Universidad de Alcalá

ELECTRONICS DEPARTMENT

ADVANCED DIGITAL ELECTRONIC SYSTEMS

SOFTWARE DEVELOPMENT

Raúl Mateos Gil

OFFICIAL MASTER IN ADVANCED ELECTRONIC SYSTEMS.

INTELLIGENT SYSTEMS

OUTLINE

- **INTRODUCTION**
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

SOFTWARE DEVELOPMENT FOR
DESKTOP COMPUTERS vs EMBEDDED SYSTEMS

DESKTOP COMPUTERS

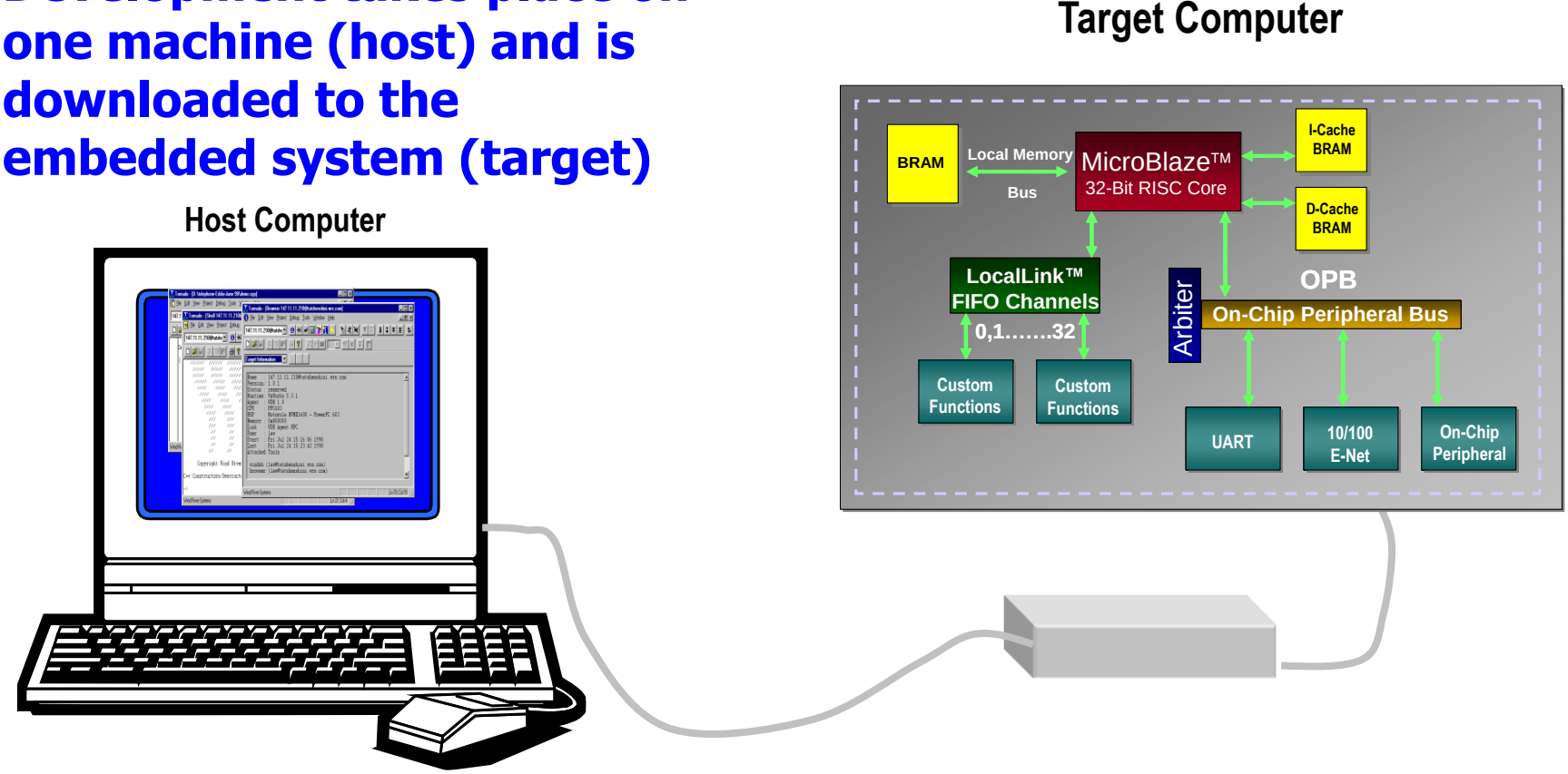
- Desktop development: written, debugged, and run on the same machine
- OS loads the program into the memory when the program has been requested to run
- Address resolution takes place at the time of loading by a program called the loader
  - The loader is included in the OS

EMBEDDED SYSTEMS

- The programmer glues into one executable file called ELF
  - Boot code, application code, RTOS, and ISRs
  - Address resolution takes place during the gluing stage
- The executable file is downloaded into the target system through different methods
  - Ethernet, serial, JTAG, BDM, ROM programmer

EMBEDDED vs DESKTOP

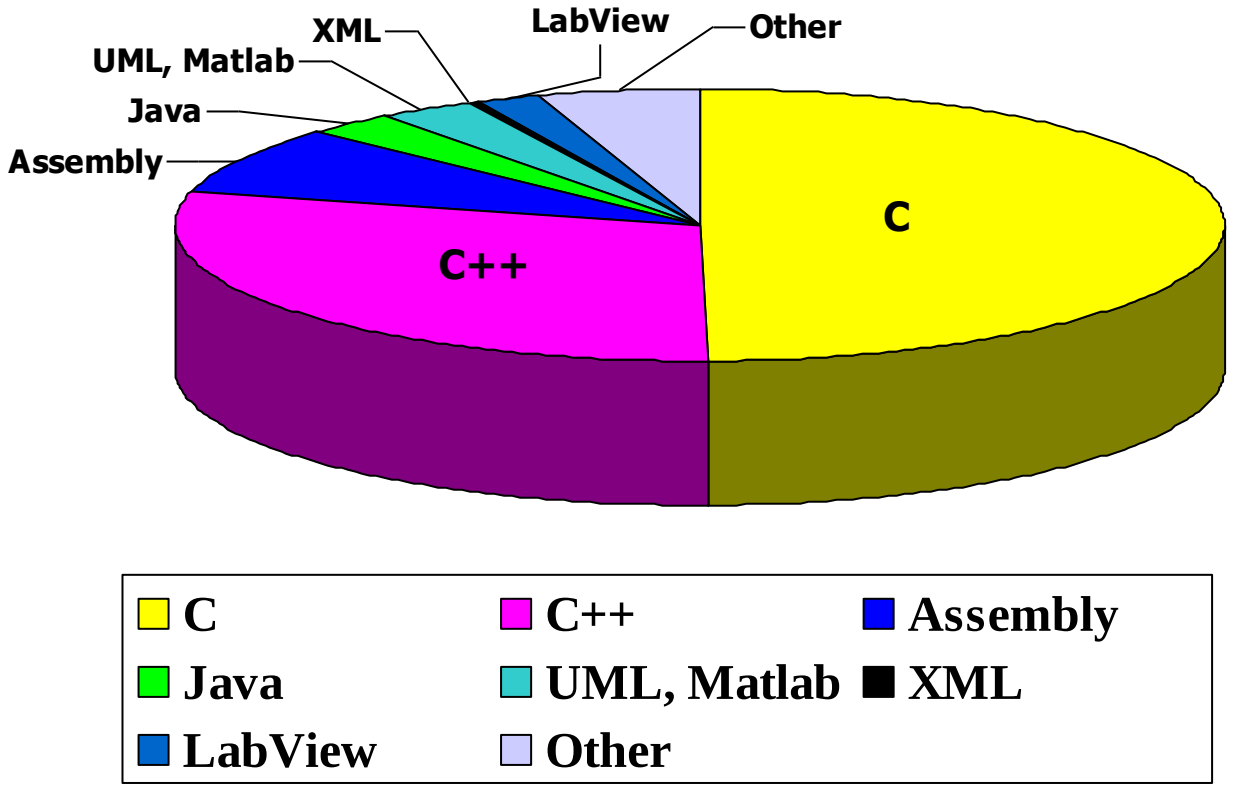
- **Development takes place on one machine (host) and is downloaded to the embedded system (target)**



A cross-compiler is run on the host

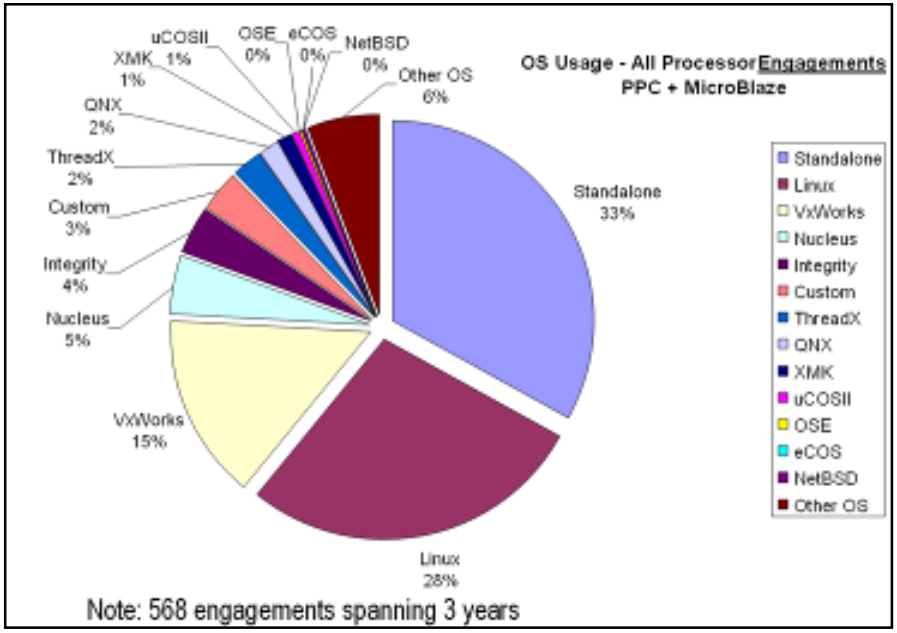
- Different set of problems
  - Unique hardware for every design
  - Reliability
  - Real-time response requirement (sometimes)
  - RTOS versus OS
  - Code compactness
  - High-level languages and assembly

PROGRAMING LANGUAGE USAGE



Source: 2006 Embedded Systems Design State of Embedded Market Survey

- Several alternatives
  - Standalone (No operating system)
  - Embedded Operating System (eOS)
    - General Purpose Operating Systems (GPOS)
    - Real Time Operating System (RTOS)



SOFTWARE PLATFORMS

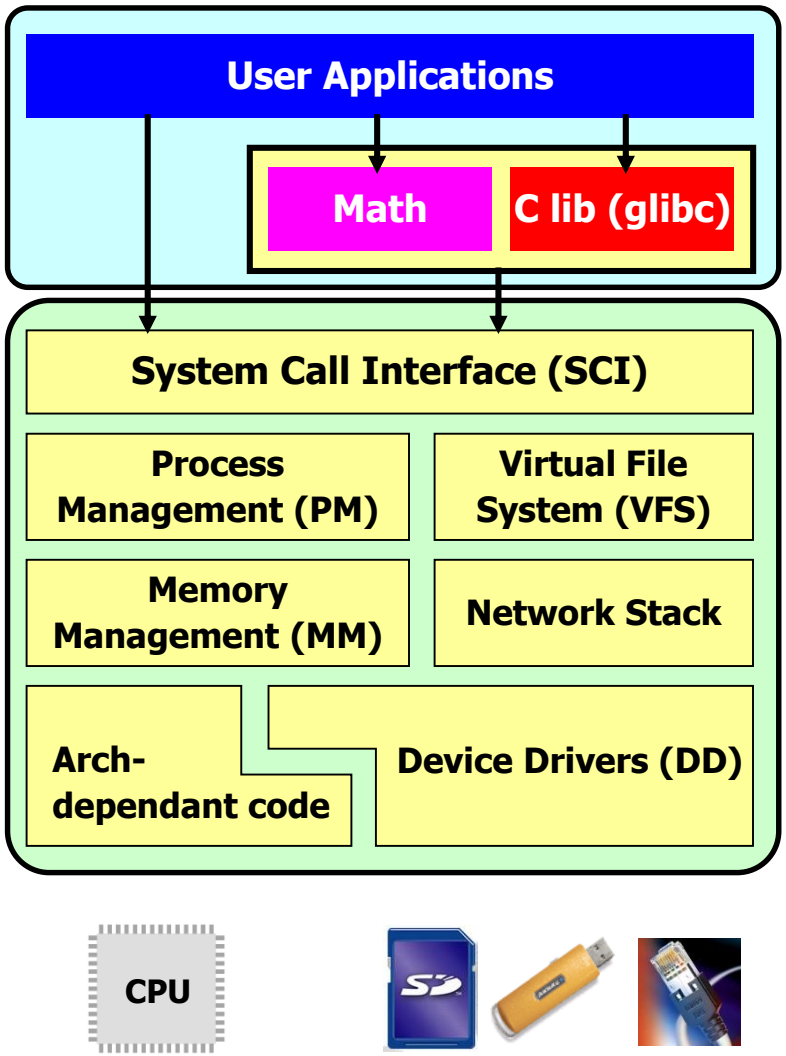
Embedded RTOS

- Multi-tasking
- Minimize latency
- Generally pre-emptive
  - VxWorks, ThreadX, etc.
- Hard and soft real time deadlines
  - execution deterministic
- Minimize CPU loading

Embedded OS

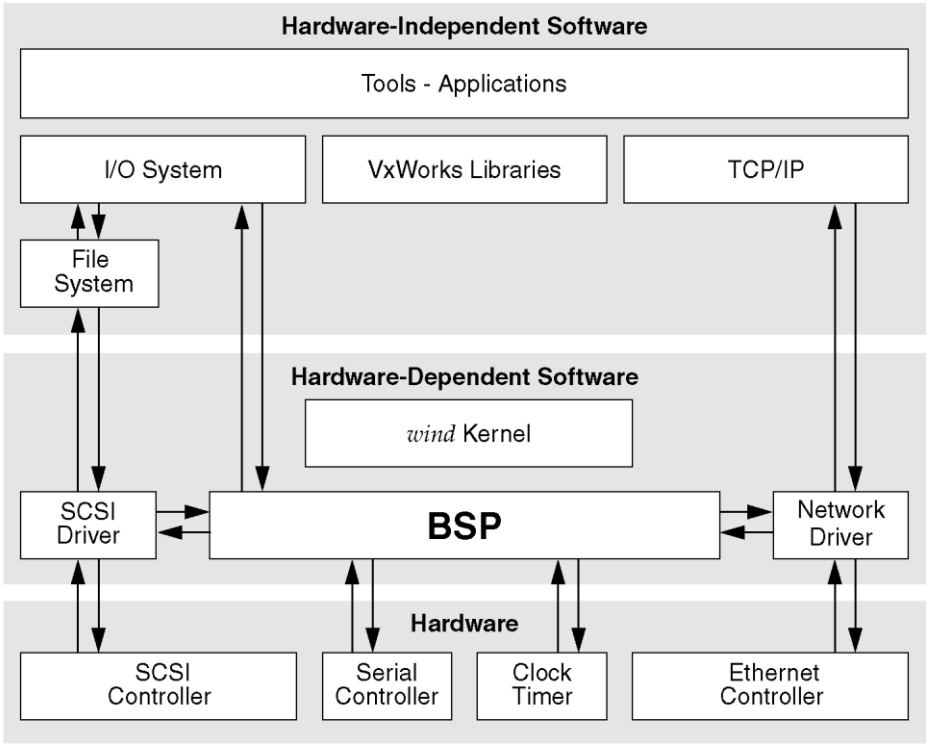
- Multi-tasking
- Maximize throughput
- Pre-emption not critical
  - Linux 2.4, Clinux
- Soft real time deadlines
  - execution less predictable
- Maximize CPU loading

