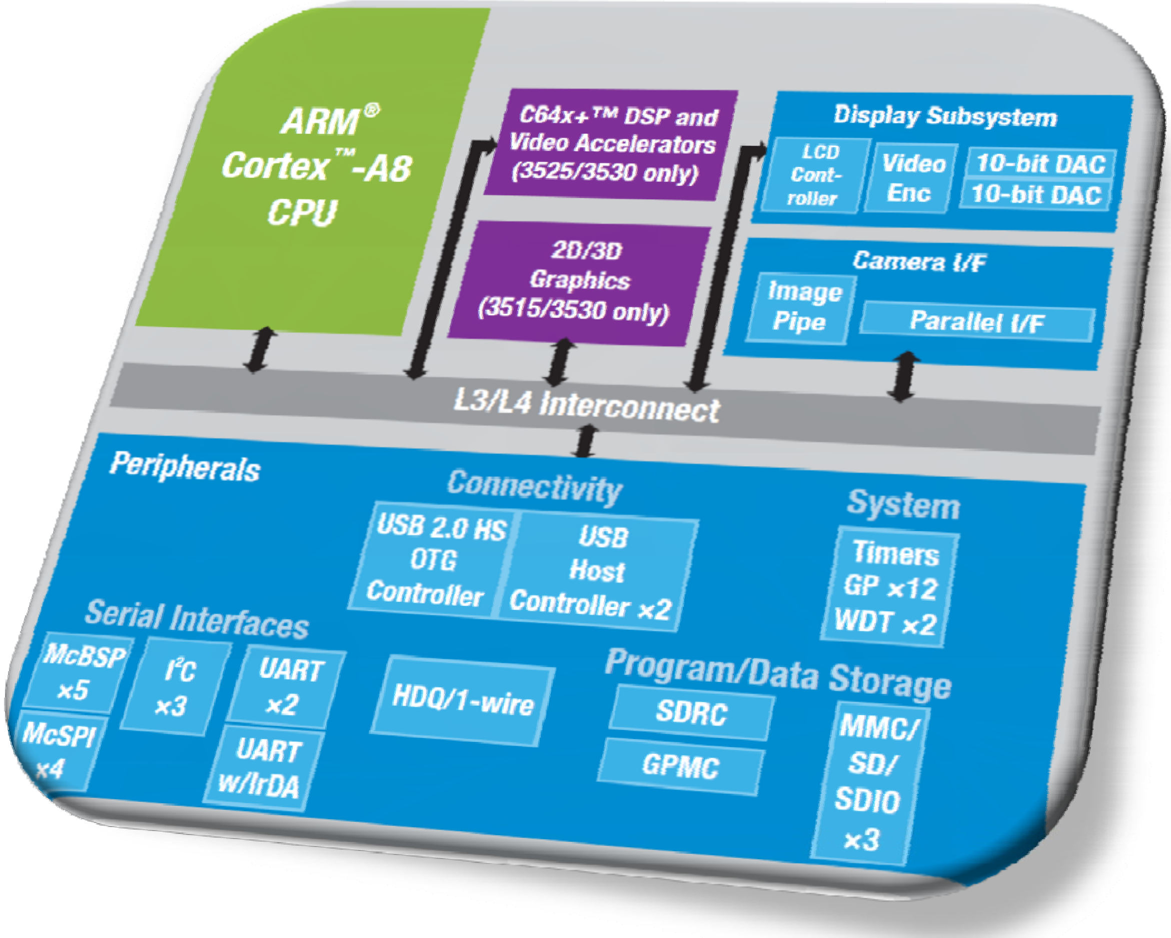
INF3430/4430: EDK

- Forelesningen gir en innføring i bruk av EDK for å konstruere et System-on-Chip (SoC).
- Oppsett av hardware til et System-On-Chip-design
- Legge til egen IP-kjerne
- Software-design
- Debugging og simulering
- Relatert til tidligere års laboppgave 4
- Forelesningsnotatene er for det meste satt sammen av materiale fra Xilinx University Program (http://www.xilinx.com/univ/)

Hva er System-on-Chip?



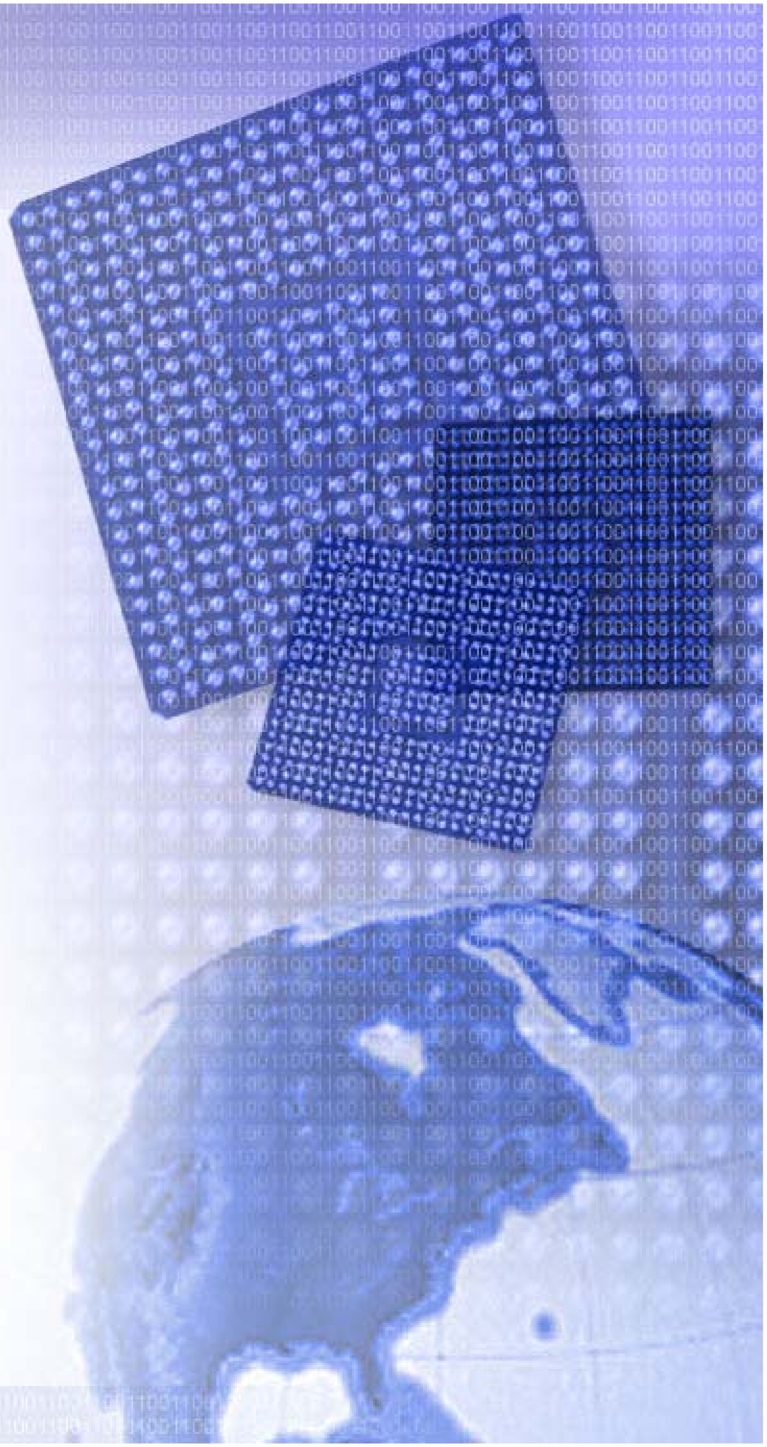
TI OMAP35xx

- Integrering av mange (tidligere separate) komponenter på en brikke

Fordeler ved SoC

- Sparer plass – perfekt for mobile og ”innbakte” (embedded) systemer
  - Embedded: spesialisert datamaskin innbakt i en enhet (i motsetning til PC)
- Enklere kretskortdesign og montasje
- Man kan plukke komponenter som ferdigtestede IP(intellectual property)-kjerner fra leverandører
  - Disse er gjerne i VHDL eller lignende, så de kan ofte skreddersys til eget design
  - Open Source-kjerner: www.opencores.org
- Kan gi mulighet til økt ytelse pga. tettere kommunikasjon mellom komponenter

Embedded Design with EDK



EDK Tools

- EDK = Embedded Development Kit
- XPS = Xilinx Platform Studio
- PlatGen = Platform Generator
  - Uses an MHS file to create an implementation netlist of a bus-based subsystem
- LibGen = Library Generator
  - Uses the MHS and MSS files, software libraries, and source files to generate an executable image
- SimGen = Simulation Generator
  - Uses the MHS file to generate a simulation environment including simulation models, HDL wrappers, simulation scripts, etc.
- XMD = Xilinx Microprocessor Debugger
  - Provides communication between the GDB and the processor
- CreateIP = Create/Import Peripheral Wizard
  - Helps you create your own peripherals and import them into EDK compliant repositories or Xilinx Platform Studio (XPS) projects

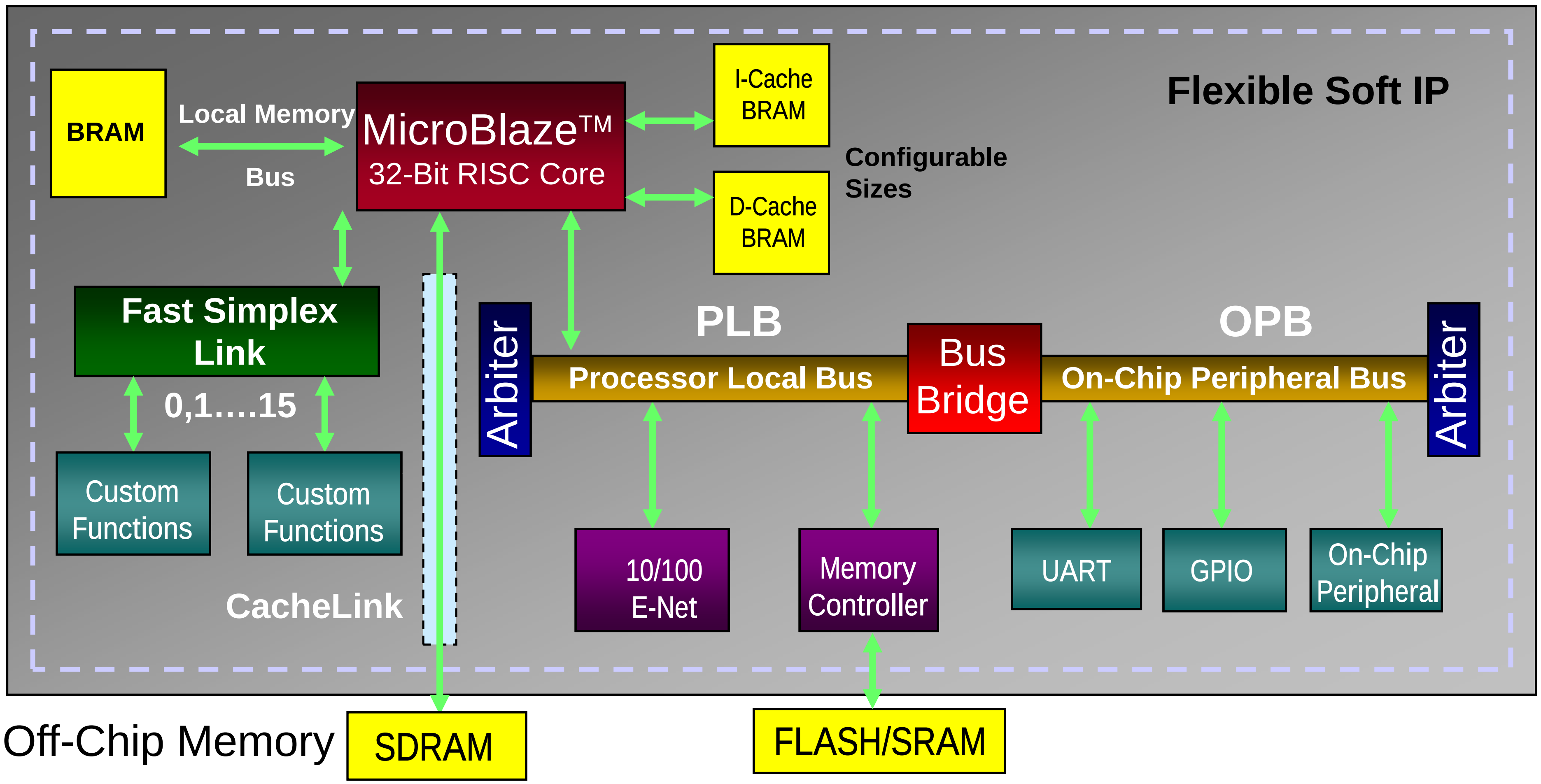
Tear this page out for reference during the course

EDK Files

- MHS = Microprocessor Hardware Specification
- MSS = Microprocessor Software Specification
- MPD = Microprocessor Peripheral Description
- PAO = Peripheral Analyze Order
- BBD = Black-Box Definition
- MDD = Microprocessor Driver Description
- BMM = BRAM Memory Map

