SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

Újrafonkigurálható digitális áramkörök

2023
Projekt dokumentáció

Hallgató neve:
Molnár Orsolya-Izabella
Szak:
Számítástechnika IV.

Felügyelő tanár:
Brassai Sándor Tihamér

Ultrahangos távolságmérés FPGA alapú rendszerekkel

1. Bevezető

Az ultrahangos távolságmérés egy olyan technológia, amely ultrahangos hullámokat használ a távolság meghatározására egy tárgy és a mérőeszköz között. Az elv alapja az ultrahangos hullámok terjedési sebességének ismeretében történik. A távolságmérő készülékek egy ultrahangos jelzőjelet küldenek ki a tárgy felé, majd a jel visszaverődik a tárgytól, és a visszavert jel érkezési idejének mérésével határozzák meg a távolságot.

A folyamat lépései általában a következők:

1. Jel kibocsátása: Az ultrahangos távolságmérő kibocsát egy ultrahangos jelzőjelet, például ultrahangos hanghullámokat.
2. Jel visszaverődése: A kibocsátott jel eléri a tárgyat, és visszaverődik róla.
3. Érkezési idő mérése: Az érzékelő észleli a visszavert jelet, és meghatározza az érkezési idejét.
4. Távolság kiszámítása: A távolság kiszámítása az érkezési idő, az ultrahang terjedési sebessége és más paraméterek ismeretében történik.

Az ultrahang egy hanghullám 20 kHz feletti frekvenciatartományban, amit az emberi fül már nem érzékel. Az ultrahang anyagban való terjedésének alap feltétele az anyag rugalmassága. Képes bármilyen anyagot érzékelni, kivéve a habokat. Az ultrahangos szenzorok érzékenyek a nyomás, sűrűség és hőmérséklet változásra. Az ultrahangos távolságméréssel általában 40-180 kHz közötti hanghullámot sugároznak, és egy vevővel pedig érzékelik a visszaverődött hanghullámokat. Az ultrahangos távolságmérők egy rövid ultrahang hullám csomagot kibocsátó hangszóróból (adó) és a céltárgyról visszaverődött ultrahang érzékelésére szolgáló mikrofonból (vevő) állnak.

Az ultrahangos távolságmérés széles körben alkalmazható ipari és egyéb alkalmazásokban, például robotika, autóparkoló érzékelők, alagútmonitoring, és orvosi eszközök területén. Az előnye közé tartozik a jó pontosság, a nagy távolságok mérési lehetősége, és a mérési folyamatok jó szabályozhatósága.

Azért előnyös az ultrahangos távolságmérés, mert a magasabb frekvenciájú hangok kevesebb energiát disszipálnak, ezért az ultrahangok koncentráltabb, szűkebb nyalábban terjednek, mint a hagyományos hanghullámok. Ez a tulajdonság segít a szenzornak abban, hogy csak az előtte lévő viszonylag szűk szögtartományban elhelyezkedő tárgyakat észlelje. A távolság meghatározása nem túl bonyolult feladat, hiszen mint tudjuk a kibocsátás és a visszaolvasás között eltelt idő arányos a távolsággal. Ismerve azt, hogy a hang terjedési sebessége légüres térben 340 m/s, és ismerve a kibocsátás és a visszaolvasás közötti időt, egy egyszerű összefüggés segítségével kiszámíthatjuk a távolságot:

$$d = t/2 * 340$$

Ahol,

d - távolság

t - a kibocsátás és a visszaolvasás között eltelt idő.

2. Alkatrész specifikációk

A Nexys A7-100T egy FPGA (Field-Programmable Gate Array) fejlesztőkártya, amelyet a Digilent tervezett és gyártott.

A Nexys A7-100T kártya egy Xilinx Artix-7 sorozatú FPGA chipet (FPGA: xc7a100t) tartalmaz. Az Artix-7 FPGA-k sokoldalúak és széles körben használhatóak digitális tervezési projektekben.

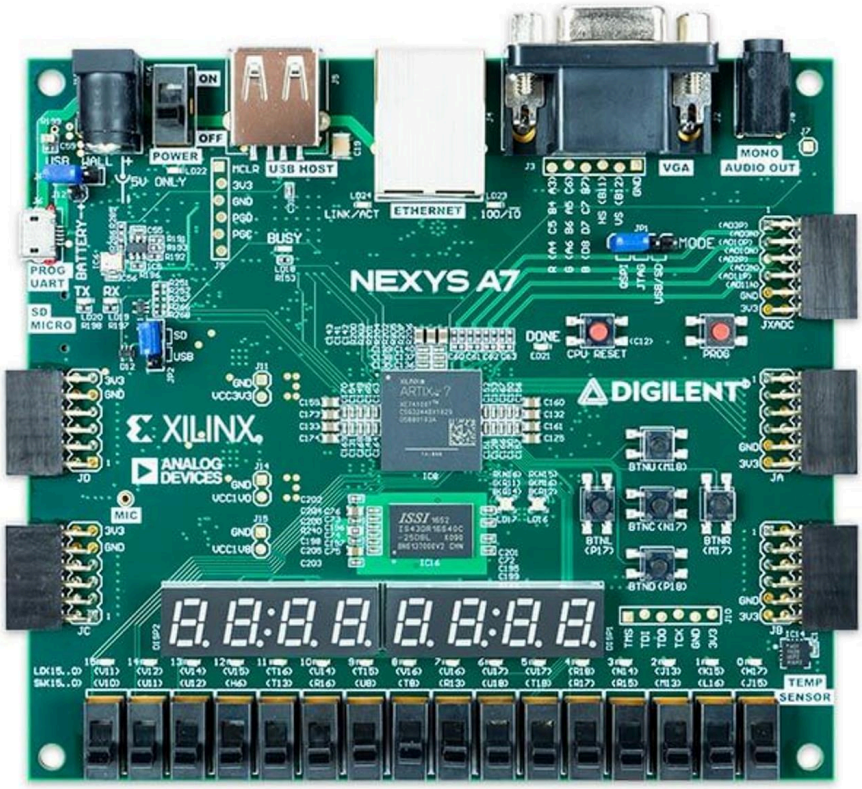
A kártya számos be- és kimeneti lehetőséget kínál, beleértve a GPIO-t, LED-eket, kapcsolókat, illetve számos kommunikációs portot, például USB-t és Ethernetet.

