BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Digitális technika
VIMIAA01

Fehér Béla
BME MIT

Digitális rendszerek tervezése

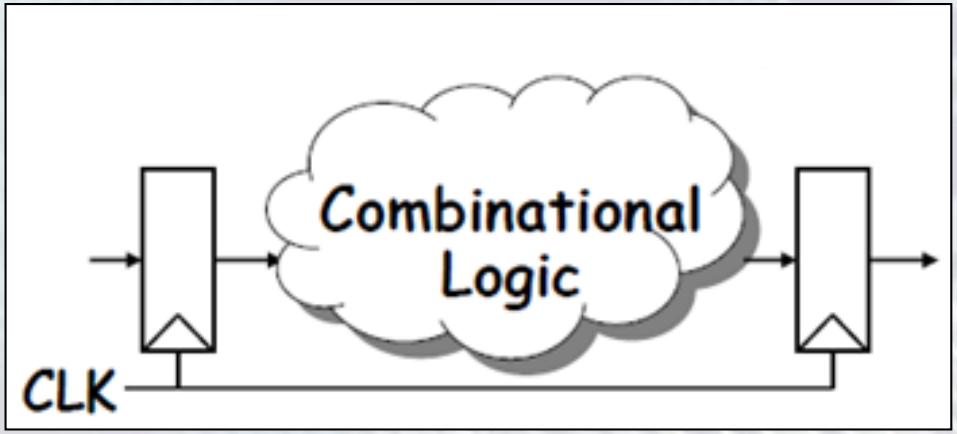
- **Kombinációs logikák**
  - Logikai függvények,
  - Kétszintű realizáció, SOP, POS, minimalizálás
  - Többszintű realizáció, több kimenetű hálózatok
  - Funkcionális elemek, modulszintű építkezés
  - Műveleti idő: maximális jelterjedési idő
- **Szinkron sorrendi hálózatok**
  - Órajelél vezérelt működés, DFF: mintavételezés, tartás
  - Kimeneti érték korábbi állapotok függvénye is
  - Véges állapotú vezérlő, tetszőleges állapotátmenetek
  - Multifunkciós regiszterek, számlálók, memóriák
  - Műveleti sebesség korlátja:

$$\mathbf{T_{clk} \geq t_{cqmax} + t_{pmax} + t_{su}}$$

  - Az órajel ütemez, minden műveletre T_{clk} idő jut

Digitális rendszerek tervezése

- **Összetett digitális rendszerek**
- **Regiszterek + Kombinációs logikák**
- **RTL Regisztertranszfer szintű tervezési modell**



- A változók aktuális értékét az első regiszter mintavételezi és egy teljes órajel periódusnyi ideig stabilan tartja
- A valódi műveletvégzés, adat átalakítás, adattranszformáció a kombinációs logikai hálózatban történik
- A számított/generált eredményt a második regiszter mintavételezi és tárolja a következő számításokhoz

Digitális rendszerek tervezése

- **RTL Regisztertranszfer szintű tervezési modell**
  - Nagymértékben támogatott a Verilog HDL nyelvben
  - Egyértelmű modell a szinkron regiszterek működésének leírására
    - Élvezérelt működés **always @ (posedge clk)**
    - Nemblokkoló értékadás **<=**
  - Gazdag operátorkészlet és adat formátumok a különböző műveletekhez
    - Normál és előjeles (2'C) számábrázolás, tetszőleges adatméret, előjel kiterjesztés, számrendszerek
    - Logikai műveletek bitekre, bitvektorokra
    - Aritmetikai műveletek, léptetések, {} operátor
    - Kombinációs logikák viselkedési leírása

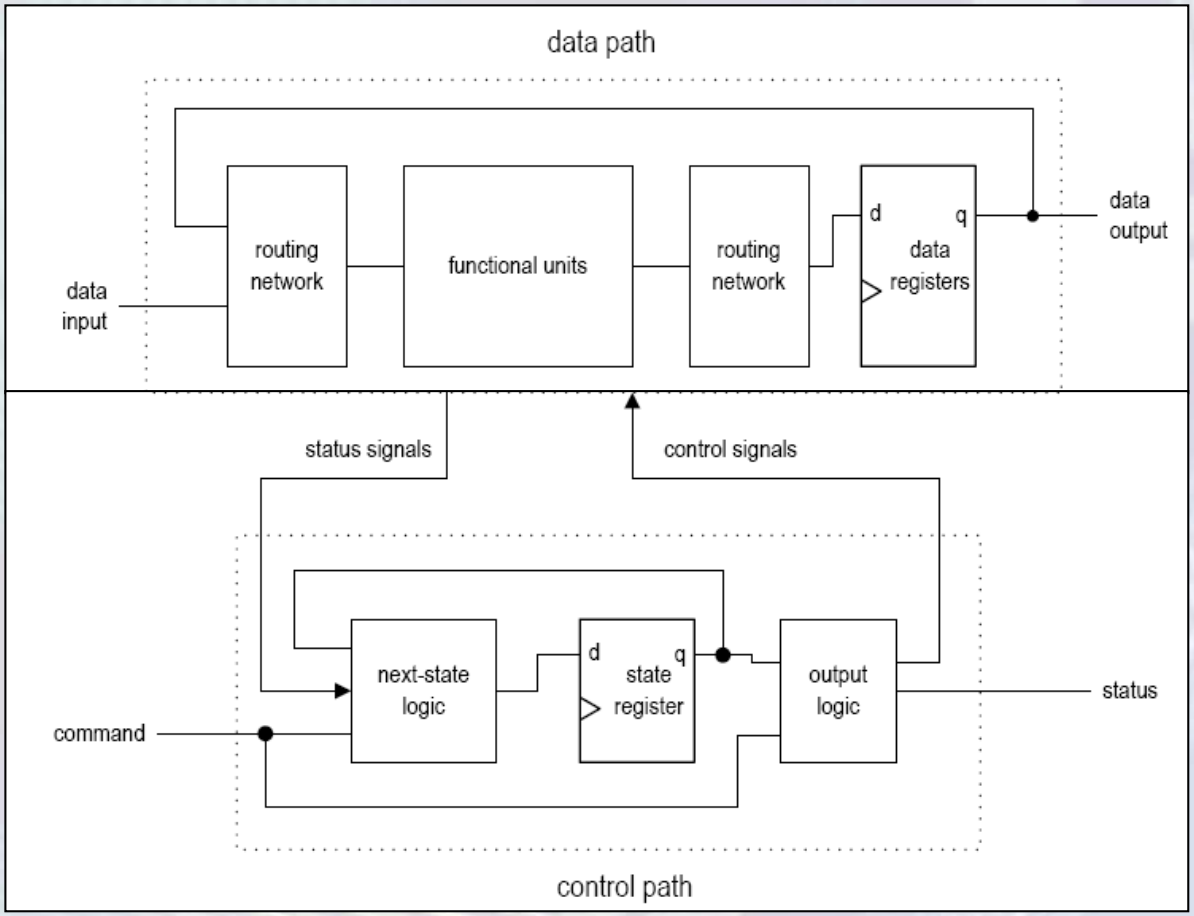
Digitális rendszerek tervezése

- **RTL Regisztertranszfer szintű tervezési modell**
  - Minden eddig tanult ismeretünket felhasználjuk
  - A bonyolult feladatokat magas szintű specifikációval írjuk le, együttesen kezelve a teljes algoritmust
  - Különböző specifikációs módszerek:
    - **HLSM**: Magasszintű állapotvezérlő
    - **ASM**: Algoritmikus állapotvezérlő
  - A tervezésnél az eddigi bitváltozókon felül összetett változókat, bitvektorokat is használunk FSM + adat
  - Az algoritmusok végrehajtását sok esetben szekvenciális módon, lépésenként szervezzük, előírt ütemezés és a működés során generálódó feltételek szerint