Tear this page out for reference during the course

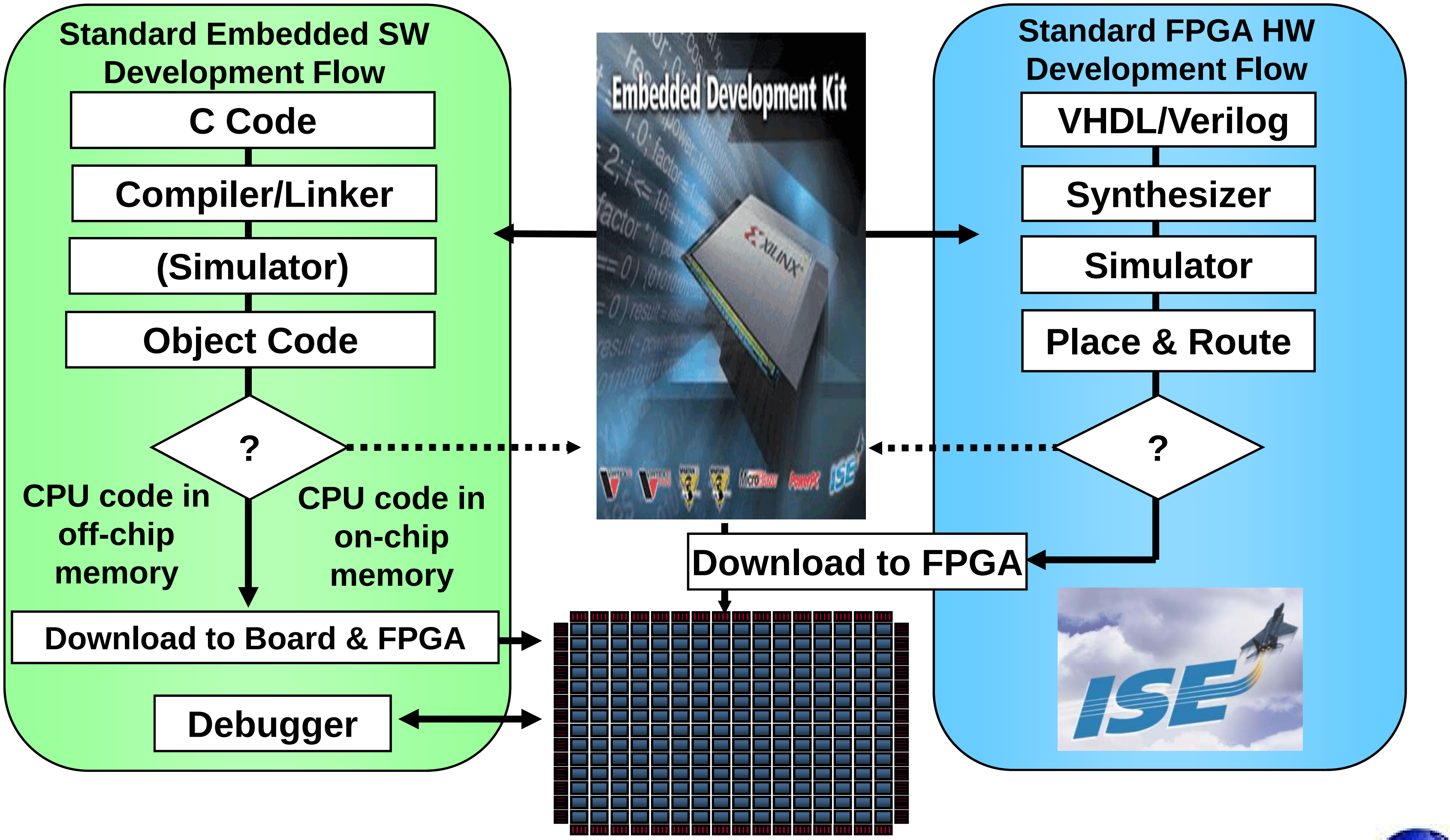
MicroBlaze Processor-Based Embedded Design



This is a v7.1 architecture. Versions 6.0 or earlier do not support PLB bus off the processor. Instead they have OPB bus

Embedded Development

Tool Flow Overview



Standard Embedded SW Development Flow

C Code

Compiler/Linker

(Simulator)

Object Code

?

CPU code in off-chip memory

CPU code in on-chip memory

Download to Board & FPGA

Debugger

Standard FPGA HW Development Flow

VHDL/Verilog

Synthesizer

Simulator

Place & Route

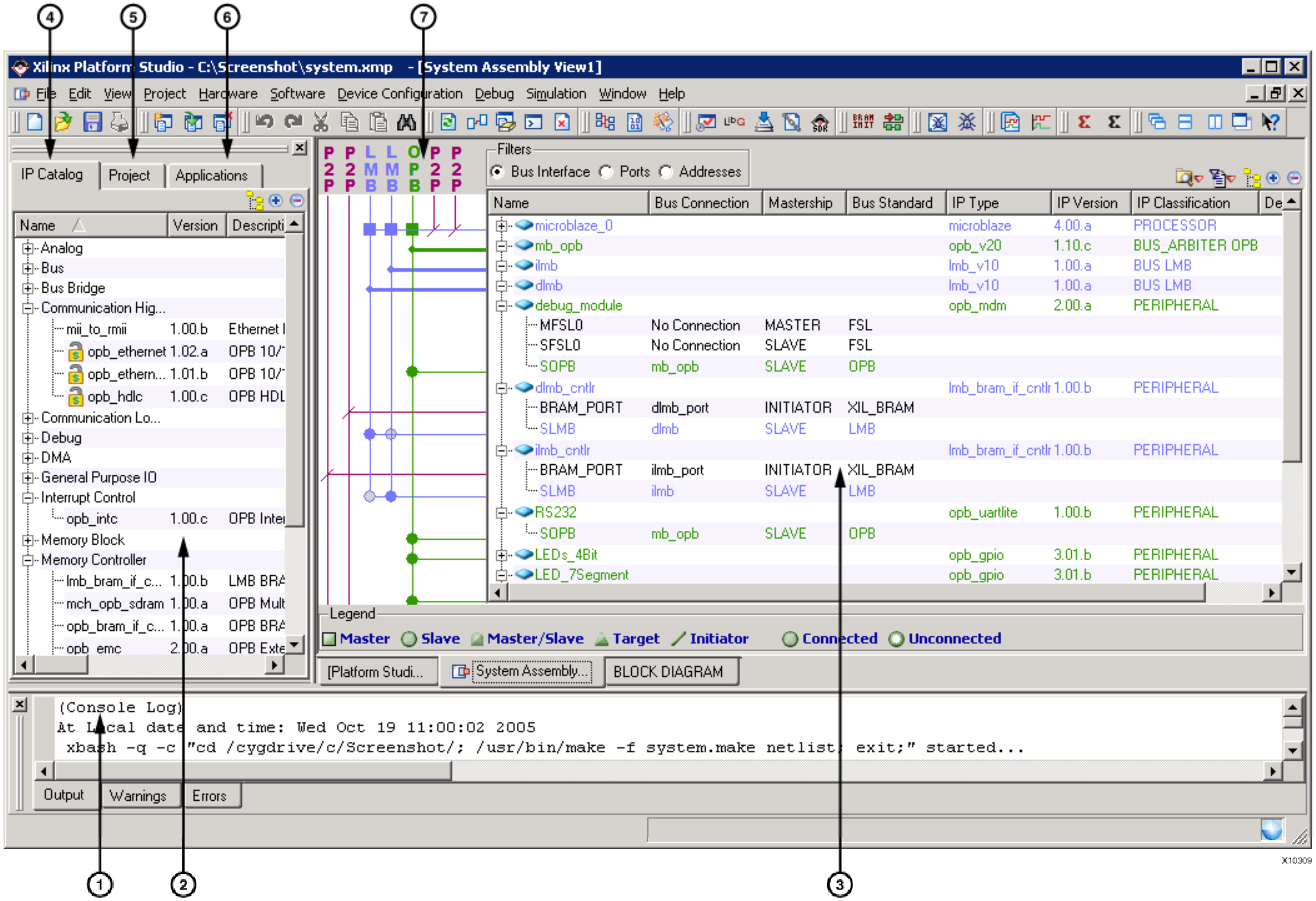
?

Download to FPGA

EDK

- The Embedded Development Kit (EDK) consists of the following:
  - Xilinx Platform Studio – XPS
  - Base System Builder – BSB
  - Create and Import Peripheral Wizard
  - Hardware generation tool – PlatGen
  - Library generation tool – LibGen
  - Simulation generation tool – SimGen
  - GNU software development tools
  - System verification tool – XMD
  - Virtual Platform generation tool - VPgen
  - Software Development Kit (Eclipse)
  - Processor IP
  - Drivers for IP
  - Documentation
- Use the GUI or the shell command tool to run EDK

Xilinx Platform Studio (XPS)



Name	Version	Descripti
⊞ Analog		
⊞ Bus		
⊞ Bus Bridge		
⊟ Communication Hig...		
mii_to_rmii	1.00.b	Ethernet I
opb_ethernet	1.02.a	OPB 10/1
opb_ethern...	1.01.b	OPB 10/1
opb_hdlc	1.00.c	OPB HDL
⊞ Communication Lo...		
⊞ Debug		
⊞ DMA		
⊞ General Purpose IO		
⊟ Interrupt Control		
opb_intc	1.00.c	OPB Inter
⊞ Memory Block		
⊟ Memory Controller		
lmb_bram_if_c...	1.00.b	LMB BRA
mch_opb_sdram	1.00.a	OPB Mult
opb_bram_if_c...	1.00.a	OPB BRA
opb_emc	2.00.a	OPB Exte

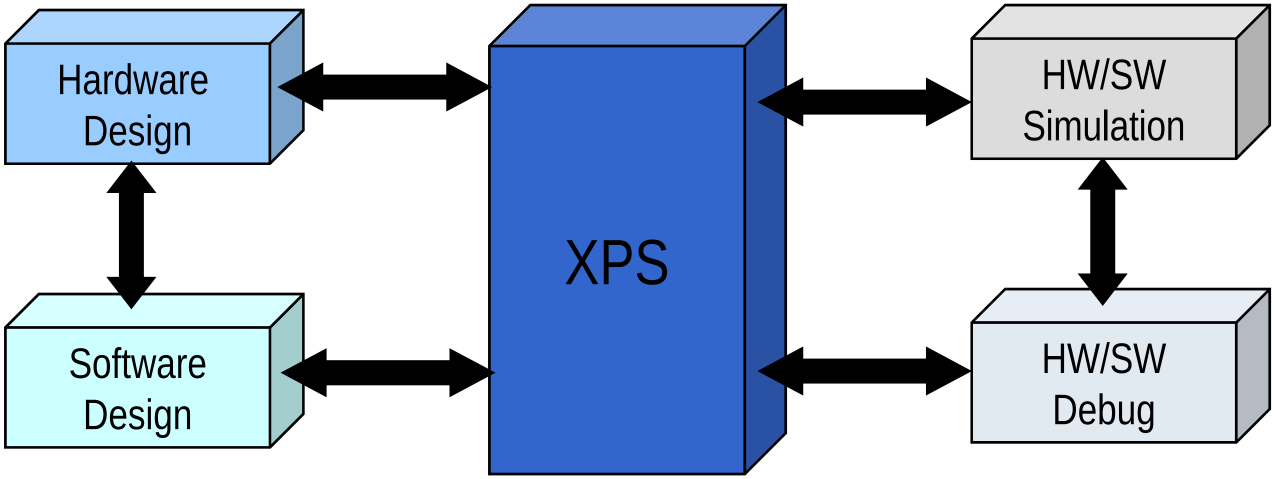
Name	Bus Connection	Mastership	Bus Standard	IP Type	IP Version	IP Classification	De
⊞ microblaze_0				microblaze	4.00.a	PROCESSOR	
⊟ mb_opb				opb_v20	1.10.c	BUS_ARBITER OPB	
⊟ ilmb				lmb_v10	1.00.a	BUS LMB	
⊟ dlmb				lmb_v10	1.00.a	BUS LMB	
⊟ debug_module				opb_mdm	2.00.a	PERIPHERAL	
MFSL0	No Connection	MASTER	FSL				
SFSL0	No Connection	SLAVE	FSL				
SOPB	mb_opb	SLAVE	OPB				
⊟ dlmb_cntlr				lmb_bram_if_cntlr	1.00.b	PERIPHERAL	
BRAM_PORT	dlmb_port	INITIATOR	XIL_BRAM				
SLMB	dlmb	SLAVE	LMB				
⊟ ilmb_cntlr				lmb_bram_if_cntlr	1.00.b	PERIPHERAL	
BRAM_PORT	ilmb_port	INITIATOR	XIL_BRAM				
SLMB	ilmb	SLAVE	LMB				
⊟ RS232				opb_uartlite	1.00.b	PERIPHERAL	
SOPB	mb_opb	SLAVE	OPB				
⊞ LEDs_4Bit				opb_gpio	3.01.b	PERIPHERAL	
⊟ LED_7Segment				opb_gpio	3.01.b	PERIPHERAL	

```
(Console Log)
At Local date and time: Wed Oct 19 11:00:02 2005
 xbash -q -c "cd /cygdrive/c/Screenshot/; /usr/bin/make -f system.make netlist; exit;" started...
```

XPS Functions

- Project management
  - MHS or MSS file
  - XMP file
- Software application management

- Platform management
  - Tool flow settings
  - Software platform settings
  - Tool invocation
  - Debug and simulation



XPS Platform Management

- Platform management tasks of XPS include:
  - Hardware Generation (PlatGen)
  - Library and device driver configuration (LibGen)
  - Simulation model generation (SimGen)
  - Implementation (Xflow or ISE™)
  - Compilation (GNU Compiler)
  - Bitstream initialization (Data2MEM)
- For changing the system specification and software settings, XPS supports the following features and processes:
  - Add cores, edit core parameters, and make bus and port connections through System Assembly view
  - Generate and modify MSS file through Software Platform Settings
  - Tool Flow Settings
  - Tool Invocation

Modifying the Hardware

- Add cores, edit core parameters, and make bus and port connections through System Assembly view

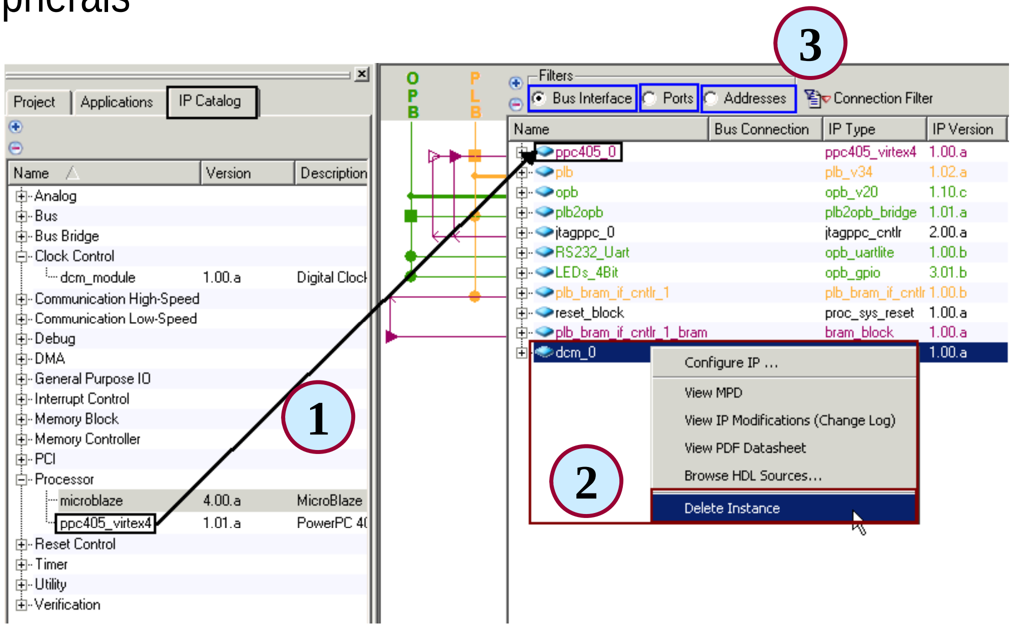
1 Select IP Catalog tab to add peripherals

