

# Programozható LED-fűzéren alapuló reklámpanel

## LED fűzér vezérlése: adatok kiírása

Patka Zsolt-András  
Számítástechnika BSc

2019.12.16

# Tartalomjegyzék

<b>1. Bevezető</b>	<b>3</b>
<b>2. Követelmények</b>	<b>3</b>
2.1. Funkcionális követelmények . . . . .	3
2.2. Nem funkcionális követelmények . . . . .	3
2.3. Fejlesztési követelmények . . . . .	3
<b>3. Rendszer-specifikáció</b>	<b>4</b>
<b>4. Tervezés</b>	<b>4</b>
4.1. WS2813 egyszálú adatátvitel protokoll leírása . . . . .	4
4.1.1. A 24 bit-es kód . . . . .	4
4.1.2. Bit-ek küldési sorrendje . . . . .	4
4.1.3. Időzítések . . . . .	4
4.2. Véges állapotú adatútas automata . . . . .	5
4.3. WS2813_Driver tervezése . . . . .	5
4.3.1. Elvégezendő RT műveletek azonosítása . . . . .	5
4.3.2. Adatfüggőségek identifikálása . . . . .	5
4.3.3. Célregiszterek azonosítása . . . . .	6
4.3.4. Különböző fázisokban elvégzendő műveletek . . . . .	6
4.3.5. Kapcsolási rajz . . . . .	6
4.4. Állapotok . . . . .	6
4.5. Alegységek . . . . .	12
4.5.1. Küldési logika regiszter . . . . .	12
4.6. Idődiagram . . . . .	14
<b>5. Szimuláció eredménye</b>	<b>14</b>
5.1. WS2813_Driver . . . . .	14
5.2. BRAM memória . . . . .	16
<b>6. Működés közbeni tesztelés - Integrated Logic Analyser</b>	<b>16</b>
6.1. WS2813_Driver . . . . .	16
6.2. Teljes rendszer működés közbeni tesztelése . . . . .	18
<b>A. tervezési lépések</b>	<b>19</b>

# 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
  - Külön modul egy 24 bit-es blokk küldésére (WS2813\_Driver)
  - Külön modul a WS2813\_Driver modul BRAM-al való összekötésére (Controller)
  - Top level modul a Controllerek összekötésére a BRAM-okkal
- 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 képezi. 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 az [alábbi](#) útmutatóból lettek kivéve. Az útmutató szerint az adatlapban levő értékek rosszul vannak kiszámolva.

Ha az útmutatóban megadott értékek nem megfelelő működéshez vezetnek, akkor az adatlapban 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[1] 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

### 4.3.1. Elvégezendő 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 feszültségszinten, 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 bitek 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.
- $R_{bit\_count}$ : Biteket számláló regiszter, a 24 bit-es blokk elküldésének érzékeléséhez.
- $R_{data}$ : Elküldendő adatot tartalmazó regiszter

### 4.3.4. Különböző fázisokban elvégzendő műveletek

1. táblázat. 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

TODO

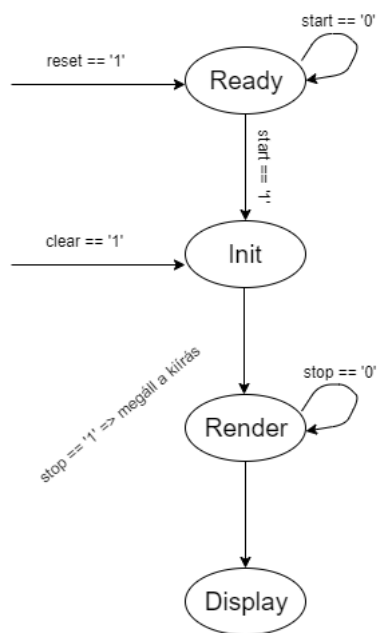
## 4.4. Állapotok

2019.10.14

Állapotok:

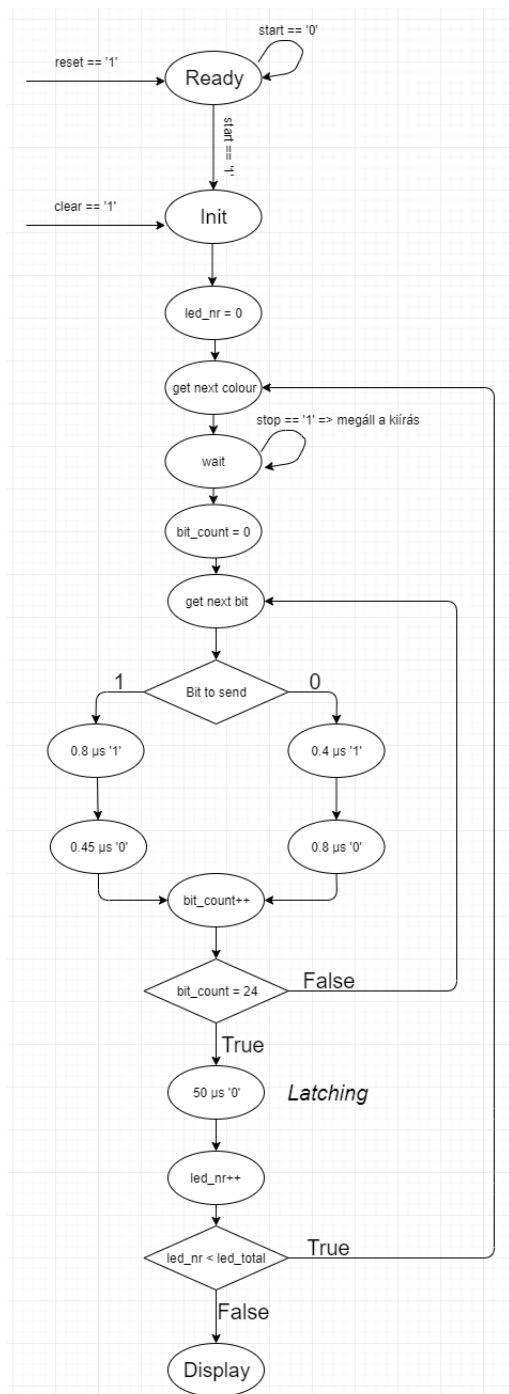
- READY
  - Alap állapot
  - "reset" jel esetén ide kerül vissza az automata
- INIT

- 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



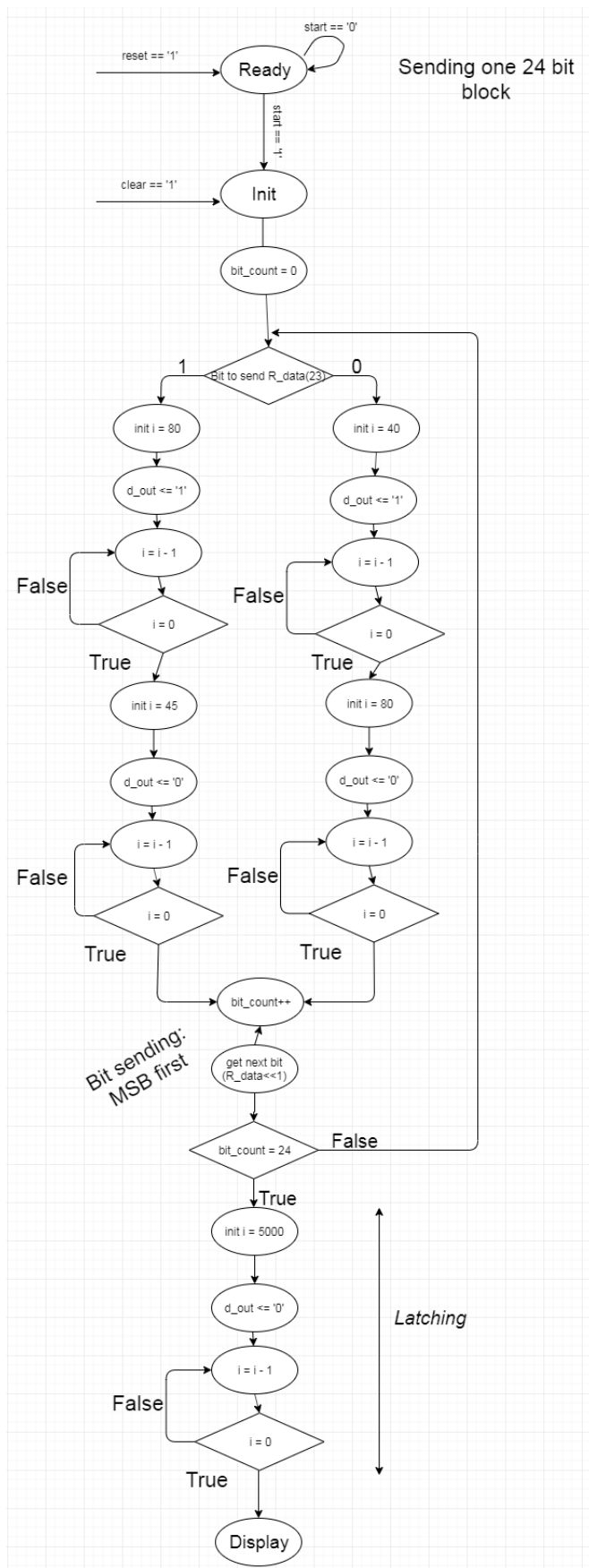
**2019.10.28**

Állapotdiagram átírva úgy, hogy a küldési logikát is tartalmazza:



