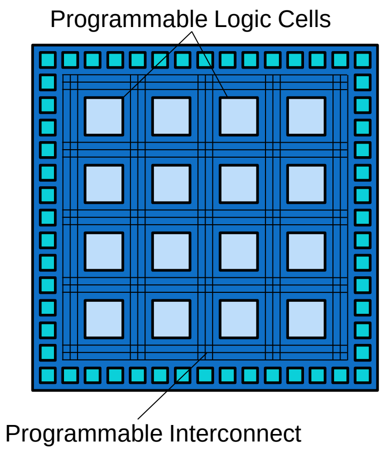
Lab Environment
Altera DE2 Board, FPGAs, Nios II

Ingo Sander
ingo@kth.se

1

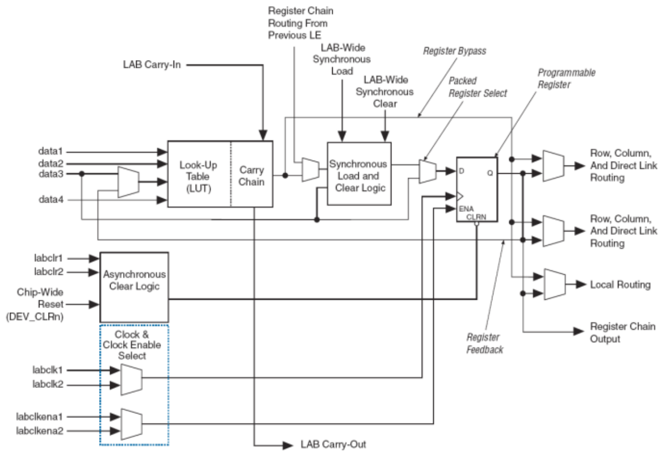
Field Programmable Gate Arrays

- An FPGA consists of programmable logic cells and a programmable interconnect
- The logic cells are configurable digital hardware blocks
- If large enough, the FPGA can be used to implement any digital functionality

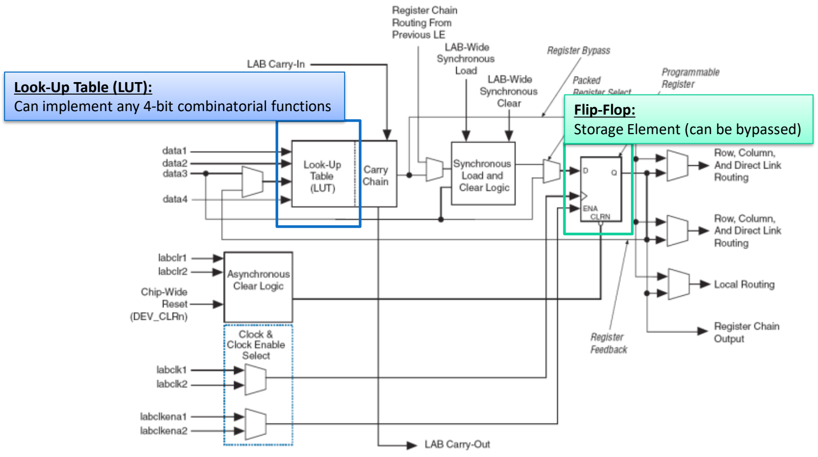


2

Altera Cyclone II FPGA Logic Element



Altera Cyclone II FPGA Logic Element



Look-Up Table (LUT):
Can implement any 4-bit combinatorial functions
Flip-Flop:
Storage Element (can be bypassed)
Register Chain Routing From Previous LE
LAB-Wide Synchronous Load
LAB-Wide Synchronous Clear
Register Bypass
Packed Register Select
Programmable Register
LAB Carry-In
data1
data2
data3
data4
Look-Up Table (LUT)
Carry Chain
Synchronous Load and Clear Logic
D
Q
ENA
CLRN
Row, Column, And Direct Link Routing
Row, Column, And Direct Link Routing
Local Routing
Register Chain Output
labclr1
labclr2
Chip-Wide Reset (DEV_CLRn)
Asynchronous Clear Logic
Clock & Clock Enable Select
labclk1
labclk2
labclkena1
labclkena2
Register Feedback
LAB Carry-Out

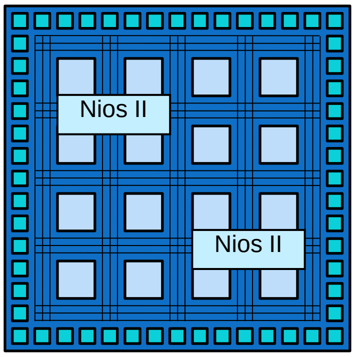
FPGAs: Not only logic cells

- In addition to logic cells FPGAs often also contain
  - memory elements
  - multipliers
  - other resources

Nios II Cores can be implemented on an FPGA

- The Nios II processor can be implemented on an Altera FPGA, if enough logic cells, memory elements, multipliers, and interconnection resources are available

- More than one processor and additional peripherals can be implemented on an FPGA

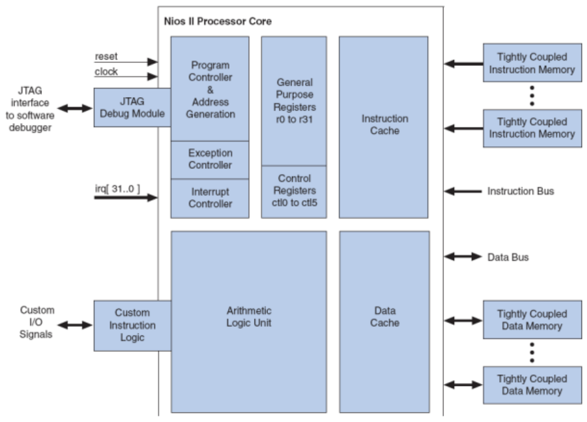


Powerful parallel computer systems become possible!

Nios II Processor Overview

- RISC (Reduced Instruction Set Computer Architecture)
- 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- Instruction and data caches
- Single instruction 32x32 bit multiply and divide instruction, result is 32 bit
- Configurable processor
- Custom Instructions: Additional instructions can be defined using extra hardware

