

Programozható LED-fűzőren alapuló reklámpanel - LED fűzőr vezérlése, adatok kiírása

Patka Zsolt-András | Számítástechnika BSc

2019.11.11

1. Bevezető

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

2. Követelmények

2.1. Funkcionális követelmények

- Lehetséges legyen egy reklámszöveget kiírni a ledfűzőrek által létrehozott mátrix-ra.
- Egy sorban minimum 100 LED található
- Az egyes ledfűzőrek szinkronba működjenek
- A kiírás párhuzamosan történik az adott ledfűzőreken
- 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 fűzőrre
- Modularitás
 - Külön modul egy 24 bit-es blokk küldésére (WS2813_Driver)
 - Külön modul egy 24 bit-es blokkot küldő modul BRAM-al való összekötésére
- Opcionális:
 - Pár betű kódolása (3-4)
 - Betűk tárolása BRAM memóriában

3. Rendszer-specifikáció

- WS2813 egyszálú adatátvitel protokolljának helyes használata
- Worldsemi WS2813 100 LED-es ledfűzőr van felhasználva
- Digilent Basys3 FPGA vezérli a ledfűzőrt

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 tudjunk megjeleníteni a LED-fűzőren.

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

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 történnjen:

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:

- **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 μ 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 μ s-ot magas feszültségen
 - 0.45 μ s-ot alacson feszültségen
- Logikai 0-ás
 - 0.4 μ s-ot magas feszültségen
 - 0.85 μ s-ot alacson feszültségen
- 24 bit-es adatblokk küldése után:
 - > 50 μ s-ot alacsony feszültségen

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

A várakozási értékeket nem az adatlapból, hanem az alábbi útmutatóból vettem. Az útmutató szerint az adatlapban levő értékek rosszul vannak kiszámolva.

Egyelőre megpróbálok az útmutatóban megadott értékekkel dolgozni. Ha ez nem megfelelő működéshez vezet, akkor veszem az adatlapban levő értékeket.

4.2. Véges állapotú adatútas automata tervezése

A feladatot egy véges állapotú adatútas automatával fogom megoldani. Ennek megfelelően mutatom be a tervezést.

4.2.1. Elvégzendő RT műveletek azonosítása

Az időzítések implementálásához egy számláló lesz használva. A 100 MHz-es órajel ami fel lesz használva időben 0.01 μ 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: $data \leftarrow data << 1$

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.

4.2.2. Adatfüggőségek identifikálása

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

4.2.3. 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

4.2.4. 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

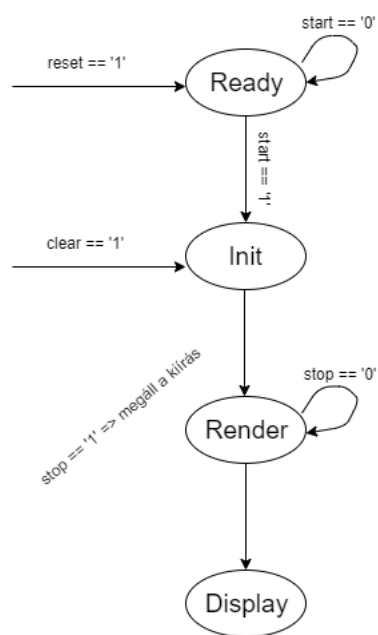
4.3. Állapotok

2019.10.14

Állapotok:

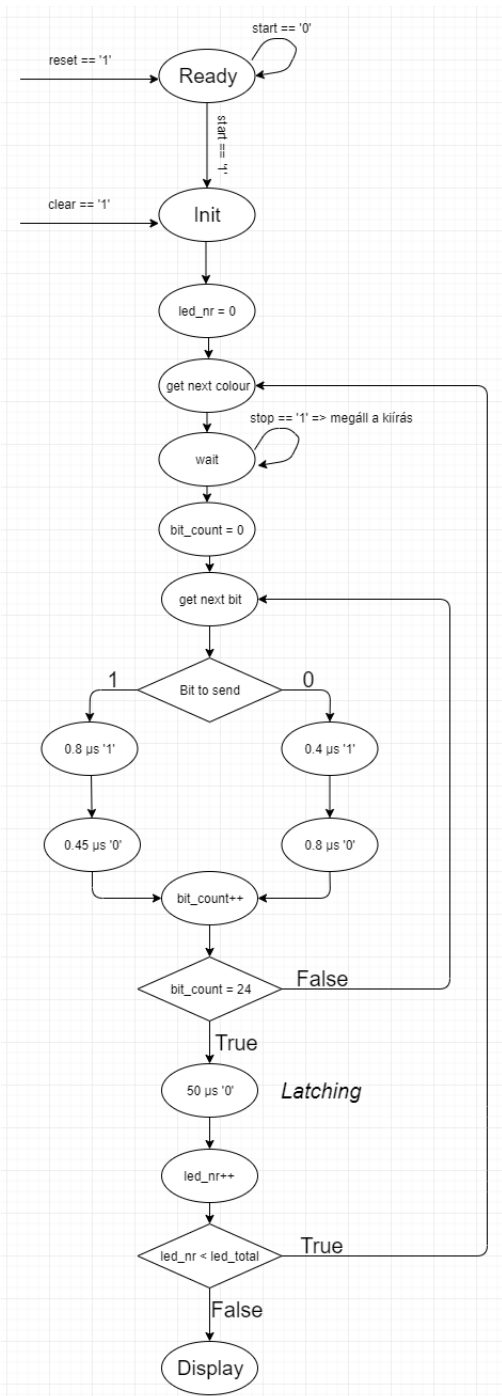
- READY
 - Alap állapot
 - "reset" jel esetén ide kerül vissza az automata

