Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Digitális technika
VIMIAA02
7. hét

Fehér Béla, Benesóczky Zoltán, Raikovich Tamás
BME MIT

Digitális rendszerek tervezése

- **Komplex digitális rendszerekben** a megismert kombinációs és sorrendi funkcionális elemeket és az FSM-et az alábbi feladatokra használjuk:
- **Kombinációs funkcionális elemek:**
  - Matematikai ***művelet végzés:*****Összeadó, kivonó, inkrementer, dekrementer**
  - Adatok közötti ***relációk jelzése*** (<, =, >): teljes funkciós **komparátor** (magnitudó komparátor)
  - ***Adat út választás:*** **multiplexer**
- **Szinkron sorrendi funkcionális elemek**:
  - ***Adat tárolás:*** **tölthető regiszter, multifunkciós regiszter**
  - ***Számolás, időzítés***: **számláló**
  - ***Párhuzamos-soros, soros-párhuzamos átalakítás, shiftelés***: **shiftgeriszter**
- Az ***állapotgépet (FSM)***: **Vezérlési feladatokra:** funkcionális elemekből megépített rendszer működtetésére.
  (Pl. Regiszterbe írás engedélyezése, számláló léptetése, multiplexer kiválasztó jelének beállítása stb.) Ehhez a feladathoz az FSM feltétel bemenetként felhasználhatja a funkcionális elemek és egyéb logikák kimenő jeleit.
  Pl. komparátor kimenete, számláló tc jele, összeadó co jele stb.

Digitális rendszerek tervezése

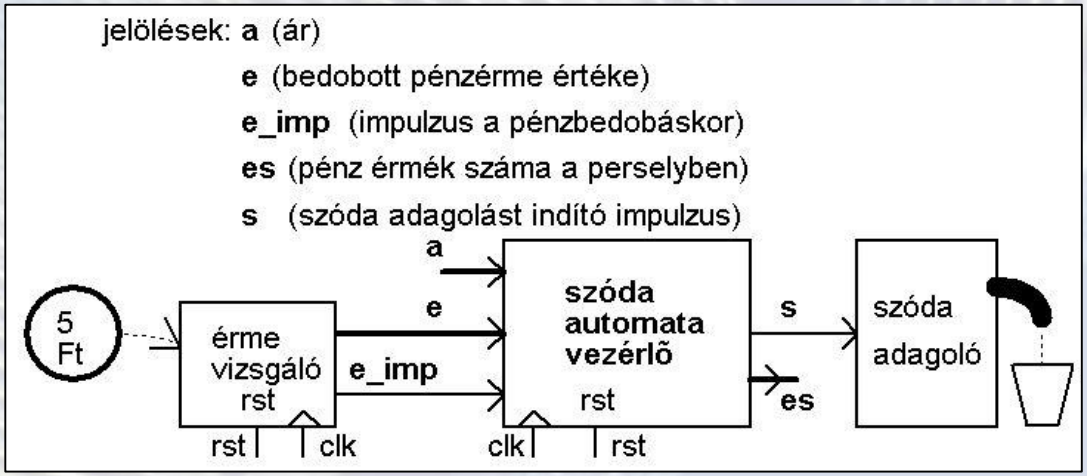
- **Komplex digitális rendszer specifikációja.**
  - A bonyolult feladatok ***teljes algoritmusát*** **magas szintű specifikációval** írjuk le.
  - Különböző specifikációs módszerek léteznek:
    - **HLSM**: ***Magas szintű állapotép specifikáció***
      - Az algoritmus végrehajtása során egyszerre több (magas szintű) művelet is elvégezhető. A specifikáció állapotgépszerű, egy állapotból 2-nél többfelé ágazás is lehetséges.
    - **ASM**: ***Algoritmikus állapotgép specifikáció***
      - Az algoritmusok végrehajtását szekvenciálisan (egymás után végrehajtandó műveletek sorozataként) szervezzük. A specifikáció folyamatábra szerű, egy állapotból tipikusan két felé ágazás lehetséges.

Digitális rendszerek tervezése
HLSM leírás

- A ***komplex digitális rendszer leírása*** **HLSM-el** **(**High Level State Machine)
- A HLSM leírást egyszerű példa segítségével mutatjuk be.
- Egy pénzbedobásos szóda adagoló gép mechanikáját ***vezérlő egységet*** tervezzük meg.
  A gép specifikációja nagyon leegyszerűsített. Pénzt nem ad vissza, nem figyeli, ha a szóda kifogyott, nem figyeli, hogy nem fér több pénz a perselybe, minden érték < 256 stb.

Digitális rendszerek tervezése
HLSM leírás

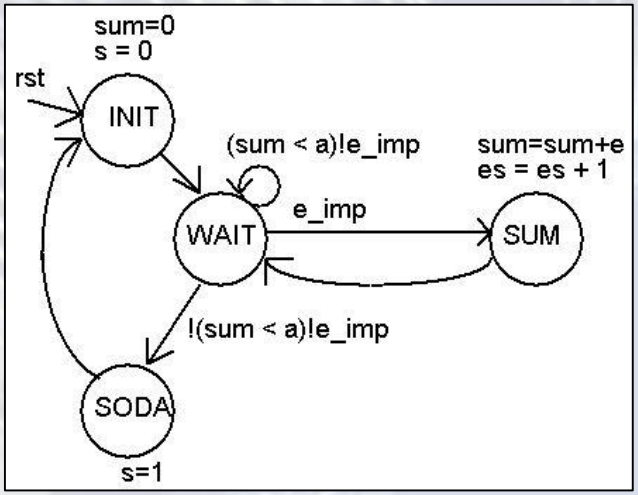
1. Az ***érme viszgálóba*** bedobnak egy pénz érmét.
2. Az ***érme vizsgáló***, ha érvényes az érme, az ***e*** kimenetén kiadja az ***érme értékét*** 8 bites digitális számként és kiad ***egy 1 órajel hosszúságú a rendszerórajellel szinkron impulzust*** az ***e_imp*** kimenetén.
3. A ***szóda automata vezérlő*** (ezt kell megtervezni) 8 bites ***a*** bemenetén be van állítva a szóda ára. Amikor megjön az ***e_imp***, hozzáadja az eddig bejött pénzösszeghez a most bedobottat (***sum = sum+e***) és az érme számláló ***es*** értékét is növeli (***es = es + 1***). Ha az összeg eléri a szóda árát (**sum >= a**), akkor az ***s*** kimenetén kiad egy ***1 órajel hosszú impulzust***. A ***szóda adagoló*** ennek hatására kiadja a poharat és teletölti szódával. Az érmék száma (**es**) a gépkelezőnek szóló információ. Ez alapján tudja, hogy mikor kell üríteni a pénztartályt.



Digitális rendszerek tervezése
HLSM leírás

- A feladat a ***szóda automata vezérlő*** megtervezése (a többi egység adott)
- A feladatot **HLSM állapotgráffal** írjuk le. Nagyon hasonlít az FSM állapotgráfhoz, de:
  - Az **állapotváltás feltétele *magas szintű reláció*** is lehet. (Pl. !(sum < a)) (FSM-ben a feltétel csak 1 bites jelnévként szepelhet, abban reláció jel nem lehet.)
  - A **kimenet *magas szintű művelet (értékadás)*** is lehet. (Pl: sum = sum + e)

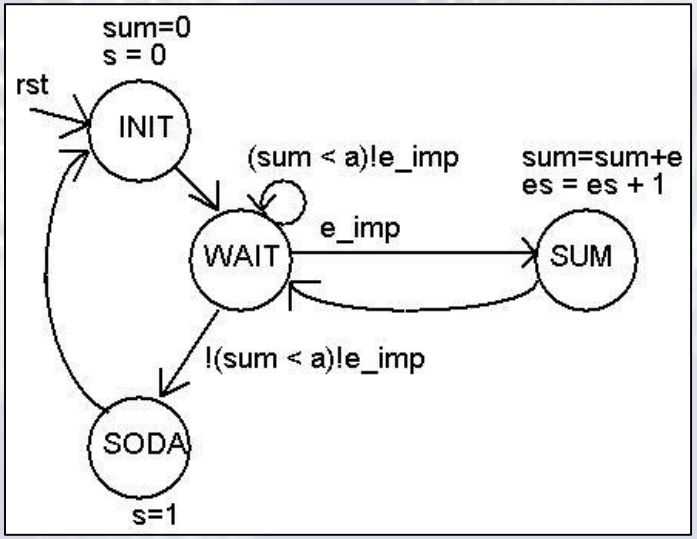
(FSM-ben egy kimeneti jelnek csak konstans érték adható. Az értékadásban nem szerepelhet műveleti jel.)



