

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

**Villamosmérnöki és Informatikai Kar**

**Programozható logikai eszközök alkalmazástechnikája (MIT)**

**FPGA Tervezői Laboratórium**

**Készítették: Cseh Péter (DM5HMB)**

**Hajnal Bence (EWHV0Q)**

**2018.02.11.**

Tartalom

[1. Mérés: Egyszerű processzoros rendszer létrehozása 3](#_Toc506839643)

[1.1. Hardver 3](#_Toc506839644)

[1.2. Szoftver 4](#_Toc506839645)

[1.3. Tesztelés 5](#_Toc506839646)

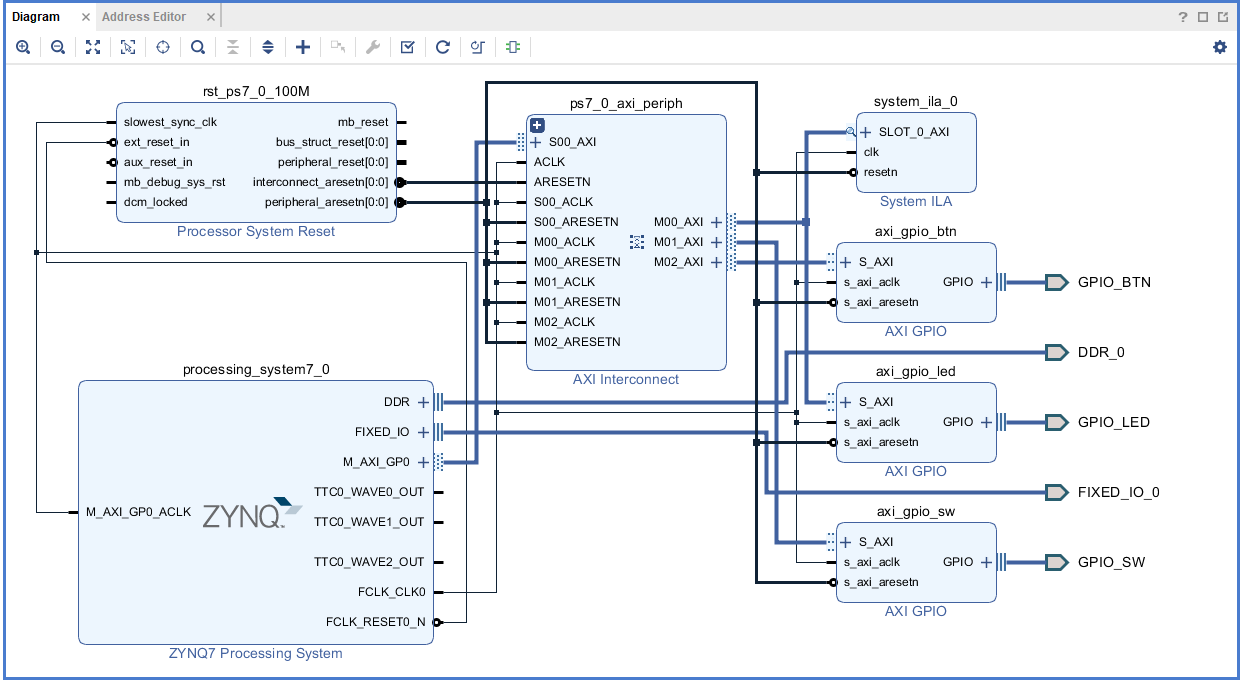
[2. Mérés: DMA vezérlő megvalósítása és tesztelése 6](#_Toc506839647)

[2.1 A DMA vezérlő rendszerbe illesztése 6](#_Toc506839648)

# Mérés: Egyszerű processzoros rendszer létrehozása

## 1.1. Hardver

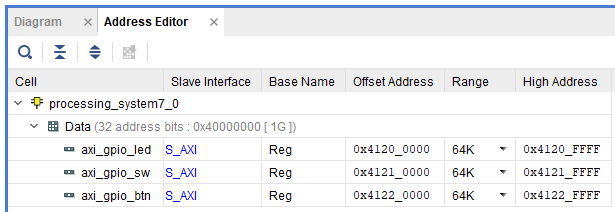
A processzoros rendszert a Xilinx Vivado Block Design-jával hoztuk létre. A rendszer blokkvázlata az **1. ábrán** látható.