Nios II Block Diagram

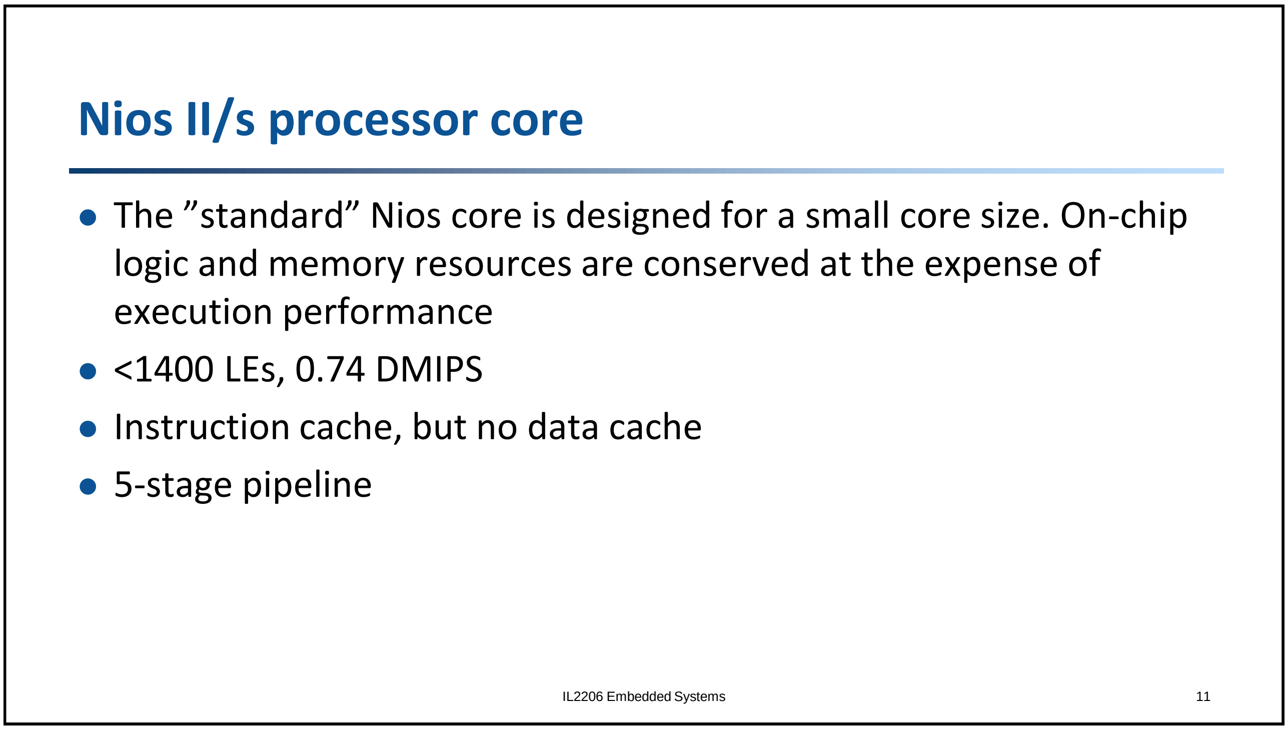

Nios II Processor Core
reset
clock
JTAG interface to software debugger
JTAG Debug Module
irq[31..0]
Program Controller & Address Generation
Exception Controller
Interrupt Controller
General Purpose Registers r0 to r31
Control Registers ctl0 to ctl5
Instruction Cache
Tightly Coupled Instruction Memory
Tightly Coupled Instruction Memory
Instruction Bus
Data Bus
Custom I/O Signals
Custom Instruction Logic
Arithmetic Logic Unit
Data Cache
Tightly Coupled Data Memory
Tightly Coupled Data Memory

Nios II platforms

- When creating a Nios II platform the designer can
  - choose a special type of Nios II processor (II/e, II/s, II/f)
  - choose the amount of cache memory and on-chip memory
  - choose peripheral circuits and external memories that will be integrated into the core
  - code is in general portable from one Nios core to another

The Nios II/f processor core

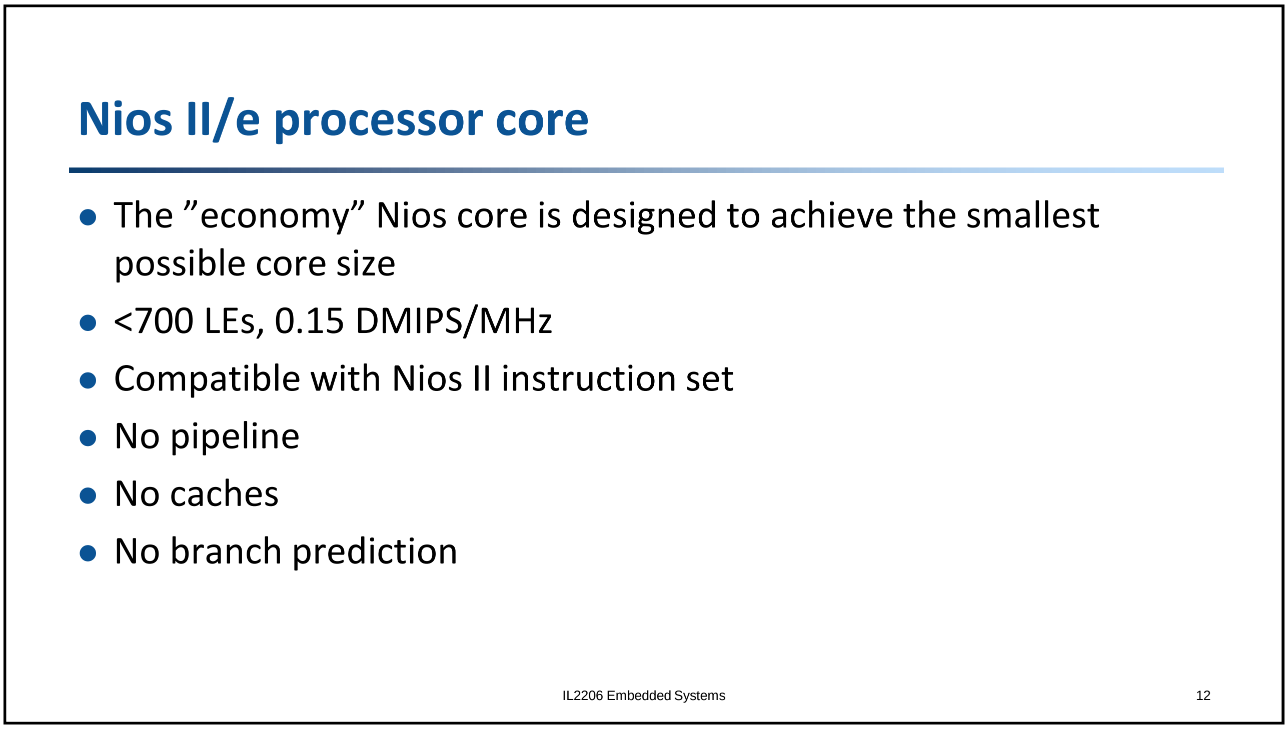
- The “fast” Nios core is designed for high execution performance
- <1800 LEs, max 1.16 DMIPS/MHz (DMIPS = Dhrystone Million Instructions per Second)
- Characteristics:
  - Separate Instruction and Data Caches
  - 2 GByte of external address space
  - Optional tightly-coupled memory
  - 6-stage pipeline
  - Dynamic branch prediction
  - Hardware multiply/divide
  - Possibility to add custom instructions



Nios II/s processor core

- The ”standard” Nios core is designed for a small core size. On-chip logic and memory resources are conserved at the expense of execution performance
- <1400 LEs, 0.74 DMIPS
- Instruction cache, but no data cache
- 5-stage pipeline

11



Nios II/e processor core

- The ”economy” Nios core is designed to achieve the smallest possible core size
- <700 LEs, 0.15 DMIPS/MHz
- Compatible with Nios II instruction set
- No pipeline
- No caches
- No branch prediction

12

Overview Nios II family (Extract)

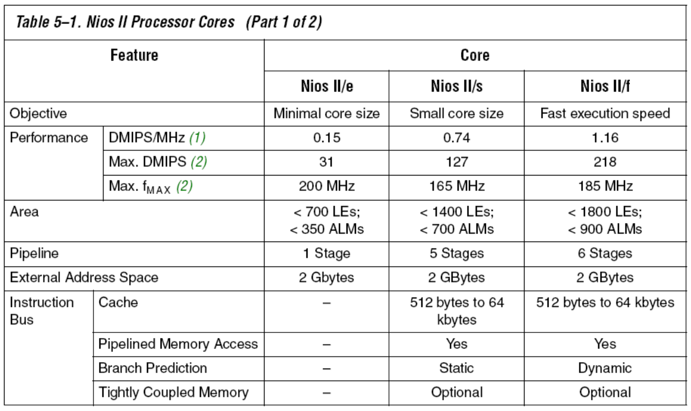


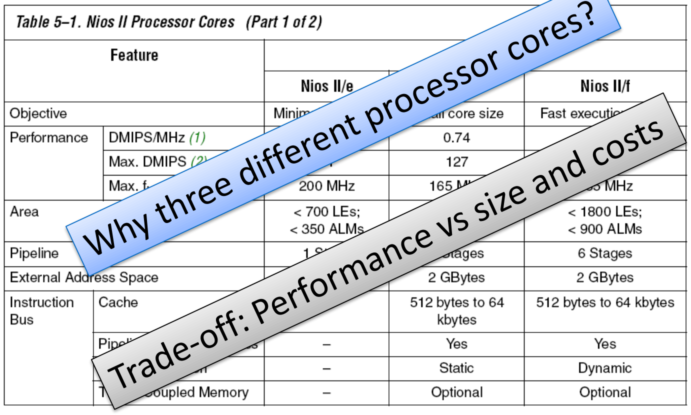
Table 5–1. Nios II Processor Cores (Part 1 of 2)

