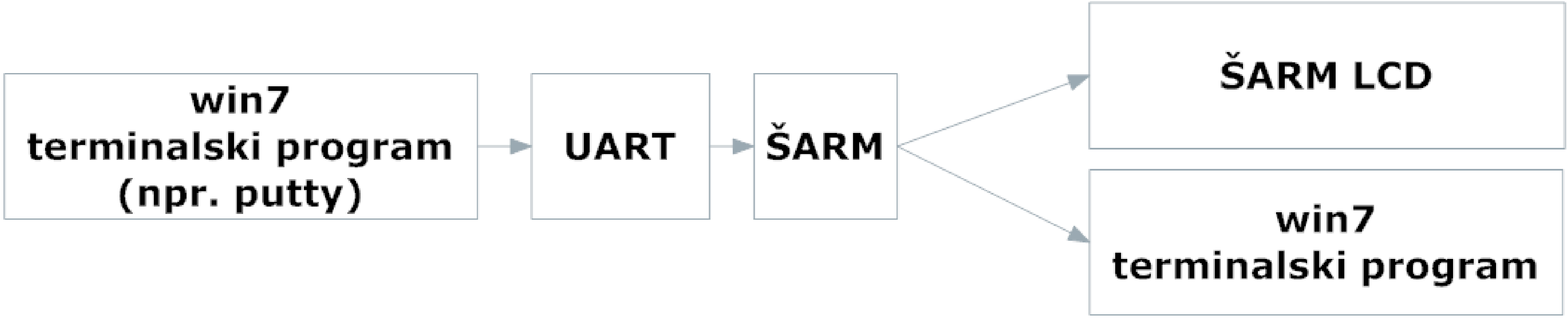
Excercise 6
Universal Asyncronous receiver and transmitter
(UART1)

1. **Write a program, that will allow communication between the ŠARM development board and the PC**
2. **The program must receive the characters from the PC and display them on the LCD.**
3. **Receiving the character ESC (escape, ASCII 27) toggles the output between ŠARM LCD and the PC terminal program**



win7
terminalski program
(npr. putty)

UART

ŠARM

ŠARM LCD

win7
terminalski program

When receiving the ESC character, ŠARM must toggle the output between LCD and the PC)

UART1 (see uart.c, uart.h)

All the macros and bit masks used are defined in uart.h file

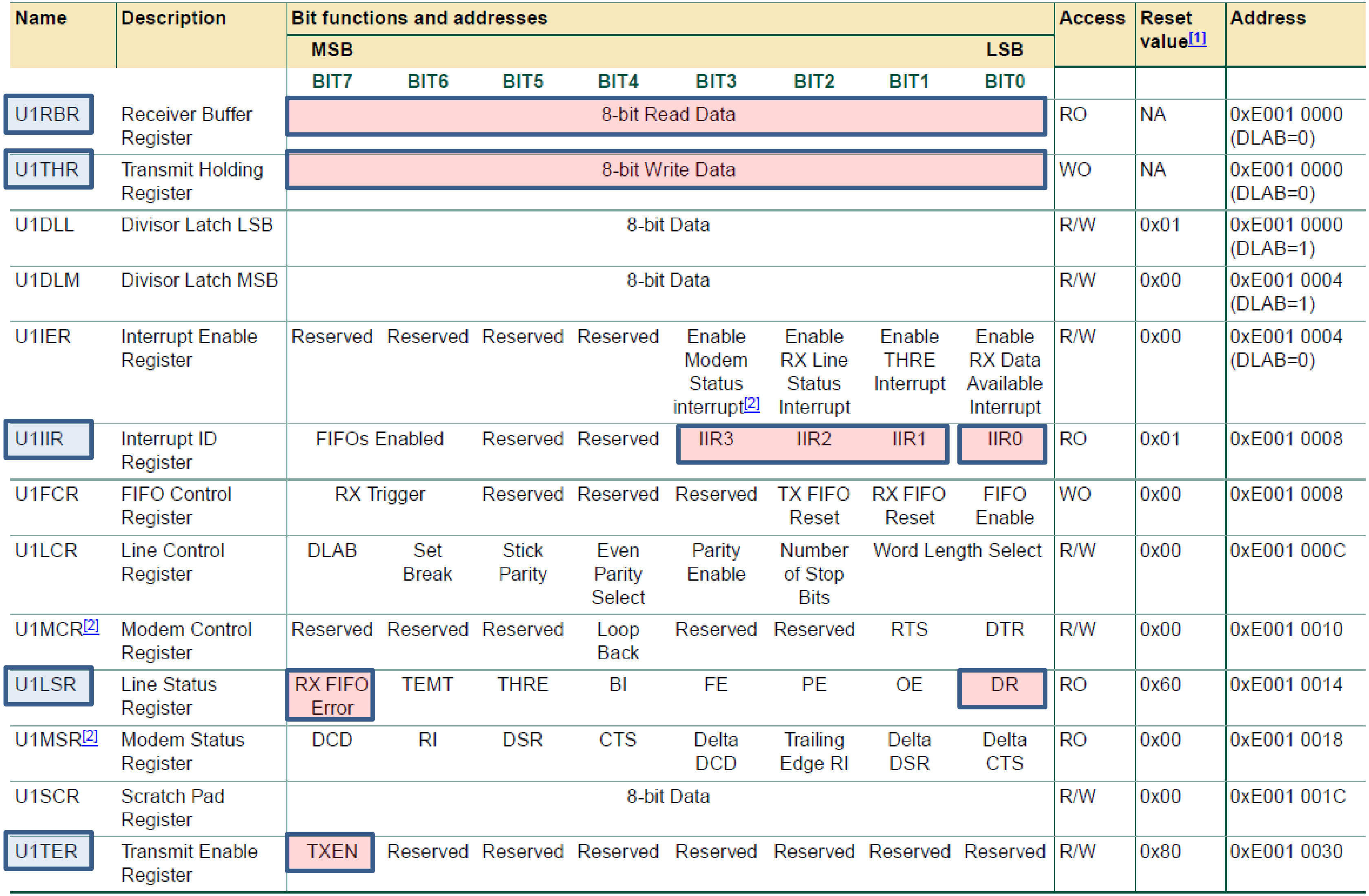
UART1 initialization:
uart1_init(baud_rate,word_length, stop_bit, parity, parity_type, handshake, interrupts);

