Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Automatizálási és Infokommunikációs
Intézeti Tanszék

2019/2020. tanév
I-II. félév

Beágyazott rendszerek c. tantárgy
előadásának és gyakorlatának ütemterve
BSC Villamosmérnöki Szakirányos hallgatók részére
Tervezés-gyártás szakirány
GEVAU519B

Tárgynév:	Beágyazott rendszerek					
Rövid név:	Beágy. rendsz.			*Kód*	GEVAU519B	
Angol név:	Embedded Systems					
Tanszék:	Automatizálási Tanszék					
Tárgyfelelős:	Dr. Vásárhelyi József vajo@uni-miskolc.hu					
Előtanulmányok:	Digitális rendszerek I.,II.,III			*Kódja:*	GEVAU 505B	
Kredit:	5		*Követelmény:*	Aláírás, Kollokvium		
Heti óraszámok	*Előadás:*	2	*Gyakorlat:*	2	*Labor: 2*	
Oktatási cél:	A digitális rendszerek és a beágyazott rendszerek tervezésében alkalmazott elvek és elméleti ismeretek elsajátítása					
Tárgy tartalom:	Beágyazott rendszerek áttekintése, Beágyazott rendszer elemzése, követelmények, trendek, Moore törvénye, Hardver elemek, FPGA és CSOC struktúrák, processzor technológiák, IC technológiák, tervezési technológiák. Tesztelés és ellenőrzés (verifikáció). Általános és beágyazott célú hardverek és szoftverek. Beágyazott rendszerek felépítése. Fejlesztési környezet. Xilinx Vivado fejlesztési környezet. Memória szerepe a beágyazott rendszerekben. Interfész technika. Szabványos kommunikációs protokollok. Tervezési példa: digitális kamera.					
Irodalom:	1. Vahid F., Givargis T.:Embedded System Design, a Unified Hardware/ Software Indtroduction,Wiley and Sons, ISBN 0-471-38678-2, 2002, pp. 324. (k) 2. Li Q., Yao C.: Real-Time Concepts for Embedded Systems, CMP Books, ISBN: 1-57820-124-11993 (a) 3. elearning anyag az ekönyvtárban 4. Peter Wilson, Design Recipes for FPGAs using Verilog and VHDL, Newnes, ISBN 978-0-08-097129-2,2007, pp. 370 5. C. “Max” Maxfield: The Design Warrior’s Guide to FPGAs, Elsvier,ISBN: 0-7506-7604-3, 2004, pp. 560 6. L. H. Crocket, Ross A. Eliott, M. A. Enderwitz, R. W. Stewart, The Zynq Book,Strathclyde Academic Media, www.zynqbook.com, 2014, pp. 460					

Mintatantervi elhelyezkedés szakok szerint

Szak	*Szakirány/sáv*	*Tantervi modul-tantervi kód*	*Mintatantervi félév*	*Választhatóság*
Villamosmérnöki Szak	Elektronikus tervezés és gyártás		6	kötelező

Jellemző oktatási módok	
Oktatási nyelv:	Magyar, angol
Előadás:	Minden hallgatónak előadás, számítógépes vetítés és tábla
Gyakorlat:	Laboratóriumi és tantermi gyakorlatok
Labor:	Maximum 16 fős csoportokban, Digitális rendszertechnikai laboratóriumban vezetett gyakorlatok, önálló mérések és feladatok teljesítésével.
Évközi feladatok, zárthelyik:	Kétszer 2x1 órás évközi zárthelyi dolgozat. Egy tervezési feladat megoldása önálló terv-feladat keretében, jegyzőkönyvvel.

Lezárási feltételek:	Gyakorlatokon aktív részvétel; az előírt tervezési feladatok teljesítése; a két évközi zárthelyi dolgozat eredményes megírása; A mérési jegyzőkönyv beadásának a tanszéki feladatbeadás határideje a következő gyakorlat kezdete, beadási mód: elektronikus; értékelés 1-5.-ig; Az évközi munka értékelése: Zárthelyi dolgozatok eredménye legalább elégséges > 60%, Gyakorlati feladatok önálló teljesítése legalább elégséges > 60%; - 24-28 elégséges, 28-32 közepes 32-36 jó, 36-40 jeles
Ütemterv	
1.	Ea Beágyazott rendszerek áttekintése, Beágyazott rendszer elemzése tervezési kihívások, követelmények, trendek, Moore törvénye. Lab: Vivado Xilinx Embedded workshop lab 1.
2.	Ea: Xilinx Vivado fejlesztési környezet sajátosságai. A fejlesztőkörnyezet jellemzői. Lab: Vivado Xilinx Embedded workshop lab 2.
3.	Ea: Hardver elemek, FPGA és CSOC struktúrák, processzor technológiák, IC technológiák, tervezési technológiák a beágyazott rendszerek tervezésében. Lab: Vivado Xilinx Embedded workshop lab 3.
4.	Ea: Általános célú processzorok, célprocesszorok, feladat-specifikus processzorok használata a beágyazott rendszerek tervezésében. Lab: Vivado Xilinx Embedded workshop lab 4.
5.	Ea:. Tesztelés és ellenőrzés (verifikáció). Általános és beágyazott célú hardverek és szoftverek. Beágyazott rendszerek felépítése. Lab: Vivado Xilinx Embedded workshop lab 5.
6.	Ea: Szoftvertervezés, hardvertervezés, hardver-szoftver együttes tervezése és szimulációja. Lab: Egyéni feladat hardwer és szoftver fejlesztés.
7.	Ea: Memória szerepe a beágyazott rendszerekben. Interfész technika. Beágyazott rendszerekben használt szabványos interfészek ismertetése, kezelése. Lab: Egyéni feladat készítése, jegyzőkönyvvel. Feladatbeadás a 9. héten.
8.	Ea: Szabványos kommunikációs protokollok. Beágyazott rendszerekben haszált szabványos kommunikációs protokollok ismertetése, kezelése. Lab: Egyéni feladat hardwer és szoftver fejlesztés.
9.	Ea: Memória szerepe a beágyazott rendszerekben. Memóriakezelés. Külső és belső memóriák kezelése FPGA illetve SOC rendszerekben.. Lab: Egyéni feladat hardwer és szoftver fejlesztés.
10.	Ea:. Motorvezérlés, mint beágyazott rendszer feladat. Léptetőmotorok és váltóáramú motorok vezérlése szabályzása. Tervezési példa: digitális kamera tervezése. Állapotgépek és konkurens folyamatok kezelése Lab: Egyéni feladat hardwer és szoftver fejlesztés.
11.	Ea:. : Modellek és programozási nyelvek, programozási nyelvek és grafikus tervbeviteli módszerek összehasonlítása. Véges állapotú állapotgép tervezése Lab: Egyéni feladat hardwer és szoftver fejlesztés.
12.	Ea. Processzek/folyamatok kommunikációja, szinkronizálása, megvalósítása/implementációja. Valós idejű operációs rendszerek. Digitális szabályozási rendszerek tervezése. Lab: Egyéni feladat hardwer és szoftver fejlesztés.
13.	Ea: IC technológiák szerepe a beágyazott rendszerekben. Lab: Pótlás
14.	Ea: konzultáció, zárthelyi Lab: Pótlás

Intézetigazgató

Dr. Trohák Attila
egyetemi docens

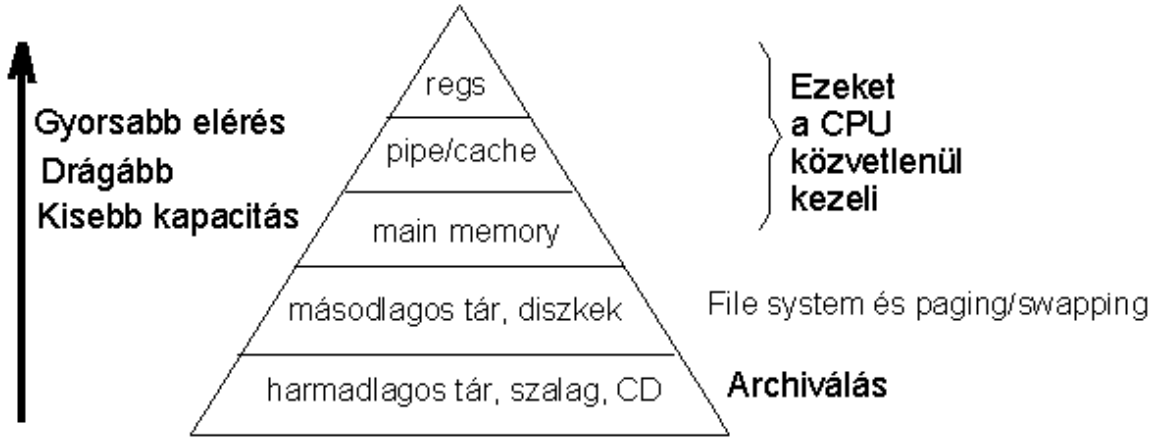
Tárgyfelelős:

Dr. Vásárhelyi József
egyetemi docens

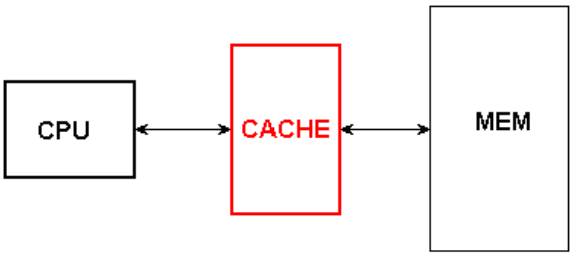
GVAU 519B ZH és megoldás:

Ísmertesse a CACHE TÁRAK (GYORSÍTÓ-TÁRAK) fajtáit és ismertesse a teljesen asszociatív cache működését.

