Embedded and ambient systems 2020.11.18.

Practice 5
Development of UART communications: even more sophisticated approach

Méréstechnika és Információs Rendszerek Tanszék

Optimization if IT

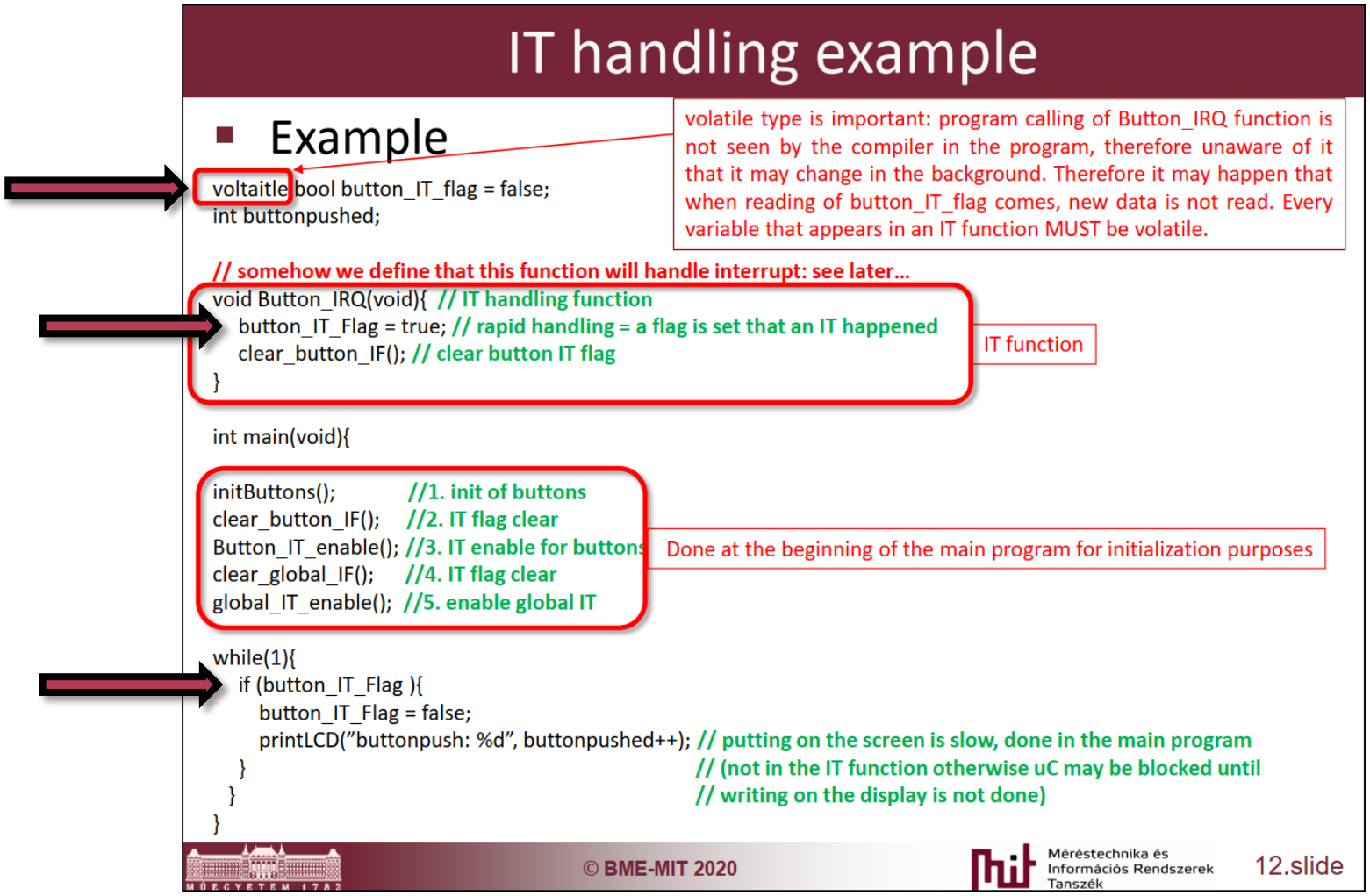
- Recall IT topic from lecture:

IT handling for a peripheral

- Execution of IT, the proposed way:
  - Rapid execution inside a function
  - Larger tasks are handled outside the function
  - When a peripheral is used more than only once in the program mutual exclusion has to be assured (e.g. when part of our code sends data using UART, and data is also sent in an IT routine, then the two data can be messed up)
    - It is advised to use flags for such kind of tasks, and execute them in the main program it they are not time-critical
  - Tightly connected tasks for IT handling (take care of the order!):
    - Peripheral handling (e.g. data of UART must be read or GPIO state must be read, etc.),
    - Clearing the corresponding IT flag (not needed when done automatically, but better done twice than never),
    - Enabling the IT (done automatically in most cases)

Optimization if IT

- Recall IT topic from lecture:



Optimization if IT

- In our IT function three functions are used:

```
uint8_t rx_data; // a variable is defined for USART_Tx data
                 // (whose type is uint8_t)
void UART0_RX_IRQHandler(void){ // definition of IT function:
    rx_data=USART_Rx(UART0);    // data read from UART0 and stored in rx_data
    USART_Tx(UART0, rx_data);   // rx_data is sent to UART0
    USART_IntClear(UART0, _USART_IFC_MASK); // IT flag is cleared
}
```

  - An IT function has to be rapid not to block other ITs and the running of the main program for a long time
  - Those tasks that are not necessary in the IT routine should be moved into the main program
  - Time critical tasks can remain in the IT routine
  - What functions can be moved into the main program?

Optimization if IT

- In our IT function three functions are used:

```
uint8_t rx_data; // a variable is defined for USART_Tx data
                 // (whose type is uint8_t)
void UART0_RX_IRQHandler(void){ // definition of IT function:
    rx_data=USART_Rx(UART0);     // data read from UART0 and stored in rx_data
    USART_Tx(UART0, rx_data);    // rx_data is sent to UART0
    USART_IntClear(UART0, _USART_IFC_MASK); // IT flag is cleared
}
```

  - What functions can be moved into the main program?
    - USART_IntClear() is obviously needed in the IT function
    - USART_Rx() seems that can be moved into the main program but cannot be moved since data can only be read in IT routine otherwise data remains in RxData buffer that generates a new IT immediately when IT routine is left

      -> we stuck in the IT
    - USART_Tx() really can be and advised to be moved into the main program since it is a blocking function and in case of large amount of data can block e.g. other interrupts

Optimization if IT

- Previous solution for IT routine:

```
uint8_t rx_data; // a variable is defined for USART_Tx data
                 // (whose type is uint8_t)
void UART0_RX_IRQHandler(void){ // definition of IT function:
    rx_data=USART_Rx(UART0);     // data read from UART0 and stored in rx_data
    USART_Tx(UART0, rx_data);    // rx_data is sent to UART0
    USART_IntClear(UART0, _USART_IFC_MASK); // IT flag is cleared
}
```

- Advanced solution for IT routine:

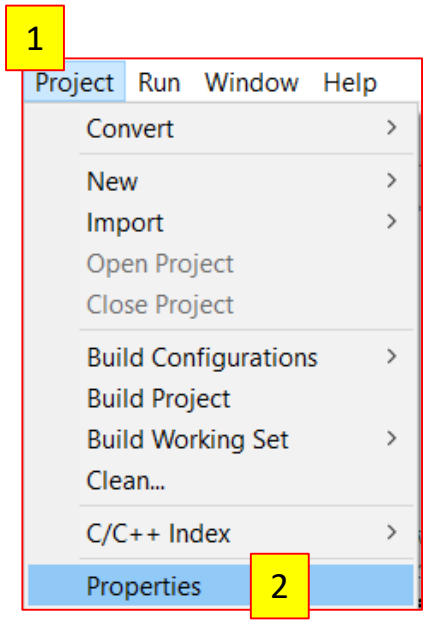
```
_Bool rx_flag = false; //a flag is defined to indicate IT
uint8_t rx_data; // a variable is defined for USART_Tx data
                 // (whose type is uint8_t)
void UART0_RX_IRQHandler(void){ // definition of IT function:
    rx_flag = true;                 // IT occurred->flag becomes true
    rx_data = USART_Rx(UART0);      // data read from UART0 and stored in rx_data
    //USART_Tx(UART0, rx_data);     // USART_Tx is moved into main program
    USART_IntClear(UART0, _USART_IFC_MASK); // IT flag is cleared
}
```

