Embedded and ambient systems 2022.11.23.

Practice 4
Application of serial port to implement communications via UART

Méréstechnika és Információs Rendszerek Tanszék

Needed during practice

- 01_EFM32_User_guide_efm32gg-stk3700-user_guide.pdf
- 02_EFM32_Schematic_EFM32GG-BRD2200A-A03-schematic.pdf
- 03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf
- 04_EFM32_Datasheet_efm32gg990_datasheet.pdf
- Terminal program

Difference between datasheet and user guide:

-Reference manual contains general info of the whole IC family

-Datasheet contains specific info of a certain type of IC (from the IC family)

UART / USRT / USART

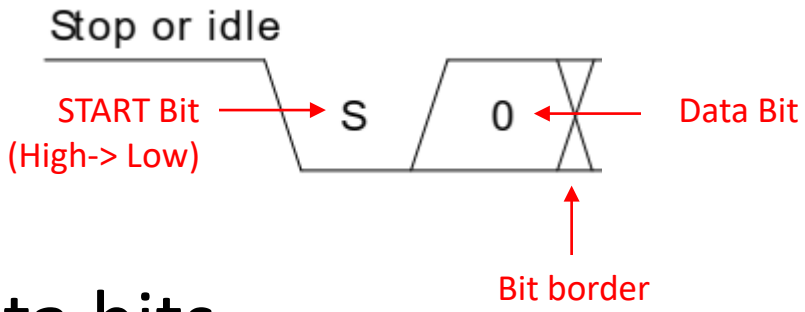
- UART or USRT or USART?
  - UART: Universal Asynchronous Receiver/Transmitter
    - Serial communication without application of CLK line
  - USRT: Universal Synchronous Receiver/Transmitter
    - Serial communication based on CLK signal
  - USART: Universal Synchronous Asynchronous Receiver/Transmitter
    - Since the operation is very similar (main difference is the CLK signal) sometimes both are discussed without distinction

UART properties

- No CLK signal, i.e., CLK line not needed->less wire
- 2 data lines: transmitter (Tx) and receiver (Rx) line
- Communication speed (=bit duration) has to be set -> defines the bit borders in the system
  - Reference oscillators at both the Tx and Rx sides has to be precise otherwise frequency difference will occur between Tx and Rx side and bit duration will change
  - If CLK existed it would define the bit borders (as done in USRT)

UART Communications

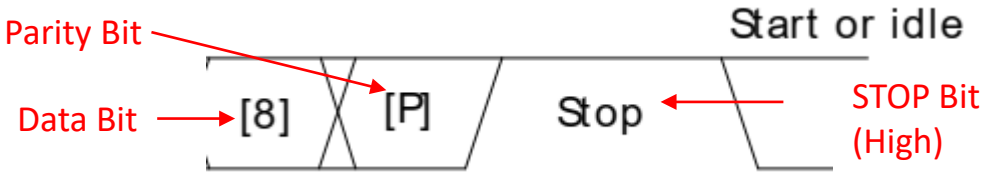
- Start of communications: edge change from H->L for 1 bit duration
  - Start of frame bit (Start bit)
  - Used for synchronization



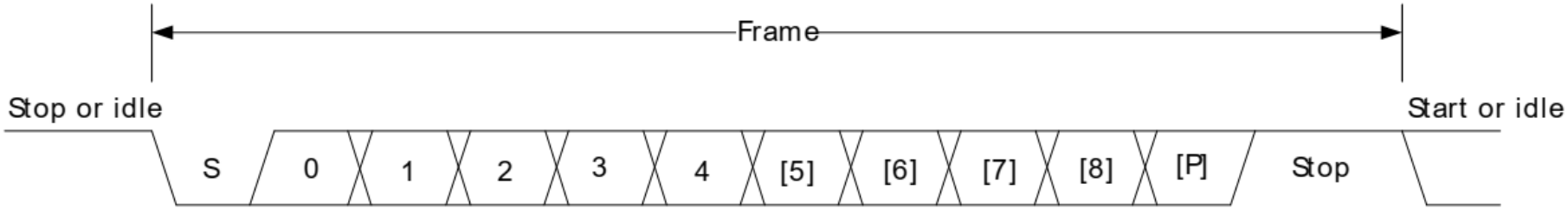
- Data bits: from 4 up to 16 data bits
- Parity bit (P): optional
  - Used for error detection->error is not corrected
  - Even parity: count of 1-bits is even->P=0, otherwise P=1
  - Odd parity: count of 1-bits is odd->P=0, otherwise P=1
- End of communications: edge change from L->H

UART Communications

- End of communications: line is High for 1 or 1.5 or 2 bit duration



- Full frame:



Refer to page 451 of 03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf
(Full USART: pp. 449-495)

UART connection

- Checking the datasheet (for IC specific info) is a must -> see page 13.

---

7 Board Controller

The kit contains a board controller that is responsible for performing various board level tasks, such as handling the debugger and the Advanced Energy Monitor. An interface is provided between the EFM32 and the board controller in the form of a UART connection. The connection is enabled by setting the EFM_BC_EN (PF7) line high, and using the lines EFM_BC_TX (PE0) and EFM_BC_RX (PE1) for communicating.

Specific library functions has been provided in the kit Board Support Package that supports various requests to be made to the board controller, such as quering AEM voltage or current. To use these functions, the Board Support Package must be installed. See the Chapter 8 to find out more.

Note

The board controller is only available when USB power is connected.

Refer to page 13 of 01_EFM32_User_guide_efm32gg-stk3700-user_guide.pdf

UART connection

- Also see page 14.

---

8 Board Support Package

The Board Support Package (BSP) is a set of C source and header files that enables easy access to, and control over some board specific features.

Compared to the Energy Micro development kit, the functionality is limited. Unless you need/want some of the functions contained in the BSP, there is really no need to include or use it. The EFM32 in the Starter Kit is fully usable without BSP support, and you can use all peripherals in the emlib without the BSP.

The BSP use EFM32 peripheral UART0, Location 1 (TX pin PE0, RX pin PE1) on baudrate 115200-8-N-1 to communicate with the board controller.

Note

The BSP is only functional when the Starter Kit is USB-powered, using these function calls with USB disconnected will give unpredictable results.

Refer to page 14 of 01_EFM32_User_guide_efm32gg-stk3700-user_guide.pdf

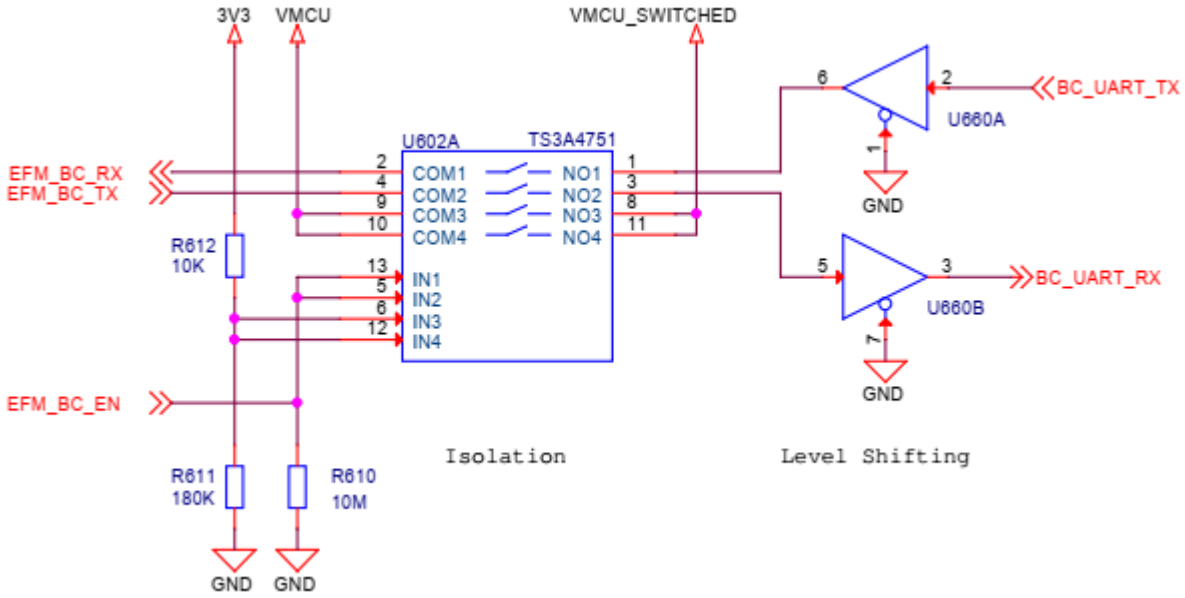
UART connection on uC

- Checking the schematic

Port E (PE) connections:

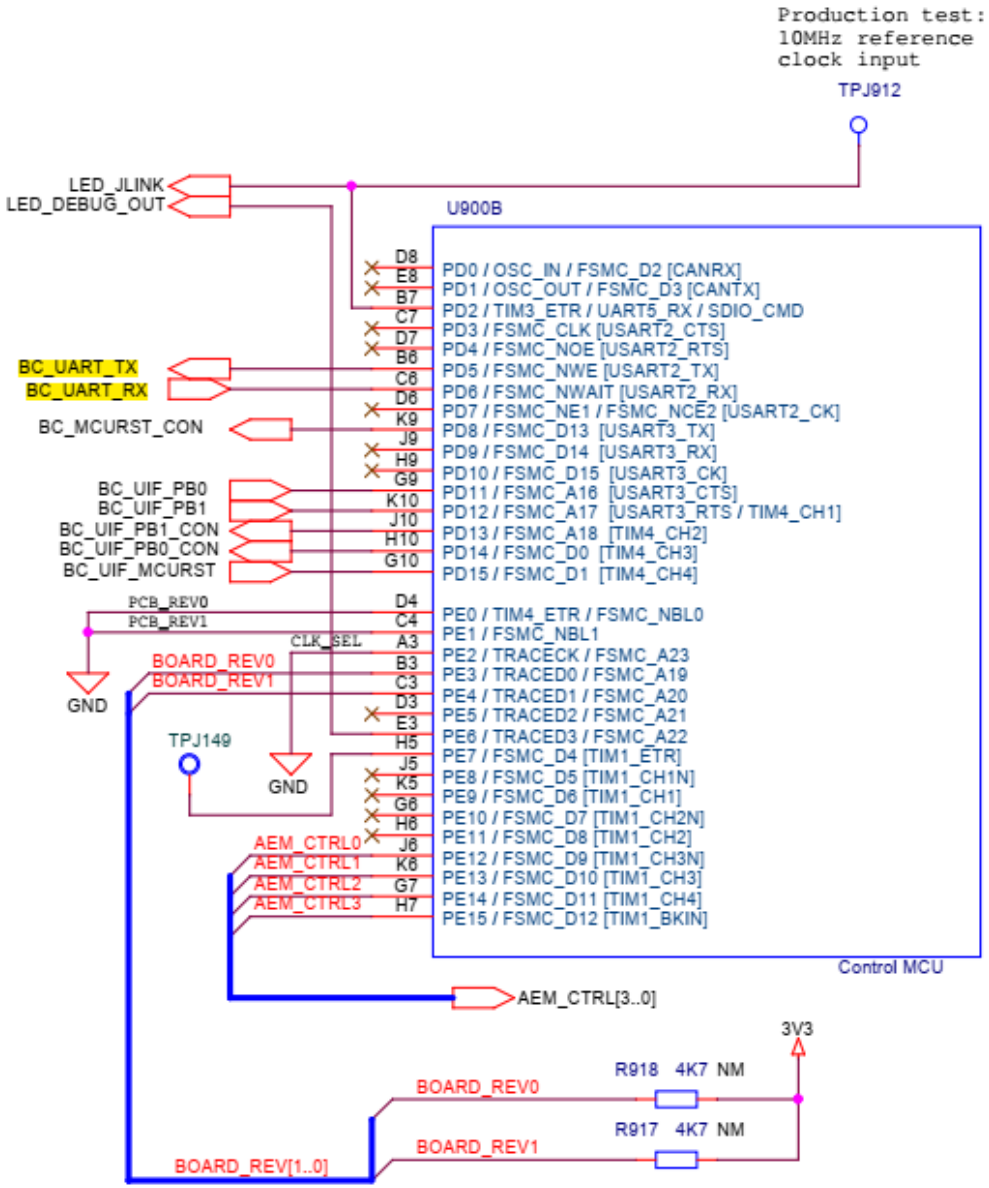
Port F (PF) connections:

Enabling UART:



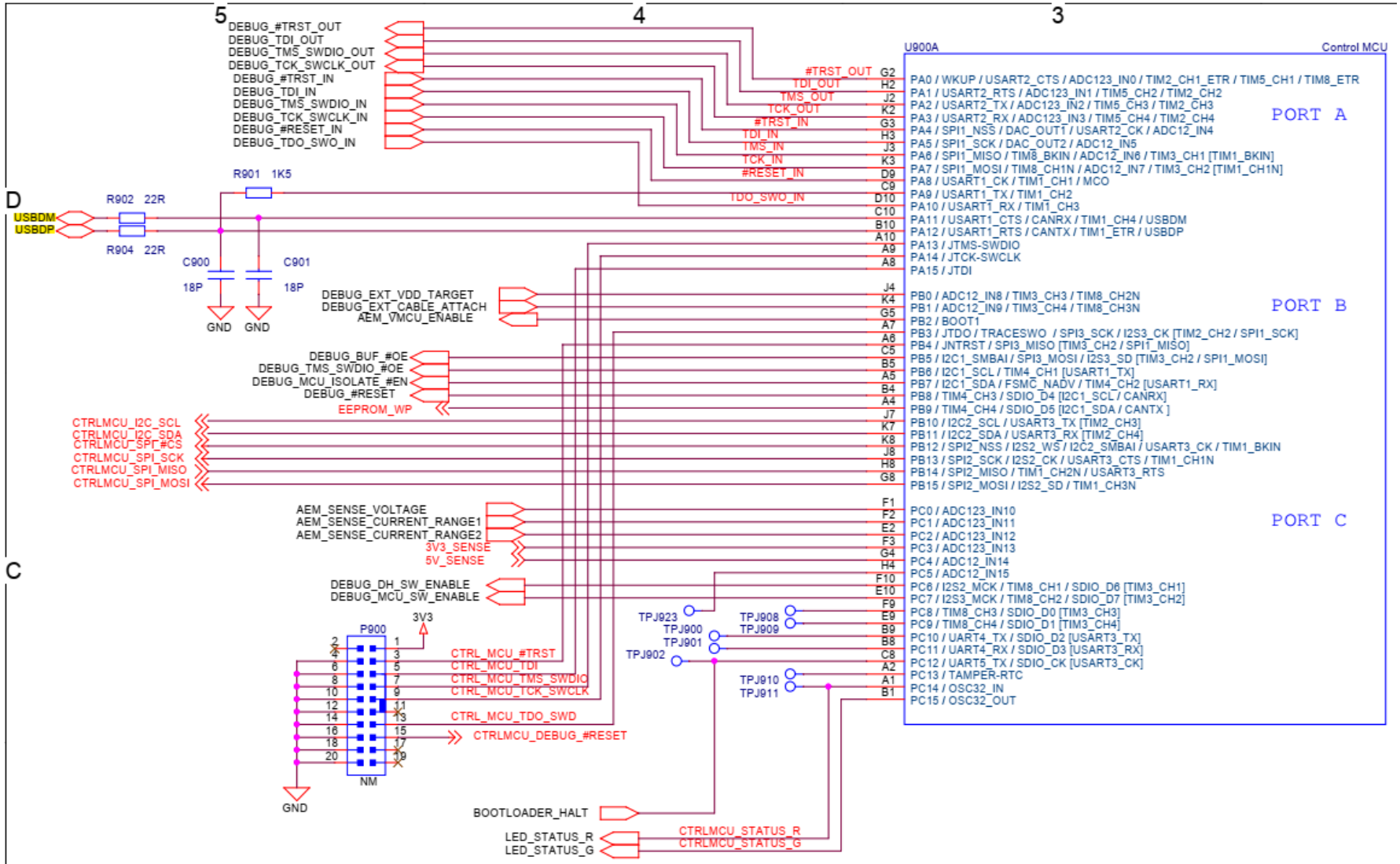
UART connection – Board Controller

Board Controller:



UART connection – Board Controller

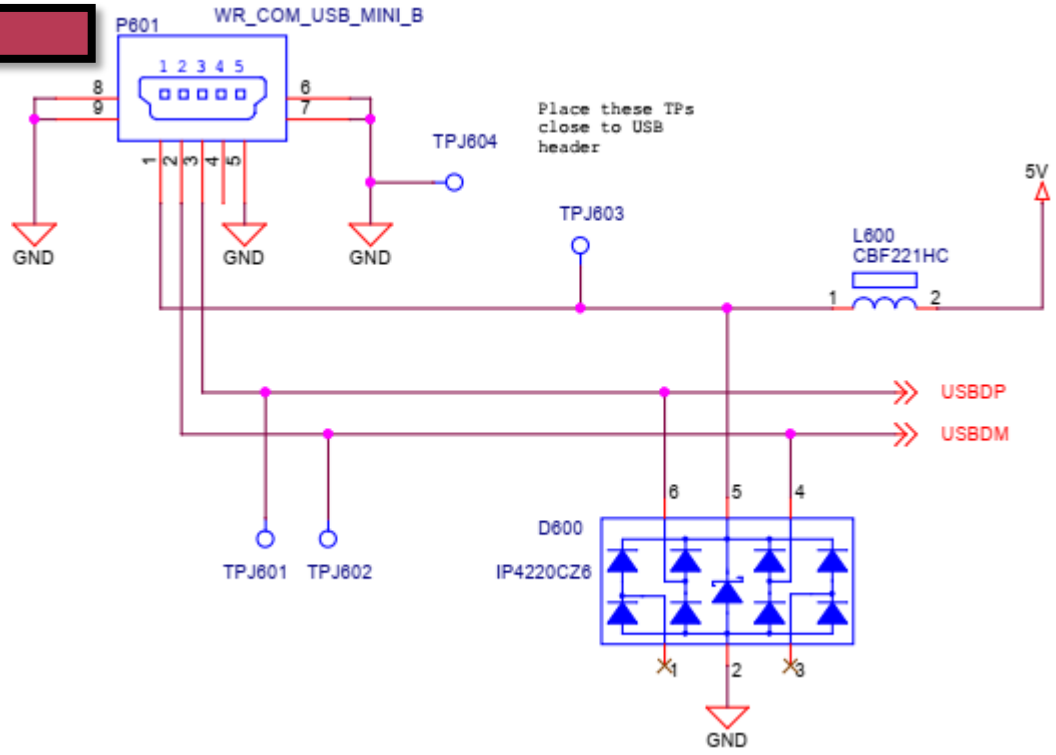
Board Controller:



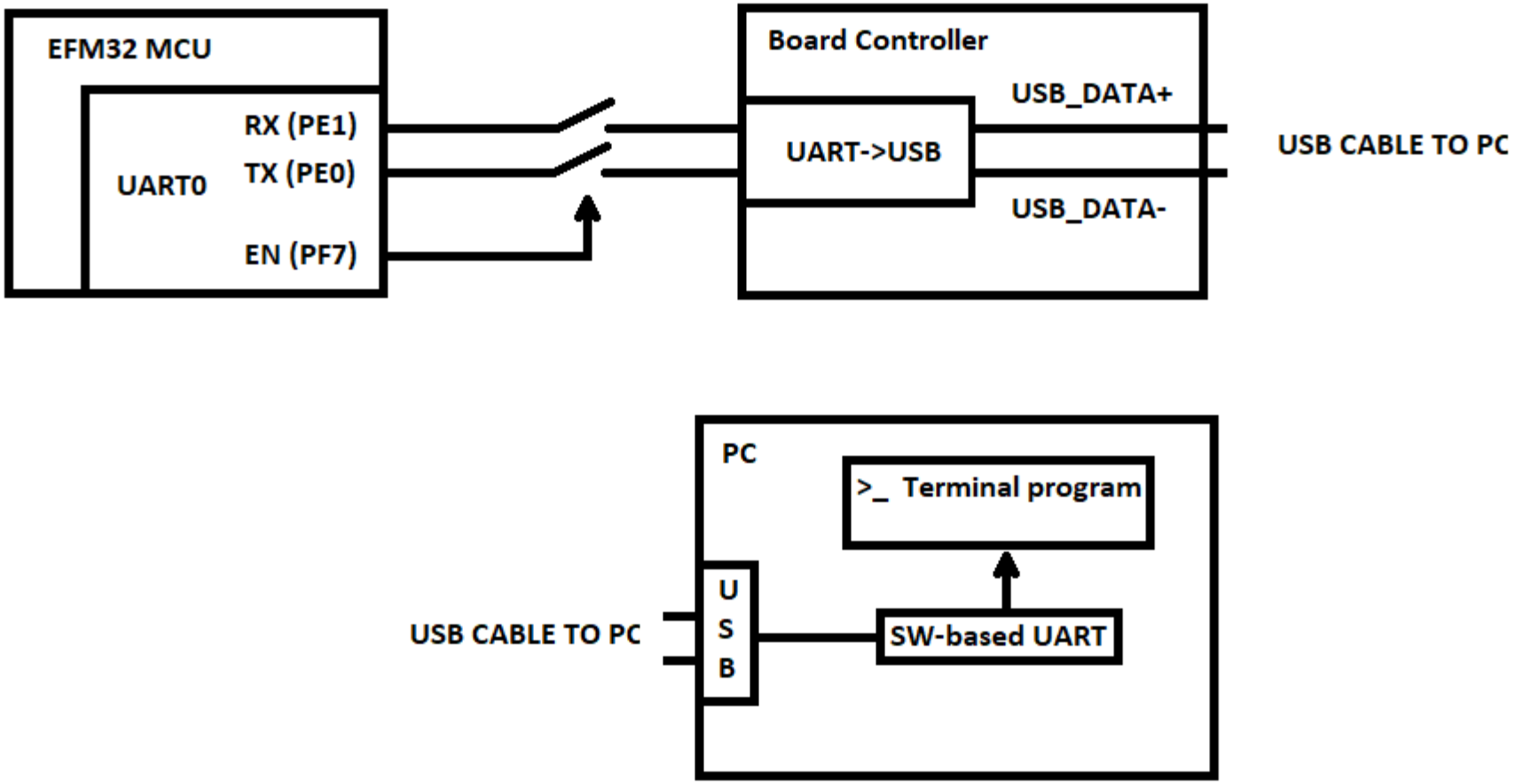
UART connection – USB PHY

PC via USB cable

USB:

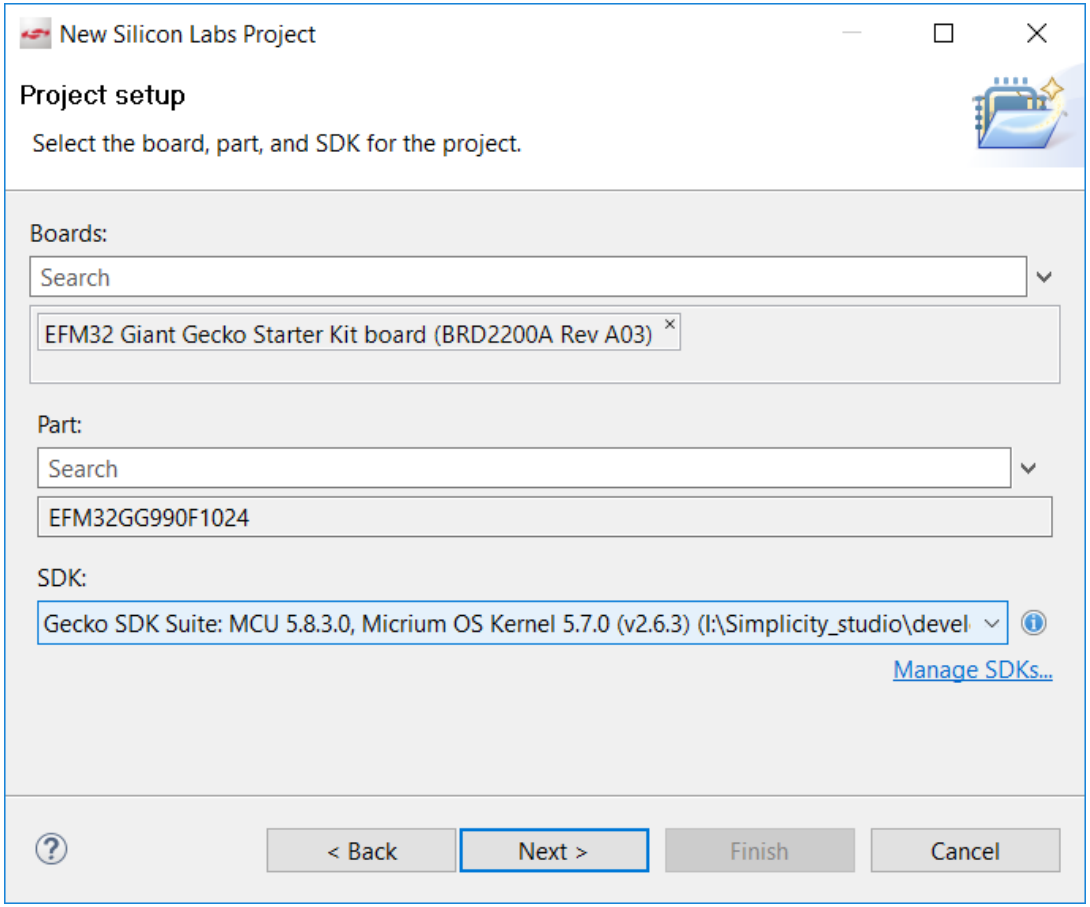


UART connection – Block diagram



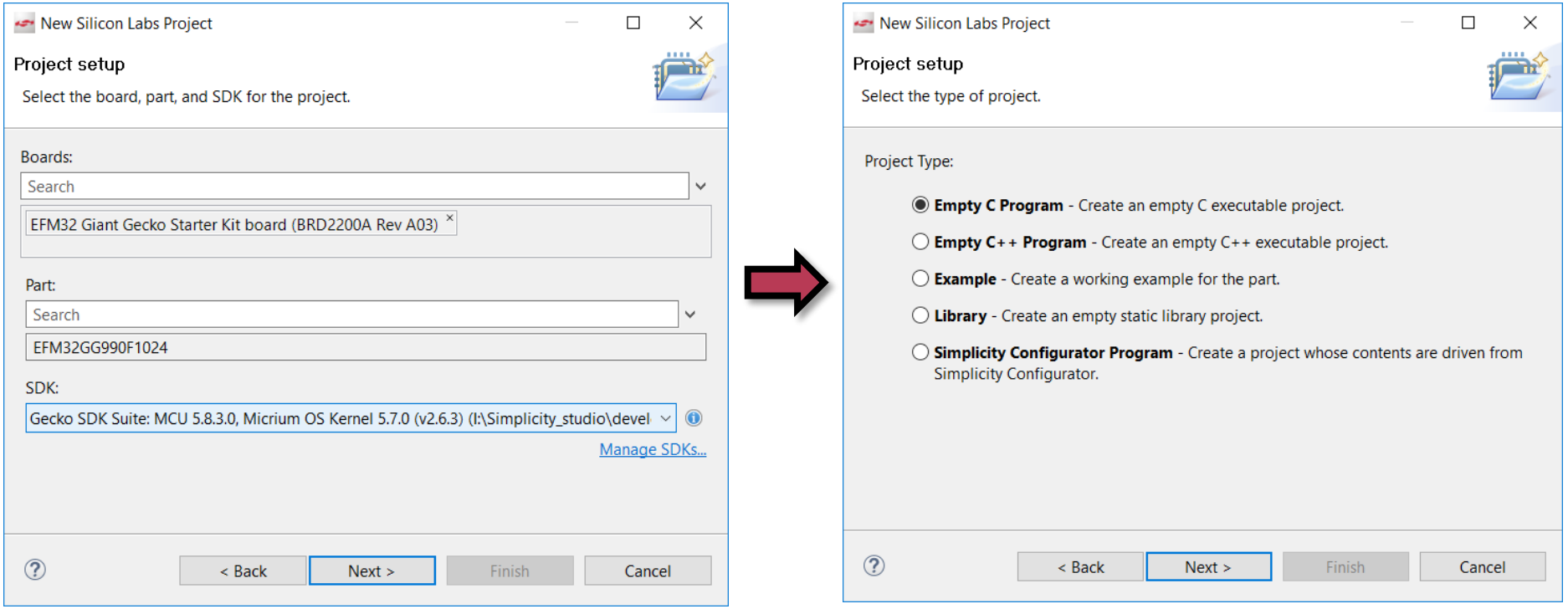
Strating with a new project

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



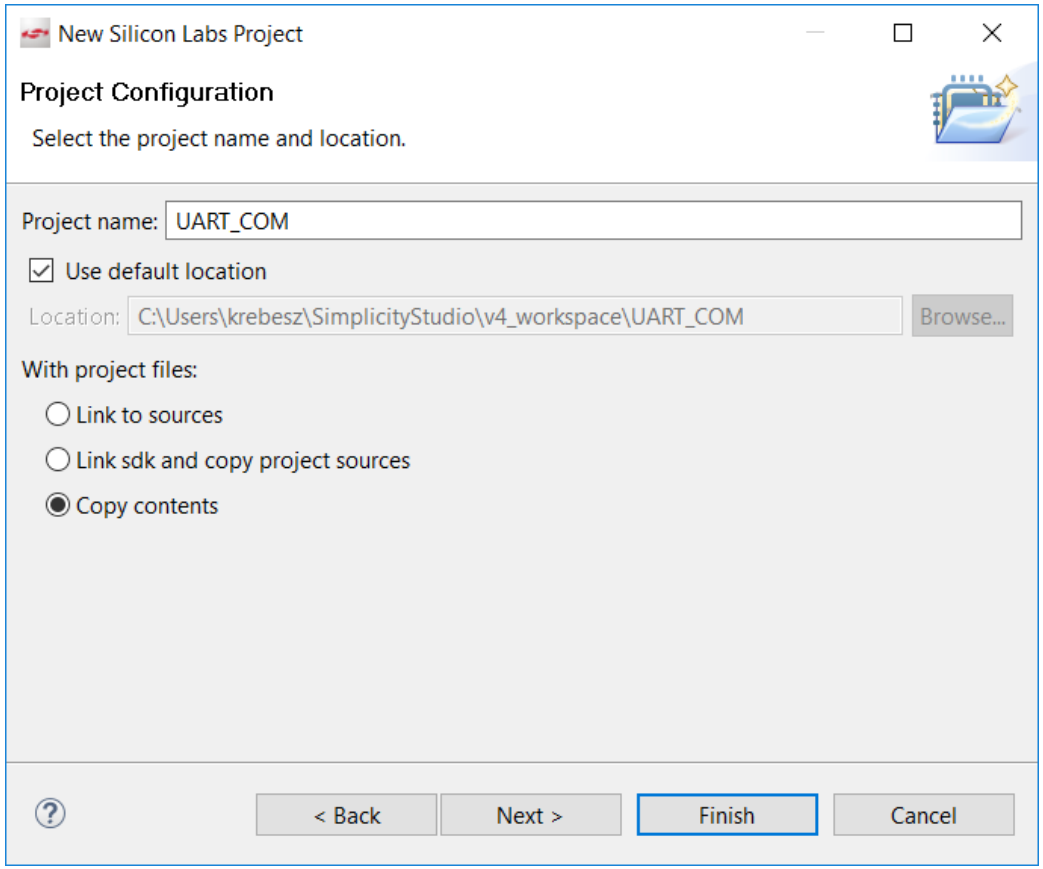
Strating with a new project

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



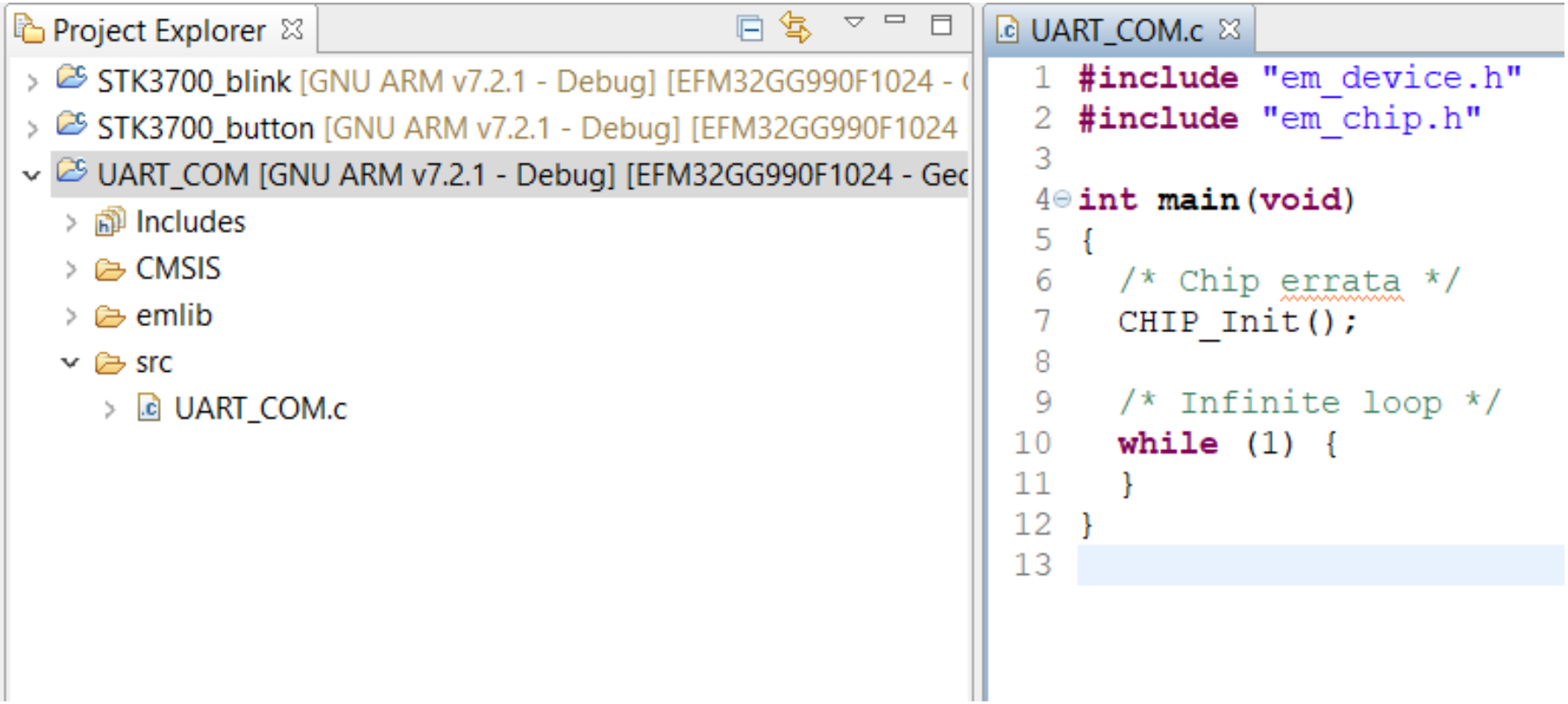
Strating with a new project

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



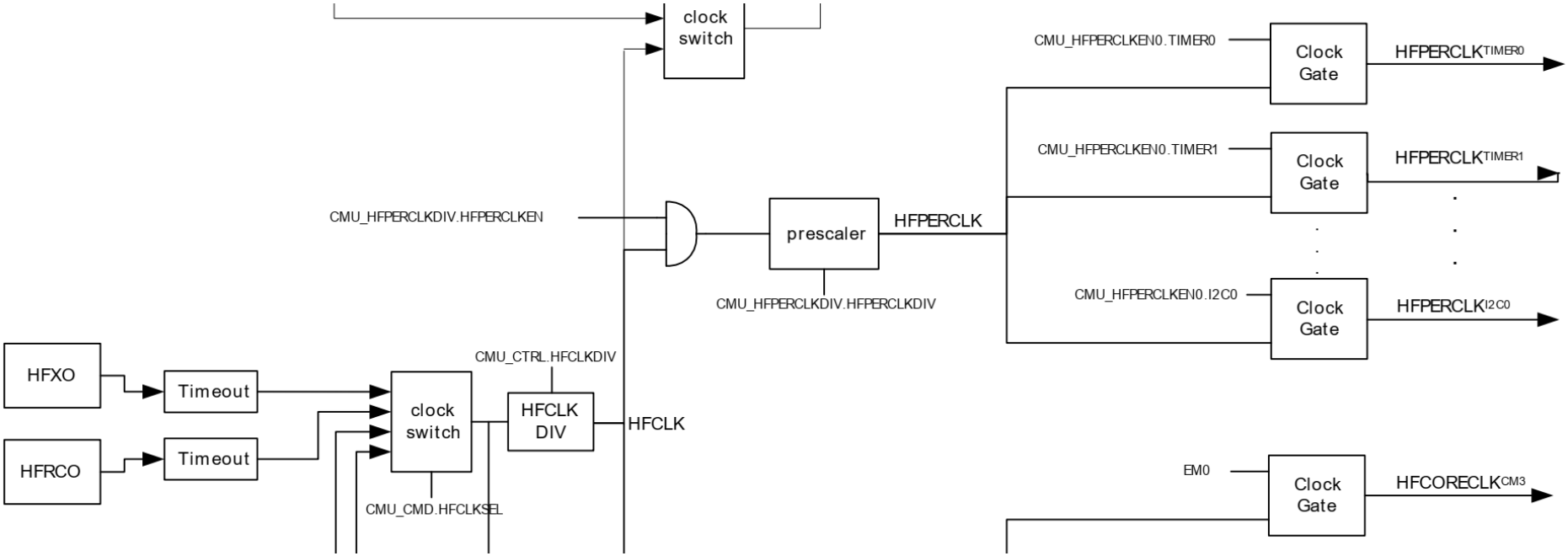
Project created – start programming

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



CLK for GPIO peripheral (CMU system)

- Every peripheral has and needs a CLK to operate



Refer to page 128 of
