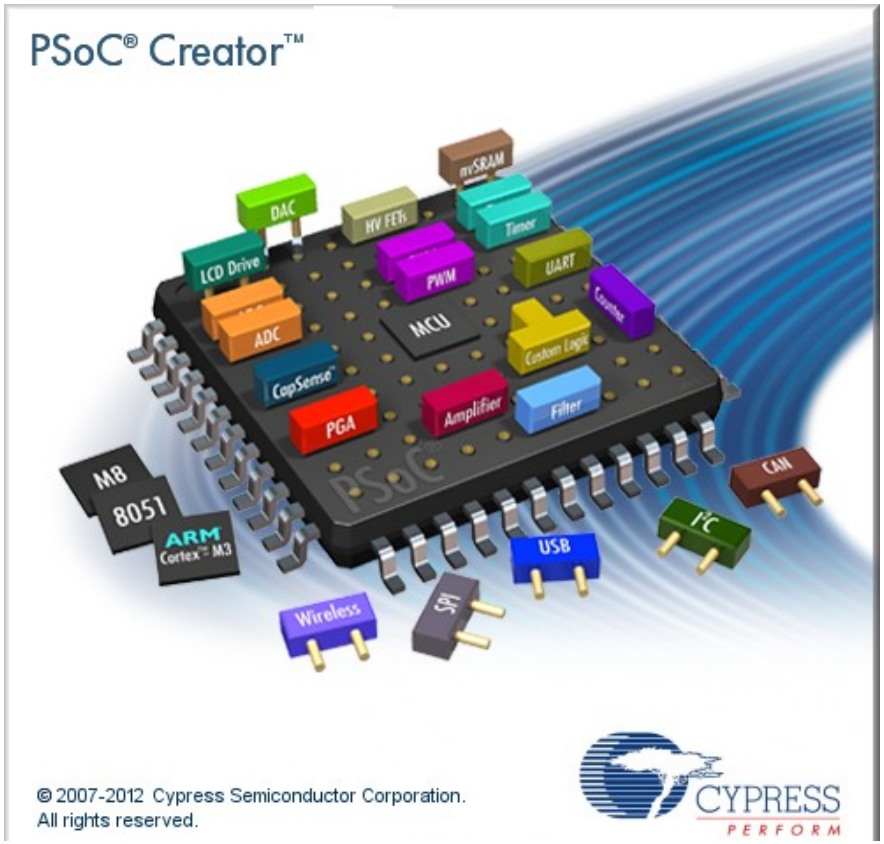
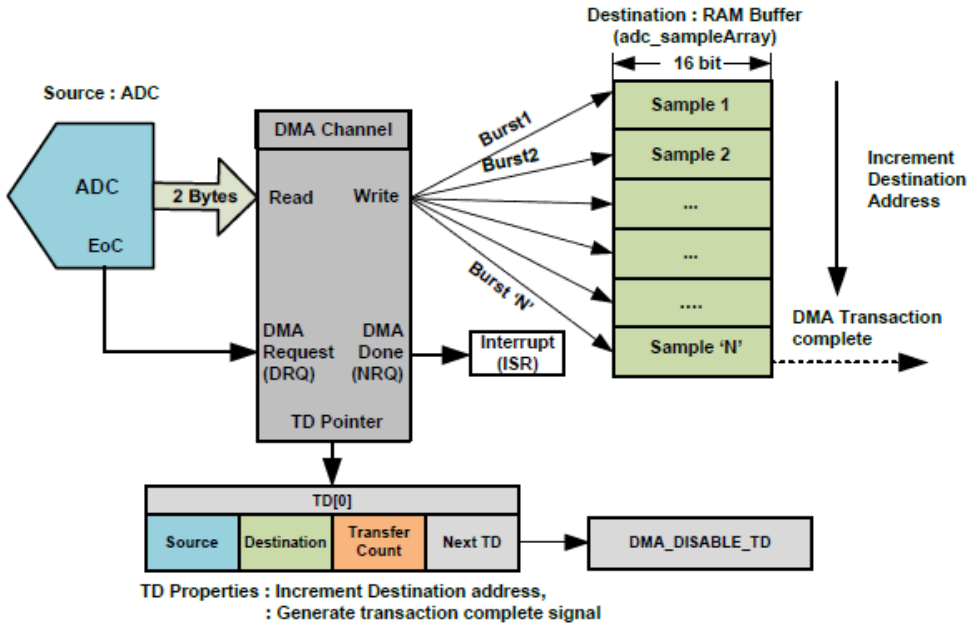
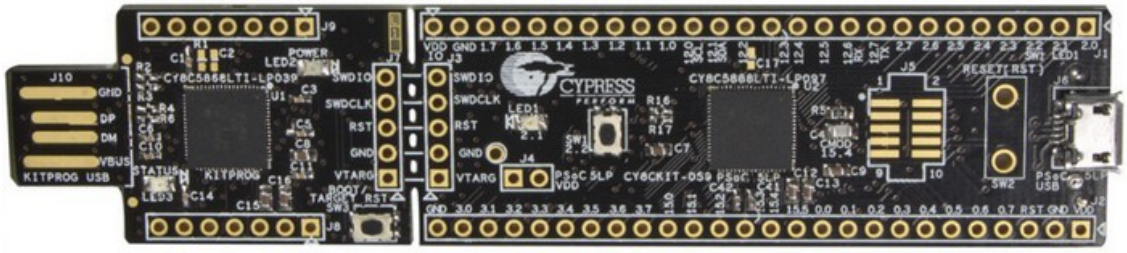
Újrakonfigurálható eszközök







