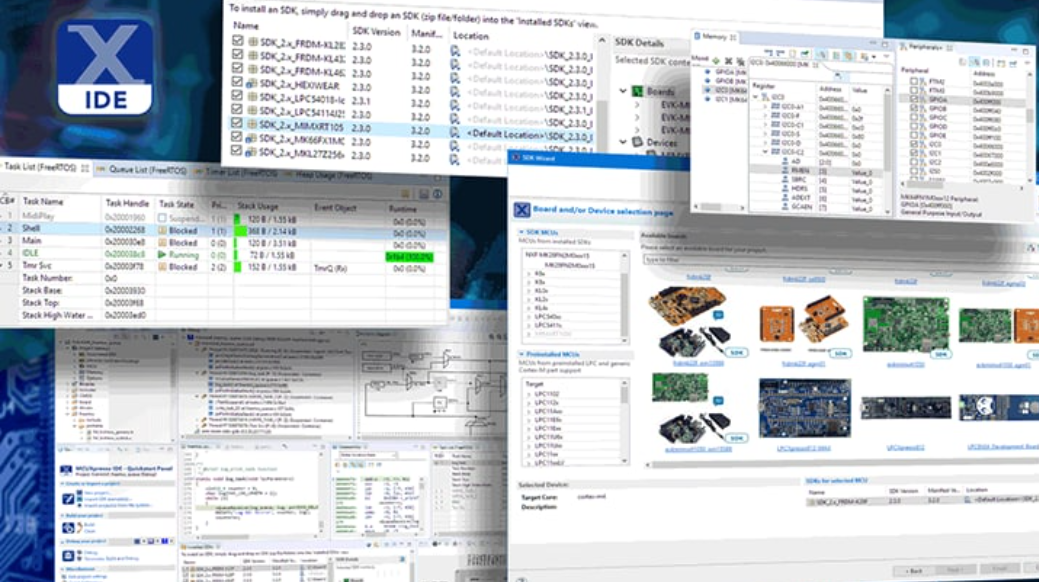
Lab Notes

Overview of the tools:

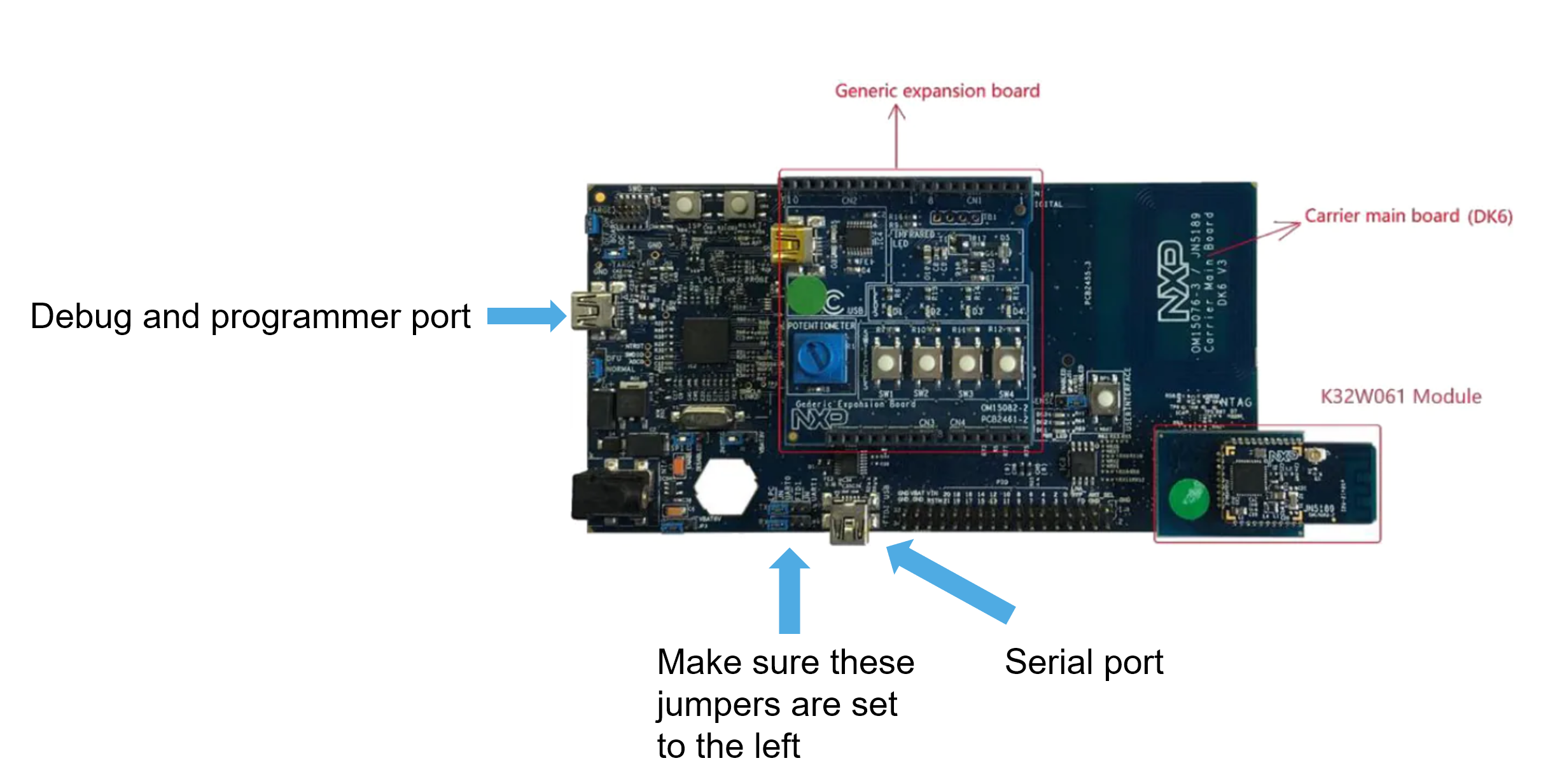
MCUXpresso

* + <https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE>
  + IDE used to flash and debug



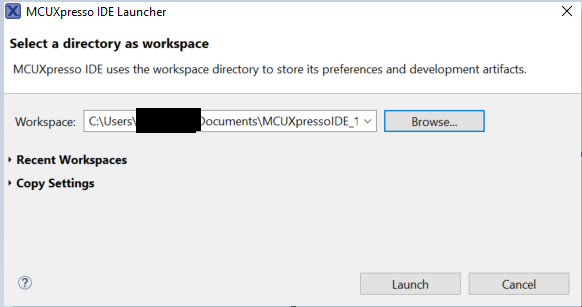
Board we will use:

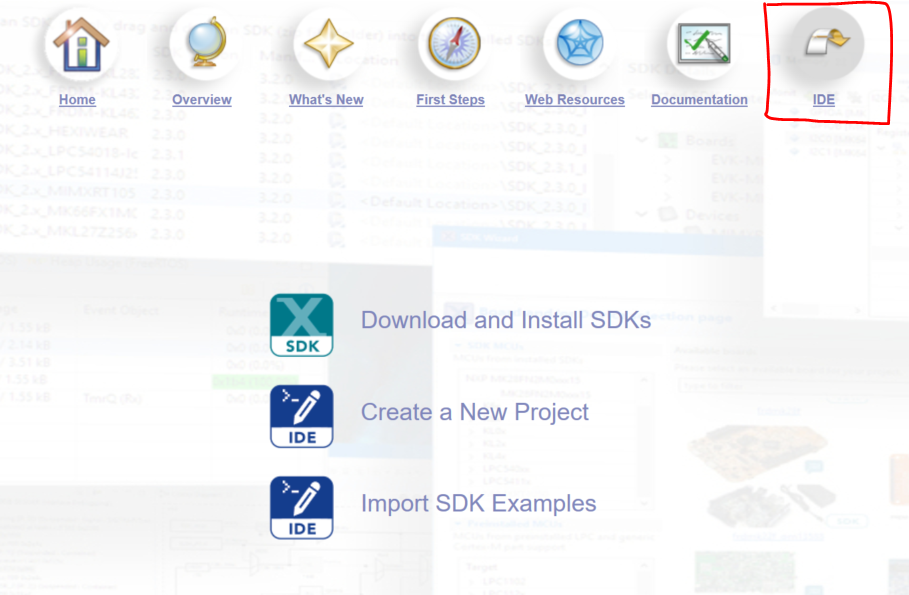
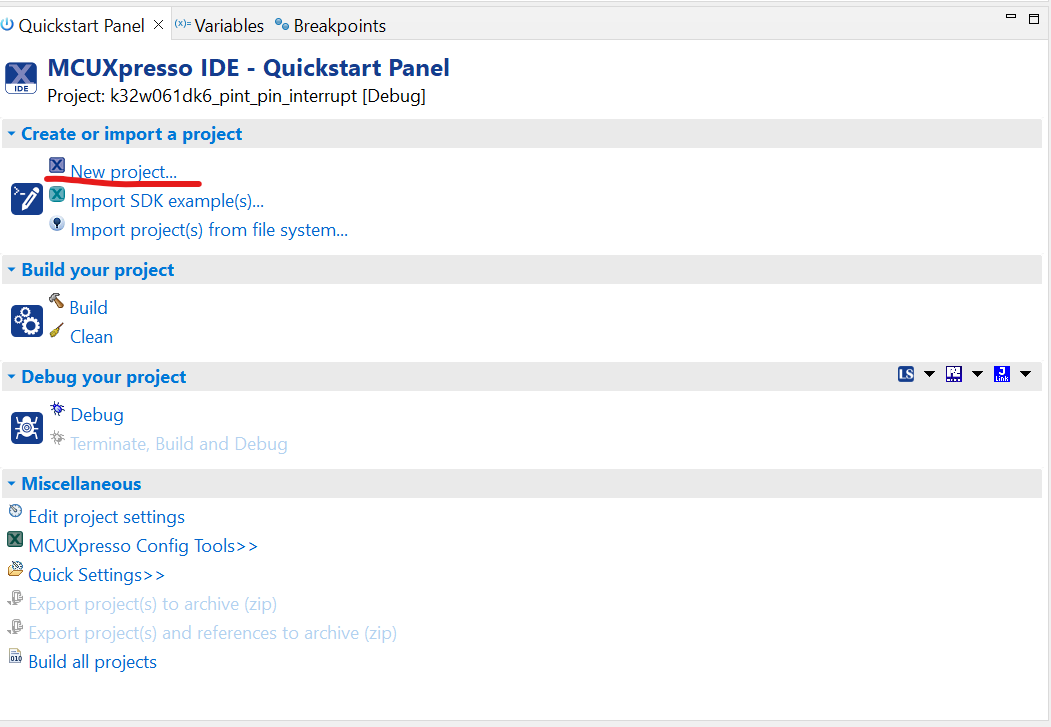
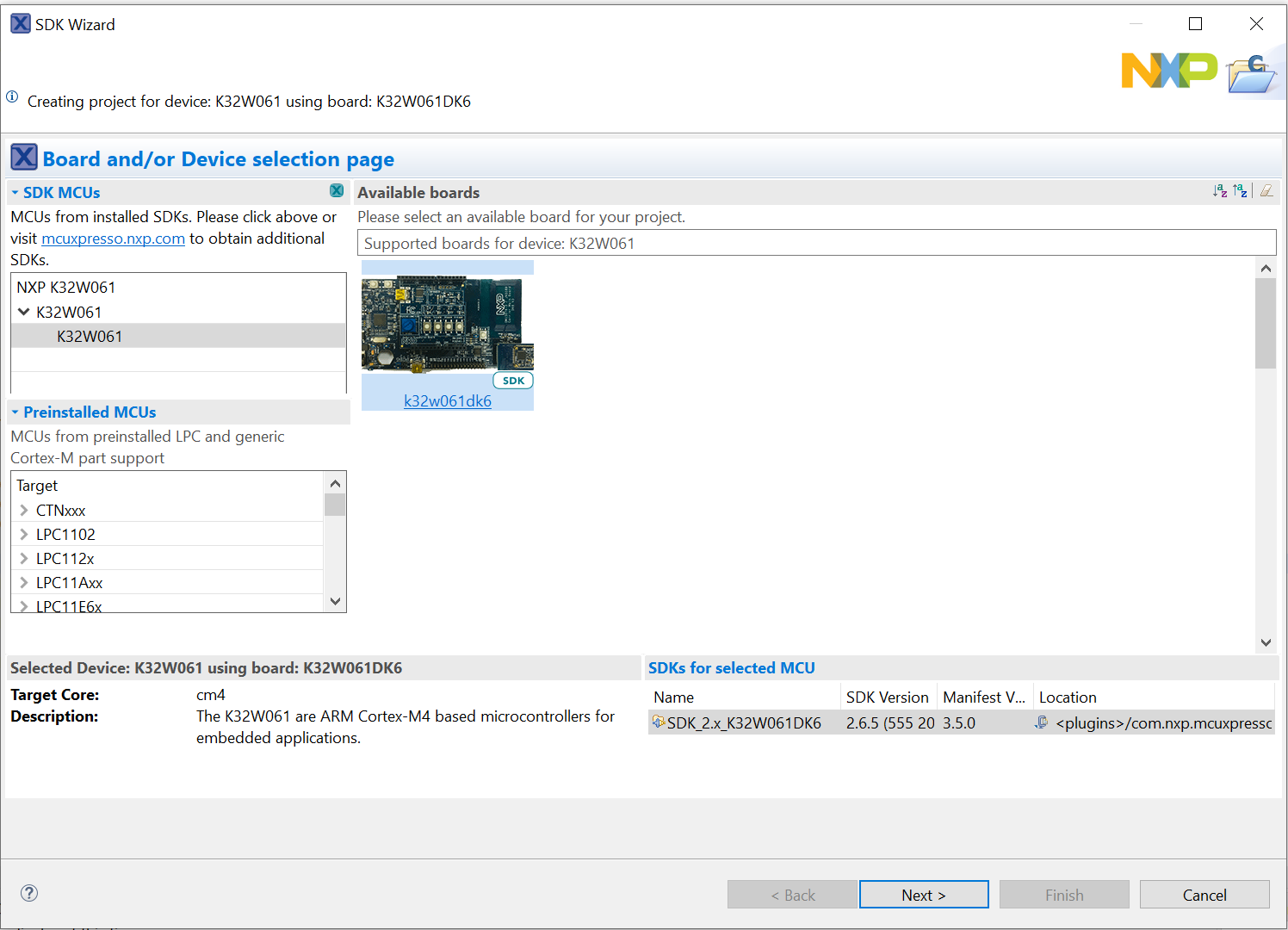
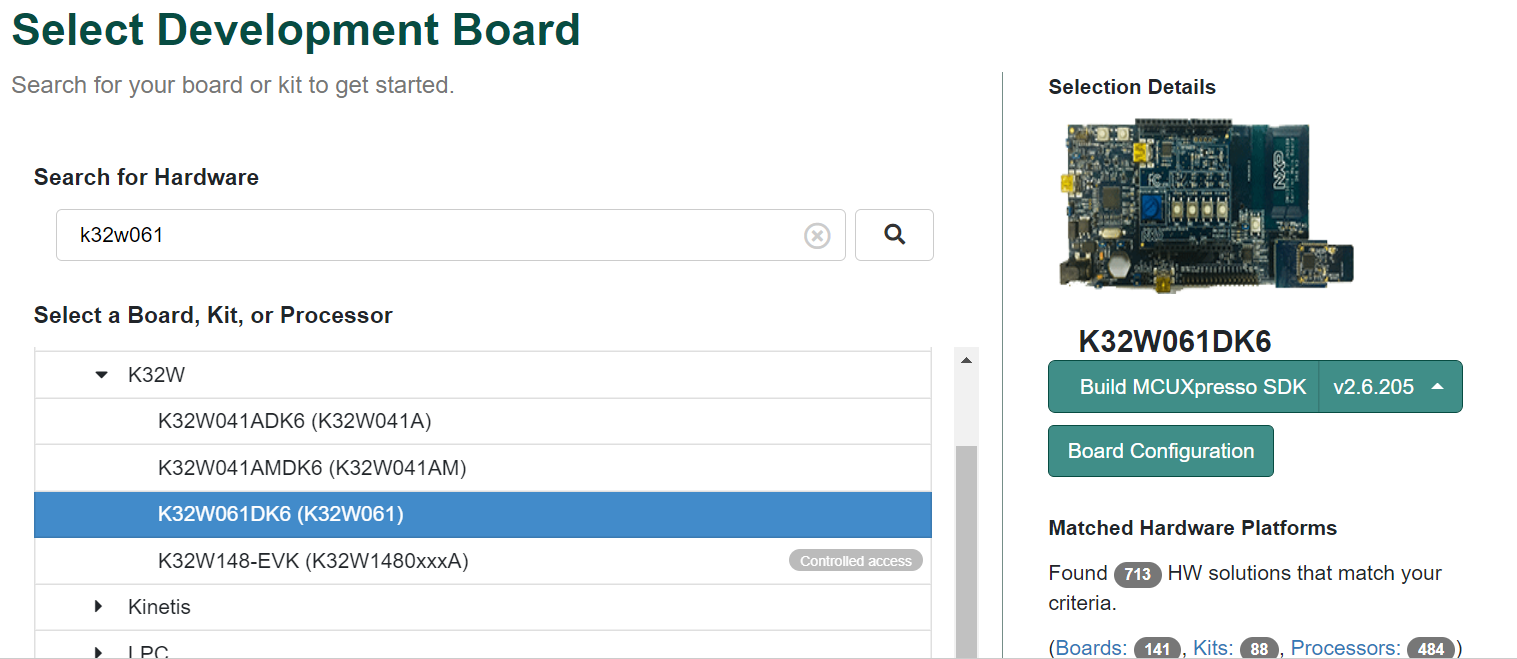
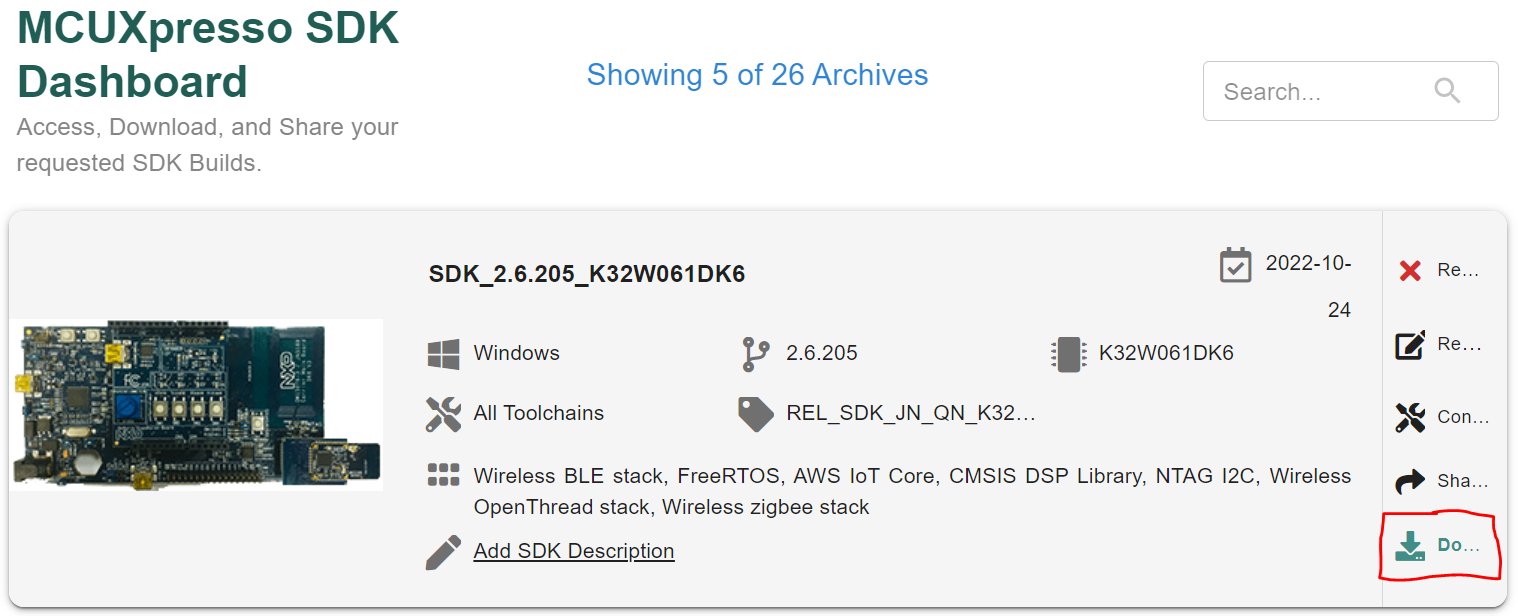