03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf

CLK for GPIO peripheral

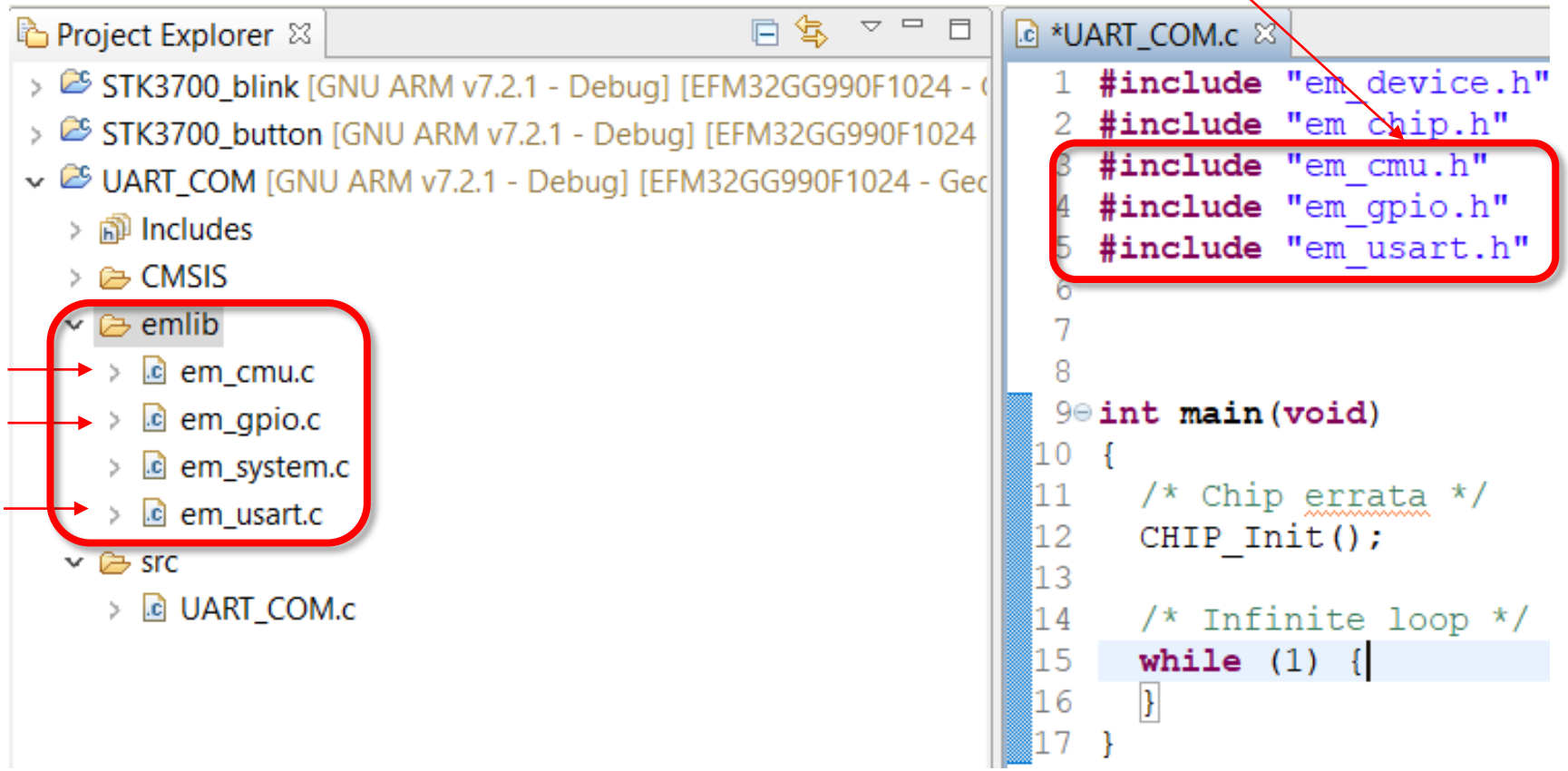
- CLK for GPIO peripheral must be enabled
- Search the library where Simplicity Studio is installed
  - Contains include (inc: *.c) and source (src: *.h) files:

i:\Simplicity_studio\developer\sdks\gecko_sdk_suite\v2.6\platform\emlib\

- Following files has to be drag-and-dropped into emlib library of the project (see next slide):
  - em_cmu.c (clock management unit)
  - em_gpio.c
  - em_usart.c

CLK for GPIO peripheral

- Furthermore they have to be included into the program:



- Check how the CLK for GPIO can be enabled:

CLK for GPIO peripheral (check .h files)

