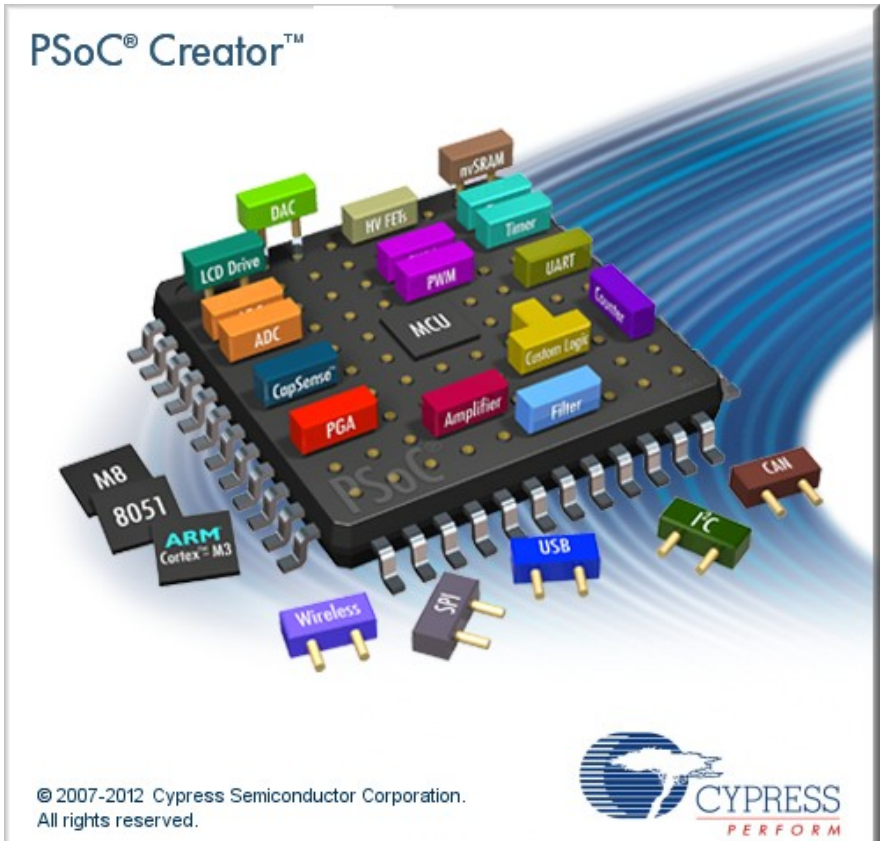
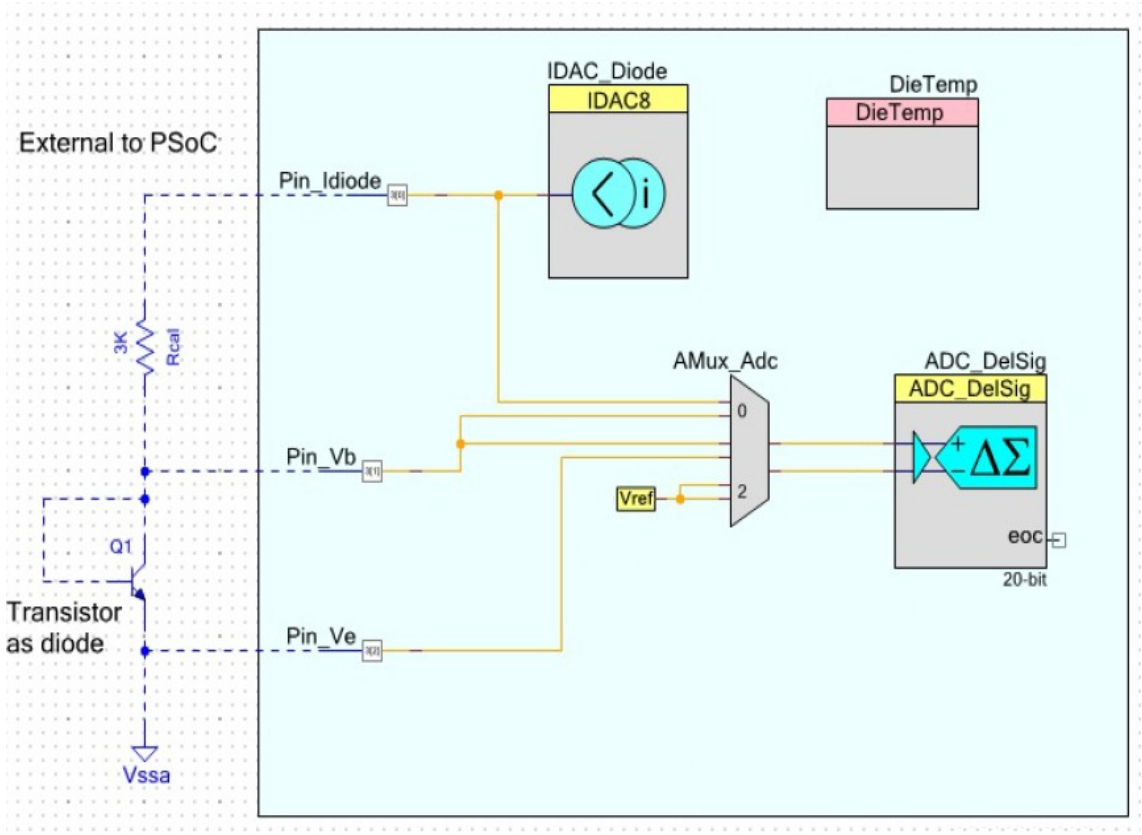
Újrakonfigurálható eszközök





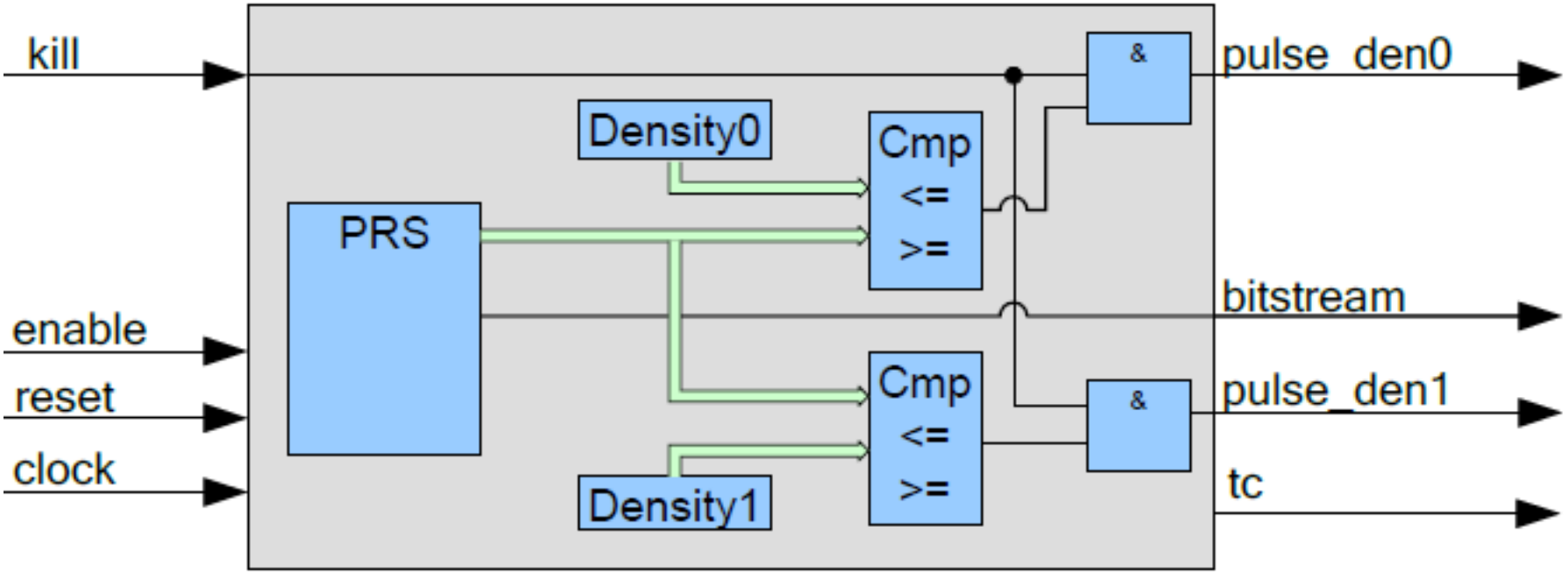
12. Cypress PSOC 5LP analóg perifériák – 2. rész

