



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

COVID-19 előadások

- Ebben az évben az előadások további két teremben is, kivetítőről követhetők, a kapcsolat így nem elég közvetlen
- Ennek javítására beiktatunk visszajelzési lehetőséget
- Bármelyik teremből kérdések, megjegyzések emailben küldhetők a benes@mit.bme.hu címre, de csak ... @edu.bme.hu címről, tárgy: DIGIT Résztemák végén a kérdéseket megnyitjuk és próbálunk válaszolni

Digitális rendszerek tervezése

- **Eddig tanultak:**
- **Kombinációs logikák**
 - Logikai függvények,
 - Kétszintű realizáció, SOP, (POS), minimalizálás kell
 - Többszintű realizáció, több kimenetű hálózatok
 - *Funkcionális elemek, modulszintű építkezés*
- **Szinkron sorrendi hálózatok**
 - Órajel él vezérelt működés, DFF: mintavételezés, tartás
 - Kimeneti érték korábbi állapotok függvénye is
 - *Véges állapotú vezérlő*, tetszőleges állapotátmenetek
 - *Multifunkciós regiszterek, számlálók, memóriák*
 - Az órajel ütemez, minden műveletre T_{clk} idő jut

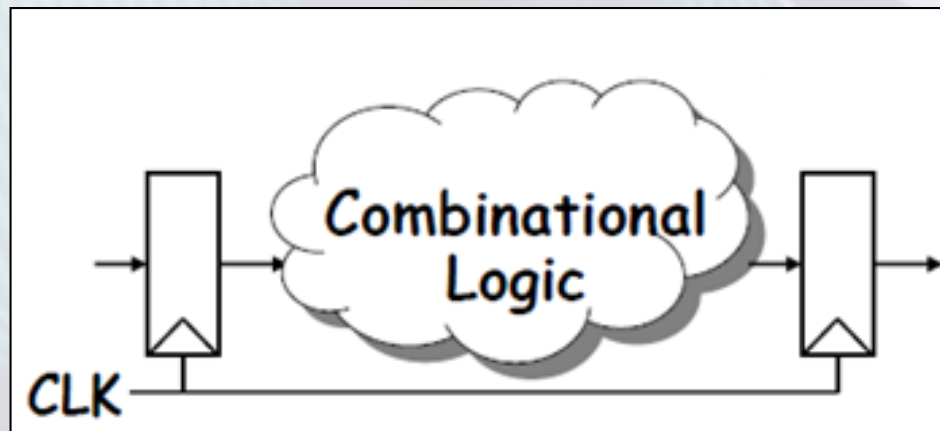
Digitális rendszerek tervezése

- Ismerünk *kombinációs funkcionális elemeket*, amelyek:
 - Bizonyos matematikai *műveleteket* el tudnak végezni (Összeadó, kivonó, inkrementer, dekrementer)
 - Adatok közötti *relációkat* tud jelezni (teljes funkciós komparátor)
 - *Adat út választásra* alkalmasak (multiplexer)
- Ismerünk *szinkron sorrendi funkcionális elemeket*, melyek:
 - *Adat tárolásra* alkalmasak (tölthető regiszter, multifunkciós regiszter)
 - Számolásra (counter), shiftelésre (shiftgeriszter) alkalmasak
- Az eddig tanultak alapján tudunk *állapotgépet tervezni* állapotgráfos leírás alapján.
- Ezek felhasználásával már *komplex digitális rendszer* létrehozható

Digitális rendszerek tervezése

Regiszter transzfer tervezési modell

- Összetett digitális rendszerek felépítése:
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, adat transzformáció **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*
 - A szinkron regiszterek működésének leírása:
 - Él vezérelt működés **always @ (posedge clk)**
 - Nem blokkoló értékadás **<=**
 - *Gazdag operátorkészlet és adat formátumok a különböző műveletekhez*
 - Normál és előjeles (2's komp.) 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:** Magas szintű 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

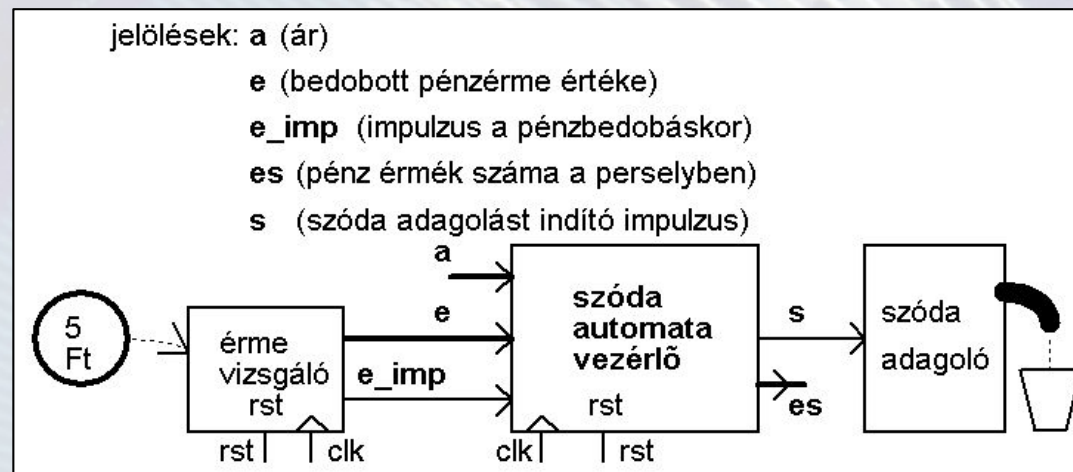
Digitális rendszerek tervezése

- A *komplex digitális rendszer leírására* több módszer is létezik.
- Egyik módszer a High Level State Machine (**HLSM**)
- Ezt a leírást egyszerű példa segítségével mutatjuk be.
- Egy pénzbedobásos szóda adagoló gép *vezérlő egységét* tervezzük meg. (A gép specifikációja nagyon leegyszerűsített. Pénzt nem ad vissza, nem figyel, ha a szóda kifogyott, nem figyel, hogy nem fér több pénz a perselybe, minden érték < 256 stb.)

