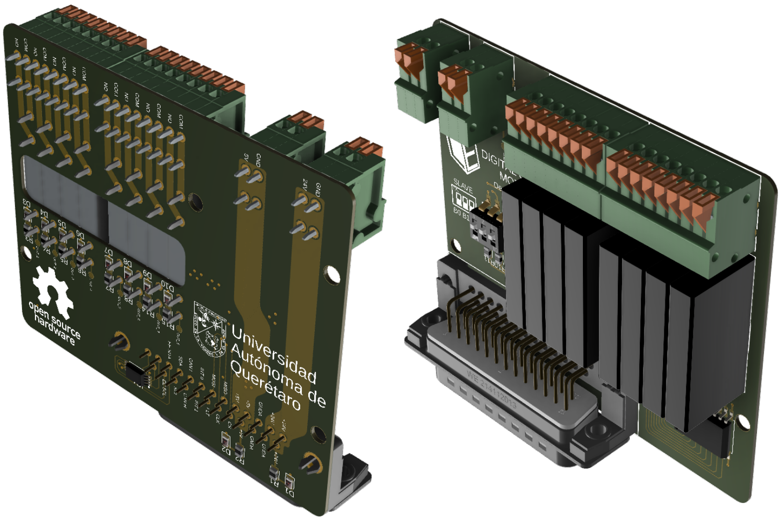
LOW-level Engineering

8 Digital Output Module (Rev. D)

1 Overview

- SPI Serial Interface
- 8 Relay Outputs
- $5V$ and $24V$ Supply Passthrough
- Selectable Slave Address (up to 8 modules per bus)
- Included I2C EEPROM with module identifier
- DB-25 Connector Interface



2 Description

- General purpose output expansion module with SPI serial interface compatible with the DB-25 connectors used on the LOW-level Engineering expansion module base.
- User selectable module address.
- Each module provides an additional 8 digital outputs with SPST Relays to interface with up to $6A$ loads and quick connect spring terminals for ease of use.
- LED indicators are provided for the state of each of the outpus in addition to the supply voltage.
- 2 Additional quick connect spring terminals are provided with supply passthrough to provide a single supply solution for the entire system.
- Integrated I2C EEPROM is provided to save module identification information.
- The MCP32S17 I/O Expander is used as main interface IC. Further information can be found in its own **Datasheet**.
- 4 layer PCB stack-up is used to provide power and signal reference plains (Signal, Power, Ground, Signal).

3 Suggested Applications

- General purpose output expansion module for control applications.
- High current load switching, up to $6A$ at 100 Hz switching frequency.
- SPI to Parallel serial interface for one way data transmission.

4 Technical specification

	Unit	Value		
		Min	Rated	Max
Supply voltage	V	3.3	5	-
Supply current	mA	-	100	350
Internal Logic Level Voltage	V	-	5	-
Operating frequency	MHz	-	8	10
Relay switching frequency	Hz	-		100
Dimensions	mm	67.95 x 80.17 x 13.67		
Weight	g	-	80	-
Operating Temperature range	$^{\circ}C$	0	-	85

5 Connector pinout

5.1 DB-25 Connector

Pin	Signal
1	24V Supply passthrough
2	24V Supply passthrough
3	Ground
4	5V Supply passthrough
5	5V Supply passthrough
6	SPI MISO
7	SPI MOSI
8	Interrupt pin 0 (not available)
9	CAN Bus Low (not available)
10	I2C SDA (not available)
11	I2C EEPROM SDA
12	I2C EEPROM Address pin 0
13	I2C EEPROM Address pin 1
14	I2C EEPROM Address pin 2
15	I2C EEPROM SCL
16	I2C SCL (not available)
17	CAN Bus High (not available)
18	Interrupt pin 1 (not available)
19	Fault pin (not available)
20	SPI CLK
21	SPI CS
22	5V Supply passthrough
23	Ground
24	Ground
25	24V Supply passthrough