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
- Yuri Magda: Cypress PSoC 5LP Prototyping Kit Measurement Electronics
- Cserny István: PSOC 5LP Mikrokontrollerek programozása
- CE95 348 - Precise Illumination Signal Modulation (PrISM) with PS oC 3/4/5LP
- AN60 590 - PSoC® 3, PSoC 4, and PSoC 5LP - Temperature Measure ment with a Diode

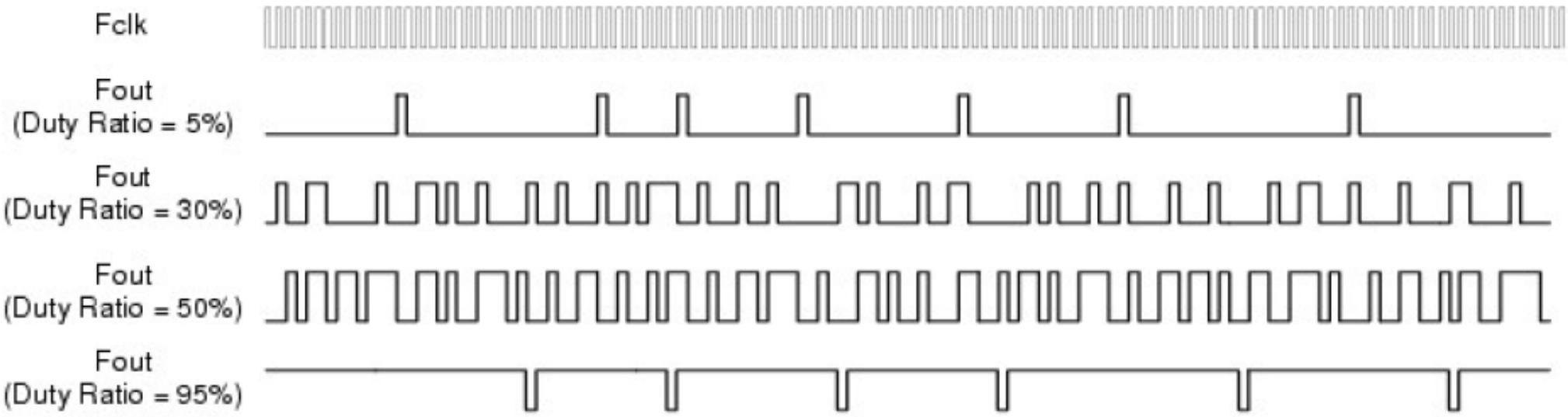
Mit csinál a PrISM modul?

- A **Precision Illumination Signal Modulation (PrISM)** modul egy 2-32 bites Galois LFSR regisztert, két sűrűség-regisztert és két összehasonlítót (digitális komparátor tartalmaz). Ezek segítségével két, különböző sűrűségű (kitöltésű) álvéletlen impulzus-sorozatot állíthatunk elő
- Valójában ez egy digitális modul, de a PWM-hez hasonlóan „kvázi analóg” szabályozásra használjuk



PrISM kimeneti jelalak

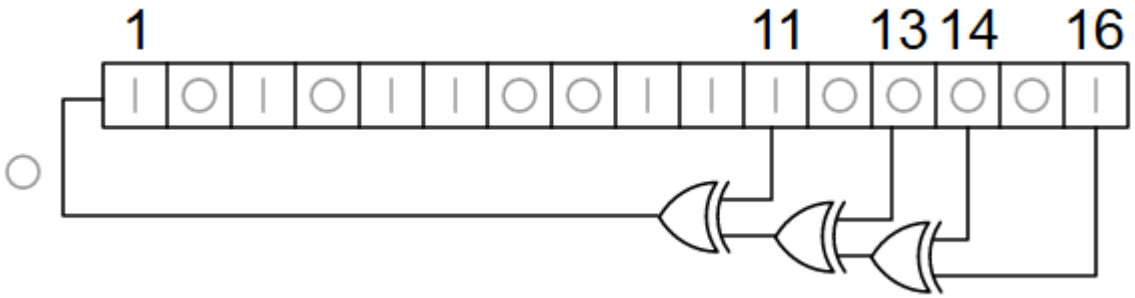
- Az alábbi ábrán különböző sűrűségű (kitöltésű) jelsorozat idődiagramját mutatjuk be
- A **PWM**-hez képest sűrűbb a kimenőjel váltakozása, ezért hatékonyabban szűrhető, a szemünket pedig kevésbé terheli a villogás
- Az ismétlődési periódus az **LFSR** bitszámától és a bemeneti frekvenciától függ: $T = (2^N - 1) / F_{clk}$



- Megjegyzés: a fenti ábra nem pontos, csupán illusztráció!

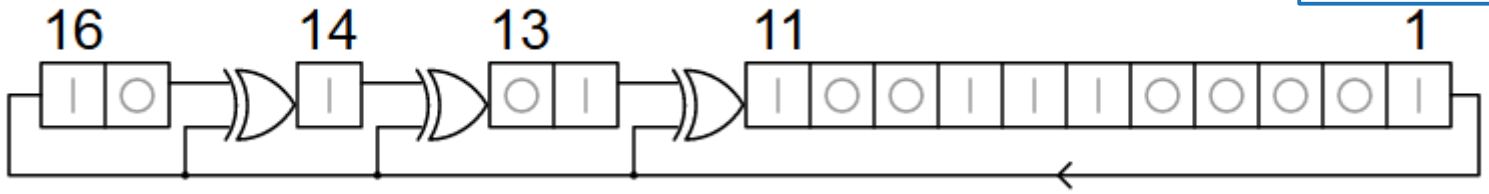
LFSR – Linear-feedback shift register

- A lineárisan visszacsatolt shift regiszterek segítségével álvéletlen bitsorozatot állíthatunk elő, vagy „megkevert sorrendű” számlálóként használhatjuk
- Jól megválasztott „csapolási helyekkel” n-bites regiszterrel 2^n-1 periódus érhető el (a csupa 0 állapot kizárt!)
- Fibonacci LFSR