Digitális rendszerek tervezése

HLSM leírás

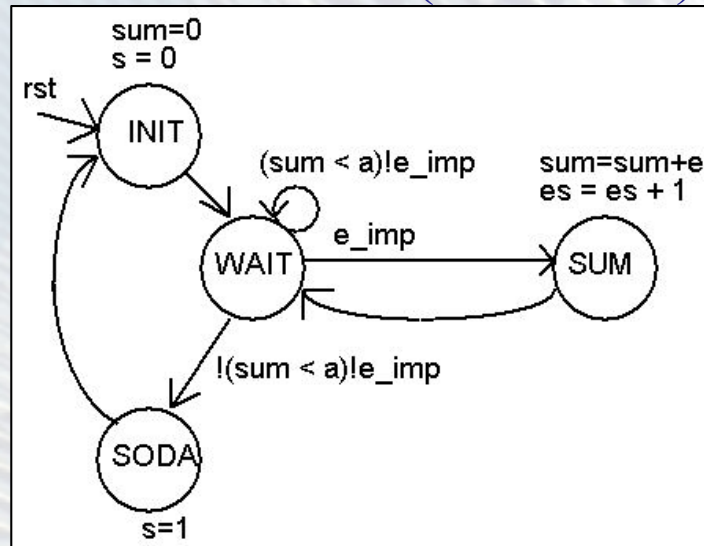
1. Az *érme vizsgálóba* bedobnak egy pénz érmét.
2. Az *érme vizsgáló*, ha érvényes az érme, az *e* kimeneten kiadja az *érme értékét* 8 bites digitális számként és kiad *egy 1 órajel hosszúságú impulzust* az *e_imp* kimeneten.
3. A *szóda automata vezérlő* 8 bites *a* bemenetén be van állítva a szóda ára. Amikor megjön az *e_imp*, hozzáadja az eddig bejött pénzösszeghez a most bedobottat ($sum = sum + e$) és az érme számláló *ec* értékét is növeli ($ec = ec + 1$). Ha az összeg az eléri a szóda árát ($sum \geq a$), akkor az *s* kimeneten kiad egy *1 órajel hosszú impulzust*, a *szóda adagoló* ennek hatására kiadja a poharat és teletölti szódával.



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. $!(\text{sum} < a)$)
 - A kimenet magas szintű művelet (értékadás) is lehet. (Pl: $\text{sum} = \text{sum} + e$)

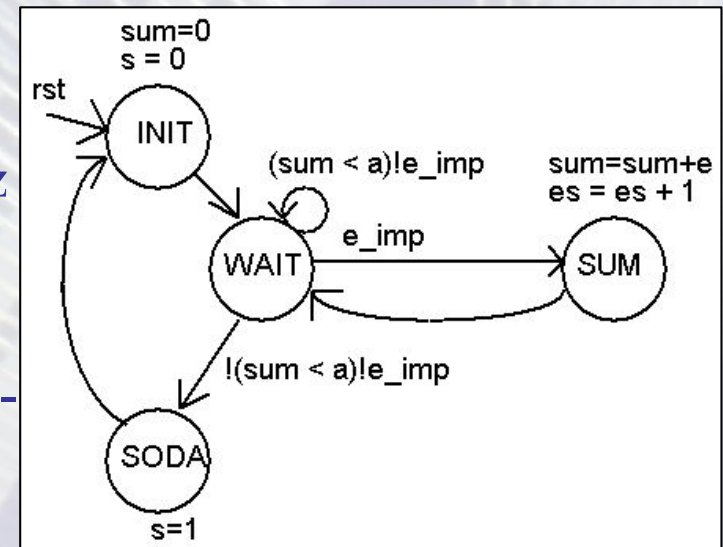


Digitális rendszerek tervezése

HLSM leírás

A szóda vezérlő **HLSM** állapotdigramja

- Az **INIT** állapotban törli a szóda vezérlő s kimenetét és az bedobott pénz összegét tároló **sum** tárolót, törli az **es** érme számlálót, majd átmegy a pénz bedobás váró **WAIT** állapotba.
- A **WAIT** állapotban, ha pénz bedobás történt ($e_imp = 1$), akkor átmegy az számolásokat elvégző **SUM** állapotba. Ha nincs pénz bedobás és az összeg még kisebb, mint az ár, akkor marad **WAIT**-ben. 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, növeli az érme számlálót és 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. Tartalmazza a feladat során *elvégzendő műveleteket*, hogy *mely adatokat kell tárolni* (értékek), az adatok honnan hova mennek (*adat utakat*).
 2. Tartalmazza a feladat során figyelembe veendő *feltételeket*, melyek befolyásolhatják a műveletek elvégzését ill. annak sorrendjét.
 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 a szükséges műveletek (akár egyszerre több) végrehajtására és a figyelendő feltételek előállítására*. Ennek a hardver résznek a neve **ADATSTRUKTÚRA**

