iEthernet W5200 Datasheet

Version 1.21

http://www.wiznet.co.kr

W5200

W5200은 위즈네트의 Hardware TCP/IP 기술을 이용한 임베디드 시스템을 위한 인터넷 솔루션 중 SPI (Serial Peripheral Interface) 성능을 향상시킨 제품이다. SPI Interface를 채택하여 패키지 사이즈를 줄여 작은 임베디드 시스템의 구현이 보다 용이해 졌다. W5200 하나의 칩으로 TCP/IP 프로토콜 처리 및 10/100 Ethernet PHY와 MAC을 구현하여 개발하고자 하는 Application에 Internet Connectivity를 손쉽게 구현할 수 있도록 지원한다.

위즈네트에서는 TCP, UDP, IPv4, ICMP, IGMP, ARP, PPPoE 등의 통신 프로토콜을 Full hardware logic으로 개발하여 여러 제품에서 사용하고 있다. W5200에서는 데이터 통신을 위해서 data communication memory를 32Kbyte로 사용하였다. 이에 W5200에서는 하드웨어로 처리되는 8개의 독립적인 하드웨어 SOCKET을 사용할 수 있다.

SPI Interface 방식은 외부 MCU와 W5200의 연결을 간단하게 구현 할 수 있다. W5200의 SPI는 최대 80MHz의 High Speed 통신속도를 지원한다. 또한, 시스템의 소비전력을 낮추기 위해 WOL (Wake On LAN) 기능과 Power Down mode를 제공한다. WOL모드에서 외부에서 Wake-up시키기 위해 Raw Ethernet Packet 형식의 MACGIC Packet이 사용된다.

Features

- Support Hardwired TCP/IP Protocols : TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE
- Supports 8 independent sockets simultaneously
- Very small 48 Pin QFN Package
- Support Power down mode
- Support Wake on LAN
- Support High Speed Serial Peripheral Interface(SPI MODE 0, 3)
- Internal 32Kbytes Memory for Tx/Rx Buffers
- 10BaseT/100BaseTX Ethernet PHY embedded
- Support Auto Negotiation (Full and half duplex, 10 and 100-based)
- Support Auto MDI/MDIX
- Support ADSL connection (with support PPPoE Protocol with PAP/CHAP Authentication mode)
- Not support IP Fragmentation
- 3.3V operation with 5V I/O signal tolerance
- Lead-Free Package
- Multi-function LED outputs (Full/Half duplex, Link, Speed)

Target Applications

W5200은 다음과 같은 Embedded application에 적합하다.

- Home Network Devices: Set-Top Boxes, PVRs, Digital Media Adapters
- Serial-to-Ethernet: Access Controls, LED displays, Wireless AP relays, etc.
- Parallel-to-Ethernet: POS / Mini Printers, Copiers
- USB-to-Ethernet: Storage Devices, Network Printers
- GPIO-to-Ethernet: Home Network Sensors
- Security Systems: DVRs, Network Cameras, Kiosks
- Factory and Building Automations
- Medical Monitoring Equipments
- Embedded Servers

Block Diagram

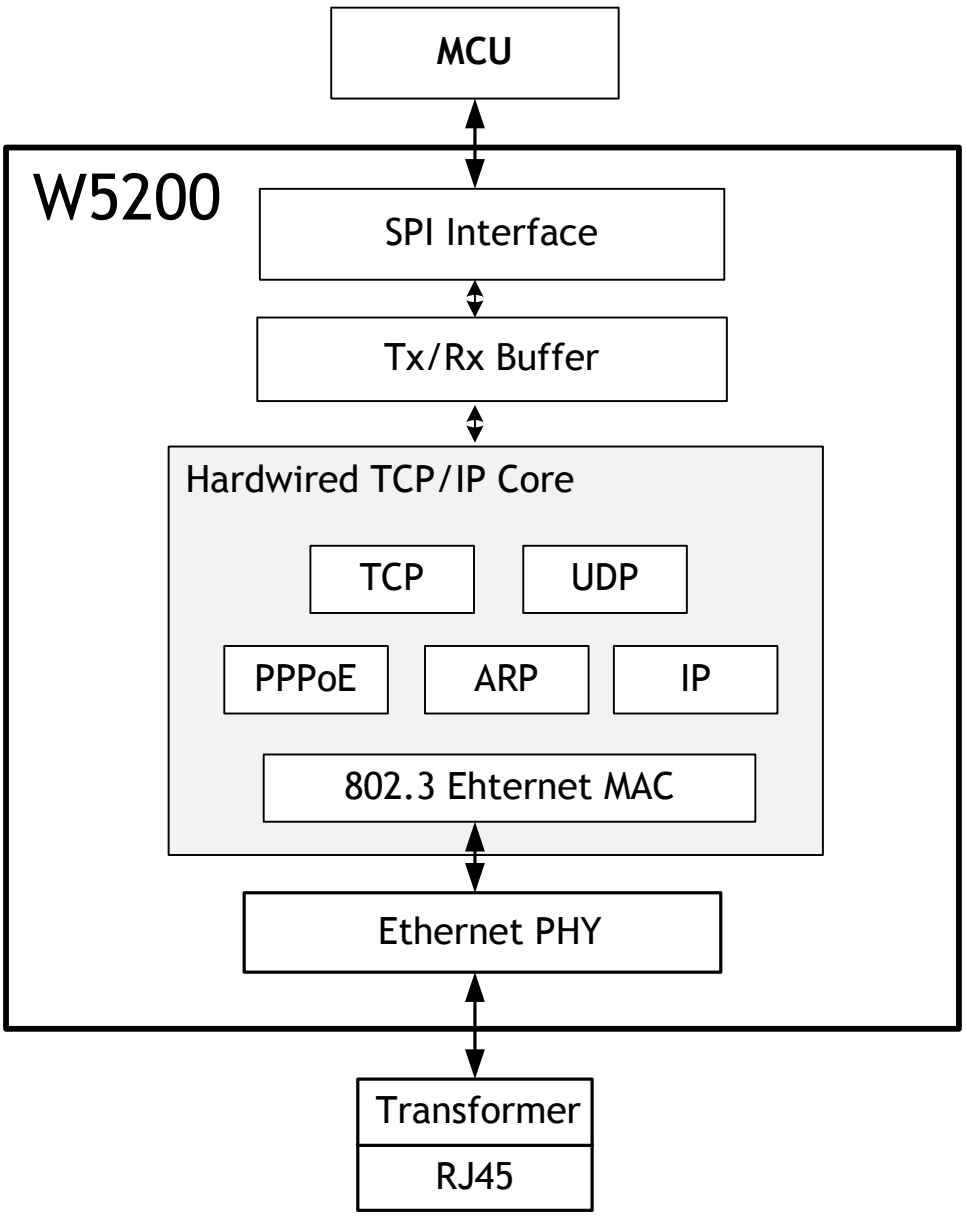


Table of Contents

Table of Figure

1 Pin Assignment

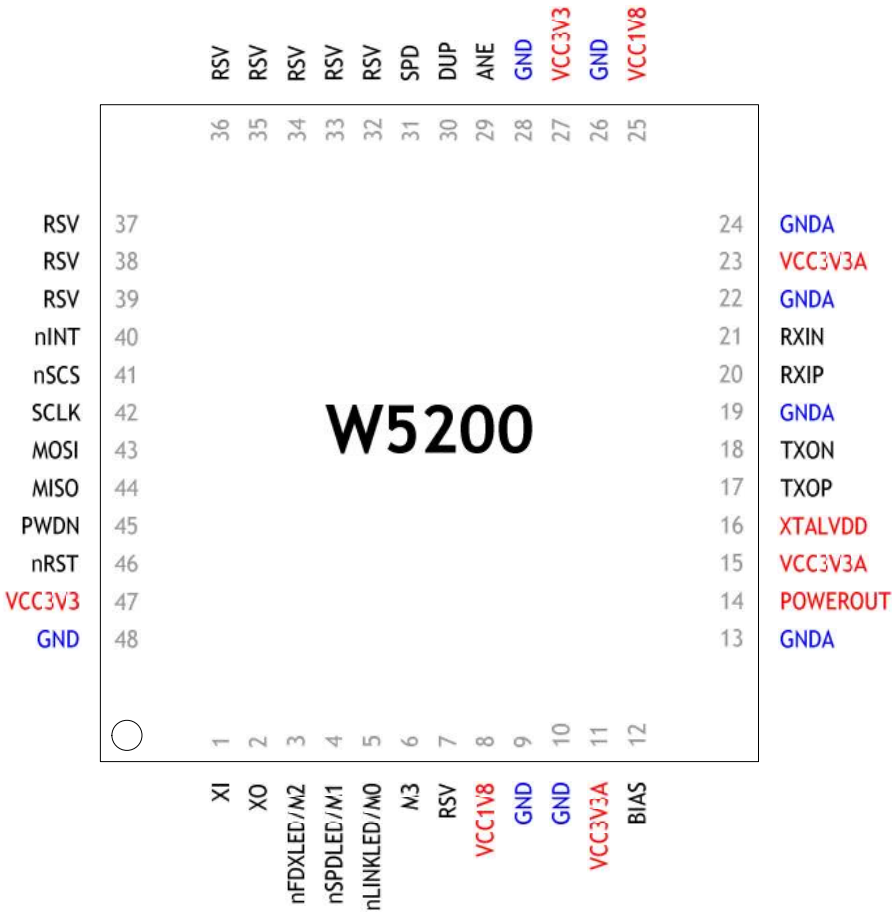


Figure 1 Pin Description W5200

1.1 MCU Interface Signals

Symbol	Type	Pin No	Description
nRST	I	46	**RESET (Active LOW)** active가 Low인 W5200을 최기화 하기 위한 핀이다. RESET Signal은 low 이후 최소 2us이상 유지해야 하고, High de-assert 이후 내부 PLL이 안정화 될 때가지 최소150ms를 유지해야 한다. "7 Electrical Specification" 참조
nSCS	I	41	**SPI SLAVE SELECT (Active LOW)** SPI interface에서 SPI Slave인 W5200을 선택할 때 사용한다.
nINT	O	40	**INTERRUPT (Active LOW)** W5200의 Interrupt Sources가 발행 할 경우 Low로 되며 다음과 같은 경우에 발생한다. socket connecting, disconnecting, data receiving timeout, and WOL (Wake

			on LAN). Interrupt는 IR(Interrupt Register)이 Clear될 때 High assert가 된다.
SCLK	I	42	**SPI CLOCK** SPI interface에서 SPI Clock signal로 사용한다.
MOSI	I	43	**SPI MASTER OUT SLAVE IN** SPI interface에서 SPI MOSI로 사용한다.
MISO	O	44	**SPI MASTER IN SLAVE OUT** SPI interface에서 SPI MISO로 사용한다.
PWDN	I	45	**POWER DOWN (Active HIGH)** 이 핀은 W5200의 Power down을 설정하는데 사용한다. Low: Normal Mode 활성화 High: Power Down Mode 활성화

1.2 1.2 PHY Signals

Symbol	Type	Pin No	Description
RXIP	I	20	**RXIP/RXIN Signal Pair** RXIN/RXIP differential signal, 수신 시에 differential 데이터는 RXIN/RXIP differential signal 을 통해 수신된다.
RXIN	I	21	
TXOP	O	17	**TXOP/TXON Signal Pair** TXON/TXOP differential signal, 전송 시에 differential 데이터는 TXON/TXOP differential signal 을 통해 전송된다.
TXON	O	18	
BIAS	O	12	**BIAS Register** 28.7kΩ±1% 저항을 통해 그라운드에 연결 된다. “Reference schematic” 참고.
ANE	I	29	**Auto Negotiation Mode Enable** Auto Negotiation Mode의 Enable/Disable을 한다. Low: Auto Negotiation Mode 비활성화 High: Auto Negotiation Mode 활성화
DUP	I	30	**Full Duplex Mode Enable** Full Duplex Mode의 Enable/Disable을 설정한다. Low: Half Duplex Mode 활성화 High: Full Duplex Mode 활성화 이 핀은 RESET period동안 활성화 된다.
SPD	I	31	**Speed Mode** Speed Mode를 100M/10M로 설정한다. Low: 10M Speed Mode 활성화 High: 100M Speed Mode 활성화 이 핀은 RESET period동안 활성화 된다.

1.3 Miscellaneous Signals

Symbol	Type	Pin No	Description
nFDXLED/M2 nSPDLED/M1 nLINKLED/M0	I	3, 4, 5	**W5200 MODE SELECT** Normal mode : 111 다른 modes는 internal test mode로 쓰인다. 이 핀은 RESET period동한 활성화 된다
M3	I	6	이 핀은 반드시 pull-up해야 한다.
RSV	-	7,32,33,34,35,36 ,37,38,39	**Reserved Pin** 7번 핀은 반드시 pull-up해야한다. 7번 이외의 핀들은 반드시 pull-down 또는 접지를 해야 한다.

- Notes: Pull-Up/Down register = 40KΩ to 100KΩ. Typical values are 75KΩ.

1.4 Power Supply Signals

Symbol	Type	Pin No	Description
VCC3V3A	Power	11, 15, 23	**3.3V power supply for Analog part**
VCC3V3	Power	27, 47	**3.3V power supply for Digital part**
VCC1V8	Power	8, 25	**1.8V power supply for Digital part**
GNDA	Ground	13, 19, 22, 24	**Analog ground**
GND	Ground	9, 10, 26, 28, 48	**Digital ground**
1V8O	O	14	**1.8V regulator output voltage** Core operation 전원과 내부 regulator에 의해 1.8V/200mA 전원을 만듦 (VCC1A8, VCC1V8) 출력 주파수를 안정시키기 위해 1V8O과 GND사이에 tantalum capacitor를 연결하고 고주파 noise decoupling을 위해 0.1uF capacitor를 연결한다. <Notice> 1V80은 W5200 전원으로만 사용하고 다른 device와는 연결하지 않는다.
XTALVDD	I	16	10.1uF capacitor를 그라운드와 연결한다.

