SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM
Marosvásárhelyi Kar

Légnyomás mérés BMP180 as szenzorral és I2C kommunikáció

Halgató neve: *Demeter Ákos-Elemér*

Szak: *Számítástechnika*

Évfolyam: ***4***

Tantárgy: Újrakonfigurálható digitális áramkörök

Projekt véglegesítésének időpontja: 2025.01.04

A).Projekt célja

A projekt célja egy BMP180 légnyomásmérő szenzorral való kommunikációt megvalósító digitális rendszer fejlesztése és tesztelése, amely az I2C protokollt használja az adatátvitelhez. A rendszer feladata, hogy a szenzortól érkező adatokat hatékonyan olvassa, feldolgozza, majd olyan formában adja vissza, amely mindennapi alkalmazásokhoz, például időjárás-figyeléshez vagy magasságméréshez használható.

B) Követelmények

a. Funkcionális követelmények

1.I2C protokoll implementáció:

- Az SDA és SCL vonalak szabályos működésének biztosítása az I2C szabvány szerint.
- Az eszközcím (0xEE) és a regisztercímek kezelésének pontos megvalósítása.
- Írási (Write) és olvasási (Read) műveletek teljes körű támogatása.

2.BMP180 kommunikáció:

- A BMP180 érzékelő helyes inicializálása.
- Kalibrációs paraméterek kiolvasása a pontos adatfeldolgozás érdekében.
- Nyomás adatok lekérdezése és a szenzor által biztosított nyers adatok feldolgozása.

3.Adatfeldolgozás:

- A BMP180-ból származó adatok konverziója:
    - Hőmérséklet Celsius fokban.
    - Légnyomás hPa-ban (hektopascal).
    - Magasság számítása a tengerszint feletti magasság becsléséhez.
- Az adatok exportálása továbbfelhasználásra (pl. időjárásfigyelés vagy magasságmérés).

4.Szimuláció és teszt:

- Az I2C jelek idődiagramjának helyességének validálása (SDA és SCL jelváltozások).
- A rendszer tesztkörnyezeti viselkedésének szimulációja

b. Nem funkcionális követelmények

1.Teljesítmény:

- Az I2C kommunikációnak valós időben kell működnie (órajel frekvenciája: max. 32 kHz)

2.Pontosság:

- A légnyomás számítása ne térjen el ±5 hPa-nál többel a szenzor specifikációtól.

3.Megbízhatóság:

- A rendszernek stabilan kell működnie hosszú távú használat során, például folyamatos I2C kommunikációval több percen keresztül is.

4.Skálázhatóság:

- A rendszer könnyen bővíthető legyen más I2C eszközökkel való együttműködésre.

5.Kódminőség:

- A VHDL kód olvasható, moduláris, és újrafelhasználható legyen.
- Megfeleljen a szabványos kódolási irányelveknek.

6.Szimulációs környezet kompatibilitása:

- A tesztek és szimulációk kompatibilisek legyenek a Xilinx Vivado fejlesztöi környezettel.

C) Tervezés

1.C I2C kommunikációs protokol általános ismertetése:

Az I2C protokoll (Inter-Integrated Circuit) egy szinkron soros kommunikációs protokoll, amelyet az elektronikus eszközök közötti adatátvitelre használnak. Mára széles körben alkalmazzák mikrovezérlők, érzékelők, kijelzők és más perifériák összekapcsolására.

Alapvető jellemzők:

1.Kétvezetékes interfész:

- SDA (Serial Data): Adatvonal, amely az adatok átvitelére szolgál.
- SCL (Serial Clock): Órajelvonal, amely az adatok szinkronizálásához szükséges.

2.Master-Slave architektúra:

- Az I2C rendszer egy vagy több master eszközt és egy vagy több slave eszközt tartalmazhat.
- A master kezdeményezi a kommunikációt és vezérli az órajelet, míg a slave követi az utasításokat.

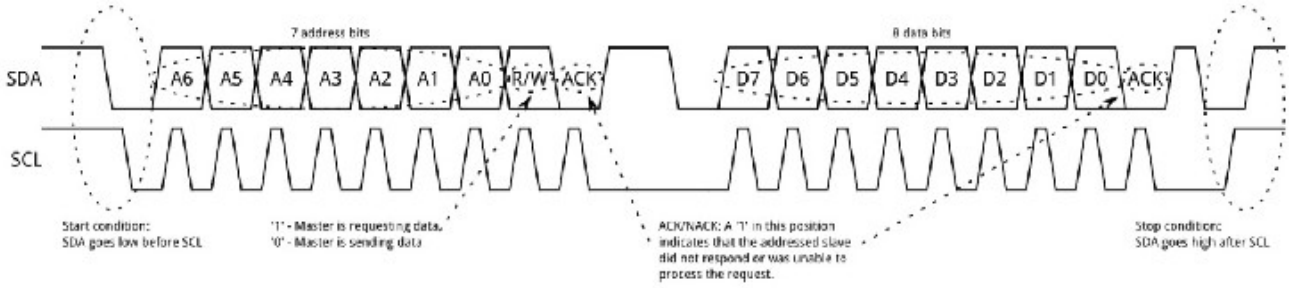
3.Címzés:

- Minden slave eszköz rendelkezik egy egyedi címmel (általában 7 vagy 10 bites).
- A master ezen cím segítségével azonosítja, hogy melyik slave-vel szeretne kommunikálni.

4.Kommunikáció módja:

- Az adatátvitel kétirányú, így ugyanaz az SDA vonal használható mind az adatok küldésére, mind a fogadására.
- Az adatok byte-alapúak, és minden byte után egy ACK (Acknowledge) jel érkezik.

Hogyan működik?



I2C általános kommunikációs időidiagram

Az üzenetek két részre oszlanak: a címzésre, ahol a master jelzi, hogy melyik slave eszközzel kíván kommunikálni és egy vagy több 8 bites adatcsomagra, amelyet küldhet a master a slave eszköznek, vagy a slave is a masternek. Az adatokat az SDA vonalra helyezzük, miután az SCL alacsonyra csökken, és akkor kerül feldolgozásra, amikor az SCL vonal magasra emelkedik. Az órajel magas szintre emelkedése és az adatok beolvasásának vége közötti időt az IC gyártója adja meg.

A kommunikáció indítása

A címzés elküldése előtt a master az SCL lábat magasra hagyja, és az SDA-t alacsonyra húzza. Ez minden slave eszközt figyelmeztet arra, hogy az átvitel megkezdődik. Ha két mester eszköz egyidejűleg kívánja elfoglalni a buszt, akkor az az eszköz, amelyik az SDA-t vonalat először húzza le alacsonyra, az lesz jogosult a kommunikációra. Lehetőség van ismételt üzenet küldésére anélkül, hogy a busz feletti rendelkezést a master átadná másik eszköznek, erről később beszélünk.

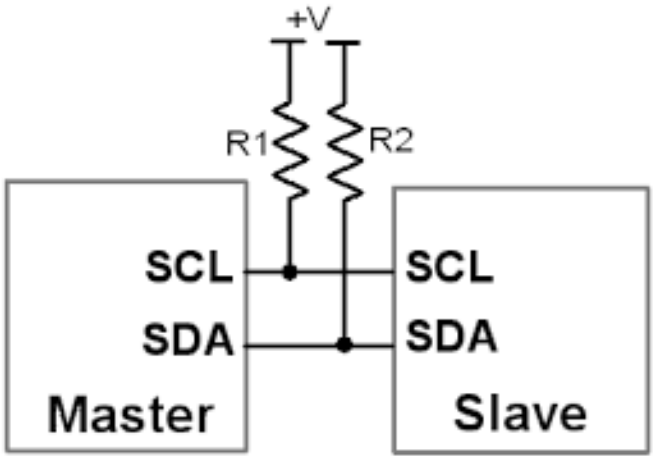
Címzés

Minden új kommunikációs csomagban először a címzés kerül továbbításra. 7 bites címek esetén először a címet a legfontosabb bittel (MSB) kell lezárni, amelyet egy R/W (Read/Write, Olvasás/Írás) bit követ, amely jelzi, hogy ez olvasási (1) vagy írási (0) művelet.

A cím 9. bitje a NACK/ACK (jóváhagyás, ACKnowledge) bit (ez vonatkozik a címzésre és az adatátvitelre is). Miután a címzés első 8 bitjét elküldtük (7+R/W), a fogadó eszköznek kell az adatvonalat alacsonyra húznia, ezzel jelzi, hogy tudomásul vette, hogy vele akar a master kommunikálni és egyben azt is jelzi, hogy készen áll a kommunikációra.

Adattovábbítás

A címzés elküldése után az adatok következnek. A mester egyszerűen folytatja az órajel impulzusok generálását, és az adatokat az master vagy a slave helyezi az SDA vonalra, attól függően, hogy az R/W bit olvasási vagy írási műveletet jelez-e. Az adatcsomagok száma tetszőleges, és a legtöbb slave eszköz automatikusan növeli a belső regisztert, azaz a későbbi olvasások vagy írások a sorban lévő következő regiszterből származnak.