Megoldás:
A tároló hierarchia ismertetésekor már említésre került a cache-tárak fontos szerepe az adatforgalom gyorsítása és egyenletessé tétele szempontjából.



Az ebben a pontban tárgyaltak elsősorban a processzor és a központi memória közötti tárolóra vonatkoznak, de elveit, jellegét tekintve, a központi tár és a háttértárolók közötti adatátvitelnél használt cache-tároló is ugyanúgy működik.

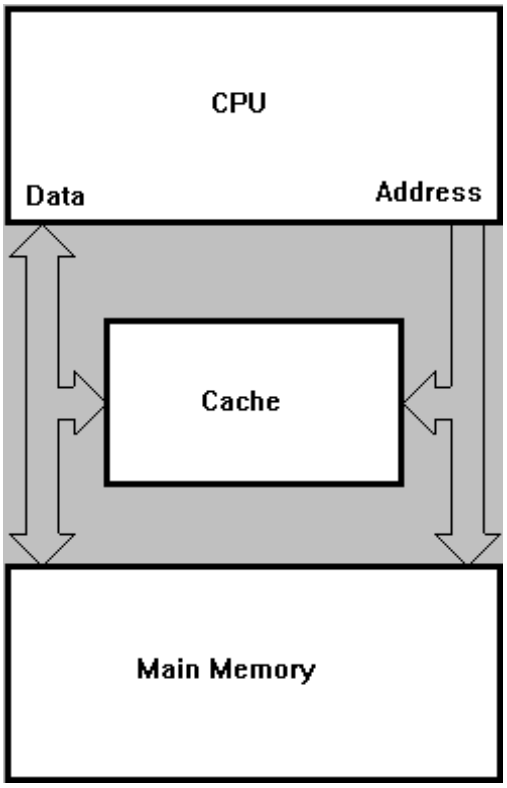


A cache-tárak utasítások és adatok átmeneti tárolására egyaránt szolgáló, gyors működésű, a felhasználó számára nem elérhető tárolók.

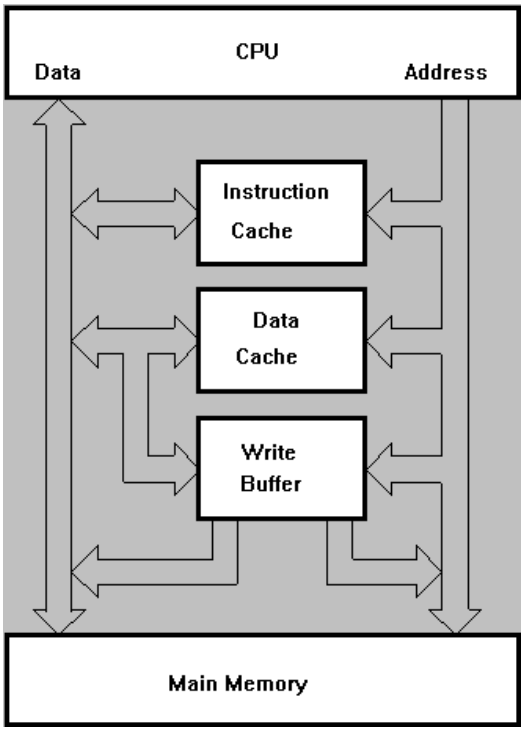
A cache-tárak fontosabb jellemzői és kialakítási szempontjai a kkövetkezőkben foglalhatók össze:

Elhelyezkedése szempontjából, a tároló lehet a mikroprocesszorba beépítve(on-chip cache), vagy azon kívüli, önálló tárolóeszköz(off-chip cache).Az általános célú belső cache szokásos mérete 8-32 Kbyte, a külső cache 64-256 Kbyte körüli.

Az adatátvitel a cache - tároló és a memória között mindig blokkos formájú, azaz egyszerre több byte-ot (4-32) visz át a processzor. Az adatátvitel formájából következően, ezek a byte-ok csak egymást követő byte-ok lehetnek a memóriában. Ez a megoldás nem kényszer szülte megoldás, hanem abból is adódik, hogy nagy valószínűséggel az utasítások, de az adatok felhasználása is az egymást követő tároló helyekről történik többnyire.



A cache-tárak szolgálhatnak kizárólag utasítások tárolására, de tárolhatnak utasításokat és adatokat együtt is, ill. lehet külön tároló mind a programutasításoknak, mind az adatoknak is.



A cache - tárolóban a memória egyes egymást követő rekeszeinek tartalmát tároljuk, a tárolóbeli hely címével együtt. A visszakeresés módja úgy nevezett tartalom szerinti

(asszociatív, CAM = content address memory), ami azt jelenti, hogy a vizsgált adatnak a cachhe-ben tárolt adattal való egyezőségét vizsgálja a processzor a kiolvasáskor, kereséskor. Ez a vizsgálat a keresett adat címének az összehasonlítását jelenti a cache-ben tárolt címekkel, vagy azok egy részével.

A cache-tár akkor működik hatékonyan, ha a keresett adat a kiolvasások többségében a cache-ben és nem a memóriában található. A találatok (cache-hit) száma függ a cache-tár méretétől és szervezési módjától. 10%-os találati hiba (cache-miss) általában még elfogadható arány.

A cache-tároló tartalmának cseréjekor, a találati arány fenntartása érdekében, lényeges a megfelelő helyettesítési stratégia (replacement policy) kiválasztása.

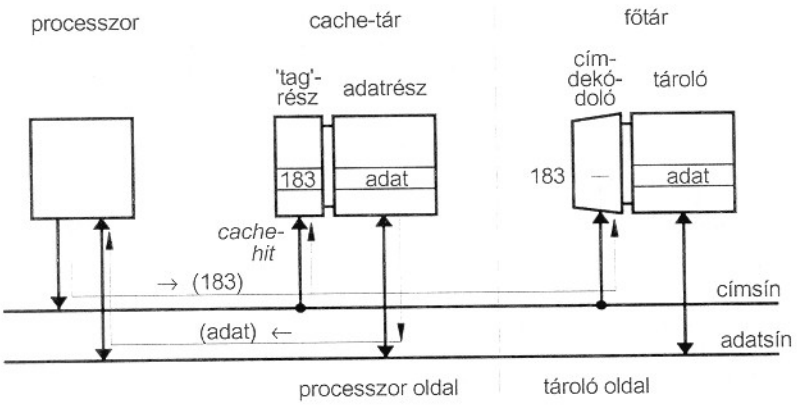
Lényeges szempont a cache-tár tartalmának és a központi tár azonos részei tartalmának egyezőségét biztosítani.

Kívánatos, hogy a processzor és a cache-tár működési sebessége azonos legyen.

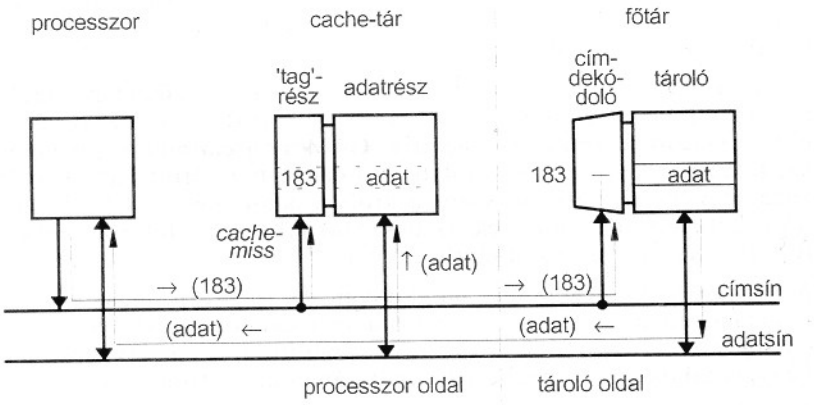
A processzor, a cache-tár és a memória közötti adatmozgatást mutatja be találat (cache-hit) és nem-találat (cache-miss) esetére a következő, 4-4 ábra.

