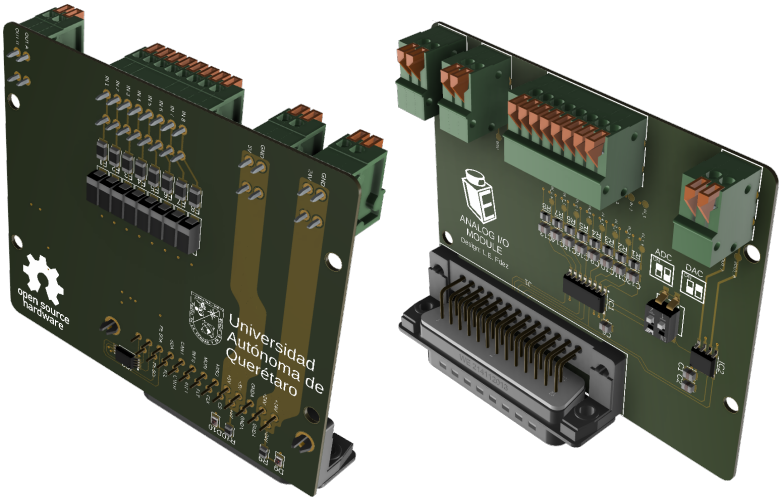
LOW-level Engineering

Analog I/O Module (Rev. D)

1 Overview



- SPI Serial Interface
- Selectable 8 Additional 0-5V Analog Input Ports or 2 Additional 0-5V Analog Output Ports
- 5V and 24V Supply Passthrough
- Included I2C EEPROM with module identifier
- DB-25 Connector Interface

2 Description

- General purpose analog I/O expansion module with SPI serial interface compatible with the DB-25 connectors used on the LOW-level Engineering expansion module base
- Each module provides an additional 8 analog inputs with over-current and over-voltage protection, 2 digital to analog outputs and quick connect spring terminals for ease of use.
- LED indicators are provided for the state of each of the inputs 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 input identification information.
- The MCP3208 analog to digital converter is used as input interface IC. Further information can be found in its own **Datasheet**
- The MCP4822 digital to analog converter is used as output 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 analog I/O expansion module for control applications.
- DAC for external control through voltage reference.
- Analog signal sampling and acquisition.

4 Technical specification

	Unit	Value		
		Min	Rated	Max
Supply voltage	V	2.7	5	-
Supply current	mA	-	10	-
Internal Logic Level Voltage	V	-	5	-
Dimensions	mm	67.95 x 80.17 x 13.67		
Weight	g	-	40	-
Operating Temperature range	$^{\circ}C$	0	-	85
Analog to Digital Converter (ADC)				
Reference voltage	V	-	5	-
Conversion time	$clockcycles$	-	-	12
Analog input sample time	$clockcycles$		1.5	
Throughput rate	$ksps$	-	-	100
Resolution	$bits$		12	
Input voltage range	V	0	-	5
Operating frequency	MHz	-	-	2
Digital to Analog Converter (ADC)				
Reference voltage	V	- 5	-	
Settling time	μs	-	4.5	-
Resolution	$bits$		12	
Output voltage range	V	0.01	-	4.96
Operating frequency	MHz	-	-	20

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
ADC Inputs	
1	Analog input #1
2	Analog input #2
3	Analog input #3
4	Analog input #4
5	Analog input #5
6	Analog input #6
7	Analog input #7
8	Analog input #8
DAC Outputs	
1	Analog output #1
2	Analog output #2

6 Sample Arduino DAC Test Code

```
//Test for the MCP4822 12-Bit ADC

#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 10;
const float SUPPLY_VOLTAGE = 4570;
int voltage = 100;

void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);
}

void loop() {
  writeDAC(0, voltageConvert(voltage));
  writeDAC(1, voltageConvert(voltage*2));
  voltage = voltage + 100;
  if(voltage*2 > SUPPLY_VOLTAGE){
    voltage = 100;
  }
}

uint16_t voltageConvert(float voltage){
  return voltage*4095/SUPPLY_VOLTAGE;
}