- In programming window click on em_device.h and press F3 -> em_device.h opens
- Defines for different processors from EFM32 family are found -> search for your own type (EFM32GG990F1024):

*UART_COM.c | em_device.h

```
147 #elif defined(EFM32GG942F1024)
148 #include "efm32gg942f1024.h"
149
150 #elif defined(EFM32GG942F512)
151 #include "efm32gg942f512.h"
152
153 #elif defined(EFM32GG980F1024)
154 #include "efm32gg980f1024.h"
155
156 #elif defined(EFM32GG980F512)
157 #include "efm32gg980f512.h"
158
159 #elif defined(EFM32GG990F1024)
160 #include "efm32gg990f1024.h"
```

CLK for GPIO peripheral (check .h files)

- Click on EFM32GG990F1024.h and press F3
- EFM32GG990F1024.h contains (among others)
  - IT number that belongs to a certain peripheral

```
69 /******  EFM32G Peripheral Interrupt Numbers ********************
70   DMA_IRQn                = 0,   /*!< 0 EFM32 DMA Interrupt */
71   GPIO_EVEN_IRQn          = 1,   /*!< 1 EFM32 GPIO_EVEN Interrupt */
72   TIMER0_IRQn             = 2,   /*!< 2 EFM32 TIMER0 Interrupt */
```

  - Memory addresses, e.g. base addresses

```
359 #define CMU_BASE          (0x400C8000UL) /**< CMU base address  */
360 #define GPIO_BASE         (0x40006000UL) /**< GPIO base address  */
```

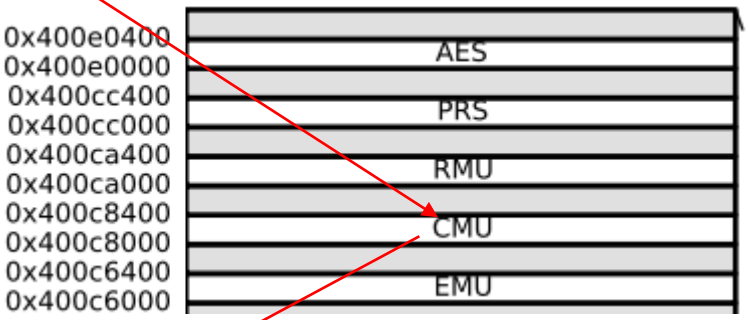
    - No need to check reference manual for e.g. base addresses
      - Refer to page 17 of 03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf to base addresses

CLK for GPIO peripheral (check .h files)

- Defines types that are pointers for the base address

```
410 #define CMU          ((CMU_TypeDef *) CMU_BASE)          /**< CMU base pointer */
411 #define GPIO         ((GPIO_TypeDef *) GPIO_BASE)        /**< GPIO base pointer */
```

  - Click on (CMU_TypeDef *) and press F3
    - Type definition of CMU pops-up in efm32gg_cmu.h which is a structure
    - Elegant solution




```
typedef struct {
  __IOM uint32_t CTRL;
  __IOM uint32_t HFCORECLKDIV;
.
.
.
} CMU_TypeDef;
```

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

Offset	Name	Type	Description
0x000	CMU_CTRL	RW	CMU Control Register
0x004	CMU_HFCORECLKDIV	RW	High Frequency Core Clock Division Register

Elements of structure is assigned to the memory registers via base-address pointer!

CLK for GPIO peripheral (check .h files)

In the header file:

```
typedef struct {
  __IOM uint32_t CTRL;
  __IOM uint32_t HFCORECLKDIV;
  __IOM uint32_t HFPERCLKDIV;
  __IOM uint32_t HFRCOCTRL;
  __IOM uint32_t LFRCOCTRL;
  __IOM uint32_t AUXHFRCOCTRL;
  __IOM uint32_t CALCTRL;
  __IOM uint32_t CALCNT;
  __IOM uint32_t OSCENCMD;
  __IOM uint32_t CMD;
  __IOM uint32_t LFCLKSEL;
  __IM uint32_t  STATUS;
  __IM uint32_t  IF;
  __IOM uint32_t IFS;
  __IOM uint32_t IFC;
  __IOM uint32_t IEN;
  __IOM uint32_t HFCORECLKEN0;
  __IOM uint32_t HFPERCLKEN0;
  uint32_t       RESERVED0[2U];
  __IM uint32_t  SYNCBUSY;
  __IOM uint32_t FREEZE;
  __IOM uint32_t LFACLKEN0;
  uint32_t       RESERVED1[1U];
  __IOM uint32_t LFBCLKEN0;

  uint32_t       RESERVED2[1U];
  __IOM uint32_t LFAPRESC0;
  uint32_t       RESERVED3[1U];
  __IOM uint32_t LFBPRESC0;
  uint32_t       RESERVED4[1U];
  __IOM uint32_t PCNTCTRL;
  __IOM uint32_t LCDCTRL;
  __IOM uint32_t ROUTE;
  __IOM uint32_t LOCK;
} CMU_TypeDef;
```

In the RM see p.136 (03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf):

0x000	CMU_CTRL	RW	CMU Control Register
0x004	CMU_HFCORECLKDIV	RW	High Frequency Core Clock Division Register
0x008	CMU_HFPERCLKDIV	RW	High Frequency Peripheral Clock Division Register
0x00C	CMU_HFRCOCTRL	RW	HFRCO Control Register
0x010	CMU_LFRCOCTRL	RW	LFRCO Control Register
0x014	CMU_AUXHFRCOCTRL	RW	AUXHFRCO Control Register
0x018	CMU_CALCTRL	RW	Calibration Control Register
0x01C	CMU_CALCNT	RWH	Calibration Counter Register
0x020	CMU_OSCENCMD	W1	Oscillator Enable/Disable Command Register
0x024	CMU_CMD	W1	Command Register
0x028	CMU_LFCLKSEL	RW	Low Frequency Clock Select Register
0x02C	CMU_STATUS	R	Status Register
0x030	CMU_IF	R	Interrupt Flag Register
0x034	CMU_IFS	W1	Interrupt Flag Set Register
0x038	CMU_IFC	W1	Interrupt Flag Clear Register
0x03C	CMU_IEN	RW	Interrupt Enable Register
0x040	CMU_HFCORECLKEN0	RW	High Frequency Core Clock Enable Register 0
0x044	CMU_HFPERCLKEN0	RW	High Frequency Peripheral Clock Enable Register 0
0x050	CMU_SYNCBUSY	R	Synchronization Busy Register
0x054	CMU_FREEZE	RW	Freeze Register
0x058	CMU_LFACLKEN0	RW	Low Frequency A Clock Enable Register 0 (Async Reg)
0x060	CMU_LFBCLKEN0	RW	Low Frequency B Clock Enable Register 0 (Async Reg)
0x068	CMU_LFAPRESC0	RW	Low Frequency A Prescaler Register 0 (Async Reg)
0x070	CMU_LFBPRESC0	RW	Low Frequency B Prescaler Register 0 (Async Reg)
0x078	CMU_PCNTCTRL	RW	PCNT Control Register
0x07C	CMU_LCDCTRL	RW	LCD Control Register
0x080	CMU_ROUTE	RW	I/O Routing Register
0x084	CMU_LOCK	RW	Configuration Lock Register

CLK for GPIO peripheral

- Using the structure CMU
  - CMU is a structure pointer: arrow is used ->
  - *CMU->* (Ctrl+Space will complement)
    - Needed: HFPERCLKEN0 (Bit 13 is used for GPIO CLK)

11.5.18 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

See ref.man. P150:

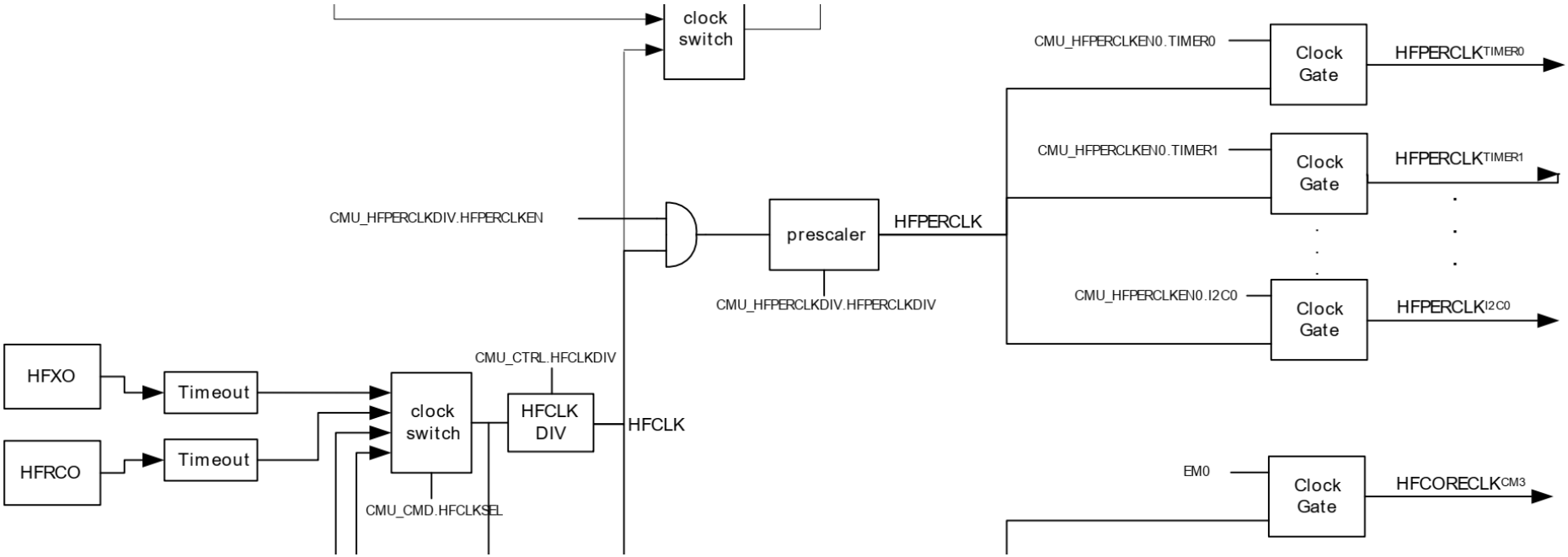
Offset	Bit Position																														
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access															RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name															DAC0	ADC0	PRS	VCMP	GPIO	I2C1	I2C0	ACMP1	ACMP0	TIMER3	TIMER2	TIMER1	TIMER0	UART1	UART0	USART2	USART1	USART0

- A define is available for Bit 13 in efm32gg_cmu.h

```
919 #define CMU_HFPERCLKEN0_GPIO                    (0x1UL << 13)
```

CLK for GPIO peripheral (CMU system)

