# Sharp típusú optikai távolságmérőn alapuló 3D Lidar szerű szkenner

## Újrakonfigurálható digitális áramkörök projekt

***Fejér Norbert***
*Számítástechnika IV.*

# A projekt bemutatása

A projekt célja egy **Lidar szerű 3D szkenner** megvalósítása FPGA áramkör segítségével. Első lépésben a tér feltérképezése vízszintes irányban történik **180º**-ban, egy szervomotorra szerelt Sharp típusú ultrahangos távolságérzékelő segítségével. A projekt továbbfejlesztésének következő lépése ennek a megvalósításnak a kibővítése lenne, mely segítségével 3D-ben is megvalósulna a feltérképezés.

A feladat további része az így kapott távolság megjelenítése, vizualizálása System Generator segítségével. Továbbfejlesztési lehetőségként feltehető ugyanakkor a kapott értékek térben történő megjelenítése különböző vizualizálási technikák segítségével, például egy adott felületen.

# Rendszerkövetelmények

A projekt egy **FPGA** típusú fejlesztői lap segítségével lesz megvalósítva VHDL nyelvben, illetve VIVADO környezet alatt. A modulok tervezése, implementálása és tesztelése során **Vivado 2018.1**-es szoftvert, illetve a Symulink használatához **MatLab R2017b** verzióját használom.

A felhasznált hardverek tekintetében a következő elemre lesz szükségem:

- Zybo fejlesztő lap (Zynq-7000 ARM/FPGA SoC Trainer Board)
- Analóg- digitális átalakító (ADC121S021)
- Sharp típusú optikai távolságmérő (Sharp GP2Y0A60SZLF)
- Szervomotor (Futaba s3003)

# A különböző hardverek által elvárt karakterisztikák

## Zybo fejlesztői lap

- Xilinx Zynq-7000 családra épülő beágyazott szoftver és digitális áramkör fejlesztő platform
- Block RAM mérete: 240 Kb
- DDR3 RAM: 512 MB
- Órajel frekvencia: 450 MHz+
- Pmod csatlakozók száma: 6 Pmod port

## Analóg- digitális átalakító

- Referencia feszültség: 2.7 V - 5.25 V
- Bementi órajel értéke: 1 MHz - 4 MHz
- Mintavételezés értéke: 50 - 200 ksps
- Üzemi hőmérséklet: -40ºC - 85ºC

### Optikai távolságmérő

- Bementi feszültség ($V_{cc}$): 3V vagy 5V
- Kontrol veszültség: 2.3-tól $V_{cc}$-ig - magas feszültség, max 0.2V - alacsony feszültség
- Mérési tartomány: 10- 150 cm
- Egy mérés elvégzéséhez szűkséges idő: minimális idő 16.5 ms, maximális idő: 25.2 ms
- Kimeneti feszültség értéke: 0-5V között
- A kimeneti feszültséget logaritmikusan kapjuk meg, így a későbbiekben ezt linearializálnunk kell az adatok helyes kiolvasása végett

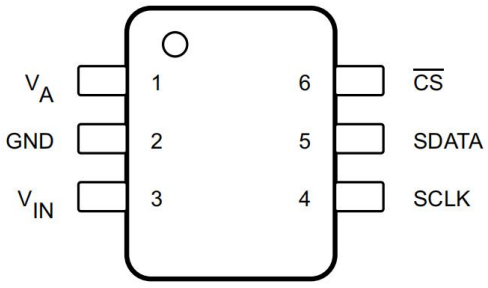
### Szervomotor

- Bementi feszültség ($V_{cc}$): 3V vagy 5V
- Üzemi hőmérséklet: -20°C - 60°C
- Működési sebesség: 0.23 sec / 60° - teher nélkül
- Áttételi nyomaték: 3.2 kg / cm
- Vezérlő rendszer: PWM (Pulse Width Control)

## Rendszer specifikáció

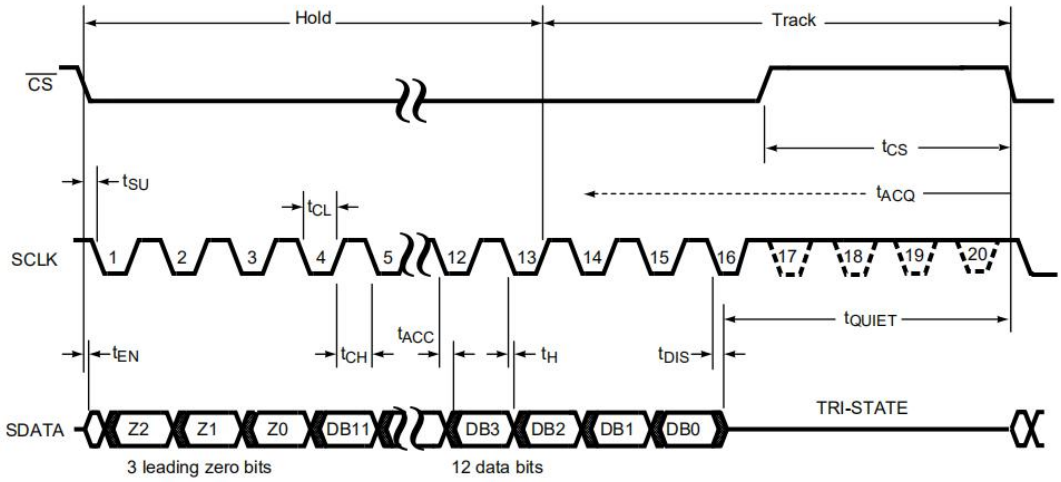
### Az ADC felépítése

### Be- kimeneti jelek



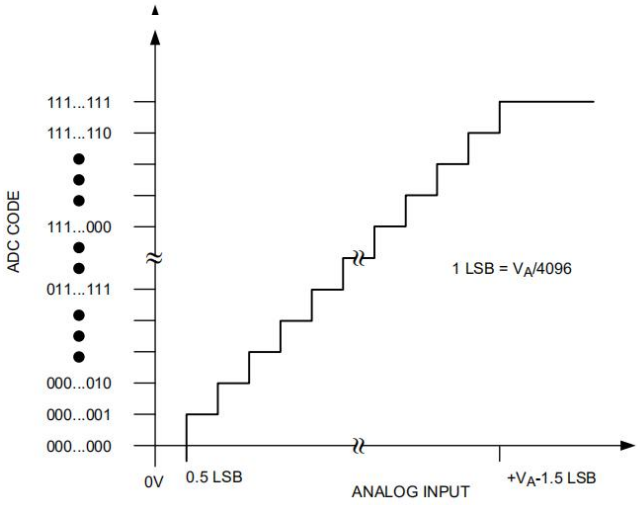

**$V_A$** - referencia feszültség
**GND** - földelés
**$V_{IN}$** - aktuális bementi feszültség, amit átakaítunk majd bináris számmá
**CS** - chip select
**SDATA** - egybites érték, ahol sorra olvasunk majd egy 12 bites bináris számot
**SCLK** - órajel, melynek ütemére az átalakítás megtörténik

