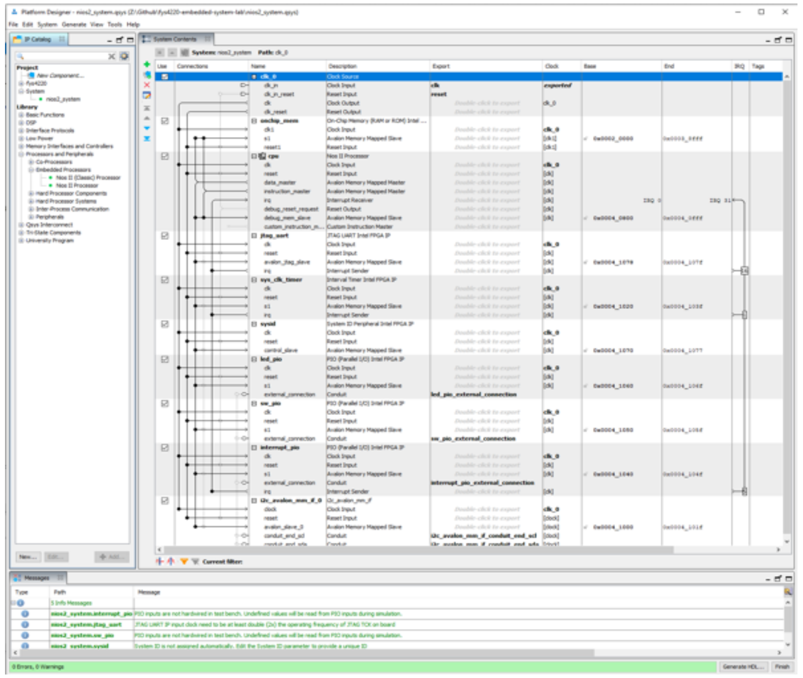
Embedded systems:

Nios II Software development

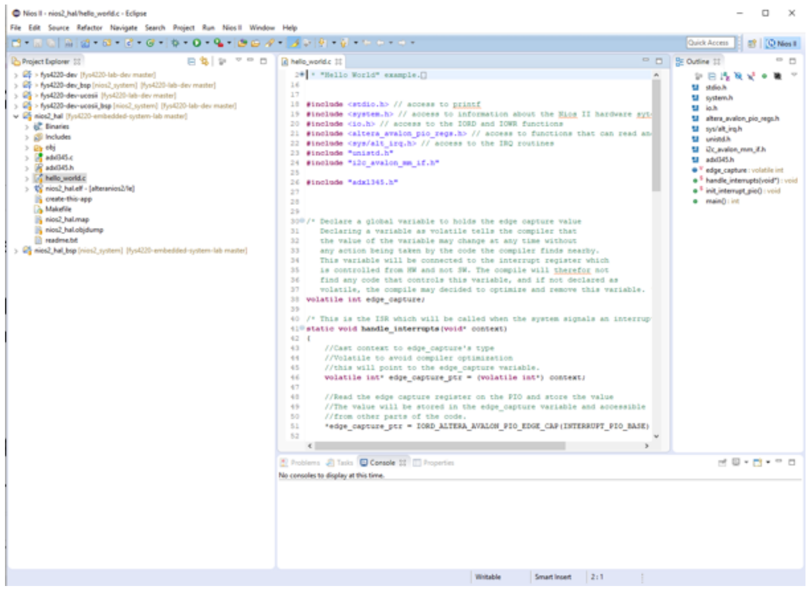
Nios II system development flow

- Hardware generation process
  - Platform designer to configure and generate a NIOS II system
- Software creation process
  - NIOS II software build tools for Eclipse
  - C/C++ compiler based on GNU toolchain



Type	Path	Message
	5 Info Messages	
	nios2_system.interrupt_pio	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
	nios2_system.jtag_uart	JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board
	nios2_system.sw_pio	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
	nios2_system.sysid	System ID is not assigned automatically. Edit the System ID parameter to provide a unique ID

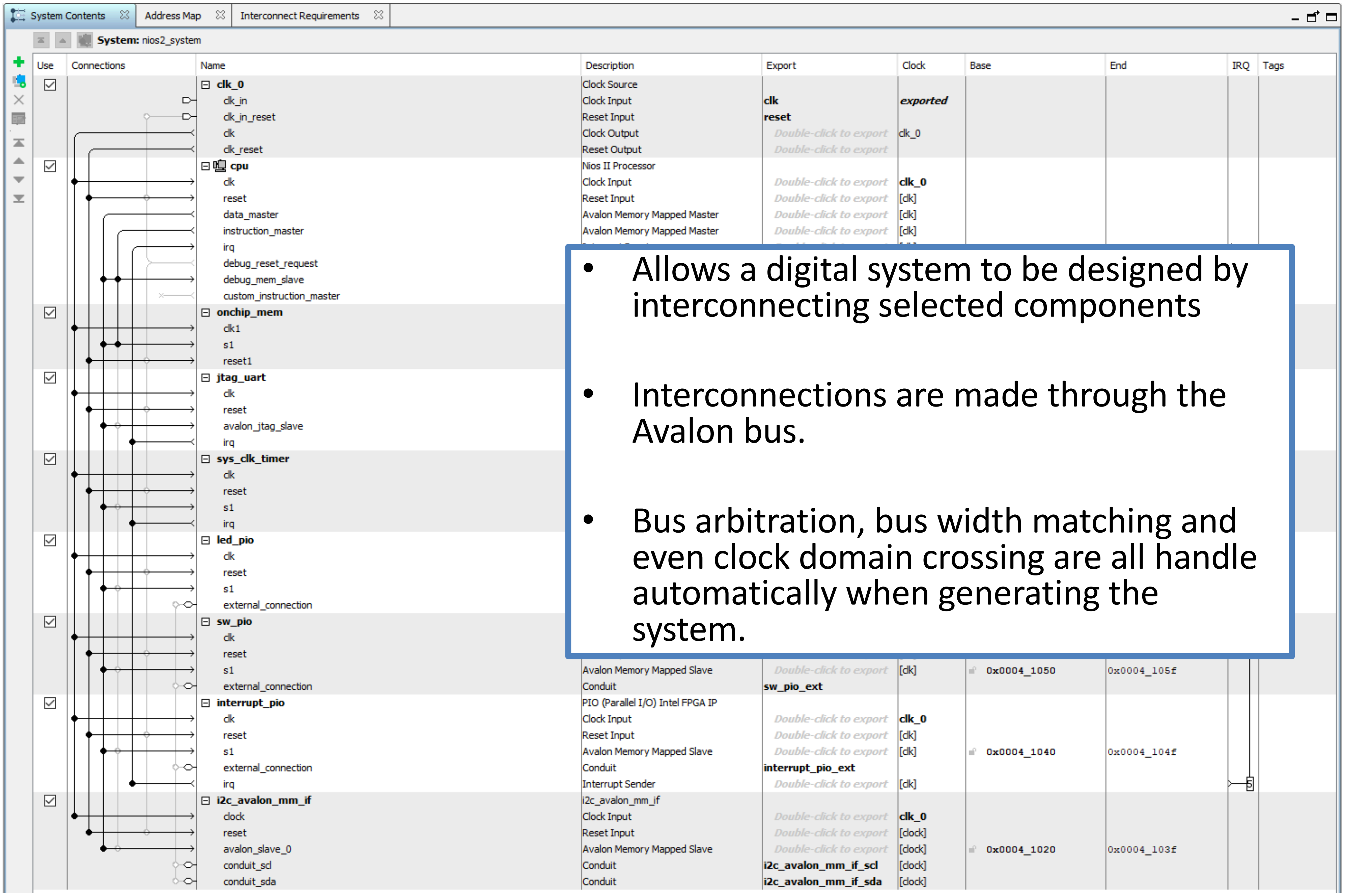
0 Errors, 0 Warnings



```
 * "Hello World" example.

#include <stdio.h> // access to printf
#include <system.h> // access to information about the Nios II hardware syt
#include <io.h> // access to the IORD and IOWR functions
#include <altera_avalon_pio_regs.h> // access to functions that can read an
#include <sys/alt_irq.h> // access to the IRQ routines
#include "unistd.h"
#include "i2c_avalon_mm_if.h"

#include "adxl345.h"



/* Declare a global variable to holds the edge capture value
   Declaring a variable as volatile tells the compiler that
   the value of the variable may change at any time without
   any action being taken by the code the compiler finds nearby.
   This variable will be connected to the interrupt register which
   is controlled from HW and not SW. The compile will therefor not
   find any code that controls this variable, and if not declared as
   volatile, the compile may decided to optimize and remove this variable.
volatile int edge_capture;

/* This is the ISR which will be called when the system signals an interrup
static void handle_interrupts(void* context)
{
    //Cast context to edge_capture's type
    //Volatile to avoid compiler optimization
    //this will point to the edge_capture variable.
    volatile int* edge_capture_ptr = (volatile int*) context;

    //Read the edge capture register on the PIO and store the value
    //The value will be stored in the edge_capture variable and accessible
    //from other parts of the code.
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(INTERRUPT_PIO_BASE)

```

No consoles to display at this time.

