[Microprocessor Applications]
Lab 1: Board Test

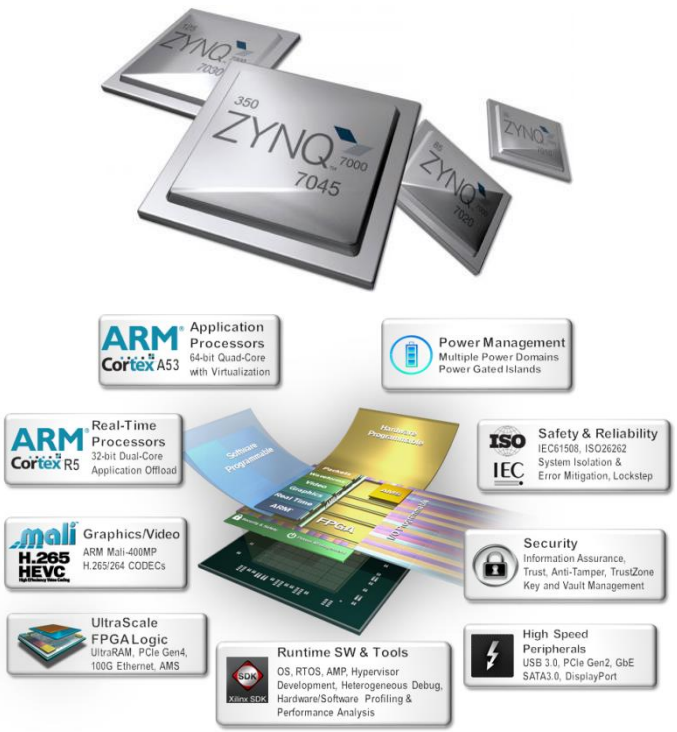
Chester Sungchung Park
SoC Design Lab, Konkuk University
Webpage: http://soclab.konkuk.ac.kr

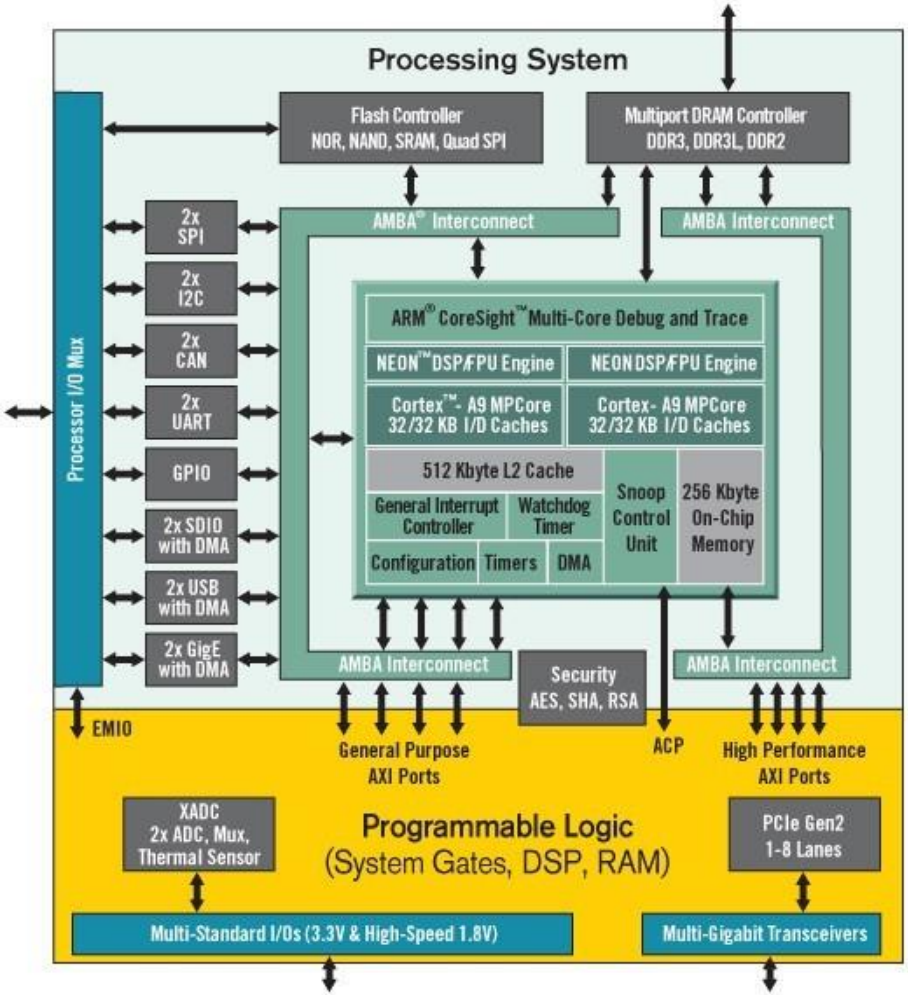
Outline

- Target system
  - Xilinx ZYNQ
  - Avnet ZedBoard
- Lab1: board test

Target Microcontroller

- Xilinx ZYNQ





Xilinx ZYNQ

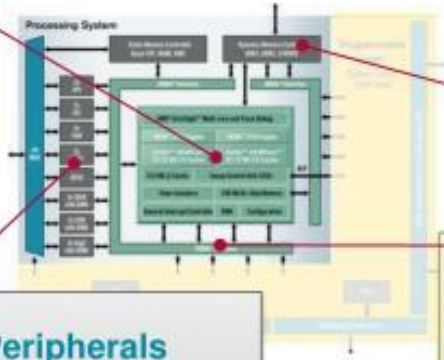
- System overview
  - Complete ARM-based processing system (PS)
    - ✓ Application processor unit
      - Dual ARM Cortex-A9 processors
      - Caches and support blocks
    - ✓ Fully integrated memory controllers
    - ✓ I/O peripherals
  - Tightly integrated programmable logic (PL)
    - ✓ Hardware acceleration
  - Flexible array of I/O
    - ✓ Wide range of external multi-standard I/O
    - ✓ High-performance integrated serial transceiver
    - ✓ Analog-to-digital converter inputs

Xilinx ZYNQ

- Processing System (PS)

Processor Core Complex
- Dual ARM Cortex-A9 MPCore with NEON™ extensions
- Single / Double Precision Floating Point support
- Up to 1 GHz operation

High BW Memory
- Internal
  - L1 Cache – 32KB/32KB (per Core)
  - L2 Cache – 512KB Unified
- On-Chip Memory of 256KB
- Integrated Memory Controllers (DDR3, DDR2, LPDDR2, 2xQSPI, NOR, NAND Flash)



Integrated Memory Mapped Peripherals
- 2x USB 2.0 (OTG) w/DMA
- 2x Tri-mode Gigabit Ethernet w/DMA
- 2x SD/SDIO w/DMA
- 2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 32b GPIO

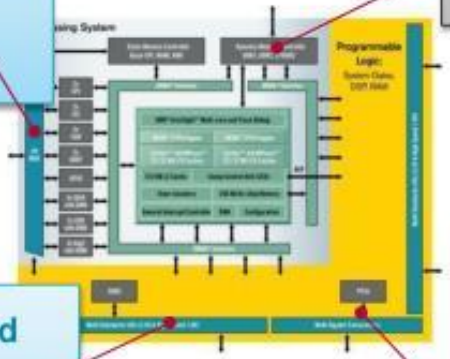
AMBA Open Standard Interconnect
- High bandwidth interconnect between Processing System and Programmable Logic
- ACP port for enhanced hardware acceleration and cache coherency for additional soft processors

Xilinx ZYNQ

- External I/Os

54 Dedicated Peripheral I/Os
- Supports integrated peripherals
- Static memory (NAND, NOR, QSPI)
- More I/Os available though the Programmable Logic

73 Dedicated Memory I/Os
- DDR3 / DDR2 / LPDDR2 Memory Interfaces
- Configurable as 16bit or 32bit



Over 350 Multi-Standard and High Performance I/O
- Up to 212 3.3V capable multi-standard I/O
- Up to 150 high performance I/O
- Up to differential 17 ADC inputs

High Performance Integrated Serial Tranceivers
(Two largest devices only)
- Up to 16 transceivers
- Operates up to 12.5Gbs
- Supports popular protocols
- Integrated PCIe Gen2 block

