

Cvičení z předmětu

KAE/MPP

verze 2013-4

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
	Ing. Lukáš Paločko	lpalocko@kae.zcu.cz	EL505

Plán cvičení

1. **Úvod a seznámení – laboratoř, bloky uP, opak. C**
2. Prostředí CodeWarrior – simulátor + debugger
3. I/O porty
4. Časovač
5. Přerušení
6. Sériový port
7. LCD displej
8. Využití časovače v režimu PWM
9. Další možnosti využití časovače a přerušení
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

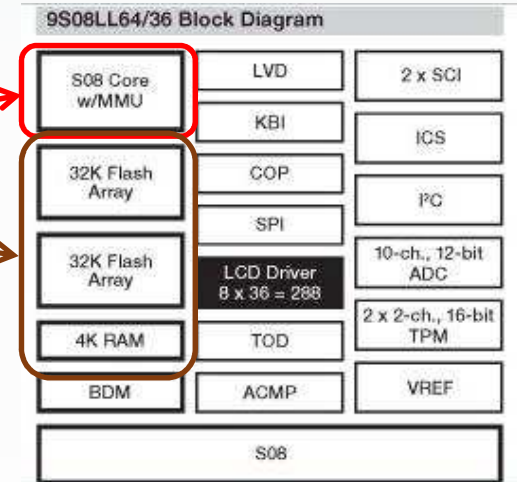
- Konzultace nejlépe v rámci cvičení
- Možno využívat laboratoř mimo své cvičení, respektovat rozvrh
- Ve volných hodinách mají v laboratoři přednost diplomanti

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 – 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"

Vnitřní bloky jednočipových mikropočítačů

- Jednočipový mikropočítač se skládá:
 - 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)
 - Hlídaní 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

Opakování C

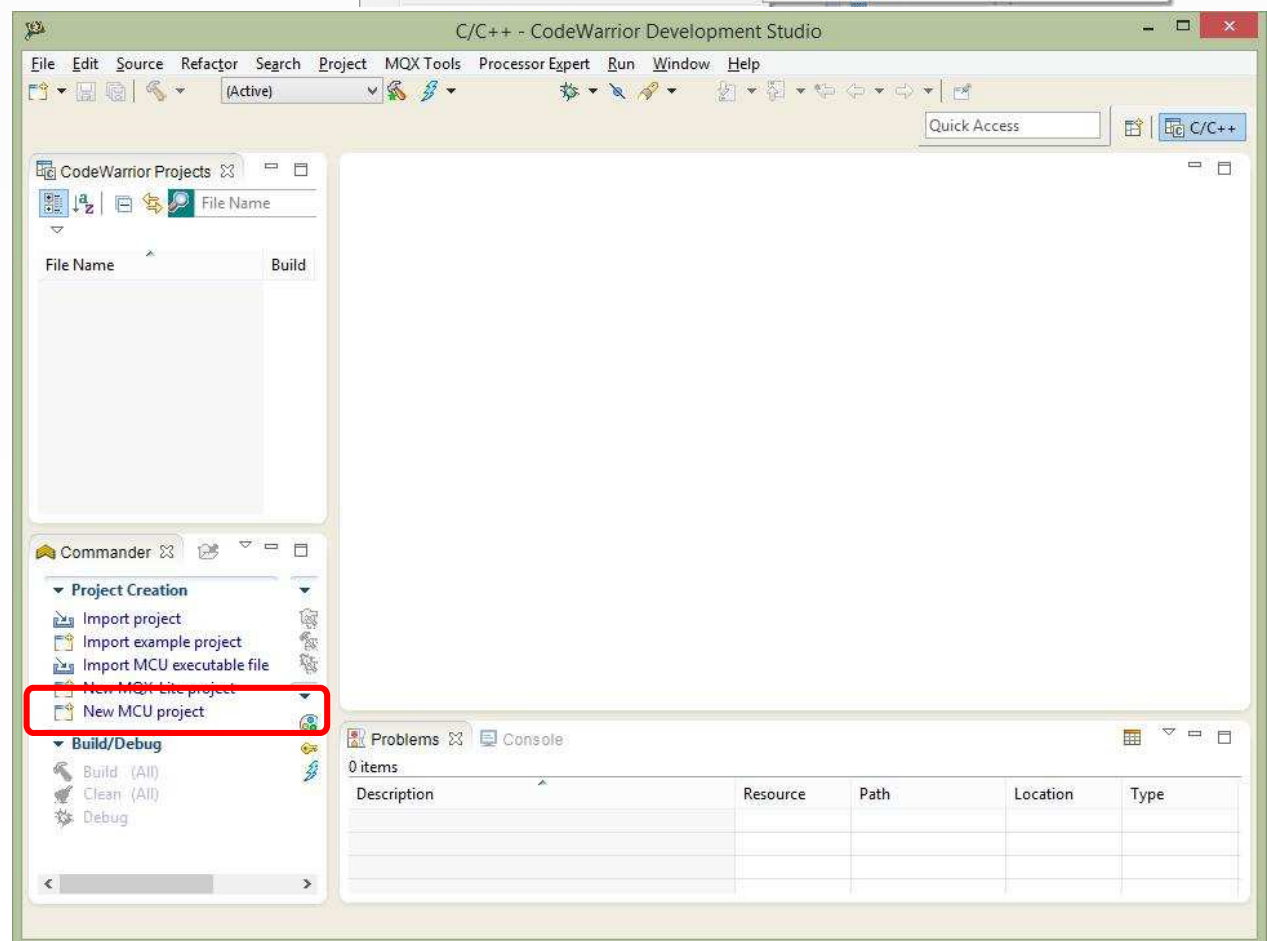
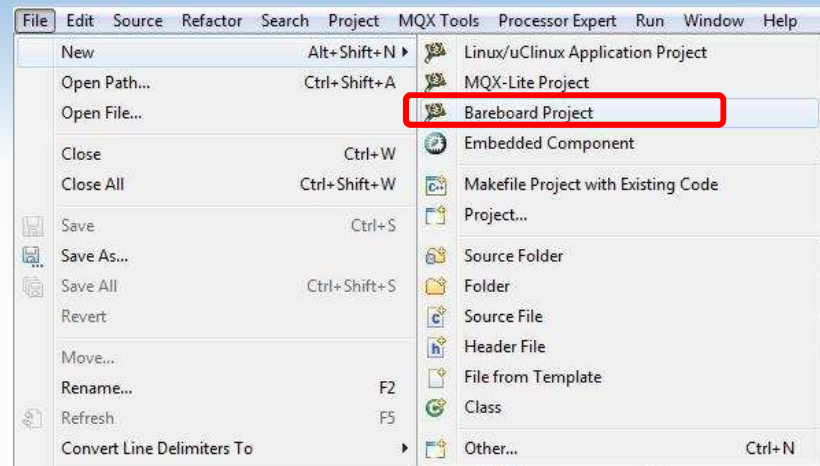
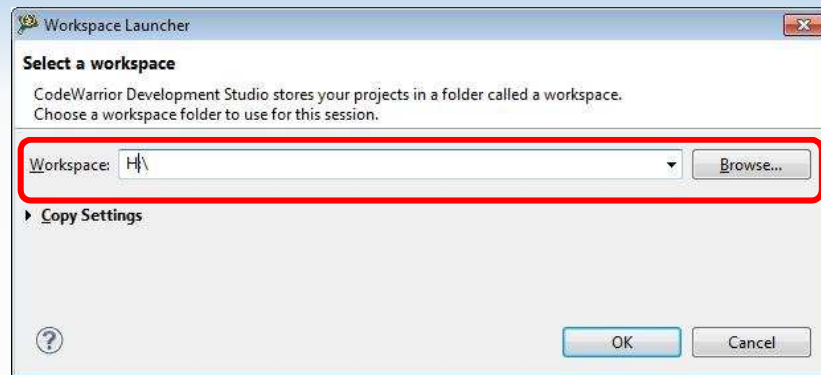
- Ověření znalostí základních konstrukcí

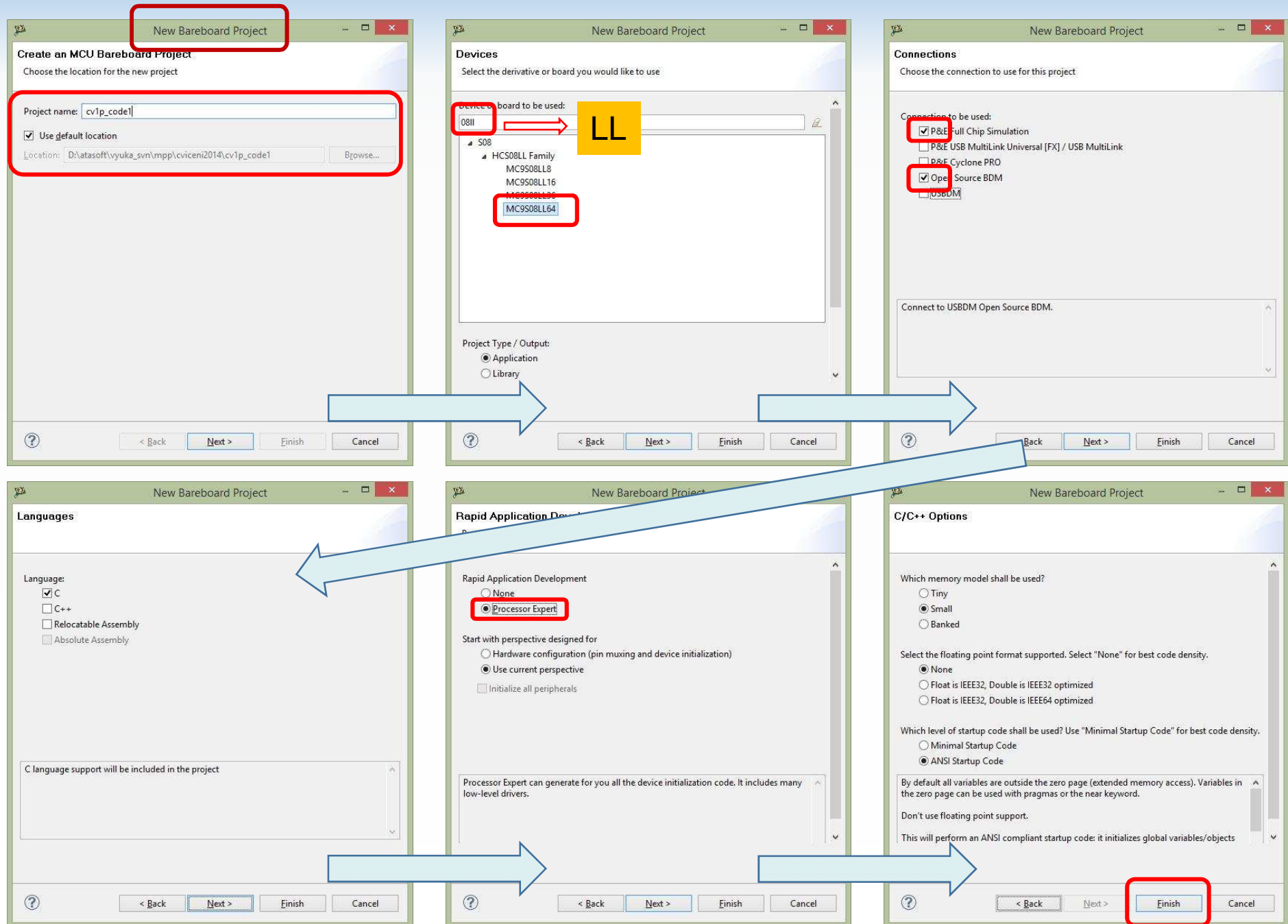
Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, opak. C
- 2. Prostředí CodeWarrior – simulátor + debugger**
3. I/O porty
4. Časovač
5. Přerušení
6. Sériový port
7. LCD displej
8. Využití časovače v režimu PWM
9. Další možnosti využití časovače a přerušení
10. Pokročilé algoritmy
11. Semestrální práce
12. Semestrální práce
13. Semestrální práce + Zápočet

