FPGA tervezői laboratórium

Tartalom

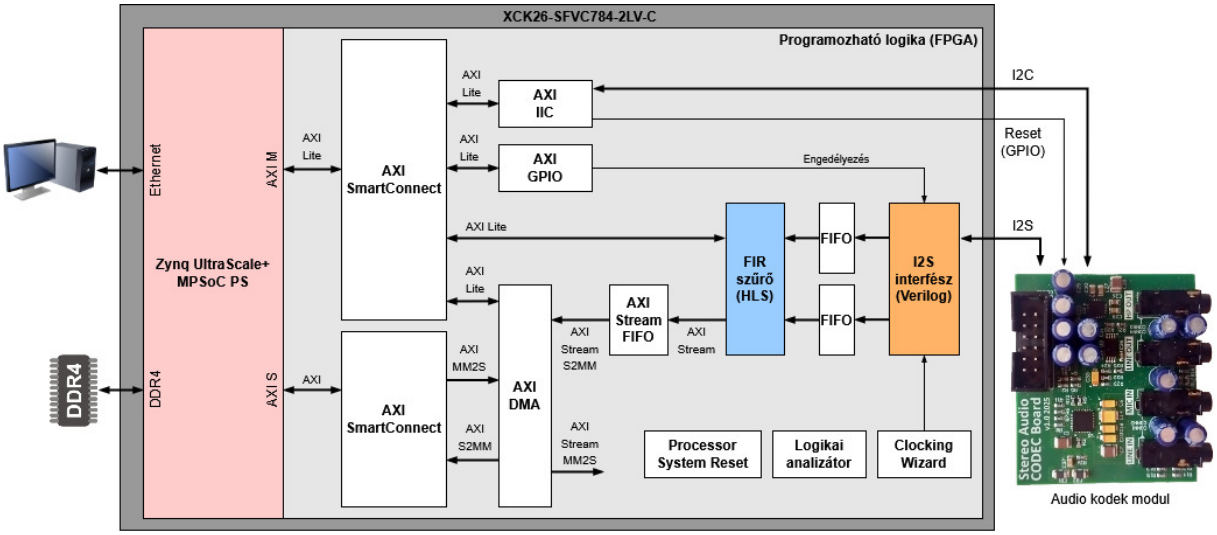
1 Bevezetés

Az FPGA tervezői laboratórium célja egy komplex, FPGA áramkör alapú SoC (System-on-Chip) rendszer megvalósítása a Kria KV260 Vision AI Starter Kit fejesztői kártyán. A rendszer tartalmaz egy ARM Cortex-A55 alapú mikroprocesszoros részt (ez a Zynq UltraScale+ MPSoC-ban rendelkezésre áll a szilíciumon), valamint az FPGA általános erőforrásaiból kialakított perifériákat.

Az első részben elkészítendő rendszer egy Ethernet interfésszel rendelkező, kétcsatornás analóg adatgyűjtő eszköz, melynek végleges változatának egyszerűsített blokkvázlatát az 1. ábra mutatja.

A **Zynq UltraScale+ MSoC Processing System** a rendelkezésre álló (hard-core) mikroprocesszoros alrendszer, mely többek között DDR memóriavezérlőt és Ethernet MAC egységet is tartalmaz. A processzoros alrendszerhez az AXI Lite interfészen (SmartConnect) keresztül a következő lassú, regisztervezérelt perifériák kapcsolódnak:

- AXI I2C periféria az audio kodek programozásához és a reset jelének előállításához
- AXI GPIO periféria Verilog hardverleíró nyelven megvalósított I2S interfész engedélyezéséhez
- AXI DMA vezérlő regiszterei
- FIR szűrő konfigurációs regiszterei



1. ábra: A teljes megvalósítandó rendszer blokkvázlata.

A processzoros rész nagyteljesítményű slave AXI portjához kapcsolódik továbbá egy AXI DMA vezérlő, melynek segítségével nagysebességű adatátvitel valósítható meg a processzoros rész és az általános FPGA logika között.

Az FPGA általános erőforrásait felhasználva a következő részegységek kerülnek kialakításra:

- **Clocking Wizard**: a processzoros alrendszer 100 MHz-es órajeléből a sztereó audio kodek vezérléséhez szükséges órajelet állítja elő.
- **I2S interfész**: az FPGA-hoz kapcsolódó sztereó audio kodek soros adat interfésze, ez állítja elő a szűrő párhuzamos bemeneti adatait.
- **FIR szűrő**: Vitis HLS-ben létrehozott almodul, amely IP-ként beilleszthető a „Block Design”-ba. Két csatornás szűrő, mely képes 1-szeres (áteresztő), 2-szeres, valamint 4-szeres decimálást végezni két bemeneti csatornáján. A szűréshez használt együttható készlet, a szűrő fokszáma (TAP szám), valamint a decimálási faktor a processzor oldaláról módosítható paraméterek. A szűrő bemeneti adatai csatornánként egy-egy natív interfésszel rendelkező FWFT FIFO-ból kerülnek beolvasásra. A kimeneti adatok ugyancsak egy FIFO-ba kerülnek, ennek viszont mind

az írási, mind pedig az olvasás oldala AXI Stream típusú. Az ehhez szükséges vezérlőjeleket a HLS kód állítja elő.

- FPGA-ban megvalósított logikai analizátor debug célokra.

Mivel a használt fejlesztői kártya nem tartalmaz a feladatok megoldásához szükséges egyszerű perifériákat (LED-ek, nyomógombok) és audio kodeket, ezért ezeket külön modulként kell illeszteni a rendszerhez a kártya PMOD csatlakozóján keresztül. A modulok jeleinek hozzárendelését az FPGA lábakhoz az 1. táblázat tartalmazza. Minden jel LVCMOS33 I/O szabványú.

1. táblázat: A modul jelek és az FPGA lábak összerendelése

PMOD csatlakozó láb	FPGA láb	LED és nyomógomb modul jelek	Audio kodek modul jelek
1	H12	LED0	I2C SCL
2	B10	BTN0	I2C SDA
3	E10	LED1	SDOUT (soros adat az FPGA felé)
4	E12	BTN1	SDIN (soros adat az FPGA felől)
5	D10	LED2	LRCLK
6	D11	BTN2	SCLK
7	C11	LED3	MCLK
8	B11	BTN3	RESETn

2 Egyszerű processzoros rendszer létrehozása (1. hét)

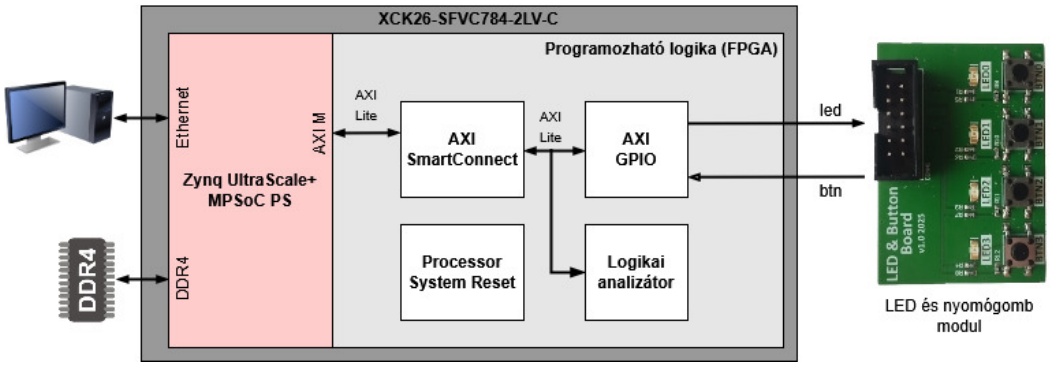
A feladat megoldását az alábbi információk segítik:

- **Az FPGA alapú rendszerek tárgy előadásainak és gyakorlatainak anyaga közül az AXI interfész, a Vivado logikai analizátor ismertető és gyakorlat, valamint a DMA adatátvitel**
- **ARM AMBA AXI specifikáció A2.1 - A2.6 része**

Ez a feladat nagyrészt vezetett mérés jellegű, a Vivado és a Vitis fejlesztői környezettel való ismerkedésre szolgál. A továbbiak ezzel ellentétben valós tervezési feladatok, azaz nagyrészt önálló munkára, alkotó tevékenységre épülnek.

