Aritmetikai processzor fejlesztése FPGA-n

Dokumentáció

Olasz-Szabó Bence
oszb92@gmail.com
2014-05-13

Tartalomjegyzék

1. A feladat specifikációja

A feladatom aritmetikai processzor fejlesztése FPGA-n.

A megoldásnak bemutathatónak kell lennie Altera DE2 típusú panelen egy egyszerű számológép alkalmazás formájában, ami képes a négy alapművelet elvégzésére tizedestört alakú számokon.

Az operandusok bevitele a panelhez PS/2 csatlakozó segítségével illeszthető billentyűzet segítségével történik. A számítások eredménye a panelen található LCD kijelzőn jelenik meg.

2. Megoldási vázlat

A feladat az alábbi fő részfeladatokra bontható:

- Input beolvasása a billentyűzetről
- Input feldolgozása és a számítás elvégzése
- Eredmény kijelzése

A fenti 3 részfeladat megoldását külön modulok végzik, amik között az összeköttetést a legfelső szintű modul (calc_top) biztosítja.

A feladat egyik nehézsége az, hogy a billentyűzetről érkező adat formátuma és az LCD kijelzőnek küldendő adat formátuma különböznek egymástól és a számításokhoz egyszerűen felhasználható adatformátumtól is. Ezért a megoldás jelentős részében a különböző adatformátumok közötti konverziót kell megoldani.

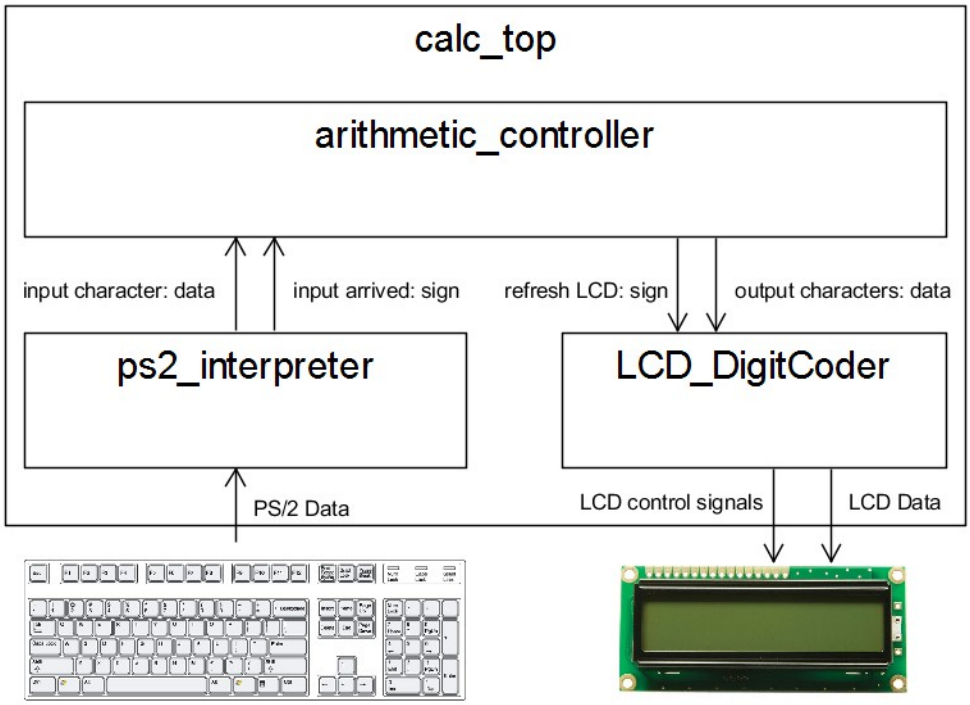
Az input beolvasását végző modul (ps2_interpreter) a billentyűzetről a PS/2 formátumnak megfelelő adatot alakítja át az aritmetikai modul számára könnyebben értelmezhető formára. Ezen kívül már ez a modul kiszűri a feladat szempontjából lényegtelen input karaktereket. Továbbá ez a modul értesíti az aritmetikai modult, ha a feladat szempontjából értékes bemenet érkezett a billentyűzetről.

Az input feldolgozását és az eredmény számítását végző modul az aritmetikai modul (arithmetic_controller). Ez a modul a beérkező értékes karaktereket egy állapotgép alapján feldolgozza, majd a számítások elvégzéséhez optimális formátumra alakítja és vezérli a számítások elvégzését, majd az eredményt átalakítja a kijelző modul számára megfelelő formátumra, és kezdeményezi a kijelzés frissítését.

A kijelzésért felelős modul (LCD_DigitCoder) átalakítja az aritmetikai modultól kapott karaktereket az LCD kijelző adatformátumára, és frissíti a kijelzőt, amikor erre az aritmetikai modultól utasítást kap.

A megoldás alapelve: a felhasználótól a billentyűzeten keresztül érkező input események hatására történik a számítások elvégzése, tisztán eseményvezérelt módon, adatáramlásos információfeldolgozási modell szerint. A megvalósítás tisztán hardveres, nincs beépített mikrovezérlő, a vezérlési feladatokat hardveresen megvalósított állapotgép látja el.

A fő modulokat és kapcsolataikat az alábbi ábra szemlélteti:



3. A megoldás részletezése

3.1. A fejlesztési folyamat

3.1.1. Ismerkedés a fejlesztői környezettel

A feladat megoldását az Altera Quartus II fejlesztői környezet és az Altera DE2 panel megismerésével kezdtem.

