

# Programozható LED-fűzéren alapuló reklámpanel

## LED fűzér vezérlése: adatok kiírása

Patka Zsolt-András  
Számítástechnika BSc

2019.12.16

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>1</b>
<b>2. Követelmények</b>	<b>1</b>
2.1. Funkcionális követelmények . . . . .	1
2.2. Nem funkcionális követelmények . . . . .	1
2.3. Fejlesztési követelmények . . . . .	1
<b>3. Rendszer-specifikáció</b>	<b>2</b>
<b>4. Tervezés</b>	<b>2</b>
4.1. WS2813 egyszálú adatátvitel protokoll leírása . . . . .	2
4.1.1. A 24 bit-es kód . . . . .	2
4.1.2. Bit-ek küldési sorrendje . . . . .	2
4.1.3. Időzítések . . . . .	2
4.2. Véges állapotú adatútas automata . . . . .	3
4.3. WS2813_Driver tervezése . . . . .	3
4.3.1. Elvégzendő RT műveletek azonosítása . . . . .	3
4.3.2. Adatfüggőségek identifikálása . . . . .	3
4.3.3. Célregiszterek azonosítása . . . . .	4
4.3.4. Különböző fázisokban elvégzendő műveletek . . . . .	4
4.3.5. Kapcsolási rajz . . . . .	5
4.3.6. Állapotdiagram . . . . .	6
4.4. WS2813_Controller tervezése . . . . .	8
4.4.1. Elvégzendő RT műveletek azonosítása . . . . .	8
4.4.2. Adatfüggőségek identifikálása . . . . .	8
4.4.3. Célregiszterek azonosítása . . . . .	8
4.4.4. Különböző fázisokban elvégzendő műveletek . . . . .	9
4.4.5. Kapcsolási rajz . . . . .	9
4.4.6. Állapotdiagram . . . . .	10
4.5. WS2813_TopLevel tervezése . . . . .	10
4.5.1. Elvégzendő RT műveletek azonosítása . . . . .	10
4.5.2. Adatfüggőségek identifikálása . . . . .	10
4.5.3. Célregiszterek azonosítása . . . . .	11
4.5.4. Különböző fázisokban elvégzendő műveletek . . . . .	11
4.5.5. Kapcsolási rajz . . . . .	11
4.5.6. Állapotdiagram . . . . .	12
4.6. Alegységek . . . . .	12
4.7. Idődiagram . . . . .	12
<b>5. Szimuláció eredménye</b>	<b>12</b>
5.1. WS2813_Driver . . . . .	12
5.2. BRAM memória . . . . .	14
<b>6. Működés közbeni tesztelés - Integrated Logic Analyser</b>	<b>14</b>
6.1. WS2813_Driver . . . . .	14
6.2. Teljes rendszer működés közbeni tesztelése . . . . .	16

<b>A. Tervezési lépések</b>	<b>18</b>
A.1. Állapotok . . . . .	18
A.1.1. Első iteráció - 2019.10.14 . . . . .	18
A.1.2. Második iteráció - 2019.10.28 . . . . .	19
A.2. Alegységek . . . . .	21
A.2.1. Első iteráció - 2019.10.14 . . . . .	21
A.2.2. Második iteráció - 2019.10.28 . . . . .	22
A.3. Idődiagram . . . . .	23
A.3.1. Első iteráció - 2019.10.26 . . . . .	23

# 1. Bevezető

A projekt célja egy LED-fűzér vezérlése és ennek segítségével egy reklámszöveg megjelenítése. Ehhez egy Basys3 FPGA lap és egy Worldsemi WS2813 ledfűzér lesz felhasználva.

## 2. Követelmények

### 2.1. Funkcionális követelmények

- Lehetséges legyen egy reklámszöveget kiírni a ledfűzerek által létrehozott mátrix-ra.
- Egy sorban minimum 100 LED található
- Az egyes ledfűzerek szinkronba működjenek
- A kiírás párhuzamosan történik az adott ledfűzereken
- Másodpercenként 60 frissítés (60 Hz)
- Mozgó szöveg

### 2.2. Nem funkcionális követelmények

- Benti használatra van tervezve
- Ha az FPGA-lapnak lesz készítve külön tokozat, akkor IP31-es standardnak kell megfeleljen
- Ha az FPGA-lapnak nem lesz készítve külön tokozat, akkor nem felel meg IP standardnak (IP00)

### 2.3. Fejlesztési követelmények

- Implementáció VHDL nyelvben
- Szimulációs állomány a rendszer tesztelésére
- Adatok kiírása lehetséges a LED fűzérre
- Modularitás
  - WS2813\_Driver: Egy 24 bit-es blokkot küldő modul
  - WS2813\_Controller: Egy szálon több 24 bit-es blokkot kiküldő, a WS2813\_Driver modult felhasználó modul
  - WS2813\_TopLevel: Több ledfűzérre való kiírást lehetővé tevő modul, több WS2813\_Controller modul BRAM-al való összekötése által
- Opcionális:
  - Pár betű kódolása (3-4)
  - Betűk tárolása BRAM memóriában

### 3. Rendszer-specifikáció

- WS2813 egyszálú adatátvitel protokolljának helyes használata
- Worldsemi WS2813 90 LED-es ledfűzér van felhasználva
- Digilent Basys3 FPGA vezérli a ledfűzért

### 4. Tervezés

#### 4.1. WS2813 egyszálú adatátvitel protokoll leírása

A LED-eket vezérlő áramkörök egymás után vannak bekötve úgy, hogy az egyik áramkörnek az adatkimenete a következő áramkörnek az adatbemenetét képi. Egyszálú az adatátvitel, fontos a protokoll betartása, ahhoz, hogy adatokat tudjanak megjeleníteni a LED-fűzéken.

Amikor egy áramkör megkap egy 24 bit-es kódot, akkor ezt addig tárolja amíg más kódot nem kap, vagy a tápforrást el nem veszi.

##### 4.1.1. A 24 bit-es kód

A 24 bit-es kód a következőképpen kell kinézzen:

**8 bit GREEN | 8 bit RED | 8 bit BLUE**

Az adatátvitel a következő sorrendben kell történjen:

1. GREEN
2. RED
3. BLUE

##### 4.1.2. Bit-ek küldési sorrendje

Az egyes byte-ok küldését úgy kell elvégezni, hogy az **MSB**-vel kell kezdeni és haladni az **LSB** fele. 24 bit-es kód részletesebb felbontása:

- *G7 G6 G5 G4 G3 G2 G1 G0 | R7 R6 R5 R4 R3 R2 R1 R0 | B7 B6 B5 B4 B3 B2 B1 B0*

A küldés a következő sorrendben kell elvégződjön (balról jobbra):

- **G7 G6 G5 G4 G3 G2 G1 G0 | R7 R6 R5 R4 R3 R2 R1 R0 | B7 B6 B5 B4 B3 B2 B1 B0**

##### 4.1.3. Időzítések

Minden 24 bit-es adatátvitel után kell legalább 50  $\mu$ s-ot várakozni, alacsony feszültségen. Ez jelzi azt, hogy egy 24 bit-es blokk továbbítása megtörtént.

Az egyes bit-ek átvitele a következőképp történik:

- Logikai 1-es
  - 0.8  $\mu$ s-ot magas feszültségen
  - 0.45  $\mu$ s-ot alacson feszültségen

- Logikai 0-ás
  - 0.4  $\mu$ s-ot magas feszültségen
  - 0.85  $\mu$ s-ot alacson feszültségen
- 24 bit-es adatblokk küldése után:
  - > 50  $\mu$ s-ot alacsony feszültségen

A bit-ek továbbításánál egy +/- 150 ns-os eltérés megengedett.

A várakozási értékek nem az adatlapból, hanem egy útmutatóból[1] lettek kivéve. Az útmutató szerint az adatlapban levő értékek rosszul vannak kiszámolva.

Ha az útmutatóban megadott értékek nem megfelelő működéshez vezetnek, akkor az adatlapban[2] levő értékek lesznek felhasználva. Ez a valós rendszeren levő tesztelés közben fog kiderülni.

## 4.2. Véges állapotú adatútas automata

A fejlesztési követelményekben levő három modul implementációja a véges állapotú adatútas automata[3] tervezési mintával lett kivitelezve. Az adott tervezési minta egyszerűvé teszi a tervezést, mivel jól definiált lépéseken keresztül el lehet jutni egy működő tervhez.

## 4.3. WS2813\_Driver tervezése

Ez a modul felelős egy 24 bit-es blokk kiküldéséért. Ehhez a WS2813 egyszálú adatátviteli protokollját be kell, hogy tartsa.

