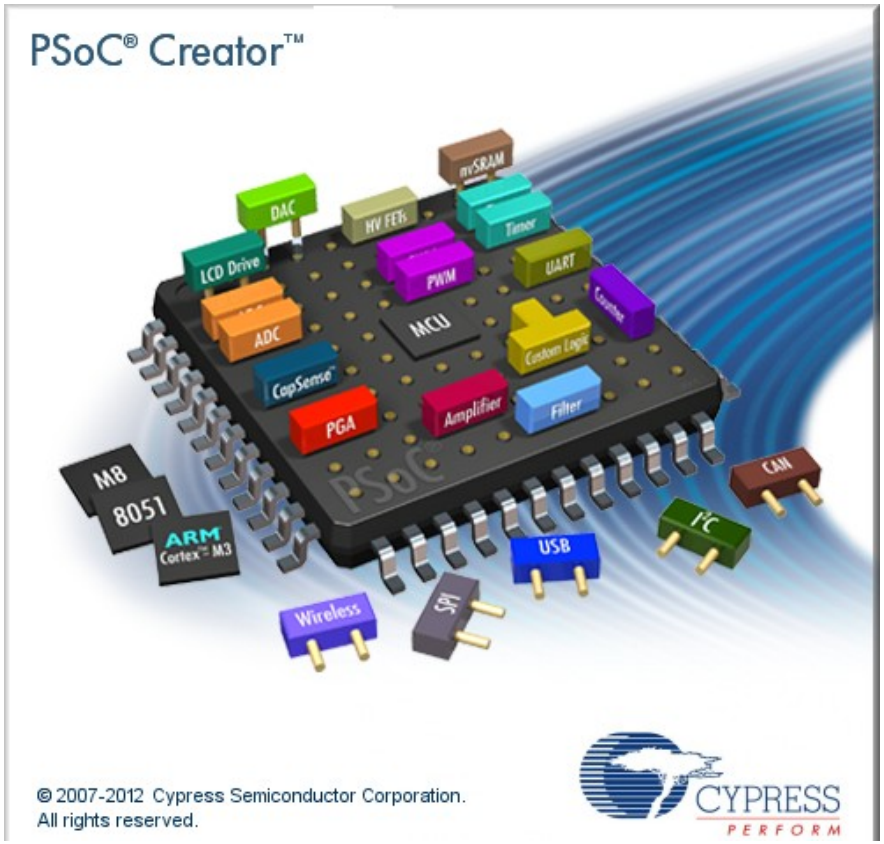
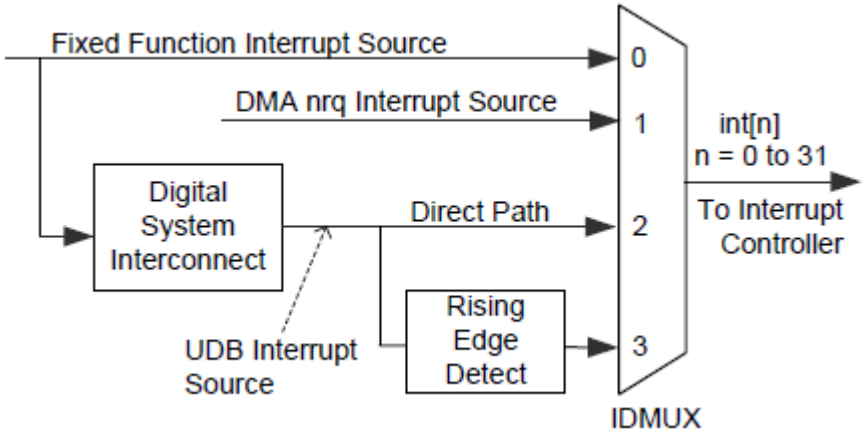
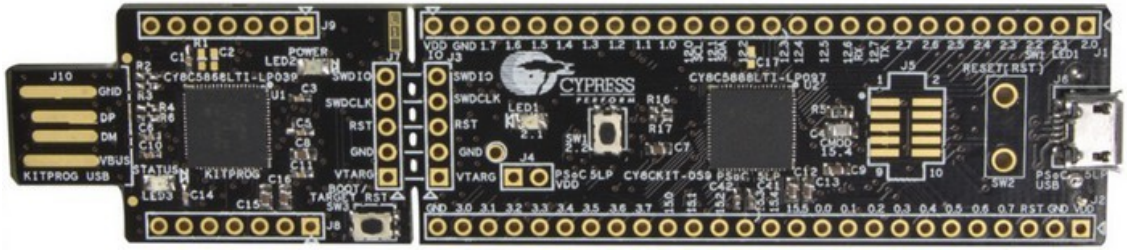
Újrakonfigurálható eszközök







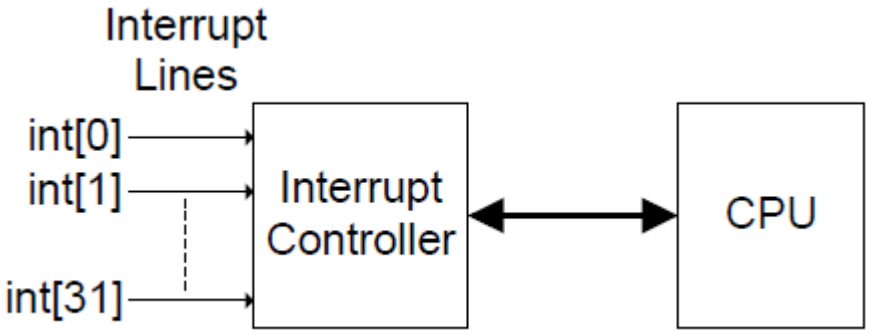
14. Cypress PSOC 5LP programmegszakítások

Felhasznált irodalom és segédanyagok

- Cypress: CY8C58LP FamilyDatasheet
- Cypress: PSOC 5LP Architecture Technical Reference Manual)
- Cypress: CY8CKIT-059 Prototyping Kit Guide
- Cypress: AN77759: Getting Started with PSoC®5LP
- Cypress: PSoC®Creator™ User Guide
- Cypress: PSoC® 5LP Registers TRM
- Yuri Magda: Cypress PSoC 5LP Prototyping Kit Measurement Electronics
- Cserny István: PSOC 5LP Mikrokontrollerek programozása
- Cypress: AN54460 - PSoC® 3 and PSoC 5LP Interrupts

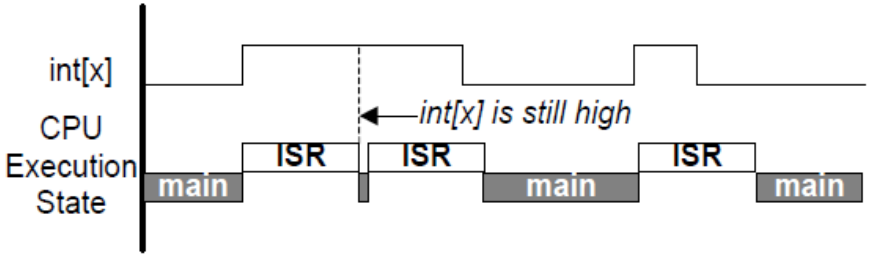
A megszakítási rendszer jellemzői

A PSoC mikrovezérlők megszakítási rendszere 16 kivételt és 32 db megszakítási vonalat tud kezelni.