```

```
//Command: write dac value to selected output
void writeDAC(uint8_t channel, uint16_t value){
  uint8_t dataOut = 0;
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  dataOut = (channel == 0) ? 0x10 : 0x90;
  dataOut = dataOut ^ (value >> 8);
  SPI.transfer(dataOut);
  SPI.transfer(value & 0x0FF);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
}
```

7 Sample Arduino ADC Test Code

```
//Test for the MCP3202 12-Bit ADC

#include <SPI.h>

SPISettings portExpanderSettings(16000000, MSBFIRST, SPI_MODE0);

const int PORT_EXPANDER_SS_PIN = 10;
float channel1 = 0;
float channel2 = 0;

void setup() {
  pinMode(PORT_EXPANDER_SS_PIN, OUTPUT);
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  SPI.begin();
  SPI.beginTransaction(portExpanderSettings);
  Serial.begin(115200);
}

void loop() {
  channel1 = readADC(0);
  channel2 = readADC(1);
  Serial.print(channel1);
  Serial.print("\t");
  Serial.println(channel2);
}

//Command: request ADC data from selected channel
uint16_t readADC(uint8_t channel){
  uint8_t dataIn = 0;
  uint8_t result = 0;
  digitalWrite(PORT_EXPANDER_SS_PIN, LOW);
  uint8_t dataOut = 0x01;
  dataIn = SPI.transfer(dataOut);
  dataOut = (channel == 0) ? 0xA0 : 0xE0;
  dataIn = SPI.transfer(dataOut);
  result = dataIn & 0x0F;
  dataIn = SPI.transfer(0x00);
  result = result << 8;
  result = result | dataIn;
  //input = input << 1;
  digitalWrite(PORT_EXPANDER_SS_PIN, HIGH);
  return result;
}
```

8 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 ADC Functions
alt_u16 MCP3202_GET_CHAR(unsigned char spiChannel, unsigned char adcChannel);

//External DAC Functions
void MCP4822_PUT_CHAR(unsigned char spiChannel, unsigned char dacChannel, alt_u16 value
    );
alt_u16 mVConvert(alt_u16 voltage);
void test_dac(unsigned char spiChannel);