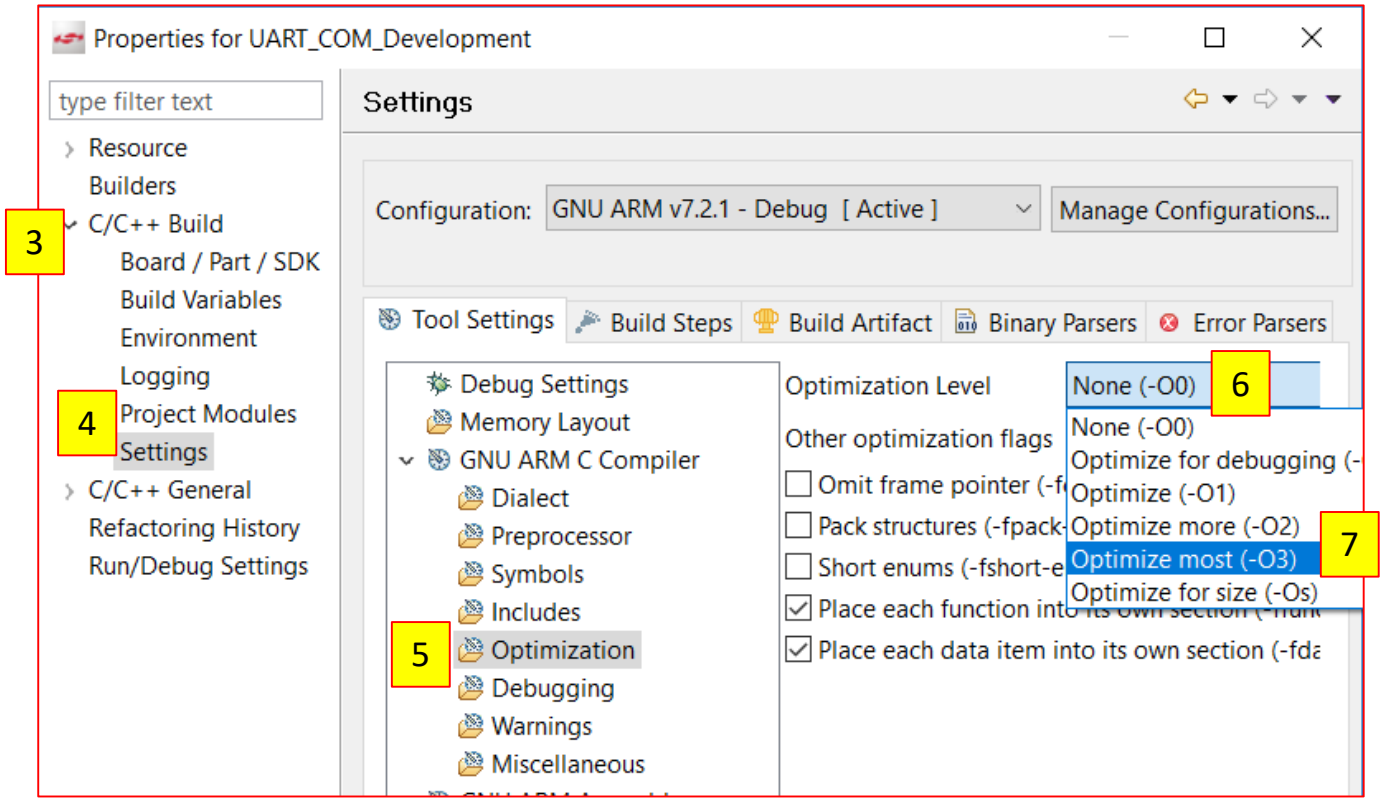
USART_Tx() is moved into the main program:

```
  while (1) {
      if(rx_flag){ // IT occurs->flag becomes true in IT routine
          rx_flag = false; // clear flag
          USART_Tx(UART0, rx_data);    // rx_data is sent to UART0
      }
  }
}
```

Efficient code: optimization level

- To generate a more efficient (in terms of memory usage, runtime, etc.) code optimization should be applied: (O3)





Efficient code: optimization level

- After applying the ‘most optimized’ –O3 optimization level the code will not work any more
- Explanation:
  - The optimizer replaces variables with constants whose value does not change in the main program (according to the compiler)
    - Such variables are rx_flag and rx_data!
    - The compiler optimizes the main program and does not consider the IT function definition before the main program (where rx_flag and rx_data change their values)
    - To prevent the optimizer changing rx_flag and rx_data they must be **volatile**