3.1.2. Az LCD kijelző vezérlésének elkészítése

A fejlesztői környezet megismerése után az LCD kijelző vezérlését kellett megoldani, hogy a később készített modulok könnyen tesztelhetőek legyenek.

Ehhez a feladatrészhez felhasználtam a fejlesztői panelhez tartozó DVD-n található LCD_Controller[i] és LCD_TEST[ii] modulokat, amik megoldják az LCD kijelző alacsony szintű kezelését.

Az LCD_TEST modult átalakítottam úgy, hogy ne konstans szöveget írjon ki, hanem paraméterül kapja a kiírandó szöveget, ez lett az LCD_Driver modul. Az LCD_Driver modul így bemenő paraméterül a kiírandó szöveg karaktereinek tömbjeit kéri, ahol a karakterek formátuma az LCD kijelző számára értelmezhető, 9 biten tárolt karakterformátum[iii].

3.1.3. Köztes karakterkód létrehozása

Az LCD kijelző számára értelmezhető karakterformátum a feladat szempontjából nem optimális, mivel nem kompatibilis a PS/2 billentyűzetről érkező karakterformátummal, valamint 9 bit feleslegesen sok a feladat szempontjából, mivel csak az alábbi karakterek megjelenítésére van szükség:

- Számjegyek 0-9-ig
- Üres hely, . (pont), =, +, -, *, /, e (hiba jelzésére)

Ez 18 különböző karakter, ami 5 biten kódolható a bővíthetőség szem előtt tartása mellett is.

Ezért létrehoztam egy köztes karakterformátumot, amit a különböző karakter típusok és számformátumok közötti konverzió egyszerűsítésére használok. A formátummal szemben elvárás, hogy a számjegyek könnyen konvertálhatóak legyenek BCD formátumba a számításokhoz, valamint a vezérlő jelek kódjainak feldolgozása is egyszerű legyen. A formátumra a továbbiakban (és a forráskódban) SBCD formátum néven hivatkozok.

Az 5 bites SBCD formátum MSB bitje alapján lehet megkülönböztetni a számjegyek kódjait a vezérlő- és megjelenítést segítő karakterek kódjaitól (a 0 MSB érték jelenti a számjegyet). Számjegy esetén a maradék 4 biten a számjegy BCD kódja található. A műveleti jelek kódjai az utolsó 2 biten különböznek egymástól, így az elvégzendő művelet könnyen dekódolható.

3.1.4. Input beolvasása PS/2 billentyűzetről

A következő feladat a felhasználói bemenet beolvasása és értelmezése volt.

A PS/2 billentyűzet illesztését a PS2_Controller, Altera_UP_PS2_Command_Out és Altera_UP_PS2_Data_In modulok[iv] végzik.

A PS2_Controller a beérkező karaktereket 8 bites „keyboard scan code[v]” formában adja tovább a ps2_interpreter modulnak. A ps2_interpreter modul kiszűri az érkező karakterek közül a feladat szempontjából értékeseket, és ha érkezett ilyen karakter, akkor jelzést küld az arithmetic_controller modulnak, valamint megoldja a konverziót SBCD formátumba. A billentyűzetről érkező speciális vezérlő karakterek (ENTER, DEL, ESC) miatt az SBCD kódot itt ki kellett bővíteni.

A teljes SBCD – LCD – PS/2 kódtábla:

SBCD char	SBCD code	LCD char	LCD code	PS/2 char	PS/2 code
0	00000b	0	100110000b	NUM 0	70h
				0 (english keyboard)	45h
				0 (hungarian keyboard)	0Eh
1	00001b	1	100110001b	NUM 1	69h
				1	16h
2	00010b	2	100110010b	NUM 2	72h
				2	1Eh
3	00011b	3	100110011b	NUM 3	7Ah
				3	26h
4	00100b	4	100110100b	NUM 4	6Bh
				4	25h

SBCD char	SBCD code	LCD char	LCD code	PS/2 char	PS/2 code
5	00101b	5	100110101b	NUM 5	73h
				5	2Eh
6	00110b	6	100110110b	NUM 6	74h
				6	36h
7	00111b	7	100110111b	NUM 7	6Ch
				7	3Dh
8	01000b	8	100111000b	NUM 8	75h
				8	3Eh
9	01001b	9	100111001b	NUM 9	7Dh
				9	46h
01010b-01111b: reserved for hexadecimal characters (not implemented)					
(space)	10000b	space	100100000b	Not needed for input	
. (dot)	10001b	.	100101110b	NUM .	71h
				.	49h
=	10010b	=	100111101b	Not needed for input	
+	10011b	+	100101011b	NUM +	79h
				A	1Ch
-	10100b	-	100101101b	NUM -	7Bh
				S	1Bh
*	10101b	*	100101010b	NUM *	7Ch
				M	3Ah
/	10110b	/	100101111b	NUM /	E0h, 4Ah
				D	23h
(other key / error)	10111b	e	101100101b	Unexpected key	
11000b-11011b: reserved for other special characters					
ok	11100b	Not displayed		NUM ENTER	E0h, 5Ah
				ENTER	5Ah
delete digit (not implemented)	11101b	Not displayed		BACKSPACE	66h
delete number	11110b	Not displayed		DEL	E0h, 71h
reset	11111b	Not displayed		ESC	76h

Megjegyzés: egy SBCD kódhoz több PS/2 kód is tartozhat, ezért lehet például a számjegyeket a NUM billentyűk segítségével is beírni.