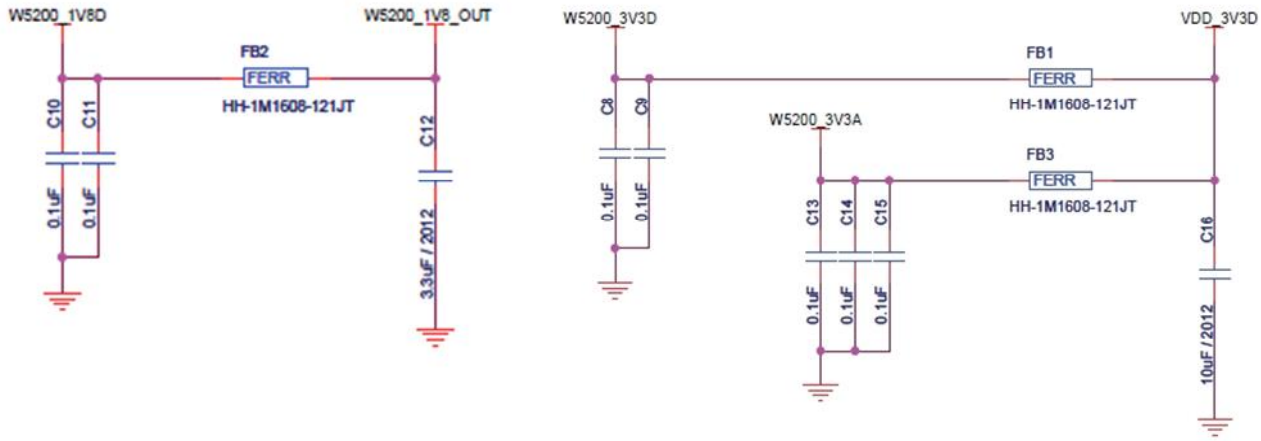
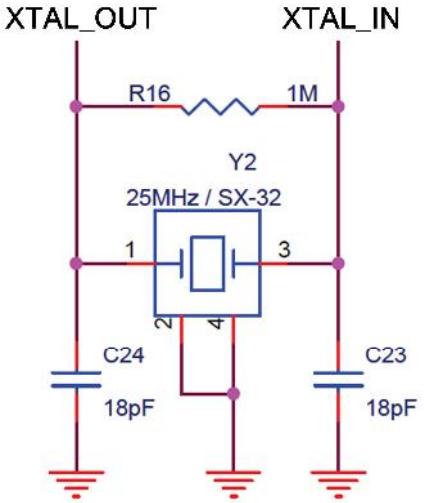


Figure 2 Power Design

안정적인 동작을 위한 권장사항이다.

1. RXIP/RXIN signal pair(RX)의 길이를 가능한 같게 한다.
2. TXOP/TXON signal pair(TX) 의 길이를 가능한 같게 한다.
3. RXIP와 RXIN signal은 최대한 가깝게 위치시킨다.
4. TXOP와 TXON signal은 최대한 가깝게 위치시킨다.
5. RX와 TX signal pair는 bias resistor나 crystal 같은 noisy signals과는 최대한 멀리 떨어지도록 한다.

1.5 Clock Signals

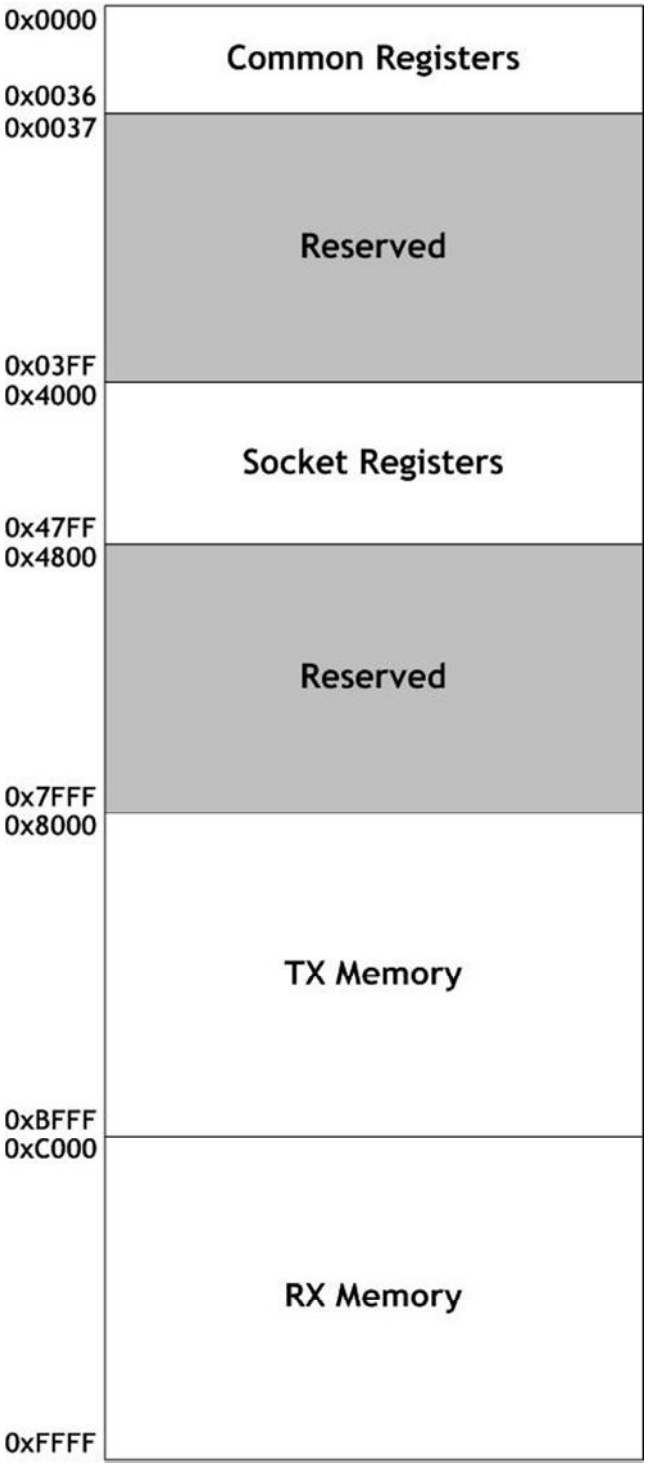
Symbol	Type	Pin No	Description
XI	I	1	**25MHz crystal input/output** clock input/output, 25MHz Crystal 혹은 Oscillator 를 연결한다.  Figure 3 Crystal Reference Schematic
XO	O	2	

1.6 LED Signals

Symbol	Type	Pin No	Description
nFDXLED/M2	O	3	**Full Duplex/Collision LED** Low: Full-duplex High: Half-duplex.
nSPDLED/M1	O	4	**Link speed LED** Low: 100Mbps High: 10Mbps
nLINKLED/M0	O	5	**Link LED** Low: Link (10/100M) High: Un-Link blink: TX or RX state on Link

2 Memory Map

W5200은 아래 그림처럼 Common register와 SOCKET register, TX memory, RX memory의 조합으로 구성되어있다.



W5200 Memory Map

3 W5200 Registers

3.1 common registers

Address	Register
0x0000	Mode (MR)
0x0001	Gateway Address (GAR0)
0x0002	(GAR1)
0x0003	(GAR2)
0x0004	(GAR3)
0x0005	Subnet Mask Address (SUBR0)
0x0006	(SUBR1)
0x0007	(SUBR2)
0x0008	(SUBR3)
0x0009	Source Hardware Address (SHAR0)
0x000A	(SHAR1)
0x000B	(SHAR2)
0x000C	(SHAR3)
0x000D	(SHAR4)
0x000E	(SHAR5)
0x000F	Source IP Address (SIPR0)
0x0010	(SIPR1)
0x0011	(SIPR2)
0x0012	(SIPR3)
0x0013 ~ 0x0014	Reserved
0x0015	Interrupt (IR)
0x0016	Socket Interrupt Mask (IMR2)
0x0017	Retry Time (RTR0)
0x0018	(RTR1)
0x0019	Retry Count (RCR)
0x001A ~ 0x001B	Reserved

Address	Register
0x001C	Authentication Type in PPPoE (PATR0)
0x001D	(PATR1)
0x001E	Authentication Algorithm in PPPoE (PPPALGO)
0x001F	Chip version(VERSIONR)
0x0020 ~ 0x0027	Reserved
0x0028	PPP LCP RequestTimer (PTIMER)
0x0029	PPP LCP Magic number (PMAGIC)
0x002A ~ 0x002F	Reserved
0x0030	Interrupt Low Level Timer (INTLEVEL0)
0x0031	(INTLEVEL1)
0x0032 ~ 0x0033	Reserved
0x0034	Socket Interrupt (IR2)
0x0035	PHY Status(PSTATUS)
0x0036	Interrupt Mask (IMR)

3.2 Socket registers

Note: n is socket number (0, 1, 2, 3, 4, 5, 6, 7)

Address	Register
0x4n00	Socket n Mode (Sn_MR)
0x4n01	Socket n Command (Sn_CR)
0x4n02	Socket n Interrupt (Sn_IR)
0x4n03	Socket n Status (Sn_SR)
0x4n04 0x4n05	Socket n Source Port (SN_PORT0) (SN_PORT1)
0x4n06 0x4n07 0x4n08 0x4n09 0x4n0A 0x4n0B	Socket n Destination Hardware Address (Sn_DHAR0) (Sn_DHAR1) (Sn_DHAR2) (Sn_DHAR3) (Sn_DHAR4) (Sn_DHAR5)
0x4n0C 0x4n0D 0x4n0E 0x4n0F	Socket 0 Destination IP Address (Sn_DIPR0) (Sn_DIPR1) (Sn_DIPR2) (Sn_DIPR3)
0x4n10 0x4n11	Socket 0 Destination Port (Sn_DPORT0) (Sn_DPORT1)
0x4n12 0x4n13	Socket 0 Maximum Segment Size (Sn_MSSR0) (Sn_MSSR1)
0x4n14	Socket 0 Protocol in IP Raw mode (Sn_PROTO)
0x4n15	Socket n IP TOS (Sn_TOS)
0x4n16	Socket n IP TTL (Sn_TTL)
0x4n17 ~ 0x4n1D	Reserved

Address	Register
0x4n1E	Receive Memory Size (Sn_RXMEM_SIZE)
0x4n1F	Transmit Memory Size (Sn_TXMEM_SIZE)
0x4n20 0x4n21	Socket 0 TX Free Size (Sn_TX_FSR0) (Sn_TX_FSR1)
0x4n22 0x4n23	Socket 0 TX Read Pointer (Sn_TX_RD0) (Sn_TX_RD1)
0x4n24 0x4n25	Socket 0 TX Write Pointer (Sn_TX_WR0) (Sn_TX_WR1)
0x4n26 0x4n27	Socket 0 RX Received Size (Sn_RX_RSR0) (Sn_RX_RSR1)
0x4n28 0x4n29	Socket 0 RX Read Pointer (Sn_RX_RD0) (Sn_RX_RD1)
0x4n2A 0x4n2B	Socket 0 RX Write Pointer (Sn_RX_WR0) (Sn_RX_WR1)
0x4n2C	Socket Interrupt Mask (Sn_IMR)
0x4n2D 0x4n2E	Fragment Offset in IP header (Sn_FRAG0) (Sn_FRAG1)
0x4n30 ~ 0x4nFF	Reserved

4 Register Descriptions

4.1 Common Registers

MR (Mode Register) [R/W] [0x0000] [0x00]

MR은 S/W reset, ping block mode, PPPoE mode에 사용된다.

7	6	5	4	3	2	1	0
RST			PB	PPPoE			

Bit	Symbol	Description
7	RST	**S/W Reset** 이 Bit가 '1'인 경우 내부 register는 초기화 되고 reset후에 자동으로 clear 된다.
6	Reserved	Reserved
5	Reserved	Reserved
4	PB	**Ping Block Mode** 0 : Disable Ping block 1 : Enable Ping block 만약 '1'로 설정 할 경우 Ping request에 대한 response를 하지 않는다.
3	PPPoE	**PPPoE Mode** 0 : DisablePPPoE mode 1 : EnablePPPoE mode 사용자가 router와 같은 장비 없이 ADSL을 사용하고자 한다면, 이 Bit를 '1' 로 설정하여 ADSL 서버에 연결한다. 자세한 사항은 'How to connect ADSL' 문서를 참고한다.
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

GAR (Gateway IP Address Register) [R/W] [0x0001 – 0x0004] [0x00]

GAR은 default gateway address를 설정할 때 사용한다.

Ex) In case of "192.168.0.1"

0x0001	0x0002	0x0003	0x0004
192 (0xC0)	168 (0xA8)	0 (0x00)	1 (0x01)

SUBR (Subnet Mask Register) [R/W] [0x0005 – 0x0008] [0x00]

SUBR은 subnet Mask address를 설정할 때 사용한다.

Ex) In case of "255.255.255.0"

0x0005	0x0006	0x0007	0x0008
255 (0xFF)	255 (0xFF)	255 (0xFF)	0 (0x00)

SHAR (Source Hardware Address Register) [R/W] [0x0009 – 0x000E] [0x00]

SHAR은 Source Hardware address를 설정할 때 사용한다.