Platform designer



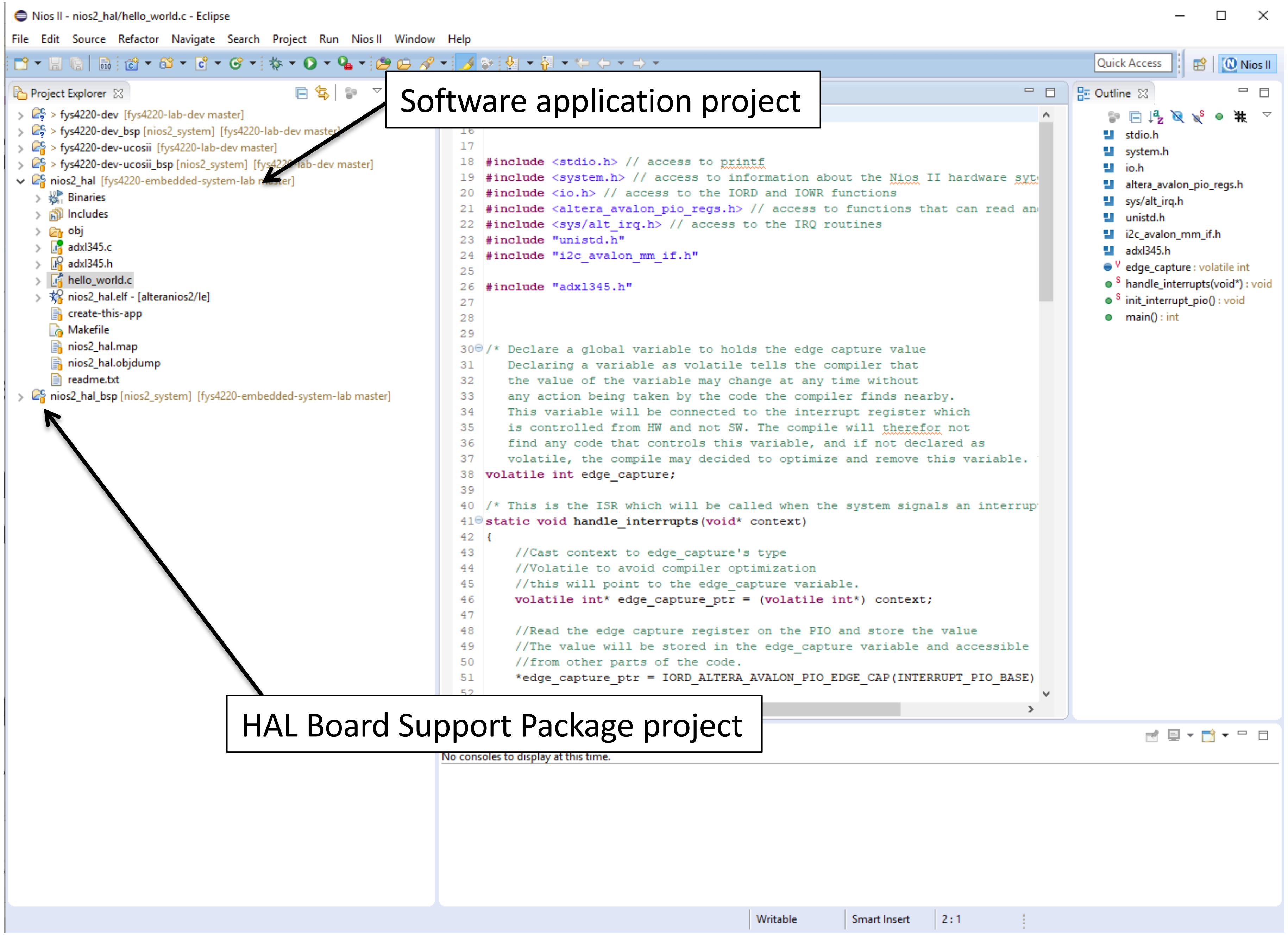
System Contents | Address Map | Interconnect Requirements

System: nios2_system

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		**clk_0**	Clock Source						
		clk_in	Clock Input	**clk**	***exported***				
		clk_in_reset	Reset Input	**reset**					
		clk	Clock Output	*Double-click to export*	clk_0				
		clk_reset	Reset Output	*Double-click to export*					
✓		**cpu**	Nios II Processor						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]				
		irq							
		debug_reset_request							
		debug_mem_slave							
		custom_instruction_master							
✓		**onchip_mem**							
		clk1							
		s1							
		reset1							
✓		**jtag_uart**							
		clk							
		reset							
		avalon_jtag_slave							
		irq							
✓		**sys_clk_timer**							
		clk							
		reset							
		s1							
		irq							
✓		**led_pio**							
		clk							
		reset							
		s1							
		external_connection							
✓		**sw_pio**							
		clk							
		reset							
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0004_1050	0x0004_105f		
		external_connection	Conduit	**sw_pio_ext**					
✓		**interrupt_pio**	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clk]				
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0004_1040	0x0004_104f		
		external_connection	Conduit	**interrupt_pio_ext**					
		irq	Interrupt Sender	*Double-click to export*	[clk]			5	
✓		**i2c_avalon_mm_if**	i2c_avalon_mm_if						
		clock	Clock Input	*Double-click to export*	**clk_0**				
		reset	Reset Input	*Double-click to export*	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0004_1020	0x0004_103f		
		conduit_scl	Conduit	**i2c_avalon_mm_if_scl**	[clock]				
		conduit_sda	Conduit	**i2c_avalon_mm_if_sda**	[clock]				

- Allows a digital system to be designed by interconnecting selected components
- Interconnections are made through the Avalon bus.
- Bus arbitration, bus width matching and even clock domain crossing are all handle automatically when generating the system.

Nios II software build tools (SBT)


Nios II - nios2_hal/hello_world.c - Eclipse
File Edit Source Refactor Navigate Search Project Run Nios II Window Help
Quick Access
Nios II
Project Explorer
Outline
Software application project
HAL Board Support Package project
Writable
Smart Insert
2 : 1

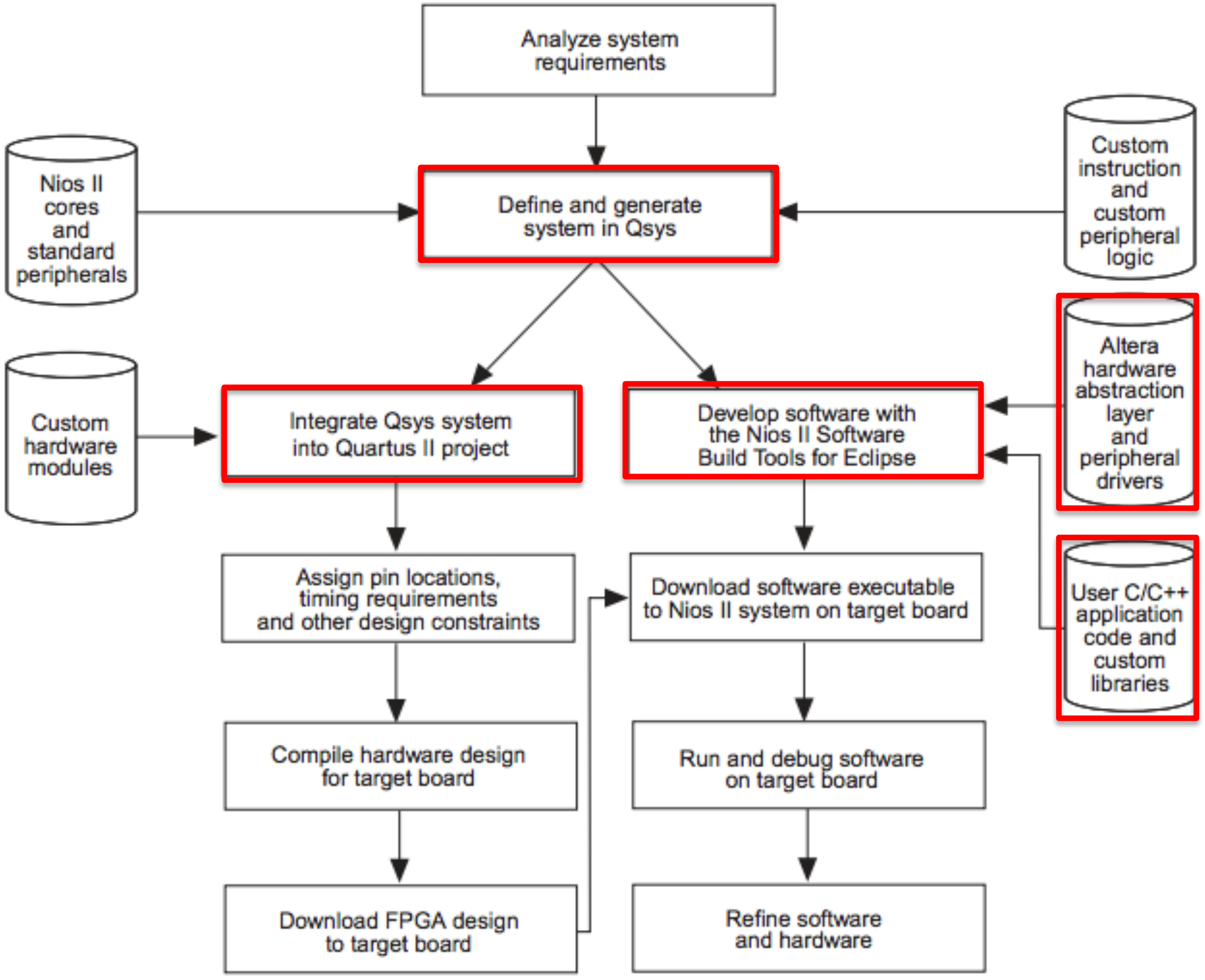
```
> fys4220-dev [fys4220-lab-dev master]
> fys4220-dev_bsp [nios2_system] [fys4220-lab-dev master]
> fys4220-dev-ucosii [fys4220-lab-dev master]
> fys4220-dev-ucosii_bsp [nios2_system] [fys4220-lab-dev master]
nios2_hal [fys4220-embedded-system-lab master]
    Binaries
    Includes
    obj
    adxl345.c
    adxl345.h
    hello_world.c
    nios2_hal.elf - [alteranios2/le]
    create-this-app
    Makefile
    nios2_hal.map
    nios2_hal.objdump
    readme.txt
nios2_hal_bsp [nios2_system] [fys4220-embedded-system-lab master]
```

```c
#include <stdio.h> // access to printf
#include <system.h> // access to information about the Nios II hardware syt
#include <io.h> // access to the IORD and IOWR functions
#include <altera_avalon_pio_regs.h> // access to functions that can read an
#include <sys/alt_irq.h> // access to the IRQ routines
#include "unistd.h"
#include "i2c_avalon_mm_if.h"

#include "adxl345.h"



/* Declare a global variable to holds the edge capture value
   Declaring a variable as volatile tells the compiler that
   the value of the variable may change at any time without
   any action being taken by the code the compiler finds nearby.
   This variable will be connected to the interrupt register which
   is controlled from HW and not SW. The compile will therefor not
   find any code that controls this variable, and if not declared as
   volatile, the compile may decided to optimize and remove this variable.
volatile int edge_capture;

/* This is the ISR which will be called when the system signals an interrup
static void handle_interrupts(void* context)
{
    //Cast context to edge_capture's type
    //Volatile to avoid compiler optimization
    //this will point to the edge_capture variable.
    volatile int* edge_capture_ptr = (volatile int*) context;

    //Read the edge capture register on the PIO and store the value
    //The value will be stored in the edge_capture variable and accessible
    //from other parts of the code.
    *edge_capture_ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(INTERRUPT_PIO_BASE)
```