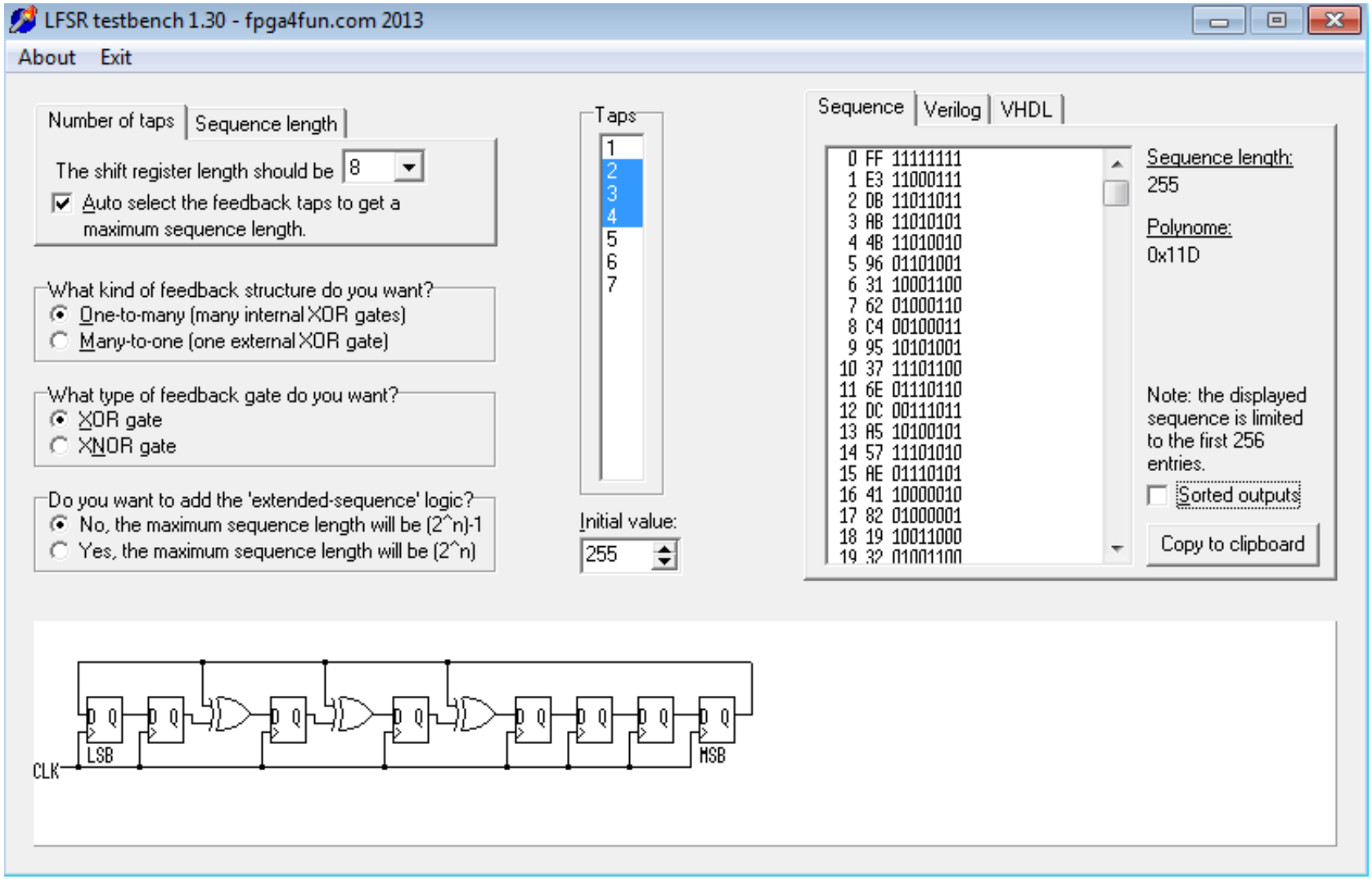

```
// 8-bites Galois LFSR Verilog kódja
module LFSR8_11D(  input clk,
  output reg [7:0] LFSR = 255 );
  wire feedback = LFSR[7];

  always @(posedge clk) begin
    LFSR[0] <= feedback;
    LFSR[1] <= LFSR[0];
    LFSR[2] <= LFSR[1] ^ feedback;
    LFSR[3] <= LFSR[2] ^ feedback;
    LFSR[4] <= LFSR[3] ^ feedback;
    LFSR[5] <= LFSR[4];
    LFSR[6] <= LFSR[5];
    LFSR[7] <= LFSR[6];
  end
endmodule
```

- Galois LFSR



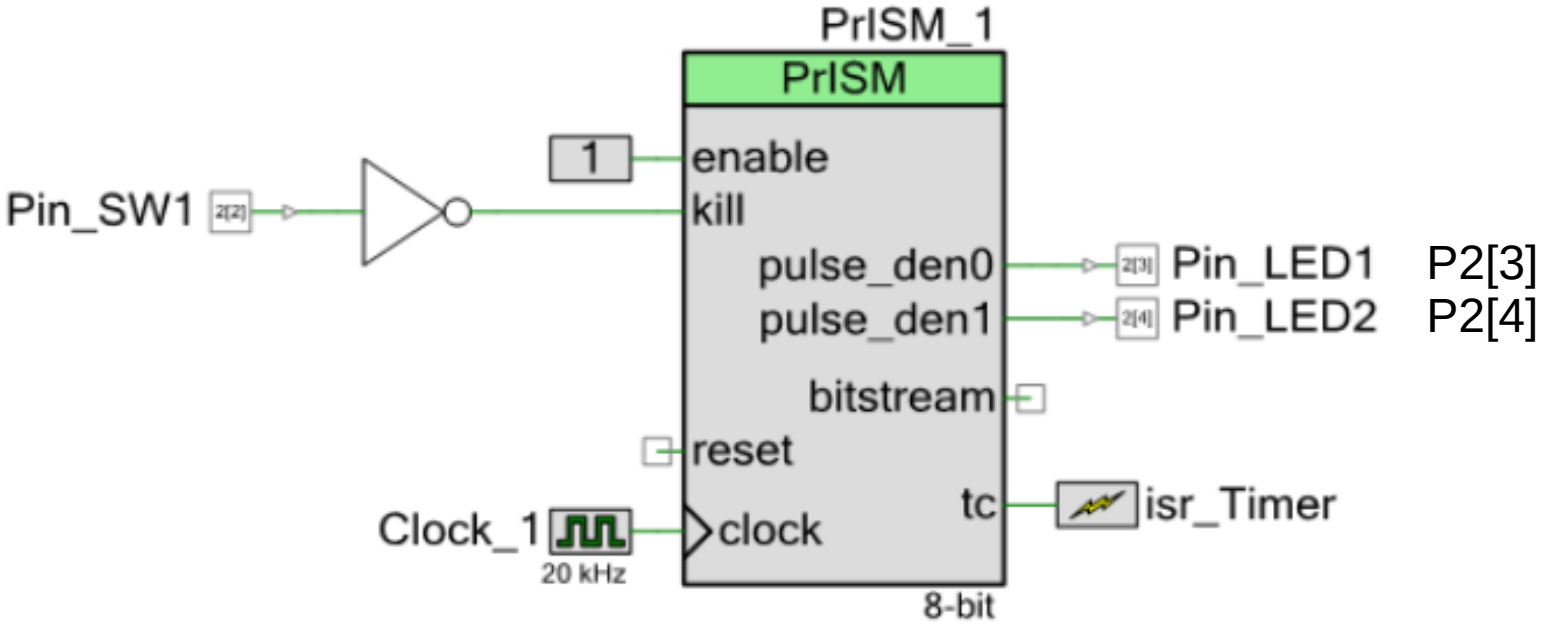
Az LFSR testbench alkalmazás

- Az fpga4fun.com/Counters3.html oldalon található program segíti az LFSR számlálók működésének megértését és megtervezését