5.2 Quick Connect Terminals

Pin	Signal
24V Supply passthrough	
1	Power 24V
2	Ground
5V Supply passthrough	
1	Power 5V
2	Ground
Digital Outputs	
1	Digital output #1
2	Digital output #2
3	Digital output #3
4	Digital output #4
5	Digital output #5
6	Digital output #6
7	Digital output #7
8	Digital output #8

6 Sample Arduino Code

```
//Test for the MCP23S17 16-Bit I/O Expander
#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 7;
const uint8_t PORT_EXPANDER_ADDRESS = 0;
const uint8_t SLAVE_CONTROL_BYTE = 0b1000000 | (PORT_EXPANDER_ADDRESS << 1);

#define    IOCON     (0x0A)
#define    IODIRA    (0x00)
#define    IODIRB    (0x01)
#define    IOPOLA    (0x02)
#define    IOPOLB    (0x03)
#define    GPIOA     (0x12)
#define    GPIOB     (0x13)

uint8_t OUTPUT_PIN_1 = 1; // GPB1
uint8_t OUTPUT_PIN_2 = 2; // GPB2
uint8_t OUTPUT_PIN_3 = 3; // GPB3
uint8_t OUTPUT_PIN_4 = 4; // GPB4
uint8_t OUTPUT_PIN_5 = 5; // GPB5
uint8_t OUTPUT_PIN_6 = 6; // GPB6
uint8_t OUTPUT_PIN_7 = 7; // GPB7
uint8_t OUTPUT_PIN_8 = 8; // GPB8

uint8_t GPIOB_value = 0x00;
uint8_t GPIOA_value = 0x00;


//Command: setup SPI, ports and interrupts.
void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);
  writeByte(IOCON,  0b00001000); // enable hardware address pins; bank=0 addressing
  writeByte(IODIRA, 0xFF);// set input ports
  writeByte(IODIRB, 0x00);// set output ports
}

void loop() {
  //test_outputs();
  test_inputs();
  delay(500);
}

void test_outputs(){
  writeByte(GPIOB, GPIOB_value);
  GPIOB_value = GPIOB_value<<1;
  if (!GPIOB_value) GPIOB_value = 0x01;
}

//Command: write a single byte to the specified register
void writeByte(uint8_t reg, uint8_t data) {
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  SPI.transfer(SLAVE_CONTROL_BYTE);
  SPI.transfer(reg);
  SPI.transfer(data);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}
```

```

//Command: write two bytes to specified register
//demonstrates sequential write and transfer16 alternate SPI method.
void writeSequentialBytes(uint8_t reg, uint8_t first, uint8_t last) {
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  SPI.transfer16((uint16_t)SLAVE_CONTROL_BYTE << 8 | reg);
  SPI.transfer16((uint16_t)first << 8 | last);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}
```

