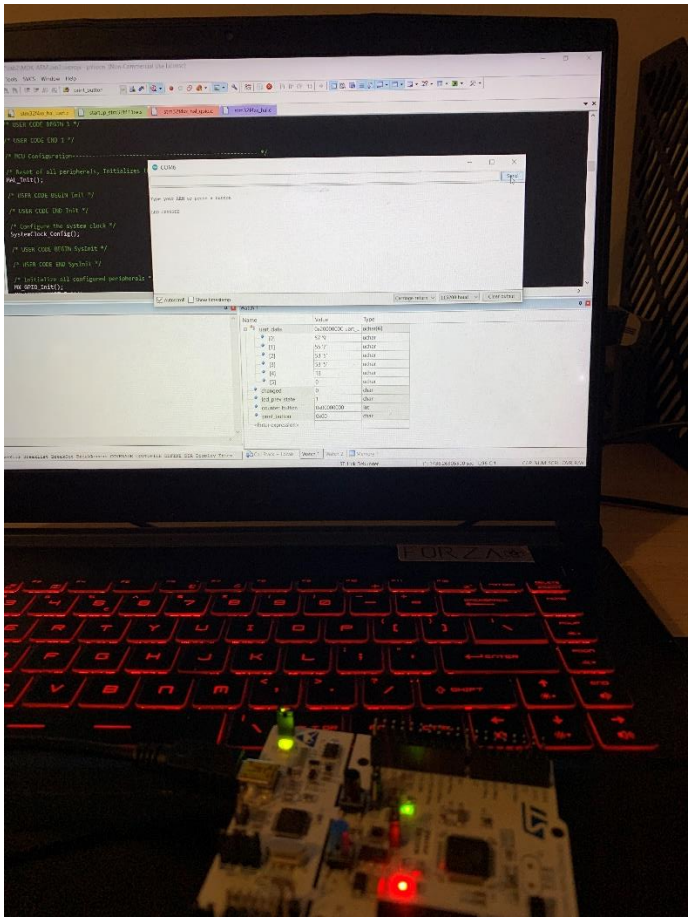
REPORT

March 2023

LAB 2

ΓΚΟΥΜΑ ΒΑΣΙΛΙΚΗ 9755 , ΚΩΣΤΑΣ ΑΝΔΡΟΝΙΚΟΣ 9754



Εικόνα 1 : Proof of Concept

- ## Εκφώνηση

Η παρούσα εργασία καλύπτει τον προγραμματισμό σε assembly ενός μικρό-ελεγκτή ARM με χρήση των εργαλείων Keil όπως σας έχουν παρουσιαστεί στο 2ο εργαστηριακό μάθημα. Στα πλαίσια της εργασίας θα γράψετε ένα πρόγραμμα, σε C, το οποίο διαμορφώνεται ως εξής:

α) Θα ζητάει το ΑΕΜ του χρήστη μέσω της σειριακής διεπαφής UART.

β) Σε περίπτωση που το ΑΕΜ τελειώνει σε μονό αριθμό θα καλεί μία ISR (Interrupt Service Routine) η οποία θα ανάβει το LED.

γ) Σε περίπτωση που το ΑΕΜ τελειώνει σε ζυγό αριθμό θα καλεί μία ISR (Interrupt Service Routine) η οποία θα απενεργοποιεί το LED.

δ) Σε περίπτωση που πατηθεί ο διακόπτης, σε οποιαδήποτε φάση, θα καλεί μία ISR (Interrupt Service Routine) η οποία θα ανάβει το LED και θα εκτυπώνει πόσες φορές έχει πατηθεί ο διακόπτης.

ε) Σε περίπτωση που πατηθεί ο διακόπτης, και το LED είναι ήδη αναμμένο, θα καλεί μία ISR (Interrupt Service Routine) η οποία θα απενεργοποιεί το LED και θα εκτυπώνει πόσες φορές έχει πατηθεί ο διακόπτης.

ζ) Κάθε αλλαγή της κατάστασης του LED, θα πρέπει να εκτυπώνεται μέσω UART

Προτείνεται στο Keil να επιλέξετε τον μικρο-ελεγκτή NUCLEO M4 που σας έχει υποδειχθεί (και το ανάλογο Board) και περιγράφεται αναλυτικά και στο υλικό που έχει αναρτηθεί στο elearning.