Efficient code: optimization level

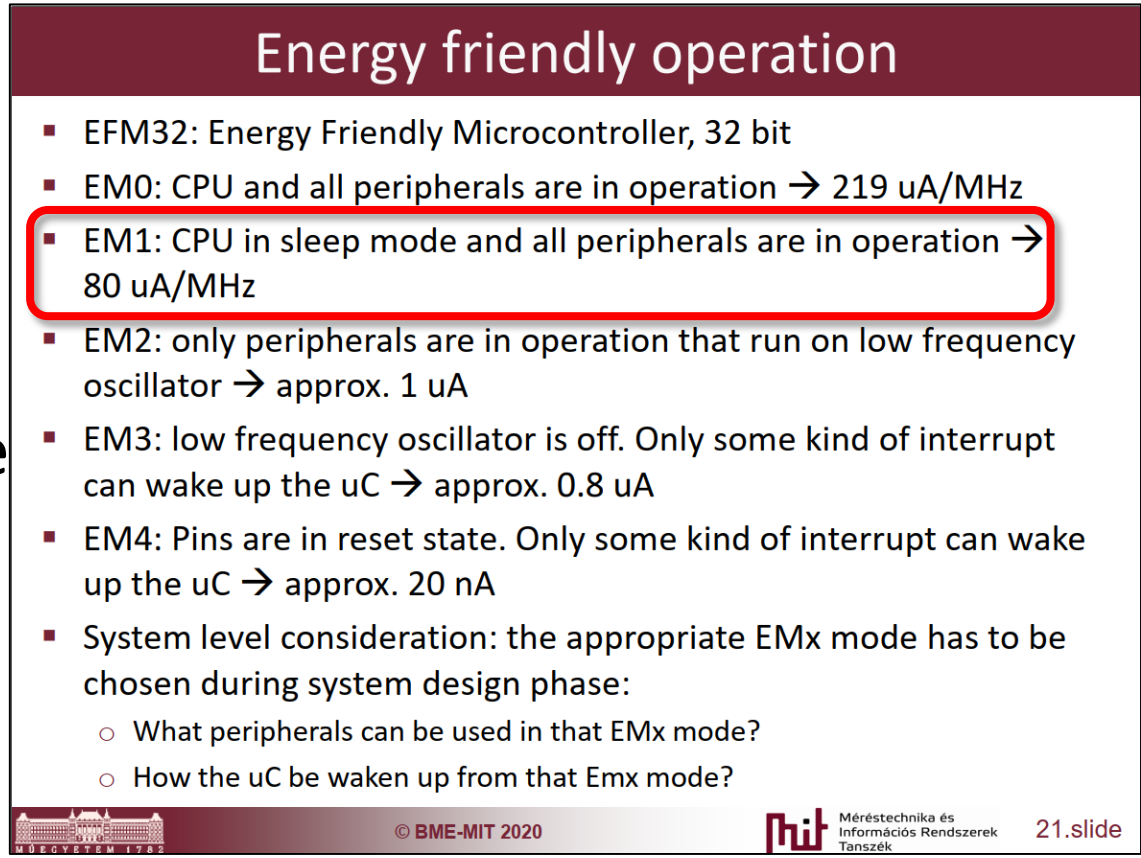
- To prevent the optimizer changing rx_flag and rx_data to constants they must be **volatile**

```
volatile _Bool rx_flag = false; //a flag is defined to indicate IT
volatile uint8_t rx_data; // a variable is defined for USART_Tx data
                          // (whose type is uint8_t)
void UART0_RX_IRQHandler(void){ // definition of IT function:
    rx_flag = true;                 // IT occurred->flag becomes true
    rx_data = USART_Rx(UART0);      // data read from UART0 and stored in rx_data
    //USART_Tx(UART0, rx_data);     // USART_Tx is moved into main program
    USART_IntClear(UART0, _USART_IFC_MASK); // IT flag is cleared
}
```

- In IT functions the variables used must be volatile
  - This type indicate to the compiler that the value can change and should not be optimized
- This kind of errors (if any) are difficult to discover

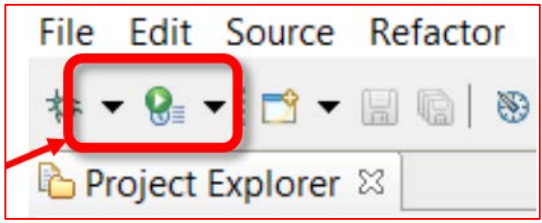
Energy friendly operation

- In many cases embedded systems are operated without any maintenance therefore energy friendly operation can be a real issue (battery life)
- Recall lecture:
  - EM1 energy friendly operation mode is promising
  - Proc. is in idle state but an IT can wake it up
  - Can we save energy?