## Idődiagram



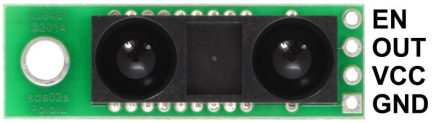

Amikor a CS 1-re állítódik megtörténik a konverzió. Az órajel minden lefutó éllén megtörténik egy új bit kiírása, illetve a felfutó órajel segítségével ki tudjuk olvasni az aktuális bit értékét. A teljes ciklus 20 órajelig tart, az első 3 órajel zérós bitteket fog beolvasni, ezt követi a valós 12 bites érték, mely a számunkra hasznos információt fogja tartalmazni. Az utolsó 4 órajelre történő beolvasásra nem lesz szűkségünk, ez gyakorlatilag csak a teljes ciklus megvárásának idejét és az adatok koherenciáját fogja megvalósítani.

## Az ADC eredménye



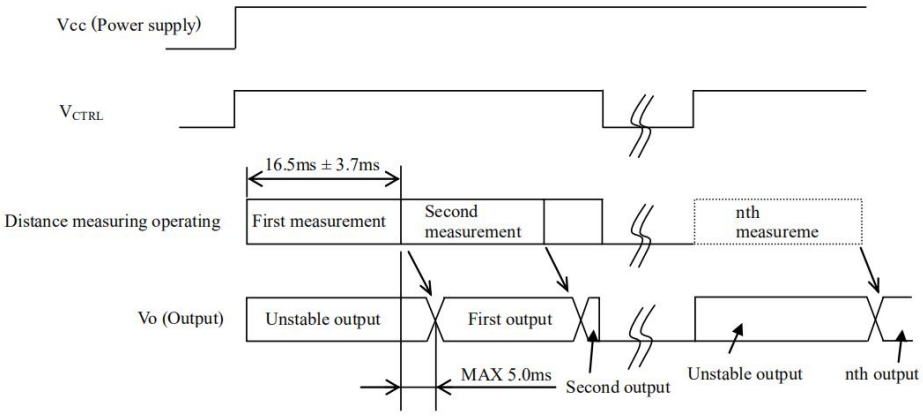

Az ADC kimenete egy 12 bites bináris szám lesz, mely a fenti diagramm szerint vant számolva. A maximális $V_A$ értéknek megfelel egy 12 bites csupán 1-eseket tartalmazó bináris érték, amely gyakorlatban a 4095-ös számnak felel meg, mivel 12 biten ez a maximákisan fel1rható legnagyobb szám. Szóval az aktuális értékünket egy hármas szabálynak megfelelően kell kiszámítanunk, amely a valós értéket fogja megadni és REAL típusként majd visszatérítenünk.

## A távolságmérő felépítése

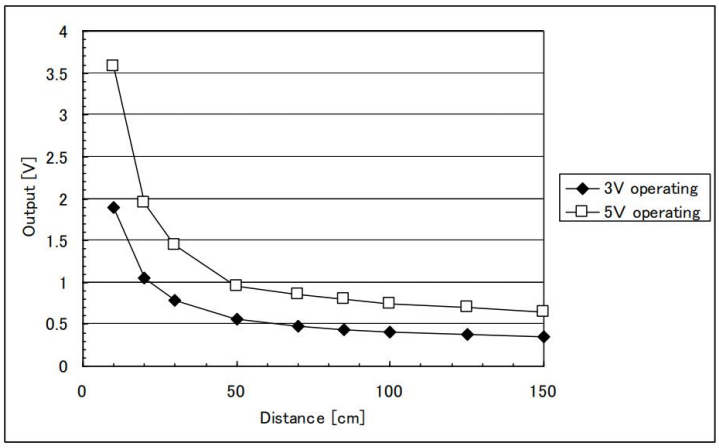



**EN** - enable jel a mérés megkezdése érdekében
**OUT** - kimenő feszültség, melyből ki tudjuk számolni az aktuálisan mért távolságot
**VCC** - tápfeszültség
**GND** - földelés

## Idődiagramm




A $V_{CRTL}$ jel segítégével elindítjuk a mérés elkezdését. Egy max 0.2 V-os jelig az érzékelő standby állapotban van, ha ennél nagyobb jelet adnunk meg, amelyik 2.3V és $V_{CC}$ tartományban van, akkor elkezdődik a mérés. Egy mérés a fent megadott ms időtartamig tart, viszont ez nem stabil mérés lesz, szóval ugyanezt a mérést megismételve megkapjuk az aktuális, stabil és pontos mérés eredményét. Kimenetként szintén egy feszültséget kapunk meg, amelyet a lent látható logaritmikus függvény segítségével tudunk visszaszámolni távolságra, az eredményt cm-ben kapjuk meg. A modulunk 10cm és 150cm- es tartományban mér pontosan.



## A szervomotor felépítése


A Futaba s3003-as termék egy szervomotor, melyet PWM jel segítségével tudunk vezérelni. 3 pin szolgál bemenetként:

**GND** - a föld meghatározása

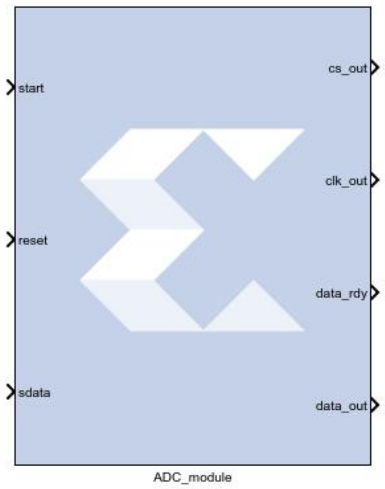
**VCC -** tápfeszültség

**PWM** - PWM jel, melynek segítségével a motrot tudjuk vezérelni.

# Modulok tervezése

A következőkben a VHDL nyelvben leírt modulokat fogom szemléltetni. Fontos megemlítenem, hogy minden modulhoz tartozik egy **src_clk** jel, amely a bemeneti órajelet jelenti, a mi esetünkben az FPGA 100MHz-es jelét, illetve egy **src_ce**, melynek segítségével az órajelet tudom engedélyezni. Ez azért is fontos, mivel ha ilyen formában definiáltam a két jelet a System Generator képes volt felismerni, így megkönnyítette a munkámat is.