Digitális rendszerek tervezése
HLSM leírás

A szóda vezérlő ***HLSM*** állapotdigaramja

- Az ***INIT*** állapotban törli a szóda vezérlő ***s*** kimenetét és a bedobott pénz összegét tároló ***sum*** tárolót (de nem törli az ***es*** érme számlálót). Ezután átmegy a pénz bedobás váró ***WAIT*** állapotba.
- A ***WAIT*** állapotban marad, ha nincs pénz bedobás és az összeg még kisebb, mint az ár. Ha pénz bedobás történt (***e_imp = 1***), akkor átmegy az számolásokat elvégző ***SUM*** állapotba. Ha nincs pénz bedobás és az összeg nem kisebb, mint az ár, akkor átmegy a ***SODA*** (szódát kiadó) állapotba.
- A ***SUM*** állapotban ***egyszerre 2 műveletet végez***, hozzáadja a bedobott pénz értékét az eddigi összeghez és növeli az érme számlálót majd visszatér WAIT állapotba.
- ***SODA*** állapotban kiadja a szóda kiadást vezérlő ***s*** jelet és visszamegy INIT állapotba.



Digitális rendszerek tervezése
HLSM leírás

A HLSM leírás információ tartalma

1. Tartlamazza a feladat során ***elvégzendő műveleteket,*** hogy ***mely adatokat kell tárolni*** (értékadások), az adatok honnan hova mennek (***adat utakat***).
2. Tartalmazza a feladat során ***figyelendő feltételeket***, amelyek befolyásolják az állapotátmenteket.
3. Tartalmazza ***a feladat algoritmusát.*** A műveleteket milyen sorrendben kell elvégezni (a feltételeket figyelembe véve). Ez a gráf felépítésébe van belekódolva.

- Az ***1. és 2. alapján*** meg lehet alkotni a feladat hardverének egy olyan részét, mely ***képes az elvégzendő műveletek (akár egyszerre több) végrehajtására és a figyelendő feltételek előállítására***. Ennek a hardver résznek a neve **ADATSTRUKTÚRA**

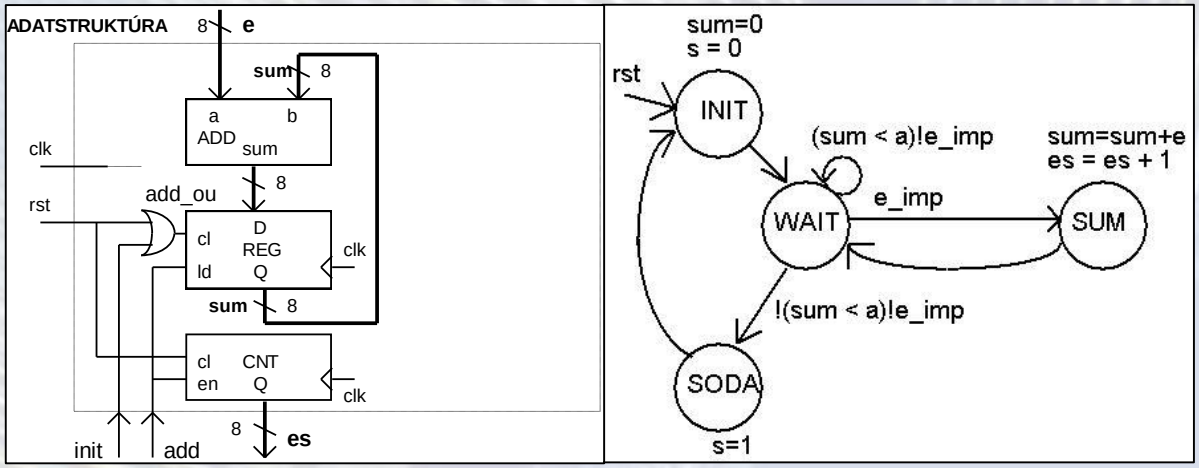
Digitális rendszerek tervezése
ADATSTRUKTÚRA

- A példa feladatban ***elvégzendő műveletek***:

  sum = 0, sum = sum + e Kell egy ***összeadó*** (előállítja (sum+e)-t. Kell egy törölhető, tölthető *regiszter*, kimenete a sum. Minden újrakezdéskor töröljük egy ***init*** nevű **vezérlő jellel** és rst-re is. Egy ***add*** nevű *vezérlőjellel* írjuk be (sum+e)-t). Az *adatút*: a *sum* regiszter bemetére megy az összeadó kimenete (add_ou), az összeadó bemeneteire megy a *sum* regiszter kimenete, és az *e* adat.

- ***es = 0, es = es +1*** kell egy törölhető, engedélyezhető ***felfele számláló.*** A rst-re törölni kell, de új szóda kiadás előtt nem. Az összeadással egyidőben kell inkrementálni, erre az *add vezérlőjel* megfelel.

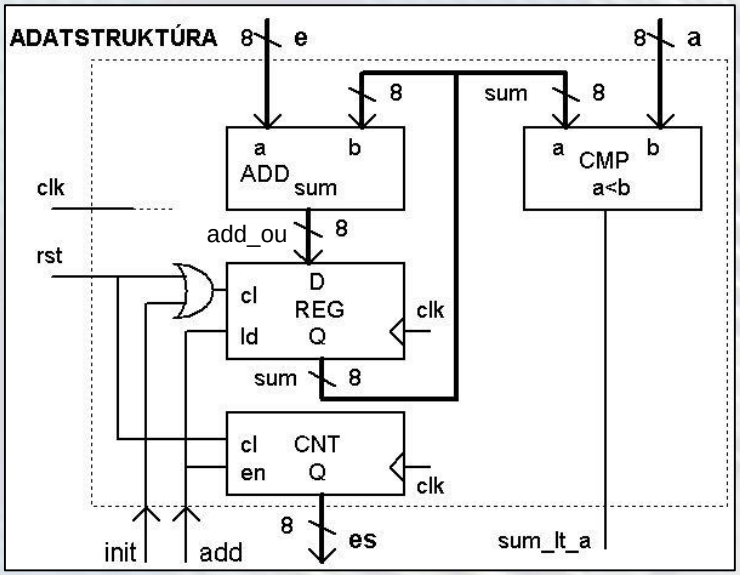
- ***s = 0, s = 1*** ez lehet egy ***hagyományos FSM kimeneti jel***

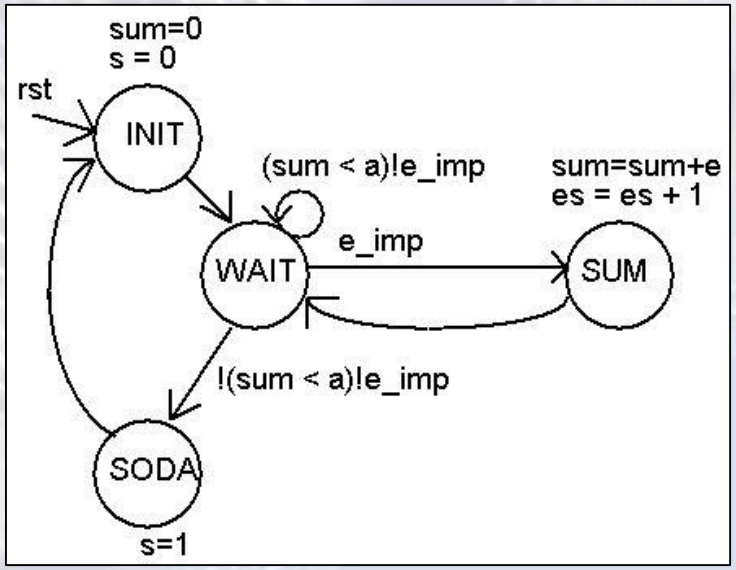


Digitális rendszerek tervezése
ADATSTRUKTÚRA

- A ***figyelendő feltételek***:
  e_imp: kívülről jön, hagyományos ***FSM feltétel jel***
  (sum < a): nagyság komparátor kimenete állítja elő,
  neve: ***sum_lt_a***.
  (sum < a)!e_imp, !(sum < a)!e_imp:

