LAB REPORT: EMBEDDED COMPUTING

LAB IV: DRIVER FOR THE I/O EXPANDER

Tailei WANG, Hao DENG

Electric Vehicle Propulsion and Control (E-PiCo)

ÉCOLE CENTRALE DE NANTES

January 2022

1 Principle

This lab focuses on a standard communication interface (SPI) to interact with another chip. The slave component is a `MCP23S017` I/O Extender from Microchip that adds 2 8-bits GPIOs. The component has 2 versions, one with an `i2c` interface (`MCP23S017`), and the other with a `spi` interface (`MCP23S017`). We will use the `spi` version. The functional diagram is in figure 1.

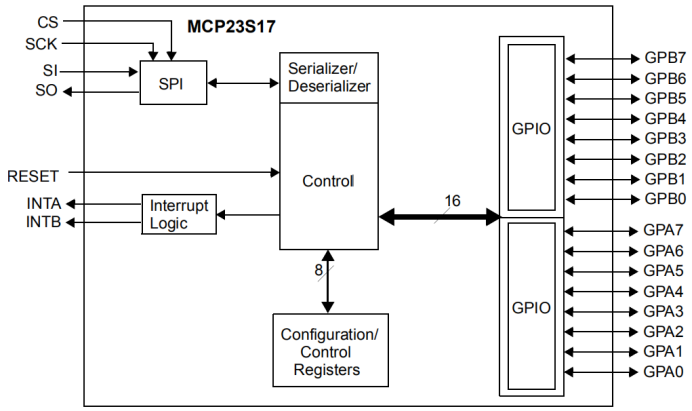

Figure 1: Functional Block Diagram of the (`MCP23S017`).

1.1 Hardware Part

On the board, the component is the one with the board number, at the left of the tft.

The 2 ports are used as:

PORTA 8 leds (seen as EXP A on the board).

PORTB 4 DIP Switches (B0 to B3), and 4 push buttons (B4 to B7).

1.2 Software Part

This lab consists of writing a driver to control the component with high-level functions. The API (Application Programming Interface) is a set of high level functions that are available for an easy use of the component in the application. The lab implements these functions, one after the other.

The architecture of the whole driver is defined in figure 2. The `spi` low level driver is given (see files `spi.c/h`). The driver is organized as a stack of different stages, and arrows shows the relationship between each stage (API functions).

2 Low Level Driver

2.1 Remote Register access

The component is seen as a set of registers that can be read/written using the SPI interface. Microchip defines two modes for the register access (in `IOCON.BANK`) register field, which only have an impact on register addresses. In this lab, we use only the default mode 0. Registers are defined in the datasheet, table 1-2, p.5.

In this section, basic functions to read/write to a remote register are defined (stage "Low level R/W on MCP23S17" on figure 2). The SPI communication frame is defined in figure 3, adapted from figure 1-5 of the datasheet, p.8.

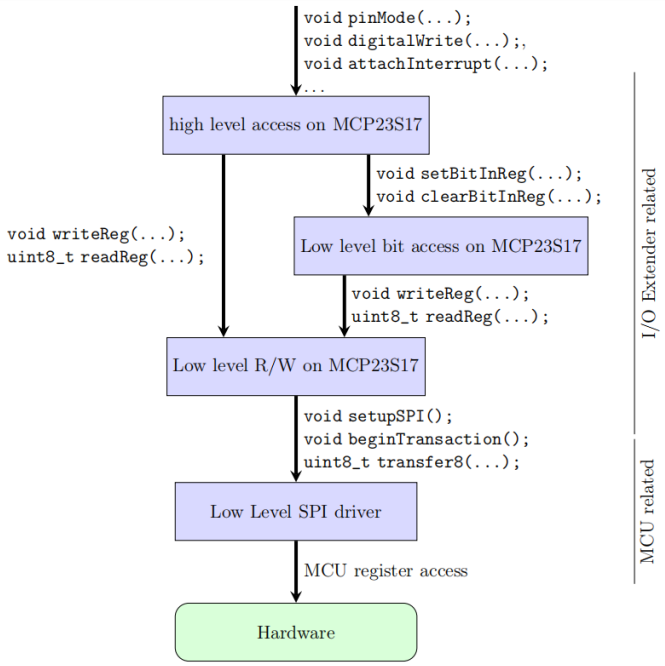

Figure 2: MCP23S017 Driver Architecture.