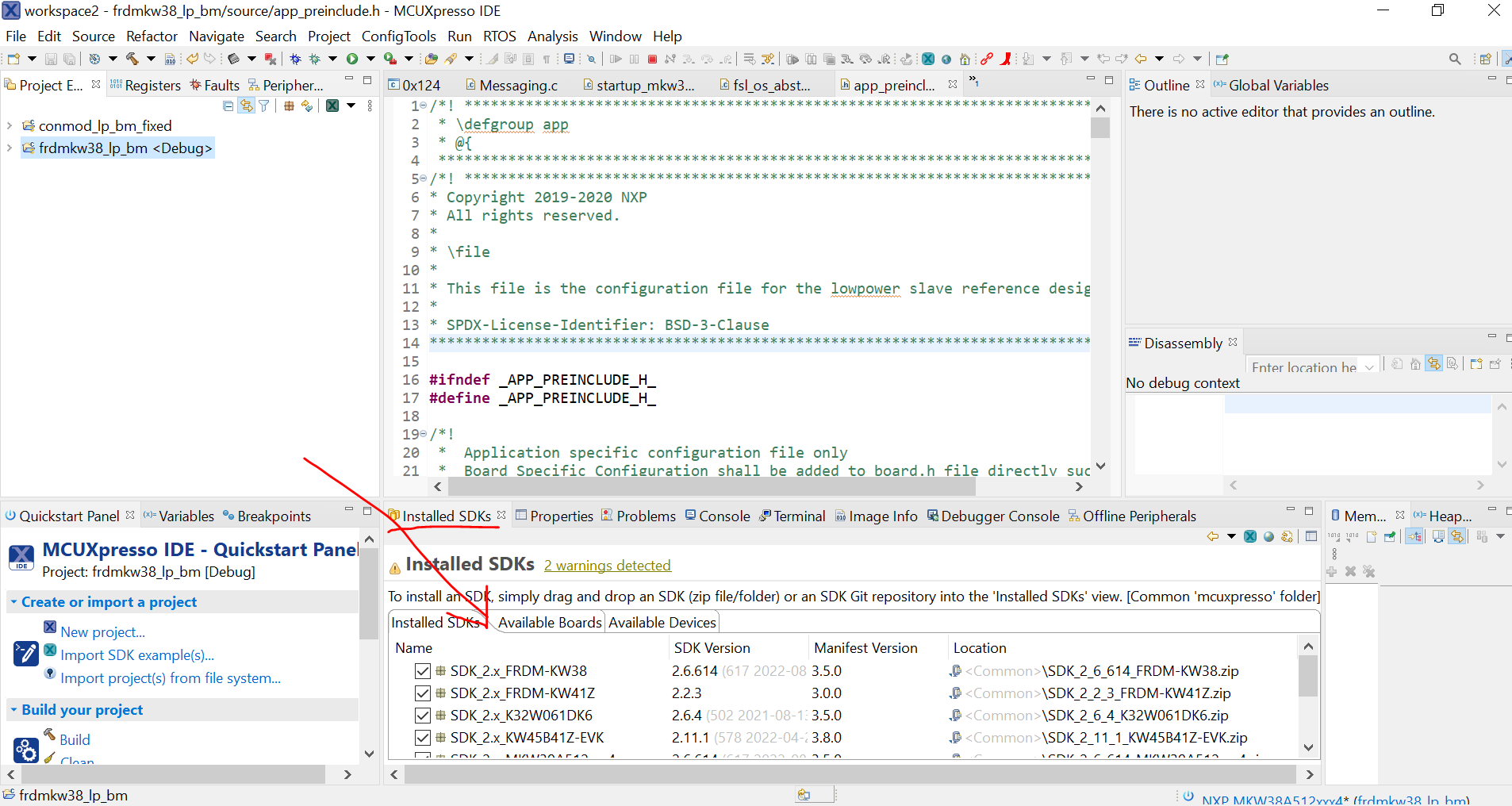
[**K32W061/41**](https://www.nxp.com/products/wireless/multiprotocol-mcus/k32w061-41-high-performance-secure-and-ultra-low-power-mcu-for-zigbeethread-and-bluetooth-le-5-0-with-built-in-nfc-option:K32W061_41)**: High-Performance, Secure and Ultra-Low-Power MCU for Zigbee®,Thread™, and Bluetooth® LE 5.0 with Built-In NFC Option**