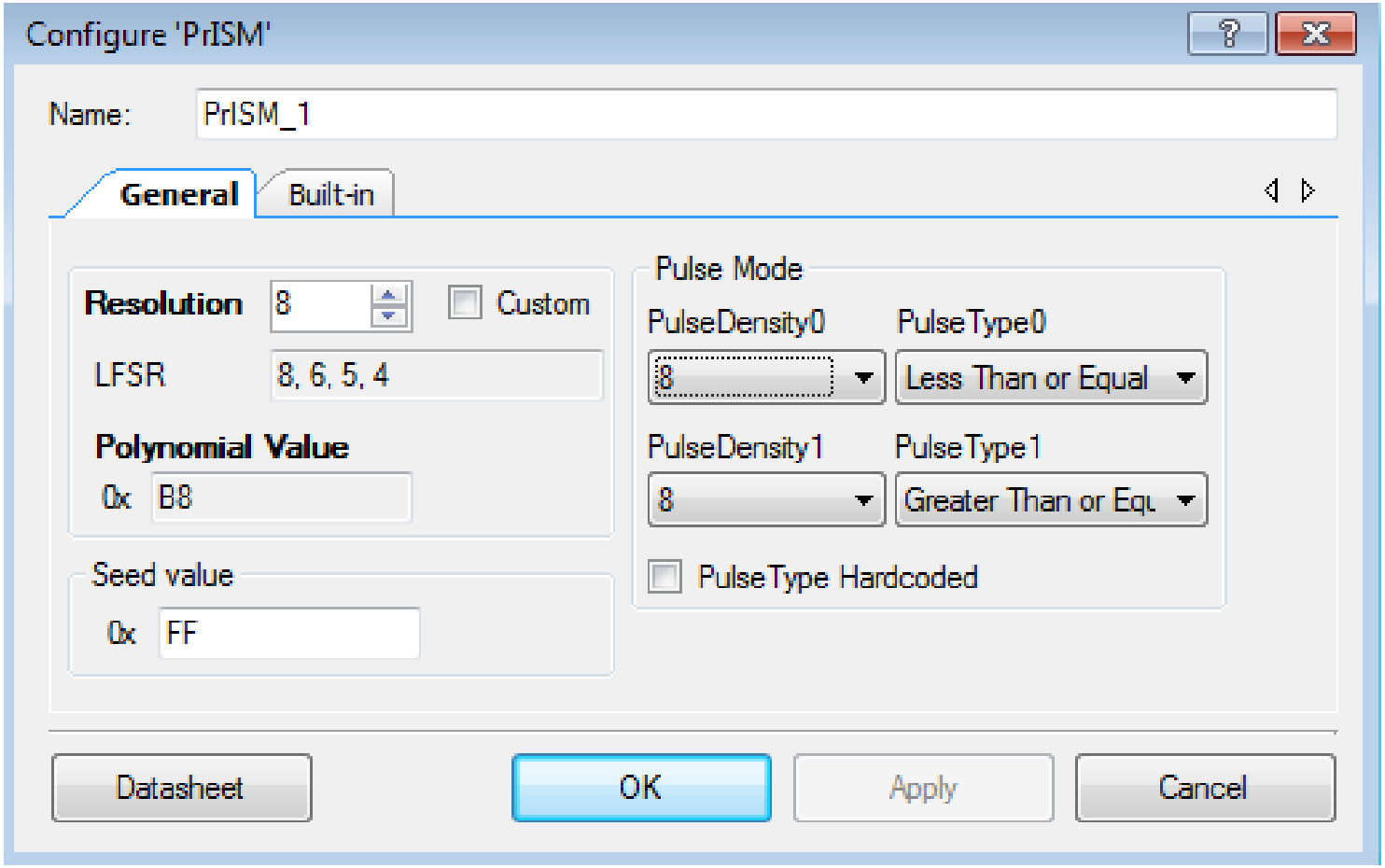
CE95 348_PrISM mintapélda

- A **CE95 348 mintaprojekt** eredetileg a **CY8CKIT-001** fejlesztői kártyához és **PSOC3 CY8C3866AXI-040** processzorhoz készült, de adaptáltuk a **CY8CKIT-059** kártyához
- A projekt a **Precision Illumination Signal Modulation** (PrISM) modul egyszerű használatát mutatja be
- A projekt két LED fényerejét ellenütemben folyamatosan változtatja. Az SW1 nyomógombbal mindkét LED letiltható.



A PrISM modul konfigurálása

- A bemenő frekvencia 20 kHz legyen!
- Az LFSR regiszter 8 bites, az automatikusan választott polinommal a periódus 255 órajel ciklus, az ismétlődési frekvencia kb. 78 Hz



CE95348_PrISM main.c

- A főprogram csak elindítja a modulokat...

```
int main()
{
    /* Init ISR connected to the TC pin for change LED contrast */
    isr_Timer_StartEx(Timer);

    /* Start PrISM */
    PrISM_1_Start();

    /* Enable Global Interrupts */
    CyGlobalIntEnable;

    /* No work has to be done in main loop */
    while(1u);

}
```

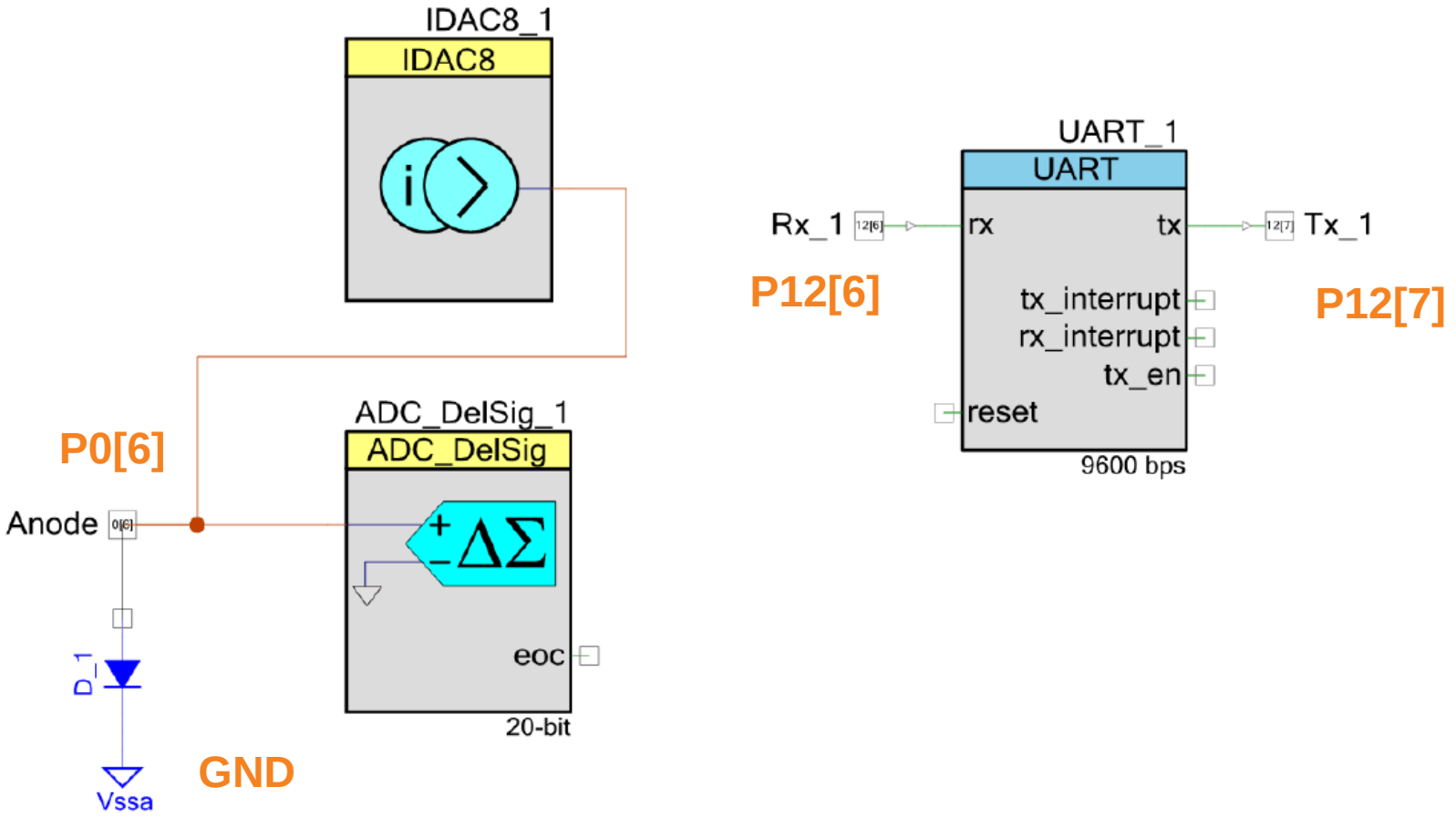
CE95348_PrISM main.c

- A Timer megszakításban léptetjük a kitöltést szabályozó density változót (0, ill. 255 értékhatárnál irányt is váltunk)

```
CY_ISR(Timer)
{
    /* flip direction at max and min points */
    if((density == 0u) ||(density == 0xffu))
    {
        direction ^= FORWARD;
    }
    /* change density for the LEDs*/
    if(direction & FORWARD)
    {
        density++;
    }
    else
    {
        density--;
    }
    /* overwrite density to PrISM registers */
    PrISM_1_WritePulse0(density);
    PrISM_1_WritePulse1(density);
}
```