I2C kommunikációs vonalak

Előnyök

- **Egyszerűség:** Csak két vezeték szükséges a kommunikációhoz.
- **Több eszköz támogatása:** Egy buszon akár több master és slave eszköz is lehet.
- **Költséghatékonyság:** Kevés hardverigénye miatt költséghatékony.

Hátrányok

- **Korlátozott távolság:** Az I2C-t rövid távolságokra tervezték.
- **Alacsony sebesség:** Nem alkalmas nagy sebességű adatátvitelre.
- **Busz ütközés:** Több master használata esetén ütközések előfordulhatnak.

2.C BMP 180 as légnyomás mérő szenzor általános leirás:

A BMP180 egy népszerű, rendkívül precíz légnyomás- és hőmérséklet-érzékelő, amelyet a Bosch Sensortec fejlesztett ki. Széles körben használják időjárás-állomásokban, magasságmérésben, valamint különféle IoT és beágyazott rendszerekben. Az I2C protokollon keresztül kommunikál, így egyszerűen csatlakoztatható mikrovezérlőkhöz és más digitális rendszerekhez.

Főbb jellemzők:

1. **Mérések:**
    - **Légnyomás:** 300 hPa - 1100 hPa tartományban.
    - **Hőmérséklet:** -40 °C és +85 °C között.
2. **Felbontás és pontosság:**
    - Nyomásérzékelés: ±0.02 hPa pontosság.
    - Hőmérséklet: ±2 °C pontosság.
3. **Kommunikáció:**
    - **Interfész:** I2C protokol
    - **Orajel frekvencia (fSCL):** Max. 3.4 MHz

- **SDA és SCL pull-up elenállás:** 2.2 – 10 kOhm

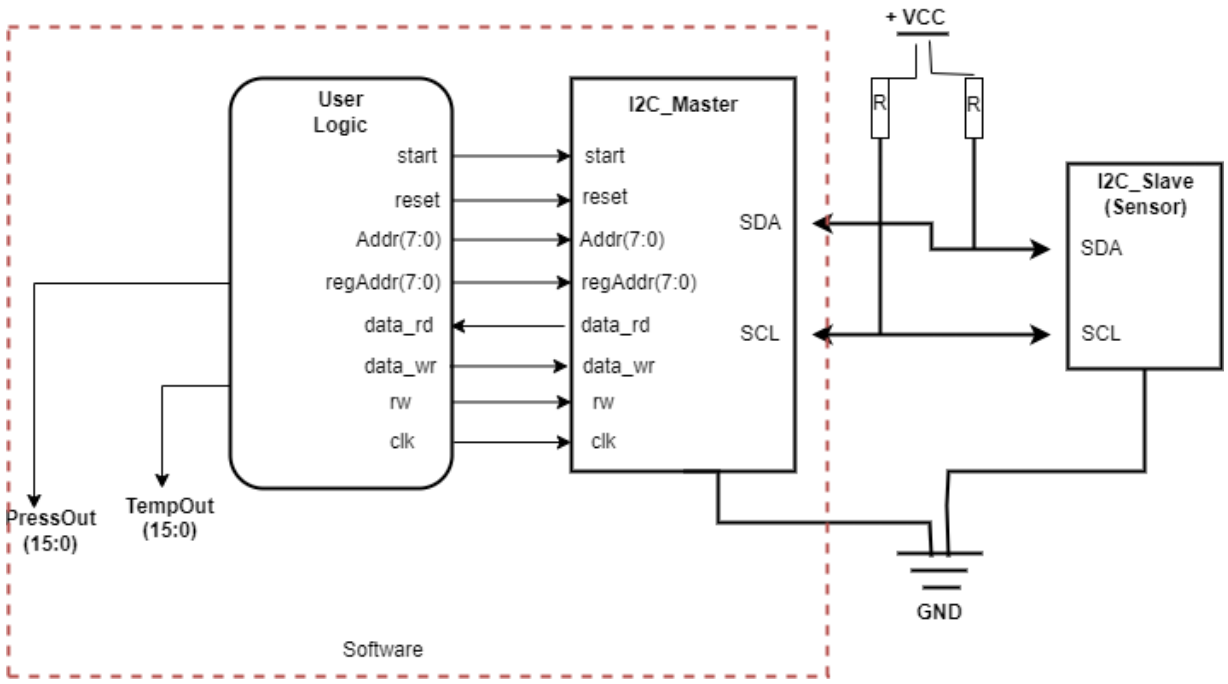
Hogyan működik?

A BMP180 egy MEMS technológiával készült érzékelőt használ, amely a légköri nyomást méri. A szenzor nyomásértékeiből a magasság is kiszámítható. A szenzor által mért adatokat egy belső kalibrációs algoritmus dolgozza fel, amely az érzékelő hőmérsékletét és nyomását figyelembe véve ad pontos eredményt.

Felhasználási módok

1. **Időjárás-állomások:** Légnyomás és hőmérséklet mérése.
2. **Repüléstechnika és drónok:** Magasság meghatározása.
3. **Okostelefonok:** Magasság alapú szolgáltatások támogatása (pl. lépésszámlálók vagy helymeghatározás).

3.C Tömbvázlat:



Projekt tömbvázlata

Modulok és jelek magyarázata:

1.User Logic (Felhasználói Logika)

Ez az egység felel az I2C kommunikáció vezérléséért és az adatok feldolgozásáért. A következő bemenetek és kimenetek találhatók itt

- **start**: Jelzi az adatátviteli művelet elindítását
- **reset**: A rendszer visszaállítására szolgáló jel.
- **Addr[7:0]**: Az I2C eszköz címe (pl. a BMP180 esetében 0xEE).
- **regAddr[7:0]**: Az eszköz regiszterének címe, amelyhez írás vagy olvasás történik.
- **rw**: Az írási (0) vagy olvasási (1) művelet meghatározása.
- **clk**: Órajel, amely szinkronizálja a kommunikációt.
- **data_rd**: Az érzékelőtől olvasott adat.
- **data_wr**: Az érzékelő felé küldött adat.
- **PressOut[15:0]**: A feldolgozott légnyomásérték.
- **TempOut[15:0]**: A feldolgozott hőmérsékletérték.

A User Logic végzi az I2C vezérléshez szükséges parancsok előállítását és a szenzortól érkező adatok értelmezését.

2.I2C_Master

Ez az egység az I2C protokoll működését valósítja meg. Fő feladatai:

- **Bemenetek**:
  - A User Logic által biztosított jelek, például **start**, **reset**, **Addr[7:0]**, **regAddr[7:0]**, **data_wr**, **rw**, és az **clk**.
- **Kimenetek**:
  - **SDA**: Az I2C adatvonal (bidirekcionális jel).
  - **SCL**: Az I2C órajel.

Az I2C_Master felelős az SDA és SCL vonalak kezeléséért az I2C szabvány szerint. Ez biztosítja, hogy a megfelelő eszköz azonosítás, adatküldés és fogadás valósuljon meg az érzékelővel.

3.I2C_Slave (Sensor)

Ez az egység maga a BMP180 szenzor, amely a nyomás- és hőmérsékletadatokat biztosítja. Jellemzői:

- **SDA** és **SCL**: Az adat- és órajelek fogadására és küldésére szolgáló vonalak.
- Az I2C_Slave kommunikál az I2C_Master modullal, és a megadott regiszterekből adatot szolgáltat, illetve fogad.

4.Pull-up ellenállások

Az SDA és SCL vonalak mindkét végén található pull-up ellenállások (+VCC felé kötve) biztosítják az I2C kommunikáció megfelelő működését. Ezek megakadályozzák az SDA és SCL lebegését, amikor egyik eszköz sem aktív.

5.Áramellátás és földelés (+VCC, GND)

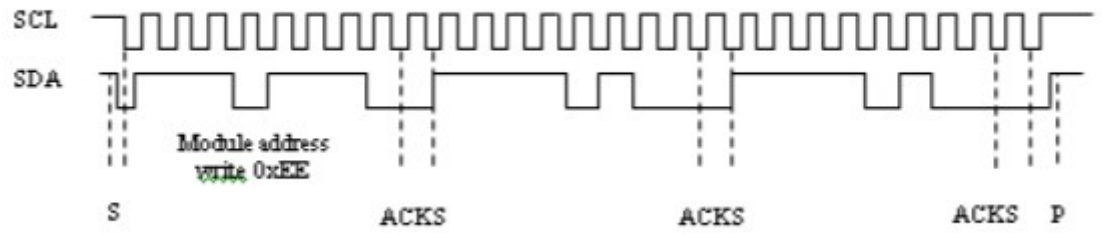
Az I2C rendszer működéséhez szükséges tápellátást és földelést biztosítják. Ez garantálja, hogy minden modul közös referenciapontra legyen kötve, és az áramkör helyesen működjön.

