

Cvičení z předmětu

KAE/MPP

verze 2015-16

Katedra aplikované elektroniky a telekomunikací

přednášky:	prof. Ing. Jiří Pinker, CSc.	pinker@kae.zcu.cz	EK517
cvičení:	Ing. Petr Weissar, Ph.D.	weissar@kae.zcu.cz	EK515
	Ing. Kamil Kosturik, Ph.D.	kosturik@kae.zcu.cz	EK515
	Ing. Petr Krist, Ph.D.	krist@kae.zcu.cz	EK507

Plán cvičení

1. **Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)**
2. I/O porty – tlačítka a LEDky
3. Časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Sem. práce - podrobnosti

- Semestrální práce se vypracovává ve skupinách (=dvojice)
- Rozsah odpovídající času/počtu cvičení
- Typicky využívá HW vývojové desky
- Po domluvě možný jiný HW/mikroprocesor
 - AVR, x51, ARM, ...
 - možno využít stávající projekt, bakalářku, ...
- Programováno min. ze 2/3 v C/C++

Bezpečnost v laboratoři

- Protokol s datem zkoušky "padesátky"
- Pracuje se s bezpečným napětím, deska napájena z USB
- Změny HW konfigurace nechat schválit cvičícím
- Změny zásadně provádět při odpojení napájení !!!

Podmínky zápočtu

- Semestrální práce – obhájená, funkční, nutná aktivní znalost !!!
- Praktická účast na cvičeních, znalost probírané problematiky

Doporučená literatura

- Libovolná učebnice programovacího jazyka C
- Dokumentace v elektronické podobě – lokálně + CW
 - 8-bitové jednočipové mikropočítače řady S08
 - Vývojová deska (TWR) s mikropočítačem MC9S08LL64
 - Periferních obvodů, příp. vlastního HW
- Knihy
 - Mikroprocesory a mikropočítače : obecné principy konstrukce současných mikroprocesorů a mikropočítačů – prof. Pinker (BEN 2004)
 - C pro mikrokontroléry – Burkhard Mann (BEN)
- Courseware
 - V "cvičení" a "studijní materiály" k dispozici odkazy a dokumenty PDF



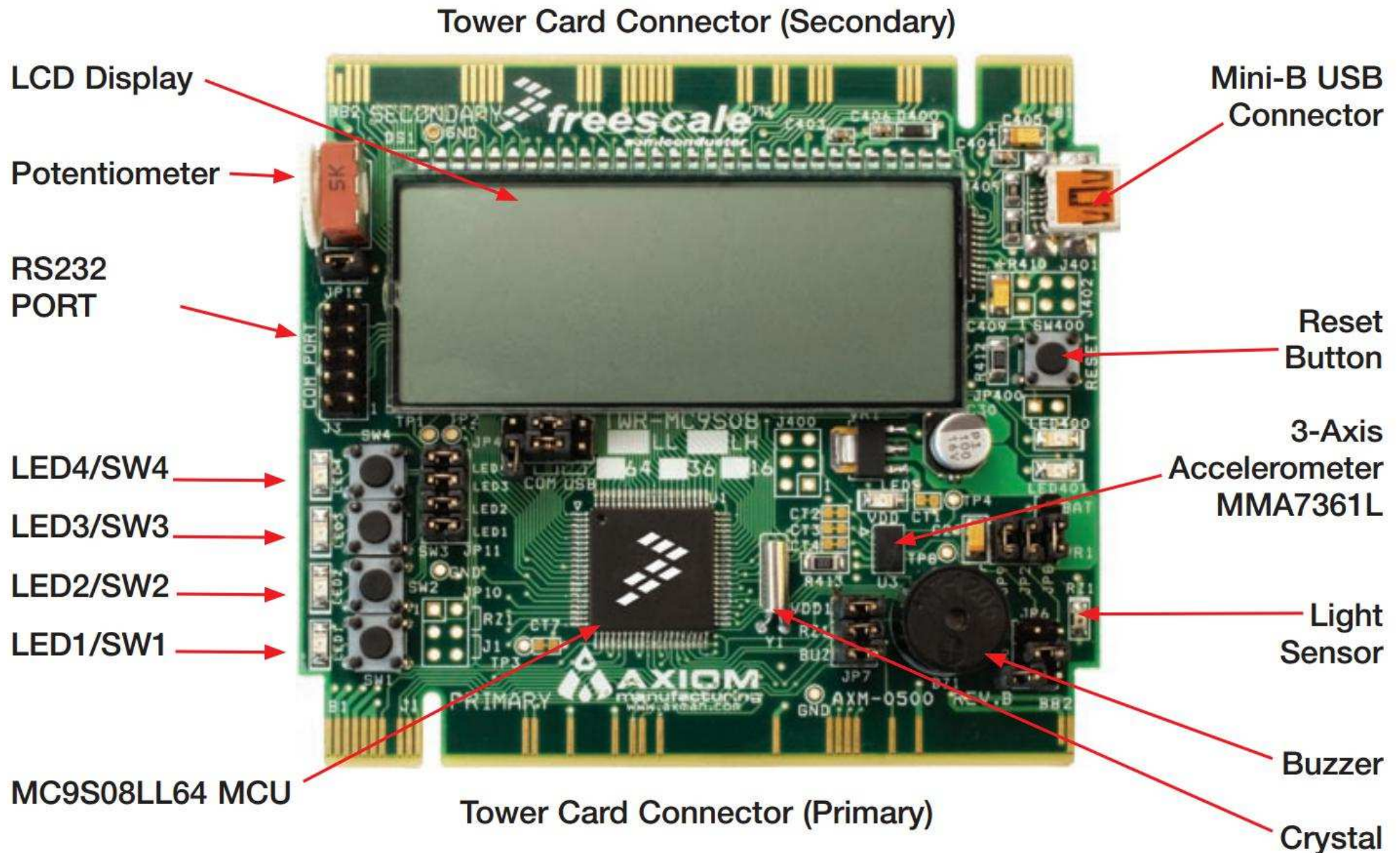
Další provozní informace

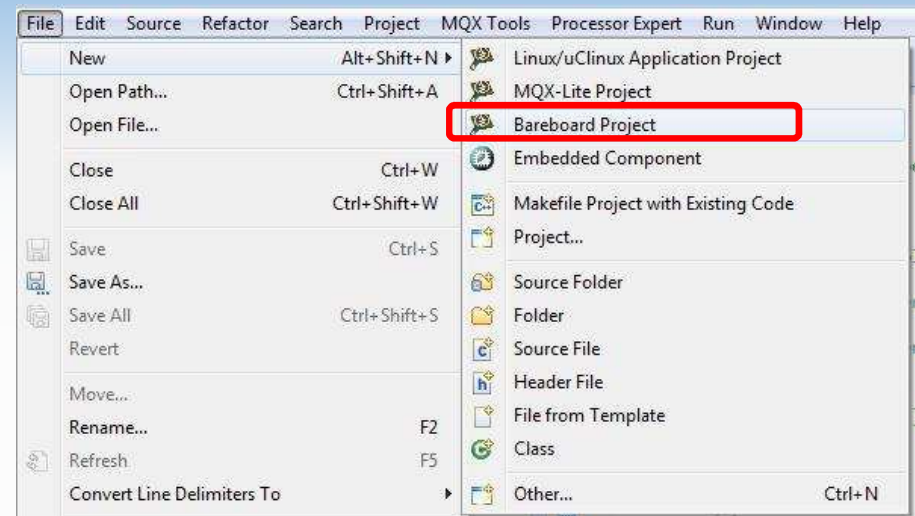
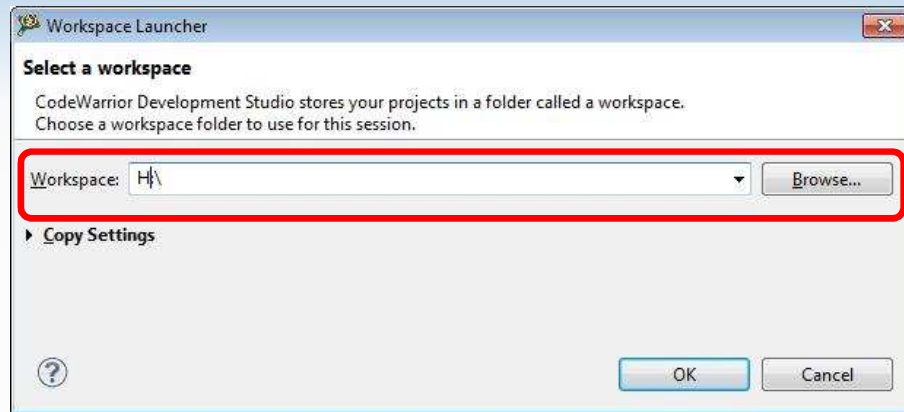
- Konzultace nejlépe v rámci cvičení
- Možno využívat laboratoř mimo své cvičení, respektovat rozvrh
- Ve volných hodinách mají v laboratoři přednost diplomanti

Pracovní soubory

- Interní síť se souborovým serverem mimo AFS prostor s mapovanými disky
 - disky X:, Z: - servisní a SW/systémový – ReadOnly
 - **Z:\podklady\MPP – soubory, dokumentace, ...**
 - disk T: - dočasný datový, společný, může se mazat o půlnoci
 - disk H: - osobní data každé skupiny
- Přilogování – **Win7** – uživatel **lab** (lokální uživatel bez hesla)
 - automaticky se spustí "logon" dávka s prohlížečem
 - přihlášení pomocí Orion jména/hesla člena skupiny
 - vyberte si předmět, kterému se hodláte věnovat
 - po ukončení prohlížeče se příslušně namapuje disk H:
- Po skončení práce se "odlogujte" nebo restartujte PC

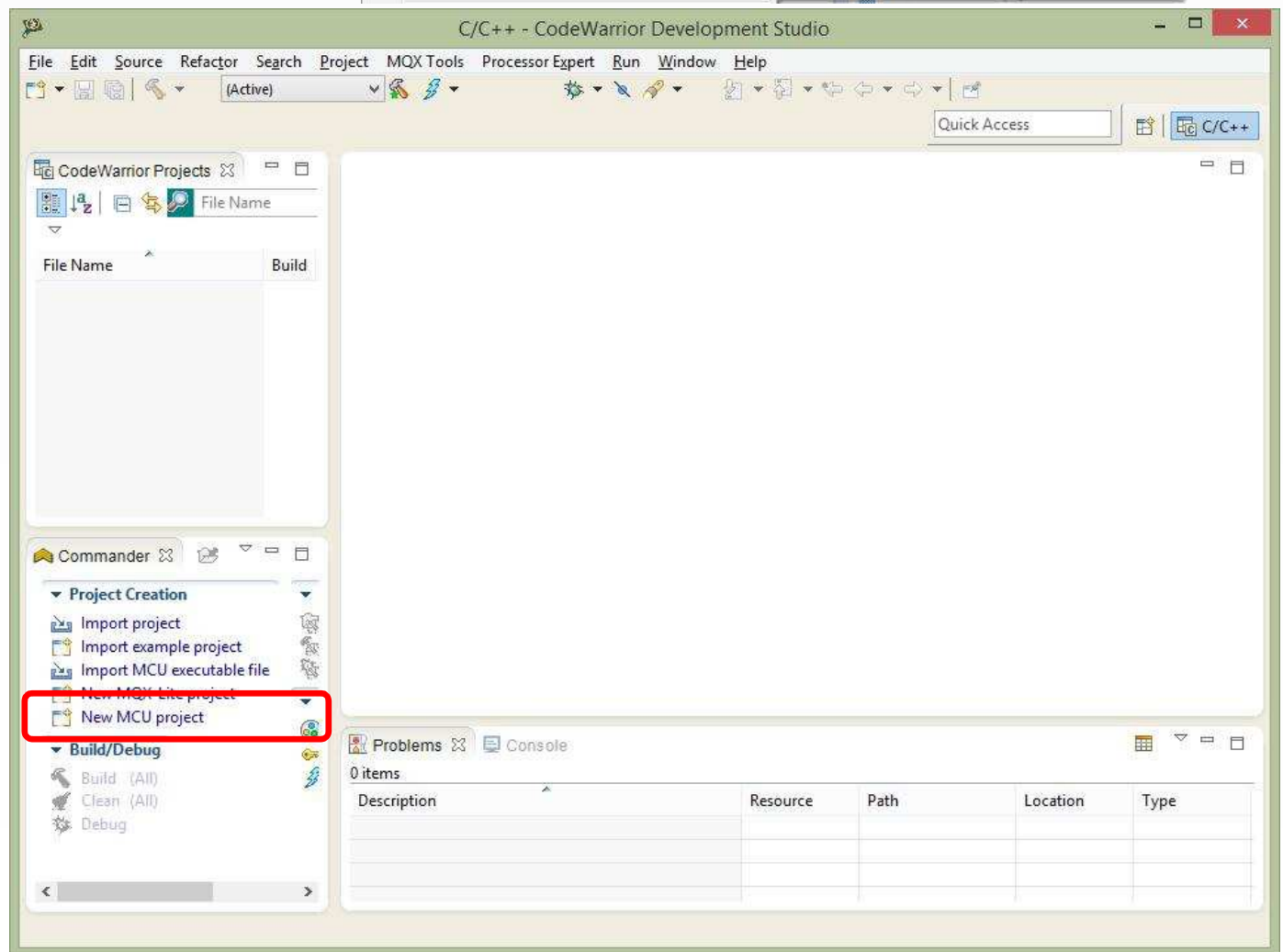
Vývojový kit Freescale S08LL64





Spuštění prostředí

- Výběr pracovního prostoru (= Workspace)
 - H: - Home adresář
- Přidat projekt
 - New MCU project
 - V menu New Bareboard Project



New Bareboard Project

Create an MCU Bareboard Project

Choose the location for the new project

Project name:

☒ Use default location

Location:

New Bareboard Project

Devices

Select the derivative or board you would like to use

Device or board to be used:

☒ S08

- HCS08LL Family
 - MC9S08LL8
 - MC9S08LL16
 - MC9S08LL32
 - ☒ MC9S08LL64

Project Type / Output:

☒ Application

☐ Library

New Bareboard Project

Connections

Choose the connection to use for this project

Connection to be used:

☒ P&E Full Chip Simulation

☐ P&E USB MultiLink Universal [FX] / USB MultiLink

☒ P&E Cyclone PRO

☒ Open Source BDM

☐ USBDM

Connect to USBDM Open Source BDM.

New Bareboard Project

Languages

Language:

☒ C

☐ C++

☐ Relocatable Assembly

☐ Absolute Assembly

C language support will be included in the project

New Bareboard Project

Rapid Application Development

☐ None

☒ Processor Expert

Start with perspective designed for

☐ Hardware configuration (pin muxing and device initialization)

☒ Use current perspective

☐ Initialize all peripherals

Processor Expert can generate for you all the device initialization code. It includes many low-level drivers.

New Bareboard Project

C/C++ Options

Which memory model shall be used?

☐ Tiny

☒ Small

☐ Banked

Select the floating point format supported. Select "None" for best code density.

☒ None

☐ Float is IEEE32, Double is IEEE32 optimized

☐ Float is IEEE32, Double is IEEE64 optimized

Which level of startup code shall be used? Use "Minimal Startup Code" for best code density.

☐ Minimal Startup Code

☒ ANSI Startup Code

By default all variables are outside the zero page (extended memory access). Variables in the zero page can be used with pragmas or the near keyword.

Don't use floating point support.

This will perform an ANSI compliant startup code: it initializes global variables/objects

Vygenerovaná kostra aplikace

The screenshot displays the CodeWarrior Development Studio interface. The main window shows the source file `main.c` with the following content:

```
24 ** @addtogroup main_module main module documentation
25 ** @{}
26 */
27 /* MODULE main */
28
29 /* Including needed modules to compile this module/procedure */
30 #include "Cpu.h"
31 #include "Events.h"
32 /* Include shared modules, which are used for whole project */
33 #include "PE_Types.h"
34 #include "PE_Error.h"
35 #include "PE_Const.h"
36 #include "IO_Map.h"
37
38 /* User includes (#include below this line is not maintained by Processor Expert) */
39
40
41 void main(void)
42 {
43     /* Write your local variable definition here */
44
45     /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
46     PE_low_level_init();
47     /*** End of Processor Expert internal initialization. ***/
48
49     /* Write your code here */
50     /* For example: for(;;) { } */
51
52     /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
53     /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
54     #ifdef PEX_RTOS_START
55         PEX_RTOS_START(); /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
56     #endif
57     /*** End of RTOS startup code. ***/
58     /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
59     for(;;){}
60     /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
61 } /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
62
63 /* END main */
64 /*!
65 ** @}
66 */
67 /*
68 ** =====
69 **
70 ** This file was created by Processor Expert 10.3 [05.09]
```

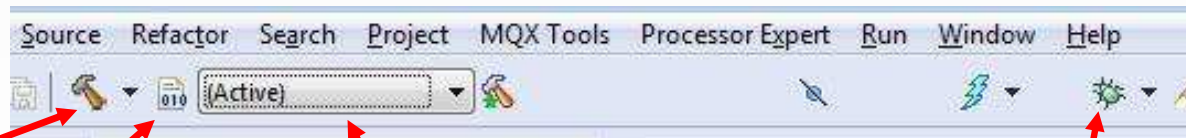
The left sidebar shows the project structure for `cv1_pr : FLASH`, including folders like `Documentation`, `FLASH`, `Generated_Code`, `Lib`, `ProcessorExpert.pe`, `Project_Headers`, `Project_Settings`, and `Sources`. The `Sources` folder contains `Events.c`, `Events.h`, and `main.c`. The Commander window shows options for Project Creation, Build/Debug, Settings, and Miscellaneous.

On the right side, a code snippet is shown:

```
...
void main(void)
{
    ...
    char c = 0;
    ...
    while(1)
    {
        C++;
    }
    ...
}
```

Spuštění programu v simulátoru

- Před spuštěním debuggeru se IDE zeptá, zda uložit a přeložit (pokud došlo ke změnám)
 - Přesněji "provést Build" = kompilace jednotlivých modulů + linkování dohromady i s knihovnami
- V případě chyb při překladu nutno opravit



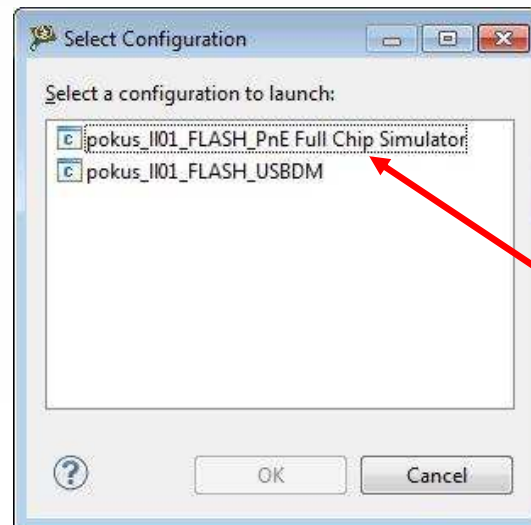
Build

Build All

Výběr konfigurace "Buildu"

Debug (výběr známé konfigurace)

- Full Chip Simulator
- FLASH

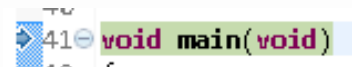


Použije poslední konfiguraci
Pokud ještě nebyla žádná zvolena, "zeptá se"

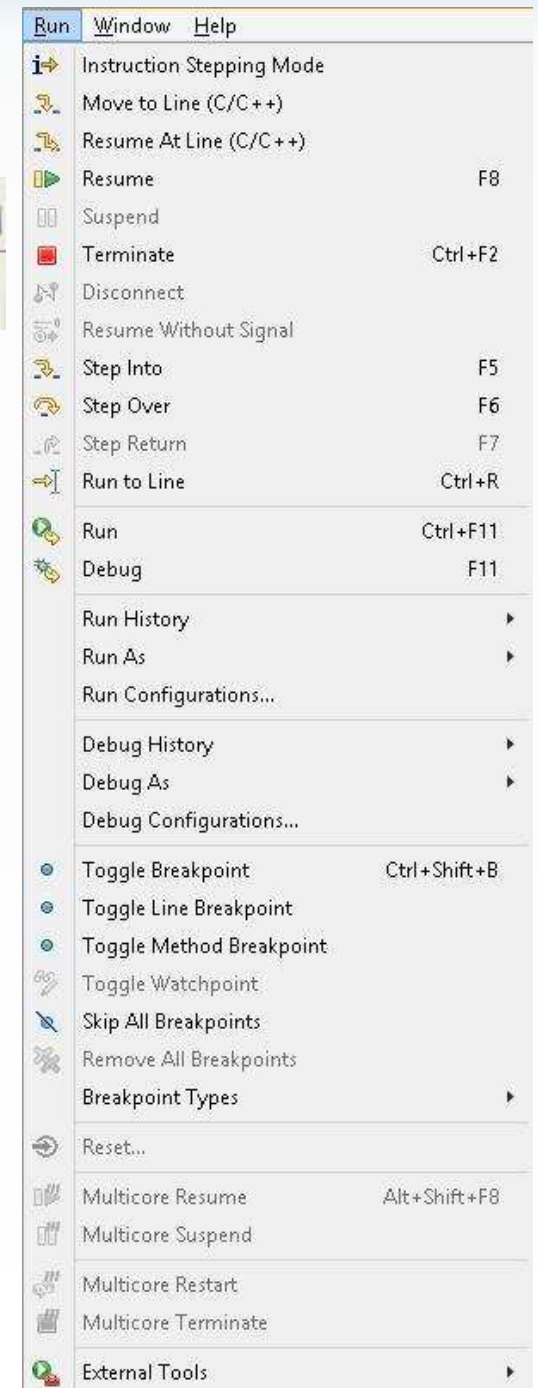
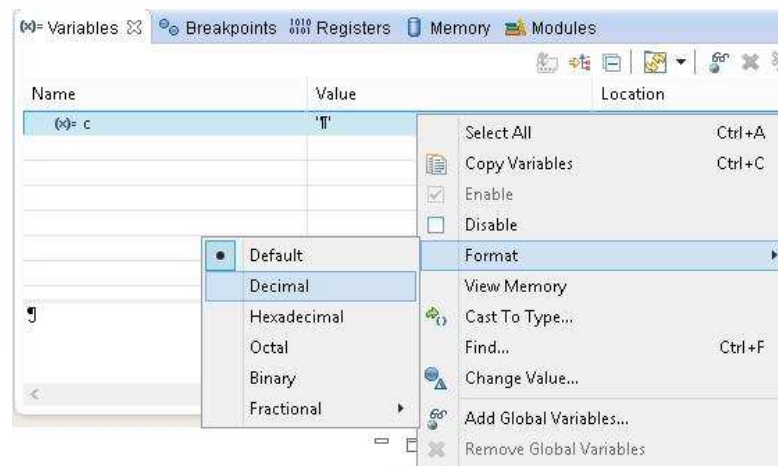
Základy debuggeru - simulátor



- Debugger lze ovládat pomocí ikon, menu nebo klávesových zkratk
- Ladit lze jak pomocí emulátoru, tak HW
 - Je nutno mít zkonfigurované připojení (OSBDM)
- Řádek „k provedení“ uvozen „šipkou“



- Proměnné mají defaultní formát s možností změny



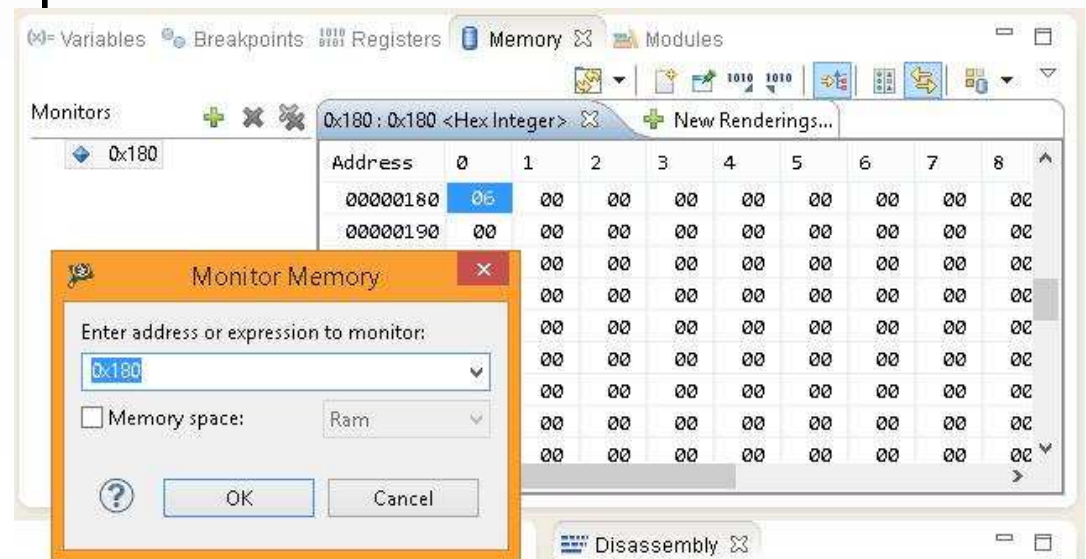
Debugger - paměť

- Při zobrazení obsahu proměnné v podstatě debugger ukazuje obsah paměti v místě, kde „leží“ proměnná
- Proměnná c na adrese 0x180



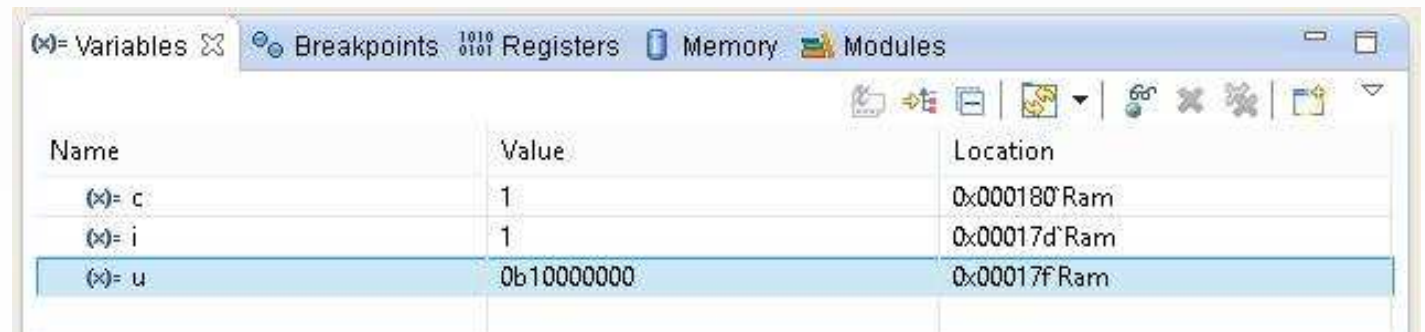
- Je také možné použít C-čkový operátor reference &, který reprezentuje adresu proměnné
- Změny proměnné = změny v paměti
 - Možno např. editovat

- Ověřte chování pro c = 126
 - Neukazuje se přetékání !!!
- Přidejte prom. typu na int
 - Ověřte chování pro 32787



Bitové operace

- Přidejte proměnnou u typu uint_8
 - Je stejný jako unsigned char v klasickém C/C++
 - Existují i typy uint16_t, uint32_t
 - Standardní typy dle ANSI C
 - Výhodou je zaručená přenositelnost mezi platformami
 - U „našeho“ překladače definovány v PE_Types.h
- Nastavte zobrazení „binary“ v záložce Variables



- Úkol: V každém průchodu cyklem měňte hodnotu nejvyššího bitu
 - Ověřte funkci debuggerem
 - POZOR, v C nelze zapisovat binární čísla