****

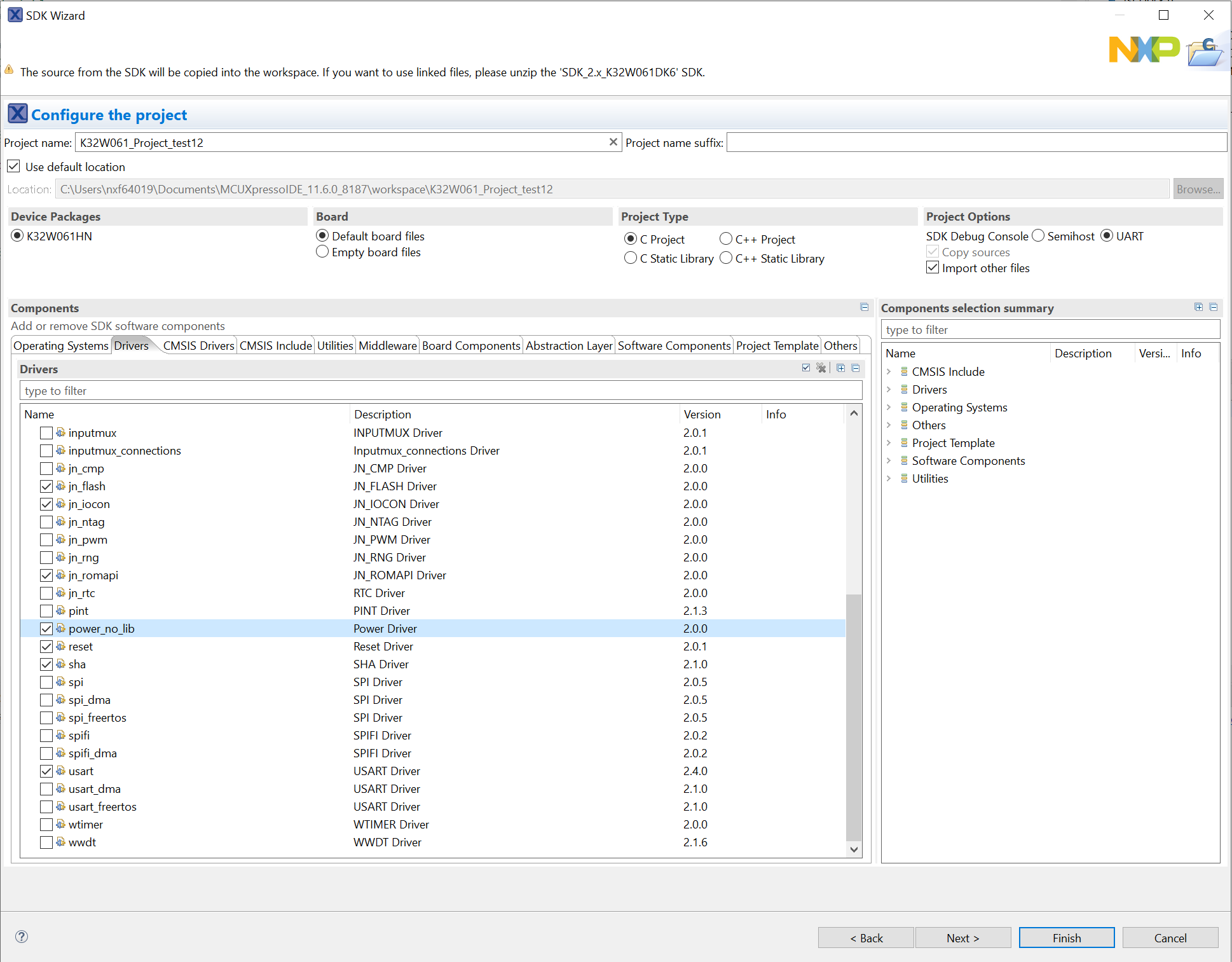
**Starting MCUXPRESSO**

1. **Set the working directory:**

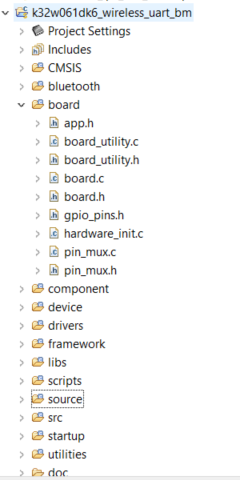
****

1. **Dismiss welcome screen**
2. **On the left bottom side:** ****
3. **Select our board:** ****
4. If the board is not in this dialog
   * Download the sdk from <https://mcuxpresso.nxp.com/en/welcome>
   * After creating an account click “Select Development Board” to access the tool
   * 
   * **Choose our board:** ****
   * **Press the download button and download the SDK and Documentation archive** ****
   * **Drag and drop the SDK archive in MCUXpresso in the designated space **
   * **Go back to step 4**
5. **Give the project a representative name and add the following check the following boxes from the Drivers tab:**
   * **aes**
   * **ctimer**
   * **sha**
   * **jn\_pwm**

**We will use this configuration for all exercises, please create different projects for each exercise**

****

**Project structure:**

****

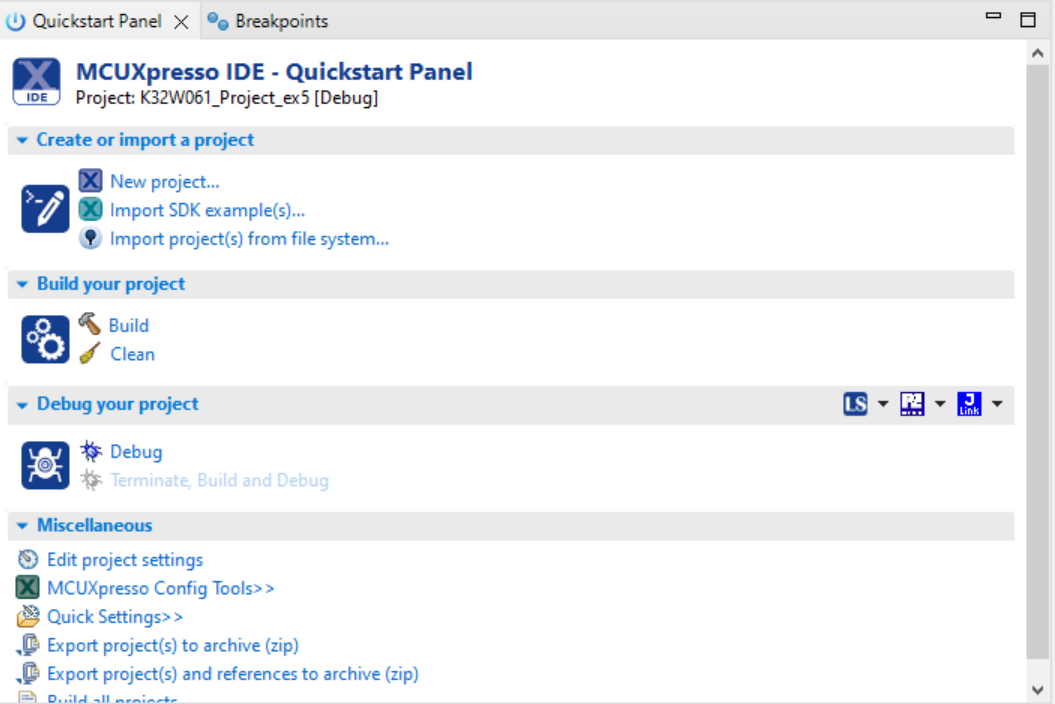
**Board folder contains the configurations for GPIO clock and pins of the board.**