Hozzon létre egy új projektet a Vivado fejleszői környezetben és az eszközöknél válassza ki a ***Kria KV260 Vision AI Starter Kit SOM*** fejlesztői kártyát. A kártya kiválasztásánál látható egy Connections felirat, amelyre kattintva a megjelenő ablakban válassza ki a ***Vision AI Starter Kit carrier Card*** lehetőséget (ennek kiválasztása lehetőséget biztosít, hogy egyes perifériák portjait a kártyán lévő eszközökhöz rendeljük).

Az üres projekthez adjon hozzá egy block design-t cpu_system néven, amelybe a 2. ábrán látható rendszer kerül megvalósításra.



2. ábra: A feladatban megvalósítandó rendszer blokkvázlata.

A block design-ba helyezze el a következő egységeket:

- **Zynq UltraScale+ MPSoC** processzoros alrendszer. A megjelenő ***Run Block Automation*** lehetőségnél állítsa be a fejlesztői kártyához tartozó beállításokat (board preset). A PS-PL interfészeknél szükség van egy 32 bites AXI master kapcsolatra (AXI HPM0 FPD). Megszakításkérő bemenetre most nincs szükségünk. A pl_clk0 órajel kimenetet kösse be az AXI interfész órajel bemenetére.
- **Processor System Reset** modul, amely előállítja a perifériák és az AXI SmartConnect-ek számára a reset jelet. Ebbe kösse be a processzoros alrendszer órajel és reset kimeneteit.
- **AXI SmartConnect** az AXI Lite interfészek számára. Állítsa be a slave és maser portok szükséges számát, majd kösse be a processzoros alrendszer AXI HPM0 FPD interfészét, valamint az órajelet és az interconnect reset jelet.
- **AXI GPIO** periféria a LED-ek és a nyomógombok ilesztéséhez. Kösse be az AXI Lite interfészt, az órajelet és a periféria reset jelet. A GPIO periféria első csatornáját vezesse ki led nevű külső portként a LED-ek illesztéséhez, a második csatornát pedig btn nevű külső portként a nyomógombok illesztéséhez.
- **System ILA** (logikai analizátor) a GPIO periféria AXI interfészének vizsgálatához. A mintatárat állítsa be 1024 mélységűre, a komparátorok számát 2-re, valamint engedélyezze a capture control és az advanced trigger lehetőségeket.

Miután elkészült a block design, az Address Editor-ban rendeljen címtartományokat a perifériákhoz. A DMA vezérlő AXI master interfészei a külső DDR4 memóriát érik el, így ezek címtartományából a QSPI interfész kizárható (exclude).

A block design validálása után készítsen hozzá egy HDL wrapper-t, amely a rendszer top-level modulja lesz. Exportálja a hardvert a Vitis számára a konfigurációs bitfolyam nélkül.

Miközben zajlik a szintézis és az implementáció, a Vitis fejlesztői környezetben készítsen egy egyszerű C nyelvű szoftvert, amely a nyomógombok állapotát megjeleníti a LED-eken. A szintézis befejeződése után ne felejtse el a GPIO portokhoz hozzárendelni a megfelelő FPGA lábakat (szintetizált rendszer megnyitása, I/O Ports ablak), amelyek a bevezető részben lévő 1. táblázatban találhatók meg.

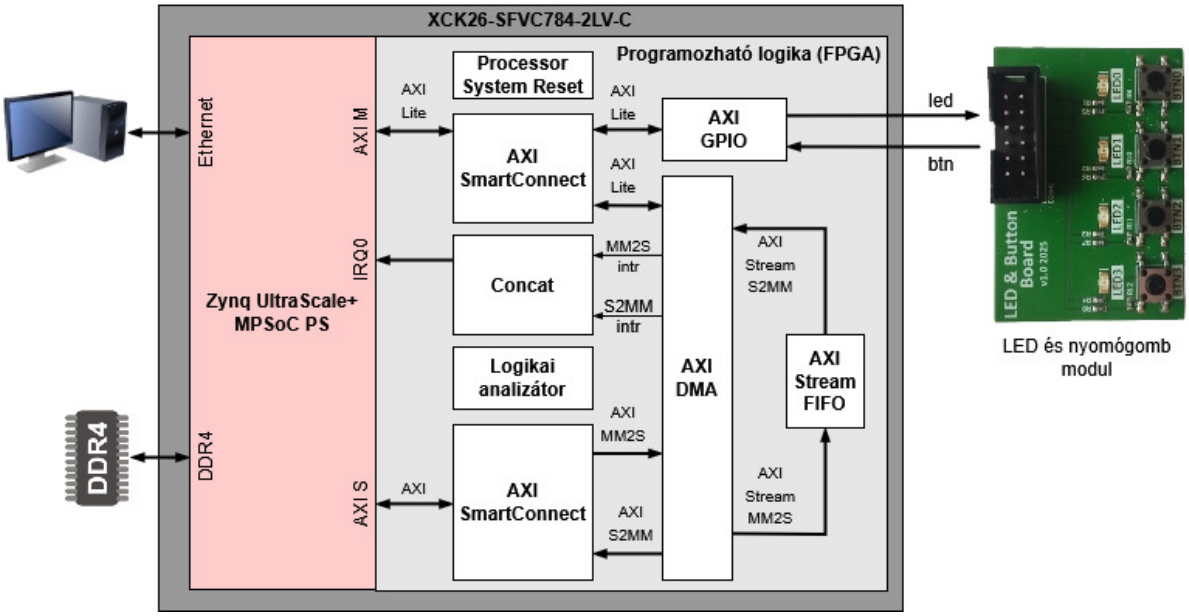
Az implementáció befejeztével konfigurálja az FPGA-t, majd töltse le a megírt szoftvert debug módban és próbálja ki a működést. A szoftver letöltése előtt állítsa be a futtatási konfigurációt a ***Hasznos információk*** pontban leírtaknak megfelelően. Vizsgáljon meg egy AXI Lite írást és egy AXI Lite olvasást a logikai analiztor segítségével.

3 DMA átvitel megvalósítása és tesztelése (2. hét)

A feladat megoldását az alábbi információk segítik:

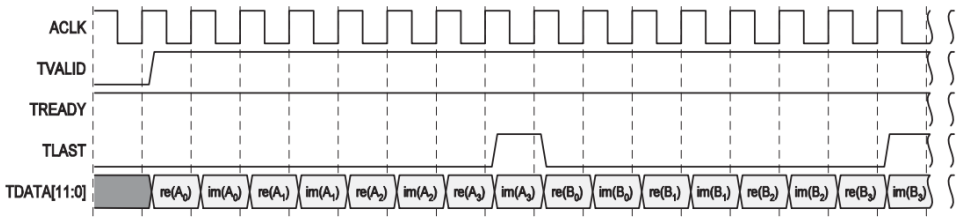
- **Az FPGA alapú rendszerek tárgy előadásainak és gyakorlatainak anyaga közül az AXI interfész, a Vivado logikai analizátor ismertető és gyakorlat, valamint a DMA adatátvitel**
- **ARM AMBA AXI specifikáció A2.1 - A2.6 része**
- **Xilinx AXI DMA Product Guide (PG021)**

Ebben a feladatban kiegészítjük a korábban megvalósított processzoros rendszert DMA adatátviteli képességgel. Ehhez a Vivado környezetben rendelkezésre álló AXI DMA vezérlőt használjuk, amelynek AXI Memory Mapped (MM) és AXI Stream interfészei vannak. Az adatátvitel teszteléséhez egy AXI Stream FIFO-t fogunk használni, melyet a DMA vezérlő képes írni és olvasni, azaz egy DMA vezérlő által végrehajtott adatmozgatást tudunk megvalósítani a processzoros rendszerhez kapcsolódó DDR memóriában. A módosítás utáni rendszer a 3. ábrán látható.



3. ábra: A DMA vezérlővel kiegészített rendszer blokkvázlata.

Az AXI DMA egység a felhasználó felé AXI Stream interfészt biztosít, ami egy egyszerű csomag (packet) alapú, adatfolyam jellegű interfész. Ennek jellemző időzítési diagramját a 4. ábra mutatja.



4. ábra: AXI Stream adatátvitel.

Az AXI Stream interfész esetén a kötelező jelek az alábbiak:

- TVALID: az adat forrás kimenete érvényes
- TREADY: az adat nyelő képes adatot fogadni
- TLAST: az adatcsomag utolsó szavát jelzi
- TDATA: adatbusz

- TKEEP: bájt érvényes jelzés. Ha a teljes 32 bites adat érvényes, értéke 4'b1111. (Nem szerepel az idődiagramban, értéke statikus.)

