Embedded and ambient systems 2020.11.03.

Practice 4
Development of UART communications: a more sophisticated approach

Méréstechnika és Információs Rendszerek Tanszék

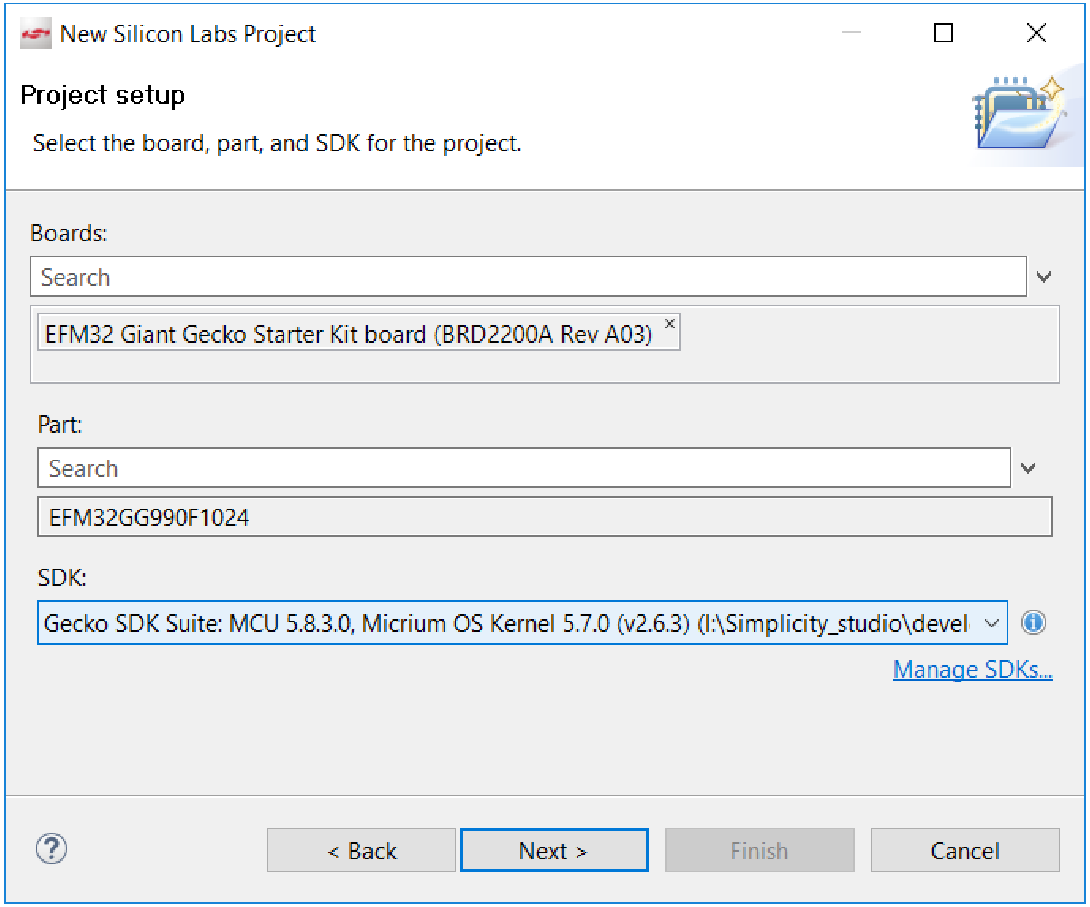
Problems with our UART implementation

```
/* Infinite loop */
while (1) {
    USART_Tx(UART0, USART_Rx(UART0));
}
```

- Remember the final solution:
  - This solution is a blocking implementation since USART_Rx will not return until data is received
  - Better solution to call USART_Rx function only if a character can be found in the buffer
  - An other good way to use interrupt
- Better to start a new project in the same way done before
- See the following sides to remember stating a new project

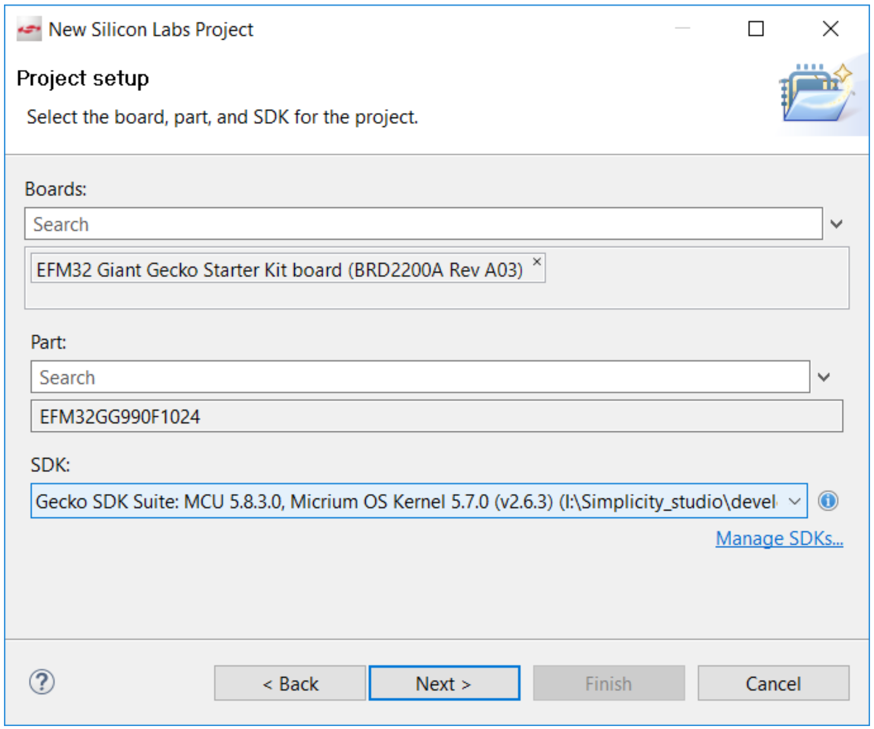
Strating with a new project

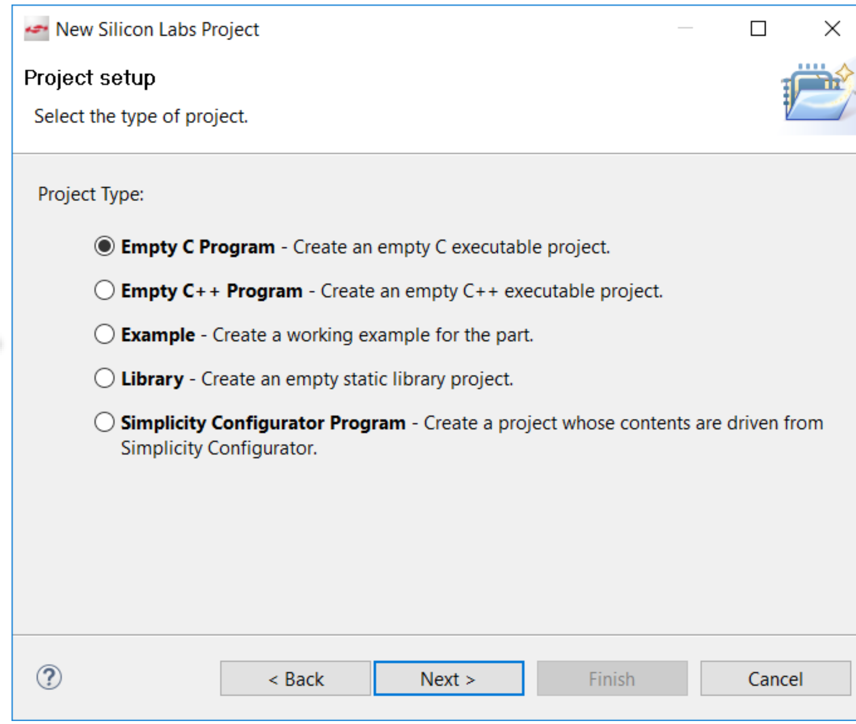
- File->New->Project->Silicon Labs MCU Project:



Strating with a new project

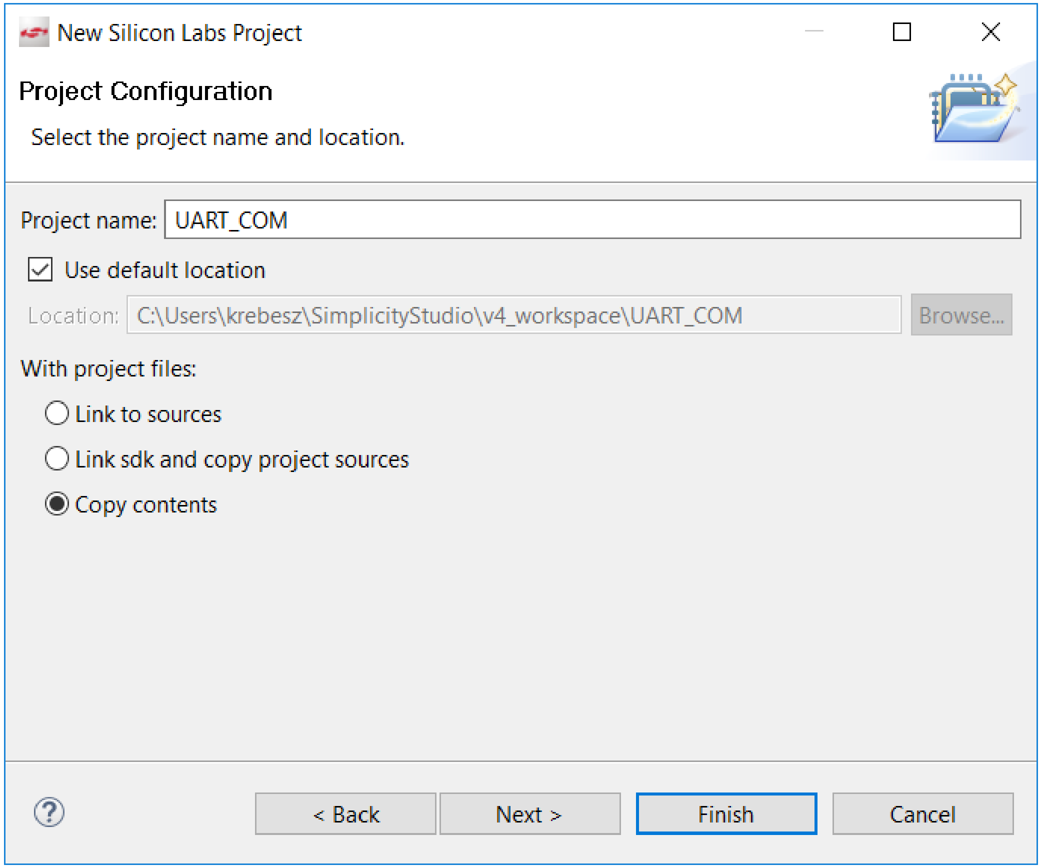
- File->New->Project->Silicon Labs MCU Project:





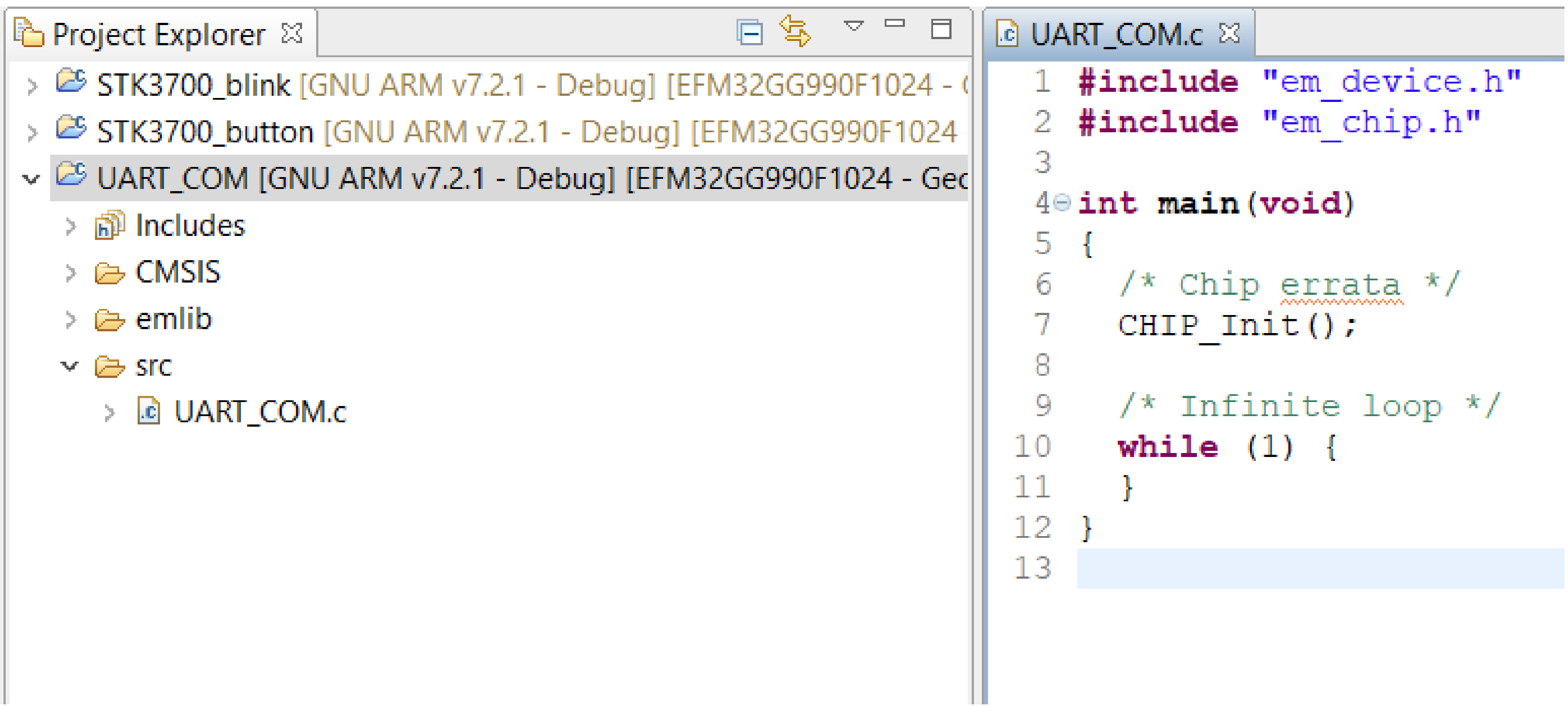
Strating with a new project

- Give project name and location, and set Copy content:



Project created – start programming

- Main.c can be also renamed to UART_COM.c
- Although an empty C project has been created a program skeleton is offered automaticly