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz (1)	0.15	0.74	1.16
	Max. DMIPS (2)	31	127	218
	Max. f_{MAX} (2)	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	< 1800 LEs; < 900 ALMs
Pipeline		1 Stage	5 Stages	6 Stages
External Address Space		2 Gbytes	2 GBytes	2 GBytes
Instruction Bus	Cache	–	512 bytes to 64 kbytes	512 bytes to 64 kbytes
	Pipelined Memory Access	–	Yes	Yes
	Branch Prediction	–	Static	Dynamic
	Tightly Coupled Memory	–	Optional	Optional

(2) Numbers are for the fastest device in the Stratix II family

13

Overview Nios II family (Extract)



Why three different processor cores?

Trade-off: Performance vs size and costs

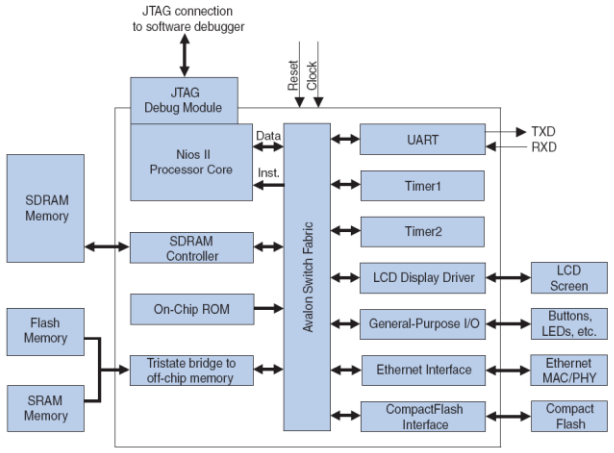
Table 5–1. Nios II Processor Cores (Part 1 of 2)

Feature		[illegible]		
		Nios II/e	[illegible]	Nios II/f
Objective		Minim	ll core size	Fast executio
Performance	DMIPS/MHz (1)	[illegible]	0.74	[illegible]
	Max. DMIPS (2	[illegible]	127	[illegible]
	Max. f	200 MHz	165 M	MHz
Area		< 700 LEs; < 350 ALMs	[illegible]	< 1800 LEs; < 900 ALMs
Pipeline		1 S	tages	6 Stages
External Address Space		[illegible]	2 GBytes	2 GBytes
Instruction Bus	Cache	[illegible]	512 bytes to 64 kbytes	512 bytes to 64 kbytes
	Pip	–	Yes	Yes
	n	–	Static	Dynamic
	T oupled Memory	–	Optional	Optional

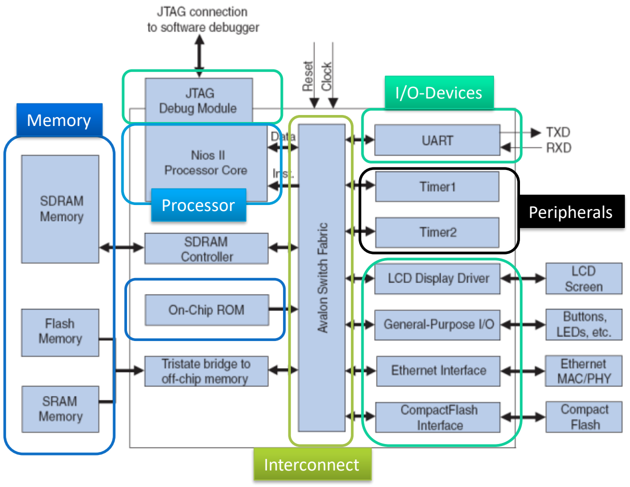
(2) Numbers are for the fastest device in the Stratix II family

14

Example
Nios II platform configuration



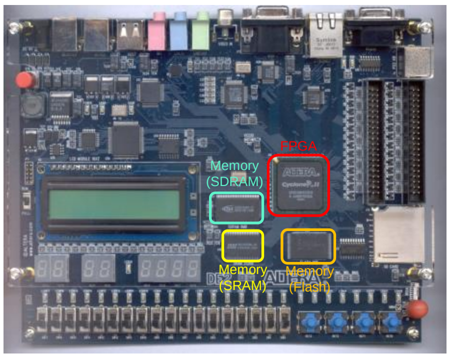
Example
Nios II platform configuration



The Laboratory Environment

- DE2 Board
  - Cyclone II EP2C35 FPGA
  - 4 Mbytes of flash memory
  - 512 Kbytes of static RAM
  - 8 Mbytes of SDRAM
  - Several I/O-Devices
  - 50 MHz oscillator

NOTE: Cyclone II family has been released 2004, now FPGAs are much more powerful, and often include hard processor cores!

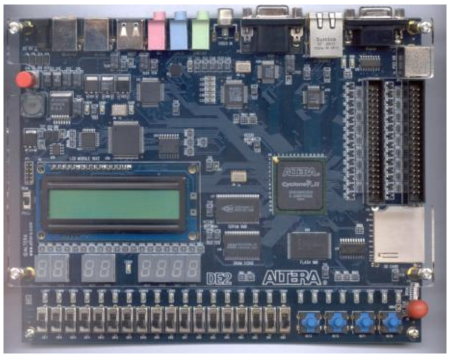


We might also use the DE2-115 board in the laboratory course (uses Cyclone IV FPGA)

17

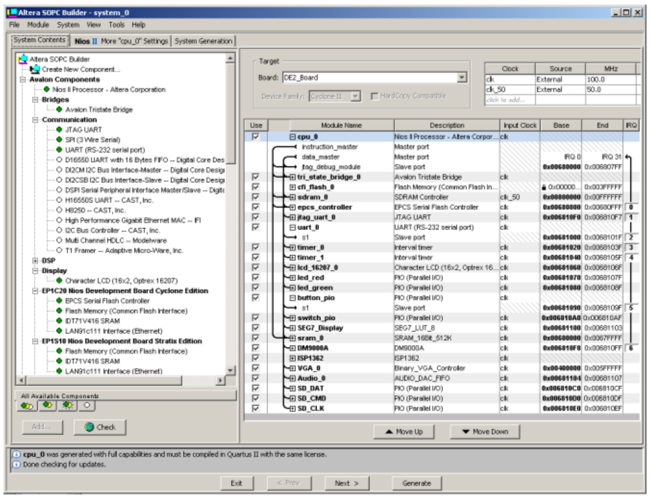
Design Flow – Nios II System-on-Chip

1. Design a Nios II platform with peripherals using the platform design tool QSYS (previously SOPC builder)
2. Write your software application using the symbolic addresses and values that have been defined in QSYS
3. Download the Nios platform to the FPGA
4. Download the software program to the memory



