Embedded and ambient systems
2021.11.10.

Practice 6
Runtime measurement

Méréstechnika és
Információs Rendszerek
Tanszék

Runtime measurement

- Recall debug topic from lecture – timer for meas.

Measurement of runtime using timer

- Options:
  - Starting the timer at the beginning of the code part to be measured and stopping the timer when the code part is finished
  - Starting the timer (even independently of the code part to be measured) and reading its value at the beginning of the code part to be measured then reading the timer value again when the code part to be measured reaches its end. The runtime is the time difference between the two timer values.
- Error: time needed to read timer value increases the runtime
- Core timer: special timer, measures the processor runtime in CLK ticks
- Example for using the core timer:
  - ARM cortex M3 (reading and calculating the difference requires 10 CLK cycles!!!)

```
printStart = DWT->CYCCNT;
printTime = DWT->CYCCNT - printStart;
```

(x)= printStart	uint32_t	0xd7cda8
(x)= printTime	uint32_t	10 (Decimal)

```
#define CYCLE_COUNT_START( cntr ) \
    asm("r0 = emuclk; %0 = r0;": \
    "=k" (cntr):"d" (cntr): \
    "r0")

#define CYCLE_COUNT_STOP( cntr ) \
    asm("r0 = emuclk; r1 = %1; r2 = 4; r0 = r0 - r2; r0 = r0 - r1; %0 = r0;" : \
    "=k" (cntr) : \
    "d" (cntr) : "r0", "r1")
```

Runtime measurement

- uC has a special built in timer/counter inside *Data Watch point and Trace (DWT)* unit of Debug interface
- DWT is a 32-bit one, i.e., using default 14MHz CLK signal the maximum amount of time that can be measured is T_max=2^32/14MHz=5min
- Runtime measurement is actually measurement of CLK cycles that can be easily transformed into time via CLK frequency
- Counter used to measure runtime: Cycle Count Register

Runtime measurement

- Counter used to measure runtime: Cycle Count Register (CYCCNT)
  - When processor starts CYCCNT is zero
  - Register can be accessed in the following way:
    - DWT -> CYCCNT
  - A possible solution to measure runtime:

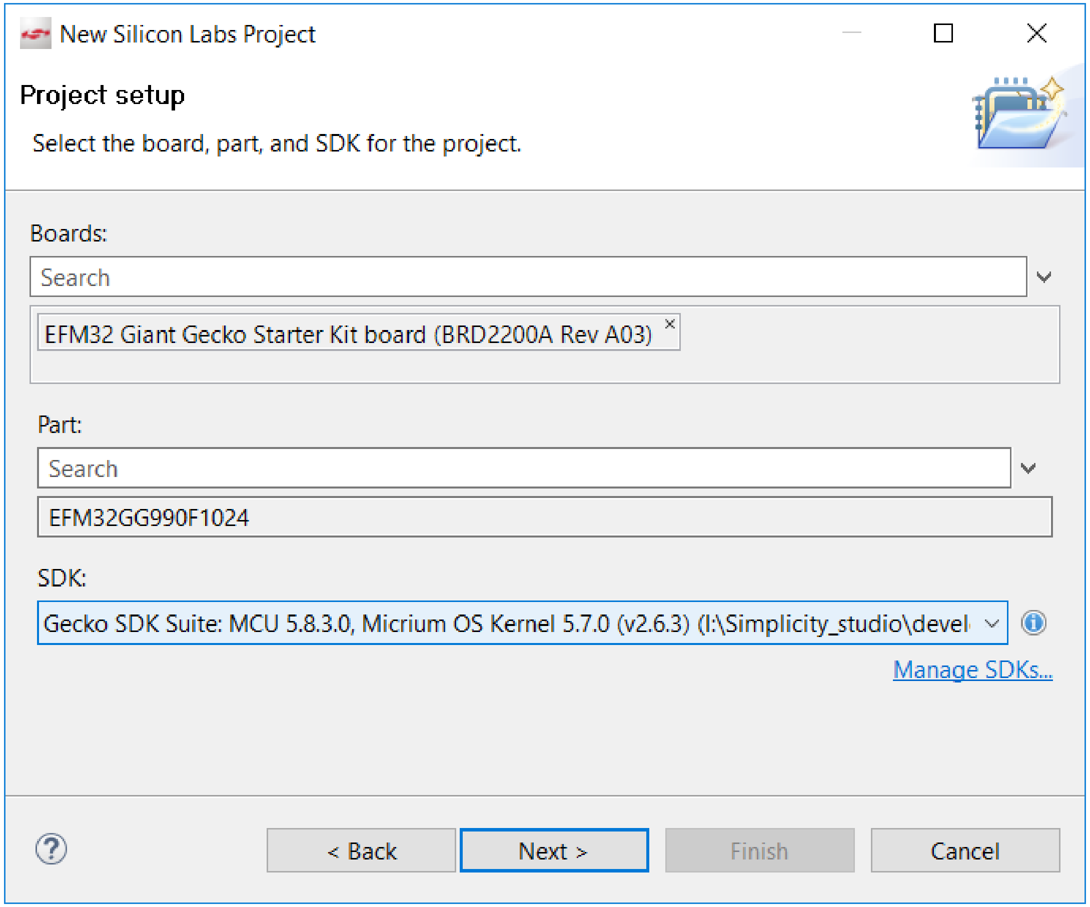
    runTime = DWT -> CYCCNT;

    here comes the code whose runtime is to be measured

    runTime = DWT -> CYCCNT – runtime – COMP_CONST;
  - COMP_CONST is used to get zero runtime when no *code* is applied -> reading of registers, calculations are not part of the *code* to be measured itself