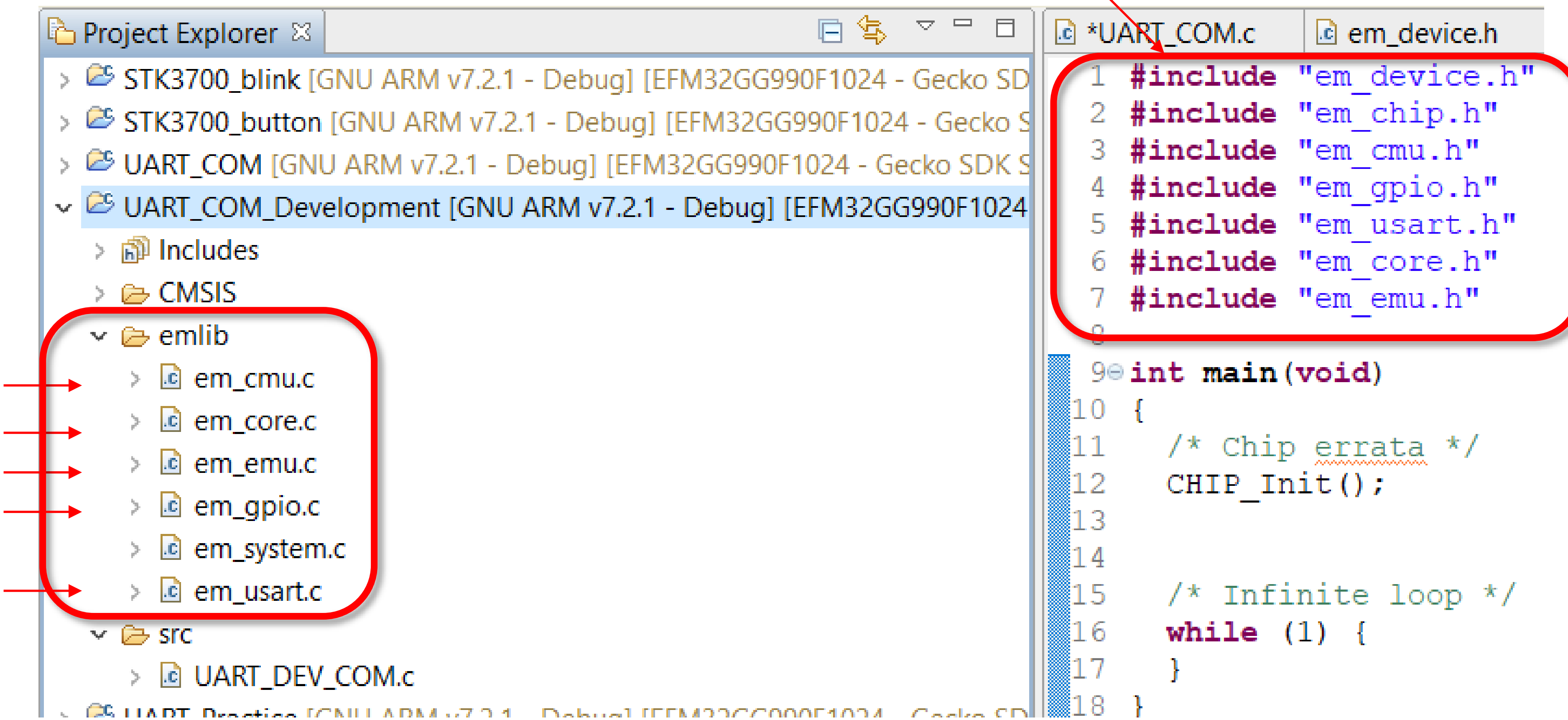
Files to be added to the project

- Search the library where Simplicity Studio is installed
  - o Contains include (inc: *.c) and source (src: *.h) files:

  i:\Simplicity_studio\developer\sdks\gecko_sdk_suite\v2.6\platform\emlib\
- Following files have to be drag-and-dropped into emlib library of the project (see next slide):
  - o em_cmu.c (clock management unit)
  - o em_gpio.c
  - o em_usart.c
  - o em_core.c
  - o em_emu.c (energy management unit)

Files to be added to the project

- Furthermore they have to be included into the program:



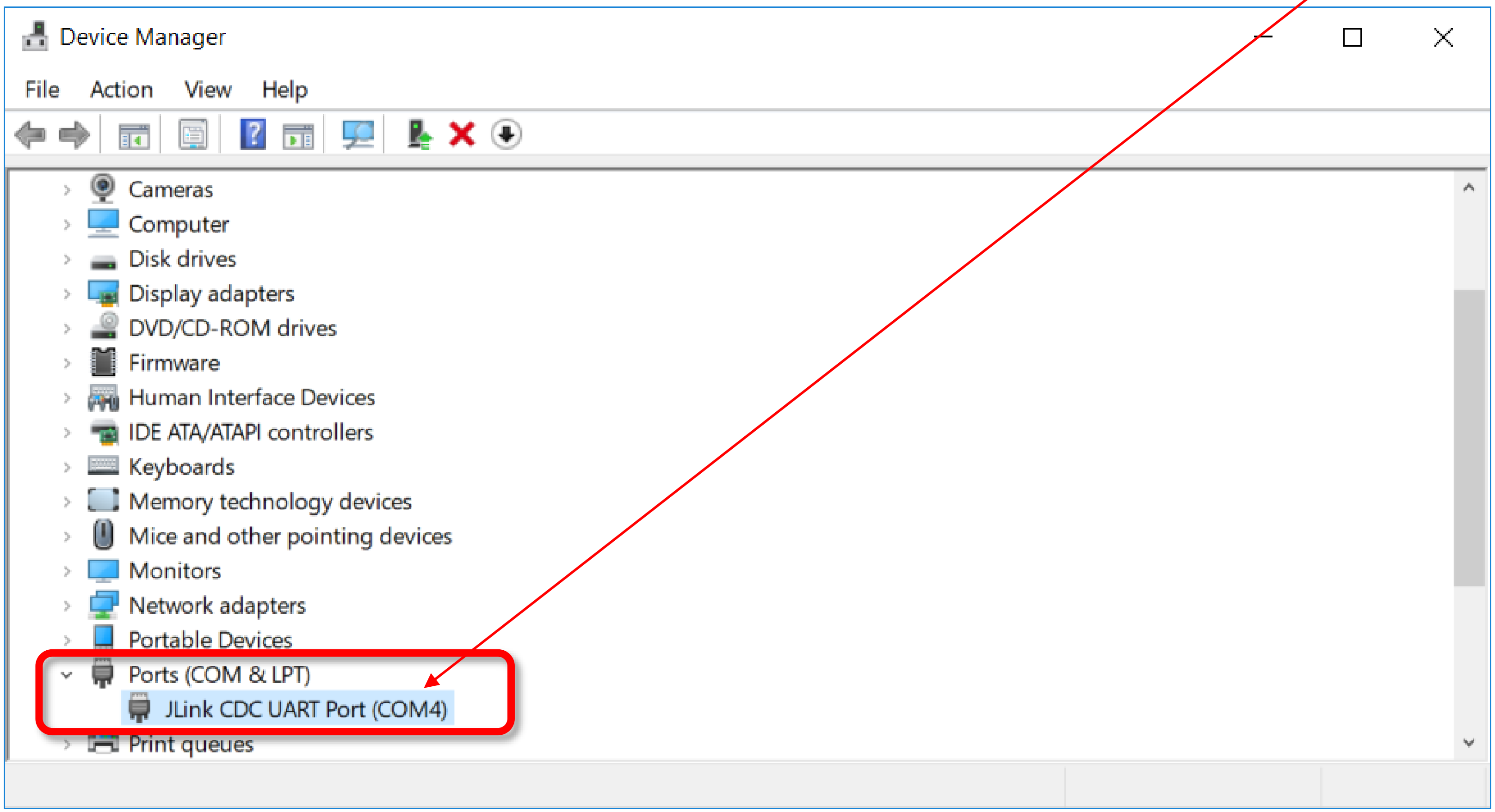
Code to start with

Use the following code as a reference for your work (continue from previous result):

```
#include "em_device.h"
#include "em_chip.h"
#include "em_cmu.h"
#include "em_gpio.h"
#include "em_usart.h"
#include "em_core.h"
#include "em_emu.h"

int main(void)
{
  /* Chip errata */
  CHIP_Init();

  // Enable clock for GPIO
  CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

  // Set PF7 to high
  GPIO_PinModeSet(gpioPortF, 7, gpioModePushPull, 1);

  // Configure UART0
  // (Now use the "emlib" functions whenever possible.)

  // Enable clock for UART0
  CMU_ClockEnable(cmuClock_UART0, true);

 // Initialize UART0 (115200 Baud, 8N1 frame format)

  // To initialize the UART0, we need a structure to hold
  // configuration data. It is a good practice to initialize it with
  // default values, then set individual parameters only where needed.
  USART_InitAsync_TypeDef UART0_init = USART_INITASYNC_DEFAULT;

  USART_InitAsync(UART0, &UART0_init);
  // USART0: see in efm32ggf1024.h

  // Set TX (PE0) and RX (PE1) pins as push-pull output and input resp.
  // DOUT for TX is 1, as it is the idle state for UART communication
  GPIO_PinModeSet(gpioPortE, 0, gpioModePushPull, 1);
  // DOUT for RX is 0, as DOUT can enable a glitch filter for inputs,
  // and we are fine without such a filter
  GPIO_PinModeSet(gpioPortE, 1, gpioModeInput, 0);

  // Use PE0 as TX and PE1 as RX (Location 1, see datasheet (not refman))
  // Enable both RX and TX for routing
  UART0->ROUTE |= UART_ROUTE_LOCATION_LOC1;
  // Select "Location 1" as the routing configuration
  UART0->ROUTE |= UART_ROUTE_TXPEN | UART_ROUTE_RXPEN;

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

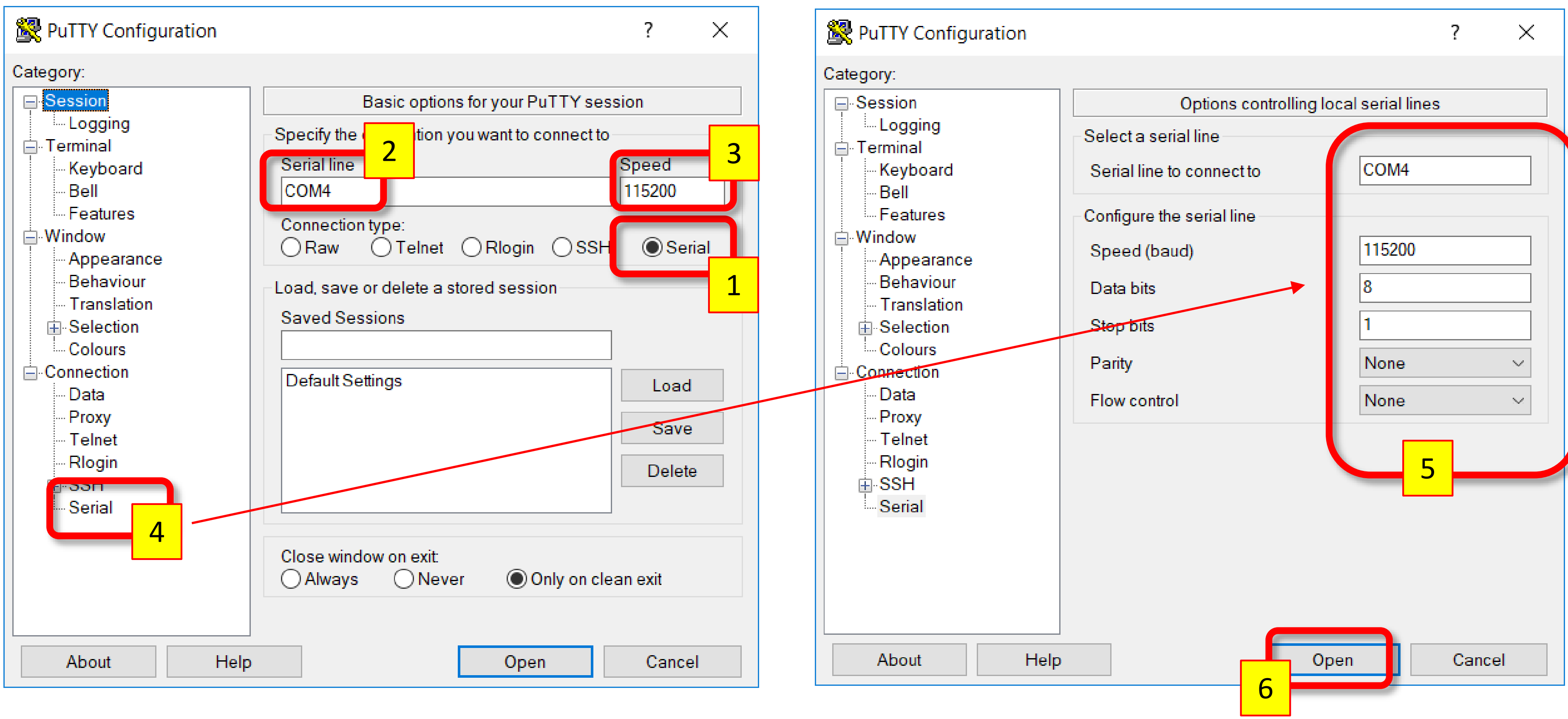
Setting the terminal program

- Check UART (COM port number and its settings) in Device Manager in Windows (now it is COM4)



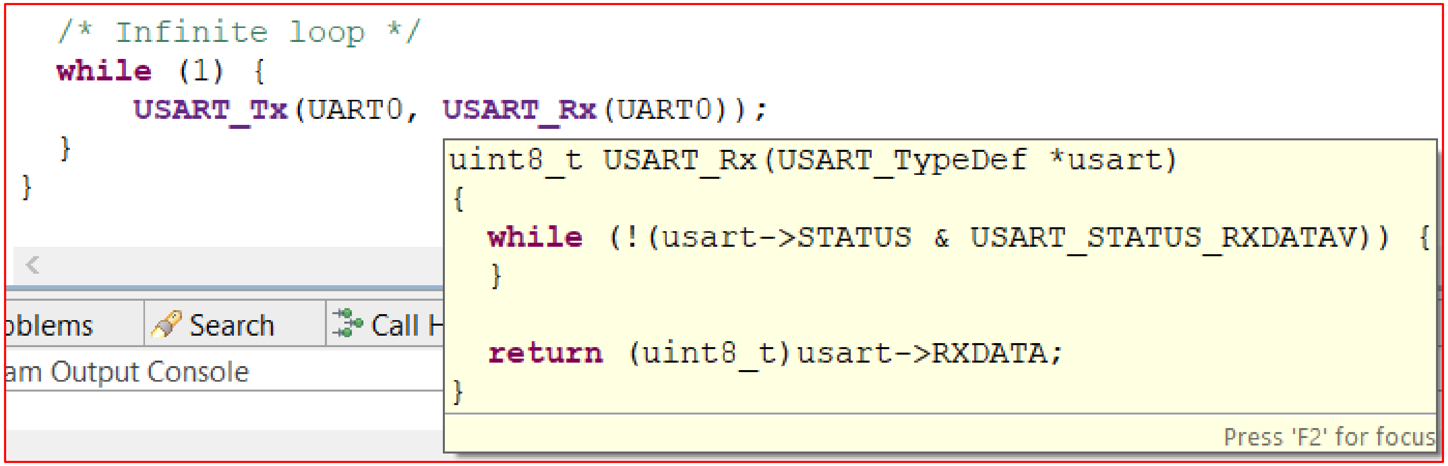
Setting the terminal program

- A PC-based terminal program is needed to get access to COM4 port: an option is putty.exe



Non-blocking character reception

- Check our previous solution again
  - What does USART_Rx do(stay on it by mouse pointer)?



  - Operation: remains in while loop until in USART_STATUS_RXDATAV bit flips to 1, then returns with the received character (RXDATA)
    - See 03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf on page 481 (and next slide)
  - Blocking can be avoided if we check the STATUS reg

Non-blocking character reception

17.5.5 USARTn_STATUS - USART Status Register

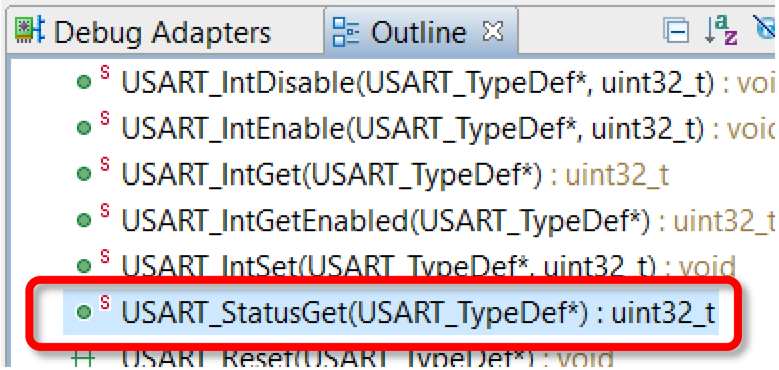
Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	1	0	0	0	0	0	0
Access																				R	R	R	R	R	R	R	R	R	R	R	R	R
Name																				RXFULLRIGHT	RXDATAVRIGHT	TXBSRIGHT	TXBDRIGHT	RXFULL	RXDATAV	TXBL	TXC	TXTRI	RXBLOCK	MASTER	TXENS	RXENS

7	RXDATAV	0	R	**RX Data Valid**
	Set when data is available in the receive buffer. Cleared when the receive buffer is empty.			

- Non-blocking solution: check STATUS reg. and call USART_Rx() function only if incoming character is available

Non-blocking character reception

- Search em_usart.h for a function that checks STATUS register (if available, hopefully it is):