## ADC modul tervezése

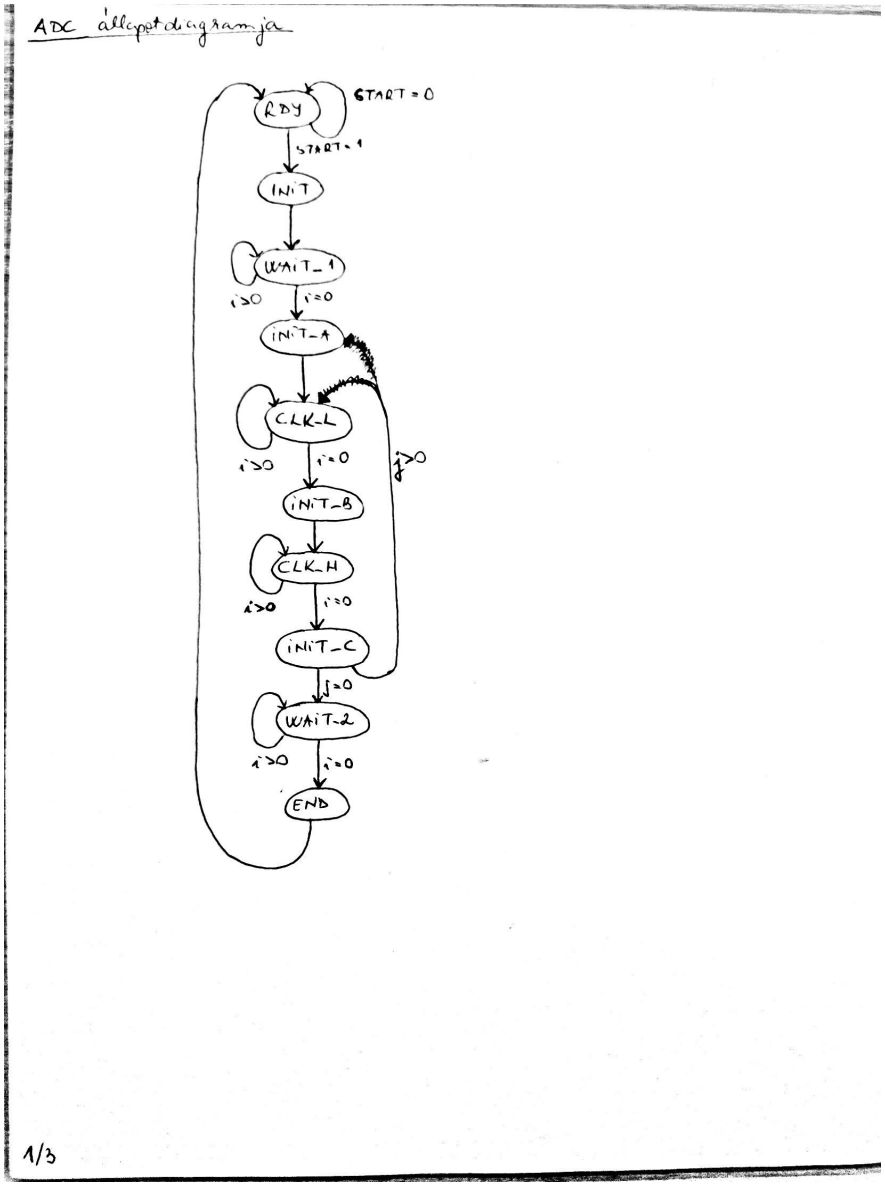


## Jelek definiálása

**start** - elindítom a modul működését
**reset** - kezdeti állapotba helyezem a modult
**sdata** - bemeneti 1 bites adat, itt fogom kapni az ADC-ről érkező biteket minden egyes leosztott órajelre
**cs_out** - engedélyezem a hardver működését
**clk_out** - 1 MHz-es leosztott órajel, mely az ADC hardvert fogja vezérelni
**data_rdy** - jelzem, hogy a modul befejezte a működését
**data_out** - 15 bites kimeneti érték, amely az eredményt fogja tartalmazni

A modul az ADC hardvert fogja vezérelni, amely a leosztott órajel segítségével minden egyes coklusba fog küldeni egy adat bitet, a feladat végeztével a modulom visszatéríti a mérés eredményét egy 15 bites vektor formájában.

## Az ADC modul állapotdiagramja



**Különböző fázisokban elvégzendő műveletek**

| | $R_{CLK}$ | $R_i$ | $R_j$ | $R_D$ | $R_S$ |
|---|---|---|---|---|---|
| RDY | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow R_i$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| INIT | $R_{CLK} \leftarrow 0$ | $R_i \leftarrow M$ | $R_j \leftarrow 16$ | $R_D \leftarrow 0$ | $R_S \leftarrow 0$ |
| WAIT_1 | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow R_i - 1$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| INIT_A | $R_{CLK} \leftarrow 0$ | $R_i \leftarrow N$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| CLK_L | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow R_i - 1$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| INIT_B | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow N$ | $R_j \leftarrow R_j$ | $R_D \leftarrow (R_D << 1) \mid SDATA$ | $R_S \leftarrow R_S$ |
| CLK_H | $R_{CLK} \leftarrow 1$ | $R_i \leftarrow R_i - 1$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| INIT_C | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow M$ | $R_j \leftarrow R_j - 1$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| WAIT_2 | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow R_i - 1$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow R_S$ |
| END | $R_{CLK} \leftarrow R_{CLK}$ | $R_i \leftarrow R_i$ | $R_j \leftarrow R_j$ | $R_D \leftarrow R_D$ | $R_S \leftarrow 1$ |