GENERAL PURPOSE eOS: LINUX



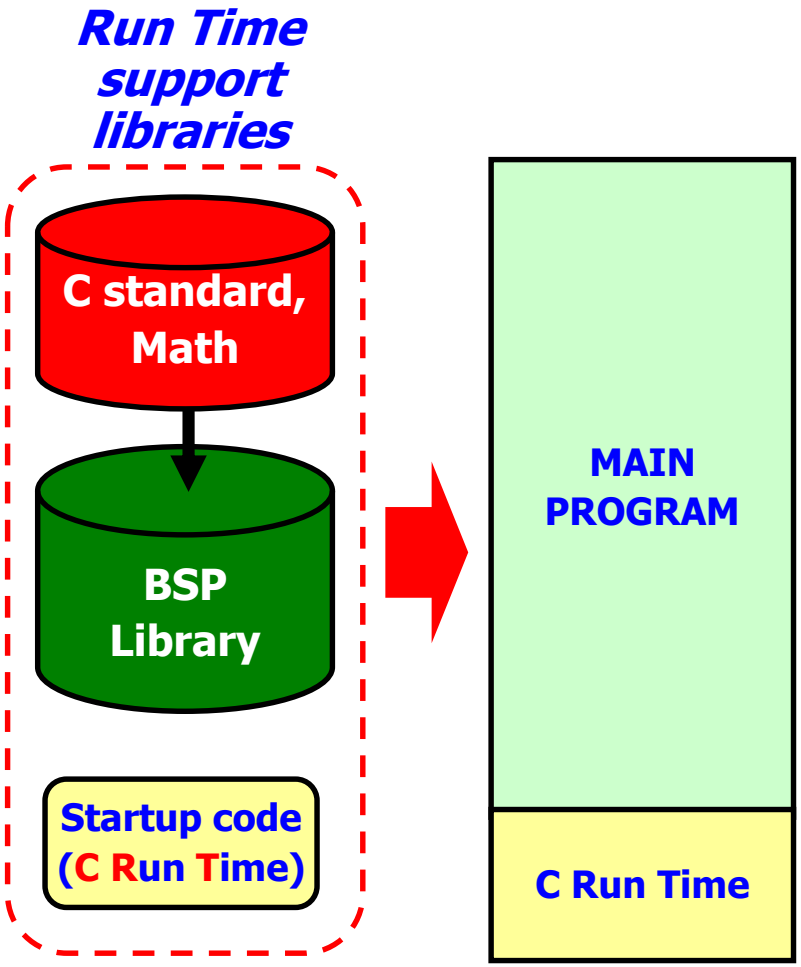
- Rich set of toolsets and utilities is available
- Multitasking
- Memory protection
- Portability

REAL TIME eOS: VxWorks



- Modular design
  - Scalability
- Microkernel provides minimal functionality
  - Process and memory management
- Other modules for the rest of the functionality
  - Client-server approach
- Hardware specific functionality in BSP libraries

STANDALONE PLATFORM



- Valid approach for very simple systems
  - Difficult to scale
  - One single task
- Monolithic structure
- Lightweight runtime support libraries

We will focus on this alternative

BOARD SUPPORT PACKAGES

- A Board Support Package (BSP) is the lowest layer of software modules used to access processor specific functions and custom hardware on the embedded system for a given target O/S
  - Virtualizes the platform hardware so that the different drivers can be ported easily on any hardware
  - Hides the board- and CPU-specific details from the rest of the OS
- The BSP has two components:
  - The microprocessor support
    - Interrupts, cache, endiannes
  - Board specific routines
    - Boot loader support
    - Memory map support
    - System timers
    - Interrupt controller support
    - Real-Time Clock (RTC)

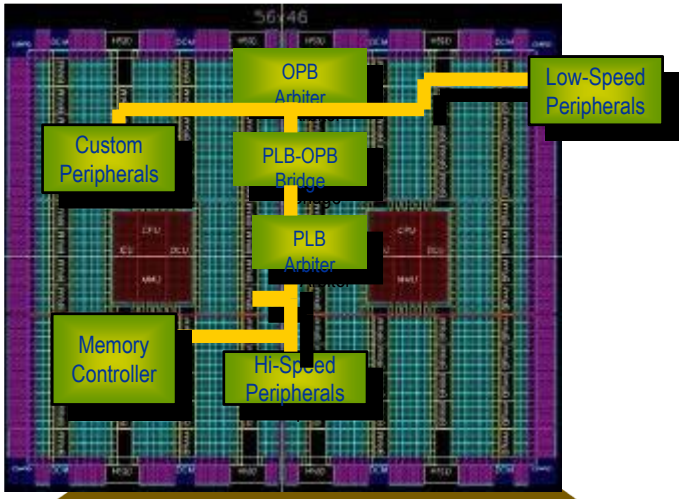
BOARD SUPPORT PACKAGES

Classical Embedded Platforms

- Fixed peripherals
- Fixed address maps
- Fixed BSP

Different

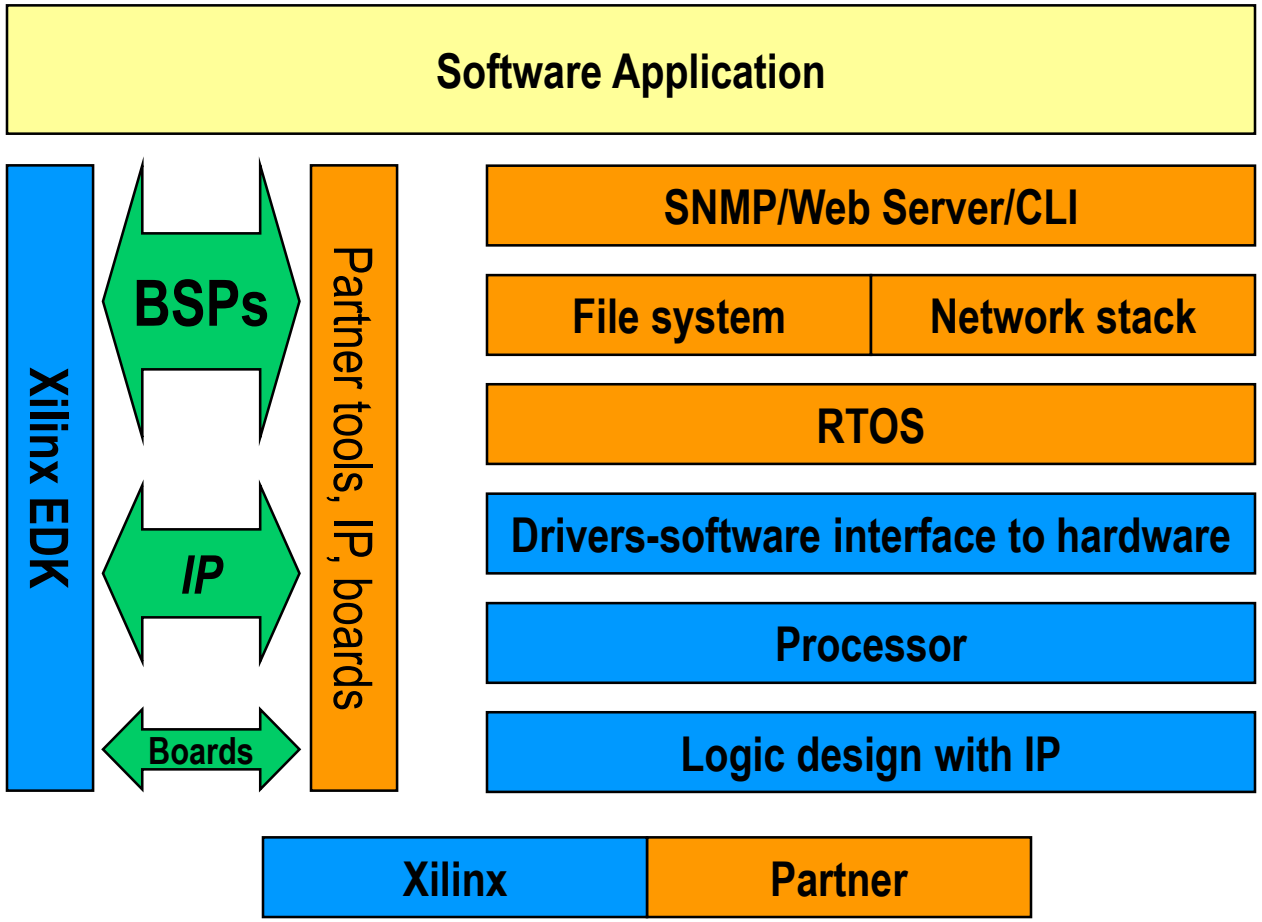
Platform FPGA



- Design dependent peripherals
- Each board is unique and custom
- Need efficient custom BSP creation

In an environment where hardware is defined in a programmable System-on-Chip (SoC), hardware changes can come about much more rapidly, making it difficult for the BSP to remain current with the revisions in hardware

BOARD SUPPORT PACKAGES

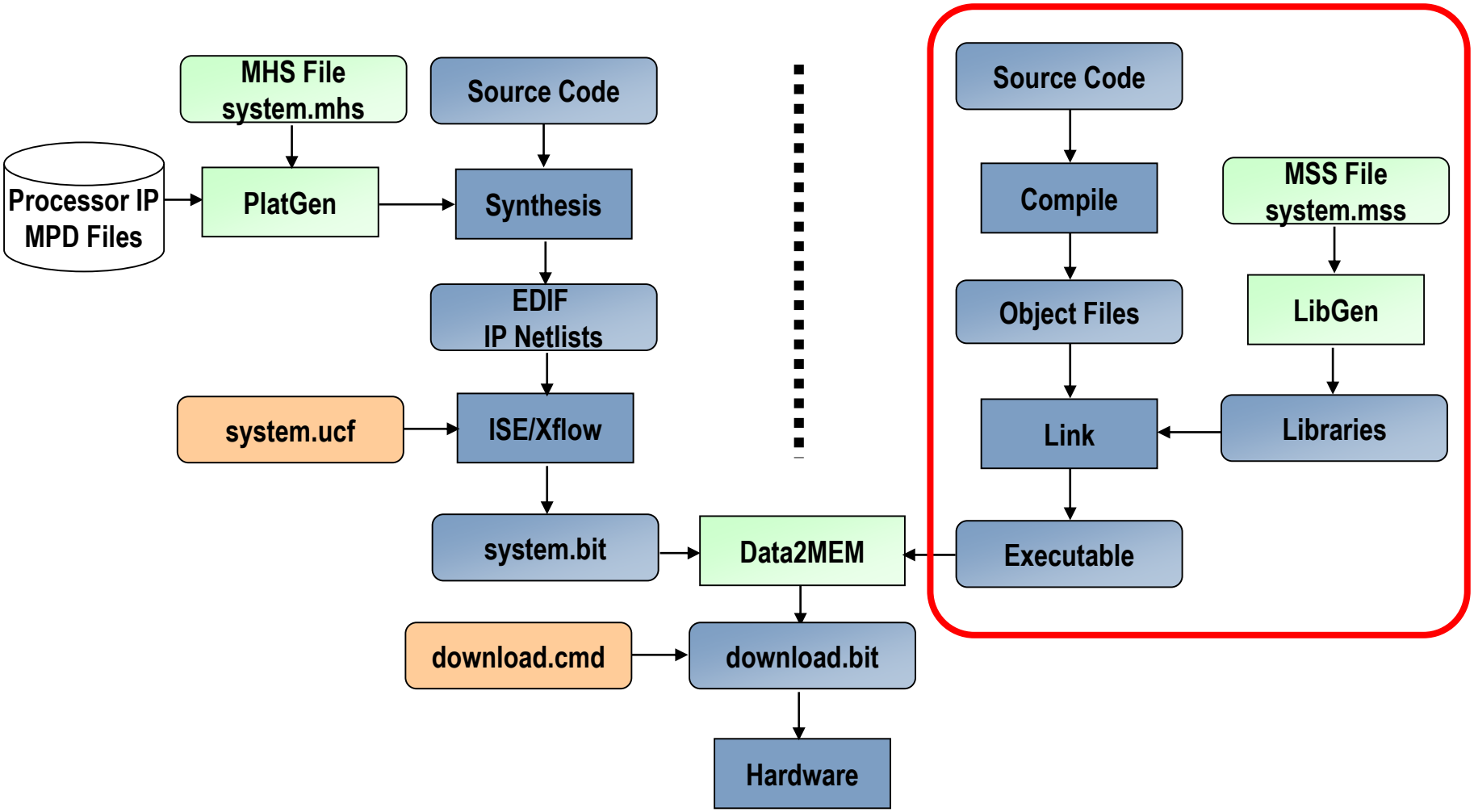


EDK provides a process called Automatic BSP Generation that tailors a BSP according to the current hardware configuration of the FPGA

OUTLINE

- INTRODUCTION
- **SOFTWARE DESIGN FLOW**
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

SW DESIGN FLOW



SW DESIGN ENVIRONMENT

- The Library Generator (LibGen) utility generates the necessary libraries and drivers for the embedded system
- 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
  - other related software features
- The MSS file is generated by XPS by using the software settings specified

LibGen configures libraries and device drivers

LibGen Generated Directories

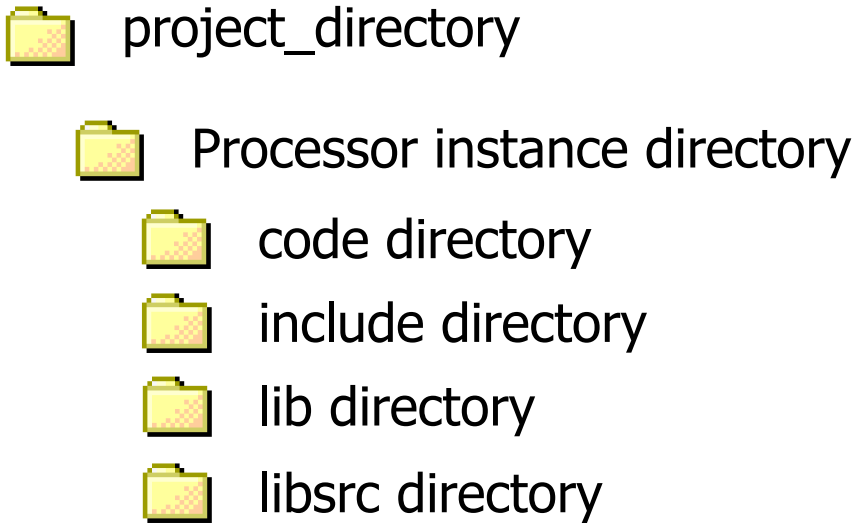
- project_directory
  - Processor instance directory
    - code directory
    - include directory
    - lib directory
    - libsrc directory

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
  - Creates `xmdstub.elf` for MB here
- 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