Xilinx ZYNQ

❑ Specifications

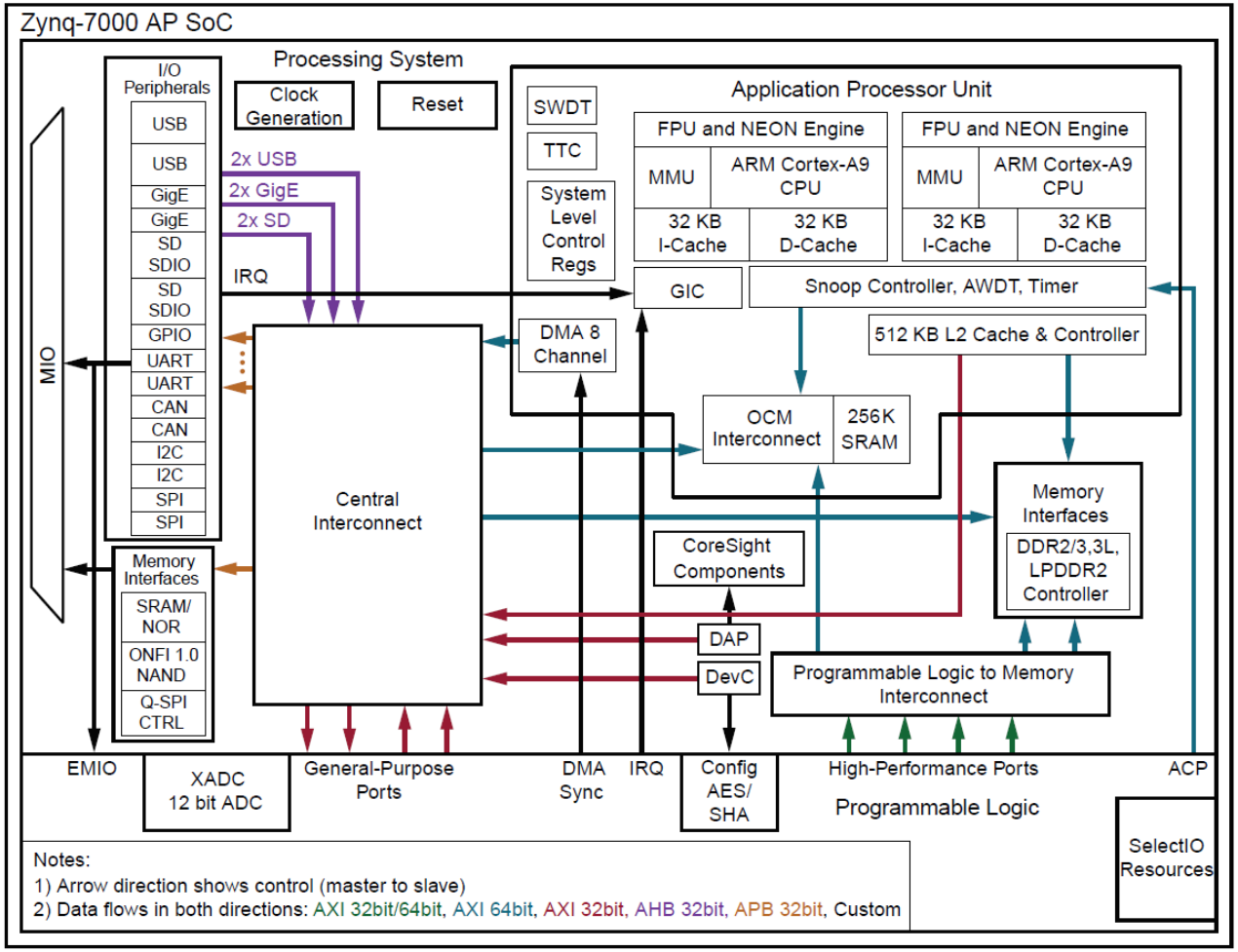
		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades: Commercial	-1			-1			-1			-1
	Speed Grades: Extended	-2			-2,-3			-2,-3			-2
	Speed Grades: Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

Xilinx ZYNQ

- Applications
  - Automotive driver assistance, driver information and infotainment
  - Broadcast camera
  - Industrial motor control, industrial networking and machine vision
  - IP and smart camera
  - LTE radio and baseband
  - Medical diagnostics and imaging
  - Multifunction printers
  - Video and night vision equipment

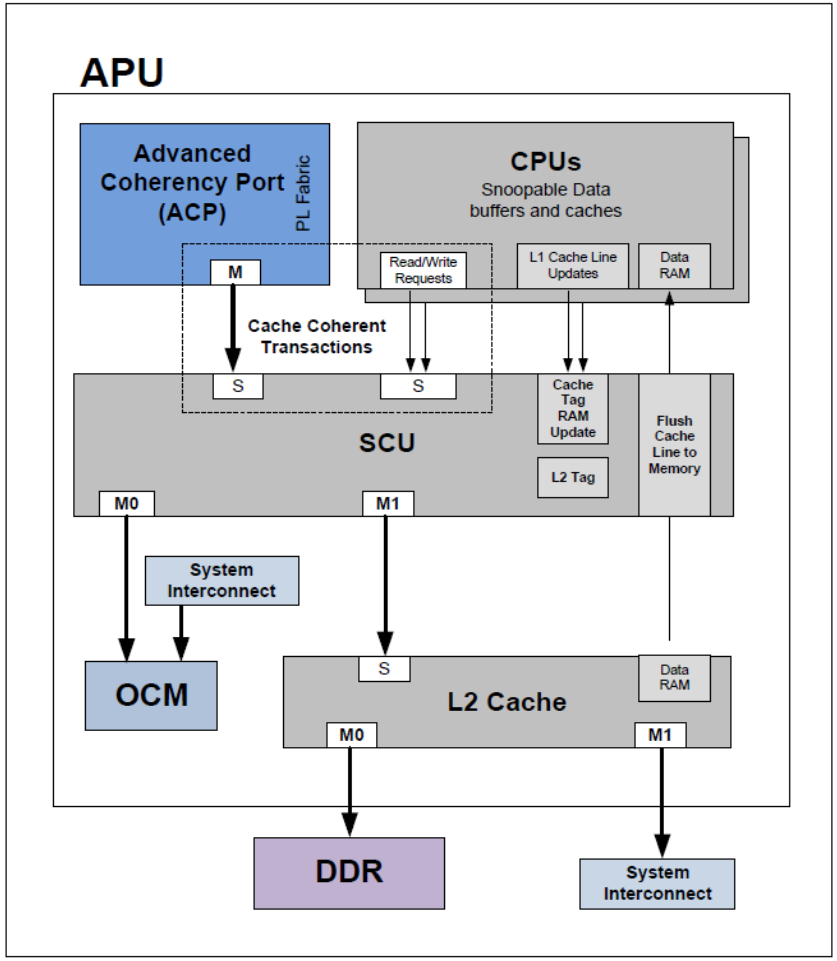
Xilinx ZYNQ

- Block diagram



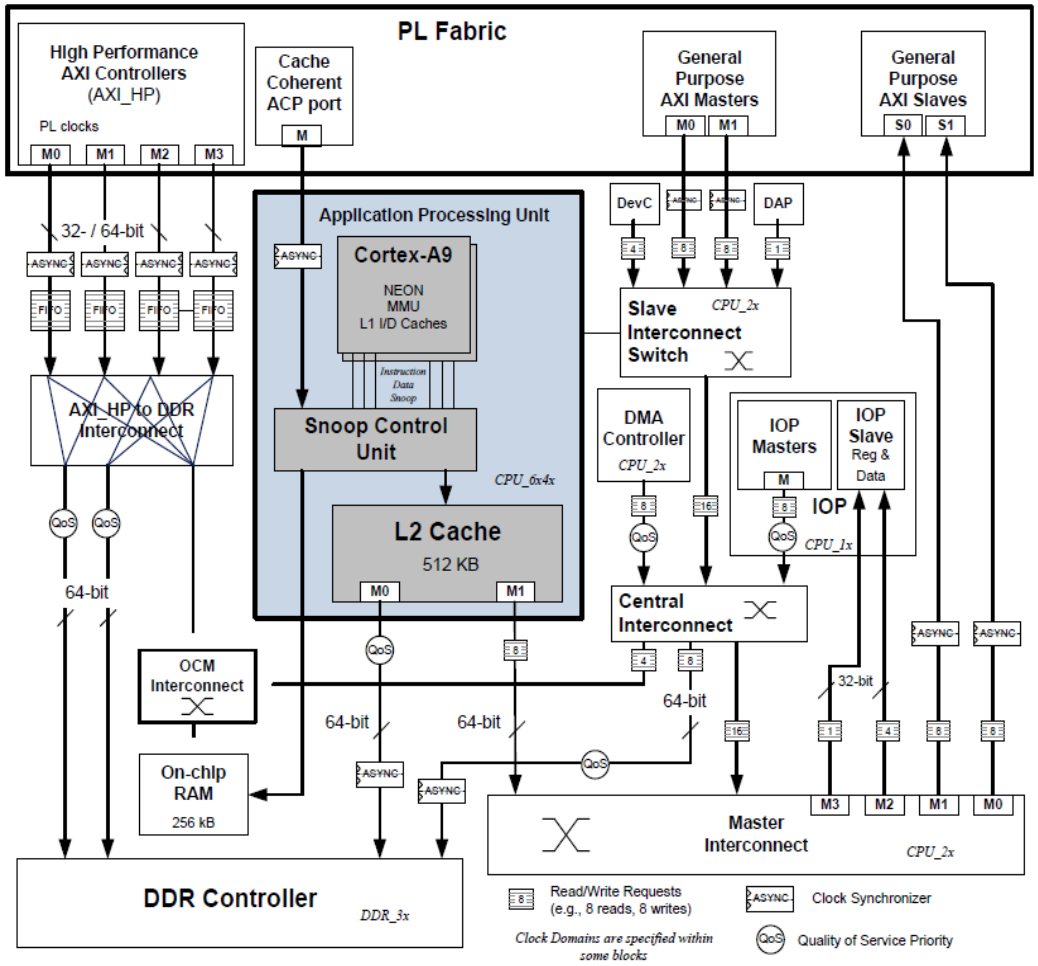
Xilinx ZYNQ

- Application Processor Unit (APU)