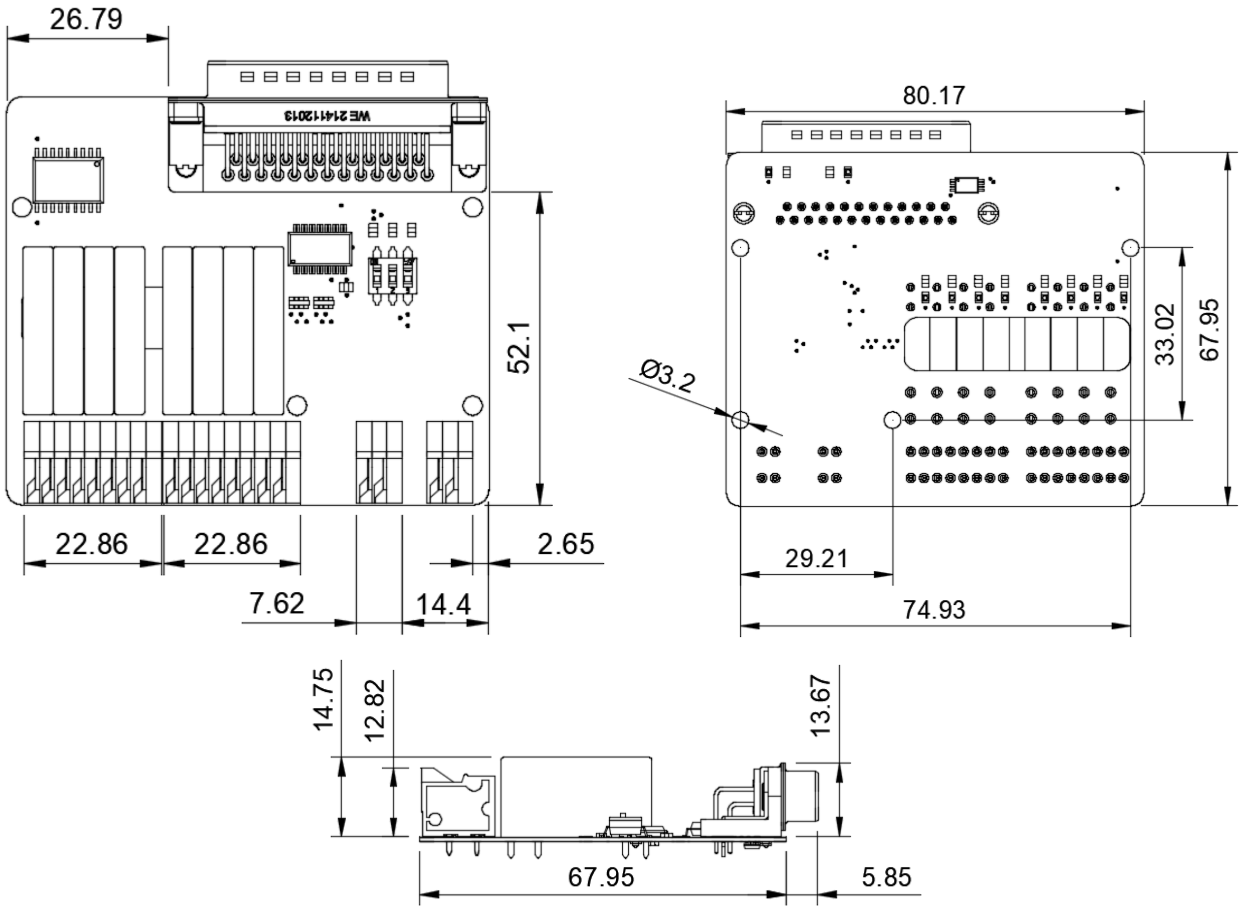
7 Sample NIOS II Test Code Header

```
//-----------------------------------------------------------------------------
//Description : SPI Slave Select & Buffer For Nios II
//-----------------------------------------------------------------------------
#ifndef SPI_H__
#define SPI_H__
/*****************************************************************************
* Public function prototypes
*****************************************************************************/
void SPI_ISR();
unsigned char SPI_EMPTY();
unsigned char SPI_GET_CHAR(unsigned char slave, unsigned char reg);
void SPI_PUT_CHAR(unsigned char slave, unsigned char data);

void slaveSelect(unsigned char spiChannel);
void slaveDeSelect(unsigned char spiChannel);

//External I/O Functions
void MCP23S17_INIT(unsigned char slave, unsigned char address);
void MCP23S17_PUT_CHAR(unsigned char slave, unsigned char address, unsigned char data);
void test_outputs(unsigned char slave, unsigned char address);
#endif /* SPI_H_ */
```