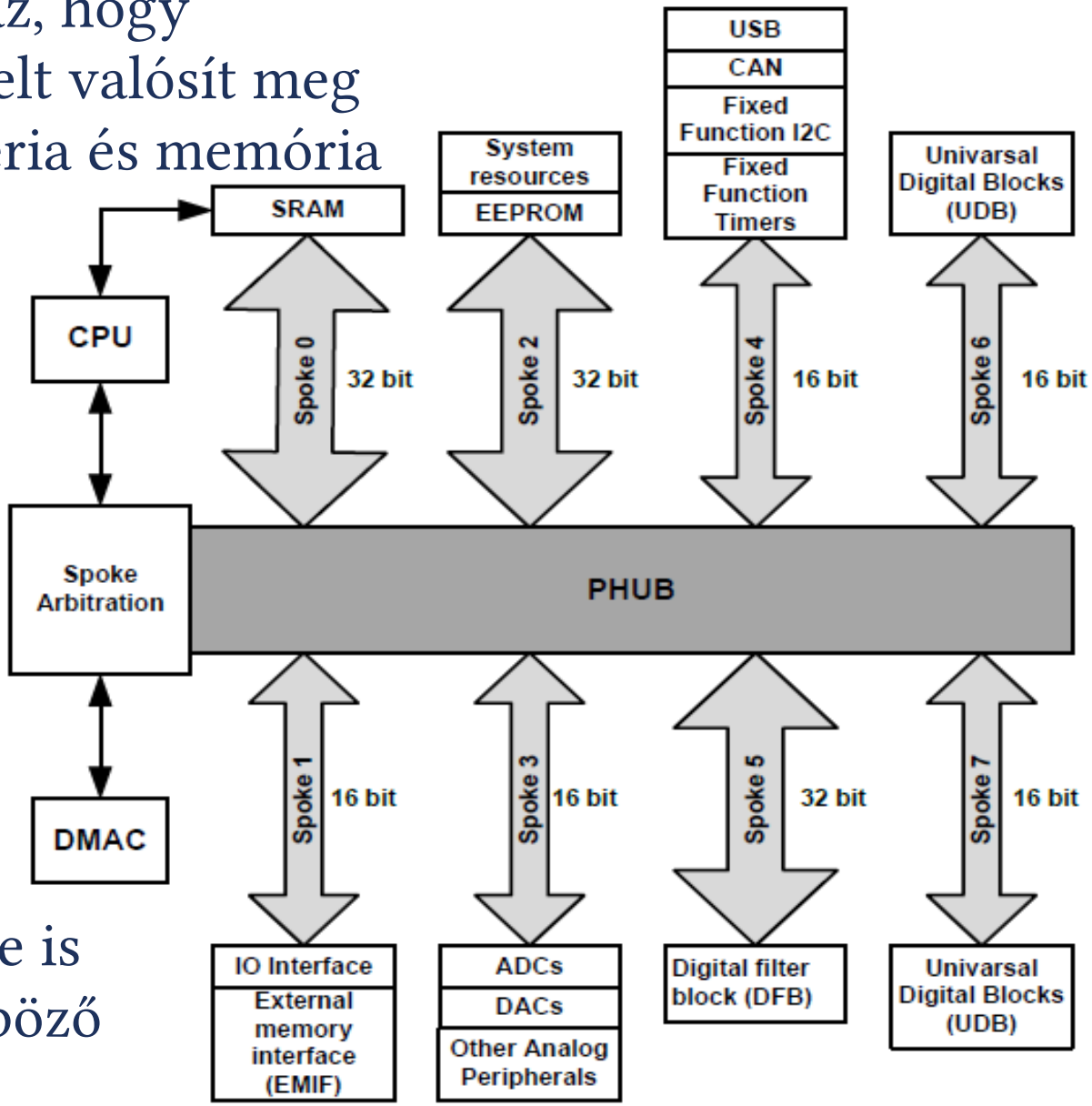
15. Cypress PSOC 5LP DMA adatátvitel

Felhasznált irodalom és segédanyagok

- Cypress: CY8C58LP FamilyDatasheet
- Cypress: PSOC 5LP Architecture Technical Reference Manual)
- Cypress: CY8CKIT-059 Prototyping Kit Guide
- Cypress: AN77 759: 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: AN52 705 PSOC 3 and PSOC 5 LP Getting Started with DMA

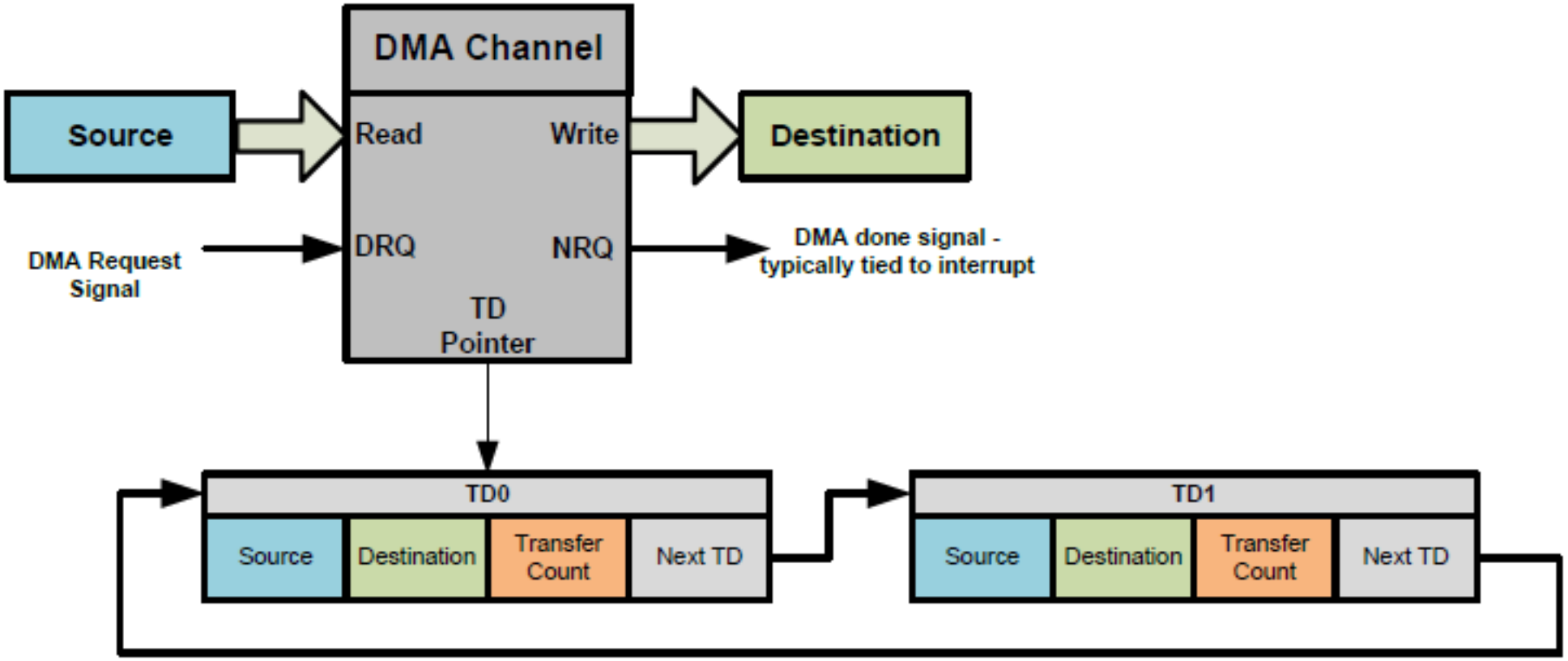
DMA – közvetlen memória hozzáférés

- A DMA átvitel lényege az, hogy CPU független adatátvitelt valósít meg két periféria, vagy periféria és memória között
- Az adatátvitel központja a periféria HUB (a HUB kerékagyat jelent) amelyhez 8 db „küllő” (adatbusz) kapcsolódik
- A buszvezérlők a CPU és a DMA vezérlője (DMAC) lehetnek
- A buszvezérlők egyszerre is működhetnek, ha különböző „küllőket” kezelnek



DMA csatornák

- A 24 db. DMA csatorna egymástól függetlenül működik
- Minden csatornához tartozik egy tranzakció leíró (TD) lánc
- Összesen legfeljebb 128 db. tranzakció leíró lehet
- Minden csatorna saját átviteli kérelem bemenettel rendelkezik



DMA konfiguráció

- A DMA átvitelt a csatorna és a tranzakciós leíró regiszterei konfigurálják
- A csatornához egy vagy több leíró láncolata csatlakozik

Csatorna konfiguráció

Forráscím felső 16 bitje

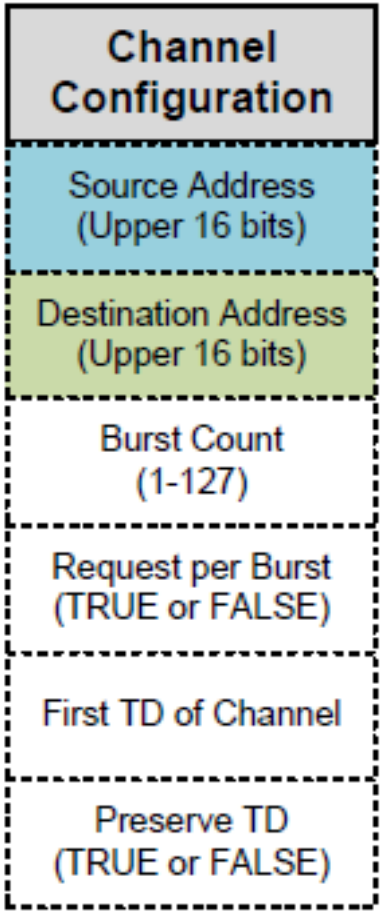
Cél cím felső 16 bitje

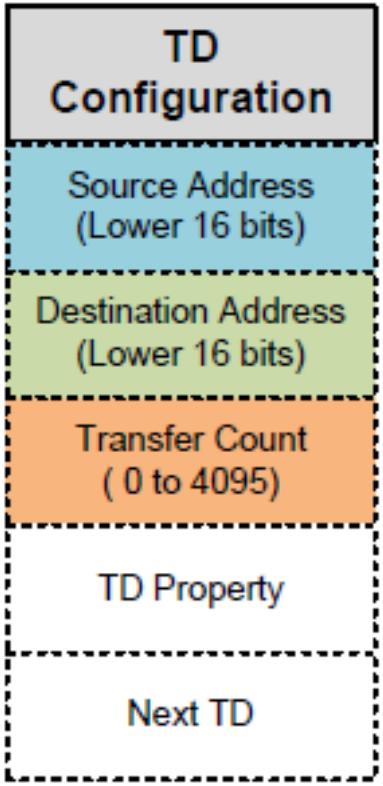
Adag mérete

Kérelem adagonként?