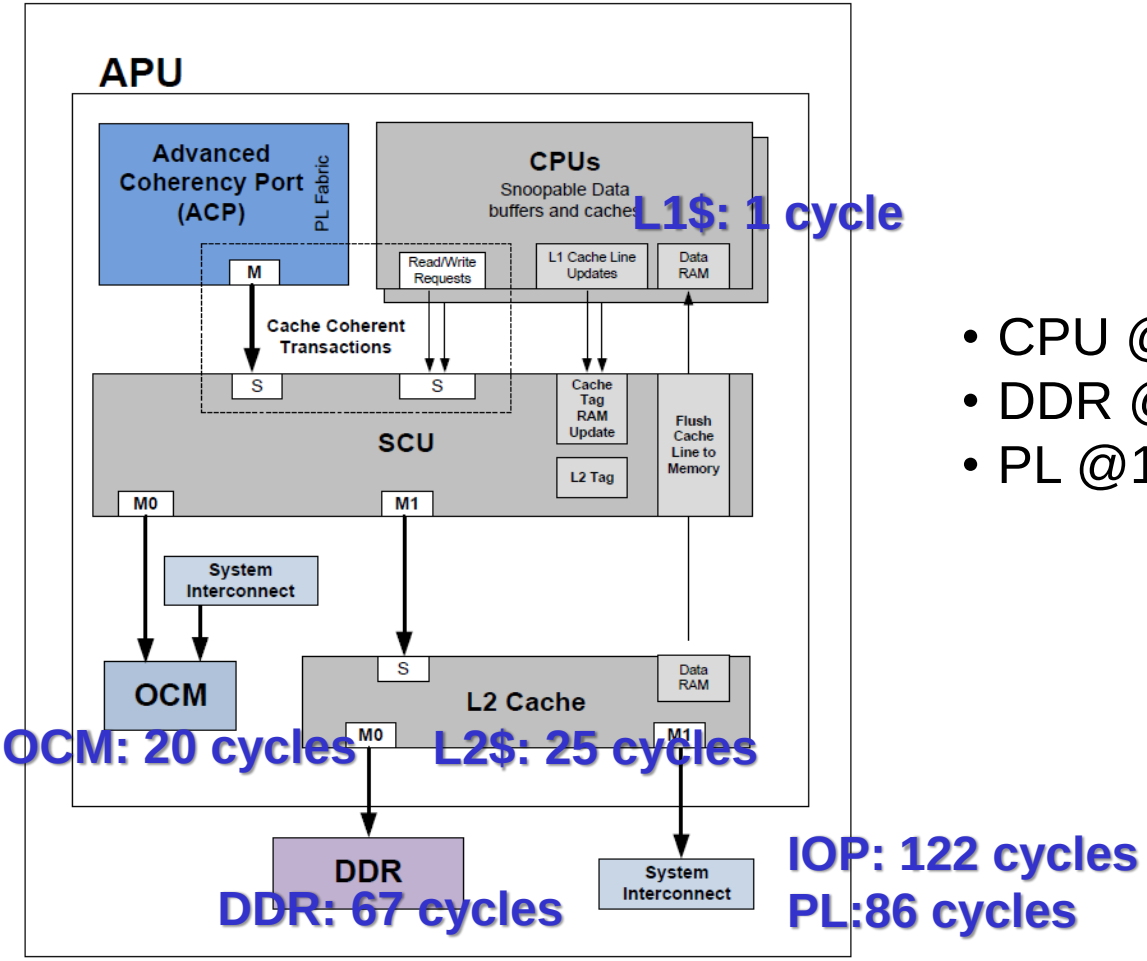
Xilinx ZYNQ

- Application Processor Unit (APU) (cont'd)



Xilinx ZYNQ

- Access latency (CPU cycle)



- CPU @ 667 MHz
- DDR @ 533 MHz
- PL @150 MHz

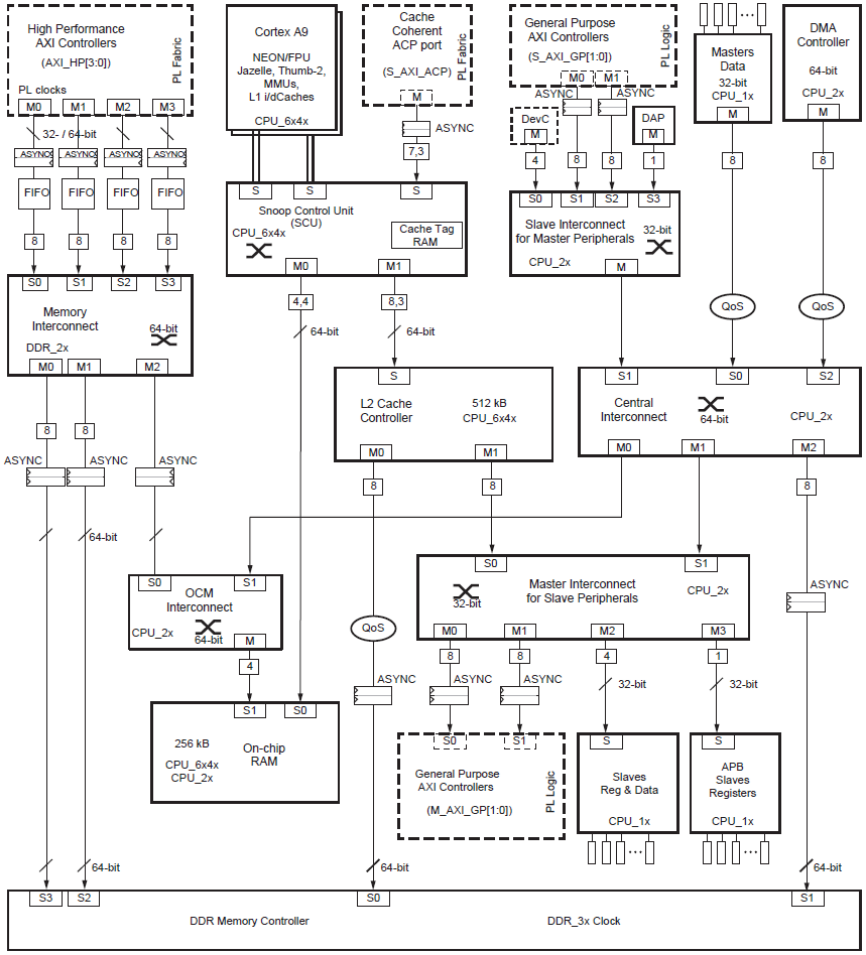
Xilinx ZYNQ

❑ Address range

Address Range	CPUs and ACP	AXI_HP	Other Bus Masters[1]	Notes
0000_0000 to 0003_FFFF[2]	OCM	OCM	OCM	Address not filtered by SCU and OCM is mapped low
	DDR	OCM	OCM	Address filtered by SCU and OCM is mapped low
	DDR			Address filtered by SCU and OCM is not mapped low
				Address not filtered by SCU and OCM is not mapped low
0004_0000 to 0007_FFFF	DDR			Address filtered by SCU
				Address not filtered by SCU
0008_0000 to 000F_FFFF	DDR	DDR	DDR	Address filtered by SCU
		DDR	DDR	Address not filtered by SCU[3]
0010_0000 to 3FFF_FFFF	DDR	DDR	DDR	Accessible to all interconnect masters
4000_0000 to 7FFF_FFFF	PL		PL	General Purpose Port #0 to the PL, M_AXI_GP0
8000_0000 to BFFF_FFFF	PL		PL	General Purpose Port #1 to the PL, M_AXI_GP1
E000_0000 to E02F_FFFF	IOP		IOP	I/O Peripheral registers, see Table 4-6
E100_0000 to E5FF_FFFF	SMC		SMC	SMC Memories, see Table 4-5
F800_0000 to F800_0BFF	SLCR		SLCR	SLCR registers, see Table 4-3
F800_1000 to F880_FFFF	PS		PS	PS System registers, see Table 4-7
F890_0000 to F8F0_2FFF	CPU			CPU Private registers, see Table 4-4
FC00_0000 to FDFF_FFFF[4]	Quad-SPI		Quad-SPI	Quad-SPI linear address for linear mode
FFFC_0000 to FFFF_FFFF[2]	OCM	OCM	OCM	OCM is mapped high
				OCM is not mapped high

Xilinx ZYNQ

- Interconnect



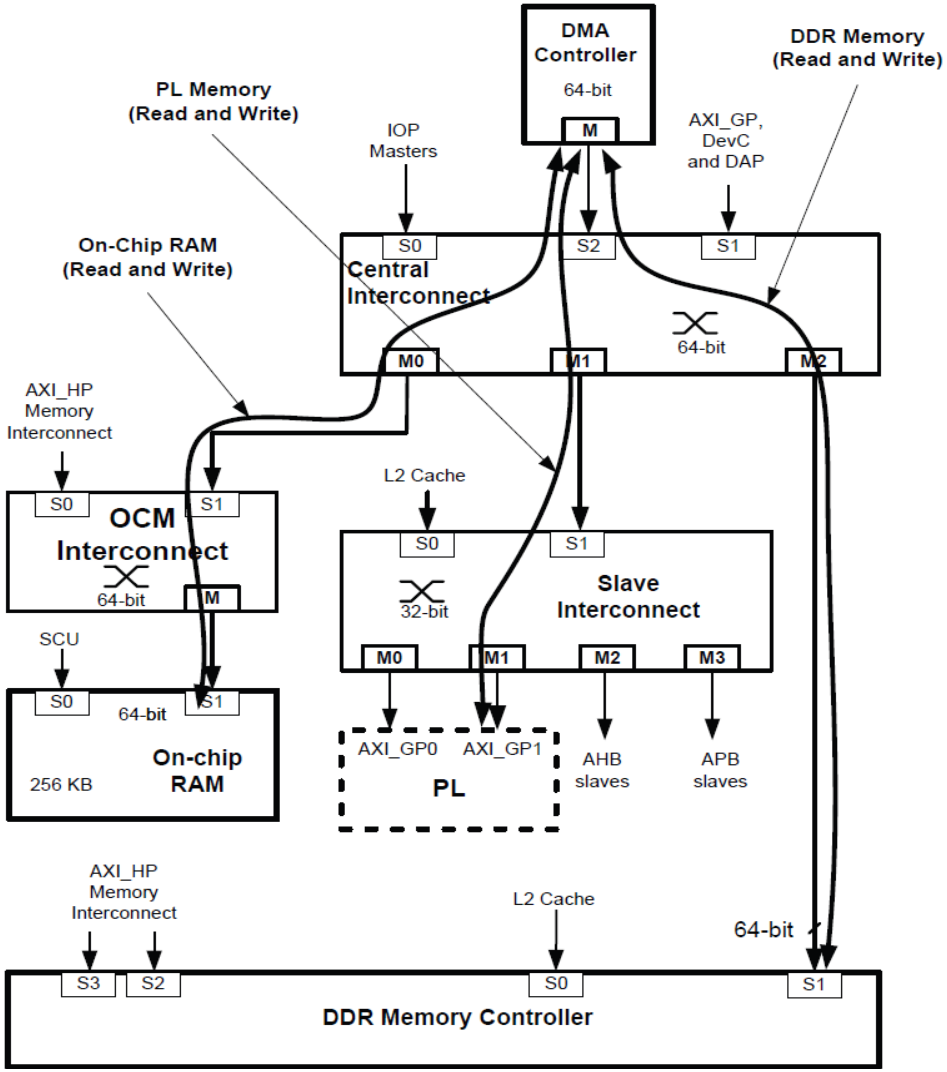
Xilinx ZYNQ

- Interconnect

Master \ Slave	On-chip RAM	DDR Port 0	DDR Port 1	DDR Port 2	DDR Port 3	M_AXI _GP	AHB Slaves	APB Slaves	GPV
CPUs	X	X				X	X	X	X
AXI_ACP	X	X				X	X	X	X
AXI_HP{0,1}	X				X				
AXI_HP{2,3}	X			X					
S_AXI_GP{0,1}	X		X			X	X	X	
DMA Controller	X		X			X	X	X	
AHB Masters	X		X			X	X	X	
DevC, DAP	X		X			X	X	X	