Function arguments:

- **int baud_rate : baud rate (bit/sec), 9600,14400,...**
- **int word_length : macro** *word_length_5_bit, word_length_6_bit, word_length_7_bit* **or** *word_length_8_bit*
- **int stop_bit : number of stop bits [macro** *one_stop_bit* **or** *two_stop_bits***]**
- **int parity : parity checking [macro** *disable_parity* **or** *enable_parity***]**
- **int parity_type : parity checking type [macro** *odd_parity***,** *even_parity***,** *llways_one_parity* **or** *allways_zero_parity***]**
- **int handshake : enable hardware flow control**
  - **0 => only pins P0.8 (txd) in P0.9 (rxd) are used**
  - **1 => use additional pins: P0.10 (rts), P0.11 (cts), P0.12 (dsr), P0.13 (dtr), P0.14 (dcd) in P0.15 (ri)**
- **int interrupts : enable interrupts [bitwise OR with 0 in flags** *rx_data_available***,** *thre_ier***,** *rx_line_status* **and** *modem_status***]**

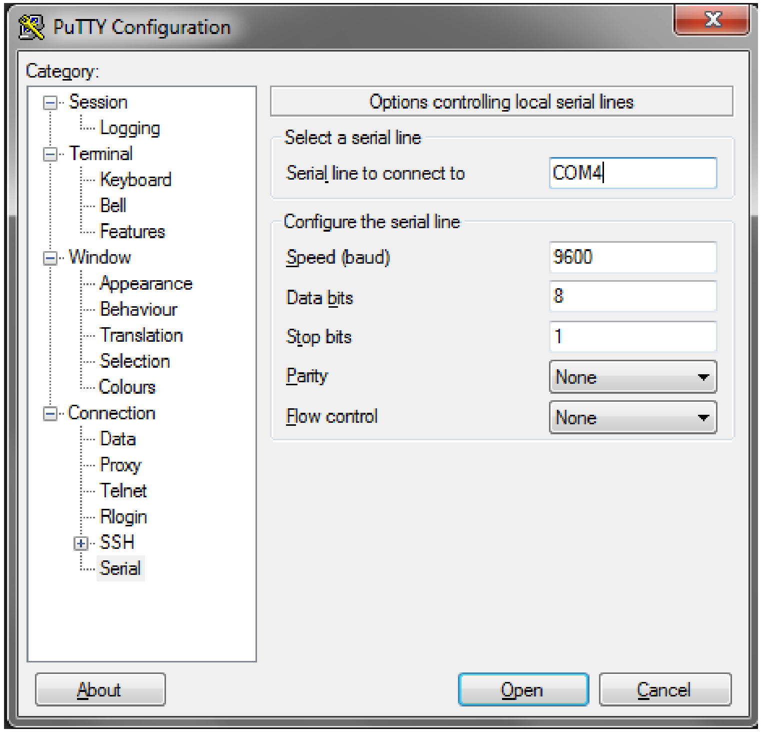
UART1 (uart.c, uart.h)

Registers:



Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U1RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE001 0000 (DLAB=0)
U1DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE001 0000 (DLAB=1)
U1DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE001 0004 (DLAB=1)
U1IER	Interrupt Enable Register	Reserved	Reserved	Reserved	Reserved	Enable Modem Status interrupt[2]	Enable RX Line Status Interrupt	Enable THRE Interrupt	Enable RX Data Available Interrupt	R/W	0x00	0xE001 0004 (DLAB=0)
U1IIR	Interrupt ID Register	FIFOs Enabled		Reserved	Reserved	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE001 0008
U1FCR	FIFO Control Register	RX Trigger		Reserved	Reserved	Reserved	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0x00	0xE001 000C
U1MCR[2]	Modem Control Register	Reserved	Reserved	Reserved	Loop Back	Reserved	Reserved	RTS	DTR	R/W	0x00	0xE001 0010
U1LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE001 0014
U1MSR[2]	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing Edge RI	Delta DSR	Delta CTS	RO	0x00	0xE001 0018
U1SCR	Scratch Pad Register	8-bit Data								R/W	0x00	0xE001 001C
U1TER	Transmit Enable Register	TXEN	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W	0x80	0xE001 0030

Terminal program PUTTY: configuration



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM4

Configure the serial line

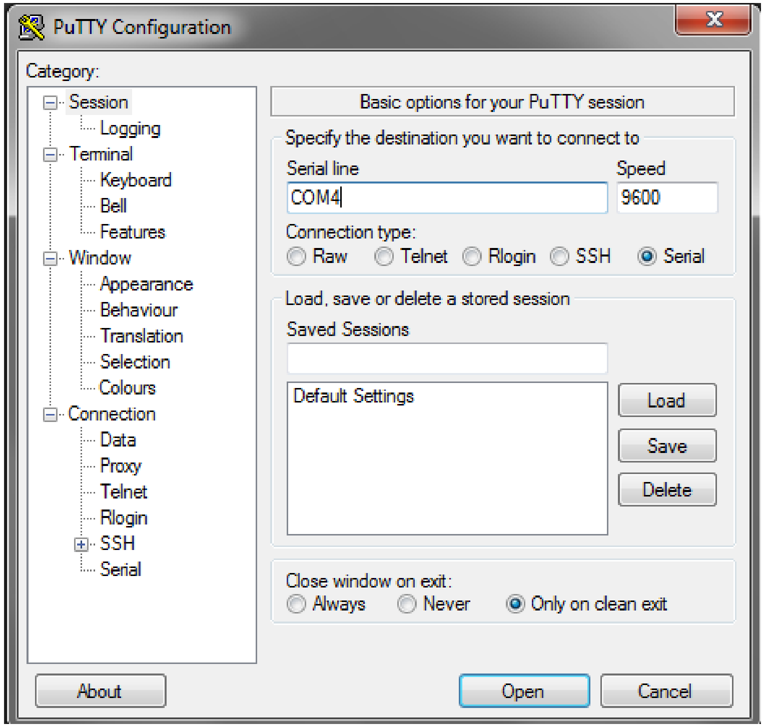
Speed (baud): 9600

Data bits: 8

Stop bits: 1

Parity: None

Flow control: None



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM4

Speed: 9600

Connection type:
○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit:
○ Always ○ Never ● Only on clean exit

Implementation with a loop (polling) :

1. **start_up():**
    1. **uart1_init(...)**
    2. **Enable the transmitter: setting bit 7 (bit mask *txen*) in register *U1TER***

2. **main():**
    1. **Read the line status (reg. *U1LSR*) => store it in a variable (e.g. *status)***
    2. **Cleard bit 0 (bit masks *rdr*) in *status* indicates that new data is available**
    3. **Data is retrieved from register *U1RBR***
    4. **Cleared bit 7 (bit mask *rxfe*) in line *status* indicates that no errors occured during transmition => data is valid**
    5. **If data is valid**
        1. **If the received character is ESC (ascii 27) => toggle the output**
        2. **Display the character on the LCD or**
        3. **send the character back to the PC (by writing it to the transmit holding register *U1THR)***

Implementation with interrupts:

start_up():

1. Initialize the VIC: enable uart1 interrupts
2. When initializing uart1, enable *rx_data_available* interrupt. It is triggered when:
    1. new data is available
    2. when timeout occurs (data remained unread for too long)

Interrupt service routine uart1_ISR():

1. Read the status of the interrupt (reg. *U1IIR*) => store in variable (e.g. *status)*
2. Cleard bit 0 (bit mask *interrupt_pending*) in *status* => new interrupt is pending
3. The source of the interrupt is decoded from bits 3-1 (bit mask *interrupt_id*) in *status*
    1. 0x4 (macro *rx_data_available_id*) => new data received
        1. read data from *U1RBR* (will also clear the uart interrupt reques flags)
        2. process the character (toggle output, or display on LCD/PC)
    2. 0xE (makro *character_time_out_id*) => timeout occured
        1. Read the data from *U1RBR* (will also clear the interrupt request)
            1. Data is not valid so no other action is ŁŁrequired)
4. Reset VIC