Microblaze instance name in MHS file (e.g. microblaze_0)

LibGen Generated Directories



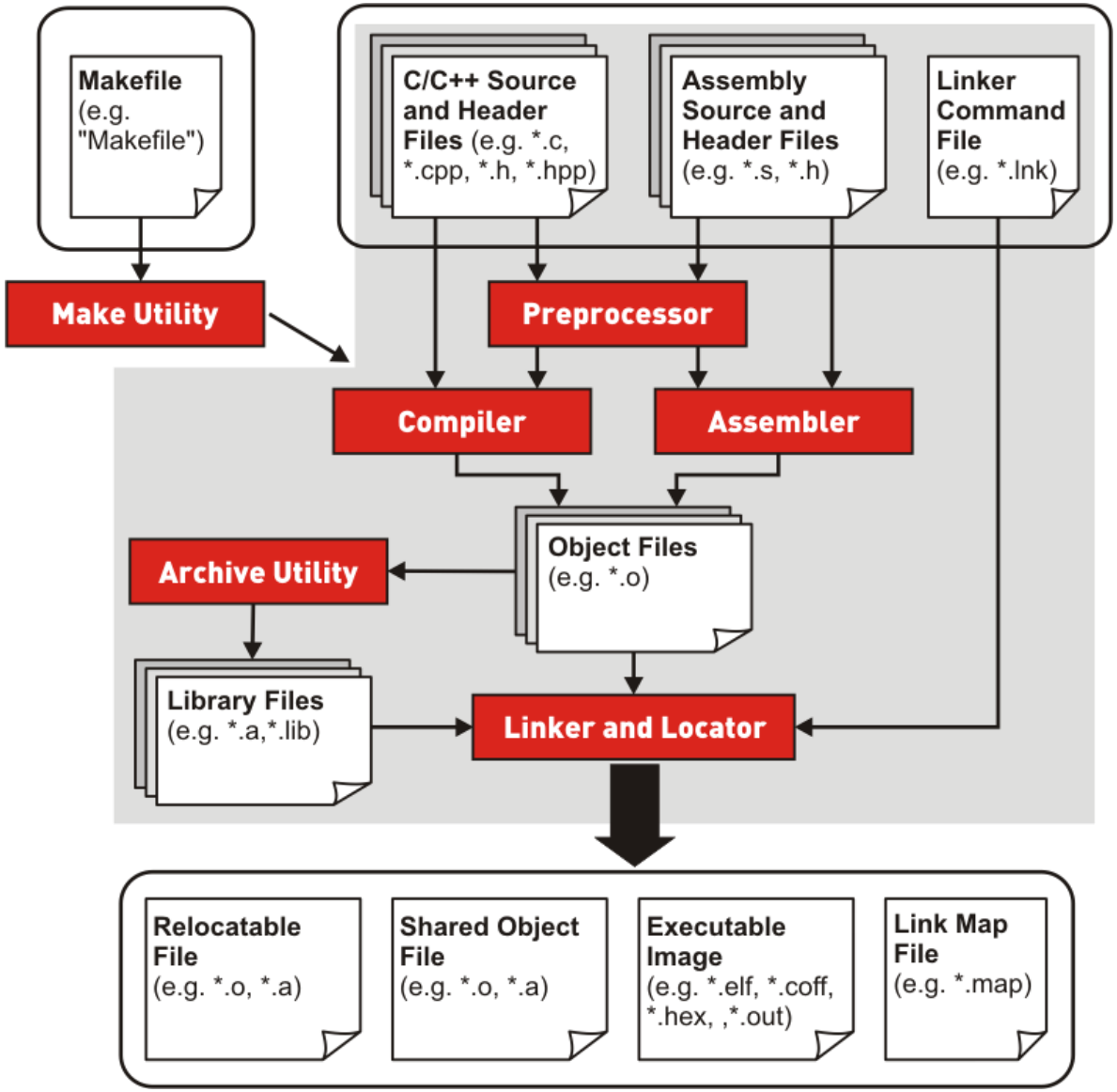
Note: The processor instance directories content is overwritten every time LibGen is run

- `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
- `lib` directory
  - System support libraries:
    - `libc.a`
    - `libm.a`
    - `libxil.a`

OUTLINE

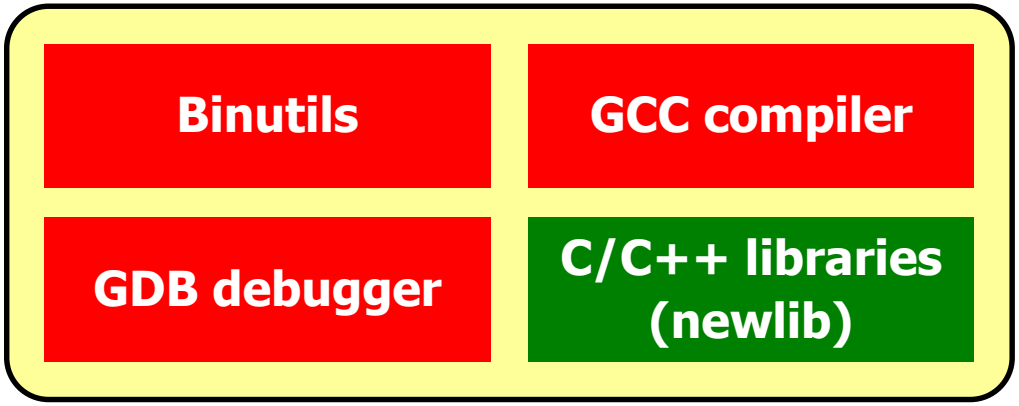
- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - **GNU TOOLCHAIN**
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

SOFTWARE TOOLCHAIN



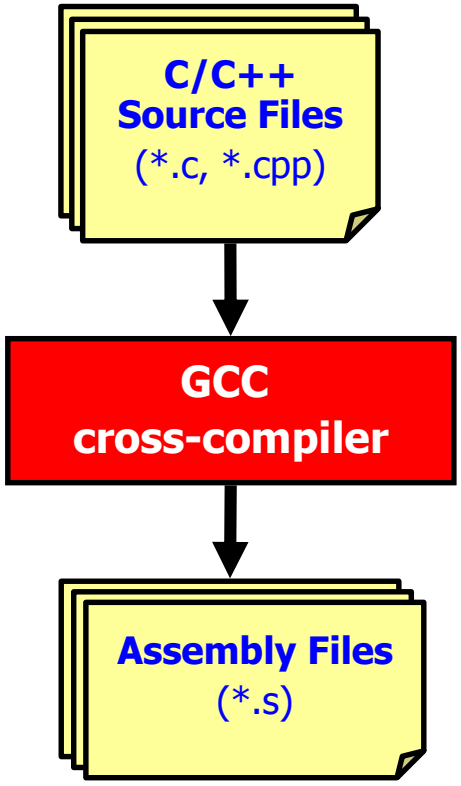
SOFTWARE TOOLCHAIN

- Microblaze uses the GNU toolchain as software development tools
  - mb-gcc: C/C++ Compiler
  - mb-as: Assembler
  - mb-ld: Linker
  - mb-ar: Archiver (Library management)
  - mb-objdump: Provides info about object files
  - mb-readelf: Provides info about elf (executable) files
  - etc.



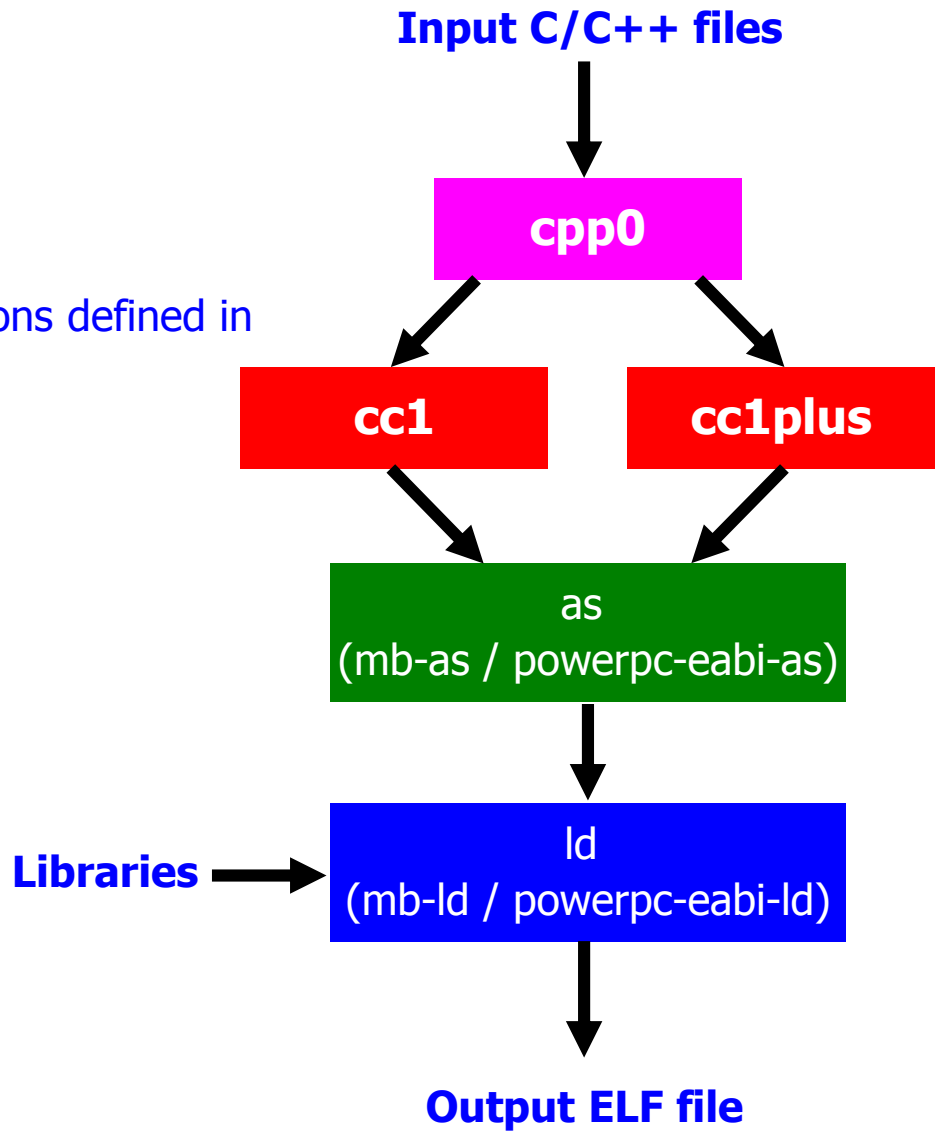
GNU COMPILER: GCC

- 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



GNU COMPILER: GCC

- Calls four different executables
  - Preprocessor (cpp0)
    - Analyzes and expands directives
    - Replaces all macros with definitions defined in the source and header files
  - Language specific c-compiler
    - cc1 C-programming language
    - cc1plus C++ language
  - Assembler
    - mb-as (MicroBlaze)
    - powerpc-eabi-as (PowerPC)
  - Linker and loader
    - mb-ld (MicroBlaze)
    - powerpc-eabi-ld (PowerPC)



MICROBLAZE SPECIFIC OPTIONS:

- Processor feature selection options:
  - -mcpu=vX.YY.Z
  - -mno-xl-soft-mul
  - -mxl-multiply-high
  - -mno-xl-multiply-high
  - -mxl-soft-mul
  - -mno-xl-soft-div
  - -mxl-soft-div
  - -mxl-barrel-shift
  - -mno-xl-barrel-shift
  - -mxl-pattern-compare
  - -mno-xl-pattern-compare
  - -mhard-float
  - -msoft-float

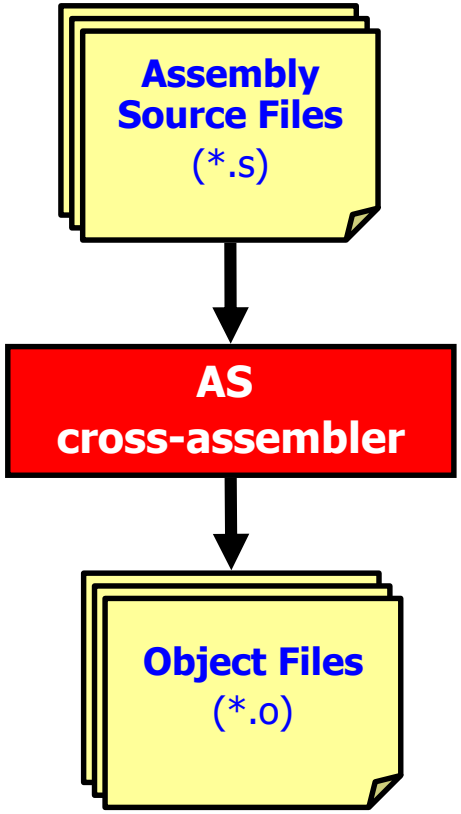
- General Program Options
  - -msmall-divides
  - -mxl-gp-opt
  - -mno-clearbss
  - -mxl-stack-check

- Application Execution Modes
  - -xl-mode-executable
  - -xl-mode-xmdstub
  - -xl-mode-bootstrap
  - -xl-mode-novectors

EDK automatically fixes the right options depending on MHS setting and compiler user dialogs

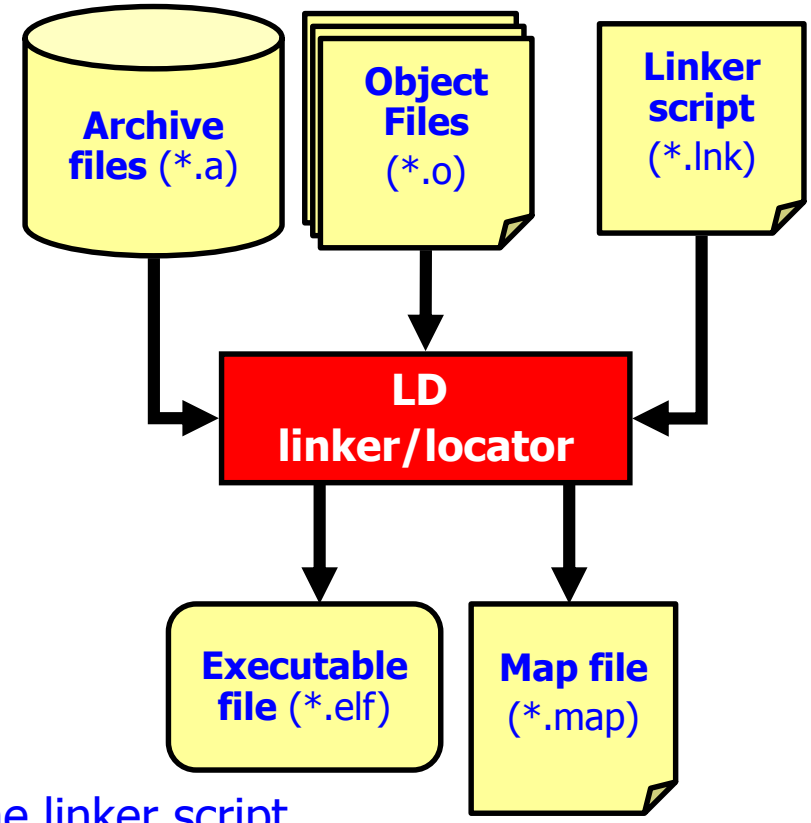
GNU ASSEMBLER: AS

- Input: Assembly language files
  - File extension: **.s**
- Output: Object code
  - File extension: **.o**
  - Contains
    - Assembled piece of code
    - Constant data
    - External references
    - Debugging information
- Typically, the compiler automatically calls the assembler
- It's possible to assemble a source file from gcc
  - Use `-Wa, xxx` to pass the xxx assembler option to mb-as from the gcc command line