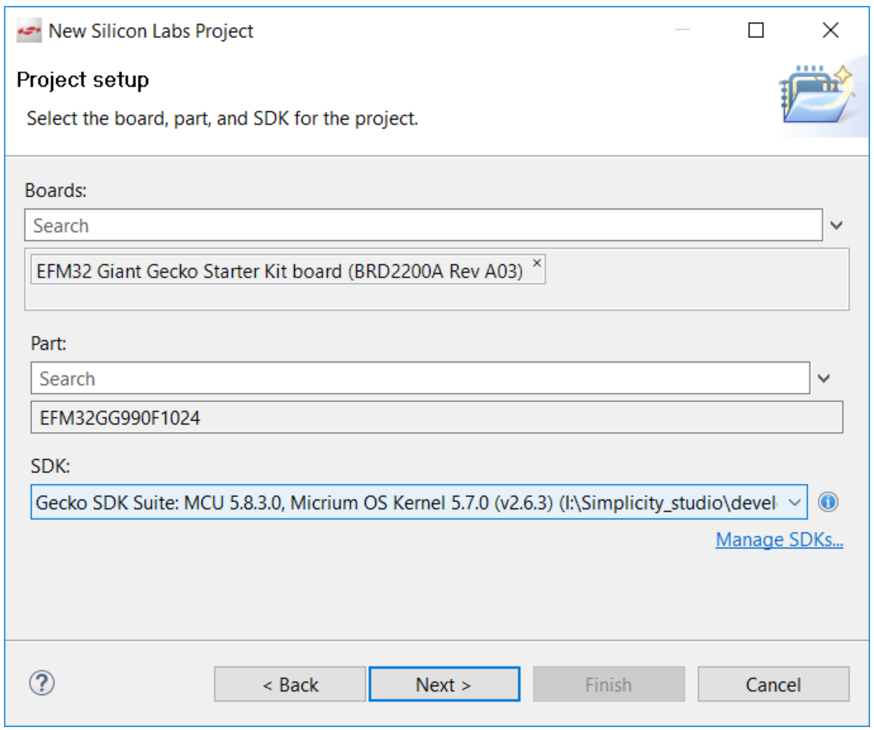
Strating with a new project

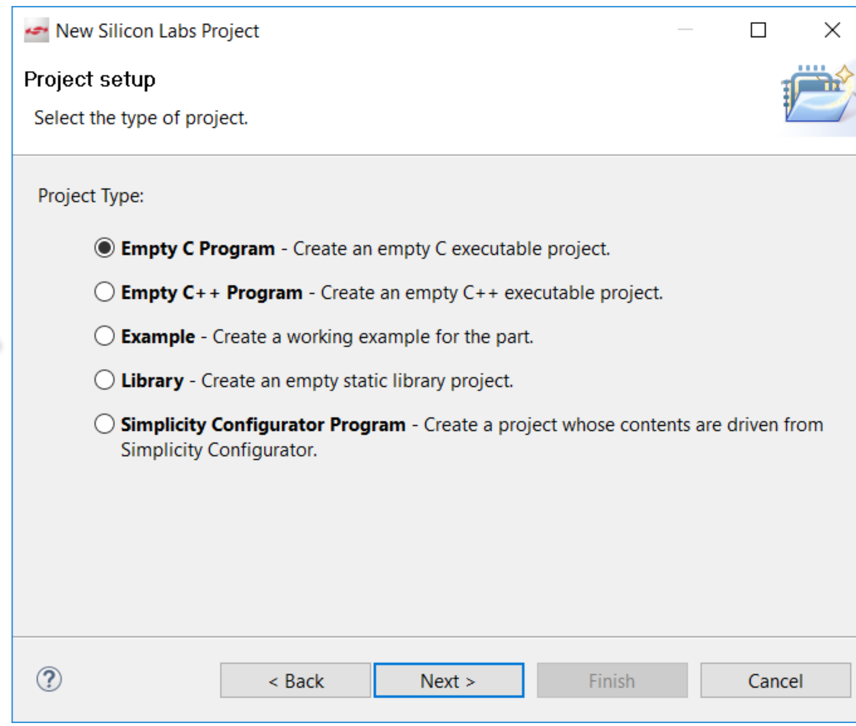
- File->New->Project->Silicon Labs MCU Project:



Strating with a new project

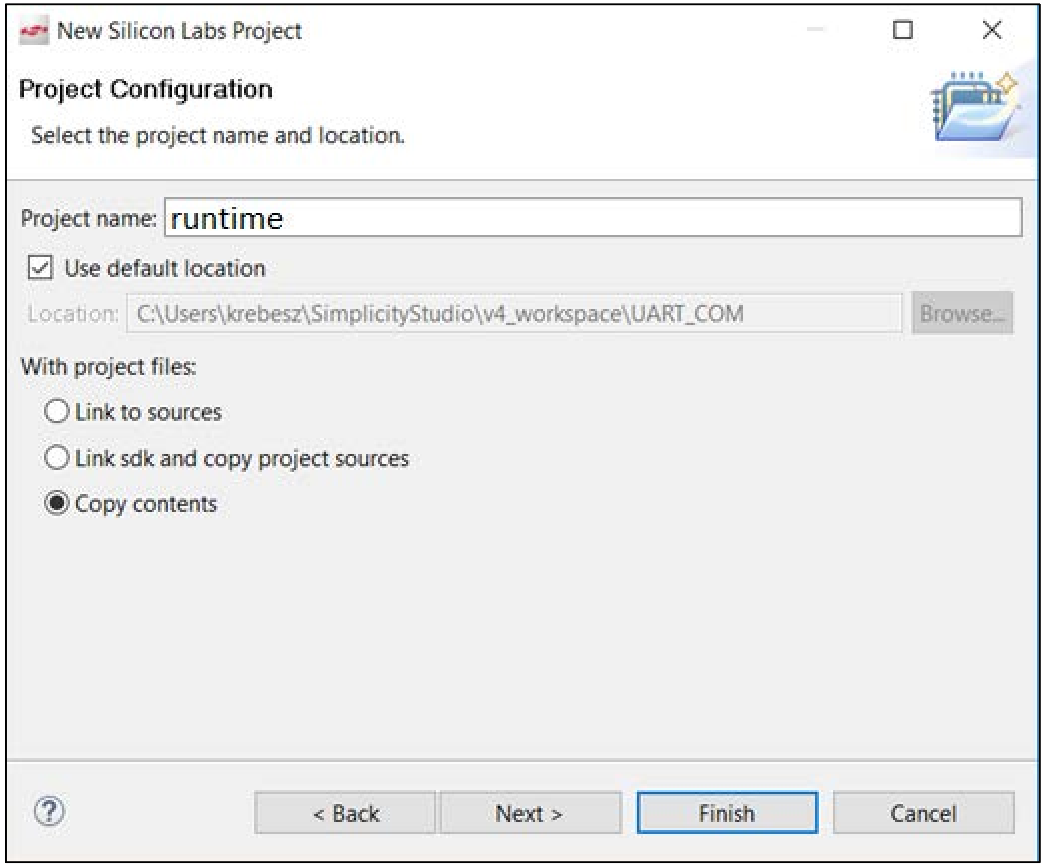
- File->New->Project->Silicon Labs MCU Project:





Strating with a new project

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



Runtime measurement

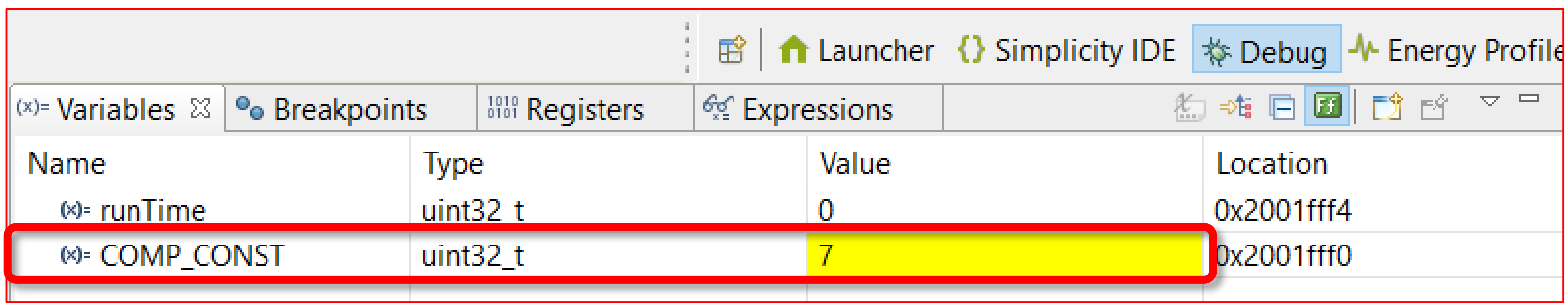
- The value of COMP_CONST has to be measured
  - Code: (CHIP_Init() has been removed)

```c
int main(void)
{

  uint32_t runTime;
  uint32_t COMP_CONST;


  runTime = DWT->CYCCNT;
  COMP_CONST = DWT->CYCCNT - runTime; //calculation of COMP_CONST

  runTime = DWT->CYCCNT;
  runTime = DWT->CYCCNT - runTime - COMP_CONST; //checking of COMP_CONST
```



  - Value of COMP_CONST is 7

Runtime measurement

- Determination of runtime of 3 type of operations for 3 data types
- Use optimization level –O0 (= no optimization)
- Code that can be used:

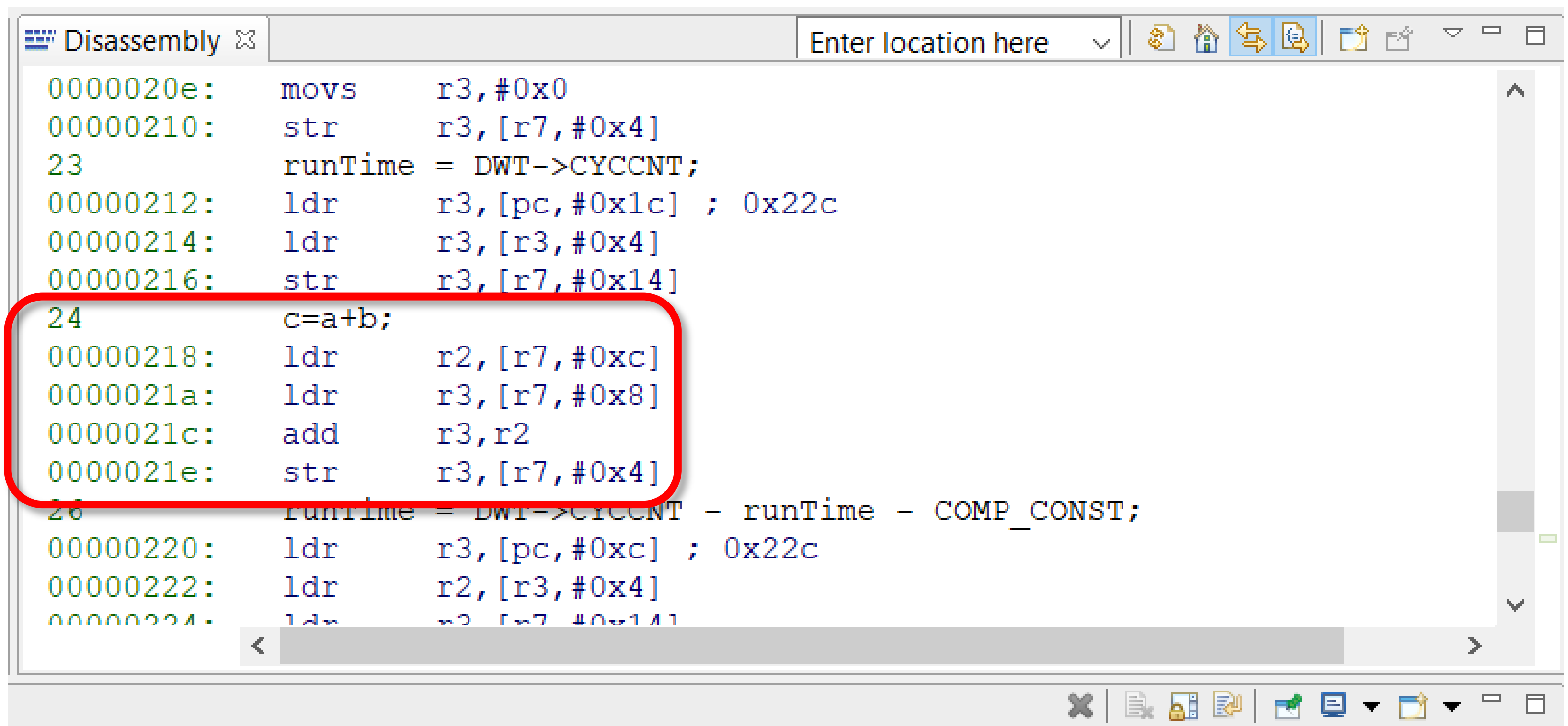
```
int32_t a=300, b=20, c=0;
runTime = DWT->CYCCNT;
c=a+b;
runTime = DWT->CYCCNT - runTime - COMP_CONST;
```

-O0 optimization	Int32_t	float	double
Summation: +	6	91	134
Multiplication: *	7	64	120
Division: /	11	91	164

- Remark: runtime may depend on where the variables are declared: before the main function (longer runT) or inside the main function
  - Variables are stored in different parts of the memory, and addressing method may be different (relative or direct)