3.1.5. Input értelmező állapotgép elkészítése

Az arithmetic_controller modul a ps2_interpreter modultól kap egy 1 órajel hosszú impulzust amikor új értékes karakter érkezett a billentyűzetről (keyCode_arrived jel), valamint megkapja a legutóbbi értékes karakter kódját is SBCD formátumban (keyCodeSBCD). Ezek alapján a vezérlő modulban egy állapotgép végzi a bemenet értelmezését, és a megfelelő események hatására kezdeményezi a műveletek elvégzését.

Állapotváltás a keyCode_arrived események hatására történhet, a keyCodeSBCD kód értelmezésére ekkor kerül sor.

3.1.5.1. Állapotok

A bemenet értelmezése szempontjából az alábbi 6 fő állapot különböztethető meg:

- **0: start:** reset utáni állapot
- **1: op1_int:** az első operandus egészrészéhez tartozó számjegy érkezett
- **2: op1_fract:** az első operandus törtrészéhez tartozó számjegy érkezett
- **3: operation (művelet):** műveleti jel (nem előjel!) érkezett
- **4: op2_int:** a második operandus egészrészéhez tartozó számjegy érkezett
- **5: op2_fract:** a második operandus törtrészéhez tartozó számjegy érkezett
- **6: result (eredmény):** op2_int vagy op2_fract állapotból OK jel (ENTER) hatására ebbe az állapotba kerül az állapotgép.

Megjegyzés: az eredmény állapotból műveleti jel hatására egyből a művelet állapotba lehet átmenni, mivel ha már van eredmény, akkor az bemásolódik az op1-be a megfelelő esemény hatására, ezért az op1-et nem kell újra beolvasni. Ezzel lehetőség adódik több művelet egymás utáni elvégzésére.

3.1.5.2. Események

Az állapotváltás szempontjából a lehetséges eseményeket (azaz az összes különböző értékes karakter érkezését) az alábbiak szerint lehet csoportosítani:

- **digit_arrived** esemény: Számjegy érkezett
- **dot_arrived** esemény: Tizedespont érkezett
- **sign_arrived** esemény: Műveleti jel érkezett
- **delete_arrived** esemény: Törlés jel érkezett
- **ok_arrived** esemény: OK jel (ENTER) érkezett

Megjegyzés: a fenti eseményeket a keyCodeSBCD aktuális értékéből lehet előállítani. Itt ki lehet használni az SBCD formátum előnyeit, például azt, hogy a számjegyek kódja 0-val kezdődik.

3.1.5.3. Az input értelmező állapotgép állapottáblája

	digit_arrived	dot_arrived	sign_arrived	delete_arrived	ok_arrived
0: start	op1_int	op1_fract	-	-	-
1: op1_int	-	op1_fract	operation	start	-
2: op1_fract	-	-	operation	start	-
3: operation	op2_int	op2_fract	-	-	-
4: op2_int	-	op2_fract	-	operation	result
5: op2_fract	-	-	-	operation	result
6: result	-	-	operation	start	-

Megjegyzések:

- Az állapottáblában a könnyebb áttekinthetőség érdekében csak az állapotátmeneteket tüntettem fel, az események hatására elvégzendő egyéb feladatokat nem. A teljes állapotgép megvalósítása az arithmetic_controller.v fájlban található.
- Eredmény állapotból csak akkor lehet más állapotba lépni, ha az aktuális eredmény számításának befejeződése (BCD számjegyek tömbjeként rendelkezésre áll az aktuális eredmény) után következett be az állapotváltó esemény.

3.1.6. Számformátum választása a számításokhoz

A feladat specifikációja szerint az aritmetikai processzornak képesnek kell lennie tizedestört alakú számokon végeznie műveleteket. Tizedestörtek ábrázolására bináris formában a fixpontos és a lebegőpontos számábrázolási módszerek a legelterjedtebbek.

Lebegőpontos számábrázolás esetén az alap (fraction) és a kitevő (exponent) segítségével, a matematikai normálalakhoz hasonló módon lehet a számokat ábrázolni. A módszer előnye a nagy ábrázolható számtartomány, azonban a számtartomány szélső részein végzett műveletek pontossága korlátozott (például ha elég nagy a különbség két szám abszolútértéke között, akkor a számok összeadásakor a kisebbik elhanyagolódhat).

Fixpontos számábrázolás használatával előre meghatározott számú egész- és tört számjegyeket lehet tárolni. A módszer előnye, hogy az ábrázolt számtartományon végzett műveletek pontos eredményt adnak.

A megoldás során a fixpontos számábrázolási módszert választottam, mivel így az elvégzendő műveletek implementálása során kisebb módosításokkal használhatóak az egész számokon végzett műveletek algoritmusai, amik egyszerűbbek a lebegőpontos algoritmusoknál. Egyszerűbb algoritmusok használatával készített harver leírásból általában kisebb komplexitású hardvert lehet szintetizálni, ami gyorsabb végrehajtást vagy kisebb elfoglalt területet jelent az FPGA-n.

Az LCD kijelző összesen 32 karaktert tud megjeleníteni. Ebből 1-1 karaktert elfoglalhat a tizedespont és az előjel, ezért 30 karakter marad a számjegyek ábrázolására. A 30 értékes tízes számrendszerbeli számjegyet kettes számrendszerben $\lfloor \log_2 10^{30} \rfloor + 1 = 100$ biten lehet ábrázolni fixpontos formában. A 30 számjegyből 15 számjegy az egészrész és 15 számjegy a törtrész a tízes számrendszerbeli alakban.