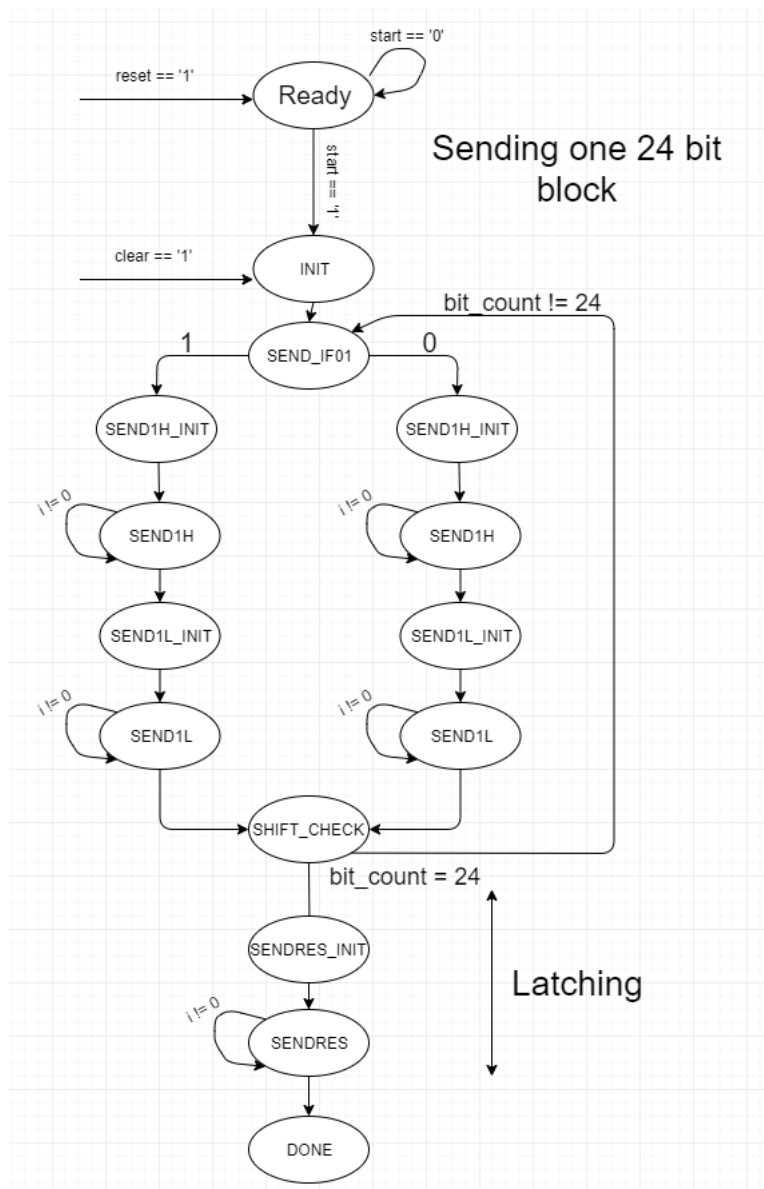
Állapotdiagram egy 100 LED-et vezérlő modulra





2019.11.11

Végleges állapotdiagram, az implementációban is használt.



- **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

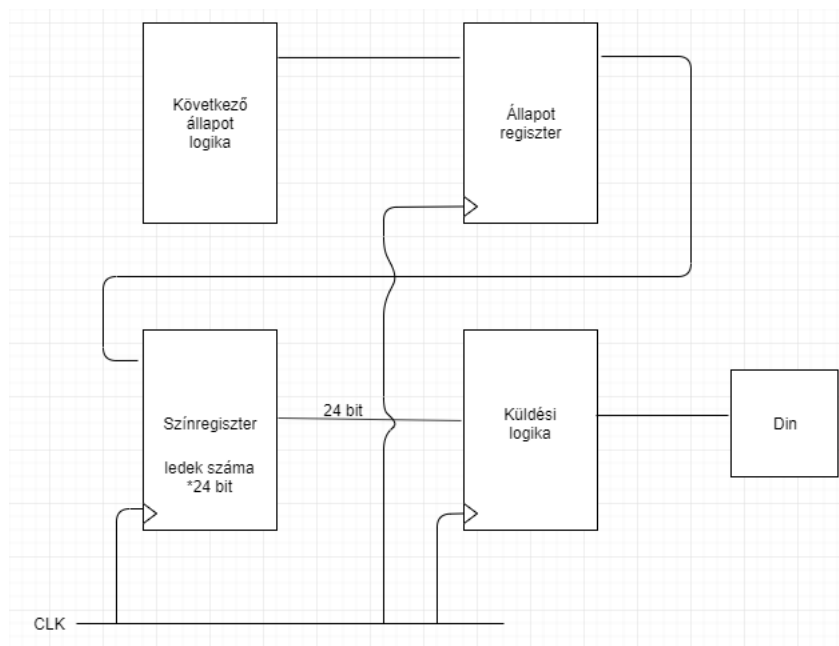
- SEND1H
  - Dekrementálja az **i** változót
  - Amikor az **i** változó nulla lesz, akkor tovább megy a **SEND1L\_INIT** állapotra
- SEND1L\_INIT
  - Inicializálja az **i** változót a T1L értékre
  - A **d\_out** output-ot 0-ra állítja
- SEND1L
  - Dekrementálja az **i** változót
  - Amikor az **i** 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