- Select a core and drop it in the system view or double-click on it to add

2 In the System View select an instance, right click, and then select Delete Instance

3 Change settings using appropriate filters and select an instance

- Base and end addresses
- Parameters
- Ports



Name	Version	Description
Analog		
Bus		
Bus Bridge		
Clock Control		
dcm_module	1.00.a	Digital Clock
Communication High-Speed		
Communication Low-Speed		
Debug		
DMA		
General Purpose IO		
Interrupt Control		
Memory Block		
Memory Controller		
PCI		
Processor		
microblaze	4.00.a	MicroBlaze
ppc405_virtex4	1.01.a	PowerPC 40
Reset Control		
Timer		
Utility		
Verification		

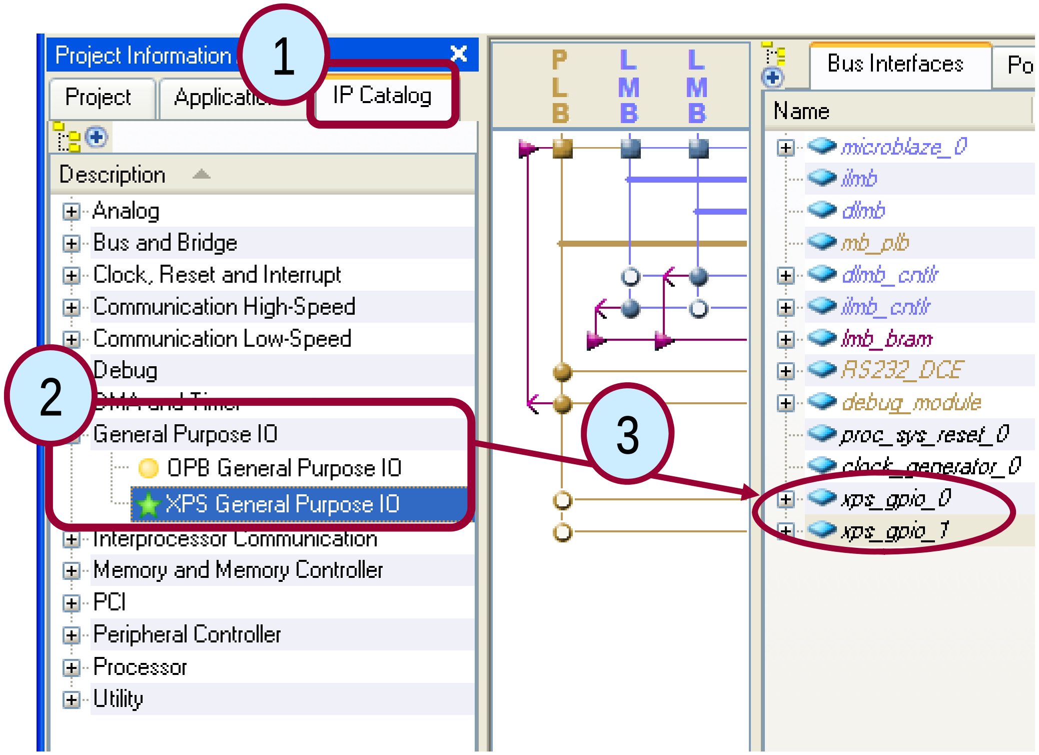
Name	Bus Connection	IP Type	IP Version
ppc405_0		ppc405_virtex4	1.00.a
plb		plb_v34	1.02.a
opb		opb_v20	1.10.c
plb2opb		plb2opb_bridge	1.01.a
jtagppc_0		jtagppc_cntlr	2.00.a
RS232_Uart		opb_uartlite	1.00.b
LEDs_4Bit		opb_gpio	3.01.b
plb_bram_if_cntlr_1		plb_bram_if_cntlr	1.00.b
reset_block		proc_sys_reset	1.00.a
plb_bram_if_cntlr_1_bram		bram_block	1.00.a
dcm_0			1.00.a

Adding IP to Design

1 To add hardware in a new, empty project or to an existing project, select IP Catalog tab in XPS

2 Expand group(s) of IP in the left window

3 Select an IP and drag it to the System Assembly View window or double-click on the selected IP to be included into the system MHS file



Making Bus Connections

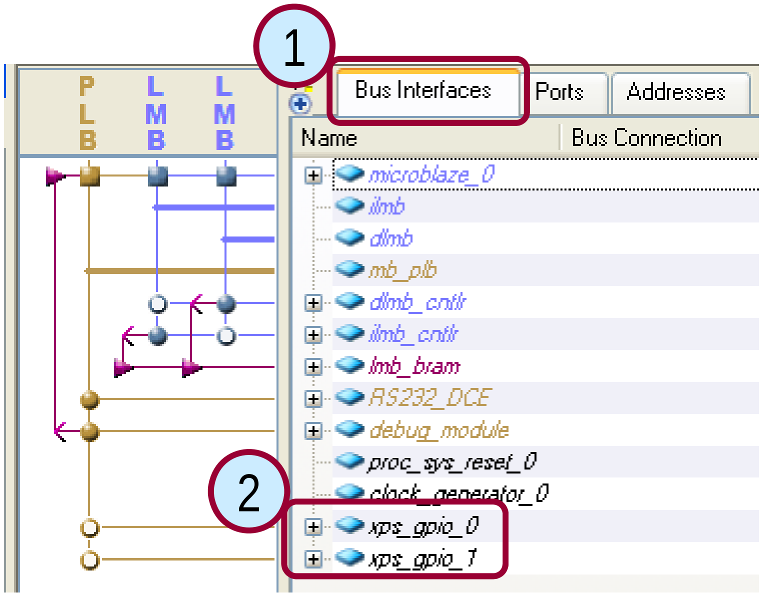
MicroBlaze communicates with external peripheral devices using busses

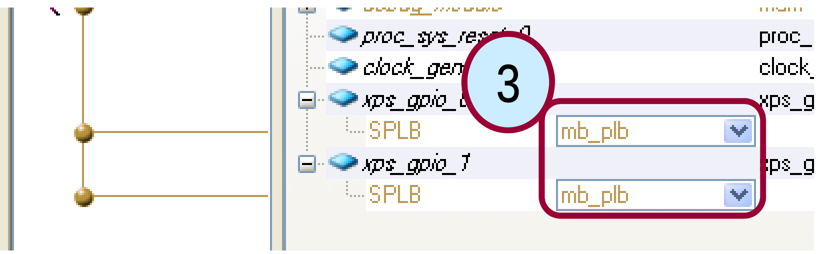
1 Select Bus Interfaces tab

- Expand Peripherals in System View

2 Click under Bus Connection column, and select a bus instance to which it needs to connect

3





xps_gpio_0 SPLB mb_plb

xps_gpio_1 SPLB mb_plb

Assigning Addresses

MicroBlaze communicates with external devices through registers or memories at specific address ranges

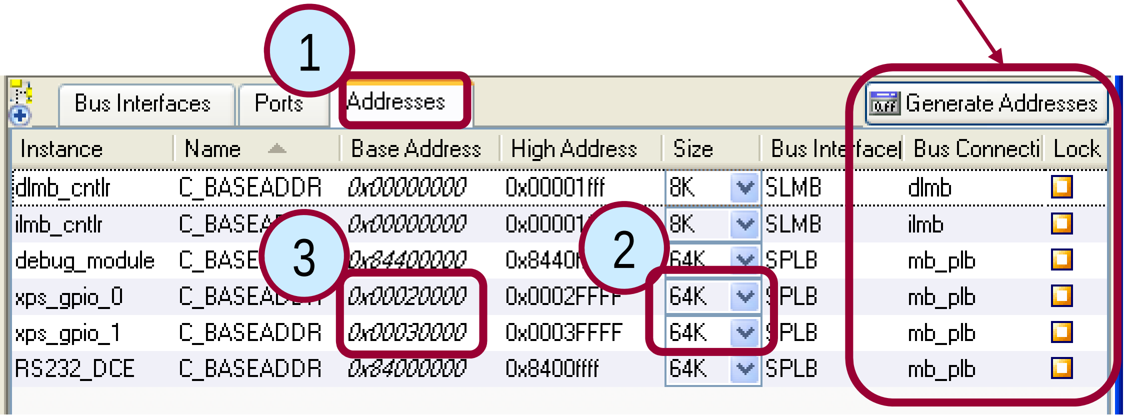
1. Select **Addresses** filter

2. Click in the size column and select desired size

3. Enter base address
   - XPS will calculate the high address from base address and size entries

4. Instead of entering base address, lock addresses of instances for which you don’t want XPS to change address and then click Generate Addresses button

4 Lock addresses and click generate



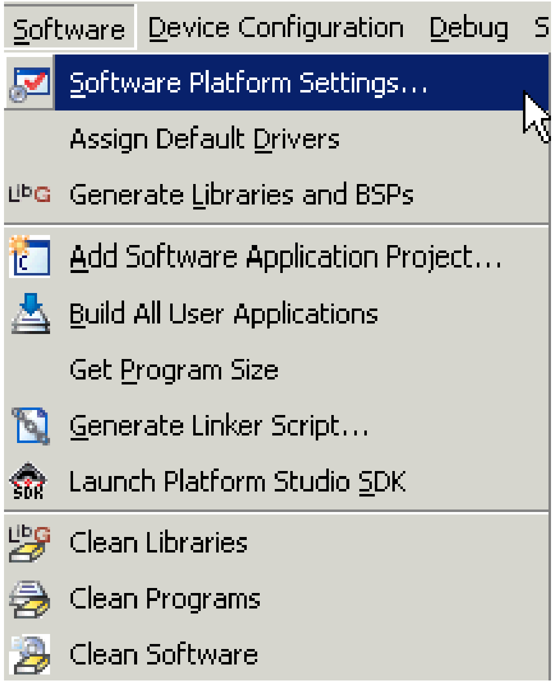
Instance	Name	Base Address	High Address	Size	Bus Interface	Bus Connecti	Lock
dlmb_cntlr	C_BASEADDR	0x00000000	0x00001fff	8K	SLMB	dlmb	□
ilmb_cntlr	C_BASEA	0x00000000	0x00001	8K	SLMB	ilmb	□
debug_module	C_BASE	0x84400000	0x8440f	64K	SPLB	mb_plb	□
xps_gpio_0	C_BASEAD	0x00020000	0x0002FFFF	64K	SPLB	mb_plb	□
xps_gpio_1	C_BASEADDR	0x00030000	0x0003FFFF	64K	SPLB	mb_plb	□
RS232_DCE	C_BASEADDR	0x84000000	0x8400ffff	64K	SPLB	mb_plb	□

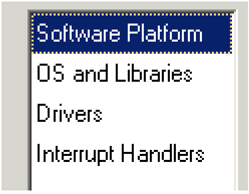
Software Application Management

- XPS supports test application creation and linker script management through BSB
- XPS lets you specify multiple application projects in the **Applications** tab
- XPS has an integrated editor for viewing and editing the C source and header files of the user program
- The source code is grouped for each processor instance. You can add or delete the list of source code files for each processor
- All of the source code files for a processor are compiled by using the compiler specified for that processor
- XPS tracks changes to C/C++ source files and recompiles when necessary
- Can launch the Platform Studio Software Development Kit (SDK)

Editing Software Settings

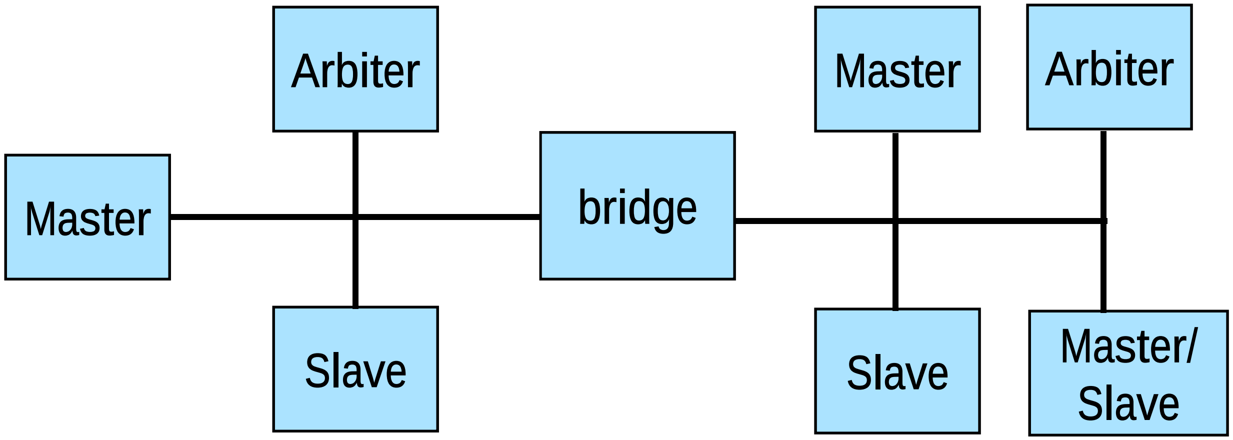
- Sets all of the software platform-related options in the design
- Has multiple forms selection:
  - Software Platform
    - CPU Driver
    - OS and OS Version selection
    - Libraries selection
    - Set core clock frequencies
  - OS and Libraries
    - Identify stdin and stdout devices
    - Configure OS and selected libraries
  - Drivers
    - Select drivers and versions
    - Core clock frequency
  - Interrupt Handlers
    - Enter interrupt handler function names


Software
Device Configuration
Debug
S
Software Platform Settings...
Assign Default Drivers
Generate Libraries and BSPs
Add Software Application Project...
Build All User Applications
Get Program Size
Generate Linker Script...
Launch Platform Studio SDK
Clean Libraries
Clean Programs
Clean Software