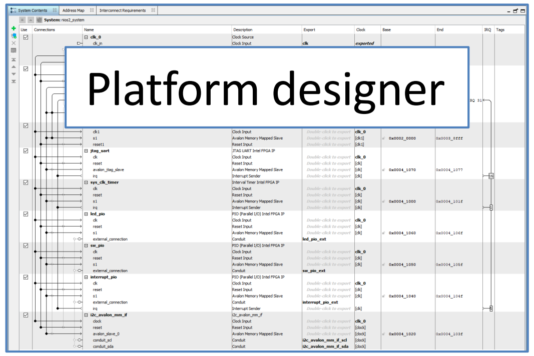
```
stdio.h
system.h
io.h
altera_avalon_pio_regs.h
sys/alt_irq.h
unistd.h
i2c_avalon_mm_if.h
adxl345.h
edge_capture : volatile int
handle_interrupts(void*) : void
init_interrupt_pio() : void
main() : int
```

No consoles to display at this time.

Nios II system development flow


Analyze system requirements
Nios II cores and standard peripherals
Define and generate system in Qsys
Custom instruction and custom peripheral logic
Custom hardware modules
Integrate Qsys system into Quartus II project
Develop software with the Nios II Software Build Tools for Eclipse
Altera hardware abstraction layer and peripheral drivers
Assign pin locations, timing requirements and other design constraints
Download software executable to Nios II system on target board
User C/C++ application code and custom libraries
Compile hardware design for target board
Run and debug software on target board
Download FPGA design to target board
Refine software and hardware

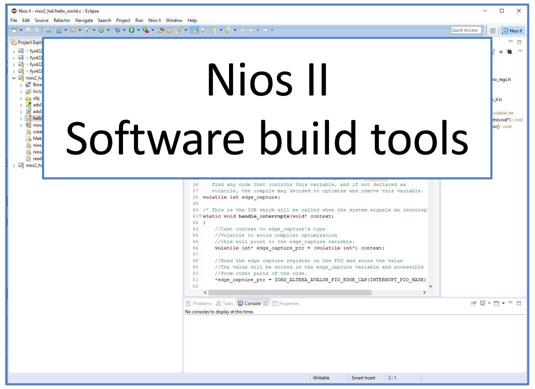
Hardware and Software development flow



Generates hardware system

(.vhd + .sopcinfo)

Eextract system information from the SOPC information file (.sopcinfo).



Generates a custom HAL board support package (BSP) specific to your hardware configuration

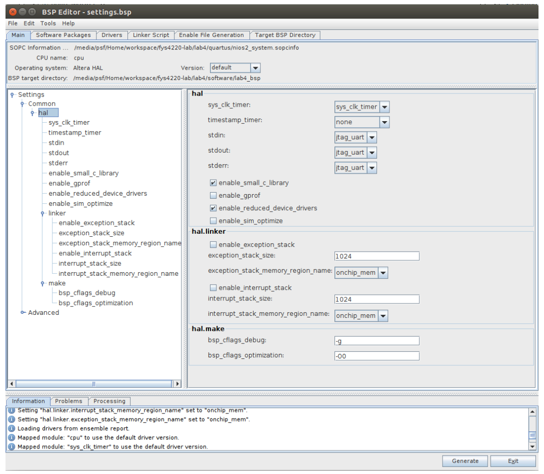
(settings.bsp + system.h)

System.h:
Complete software description of the NIOS II system

Generate SW programming file

(.elf)

BSP Editor



BSP Editor - settings.bsp

File Edit Tools Help

Main | Software Packages | Drivers | Linker Script | Enable File Generation | Target BSP Directory

SOPC Information ... /media/psf/Home/workspace/fys4220-lab/lab4/quartus/nios2_system.sopcinfo
CPU name: cpu
Operating system: Altera HAL   Version: default
BSP target directory: /media/psf/Home/workspace/fys4220-lab/lab4/software/lab4_bsp

```
Settings
  Common
    hal
      sys_clk_timer
      timestamp_timer
      stdin
      stdout
      stderr
      enable_small_c_library
      enable_gprof
      enable_reduced_device_drivers
      enable_sim_optimize
      linker
        enable_exception_stack
        exception_stack_size
        exception_stack_memory_region_name
        enable_interrupt_stack
        interrupt_stack_size
        interrupt_stack_memory_region_name
      make
        bsp_cflags_debug
        bsp_cflags_optimization
  Advanced
```

hal

sys_clk_timer: sys_clk_timer
timestamp_timer: none
stdin: jtag_uart
stdout: jtag_uart
stderr: jtag_uart
☑ enable_small_c_library
☐ enable_gprof
☑ enable_reduced_device_drivers
☐ enable_sim_optimize

hal.linker

☐ enable_exception_stack
exception_stack_size: 1024
exception_stack_memory_region_name: onchip_mem
☐ enable_interrupt_stack
interrupt_stack_size: 1024
interrupt_stack_memory_region_name: onchip_mem

hal.make

bsp_cflags_debug: -g
bsp_cflags_optimization: -O0

Information | Problems | Processing

```
Setting "hal.linker.interrupt_stack_memory_region_name" set to "onchip_mem".
Setting "hal.linker.exception_stack_memory_region_name" set to "onchip_mem".
Loading drivers from ensemble report.
Mapped module: "cpu" to use the default driver version.
Mapped module: "sys_clk_timer" to use the default driver version.
```

Generate | Exit

Hardware Abstraction Layer (HAL)

- HAL must be based on a specific hardware system

- HAL library generation (Nios II):
  - Platform designer generates a hardware system (.vhd + .sopcinfo)
  - NIOS II software Build Tools (SBT) extract system information from the SOPC information file (.sopcinfo).
  - NIOS II Software Build Tools (SBT) generates a custom HAL board support package (BSP) specific to your hardware configuration.
    - System.h: Complete software description of the NIOS II system
  - Changes in the hardware configuration automatically propagate to the HAL device driver configuration when the BSP is re-generated.

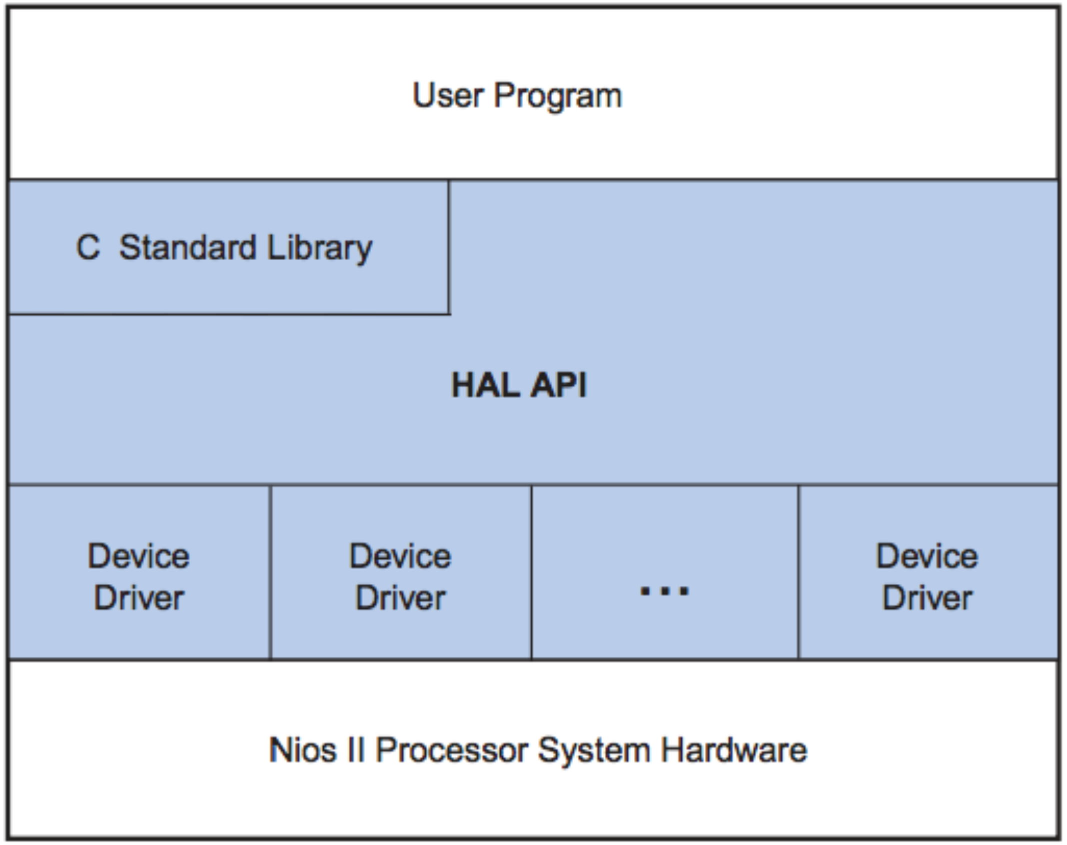
- HAL device driver abstraction provides a clear distinction between application and device driver software.
  - Promotes reusable application code that is resistant to changes in the underlying hardware.