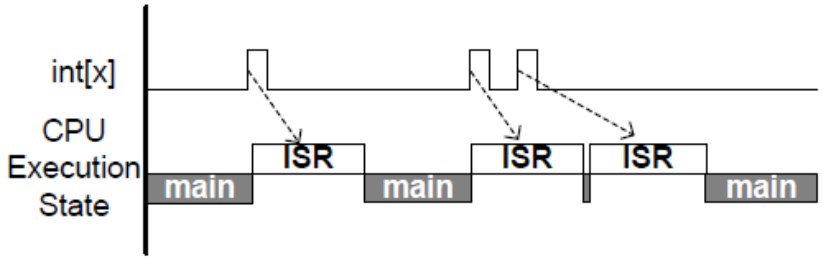


A PSoC 5LP megszakítási rendszere:

- Vektoros
- Prioritásos (8 szintű) ← Az alacsonyabb prioritású kiszolgálás megszakítható
- Egymásba ágyazott
- Konfigurálható vektorcímek ← A kiszolgáló eljárás címe dinamikusan változtatható
- Flexibilis megszakítási források ← A megszakítási források nincsenek mereven hozzárendelve a megszakítási vonalakhoz



Szintvezérelt megszakítások

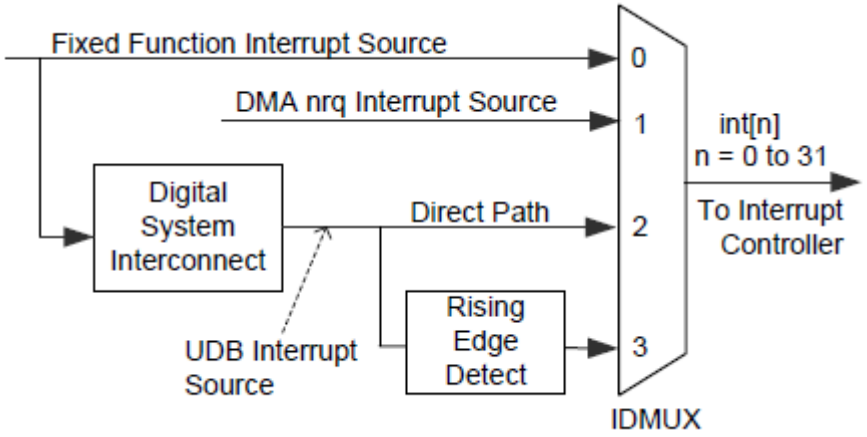


Élvezérelt megszakítások

A megszakítási rendszer jellemzői

Minden megszakítási vonal három forrásból kaphat jelet:

- Fix funkciójú periféria
- DMA csatorna
- UDB modul



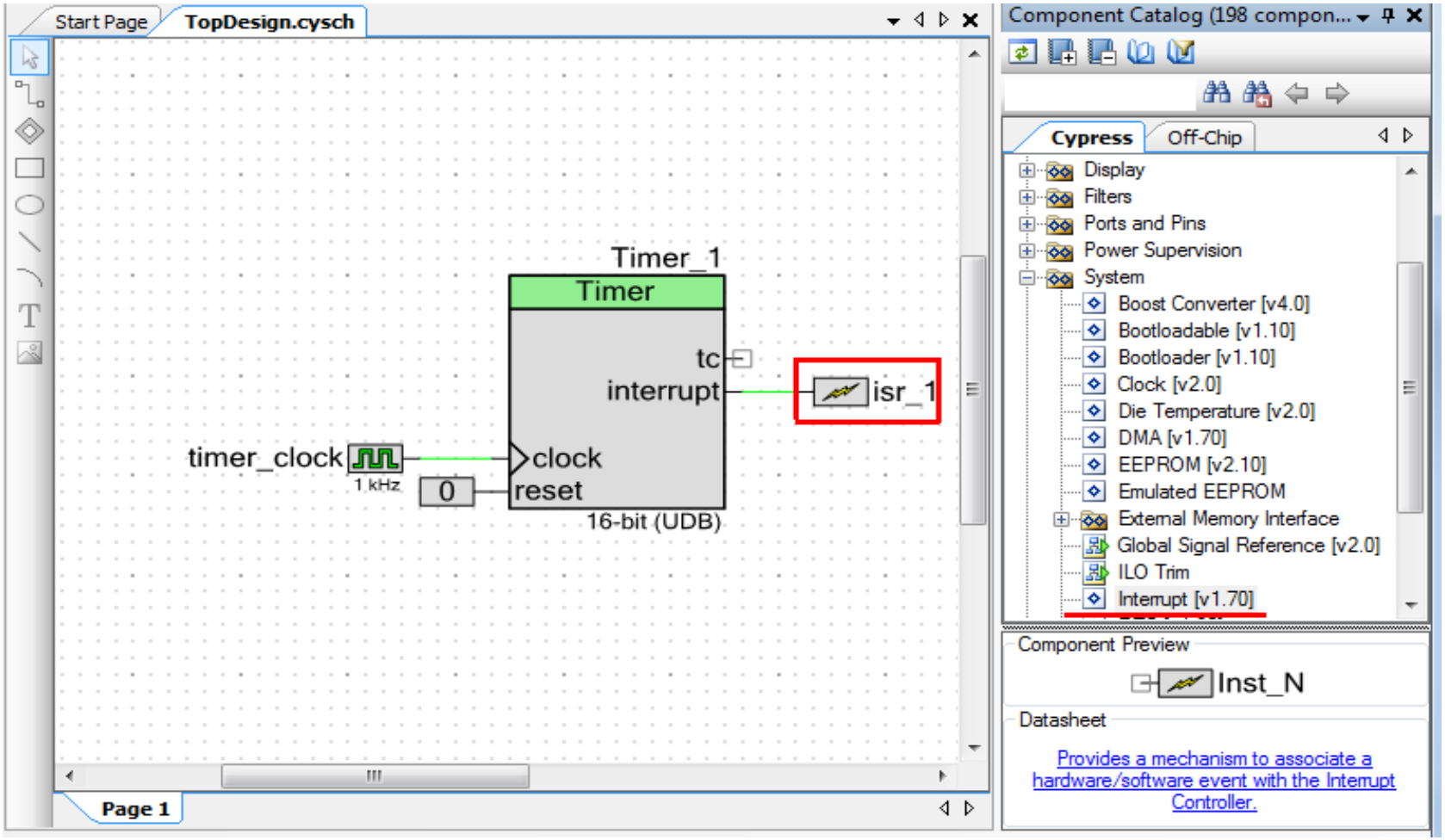
Fix funkciójú megszakítási források:
PICU (Port Interrupt Control Unit), LVD (Low Voltage Detect), RTC, SleepTimer, I2C, CAN, USB, FF Timer/Counter/PWM, Szegmenses LCD, Delta-Sigma ADC)

