Micromega Corporation

Using uM-FPU64 with Arduino

Introduction

This document describes how to use the uM-FPU64 floating point coprocessor (FPU) with the Arduino open-source physical computing platform. The support routines are compatible with both Arduino release 1.0 release and release 0022. For a full description of the uM-FPU64 chip, please refer to the *uM-FPU64 Datasheet* and *uM-FPU64 Instruction Reference*. Application notes are also available on the Micromega website.

Installing uM-FPU64 Support Software for Arduino

The support software for using the uM-FPU64 chip with Arduino is contained in a file called *uM-FPU64-Arduino.zip*. Download and unzip the file. It contains two folders, *Fpu64* and *FpuSerial64*, which are Arduino libraries for using the uM-FPU64 chip. Move these folders to the user *libraries* subdirectory used by *Arduino*.
e.g.

Windows: My Documents\Arduino\libraries
Macintosh: ~/Documents\Arduino\libraries

The *Fpu64* library provides support functions for using the uM-FPU64 chip with Arduino. The *FpuSerial64* library provides functions for sending FPU strings to the Arduino serial monitor. A descriptions of each library functions is provided later in this document.

Various examples are also provided with the *Fpu* library. The examples can be opened from the Arduino software using the *File>Sketchbook>Examples>Library-Fpu* command.

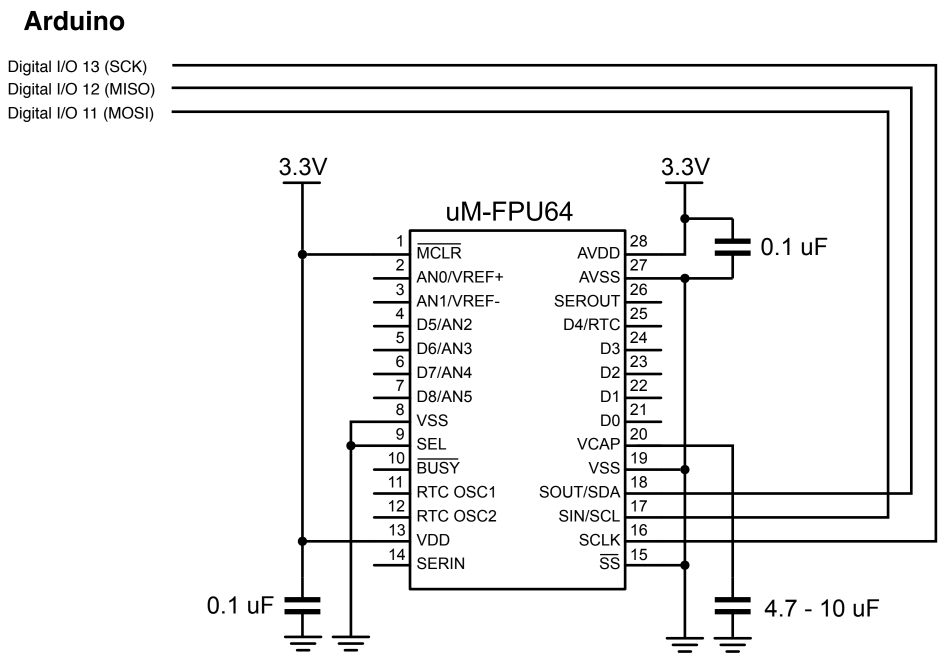
Connecting the uM-FPU64 chip to Arduino

The uM-FPU64 needs to be powered at 3.3V, but the interface pins used for the SPI connection are 5V tolerant. The uM-FPU64 is connected to Arduino using the hardware SPI interface. The pins used on Arduino to interface with the FPU are as follows:

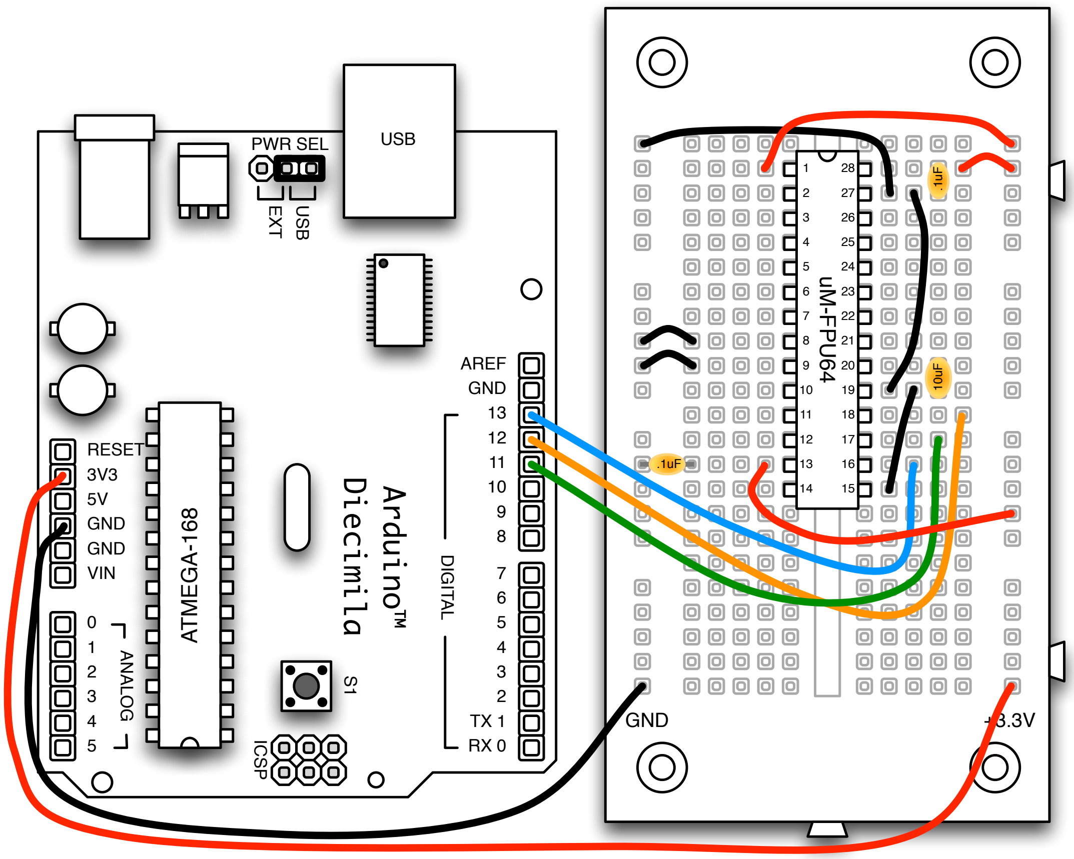
```
pin 13   SCK      SPI clock (connects to FPU SCLK pin)
pin 12   MISO     SPI master in, slave out (connects to FPU SOUT pin)
pin 11   MOSI     SPI master out, slave in (connects to FPU SIN pin)
pin 10   SS       SPI slave select
```

Initially, the FPU will be connected to Arduino as the only SPI device. The FPU /SS pin will be tied to ground, and the SPI slave select pin will not be connected. If multiple devices are connected to the SPI interface, the FPU needs to be configured for chip select. This is discussed later in the document. For now, only the FPU will be connected to the SPI interface. The following diagram shows the connections.

Connecting the uM-FPU64 as a single device on the SPI bus
Schematic Diagram



Breadboard Example using Arduino Diecimila



Example Programs

Several example programs are provided with the FPU library. After connecting the FPU as shown above, run the Arduino software and load the *sample* program with the following menu command:

File> Examples> Fpu64> Examples> sample

The *sample* program takes the diameter of a circle in centimeters, converts it to inches, and calculates the circumference and area. Compile and upload the program to the Arduino. Open the Serial Monitor and set the baud rate to 9600 baud. The following output should be displayed on the Serial Monitor:

```
Sample
uM-FPU64 r404

Diameter (cm):       25
Diameter (in.):      9.84252
Circumference (in.): 30.92119
Area (sq.in.):       76.0856

Done.
```

If you would like to try another example, load the graph program with the following menu command:

File> Examples> Fpu64> Examples> graph

The *graph* program displays a simple graph of the sine and cosine function on the Serial Monitor. An example of the sine graph is shown below:

```
Graph
uM-FPU64 r404

Graph of sine
-1        0         1
---------------------
          *            0.00000
          |  *         0.30902
          |     *      0.58779
          |       *    0.80902
          |         *  0.95106
          |         *  1.00000
          |         *  0.95106
          |       *    0.80902
          |     *      0.58779
          |  *         0.30902
          *           -0.00000
       *  |           -0.30902
    *     |           -0.58779
  *       |           -0.80902
*         |           -0.95106
*         |           -1.00000
*         |           -0.95106
  *       |           -0.80902
    *     |           -0.58779
       *  |           -0.30902
          *            0.00000
```

Let's look at the code for the *sample* program, section by section, to understand how it works.

After the initial comments the *SPI*, *Fpu64* and *FpuSerial64* libraries are included as follows:

```
#include <SPI.h>
#include <Fpu64.h>
#include <FpuSerial64.h>
```

The uM-FPU64 has 128 32-bit registers and 128 64-bit registers that can be used for calculations. Meaningful symbol names are defined for the FPU registers used by the program.

```
//-------------------- uM-FPU register definitions -------------------------

#define DiameterIn      10                    // diameter in inches
#define Circumference   11                    // circumference
#define Area            12                    // area
#define Pi              13                    // constant pi
```

The setup() function initializes the devices. The serial port is initialized to 9600 baud, and the name of the program is displayed.

```
//-------------------- setup ------------------------------------------------

void setup()
{
  Serial.begin(9600);
  Serial.println("Sample");
```

The *SPI* and *Fpu* libraries are initialized using the default pin assignment.

```
  SPI.begin();
  Fpu.begin();
```

The FPU is reset. If successful, a sync character is returned, and the FPU version string is displayed on the Serial Monitor.

```
  if (Fpu.reset() == SYNC_CHAR)
  {
    FpuSerial.printVersion();
    Serial.println();
  }
```

If the reset fails, an error message is displayed and further execution is stopped.

```
  else
  {
    Serial.print("uM-FPU not detected");
    while(1) ;
  }
}
```

