Processor Peripherals
Third Laboratory Session

Prof. Claudio Passerone

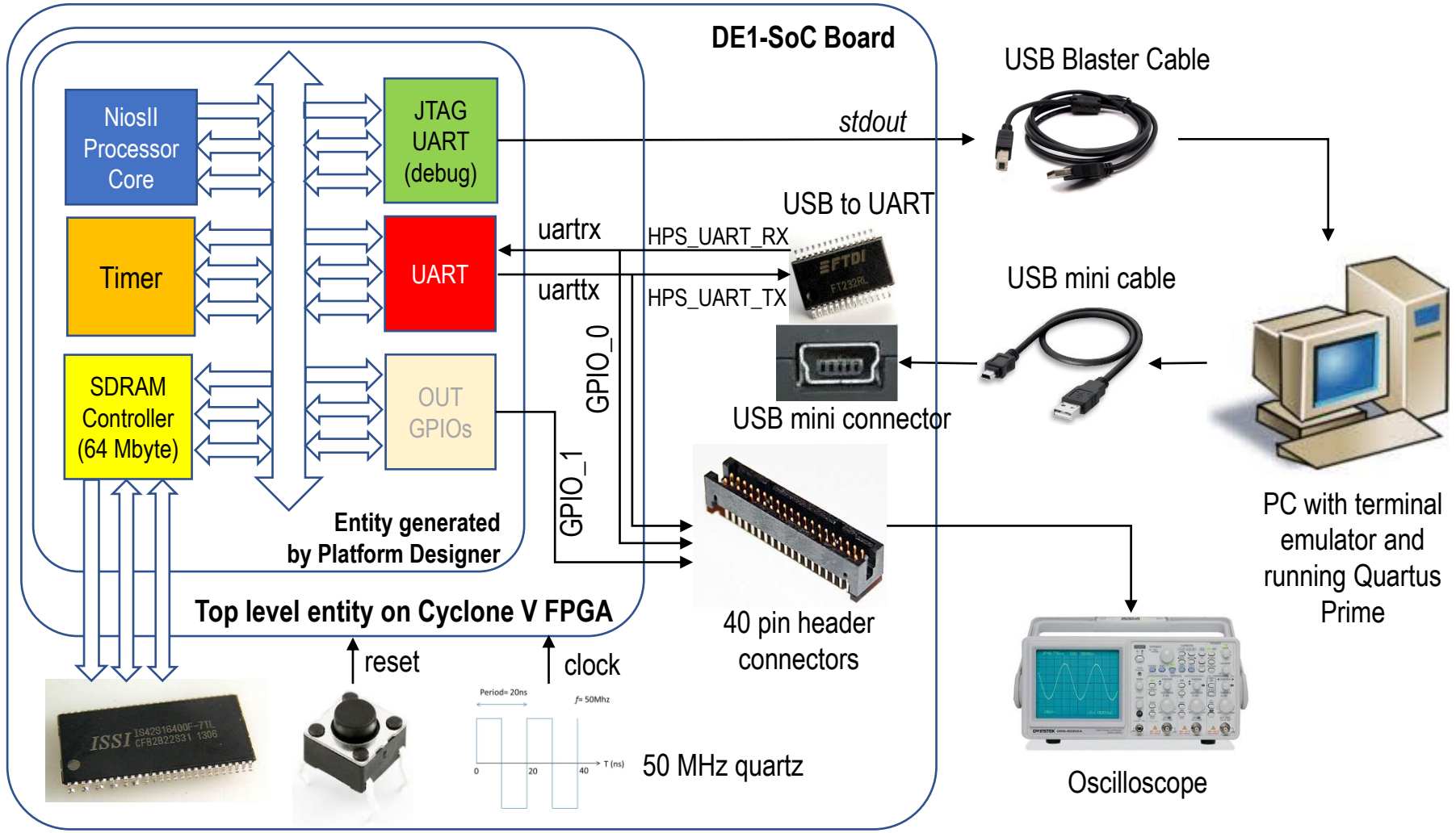
(Electronics for Embedded Systems)

Introduction

- Manage UART transmissions using a hardware peripheral
  - Use the UART peripheral of the Nios II processor
  - VHDL code provided by the instructor
    - The same as the second laboratory session
  - Assignment: write software to
    - Configure the UART peripheral
    - Send characters over UART to a terminal emulator
    - Receive characters over UART from a terminal emulator
  - Hardware implementation gives several advantages
    - Higher performance
      - Easily reach 115200 baud rate
    - Simplified software on the processor
      - Three lines of code to read a character
    - More time available for the processor core to perform other tasks