```
/*****************************************************************
 * @brief
 *   Get USART STATUS register.
 *
 * @param[in] usart
 *   Pointer to USART/UART peripheral register block.
 *
 * @return
 *  STATUS register value.
 *
 *****************************************************************
__STATIC_INLINE uint32_t USART_StatusGet(USART_TypeDef *usart)
{
  return usart->STATUS;
}
```

Non-blocking character reception

- Application of USART_StatusGet() function:

```
/* Infinite loop */
while (1) {
    if (USART_StatusGet(UART0) & USART_STATUS_RXDATAV) {
        USART_Tx(UART0, USART_Rx(UART0));
    }
}
```

- Even more elegant solution if we implement an own non-blocking function to receive characters

```
int USART_RxNonblocking(USART_TypeDef *usart)
{
  int retVal = -1;

  if (usart->STATUS & USART_STATUS_RXDATAV) {
        retVal = (int)(usart->RXDATA);
  }

  return retVal;
}
```

Implementation of non-blocking function
(put it before the main function)

```
int ch;
ch = USART_RxNonblocking(UART0);
if (ch != -1) {
    USART_Tx(UART0, ch);
}
```

Application of non-blocking function
(put it in the main function)

Non-blocking character reception

- Remark on USART_Tx() function:
  - If data to be sent is too much even USART_Tx() function can be blocking – have a look at USART_Tx()

```
/* Infinite loop */
while (1) {
    //USART_StatusGet(USART_TypeDef *usart)
    if (USART_StatusGet(UART0) & USART_STATUS_RXDATAV) {
        USART_Tx(UART0, USART_Rx(UART0));
    }
}
```

```
void USART_Tx(USART_TypeDef *usart, uint8_t data)
{
  /* Check that transmit buffer is empty */
  while (!(usart->STATUS & USART_STATUS_TXBL)) {
  }
  usart->TXDATA = (uint32_t)data;
}
```

Press 'F2' for focus

  - Clearly seen that blocking may happen but “less severe” –> USART_STATUS_TXBL bit is checked in STATUS register

Non-blocking character reception

- USART_STATUS_TXBL bit (TXBL may appear in other registers- be careful)

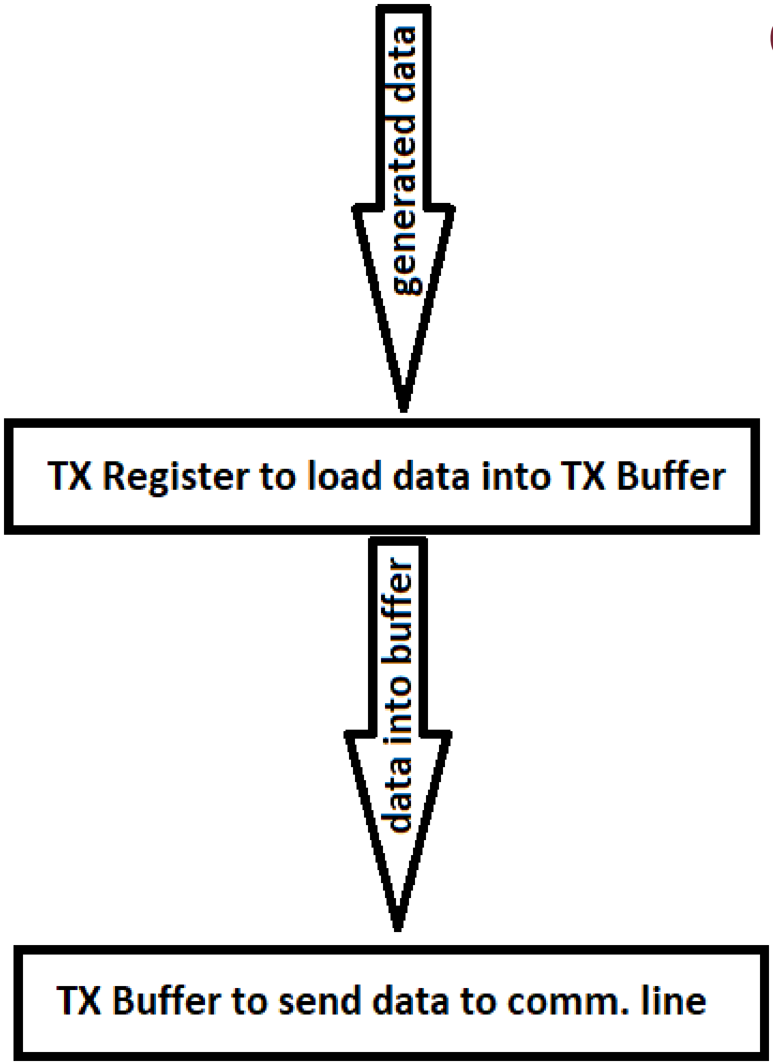
17.5.5 USARTn_STATUS - USART Status Register

Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	1	0	0	0	0	0	0
Access																				R	R	R	R	R	R	R	R	R	R	R	R	R
Name																				RXFULLRIGHT	RXDATAVRIGHT	TXBSRIGHT	TXBDRIGHT	RXFULL	RXDATAV	TXBL	TXC	TXTRI	RXBLOCK	MASTER	TXENS	RXENS

Bit	Name	Reset	Access	Description
6	TXBL	1	R	**TX Buffer Level**
	Indicates the level of the transmit buffer. If TXBIL is cleared, TXBL is set whenever the transmit buffer is empty, and if TXBIL is set, TXBL is set whenever the transmit buffer is half-full or empty.			

See 03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf on page 481

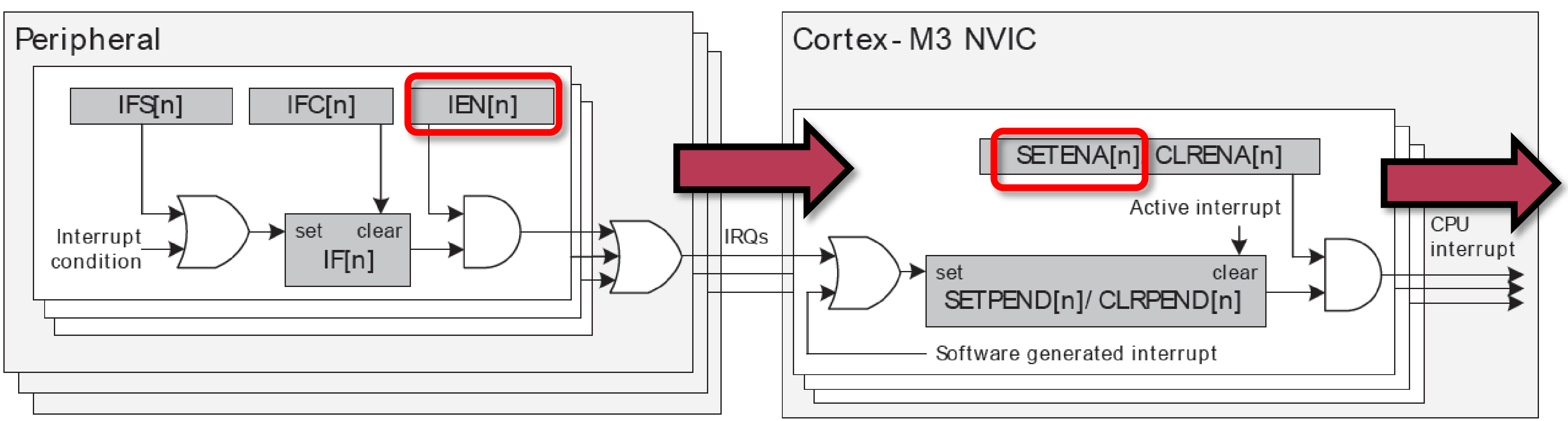
Non-blocking character reception



- Operation of data transmission:
  - Generated data is loaded into TX Register only if TX Register is empty
    - Otherwise data in TX Register is overwritten and data loss may occur
  - If TX Buffer is empty data is loaded into it from TX Register
  - From TX Buffer data is sent out via the communication line (UART)
  - R=115200bps->1byte needs 70us
  - T_clk=1/14MHz=70ns->1000cycles per byte!!!

Interrupt-based character reception

- Problem with non-blocking character reception
  - If the main program executes a long-lasting task before repeated checking of character is done data loss may occur
  - To prevent that kind of data loss application of interrupt can be a solution



IT initialization for a peripheral

- Initialization of IT in a general case:
  - Enabling peripheral (turn perif. on, config., etc.)
  - Determination of IT-handling function
  - Clear of IT flag belonging to the certain IT
    - An IT request may be stuck from a previous state that can cause problem since after enabling IT a false interrupt can take action. A stuck IF can be the consequence of a non-initialized peripheral (e.g. IT occurs on a floating input)
  - Enabling the IT of a certain peripheral
  - Clearing of global IT flag (if needed)
  - Enabling of global IT

DONE previously (UART_init)

C O M E S N O W

L A T E R

NOTE: THIS SLIDE COMES FROM THE INTERRUPT TOPIC OF LECTURES USE THAT LECTURE AS A REFERENCE IF NEEDED

Interrupt-based character reception

- Interrupt has to be enabled for UART

17.4 Register Map

03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf
See page 475

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x040	USARTn_IF	R	Interrupt Flag Register
0x044	USARTn_IFS	W1	Interrupt Flag Set Register
0x048	USARTn_IFC	W1	Interrupt Flag Clear Register
0x04C	USARTn_IEN	RW	Interrupt Enable Register

17.5.20 USARTn_IEN - Interrupt Enable Register

03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf
See page 490

Offset	Bit Position																														
0x04C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0	0	0	0
Access																				RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL	RXDATAV	TXBL	TXC

2	RXDATAV	0	RW	**RX Data Valid Interrupt Enable**
	Enable interrupt on RX data.			

Interrupt-based character reception

- Check em_usart.h for interrupt enable function