Ex) In case of "00.08.DC.01.02.03"

0x0009	0x000A	0x000B	0x000C	0x000D	0x000E
0x00	0x08	0xDC	0x01	0x02	0x03

SIPR (Source IP Address Register) [R/W] [0x000F – 0x0012] [0x00]

SIPR은 Source IP address를 설정할 때 사용한다.

Ex) In case of "192.168.0.2"

0x000F	0x0010	0x0011	0x0012
192 (0xC0)	168 (0xA8)	0 (0x00)	2 (0x02)

IR (Interrupt Register) [R] [0x0015] [0x00]

IR은 Interrupt 발생여부를 판단하기 위해 Host processor에 MCU에서 access한다. IR Bit가 set되면 이 nINT 신호는 low 상태 asserted되고 IR의 모든 Bit들을 clear하지 않는 이상 high상태로 변하지 않는다. IR은 MCU에 Interrupt신호를 발생시킨다

7	6	5	4	3	2	1	0
CONFLICT	Reserved	PPPoE	Reserved	Reserved	Reserved	Reserved	Reserved

Bit	Symbol	Description
7	CONFLICT	**IP Conflict** ARP 요청에 Source IP address와 같은 IP address응답이 있다면, 이 Bit는 '1'로 set된다. 이 Bit는 '1'을 write함으로써 '0'으로 clear할 수 있다.
6	Reserved	Reserved
5	PPPoE	**PPPoE Connection Close** PPPoE mode에서 이 Bit가 '1'인 경우는 PPPoE 연결이 closed임을 나타낸다. 이 Bit는 '1'을 write함으로써 '1'으로 clear할 수 있다.
4	Reserved	Reserved
3	Reserved	Reserved
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

IMR (Interrupt Mask Register) [R/W] [0x0036] [0x00]

IMR (Interrupt Mask Register)는 Interrupt를 Mask하는데 사용한다. 각 Interrupt Mask Bit는 Interrupt register (IR)의 Bit와 같다. Interrupt Mask Bit가 set되어있다면, IR의 해당 Bit가 set되었을 때 Interrupt가 발생 할 것이다. 만약 IMR이 '0'으로 set되어 있다면, IR의 해당 Bit가 set되더라도 Interrupt는 발생하지 않을 것이다.

7	6	5	4	3	2	1	0
IM_IR7	Reserved	IM_IR5	Reserved	Reserved	Reserved	Reserved	Reserved

Bit	Symbol	Description
7	IM_IR7	**IP Conflict Enable**
6	Reserved	Reserved
5	IM_IR5	**PPPoE Close Enable**
4	Reserved	Reserved
3	Reserved	Reserved
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

RTR (Retry Time-value Register) [R/W] [0x0017 – 0x0018] [0x07D0]

RTR은 timeout 주기를 설정한다. 이 register에서 1값이 갖는 의미는 100us와 같다. Default timeout은 2000 (0x07D0) 즉, 200ms이다.

Ex) When timeout-period is set as 400ms, RTR = (400ms / 1ms) X 10 = 4000(0x0FA0)

0x0017	0x0018
0x0F	0xA0

만약 peer로 부터 응답이 없거나 정해진 timeout시간 보다 delay가 길어질 경우 재 전송이 발생하며, CONNECT, DISCON, CLOSE, SEND, SEND_MAC, SEND_KEEP 등에서 발생한다.

RCR (Retry Count Register) [R/W] [0x0019] [0x08]

RCR은 재 전송 횟수를 설정한다. 만약 재 전송횟수가 RCR에 저장된 횟수 이상으로 발생할 경우, Timeout Interrupt가 발생한다. (SOCKET*n* Interrupt Register (Sn_IR)의 TIMEOUT Bit는 '1'로 설정함)

TCP 통신인 경우, Sn_IR(TIMEOUT)= '1'과 동시에 Sn_SR의 값이 'SOCK_CLOSED'로 변경된다.

TCP 통신이 아닌 경우, Sn_IR(TIMEOUT) = '1'만 된다.

Ex) RCR = 0x0007

0x0019
0x07

W5200에서의 Timeout은 RTR과 RCR로 Data 재전송의 시간과 횟수를 설정할 수 있다. W5200의 Timeout에 대해 좀더 살펴 보면, ARP retransmission timeout과 TCP retransmission timeout

2가지가 있다.

먼저 ARP("RFC 826" 참조, http://www.ietf.org/rfc.html) retransmission timeout 살펴보면, W5200은 IP, UDP, TCP를 이용한 통신시 상대방의 IP address로 MAC address를 알기 위해 자동으로 ARP-request를 전송한다. 이때 상대방의 ARP-response 수신을 기다리는데, RTR의 설정 대기 시간 동안 ARP-response의 수신이 없으면, Timeout이 발생하고 ARP-request를 Retransmission한다. 이와 같은 작업은 'RCR + 1'만큼 반복하게 된다.

'RCR + 1'개의 ARP-request retransmission이 일어나고, 그에 대한 ARP-response가 없다면, Final timeout이 발생하게 되고, Sn_IR(TIMEOUT) = '1' 된다.

ARP-request의 Final timeout(ARP_{TO}) 값은 다음과 같다.

$$ARP_{TO} = (RTR \times 0.1ms) \times (RCR + 1)$$

TCP packet retransmission timeout을 살펴보면, W5200은 TCP packet (SYN, FIN, RST, DATA packet)을 전송하고 그에 대한 Acknowledgment(ACK)을 RTR과 RCR에 의해 설정된 대기 시간 동안 기다리게 된다. 이때 상대방으로부터 ACK가 없으면 Timeout이 발생하고 이전에 보냈던 TCP packet을 Retransmission한다. 이와 같은 작업은 'RCR + 1'만큼 반복하게 된다.

'RCR + 1'개의 TCP packet retransmission이 일어나고, 그에 대한 ACK 수신이 없다면, Final timeout이 발생하게 되고, Sn_IR(TIMEOUT) = '1'과 동시에 Sn_SR이 'SOCK_CLOSED"로 변경된다. TCP packet retransmission의 Final timeout(TCP_{TO}) 값은 다음과 같다1'

$$TCP_{TO} = \left(\sum_{N=0}^{M} (RTR \times 2^N) + ((RCR-M) \times RTR_{MAX}) \right) \times 0.1ms$$

N : Retransmission count, 0<= N <= M
M : Minimum value when RTR X $2^{(M+1)}$ > 65535 and 0 <= M <= RCR
RTR_{MAX}: RTR X 2^M

Ex) When RTR = 2000(0x07D0), RCR = 8(0x0008),

ARP_{TO} = 2000 X 0.1ms X 9 = 1800ms = 1.8s

TCP_{TO} = (0x07D0+0x0FA0+0x1F40+0x3E80+0x7D00+0xFA00+0xFA00+0xFA00+0xFA00) X 0.1ms
= (2000 + 4000 + 8000 + 16000 + 32000 + ((8 - 4) X 64000)) X 0.1ms
= 318000 X 0.1ms = 31.8s

PATR (Authentication Type in PPPoE mode) [R] [0x001C-0x001D] [0x0000]

PATR은 PPPoE 연결을 위한 인증 type을 알려준다. W5200은 PAP와 CHAP의 2가지 type의 인증 방법을 지원한다.

Value	Authentication Type
0xC023	PAP
0xC223	CHAP

PPPALGO (Authentication Algorithm in PPPoE mode)[R][0x001E][0x00]

PPPALGO는 PPPoE연결의 인증 알고리즘을 알려준다. 자세한 정보는 PPPoE application note를 참고하기 바란다.

VERSIONR (W5200 Chip Version Register)[R][0x001F][0x03]

VERSIONR은 W5200 chip version을 나타내는 register이다.

PTIMER (PPP Link Control Protocol Request Timer Register) [R/W] [0x0028]

PTIMER은 LCP echo request를 보내는 지속시간을 나타낸다. 1의 값은 25ms를 의미한다.

Ex) in case that PTIMER is 200,

200 * 25(ms) = 5000(ms) = 5 seconds

PMAGIC (PPP Link Control Protocol Magic number Register) [R/W] [0x0029][0x00]

PMAGIC은 LCP negotiation도중에 Magic number option을 설정하는데 사용된다. Application note 'How to connect ADSL'을 참고하기 바란다.

INTLEVEL (Interrupt Low Level Timer Register)[R/W][0x0030 – 0x0031][0x0000]

INTLEVEL register는 Interrupt Assert wait time(I_{AWT})을 설정한다. 다음 Interrupt가 발생했을 때 설정한 시간(I_{AWT})만큼 기다린 다음 칩 내부의 nINT신호를 Low로 assert한다.

I_{AWT} = (INTLEVEL0 + 1) * PLL_CLK (when INTLEVEL0 > 0)

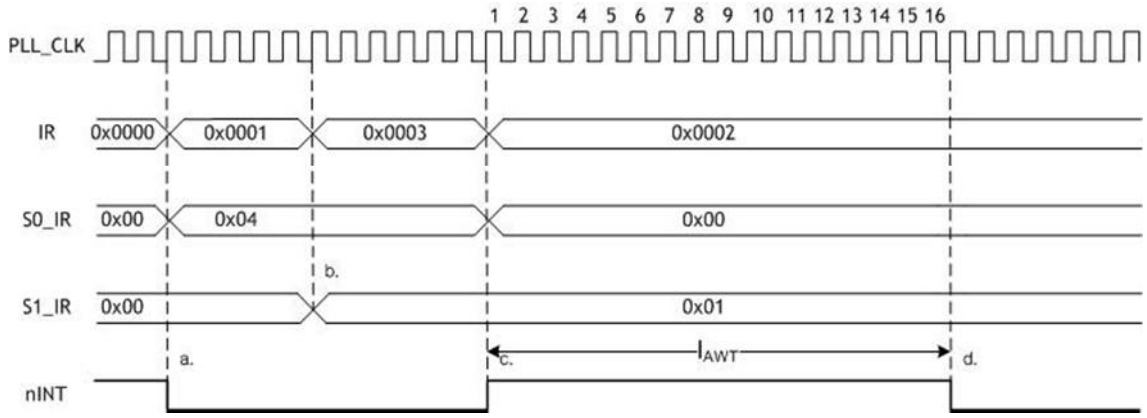


Figure 4 INTLEVEL Timing

a. 소켓0 에서 Interrupt가 발생했다면 (S0_IR(3) = '1' 해당 IR2 Bit도 '1'로 set되고 nINT신호는 Low로 된다.
b. 연속해서 소켓1 에서 Interrupt가 발생하면 (S1_IR1(0) = '1') 해당 IR2 Bit가 '1'로 set된다.
c. MCU는 S0_IR을 clear(S0_IR1 = 0x00) 하고 해당 IR2 Bit 또한 clear한다. 칩 내부의 nINT신호는 High로 된다.
d. 여기서 S0_IR이 clear되었지만, socket1 Interrupt 때문에 IR2의 값은 0x00이 아니다. 따라서 칩 내부의 nINT신호는 Low로 되어야 한다. 이 때 INTLEVEL register의 값이 0x000F라면 칩 내부의 nINT신호는 I_{AWT}(16 PLL_CLK) time 후에 Low로 된다.

IR2(W5200 SOCKET Interrupt Register)[R/W][0x0034][0x00]

IR2는 W5200 SOCKET Interrupt가 발생할 경우 이것을 알려주는 register이다. Interrupt가 발생했을 때, IR2의 해당 Bit가 set된다. 이 경우 IR2의 모든 Bit들이 '0'으로 clear될 때까지 nINT 신호는 low상태가 된다. Sn_IR Bit들을 이용해서 IR2 register를 clear하면 nINT 신호는 high 상태가 된다.

7	6	5	4	3	2	1	0
S7_INT	S6_INT	S5_INT	S4_INT	S3_INT	S2_INT	S1_INT	S0_INT

Bit	Symbol	Description
7	S7_INT	SOCKET 7에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S7_IR에 반영된다. 이 Bit는 사용자에 의해 S7_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
6	S6_INT	SOCKET 6에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S6_IR에 반영된다. 이 Bit는 사용자에 의해 S6_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
5	S5_INT	SOCKET 5에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S5_IR에 반영된다. 이 Bit는 사용자에 의해 S5_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
4	S4_INT	SOCKET 4에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S5_IR에 반영된다. 이 Bit는 사용자에 의해 S4_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
3	S3_INT	SOCKET 3에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S3_IR에 반영된다. 이 Bit는 사용자에 의해 S3_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
2	S2_INT	SOCKET 2에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S2_IR에 반영된다. 이 Bit는 사용자에 의해 S2_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
1	S1_INT	SOCKET 1에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S1_IR에 반영된다. 이 Bit는 사용자에 의해 S1_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.
0	S0_INT	SOCKET 0에서 Interrupt가 발생한 경우, 이 Bit는 '1'로 되고 이 Interrupt 정보는 S0_IR에 반영된다. 이 Bit는 사용자에 의해 S0_IR가 0x00으로 clear되면 이에 따라 자동으로 clear된다.

PHYSTATUS(W5200 PHY status Register)[R/W][0x17]

W5200 PHY의 상태를 나타내는 레지스터이다.