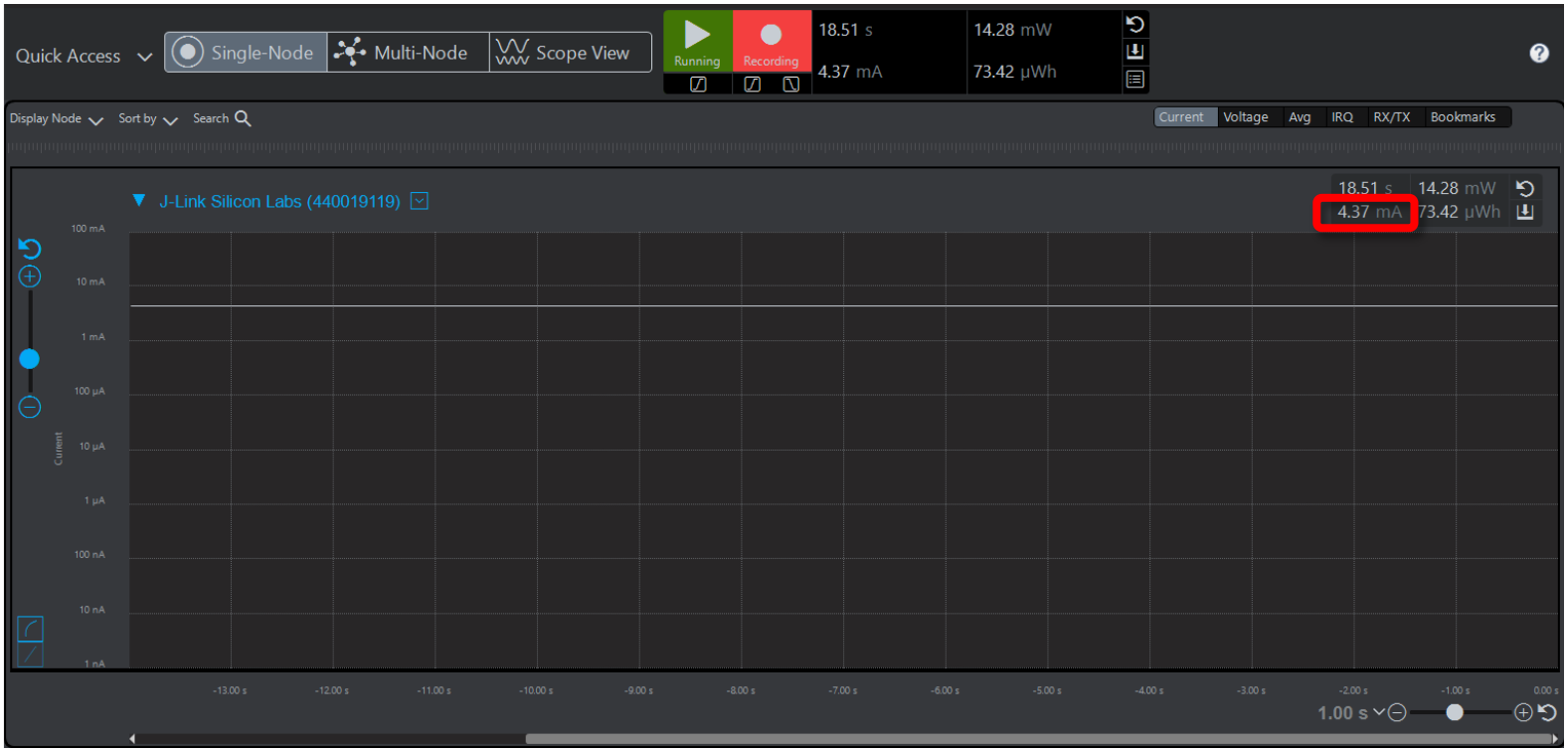


Energy friendly operation

- Energy consumption can be checked using Energy Profiler in Simplicity St.