***Σημείωση:** *Σε περίπτωση που πατηθεί το κουμπί ενώ ο χρήστης καταχωρεί το input του θα πρέπει να έχει προτεραιότητα η είσοδος από το διακόπτη*

Εικόνα 2 : Εκφώνηση Εργαστηρίου

- ## Σύντομη Περιγραφή

Για την υλοποίηση του εργαστηρίου χρησιμοποιήσαμε αφενός το IDE της ARM, Keil αλλά και το STM32CubeMX που είναι ένα βοηθητικό εργαλείο κατάλληλο για τις αρχικοποιήσεις του μικροελεγκτή όπως για παράδειγμα ποιο να είναι το clock του επεξεργαστή, ποιο το baud rate για την UART ζεύξη κ.λ.π. Όλες οι συναρτήσεις που χρησιμοποιήθηκαν είναι από τους drivers της ARM. Για παράδειγμα, για το UART χρησιμοποιήσαμε τα αρχεία stm32f44xx_hal_uart.c και .h όπου εκεί βρήκαμε τις συναρτήσεις για λήψη και αποστολή δεδομένων σε non-blocking mode όπως και τις Callback Functions που χρησιμοποιούνται όταν καλείται μια ISR.
Ο χρήστης μπορεί να δώσει ΑΕΜ με 4 ή 5 ψηφία. Ο διαχωρισμός για το αν έχουμε 4 ή 5 ψηφία γίνεται μέσω της μεταβλητής ΑΕΜ. Αν το ΑΕΜ είναι μικρότερο ή ίσο από 9999 ελέγχουμε το στοιχείο uart_data[3] αν είναι μονός ή ζυγός αλλιώς ελέγχουμε το uart_data[4].

- ## Δήλωση Μεταβλητών και Πινάκων

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "stdbool.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

// uint8_t array to store the AEM of the user //
//
uint8_t uart_data[6];
// counter to count how many times the button is pressed //
int counter_button = 0;
// character array to pack how many times the button pressed and sent it via UART //
char counter_button_string[2];
  // flag that is being changed whether the Button ISR is called //
bool print_button = 0;
// changed = 1 if the state of the LED is changed //
bool changed = 0;
// variable to hold the previous state of the LED to define if the changed variable must be 1 or 0 //
// led_prev_state = 1 means LED is turned ON //
bool led_prev_state = 0;

int AEM = 0;
```

Εικόνα 3 : Αρχικοποιήσεις Μεταβλητών

- ## Πρώτη επικοινωνία με τον χρήστη μέσω UART

```
// maximum lengh of AEM is 5 digits...we store 6 bytes cause each time we press enter another character is sent via UART //
// here we have the MCU ready to get data via UART from the user in non-blocking mode //
HAL_UART_Receive_IT(&huart2, uart_data, 6);
// ask the user its AEM //
do{
  HAL_UART_Transmit(&huart2, (uint8_t *)"\nType your AEM or press a button\n", strlen("\nType your AEM or press a button\n"), 10);
  HAL_Delay(1500);
}while(counter_button == 0 && uart_data[4] == 0); // if one of the two expressions are not satisfied then the loop code stops because only 1 AND 1 = 1