3.1.7. BCD → fixpontos bináris konverzió

Az arithmetic_controller modulban működő állapotgép a billentyűzetről egymás után érkező SBCD karakterekből kiválogatja a számjegyeket, majd az aktuális állapottól függően eltárolja az érkezett számjegyeket a megfelelő operandus egész- vagy törtrészét tároló BCD karakterek tömbjeként.

A műveletek elvégzéséhez a BCD számjegyek tömbjeit át kell alakítani fixpontos operandusokká. Ezt a feladatot a BCDarrayToBinary modul végzi.

3.1.7.1. BCD számjegyek sorrendezése

A BCDarrayToBinary modul először a megfelelő sorrendbe rendezi a számjegyeket. Az egészrész számjegyek beolvasáskor fordított sorrendben kerülnek be az op1_int_digits és op2_int_digits tömbökbe, mivel a felhasználó először az aktuálisan legnagyobb helyiértékű számjegyet írja be, ami azonban nem feltétlenül egyezik meg a tárolható legnagyobb helyiértékkel (azaz a felhasználó nem ír be felesleges 0-kat a szám elejére), ezért ezeket a számjegyeket a megfelelő sorrendbe kell rendezni. A sorrendezéshez szükség van arra az információra, hogy a felhasználó hány darab egész számjegyet írt be, ezért a modul paraméterül megkapja ezt az adatot (int_digitCount paraméter).

Azonban ha az operandust nem a felhasználó írja be, hanem az már rendelkezésre áll (például az előző számítás eredményeként), akkor az egész számjegyek a megfelelő sorrendben rendelkezésre állnak, ezért ilyenkor nem kell sorrendezni azokat. A BCDarrayToBinary modulnak ezt az információt úgy lehet átadni, hogy az int_digitCount paraméter értékeként 0-t állítunk be (0 db egész számjegy esetén teljesen mindegy, hogy megcseréljük-e a számjegyeket, ezért ez a választás nem rontja el a konverziót).

A törtrész számjegyek esetén nem áll fenn a sorrendezési probléma, mivel ilyenkor a felhasználó mindenképpen a tizedes helyiértéken szereplő számjegyet írja be először.

3.1.7.2. Operandusok egész- és törtrészének átalakítása bináris formába

A BCDarrayToBinary modul a sorrendezéssel egy lépésben (ugyanabban az órajelciklusban) átalakítja a BCD számjegyeket tároló 2 dimenziós tömböt 1 dimenziós tömbbé, hogy az átadható legyen a bcd_to_binary[vi] modulnak, ami a tényleges BCD → bináris átalakítást végzi.

A bcd_to_binary modulnak az egyes operandusok egész- és törtrészét egymás után fűzve kell átadni, különben az átalakítás után keletkező külön törtrész számnak nem lehetne egyszerűen meghatározni, hogy pontosan melyik biten van az MSB helyiértéke a kettes számrendszerbeli alakban, emiatt a törtrész-egészrész közötti átvitel nem működne helyesen. Így azonban a kettes számrendszerbe konvertált formában nem tudjuk, hogy pontosan melyik bit a törtrész és az egészrész határa, ezért ezt a műveleteknél nem tudjuk kihasználni.

3.1.8. Fixpontos bináris → BCD konverzió

A műveletek az eredményt fixpontos bináris formában állítják elő. Ezt a kijelzéshez előbb vissza kell alakítani BCD számjegyek tömbjévé, majd a megfelelő helyiértékre kell rendezni a számjegyeket.

A fixpontos bináris → BCD konverziót végző modul (binary_to_bcd[vii]) 200 bites fixpontos bináris számot alakít át 60 számjegyből álló BCD formába (azért kell kétszer annyi bit, mint a BCD → bináris konverzió esetén, mert a szorzás művelet eredményét az operandusok hosszának kétszeresében lehet veszteség nélkül eltárolni).

Az eredmény BCD számjegyeinek megfelelő helyiértékre rendezett, 2 dimenziós tömb formátumú változatát az arithmetic_controller modul állítja elő. A rendezés paraméterei a végrehajtott művelet típusától is függenek.

3.1.9. Fixpontos összeadó- és kivonó modul implementálása

A BCD → fixpontos bináris konverzió elvégzése után az operandusokat bináris egészekként lehet kezelni az összeadás és a kivonás során. Előjeles számokon kell összeadás és kivonás műveleteket végezni, ezért célszerű az előjeles-abszolútértékes formátumot kettes komplemens módba alakítani az összeadás és kivonás műveletek elvégzése előtt, majd a műveletek elvégzése után a kapott eredményt visszaalakítani előjeles-abszolútértékes formába.

A kettes komplemens ↔ előjeles-abszolútértékes formátum közötti konverziók miatt az összeadás és kivonás műveletek végrehajtása 3 órajelciklus alatt történik meg.

A modul forráskódja az addSub.v fájlban található.

3.1.10. Fixpontos szorzás implementálása

A fixpontos szorzás művelet az FPGA-ban található beépített 70db 18 bites hardveres szorzó áramkör segítségével 1 órajelciklus alatt megoldható. A hardveres szorzók összekötéséről a szintézer automatikusan gondoskodik, a verilog kódban elegendő a * operátort használni. A megoldás hátránya, hogy viszonylag nagy területet foglal el az FPGA-ból.

