TG057: SPM / GP DRIVE BOARD

CONNECTED TO ARDUINO UART DEMO

1. INTRODUCTION

1.1. About this Technical Guide

The Smart Pump Module (SPM) and General Purpose drive board (GP) can be controlled through UART. The UART communications protocol is commonly used and simple to set up.

This guide shows how to use an Arduino Mega* to control an SPM and GP over UART and consists of the following stages:

- Connecting the Arduino Mega to the SPM/GP
- Sending and receiving commands
- Example sketch for controlling power and pressure

** Note: This example uses the Arduino Mega as it supports multiple UART connections. One UART connection (Serial) is used to communicate with the computer to send information and a second UART connection (Serial1) is used to communicate with the SPM/GP.*
The Arduino Uno supports only one UART connection and therefore cannot simultaneously communicate with the computer and the SPM/GP. It is still possible to use an Arduino Uno, however it will not be able to communicate with the computer.

2. DISCLAIMER

This resource is provided "as is" and without any warranty of any kind, and its use is at your own risk. The Lee Company does not warrant the performance or results that you may obtain by using this resource. The Lee Company makes no warranties regarding this resource, express or implied, including as to non-infringement, merchantability, or fitness for any particular purpose. To the maximum extent permitted by law The Lee Company disclaims liability for any loss or damage resulting from use of this resource, whether arising under contract, tort (including negligence), strict liability, or otherwise, and whether direct, consequential, indirect, or otherwise, even if The Lee Company has been advised of the possibility of such damages, or for any claim from any third party.

3. HEALTH AND SAFETY

WARNING

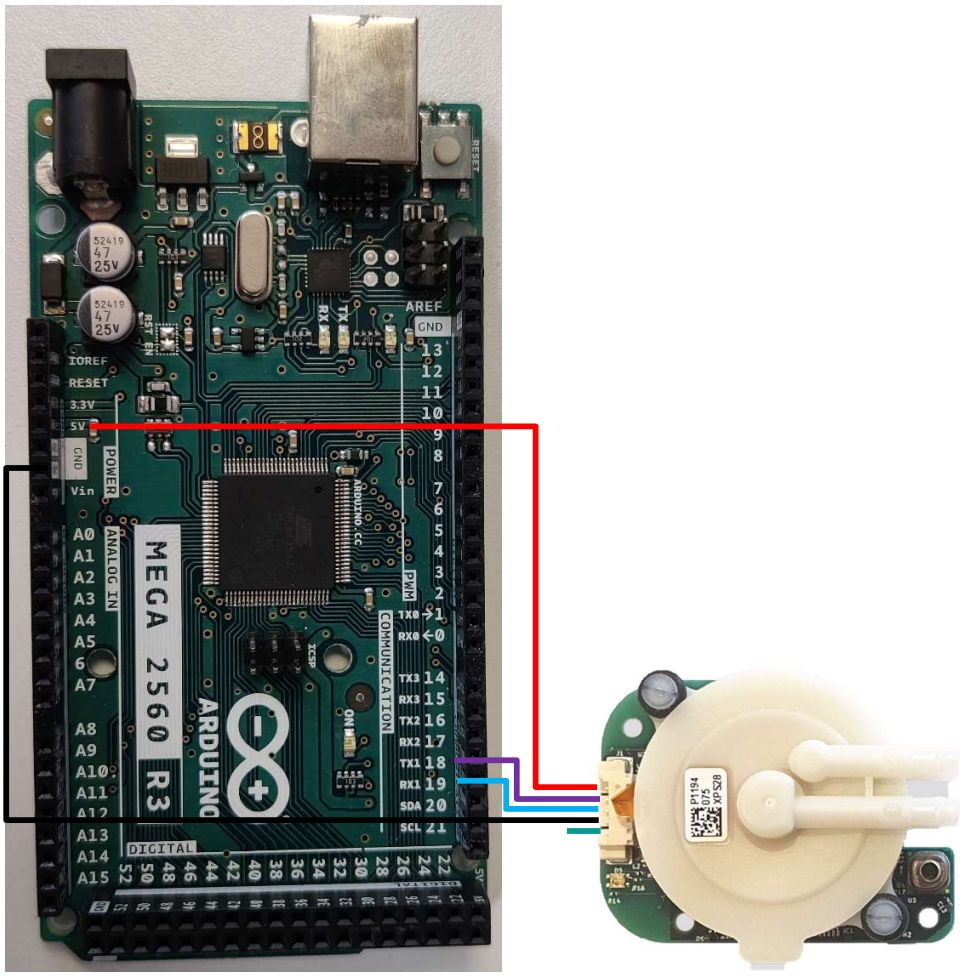
The Disc Pump Driver PCB Voltage must not exceed $48V_{r.m.s.}$ (where for a typical square-wave drive $V_{r.m.s.} \approx V_{pk}$) at frequencies between 20 and 22 kHz. It is the user's responsibility to ensure that the Disc Pump Driver PCB is used and/or integrated within any product in a safe manner. Read the appropriate user manual prior to first operation and take note of all safety notices.