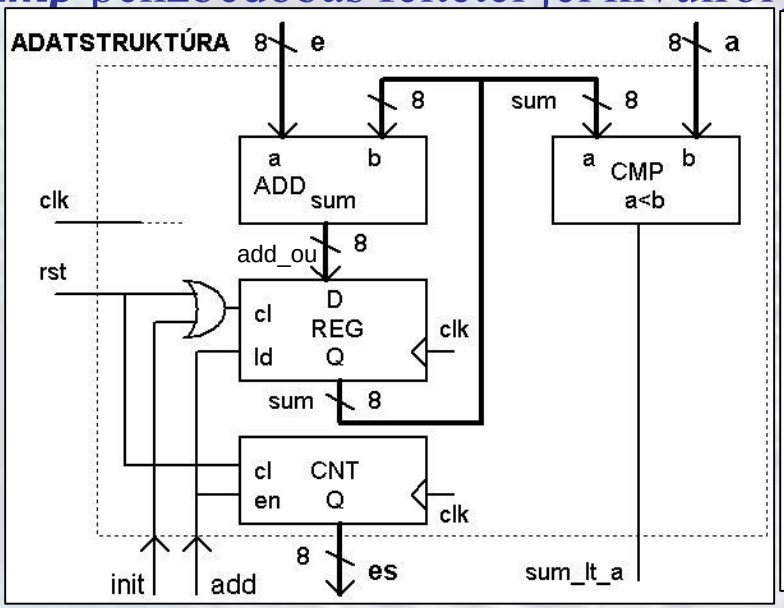
A vezérlő logikában írjuk le a megfelelő viselkedést (lásd később).
(sum_lt_a&!e_imp, !sum_lt_a&!e_imp)

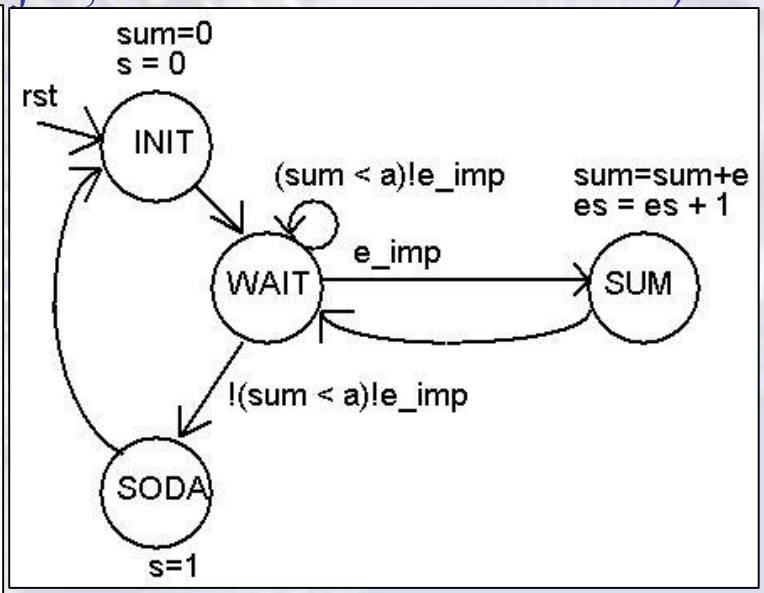




Digitális rendszerek tervezése
ADATSTRUKTÚRA

- Az előbbiek alapján lerajzoljuk a minta feladat ***adatstruktúráját***.
- ***Az adatstuktúra képes a feladathoz szükséges minden művelet elvégzésére, ha a művelet elvégzéséhez szüksége vezérlő jelet aktivizáljuk***:
  rst = 1 hatására ***sum = 0, es = 0***, **init = 1** hatására ***sum = 0***
  add = 1 hatására ***sum = sum + e, es = es + 1***
- ***Az adatstruktúra előállítja az adatstuktúrából származtatható feltétel jeleket***: Az összeg kisebb mint az ár: ***sum_lt_a***
  (Az ***e_imp*** pénzbedobás feltétel jel kívülről jön, nem kell külön előállítani.)

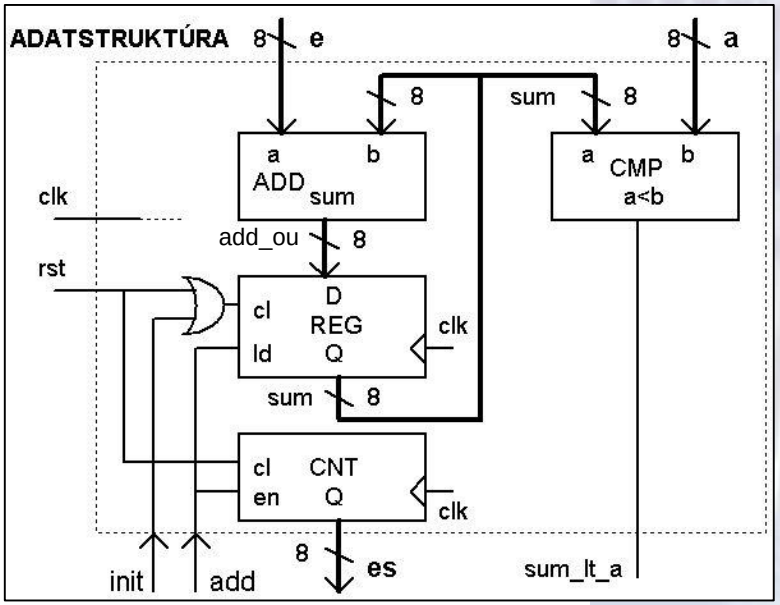




Digitális rendszerek tervezése
ADATSTRUKTÚRA

Az adastruktúra Verilog leírása

```
21  module adat_strukt(input clk, rst, init, add,
22     input [7:0] a, e, output sum_lt_a, output reg [7:0] es);
23
24  wire [7:0] add_ou;
25  reg [7:0] sum;
26
27  //összeadó
28  assign add_ou = sum + e;
29
30  //pénz öszegző regiszter
31  always@(posedge clk)
32  if(rst | init) sum <= 8'h00;
33  else if(add) sum <= add_ou;
34
35  //érme számláló
36  always@(posedge clk)
37  if(rst) es <= 8'h00;
38  else if(add) es <= es + 1;
39
40  //feltétel logkika
41  assign sum_lt_a = (sum < a);
42  endmodule
```



Digitális rendszerek tervezése
VEZÉRLŐ

- Szükség van a műveleteket azt adatstruktúrával elvégeztető hardver egységre. Ennek a neve **VEZÉRLŐ.**
- A vezérlő valósítja meg az algoritmust az adatsruktúra segítségével. A ***feltételeket*** figyelembe véve (a példában *e_imp*, *sum_lt_a*), megfelelő sorrendben előállítja az adatstruktúrát működtető ***vezérlő jelek***et (példában *init*, *add*).
- ***Egyszerre több műveletet*** is képes elvégezni. (A példában: sum = sum+1, es = es + 1 egyszerre történik.)
- A kívülről (nem az adatstuktúrából) jövő feltétel jeleket ***parancs jeleknek*** nevezzük. Például az *e_imp* itt egy parancs jel.
- A vezérlőjelek egy része a működési állapotról adhat visszajelzést. Ezeket nevezzük ***státus jelek***nek. Pl. A készülék elkészült a munkával (*ready*). A példában ilyen jelet nem használtunk, de az ***s*** jel akár ezt is jelezhetné.