A számformátumot szorzás esetén előjeles-abszolútértékes formában érdemes hagyni, mert így az eredmény előjelének számítása egy egyszerű kizáró vagy művelet segítségével megoldható.

A szorzás művelet eredményét az operandusok hosszának kétszeresében lehet veszteség nélkül eltárolni, ez 200 bitet jelent bináris formában, ami a fixpontos bináris → BCD átalakítás után 60 számjegynek felel meg. A kijelzőn azonban csak 30 értékes számjegyet lehet biztosan megjeleníteni. Ezért a kijelzés módját a panelen található SW17 kapcsoló segítségével át lehet állítani „debug” üzemmódba, ahol az SW0 kapcsoló állásától függően külön-külön lehet az eredmény egész- és törtrészét megnézni. Ez a funkció más műveletek elvégzése után is használható, de a szorzás esetén van igazán nagy jelentősége, mivel így a számítás pontos eredményét is meg lehet tekinteni.

Azonban a megjelenített pontos eredmény 15-15 számjegyen kívül eső részét a műveletek kaszkádosításakor nem lehet továbbvinni a következő művelet operandusaként, mivel az továbbra is csak 30 számjegy pontosságú. Ezért az operandusok ábrázolási tartományán kívül eső számjegyek az eredmény első operandusba másolásakor elvesznek!

A modul forráskódja a mult.v fájlban található.

3.1.11. Fixpontos osztás implementálása

3.1.11.1. Az osztandó megfelelő hosszúságúra konvertálása

A fixpontos osztás művelet implementálható az egészek osztására használható algoritmusok segítségével, azonban figyelni kell a helyiértékek megfelelő kezelésére. Az előző alfejezetben leírtak alapján a szorzás művelet eredményét az operandusok hosszának kétszeresében lehet veszteség nélkül eltárolni. Ez a fixpontos osztásra visszafele alkalmazva azt jelenti, hogy az osztandónak kétszer olyan hosszúnak kell lennie mint az osztónak és az eredménynek ahhoz, hogy az eredmény megjelenítése a megfelelő helyiértéken, veszteség nélkül történhessen (ennek hiányában a fixpontos osztás egészek osztására használható algoritmusok segítségével megvalósított verziója nem veszi figyelembe a törtrészt, helyette tényleges egészosztást végez).

Azonban az osztandó csak 100 biten van tárolva a szükséges 200 helyett. Az osztandót bináris formában már nem szabad változtatni, mivel nem lehet pontosan tudni, hogy melyik bitnél van a törtrész-egészrész határ benne. Ezért az osztandót még a BCD → fixpontos bináris konverzió előtt kell eltolni balra 30 számjeggyel, majd a kapott BCD számot kell átadni a BCD → fixpontos bináris átalakítást végző modulnak.

A BCDarrayToBinary modul kimenete viszont csak 100 bit hosszú. Ezért az osztás művelet osztandójának előállításához szükség van a BCDarrayToBinary200 modulra, ami a BCDarrayToBinary modulhoz hasonló módon működik. A különbség csak annyi, hogy a BCDarrayToBinary200 modul a bcd_to_binary200[viii] modulnak adja át az osztandó BCD alakját, ami az első operandus BCD alakjának és 120 db 0 bitnek az egymás után fűzésével jön létre.

3.1.11.2. Az osztás algoritmusa

A fixpontos bináris osztást az alábbi bináris egészosztást végző algoritmus alapján implementálam:

```
if D == 0 then throw DivisionByZeroException end
Q := 0                //initialize quotient and remainder to zero
R := 0
for i = n-1...0 do    //where n is number of bits in N
  R := R << 1         //left-shift R by 1 bit
  R(0) := N(i)        //set the least-significant bit of R equal to
                      //bit i of the numerator
  if R >= D then
     R = R - D
     Q(i) := 1
  end
end[ix]
```

Az algoritmus változóinak jelentése:

Q: kvóciens, az osztás eredménye

R: egészosztási maradék, fixpontos esetben ez végeredményben az ábrázolási tartományon kívül eső maradék törtszámjegy lesz

N: osztandó

D: osztó

Az algoritmus az általános iskolából ismert osztási módszer kettes számrendszerbeli megfelelője.

Az algoritmus először teszteli, hogy az osztó nulla-e. Ha igen, akkor hibajelzést ad.

A következő lépés az R és a Q változók inicializálása.

Ezután egy cikluson belül 2 fő lépést kell egymás után végrehajtani. Az első lépésben az aktuális maradékot tároló regiszter (R) értékét kell balra shiftelni. Az R regiszter LSB bitjére az osztandó i-edik bitjét kell beírni, ahol az i-edik bit az MSB-től csökkenően halad. A következő lépésben azt kell megvizsgálni, hogy az aktuális maradék nagyobb-e, mint az osztó. Ha igen, akkor a maradék értékét az osztó értékével kell csökkenteni, és az eredmény i-edik bitjére 1-et kell írni. Ha nem, akkor az eredmény i-edik bitje marad 0.

3.1.11.3. Az osztás algoritmusának implementálása Verilog nyelven

Verilog nyelven a for ciklus szintézis időben értékelődik ki, ezért nem használható futási idejű sorrendiség leírására. Ezért a fenti algoritmusban szereplő i ciklusváltozót egy lefelé számláló (counter változó) segítségével valósítom meg.