Digitális rendszerek tervezése

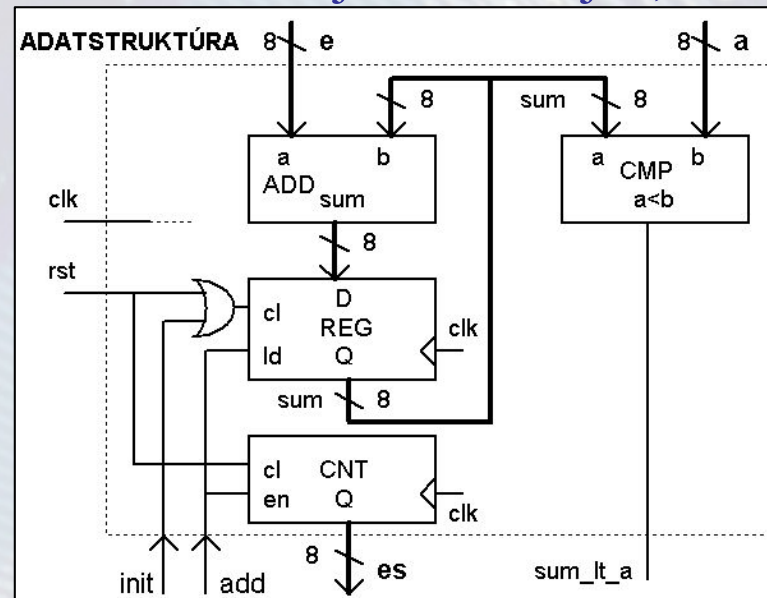
ADATSTRUKTÚRA

- A példa feladatban *elvégezendő műveletek*:
 $sum = 0$, $sum = sum + e$ Kell egy összeadó (előállítja $(sum+e)$ -t. Kell egy törölhető, tölthető *regiszter* (kimenete a sum , beleírjuk $(sum+e)$ -t). Az *adatút*: a sum regiszter bemenetére megy az összeadó kimenete, az összeadó bemeneteire megy a sum regiszter kimenete, és az e adat.
- $es = 0$, $es = es + 1$ kell egy törölhető, engedélyezhető számláló
 $s = 0$, $s = 1$ ez lehet egy *hagyományos FSM kimeneti jel* (állapot alapján kombinációs hálózattal) előállított jel
- A *figyelendő feltételek*:
 e_imp : kívülről jön, hagyományos *FSM feltétel jel*
 $(sum < a)$: nagyság komparátorral állítjuk elő.
 $(sum < a)!e_imp$, $!(sum < a)!e_imp$: A további logikában írjuk le a megfelelő viselkedést, lásd később.

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 összegző 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 logika
41 assign sum_lt_a = (sum < a);
42 endmodule
```

Digitális rendszerek tervezése

VEZÉRLŐ

- Szükség van a műveleteket azt adatstruktúrával elvégzettető 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 jeleket* (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, ennek hatására kezdi el a működését a vezérlő.
- A vezérlőjelek egy része a működési állapotról adhat visszajelzést. Ezeket nevezzük *státus jeleknek*. 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 gráf magas szintű feltétel jeleit helyettesítjük az adatstruktúrából és kívülről jövő jelekből előállított feltétel jelekkel.

A példában:

!(sum < a)!e_imp helyett *!(a_lt_b)&!e_imp*

(sum < a)!e_imp helyett *!a_lt_b)&!e_imp*

- A HLSM gráf magas szintű műveleteit helyettesítjük a műveleteket az adatstruktúrával végrehajttató vezérlő jelekkel.

A példában:

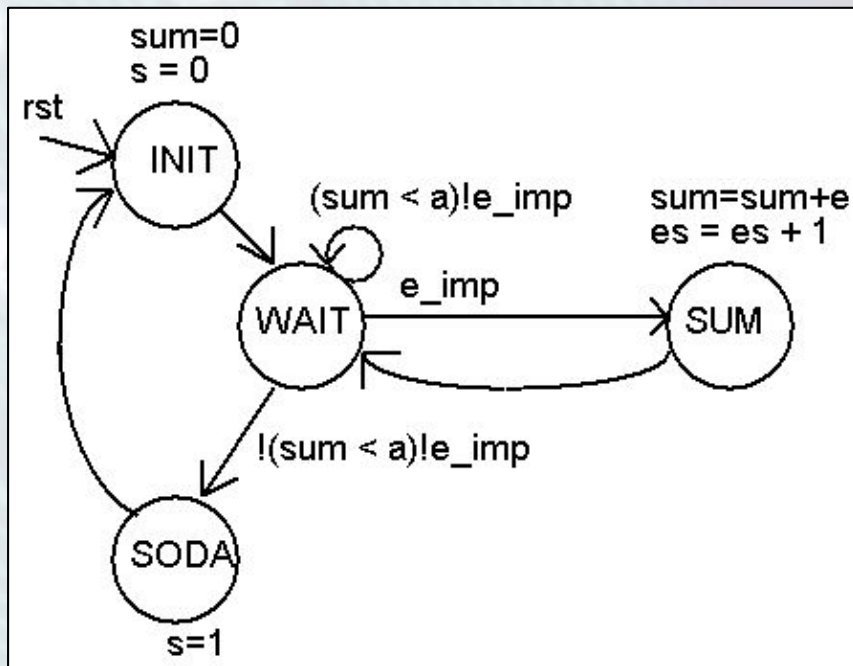
sum = 0 és *es = 0* helyett *init*

sum = sum + e és *es = es+1* helyett *add*.

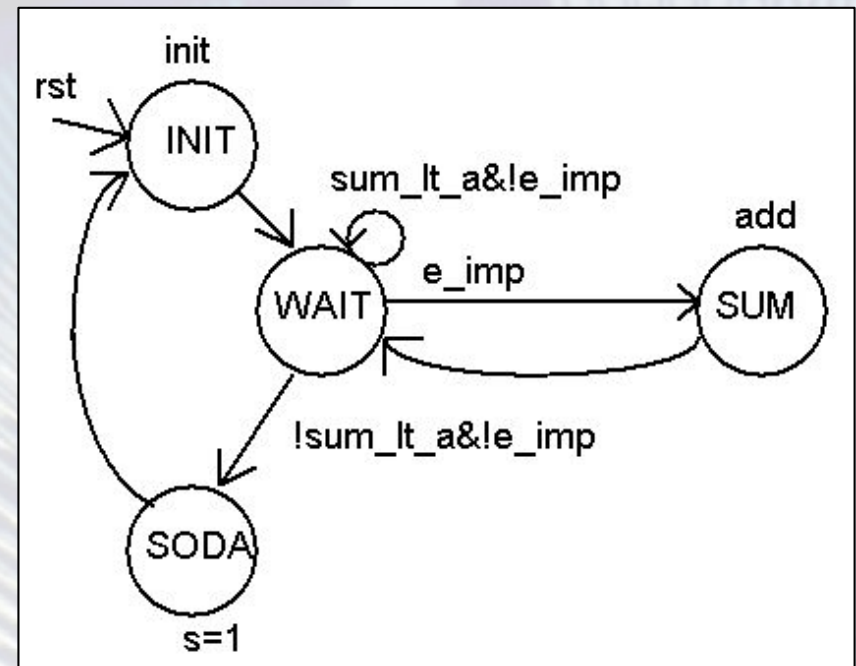
Digitális rendszerek tervezése

VEZÉRLŐ

HLSM állapotgráf
(állapotdiagram)



VEZÉRLŐ FSM állapotgráf
(állapotdiagram)



Digitális rendszerek tervezése

VEZÉRLŐ

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

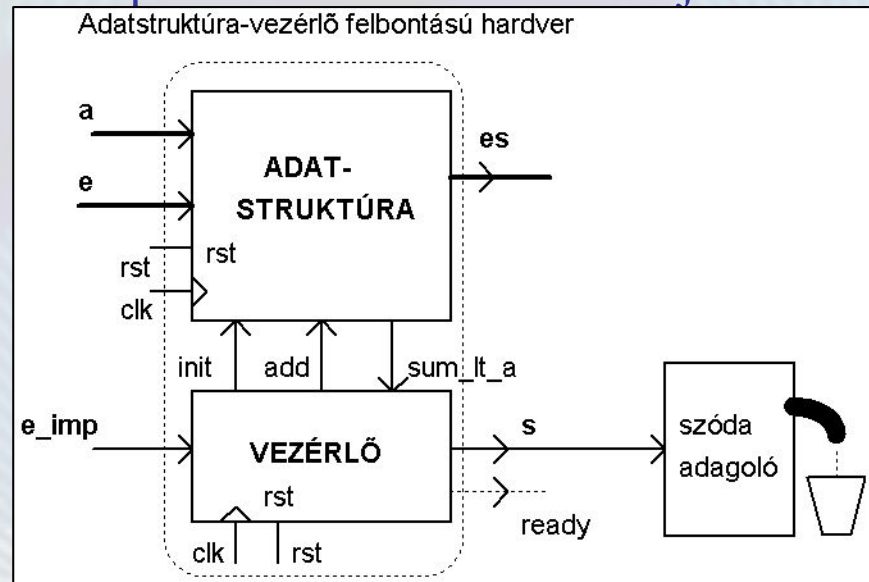
```
22 module vezerlo( input clk, rst, e_imp,
23                 sum_lt_a, output init, add, s);
24
25 reg [1:0] state, next_state;
26 localparam INIT = 2'b00, WAIT = 2'b01,
27             SUM = 2'b10, SODA = 2'b11;
28 //állapotregiszter
29 always@(posedge clk)
30 if(rst) state <= INIT;
31 else    state <= 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