Bit	Symbol	Description
7	Reserved	Reserved
6	Reserved	Reserved
5	LINK	**Link Status Register[Read Only]** 0 : Link down 1 : Link Up
4	Reserved	Reserved
3	POWERDOWN	**Power down mode of PHY[Read Only]** 0 : Disable Power down mode(operates normal mode) 1 : Enable Power down mode
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

IMR2(Interrupt Mask Register2)[R/W][0x0016][0x00]

IMR2 (Interrupt Mask Register)는 Interrupt를 Mask하기 위해 사용한다. 각 Interrupt Mask Bit는 Interrupt Register2 (IR2)의 Bit와 같다. Interrupt Mask Bit가 set되어있다면, IR2의 해당 Bit가 set되었을 때 Interrupt가 발생 할 것이다. 만약 IMR이 '0'으로 set되어 있다면, IR2의 해당 Bit가 set되더라도 Interrupt는 발생하지 않는다

7	6	5	4	3	2	1	0
S7_INT	S6_INT	S5_INT	S4_INT	S3_INT	S2_INT	S1_INT	S0_INT

Bit	Symbol	Description
7	S7_INT	**IR(S7_INT) Interrupt Mask**
6	S6_INT	**IR(S6_INT) Interrupt Mask**
5	S5_INT	**IR(S5_INT) Interrupt Mask**
4	S4_INT	**IR(S4_INT) Interrupt Mask**
3	S3_INT	**IR(S3_INT) Interrupt Mask**
2	S2_INT	**IR(S2_INT) Interrupt Mask**
1	S1_INT	**IR(S1_INT) Interrupt Mask**
0	S0_INT	**IR(S0_INT) Interrupt Mask**

4.2 Socket Registers

Sn[1]_MR (SOCKET n-th Mode Register) [R/W] [0x4000+0x0n00] [0x00][2]

Sn_MR은 SOCKET n의 option이나 protocol type등을 설정한다.

7	6	5	4	3	2	1	0
MULTI		ND / MC		P3	P2	P1	P0

Bit	Symbol	Description
7	MULTI	**Multicasting** 0 : disable Multicasting 1 : enable Multicasting 이 Bit는 UDP(P3-03: "0010")일 경우에만 유효하다. Multicasting을 사용하기 위해 OPEN 명령 이전에 SOCKET n destination IP와 port register에 각각 multicast group address와 port number를 write한다.
6	MF	**MAC Filter** 0 : Disable MAC filter 1 : Enable MAC filter 이 Bit는 MACRAW(P3~P0: "0100")일 경우에만 유효하다. '1'로 설정될 경우, W5200은 Broadcasting packet이나 자신에게 전송되는 Packet만을 수신하게 된다. '0'으로 설정될 경우, W5200은 Ethernet 상의 모든 Packet을 수신하게 된다. Hybrid TCP/IP stack을 구현하고자 하는 경우, Host의 수신 Overhead를 감소시키기 위해 이 Bit를 '1'로 설정할 것을 권장한다.
5	ND/MC	**Use No Delayed ACK** 0 : Disable No Delayed ACK option 1 : Enable No Delayed ACK option, 이 기능은 TCP의 경우에만 적용됨 (P3-P0: '0001') 만약 이 Bit가 '1'로 set되어있다면 peer로부터 데이터 packet을 수신한 다음 곧바로 ACK packet이 전송될 것이다. 만약 이 Bit가 '0'이라면 ACK packet은 내부 timeout 메커니즘에 따라 전송된다. **Multicast** 0 : using IGMP version 2 1 : using IGMP version 1 이 Bit는 MULTI Bit가 enable상태이고 UDP모드일 때 유효함 (P3-P0: '0010') 추가적으로 multicast는 IGMP message에 Join/Leave/Report와 같은 version number를 Multicast group으로 보낸다.
4	Reserved	Reserved

[1] *n* is SOCKET number (0, 1, 2, 3, 4, 5, 6, 7).
[2] [Read/Write] [address of socket 0, address of socket 1, address of socket 2, address of socket 3,address of socket 4, address of socket 5, address of socket 6, address of socket 7] [Reset value]

3	P3	**Protocol** 해당 SOCKET의 TCP, UDP, IPRAW등의 protocol을 설정한다.
2	P2	
1	P1	
0	P0	* S0_MR_MACRAW와 S0_MR_PPPoE는 SOCKET 0에만 쓸 수 있다. S0_MR_PPPoE는 임시로 PPPoE server connection/Termination에 사용되지만, Connection이 이루어지면, 다른 protocol로 사용 할 수 있다.

P3	P2	P1	P0	Meaning
0	0	0	0	Closed
0	0	0	1	TCP
0	0	1	0	UDP
0	0	1	1	IPRAW

P3	P2	P1	P0	Meaning
0	1	0	0	MACRAW
0	1	0	1	PPPoE

Sn_CR (SOCKET n Command Register) [R/W] [0x4001+0x0n00] [0x00]

Sn_CR은 OPEN, CLOSE, CONNECT, LISTEN, SEND, RECEIVE와 같은 SOCKET n의 명령을 설정하는데 사용한다. W5200이 명령을 인식하고 난 다음 Sn_CR은 W5200에 의해 자동으로 clear 된다. Sn_CR이 0x00으로 clear되었더라도, 해당 명령은 여전히 처리 중 일 수 있다. Sn_CR의 명령처리가 완료되었는지는 Sn_IR이나 Sn_SR을 확인하면 된다.

<table>
<tr><th>Value</th><th>Symbol</th><th>Description</th></tr>
<tr><td>0x01</td><td>OPEN</td><td>SOCKET n은 초기화 되고 Sn_MR (P3:P0)로 선택한 protocol에 따라 open된다. 아래 테이블은 Sn_MR에 따른 Sn_SR값을 보여준다
<table>
<tr><th>Sn_MR(P3:P0)</th><th>Sn_SR</th></tr>
<tr><td>Sn_MR_CLOSE (0x00)</td><td>-</td></tr>
<tr><td>Sn_MR_TCP (0x01)</td><td>SOCK_INIT (0x13)</td></tr>
<tr><td>Sn_MR_UDP (0x02)</td><td>SOCK_UDP (0x22)</td></tr>
<tr><td>Sn_MR_IPRAW (0x03)</td><td>SOCK_IPRAW (0x32)</td></tr>
<tr><td>S0_MR_MACRAW (0x04)</td><td>SOCK_MACRAW (0x42)</td></tr>
<tr><td>S0_MR_PPPoE (0x05)</td><td>SOCK_PPPoE (0x5F)</td></tr>
</table></td></tr>
<tr><td>0x02</td><td>LISTEN</td><td>LISTEN은 TCP mode (Sn_MR(P3:P0) = Sn_MR_TCP)에서만 유효하다
이 모드에서, SOCKET n 은 ‘TCP CLIENT’로부터 connection-request (SYN packet)을 기다리는 TCP server로 설정된다. 이 경우 Sn_SR의 상태는 SOCKET_INIT에서 SOCKET_LISTEN으로 바뀐다.
Client의 connection-request가 성공적으로 established되면 Sn_SR의 상태는 SOCK_LISTEN에서 SOCK_ESTABLISHED로 변하고 Sn_IR(0)은 ‘1’로 된다. 반면에 connection failure (SYN/ACK packet전송 실패)의 경우 Sn_IR(3)은 ‘1’로 set되고 Sn_SR의 상태는 SOCK_CLOSED로 변한다.
cf> 만약 connection request동안 TCP client의 destination port가 존재하지 않을 경우, W5200은 RST packet을 전송하고 Sn_SR의 상태는 변하지 않는다.</td></tr>
<tr><td>0x04</td><td>CONNECT</td><td>CONNECT는 TCP mode(Sn_MR(P3:P0) = Sn_MR_TCP)에서만 유효하고SOCKET n이 ‘TCP CLIENT’로 동작할 경우 사용된다. CONNECT는 Sn_DIPR와 Sn_DPORTR로 설정된 ‘TCP SERVER’에게 Connect-request(SYN packet)를 전송한다. Connect-request가 성공했을 경우(SYN/ACK packet을 수신했을 경우), Sn_IR(0)=‘1’로 되고 Sn_SR은 SOCK_ESTABLISHED로 변경된다.
Connect-request가 실패했을 경우는 다음과 같이 3가지가 있다.
- ARP-process를 통해 Destination hardware address를 얻지 못하여 ARP_{TO}가 발생(Sn_IR(3)=‘1’)한 경우
- SYN/ACK packet를 수신 못하고 TCP_{TO}가 발생(Sn_IR(3)= ‘1’)한 경우
- SYN/ACK packet 대신 RST packet을 수신했을 경우.
위와 같은 경우 Sn_SR은 SOCK_CLOSED상태로 바뀐다.</td></tr>
<tr><td>0x08</td><td>DISCON</td><td>DISCON은 TCP mode일 때만 유효하다.
W5200은 ‘TCP SERVER’와 ‘TCP CLIENT’에 상관없이, 접속중인 상대방에게</td></tr>
</table>

		Disconnect-request(FIN packet)를 전송하거나(Active close), 상대방으로부터 Disconnect-request(FIN packet)을 수신했을 때(Passive close), W5200은 FIN packet을 전송한다(Disconnect-process). Disconnect-request가 성공했다면(FIN/ACK packet을 수신했을 경우), Sn_SR은 SOCK_CLOSED로 변경된다. 그러나 Disconnect-request가 실패했다면, TCP_{TO}가 발생(Sn_IR(3)= '1')하고 Sn_SR은 SOCK_CLOSED로 변경된다. cf> ISCON 대신 CLOSE를 사용할 경우, Disconnect-process (disconnect-request 전송) 이, 단지 Sn_SR만 SOCK_CLOSED로 변경된다. 그리고 통신 중 상대방으로부터 RST packet을 수신할 경우, 무조건 Sn_SR은 SOCK_CLOSED로 변경된다.
0x10	CLOSE	SOCKET n 을 close한다. 이 때 Sn_SR은 SOCK_CLOSED로 변경된다.
0x20	SEND	SEND는 TX memory의 buffer에 있는 데이터를 송신하려는 Data Length만큼 송신한다. 자세한 사항은 SOCKET n TX Free Size Register (Sn_TX_FSR0), SOCKET n TX Write Pointer Register (Sn_TX_WR0), SOCKET n TX Read Pointer Register(Sn_TX_RD0)를 참고하기 바란다.
0x21	SEND_MAC	SEND_MAC은 UDP mode일 때만 유효하다. 기본동작은 SEND와 같다. SEND는 자동으로 ARP-process를 통해 Destination hardware address를 얻은 후 Data를 전송하는 반면, SEND_MAC은 Host가 설정한 Sn_DHAR을 Destination hardware address로 하여 Data를 전송한다.
0x22	SEND_KEEP	SEND_KEEP은 TCP mode일 때만 유효하다. Keep alive packet을 송신하여 connection이 유효한지 확인한다. 만약 상대방이 더 이상 응답이 없어서 connection이 유효하지 않은 경우 connection을 종료한다. Timeout Interrupt가 발생한다.
0x40	RECV	RECV는 RX read pointer register (Sn_RX_RD0)를 이용해서 데이터를 수신한다. 자세한 사항은 5.2.1.1 SERVER mode의 Receiving Process와 SOCKET n RX Received Size Register (Sn_RX_RSR0), SOCKET n RX Write Pointer Register(Sn_RX_WR0), and SOCKET n RX Read Pointer Register(Sn_RX_RD0)를 참고하기 바란다.

아래 command들은 SOCKET0이고 S0_MR(P3:P0)=S0_MR_PPPoE일 때만 유효하다.

Value	Symbol	Description
0x23	PCON	PPPoE Discovery Packet을 전송하여 ADSL 연결 시작한다.
0x24	PDISCON	ADSL connection 종료한다.
0x25	PCR	각 과정에서, REQ message 전송한다.
0x26	PCN	각 과정에서, NAK message 전송한다.
0x27	PCJ	각 과정에서, REJECT message 전송한다.

Sn_IR (SOCKET n Interrupt Register) [R] [0x4002+0x0n00] [0x00]

Sn_IR register는 SOCKET n의 Interrupt (establishment, termination, receiving data, timeout) type과 같은 정보를 제공한다. Interrupt가 발생하고 Sn_IMR의 해당 Mask Bit가 '1'인 경우 Sn_IR의 Interrupt Bit는 '1'로 된다. Sn_IR Bit를 clear하기 위해서는, 해당 Bit에 다시 '1'을 write해야 한다.

7	6	5	4	3	2	1	0
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	CON

Bit	Symbol	Description
7	PRECV	**Sn_IR(PRECV) Interrupt Mask** 'SOCKET=0'이고' S0_MR(P3:P0)=S0_MR_PPPoE'인 경우의 Receive Interrupt, 지원하지 않는 옵션 데이터를 수신 시 발생한다.
6	PFAIL	**Sn_IR(PFAIL) Interrupt Mask** 'SOCKET=0'이고' S0_MR(P3:P0)=S0_MR_PPPoE'인 경우의 PPP Fail Interrupt, PAP 인증이 실패한 경우 발생한다.
5	PNEXT	**Sn_IR(PNEXT) Interrupt Mask** 'SOCKET=0'이고' S0_MR(P3:P0)=S0_MR_PPPoE'인 경우의 PPP Next Phase Interrupt, ADSL연결 과정에서 phase가 변할 때 발생한다.
4	SEND_OK	**Sn_IR(SENDOK) Interrupt Mask** SEND OK Interrupt, SEND명령이 완료되면 발생한다.
3	TIMEOUT	**Sn_IR(TIMEOUT) Interrupt Mask** TIMEOUT Interrupt, ARP_{TO} 혹은 TCP_{TO} 가 발생한 경우 발생한다.
2	RECV	**Sn_IR(RECV) Interrupt Mask** Receive Interrupt, peer로 부터 데이터 packet이 수신된 경우 발생한다.
1	DISCON	**Sn_IR(DISCON) Interrupt Mask** Disconnect Interrupt, peer로 부터 FIN/ACK packet의 FIN이 수신된 경우 발생한다.
0	CON	**Sn_IR(CON) Interrupt Mask** Connect Interrupt, peer와 연결이 성립되어 SOCKET status가 established로 바뀔 때 1번 발생한다.