GNU LINKER: LD

- Combines several object files and archive files, relocates their data and generates an executable file
- Inputs:
  - Several object files
  - Archived object files (library)
  - Linker script (mapfile)
- Output:
  - Executable image (.elf)
  - Mapping info file (.map)
- Command line options:
  - Have priority over those included in the linker script
  - Symbol definition: `-defsym symbol=expression`
- Use `-Wl,xxx` to set the xxx linker option from the GCC command line
  - `mb-gcc -Wl,defsym -Wl,_STACK_SIZE=0x800`



GNU BINARY UTILITIES: BINUTILS

- **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture
  - as, the assembler, that generates binary code from assembler source code
  - ld, the linker
  - ar, ranlib, to generate .a archives, used for libraries
  - objdump, readelf, size, nm, strings, to inspect binaries.
  - Very useful analysis tools !
  - strip, to strip useless parts of binaries in order to reduce their size

GNU BINARY UTILITIES: BINUTILS

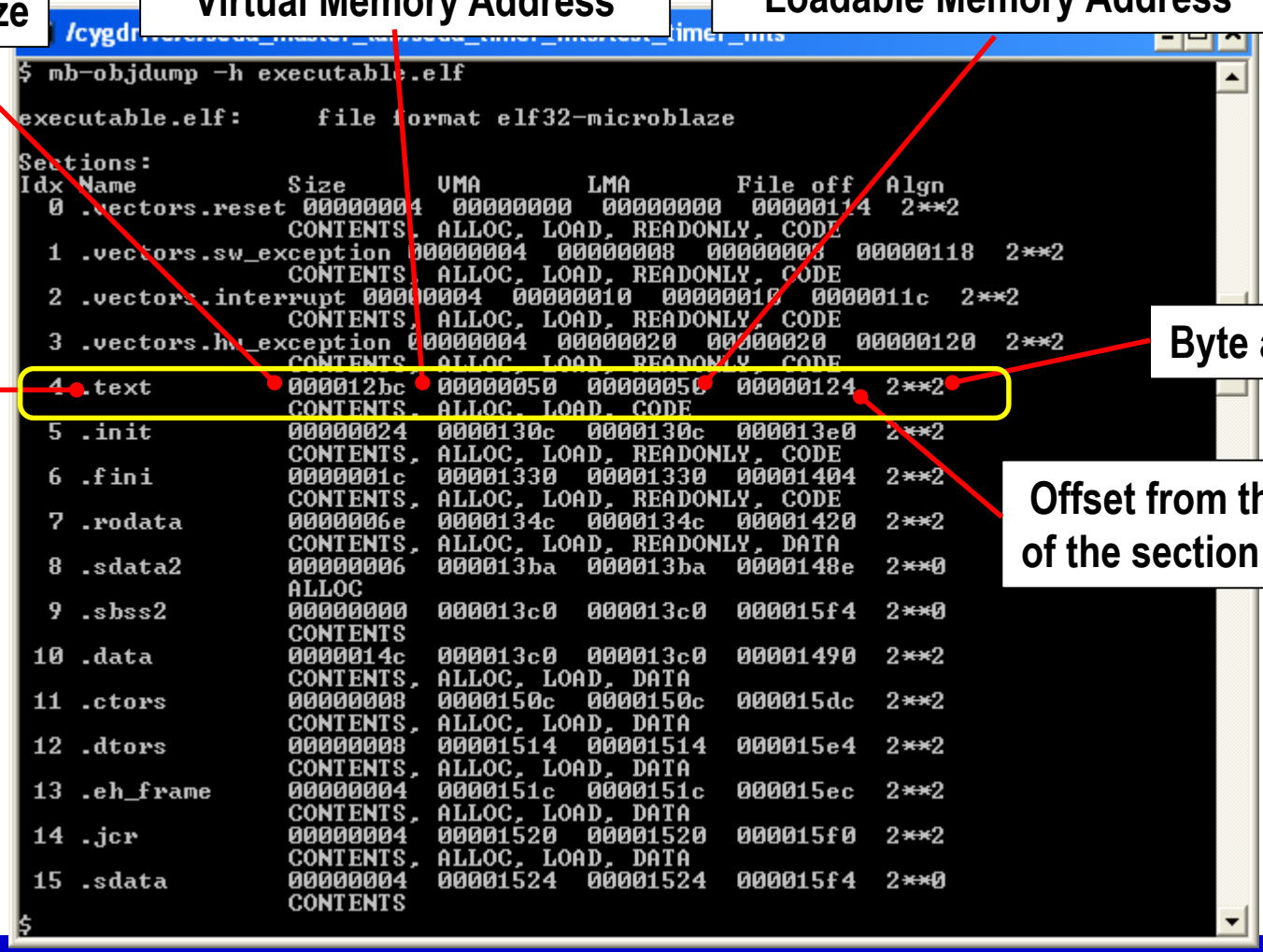
- AR Archiver
  - Library management
  - Create, modify, and extract object modules from libraries
  - Used in EDK to combine the object files of the Board Support Package (BSP) in a library
  - Used in EDK to extract object files from different libraries

- Object Dump
  - Display information from object files and executables
    - Header information, memory map
    - Data
    - Disassemble code
  - Two versions:
    - powerpc-eabi-objdump
    - mb-objdump

MICROBLAZE OBJECT DUMP

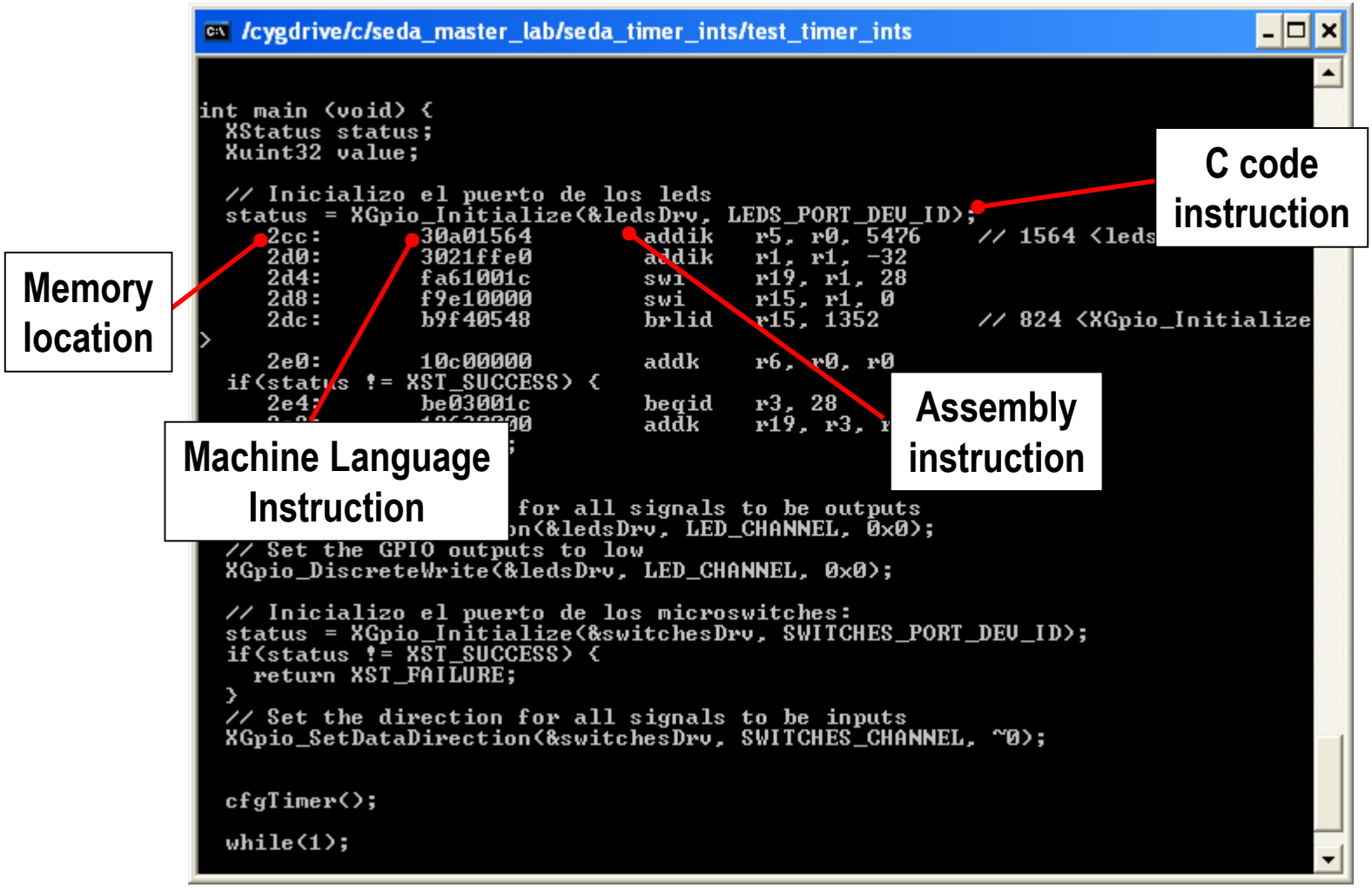
- Display summary information from the section headers:

```
mb-objdump -h executable.elf
```



- Dumping the source and assembly code:

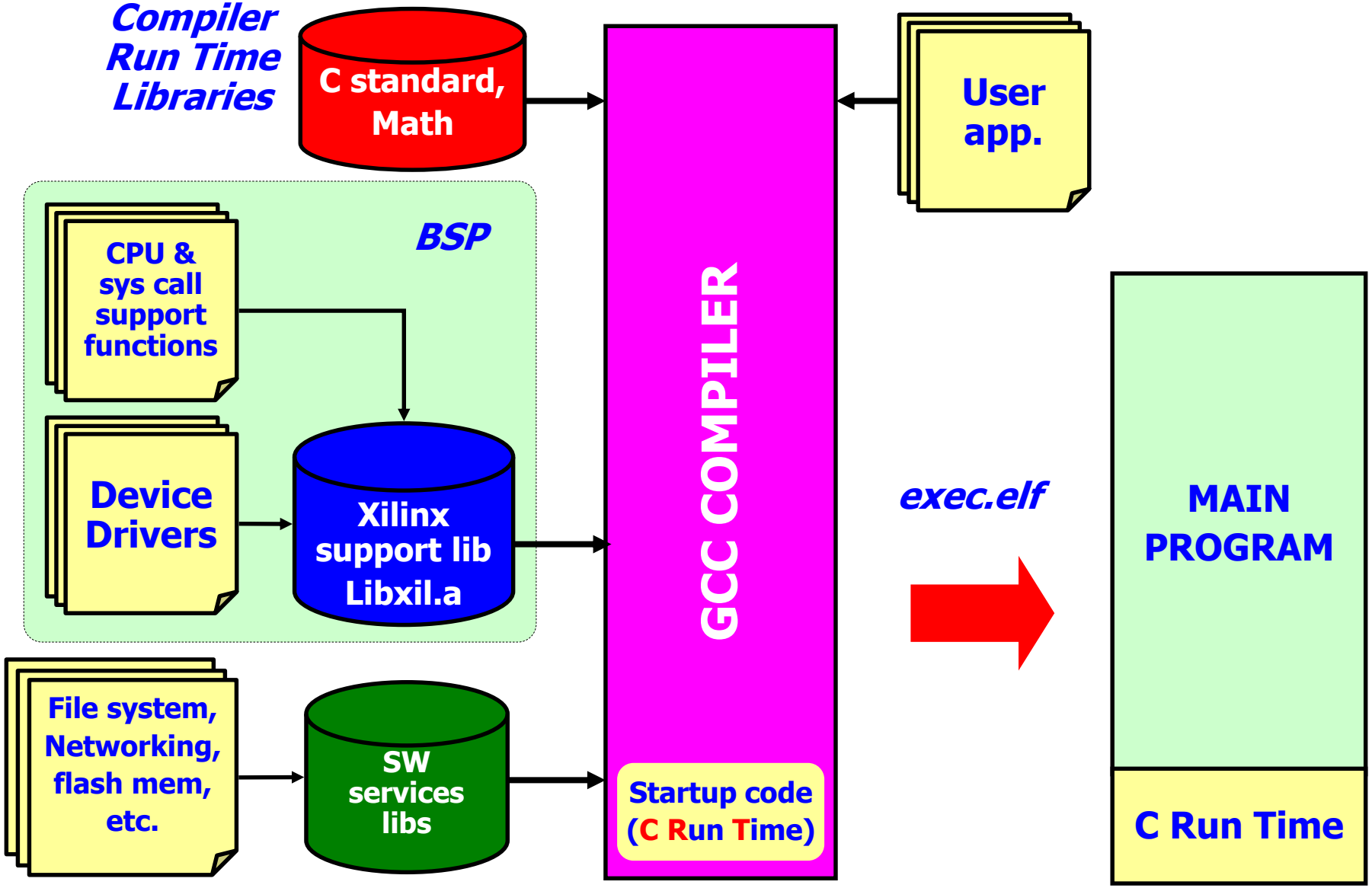
```
mb-objdump -S executable.elf
```



OUTLINE

- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - **LIBRARIES**
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

STANDALONE SW PLATFORM



COMPILER RUN-TIME LIBRARIES

- Desktop GCC compiler uses the GNU C library as the C standard library
  - Provides the system call interface that connects to the OS kernel
  - Shipped as a package that is made up of several modules: libc, libm
  - Full-featured but too large for embedded systems
  - More lightweight alternatives: uClibc, eglibc, dietlibc, and **newlib**

- Microblaze GCC uses **newlib** as run-time libraries for standalone and Xilinx Microkernel platforms
  - System services implemented by the Xilinx support library
  - Two files: libc, libm
  - EDK includes several precompiled variants for different CPU features
    - _bs Barrel shifter
    - _m HW multiplier
    - _p Patter comparator
    - _mh Extended HW multiplier
    - _fps Single precision
    - _fpd Double precision
  - LibGen copies the appropriated one to the project directory based on the used Microblaze HW configuration (MHS)

COMPILER RUN-TIME LIBRARIES

- Standard C language library (libc)
  - Contains standard C functions such us stdlib, stdio, string, etc.
  - Automatically linked with user code
    - No need to add the `-lc` option to the gcc command line
  - Limited functionality for dynamic memory allocation
  - I/O functions can result too heavy for embedded system
    - Consider to use simplified versions included in the Xilinx support library
- Math library (libm)
  - The math library is an improvement over the newlib math library
  - It must be explicitly linked with user code
    - Add the `-lm` option to the gcc commnad line

XILINX SUPPORT LIBRARY

- EDK includes a **libxil** precompiled basic version that provides:
  - OS-like support functionality for compiler libraries (sys call interface)
  - Default interrupt and exception handlers
  - Simplified I/O functions: xil_printf, print, putnum
  - Some low level support for GCC compiler (64 bits arithmetic)
- LibGen makes a local copy to the project directory and this copy is used to store platform BSP functions:
  - Microprocessor support: cache, interrupt and FSL management
  - Device drivers
- Automatically linked with user code
  - No need to add the `-lxil` option to gcc command line