DMA megszakítások
A DMA adatátvitel végén élvezérelt megszakításkérő jel generálható.

UDB megszakítási források
Minden digitális jel beállítható megszakítási jelforrásként, s a digitális összekötési rendszer (DSI) segítségével valamelyik megszakítási vonalra köthető.

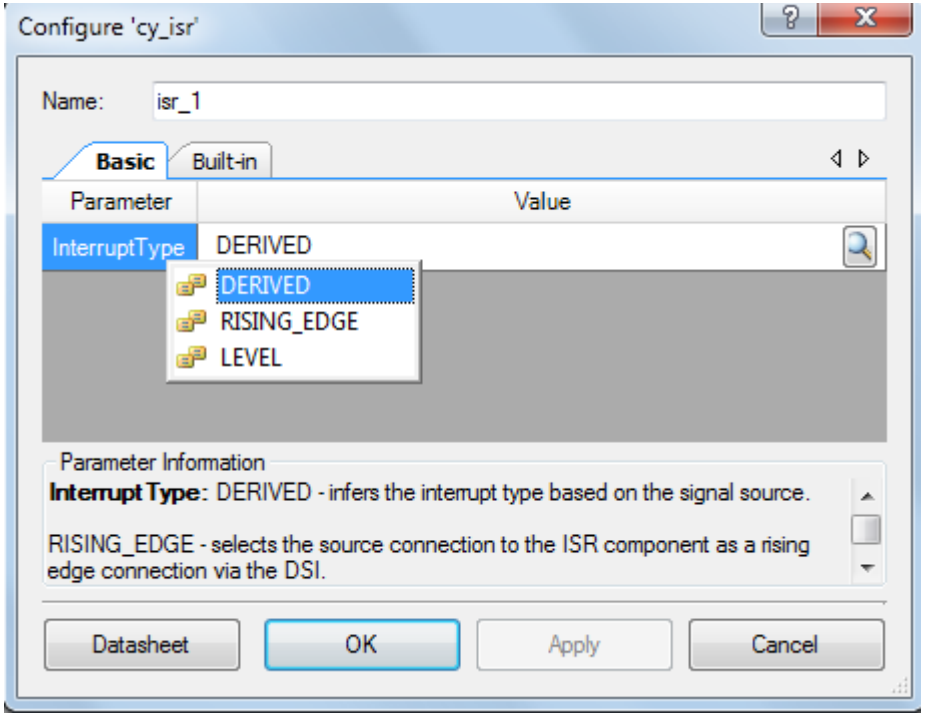
PSoC Creator támogatás

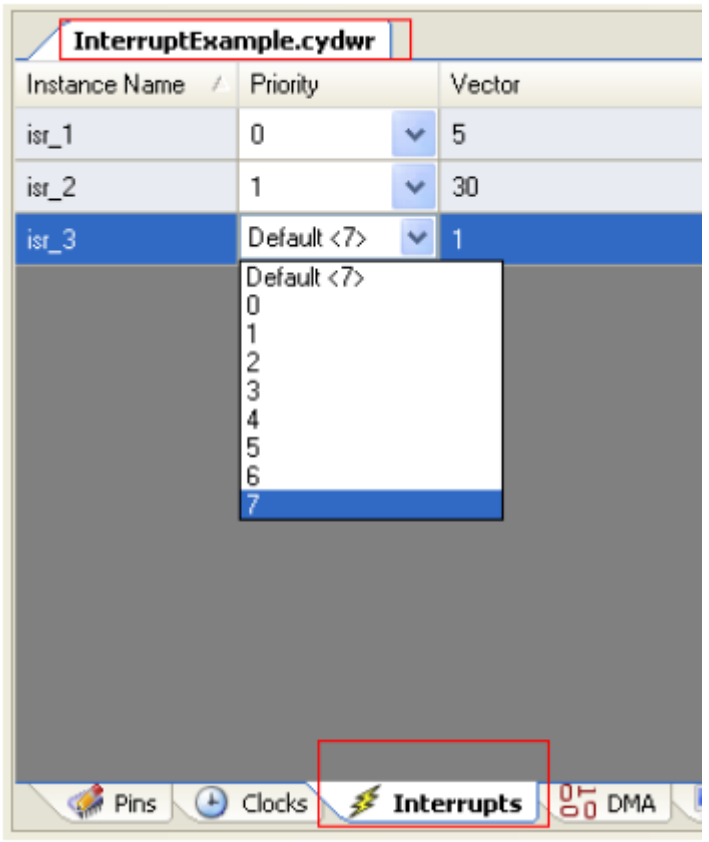
- Az ISR komponens bármelyik digitális jelhez hozzárendelhető
- Hardveresen triggerelt megszakítást definiál
- Szoftveres API-t biztosít a kezeléséhez