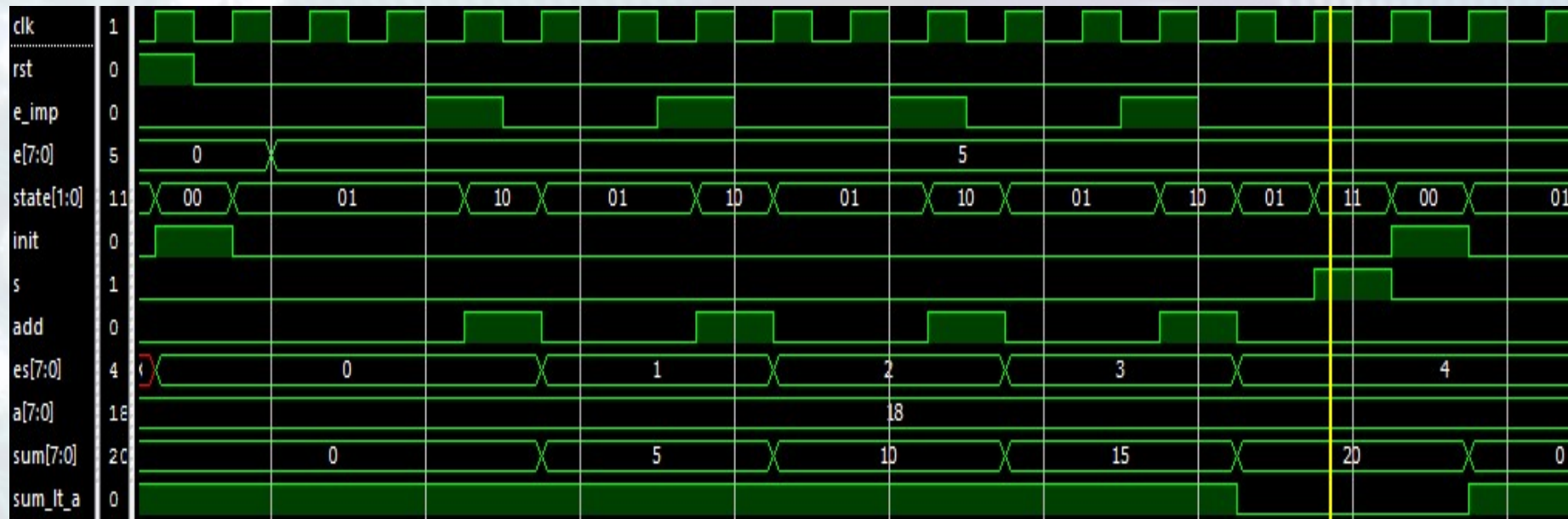


```
21 module soda(input rst, clk, e_imp, input [7:0] a,e,
22               output [7:0] es, output s);
23
24 //parancs jel és feltétel jelek:
25 wire sum_lt_a;
26 //vezérlő jelek:
27 wire init, add;
28
29 adat_strukt soda_adat_strukt(.clk(clk), .rst(rst), .init(init), .add(add),
30                 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

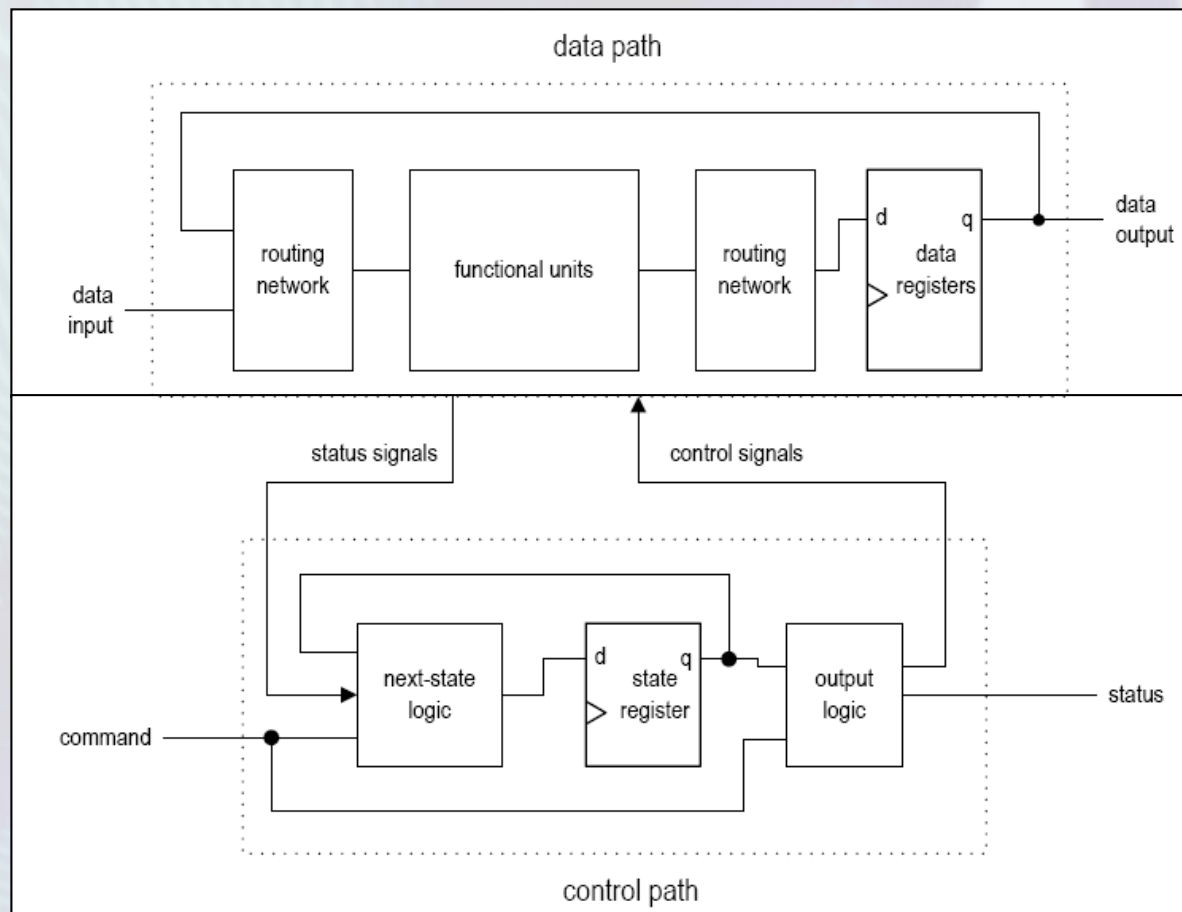
A rendszer szimulációja



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

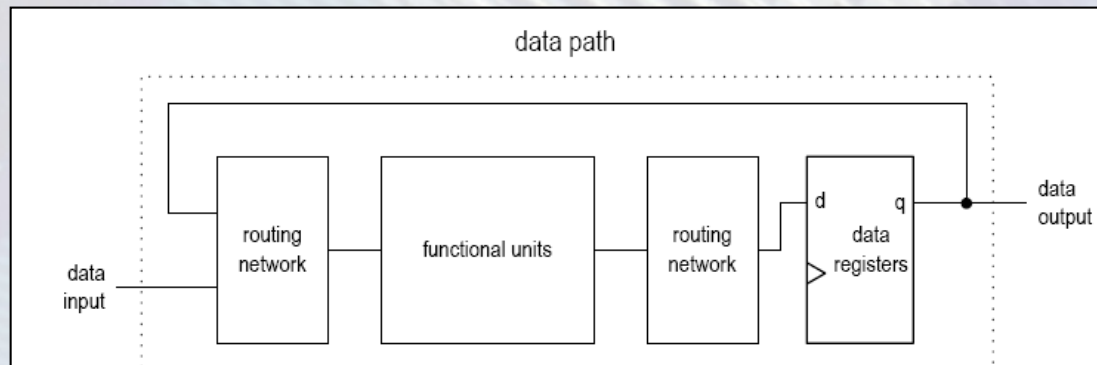
Digitális rendszerek tervezése

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



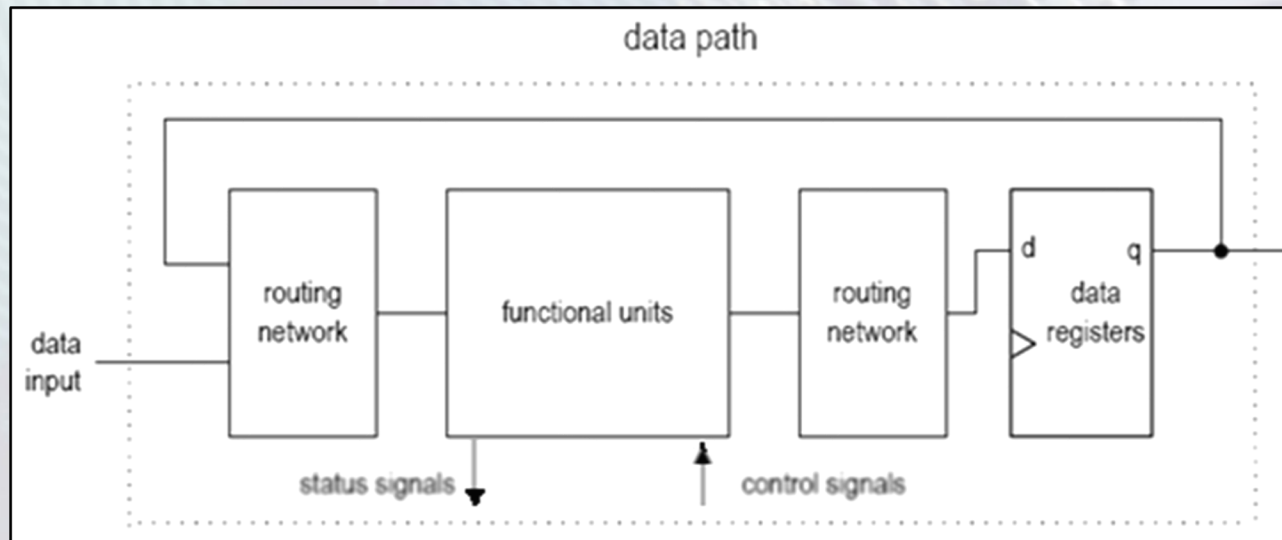
Digitális rendszerek tervezése

- **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

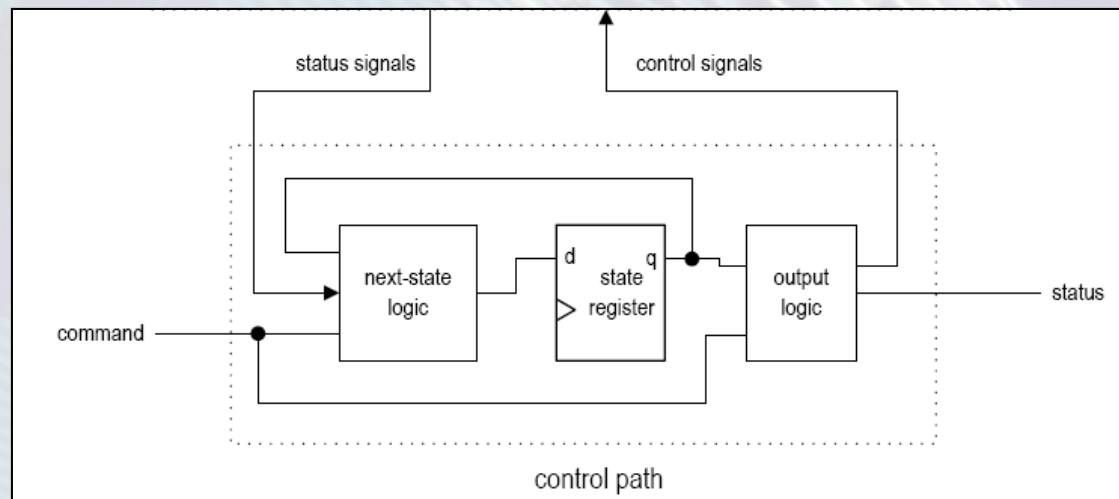
- **Az adatstruktúra, azaz az adatfeldolgozó rész**
 - *Az elsődleges adat bemenet és adat kimenet formátuma alkalmazásfüggő* (bitek, bitvektorok, előjel nélküli számok, 2-es komplementes számok stb.)