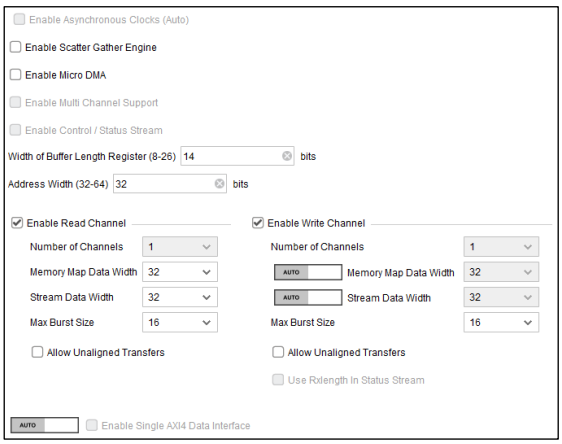
Tényleges adatátvitel akkor történik, amikor a forrás rendelkezik érvényes adattal (TVALID=1), a vevő pedig képes fogadni ezt az adatot (TREADY=1).

A Xilinx DMA vezérlője (meglehetősen buta módon) igényli a TLAST jel használatát is. TLAST jelet akkor kell generálni, amikor a DMA vezérlőben beprogramozott hosszúságú adatátvitel utolsó adata kerül továbbításra. Tehát ha például egy 128 bájtos átvitelt programozunk be és 32 bit széles adatfolyam interfészt használunk, akkor a 128/4=32-ik adat alatt kell 1 értékűnek lenni a TLAST jelnek.

Az AXI Stream interfész irányai a port nevéből kikövetkeztethetők: az MM2S betűszó a Memory-Map-to-Stream (CPU→Stream), míg az S2MM betűszó a Stream-to-Memory-Map (stream→CPU) kifejezéseket takarja.

A block design-t módosítsa az alábbiakban leírtaknak megfelelően:

- A DMA átvitelhez a **Zynq UltraScale+ MPSoC** processzoros alrendszerben szükség van egy új 32 bites AXI slave kapcsolatra (AXI HP0 FPD). A PL→PS irányú megszakításoknál engedélyezze az IRQ0 portot. A pl_clk0 órajel kimenetet kösse be az új AXI port órajel bemenetére.
- Adjon a rendszerhez egy **AXI SmartConnect** IP-t a DMA vezérlő AXI master interfészei számára. Állítsa be a slave és maser portok szükséges számát, majd kösse be a processzoros alrendszer AXI HP0 FPD interfészét, valamint az órajelet és az interconnect reset jelet.
- Adjon a rendszerhez egy **AXI Direct Memory Access** IP-t a DMA adatátvitel megvalósításához. Állítsa be az 5. ábrán lévő konfigurációs paramétereket, majd kösse be az AXI interfészeket, valamint az órajelet, a periféria reset jelet és a megszakításkérő kimeneteket. Az utóbbiakat egy **Concat** blokkon keresztül kell bekötni a processzoros alrendszer IRQ0 bemenetére.
- Adjon a rendszerhez egy **FIFO Generator** IP-t a DMA vezérlő két AXI Stream interfészének összekötéséhez. A létrehozott FIFO legyen 512 mélységű, mindkét portja legyen AXI Stream típusú, az adatszélesség legyen 4 bájtos, valamint biztosítsa a TLAST jel átvitelét is (így a FIFO szélessége 32 + 1, azaz 33 bit lesz).
- A korábbi logikai analizátort törölje és adjon a rendszerhez egy új **System ILA** blokkot a DMA átvitel vizsgálatához. Az egység 4 interfész slot-tal rendelkezzen, melyek közül a SLOT0 és SLOT1 típusa legyen AXI (ez az alapértelmezett), valamint a SLOT2 és a SLOT3 típusa legyen AXI Stream (xilinx.com:interface:axis rtl:1.0). Ezekbe kösse be a DMA vezérlő két nagysebességű AXI interfészét és a két AXI Stream interfészét. A mintatárat állítsa be 2048 mélységűre, a komparátorok számát 2-re, valamint engedélyezze a capture control és az advanced trigger lehetőségeket.



5. ábra: A DMA vezérlő konfigurációs paraméterei.

Miután elkészült a módosított block design, az Address Editor-ban rendeljen címtartományt az új perifériához. A DMA vezérlő AXI master interfészei a külső DDR4 memóriát érik el, így ezek címtartományából a QSPI interfész kizárható (exclude).

Az elkészítendő szoftver segítségével a DMA adatátvitelt fogjuk vizsgálni. A DMA vezérlő adatlapjában nézze át annak regiszterkészletét és gondolja át, hogy mit és milyen sorrendben kell felprogramozni. Ne használja a BSP-ben rendelkezésre álló meghajtót a DMA vezérlőhöz, hanem készítsen hozzá saját függvényeket, melyek lekérdezéses perifériakezelést valósítsanak meg (inicializálás, adatátvitel indítás, várakozás az adatátvitel végére). Az alkalmazásnak a következő DMA adatátviteli módokat kell támogatnia:

- Egyetlen 128 bájtos adat átvitele mindkét irányban
- 1024 bájt átvitele 128 bájtos blokkokban
- 1024 bájt egyszerre történő átvitele (azaz egy DMA vezérlő felprogramozással)

Ne feledkezzen el a DMA átvitelek esetén szükséges gyorsítótár műveletek végrehajtásáról sem! Az ide tartozó függvények a xil_cache.h fájlban találhatók meg.

Vizsgálja meg mindhárom esetet a logikai analizátorban.

4 Az audio interfész megvalósítása (3. hét)

A feladat megoldását az alábbi információk segítik:

- **Az FPGA alapú rendszerek tárgy gyakorlatai közül a kodek (I2S) interfész tervezése**
- **A CS4265 kodek adatlapjának 21. (I2S hullámformák) és 24-25. (órajel előállítás) oldala**

A megvalósítandó I2S interfész egyirányú, csak a kodek ADC részét használjuk, analóg jelet nem generálunk. A kodeket master módban használjuk, azaz az LRCLK és BCLK órajeleket az eszköz szolgáltatja. Az adatátviteli mód 24 bites ***left justified*** igazítással (a 32 bitből az LRCLK élt követő 24 bit az érvényes). Ennek megfelelően az interfész feladata, hogy a kodek irányából érkező LRCLK és BCLK jelek felhasználásával (mintavételezésével és a megfelelő él detektálásával) a sorosan érkező adatokat mintavételezze, párhuzamosítsa, és 32 adatbit érkezése után generáljon egy egyetlen rendszerórajel periódus hosszúságú ***adat érvényes (data valid)*** jelzést a két csatornára külön-külön. Amennyiben emlékei megfakultak, az I2S átvitel időzítési diagramja megtalálható a CODEC adatlapjában (lásd fent).

Az interfész feladata továbbá a kodek MCLK órajelének kiadása is. Az ehhez szükséges alap órajel egy **Clocking Wizard** IP-vel kerül majd előállításra, azonban órajelet kivezetni FPGA lábra közvetlenül nem lehet, hanem az alábbi két lehetőség közül választhatunk:

- Kimeneti DDR flip-flop-ot alkalmazunk (ODDRE1 primitív). Ekkor majd a Clocking Wizard kimenetén kell beállítani a megfelelő frekvenciát, amely változatlanul kerül az FPGA I/O lábra.
- Bináris számlálót alkalmazunk, amelynek egy kimeneti bitjét már kivezethetjük az FPGA I/O lábra és a Clocking Wizard által előállított órajel a választott bitnek megfelelő kettő hatvány értékkel leosztásra kerül. A számláló mérete (azaz a frekvenciaosztás) a C_CLKDIVCNT_WIDTH paraméterrel legyen beállítható (értéke módosítható az IP konfigurációs felületén).