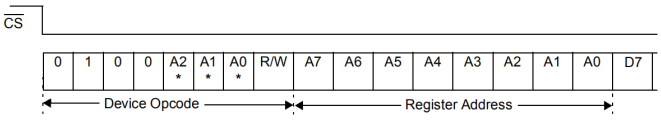

Figure 3: SPI Register Access. The A2-0 bits should be set to 0.

There must be an additionnal byte at the end of the frame: In write mode, this is the data that should be written to the register, and in read mode, this is the answer from the component.

The R/W functions have to set the Chip Select, send a frame of 3 bytes (2W and 1R, or 3W), and unset the chip select.

Question 1 *Write the two low level functions to access to a remote registers.*

The API of the low level driver consists in only 2 functions: `writeReg` and `readReg`. Remote register addresses are defined in advance by using a `#define`.

```
#include "stm32f3xx.h"
#include "pinAccess.h"
#include "spi.h"

enum reg { // mode: IOCON.BANK = 0
    IODIRA = 0x0, IODIRB = 0x1, IOPOLA = 0x2,IPOLB = 0x3, GPINTENA = 0x4, GPINTENB = 0x5,
    DEFVALA = 0x6, DEFVALB = 0x7, INTCONA = 0x8, INTCONB = 0x9, IOCON = 0xA,
    GPPUA = 0xC, GPPUB = 0xD, INTFA=0xE, INTFB = 0xF, INTCAPA = 0x10, INTCAPB = 0x11,
    newGPIOA = 0x12, newGPIOB = 0x13, OLATA = 0x14, OLATB =0x15,
};

void setup() {
```

```
        setupSPI();
}

// write to a MCP register, using spi.
void writeReg(uint8_t reg, uint8_t val){
    beginTransaction(); // CS=0: start SPI write/read operation
    // write Opcode
    transfer8(0x40); // R/W = 0 = write
    // write Register Address
    transfer8(reg);
    // write to SPI1->DR (Data register)
    transfer8(val);
    endTransaction(); // CS=1
    }
// read a MCP register, using spi.
uint8_t readReg(uint8_t reg){
    uint8_t val;
    beginTransaction();
    transfer8(0x41); // R/W = 1 = read
    transfer8(reg);
    // read from SPI1->DR (Data register)
    val = transfer8(reg);
    endTransaction();
    return val;
}
```

2.2 Remote register access: bit access

In this section, we add 2 useful functions to update only one bit of a remote register. This is the stage "Low level bit access on MCP23S17 " of the driver in figure 2.

These two functions do not access directly to the `spi` driver, but use the functions defined in the previous section.

Question 2 *Write these two functions to modify a single bit of a remote registers.*

The two functions are defined as below. Where `reg` is the remote register address defined in the previous section, and `bitNum` the bit number that should be updated.

```
void setBitInReg(uint8_t reg, uint8_t bitNum){
    uint8_t val;
    val = readReg(reg);
    val |= 1<<bitNum;
    writeReg(reg,val);
}

void clearBitInReg(uint8_t reg, uint8_t bitNum){
    uint8_t val;
    val = readReg(reg);
    val &= ~(1<<bitNum);
    writeReg(reg,val);
}
```

3 High Level Driver

3.1 Output mode

The High Level driver stage can now be defined to allows an easy access to the device.

Question 3 *Define the functions of the output mode of the ports. This means:*

- `pinMode()` *that configures a pin as output/ input/ input pullup*;
- `digitalWrite()` *that controls a single pin.*

The code for this question is shown as below. Here OLAT register is used since a write to this register modifies the output latches that modifies the pins configured as outputs.

```
//Q3 definition
enum port {PORTA=0, PORTB=1};
enum mode {MCP_OUTPUT=0, MCP_INPUT=1, MCP_INPUT_PULLUP=2};
enum itType {RISING, FALLING, BOTH};
// configure a pin
// - port is PORTA or PORTB
// - numBit is the pin number (0 to 7)
// - mode is in DISABLE, OUTPUT, INPUT, ...)
void mcpPinMode(enum port p, unsigned char bitNum, enum mode m){
    // PORT A
    if(p == 0){
        if(m == 0){
        clearBitInReg(IODIRA,bitNum); // 0 = Pin is configured as an output.
        }
        else if(m == 1){
        setBitInReg(IODIRA,bitNum); // 1 = Pin is configured as an input.
        clearBitInReg(GPPUA,bitNum);
        }
        else{
        setBitInReg(IODIRA,bitNum);
        setBitInReg(GPPUA,bitNum); // pull-up.
        }
    }
    // PORT B
    else{
        if(m == 0){
        clearBitInReg(IODIRB,bitNum);
        }
        else if(m == 1){
        setBitInReg(IODIRB,bitNum);
        clearBitInReg(GPPUB,bitNum);
        }
        else{
        setBitInReg(IODIRB,bitNum);
        setBitInReg(GPPUB,bitNum);
        }
    }
}
// writing to port register modifies the Output Latch (OLAT) register.
void mcpDigitalWrite(enum port p, unsigned char bitNum, uint8_t value){
```

```
    if (p == 0){
        if (value == 1){
            setBitInReg(OLATA, bitNum);
        }
        else {
            clearBitInReg(OLATA, bitNum);
        }
    }
    else {
        if (value == 1){
            setBitInReg(OLATB, bitNum);
        }
        else {
            clearBitInReg(OLATB, bitNum);
        }
    }
}
```