- Every peripheral has and needs a CLK to operate

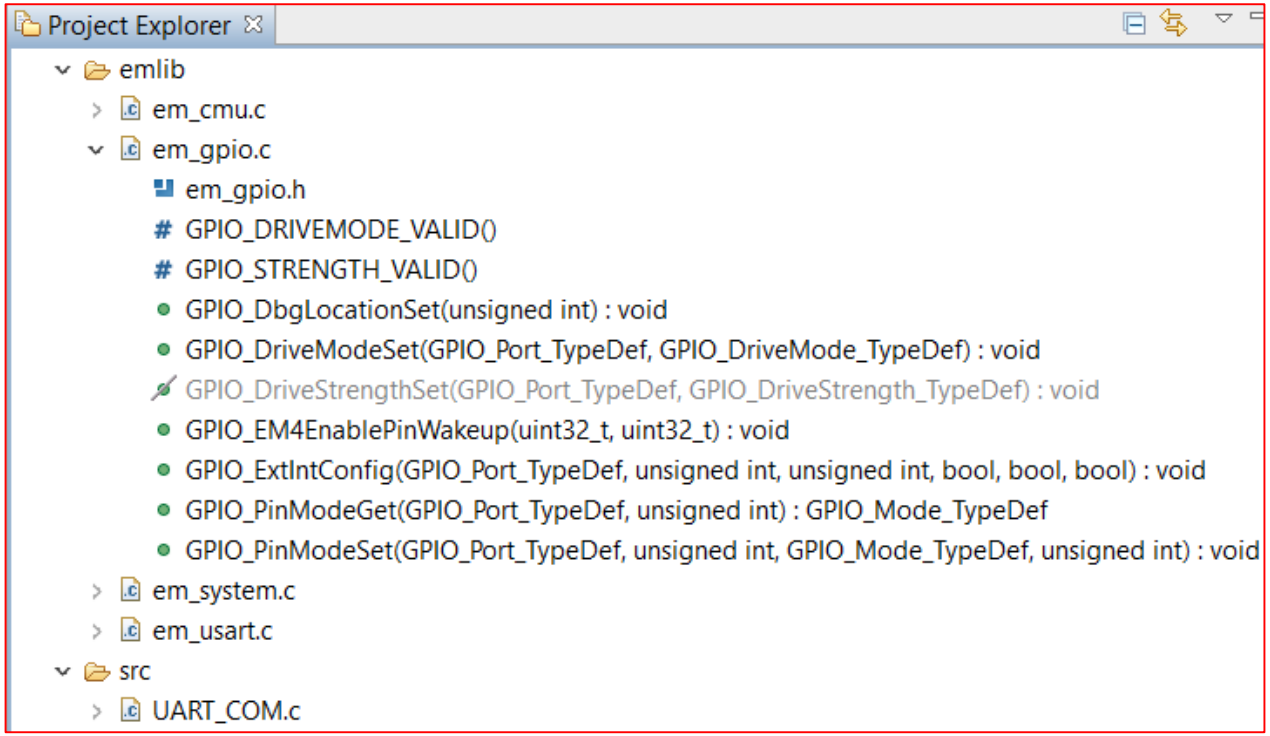


Refer to page 128 of
03_EFM32_Reference_manual_EFM32GG-reference_manual.pdf

- Code to be used:
  - CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

Setting GPIO in code (enable)

- Remember: PF7=1 has to be set
- More elegant approach if a function can be found for a problem -> Results in more readable code
  - Check functions under em_gpio.c in the project



Setting GPIO in code (enable)

- Open (double click on) **GPIO_PinModeSet**
  - em_gpio.c opens at the function implementation
  - Remark: em_gpio.h also contains the definition of functions and even more, e.g. static functions available only in header files
  - Note: these functions are independent of the type of processor, since the processor dependent specialities are defined in efm32gg_xxx.h
    - Helps to develop portable code that is compatible with other processors (from the same processor family at least)
  - Hint: copy the function and paste it into code; make it one-line; comment the orig. and make a work copy

Setting GPIO in code (enable)

- Read the function description: placed above the function definition

```
/*******************************************************************************//**
 * @brief
 *   Set the mode for a GPIO pin.
 *
 * @param[in] port
 *   The GPIO port to access.
 *
 * @param[in] pin
 *   The pin number in the port.
 *
 * @param[in] mode
 *   The desired pin mode.
 *
 * @param[in] out
 *   A value to set for the pin in the DOUT register. The DOUT setting is important for
 *   some input mode configurations to determine the pull-up/down direction.
 ******************************************************************************/
void GPIO_PinModeSet(GPIO_Port_TypeDef port,
                     unsigned int pin,
                     GPIO_Mode_TypeDef mode,
                     unsigned int out)
```

Setting GPIO in code (enable)

- Function to be used:

```
void GPIO_PinModeSet(GPIO_Port_TypeDef port,
                     unsigned int pin,
                     GPIO_Mode_TypeDef mode,
                     unsigned int out);
```

  - GPIO_Port_TypeDef + F3

```
/** GPIO ports IDs. */
typedef enum {
#if (_GPIO_PORT_A_PIN_COUNT > 0)
  gpioPortA = 0,
#endif
#if (_GPIO_PORT_B_PIN_COUNT > 0)
  gpioPortB = 1,
#endif
#if (_GPIO_PORT_C_PIN_COUNT > 0)
  gpioPortC = 2,
#endif
#if (_GPIO_PORT_D_PIN_COUNT > 0)
  gpioPortD = 3,
#endif
#if (_GPIO_PORT_E_PIN_COUNT > 0)
  gpioPortE = 4,
#endif
#if (_GPIO_PORT_F_PIN_COUNT > 0)
  gpioPortF = 5,
#endif
} GPIO_Port_TypeDef;
```

Use the names given in the enum type definition

Setting GPIO in code (enable)

- Function to be used:

```
void GPIO_PinModeSet(GPIO_Port_TypeDef port,
                     unsigned int pin,
                     GPIO_Mode_TypeDef mode,
                     unsigned int out);
```

  - pin – port number, now it is 7
    - No specific name is given
  - GPIO_Mode_TypeDef + F3

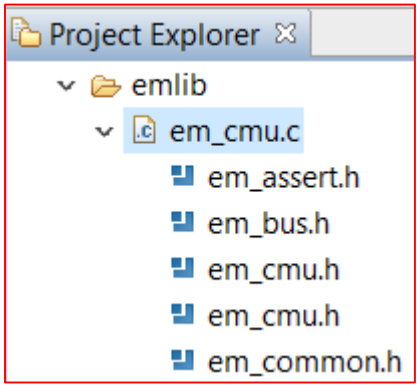
```
typedef enum {
  /** Input disabled. Pull-up if DOUT is set. */
  gpioModeDisabled                  = _GPIO_P_MODEL_MODE0_DISABLED,
  /** Input enabled. Filter if DOUT is set. */
  gpioModeInput                     = _GPIO_P_MODEL_MODE0_INPUT,
  /** Input enabled. DOUT determines pull direction. */
  gpioModeInputPull                 = _GPIO_P_MODEL_MODE0_INPUTPULL,
  /** Input enabled with filter. DOUT determines pull direction. */
  gpioModeInputPullFilter           = _GPIO_P_MODEL_MODE0_INPUTPULLFILTER,
  /** Push-pull output. */
  gpioModePushPull                  = _GPIO_P_MODEL_MODE0_PUSHPULL,
```

  - out – initial value of pin, use 1

Use the names given in the enum type definition

Setting the UART (CLK)

- CLK is needed again!
  - Already used approach is also possible: setting CMU register
  - Better way is using a function for that purpose
    - Check em_cmu.c in the project by unfolding it:
    - Find CMU_ClockEnable among functions
    - Copy the function and paste it into the code:



CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable);
  - enable – it should be true obviously
  - CMU_Clock_TypeDef + F3
    - » cmuClock_UART0 should be used
  - Code to be used:
    CMU_ClockEnable(cmuClock_UART0, true);

Setting the UART (Tx and Rx)

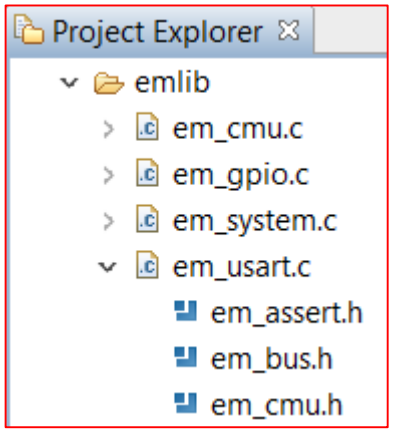
- Remember: port settings for communications
  - PE0 = Tx -> PE0 is output
  - PE1 is Rx -> PE1 is input
- Use the function **GPIO_PinModeSet** again
  - GPIO_PinModeSet(*gpioPortF,7,gpioModePushPull,1);*
    - Used for setting PF7 into 1 to enable the UART comm.
  - GPIO_PinModeSet(*gpioPortE,0,gpioModePushPull,1);*
    - See changes in red for setting Tx line (PE0 is now output)
  - GPIO_PinModeSet(*gpioPortE,1,gpioModeInput,1);*
    - See changes in red for setting Rx line (PE1 is now input)

Delete back until *gpioMode*, then push F3

This boolean is don't care now

Configuration of the UART

- Check em_usart.c in project explorer
  - Find USART_InitAsync and double click
    - em_usart.c opens at the function implement.
    - Read description of the function
    - Copy the function and paste it into the code
      - USART_InitAsync(USART_TypeDef *usart, const USART_InitAsync_TypeDef *init)



- USART_InitAsync()
  - USART_TypeDef + F3 : it is a structure again defined in efm32gg_usart.h
    - Remember that a pointer is used here!
      - check out for its define in efm32gg990F1024.h

Configuration of the UART

– Define of USART_TypeDef in efm32gg990F1024.h

```
398 #define USART0        ((USART_TypeDef *) USART0_BASE)        /**< USART0 base pointer */
399 #define USART1        ((USART_TypeDef *) USART1_BASE)        /**< USART1 base pointer */
400 #define USART2        ((USART_TypeDef *) USART2_BASE)        /**< USART2 base pointer */
401 #define UART0         ((USART_TypeDef *) UART0_BASE)         /**< UART0 base pointer */
402 #define UART1         ((USART_TypeDef *) UART1_BASE)         /**< UART1 base pointer */
```

– More than only one USART is available: USART0 is our choice (& is not needed since it is a pointer: see later)

o USART_InitAsync_TypeDef + F3

- Important parameters for the USART
- Unfortunately this structure is not existing, therefore it has to be implemented
  – implementation is advised before the main function in the .c main file as a global variable. In this case its initial value becomes zero while when implementation is done inside the main function it fills up the structure with memory garbage
  – USART_InitAsync_TypeDef UART0_init; ← It can be any name
- Not UART0_init is used but a memory address: &UART0_init