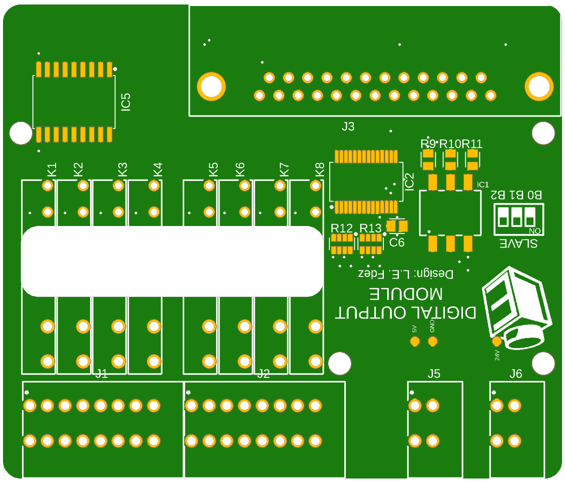
8 Sample NIOS II Test Code

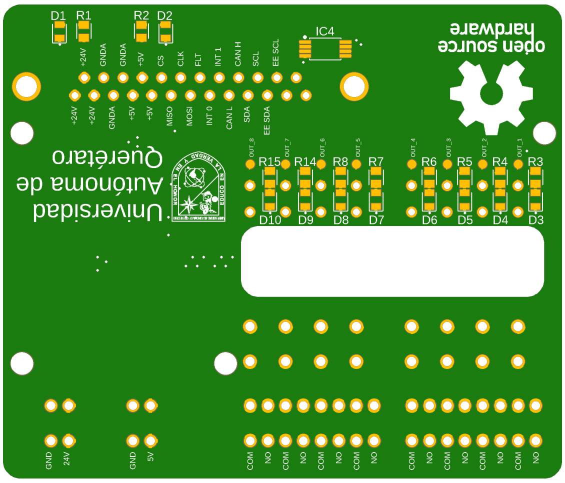
```
//-----------------------------------------------------------------------------
//Description : SPI Functions For Read/Write Operations For Nios II
//-----------------------------------------------------------------------------
#include "system.h"
#include "altera_avalon_spi.h"
#include "altera_avalon_spi_regs.h"
#include "altera_avalon_pio_regs.h"

void slaveSelect(unsigned char spiChannel){
  alt_u16 controlByte;
  if(0 <= spiChannel && spiChannel < 4){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_0_BASE, 1<<0); /* no need to setup slave select register as only one slave but just in case*/
    controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE, (controlByte|ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_0_MUX_BASE, spiChannel);
  }else if(4 <= spiChannel && spiChannel < 8){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_1_BASE, 1<<0); /* no need to setup slave select register as only one slave but just in case*/
    controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE, (controlByte|ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_1_MUX_BASE, spiChannel);
  }
}

void slaveDeSelect(unsigned char spiChannel){
  if(0 <= spiChannel && spiChannel < 4){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_0_BASE, 1<<0); /* no need to setup slave select register as only one slave but just in case*/
    //controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE, 0);//(controlByte|(~ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_0_MUX_BASE, 0);
  }else if(4 <= spiChannel && spiChannel < 8){
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_1_BASE, 1<<0); /* no need to setup slave select register as only one slave but just in case*/
    //controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_1_BASE, 0);//(controlByte|(~ALTERA_AVALON_SPI_CONTROL_SSO_MSK));
    IOWR_ALTERA_AVALON_PIO_DATA(SPI_EXPANSSION_1_MUX_BASE, 0);
  }
}

//MCP23S17 Functions
unsigned char IOCON   = 0x0A;
unsigned char IODIRA  = 0x00;
unsigned char IODIRB  = 0x01;
unsigned char IOPOLA  = 0x02;
unsigned char IOPOLB  = 0x03;
unsigned char GPIOA   = 0x12;
unsigned char GPIOB   = 0x13;

unsigned char GPIOA_value = 0x00;
unsigned char GPIOB_value = 0x00;

//Command: setup port expander.
void MCP23S17_INIT(unsigned char spiChannel, unsigned char address){
  //Data to be transmitted to MCP23S17 to configure the device
  alt_u8 SLAVE_CONTROL_BYTE = 0b1000000 | (address << 1);
  alt_u8  spiData[9] = {SLAVE_CONTROL_BYTE, IOCON, 0x08,
              SLAVE_CONTROL_BYTE, IODIRA, 0xFF,
              SLAVE_CONTROL_BYTE, IODIRB, 0x00};  // Initialization data for Port Expander