Első leíró sorszáma

Leíró megőrzés?





Tranzakció leíró konfiguráció

Forráscím alsó 16 bitje

Cél cím alsó 16 bitje

Tranzakciók száma

Leíró tulajdonsága

következő leíró sorszáma

A DMA csatorna konfigurálása

A PSOC Creator a DMA komponenshez API függvényeket generál, ezekkel konfiguráljuk a DMA csatornát az alábbi lépésekben:

1) A DMA csatorna inicializálása

```
Channel_Handle = DMA_DmaInitialize(DMA_BYTES_PER_BURST,
DMA_REQUEST_PER_BURST, HI16(Source Address), HI16(Destination Address))
```

2) Tranzakció leíró(k) példányosítása

```
TD_Handle = CyDmaTdAllocate();
```

3) A tranzakció leíró(k) konfigurálása

```
CyDmaTdSetConfiguration(TD_Handle,Transfer_Count,Next_TD,TD_Property);
```

4) Címek konfigurálása a tranzakció leíró(k)ban

```
CyDmaTdSetAddress(TD_Handle, LO16(Source Address), LO16(Destination
Address))
```

5) A csatorna kezdő tranzakció leírójának megadása

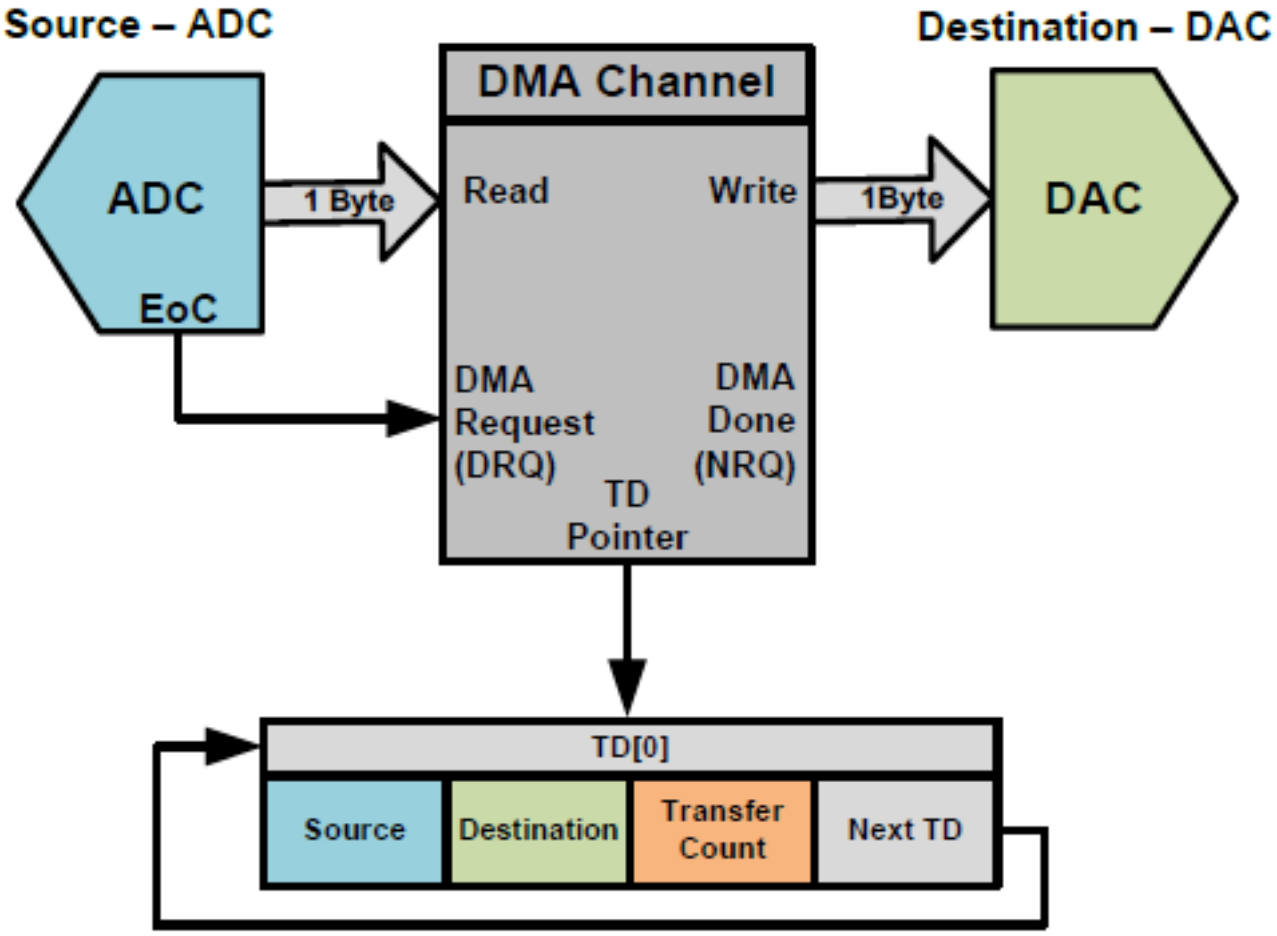
```
CyDmaChSetInitialTd(Channel_Handle, TD_Handle)
```

6) A DMA csatorna engedélyezése

```
CyDmaChEnable(Channel_Handle, preserve_TD)
```

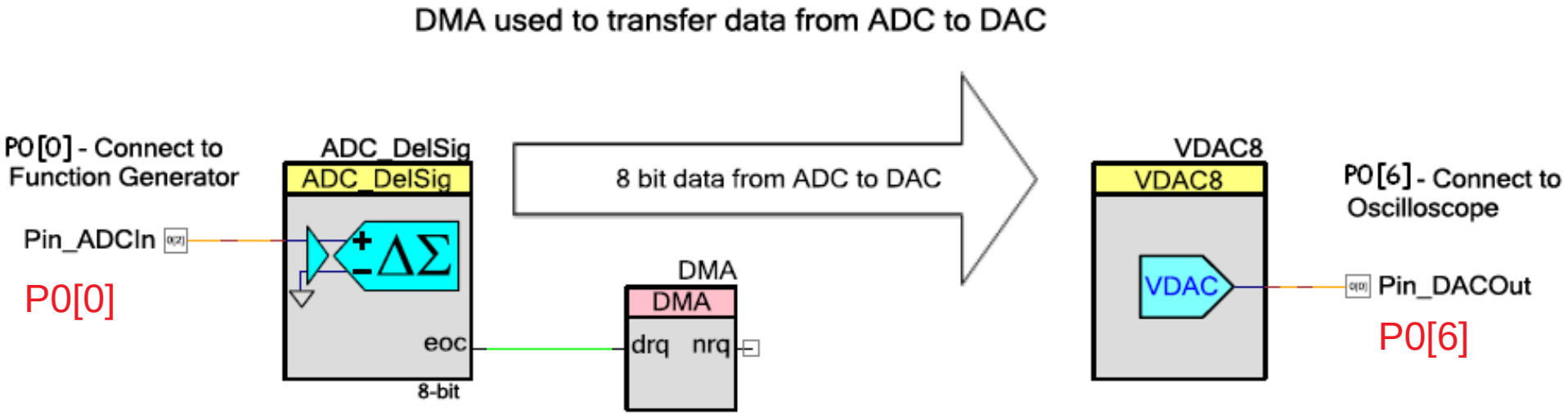
1. példa: periféria – periféria átvitel

- Egyszerű példa gyanánt egy periféria – periféria átvitelt mutatunk be: az ADC által konvertált 8 bites adatokat egy DAC-nak adjuk át
- Az átvitelt az ADC „konverzió vége” jele indítja