A Nexys A7-100T rendelkezik belső és külső memóriával is, amelyeket a tervezési projektekhez lehet felhasználni.

A kártya tartalmaz egy VGA csatlakozót és egy 3,5 mm-es audio kimenetet, ami lehetővé teszi a videó- és audio interfészek létrehozását.

A kártya rendelkezik egy USB-JTAG interfésszel a programozáshoz és konfiguráláshoz.

A kártyát a Xilinx Vivado tervezőszoftverrel lehet konfigurálni és programozni.



Nexys A7-100T fejlesztői lap

3. Követelmények

a. Funkcionális követelmények:

- A rendszernek képesnek kell lennie ultrahangos jelek küldésére és fogadására.
- Távolságmérés funkció: A projektnek pontosan kell mérnie a távolságot az ultrahangos jelek alapján.
- Egy mérést mennyi idő alatt végzi el.

b. Rendszerkövetelmények:

- A rendszernek ultrahangos szenzorral kell rendelkeznie.
- Szükség van erősítőre
- Nexys A7-100T board

4. Specifikációk

- 40 kHz adó-vevő modulok
- 100 MHz órajel
- VHDL hardware leíró nyelv
- Vivado 2021.2 (minimum)

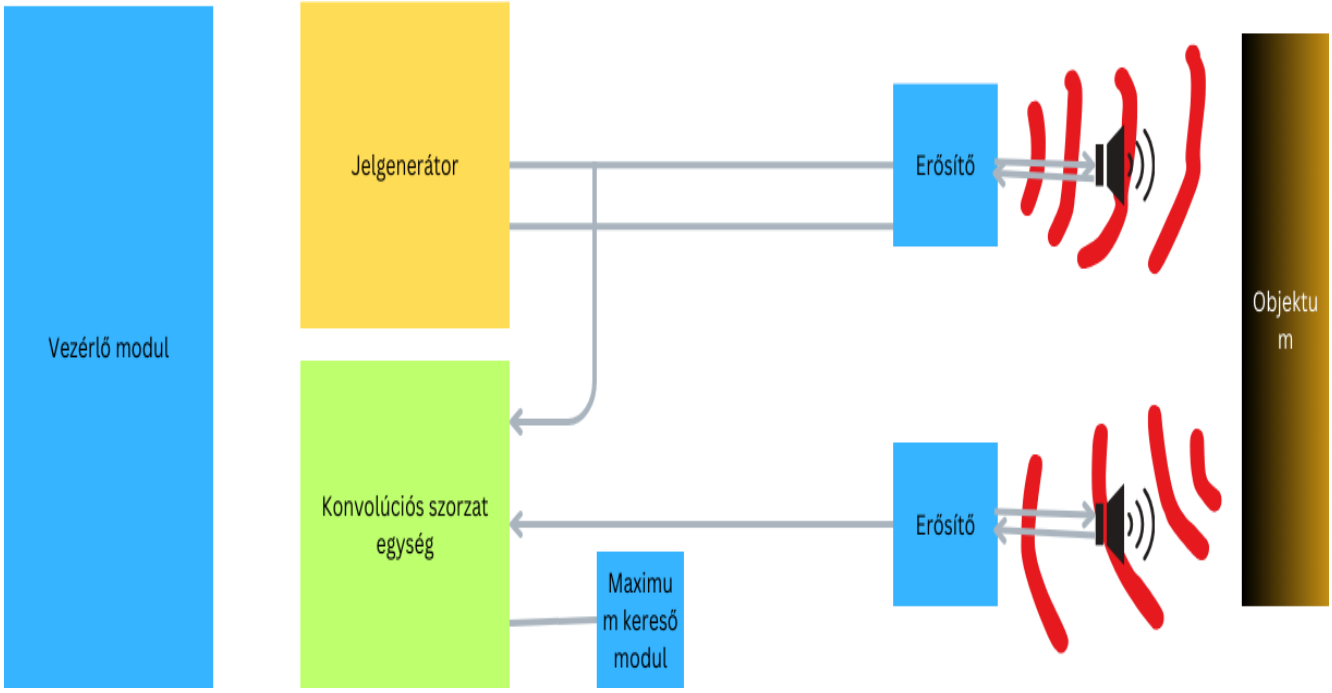
5. Projekt célja

A projekt célja egy ultrahangos távolságmérő megvalósítása. Ezen belül az ultrahangos szenzor által kibocsátott jeleket FPGA lapon dolgozzuk fel. A távolságmérőnek eszköznek kell tudjuk változtatni a frekvenciáját, amit adatutas véges állapotú automatával(Finite State Machine with Datapath-FSMD) oldunk meg. A frekvenciák 43 kHz és 37kHz között mozog, ami azt jelenti hogy 43 a legnagyobb és a 37 a legkisebb frekvencia amit mérni tudunk. A projekten belül több modult építünk fel, ezek között a jelgenerátor, konvolúciós egység, összeadó és maximum kereső modulok, illetve legvégén a vezérlő egység ami mindezeket kontrollálja.