Spuštění prostředí

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





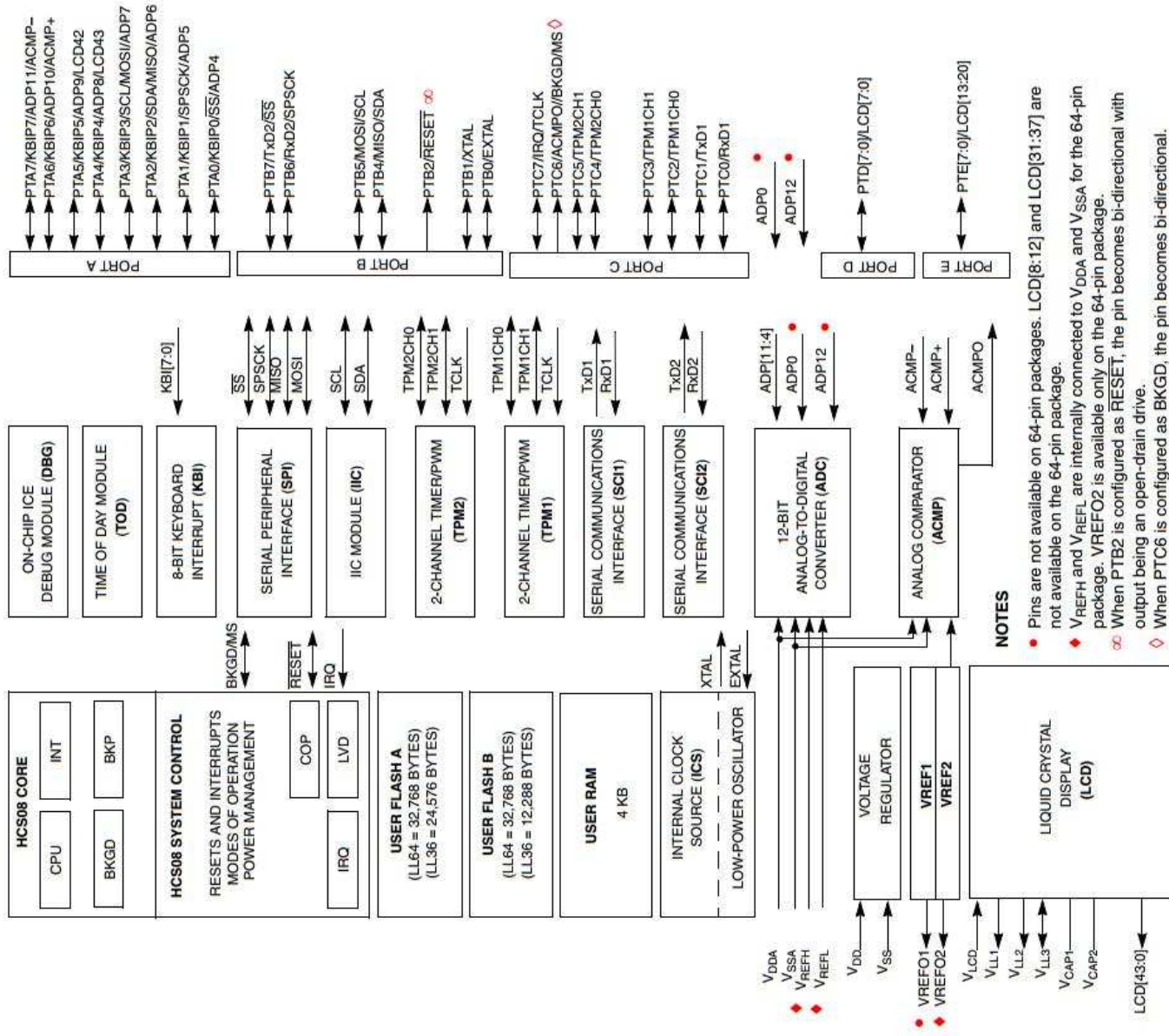


Figure 1-1. MC9S08LL64 Series Block Diagram

Processor Expert

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

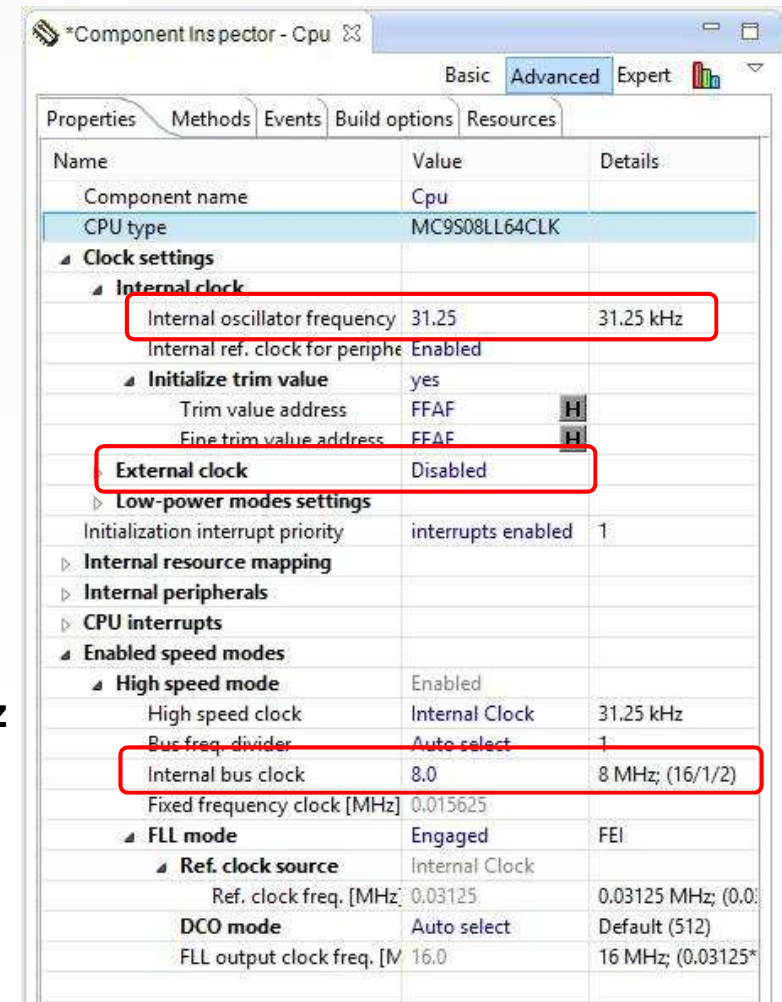
The screenshot displays the CodeWarrior Development Studio interface with the Processor Expert configuration for the MC9S08LL64CLK processor. The 'Select Package For The Processor' dialog is open, showing available packages: MC9S08LL64CLK 80-pins LQFP and MC9S08LL64CLH 64-pins LQFP. The 'Configuration R...' window shows the peripheral register table. The 'Processor' window displays the pin diagram and the processor name MC9S08LL64CLK. The 'Component Inspector - Cpu' window shows the properties of the CPU component, including clock settings and initialization interrupt priority.

Reg. name	Init. value
ICSC1	06
ICSC2	40
ICSTRM	????????
ICSSC	00010000?
SRS	82
SOPT1	43
SOPT2	00
SDIDL	????0000
SDIDL	26
SPMSC1	1C
SPMSC2	02
SPMSC3	00
SCGC1	FF
SCGC2	FF
PTASE	00
PTADS	FF
PTBSE	08
PTBDS	F7
PTCSE	00

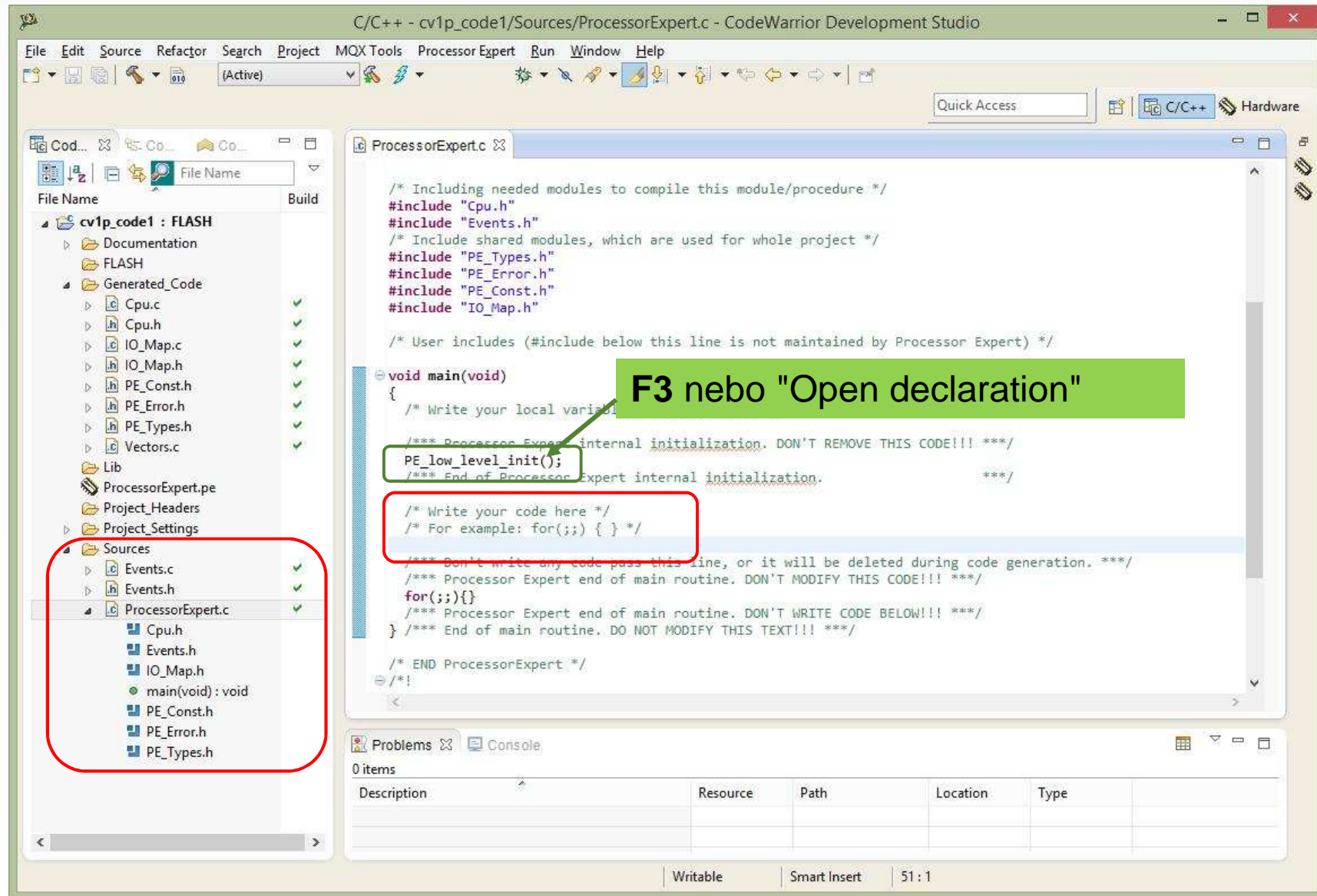
Name	Value
CPU type	MC9S08LL64CLK
Clock settings	
Initialization interrupt priority	interrupts enabled
CPU interrupts	
Enabled speed modes	
High speed mode	Enabled
High speed clock	Internal Clock
Internal bus clock	4,194304
Fixed frequency clock [MHz]	0,016384
FLL mode	Engaged
Ref. clock source	Internal Clock
Ref. clock freq. [MHz]	0,032768
DCO mode	Auto select

Volba zdroje hodin

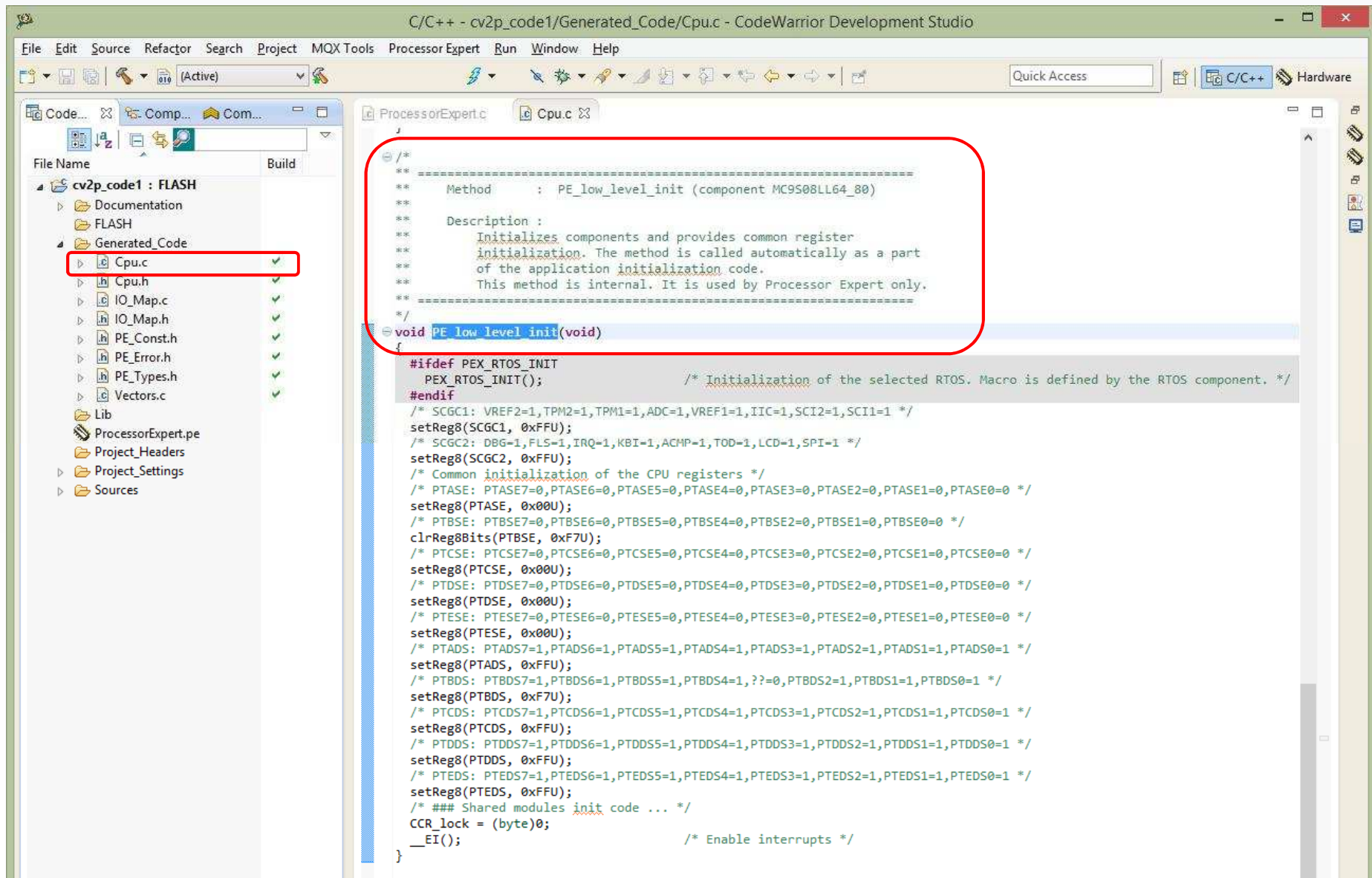
- Procesor potřebuje pro svoji činnost hodiny
 - Pro každou periférii je možné hodiny zapínat, příp. vybrat zdroj a volitelně i děličky
- Na S08 máme k dispozici
 - Interní (méně přesný) generátor
 - Možnost externího krystalu nebo generátoru
 - Vnitřními děličkami a násobičkami (s PLL) možno vytvořit 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