Ha a keresett adat a cache-tárban található(cache-hit, read/write-hit), akkor onnét veszi ki a processzor. Ha nincs a cache-tárban a keresett adat (cache-miss, read/write-miss esete), akkor az, a memóriából kiolvasva, egyúttal a cache-tárba is beírásra kerül.

a.) cache-hit esete



b.)cache-miss esete



b.)cache-miss esete

4-4.ábra: Adatmozgatás cache-találat és –hiba esetében

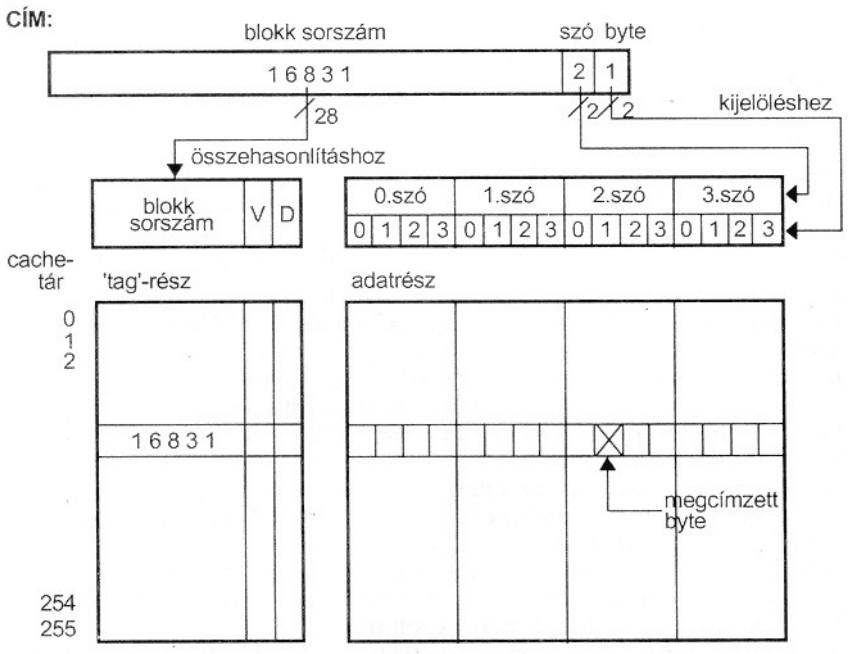
4.3.1. Cache-tárak típusai

A bevezető részben említésre került már, hogy a cache-tárak tartalom szerinti visszakeresést tesznek lehetővé. A visszakeresés a keresett adat címe alapján történik, ezért azt valamilyen módon a cache-tárban el kell helyezni. A cím tárolásakor, annak csak egy részét szükséges magában a cache-ben elhelyezni, még pedig csak akkora részt, amelynek alapján közvetlenül (tehát a tárolt értékből), vagy közvetve (a tárolt értékből és annak cache-beni helyéből, sorából) a blokk kezdőcíme meghatározható. A címnek azt a részét, amelyet a cache-tárban elhelyez a processzor és amelynek összehasonlításával történik a választás, „**tag**”-nek nevezik. A cache-ben „tag”-ként tárolt címrész származhat a virtuális címből, vagy a fizikai címből, attól függően, hogy a cache-tár a processzor és a címfordító egység (MMU), vagy címfordító egység és a főtár között helyezkedik-e el.

A választást meghatározó cím mellett, a tárolt adatok állapotára vonatkozó információkat is tárol a cache-tár. Ezek a vezérlést és a helyettesítési eljárást kiszolgáló bitek, amelyek mindegyike nem található meg mindig a cache-tárban. A két legfontosabb vezérlő bit:

V (valid bit), a cache-tár tartalmának(blokk, sor, byte) az érvényességét jelzi, azaz azt, hogy az adat a megadott című tárolóhelyhez tartozik és az aktuálisan érvényes adat. A cache-tár törlésekor (flushing, reset), minden V-bit 0-ra lesz beállítva és egy új adat betöltésekor lesz V=1 értékű. Minden blokkhoz legalább egy V-bit tartozik, de van olyan megoldás is, ahol minden egyes byte-hoz 1-1 V-bit tartozik.

D(dirty bit), a blokk valamely részének a módosítását, felülírását jelzi. Az ilyen blokk helyére (ha D=1) nem lehet betölteni új blokkot, előbb a régit ki kell vinni a főtárba.



A jelzőbitek közül a legfontosabbak az érvényességet(V) és a módosítást (D) jelző bitek, amelyek használata a cache-tár működtetéséhez elengedhetetlen.

A cache-tárak legfontosabb jellemzőiként az alábbiak adhatók meg:
a cache-tár mérete(cache-size), amely 8-256 KB között mozog, attól függően is, hogy belső(on-chip), vagy külső (off-chip) cache-tárról van szó;

blokk-méret (block-size, block refill-size),amely megadja a főtár és a cache-tár között az egy egységben mozgatott adatmennyiséget; ez utasításoknál nagyobb, adatoknál kisebb érték szokott lenni, 1-8 szó(4-32byte) nagyság között;

sorméret (line size), az adatmennyiség, amely egy-egy összehasonlítással kijelölhető és amelynek mérete a blokk méreténél kisebb, de általában avval megegyező;

helyettesítési algoritmus (replacement policy), amely meghatározza a módot, ahogy a felesleges(kicserélhető) blokkot a cache-tárban kiválasztjuk egy-egy új blokk betöltésekor;

adataktualizálási módszer (write strategy), az eljárás, amellyel a módosítandó adatot a cache-tárba és a főtárba írjuk;

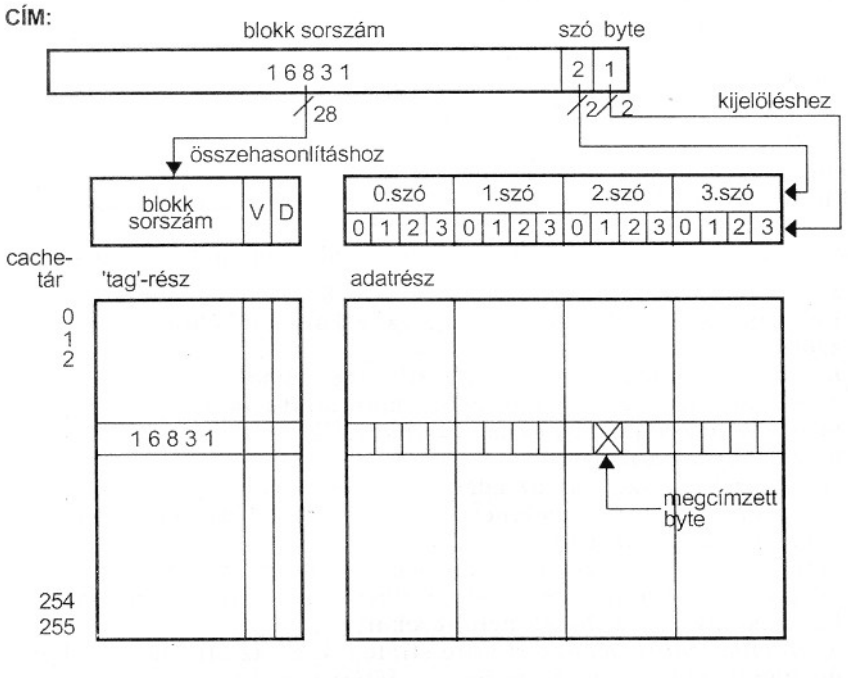
adategyezőség-biztosítási mód (coherency mechanism), amely meghatározza azt a módszert, amellyel biztosítani lehet a főtár és a cache-tár(ak) tartalmának az egyezőségét.

A különböző tárolási módszerek bemutatásához használt példában azzal a feltételezéssel élünk, hogy az adatátvitelnél a blokkok mérete: 4 szó=16 byte és a cache-tár 256 ilyen blokk + cím + jelzőbitek befogadására alkalmas, azaz 256 cellája, sora (line) van. A tároló helyek virtuális, vagy fizikai címe 32 bites. Mivel a blokk hosszúsága 16 byte, ezért csak 16-tal osztható címek jöhetnek szóba, mint kezdőcímek. A cím alsó 4 bitje ugyanis a blokkon belüli byte-ot jelöli ki. (A konkrét esetekben ezek a méretek a megadottaktól eltérőek lehetnek!)

A cache-tár fizikai megvalósításában külön részt képez az adattároló rész és külön részt a címet („tag”-et) tároló rész.

a.) Teljesen asszociatív (fully associative) cache

A teljesen asszociatív tárban (amelyet másképpen, a jelenlegi példa 256 sora alapján, 256-way set associatív cache-nek is nevezhetünk) a beolvasott blokk bárhová elhelyezhető, bármelyik sorba kerülhet. Az elhelyezés sorát a helyettesítési algoritmus határozza meg.



A beolvasott blokk 16 byte-ja mellett, a virtuális, vagy fizikai memóriacím egy része is („tag” gyanánt) tárolásra kerül. Az ábrán bemutatott esetben ez egy 28 bites blokksorszám (4-5. ábra). A cím alsó 4 bitje a szó(4 byte) és azon belül a byte helyét határozza meg.

Amikor a processzor egy adatot keres a cache-ben, akkor a memóriabeli cím felső 28 bitjét (blokksorszám) összehasonlítja a cache-beli blokksorszámokkal (tags). Ez az összehasonlítás az összes sorban egyidőben történik, azaz a cache-hez egy olyan áramkör tartozik, amely a jelen esetben 256 párhuzamos, összehasonlító áramkört tartalmaz.

Ha a keresés sikeres (cache-hit/read-, vagy write-hit/), akkor a cím alsó 4 bitje alapján kijelöli az adott sorbeli byte-ot; ha a keresés sikertelen (cache-miss/read-
vagy write-miss/), akkor a memóriában kikeresi a kívánt byte-ot (szót) és beolvassa vagy módosítja. A cache tartalmának átírása, vagy eredeti állapotban hagyása az alkalmazott aktualizálási eljárástól függ (4.3.2.pont).

A teljesen asszociatív cache-tár előnye a rugalmasság a betöltésnél (bármelyik blokk bárhová kerülhet), de a visszakereséshez ugyanannyi összehasonlító áramkör kell, mint ahány sora a cache-tárnak van és ez költséges. Emiatt az ilyen cache-táraknál, általában 64 sornál többet nem alkalmaznak. Előnye továbbá az igen jó találati arány lehetősége, ugyanakkor hátrány a helyettesítési eljárás alkalmazásának a szükségessége.