Digitális rendszerek tervezése

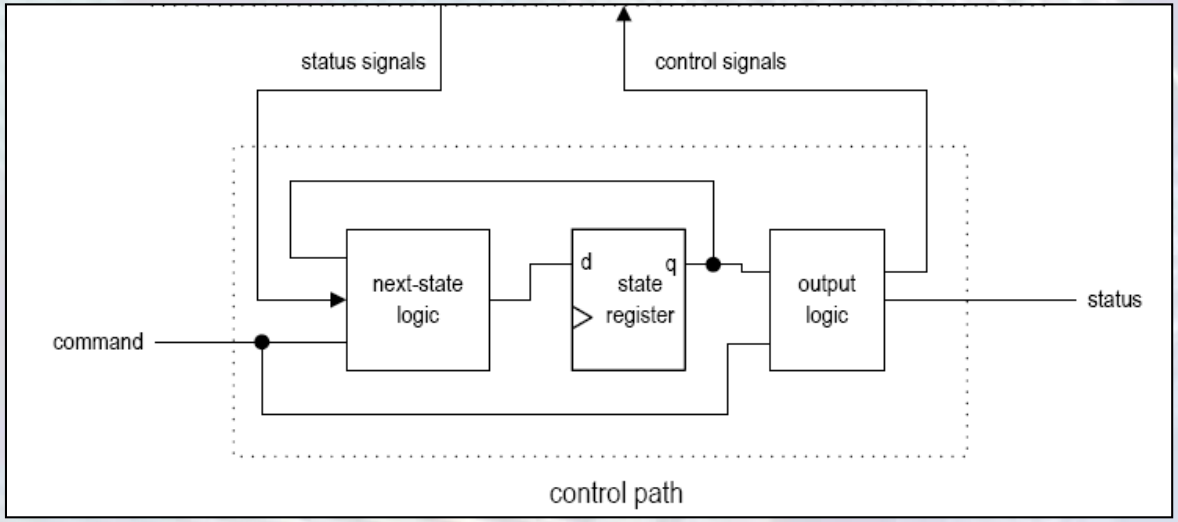
- **RTL Regisztertranszfer szintű tervezési modell**
  - Két fontos alrendszert különböztetünk meg

ADATSTRUKTÚRA + VEZÉRLÉS



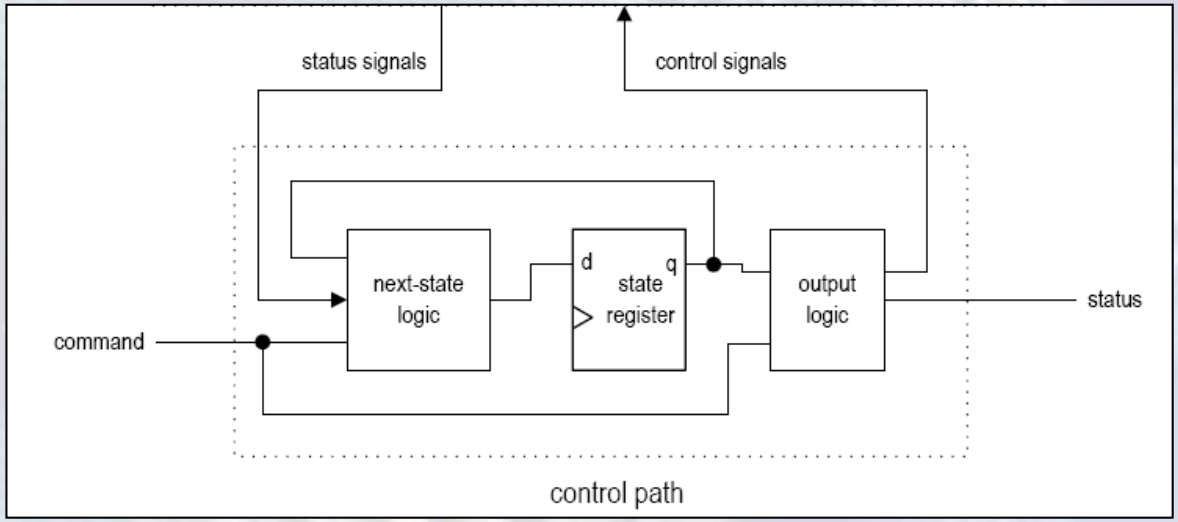
Digitális rendszerek tervezése

- **A vezérlési alrendszer ugyanaz az FSM, amit eddig is terveztünk, de**
  - A bemeneteket megkülönböztetjük, mint
    - **Külső parancsjelek**: START, STOP, egyéb üzemmód,…
    - Belső státuszjelek (az adatstruktúrából): pl. TC=1?, A>B ?
  - A kimeneteket is megkülönböztetjük
    - Belső vezérlőjelek (az adatstruktúrába): INC, LD, ADD, EN
    - **Külső jelzések**: READY, ERROR, VALID…



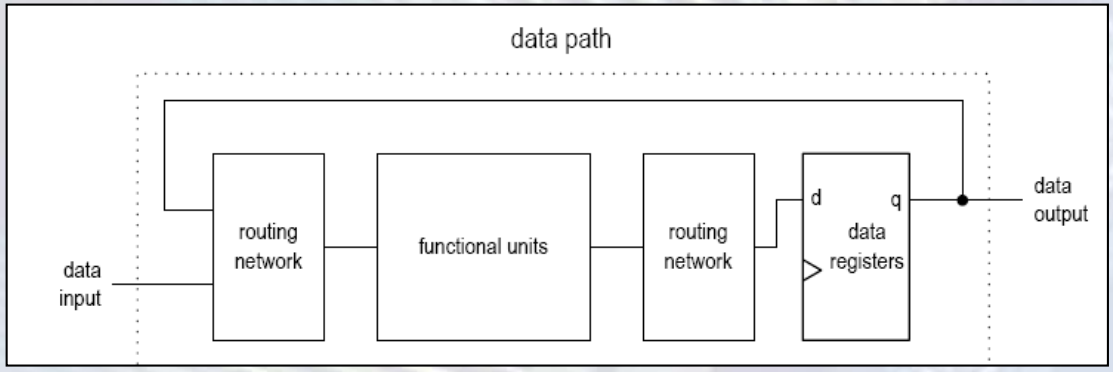
Digitális rendszerek tervezése

- **Az FSM tulajdonságai**
  - Mindig SZINKRON FSM,
  - Globálisan lehet Moore, lehet Mealy, vagy vegyes is
  - Általánosan igaz, hogy a Moore típus választása több állapotot igényel, de „tisztább” tervezés, megvalósítás
  - Az állapotregiszter kódolásának módja a működés szempontjából közömbös (1-az-N-ből, bináris, stb.)