- 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

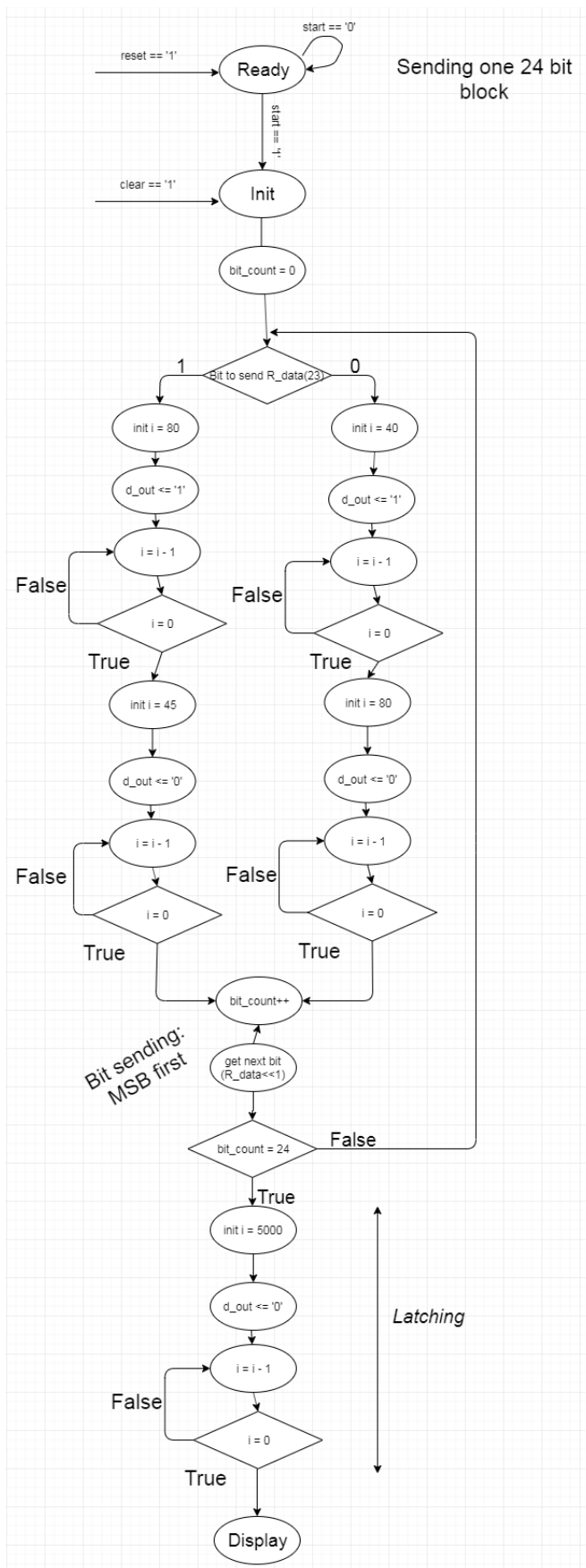


2019.10.28

Állapotdiagram átírva úgy, hogy a küldési logikát is tartalmazza:

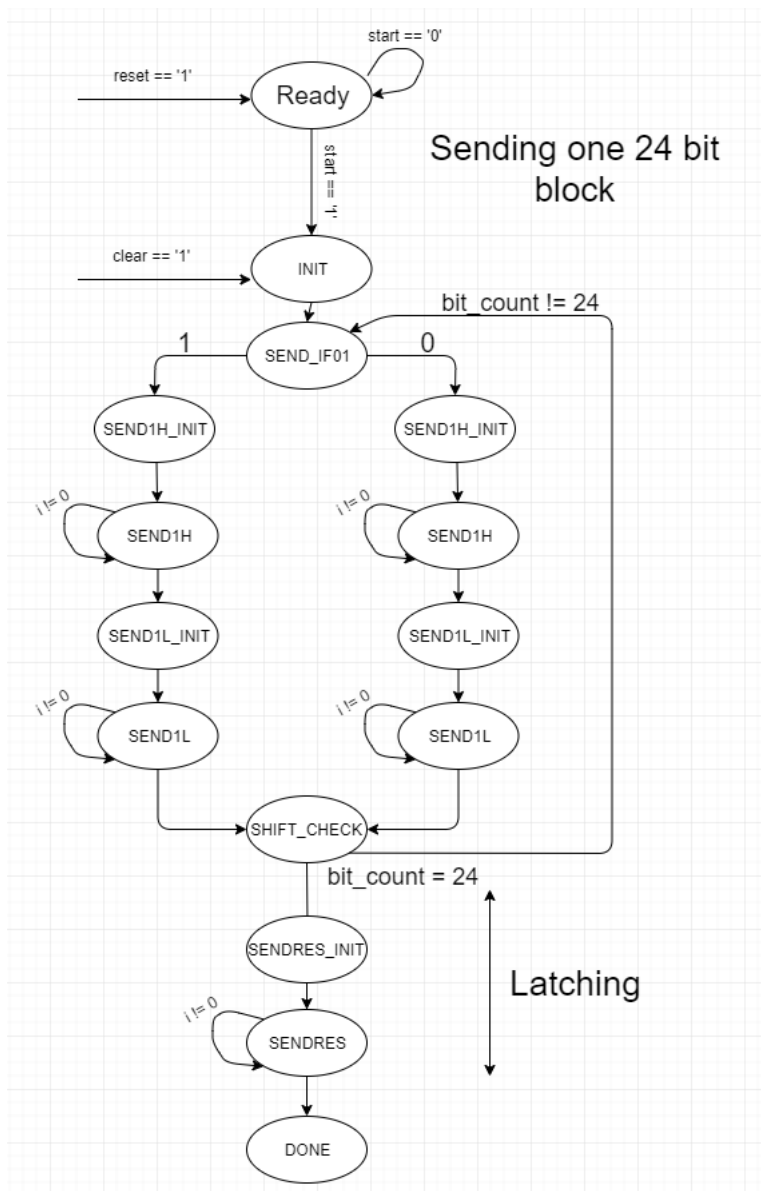


Állapotdiagram egy 100 LED-et vezérlő modulra



2019.11.11

Végleges állapotdiagram, az implementációban is használt.



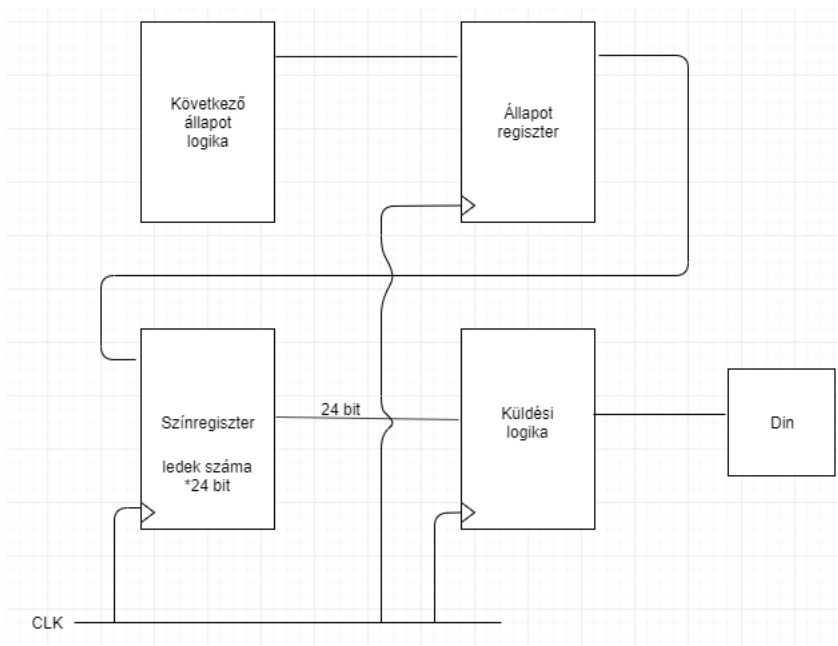
- **READY**
 - Alap állapot
 - "reset" jel esetén ide kerül vissza az automata
- **INIT**
 - inicializálja a bit-count-ot nullára
- **SENDIF_01**
 - Megvizsgálja data[23]-as bit-et. Ha 1, akkor a következő állapot a **SEND1H_INIT**, ha 0 akkor **SEND0H_INIT**
- **SEND1H_INIT**
 - Inicializálja az **i** 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