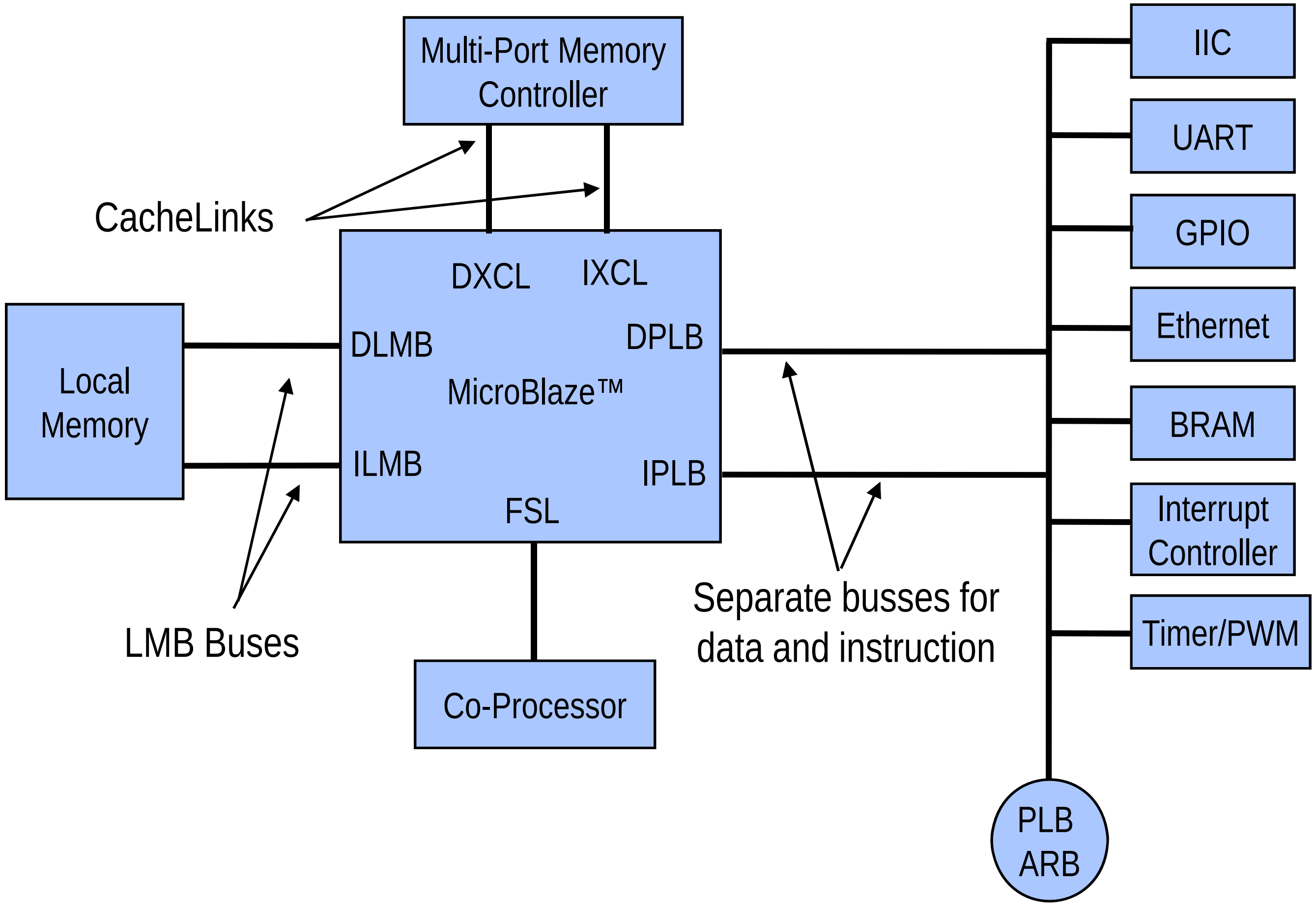
Buses 101

- A bus is a multi-wire path on which related information is delivered
  - Address, data, and control buses
- Processor and peripherals communicate through buses
- Peripherals may be classified as
  - Arbiter, master, slave, or master/slave



MicroBlaze Bus Example

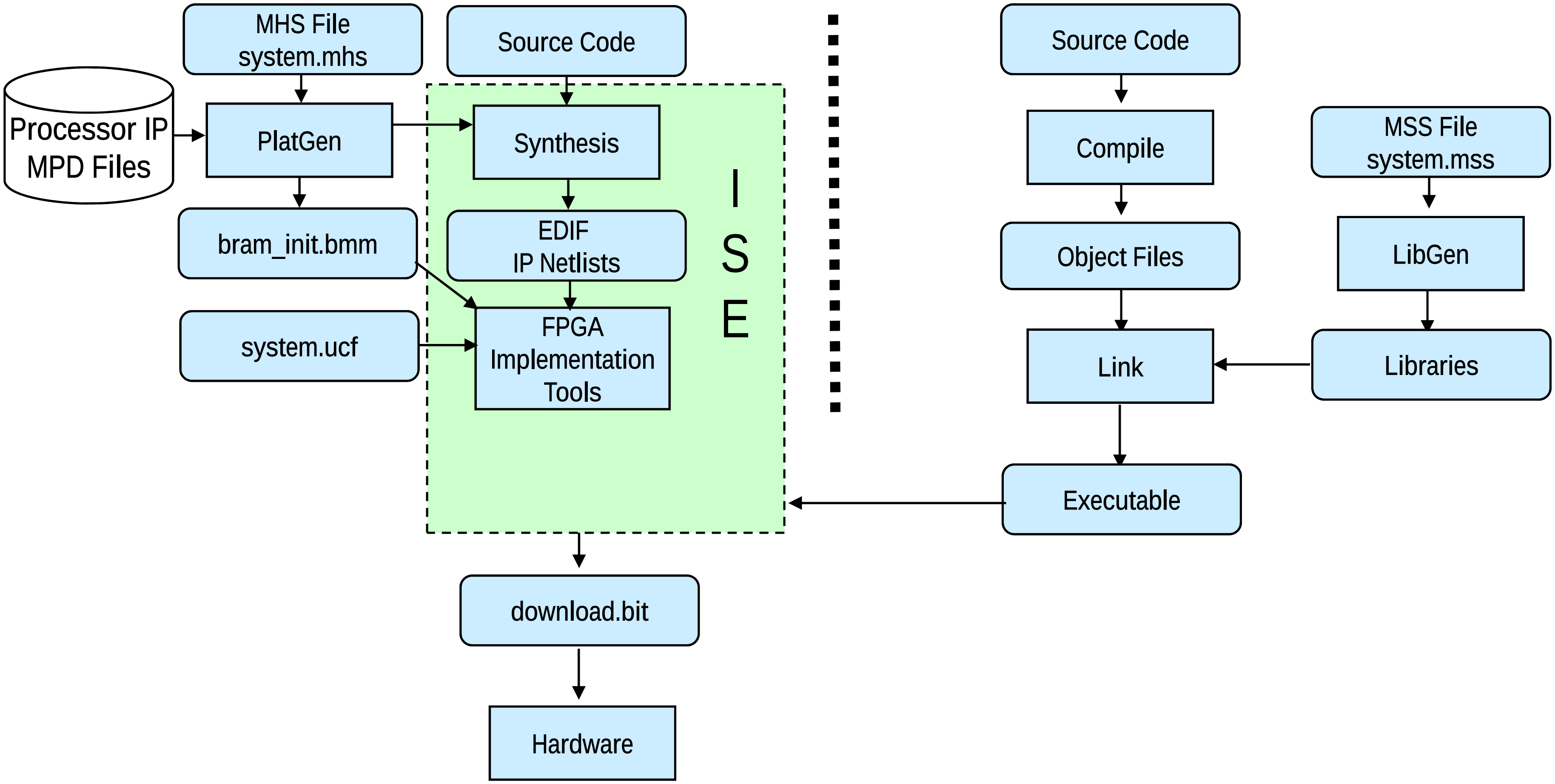
The MicroBlaze processor core is organized as a Harvard architecture



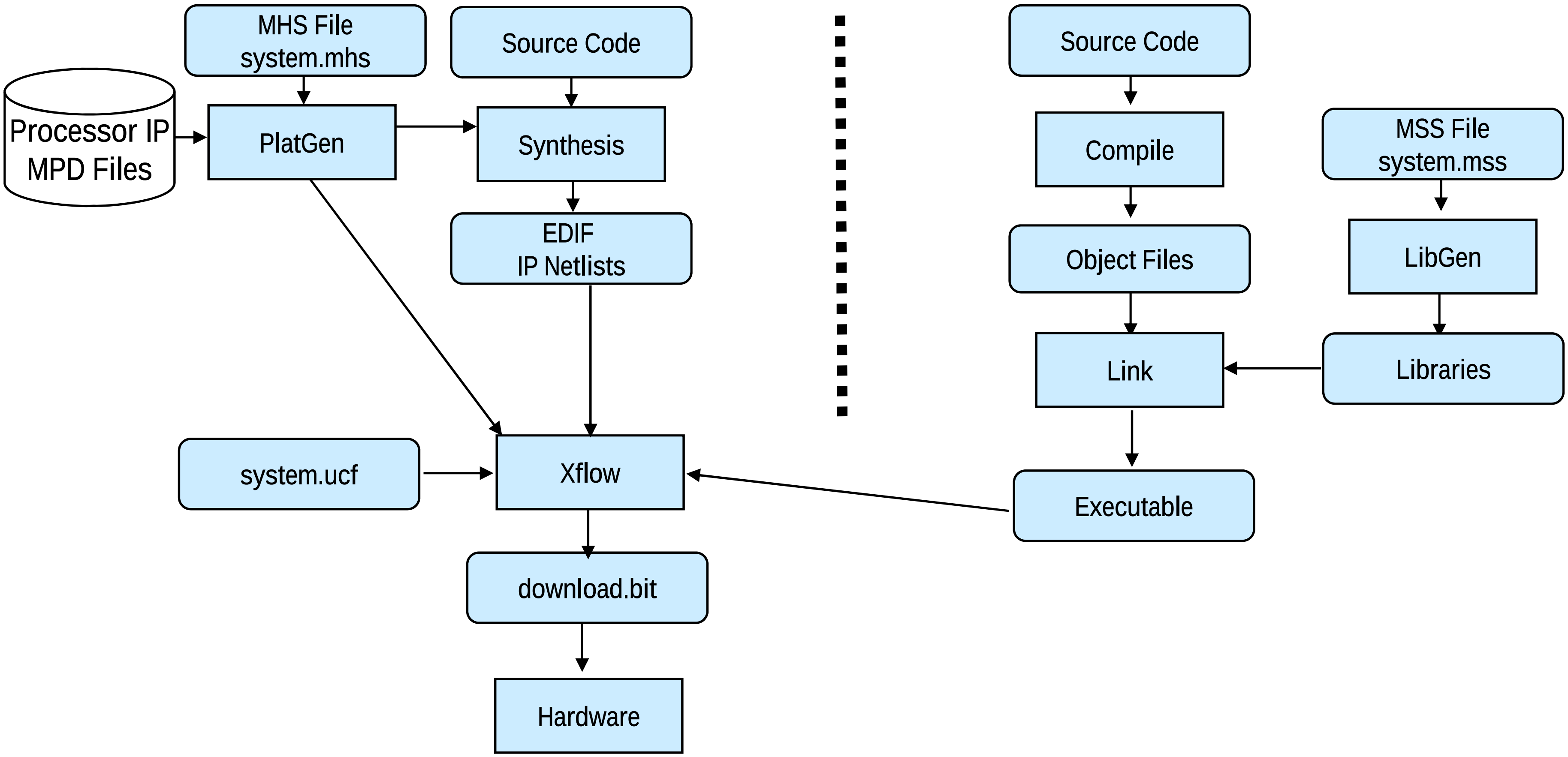
MicroBlaze Processor

- Scalable 32-bit Core
  - Single-Issue pipeline
    - Supports either 3-stage (resource focused) or 5-stage pipeline (performance focused)
  - Configurable Instruction and Data Caches
    - Direct mapped (1-way associative)
  - Optional Memory Mgt or Memory Protection Unit
    - Required for Linux OS (Linux 2.6 is currently supported)
  - Floating-point unit (FPU)
    - Based upon IEEE 754 format
  - Barrel Shifter
  - Hardware multiplier
    - 32x32 multiplication to generate a 64-bit result
  - Hardware Divider
  - Fast Simplex Link FIFO Channels for Easy, Direct Access to Fabric and Hardware Acceleration
  - Hardware Debug and Trace Module

EDK: ISE


MHS File
system.mhs
Source Code
Processor IP
MPD Files
PlatGen
Synthesis
bram_init.bmm
EDIF
IP Netlists
system.ucf
FPGA
Implementation
Tools
ISE
download.bit
Hardware
Source Code
Compile
MSS File
system.mss
Object Files
LibGen
Link
Libraries
Executable

EDK: Xflow


MHS File
system.mhs
Source Code
Processor IP
MPD Files
PlatGen
Synthesis
EDIF
IP Netlists
system.ucf
Xflow
download.bit
Hardware
Source Code
Compile
MSS File
system.mss
Object Files
LibGen
Link
Libraries
Executable

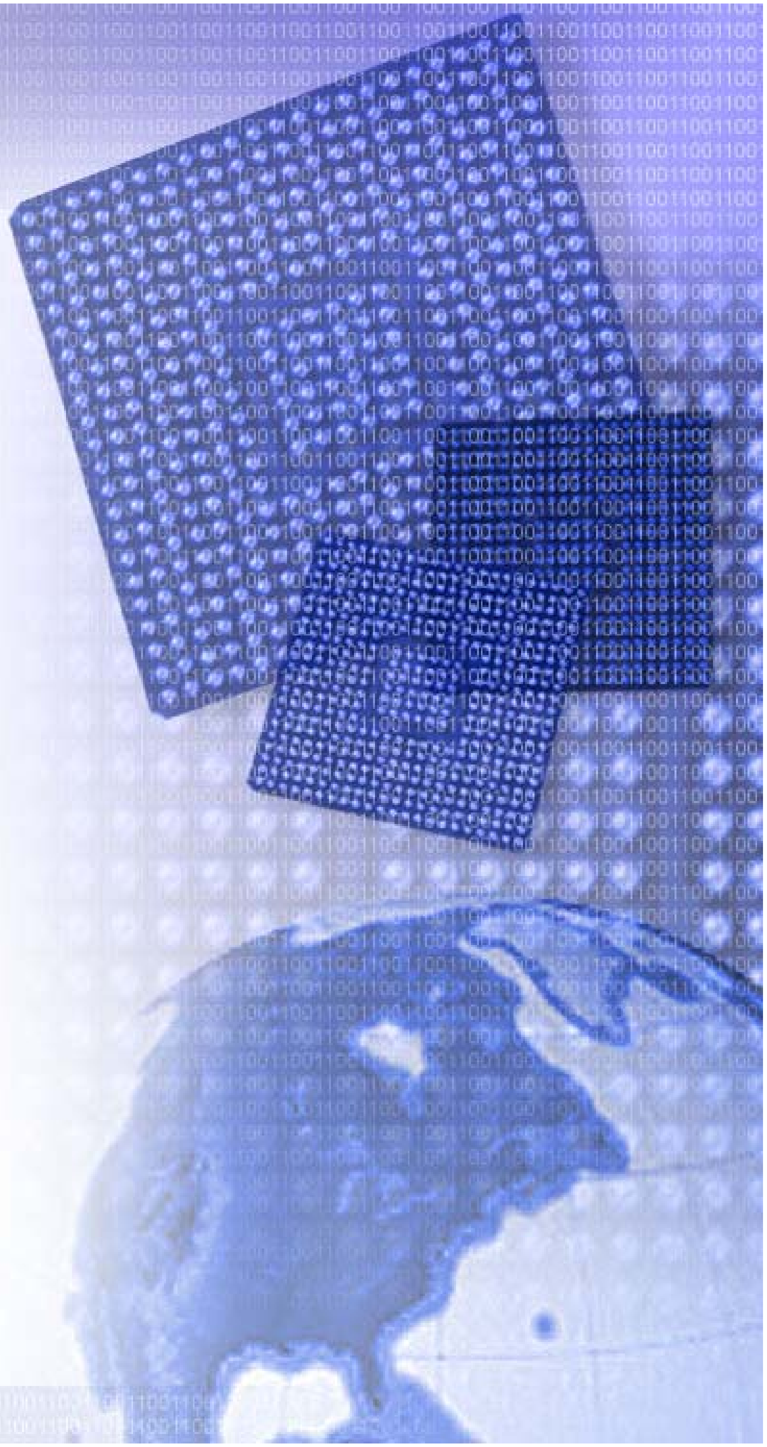
Xflow

Required XPS Directory Structure

- project_directory
  - code directory
  - data directory
  - etc directory
  - pcores
  - synthesis
  - TestApp [optional]