/* USER CODE END 2 */
```

Εικόνα 4 : 1η Αλληλεπίδραση με τον Χρήστη

Παρατηρήσεις :

- Αρχικά, θέτουμε τον πίνακα *uart_data* ως τον πίνακα που θα λάβει τα δεδομένα του χρήστη. Η συγκεκριμένη συνάρτηση είναι σε λειτουργία non-blocking mode οπότε μετά την εκτέλεση της ο επεξεργαστής δεν χρειάζεται συνεχώς να κοιτάει τον πίνακα αν έχει δεδομένα αλλά θα τον ενημερώσει το UART module με την χρήση ενός interrupt και το processing των δεδομένων θα γίνει στην κατάλληλη callback function που θα δούμε παρακάτω.
- Το 3ο όρισμα που είναι 5 σημαίνει 5 bytes και αυτό επιλέχθηκε καθώς θεωρούμε ΑΕΜ με 4 ψηφία και 1 byte για να αποθηκεύσουμε τον χαρακτήρα που στέλνεται όταν πατάμε ENTER ως χρήστης.
- Το huart2 είναι μια μεταβλητή τύπου UART_HandleTypeDef το οποίο είναι ένα struct που περιέχει απαραίτητα δεδομένα για την λειτουργία των συναρτήσεων UART.
- Όσον αφορά την do-while, στέλνουμε ένα μήνυμα στο χρήστη έως ότου δώσει το ΑΕΜ του ή πατήσει το κουμπί. Ο κώδικας βγαίνει από την επανάληψη μόλις μία από τις εκφράσεις της while σταματήσει να ισχύει δηλαδή γίνει 0. Αυτό υλοποιείται με την AND.

- ## Συναρτήσεις ISR

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  // check from which UART module came the Interrupt
  if(huart -> Instance == USART2)
  {
    // if the AEM is even
    sscanf((char *)uart_data, "%d", &AEM);
    if(AEM <= 9999)
    {
        if(uart_data[3] % 2 == 0)
        {
          if(led_prev_state == 1)
          {
              changed = 1;
              led_prev_state = 0;
          }
        HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_RESET);
        }
        else{
            if(led_prev_state == 0)
            {
              changed = 1;
              led_prev_state = 1;
            }
            HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_SET);
        }
    }
    else if(AEM > 9999)
    {
      if(uart_data[4] % 2 == 0)
        {
          if(led_prev_state == 1)
          {
              changed = 1;
              led_prev_state = 0;
          }
        HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_RESET);
        }
        else{
            if(led_prev_state == 0)
            {
              changed = 1;
              led_prev_state = 1;
            }
            HAL_GPIO_WritePin(USER_LED_GPIO_Port, USER_LED_Pin, GPIO_PIN_SET);
        }
    }
  }

  HAL_UART_Receive_IT(&huart2, uart_data, 6);
```

Εικόνα 5 : UART_ISR

Παρατηρήσεις :