Configuration of the UART

- Function to be used in the code:
  USART_InitAsync(UART0, &UART0_init);
- UART0_init structure has to be uploaded with values
  - USART_InitAsync_TypeDef + F3 again -> em_usart.h
    - » Stay above the writing and options pop-up

```
/** Asynchronous mode initialization structure. */
typedef struct {
  /** Specifies whether TX and/or RX is enabled when initialization is completed. */
  USART_Enable_TypeDef   enable;
```

```
  /** Disable both receiver and transmitter. */
  usartDisable   = 0x0,

  /** Enable receiver only, transmitter disabled. */
  usartEnableRx = USART_CMD_RXEN,

  /** Enable transmitter only, receiver disabled. */
  usartEnableTx = USART_CMD_TXEN,

  /** Enable both receiver and transmitter. */
  usartEnable   = (USART_CMD_RXEN | USART_CMD_TXEN)
} USART_Enable_TypeDef;
```

```
baud rate setup.
clock.
```

  - Code to be used: UART0_init.enable = usartEnable;

Configuration of the UART

- Same way all the other properties has to be filled up
- Initialization has to be done before using it

```
//CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable);
CMU_ClockEnable(cmuClock_UART0, true);

UART0_init.enable = usartEnable;
UART0_init.refFreq = 0;
UART0_init.baudrate = 115200;
UART0_init.oversampling = usartOVS16;
UART0_init.databits = usartDatabits8;
UART0_init.parity = usartNoParity;
UART0_init.stopbits = usartStopbits1;
UART0_init.mvdis = false;
UART0_init.prsRxEnable = false;
UART0_init.autoCsEnable = false;
//USART_InitAsync(USART_TypeDef *usart, const USART_InitAsync_TypeDef *init)
USART_InitAsync(UART0, &UART0_init);
```

- Note: every name has to checked!
  -> e.g. usartDatabits8 is not equal value 8 but value 5

Configuration of the UART

- Oversampling: see ref.man. page 458:

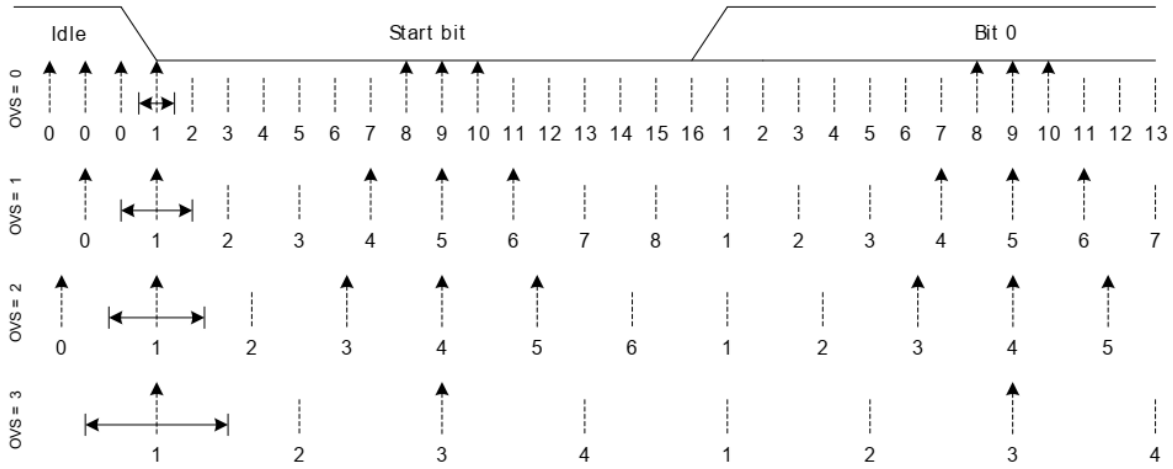
For oversampling modes 16, 8 and 6, every bit in the incoming frame is sampled three times to gain a level of noise immunity. These samples are aimed at the middle of the bit-periods, as visualized in Figure 17.5 (p. 458) . With OVS=0 in USARTn_CTRL, the start and data bits are thus sampled at locations 8, 9 and 10 in the figure, locations 4, 5 and 6 for OVS=1 and locations 3, 4, and 5 for OVS=2. The value of a sampled bit is determined by majority vote. If two or more of the three bit-samples are high, the resulting bit value is high. If the majority is low, the resulting bit value is low.

Majority vote is used for all oversampling modes except 4x oversampling. In this mode, a single sample is taken at position 3 as shown in Figure 17.5 (p. 458) .

Majority vote can be disabled by setting MVDIS in USARTn_CTRL.

If the value of the start bit is found to be high, the reception of the frame is aborted, filtering out false start bits possibly generated by noise on the input.

Figure 17.5. USART Sampling of Start and Data Bits



Configuration of the UART

- The faster way
  - Look for the USART_InitAsync_TypeDef structure (F3) and scroll down in em_usart.h to find

```
#define USART_INITASYNC_DEFAULT                                                \
  {                                                                            \
    usartEnable,            /* Enable RX/TX when initialization is complete. */ \
    0,                      /* Use current configured reference clock for configuring baud rate. */ \
    115200,                 /* 115200 bits/s. */                             \
    usartOVS16,             /* 16x oversampling. */                          \
    usartDatabits8,         /* 8 data bits. */                               \
    usartNoParity,          /* No parity. */                                 \
    usartStopbits1,         /* 1 stop bit. */                                \
    false,                  /* Do not disable majority vote. */              \
    false,                  /* Not USART PRS input mode. */                  \
    0,                      /* PRS channel 0. */                             \
    false,                  /* Auto CS functionality enable/disable switch */ \
  }
```

  - It is a predefined default structure
  - Before the main function it can be used for initialization:
    USART_InitAsync_TypeDef UART0_init = USART_INITASYNC_DEFAULT;

Configuration of the UART

- Interesting difficulty with PE0 and PE1 pins
  - Check datasheet on page 65.

Alternate	LOCATION							
Functionality	0	1	2	3	4	5	6	Description
U0_RX	PF7	PE1	PA4					UART0 Receive input.
U0_TX	PF6	PE0	PA3					UART0 Transmit output. Also used as receive input in half duplex communication.

  - U0_RX and U0_TX default locations are PF7 and PF6, respectively, that has to be changed since the circuit (i.e. the board) has been designed for UART communication at Location 1
    - Datasheet is valid for the IC not for the board but a freedom is given this way for the board designer
  - Location 1 has to be set for correct operation

Configuration of the UART

- Check reference manual at page 492

17.5.22 USARTn_ROUTE - I/O Routing Register

Offset	Bit Position																													
0x054	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																						0x0							0	0	0	0
Access																						RW							RW	RW	RW	RW
Name																						LOCATION							CLKPEN	CSPEN	TXPEN	RXPEN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	LOCATION	0x0	RW	**I/O Location**

Decides the location of the USART I/O pins.

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4
5	LOC5	Location 5

Configuration of the UART

- Check reference manual at page 492

17.5.22 USARTn_ROUTE - I/O Routing Register

Offset	Bit Position																											
0x054	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																						0x0							0	0	0	0
Access																						RW							RW	RW	RW	RW
Name																						LOCATION							CLKPEN	CSPEN	TXPEN	RXPEN

Bit	Name	Reset	Access	Description
1	TXPEN	0	RW	**TX Pin Enable**

When set, the TX/MOSI pin of the USART is enabled

Value	Description
0	The U(S)n_TX (MOSI) pin is disabled
1	The U(S)n_TX (MOSI) pin is enabled

Bit	Name	Reset	Access	Description
0	RXPEN	0	RW	**RX Pin Enable**

When set, the RX/MISO pin of the USART is enabled.

Value	Description
0	The U(S)n_RX (MISO) pin is disabled
1	The U(S)n_RX (MISO) pin is enabled

Configuration of the UART

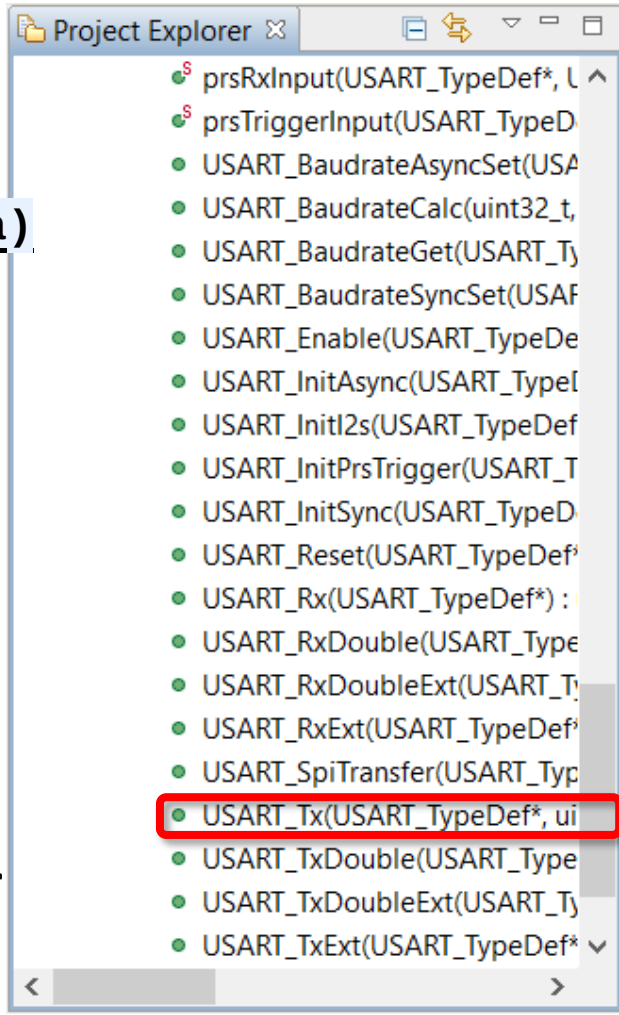
- Setting the I/O Routing register (i) for LOC1 (PE0 and PE1 pins for UART communication) and enabling these lines for transmission and reception of serial data
  - `UART0->ROUTE |= (1) << 8;`
    - Although correct but not too informative
  - A definition can be used for this purpose in efm32gg_usart.h (search for 'LOC1')
    - `#define USART_ROUTE_LOCATION_LOC1 (_USART_ROUTE_LOCATION_LOC1 << 8)`
  - `UART0->ROUTE |= USART_ROUTE_LOCATION_LOC1;`

Configuration of the UART

- Enabling RX and TX via RXPEN and TXPEN bits respectively
  - A definition can be used for this purpose in efm32gg_usart.h (search for ‘RXPEN’ and ‘TXPEN’)
    - `UART0->ROUTE |=(USART_ROUTE_RXPEN | USART_ROUTE_TXPEN);`
- Everything is ready for sending data via UART