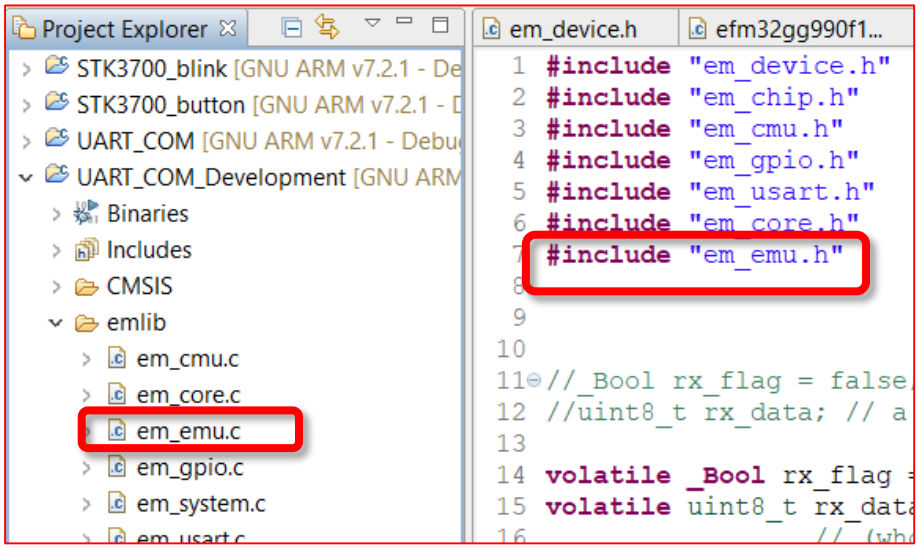


- Current consumption without EM1 energy saving mode: 4.37mA



Energy friendly operation

- EMU_EnterEM1() function shall be applied and em_emu.c and h header file must be included into the project and into the program, respectively



- The code should be modified as:

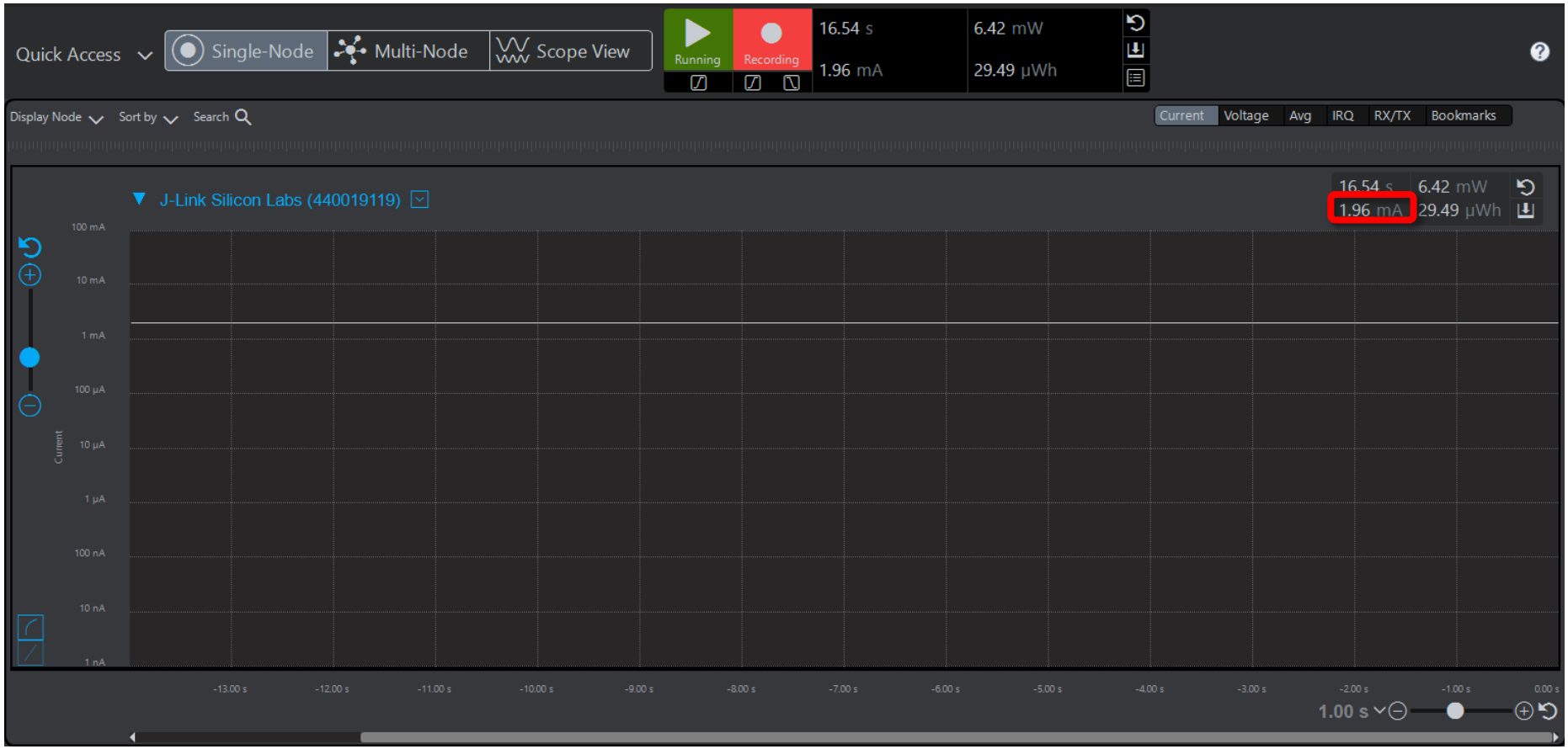
```
while (1) {

    EMU_EnterEM1(); // EM1 energy

    if(rx_flag){ // IT occurs->fla
        rx_flag = false; // clear
        USART_Tx(UART0, rx_data);
    }
}
```