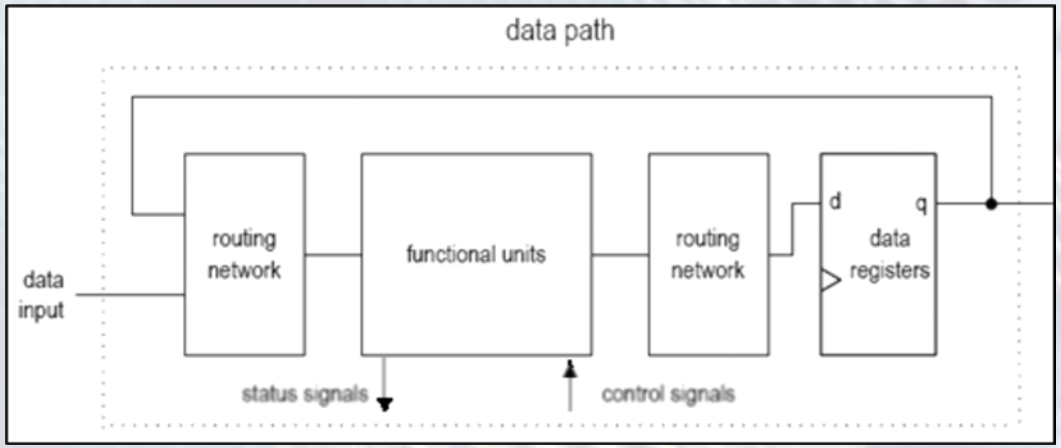
Digitális rendszerek tervezése

- **Az adatstruktúra, azaz az adatfeldolgozó rész**
  - Itt történik a „tényleges” műveletvégzés, tehát a kívánt funkcionalitás végrehajtása
  - Az adatstruktúra elemei:
    - **Adatregiszterek**: az algoritmus változóinak aktuális értékét tárolják, méretük alkalmazásfüggő
    - **Funkcionális egységek**: az ismert/tanult műveletvégzők, mint összeadó, komparátor, shifter, paritásgenerátor, stb.
    - **Adatútválasztók**: Multiplexer hálózatok, feladatuk, ugyanaz, mint a multifunkciós regiszterek esetén



Digitális rendszerek tervezése

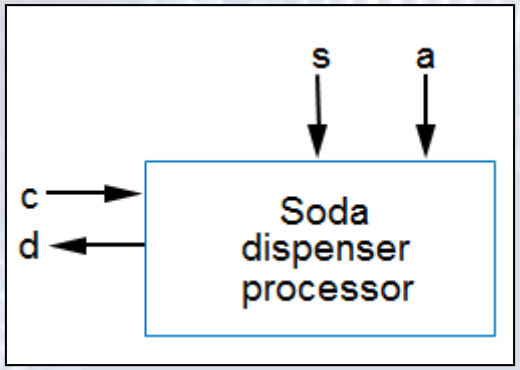
- **Az adatstruktúra, azaz az adatfeldolgozó rész**
  - Az elsődleges adat bemenet és adat kimenet formátuma alkalmazásfüggő (bitvektor/ok)
  - Az adatfeldolgozás lehet többlépcsős, azaz az egyes részfeladatokat végrehajtó funkcionális egységeket regiszterek választják el egymástól



  - Az adatstruktúra státuszjelei a vezérlőbe jutnak, a vezérlőből származó belső vezérlőjelek működtetik az adatstruktúrát

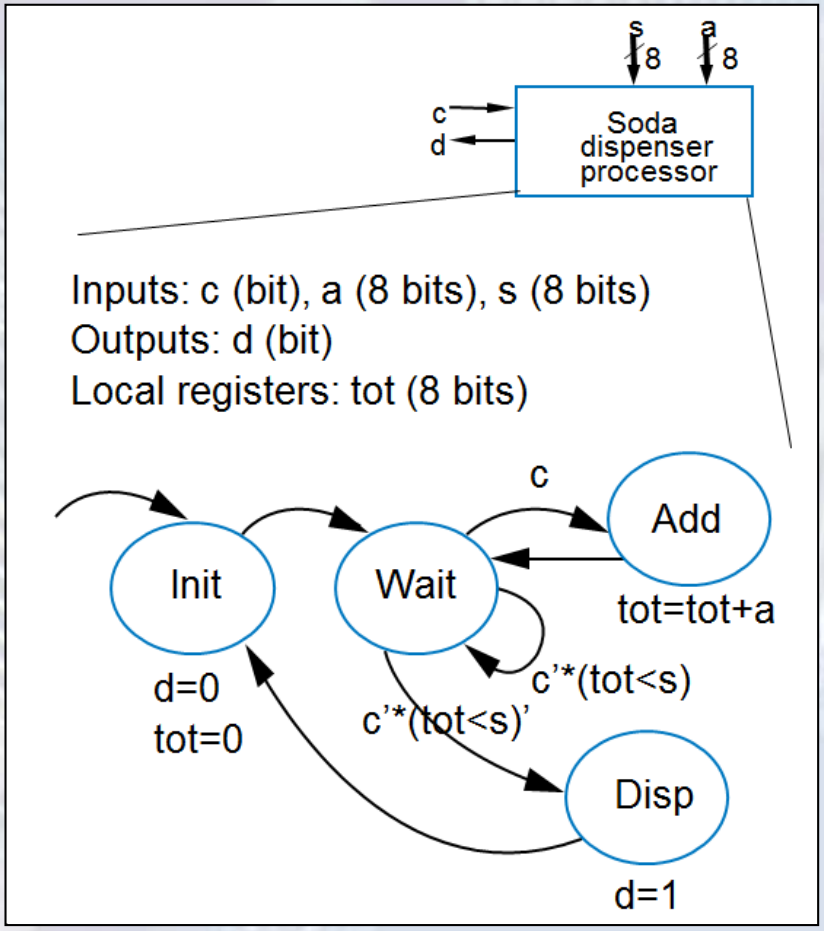
Digitális rendszerek tervezése

- **Specifikációs módszerek**
- **A HLSM, azaz magas szintű állapotgép**
  - Lényegében a megismert állapotdiagram kiegészítése magasszintű műveletvégzési előírásokkal
  - Egy példa: szódavíz automata (lehetne a folyosói kávéautomata is, de ott túl nagy a választék, sok gomb, a bonyolultság eltakarná a lényeget)
  - Interfészek:
    - Bitvektorok: s: a szóda ára, a: a bedobott pénzérmék értéke
    - Bitek: c: pénzérme bevétele, d: pohár szóda kiadása
    - Pl. s = 50, a szóda ára, a fizetés módja 3 érmével, a = 20, 20, 10, ekkor c állapotai 0101010, azaz 3 érme bevétele, ezalatt belül egy „tot” változó a 0-20-40-50 értéket veszi fel, végül d a 010-t adja