 - *Az adatstruktúra státuszjelei a vezérlőbe jutnak, a vezérlőből származó belső vezérlőjelek pedig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



Digitális rendszerek tervezése

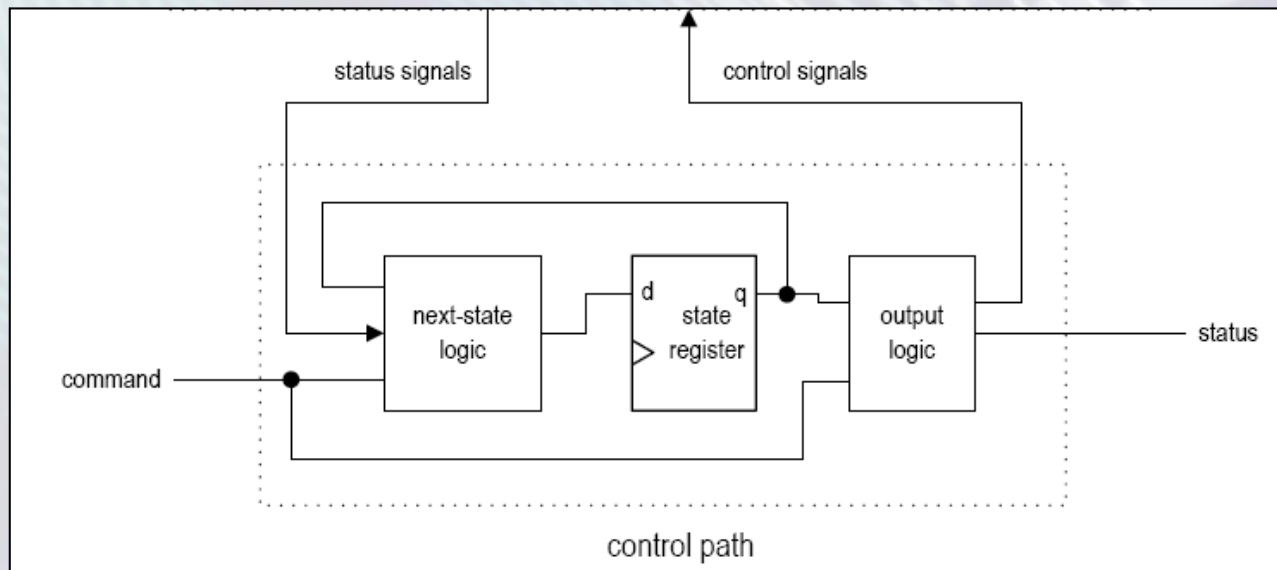
- **A vezérlési alrendszer: FSM**

- 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 (státusz 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*.
 - *Az állapotkódolás a működés szempontjából közömbös*, hatása csak a logikai hálózatrész komplexitására van.

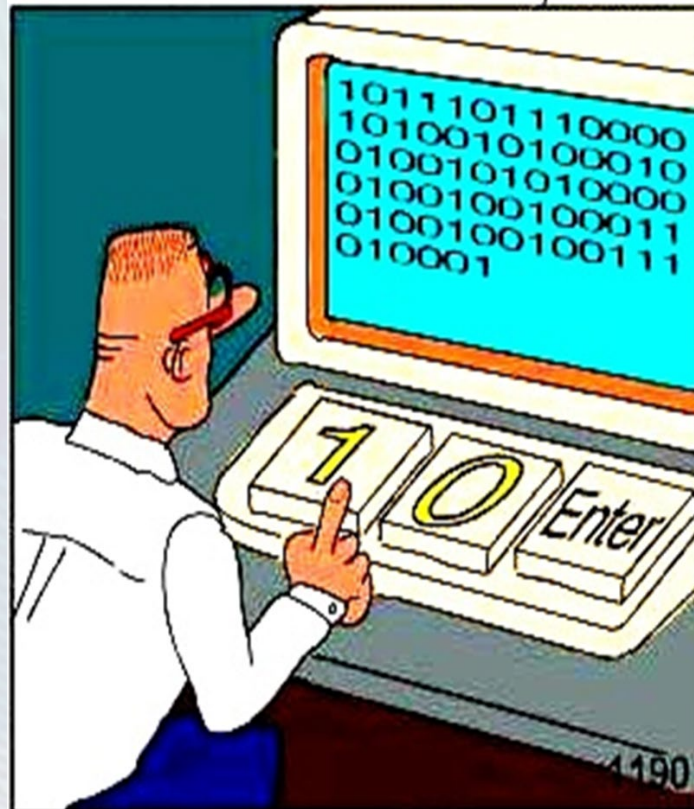


Digitális rendszerek tervezése

- **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.**

Kérdések, észrevételek

- A kérdéseket a benes@mit.bme.hu címre várjuk de csak ... @edu.bme.hu címről, tárgy: DIGIT

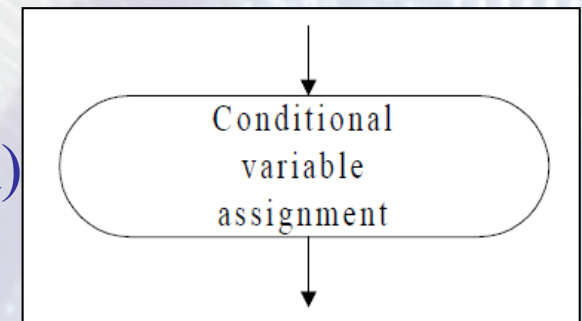
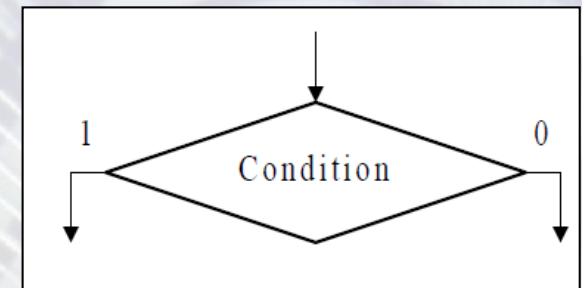
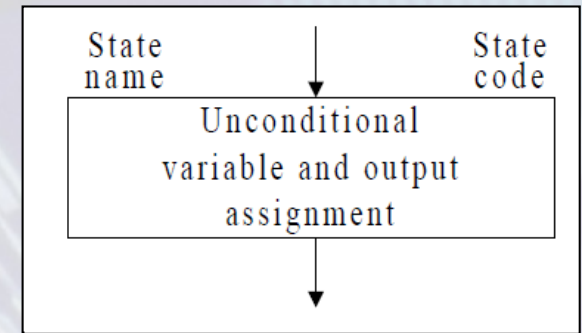


Digitális rendszerek tervezése