  alt_16  status;                       // Avalon SPI Status Register, to check TRDY and RRRDY bits
  alt_u8  i, j;

  if(0 <= spiChannel && spiChannel < 4){
    for (i = 0; i<3; i++){
        slaveSelect(0);
        for (j = 0; j<3; j++){
          do{
            status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_0_BASE);
          } while ((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);
          //wait for tx_ready bit to go high, SPI master

          IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_EXPANSSION_0_BASE,spiData[(3*i)+j]); // 3 8-bit writes to ADC to initialize it

        }
        slaveDeSelect(0);
      }
  }else if(4 <= spiChannel && spiChannel < 8){
    for (i = 0; i<3; i++){
      slaveSelect(0);
      for (j = 0; j<3; j++){
        do{
          status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_1_BASE);
        } while ((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);
        //wait for tx_ready bit to go high, SPI master

        IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_EXPANSSION_1_BASE,spiData[(3*i)+j]); // 3 8-bit writes to ADC to initialize it

      }
      slaveDeSelect(0);
    }
  }
}

//Command: write a single byte to the port expander
void MCP23S17_PUT_CHAR(unsigned char spiChannel, unsigned char address, unsigned char data){
  alt_u8 SLAVE_CONTROL_BYTE = 0b1000000 | (address << 1);
  GPIOB_value = data;                         //Data to be transmitted to MCP23S17 to GPIOB (outputs)
  alt_u8  spiData[3] = {SLAVE_CONTROL_BYTE, GPIOB, GPIOB_value};  // Initialization data for Port Expander
  alt_16  status;                           // Avalon SPI Status Register, to check TRDY and RRRDY bits
  alt_u8  i;

  slaveSelect(spiChannel);
  if(0 <= spiChannel && spiChannel < 4){
    for (i = 0; i<3; i++){
      do{
        status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_0_BASE);
      } while ((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);
      //wait for tx_ready bit to go high, SPI master

      IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_EXPANSSION_0_BASE,spiData[i]); // 3 8-bit writes to Port Expander to initialize it
    }
  }else if(4 <= spiChannel && spiChannel < 8){
    for (i = 0; i<3; i++){
      do{
        status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_1_BASE);
      } while ((status & ALTERA_AVALON_SPI_STATUS_TRDY_MSK) == 0);
      //wait for tx_ready bit to go high, SPI master

      IOWR_ALTERA_AVALON_SPI_TXDATA(SPI_EXPANSSION_1_BASE,spiData[i]); // 3 8-bit writes to Port Expander to initialize it
    }
  }
  slaveDeSelect(spiChannel);
}


void test_outputs(unsigned char slave, unsigned char address){
  MCP23S17_PUT_CHAR(slave, address, GPIOB_value);
  GPIOB_value = GPIOB_value<<1;
  if (!GPIOB_value) GPIOB_value = 0x01;
}
```