A legnagyobb kihívás az adott modul megtervezésében a késleltetések implementálása. Ez úgy lett megvalósítva, hogy egy bizonyos állapotban annyit marad az automata, amíg a megfelelő mennyiségű idő eltelik.

### 4.3.1. Elvégzendő RT műveletek azonosítása

Az időzítések implementálásához egy számláló lesz használva. A 100 MHz-es órajel ami fel lesz használva időben 0.01  $\mu$ s-nak felel meg. Így a szükséges ciklusok száma (amennyit várakoztatni kell bizonyos feszültség szinten, ahhoz, hogy a WS2813 protokollja be legyen tartva) egész. Ebből már látható, hogy definiálható két művelet:  $i \leftarrow ciklus\_szam$ ,  $i \leftarrow i - 1$  (ahol  $i$  a számláló).

Az egyszerűség kedvéért, egy belső regiszternek, amely a színadatokat tartalmazza (**data**), mindig a 24. bitje kerül kiküldésre. Tehát minden bit kiküldése után ennek a regiszternek az értékeit el kell csúsztatni (shift-elni) egyet balra:  $data \leftarrow data \ll 1$

Egy 24-bites blokk kiküldése után kell küldeni egy "RES" jelet, vagyis több ideig alacsony feszültségen kell tartani a kimenetet. Ehhez számolni kell a már elküldött bitek számát  $bit\_count \leftarrow bit\_count - 1$ , ahol  $bit\_count$  az eddig elküldött bitek számát tartalmazza. Az elküldött bit-ek 23-tól számolódnak lefele, mivel a nullához való hasonlítás hatékonyabb, mint egy adott értékhez való hasonlítás.

### 4.3.2. Adatfüggőségek identifikálása

Mivel nagyon egyszerű RT műveletekkel meg lehet oldani az adott feladatot, nem merülnek fel adatfüggőségek.

### 4.3.3. Célregiszterek azonosítása

Szükséges regiszterek:

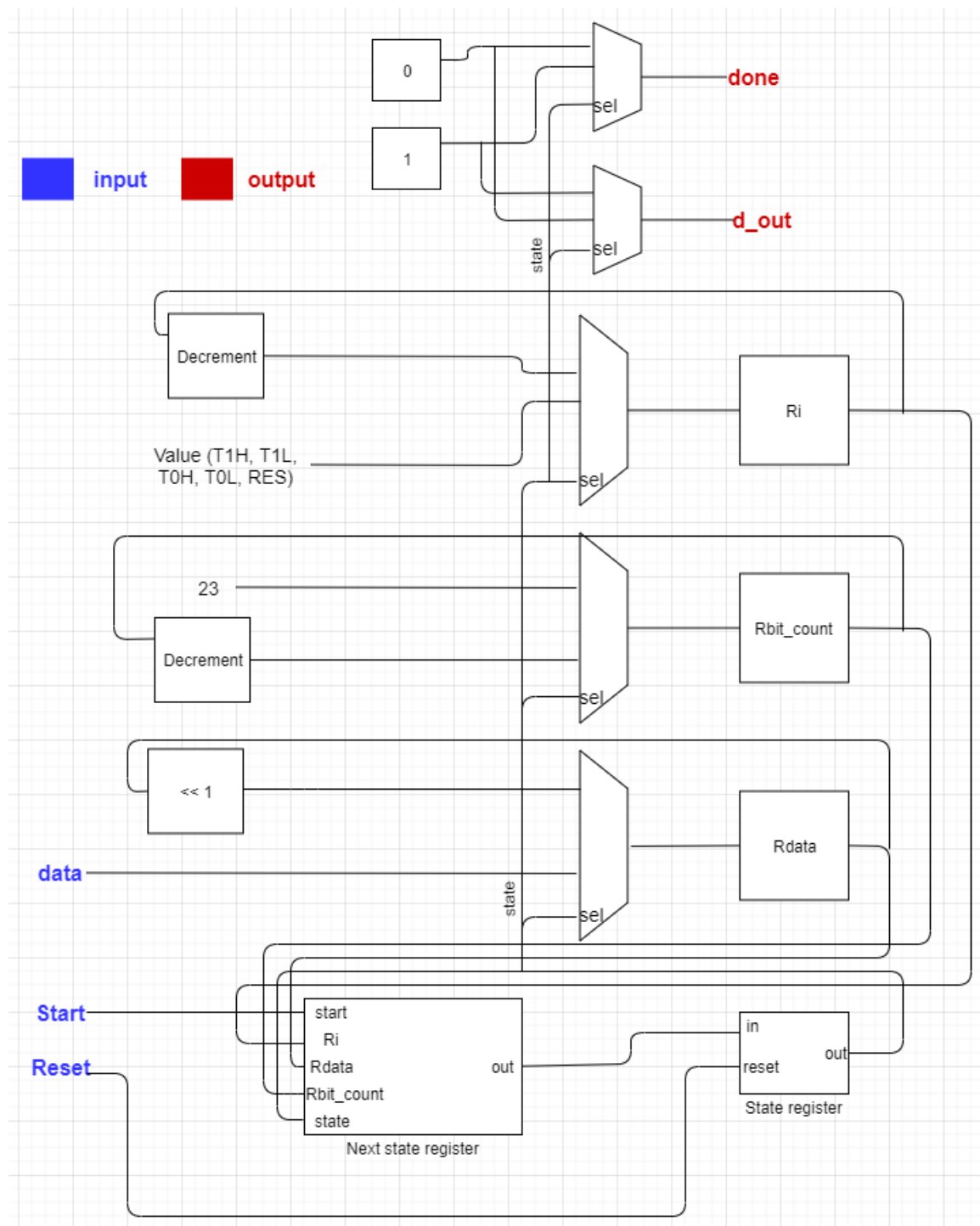
- $R_i$ : ciklusszámláló regiszter a késleltetésekhez.
- $R_{bit\_count}$ : Bitek számláló regiszter, a 24 bit-es blokk elküldésének érzékeléséhez.
- $R_{data}$ : Elküldendő adatot tartalmazó regiszter

### 4.3.4. Különböző fázisokban elvégzendő műveletek

1. táblázat. WS2813\_Driver - Különböző fázisokban elvégzendő műveletek

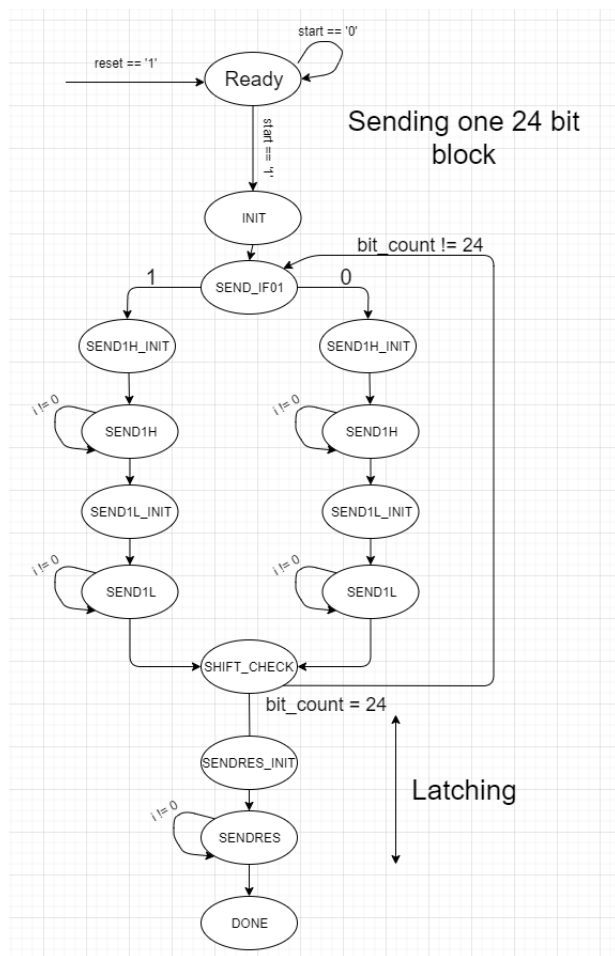
Állapot	$R_i$	$R_{bit\_count}$	$R_{data}$	d_out	done
READY	$R_i$	$R_{bit\_count}$	$R_{data}$	'U'	'0'
INIT	$R_i$	24	data	'U'	done
SEND_IF01	$R_i$	$R_{bit\_count}$	$R_{data}$	'U'	done
SEND1H_INIT	$T1H$	$R_{bit\_count}$	$R_{data}$	'1'	done
SEND1H	$R_i - 1$	$R_{bit\_count}$	$R_{data}$	d_out	done
SEND1L_INIT	$T1L$	$R_{bit\_count}$	$R_{data}$	'0'	done
SEND1L	$R_i - 1$	$R_{bit\_count}$	$R_{data}$	d_out	done
SEND0H_INIT	$T0H$	$R_{bit\_count}$	$R_{data}$	'1'	done
SEND0H	$R_i - 1$	$R_{bit\_count}$	$R_{data}$	d_out	done
SEND0L_INIT	$T0L$	$R_{bit\_count}$	$R_{data}$	'0'	done
SEND0L	$R_i - 1$	$R_{bit\_count}$	$R_{data}$	d_out	done
SHIFT_CHECK	$R_i$	$R_{bit\_count} - 1$	$R_{data}$	d_out	done
SHIFT	$R_i$	$R_{bit\_count}$	$R_{data} << 1$	d_out	done
SENDRES_INIT	$TRES$	$R_{bit\_count}$	$R_{data}$	'0'	done
SENDRES	$R_i - 1$	$R_{bit\_count}$	$R_{data}$	d_out	done
SEND_DONE	$R_i$	$R_{bit\_count}$	$R_{data}$	'U'	'1'
DONE_TODO	$R_i$	$R_{bit\_count}$	$R_{data}$	'U'	done