A ciklus kilépési feltétele az, ha elértük az inputhosszt. Ez akkor történik meg, ha a counter ciklusváltozó túlcsordult, azaz az értéke 255 lesz. Mivel az inputhossz 200 bit (az osztandó hossza), ezért a counter kezdőértéke 199, így a 255 érték használható a ciklus kilépési feltételeként. Ha ez az esemény bekövetkezik, a ready flag 1 értékre állítódik be és az osztást közvetlenül vezérlő állapotgép olyan állapotba kerül, ahonnan csak reset vagy újabb osztási művelet kezdése parancs hatására léphet ki, így befejeződik az algoritmus futása.

A cikluson belül 2 fő lépést kell egymás után végrehajtani, ez egy 2 állapotú állapotgép segítségével oldható meg, aminek az aktuális állapotát a divState regiszter tárolja. divState==0 esetén az első lépés kerül végrehajtásra, továbbá a divState értékének 1-be állítása is megtörténik. Ennek hatására a következő órajelre a 2. lépés kerül végrehajtásra, továbbá a divState értéke újra nullázódik, a ciklusszámláló (counter) értéke pedig csökken 1-el.

Az eredmény előjelét a szorzásnál ismertetett módon, kizáró vagy művelet segítségével lehet előállítani az osztás esetén is.

A fixpontos osztó modul Verilog nyelvű forráskódja a div.v fájlban található.

3.1.11.4. Az osztási algoritmus értékelése

Az algoritmus sorrendi jellege miatt az osztás végrehajtási idejének nagyságrendje az inputhosszal lineárisan arányos. Mivel a cikluson belül 2 lépést kell egymás után végrehajtani, az inputhossz pedig az osztandó miatt 200 bit, ezért az osztás végrehajtása 200*2+1(inicializációs lépés)=401 órajelciklus alatt történik meg. Ez az érték 2 nagyságrenddel meghaladja a többi művelet végrehajtási idejét, ezért az osztást érdemes lehet hatékonyabb algoritmus segítségével implementálni. Ugyanakkor a jelenlegi megoldás az FPGA-ból felhasznált terület szempontjából jónak tekinthető, továbbá a 401 órajelciklus végrehajtása a panelen található 50MHz frekvenciájú órajel mellett 8,02us-ig tart, ami a jelenlegi alkalmazásban (számológép) nem okoz észrevehető lassulást.

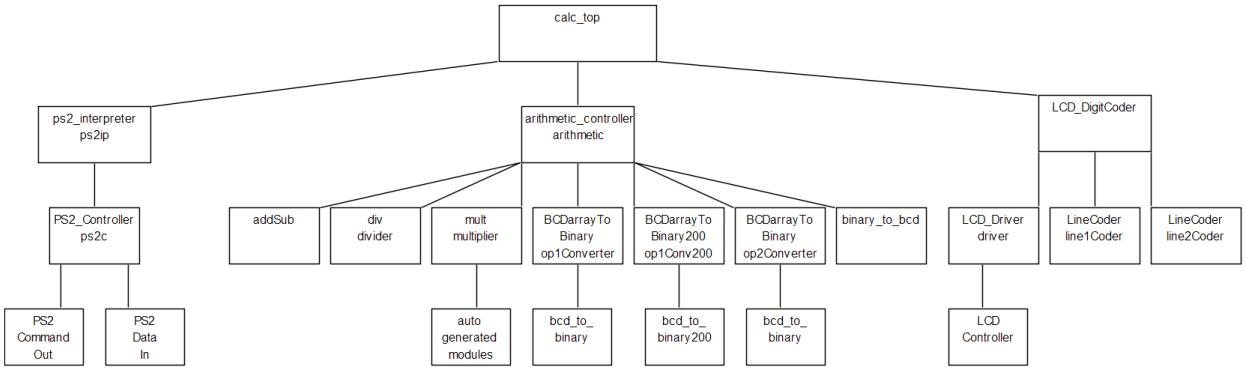
3.1.12. Műveletek kaszkádosítása

A műveletek kaszkádosítása alatt azt értem, hogy az előző művelet eredményét a következő művelet operandusként felhasználja. Ehhez az eredményt az első operandusba kell másolni, amikor az eredmény állapotban új műveleti jel érkezik, aminek hatására a következő órajelre a művelet állapotba kerül az állapotgép (azaz az első operandust már nem olvassa be újra).

Ebben az esetben az első operandust nem a felhasználó írja be, tehát az egész számjegyek a megfelelő sorrendben rendelkezésre állnak, ezért ilyenkor nem kell sorrendezni azokat, ezért a BCDarrayToBinary modulnak az int_digitCount paraméter értékeként 0-t kell beállítani.

3.2. Architektúra

3.2.1. Modulok hierarchiája



Megjegyzés: az ábrán egyes modulnevek rövidítve szerepelnek.

3.2.2. Modulok funkciói

calc_top: Legfelső szintű modul

- Kezeli a be- és kimeneteket
- Összeköti az almodulokat
- Értéket ad a reset jelnek

ps2_interpreter: ps2ip

- Kiválogatja a PS2_Controller által küldött karakterek közül az értékeseket

PS2_Controller: ps2c

- A PS/2 kommunikációt vezérlő modul

Altera_UP_PS2_Command_Out: PS2_Command_Out

- Parancsokat küld a PS/2 billentyűzetnek

Altera_UP_PS2_Data_In: PS2_Data_in

- Fogadja a billentyűzetről érkező adatot

arithmetic_controller: arithmetic

- Feldolgozza az inputot
- Kezdeményezi a műveletek végrehajtását
- Előállítja a kijelzendő „szöveget”

addSub: adderSubtractor

- Elvégzi az összeadást és a kivonást

div: divider

- Elvégzi az osztást

mult: multiplier

- Elvégzi a szorzást

auto-generated multiplier modules: A fejlesztői környezet által generált modulok.