6. Megvalósítás

Működési elv: Jelnek kiküldése, idő mérése amíg a jel teljesen visszaverődött, majd ebből a mért időből kiszámítani a távolságot.

Alábbi képen bemutatom a megvalósítandó projekt tömbvázlatát:

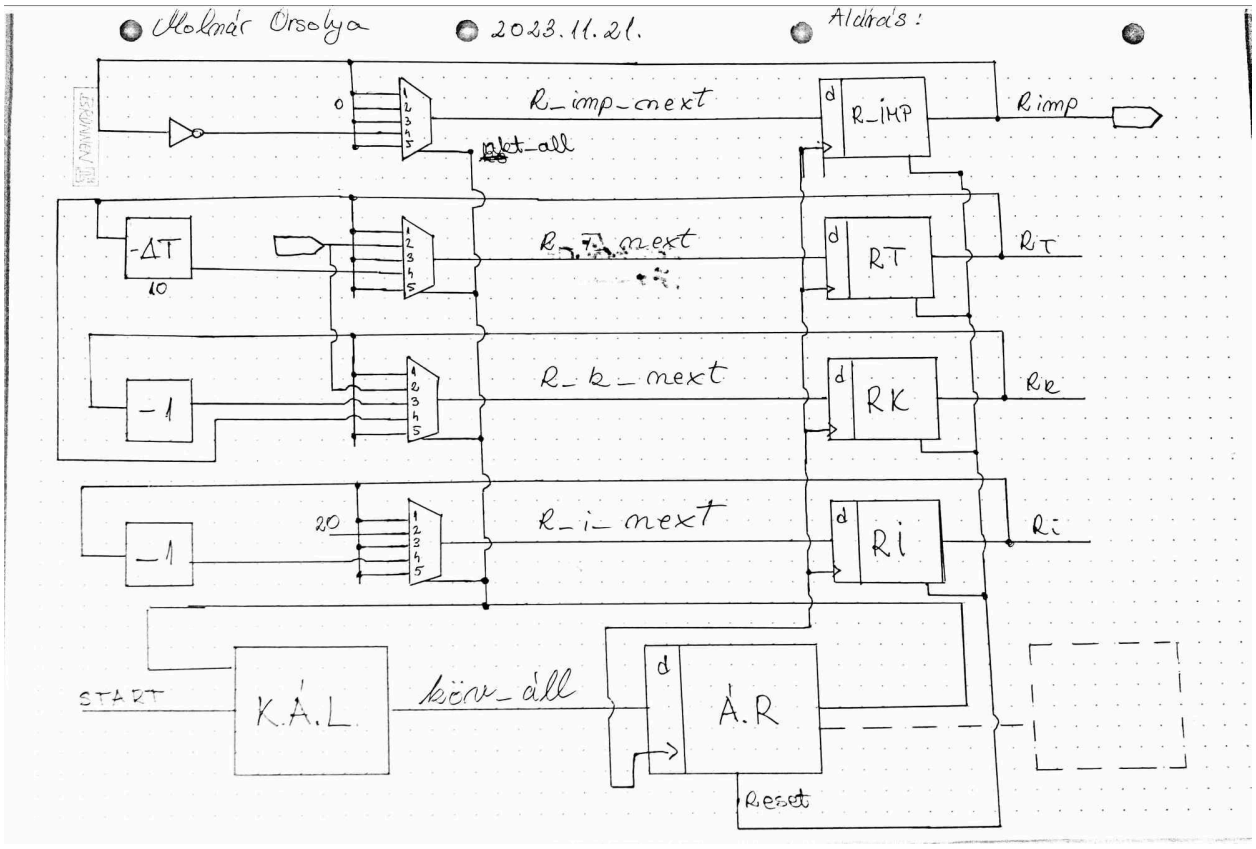


Tervezés tömbvázlata

a. Vezérlő modul

A vezérlő modul feladata az, hogy irányítja a szerkezet többi modulját.