4.4. Alegységek

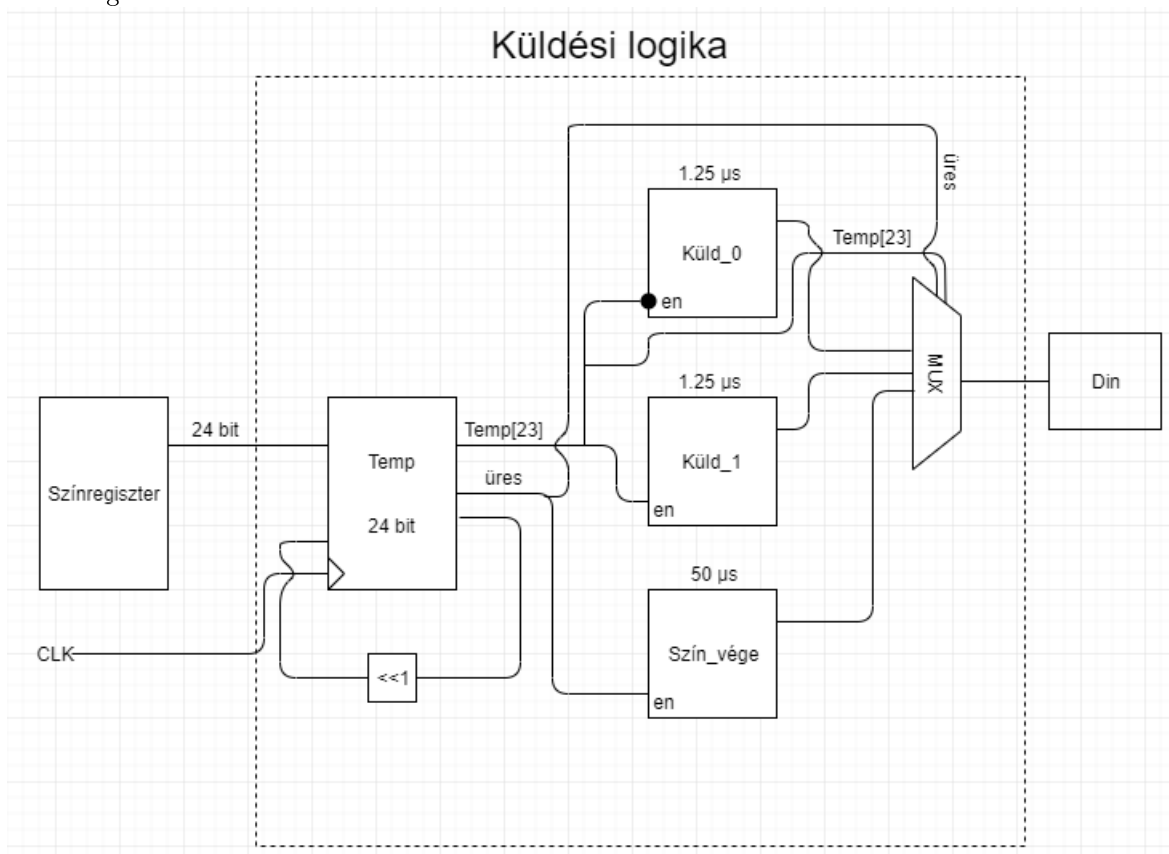
2019.10.14

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



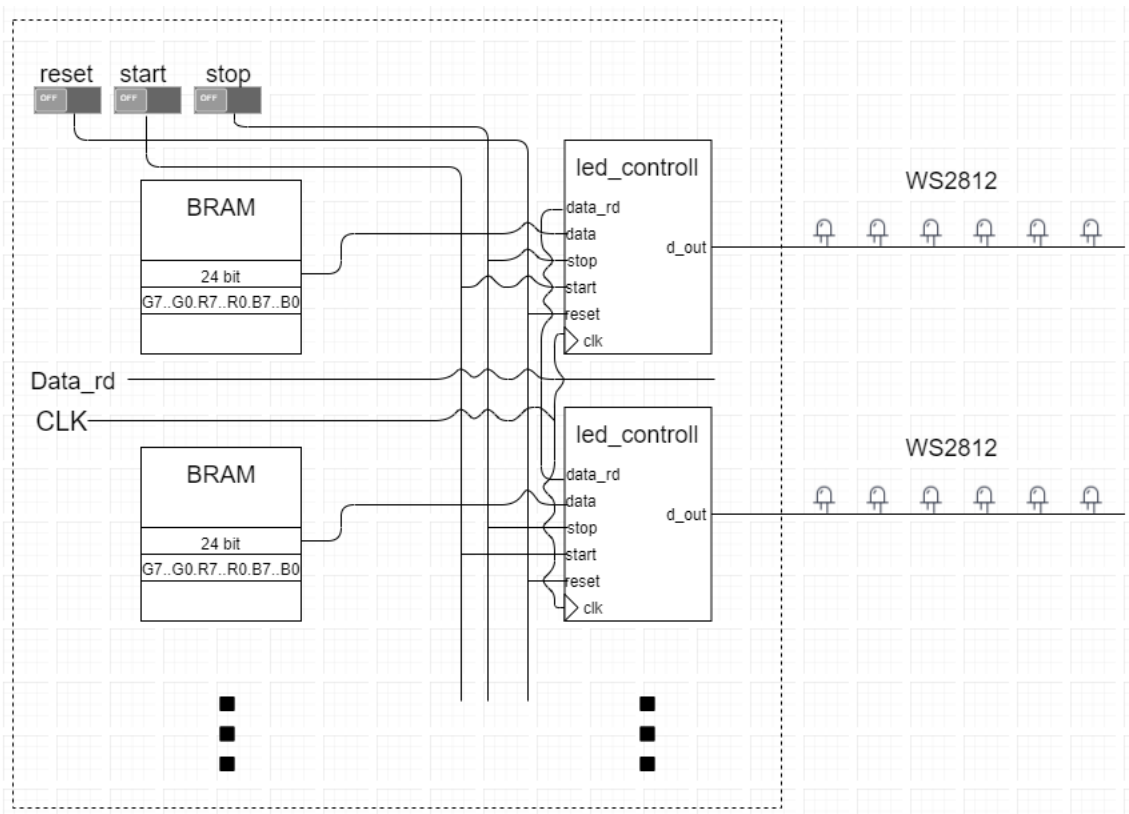
4.4.1. Küldési logika regiszter

A küldési logika modul részletesebb lebontása:

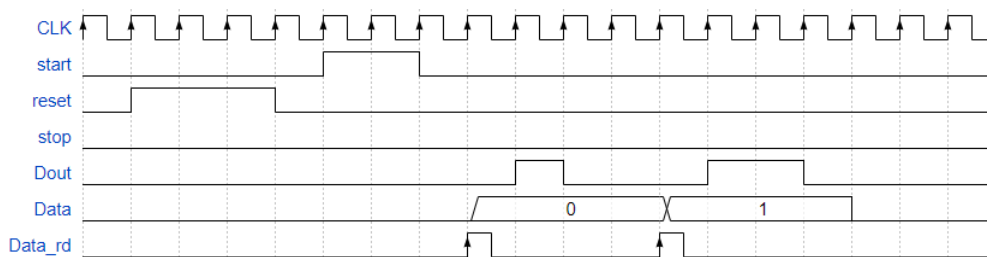


2019.10.28

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.