WARNING

Take care during use of the Disc Pump Drive PCB not to create short circuits between exposed conductive parts of the board. Short circuits may lead to malfunctioning and heating.

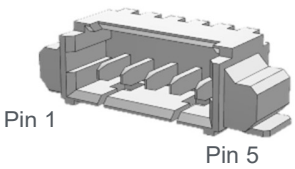
4. CONNECTING THE ARDUINO MEGA

4.1. SPM

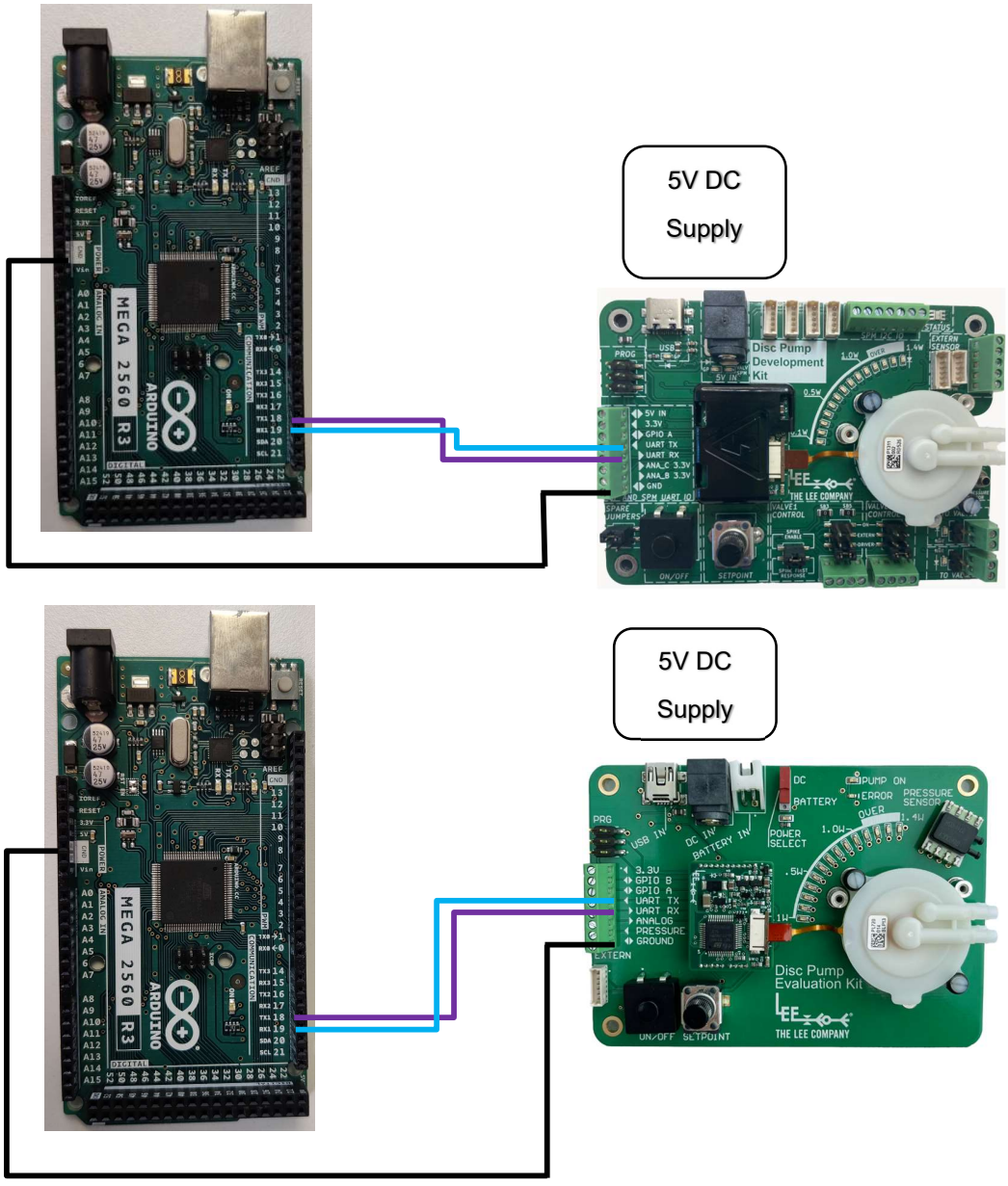


Connect the following:

- SPM pin 1 (VCC) – Arduino 5V pin
- SPM pin 2 (UART RX) – Arduino TX1 pin
- SPM pin 3 (UART TX) – Arduino RX1 pin
- SPM pin 4 (GND) – Arduino GND pin
- SPM pin 5 (Analog in) – Not connected



4.2. GP Drive Board on Development Kit or Evaluation Kit motherboard



Connect the following:

- Development/Evaluation kit UART RX – Arduino TX1 pin
- Development/Evaluation kit UART TX – Arduino RX1 pin
- SPM pin 4 (GND) – Arduino GND pin
- Connect an external power supply to the kit (e.g., through the DC in barrel plug).
- **Do not connect the USB connector to the Development Kit or Evaluation Kit** as it uses the same UART connection as the screw header and will interfere with the communication between the Arduino and the driver board.

5. FULL LIST OF COMMANDS

For an up-to-date list of commands, UART message structure and UART clock speed, please consult "*TG003: PCB Serial Communications Guide*". The guide also includes which commands need to be *int* and which ones should be *float*.

6. EXAMPLE SKETCH FOR CONTROLLING POWER AND PRESSURE

The most up-to-date version of the following code example is available on the Lee Ventus GitHub repository (https://github.com/The-Lee-Company).

The functions for sending and receiving UART commands and the functions for setting up manual power control and PID pressure control can be found in:

```
#include "lee_ventus_uart.h"
```

The macros for register IDs can be found in:

```
#include "lee_ventus_register.h"
```

This simple program is not configured to work with streaming mode. It will be able to send write commands, however it will not be able read values from the board as the streaming mode output will interfere with the simple logic of the read command.

6.1. Sending and receiving UART commands

6.1.1. Sending integer commands

```
//command to write an integer value to the board
//returns if the write was successful or not
bool uart_write_int16(int register_id, int16_t value_to_write)
{
  String command = String("#W");
  command = command + register_id;
  command = command + ",";
  command = command + value_to_write;
  command = command + "\n";

  Serial1.print(command);

  return true;
}
```

6.1.2. Sending float command

```
//command to write a float value to the board
//returns if the write was successful or not
bool uart_write_float(int register_id, float value_to_write)
{
  String command = String("#W");
  command = command + register_id;
  command = command + ",";
  command = command + value_to_write;
  command = command + "\n";

  Serial1.print(command);

  return true;
}
```