Xilinx ZYNQ

- DMA controller



Xilinx ZYNQ

- Development tools
  - **HW: Vivado Design Suites**
    - ✓ PS configuration wizard
    - ✓ IP integrator
    - ✓ RTL synthesis, implementation, generate bit stream
  - **SW: Eclipse IDE-based Software Development Kit (SDK)**
    - ✓ Project workspace
    - ✓ C/C++ compiler for the ARM Cortex-A9 processor
    - ✓ Debugger for the ARM Cortex-A9 processor
    - ✓ Instruction Set Simulation

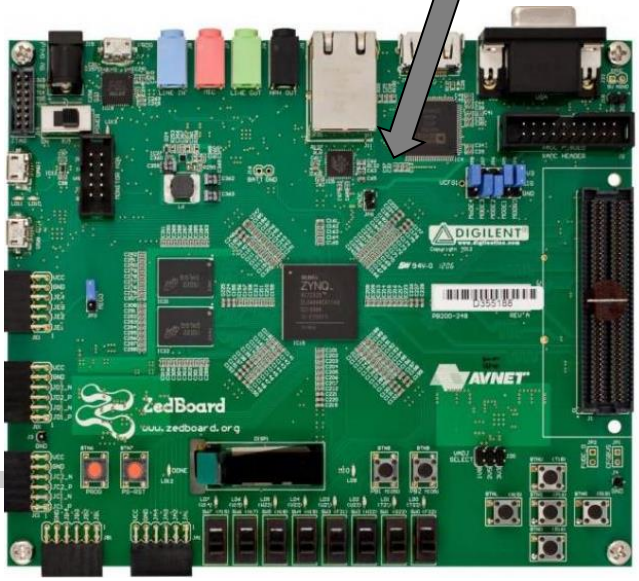
Xilinx ZYNQ

- Design flow

Vivado

SDK

ZYNQ/ZedBoard



Xilinx ZYNQ

❑ Design flow (cont'd)

Vivado

1. Launch Vivado
2. Create IP block [IP integrator]
3. Configuration PS settings
4. Add IP
5. Add Top-Level HDL
6. Add Constraints file
7. Add Generate Bitstream
8. Export hardware to SDK

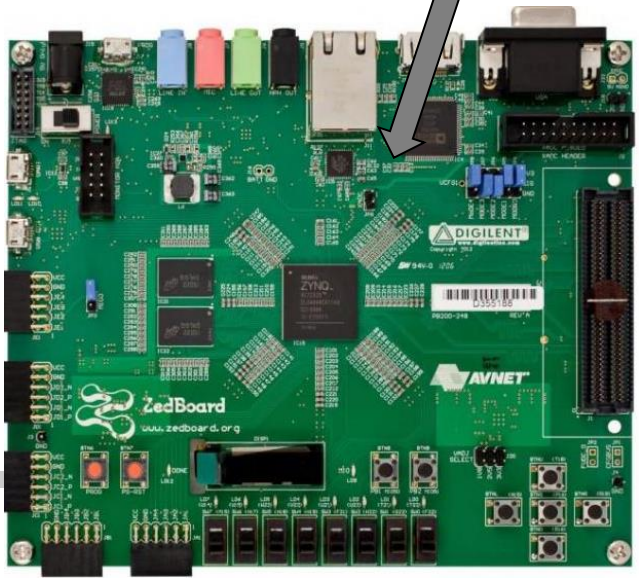
SDK

9. Specify hardware built from Vivado
10. Add software project & build
11. Program bitstream

Xilinx SDK

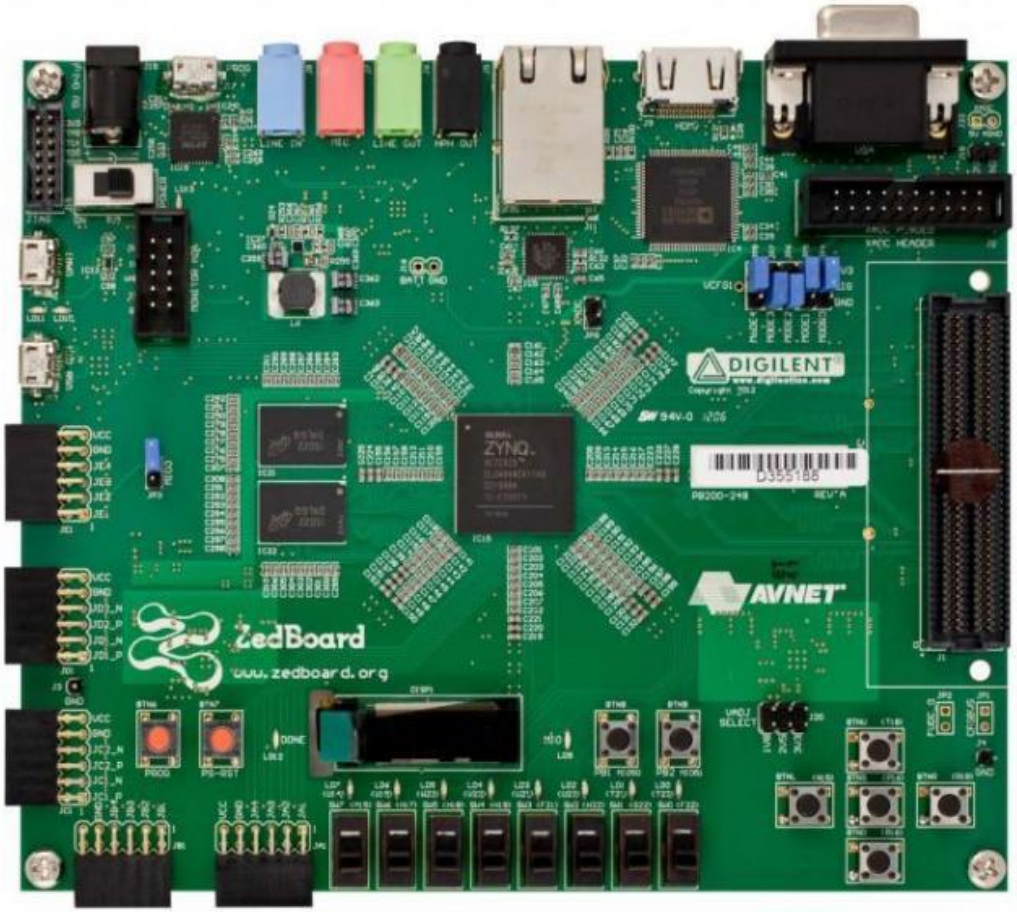
This board test will run only the software part (Step 10)