A létrehozandó Verilog modul portjai tehát a következők lesznek:

- clk: a 100 MHz-es rendszerórajel
- mclk_in: a Clocking Wizard-ból érkező, a master clock előállításához használt órajel
- rst_n: aktív alacsony globális szinkron reset jel
- en: a kimeneti adat érvényes jel generálását engedélyező bemenet
- codec_mclk: a kodek felé menő master órajel
- codec_lrclk: a kodek felől érkező left/right órajel
- codec_bclk: a kodek felől érkező bit órajel
- codec_sdout: a kodek felől érkező soros adat
- codec_sdin: a kodek felé menő soros adat (esetünkben konstans 0)
- adc_data: 24 bites párhuzamos kimeneti adat
- adc_valid_l: az adc_data érvényes bal csatorna adatot tartalmaz és az en bemenet 1 értékű
- adc_valid_r: az adc_data érvényes jobb csatorna adatot tartalmaz és az en bemenet 1 értékű

A Logikai tervezés tárgy gyakorlatán tervezett interfészhez részben hasonlít a tervezendő modul. Ott master I2S interfészt készítettünk, most pedig egy slave módban működő egységre van szükség. Készítsen hozzá egy egyszerű tesztkörnyezetet is, amellyel az egység alapvető funkcionalitása ellenőrizhető. **A felhasználható funkcionális elemek kizárólag a következők lehetnek**:

- 3 bites shiftregiszterek és éldetektáló logika az LRCLK és BCLK jelekhez
- 2 bites shiftregiszter a bejövő soros adat szinkronizálásához
- 32 bites shiftregiszter a soros-párhuzamos átalakításhoz
- ODDRE1 vagy bináris számláló az MCLK kimenethez

5 Az audio kodek illesztése a rendszerbe (4. és 5. hét)

A feladat megoldását az alábbi információk segítik:

- **Az FPGA alapú rendszerek tárgy anyagából a perifériaillesztés**
- **AXI IIC Bus Interface Product Guide (PG090) 11-29. oldala (regiszterkészlet) és 40-42. oldala (I2C adatátvitel idődiagramok)**
- **A CS4265 kodek adatlapjának 24-25. (órajel előállítás) oldala, 31-32. oldala (I2C kommunikáció) és 34-45. oldala (regiszterkészlet)**

5.1 Az I2C konfigurációs interfész megvalósítása

A modulon található CS4265 típusú audio kodek a konfiguráció szempontjából nem egy bonyolult eszköz, a belső konfigurációs regisztereit I2C interfészen keresztül lehet elérni. **A kodek 7 bites I2C slave címe 0x4E.**

Mielőtt a hardver terv módosításához fogna, határozza meg, hogy milyen frekvenciájú MCLK órajelre van szükség. Ennek kitalálásához tanulmányozza át az eszköz adatlapjában az ide tartozó részt (az oldalszámokat lásd fentebb). A használni kívánt mintavételi frekvencia 96 kHz. A megfelelő órajelet az FPGA-ban állítjuk elő az előző pontban leírt lehetőségek valamelyikével.

A meghatározott beállítást beszélje meg egy mérésvezetővel!

A hardver terv módosítása az alábbi lépésekből áll:

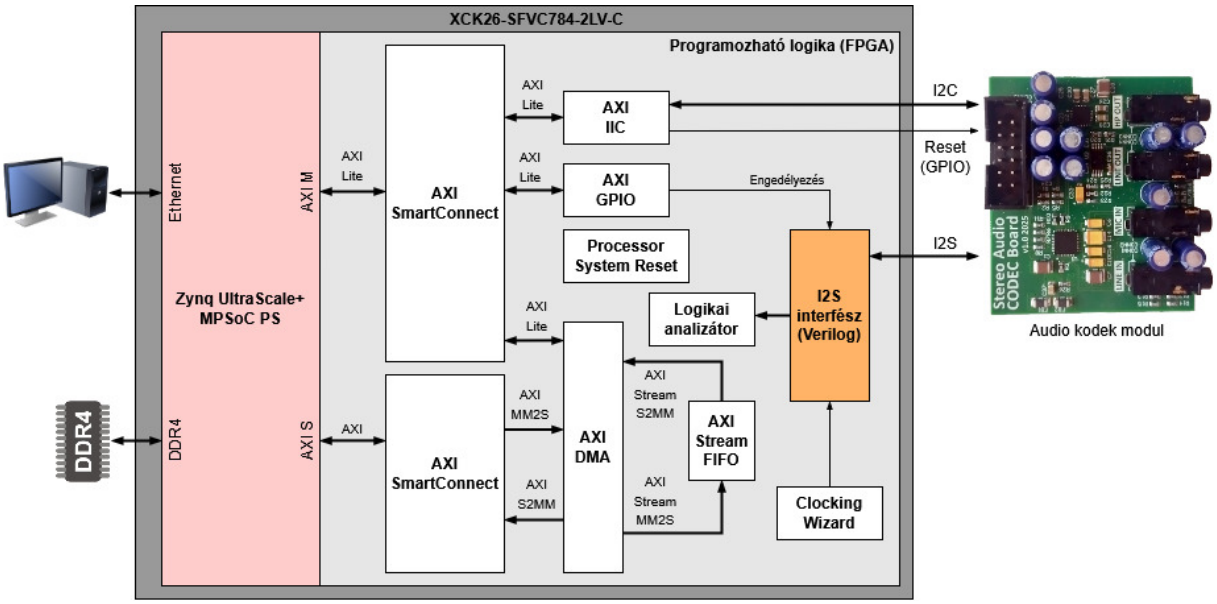
- Adjon a rendszerhez egy **AXI IIC** perifériát. Az SCL órajel frekvencia legyen 100 kHz, a GPO kimenet legyen 1 bites és az alapértelmezett értéke pedig legyen 0x01. A megfelelő helyre kösse be a periféria AXI Lite interfészét, az órajelét és a reset jelét. Az IIC interfészt vezesse ki külső portként i2c néven, a GPO kimenetet pedig codec_rstn néven (a kodek aktív alacsony reset jele). Az I2C interfészhez szükséges háromállapotú meghajtók a wrapper modulban kerülnek példányosításra.
- Törölje a **led** és a **btn** protokat. A meglévő GPIO periféria konfigurációját módosítsa úgy, hogy egyetlen 1 bites kimenettel rendelkezzen, amely az I2S engedélyező bemenetét vezérli.
- Adjon a rendszerhez egy **Clocking Wizard**-ot. A bemenő órajel forrása legyen Global Buffer, a kimenő órajel frekvenciáját határozza meg a választott MMCLK előállítási módnak megfelelően, valamint az opcionális jelek közül egy aktív alacsony reset bemenetre van még szükség. A bemenő órajelet és reset jelet kösse be a processzoros alrendszer megfelelő kimeneteire. (Megjegyzés: a 100 MHz-es bemeneti órajelből nem lehet pontosan előállítani a szükséges MCLK órajel frekvenciát. Az erre vonatkozó figyelmeztetést figyelmen kívül hagyhatjuk, az eltérés minimális).
- A projekt forrásfájlok közül húzza be a block design-ba a korábban elkészített I2S interfész Verilog modult, ennek hatására az egy RTL blokként meg fog jelenni. Kösse be a szükséges belső jeleket, a kodek modulhoz tartozó jeleket pedig vezesse ki.
- A hibakereséshez szeretnénk megnézni egy teljes I2C írási és olvasási hullámformát a logikai analizátorban, hogy össze tudjuk azokat vetni a kodek adatlapjában szereplő hullámformákkal. Ha minden rendszerórajel periódusban mintavételeznénk, akkor a logikai analizátorban igen nagy mélységű mintatárra lenne szükség, mert a rendszerórajel frekvenciája (100 MHz) sokkal nagyobb az SCL frekvenciájánál (100 kHz). Ahhoz, hogy az I2C hullámformákat megfelelően lássuk, elegendő lenne SCK periódusonként 8-16 mintavétel. Ehhez szükség van egy olyan logikai analizátor bemenetre, amellyel a mintavétel engedélyezésének feltétele beállítható.

Erre a 100 MHz-es rendszerórajelről működő **Binary Counter** IP-t használunk. Hány bites legyen a számláló a kívánt mintavételi gyakoriság eléréséhez?

- Az eddigi System ILA blokkokat törölje a rendszerből. Az I2C és az I2S jelek figyeléséhez két új System ILA-t fogunk használni. Mindkét esetben ügyeljen rá, hogy a capture control és az advanced trigger legyen bekapcsolava. Az órajel bemenetükre a rendszerórajelet kössük be.
- Az I2C interfész vizsgálatára szolgáló **System ILA** konfigurációja a következő legyen:
  - Monitor type: mixed
  - 1 IIC slot (xilinx.com:interface:iic rtl:1.0), melybe az I2C interfész kerül bekötésre
  - 1 probe, melybe a Binary Counter kimenete kerül bekötésre
  - A mintatár mélységét úgy válassza meg, hogy beleférjen egy teljes írási és egy teljes olvasási I2C adatátviteli ciklus
- Az I2S interfész vizsgálatára szolgáló **System ILA** konfigurációja a következő legyen:
  - Monitor type: native
  - 7 probe
  - Az I2S modul adat kimenete, a két valid kimenete és az engedélyező jele az első 4 probe-ba legyenen bekötve
  - A kodek felől érkező LRCLK, BCLK és soros adat jelek az utolsó 3 probe-ba legyenek bekötve

Miután elkészült a hardver módosítása, generálja újra a HDL wrapper-t a block design-hoz és az ebben található port nevek felhasználásával módosítsa a projektben lévő XDC fájlt. Ebben segítségre van a bevetető részben található 1. táblázat is. (Megjegyzés: Ezt alkalmazva elkerülhető a szintézis többszöri futtatása. Másik lehetőség, hogy a szintézis előtt eltávolítjuk az XDC fájlt, majd a szintetizált rendszer megnyitása után az I/O Ports ablakban újból megadjuk a lábkiosztást.)