SOPC: System On Programmable Chip

Nios II HAL

- Lightweight embedded runtime environment that provides a simple device driver interface for programs to connect to the underlying hardware.
- NIOS II HAL application program interface (API) is integrated with the ***newlib ANSI C standard library*.**
- Newlib intended for use with embedded system that lack any kind of operating system.
  - Use HW independent parts of the standard C-library
  - Rely on calls to Board Support Package (BSP) for HW specific information
- HAL allows to access devices and files using familiar C library functions such as e.g. printf()

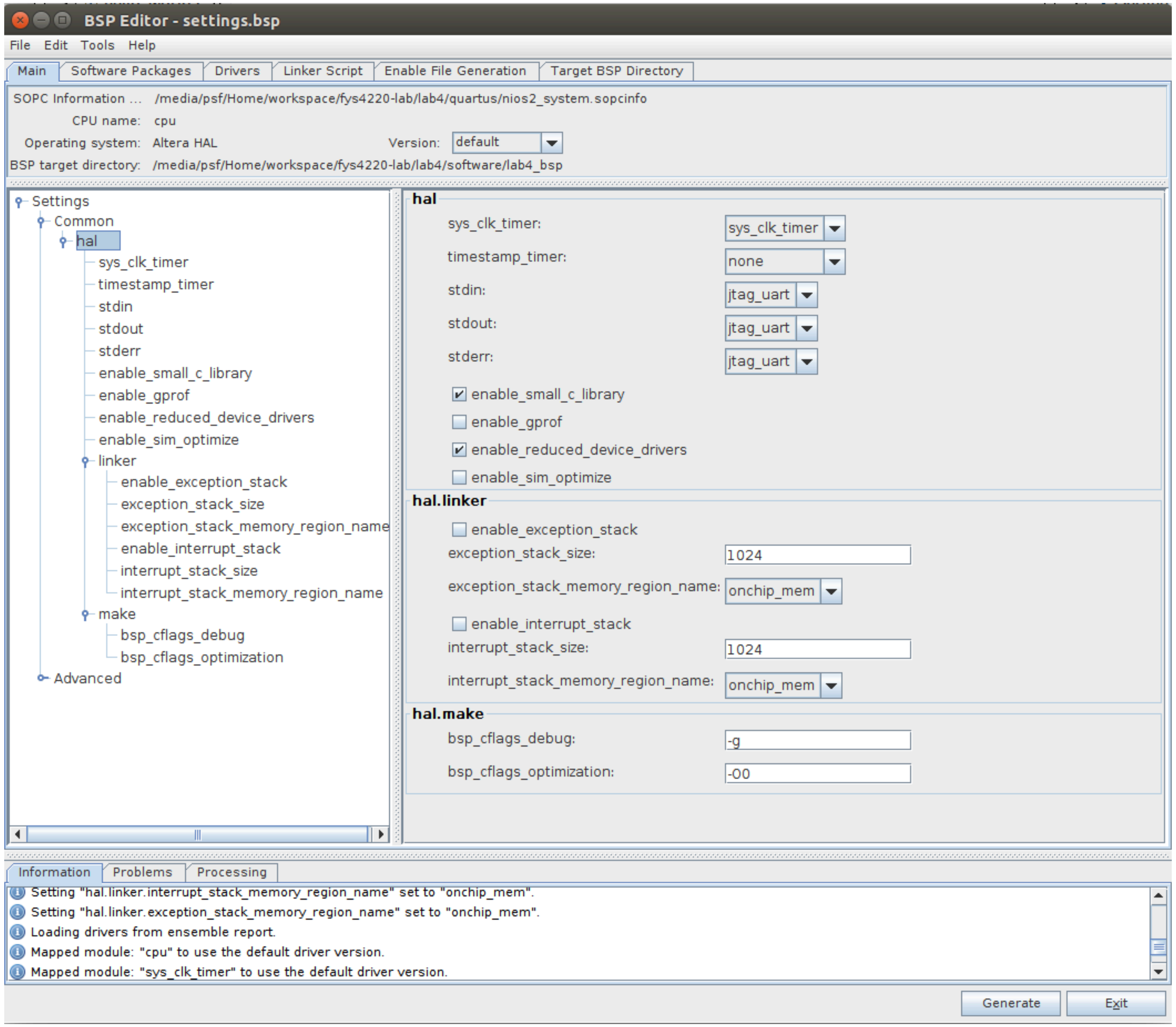
Chap 6-8 in Nios II Software developer's handbook:
https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf



```
#include <stdio.h>
```

Newlib: https://sourceware.org/newlib/

BSP Editor


BSP Editor - settings.bsp
File Edit Tools Help
Main | Software Packages | Drivers | Linker Script | Enable File Generation | Target BSP Directory

SOPC Information ... /media/psf/Home/workspace/fys4220-lab/lab4/quartus/nios2_system.sopcinfo
CPU name: cpu
Operating system: Altera HAL Version: default
BSP target directory: /media/psf/Home/workspace/fys4220-lab/lab4/software/lab4_bsp

```
Settings
  Common
    hal
      sys_clk_timer
      timestamp_timer
      stdin
      stdout
      stderr
      enable_small_c_library
      enable_gprof
      enable_reduced_device_drivers
      enable_sim_optimize
      linker
        enable_exception_stack
        exception_stack_size
        exception_stack_memory_region_name
        enable_interrupt_stack
        interrupt_stack_size
        interrupt_stack_memory_region_name
      make
        bsp_cflags_debug
        bsp_cflags_optimization
  Advanced
```

hal

sys_clk_timer:	sys_clk_timer
timestamp_timer:	none
stdin:	jtag_uart
stdout:	jtag_uart
stderr:	jtag_uart

[✓] enable_small_c_library
[ ] enable_gprof
[✓] enable_reduced_device_drivers
[ ] enable_sim_optimize

hal.linker

[ ] enable_exception_stack

exception_stack_size:	1024
exception_stack_memory_region_name:	onchip_mem

[ ] enable_interrupt_stack

interrupt_stack_size:	1024
interrupt_stack_memory_region_name:	onchip_mem

hal.make

bsp_cflags_debug:	-g
bsp_cflags_optimization:	-O0

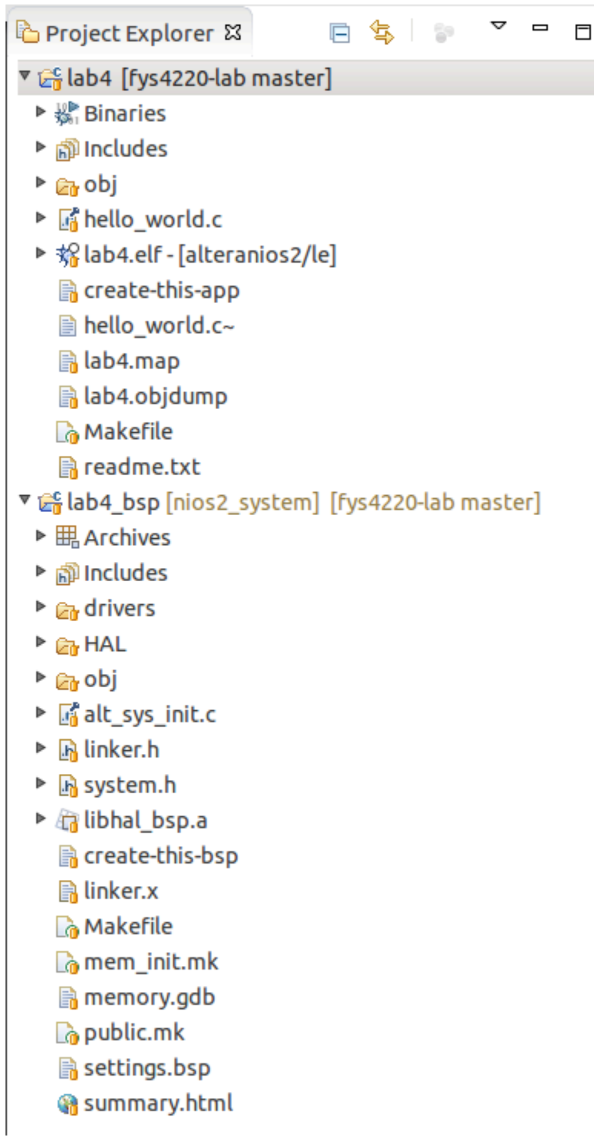
```
Setting "hal.linker.interrupt_stack_memory_region_name" set to "onchip_mem".
Setting "hal.linker.exception_stack_memory_region_name" set to "onchip_mem".
Loading drivers from ensemble report.
Mapped module: "cpu" to use the default driver version.
Mapped module: "sys_clk_timer" to use the default driver version.
```

HAL settings are reflected in the system.h file

NIOS II SW Development

- Each NIOS II program consists of:
  – an application project,
  – optional user library projects, and
  – a board support package (BSP) project