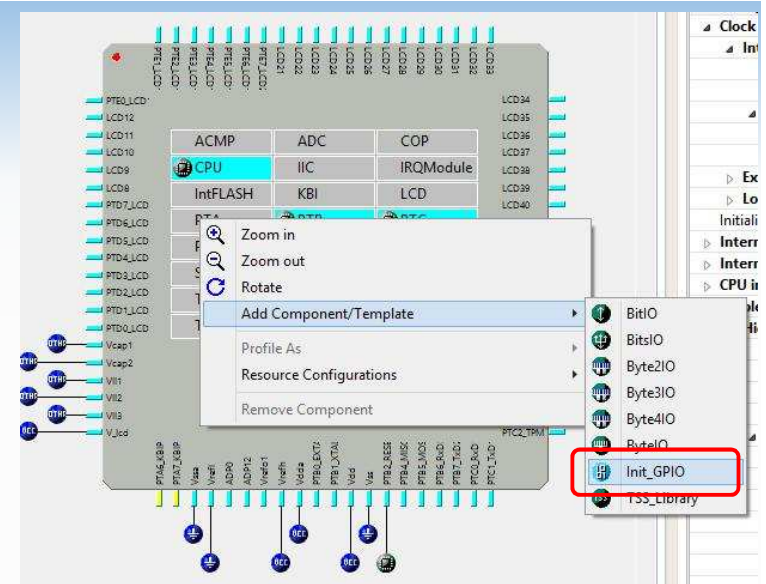


Vygenerovaná inicializace – CPU.C



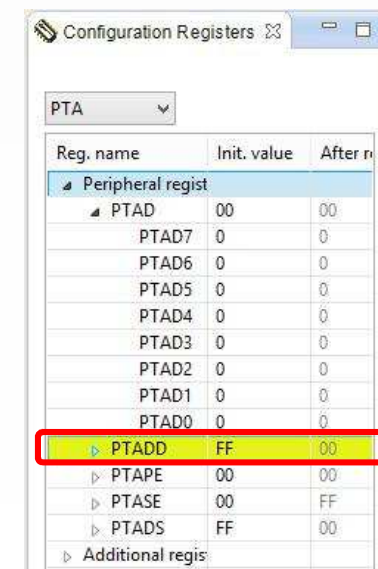
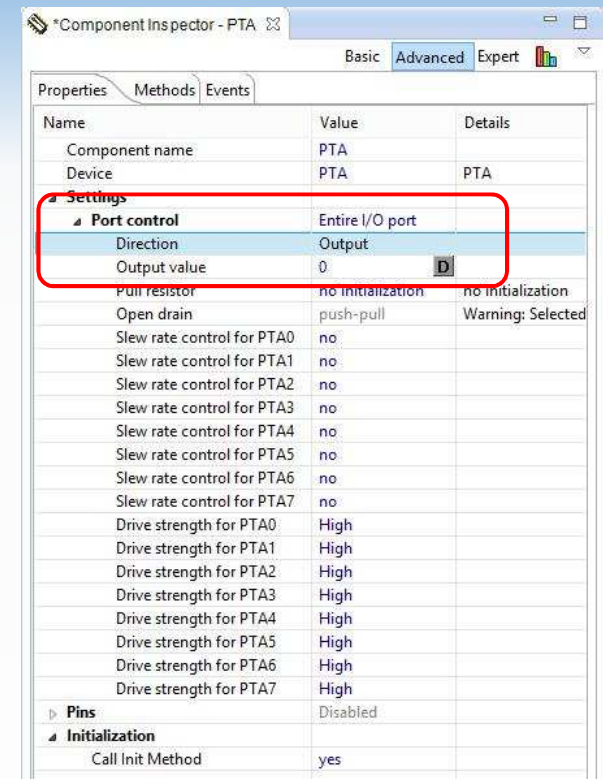
Přidání komponenty do PE

- 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 Register brány A)
- Pokud bychom nechtěli oddělenou deklaraci a inicializaci proměnných od samotného kódu, je možné použít trik s C-čkovým blokem

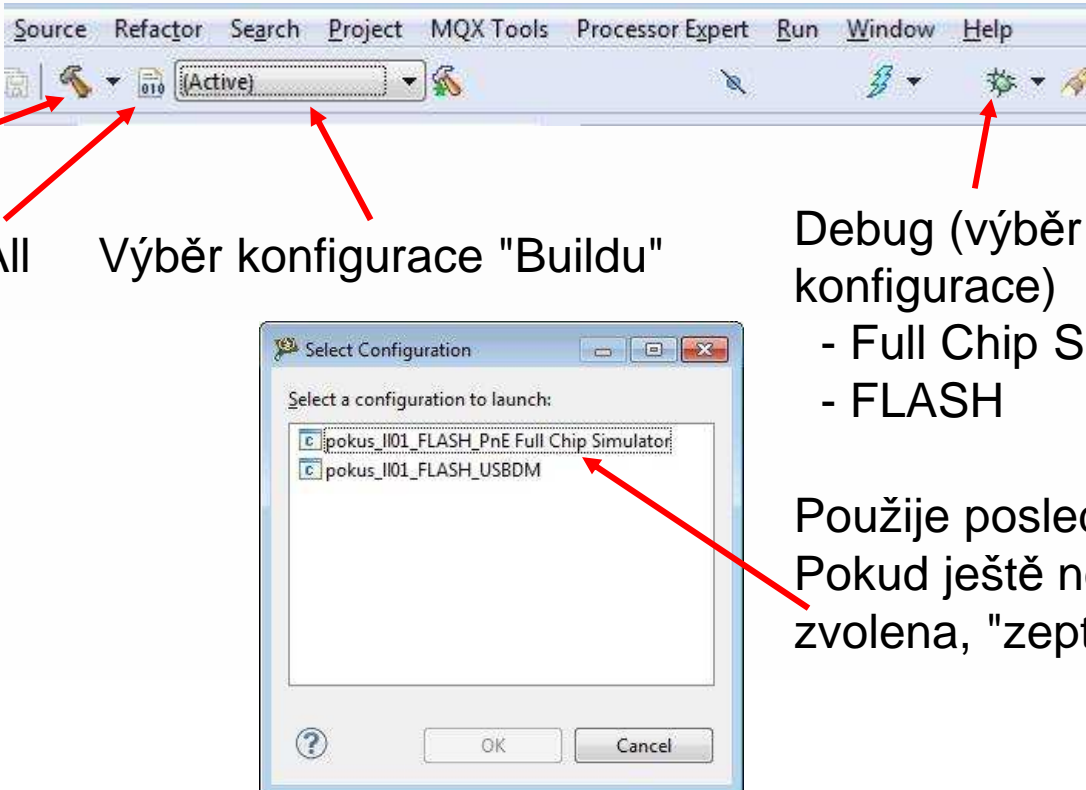
```
42 void main(void)
43 {
44     /* Write your local variable definition here */
45     byte b;
46
47     /** Processor Expert internal initialization. DON'T REMOVE */
48     PE_low_level_init();
49     /** End of Processor Expert internal initialization.
50
51     /* Write your code here */
52     /* 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++;
    }
}
...
```

Spuštění programu v simulátoru

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



The image shows a screenshot of an IDE toolbar and a 'Select Configuration' dialog box. The toolbar has several icons: a hammer (Build), a hammer with a gear (Build All), a dropdown menu (Výběr konfigurace "Buildu"), and a bug (Debug). Red arrows point from these icons to labels below them: 'Build', 'Build All', 'Výběr konfigurace "Buildu"', and 'Debug (výběr známé konfigurace)'. The 'Debug' label also points to a list of configurations in the 'Select Configuration' dialog box. The dialog box has a title bar 'Select Configuration' and a list of configurations: 'pokus_I101_FLASH_PnE Full Chip Simulator' and 'pokus_I101_FLASH_USBDM'. A red arrow points from the text 'Použije poslední konfiguraci Pokud ještě nebyla žádná zvolena, "zeptá se"' to the 'pokus_I101_FLASH_PnE Full Chip Simulator' configuration.

Build Build All Výběr konfigurace "Buildu" Debug (výběr známé konfigurace)

- Full Chip Simulator
- FLASH

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

Debug - cv2_p1\Sources\ProcessorExpert.c - CodeWarrior Development Studio

File Edit Source Refactor Search Project RTCS MQX MQXTools PEMicro Run Window Help

Debug

cv2_p1_FLASH_PnE Full Chip Simulator [CodeWarrior]

HCS08, cv2_p1.abs (Suspended)

Thread [ID: 0x0] (Suspended: Signal 'Halt' received. Description: User halted thread.)

1 main() ProcessorExpert.c:42 0x00192d

F:\cv2_p1\FLASH\cv2_p1.abs (2/17/14 8:54 AM)

Quick Access

Memory

Registers

Breakpoints

Variables

Modules

Hardware

Debug

Location

0x000180 Ram

Name

Value

0x b

1

Disassembly

void main(void)

42 88 PSHH

47 PE_low_level_init();

00192e: CD18E5 JSR 0x18E5 PE_low_level_init (0x18e5)

57 PTAD = b;

001931: 95 TSX

001932: F6 LDA ,X

001933: B700 STA 0x00

58 b++;

001935: 7C INC ,X

55 while(1)

001936: 20F9 BRA *-5 main+0x4 (0x1931)

loadByte:

74 88 PSHH

001938: 88 PSHH

75 89 PSXH

001939: 89 PSXH

77 LDHX 5,SP

00193a: 9EFE05 LDHX 5,SP

78 LDA 0,X

00193d: F6 LDA ,X

79 AIX #1

00193e: AF01 AIX #1

8a CTHY 5 CD

ProcessorExpert.c

40 /* User includes (#include below this line is not maintained by Processor Expert) */

41

42 void main(void)

43 {

44 /* Write your local variable definition here */

45

46 /** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! */

47 PE_low_level_init();

48 /** End of Processor Expert internal initialization. */

49

50 /* Write your code here */

51 /* For example: for(;;) { } */

52 {

53 byte b;

54

55 while(1)

56 {

57 PTAD = b;

58 b++;

59 }

60 }

61

62

63 /** Don't write any code pass this line, or it will be deleted during code generation

Console

Problems

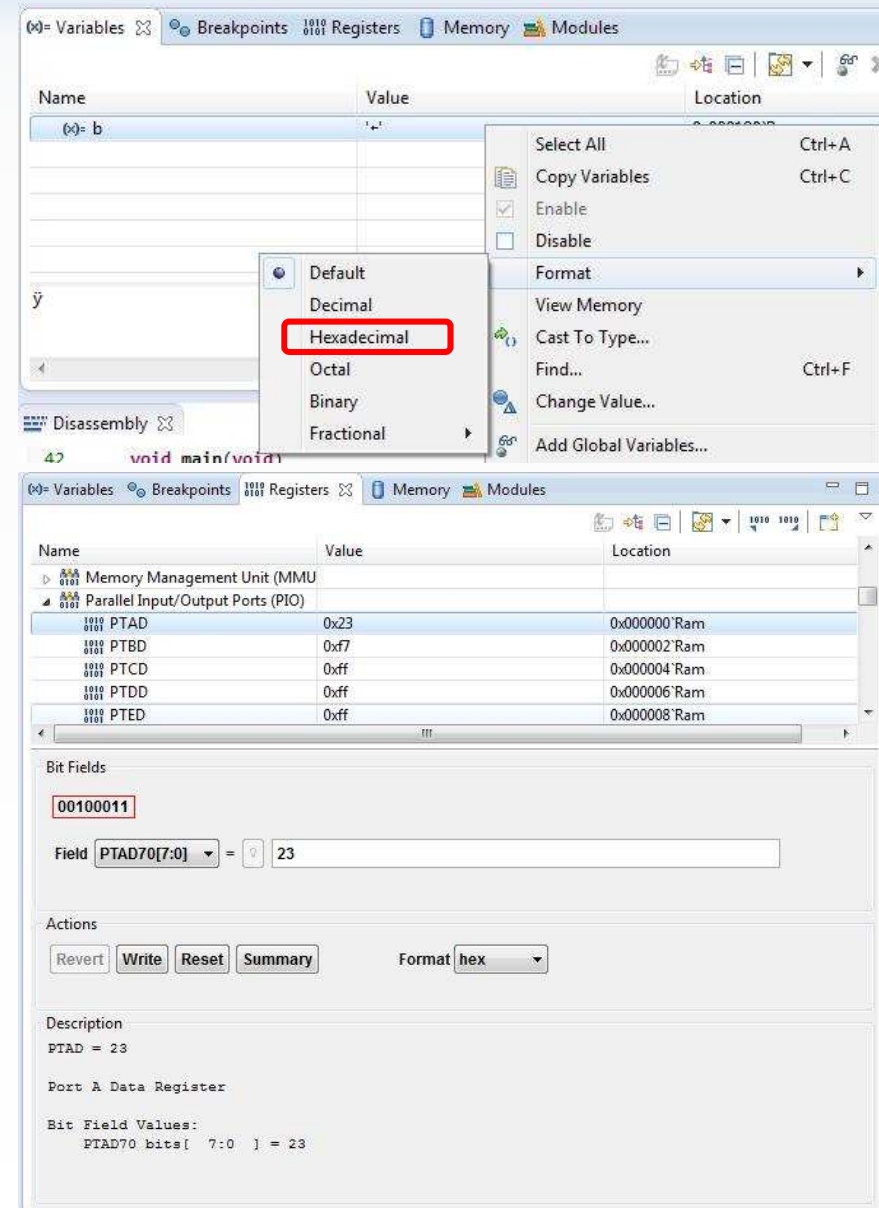
HCS08, cv2_p1.abs

Writable Smart Insert 42:1

Krokování



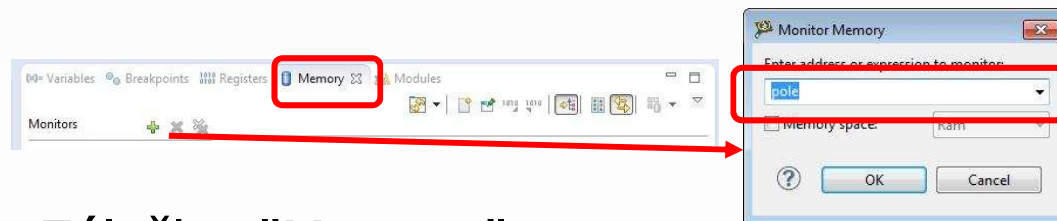
- **F5 – Step Into** – krok, vstupuje do funkcí
- **F6 – Step Over** – krok, "nevleze" do funkce
- **F7 – Step Return** – dokončí funkci
- **F8 – Resume** – pokračuje ve vykonávání, příp. spustí od začátku (napoprvé)
- Aktuální řádek vyznačen ➡
- Je možné krokovat i po instrukcích assembleru – přepínání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



