Julia

You can use the Julia programming language to control ASI hardware.

Here is an example script for the MS2000 and XY stage:

Last tested on Windows 10 64-Bit and Julia 1.8.0

```
using LibSerialPort

# First open the Julia REPL and enter:
# using Pkg
# Pkg.add("LibSerialPort")
# => This will install the LibSerialPort package

# get a list of serial ports
ports = get_port_list()
println("ports = $(ports)")

# setup serial port
port_name = "COM4"
baud_rate = 115200

function serial_send(serial_port::SerialPort, data::String,
report::Bool = true)
    write(serial_port, "$(data)\r\n")
    if report
        println("Send: $(data)")
    end
end

function serial_readline(serial_port::SerialPort, report::Bool =
true)::String
    line = readline(serial_port)
    if report
        println("Recv: $(line)")
    end
    return line
end

function serial_send_recv(serial_port::SerialPort, data::String,
wait_time_sec::Float64 = 0.001)
    write(serial_port, "$(data)\r\n")
    sleep(wait_time_sec) # 1 ms default
    return readline(serial_port)
end

function wait_for_device(serial_port::SerialPort,
wait_time_sec::Float64 = 0.001, report::Bool = true)
```

```
    if report
        print("Waiting for device...")
    end
    while is_device_busy(serial_port)
        sleep(wait_time_sec) # 1 ms default
    end
    if report
        println("Finished!")
    end
end

function wait_for_axis(serial_port::SerialPort, axis::String,
wait_time_sec::Float64 = 0.001, report::Bool = true)
    if report
        print("Waiting for axis $(axis)...")
    end
    while is_axis_busy(serial_port, axis)
        sleep(wait_time_sec) # 1 ms default
    end
    if report
        println("Finished!")
    end
end

function is_device_busy(serial_port::SerialPort)::Bool
    if serial_send_recv(serial_port, "/") == "B"
        true
    else
        false
    end
end

function is_axis_busy(serial_port::SerialPort, axis::String)::Bool
    if serial_send_recv(serial_port, "RS $(axis)") == "B"
        true
    else
        false
    end
end

function move(serial_port::SerialPort, x::Integer = 0, y::Integer = 0,
z::Integer = 0)
    serial_send(serial_port, "M X=$(x) Y=$(y) Z=$(z)")
    serial_readline(serial_port)
end

function relative_move(serial_port::SerialPort, x::Integer = 0,
y::Integer = 0, z::Integer = 0)
    serial_send(serial_port, "R X=$(x) Y=$(y) Z=$(z)")
    serial_readline(serial_port)
end
```

```
function set_max_speed(serial_port::SerialPort, axis::String,
speed::Float64)
    serial_send(serial_port, "S $(axis)=$(speed)")
    serial_readline(serial_port)
end

function main()
    # LibSerialPort.open defaults to =>
    # 8 data bits, no parity, one stop bit
    LibSerialPort.open(port_name, baud_rate) do ms2k
        # banner request
        serial_send(ms2k, "who")
        serial_readline(ms2k)

        # adjust speed
        set_max_speed(ms2k, "X", 2.5)
        set_max_speed(ms2k, "Y", 2.5)
        # move in a square pattern
        relative_move(ms2k, 10000, 0)
        wait_for_device(ms2k)
        relative_move(ms2k, 0, 10000)
        wait_for_device(ms2k)
        relative_move(ms2k, -10000, 0)
        wait_for_device(ms2k)
        relative_move(ms2k, 0, -10000)
        wait_for_device(ms2k)
    end
end

main()
```