Hardware Project

- SoC components
  - Nios II processor core
  - JTAG_UART (to manage stdout)
  - UART peripheral
  - Output GPIOs (to drive the header connector)
  - SDRAM Controller
  - Timer (not used to measure UART transfers)

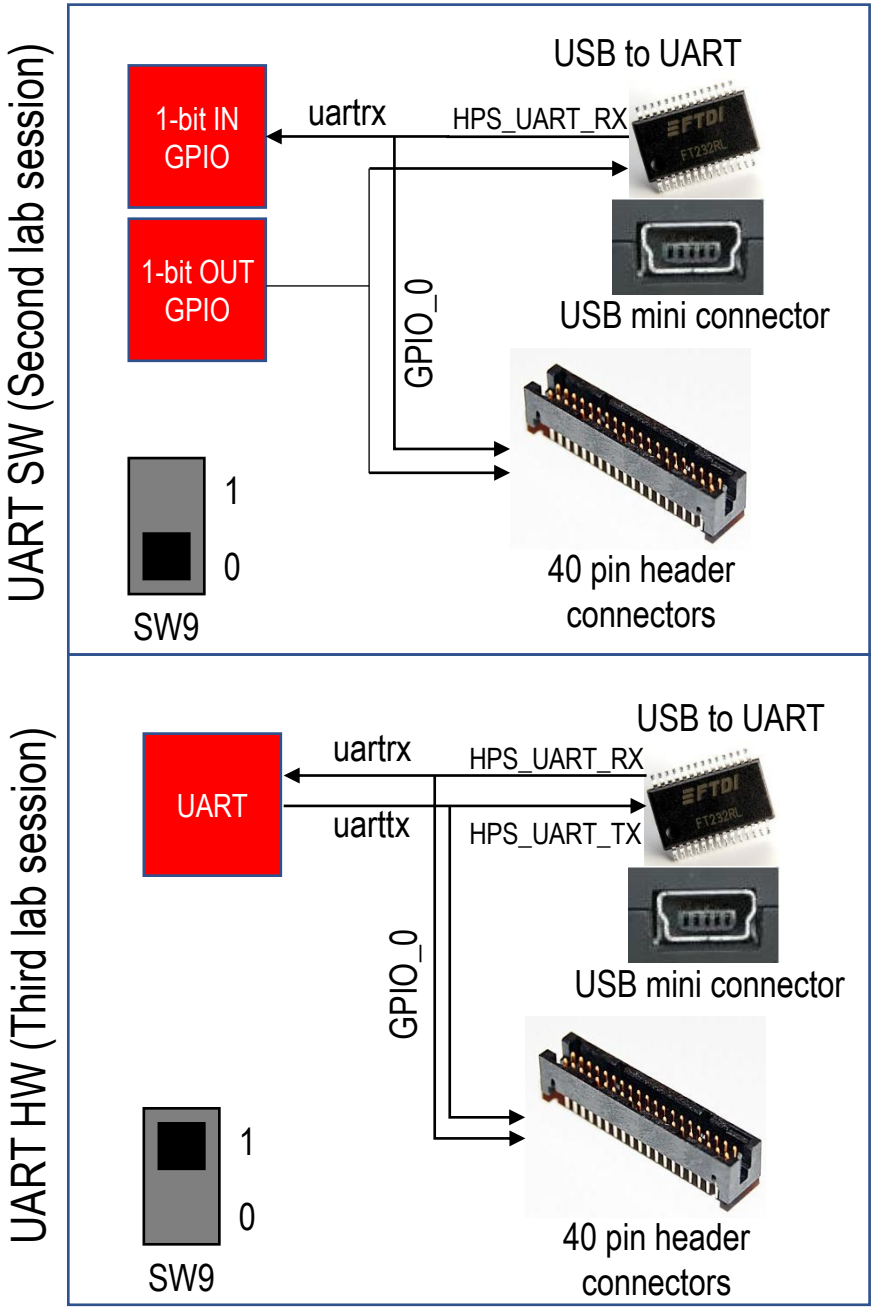


Hardware Project

- The full HW project also includes
  - PLL to generate clocks
    - For the Nios II processor
    - For the external 64 Mbyte SDRAM
  - ARM9 processor
    - With a microSD Card to bootstrap it…
    - … and forward UART lines to the FPGA fabric (and the UART peripheral of the Nios II processor)
    - Same microSD Card as in the second laboratory session
  - Several GPIOs to manage
    - LEDs, switches and buttons
    - Seven segment displays
  - Other peripherals
    - I2C, SPI
      - Don’t use, reserved for future students