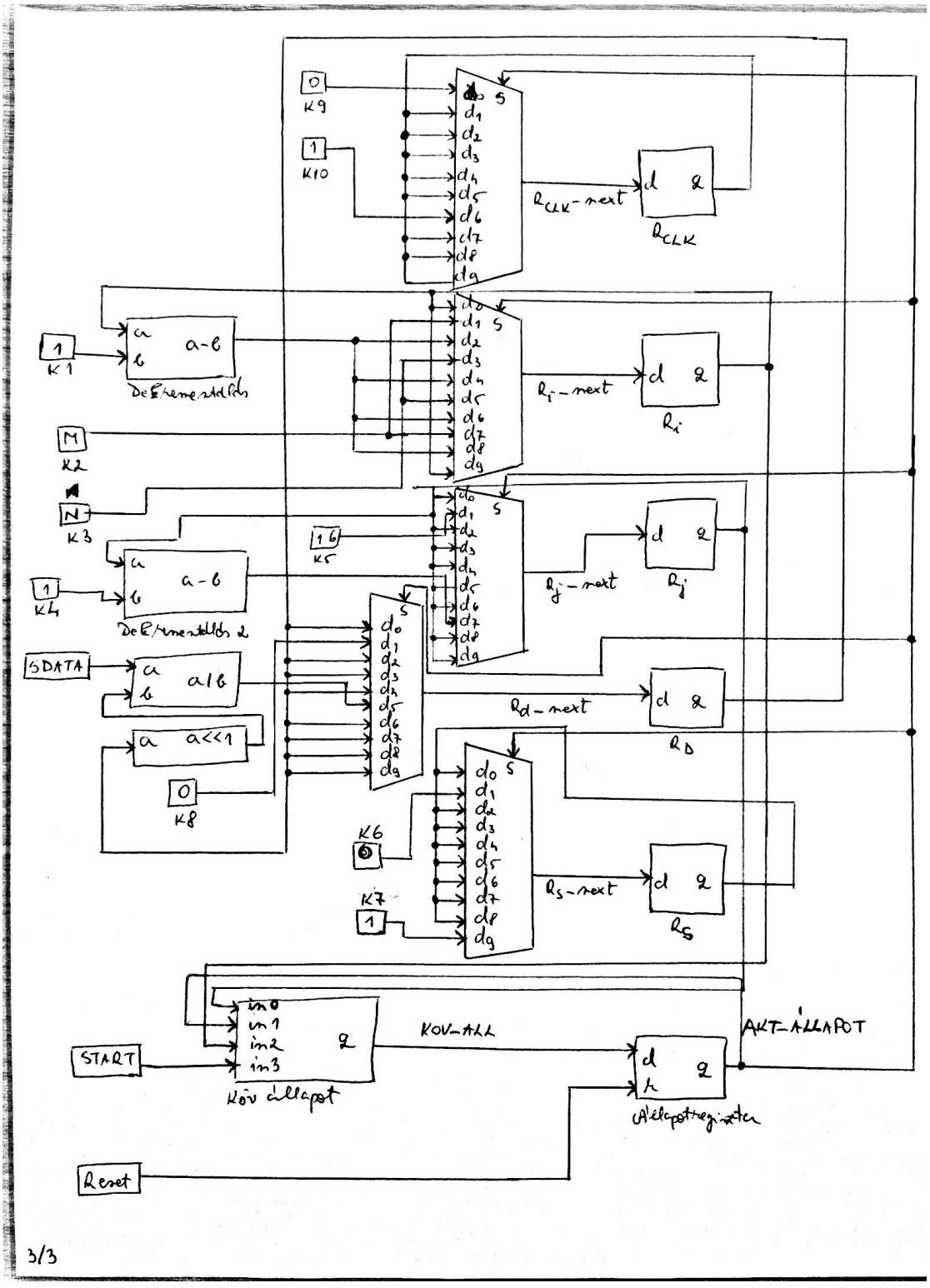
N = 2

~~N = 16~~

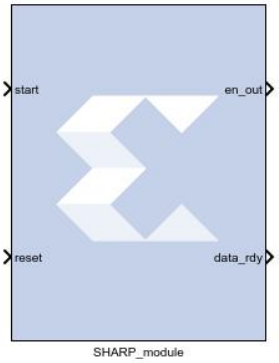
N = 50

2/3

**Az ADC modul tömbvázlata**



## SHARP modul tervezése




### Jelek definiálása

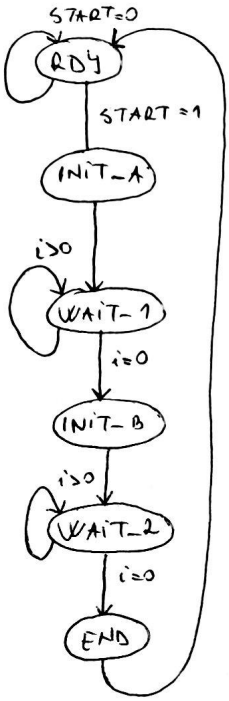
**start** - elindítom a modul működését
**reset** - kezdeti állapotba helyezem a modult
**en_out** - engedélyezem a hardver működését
**data_rdy** - jelzem, hogy a modul befejezte a működését

A SHARP modul segítségével végzem el a távolságmérés levezénylését. Amikor engedélyezem a hardver működését az elkezd mérni, majd a mérés befejeztével a mért feszültséget küldöm az ADC modulnak, amely a megfelelő bináris számmá fogja alakítani azt.

A hardever működésekor 25,7ms idő szükséges ahhoz, hogy a mért érték stabil legyen, illetve a mérés befejezése után ezt az értéket a kimenetén még kell tartsa 120ns-ot, mivel az ADC modulom ennyi idő alatt végez egy átalakítást a kapott feszültségből. Alkalmaztam egy órajelosztót, aminek segítségével a frekvenciát leosztottam 100kHz-re, így sokkal átláthatóbbá tettem a modul működését és nem volt szűkségem hatalmas számok, számlálók használatára a működés során.

**A SHARP modul állapotdiagramja**

SHARP állapotdiagramja




| | $R_i$ regiszter | $R_{EN}$ regiszter | $R_{end}$ regiszter |
|---|---|---|---|
| RDY | $R_i \leftarrow R_i$ | $R_{EN} \leftarrow R_{EN}$ | $R_{end} \leftarrow R_{end}$ |
| INIT-A | $R_i \leftarrow N$ | $R_{EN} \leftarrow 1$ | $R_{end} \leftarrow 0$ |
| WAIT_1 | $R_i \leftarrow R_i - 1$ | $R_{EN} \leftarrow R_{EN}$ | $R_{end} \leftarrow R_{end}$ |
| INIT-B | $R_i \leftarrow M$ | $R_{EN} \leftarrow R_{EN}$ | $R_{end} \leftarrow R_{end}$ |
| WAIT-2 | $R_i \leftarrow R_i - 1$ | $R_{EN} \leftarrow R_{EN}$ | $R_{end} \leftarrow R_{end}$ |
| END | $R_i \leftarrow R_i$ | $R_{EN} \leftarrow 0$ | $R_{end} \leftarrow 1$ |