A rendszer működésének összefoglalása:

1. A **User Logic** parancsot ad az I2C_Master modulnak az adatok írására vagy olvasására.
2. Az **I2C_Master** az SDA és SCL vonalak segítségével kommunikál a BMP180 szenzorral (I2C_Slave).
3. A BMP180 érzékelő visszaküldi a mért adatokat, amelyeket a User Logic feldolgoz, és ember számára értelmezhető formában (hőmérséklet és légnyomás) továbbít.
4. A rendszer az adatokat tovább felhasználhatja (pl. időjárásfigyelés, tengerszint feletti magasság számítása).

D) Tervezésnek lépései:

1. BMP 180 szenzor légnyomás és hőmérséklet olvasási időd diagram:



S = Start
P = Stop
ACKS = Acknowledge by Slave
ACKM = Acknowledge by Master
NACKM = Not Acknowledge by Master

Magyarázat:

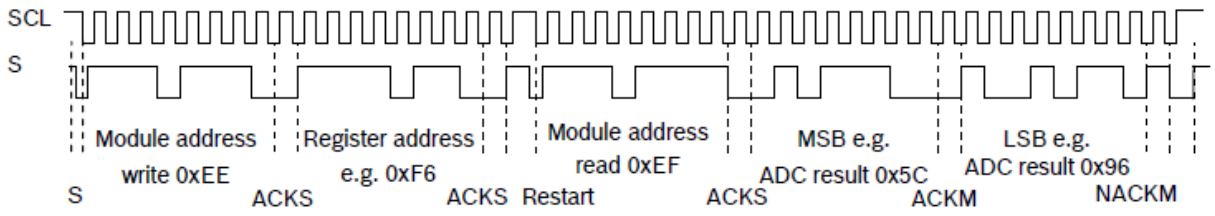
Az alábbiakban láthatók az időzítési diagramok a hőmérsékleti érték (UT) és a nyomásérték (UP) mérésének elindításához.

A kezdőfeltétel (start condition) után a mester (Master) elküldi az eszköz címét írási művelettel (write), a regiszter címét és a vezérlőregiszter adatát.

A BMP180 minden 8 adatbites blokk után küld egy megerősítést (ACKS - az érzékelő által küldött Acknowledge), amikor az adatot megkapta.

Az utolsó megerősítés (ACKS) után a mester stop feltételt küld.

2.BMP 180 szenzor A/D konverzión alapuló adatolvasási időd iagram:

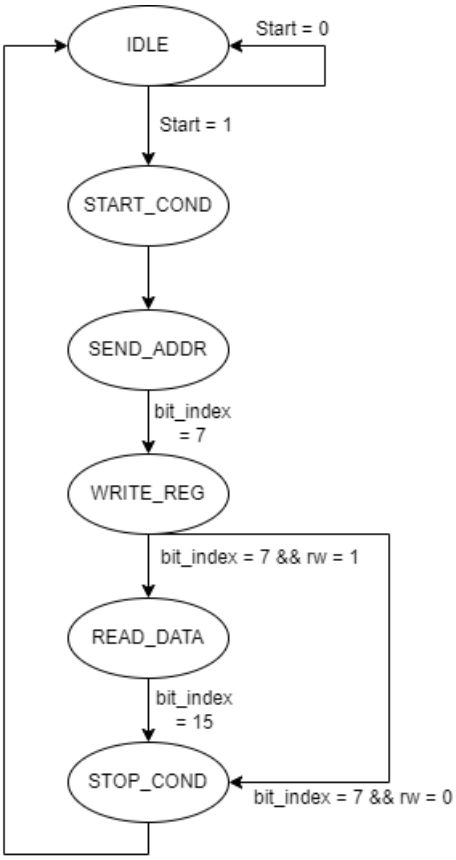


A fenti időd iagram működésének magyarázata:

Start feltétel és írási művelet indítása

- A kezdőfeltétel után a mester (Master) elküldi a modul címét **írási parancsként** és a regiszter címét.
- A regiszter cím választja ki az olvasni kívánt regisztert:
  - **EEPROM adatregiszterek**: 0xAA–0xBF
  - **Hőmérséklet vagy nyomás érték regiszterek (UT vagy UP)**:
    - 0xF6 (MSB - legmagasabb helyiértékű bájt),
    - 0xF7 (LSB - legalacsonyabb helyiértékű bájt),
    - **Opcionálisan**: 0xF8 (XLSB - extra bájt a nagyfelbontású olvasáshoz).
- **Újraindítási feltétel és olvasási művelet indítása**
- A mester küld egy **újraindítási feltételt**, majd elküldi a modul címét **olvasási parancsként**.
- Ezt a BMP180 érzékelő megerősíti (ACKS - Acknowledge by Sensor).
- A BMP180 először elküldi a 8 legmagasabb helyiértékű bitet (MSB), amelyet a mester megerősít (ACKM - Acknowledge by Master).
- Ezután az érzékelő elküldi a 8 legalacsonyabb helyiértékű bitet (LSB).
- A mester egy „nem megerősítés” jelet küld (NACKM - Not Acknowledge by Master), végül egy **stop feltételt**.
- **Ultra nagyfelbontású mód (opcionális)**
- Ha ultra nagyfelbontásra van szükség, a 0xF8 című XLSB regiszter is olvasható, amellyel a 16 bites adatszót akár 19 bitre is ki lehet terjeszteni.

3. Állapotdiagram Master-Slave:



I2C Master-Slave állapotdiagram

Az alábbi állapotdiagram a BMP 180 as légnyomásmérő szenzor időd iagramjából kiindulva van megvalositva a fenti állapotdiagram. Bemutatja a mester (Master) és a BMP180 szenzor közötti kommunikáció folyamatát I2C protokoll használatával. A folyamat a hőmérséklet vagy nyomásadatok kiolvasásához, illetve regiszterek írásához szükséges. Az alábbiakban az egyes állapotok magyarázata található:

IDLE (Készenléti állapot)
A rendszer várakozik, amíg a **Start** jel nem lesz aktív (**Start = 1**). Ebben az állapotban nincs adatforgalom az I2C buszon.

START_COND (Start feltétel)
Amikor a **Start** jel aktívvá válik (**Start = 1**), a rendszer létrehozza az I2C protokoll szerinti **start feltételt**, amely az adatkommunikáció kezdetét jelzi.

SEND_ADDR (Cím küldése)
A mester elküldi a BMP180 szenzor eszközcímét és az írási/olvasási műveletet jelző bitet. Az állapot addig tart, amíg az I2C cím 7 bitjének küldése be nem fejeződik (**bit_index = 7**).

WRITE_REG (Regiszter írása)
A mester elküldi a regiszter címét, amelyet írni vagy olvasni szeretne a BMP180-tól. Ha az olvasási műveletet jelző **rw = 1**, akkor az állapot a **READ_DATA** állapotba vált.

READ_DATA (Adatok olvasása)
Ha a mester olvasási műveletet hajt végre (**rw = 1**), akkor a BMP180 elküldi az adatokat a mester felé. Az adatok olvasása addig tart, amíg a teljes adat (16 bit) be nem érkezik (**bit_index = 15**).

STOP_COND (Stop feltétel)
A kommunikáció befejezésekor a mester létrehozza az I2C **stop feltételt**, amely az adatátvitel végét jelzi. Az állapot akkor érhető el, ha a mester befejezte az írási műveletet (**bit_index = 7 és rw = 0**), vagy az olvasási műveletet.

Visszatérés IDLE állapotba
Az adatátvitel befejezése után a rendszer visszatér az **IDLE** állapotba, és készen áll egy újabb kommunikációs ciklusra.

4. Táblázat állapotműveletek:

Az alábbi táblázat bemutatja a különböző állapotokban levő műveleteket, az állapotokban levő jelek változását.