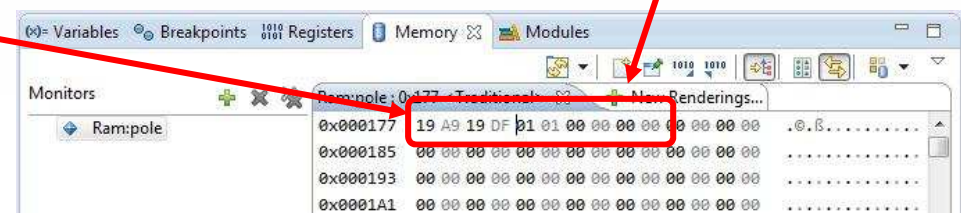
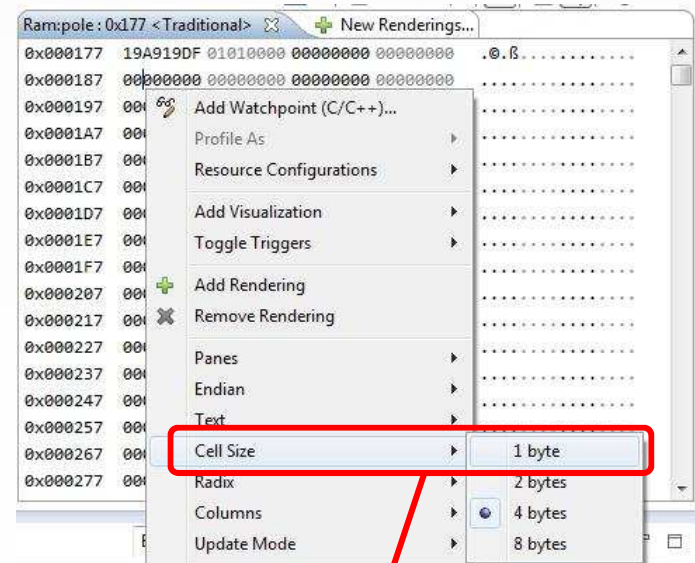
Ladění obsahu paměti

- Kód plní 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



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

```
...
{
    while(1)
    {
        PTAD_PTAD0 = 1;

        _PTAD.Bits.PTAD0 = 0;
    }
}
...
```

```
80 {
81     while(1)
82     {
83         PTAD_PTAD0 = 1;
84
85         _PTAD.Bits.PTAD0 = 0;
86     }
87 }
88
```

Bits: {IO_Map.h:4357}
Byte: byte

```
84
85 _PTAD.Bits.PTAD0 = 0;
86 }
87 }
88
89
90 /** Don't write to PTAD register */
91 /** Processor will write to PTAD register */
```

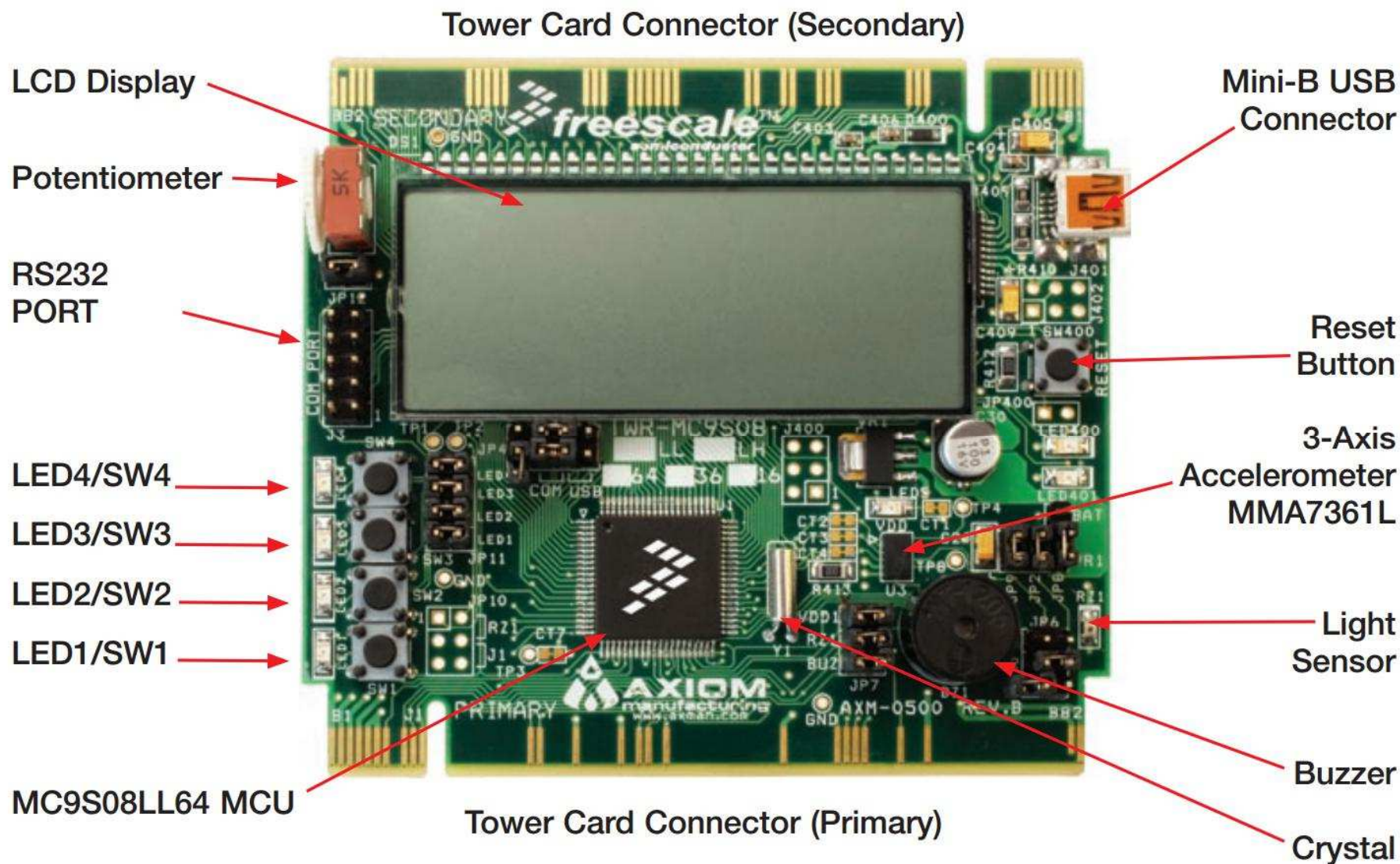
PTAD0: byte
PTAD1: byte
PTAD2: byte
PTAD3: byte

```
106 /** PTAD - Port A Data Register; 0x00000000 */
107 typedef union {
108     byte Byte;
109     struct {
110         byte PTAD0 :1; /* Port A Data Register Bit 0 */
111         byte PTAD1 :1; /* Port A Data Register Bit 1 */
112         byte PTAD2 :1; /* Port A Data Register Bit 2 */
113         byte PTAD3 :1; /* Port A Data Register Bit 3 */
114         byte PTAD4 :1; /* Port A Data Register Bit 4 */
115         byte PTAD5 :1; /* Port A Data Register Bit 5 */
116         byte PTAD6 :1; /* Port A Data Register Bit 6 */
117         byte PTAD7 :1; /* Port A Data Register Bit 7 */
118     } Bits;
119 } PTADSTR;
120 extern volatile PTADSTR _PTAD @0x00000000;
121 #define PTAD _PTAD.Byte
122 #define PTAD_PTAD0 _PTAD.Bits.PTAD0
123 #define PTAD_PTAD1 _PTAD.Bits.PTAD1
124 #define PTAD_PTAD2 _PTAD.Bits.PTAD2
125 #define PTAD_PTAD3 _PTAD.Bits.PTAD3
126 #define PTAD_PTAD4 _PTAD.Bits.PTAD4
127 #define PTAD_PTAD5 _PTAD.Bits.PTAD5
128 #define PTAD_PTAD6 _PTAD.Bits.PTAD6
129 #define PTAD_PTAD7 _PTAD.Bits.PTAD7
```

Plán cvičení

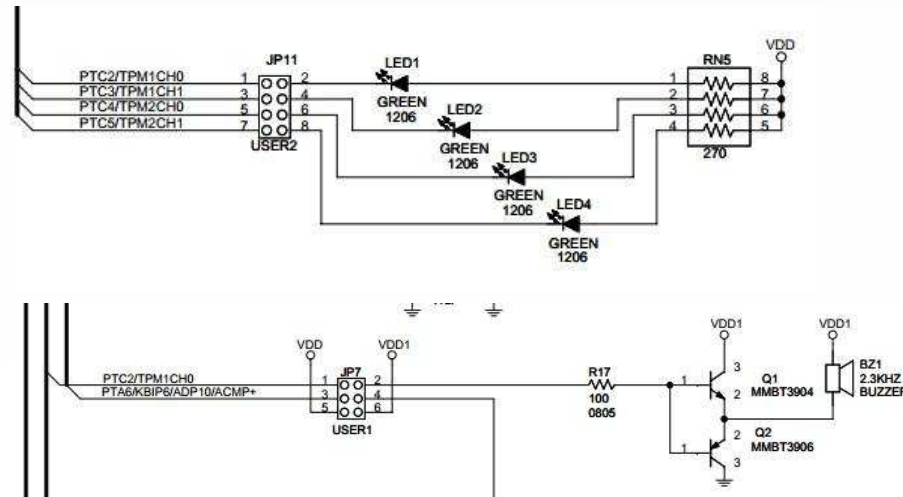
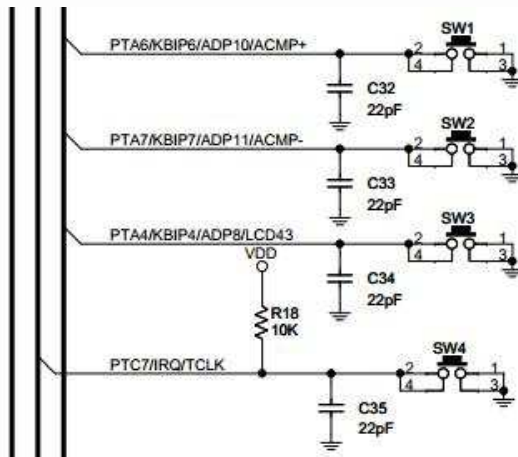
1. Úvod a seznámení – laboratoř, bloky uP, opak. C
2. Prostředí CodeWarrior – simulátor + debugger
3. Opakování C algoritmů
- 4. I/O porty – tlačítka a LEDky**
5. Časovač
6. Přerušení
7. Sériový port
8. LCD displej
9. Využití časovače v režimu PWM
10. Další možnosti využití časovače a přerušení
11. Pokročilé algoritmy
12. Semestrální práce
13. Semestrální práce + Zápočet

Vývojový kit Freescale S08LL64



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
- Na PTC2 připojen ještě "buzzer"



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

Nastavení v CW - PE

Name	Value	Details
Device	PTA	PTA
Settings	Individual pins	
Port control	Enabled	
Pins	Enabled	
Pin0	Disabled	PTA0_KBIP0_SS_ADP4
Pin1	Disabled	PTA1_KBIP1_SPSCK_ADP5
Pin2	Disabled	PTA2_KBIP2_SDA_MISO_ADP6
Pin3	Disabled	PTA3_KBIP3_SCL_MOSI_ADP7
Pin4	Enabled	PTA4_KBIP4_ADP8_LCD43
Pin	PTA4_KBIP4_ADP...	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	PTA6_KBIP6_ADP10_ACMPLUS
Pin	PTA6_KBIP6_ADP...	PTA6_KBIP6_ADP10_ACMPLUS
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	PTA7_KBIP7_ADP11_ACMPLUS
Pin	PTA7_KBIP7_ADP...	PTA7_KBIP7_ADP11_ACMPLUS
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	Individual pins	
Port control	Enabled	
Pins	Enabled	
Pin0	Disabled	PTC0_RxD1
Pin1	Disabled	PTC1_TxD1
Pin2	Enabled	PTC2_TPM1CH0
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	PTC3_TPM1CH1
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	PTC4_TPM2CH0
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	PTC5_TPM2CH1
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	PTC7_IRQ_TCLK
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	

Vlastní kód v ProcessorExpert.c

- Platforma pro DEBUG – OSBDM
- Doporučené "define"
 - Nemusí se stále vypisovat porty a piny
 - V případě změny HW stačí jen upravit na jednom místě
- Místo v kódu doporučené komentářem

```
/* User includes (#include below this line is not maintained by Processor Expert) */
#define LED1          PTCB_PTCB2
#define LED2          PTCB_PTCB3
#define LED3          PTCB_PTCB4
#define LED4          PTCB_PTCB5

#define BUTTON1       PTAD_PTAD6
#define BUTTON2       PTAD_PTAD7
#define BUTTON3       PTAD_PTAD4
#define BUTTON4       PTCB_PTCB7
```

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++)
        ;
}
```

- 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;

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

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

Běžící bod na LED – v2

```
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++)
        ;
}
```

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;

    PTCR = (PTCR & 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 PTCR registru

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

PTCR	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 PTCR, B = negovane spodni 4 bity z b

LED kopírují tlačítka

```
while(1)
{
    LED1 = BUTTON1;
    LED2 = BUTTON2;
    LED3 = BUTTON3;
    LED4 = BUTTON4;
}
```

Zap/vyp jedním tlačítkem

```
boolean bb = FALSE;  
  
...  
  