1. ábra: Processzoros rendszer blokkvázlata

A blokkvázlat elemei:

* Processor System Reset: A processzoros rendszer megfelelő resetelése a feladata. Külön reset vonallal rendelkezik az interconnect és a perifériák felé
* ZYNQ7 Processing System: A Zynq mag
* AXI Interconnect: A processzor mag a perifériákhoz AXI buszon keresztül kapcsolódik. Ezt a kapcsolatot az AXI interconnect teszi lehetővé, amely crossbar belső struktúrával rendelkezik. Az interconnect a processzr felé Slave-ként viselkedik, a perifériák felé Master-ként
* System ILA: A ChipScope-nak az Integrated Logic Analyzer blokkja, amely segít a rendszer debug-olásában
* AXI GPIO blokkok: A rendszer három perifériát kezel. A gombokat, amely 5 bit széles bemeneti periféria. A LED-eket, amely 8 bit széles kimenet. A kapcsolókat, amely 8 bit széles bemenet. Ezek a perifériák az interconnect-hez Slave módban kapcsolódnak



2. ábra: Periféria címkiosztás

A perifériák címkiosztása a **2. ábrán** látható.

## Szoftver

Az elkészült hardverhez ez egyszerű 8 bites bináris számlálót írtunk, amely értékét a LED-eken jelenítjük meg, másodpercenként elvégezve a számláló inkrementálását.

A C nyelven írt szoftver kódja (main.c):

|  |
| --- |
| //xparameters, xil\_io  #include "xparameters.h"  #include "xil\_io.h"  int main()  {  //init gpio  Xil\_Out32(XPAR\_AXI\_GPIO\_LED\_BASEADDR + 4, 0x00);  Xil\_Out32(XPAR\_AXI\_GPIO\_SW\_BASEADDR + 4, 0xFF);  Xil\_Out32(XPAR\_AXI\_GPIO\_BTN\_BASEADDR + 4, 0xFF);  volatile int delay;  int cntr = 0;  while(1)  {  //1mp wait  for(delay = 0; delay < (XPAR\_CPU\_CORTEXA9\_0\_CPU\_CLK\_FREQ\_HZ / 20); delay++) ;  //write to led  Xil\_Out32(XPAR\_AXI\_GPIO\_LED\_BASEADDR, cntr);  //increment counter  cntr++;  }  } |

A kód két header inkludolásával kezdődik. Az első az „xparameters.h”, amely tartalmazza a perifériák báziscímét, és egyéb konstansokat. A „xil\_io.h” a perifériakezelésben segít, írás/olvasás függvényeket tartalmaz.

A perifériák inicializálásakor a LED-ek irányát meghatározó regiszterbe (báziscím + 4) 0x00-át írunk, tehát ez kimenet lesz. A másik két periféria regiszterébe 0xFF-et írunk, tehát ezek bemenetek lesznek.

