Computer Architecture and Organization

8051 Project 3: LED blink (in Simulation)

Project's Goal

- Controlling of 8051's parallel ports P0 to P3
- Programming timers
- Handling interrupts

Assume that each pin of port P1 of the 8051 microcontroller is connected to an LED. According to the output value of port P1 the LEDs are switched on or off, where "1" for "on" and "0" for "off". The goal is to develop programs in assembly and C that control the output ports of the 8051 in such a way that the LEDs connected to this port blink in a defined fashion. Since our exercises are simulator based, we will perform some configurations for ports, timers and interrupts etc. and observe the blinking effect of the LEDs with the simulation tools provided by the Keil software.

Note that the timing effect with simulation could show some deviation from real hardware, but we take the logical correctness of the programming and the concepts behind it for most important in this exercise project.

Task 1 Busy loop LED blink

1.1 Busy loop

Write an assembly subroutine BUSYWAIT which waits for a specified amount of time before returning:

- write a loop whose execution time is approx 10 ms, assuming 11.0592 MHz system clock and the 8051 timing (1 machine cycle equals to 12 oscillator cycles).
- use R5 to pass the factor to the subroutine which makes it wait R5 · 10 ms

1.2 Port control in assembly

Use the subroutine BUSYWAIT for timing the assembly program execution.

1.2.1 Toggle all the pins of a port

Write an assembly program which switches the state of the port P1 between two distinct states (e.g. AAh and 55h or 00h and 0FFh etc.). Make sure that there are approx. 500 ms between state changes.

Hints: In Debug mode, the state of the ports can be shown by selecting *Peripherals → I/O Ports → Port 1*:

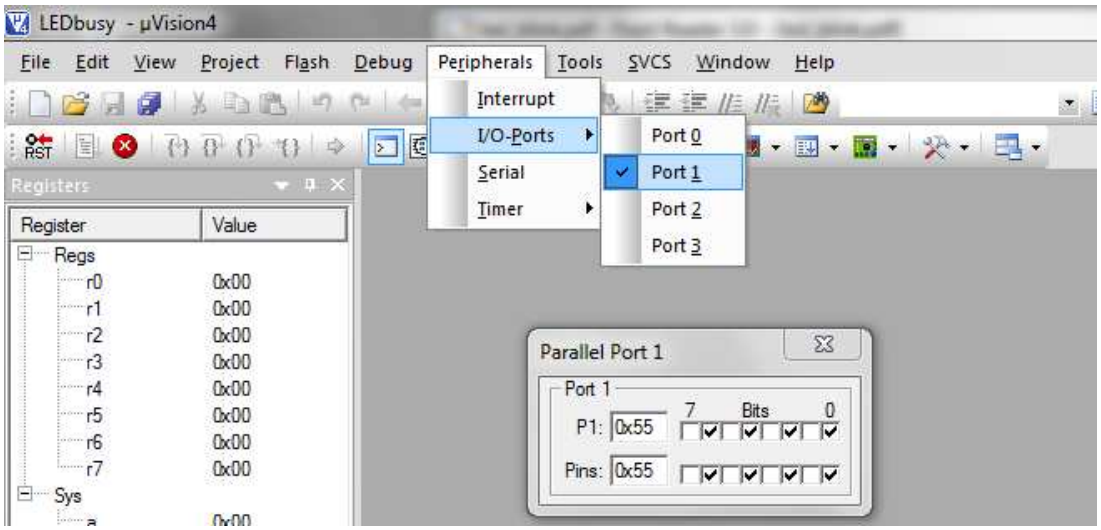

Figure 1: Observe the state of the ports in simulation

You might also observe the signal form (e.g. of the pin P1.0) with logic analyzer:

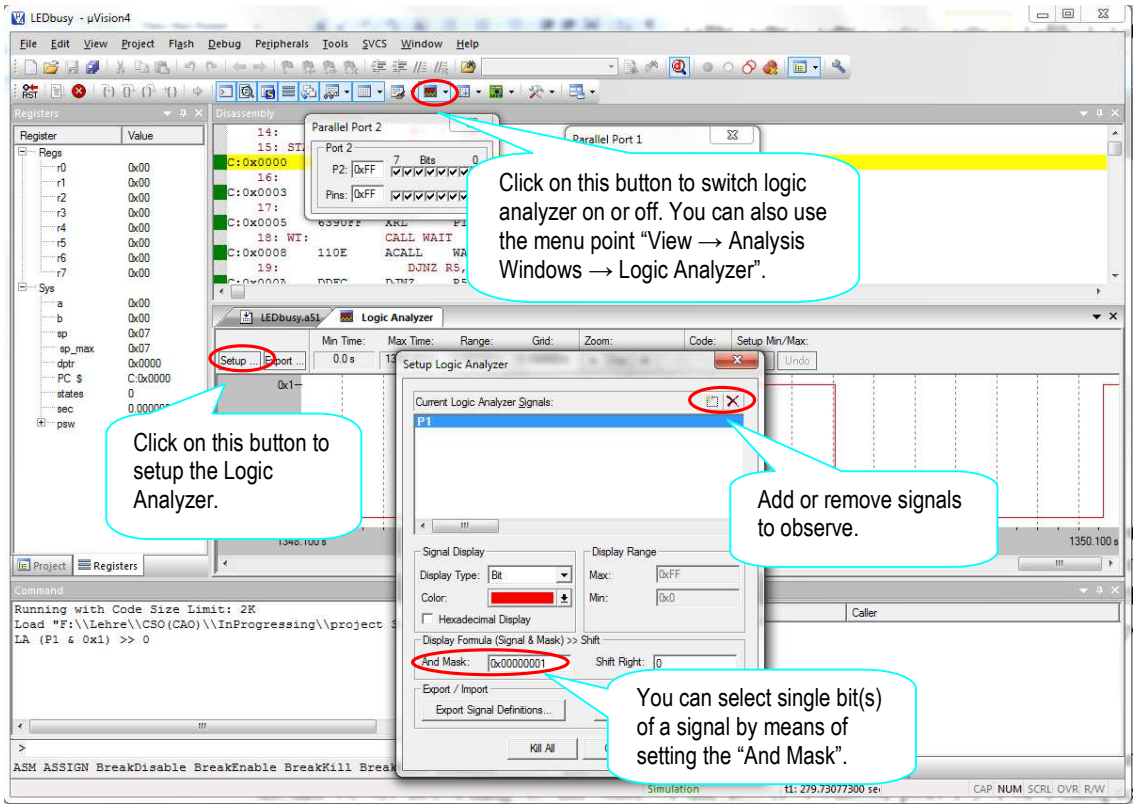

Figure 2 Observe signals with Logic Analyzer

1.2.2 Generate PWM signal at a pin

The light intensity of an LED can be controlled by changing the amount of time it remains in the on state. In order to observe dimming effect on an LED, we need a PWM (Pulse Width Modulation) signal that periodically switches the LED on and off with certain ratio.

Write an assembly program which generates a PWM signal at pin 0 of port P1 conforming to the following requirements:

- The period of the PWM signal should be set to 150ms.
- There are 16 possible duty-cycles, namely 0/15, 1/15, 2/15, …, 15/15. The duty-cycle should be set according to the state of the lower 4 bits of port P2. For example, if "0111" appear at the lower 4 pins (P1.3 through P1.0) of port P2, the duty-cycle should be then 7/15.

Hints:

The term **duty cycle** describes the proportion of "on" time to the regular interval or period "off" time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on.

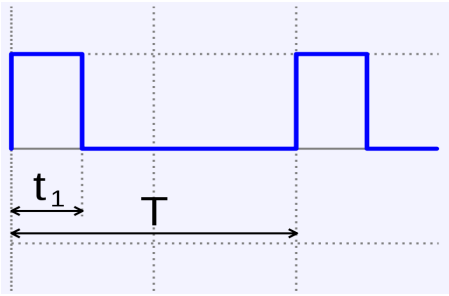

Figure 3 PWM-Signal with a duty cycle t_1 / T = 0.25 = 25 %

1.3 Calculate two's complement of an integer with C51

Write a C51 program that reads the state of ports P1 and P0. Regard P1 as MSB and P0 as LSB of a 16-bit integer, calculate its 2's complement and output the result to ports P3 and P2, where P3 is used for MSB and P2 is used for LSB.

Hints:

- `#include <REG51.h>` to enable the SFR names known by the assembler in the C-code.
- You can solve the problem using either an `unsigned int` or a `signed int`. The code that needs to be written in both cases is very different. Does the simpler C implementation translate into better (faster, smaller) machine code?

Task 2 Timer based LED blink

Unlike in the busy loop project the MCU should wait in idle mode for the next P1 switch. This should be achieved by using timer 0 and timer 0 interrupt. The state of the corresponding SFRs can be seen by selecting the appropriate entry from the *Peripherals* menu.