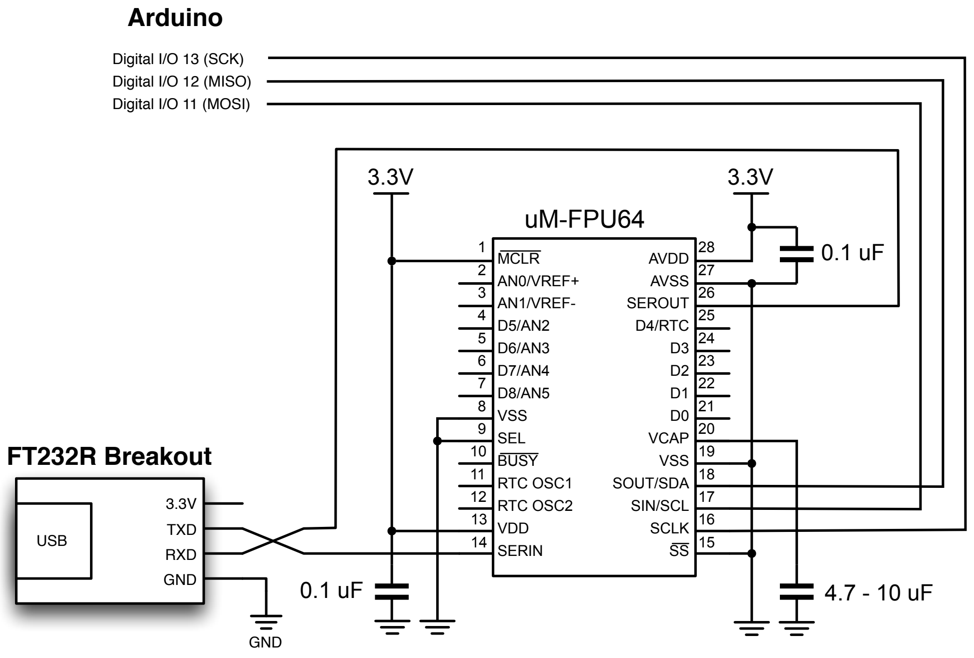
The sample calculations are performed in the loop() function. Instructions are data are sent to the FPU using the write function. Definitions for all FPU opcodes are included in the *fpu64.h* header file. Comments in the program explain each part of the calculations. The results are displayed on the Serial Monitor.

```
//-------------------- loop -------------------------------------------------

void loop()
{
  byte diameterCm;

  // Load constant for later use.
  Fpu.write(SELECTA, Pi, LOADPI, FSET0);
  // Get diameter in centimeters.  The value would typically come from a
  // sensor reading, but in this example an assumed value of 25 is used.
  diameterCm = 25;
  Serial.print("\r\nDiameter (cm):       ");
  Serial.println(diameterCm, DEC);

  // Convert inches to centimeters
  Fpu.write(SELECTA, DiameterIn, FSETI, diameterCm);
  Fpu.write(FCNV, 5);
  Serial.print("Diameter (in.):      ");
  FpuSerial.printFloat(0);
  Serial.println();

  // circumference = diameter * pi
  Fpu.write(SELECTA, Circumference, FSET, DiameterIn);
  Fpu.write(FMUL, Pi);
  Serial.print("Circumference (in.): ");
  FpuSerial.printFloat(0);
  Serial.println();

  // area = (diameter / 2)^2 * pi
  Fpu.write(SELECTA, Area, FSET, DiameterIn);
  Fpu.write(FDIVI, 2);
  Fpu.write(FMUL, Area, FMUL, Pi);
  Serial.print("Area (sq.in.):       ");
  FpuSerial.printFloat(0);
  Serial.println();

  Serial.println("\r\nDone.");
  while(1) ;
}
```