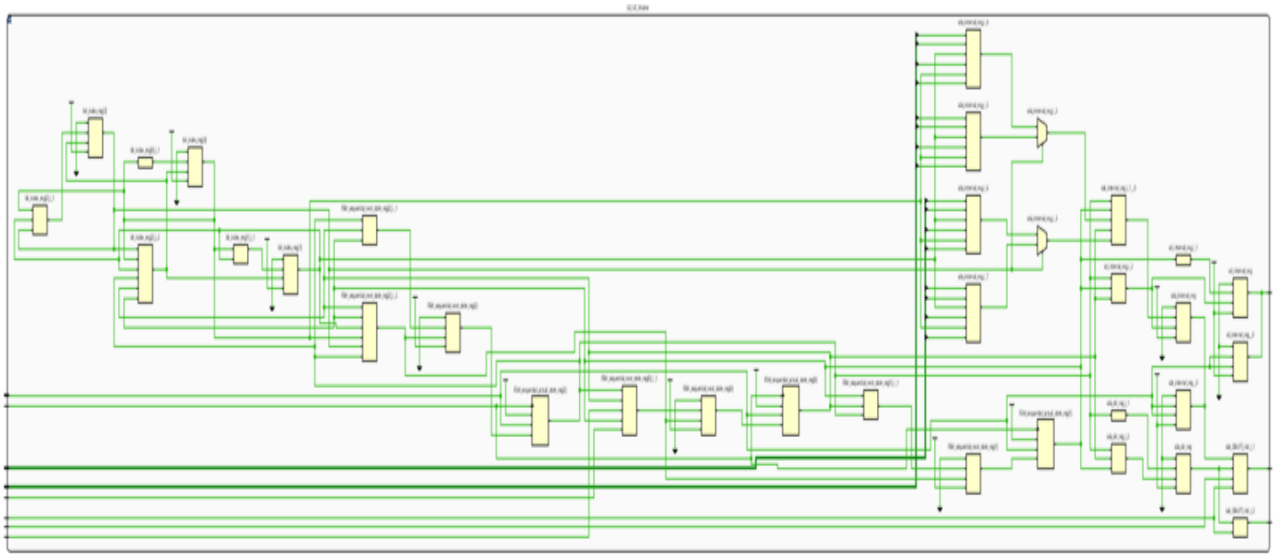
Állapot / Jelek	sda_internal	sda_dir	scl_internal	bit_index	temp_data
IDLE	X	X	X	X	X
START_COND	<= '0'	<= '1'	<= '1'	X	X
SEND_ADDR	<= addr(bit_index)	X	<= '0'	<= bit_index + 1	X
WRITE_REG	<= reg_addr(bit_index)	X	<= '0'	<= bit_index + 1	X
READ_DATA	X	<= '0'	X	bit_index + 1	bit_index = 15 => temp_data(bit_index) <= sda
STOP_COND	<= '1'	X	<= '1'	X	X

Táblázat állapotműveletekkel

5. Áramköri rajtok modulonként:

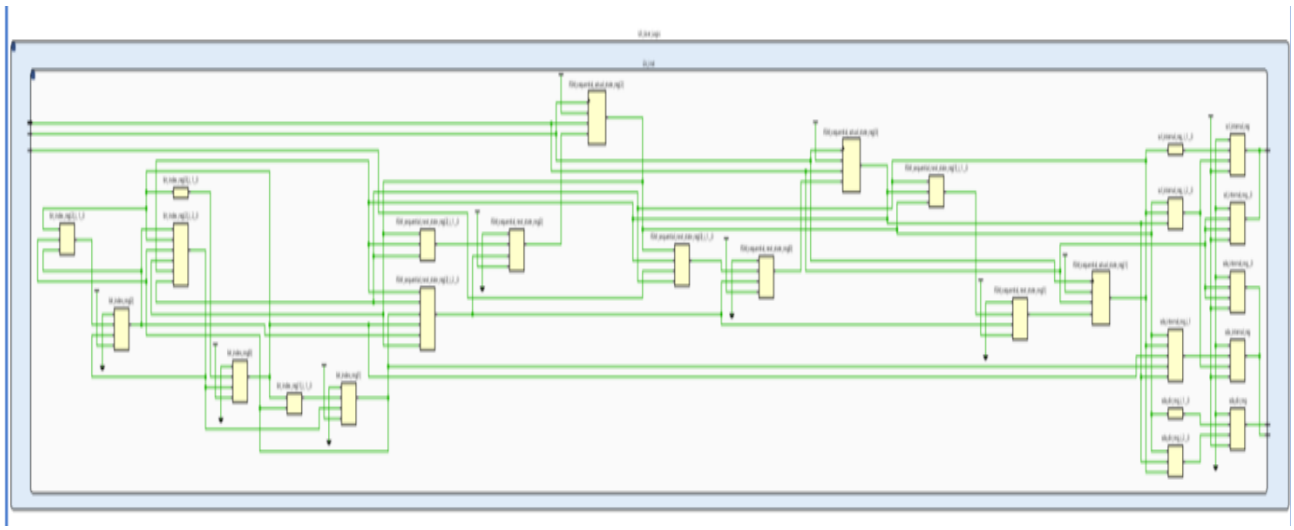
A fő modul aminek az időd iagram alapján elkészített állapotdiagram és tervezési fázisok megvalósultak az a tömbvázlatban látható I2C_Master -nek nevezett modul logikája valósítja meg az állapot logikát.
A továbbiakban felhasználva a Vivado fejlesztői környezetben található RTL Schematic – ot használva az alábbi áramköri rajzokat tudjuk felvázolni az egyes modulokra.

5.1 I2C_Master modul áramköri rajza:



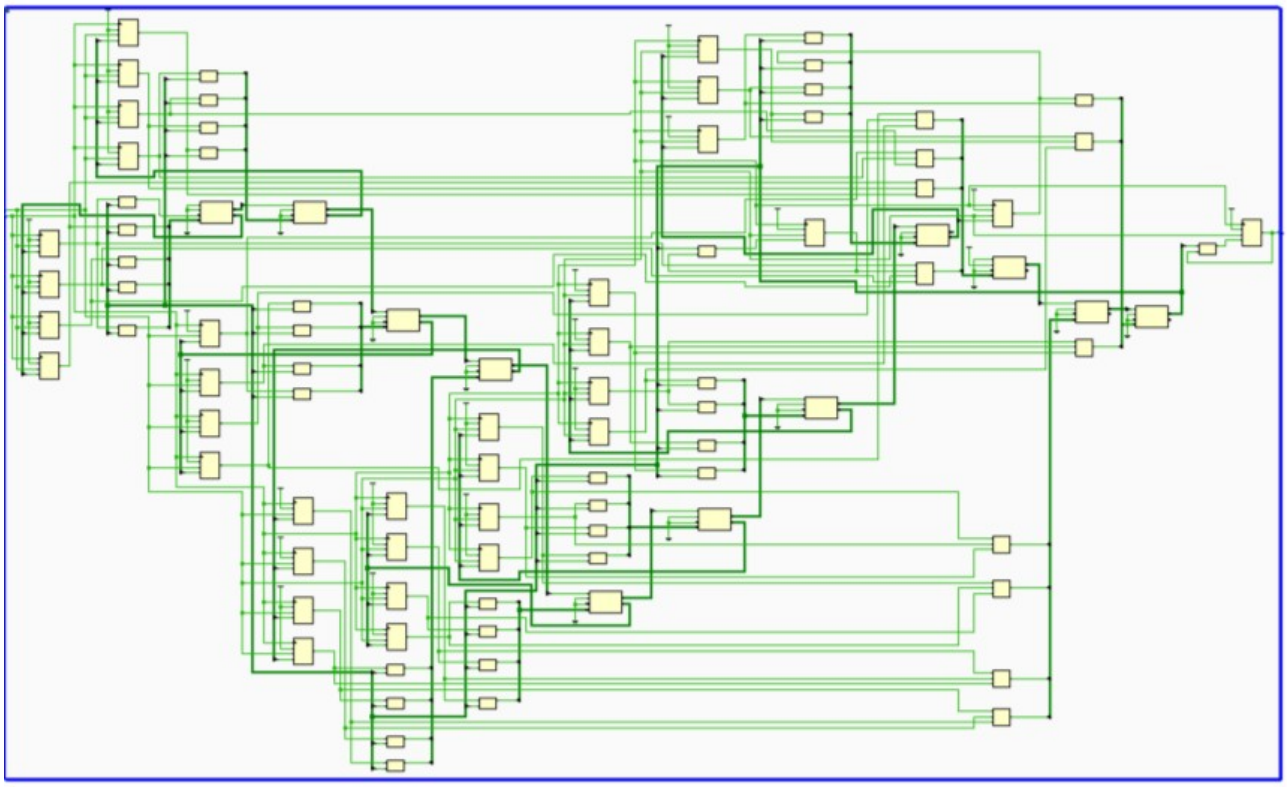
Az alábbi képen a részletek sajnos nem látszanak a magas képfelbontás miatt torzul a kép a részletes áramkör megtekintése érdekében az alábi I2C_Master_Schematic PDF -re kattintva látható a részletes ábra.

5.2 UserLogic (Felhasználoi logika) áramköri rajza:



Az alábbi képen a részletek sajnos nem látszanak a magas képfelbontás miatt torzul a kép a részletes áramkör megtekintése érdekében az alábi UserLogic_Schematic PDF -re kattintva látható a részletes ábra.

5.3 FrequencyDivider(Frekvencia Osztó) áramköri rajza:



Az alábbi képen a részletek sajnos nem látszanak a magas képfelbontás miatt torzul a kép a részletes áramkör megtekintése érdekében az alábi FrequencyDivider_Schematic PDF -re kattintva látható a részletes ábra.