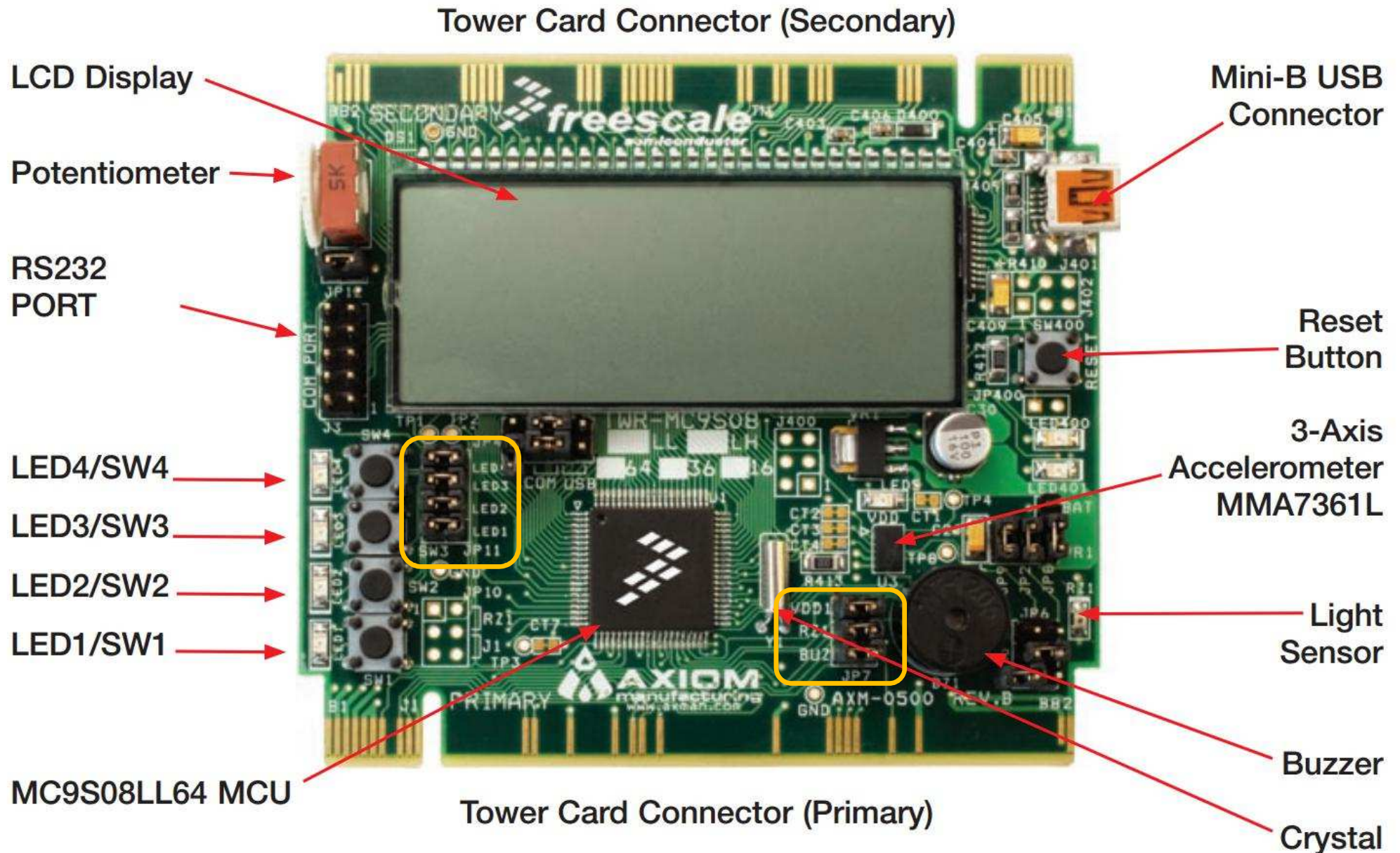
Bitové operace II

- Úkol 2: v proměnné `u` (`uint8_t`) vytvořte efekt „běžící log. 1“
 - Hodnoty `b00000001`, `b00000010`, ..., `b10000000`
- Upravte na opačný směr efektu
- Upravte na „běžící log. 0“
- Ověřte skutečnou hodnotu v paměti na adrese proměnné

Plán cvičení

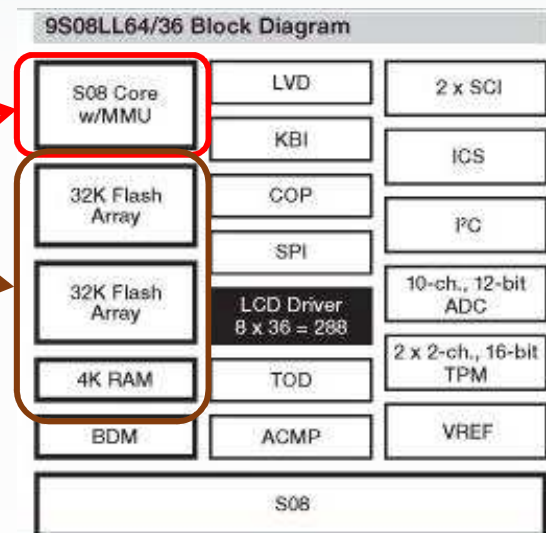
1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
- 2. I/O porty – tlačítka a LEDky**
3. Časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Vývojový kit Freescale S08LL64



Vnitřní bloky jednočipových mikropočítačů

- Jednočipový mikropočítač se skládá z:
 - Základní výpočetní jádro (jako mikroprocesor)
 - Paměť (někdy mohou být i externí)
 - RAM – typicky pro data/proměnné, zásobník
 - Typicky menší velikost
 - Realizováno jako SRAM
 - "ROM" – kód, konstanty
 - Dříve PROM, EPROM, nyní většinou FLASH
 - Periférie
 - Specializované funkční bloky, typicky možno vypínat kvůli spotřebě
 - Další podpůrné bloky
 - ISP - Programování v systému (u Freescale BDM – umí i debug)
 - Hlídní napájení, bloky Resetu, ...
- Organizace paměti (=paměťová mapa)
 - Popisuje umístění jednotlivých paměťových bloků v paměťovém prostoru (= na kterých adresách leží)
 - **Pozor, 8-bitové uP mají adresovou sběrnici 16b, takže max. rozsah paměti 64kB**



Ovládání vnitřních bloků uP

- V paměťovém prostoru na se určených adresách nachází "registry"
 - Zápisem vhodných hodnot se nastavuje činnost určitého bloku/periférie
 - Čtením se typicky získá stav
 - Pokud periférie poskytuje/vyžaduje data, má také "datový" registr
 - Podle architektury jádra je možný i bitový přístup k obsahu
 - Jinak nutné bity "maskovat" – využití operací AND/OR
- Seznam a popis registrů je hlavní součástí dokumentace ("Reference manual")
- Pro využití v C jsou názvy registrů a jejich bitů připraveny v .H souborech
 - Není třeba znát adresu, stačí název
 - Pracuje se formálně podobně jako s proměnnými

Typické vnitřní periférie

- I/O porty (někdy též GPIO = General Purpose I/O)
 - Organizovány ve formě "portů"
 - Šířka 8 bitů (pro 8b mikropočítač)
 - Přístup na celý port, u některých architektur také po bitech
 - Nastavitelný směr (I nebo O)
 - Nastavitelné parametry – pullup odpory, budiče, ...
 - Fyzické vývody sdílejí na obvodu s dalšími perifériemi – pozor při výběru pouzdra
- Komunikační sběrnice (sériové)
 - SPI – synchronní, data-in, data-out
 - I²C – synchronní, obousměrná data
 - UART – asynchronní
 - Typicky odpovídá COM portu u PC
 - Alternativně pojmenováno **SCI** (např. u Freescale !)
 - Specializované – CAN, USB, Ethernet, ...

Typické vnitřní periférie - II

- Časovače
 - Základem speciální typ registru, který přičítá impulsy (příp. odečítá) – viz "čítače" z KAE/CESx
 - Zdroj impulsů ("tiků") může být interní nebo externí
 - Další podpůrné obvody/bloky řídí činnost časovače
 - Děličky umožňují snižovat vstupní frekvenci
 - Registry pro zkrácení běhu (nepočítá se jen 0-MAX_ROZSAH, ale omezený rozsah) – různá řešení pro různé architektury
- A/D převodník
 - Typicky více vstupů, výběr konkrétního pomocí multiplexeru
 - Možnost vnitřní nebo vnější reference
 - Nastavitelná rychlost převodu

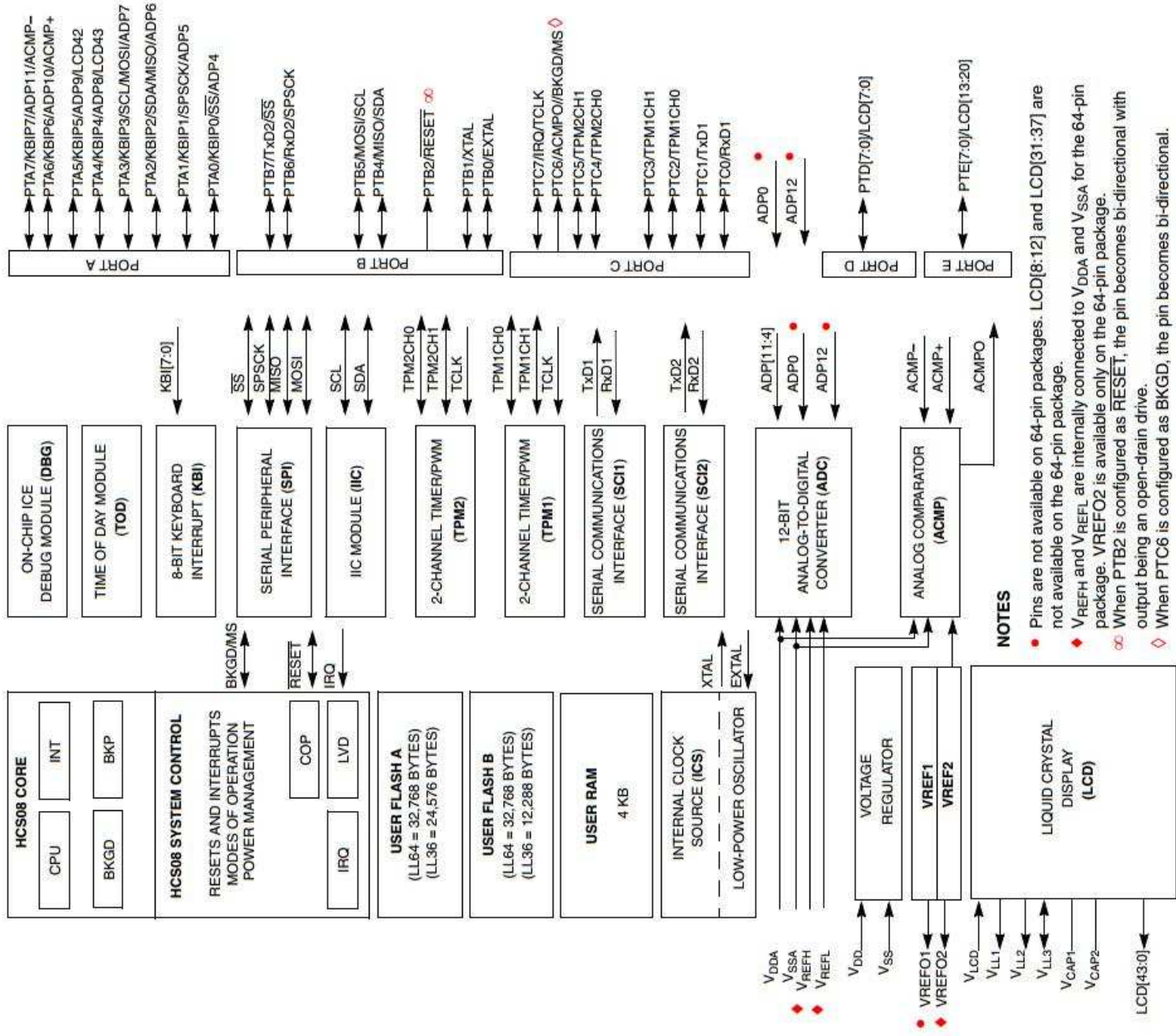
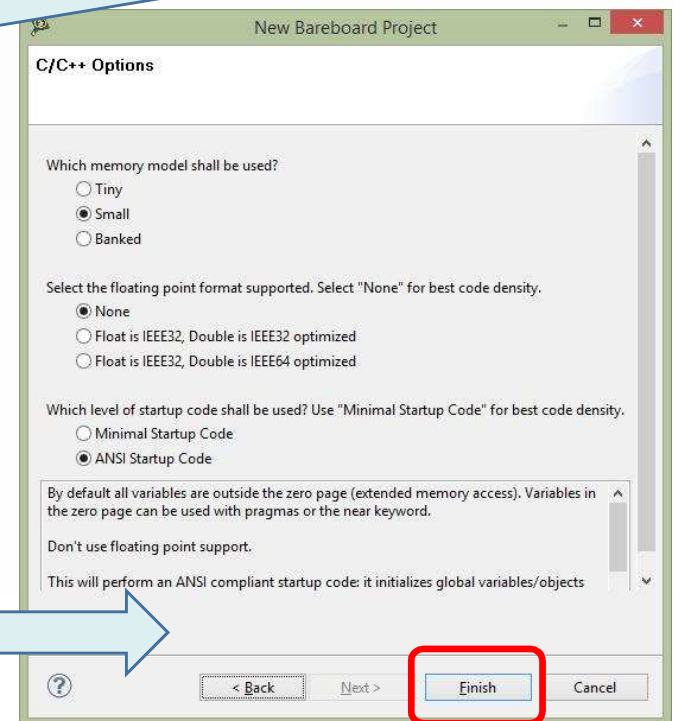
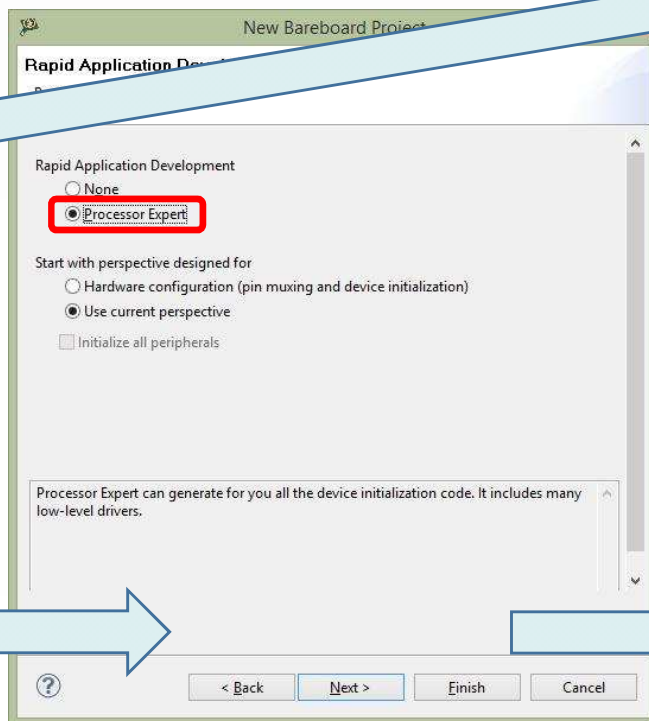
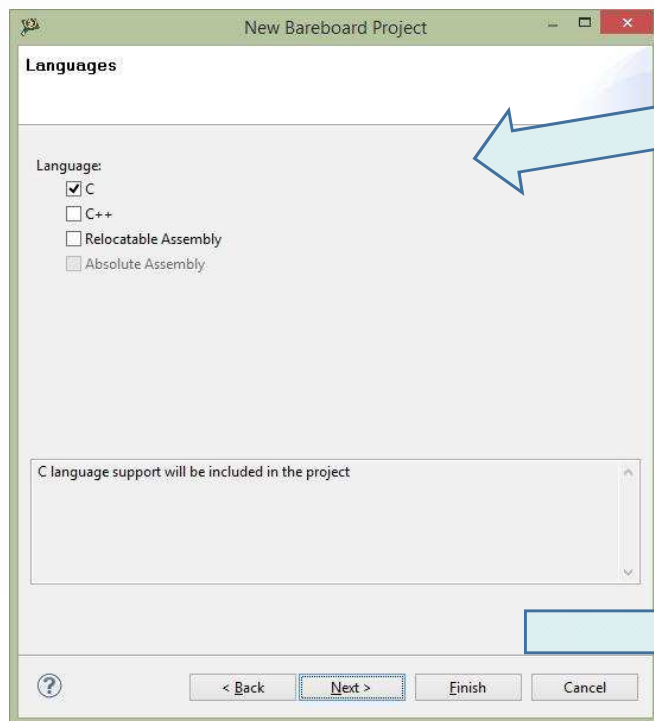
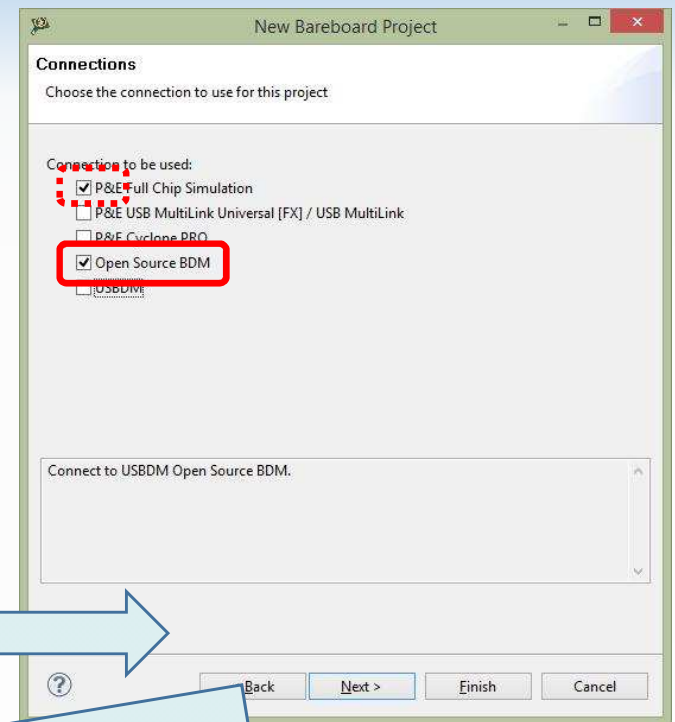
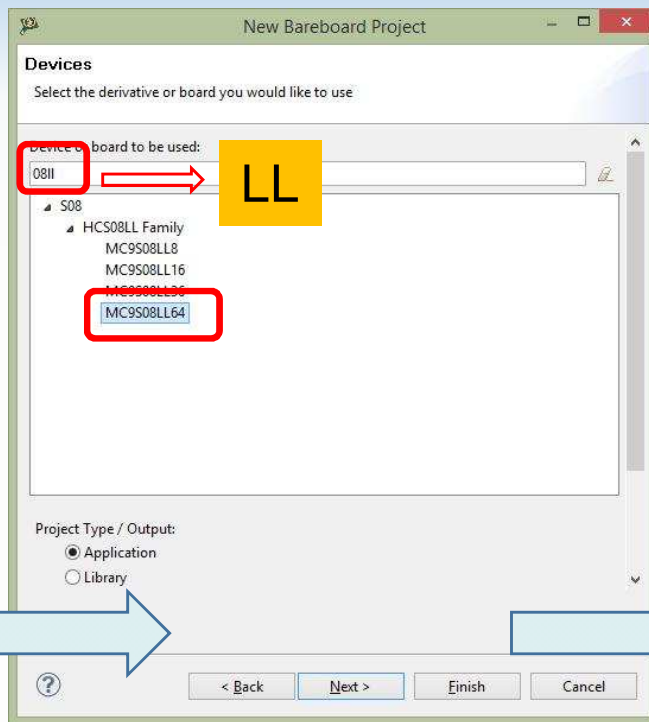
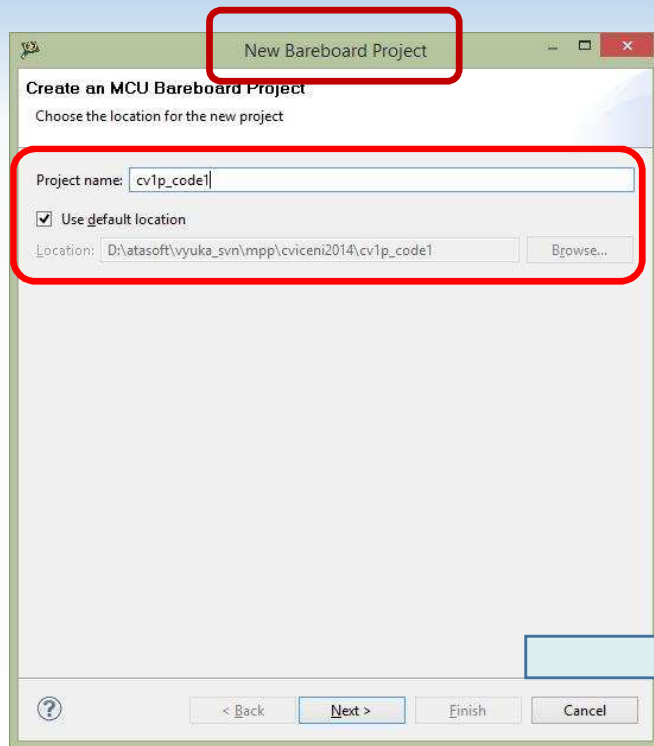


Figure 1-1. MC9S08LL64 Series Block Diagram



Processor Expert

- Přidat "Perspective" – Hardware
- Processor – nastavení
 - Typ pouzdra
 - Nezapomenout "Generate Code" !

The screenshot displays the CodeWarrior Development Studio interface with the Processor Expert configuration for the MC9S08LL64CLK microcontroller. The main window shows the chip pinout and internal components. The 'Component Inspector - Cpu' window on the right provides detailed configuration options.

Available Processor Packages:

- MC9S08LL64CLK 80-pins LQFP
- MC9S08LL64CLH 64-pins LQFP

Configuration R...

Reg. name	Init. value
Peripheral register	
ICSC1	06
ICSC2	40
ICSTRM	????????
ICSSC	0001000?
SRS	82
SOPT1	43
SOPT2	00
SDIDH	????0000
SDIDL	26
SPMSC1	1C
SPMSC2	02
SPMSC3	00
SCGC1	FF
SCGC2	FF
PTASE	00
PTADS	FF
PTBSE	08
PTBDS	F7
PTCSE	00

Processor

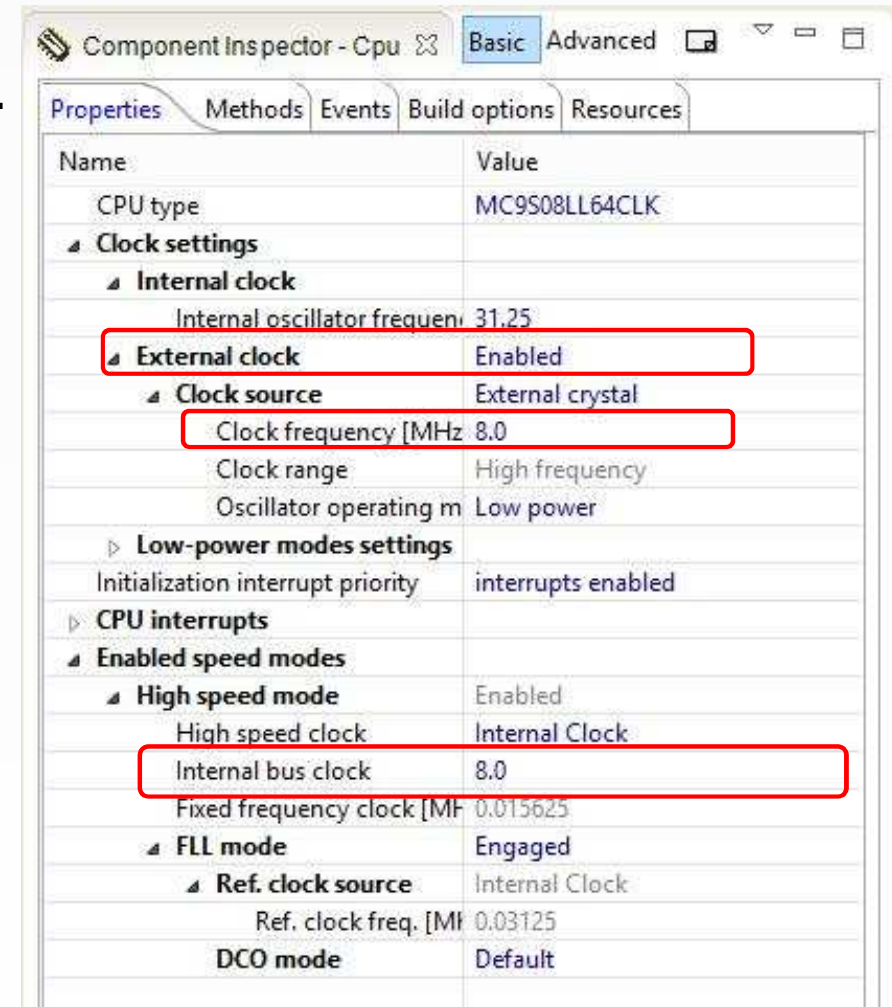
MC9S08LL64CLK

Component Inspector - Cpu

Name	Value
CPU type	MC9S08LL64CLK
Clock settings	
Initialization interrupt priority	interrupts enabled
CPU interrupts	
Enabled speed modes	
High speed mode	Enabled
High speed clock	Internal Clock
Internal bus clock	4.194304
Fixed frequency clock [MHz]	0.016384
FLL mode	Engaged
Ref. clock source	Internal Clock
Ref. clock freq. [MHz]	0.032768
DCO mode	Auto select

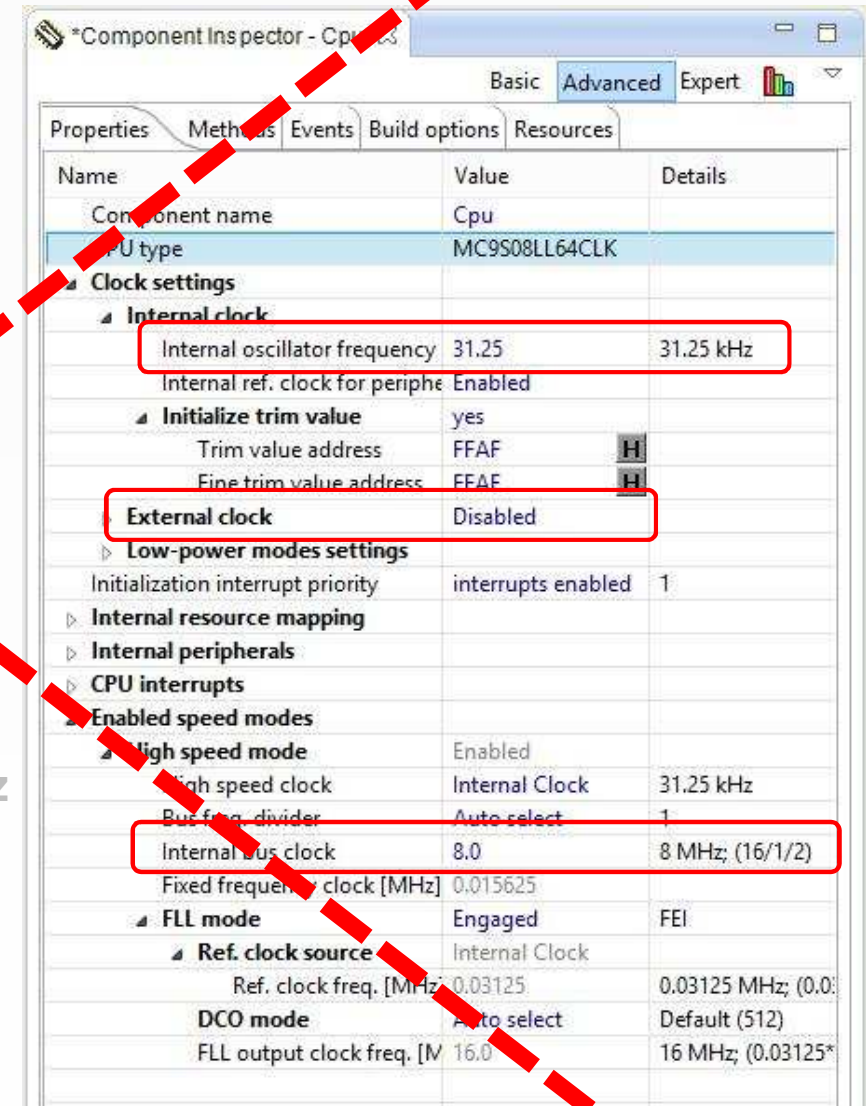
Volba zdroje hodin – externí krystal

- Procesor potřebuje pro svoji činnost hodiny
 - Pro každou periférii je možné hodiny zapínat, příp. vybrat zdroj a volitelně i děličky
- Na S08 máme k dispozici
 - Interní (méně přesný) RC generátor
 - Možnost externího krystalu nebo generátoru
 - Vnitřními děličkami a násobičkami (s PLL) možno vytvořit základní takt sběrnice
- Na vývojové desce osazen externí 8MHz
 - Budeme používat
 - Nastavíme "Internal System Bus" = **8.0MHz**
 - Max. kmitočet jádra 16MHz
- "Processor expert" napovídá použitelné hodnoty a v případě kolize nebo nemožnosti použít volbu varuje "vykřičníkem"