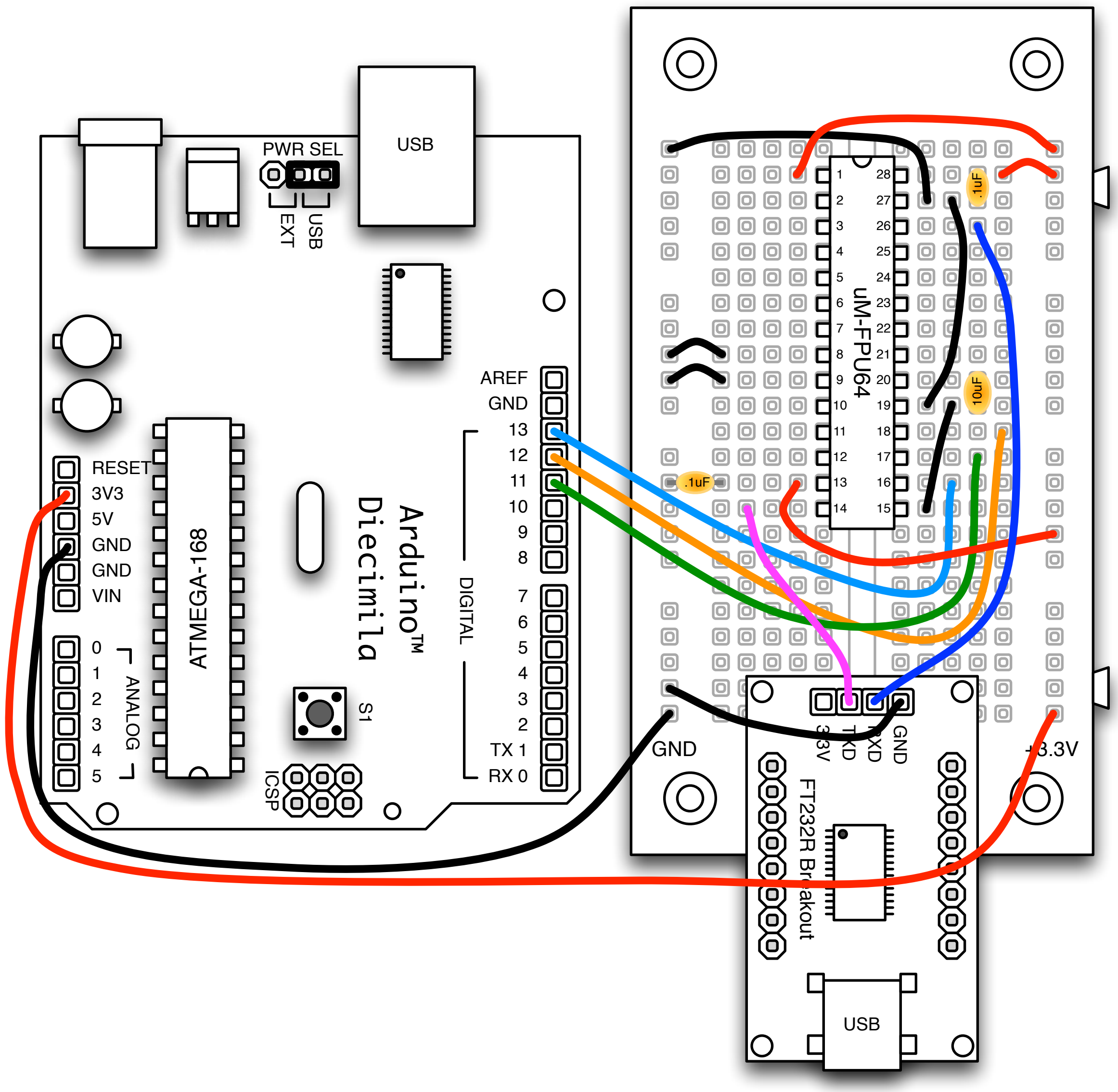
Enabling the uM-FPU64 Debug Monitor

The uM-FP64 chip has a built-in debug monitor that can be used to display the contents of uM-FPU registers, trace the execution of uM-FPU instructions, set breakpoints for debugging, and program user functions in Flash memory on the FPU. To enable the debug monitor, the `SERIN` and `SEROUT` pins on the FPU must be connected through a serial interface to a PC. The *uM-FPU64 IDE* software running on a PC communicates with the debug monitor and provides a graphical user interface for debugging and programming the FPU. The following example shows the connection using an FT232R breakout board from SparkFun Electronics.

