Összetett feladatok megoldása

F1. A laboratóriumi feladat a legnagyobb közös osztó kiszámító algoritmusának realizálása digitális hardver eszközökkel. Az Euklideszi algoritmus alapja a maradékos osztás, azaz az lnko(a,b) a következőképpen számítható:

a = b * q1 + r1	r1 = a % b
b = r1 * q2 + r2	r2 = b % r1
r1 = r2 * q3 + r3	r3 = r1 % r2

ahol $|b|$ > r1 > r2 > r3 >=0 és a % a Verilog moduló osztás művelet eredménye.

Mivel a % operátor nem szintetizálható, ezért egyszerűbb algoritmussal dolgozunk.

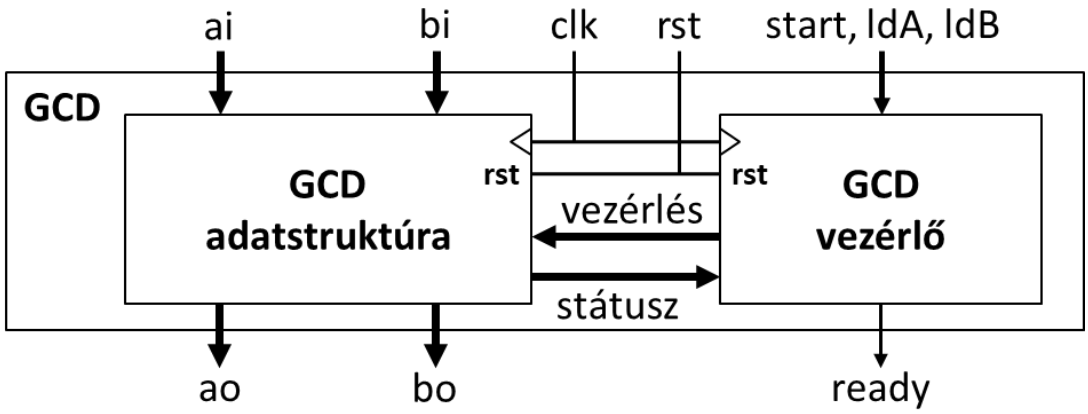
lnko(a,b) = lnko (a-b, b) ha a > b

lnko(a,b) = lnko (a, b-a) ha b > a

A számítási lépéseket addig folytatjuk, amíg a két regiszter tartalma egyenlővé válik, amikor a regiszterek tartalma az lnko értékét adja.

Tervezzük meg az lnko(a,b) számítását végző digitális rendszert!

A digitális rendszert az alábbi általános adatstruktúra – vezérlő szerinti felépítéssel valósítjuk meg:



F1.a Tegyünk javaslatot az lnko(a,b) számítására szolgáló rendszer adatstruktúrájára. Az adatstruktúra 3 féle verzióban, egymástól kissé eltérő felépítéssel, eltérő műveleti elemekkel építhető fel. Röviden tekintsük át a működési elvet mindhárom verzióra, értékeljük a szükséges funkcionális egységeket, a működés lépéseit:

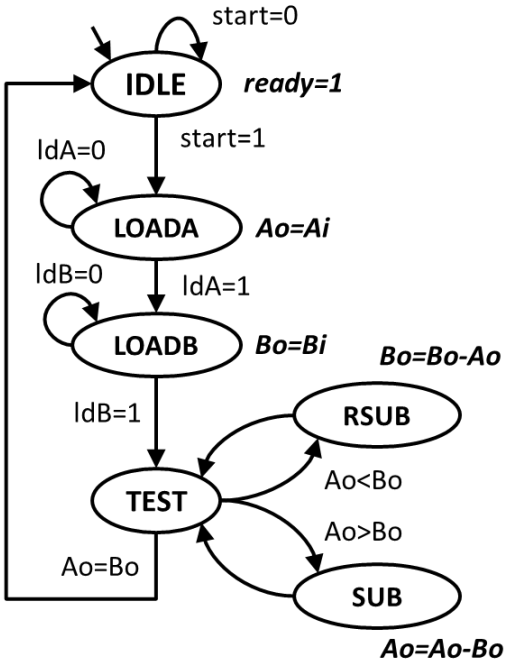
1. Végrehajtás a ↔ b regisztertartalom cserével, egyszerű kivonóval
2. ***Végrehajtás komplex SUB - RSUB műveleti egységgel (a – b ill. b – a egy egységben)***
3. Végrehajtás két kivonó egységgel

A lnko(a,b) számítására az egyszerű algoritmus mellett is van választási lehetőségünk. Biztosan szükségünk van 2 adattároló regiszterre, amibe az elsődleges bemeneti adatokat és a későbbiek során az iterációs részeredményeket tölthetjük.