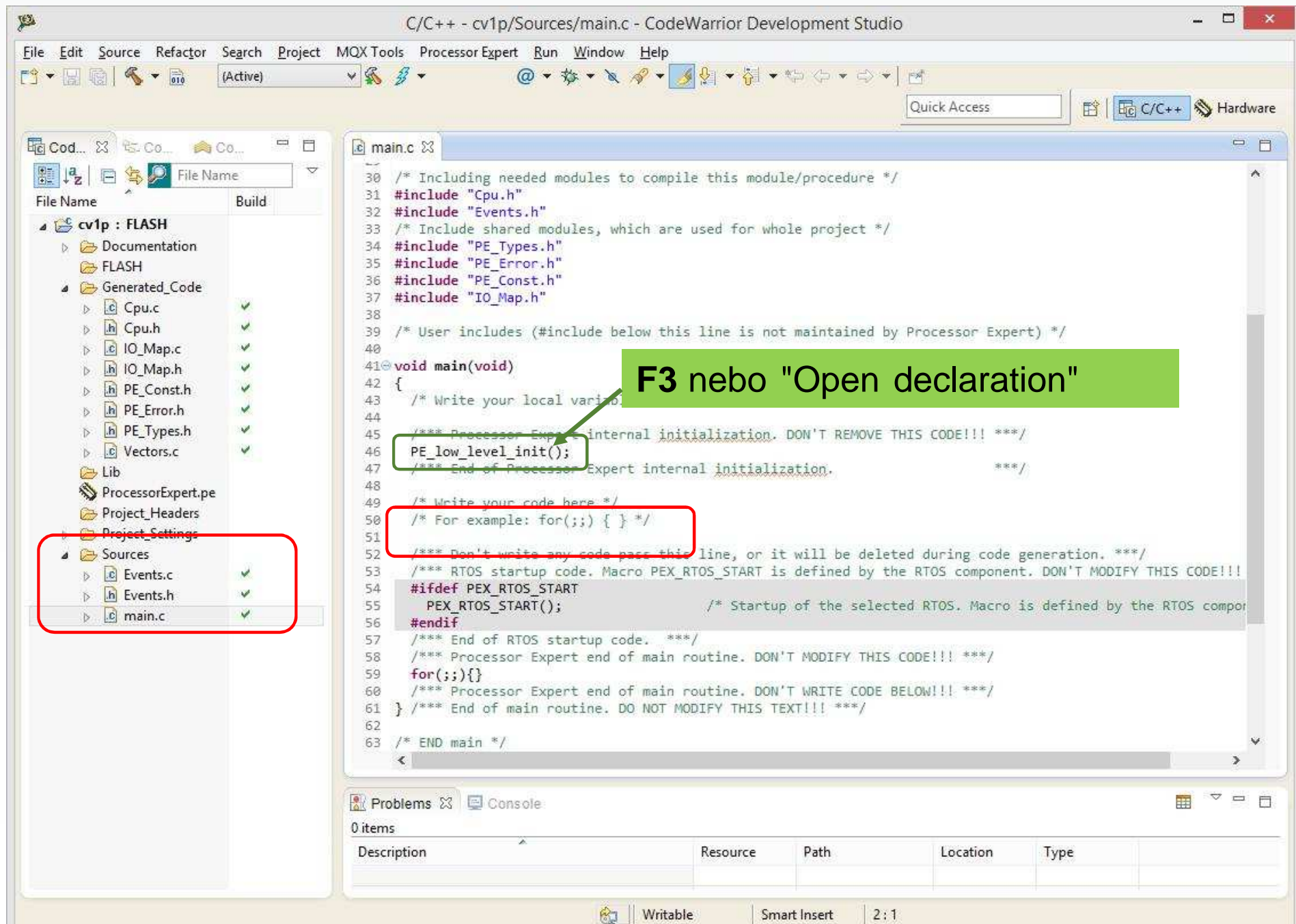


Volba zdroje hodin - interní

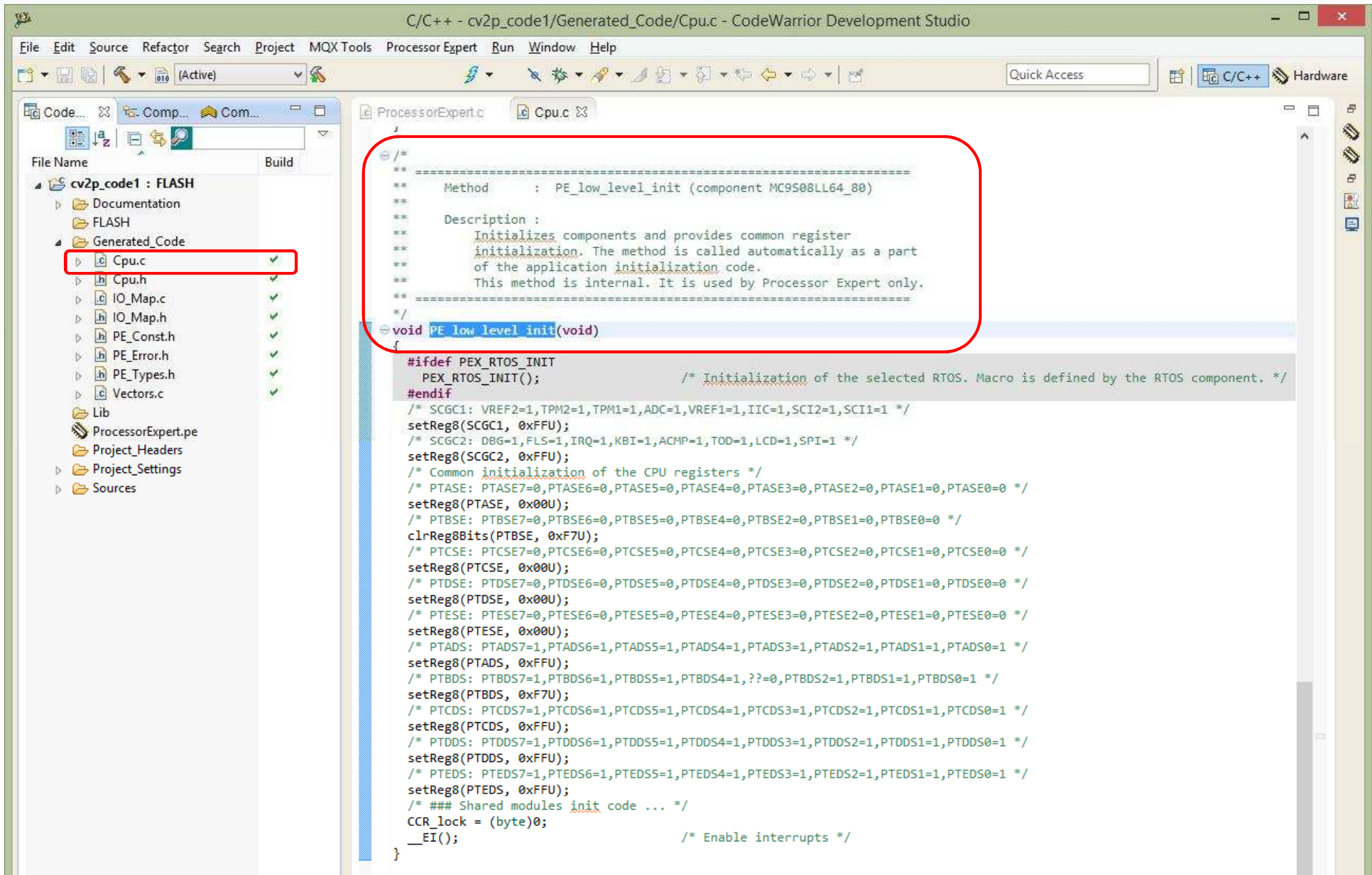
- Procesor potřebuje pro svoji činnost hodiny
 - Pro každou periférii je možné hodiny zapínat, příp. vybrat zdroj a volitelně i děličky
- Na S08 máme k dispozici
 - Interní (méně přesný) generátor
 - Možnost externího krystalu nebo generátoru
 - Vnitřními děličkami a násobkami (s PLL) možno vytvořit základní takt sběrnice
- Na vývojové desce externí 32.768kHz
 - Nelze z něj vydělit vhodný takt sběrnice
 - Určeno pro "časové" aplikace
- Budeme používat interní zdroj
 - Nastavitelný v rozsahu 25-41.66kHz – volíme 31.25kHz
 - Pozor, defaultně 32.768kHz, to nechceme
 - Dobře se násobí na "Internal System Bus" **8.0MHz**
- "Processor expert" napovídá použitelné hodnoty a v případě kolize nebo nemožnosti použít volbu varuje "vykřičníkem"



Vygenerovaná kostra aplikace



Vygenerovaná inicializace – CPU.C

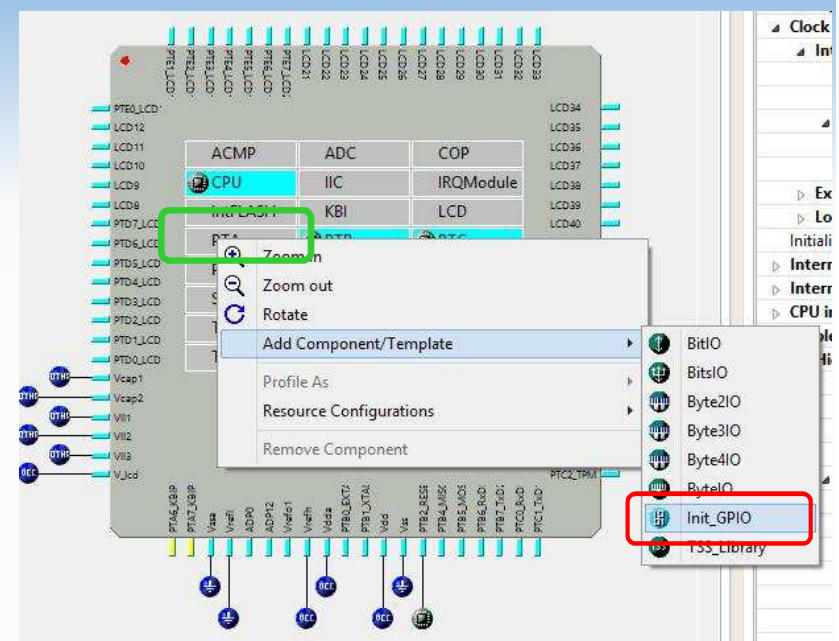


The screenshot displays the CodeWarrior Development Studio interface. On the left, the project tree shows the file `Cpu.c` under the `Generated_Code` folder, which is highlighted with a red box. The main editor window shows the code for `PE_low_level_init`, which is also highlighted with a red box. The code includes a description of the method and its initialization logic.

```
/*  
** Method      : PE_low_level_init (component MC9S08LL64_80)  
**  
** Description :  
**      Initializes components and provides common register  
**      initialization. The method is called automatically as a part  
**      of the application initialization code.  
**      This method is internal. It is used by Processor Expert only.  
**  
**  
**  
**  
**  
*/  
void PE_low_level_init(void)  
{  
    #ifdef PEX_RTOS_INIT  
        PEX_RTOS_INIT(); /* Initialization of the selected RTOS. Macro is defined by the RTOS component. */  
    #endif  
    /* SCGC1: VREF2=1,TPM2=1,TPM1=1,ADC=1,VREF1=1,IIC=1,SCI2=1,SCI1=1 */  
    setReg8(SCGC1, 0xFFU);  
    /* SCGC2: DBG=1,FLS=1,IRQ=1,KBI=1,ACMP=1,TOD=1,LCD=1,SPI=1 */  
    setReg8(SCGC2, 0xFFU);  
    /* Common initialization of the CPU registers */  
    /* PTASE: PTASE7=0,PTASE6=0,PTASE5=0,PTASE4=0,PTASE3=0,PTASE2=0,PTASE1=0,PTASE0=0 */  
    setReg8(PTASE, 0x00U);  
    /* PTBSE: PTBSE7=0,PTBSE6=0,PTBSE5=0,PTBSE4=0,PTBSE3=0,PTBSE2=0,PTBSE1=0,PTBSE0=0 */  
    clrReg8Bits(PTBSE, 0xF7U);  
    /* PTCSE: PTCSE7=0,PTCSE6=0,PTCSE5=0,PTCSE4=0,PTCSE3=0,PTCSE2=0,PTCSE1=0,PTCSE0=0 */  
    setReg8(PTCSE, 0x00U);  
    /* PTDS: PTDS7=0,PTDS6=0,PTDS5=0,PTDS4=0,PTDS3=0,PTDS2=0,PTDS1=0,PTDS0=0 */  
    setReg8(PTDS, 0x00U);  
    /* PTESE: PTESE7=0,PTESE6=0,PTESE5=0,PTESE4=0,PTESE3=0,PTESE2=0,PTESE1=0,PTESE0=0 */  
    setReg8(PTESE, 0x00U);  
    /* PTADS: PTADS7=1,PTADS6=1,PTADS5=1,PTADS4=1,PTADS3=1,PTADS2=1,PTADS1=1,PTADS0=1 */  
    setReg8(PTADS, 0xFFU);  
    /* PTBDS: PTBDS7=1,PTBDS6=1,PTBDS5=1,PTBDS4=1,PTBDS3=1,PTBDS2=1,PTBDS1=1,PTBDS0=1 */  
    setReg8(PTBDS, 0xF7U);  
    /* PTCDS: PTCDS7=1,PTCDS6=1,PTCDS5=1,PTCDS4=1,PTCDS3=1,PTCDS2=1,PTCDS1=1,PTCDS0=1 */  
    setReg8(PTCDS, 0xFFU);  
    /* PTDDS: PTDDS7=1,PTDDS6=1,PTDDS5=1,PTDDS4=1,PTDDS3=1,PTDDS2=1,PTDDS1=1,PTDDS0=1 */  
    setReg8(PTDDS, 0xFFU);  
    /* PTEDS: PTEDS7=1,PTEDS6=1,PTEDS5=1,PTEDS4=1,PTEDS3=1,PTEDS2=1,PTEDS1=1,PTEDS0=1 */  
    setReg8(PTEDS, 0xFFU);  
    /* ### Shared modules init code ... */  
    CCR_lock = (byte)0;  
    __EI(); /* Enable interrupts */  
}
```

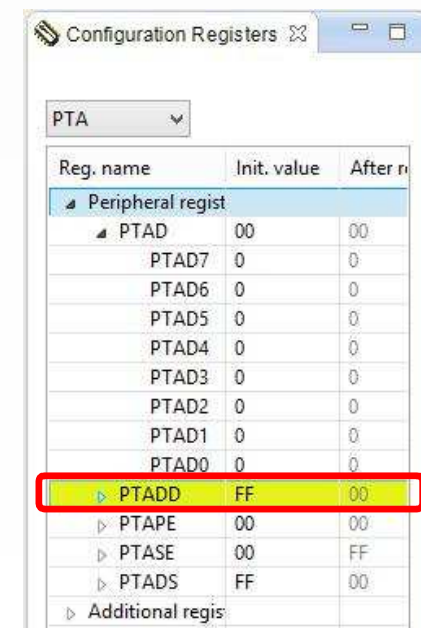
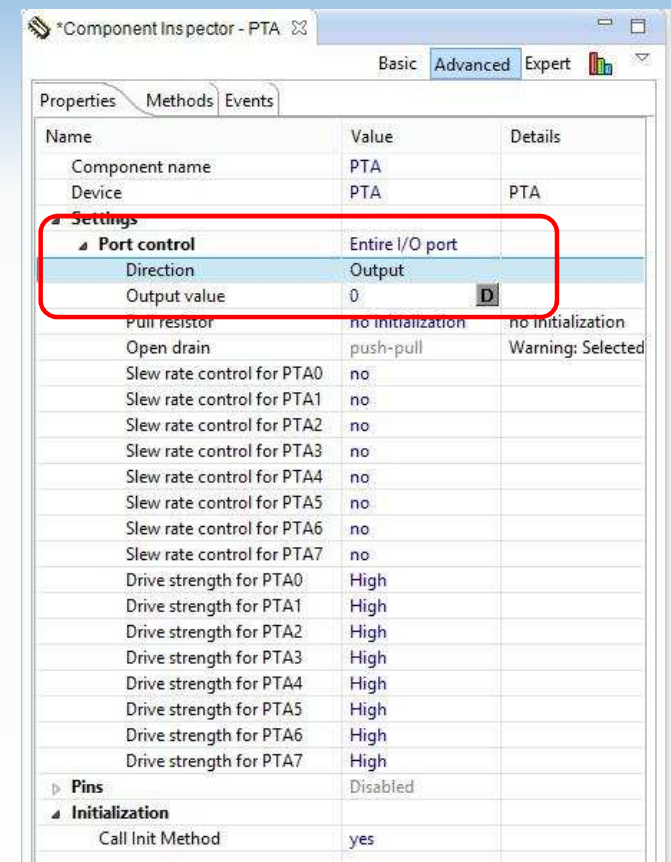
Přidání komponenty do PE

- Přidávat další bloky lze v PE
- Pravým "myšítkem"
 - "Add Component/Template"
 - Doporučuji pouze Init_xxx
 - Pak se objeví vlastnosti a komponenta je aktivní
 - Ukazují se i dotčené vývody na pouzdře
- Přidáme **PTA** – 8-bitová výstupní brána
- Podobně lze komponentu odebrat – "Remove Component"
- Pozor, neodebírat "jádro procesoru" – komponenty se symbolem čipu
 - Projekt by přestal být kompilovatelný (neznámý procesor)
 - U našeho **S08LL** si jádro rezervuje ještě po jednom bitu z PTC a PTB
 - Nebudeme moci využít plně všech jejich 8 I/O bitů
 - **Symbol jádra procesoru je zde 3x, neodebírat !!**



Konfigurace I/O brány

- Nastavení v "Properties"
 - Defaultně "Input"
 - Nastavíme "Output"
 - Pracujeme s "Entire I/O Port"
 - Je možné nastavovat i bity jednotlivě
 - Volba "Individual Pins"
 - Je povoleno "Call Init Method"
- Vlevo se aktualizují "Configuration Registers"
 - I/O brány mají více registrů
 - **PTxD** – Data – datový registr
 - **PTxDD** – Direction – určuje směr I/O
 - **PTxAPE** – Pull Enable – interní pullupy pro vstupy
 - **PTxASE** – Slew Rate – omezuje strmost hran pro Output
 - **PTxADS** – Drive Strength – umožňuje pinu dodávat větší proud
 - Jednotlivé bity jsou většinou pojmenovány a přístupné v kódu
- Nezapomenout uložit změny – "Generate Code"



Práce s bránou v kódu

- Vytvořit proměnnou typu byte
 - Pohled na deklaraci (F3)
 - Soubor **PE_Types.h**
 - **typedef unsigned char byte;**
- V nekonečném cyklu přičítáme 8-bitovou hodnotu a vkládáme do PTAD (Data Registr brány A)
- Pokud bychom nechtěli oddělenou deklaraci a inicializaci proměnných od samotného kódu, je možné použít trik s C-čkovým blokem



```
42 void main(void)
43 {
44     /* Write your local variable definition here */
45     byte b;
46
47     /** Processor Expert internal initialization. DON'T REMOVE */
48     PE_low_level_init();
49     /** End of Processor Expert internal initialization.
50
51     /* Write your code here */
52     /* For example: for(;;) { } */
53
54     while(1)
55     {
56         PTAD = b;
57         b++;
58     }
59
60     /** Don't write any code pass this line, or it will be
61     /** Processor Expert end of main routine. DON'T MODIFY
62     for(;;){}
63     /** Processor Expert end of main routine. DON'T WRITE C
64     } /** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/
65 }
```

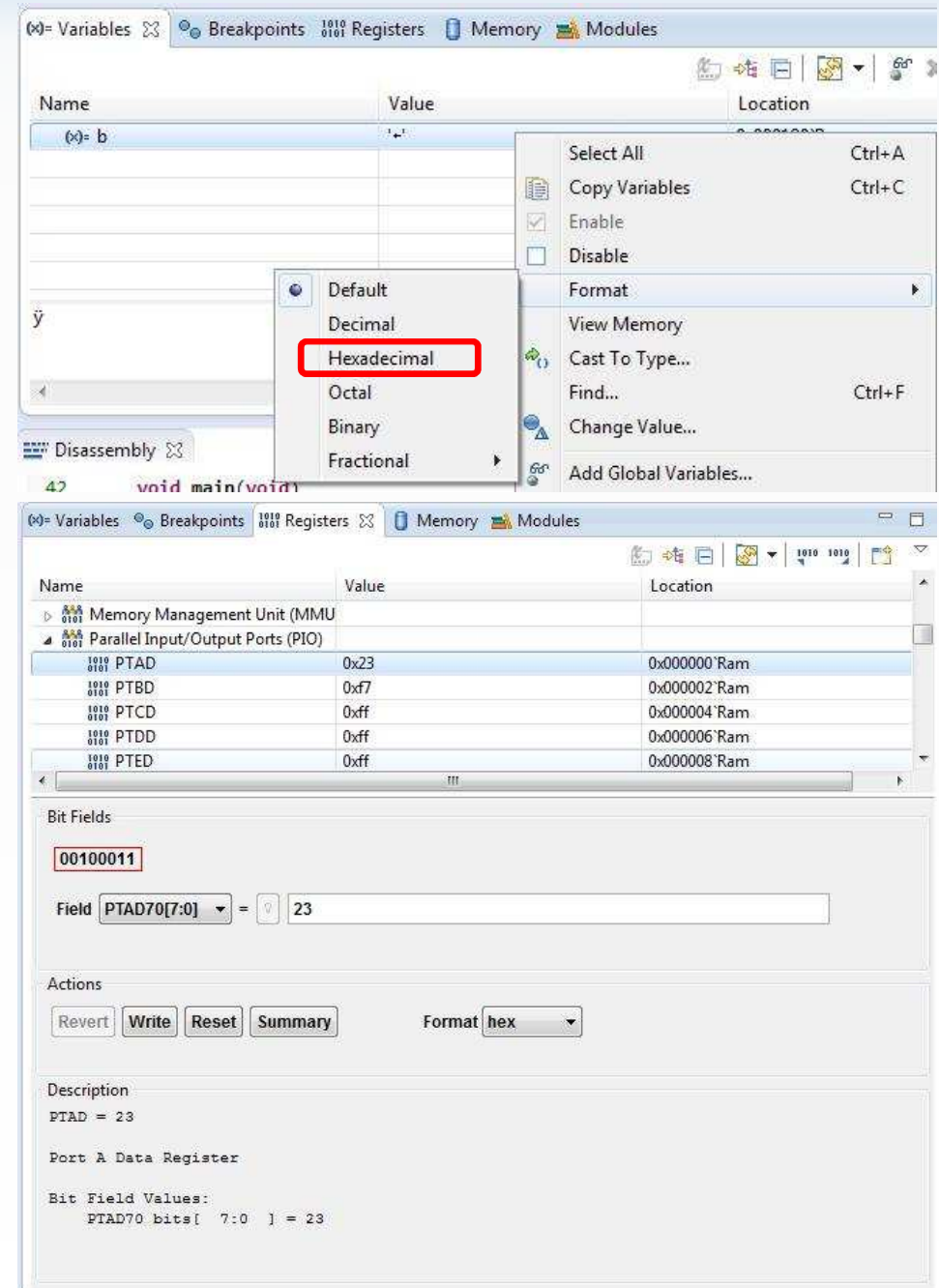
```
...
{
    byte b;

    while(1)
    {
        PTAD = b;
        b++;
    }
}
...
```


Krokování



- **F5 – Step Into** – krok, vstupuje do funkcí
- **F6 – Step Over** – krok, "nevleze" do funkce
- **F7 – Step Return** – dokončí funkci
- **F8 – Resume** – pokračuje ve vykonávání, příp. spustí od začátku (napoprvé)
- Aktuální řádek vyznačen 
- Je možné krokovat i po instrukcích assembleru – přepínání 
- Pokud je program zastaven, je možné prohlížet proměnné, paměť i registry
 - Obsah je možné i měnit



Ladění obsahu paměti

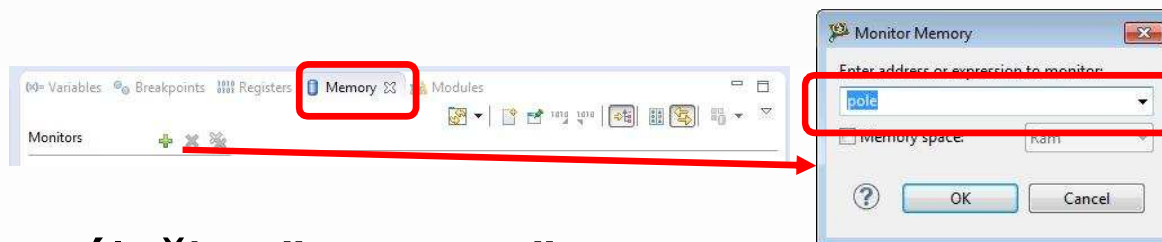
- Kód plnicí pole hodnotou
- Jednou 0x00, pak 0xFF
- Spustit v Debug režimu

```
...
{
#define VELIKOST 10

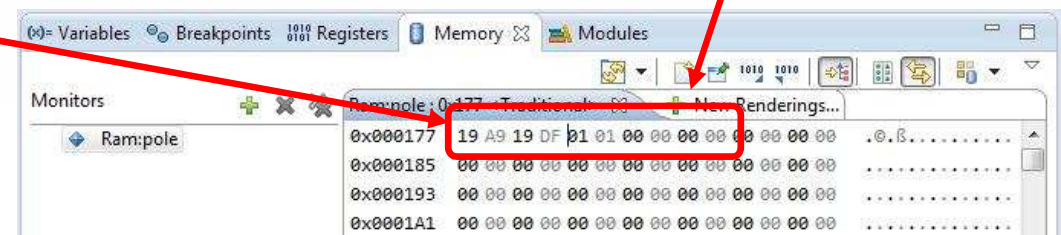
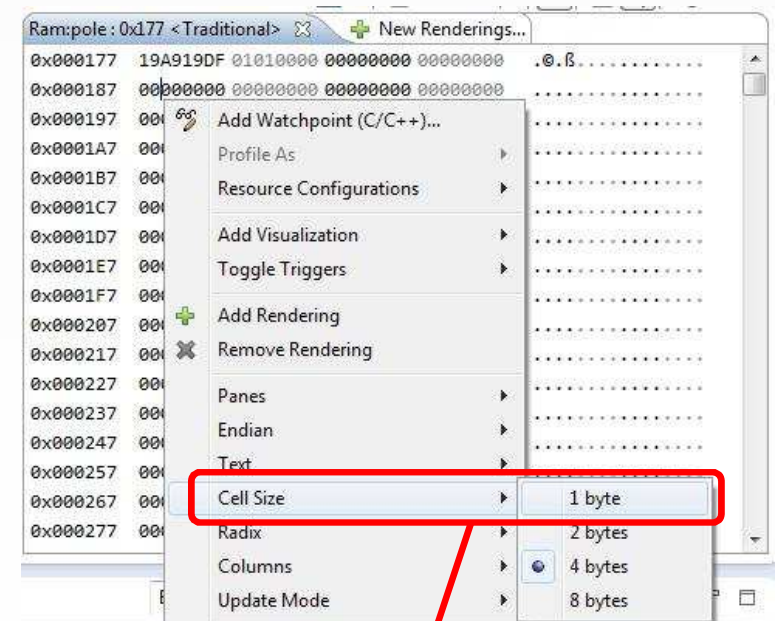
    byte b;
    byte x = 0x00;
    byte pole[VELIKOST];

    while(1)
    {
        for (b = 0; b < VELIKOST; b++)
            pole[b] = x;

        x = ~x;
    }
}
...
```