#### 4.3.5. Kapcsolási rajz





#### 4.3.6. Állapotdiagram



- **READY**
  - Alap állapot
  - "reset" jel esetén ide kerül vissza az automata
- **INIT**
  - inicializálja a bit-count-ot nullára
- **SENDIF\_01**
  - Megvizsgálja `data[23]`-as bit-et. Ha 1, akkor a következő állapot a **SEND1H\_INIT**, ha 0 akkor **SEND0H\_INIT**
- **SEND1H\_INIT**
  - Inicializálja az `i` változót a T1H értékre
  - A `d_out` output-ot 1-re állítja
- **SEND1H**
  - Dekrementálja az `i` változót
  - Amikor az `i` változó nulla lesz, akkor tovább megy a **SEND1L\_INIT** állapotra

- SEND1L\_INIT
  - Inicializálja az **i** változót a T1L értékre
  - A d\_out output-ot 0-ra állítja
- SEND1L
  - Dekrementálja az **i** változót
  - Amikor az **i** változó nulla lesz, akkor tovább megy a **SHIFT\_CHECK** állapotra
- SEND0H\_INIT
  - Inicializálja az **i** változót a T0H értékre
  - A d\_out output-ot 1-re állítja
- SEND0H
  - Dekrementálja az **i** változót
  - Amikor az **i** változó nulla lesz, akkor tovább megy a **SEND0L\_INIT** állapotra
- SEND0L\_INIT
  - Inicializálja az **i** változót a T0L értékre
  - A d\_out output-ot 0-ra állítja
- SEND0L
  - Dekrementálja az **i** változót
  - Amikor az **i** változó nulla lesz, akkor tovább megy a **SHIFT\_CHECK** állapotra
- SHIFT\_CHECK
  - Megnézi, hogy a **bit\_count** változó nulla-e, ha igen, akkor tovább megy a **SENDRES\_INIT** állapotra
  - Ha a **bit\_count** változó nem egyenlő nullával, akkor tovább megy a **SHIFT** állapotra
- SHIFT
  - Shift-eli a **data** std logic vectort balra eggyel; vissza megy a **SEND\_IF01** állapotra
- SENDRES\_INIT
  - Inicializálja az **i** változót a TRES értékre
  - A d\_out output-ot 0-ra állítja
- SENDRES
  - Dekrementálja az **i** változót
  - Amikor az **i** változó nulla lesz, akkor tovább megy a **SEND\_DONE** állapotra
- SEND\_DONE
  - Befejeződött a 24 bit-es blokk kiírása
  - Beállítja a **done** kimenetet 1-esre

## 4.4. WS2813\_Controller tervezése

Ennek a modulnak a feladata az adat (24 bit-es blokkok) adagolása a **WS2813\_Driver**-nek. Az adagolandó adat BRAM memóriában van tárolva. A modul végzi a címzést.

A **WS2813\_Driver** modul képes egy 24 bit-es blokk kiküldésére. Ez a modul teszi lehetővé, hogy több 24 bit-es blokk kiküldhető legyen a LED-fűzérre.

A **WS2813\_Driver** modul egy 24 bit-es blokk elküldése után a **done** jelét '1'-esre állítja, ezáltal jelzi a felsőbb szintű modulnak, hogy a küldés befejeződött. Ebben a szekcióban a **WS2813\_Driver** modul **done** jele  $Driver_{done}$ -ként lesz hivatkozva. A **done** a controller modul **done** jelére fog vonatkozni. Ugyanúgy a **WS2813\_Driver** modul **reset**, **start**  $Driver_{reset}$  és  $Driver_{start}$  lesznek.

### 4.4.1. Elvégzendő RT műveletek azonosítása

A címzéshez szükséges egy érték dekrementálása. A cím 0-tól 89-ig lehet (90 LED-es ledfűzér). A nullához való hasonlítás hatékonyabb, mint a fix értékhez való, de ebben az esetben nem ezért volt ez a megoldás felhasználva (max értékről dekrementálás nulláig).

A BRAM feltöltésekor az az elvárt, hogy az az adat ami a BRAM memóriában a 0-ás címen van, az első LED-en lesz megjelenítve (első LED-nek számít az ami a legközelebb van az FPGA laphoz). A WS2813 proktoll úgy van definiálva, hogy amint egy új adat érkezik, az összes LED elcsúsztatja a rajta levő értéket jobbra. Így az az adat ami elsőnek küldődik ki, az utolsó LED-en lesz megjelenítve. Tehát, ha 89-es címen levő adat küldődik ki elsőnek, ez lesz az utolsó LED-en megjelenítve. Értelemszerűen a 0-ás címen levő adat lesz az első LED-en megjelenítve.

Definiálható RT utasítások:  $addr \Leftarrow num_{leds}$ ,  $addr \Leftarrow addr - 1$

### 4.4.2. Adatfüggőségek identifikálása

Mivel nagyon egyszerű RT műveletekkel meg lehet oldani az adott feladatot, nem merülnek fel adatfüggőségek.

### 4.4.3. Célregiszterek azonosítása

Szükséges regiszterek:

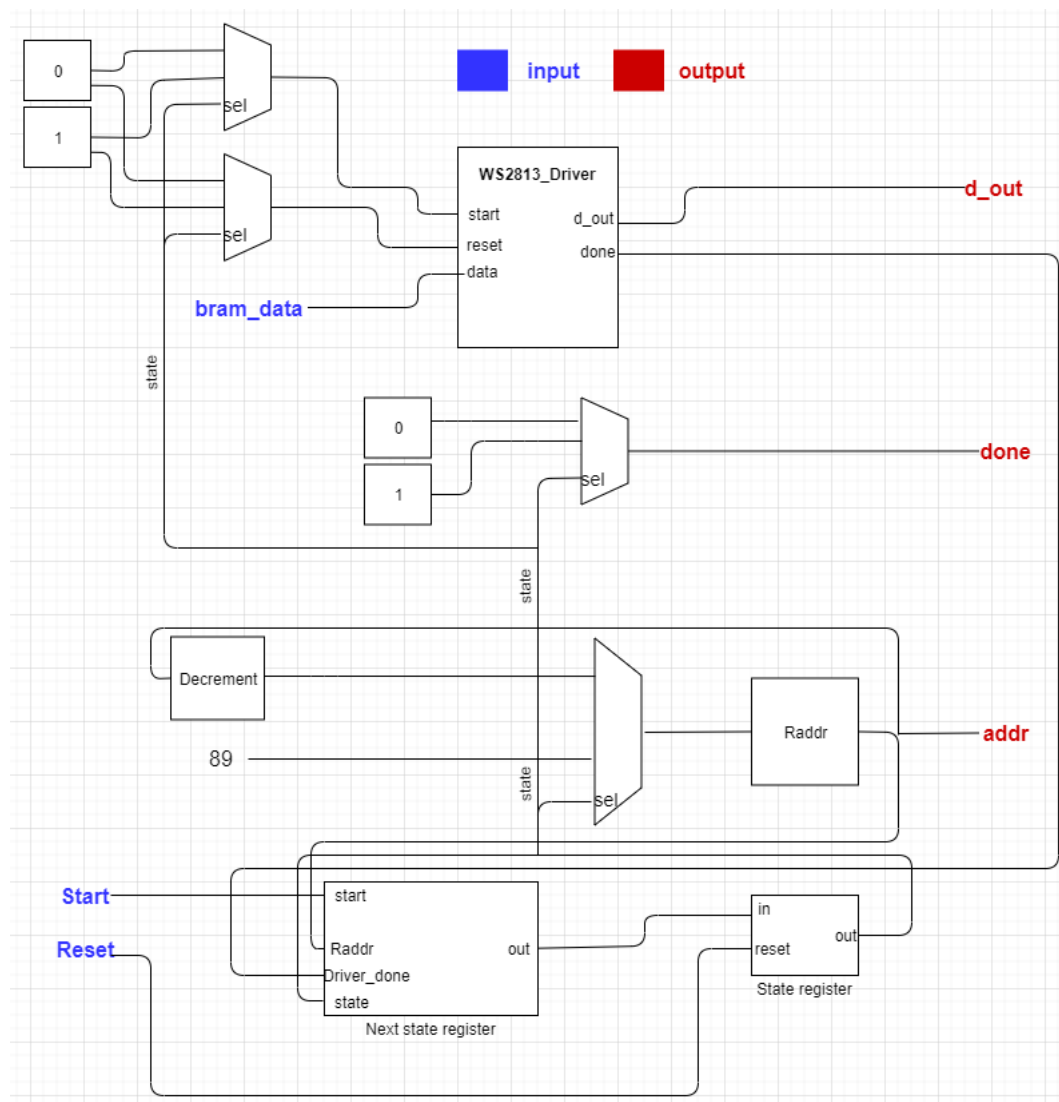
- $R_{addr}$ : címet tartalmazó regiszter, a BRAM címzéséhez van felhasználva