Connecting the Debug Monitor - Schematic Diagram



Breadboard Example using Arduino Diecimila

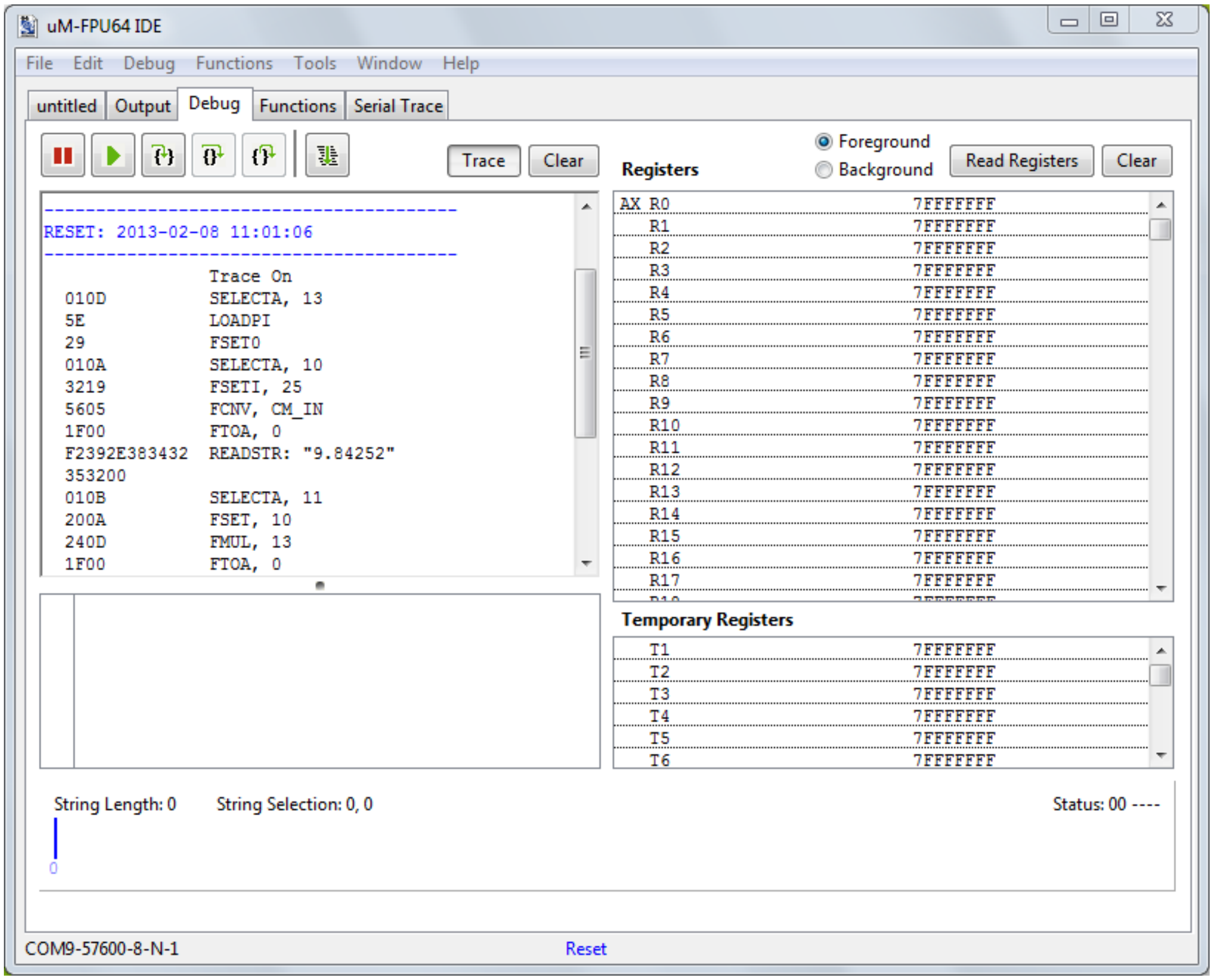


The debug trace can be turned on and off with the `TRACEON` and `TRACEOFF` instructions. To turn on the debug trace, the following instruction should be added to the end of the setup() function:

```
Fpu.write(TRACEON);
```

Alternatively, the *Trace on Reset* setting can be enabled with the *uM-FPU64 IDE* using the *Functions> Set Parameters...* menu command. A debug trace and the contents of the FPU registers are shown below for the *sample* program discussed above.

uM-FPU64 IDE - Debug Window



```
----------------------------------------
RESET: 2013-02-08 11:01:06
----------------------------------------
                Trace On
  010D          SELECTA, 13
  5E            LOADPI
  29            FSET0
  010A          SELECTA, 10
  3219          FSETI, 25
  5605          FCNV, CM_IN
  1F00          FTOA, 0
  F2392E383432  READSTR: "9.84252"
  353200
  010B          SELECTA, 11
  200A          FSET, 10
  240D          FMUL, 13
  1F00          FTOA, 0
```

```
AX R0    7FFFFFFF
   R1    7FFFFFFF
   R2    7FFFFFFF
   R3    7FFFFFFF
   R4    7FFFFFFF
   R5    7FFFFFFF
   R6    7FFFFFFF
   R7    7FFFFFFF
   R8    7FFFFFFF
   R9    7FFFFFFF
   R10   7FFFFFFF
   R11   7FFFFFFF
   R12   7FFFFFFF
   R13   7FFFFFFF
   R14   7FFFFFFF
   R15   7FFFFFFF
   R16   7FFFFFFF
   R17   7FFFFFFF
```

```
   T1    7FFFFFFF
   T2    7FFFFFFF
   T3    7FFFFFFF
   T4    7FFFFFFF
   T5    7FFFFFFF
   T6    7FFFFFFF
```

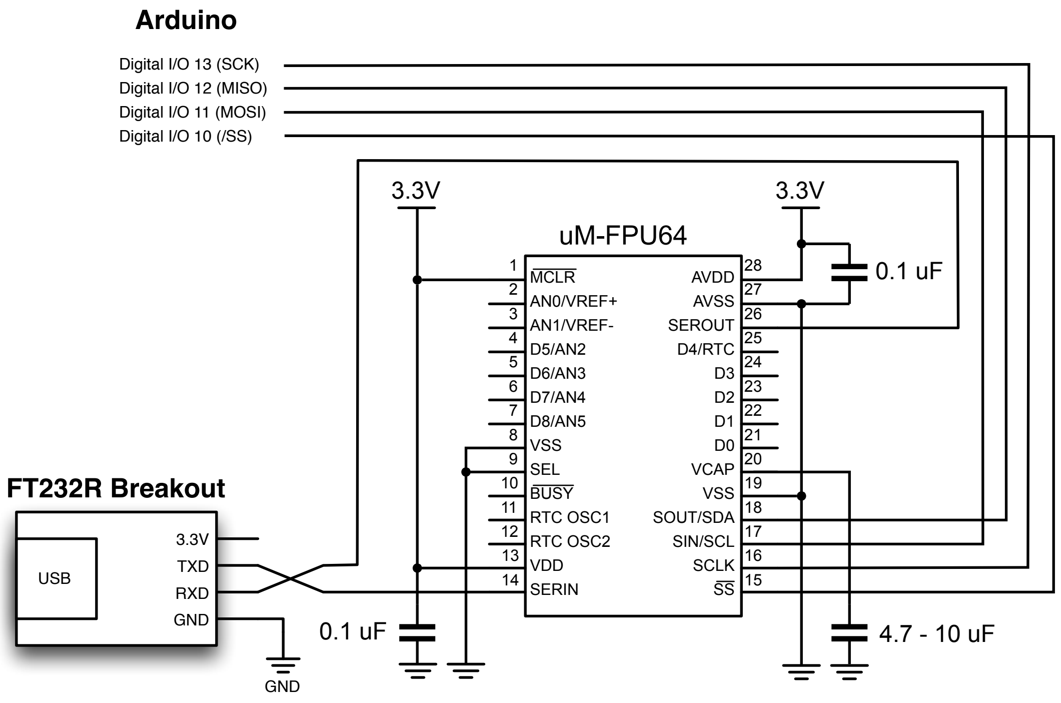
String Length: 0   String Selection: 0, 0   Status: 00 ----