Energy friendly operation

- Current consumption with EM1 energy saving mode: 1.96mA (consumption reduced by 55%!!!)



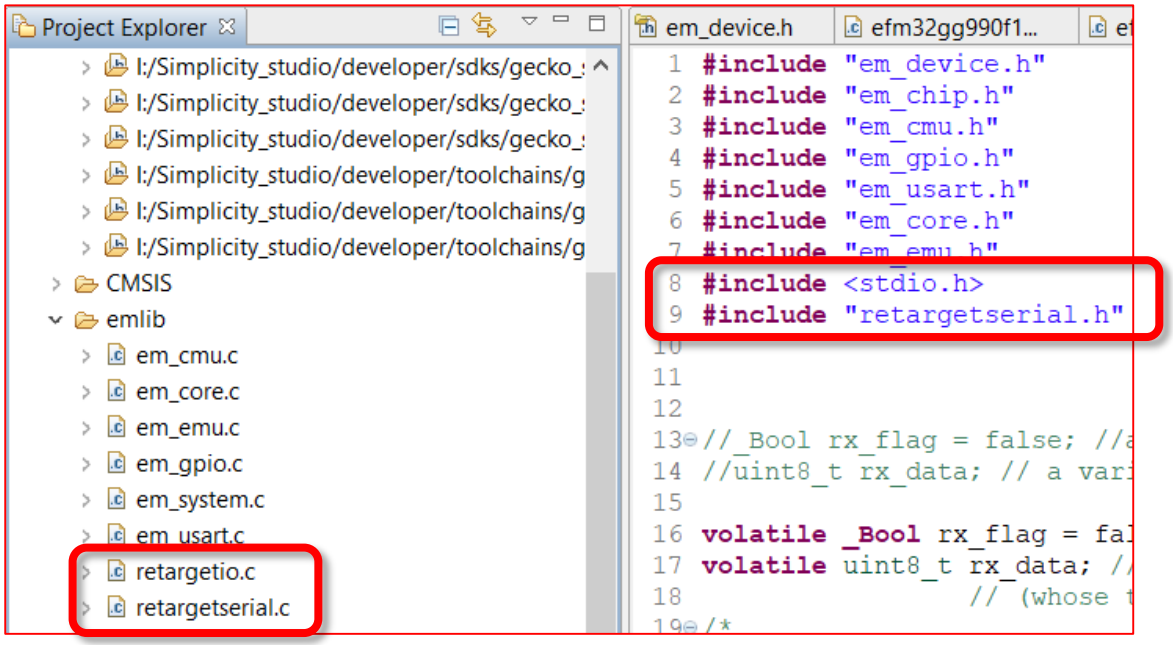
Application of Stdio (UART0)

- So far UART0 has been used directly via USART_Rx() and USART_Tx() functions
- I/O of standard C can be rerouted and upper level routines may have been already implemented to use low level functions for character transmitting and reception
- If these functions are applied for UART peripheral then actually a printf() function will send characters to the serial port

Application of Stdio (UART0)

- To achieve this high level functionality
  - retargetio.c and retargetserial.c files have to be added to the project (drag and drop)
  - The files are found in:
    - [installation folder]\Simplicity_studio\developer\sdks \gecko_sdk_suite\v2.6\hardware\kit\common\drivers\
  - retargetserial.h and stdio.h must be included into the program
  - IT function and IT handling (enable, clear) should not be active (comment them out) because the high level print function will handle them
  - Neither character sending in while is needed