#### 4.4.4. Különböző fázisokban elvégzendő műveletek

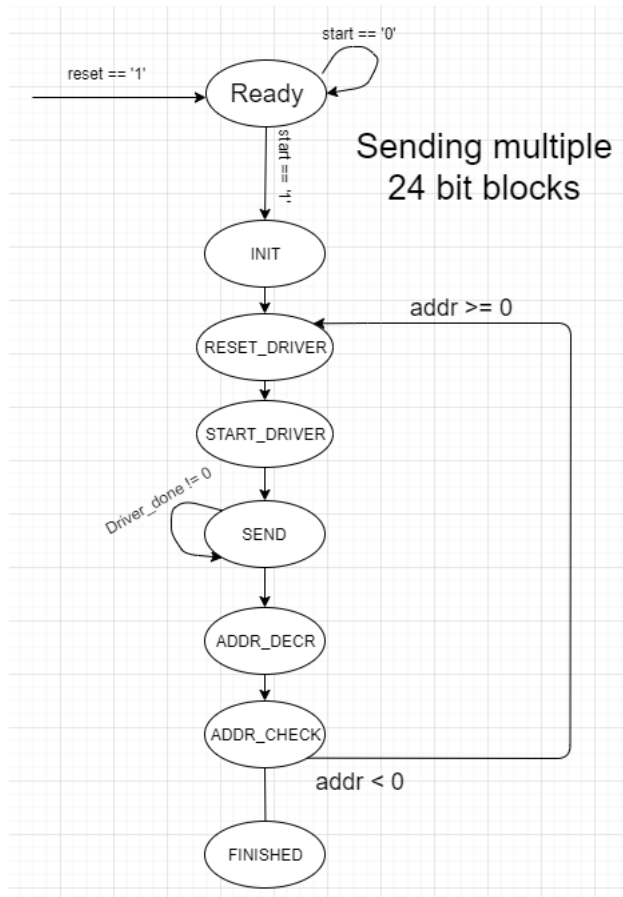
2. táblázat. WS2813\_Controller - Különböző fázisokban elvégzendő műveletek

<b>Állapot</b>	$R_{addr}$	$Driver_{reset}$	$Driver_{start}$	done
READY	$R_{addr}$	'0'	'0'	'0'
INIT	89	'0'	'0'	'0'
RESET_DRIVER	$R_{addr}$	'1'	'0'	'0'
START_DRIVER	$R_{addr}$	'0'	'1'	'0'
SEND	$R_{addr}$	'0'	'0'	'0'
ADDR_CHECK	$R_{addr}$	'0'	'0'	'0'
ADDR_DECR	$R_{addr} - 1$	'0'	'0'	'0'
FINISHED	$R_{addr}$	'0'	'0'	'1'

#### 4.4.5. Kapcsolási rajz



#### 4.4.6. Állapotdiagram



#### 4.5. WS2813\_TopLevel tervezése

Ennek a modulnak a feladata a LED-fűzők összekötése a BRAM modulokkal. Egy generikus változó által lehet definiálni, hogy hány **WS2813\_Controller** szükséges a modulban (hány LED-fűzért kell megvezérelni). A **WS2813\_Controller** modulok automatikusan lesznek kigenerálva, a generikus változó függvényében, viszont a BRAM modulokat muszáj kézzel kigenerálni és beilleszteni az adott modulba a megfelelő helyre.

##### 4.5.1. Elvégzendő RT műveletek azonosítása

A modulnak a feladata már egyszerű, csak össze kell kösse a már meglévő modulokat, ezért kevés logikát is tartalmaz, kevés RT művelet azonosítható. A modulnak tudnia kell, hogy a **WS2813\_Controller** moduljai mikor fejezték be a küldést. Ezért szükséges ezeknek a moduloknak a **done** jelei között egy logikai ÉS művelet elvégzése.

RT művelet:  $done \leftarrow Controller\_done(0) \& \dots \& Controller\_done(n - 1)$

##### 4.5.2. Adatfüggőségek identifikálása

Mivel nagyon egyszerű RT műveletekkel meg lehet oldani az adott feladatot, nem merülnek fel adatfüggőségek.

### 4.5.3. Célregiszterek azonosítása

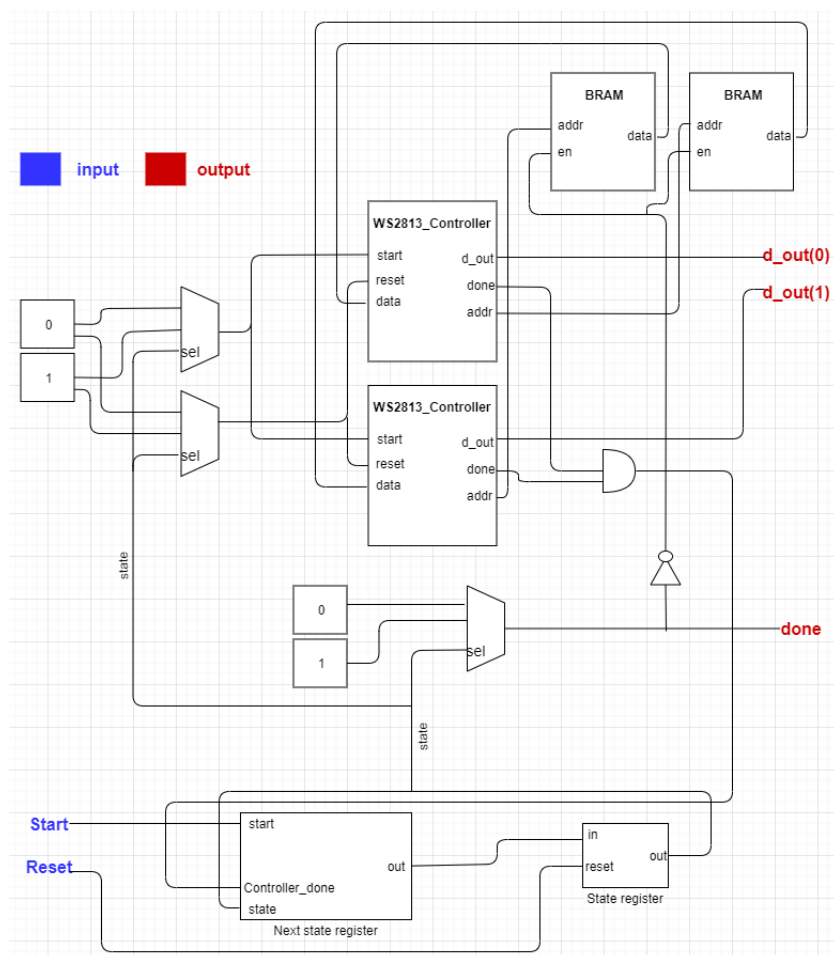
Nincs szükség külön célregiszterekre a feladat megoldása érdekében.