6.VHDL kódok modulonként:

6.1 I2C_Master modul VHDL kódja és magyarázata:

Ez a VHDL kód egy I2C mester modul implementációja, amely egy állapotgép segítségével kommunikál egy I2C eszközzel (slave). A modul végrehajtja az I2C kommunikáció alapvető lépéseit: START feltétel, címzés, adatküldés/olvasás, STOP feltétel.

```
entity I2C_Master is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           start : in STD_LOGIC;
           scl : out STD_LOGIC;
           sda : inout STD_LOGIC;
           addr : in STD_LOGIC_VECTOR (7 downto 0);
           reg_addr : in STD_LOGIC_VECTOR (7 downto 0);
           data_in : out STD_LOGIC_VECTOR (15 downto 0);
           rw : in STD_LOGIC);
end I2C_Master;

architecture Behavioral of I2C_Master is

--allapotok inicializalasa
type state_type is (IDLE, START_COND, SEND_ADDR, WRITE_REG, READ_DATA, STOP_COND);
signal actual_state, next_state : state_type;

--SDA tri-state
signal sda_internal : STD_LOGIC :='1';
signal sda_dir : STD_LOGIC :='0';

--idoziteshez
signal scl_internal : STD_LOGIC :='1';

--kommunikacios jelek
signal bit_index : INTEGER range 0 to 7 :=0;
signal temp_data : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
```

Entitás Definíciója:

Az **I2C_Master** entitás bemeneti és kimeneti portokat határoz meg:

- **Bemenetek:**
  - clk: Az órajelforrás a működés szinkronizálásához.
  - reset: A modul resetelésére szolgáló jel.
  - start: A kommunikáció kezdeményezésére használt vezérlőjel.
  - addr: Az I2C slave eszköz címe (8 bit).

    - reg_addr: A slave eszköz regisztercíme (8 bit).
    - rw: Írás/olvasás vezérlőjel ('0': írás, '1': olvasás).
- **Kimenetek:**
    - scl: Az I2C órajel (Serial Clock Line).
    - sda: Az I2C adatvonal (kétirányú, inout port).
    - data_in: Az olvasott adatot tartalmazza (16 bit).

Tri-State SDA Vezérlés

- Az SDA vonalat két jel vezérli:
    - sda_internal: Az SDA kimeneti értéke.
    - sda_dir: Meghatározza az SDA vonal irányát ('1': kimenet, '0': bemenet).

Időzítéshez Kapcsolódó Jel

- Az scl_internal jel az I2C órajelet vezérli, és a protokoll időzítési követelményeit teljesíti.

Kommunikációs Jelek

- bit_index: A küldendő/olvasandó bit indexét tartja nyilván (0-tól 7-ig).
- temp_data: Az olvasott adat tárolására szolgáló 16 bites jel.

```
begin

    --SDA tri-state meghajtasa
    sda <= sda_internal when sda_dir = '1' else 'Z';
    scl <= scl_internal;

    --Allapotgep mukodese
    process(clk,reset)
    begin
        if reset = '1' then
            actual_state <= IDLE;
            scl_internal <= '1';
            sda_internal <= '1';

        elsif rising_edge(clk) then
            actual_state <= next_state;
        end if;
    end process;
```

Ha a reset jel aktív (reset = '1'), akkor:

- Az állapotgép visszaáll az **IDLE** (alapértelmezett) állapotba: actual_state <= IDLE.
- Az I2C protokollhoz kapcsolódó belső jelek is alaphelyzetbe kerülnek:
  - scl_internal <= '1';: Az órajel vonal (SCL) magas szintre kerül.
  - sda_internal <= '1';: Az adatvonal (SDA) magas szintre kerül.

- **Órajel Felfutó Él Észlelése:**
- Ha a **reset** nincs aktív, és az órajel felfutó élét észleli (**rising_edge(clk)**):
  - Az állapotgép a következő állapotba lép: actual_state <= next_state.

Állapotgép logikája:

```
--Allapotgep logikaja
process(actual_state, start, rw, addr, reg_addr, bit_index)
begin
    case actual_state is
        when IDLE =>
            if start = '1' then
                next_state <= START_COND;
            else
                next_state <= IDLE;
            end if;
        when START_COND =>
            -- START feltétel (SDA HIGH -> LOW, SCL HIGH)
            sda_internal <= '0';
            sda_dir <= '1';
            scl_internal <= '1';
            next_state <= SEND_ADDR;

        when SEND_ADDR =>
            --Slave cim kuldese
            sda_internal <= addr(bit_index);
            scl_internal <='0'; -- orajel lefuto el
            if bit_index = 7 then
                next_state <= WRITE_REG;
            else
                bit_index <= bit_index + 1;
            end if;
```

IDLE Állapot

- Várakozik, amíg a start jel aktívvá nem válik.
- Ha start = '1', az állapotgép a **START_COND** állapotba lép.

START_COND

- Létrehozza az I2C START feltételt:
  - sda_internal <= '0'; (SDA alacsonyra vált).
  - sda_dir <= '1'; (SDA kimenetként van beállítva).
  - scl_internal <= '1'; (SCL magas).
- Átlép a **SEND_ADDR** állapotba.

SEND_ADDR

- A slave címét bitenként küldi az SDA vonalon:
  - Az aktuális bit az addr(bit_index).
  - Az órajelet (scl_internal) vezérli, hogy a bit érvényes legyen.
- Ha az összes bit elküldésre került (bit_index = 7), az állapotgép a **WRITE_REG** állapotba lép.

```
when WRITE_REG =>
    --Regiszter cim elkuldese
    sda_internal <= reg_addr(bit_index);
    scl_internal <= '0';
    if bit_index = 7 then
        if rw = '1' then
            next_state <= READ_DATA;
        else
            next_state <= STOP_COND;
        end if;
    else
        bit_index <= bit_index + 1;
    end if;

  when READ_DATA =>
    sda_dir <= '0'; --SDA bemenet
    if bit_index = 15 then
        temp_data(bit_index) <= sda;
        next_state <= STOP_COND;
    else
        temp_data(bit_index) <= sda;
        bit_index <= bit_index + 1;
    end if;

    when STOP_COND =>
        sda_internal <= '1';
        scl_internal <= '1';
        next_state <= IDLE;

    when others =>
        next_state <= IDLE;

end case;

end process;
```

WRITE_REG

- A slave regisztercímét küldi el bitenként az SDA vonalon.
- Ha az összes bit elküldésre került:
    - Ha olvasási művelet (rw = '1'), átlép a **READ_DATA** állapotba.
    - Ha írási művelet, átlép a **STOP_COND** állapotba.

READ_DATA

- Az SDA vonalat bemenetként állítja be (sda_dir <= '0';).
- Bitenként olvassa az adatot a slave-től, és elmenti a temp_data jelbe.
- Ha az összes bit kiolvasásra került (bit_index = 15), átlép a **STOP_COND** állapotba.

STOP_COND

- Létrehozza az I2C STOP feltételt:
    - sda_internal <= '1'; (SDA magasra vált).
    - scl_internal <= '1'; (SCL magas).
- Az állapotgép visszatér az **IDLE** állapotba.

6.2 UserLogic (Felhasználoi logika) VHDL kódja és magyarázata:

Ez a kód a User_Logic modul logikáját valósítja meg, amely az I2C interfész segítségével kommunikál a BMP180 szenzorral, és a nyers hőmérséklet- és nyomásadatokat dolgozza fel a kimeneti jelekhez.

```
entity User_Logic is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           start : in STD_LOGIC;
           scl : out STD_LOGIC;
           sda : inout STD_LOGIC;
           temp_out : out STD_LOGIC_VECTOR (15 downto 0);  --homerseklet adat
           press_out : out STD_LOGIC_VECTOR (15 downto 0)); --legnyomas adat
end User_Logic;

architecture Behavioral of User_Logic is

--I2C interfesz jelei
signal addr : STD_LOGIC_VECTOR(7 downto 0) := "11101110"; --BMP180 szenzor cime ami a
signal reg_addr : STD_LOGIC_VECTOR(7 downto 0);
signal data_in : STD_LOGIC_VECTOR(15 downto 0);
signal rw : STD_LOGIC;

--adatok feldolgozasa

signal temp_raw : INTEGER;
signal press_raw : INTEGER;

component I2C_Master
    Port(
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           start : in STD_LOGIC;
           scl : out STD_LOGIC;
           sda : inout STD_LOGIC;
           addr : in STD_LOGIC_VECTOR (7 downto 0);
           reg_addr : in STD_LOGIC_VECTOR (7 downto 0);
           data_in : out STD_LOGIC_VECTOR (15 downto 0);
           rw : in STD_LOGIC
    );
end component;
```

Entitás (Entity) Definíció

Az entitás a modul bemeneti és kimeneti portjait határozza meg:

- **Bemenetek:**
  - clk: Órajelforrás.
  - reset: Reset jel.
  - start: A kommunikáció indításához szükséges vezérlőjel.
- **Kimenetek:**

- scl: Az I2C órajel (Serial Clock Line).
- sda: Az I2C adatvonal (Serial Data Line).
- temp_out: A kiolvasott hőmérséklet adat (16 bites).
- press_out: A kiolvasott nyomás adat (16 bites).

