Programming Embedded Systems – LAB01 report

Stefano Allegretti 514050
Nico Ferrari 514034

Task 2

First, we change the CPU frequency to 12MHz from OSC0.
Then, the program turns on LED6 on button0 push, using polling method.

```
// Include Files
#include "board.h"
#include "gpio.h"
#include "pm.h"


int main(void) {

    volatile avr32_pm_t* pm = &AVR32_PM;

    //setting CPU frequency
    pm_switch_to_osc0(pm, FOSC0, OSC0_STARTUP);

    //infinite cycle for polling approach
    while(1){
        if(GPIO_PUSH_BUTTON_0_PRESSED==gpio_get_pin_value(GPIO_PUSH_BUTTON_0)){
            LED_On(LED6);
        }
        else
            LED_Off(LED6);
    }
    return 0;
}
```

Task 3

Here, we measure the time spent by the CPU for multiplication of two different data types, the unsigned int and the float.
Then, we show the result on the LCD display, initialized by a proper function.

```
void init(void) {
    static const gpio_map_t DIP204_SPI_GPIO_MAP = {
            { DIP204_SPI_SCK_PIN,
                    DIP204_SPI_SCK_FUNCTION }, // SPI Clock.
            { DIP204_SPI_MISO_PIN, DIP204_SPI_MISO_FUNCTION }, // MISO.
            { DIP204_SPI_MOSI_PIN, DIP204_SPI_MOSI_FUNCTION }, // MOSI.
            { DIP204_SPI_NPCS_PIN, DIP204_SPI_NPCS_FUNCTION } // Chip Select NPCS.
    };

    // Switch the CPU main clock to oscillator 0
    pm_switch_to_osc0(&AVR32_PM, FOSC0, OSC0_STARTUP);

    // add the spi options driver structure for the LCD DIP204
    spi_options_t spiOptions = { .reg = DIP204_SPI_NPCS, .baudrate = 1000000,
            .bits = 8, .spck_delay = 0, .trans_delay = 0, .stay_act = 1,
            .spi_mode = 0, .modfdis = 1 };

    // Assign I/Os to SPI
    gpio_enable_module(DIP204_SPI_GPIO_MAP, sizeof(DIP204_SPI_GPIO_MAP)
            / sizeof(DIP204_SPI_GPIO_MAP[0]));

    // Initialize as master
    spi_initMaster(DIP204_SPI, &spiOptions);

    // Set selection mode: variable_ps, pcs_decode, delay
    spi_selectionMode(DIP204_SPI, 0, 0, 0);

    // Enable SPI
    spi_enable(DIP204_SPI);

    // setup chip registers
    spi_setupChipReg(DIP204_SPI, &spiOptions, FOSC0);

    // initialize delay driver
    delay_init(FOSC0);
}
```

```c
int main(void) {
    unsigned int x = 12345678;
    unsigned int y = 87654321;
    float a = 1234.5678;
    float b = 8765.4321;
    float c;
    float time;
    int start, end;
    long unsigned int z;

    //Initialize LCD display
    init();
    dip204_init(backlight_IO, 1);

    //measure unsigned int multiplication time
    start = Get_sys_count();
    z = x * y;
    end = Get_sys_count();

    time = (float) ((end - start) * 1000000) / FOSC0;
    dip204_printf_string("us=%f cyc=%d    ", time, end - start);

    //measure float multiplication time
    start = Get_sys_count();
    c = a * b;
    end = Get_sys_count();

    time = (float) ((end - start) * 1000000) / FOSC0;
    dip204_printf_string("us=%f cyc=%d", time, end - start);

    return 0;
}
```

The results are:

Optimization level 0

unsigned int:	CPU cycles = 6	microseconds = 0.5
float:	CPU cycles = 53	microseconds = 4.416667

Optimization level 1

unsigned int:	CPU cycles = 1	microseconds = 0.083333
float:	CPU cycles = 1	microseconds = 0.083333

Level 0
Reduce compilation time and make debugging produce the expected results[1].

Level 1
Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With ‘-O’, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time[1].

With optimization level 1, multiplications are skipped because the results aren't used in the rest of the program, so it only takes one clock cycle.

Bibliography

1: Richard M. Stallman and the GCC Developer Community, Using the GNU Compiler Collection, 2003