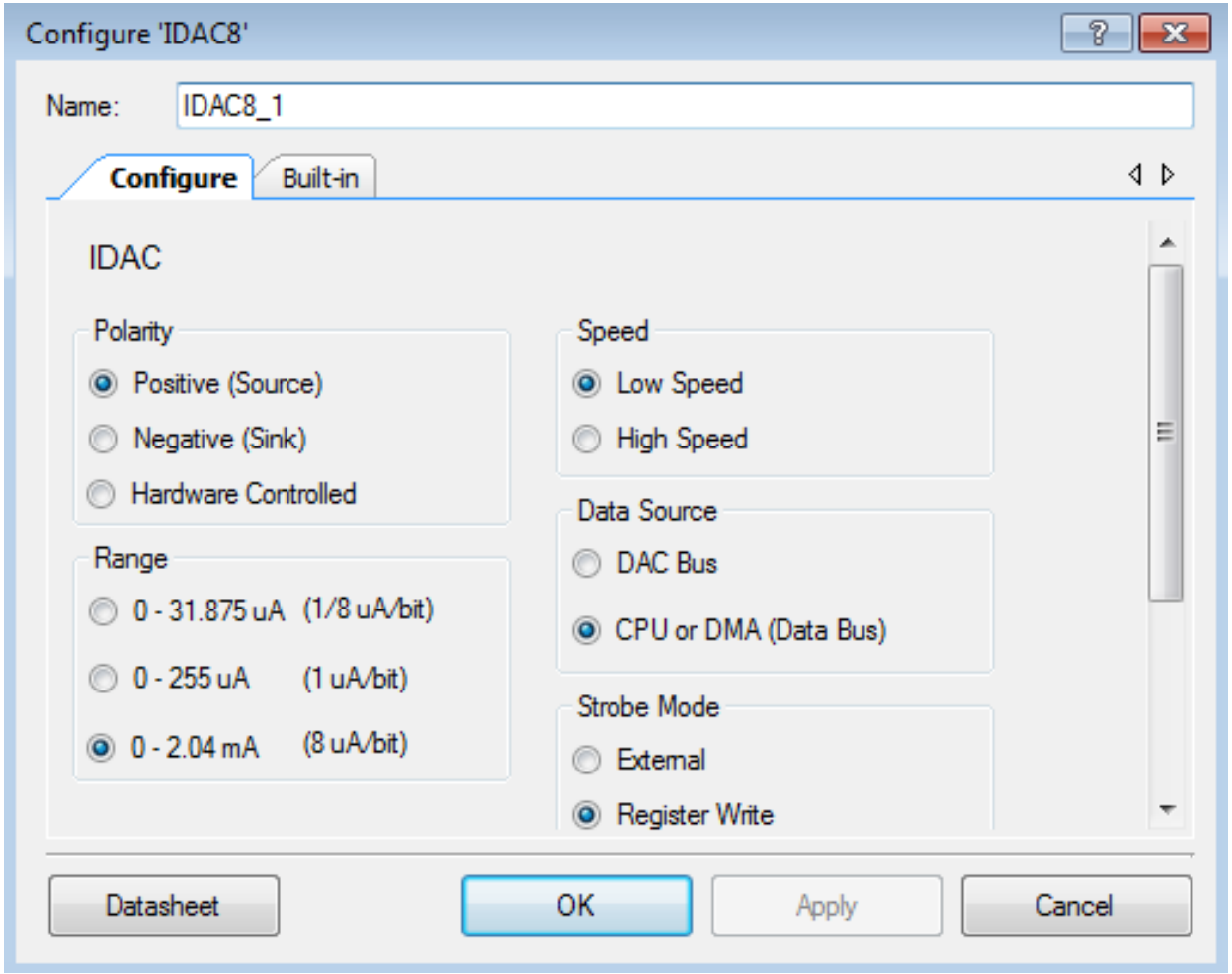
Diode_test projekt

- Az automata diódavizsgáló az UART porton érkező s vagy S paranccsal indítható. Egy áramkimenetű IDAC8 egység 0 – 2 mA közötti áramot hajt át a diódán 255 lépésben, közben mérjük a diódán eső feszültséget és kiíratjuk az UART porton.



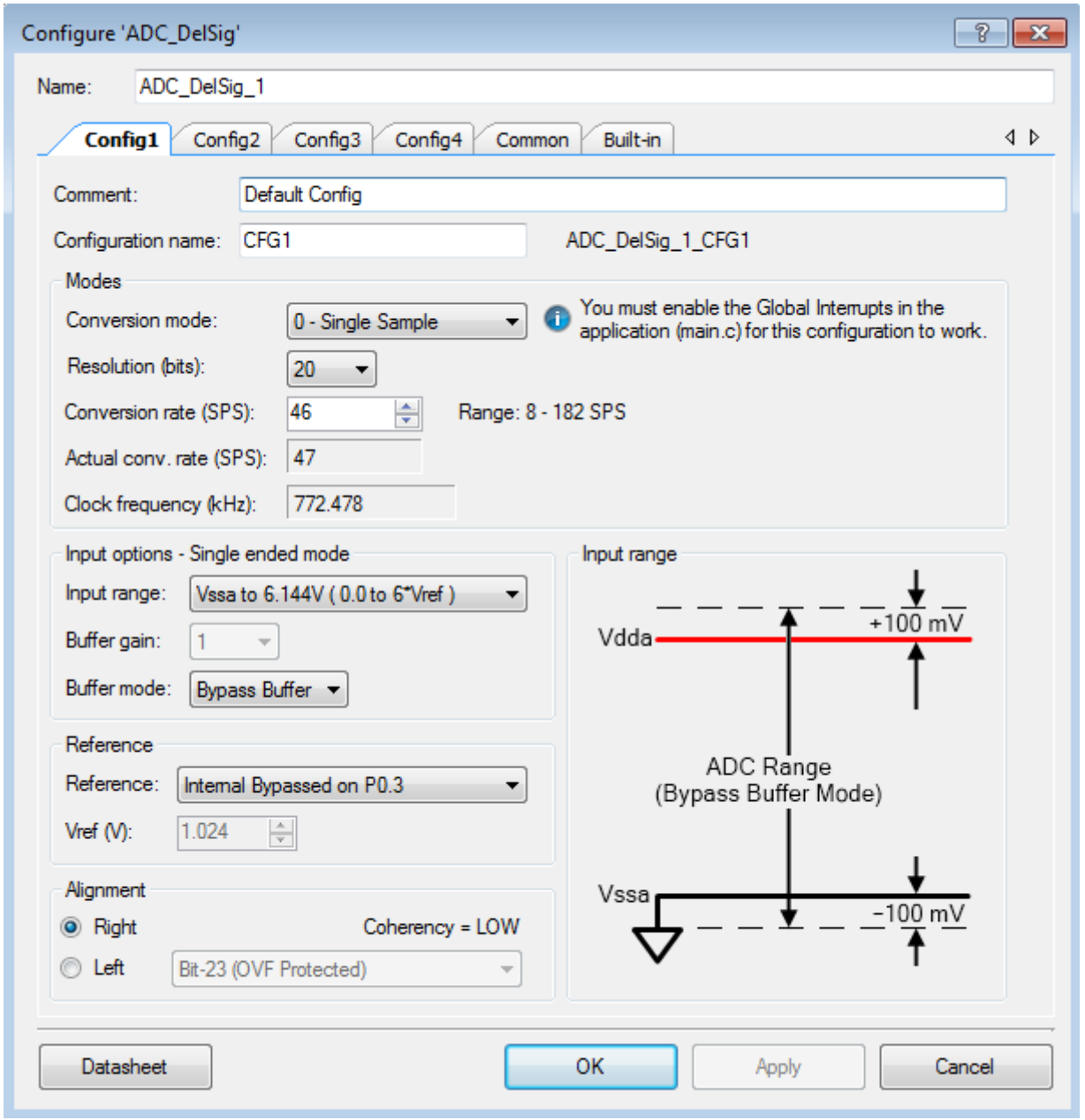
IDAC8 konfigurálása

- Pozitív (kifolyó) áramra konfiguráljuk (P0[6] az anódra megy)
- A 0 – 2.04 mA tartományt választjuk ki