Belső Jelek

- **I2C interfész jelei:**
  - addr: Az I2C címe a BMP180 szenzornak (**0xEE**).
  - reg_addr: A regisztercím, amit a szenzornak kell küldeni.
  - data_in: Az I2C interfész által visszaküldött nyers adat.
  - rw: Az írás/olvasás mód jelzése ('0': írás, '1': olvasás).
- **Adatfeldolgozó jelek:**
  - temp_raw: Nyers hőmérsékletadat tárolására szolgáló **INTEGER**.
  - press_raw: Nyers nyomásadat tárolására szolgáló **INTEGER**.

I2C_Master Modul

A I2C_Master egy almodul, amely az I2C kommunikációt valósítja meg. Ezt a modul komponensként definiálták, és a fő User_Logic modul bekötötte az alábbi jelekkel:

- A modul az órajelet, resetet, start jelet, I2C címet, regisztercímet, adatvonalakat (sda, scl) és a nyers adatot (data_in) kezeli.
- Ez a modul felelős a BMP180 szenzorral való alacsony szintű kommunikációért.

```
begin

    --I2C master osszekapcsolasa
    i2c_inst : I2C_Master
        port map (
            clk => clk,
            reset => reset,
            start => start,
            scl => scl,
            sda => sda,
            addr => addr,
            reg_addr => reg_addr,
            data_in => data_in,
            rw => rw
        );

     --Adatok feldolgozasa
     process(clk, reset)
     begin
        if reset = '1' then
            temp_raw <= 0;
            press_raw <= 0;
        elsif clk' event and clk ='1' then
            temp_raw <= to_integer(signed(data_in(15 downto 8)));
            press_raw <= to_integer(signed(data_in(7 downto 0)));
        end if;
      end process;

    --Kimeneti adatok
    temp_out <= STD_LOGIC_VECTOR(to_unsigned(temp_raw, 16));
    press_out <= STD_LOGIC_VECTOR(to_unsigned(press_raw, 16));
```

Adatok Feldolgozása

- Egy **process** blokkon belül történik a nyers adatok feldolgozása.
- **Reset** esetén a nyers adatok nullára állnak (temp_raw, press_raw).
- Ha az órajel emelkedő élén (clk'event and clk='1'):
    - A kiolvasott 16 bites adat két részre bontva kerül feldolgozásra:
        - temp_raw: Az adat **felső 8 bitjéből** (15:8).
        - press_raw: Az adat **alsó 8 bitjéből** (7:0).
    - Az átalakítás során a **signed** függvény biztosítja a jelspecifikus konverziót, míg a **to_integer** az **INTEGER** típusra konvertálja az adatokat.

A nyers hőmérséklet- és nyomásadatokat visszafordítják STD_LOGIC_VECTOR típusúvá, hogy a megfelelő formátumban továbbítsák:

- temp_out: A temp_raw 16 bites unsigned reprezentációja.
- press_out: A press_raw 16 bites unsigned reprezentációja.

6.3 FrequencyDivider(Frekvencia Osztó) VHDL kódja és magyarázata:

```
entity FrequencyDivider is
    Port ( clk_in : in STD_LOGIC;
           reset : in STD_LOGIC;
           clk_out : out STD_LOGIC);
end FrequencyDivider;

architecture Behavioral of FrequencyDivider is

signal counter : INTEGER := 0;
signal temp_clk : STD_LOGIC := '0';
signal toggle_count : INTEGER := 30;

begin

process(clk_in, reset)
    begin
        if reset = '1' then
            counter <= 0;
            temp_clk <= '0';
            toggle_count <= 30;
        elsif rising_edge(clk_in) then
            if counter = toggle_count then
                counter <= 0;
                temp_clk <= not temp_clk;
                if toggle_count = 30 then
                    toggle_count <= 31;
                else
                    toggle_count <= 30;
                end if;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temp_clk;


end Behavioral;
```

Ez a VHDL kód egy frekvenciaosztó modult valósít meg. A modul a bemeneti órajel (clk_in) frekvenciáját osztja le egy alacsonyabb frekvenciájú kimeneti órajellé (clk_out). Ez azért fontos mivel ugyebár a szenzortol kapott adatokat az I2C adat buszon csak egy meghatározott megfelelő órajel szinkronizációval tudjuk elérni.

Bemenetek és kimenetek:

- **clk_in**: A magas frekvenciájú bemeneti órajel.
- **reset**: Az aszinkron visszaállító jel.
- **clk_out**: A modul által generált alacsonyabb frekvenciájú kimeneti órajel.
- **Belső jelek:**
- **counter**: Egy számláló, amely a bemeneti órajel ciklusait számolja.
- **temp_clk**: Egy ideiglenes jel, amely a kimeneti órajelet hozza létre.

- **toggle_count**: Az a maximális számláló érték, amely után a kimeneti órajelet meg kell változtatni.
- **Fő működési elv:**
- A modul egy **számlálót** használ, amely a bemeneti órajel ciklusait számolja.
- Ha a számláló eléri a **toggle_count** értéket, akkor:
    - A számláló alaphelyzetbe kerül (counter <= 0).
    - A **temp_clk** jel értéke megváltozik (flip-flop viselkedés: temp_clk <= not temp_clk).
    - A **toggle_count** értéke váltogatja a 30 és 31 értéket, hogy enyhén szabálytalan legyen az órajel (ez nem gyakori megoldás frekvenciaosztókban, de itt látható).
- **Reset kezelés:**
- Ha a **reset** jel aktív (reset = '1'), akkor:
    - A számláló (**counter**) visszaáll 0-ra.
    - A kimeneti órajel (**temp_clk**) alaphelyzetbe kerül ('0').
    - A **toggle_count** alapértéke 30-ra áll.
- **Órajel felfutó élén:**
- Ha a számláló eléri a **toggle_count** értéket:
    - Az órajelet váltogatja (flip-flop).
    - A számláló újraindul.
    - A **toggle_count** értéke váltogatja a 30 és 31 értékeket.
- Ha nem éri el a számláló a **toggle_count** értéket, akkor:
    - A számláló növekszik (counter <= counter + 1).
- **Kimeneti órajel:**
- A generált órajel (**clk_out**) a **temp_clk** értékére van csatolva: clk_out <= temp_clk.

E) Tesztelés

A tervezési fázis és implementációs fázis után ideje tesztelni, szimulálni a megvalósított alkalmazásunkat. Ahoz hogy tesztelni tudjunk egy megépitett rendszert ismerjük meg röviden, általánosan hogy milyen FPGA lapot használunk a tesztelés során.

NexysA7100T típusú fejlesztői FPGA lap általános rövid leirása:

A **Nexys A7-100T** fejlesztői lap egy fejlett, FPGA-alapú fejlesztőplatform, amelyet a Digilent gyártott. Kifejezetten oktatási célokra, digitális rendszerek tervezésére és beágyazott rendszerek fejlesztésére tervezték. Az eszköz ideális választás mind kezdők, mind tapasztalt fejlesztők számára, akik a digitális áramkörök, FPGA alapú rendszerek vagy processzor-implementációk területén szeretnének dolgozni.

Főbb jellemzők:

1. **FPGA chip**:
    - A Nexys A7 kétféle változatban érhető el:
        - **Nexys A7-50T**: 50K logikai cellával.
        - **Nexys A7-100T**: 100K logikai cellával.
    - Mindkét verzió alapja a **Xilinx Artix-7 FPGA**, amely kiemelkedő teljesítményt nyújt alacsony energiafogyasztás mellett.
    - A fejlesztők számára elérhető a **Vivado Design Suite** fejlesztői környezet, amely támogatja az eszközt.
2. **Memória**:
    - **DDR3L RAM**: 128 MB nagy sebességű memória komplex rendszerek számára, például beágyazott processzorokhoz.
    - **Quad-SPI Flash**: 16 MB Flash memória, amely lehetővé teszi a konfigurációk és programok tárolását.
3. **Csatlakozók és interfészek**:
    - **USB-JTAG programozás**: Az eszköz közvetlenül csatlakoztatható számítógéphez USB-n keresztül a konfigurációhoz és programozáshoz.

- **UART-USB**: Soros kommunikációhoz.
- **PMOD csatlakozók**: További perifériák és szenzorok csatlakoztatásához (különösen Digilent PMOD modulokkal).
- **VGA kimenet**: Alapvető vizuális kijelző projektekhez.
- **Ethernet interfész**: Hálózati alkalmazások fejlesztéséhez.
- **Audio I/O**: Mikrofon és fejhallgató csatlakoztatásához.