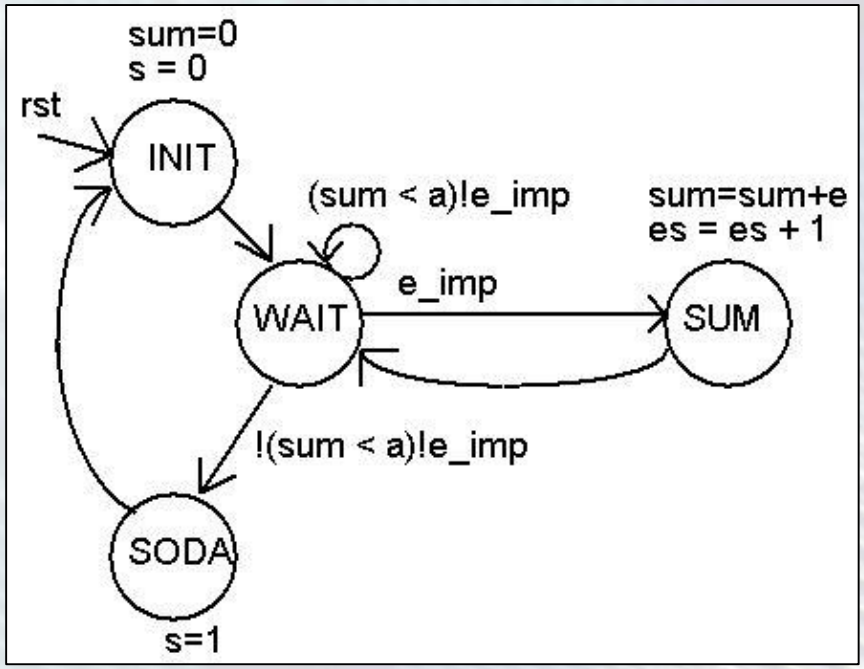
Digitális rendszerek tervezése
VEZÉRLŐ

- A vezérlőt meg tudjuk valósítani hagyományos állapotgéppel (FSM)
- **Az állapotgép gráf struktúrája megegyezik a HLSM gráféval.**
- **A HLSM gáf magas szintű feltétel jeleit helyettesítjük az adatstruktúrából és kívülről jövő jelekből előállított feltétel jelekkel.**
  A példában:
  !(sum < a)!e_imp helyett ***!sum_lt_a&!e_imp***
  (sum < a)!e_imp helyett ***sum_lt_a&!e_imp***
- **A HLSM gráf magas szintű műveleteit helyettesítjük a műveleteket az adatstruktúrával végrehajttató vezérlő jelekkel.**
  A példában:
  sum = 0 és helyett ***init***
  sum = sum + e és ***es = es+1*** helyett ***add***.

Digitális rendszerek tervezése
VEZÉRLŐ

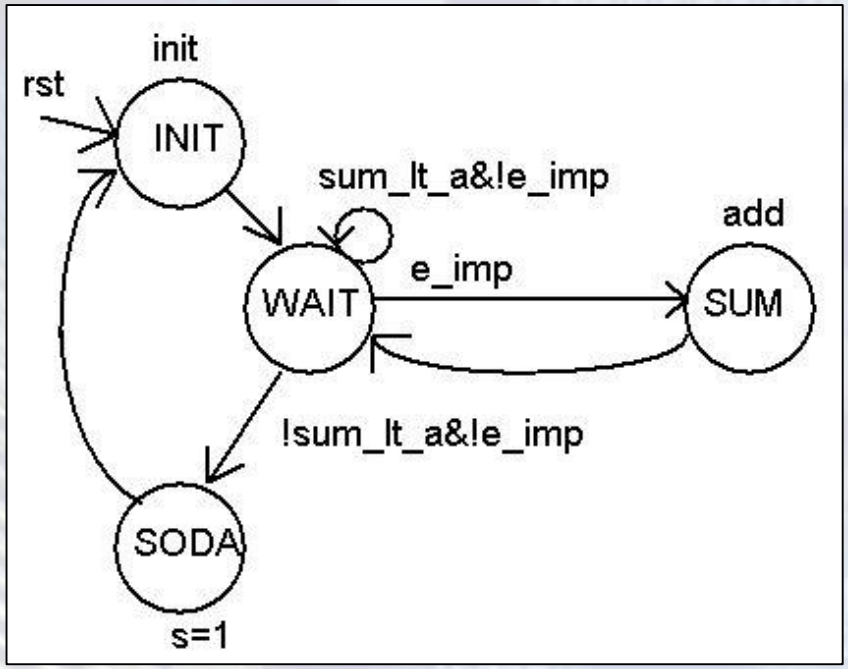
HLSM állapotgráf

(állapotdiagram)



VEZÉRLŐ FSM állapotgráf

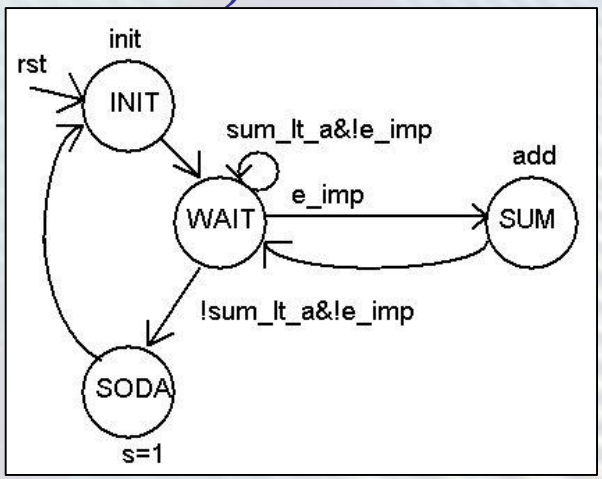
(állapotdiagram)



Az **s=1** jelet a vezérlő SODA állapotában adjuk ki az FSM-el, nem kell hozzá külön regiszter.

Digitális rendszerek tervezése
VEZÉRLŐ

- **A vezérlő FSM Verilog leírása**
  - Az állapotregisztert, next state logikát, kimeneti logikát külön egységekben írtuk le. (Ahogy tanultuk.)

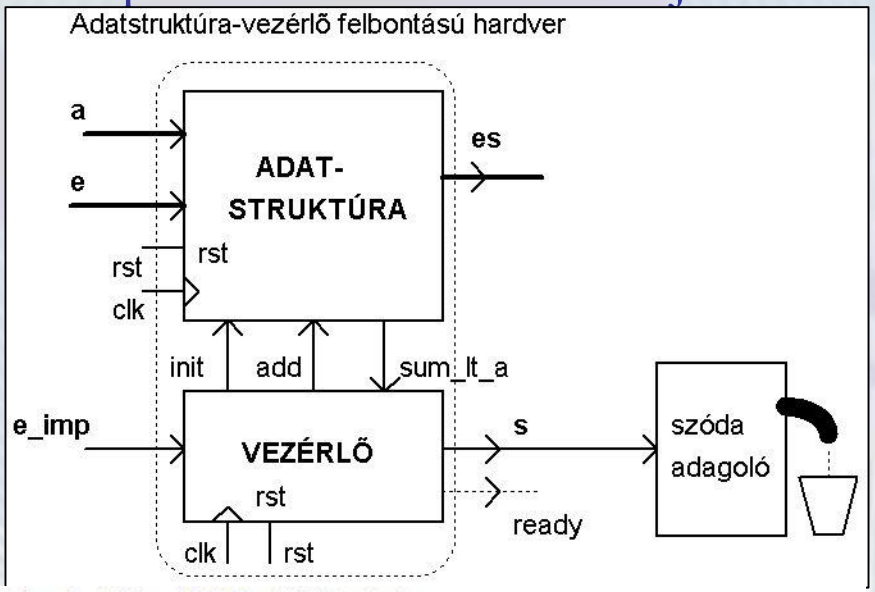



```
22  module vezerlo( input clk, rst, e_imp,
23                  sum_lt_a, output init, add, s);
24
25  reg [1:0] state, next_state;
26  localparam INIT = 2'b00, WAIT = 2'b01,
27             SUM = 2'b10, SODA = 2'b11;
28  //állapotrgeiszter
29  always@(posedge clk)
30  if(rst) state <= INIT;
31  else    state <= next_state;
32
33  //next_state logika
34  always@(*)
35  case(state)
36  INIT: next_state <= WAIT;
37  WAIT: if(e_imp) next_state <= SUM;
38        else if(!sum_lt_a) next_state <= SODA;
39        else next_state <= WAIT;
40  SUM:  next_state <= WAIT;
41  SODA: next_state <= INIT;
42  default: next_state <= INIT;
43  endcase
44
45  //kimeneti logika
46  assign init = (state == INIT);
47  assign add = (state == SUM);
48  assign s = (state == SODA);
49  endmodule
```