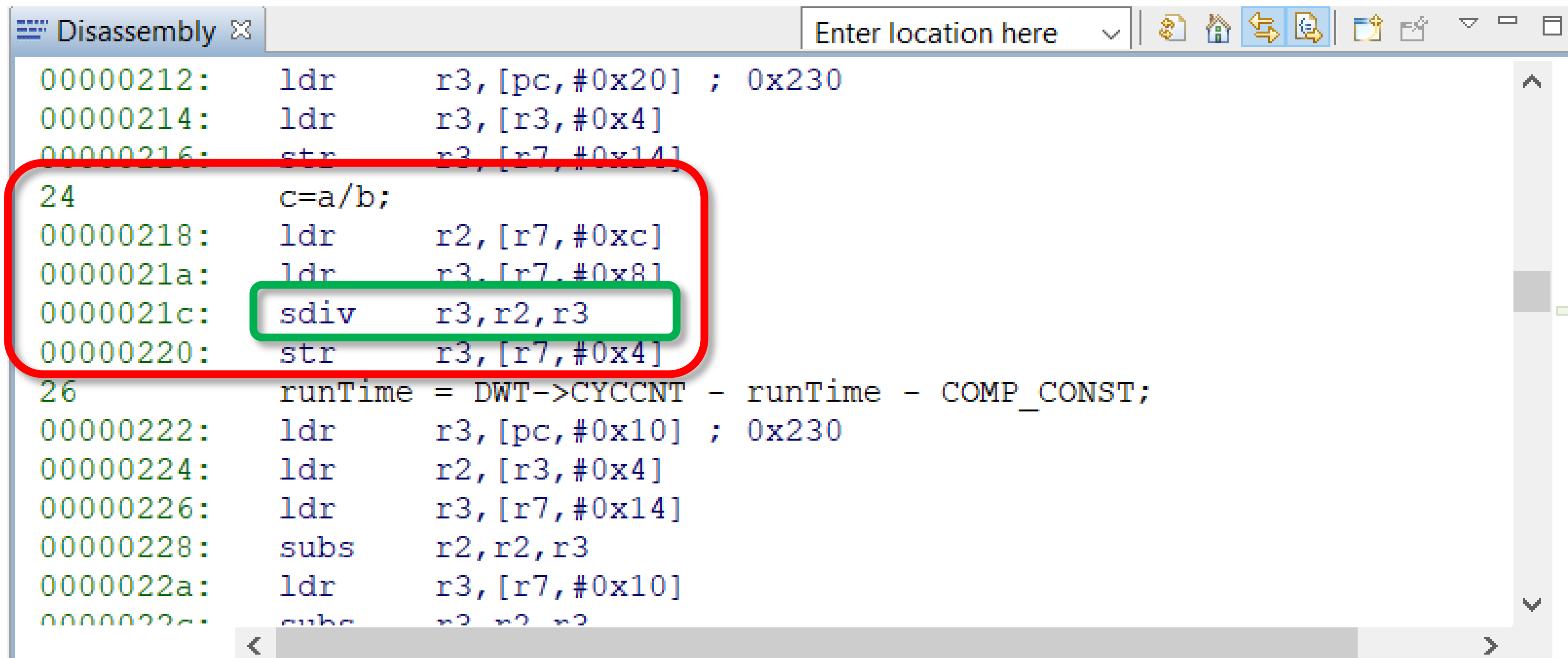
Runtime measurement

- Checking the disassembled code can also indicate the runtime
  - Note: not every instruction can be executed in one CLK cycle -> this method is just a rough guess



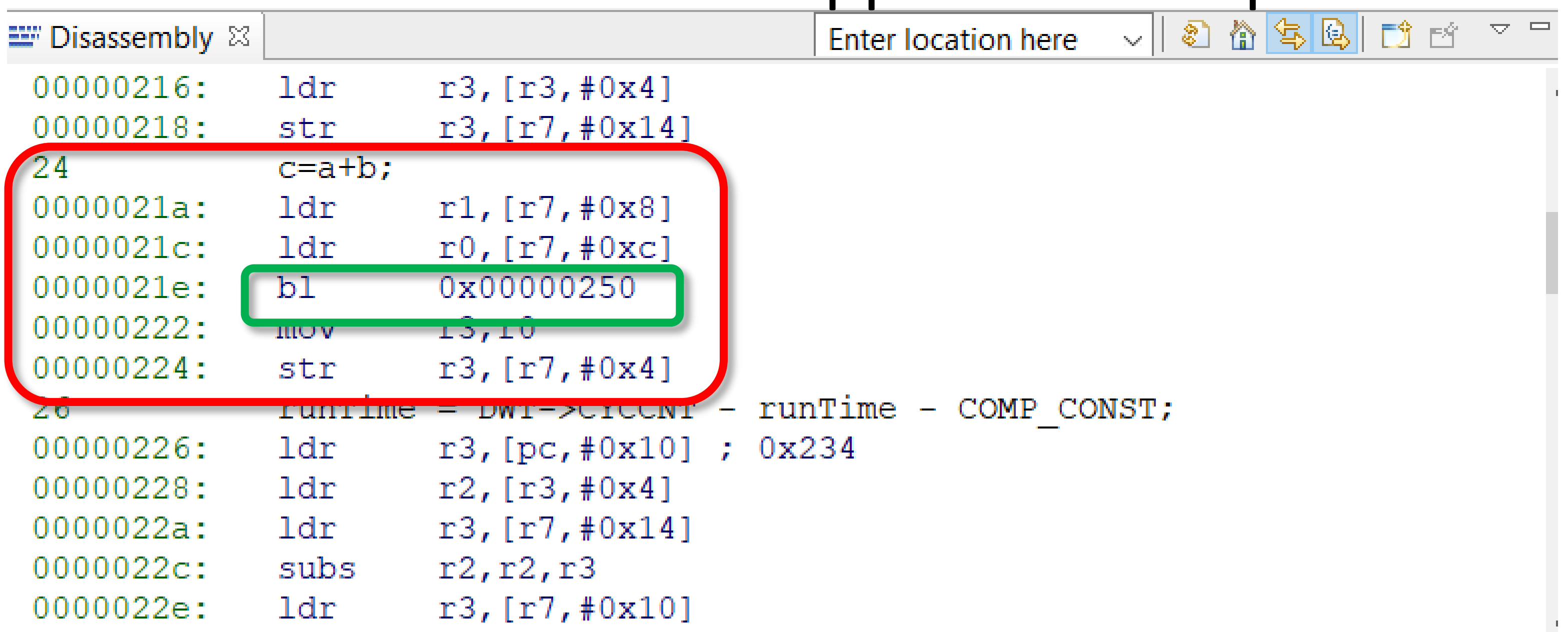
Runtime measurement

- Division is always a time consuming operation for an embedded system compared to summation
- HW support of division sometimes applied in a uC that significantly reduces the runtime