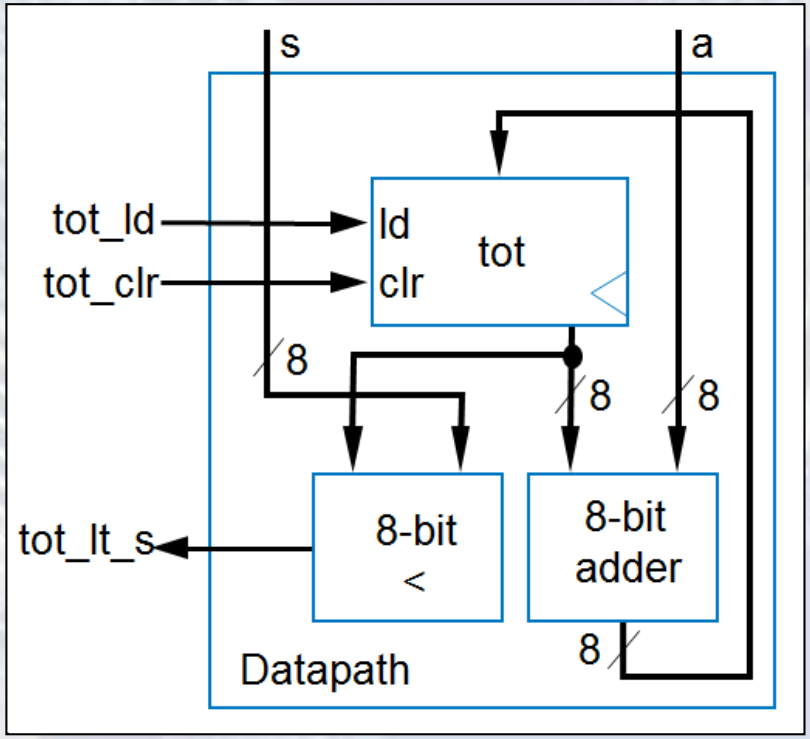
Digitális rendszerek tervezése

- **A HLSM, azaz magas szintű állapotgép**
- **Az állapotdiagramon a szokásos bit jelek mellett megjelennek magasszintű értékadások és műveletek is (ezért hívjuk HLSM-nek)**
- **Ilyenek pl.**
  - 8 bites akkumulátor, tot = tot + a
  - Összetett feltételvizsgálat not (c) and (tot < s)
- **A HLSM alapján az adat-struktúra származtatható**
  - Megjegyzés: A példa nem ad visszajárót, és nem jelzi, ha kifogyott a szóda



Digitális rendszerek tervezése

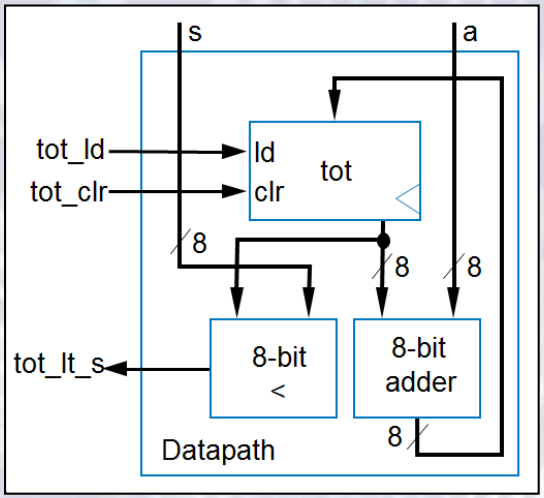
- **Az adatstruktúra felépítése**
  - Szükséges komponensek:
    - 8 bites regiszter a bedobott érmék összegének tárolására
    - 8 bites összeadó az eddigi érmék összegének számolására
    - 8 bites komparátor, ellenőrzi, hogy elegendő-e már?
    - Adatutak kialakítása a rendszer működéséhez
    - Az egyes komponensek vezérlőjeleinek azonosítása (tot_clr, tot_ld)
    - A rendszer státuszjeleinek azonosítása (tot_lt_s)
    - Megtervezhető?
      - Viselkedési leírással egyszerű



Digitális rendszerek tervezése

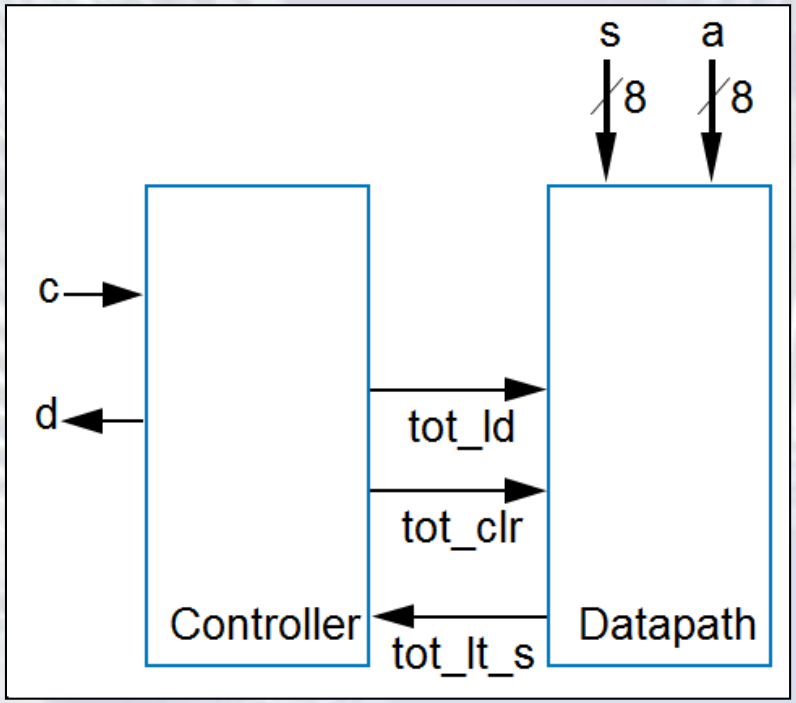
- **Az adatstruktúra Verilog HDL kódja**
  - 4 bites változókkal

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Soda Dispenser példa Frank Vahid Digital Design Handbook alapján
// 4 bites változók az egyszerűbb kijelzéshez a LED-eken
//////////////////////////////////////////////////////////////////////////////////
module soda_datapath(
    input clk,
    input rst,
    input [3:0] soda_price,
    input [3:0] add_value,
    input tot_clr,
    input tot_ld,
    output tot_lt_s,
    output reg [4:0] sum      // Csak a folymatos kijelezhetőség miatt
    );

wire [4:0] add_out;

// Az összeadó kimenet külön megjelenítése
assign add_out = sum + add_value;

// A bedobott érmék összegzése
always @ (posedge clk)
   if (rst | tot_clr) sum <= 0;        // Indulánál vagy új ciklusnál
   else
      if (tot_ld) sum <= add_out;      // Az összeg frissítése az új értékkel