4. **Perifériák**:
    - **LED-ek**: 16 db általános célú LED a státuszjelzésekhez.
    - **Nyomógombok és kapcsolók**: 5 db gomb és 16 db DIP kapcsoló, amelyekkel könnyen implementálhatók bemeneti interfészek.
    - **7-szegmenses kijelző**: Négy darab 7-szegmenses kijelző számok és szimbólumok megjelenítéséhez.
    - **Órajelforrás**: 100 MHz-es alapórajel-kristály, amely az FPGA számára biztosít stabil időzítést.
5. **Energiaellátás**:
    - Az eszköz táplálható **USB-ről** vagy külső tápegységről.
    - Beépített feszültségszabályozó biztosítja az FPGA és perifériák számára szükséges áramellátást.
6. **Szoftvertámogatás**:
    - **Xilinx Vivado**: Tervezési és szintetizálási eszköz VHDL, Verilog és más HDL nyelvekhez.
    - Támogatás **beágyazott processzorok** implementálására, például MicroBlaze soft processor használatával.

Előnyei:

- Kompakt és jól integrált fejlesztői lap, amely számos interfésszel és perifériával rendelkezik.
- Ideális platform kezdők számára az egyszerű hardveres tervezési projektekhez, ugyanakkor kellően erős bonyolultabb rendszerek implementálásához is.
- Erőteljes Artix-7 FPGA, amely alacsony energiafogyasztást és nagy teljesítményt kínál.

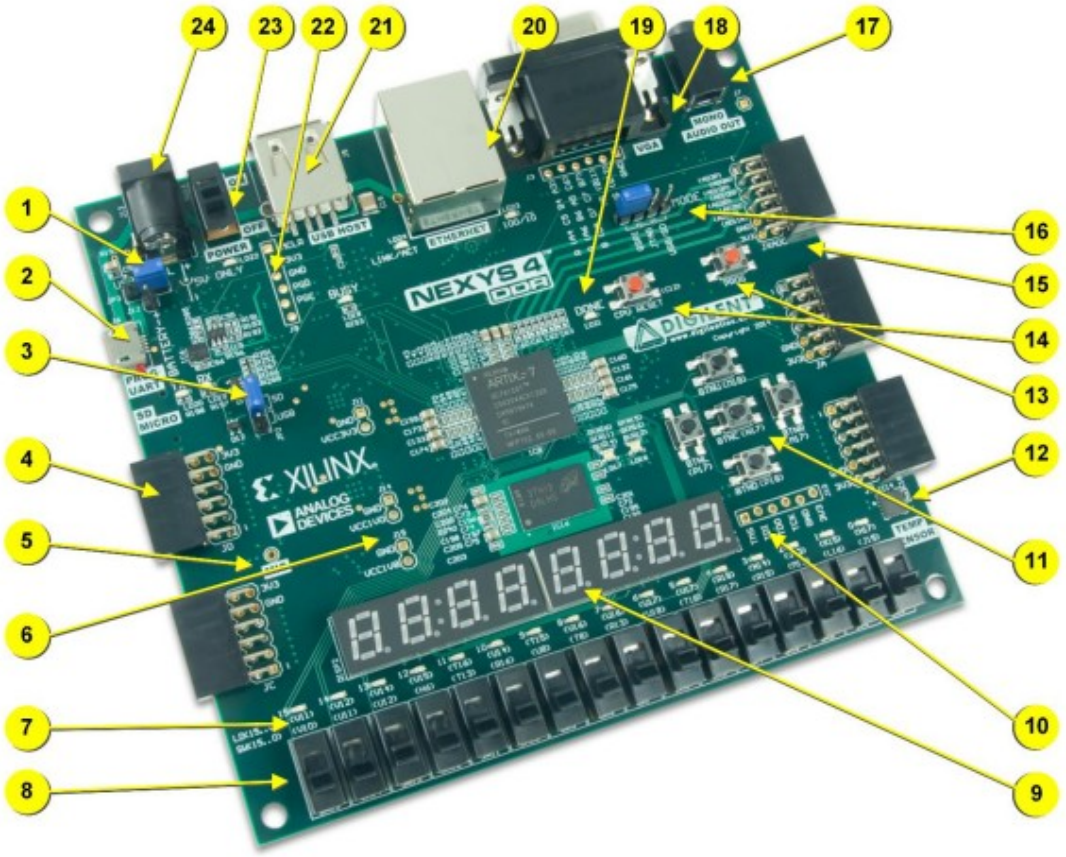


Figure 1. Nexys4 DDR board features.

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod connector (XADC)
4	Pmod connector(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

További részletes leírást a Digilent oldalán található pdf formátumu adatlapon lehet utánajárni erre kattintva megtalálható az adatlap : **NexsysA7100T adatlap**

1.Szimuláció:

Ez a VHDL program egy tesztpadot (testbench) definiál a Top_Level nevű modul szimulációjához. A tesztpad célja a Top_Level entitás viselkedésének szimulálása és ellenőrzése egy mesterséges környezetben, ahol különböző bemeneti jeleket adunk, és figyeljük a kimeneteket.

```
entity Top_Level_TB is
--  Port ( );
end Top_Level_TB;

architecture Behavioral of Top_Level_TB is

    component Top_Level
        Port(
           clk_in : in STD_LOGIC;
           reset : in STD_LOGIC;
           start : in STD_LOGIC;
           rw : in STD_LOGIC;
           addr : in STD_LOGIC_VECTOR (7 downto 0);
           reg_addr : in STD_LOGIC_VECTOR (7 downto 0);
           scl : out STD_LOGIC;
           sda : inout STD_LOGIC;
           data_in : out STD_LOGIC_VECTOR (15 downto 0);
           temp_out : out STD_LOGIC_VECTOR (15 downto 0);
           press_out : out STD_LOGIC_VECTOR (15 downto 0)
        );
     end component;


     -- Bemeneti jelek
    signal clk_in    : STD_LOGIC := '0';
    signal reset     : STD_LOGIC := '1';
    signal start     : STD_LOGIC := '0';
    signal rw        : STD_LOGIC := '0';
    signal addr      : STD_LOGIC_VECTOR (7 downto 0) := "11101110"; -- I2C eszköz címe (0xEE)
    signal reg_addr  : STD_LOGIC_VECTOR (7 downto 0) := "11110110"; -- Regiszter címe
    signal scl       : STD_LOGIC;
    signal sda       : STD_LOGIC := 'Z';
    signal data_in   : STD_LOGIC_VECTOR (15 downto 0);
    signal temp_out  : STD_LOGIC_VECTOR (15 downto 0);
    signal press_out : STD_LOGIC_VECTOR (15 downto 0);

    -- Órajel periódusa
    constant clk_period : time := 10 ns;
```

Entitás definiálása:

Az entitás neve **Top_Level_TB**, amely nem tartalmaz portokat, mivel egy tesztpad általában önálló modul. Minden tesztbemenetet és kimenetet belső jelek szimulálnak.

Top_Level komponens deklarációja:

A tesztpad egy másik modul, a **Top_Level** entitás szimulációját végzi. A **Top_Level** portjai itt vannak deklarálva, hogy a tesztpad hozzá tudjon férni azokhoz.

A Top_Level entitás portjai a következők:

- **Bemenetek**: clk_in, reset, start, rw, addr, reg_addr.
- **Kimenetek**: scl, sda (bidi), data_in, temp_out, press_out

Jelek és paraméterek inicializálása:

A tesztpad szimulált bemeneti jeleit és egyéb szükséges változókat definiálja:

- **Órajellel kapcsolatos jelek**:
  - clk_in: A rendszer órajelét szimulálja.
  - **clk_period:** Az órajel periódusideje 10 ns.
- **Rendszer-specifikus jelek**:
  - reset: A rendszer inicializálásához.
  - start: Az I2C kommunikáció indítására.
  - rw: Olvasási (1) vagy írási (0) művelet kiválasztása.
  - addr: Az I2C eszköz címe (például egy szenzoré).
  - reg_addr: A célregiszter címe.
- **I/O jelek**:
  - scl: Az I2C órajel vonala.
  - sda: Az I2C adatvonal (kétirányú).
  - data_in, temp_out, press_out: Az adatátvitelhez és kimeneti ellenőrzéshez használt jelek.

```
begin

    UUT : Top_Level
        port map (
            clk_in      => clk_in,
            reset       => reset,
            start       => start,
            rw          => rw,
            addr        => addr,
            reg_addr    => reg_addr,
            scl         => scl,
            sda         => sda,
            data_in     => data_in,
            temp_out    => temp_out,
            press_out   => press_out
        );

    -- Orajel generalasa
    clk_process : process
    begin
        while true loop
            clk_in <= '0';
            wait for clk_period / 2;
            clk_in <= '1';
            wait for clk_period / 2;
        end loop;
    end process;
```

A Top_Level modul példánya:

A tesztpad instanciálja a Top_Level modult a **UUT (Unit Under Test)** névvel. Ez a modul az összes szimulált jelhez hozzárendelhető a portokon keresztül, így lehetővé válik a modul működésének tesztelése.

Órajel generálása:

Az **clk_process** folyamatosan szimulálja az órajelet:

- Fél periódusonként (clk_period / 2) a clk_in jel állapota változik (0 → 1, majd 1 → 0).

