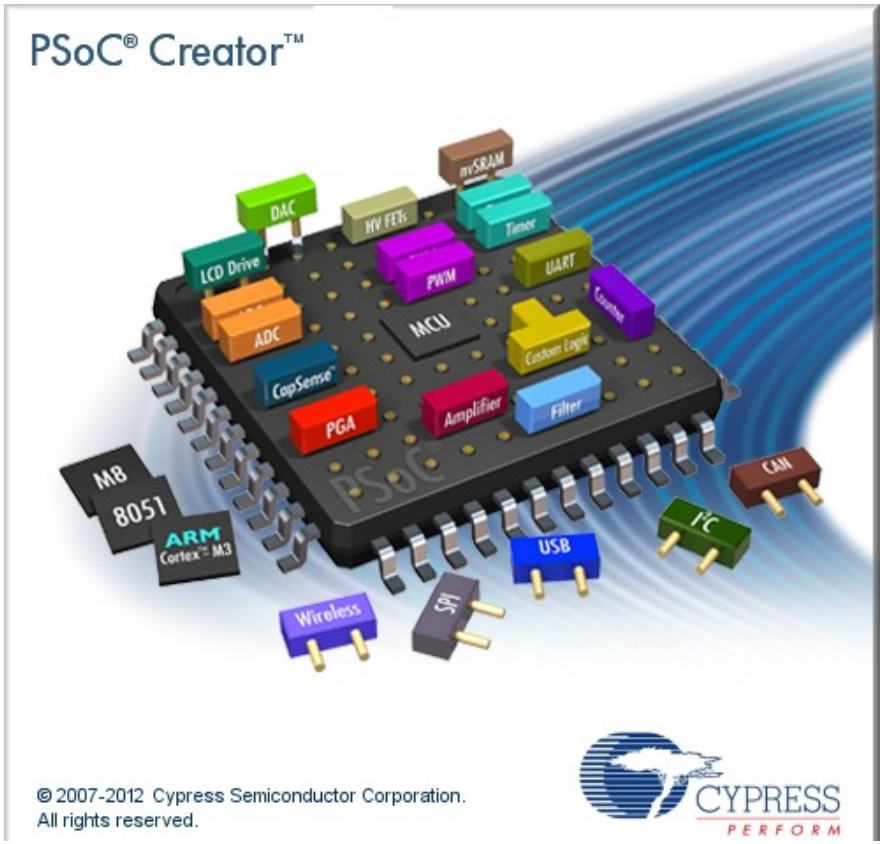
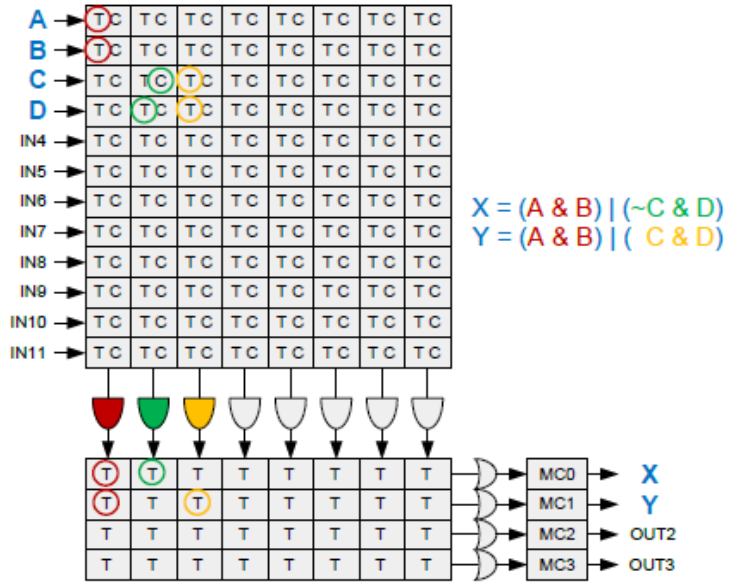
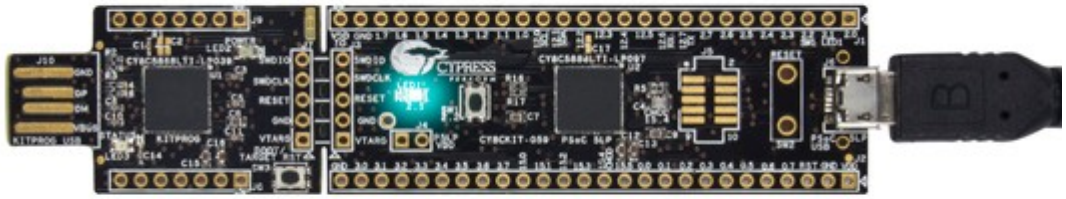
Újrakonfigurálható eszközök







16. Cypress PSOC 5LP – új alkatrészeket definiálunk Verilog nyelven

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
- Cypress: AN82 250 – Implementing Programmable Logic Designs with Verilog
- Cypress: Just Enough Verilog for PSoC

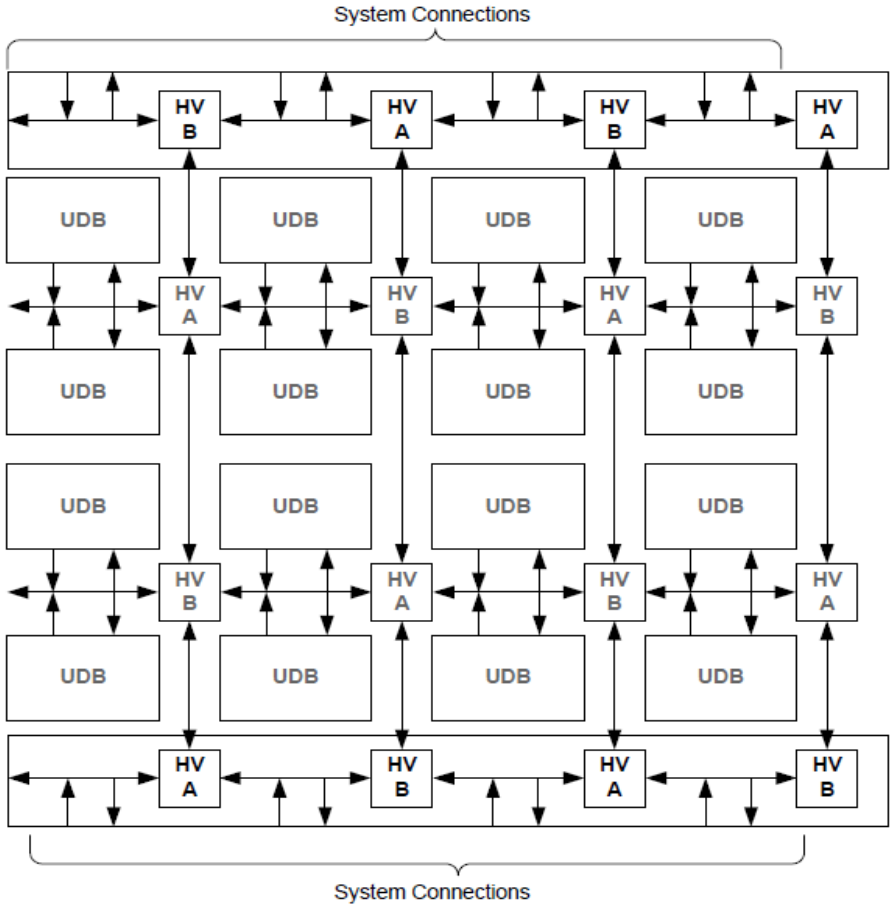
PSOC = Programozható System-on-chip

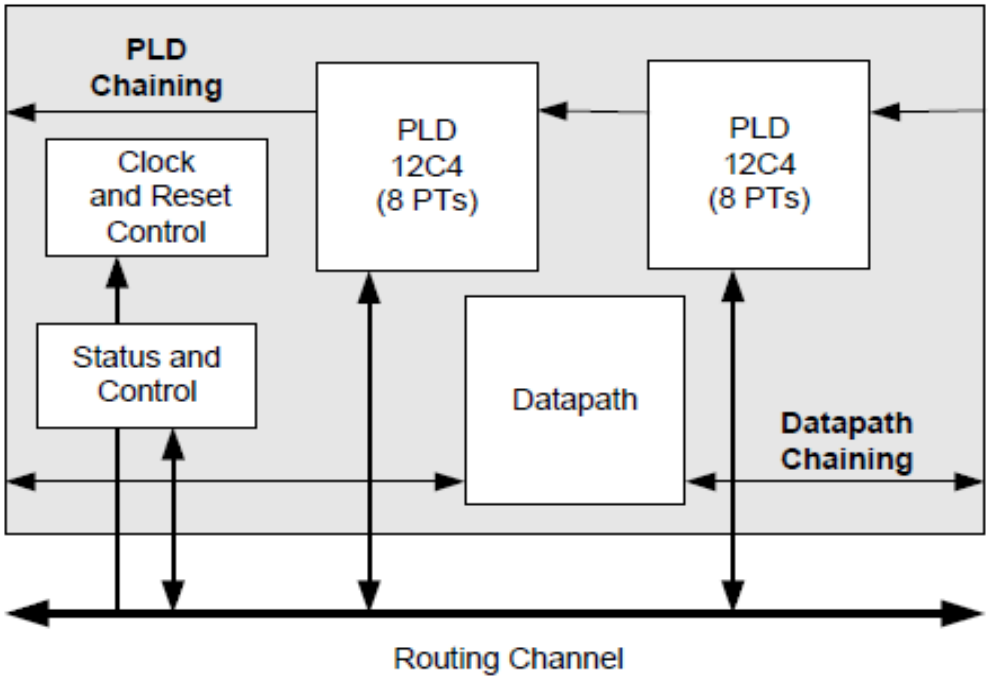
- A **PSOC** mikrovezérlőkben a számtalan fix funkciós periféria mellett az univerzális digitális blokkokból *egyedi perifériákat*, vagy *kiegészítő logikai áramköröket* is kialakíthatunk.

- A **PSOC Creator** gazdag alkatészkínálata számos előredefiniált perifériát, illetve alapelemet kínál (logikai kapuk, flip-flopok, számláló- és shift regiszterek, stb.)
- Speciális igény esetén ***magunk is definiálhatunk új alkatrészeket,*** részegységeket, amelyek működését Verilog HDL nyelven írhatjuk le.
  - **PLD blokkokból:** kombinációs vagy szekvenciális áramkörök
  - **Datapath elemekből:** regiszterek, aritmetikai/logikai műveleteket végző áramkörök
- A mai előadásban az AN82 250 alkalmazási mintapélda nyomán a PLD blokkok használatát mutatjuk be, s egy 4 bites számlálót tervezünk.

Univerzális digitális blokkok (UDB)

- Az Univerzális Digitális Blokkokból (UDB) a **CYC8C5888** PSOC 5LP mikrovezérlő 24 db-ot tartalmaz
- Minden UDB 2 db 12 bemenetű és 4 makrocella kimenetű szorzat-tag előállító áramkört (PLD) és egy „adatút” modult tartalmaz.