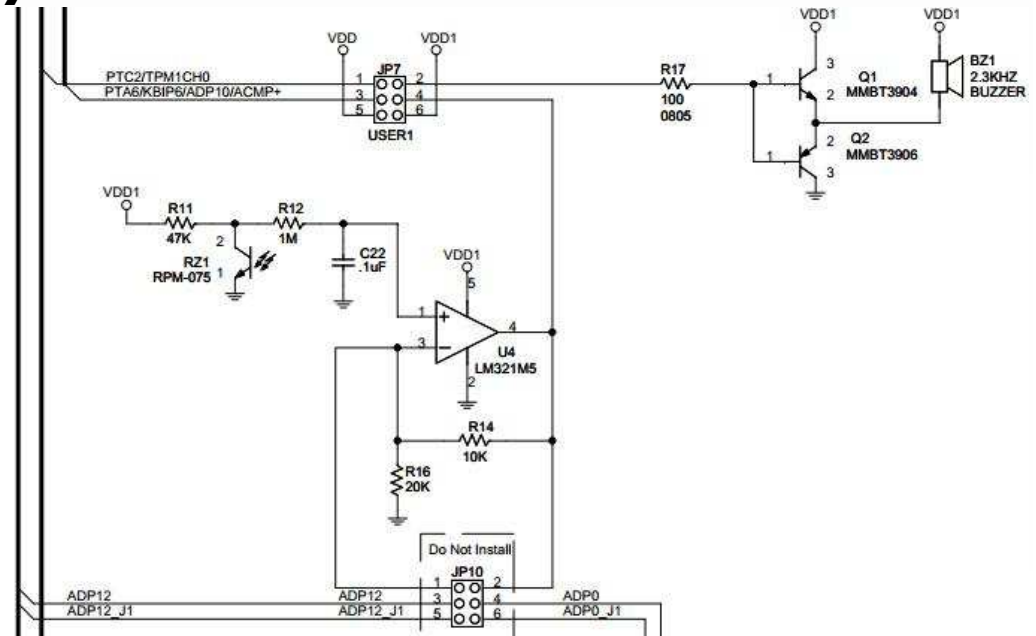
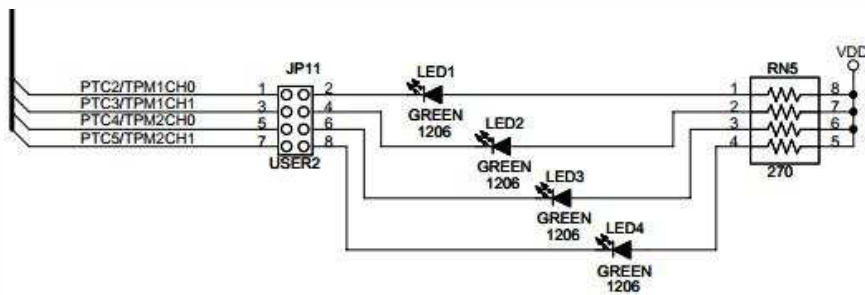
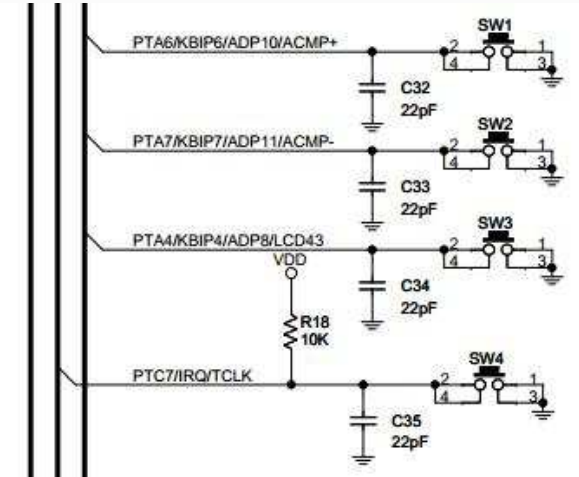


- Záložka "Memory"
 - Přidat "adresu" – možno též název pole (= ukazatel)
 - Možnost výběru zobrazení – např. po bytech
- Pozor, paměť obsazená polem není inicializovaná
- Sledujte změny paměti
 - Krokování programu



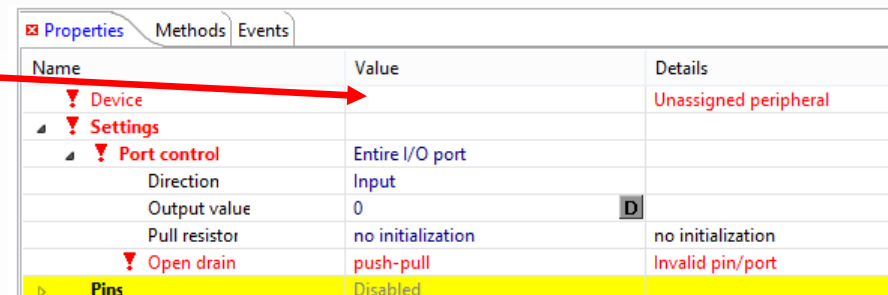
Zapojení externího HW – LEDky a tlačítka

- Čtveřice tlačítek připojena na **PTA** (3x) a na **PTC** (1x)
 - Aktivní (=stisknuto) je log. 0
- LED připojeny na **PTC** – bity 2–5
 - Zapojeny přes R z VDD
 - Svíí se log. 0
 - Příp. zkontrolovat pole "jumperů" **JP11**
- Na **PTC2** připojen ještě "buzzer"
 - Možno vypnout jumperem **BUZ** na **JP7**
- Na **PTA6** připojen snímač osvětlení
 - Rozpojit střední jumper **RZ1** na **JP7**



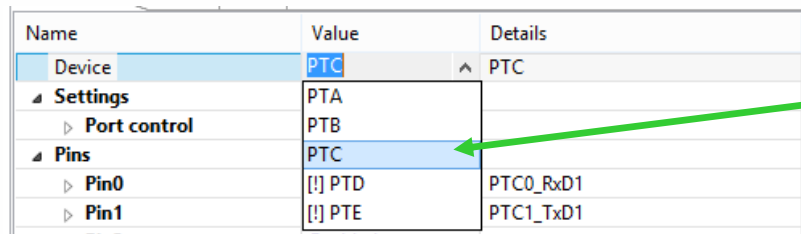
Nastavení vývodu – ProcessorExpert

- V minulém příkladu nastavena celá brána PTA
- Pro PTB a PTC v našem procesoru nelze vybrat celý port
 - Sdílí se vývody se signálem RESET a BDM
 - Nutno nastavit individuálně
- Kliknutí pravým tlačítkem na PTC (ne na ikonu procesoru)
 - Add component/template – Init GPIO
 - Hlášena chyba = není vybrán port
 - Nutno vybrat PTC
- Port Control = Individual pins
 - Pin2 Enabled a Output



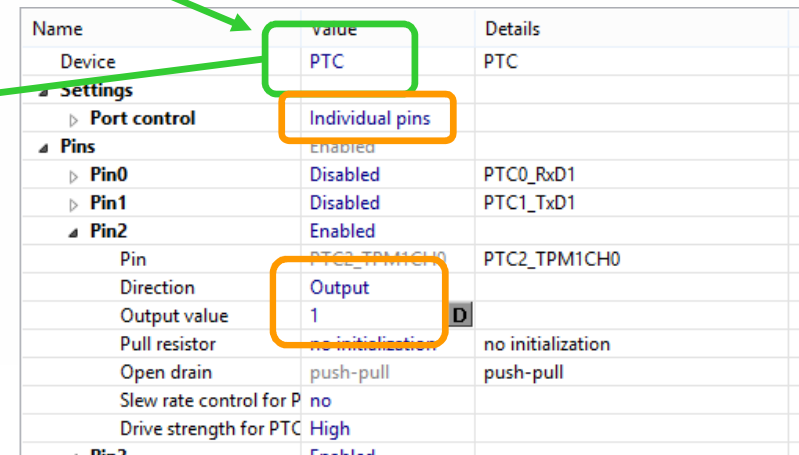
This screenshot shows the 'Properties' window with the 'Port control' section expanded. A red arrow points from the text 'Hlášena chyba = není vybrán port' to the 'Device' field, which is marked with a red warning icon and labeled 'Unassigned peripheral'. Another red arrow points from the text 'Nutno vybrat PTC' to the 'Port control' section.

Name	Value	Details
Device		Unassigned peripheral
Settings		
Port control	Entire I/O port	
Direction	Input	
Output value	0	
Pull resistor	no initialization	no initialization
Open drain	push-pull	Invalid pin/port
Pins	Disabled	



This screenshot shows the 'Port control' dropdown menu open, with 'PTC' selected. A green arrow points from the text 'Nutno vybrat PTC' to this selection.

Name	Value	Details
Device	PTC	PTC
Settings		
Port control	PTA	
	PTB	
	PTC	
Pins		
Pin0	[!]	PTD RxD1
Pin1	[!]	PTE TxD1



This screenshot shows the 'Pin2' configuration. A green box highlights the 'Device' field set to 'PTC'. An orange box highlights the 'Port control' dropdown set to 'Individual pins'. Another orange box highlights the 'Output' checkbox, which is checked, and the 'Output value' is set to '1'. A green arrow points from the text 'Pin2 Enabled a Output' to the 'Output' checkbox.

Name	Value	Details
Device	PTC	PTC
Settings		
Port control	Individual pins	
Pins		
Pin0	Disabled	PTC0_RxD1
Pin1	Disabled	PTC1_TxD1
Pin2	Enabled	PTC2_TPM1CH0
Pin	PTC2_TPM1CH0	
Direction	Output	
Output value	1	
Pull resistor	no initialization	no initialization
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	

Blikání LED

```
/* Write your local variable definition here */

/** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
PE_low_level_init();
/** End of Processor Expert internal initialization. */

/* Write your code here */
/* For example: for(;;) { } */

while(1)
{
    int w;    // pomocna promenna pro cekaci cyklus, lokalni v cyklu

    PTCD_PTC2 = !PTCD_PTC2; // negace stavu, zde muze byt "logicka", je bin promenna

    for (w = 0; w < 30000; w++)
        ;
}
```

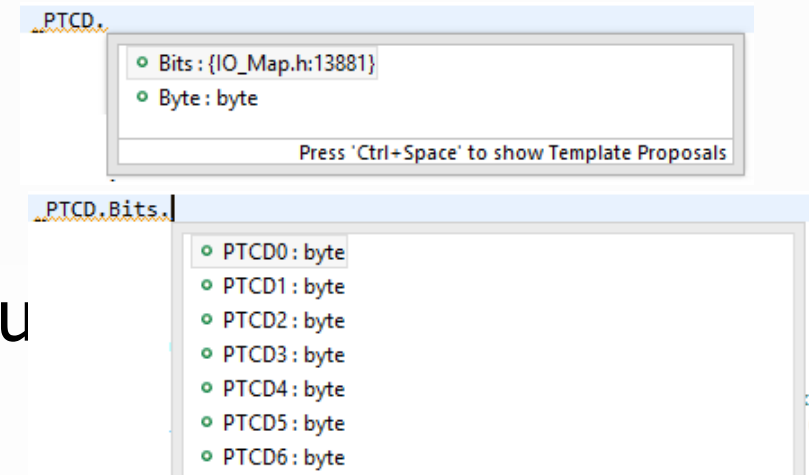
- Umístěno ve funkci **main** (viz. komentáře)

Bitový přístup k hodnotám registrů

- Většina registrů umožňuje bitový přístup
- Je možno buď použít přímo bit
 - Jméno složeno jako **Registr_Bit**
 - Definováno v **IO_Map.H**
- Nebo přistoupit přes bitovou strukturu a union s celou hodnotou
 - Místní nápověda nabízí prvky struktur
 - Jména registrů začínají _
- Názvy dle dokumentace
- Jména viditelná též v PE
 - Panel vlevo

```
...  
// možný zápis  
PTCD_PTC2 = 1;  
// nebo  
_PTCD.Bits.PTC2 = 1;  
...
```

Macro Expansion
_PTCD.Bits.PTC2
Press 'F2' for focus



```
289 /** PTCD - Port C Data Register; 0x00000004 */  
290 typedef union {  
291     byte Byte;  
292     struct {  
293         byte PTC0      :1; /* Port C Data Register Bit 0 */  
294         byte PTC1      :1; /* Port C Data Register Bit 1 */  
295         byte PTC2      :1; /* Port C Data Register Bit 2 */  
296         byte PTC3      :1; /* Port C Data Register Bit 3 */  
297         byte PTC4      :1; /* Port C Data Register Bit 4 */  
298         byte PTC5      :1; /* Port C Data Register Bit 5 */  
299         byte PTC6      :1; /* Port C Data Register Bit 6 */  
300         byte PTC7      :1; /* Port C Data Register Bit 7 */  
301     } Bits;  
302 } PTCDSTR;  
303 extern volatile PTCDSTR _PTCD @0x00000004;  
304 #define PTCD _PTCD.Byte  
305 #define PTCD_PTC0 _PTCD.Bits.PTC0  
306 #define PTCD_PTC1 _PTCD.Bits.PTC1  
307 #define PTCD_PTC2 _PTCD.Bits.PTC2  
308 #define PTCD_PTC3 _PTCD.Bits.PTC3  
309 #define PTCD_PTC4 _PTCD.Bits.PTC4  
310 #define PTCD_PTC5 _PTCD.Bits.PTC5  
311 #define PTCD_PTC6 _PTCD.Bits.PTC6  
312 #define PTCD_PTC7 _PTCD.Bits.PTC7
```

Name	Value	Details
Device	PTC	PTC
Settings		
Port control		
Pins		
Pin0	[I] PTD	PTC0_RxD1
Pin1	[I] PTE	PTC1_TxD1

Nastavení v CW - PE

Name	Value	Details
Device	PTA	PTA
Settings		
Port control	Individual pins	
Pins	Enabled	
Pin0	Disabled	PTA0_KBIP0_SS_ADP4
Pin1	Disabled	PTA1_KBIP1_SPSCCK_ADP5
Pin2	Disabled	PTA2_KBIP2_SDA_MISO_ADP6
Pin3	Disabled	PTA3_KBIP3_SCL_MOSI_ADP7
Pin4	Enabled	
Pin	PTA4_KBIP4_ADP8	PTA4_KBIP4_ADP8_LCD43
Direction	Input	
Output value	0	
Pull resistor	pull up	pull up
Open drain	open drain	open drain
Slew rate control for P	no	
Drive strength for PTA	High	
Pin5	Disabled	PTA5_KBIP5_ADP9_LCD42
Pin6	Enabled	
Pin	PTA6_KBIP6_ADP10	PTA6_KBIP6_ADP10_ACMPPPLUS
Direction	Input	
Output value	0	
Pull resistor	pull up	pull up
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTA	High	
Pin7	Enabled	
Pin	PTA7_KBIP7_ADP11	PTA7_KBIP7_ADP11_ACMPPMINUS
Direction	Input	
Output value	0	
Pull resistor	pull up	pull up
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTA	High	
Initialization		
Call Init Method	yes	

Name	Value	Details
Device	PTC	PTC
Settings		
Port control	Individual pins	
Pins	Enabled	
Pin0	Disabled	PTC0_RxD1
Pin1	Disabled	PTC1_TxD1
Pin2	Enabled	
Pin	PTC2_TPM1CH0	PTC2_TPM1CH0
Direction	Output	
Output value	1	
Pull resistor	no initialization	no initialization
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	
Pin3	Enabled	
Pin	PTC3_TPM1CH1	PTC3_TPM1CH1
Direction	Output	
Output value	1	
Pull resistor	no initialization	no initialization
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	
Pin4	Enabled	
Pin	PTC4_TPM2CH0	PTC4_TPM2CH0
Direction	Output	
Output value	1	
Pull resistor	no initialization	no initialization
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	
Pin5	Enabled	
Pin	PTC5_TPM2CH1	PTC5_TPM2CH1
Direction	Output	
Output value	1	
Pull resistor	no initialization	no initialization
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	
Pin6	Disabled	PTC6_ACMPO_BKGD_MS
Pin7	Enabled	
Pin	PTC7_IRQ_TCLK	PTC7_IRQ_TCLK
Direction	Input	
Output value	0	
Pull resistor	pull up	pull up
Open drain	push-pull	push-pull
Slew rate control for P	no	
Drive strength for PTC	High	
Initialization		
Call Init Method	yes	

Přidán definiční kód v main.c

- Doporučené "define"
 - Nemusí se stále vypisovat porty a piny
 - V případě změny HW stačí jen upravit na jednom místě
- Místo v kódu doporučené komentářem
 - Před funkcí main

```
/* User includes (#include below this line is not maintained by Processor Expert) */
#define LED1          PTCD_PTCD2
#define LED2          PTCD_PTCD3
#define LED3          PTCD_PTCD4
#define LED4          PTCD_PTCD5

#define BUTTON1       PTAD_PTAD6
#define BUTTON2       PTAD_PTAD7
#define BUTTON3       PTAD_PTAD4
#define BUTTON4       PTCD_PTCD7
```

Blikání LED

```
/* Write your local variable definition here */

/** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */
PE_low_level_init();
/** End of Processor Expert internal initialization. */

/* Write your code here */
/* For example: for(;;) { } */

while(1)
{
    int w;    // pomocna promenna pro cekaci cyklus, lokalni v cyklu

    LED2 = !LED2; // negace stavu, zde muze byt "logicka", protoze binarni promenna

    for (w = 0; w < 30000; w++)
        ;
}
```

- Stále umístěno ve funkci **main** (viz. komentáře)
- K zamyšlení – delší čekání

Běžící bod na LED – v1

```
byte b = 0;
word w;

...

while(1)
{
    b++;
    if (b >= 4)
        b = 0;

    PTCD |= 0x3C;                // 0011 1100 = nastaví log.1 na vystupy pro LED

    switch(b)
    {
        case 0: LED1 = 0; break; // opet svitime log.0 !!
        case 1: LED2 = 0; break;
        case 2: LED3 = 0; break;
        case 3: LED4 = 0; break;
    }

    for (w = 0; w < 60000; w++)
        ;
}
```

Běžící bod na LED – v2

```
byte b = 0;
word w;

...

while(1)
{
    b++;
    if (b >= 4)
        b = 0;

    LED1 = !(b == 0);           // negace, protoze svitime log.0 !!
    LED2 = !(b == 1);
    LED3 = !(b == 2);
    LED4 = !(b == 3);

    for (w = 0; w < 60000; w++)
        ;
}
```

Počítadlo na 4xLED

```
byte b = 0;
word w;
...
while(1)
{
    b++;
    if (b > 0x0f)                // cislo vetsi nez 15 (0000 1111) uz prekracuje 4 bity
        b = 0;

    PTCD = (PTCD & 0xc3) | (~(b << 2) & 0x3c);
        // 1100 0011 - vynulovat stredni 4 bity
        // obsah b posunout na stredni 4b (tedy o 2 vpravo)
        // sviti se log.0, takže nutno hodnotu v b bitove znegovat
        // 0011 1100 - ostatni bity (horni 2 a dolni 2) vynulovat
        // pomoci bitoveho OR nastavit bity v PTCD registru

    for (w = 0; w < 60000; w++)
        ;
}
```

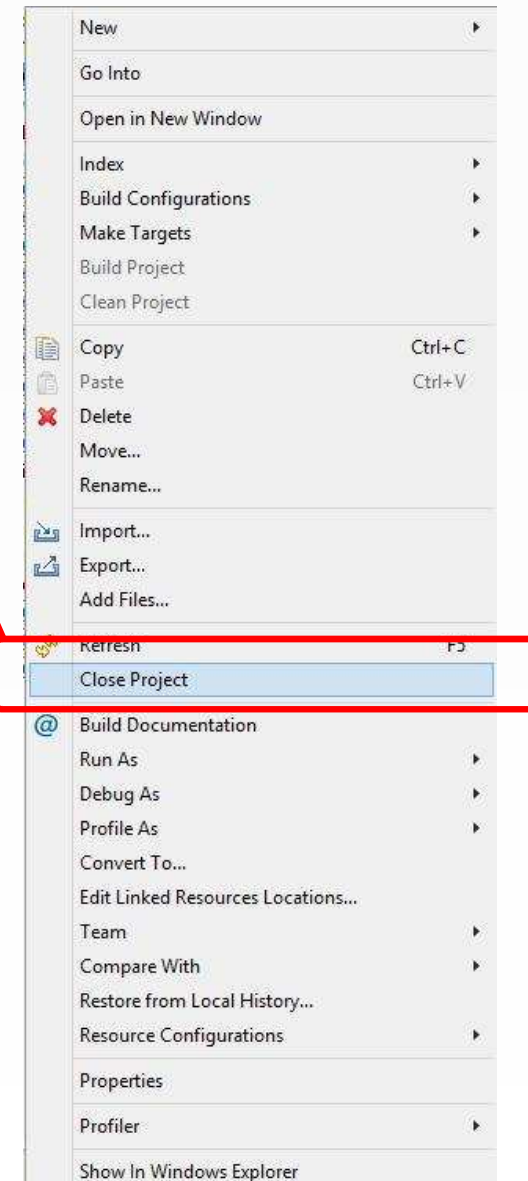
PTCD	PPPP PPPP	
& 0xc3	PP00 00PP	1100 0011
B	XXXX AAAA	
b << 2	xxAA AA00	
negace	XXBB BB11	
& 0x3c	00BB BB00	0011 1100
finále	PPBB BBPP	P = puvodni obsah PTCD, B = negovane spodni 4 bity z b

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
- 3. Tlačítka, časovač a přerušení**
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Sbalení neaktivních projektů

- Při více otevřených projektech může být náročnější orientace v tom, co se aktuálně edituje, překládá a spouští
 - Zároveň mnoho aktivních projektů zpomaluje IDE
- Je vhodné nepoužívané projekty "zavřít"
 - Ve vlastnostech projektu
- Není problém dle potřeby zavřený soubor otevřít
 - Volba "Open Project"



Zap/vyp jedním tlačítkem

```
bool bb = FALSE;

...

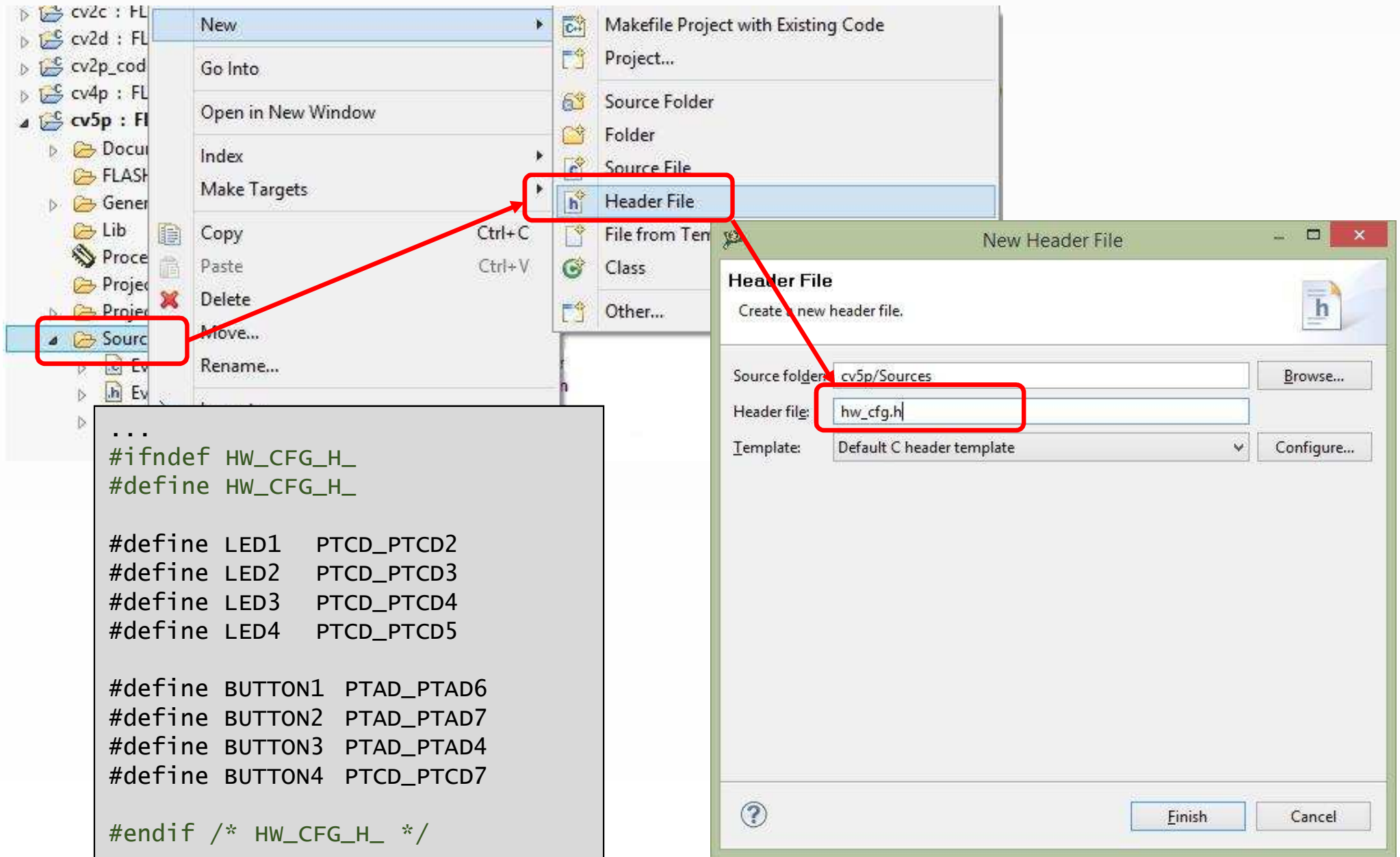
while(1)
{
    if (bb != BUTTON1)
    {
        bb = BUTTON1;

        if (bb)
            LED1 = !LED1;
    }
}
```

- V podstatě vytváříme klopný obvod
 - Změna stavu po uvolnění tlačítka
 - Nestisknuté je log. 1
 - Potřeba pomocná proměnná, která si "pamatuje" minulý stav
 - Vyhodnocujeme změnu stavu
 - Vhodný datový typ je "binární" – nabývá hodnotu 0 nebo 1 (false/true)
 - Zde se používá datový typ **bool**
 - Na jiných systémech se lze setkat i s boolean
 - Interně je to běžně jako **typedef**:

```
typedef unsigned char bool;
```

Hlavičkový soubor s HW konfigurací

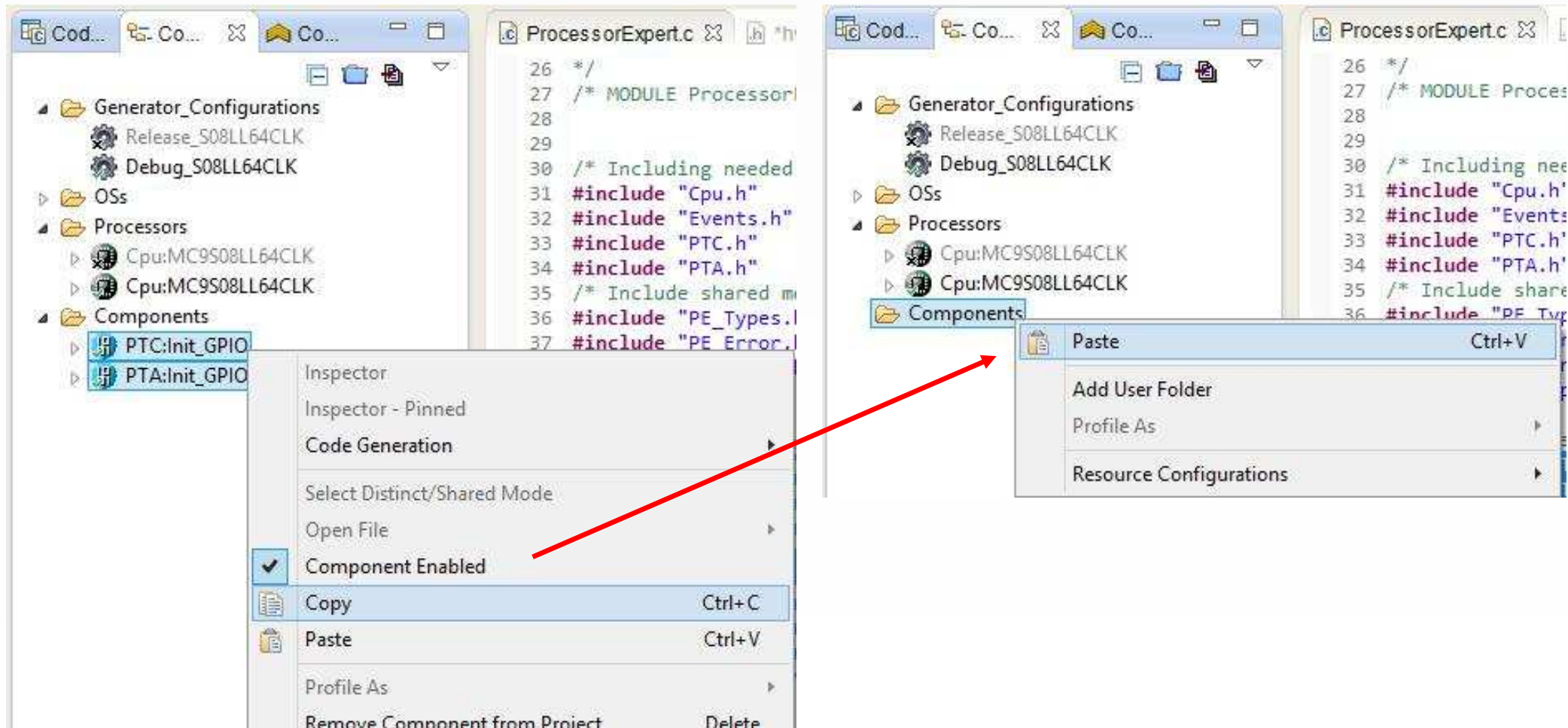


The image illustrates the steps to create a new header file in an IDE. The 'Sources' folder is selected in the project tree, and the 'Header File' option is chosen from the 'New' menu. The 'New Header File' dialog box shows the source folder as 'cv5p/Sources' and the header file name as 'hw_cfg.h'. The resulting header file content is shown in the code snippet below.