$N = 2570000$

$16{,}5\,ms + 3{,}7\,ms + 5\,ms + 2\,ms = 25{,}7\,ms = 25700000\,ns$

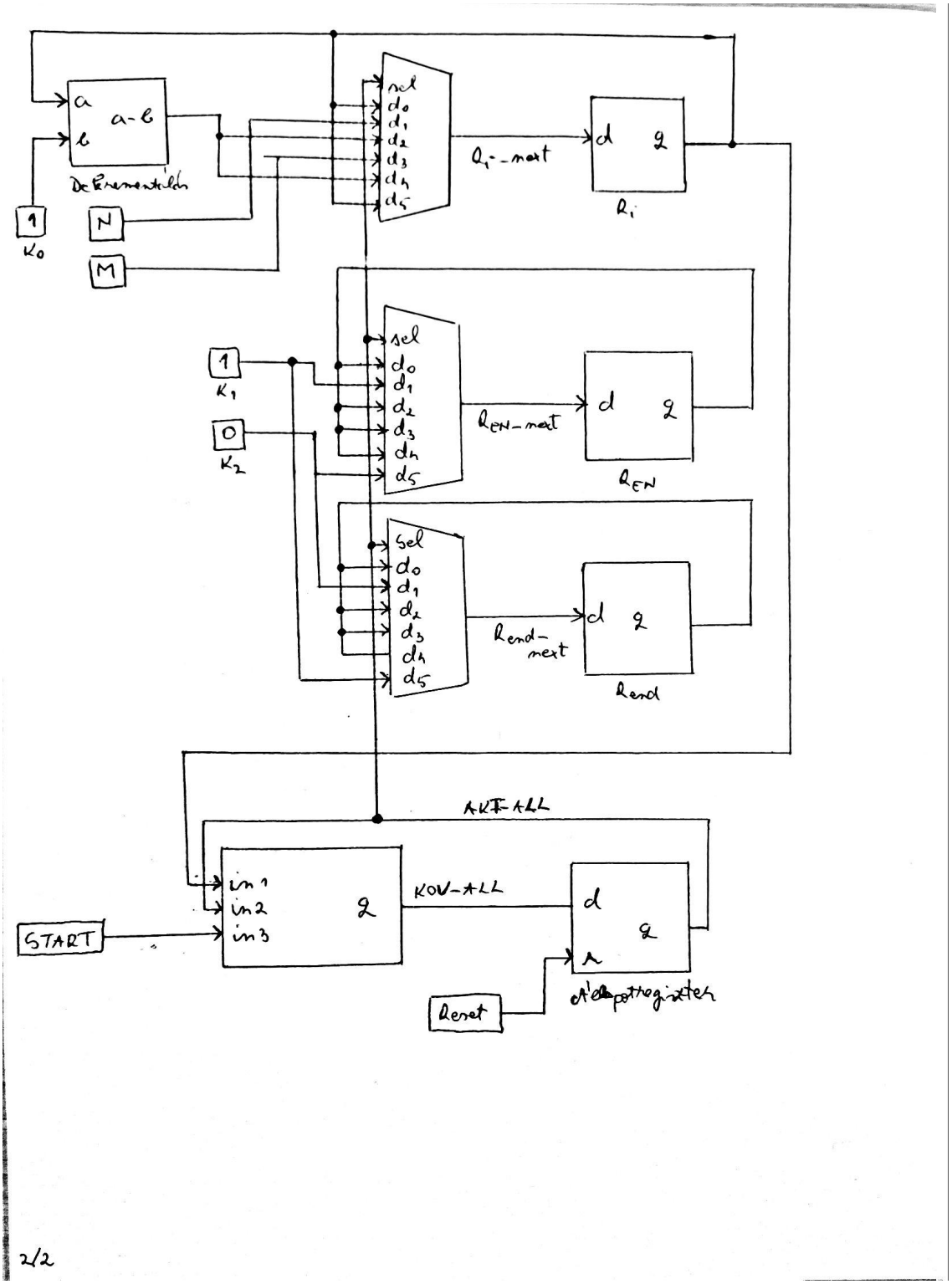
FPGA frekvenciája 100 MHz

$T = \frac{1}{100 \cdot 10^6} = 0{,}01\,ms = 10\,ns$ 1 ciklus ideje

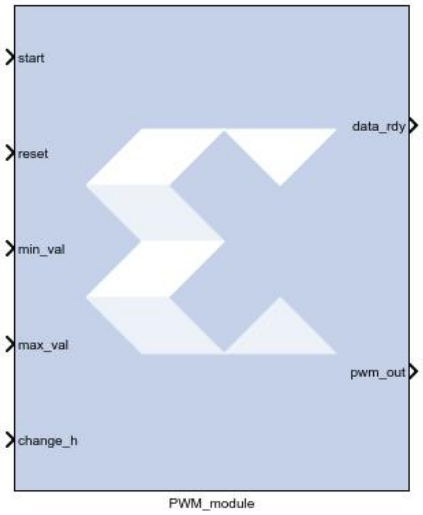
$M = 120_{ms}$ - ennyi idő szükséges az ADC-nek az átalakításhoz

1/2

**A SHARP modul tömbvázlata**



## PWM modul tervezése



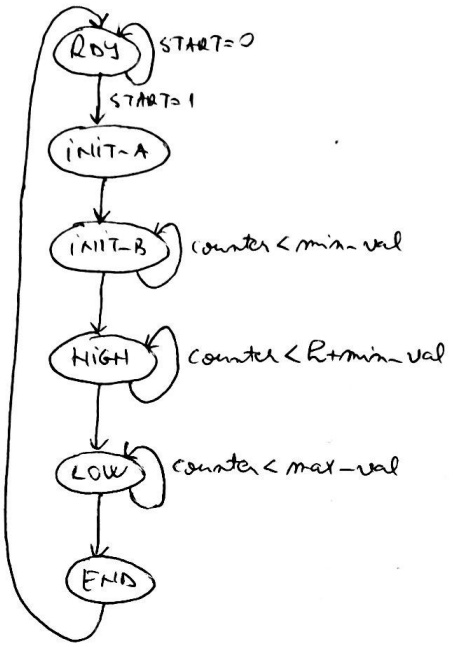

### Jelek definiálása

**start** - elindítom a modul működését
**reset** - kezdeti állapotba helyezem a modulomat
**min_val** - a minimális érték, amíg a PWM jelemet HIGH állapotba tartom
**max_val** - a PWM jel periódusának meghatározása
**change_h** - ha erre a jelre 1-et adok, akkor megváltoztatom a kitöltési tényezőt. Ha elért egy bizonyos felső határt akkor csökkenteni fogom ezt, ha elért egy előre megadott alsó határt akkor növelni fogja ezt az értéket (h értékét)
**data_rdy** - jelzem, amikor a modulom befejezte egy periódus elvégzését, ami azt jelenti, hogy fordított egyet a motron (jelen esetben a fordulat $1{,}5^{o}$ -ot jelent)
**pwm_out** - a tényleges kimeneti PWM jelem

A PWM modul segítségével vezérlem a szervomotoromat. A jel periódusa egy előre definiált érték, ami jelen esetben 20ms. Ebből a min_val érték 0.9ms és a max_val 2.1ms, ami azt jelenti, hogy a kitöltési tényező [0, 1.5] intervallumban változik. Minden egyes change_h érték 1-re állításával változtatom a kitöltési tényező (h) értékét. Ez az érték 0.01ms-al növekszik mindig, ami azt jelenti, hogy a szervomotorom $1.5^{o}$-ot fog fordulni minden egyes esetben, ami egy igencsak jó felbontáshoz vezet a távolságmérés során is. [$0^{o}$, $180^{o}$]-ban forgatom a motromat, ami azt jelenti, hogy ha a (h+min_val) eléri a 2.1ms-ot csökkentem a h-t, ellenkező esetben ha h > 0.9 ms növelem a h tényleges kitöltési tényező értékét.