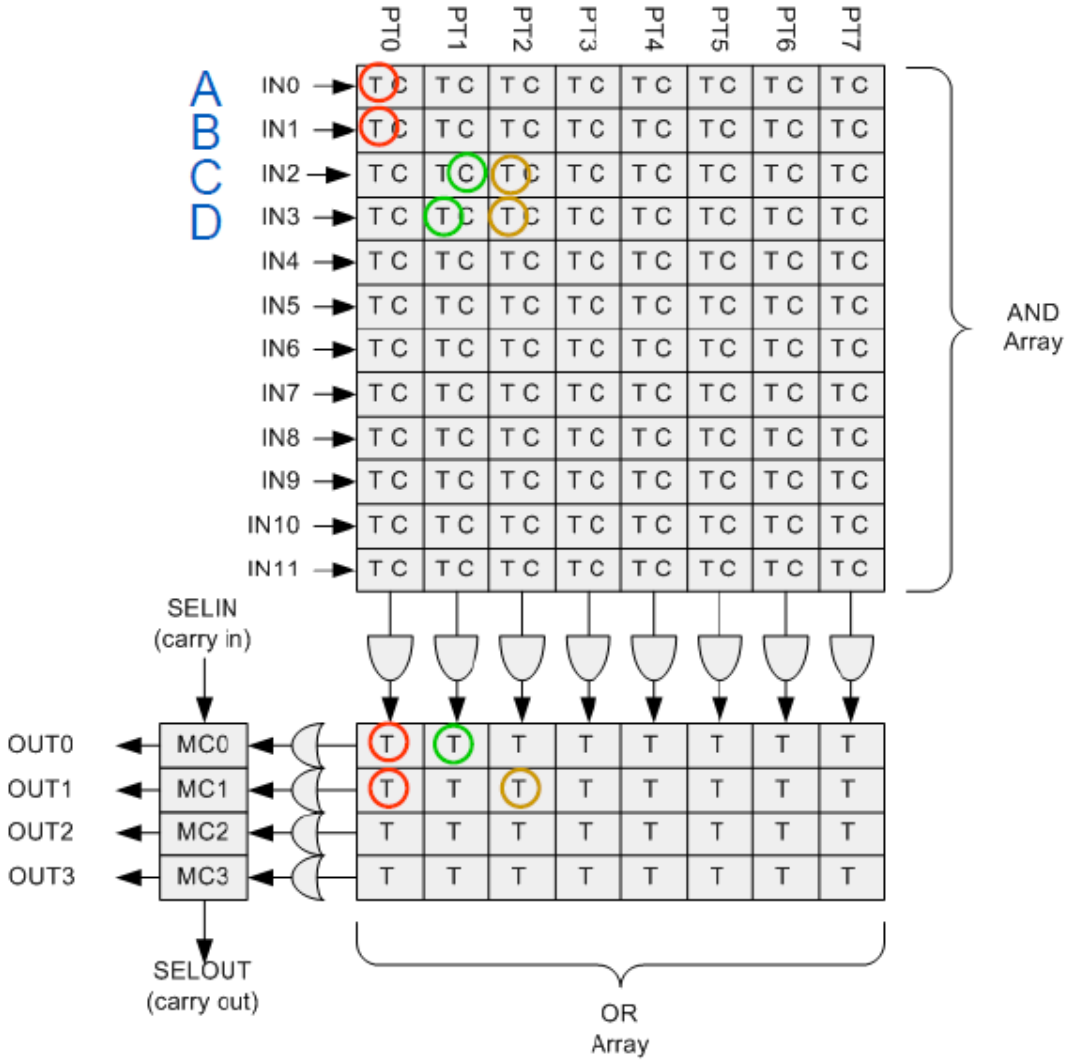
Egy 12C4 PLD blokk felépítése

- PLD blokk jellemzők:
  - 12 bemenet
  - 8 szorzat tag (product term)
  - 4 makrocella kimenet

$$X = (A\&B) \mid (\sim C\&D)$$

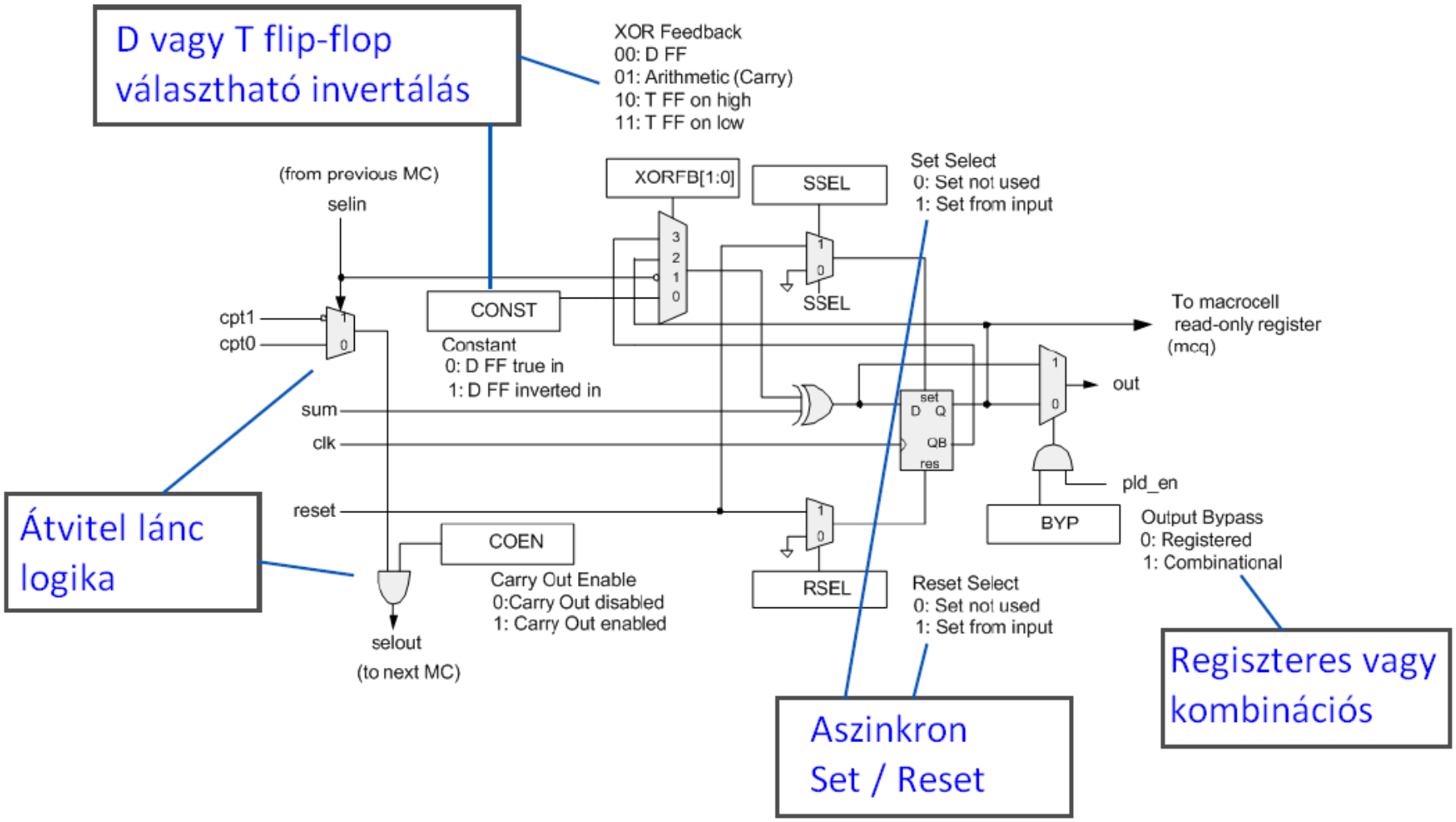
$$Y = (A\&B) \mid (C\&D)$$

  - TC = True vagy Complement
  - AND = logikai ÉS kapuk
  - OR = logika VAGY kapuk
  - MC = makrocella



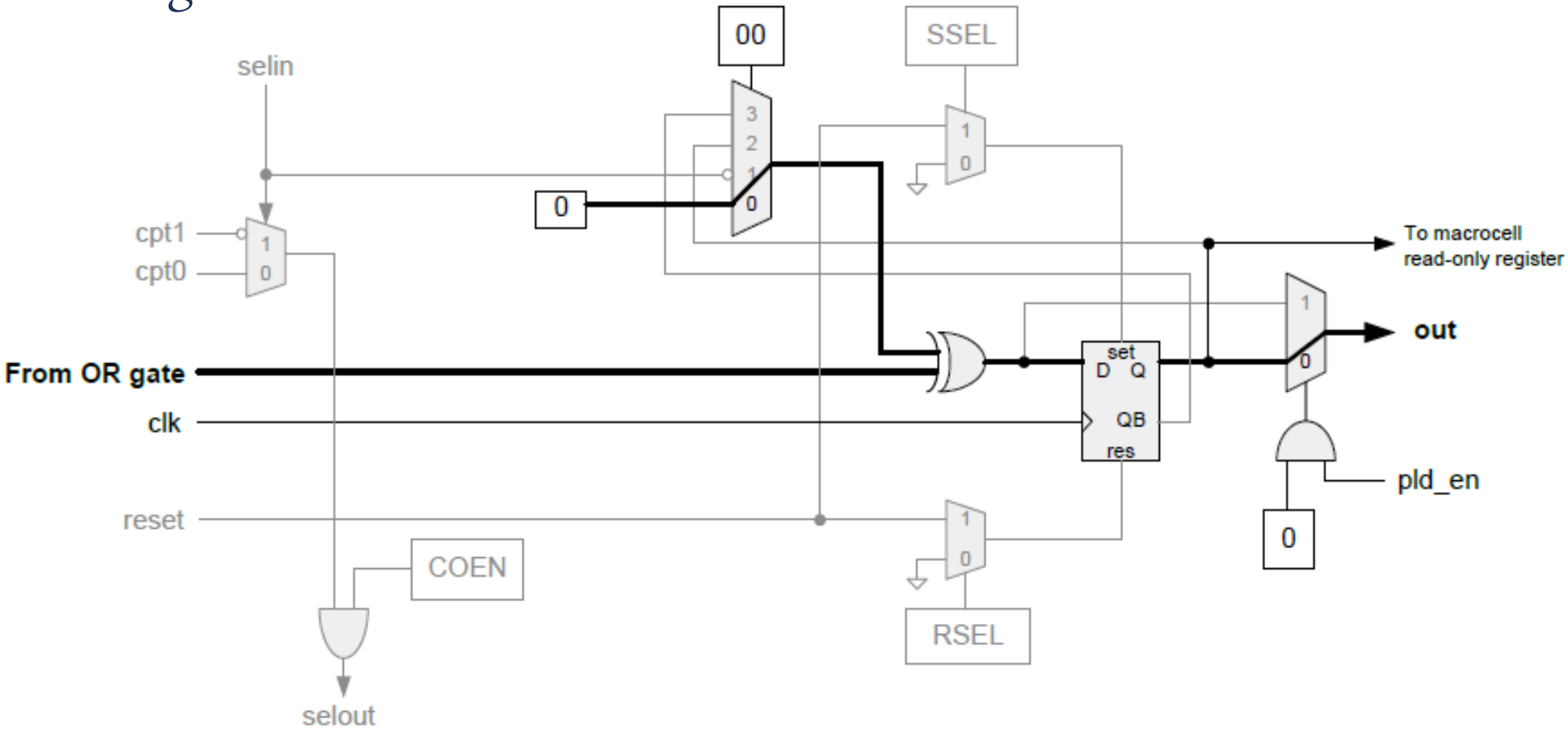
Egy makrocella felépítése

- A makrocella működhet regiszterként vagy kombinációs logikai áramkörként



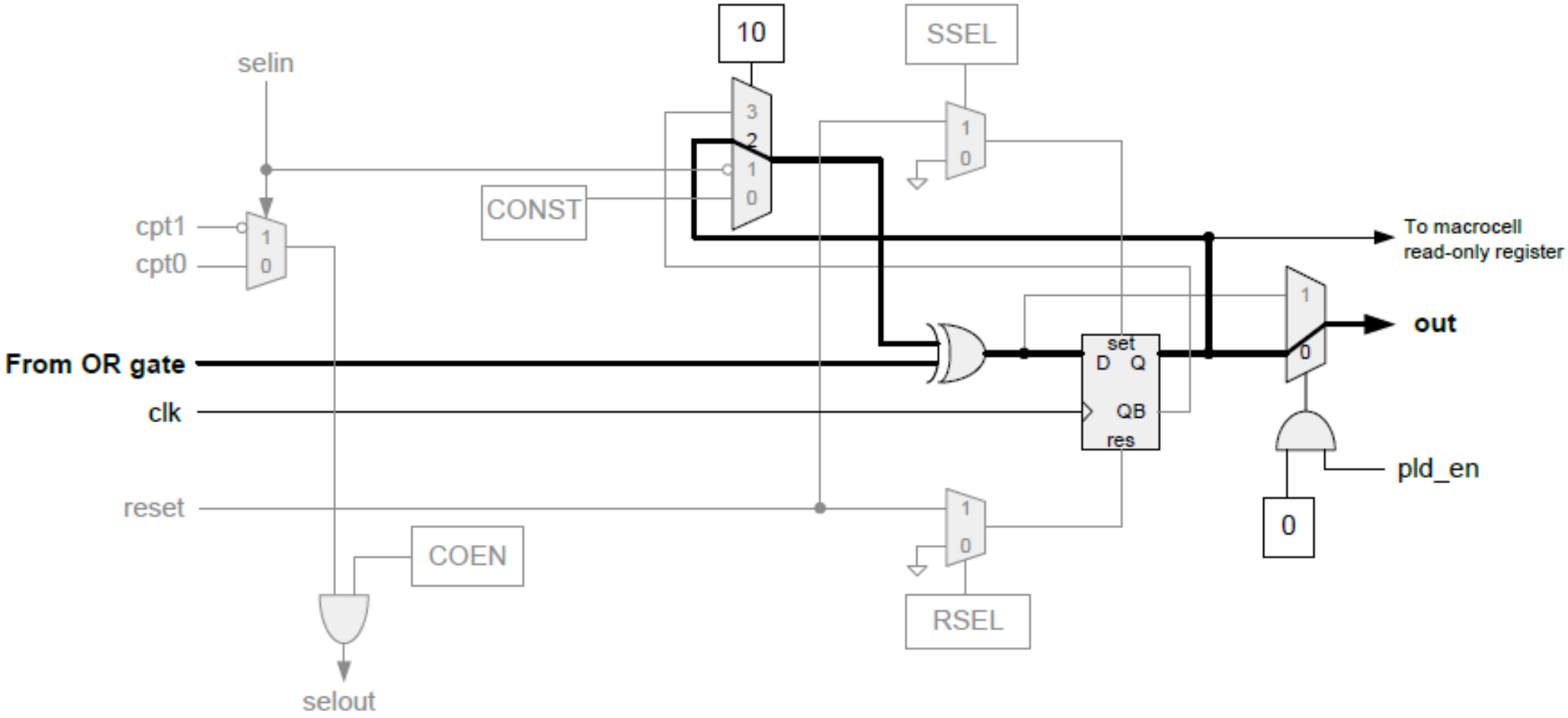
D flip-flopnak konfigurált makrocella

- A macrocellák lényegében flip-flopok, járulékos kombinációs logikai áramkörökkel kiegészítve
- Az alábbi ábrán az adatáramlás útját láthatjuk, D tárolónak konfigurált makrocella esetén