Eg1_ADC_DMA_DAC projekt

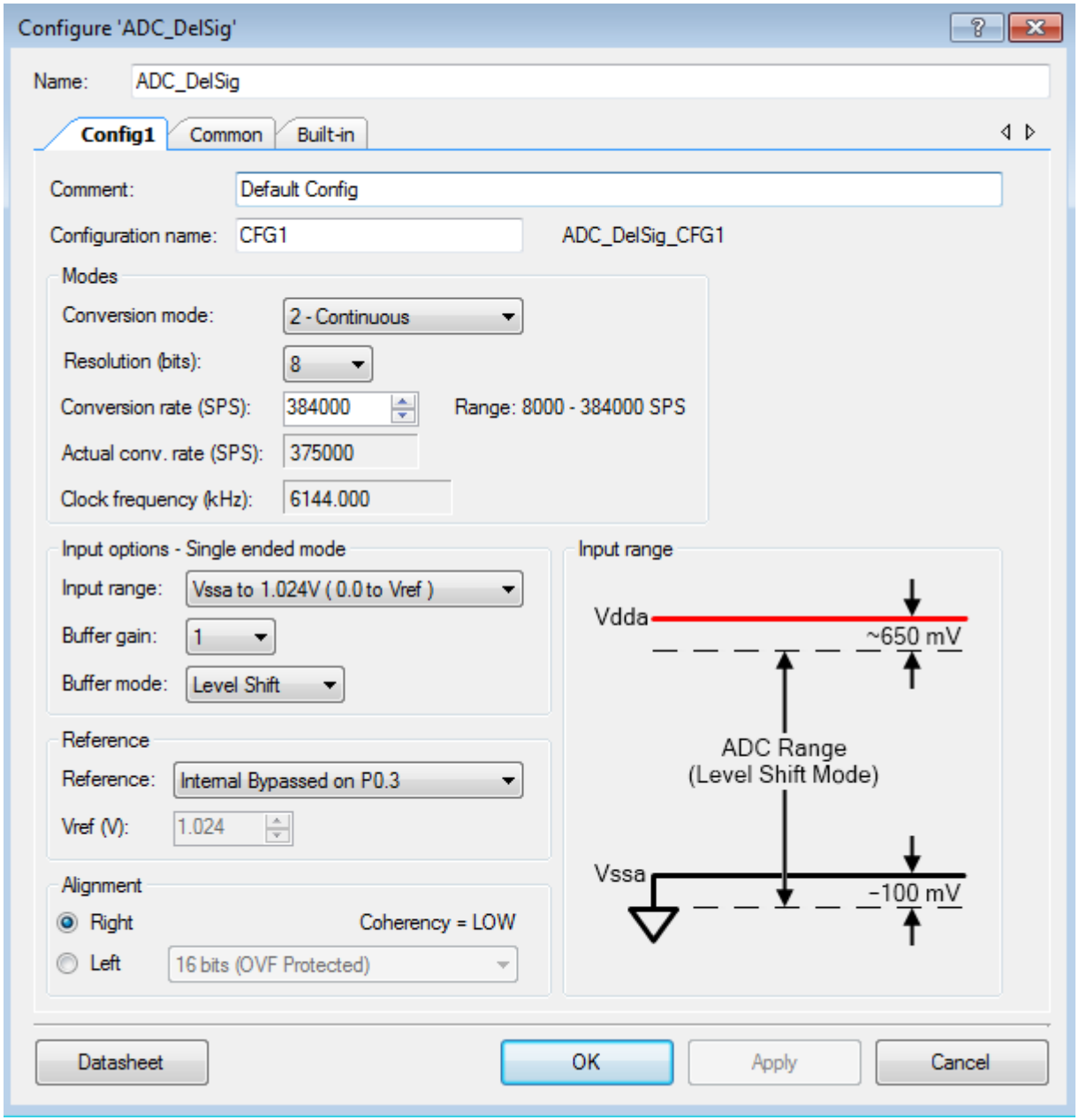
- Ebben a projektben a Delta-Sigma ADC-t 8 bites módba, 0 –1 V közötti bemenő feszültségre konfiguráltuk
- A projektet a **CY8CKIT-059** fejlesztői kártyára adaptáltuk
- Minden ADC konverzió után (375 000 mintavétel/s) a 8 bites eredményt átíratjuk a feszültségkimenetű VDAC8 eszközre
- A **P0[0]** bemenetre függvénygenerátort, a **P0[6]** kimenetre oszcilloszkópot csatlakoztatva ellenőrizhetjük az eredményt



Az ADC **0 – 1,024 V** közötti bemenő feszültséget fogad, úgy konfiguráltuk!

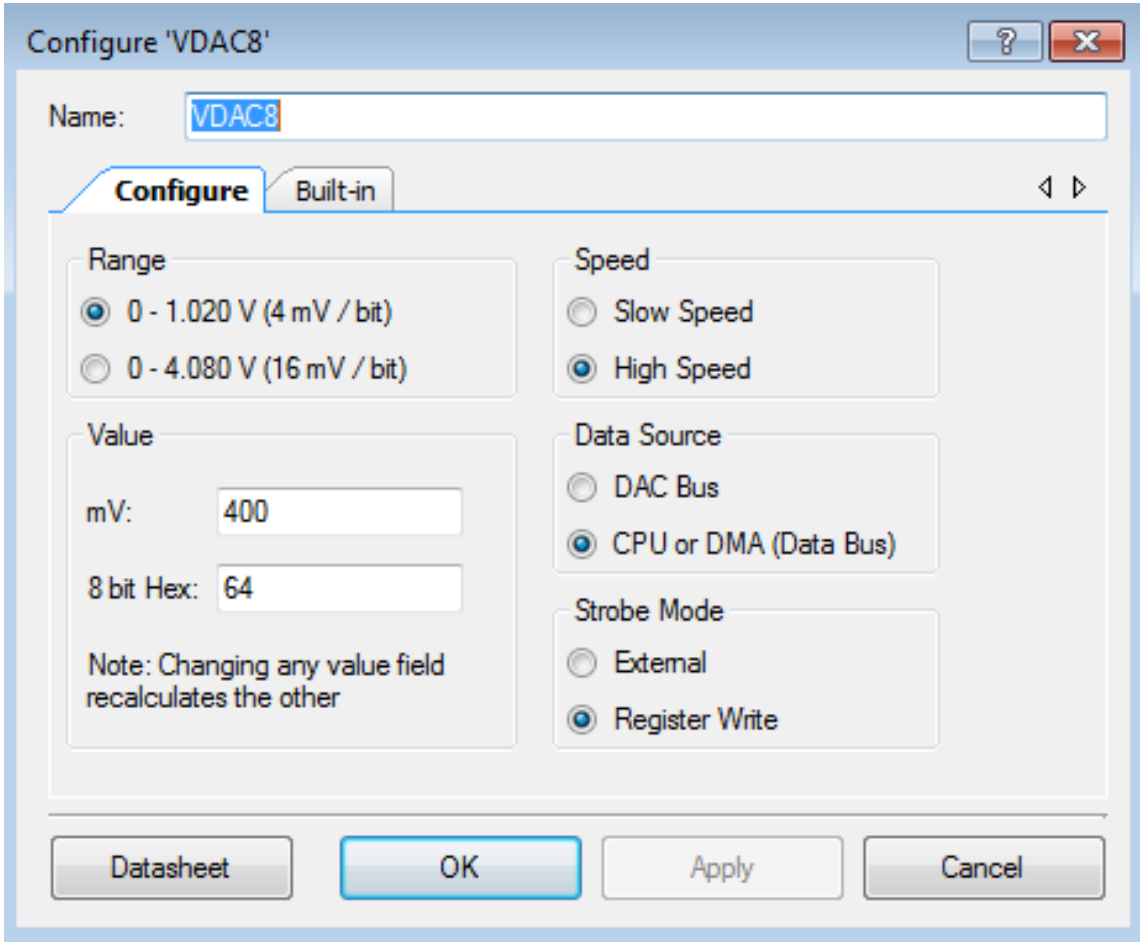
Az ADC konfigurálása

- Az ADC-t „Single ended” módba konfiguráljuk
- Folytonos mintavételezést állítunk be, 8 bites felbontással
- A mintavételezési frekvencia 375 000 Hz
- A belső referenciát szűréssel (P0[3]-on át) használjuk
- A bemeneti tartomány 0 – 1,024 V, szinteltolásos buffer móddal