- **1. verzió:** Ha ***a > b*** nem teljesül, írjuk át egymásba a regiszterek tartalmát, ezután már teljesül ***a > b*** és a kivonás elvégezhető. A műveletvégzés után a különbséget mindig írjuk az ***a*** regiszterbe. Amikor ***a = b***, leállás.
- **2. verzió:** Ha ***a > b***, elvégezzük a kivonást és a különbséget írjuk az ***a*** regiszterbe. Ha ***a < b***, akkor felcseréljük az operandusokat a kivonó bemenetén (azaz egy fordított operandus sorrendű kivonást, ***RSUB*** hajtunk végre), és a különbséget a ***b*** regiszterbe írjuk. Amikor ***a = b***, leállás.
- **3. verzió:** Két azonos felépítésű egységet tervezünk, mindkettő képes minden ütemben kivonást végezni a saját tartalma és a másik regiszter tartalma között. Csak azt az eredményt tároljuk, ahol a nagyobb operandus volt. Amikor ***a = b***, leállás.

F1.b Válasszuk a ***2. verziót*** és ehhez rajzoljuk fel a lnko(a,b) számítására szolgáló rendszer magas szintű állapotvezérlőjét, 8 bites előjel nélküli számábrázolású operandusokat feltételezve. A kimeneti eredmény is legyen 8 bites formátumú. A rendszer számára rendelkezésre áll egy ***clk*** és egy ***rst*** jel a működtetéshez. A számítás indíthatóságát a ***READY*** jel magas szintje jelzi, ezután a működés a ***START*** pulzussal indítható. A számítás alatt a ***READY*** jel inaktív.

A második verziót, tehát az operandusok cseréjéhez a kivonó előtt multiplexereket alkalmazó módszert választva a bemutatáshoz a magasszintű állapotdiagram (HLSM) a következő:



6 állapotot különböztetünk meg: IDLE, LOADA, LOADB, TEST, SUB és RSUB. A LOADA és LOADB előkészítési lépésekben történik meg a bementi adatok betöltése, míg a TEST vizsgálati állapotban a következő állapot kijelölése. A tényleges műveletvégzés a SUB és RSUB állapotokban történik. Előbbiben az A-B eredményt töltjük az A regiszterbe, utóbbiban pedig a felcserélt operandusú B-A eredményt a B regiszterbe. A magasszintű állapotdiagram interfész jelei:

start: a rendszer indítóimpulzusa, bemeneti parancs, bitváltozó

ready: „számítás indítható” jelzés, kimeneti státuszjel, bitváltozó

Ai, Bi: bemeneti adatok, 8 bites pozitív egész bitvektor értékek (0 – 255)

ldA, ldB: a bemeneti adatok betöltő jelei, bemeneti parancs, bitváltozó

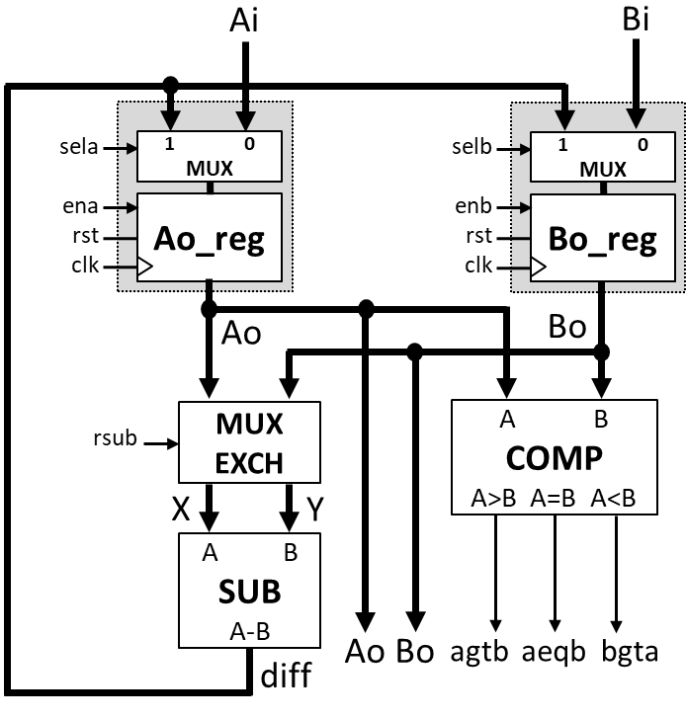
Ao, Bo: a kimeneti lnko érték (egyenlőek, egyik felesleges), 8 bites pozitív egész bitvektor értékek