- code directory
  - <application>.c
- data directory
  - <system>.ucf
- etc directory
  - .opt
  - bitgen.ut
  - download.cmd
  - fast_runtime.opt
  - BSDL files
- pcores directory
  - User IP
  - Customized BRAM controllers

Adding Your Own IP to the PLB / OPB Bus

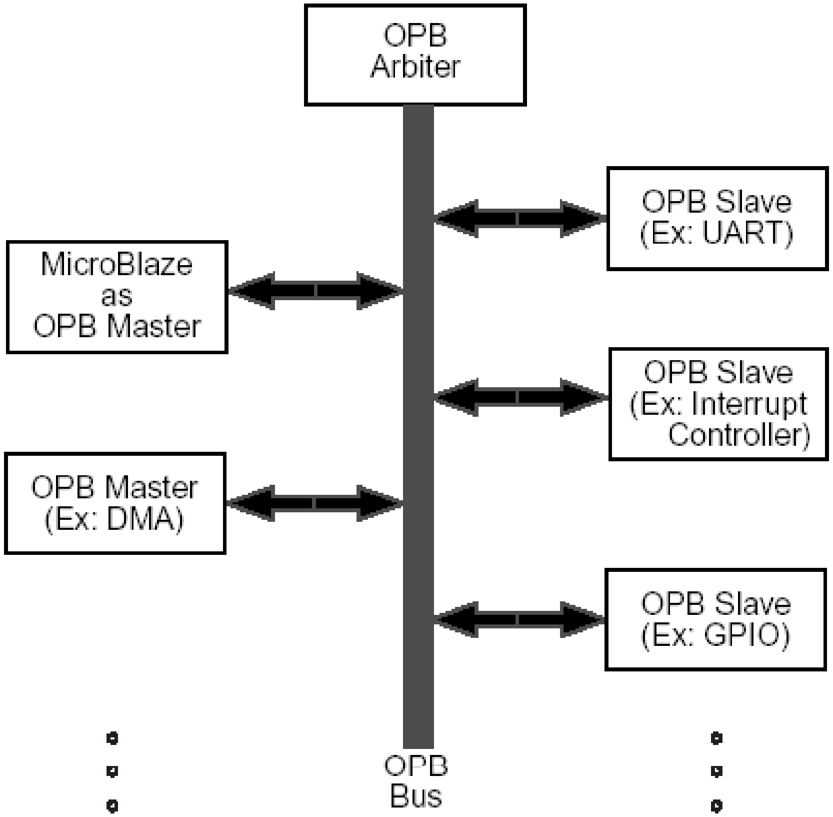


Overview

- Peripherals are connected to the microprocessor by using the data and address buses
- Xilinx has implemented the IBM CoreConnect bus architecture
- Processor Local Bus (PLB) version 4.6 of the CoreConnect bus architecture is designed for easy connection of on-chip peripheral devices
- Any custom peripheral that connects to the PLB bus must do the following:
  - Meet the principles of the PLB protocol
  - Meet the requirements of the Platform Generator
    - This allows you to take advantage of the simple automated flow that generates system-level architecture

Features

- MicroBlaze™ embedded system (here using OPB)



IP Cores

Example free EDK included cores

	MicroBlaze	MicroBlaze PowerPC™	PowerPC™
Bus	fsl, lmb	opb	dcr, ocm, plb, fcb
Bus Bridge		opb2opb, opb2dcr, opb2plb	fcb2fsl, plb2opb
Communication		opb_spi	hard_temac, plb_temac
Debug		icon, iba, ila, vio, mdm	Iba, jtagppc_cntlr
GPIO		opb_gpio	plb_gpio
Interrupt Controller		opb_intc	dcr_intc
Memory Controller		mch_opb_ddr, mch_opb_sdram, opb_bram, opb_ddr, opb_emc, opb_sdram, opb_sysace	dsbram, isbram, pb_ddr, plb_emc, plb_sdram
Timer		fit_timer, opb_timer, opb_timebase_wdt	
Utility		bus_split, flipflop, reduced_logic, vector_logic	

IP Cores

Example evaluation cores

- OPB UART-16550
- OPB HDLC
- OPB IIC
- OPB Ethernet 10/100 MAC and Ethernet-Lite 10/100 MAC
- OPB ATM Master Utopia Level 2
- OPB ATM Slave Utopia Level 2
- OPB PCI 32 Bridge
- OPB ATM Master Utopia Level 3
- OPB ATM Slave Utopia Level 3
- PLB ATM Master Utopia Level 2
- PLB ATM Slave Utopia Level 2
- PLB Ethernet
- PLB RapidIO

Create/Import Peripheral Wizard

- The wizard helps you create your own peripheral and then import it into your design
- The wizard will generate the necessary core description files into the user selected directory
- You can start the wizard after creating a new project or opening an existing project in XPS
- The user peripheral can be imported directly through the wizard by skipping the creation option
  - Ensure that the peripheral complies with Xilinx implementation of the IBM CoreConnect™ Bus Standard

MPD File

```
## MPD file created automatically for design OPB_SEMAPHORE

BEGIN opb_pwm, IPTYPE=PERIPHERAL

## Parameter list for the generics



PARAMETER C_OPB_AWIDTH = 32, DT = integer
PARAMETER C_OPB_DWIDTH = 32, DT = integer
PARAMETER C_BASEADDR = 0xFFFF8000, DT = std_logic_vector
PARAMETER C_HIGHADDR = 0xFFFF80FF, DT = std_logic_vector
PARAMETER C_NO_CHANNELS = 4, DT = integer
PARAMETER C_MAX_RESOLUTION = 16, DT = integer

OPTION SIM_MODELS = BEHAVIORAL : STRUCTURAL

BUS_INTERFACE BUS=SOPB, BUS_STD=OPB, BUS_TYPE=SLAVE
```

Parameters override generics in VHDL

```
entity OPB_PWM is
 generic (
   C_OPB_AWIDTH     : integer                      := 32;
   C_OPB_DWIDTH     : integer                      := 32;
   C_BASEADDR       : std_logic_vector(0 to 31)    := X"FFFFA000";
   C_HIGHADDR       : std_logic_vector             := X"FFFFA0FF";
   C_NO_CHANNELS    : integer range 0 to 15        := 4;
   C_MAX_RESOLUTION : integer range 4 to 32        := 16
   );
```

MPD File

```
## Port list for the signals
## Global signals
PORT OPB_Clk = "", DIR = in, SIGIS=CLK, BUS=SOPB
PORT OPB_Rst = OPB_Rst, DIR = in, BUS=SOPB
## OPB signals
PORT OPB_ABus = OPB_ABus, DIR = in, VEC = [0:31], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR = in, VEC = [0:3], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR = in, BUS=SOPB
PORT OPB_select = OPB_select, DIR = in, BUS=SOPB
PORT OPB_seqAddr = OPB_seqAddr, DIR = in, BUS=SOPB
PORT OPB_DBus = OPB_DBus, DIR = in, VEC = [0:31], BUS=SOPB

PORT PWM_DBus = Sl_DBus, DIR = out, VEC = [0:31], BUS=SOPB
PORT PWM_errAck = Sl_errAck, DIR = out, BUS=SOPB
PORT PWM_retry = Sl_retry, DIR = out, BUS=SOPB
PORT PWM_toutSup = Sl_toutSup, DIR = out, BUS=SOPB
PORT PWM_xferAck = Sl_xferAck, DIR = out, BUS=SOPB

PORT PWM = "", DIR = out, VEC = [0:C_NO_CHANNELS-1]
END
```

OPB Bus Signals

```
port (
   -- Global signals
   OPB_Clk : in std_logic;
   OPB_Rst : in std_logic;

   -- OPB signals
   OPB_ABus    : in std_logic_vector(0 to 31);
   OPB_BE      : in std_logic_vector(0 to 3);
   OPB_RNW     : in std_logic;
   OPB_select  : in std_logic;
   OPB_seqAddr : in std_logic;
   OPB_DBus    : in std_logic_vector(0 to 31);
```

Slave Signals

```
   PWM_DBus    : out std_logic_vector(0 to 31);
   PWM_errAck  : out std_logic;
   PWM_retry   : out std_logic;
   PWM_toutSup : out std_logic;
   PWM_xferAck : out std_logic;

   PWM : out std_logic_vector(0 to C_NO_CHANNELS-1)
   );
```

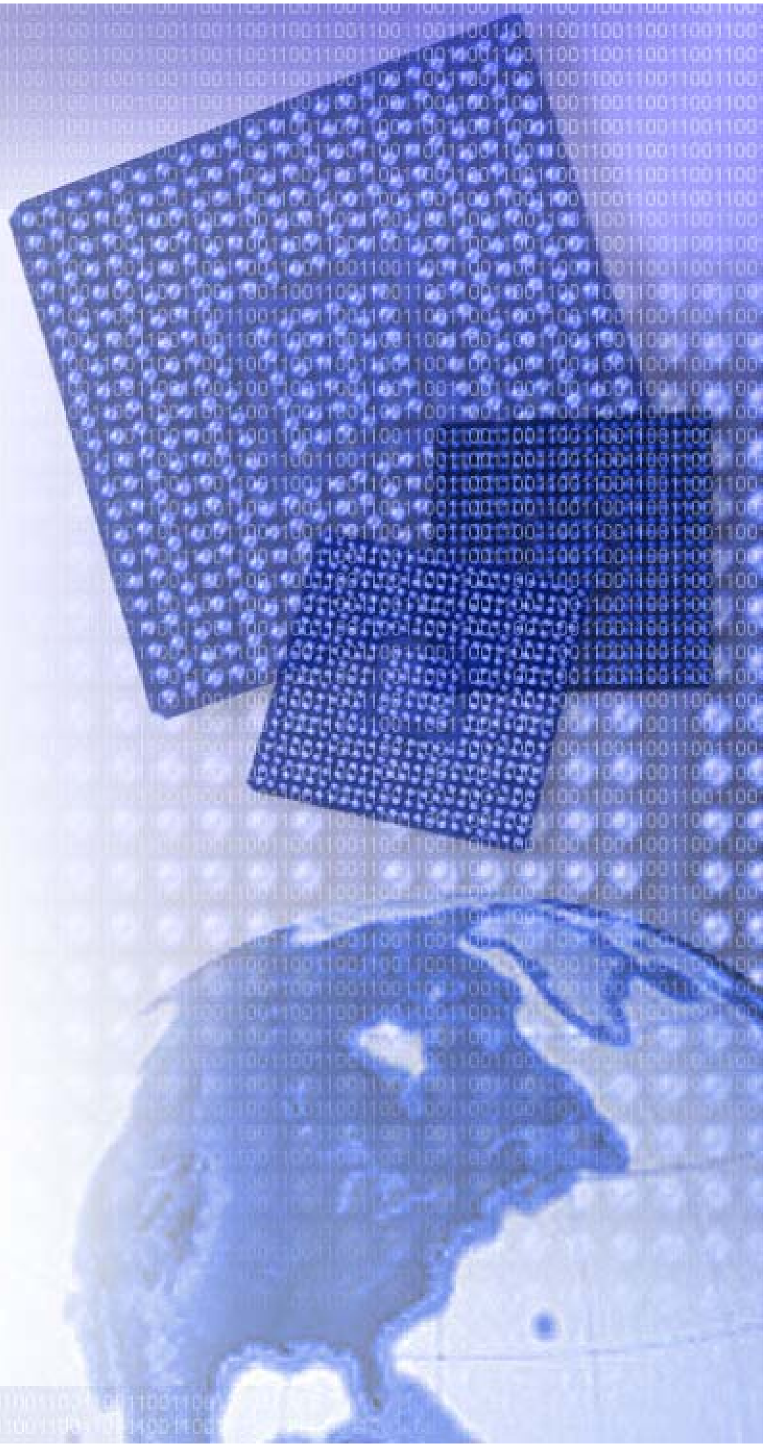
PAO File

```
#######################################################################
# opb_core_ssp0 pao file
#######################################################################
lib proc_common_v1_00_b    proc_common_pkg
lib proc_common_v1_00_b    pselect
lib proc_common_v1_00_b    or_muxcy
lib ipif_common_v1_00_a    ipif_pkg
lib ipif_common_v1_00_a    ipif_steer
lib opb_bus_attach_v1_00_a reset_mir
lib opb_bus_attach_v1_00_a opb_bus_attach
lib opb_ipif_ssp0_v1_00_a  opb_ipif_ssp0

# --USER-- add all user core source files and change the following source to
#   your top level core name and library

lib opb_core_ssp0_v1_00_a pwm
lib opb_core_ssp0_v1_00_a user_logic
lib opb_core_ssp0_v1_00_a opb_core_ssp0
```