ZYNQ/ZedBoard



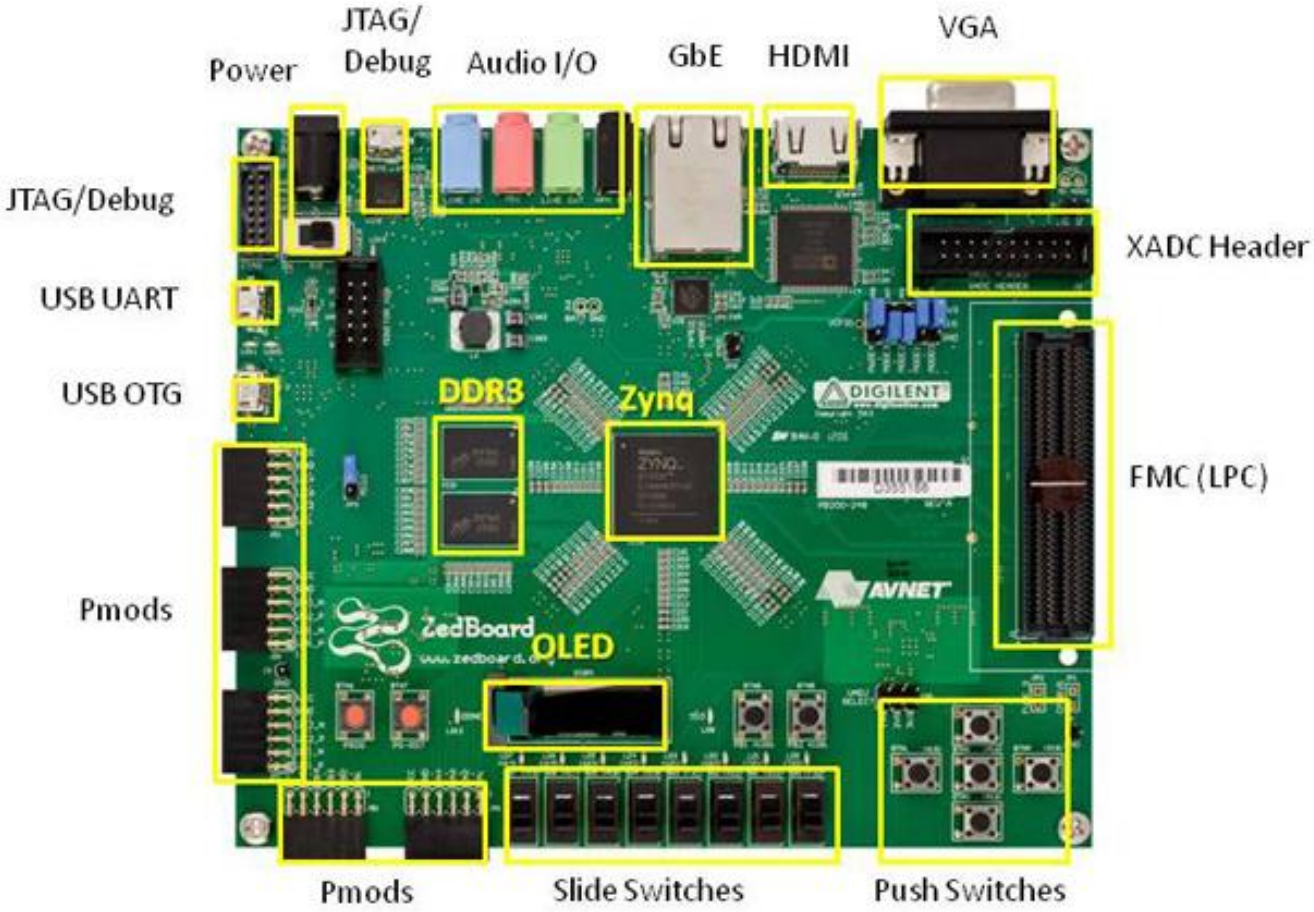
Target Board

- Avnet ZedBoard



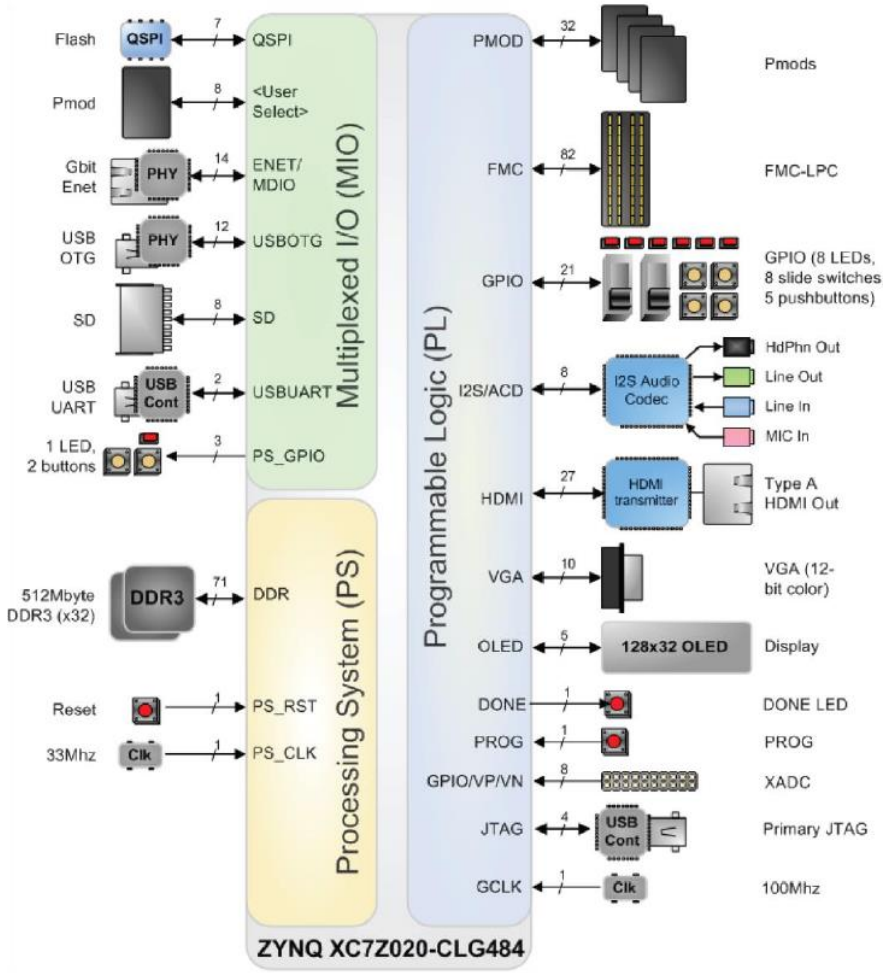
ZedBoard

- Block diagram



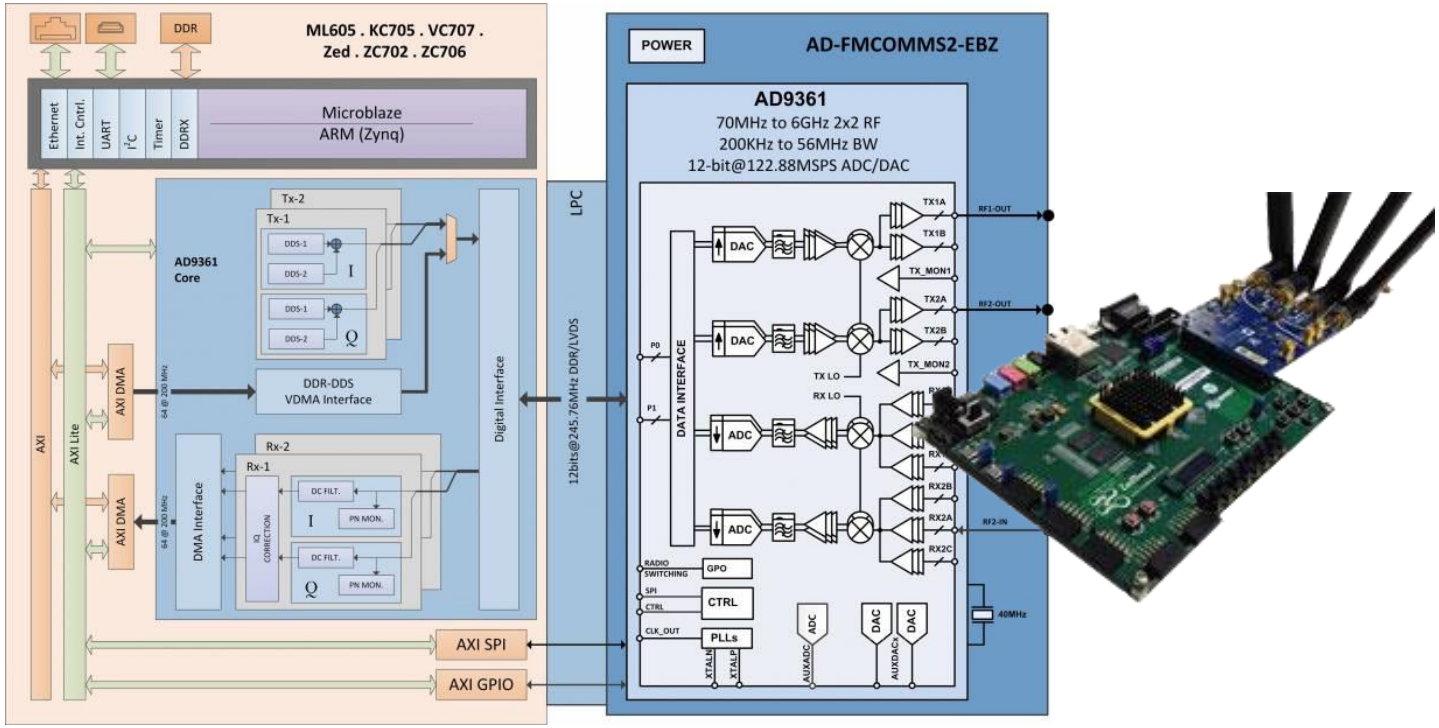
ZedBoard

- Block diagram (cont'd)