#endif /* SPI_H_ */
```

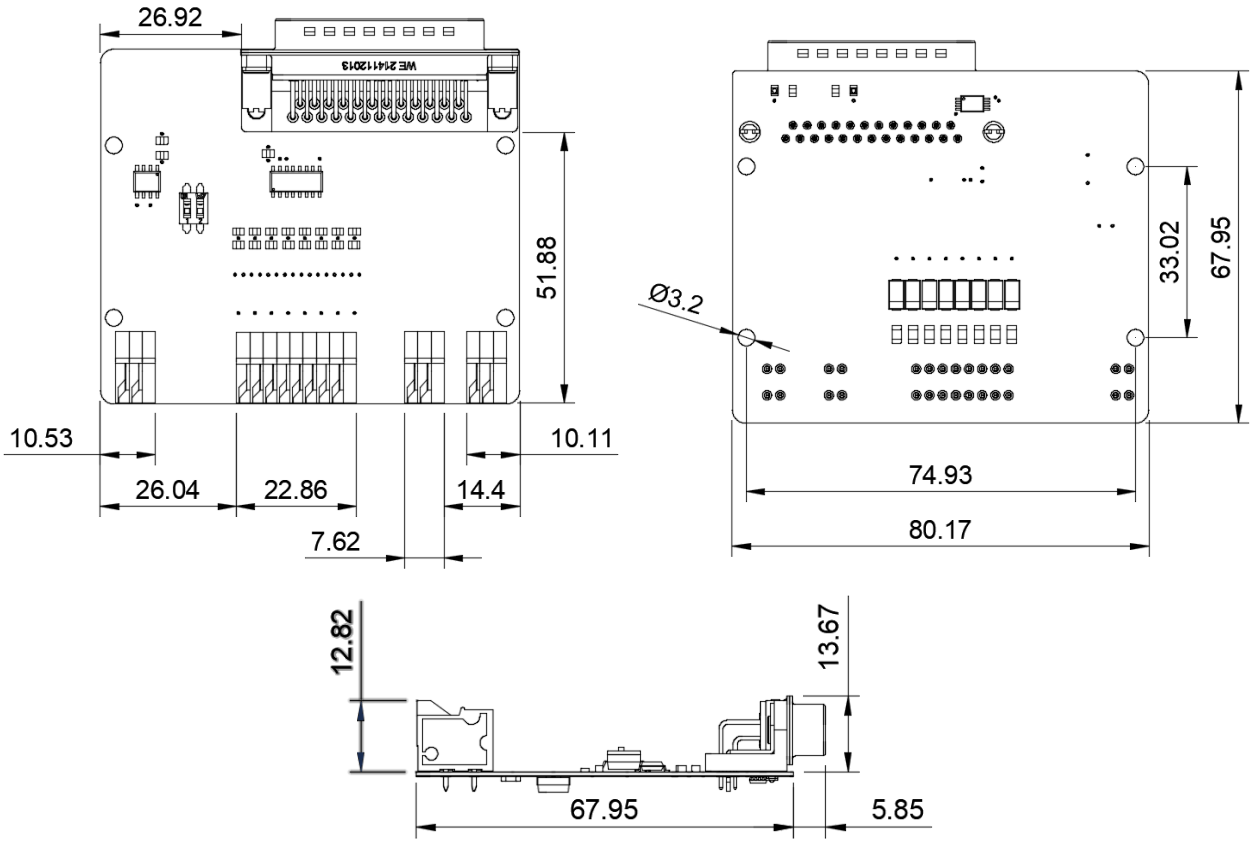
9 Sample NIOS II Test Code

```
//-------------------------------------------------------------------------------
//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){
```

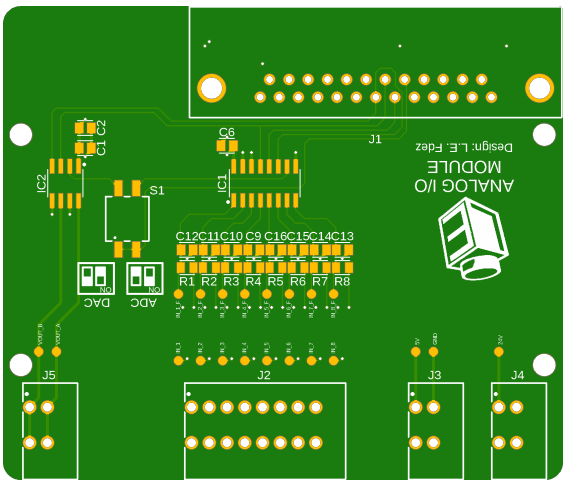
```
    IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_EXPANSSION_0_BASE, 1<<0); /* no need to setup slave select register as only one slave but just in case*/
    //controlByte = IORD_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE);
    IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_EXPANSSION_0_BASE, 0);//(controlByte|(~ALTERA_AVALON_SPI_CONTROL_SSO_MSK)));
    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);
  }
}

//MCP3202 Functions
//Command: request ADC data from selected channel
alt_u16 MCP3202_GET_CHAR(unsigned char spiChannel, unsigned char adcChannel){
  alt_u8  spiData[3] = {0x00};  // Initialization data for ADC
  alt_u8  spiOut[3] = {0x00};   // Digital Data Read From ADC
  alt_u16 response;       // 12-bit response generated from Digital bits
  alt_16  status;         // Avalon SPI Status Register, to check TRDY and RRRDY bits
  alt_u8  i;