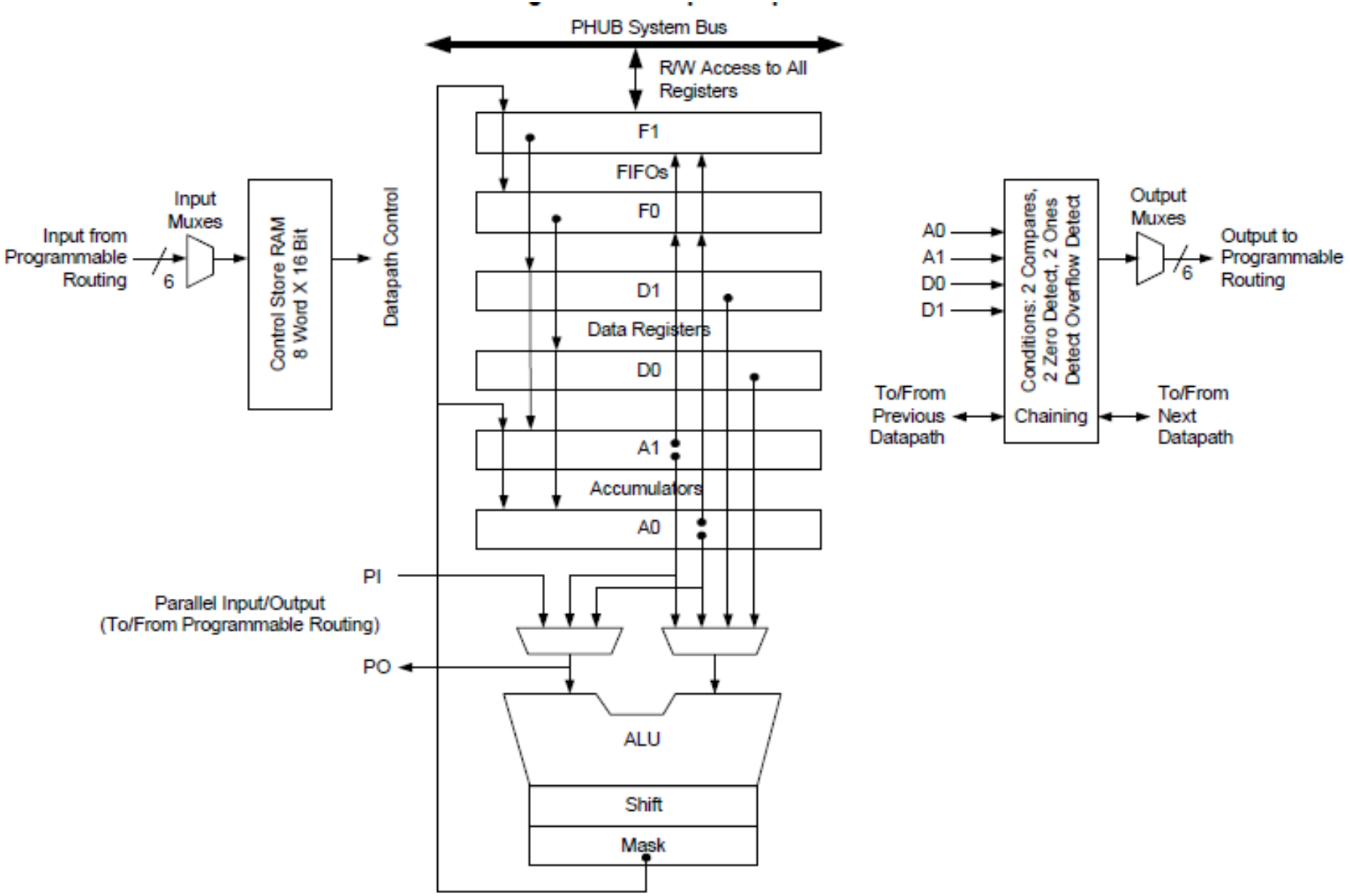


T flip-flopnak konfigurált makrocella

- Az alábbi ábrán egy T tárolónak konfigurált makrocellát láthatunk
- A T flip-flop magas szintű T bemenőjel esetén az órajel felfutásánál a kimenetét negálja (állapotot vált)



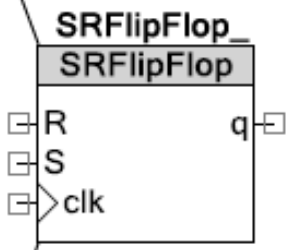
Adatút (Data path)



PLD alkatrészek definiálása Verilog nyelven

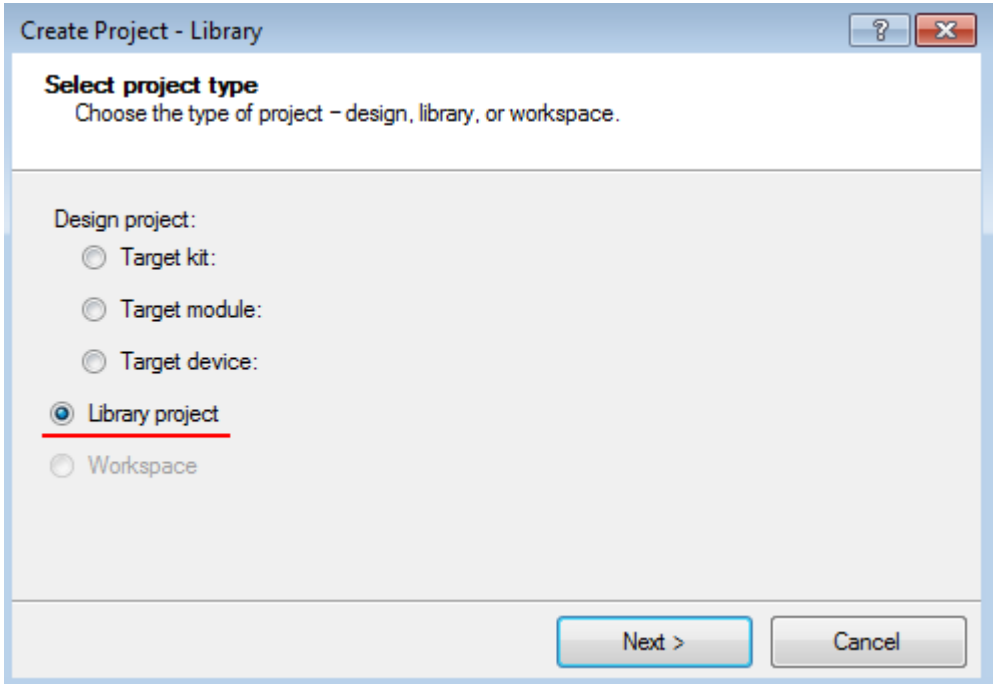
- Bár a PSOC Creator sokféle kész alkatrészt kínál, előfordulhat, hogy magunknak kell definiálni egy újat
- Létrehozunk egy programkönyvtár projektet
- Definiáljuk a ki- és bemeneteket, s az esetleges paramétereket
- Az így létrehozott alkatrész vázhoz készítünk egy Verilog állományt, ami definiálja a működést

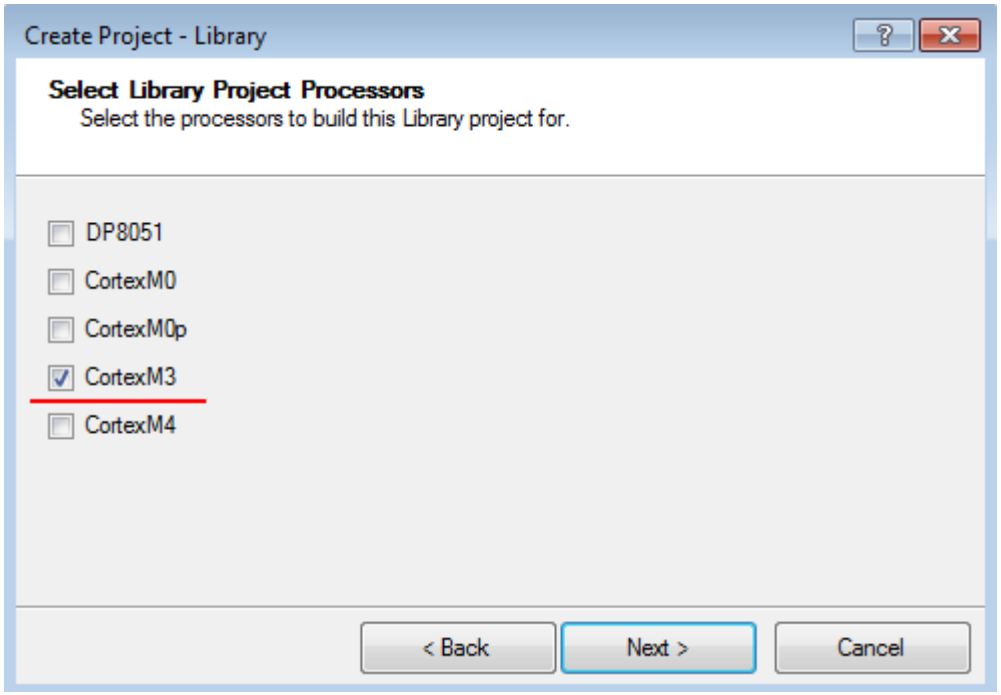
```
17  module SRFlipFlop ( output reg q,
18                      input  clk,
19                      input  R,
20                      input  S);
21
22      always @ (posedge clk)
23      begin
24          case({R,S})
25              2'b00: // preserve state
26              begin
27                  q <= q;
28              end
29
30              2'b10: //Reset
31              begin
32                  q <= 1'b0;
33              end
34
35              2'b01: //Set
36              begin
37                  q <= 1'b1;
38              end
39
40              2'b11: // illegal state
41              begin
42                  q <= q;
43              end
44
45              default: // never get here
46              begin
47                  q <= q;
48              end
49          endcase
50      end
51  endmodule
```



Saját alkatrészkönyvtár létrehozása

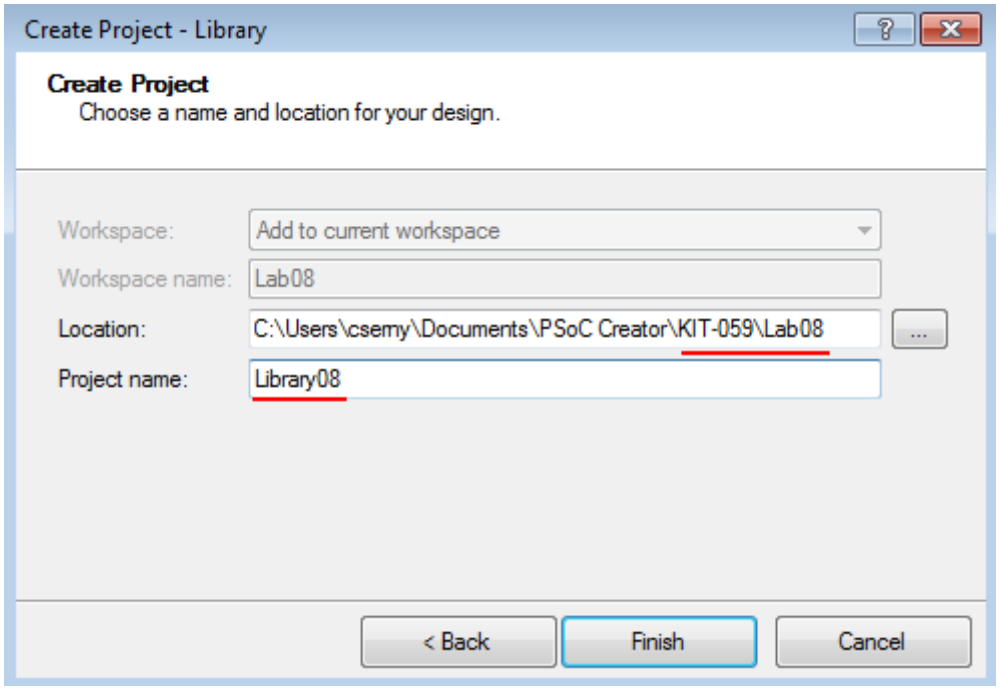
- Az alábbiakban részletesen bemutatjuk egy alkatrészkönyvtár és azon belül egy alkatrész létrehozásának lépéseit. A részletes bemutatást az indokolja, hogy az elmúlt évek során a PSOC Creator újabb változataiban a menük és dialógusablakok megváltoztak, emiatt nehéz követni a régebben kiadott útmutatókat
- Hozzunk létre egy új munkaterületet, s abban egy új projektet, ami könyvtár típusú legyen! Next után Cortex-M3 kijelölés és Next…