### 4.5.4. Különböző fázisokban elvégzendő műveletek

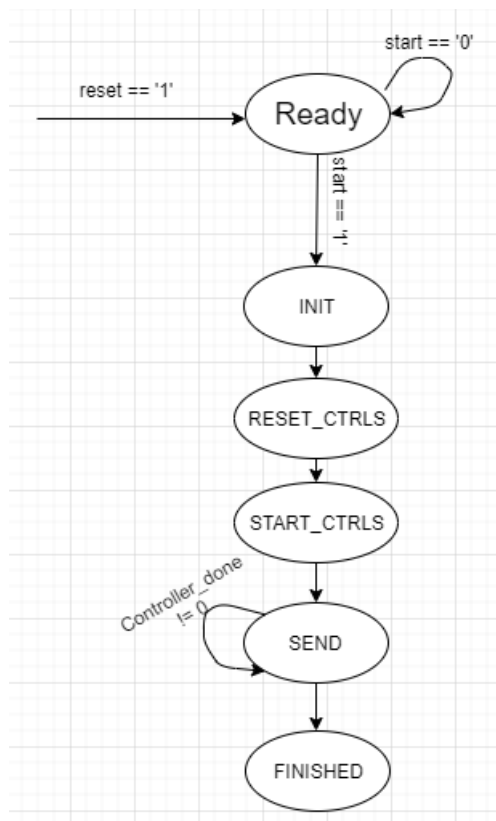
3. táblázat. WS2813\_TopLevel - Különböző fázisokban elvégzendő műveletek

Állapot	$Controller_{reset}$	$Controller_{start}$	$BRAM_{enable}$	done
READY	'0'	'0'	'0'	'0'
INIT	'0'	'0'	'1'	'0'
RESET_CTRL	'1'	'0'	$BRAM_{enable}$	'0'
START_CTRL	'0'	'1'	$BRAM_{enable}$	'0'
SEND	'0'	'0'	$BRAM_{enable}$	'0'
FINISHED	'0'	'0'	'0'	'1'

### 4.5.5. Kapcsolási rajz

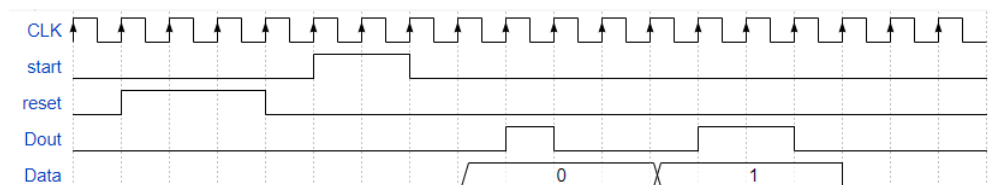


#### 4.5.6. Állapotdiagram



#### 4.6. Alegységek

#### 4.7. Idődiagram



- CLK: 100 MHz-es órajel
- start: jel a folyamat elindításához
- reset: jel a folyamat resetálásához
- Dout: Egyszálú adatsín a LED-ekre.
- Data: Kírandó adat

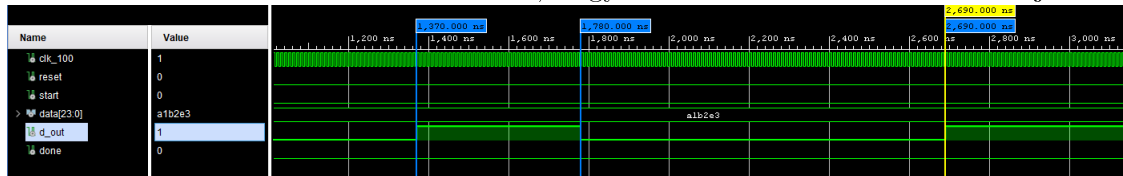
### 5. Szimuláció eredménye

#### 5.1. WS2813\_Driver

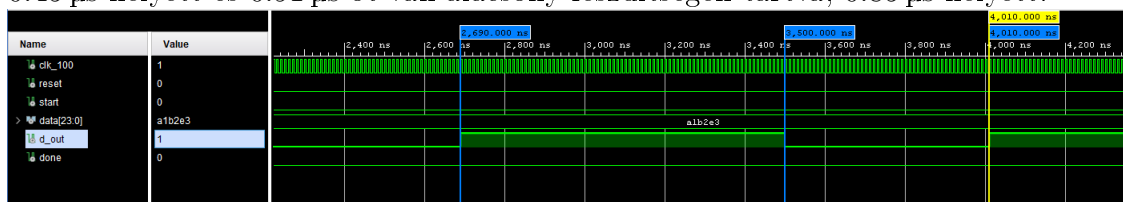
A **WS2813\_Driver** modul szimulációja alatt a következőkre figyeltem:

- WS2813 egyszálú protokollja be van-e tartva
- A 24 bit-es blokkot sikerült-e legalább 80  $\mu$ s alatt elküldeni (egy bitet elküldeni 1.25  $\mu$ s, 24 bit-es blokk elküldése után min. 50  $\mu$ s-ot kell várakozni)  $1.25 * 24 + 50 = 80$  ( $\mu$ s)
- A küldés befejezése után a **done** jel 1-esre lett-e állítva

A szimuláció során észrevevődött, hogy már az első kritérium nem teljesül:



Látható, hogy nullás küldése esetén a kimenet 0.41  $\mu$ s-ot van magas feszültségen tartva 0.45  $\mu$ s helyett és 0.91  $\mu$ s-ot van alacsony feszültségen tartva, 0.85  $\mu$ s helyett.



Itt látható, hogy egyes küldése esetén a kimenet 0.81  $\mu$ s-ot van magas feszültségen tartva 0.8  $\mu$ s helyett és 0.51  $\mu$ s-ot van alacsony feszültségen tartva, 0.45  $\mu$ s helyett.

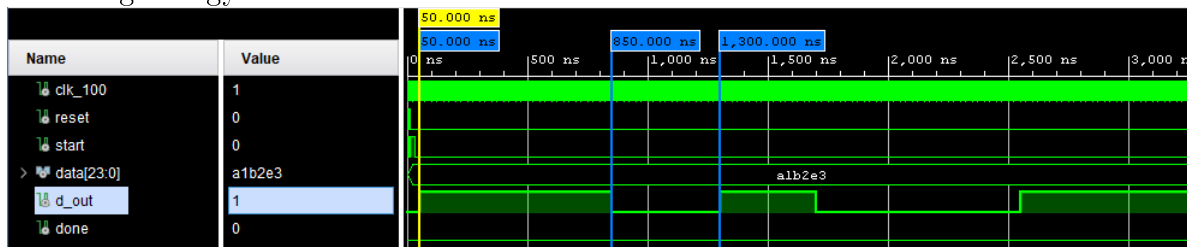
Ezt egyszerűen meg lehet oldani a ciklusszámok módosítása által.

Módosított ciklusszámok:

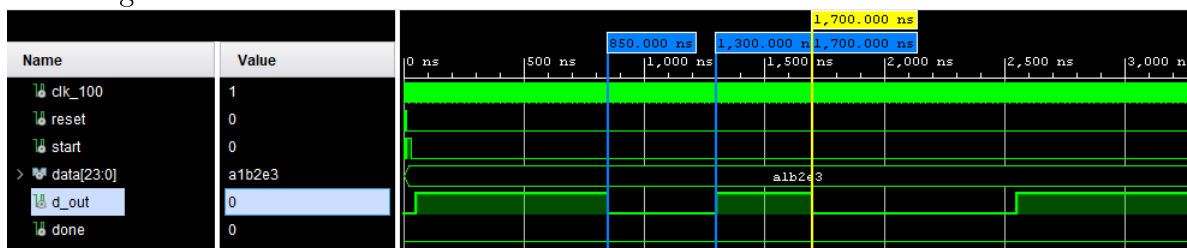
- Logikai 1-es
  - 79 ciklus magas feszültségen
  - 39 ciklus alacsony feszültségen
- Logikai 0-ás
  - 39 ciklus magas feszültségen
  - 79 ciklus alacsony feszültségen

A frissített értékekkel a WS2813 protokollja már pontosan be van tartva.

Logikai egyesre:

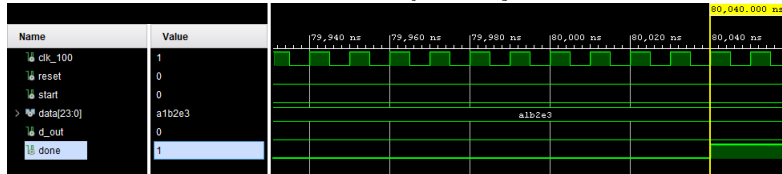


Logikai nullásra:





A másik két követelmény is teljesül:



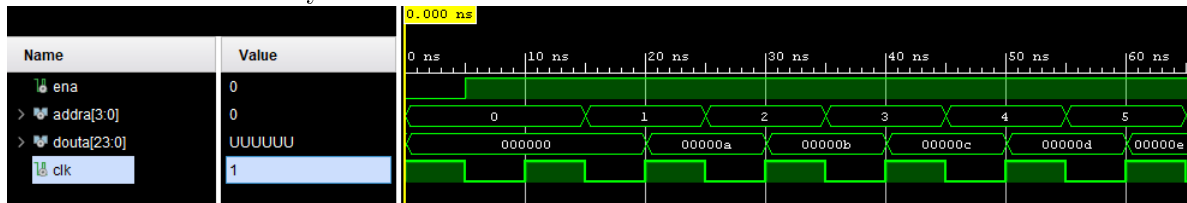
A 24 bit-es blokk elküldése után a done jel egyesre lett állítva és 80.04  $\mu$ s alatt sikerült az egész 24 bit-es blokkot elküldeni.