*Sn*_SR (SOCKET n Status Register) [R] [0x4003+0x0n00] [0x00]

Sn_SR은 SOCKETn의 SOCKET 상태를 알려준다. SOCKET status는 Sn_CR의 Command나, packet 송수신중에 변경될 수 있다.

Value	Symbol	Description
0x00	SOCK_CLOSED	SOCKET n의 resource가 release된 상태로서 DISCON, CLOSE command가 수행되거나 ARP_{TO}, TCP_{TO}가 발생했을 경우 이전 값에 관계없이 상태가 변한다.
0x13	SOCK_INIT	SOCKET n이 TCP mode로 open되고 TCP연결의 첫 단계로 initialize된 상태이다. 사용자는 LISTEN과 CONNECT명령을 사용할 수 있다. Sn_MR(P3:P0)이 Sn_MR_TCP이고 OPEN명령을 사용했을 때, Sn_SR의 상태는 SOCK_INIT으로 변한다.
0x14	SOCK_LISTEN	SOCKET n이 TCP server mode로 동작하며, 'TCP CLIENT'로부터 connection-request(SYN packet)를 기다리는 상태다. LISTEN 명령을 사용하면, Sn_SR의 상태는 SOCK_LISTEN으로 변한다. SOCK_LISTEN상태에서 'TCP CLIENT'의 Connect-request (SYN packet)를 성공적으로 처리했을 경우 Sn_SR의 상태는 SOCK_ESTABLISHED로 바뀌고, 실패했을 경우 TCP_{TO}가 발생(Sn_IR(TIME OUT)='1')하고 SOCK_CLOSED로 바뀐다.
0x17	SOCK_ESTABLISHED	TCP 연결이 성립된 상태로서 SOCK_LISTEN상태에서 'TCP CLIENT'의 SYN packet 처리를 성공했을 경우나 CONNECT command에 수행이 성공했을 경우 Sn_SR의 상태는 SOCK_ESTABLEISHED로 바뀐다. 이 상태에서는 DATA packet 송수신이 가능하다. 즉 SEND나 RECV command를 수행할 수 있다.
0x1C	SOCK_CLOSE_WAIT	Peer로부터 disconnect-request(FIN packet)를 수신한 상태로서 TCP connection이 완전히 disconnect된 것이 아닌 half close 상태이므로 DATA packet 송수신이 가능하다. TCP connection을 완전히 disconnect하기 위해서는 DISCON 명령을 수행해야 한다. 하지만 단순히 SOCKET을 close하려면 CLOSE 명령을 수행한다.
0x22	SOCK_UDP	SOCKET n이 UDP mode로 Open된 상태, Sn_MR(P3:P0) = Sn_MR_UDP인 상태에서 OPEN 명령이 수행되었을 때 Sn_SR은 SOCK_UDP상태로 바뀐다. TCP mode SOCKET과 달리 connection-process없이 직접 DATA packet을 송수신할 수 있다. (3 way handshake)
0x32	SOCK_IPRAW	SOCKET n이 IPRAW mode로 Open된 상태, Sn_MR(P3:P0) = Sn_MR_IPRAW인 상태에서 OPEN 명령이 수행되었을 때 Sn_SR은 SOCK_IPRAW상태로 바뀐다. UDP mode SOCKET 처럼 connection-process없이 직접 IP packet을 송수신할 수 있다.
0x42	SOCK_MACRAW	Socket 0-th가 MACRAW mode로 Open된 상태, S0_MR(P3:P0) = S0_MR_MACRAW인 상태에서 OPEN 명령이 수행되었을 때 Sn_SR은 SOCK_MACRAW상태로 바뀐다. UDP mode SOCKET처럼 connection-process없이 직접 MAC packet (Ethernet frame)을 송수신할 수 있다.

0x5F	SOCK_PPPOE	Socket 0-th가 이 PPPoE mode로 Open된 상태, S0_MR(P3:P0) = S0_MR_PPPoE인 상태에서 OPEN 명령이 수행되었을 때 Sn_SR은 SOCK_PPPOE상태로 바뀐다.

아래의 Socket status는 Sn_SR의 천이 과정에서 관찰되는 Temporary Status들 이다.

Value	Symbol	Description
0x15	SOCK_SYNSENT	SOCK_SYNSENT상태는 'TCP SERVER'에게 Connect-request (SYN packet)를 전송한 상태로서, CONNECT 명령에 의해 Sn_SR의 상태가 SOCK_INIT에서 SOCK_ESTABLISEHD로 바뀔 때 나타난다. 이 상태에서 'TCP SEVER'로부터 Connect-accept (SYN/ACK packet)를 수신할 경우 자동으로 SOCK_ ESTABLISHED상태로 바뀐다. 하지만 'TCP SEVER'로부터 TCP_{TO} 가 발생하기 전까지 (Sn_IR(TIMEOUT)='1') SYN/ACK packet을 수신하지 못할 경우에는 SOCK_CLOSED 상태로 바뀐다.
0x16	SOCK_SYNRECV	SOCK_SYNRECV상태는 'TCP CLIENT'로부터 connect-request (SYN packet)를 수신한 상태로서, 이 상태에서 W5200이 connect-request에 대한 응답으로 connect-accept (SYN/ACK packet)을 'TCP CLIENT'에게 성공적으로 전송하였을 경우에는 자동으로 SOCK_ESTABLISHED상태로 바뀐다. 하지만 전송에 실패하였을 경우 Timeout Interrupt가 발생하고 (Sn_IR(TIME OUT)='1') SOCK_CLOSED 상태로 바뀐다.
0x18	SOCK_FIN_WAIT	SOCKETn이 Closing되는 상태로서, Active close나 Passive close인 경우의 Disconnect-process에서 나타나는 상태다. Disconnect-process 과정이 성공적으로 완료되거나, Timeout Interrupt가 발생하면 (Sn_IR(TIMEOUT)='1') SOCK_ CLOSED상태로 변한다.
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	Passive Closing인 경우 W5200이 전송한 FIN패킷에 대한 ACK를 기다리는 상태이며, Timeout Interrupt가 발생하면 (Sn_IR(TIMEOUT)='1') SOCK_ CLOSED상태로 변한다
0x01	SOCK_ARP	Destination hardware address를 찾기 위해 peer로 ARP-request를 전송하는 상태로서 SOCK_UDP나 SOCK_IPRAW에서 SEND 명령을 수행 할 경우 나타나는 상태다. Peer로부터 Hardware address를 성공적으로 수신한 경우 (ARP-response을 수신한 경우), SOCK_UDP, SOCK_IPRAW, SOCK_SYNSENT로 각각 상태가 변하고, 실패할 경우 timeout Interrupt가 발생하고 (Sn_IR(TIMEOUT)='1'), UDP나 IPRAW mode일 경우 이전 Status인 SOCK_UDP나 SOCK_IPRAW로 되돌아가며, TCP인 경우 SOCK_CLOSED로 상태가 바뀐다. cf> SOCK_UDP나 SOCK_IPRAW에서, 이전 SEND command에 대한 Sn_DIPR와 현재 SEND command의 Sn_DIPR이 다를 경우에만 ARP-process가 동작한다.

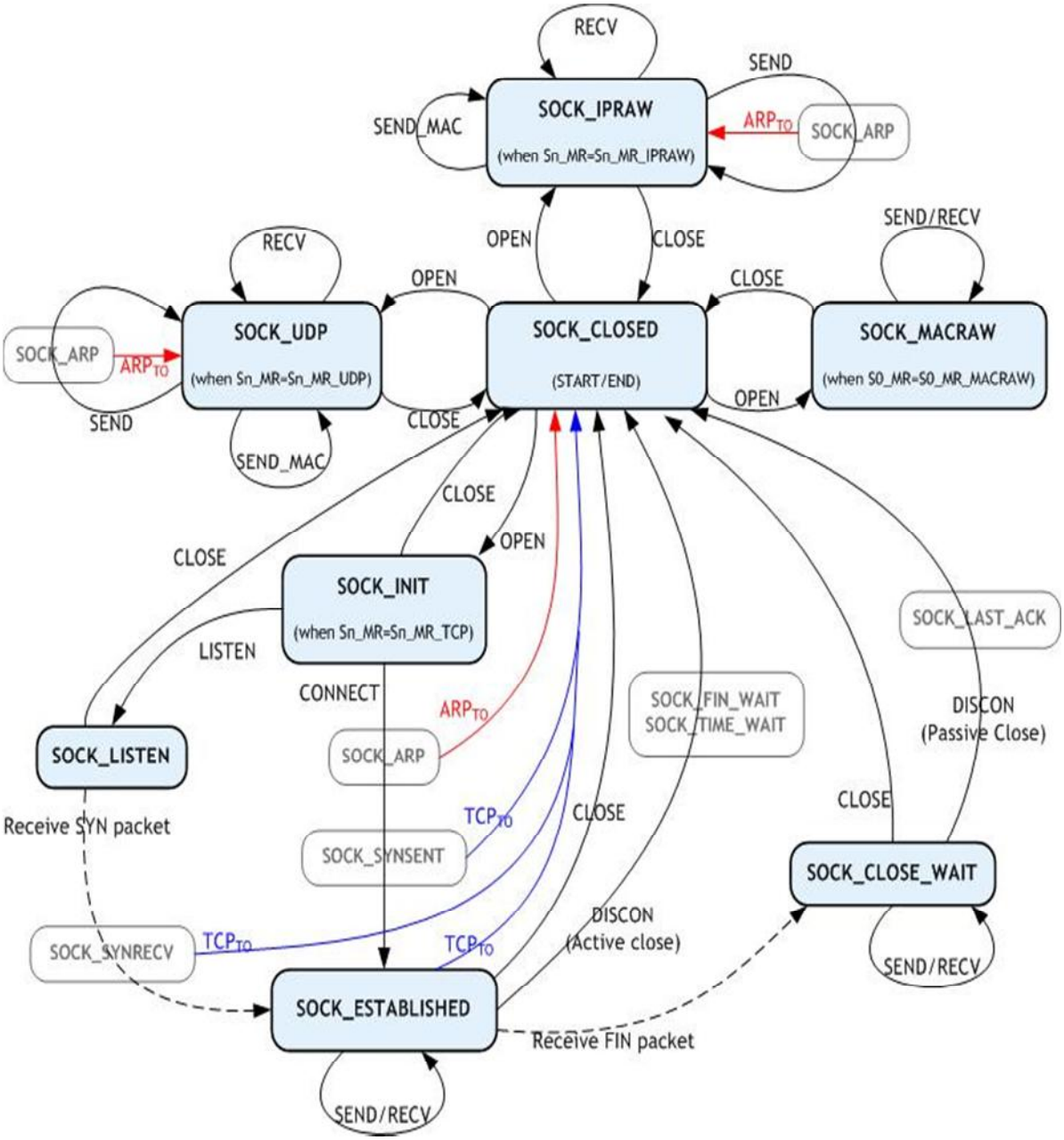


Figure 5 Socket Status Transition

S*n*_PORT (SOCKET n Source Port Register) [R/W] [0x4004+0x0n00-0x4005+0x0n00] [0x0000]

Sn_PORT는 source port number를 설정한다. SOCKETn을 TCP나 UDP mode로 사용할 때만 유효하며, 그 외 mode에서는 무시된다. OPEN Command 이전에 반드시 설정해야 한다.

Ex) In case of Socket 0 Port = 5000(0x1388), configure as below,

0x4004	0x4005
0x13	0x88

S*n*_DHAR (SOCKET n Destination Hardware Address Register) [R/W] [0x4006+0x0n00-0x400B+0x0n00] [0xFFFFFFFFFFFF]

Sn_DHAR은 SOCKET n의 Destination hardware address를 설정한다. 또한 SOCKET0이 PPPoE mode로 사용될 경우 S0_DHAR은 이미 알고 있는 PPPoE server hardware address로 설정한다.

UDP나 IPRAW mode에서 SEND_MAC command를 사용할 경우 SOCKETn의 Destination hardware address를 설정한다. 또한 TCP, UDP, IPRAW mode에서 Sn_DHAR은 CONNECT나 SEND command에 의한 ARP-process를 통해 획득한 Destination hardware address로 설정된다. Host는 CONNECT나 SEND command 성공 이후 Sn_DHAR을 통해 Destination hardware address를 알 수 있다.

PPPoE mode에서, W5200의 PPPoE-process를 이용할 경우 PPPoE server hardware address를 따로 설정할 필요는 없다. 하지만 W5200의 PPPoE-process를 이용하지 못하고 MACRAW mode로 PPPoE-process를 직접 구현하여 처리한 경우라 할지라도, PPPoE packet을 송수신하기 위해서는, 직접 구현한 PPPoE-process를 통해 획득한 PPPoE server hardware address, PPPoE server IP address, PPP session ID를 설정하고 MR(PPPoE)를 '1'로 반드시 설정한다. S0_DHAR은 이미 알고 있는 PPPoE server hardware address를 설정하며, OPEN command 이전에 설정한다. S0_DHAR을 통해 설정된 PPPoE server hardware address는 OPEN command 이후 PDHAR에 반영된다. 설정된 PPPoE information은 CLOSE command 이후에도 내부적으로 계속 유효하다.