9 Physical dimensions

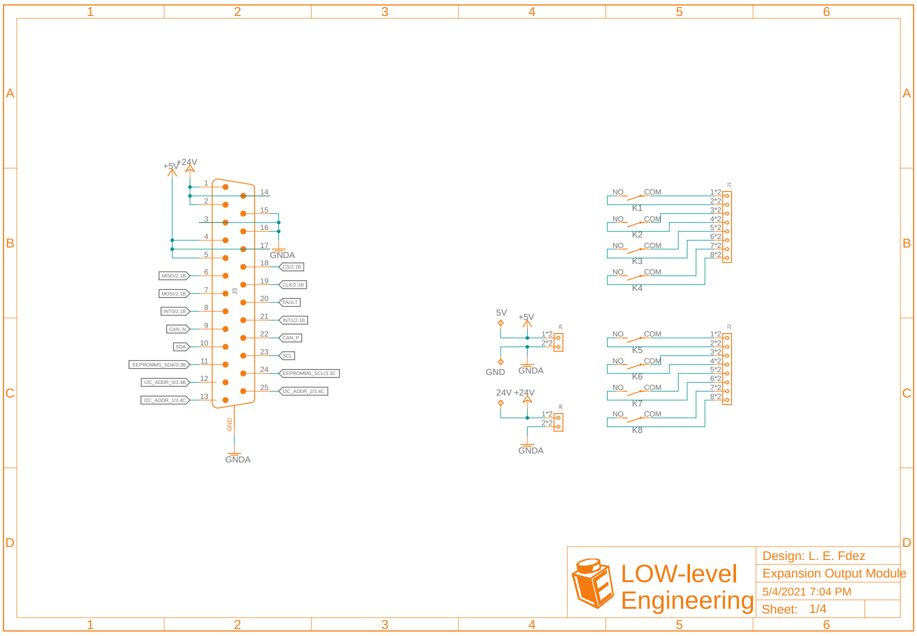


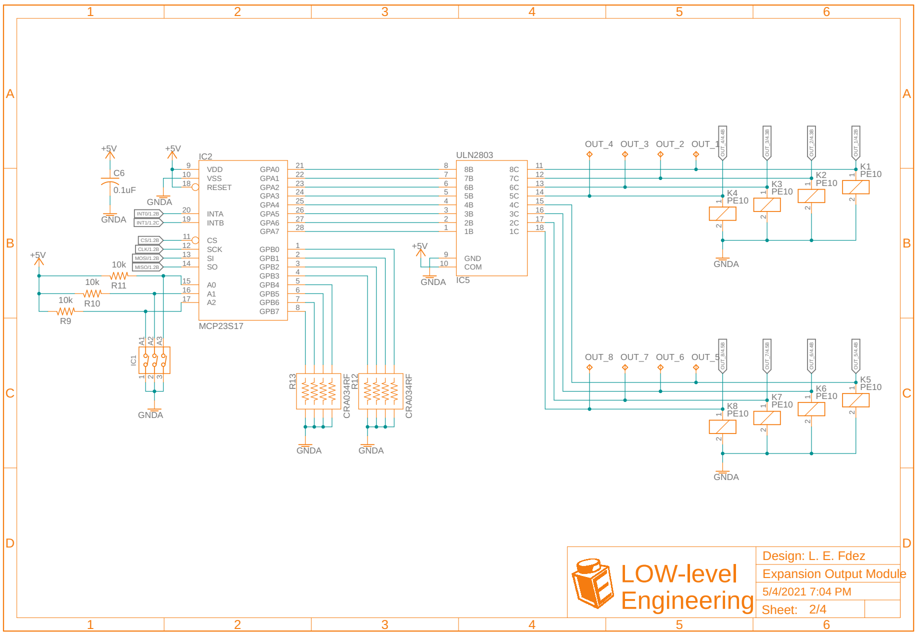
10 Printed circuit board

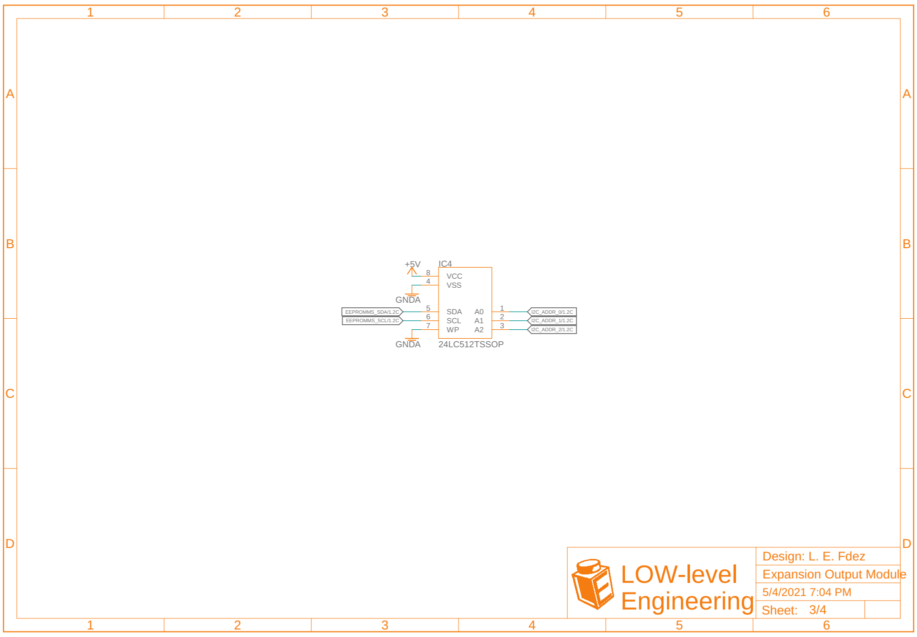


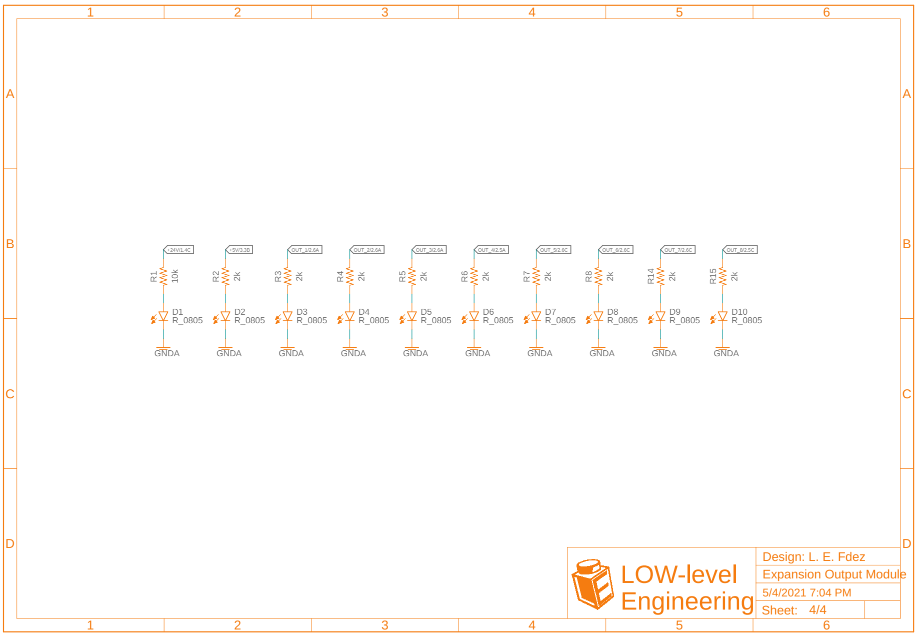

D1 R1
R2 D2
IC4
open source
hardware
+24V GNDA GNDA +5V CS CLK FLT INT 1 CAN H SCL EE SCL
+24V +24V GNDA +5V +5V MISO MOSI INT 0 CAN L SDA EE SDA
Universidad
Autónoma de
Querétaro
R15 R14 R8 R7 R6 R5 R4 R3
D10 D9 D8 D7 D6 D5 D4 D3
GND 24V
GND 5V
COM NO COM NO COM NO COM NO
COM NO COM NO COM NO COM NO

11 Schematic diagram






1
2
3
4
5
6
A
B
C
D
+5V
IC4
8
VCC
4
VSS
GNDA
EEPROMMS_SDA/1.2C
5
SDA
A0
1