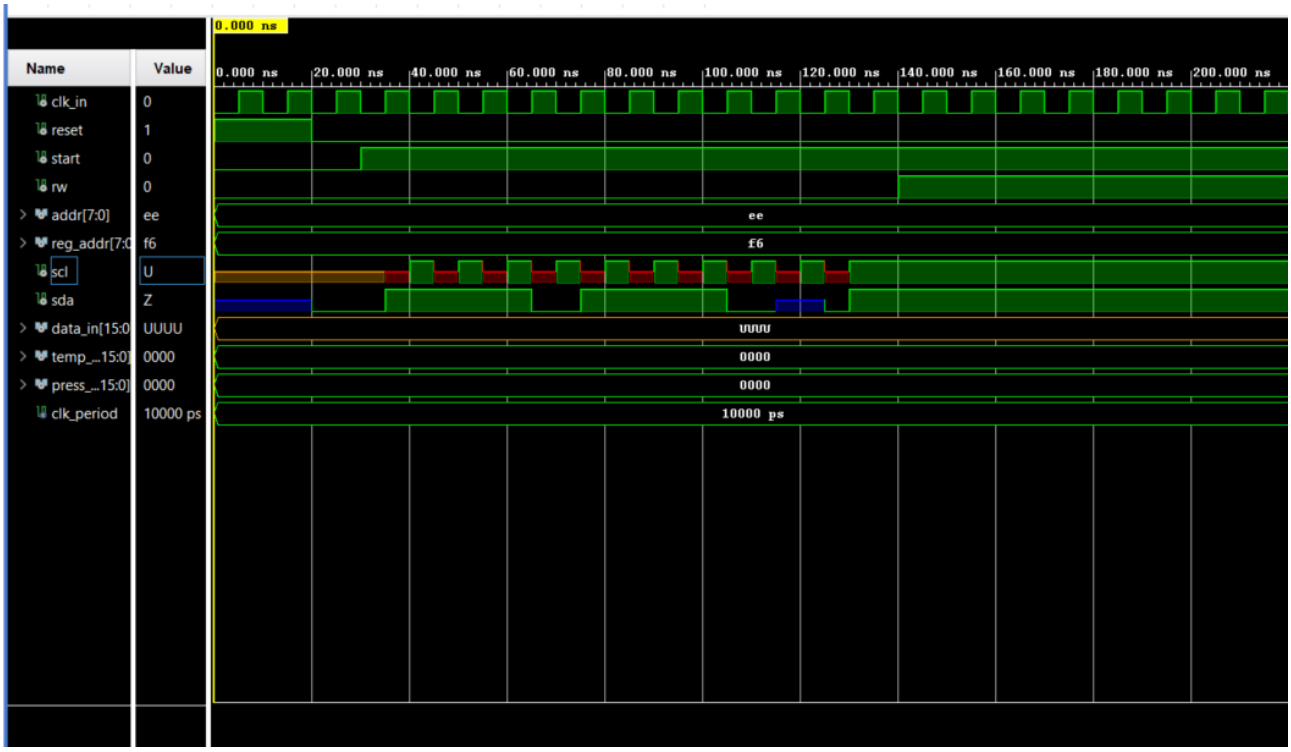
```
stimulus_process : process
begin
    -- Rendszer reset
    reset <= '1';
    wait for 20 ns;
    reset <= '0';

    -- START feltétel (SDA lemegy, SCL magas)
    sda <= '0';
    wait for clk_period;
    start <= '1';

    -- Adat elküldése bitenként (8 bit)
    for i in 0 to 7 loop
        wait for clk_period / 2;  -- SCL low
        scl <= '0';
        sda <= addr(7 - i);  -- Adat bit beállítása
        wait for clk_period / 2;  -- SCL high
        scl <= '1';
    end loop;

    -- Acknowledge bit (slave SDA lehúzása)
    wait for clk_period / 2;
    scl <= '0';
    sda <= 'Z'; -- Slave felelős az SDA-ért
    wait for clk_period / 2;
    scl <= '1';

    -- STOP feltétel (SDA felmegy, SCL magas)
    wait for clk_period / 2;
    scl <= '0';
    sda <= '0';
    wait for clk_period / 2;
    scl <= '1';
    sda <= '1';
    wait for clk_period;

    -- Tesztelés olvasási és írási üzemmódban
    rw <= '1'; -- Olvasás
    wait for 500 ns;

    rw <= '0'; -- Írás
    wait for 500 ns;

    -- Szimuláció vége
```

Stimulus folyamat:

Az **stimulus_process** a tesztelési szimuláció fő része, amely a következő műveleteket végzi:

- **Rendszer inicializálása**:
    - A **reset** jel 1-es állapotban indul, hogy alaphelyzetbe állítsa a rendszert. Ezt 20 ns után 0-ra állítja, hogy elindítsa a szimulációt.
- **I2C START feltétel**:
    - Az I2C protokoll szerint az adatvonal (**SDA**) "magasból" ("1") "alacsonyba" ("0") megy, miközben az órajel (**SCL**) magas ("1").
- **I2C cím küldése**:
    - Egy ciklus iteráció során (for i in 0 to 7) az I2C cím bitenként kerül kiküldésre az **addr** jel alapján. Az órajel (**SCL**) minden bit küldésekor alacsony és magas szint között változik.
- **Acknowledge (ACK) bit kezelése**:
    - A cím kiküldése után az **SDA** vonalat a slave eszköz húzza le ('Z' állapot az SDA-nál a master részéről).
- **I2C STOP feltétel**:
    - Az I2C protokoll szerint az **SDA** magasba ("1") kerül, miközben az **SCL** is magas ("1").
- **Tesztelés olvasási és írási műveletekkel**:
    - A **rw** jel 1-re állítása elindít egy olvasási műveletet.
    - A **rw** jel 0-ra állítása egy írási műveletet szimulál.
- **Szimuláció vége**:
    - A wait parancs a szimulációt futva hagyja.

2. Szimulációs eredmény időd iagramja:



Az időd iagram az előzőleg bemutatott tesztpad (testbench) szimulációs eredményeit mutatja, különösen az I2C kommunikációs protokollra vonatkozóan.

Az órajel

- Az órajel (óra) periódusa 10 ns, ahogy a tesztpadban definiáltuk (clk_period = 10 ns).
- Ez a jel meghatározza az I2C kommunikáció időzítését, például a bitküldést és a többi szinkronizált eseményt.

reset: Reset jel

- A reset jel kezdetben 1 (20 ns-ig), majd 0-ra vált.
- A reset jel a rendszer inicializálását szimulálja. Ezért a kezdeti időszakban a többi jel nincs aktív állapotban.

start: Kommunikáció indítása

- A **start** jel 0-ból 1-be vált az 50 ns környékén.

- Ez jelzi az I2C kommunikáció kezdetét, ahol a START feltétel teljesül: az SDA (adatvonal) leesik (0-ra vált), miközben az SCL (órajel) magas szinten van.

addr: I2C cím

- Az addr jel értéke ee hexadecimális formátumban, amely az I2C eszköz címét adja meg. Ez az érték az SDA vonalon kerül bitenként elküldésre.

reg_addr: Regisztercím

- A reg_addr jel értéke f6 hexadecimális formátumban, amely a célregiszter címét jelzi, ahová vagy ahonnan adatot szeretnénk küldeni vagy olvasni.

scl: Órajel vonal (I2C)

- Az **SCL** jel az I2C órajelet képviseli.
- Az idődiagramon jól látszik az SCL vonal periodikus változása. Ez a jel szinkronizálja az adatküldést az SDA vonalon.
- Amikor az SCL magas, az SDA vonal állapota az adott bit értékét tükrözi.

sda: Adatvonal (I2C)

- Az SDA vonal kezdetben lebegő állapotban van (Z), amely az I2C szabvány szerint az alaphelyzet.
- A START feltétel után (50 ns körül) az SDA vonal aktívvá válik.
- Az idődiagramon látszik, hogy az SDA vonalon a ee és f6 címek bitenként kerülnek kiküldésre, szinkronban az SCL vonallal.

data_in: Bemeneti adat

- A data_in jel értéke UUUU, ami azt jelzi, hogy még nem történt adatbevitel. Ez a jel a későbbi olvasási műveletek során frissülhetne.

temp_out, press_out: Kimeneti adatok

- A temp_out és press_out jelek értéke 0000, ami azt jelenti, hogy még nem történt eredménykiolvasás.

STOP feltétel

- A STOP feltétel a szimuláció során is teljesül: az SDA vonal alacsony szintről magas szintre vált, miközben az SCL magas szinten van. Ez a STOP feltétel szimbolizálja az I2C kommunikáció végét.

3. Működés közbeni viselkedés:

4. Mérések:

F) Bibliográfia:

1. Brassai, Sándor Tihamér : **Újrakonfigurálható digitális áramkörök tervezési és tesztelési módszerei**

2. **NexsysA7100T datasheet**

3. Előadás jegyzet, Labor jegyzet