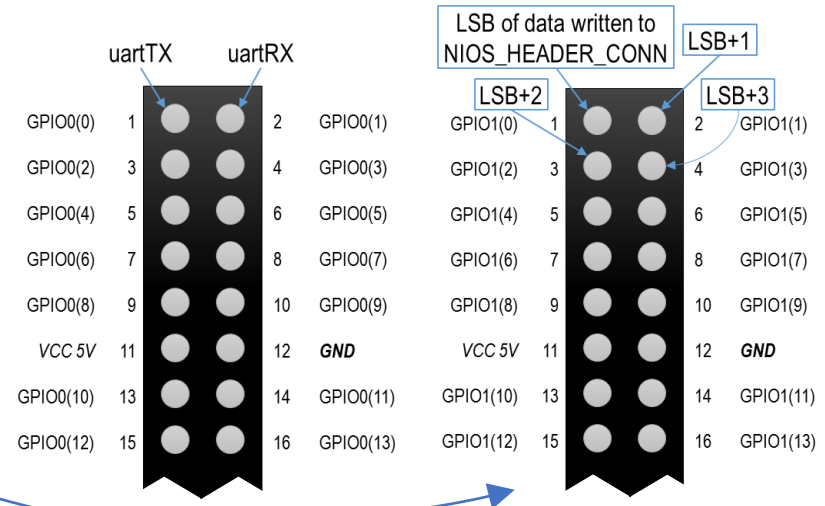
Hardware Project

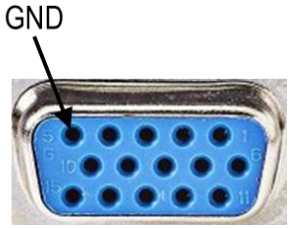
- HW Project implementation
  - Download the zip file
    - Identical to the second laboratory session project
  - Uncompress it in some folder
    - Where you have write permission
    - Without spaces, parentheses or special characters in the full path
  - Insert the microSD Card in the Card Reader of the DE1-Soc Board
  - Open the HW project with Quartus Prime
    - You don't have to recompile it
    - Program the board using the programmer
  - <u>Put SW9 (the left most switch) on the 1 position</u>
    - Towards the red LED9
    - In the 0 position it is like the second laboratory session



Hardware Project

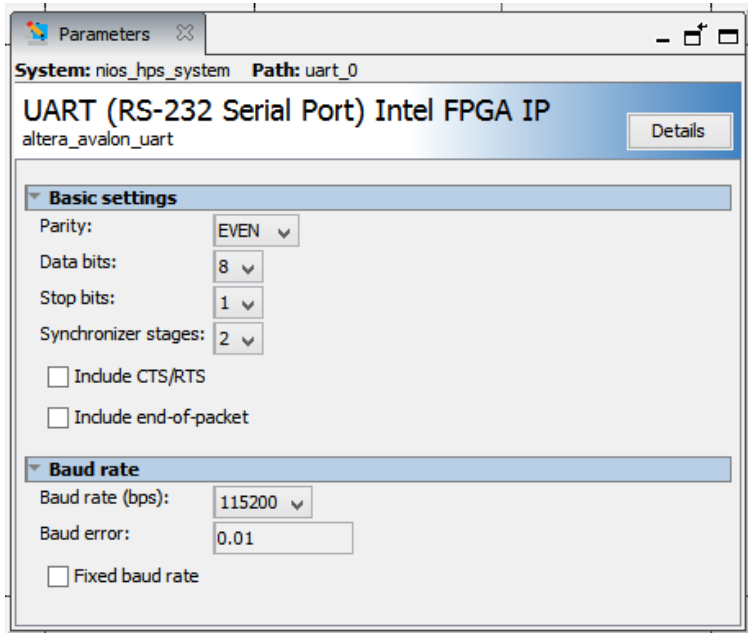
- Connections on the DE1-SoC
  - USB type A Host-to-Device cable for JTAG
    - Programming the hardware
    - Downloading the software
    - Implement stdin and stdout over JTAG
  - USB mini cable for UART connection
    - Using the FDTI232R integrated circuit
  - Female/Male jumpers for the oscilloscope
    - To display UART RX and TX signals
      - On the left header connector
    - To display software controlled pins
      - On the right header connector
  - Power Supply





