Pre-requisites: Must have skimmed through the [Digital FT Modbus Interface.pdf](https://drive.google.com/file/d/1Zgh-KinXeTZ4m7_ZIznalqFP7uqSq-Ya/view?usp=sharing) document and have sample code at hand to compare

To read and write from the ATI load cell using NETCANOEM, the Modbus interface is used.

The typical structure of the modbus packet that is sent looks like

AddressField (always 10 for digital f/t)--Function Code -- Data -- CRC high byte -- CRC low byte

This structure can be used to “read holding registers” or “write to single/multiple registers” as standard Modbus commands. The list of holding register addresses is given in Table 3 of the documentation.

**Example: Read Status Word**

To read the status word register, which is a single register, the modbus packet will look like this:

0a (which is 10 in hexadecimal)

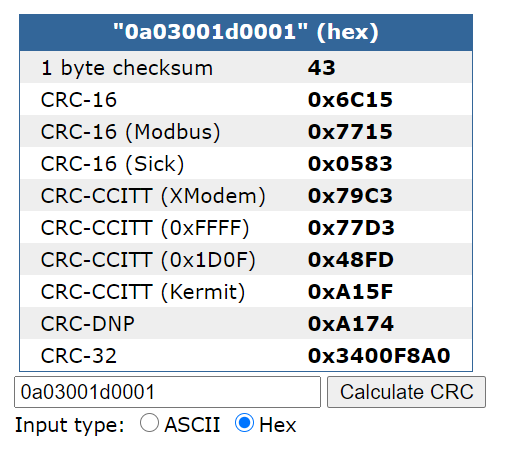
03 (which is the standard modbus read register command)

001d (which is the status word register)

0001 (which is the first address of the register)

1577 (which are the two CRC bytes calculated using this [crc calculating tool](https://www.lammertbies.nl/comm/info/crc-calculation) )

Notice that CRC-16 (Modbus) says 0x7715; in this protocol we reverse the two so that it is 0x1577 instead



When the digital F/T spits out bytes, they will be of the form “0a03001d (data) (crc bytes)”

A status of 0 means all is well.

To read multiple registers such as the calibration register, you will need an address range within the register. Some commands are in the “firsthello.py” file, but I can’t recall how they were calculated, such as, to read the calibration serial number, I used the command '0a0300e30004b484’ where I can recognize 00e3 as the calibration 1 register, and 0004 may be the location of the calibration ID further within the register.

**Start Streaming Data**

The custom commands specific to this device are for reading data and for changing gains etc. I have only used the streaming command for reading data, which is “**0a4655a39d**”

Let’s break it down:

0x0a - Slave address (10)

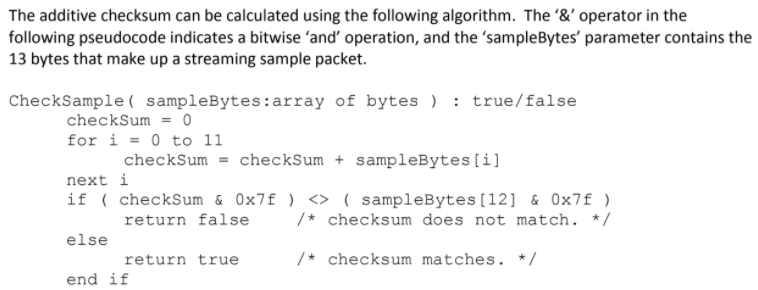
0x46 - Start streaming command (70)

0x55 - Just a data byte you always send with the streaming command

0x9d and 0xa3 - The two CRC bytes calculated using the online calculator tool

**Read Streaming Data**

To read the streaming data, read 13 bytes at a time, but you’ll have to see if the checksum is correct, and they have a specific algorithm for that. These 13 bytes on your lucky day can be your full packet or it can be two halves of two packets. The only way you can tell is the checksum and sometimes on your very unlucky day, two half-packets can also result in a correct checksum, and you get a garbage reading. I have not yet figured out how to deal with this.



This is encoded in python as

**def CheckSample(sampleBytes):**

**checkSum=0**

**for i in range (0,11):**

**checkSum = checkSum + sampleBytes[i]**

**if (checkSum & 127) <> (sampleBytes[12] & 127):**

**return False**

**else:**

**return True**

A straightforward code to take a reading would have been:

reading = loadcell.read(13) #read 13 bytes  
if(CheckSample(reading): print(reading.encode(‘hex’)

But I wanted to check everything in the reading, so the code looks like this:

reading=loadcell.read(13)

sample=bytearray(reading)

if CheckSample(sample):

for i in range(0,11,2):

print(binacii.hexlify(reading[i]))

print(' ')

print(reading[12])

print(binascii.unhexlify(reading))

print(' ')

print(time.time()-t1)

print('\n')

else:

continue

Note: binascii.hexlify(sample) and binascii.unhexlify(sample) does the same task as sample.encode(‘hex’) and sample.decode(‘hex’)

**Jam Stream:**

To jam the stream, just send 14 bytes, they could be garbage, and this has worked for me in the past: “0affefff570144007fef1f” where 0a is the slave address