- Η UART_ISR καλείται αυτόματα κάθε φορά που υπάρχει πακέτο να διαβαστεί από το UART module δηλαδή κάθε φορά που ο χρήστης γράφει στην σειριακή κονσόλα.
- Με το που γίνει η κλήση της τα δεδομένα έχουν ήδη αποθηκευτεί στον πίνακα uart_data οπότε μπορούμε να τα επεξεργαστούμε σύμφωνα με την εκφώνηση.
- Αν το ΑΕΜ είναι ζυγός θέσε το USER_LED στο LOW, αλλιώς θέσε το στο HIGH.
- Η μεταβλητές changed και led_prev_state λειτουργούν παράλληλα για να ικανοποιείται το ζ) δηλαδή να τυπώνεται κάθε αλλαγή κατάστασης του LED. Η τύπωση αυτή όπως θα δούμε γίνεται στην main() καθώς δεν μπορούσε να πραγματοποιηθεί μέσα στην UART_ISR. Αυτός εξάλλου ήταν και ο λόγος δημιουργίας του "flag" changed.
- Η κλήση της HAL_UART_Receive_IT() στο τέλος της UART_ISR εξασφαλίζει ότι θα μπορούν να αποθηκευτούν εκ νέου δεδομένα αν ο χρήστης θέλει να πληκτρολογήσει πάλι το ΑΕΜ του. Στην ουσία κάνει κυκλικό το πρόγραμμα.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    HAL_GPIO_TogglePin(USER_LED_GPIO_Port, USER_LED_Pin);
    changed = 1;
    print_button = 1;
    counter_button += 1;
}
```

Εικόνα 6 : Button_ISR

Παρατηρήσεις :

- Η Button_ISR καλείται αυτόματα κάθε φορά που ο χρήστης πατάει το USER_BUTTON που βρίσκεται στον pin PC13.
- Χρησιμοποιούμε την togglePin() που αλλάζει το state του LED όποιο και αν είναι αυτό.
- print_button είναι ένα flag που θα το χρησιμοποιήσουμε στην main() για να τυπώνουμε τον μετρητή counter_button. Αυτό διότι η sprintf() μηδένιζε τον counter_button όταν έφτανε 10 όποτε μεταφέραμε όλο τον υπόλοιπο κώδικα της μετατροπής της τύπωσης των πόσο φορών πάτησε ο χρήστης το κουμπί στην main όπως θα δούμε και παρακάτω.

- **Main ()**

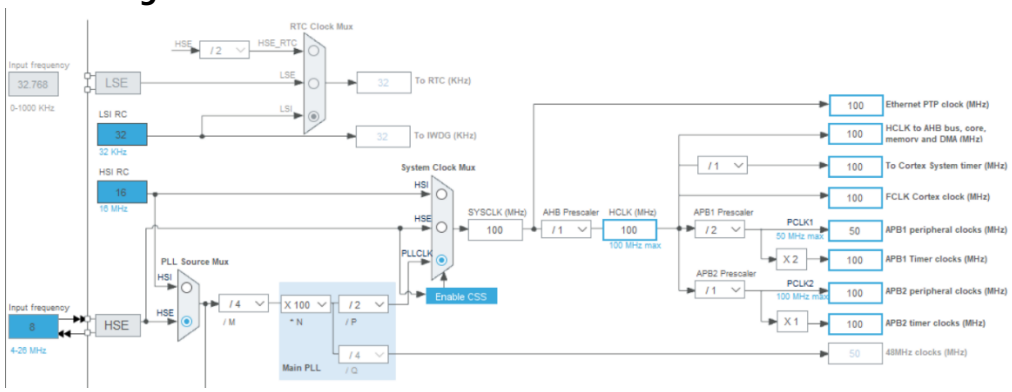