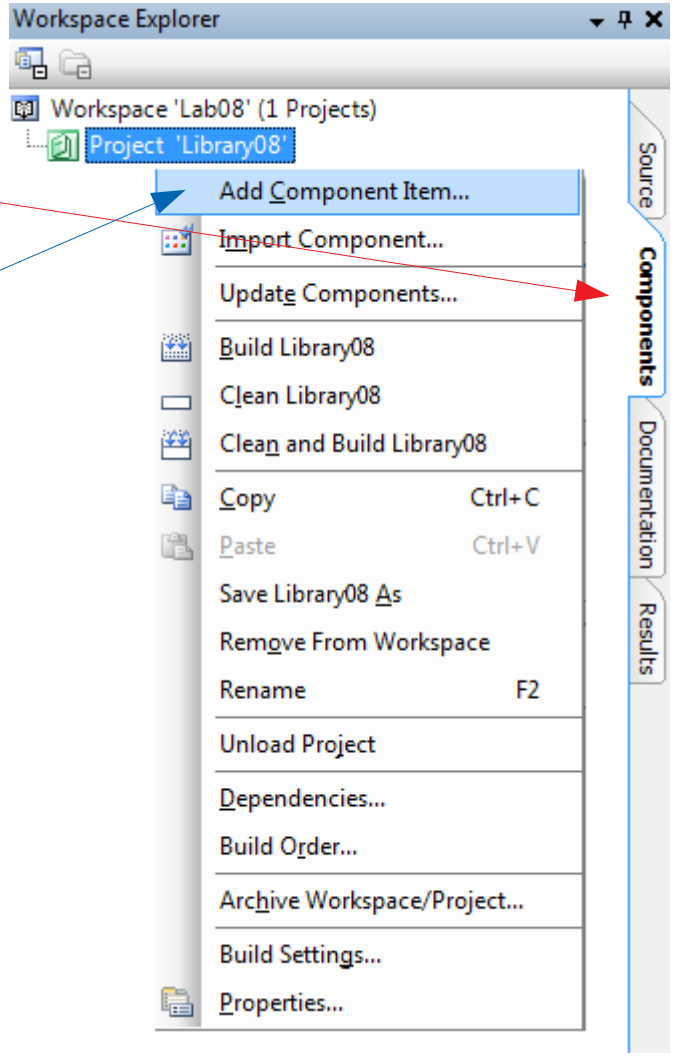
A könyvtár projekt helye és neve

- Adjuk meg a projekt helyét (az előzőekben létrehozott munkaterület mappáját tallózzuk be)!
- Adjuk meg az új projekt nevét (az alábbi példában Library08, a mintaprojektünkben pedig majd AN82 250 Components lesz)!



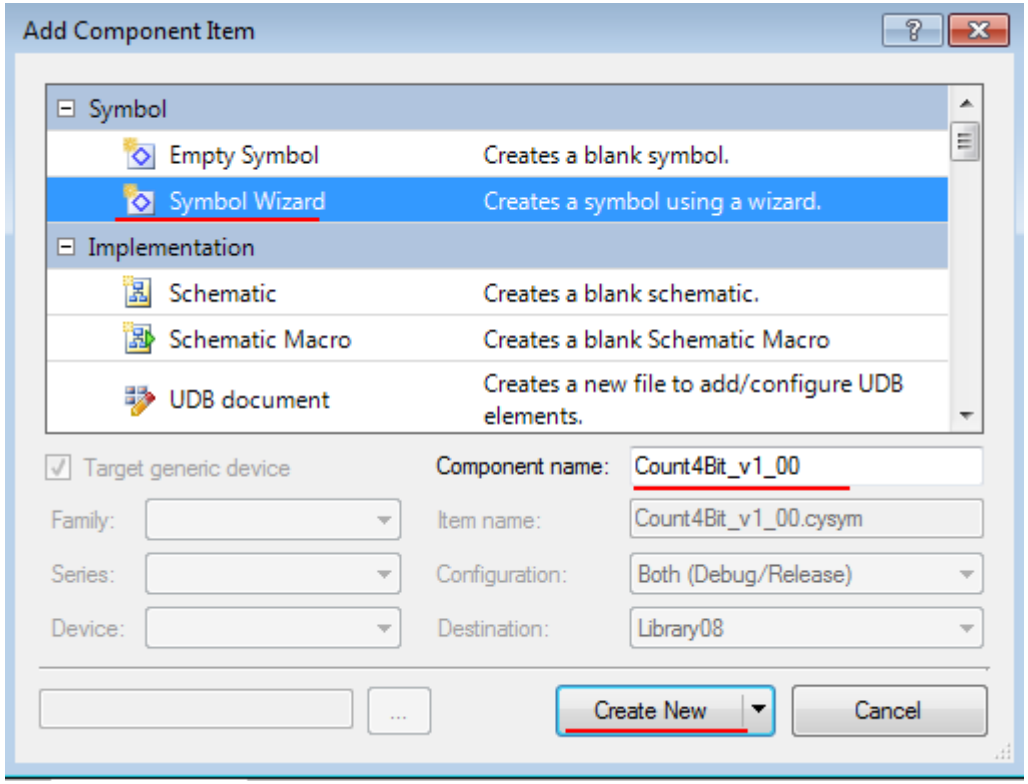
Új alkatrész létrehozása

- A következő oldalakon egy 4 bites számláló definiálásának lépéseit mutatjuk be
- A Components fület válasszuk ki!
- Jobb gombbal kattintunk, majd a felbukkanó menüben az **Add Component Item** menüpontot jelöljük ki!



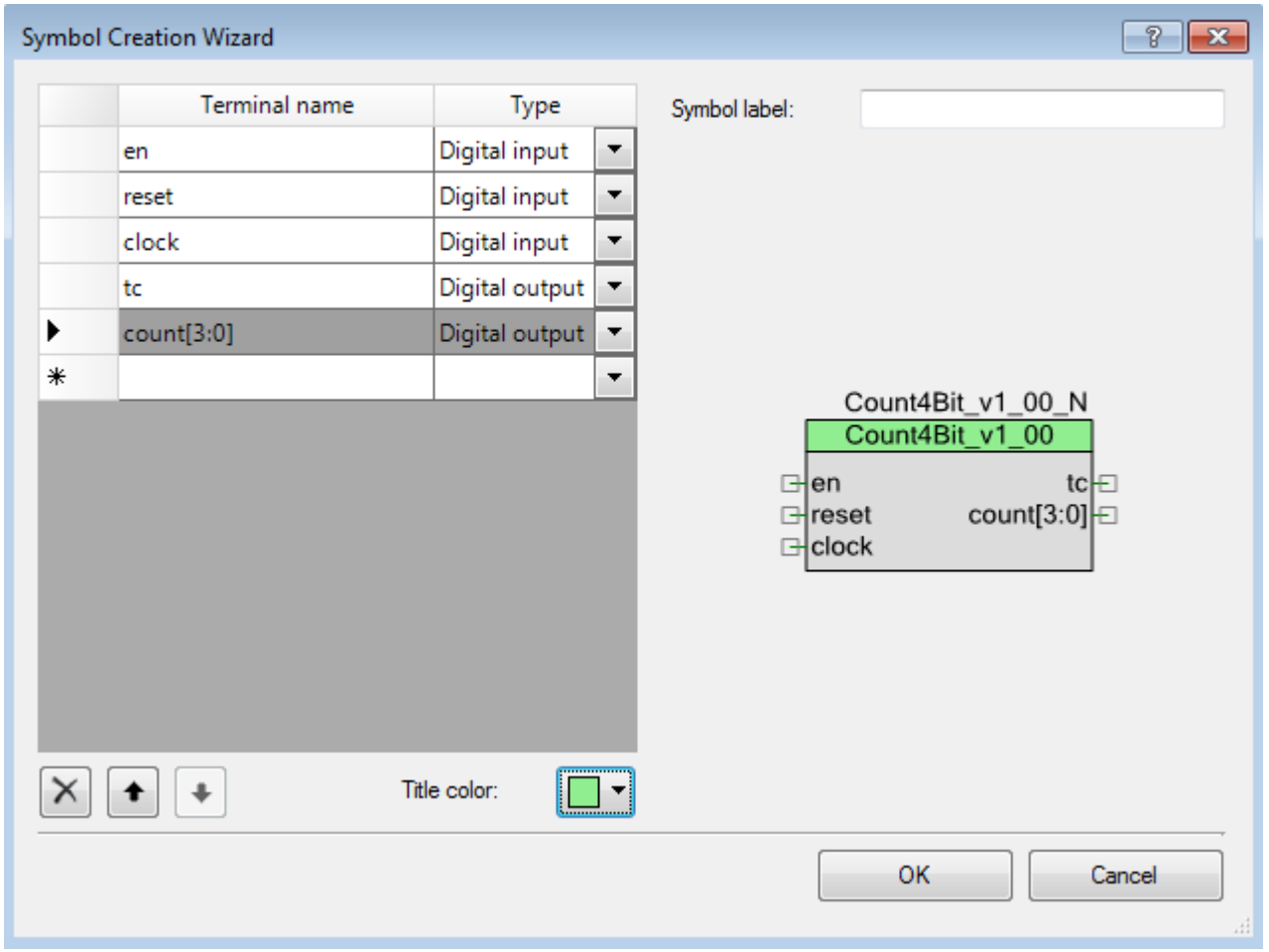
Alkatrész neve és létrehozásának módja

- Az új alkatrész neve legyen **Count4Bit_v1_00** (célszerű a névbe belefoglalni a verziószámot is)!
- Az alkatrész rajzjelének létrehozásához a **Symbol Wizard**-ot fogjuk használni, ezt jelöljük ki, majd kattintsunk a Create New gombra!