I2C_ADDR_0/1.2C
EEPROMMS_SCL/1.2C
6
SCL
A1
2
I2C_ADDR_1/1.2C
7
WP
A2
3
I2C_ADDR_2/1.2C
GNDA
24LC512TSSOP
LOW-level Engineering
Design: L. E. Fdez
Expansion Output Module
5/4/2021 7:04 PM
Sheet: 3/4


1
2
3
4
5
6
A
B
C
D
+24V/1.4C
+5V/3.3B
OUT_1/2.6A
OUT_2/2.6A
OUT_3/2.6A
OUT_4/2.5A
OUT_5/2.6C
OUT_6/2.6C
OUT_7/2.6C
OUT_8/2.5C
R1
10k
R2
2k
R3
2k
R4
2k
R5
2k
R6
2k
R7
2k
R8
2k
R14
2k
R15
2k
D1
R_0805
D2
R_0805
D3
R_0805
D4
R_0805
D5
R_0805
D6
R_0805
D7
R_0805
D8
R_0805
D9
R_0805
D10
R_0805
GNDA
GNDA
GNDA
GNDA
GNDA
GNDA
GNDA
GNDA
GNDA
GNDA
LOW-level Engineering
Design: L. E. Fdez
Expansion Output Module
5/4/2021 7:04 PM
Sheet: 4/4