6.1.3. Receiving integer command

```
//command to read a int value from the board
int16_t uart_read_int16(int register_id)
{
  String command = String("#R");
  command = command + register_id;
  command = command + "\n";

  char rc;
  static bool recvInProgress = false; // variable for sensing if we are in the
data region of the received command
  static byte ndx = 0;                // variable for indexing the buffer

  while (Serial1.available() > 0) {       //clear buffer
    rc = Serial1.read();
  }

  // request data
  Serial1.print(command);
  delay(10);

  //read serial character by character and store those between "," and "\n"
  while (Serial1.available() > 0) {
    rc = Serial1.read();            // rc stores every character of the received
command in sequence
    if (recvInProgress == true) {   // this if triggers if we are in the data
region between "," and "\n"
      if (rc != '\n') {             // if we have not reached the end of the data
region which is the "\n"
        receivedChars[ndx] = rc;    // store the data
        ndx++;                      // and increment the counter for which digit
we are receiving
        if (ndx >= numChars) {      // make sure we do not exceed the limit of
the buffer
          ndx = numChars - 1;
        }
      }
      else {                        // this else triggers when we are past the
data region, so at the "\n"
        receivedChars[ndx] = '\0';  // terminate the string. Required to properly
handle the string
        recvInProgress = false;     // reset the state variables
```

```
        ndx = 0;
      }
    }


    else if (rc == ',') {           // this if senses when we reach the ","
      recvInProgress = true;        // and triggers that we are in the data
region
    }
  }

  //convert the string value into int
  String value_txt = receivedChars;
  int16_t value = (int16_t)(value_txt.toInt());
  return value;
}
```

6.1.4. Receiving float command

```
//command to read an float value from the board
float uart_read_float(int register_id)
{
  String command = String("#R");
  command = command + register_id;
  command = command + "\n";

  char rc;
  static bool recvInProgress = false; // variable for sensing if we are in the
data region of the received command
  static byte ndx = 0;                // variable for indexing the buffer

  while (Serial1.available() > 0) {       //clear buffer
    rc = Serial1.read();
  }

  // request data
  Serial1.print(command);
  delay(10);

  //read serial character by character and store those between "," and "\n"
  while (Serial1.available() > 0) {
    rc = Serial1.read();            // rc stores every character of the received
command in sequence
    if (recvInProgress == true) {   // this if triggers if we are in the data
region between "," and "\n"
      if (rc != '\n') {             // if we have not reached the end of the data
region which is the "\n"
        receivedChars[ndx] = rc;    // store the data
        ndx++;                      // and increment the counter for which digit
we are receiving
        if (ndx >= numChars) {      // make sure we do not exceed the limit of
the buffer
          ndx = numChars - 1;
        }
      }
      else {                        // this else triggers when we are past the
data region, so at the "\n"
        receivedChars[ndx] = '\0';  // terminate the string. Required to properly
handle the string
        recvInProgress = false;     // reset the state variables
```

```
        ndx = 0;
      }
    }


    else if (rc == ',') {           // this if senses when we reach the ","
      recvInProgress = true;        // and triggers that we are in the data
region
    }
  }

  //convert the string value into float
  String value_txt = receivedChars;
  float value = value_txt.toFloat();
  return value;
}
```

6.2. Manual power control example

```
    uart_write_float(REGISTER_SET_VAL, target_power);
    delay(100);

    Serial.print("\nTarget power: ");
    Serial.print(target_power);
    Serial.print("mW");

    //read the power level and pressure
    for(int j=0; j<4; j++)
    {
      float measured_power, measured_pressure;
      measured_power = uart_read_float(REGISTER_MEAS_DRIVE_MILLIWATTS);
      // the GP (on Evaluation kit) and SPM boards use different channels for
their pressure sensors
      if(device_type == DEVICE_TYPE_SPM) measured_pressure =
uart_read_float(REGISTER_MEAS_DIGITAL_PRESSURE);
      if(device_type == DEVICE_TYPE_GP) measured_pressure =
uart_read_float(REGISTER_MEAS_ANA_2);

      Serial.print("\nPower: ");
      Serial.print(measured_power);
      Serial.print("mW, ");
      Serial.print("Pressure: ");
      Serial.print(measured_pressure);
      Serial.print("mbar");

      delay(500);
    }
  }

  //disable pump
  uart_write_int16(REGISTER_PUMP_ENABLE, 0);
}
```