Runtime measurement

- Operations performed on floating point numbers takes more runtime
  - A function call is needed for floating point operations
  - Operation on mantissa and exponents takes more time for summation than HW-supported multiplication



Runtime measurement

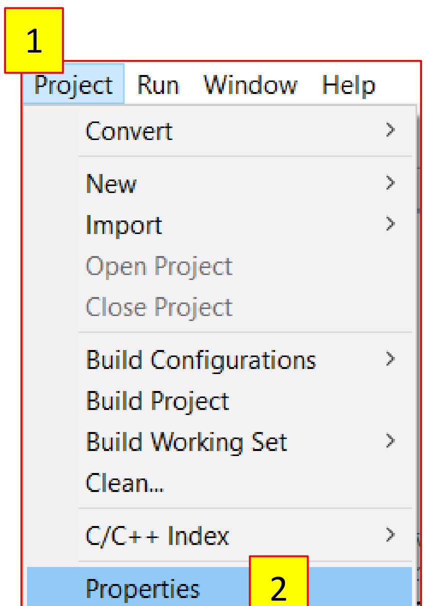
- Determination of runtime of type conversions
- Use optimization level –O0 (= no optimization)
- Recall: no explicit operation is done "just" type conversion
- Code that can be used:

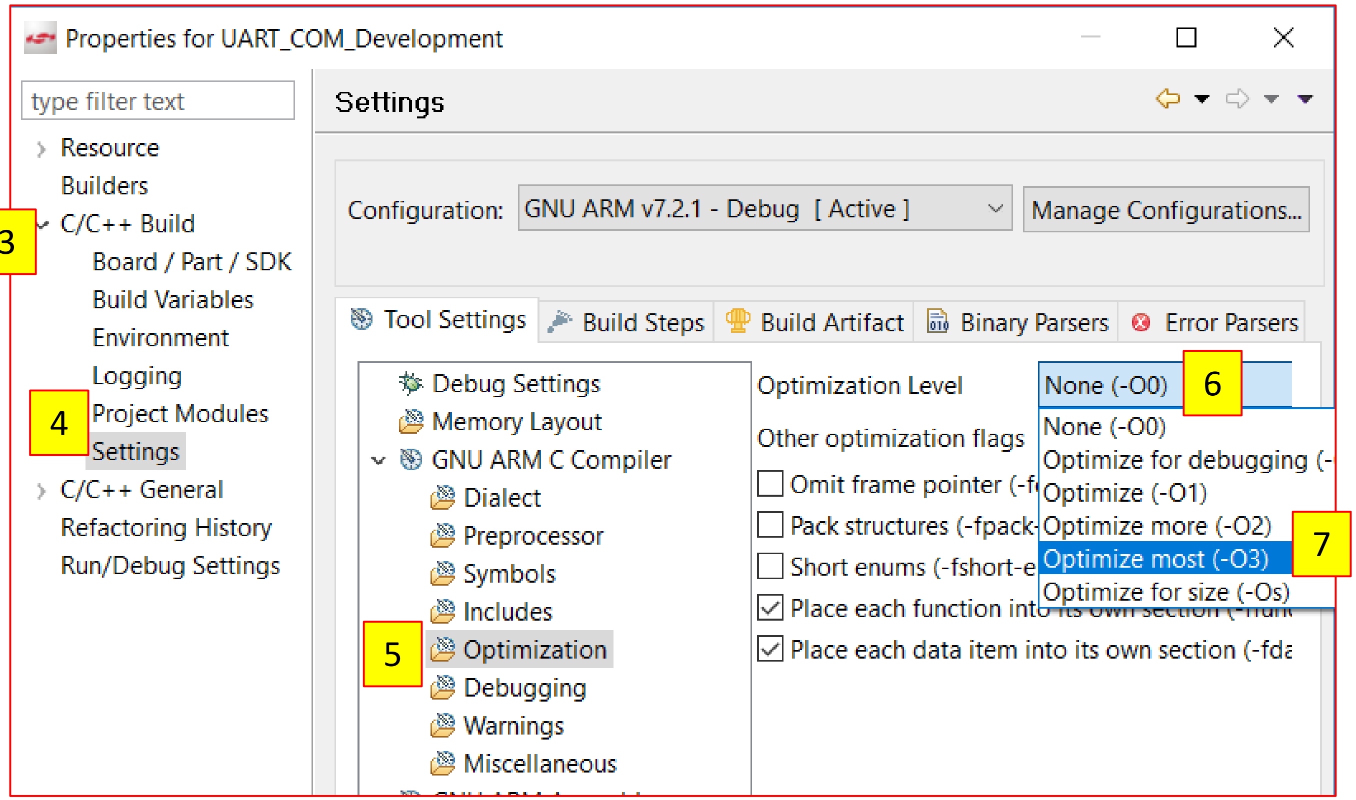
a_float = (float) a_int; OR **a_float = a_int;**

Source/target	int32_t	float	double
int32_t	-------------------------	50	70
float	31	-------------------------	26
double	36	36	-------------------------

Change of optimization level

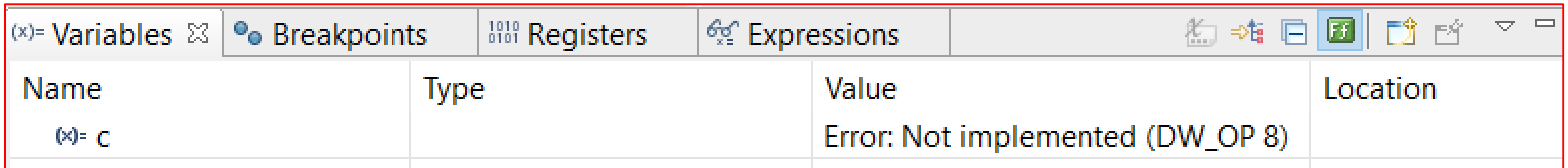
- To generate a more efficient (in terms of memory usage, runtime, etc.) code optimization should be applied: (O3)





Runtime measurement

- When optimization applied no result can be found since the optimizer “optimized out” the result and all those variables that are not used later