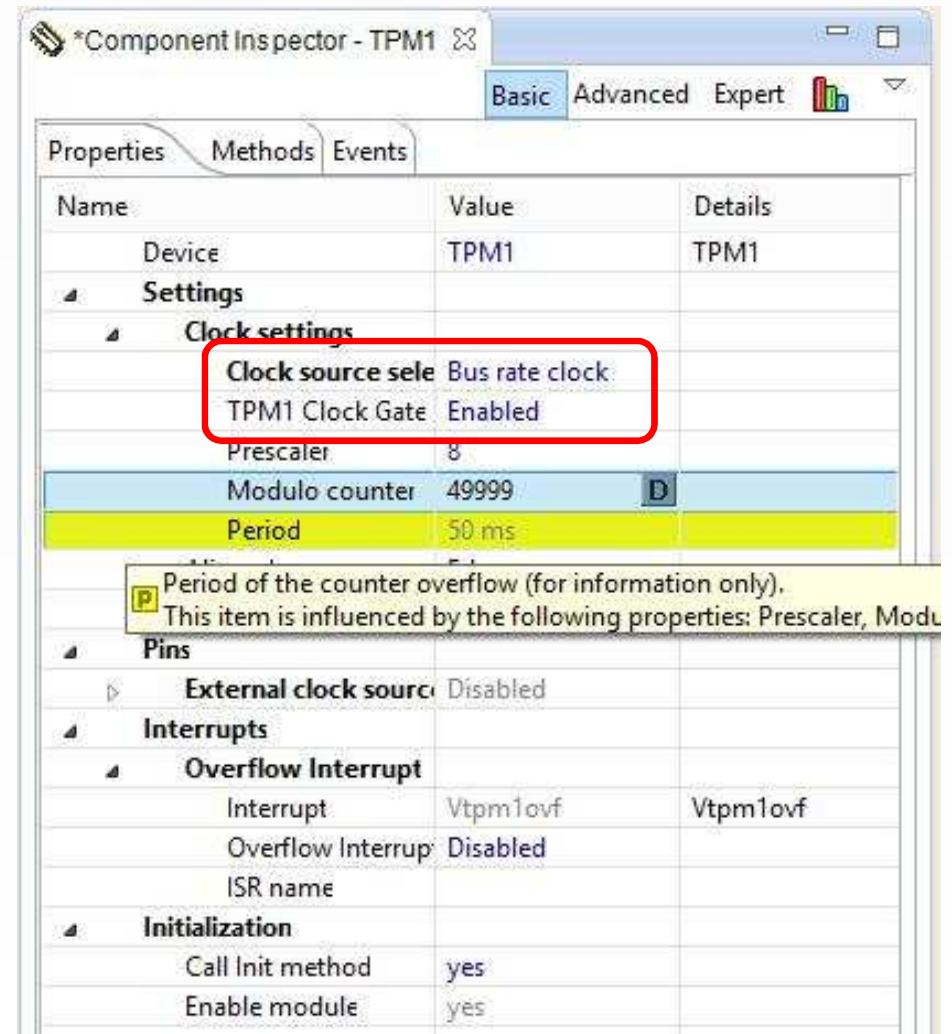
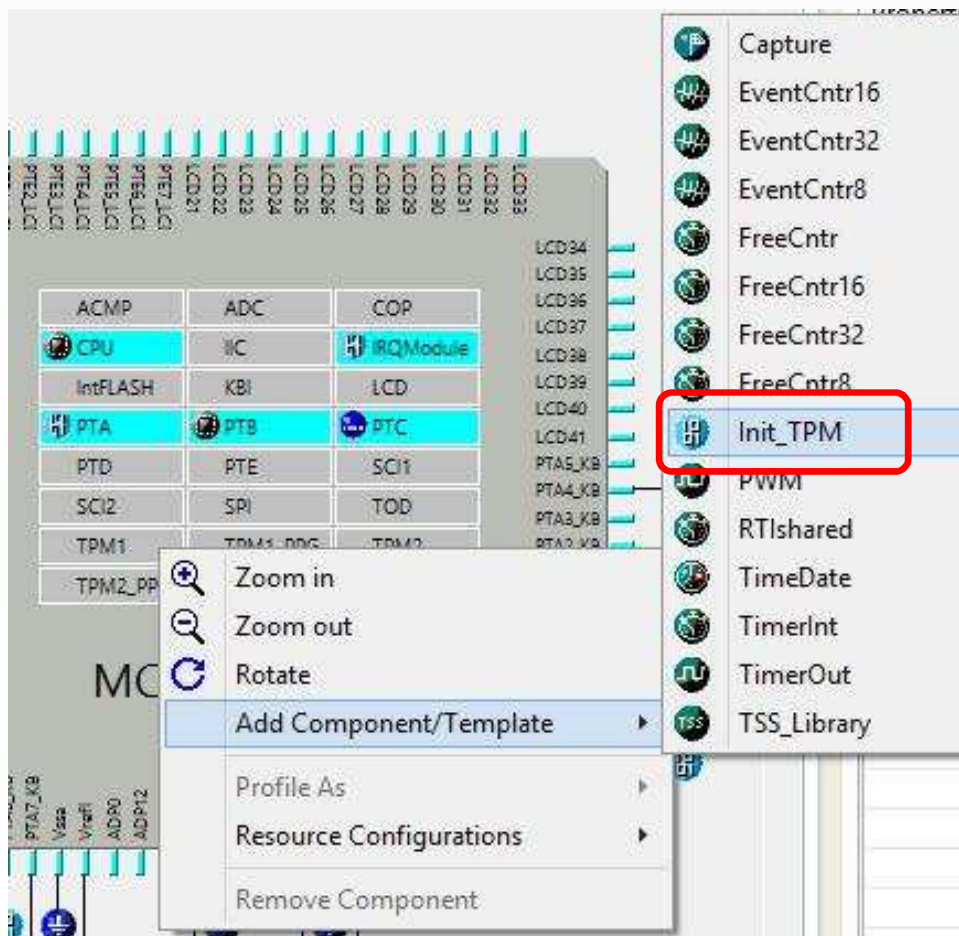
```
...  
#ifndef HW_CFG_H_  
#define HW_CFG_H_  
  
#define LED1    PTC_D_PTC_D2  
#define LED2    PTC_D_PTC_D3  
#define LED3    PTC_D_PTC_D4  
#define LED4    PTC_D_PTC_D5  
  
#define BUTTON1 PTAD_PTAD6  
#define BUTTON2 PTAD_PTAD7  
#define BUTTON3 PTAD_PTAD4  
#define BUTTON4 PTC_D_PTC_D7  
  
#endif /* HW_CFG_H_ */
```

Kopírování komponent mezi projekty

- Nezapomenout externí "hodiny" na **8MHz** a Internal BUS na **8MHz**
- Záložka "Components"
 - Komponenty z jednoho projektu možno kopírovat do jiného včetně všech nastavení



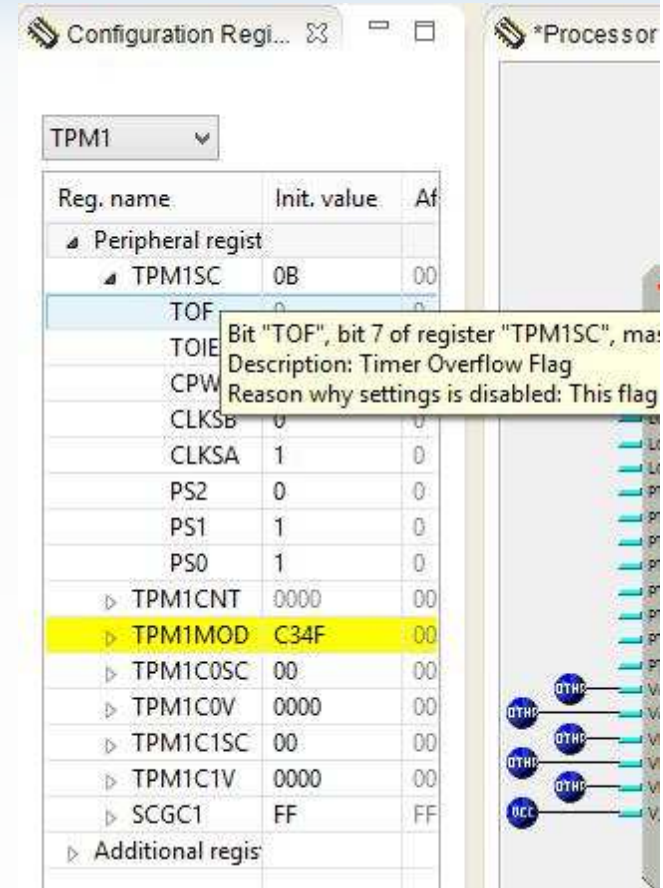
Přidání inicializace časovače



Práce s časovačem

- Každý blok TPM má svoji sadu registrů
 - Kap. 17 v Ref. Manual
 - TPMxSC – Status and Control Register
 - TOF – příznak Overflow
 - Nastává při přetečení do 0
 - Nastaven na log.1
 - Programově nutno "shodit" = nastavit log.0
 - Ostatní nastavovány v PE
 - Prescaler – dělení vstupních hodin
 - Výběr kombinací 3 bitů v TPMxSC
 - V PE se automaticky vypočte perioda
- Nahradíme čekání for cyklem čekáním na dopočítání časovače (= příznak TOF)

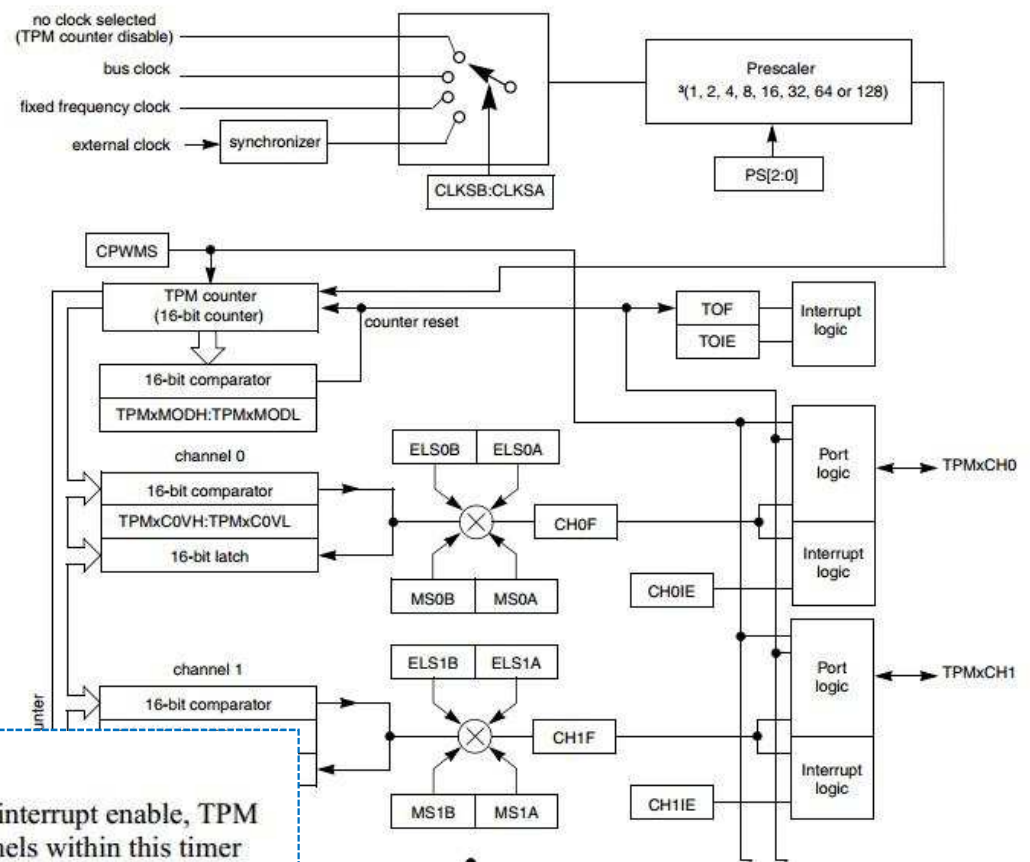
```
...  
// for(w = 0; w < 50000; w++)  
// while(!TPM1SC_TOF)  
//     ;  
//  
//     TPM1SC_TOF = 0;  
...
```



Clock settings	
Clock source sele	Bus rate clock
TPM1 Clock Gate	Enabled
Prescaler	32
Modulo counter	1
Period	2
Aligned	4
Channels	8
ins	16
External clock source	32
Interrupts	64
Overflow Interrupt	128

Čítače/časovače S08

- Základem jsou vstupní hodiny
- Nastavitelný před-dělič
- 2-kanálový blok
 - PWM, komparátor, CAPCOM



17.3.1 TPM Status and Control Register (TPMxSC)

TPMxSC contains the overflow status flag and control bits used to configure the interrupt enable, TPM configuration, clock source, and prescale factor. These controls relate to all channels within this timer module.

	7	6	5	4	3	2	1	0
R	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
W	0							
Reset	0	0	0	0	0	0	0	0

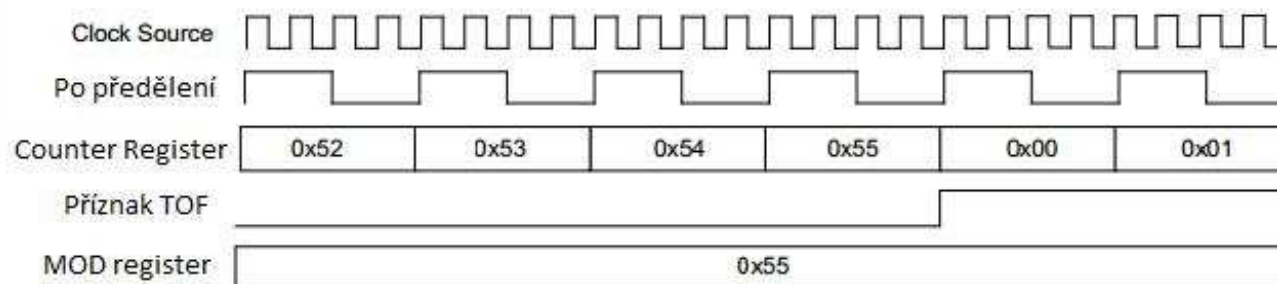
Figure 17-7. TPM Status and Control Register (TPMxSC)

Table 17-2. TPMxSC Field Descriptions

Field	Description
7 TOF	Timer overflow flag. This read/write flag is set when the TPM counter resets to 0x0000 after reaching the modulo value programmed in the TPM counter modulo registers. <u>Clear TOF by reading the TPM status and control register when TOF is set and then writing a logic 0 to TOF. If another TPM overflow occurs before the clearing sequence is completed, the sequence is reset so TOF remains set after the clear sequence was completed for the earlier TOF. This is done so a TOF interrupt request cannot be lost during the clearing sequence for a previous TOF. Reset clears TOF. Writing a logic 1 to TOF has no effect.</u> 0 TPM counter has not reached modulo value or overflow. 1 TPM counter has overflowed.
6	Timer overflow interrupt enable. This read/write bit enables TPM overflow interrupts. If TOIE is set, an interrupt is

- Příznak přetečení TOF
- Po detekci přetečení (čtením registru SC) je nutno příznak vynulovat (=smazat)

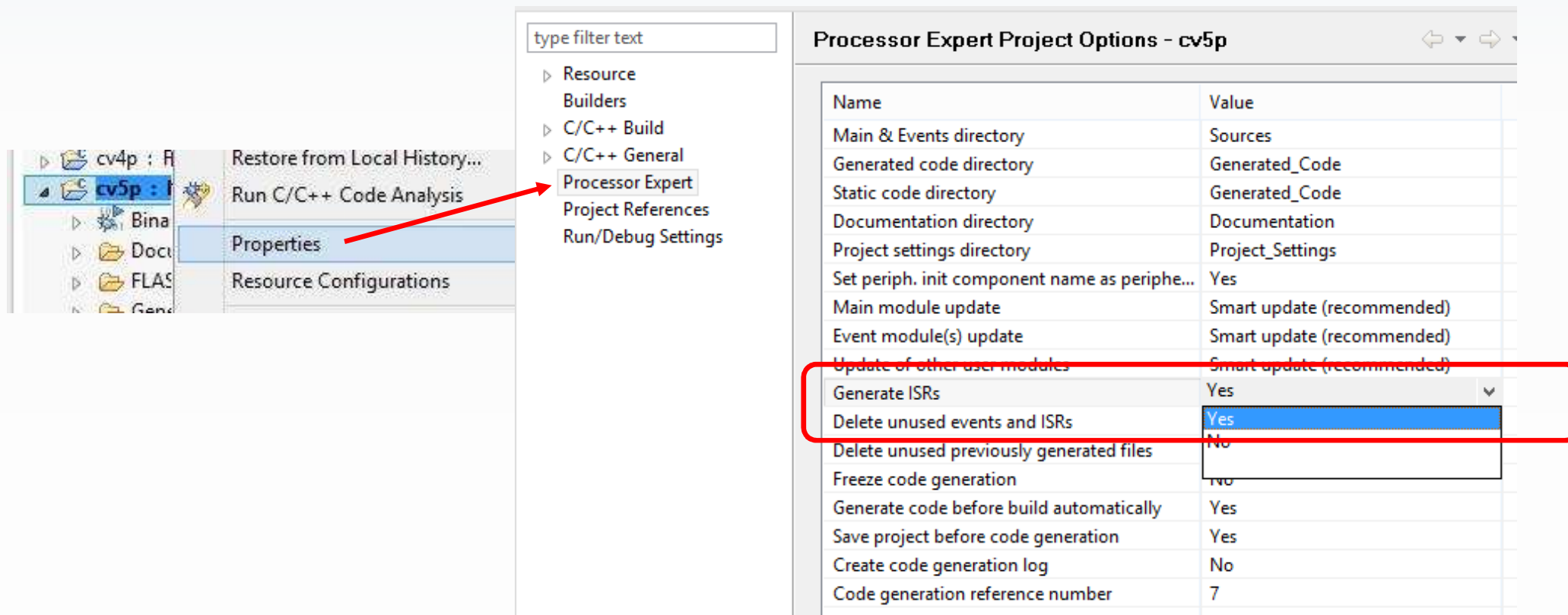
- V blocích časovačů u různých výrobců se používá různých strategií pro nastavení periody opakování přetečení
- Vyhodnocuje se hodnota čítacího (counter) registru
 - Zde 16-bitový **TPMxCNT**(L/H) přístupný přes 8-bitové poloviny
- Freescale řešení využívá modulo registr
 - 16-bitový registr **TPMxMOD**(L/H)
 - Když hodnota v CNT dosáhne MOD, příští takt hodin:
 - Vynuluje obsah CNT
 - Nastaví příznak přetečení TOF
 - Příp. vyvolá přerušení, pokud je povoleno
 - Důsledky
 - Pro MOD == 1 se provedou 2 kroky CNT (minimum)
 - Pro MOD == 0 se provede 65535 kroků do přetečení
 - Hodnota do MOD je tedy (pocet_tiku – 1)
 - Pro 50000 tiků nutno nastavit 49999



Náhrada přerušením

- Testování přetečení blokuje vykonávání programu, dokud nenastane
 - Tzv. **polling** (= programové testování stavu)
- Lépe využít přerušení
 - Asynchronní událost
 - Vzniká typicky při události v HW
 - Pokud je povoleno zpracování zdroje přerušení ("povolovací bit"):
 - Pozastaveno vykonávání programu
 - Vyvolána přerušovací funkce
 - Dána vektorem obsluhy (adresa funkce uložena na určené místo paměti)
 - Uloženy pracovní registry
 - Vykonán kód přerušovací rutiny
 - Obnoveny pracovní registry a řízení vráceno kódu předtím vykonávanému

- **PE defaultně negeneruje rámec kódu pro přerušení = nutno povolit ve vlastnostech projektu**



The screenshot shows the 'Processor Expert Project Options - cv5p' dialog. The 'Processor Expert' tab is selected. The 'Delete unused events and ISRs' option is highlighted with a red box. The 'Generate ISRs' option is also highlighted with a red box.

Name	Value
Main & Events directory	Sources
Generated code directory	Generated_Code
Static code directory	Generated_Code
Documentation directory	Documentation
Project settings directory	Project_Settings
Set periph. init component name as periphe...	Yes
Main module update	Smart update (recommended)
Event module(s) update	Smart update (recommended)
Update of other user modules	Smart update (recommended)
Generate ISRs	Yes
Delete unused events and ISRs	Yes
Delete unused previously generated files	No
Freeze code generation	No
Generate code before build automatically	Yes
Save project before code generation	Yes
Create code generation log	No
Code generation reference number	7

- V komponentě Timer pak:
 - Povolit "přerušení při přetečení"
 - Zvolit jméno obsluhy přerušení
 - ISR = Interrupt Service Routine

▶	External clock source	Disabled	
▲	❗ Interrupts		
▲	❗ Overflow Interrupt		
	Interrupt	Vtpm1ovf	Vtpm1ovf
	Overflow Interrupt	Enabled	
	❗ ISR name	pretecení	No identifier defined !
▲	Initialization		
	Call Init method	yes	
	Enable module	yes	

Kód funkce přerušení

- PE připraví tělo funkcí v souboru **Events.c**
 - V **PE_Types.h** je **#define ISR(x) __interrupt void x(void)**
- V obsluze přerušení nutno (viz. datasheet)
 - Přečíst status-registr
 - "shodit" příznak přetečení časovače
- Budeme blikat další LEDkou

```
...
/* write your code here */
/* For example: for(;;) { } */
{
    word w;

    while(1)
    {
        LED1 = !LED1;

        for(w = 0; w < 50000; w++)
            ;
    }
}
...
```

```
/* User includes (#include ...) */
#include "hw_cfg.h"

#ifdef ISR_IN_NONBANKED
#pragma CODE_SEG __NEAR_SEG NON_BANKED
#endif
/*
** =====
**      Interrupt handler : pretečení
** =====
*/
ISR(pretečení)
{
    (void)TPM1SC;      // použít trik k vynucenímu čtení
    TPM1SC_TOF = 0;

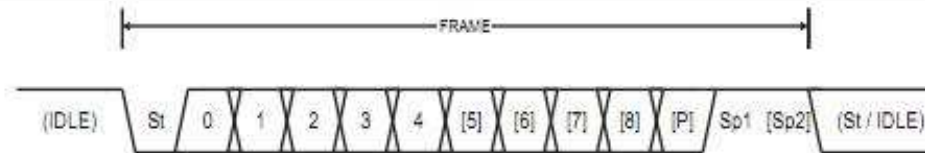
    LED2 = !LED2;
}
```

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
- 4. Sériový port**
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Sériový port – UART/SCI

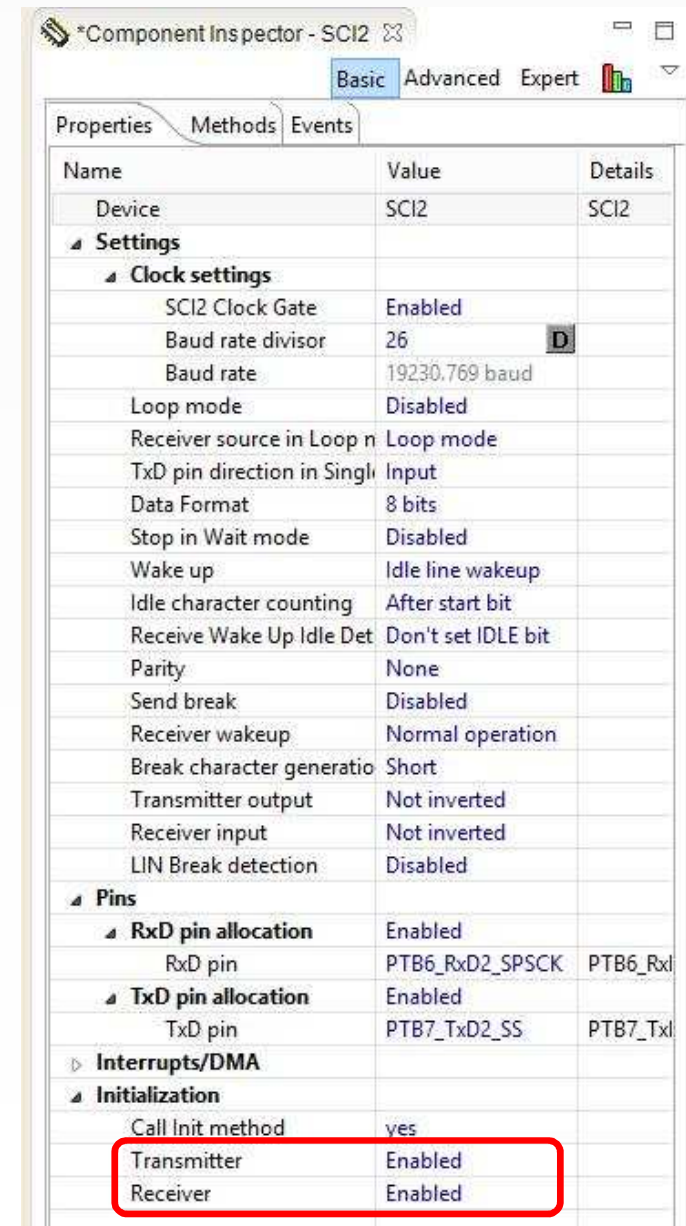
- Sériová komunikace vystačí s 2 signály (a GND)
 - **RxD** – receive data – příjem dat na zařízení
 - **TxD** – transmit data – vysílání dat ze zařízení
 - Typicky 8 datových bitů (může být 5-9b)
 - Přenos začíná start-bit, ukončuje stop-bit, příp. k datům se přidává parita