A Delta-Sigma ADC konfigurálása

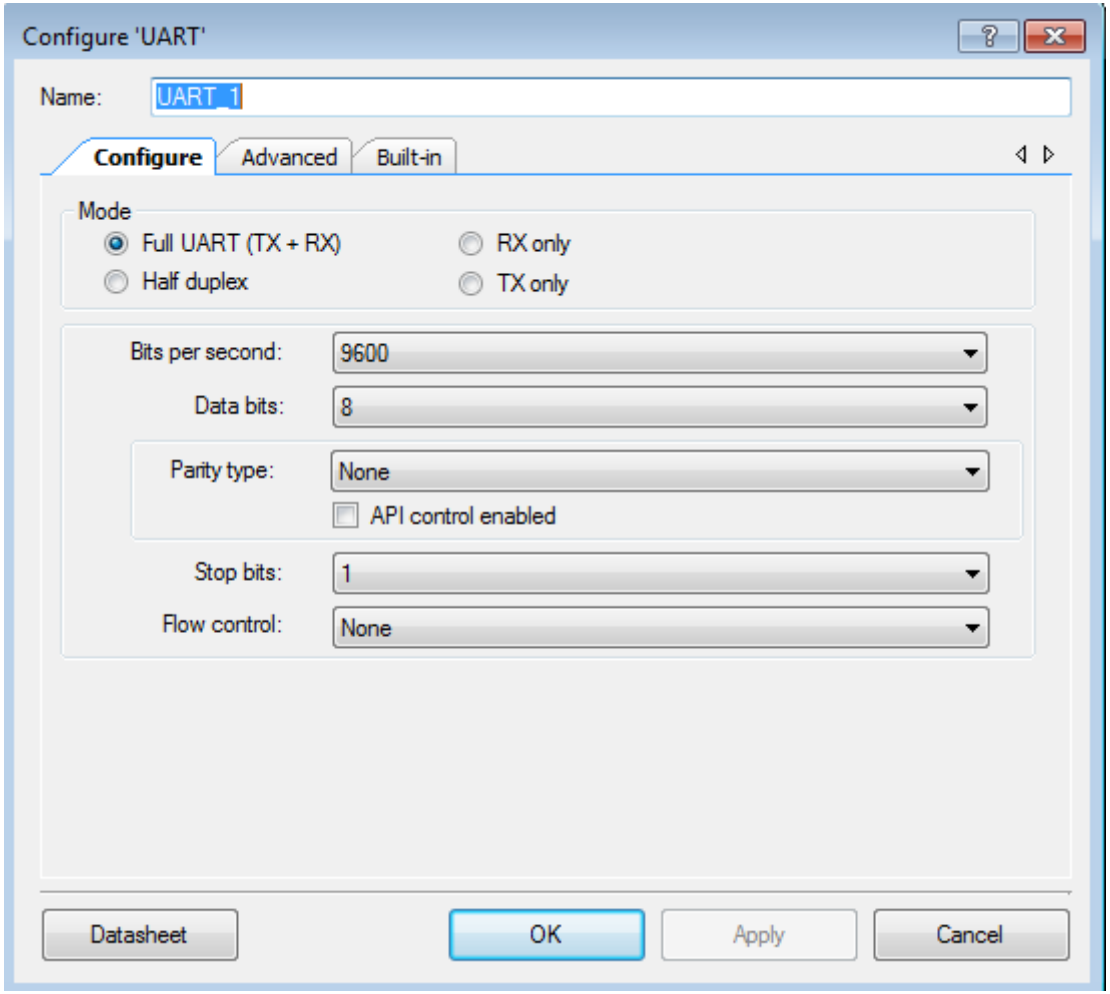
- Single-ended mód
- Single-sample mód
- 20 bites felbontás
- Belső referencia (1.024V a P0[3] lábon átvezetve amin egy 1 µF szűrő-kondenzátor is található)
- 0 – 6.144 V bemeneti tartomány (buffer áthidalással)



UART konfigurálása

- Full duplex mód (9600, 8, 1, N)
- Nem kérünk megszakítást (Advenced lapon)
- Leválasztott KitProg esetén kössük össze:

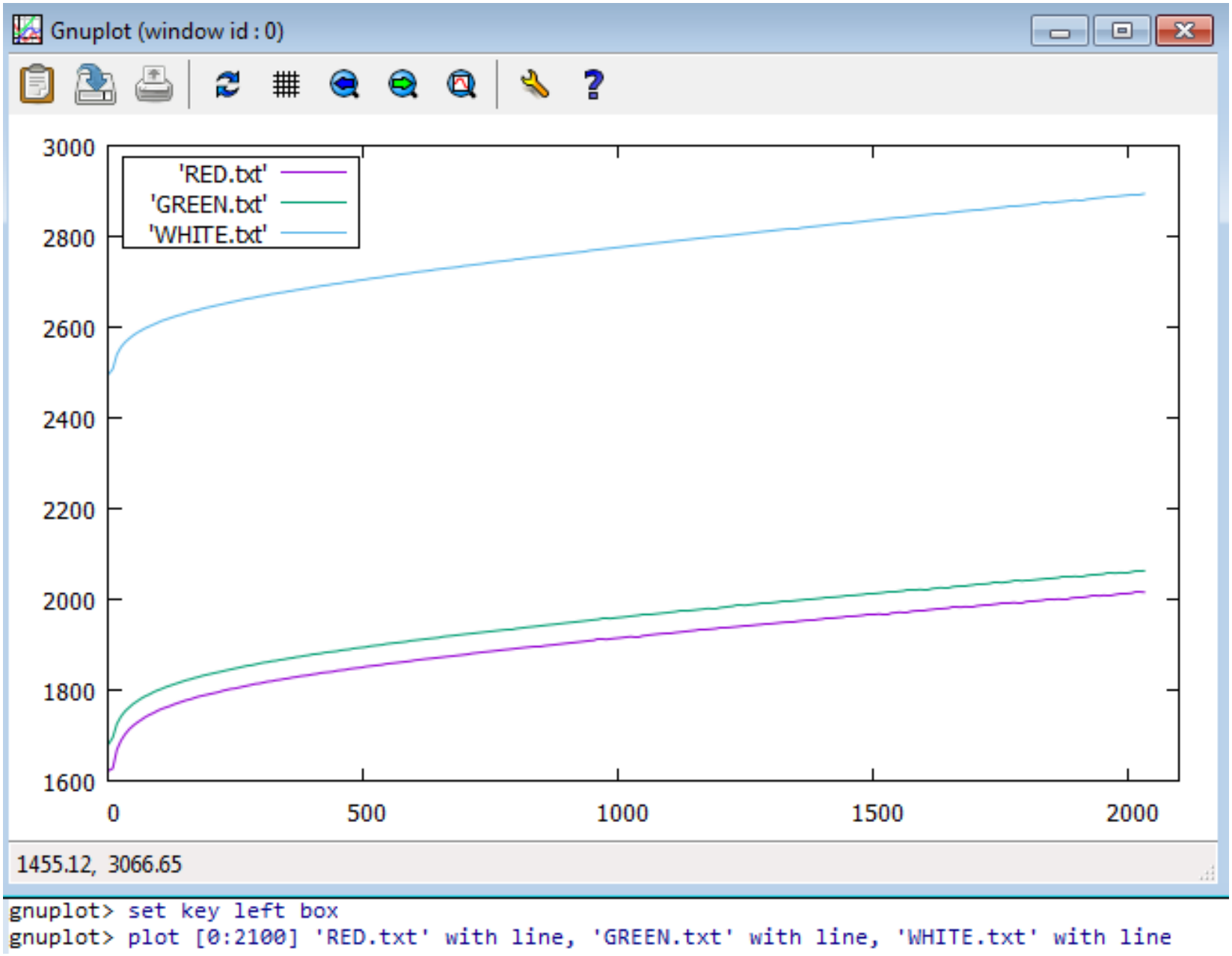
KitProg Target
Tx ------ Rx (P12[6])
Rx ------ Tx (P12[7])



main.c

```
#include "project.h"
#include "stdio.h"
char buf[64];
int main(void) {
    uint32 result, milliVolts;
    uint8 Ch, i;
    UART_1_Start();
    IDAC8_1_Start();
    IDAC8_1_SetValue(0);
    ADC_DelSig_1_Start();
    CyGlobalIntEnable; /* Ez kell a Single Sample módhoz! */
    ADC_DelSig_1_StartConvert();
    UART_1_PutString("\r\nCOM Port Open\r\n");

    for(;;) {
        Ch = UART_1_GetChar();
        /* Set flags based on UART command */
        if(Ch == 's' || Ch == 'S') {
            for(i=0;i<255; i++) {
                IDAC8_1_SetValue(i);
                ADC_DelSig_1_StartConvert();
                while(!ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_RETURN_STATUS));
                result = ADC_DelSig_1_GetResult32();
                milliVolts = ADC_DelSig_1_CountsTo_mVolts(result);
                sprintf(buf, "%d  %lu\r\n", (int)i*8, milliVolts);
                UART_1_PutString(buf);
            }
        }
    }
}
```