Altera Nios II UART Peripheral

- Instantiated in Platform Designer
  - Simple UART peripheral
    - No support for FIFOs
  - 115200, 8E1
  - Baud Rate configurable at run-time
  - Parity, Data bits and Stop bits fixed at generation time
  - Base address and range
    - 0x08001060 – 0x0800107f
    - Range is 32 bytes



Memory map

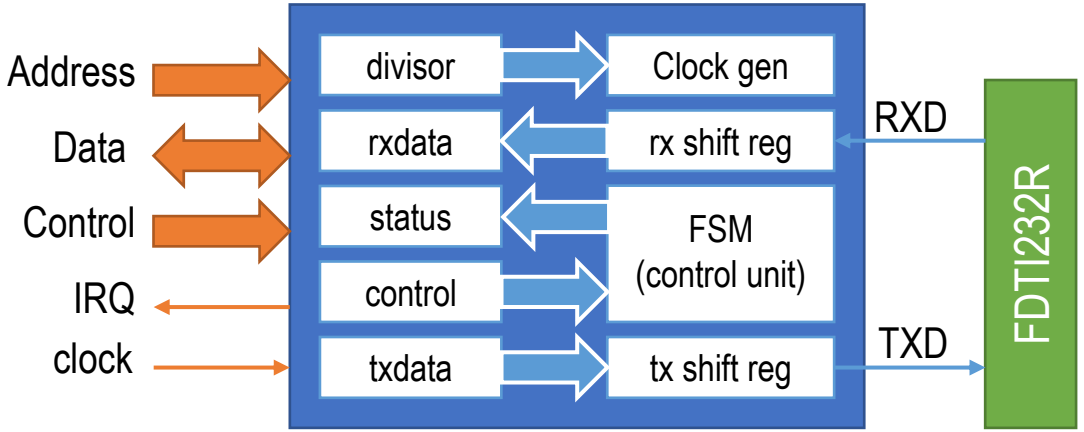
System: nios_hps_system **Path:** nios2_qsys_0

	hps_0.h2f_lw_axi_master	nios2_qsys_0.data_master	nios2_qsys_0.instruction_master
i2c_0.csr		0x0800_1000 – 0x0800_103f	
jtag_uart_0.avalon_jtag_slave		0x0800_1148 – 0x0800_114f	
nios2_qsys_0.jtag_debug_module		0x0800_0800 – 0x0800_0fff	0x0800_0800 – 0x0800_0fff
nios_7seg.s1		0x0800_10b0 – 0x0800_10bf	
nios_buttons.s1		0x0800_1110 – 0x0800_111f	
nios_header_conn.s1		0x0800_10a0 – 0x0800_10af	
nios_i2cclk.s1		0x0800_10f0 – 0x0800_10ff	
nios_i2cdat.s1		0x0800_10d0 – 0x0800_10df	
nios_i2crw.s1		0x0800_10c0 – 0x0800_10cf	
nios_leds.s1		0x0800_1130 – 0x0800_113f	
nios_switches.s1		0x0800_1120 – 0x0800_112f	
nios_uartrx.s1		0x0800_1100 – 0x0800_110f	
nios_uarttx.s1		0x0800_10e0 – 0x0800_10ef	
sdram_controller_0.s1		0x0400_0000 – 0x07ff_ffff	0x0400_0000 – 0x07ff_ffff
spi_0.spi_control_port		0x0800_1040 – 0x0800_105f	
sysid_qsys_0.control_slave		0x0800_1140 – 0x0800_1147	
timer_0.s1		0x0800_1080 – 0x0800_109f	
uart_0.s1		0x0800_1060 – 0x0800_107f	

64 MByte

Altera Nios II UART Peripheral

- Peripheral implementation
  - Clock generation unit
    - Starting from the SoC clock signal
  - Receive and transmit shift registers
    - No FIFOs
  - Control unit
    - Peripheral management
  - Register file
    - 5 registers
    - Detailed description in the next slides
    - Not directly accessible from the On-Chip Bus
  - Bus interface