A cache-tárban minden blokkhoz (esetleg minden byte-hoz) használnak egy **érvényességi jelzőbit**et (V=valid bit), amely arról ad információt, hogy a tár adott sora ténylegesen az aktuálisan érvényes adatokat tartalmazza-e, vagy sem.
(Egyrészt, ugyanis a byte-ok átvitele az adatsin mérete miatt nem egyszerre történik és így lehetséges, hogy a sor egy része másik blokkhoz tartozó byte-okat foglal magában, másrészt a főtár azonos helyen lévő adatait felülírhatja a valamilyen más memóriaművelet.)

A másik jelzőbit a **módosítási jelzőbit (**D=dirty bit), amely arról informál, hogy történt-e módosítás a blokk valamelyik byte-ja esetében.

b.) Közvetlen leképzésű (direct mapping) cache
c.) Csoport asszociatív (set associative) cache
d.) Szektor leképzésű (sector mapping) cache

A csoport asszociatív cache-tárhoz hasonló, köztes megoldást képező és ma a mikroprocesszorokban ritkábban használt megoldás a szektor leképzésű cache-tár.

Ennél a változatnál a processzor a csoport helyét jelöli ki asszociatív módon és azon belül a blokk helye, a lapon belüli elhelyezkedésének megfelelően kötött.
Tehát az előző megoldás fordítottja a szektor leképzésű cache-tár.

Adding IP cores in PL

Introduction

This lab guides you through the process of extending the processing system you created in the previous lab by adding two GPIO (General Purpose Input/Output) IPs

Objectives

After completing this lab, you will be able to:

- Configure the GP Master port of the PS to connect to IP in the PL
- Add additional IP to a hardware design
- Setup some of the compiler settings

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

This lab comprises 6 primary steps: You will open the project in Vivado, add and configure GPIO peripherals in the system using IP Integrator, connect external ports, generate bitstream and export to SDK, create a TestApp application in SDK, and, finally, verify the design in hardware.

Design Description

The purpose of this lab exercise is to extend the hardware design (**Figure 1**) created in Lab 1

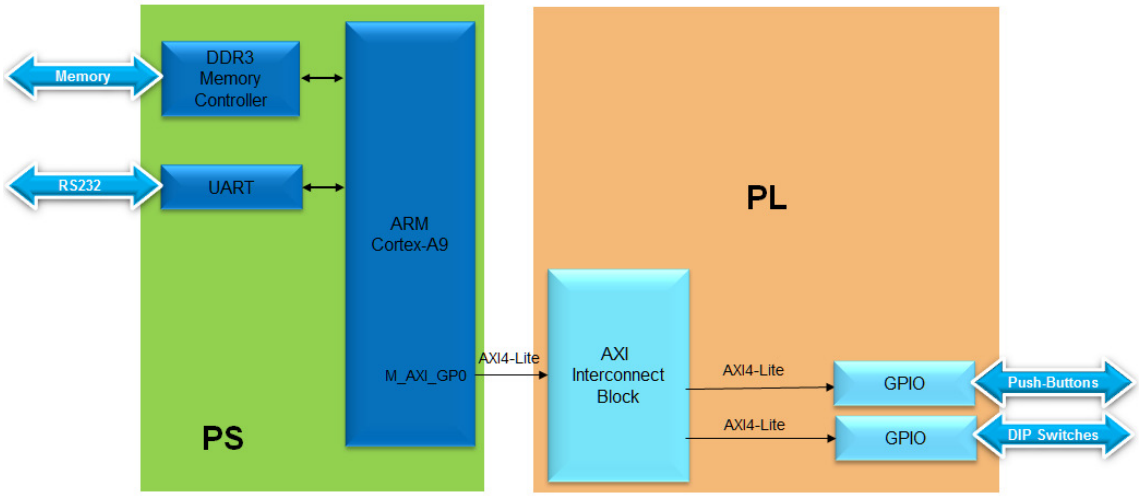
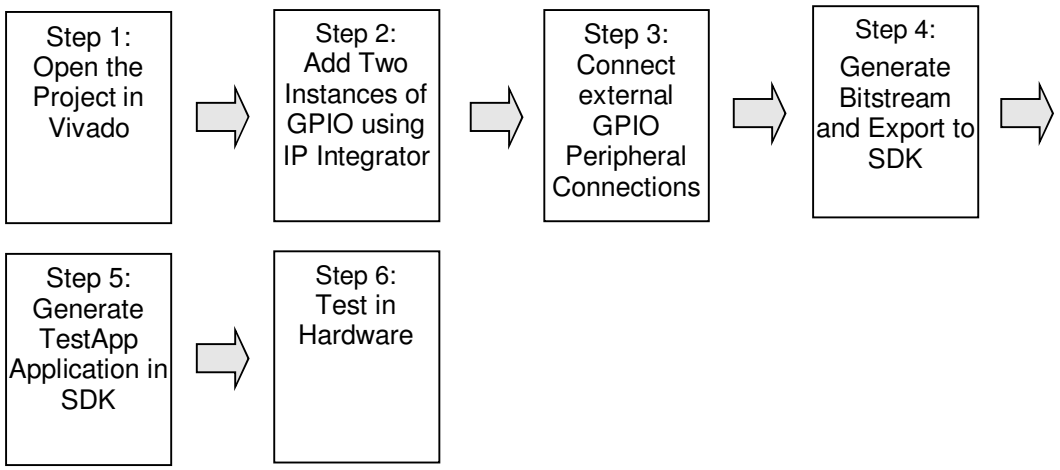


Figure 1. Extend the System from the Previous Lab

General Flow for this Lab



In the instructions below;
{***sources***} refers to: C:\xup\embedded\2015_2_zynq_sources
{***labs***} refers to : C:\xup\embedded\2015_2_zynq_labs
{***labsolutions***} for the ZedBoard refers to: C:\xup\embedded\2015_2_zedboard_labsolution
or for the Zybo refers to: C:\xup\embedded\2015_2_zybo_labsolution

1Open the Project Step 1

1-1. Open the previous project, or the lab1 project from the {labsolutions} directory, and save the project as lab2. Open the Block Design.

1-1-1. Start Vivado, if necessary, and open either the lab1 project (lab1.xpr) you created in the previous lab or from the {*labsolutions}* directory using the **Open Project** link in the Getting Started page.

1-1-2. Select **File > Save Project As …** to open the *Save Project As* dialog box. Enter **lab2** as the project name. Make sure that the *Create Project Subdirectory* option is checked, the project directory path is {labs} and click **OK**.

This will create the lab2 directory and save the project and associated directory with lab2 name.

2Add Two Instances of GPIO Step 2

2-1. Enable AXI_M_GP0 interface, FCLK_RESET0_N, and FCLK_CLK0 ports, Add two instances of a GPIO Peripheral from the IP catalog to the processor system.

2-1-1. In the *Sources* panel, expand system_*wrapper,* and double-click on the **system.bd (system_i)** file to invoke IP Integrator. (The Block Design can also be opened from the Flow Navigator)

2-1-2. Double click on the Zynq block in the diagram to open the *Zynq configuration* window.

2-1-3. Select **PS-PL Configuration** page menu on the left, or click **32b GP AXI Master Ports** block in the Zynq Block Design view.

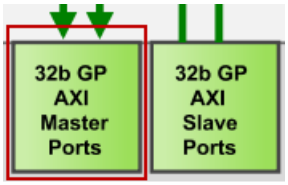


Figure 2. AXI Port Configuration

2-1-4. Expand *AXI Non Secure Enablement > GP Master AXI Interfaces,* if necessary, and click on **Enable M_AXI_GP0 interface** check box under the field to enable the AXI GP0 port.

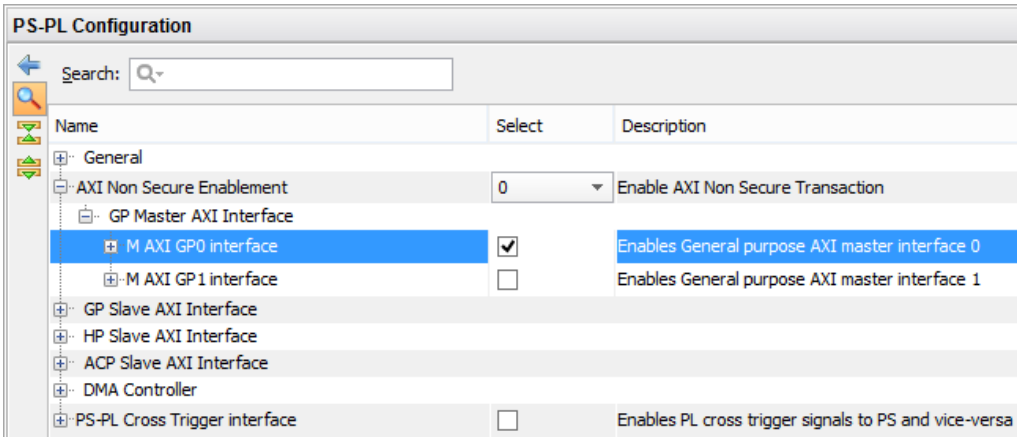


Figure 3. Configuration of 32b Master GP Block

2-1-5. Expand **General > Enable Clock Resets** and select the **FCLK_RESET0_N** option.