**A PWM modul állapotdiagramja**

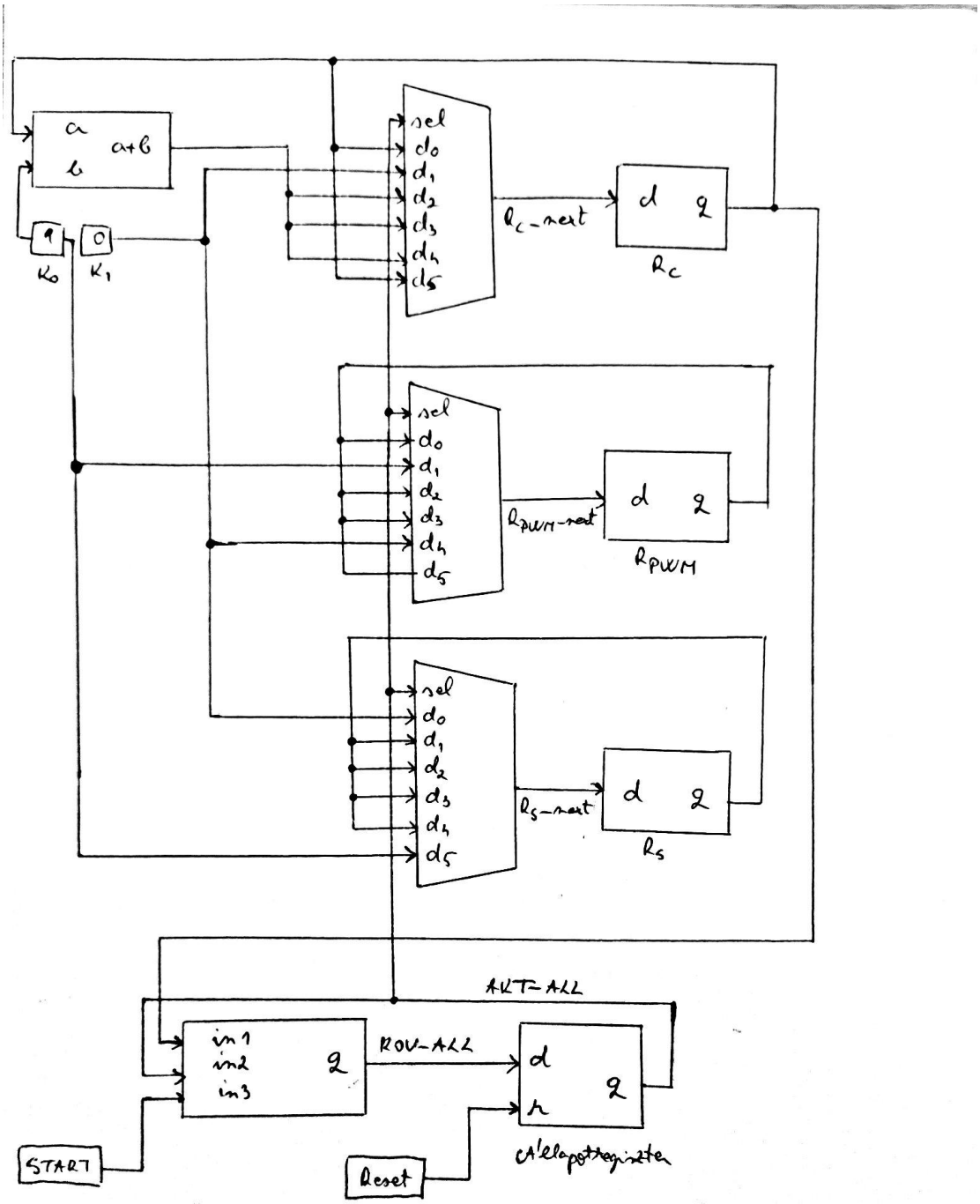
PWM Modul állapotdiagramja




| | $R_C$ | $R_{PWM}$ | $R_S$ |
|---|---|---|---|
| RDY | $R_C \leftarrow R_C$ | $R_{PWM} \leftarrow R_{PWM}$ | $R_S \leftarrow R_S$ |
| INIT-A | $R_C \leftarrow 0$ | $R_{PWM} \leftarrow 1$ | $R_S \leftarrow 0$ |
| INIT-B | $R_C \leftarrow R_C + 1$ | $R_{PWM} \leftarrow R_{PWM}$ | $R_S \leftarrow R_S$ |
| HIGH | $R_C \leftarrow R_C + 1$ | $R_{PWM} \leftarrow R_{PWM}$ | $R_S \leftarrow R_S$ |
| LOW | $R_C \leftarrow R_C + 1$ | $R_{PWM} \leftarrow 0$ | $R_S \leftarrow R_S$ |
| END | $R_C \leftarrow R_C$ | $R_{PWM} \leftarrow R_{PWM}$ | $R_S \leftarrow 1$ |