  - see warnings in the code
- To avid this use *volatile* to force the optimizer not to “optimize out” these variables (even runtime)
- However optimizer may use different order or removes operations that makes extremely difficult to follow and runtime measurement is not easy to be correctly done

Runtime measurement

- Determination of runtime of 3 type of operations for 3 data types
- Use optimization level –O3
- Code that can be used:

```
int32_t a=300, b=20, c=0;
runTime = DWT->CYCCNT;
c=a+b;
runTime = DWT->CYCCNT - runTime - COMP_CONST;
```

-O3 optimization	Int32_t	float	double
Summation: +	6	86	129
Multiplication: *	6	59	115
Division: /	10	86	159

Runtime measurement

- Determination of runtime of type conversions
- Use optimization level –O3 (= no optimization)
- Code that can be used:

a_float = (float) a_int;

Source/target	int32_t	float	double
Int32_t	--------------------------	48	67
float	29	--------------------------	23
double	33	33	--------------------------

Runtime measurement

- Sum operation using arrays: $s = \sum_{i=0}^{N-1} A[i] * B[i]$
- Use optimization level –O3
- Arrays should be *volatile int32_t*
- Code to be used to measure its runtime for different N values:

```
#define N 15

  volatile int32_t sum;
  volatile int32_t A[N];
  volatile int32_t B[N];

  int ii;

  runTime = DWT->CYCCNT;

  sum = 0;
  for (ii=0; ii<N; ii++){
      sum += A[ii]*B[ii];
  }

  runTime = DWT->CYCCNT - runTime - COMP_CONST;
```

Runtime measurement

- What are the runtimes for array sizes H=15…100?

N	15	16	17	18	19	20	50	100
CLK cycles (-O3)	121	129	137	257	271	285	657	1207
CLK cycles (-O0)	477	528	560	592	624	656	1513	2616

- Compare the CLK cycles for N=<15,16,17> and N=<18,19,20,50,100>
  - There is a jump in the runtime
  - Explanation: loop unroll operation due to optimization level –O3

Runtime measurement

- Loop unroll operation: (only when N is constant)
  - The optimizer extracts the FOR loop and perform multiply and accumulate operation N times when N<18
  - When N>=18 the optimizer performs the FOR loop
  - Check disassembled code:

N<18

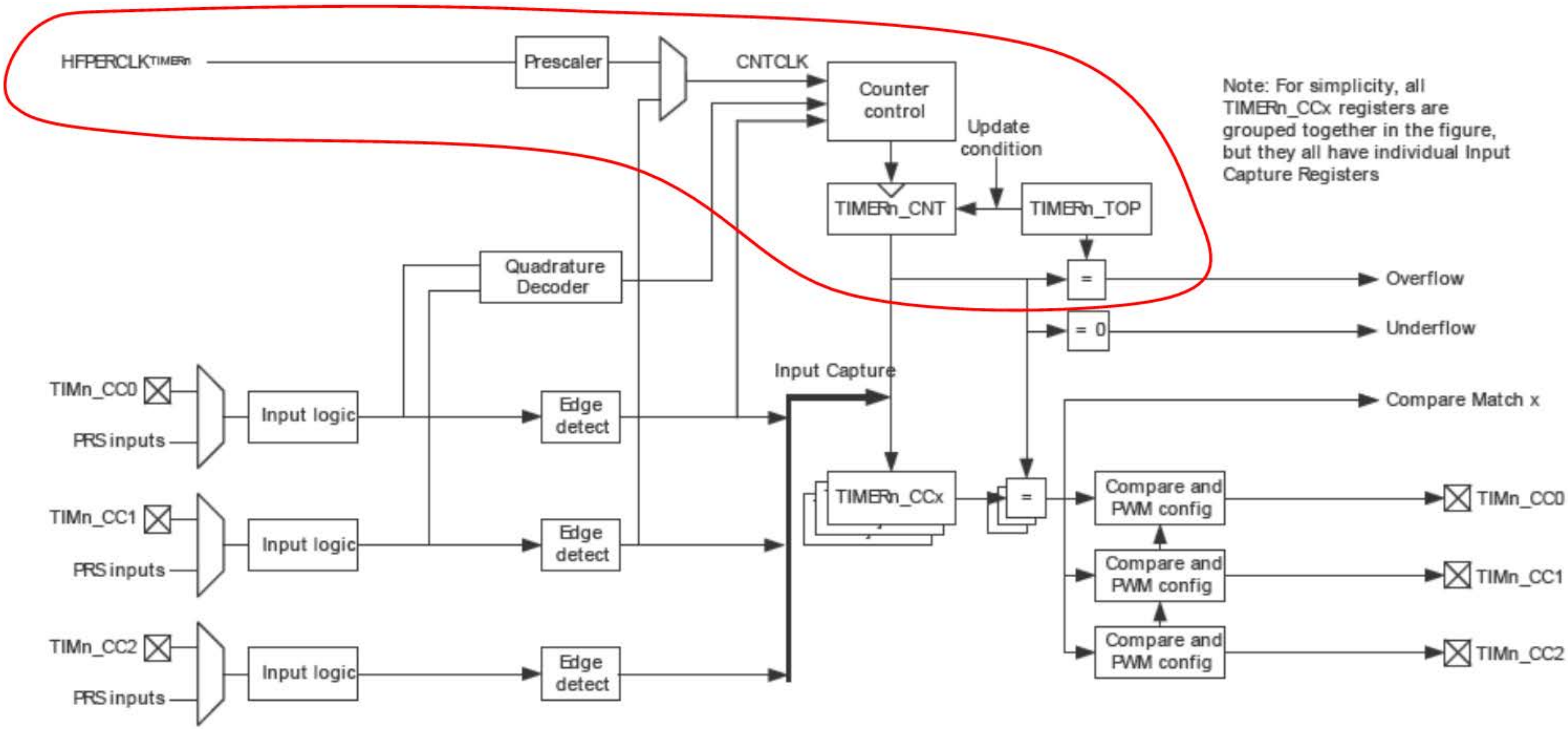
```
49                    sum += A[ii]*B[ii];
000001fa:    ldr       r2,[sp,#0x10]
000001fc:    ldr       r0,[sp,#0x4c]
000001fe:    ldr       r1,[sp,#0xc]
00000200:    mla       r2,r0,r2,r1
00000204:    str       r2,[sp,#0xc]
00000206:    ldr       r2,[sp,#0x14]
00000208:    ldr       r0,[sp,#0x50]
0000020a:    ldr       r1,[sp,#0xc]
0000020c:    mla       r2,r0,r2,r1
00000210:    str       r2,[sp,#0xc]
00000212:    ldr       r2,[sp,#0x18]
00000214:    ldr       r0,[sp,#0x54]
00000216:    ldr       r1,[sp,#0xc]
```