  //Data to be transmitted to MCP3208 to start conversion
  spiData[0] = 0x01;
  spiData[1] = (adcChannel == 0) ? 0xA0 : 0xE0;
  spiData[2] = 0x00;

  slaveSelect(0);
  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 ADC to initialize it

      status = 0;
      do{
        status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_0_BASE);
      } while ((status & ALTERA_AVALON_SPI_STATUS_RRDY_MSK ) == 0);

      spiOut[i] = IORD_ALTERA_AVALON_SPI_RXDATA(SPI_EXPANSSION_0_BASE); // return sample
    }
  }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 ADC to initialize it

      status = 0;
      do{
        status = IORD_ALTERA_AVALON_SPI_STATUS(SPI_EXPANSSION_1_BASE);
      } while ((status & ALTERA_AVALON_SPI_STATUS_RRDY_MSK ) == 0);

```

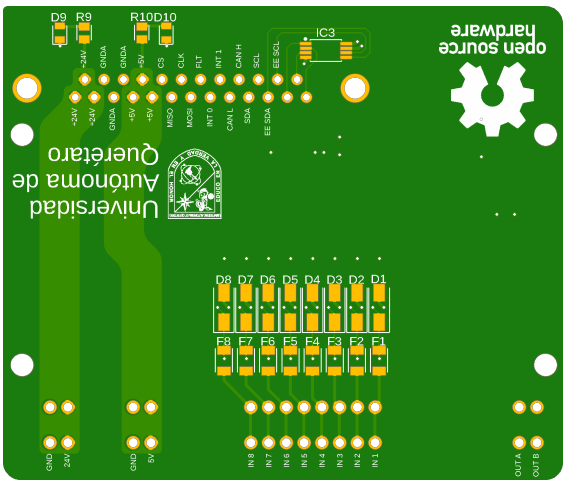
```
      spiOut[i] = IORD_ALTERA_AVALON_SPI_RXDATA(SPI_EXPANSSION_1_BASE); // return sample
    }
  }
  slaveDeSelect(0);
  spiOut[1] = (spiOut[1] << 1) | ((spiOut[2] & 0x80) >> 7);
  spiOut[2] = spiOut[2] << 1;
  response = ((spiOut[1] & 0x0F) << 8) | spiOut[2];
  return response;
}

//MCP4822 Functions
alt_u16 SUPPLY_VOLTAGE = 3300;
alt_u16 DAC_value = 0;

//Command: write dac value to selected output
void MCP4822_PUT_CHAR(unsigned char spiChannel, unsigned char dacChannel, alt_u16 value ){
  alt_u8  spiData[2] = {0x00};  // Initialization data for DAC
  alt_16  status;          // Avalon SPI Status Register, to check TRDY and RRRDY bits
  alt_u8  i;

  //Data to be transmitted to MCP4822 to start conversion
  spiData[0] = (dacChannel == 0) ? 0x10 : 0x90;
  spiData[0] = spiData[0] ^ (value >> 8);
  spiData[1] = value & 0x00FF;

  slaveSelect(spiChannel);
  if(0 <= spiChannel && spiChannel < 4){
    for (i = 0; i<2; 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 ADC to initialize it
    }
  }else if(4 <= spiChannel && spiChannel < 8){
    for (i = 0; i<2; 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 ADC to initialize it
    }
  }
  slaveDeSelect(spiChannel);
}