The Arduino IDE Serial Monitor output of the program should look like:

```
MANUAL POWER CONTROL
Target power: 100mW
Power: 97.01mW, Pressure: 243.10mbar
Power: 99.69mW, Pressure: 214.33mbar
Power: 99.93mW, Pressure: 186.35mbar
Power: 100.07mW, Pressure: 161.60mbar
Target power: 200mW
Power: 199.99mW, Pressure: 136.94mbar
Power: 199.91mW, Pressure: 130.17mbar
Power: 199.93mW, Pressure: 128.51mbar
Power: 199.57mW, Pressure: 128.26mbar
```

6.3. PID pressure control example

There is a slight difference between the GP and SPM driver boards. The GP diver on the Evaluation kit uses the Analog 2 channel for detecting pressure, while the SPM uses a digital pressure sensor. Therefore, the program checks the type of drive board to determine where to read pressure from.

```
#include <Arduino.h>
#include "lee_ventus_uart.h"

int device_type = 0;

// example function to set the SPM settings to do PID pressure control
// use register REGISTER_PUMP_ENABLE to enable the pump
// use register REGISTER_SET_VAL to set the target pressure
void uart_spm_setup_PID_pressure_control()
{
  uart_write_int16(REGISTER_PUMP_ENABLE        ,0);            //pump off
  uart_write_int16(REGISTER_STREAM_MODE_ENABLE ,0);            //streaming off
  uart_write_int16(REGISTER_CONTROL_MODE       ,MODE_PID);     //PID mode
  uart_write_int16(REGISTER_PID_MODE_SETPOINT_SOURCE     ,SOURCE_SETVAL);//PID
setpoint source (0=Set val (register 23), 1=Analog A,  2=Analog B, 3= Analog C
  uart_write_int16(REGISTER_PID_MODE_MEAS_SOURCE         ,SOURCE_DIGITAL_PRESSURE
);//PID measurement source (0=Set val (register 23), 1=Analog A  2=Analog B, 3=
Analog C, 4=Flow sensor, 5= digital pressure sensor
  uart_write_float(REGISTER_PID_PROPORTIONAL_COEFF,5);            //PID  'P=5'
  uart_write_float(REGISTER_PID_INTEGRAL_COEFF   ,10);           //PID  'I=10'
  uart_write_float(REGISTER_PID_DIFFERENTIAL_COEFF  ,0);            //PID  'D=0'
Advisable to keep this at zero
  uart_write_float(REGISTER_SET_VAL                        ,0);           //
register 23 (PID target). Set to zero initially
}

// example function to set the GP board settings to do PID pressure control
// use register REGISTER_PUMP_ENABLE to enable the pump
// use register REGISTER_SET_VAL to set the target pressure
void uart_gp_setup_PID_pressure_control()
{
  uart_write_int16(REGISTER_PUMP_ENABLE        ,0);            //pump off
  uart_write_int16(REGISTER_STREAM_MODE_ENABLE ,0);            //streaming off
  uart_write_int16(REGISTER_CONTROL_MODE       ,MODE_PID);     //PID mode
```

```
  uart_write_int16(REGISTER_PID_MODE_SETPOINT_SOURCE     ,SOURCE_SETVAL);//PID
setpoint source (0=Set val (register 23), 1=Analog A,  2=Analog B, 3= Analog C
  uart_write_int16(REGISTER_PID_MODE_MEAS_SOURCE         ,SOURCE_ANA2);  //PID
measurement source (0=Set val (register 23), 1=Analog A  2=Analog B (pressure
sensor on the eval kit), 3= Analog C, 4=Flow sensor, 5= digital pressure sensor
  uart_write_float(REGISTER_PID_PROPORTIONAL_COEFF,5);            //PID  'P=5'
  uart_write_float(REGISTER_PID_INTEGRAL_COEFF   ,10);           //PID  'I=10'
  uart_write_float(REGISTER_PID_DIFFERENTIAL_COEFF,0);            //PID  'D=0'
Advisable to keep this at zero
  uart_write_float(REGISTER_SET_VAL               ,0);            // register 23
(PID target). Set to zero initially
}


void setup()
{
  //Connect two Serial ports
  Serial.begin(115200);       // initialize UART connected to RX0 TX0 (connected
to inbuilt USB)
  Serial1.begin(115200);       // initialize UART connected to RX1 TX1 (connected
to Pump Drive Board)

  delay(100);
  // the GP (on Evaluation kit) and SPM boards use different channels for their
pressure sensors
  device_type = uart_read_int16(REGISTER_DEVICE_TYPE);
}


void loop()
{
// ------------------------------------------------------------------------
// PID pressure control example
// ------------------------------------------------------------------------
  Serial.print("\n\nPID PRESSURE CONTROL");
  Serial.print("\nWaiting for pressure in the system to die down...");
  delay(5000); // wait for the pressure in the system to die down


  //set to PID pressure control
  // the GP (on Evaluation kit) and SPM boards use different channels for their
pressure sensors
  if(device_type == DEVICE_TYPE_SPM) uart_spm_setup_PID_pressure_control();
```

```
  if(device_type == DEVICE_TYPE_GP) uart_gp_setup_PID_pressure_control();



  //and enable pump
  uart_write_int16(REGISTER_PUMP_ENABLE, 1);


  //cycle through different pressure levels
  for (int target_pressure = 100; target_pressure <= 250; target_pressure += 50)
  {
    uart_write_float(REGISTER_SET_VAL, target_pressure);
    delay(100);

    Serial.print("\nTarget pressure: ");
    Serial.print(target_pressure);
    Serial.print("mbar");

    //read the power level and pressure
    for(int j=0; j<10; j++)
    {
      float measured_power, measured_pressure;
      measured_power = uart_read_float(REGISTER_MEAS_DRIVE_MILLIWATTS);
      // the GP (on Evaluation kit) and SPM boards use different channels for
their pressure sensors
      if(device_type == DEVICE_TYPE_SPM) measured_pressure =
uart_read_float(REGISTER_MEAS_DIGITAL_PRESSURE);
      if(device_type == DEVICE_TYPE_GP) measured_pressure =
uart_read_float(REGISTER_MEAS_ANA_2);

      Serial.print("\nPower: ");
      Serial.print(measured_power);
      Serial.print("mW, ");
      Serial.print("Pressure: ");
      Serial.print(measured_pressure);
      Serial.print("mbar");

      delay(500);
    }
  }

  //disable pump
  uart_write_int16(REGISTER_PUMP_ENABLE, 0);
```