18

Customizing a Nios II Core
The Tool SOPC Builder


Altera SOPC Builder - system_0

- Altera SOPC Builder
  - Create New Component...
- **Avalon Components**
  - Nios II Processor - Altera Corporation
  - **Bridges**
    - Avalon Tristate Bridge
  - **Communication**
    - JTAG UART
    - SPI (3 Wire Serial)
    - UART (RS-232 serial port)
    - D16550 UART with 16 Bytes FIFO -- Digital Core Des
    - DI2CM I2C Bus Interface-Master -- Digital Core Desig
    - DI2CSB I2C Bus Interface-Slave -- Digital Core Desig
    - DSPI Serial Peripheral Interface Master/Slave -- Digit
    - H16550S UART -- CAST, Inc.
    - H8250 -- CAST, Inc.
    - High Performance Gigabit Ethernet MAC -- IFI
    - I2C Bus Controller -- CAST, Inc.
    - Multi Channel HDLC -- Modelware
    - T1 Framer -- Adaptive Micro-Ware, Inc.
  - **DSP**
  - **Display**
    - Character LCD (16x2, Optrex 16207)
  - **EP1C20 Nios Development Board Cyclone Edition**
    - EPCS Serial Flash Controller
    - Flash Memory (Common Flash Interface)
    - IDT71V416 SRAM
    - LAN91c111 Interface (Ethernet)
  - **EP1S10 Nios Development Board Stratix Edition**
    - Flash Memory (Common Flash Interface)
    - IDT71V416 SRAM
    - LAN91c111 Interface (Ethernet)

Target
Board: DE2_Board
Device Family: Cyclone II
HardCopy Compatible

Clock	Source	MHz
clk	External	100.0
clk_50	External	50.0
click to add...		

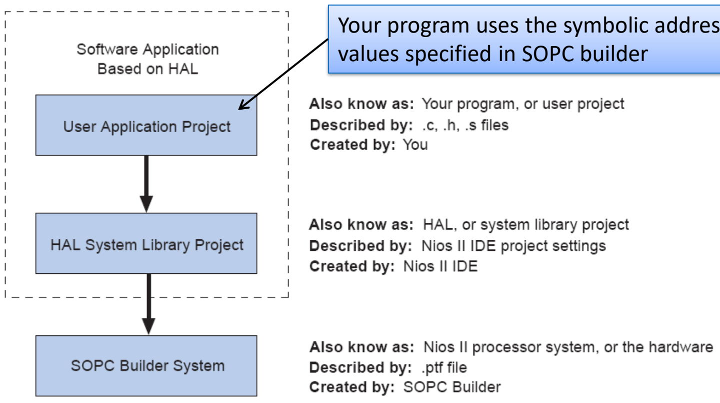
Use	Module Name	Description	Input Clock	Base	End	IRQ
✓	cpu_0	Nios II Processor - Altera Corpor...	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00680000	0x006807FF	
✓	tri_state_bridge_0	Avalon Tristate Bridge	clk			
✓	cfi_flash_0	Flash Memory (Common Flash In...		0x00000...	0x003FFFFF	
✓	sdram_0	SDRAM Controller	clk_50	0x00800000	0x00FFFFFF	
✓	epcs_controller	EPCS Serial Flash Controller	clk	0x00680800	0x00680FFF	0
✓	jtag_uart_0	JTAG UART	clk	0x006810F0	0x006810F7	1
✓	uart_0	UART (RS-232 serial port)	clk			
	s1	Slave port		0x00681000	0x0068101F	2
✓	timer_0	Interval timer	clk	0x00681020	0x0068103F	3
✓	timer_1	Interval timer	clk	0x00681040	0x0068105F	4
✓	lcd_16207_0	Character LCD (16x2, Optrex 16...	clk	0x00681060	0x0068106F	
✓	led_red	PIO (Parallel I/O)	clk	0x00681070	0x0068107F	
✓	led_green	PIO (Parallel I/O)	clk	0x00681080	0x0068108F	
✓	button_pio	PIO (Parallel I/O)	clk			
	s1	Slave port		0x00681090	0x0068109F	5
✓	switch_pio	PIO (Parallel I/O)	clk	0x006810A0	0x006810AF	
✓	SEG7_Display	SEG7_LUT_8	clk	0x00681100	0x00681103	
✓	sram_0	SRAM_16Bit_512K	clk	0x00600000	0x0067FFFF	
✓	DM9000A	DM9000A	clk	0x006810F8	0x006810FF	6
✓	ISP1362	ISP1362	clk			
✓	VGA_0	Binary_VGA_Controller	clk	0x00400000	0x005FFFFF	
✓	Audio_0	AUDIO_DAC_FIFO	clk	0x00681104	0x00681107	
✓	SD_DAT	PIO (Parallel I/O)	clk	0x006810C0	0x006810CF	
✓	SD_CMD	PIO (Parallel I/O)	clk	0x006810D0	0x006810DF	
✓	SD_CLK	PIO (Parallel I/O)	clk	0x006810E0	0x006810EF	

cpu_0 was generated with full capabilities and must be compiled in Quartus II with the same license.
Done checking for updates.

- Designer can select and configure
  - CPU(s)
  - peripherals
- SOPC builder creates core that can be instantiated on FPGA
- Symbolic names are accessible for software designers

The 'QSYS' tool is used in recent versions, but the principle is the same as in SOPC Builder!

The role of the HAL in a Nios II project



Your program uses the symbolic addresses and values specified in SOPC builder

Also know as: Your program, or user project
Described by: .c, .h, .s files
Created by: You

Also know as: HAL, or system library project
Described by: Nios II IDE project settings
Created by: Nios II IDE

Also know as: Nios II processor system, or the hardware
Described by: .ptf file
Created by: SOPC Builder

Reference: Nios II Software Developers Handbook

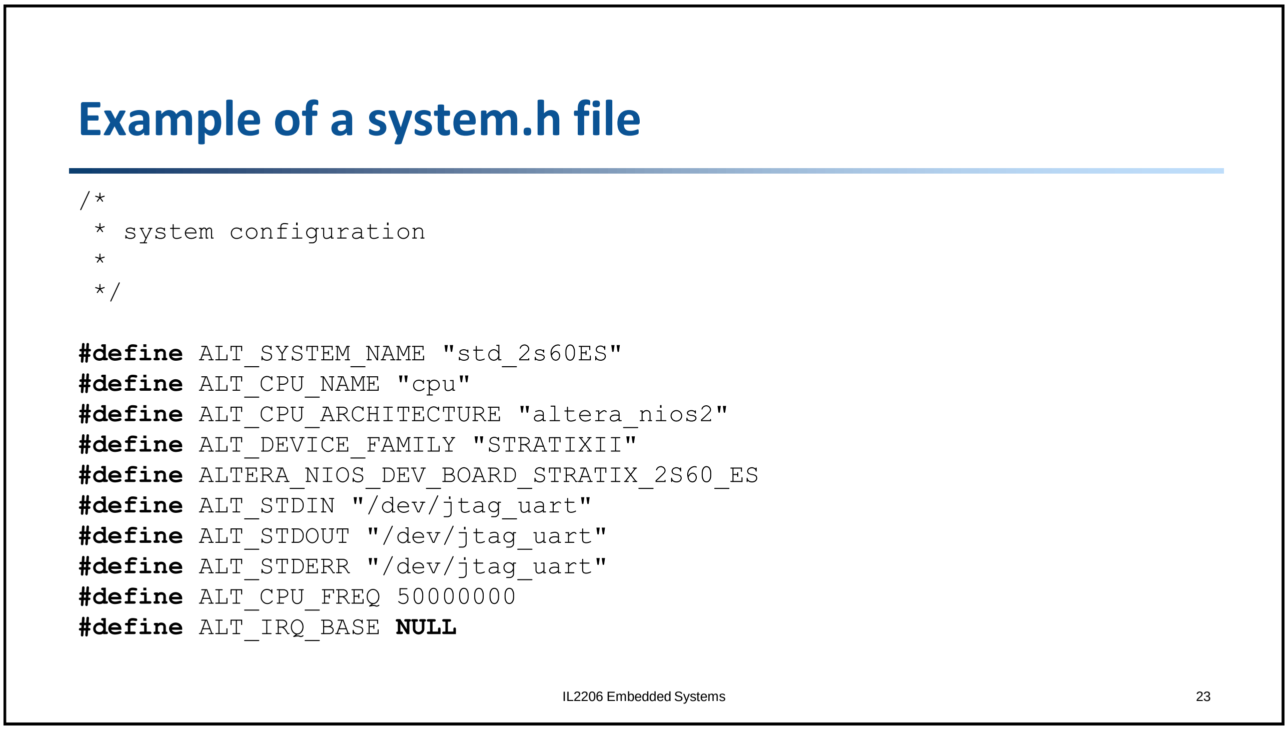
The system.h file

- The system.h file provides a complete software description of the Nios II system hardware
- The system.h file reflects the actual Nios II hardware, which is given by the *.ptf file.
- A new core means a new *.ptf file and a new system.h file