LED-ek karakterisztikája



Hőmérés diódával – elvi alapok

- A diódán átfolyó I áram, a diódán eső V feszültség és a T hőmérséklet között az alábbi összefüggés áll fenn:

ahol:

$$I = I_s \, e^{\frac{V}{\eta V_T}}$$

$\mathbf{V_T}$ – a termikus feszültség ($V_T = kT/q$)

$\boldsymbol{k}$ – a Boltzmann-állandó (1.38×10^{-23} Joule/Kelvin)

$\boldsymbol{q}$ – az elemi töltés (1.602×10^{-19} Coulomb)

$\mathbf{T}$ – az abszolút hőmérséklet, $\boldsymbol{\eta}$ – 1–2 közötti konstans

- Két, különböző áramnál mérve:

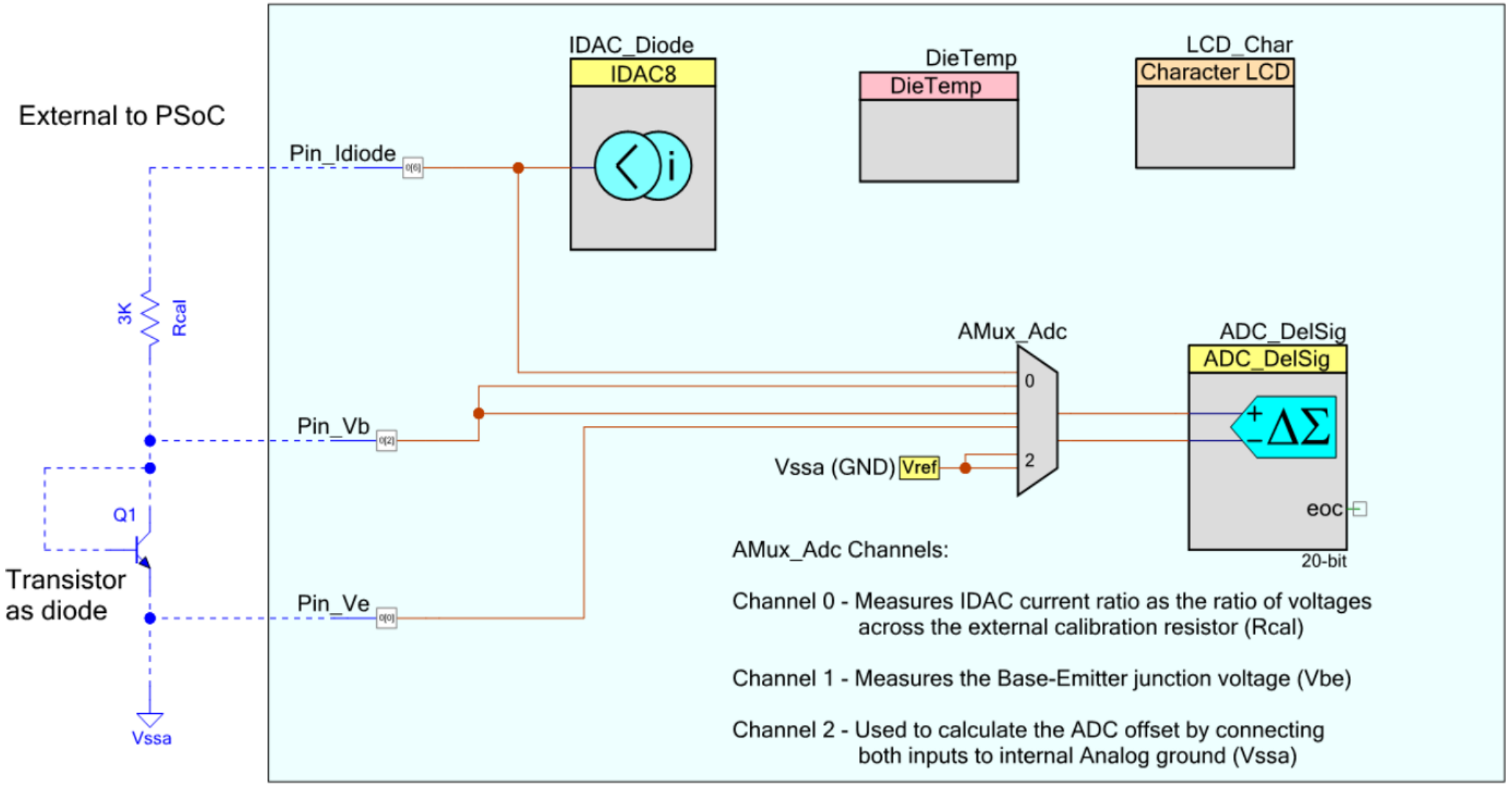
$$I_1 = I_s \, e^{\frac{V1}{\eta V_T}} \qquad I_2 = I_s \, e^{\frac{V_2}{\eta V_T}} \quad \Rightarrow \quad \frac{I_2}{I_1} = N = e^{\frac{V_2 - V_1}{\eta V_T}}$$

- Átrendezve és behelyettesítve:

$$T\,(in\ Kelvin) = (V_2 - V_1) * \frac{q}{\ln(N) * k * \eta}$$

A_SingleDiode

- Ez a projekt az AN60 590 – Temperature Measurement with a Diode c. kiadvány 1. példaprogramja, amit adaptáltunk CY8CKIT-059-re



Konfigurálás, lábkiosztás

- **IDAC8:** pozitív áram, 0 – 255 µA, Fast mód
- **ADC:** 20-bites, differenciális mód, 0 – ±1.024 V, buffer bypassed
- **Amux:** 3 db. differenciális csatorna
- **LCD:** semmi extra, lábkiosztás P2[7:1], ugyanúgy, mint a korábbi mintapéldákban
- **DieTemp:** nem kell konfigurálni

- **Kivezetések bekötése:**
  - **Pin_Idiode – P0[6]** (az IDAC8 kimenete)
  - **Pin_Vb – P0[2]** (a tranzisztor összekötött kollektora és bázisa)
  - **Pin_Ve – P0[0]** (a tranzisztor emittere) – ezt a pontot össze kell kötni a kártya Vssa/GND pontjával is!
  - A **P0[6] – P0[2]** közé kötött ellenállás 1 kΩ – 4.7 kΩ közötti érték legyen!

main.c

```
#include <device.h>
/* "sprintf" routine used in project is in stdio.h file */
#include <stdio.h>
/* Diode temperature measurement functions are declared in below header file */
#include "SingleDiode.h"

/* Global variable used to store the PSoC die temperature when IDAC
   was last calibrated. IDAC recalibrated for every 10 C change in PSoC temperature
*/
int16 IdacCalib_temperature = 0;

int main()
{
    float diode_temperature;
    int16 psoc_temperature;
    int16 temperature_diff;
    char str[7]; /* String used to display floating point temperature value  */

    CyGlobalIntEnable; /* enable global interrupts. */

    /* Initialize and Start the LCD */
    LCD_Char_Start();

    /* Initialize the components for diode temperature measurement */
    InitializeDiodeMeasurement();

    /* Store the PSoC die temperature when IDAC was calibrated initially. */
    DieTemp_GetTemp(&IdacCalib_temperature);
```

main.c (folytatás)

```
    for(;;)
    {
        /* Get diode temperature and print on LCD */
        diode_temperature = GetDiodeTemperature();
        sprintf(str, "%.1f", diode_temperature);
       LCD_Char_Position(0,0);
        LCD_Char_PrintString(str);

        /* Measure the PSoC die temperature */
        DieTemp_GetTemp(&psoc_temperature);

    /* Check to see if the PSoC temperature change requires a IDAC recalibration */
        temperature_diff = psoc_temperature - IdacCalib_temperature;
        if((temperature_diff > CALIB_TEMP_INTERVAL_POS) ||
            (temperature_diff < CALIB_TEMP_INTERVAL_NEG)) {
            CalibrateIdac();

            /* Set Calibrated temperature to current psoc temperature */
            IdacCalib_temperature = psoc_temperature;
        }
    }
}
```