while(1)  
{  
    if (bb != BUTTON1)  
    {  
        bb = BUTTON1;  
  
        if (!bb)  
            LED1 = !LED1;  
    }  
}
```

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, opak. C
2. Prostředí CodeWarrior – simulátor + debugger
3. Opakování C algoritmů
4. I/O porty – tlačítka a LEDky
- 5. Časovač a přerušení**
6. Přerušení – pokračování
7. Sériový port
8. LCD displej
9. Využití časovače v režimu PWM
10. Další možnosti využití časovače a přerušení
11. Pokročilé algoritmy
12. Semestrální práce
13. Semestrální práce + Zápočet

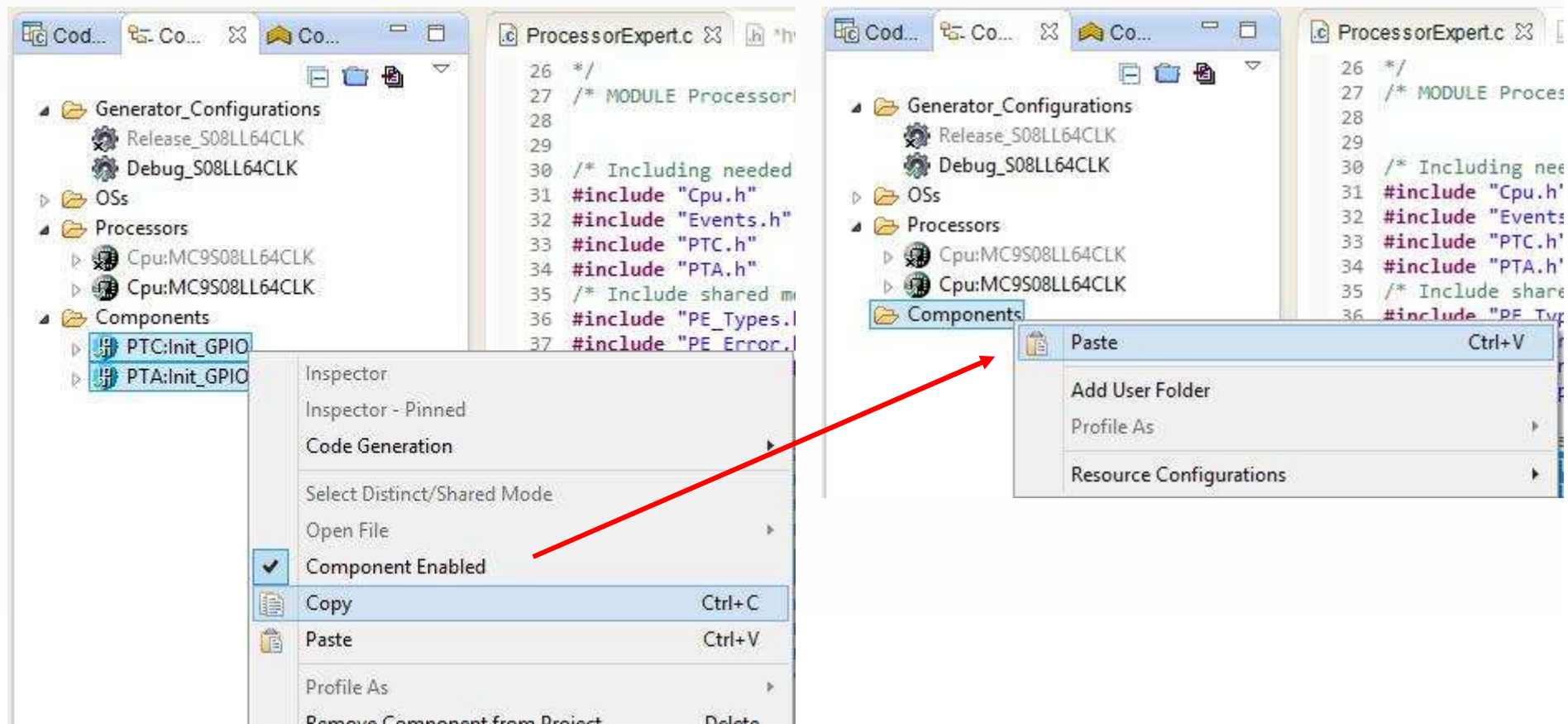
Hlavičkový soubor s HW konfigurací

The image shows a screenshot of an IDE interface. On the left, a project tree shows a folder named 'Source' highlighted with a red box. A right-click context menu is open over this folder, with the 'New' option selected. The 'New' submenu is open, and 'Header File' is highlighted with a red box. A red arrow points from the 'Source' folder to the 'Header File' option. To the right, a 'New Header File' dialog box is open. The 'Source folder' field is set to 'cv5p/Sources'. The 'Header file' field is set to 'hw_cfg.h', which is also highlighted with a red box. The 'Template' dropdown is set to 'Default C header template'. At the bottom of the dialog are 'Finish' and 'Cancel' buttons. Below the IDE screenshot, a code snippet is shown in a grey box:

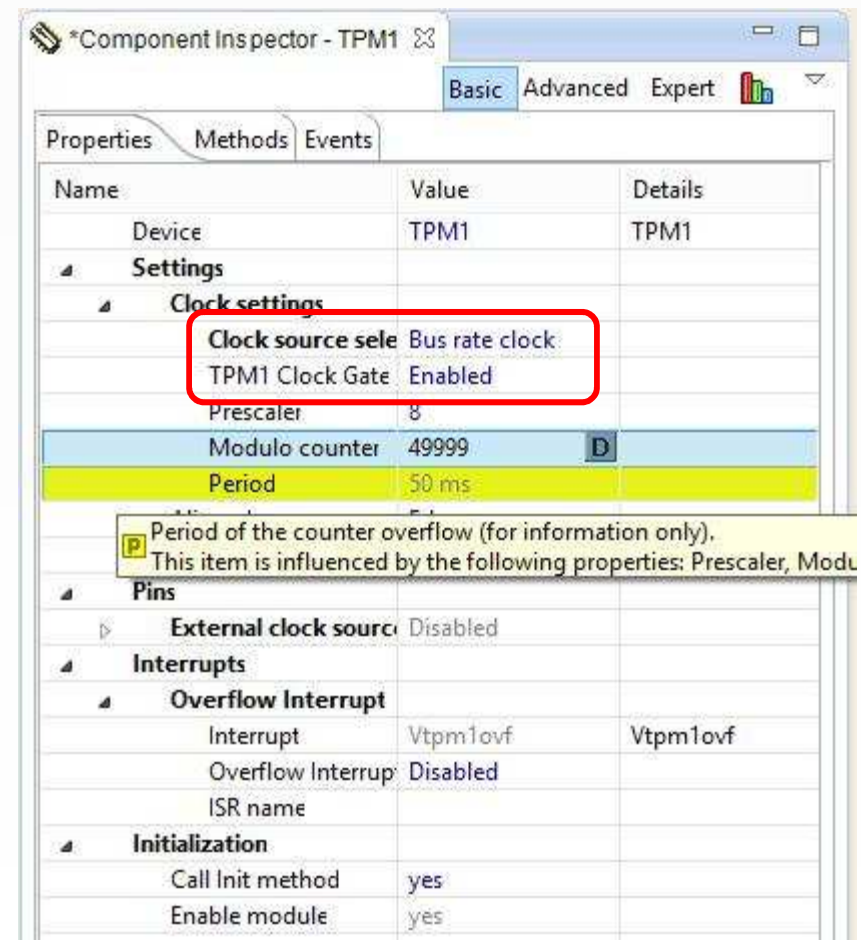
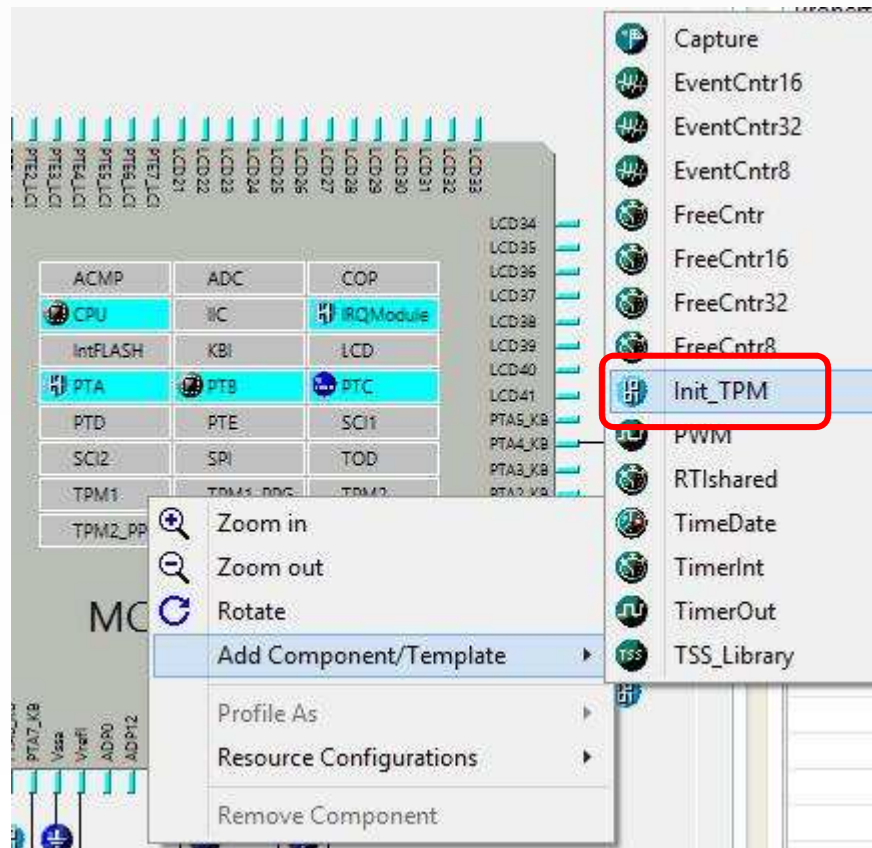
```
...  
#ifndef HW_CFG_H_  
#define HW_CFG_H_  
  
#define LED1    PTC_D_PTC_D2  
#define LED2    PTC_D_PTC_D3  
#define LED3    PTC_D_PTC_D4  
#define LED4    PTC_D_PTC_D5  
  
#define BUTTON1 PTAD_PTAD6  
#define BUTTON2 PTAD_PTAD7  
#define BUTTON3 PTAD_PTAD4  
#define BUTTON4 PTC_D_PTC_D7  
  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

Kopírování komponent mezi projekty

- Nezapomenout interní "hodiny" na **31.25kHz** a BUS na **8MHz**
- Záložka "Components"
 - Komponenty z jednoho projektu možno kopírovat do jiného včetně všech nastavení

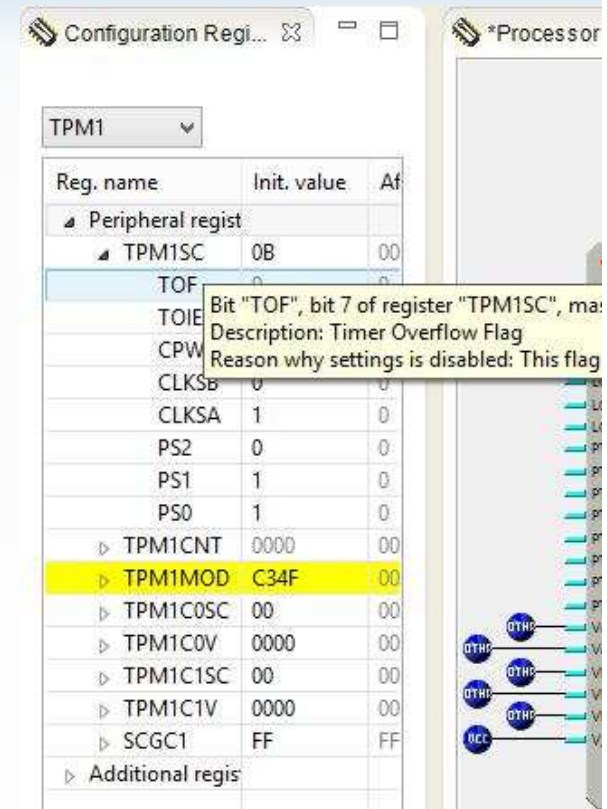


Přidání inicializace časovače



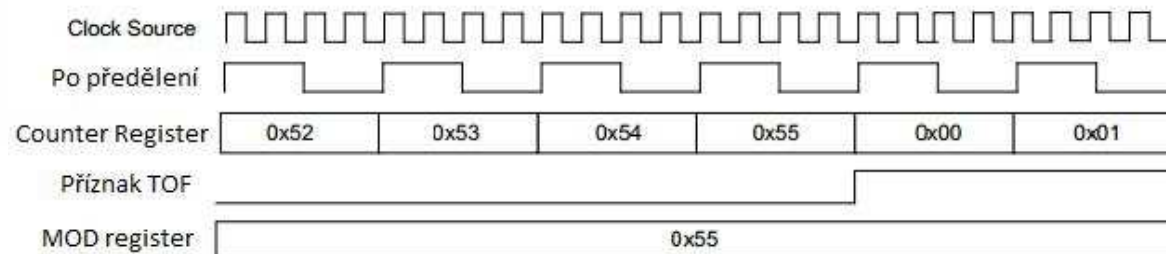
Práce s časovačem

- 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



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

- 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 rámeček kódu pro přerušení = nutno povolit ve vlastnostech projektu
- V komponentě Timer pak:
 - Povolit "přerušení pro přetečení"
 - Zvolit jméno obsluhy přerušení (Interrupt Service Routine)

The screenshot shows the Processor Expert IDE interface. On the left, the 'Properties' window is open for the 'cv5p' project, with the 'Processor Expert' tab selected. A red arrow points from the 'Properties' menu item in the top toolbar to this tab. The 'Processor Expert Project Options - cv5p' dialog is displayed in the center. The 'Generate ISRs' option is checked, and the 'Delete unused events and ISRs' option is also checked, with a red box highlighting these two options. On the right, the 'Interrupts' section is expanded, showing the 'Overflow Interrupt' configuration. The 'ISR name' is set to 'pretečení', and the 'Call Init method' and 'Enable module' options are both checked. A blue box highlights the 'Interrupts' section.