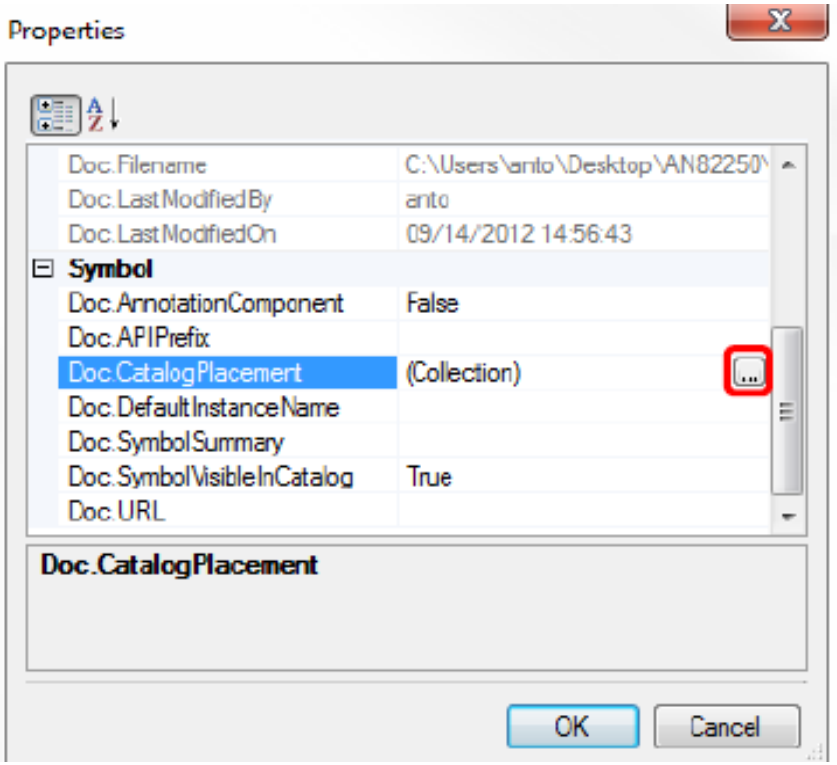
A rajzjel varázsló használata

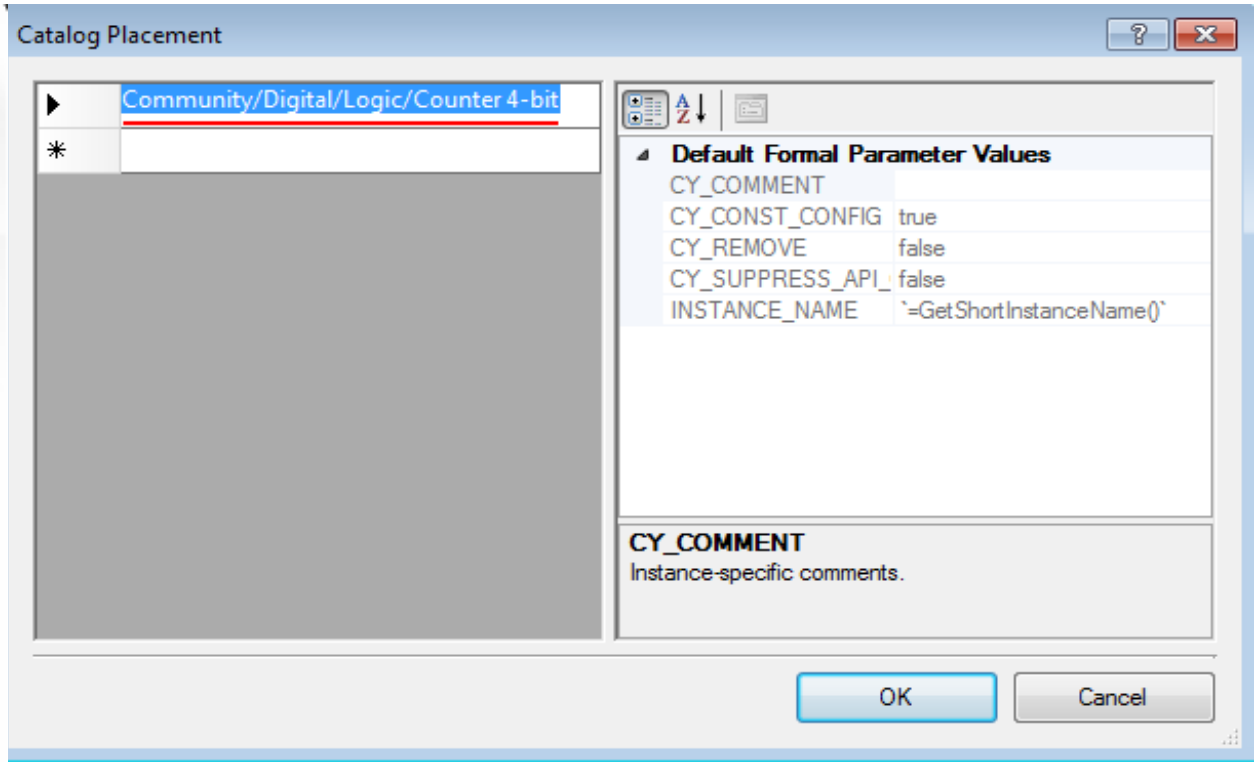
- A varázsló segítségével definiáljuk a három digitális bemenetet és a két digitális kimenetet!
- Adjuk meg az alkatrész rajzjele fejlécének színét!



Az alkatrész besorolása

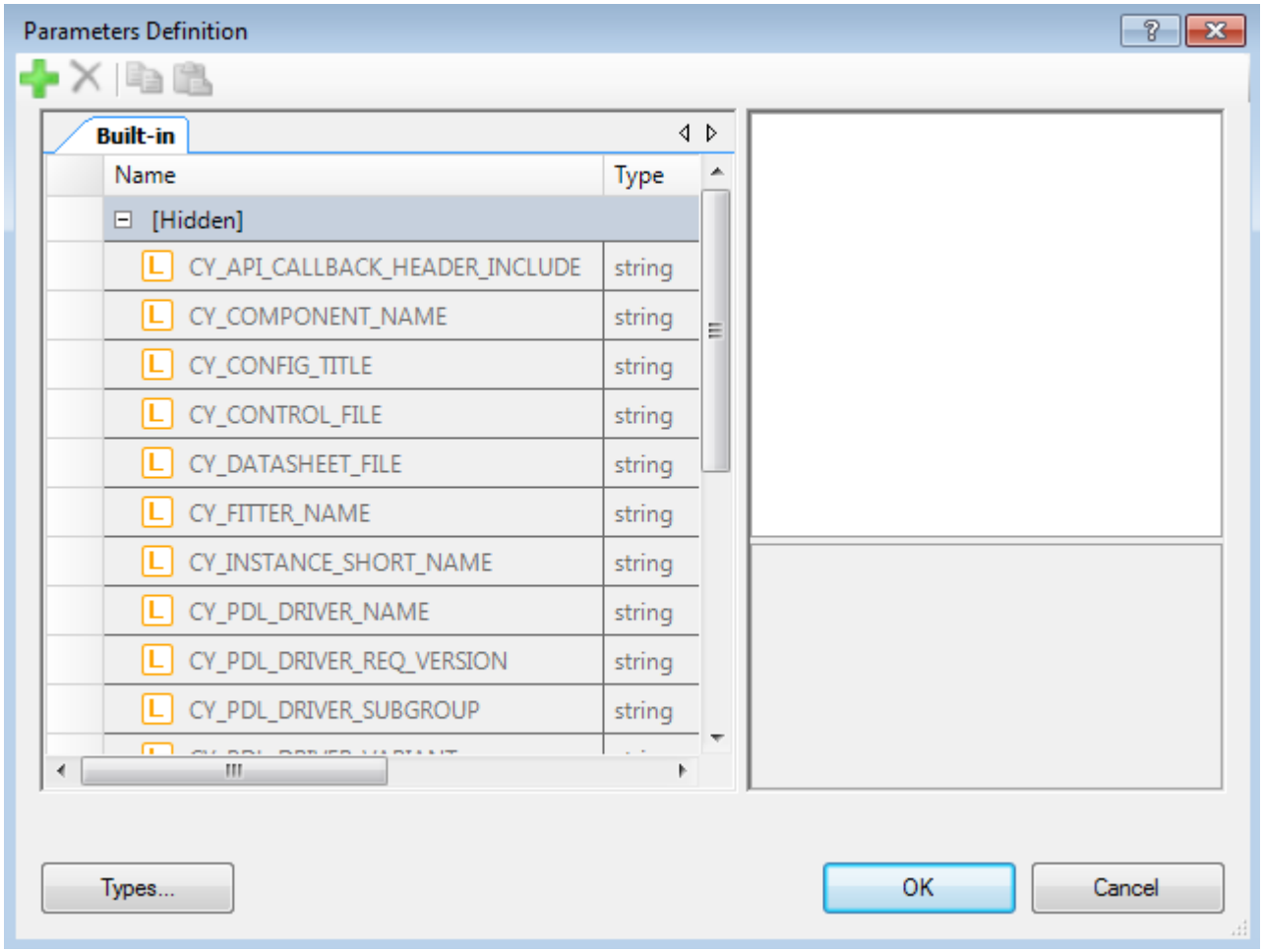
- Az alkatrész rajzjelének lapján (**Count4Bit_v1_00.cysym** állomány) jobb klikk után **Properties** menüpontot válasszuk!
- A **Symbol/Doc.CatalogPlacement** paraméternél kattintsunk a három pontra!
- Írjuk be ezt: **Community/Digital/Logic/Counter 4-bit**





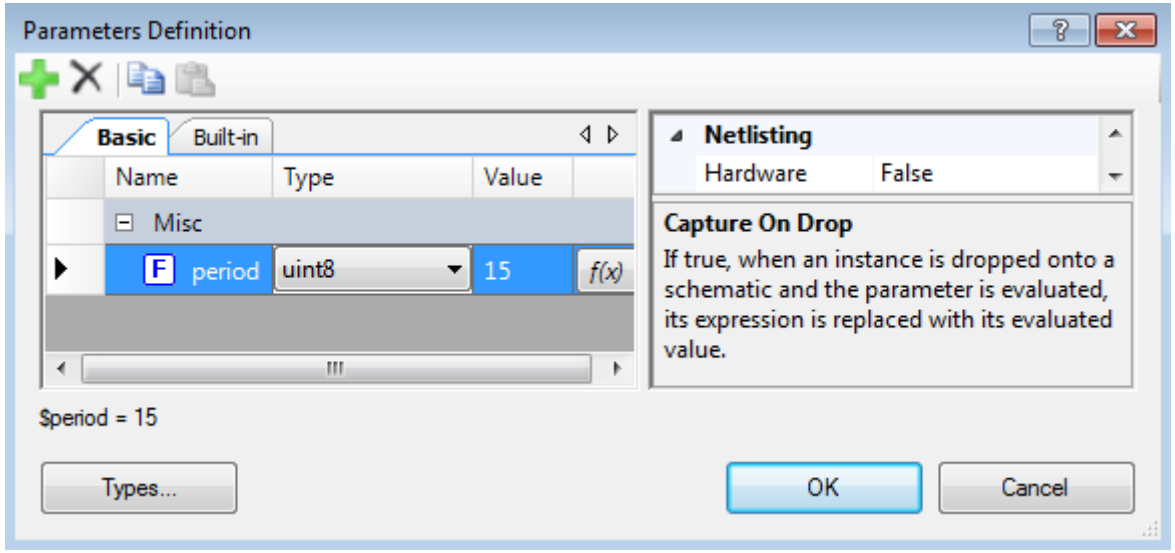
Alkatrész paraméter hozzáadása

- Az alkatrész rajzjelének lapján (**Count4Bit_v1_00.cysym** állomány) jobb klikk után a **Symbol parameters** menüpontot válasszuk ki!
- Új paraméter megadásához kattintsunk a zöld keresztre!



Alkatrész paraméter hozzáadása

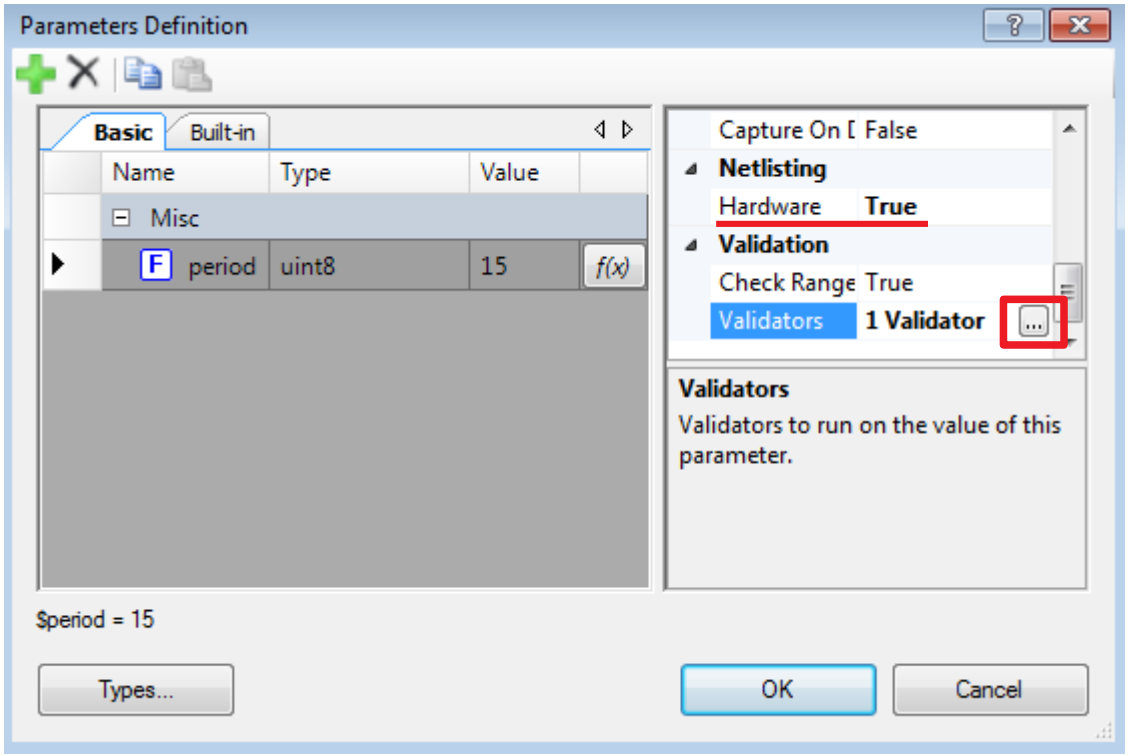
- A paraméter neve legyen **period** (azaz periódus)
- A típusa legyen **uint8**
- Az értéke legyen **15** (négybites számláló 0 – 15 között számlál)



- **Megjegyzés:** a paraméter lehet formális (F) vagy lokális (L), itt hagyjuk meg az alapértelmezett „formális” beállítást!