Exportálja a hardvert a Vitis fejlesztői környezet számára a konfigurációs bitfolyam nélkül, majd indítsa el az implementációt. Eközben folytatható a munka a következő feladattal. Az ebben a fázisban elkészült rendszer blokkvázlata a 6. ábrán látható.



6. ábra: Az I2C és az I2S interfészek tesztelésére szolgáló rendszer blokkvázlata.

5.2 Az audio kodek konfigurálása és tesztelése

Az I2S interfész megfelelő működésének ellenőrzéséhez konfigurálni kell a kodeket az I2C interfészen keresztül. Ehhez nem fogjuk használni a BSP-ben lévő meghajtót, hanem regiszterszintű hozzáféréssel mi magunk kezeljük az I2C perifériát. A ***Dynamic Controller Logic Flow*** üzemmódot használjuk, ennek működését a periféria dokumentációjának 37. oldala tartalmazza. A regiszterek leírása a 11. oldalon kezdődik.

Három függvény megírására lesz szükség, melyeknek neve i2c_init, i2c_write és i2c_read. Ezeknek a dokumentáció alapján az alábbiakat kell tartalmaznia.

i2c_init():

- Állítsa alapállapotba a perifériát a Soft Reset Register segítségével
- Állítsa az I2C interfész GPO kimenetét 1-be (a kodek reset jele)
- Állítsa az RX FIFO méretét a maximumra (RX_FIFO_PIRQ = 0x0F)
- Állítsa alapállapotba (reset) a TX FIFO-t
- Engedélyezze a periféria működését és kapcsolja ki a TX FIFO reset jelét

i2c_write(uint8_t slv_addr, uint8_t reg_addr, uint8_t data):

- Várakozzon, amíg valamelyik FIFO nem üres vagy az I2C busz foglalt (státusz regiszter olvasás)
- TX FIFO írása (start feltétel küldés, slave cím, I2C írás)
- TX FIFO írása (regiszter cím)
- TX FIFO írása (stop feltétel küldés, adat bájt)
- Várakozzon, amíg a TX FIFO nem üres vagy az I2C busz foglalt (státusz regiszter olvasás)

uint8_t i2c_read(uint8_t slv_addr, uint8_t reg_addr):

- Várakozzon, amíg valamelyik FIFO nem üres vagy az I2C busz foglalt (státusz regiszter olvasás)
- TX FIFO írása (start feltétel küldés, slave cím, I2C írás)
- TX FIFO írása (regiszter cím)
- TX FIFO írása (ismételt start feltétel küldés, slave cím, I2C olvasás)
- TX FIFO írása (stop feltétel küldés, 1 bájt olvasása)
- Várakozzon, amíg a TX FIFO nem üres vagy az RX FIFO üres vagy az I2C busz foglalt (státusz regiszter olvasás)
- Térjen vissza az RX FIFO-ból kiolvasott adattal

Készítsen el egy ***codec_init()*** függvényt, amely először kiad a kodek számára egy 1 ms hosszú reset pulzust és utána ugyanennyi ideig még várakozik, majd a 2. táblázatban megadott sorrendben elvégzi a kodek regisztereibe a megfelelő értékek beírását. A ***main()*** függvényből hívja meg a kodek inicializálását elvégző függvényt.

Az I2C interfész nem megfelelő működése esetén a szoftver debug módban történő végrehajtása és a logikai analizátor segítségével próbálja meg kideríteni az esetleges hiba okát!

Ellenőrizze a logikai analizátorral, hogy az I2S interfész megfelelően működik! Azaz kösse össze a kodek modul analóg vonali bemenetét a PC audio kimenetével, generáljon a PC-n szinusz jelet (pl. az Audacity program használatával) és ellenőrizze a logikai analizátorral a két csatorna mintavételezett jelének hullámformáját. Ehhez állítsa az adat típust előjelesre, a megjelenítést pedig analógra. A logikai analizátort úgy állítsa be, hogy mintavétel csak akkor történjen, ha az I2S interfész adc_valid_l kimenete 1 értékű (vagy a jobb csatorna vizsgálata esetén az adc_valid_r kimenet).

2. táblázat: A kodek regisztereinek tartalma.

Regiszter cím	A regiszter funkciója	Megjegyzés	Érték
0x03	DAC control 1 register		0x00
0x04	ADC control register	Üzemmód, adatformátum, működési sebesség.	?
0x05	MCLK frequency register	MCLK bemeneti osztás.	?
0x06	Signal selection register	Az SDIN1 soros adat bemenetet használjuk.	?
0x07	PGA B gain register	Állítsuk az erősítést -6.0 dB-re.	?
0x08	PGA A gain register	Állítsuk az erősítést -6.0 dB-re.	?
0x09	ADC input control register	Az ADC-re a vonali bemenet csatlakozzon.	?
0x0a	DAC A volume control register		0x00
0x0b	DAC B volume control register		0x00
0x0c	DAC control 2 register		0x00
0x02	Power control	A nem használt funkciók kikapcsolhatók.	?

6 I2S→AXI Stream interfész megvalósítása Vitis HLS-ben (6. hét)

A feladat megoldását az alábbi információk segítik:

- **A Heterogén SoC rendszerek tárgy HLS előadás anyagából a „Block and Port Level Protocols" és az „Arbitrary Precision: C++ ap_fixed types" részek, valamint a FIR szűrő tervezés HLS gyakorlat**

A FIR szűrő megvalósítása előtt egy egyszerű ap_hs → AXI Stream interfész konverziót valósítunk meg Vitis HLS-ben. Ez praktikusan azt jelenti, hogy a Xilinx DMA vezérlő által igényelt TLAST jel előállításáról kell gondoskodni.

A feladat megoldását a HLS kód megírásával kezdjük. A létrehozandó blokknak az alábbi portjai lesznek:

1. tlast_dnum: 16 bites unsigned, az interfész típusa AXI Lite. Azt mutatja meg, hogy hány kimeneti mintánként kell TLAST jelzést generálni.
2. input_l: 24 bites (előjeles, 23 bit törtrész), az interfész típusa ap_hs. A bal csatorna adata.
3. input_r 24 bites (előjeles, 23 bit törtrész), az interfész típusa ap_hs. A jobb csatorna adata.
4. res: kimeneti AXI Stream, az adat: 32 bites előjeles, 31 bitnyi törtrésszel

Mivel mindkét bementünk rendelkezik port szintű handshake jelekkel, így a „Block level handshake" jeleket kapcsolja ki.

A megvalósítandó **C++** kódnak a következő funkciót kell megvalósítania: a két bemeneti adatot felváltva a kimeneti portra írja (azaz a kimeneten a két csatorna adatai felváltva jelennek meg). A tlast jel legyen 1, amennyiben az éppen kiírt minta tlast_dnum-adik, egyébként 0. Ügyeljen rá, hogy a kimeneti struktúrát egyetlen értékadással írja (tehát NEM elemenként). A HLS-ben szintetizálandó C függvény tehát:

```
void fir_hw(    ap_uint<16> tlast_dnum,
                din_t *input_l, din_t *input_r,
                stream_type &res)
```

A Vitis HLS sztenderd C kódból csak olyan AXI Stream interflszt generál, amely csak a handshake és adat jeleket tartalmazza, ahhoz, hogy a TLAST jel is megjelenjen, a HLS megfelelő template-jét kell használni:

```
typedef hls::axis<dout_t, 0, 0, 0> axis_type;
typedef hls::stream<axis_type> stream_type;
```

A szükséges definíciókat az ap_axi_sdata.h és hls_stream.h header fájlok tartalmazzák. A definiált axis_type típus esetén a busz jeleire történő hivatkozásra példa:

```
axis_type out_data;
out_data.data = *input_l;
out_data.last = 0;
out_data.keep = -1;
```

Kimeneti stream írása:

```
res.write(out_data);
```

A kód megírása után generáljon HDL kimenetet, majd IP_XACT formátumban exportálja a létrejövő modult. Vivado-ban nyissa meg az „IP Catalog"-t és adja hozzá a HLS kimeneti könyvtárát az IP repository-hoz (solution/impl/ip).