// A jelenlegi állapot kijelzése
assign tot_lt_s = (sum < {1'b0, soda_price}) ? 1'b1 : 1'b0;

endmodule
```



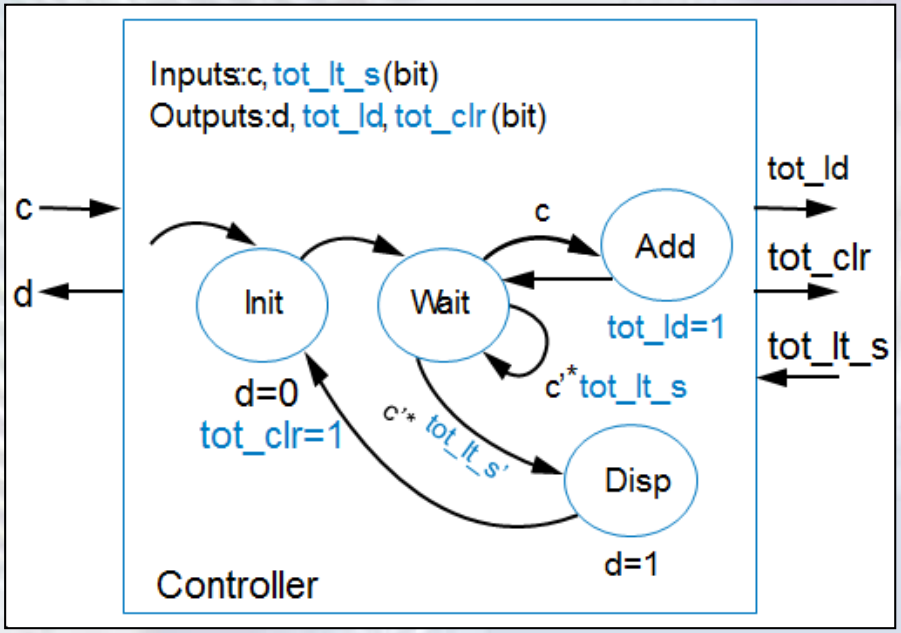
Digitális rendszerek tervezése

- **Az adatstruktúra kiegészítése a vezérlővel**
  - **Külső parancs jel**: c érme bedobás érzékelőjele
  - **Külső vezérlő jel**: d pohár szóda kiadása
  - **Belső vezérlőjelek**: tot_clr, tot_ld, összeg regiszter vezérlése, törlés, töltés
  - **Belső státuszjel**: tot_lt_s, az eddig bedobott összeg kevesebb, mint a szóda ára
  - Itt a vezérlő már egyszerű FSM, bitszintű jelekkel



Digitális rendszerek tervezése

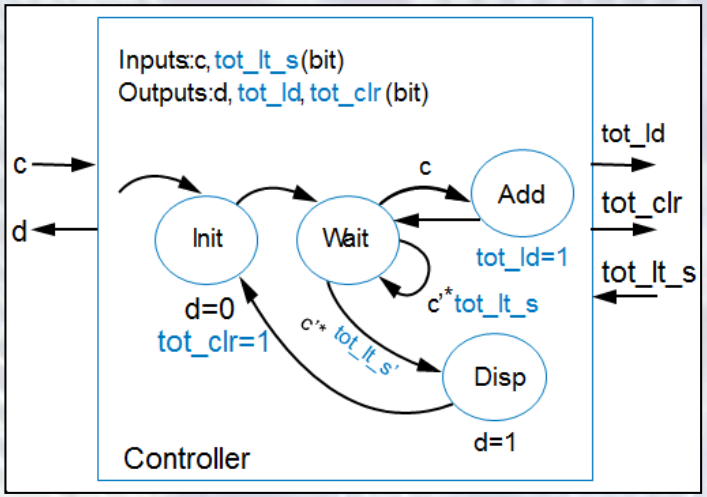
- **A vezérlő egység**
  - Az állapotdiagram ugyanaz, 4 állapot, köztük az eddigi állapotátmenetekkel, ugyanaz a topológia
  - Minden bemeneti és kimeneti jel egy bites
  - A magas szintű műveleteket a hozzájuk tartozó vezérlőjel ill. státuszjel reprezentálja
  - A működés órajel vezérelt
  - Meg tudjuk tervezni?
    - 2 bites állapot regiszter
    - Állapot átmeneti logika specifikálása viselkedési leírással
    - Kimeneti jelek állapothoz rendelve → Moore modell



Digitális rendszerek tervezése

- **A vezérlő egység**

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Soda Dispenser példa Frank Vahid Digital Design Handbook alapján
// A vezérlő egység Verilog kódja
//////////////////////////////////////////////////////////////////////////////////
module soda_controller(
    input clk,
    input rst,
    input coin,
    output dispense,
    output tot_clr,
    output tot_ld,
    input tot_lt_s,
    output reg [1:0] state
    );

parameter INIT = 2'b00;                 // Szimbolikus állapotkódok
parameter WAIT = 2'b01;
parameter ADD  = 2'b10;
parameter DISP = 2'b11;


always @ (posedge clk)
   if (rst)  state <= INIT;              // Indulás INIT állapotban
   else
      case (state)
      INIT:    state <= WAIT;            // Utána azonnal WAIT következik

      WAIT:    if (coin) state <= ADD;   // Ha új érme jött, összegzés ADD-ban
               else if (~tot_lt_s)
                        state <= DISP; // Ha már elég az érme, szóda kiadása
      ADD:     state <= WAIT;            // Összegzés és visszatérés WAIT-be
      DISP:    state <= INIT;            // Szóda kiadás és új működési ciklus
      default: state <= INIT;            // Bármi nem tervezett, vissza INIT-be
      endcase


// Vezérlőjelek generálása állapotok alapján, Moore modell
assign tot_clr = (state == INIT);
assign tot_ld  = (state == ADD);
assign dispense = (state == DISP);

endmodule
```



Digitális rendszerek tervezése

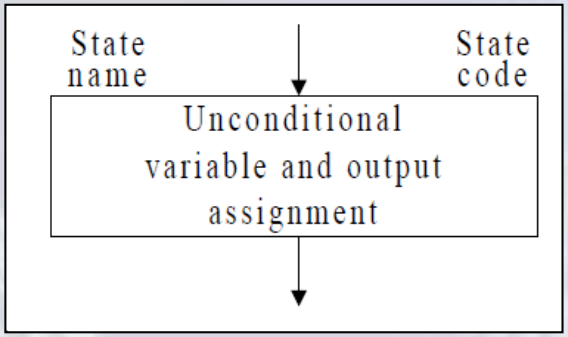
- **A HLSM jó módszer a specifikálásra**
- **Lényegében ugyanaz az eszközkészlet, amit eddig használtunk, megismertünk**
- **A kezdeti specifikációt egy állapotdiagramban fogalmazzuk meg, kis kiegészítéssel**
- **Ezek a kiegészítések az eddigiekhez képest:**
  - Több bites bemeneti és kimeneti jelek
  - Belső tárolók a változók számára
  - Aritmetikai műveletek, a logikai műveletek mellett
- **Természetesen a HLSM specifikálása, felépítése gyakorlatot és tapasztalatot igényel**
  - De nem igényli a működés elemi lépésekre bontását, meghagyja a hardver párhuzamos működésének lényegét

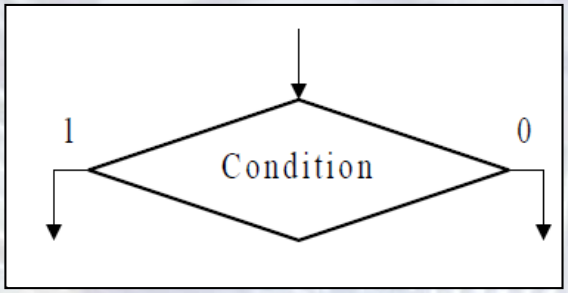
Digitális rendszerek tervezése

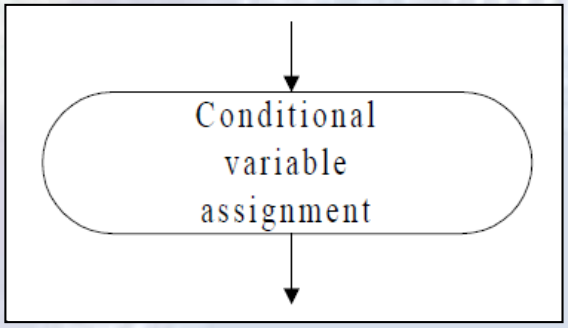
- **Az ASM, azaz algoritmikus állapotvezérlő**
  - Az ASM egy folyamatábra jellegű specifikáció
  - A működés itt is órajel vezérelt (digitális HW), minden állapotváltozás egy-egy órajelet igényel
  - Az ASM erősen strukturált jellegű, nem annyira szabad szerkezetű, mint egy HLSM
  - Az állapotátmenetek nem tetszőlegesek (mint a HLSM-nél), hanem egy (esetleg kettő, de nem tetszőleges számú) következő állapotra korlátozottak, hasonlóan a számítógépes algoritmusok stílusához
  - Ebből következően nem annyira hatékony HW tervezéshez, mint a HLSM, de segít bevezetni a későbbiekben a programvezérelt processzor működését