```
  // if the LED changed its state just printed //
  if(changed == 1)
  {
    // 12 characters must be sent in 1/(115200/(12*8)) = 0.8333 ms . So we put 5ms as a timeout
    HAL_UART_Transmit(&huart2, (uint8_t *)"\nLED CHANGED\n", strlen("\nLED CHANGED\n"), 5);
    changed = 0;
  }
  if(print_button == 1)
  {
    print_button = 0;
    HAL_UART_Transmit(&huart2, (uint8_t *)"\nTimes pressed: ", strlen("\nTimes pressed: "), 5);
  // convert integer to string in order to be sent via UART
    sprintf(counter_button_string, "%d", counter_button);
    HAL_UART_Transmit(&huart2, (uint8_t *)counter_button_string, 2, 1);
    HAL_UART_Transmit(&huart2, (uint8_t *)"\n", strlen("\n"), 1);
  }
}
/* USER CODE END 3 */
```

Εικόνα 7 : main()

Παρατηρήσεις :

- Τα flags changed και print_button ενεργοποιούνται (γίνονται 1) μέσα στις αντίστοιχες ISR όπως είδαμε παραπάνω

- **STM32CubeMX**
  - **Clock Configuration**

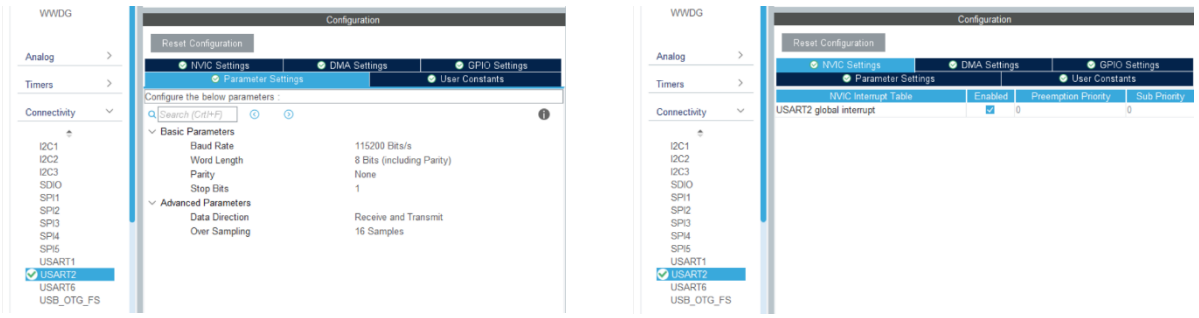


Εικόνα 8 : Clock Configuration

Παρατηρήσεις :

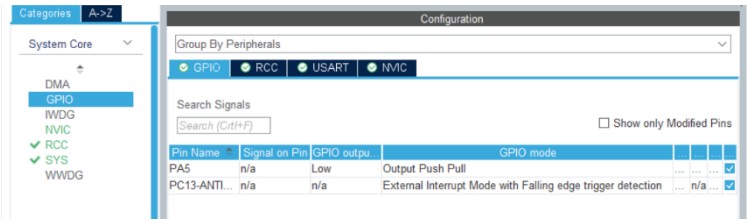
- Θέτοντας Input Frequency = 8 MHzείναι ο εξωτερικός crystal oscillator που είναι κολλημένος στο Nucleo επιλέγοντας HSE (High Speed Clock), PLLCLK και γράφοντας 100 στο κουτάκι του HCLK ο solver ψάχνει και τελικά βρίσκει λύση για όλα τα υπόλοιπα κουτάκια.

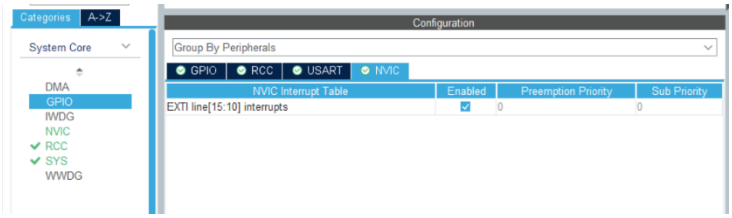
  - **UART Configuration**

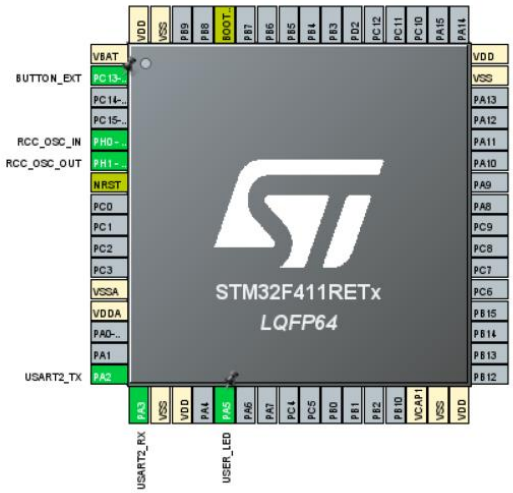


Εικόνες 9,10 : UART Configuration

- **PIN Configuration**

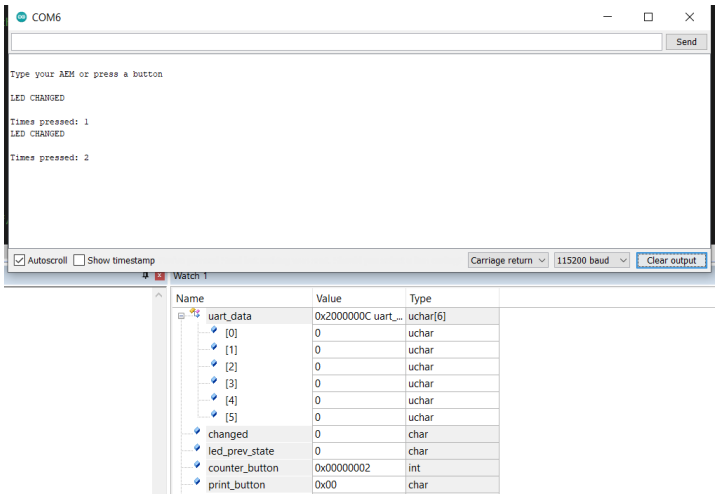




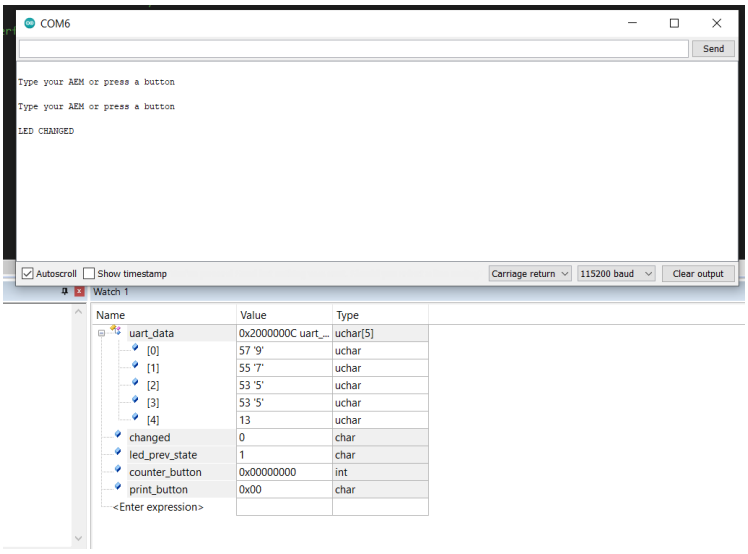


Εικόνες 11,12,13 : Pin Configuration

- **Results**



Εικόνα 14 : Button_ISR functionality



Εικόνα 15 : UART_ISR functionality