A Block design módosítása az alábbi lépésekből áll:

1. IP-k hozzáadása a tervhez
    - Adja hozzá a **HLS**-ből generált IP-t a tervhez.
    - A „Block design”-ban hozzon létre két darab **FIFO**-t a két audió csatorna adatainak tárolásához. Mindkét FIFO 512x24 bites FWFT (First Word Fall Through) FIFO legyen. A sztenderd portok mellett generáljon Valid portot is.
    - A HLS blokk kimenete egy **FIFO**-ba kerül, ez a DMA teszt óta része a tervnek.
2. IP-k összekötése
    - A két FWFT FIFO Write portját kösse össze az I2S modul kimeneti portjaival.
    - Az FWFT FIFO-k kimeneti portjait kösse össze a HLS blokk bemeneti portjaival (adat, valid, ack).
    - Az AXI Stream FIFO bemenetét csatlakoztassa a HLS blokk kimenetéhez, kimeneti portját pedig kösse be a DMA vezérlőbe. A DMA vezérlő MM2S kimenetére nincs szükségünk.
    - Kösse be a megfelelő órajel és reset jeleket.
3. Debug
    - Távolítsa el az eddig használt ILA-t, majd adjon hozzá egy újat, továbbra is Mixed típussal.
    - Az interfész Slot-hoz adja hozzá a HLS modul kimeneti AXI Stream portját, míg a megfelelő számú Probe-hoz a HLS modul bemeneti jeleit (adat, valid, ack).
4. Implementálja a tervet.
5. Exportálja a tervet a Vitis-be. A HLS-ből exportált hardver modulhoz tartozó definíciók (pl. regiszter offszetek) a BSP include könyvtárában található fir_hw.h és fir_hw_hw.h fájlokban találhatók!
6. A korábban megírt DMA teszt szoftvert módosítsa úgy, hogy folyamatosan fogadja az I2S interfész felől érkező adatokat. Ellenőrizze ChipScope-ban, illetve a DMA-zás cél címén, hogy nagyjából megfelelő adatokat kerülnek a memóriába.

A rendszer blokkvázlata ekkor megegyezik az 1. ábrán levővel, csak a funkcionalitása hiányos.

7 FIR szűrő megvalósítása Vitis HLS-ben (7. és 8. hét)

A feladat megoldását az alábbi információk segítik:

- **A Heterogén SoC rendszerek tárgyból a FIR szűrő tervezés HLS gyakorlat**

A tényleges jelfeldolgozási feladatokat egy, az FPGA-ban implementált FIR szűrő valósítja meg. A szűrő specifikációja a következő:

- 2 csatorna
- 96 kHz bemeneti mintavételi frekvencia
- Programozható 1x, 2x és 4x decimálás
- Programozható számú együttható (legfeljebb 512 darab, 1x decimálásnál 128, 2x-esnél 256, 4x-esnél pedig 512 együtthatót használunk)
- Programozható együttható készlet
- A szűréshez szükséges mintákat cirkuláris pufferben tárolja
- 24 bites 1.23 formátumú bemeneti adatok
- 32 bites 1.31 formátumú együtthatók
- 32 bites 1.31 formátumú kimenet

A HLS-ben szintetizálandó **C++** függvény prototípusa a következő:

void fir_hw(ap_uint<16> tlast_dnum, ap_uint<3> smpl_rd_num, ap_uint<9> tap_num_m1,
*coeff_t coeff_hw[512], din_t *input_l, din_t *input_r, stream_type &res);*

Az előző részfeladathoz képest új jelek funkciója:

- smpl_rd_num: egy kimeneti minta előállításához beolvasandó adatok száma, azaz a decimálási faktor (az interfész típusa AXI Lite)
- tap_num_m1: használt együtthatók száma – 1 (az interfész típusa AXI Lite)
- coeff_hw[512]: együttható készletet tároló tömb, az elemek típusa legyen 32 bites, előjeles és 31 bit törtrészt tartalmazó fixpontos érték (az interfész típusa AXI Lite)

A „Block level handshake” jeleket kapcsolja ki, az input_l és input_r portokra pedig ap_hs (azaz kétirányú) handshake jeleket állítson be!

Az elvégzendő feladatok az alábbiak:

- Készítse el a szűrő C++ nyelvű leírását, majd egy C++ tesztkörnyezet segítségével ellenőrizze a működést (dirac jellegű gerjesztésre a kimeneten megjelenik az együttható készlet). A 3 decimálási módhoz 3 darab Matlab által generált együttható készlet a tárgy honlapjáról letölthető. A mintatárat cirkuláris bufferben valósítsa meg!
- Adja meg a megfelelő direktívákat, amivel minimális erőforrás igényű, de valós idejű működésre képes hardver generálható. Mivel a ciklusok iteráció száma változó, használja a LOOP_TRIPCOUNT direktívát is. A szükséges párhuzamosítás mértékét gondolja át (decimálásról van szó!), és egyeztesse laborvezetővel!
- Generáljon Verilog kimenetet, majd IP_XACT IP-t, frissítse Vivado-ban az IP repository-t, majd fordítsa újra a tervet. Exportálja a hardvert a Vitis-be, majd hozzon létre új szoftver projektet új BSP-vel.
- Ellenőrizze a szűrő működését a kimeneti adatok logikai analizátorral történő vizsgálatával!

8 A hálózati kommunikáció megvalósítása (9. és 10. hét)

A fejlesztői kártya és a PC közötti hálózati kommunikációhoz a processzoros rendszerben lévő Ethernet interfészt és az lwIP TCP/IP stack-et használjuk.

- A Board Support Package beállításoknál a rendelkezésre álló szoftverkönyvtárak listájában válasszuk ki az **lwip220**-at. Az lwIP alapértelmezett beállításai megfelelőek, ezeken módosítani nem kell.
- Hozzunk létre egy új példa szoftver projektet (File→New Example) és a sablonok közül válasszuk ki az **lwIP Echo Server**-t.

Az lwIP felhasználja a xiltimer könyvtárat, amelynek alapértelmezett konfigurációját ellenőrizzük a BSP beállításoknál. Amennyiben szükséges, akkor módosítsuk ezt az alábbiaknak megfelelően:

- XILTIMER_en_interval_timer: true
- XILTIMER_sleep_timer: Default
- XILTIMER_tick_timer: psu_ttc_1

Az lwIP Echo Server alkalmazás már tartalmazza az Ethernet interfész kezeléséhez szükséges meghajtót és a TCP/IP stack inicializálását, hozzáadni csak az egyedi UDP kommunikációt megvalósító kódrészletet kell. **Fontos, hogy a program letöltése és futtatása előtt a kártya legyen csatlakoztatva a hálózatra, ellenkező esetben a program nem fog megfelelően működni!**

- A ***main.c*** fájlban keressük meg a MAC cím és a statikus IP cím beállításokat és módosítsuk ezeket egyedi értékekre. Minden hálózatra kapcsolt eszköz egyedi MAC és IP címekkel kell, hogy rendelkezzen. A MAC cím legyen 02:00:00:00:00:xx (ahol xx egy 00 és FF közötti egyedi érték). Az IP cím legyen 192.168.1.x (ahol x egy 10 és 99 közötti egyedi érték), az alhálózati maszk 255.255.255.0 és az alapértelmezett átjáró címe pedig 192.168.1.1.
- Az ***echo.c*** fájl tartalmazza az alkalmazásspecifikus részt, ezt nevezzük át ***application.c***-re.
- Adjuk hozzá a projekthez a korábban elkészített CODEC inicializáló kódot.