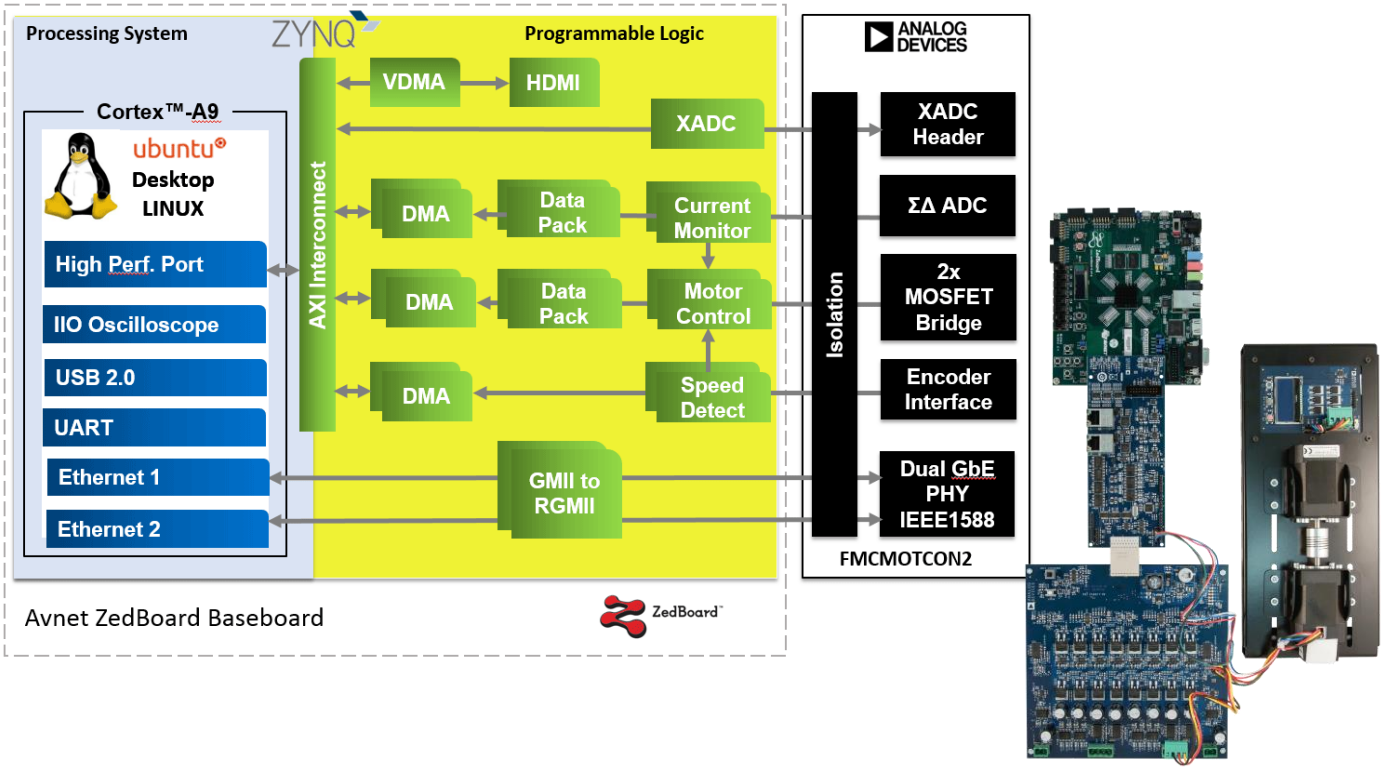
ZedBoard

- Applications
  - SDR II Evaluation Kit



ZedBoard

- Applications (cont'd)
  - Intelligent Drives Kit



ZedBoard

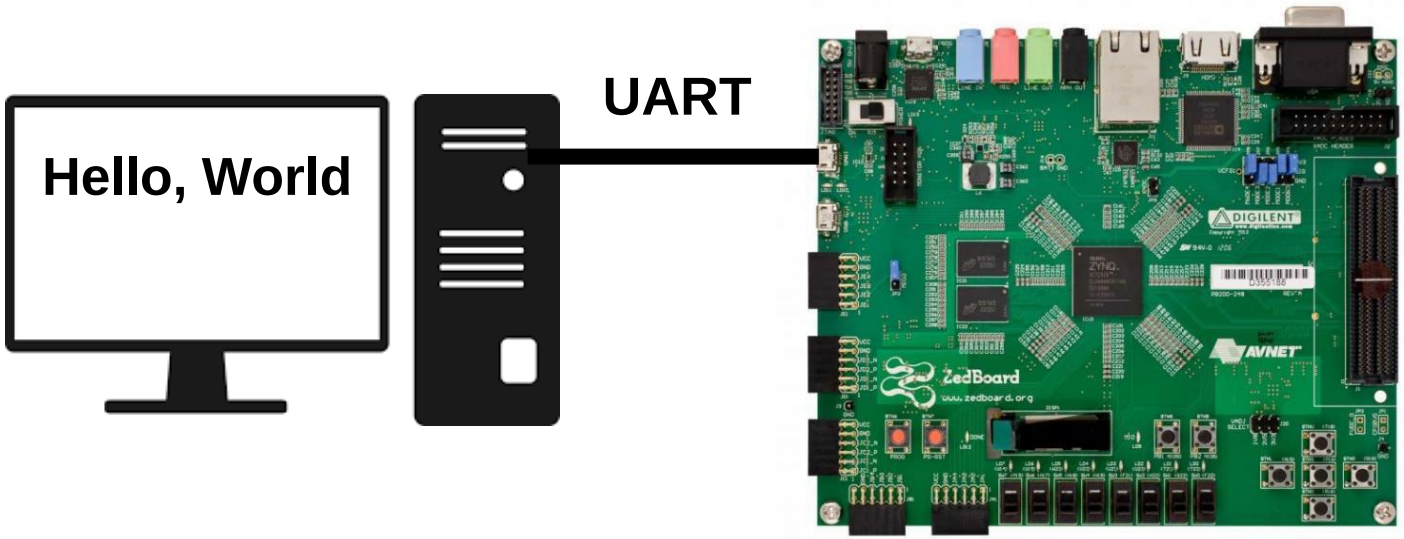
- Design examples @KU
  - WiFi baseband modem (SDR II Evaluation Kit)
    - ✓ https://www.sites.google.com/site/kusocdesignlab/demos/graduationworks2015
  - Convolutional neural network for WiFi
    - ✓ https://www.sites.google.com/site/kusocdesignlab/demos/socforcnn-basedlinkadaptation
  - Convolutional neural network for image recognition
    - ✓ https://www.sites.google.com/site/kusocdesignlab/demos/cnnacceleratoronzynq

Lab 1: Board Test

- Objectives
  - Creating a project using SDK
  - Running a C application using SDK
  - Communicating with the board using UART

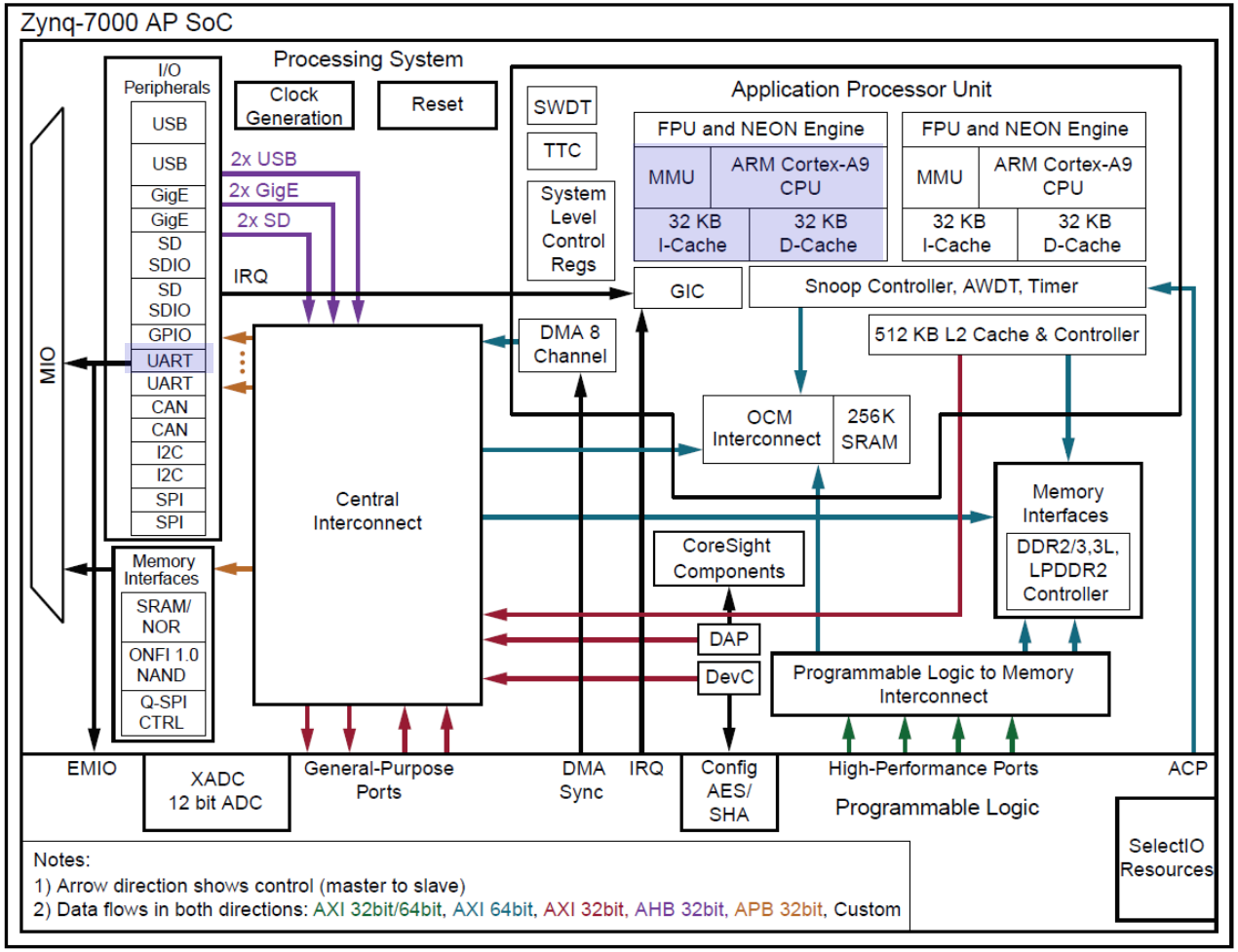
Lab 1: Board Test

- Design description
  - Showing how to create a simple software design



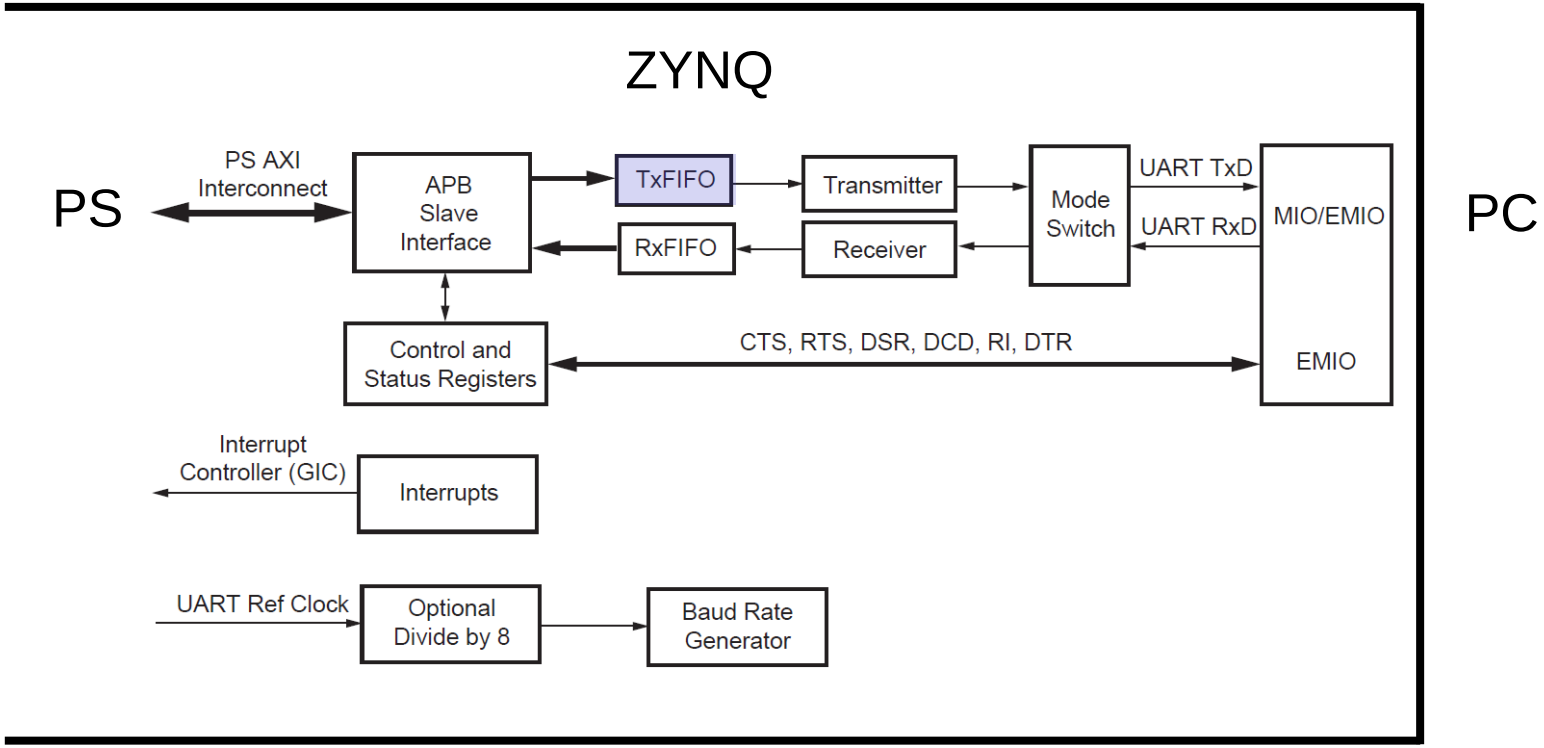
Lab 1: Board Test

- Block diagram: ZYNQ



Lab 1: Board Test

- Block diagram: UART



Lab 1: Board Test

- Address map

system.xml | *helloworld.c | platform.c | print.c | outbyte.c | xuartps_hw.c

MicroZed_hw_platform Hardware Platform Specification

Design Information

Target FPGA Device: 7z010
Created With: Vivado 2013.4
Created On: Mon Nov 11 11:07:46 2013