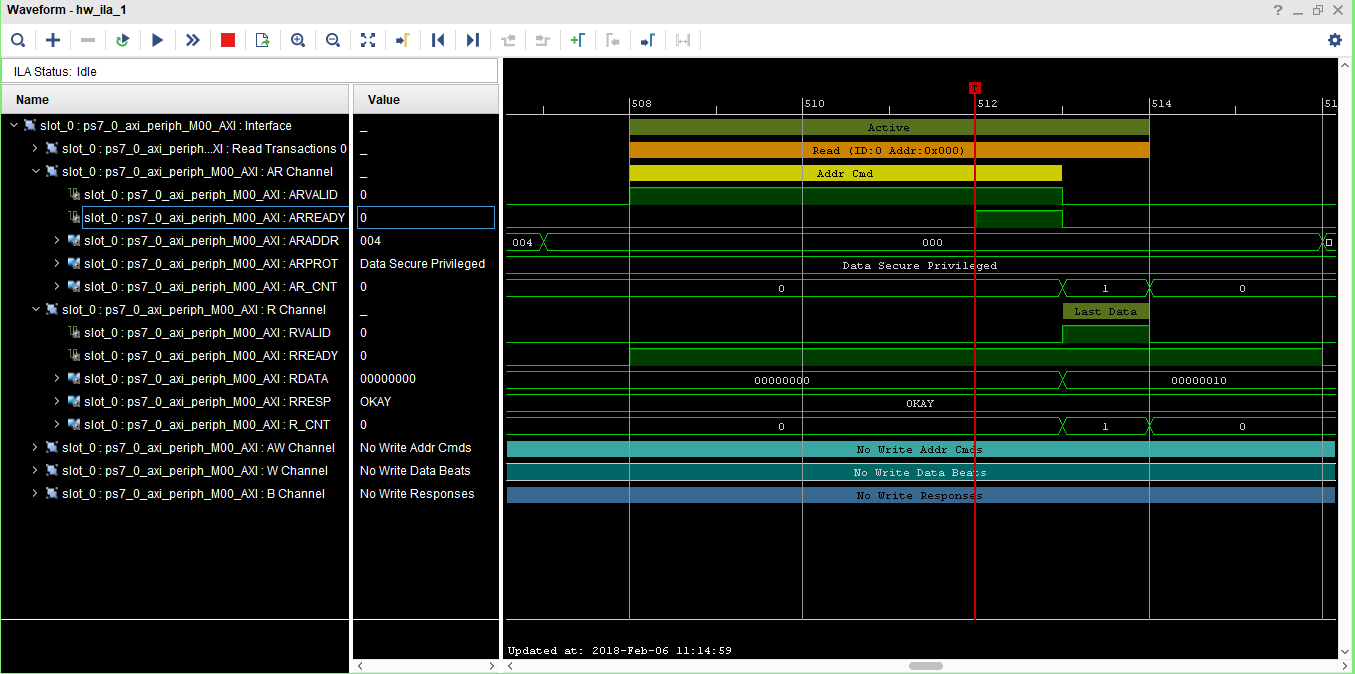
A másodperces késleltetés előállítását egy for ciklussal valósítjuk meg, amelynek üres a törzse. Itt fontos, hogy a ciklusszámláló változó (delay) volatile legyen, mert így a fordító nem optimalizálja ki az üres for ciklust, tehát a késleltetés megmarad. A késleltetéshez meg kell határozni, hogy a ciklus meddig számoljon. Ehhez használjuk a processzor órajelét leíró makrót (az „xparameters.h” fájlból), amely értéke 666666687. Ez a processzor órajelének frekvenciája Hz-ben. Az egy másodperc késleltetéshez a processzornak 666666687 órajelet kell várnia, ezért elsőre logikusnak tűnik ezt a számot beírni a for ciklus számolásának felső határához. Azonban a processzor nem hajt végre egy iterációt egy órajel alatt. A for ciklus üres ciklustörzs esetén, többnyire három utasításból áll, inkrementálás, komparálás, feltételes ugrás. A késleltetés értéke függ még a pipeline latency-től, a branch prediction-től és a memóriahozzáférés idejétől. Ezek miatt az okok miatt lett a for ciklus felső határa az órajel / 20 körülbelüli érték.

A késleltetés után történik a LED-ek meghajtása a számláló értékével. Ez után történik a számláló inkrementálása.

## Tesztelés

A teszteléshez a ChipScope logikai analizátort használtuk. Az AXI busz kommunikációját ellenőriztük olvasás és írás esetén. A ChipScope ILA egységét úgy kellett beállítani, hogy a triggerelés feltétele a handshake jelek (Valid és Ready) logikai ÉS kapcsolata legyen, és a mintatár tartalmazza a trigger esemény bekövetkezése előtti és utáni adatokat (jelen esetben 512 – 512 adat).

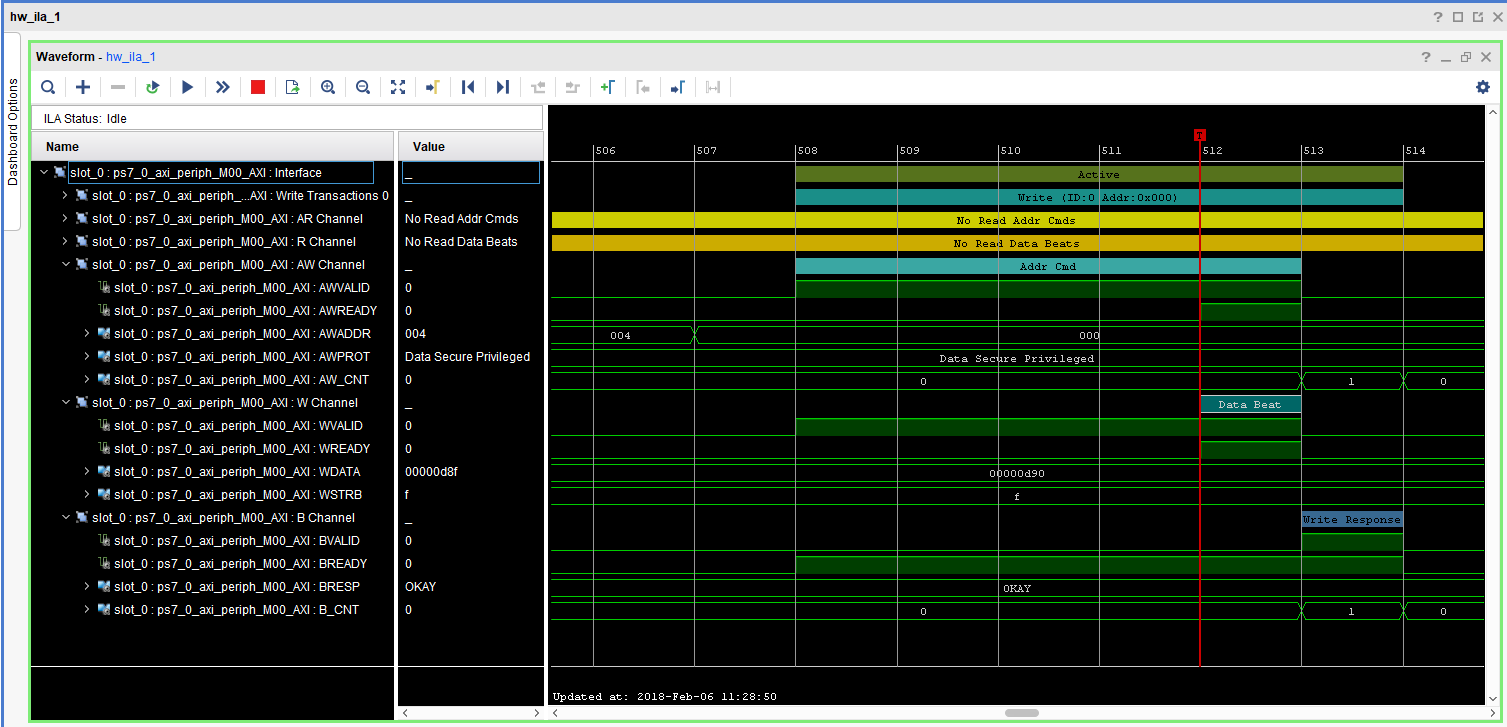
Az AXI olvasás a **3. ábrán** látható.