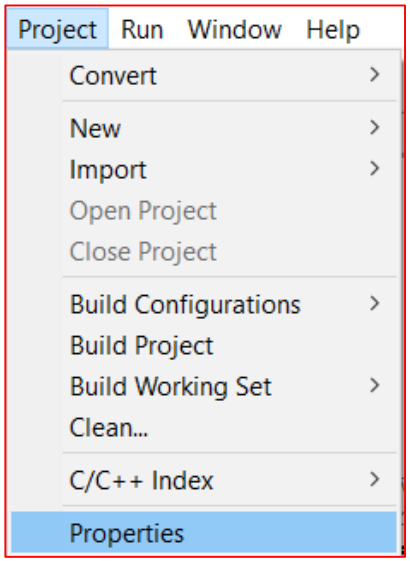
Application of Stdio (UART0)

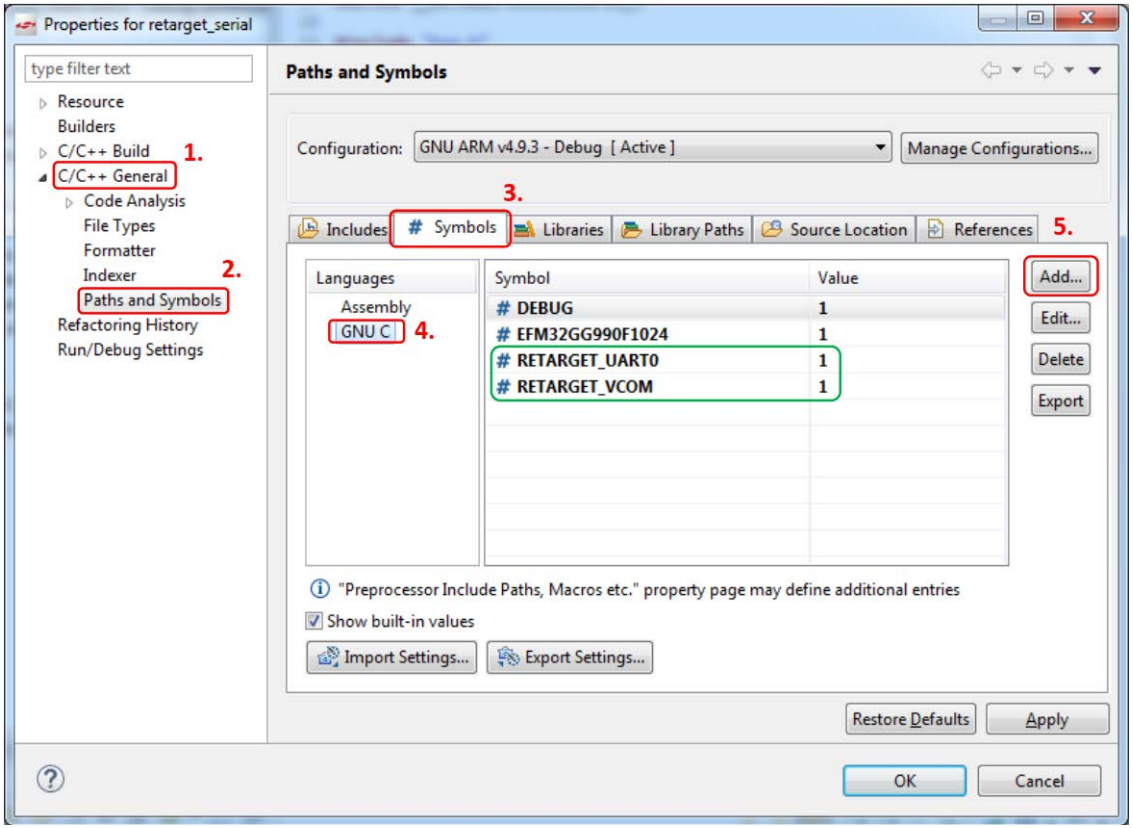


- Preprocessor directives have to be defined for the project:
  - RETARGET_UART0 : we want to use UART0
  - RETARGET_VCOM : UART0 be connected to USB port as virtual serial port via board controller

Application of Stdio (UART0)

- RETARGET_UART0 : we want to use UART0
- RETARGET_VCOM : UART0 be connected to USB port as virtual serial port via board controller
  - Both values are 1
  - Set the in Project ->properties





Application of Stdio (UART0)

- Now everything is set in the project to be able to use printf() to write to the UART
- We have to work on the code to use it:
  - Initialization:
    - RETARGET_SerialInit(); and
    - RETARGET_SerialCrLf(true); function must be called
  - Application of printf function:
    - printf("\n ********** Hello! ********** \n");
    - Code to be applied:
      - Note: new functions are before the while

```
RETARGET_SerialInit();
RETARGET_SerialCrLf(true);
printf("\n  ****  Hello!  *****  \n");

/* Infinite loop */
while (1) {
```