- Běžné komunikační rychlosti vycházejí z dob modemů
 - 75, 110, 300, 1200, 2400, 4800, 9600, **19200**, 38400, 57600, 115200 bit/s
 - Vytváří je blok sériového portu dělením z vnitřních hodin (sběrnice)
 - Max. odchylka rychlosti je 5%
 - Chyba nesmí být větší než ½ bitu (typicky celkem 10 bitů = 8b datových + start + stop)
- U procesorů Freescale se blok sériové komunikace nazývá **SCI** (Serial Communications Interface)
 - LL08 má 2 SCI bloky
 - SCI1 připojen na PTC0 (RxD) a PTC1 (TxD), fyzicky header na vývojové desce
 - **SCI2** připojen na PTB6 (RxD) a PTB7 (TxD), propojen na "komunikační" desku

Nastavení a registry SCIx

- Processor Expert – opět jen "init"
 - Povolit hodiny (SCI2 Clock Gate)
 - Komunikační rychlost = Baud rate divisor
 - Ověřit skutečnou Baud rate
 - !! Povolit Transmitter a Receiver
 - Defaultně jsou Disabled !!
- Registr **SCI2D** – datový
 - Čtení vyčítá přijatý bajt
 - Nutno ověřit, že nějaká data jsou přijata
 - Pokud nestihneme data odebrat, nastaví se chybový příznak
 - Pokud data jsou chybná (parita, rámeček, ...) nastaví se chybový příznak
 - Zápis spouští vysílací sekvenci
 - Nutno příp. počkat na odvysílání předchozího (!!)
 - Pro datové operace možno využít přerušení
 - Tx interrupt – Transmit completed
 - Rx Interrupt – Receive request
 - Error interrupt



Stavový registr SCIxS1

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
Reset	1	1	0	0	0	0	0	0

Figure 14-8. SCI Status Register 1 (SCIxS1)

- 7 – **TDRE** – Transmit Data Register Empty
 - Data přenesena z datového do vysílacího shift-registru
 - Možno zapsat další data k odeslání do SCIxD
 - Po přečtení log. 1 se bit nuluje automaticky dalším zápisem do datového SCIxD
- 6 – **TC** – Transmission Complete
 - Dokončeno odeslání posledního bitu na lince
 - Po přečtení log. 1 se bit nuluje automaticky dalším zápisem do SCIxD nebo vypnutím SCI
- 5 – **RDRF** – Receive Data Register Full
 - Korektně přijata data do přijímacího registru
 - Po přečtení log. 1 se nuluje automaticky čtení SCIxD
- 4 – **IDLE**
- 3 – **OR** – Receiver Overrun
 - Přijata další data a předchozí nejsou odebrána
 - Nuluje se vyčtením SCIxD, to obsahuje původní data, nová (= "špatná") jsou zahozena
- 2 – **NF** – Noise
 - Hodnoty bitů nemají správné úrovně v době jejich vzorkování
 - Nuluje se čtením SCIxD, chybou se nastavuje také RDRF, aby se vyvolal požadavek čtení (i když chybného obsahu)
- 1 – **FE** – Framing Error
 - Chyba rámce, chybí stop-bit v úrovni log. 1
 - Nastavuje se s **RDRF**, nuluje se čtení SCIxD
- 0 – **PF** – Parity Error
 - Nevyšla parita v přijímaných datech, nastavuje se společně s SCIxD

Funkce pro práci s SCI

- Pro vysílání a příjem znaků je výhodné si vytvořit funkce
 - `void uart_send(byte b)`
 - `byte uart_recv(void)`
- Pomocné makro pro zjištění přijatých dat
 - `UART_RECVREADY`
- Jednoduchý program pro otestování
 - Na straně PC využít např. Putty
 - Nastavit COM1 a rychlost 19200
- Pomocná funkce pro převod 4 bitů na hexadecimální cifru
 - `byte nibbleToHex(byte b)`

```
{
    byte b = 'a';
    word w;

    while(1)
    {
        uart_send(b);

        b++;
        if (b > 'z')
            b = 'a';

        if (UART_RECVREADY)
        {
            byte x = uart_recv();

            uart_send(':');
            uart_send(x);
            uart_send(':');
            uart_send(nibbleToHex(x >> 4));
            uart_send(nibbleToHex(x & 0x0f));
            uart_send(':');

            continue;
        }

        for(w = 0; w < 60000; w++)
            ;
    }
}
```

```
byte nibbleToHex(byte b)
{
    b &= 0x0f;
    return (byte)((b < 10) ? (b + '0') : (b + 'A' - 10));
}
```

Funkce pro příjem/odesílání

```
void uart_send(byte b)
{
    while(!SCI2S1_TDRE)
        ;

    SCI2D = b;
}

byte uart_recv(void)
{
    while(!SCI2S1_RDRF)
        ;

    return SCI2D;
}

#define UART_RECVREADY    (SCI2S1_RDRF)
```

Použití knihovny stdio

- Obsahuje řadu užitečných I/O funkcí
- Bez úprav možno použít např. `sprintf` (formátovaný výstup do řetězce – nutno připravit vhodný buffer)
- Ostatní funkce počítají s tím, že pro výstup se nakonec volá jedna společná funkce `TERMIO_PutChar`
 - Není v knihovně a je třeba ji doprogramovat
 - Pokud ji neposkytneme (nevytvoříme), generuje se chyba

Symbol `TERMIO_PutChar` in file `C:/Freescale/CW MCU v10.6 /MCU/lib/hc08c/lib\ansiis.lib` is undefined

- Pro vstup se podobně volá `TERMIO_GetChar`
- Jako parametry se v C/C++ používají **int**, ale zde funguje i **byte**
- Běžně se pak používají **putchar/getchar**, **printf**, **puts**
 - Pozor na `scanf`, bývá omezeně implementována a **NEDOPORUČUJE** se

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
4. Sériový port
- 5. Procvičení – sériový port, I/O, časovač, přerušení**
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Opakování probíraných částí

- Zatím máme předvedeno a je nutné vyzkoušet dohromady
 - I/O porty
 - LED-ky na portu PTC
 - tlačítka na PTA a PTC7
 - Čítač/časovač v režimu časovače
 - zdroj signálu – takt sběrnice, předdělič
 - modulo registr – zkrácení cyklu přetečení
 - příznakový bit TOF
 - Přerušení (zatím jen od časovače)
 - kód v souboru events.c
 - nutno "shodit" příznakový bit, nutná sekvence popsána v "Reference Manual"
 - Sériový port – SCI2 připojen na rozšiřující desku
 - nastavení rychlosti předděličkou, nutno vybrat některou ze standardních
 - příznak přijetí bajtu
 - příznak volného odesílací bufferu a příznak odvysílání

Důležité postupy dle RM

7 TDRE	Transmit Data Register Empty Flag. TDRE is set out of reset and when a transmit data value transfers from the transmit data buffer to the transmit shifter, leaving room for a new character in the buffer. To clear TDRE, read SC1xS1 with TDRE set and then write to the SCI data register (SC1xD). 0 Transmit data register (buffer) full. 1 Transmit data register (buffer) empty.
5 RDRF	Receive Data Register Full Flag. RDRF becomes set when a character transfers from the receive shifter into the receive data register (SC1xD). To clear RDRF, read SC1xS1 with RDRF set and then read the SCI data register (SC1xD). 0 Receive data register empty. 1 Receive data register full.

7 TOF	Timer overflow flag. This read/write flag is set when the TPM counter resets to 0x0000 after reaching the modulo value programmed in the TPM counter modulo registers. Clear TOF by reading the TPM status and control register when TOF is set and then writing a logic 0 to TOF. If another TPM overflow occurs before the clearing sequence is completed, the sequence is reset so TOF remains set after the clear sequence was completed for the earlier TOF. This is done so a TOF interrupt request cannot be lost during the clearing sequence for a previous TOF. Reset clears TOF. Writing a logic 1 to TOF has no effect. 0 TPM counter has not reached modulo value or overflow. 1 TPM counter has overflowed.
----------	---

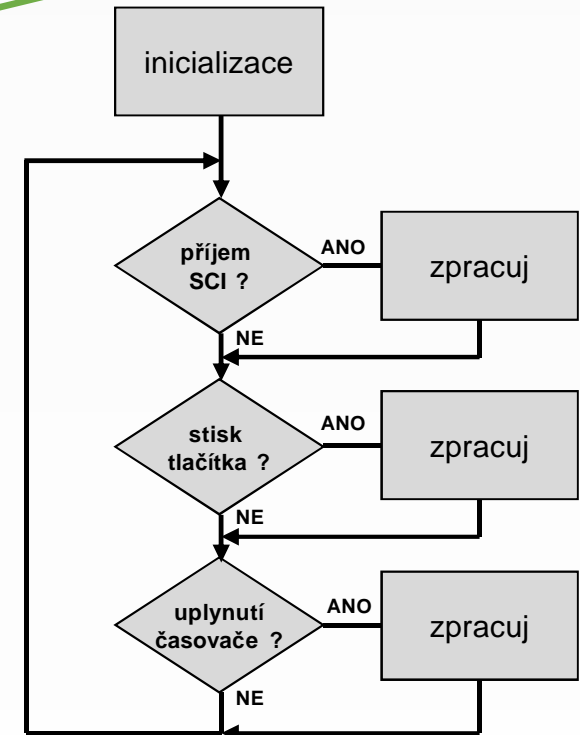
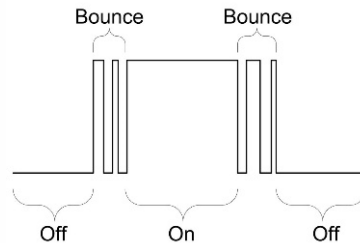
```
ISR(pretezeni)
{
    (void)TPM1SC;    // pouzit trik k vynucenímu ctení

    TPM1SC_TOF = 0;

    ...
    (
```

Náměty na spojení probraného

- Odesílání po SCI dle taktu časovače
 - zatím bez interruptu
- Odesílání stisku tlačítek
- Počítadlo na LED podle přijatých znaků
- Debouncing tlačítek využitím časovače/přerušení
 - ošetření možný zákmitů
 - nutno vyhodnotit několikrát za sebou stejný stav
 - nutno napsat funkci/funkce + statické proměnné apod.



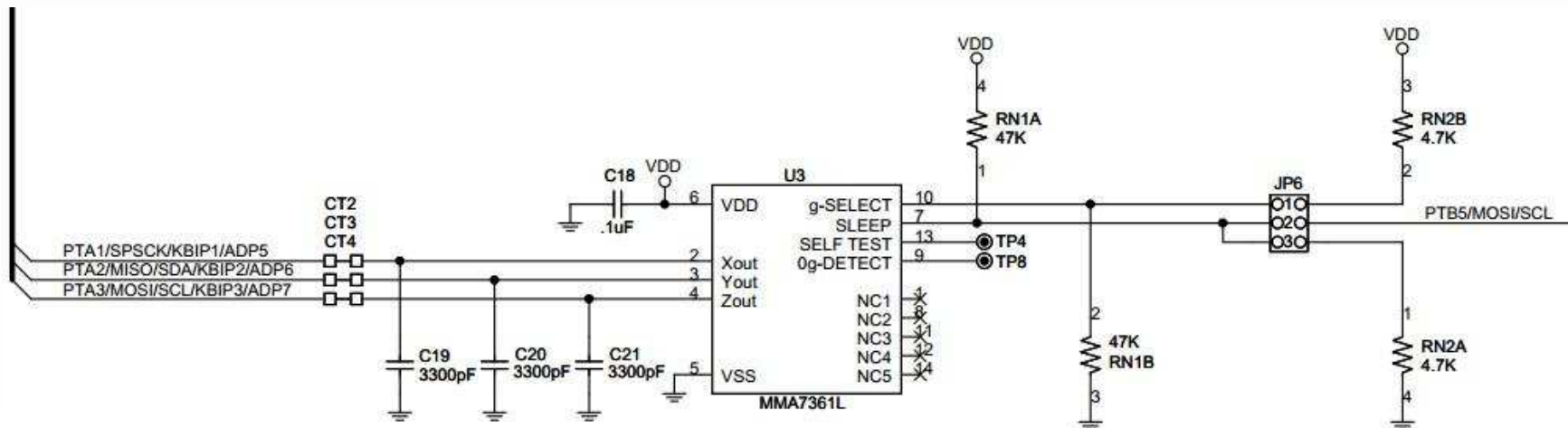
- Sériový port a interrupt – kruhové buffery pro příjem i vysílání

Plán cvičení

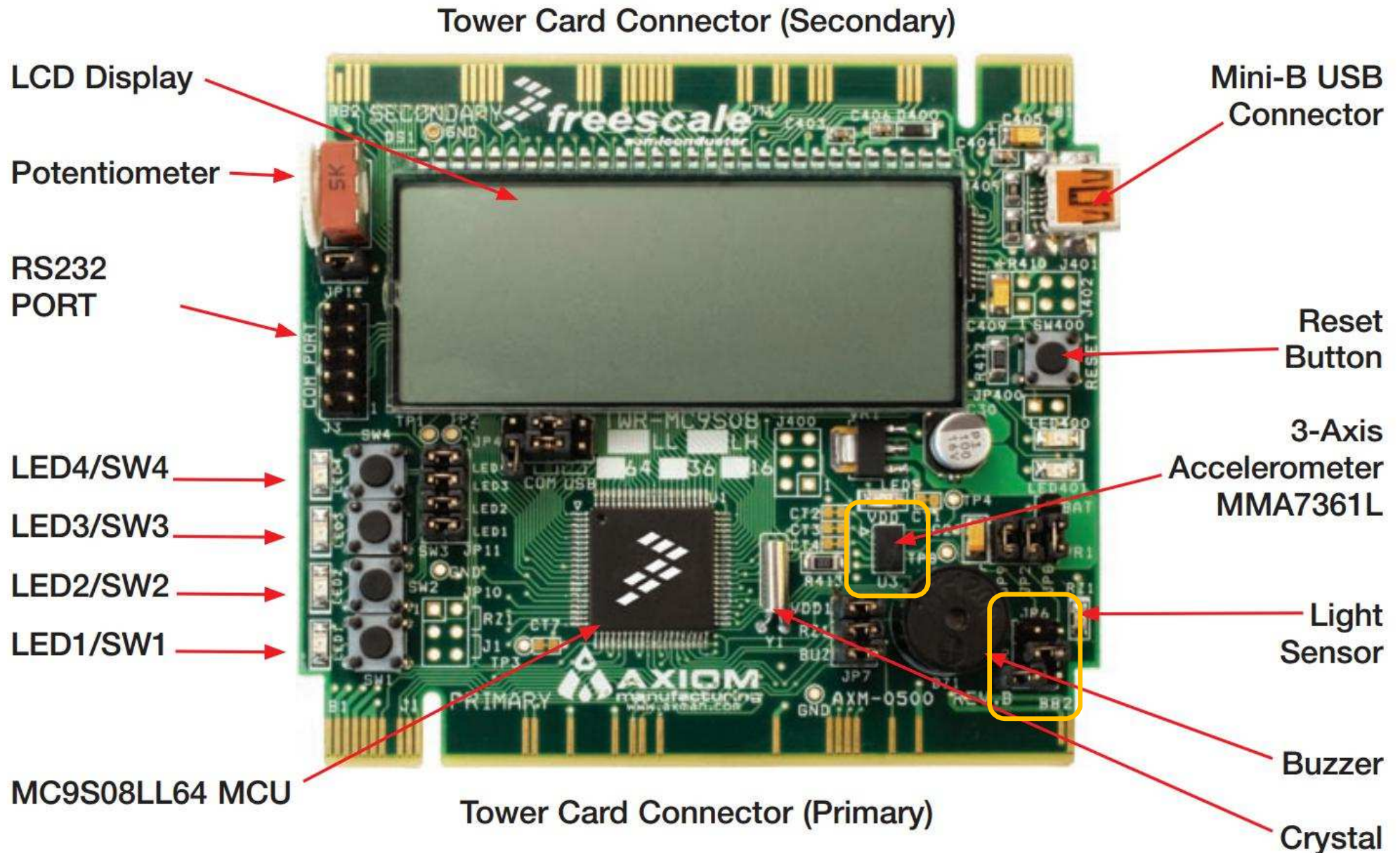
1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
- 6. A/D převodník + akcelerometr**
7. Další možnosti využití časovače a přerušení
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

3-osý akcelerometr – připojení

- MMA7361L - $\pm 1.5g$ nebo $\pm 6g$
 - Výstupní signál typicky 800 (nebo 206) mV/g
 - V klidu 2 osy 1.65V (napájení / 2), třetí 2.45V (+1g) nebo 0.85V (-1g)
 - Napájení 3.3V stejně jako procesor (max. 3,6V)
- Xout = PTA1 (ADP5)
- Yout = PTA2 (ADP6)
- Zout = PTA3 (ADP7)



Kit Freescale S08LL64 + Accelerometer MMA7361L



A/D převodník

- **RM - Chapter 11**

- Napěťové reference jsou pro 64-pinové pouzdro shodné s napájecím napětím (V_{DDA} a V_{SSA})
- Hodiny jsou automaticky po Resetu zapnuté (bit ADC v SCGC1)
- Výstupní hodnota 8-, 10- nebo 12-bitová, zarovnaná doprava
- Celkem 31 vstupů, výběr pomocí multiplexu
 - Některé pouze interní
 - Pro 64-vývodové pouzdro nejsou všechny dostupné
 - Externí signály sdílejí vývody s dalšími perifériemi (např. GPIO)
 - Vstup určen 5-bitovou kombinací **ADCH** v registru **ADCSC1**
 - Kombinace 11111 znamená "module disabled" (= žádné převody)
- Možnost různých režimů – komparátor apod. – nevyužíváme
- Další bity v **ADCSC1**
 - **COCO – Conversion Complete** – nastaven po skončení převodu
 - Nulován po zápisu do ADCSC1 nebo vyčtením dat z ADCRL
 - **AIEN** – Enable Interrupt

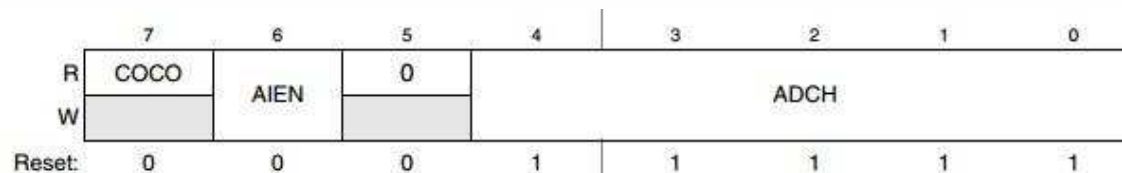


Figure 11-3. Status and Channel Control Register 1n (ADCSC1)

A/D převodník – II

- Převedená data v registrech ADCRH a ADCRL
 - Pro 8-bitový převod stačí vyčíst ADCRL
 - Automatiky nuluje COCO příznak
 - Pro 10- a 12-bitový převod nutno napřed číst ADCRH

Table 11-10. Data Result Register Description

Conversion Mode	DATA												Format
	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
12b single-ended	D	D	D	D	D	D	D	D	D	D	D	D	unsigned right justified
10b single-ended	0	0	D	D	D	D	D	D	D	D	D	D	unsigned right justified
8b single-ended	0	0	0	0	D	D	D	D	D	D	D	D	unsigned right justified

D: Data

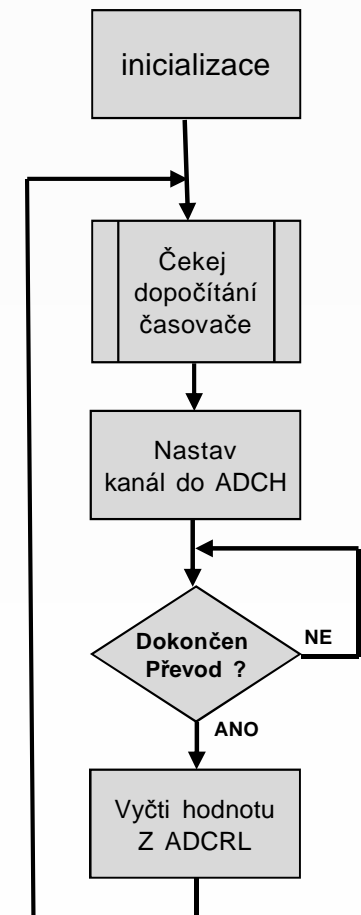
ADCH	Channel	Input	Pin Control
00000	AD0	ADP0	ADPC0
00001	AD1	Reserved	ADPC1
00010	AD2	Reserved	ADPC2
00011	AD3	Reserved	ADPC3
00100	AD4	PTA0/ADP4	ADPC4
00101	AD5	PTA1/ADP5	ADPC5
00110	AD6	PTA2/ADP6	ADPC6
00111	AD7	PTA3/ADP7	ADPC7
01000	AD8	PTA4/ADP8	ADPC8
01001	AD9	PTA5/ADP9	ADPC9
01010	AD10	PTA6/ADP10	ADPC10
01011	AD11	PTA7/ADP11	ADPC11
01100	AD12	ADP12	ADPC12
01101	AD13	Reserved	N/A
01110	AD14	Reserved	N/A
01111	AD15	Reserved	N/A

ADCH	Channel	Input	Pin Control
10000	AD16	Reserved	N/A
10001	AD17	Reserved	N/A
10010	AD18	Reserved	N/A
10011	AD19	VREF0	N/A
10100	AD20	Reserved	N/A
10101	AD21	Reserved	N/A
10110	AD22	Reserved	N/A
10111	AD23	VLCD	N/A
11000	AD24	VLL1	N/A
11001	AD25	Reserved	N/A
11010	AD26	Temperature Sensor ¹	N/A
11011	AD27	Internal Bandgap	N/A
11100	AD28	Reserved	N/A
11101	V _{REFH}	V _{REFH}	N/A
11110	V _{REFL}	V _{REFL}	N/A
11111	Module Disabled	None	N/A

Name	Value	Details
Device	ADC	ADC
Settings		
Clock settings		
Input clock select	BusClk	
ADC Clock Gate	Enabled	
Prescaler	1	
High-speed conversion	Disabled	
Asynchro clock output	Disabled	
Long sample time	no	
Long sample time length	20	
Frequency	8000 kHz	
ADC timing details		
Single conversion time	3.38 us; not supported	96.296 kHz
Continuous conversion	2.12 us; not supported	470.588 kHz
Conversion mode	Single conversion	
Result data format	8-bit right	
Low power mode	Disabled	
Conversion trigger	Software	
Hardware trigger select	TOD - match con...	
Compare Function		
Compare function	Disabled	
Compare type	less than	
Compare value	0	D
Internal bandgap buffer	Disabled	
Pins		
ADC Input Pins		
Input Pin0		
Pin	PTA1_KBIP1_SPS...	PTA1_KBIP1
Pin I/O control disabled	no	
Input Pin1		
Pin	PTA2_KBIP2_SDA...	PTA2_KBIP2
Pin I/O control disabled	no	
Input Pin2		
Pin	PTA3_KBIP3_SCL...	PTA3_KBIP3
Pin I/O control disabled	no	
Interrupts/DMA		
Initialization		
ADC type		
Initial channel select	ADC Disabled	
Call Init method	yes	

A/D převodník – vlastní kód

- Pro vyhodnocení polohy zařízení (tj. akcelerometru připájeného na PCB) stačí A/D převod v pravidelných intervalech
 - Zvolíme např. 50ms – použijeme časovač
 - Stačí polling režim (=testování TOF bitu)
 - Ve vlastnostech A/D převodníku necháme "Initial Channel Select" na Disabled
 - Před převodem zvolíme "kanál"
 - Vyčkáme dokončení převodu (bit COCO)
 - Vyzvedneme příslušná data z ADCRL (pro 8-bitový převod)
 - Nebo ADCRH a ADCRL (pro 10- a 12-bitový převod)
 - Po převedení a vyčtení se automaticky převodník převede do idle-módu
- Na výsledek v "nějaké" proměnné se zatím podíváme pomocí Debug
 - Ověřte změnu hodnoty otočením "toweru"



Přenos naměřených dat

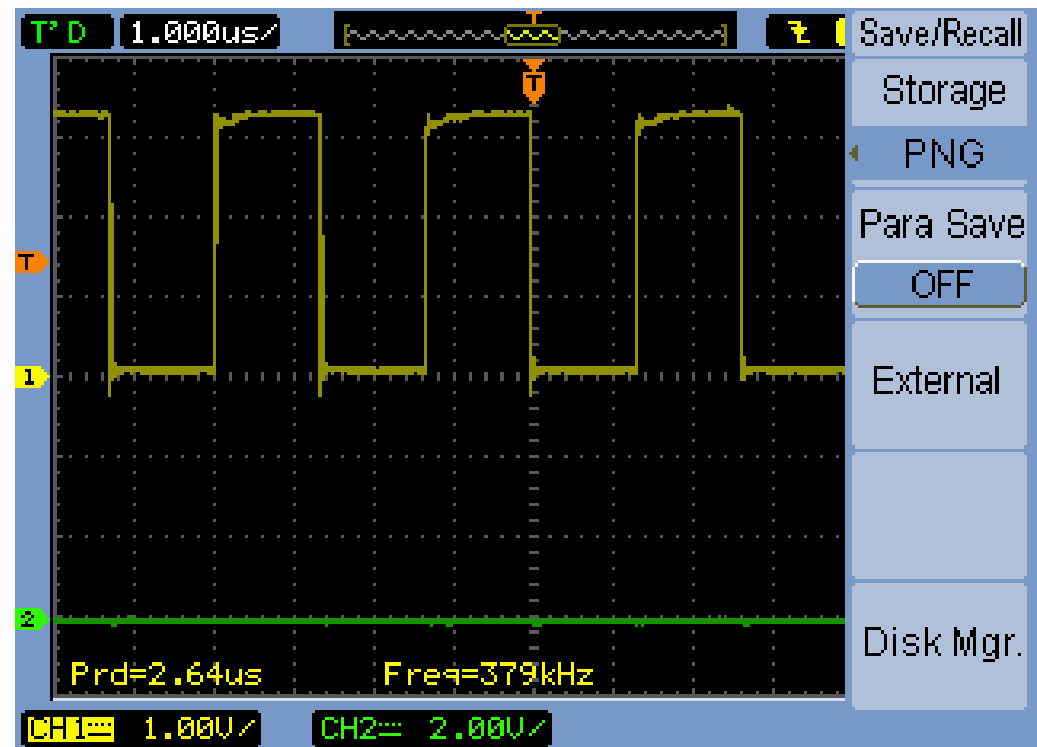
- Stejným způsobem možno měřit z ostatních kanálů
 - Přemístěte kód měření do funkce
byte read8ADC(byte channelCode)
 - Parametrem číslo kanálu (pro $X_{\text{out}} = 5$, $Y_{\text{out}} = 6$, $Z_{\text{out}} = 7$)
 - Návratová hodnota = výsledek z A/D převodu
- Výpis dat na terminál
 - Lépe v "čitelné" formě
 - Řádky ukončovat CR-LF, tj. `\r\n`
 - Použijme funkci `printf` a formátovaný výstup 3 celočíselných hodnot
 - Nutný **#include <stdio.h>**
 - **Error** – "Symbol `TERMIO_PutChar` in file `C:/Freescale ...\ansiis.lib` is undefined"
 - Knihovna **stdio** očekává, že aplikace bude mít funkci **TERMIO_PutChar**
 - Funkce má implementovat std. výstup
 - V našem případě sériový port SCI – viz. minulý příklad funkce `putchar`
 - Pro **stdin** funkce logicky vyžadována funkce **byte TERMIO_GetChar(void)**
 - **Warning** – "C1420 Result of function-call is ignored"
 - Problém u funkcí, které vrací hodnotu, ale náš kód ji nevyužívá
 - Buď vypnout v nastavení překladače, nebo naznačit, že výsledek nechceme
 - **(void)printf("...**
- Vizualizace "grafovým sw" – očekává řádky 3 hodnot oddělené čárkou

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
- 7. Další možnosti využití časovače a přerušení**
8. Kruhový buffer, časovač v režimu PWM
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Vizualizace časování pomocí LED

