Table of Contents

general info about the ISP handshake protocol

There are several Commands & Sequences of Commands used repetitively

1. “start” new command
2. “Hello” sequence
3. “Exit” sequence
4. “where” am I ?
5. goto DATA mem
6. goto CODE mem
7. “blank check” = read LOCK bit status
8. “set” LOCK bit, or remove.
9. “read” the CODE or DATA flash memory
10. “erase” selected mem.
11. “write” = program the CODE or DATA flash memory
12. “RUN” = exit Boot Mode and start the App.

"start" 0x3C sequence

PC ↔ MCU
0x3C → - *means “new command”*
← 0x3C - *confirmation return*
This sequence is used always before any command in the communication.

"Hello" sequence 0x55

PC ↔ MCU
0x3C →
← 0x3C - *“new command” + “confirmation” again*
0x55 → - *this is the “hello” command*
← 0x55 - *the “hello” confirmation answer*
Now the Handshaking can start.
This is critical now !
every 0x3C + more bytes will trigger some functions on the FLASH memory. !

"Exit" sequence 0xDC

PC ↔ MCU
0x3C →
← 0x3C - *this is the “new command” + answer sequence again*
0xDC → - *this is the “exit” command*
← 0xDC - *“exit” confirmation answer*
← 0x3E - *“end of answer”*
Now the connection can be closed & power off or reboot without the Boot_EN signal.
sending more bytes even 0x3c does not effect the commands here any more.
But sending 0x3C and 0x55 again open the Flash commands again.

"where" am I ? 0x01

PC ↔ MCU
0x3C →
← 0x3C
0x01 → - *send 0x01 = in what mem is the focus ?*
← 0x01 - *confirm*
← 0x55 or 0xAA - *0x55 = DATA; 0xAA = CODE mem*
← 0x3E - *END*
Now we can read out status of this mem or change focus

"goto DATA" sequence 0x81 + 0x55

PC ↔ MCU

0x81 → - *send 0x81*
← 0x81 - *confirm*
0x55 → - *send 0x55 = goto DATA mem*
← 0x55 - *confirm*
← 0x3E - *END*
This is used to address DATA memory now after here

"goto CODE" sequence 0x81 + 0xAA

PC ↔ MCU
0x81 → - *send 0x81*
← 0x81 - *confirm*
0xAA → - *send 0xAA = goto CODE mem*
← 0xAA - *confirm*
← 0x3E - *END*
This is used to address CODE memory now after here

read first Byte 0x06

Please select the DATA or CODE mem. with commend 0x81 first !

PC ↔ MCU
0x3C →
← 0x3C
0x06 → - *this is the "read first byte" command*
← 0x06 - *confirm*
…
← *here comes an Answer that can be a) or b)*
…

← 0x3E - *“end of answer”*

Answer:
a) with no LOCK bit: 0x55, 0x00, 0x00, 0x.. *where 0x.. is very first byte of DATA/CODE mem*
example: 3C,06,55,00,00,66,3E - *here 0x66 is first byte.*
b) if there is only 0xAA as answer the selected mem is locked.
example: 3C,06,AA,3E - here the DATA/CODE mem is locked. 😟

---

change "LOCK" bytes 0xEE

PC ↔ MCU
0x3C →
← 0x3C
0xEE → - *this is the “change LOCK” command*
← 0xEE - *confirm*
0x.. → - *00; 01; 10 or 11 - see table below*
← 0x.. - *confirm*
← 0x3E - *this is “end of answer” character*

00 = delete all Lock
01 = del. DATA Lock + set CODE Lock
10 = set DATA Lock + del. CODE Lock
11 = Lock both mem.

if any LOCK bit was set before, by del. a Lock the mem is also deleted, but unlocked.
If the mem was unlocked before, no changes were made = no problem. The mem is not deleted
This is critical !
If you delete any previously set LOCK bit, the mem is deleted too !