SingleDiode.c

```
#include <device.h>  /* Include all generated header files  */
#include "math.h"    /* Included for using logarithm function */
#include "SingleDiode.h"
volatile float multiply_factor = 0;

void CalibrateIdac() {
    float CurrentRatio;
    uint8 loop = 0;
    int32 Adc_I1 = 0;
    int32 Adc_I2 = 0;
    int32 Adc_Offset = 0;
    AMux_Adc_Select(AMUX_ADC_OFFSET_CHANNEL);
    for(loop = 0; loop < NUM_OF_SAMPLES_CALIB; loop++) {
        ADC_DelSig_StartConvert();
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        Adc_Offset = Adc_Offset +  ADC_DelSig_GetResult32();
        ADC_DelSig_StopConvert();
    }

    AMux_Adc_Select(AMUX_ADC_RESISTOR_CHANNEL);
    IDAC_Diode_SetValue(IDAC_LOW_CURRENT);     /* Bias current I1 */
    for(loop = 0; loop < NUM_OF_SAMPLES_CALIB; loop++)
    {
        ADC_DelSig_StartConvert();
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        Adc_I1 = Adc_I1 + ADC_DelSig_GetResult32();
        ADC_DelSig_StopConvert();
    }
```

SingleDiode.c (folytatás)

```
    IDAC_Diode_SetValue(IDAC_HIGH_CURRENT);   /* Bias current I2 */
    /*  Measure voltage across calibration resistor for high current (I2)
        for specified number of samples   */
    for(loop = 0; loop < NUM_OF_SAMPLES_CALIB; loop++)     {
        ADC_DelSig_StartConvert();
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        Adc_I2 = Adc_I2 + ADC_DelSig_GetResult32();
        ADC_DelSig_StopConvert();
    }

    /* Set IDAC current to zero when not using the IDAC */
    IDAC_Diode_SetValue(0);

    /* Subtract the ADC offset voltage from the calibration resistor
       voltage readings and calculate the current ratio - I2/I1 */
    Adc_I1 = Adc_I1 - Adc_Offset;
    Adc_I2 = Adc_I2 - Adc_Offset;
    CurrentRatio = (float)Adc_I2 / (float)Adc_I1;

    /* Calculate the diode multiplication factor, which is a aglobal variable */
    multiply_factor = (Q_BY_K * ADC_VOLTAGE_RESOLUTION)/(IDEALITY_FACTOR *
                      (log(CurrentRatio)));
}
```

SingleDiode.c (folytatás)

```
float GetDiodeTemperature() {
    uint8 loop;
    int32 Vbe1_count = 0;
    int32 Vbe2_count = 0;
    float Vbe_diff_count;
    float diode_temperature;
    AMux_Adc_Select(AMUX_ADC_DIODE_CHANNEL); /* Select VBE channel as ADC input */
    IDAC_Diode_SetValue(IDAC_LOW_CURRENT); /* Bias current I1 */
    for(loop = 0; loop < NUM_OF_SAMPLES_VBEDIFF; loop++) {
        ADC_DelSig_StartConvert();
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        ADC_DelSig_StopConvert();
        Vbe1_count = Vbe1_count +  ADC_DelSig_GetResult32();
    }

    IDAC_Diode_SetValue(IDAC_HIGH_CURRENT); /* Bias current I2 */
    for(loop = 0; loop < NUM_OF_SAMPLES_VBEDIFF; loop++) {
        ADC_DelSig_StartConvert();
        ADC_DelSig_IsEndConversion(ADC_DelSig_WAIT_FOR_RESULT);
        ADC_DelSig_StopConvert();
        Vbe2_count = Vbe2_count +  ADC_DelSig_GetResult32();
    }
    IDAC_Diode_SetValue(0); /* Set IDAC current to zero when not using the IDAC */
    Vbe_diff_count = ((float)(Vbe2_count - Vbe1_count)) /
                     ((float)(NUM_OF_SAMPLES_VBEDIFF));
    diode_temperature =(Vbe_diff_count * multiply_factor) -
                        ZERO_DEGREE_CELSIUS_KELVIN;
    return(diode_temperature);
}
```

CY8CKIT-059 fejlesztői kártya

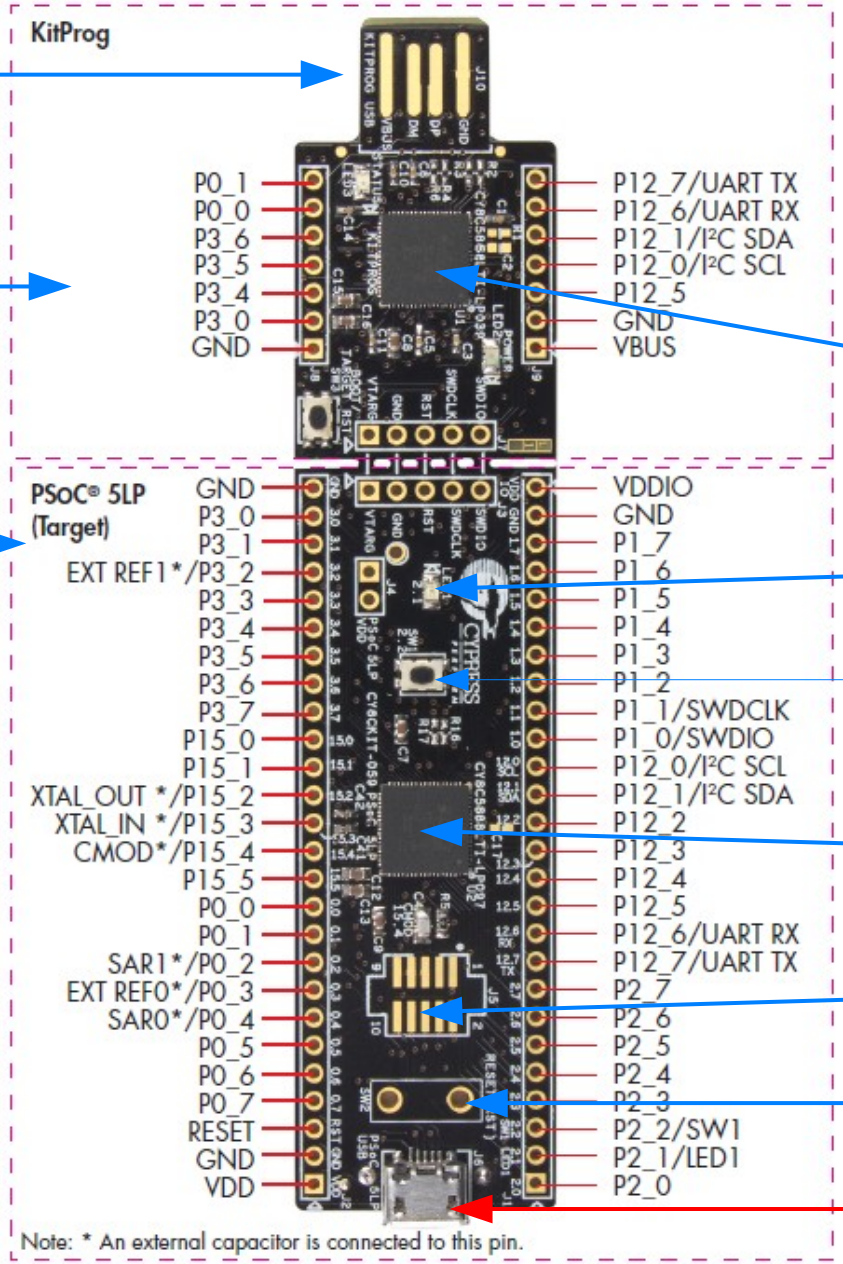
USB csatlakozás a PC-hez

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