## 5.2. BRAM memória

A BRAM memóriák IP Catalog-ból lettek kigenerálva[4]. Kezdőértékek egy **coefficient** file-ból lettek feltöltve[5]. A **coefficient** file:

```
memory_initialization_radix=16;
memory_initialization_vector=00A,00B,00C,00D,
00E,00F,010,011,012,013,014,015,016,017,018,019;
```

A szimuláció eredménye:



Megfigyelésem alapján a generált BRAM modul felfutó órajelre olvassa be a következő címet, és ugyanúgy felfutó órajelre teszi ki az adatot a kimenetre, de csak egy egész órajellel a cím beolvasása után.

## 6. Működés közbeni tesztelés - Integrated Logic Analyser

Működés közbeni tesztelésre az FPGA-ba beépített ILA (Integrated Logic Analyser) volt felhasználva. A tesztelés beállításához a TCL script-es megoldás[6] volt felhasználva.

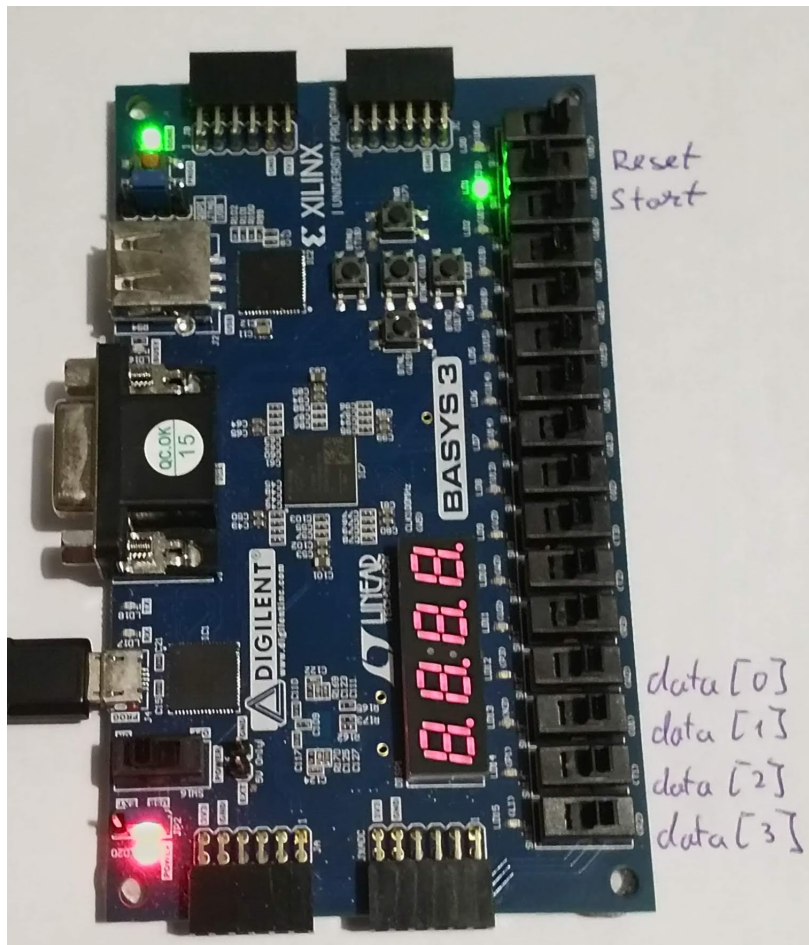
### 6.1. WS2813\_Driver

Ennél a modulnál szükség van egy 24 bit-es bementre. Mivel annyi kapcsoló nem található a választott FPGA lapon, ezért a modul le lett egyszerűsítve, a tesztelés megkönnyebítéséért. 24 bemenet helyett csak 4 lett felhasználva, ezek be is lettek konfigurálva az fpga négy kapcsolójára.

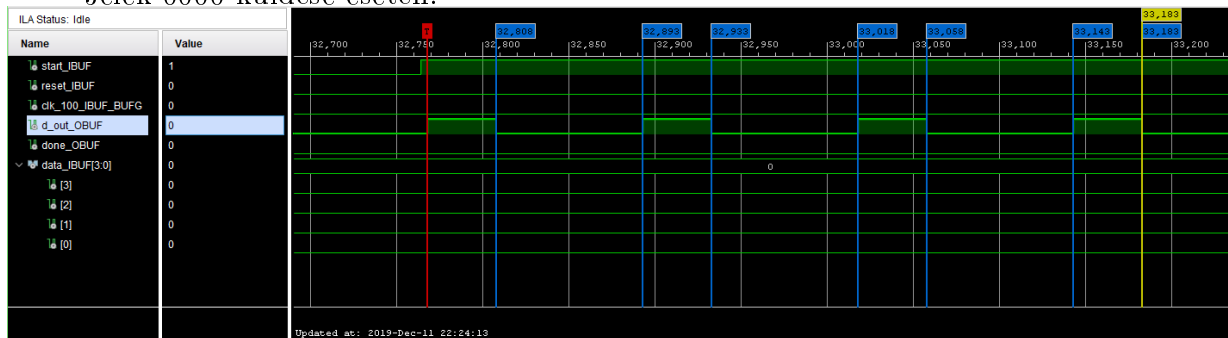
Ezen kívül a done és a d\_out jelek egy-egy LED-re lettek kötve. A d\_out jelnek ledre való kötése később értelmetlennek bizonyult, mivel olyan gyors a váltakozás magas feszültséggről alacsony feszültségre a küldés során, hogy a LED fel sem gyúl.

A rendszer működés közbeni teszteléséhez szükséges volt egy trigger definiálásához: amikor a d\_out jel 1-esre vált, akkor kezdődjön a mintavételezés. Ez lehetővé teszi, hogy a küldés kezdetétől legyen a mintavételezés.

FPGA állapota 0000 küldése esetén:

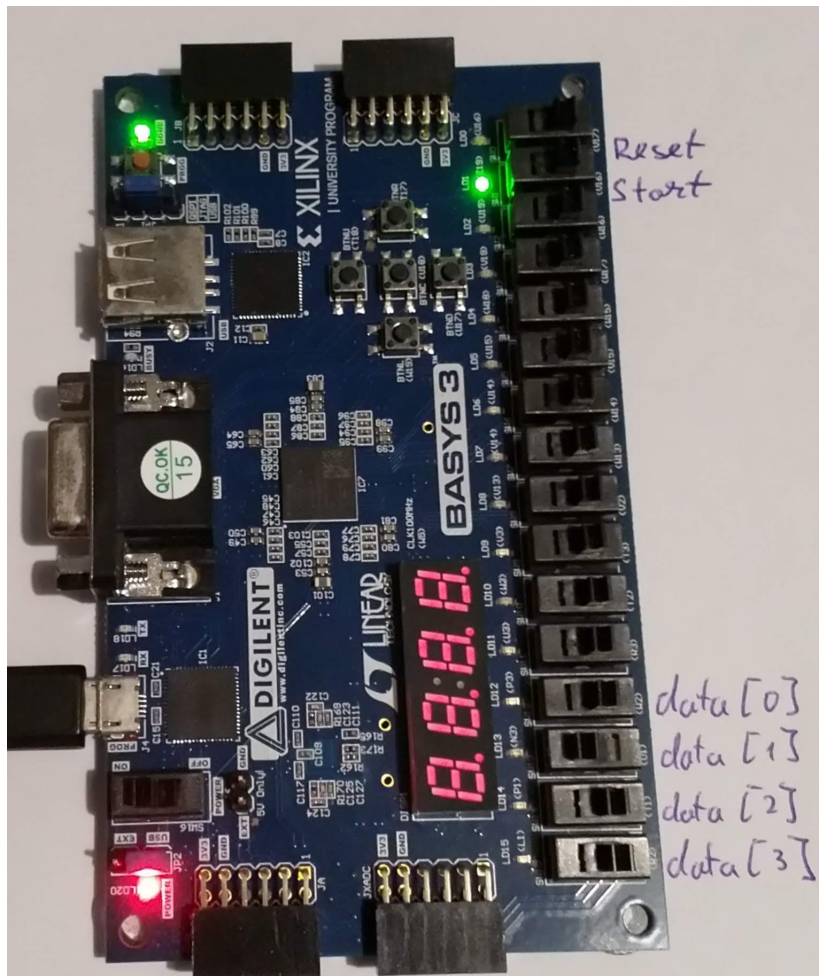


Jelek 0000 küldése esetén:

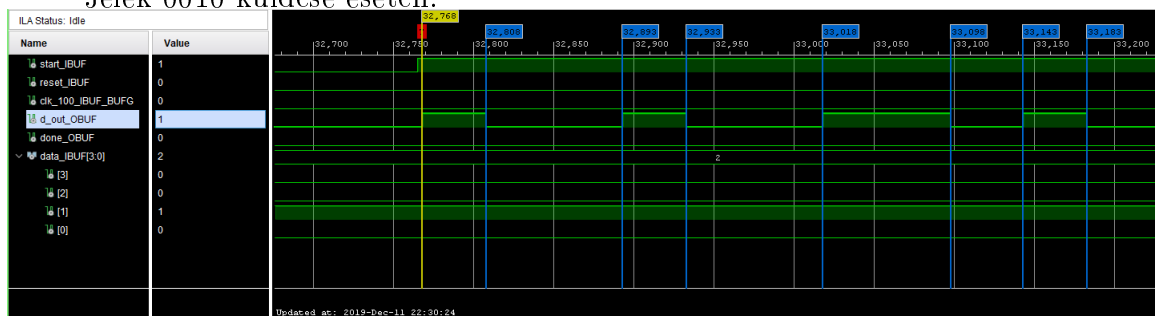


Amint az FPGA-ról és a jelanalizálása közben is látható, az elküldött adat 0000. Ugyanakkor az is látható, hogy a d\_out jel annyi időt van magas és alacsony feszültségen, amennyit a protokoll megkövetel. Ezen a példán nem látszik, hogy a küldés az MSB-től (Most Significant Bit) lenne elkezdve. Ennek demonstrálásához egy "asszimmetrikus" adat szükséges, mint például: 0010

FPGA állapota 0010 küldése esetén:



Jelek 0010 küldése esetén:



Látható, hogy a küldés az MSB-től van elkezdve. Ugyanakkor az is megfigyelhető, hogy a WS2813 protokollja be van tartva.

## 6.2. Teljes rendszer működés közbeni tesztelése

TODO

## Hivatkozások

- [1] P. Burgess, „Adafruit neopixel Überguide,” pp. 97–99, 2019.
- [2] Shen JinGuo, Yin HuaPing, *WS2813 Intelligent control LED integrated light source*.
- [3] Brassai Sándor Tihamér, *Újrakonfigurálható Digitális Áramkörök Tervezési És Tesztelési Módszerei*, pp. 125–143. Scientia, 2018.
- [4] Vipin from vhdlguru.blogspot.com, „Design and simulation of bram using xilinx core generator,” 2010.
- [5] Vipin from vhdlguru.blogspot.com, „How to use coe file for initializing bram,” 2010.
- [6] Brassai Sándor Tihamér, *Újrakonfigurálható Digitális Áramkörök Tervezési És Tesztelési Módszerei*, pp. 110–124. Scientia, 2018.

# A. Tervezési lépések

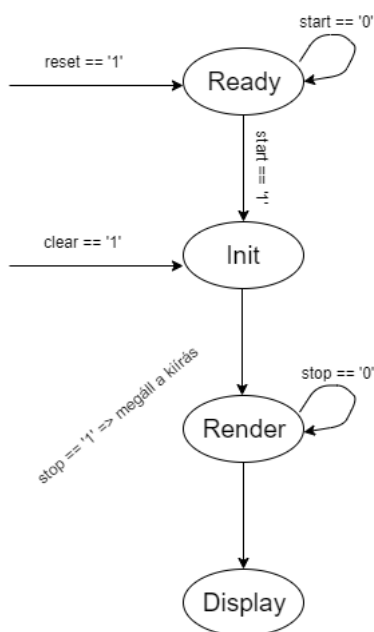
## A.1. Állapotok

### A.1.1. Első iteráció - 2019.10.14

*Ebben a fázisban még nem volt meg a megfelelő tudás a véges állapotú adatútas automatáról. Itt a tervezés elnagyolva volt elvégezve, még nem volt kellőképpen átlátva a megoldandó probléma. A négy kontrolljel közül (start, reset, clear, stop) csak kettő lett végül felhasználva: start és reset.*

Állapotok:

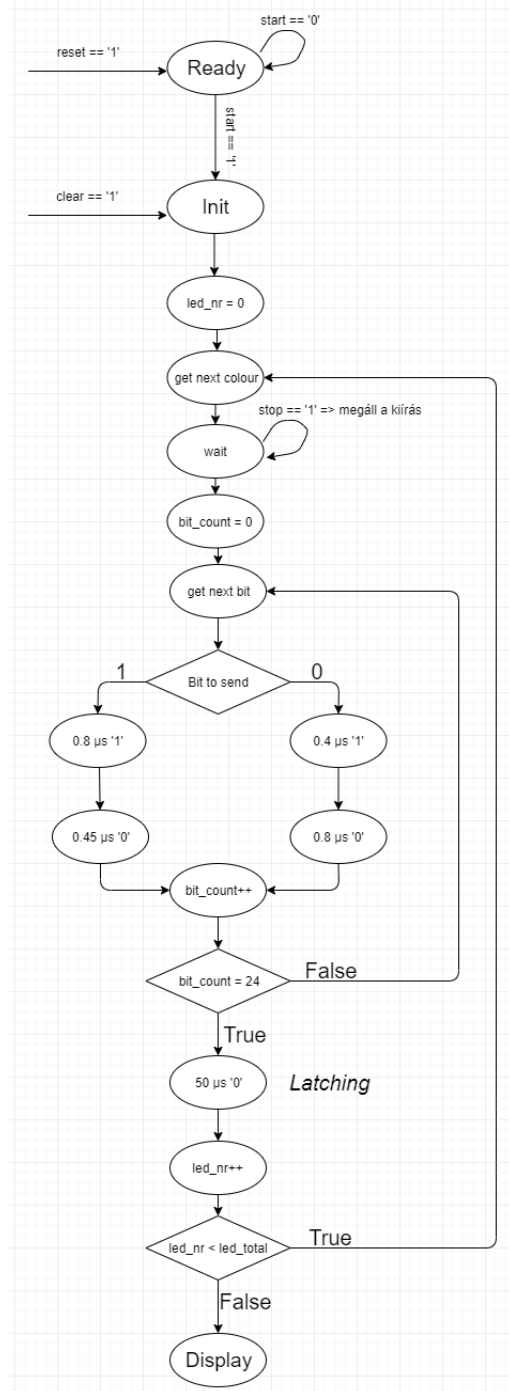
- READY
  - Alap állapot
  - "reset" jel esetén ide kerül vissza az automata
- INIT
  - minden LED-et kikapcsol (0x000000-t ír)
  - "clear" jel esetén ide kerül az automata
- RENDER
  - egyenként küldi a szín információt a LED-ekre
  - annyiszor végződik el itt a művelet, ahány LED-ünk van
  - "stop" jel esetén megáll a kiírás
- DISPLAY
- megtörtént a kiírás