3.2 First application

To test our driver, we want to make a single chaser with the leds of MCP GPIOA.

Question 4 *Write this single chaser, using your driver and a timer.*

We use `mcpPinMode()` and `mcpDigitalWrite()` established in the last question. And a `delay()` function is applied to make the chaser more visible.

```
void chaser_setup(){
    // input clock = 64MHz.
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
    __asm("nop");
    // reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;
    __asm("nop");
    // leds chaser
    for (uint8_t led=0; led<8; led++){
        mcpPinMode(PORTA,i, MCP_OUTPUT);
    }
}
// ms in milli-seconds
// ms should be <= 60 000
void delay(unsigned int ms){
    // check argument
    int arr = ms;
    if(arr > 60000) arr = 60000;

    TIM6->PSC = 64000-1;    //prescaler : tick@1ms
    TIM6->CNT = 0;
    TIM6->ARR = arr-1;      //auto-reload: counts 100 ticks
    TIM6->CR1 |= TIM_CR1_CEN; //config reg : enable
    while (!(TIM6->SR & TIM_SR_UIF)); //wait...
}
// make a single chaser
```

```
void chaser(){
    chaser_setup();
    uint8_t led = 0;
    while(1){
        if(led >= 0 & led < 8){
            for (uint8_t i=0; i<8; i++){
                if(i = led)
                    mcpDigitalWrite(PORTA,i,1); // led on
                else
                    mcpDigitalWrite(PORTA,i,0); // led off
            }
        }
        // protection
        else{
            for (uint8_t i=0; i<8; i++){
                mcpDigitalWrite(PORTA,i,1);
            }
        }
        led++;
        if(led == 8)
            led = 0;
        delay(2000);
    }
}
```

3.3 Input Mode

The input mode is now easy to write, as the configuration function is already written (`mcpPinMode()`).

Question 5 *write the input read function. We don't provide a* `digitalRead()` *but a function that reads the whole port.*

```
// reading the GPIOn register reads the value on the port.
// reading the OLATn register only reads the latches, not the actual value on the port.
uint8_t readBits(enum port p) {
    uint8_t reading;
    if (p == 0){
        for (uint8_t i=0; i<8; i++){
            mcpPinMode(PORTA, i, MCP_INPUT_PULLUP);
        }
        reading = readReg(newGPIOA);
    }
    else {
        for (uint8_t i=0; i<8; i++){
            mcpPinMode(PORTB, i, MCP_INPUT_PULLUP);
        }
        reading = readReg(newGPIOB);
    }
    return reading;
}
```

Question 6 *update the application (chaser) so that Dip Switch 0 (PORTB.0) defines the direction of the chaser.*

It is convenient to write this application based on the `chaser()` function introduced in question 4. And it should be noticed that the switch needs an input pull-up configuration. Plus, we need to set the whole PORT B because of the mechanism of `readBits()`.

```
void switch_setup(){
    for (uint8_t i=0; i<8; i++){
        mcpPinMode(PORTB,i, MCP_INPUT_PULLUP);
    }
}
// new chaser function.
void chaser_switch(){
    chaser_setup();
    switch_setup();
    uint8_t led = 0;
    while(1){
        if(led >= 0 & led < 8){
            for (uint8_t i=0; i<8; i++){
                if(i = led)
                    mcpDigitalWrite(PORTA,i,1); // led on
                else
                    mcpDigitalWrite(PORTA,i,0); // led off
            }
        }
        // protection
        else{
            for (uint8_t i=0; i<8; i++){
                mcpDigitalWrite(PORTA,i,1);
            }
        }
        if(readBits(PORTB) & 0x01){
            if(led == 0)
                led = 8;
            led--;
        }
        else {
            led++;
            if(led == 8)
                led = 0;
        }
        delay(1000);
    }
}
```