## 4.5. Alegységek

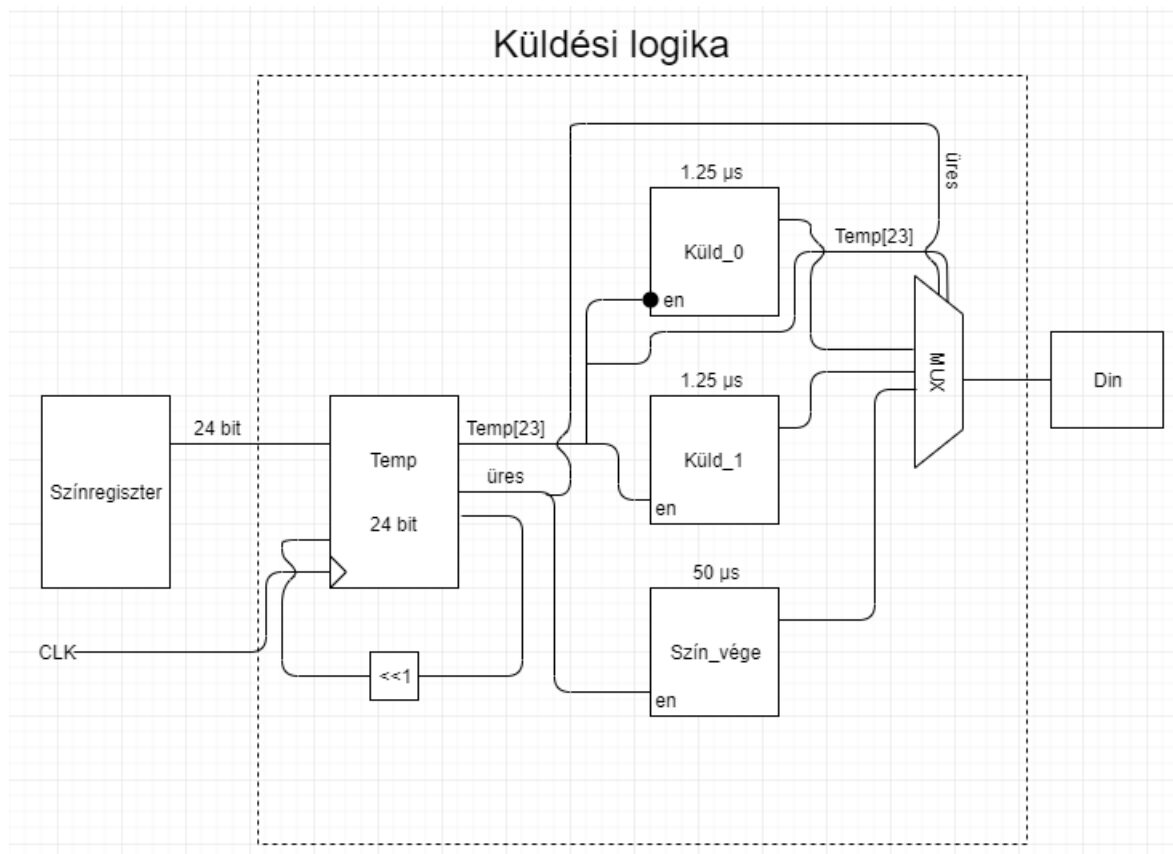
2019.10.14

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



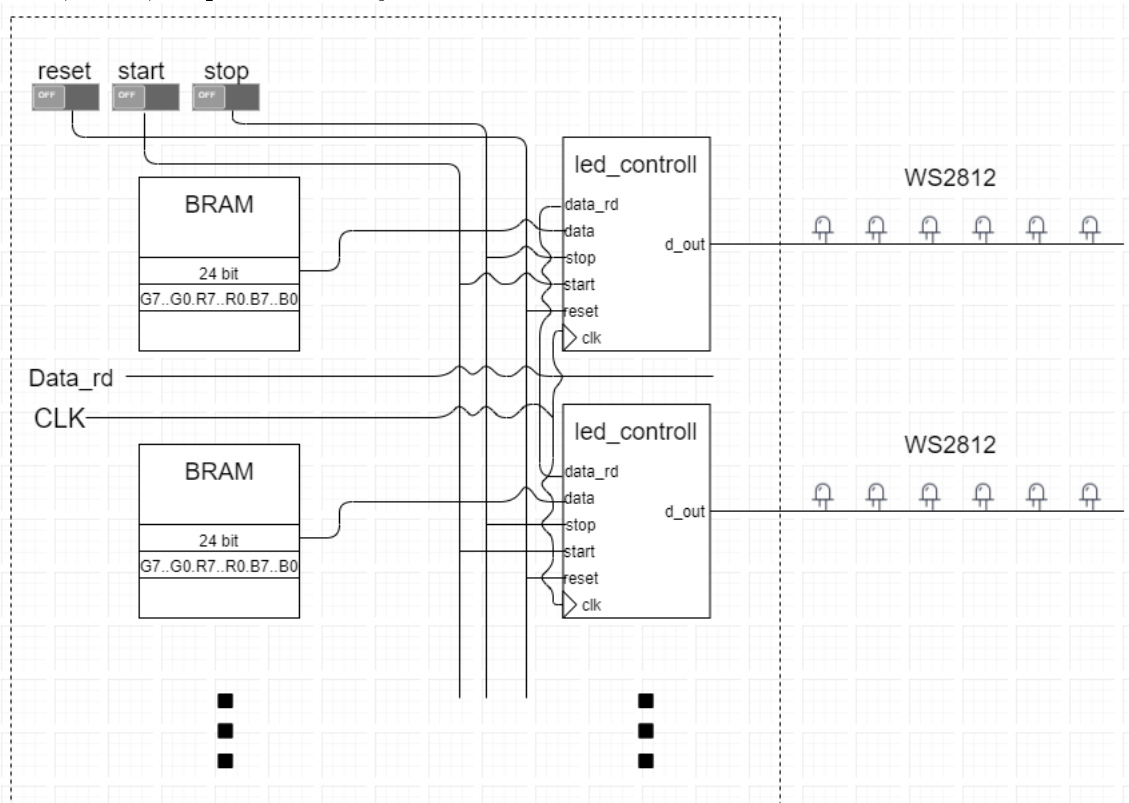
### 4.5.1. Küldési logika regiszter

A küldési logika modul részletesebb lebontása:

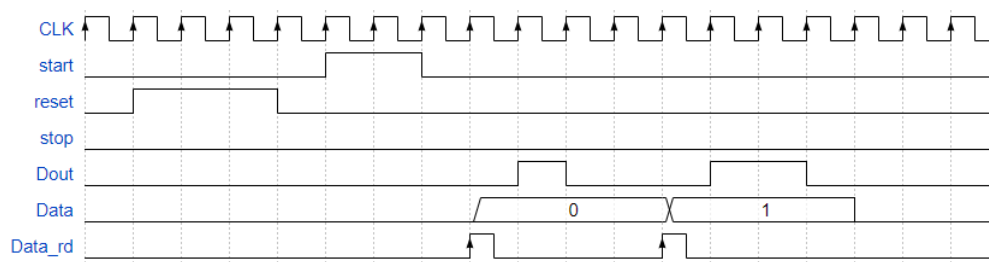


**2019.10.28**

Minden **led\_controll** modulhoz tartozik egy BRAM blokk és minden ilyen modul egy ledfűzért vezérel meg. Öt ilyen blokk megvezérel öt ledfűzért, ezáltal létrehozva a ledmátrixot. Az órajel, start, reset, stop és data-rd jelek közősek minden modulnak.