Digitális rendszerek tervezése

- **Az ASM, azaz algoritmikus állapotvezérlő**
  - Az ASM leírás elemei:
  - **Állapot doboz:** Azonosítja az állapotot, a kódját és végrehajtja az állapotváltozók és a Moore típusú kimenetek megújítását
  - **Feltétel doboz:** Az adatstruktúra vagy a következő állapot logika működését vezérli
  - **Kimeneti doboz:** Az esetleges feltételes értékadások művelete, jellemzően Mealy típusú kimenetekhez

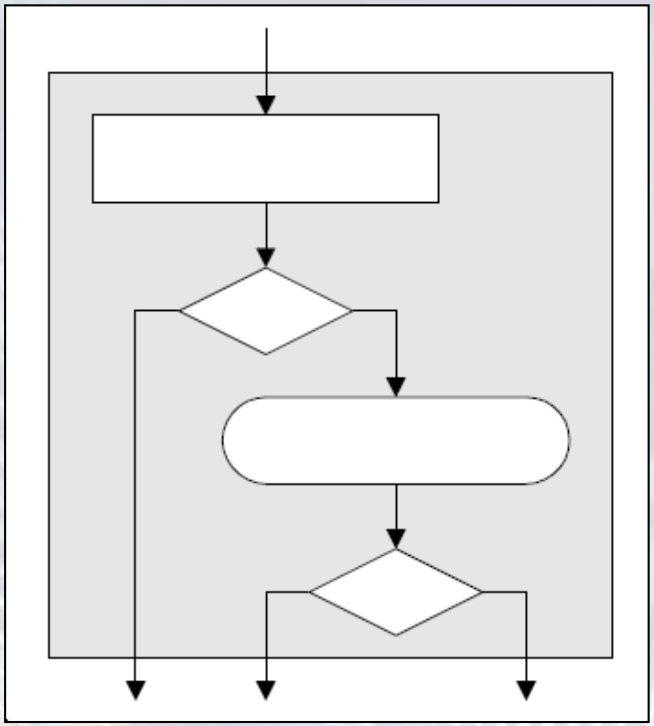






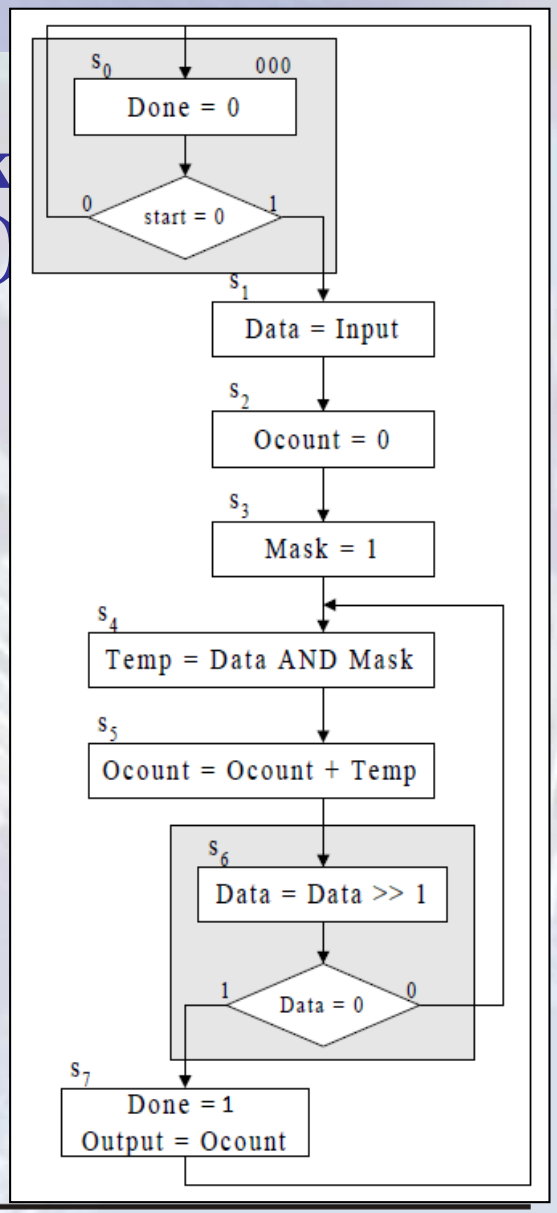
Digitális rendszerek tervezése

- **Az ASM, azaz algoritmikus állapotvezérlő**
- **Az ASM blokk szerkezete:**
  - Minden eddigi „FSM” állapot egy ASM blokk lesz (egyértelmű megfeleltetés)
  - Az ASM blokkban mindig van állapotdoboz (következik a fenti megjegyzésből)
  - Az ASM blokk tartalmazhat egy vagy több feltételvizsgálatot és feltételes értékadást
  - A tipikus használati mód nagyon egyszerű ASM blokkokkal dolgozik.



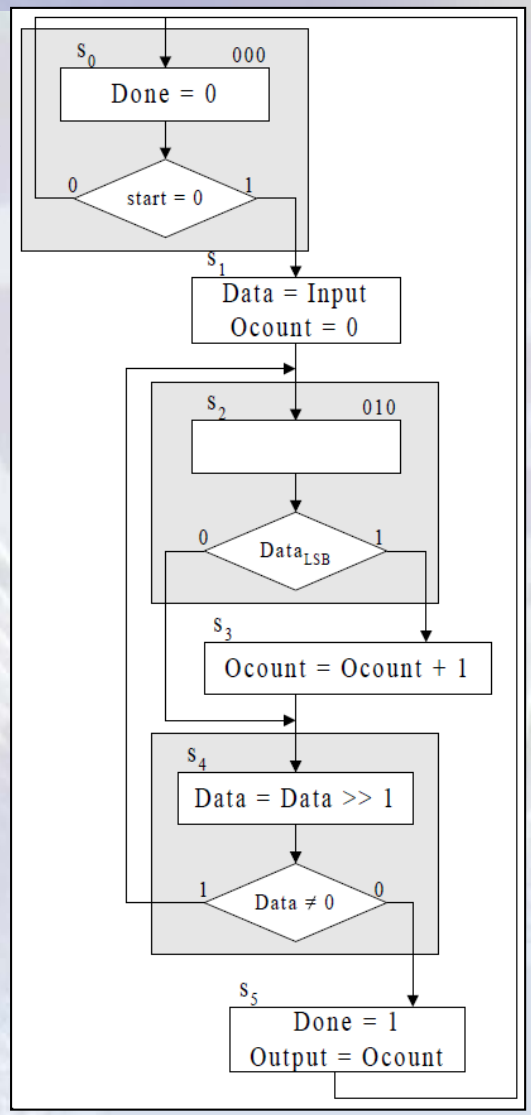
Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- **Példa: Egy n bites adat aktív bitjeinek megszámolása (Ones_count/Popcount)**
  - 8 állapot, 8 ASM blokk
  - S0: Várakozás START jelre
  - S1,S2,S3: Változók inicializálása
  - S4, S5, S6: Számolás ciklusban, amíg még van aktív bit („1”-es)
  - S7: érték kijelzés, kész jelzés, új ciklusra visszatérés
- **Láthatóan egyszerű ASM blokkok**
  - Nagyon elemi feladatlebontás, mint a mikroprocesszorok gépi utasításai