$$1 \leq (Ao=Bo) \leq \min\{Ai, Bi\}$$

A felrajzolt magasszintű állapotdiagram (HLSM) Moore modell szerinti működést modellez. A bemeneti adatok betöltése azért történik két lépésben, mert a rendszernek az FPGA kártyán ily módon történő megvalósítása egyszerűbb, jobb kezelhetőséget biztosít.

F1.c Rajzoljuk fel részletesen a választott adatstruktúrát, a szükséges a regisztereket, adattárolókat, a műveletvégzőket, az ezeket összekapcsoló adatutakat és az útválasztó multiplexereket!

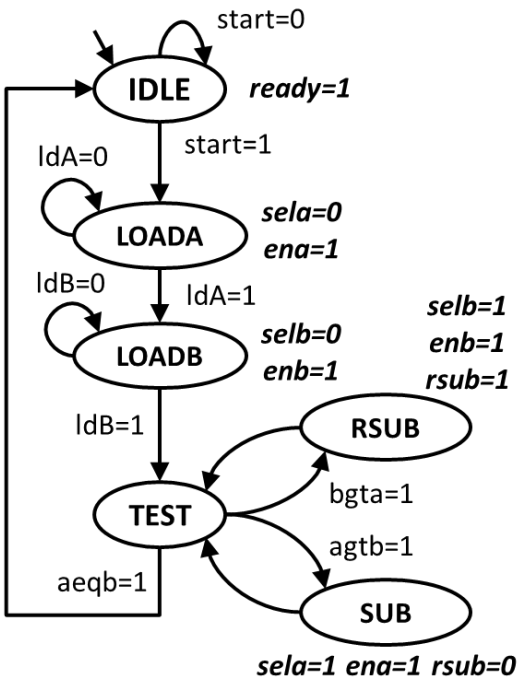
Ehhez a magasszintű működési modellhez a következő részletes adatstruktúra tartozik.



Tartalmaz két 8 bites, engedélyező bemenettel (ezekre az ***ena*** és ***enb*** nevű jelek kapcsolódnak) rendelkező regisztert. A regiszterek bemenetére 1-1 multiplexer kapcsolódik, amelyek a bemeneti adatok (***Ai*** és ***Bi***), illetve a kivonó áramkör ***diff*** kimenete között választanak (kiválasztó jelük rendre ***sela*** és ***selb***). Az iterációs lépések műveleteit egy darab 8 bites kivonó végzi (**SUB**), amely előtt található a két 2/1 multiplexert tartalmazó **MUX EXCH** egység a kivonás operandusainak esetleges felcseréléséhez (***rsub***=1 esetén van csere). A **COMP** komparátor a regiszterek tartalmát hasonlítja össze és biztosítja mindhárom lehetséges feltétel jelet (kettő elegendő lenne, mert a három reláció egymást kölcsönösen kizáró).

F1.d. Tervezzük meg a vezérlőegységet, a korábban megismert FSM tervezési módszer alkalmazásával.

A vezérlőegységnek 6 állapota van. Az FSM már csak az egybites vezérlő és státuszjelekkel dolgozik. Topológiailag ugyanaz, mint a HLSM, de itt már csak a státuszjeleket és a vezérlőjeleket tüntetjük fel.


start=0
IDLE
ready=1
start=1
ldA=0
LOADA
sela=0
ena=1
ldB=0
ldA=1
LOADB
selb=0
enb=1
selb=1
enb=1
rsub=1
ldB=1
RSUB
TEST
bgta=1
agtb=1
aeqb=1
SUB
sela=1 ena=1 rsub=0