2-1-6. Select the **Clock Configuration** tab on the left. Expand the **PL Fabric Clocks** and select the **FCLK_CLK0** option (with requested clock frequency of 100.000000 MHz) and click **OK.**

2-1-7. Notice the additional M_AXI_GPO interface, and M_AXI_GPO_ACLK, FCLK_CLK0, and FCLK_RESET0_N ports are now included on the Zynq block. You can click the regenerate button ( ) to redraw the diagram.

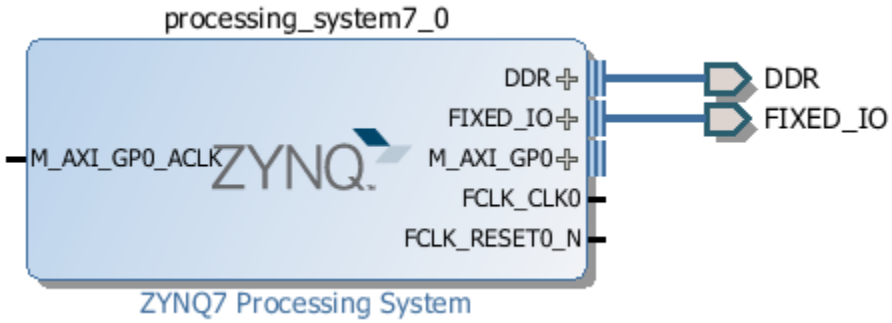


Figure 4. Zynq system with AXI and clock interfaces

2-1-8. Click the Add IP icon and search for **AXI GPIO** in the catalog

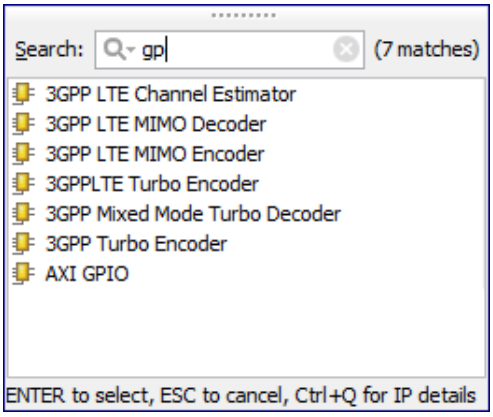


Figure 5. Add GPIO IP

2-1-9. Double-click the **AXI GPIO** to add the core to the design. The core will be added to the design and the block diagram will be updated.

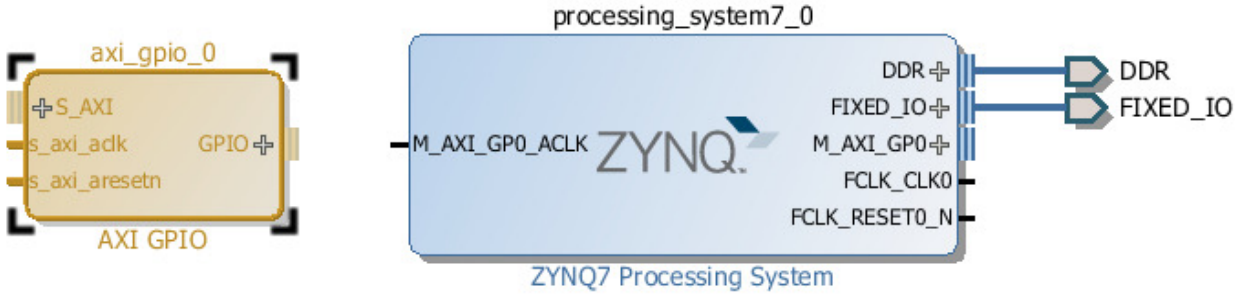


Figure 6. Zynq system with AXI GPIO added

2-1-10. Click on the **AXI GPIO** block to select it, and in the properties tab, change the name to **switches**

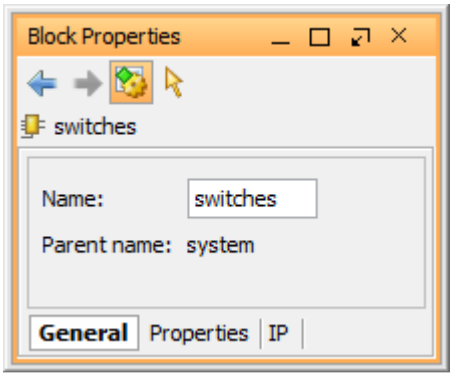


Figure 7. Change AXI GPIO default name

2-1-11. Double click on the **AXI GPIO** block to open the customization window.

2-1-12. From the **Board Interface** drop down, select **sws 8bits** for ZedBoard or **sws 4bits** for Zybo for GPIO.

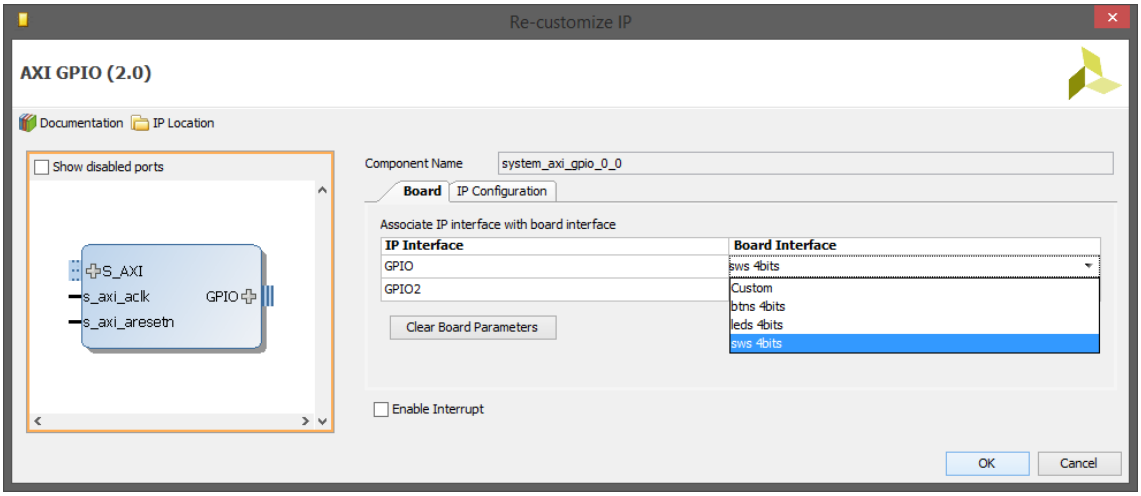


Figure 8. Configuring GPIO instance

2-1-13. Click the IP configuration tab, and notice the width has already been set to match the switches on the ZedBoard (8) or Zybo (4)

Notice that the peripheral can be configured for two channels, but, since we want to use only one channel without interrupt, leave the *Enable Interrupt* and *Enable Dual Channel* unchecked.

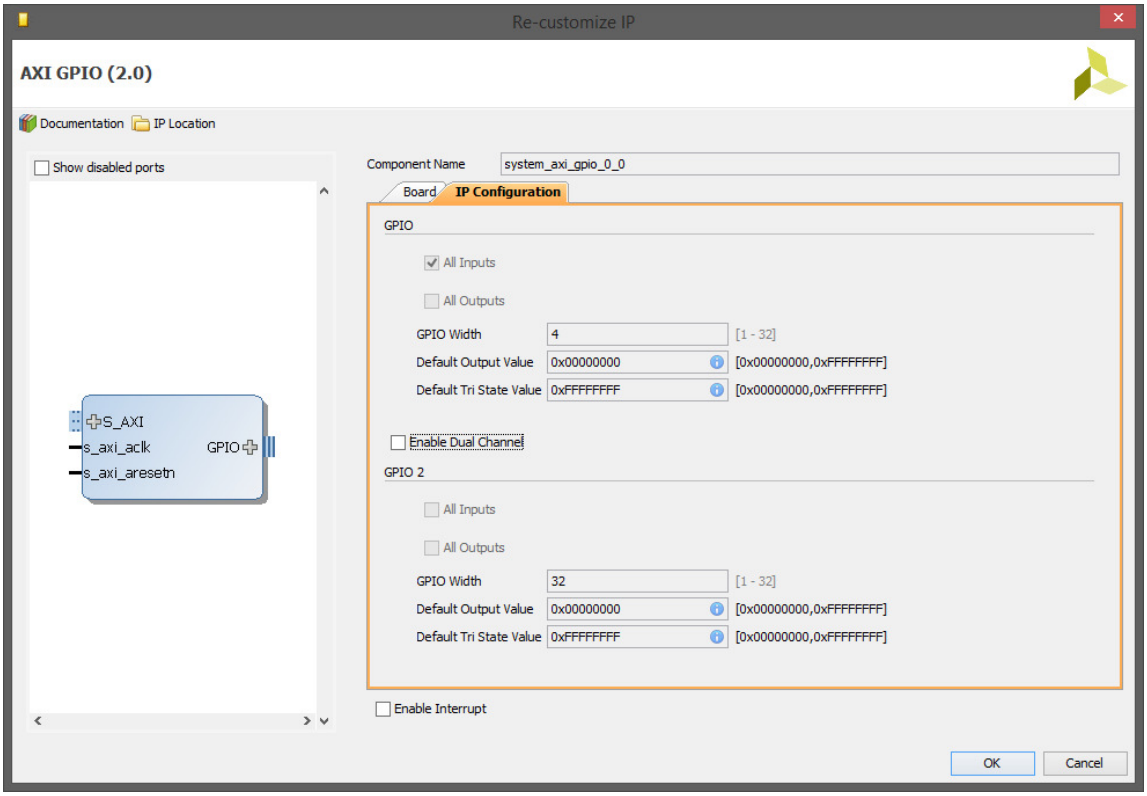


Figure 9. Configuring GPIO instance

2-1-14. Click **OK** to save and close the customization window

2-1-15. Notice that *Designer assistance* is available. Click on **Run Connection Automation**, and select **/switches/S_AXI**