- 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 egy (esetleg *kettő*, de nem tetszőleges számú) *következő állapotra* 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.

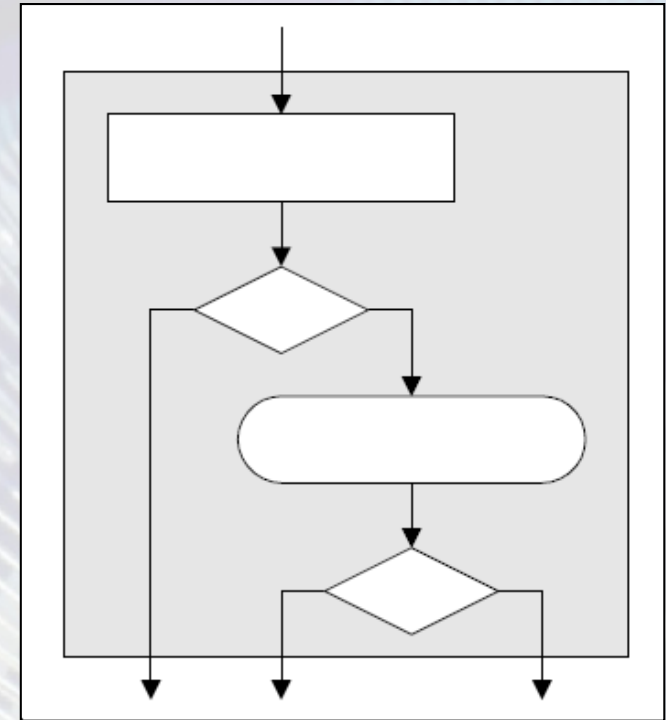
Digitális rendszerek tervezése

- 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)



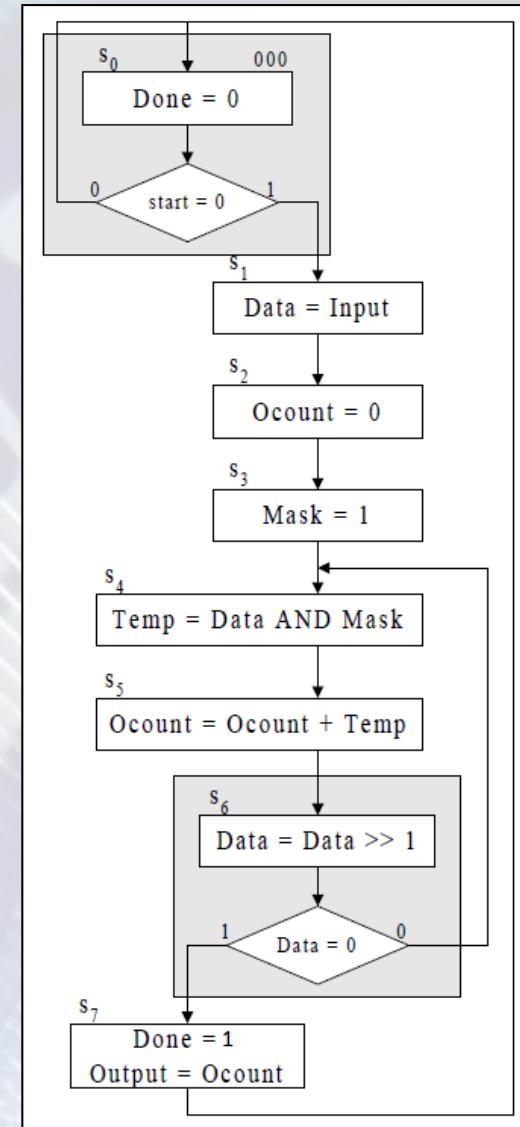
Digitális rendszerek tervezése

- 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 (következik a fenti megjegyzésből)
 - Az ASM blokk tartalmazhat egy vagy több feltételvizsgálatot és feltételes értékadást (de utóbbit mi nem használjuk)
 - A tipikus használati mód nagyon egyszerű ASM blokkokkal dolgozik (csak állapot, vagy áll.+felt.)



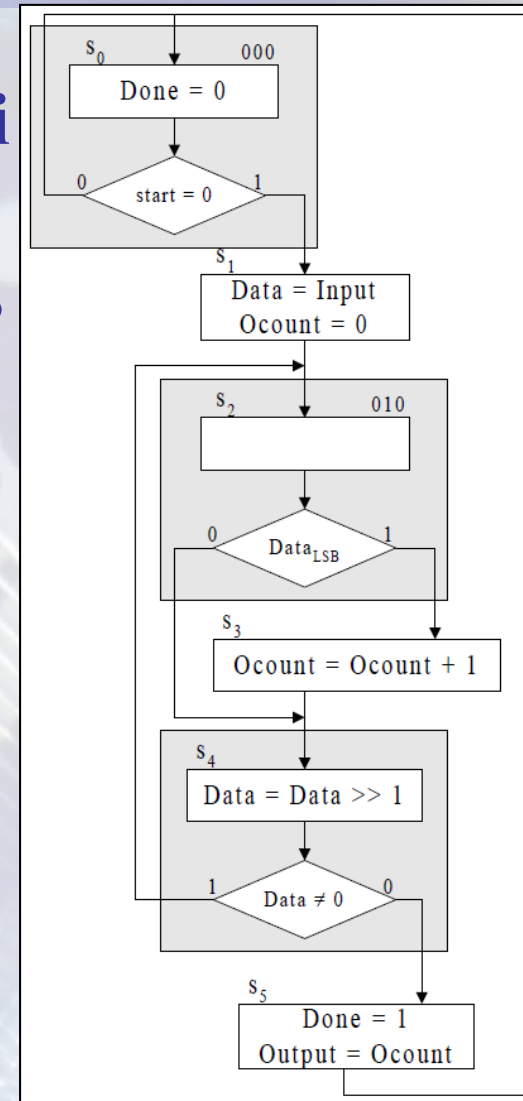
Digitális rendszerek tervezése

- Az ASM algoritmikus állapotvezérlő
- Példa: egy n bites adat aktív bitjeinek megszámlálása (Ones_count/Popcount)