3. ábra: AXI olvasási buszciklus

Először az olvasási cím vonalon történik meg az információcsere, majd az olvasási adat vonalon. A read response (RRESP) vonal végig „Okay” állapotban van, tehát a kommunikáció közben nem lépett fel hiba.

Az AXI írás a **4. ábrán** látható.



4. ábra: AXI írási buszciklus

Az AXI íráskor a címet és az adatot ugyanabban az órajelben küldjük. Erre érkezik egy válasz a response csatornán.

# Mérés: DMA vezérlő megvalósítása és tesztelése

## 2.1 A DMA vezérlő rendszerbe illesztése

Az előző laborgyakorlat elvégzésével kapott rendszer-blokkvázlatot több új blokkal egészítettük ki, valamint módosítottuk a meglévőket, úgy, hogy az alkalmas legyen egy DMA (közvetlen memóriaelérés) adatátvitelre. A feladat kiírása szerint a Zynq CPU egy 128 byte-os csomag átvitelére utasítja a DMA egységet, amely egy AXI-Stream buszon közvetlen összeköttetésben van egy FIFO-val. Ezután a FIFO kimenetén a legrövidebb idő alatt megjelenik a még érkező csomag, amit a DMA egység továbbít a DDR memória felé. Továbbá egy összefűző (Concat) blokkon keresztül a DMA megszakításvezérléshez szükséges jeleit a CPU-ba vezettük.   
A DMA-egységen áthaladó csomagok követésére a DMA-hoz csatlakozó adatot szállító buszokra a ChipScope 1-1 csatornáját rögzítettük, mind az MM2S, mind az S2MM irányba.  
A korábban már meglévő AXI Interconnect blokkba egy új AXI Lite busz kapcsolatot is kialakítottunk a CPU és a DMA egység között a DMA felprogramozásának céljából.

A blokkok hozzáadása után elvégeztük a mérési segédletben megadott paraméterezéseket, és a reset, valamint a clock jelek bevezetését is az új blokkokba.

A képen szöveg látható

A leírás teljesen megbízható

2‑1. ábra: A kiegészített rendszer Blokkvázlata

## 2.2 A CPU-n futó kód

A hardverterv szintetizálása után az SDK-ban betöltöttük a megadott mintaalkalmazást. Ebben a fájlban a következő módosításokat hajtottuk végre:

* 128 byte-os csomaghossz használatához a MAX\_PKT\_LEN szimbólumot 0x80-ra módosítottuk.
* 1 csomag küldéséhez a NUMBER\_OF\_TRANSFERS szimbólumot 1-re módosítottuk.

A működőképesség, és az elvárt feladat teljesítéséhez más módosítást nem végeztünk a kódban. További egyszerűsítést jelenthetett volna a xil\_printf függvények törlése, amelyeket a fejlesztés során nem használtunk.