- Összekötik és vezérlik a beépített szorzó áramköröket

BCDarrayToBinary: op1ConverterToBinary

- Megfelelő sorrendbe rendezi az első operandus BCD számjegyeit
- Átalakítja a BCD számjegyeket tároló 2 dimenziós tömböt 1 dimenziós tömbbé

bcd_to_binary: conv

- Bináris formába konvertálja az első operandust

BCDarrayToBinary200: op1ConverterToBinary200

- Megfelelő sorrendbe rendezi az első operandus BCD számjegyeit
- Átalakítja a BCD számjegyeket tároló 2 dimenziós tömböt 1 dimenziós tömbbé
- Hozzáfűz a BCD számjegyeket tároló 1 dimenziós tömb végére 120db nullát

bcd_to_binary200: conv

- Bináris formába konvertálja az első operandust
- 200 bit hosszú eredményt ad

BCDarrayToBinary: op2ConverterToBinary

- Megfelelő sorrendbe rendezi a második operandus BCD számjegyeit
- Átalakítja a BCD számjegyeket tároló 2 dimenziós tömböt 1 dimenziós tömbbé

bcd_to_binary: conv

- Bináris formába konvertálja a második operandust

binary_to_bcd: res_intFract_conv

- Az eredményt bináris formából BCD formába alakítja

LCD_DigitCoder: lcd

- Előállítja az LCD kijelző számára értelmezhető karakterkódokat

LCD_Driver: driver

- A kijelzendő szöveget karakterenként átadja az LCD_Controller modulnak

LCD_Controller: controller

- Vezérli az LCD kijelzőt

LineCoder: line1Coder

- A kijelzendő szöveg első sorát kódolja át SBCD formátumból az LCD kijelző számára értelmezhető formátumra

LineCoder: line2Coder

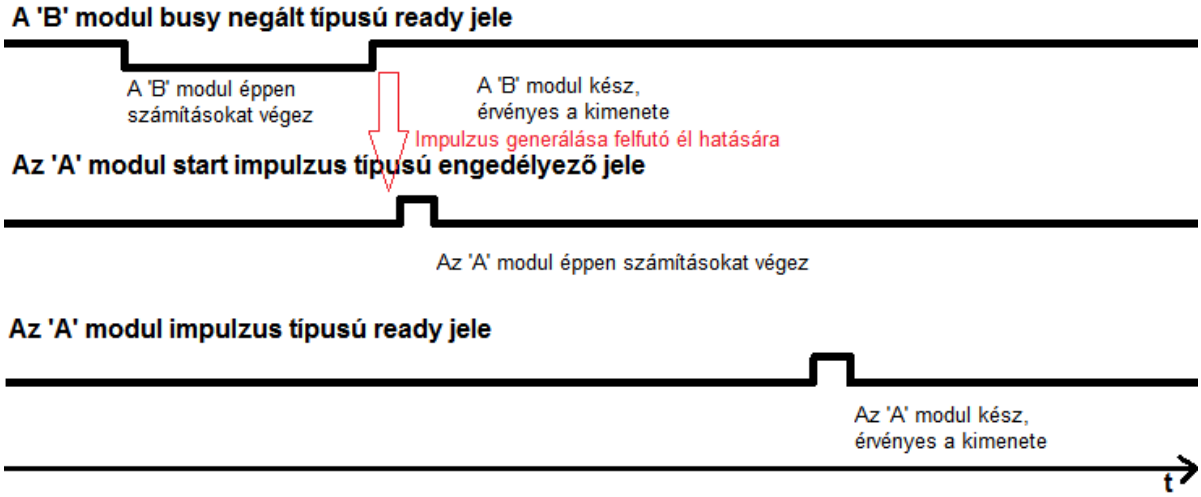
- A kijelzendő szöveg második sorát kódolja át SBCD formátumból az LCD kijelző számára értelmezhető formátumra

3.2.3. A sorrendiség biztosítása a feldolgozás lépéseihez

A feldolgozás egyes lépéseinek sorrendiségét engedélyező jelek segítségével lehet megvalósítani.