b. Jelgenerátor



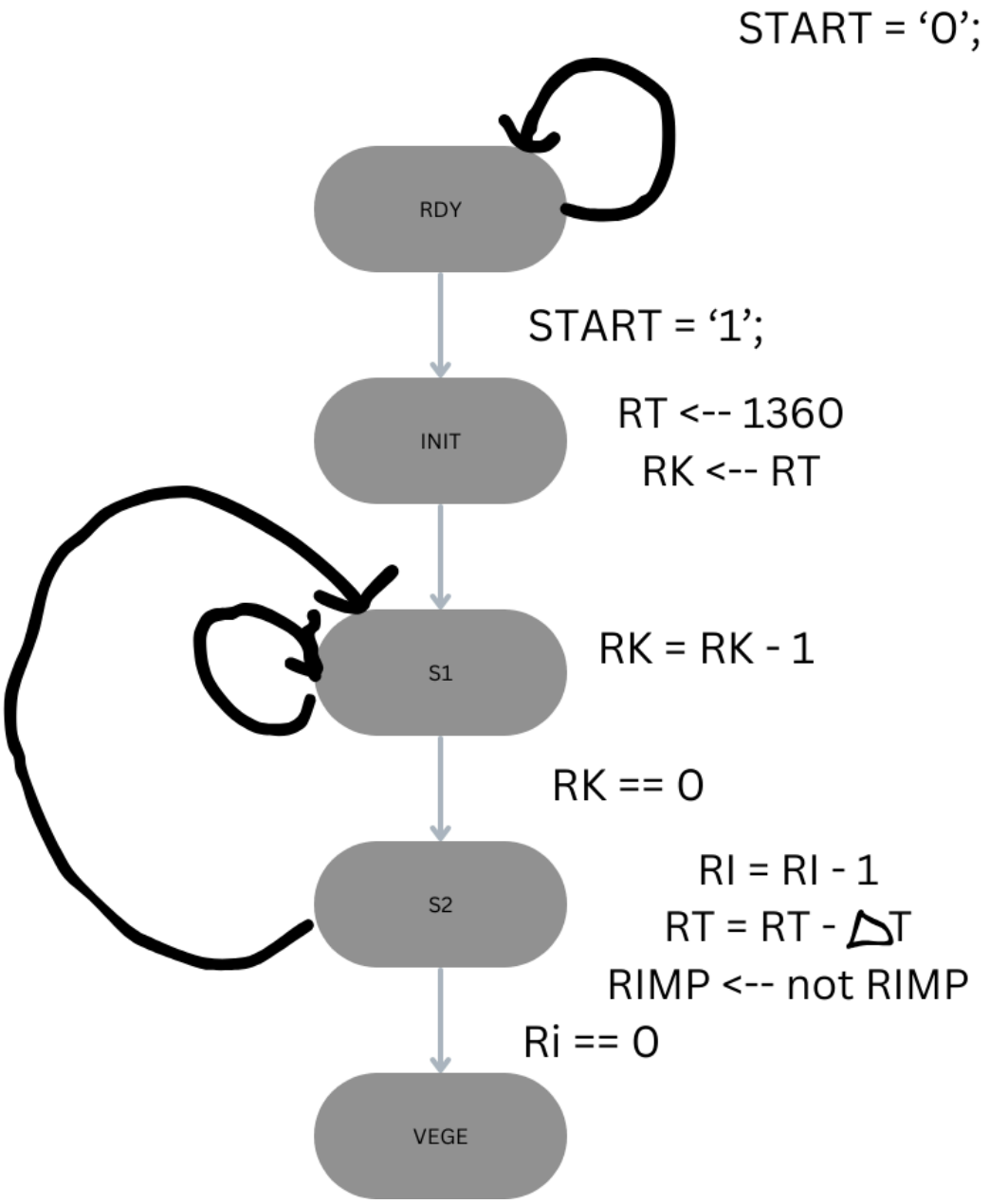
Jelgenerátor - tömbvázlat

A fenti ábrán a jelgenerátor tömbvázlata látható. Működési elve a következő:

A jelgenerátor egy olyan eszköz, amely konstans négyszögjelet generál. Ennek a jelnek az amplitúdóját az FPGA korlátozza 5V(volt)-ig. A generált jel frekvenciája a tartományban 37kHZ és 43kHz között mozog.

FSMD típusú megoldás: A rendszer működését egy FSM(Finite State Machine) vagy más néven FSMD(Finite State Machine with Datapath) típusú megoldás irányítja. Összesen öt állapota van, amelyek egymásra következnek a következő módon:

ÁLLAPOT	RI	RT	RK	RIMP	0
RDY	Ri = Ri	Rt ← Rt	Rk ← Rk	Rimp←Rimp	0
INIT	Ri = 20	Rt = 1360	Rk = 1360	Rimp = 0	0
S1	Ri ← Ri	Rt ← Rt	Rk ← Rk - 1	Rimp←Rimp	0
S2	Ri ← Ri - 1	Rt←Rt- Δ T	Rk ← Rt	Rimp←¬Rimp	0
VEGE	Ri ← Ri	Rt ← Rt	Rk ← Rk	Rimp ← Rimp	1



Véges állapotú automata(FSMD)