2-1-16. Click **OK** when prompted to automatically connect the master and slave interfaces

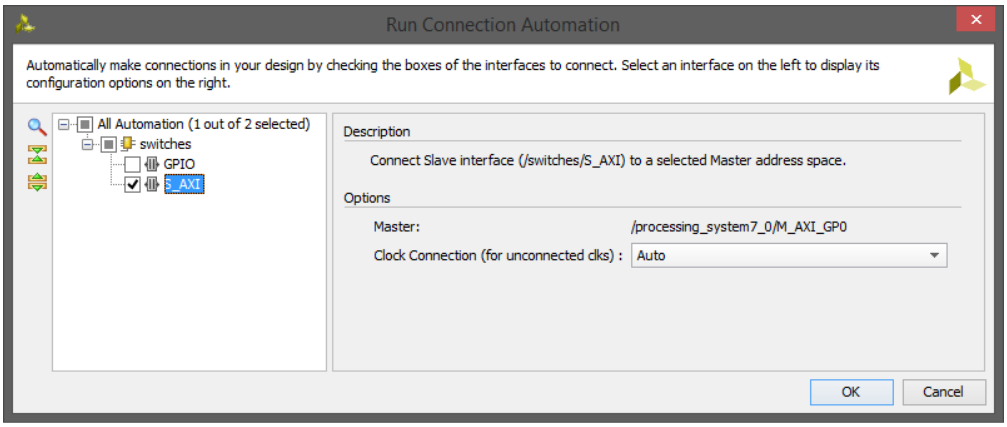


Figure 10. Run connection automation

2-1-17. Notice two additional blocks, *Processor System Reset*, and *AXI Interconnect* have automatically been added to the design. (The blocks can be dragged to be rearranged, or the design can be redrawn.)

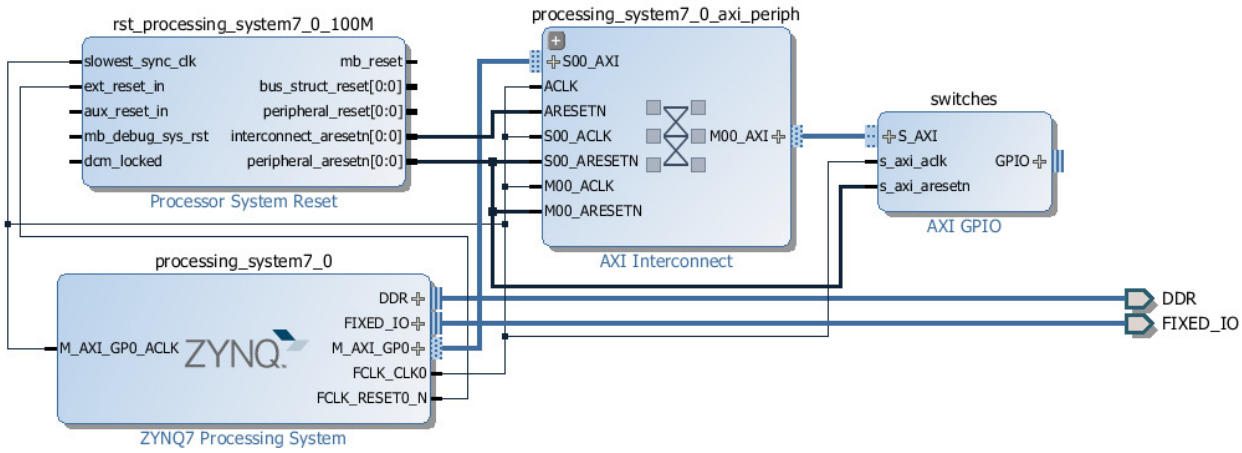


Figure 11. Design with switches automatically connected

2-1-18. Add another instance of the *GPIO* peripheral (**Add IP**). Name it as **buttons**

2-1-19. Double click on the IP block, select the *btns* GPIO interface (*btns_5bits* for the Zedboard, *btns_4bits* for the Zybo) and click **OK**.

At this point connection automation could be run, or the block could be connected manually. This time the block will be connected manually.

2-1-20. Double click on the AXI Interconnect and change the *Number of Master* Interfaces to **2** and click **OK**

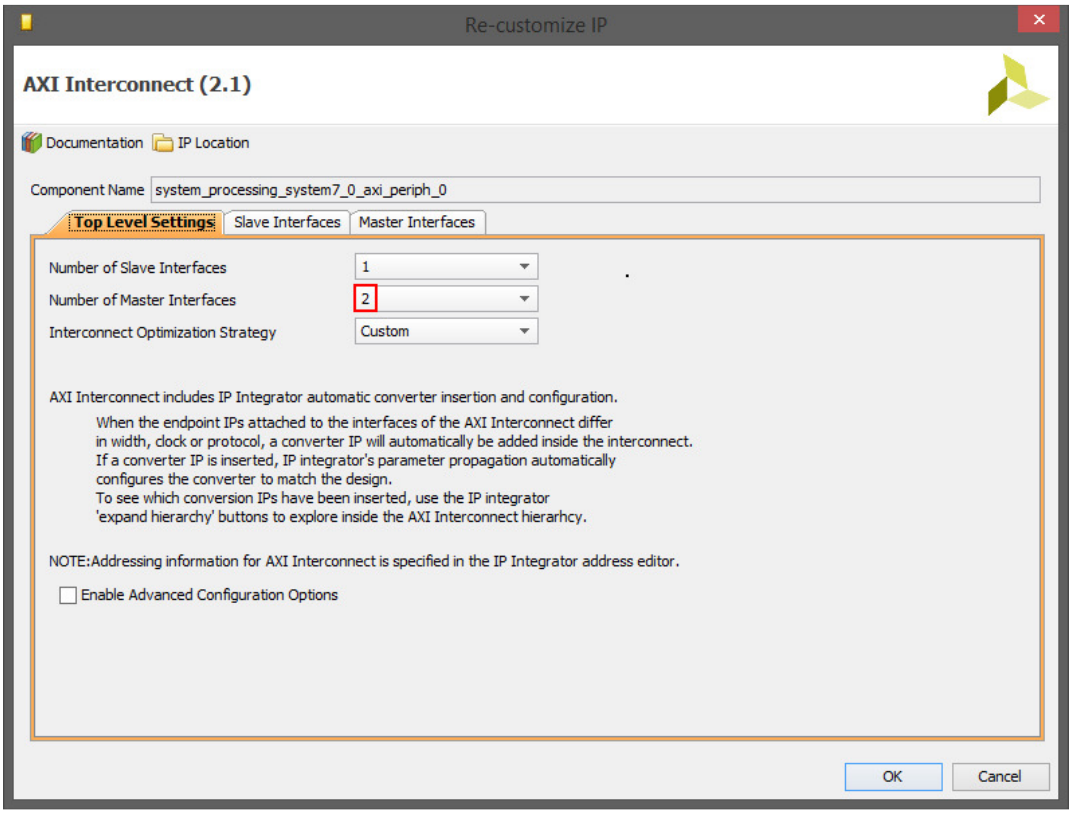


Figure 12. Add master port to AXI Interconnect

2-1-21. Click on the *s_axi* port of the buttons AXI GPIO block, and drag the pointer towards the AXI Interconnect block. The message *Found 1* interface should appear, and a green tick should appear beside the *M01_AXI* port on the AXI Interconnect indicating this is a valid port to connect to. Drag the pointer to this port and release the mouse button to make the connection.

2-1-22. In a similar way, connect the following ports:
buttons **s_axi_aclk** -> *Zynq7 Processing System* **FCLK_CLK0**
buttons **s_axi_aresetn** -> *Processor System Reset* **peripheral_aresetn**
AXI Interconnect **M01_ACLK** -> *Zynq7 Processing System* **FCLK_CLK0**
AXI Interconnect **M01_ARESETN** -> *Processor System Reset* **peripheral_aresetn**

The block diagram should look similar to this:

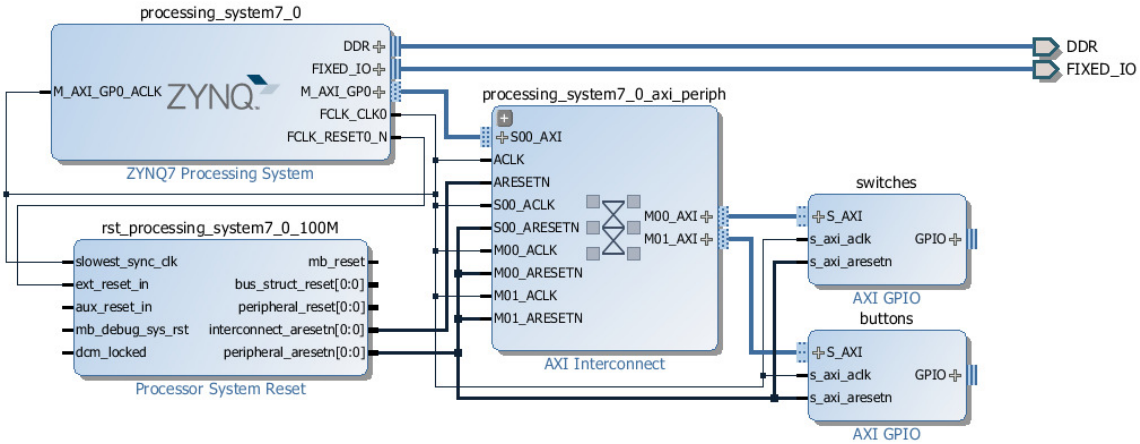


Figure 13. System Assembly View after Adding the Peripherals

2-1-23. Click on the *Address Editor* tab, and expand **processing_system7_0 > Data > Unmapped Slaves** if necessary

2-1-24. Notice that *switches* has been automatically assigned an address, but *buttons* has not (since it was manually connected). Right click on *btns_4bit* and select **Assign Address** or click on the button.