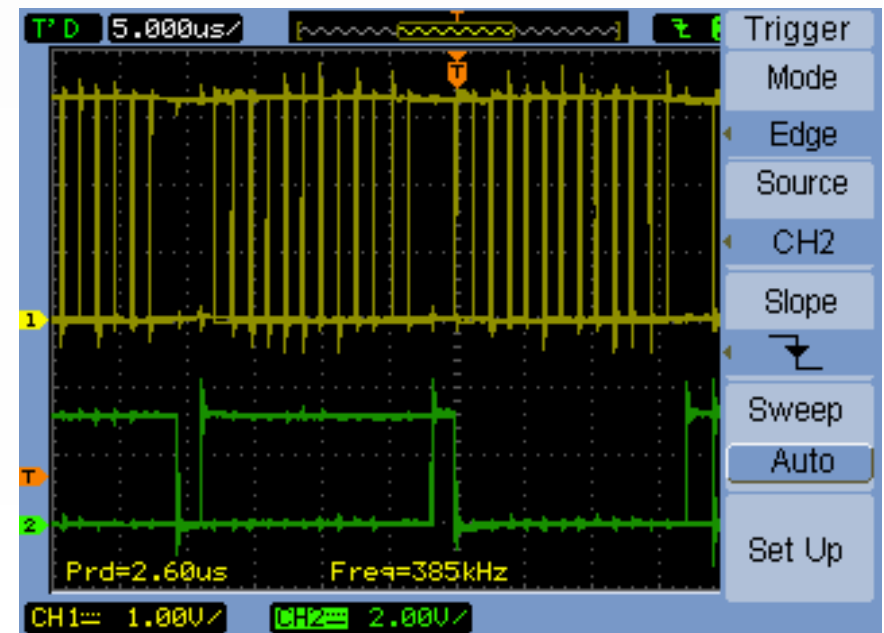
- Využití komponent a znalostí z minulých projektů
 - Jádro CPU – bus-clock 8MHz generovaná z externích 8MHz
 - PTA a PTC porty – připojené LEDs a BUTTONs
- Zatím žádné využití přerušení
- V hlavní smyčce (while) negovat LED1
 - Generuje se obdélníkový průběh
 - Možno sledovat osciloskopem



Vizualizace interruptu pomocí LED

- Přidat blok TPM1
 - Zdroj hodin BusClock, předdělič 8x (= čítají se mikrosekundy)
 - Nastavit modulo registr na přetékaní každých 20us
- Ve vlastnostech TPM1 povolíme "Overflow Interrupt" a nastavíme jméno funkce
 - Nezapomenout povolit generování kódu přerušení
 - Vlastnosti projektu - **Project – Properties – ProcessorExpert – Generate ISRs**
- Soubor Events.c – vygenerovaná nová ISR funkce
 - V rutině (funkci) ISR budeme negovat stav LED2
 - Žádné čekání na přetečení (to se dělá "samo")
 - Nezapomenout shodit TOF příznak !!
- V main() zůstává negace LED1
- Obdélníky LED1 každých 100us "chybí"
 - Vykonává se kód přerušení

• **CHYBY v průběhu LED2 !!**



Problém ne-atomických akcí

- Obyčejná negace bitu není na S08 jedna instrukce
 - Překládá se jako tzv. sekvence Read-Modify-Write
 - Skládá se interně ze 3 instrukcí
- Přerušení může nastat "mezi" jednotlivými částmi
 - Zde má celá hlavní smyčka jen 4 instrukce = velmi pravděpodobný vznik

Main – while(1)	Acc	Přerušení	Acc	PTCD
Načti z PTCD	0000 0000			0000 0 0 00
XOR 0x40	0000 0100			0000 0 0 00
		Uložit registry		0000 0 0 00
		Nuluj TOF		0000 0 0 00
		Načti z PTCD	0000 0000	0000 0 0 00
		XOR 0x80	0000 1000	0000 0 0 00
		Ulož do PTCD	0000 1000	0000 1 0 00
		Vrať zpět reg.		0000 1 0 00
		Konec INT		0000 1 0 00
Ulož do PTCD	0000 0100			0000 0 1 00
Skok while				0000 0 1 00
Načti z PTCD	0000 0100			0000 0 1 00
XOR 0x40	0000 0000			0000 0 1 00
Ulož do PTCD	0000 0000			0000 0 0 00
Skok while				0000 0 0 00

Řešení problému atomicity operací

- Použití atomických operací
 - Na S08 existuje instrukce BitSet a BitClear
 - Lze použít na adresy v rozsahu 0-0xff (= běžné registry periférií)
 - Pro náš případ tedy stačí ...
 - Střída není 50%, protože jeden stav trvá jednu instrukci, druhý navíc dobu skoku (!)
 - Možno využít podmínku, bit LED1 se v přerušení nemění
 - Jiné architektury mají speciální GPIO registry – "ToggleData"
 - Zápisem log 1 na příslušný bit se neguje daný výstup
 - Existuje řada dalších HW triků
 - Např. BitBanding pro ARM Cortex-M = každý bit je mapován na svojí adresu
- Eliminace vícevláknového přístupu k jednomu zdroji – např.:
 - Bity upravovat jen v přerušeních se stejnou prioritou
 - Změny bitů dělat pouze v jednom přerušení
 - Např. 10us od časovače
 - Zavést frontu požadavků, kam různé části aplikace zasílají požadavky a v intu se provedou všechny "hromadně" – nejpozději do 10us
 - ...

Atomické operace - příklad

- V přerušení zůstává $LED2 = !LED2$
- Použít atomické operace

```
while(1)
{
    LED1 = 1;
    LED1 = 0;
}
```

- Co ten Jitter na CH2 ? ☺
 - Kap. 5.5 Interrupts v RM

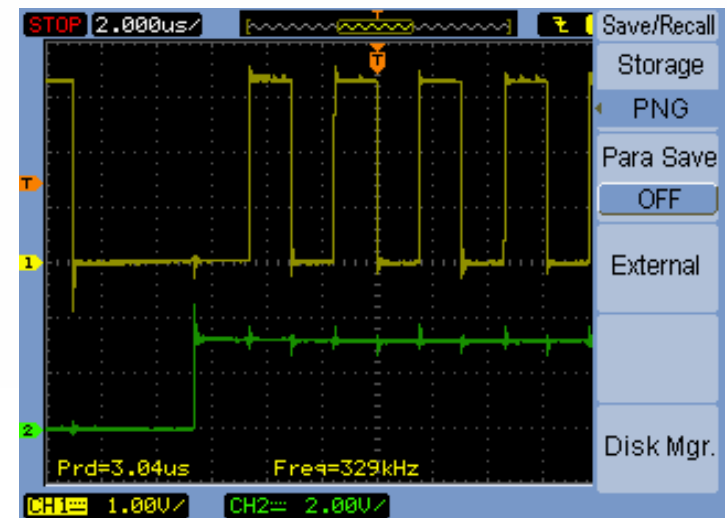
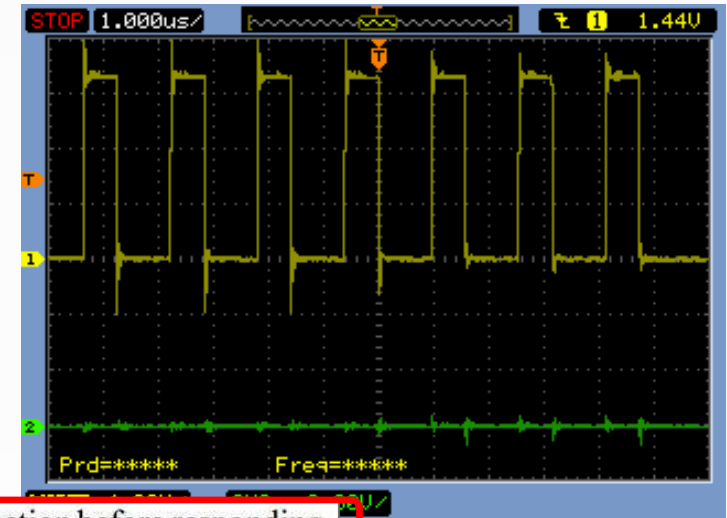
When the CPU receives a qualified interrupt request, it completes the current instruction before responding to the interrupt. The interrupt sequence obeys the same cycle-by-cycle sequence as the SWI instruction and consists of:

- Synchronizovat od CH1
- Není 50% střída

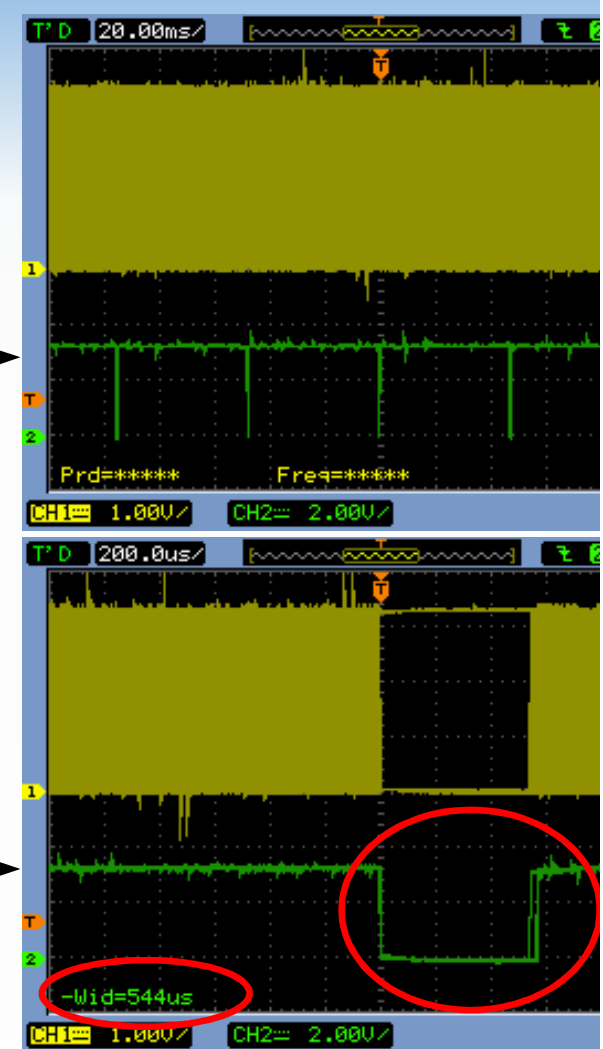
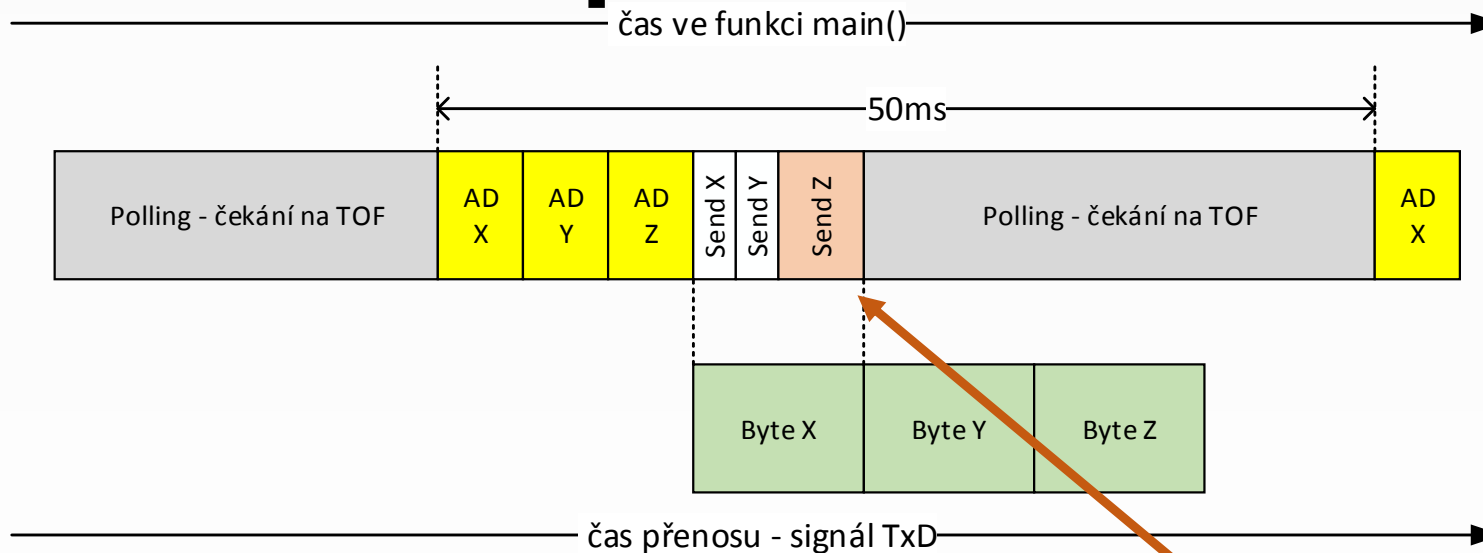
- Použití podmínky
 - Střída 50%, pomalejší
 - Lze použít i ternární operátor

```
while(1)
{
    LED1 = LED1 ? 0 : 1;
}
```

```
while(1)
{
    if (LED1)
        LED1 = 0;
    else
        LED1 = 1;
}
```

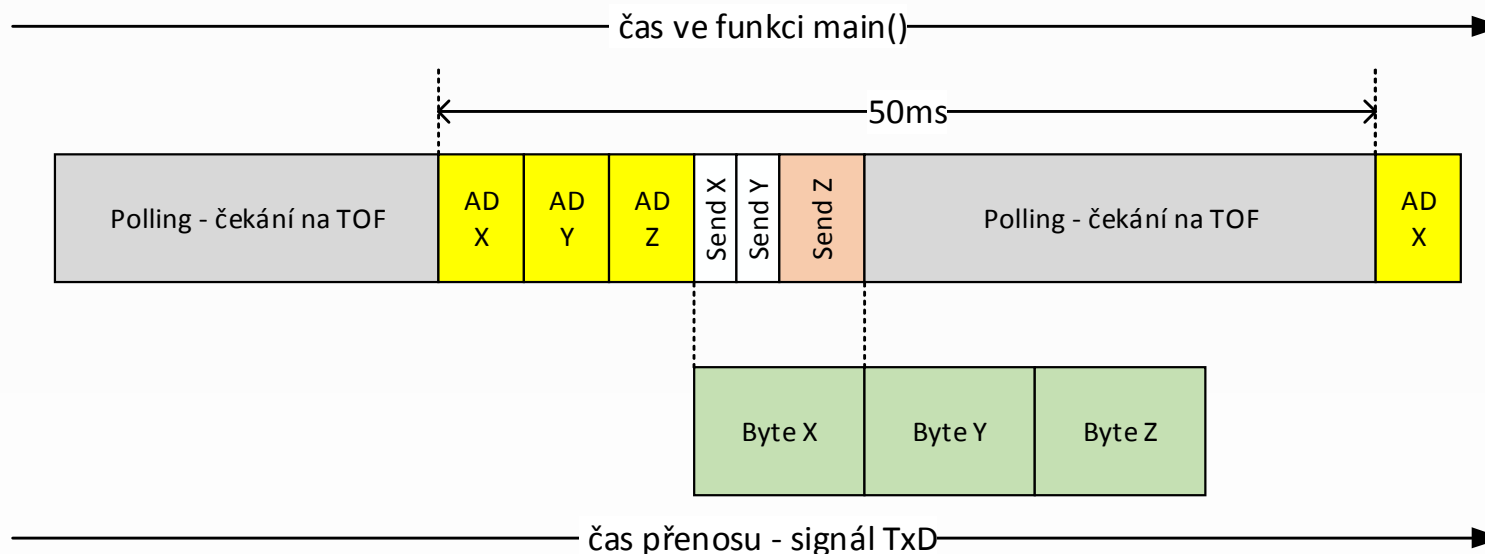


Časování pollingem = bez přerušení



- Veškerý čas procesor stráví ve funkci main
 - Po detekci přetečení se 3x změří
 - Při odesílání se čeká pouze jednou na "doodeslání" – proč ?
 - Pro ověření po vynulování TOF rozsvítit LED2 a po 3. vysílání LED2 zhasnout – můžeme sledovat na osciloskopu šířku pulsu
- Kód napište s využitím dřívějších příkladů
 - A/D převod ve funkci read8ADC()
 - Přetékání časovače nastavit na 50ms
 - SCI2 nastavit na rychlost 19200 (= jak dlouho trvá jeden znak ??)
 - Hodnoty odesílat pomocí putchar (3x) – aplikace na PC je umí také

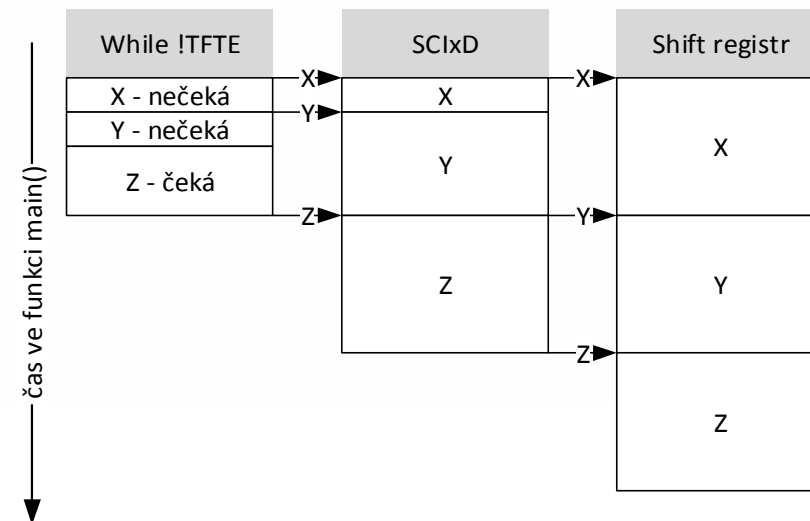
Časování pollingem – II



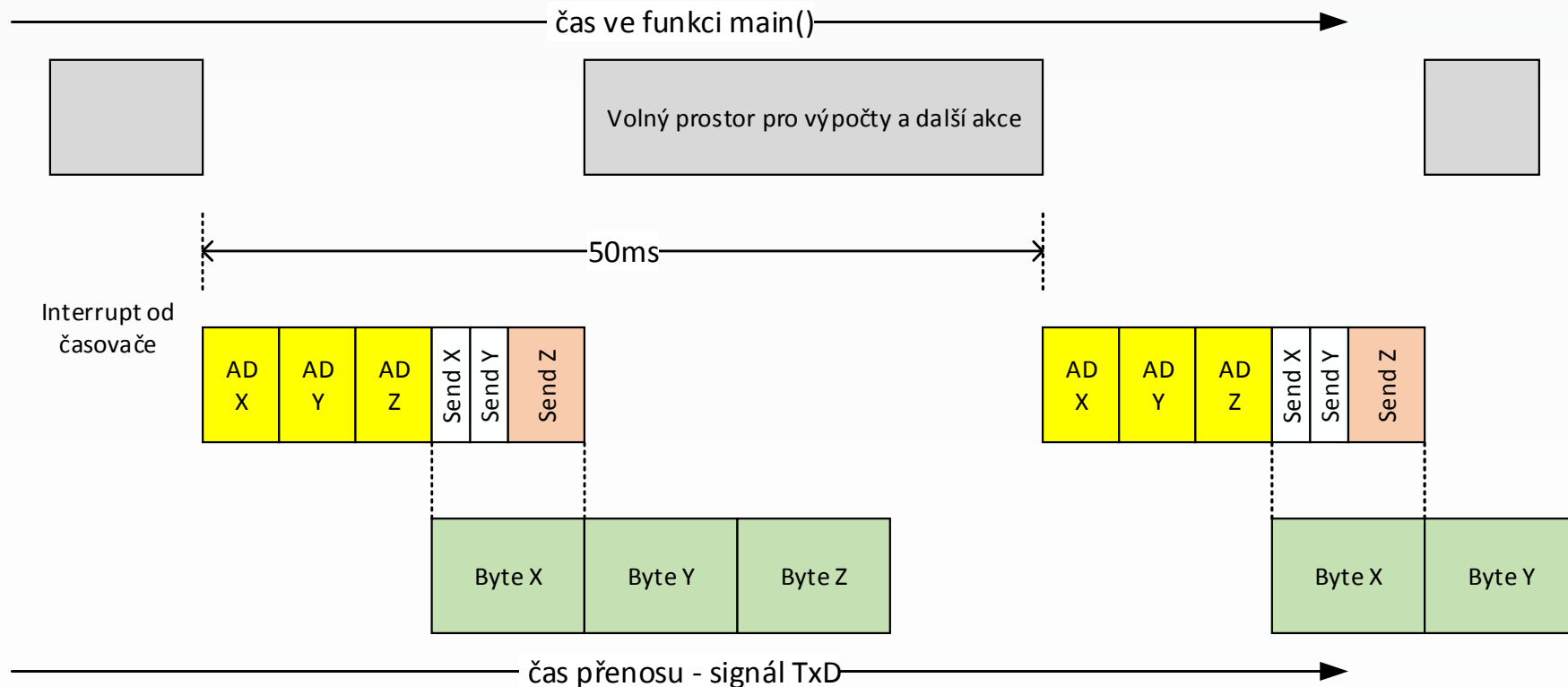
- Veškerý čas procesor stráví ve funkci main

- Po detekci přetečení se 3x změří
- Při odesílání se čeká pouze jednou na "doodeslání" – proč ?

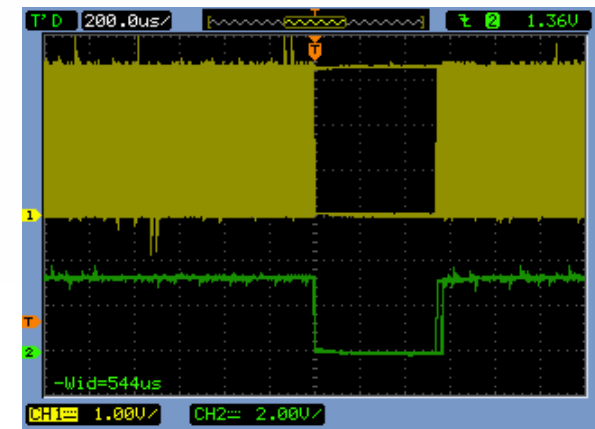
- Ve while se čeká na příznak, že SCIdx uvolněn = po přesunu do shift registru



Časování přerušením

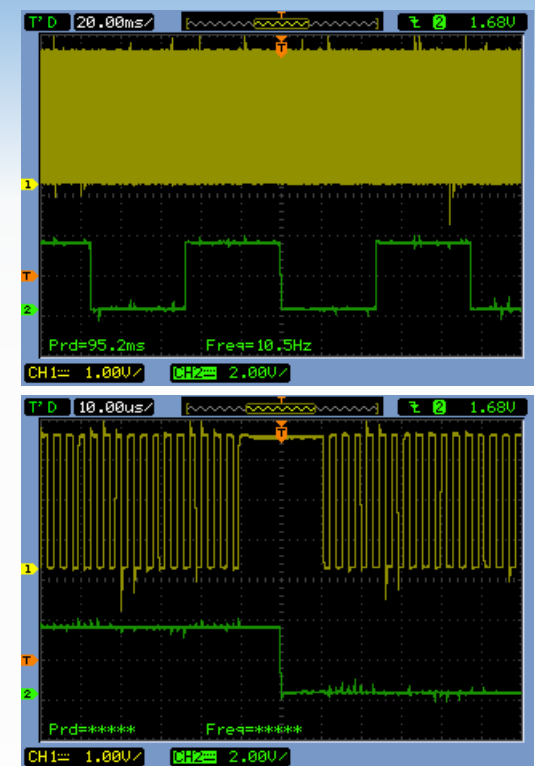


- Ve vlastnostech TPM1 povolíme "Overflow Interrupt" a nastavíme jméno funkce
 - Nezapomenout povolit v *Projekt – Properties – ProcessorExpert – Generate ISRs*
- Soubor Events.c – nová ISR funkce
 - Přesuneme komplet rutinu měření (tj. read8ADC) a odesílání do ISR
 - Včetně bliknutí LED2 pro možnost měření
 - Žádné čekání na přetečení (to se dělá "samo")
 - Nezapomenout shodit TOF příznak !!
 - Včetně odeslání 3 hodnot (lze využít putchar)
- V main() tedy pouze nekonečná smyčka "blikající" LED1



Využití přerušení od AD převodníku

- Každým přerušením od TPM1 se spustí jeden převod
 - Inkrementace čísla AD kanálu v hodnotách 5, 6, 7
 - Použitelná statická nebo globální proměnná
- Povolení přerušování od ADC
 - V obslužné funkci (je v Events.c) se odešle změřená hodnota po SCI2
 - Příznak COCO netřeba nulovat, "shodí" se čtením dat