N>=18

```
49                    sum += A[ii]*B[ii];
000001fc:    add       r3,sp,#0xa0
000001fe:    add.w     r2,r3,r1,lsl #2
00000202:    ldr       r3,[r2,#-0x90]
00000206:    ldr       r0,[r2,#-0x48]
0000020a:    ldr       r2,[sp,#0xc]
48                  for (ii=0; ii<N; ii++){
0000020c:    adds      r1,#0x1
49                    sum += A[ii]*B[ii];
0000020e:    mla       r3,r0,r3,r2
48                  for (ii=0; ii<N; ii++){
00000212:    cmp       r1,#0x12
49                    sum += A[ii]*B[ii];
00000214:    str       r3,[sp,#0xc]
```

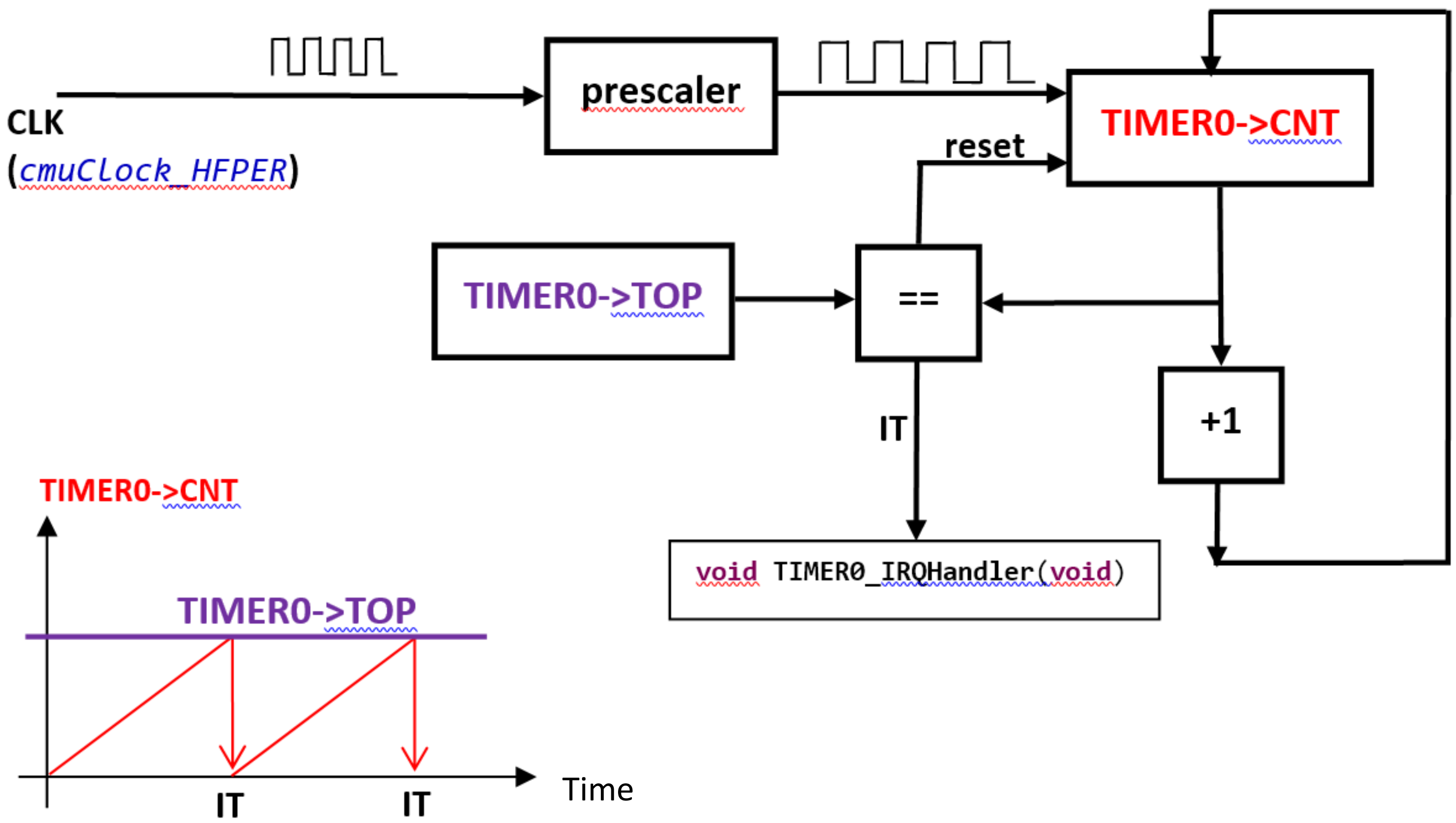
Time measurement, timers

- HW-based timers in Giant Gecko



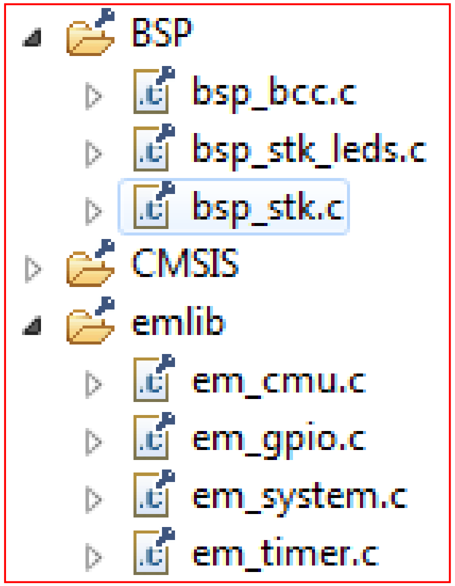
Time measurement, timers

- HW-based timers in Giant Gecko: simplified block diagram



Time measurement, timers

- To handle the timer the following files needed to be added to the project:
  - em_timer.c   em_cmu.c
  - Include the corresponding .h files into the code
- To handle the LEDs the following files needed to be added to the project:
  - bsp_bcc.c bsp_stk_leds.c bsp_stk.c em_gpio.c
  - Include bsp.h file into the code
- Paths: SimplicityStudio\developer\sdks\gecko_sdk_suite\v1.1\platform\emlib\src\
  SimplicityStudio\developer\sdks\gecko_sdk_suite\v1.1\hardware\kit\common\bsp\




```
#include "em_device.h"
#include "em_chip.h"
#include "em_cmu.h"
#include "em_timer.h"
#include "bsp.h"
```