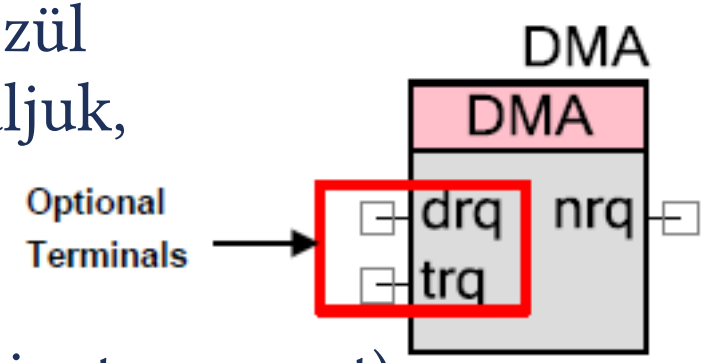
A DAC konfigurálása

- 0 – 1, 020 V tartomány
- Nagy sebesség
- CPU vagy DMA írja
- Alaphelyzet 400 mV

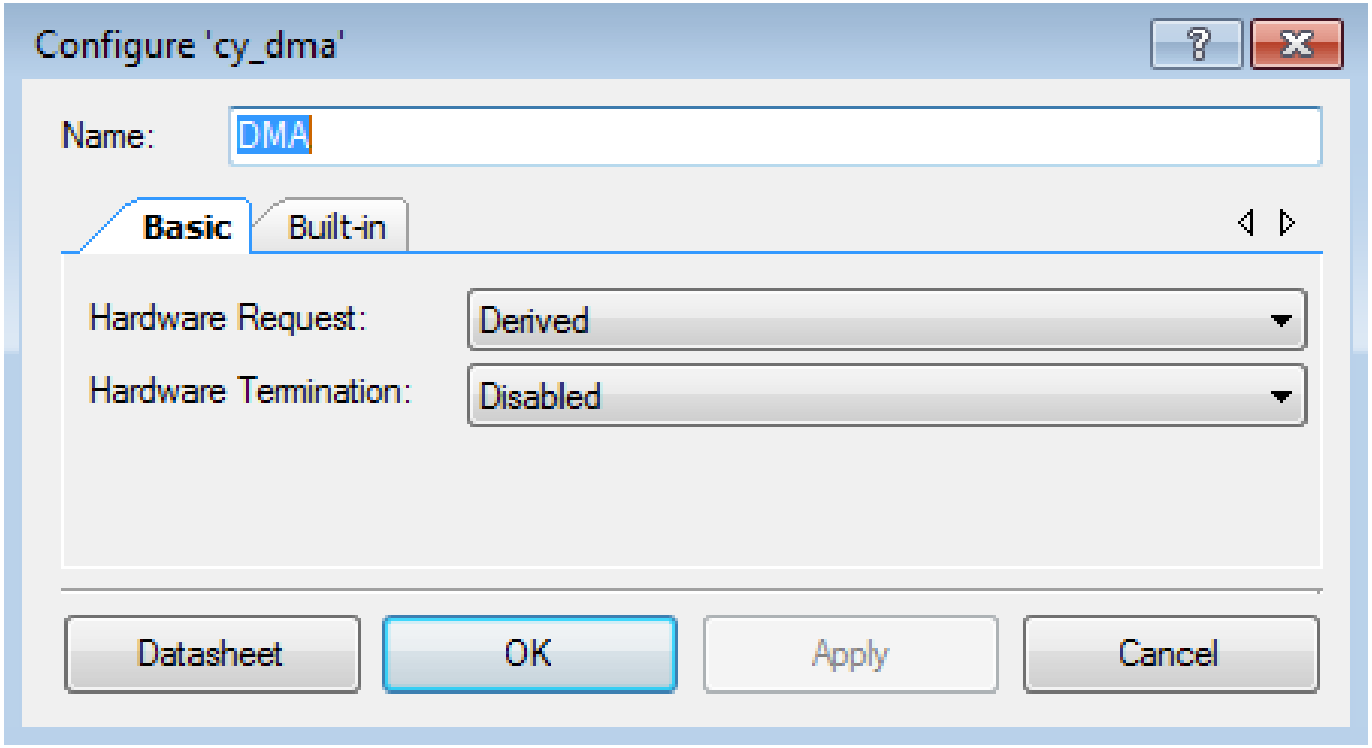


A DMA konfigurálása

- A DMA alaktrész opcionális bemenetei közül most csak a DMA átviteli kérelmet használjuk, amelyhez az ADC „konverzió vége” kimenőjele csatlakozik



- Az átvitel megszakítására való **TRQ** (terminate request) bemenetre most nincs szükségünk, ezért letiltjuk (Disabled)



main.c

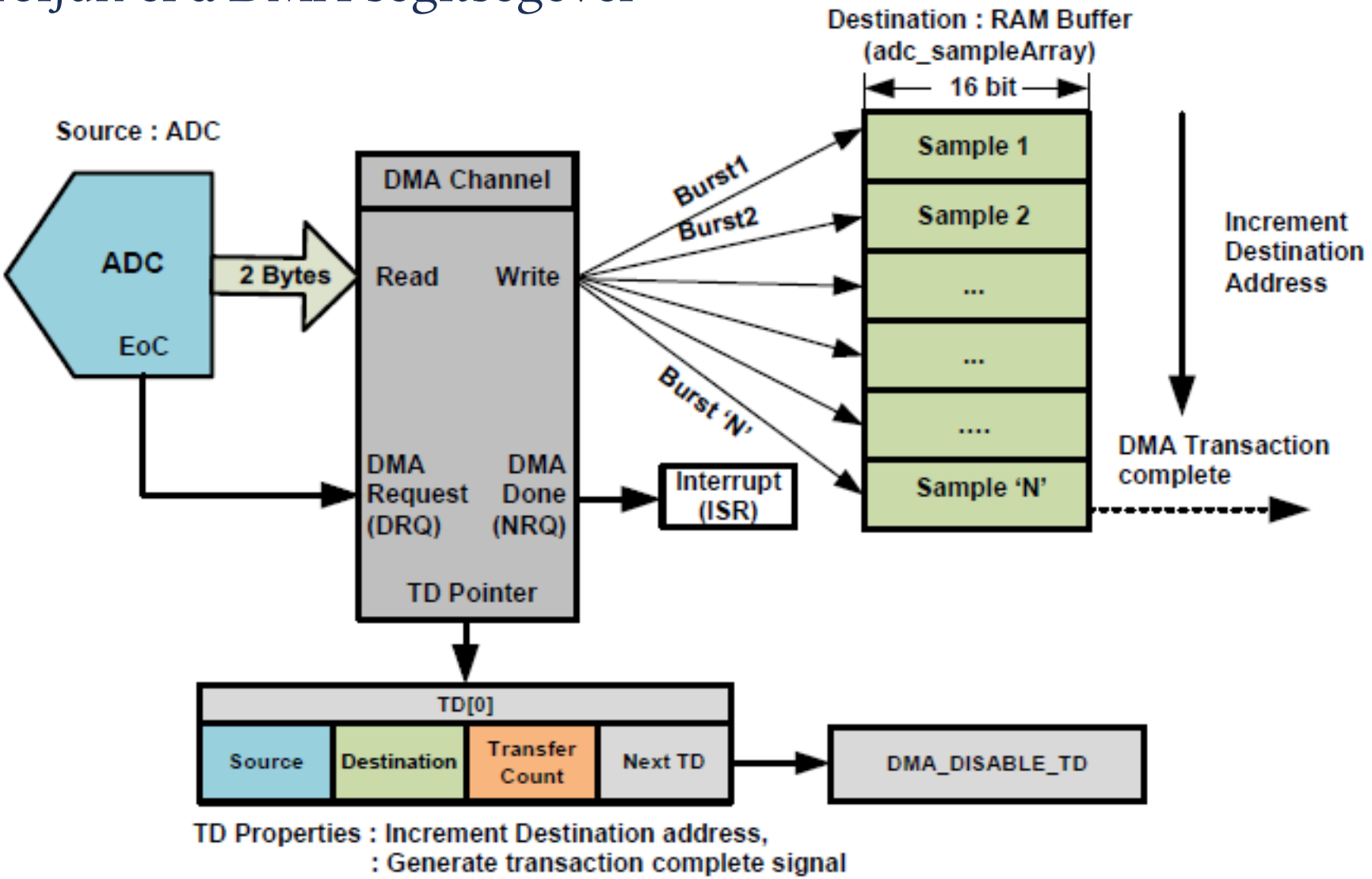
```
#include <device.h>
uint8 DMA_Chan;
uint8 DMA_TD[1];

int main(void) {
    #define DMA_BYTES_PER_BURST 1
    #define DMA_REQUEST_PER_BURST 1
    #define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
    #define DMA_DST_BASE (CYDEV_PERIPH_BASE)
    /* Step 1: DmaInitialize - Initialize the DMA channel */
    DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                                 HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));
    DMA_TD[0] = CyDmaTdAllocate();            // Step 2: Allocate TD */
    CyDmaTdSetConfiguration(DMA_TD[0],1,DMA_TD[0],0);
                                  // Step 3: Configure transfer count, next TD, property
    /* Step 4: Configure source and destination addresses */
    CyDmaTdSetAddress(DMA_TD[0],LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),
                                LO16((uint32)VDAC8_Data_PTR));
    CyDmaChSetInitialTd(DMA_Chan,DMA_TD[0]); // Step 5: Map the TD to the DMA Channel
    CyDmaChEnable(DMA_Chan, 1);               // Step 6: Enable the channel
    ADC_DelSig_Start();                       // Start ADC and DAC
    VDAC8_Start();
    ADC_DelSig_StartConvert();                // Start ADC Conversion

    for(;;) {   }
}
```