```
No Adapters    Outline
USART_InitPrsTrigger(USART_TypeDef*, const USART
USART_InitIrDA(const USART_InitIrDA_TypeDef*) : vo
USART_IntClear(USART_TypeDef*, uint32_t) : void
USART_IntDisable(USART_TypeDef*, uint32_t) : void
USART_IntEnable(USART_TypeDef*, uint32_t) : void
USART_IntGet(USART_TypeDef*) : uint32_t
USART_IntGetEnabled(USART_TypeDef*) : uint32_t
```

```c
 * @param[in] usart
 *   Pointer to USART/UART peripheral register block.
 *
 * @param[in] flags
 *   USART/UART interrupt source(s) to enable. Use one or more valid
 *   interrupt flags for the USART module (USART_IF_nnn) OR'ed together.
 ******************************************************************************
__STATIC_INLINE void USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
{
  usart->IEN |= flags;
}
```

- Insert USART_IntEnable() function
  flags = register content, here the 2nd bit is interesting (see previous slide)
  - Check efm32_gg_usart.h

```c
#define USART_IEN_RXDATAV (0x1UL << 2) /**< RX Data Valid Interrupt Enable */
```

  - Code to be applied:

```c
    //USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
    USART_IntEnable(UART0, USART_IEN_RXDATAV)

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

Interrupt-based character reception

- Interrupts have to be cleared (all ITs) for UART
  - Check em_usart.h for interrupt clear function

No Adapters | Outline

```
USART_InitPrsTrigger(USART_TypeDef*, const USART
USART_InitIrDA(const USART_InitIrDA_TypeDef*) : vo
USART_IntClear(USART_TypeDef*, uint32_t) : void
USART_IntDisable(USART_TypeDef*, uint32_t) : void
USART_IntEnable(USART_TypeDef*, uint32_t) : void
USART_IntGet(USART_TypeDef*) : uint32_t
USART_IntGetEnabled(USART_TypeDef*) : uint32_t
USART_IntSet(USART_TypeDef*, uint32_t) : void
USART_StatusGet(USART_TypeDef*) : uint32_t
```

```
 * @param[in] flags
 *   Pending USART/UART interrupt source(s) to clear. Use one or more val
 *   interrupt flags for the USART module (USART_IF_nnn) OR'ed together.
 ***********************************************************************
__STATIC_INLINE void USART_IntClear(USART_TypeDef *usart, uint32_t flags)
{
#if defined (USART_HAS_SET_CLEAR)
  usart->IF_CLR = flags;
#else
  usart->IFC = flags;
#endif
}
```

17.5.19 USARTn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																															
0x048	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0			0
Access																				W1	W1	W1	W1	W1	W1	W1	W1	W1	W1			W1
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL			TXC

03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf See page 489

Interrupt-based character reception

- All bits in USARTn_IFC register have to be cleared
  - A define can be found in efm32gg_usart.h for that purpose:

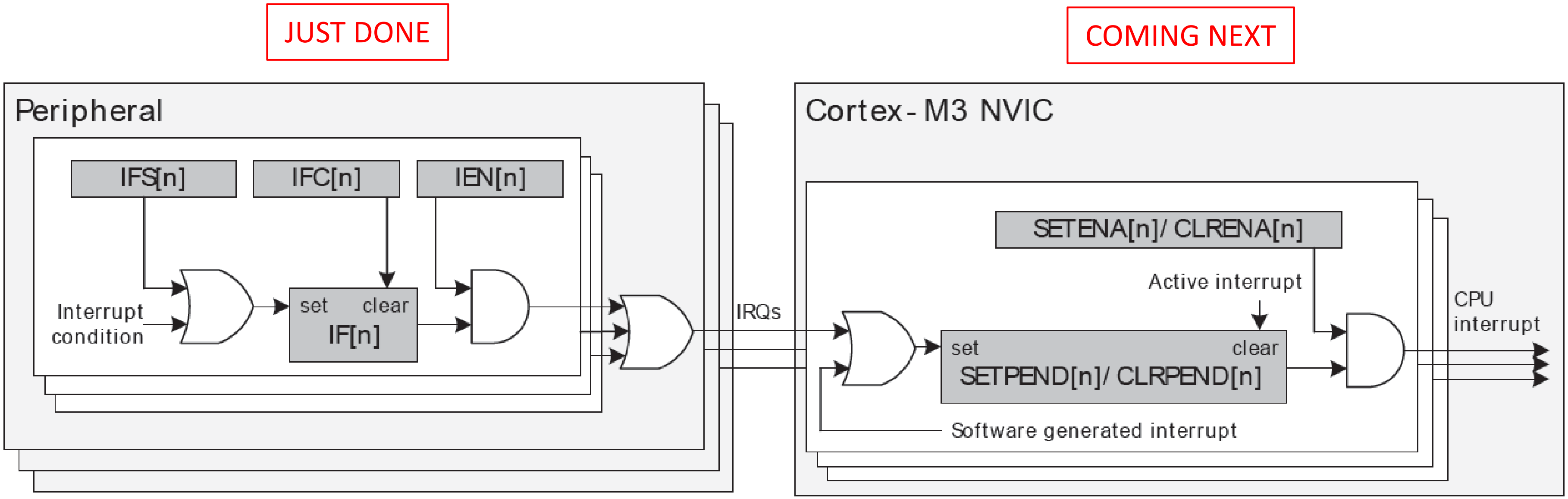
```
#define _USART_IFC_MASK   0x00001FF9UL   /**< Mask for USART_IFC */
```