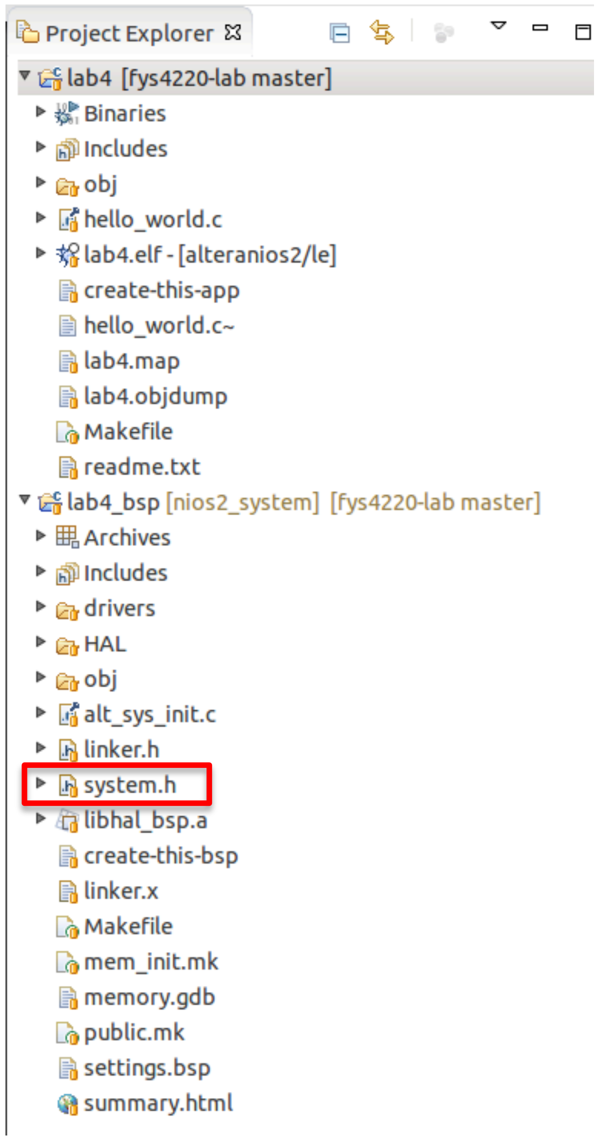
- The build process creates an Executable and Linking Format File (**.elf**) which runs on a NIOS II processor



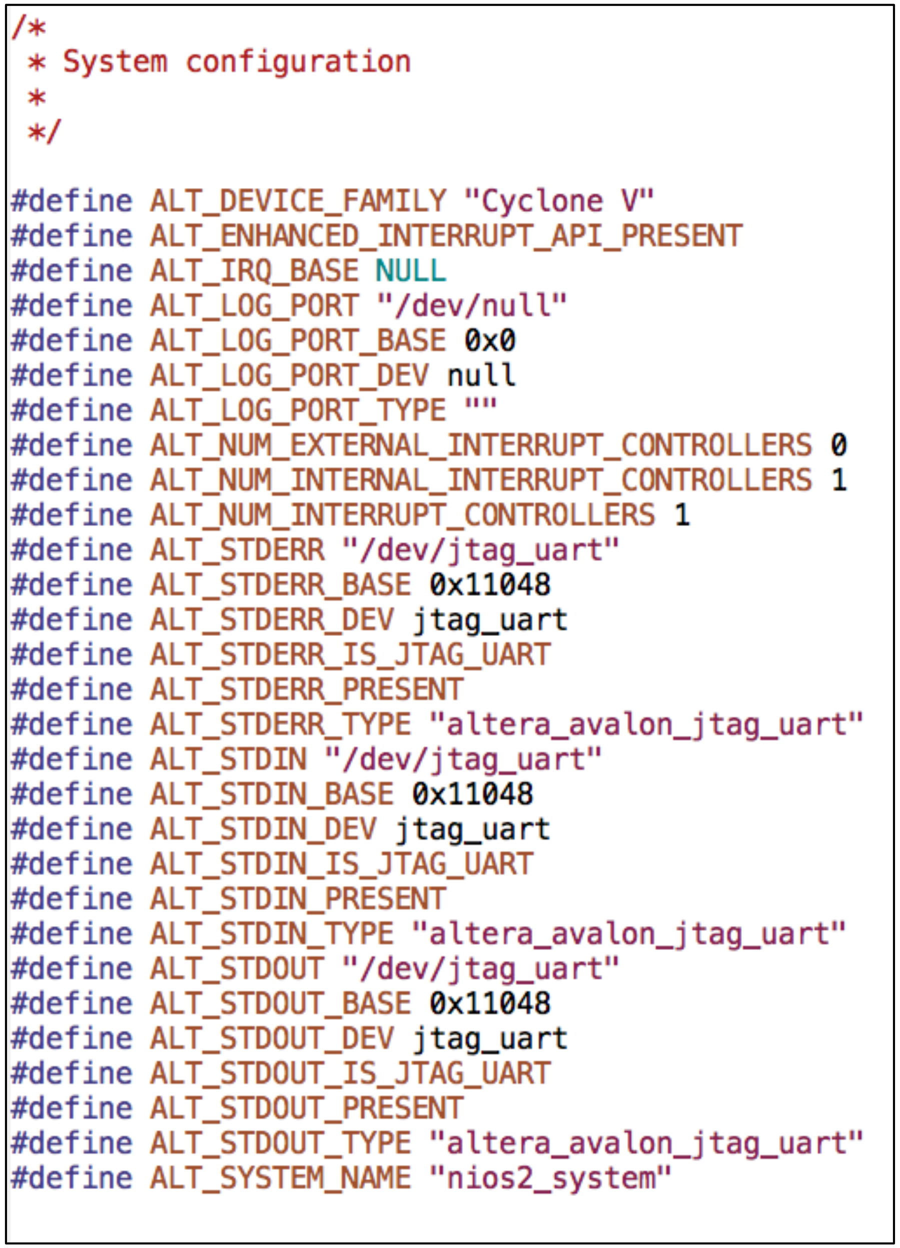
```
Project Explorer
▼ lab4 [fys4220-lab master]
  ▶ Binaries
  ▶ Includes
  ▶ obj
  ▶ hello_world.c
  ▶ lab4.elf - [alteranios2/le]
    create-this-app
    hello_world.c~
    lab4.map
    lab4.objdump
    Makefile
    readme.txt
▼ lab4_bsp [nios2_system] [fys4220-lab master]
  ▶ Archives
  ▶ Includes
  ▶ drivers
  ▶ HAL
  ▶ obj
  ▶ alt_sys_init.c
  ▶ linker.h
  ▶ system.h
  ▶ libhal_bsp.a
    create-this-bsp
    linker.x
    Makefile
    mem_init.mk
    memory.gdb
    public.mk
    settings.bsp
    summary.html
```

#include <system.h>

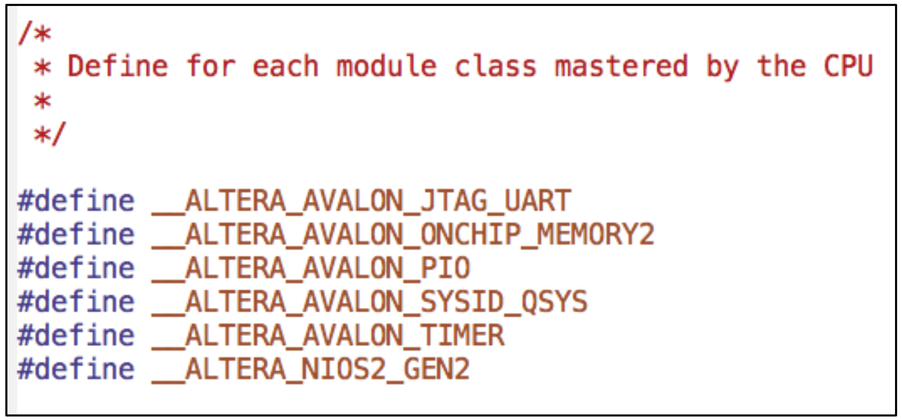
- Provides a complete software description of the NIOS II system hardware
- Describes each peripheral in the system
  - The hardware configuration of peripheral
  - The base address
  - Interrupt request (IRQ) information (if any)
  - A symbolic name for the peripheral
- NIOS II SBT generates system.h file for HAL BSP projects
- Do not edit system.h !!