Address Map for processor ps7_cortexa9_0

ps7_afi_0	0xf8008000	0xf8008fff
ps7_afi_1	0xf8009000	0xf8009fff
ps7_afi_2	0xf800a000	0xf800afff
ps7_afi_3	0xf800b000	0xf800bfff
ps7_coresight_comp_0	0xf8800000	0xf88fffff
ps7_ddr_0	0x00100000	0x3fffffff
ps7_ddrc_0	0xf8006000	0xf8006fff
ps7_dev_cfg_0	0xf8007000	0xf80070ff
ps7_dma_ns	0xf8004000	0xf8004fff
ps7_dma_s	0xf8003000	0xf8003fff
ps7_ethernet_0	0xe000b000	0xe000bfff
ps7_globaltimer_0	0xf8f00200	0xf8f002ff
ps7_gpio_0	0xe000a000	0xe000afff
ps7_gpv_0	0xf8900000	0xf89fffff
ps7_intc_dist_0	0xf8f01000	0xf8f01fff
ps7_iop_bus_config_0	0xe0200000	0xe0200fff
ps7_l2cachec_0	0xf8f02000	0xf8f02fff
ps7_ocmc_0	0xf800c000	0xf800cfff
ps7_qspi_0	0xe000d000	0xe000dfff
ps7_qspi_linear_0	0xfc000000	0xfcffffff
ps7_ram_0	0x00000000	0x0002ffff
ps7_ram_1	0xffff0000	0xffffffff
ps7_scuc_0	0xf8f00000	0xf8f000fc
ps7_scugic_0	0xf8f00100	0xf8f001ff
ps7_scutimer_0	0xf8f00600	0xf8f0061f
ps7_scuwdt_0	0xf8f00620	0xf8f006ff
ps7_sd_0	0xe0100000	0xe0100fff
ps7_slcr_0	0xf8000000	0xf8000fff
ps7_ttc_0	0xf8001000	0xf8001fff
ps7_uart_1	0xe0001000	0xe0001fff
ps7_usb_0	0xe0002000	0xe0002fff
ps7_xadc_0	0xf8007100	0xf8007120

Overview | Source

Lab 1: Board Test

- Section map

*helloworld.c | *platform.c | print.c | outbyte.c | xuartps_hw.c | lscript.ld

Linker Script: lscript.ld

A linker script is used to control where different sections of an executable are placed in memory.
In this page, you can define new memory regions, and change the assignment of sectionsto memory regions.

Available Memory Regions

Name	Base Address	Size
ps7_ddr_0_S_AXI_BASEADDR	0x00100000	0x3FF00000
ps7_ram_0_S_AXI_BASEADDR	0x00000000	0x00030000
ps7_ram_1_S_AXI_BASEADDR	0xFFFF0000	0x0000FE00

Stack and Heap Sizes

Stack Size 0x2000

Heap Size 0x2000

Section to Memory Region Mapping

Section Name	Memory Region
.text	ps7_ddr_0_S_AXI_BASEADDR
.init	ps7_ddr_0_S_AXI_BASEADDR
.fini	ps7_ddr_0_S_AXI_BASEADDR
.rodata	ps7_ddr_0_S_AXI_BASEADDR
.rodata1	ps7_ddr_0_S_AXI_BASEADDR
.sdata2	ps7_ddr_0_S_AXI_BASEADDR
.sbss2	ps7_ddr_0_S_AXI_BASEADDR
.data	ps7_ddr_0_S_AXI_BASEADDR
.data1	ps7_ddr_0_S_AXI_BASEADDR
.got	ps7_ddr_0_S_AXI_BASEADDR
.ctors	ps7_ddr_0_S_AXI_BASEADDR
.dtors	ps7_ddr_0_S_AXI_BASEADDR
.fixup	ps7_ddr_0_S_AXI_BASEADDR
.eh_frame	ps7_ddr_0_S_AXI_BASEADDR
.eh_framehdr	ps7_ddr_0_S_AXI_BASEADDR
.gcc_except_table	ps7_ddr_0_S_AXI_BASEADDR
.mmu_tbl	ps7_ddr_0_S_AXI_BASEADDR
.ARM.exidx	ps7_ddr_0_S_AXI_BASEADDR
.preinit_array	ps7_ddr_0_S_AXI_BASEADDR
.init_array	ps7_ddr_0_S_AXI_BASEADDR
.fini_array	ps7_ddr_0_S_AXI_BASEADDR
.ARM.attributes	ps7_ddr_0_S_AXI_BASEADDR

Summary | Source

Lab 1: Board Test

- Source code: xparameter.h

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c | xparameters.h

```c
* CAUTION: This file is automatically generated by libgen.

#include "xparameters_ps.h"

#define STDIN_BASEADDRESS 0xE0001000
#define STDOUT_BASEADDRESS 0xE0001000

/*******************************************************************/
```

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c | xparameters.h

```c
#define XPAR_XTTCPS_2_TTC_CLK_FREQ_HZ 111111115
#define XPAR_XTTCPS_2_TTC_CLK_CLKSRC 0


/*******************************************************************/

/* Definitions for driver UARTPS */
#define XPAR_XUARTPS_NUM_INSTANCES 1

/* Definitions for peripheral PS7_UART_1 */
#define XPAR_PS7_UART_1_DEVICE_ID 0
#define XPAR_PS7_UART_1_BASEADDR 0xE0001000
#define XPAR_PS7_UART_1_HIGHADDR 0xE0001FFF
#define XPAR_PS7_UART_1_UART_CLK_FREQ_HZ 50000000
#define XPAR_PS7_UART_1_HAS_MODEM 0


/*******************************************************************/
```

Lab 1: Board Test

- Source code: helloworld.c

system.xml | *helloworld.c | platform.c | print.c | outbyte.c | xuartps_hw.c

```c
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ----------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ----------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"

void print(char *str);

int main()
{
    init_platform();

    print("Hello World\n\r");

    return 0;
}
```

Lab 1: Board Test

- Source code: platform.c

system.xml | *helloworld.c | *platform.c | print.c | outbyte.c | xuartps_hw.c

```c
void
init_uart()
{
#ifdef STDOUT_IS_16550
    XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, UART_BAUD);
    XUartNs550_SetLineControlReg(STDOUT_BASEADDR, XUN_LCR_8_DATA_BITS);
#endif
#ifdef STDOUT_IS_PS7_UART
    /* Bootrom/BSP configures PS7 UART to 115200 bps */
#endif
}

void
init_platform()
{
    /*
     * If you want to run this example outside of SDK,
     * uncomment the following line and also #include "ps7_init.h" at the top.
     * Make sure that the ps7_init.c and ps7_init.h files are included
     * along with this example source files for compilation.
     */
    /* ps7_init();*/
    enable_caches();
    init_uart();
}
```

Lab 1: Board Test

- Source code: print.c

system.xml | *helloworld.c | *platform.c | print.c | outbyte.c | xuartps_hw.c

```c
/* print.c -- print a string on the output device.

/*
 * print -- do a raw print of a string
 */
#include "xil_printf.h"

void print(const char *ptr)
{
#ifdef STDOUT_BASEADDRESS
  while (*ptr) {
    outbyte (*ptr++);
  }
#else
(void)ptr;
#endif
}
```

Lab 1: Board Test

- Source code: outbyte.c

system.xml | *helloworld.c | *platform.c | print.c | outbyte.c | xuartps_hw.c

```c
#include "xparameters.h"
#include "xuartps_hw.h"

#ifdef __cplusplus
extern "C" {
#endif
void outbyte(char c);

#ifdef __cplusplus
}
#endif

void outbyte(char c) {
    XUartPs_SendByte(STDOUT_BASEADDRESS, c);
}
```

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c | xparameters.h

```c
* CAUTION: This file is automatically generated by libgen.

#include "xparameters_ps.h"

#define STDIN_BASEADDRESS 0xE0001000
#define STDOUT_BASEADDRESS 0xE0001000

/*****************************************************************/
```

Lab 1: Board Test

- Source code: xuartps_hw.c

*helloworld.c | *platform.c | print.c | outbyte.c | xuartps_hw.c | lscript.ld

```c
* This function sends one byte using the device. This function operates in
* polled mode and blocks until the data has been put into the TX FIFO register.
*
* @param    BaseAddress contains the base address of the device.
* @param    Data contains the byte to be sent.
*
* @return   None.
*
* @note     None.
*
*****************************************************************************/
void XUartPs_SendByte(u32 BaseAddress, u8 Data)
{
        /*
         * Wait until there is space in TX FIFO
         */
        while (XUartPs_IsTransmitFull(BaseAddress));

        /*
         * Write the byte into the TX FIFO
         */
        XUartPs_WriteReg(BaseAddress, XUARTPS_FIFO_OFFSET, Data);
}

/*********************
```

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c

```c
#define XUARTPS_MODEMSR_OFFSET  0x28  /**< Modem Status [8:0] */
#define XUARTPS_SR_OFFSET   0x2C  /**< Channel Status [14:0] */
#define XUARTPS_FIFO_OFFSET 0x30  /**< FIFO [7:0] */
#define XUARTPS_BAUDDIV_OFFSET  0x34  /**< Baud Rate Divider [7:0] */
#define XUARTPS_FLOWDEL_OFFSET  0x38  /**< Flow Delay [5:0] */
#define XUARTPS_TXWM_OFFSET 0x44  /**< TX FIFO Trigger Level [5:0] */
/* @} */
```