Az ISR komponens konfigurálása

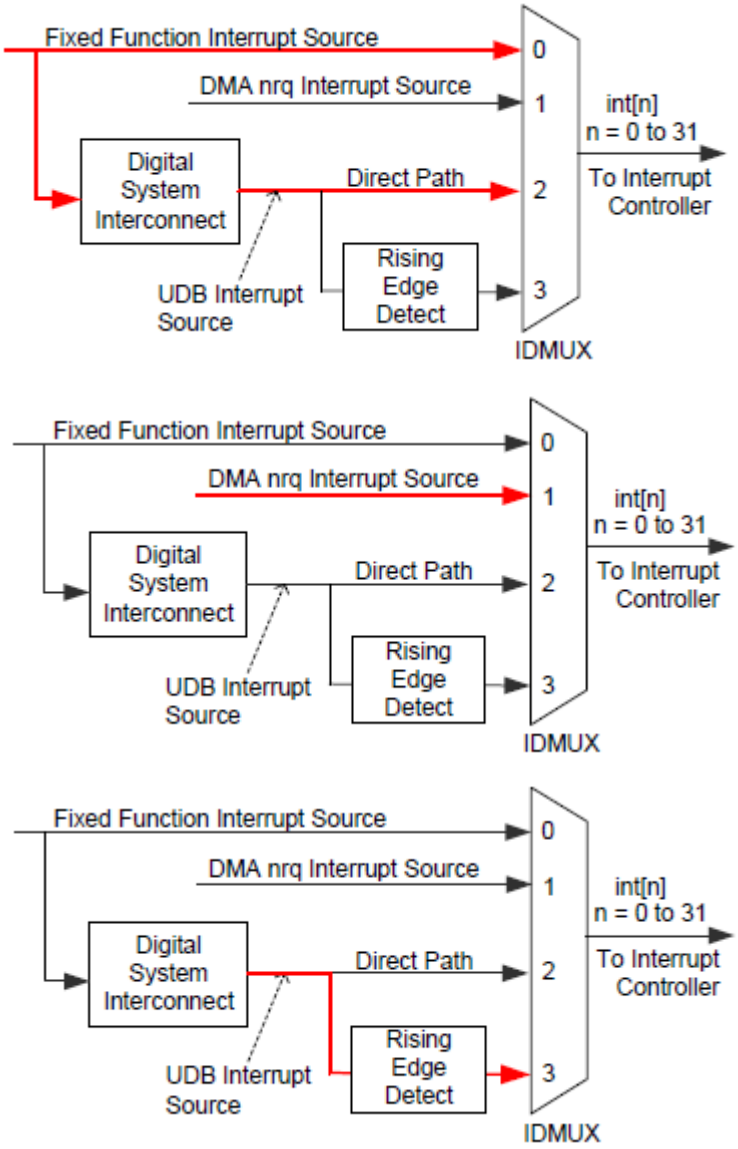
- A megszakítás típusát (él, szint, vagy származtatott) és a
- megszakítás prioritását (0 – 7, ahol 0 a legmagasabb szint) lehet beállítani





Mit jelent a „Derived” útvonalválasztás?

- A *derived* megszakítási mód választása automatikusan konfigurálja a bemeneti multiplexert a csatlakozó megszakítási forrás típusa szerint
- **Fix funkciójú eszközök**
  Szintvezérelt megszakításkezelés, dedikált vonalon, vagy DSI összeköttetésen keresztül.
- **DMA megszakítási források**
  Élvezérelt megszakításkezelés, dedikált vonalakon keresztül.
- **UDB komponensek:**
  Élvezérelt megszakításkezelés



Alkalmazásprogramozói függvények

- Az **ISR** komponens kezeléséhez a **PsoC Creator** az alábbi alkalmazásprogramozói függvényeket biztosítja

Function	Description
ISR_Start()	Sets up the interrupt to function.
ISR_StartEx()	Sets up the interrupt to function and sets address as the ISR vector for the interrupt.
ISR_Stop()	Disables and un-configures the interrupt.
ISR_Interrupt()	The default interrupt handler for ISR.
ISR_SetVector()	Sets address as the new ISR vector for the Interrupt.
ISR_GetVector()	Gets the address of the current ISR vector for the interrupt.
ISR_SetPriority()	Sets the priority of the interrupt.
ISR_GetPriority()	Gets the priority of the interrupt.
ISR_Enable()	Enables the interrupt to the interrupt controller.
ISR_GetState()	Gets the state (enabled, disabled) of the interrupt.
ISR_Disable()	Disables the interrupt.
ISR_SetPending()	Causes the interrupt to enter the pending state, a software method of generating the interrupt.
ISR_ClearPending()	Clears a pending interrupt.

A megszakítások kiszolgálása

- A megszakítást kiszolgáló eljárást többféle módon is megadhatjuk:
  1) A PSoC Creator által generált alapértelmezett ISR_Interrupt kiszolgáló eljárás bővítésével az ISR.c állományban.
  2) Saját kiszolgáló eljárás definiálásával.
  3) A PSoC Creator által nem támogatott megszakításoknál: Saját kiszolgáló eljárás definiálásával és a megszakítási vektor feltöltésével.
- Bármelyik módszert választjuk, a főprogramban gondoskodni kell a megszakítások inicializálásáról és engedélyezéséről. Például:

1.

```
ISR_Start();
CyGlobalIntEnable;
```

Az **ISR_Interrupt** eljárást ki kell egészíteni!

2.

```
CY_ISR_PROTO(MyISR);
ISR_StartEx(MyISR);
CyGlobalIntEnable;
```

3.

```
CY_ISR_PROTO(MyISR);
CyIntSetSysVector(VECNUM,MyISR);
CyGlobalIntEnable;
```

A **MyISR** saját eljárást definiálnunk is kell!