The system.h file

- The system.h file describes each peripheral and provides the following details:
  - The hardware configuration of the peripheral
  - The base address
  - The IRQ (interrupt request) priority
  - A symbolic name for the peripheral
- If the hardware changes, the source code is still valid, it will only use another system.h file

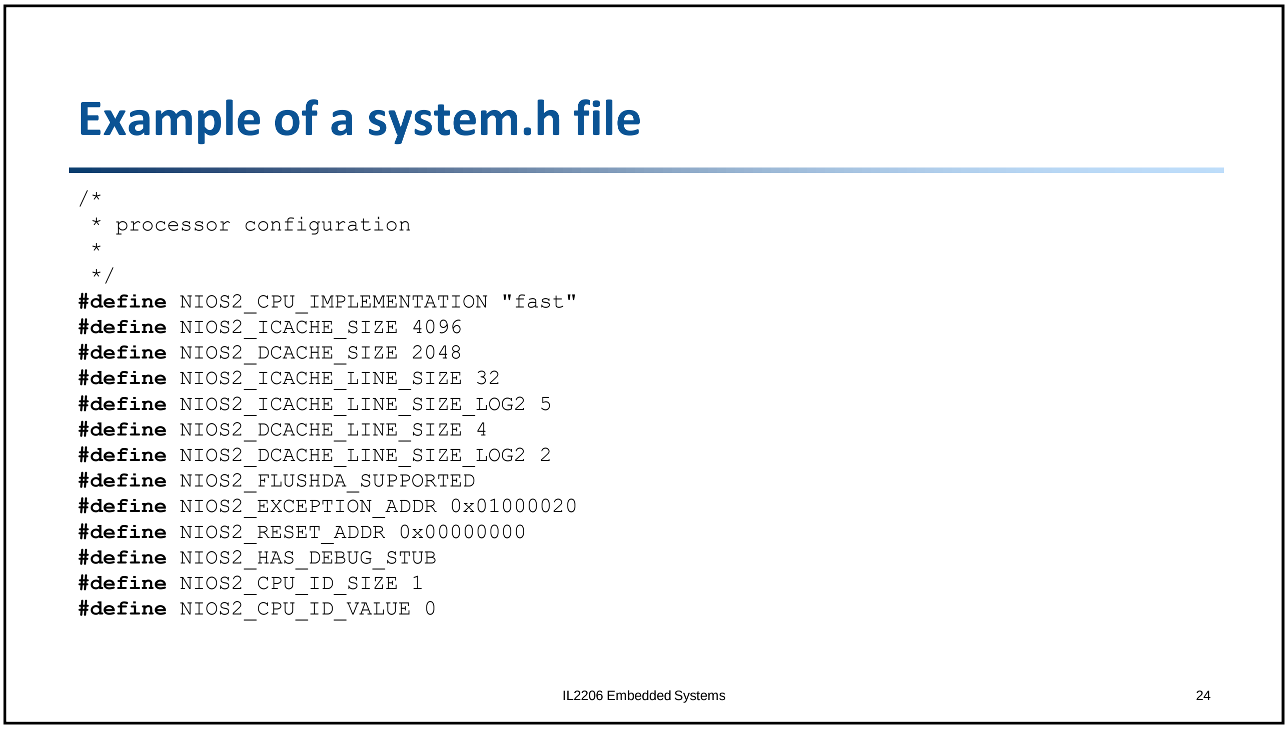
NEVER edit the system.h file!!!



Example of a system.h file

```
/*
 * system configuration
 *
 */

#define ALT_SYSTEM_NAME "std_2s60ES"
#define ALT_CPU_NAME "cpu"
#define ALT_CPU_ARCHITECTURE "altera_nios2"
#define ALT_DEVICE_FAMILY "STRATIXII"
#define ALTERA_NIOS_DEV_BOARD_STRATIX_2S60_ES
#define ALT_STDIN "/dev/jtag_uart"
#define ALT_STDOUT "/dev/jtag_uart"
#define ALT_STDERR "/dev/jtag_uart"
#define ALT_CPU_FREQ 50000000
#define ALT_IRQ_BASE NULL
```

23



Example of a system.h file

```
/*
 * processor configuration
 *
 */
#define NIOS2_CPU_IMPLEMENTATION "fast"
#define NIOS2_ICACHE_SIZE 4096
#define NIOS2_DCACHE_SIZE 2048
#define NIOS2_ICACHE_LINE_SIZE 32
#define NIOS2_ICACHE_LINE_SIZE_LOG2 5
#define NIOS2_DCACHE_LINE_SIZE 4
#define NIOS2_DCACHE_LINE_SIZE_LOG2 2
#define NIOS2_FLUSHDA_SUPPORTED
#define NIOS2_EXCEPTION_ADDR 0x01000020
#define NIOS2_RESET_ADDR 0x00000000
#define NIOS2_HAS_DEBUG_STUB
#define NIOS2_CPU_ID_SIZE 1
#define NIOS2_CPU_ID_VALUE 0
```