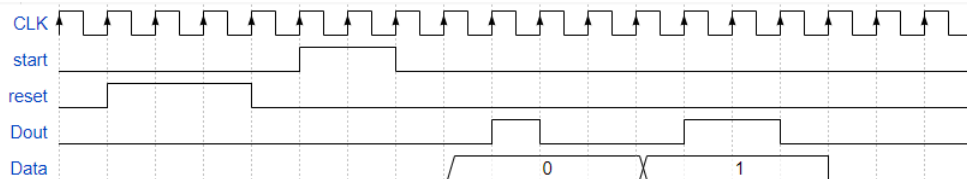


4.5. Idődiagram



- 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

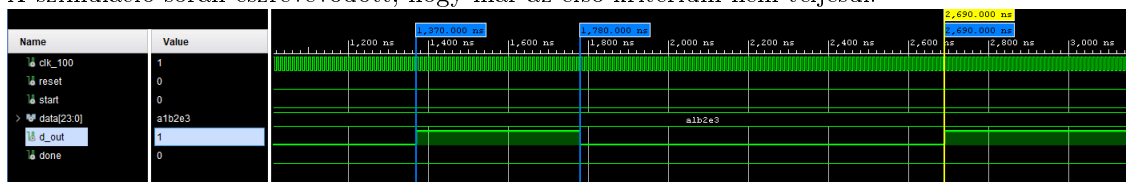
4.6. Szimuláció eredménye

4.6.1. WS2813_Driver

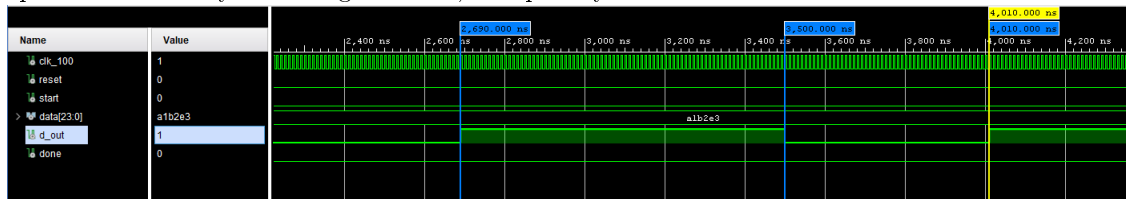
A **WS2813_Driver** modul szimulációja alatt a következőkre figyeltem:

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

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



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



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

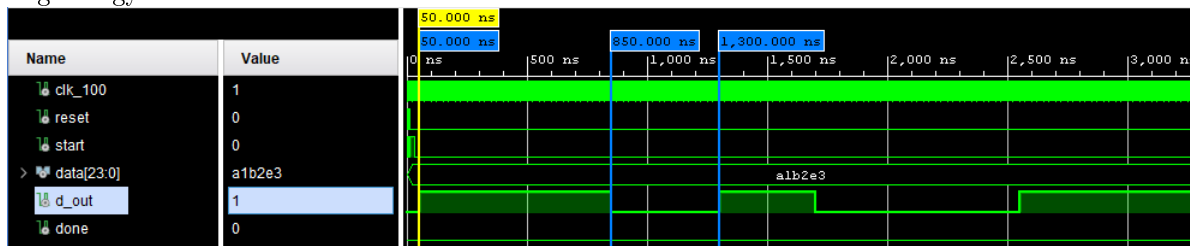
Ezt egyszerűen meg lehet oldani a ciklusszámok módosítása által.

Módosított ciklusszámok:

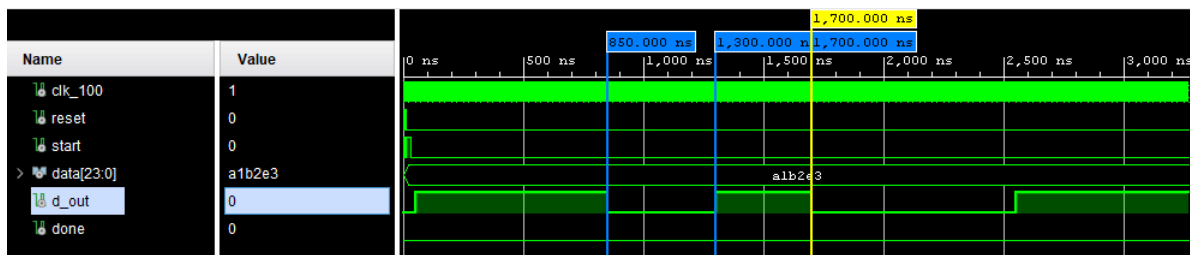
- Logikai 1-es
 - 79 ciklus magas feszültségen
 - 39 ciklus alacsony feszültségen
- Logikai 0-ás
 - 39 ciklus magas feszültségen
 - 79 ciklus alacsony feszültségen

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

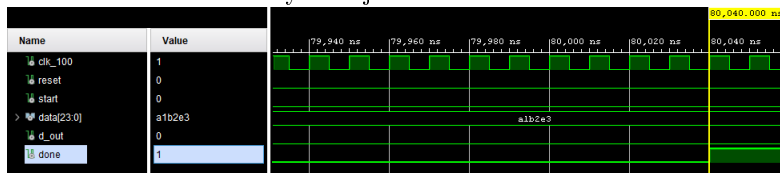
Logikai egyesre:



Logikai nullásra:



A másik két követelmény is teljesül:



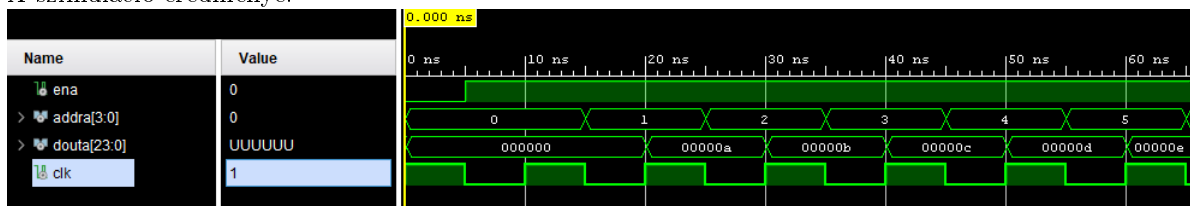
A 24 bit-es blokk elküldése után a done jel egyesre lett állítva és 80.04 μ s alatt sikerült az egész 24 bit-es blokkot elküldeni.