Processor Expert Project Options - cv5p		
Name		
Main & Events directory		
Generated code directory	Generated_Code	
Static code directory	Documentation	
Documentation directory	Project_Settings	
Project settings directory	Yes	
Set periph. init component name as periphe...	Smart update (recommended)	
Main module update	Smart update (recommended)	
Event module(s) update	Smart update (recommended)	
Update of other user modules	Yes	
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	

Interrupts		
External clock source	Disabled	
Overflow Interrupt	Enabled	
Interrupt	Vtpm1ovf	Vtpm1ovf
Overflow Interrupt	Enabled	
ISR name	pretečení	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;

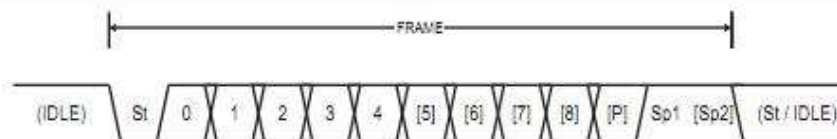
    LED2 = !LED2;
}
```

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, opak. C
2. Prostředí CodeWarrior – simulátor + debugger
3. Opakování C algoritmů
4. I/O porty – tlačítka a LEDky
5. Časovač a přerušení
- 6. Sériový port**
7. Přerušení – pokračování
8. LCD displej
9. Využití časovače v režimu PWM
10. Další možnosti využití časovače a přerušení
11. Pokročilé algoritmy
12. Semestrální práce
13. Semestrální práce + Zápočet

Sériový port – UART/SCI

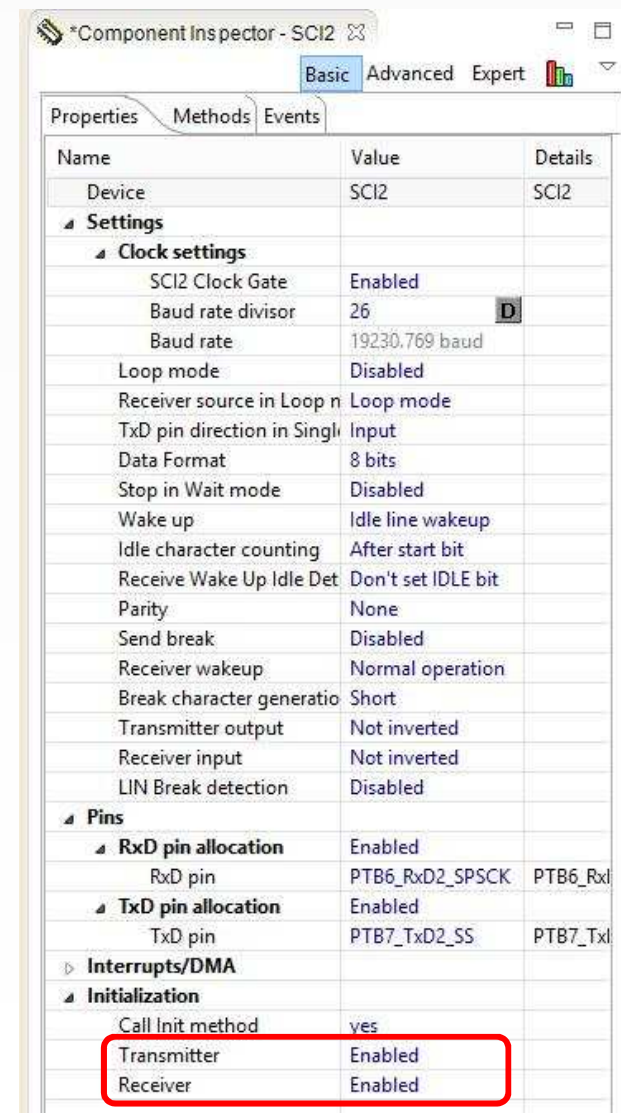
- Sériová komunikace vystačí s 2 signály + GND
 - RxD – receive data – příjem dat na zařízení
 - TxD – transmit data – vysílání dat ze zařízení
 - Typicky 8 datových bitů (může být 5-9b)
 - Přenos začíná start-bit, ukončuje stop-bit, příp. k datům se přidává parita



- 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 $\frac{1}{2}$ 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, rámec, ...) nastaví se chybový příznak
 - Zápis spouští vysílací sekvenci
 - Nutno příp. počkat na odvysílání předchozího
 - Pro datové operace možno využít interrupty
 - Tx interrupt – Transmit completed
 - Rx Interrupt – Receive request
 - Error interrupt



Stavový registr SCIS1

- 7 – TDRE – Transmit Data Register Empty
 - Data přenesena z datového do vysílacího shift-registru
 - Možno zapsat další data k odeslání do SCID
 - Po přečtení log. 1 se bit nuluje automaticky dalším zápisem do datového SCID
- 6 – TC – Transmission Complete
 - Dokončeno odeslání posledního bitu na lince
 - Po přečtení log. 1 se bit nuluje automaticky dalším zápisem do SCID nebo vypnutím SCI
- 5 – RDRF – Receive Data Register Full
 - Korektně přijata data do přijímacího registru
 - Po přečtení log. 1 se nuluje automaticky čtení SCID
- 4 – IDLE
- 3 – OR – Receiver Overrun
 - Přijata další data a předchozí nejsou odebrána
 - Nuluje se vyčtením SCID, obsahuje původní data, nová jsou zahozena
- 2 – NF – Noise
 - Hodnoty bitů nemají správné úrovně v době jejich vzorkování
 - Nuluje se čtením SCID, 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 SCID
- 0 – PF – Parity Error
 - Nevyšla parita v přijímaných datech, stavuje se společně s SCID

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)`

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 v10.5/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, opak. C
2. Prostředí CodeWarrior – simulátor + debugger
3. Opakování C algoritmů
4. I/O porty – tlačítka a LEDky
5. Časovač a přerušení
6. Sériový port
- 7. Procvičení – sériový port, I/O, časovač, přerušení**
8. LCD displej
9. Využití časovače v režimu PWM
10. Další možnosti využití časovače a přerušení
11. Pokročilé algoritmy
12. Semestrální práce
13. Semestrální práce + Zápočet

Opakování probraných 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.

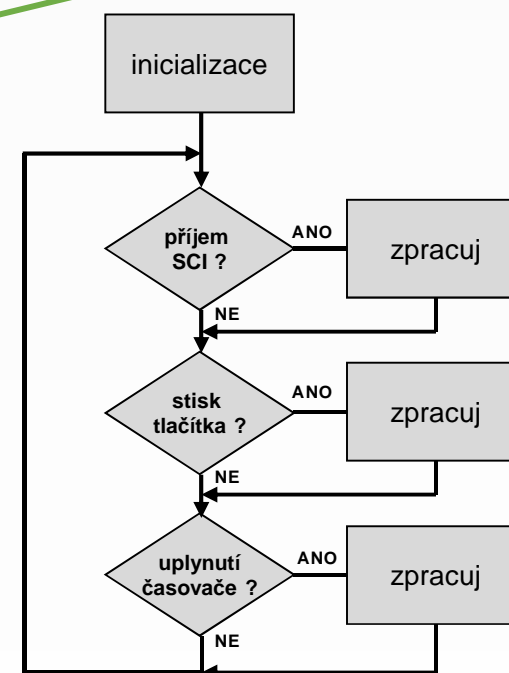
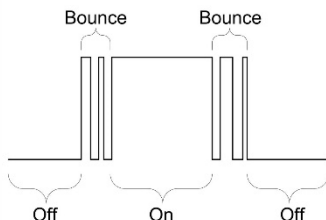
```
ISR(pretečení)
{
    (void)TPM1SC;    // použít trik k vynucení čtení

    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.



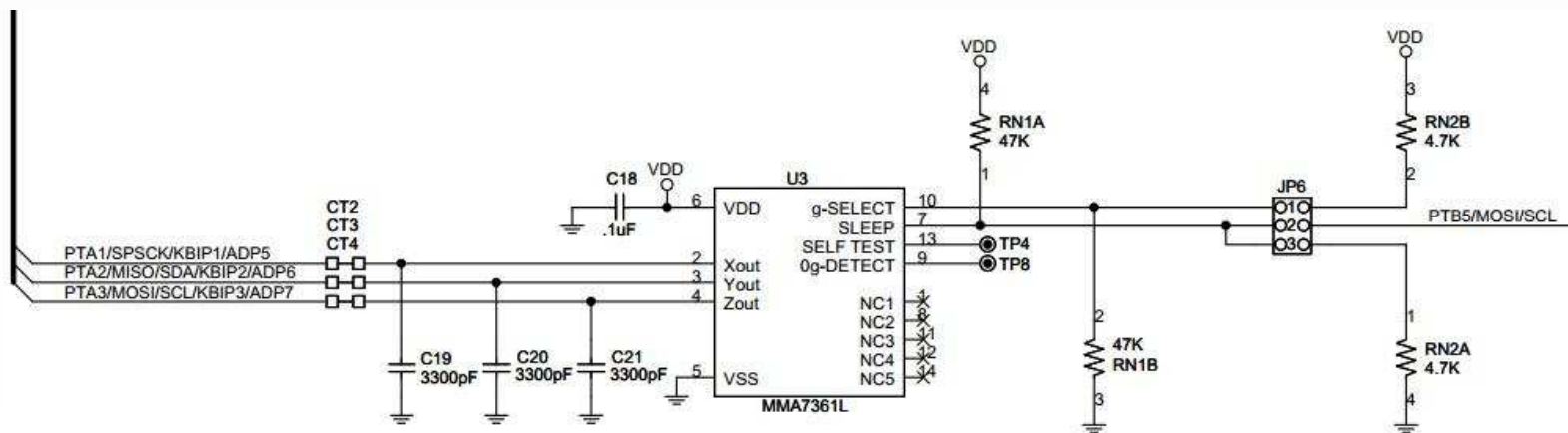
- Sériový port a interrupt – kruhové buffery pro příjem i vysílání

Plán cvičení

1. Úvod a seznámení – laboratoř, bloky uP, opak. C
2. Prostředí CodeWarrior – simulátor + debugger
3. Opakování C algoritmů
4. I/O porty – tlačítka a LEDky
5. Časovač a přerušení
6. Sériový port
7. Procvičení – sériový port, I/O, časovač, přerušení
- 8. A/D převodník + akcelerometr**
9. Využití časovače v režimu PWM
10. Další možnosti využití časovače a přerušení
11. Pokročilé algoritmy
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)



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

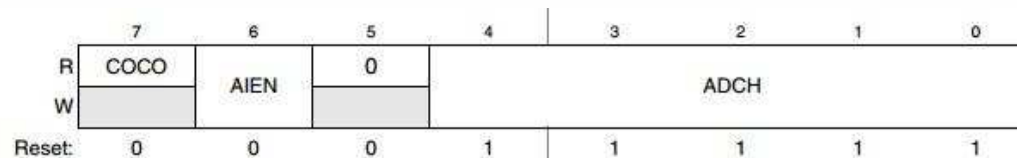


Figure 11-3. Status and Channel Control Register 1n (ADCSC1)

A/D převodník – II

- Převezená 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

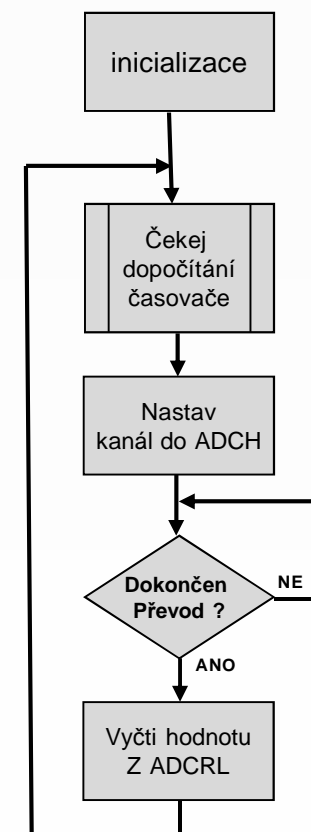
ADCH	Channel	Input	Pin Control
00000	AD0	ADP0	ADPC0
00001	AD1	Reserved	ADPC1
00010	AD2	Reserved	ADPC2
00011	AD3	Reserved	ADPC3
00100	AD4	PTA0/ADP4	ADPC4
00101	AD5	PTA1/ADP5	ADPC5
00110	AD6	PTA2/ADP6	ADPC6
00111	AD7	PTA3/ADP7	ADPC7
01000	AD8	PTA4/ADP8	ADPC8
01001	AD9	PTA5/ADP9	ADPC9
01010	AD10	PTA6/ADP10	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	VREFO	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	VREFH	VREFH	N/A
11110	VREFL	VREFL	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	196.296 kHz
Continuous conversion time	2.12 us; not supported	170.588 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 připojeného 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 Xout = 5, Yout = 6, Zout = 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 vrátí 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

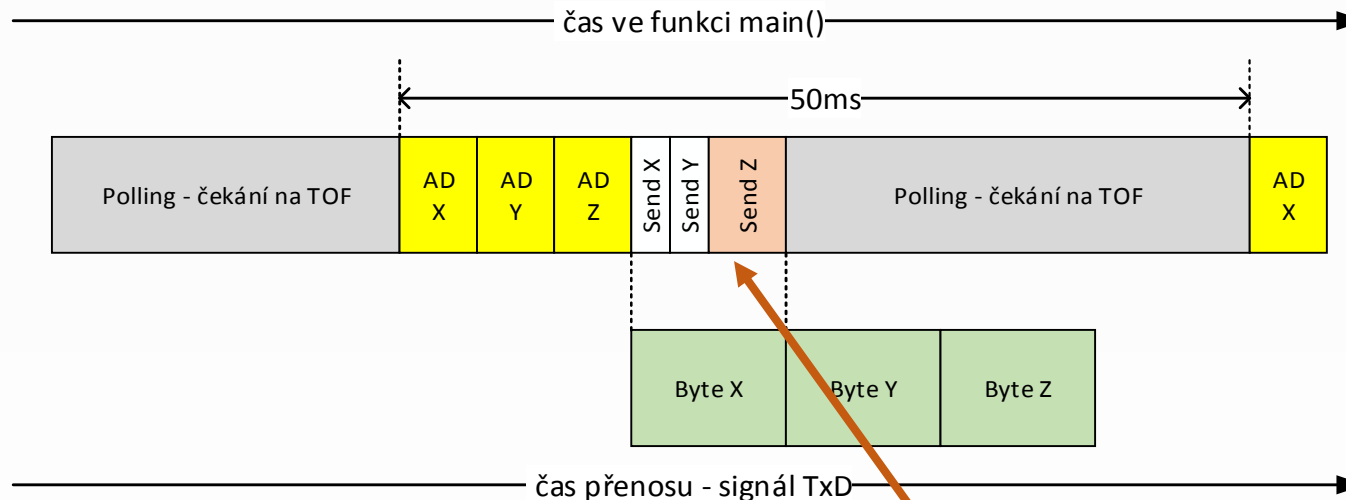
Plán cvičení

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

Nastavení projektu

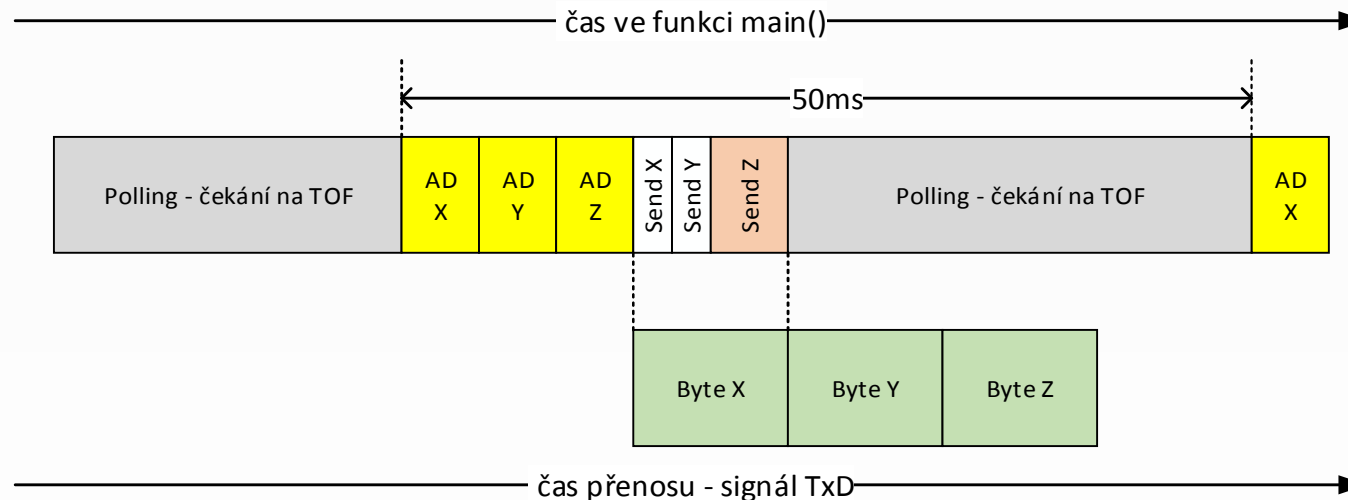
- Využití komponent a znalostí z minulých projektů
 - PTA a PTC porty – připojené LEDs a BUTTONs
 - Sériový port SCI2 – rychlost 19200 nebo 9600
 - Časovač TPM1 – modulo čítač pro opakování 50ms
 - AD převodník ADC – vstupy ADP5-7, 8-bitové převody
 - Jádru CPU – bus-clock 8MHz generovaná z interních 31.25kHz
- Zatím žádné využití přerušení
- Funkce pro odesílání/příjem přes SCI2
 - void uart_send(byte)
 - byte uart_recv(void)
- Funkce pro měření z AD vstupů
 - byte readADC(byte kanal)
- Měření AD z akcelerometru pomocí pollingu
 - Čekám na (modulo) dopočítání TPM1
 - 3x převedu z AD a odešlu "bajty" do PC ke zobrazení
 - Program na Z:\podklady\MPP\SerialView.exe