Digitális rendszerek tervezése
ADATSTRUKTÚRA + VEZÉRLŐ

A teljes rendszer blokkvázlata

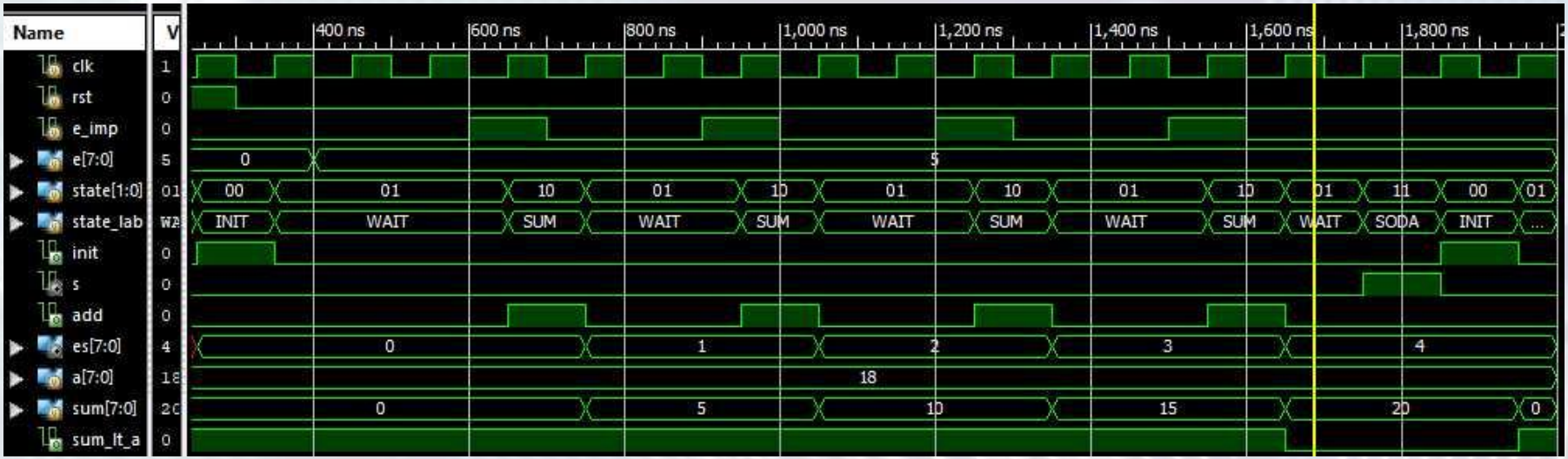
- Az adatstruktúra és a vezérlő összekapcsolásával elkészült a teljes szóda automata vezérlője
- A teljes rendszer blokkvázlata
- A top modul Verilog leírása




```
21  module soda(input rst, clk, e_imp, input [7:0] a,e,
22                                  output [7:0] es, output s);
23
24  //parancs jel és feltétel jelek:
25  wire sum_lt_a;
26  //vezérlő jelek:
27  wire init, add;
28
29  adat_strukt soda_adat_strukt(.clk(clk), .rst(rst), .init(init), .add(add),
30                    .a(a), .e(e), .sum_lt_a(sum_lt_a), .es(es));
31  vezerlo soda_vezerlo(.clk(clk), .rst(rst), .e_imp(e_imp), .sum_lt_a(sum_lt_a),
32                       .init(init), .add(add), .s(s));
33  endmodule
```

Digitális rendszerek tervezése
ADATSTRUKTÚRA + VEZÉRLŐ

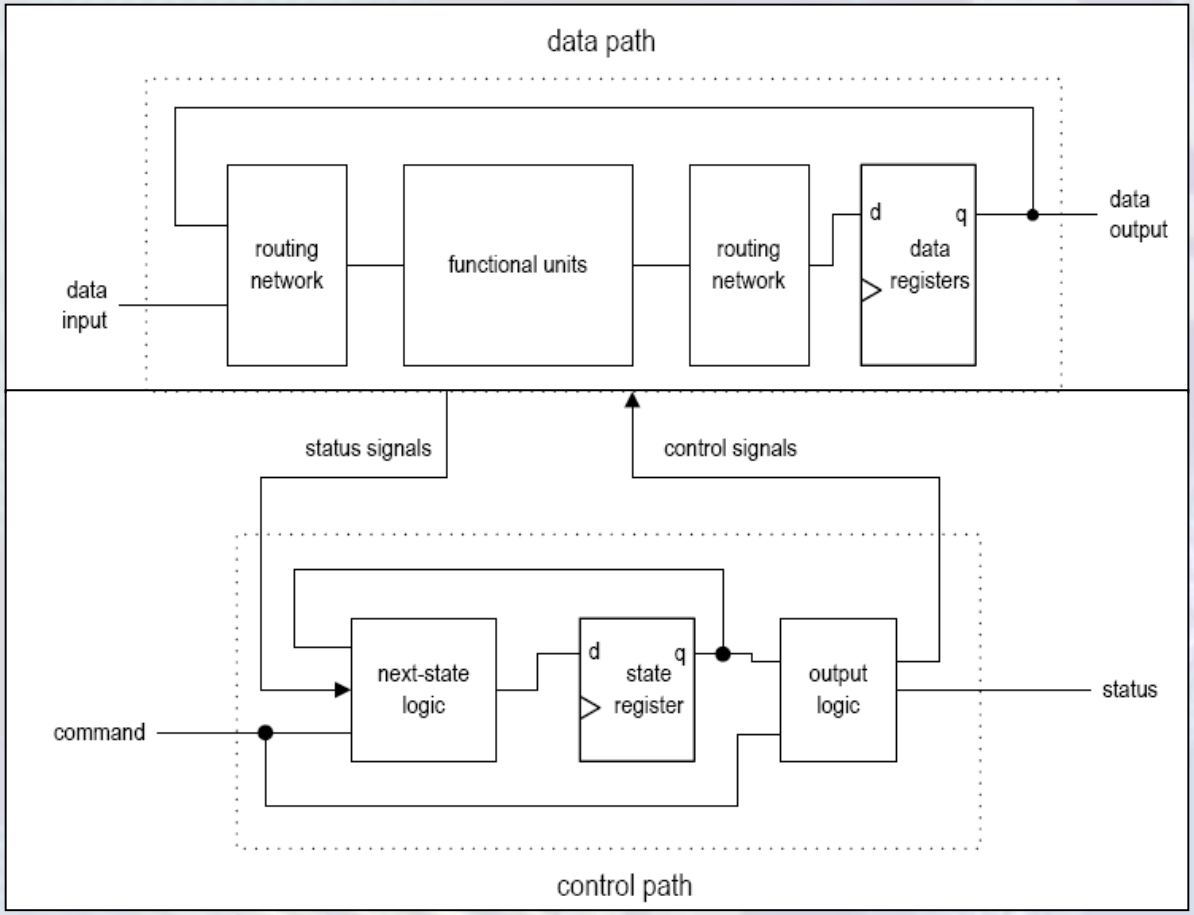
A rendszer szimulációja



- A szóda ára a=18 Ft, a 4 db pénzérme e = 5 Ft-osokból áll. Az utolsó bedobása után (sum = 20) s = 1 lesz (szóda kiadás indítása), majd a vezérlő alaphelyzetbe áll és törli a sum regisztert.