Altera Nios II UART Peripheral

Offset	Register Name	R/W	Description / Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved						Receive Data							
1	txdata	WO	Reserved						Transmit Data							
2	status	RW	Reserved	eop	cts	dcts		e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop	rts	idcts	tbrk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor	RW	Baud Rate Divisor													

- Peripheral registers
  - RXDATA: Receive Data register
  - TXDATA: Transmit Data register
  - STATUS: Status register
  - CONTROL: Control register
  - DIVISOR: Baud Rate Divisor register

Register	Address
rxdata	0x08001060
txdata	0x08001064
status	0x08001068
control	0x0800106C
divisor	0x08001070

Altera Nios II UART Peripheral

- Divisor register
  - Used to generate the UART clock frequency (Baud Rate, BR) starting from the SoC clock frequency

$$DIVISOR = \frac{f_{ck}}{BR} - 1$$

$$BR = \frac{f_{ck}}{DIVISOR + 1}$$

  - Example
    - Obtain a Baud Rate of 28800, with $f_{ck} = 50\ MHz$
      - $DIVISOR = \frac{f_{ck}}{BR} - 1 = \frac{50 \cdot 10^6}{28800} - 1 = 1735$
    - Effective Baud Rate
      - $BR = \frac{f_{ck}}{DIVISOR+1} = \frac{50 \cdot 10^6}{1736} = 28801.8$
  - Integer arithmetic causes some tolerance

Altera Nios II UART Peripheral

- Status register

Bit	Name	Full Name	Description
0	**PE**	**Parity Error**	Set to 1 when parity error is detected. Should be cleared by the processor.
1	FE	Frame Error	Set to 1 when the STOP bit was not detected correctly. Should be cleared by the processor.
2	BRK	Break Error	Set to 1 when break is detected (line at 0 for more than a frame duration). Should be cleared by the processor.
3	**ROE**	**Receive Overrun Error**	Set to 1 when new data is received before previous data in the RXDATA register was read. Should be cleared by the processor.
4	**TOE**	**Transmit Overrun Error**	Set to 1 when new data is written to the TXDATA register before the previous data is transferred to the TX shift register. Should be cleared by the processor.
5	**TMT**	**Transmit Empty**	Set to 1 when the TX shift register is empty (i.e. the transmit path is idle), otherwise it is 0.
6	**TRDY**	**Transmit Ready**	Set to 1 when the TXDATA register is ready to receive a new character, otherwise it is 0.
7	**RRDY**	**Receive Ready**	Set to 1 when new data is available in the RXDATA register. Automatically set to 0 when RXDATA is read.
8	E	Error	Set to 1 when any of the errors occur. Should be cleared by the processor.