}

The Arduino IDE Serial Monitor output of the program should look like:

```
Target pressure: 150mbar
Power: 376.14mW, Pressure: 130.00mbar
Power: 391.68mW, Pressure: 138.22mbar
Power: 398.21mW, Pressure: 145.65mbar
Power: 406.37mW, Pressure: 147.78mbar
Power: 409.93mW, Pressure: 149.06mbar
Power: 416.76mW, Pressure: 148.86mbar
Power: 422.98mW, Pressure: 149.22mbar
Power: 418.92mW, Pressure: 149.35mbar
Power: 416.66mW, Pressure: 150.24mbar
Power: 414.47mW, Pressure: 150.87mbar
```

7. ADDITIONAL SUPPORT

The Lee Company Website (https://www.theleeco.com/disc-pumps/) provides advice on:

- Getting Started
- Applications
- Development Process
- Downloads (including datasheets, manuals, application notes, case studies and 3D models)
- Frequently Asked Questions

The Lee Company is happy to discuss next steps beyond prototyping, including system design. If you would like to discuss this with us, or for any other additional support, please contact your Lee Sales Engineer.

8. REVISION HISTORY

Date	Version	Change
06/06/24	R240604	Add Development Kit
03/08/2023	R030823	Rebranded.
04/03/2023	R040323	Document created.