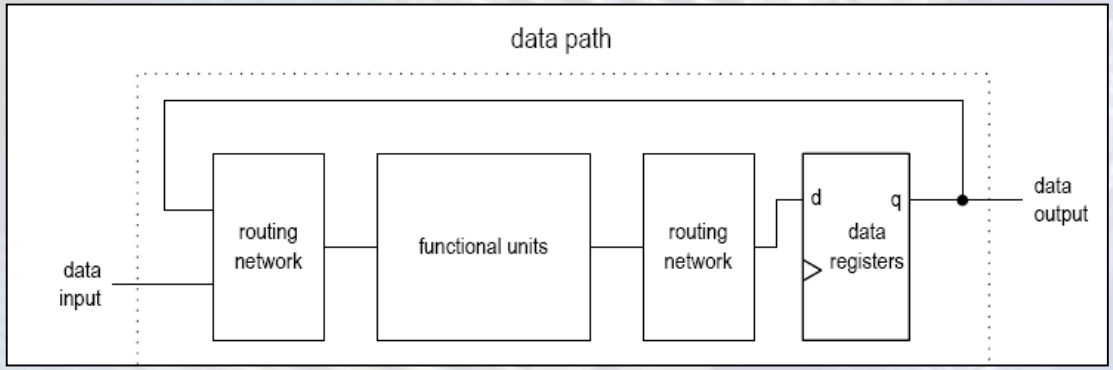
Digitális rendszerek tervezése

- Két fontos alrendszert különböztetünk meg
  ADATSTRUKTÚRA + VEZÉRLÉS



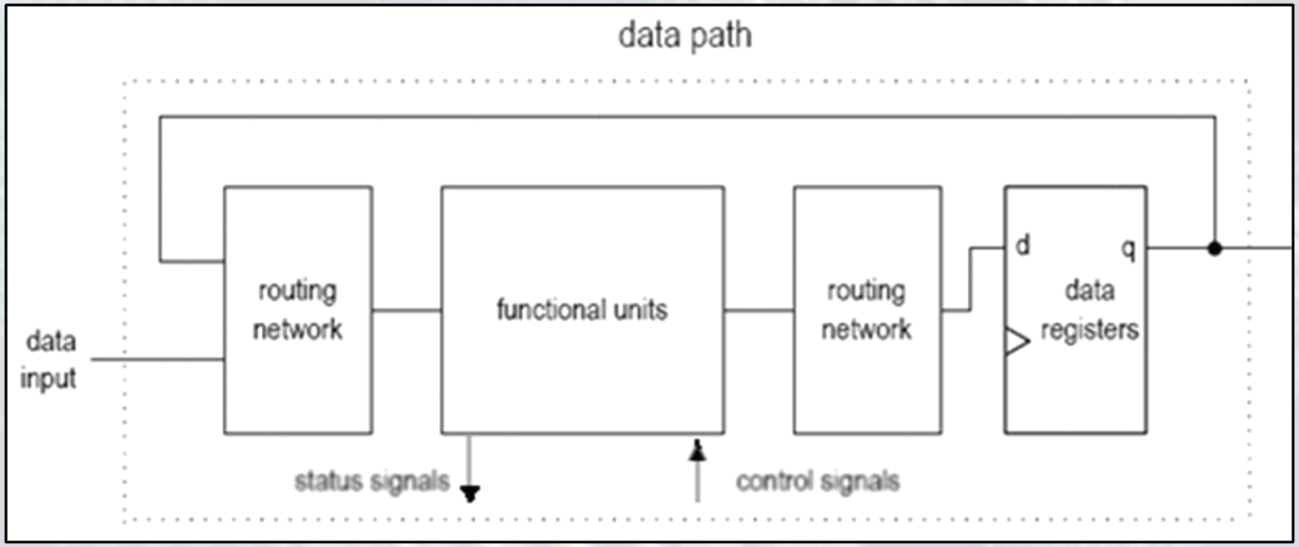
Digitális rendszerek tervezése

- **Az adatstruktúra, azaz az adatfeldolgozó rész**
  - Itt történik a feladathoz szükséges műveletek végrehajtása
  - Az adatstruktúra elemei:
    - **Adatregiszterek**: az algoritmus változóinak aktuális értékét tárolják, méretük alkalmazásfüggő
    - **Funkcionális egységek**: az ismert/tanult műveletvégzők, mint összeadó, komparátor, inkremeneter, stb.
    - **Adatútválasztók**: Multiplexer hálózatok, feladatuk, ugyanaz, mint a multifunkciós regiszterek esetén



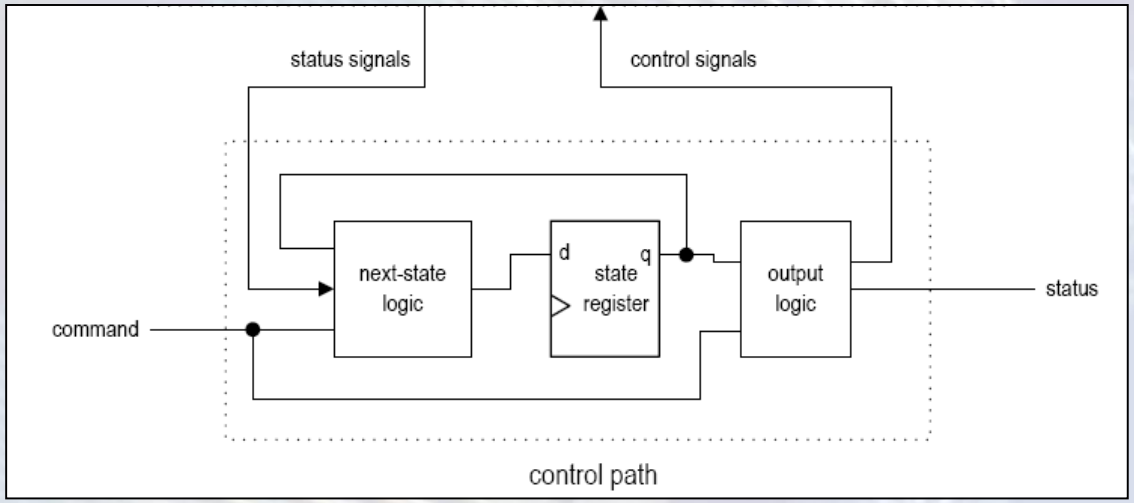
Digitális rendszerek tervezése

- **Az adatstruktúra, azaz az adatfeldolgozó rész**
  - Az ***elsődleges adat bemenet és adat kimenet formátuma alkalmazásfüggő*** (bitek, bitvektorok, előjel nélküli számok, 2-es komplemens számok stb.)
  - Az ***adatstruktúra státuszjelei a vezérlőbe jutnak***, a ***vezérlőből származó belső vezérlőjelek*** pedig a feltételeket figyelembe véve ***működtetik az adatstruktúrát***



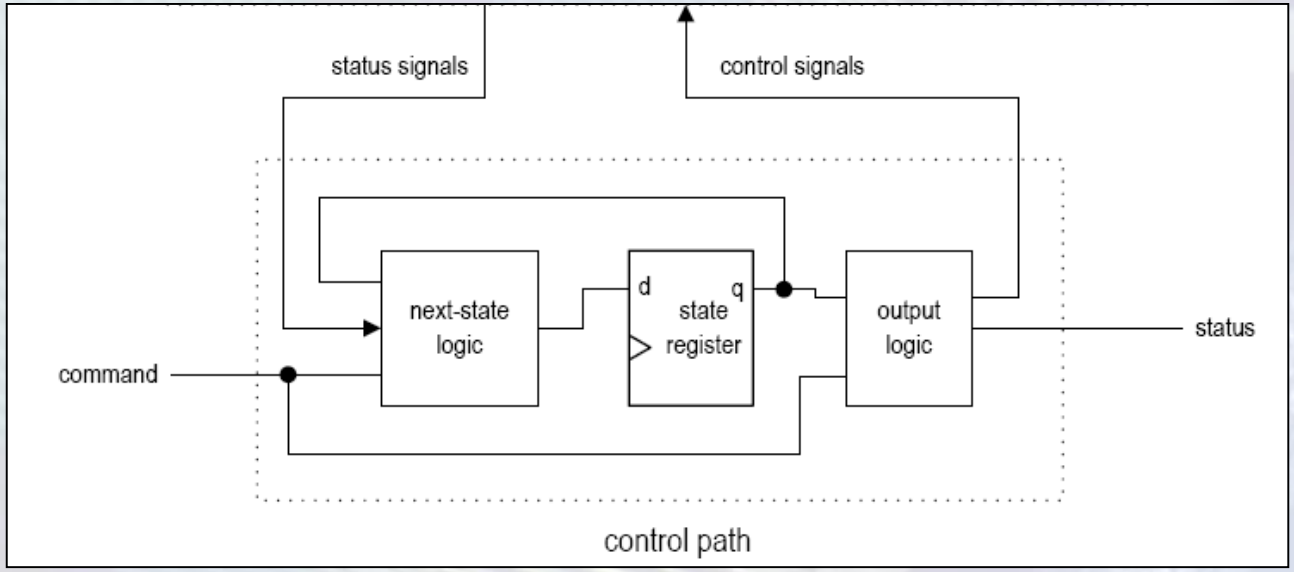
Digitális rendszerek tervezése

- **A vezérlési alrendszer: FSM**
  - A bemeneteket megkülönböztetjük
    - **Külső parancsjelek** a vezérlőnek (START, STOP, üzemmód stb.)
    - **Belső státuszjelek** (***az adatstruktúrából***) (pl. TC, A>B stb.)
  - A kimeneteket is megkülönböztetjük
    - **Belső vezérlőjelek:** az adatstruktúra működtetésére (LD, EN, DIR stb.)
    - **Külső jelzések (státus jelek)**: működési állapot jelzése külső egységnek, felhasználónak (READY, ERROR, VALID…)



Digitális rendszerek tervezése

- **Az vezérlő FSM tulajdonságai**
  - Mindig ***szinkron FSM***, minden órajel élre működik
  - Lehet Moore, lehet Mealy, (lehet vegyes)
  - A ***Moore modell*** választása több állapotot igényel, de ***átláthatóbb a tervezés***.