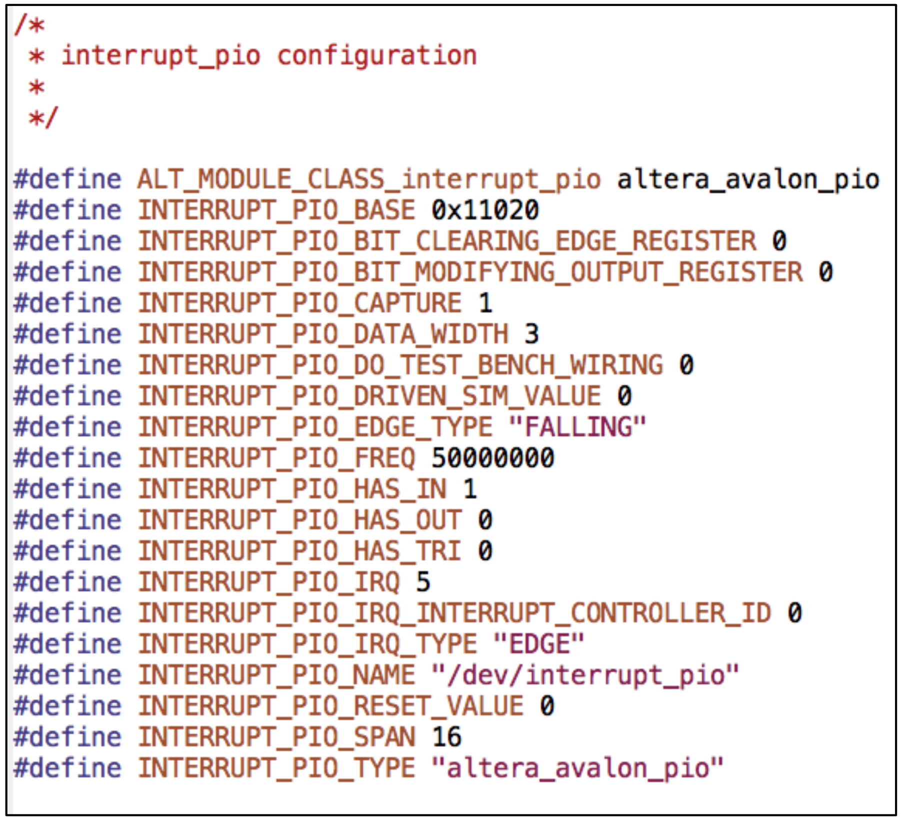
Ex. from a system.h



```
/*
 * System configuration
 *
 */

#define ALT_DEVICE_FAMILY "Cyclone V"
#define ALT_ENHANCED_INTERRUPT_API_PRESENT
#define ALT_IRQ_BASE NULL
#define ALT_LOG_PORT "/dev/null"
#define ALT_LOG_PORT_BASE 0x0
#define ALT_LOG_PORT_DEV null
#define ALT_LOG_PORT_TYPE ""
#define ALT_NUM_EXTERNAL_INTERRUPT_CONTROLLERS 0
#define ALT_NUM_INTERNAL_INTERRUPT_CONTROLLERS 1
#define ALT_NUM_INTERRUPT_CONTROLLERS 1
#define ALT_STDERR "/dev/jtag_uart"
#define ALT_STDERR_BASE 0x11048
#define ALT_STDERR_DEV jtag_uart
#define ALT_STDERR_IS_JTAG_UART
#define ALT_STDERR_PRESENT
#define ALT_STDERR_TYPE "altera_avalon_jtag_uart"
#define ALT_STDIN "/dev/jtag_uart"
#define ALT_STDIN_BASE 0x11048
#define ALT_STDIN_DEV jtag_uart
#define ALT_STDIN_IS_JTAG_UART
#define ALT_STDIN_PRESENT
#define ALT_STDIN_TYPE "altera_avalon_jtag_uart"
#define ALT_STDOUT "/dev/jtag_uart"
#define ALT_STDOUT_BASE 0x11048
#define ALT_STDOUT_DEV jtag_uart
#define ALT_STDOUT_IS_JTAG_UART
#define ALT_STDOUT_PRESENT
#define ALT_STDOUT_TYPE "altera_avalon_jtag_uart"
#define ALT_SYSTEM_NAME "nios2_system"
```

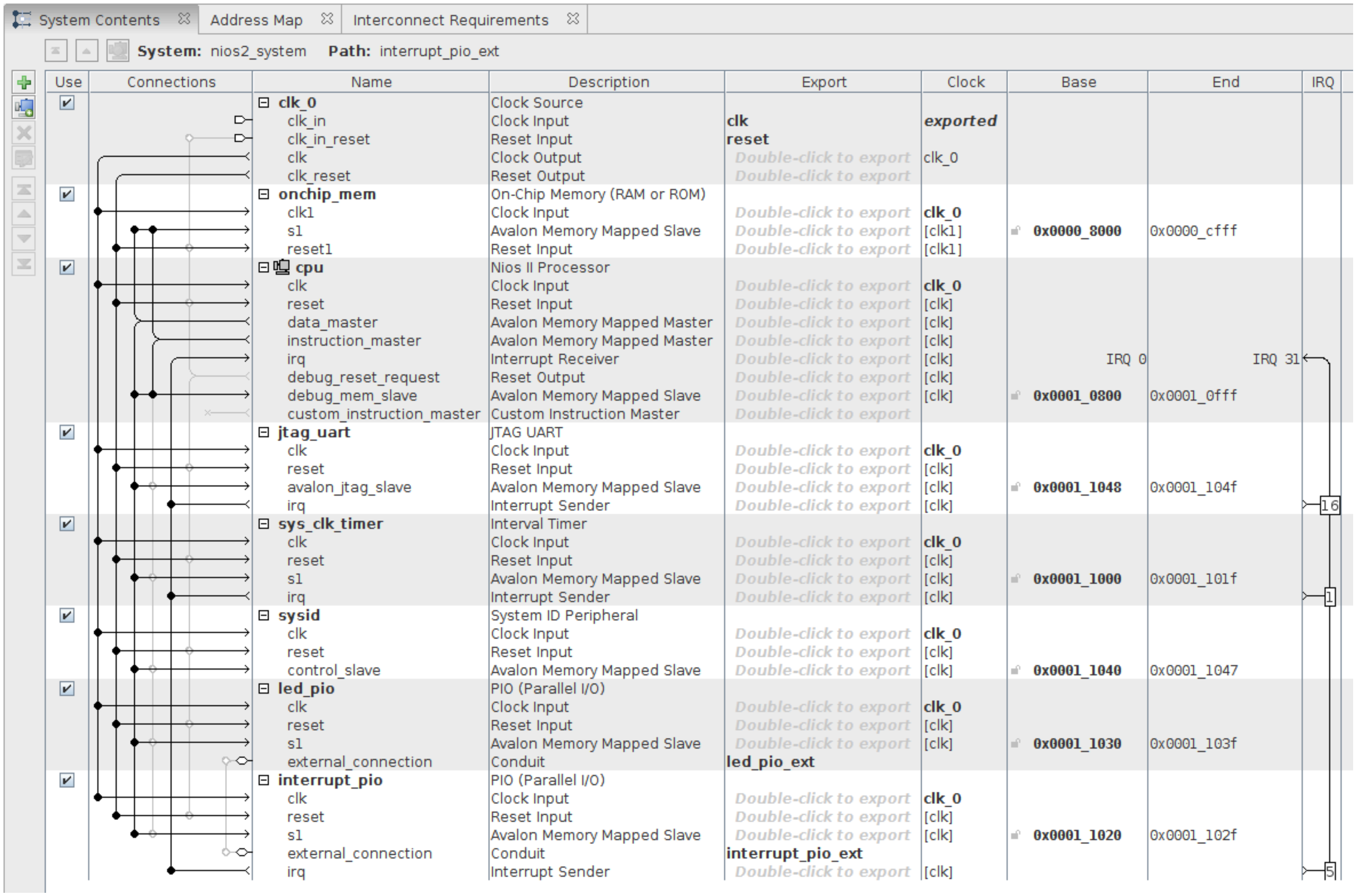


```
/*
 * Define for each module class mastered by the CPU
 *
 */

#define __ALTERA_AVALON_JTAG_UART
#define __ALTERA_AVALON_ONCHIP_MEMORY2
#define __ALTERA_AVALON_PIO
#define __ALTERA_AVALON_SYSID_QSYS
#define __ALTERA_AVALON_TIMER
#define __ALTERA_NIOS2_GEN2
```



```
/*
 * interrupt_pio configuration
 *
 */

#define ALT_MODULE_CLASS_interrupt_pio altera_avalon_pio
#define INTERRUPT_PIO_BASE 0x11020
#define INTERRUPT_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define INTERRUPT_PIO_BIT_MODIFYING_OUTPUT_REGISTER 0
#define INTERRUPT_PIO_CAPTURE 1
#define INTERRUPT_PIO_DATA_WIDTH 3
#define INTERRUPT_PIO_DO_TEST_BENCH_WIRING 0
#define INTERRUPT_PIO_DRIVEN_SIM_VALUE 0
#define INTERRUPT_PIO_EDGE_TYPE "FALLING"
#define INTERRUPT_PIO_FREQ 50000000
#define INTERRUPT_PIO_HAS_IN 1
#define INTERRUPT_PIO_HAS_OUT 0
#define INTERRUPT_PIO_HAS_TRI 0
#define INTERRUPT_PIO_IRQ 5
#define INTERRUPT_PIO_IRQ_INTERRUPT_CONTROLLER_ID 0
#define INTERRUPT_PIO_IRQ_TYPE "EDGE"
#define INTERRUPT_PIO_NAME "/dev/interrupt_pio"
#define INTERRUPT_PIO_RESET_VALUE 0
#define INTERRUPT_PIO_SPAN 16
#define INTERRUPT_PIO_TYPE "altera_avalon_pio"
```