Časování pollingem



- Veškerý čas procesor stráví ve funkci main
 - Po detekci přetečení se 3x změří
 - Při odesílání se čeká pouze jednou na "doodoslání" – proč ?
 - Pro ověření po vynulování TOF rozsvítit LED1 a po 3 vysílání LED1 zhasnout – můžeme sledovat na osciloskopu šířku pulsu

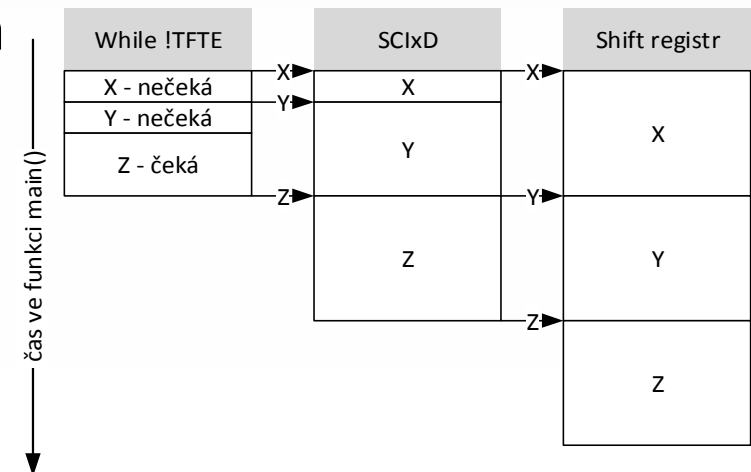
Časování pollingem – II



- Veškerý čas procesor stráví ve funkci main

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

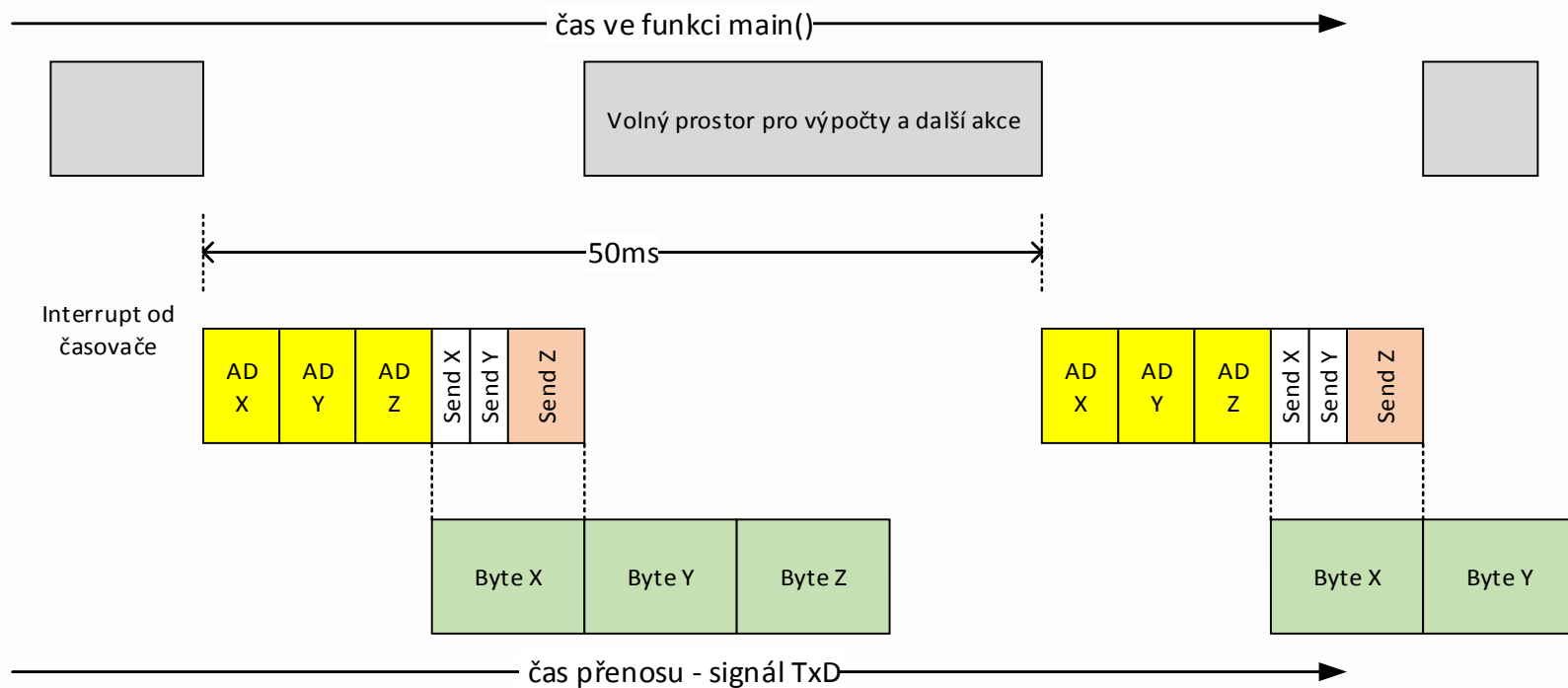
- Ve while se čeká na příznak, že SCIdx uvolněn = po přesunu do shift registru



Přerušení od časovače

- Ve vlastnostech TPM1 povolíme "Overflow Interrupt" a nastavíme jméno funkce
 - Nezapomenout v Projekt – Properties – ProcessorExpert – Generate ISRs povolit
- Soubor Events.c – nová ISR funkce
 - Přesuneme komplet rutinu měření a odesílání do ISR
 - Včetně bliknutí LED1 pro možnost měření
 - Žádné čekání na přetečení (to se dělá "samo")
 - Nezapomenout shodit TOF příznak !!
- V main() tedy pouze prázdná nekonečná smyčka
- LED1, Uart_send a readADC není známa
 - #include "hw_cfg.h"
 - Tělo funkce readADC zatím umístit do Events.c
- Nový modul se sériovou komunikací
 - Serial.c – obsahuje uart_send a uart_recv
 - Na začátku musí obsahovat #include "Cpu.h" – jména registrů apod.
 - Serial.h – obsahuje hlavičky těchto funkcí
 - #include jen v Events.h
- Časově prakticky žádná změna

Časování přerušením – I



Využití přerušení od AD převodníku

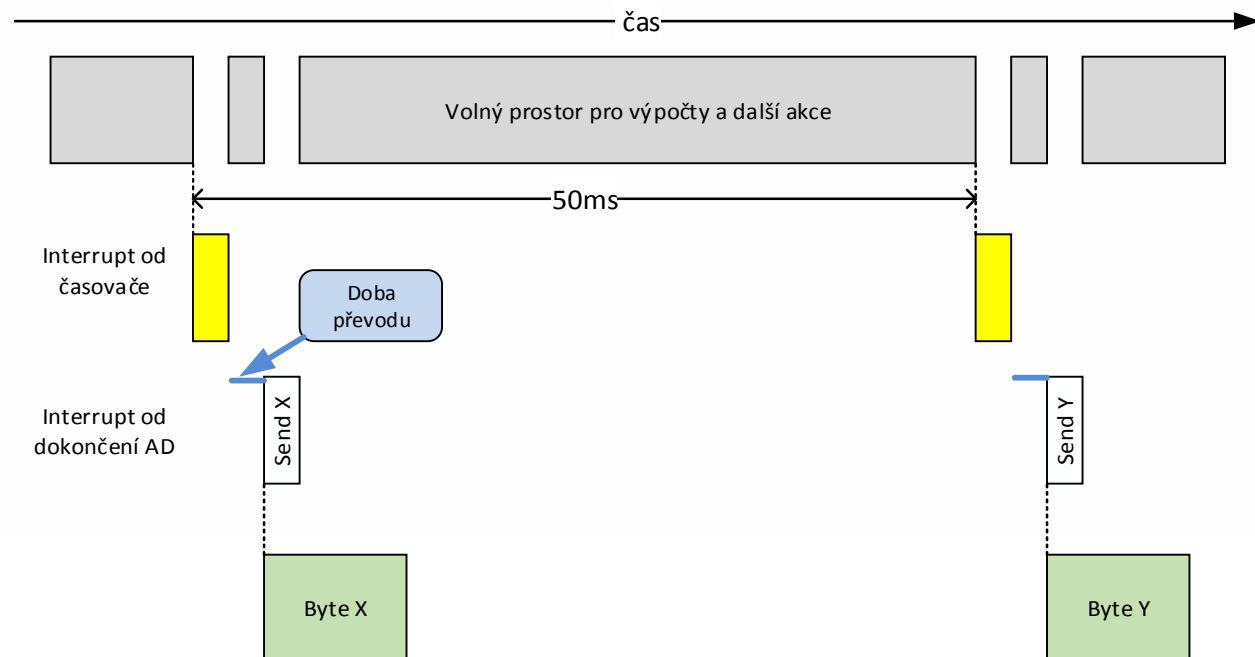
- Každým přerušením od TPM1 se spustí jeden převod
 - Inkrementace čísla AD kanálu v hodnotách 5, 6, 7
 - Použitelná statická nebo globální proměnná
- Povoleno přerušení od ADC
 - V obslužné funkci (je v Events.c) se odešle změřená hodnota
 - Příznak COCO netřeba nulovat, "shodí" se čtením dat

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

    TPM1SC_TOF = 0;
    ADCSC1_ADCH = kanal;

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

```
ISR(ADComplete)
{
    LED1 = !LED1;
    uart_send(ADCRL);
}
```

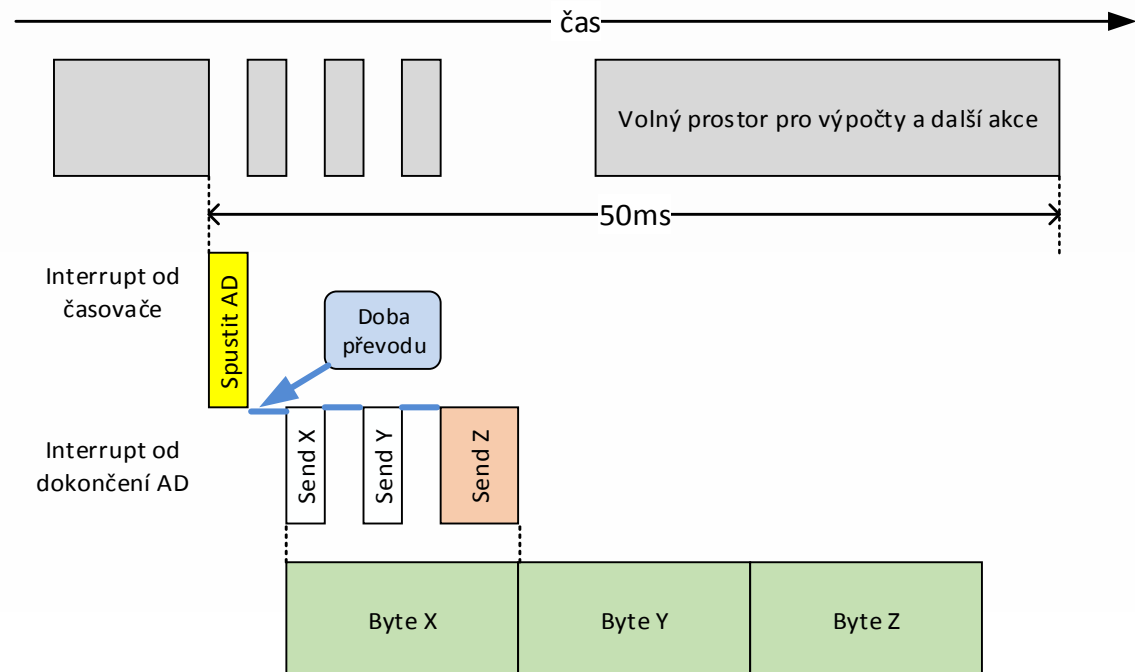


Přerušení od AD převodníku – II

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

```
volatile byte gKanal;  
  
ISR(TimerOver)  
{  
    TPM1SC_TOF = 0;  
  
    LED1 = 0;    // on  
  
    gKanal = 5;  
    ADCSC1_ADCH = gKanal;  
}
```

```
ISR(ADComplete)  
{  
    switch(gKanal)  
    {  
        case 5:    // AD5 - axis X  
            uart_send(ADCRL);  
            gKanal++;  
            break;  
        case 6:    // AD6 - axis Y  
            uart_send(ADCRL);  
            gKanal++;  
            break;  
        case 7:    // AD5 - axis Z  
            uart_send(ADCRL);  
            gKanal = 0x1f;  
            break;  
    }  
  
    if (gKanal > 7)    // konec mereni ?  
        LED1 = 1;    // off  
    else  
        ADCSC1_ADCH = gKanal; // start nxt  
}
```