XILINX SW SERVICES LIBRARIES

- Software service library package shipped as source code
  - LibXil MFS: Simple memory based file system
  - LibXil Flash: Access to parallel flash devices that conform to the Common Flash Interface (CFI)
  - LibXil Isf: Serial flash communication library
  - XilFATFS: File system support for System ACE CompactFlash or IBM microdrive devices
  - lwIP: open source TCP/IP stack to build network applications

OUTLINE

- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- **BOARD SUPPORT PACKAGE**
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

BOARD SUPPORT PACKAGES

- Currently EDK supports:
  - *Standalone BSP*: No OS is used and the user application accesses board or processor features directly
  - Wind River VxWorks (only for PPC)
  - Linux 2.4 & 2.6: (uCLinux for Microblaze without MMU)
  - Xilinx MicroKernel (XMK). It's a simple embedded processor Kernel that can be customized to a great degree for a given system

BSP

- Board Support Package (BSP):
  - Lowest layer of software modules used to access processor specific functions
    - Interrupt and Exception Handling
    - Instruction and Data Cache Handling
    - Fast Simplex Link interface macros
    - Program Profiling
  - Allows you to use IP peripheral-device drivers
    - GPIO, IIC controller, PCI controller, UART
  - Offers glue functionality to link code against standard libraries
    - Time, sleep
    - Files
    - Memory
  - Standalone BSP (no operating system)
    - Libgen generates `libxil.a` library

BSP: INTERRUPT MANAGEMENT FUNCTIONS

- Functions to enable, disable and register handlers (ISR)
  - Declared in **mb_interface.h**

```
void microblaze_enable_interrupts(void);
void microblaze_disable_interrupts(void);
void microblaze_enable_exceptions(void);
void microblaze_disable_exceptions(void);
void microblaze_register_handler(
         XInterruptHandler Handler,
         void *DataPtr);
void microblaze_register_exception_handler(
         Xuint8 ExceptionId,
         XExceptionHandler Handler,
         void *DataPtr);
```

BSP: CACHE MANAGEMENT FUNCTIONS

- Functions to enable, disable and update a single cache line or initialize a range of cache lines
  - Declared in **`mb_interface.h`**

```
void microblaze_enable_icache(void);
void microblaze_disable_icache(void);
void microblaze_enable_dcache(void);
void microblaze_disable_dcache(void);

void microblaze_update_icache(int tag, int inst, int lockValid);
void microblaze_init_icache_range(int cacheAddr, int cacheRange);
void microblaze_update_dcache(int tag, int inst, int lockValid);
void microblaze_init_dcache_range(int cacheAddr, int cacheRange);
```

BSP: FSL COMMUNITACION FUNCTIONS

- Blocking and not-blocking functions to send and receive write data through FSL channels
  - Defined as macros in **mb_interface.h**
  - Use in-line assembly

```
getfsl(val, id)    // Blocking Data Read and Write to FSL no. id
putfsl(val, id)
ngetfsl(val, id)   // Non-blocking Data Read and Write to FSL no. id
nputfsl(val, id)
cgetfsl(val, id)   // Blocking Control Read and Write to FSL no. id
cputfsl(val, id)
ncgetfsl(val, id) //Non-blocking Control Read and Write to FSL no. id
ncputfsl(val, id)
// Polling versions of FSL access macros.
// This makes the FSL access interruptible
getfsl_interruptible(val, id)
putfsl_interruptible(val, id)
cgetfsl_interruptible(val, id)
cputfsl_interruptible(val, id)
// FSL valid and error check macros.
fsl_isinvalid(result)
fsl_iserror(error)
```

OUTLINE

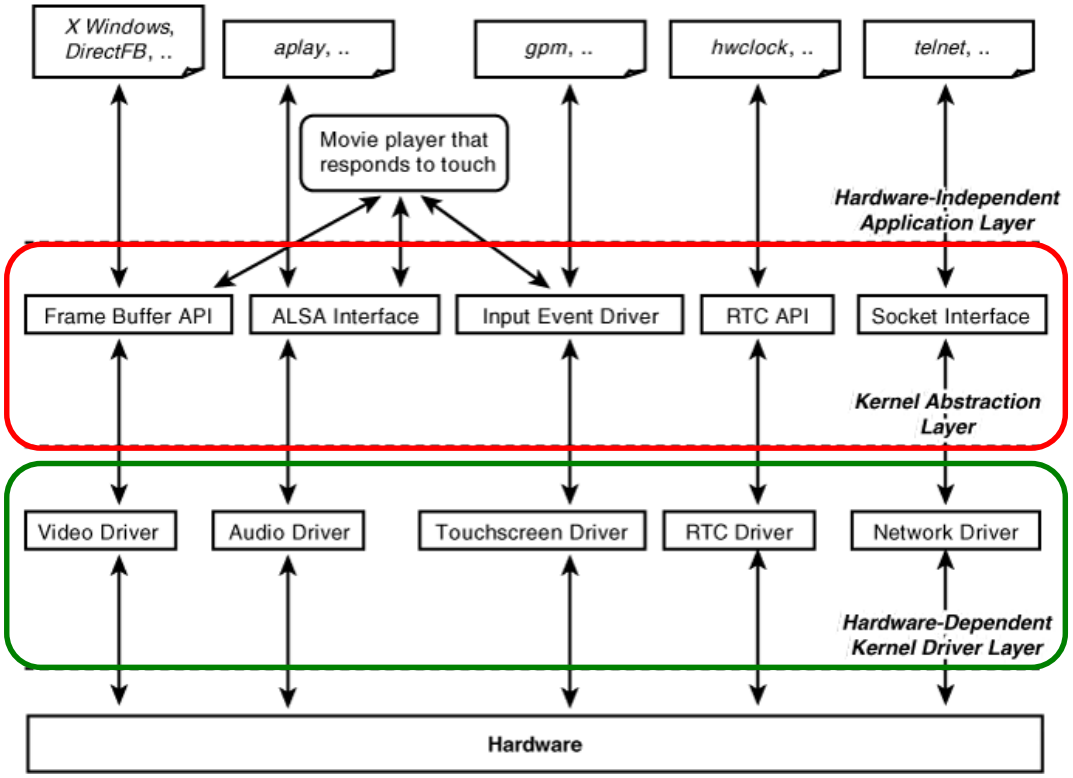
- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - **DEVICE DRIVERS**
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

DEVICE DRIVERS

- A device driver is a SW module that directly interfaces with and controls a HW device
  - It manages access to HW by higher software layers (kernel, user apps)
- Every OS defines it own device driver model (or architecture)
  - Standard API to communicate from upper layers

VxWorks

µC/OS-II™
The Real-Time Kernel

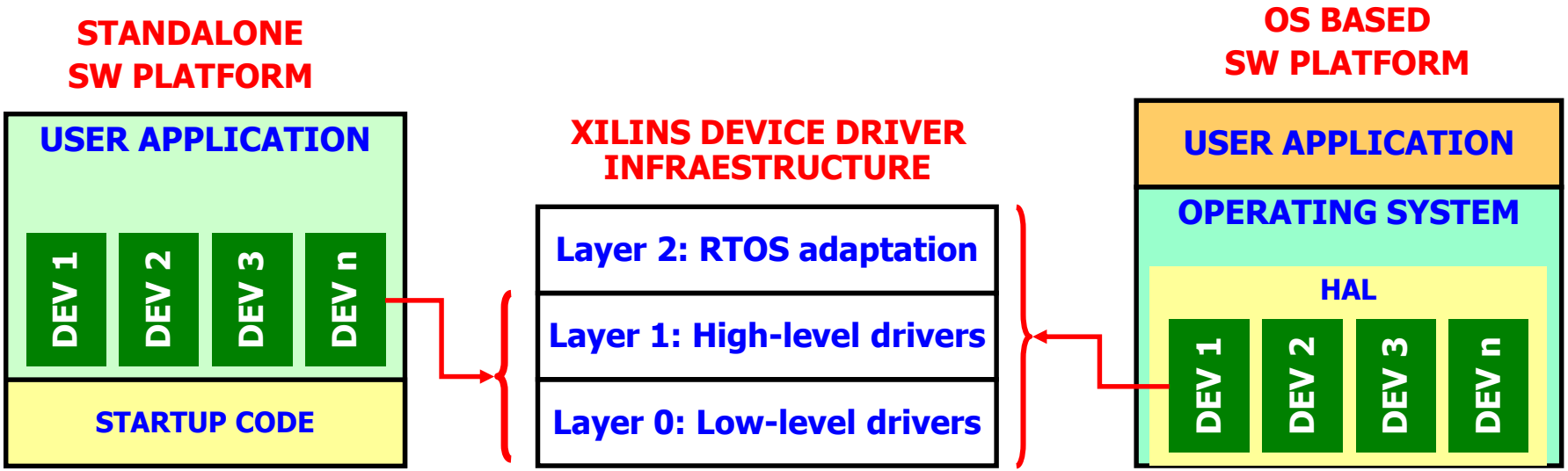


DEVICE DRIVERS

- The Xilinx device drivers are designed to meet the following objectives:
  - Provide maximum portability
    - The device drivers are provided as ANSI C source code
  - Support FPGA configurability
    - Supports multiple instances of the device without code duplication for each instance, while at the same time managing unique characteristics on a per-instance basis
  - Support simple and complex use cases
    - A layered device driver architecture provides both
      - Simple device drivers with minimal memory footprints
      - Full-featured device drivers with larger memory footprints
  - Ease of use and maintenance
    - Xilinx uses coding standards and provides well-documented source code for developers

DEVICE DRIVERS

- LibGen tool automatically generates drivers for devices included in the hardware platform
  - Designed to be portable across processor architectures and OS
  - Delivered as source code, allowing it to be built and optimized
  - C coded, limited use of assembly language
- Xilinx device driver infrastructure uses a multilayered approach
  - Layer 0: Low level drivers for simple use cases
  - Layer 1: High-level device drivers that are full-featured and portable across operating systems and processors
  - Layer 2: RTOS adaptation layer



DEVICE DRIVERS: LAYER 0

- Direct access to HW: Consists of very low level functions and macros
  - Designed to allow developer to create a small system
- Characteristics:
  - Small memory footprint, low function calling overhead
  - Include symbols (register offset, bit masks, etc.) and simple macros that give the user access to the HW registers
  - Little to no error checking is performed
  - Supports primary device features only
  - No support of device configuration parameters and device state
  - Supports multiple instances of a device
    - The API uses the device base address as identifier
  - Polled I/O only ➔ Blocking functions for simple use cases
  - Header files use '_l' suffix:
    - E.g. `xuartlite_l.h`

DEVICE DRIVERS: LAYER 1

- Provides high level functionality
- Implemented as macros and functions and designed to allow a developer to utilize all of the features of a device
- Characteristics
  - Abstract API that isolates the API from hardware device changes
  - Uses data structure to control the device configuration and/or status
  - Polled and interrupt driven I/O
  - Non-blocking function calls to aid complex applications
  - May have a large memory footprint
  - Typically, provides buffer interfaces for data transfers
    - as opposed to byte interfaces
  - Utilizes asynchronous callbacks for upward communication
  - Header files without '_' suffix
    - E.g. `xuartlite.h`

DEVICE DRIVERS: LAYER 2

- This layer adapts lower layers to the requirements of a given OS
  - Specific to every BSP
  - Every OS defines its device driver model
  - It converts a Layer 1 device driver to an interface that matches the requirements of the driver model for an RTOS
- Characteristics:
  - Communicates directly to the RTOS as well as the Layer 1 interface of the device driver
  - Not portable across operating systems
    - It contains references functions and identifiers specific to the RTOS
  - Can use memory management (i.e. malloc)
  - Can use RTOS services:
    - Threading, inter-task communication, etc.
  - Complexity depending on the RTOS interface and requirements for the device

DEVICE DRIVERS: EXAMPLE

Uartlite Level 1

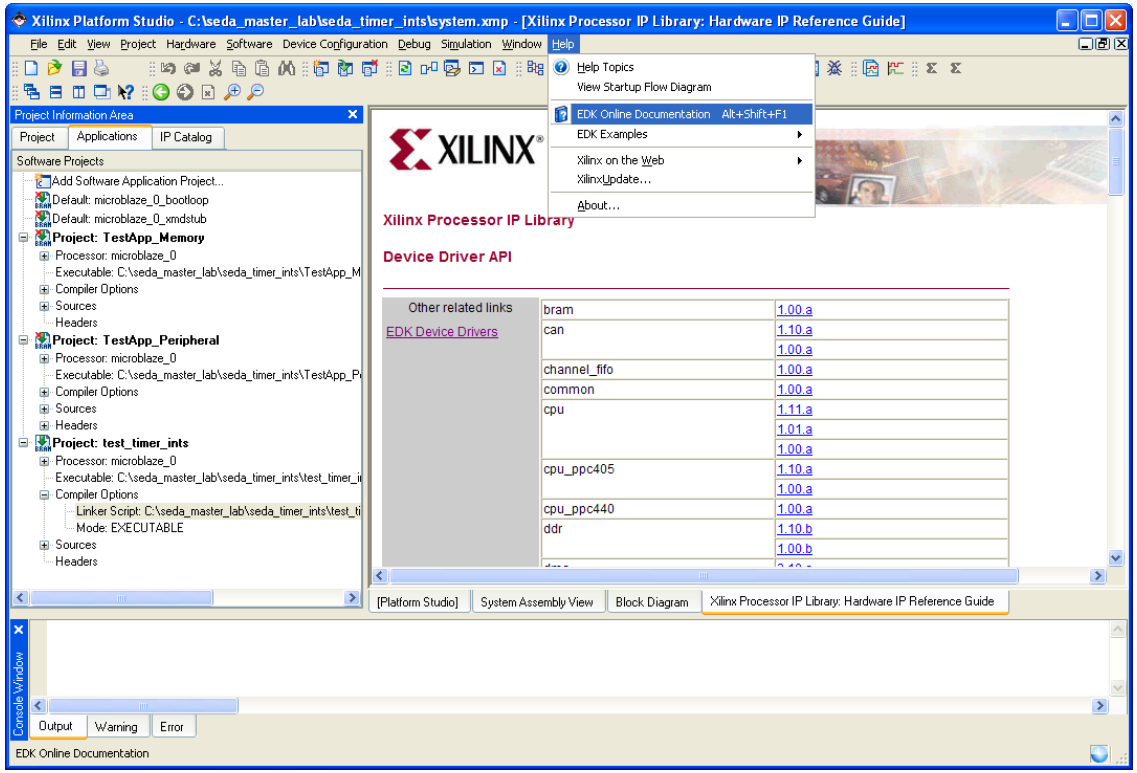
- XStatus XUartLite_Initialize (XUartLite *InstancePtr, Xuint16 DeviceId)
- void XUartLite_ResetFifos (XUartLite *InstancePtr)
- unsigned int XUartLite_Send (XUartLite *InstancePtr, Xuint8 *DataBufferPtr, unsigned int NumBytes)
- unsigned int XUartLite_Recv (XUartLite *InstancePtr, Xuint8 *DataBufferPtr, unsigned int NumBytes)
- Xboolean XUartLite_IsSending (XUartLite *InstancePtr)
- void XUartLite_GetStats (XUartLite *InstancePtr, XUartLite_Stats *StatsPtr)
- void XUartLite_ClearStats (XUartLite *InstancePtr)
- XStatus XUartLite_SelfTest (XUartLite *InstancePtr)
- void XUartLite_EnableInterrupt (XUartLite *InstancePtr)
- void XUartLite_DisableInterrupt (XUartLite *InstancePtr)
- void XUartLite_SetRecvHandler (XUartLite *InstancePtr, XUartLite_Handler FuncPtr, void *CallBackRef)
- void XUartLite_SetSendHandler (XUartLite *InstancePtr, XUartLite_Handler FuncPtr, void *CallBackRef)
- void XUartLite_InterruptHandler (XUartLite *InstancePtr)