 - 8 állapot, 8 ASM blokk: *ha az állapotok kódja egymást követő számok, akkor az állapot regiszter tölthető számláló lehet*
 - 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
- Egyszerű ASM blokkok
 - Nagyon elemi feladatlebonthatás, 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 szükséges
 - S4 lépteti az adatot és tesztel: ha nem kész, vissza S2-be
 - S5: 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, lépésenkénti *elemi utasításokon alapuló végrehajtás* megtervezését támogatja
- Az elemi ASM blokkokkal építkezve könnyű lesz az áttérés a gépközel SW algoritmusok tervezésének tanulására

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 algoritmikus állapotvezérlő
- A vezérlő ebben az esetben lehet egy egyszerű bináris számlálón alapuló vezérlőegység (nem szükséges a sokkal általánosabb FSM állapotvezérlő felépítés)
- Az állapotátmenetek vezérlése:
 - CONT: $\text{state} \leq \text{state} + 1$; // INC következő áll.
 - JUMP : $\text{state} \leq \text{new_state}$; // LOAD közv. ugrás
 - CJMP : if (COND) then $\text{state} \leq \text{new_state}$ // LOAD
else $\text{state} \leq \text{state} + 1$ // INC
- Ezt a feladatot egy törölhető, tölthető bináris felfelé számláló tudja biztosítani → Nevezhetjük állapotszámlálónak vagy akár utasításszámlálónak (programszámlálónak) is (a jövőbeni processzorhoz)

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

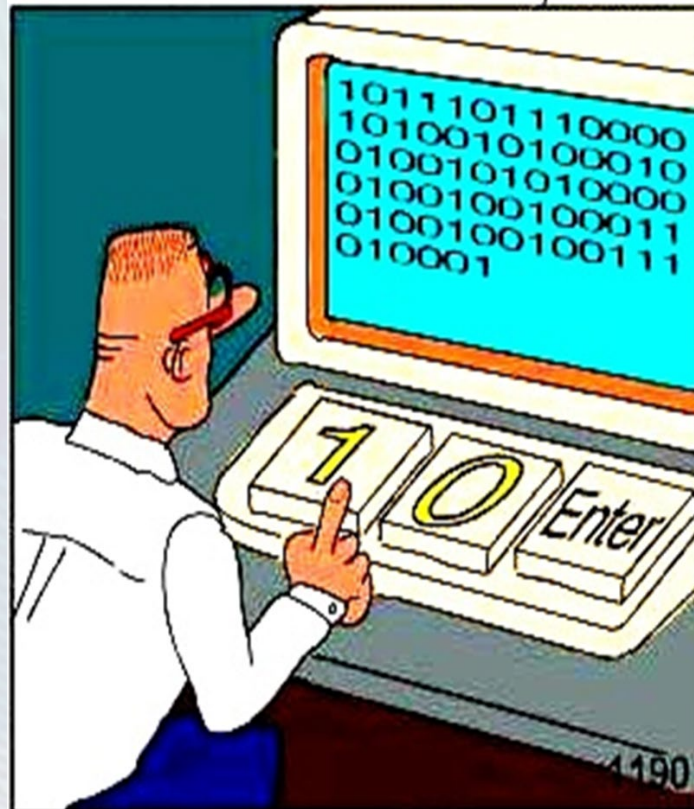
- **Megjegyzés:**
- Egyszerűbb esetekben nem különböztetjük meg explicit módon az adatstruktúrát és vezérlőt
- Az adatstruktúra viselkedési leírásában, a beépített feltételeket, mint vezérlési feltételeket kiértékelve tudjuk a kívánt működési állapotokat elérni
- Ez egy intuitív tervezési módszer. Sokszor nagyon egyszerű, 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)

Szolgálati közlemény

A ZH anyag eddig tart.

Kérdések, észrevételek

- A kérdéseket az rtamas@mit.bme.hu címre várjuk
Tárgy: DIGIT



Digitális technika

7. EA vége