**The drivers folder contains the drivers used in the examples.**

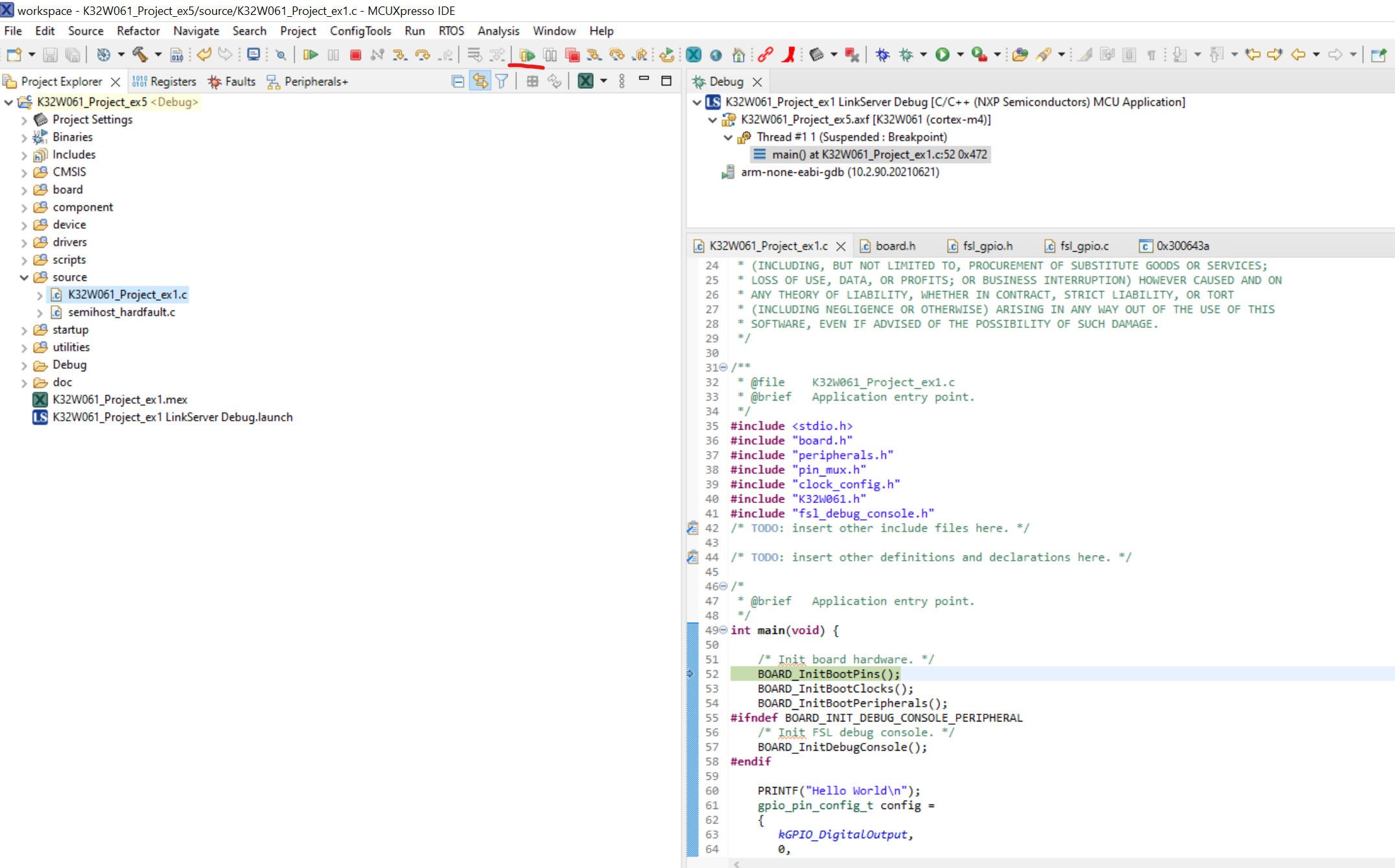
**The source folder contains our main c file.**

**Building and debugging**

If you have only one project in the workspace fastest way is to use the quickstart panel:

****

Wait for the project to load and you will get into the main program. Press resume to start the program:

****

Step over, step into and the Variables view will work as you would expect from any C IDE.

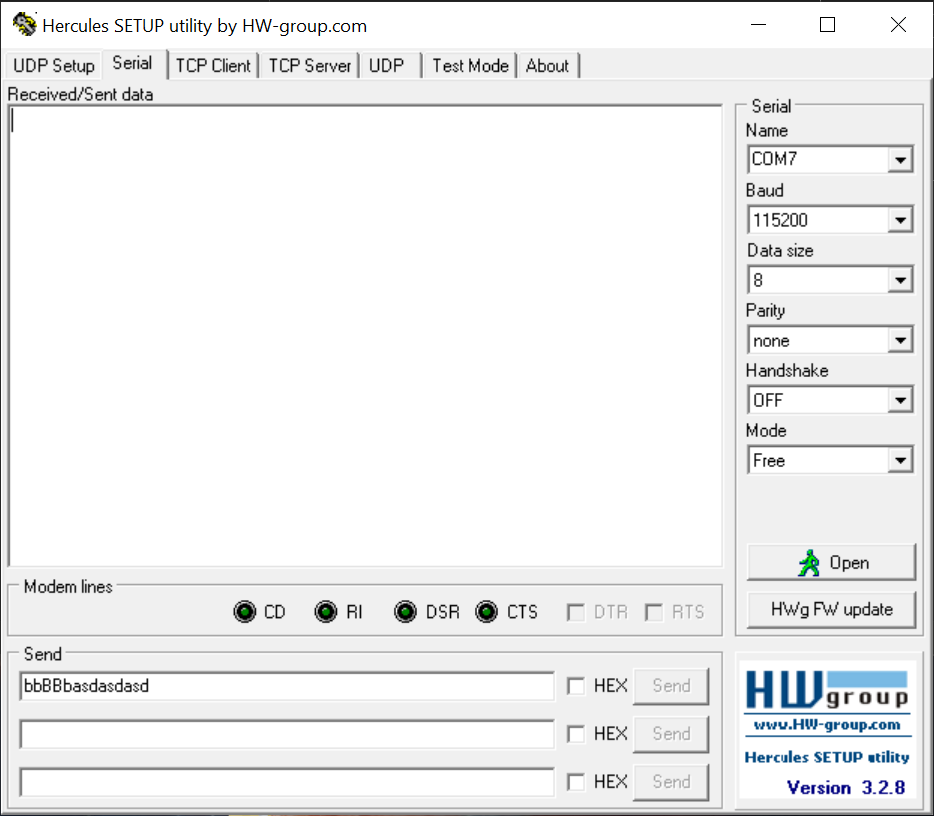
**Remember to TERMINATE the program before rebuilding, or triggering another debug session.**