4.6.2. 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,010,011,012,013,014,015,016,017,018,019;
```

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.

4.7. Működés közbeni tesztelés - Integrated Logic Analyser

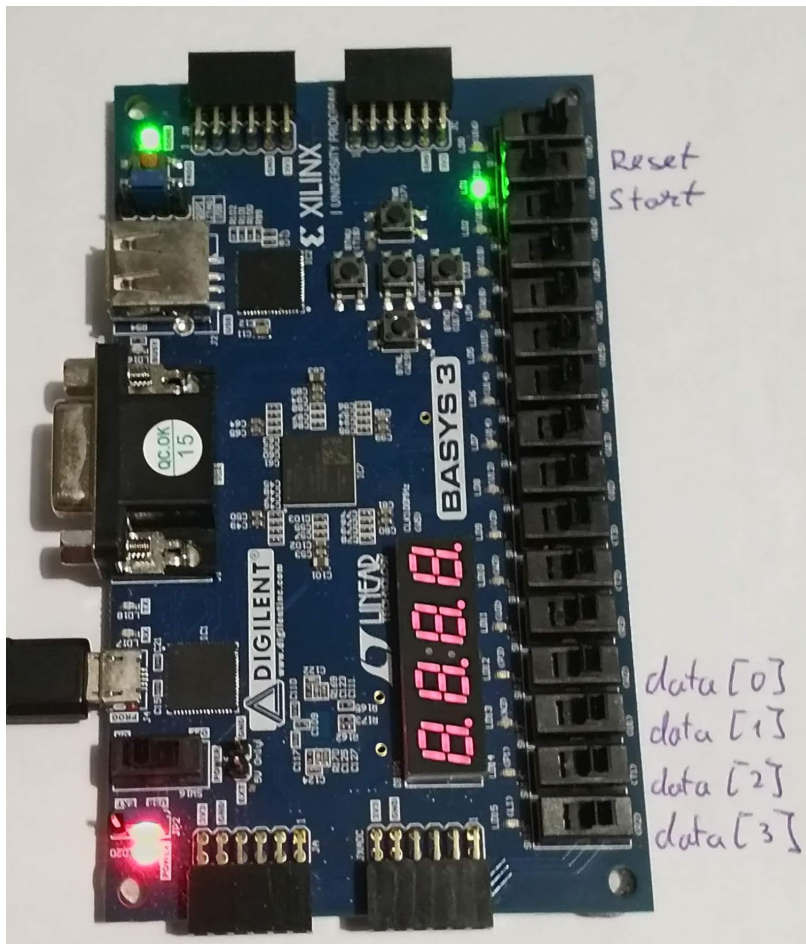
Működés közbeni tesztelésre az FPGA-ba beépített ILA (Integrated Logic Analyser) volt felhasználva.

4.7.1. WS2813_Driver

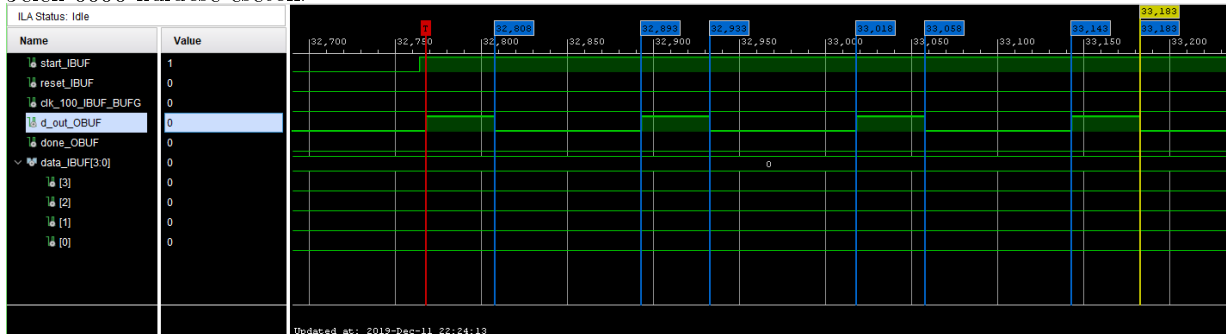
Ennél a modulnál szükség van egy 24 bit-es bemenetre. Mivel annyi kapcsoló nem található a választott FPGA lapon, ezért a modul le lett egyszerűsítve, a tesztelés megkönnyebítéséért. 24 bemenet helyett csak 4 lett felhasználva, ezek be is lettek konfigurálva az fpga négy kapcsolójára.