Nios II hardware development



System Contents | Address Map | Interconnect Requirements

System: nios2_system **Path:** interrupt_pio_ext

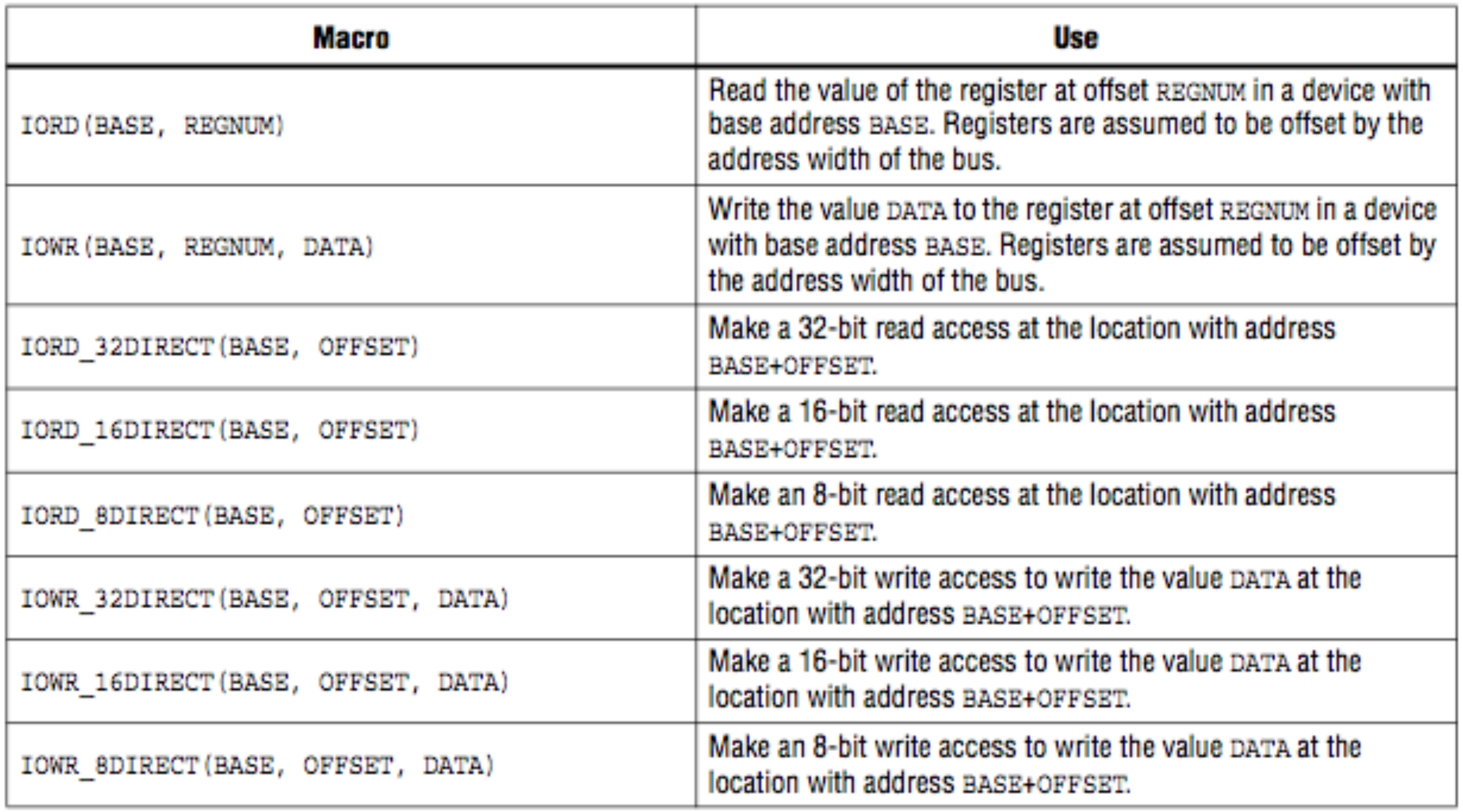
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		**clk_0**	Clock Source					
		clk_in	Clock Input	**clk**	***exported***			
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	*Double-click to export*	clk_0			
		clk_reset	Reset Output	*Double-click to export*				
✓		**onchip_mem**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	*Double-click to export*	**clk_0**			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk1]	**0x0000_8000**	0x0000_cfff	
		reset1	Reset Input	*Double-click to export*	[clk1]			
✓		**cpu**	Nios II Processor					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		data_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		instruction_master	Avalon Memory Mapped Master	*Double-click to export*	[clk]			
		irq	Interrupt Receiver	*Double-click to export*	[clk]	IRQ 0	IRQ 31	
		debug_reset_request	Reset Output	*Double-click to export*	[clk]			
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_0800**	0x0001_0fff	
		custom_instruction_master	Custom Instruction Master	*Double-click to export*				
✓		**jtag_uart**	JTAG UART					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_1048**	0x0001_104f	
		irq	Interrupt Sender	*Double-click to export*	[clk]			16
✓		**sys_clk_timer**	Interval Timer					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_1000**	0x0001_101f	
		irq	Interrupt Sender	*Double-click to export*	[clk]			1
✓		**sysid**	System ID Peripheral					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_1040**	0x0001_1047	
✓		**led_pio**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_1030**	0x0001_103f	
		external_connection	Conduit	**led_pio_ext**				
✓		**interrupt_pio**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to export*	**clk_0**			
		reset	Reset Input	*Double-click to export*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	**0x0001_1020**	0x0001_102f	
		external_connection	Conduit	**interrupt_pio_ext**				
		irq	Interrupt Sender	*Double-click to export*	[clk]			5

Accessing HAL peripherals

Useful HAL macros

#include <io.h>

- Provides C language macros IORD and IOWR
- Enables HAL device drivers to access hardware registers
- Components can easily be moved to different address areas without changing the software



Macro	Use
`IORD(BASE, REGNUM)`	Read the value of the register at offset REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus.
`IOWR(BASE, REGNUM, DATA)`	Write the value DATA to the register at offset REGNUM in a device with base address BASE. Registers are assumed to be offset by the address width of the bus.
`IORD_32DIRECT(BASE, OFFSET)`	Make a 32-bit read access at the location with address BASE+OFFSET.
`IORD_16DIRECT(BASE, OFFSET)`	Make a 16-bit read access at the location with address BASE+OFFSET.
`IORD_8DIRECT(BASE, OFFSET)`	Make an 8-bit read access at the location with address BASE+OFFSET.
`IOWR_32DIRECT(BASE, OFFSET, DATA)`	Make a 32-bit write access to write the value DATA at the location with address BASE+OFFSET.
`IOWR_16DIRECT(BASE, OFFSET, DATA)`	Make a 16-bit write access to write the value DATA at the location with address BASE+OFFSET.
`IOWR_8DIRECT(BASE, OFFSET, DATA)`	Make an 8-bit write access to write the value DATA at the location with address BASE+OFFSET.

#include <io.h>

- IORD() / IOWR()
  - Offset is the word offset of the register
  - Word size assumed to be 32-bit so offsets 0,1,2,3 etc, maps to byte offsets 0,4,8,12

- IORD_xxDIRECT() / IOWR_xxDIRECT()
  - Data size oriented
  - Offset is in bytes and choice of macro dictates the width of the access.
  - Can be used to access slave ports that contains byte enables and has multiple values stored in a single wide register

HAL macros

IORD (BASE, 0x1);

3	2	1	0
7	**6**	**5**	**4**

Word oriented

IORD_32DIRECT (BASE, 0x4);

3	2	1	0
7	**6**	**5**	**4**

IORD_16DIRECT(BASE, 0x2);

3	**2**	1	0
7	6	5	4

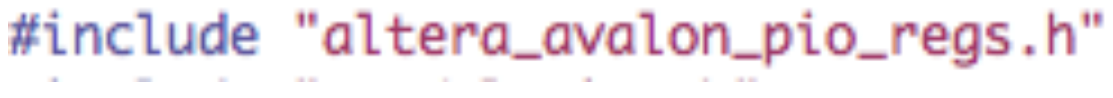
IORD_8DIRECT(BASE, 0x7);

3	2	1	0
7	6	5	4

byte oriented

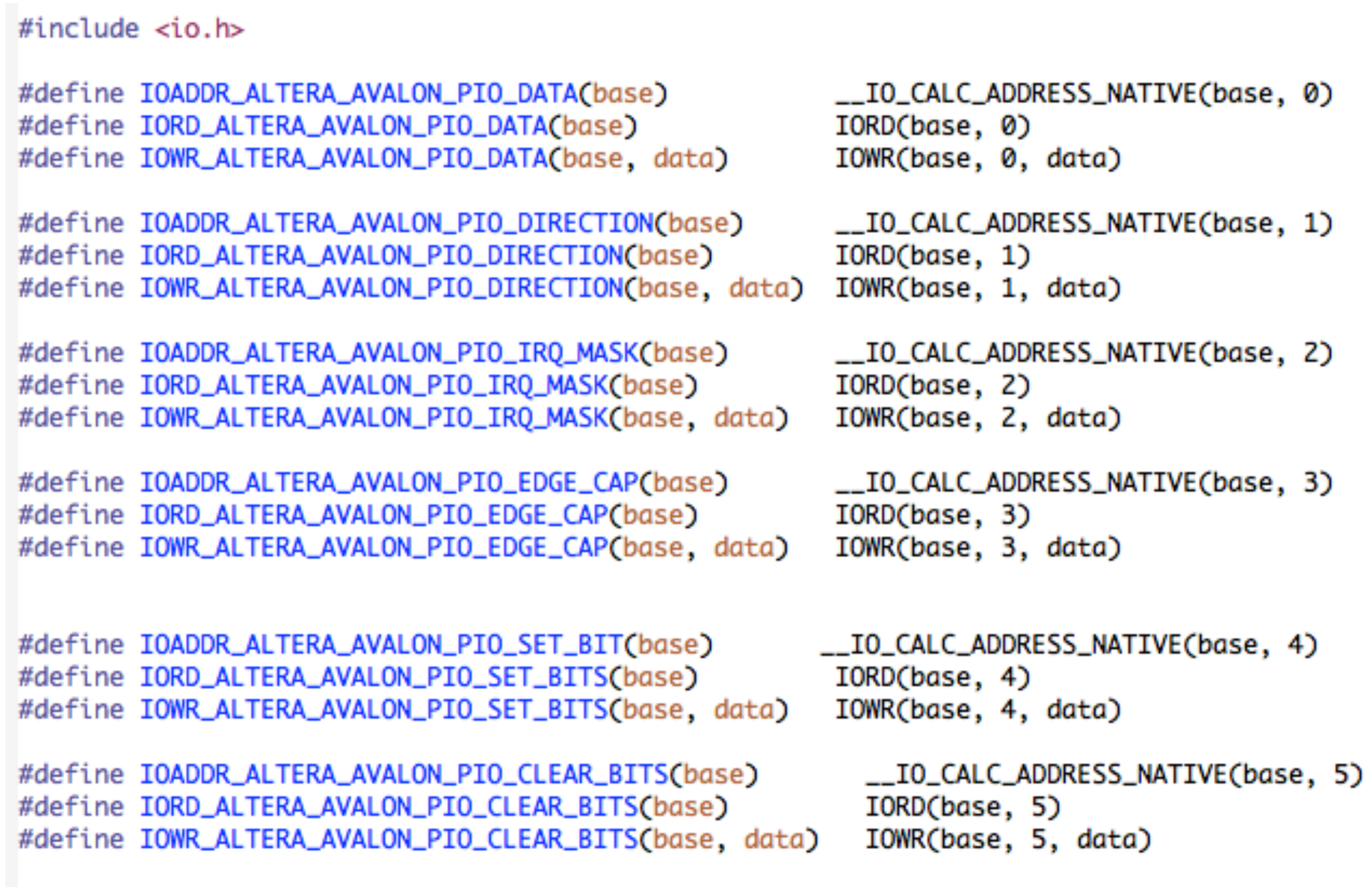
HAL Peripherals

- All peripherals must have a header file that defines the peripheral’s low-level interface to hardware
- Therefore, all peripherals support the HAL to some extent
- However, some peripherals might not provide device drivers
- If drivers are not available, use only the definitions provided in the header files to access the hardware
- Some peripherals provide functions that are not based on the HAL generic device models
  - For example, Altera provides a general-purpose parallel I/O (PIO) core for use with the NIOS II processor system
  - The PIO peripheral does not fit in any class of generic device models provided by the HAL, and so it provides a header file and a few dedicated functions only



```
#include "altera_avalon_pio_regs.h"
```