Ex) In case of Socket 0 Destination Hardware address = 08.DC.00.01.02.10, configuration is as below,

0x4006	0x4007	0x4008	0x4009	0x400A	0x400B
0x08	0xDC	0x00	0x01	0x02	0x0A

S*n*_DIPR (SOCKET n Destination IP Address Register)[R/W][0x400C+0x0n00 0x400F+0x0n00] [0x00000000]

Sn_DIPR은 SOCKETn의 Destination IP address를 설정하거나 설정되며, SOCKET0이 PPPoE mode로 사용될 경우 S0_DIPR은 이미 알고 있는 PPPoE server IP address로 설정한다.

Sn_DIPR은 TCP, UDP, IPRAW, PPPoE mode에서만 유효하고, MACRAW mode에서는 무시된다. TCP mode에서, 'TCP CLIENT'로 동작할 경우 접속하기 위한 'TCP SERVER'의 IP address로 설정하고, CONNECT command 이전에 설정한다. 'TCP SERVER'로 동작할 경우 'TCP CLIENT'와 접속 성공 이후 내부적으로 'TCP CLIENT'의 IP address로 설정된다. UDP나 IPRAW mode에서는, Sn_DIPR은 UDP나 IP DATA packet 전송에 사용될 Destination IP address로 SEND나 SEND_MAC command 이전에 설정한다. PPPoE mode에서는, S0_DHAR과 같은 경우로 S0_DIPR는 이미 알고 있는 PPPoE server IP address를 설정한다.

Ex) In case of Socket 0 Destination IP address = 192.168.0.11, configure as below.

0x400C	0x040D	0x400E	0x040F
192 (0xC0)	168 (0xA8)	0 (0x00)	11 (0x0B)

S*n*_DPORT (SOCKET n Destination Port Register)[R/W][0x4010+0x0n00-0x4011+0x0n00] [0x00]

Sn_DIPR은 SOCKETn의 Destination port number를 설정한다. 만약 SOCKET0이 PPPoE mode로 사용되는 경우 S0_DPORTR은 이미 알고 있는 PPP Session ID로 설정한다.

Sn_DIPR은 TCP, UDP, PPPoE mode에서만 유효하고, 그 외의 mode에서는 무시된다. TCP mode에서, 'TCP

CLIENT'로 동작할 경우 접속하기 위한 'TCP SERVER'의 Listen port number로 설정하고, CONNECT command 이전에 설정한다. UDP mode에서, Sn_DPORTR은 UDP DATA packet 전송에 사용될 Port number로 SEND나 SEND_MAC command 이전에 설정한다. PPPoE mode에서는, S0_PDHAR과 같은 경우로 S0_DPORTR는 이미 알고 있는 PPP Session ID를 설정한다. S0_DPORTR을 통해 설정된 PPP Session ID는 OPEN command 이후 PSIDR에 반영된다.

Ex) In case of Socket 0 Destination Port = 5000(0x1388), configure as below,

0x4010	0x4011
0x13	0x88

S*n*_MSS (SOCKET n Maximum Segment Size Register)[R/W][0x4012+0x0n00-0x4013+0x0n00] [0x0000]

Sn_MSSR은 SOCKETn의 MTU(Maximum Transfer Unit)를 설정하거나, 설정된 MTU를 알려준다. Host가 Sn_MSSR를 설정하지 않을 경우는 Default MTU로 설정된다. TCP나 UDP mode만 지원하며, PPPoE를 사용할 경우(MR(PPPoE)='1') PPPoE의 MTU내에서 TCP나 UDP mode의 MTU가 결정된다.

Mode	**Normal** (MR(PPPoE)='0')		**PPPoE** (MR(PPPoE)='1')	
	Default MTU	Range	Default MTU	Range
TCP	1460	1 ~ 1460	1452	1 ~ 1452
UDP	1472	1 ~ 1472	1464	1 ~ 1464
IPRAW	1480		1472	
MACRAW	1514			

IPRAW나 MACRAW는 내부적으로 MTU를 처리하지 않고 Default MTU가 적용되므로, Host는 Default MTU보다 큰 Data를 전송할 때 Data를 Default MTU 단위로 직접(Manually) 나누어 전송해야 한다.
UDP mode에서는 TCP mode와 같은 Connection-process가 없고 Host-Written-Value를 그대로 사용한다. MTU가 서로 다른 상대방과 통신 할 경우, W5200은 ICMP(Fragment MTU) packet을 수신할 수 있다. 이 경우 IR(FMTU)='1'가 되고 Host는 FMTUR과 UIPR을 통해 Fragment MTU와 Destination IP address를 알 수 있다. IR(FMTU)='1'일 경우 그 상대방과는 UDP 통신이 불가능하므로, 해당 SOCKET을 close하고 알아낸 FMTU를 Sn_MSSR로 설정한 후 OPEN command로 open하여 다시 통신을 시도한다.

Ex) In case of Socket 0 MSS = 1460(0x05B4), configure as below,

0x4012	0x4013
0x05	0xB4

S*n*_PROTO (SOCKET n IP Protocol Register) [R/W] [0x4014+0x0n00] [0x00]

Sn_PROTO는 1 byte register로 IP layer에서 IP header의 Protocol number field를 설정한다. IPRAW mode에서만 유효하며, 그 외 mode는 무시된다. Sn_PROTOR은 OPEN command 이전에 설정한다. IPRAW mode로 Open된 SOCKETn은 Sn_PROTOR에 설정된 Protocol number의 Data만을 송수신한다. Sn_PROTOR은 0x00 ~ 0xFF 의 범위 내에서 설정 가능하나, W5200은 TCP(0x06), UDP(0x11) protocol number은 지원하지 않는다. Protocol number는 IANA (Internet Assigned Numbers Authority)에서 정의하고 있으며, 더 자세한 사항은 IANA의 online document (http://www.iana.org/assignments/protocol-numbers)를 참조하기 바란다

Ex) Internet Control Message Protocol (ICMP) = 0x01, Internet Group Management Protocol = 0x02

S*n*_TOS (SOCKET n IP Type Of Service Register) [R/W] [0x4015+0x0n00] [0x00]

Sn_TOS는 IP layer에서 IP header의 TOS(Type of service) field를 설정한다. Sn_TOS는 OPEN command 이전에 설정해야 한다. 자세한 사항은 http://www.iana.org/assignments/ip-parameters 참조하기 바란다.

S*n*_TTL (SOCKET n TTL Register) [R/W] [0x4016+0x0n00] [0x80]

Sn_TTL은 IP layer에서 IP header의 TTL(Time to live) field를 설정한다. Sn_TTL은 OPEN command 이전에 설정해야 한다. 자세한 사항은 http://www.iana.org/assignments/ip-parameters 참조하기 바란다.

Sn_RXMEM_SIZE (SOCKET n Receive Memory Size Register) [R/W] [0x401E+0x0n00] [0x02]

Sn_RXMEM_SIZE는 각 SOCKET의 RX memory size를 설정한다. 각 SOCKET의 RX memory 1, 2, 4, 8, 16Kbyte크기로 설정할 수 있다. Reset후에 초기값으로 2Kbyte의 값을 갖는다. $Sn\_RXMEM\_SIZE_{SUM}$(각 Sn_RXMEM_SIZE의 총 합)은 최대 16Kbyte를 넘을 수 없다.

Value	0x01	0x02	0x04	0x08	0x0F
Memory size	1KB	2KB	4KB	8KB	16KB

Ex1) SOCKET 0 : 8KB, SOCKET 1 : 2KB

0xFE401E	0xFE411E
0x08	0x02

Ex2) SOCKET 2 : 1KB, SOCKET 3 : 1KB

0xFE421E	0xFE431E
0x01	0x01

Ex3) SOCKET 4 : 1KB, SOCKET 5 : 1KB

0xFE441E	0xFE451E
0x01	0x01

Ex4) SOCKET 6 : 1KB, SOCKET 7 : 1KB

0xFE461E	0xFE471E
0x01	0x01

Sn_TXMEM_SIZE (SOCKET n IP Transmit Memory Size Register) [R/W] [0x401E+0x0n00] [0x02]

Sn_TXMEM_SIZE는 각 SOCKET의 TX memory size를 설정한다. 각 SOCKET의 TX memory 1, 2, 4, 8, 16Kbyte크기로 설정할 수 있다. Reset후에 초기값으로 2Kbyte의 값을 갖는다. $Sn\_TXMEM\_SIZE_{SUM}$(각 Sn_TXMEM_SIZE의 총 합)은 최대 16Kbyte를 넘을 수 없다.

Ex1) SOCKET 0 : 4KB, SOCKET 1 : 1KB

0xFE401F	0xFE411F
0x04	0x01

Ex2) SOCKET 2 : 2KB, SOCKET 3 : 1KB

0xFE421F	0xFE431F
0x02	0x01

Ex3) SOCKET 4 : 2KB, SOCKET 5 : 2KB

0xFE441F	0xFE451F
0x02	0x02

Ex4) SOCKET 6 : 2KB, SOCKET 7 : 2KB

0xFE461F	0xFE471F
0x02	0x02

Sn_TX_FSR (SOCKET n TX Free Size Register) [R] [0x4020+0x0n00-0x4021+0x0n00]

[0x0800]

Sn_TX_FSR은 SOCKET n의 Internal TX memory의 Free size(전송 가능한 데이터의 byte size)를 알려준다. HOST는 Sn_TX_FSR보다 크게 TX memory에 Data를 write하면 안 된다. 따라서 데이터 전송 전에 Sn_TX_FSR를 반드시 확인하고, 전송할 데이터의 크기가 Sn_TX_FSR보다 작거나 같으면 SEND나 SEND_MAC command로 데이터를 전송한다. TCP mode에서는 상대방으로부터 데이터 수신이 확인(DATA/ACK packet 수신)되면, Sn_TX_FSR은 상대방이 수신한 DATA packet 크기만큼 내부적으로 증가하게 된다. 그 외 mode에서는 Sn_IR(SENDOK) = '1'인 경우 Sn_TX_FSR은 전송한 Data size만큼 내부적으로 증가하게 된다.

이 register를 읽을 때, 사용자는 정확한 값을 얻기 위해 upper byte (0x4020, 0x4120, 0x4220, 0x4320, 0x4420, 0x4520, 0x4620, 0x4720) 들을 먼저 읽어야 하고 다음 lower byte (0x4021, 0x4121, 0x4221, 0x4321, 0x4421, 0x4521, 0x4621, 0x4721) 들을 읽어야 한다.

Ex) In case of 2048(0x0800) in S0_TX_FSR,

0x4020	0x4021
0x08	0x00

Sn_TX_RD (SOCKET n TX Read Pointer Register) [R] [0x4022+0x0n00-0x4023+0x0n00] [0x0000]

Sn_TX_RD는 TX memory에서 마지막 전송이 끝날 때 address를 알려준다. SOCKET n Command Register의 SEND 명령으로 현재 Sn_TX_RD부터 Sn_TX_WR까지 데이터를 전송한다. 전송이 끝나면 자동으로 그 값이 갱신된다. 따라서 전송이 끝나면 Sn_TX_RD와 Sn_TX_WR은 같은 값을 가질 것이다. 이 register를 읽을 때, 사용자는 정확한 값을 얻기 위해 upper byte (0x4022, 0x4122, 0x4222, 0x4322, 0x4422, 0x4522, 0x4622, 0x4722) 들을 먼저 읽어야 하고 다음 lower byte (0x4023, 0x4123, 0x4223, 0x4323, 0x4423, 0x4523, 0x4623, 0x4723) 들을 읽어야 한다.

Sn_TX_WR (SOCKET n TX Write Pointer Register) [R/W] [0x4024+0x0n00-0x4025+0x0n00] [0x0000]

Sn_TX_WR은 전송할 데이터가 write되어야 할 위치정보를 알려준다. 이 register를 읽을 때, 사용자는 정확한 값을 얻기 위해 upper byte (0x4024, 0x4124, 0x4224, 0x4324, 0x4424, 0x4524, 0x4624, 0x4724) 들을 먼저 읽어야 하고 다음 lower byte (0x4025, 0x4125, 0x4225, 0x4325, 0x4425, 0x4525, 0x4625, 0x4725) 들을 읽어야 한다

Caution: This register value is changed after the send command is successfully executed to Sn_CR.

Ex) In case of 2048(0x0800) in S0_TX_WR,

0x4024	0x4025
0x08	0x00

하지만 이 값은 write할 physical address가 아니므로, physical address는 다음과 같이 계산해야 한다.

1. Sn_TXMEM_SIZE(n)에서 SOCKET n TX Base Address (SBUFBASEADDRESS(n))와 SOCKET n TX Mask Address (SMASK(n))가 계산된다.
2. 위에서 계산한 두 값을 Bitwise-AND operation하고, SOCKET의 TX memory 범위에서

Sn_TX_WR과 SMASK(n)를 통해 offset address (dst_mask)를 얻는다.

3. Dst_mask와 SBUFBASEADDRESS(n)를 더해서 physical address (dst_ptr)를 얻는다.

이제, 전송할 데이터를 dst_ptr에 사용자가 원하는 만큼 write한다. (만약 SOCKET의 TX memory의 upper bound이상으로 데이터를 write하는 경우, TX memory의 upper bound만큼의 데이터를 먼저 write 한다. 그리고 난 다음 SBUFBASEADDRESS(*n*)에 physical address를 변경하여 나머지 데이터를 write한다. 그 후, Sn_TX_WR값을 writing 데이터 크기만큼 증가시킨다. 마지막으로 Sn_CR (SOCKET n Command Register)에 SEND명령을 수행한다.