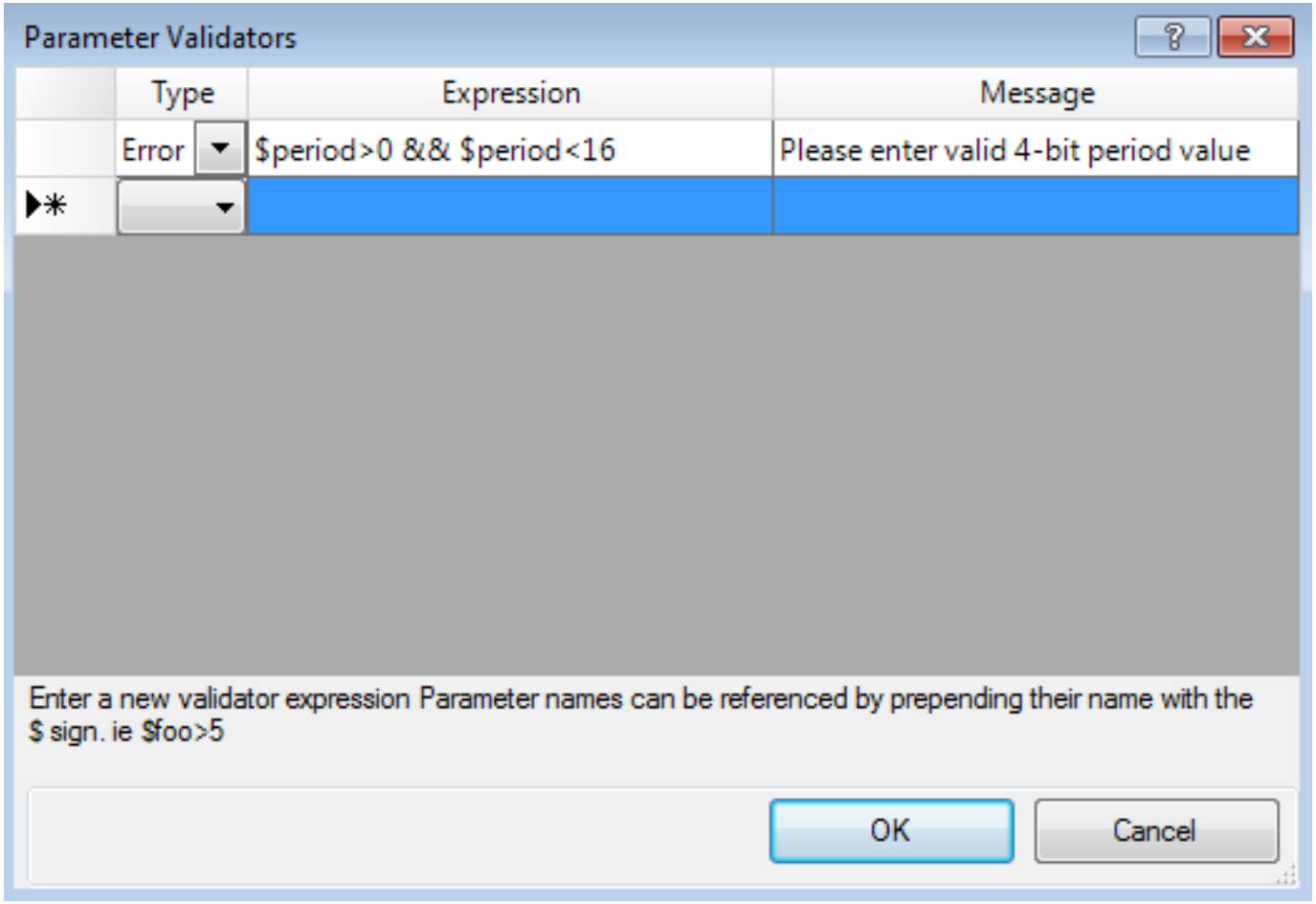
Paraméter érvényességének ellenőrzése

- A paraméter tulajdonságoknál Netlisting/Hardware **True** legyen!
- A **Validation** szekcióban pedig definiáljunk új validatort (érvényességi feltételt) a három gombra kattintva!
  Erre azért van szükség, hogy a projekt konfiguráláskor megadott érték érvényességét ellenőrizni tudjuk.



Paraméter érvényességének ellenőrzése

- Az érvényesség feltétele: **`$period>0 && $period<16`**
- Ha a feltétel nem teljesül, ez legyen a hibaüzenet:
  Please enter valid 4-bit period data



Verilog keretprogram generálása

- Az alkatrész lapján a **Generate Verilog** menüpontot válasszuk ki!

```
//`#start header` -- edit after this line, do not edit this line

`include "cypress.v"
//`#end` -- edit above this line, do not edit this line
// Generated on 04/24/2018 at 19:49
// Component: Count4Bit_v1_00
module Count4Bit_v1_00 (
    output  [3:0] count,
    output  tc,
    input   clock,
    input   en,
    input   reset
);
    parameter period = 15;

//`#start body` -- edit after this line, do not edit this line

//      Ide írhatjuk a működtető kódot!

//`#end` -- edit above this line, do not edit this line
endmodule
//`#start footer` -- edit after this line, do not edit this line
//`#end` -- edit above this line, do not edit this line
```

```
output reg [3:0] count,
output reg tc,
```

Írjuk át **reg** típusra!

Count4Bit implementálása

- Szinkronizált eszközöknél minden tevékenység az órajelhez kötött, ezért minden kód az alábbi blokkban helyezkedik el:

```
always @ (posedge clock)
begin
. . .
end
```

- Van egy szinkronizált **Reset** bemenetünk, amely nullázza a kimeneteket:

```
if(reset)
begin
  count <= 4'b0000;
  tc <= 1'b0;
end
```

- A számláló csak akkor számlál, ha az **Enable** bemenet aktív:

```
if(en)          /* start counting */
begin
. . .
end
else            /* preserve state */
begin
  count <= count;
  tc <= tc;
end
```

Count4Bit implementálása

- Ha a számláló elérte a számlálás felső korlátját, akkor nullázódik, a **tc** kimenet pedig aktív állapotba kerül.:

```
if(count == period)
begin
  tc <= 1'b1;
  count <= 4'b0000;
end
```

- Egyébként pedig minden órajelre növeljük a számláló értékét, s töröljük a tc kimenetet

```
else
begin
  count <= count + 1;
  tc <= 1'b0;
end
```

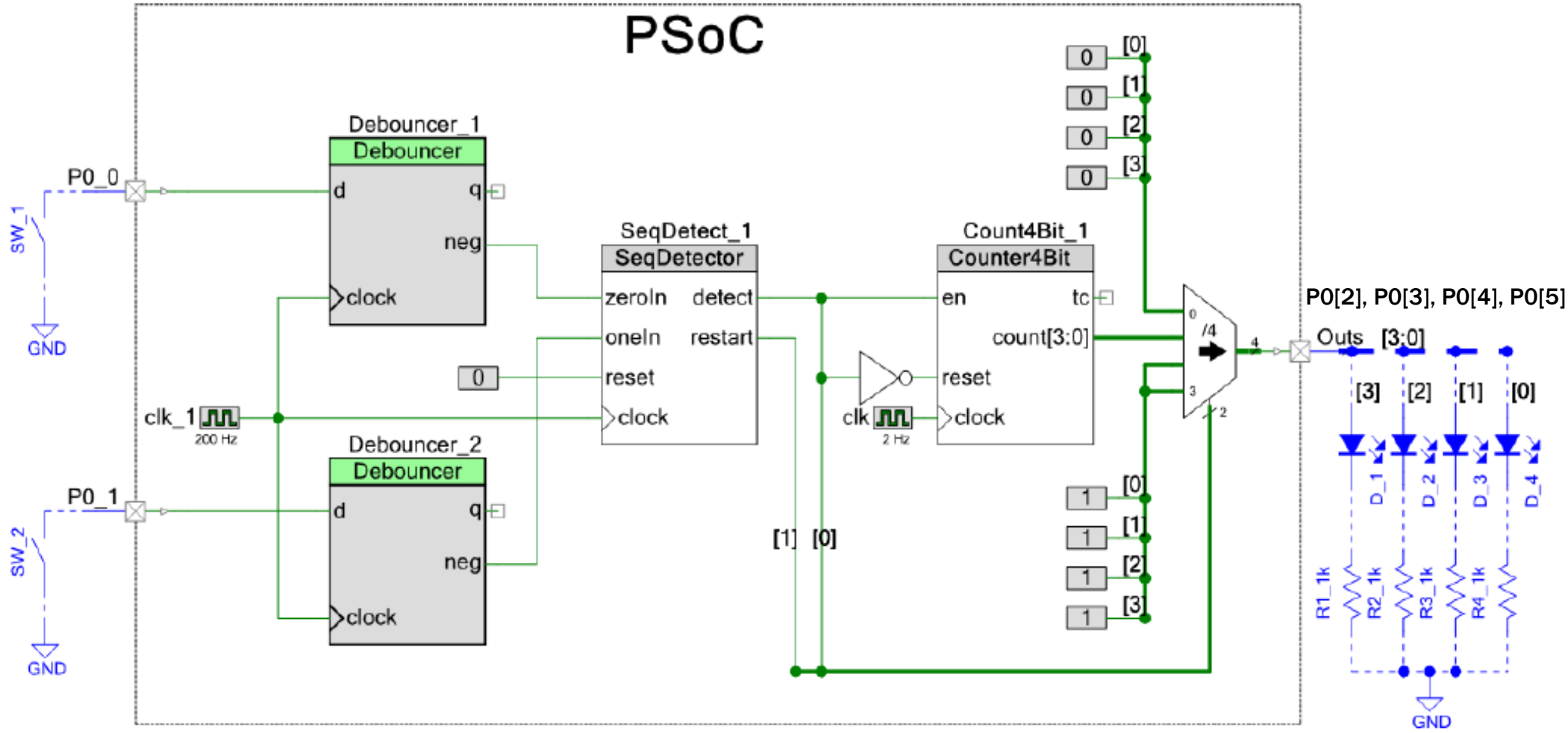
Count4Bit implementálása

- Az összesített Verilog kód végeredményben így néz ki:

```
module Count4Bit_v1_00 (
  output reg [3:0] count,
  output reg tc,
  input   clock,
  input   en,
  input   reset
);

  parameter period = 0;

//`#start body` -- edit after this l
not edit this line

  always @ (posedge clock)
  begin
    if(reset)
    begin
      count <= 4'b0000;
      tc <= 1'b0;
    end
    else
    begin
    if(en)
    begin
      if(count == period)
      begin
        tc <= 1'b1;
        count <= 4'b0000;
      end
      else
      begin
        count <= count + 1;
        tc <= 1'b0;
      end
    end
    else
    begin
      count <= count;
      tc <= tc;
    end
  end
end

//`#end` -- edit above this line, do not
edit this line
endmodule
```

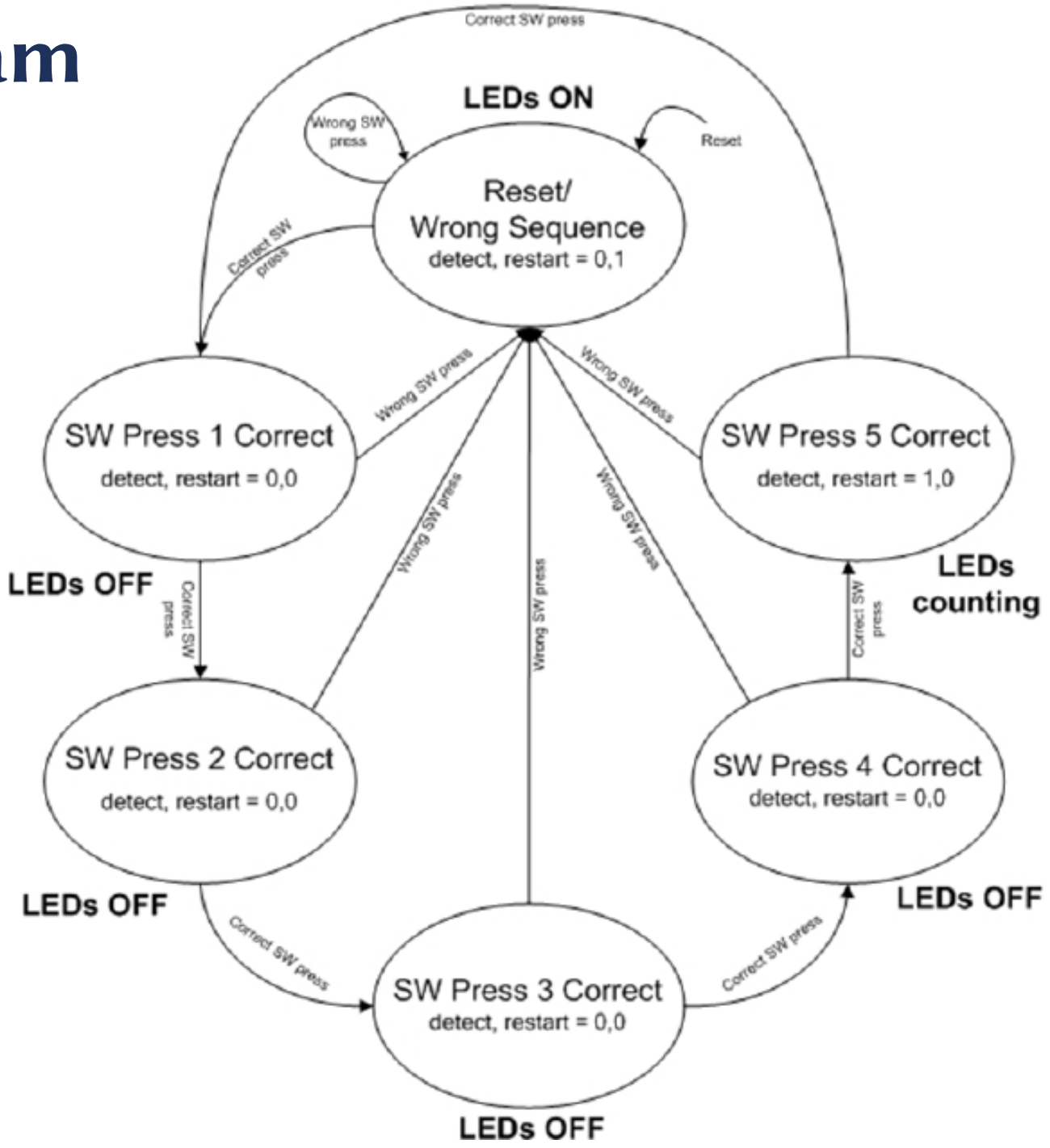
AN82250 mintaprojekt

- Az alábbi kapcsolásban egy ötjegyű bináris számot írhatunk be (SW1 a 0, SW2 az 1 bevitelére szolgál). Ha eltaláltuk a számot, elindul a számláló. **SeqDetector** egy 6 állapotú véges állapotgép.