Time measurement, timers

- The timer shall be configured using the library functions as follows:
  - Setting the prescaler of the peripheral clock
  - Enabling the clock of the timer
  - Generation of the parameter structure for initialization
    - Prescaler is set to the appropriate value
  - Reset the timer
  - Setting the value of TOP
  - Clear the interrupt
  - Enable the interrupt
    - Enable the peripheral interrupt
    - Enable the core-based interrupt for the Timer (NVIC)

Time measurement, timers

Possible implementation:

```
// Setting the prescaler of the peripheral clock
CMU_ClockDivSet(cmuClock_HFPER, cmuClkDiv_1);

// ******************************
// *    TIMER inicialization    *
// ******************************

// Enable the clock of the timer
CMU_ClockEnable(cmuClock_TIMER0, true);

// Generation of the parameter structure for initializatio
TIMER_Init_TypeDef TIMER0_init = TIMER_INIT_DEFAULT;
// Setting the prescaler
TIMER0_init.prescale = timerPrescale1; // timerPrescale1...timerPrescale1024
// Initialization using the parameter sturcture
//void TIMER_Init(TIMER_TypeDef *timer, const TIMER_Init_TypeDef *init);
TIMER_Init(TIMER0, &TIMER0_init);

// Reset the counter
TIMER_CounterSet(TIMER0, 0); //
```

Time measurement, timers

```
// Setting the TOP value
//__STATIC_INLINE void TIMER_TopSet(TIMER_TypeDef *timer, uint32_t val)
TIMER_TopSet(TIMER0, WRITE_HERE_THE_TOP_VALUE); // 14MHz/presc/TOP

// Clear the interrupt
//__STATIC_INLINE void TIMER_IntClear(TIMER_TypeDef *timer, uint32_t flags);
TIMER_IntClear(TIMER0, TIMER_IF_OF);

// Enable interrupt at peripheral
//TIMER_IntEnable(TIMER_TypeDef *timer, uint32_t flags);
TIMER_IntEnable(TIMER0, TIMER_IF_OF);

// Enable interrupt at NVIC
NVIC_EnableIRQ(TIMER0_IRQn);


// *******************************
// *     LED initialization      *
// *******************************
BSP_LedsInit();
```

Time measurement, timers

- Implementation of IT function to toggle LEDs:

```
//IT function that implements LED toggling – comes before main{}
void TIMER0_IRQHandler(void){
        BSP_LedToggle(0);
        TIMER_IntClear(TIMER0, TIMER_IF_OF); //TIMER flag clear
}
```

- Timer_Init_Default:

```
//Default values for timer init
#define TIMER_INIT_DEFAULT
{
    1,                   /* Enable timer when init complete. */
    0,                  /* Stop counter during debug halt. */
    timerPrescale1,         /* No prescaling. */
    timerClkSelHFPerClk,   /* Select HFPER clock. */
    0,                  /* Not 2x count mode. */
    0,                  /* No ATI. */
    timerInputActionNone, /* No action on falling input edge. */
    timerInputActionNone, /* No action on rising input edge. */
    timerModeUp,           /* Up-counting. */
    0,                  /* Do not clear DMA requests when DMA channel is active. */
    0,                  /* Select X2 quadrature decode mode (if used). */
    0,                  /* Disable one shot. */
    0                   /* Not started/stopped/reloaded by other timers. */
  }
```

Time measurement, timers

- Calculation of TOP value:
  - Toggle the LEDs in every T=1s
  - CLK frequency = 14MHz (default value for this uC)
  - Tick time = T_tick = 1/14MHz
  - Timer value where the timer should be reset = TOP value = N
    - N = T / T_tick = 1s / (1/14MHz) = 14*10^6 -> very large number
    - Can we store such a large number in the Timer? What is the data width?
    - When a timer data width is not enough the prescaler must be used
- Example:
  - Timer is 16-bit wide -> 2^16 = 65535 is the largest number to store
  - Prescaler must be applied, e.g., Prescale_value256
    - 14 000 000 / 256 = 54687.5
      -> N=54688 will correspond to 1s (not precise: error ~9ppm)

Working code

```c
#include "em_device.h"
#include "em_chip.h"
#include "em_timer.h"
#include "em_cmu.h"
#include "em_gpio.h"
#include "bsp.h"

//IT function that implements LED toggling - comes before main{}
void TIMER0_IRQHandler(void){
    BSP_LedToggle(0);
    TIMER_IntClear(TIMER0, TIMER_IF_OF); //TIMER flag clear
}


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

  // Setting the prescaler of the peripheral clock
    CMU_ClockDivSet(cmuClock_HFPER, cmuClkDiv_1);

    // ******************************
    // *    TIMER inicialization    *
    // ******************************

    // Enable the clock of the timer
    CMU_ClockEnable(cmuClock_TIMER0, true);

    // Generation of the parameter structure for initializatio
    TIMER_Init_TypeDef TIMER0_init = TIMER_INIT_DEFAULT;
    // Setting the prescaler
    TIMER0_init.prescale = timerPrescale256; // timerPrescale1...timerPrescale1024
```

Working code

```
    // Initialization using the parameter sturcture
    //void TIMER_Init(TIMER_TypeDef *timer, const TIMER_Init_TypeDef *init);
    TIMER_Init(TIMER0, &TIMER0_init);

    // Reset the counter
    TIMER_CounterSet(TIMER0, 0); //

    // Setting the TOP value
    //__STATIC_INLINE void TIMER_TopSet(TIMER_TypeDef *timer, uint32_t val)
    TIMER_TopSet(TIMER0, 54688); // 14MHz/presc/TOP

    // Clear the interrupt
    //__STATIC_INLINE void TIMER_IntClear(TIMER_TypeDef *timer, uint32_t flags);
    TIMER_IntClear(TIMER0, TIMER_IF_OF);

    // Enable interrupt at peripheral
    //TIMER_IntEnable(TIMER_TypeDef *timer, uint32_t flags);
    TIMER_IntEnable(TIMER0, TIMER_IF_OF);

    // Enable interrupt at NVIC
    NVIC_EnableIRQ(TIMER0_IRQn);


    // ********************************
    // *      LED initialization      *
    // ********************************
    BSP_LedsInit();


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