Chip Base Address = 0x0000, 512(0x0200) bytes send

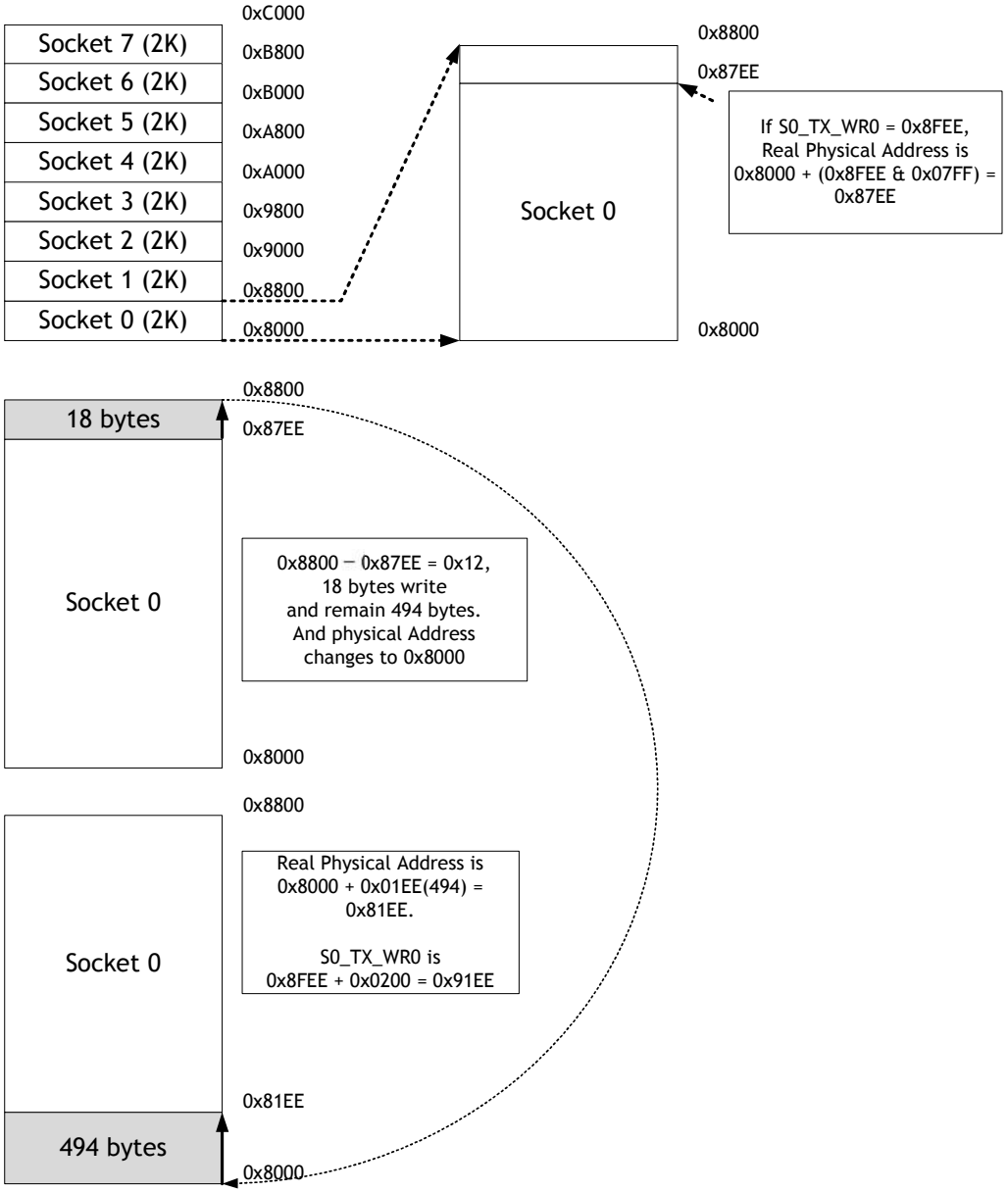


Figure 6 Physical Address Calculation

Sn_RX_RSR (RX Received Size Register) [R] [0x4026+0x0n00-0x4027+0x0n00] [0x0000]

Sn_RX_RSR은 SOCKETn의 Internal RX memory의 수신데이터 byte size를 알려준다. 이 값은 SOCKET n Command Register (Sn_CR) RECV 명령어에 의해 내부적으로 변하고 remote peer로 부터 데이터를 수신

한다. 이 register를 read할 때, 정확한 값을 얻기 위해 사용자는 상위 byte (0x4026, 0x4126, 0x4226, 0x4326, 0x4426, 0x4526, 0x4626, 0x4726)를 먼저 read 하고 그 다음 하위 byte (0x4027, 0x4127, 0x4227, 0x4327, 0x4427, 0x4527, 0x4627, 0x4727)를 read한다

Ex) In case of 2048(0x0800) in S0_RX_RSR,

0x4026	0x04027
0x08	0x00

The total size of this value can be decided according to the value of RX Memory Size Register.

S*n*_RX_RD (SOCKET n RX Read Pointer Register) [R/W] [0x4028+0x0n00-0x4029+0x0n00] [0x0000]

Sn_RX_RD는 수신 데이터를 read하기 위한 pointer의 위치 정보를 제공한다. 이 register를 read할 때, 사용자는 정확한 값을 read하기 위해 상위 byte (0x4028, 0x4128, 0x4228, 0x4328, 0x4428, 0x4528, 0x4628, 0x4728)를 먼저 read한 다음 하위 byte (0x4029, 0x4129, 0x4229, 0x4329, 0x4429, 0x4529, 0x4629, 0x4729)를 read해야 한다.

Ex) In case of 2048(0x0800) in S0_RX_RD,

0x4028	0x4029
0x08	0x00

하지만 이 값은 read할 physical address가 아니므로, physical address는 다음과 같이 계산해야 한다.

1. Sn_RXMEM_SIZE(n)에서 SOCKET n RX Base Address (RBUFBASEADDRESS(n))와 SOCKET n RX Mask Address (RMASK(n))가 계산된다.
2. 위에서 계산한 두 값을 bitwise-AND operation하고, SOCKET의 RX memory 범위에서 Sn_RX_WR과 RMASK(n)를 통해 offset address (src_mask)를 얻는다.
3. src_mask와 RBUFBASEADDRESS(n)를 더해서 physical address(src_ptr)를 얻는다.

이제, 수신 데이터를 src_ptr부터 사용자가 원하는 만큼 read한다. (만약 SOCKET의 RX memory의 upper bound이상으로 데이터를 read하는 경우, RX memory의 upper bound만큼의 데이터를 먼저 read한다. 그리고 난 다음 RBUFBASEADDRESS(*n*)에 physical address를 변경하여 나머지 데이터를 read한다. 그 후, Sn_RX_RD값을 read한 만큼 증가시킨다. (수신된 데이터 크기 이상으로 증가해서는 안되므로, 수신 전에 반드시 Sn_RX_RSR을 check해야 한다.) 마지막으로 Sn_CR (SOCKET n Command Register)에 RECV명령을 수행한다..

Sn_RX_WR (SOCKET n RX Write Pointer Register)[R/W][(0x402A+0xn00)-(0x402B+0x100n)][0x0000]

Sn_RX_WR은 수신 데이터를 write하기 위한 pointer의 위치 정보를 제공한다. 0x442A, 0x452A, 0x462A, 0x472A)를 먼저 read하고 난 다음 하위 byte (0x402B, 0x412B, 0x422B, 0x432B, 0x442B, 0x452B, 0x462B, 0x472B)를 read해야 한다.

Ex) In case of 2048(0x0800) in S0_RX_WR,

0x402A	0x402B
0x08	0x00

Sn_IMR (SOCKET n Interrupt Mask Register)[R/W][0x402C+0x0n00][0xFF]

Sn_IMR은 Host로 알려줄 SOCKET n의 Interrupt를 설정한다. Sn_IMR의 Interrupt Mask Bit들은 Sn_IR의 Interrupt Bit들과 각각 대응된다. 임의의 SOCKET Interrupt가 발생하고 Sn_IMR의 그 Bit가 '1'로 설정되어있을 경우 Sn_IR의 대응 Bit가 '1'로 설정된다. Sn_IMR과 Sn_IR의 임의 Bit가 모두 '1'일 때 IR(n) = '1'이 된다. 이때 IMR(n) = '1'이라면 Host에 Interrupt가 발생('/INT' signal low assert)한다)

7	6	5	4	3	2	1	0
PRECV	PFAIL	PNEXT	SEND_OK	TIMEOUT	RECV	DISCON	CON

Bit	Symbol	Description
7	PRECV	Sn_IR(PRECV) Interrupt Mask Valid only in case of 'SOCKET = 0' & 'S0_MR(P3:P0) = S0_MR_PPPoE'
6	PFAIL	Sn_IR(PFAIL) Interrupt Mask Valid only in case of 'SOCKET = 0' & 'S0_MR(P3:P0) = S0_MR_PPPoE'
5	PNEXT	Sn_IR(PNEXT) Interrupt Mask Valid only in case of 'SOCKET = 0' & 'S0_MR(P3:P0) = S0_MR_PPPoE'
4	SENDOK	Sn_IR(SENDOK) Interrupt Mask
3	TIMEOUT	Sn_IR(TIMEOUT) Interrupt Mask
2	RECV	Sn_IR(RECV) Interrupt Mask
1	DISCON	Sn_IR(DISCON) Interrupt Mask
0	CON	Sn_IR(CON) Interrupt Mask

Sn_FRAG (SOCKET n Fragment Register)[R/W][0x402D+0x0n00-0x402E+0x0n100][0x4000]

Sn_FRAG는 IP layer에서 IP header의 Fragment field를 설정한다. W5200은 IP layer의 packet fragment를 지원하지 않는다. 따라서 Sn_FRAG를 설정하더라도 IP data는 fragment되지 않으며 이를 설정하는 것은 권장하지 않는다. Sn_FRAG는 OPEN command 이전에 설정한다.

Ex) Sn_FRAG0 = 0x4000 (Don't Fragment)

0xFE402D	0xFE402E
0x40	0x00

5 Functional Descriptions

W5200은 Register와 Memory operation을 통해 간단히 Internet에 연결할 수 있다. 이 장에서는 W5200가 어떻게 동작되는 지를 설명한다.

5.1 Initialization

- **Basic Setting**

W5200의 동작을 위해 아래의 Register들을 사용자의 네트워크 환경에 알맞게 설정한다.

1. Mode Register (MR)
2. Interrupt Mask Register (IMR)
3. Retry Time-value Register (RTR)
4. Retry Count Register (RCR)

위의 Register들의 보다 자세한 내용은 Register Descriptions을 참조한다.

- **Setting network information**

통신을 위한 기본 Network 정보 설정:
다음의 기본적인 Network 정보를 반드시 설정해 주어야 한다.

1. SHAR(Source Hardware Address Register)

   SHAR에 의해 설정되는 Source hardware address는 모든 Device에 대해 유일한 Hardware address(Ethernet MAC address)값을 Ethernet MAC layer에서 사용하도록 정해져 있다. 이 MAC address의 할당은 EEE에서 관장하고 있으며, Network device를 생산하는 Manufacture는 생산된 Network device에 IEEE로부터 할당 받은 MAC address를 부여하여야 한다.
   http://www.ieee.org/, http://standards.ieee.org/regauth/oui/index.shtml를 참조하라

2. GAR(Gateway Address Register)

3. SUBR(Subnet Mask Register)

4. SIPR(Source IP Address Register)

➢ **Set socket memory information**

W5200의 설정 가능한 TX, RX의 최대 메모리 사이즈는 16Kbytes이다. 16Kbytes의 범위 안에서는 1KB, 2KB, 4KB, 8KB, 16KB의 크기로 8개의 소켓까지 자유롭게 설정이 가능하지만 TX, RX의 사이즈가 각각 16Kbyte를 넘어가서는 안 된다.