Note that both peripherals are assigned in the address range of 0x40000000 to 0x7FFFFFFF (GP0 range).

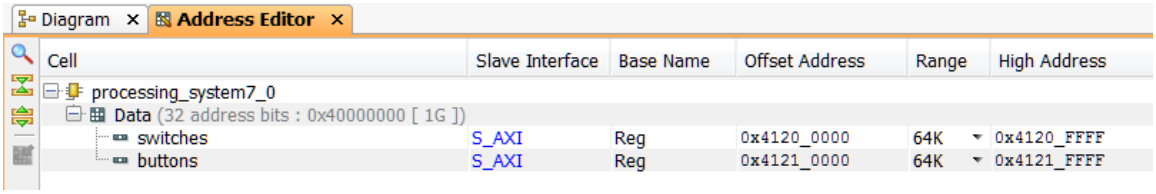


Figure 14. Peripherals Memory Map

3Make GPIO Peripheral Connections External Step 3

3-1. The push button and dip switch instances will be connected to corresponding pins on the board. This can be done manually, or using Designer Assistance. Normally, one would consult the board's user manual to find this information.

3-1-1. In the Diagram view, notice that *Designer Assistance* is available. We will manually create the ports and connect.

3-1-2. Right-Click on the *GPIO* port of the *switches* instance and select **Make External** to create the external port. This will create the external port named *gpio* and connect it to the peripheral. Because Vivado is "board aware", the pin constraints will be automatically applied to the port.

3-1-3. Select the *gpio* port and change the name to **switches** in its properties form.

The width of the interface will be automatically determined by the upstream block.

3-1-4. For the **buttons** GPIO, click on the *Run Connection Automation* link.

3-1-5. In the opened GUI, select *btns_5bits* (for ZedBoard) or *btns_4bits* (for Zybo) under the options section.

3-1-6. Click **OK**.

3-1-7. Select the created external port and change its name as **buttons**

3-1-8. Run Design Validation (**Tools -> Validate Design**) and verify there are no errors.

The design should now look similar to the diagram below

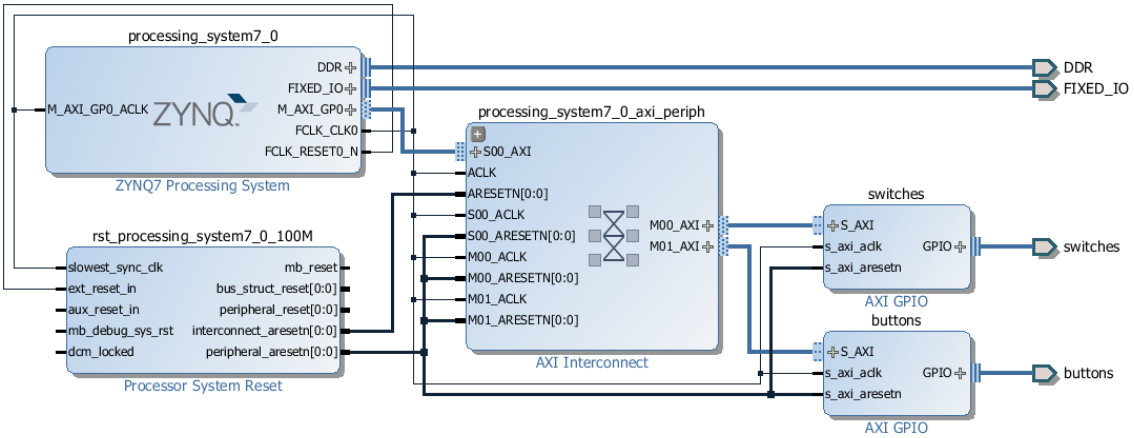


Figure 15. Completed design

3-2. Synthesize the design, open the I/O Planning layout, and check the constraints using the I/O planning tool.

3-2-1. In the Flow Navigator, click **Run Synthesis**. (Click **Save** if prompted) and when synthesis completes, select **Open Synthesized Design** and click **OK**

3-2-2. In the shortcut Bar, select **I/O Planning** from the *Layout* dropdown menu

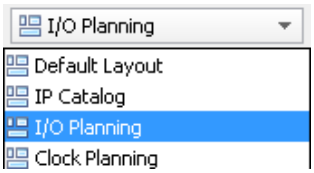


Figure 16. Switch to the IO planning view

3-2-3. In the I/O ports tab, expand the two GPIO icons, and expand *buttons_tri_i*, and *switches_tri_i*, and notice that the ports have been automatically assigned pin locations, along with the other *Fixed IO* ports in the design, and an I/O Std of LVCMOS25 (for ZedBoard) and LVCMOS33 (for Zybo) has been applied. If they were not automatically applied, pin constraints can be included in a constraints file, or entered manually or modified through the I/O Ports tab.

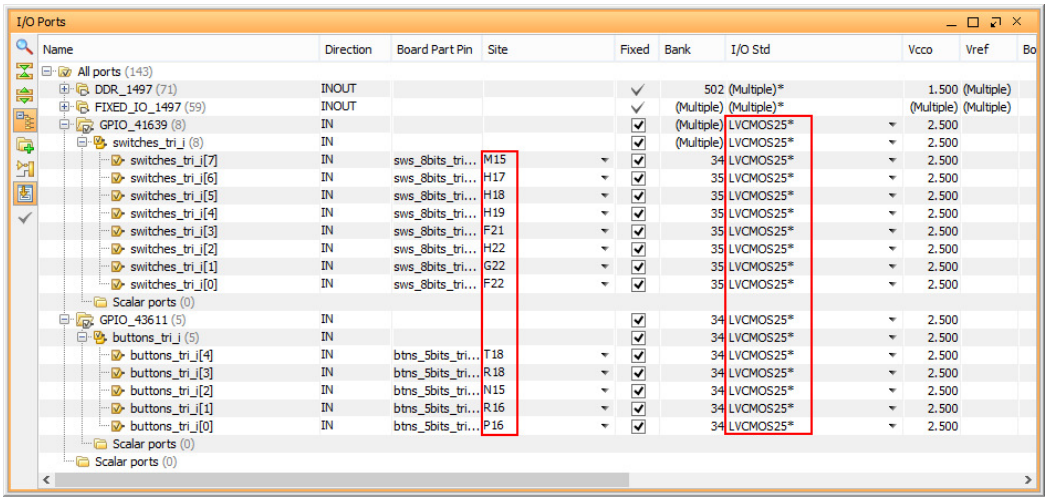


Figure 17. The IP port pin constraints for the ZedBoard

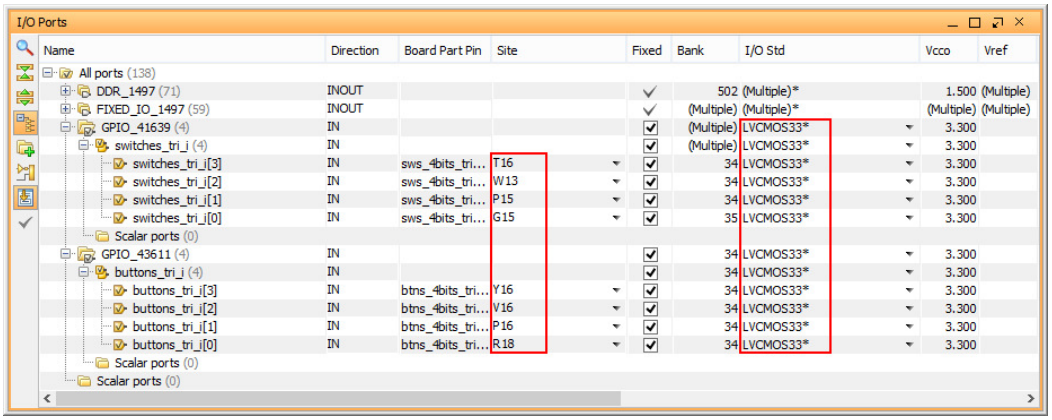


Figure 18. The IP port pin constraints for the Zybo

4Generate Bitstream and Export to SDK Step 4

4-1. Generate the bistream, and export the hardware along with the generated bitstream to SDK.

4-1-1. Click on **Generate Bitstream**, and click **Yes** if prompted to Launch Implementation (Click **Yes** if prompted to save the design)

4-1-2. Click **Cancel**

4-1-3. Export the hardware by clicking **File > Export > Export Hardware** and click **OK**. This time, there is hardware in Programmable Logic (PL) and a bitstream has been generated and should be included in the export to SDK.