Állapotdiagram

- Kezdetben minden LED világít
- Helyes számjegy beírásakor a LED-ek kialszanak
- Hibás számjegy esetén visszaugrás a kezdőállapotba
- Helyes számsor esetén a számlálás elindul
- Újabb gombnyo-másra újrakezdés a 2. pontnál



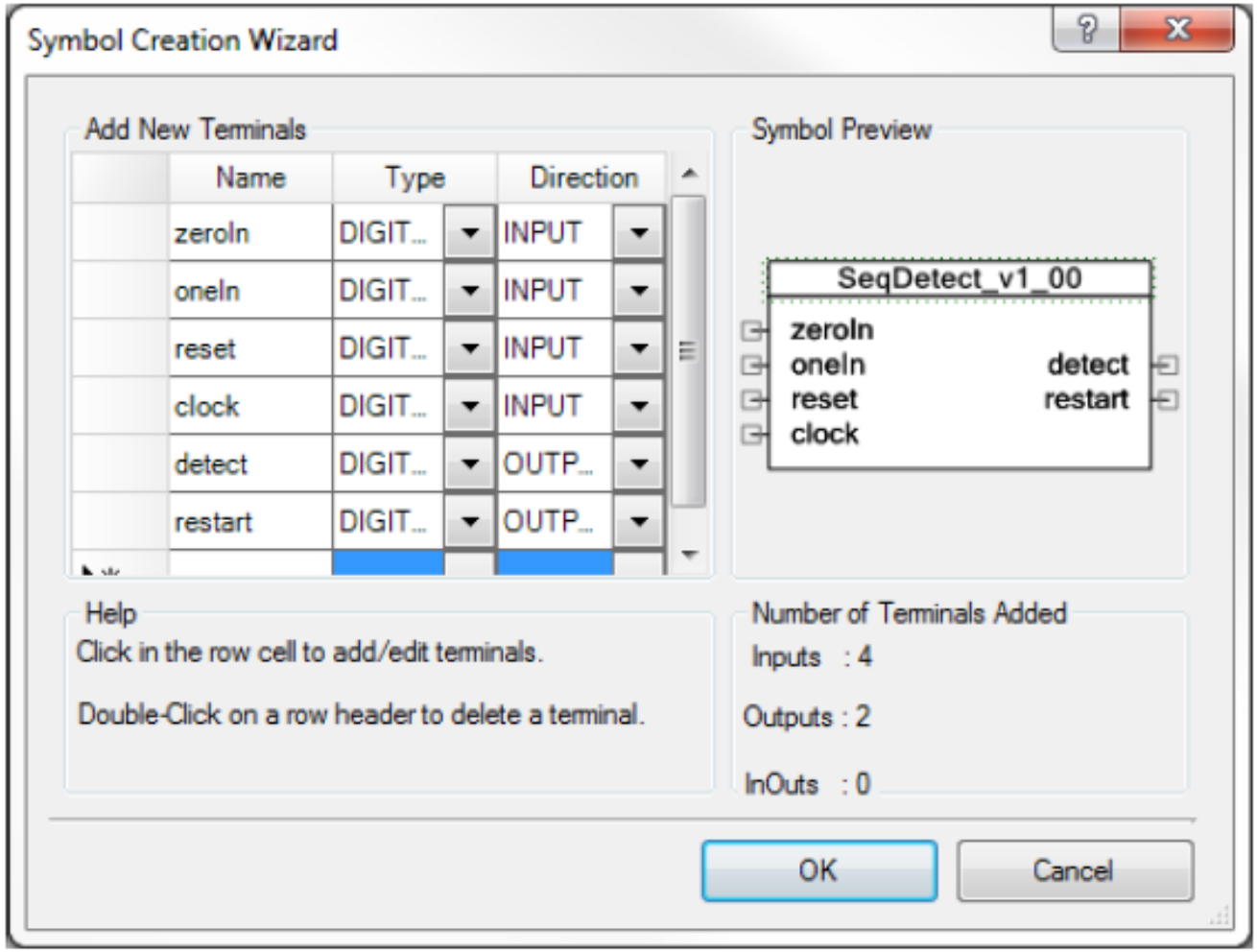
main.c

- A főprogram (és a CPU) gyakorlatilag nem csinál semmit

```
#include <device.h>

int main(){
    /* Place your initialization code here */

    /* CyGlobalIntEnable; */
   for(;;) {
      /* Place your application code here. */
    }
}
```

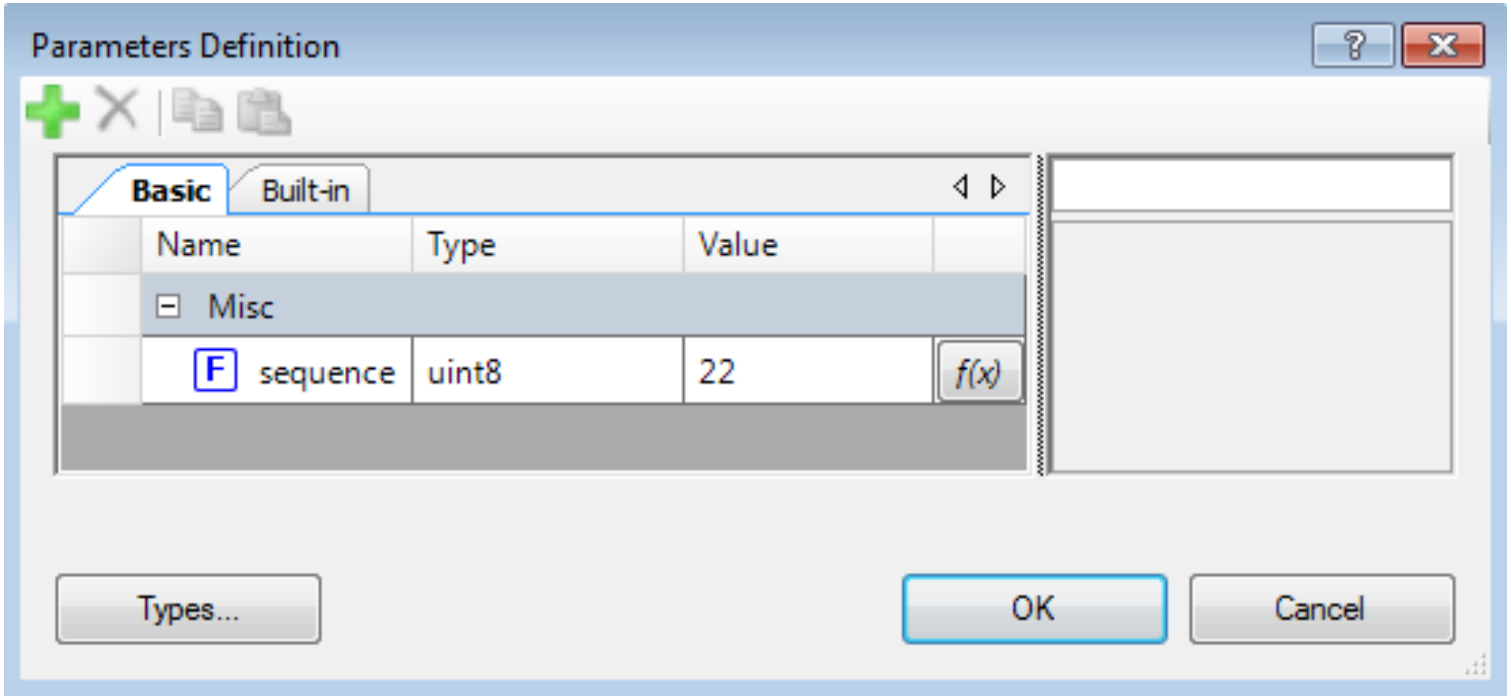
SeqDetect a sorrendfelismerő alkatrész

- Alkatrész neve: ***SeqDetect_v1_00,*** katalógusbeli helye: ***Community/Digital/Logic/Sequence Detector 5-bit***
- Két digitális kimenete és négy digitális bemenete legyen



Alkatrész paraméter hozzáadása

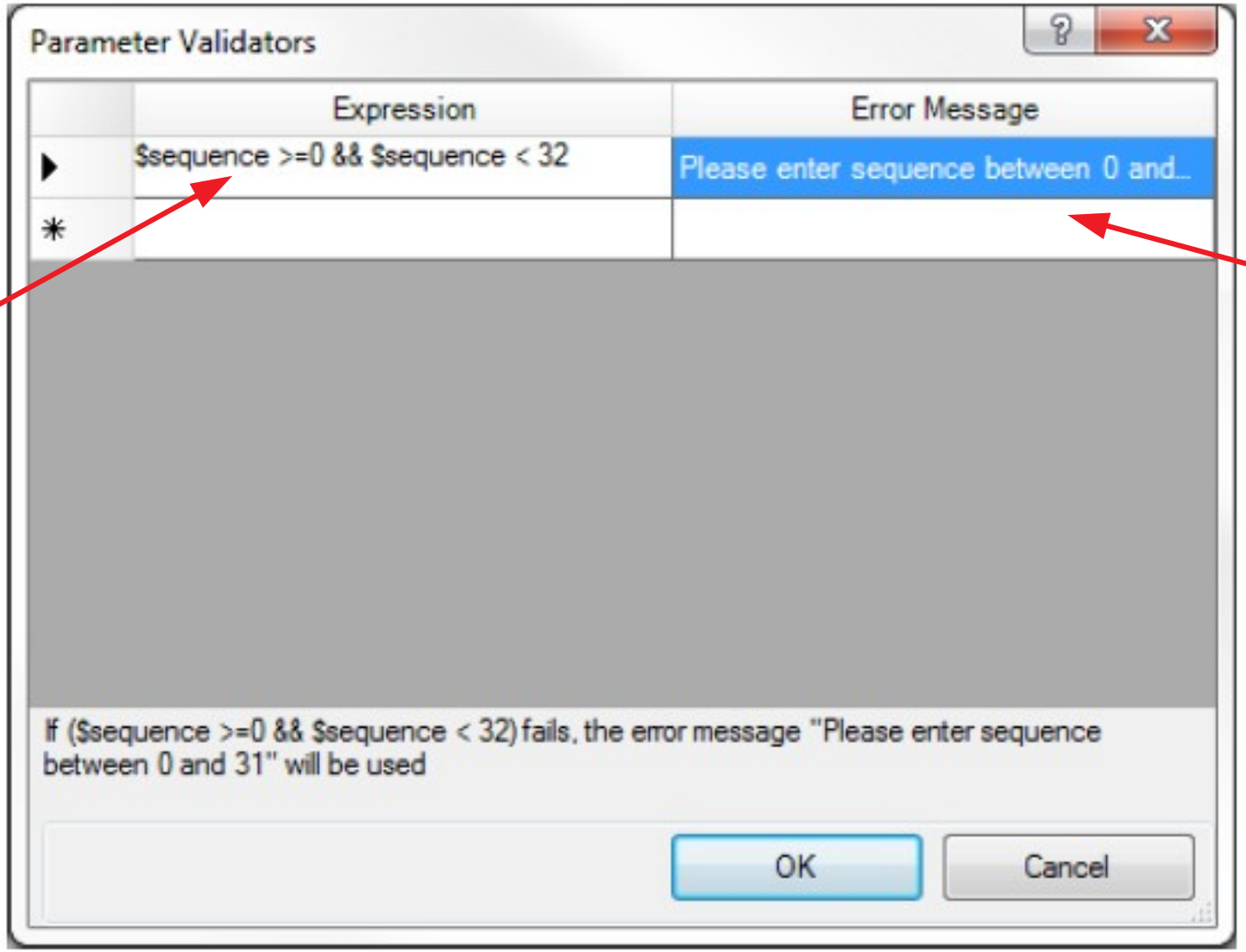
- A paraméter neve legyen **sequence** (azaz sorozat)
- A típusa legyen **uint8**
- Az értéke legyen **22** (binárisan 10 110), konfiguráláskor módosítható



- **Megjegyzés:** a paraméter lehet formális (F) vagy lokális (L), itt hagyjuk meg az alapértelmezett „formális” beállítást!