Digitális rendszerek tervezése

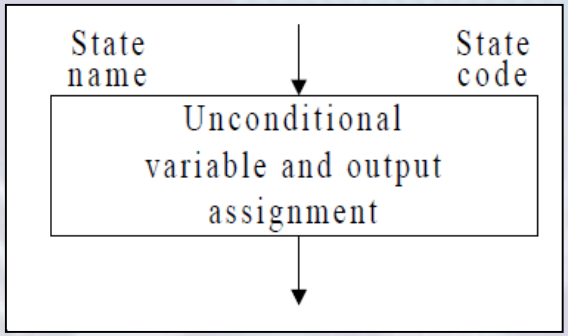
- Bonyolultabb HLSM specifikálásához és felépítéséhez gyakorlat és tapasztalat szükséges.
- Azonban nem igényli a működés elemi lépésekre bontását, hanem meghagyja a hardver párhuzamos működésének (több művelet egyszerre elvégzése) lényegét. (Pl. A szóda mintapéldában az bedobott pénz hozzáadása és az érmeszámláló inkrementálá-sa.)

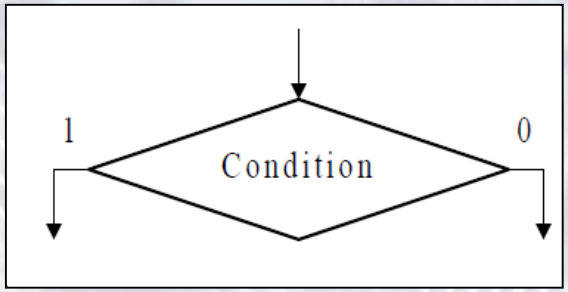
Digitális rendszerek tervezése

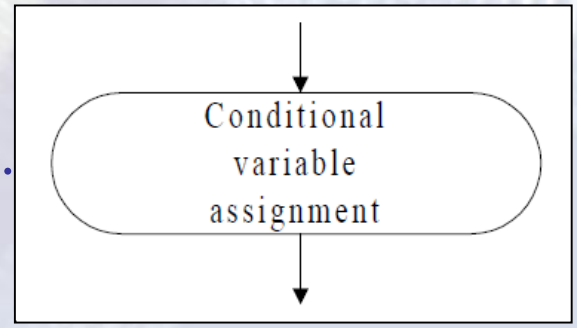
- **Az ASM, azaz algoritmikus állapotvezérlő**
  - Az ASM egy ***folyamatábra jellegű specifikáció***
  - A működés itt is órajel vezérelt (digitális HW), minden állapotváltozás egy-egy órajelet igényel.
  - Az ASM erősen strukturált jellegű, nem annyira szabad szerkezetű, mint egy HLSM (szoftveresebb…)
  - Az ***állapotátmenetek nem tetszőlegesek*** (mint a HLSM-nél), hanem ***2 következő állapot***ra korlátozottak, hasonlóan a számítógépes algoritmusok lépésenkénti stílusához.
  - Ebből következően ***nem annyira hatékony HW tervezéshez, mint a HLSM***. Arra használjuk, hogy segítsen bevezetni a programvezérelt processzor működését.

Digitális rendszerek tervezése

- **Az ASM, azaz algoritmikus állapotvezérlő**
  - Az ASM leírás elemei:
  - **Állapot doboz:** Hozzárendeljük az állapot kódját, bele írjuk belső regiszterváltozók és a Moore típusú kimenetek (regiszterek) értékadását.
  - **Feltétel doboz:** Az adatstruktúra vagy a következő állapot logika működését vezérli (feltétel függő 2-felé ágazás).
  - **Kimeneti doboz:** Az esetleges feltételes értékadások művelete, jellemzően a Mealy típusú kimenetekhez (mi nem használjuk).

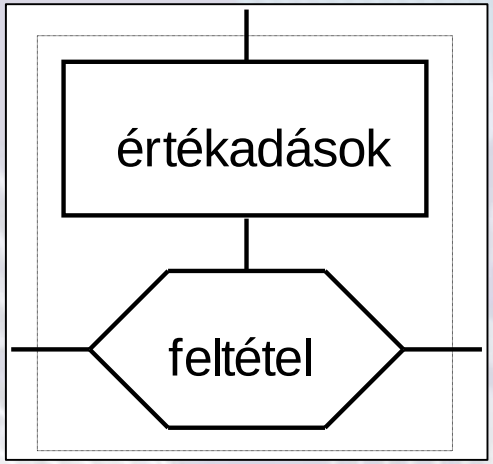






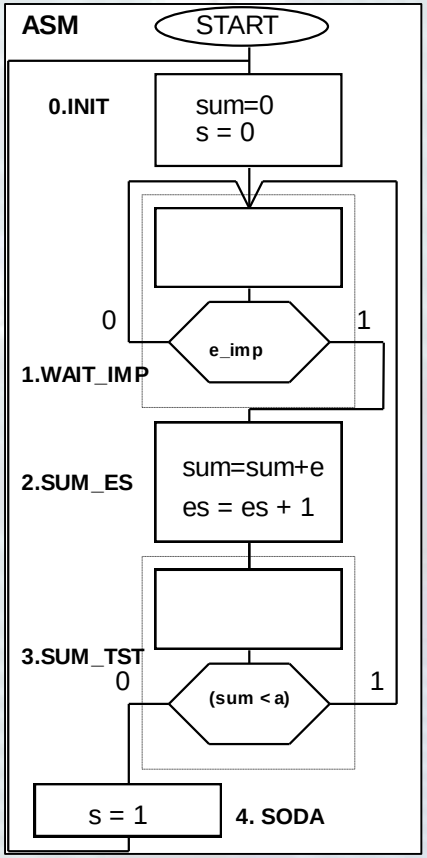
Digitális rendszerek tervezése

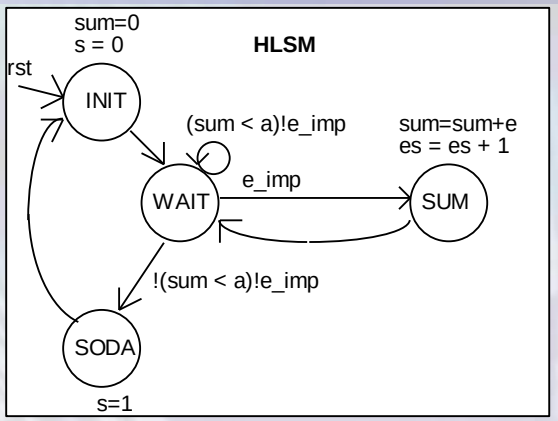
- **Az ASM, azaz algoritmikus állapotvezérlő**
- **Az ASM blokk szerkezete:**
  - ***Minden*** eddigi „FSM” ***állapot egy ASM blokk lesz*** (egyértelmű megfeleltetés)
  - Az ***ASM blokkban mindig van állapotdoboz***
  - ***Az ASM blokk tartalmazhat egy*** vagy több ***feltételvizsgálatot*** és feltételes értékadást. (Utóbbit mi nem használjuk és feltétel vizsgálatból is csak egyet.)
  - A tipikus használati mód nagyon egyszerű ASM blokkokkal dolgozik (csak állapot, vagy állapot+feltétel)



Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- Példa: Az előző szóda automata.





Az ASM-ben állapotonként csak 2 felé ágazást engedünk meg. HLSM-ben a WAIT-ből 3 fele ágazik!

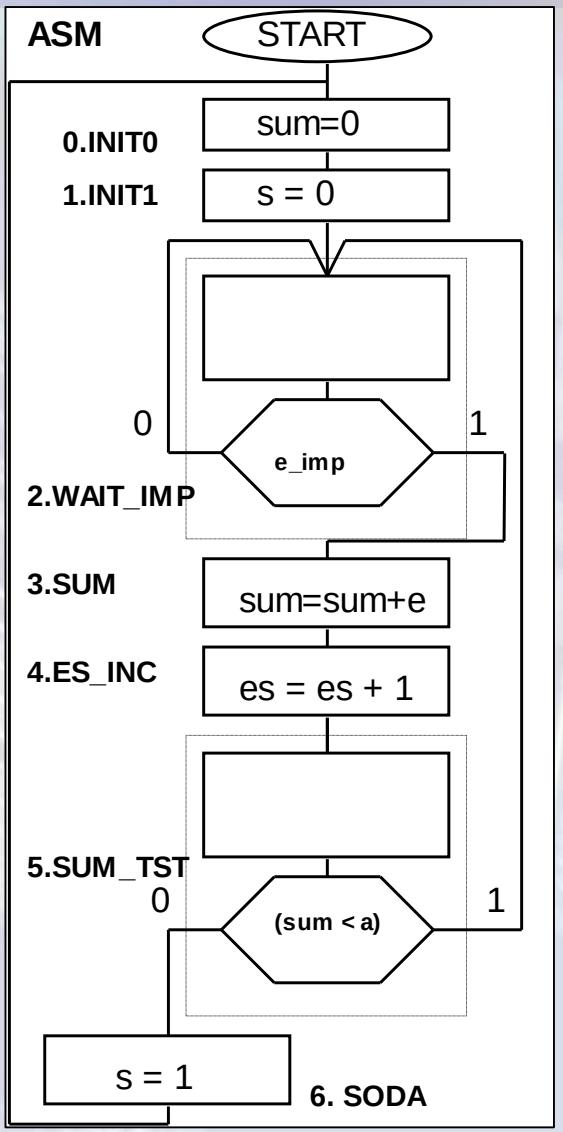
Ezért a HLSM WAIT állapota 2 db ASM állapottal valósítható meg: WAIT_IMP és SUM_TST.