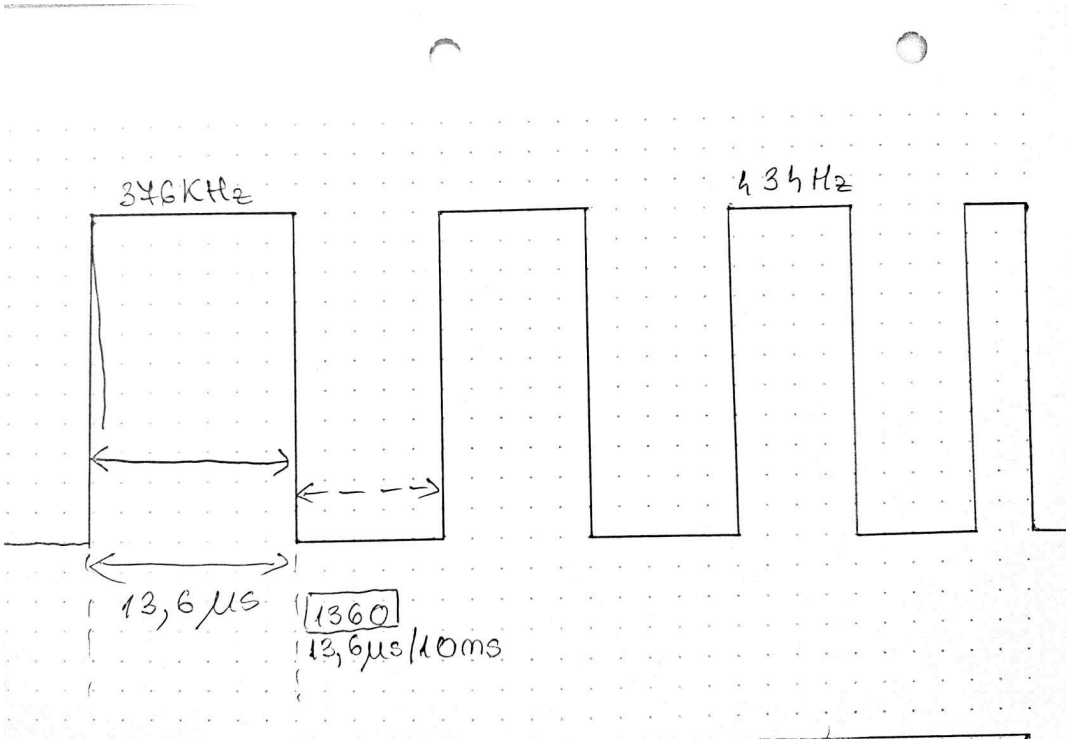
1. RDY(Ready) állapot: A rendszer készenléti állapotban van, és várja, az indulást
2. INIT(Initialization) állapot: Ebben az állapotban inicializáljuk a rendszer változóit. Ezek a változók Ri(fél impulzusok száma), Rt(jel fél impulzusainak száma) és Rk(impulzusok száma a váltásig). Az értékeik ebben az állapotban beállítódnak a következőképpen: Ri = 20, Rt = 1360, Rk = 1360, Rimp = 0).
3. S1 és S2 állapotok: Ezek az állapotok a számítások fő részét tartalmazzák. Az S1 állapotban az Rk-t számláljuk visszafelé, és amikor eléri a 0 értéket, a rendszer S2

állapotba vált. Az S2 állapotban az Rt értékét csökkentjük 10-zel, majd az Rk értéke az Rt-vel egyenlővé válik. Ezután a folyamat vissatér az S1 állapotba, amíg az Ri nagyobb, mint 0.

4. VEGE(End) állapot: Amikor az Ri értéke eléri a 0-t, a rendszer VÉGE állapotba kerül, ahol egy flag érték vált 1-re. Ez a flag jelzi, hogy a rendszer befejezte a működését, és készen áll a megszakításra.

Az Rimp változó azáltal szabályozza, hogy a jel logikai 1 vagy 0 értéket vegyen fel. Az INIT állapotban az értékeit a következőképpen számoljuk ki:

Az 1360 érték megfelel a 37 kHz frekvenciának időben, ami 13.6 mikroszekundum(µs). Mivel az FPGA órajele 100 MHz, ezért a rendszernek 1360 impulzust kell számolnia. A 43 kHz-nek megfelelő idő 11.6 mikroszekundum(µs), így a két időtartam közötti különbség 200 mikroszekundum(µs). Mivel 10-es lépéssel számolunk, ezért a teljes jel összesen 20 impulzusból fog állni.



Ebben az esetben látszik, hogy minden órajelnél 10-zel csökkentjük, ami azért van, hogy az impulzus jel ne legyen egyforma az elején és a végén. Tudni kell, hol kezdődik, hol a közepe és a vége. Így azt is tudjuk ellenőrizni, ha teljesen visszaérkezett a jel. Ha végig, minden félperiódusban ugyanannyi lenne, akkor nem tudnánk megfelelő módon ellenőrizni, ha a jelünk jól generálódott ki.

Mivel a jel kell változtassa impulzusát (0 vagy 1), az S2 állapotban miután leszámoltuk tagadjuk az RIMP-et, hogy változzon meg. 1360 ciklusban 1, majd 1360 - 10 ciklusban 0, majd 1360 - 10 - 10 (20) megint 1 stb.

Kódban

```
entity jelgenerator is
    Port ( clk : in STD_LOGIC;
           start : in STD_LOGIC;
           reset : in STD_LOGIC;
           N : in STD_LOGIC_VECTOR (7 downto 0) ; --fel impulzusok szama
           deltaT :in STD_LOGIC_VECTOR (7 downto 0) ;
           Rinit_imp : in STD_LOGIC_VECTOR (13 downto 0) ;
           imp : out STD_LOGIC );
end jelgenerator;
```

A jelgenerátor entitás portjai definiálják a modul bemeneteit és kimenetét. Az órajel (clk), a kezdési jel (start), a rendszer visszaállítása (reset), a fél impulzusok száma (N), az impulzusok közötti időtartam (deltaT), az inicializációs állapotban használt impulzusok száma (Rinit_imp), valamint a kimeneti impulzus (imp).