Az ***application.c*** forrásfájl tartalmát módosítsuk az alábbiak szerint:

- A **start_application()** függvényben kell elvégezni az UDP kommunikáció, az audio CODEC, a FIR szűrő, valamint a DMA vezérlő inicializálását. Az audio CODEC, a FIR szűrő és a DMA inicializálása az UDP kommunikáció kezdeti teszteléséhez még nem szükséges.
    - Az **udp_new()** függvény meghívásával hozzunk létre egy új UDP PCB struktúrát.
    - Az **udp_bind()** függvény meghívásával rendeljük hozzá a létrehozott UDP PCB-t minden helyi hálózati interfészhez (IP_ADDR_ANY) és egy porthoz (pl. 1234).
    - Hozzunk létre egy üres **void recv_callback(void *arg, struct udp_pcb *pcb, struct pbuf *p, ip_addr_t *addr, u16_t port)** függvényt. Az **udp_recv()** függvény meghívásával állítsuk be, hogy a **recv_callback()** hívódjon meg az UDP csomagok vételekor.
    - Végezzük el az audio CODEC inicializálását (a kezdeti teszteléshez még nem szükséges).
    - Végezzük el a FIR szűrő inicializálását (a kezdeti teszteléshez még nem szükséges).
    - Konfiguráljuk és indítsuk el a DMA átvitelt (a kezdeti teszteléshez még nem szükséges).
    - Indítsuk el a FIR szűrőt (a kezdeti teszteléshez még nem szükséges).
- A **recv_callback()** függvényben kell megvalósítani a vett UDP csomagok feldolgozását. A PC-n futó alkalmazás 1 bájtot tartalmazó UDP csomagokat küld, amikor az audio adatok küldését el kell indítani, valamint le kell állítani. A vett adatokat a kapott **pbuf** struktúra **payload** mezőjén keresztül érhetjük el. A feldolgozás végén **pbuf** struktúrát fel kell szabadítani a **pbuf_free()** függvény meghívásával. A vett UDP csomag a következő adatokat tartalmazhatja:

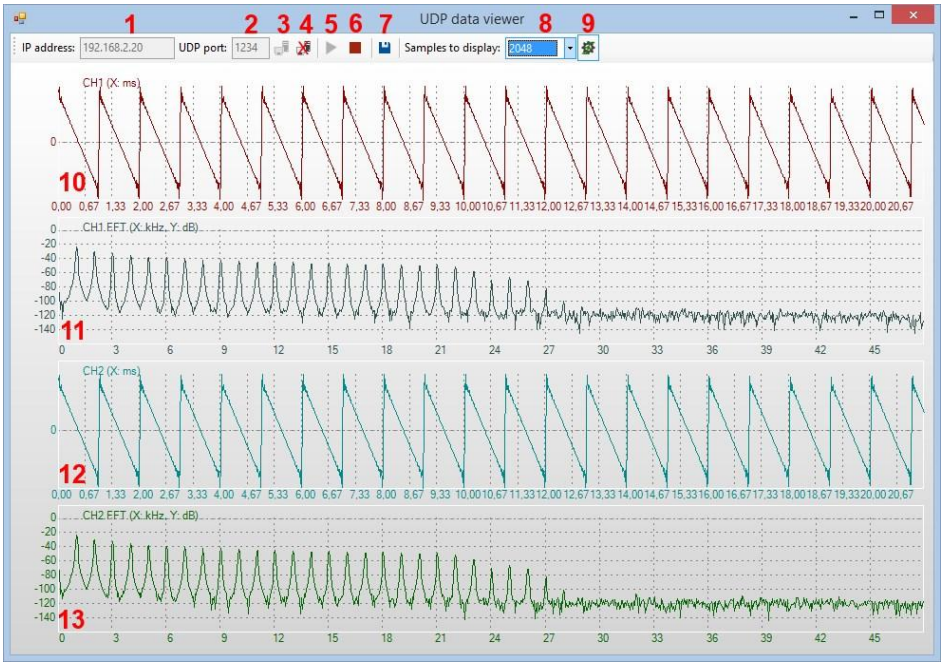
    - **0x00**: Adatküldés indítása. Az adatküldéshez el kell menteni a kapott mutatót az UDP PCB struktúrára, a forrás IP címet és UDP portot. Ezután egy flag beállításával engedélyezhető az audio adatok átvitele.
    - **0x01**: Adatküldés leállítása. Törölni kell az előbbi adatátvitelt engedélyező jelzést.
- A **transfer_data()** függvényben kell megvalósítani az audio adatok elküldését UDP-n keresztül. A PC-n futó alkalmazás 1024 bájtos csomagokat vár, amelyben a bal és a jobb csatornához tartozó 32 bites szavak felváltva követik egymást és a kisebb címen a szavak LSB-je található. A FIR szűrőnek alapból ilyen formátumú a kimenete. A küldendő adatok tárolásához hozzunk létre egy megfelelő méretű tömböt globális változóként. Ha az adatküldést engedélyező jelzés be van állítva és van rendelkezésre álló adat, akkor azt az **udp_sendto()** fügvénnyel küldhetjük el a korábban eltárolt IP címre és UDP portra. Az adatküldéshez szükség van még egy új **pbuf** struktúrára, amely a **pbuf_alloc()** függvény meghívásával hozható létre, a paramétereknek adjuk meg a PBUF_TRANSPORT, 1024 (a csomag mérete) és PBUF_REF értékeket. A létrehozott **pbuf** struktúra **payload** mezőjét állítsuk a küldendő adatok kezdőcímére. Az adatátvitel után a létrehozott **pbuf** struktúra a **pbuf_free()** függvénnyel szabadítható fel.

Az lwIP megfelelő működése és az UDP adatátvitel a FIR szűrő és a DMA átvitel nélkül is ellenőrizhető a ping parancs kiadásával, illetve egy előre legenerált egyszerű hullámforma (pl. fűrészfog, négyszög) elküldésével. Az adatküldést a TCP fast timer esemény ütemezze (ennek kezelése a **main.c** fájlban található meg), amely másodpercenként nyolcszor következik be. Ennek hatására állítsuk be az adat rendelkezésre áll jelzést. Az alábbi táblázat összefoglalja a felhasznált lwIP függvényeket.

struct udp_pcb *udp_new(void) Létrehozza az UDP kommunikációhoz szükséges PCB struktúrát. Paraméterek: nincs Visszatérési érték: Mutató az UDP PCB struktúrára, illetve hiba esetén 0.
err_t udp_bind(struct udp_pcb *pcb, ip_addr_t *ipaddr, u16_t port) Hozzáköti az UDP PCB-t a megadott helyi hálózati interfészhez (IP címhez) és UDP porthoz. Paraméterek: pcb Mutató az UDP PCB struktúrára. ipaddr Mutató a helyi hálózat IP címére, illetve IP_ADDR_ANY. port Helyi UDP port. Visszatérési érték: ERR_OK A hozzákötés sikeres volt. ERR_USE A megadott helyi IP cím és UDP port már használatban van.
void udp_recv(struct udp_pcb *pcb, udp_recv_fn recv, void *recv_arg) Beállítja a vett UDP csomagok feldolgozását végző függvényt. Paraméterek: pcb Mutató az UDP PCB struktúrára. recv Mutató a vett UDP csomagok feldolgozását végző függvényre. recv_arg Egyedi paraméter a feldolgozást végző függvény számára. Visszatérési érték: nincs
void udp_recv_fn(void *arg, struct udp_pcb *pcb, struct pbuf *p, ip_addr_t *addr, u16_t port) A vett UDP csomagok feldolgozását végző függvény prototípusa. Paraméterek: arg Az **udp_recv()** függvénynek megadott egyedi paraméter. pcb Mutató az UDP PCB struktúrára. p Mutató az adatpuffert leíró **pbuf** struktúrára.

addr	Mutató a forrás IP címét tároló struktúrára.
port	Forrás UDP port.
Visszatérési érték:	
nincs	
struct pbuf *pbuf_alloc(pbuf_layer layer, u16_t length, pbuf_type type)	
Új pbuf struktúra létrehozása.	
Paraméterek:	
layer	A hálózati kommunikáció típusa. A csomag fejlécének méretét határozza meg.
length	Az adatméret bájtokban.
type	A pbuf struktúra létrehozásának módja.
Visszatérési érték:	
Mutató a pbuf struktúrára, illetve hiba esetén 0.	
u8_t pbuf_free(struct pbuf *p)	
Korábban létrehozott pbuf struktúra felszabadítása.	
Paraméterek:	
p	Mutató a felszabadítandó pbuf struktúrára.
Visszatérési érték:	
A felszabadított pbuf struktúrák száma.	
err_t udp_sendto(struct udp_pcb *pcb, struct pbuf *p, ip_addr_t *dst_ip, u16_t dst_port)	
UDP csomag küldése a megadott távoli IP címre és portra.	
Paraméterek:	
pcb	Mutató az UDP PCB struktúrára.
p	Mutató az adatpuffert leíró **pbuf** struktúrára.
dst_ip	Mutató a távoli IP címet tároló struktúrára.
dst_port	A távoli UDP port.
Visszatérési érték:	
ERR_OK siker esetén, más visszatérési érték hibát jelez.	

A PC-n futó alkalmazás felhasználói felülete a 7. ábrán látható. **Ez Windows operációs rendszerhez készült, ezért a laborban lévő gépeken a VMware-ben megtalálható „Windows 10 ISE” virtuális gépen lehet futtatni.**



7. ábra: Az UDP csomagok tartalmát megjelenítő alkalmazás.