A feltétel vizsgálatot is szétszedtük 2 egymás utáni vizsgálatra: e_imp, (sum<a)

Az e_imp az órajellel szinkron, 1 órajelig tart és az órajelhez képest ritkán jön. Először ezt megvárjuk a WAIT_IMP állapotban. Utána elvégezzük az összegző műveleteket a SUM_ES állapotban. Ekkor az e_imp már biztosan 0. Ezután már csak a módosult sum-ot ellenőrizzük, mert az !e_imp biztosan teljesül. Tehát SODA állapotba (a SUM_ES és SUM_TST-n keresztül) akkor jut, mint a HLSM-ben: !e_imp & !(sum < a). Láthatóan az ASM kicsit másfajta gondolkodást igényel, mint a HLSM, a csak kétfelé ágazási lehetőség miatt.

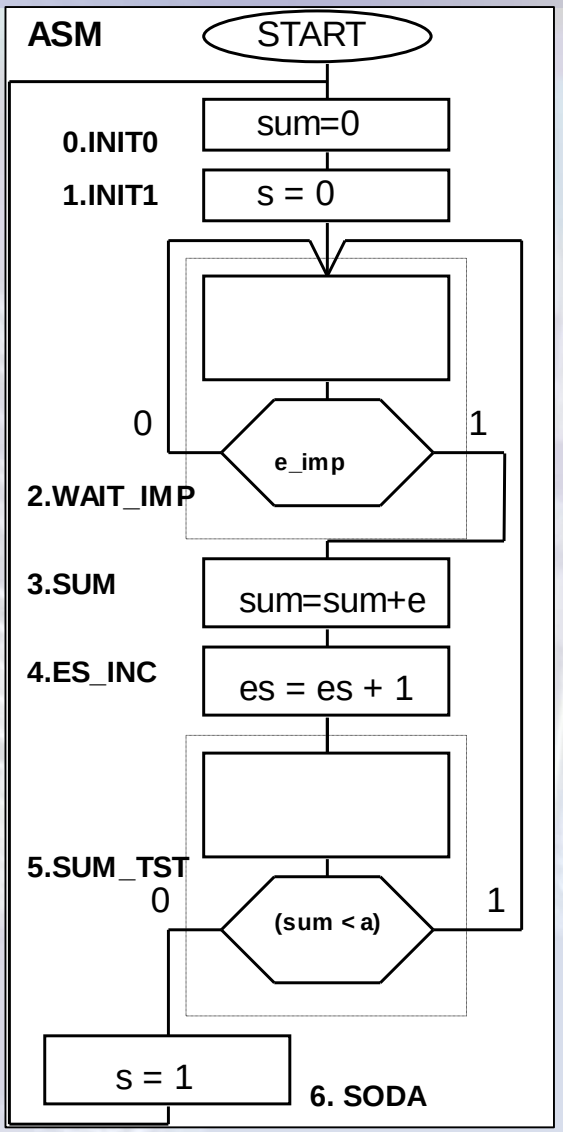
Digitális rendszerek tervezése

- Amint később látni fogjuk, a processzor egyszerre csak egy műveletet tud elvégezni.
- Az ASM ábrát most ennek megfelelően alakítottuk át, egy állapotban csak egy művelet (értékadás) szerepel. Így jól látszik, hogy az állaptok száma ettől is megnő.
- Az ASM leírást majd az assembly programozásnál fogjuk használni a programok megtervezéséhez.



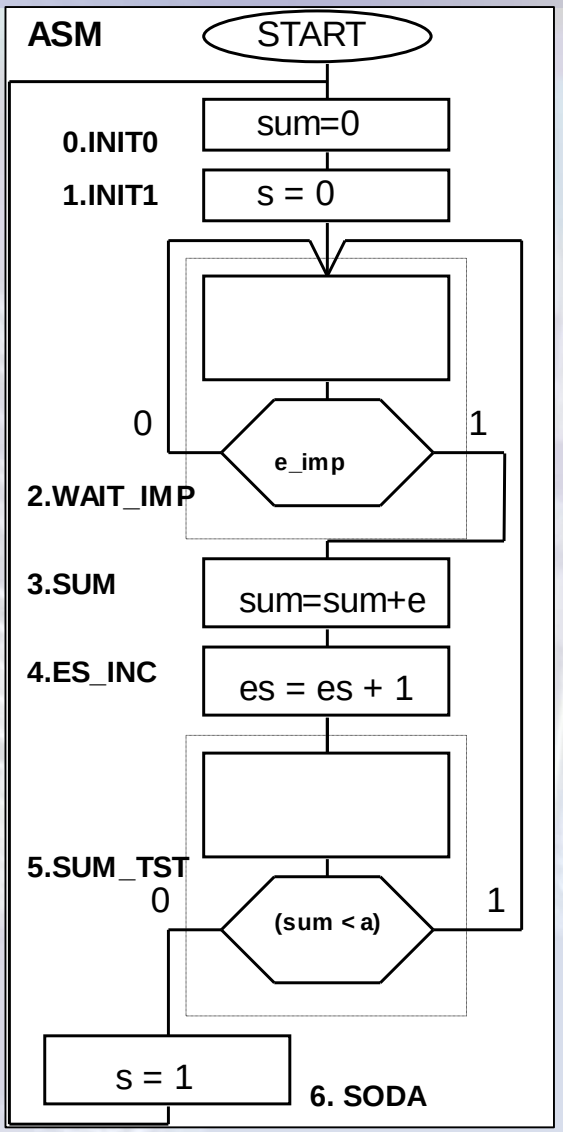
Digitális rendszerek tervezése

- **Az ASM modellben a jellemző állapotátmenetek:**
  - CONT: folytatás (feltétel nélküli lépés a következő állapotra)
  - JUMP: feltétel nélküli ugrás tetszőleges állapotra (akár saját magára is)
  - CJMP: feltételes ugrás tetszőleges állapotra, feltétel nem teljesülése esetén CONT (folytatás)



Digitális rendszerek tervezése

- Az ASM diagrammon az ***egymást követő állapotokat növekvő számokkal kódoltuk*** úgy, hogy ***egy-egy feltételes elágazásból is egyik irányban 1-gyel nagyobb kódú állapot*** következik.
- Így az ***állapotregiszter*** akár a egy ***tölthető számláló*** is lehet. Ebbe feltételes elágazásnál a feltétel egyik értéke esetén a következő állapot kódját kell tölteni, egyébként pedig inkrementál. Pl. SUM_TST-ben (5-ös állapot) (sum<a)-ra a 2-es állapot kódot kell tölteni (WAIT_IMP), egyébként inkrementálni (6-os állapot SODA). Feltétel nélküli elágazásnál mindenképpen tölteni kell (SODA 6-ból INT0 0).



Digitális rendszerek tervezése

- **Az ASM algoritmikus állapotvezérlő**
- A vezérlő tehát lehet egy egyszerű ***bináris számlálón alapuló vezérlőegység*** (nem szükséges a sokkal általánosabb FSM állapotvezérlő felépítés)
- **Az számláló állapotátmenetek vezérlése:**
  - CONT: state <= state + 1; // 1-el növelés
  - JUMP : state <= new_state; // feltétel nélküli ugrás
  - CJMP : if (COND) then state <= new _state // feltételes ugrás
    else state <= state +1 // 1-el növelés
- Ezt a feladatot egy ***törölhető, tölthető bináris felfelé számláló*** tudja biztosítani → Nevezhetjük állapotszámlálónak vagy akár utasításszámlálónak.
  A processzorban majd **program counter**nek (**PC**) nevezzük.

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 **adatstruktúrára és vezérlőre** bontjuk a feladatot.
- A vezérlő lehet Moore, vagy Mealy típusú, vagy akár kevert módon működő. A Moore-t egyszerűbb megtervezni, de több állapotot igényel.

Digitális technika
7. EA vége