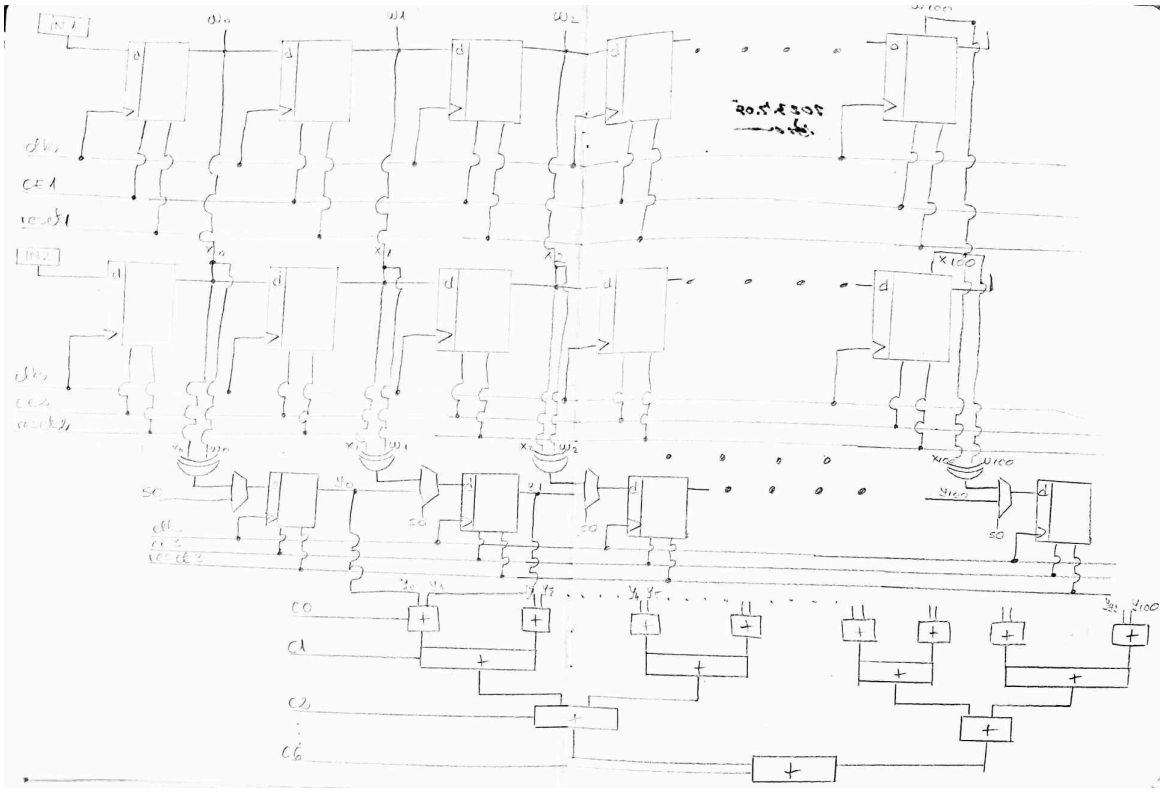
```
type allapot_tipus is ( RDY , INIT , S1 , S2, VEGE );

signal akt_all, kov_all : allapot_tipus;
```

Az állapotgép az akt_all és kov_all jelekkel van reprezentálva, és azt határozza meg, hogy a rendszer milyen állapotban van éppen. Az állapotok: RDY, INIT, S1, S2, VEGE. Az akt_all az aktuális állapot, a kov_all pedig a következő állapot.

```
AR : process ( clk , reset )
begin
    if reset = '1' then
        akt_all<= RDY ;
    elsif clk ' event and  clk = '1' then
        akt_all <= kov_all;
    end if ;
end process AR ;
```

Az AR folyamatban az órajel (clk) és a rendszer visszaállítása (reset) alapján aktualizálják az állapotokat. Ha a rendszer visszaállításra kerül, akkor az állapot RDY-ra állítódik.
A KAL folyamatban meghatározzák a következő állapotokat az aktuális állapot és a bemeneti jelek alapján. Például, ha a rendszer készen áll (RDY állapot) és elindítják (start='1'), akkor a következő állapot INIT lesz.

c. Konvolúciós modul



Konvolúciós egység tömbvázlata

A konvolúciós szorzat egy olyan folyamat, amely a kiküldött és a fogadott jelsorozatok összegzésével határozza meg a két jel közötti késést, ezzel meghatározva a távolságot. Ennek érdekében mindkét jel sorozatot mintavételezni kell, és a mintavételezési frekvenciát 1 MHz-re állítjuk be. A konvolúciós szorzat alkalmazásakor a pipeline elvet követjük, amely azt jelenti, hogy az adatokat sorban dolgozza fel a modul, ahogy azok következnek.

A konvolúciós szorzat megvalósításához D típusú tárolókat használunk. Az elküldeni kívánt jel egy sorban kerül elhelyezésre egy D típusú tárolóban, míg a visszavert jel a másik sorban egy másik D típusú tárolóba kerül. A két jel különbségét úgy számoljuk ki, hogy mindkét jelet bemutatjuk egy kizáró vagy kapunak, aminek kimenete 1, ha a két jel egyforma, és 0, ha a két jel különbözik. Ezeket az értékeket összeadva a kimeneten megkapjuk a konvolúció eredményét.