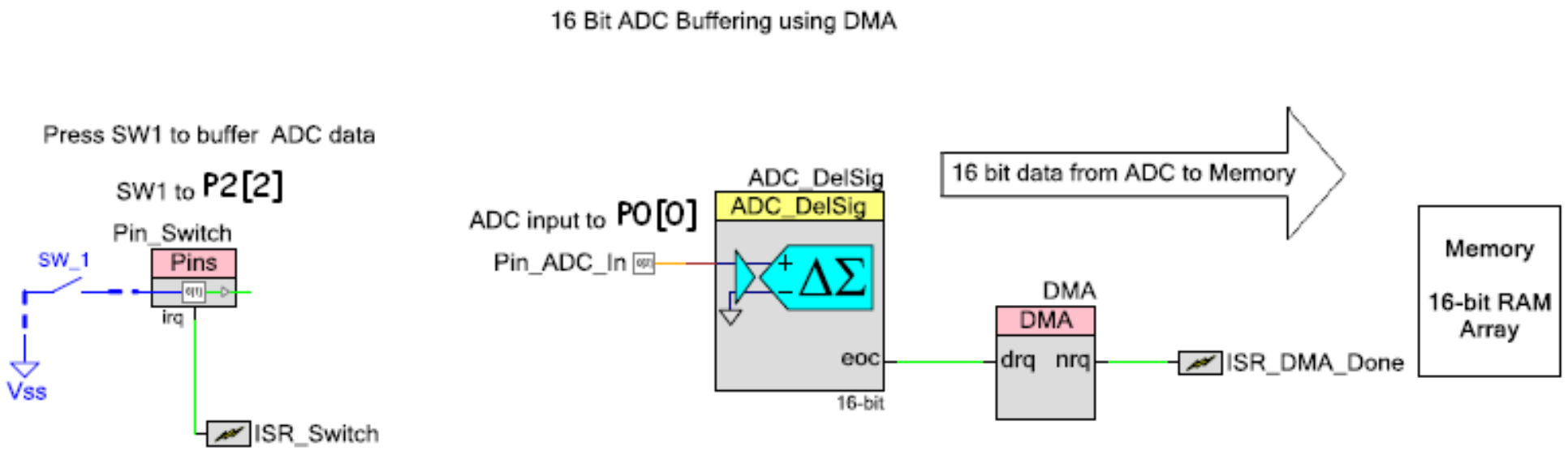
2. példa: átvitel perifériáról memóriába

- Ebben a példában az ADC-vel mért adatokat egy 16 bites tömbbe tároljuk el a DMA segítségével