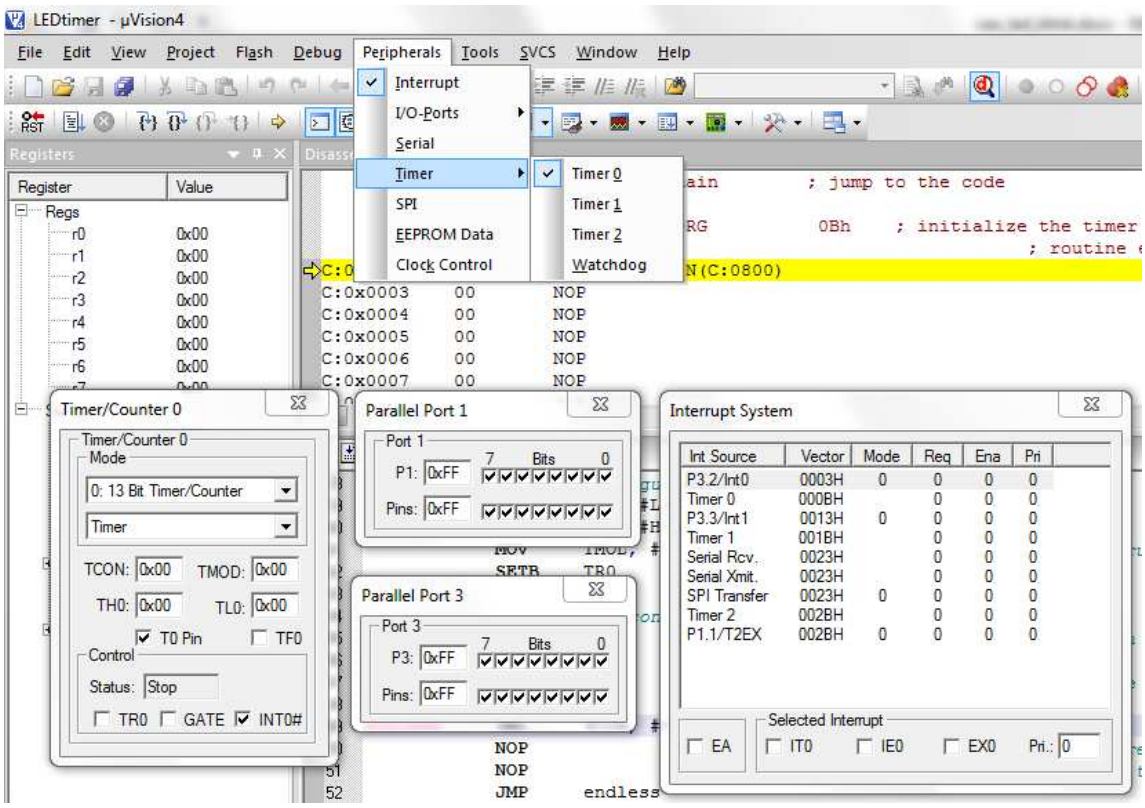

Figure 4 Observe the states of Timer/Counter, Ports and Interrupt System

2.1 Code placement

Use suitable SEGMENT, RSEG, CSEG and ORG directives to:

- reserve 10 bytes for stack area,
- reserve 2 bytes for variable storage,
- place the jump to the interrupt service routine (ISR) for timer 0 at the required location in code memory,
- make sure that the main program execution (starting at address 0) does not interfere with the ISR code.

2.2 Timer configuration

In the main program configure the timer such that:

- timer overflow occurs every 10 ms (assume 11.0592 MHz system clock),
- 16-bit mode is used,
- the timer is started.

2.3 Interrupt configuration

In the main program configure the interrupts such that:

- timer 0 interrupt is enabled and all other interrupt sources remain unchanged,
- the timer 0 priority is set to low,
- the interrupts are enabled.

2.4 Idle wait

In the main program write an infinite loop in which:

- the CPU is put into idle mode,
- after waking up:
  - at least two cycles pass before continuing,
  - the CPU returns again in the idle mode.

2.5 Interrupt service routine (ISR)

Write an interrupt service routine for the timer 0 which:

- saves the current CPU state to the stack,
- reinitializes the counter value,
- checks if the specified number of interrupt overflows has occured; for that use the two reserved memory locations (give a meaningful names to them):
  - location 1: used to store the total number of timer overflows to wait,
  - location 2: used to store the remaining number of timer overflows to wait until port state switch,
- do the state switch if it is due,
- restore the CPU state from the stack.

2.6 Timer Control and Interrupt Handling with C51

Solve the subtasks 2.2 through 2.5 by writing a C program instead of an assembly program.