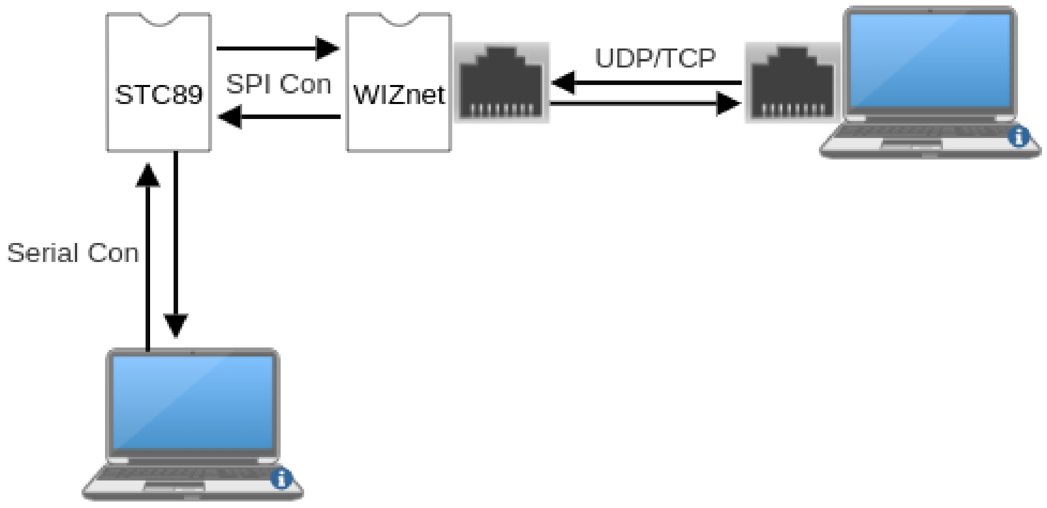


UDP/TCP RTU 8051 + W5500

Summary:
The goal of this project is the design of a simple UDP/TCP responder. An STC89/STC12 on a demo board will handle SPI transmissions between itself and a WIZNET LAN development board; it will also be connected via serial terminal to a laptop that will be used to configure parameters such as port and IP.

Required Components:
- WIZnet board (W5500)
- STC89/STC12 Demo Board or 8051 based chipset
- Wires/Transmission support equipment
- 2 LED diodes

Procedure:

Research and locate data sheets for all of the components above.

1. In your linux distro, install SDCC, and use python pip3 to install stcgal

- Here are 2 different examples of ways to write data to the MCU:
    - Using a make file: make && stcgal -p /dev/ttyUSB0 -P stc89 -a main.ihx
    - Using sdcc: sdcc stcboot.c i2c.rel lcd_1602.rel pcf.rel && stcgal -P stc89 stcboot.ihx
- Read STCgal Documentation

To get the chip into 6T Mode. Change stcgal options to include cpu_6t_enabled=true

    - Make sure to set the chip into 6T mode (6 Clock)

Part 1 done on 5/4
- When using sdcc use –model-small --model-small implemented

2. Set-up an SPI Protocol and Serial Communication
    - Set up Serial Communication Serial Communication Done
        - Read data from Serial interface
        - Display data to Serial interface
    - Set up basic SPI communication functionality in **C** SPI Communication works
        - Set up WIZnet communication using previously developed SPI
            - Read/Write to WIZnet
            - Set SOCKETS for WIZnet (UDP/TCP) Works
            - Set WIZnet registers for IP Add, MAC, Port, Subnet, and Gateway
        - Verify Ethernet Connectivity via Ping or similar

3. Set up WIZnet and Demo Board to receive UDP packets UDP Packets via ethernet done
    - Using the Ethernet functionality
        - Setup STC89 functionality to be able to read/write UDP packets
            - Be able to read from RX buffer
            - Be able to write to TX buffer Verified through rx_tx_test
            - Verify functionality using PacketSender or similar
    - Create Python Script to send/receive UDP packets Need to be tweaked
        - Protocol for Packets
            - Sending messages should follow the format of ":<#message>"
                - Where # is a RTU address 0-9
                - Where message is any characters a-z

              This is done
            - Receiving messages should follow the format of ":[#MESSAGE]"
                - Where # is the address of the RTU replying
                - Where MESSAGE is an echo of message but all uppercase
        - Configure STC89 to receive/return packets
            - Confirm if RTU has the specified address
            - Confirm that message is in correct format
            - Return message in proper format with address of RTU