### A.1.2. Második iteráció - 2019.10.28

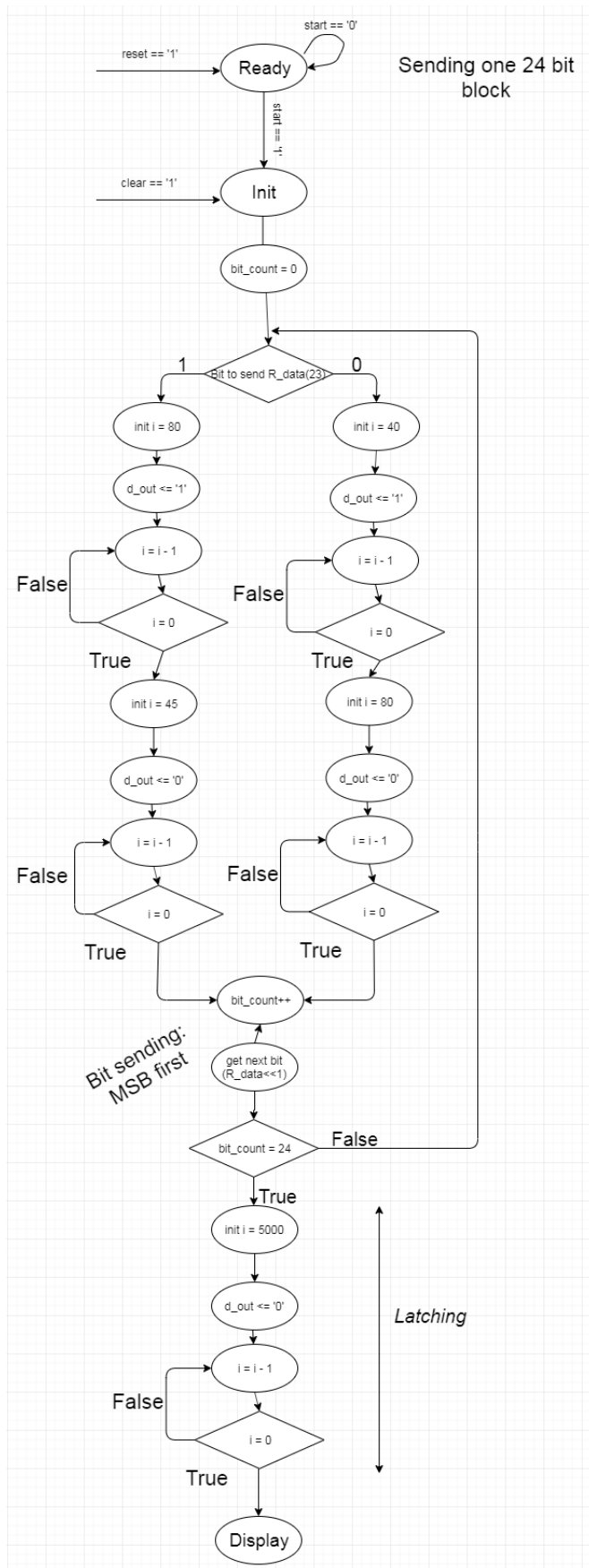
Ebben a fázisban kezdett kialakulni a szükséges tudás a feladat megoldásához, és a feladat is már jobban át volt látva. Itt már egy nagy modul helyett, két kisebb modulra van kigondolva a feladat megoldása.

Állapotdiagram átírva úgy, hogy a küldési logikát is tartalmazza:



Egy kisebb finomítás az állapotdiagramon. Itt már le van bontva részletesen az diagram RT műveletekre.

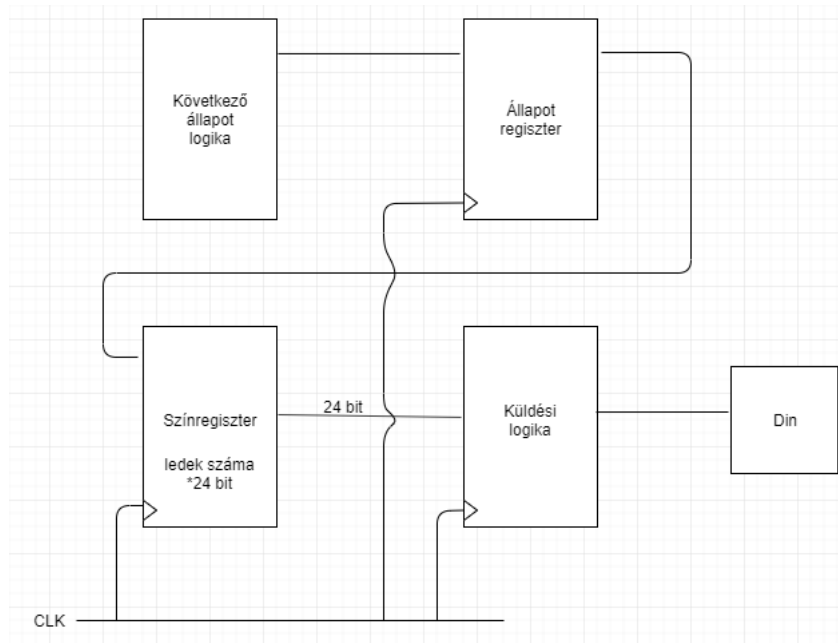
Állapotdiagram egy 90 LED-et vezérlő modulra



## A.2. Alegységek

### A.2.1. Első iteráció - 2019.10.14

- Következő állapot regiszter *Next State Register*
- Állapot regiszter *State Register*
- Szín regiszter *Colour Register*
- Küldési logika regiszter *Transmission Logic Register*

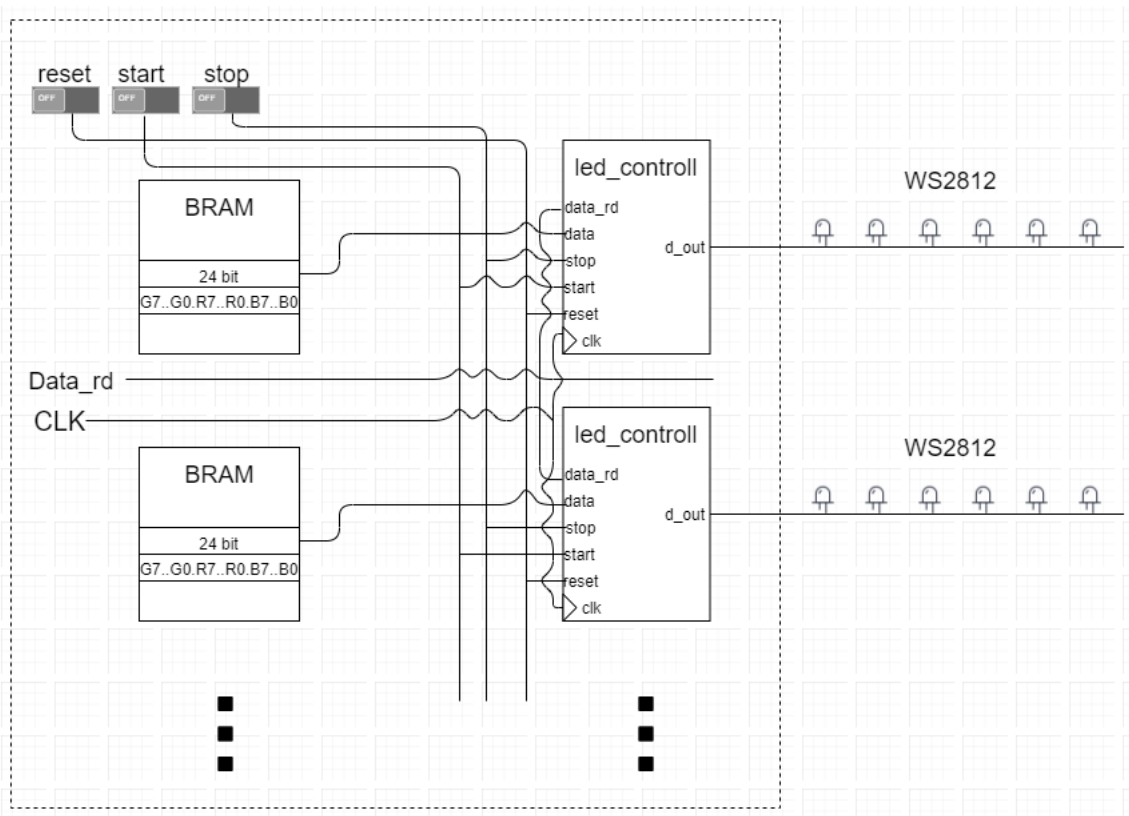


#### Küldési logika regiszter

A küldési logika modul részletesebb lebontása:



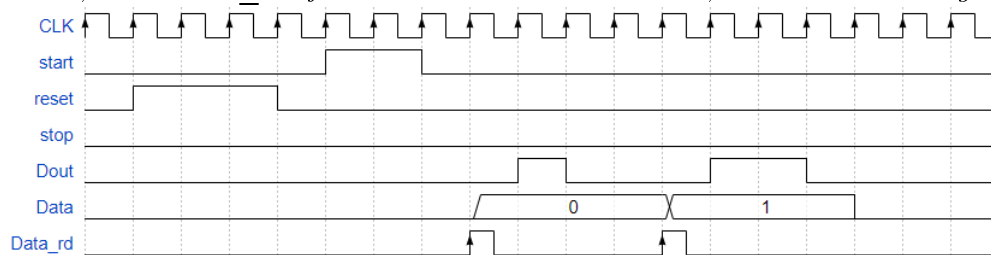




## A.3. Idődiagram

### A.3.1. Első iteráció - 2019.10.26

*Ebben a fázisban az volt az elgondolás, hogy a modul akkor olvassa be az adatot a **data** sínről, ha a **data\_rd** jel 1-es. Mint később kiderült, erre nincs szükség.*



- CLK: 100 MHz-es órajel
- start: jel a folyamat elindításához
- reset: jel a folyamat resetálásához
- stop: jel a kiírás megállításához. Csak két 24 bit-es blokk kiírása közben tudja megállítani a kiírást
- Dout: Egyszálú adatsín a LED-ekre.
- Data: Kiírandó adat, Data\_rd felmenő órajelére olvassa be az adatot.
- Data\_rd: Aktiváló bit az adat beolvasására