- Insert USART_IntClear() function after UART init
- Code to be applied:

```
    //USART_IntClear(USART_TypeDef *usart, uint32_t flags)
    USART_IntClear(UART0, _USART_IFC_MASK);

    //USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
    USART_IntEnable(UART0, USART_IEN_RXDATAV);

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

- This step is precautious: it is very probable that the program would work but in general, not clearing IT flags can cause a trouble

Interrupt-based character reception



- So far UART peripheral-related IT has been dealt with
- From now let’s see the core-related IT

Interrupt-based character reception

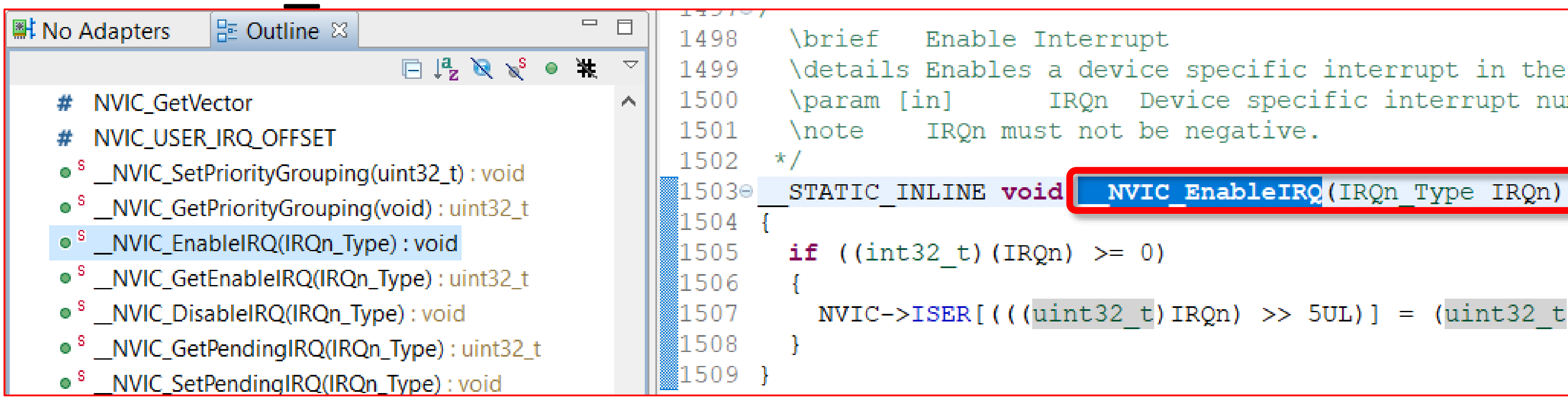
- Core-related IT– IT for the UART has to be enabled
    - em_decive.h + F3 (among included header files in at the top of the program)

      -> find in it efm32gg990f1024.h + F3

      -> find in it core_cm3.h + F3

      NVIC functions are needed

- In core_cm3.c search for



Interrupt-based character reception

- In core_cm3.c search for
  - `void __NVIC_EnableIRQ(IRQn_Type IRQn)`
    - `IRQn_Type IRQn` + F3 to check the possible ITs to find:

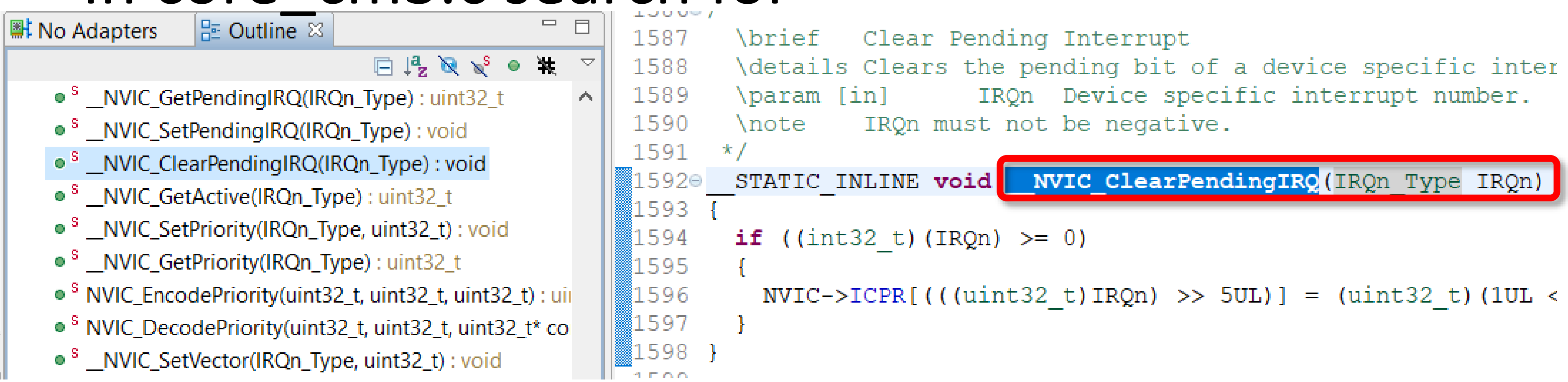
```
UART0_RX_IRQn  = 20, /*!< 20 EFM32 UART0_RX Interrupt */
```

- Code to be applied:

```
    //USART_IntClear(USART_TypeDef *usart, uint32_t flags)
    USART_IntClear(UART0, _USART_IFC_MASK);

    //USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
    USART_IntEnable(UART0, USART_IEN_RXDATAV);

    //void __NVIC_EnableIRQ(IRQn_Type IRQn)
    __NVIC_EnableIRQ(UART0_RX_IRQn);

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

Interrupt-based character reception

- Core-related IT– IT flags has to be cleared
  - em_decive.h + F3 (among included header files in at the top of the program)
    -> find in it efm32gg990f1024.h + F3
    -> find in it core_cm3.h + F3
    NVIC functions are needed
- In core_cm3.c search for



Interrupt-based character reception

- In core_cm3.c search for
  - `void __NVIC_ClearPendingIRQ(IRQn_Type IRQn)`
    - `IRQn_Type IRQn` + F3 to check the possible ITs to find:

```
UART0_RX_IRQn  = 20, /*!< 20 EFM32 UART0_RX Interrupt */
```

- Code to be applied:

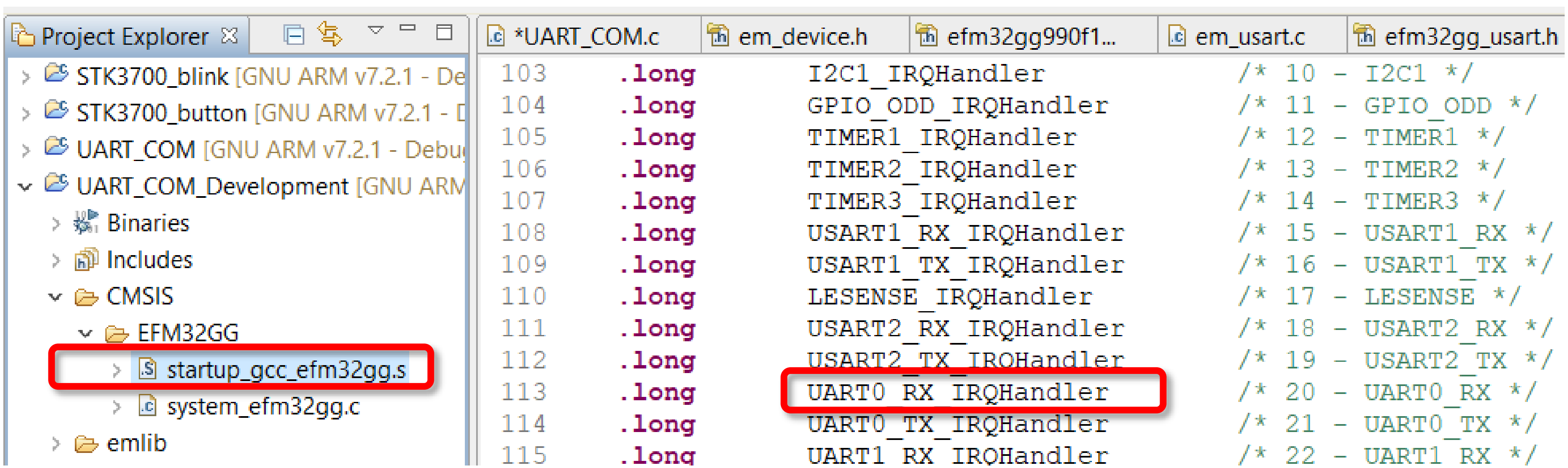
```
//USART_IntClear(USART_TypeDef *usart, uint32_t flags)
USART_IntClear(UART0, _USART_IFC_MASK);

//USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
USART_IntEnable(UART0, USART_IEN_RXDATAV);

//void __NVIC_ClearPendingIRQ(IRQn_Type IRQn)
__NVIC_ClearPendingIRQ(UART0_RX_IRQn);

//void __NVIC_EnableIRQ(IRQn_Type IRQn)
__NVIC_EnableIRQ(UART0_RX_IRQn);

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

UART Perif. IT clear and enable

Proc. core IT clear and enable

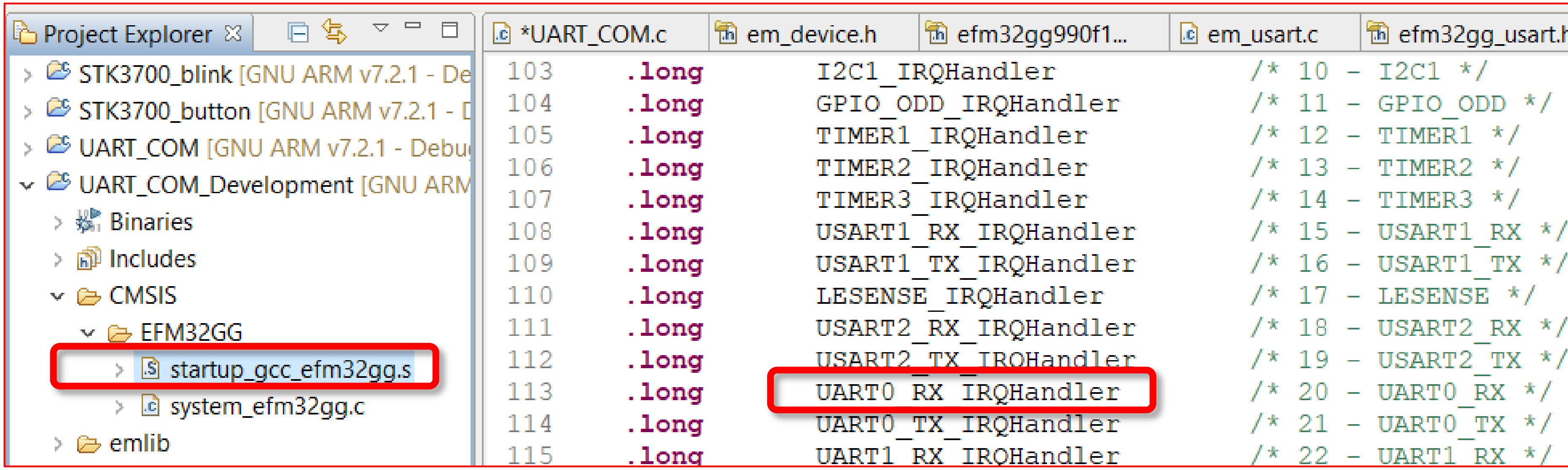
Interrupt-based character reception

- ITs have just been correctly configured
  - When a character is received at UART0, IT is generated
- IT function has to be implemented
  - What should happen when IT event occurs
  - Check startup_gcc_efm32gg.s in Project Explorer



Interrupt-based character reception

- Check startup_gcc_efm32gg.s in Project Explorer
- Search for UART0_RX_IRQHandler:



- UART0_RX_IRQHandler is a weak function so it can be overdefined in the program without causing any error:

```
/*    Macro to define default handlers. Default handler
 *    will be weak symbol and just dead loops. They can be
 *    overwritten by other handlers.
 */
    .macro  def_irq_handler handler_name
    .weak   \handler_name
    .set    \handler_name, Default_Handler
    .endm
```

Interrupt-based character reception

- Implementation of IT function in the program code
- UART_RX_IRQHandelr function has to be defined before the main function
  - During IT the received data has to be sent to UART
- Code to be applied:

```
uint8_t rx_data;

void UART0_RX_IRQHandler(void){
    rx_data=USART_Rx(UART0);
    USART_Tx(UART0, rx_data);
    USART_IntClear(UART0, _USART_IFC_MASK);
}

int main(void)
{
```

- Note: no input parameter and no return value
  -> void func(void){
  what happen during IT;
  clear IT flag; }

Appendix: code – a working version

```c
#include "em_device.h"
#include "em_chip.h"
#include "em_cmu.h"
#include "em_gpio.h"
#include "em_usart.h"
#include "em_core.h"
#include "em_emu.h"

uint8_t rx_data;

void UART0_RX_IRQHandler(void){
	rx_data=USART_Rx(UART0);
	USART_Tx(UART0, rx_data);
	USART_IntClear(UART0, _USART_IFC_MASK);
}

int main(void)
{
  /* Chip errata */
  CHIP_Init();

  // Enable clock for GPIO
	CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

	// Set PF7 to high
	GPIO_PinModeSet(gpioPortF, 7, gpioModePushPull, 1);

	// Configure UART0
	// (Now use the "emlib" functions whenever possible.)
```

Appendix: code – a working version

```
    // Enable clock for UART0
    CMU_ClockEnable(cmuClock_UART0, true);


    // Initialize UART0 (115200 Baud, 8N1 frame format)

    // To initialize the UART0, we need a structure to hold
    // configuration data. It is a good practice to initialize it with
    // default values, then set individual parameters only where needed.
    USART_InitAsync_TypeDef UART0_init = USART_INITASYNC_DEFAULT;

    USART_InitAsync(UART0, &UART0_init);
    // USART0: see in efm32ggf1024.h

    // Set TX (PE0) and RX (PE1) pins as push-pull output and input resp.
    // DOUT for TX is 1, as it is the idle state for UART communication
    GPIO_PinModeSet(gpioPortE, 0, gpioModePushPull, 1);
    // DOUT for RX is 0, as DOUT can enable a glitch filter for inputs,
    // and we are fine without such a filter
    GPIO_PinModeSet(gpioPortE, 1, gpioModeInput, 0);

    // Use PE0 as TX and PE1 as RX (Location 1, see datasheet (not refman))
        // Enable both RX and TX for routing
    UART0->ROUTE |= UART_ROUTE_LOCATION_LOC1;
        // Select "Location 1" as the routing configuration
    UART0->ROUTE |= UART_ROUTE_TXPEN | UART_ROUTE_RXPEN;
```

Appendix: code – a working version

```
    //USART_IntClear(USART_TypeDef *usart, uint32_t flags)
    USART_IntClear(UART0, _USART_IFC_MASK);

    //USART_IntEnable(USART_TypeDef *usart, uint32_t flags)
    USART_IntEnable(UART0, USART_IEN_RXDATAV);

    //void __NVIC_ClearPendingIRQ(IRQn_Type IRQn)
    __NVIC_ClearPendingIRQ(UART0_RX_IRQn);

    //void __NVIC_EnableIRQ(IRQn_Type IRQn)
    __NVIC_EnableIRQ(UART0_RX_IRQn);

  /* Infinite loop */
  while (1) {
      //USART_StatusGet(USART_TypeDef *usart)
      //if (USART_StatusGet(UART0) & USART_STATUS_RXDATAV) {
       //   USART_Tx(UART0, USART_Rx(UART0));
       // }
  }
}
```