alt_u16 mVConvert(alt_u16 voltage){
  return voltage*4095/SUPPLY_VOLTAGE;
}

void test_dac_0(unsigned char spiChannel){
  MCP4822_PUT_CHAR(spiChannel, 0, mVConvert(DAC_value));
  MCP4822_PUT_CHAR(spiChannel, 1, mVConvert(DAC_value*2));
  DAC_value = DAC_value + 100;
  if(DAC_value*2 > SUPPLY_VOLTAGE){
    DAC_value = 100;
  }
}
```

10 Physical dimensions

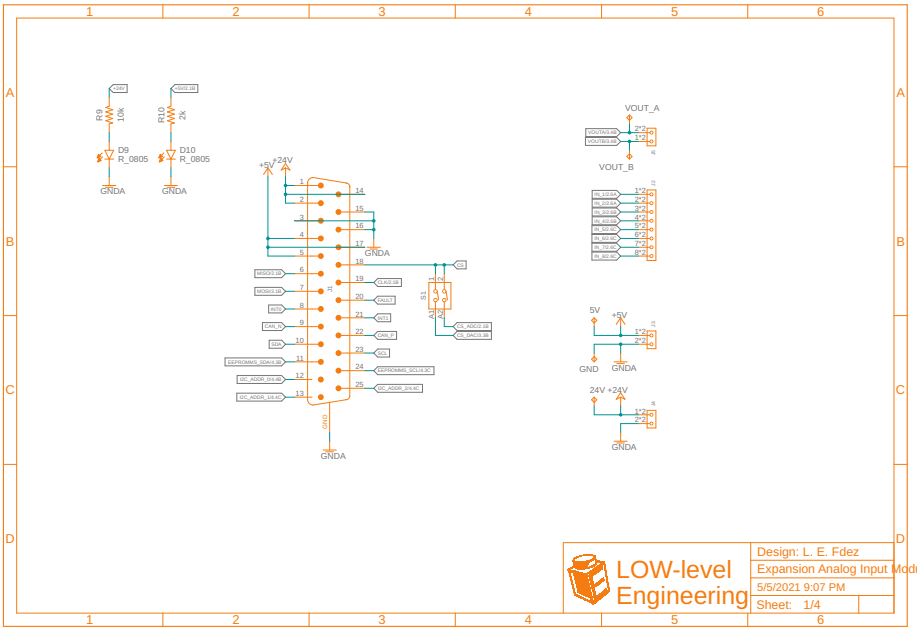


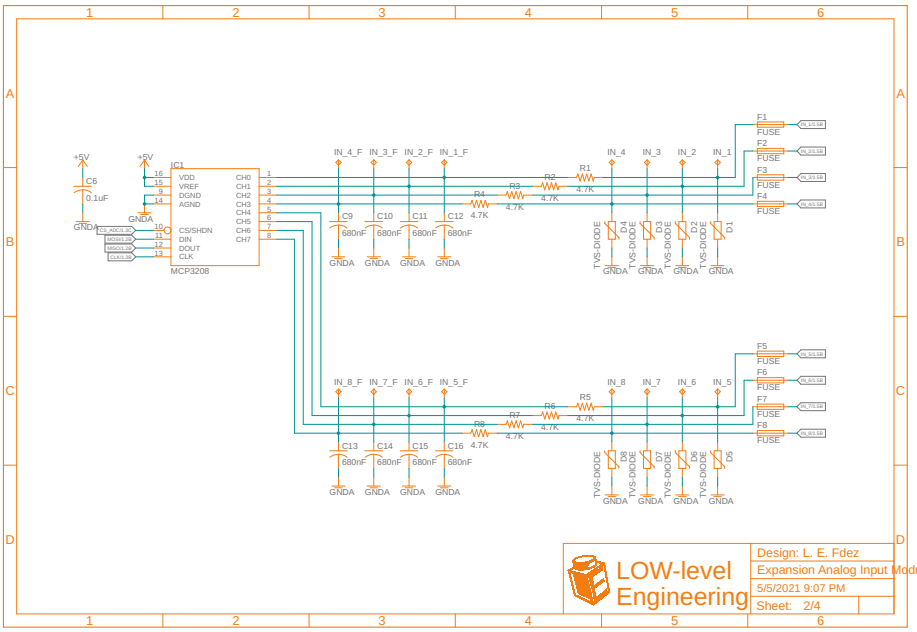