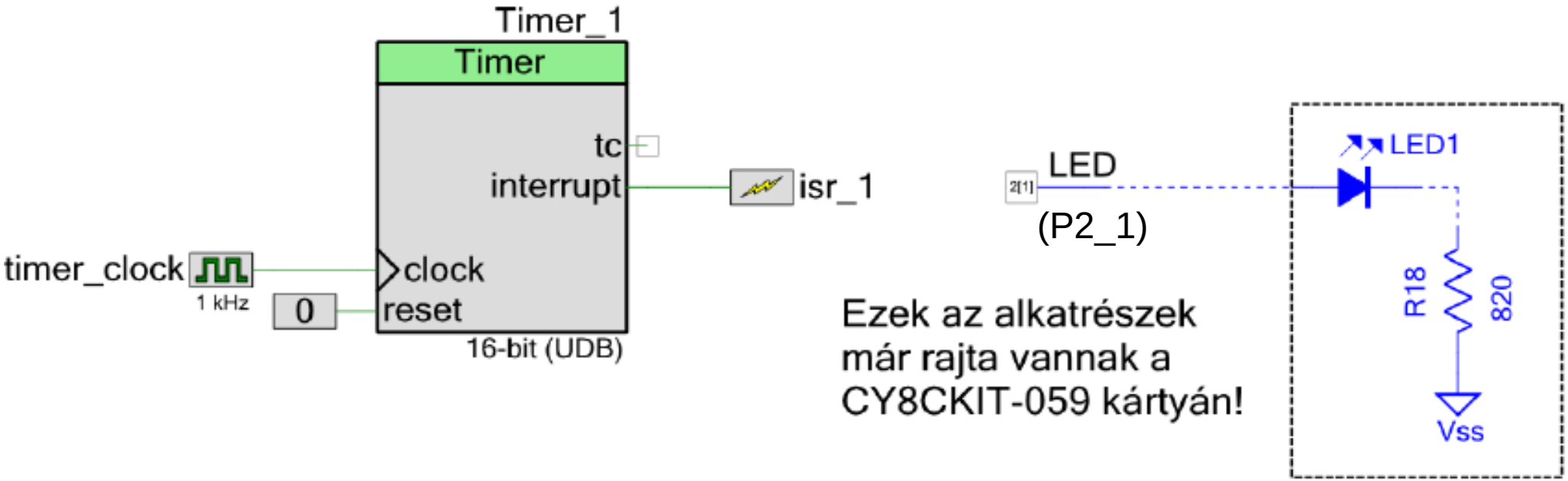
Mintaprogramok

A példaprogramokat az AN54460 alkalmazási mintapéldából vettük. Mindegyik projektet adaptálni kellett a **CY8CKIT-059** kártyához.

- **A_InterruptExample:**
  Ledvillogtatás periodikus (1 Hz) Timer megszakításokkal. Egyszerű példa az alapértelmezett megszakításkiszolgáló eljárások használatára
- **B_PICU**
  LED ki/bekapcsolása nyomógombbal. Példa a PICU megszakítás és saját megszakításkezelő függvény használatára
- **D_SysTick**
  Periodikus (2 Hz) megszakítások keltése az ARM Cortex-M3 CPU beépített időzítőjével. Példa a **PSoC Creator** által nem támogatott megszakítás használatára.

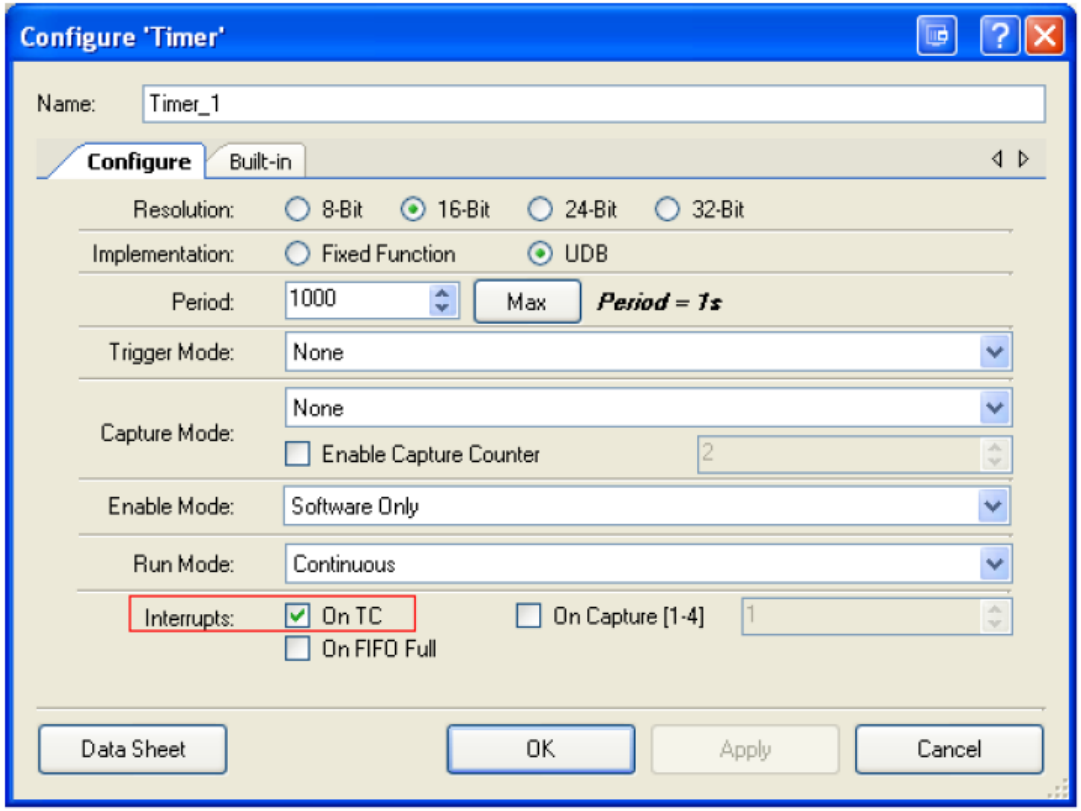
Az A_InterruptExample projekt

- Ebben a projektben egy számláló segítségével periodikusan megszakításokat keltünk (1 Hz frekvenciával)
- A megszakításokban a kártyára szerelt LED (P2_1 kivezetéshez csatlakozik) állapotát átbillentjük
- A LED így 2 másodpercenként kerül azonos állapotba



A Timer komponens konfigurálása

- A Timer komponens egy visszaszámláló.
- TC (terminal count) az az esemény, amikor a számláló elérte a nullát, és újratöltődik a következő visszaszámláláshoz.
- Programmegszakítást keltünk minden TC eseménynél. A megszakítások gyakoriságát a bemeneti órajel frekvenciája (itt 1 kHz) és a Timer Periódus értéke (itt 1000 órajel ciklus) szabja meg.