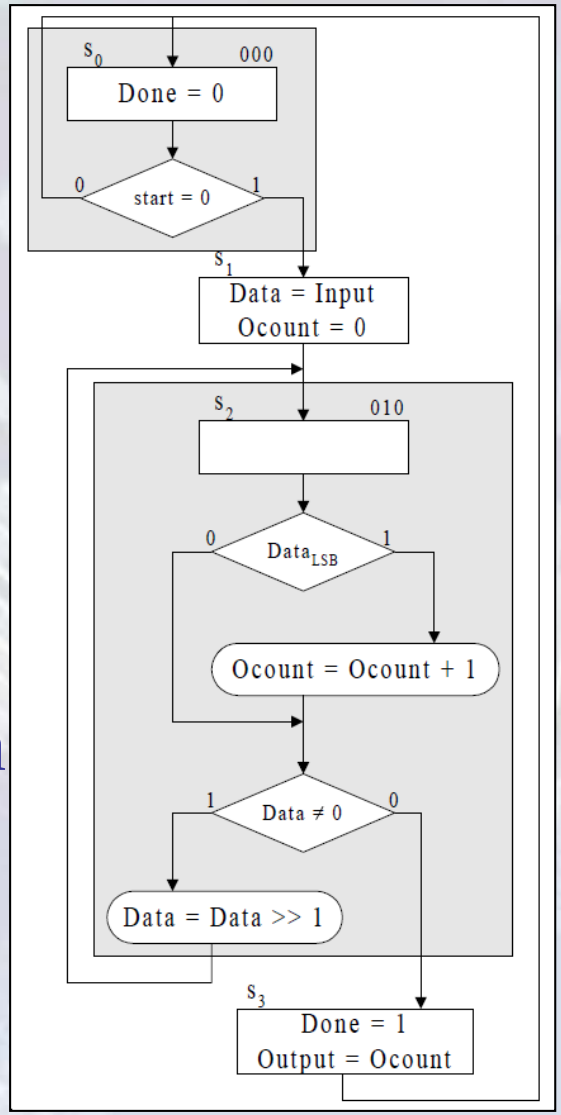
Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- **Ez így inkább SW jellegű, lépésenkénti végrehajtás, egy időben egy művelet**
- **A HW párhuzamos, időben konkurens viselkedését „visszacsempészhetjük”**
  - S0 ugyanaz, vár START-ra
  - S1 minden változót inicializál
  - S2 csak vizsgál, kell-e művelet
  - S3 inkrementál, ha igen
  - S4 lépteti az adatot és tesztel: ha nem kész, vissza S2-be (adatfüggő ciklus)
  - S5: érték kijelzés, kész jelzés, új ciklusra visszatérés



Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- **Az állapotok száma csökkenthető, Mealy típusú ASM blokk(okk)al**
- **Ez jelentős módosítás, komplex ASM blokk kell hozzá**
  - S0 ugyanaz, vár START-ra
  - S1 minden változót inicializál
  - S2 az adatvizsgálat alapján feltételesen inkrementál és ha az adat nem nulla, akkor annak léptetése után ismétli önmagát ciklusban, amíg még van „1” értékű (számolandó) bit
  - S3: érték kijelzés, kész jelzés, új ciklusra visszatérés



Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- **Az ASM specifikáció is kellően rugalmas, jelentős tervezői szabadságot ad, de inkább a SW jellegű, sorrendi elemi utasításokon alapuló végrehajtás megtervezését támogatja**
- **Az elemi ASM blokkokkal építkezve könnyű az áttérés a gépközeli SW algoritmusok tervezésének tanulására**
- **Az ASM modellben a jellemző állapotátmenetek:**
  - CONT: folytatás
  - JUMP: ugrás tetszőleges állapotra (saját magára is)
  - CJMP: feltételes ugrás tetszőleges állapotra
    - Feltétel nem teljesülése esetén CONT

Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- **A vezérlő ebben az esetben lehet egy egyszerű számlálón alapuló vezérlőegység (nem szükséges a sokkal általánosabb FSM felépítés)**
- **Az állapotátmenetek vezérlése:**
  - CONT: state <= state + 1;
  - JUMP : state <= new_state;
  - CJMP : if (COND) then state <= new _state
    else state <= state +1
- **Ezt a feladatot egy törölhető, tölthető bináris felfelé számláló tudja biztosítani → Nevezhetjük állapotszámlálónak vagy akár utasításszámlálónak is**

Digitális rendszerek tervezése

- **Összefoglalva**
- **A bonyolult digitális rendszereket adatstruktúra és vezérlő szemlélettel tudjuk hatékonyan tervezni**
- **Két hatékony specifikációs módszer:**
  - HLSM: Megőrzi a HW teljes párhuzamosságát
  - ASM: Inkább sorrendi jellegű, elemi műveletsorozat
- **Mindkét esetben több bites változókra építve funkcionális adatstruktúrát hozunk létre + vezérlés**
- **A vezérlő lehet Moore, vagy Mealy típusú, vagy akár kevert módon működő**
  - Az ASM specifikációnál a Mealy típusú ASM blokk bonyolult feladatokra képes, de nem könnyű kialakítani, tervezése rutint igényel