**Hercules SETUP utility**

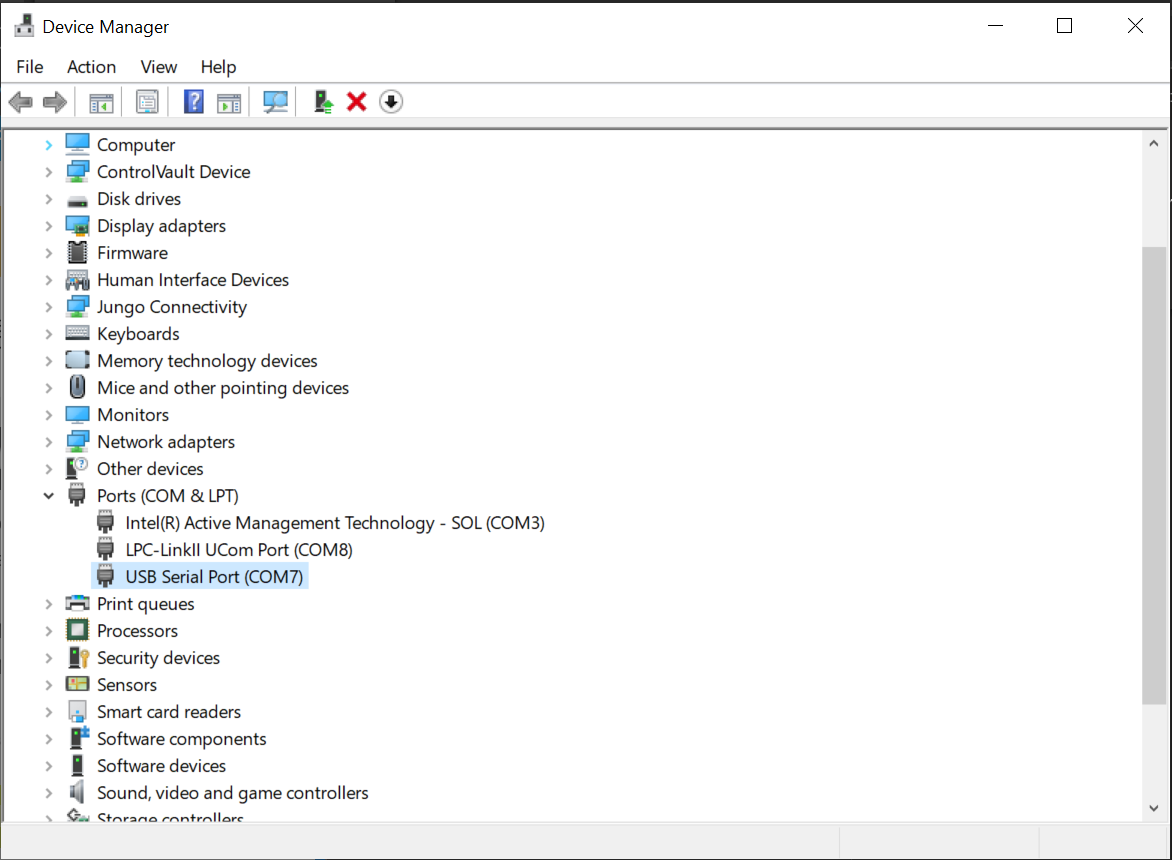
Hercules SETUP utility is a useful serial port terminal (RS-485 or RS-232 terminal).

Download it from: <https://www.hw-group.com/software/hercules-setup-utility>

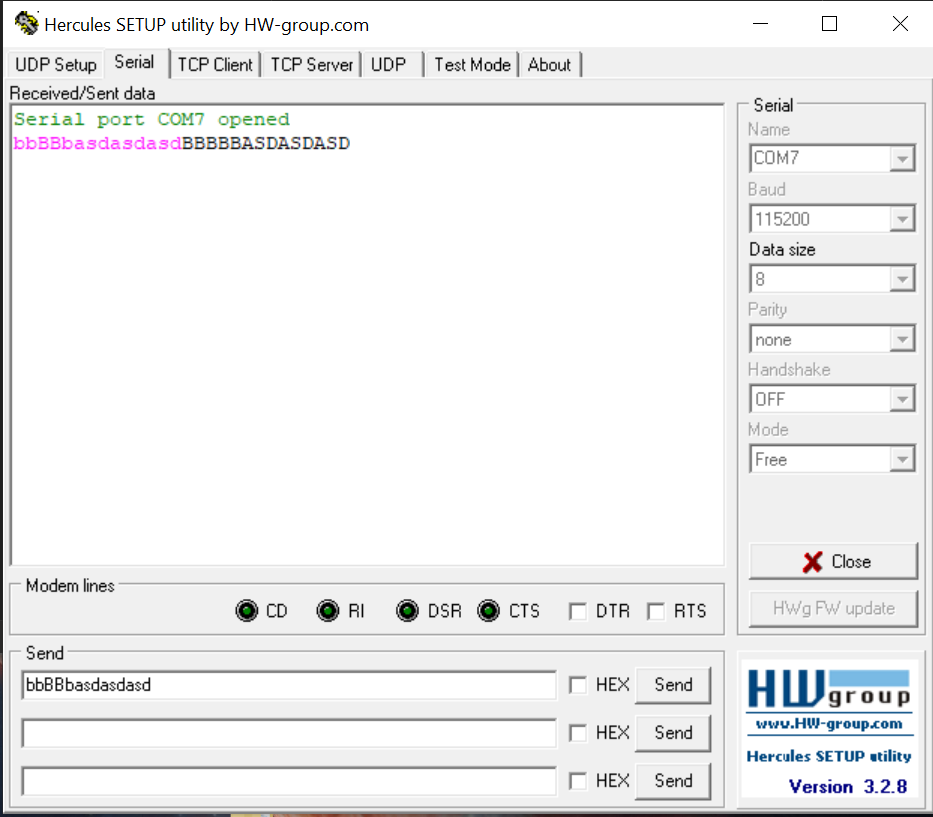
Use for Day 2+ labs for serial(UART) communication:

****

**Select the COM port and baud rate and click OPEN. To get the COM port of the device use the device manager from windows:**

****

**By unplugging and pluggin in the serial usb connection you can determine which COM port the board uses(see board picture from the first page to identify which USB cable the serial port uses).**

**The utility can be used to send ASCII data:**

* **By using one of the 3 input text boxes in the “Send” group**
* **In the Received/Send data the pink ASCII test is what is send, while the black is what is received**
* **Right clicking on the Received/Sent data text box can clear the data**
* **You can send hex data by checking the HEX checkbox in sent, for optimal viewing of hex data right clicking on the Received/Sent data text box will allow us to enabled HEX view**

**Lab 1 exercises**

1. **Blinky led**

**Use the function available in drivers/fsl\_gpio.h and the definitions available in board/board.h to create a led that blinks.**

**Note: this processor is running at 48Mhz we need to have a delay for to see the blinking. Use 0xFFFFF in an empty for loop to create a delay.**

**Use GPIO\_PinInit and GPIO\_PinWrite for this exercise.**

1. **Blinky led for both leds**

**We have two leds available on the board, adapt the code in such a way that the leds blink out of phase(when one is on the other is off).**