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 feszültségről 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:

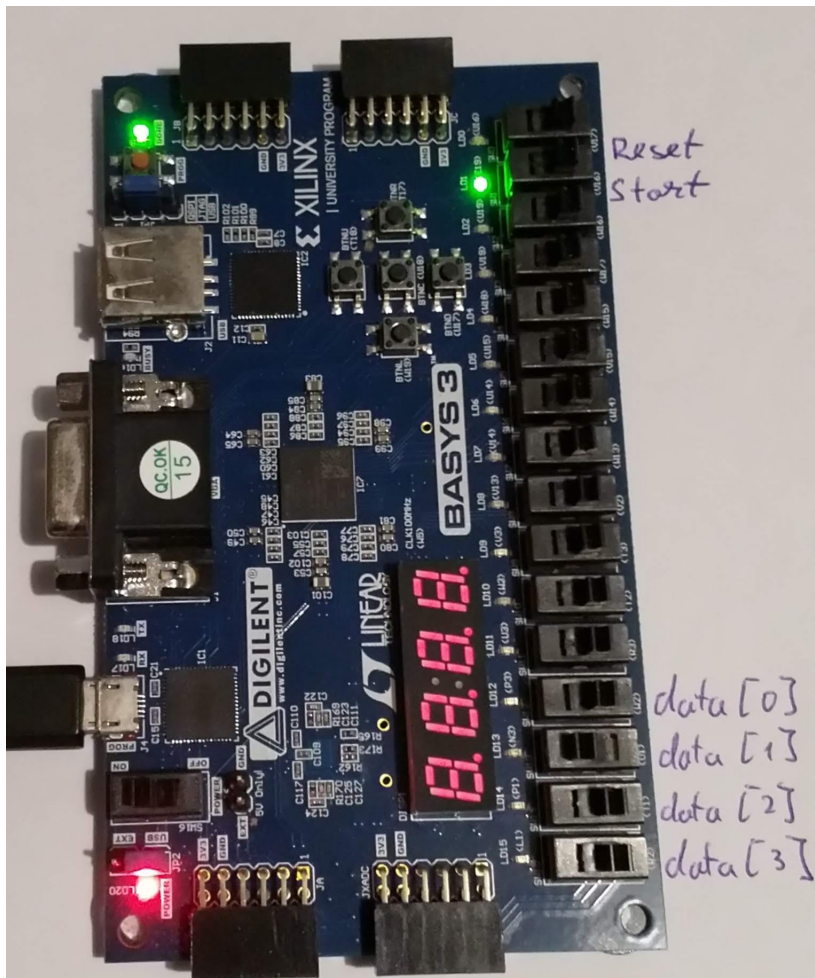


Jelek 0000 küldése esetén:

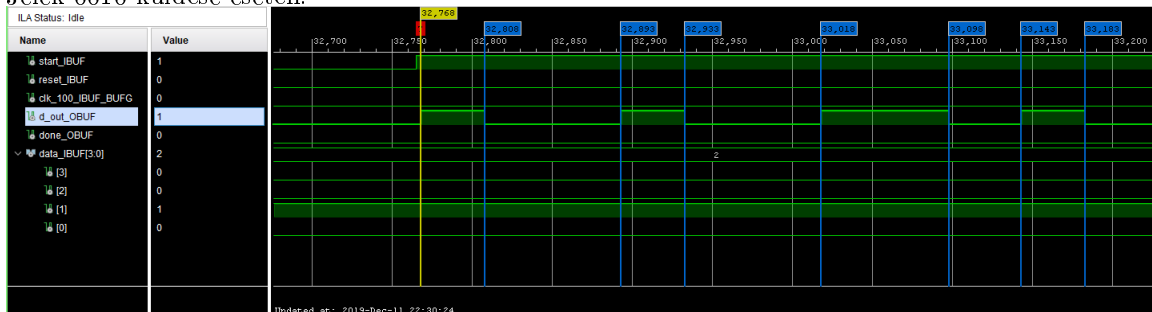


Amint az FPGA-ról és a jelanalizálása közben is látható, az elküldött adat 0000. Ugyanakkor az is látható, hogy a d_out jel annyi időt van magas és alacsony feszültségen, amennyit a protokoll megkövetel. Ezen a példán nem látszik, hogy a küldés az MSB-től (Most Significant Bit) lenne elkezdve. Ennek 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.

4.7.2. Teljes rendszer működés közbeni tesztelése

TODO