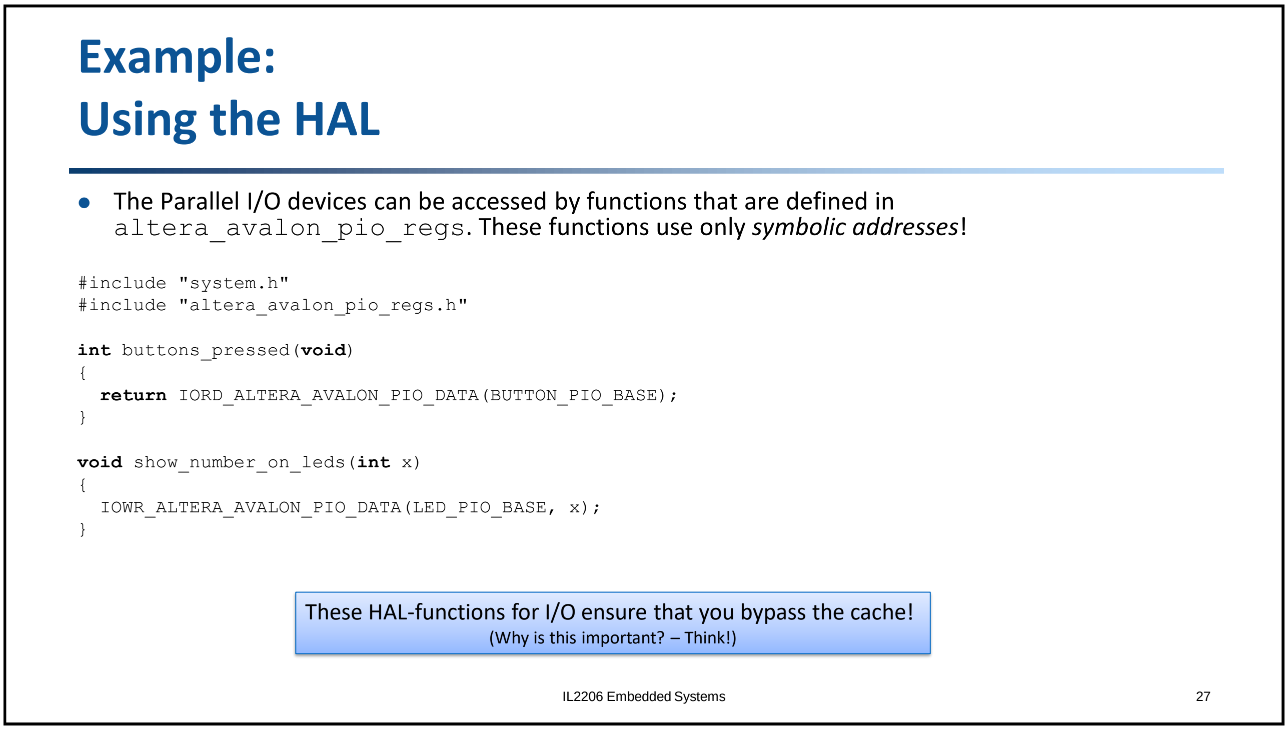
24

Example of a system.h file

```
/*
 * uart1 configuration
 *
 */

#define UART1_NAME "/dev/uart1"
#define UART1_TYPE "altera_avalon_uart"
#define UART1_BASE 0x02120840
#define UART1_IRQ 4
#define UART1_BAUD 115200
#define UART1_DATA_BITS 8
#define UART1_FIXED_BAUD 1
#define UART1_PARITY 'N'
#define UART1_STOP_BITS 1
#define UART1_USE_CTS_RTS 0
#define UART1_USE_EOP_REGISTER 0
#define UART1_SIM_TRUE_BAUD 0
#define UART1_SIM_CHAR_STREAM ""
#define UART1_FREQ 50000000
```

Example of a system.h file

```
/*
 * button_pio configuration
 *
 */

#define BUTTON_PIO_NAME "/dev/button_pio"
#define BUTTON_PIO_TYPE "altera_avalon_pio"
#define BUTTON_PIO_BASE 0x02120860
#define BUTTON_PIO_IRQ 2
#define BUTTON_PIO_HAS_TRI 0
#define BUTTON_PIO_HAS_OUT 0
#define BUTTON_PIO_HAS_IN 1
#define BUTTON_PIO_CAPTURE 1
#define BUTTON_PIO_EDGE_TYPE "ANY"
#define BUTTON_PIO_IRQ_TYPE "EDGE"
#define BUTTON_PIO_DO_TEST_BENCH_WIRING 1
#define BUTTON_PIO_DRIVEN_SIM_VALUE 0x000F
#define BUTTON_PIO_FREQ 50000000
```



Example: Using the HAL

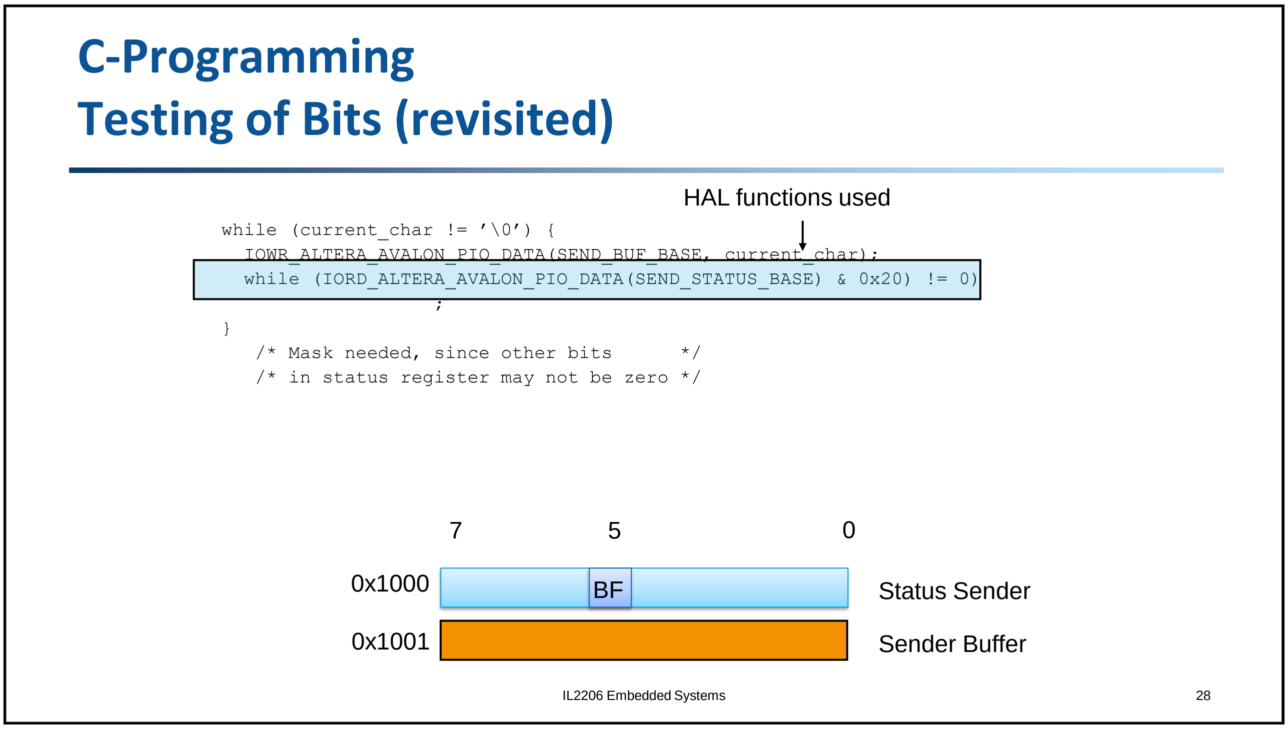
- The Parallel I/O devices can be accessed by functions that are defined in `altera_avalon_pio_regs`. These functions use only *symbolic addresses*!

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

int buttons_pressed(void)
{
  return IORD_ALTERA_AVALON_PIO_DATA(BUTTON_PIO_BASE);
}

void show_number_on_leds(int x)
{
  IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, x);
}
```

These HAL-functions for I/O ensure that you bypass the cache!
(Why is this important? – Think!)

IL2206 Embedded Systems 27

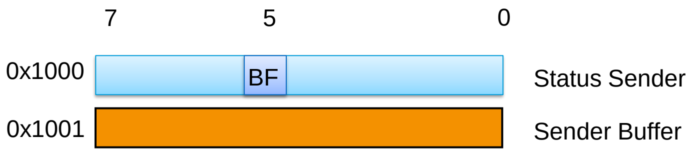
27



C-Programming Testing of Bits (revisited)

HAL functions used

```
while (current_char != '\0') {
  IOWR_ALTERA_AVALON_PIO_DATA(SEND_BUF_BASE, current_char);
  while (IORD_ALTERA_AVALON_PIO_DATA(SEND_STATUS_BASE) & 0x20) != 0)
           ;
}
   /* Mask needed, since other bits      */
   /* in status register may not be zero */
```



IL2206 Embedded Systems 28

28

Registering an ISR with `alt_irq_register()`

- The HAL registers this function pointer in a lookup table. When a specific IRQ occurs, the HAL looks up the IRQ in the lookup table and dispatches the registered ISR.
- The prototype for `alt_irq_register()` is:

```
int alt_irq_register (alt_u32 id,
                      void* context,
                      void (*isr)(void*, alt_u32));
```

Read the Altera documentation: Nios II Software Developers Handbook

Registering an ISR with `alt_irq_register()`

- The prototype has the following parameters:
  - `id` is the hardware interrupt number for the device, as defined in **`system.h`**. Interrupt priority corresponds inversely to the IRQ number. Therefore, IRQ 0 represents the highest priority interrupt and IRQ 31 is the lowest.
  - `context` is a pointer used to pass context-specific information to the ISR, and can point to any sort of ISR-specific information. The context value is opaque to the HAL; it is provided entirely for the benefit of the user-defined ISR
  - `isr` is the function that is called in response to IRQ number id. The two input arguments provided to this function are the context pointer and `id`. Registering a null pointer for isr results in the interrupt being disabled
  - If your ISR is successfully registered, the associated interrupt (as defined by `id`) is enabled on return from `alt_irq_register()`

An ISR to service a Button PIO IRQ

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"

static void handle_button_interrupts(void* context, alt_u32 id)
{
  /* cast the context pointer to an integer pointer. */
  volatile int* edge_capture_ptr = (volatile int*) context;
  /*
   * Read the edge capture register on the button PIO.
   * Store value.
   */
   *edge_capture_ptr =
   IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
   /* Write to the edge capture register to reset it. */
   IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE, 0);
   /* reset interrupt capability for the Button PIO. */
   IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE, 0xf);
}
```

31

Registering the Button PIO ISR with the HAL

```
#include "sys/alt_irq.h"
#include "system.h"
...
/* Declare a global variable to hold the
   edge capture value. */
volatile int edge_capture;
...
/* Register the interrupt handler. */
alt_irq_register(BUTTON_PIO_IRQ,
                 (void*) &edge_capture,
                 handle_button_interrupts);
```

32

What happens?

- Based on this code, the following execution flow is possible:
  1. Button is pressed, generating an IRQ.
  2. The HAL exception handler is invoked and dispatches the `handle_button_interrupts()` ISR.
  3. `handle_button_interrupts()` services the interrupt and returns.
  4. Normal program operation continues with an updated value of `edge_capture`.

Quick Question

- Which of the following architectures cannot be realized on the FPGA of the DE2 board (EP2C35)?

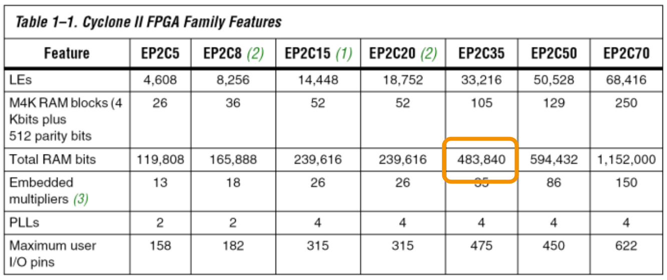


Table 1–1. Cyclone II FPGA Family Features

Feature	EP2C5	EP2C8 (2)	EP2C15 (1)	EP2C20 (2)	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers (3)	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

- 1) 4 Nios II/f processors (1800LEs) with 4 kB I-Cache and 4 kB D-cache
- 2) 2 Nios II/f processors (1800LEs) with 16 kB I-Cache and 8 kB D-cache
- 3) 1 Nios II/f processor (1800LEs) with 32 kB I-Cache and 32 kB D-cache

Stratix III Family (announced 2009)

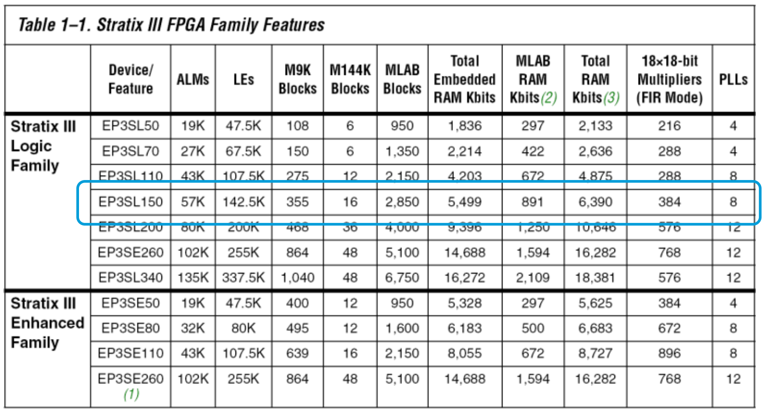


Table 1–1. Stratix III FPGA Family Features

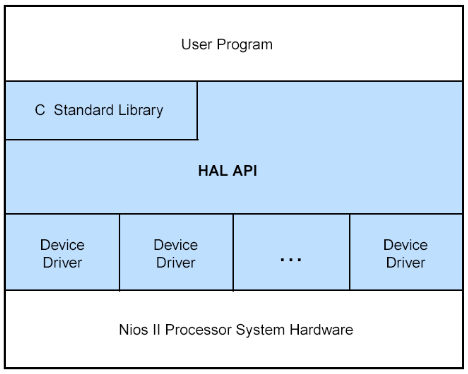
	Device/ Feature	ALMs	LEs	M9K Blocks	M144K Blocks	MLAB Blocks	Total Embedded RAM Kbits	MLAB RAM Kbits (2)	Total RAM Kbits (3)	18×18-bit Multipliers (FIR Mode)	PLLs
Stratix III Logic Family	EP3SL50	19K	47.5K	108	6	950	1,836	297	2,133	216	4
	EP3SL70	27K	67.5K	150	6	1,350	2,214	422	2,636	288	4
	EP3SL110	43K	107.5K	275	12	2,150	4,203	672	4,875	288	8
	EP3SL150	57K	142.5K	355	16	2,850	5,499	891	6,390	384	8
	EP3SL200	80K	200K	468	36	4,000	9,396	1,250	10,646	576	12
	EP3SE260	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12
	EP3SL340	135K	337.5K	1,040	48	6,750	16,272	2,109	18,381	576	12
Stratix III Enhanced Family	EP3SE50	19K	47.5K	400	12	950	5,328	297	5,625	384	4
	EP3SE80	32K	80K	495	12	1,600	6,183	500	6,683	672	8
	EP3SE110	43K	107.5K	639	16	2,150	8,055	672	8,727	896	8
	EP3SE260 (1)	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12

DE3 Board

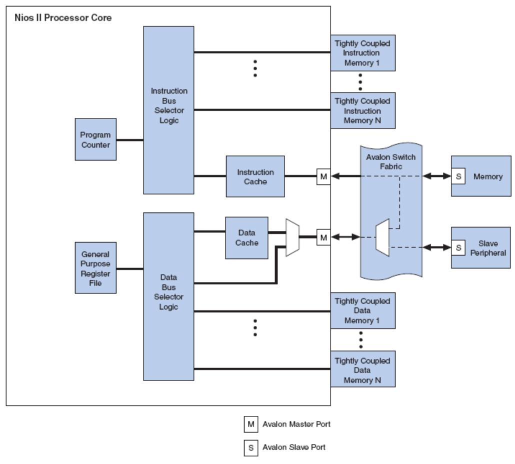
Todays FPGAs are much larger !!! Check!