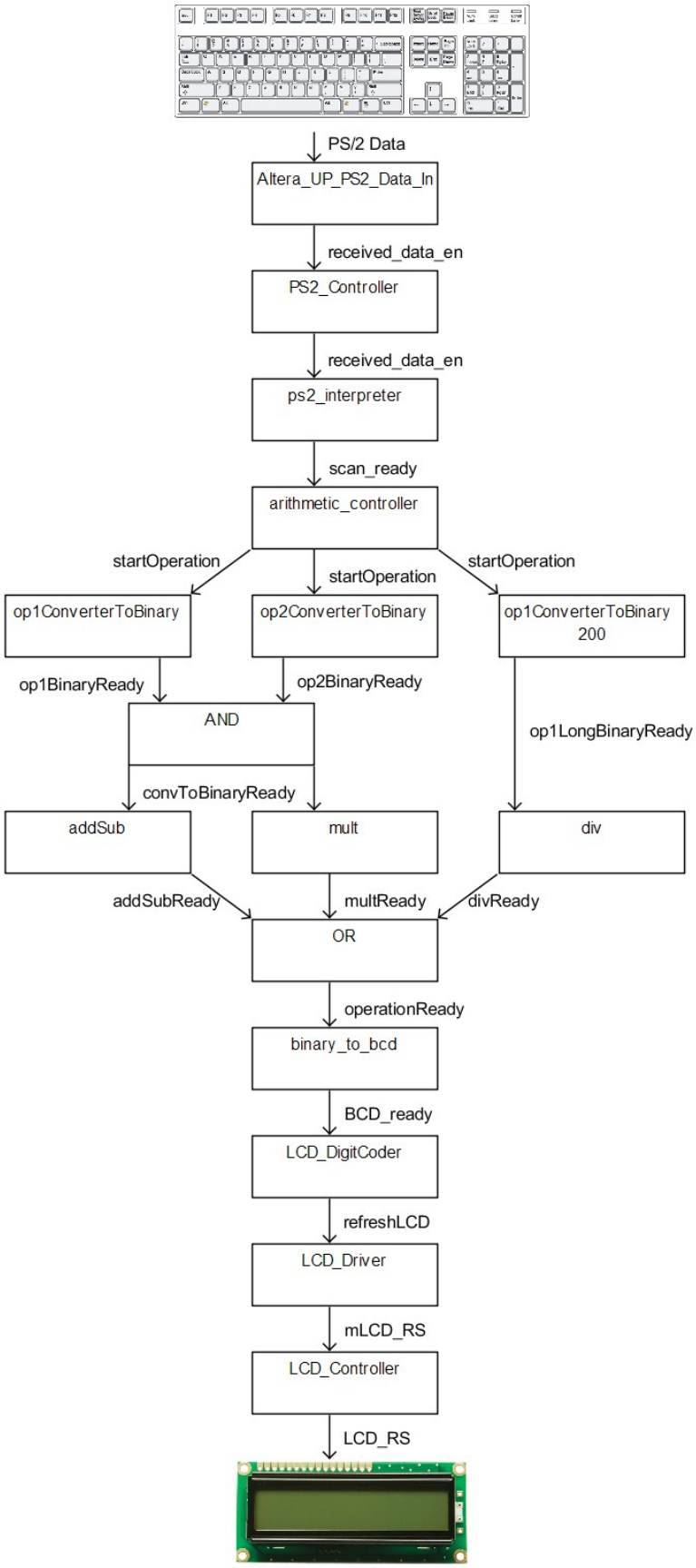
Ha az 'A' modulnak a 'B' modul által előállított eredményt kell feldolgoznia, akkor a 'B' modul „feldolgozás kész” (ready) jelét lehet az 'A' modul engedélyező jeleként használni. Ezzel a módszerrel egymás után lehet fűzni az egyes részfeladatokat ellátó modulokat. Ez azért fontos, mert az egyes részfeladatok feldolgozása több órajelciklust is igénybe vehet, így engedélyező jelek hiányában előfordulhatna, hogy az 'A' modul egy félkész részeredményt dolgoz fel.

Az interfészek egyezésére azonban minden esetben figyelni kell: a ready jel például lehet egy 1 órajel hosszú impulzus a művelet elvégzése után, vagy lehet egy olyan jel, ami csak akkor nem aktív, ha éppen folyik a számítás (busy negált jel). A ready jelhez hasonlóan az engedélyező jel is többféle lehet: start jellegű engedélyező jel, ami impulzus hatására elkezdi a számítást, vagy folyamatosan aktív engedélyező jel, amit a számítás befejezéséig high szinten kell tartani. Busy negált típusú ready jelből start jellegű engedélyező impulzust felfutó él detektálásával lehet előállítani.



A megoldásban a felhasználói input érkezése hatására kell a feldolgozást elkezdeni, a folyamat eredményeként a kijelzőt kell frissíteni. Ezek között a lépések között a részfeladatokat egymás után kell elvégezni. Az egyes részfeladatokat végző modulok az előző részfeladatért felelős modultól kapják a feldolgozandó adatot és az engedélyező jelet. Az adatot a saját funkciójuk szerint átalakítják, majd mikor végeztek, jelzik ezt a következő részfeladatot végző modulnak.

A modulok közötti engedélyező jeleket az alábbi ábra szemlélteti:



Megjegyzés: a könnyebb áttekinthetőség érdekében az ábra nem tartalmazza minden modulnak az összes engedélyező feltételét.

i Altera DE2 fejlesztői panelhez tartozó DVD: DE2Systems/DE2_demonstrations/DE2_Default/LCD_Controller.v
ii Altera DE2 fejlesztői panelhez tartozó DVD: DE2Systems/DE2_demonstrations/DE2_Default/LCD_TEST.v
iii Altera DE2 fejlesztői panelhez tartozó DVD: DE2Systems/Datasheets/LCD/CFAH1602BTMCJP.pdf 13. oldal
iv http://www.eecg.toronto.edu/~jayar/ece241_08F/AudioVideoCores/ps2/ps2.html
v http://www.computer-engineering.org/ps2keyboard/scancodes2.html
vi Copyright (C) 2002 John Clayton and OPENCORES.ORG
vii Copyright (C) 2002 John Clayton and OPENCORES.ORG
viii Copyright (C) 2002 John Clayton and OPENCORES.ORG
ix http://en.wikipedia.org/wiki/Division_algorithm