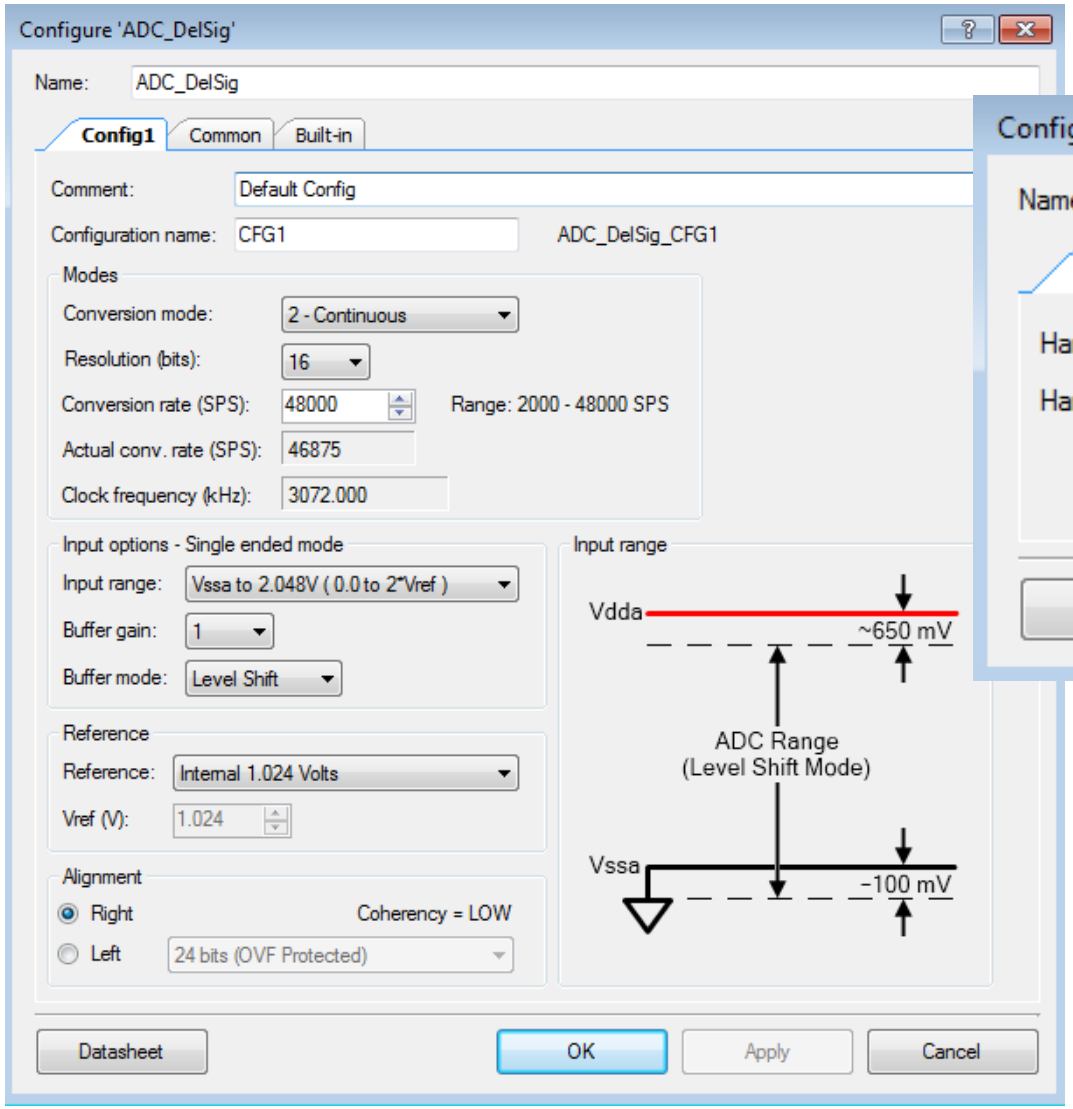


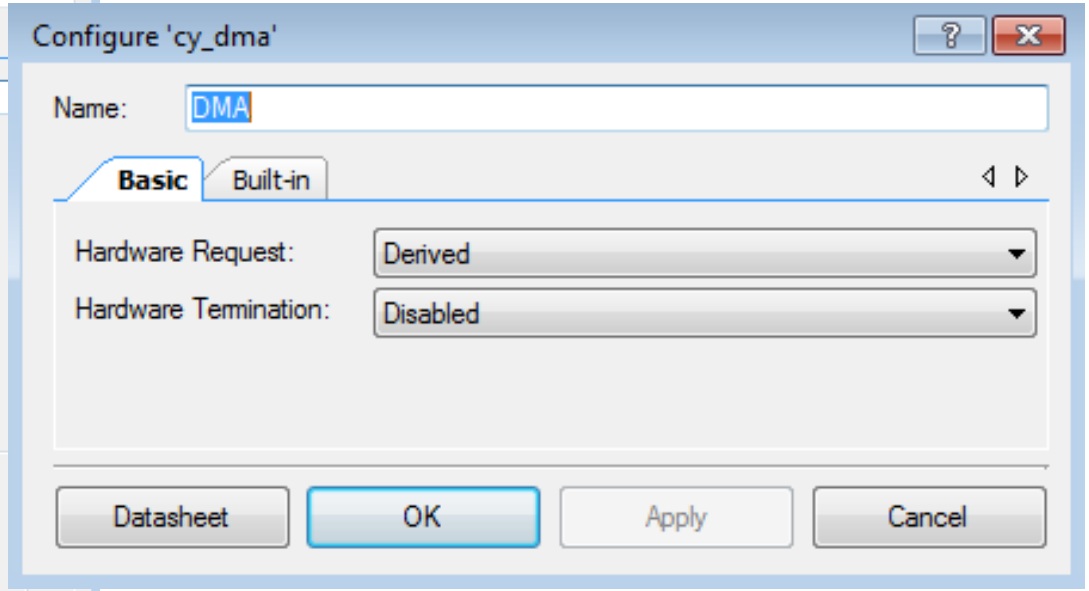
Eg2_ADC_DMA_Mem projekt

- Az **SW1** gomb lenyomásakor engedélyezzük a DMA átvitelt
- Minden ADC konverzió végén elmentésre kerül a 16 bites adat
- Az előírt számú adatmentés után a DMA átvitel véget ér, s egy megszakítás keletkezik, ahol értesíthetjük a főprogramot, hogy dolgozza fel az adatokat



ADC és DMAC Konfigurálás





A DMA csatorna konfigurálása

- A DMA csatorna és a tranzakció leíró konfigurációja az alábbi táblázatokban található
- Most 16 bites (két bájtos) adatokat viszünk át, azért az adag (burst) mérete 2 (bájt)
- A tranzakció jellemzői: cél címének léptetése, befejezési értesítés

Parameter	Project Setting
Upper Source Address	HI16(CYDEV_PERIPH_BASE)
Upper Destination Address	HI16(CYDEV_SRAM_BASE)
Burst Count	2 (Two bytes)
Request Per Burst	1 (True)
Initial TD	TD[0]
Preserve TD	1 (True)

Parameter	Project Setting
Lower Source Address	LO16(ADC_Delsig_DEC_OUTSAMP_PTR)
Lower Destination Address	LO16(&adc_sampleArray)
Transfer Count	200 x 2 (No. of samples × Bytes per sample)
TD properties	Increment Destination Address, Generate DMA done event, Swap Enable required only for PSoC 3. (TD_INC_DST_PTR | DMA__TD_TERMOUT_EN | TD_SWAP_EN)
Next TD	DMA_DISABLE_TD

main.c

```
#include <device.h>
#define NO_OF_SAMPLES 200
volatile uint8 switch_flag;
volatile uint8 DMADone_flag;
uint8 DMA_Chan;
uint8 DMA_TD[1];
int main() {
     uint16 adc_sampleArray[NO_OF_SAMPLES] = {0};
    #define DMA_BYTES_PER_BURST 2
    #define DMA_REQUEST_PER_BURST 1
    #define DMA_SRC_BASE (CYDEV_PERIPH_BASE)
    #define DMA_DST_BASE (CYDEV_SRAM_BASE)
     CYGlobalIntEnable;              // Enable global interrupt
     ISR_Switch_Start();
     ISR_DMA_Done_Start();
     DMA_Chan = DMA_DmaInitialize(DMA_BYTES_PER_BURST, DMA_REQUEST_PER_BURST,
                                  HI16(DMA_SRC_BASE), HI16(DMA_DST_BASE));
     DMA_TD[0] = CyDmaTdAllocate();
     CyDmaTdSetConfiguration(DMA_TD[0], (2 * NO_OF_SAMPLES), DMA_DISABLE_TD,
                                                  DMA__TD_TERMOUT_EN | TD_INC_DST_ADR);
     CyDmaTdSetAddress(DMA_TD[0],LO16((uint32)ADC_DelSig_DEC_SAMP_PTR),LO16((uint32)adc_sampleArray));
     CyDmaChSetInitialTd(DMA_Chan, DMA_TD[0]);
     ADC_DelSig_Start();              // Start ADC
     ADC_DelSig_IRQ_Disable();       // Disable the ADC ISR as it is not required
     ADC_DelSig_StartConvert();      // Start ADC conversion
     for(;;) {
          if(switch_flag) {
               CyDmaChEnable(DMA_Chan, 1);
               switch_flag = 0;
          }
          if(DMADone_flag) {
               DMADone_flag = 0;
          }
     }
}
```

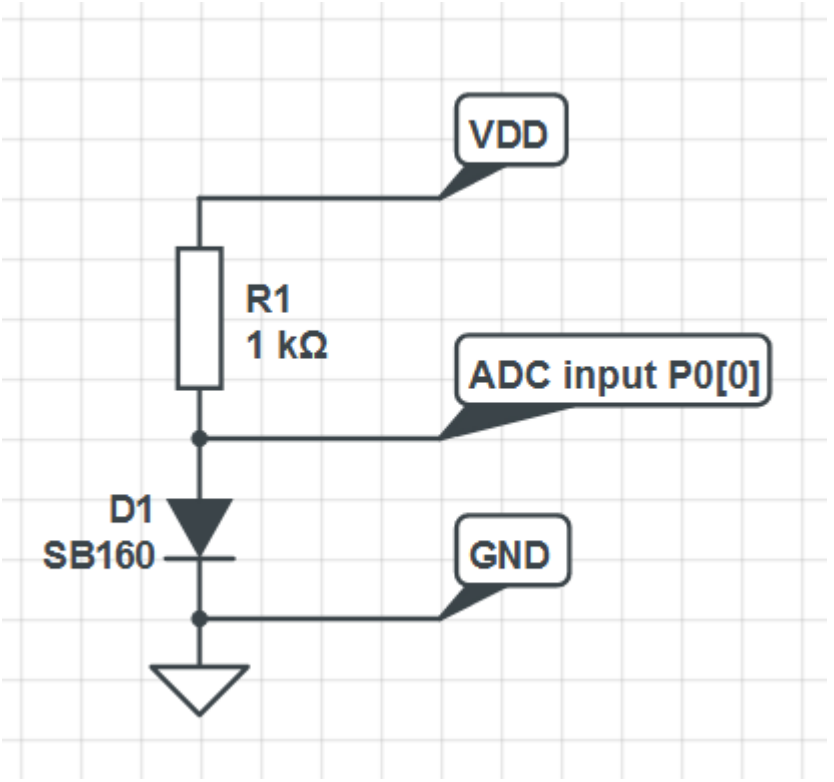
```
CY_ISR(ISR_DMA_Done_Interrupt){
     DMADone_flag = 1;
}

CY_ISR(ISR_Switch_Interrupt){
     switch_flag = 1;
     Pin_Switch_ClearInterrupt();
}
```

Ha itt elhelyezünk egy töréspontot, akkor nyomkövető módban ellenőrizhetjük az eredményt az **adc_sampleArray()** tömbben