Megjegyzés: Az ISR komponens konfigurálását már bemutattuk a 6. oldalon

A megszakítás kiszolgálása (1. módszer)

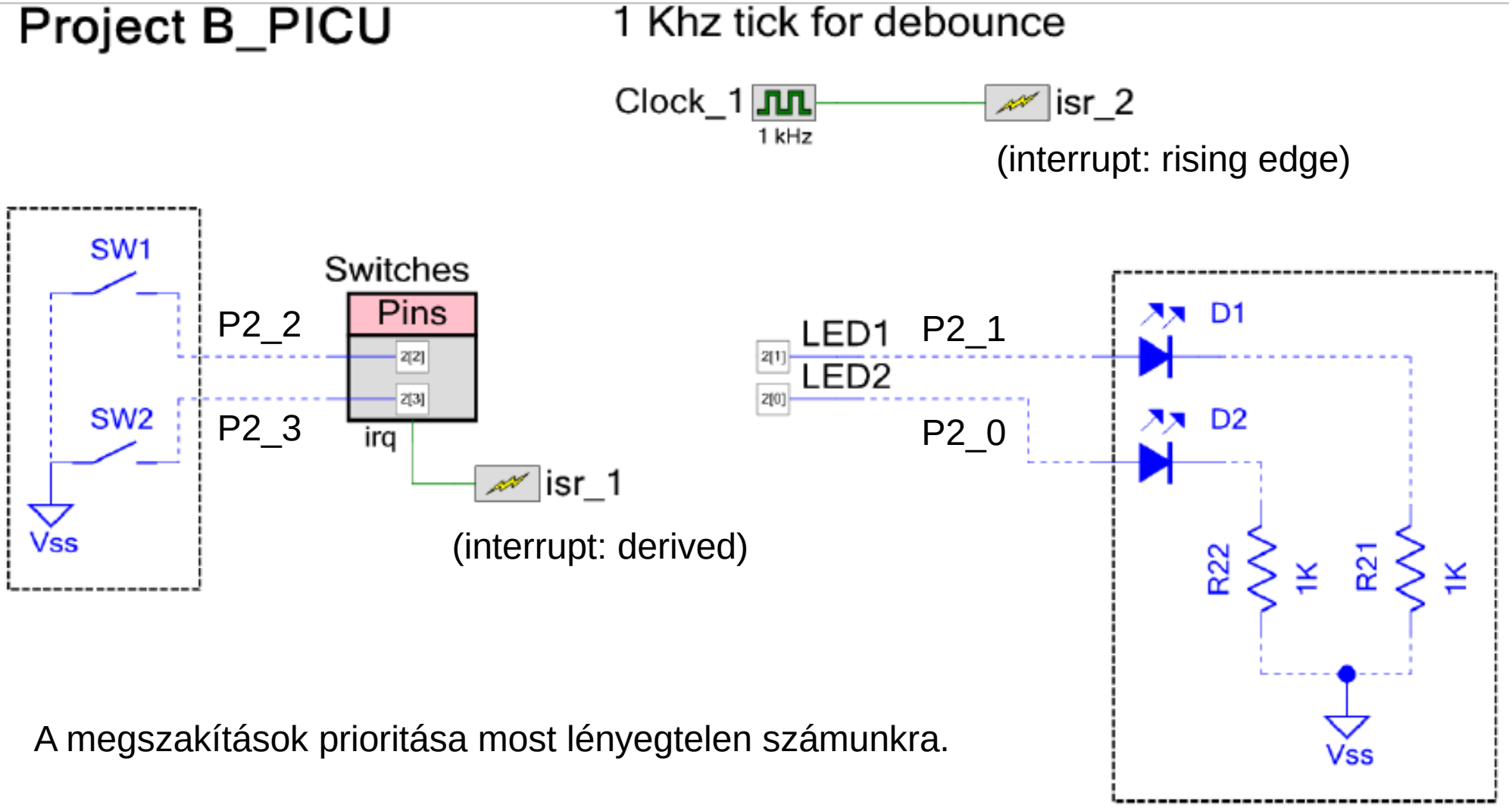
- Itt most az alapértelmezett kiszolgáló eljárást használjuk, amelyet a **PSoC Creator** a Build → Create Application menüpontban hoz létre az **isr_1.c** állományban. Saját sorainkat csak a megjelölt helyeken adjuk hozzá (csak ezek a tartalmak őrződnek meg a projekt újragenerálása során)!

```
/*******************************************************
 *  Place your includes, defines and code here
 *******************************************************/
/* `#START isr_1_intc` */
#include "Timer_1.h"    // Timer Component header file
#include "LED.h"        // LED Pin Component header file
/* `#END` */
```

```
/*******************************************************
 *  Place your Interrupt code here
 *******************************************************/
CY_ISR(isr_1_Interrupt)
{
    /* `#START isr_1_Interrupt` */
    Timer_1_ReadStatusRegister(); // Clear interrupt flag
    LED_Write(~LED_Read());       // Toggle LED state
    /* `#END` */
}
```

B_PICU projekt

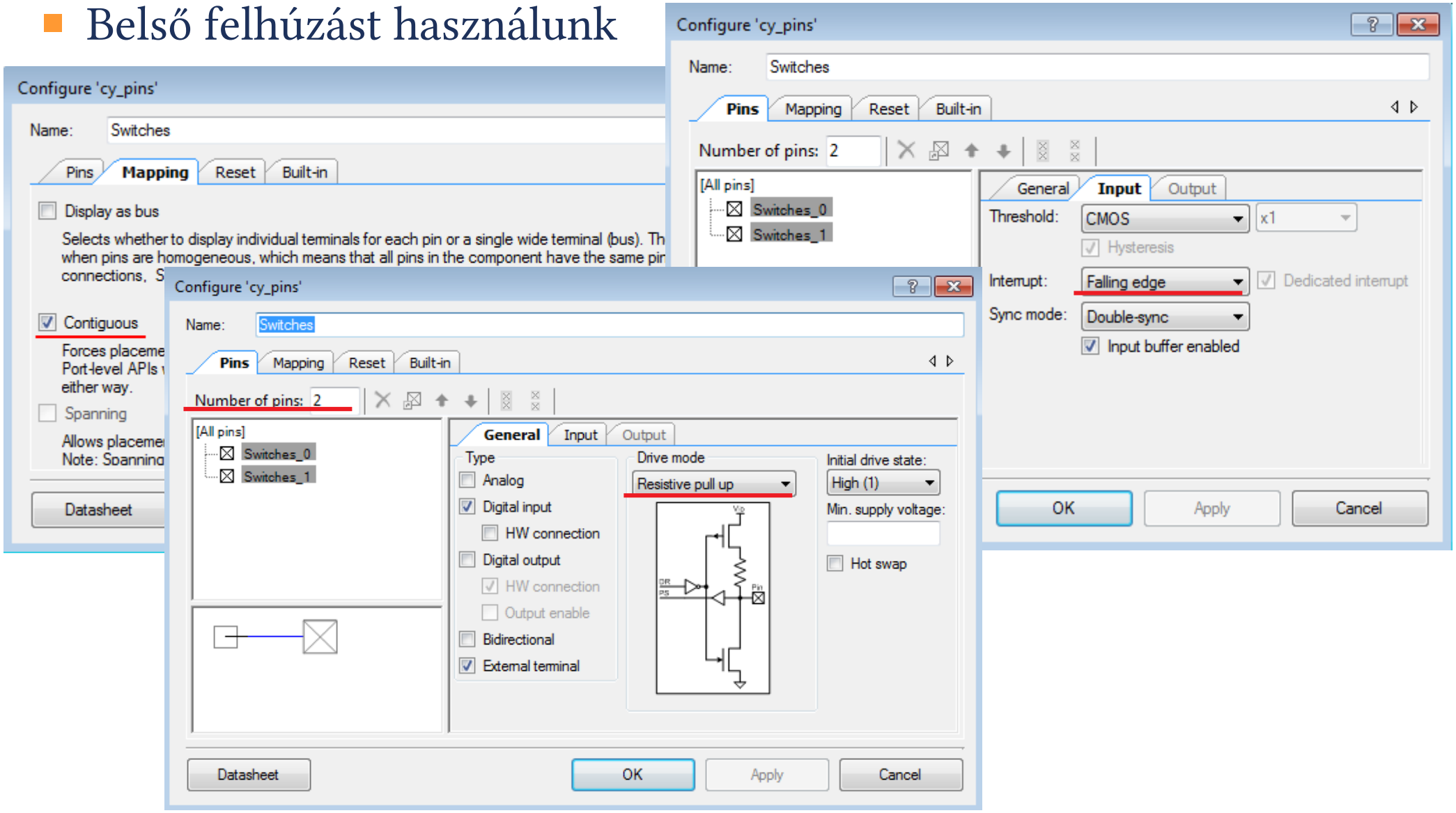
- Két nyomógombbal kapcsolgatunk két LED-et (megszakításban). Pergésmentesítés 1 kHz-es Timer megszakítás felhasználásával…