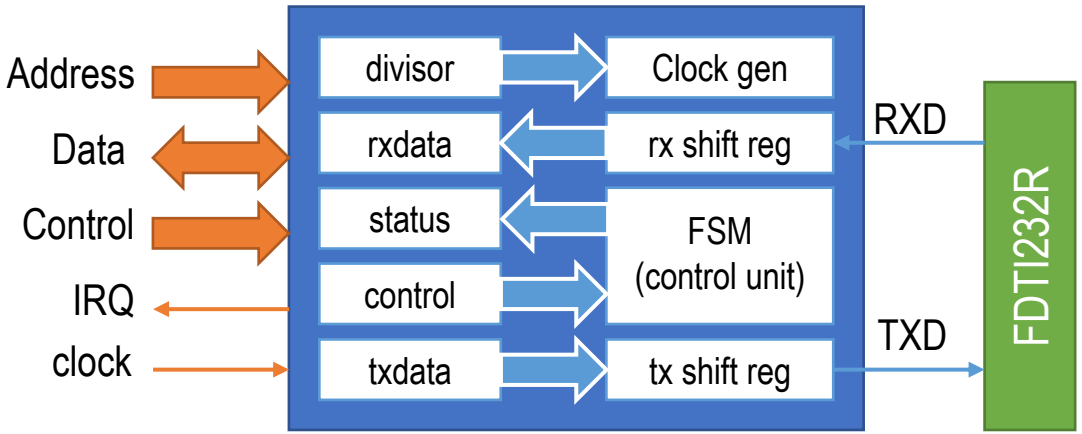
Altera Nios II UART Peripheral

- Control register

Bit	Name	Full Name	Description
0	**IPE**	**Interrupt on Parity Error**	Enable interrupt generation on parity error.
1	IFE	Interrupt on Frame Error	Enable interrupt generation on frame error.
2	IBRK	Interrupt on Break Error	Enable interrupt generation on break detection.
3	**IROE**	**Interrupt on Receive Overrun Error**	Enable interrupt generation on receive overrun error.
4	**ITOE**	**Interrupt on Transmit Overrun Error**	Enable interrupt generation on transmit overrun error.
5	**ITMT**	**Interrupt on Transmit Empty**	Enable interrupt generation on transmit empty.
6	**ITRDY**	**Interrupt on Transmit Ready**	Enable interrupt generation on transmit ready.
7	**IRRDY**	**Interrupt on Receive Ready**	Enable interrupt generation on receive ready.
8	IE	Interrupt on Error	Enable interrupt generation on error.

Altera Nios II UART Peripheral

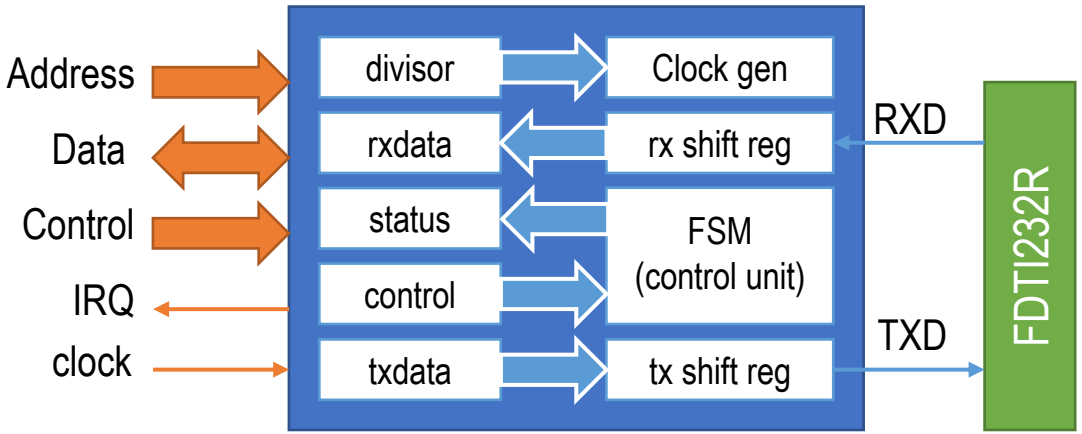
- Sending a character
  - Processor writes a character to the TXDATA register
    - First checks if TXDATA register is ready (TRDY)
  - UART Peripheral actions
    - Mark TXDATA register as full (TRDY = 0)
      - If already full, report overrun (TOE = 1)
    - If there are no ongoing transmissions
      - Transfer character to the TX Shift Register
        - Mark TXDATA register as ready (TRDY = 1)
      - Start UART transmission on serial TX line
    - If there is an ongoing transmission
      - Wait for the transmission to terminate
      - Transfer character to the TX Shift Register
        - Mark TXDATA register as ready (TRDY = 1)
      - Start UART transmission on serial TX line



The µP core writes the **txdata** register. The **tx shift reg** is loaded from **txdata** automatically when a serial transmit shift operation is not currently in progress. The **tx shift reg** directly feeds the TXD output. Data is shifted out to TXD LSB first.
These two registers provide double buffering. A master can write a new value into **txdata** while the previously written character is being shifted out. The master can monitor the transmitter's status by reading the status register's transmitter ready (TRDY), transmitter shift register empty (TMT), and transmitter overrun error (TOE) bits.
The transmitter logic automatically inserts the correct number of start, stop, and parity bits in the serial TXD data stream as required by the UART specification.