Order of dependency

Update this section

Software Development

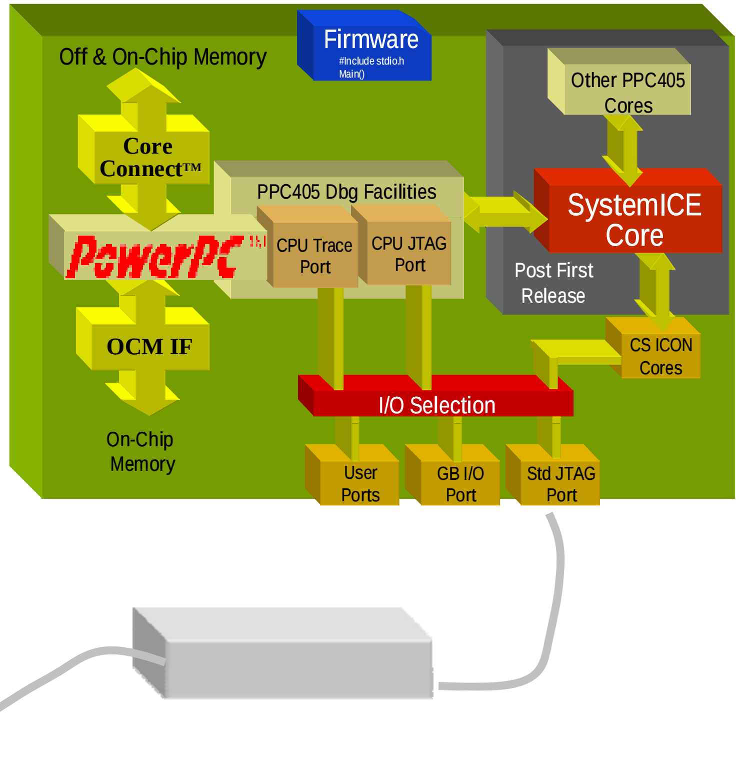


Embedded Development

- Development takes place on one machine (host) and is downloaded to the embedded system (target)

Host Computer

Target Computer



A cross-compiler is run on the host

Embedded Development

- Different set of problems
  - Unique hardware for every design
  - Reliability
  - Real-time response requirement (sometimes)
    - RTOS versus OS
  - Code compactness
  - High-level languages and assembly

Software Design
Environment

- The Library Generator (LibGen) utility generates the necessary libraries and drivers for the embedded processors
- LibGen takes an MSS (Microprocessor Software Specification) file created by the user as input. The MSS file defines the drivers associated with peripherals, standard input/output devices, interrupt handler routines, and other related software features
- The MSS file is generated by XPS by using the software settings specified

LibGen

LibGen Generated Directories

- project_directory
  - Processor instance directory
    - code directory
    - include directory
    - lib directory
    - libsrc directory

Note: The number of processor instance directories generated is related to the number of processor instances present in the system

- code directory
  - A repository for EDK executables
- include directory
  - C header files that are required by drivers
  - xparameters.h
    - Defines base and high addresses of the peripherals in the system
    - Defines the peripheral IDs required by the drivers and user programs
    - Defines the function prototypes

LibGen

LibGen Generated Directories

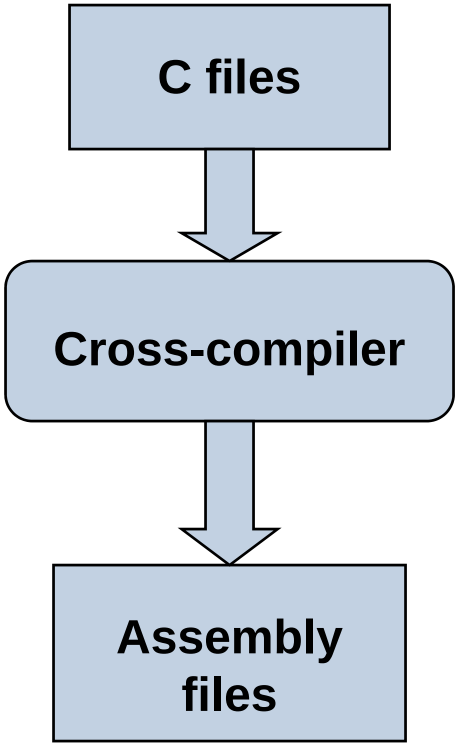
- project_directory
  - processor instance directory
    - code directory
    - include directory
    - lib directory
    - libsrc directory

Note: The processor instance directories content is overwritten every time LibGen is run

- lib directory
  - **libc.a**, **libm.a** and **libxil.a** libraries
    - The libxil library contains driver functions that the particular processor can access
- libsrc directory
  - Intermediate files and makefiles that compile the libraries and drivers
  - Peripheral-specific driver files that are copied from the EDK and user driver directories

GNU Tools: GCC

- GCC translates C source code into assembly language
- GCC also functions as the user interface to the GNU assembler and to the GNU linker, calling the assembler and the linker with the appropriate parameters
- Supported cross-compilers:
  - PowerPC™ processor compiler
    - GNU GCC (powerpc-eabi-gcc)
    - Wind River Diab™ compiler (dcc)
  - MicroBlaze™ processor compiler
    - GNU GCC (mb-gcc)
- Command line only; uses the settings set through the GUI



Hardware IP Device Drivers

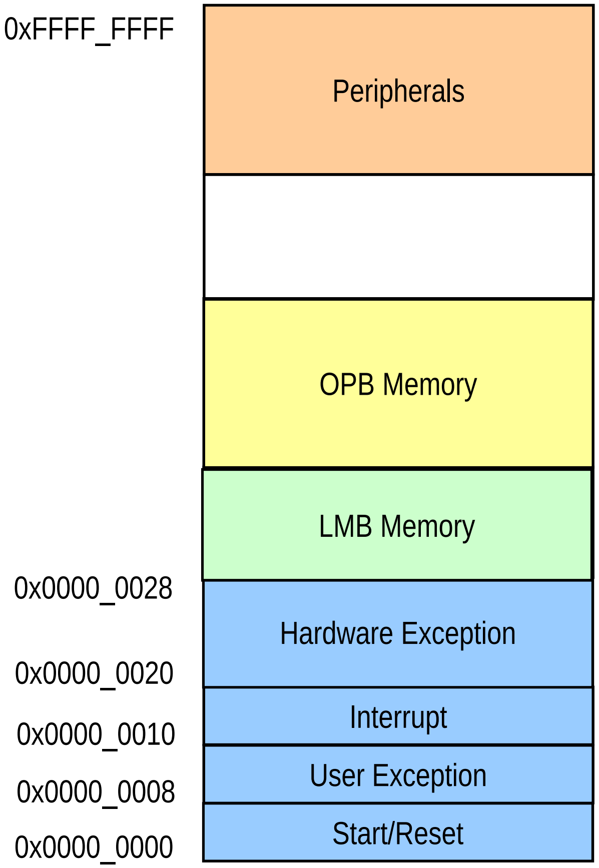
- Driver
  - Provides an interface for the software to communicate with the hardware
  - Designed to be portable across processor architectures and operating systems
- Delivery format
  - Delivered as source code, allowing it to be built and optimized
  - Minimized assembly language
  - C programming language

Address Management

- Embedded processor design requires you to manage the following:
  - Address map for the peripherals
  - Location of the application code in the memory space
    - BRAM
    - External memory
- Memory requirements for your programs are based on the following:
  - The amount of memory required for storing the instructions
  - The amount of memory required for storing the data associated with the program

MicroBlaze Processor

- Memory and peripherals
  - MicroBlaze™ processor uses 32-bit addresses
- Special addresses
  - MicroBlaze systems must have user-writable memory from 0x00000000 through 0x00000027
- BRAM size limits
  - The amount of BRAM memory that can be used is limited to the device limitations
  - The largest supported BRAM memory size for Virtex™ and VirtexE devices is 16 kilobytes; for Virtex™-II, it is 64 kilobytes

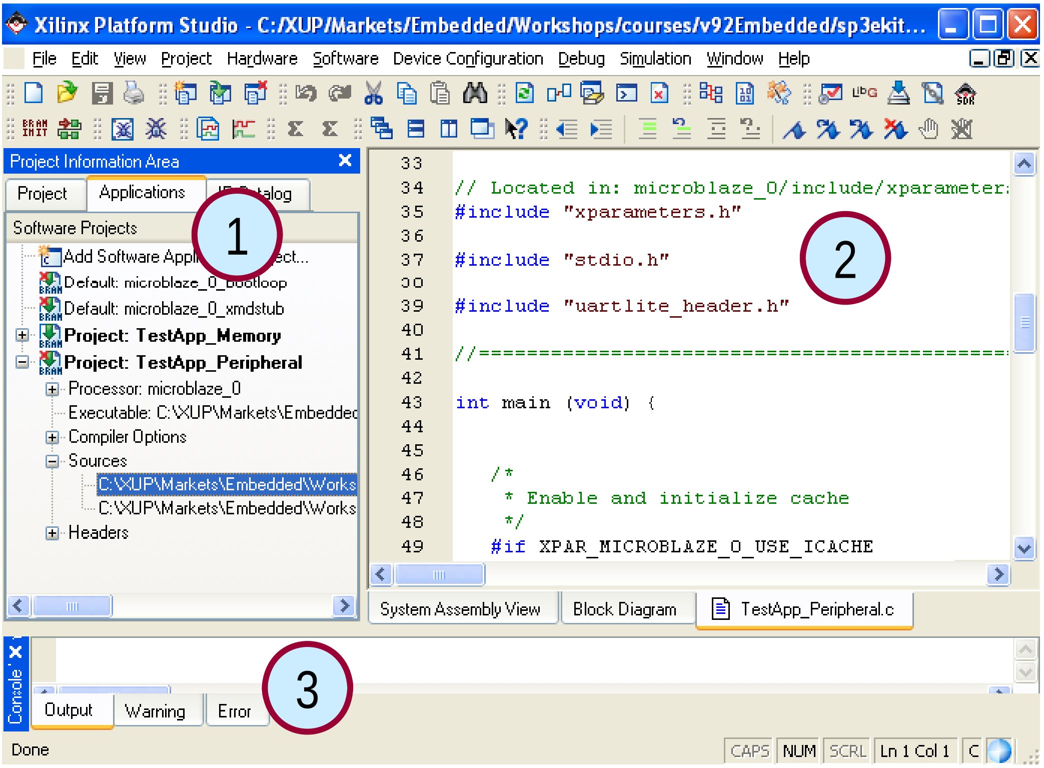


Linker Script

- Linker script
  - Controls the linking process
  - Maps the code and data to a specified memory space
  - Sets the entry point to the executable
  - Reserves space for the stack
- Required if the design contains a discontinuous memory space
- GNU GCC linker scripts will not work for the WindRiver Diab™ compiler

Software Development Environment: XPS

- Allows for simple management of smaller SW projects
- Source file listing
- Text editor (simple)
- Integrated flow


Xilinx Platform Studio - C:/XUP/Markets/Embedded/Workshops/courses/v92Embedded/sp3ekit...
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Project Information Area
Project
Applications
IP Catalog
Software Projects
Add Software Application Project...
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_Memory
Project: TestApp_Peripheral
Processor: microblaze_0
Executable: C:\XUP\Markets\Embedded
Compiler Options
Sources
C:\XUP\Markets\Embedded\Works
C:\XUP\Markets\Embedded\Works
Headers
System Assembly View
Block Diagram
TestApp_Peripheral.c
Console
Output
Warning
Error
Done
CAPS
NUM
SCRL
Ln 1 Col 1
1
2
3

```
// Located in: microblaze_0/include/xparameter
#include "xparameters.h"

#include "stdio.h"

#include "uartlite_header.h"

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

int main (void) {


   /*
    * Enable and initialize cache
    */
   #if XPAR_MICROBLAZE_0_USE_ICACHE
```

Software Development Environment: SDK

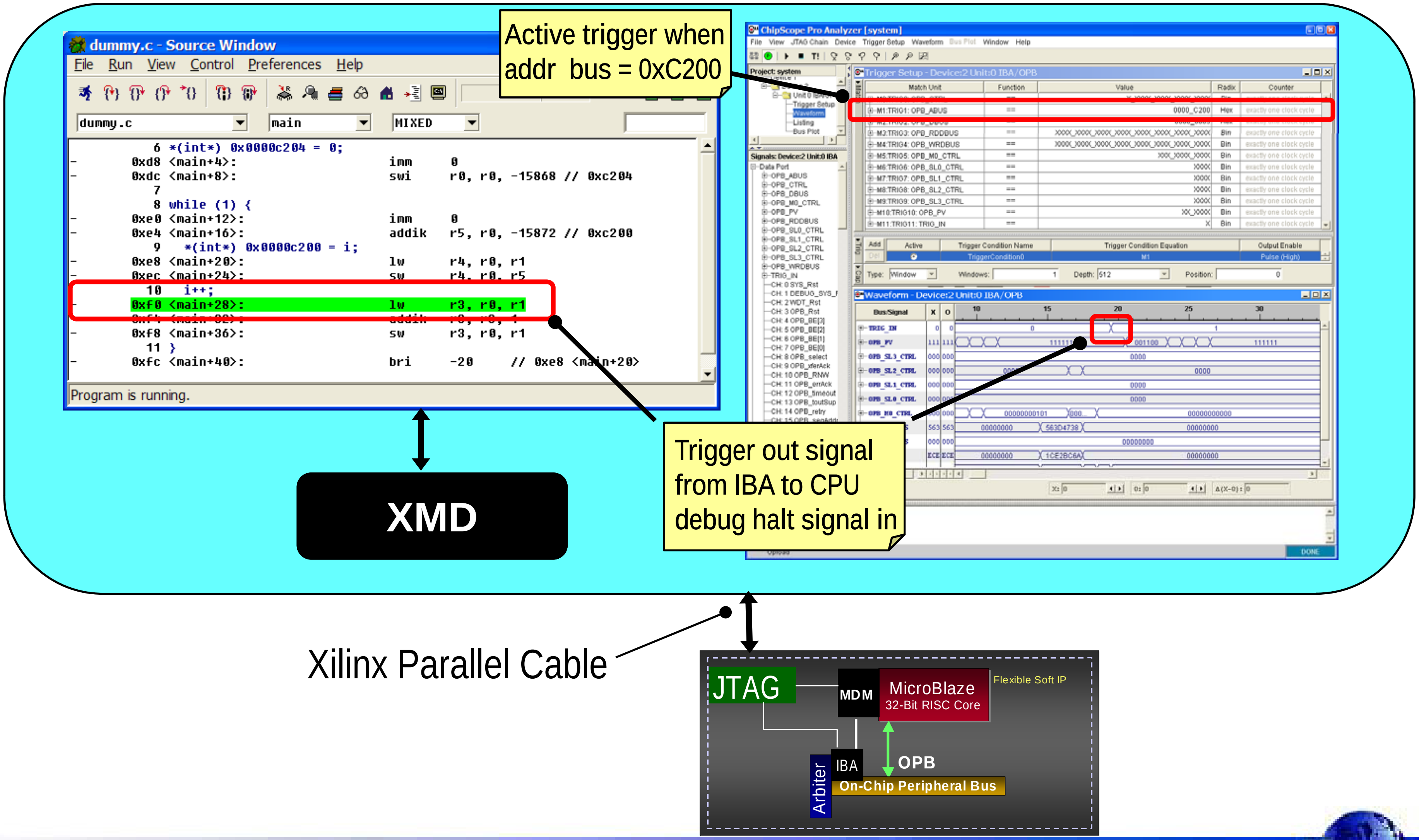
- Java-based application development environment
- Based on the open-source effort by the Eclipse Consortium\
- Feature-rich C/C++ code editor and compilation environment
- Project management
- Application build configuration and automatic Makefile generation
- Error Navigation
- Well-integrated environment for seamless debugging of embedded targets
- Source code version control