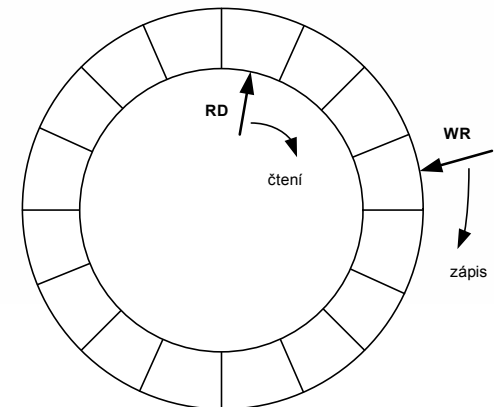
Plán cvičení

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

Problém s čekáním na odeslání

- Pokud dokážeme vygenerovat data rychleji, než se stihnou odesílat, je nutné čekat
 - Měření AD trvá řádově jednotky mikrosek
 - Odeslání jednoho byte řádově milisek (cca 1ms pro 9600b/s)
- Mezi požadavek odesílání bajtů je nutno vložit buffer – tzv. kruhový
 - Funkce `uart_send` jen zapíše do bufferu (= rychlá akce)
 - Nutno inkrementovat "zapisovací ukazovátka" modulo velikost bufferu
 - V případě plného bufferu jsou možnosti
 - Nastavit příznak – hlavní vlákno (main) pak musí zajistit "shození"
 - Počkat na uvolnění = odeslání alespoň jednoho znaku
 - Povolit přerušení od "odeslání"
 - Po uvolnění SCiXD registru (TRDE příznak) se z bufferu vyzvedne nejstarší neodeslaná hodnota
 - Využito přerušení "Transmit request" v SCI2
 - Po posledním bajtu (čtecí ukazovátka "dostihlo" zapisovací) se musí zakázat přerušení
 - Jinak by stále nastávalo přerušení od prázdného datového registru (!!)
- Pro ověření v `main()` budeme blikat LED2
 - POZOR – prostá negace LED2 není atomická !!

```
while(1)
{
    LED2 = 0;
    LED2 = 1;
}
```



```
#define SCI_BUF_SIZE 16
```

```
volatile byte gbSendBuffer[SCI_BUF_SIZE];  
volatile byte gbSendPosW = 0;  
volatile byte gbSendPosR = 0;  
volatile bool IsSendEmpty = TRUE;  
volatile bool IsSendFull = FALSE;
```

```
void uart_send(byte b)  
{  
    if (!IsSendFull)  
    {  
        gbSendBuffer[gbSendPosW] = b;  
  
        gbSendPosW++;  
        if (gbSendPosW >= SCI_BUF_SIZE)  
            gbSendPosW = 0;  
  
        IsSendEmpty = FALSE; // kazdopadne uz neni prazdny buffer  
        SCI2C2_TIE = 1;      // kazdopadne enable interrupty  
  
        // indikace plného bufferu = cekam na uvolneni:  
        while ((gbSendPosW == gbSendPosR) && !IsSendEmpty)  
            LED4 = 0;  
  
        LED4 = 1;  
  
        /* alternativni indikace plného bufferu priznakem  
        if (gbSendPosW == gbSendPosR)  
        {  
            // mel bych zapisovat na misto,  
            // kde se jeste ma cist ?  
            IsSendFull = TRUE;  
        }  
        */  
    }  
}
```

```
ISR(TxRequest)  
{  
    uart_txempty();  
}
```

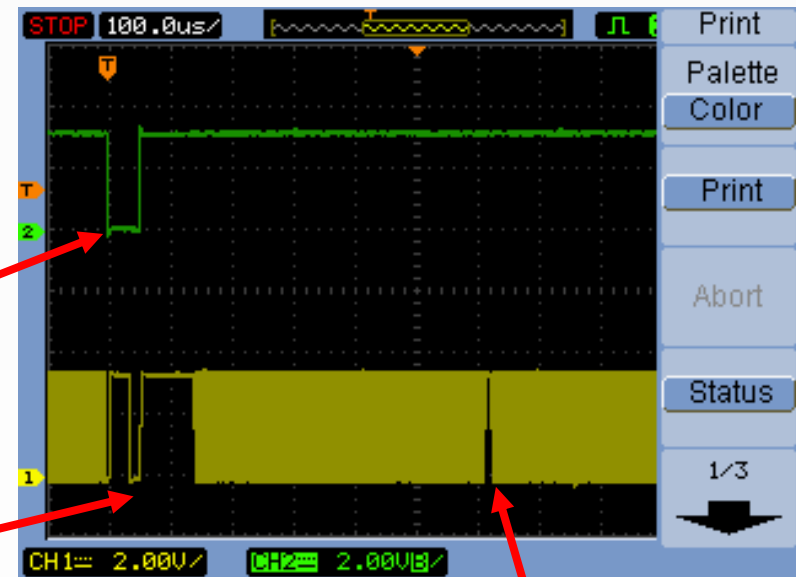
Kruhový buffer - implementace

- Je-li místo, zapíše se do bufferu
- Buffer už není prázdný, povolí se inty
 - Pokud byl předtím buffer prázdný, TRDE je nastaven a ihned se vyvolá přerušování
- Prázdné vyčtení SCIS1 registru shodí případné příznaky
- Po posledním odeslaném bajtu (buffer prázdný) se musí zakázat interrupty (TRDE zůstává na 1)
- V interruptu se volá funkce ze serial.c – všechna data jsou v jednom modulu
- **Volatile** – data používaná v přerušování i "hlavním vlákne"

```
void uart_txempty(void)  
{  
    if (!IsSendEmpty)  
    {  
        (void)SCI2S1;  
        SCI2D = gbSendBuffer[gbSendPosR];  
  
        gbSendPosR++;  
        if (gbSendPosR >= SCI_BUF_SIZE)  
            gbSendPosR = 0;  
  
        if (gbSendPosR == gbSendPosW) // ukazovatka se dobehla  
            IsSendEmpty = TRUE;      // prazdny buffer  
    }  
    else  
        SCI2C2_TIE = 0; // uz neni co vysilat, vypnout Tx inty  
}
```

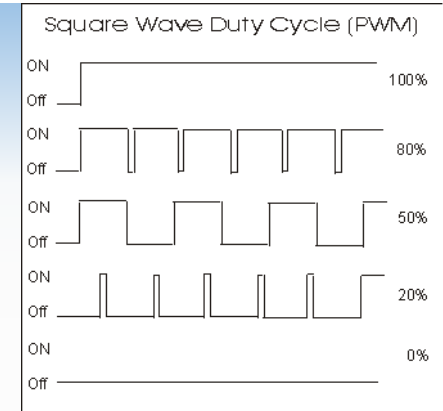
Časové průběhy

- 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)



Generování PWM

- PWM = pulsně šířková modulace
 - Nejjednodušší číslíkové 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

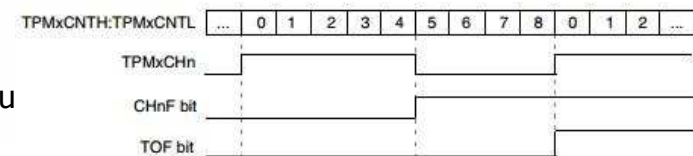


Figure 17-3. High-true pulse of an edge-aligned PWM

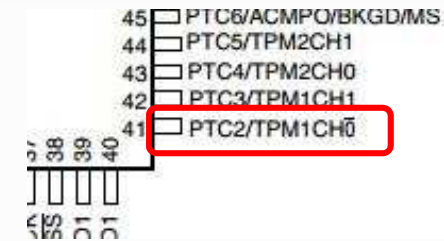
Přehled režimů čítačů

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)

Generování PWM podle ADC

- Cíl – podle náklonu desky měnit frekvenci PWM na pinu připojeném k "pípáku" (buzzer)
 - Na desce připojen k PTC2
 - Nutno použít TPM1 a jeho CH0
 - Podle ADC (X osa ?) měnit frekvenci
- K časování měřících 50ms použít TPM2
 - Nastavit TPM2 na 50ms, povolit přerušení
 - Kód pro spuštění měření A/D přesunout do přerušení od TPM2
- Překonfigurovat TPM1
 - Vypnout přerušení od TPM1
 - Přidat kanál (vznikne CH0) a zapnout jeho I/O pin
 - Nastavit modulo registr na 249 (odpovídá téměř max. hodnotě z ADC)
 - Nastavit value-registr (položka "Channel compare") na "polovinu" = střída PWM 50%
 - Nastavit předděličku tak, aby generovaná perioda byla 1ms (= 1kHz)
- Zrušit v PTC bit 2 (=disable), aby jej mohl využívat TPM1CH0
 - Zrušit kódů všechny 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)
 - Spouštět TPM1 podle stavu tlačítka BUTTON1 (v klidu nepískat !!!)
 - Nejlépe pomocí bitu TPM1 v registru SCGC1



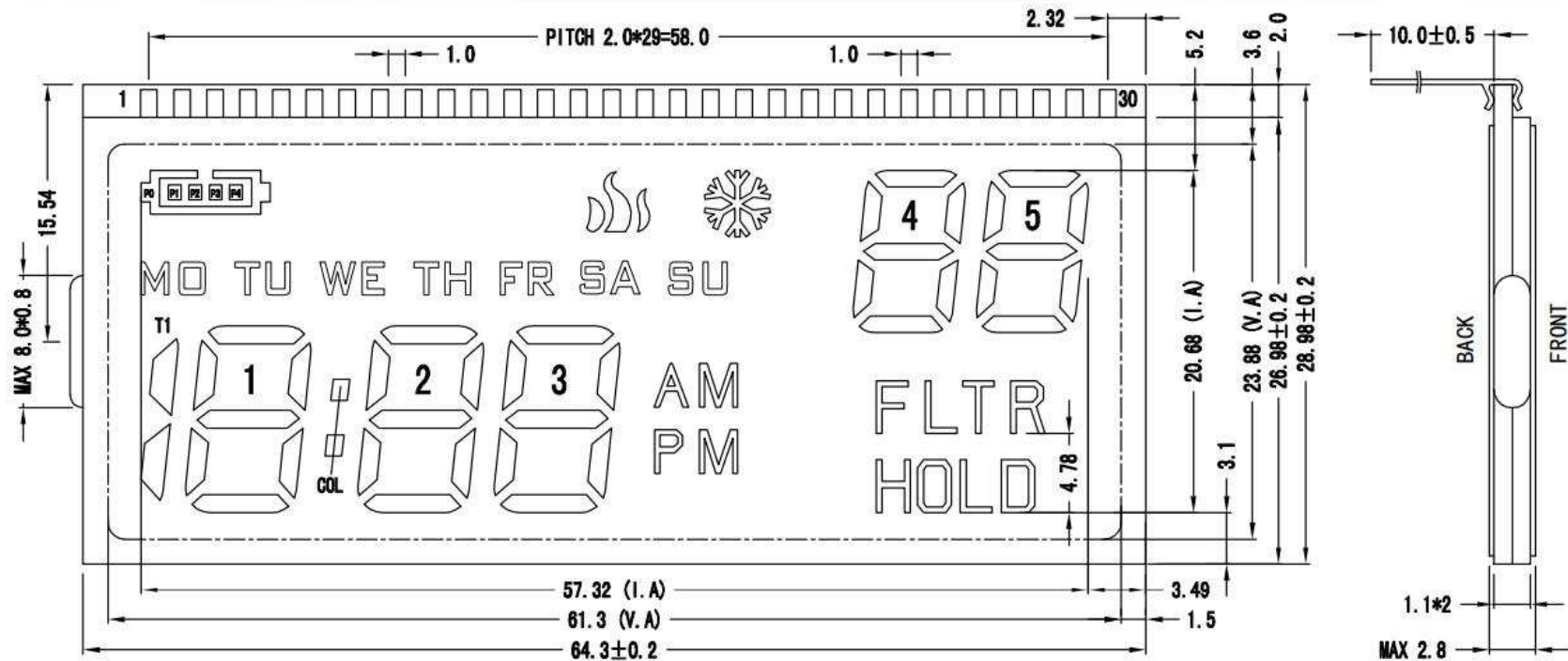
Plán cvičení

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

Semestrální práce - úvod

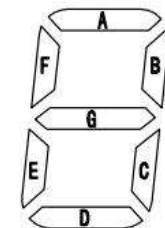
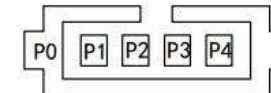
- Spojení jednotlivých bloků do funkční aplikace
- Zatím probrané
 - Sériový port
 - A/D převodník
 - Časovač
 - Jádru procesoru
 - I/O porty – LED-ky a tlačítka
- Samostatně doplnitelné bloky
 - PWM režim čítače – viz minulé "slajdy"
 - LCD displej
 - Podklady **Z:\podklady\MPP** – datasheet (TWR_LCD_GLASS_SPECIFICATION.pdf)
 - 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)

LCD



PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
COM1	MO	P0	P3	P4	TU	1B	1C	1D	WE	TH	2B	2C	2D	FR	SA	SU	3B	3C
COM2	T1	P1	P2		1A	1F	1G	1E	COL	2A	2F	2G	2E	3A	3A	3A	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

S-Tek Displays Cleveland, Ohio 44146			
6			
5			
4			
3			
2	MODIFY LOGIC AND ICON	18/06-08 L. M. L.	DESIGN BY Q B 26/05-08
1	MODIFY LOGIC	28/05-08 R. Y. Z.	CHECKED BY
VER.	MODIFY CONTENTS	DATE	DESIGN APP BY

PART NO. **GD-5360P**

DESIGN BY Q B 26/05-08

APPLICATION:
☐ Instrument ☐ Audio Equipment ☐ Watch/Clock
☐ Telephone ☐ Automobile ☐ Other
☐ Other Appliances ☐ Air-condition

SAMPLE: ☐ Yes ☒ No

SHEET 1 OF 1

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)
 - Generátor kmitočtu (hlídá max. a min. rozumnou hodnotu)
 - Generátor PWM (střída 0-100, ověřit na LED a osciloskopu)
 - Zobrazit zrychlení na LCD (výběr vstupu X/Y/Z pomocí terminálu)
- Náměty záchranné ("na trojku")
 - K programu s kruhovým bufferem odesílajícím data do PC doplnit přepínání formátu dat binární/textové
 - Tlačítkem nebo повеlem z terminálu
 - Možnosti textového výstupu
 - Vlastní funkce
 - Funkce **sprintf** ze stdio
 - Funkce **printf** a vytvořené **TERMIO_PutChar** volající naše `uart_send` (viz. závěr 6. cv)