A konvolúciós szorzat folyamata tehát nemcsak a jelek összegezésére korlátozódik, hanem a két jel közötti különbség kiértékelésére is, ami a távolság meghatározásához vezet. A D típusú tárolók és a kizáró vagy kapu alkalmazása segíti a hatékony feldolgozást, és a pipeline elv biztosítja, hogy az adatokat folyamatosan és sorrendben dolgozzuk fel.

Kódban

```
entity konvolucio is
    generic (RSZ : natural :=7);
    Port ( IN1 : in STD_LOGIC; --bemeneti jel 1
           IN2 : in STD_LOGIC; --bemeneti jel 2
           src_clk : in STD_LOGIC; --orajel
           --jelolok a bemeneti jelek engedelyezesere
           CE1 : in STD_LOGIC;
           CE2 : in STD_LOGIC;
           CE3 : in STD_LOGIC;
           CE4: in STD_LOGIC;
           --reset jelek a regiszterekhez
           R1 : in STD_LOGIC;
           R2 : in STD_LOGIC;
           R3 : in STD_LOGIC;
           S0 : in STD_LOGIC; --szuro
           konv_out : out STD_LOGIC_VECTOR(8 downto 0)); --eredmeny kimenete
end konvolucio;
```

Az állapotgépet egy generate-for loop segítségével valósítják meg. A loop RSZ alkalommal fut le, ahol minden alkalommal az állapotgép egy kisebb részét hozza létre a konvolúcióhoz.

```
architecture Behavioral of konvolucio is
    signal w : STD_LOGIC_VECTOR(RSZ downto 0);
    signal x : STD_LOGIC_VECTOR(RSZ downto 0);
    signal y : STD_LOGIC_VECTOR(RSZ downto 0);
    signal result : STD_LOGIC_VECTOR(8 downto 0) := (others => '0');
```

Az áramkörben három regiszter (w, x, y) van, amelyek tárolják az előző jelértékeket. Ezen regiszterek értékeit az óraingerek és a reset jelek alapján frissítik.

A konvolúciót a osszead_1 és osszead_i folyamatok valósítják meg. A S0 értékétől függően XOR (xnor) kapukat alkalmaznak a bemeneti jeleken, és az eredményt a result változóba számolják be.

```
case S0 is
   when '1' =>
       y(i) <= w(i) xnor x(i) ;
       if y(i) = '1' then
           result <= std_logic_vector(unsigned(result) + 1);
       end if;
   when others =>
       y(i) <= y(i-1);
end case;
```

Amikor S0 értéke '1', a konvolúciós művelet során az IN1 és IN2 bemeneti jeleken egy XOR (xor) kapuval alkalmazza a szűrőt, és az eredményt hozzáadja az result változóhoz. Amennyiben a y(i) jel értéke '1', a result értékét növeli eggyel. Amikor S0 értéke nem '1', a y(i) értéke a korábbi értékének (y(i-1)) lesz egyenlő, tehát a konvolúciós művelet nem módosítja az eredményt.

A result változó a konvolúciós modul kimeneti jelét tárolja, és a kódban a számolás során növekszik. A növelés oka az, hogy a konvolúciós modul működése során, amikor az eredmény y(i) értéke '1' (logikai igaz), a result értékét növelik.

Az állapotgép két fő részből áll. Az első rész az if_1 ág, ami a legelső állapotot kezeli külön, és a második rész a if_i ág, ami a következő állapotokra vonatkozik.

d. Összeadó modul

A fenti ábra egyben tartalmazza az összeadó modult is. Ebben a modulban összegezzük a konvolúció eredményeit, majd a legutolsó kikerül a maximum kereső modulban, amiről a következőkben lesz szó.

Kódban

```
reg_osszeado: process(y, src_clk, R3)
    variable osszeg : STD_LOGIC_VECTOR(8 downto 0) := (others => '0');
    variable osszeg_kov: STD_LOGIC_VECTOR(8 downto 0) := (others => '0');

    begin
        if R3 = '1' then
            osszeg := (others => '0');
        elsif src_clk ' event and src_clk = '1' then
            if CE3 = '1' then
                osszeg := osszeg_kov;
            end if;
        end if;
end process reg_osszeado;
```

A folyamat első része felelős a regiszterek inicializálásáért. Ha a R3 (reset) jel aktív, akkor az összeg regiszter (osszeg) nullázva lesz.

Ha az órajel (src_clk) emelkedő élét észleli, és a CE3 (clock enable) jel aktív, akkor a regiszterek frissülnek. Az összeg regiszter a következő ciklusban az előző ciklusban számolt értéket veszi fel.

A folyamat végül az összeg regiszter értékét állítja be a következő ciklusban használt értéknek. Ez a következő ciklusban kerül felhasználásra a konvolúciós modulban.

e. Maximumkeresés

A maximum kereső egység felelős azért, hogy minden egyes mintavétel alkalmával összehasonlítsa a különböző konvolúciók eredményeit, és meghatározza, melyik konvolúció összege a legnagyobb. Amint megtalálja a maximális összeget, a rendszer ellenőrzi, mennyi idő telt el a jel kiküldése óta.

Az eltelt idő és a jel terjedési sebességének ismeretében a maximum kereső egység képes kiszámolni a távolságot az objektumtól, amelyről a jel visszaverődött. Ez a számítás segít pontosan meghatározni az objektum távolságát a jel kiküldésétől számított idő alapján, és a jel terjedési sebességének ismeretében.