A megszakítások prioritása most lényegtelen számunkra.

A nyomógomb bemenetek konfigurálása

- A két digitális bemenetet (P2_2 és P2_3) egységbe foglaljuk
- Belső felhúzást használunk



main.c

- Ebben a projektben saját megszakításkiszolgáló függvényeket használunk, amelyeket az InterruptRoutines.c állományban definiálunk
- A főprogramban csak az elindítást és hozzárendelést kell elvégezni

```
#include <project.h>

/* Header file containing the custom ISR prototypes */
#include "InterruptRoutines.h"

int main() {
    /* Initialize the two custom defined ISRs */
    isr_1_StartEx(PICU_ISR);
    isr_2_StartEx(Tick_ISR);

    CyGlobalIntEnable; /* Enable global interrupts. */

    for(;;)
    {
        /* Do nothing in the main loop; code to do something
        is in the ISRs */
    }
}
```

InterruptRoutines.h

```
#ifndef INTERRUPT_ROUTINES_HEADER
#define INTERRUPT_ROUTINES_HEADER

/* including project.h gives access to all component APIs */
/* and other generated source files */
#include <project.h>

/* defines the debounce time in milliseconds, max value is 255 */
#define  DEBOUNCE_TIME  50

/* ISR function prototype declarations */
CY_ISR_PROTO(Tick_ISR);
CY_ISR_PROTO(PICU_ISR);

#endif
```

InterruptRoutines.c – Tick_ISR

```
#define  SW1_MASK  1
#define  SW2_MASK  2
#define  TIMED_OUT  0

static volatile uint8 CYDATA switch_1_timeout;
static volatile uint8 CYDATA switch_2_timeout;

/* Function Name: Tick_ISR
 * Summary: Interrupt once per millisecond. If timeout, toggle LED.
 */
CY_ISR(Tick_ISR) {
    if(switch_1_timeout != 0) {
        if(--switch_1_timeout == TIMED_OUT) {
            if((Switches_Read() & SW1_MASK) == 0) {
                LED1_Write(~LED1_Read()); /* toggle the LED */
            }
        }
    }

    if(switch_2_timeout != 0) {
        if(--switch_2_timeout == TIMED_OUT) {
            if((Switches_Read() & SW2_MASK) == 0) {
                LED2_Write(~LED2_Read()); /* toggle the LED */
            }
        }
    }
}
```

InterruptRoutines.c – PICU_ISR

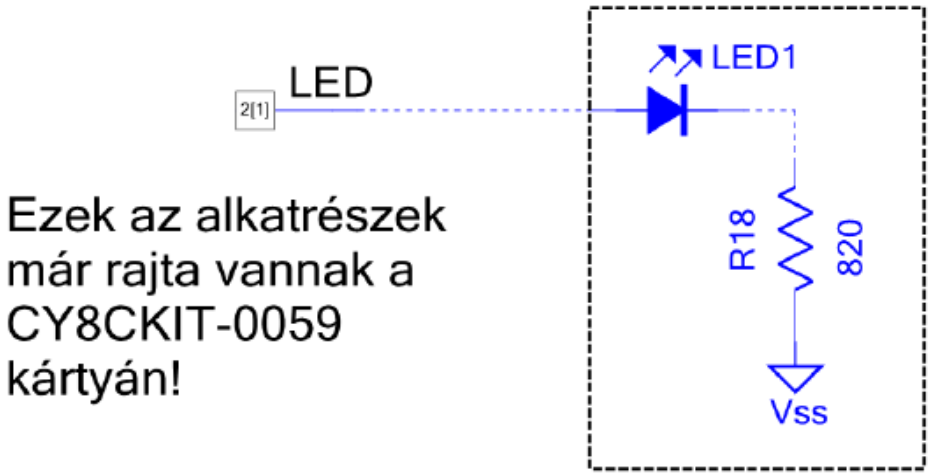
```
/* Function Name: PICU_ISR
 * Summary: Interrupt only occurs on falling edge of either pin, i.e., when
 * switch is pressed. Checks and initiates debounce timeout.
 */
CY_ISR(PICU_ISR){
    /* copies of PICU registers */
    uint8 CYDATA temp_stat;
    /* read the PICU interrupt status register, with a clear on read */
    temp_stat = Switches_ClearInterrupt();

    /* Process the PICU event on SW1 only if any
       ongoing debounce period has timed out */
    if(((temp_stat & SW1_MASK) != 0) && (switch_1_timeout == TIMED_OUT))
    {
        /* reset the debounce timer for this button */
        switch_1_timeout = DEBOUNCE_TIME;
    }

    /* Process the PICU event on SW2 only if any
       ongoing debounce period has timed out */
    if(((temp_stat & SW2_MASK) != 0) && (switch_2_timeout == TIMED_OUT))
    {
        /* reset the debounce timer for this button */
        switch_2_timeout = DEBOUNCE_TIME;
    }
}
```

D_SysTick projekt

- Ebben a projektben a **SysTick** időzítővel keltünk periodikus megszakításokat. A CY8CKIT-059 kártyán található LED1 állapotát minden megszakításkor átbillentjük.
- A SysTick megszakítási vektor a Cortex-M CPU-ban fixen 15.
- A Cortex-M NVIC 0–15 megszakítási vektorok a CPU mag számára fenntartottak (úgynevezett „kivételek”), a PSoC 5LP 0–31 megszakítások pedig a Cortex-M CPU 16–47 vektorainak felelnek meg.
- A PSoC Creator a kivételeket nem kezeli/támogatja, a CyLib azonban igen.