Altera Nios II UART Peripheral

- Receiving a character
  - Data arrives on serial RX line
    - It fills the RX shift register
  - UART Peripheral actions (after the STOP bit)
    - Decode UART frame in the RX shift register
    - Transfer character to the RXDATA register
      - Mark RXDATA register as ready (RRDY = 1)
      - If already ready, report overrun (ROE = 1)
  - Processor actions
    - Test RRDY for data available
      - Also check for errors (ROE)
    - If ready, read the character from the RXDATA register
      - UART Peripheral further action
        - Mark RXDATA register as empty (RRDY = 0)



The µP core reads the **rxdata** register. The **rxdata** register is loaded from the **rx shift reg** automatically every time a new character is fully received.
These two registers provide double buffering. The **rxdata** can hold a previously received character while the subsequent character is being shifted into **rx shift reg**.
A master can monitor the receiver's status by reading the status register's receive-ready (RRDY), receiver-overrun error (ROE), break detect (BRK), parity error (PE), and framing error (FE) bits. The receiver logic automatically detects the correct number of start, stop, and parity bits in the serial RXD stream as required by the UART specification. The receiver logic checks for four exceptional conditions, frame error, parity error, receive overrun error, and break, in the received data and sets corresponding status register bits.

Software Project

- Use the Nios II Software Build Tools for Eclipse
  - Like in the second laboratory session
- Write `main()` function
  - Altera library takes care of initializing the Nios II processor
- Include header files

```
#include <stdio.h>
#include "system.h"
#include "sys/alt_timestamp.h"
#include "altera_avalon_pio_regs.h"
```

- Use macros to make your code more general

Software Project

- Read a register
  - You may want to mask unused bits of the integer variable
    - If the register is not on 32 bits

```
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int reg;

reg = *(p + regoffset);
reg = reg & 0x000000ff;
```

- You can also use pointers to void
  - But beware of pointer arithmetic
    - It differs between integer and void pointers
    - It depends on the type size

The automatically generated header file system.h defines macros for the base addresses of peripherals. For instance, line 633 in my implementation is:

```
#define UART_0_BASE 0x8001060
```

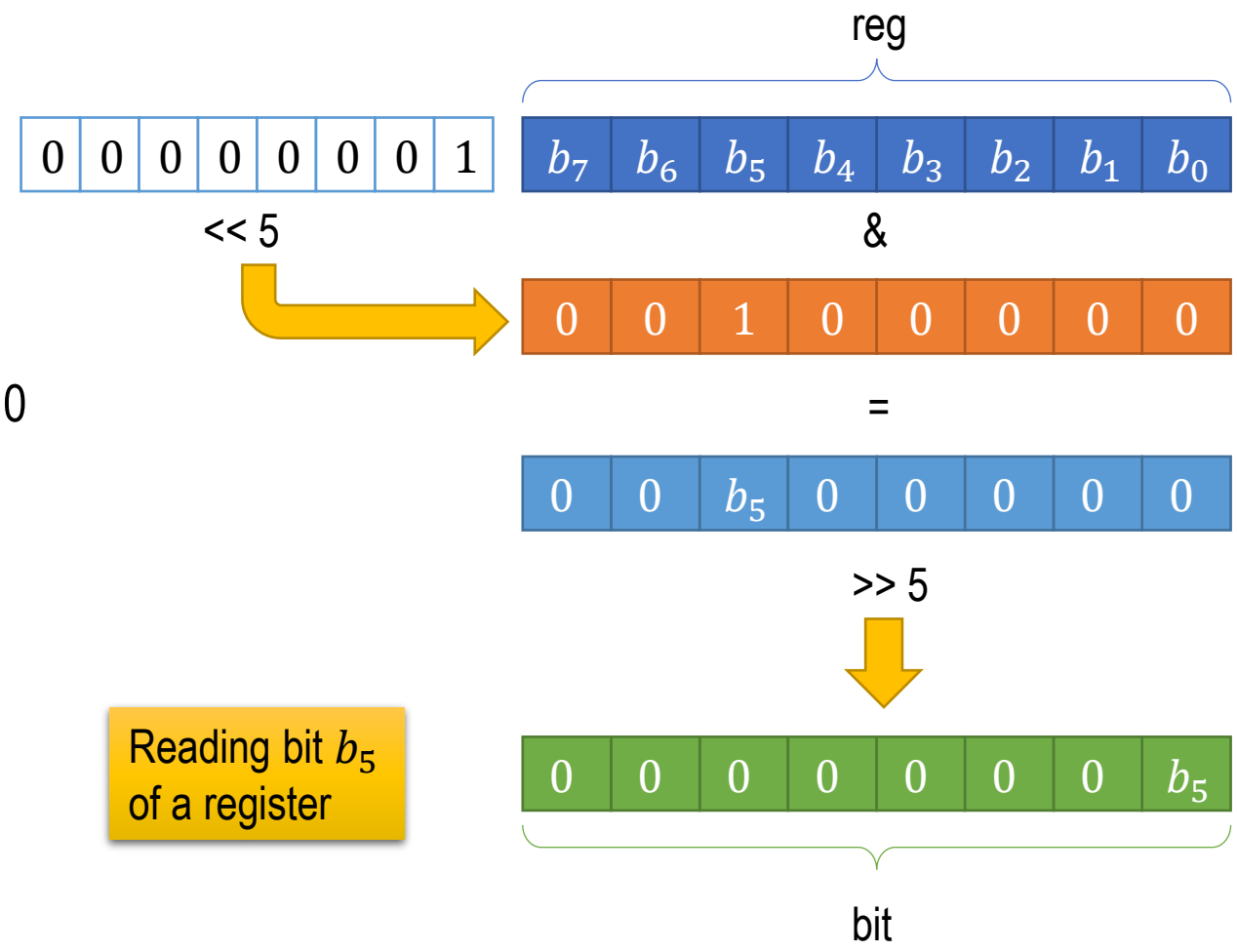
In your code you can assign to the pointer the hexadecimal base address value directly, or you can use the macro (as in the code on the left)