COM9-57600-8-N-1   Reset

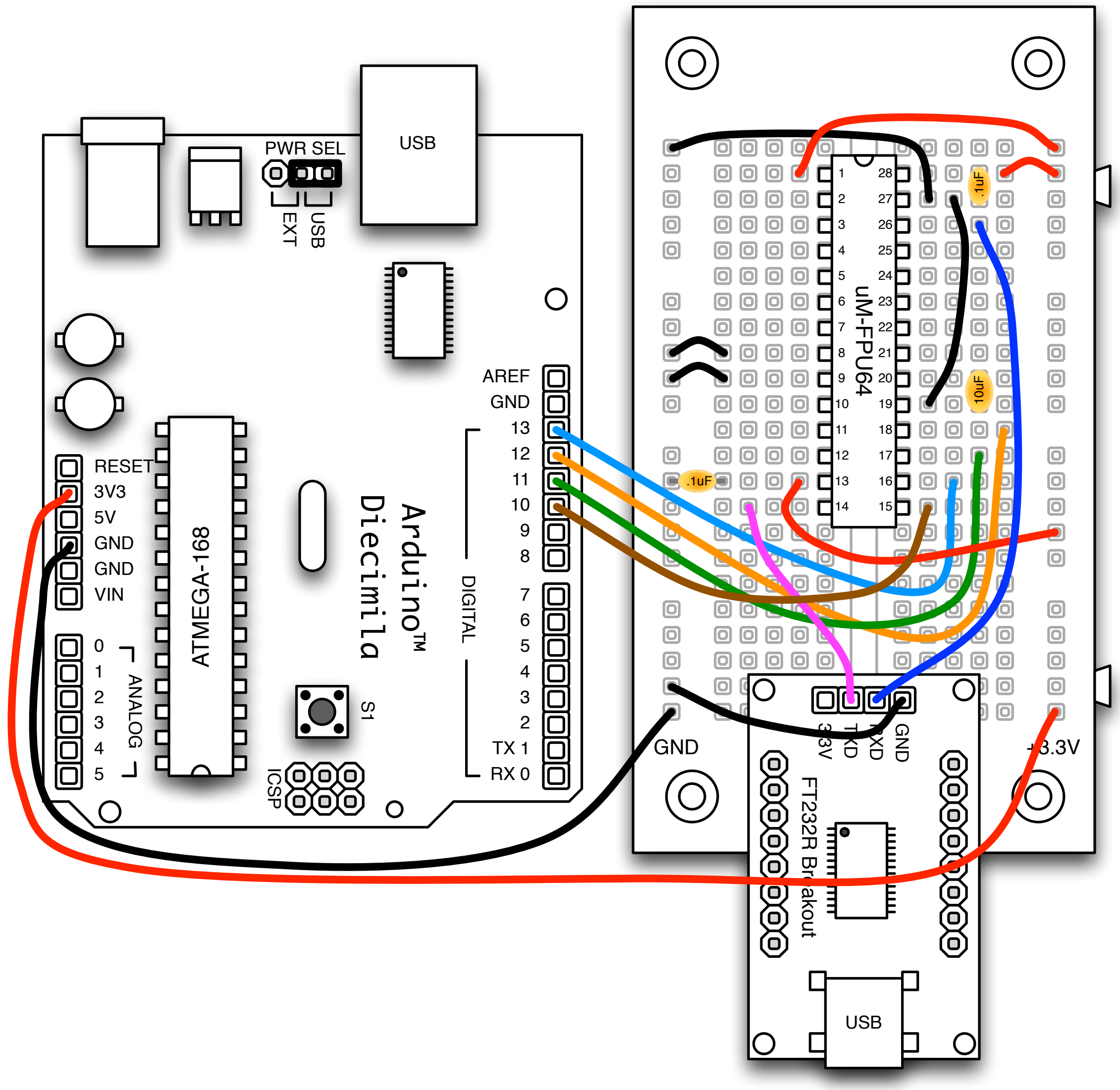
SPI bus with Multiple Devices

To use the FPU on an SPI bus with multiple devices, the /SS pin on the FPU must be used as a chip select.
By default, the *Fpu64* library uses Arduino pin 10 for the FPU chip select. This can be changed to an alternate pin using the `Fpu.begin(pin)` function. A schematic for connecting the FPU to an SPU bus is shown below:

**[Connecting the FPU on an SPI bus with Multiple Devices
Schematic Diagram**



Breadboard Example using Arduino Diecimila



Fpu64 Library Functions

The *Fpu64* library provides support functions for using the uM-FPU64 chip with Arduino.

begin()
begin(byte pin)