Programming the Nios II

- Altera assumes that the Nios II is programmed in C/C++, using the Hardware Abstraction Layer (HAL)
- Reasons are
  - Maintenance
  - Time-to-Market
  - Flexibility



Nios II Memory & I/O Organization

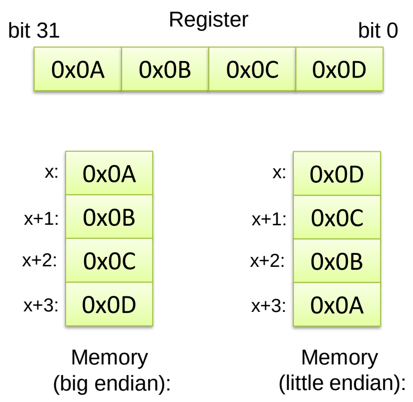

Nios II Processor Core
Program Counter
Instruction Bus Selector Logic
Instruction Cache
General Purpose Register File
Data Bus Selector Logic
Data Cache
Tightly Coupled Instruction Memory 1
Tightly Coupled Instruction Memory N
Avalon Switch Fabric
Memory
Slave Peripheral
Tightly Coupled Data Memory 1
Tightly Coupled Data Memory N
M
S
M Avalon Master Port
S Avalon Slave Port

- Harvard Architecture (Separated Instruction & Data Memory)
- Each peripheral circuits is assigned an individual address range (memory-mapped I/O)

38

Addresses and Endianness

- The Nios II uses 32-bit addresses
- A Word is 32 bits (4 bytes) long
- An Address refers to a byte (not a word)
- The Nios II processor uses the little-endian memory system
  - Little-endian: lowest-order byte resides in the low-order bits of a word
  - Big-endian: lower-order byte resides in highest bits of the word



39

Nios II Programming Model Instruction Set

Custom Instructions

- Nios reserves instruction codes for custom instruction that designers create by defining the hardware for these functions
- Critical functions can be executed much faster

Execution Performance Nios II/f

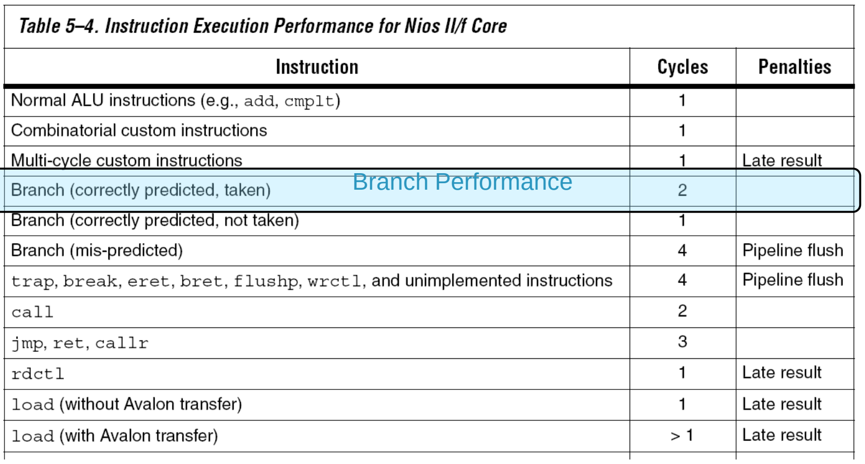


Table 5–4. Instruction Execution Performance for Nios II/f Core

Instruction	Cycles	Penalties
Normal ALU instructions (e.g., `add`, `cmplt`)	1	
Combinatorial custom instructions	1	
Multi-cycle custom instructions	1	Late result
Branch (correctly predicted, taken)	2	
Branch (correctly predicted, not taken)	1	
Branch (mis-predicted)	4	Pipeline flush
`trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, and unimplemented instructions	4	Pipeline flush
`call`	2	
`jmp`, `ret`, `callr`	3	
`rdctl`	1	Late result
`load` (without Avalon transfer)	1	Late result
`load` (with Avalon transfer)	> 1	Late result

Branch Performance

Multiplication/Division heavily depends on underlying hardware!

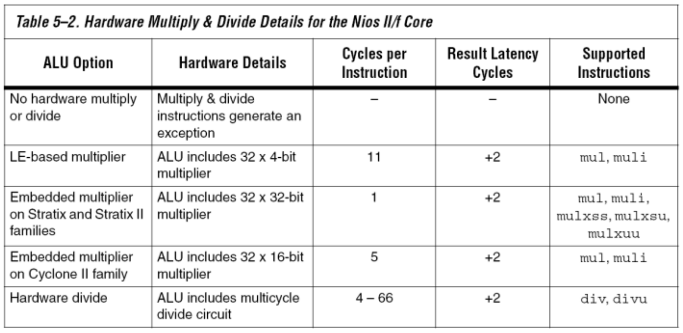


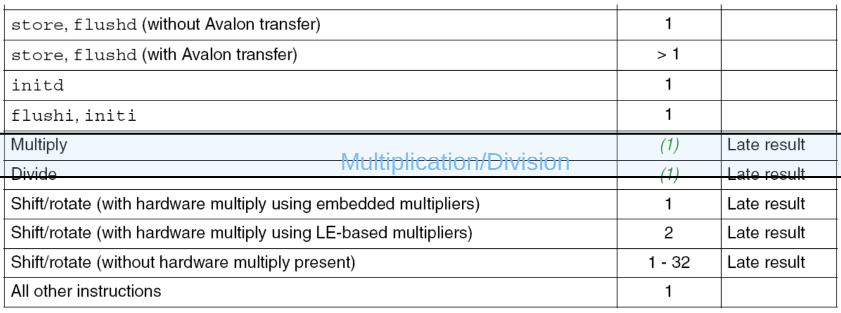
Table 5–2. Hardware Multiply & Divide Details for the Nios II/f Core

ALU Option	Hardware Details	Cycles per Instruction	Result Latency Cycles	Supported Instructions
No hardware multiply or divide	Multiply & divide instructions generate an exception	–	–	None
LE-based multiplier	ALU includes 32 x 4-bit multiplier	11	+2	`mul`, `muli`
Embedded multiplier on Stratix and Stratix II families	ALU includes 32 x 32-bit multiplier	1	+2	`mul`, `muli`, `mulxss`, `mulxsu`, `mulxuu`
Embedded multiplier on Cyclone II family	ALU includes 32 x 16-bit multiplier	5	+2	`mul`, `muli`
Hardware divide	ALU includes multicycle divide circuit	4 – 66	+2	`div`, `divu`

IL2206 Embedded Systems 42

42

Execution Performance Nios II/f



`store`, `flushd` (without Avalon transfer)	1	
`store`, `flushd` (with Avalon transfer)	> 1	
`initd`	1	
`flushi`, `initi`	1	
Multiply	*(1)*	Late result
Divide	*(1)*	Late result
Shift/rotate (with hardware multiply using embedded multipliers)	1	Late result
Shift/rotate (with hardware multiply using LE-based multipliers)	2	Late result
Shift/rotate (without hardware multiply present)	1 - 32	Late result
All other instructions	1	

Multiplication/Division

IL2206 Embedded Systems 43

43

Latency Cycles

- Latency Cycles can decrease performance, if data dependency exist
- Execution of the addi instruction is delayed by two additional cycles until the multiply operation completes

```
mul r1, r2, r3    #
addi r1, r1, 100 # (Depends on result of mul)
```

- In contrast, the code below does not stall the processor

```
mul r1, r2, r3
or r5, r5, r6     # No dependency on previous results
or r7, r7, r8     # No dependency on previous results
addi r1, r1, 100 # (Depends on result of mul)
```

Embedded Microprocessor Families

- ARM is a family of RISC architectures
  - Soft-core, market leader, proprietary instruction set
  - Cortex-M (microcontroller), Cortex-A (application), Cortex-R (real-time and safety-critical)
- RISC-V is an open RISC instruction set architecture
  - No owner of instruction set, tools (compilers) can be shared, hardware can be developed by different vendors, no licensing fees
- Xilinx Microblaze is the embedded processor supported by Xilinx for their FPGAs (similar to Nios)
- Many more specialized processors exist, among them
  - Infineon Aurix (safety-critical systems, automotive)
  - Kalray MPPA (massively parallel processor array) (Bostan: 288 cores)