Uartlite Level 0

- void XUartLite_SendByte (Xuint32 BaseAddress, Xuint8 Data)
- Xuint8 XUartLite_RecvByte (Xuint32 BaseAddress)

DEFINED API

- EDK provides a detailed description about the API for each device driver
  - HTML format generated with DOXIGEN
  - Available through EDK help
    - C:\EDA\Xilinx\10.1\EDK\doc\usenglish\xilinx_drivers_api_toc.htm
- EDK defines a standard API for level 1 drivers
  - Level 0 varies highly among different devices



LAYER 0 API

- Layer 0 is just a bundle of functions and macros that allow to access to the peripherals internal registers

```
void XGpio_mWriteReg(u32 BaseAddr, u32 RegOffset, u32 Data)
u32  XGpio_mReadReg(u32 BaseAddr, u32 RegOffset)
void XGpio_mSetDataDirection(u32 BaseAddr, unsigned Channel, u32 DirMask)
u32  XGpio_mGetDataReg(u32 BaseAddr, unsigned Channel)
void XGpio_mSetDataReg(u32 BaseAddr, unsigned Channel, u32 Data)
```

- It's necessary to know the base address in the memory map where every device instance is located
- Furthermore, IPs are highly parameterizable
  - It's necessary to know the configuration used by every device instance in order to properly control its operation
- System software also may need to know which interrupt vector the device is attached to
- LibGen tool automatically generates a file called '**`xparameter.h`**' containing all configuration info about the hardware platform

HW CONFIGURATION PARAMETERS

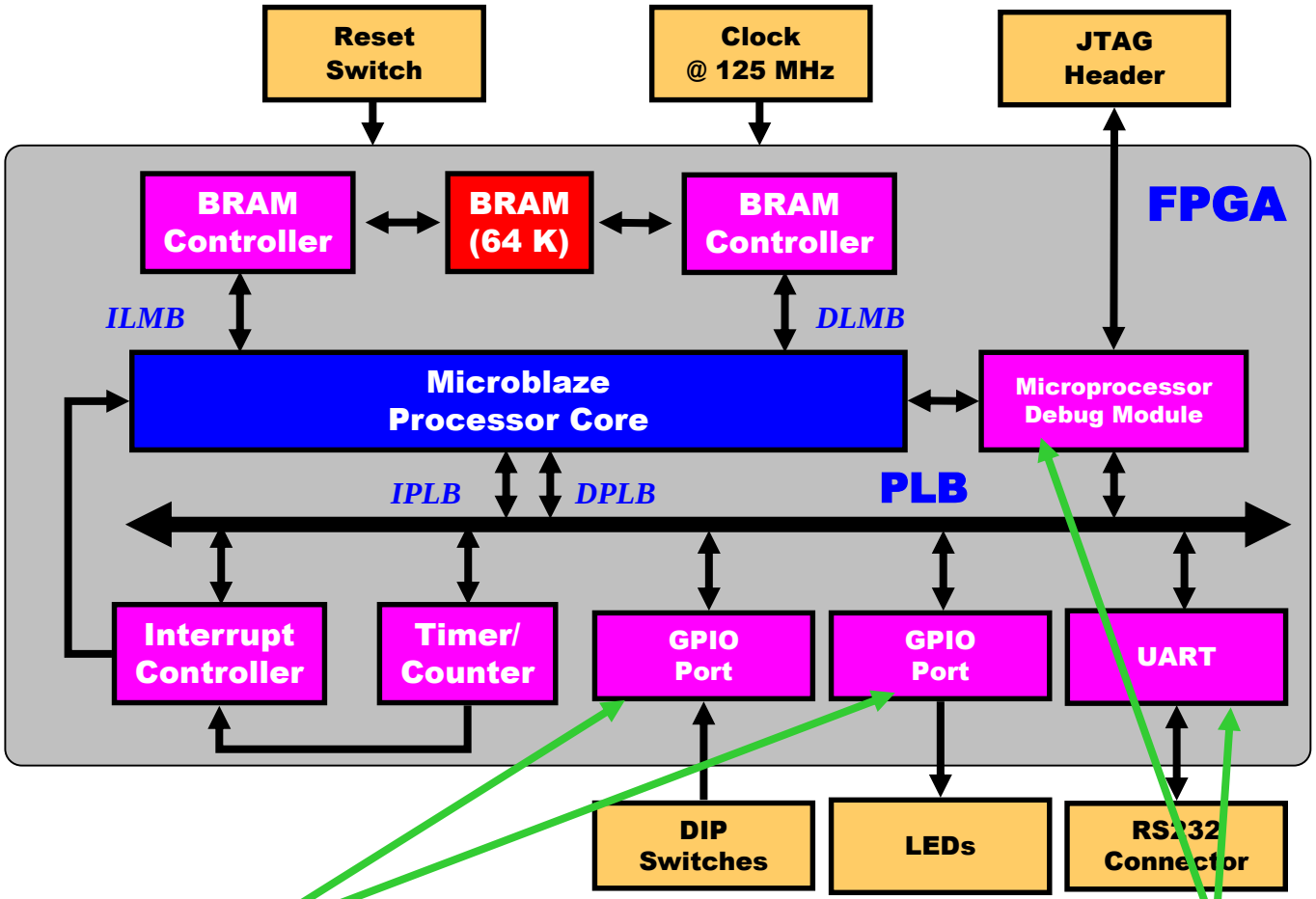
- 'xparameters.h' file contains symbols (#define) that provide info about the configuration used for every device and driver within the system
  - Used by BSP and device drivers to configure the system at runtime

- Configuration info for each device:

Common info

  - Number of device instances
  - Device ID for each instance (uint16)
    - used during initialization to perform the mapping of a device driver to a HW device
  - Base address for each device instance
  - Interrupt assignments for each device instance (if used)
  - *Other device specific info*

- For each device driver type:
  - XPAR_X<driver_name>_NUM_INSTANCES
- For each device instance:
  - XPAR_<driver_name>_<device_instance>_DEVICE_ID
  - XPAR_<driver_name>_<device_instance>_BASEADDR
  - XPAR_<driver_name>_<device_instance>_HIGHADDR
  - *XPAR_<intc_device_instance>_<interrupting_device_instance>_VEC_ID*



Two instances of the same device:
-Same driver (same code)
-One hander for each one

Two devices use the same driver:
- Driver 'uarlite' (same code)
- One handler for each one

HW CONFIGURATION PARAMETERS

MHS

```
BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_SPLB_CLK_FREQ_HZ = 62500000
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RX = fpga_0_RS232_Uart_1_RX
 PORT TX = fpga_0_RS232_Uart_1_TX
 PORT Interrupt = uart_interrupt
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_MB_DBG_PORTS = 1
 PARAMETER C_USE_UART = 1
 PARAMETER C_UART_WIDTH = 8
 PARAMETER C_BASEADDR = 0x84400000
 PARAMETER C_HIGHADDR = 0x8440ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_dbg
 PORT Debug_SYS_Rst = Debug_SYS_Rst
END
```

MSS

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.13.a
 PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.13.a
 PARAMETER HW_INSTANCE = debug_module
END
```

xparameter.h

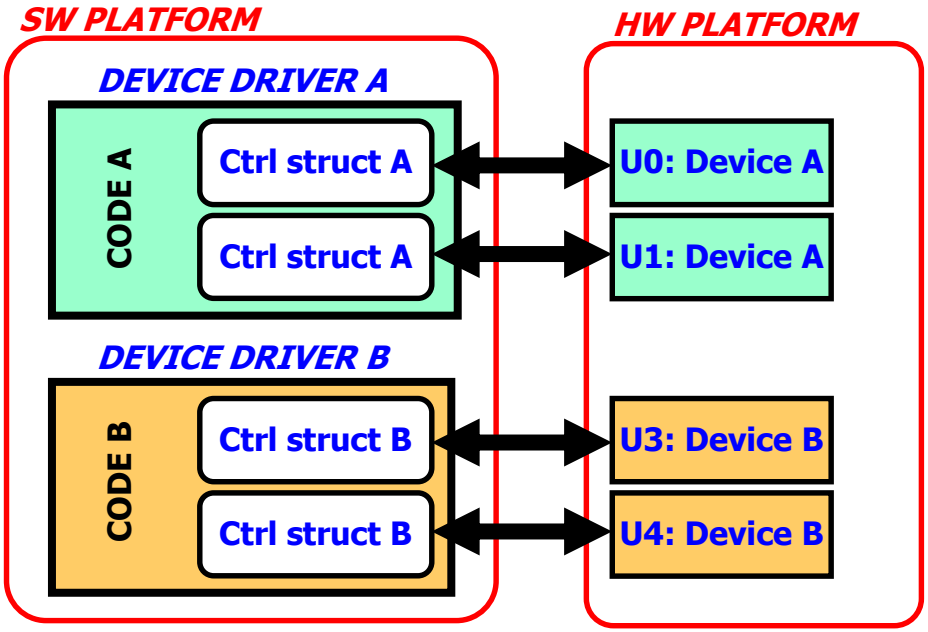
```
/* Definitions for driver UARTLITE */
#define XPAR_XUARTLITE_NUM_INSTANCES 2

/* Definitions for peripheral RS232_UART_1 */
#define XPAR_RS232_UART_1_BASEADDR 0x84000000
#define XPAR_RS232_UART_1_HIGHADDR 0x8400FFFF
#define XPAR_RS232_UART_1_DEVICE_ID 0
#define XPAR_RS232_UART_1_BAUDRATE 9600
#define XPAR_RS232_UART_1_USE_PARITY 0
#define XPAR_RS232_UART_1_ODD_PARITY 0
#define XPAR_RS232_UART_1_DATA_BITS 8

/* Definitions for peripheral DEBUG_MODULE */
#define XPAR_DEBUG_MODULE_BASEADDR 0x84400000
#define XPAR_DEBUG_MODULE_HIGHADDR 0x8440FFFF
#define XPAR_DEBUG_MODULE_DEVICE_ID 1
#define XPAR_DEBUG_MODULE_BAUDRATE 0
#define XPAR_DEBUG_MODULE_USE_PARITY 0
#define XPAR_DEBUG_MODULE_ODD_PARITY 0
#define XPAR_DEBUG_MODULE_DATA_BITS 0

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

/* Canonical definitions for peripheral RS232_UART_1 */
#define XPAR_UARTLITE_0_DEVICE_ID XPAR_RS232_UART_1_DEVICE_ID
#define XPAR_UARTLITE_0_BASEADDR 0x84000000
#define XPAR_UARTLITE_0_HIGHADDR 0x8400FFFF
#define XPAR_UARTLITE_0_BAUDRATE 9600
#define XPAR_UARTLITE_0_USE_PARITY 0
#define XPAR_UARTLITE_0_ODD_PARITY 0
#define XPAR_UARTLITE_0_DATA_BITS 8
#define XPAR_UARTLITE_0_SIO_CHAN 0


/* Canonical definitions for peripheral DEBUG_MODULE */
#define XPAR_UARTLITE_1_DEVICE_ID XPAR_DEBUG_MODULE_DEVICE_ID
#define XPAR_UARTLITE_1_BASEADDR 0x84400000
#define XPAR_UARTLITE_1_HIGHADDR 0x8440FFFF
#define XPAR_UARTLITE_1_BAUDRATE 0
#define XPAR_UARTLITE_1_USE_PARITY 0
#define XPAR_UARTLITE_1_ODD_PARITY 0
#define XPAR_UARTLITE_1_DATA_BITS 0
#define XPAR_UARTLITE_1_SIO_CHAN -1
```

LAYER 0 API EXAMPLE


```
#include "xparameters.h"
#include "xgpio_l.h"
#include "xbasic_types.h"

#define LEDS_CHANNEL      1
#define SWITCHES_CHANNEL  1

int main(int argc, char *argv[]) {
  Xuint32 value;

  // set LEDS to OUT
  XGpio_mSetDataDirection(XPAR_LEDS_8BIT_BASEADDR, LEDS_CHANNEL, 0x00);
  // set SWITCHES to IN
  XGpio_mSetDataDirection(XPAR_DIP_SWITCHES_8BIT_BASEADDR,
    SWITCHES_CHANNEL, 0xFF);

  while(1) {
    value = XGpio_mGetDataReg(XPAR_DIP_SWITCHES_8BIT_BASEADDR,
              SWITCHES_CHANNEL);
    XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR, LEDS_CHANNEL, value);
  }
  return 0;
}
```

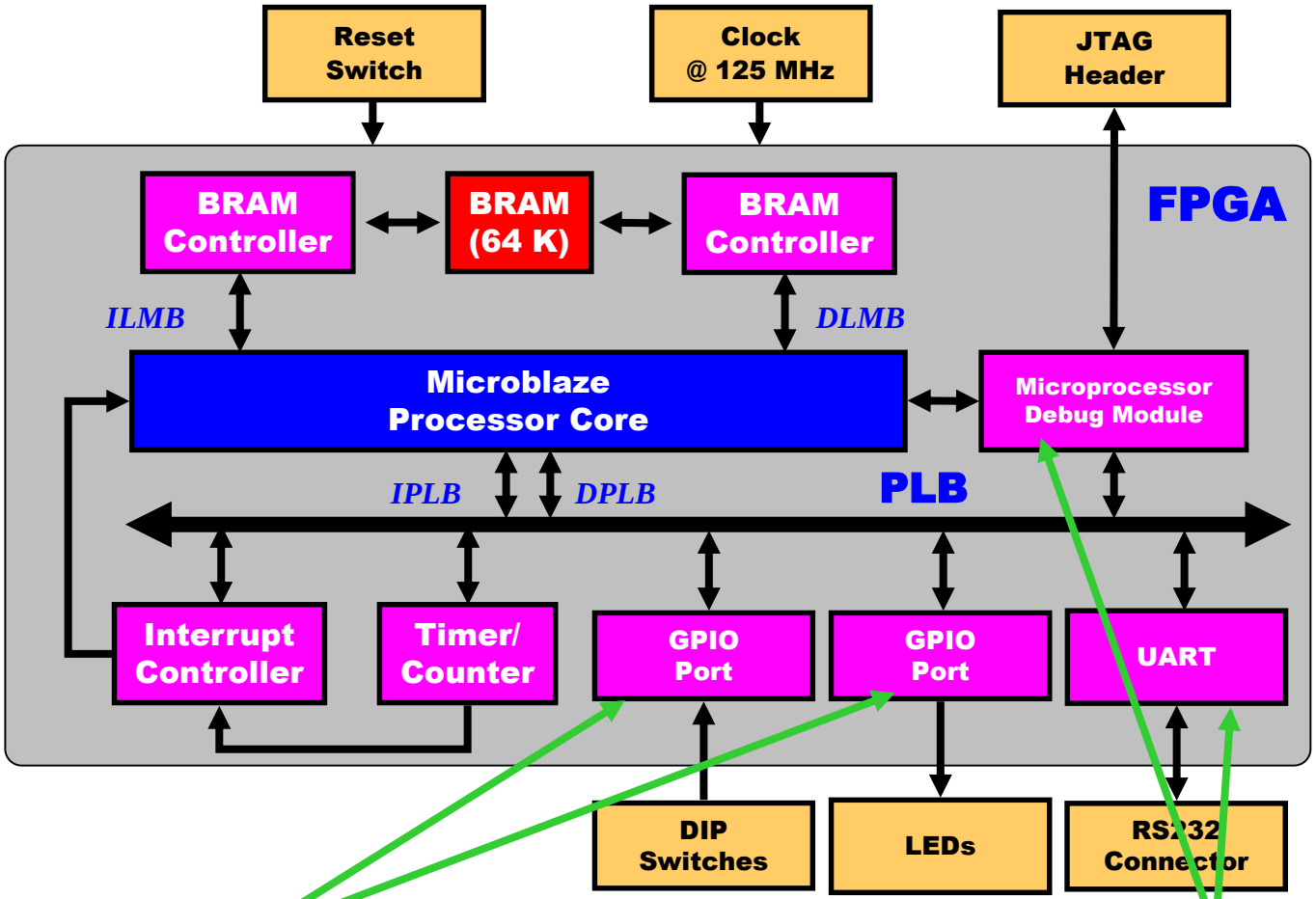
LAYER 1 STANDARD API

- For layer 1, each device instance is controlled by means of a data structure containing configuration and status info about this instance
  - Defined for each device driver
  - User must declare a variable of this type for each device instance
  - First parameter for API functions (OOP with C)
    - The pointer to the data structure serves as device driver reference
  - Deferred as *driver instance data* or *driver object*