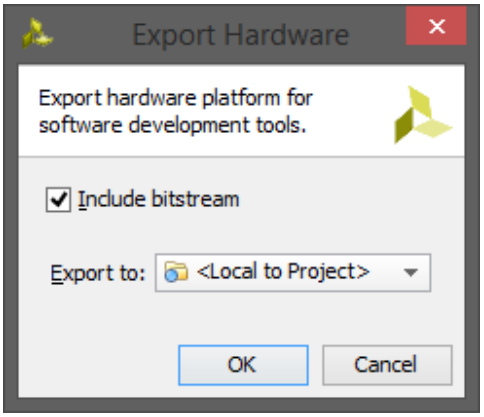


Figure 19. Export the design

4-1-4. Click **Yes** to overwrite the hardware module.

4-1-5. Start SDK by clicking **File > Launch SDK** and click **OK**

5Generate TestApp Application in SDK Step 5

5-1. Close the projects from the previous lab. Generate software platform project with default settings and default software project name (standalone_0).

5-1-1. In SDK, right click on the *mem_test* project from the previous lab and select **Close Project**

5-1-2. Do the same for *mem_test_bsp* and *system_wrapper_hw_platform_0*

5-1-3. From the *File* menu select **File** > **New** > **Board Support Package**

5-1-4. Click **Finish** with the *standalone* OS selected and default project name as *standalone_bsp_0*

5-1-5. Click **OK** to generate the board support package named *standalone_bsp_0*

5-1-6. From the *File* menu select **File** > **New** > **Application Project**

5-1-7. Name the project **TestApp**, select *Use existing* board support package, select **standalone_bsp_0** and click **Next**

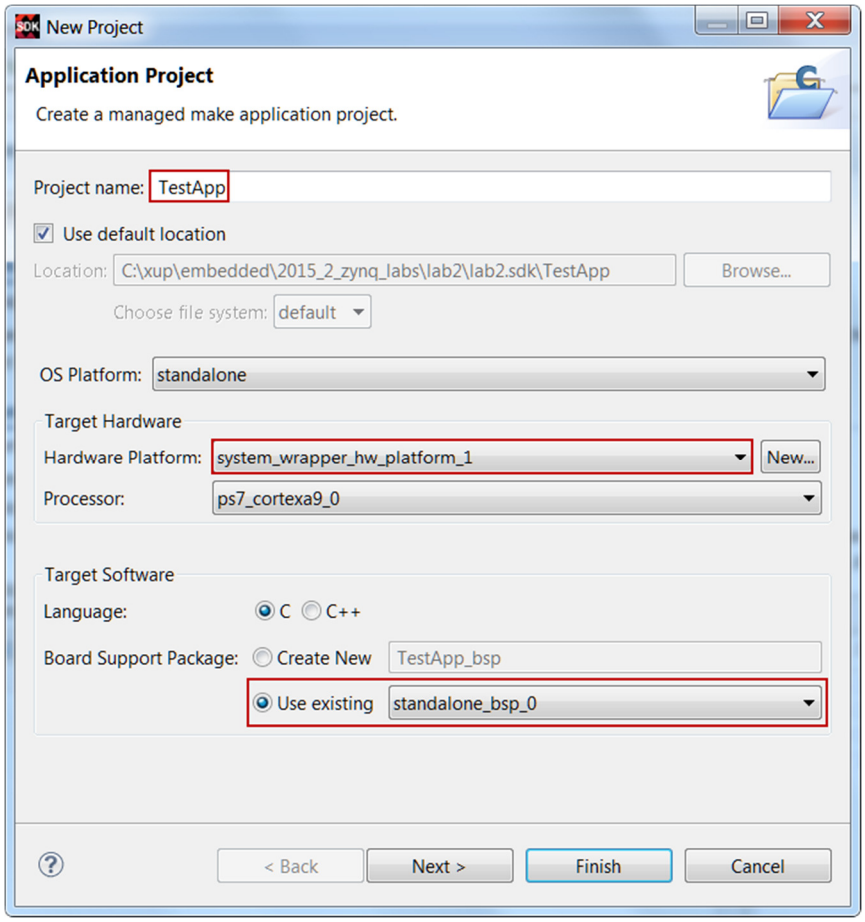


Figure 20. Application Project settings

5-1-8. Select **Empty Application** and click **Finish**

This will create a new Application project using the created board support package.

5-1-9. The library generator will run in the background and will create the **xparameters.h** file in the **lab2\lab2.sdk\standalone_bsp_0\ps7_cortexa9_0\include** directory

5-1-10. Expand **TestApp** in the project view, and right-click on the **src** folder, and select **Import**

5-1-11. Expand **General** category and double-click on **File System**

5-1-12. Browse to the {sources}**lab2** folder

5-1-13. Select **lab2.c** and click **Finish**

A snippet of the source code is shown in figure below.

```
#include "xparameters.h"
#include "xgpio.h"


//====================================================

int main (void)
{

    XGpio dip, push;
    int psb_check, dip_check;

    xil_printf("-- Start of the Program --\r\n");

    XGpio_Initialize(&dip, XPAR_SWITCHES_DEVICE_ID);
    XGpio_SetDataDirection(&dip, 1, 0xffffffff);

    XGpio_Initialize(&push, XPAR_BUTTONS_DEVICE_ID);
    XGpio_SetDataDirection(&push, 1, 0xffffffff);

    while (1)
    {
      psb_check = XGpio_DiscreteRead(&push, 1);
      xil_printf("Push Buttons Status %x\r\n", psb_check);
      dip_check = XGpio_DiscreteRead(&dip, 1);
      xil_printf("DIP Switch Status %x\r\n", dip_check);

      sleep(1);
    }

}
```

Figure 21. Snippet of source code

6Test in Hardware Step 6

6-1. Connect the board with a micro-usb cable(s) and power it ON. Establish the serial communication using SDK's Terminal tab.

6-1-1. Make sure that micro-USB cable(s) is(are) connected between the board and the PC. Turn ON the power

6-1-2. Select the Terminal tab. If it is not visible then select **Window > Show view > Terminal**

6-1-3. Click on and if required, select appropriate COM port (depends on your computer), and configure it with the parameters as shown. (These settings may have been saved from previous lab)

6-2. Program the FPGA by selecting Xilinx Tools > Program FPGA and assigning system.bit file. Run the TestApp application and verify the functionality

6-2-1. Select **Xilinx Tools** > **Program FPGA**

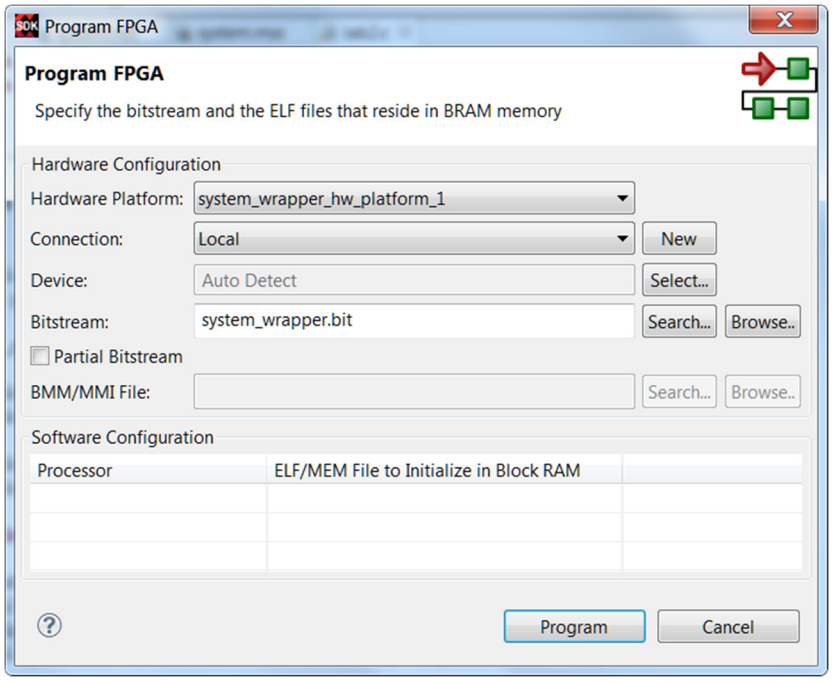


Figure 22. Program FPGA

6-2-2. Click **Program** to download the hardware bitstream. When FPGA is being programmed, the DONE LED (green color) will be off, and will turn on again when the FPGA is programmed

6-2-3. Select **TestApp** in *Project Explorer*, right-click and select **Run As > Launch on Hardware (GDB)** to download the application, execute ps7_init, and execute TestApp.elf

6-2-4. You should see the something similar to the following output on Terminal console

```
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
DIP Switch Status 6
Push Buttons Status 8
```

Figure 23. SDK Terminal output

6-2-5. Select *Console* tab and click on the *Terminate* button ( ) to stop the program

6-2-6. Close SDK and Vivado programs by selecting **File > Exit** in each program

6-2-7. Power OFF the board

Conclusion

GPIO peripherals were added from the IP catalog and connected to the Processing System through the 32b Master GP0 interface. The peripherals were configured and external FPGA connections were established. A TestApp application project was created and the functionality was verified after downloading the bitstream and executing the program.