PIO macros in altera_avalon_pio_regs.h



```c
#include <io.h>

#define IOADDR_ALTERA_AVALON_PIO_DATA(base)           __IO_CALC_ADDRESS_NATIVE(base, 0)
#define IORD_ALTERA_AVALON_PIO_DATA(base)             IORD(base, 0)
#define IOWR_ALTERA_AVALON_PIO_DATA(base, data)       IOWR(base, 0, data)

#define IOADDR_ALTERA_AVALON_PIO_DIRECTION(base)      __IO_CALC_ADDRESS_NATIVE(base, 1)
#define IORD_ALTERA_AVALON_PIO_DIRECTION(base)        IORD(base, 1)
#define IOWR_ALTERA_AVALON_PIO_DIRECTION(base, data)  IOWR(base, 1, data)

#define IOADDR_ALTERA_AVALON_PIO_IRQ_MASK(base)       __IO_CALC_ADDRESS_NATIVE(base, 2)
#define IORD_ALTERA_AVALON_PIO_IRQ_MASK(base)         IORD(base, 2)
#define IOWR_ALTERA_AVALON_PIO_IRQ_MASK(base, data)   IOWR(base, 2, data)

#define IOADDR_ALTERA_AVALON_PIO_EDGE_CAP(base)       __IO_CALC_ADDRESS_NATIVE(base, 3)
#define IORD_ALTERA_AVALON_PIO_EDGE_CAP(base)         IORD(base, 3)
#define IOWR_ALTERA_AVALON_PIO_EDGE_CAP(base, data)   IOWR(base, 3, data)


#define IOADDR_ALTERA_AVALON_PIO_SET_BIT(base)       __IO_CALC_ADDRESS_NATIVE(base, 4)
#define IORD_ALTERA_AVALON_PIO_SET_BITS(base)         IORD(base, 4)
#define IOWR_ALTERA_AVALON_PIO_SET_BITS(base, data)   IOWR(base, 4, data)

#define IOADDR_ALTERA_AVALON_PIO_CLEAR_BITS(base)       __IO_CALC_ADDRESS_NATIVE(base, 5)
#define IORD_ALTERA_AVALON_PIO_CLEAR_BITS(base)         IORD(base, 5)
#define IOWR_ALTERA_AVALON_PIO_CLEAR_BITS(base, data)   IOWR(base, 5, data)
```

PIO register map

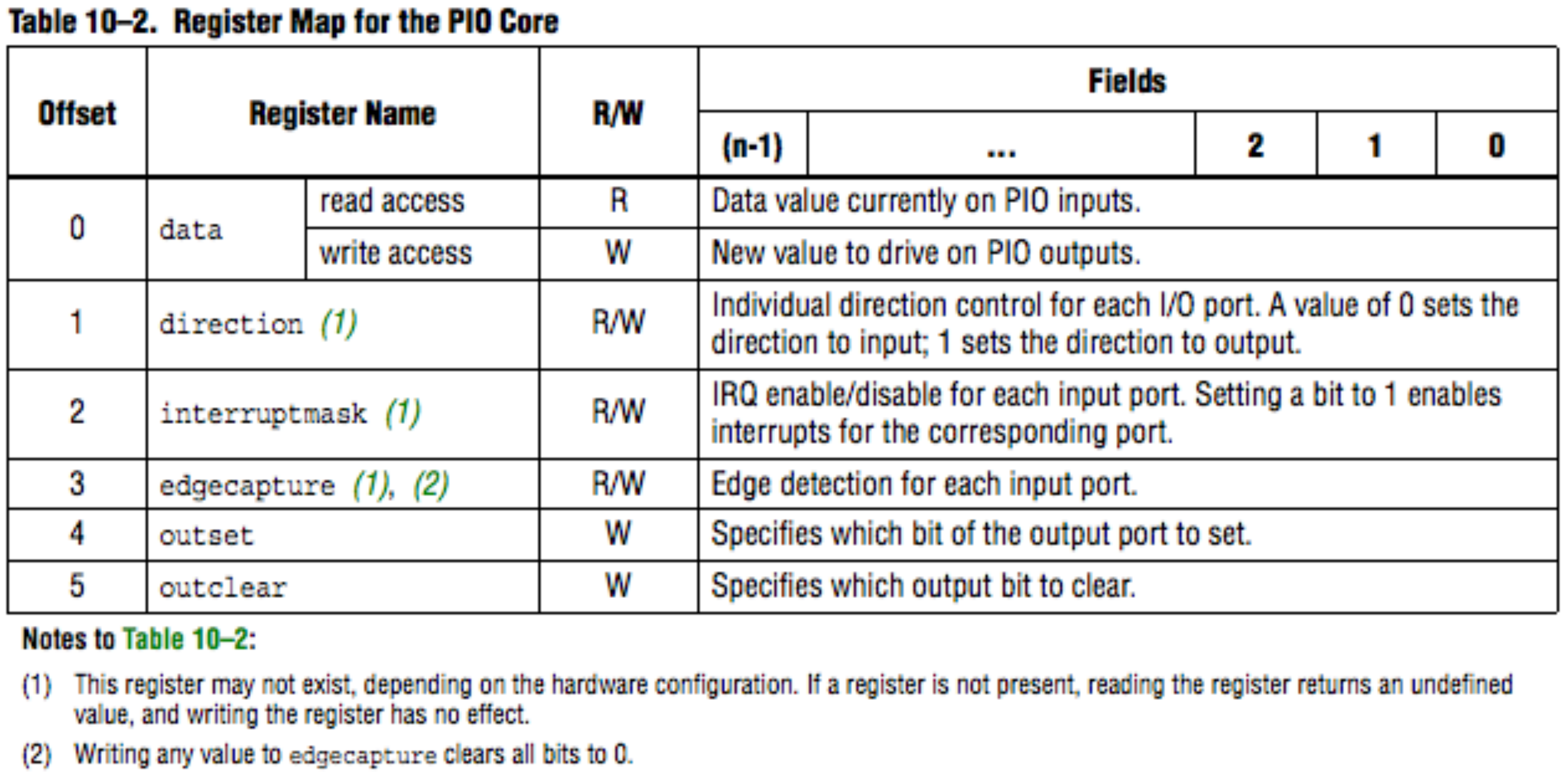


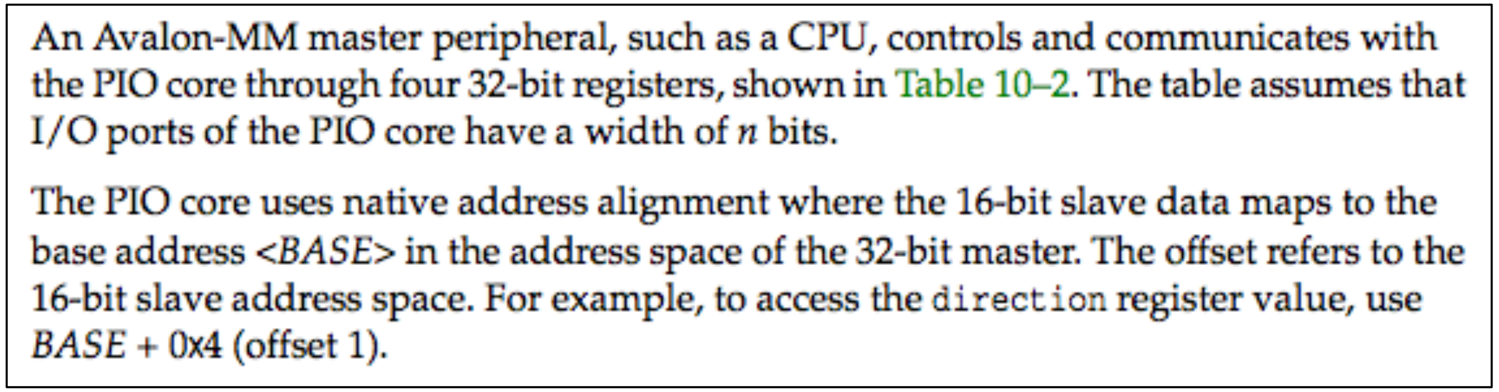
Table 10–2. Register Map for the PIO Core

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction (1)		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask (1)		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture (1), (2)		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

Notes to Table 10–2:

(1) This register may not exist, depending on the hardware configuration. If a register is not present, reading the register returns an undefined value, and writing the register has no effect.

(2) Writing any value to `edgecapture` clears all bits to 0.



An Avalon-MM master peripheral, such as a CPU, controls and communicates with the PIO core through four 32-bit registers, shown in Table 10–2. The table assumes that I/O ports of the PIO core have a width of *n* bits.

The PIO core uses native address alignment where the 16-bit slave data maps to the base address <*BASE*> in the address space of the 32-bit master. The offset refers to the 16-bit slave address space. For example, to access the `direction` register value, use *BASE* + 0x4 (offset 1).

Source: http://www.altera.com/literature/ug/ug_embedded_ip.pdf