Lab 1: Board Test

- Source code: xuartps_hw.h

outbyte.c | xuartps_hw.c | lscript.ld | *xuartps_hw.h | xil_io.c | xuartps_hw.h

```
/**
* Determine if a byte of data can be sent with the transmitter.
*
* @param    BaseAddress contains the base address of the device.
*
* @return   TRUE if the TX FIFO is full, FALSE if a byte can be put in the
*       FIFO.
*
* @note     C-Style signature:
*       u32 XUartPs_IsTransmitFull(u32 BaseAddress)
*
*****************************************************************************/
#define XUartPs_IsTransmitFull(BaseAddress)          \
    ((Xil_In32((BaseAddress) + XUARTPS_SR_OFFSET) &    \
     XUARTPS_SR_TXFULL) == XUARTPS_SR_TXFULL)
```

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c | xuartps_hw.h

```
#define XUARTPS_SR_FRAME     0x00000040 /**< RX frame error */
#define XUARTPS_SR_OVER      0x00000020 /**< RX overflow error */
#define XUARTPS_SR_TXFULL    0x00000010 /**< TX FIFO full */
#define XUARTPS_SR_TXEMPTY   0x00000008 /**< TX FIFO empty */
#define XUARTPS_SR_RXFUL
#define XUARTPS_SR_RXEMP
#define XUARTPS_SR_RXOVR
/* @} */
```

outbyte.c | xuartps_hw.c | xuartps_hw.h | xil_io.c

```
#define XUARTPS_MODEMSR_OFFSET  0x28  /**< Modem Status [8:0] */
#define XUARTPS_SR_OFFSET   0x2C  /**< Channel Status [14:0] */
#define XUARTPS_FIFO_OFFSET 0x30  /**< FIFO [7:0] */
#define XUARTPS_BAUDDIV_OFFSET  0x34  /**< Baud Rate Divider [7:0] */
#define XUARTPS_FLOWDEL_OFFSET  0x38  /**< Flow Delay [5:0] */
#define XUARTPS_TXWM_OFFSET 0x44  /**< TX FIFO Trigger Level [5:0] */
/* @} */
```

Lab 1: Board Test

- Source code: xuartps_hw.h (cont'd)

outbyte.c | xuartps_hw.c | lscript.ld | *xuartps_hw.h | xil_io.c | xuartps_hw.h

```c
/*****************************************************************************/
/**
* Write a UART register.
*
* @param    BaseAddress contains the base address of the device.
* @param    RegOffset contains the offset from the base address of the
*       device.
* @param    RegisterValue is the value to be written to the register.
*
* @return   None.
*
* @note     C-Style signature:
*       void XUartPs_WriteReg(u32 BaseAddress, int RegOffset,
*                             u16 RegisterValue)
*
*****************************************************************************/
#define XUartPs_WriteReg(BaseAddress, RegOffset, RegisterValue) \
    Xil_Out32((BaseAddress) + (RegOffset), (RegisterValue))
```

Lab 1: Board Test

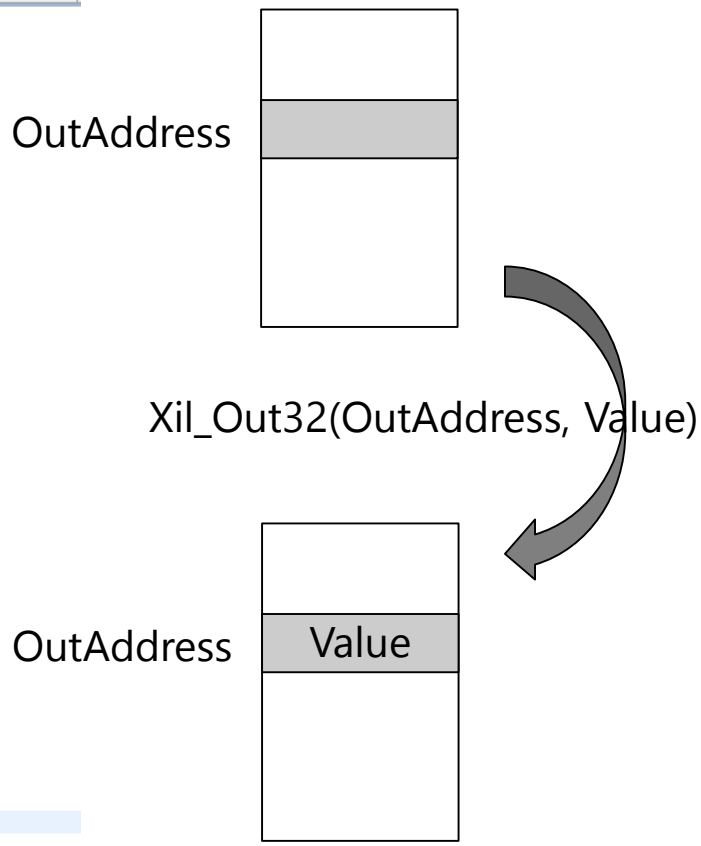
- Source code: xil_io.c

outbyte.c | xuartps_hw.c | lscript.ld | *xuartps_hw.h | xil_io.c | xuartps_hw.h

```c
/*****************************************************************************/
/**
*
* Performs an output operation for a 16-bit memory location by writing the
* specified Value to the the specified address.
*
* @param    OutAddress contains the address to perform the output operation
*           at.
* @param    Value contains the Value to be output at the specified address.
*
* @return   None.
*
* @note     None.
*
******************************************************************************/
void Xil_Out16(u32 OutAddress, u16 Value)
{
    *(volatile u16 *) OutAddress = Value;
}

/*****************************************************************************/
/**
*
* Performs an output operation for a 32-bit memory location by writing the
* specified Value to the the specified address.
*
* @param    OutAddress contains the address to perform the output operation
*           at.
* @param    Value contains the Value to be output at the specified address.
*
* @return   None.
*
* @note     None.
*
******************************************************************************/
void Xil_Out32(u32 OutAddress, u32 Value)
{
    *(volatile u32 *) OutAddress = Value;
}

/*****************************************************************************/
```



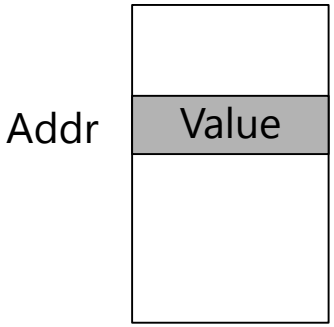
Lab 1: Board Test

- Source code: xil_io.c (cont'd)

outbyte.c | xuartps_hw.c | lscript.ld | *xuartps_hw.h | xil_io.c | xuartps_hw.h

```c
/*****************************************************************************/
/**
*
* Performs an input operation for a 16-bit memory location by reading from the
* specified address and returning the Value read from that address.
*
* @param    Addr contains the address to perform the input operation
*       at.
*
* @return   The Value read from the specified input address.
*
* @note     None.
*
******************************************************************************/
u16 Xil_In16(u32 Addr)
{
    return *(volatile u16 *) Addr;
}

/*****************************************************************************/
/**
*
* Performs an input operation for a 32-bit memory location by reading from the
* specified address and returning the Value read from that address.
*
* @param    Addr contains the address to perform the input operation
*       at.
*
* @return   The Value read from the specified input address.
*
* @note     None.
*
******************************************************************************/
u32 Xil_In32(u32 Addr)
{
    return *(volatile u32 *) Addr;
}
```



Xil_In32(Addr) returns Value

App: Registers for UART

B.33 UART Controller (UART)

Module Name	UART Controller (UART)
Software Name	XUARTPS
Base Address	0xE0000000 uart0 0xE0001000 uart1
Description	Universal Asynchronous Receiver Transmitter
Vendor Info	Cadence UART

Register Summary

Register Name	Address	Width	Type	Reset Value	Description
Control_reg0	0x00000000	32	mixed	0x00000128	UART Control Register
mode_reg0	0x00000004	32	mixed	0x00000000	UART Mode Register
Intrpt_en_reg0	0x00000008	32	mixed	0x00000000	Interrupt Enable Register
Intrpt_dis_reg0	0x0000000C	32	mixed	0x00000000	Interrupt Disable Register
Intrpt_mask_reg0	0x00000010	32	ro	0x00000000	Interrupt Mask Register
Chnl_int_sts_reg0	0x00000014	32	wtc	0x00000000	Channel Interrupt Status Register
Baud_rate_gen_reg0	0x00000018	32	mixed	0x0000028B	Baud Rate Generator Register.
Rcvr_timeout_reg0	0x0000001C	32	mixed	0x00000000	Receiver Timeout Register
Rcvr_FIFO_trigger_level0	0x00000020	32	mixed	0x00000020	Receiver FIFO Trigger Level Register
Modem_ctrl_reg0	0x00000024	32	mixed	0x00000000	Modem Control Register
Modem_sts_reg0	0x00000028	32	mixed	x	Modem Status Register
Channel_sts_reg0	0x0000002C	32	ro	0x00000000	Channel Status Register
TX_RX_FIFO0	0x00000030	32	mixed	0x00000000	Transmit and Receive FIFO
Baud_rate_divider_reg0	0x00000034	32	mixed	0x0000000F	Baud Rate Divider Register
Flow_delay_reg0	0x00000038	32	mixed	0x00000000	Flow Control Delay Register
Tx_FIFO_trigger_level0	0x00000044	32	mixed	0x00000020	Transmitter FIFO Trigger Level Register

App: Registers for UART

- Tx/Rx FIFO register

Field Name	Bits	Type	Reset Value	Description
reserved	31:8	ro	0x0	Reserved, read as zero, ignored on write.
FIFO	7:0	rw	0x0	Operates as Tx FIFO and Rx FIFO.

- Channel status register

reserved	6	ro	0x0	Reserved. Do not modify.
reserved	5	ro	0x0	Reserved. Do not modify.
TFUL (TXFULL)	4	ro	0x0	Transmitter FIFO Full continuous status: 0: Tx FIFO is not full 1: Tx FIFO is full
TEMPTY (TXEMPTY)	3	ro	0x0	Transmitter FIFO Empty continuous status: 0: Tx FIFO is not empty 1: Tx FIFO is empty

References

- Zynq-7000 All Programmable, technical reference manual, Xilinx UG585