```
typedef struct {
  XTmrCtrStats Stats;
  u32 BaseAddress;
  u32 IsReady;

  XTmrCtr_Handler Handler;
  void *CallBackRef;
} XTmrCtr;

...
void XTmrCtr_Start(
  XTmrCtr *InstancePtr,
  u8 TmrCtrNumber);
```

DEVICE AND DEVICE DRIVERS



Two instances of the same device:
- Same driver (same code)
- One hander for each one

Two devices use the same driver:
- Driver 'uarlite' (same code)
- One handler for each one

STANDARD API

- API common functions
  - Initialize: Initializes an instance of a device driver
    - Driver instance must be initialize before other API function can be called
  - Reset: resets the device driver and device with which it is associated
    - Allows to recover from exception conditions
  - SelfTest: performs a self-test on the device driver and its associated device
  - LookupConfig: retrieves a pointer to the configuration table for a device driver

- API optional functions
  - Start: Starts the device driver (device and interrupt enable)
  - Stop: Stops the device driver (device and interrupt disable)
  - GetStats: gets the statistics for the device and/or device driver
  - ClearStats: clears the statistics for the device and/or device driver
  - InterruptHandler: High level custom interrupt processing for each device
    - Interrupt acknowledge
    - User functionality provided through a callback function (hook)

LAYER 1 STANDARD API

- Before a device instance can be used its data structure must be properly initialized using the `Initialize()` function
  - It maps an instance driver with a given device
  - The user is responsible for allocating the driver handler passed as 1st parameter
  - The device instance is identified by the second parameter obtained from '**`x_parameter.h`**'

```
int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId)
```

- From that point, the driver instance is referred by its data structure address:

```
u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel)
```

HW CONFIGURATION PARAMETERS

- System software can need to obtain configuration info for each instance of a given driver at runtime
  - E.g. OS driver enumeration
- Each device driver (SW) uses a data structure to store info about the HW configuration applied to the associated device
  - Defined in `x<driver_name>.h` file
  - Contains most of the info included in `xparameters.h` file
  - It mustn't be confounded with the driver data structure (1st parameter used in Layer 1 functions)
  - Format:

```
typedef struct {
  u16 DeviceID;
  u32 BaseAddress;
  // Other device dependent data attributes
} X<driver_name>_Config;
```

- Moreover, an array of structures is defined with one element for each device instance in the system
  - Declared within `x<driver_name>_g.c`
  - The structure address for a given instance can be obtained by means of `x<driver_name>_LookupConfig()` function

LAYER 1 EXAMPLE


```
#include "xparameters.h"
#include "xtmrctr.h"


#define SYS_TIMER_DEV_ID      XPAR_XPS_TIMER_1_DEVICE_ID
XTmrCtr sysTimerDrv;  // The instance of the timer counter

int status;
Xuint32 options;