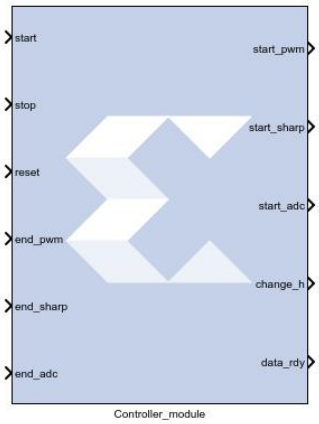
1/2

**A PWM modul tömbvázlata**




2/2

## Contoller modul tervezése



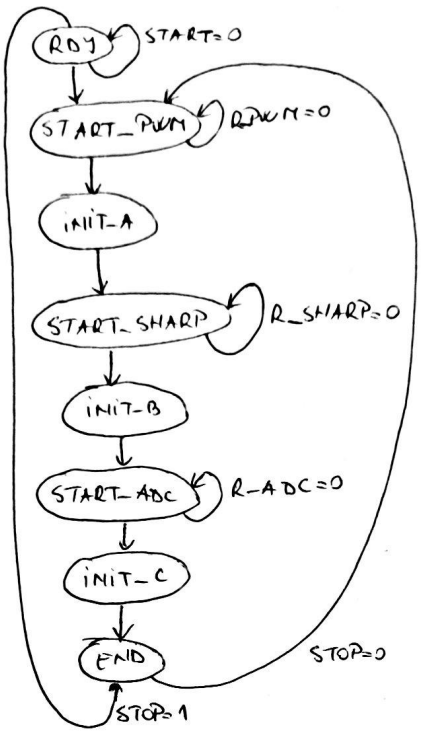

### Jelek definiálása

**start** - modult engedélyező jel
**reset** - kezdeti állapot beállítása
**stop** - leállítom a program futását
**end_pwm** - jelzi, hogy a PWM modul befejezte a működését
**end_sharp** - jelzi, hogy a SHARP modul befejezte a működését
**end_adc** - jelzi, hogy az ADC modul befejezte a működését
**start_pwm** - a PWM modul elindítására szolgáló jel
**start_sharp** - a SHARP modul elindítására szolgáló jel
**start_adc** - az ADC modul elindítására szolgáló jel
**change_h** - a kitöltési tényező megváltoztatása a PWM modulban
**data_rdy** - egy teljes mérést és az adat feldolgozásának elvégzését jelző bit

A Controller modul vezérli az összes többi modult, illetve ő felel az adatok szinkronizálásáért is a modulok között. Minden modult egymás után indít el, bevárva minden modul helyes működését, így az adatok közti konkurenciát kiküszöböli. Lényegében ez a modul fogja össze az összes többi modul működését, ami azt jelenti, hogy a megfelelő sorrendben és időben engedélyezi, tiltja és változtatja az állapotaikat a megfelelő működés következtében.

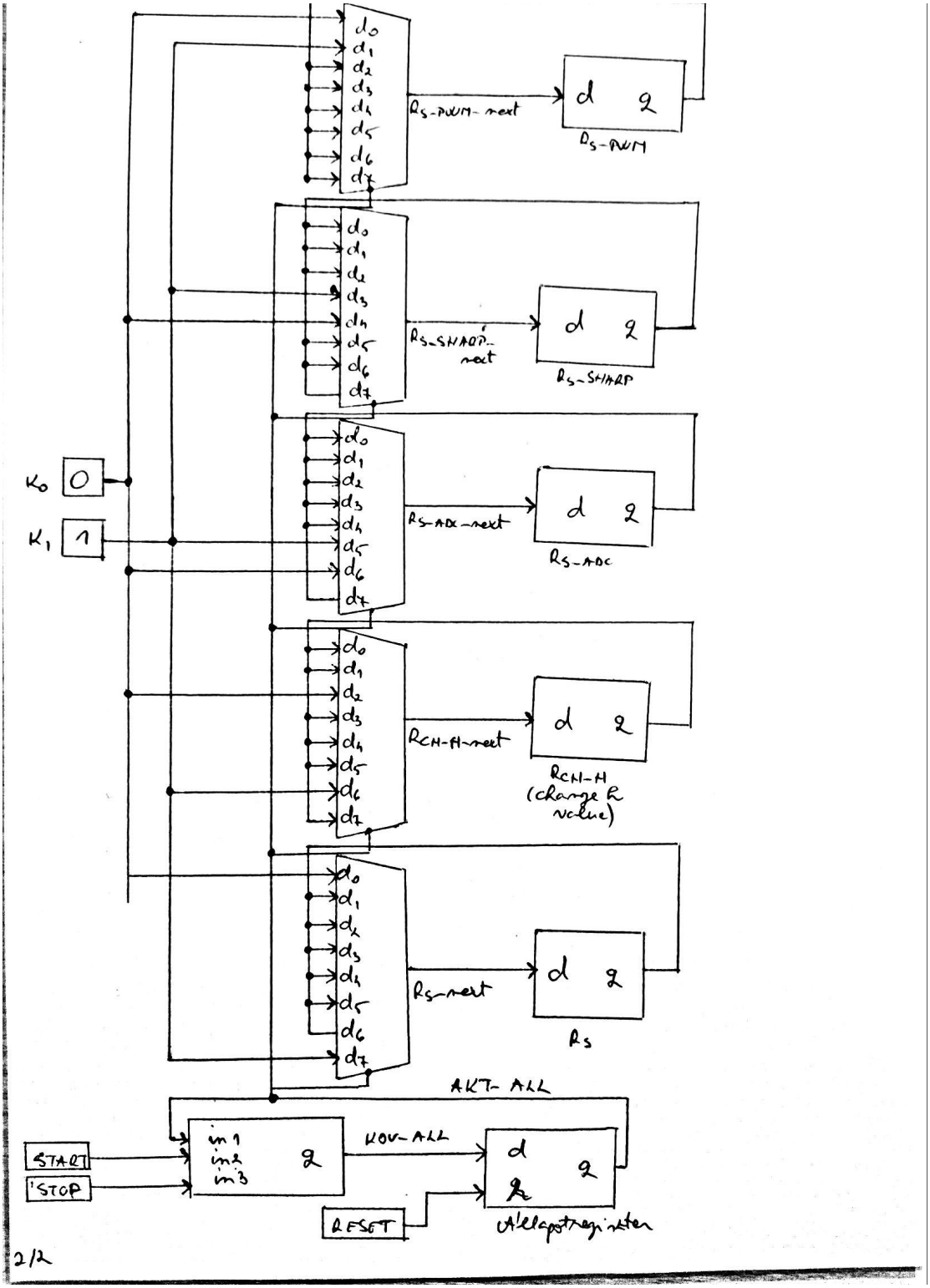
**A Contoller modul állapotdiagramja**

Vezérlő modul állapotdiagramja



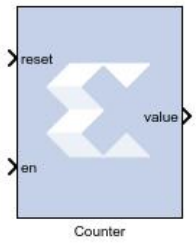

| | $R_{S\_PWM}$ | $R_{S\_SHARP}$ | $R_{S\_ADC}$ | $R_{CH-H}$ | $R_S$ |
|---|---|---|---|---|---|
| RDY | ~~$R_{S\_PWM}$~~ 0 | $R_{S\_SHARP}$ | $R_{S\_ADC}$ | $R_{CH-H}$ | $R_S$ |
| START-PWM | 1 | $R_{S\_SHARP}$ | $R_{S\_ADC}$ | $R_{CH-H}$ | 0 |
| INIT-A | $R_{S\_PWM}$ | $R_{S\_SHARP}$ | $R_{S\_ADC}$ | 0 | $R_S$ |
| START-SHARP | $R_{S\_PWM}$ | 1 | $R_{S\_ADC}$ | $R_{CH-H}$ | $R_S$ |
| INIT-B | $R_{S\_PWM}$ | 0 | $R_{S\_ADC}$ | $R_{CH-H}$ | $R_S$ |
| START-ADC | $R_{S\_PWM}$ | $R_{S\_SHARP}$ | 1 | $R_{CH-H}$ | $R_S$ |
| INIT-C | $R_{S\_PWM}$ | $R_{S\_SHARP}$ | 0 | 1 | $R_S$ |
| END | $R_{S\_PWM}$ | $R_{S\_SHARP}$ | $R_{S\_ADC}$ | $R_{CH-H}$ | 1 |

