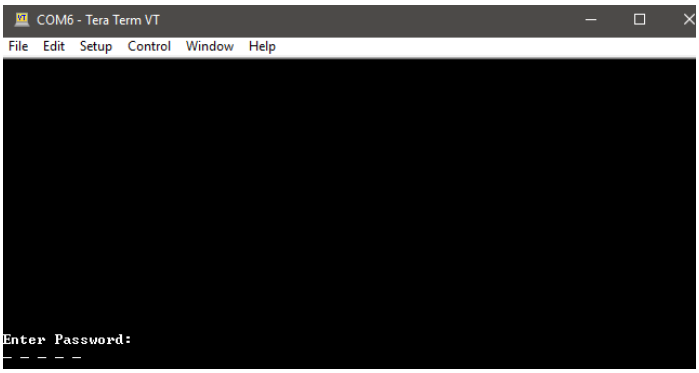
COM port may be different on your machine

Go to Setup → Terminal and set the following configuration



Press reset button on the SAML11 Xplained

If it is setup correctly, you should see the following output on the terminal



You can enter the passcode by pressing the key pad from 0 to 9, clear the previous number with key pad C and start the authenticate with key pad #

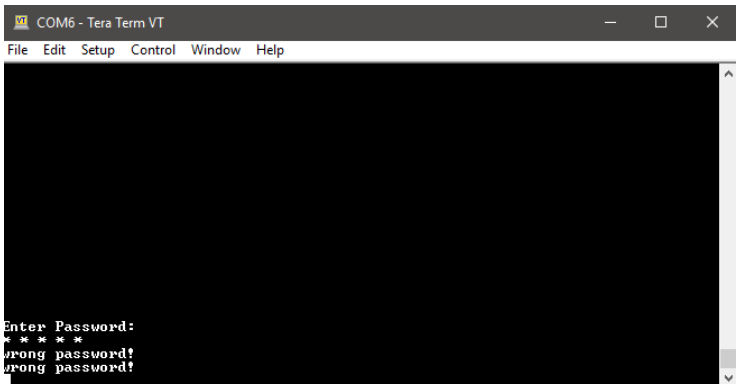


Figure 6: Enter wrong password

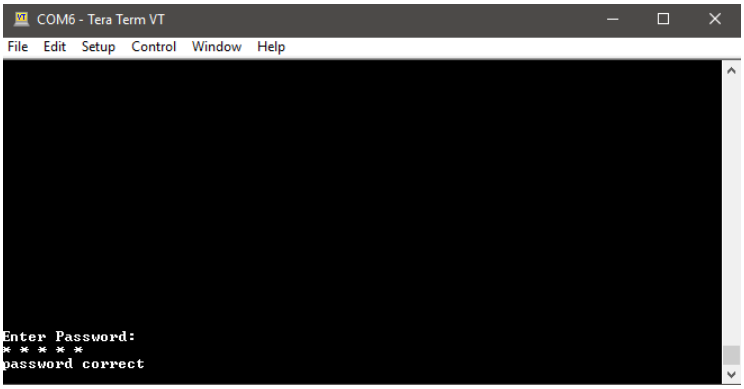


Figure 7: Enter correct password

The correct password is 63263

Please try to change the password in the code, compile, and run the project again

Congratulation! you have finished the hands-on!

The End