11 Printed circuit board

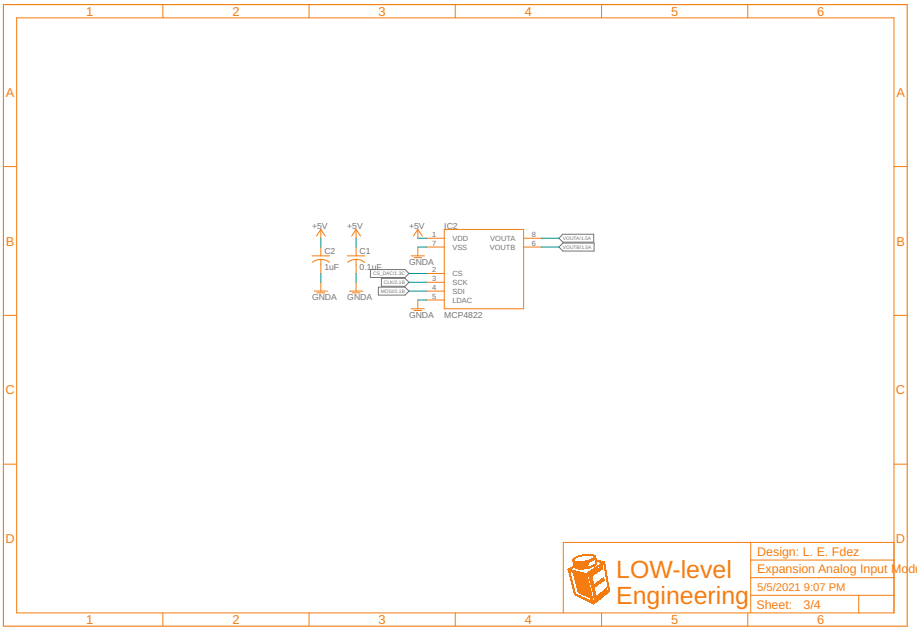


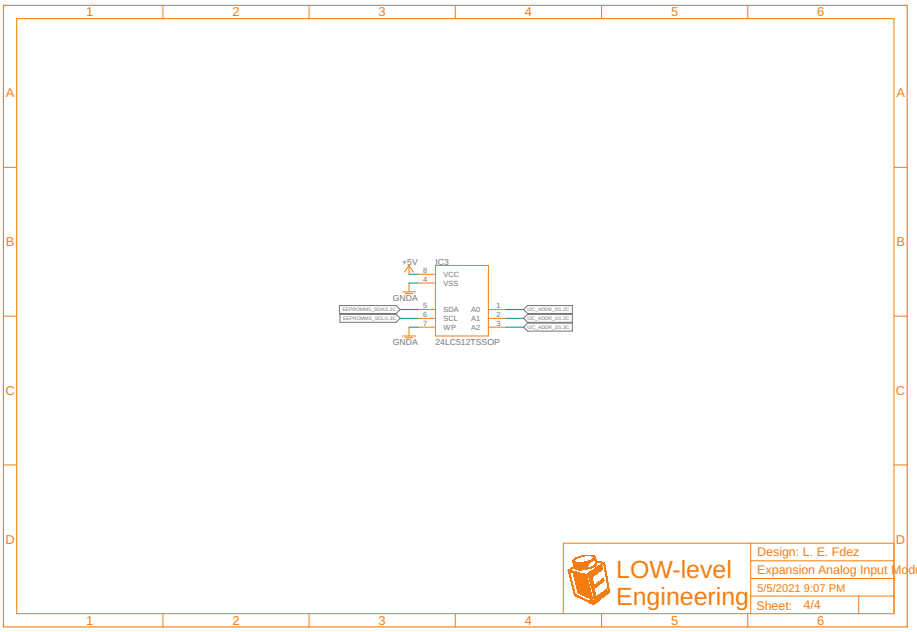


12 Schematic diagram






+5V
C2
1uF
GNDA
C1
0.1uF
IC2
VDD
VSS
VOUTA
VOUTB
CS
SCK
SDI
LDAC
MCP4822
LOW-level Engineering
Design: L. E. Fdez
Expansion Analog Input Module v5
5/5/2021 9:07 PM
Sheet: 3/4


+5V
IC3
VCC
VSS
SDA
SCL
WP
A0
A1
A2
24LC512TSSOP
LOW-level Engineering
Design: L. E. Fdez
Expansion Analog Input Module v5
5/5/2021 9:07 PM
Sheet: 4/4