This function must be called in `setup()` to initialize the pins used to interface with the uM-FPU64 chip, configure the SPI, and reset the FPU chip. It must be called before any other *Fpu* library functions are called. The pins used on the Arduino to interface with the FPU chip are as follows:

```
pin 13   SCK      SPI clock (connects to FPU SCLK pin)
pin 12   MISO     SPI master in, slave out (connects to FPU SOUT pin)
pin 11   MOSI     SPI master out, slave in (connects to FPU SIN pin)
pin 10   SS       SPI slave select
```

The `pin` parameter specifies the pin to use as the FPU slave select (connected to FPU /SS). If the `pin` parameter is not specified, pin 10 is used by default. Even if a different pin is used, pin 10 is still set as a digital output, as required by the SPI hardware module.
e.g.

```
Fpu.begin();            // (default) use pin 10 for FPU chip select
Fpu.begin(8);           // (alternative example) use pin 8 for FPU chip select
```

reset()

Reset the FPU and configure the SPI hardware. This function is called by `begin()`, so a separate call is not required in `setup()`.

byte sync()

Confirms communication with the FPU. It returns `SYNC_CHAR` (0x5C) if successful.
e.g.

```
if (Fpu.sync() == SYNC_CHAR) Serial.println("FPU sync OK");
```

call(byte b1)

This function is used to call an FPU function. It is equivalent to `write(FCALL, b1)`, where `b1` is the function number.

wait()

The uM-FPU64 chip has a 256 byte instruction buffer which allows it to process in parallel with the Arduino. Before sending an instruction to read data from the FPU, the `wait()` function must be called to ensure that all instructions in the buffer have been completed, and the FPU is ready to send data.
e.g.

```
byte bval;
Fpu.wait();                 // wait until FPU is ready, then read byte
Fpu.write(LREADBYTE);       // read 8-bit integer from register A
dataByte = Fpu.readByte();
```

If a long series of calculations is done, `wait` must be called at least every 256 bytes to ensure that the instruction buffer doesn't overflow. A `wait` call is not required before using any of the *FpuPrint* functions, since they call the `wait` function themselves.

write(byte b1)
write(byte b1, byte b2)
write(byte b1, byte b2, byte b3)
write(byte b1, byte b2, byte b3, byte b4)

write(byte b1, byte b2, byte b3, byte b4, byte b5)
write(byte b1, byte b2, byte b3, byte b4, byte b5, byte b6)
write(byte b1, byte b2, byte b3, byte b4, byte b5, byte b6, byte b7)
write(byte b1, byte b2, byte b3, byte b4, byte b5, byte b6, byte b7, byte b8)

These functions are used to send instructions and data to the FPU. Each parameter specifies an 8-bit integer value to be sent to the FPU. Definitions for all of the FPU opcodes are included in the *Fpu64.h* header file.
e.g.
```
   Fpu.write(SELECTA, 1, LOADPI, FSET0);
```

writeWord(int wval)

Sends a 16-bit integer value to the FPU.
e.g.
```
   Fpu.write(SELECTA, 1, LOADWORD);
   Fpu.writeWord(1000);
   Fpu.write(FSET0);
```

writeLong(long lval)

Sends a 32-bit integer value to the FPU.
e.g.
```
   Fpu.write(LWRITE, 1);
   Fpu.writeLong(500000);
```

writeFloat(float fval)

Sends a 32-bit floating point value to the FPU.
e.g.
```
   Fpu.write(FWRITEA);
   Fpu.writeFloat(350.75);
```

writeString(char *s)

Sends a zero-terminated string to the FPU.
e.g.
```
   Fpu.write(SELECTA, 1, ATOF);
   Fpu.writeString("125.3335");
   Fpu.write(FSET0);
```

byte read()

Reads an 8-bit value from the FPU.
e.g.
```
   byte bval;
   Fpu.wait();             // wait until FPU is ready
   Fpu.write(LREADBYTE);   // read 8-bit integer from register A
   bval = Fpu.readByte();
```

int readWord()

Reads an 16-bit value from the FPU.
e.g.
```
   int wval;
   Fpu.wait();             // wait until FPU is ready
   Fpu.write(LREADWORD);   // read 16-bit integer from register A
   wval = Fpu.readWord();
```

long readLong()

Reads an 32-bit long integer value from the FPU.
e.g.
```
long lval;
Fpu.wait();             // wait until FPU is ready
Fpu.write(LREAD0);      // read 32-bit integer from register 0
lval = Fpu.readLong();
```

float readFloat()

Reads an 32-bit floating point value from the FPU.
e.g.
```
float fval;
Fpu.wait();             // wait until FPU is ready
Fpu.write(FREAD, 2);    // read 32-bit float from register 2
fval = Fpu.readFloat();
```

char *readString(char *s)
char *readString(char *s, byte opcode)

Reads a zero-terminated string from the FPU and stores it at the location specified by the `*s` parameter. A pointer to the string is returned. By default, the entire string buffer is read using the `READSTR` opcode. If the `opcode` parameter is `READSEL`, only the current string selection is read. Note: The following functions are called internally before the string is read:

`Fpu.wait()`
`Fpu.write(READSTR)` *or* `Fpu.write(READSEL)`
`Fpu.readDelay()`

Characters are read and stored in sequential bytes until a zero terminator is read. Care must be taken to ensure that the string buffer is large enough for the string being read.
e.g.
```
Fpu.wait();             // wait until FPU is ready
Fpu.write(FREAD, 2);    // read 32-bit float from register 2
longData = Fpu.readLong();
```

byte Fpu.readStatus()

Reads the status byte from the FPU. Note: The following functions are called internally before the status byte is read:
```
Fpu.wait()
Fpu.write(READSTATUS)
Fpu.readDelay()
```

Definitions for the bits in the status byte are defined in the *Fpu64.h* header file.
e.g.
```
Fpu.write(FSTATUSA);    // get the status of register A
if ((Fpu.readStatus() && STATUS_SIGN) == 1) Serial.println(“Negative value”);
```

readDelay()

After a read instruction is sent, and before the first data byte is read, a read setup delay is required to ensure that the FPU is ready to send data. This function provides the required read setup delay. Note: All functions in the *Fpu* library that read data call `readDelay()`, so user programs do not normally call this function directly.

FpuSerial64 Library Functions

The *FpuSerial* library provides functions for sending FPU strings to the serial port.

printVersion()

Sends the `VERSION` opcode, then calls `printString` to send the version string on the serial port.
e.g.

```
   FpuSerial.printVersion(VERSION); // display FPU version string
```

printFloat(char format)

Sends the `FTOA,format` opcode to convert the floating point value in register A to a formatted string. It then calls `printString` to send the string on the serial port. The `format` parameter is used to specify the desired format. If `format` is zero, the length of the displayed value is variable and can be from 3 to 12 characters in length. Up to eight significant digits will be displayed if required, and very large or very small numbers are displayed in exponential notation. The special cases of NaN (Not a Number), +Infinity, -Infinity, and -0.0 are handled. Examples of the display format are as follows:

```
      1.0           NaN           0.0
      1.5e20        Infinity      -0.0
      3.1415927     -Infinity     1.0
      -52.333334    -3.5e-5       0.01
```

If `format` is non-zero, the tens digit specifies the total number of characters to display and the ones digit specifies the number of digits after the decimal point. If the value in register A is too large for the format specified, then asterisks will be displayed. If the number of digits after the decimal points is zero, no decimal point will be displayed. Examples of the display format are as follows:

Value in A register	format	Display format
123.567	61 (6.1)	[123.6]
123.567	62 (6.2)	[123.57]
123.567	42 (4.2)	[**.*]
0.9999	20 (2.0)	[1]
0.9999	31 (3.1)	[1.0]

e.g.

```
   Fpu.write(SELECTA, 1);         // select register 1
   FpuSerial.printFloat(63);      // display float in 6.3 format
```

printLong(char format)

Sends the `LTOA,format` opcode to convert the 32-bit integer value in register A to a formatted string. It then calls `printString` to send the string on the serial port. The `format` parameter is used to specify the desired format. If `format` is zero, the length of the displayed value is variable and the displayed value can range from 1 to 11 characters in length. Examples of the display format are as follows:

```
      1
      500000
      -3598390
```

If `format` parameter is non-zero, it determines the display format. A value between 0 and 15 specifies the number of characters to display. The formatted string is right justified, with blank fill if `format` is positive, or zero fill if `format` is negative. If the value 100 is added to `format` the value is displayed as an unsigned long integer. If the value in register A is too large for the format specified, asterisks will be displayed. If the number of character to display is specified as zero, as many characters as required to display the number with be used. Examples of the display format are as follows:

Value in register A	format	Display format
`-1`	`10  (signed 10)`	`[        -1]`
`-1`	`110 (unsigned 10)`	`[4294967295]`
`-1`	`4   (signed 4)`	`[  -1]`
`-1`	`104 (unsigned 4)`	`[****]`
`0`	`4   (signed 4)`	`[   0]`
`0`	`0   (unformatted)`	`[0]`
`1000`	`6   (signed 6)`	`[  1000]`
`1000`	`-6  (signed 6)`	`[001000]`

e.g.
```
Fpu.write(SELECTA, 1);     // select register 1
FpuSerial.printLong(5);    // display signed 32-bit integer(5 character field)
```
or
```
Fpu.write(SELECTA, 1);     // select register 1
FpuSerial.printLong(100)   // display as unsigned 32-bit integer
```

printString()
printString(byte opcode)

Reads a zero-terminated string from the FPU and sends the string to the serial port. By default, the entire string buffer is read using the `READSTR` opcode. If the `opcode` parameter is `READSEL`, only the current string selection is read. The following functions are called by `printString` before the string is read:

`wait()`
`write(READSTR)` *or* `Fpu.write(READSEL)`
`readDelay()`

e.g.
```
FpuSerial.printString();          // display string buffer
```
or
```
FpuSerial.printString(READSEL);   // display current string selection
```

SPI Library Functions

The standard Arduino *SPI* library is used for transferring information using the Serial Peripheral Interface (SPI). Documentation on this library can be found on the Arduino website.

Further Information

The following documents are also available:

uM-FPU64 Datasheet	provides hardware details and specifications
uM-FPU64 Instruction Reference	provides detailed descriptions of each instruction
uM-FPU Application Notes	various application notes and examples

Check the Micromega website at www.micromegacorp.com for up-to-date information.