4. Configure Serial Communication to change WIZnet Network configuration
    - '?' will bring up Config Menu Fixed
        - Set/Change RTU Address (0-9): USING RTU=
        - Set/Change IP Address: USING IP=
        - Set/Change Subnet Mask: USING SUB=
        - Set/Change Gateway: USING GATE=

    - Set/Change MAC Address: USING MAC=
- To change any of the above use the following format as an examples:
    - RTU=# (0-9)
    - IP=###.###.###.### -> IP=192.168.16.111
        - SUB= and GATE= follow the same format
    - MODE= UDP or MODE=TCP
    - MAC=0f0f0f0f0f0f

5. Replicate UDP functionality in TCP

- Add TCP functionality to STC89 Done
    - Keep all same formatting and functionality as UDP mode
- Add TCP functionality to Python Script
    - Let user change connection mode
    - Keep all same formatting and functionality as UDP mode

6. Combine UDP and TCP

- Create an implementation where UDP and TCP can be polled simultaneously form the python script. Polling together at the same time from script is not done yet

7.LEDS/Python Curses

- Add 3 LEDs to the STC89 Demo Board Done
    - TX LED, shows response activity
    - RX LED, shows receive activity
- Add Curses to your python script
    - For this the layout should show the following: Everything below this, is not done
        - Total number of sent packets There is something wrong with pycurses script
        - The received packet
        - Errors
        - Last message received
        - Current mode
        - Prompt same as before
- Also the PING command should now poll until the user terminates the command
    - Make this run as fast as possible
    - These polls will follow similar format as used before
        - The message should be random number of characters (a-z) between 8-16
    - Use a logic analyser and an output pin to measure the response time of the WIZnet
        - Set pin high when you get message and low after you sent the response

            - Document the response time
                - This should not be more than ~250 ms
- Add a Report function that will poll 100 times and generate a text file:
    - Max Non-Error Response Time
    - Min Non-Error Response Time
    - Number of errors
        - Log reason of error (i.e) timeout or incorrect response

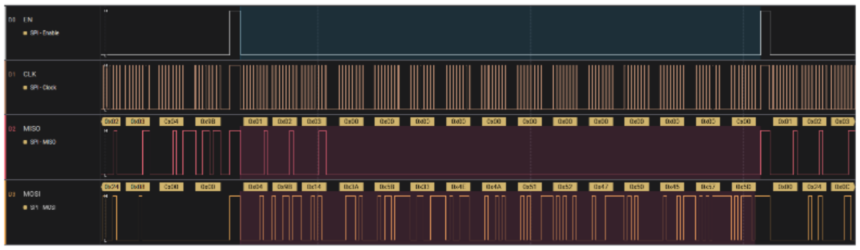
8. Interrupt Driven Polling
- Polling the STC 8051 should be done via W5500 interrupts. Wake up the board via interrupt before starting to listen for packets, to reduce the load on the chip.
- Read the Wiznet W5500 Datasheet to learn more about interrupts driven polling.

9. SPI Assembly Rewrite
- After completing and testing all points above, rewrite the SPI code into assembly for further optimization
    - After this test all components for correct functionality
    - Make this run as fast as possible with little to no delays
    - Use a logic analyser and an output pin to measure the response time of the WIZnet
        - Set pin high when you get message and low after you sent the response
        - Document the response time
            - This should not be more than ~150 ms
- Compare results of C version of SPI to ASM version of SPI

10. Documentation
- Document the speed comparisons of C version of SPI and ASM version of SPI and fingerprint the performance with logic analyzer and other tools.
- Include Graphs of Data Transfer Speed, and Network latency between both versions of SPI.

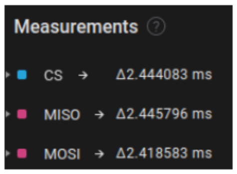
Example Graph:

Graph 1: SPI Data Transfer Speed in *C*
UDP



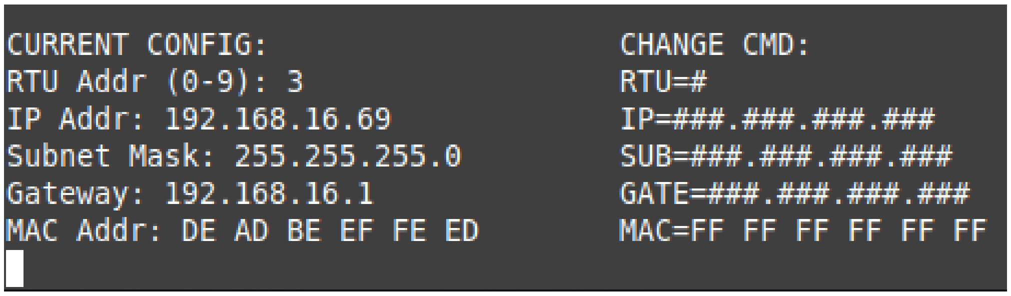
For MISO Signal:
ΔT: Time between data points ≈ 0.00143 seconds.
Edges: 8 falling and 8 rising.
Frequency Range: 2362.67 Hz to 8264.46 Hz.
Average Frequency: 5702.84 Hz.
Timing Variation: Standard deviation ≈ 0.00011 seconds.

For MOSI Signal:
ΔT: Time between data points ≈ 0.00143 seconds.
Edges: 8 falling and 8 rising.
Frequency Range: 2178.25 Hz to 31788.08 Hz.
Average Frequency: 5874.13 Hz.
Timing Variation: Standard deviation ≈ 0.00016 seconds.



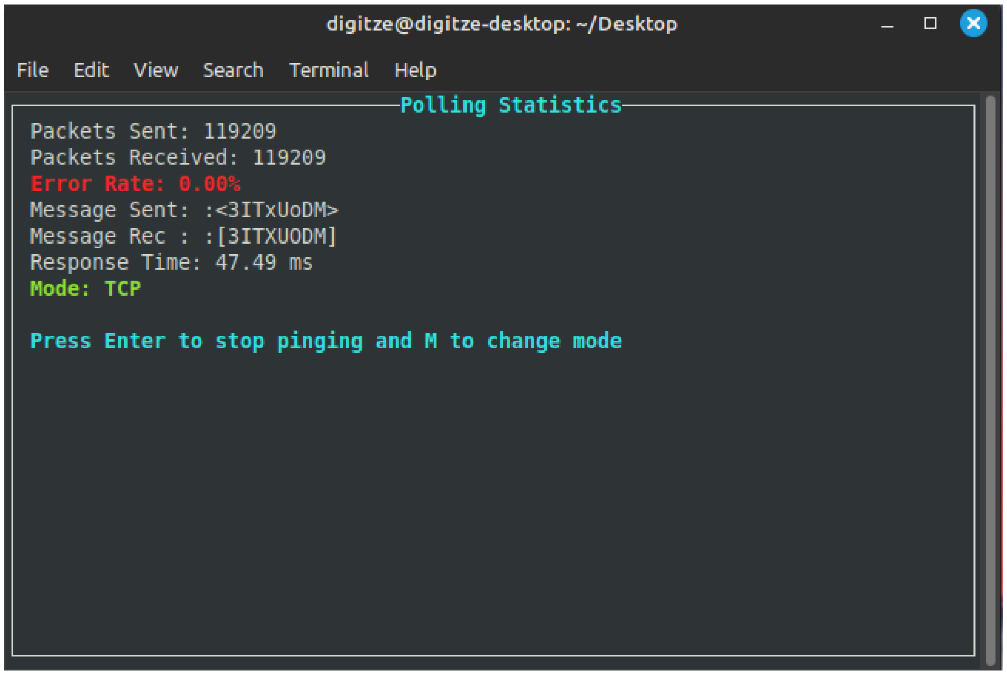
```
Measurements

CS   → Δ2.444083 ms
MISO → Δ2.445796 ms
MOSI → Δ2.418583 ms
```

Example of Serial Menu Lay



```
CURRENT CONFIG:                    CHANGE CMD:
RTU Addr (0-9): 3                  RTU=#
IP Addr: 192.168.16.69             IP=###.###.###.###
Subnet Mask: 255.255.255.0         SUB=###.###.###.###
Gateway: 192.168.16.1              GATE=###.###.###.###
MAC Addr: DE AD BE EF FE ED        MAC=FF FF FF FF FF FF
```

Example of Python Curses Menu:


digitze@digitze-desktop: ~/Desktop
File Edit View Search Terminal Help

```
Polling Statistics
Packets Sent: 119209
Packets Received: 119209
Error Rate: 0.00%
Message Sent: :<3ITxUoDM>
Message Rec : :[3ITXUODM]
Response Time: 47.49 ms
Mode: TCP

Press Enter to stop pinging and M to change mode
```