// Initialize the TmrCtr driver so that it's ready to use
status = XTmrCtr_Initialize(&sysTimerDrv, XPAR_XPS_TIMER_1_DEVICE_ID);
if(status != XST_SUCCESS) {
  return XST_FAILURE;
}
options =   XTC_DOWN_COUNT_OPTION   |
            XTC_INT_MODE_OPTION     |
            XTC_AUTO_RELOAD_OPTION;
XTmrCtr_SetOptions(&sysTimerDrv, TIMER_CNTR_0, options);
```

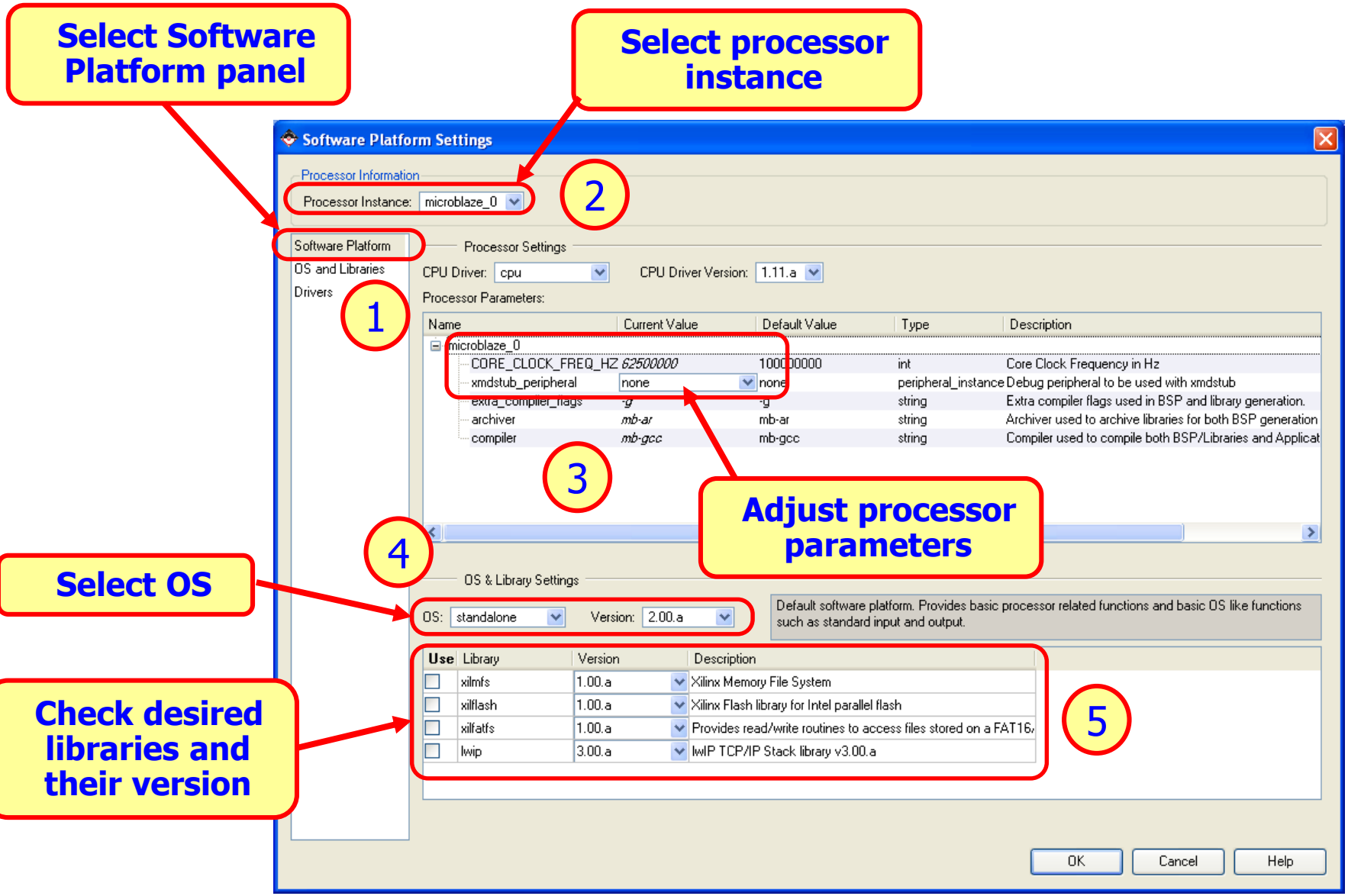
OUTLINE

- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- **XPS SOFTWARE SETTINGS**
  - SOFTWARE PLATFORM SETTINGS
  - COMPILER SETTINGS

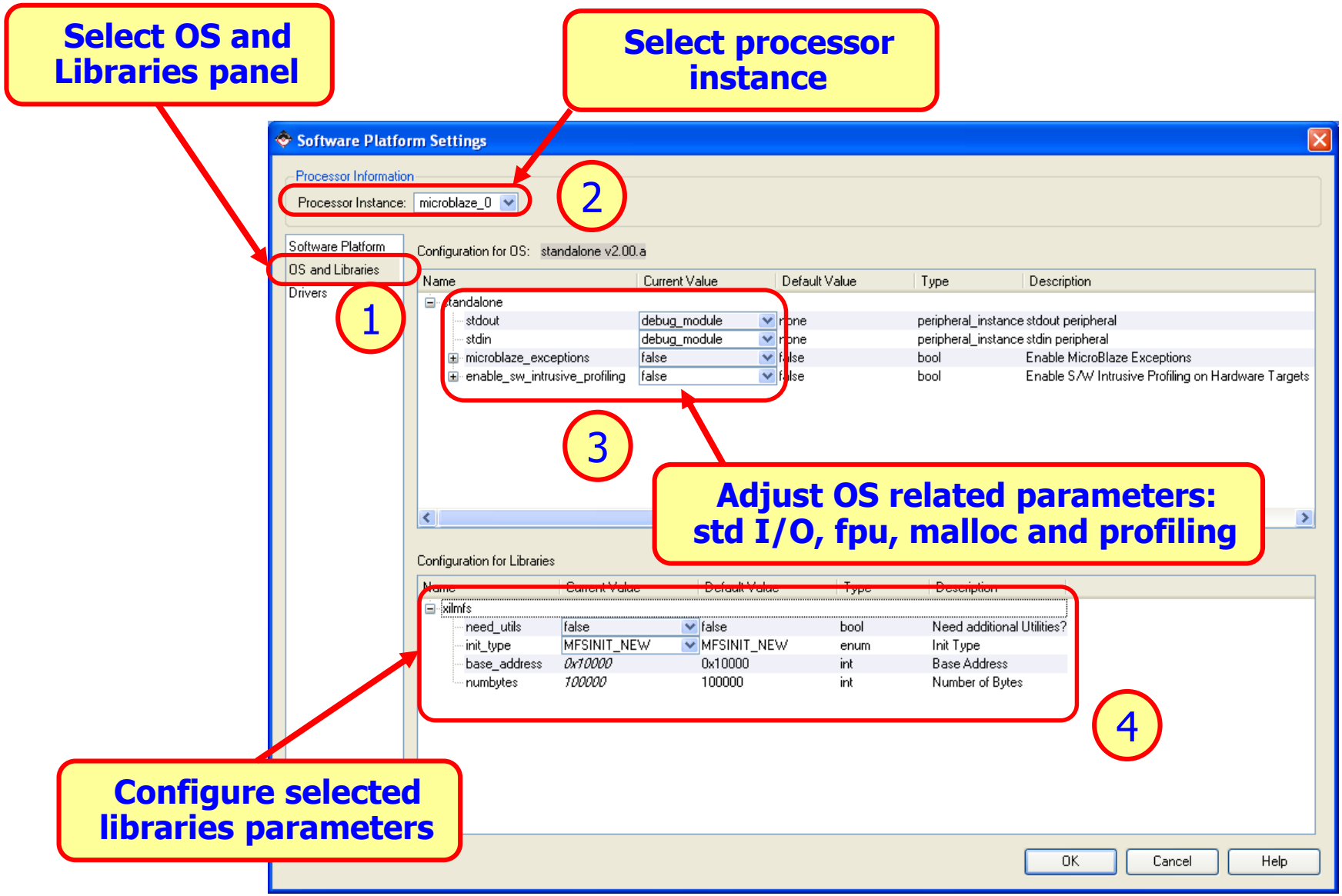
CODE GENERATION

- The following are the steps involved in generating software for EDK designs using XPS :
  1. Configure Software settings
  2. View and Set Project Options
  3. Create EDK Software libraries
  4. Open/Create your applications
  5. View and Set Applications' Options
  6. Build Applications
  7. Initialize bitstream with applications
  8. Download and execute application
  9. Download and debug applications by using XMD
- You can configure software settings for an EDK project
  - by using the XPS GUI 
  - by directly editing the MSS file.

SOFTWARE PLATFORM SETTINGS (I)

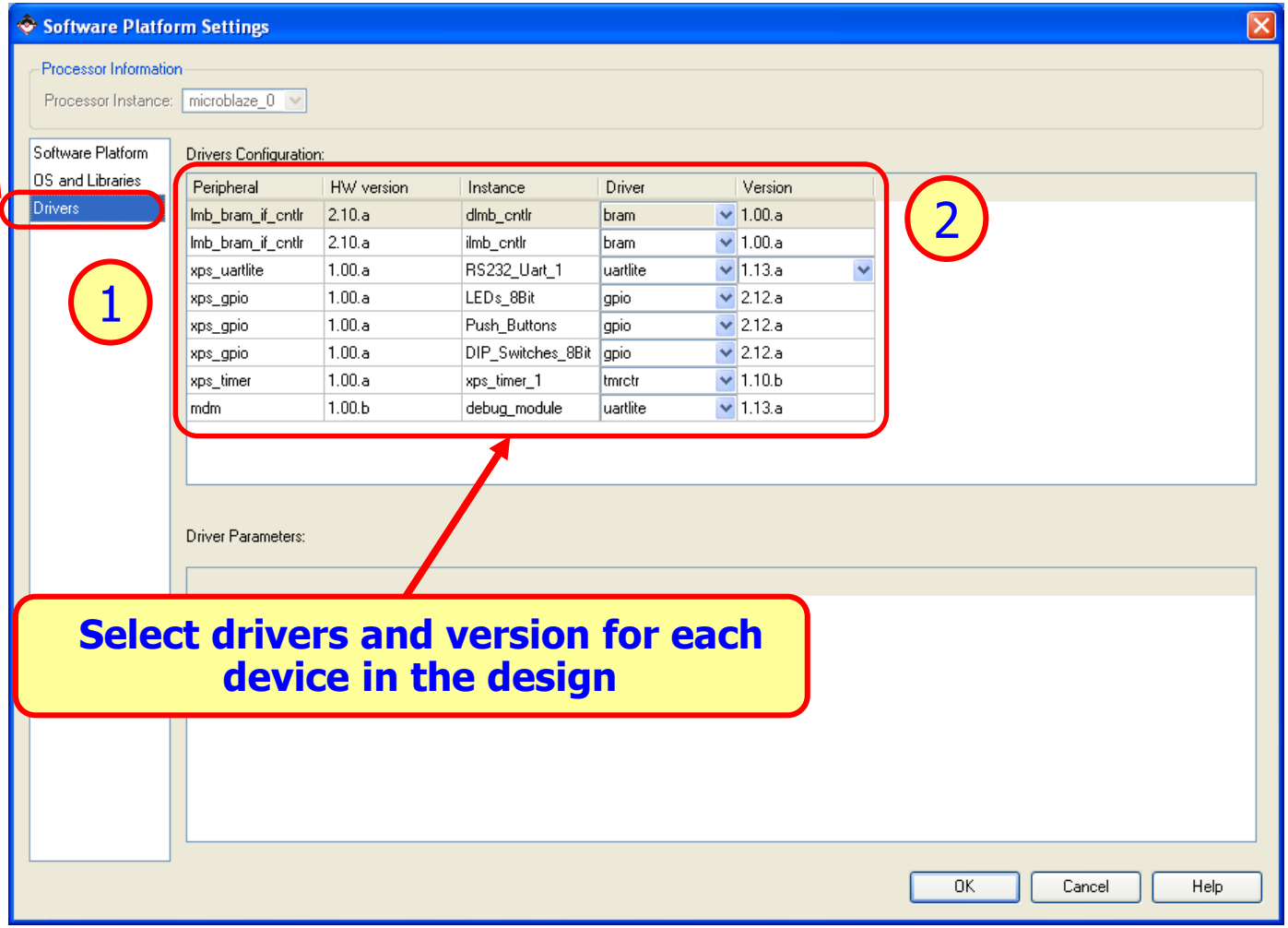


SOFTWARE PLATFORM SETTINGS (II)



SOFTWARE PLATFORM SETTINGS (III)

Select Drivers panel



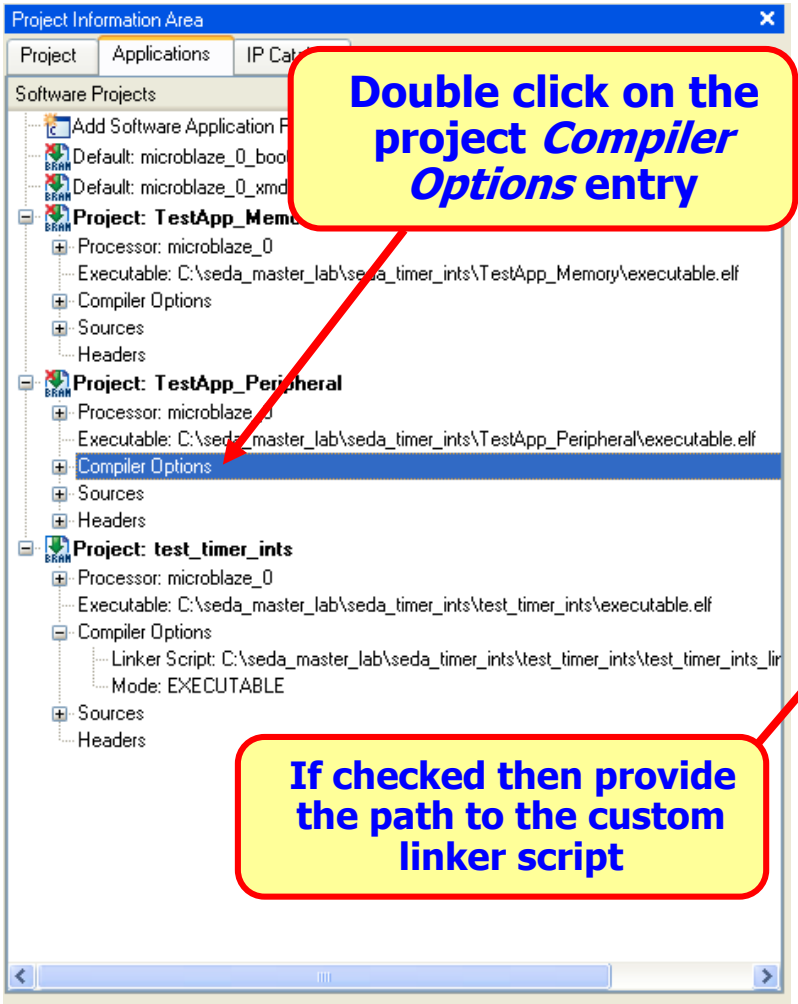
Select drivers and version for each device in the design

OUTLINE

- INTRODUCTION
- SOFTWARE DESIGN FLOW
  - LIBGEN
  - GNU TOOLCHAIN
  - LIBRARIES
- BOARD SUPPORT PACKAGE
  - STANDALONE BSP
  - DEVICE DRIVERS
- XPS SOFTWARE SETTINGS
  - SOFTWARE PLATFORM SETTINGS
  - **COMPILER SETTINGS**

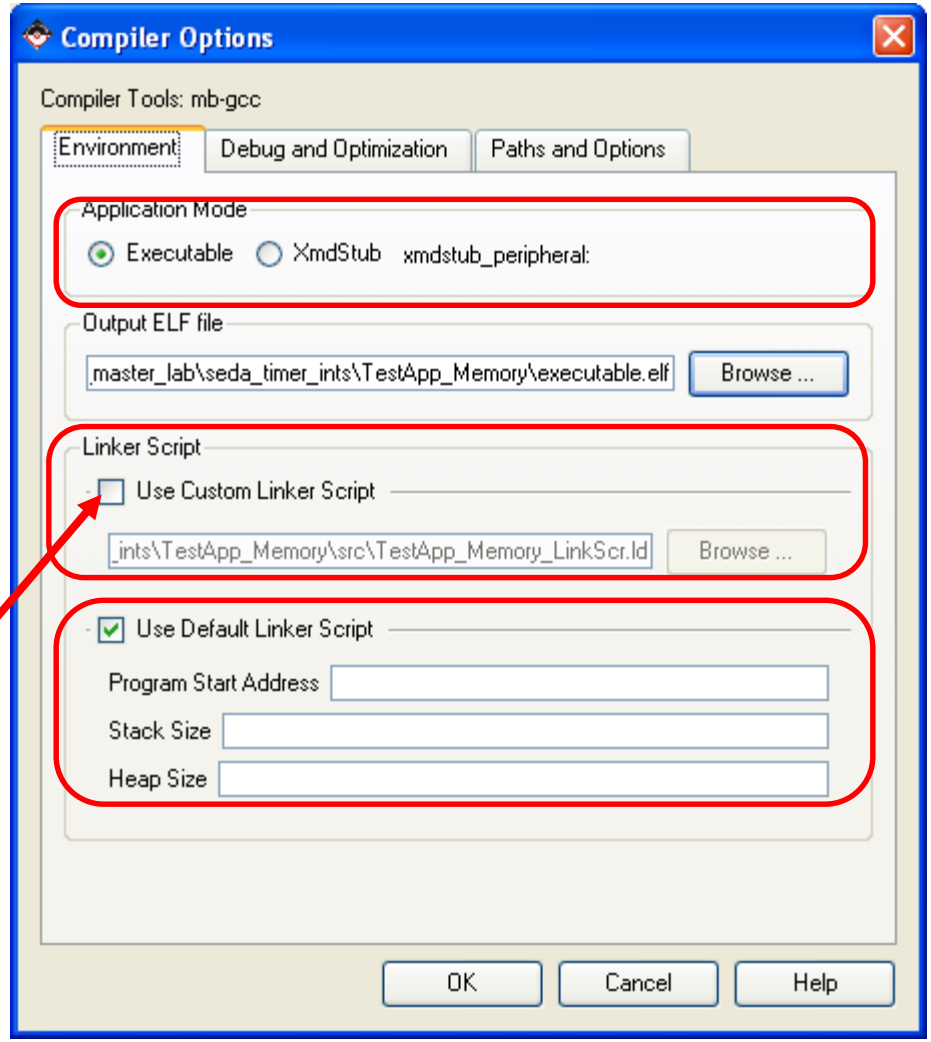
COMPILER SETTING (I)

ENVIRONMENT TAB



Double click on the project *Compiler Options* entry

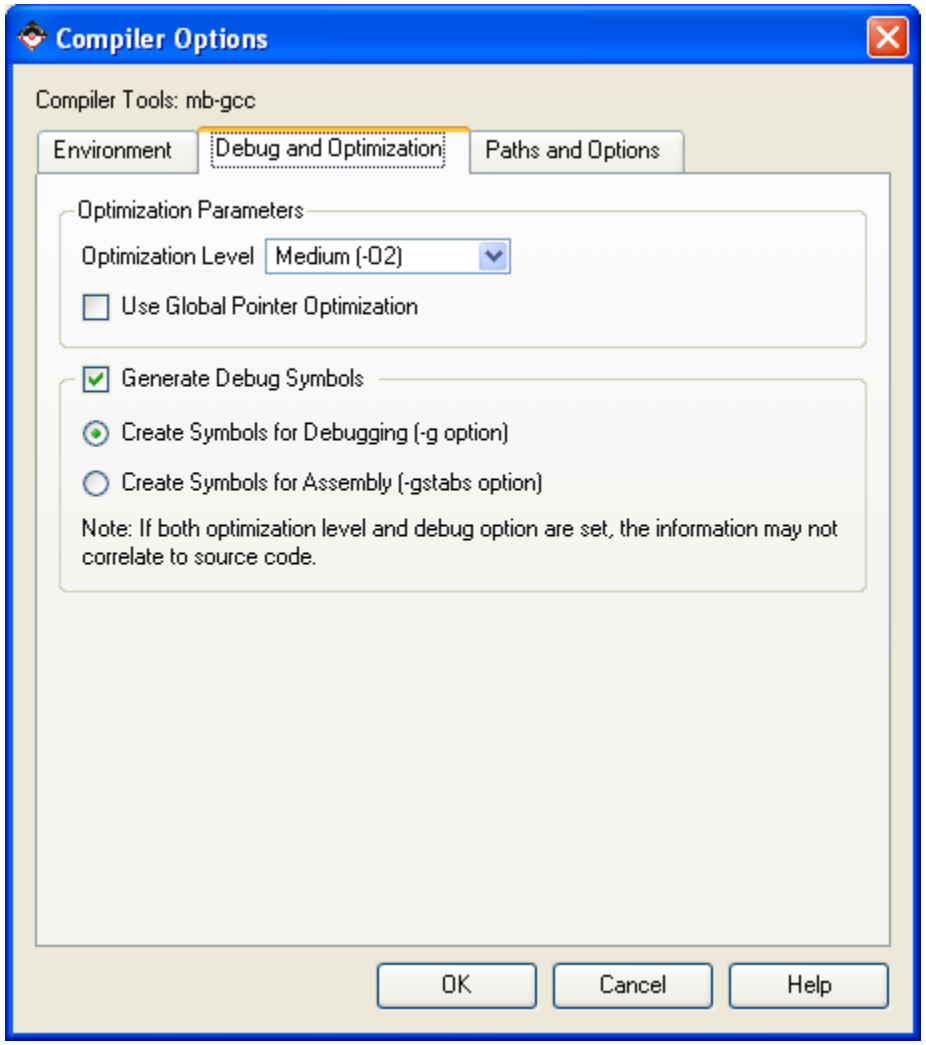
If checked then provide the path to the custom linker script



COMPILER SETTING (II)

DEBUG AND OPTIMIZATION TAB

- Optimization parameters:
  - Optimization level (0-3):
    - Level 0: No optimization, useful during debugging
    - Levels 1-3: Code reordering
- Use Global Pointer Optimization
  - Use of small data areas
    - Data size (8 bytes)
    - Registers: R2, R13
- Generate Debug Symbols
  - Include debugging information for GDB
    - C language: -g
    - Assembly: -gstabs



COMPILER SETTING (III)

PATH AND OPTIONS TAB

Search paths

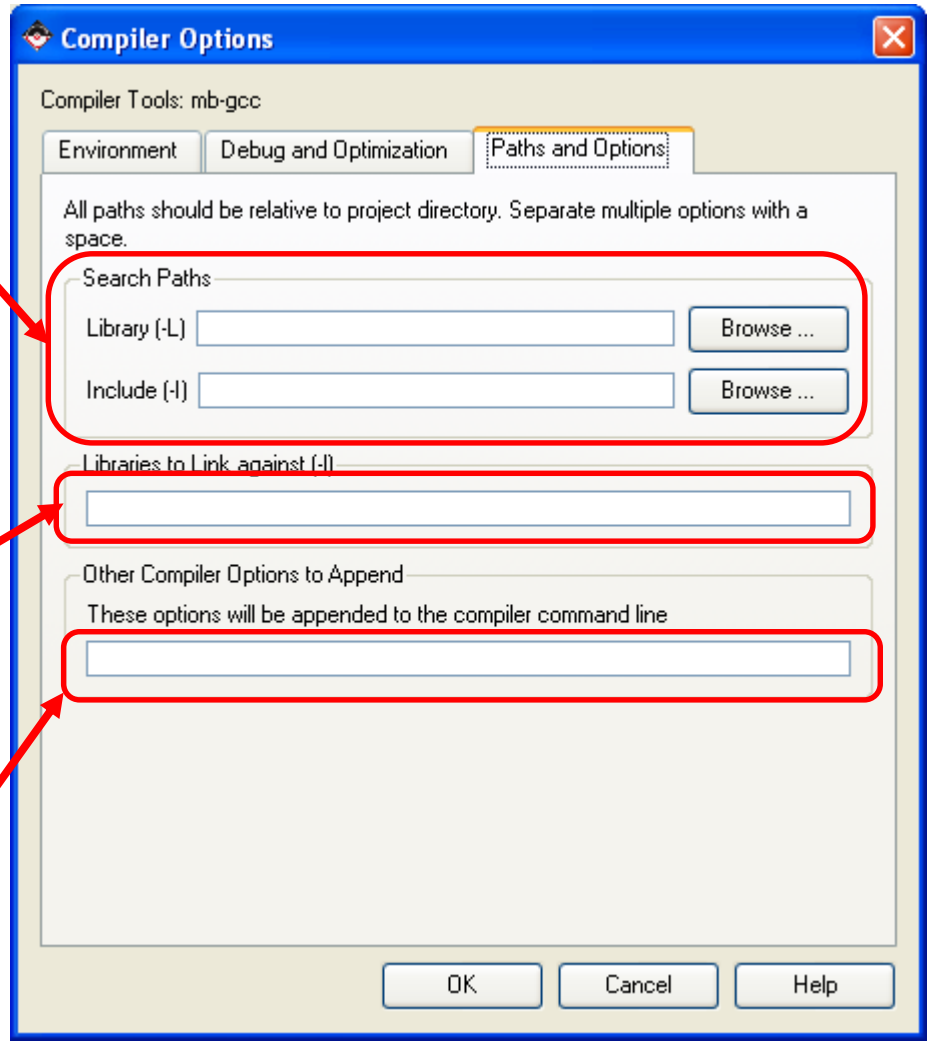
- Libraries. By default:
  - Project directory
  - MB_instance/lib directory
- Includes. By default:
  - Project directory
  - MB_instance/include directory

Additional libraries:

- User libraries

Other compiler options to append:

- E.g. Symbol definition for conditional compilations
  - -Dmy_def



END