**A Contoller modul tömbvázlata**



# Segéd modulok tervezése

## Counter modul felépítése



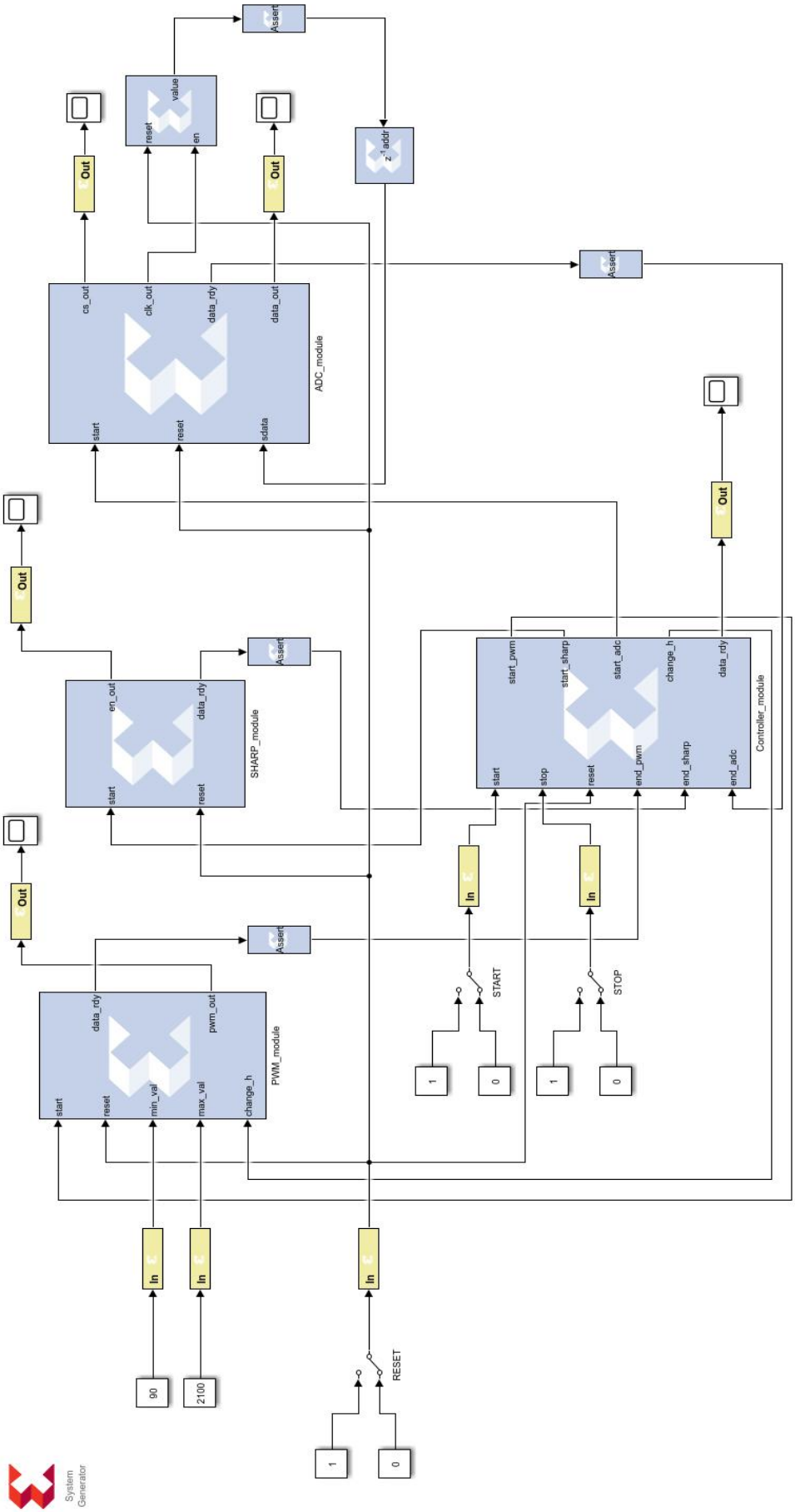

### Jelek definiálása

**reset** - újrainítom a számlálót
**en** - engedélyezem a számlálót
**value** - a számláló aktuális kimeneti értéke

A modul lényegében véve egy 0-tól 15-ig körkörösen számláló. Erre azért volt szűkségem, mert habár az FPGA órajelére működik, csak akkor kell számoljon, amikor azt az **en** jellel engedélyezem. Ennek segítéségevel címezek meg egy ROM típusú memóriát, amit a System Generator során fogok használni. Mivel minden modul különböző órajelen működik, éppen ezért szűkségem volt, hogy ez a modul csak az **en** jel felfutó órajelén számoljon, ezzel tudtam kiküszöbölni azt a problémát, hogy az adott modulomban amíg egy órajelet tartja az **en** állapotot 1-be, addig a fő modulomban sokkal több ideig van 1-es állapotban, mivel más modulokban órajel osztókat alkalmaztam.

Ez gyakorlatilag csak egy sajátos számláló, ami abban tér el a többitől, hogy csak az **en** jel felmenő éllén számol.

# A projekt System Generatorban való megvalósítása


Assert
value
reset
en
Out
Out
z_addr
Assert
cs_out
clk_out
data_rdy
data_out
ADC_module
start
reset
sdata
Out
Out
Assert
en_out
data_rdy
SHARP_module
start
reset
start_pwm
start_sharp
start_adc
change_h
data_rdy
Controller_module
start
stop
reset
end_pwm
end_sharp
end_adc
Out
Assert
In
In
START
STOP
1
0
1
0
data_rdy
pwm_out
PWM_module
start
reset
min_val
max_val
change_h
In
In
In
90
2100
RESET
1
0
System
Generator

# Könyvészet


1. Újrakonfigurálható digitális áramkörök - Brassai Sándor Tihamér
2. https://www.pololu.com/product/2474
3. http://www.ti.com/lit/ds/symlink/adc121s021.pdf
4. https://www.es.co.th/schemetic/pdf/et-servo-s3003.pdf
5. https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_ref.pdf
6. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug897-vivado-sysgen-user.pdf
7. https://www.youtube.com/watch?v=BDq8-QDXmek&list=PLZv8x7uxq5XY-IQfQFb6mC6OXzz0h8ceF
8. https://numato.com/kb/learning-fpga-verilog-beginners-guide-part-1-introduction/