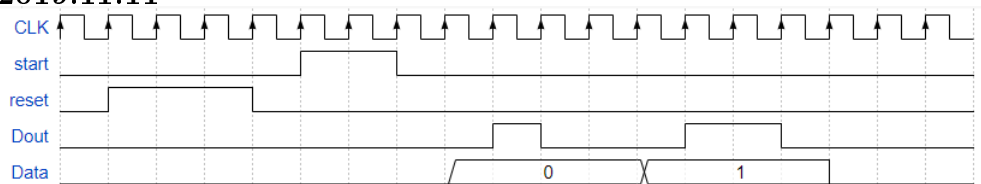


## 4.6. Idődiagram



- 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

2019.11.11



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

## 5. Szimuláció eredménye

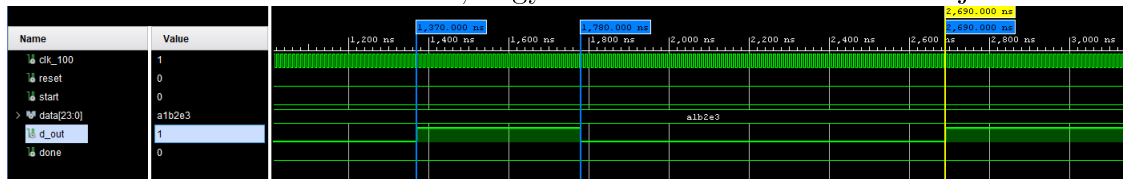
### 5.1. WS2813\_Driver

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

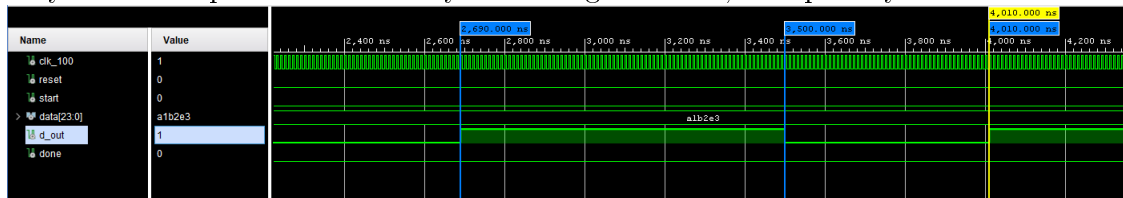
- WS2813 egyszálú protokollja be van-e tartva
- A 24 bit-es blokkot sikerült-e legalább 80  $\mu$ s alatt elküldeni (egy bitet elküldeni 1.25  $\mu$ s, 24 bit-es blokk elküldése után min. 50  $\mu$ s-ot kell várakozni)  $1.25 * 24 + 50 = 80$  ( $\mu$ s)

- 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.

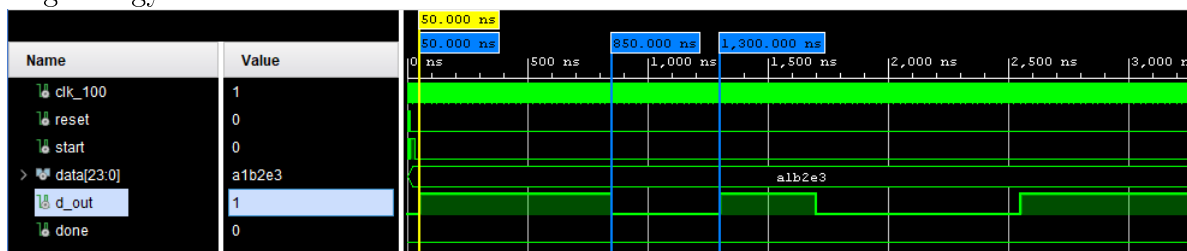
Ezt egyszerűen meg lehet oldani a ciklusszámok módosítása által.

Módosított ciklusszámok:

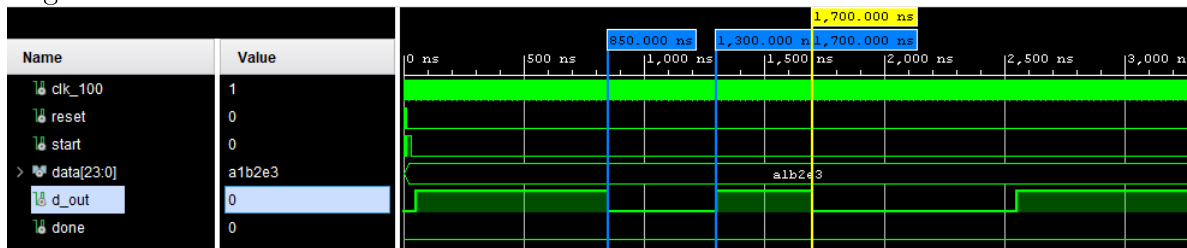
- Logikai 1-es
  - 79 ciklus magas feszültségen
  - 39 ciklus alacsony feszültségen
- Logikai 0-ás
  - 39 ciklus magas feszültségen
  - 79 ciklus alacsony feszültségen

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

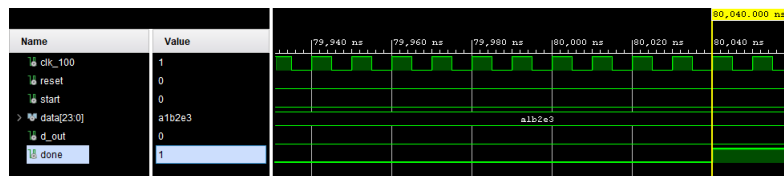
Logikai egyesre:



Logikai nullásra:



A másik két követelmény is teljesül:



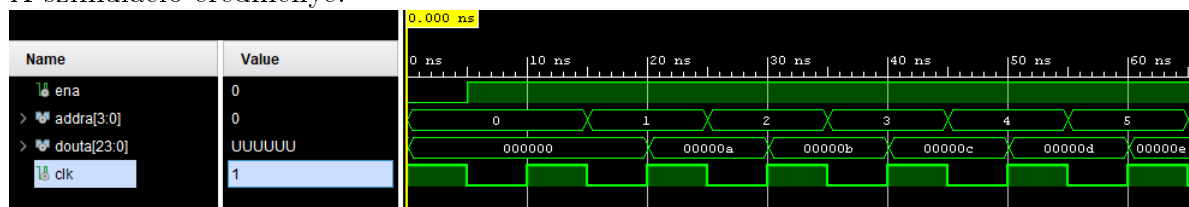
A 24 bit-es blokk elküldése után a done jel egyesre lett állítva és 80.04  $\mu$ s alatt sikerült az egész 24 bit-es blokkot elküldeni.