1. A távoli eszköz (FPGA kártya) IP címe.
2. A távoli eszközhöz (FPGA kártya) tartozó UDP port.
3. Kapcsolódás a távoli eszközhöz.
4. A kapcsolat megszűntetése.
5. Az adatátvitel indítása.
6. Az adatátvitel leállítása.
7. A vett adatok elmentése. A gombra kattintás után megadható a fájl neve, amelybe a vett nyers adatok elmentésre kerülnek. A gombra történő ismételt kattintás leállítja az adatok mentését.
8. A megjelenített minták számának beállítása.
9. Trigger beállítások (engedélyezés vagy tiltás, felfutó vagy lefutó él kiválasztása, trigger szint).
10. A bal csatornán vett minták.
11. A bal csatorna spektruma.
12. A jobb csatornán vett minták.
13. A jobb csatorna spektruma.

Miután a PC-s alkalmazás sikeresen megjelenítette az előre legenerált hullámformát, hozzáadhatjuk a programhoz a DMA átvitel kezelését. A DMA adatátvitel megvalósítása dupla pufferelésés módon történjen, így a hálózati adatátvitel ideje alatt a puffer másik felébe lehet írni adatokat a FIR szűrő kimenetéről. Hozzunk létre egy megszakításkezelést használó példa alkalmazást az AXI DMA vezérlőhöz (Platform Component beállításai→Board Support Package→drivers→axi_dma→Import Examples), ahonnan a forráskód nagy része átvehető. A példa alkalmazás mindkét irányú DMA átvitelt megvalósítja, nekünk csak a vételi irány szükséges (AXI stream → memória, DEVICE_TO_DMA). **A Vitis fejlesztői környezet a System Device Tree-t (SDT) használja a hardver platform leírásához, tehát a DMA példa alkalmazás kódjának értelmezésénél feltételezhetjük, hogy az SDT makró definiálva van.**

Hozzunk létre egy **int dma_it_init(unsigned char *buff, unsigned long buff_size, unsigned long transfer_size)** függvényt, amely elvégzi a DMA adatátvitel inicializálását. A példa alkalmazásban a **main()** függvény tartalmazza ennek megvalósítását.

- A **buff** paraméterben kapjuk meg a DMA buffer kezdőcímét, a **buff_size** paraméterben kapjuk meg a DMA puffer teljes méretét, a **transfer_size** paraméter pedig a DMA adatátvitel méretét adja meg, ami 1024 bájtos kell, hogy legyen.
- A fenti adatokat tároljuk el globális változókban, mert a megszakításkezelő rutinban szükség lesz ezekre. Hozzunk létre globális változókat, amelyek megadják, hogy az AXI DMA vezérlő és az Ethernet interfész a DMA puffer mely részéhez férhet hozzá (DMA puffer mutatók).
- Inicializáljuk a DMA vezérlőt: **XAxiDma_LookupConfig()**, **XAxiDma_CfgInitialize()**.
- A megszakításkezelést az lwIP példa alkalmazás inicializálja, ezért a következő lépésben csak a DMA vételi megszakításkezelő rutint kell regisztrálni az **XSetupInterruptSystem()** függvénnyel (ennek prototípusa a xinterrupt_wrap.h header fájlban található meg). A block design-ban nézzük meg, hogy a processzoros alrendszer IRQ0 portjának mely sorszámú bitjére lett bekötve az AXI DMA periféria S2MM irányú megszakításkérő kimenete és a megszakításkezelő rutin regisztrálásánál az ennek megfelelő indexű IntrId tömb elemet használjuk az XAxiDma_Config struktúrából. (Ha szükséges, akkor az IntrId tömb elemek értéke a debugger-ben megnézhető, melyek alsó 12 bitje a megszakítás azonosító, a felső 4 bitje pedig a megszakítás típusát adja meg. A processzoros alrendszer IRQ0 bemenetének 0. bitjéhez a 90-es megszakítás azonosító érték tartozik.)
- Tiltsuk és engedélyezzük a vételi megszakítást a DMA vezérlőben: **XAxiDma_IntrDisable()**, **XAxiDma_IntrEnable()**.
- Inicializáljuk az adat rendelkezésre áll jelzést és a DMA puffer mutatókat.

- Indítsuk el az első DMA átvitelt: **XAxiDma_SimpleTransfer()**.

Hozzunk létre egy **void RxIntrHandler(void *Callback)** függvényt, amely a vételi megszakításkezelő rutint valósítja meg. A példa alkalmazásban ez ugyanilyen néven található meg.

- Olvassuk be és nyugtázzuk az aktív megszakítást: **XAxiDma_IntrGetIrq()**, **AxiDma_IntrAckIrq()**.
- Indítsuk el a következő DMA átvitelt.
- Aktualizáljuk a DMA puffer mutatókat és állítsuk be az adat rendelkezésre áll jelzést.

A **transfer_data()** függvényt módosítsuk úgy, hogy a DMA puffer megfelelő része legyen elküldve, amennyiben rendelkezésre áll új adat. Az adatküldés előtt szükséges a DMA puffer címtartományának érvénytelenítése a processzor adat cache-ében a **Xil_DCacheInvalidateRange()** függvénnyel.

9 Hasznos információk

A laboron használt Kria KV260 Vision AI Starter Kit kejlesztői kártya esetén az azon lévő SoC eszköz boot módja fixen QSPI módba van állítva hardveresen és a QSPI interfészre csatlakozó flash memória gyárilag már fel van programozva. Ez azt jelenti, hogy a bekapcsolás után rögtön betöltődik egy konfiguráció és a fejlesztői környezetből történő program letöltés emiatt nem lesz sikeres. A probléma megoldását a kártya minden bekapcsolása után a JTAG boot mód szoftveres kiválasztása jelenti, amelyet a kria_boot.tcl szkripttel lehet megtenni. Ennek használata Linux operációs rendszer alatt a következő:

- Nyissunk meg két parancssort és menjünk el abba a könyvtárba, ahová a TCL szkriptet mentettük.
- Mindkét parancssorban állítsuk be a Vivado fejlesztői környezethez szükséges környezeti változókat a ***source /xilinx/Vivado/2025.2/settings64.sh*** parancs kiadásával.
- Az egyik parancssorban indítsuk el a hardver szervert a ***hw_server*** parancs kiadásával (ez kezeli a fejlesztői kártyával való kommunikációt). Ebből a CTRL+C billenyűkombinációval lehet kilépni, amennyiben már nincs rá szükség.
- A másik parancssorban indítsuk el a Xilinx Software Command Line Tool-t (XSCT) az ***xsct*** parancs kiadásával. Itt a JTAG boot mód beállításához adjuk ki az alábbi parancsokat:
  - ***source kria_boot.tcl*** (a szkript betöltése, csak egyszer szükséges)
  - ***connect*** (csatlakozás a hardver szerverhez, csak egyszer szükséges)
  - ***boot_jtag*** (a JTAG boot mód beállítása)
- Az XSCT-ből az ***exit*** parancs kiadásával tudunk kilépni.

A Vitis fejlesztői környezetben a szoftver alkalmazások letöltésekor alapesetben a teljes rendszer alapállapotba állításra kerül, amely során a programozható logika törlése is megtörténik. Ez a logikai analizátor használatakor esetünkben problémát okoz. A szoftver alkalmazások letöltési beállításainál ezt ki tudjuk kapcsolni, amelynek megjelenítéséhez kattintsunk duplán az alkalmazáshoz tartozó ***Settings*** csomópontban található ***launch.json*** elemre:

- A ***Reset Entire System*** opció kikapcsolása esetén a programozható logika nem fog törlődni
- Amennyiben az exportált hardver platform tartalmaz konfigurációs bitfolyamot, akkor a ***Program Device*** opció kikapcsolása esetén a programozható logika nem kerül az alkalmazás letöltése előtt konfigurálásra

Ahhoz, hogy a Vivado megtalálja a rendszerben lévő logikai analizátort a használt SoC eszköz esetén, először le kell futnia egy inicializáló kódnak, amely konfigurálja a processzoros alrendszert. Ezt a Vitis a szoftver futtatása előtt indítja el. Tehát az első bekapcsolás után:

- A Vivado Hardware Manager-ben felkonfiguráljuk a programozható logikát
- A Vitis-ben futtatjuk a szoftvert
- A Vivado Hardware Manager-ben frissítjük az eszközt (Refresh device)