Ezek az alkatrészek már rajta vannak a CY8CKIT-0059 kártyán!

D_SysTick projekt – main.c

- Támogatás hiányában magunknak kell beállítani a megszakítási vektort és definiálnunk kell a kiszolgáló függvényt is *(3. módszer)*

```
#include <project.h>

#define SYSTICK_INTERRUPT_VECTOR_NUMBER  15u     // SysTick megszakítási vektor
#define CLOCK_FREQ      BCLK__BUS_CLK__HZ        // busz órajel frekvencia    (Hz)
#define INTERRUPT_FREQ 2u                        // Megszakítások gyakorisága (Hz)

CY_ISR(SysTick_ISR) {                            // Megszakítás kiszolgálása
    LED_Write(~LED_Read());
}

int main() {
    //--- A megszakítási vektor konfigurálása ---
    CyIntSetSysVector(SYSTICK_INTERRUPT_VECTOR_NUMBER, SysTick_ISR);
    //--- SysTick konfigurálása ---
    (void)SysTick_Config(CLOCK_FREQ / INTERRUPT_FREQ);
    CyGlobalIntEnable;                           // Megszakítások engedélyezése

    for(;;)      {
        // Nincs tennivalónk!
    }
}
```

CY8CKIT-059 fejlesztői kártya

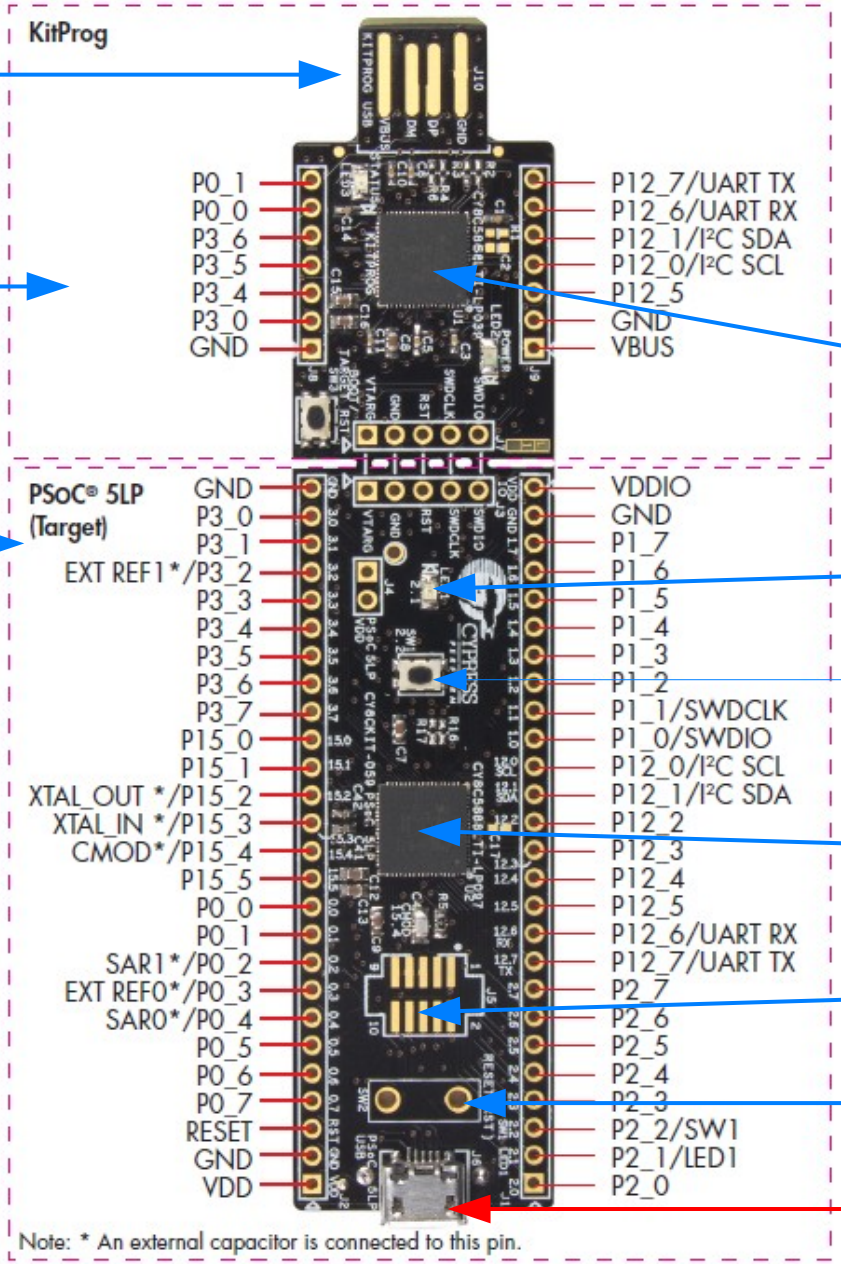
USB csatlakozás a PC-hez

KitProg programozó és hibavadász

PSOC 5LP Target áramkör

A tápellátás történhet a programozó felől (5V), Az alkalmazói USB csatlakozóról (5V), vagy a VDD csatlakozáson keresztül (3,3 – 5 V).

Utóbbi esetben a D1 és D2 diódákat el kell távolítani az USB-re csatlakozás előtt!



USB – UART Kivezetések

C8C5868LTI-LP039

LED1 (2.1 kivezetés)

SW1 (2.2 kivezetés)

CY8C5888LTI-LP097

JTAG csatlakozás

RESET gomb helye

USB alkalmazói csatl.

A céláramkör kapcsolási rajza