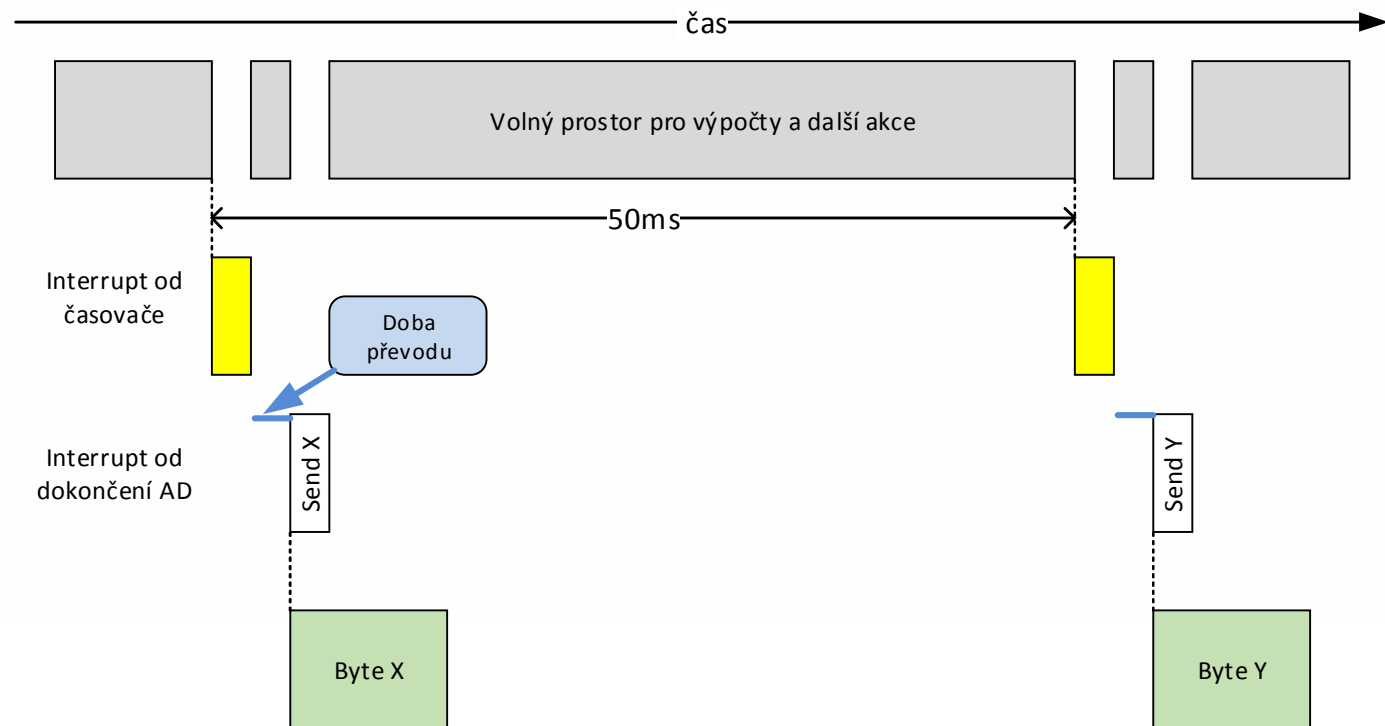


```
ISR(TimerOver)
{
    static byte kanal = 5;

    (void)TPM1SC;
    TPM1SC_TOF = 0;
    ADCSC1_ADCH = kanal;

    kanal++;
    if (kanal > 7)
        kanal = 5;
}
```

```
ISR(ADComplete)
{
    LED2 = !LED2;
    putchar(ADCRL);
}
```



```
volatile byte gKanal;
```

```
ISR(TimerOver)
{
    (void)TPM1SC;
    TPM1SC_TOF = 0;

    LED2 = 0;    // on

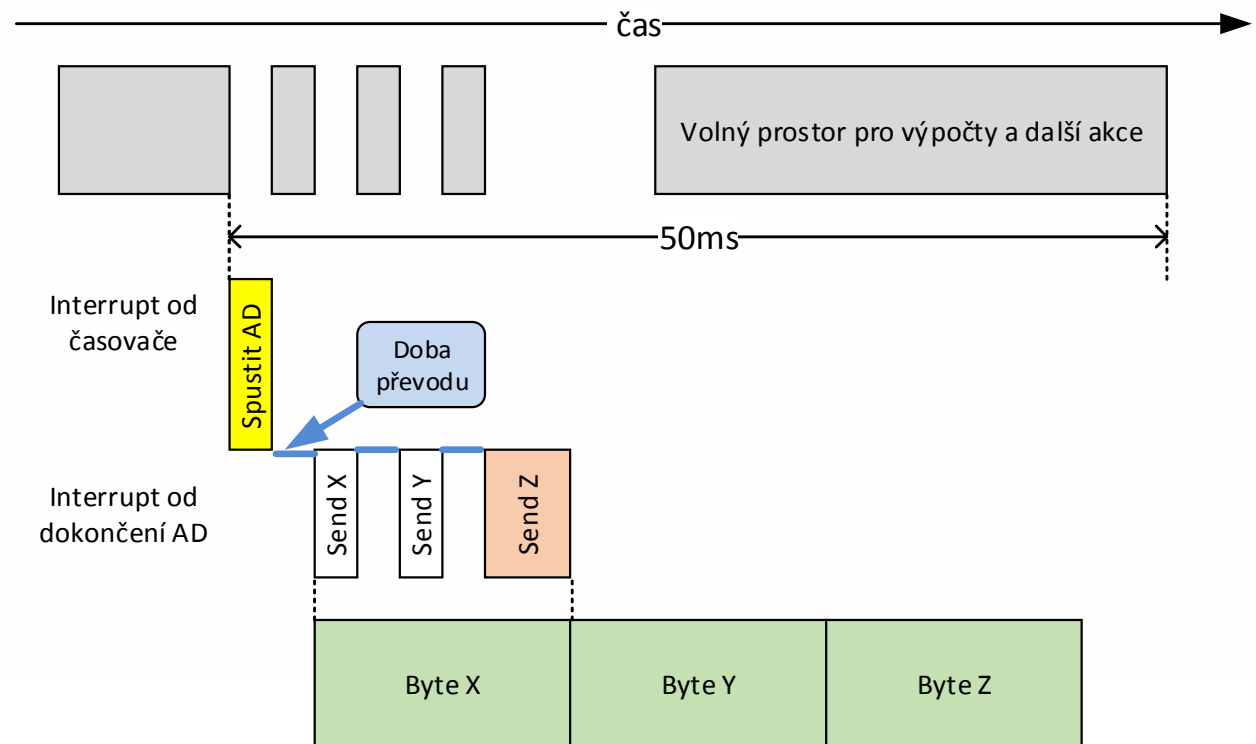
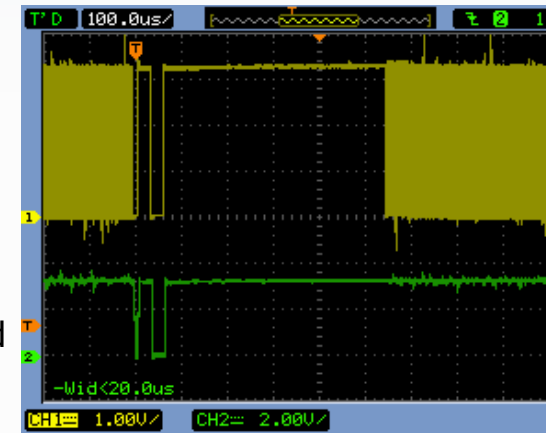
    gKanal = 5;
    ADCSC1_ADCH = gKanal;
}
```

```
ISR(ADComplete)
{
    LED2 = !LED2;
    switch(gKanal)
    {
        case 5:    // AD5 - axis X
            putchar(ADCRL);
            gKanal++;
            break;
        case 6:    // AD6 - axis Y
            putchar(ADCRL);
            gKanal++;
            break;
        case 7:    // AD5 - axis Z
            putchar(ADCRL);
            gKanal = 0x1f; // stop cnv.
            break;
    }

    if (gKanal > 7)    // konec mereni ?
        LED2 = 1;    // off
    else
        ADCSC1_ADCH = gKanal; // start nxt
}
```

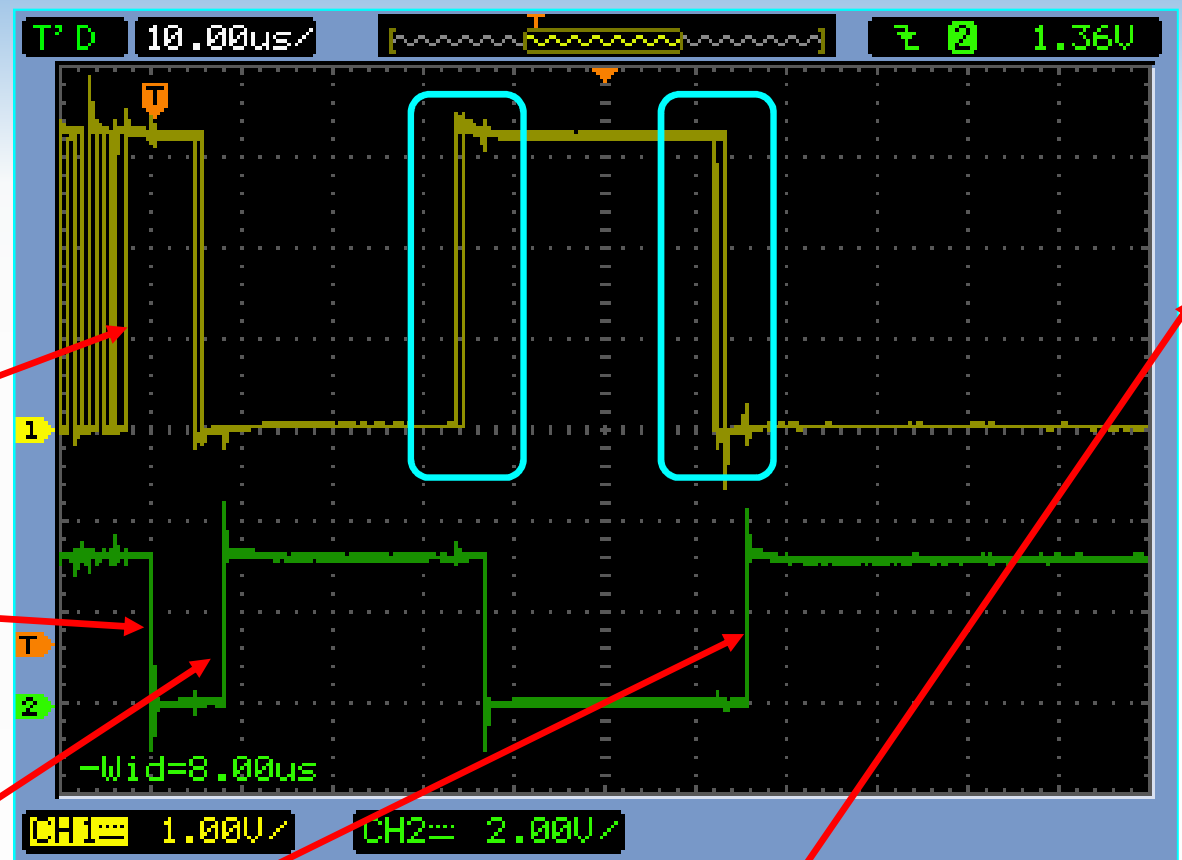
Přerušeni od AD převodníku – II

- Převod AD každé přerušeni od TPM není výhodné
 - Zpomalení dodávání dat
 - Data nejsou sbírána v "jeden společný čas"
- Každým přerušením od TPM1 se spustí první převod
 - Kanál 5, hodnotu uložíme do globální proměnné
- Přerušeni od ADC
 - Odešle se změřená hodnota
 - Inkrementuje se číslo kanálu
 - Pokud je "platné" (není větší než 7), spustí se další převod



Přerušení od AD převodníku - III

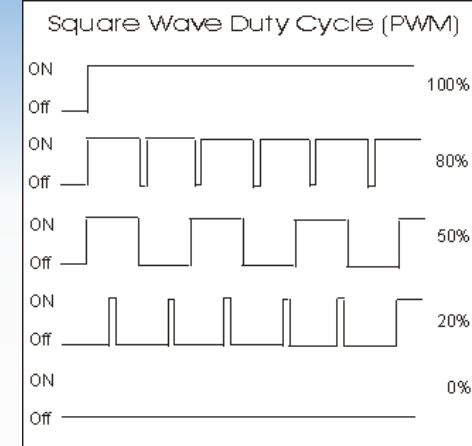
- Přerušení časovače
 - Začátek
 - LED2 = 0;
 - Pak spuštěn AD převod
- Přerušení od AD
 - Příchod do interruptu
 - Pak odeslán výsledek pomocí putchar
 - Pak spuštěn další převod
 - Dokončen 3. převod
 - Další už nebude
- Dokončení odesílání
 - Třetí znak se začal odesílat = dokončen "jeho" putchar
 - Cca 500us po spuštění akcí časovačem



Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
- 8. Dokončení přerušení, časovač v režimu PWM**
9. LCD displej
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Generování PWM



- PWM = pulsně šířková modulace
 - Nejjednodušší číslicové generování analogové hodnoty
- Na Freescale S08 mají čítače rozšiřující funkce ve formě "kanálů"
 - Počet kanálů je různý u různých typů/řad
 - Pro LL64 jsou 2 kanály pro každý TPMx
 - Každý z **n** kanálů
 - Má přiřazen I/O pin – viz. popis vývodů
 - Má 16-bitový datový registr TPMxCnV (s polovinami TPMxCnVL a TPMxCnVH)
 - Události mohou vyvolat přerušení
 - Všechny signály lze nastavit na aktivní v 0/1, hrany podobně na Hi-Lo nebo Lo-Hi
 - Základní funkce nastavitelné pomocí konfiguračních bitů/registrů
 - Input Capture
 - Při HW detekci sestupné/vzestupné hrany se uloží stav registru čítače do záchytného registru
 - Output Compare
 - Při shodě stavu registru čítače a registru se vygeneruje sestupná/vzestupná hrana nebo změna
 - **PWM**
 - Modulo-registr řídí periodu PWM signálu
 - **Hranově zarovnané (edge-aligned)**
 - Počáteční hrana PWM nastává při nulování čítače
 - Koncová hrana dána shodou stavu čítače a registru
 - Středově zarovnané (center-aligned)
 - Směr čítání se mění nahoru/dolů
 - Skutečná perioda je 2x – viz. dokumentace

TPMxMODH:TPMxMODL = 0x0008
TPMxCnVH:TPMxCnVL = 0x0005

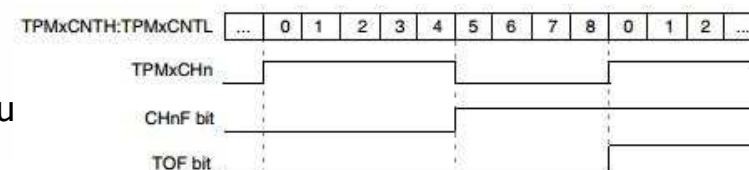


Figure 17-3. High-true pulse of an edge-aligned PWM

Table 17-6. Mode, Edge, and Level Selection

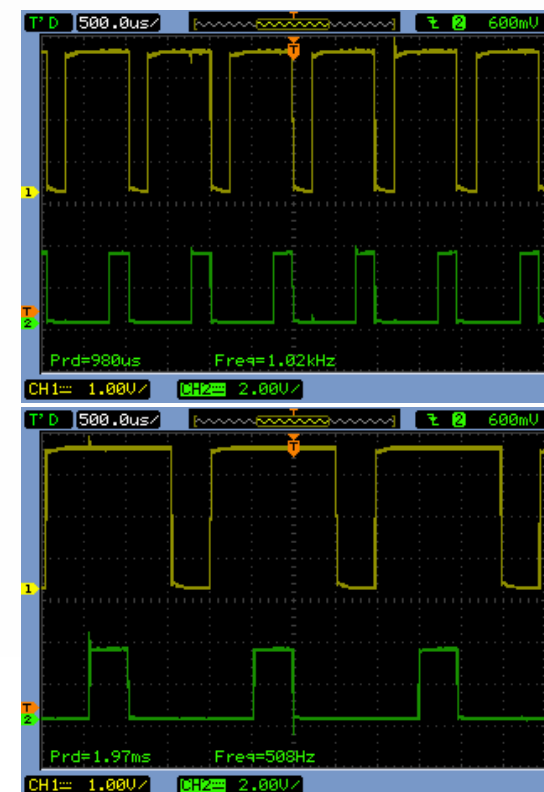
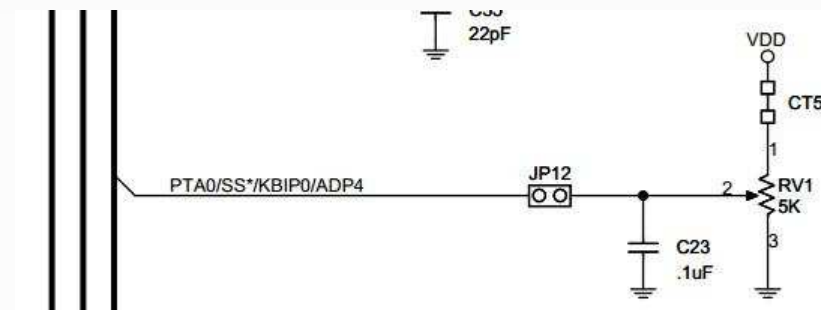
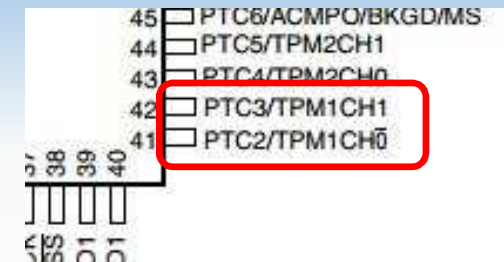
CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	XX	00	Pin is not controlled by TPM. It is reverted to general purpose I/O or other peripheral control	
0	00	01	Input capture	Capture on rising edge only
		10		Capture on falling edge only
		11		Capture on rising or falling edge
	01	00	Output compare	Software compare only
		01		Toggle output on channel match
		10		Clear output on channel match
		11		Set output on channel match
	1X	10	Edge-aligned PWM	High-true pulses (clear output on channel match)
		X1		Low-true pulses (set output on channel match)
1	XX	10	Center-aligned PWM	High-true pulses (clear output on channel match when TPM counter is counting up)
		X1		Low-true pulses (set output on channel match when TPM counter is counting up)

Přehled režimů čítačů

Period	1.024 ms
Aligned	Edge
▲ Channels	2
▲ Channel	
Capture/compare device	TPM10
▲ Settings	
▲ Mode	PWM
PWM output action	Set output on compare
Channel compare value	128
▲ Pin	Used
Channel pin	PTC2_TPM1CH0
Pull resistor	autoselected pull
▲ Interrupt	
▲ Channel Interrupt	
Channel Interrupt	Disabled
ISR name	
▲ Channel	
Capture/compare device	TPM11
▲ Settings	
▲ Mode	PWM
PWM output action	Set output on compare
Channel compare value	128
▲ Pin	Used
Channel pin	PTC3_TPM1CH1
Pull resistor	autoselected pull
▲ Interrupt	
▲ Channel Interrupt	
Channel Interrupt	Disabled
ISR name	

Generování PWM podle ADC

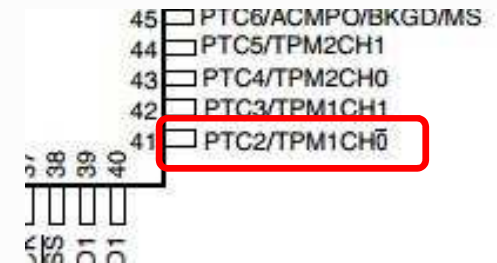
- **Cíl – měnit střidu PWM na LED pomocí potenciometru**
- **Zapojení HW:**
 - Na desce LED1 odpovídá PTC2 a sdílí tedy CH0 od TPM1
 - Na desce LED2 odpovídá PTC3 a sdílí tedy CH1 od TPM1
 - Nutno použít TPM1
 - Potenciometr připojen na ADC4
- **Nakonfigurování TPM1**
 - Nastavit modulo registr na 255 (odpovídá max. hodnotě z ADC)
 - Nastavit předděličku tak, aby generovaná perioda byla cca 1kHz (=1ms)
 - Přidat 2 kanály (vznikne CH0 a CH1) a zapnout jejich I/O pin
 - Nastavit value-registry (položka "Channel compare") na "polovinu" = střida PWM 50%
- **Zrušit v PTC bit 2 a 3 (=disable), aby jej mohl využívat CH0 a CH1**
 - Pozor, aby v kopírovaných částech kódu nezůstala práce s LED1 nebo LED2
- **Hlavní smyčka**
 - V pravidelných intervalech měřit pomocí ADC hodnotu
 - Hodnotu předávat do TPM1C0V a TPM1C1V (lze přistoupit 16-bitově)
 - Do kanálu CH1 dávat doplněk do 255 = výstup bude v protifázi
- **Vyzkoušet rozdíl "Edge-" a "Center-Aligned"**



Generování PWM podle ADC - II

- **Cíl – podle náklonu desky měnit frekvenci PWM na pinu připojeném k "pípáku" (buzzer)**

- Na desce připojen k PTC2 = TPM1 s CH0
- Podle ADC (třeba X osa ?) měnit frekvenci

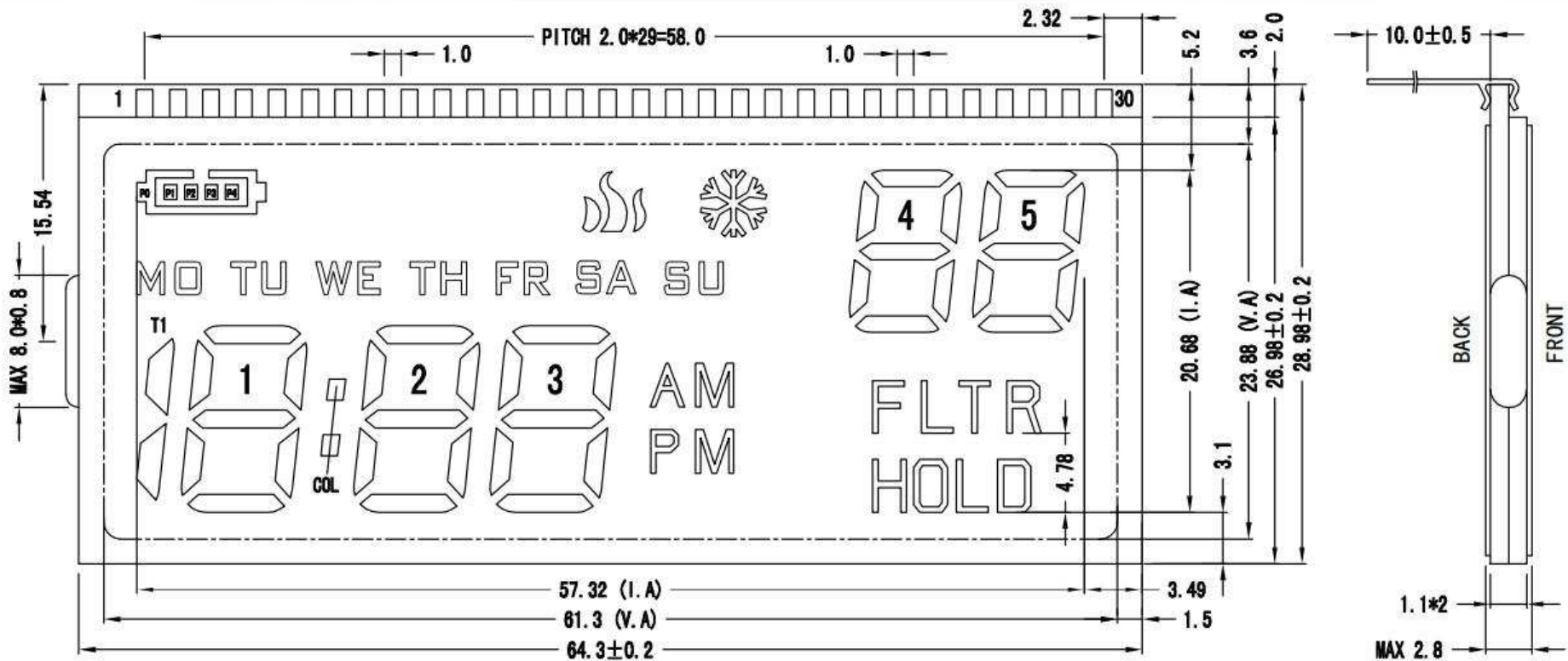


- K časování měřících 50ms použít TPM2
 - Nastavit TPM2 na 50ms, povolit přerušení
 - Převzít kód pro spuštění měření A/D a přesunout do přerušení od TPM2
- Nakonfigurovat TPM1
 - Nastavit 1 kanál (CH0) a zapnout jeho I/O pin
 - Nastavit modulo registr na 255 (odpovídá téměř max. hodnotě z ADC)
 - Nastavit value-registr (položka "Channel compare") na "polovinu" = střída PWM 50%
 - Nastavit předděličku tak, aby generovaná perioda byla cca 1kHz (= 1ms)
 - Nezapomenout v kódu nějaké použití LED1
- Přidat do modulu Events globální proměnnou plněnou podle ADC (osa X)
- V main modulu mít tuto proměnnou jako extern
- Hlavní smyčka
 - Podle proměnné plnit modulo registr TPM1MOD a poloviční hodnotou datový registr TPM1C0V (oba 16-bitové)
 - Konverze byte-word je implicitní, velikost operandů se přizpůsobí "největšímu datovému typu"
 - Předpokládáme Edge-aligned režim TPM1
- **!! Spouštět TPM1 podle stavu tlačítka BUTTON1 (v klidu nepískat !!!)**
 - Nejlépe pomocí bitu TPM1 v registru SCGC1

Plán cvičení

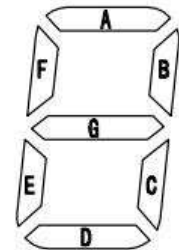
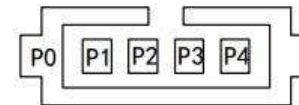
1. Úvod a seznámení – laboratoř, bloky uP, prostředí CodeWarrior (simulátor+debugger)
2. I/O porty – tlačítka a LEDky
3. Tlačítka, časovač a přerušení
4. Sériový port
5. Procvičení – sériový port, I/O, časovač, přerušení
6. A/D převodník + akcelerometr
7. Další možnosti využití časovače a přerušení
8. Dokončení přerušení, časovač v režimu PWM
- 9. LCD displej**
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

LCD



PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
COM1	MO	PO	P3	P4	TU	1B	1C	1D	WE	TH	2B	2C	2D	FR			3B	3C
COM2	T1	P1	P2		1A	1F	1G	1E	COL	2A	2F	2G	2E	3A	SA	SU	3F	3G

PIN	19	20	21	22	23	24	25	26	27	28	29	30
COM1	3D	AM	FLTR	4D	4E	4F	4A	5E	5F	5A	COM1	
COM2	3E	PM	HOLD	5D	4C	4G	4B	5C	5G	5B		COM2



NOTES:

1. VIEWING ANGLE: 6:00 O' CLOCK
2. DISPLAY MODE: POSITIVE/REFLECTIVE/TN TYPE
3. DRIVING VOLTAGE: 2.7V, DUTY:1/2, BIAS:1/2
4. OPERATING TEMP.: 0° C TO 50° C
5. STORAGE TEMP.: -10° C TO 60° C
6. CONNECTOR: PIN TYPE

6				<i>S-Tek Displays</i>		<i>Cleveland, Ohio 44146</i>	
5							
4				PART NO.	GD-5360P	TOLERANCES UNLESS OTHERWISE STATED .X ±0.00 .XX ±0.10 ANGLES ±1°	APPLICATION: <input type="checkbox"/> Instrument <input type="checkbox"/> Radio Equipment <input type="checkbox"/> Watch Clock <input type="checkbox"/> Telephone <input type="checkbox"/> Automobile <input checked="" type="checkbox"/> Other <input type="checkbox"/> Home Appliances <input type="checkbox"/> Air-condition
3						ALL DIMENSIONS IN IN UNLESS OTHERWISE STATED CUSTOMER: S-TEK	SAMPLE: <input type="checkbox"/> Yes <input checked="" type="checkbox"/> No
2	MODIFY LOGIC AND ICON	18/06-08	L. M. L.	DESIGN BY	Q B	28/05-08	
1	MODIFY LOGIC	28/05-08	R. Y. Z.	CHECKED BY			
VER. MODIFY CONTENTS		DATE	DESIGN	APP	BY	SHEET 1 OF 1	

Semestrální práce - úvod

- Spojení jednotlivých bloků do funkční aplikace
- Zatím probrané
 - Sériový port
 - A/D převodník
 - Časovač
 - Jádru procesoru
 - I/O porty – LED-ky a tlačítka
- Samostatně doplnitelné bloky
 - PWM režim čítače – viz minulé "slajdy"
 - LCD displej
 - Podklady **Z:\podklady\MPP** – datasheet (TWR_LCD_GLASS_SPECIFICATION.pdf)
 - Knihovna (**LCD_MPP.C** a **LCD_MPP.H** soubor k nakopírování do projektu)
 - K inicializaci nutno zavolat **LCD_Init()**
 - K zobrazení číslice na 7-segmentovce možno využít **LCD_ShowNumber(byte pos, byte val)**
 - Zapnutí vypnutí symbolu/segmentu možno pomocí
 - **void LCD_SegmentOn(byte lcdpin, byte compin);**
 - **void LCD_SegmentOff(byte lcdpin, byte compin);**
 - Jednotlivé LCD segmenty jsou připraveny pomocí **#define**
 - Další HW komponenty viz. datasheet (MC9S08LL64RM.pdf)

Semestrální práce - témata

- Využít více bloků a ovládání přes SCI z PC
 - Nutno vhodně zvolit filozofii ovládání
- Náměty standardní ("na jedničku/dvojku")
 - Hodiny na LCD (aktuální čas nastavitelný z terminálu)
 - Stopky na LCD (start/stop/mezičas pomocí tlačítek – jako na hodinkách)
 - Melodický zvonek (tlačítkem spuštěná sekvence tónů)
 - "Klávesy" (po stisku tlačítka zahrát tón)
 - Zobrazit zrychlení na LCD (výběr vstupu X/Y/Z pomocí terminálu)
- Hodnocení
 - Výborně – dělá co má + něco navíc (např. konfigurace terminálem, ...)
 - Velmi dobře – dělá co má dle požadavků cvičícího
 - Dobře – něco dělá, ale ne podle požadavků cvičícího
 - 4 - nefunguje

Konec aktuálních slajdů

- Následující slajdy jsou z minulých let a mohou být aktualizovány podle rychlosti zvládnání látky během cvičení
- Vhodné jako inspirace "co nás čeká"

Odložené slajdy !!

- Toto tyto podklady nejsou pro 2014/15 použity !!

Časové průběhy

- V main měníme stav LED2
 - Generování obdélníkového signálu
 - Výpadky po dobu obsluh přerušení
- Pro synchronizaci indikuje LED3
 - Do 0 při spuštění převodů
 - Do 1 při ukončení převodů
- Během "převodů" se main nestihne
 - Převody trvají cca $3\mu\text{s}$ (při 8MHz)
 - Režie vyvolání/ukončení interruptu je několik instrukcí (řádově μs)
 - Plnění bufferu také nějakou dobu trvá
- Po cca 50ms je krátký výpadek main – další byte při 19200b/s
- Podobně za dalších 50ms (nutno posunout synchronizaci)