Debugging

Introduction

- Debugging is an integral part of embedded systems development
- The debugging process is defined as testing, stabilizing, localizing, and correcting errors
- Two methods of debugging
  - Hardware debugging
    - via a logic probe or logic analyzer such as Chipscope
    - Via a simulator such as Modelsim or NCSim
  - Software debugging
    - On target via xmd using jtagppc, MDM, xmdstub, or directly to PPC. Optionally use GDB.
    - Software simulation using xmd and optionally GDB
    - Virtual Platform
    - Third Party Debugger

Simultaneous HW/SW Debug



Active trigger when addr bus = 0xC200

Trigger out signal from IBA to CPU debug halt signal in

Xilinx Parallel Cable

```
        6 *(int*) 0x0000c204 = 0;
-     0xd8 <main+4>:                imm    0
-     0xdc <main+8>:                swi    r0, r0, -15868 // 0xc204
        7
        8 while (1) {
-     0xe0 <main+12>:               imm    0
-     0xe4 <main+16>:               addik  r5, r0, -15872 // 0xc200
        9   *(int*) 0x0000c200 = i;
-     0xe8 <main+20>:               lw     r4, r0, r1
-     0xec <main+24>:               sw     r4, r0, r5
       10   i++;
-     0xf0 <main+28>:               lw     r3, r0, r1
-     0xf4 <main+32>:               addik  r3, r3, 1
-     0xf8 <main+36>:               sw     r3, r0, r1
       11 }
-     0xfc <main+40>:               bri    -20     // 0xe8 <main+20>
```

Program is running.

Match Unit	Function	Value	Radix	Counter
M0:TRIG0: OPB_CTRL	==	X_XXXX_XXXX_XXXX_XXXX	Bin	exactly one clock cycle
M1:TRIG1: OPB_ABUS	==	0000_C200	Hex	exactly one clock cycle
M2:TRIG2: OPB_DBUS	==	0000_0003	Hex	exactly one clock cycle
M3:TRIG3: OPB_RDDBUS	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	exactly one clock cycle
M4:TRIG4: OPB_WRDBUS	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	exactly one clock cycle
M5:TRIG5: OPB_M0_CTRL	==	XXX_XXXX_XXXX	Bin	exactly one clock cycle
M6:TRIG6: OPB_SL0_CTRL	==	XXXX	Bin	exactly one clock cycle
M7:TRIG7: OPB_SL1_CTRL	==	XXXX	Bin	exactly one clock cycle
M8:TRIG8: OPB_SL2_CTRL	==	XXXX	Bin	exactly one clock cycle
M9:TRIG9: OPB_SL3_CTRL	==	XXXX	Bin	exactly one clock cycle
M10:TRIG10: OPB_PV	==	XX_XXXX	Bin	exactly one clock cycle
M11:TRIG11: TRIG_IN	==	X	Bin	exactly one clock cycle

Active	Trigger Condition Name	Trigger Condition Equation	Output Enable
●	TriggerCondition0	M1	Pulse (High)

Type: Window Windows: 1 Depth: 512 Position: 0

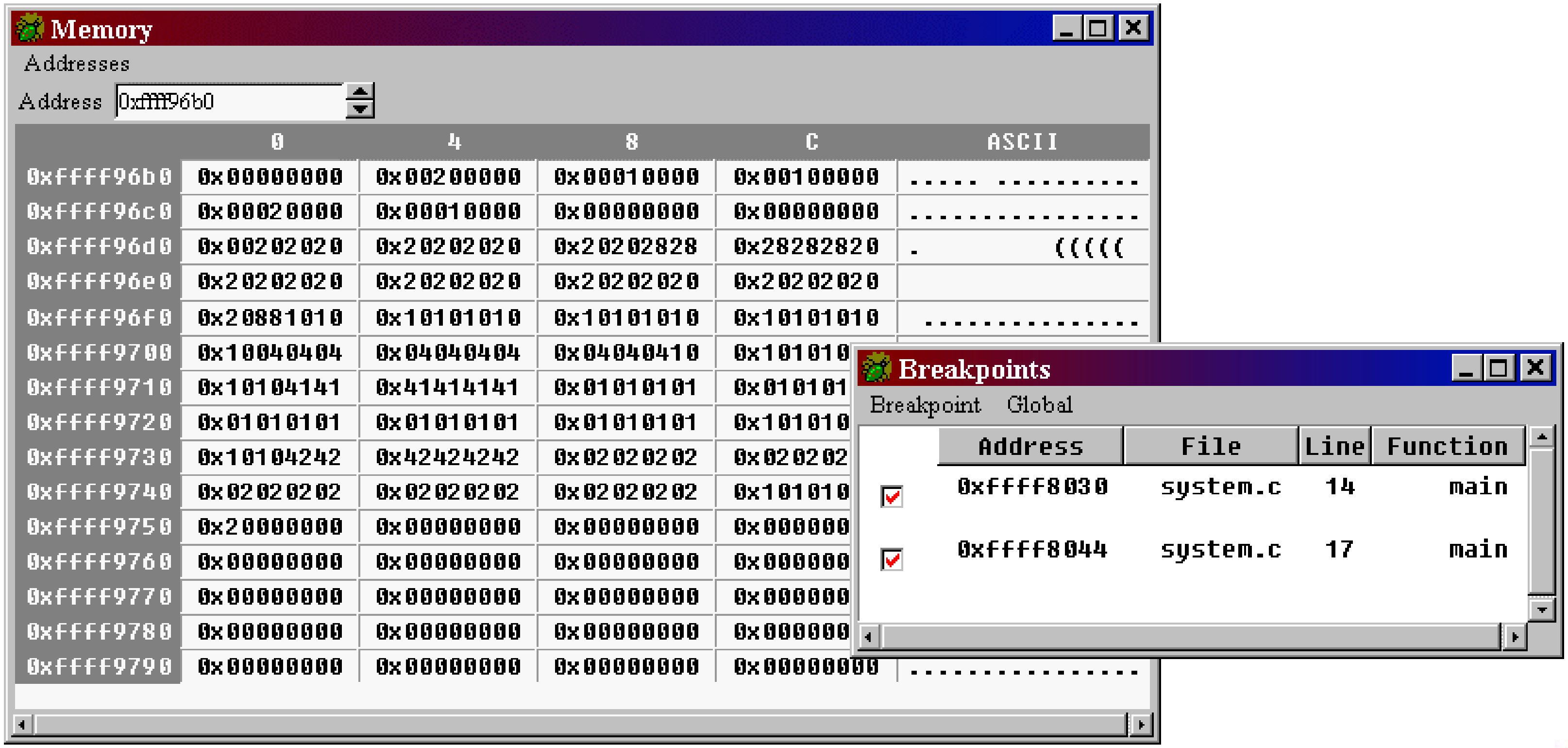
For Academic Use Only

GDB Functionality

- GDB is a source-level debugger that helps you debug your program:
  - Start your program
  - Set breakpoints (make your program stop on specified conditions)
  - Examine what has happened, when your program encounters breakpoints
    - Registers
    - Memory
    - Stack
    - Variables
    - Expressions
  - Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another
- You can use GDB to debug programs written in C and C++

GDB Functionality

- Breakpoints can be enabled or disabled
- To change any memory value, simply double-click in a memory field



	0	4	8	C	ASCII
0xffff96b0	0x00000000	0x00200000	0x00010000	0x00100000	....
0xffff96c0	0x00020000	0x00010000	0x00000000	0x00000000	....
0xffff96d0	0x00202020	0x20202020	0x20202828	0x28282820	. (((((
0xffff96e0	0x20202020	0x20202020	0x20202020	0x20202020	
0xffff96f0	0x20881010	0x10101010	0x10101010	0x10101010	....
0xffff9700	0x10040404	0x04040404	0x04040410	0x101010	
0xffff9710	0x10104141	0x41414141	0x01010101	0x010101	
0xffff9720	0x01010101	0x01010101	0x01010101	0x101010	
0xffff9730	0x10104242	0x42424242	0x02020202	0x020202	
0xffff9740	0x02020202	0x02020202	0x02020202	0x101010	
0xffff9750	0x20000000	0x00000000	0x00000000	0x000000	
0xffff9760	0x00000000	0x00000000	0x00000000	0x000000	
0xffff9770	0x00000000	0x00000000	0x00000000	0x000000	
0xffff9780	0x00000000	0x00000000	0x00000000	0x000000	
0xffff9790	0x00000000	0x00000000	0x00000000	0x00000000	....

	Address	File	Line	Function
☑	0xffff8030	system.c	14	main
☑	0xffff8044	system.c	17	main

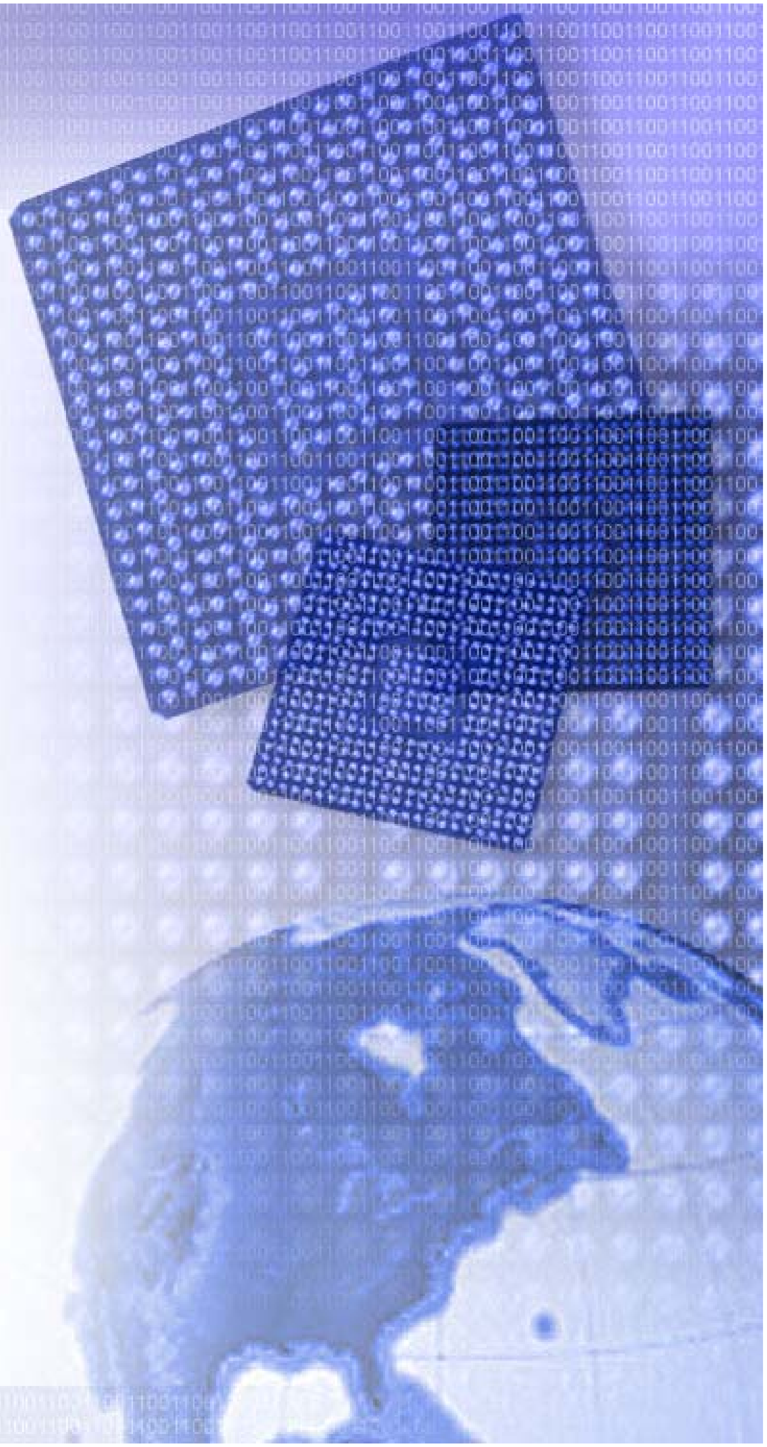
SimGen

- The Simulation Model Generation tool (SimGen) generates and configures various simulation models for the specified hardware
- SimGen will generate simulation models by using a Microprocessor Hardware Specification (MHS) file
- SimGen searches for input files in the following directories located in the project directory
  - <project_directory>/hdl/
    - system_name.[vhd|v]
    - peripheral_wrapper.[vhd|v]
  - <project_directory>/implementation/ (if any of the peripherals are black-box)
    - peripheral_wrapper.ngc
    - system_name.ngc
    - system_name.ncd