7. Gazdasági elszámolás

Költségek:

1. Hardverköltségek:
    a. Nexys A7-100T FPGA fejlesztőkártya beszerzése - 1.500 RON
    b. Ultrahangos szenzor és erősítő vásárlása - 50 RON
2. Szoftverköltségek:
    a. Vivado 2021.2(minimum) szofverlincence költsége - 13.500
3. Munkaerőköltségek:
    a. Fejlesztők, tesztelők, projektmenedszer költségei - 5.000 - 10.000 RON
4. Egyéb:
    a. Raktározás, szállítás, iroda bérlése, egyéb. - 8.000 RON

Bevételek:

1. Termékeladás: ultrahangos távolságmérő készülék eladási ára - 500 RON

Amortizáció:

1. Beruházások amortizációja: Hardverek, szoftverek és más eszközök élettartama alapján. - kb. 4.000 RON

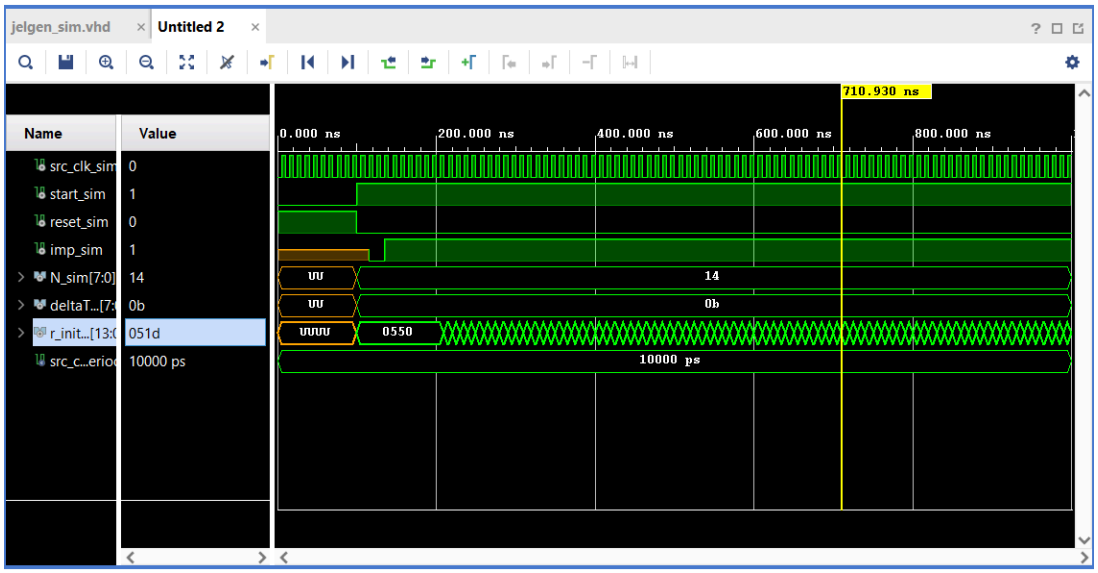
Éves bevétel: Termékeladási ár * éves eladott darabszam = 500 * 2.000 = 1.000.000

Éves költségek: Hardver Költségek + Szoftver Költségek + Munkaerőköltségek + Egyéb költségek + Amortizáció = 1550 + 13.500 + 10.000 + 8.000 + 4.000 = 37.050

Éves nettó jövedelem: Éves bevétel - Éves költségek = 1.000.000 - 37.050 = 962.950

Visszatérülési idő: Beruházás / Éves nettó jövedelem = 4.000 / 962.950 ≈ 0,004 év (vagyis 0,004 * 12 hónap ≈ 0,048 hónap vagy 0,0,48 * 30 ≈ 1,44 nap) => a beruházás rövid időn belül megtérül

8. Szimuláció



Jelgenerátor szimuláció

A fenti ábrán látható a jelgenerátor szimulációja, azonban látható, hogy nem teljesen pontos. Ez valószínűleg azért történik, mert a jelgenerátor kódjában hiba jelent meg, pontosabban az egyik with select utasításban hiányzik egy feltétel, emiatt nagyon sűrűn generálódik ki a jel.

9. Következtetések

Újra át kell gondolni, és kijavítani a hibákat. Ki lehet egészíteni azzal, hogy egy hétszegmenses kijelzőre tegyük ki a mért távolságot.

10. Könyvészet

[1] Carlos, Jesús, Jiménez, Fernández., Pilar, Parra, Fernandez., Carmen, Baena, Oliva., Manuel, Valencia, Barrero., F., Eugenio, Potestad, Ordonez. "Distance measurement as a practical example of FPGA design." null (2018). doi: 10.1109/TAEE.2018.8476143

[2] Brassai, Sándor Tihamér (2019) “Újrakonfigurálható digitális áramkörök tervezési és tesztelési módszerei.” Scientia Kiadó, Kolozsvár. ISBN 978-606-975-020-9, URI: http://real.mtak.hu/122602/1/Brassai%20Tihamer_UKDA_REAL.pdf

[3] “FPGA Implementation of distance Measurement with Ultrasonic Sensor”, URI: https://www.pantechsolutions.net/fpga-implementation-of-distance-measurement-with-ultrasonic-sensor

[4] Dr. Shirshendu, Roy (2022) “Interfacing Ultrasonic Sensor with FPGA - Digital System Design”, URI: https://digitalsystemdesign.in/interfacing-ultrasonic-sensor-with-fpga/