Software Project

- Read a bit in a register
  - Use a mask to isolate a single bit
    - Or multiple bits, if needed
    - AND Boolean function sets all other bits to 0
  - Shift the isolated bit to the LSB
    - No need to shift if only testing if different from 0

```
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int bitoffset = … ;
int reg;
int bit;

reg = *(p + regoffset);
bit = (reg & (1 << bitoffset)) >> bitoffset;
```



Reading bit b_5 of a register

Software Project

- Write a register

```
// Write absolute value
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int reg;

reg = … ;
*(p + regoffset) = reg;
```

```
// Read / Modify / Write
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int reg;

reg = *(p + regoffset);
reg = reg + 1;
*(p + regoffset) = reg;
```

```
// Shadow register (global)
int shadowreg;

// Inside a function
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;

shadowreg = … ;
*(p + regoffset) = shadowreg;

// Some code here

shadowreg = shadowreg + 1;
*(p + regoffset) = shadowreg;
```

- You can also use pointers to void
  - But beware of pointer arithmetic
    - It differs between integer and void pointers
    - It depends on the type size

Software Project

- Update a bit in a register
  - Without modifying other bits in the register
  - Shifting and masking
    - AND with 0 to set it to 0
    - OR with 1 to set it to 1

```
// Read / Modify / Write
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int bitoffset = … ;
int reg, bit;

bit = … ;
reg = *(p + regoffset);
if (bit == 0) reg = reg & (~(1 << bitoffset));
else reg = reg | (1 << bitoffset);
*(p + regoffset) = reg;
```

```
// Shadow register (global)
int shadowreg;

// Inside a function
volatile int *p = (int *) UART_0_BASE;
int regoffset = … ;
int bitoffset = … ;
int bit;

shadowreg = … ;
*(p + regoffset) = shadowreg;

// Some code here

bit = … ;
if (bit == 0) shadowreg = shadowreg & (~(1 << bitoffset));
else shadowreg = shadowreg | (1 << bitoffset);
*(p + regoffset) = shadowreg;
```

Software Project

- Drive the right header connector
  - Set top-left pin to logic 1
    - IOWR_ALTERA_AVALON_PIO_DATA(NIOS_HEADER_CONN_BASE, 1);
  - Set top-left pin to logic 0
    - IOWR_ALTERA_AVALON_PIO_DATA(NIOS_HEADER_CONN_BASE, 0);
  - Set multiple pins
    - $pin_1 = 0, pin_2 = 1, pin_3 = 1, pin_4 = 0, pin_5 = 1, pin_6 = 0, pin_7 = 1, pin_8 = 1$
    - $11010110_2 = D6_{16} = 214_{10}$

```
int value = 214;
IOWR_ALTERA_AVALON_PIO_DATA(NIOS_HEADER_CONN_BASE, value);
```

```
volatile int *head_conn_base_p = (int *) NIOS_HEADER_CONN_BASE;
int datareg_offset = 0;
int value = 214;
*(head_conn_base_p + datareg_offset) = value;
```