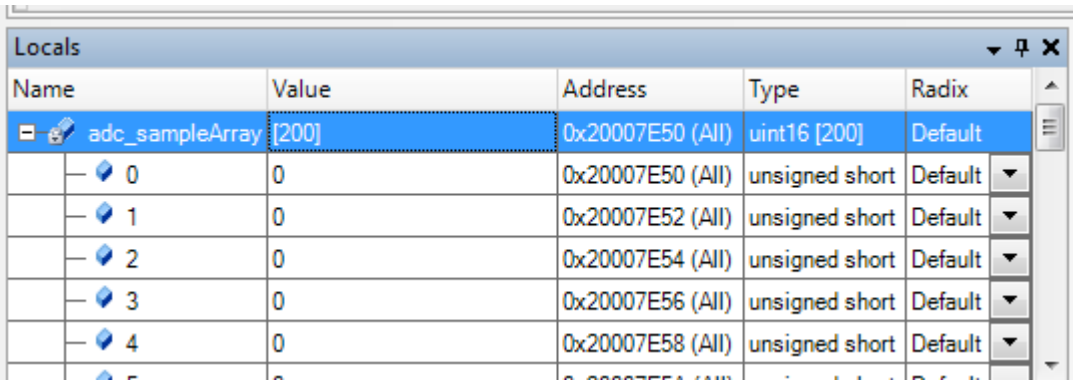
A program kipróbálása

- A programot az alábbi egyszerű kapcsolás segítségével próbáltuk ki
- Egy **SB160** Schottky diódát egy 1 kΩ-os ellenálláson keresztül kapcsoljunk a CYBCKIT-059 kártya tápfeszültségére (kb. 4.7 V)
- A diódán keresztülfolyó, kb 4.5 mA nyitóirányú áram hatására nagyjából 225 mV feszültség esik, ezt mérjük az ADC segítségével



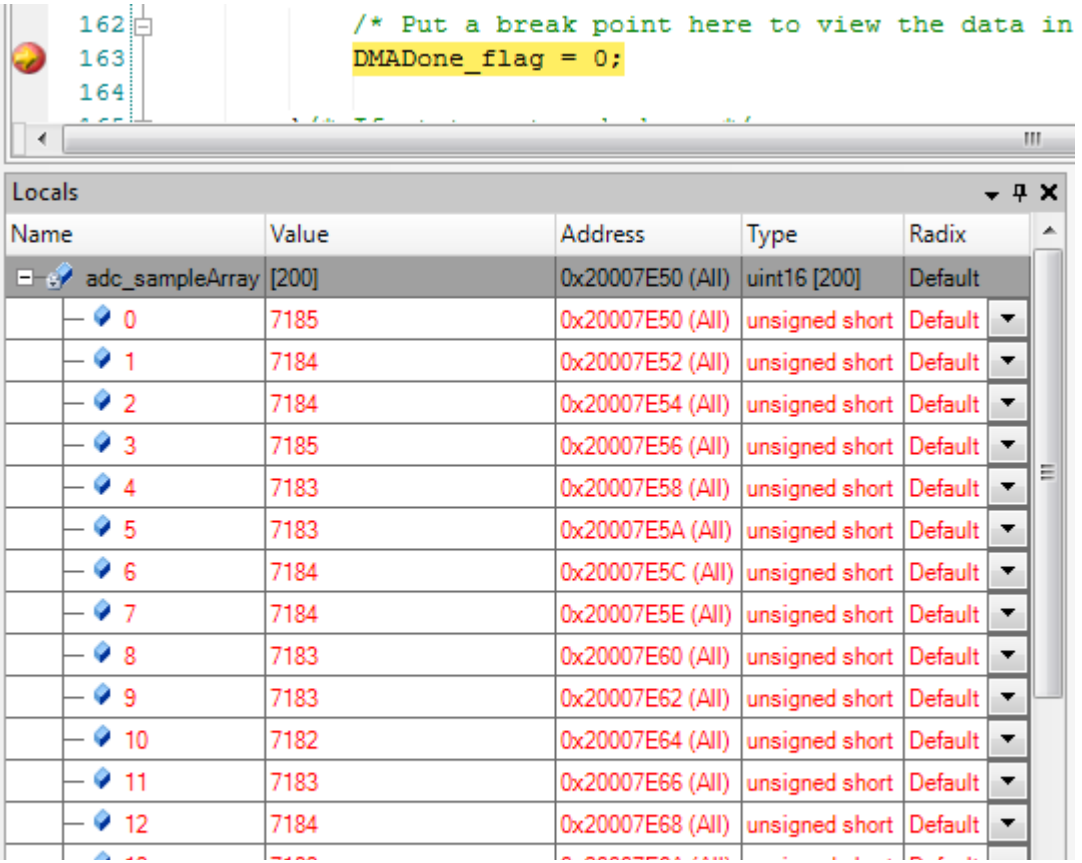
A program ellenőrzése nyomkövetéssel

- A program elején a tömb elemei csupa nullát tartalmaznak



- Ha a **DMADone_flag** bebillenésekor megnézzük az **adc_sampleArray()** tömb tartalmát, akkor láthatjuk az eredményeket. A 7184 körüli érték 224,5 mV-nak felel meg

  U = 7184 * 2048 mV/ 65 536 = 224,5 mV



CY8CKIT-059 fejlesztői kártya

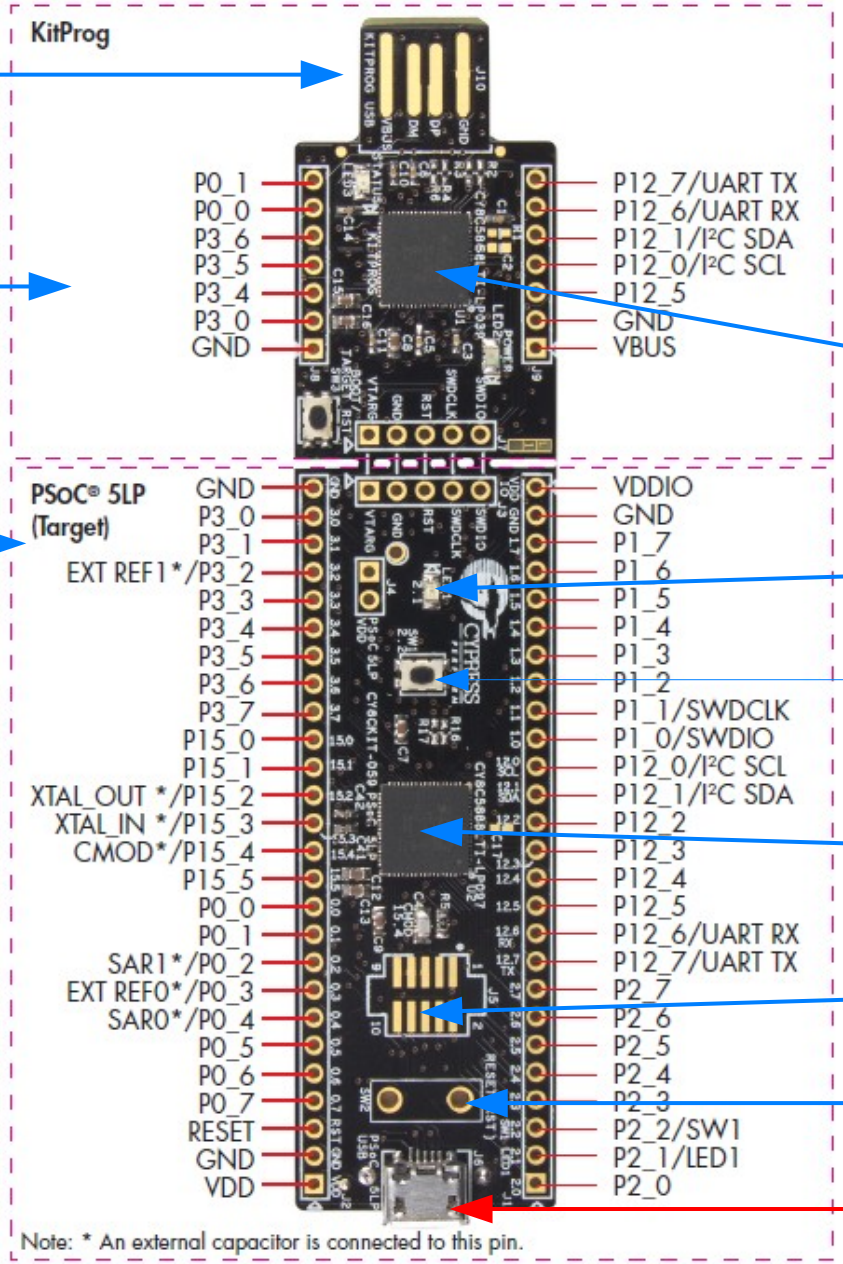
USB csatlakozás a PC-hez

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