SimGen Pcores and Simulation Libraries

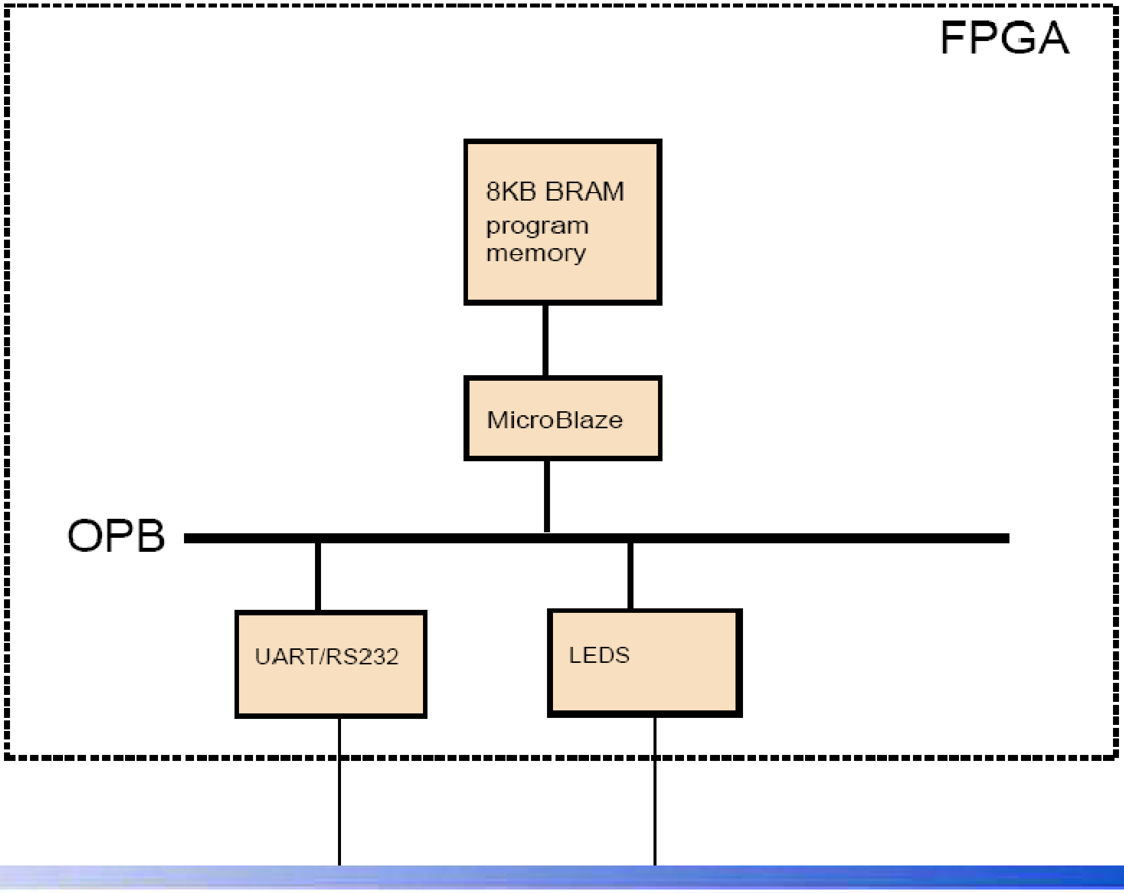
- SimGen will read user IP from pcores
  - <project_directory>/pcores/
  - <Peripheral Repository Directory>/<Library Name>/pcores
- HDL or Netlist will be used from pcore
- If the .MPD file does not exist in pcore, SimGen will check for MPD and PAO in the EDK installation
- Precompiled Simulation Libraries:
  - IP in EDK install will use precompiled simulation libraries
  - Project pcores will not use the precompiled simulation libraries
  - Peripheral Repository Directory is determined based on setting when compiling simulation libraries

Frivillig prosjekt - laboppgave 4 H07



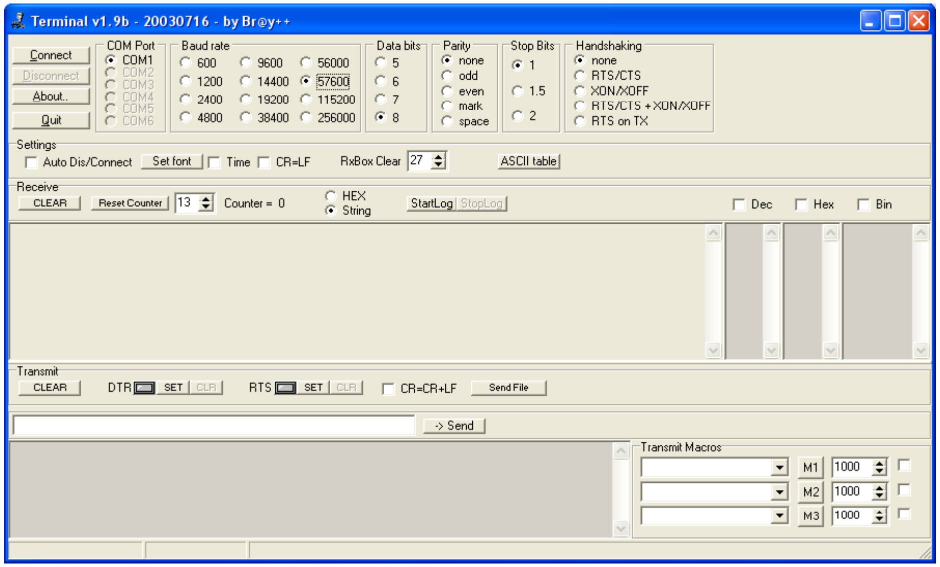
Sette opp SoC

- Først tutorial for å sette seg inn i EDK



Seriell kommunikasjon

- En UART-IP kan sende og motta bytes over seriekabel (RS-232)
- Greit å bruke for testing og debugging av systemet
  - UARTen kan settes opp som STDIN/OUT, bruk xil_printf
  - Kan være nødvendig å droppe xil_printf i det ferdige spillet pga. kodestørrelse
- Bray's Terminal



Terminal v1.9b - 20030716 - by Br@y++

COM Port: ● COM1 ○ COM2 ○ COM3 ○ COM4 ○ COM5 ○ COM6

Baud rate: ○ 600 ○ 1200 ○ 2400 ○ 4800 ○ 9600 ○ 14400 ○ 19200 ○ 38400 ○ 56000 ● 57600 ○ 115200 ○ 256000

Data bits: ○ 5 ○ 6 ○ 7 ● 8

Parity: ● none ○ odd ○ even ○ mark ○ space

Stop Bits: ● 1 ○ 1.5 ○ 2

Handshaking: ● none ○ RTS/CTS ○ XON/XOFF ○ RTS/CTS + XON/XOFF ○ RTS on TX

Settings: □ Auto Dis/Connect □ Time □ CR=LF RxBox Clear 27

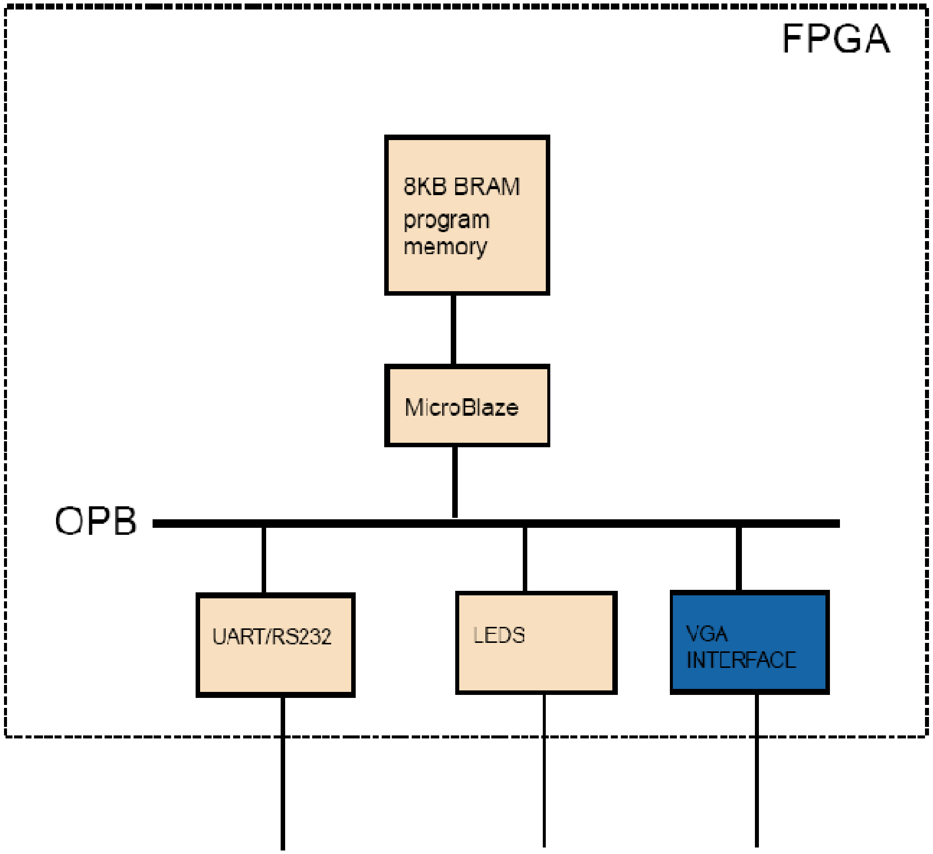
Receive: 13 Counter = 0 ○ HEX ● String □ Dec □ Hex □ Bin

Transmit: □ CR=CR+LF

Transmit Macros: M1 1000 □ M2 1000 □ M3 1000 □

VGA

- Ferdig kjerne skal kobles til systemet

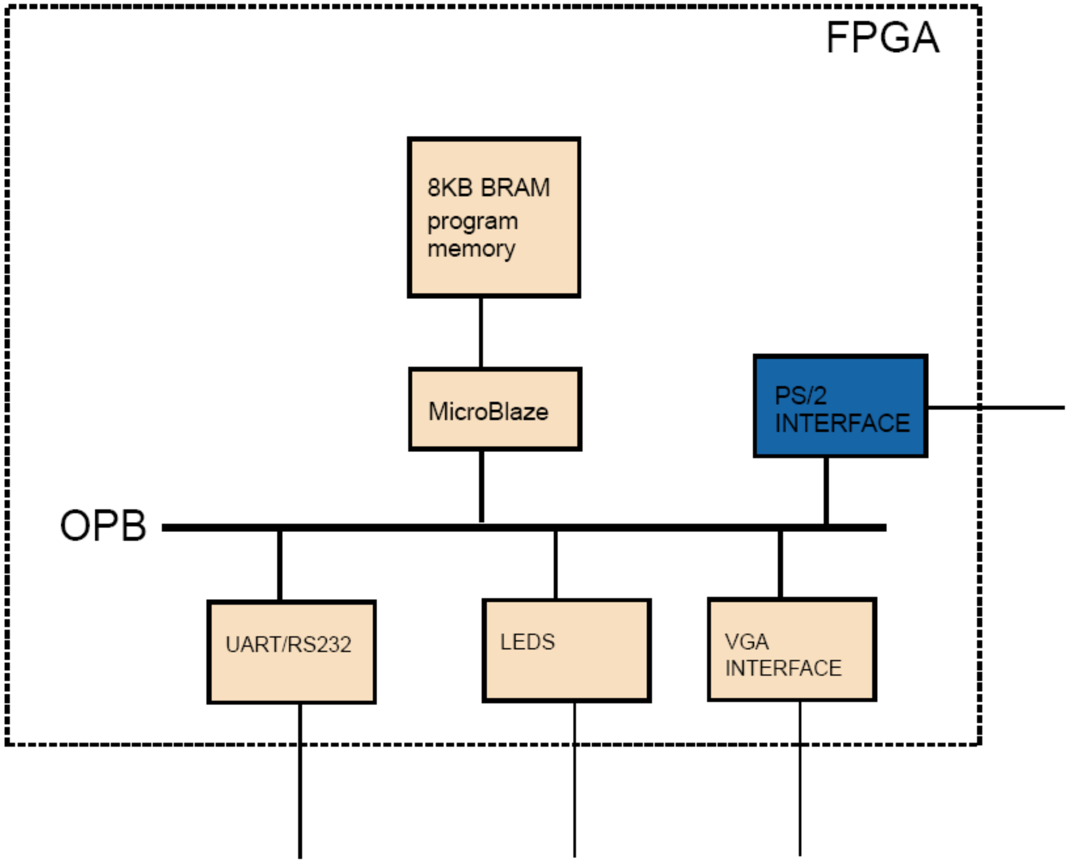


VGA(2)

- VGA-kjernen er i tegnmodus (character mode)
- Block RAM (BRAM) i FPGA-en inneholder data for tegn
- Man kan bare skrive tegn til skjermen, med en oppløsning på 100x75
- Det er mulig å trikse litt for å få bedre grafikk, ved å omdefinere tegnene
  - Dvs. forandre på innholdet i BRAM

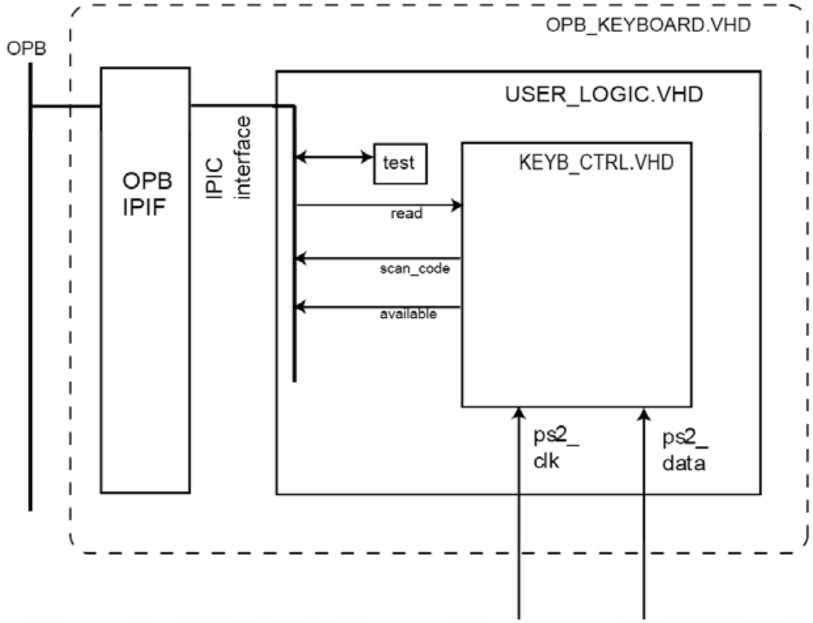
Tastatur

- Tastaturkjernen fra oppgave 3 skal tilpasses systemet



Tastatur(2)

- Tastaturet skal kobles til OPB-bussen vha. en IPIF-kjerne
- Man må sjekke fra programmet om det har blitt trykket noen nye taster siden sist (polling)
  - Dette ligger i driveren (keyboard.c/h) som følger med



”TV”-spill

- Det skal lages et enkelt spill som bruker skjerm og tastatur
- Programmeres i C
- Trenger ikke å være avansert
- Det er bare 8KB minne tilgjengelig for spillet (i utgangspunktet)
  - Kan være vanskelig å overføre spill laget på andre platformer
  - Lag funksjoner du trenger selv, spar plass!

- Ekstraoppgave: bruke syvsegmentsdisplay, lyd eller annet.
- Greiest å jobbe ut fra eksempelet som følger med oppgaven

Rammeverk for spill

```
 while(1) {
        int key;
        key=keyboard_status(0x1C);              //key A

        if(key==1)
            pos++;

        scr_draw_vert_line(pos-1,10,'h',4);     //(clears previous line)
        scr_draw_vert_line(pos,10,'b',4);       //draw a line at pos

        keyboard_wait_and_poll(70000);          //wait and update keys
}
```