1. **Blinky leds with different blink rates**

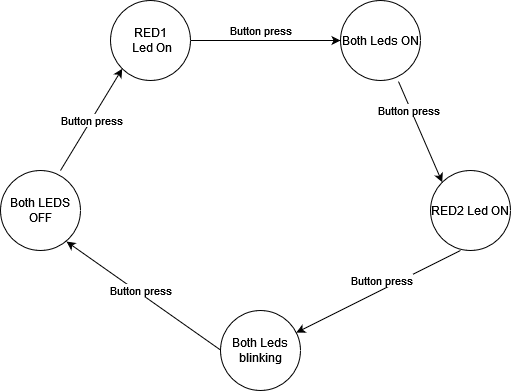
**Adapt the code such that we the two leds blink with different rates.**

1. **Read button and implement debounce**

**Read the button and implement a debounce scheme, use a led to see if the debounce works as expected: short press of button should do nothing, long press should turn the led on while button is pressed.**

**A value of 0 for the pin means that led is ON.**

1. **Finite state machine with different blink rates based on 5 states**



**Lab 2 exercises**

1. **Use the PWM driver to setup a 25% duty cycle on the led. The pin needs to be setup as a alternate function with:**

/\* Get the default source clock frequency \*/

CLOCK\_EnableClock(*kCLOCK\_Iocon*);

**const** uint32\_t port0\_pin3\_config = (/\* Pin is configured as PWM3 \*/

IOCON\_FUNC4 |

/\* Enables digital function \*/

IOCON\_PIO\_DIGITAL\_EN);

/\* PORT0 PIN3 (coords: 6) is configured as PWM3 \*/

IOCON\_PinMuxSet(IOCON, 0U, 3U, port0\_pin3\_config);

The PWM flow should be: PWM\_GetDefaultConfig, PWM\_Init, PWM\_SetupPwm and PWM\_StartTimer.

In the pwm\_setup\_t we need to set pol\_ctrl, period\_val and comp\_val.

For the period use the define USEC\_TO\_COUNT. What is this define doing?

pwmChan.period\_val = USEC\_TO\_COUNT(10000, pwmSourceClockFreq);

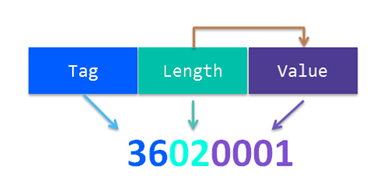
1. **Import the example for PWM. Answer the following questions:**
   1. **How is the pin configuration set up?**
   2. **When does the interrupt trigger?**
   3. **Modify the example to:**
      1. **Have the led go from dim to bright to dim and repeat**
      2. **Have the led go in increments of 25% duty cycle. This transition should be observable(seconds for each state)**

**3. Import the example for usart\_pooling. The example will echo any characters send via UART. Check how the example works with HERCULES. Use the code in the example as a starting point to integrate the serial functionality into a new project(the usart project does not have all the drivers we will need for future exercises).**

**a. Respond with all caps**

**b. Have a message to switch between to lower and to upper as response for message. The first byte of the message will make this decision: u for uppercase and l for lowercase. What happens if we send for example “uMult”**

**c. Create a TLV scheme as below to fix the issue in the above exercise:**

****

**In our main main loop receive 1 byte(the tag), use a switch statement to go to the selected function(upper or lowercase), receive the length part and then receive the number of bytes of the message(defined by length).**

**The structure is as follows:**

1. **Tag -> uint8\_t**
2. **Length -> uint8\_t**
3. **Value -> no of bytes defined by Length**
   1. **Increase the maximum data we can receive to 65k**
   2. **Add two more tags:**
      1. **z -> every even character is Uppercase,**
      2. **x-> every odd character is lowercase**

**Odd or even from the point of view or position in the input vector(first character is even(position 0) second character is odd(position 1) etc.**

* 1. **Add one more tag: b -> Remove all non printable characters from the input**

**Lab 3 exercises**

1. **Test the check\_prime function implementation (declaration available at: source\src\FUTs.h, implementation available at: source\src\check\_prime.c).**
   1. **Identify test scenarios.**
   2. **Implement a test function named test\_check\_prime using this source file: source\tests\test\_check\_prime.c. The test function should run multiple test scenarios and provide a descriptive return code and message (using error\_message) in case of failure. The Function Under Test (FUT) will be called inside this test function.**
2. **Test the memcopy\_n function implementations. To do this, you may need to implement some helper functions (since these can be used in multiple different testing scenarios and will make the testing code more readable).** 
   1. **Implement add\_fencing, check\_fencing and check\_result functions à inside source\test\_utils.c. A short description of these functionalities is provided inside source\test\_utils.h.**
   2. **Implement test\_memcopy. This test function should be able to receive a function reference as an input parameter and call any of the 3 memcopy functions: memcopy\_1, memcopy\_2, memcopy\_3, available at source\src\memcopy\_functions.c. These functions` implementations were obfuscated to create a “closed-box testing” experience.**
      1. **Suggestion:** use the test\_params buffer to make the test function more configurable (you can then send different input parameters to test different test vectors).
      2. Use Hercules to send data for: test selection, input parameters.

**Anything can be changed/updated inside this project BUT the check\_prime and memcopy\_n implementations.**

**Lab 4 exercises**

1. **Implement the Caesar cipher using a TLV enconding on UART. Have separate commands for encryption and decryption. Shift by 3 characters.**
2. **Optional: Implement the Caesar cipher with shift as an input. Use the first byte from value(the V in TLV) for the shift value(value can be only positive).**
3. **Implement the Kamasutra cipher Implement the Caesar cipher using a TLV enconding on UART. Have separate commands for encryption and decryption.**
4. **Implement the encryption scheme for a simple XOR cipher with padding and block chaining. Key is a 8 byte input(can be any random value). Message can be any length between 1 and 32 bytes. Message needs to be padded if length is not multiple of 8.**

**The XOR cipher algorithm is as follows:**

* **if message is not multiple of 8 we will pad the message with 0s until it is a multiple of 8**
* **We consider 8 bytes as the block of this cipher**
* **We xor the first block(8 bytes) of the padded message with the key**
* **If the padded message is longer than 1 block(8 bytes) for the subsequent n blocks the encryption will be the padded message xored with the key and xored with the ciphertext of the previous block**

**The encryption algorithm above in our case is still XOR**

**Key will be hardcoded in the program with:**

**uint8\_t key[]={0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};**

**For an input of 10 bytes:** {F0}{F0}{00}{00}{00}{00}{00}{00}{08}{08}

**We should receive:** {F1}{F2}{03}{04}{05}{06}{07}{08}{F8}{F8}{00}{00}{00}{00}{00}{00}