Digitális rendszerek tervezése

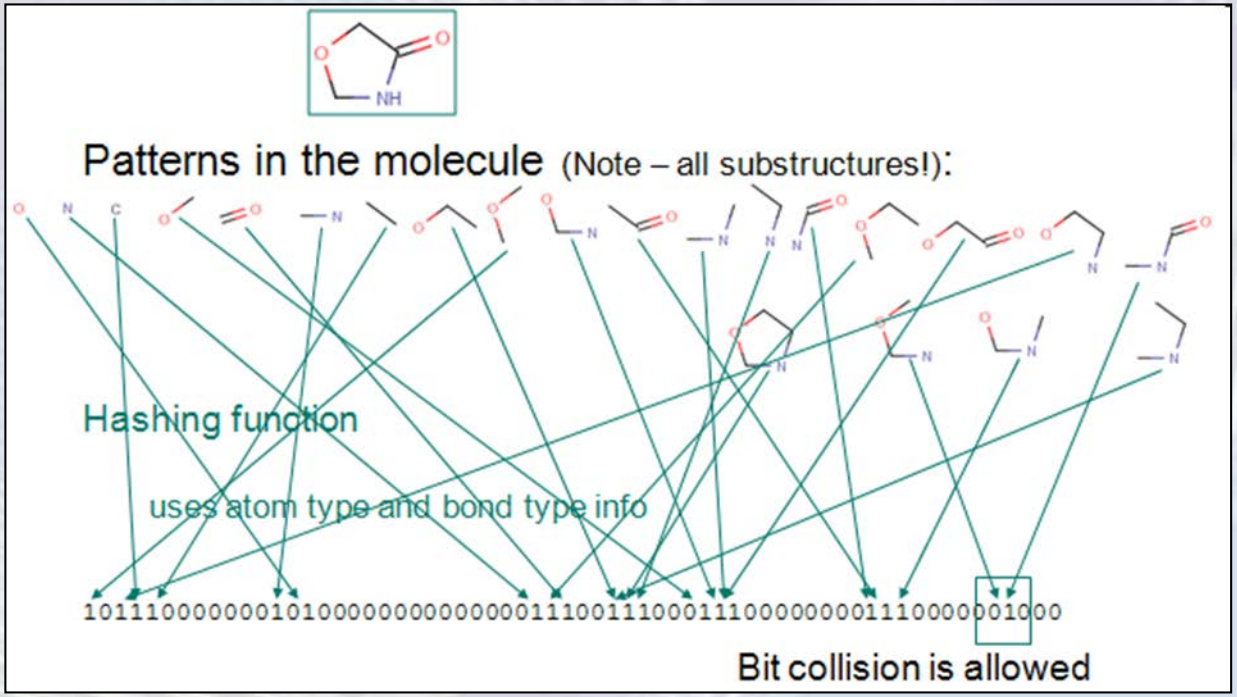
- **Megjegyzés:**
- **Egyszerűbb esetekben nem különböztetjük meg explicit módon az adatstruktúrát és vezérlőt**
- **Az adatstruktúra viselkedési leírásában, a beépített feltételeket, mint vezérlési feltételeket kiértékelve tudjuk a kívánt működési állapotokat elérni**
- **Ez egy intuitív tervezési módszer. Sokszor nagyon ésszerű, gazdaságos, „magától értetődően jó” megoldásokat eredményez, de odafigyelést igényel.**
- **Gyakran nehéz az egyes speciális feltételek korrekt kezelése, és így hajlamosak leszünk kisebb engedményekre, „kompromisszumokra” (pl. INIT állapotban kiadott READY jelzés)**

Digitális rendszerek tervezése

- **Tervezési példák:**
- **Az Ones_count/Population_count feladat**
  - Szép feladat, egyszerű, de sokfajta megoldási lehetőség
  - A kérdés, hogy mire optimalizálunk.
    - Végrehajtási időre, erőforrásra, egyszerű struktúrára?
    - Mekkora az adatméret, amire az aktív „1” bitek számát számoljuk?
    - Van-e idő végrehajtani n bit ciklikus léptetését, számolgatását?
    - Szoftveres (32 bit / 64 bit CPU) vagy hardver alapú megoldást keresünk?

Digitális rendszerek tervezése

- **Egy alkalmazási példa:**
- **Molekula ujjlenyomat alapján kémiai hasonlósági keresés (gyógyszerkutatási alkalmazás)**
- **Nagyméretű 1024 - 2048 bit, az egyes tulajdonságok megléte egy-egy bittel jelezve**
- **Hasonlóság: Bitenkénti AND és utána „1” számolgatás**
- **10^5 molekula esetén jelentős idő**



Digitális rendszerek tervezése

- **Population count közvetlen logikával:**
- **Ez egy egyszerű leképezés INPUT → OUTPUT**
- **Kombinációs logika (Tavalyi digit gyakorlat 6/5 fel.)**
- **Egyszerű összeadás műveletek, 1, 2, 3 …$\log_2 n$ méretű adatokkal, bináris fa struktúrában**
- **Nincs vezérlő, nincs állapotgép, a funkció be van építve a topológiába**
- **Esetleg a bemeneten, kimeneten adattároló regiszter használható**

```
//*********************************************************************************
//* Population count - Aktív bitek megszámolása egy adatvektorban                 *
//*********************************************************************************
module pop_cnt(
                input  wire [7:0] data,          //Bemeneti adatvektor
                output wire [3:0] population     //Kimeneti eredmény
              );

//*********************************************************************************
//* A 8 bites bináris számban található 1 biteket hét összeadó segítségével     *
//* számolhatjuk meg: össze kell adnunk a szám összes bitjét. Az összeadókat    *
//* célszerű fa formában elrendezni, az egyes szinteken növekvő bitszámú        *
//* összeadókat felhasználva az átvitel miatt.                                  *
//*                                                                             *
//*    |bit7|bit6|bit5|bit4|bit3|bit2|bit1|bit0|                                 *
//*       \   /     \   /     \   /     \   /                                    *
//*         +         +         +         +        1 bites összeadók            *
//*         |         |         |         |                                     *
//*         \--- + ---/         \--- + ---/        2 bites összeadók            *
//*              |                   |                                          *
//*              \-------- + --------/             3 bites összeadó             *
//*                        |                                                    *
//*                    num_of_1s[3:0]                                           *
//* Megjegyzés: Érdemes elgondolkodni, hogy milyen más megoldások létezhetnek,  *
//* amik kevesebb aritmetikai áramkörrel oldják meg a feladatot?                *
//*********************************************************************************
assign population = ((data[7] + data[6]) + (data[5] + data[4])) +
                    ((data[3] + data[2]) + (data[1] + data[0]));

endmodule
```

Digitális rendszerek tervezése

- **Population count ASM alapon,**
- **Ez egy egyszerű leképezés INPUT → OUTPUT**
- **Kombinációs logika (Tavalyi digit gyakorlat 6/5 fel.)**
- **Egyszerű összeadás műveletek, 1, 2, 3 …$\log_2 n$ méretű adatokkal, bináris fa struktúrában**
- **Nincs vezérlő, nincs állapotgép**
- **Esetleg bemeneten, kimeneten adattároló regsizter használható**

```
//*****************************************************************************
//* Population count - Aktív bitek megszámolása egy adatvektorban             *
//*****************************************************************************
module pop_cnt(
                input  wire [7:0] data,          //Bemeneti adatvektor
                output wire [3:0] population     //Kimeneti eredmény
              );

//*****************************************************************************
//* A 8 bites bináris számban található 1 biteket hét összeadó segítségével   *
//* számolhatjuk meg: össze kell adnunk a szám összes bitjét. Az összeadókat  *
//* célszerű fa formában elrendezni, az egyes szinteken növekvő bitszámú      *
//* összeadókat felhasználva az átvitel miatt.                                *
//*                                                                           *
//*    |bit7|bit6|bit5|bit4|bit3|bit2|bit1|bit0|                              *
//*       \   /     \   /     \   /     \   /                                 *
//*         +         +         +         +        1 bites összeadók          *
//*         |         |         |         |                                   *
//*         \--- + ---/         \--- + ---/        2 bites összeadók          *
//*              |                   |                                        *
//*              \-------- + --------/             3 bites összeadó           *
//*                        |                                                  *
//*                    num_of_1s[3:0]                                         *
//* Megjegyzés: Érdemes elgondolkodni, hogy milyen más megoldások létezhetnek, *
//* amik kevesebb aritmetikai áramkörrel oldják meg a feladatot?              *
//*****************************************************************************
assign population = ((data[7] + data[6]) + (data[5] + data[4])) +
                    ((data[3] + data[2]) + (data[1] + data[0]));

endmodule
```

Digitális technika
8. EA vége