Paraméter érvényességének ellenőrzése

- Az 5 bites érték 0 és 31 közötti szám lehet

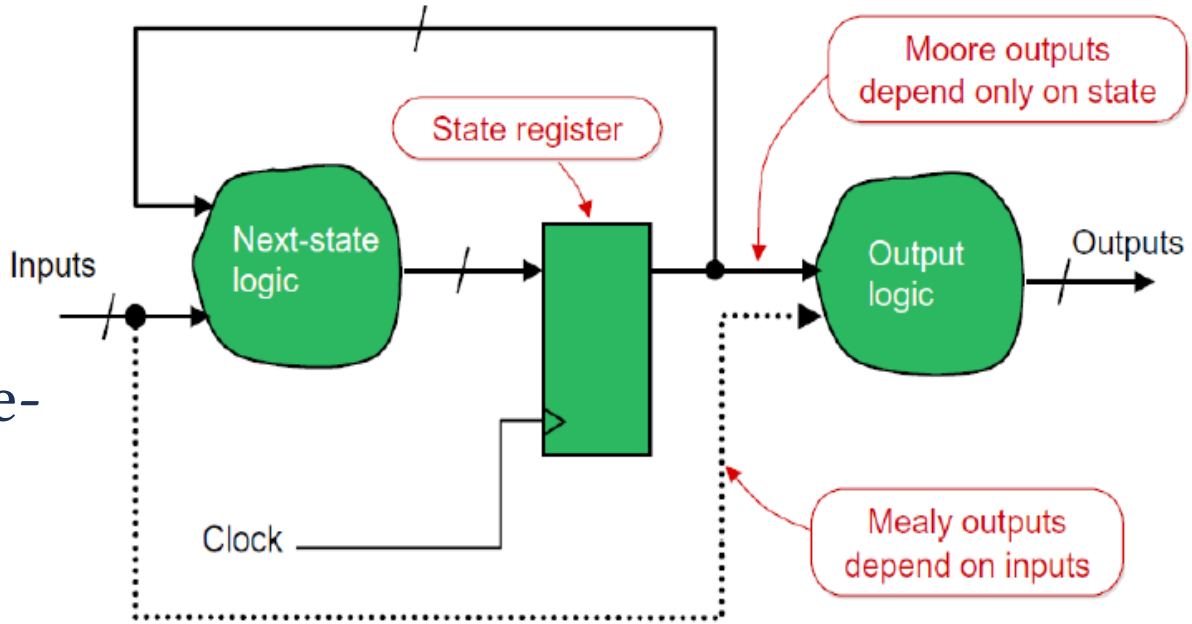


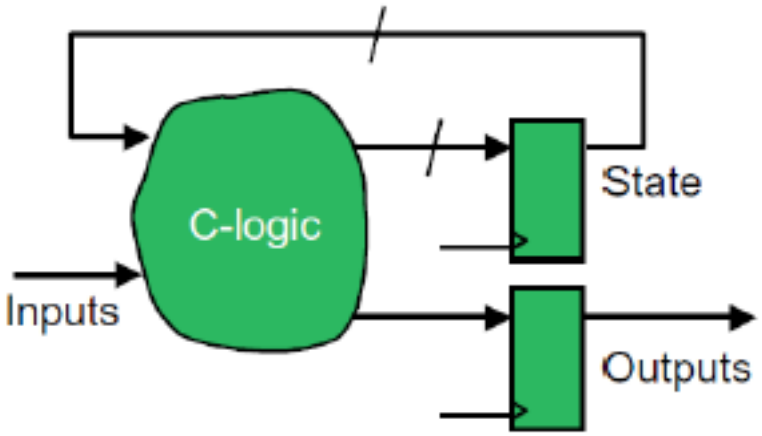
Érvényességi feltétel

Hibaüzenet szövege, amikor a feltétel nem teljesül

Emlékeztető: véges állapotgépek

- A 2017. november 9-i előadásban (cpld05.pdf) foglalkoztunk a véges állapotgépekkel. Az ott szerzett ismeretekkel érthetjük meg a **SeqDetect** sorrendfelismerő alkatrész működését
- A sorrendfelismerő egy Moore típusú állapotgép
- Állapotgépünk kombinációs és szekvenciális áramkörökből áll, ennek megfelelően két ***always*** blokkal írható le





SeqDetect_v1_10.v Verilog kódja

```
`include "cypress.v"

module SeqDetect_v1_10 (
    output reg detect,
    output reg restart,
    input   clock,
    input   oneIn,
    input   reset,
    input   zeroIn
);

parameter sequence = 0;             /* It will be set at configuration */

localparam START   = 3'b000;        /* detect, restart = 0, 1 */
localparam STATE_1 = 3'b001;        /* detect, restart = 0, 0 */
localparam STATE_2 = 3'b011;        /* detect, restart = 0, 0 */
localparam STATE_3 = 3'b010;        /* detect, restart = 0, 0 */
localparam STATE_4 = 3'b110;        /* detect, restart = 0, 0 */
localparam DETECT  = 3'b100;        /* detect, restart = 1, 0 */

/* registered value to hold 3-bit state */
reg [2:0] state_curr, state_next;

/* pattern[4:0] holds the user-supplied sequence value
 * suppose sequence = 22 then pattern[4:0] = 5'b10110
 * Note that pattern[4] is the first-entered user input
 */
wire [4:0] pattern = sequence;
```

A lehetséges állapotok felsorolása

Állapot tárolók

A megadott sorozat tárolója

SeqDetect_v1_10.v

```
/* Sequential block of the state machine - outputs are assigned here */
    always @ (posedge clock)
    begin
         /* reset causes the component to enter the START state */
         if(reset)
         begin
              state_curr <= START;
              detect <= 1'b0;
              restart <= 1'b1;
         end
         else /* reset is not asserted - go through states */
         begin
              state_curr <= state_next;
              if (state_next == DETECT)
              begin
                   detect <= 1'b1;
              end
              else begin
                   detect <= 1'b0;
              end
              if (state_next == START)
              begin
                   restart <= 1'b1;
              end
              else begin
                   restart <= 1'b0;
              end
         end
    end;
```

Újraindítás aktív RESET bemenő jel esetén

Kimenő jelek beállítása

SeqDetect_v1_10.v

```
/* Finite State Machine combinatorial block  */
always @ (oneIn or zeroIn or state_curr or pattern)
    begin
        case(state_curr)
            START:       /* Initial state */
            begin
                /* If either a one or zero has been entered, take action */
                if(oneIn | zeroIn)
                begin
                    /* check whether the first bit entered is correct */
                    if((oneIn & pattern[4]) || (zeroIn & !pattern[4]))
                    begin
                        state_next <= STATE_1;    /* advance to the next state */
                    end
                    else                          /* revert to the initial state */
                    begin
                        state_next <= START;
                    end
                end
                else    /* if neither 1 or 0 have been entered, stay in same state */
                begin
                    state_next <= state_curr;
                end
            end
```

Bemenőjel ellenőrzése

SeqDetect_v1_10.v

```
STATE_1: /* First input is correct */
begin
     if(oneIn | zeroIn) begin
          if((oneIn & pattern[3]) || (zeroIn & !pattern[3]))
          begin
               state_next <= STATE_2;
          end
          else begin
               state_next <= START;
          end
     end
     else begin
          state_next <= state_curr;
     end
end
STATE_2: /* Two inputs are correct */
begin
     if(oneIn | zeroIn) begin
          if((oneIn & pattern[2]) || (zeroIn & !pattern[2]))
          begin
               state_next <= STATE_3;
          end
          else begin
               state_next <= START;
          end
     end
     else begin
          state_next <= state_curr;
     end
end
```

SeqDetect_v1_10.v

```
STATE_3: /* Three inputs are correct */
begin
     if(oneIn | zeroIn) begin
          if((oneIn & pattern[1]) || (zeroIn & !pattern[1]))
          begin
               state_next <= STATE_4;
          end
          else begin
               state_next <= START;
          end
     end
     else begin
          state_next <= state_curr;
     end
end
STATE_4:  /* Four inputs are correct */
begin
     if(oneIn | zeroIn) begin
          if((oneIn & pattern[0]) || (zeroIn & !pattern[0]))
          begin
               state_next <= DETECT;
          end
          else begin
               state_next <= START;
          end
     end
     else begin
          state_next <= state_curr;
     end
end
```

SeqDetect_v1_10.v

```
        DETECT:      /* All five inputs are correct! */
        begin
            /* When in the detect state, an input is given, than START again */
            if((oneIn | zeroIn))
            begin
            /* check whether the bit entered is the correct beginnning */
                if((oneIn & pattern[4]) || (zeroIn & !pattern[4]))
                begin
                    state_next <= STATE_1;
                end
                else     /* revert to the initial state */
                begin
                    state_next <= START;
                end
            end
            else
            begin
                state_next <= state_curr;
            end
        end
     endcase
   end

//`#end` -- edit above this line, do not edit this line
endmodule
```

CY8CKIT-059 fejlesztői kártya

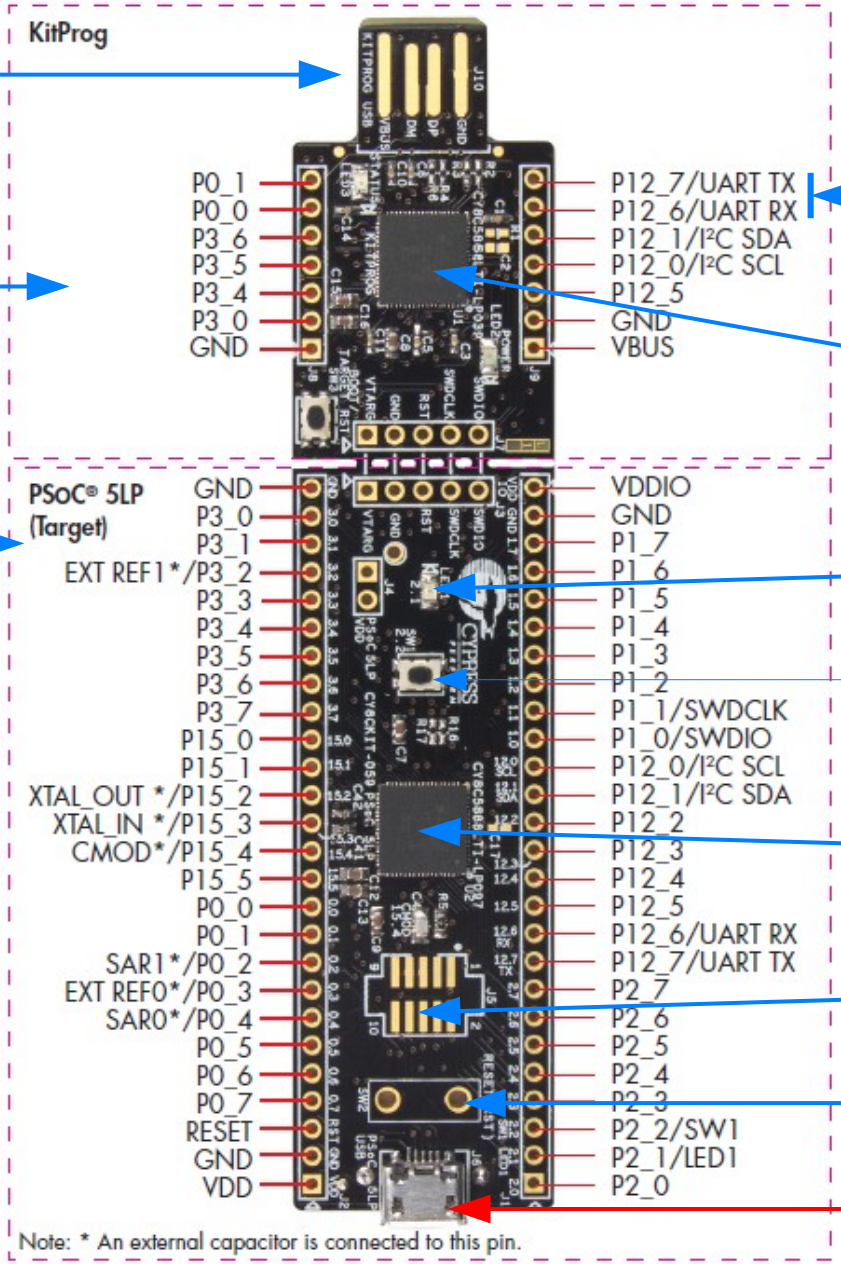
USB csatlakozás a PC-hez

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