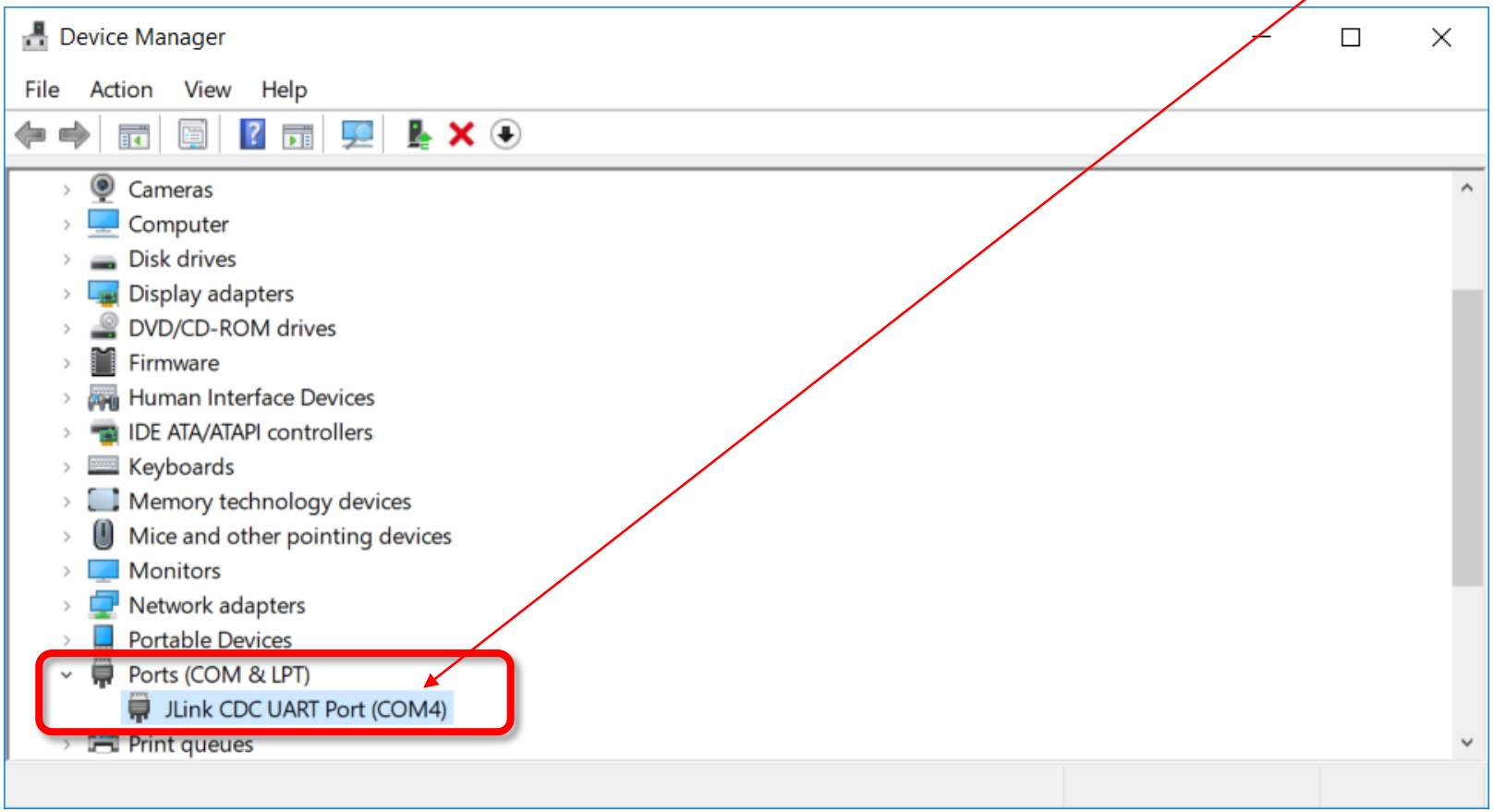
Sending data via UART

- In Project Explorer window under emlib->em_usart.c you can find
  - `USART_Tx(USART_TypeDef *usart, uint8_t data)`
- Code to be inserted:
  - `USART_Tx(UART0, '+');`
  - We send ‘+’ signal via UART0
  - Good idea to check the compilation
- Where is the UART (COMx)?
  - Check in Windows Device Manager



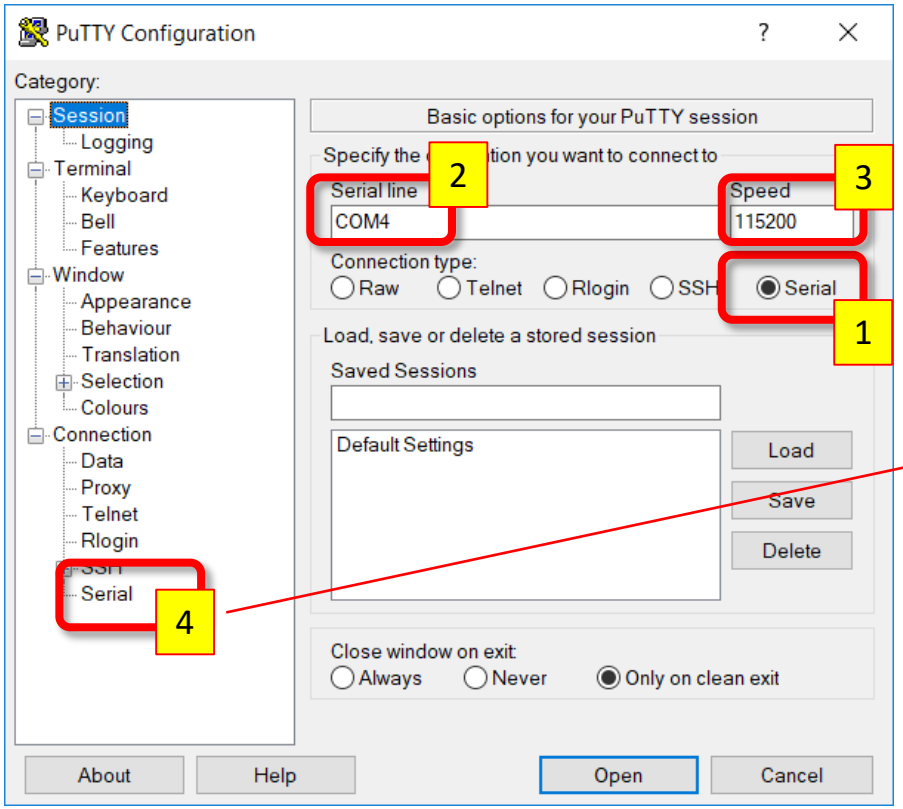
Sending data via UART

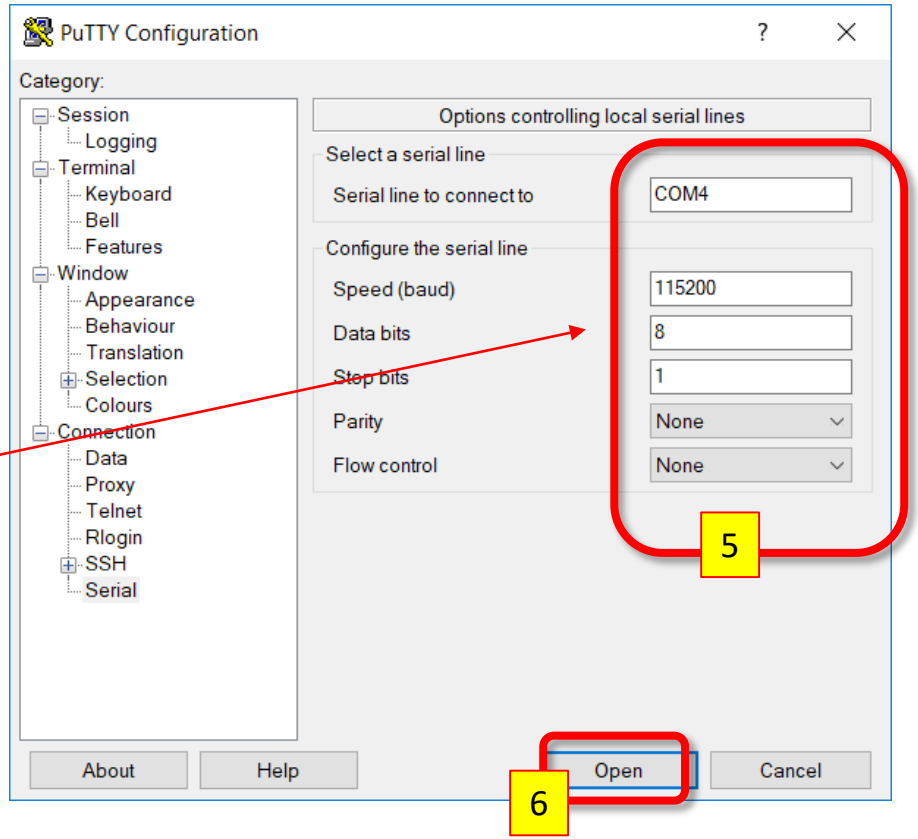
- Check UART (COM port number and its settings) in Device Manager in Windows (now it is COM4)



Sending data via UART

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





Sending data via UART

- The terminal is now open
- Compile and download the code to check operation
  - Has the ‘+’ sign appeared in the terminal window?

Sending data via UART

- This status is the starting point to develop an UART communication-based application
  - E.g. write in the terminal window the character pushed (= read a character from UART0 and send this character to UART0)
    - This function has to be added into the program (in the while loop)
    - `USART_Tx(UART0, USART_Rx(UART0));`
  - Problem: character is received in a blocking way:
    - We are always in the loop waiting for data and no other operation can be done
    - Better if the arrival of new data can be indicated not to stack in the while loop forever (non-blocking solution)

Appendix:program code(a working version)

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

USART_InitAsync_TypeDef UART0_init=USART_INITASYNC_DEFAULT;

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

  CMU->HFPERCLKEN0 |= CMU_HFPERCLKEN0_GPIO;

//CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)
  CMU_ClockEnable(cmuClock_UART0,true);


//GPIO_PinModeSet(GPIO_Port_TypeDef port,unsigned int pin,GPIO_Mode_TypeDef mode,unsigned int out)
  GPIO_PinModeSet(gpioPortF,7,gpioModePushPull,1); //EN
  GPIO_PinModeSet(gpioPortE,0,gpioModePushPull,1); //Tx
  GPIO_PinModeSet(gpioPortE,1,gpioModeInput,1); //Rx

 USART_InitAsync(UART0, &UART0_init);

  UART0->ROUTE |= (1) << 8;
  UART0->ROUTE |= (USART_ROUTE_RXPEN | USART_ROUTE_TXPEN);

  //USART_Tx(USART_TypeDef *usart, uint8_t data)
  USART_Tx(UART0, '+');

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