## 5.2. BRAM memória

A BRAM memóriának a használatát a következő két blogbejegyzés alapján tanultam meg: [bram használata és tesztelése](#) és [BRAM feltöltése coefficient file használata által](#). A második blogbejegyzés alapján tanultam meg a BRAM memóriának a **coefficient** file alapján való feltöltését. A BRAM memória szimulációja alatt azt értem, hogy megvizsgálom, hogy a BRAM memória helyesen fel lett töltve a **coefficient** file-ból. A **coefficient** file:

```
memory_initialization_radix=16;
memory_initialization_vector=00A,00B,00C,00D,00E,00F,010,011,012,013,014,015,016,017,0
```

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.

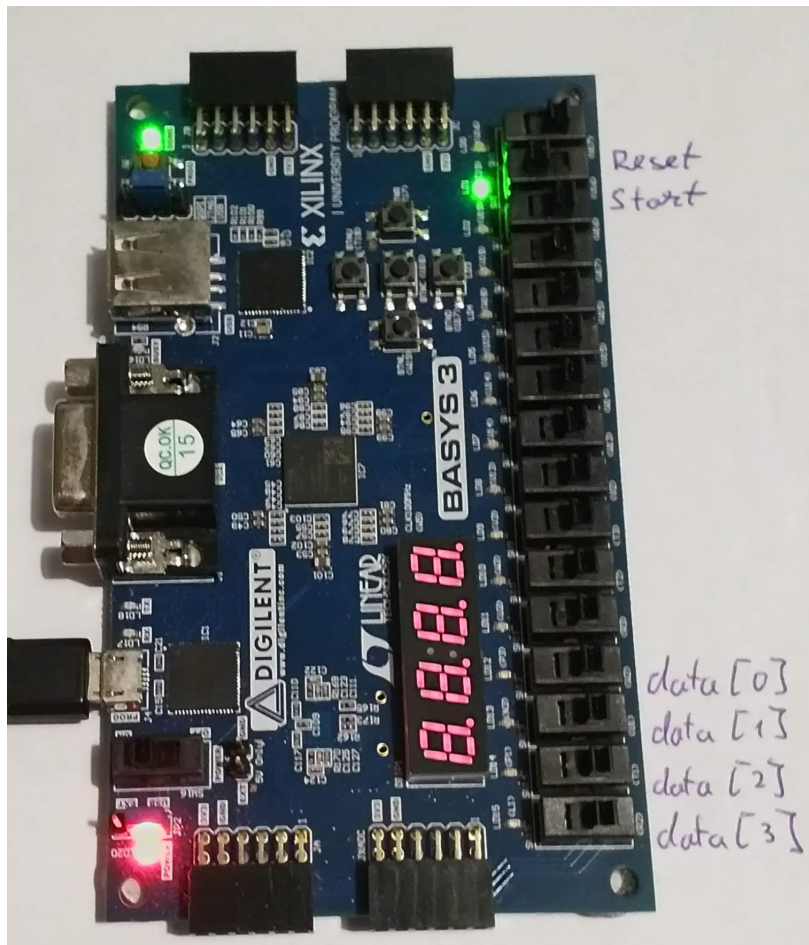
## 6. Működés közbeni tesztelés - Integrated Logic Analyser

Működés közbeni tesztelésre az FPGA-ba beépített ILA (Integrated Logic Analyser) volt felhasználva.

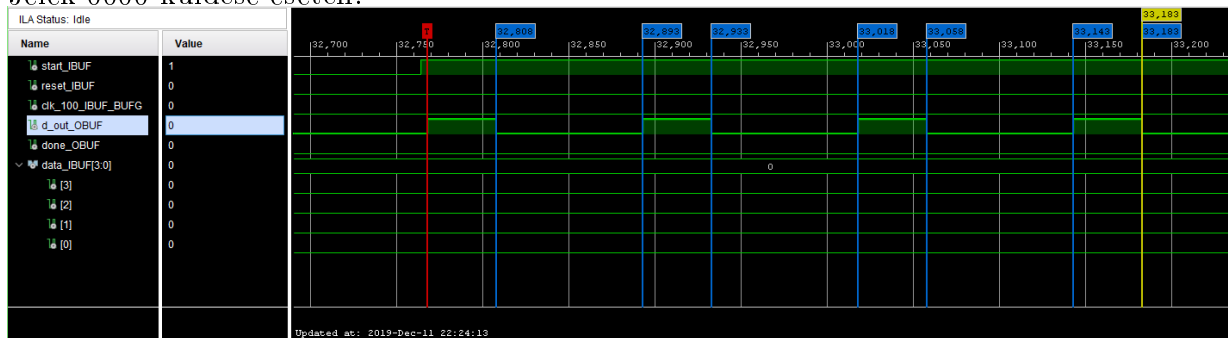
### 6.1. WS2813\_Driver

Ennél a modulnál szükség van egy 24 bit-es bemenetre. 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 feszültségéről 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:

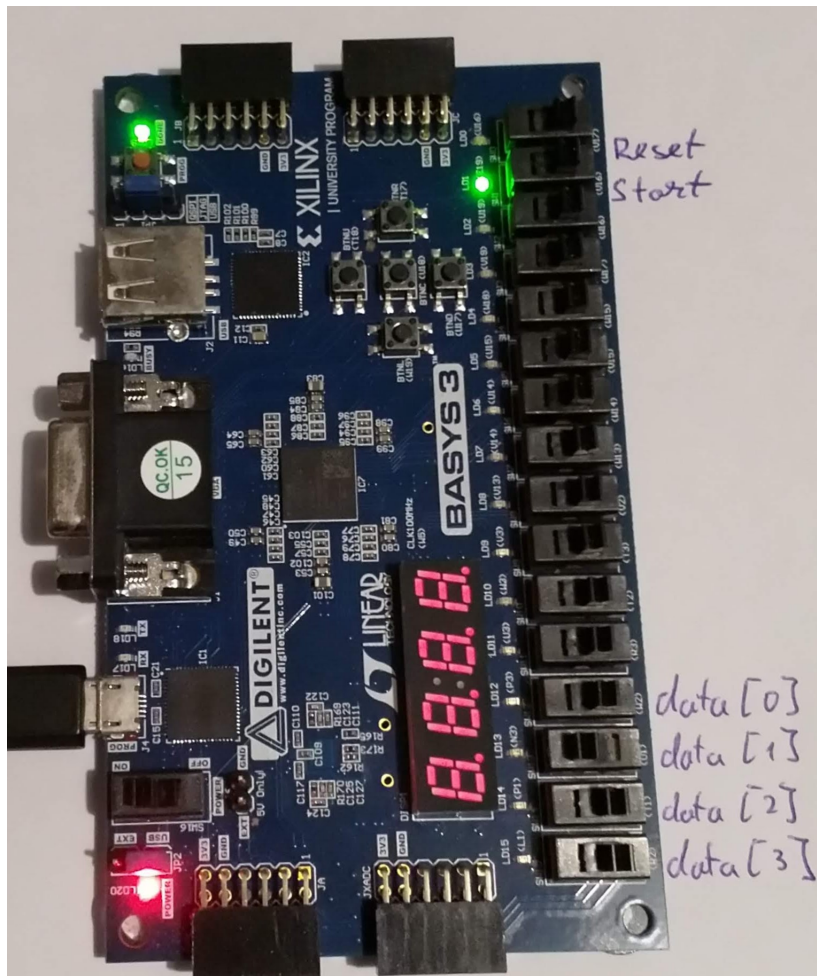




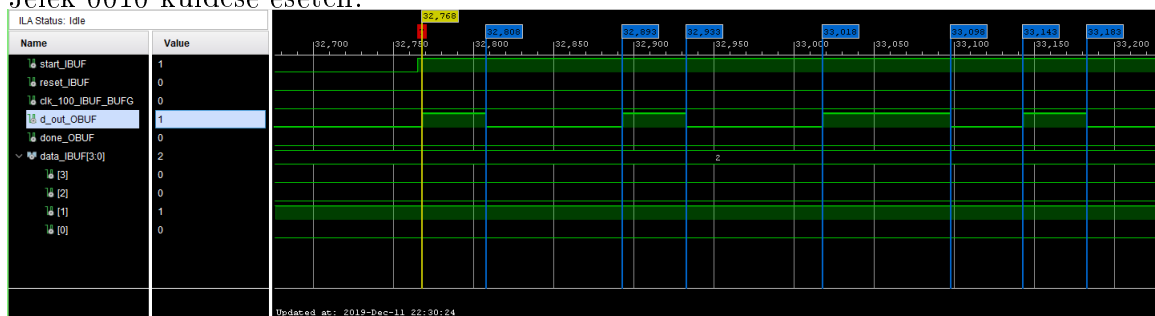
Jelek 0000 küldése esetén:



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 "asszimétrikus" 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] S. T. Brassai, *Újrakonfigurálható Digitális Áramkörök Tervezési És Tesztelési Módszerei*, pp. 125–143. Scientia, 2018.

## A. tervezési lépések

TODO