"read" mem. 0x03

make sure you selected the right mem. and it's not locked ! PC ↔ MCU
0x3c →
← 0x3c
0x03 → - *send read command*
← 0x03 - *confirm*
0x00 → - *send 0x00 , 3 times*
← 0x00 - *confirmed , 3 times*
0xFF → - *send 0xFF*
← 0xFF - *confirmed*
← 0x01 - *first byte*
← 0x02 - *second byte*
← .... - *as many bytes as the mem is size. 255 bytes for DATA, 64k for CODE mem.*
← 0x3E - *END*
example: 3C, 03, 00, 00, 00, FF, 01, 02, 03, .. , 3E

"erase" mem. 0x82

make sure you selected the right mem. with 0x81 and it's locked or unlocked as you like !

PC ↔ MCU
0x3c →
← 0x3c
0x82 → - *send erase command*
← 0x82 - *confirm*
0x.. → - *0x55 = DATA; 0xAA = CODE mem.*
← 0x.. - *confirmed*
← 0x3E - *END*

"write" mem. 0x84 & 0x85

make sure you selected the right mem. with 0x81 and it's locked or unlocked as you like ! Also erase the mem. with commend 0x82.

PC ↔ MCU
0x3C →
← 0x3C
0x84 → - *“write first byte” at address command*
← 0x84 - *confirm*
0x00 → - *send 0x00 , address high byte*
← 0x00 - *confirmed*
0x00 → - *send 0x00 , address low byte*
← 0x00 - *confirmed*
0x.. → - *send first data or code byte*
← 0x.. - *confirmed*
← 0x3E - *END*
0x3C →
← 0x3C
0x85 → - *“write next bytes” command*
← 0x85 - *confirm*
0x.. → - *send next byte*
← 0x.. - *confirmed*
← 0x3E - *END*
goto 0x85 again and again until you have no more bytes to write to the Flash mem.
you can stop anywhere, you can jump to other address by 0x84 command and write further on ...

"RUN" the App 0xFF

PC ↔ MCU
0x3c →
← 0x3c
0xFF → - *send “RUN” command*
← 0xFF - *confirm*
← 0x3E - *END*
The Application starts like you have removed Boot_EN and pressed Reset

---

"Open" connection to W7100A and read status + write DATA

First you need to set the Boot_EN signal pin to 3.3V = high logic level.
Reset the MCU by low active nReset signal pin.
Now the serial communication can start in 115200,8,N,1.
The handshaking to “open” the communication is more complex and combine some of the above commands:

```
0x3C,0x55               // open the communication, repeat until you receive the 0x55
                           here no 3C or 3E "roger" is send, no Echo here
0x3C, 0x01, 0x.., 0x3E // check if 0x.. is 55 or AA for the mem. you have access
0x3C, 0x06, 0xAA, 0x3E // here CODE mem is locked
0x3C, 0x81, 0x55, 0x3E // select DATA (0x55) mem. to access.
0x3C, 0x06, 0x55, 0x00, 0x00, 0x44, 0x3E
                        // the DATA mem. is unlocked and first byte is 0x44
```

Now you can read the DATA flash mem. with 0x03 command.
Or you can erase that mem. and write new data into it with 0x82 and 0x84 + 0x85 command.

```
0x3C, 0x81, 0x55, 0x3E // select DATA (0x55) mem. to access.
0x3C, 0xEE, 0x00, 0x3E // remove both LOCK bits
                           ! if CODE is locked, CODE mem is deleted here !
0x3C, 0x82, 0x55, 0x3E // delete DATA mem.
0x3C, 0x84, 0x00, 0x,00 0x.. // write first byte at address 00.00
0x3C, 0x85, 0x.., 0x3E // write next byte
                        // repeat write byte until you receive 0x3E, then mem. is at
the end
                        // or exit before END.
0x3C, 0x03, 0x00, 0x00, 0x00, 0xFF, 0x.., 0x.., 0x.., ...., 0x3E
                        // read from address 00.00 to 00.FF and receive bytes until
0x3E
0x3C, 0xFF, 0x3E        // exit and start the App, even with Boot_EN high.
or
0x3C, 0xDC, 0x3E        // exit communication mode. Now you need the 0x55 sequence
again.
```