다음은 SOCKET n의 RX/TX Memory를 설정하는 의사코드를 예로 나타낸 것이다.

```
In case of, assign 2KB rx, tx memory per SOCKET
{
gS0_RX_BASE = 0x0000(Chip base address) + 0xC000(Internal RX buffer address); // Set base address of RX memory for SOCKET 0
Sn_RXMEM_SIZE(ch) = (uint8 *) 2;   // Assign 2K rx memory per SOCKET
gS0_RX_MASK = 2K - 1; // 0x07FF, for getting offset address within assigned SOCKET 0 RX memory
gS1_RX_BASE = gS0_RX_BASE + (gS0_RX_MASK + 1);
gS1_RX_MASK = 2K - 1;
gS2_RX_BASE = gS1_RX_BASE + (gS1_RX_MASK + 1);
gS2_RX_MASK = 2K - 1;
gS3_RX_BASE = gS2_RX_BASE + (gS2_RX_MASK + 1);
gS3_RX_MASK = 2K - 1;
gS4_RX_BASE = gS3_RX_BASE + (gS3_RX_MASK + 1);
gS4_RX_MASK = 2K - 1;
gS5_RX_BASE = gS4_RX_BASE + (gS4_RX_MASK + 1);
gS5_RX_MASK = 2K - 1;
gS6_RX_BASE = gS5_RX_BASE + (gS5_RX_MASK + 1);
gS6_RX_MASK = 2K - 1;
gS7_RX_BASE = gS6_RX_BASE + (gS6_RX_MASK + 1);
gS7_RX_MASK = 2K - 1;
gS0_TX_BASE = 0x0000(Chip base address) + 0x8000(InternalTX buffer address); // Set base address of TX memory for SOCKET 0
Sn_TXMEM_SIZE(ch) = (uint8 *) 2;   // Assign 2K rx memory per SOCKET
gS0_TX_MASK = 2K - 1;
/* Same method, set gS1_TX_BASE, gS1_TX_MASK, gS2_TX_BASE, gS2_TX_MASK, gS3_TX_BASE, gS3_TX_MASK, gS4_TX_BASE, gS4_TX_MASK, gS5_TX_BASE, gS5_TX_MASK, gS6_TX_BASE, gS6_tx_MASK, gS7_TX_BASE, gS7_TX_MASK */
}
```

Sn_TXMEM_SIZE(ch) = 2K,
Chip base address = 0x0000

	0xC000	
Socket 7	0xB800	gS7_TX_BASE = 0xB800 gS7_TX_MASK = 0x07FF
Socket 6	0xB000	gS6_TX_BASE = 0xB000 gS6_TX_MASK = 0x07FF
Socket 5	0xA800	gS5_TX_BASE = 0xA800 gS5_TX_MASK = 0x07FF
Socket 4	0xA000	gS4_TX_BASE = 0xA000 gS4_TX_MASK = 0x07FF
Socket 3	0x9800	gS3_TX_BASE = 0x9800 gS3_TX_MASK = 0x07FF
Socket 2	0x9000	gS2_TX_BASE = 0x9000 gS2_TX_MASK = 0x07FF
Socket 1	0x8800	gS1_TX_BASE = 0x8800 gS1_TX_MASK = 0x07FF
Socket 0	0x8000	gS0_TX_BASE = 0x8000 gS0_TX_MASK = 0x07FF

(a) TX memory

Sn_RXMEM_SIZE(ch) = 2K,
Chip base address = 0x0000

Socket 7	0xF800	gS7_RX_BASE = 0xF800 gS7_RX_MASK = 0x07FF
Socket 6	0xF000	gS6_RX_BASE = 0xF000 gS6_RX_MASK = 0x07FF
Socket 5	0xE800	gS5_RX_BASE = 0xE800 gS5_RX_MASK = 0x07FF
Socket 4	0xE000	gS4_RX_BASE = 0xE000 gS4_RX_MASK = 0x07FF
Socket 3	0xD800	gS3_RX_BASE = 0xD800 gS3_RX_MASK = 0x07FF
Socket 2	0xD000	gS2_RX_BASE = 0xD000 gS2_RX_MASK = 0x07FF
Socket 1	0xC800	gS1_RX_BASE = 0xC800 gS1_RX_MASK = 0x07FF
Socket 0	0xC000	gS0_RX_BASE = 0xC000 gS0_RX_MASK = 0x07FF

(b) RX memory

Figure 7 Allocation Internal TX/RX memory of SOCKET n

5.2 Data Communications

Initialization 과정 후 W5200은 TCP, UDP, IPRAW, MACRAW mode의 SOCKET을 open하여 상대방과 data를 송수신할 수 있게 된다. W5200은 독립적으로 동시에 사용 가능한 SOCKET을 총 8개까지 지원한다. 이 section에서는 각 mode에 따른 통신 방법에 대해서 설명한다.

5.2.1 TCP

TCP는 Connection-oriented protocol이다. TCP는 자신의 IP address와 Port number 그리고 상대방의 IP address와 Port number를 한 쌍으로 Connection SOCKET을 형성하게 되고, 형성된 Connection SOCKET을 통해 Data를 송수신한다. Connection SOCKET의 형성 방법에는 'TCP SERVER'와 'TCP CLIENT' 2가지가 있다. 이는 어디에서 connect-request(SYN packet)을 전송하느냐에 따라 구분할 수 있다.

'TCP SERVER'은 상대방의 connect-request 전송을 대기하며, 전송된 connect-request을 accept하여 Connection SOCKET을 형성한다(Passive-open).

'TCP CLIENT'는 자신이 connect-request를 상대방에게 전송하여 Connection SOCKET 형성을 먼저 요구한다(Active-open).

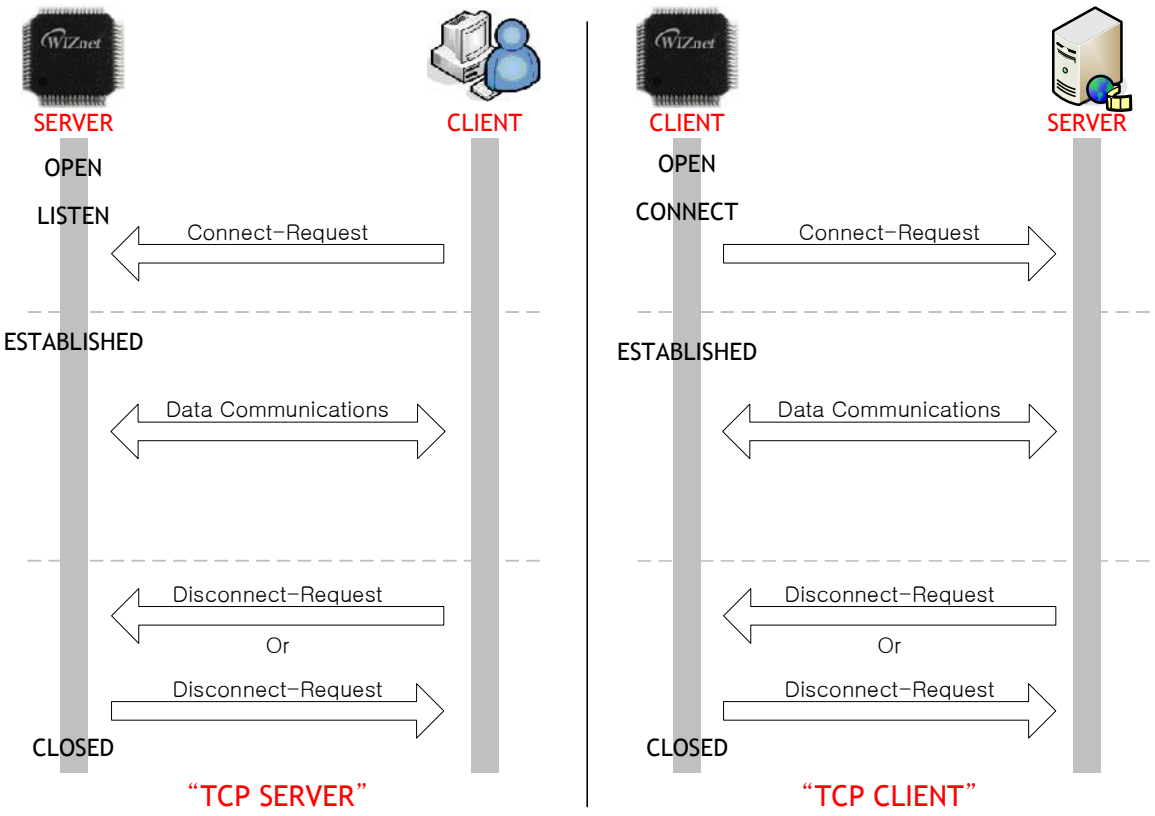


Figure 8 TCP SERVER and TCP CLIENT

5.2.1.1 TCP SERVER

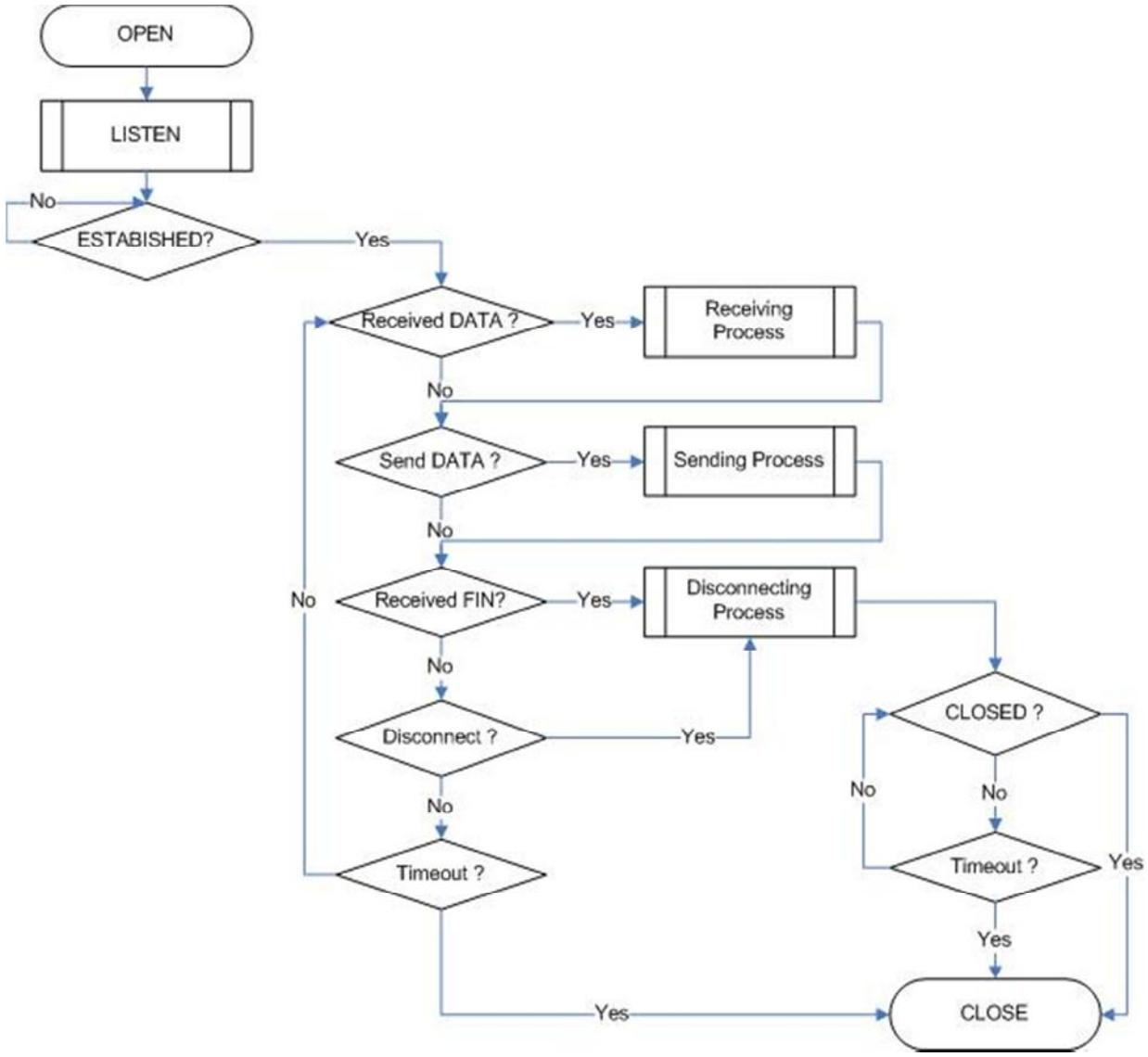


Figure 9 TCP SERVER Operation Flow

SOCKET Initialization

TCP Data communication을 위해 SOCKET Initialization 과정이 필요하다. 이는 SOCKET을 open하는 일이다. SOCKET open 과정은 W5200의 8개의 SOCKET 중 하나를 선택하여 선택된 SOCKET의 Protocol mode(Sn_MR(P3:P0))와 Source port number("TCP SERVER"에서는 Listen port number라고 함)인 Sn_PORT0을 설정한 후, OPEN command를 수행함으로써 이루어진다. OPEN command 이후 Sn_SR이 SOCK_INIT으로 변경되면 SOCKET initialization 과정은 완료된다. SOCKET initialization 과정은 "TCP SEVER"와 "TCP CLIENT"의 구분 없이 동일하게 적용된다. 다음은 SOCKET n을 TCP mode로 Open 과정이다

```
{
START:
   Sn_MR = 0x01;              // sets TCP mode
   Sn_PORT0 = source_port;          // sets source port number
   Sn_CR = OPEN;                    // sets OPEN command
   /* wait until Sn_SR is changed to SOCK_INIT */
   if (Sn_SR != SOCK_INIT) Sn_CR = CLOSE; goto START;
}
```

LISTEN

LISTEN command를 수행하여 "TCP SERVER"로 동작시킨다.

```
{
/* listen SOCKET */
Sn_CR = LISTEN;
/* wait until   Sn_SR is changed to SOCK_LISTEN */
    if (Sn_SR != SOCK_LISTEN) Sn_CR = CLOSE; goto START;
}
```

ESTABLISHMENT

Sn_SR이 SOCK_LISTEN일 때 상대방으로부터 SYN packet을 수신하게 되면 Sn_SR은 SOCK_SYNRECV로 변경되고 SYN/ACK packet을 전송 후 SOCKET n은 Connection이 형성되고 Sn_SR은 SOCK_ESTABLISHED로 바뀐다. SOCKET n의 Connection이 형성된 이후부터 data 송수신이 가능해진다. SOCKET n의 connection 형성을 확인하는 방법은 2가지가 있다

```
First method :
{
    if (Sn_IR(CON) == '1')   Sn_IR(CON) = '1'; goto ESTABLISHED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second method :
{
    if (Sn_SR == SOCK_ESTABLISHED) goto ESTABLISHED stage;
}
```

ESTABLISHMENT : Check received data

상대방으로부터의 TCP data 수신을 확인한다.

```
First method :
{
    if (Sn_IR(RECV) == '1') Sn_IR(RECV) = '1'; goto Receiving Process stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second Method :
{
    if (Sn_RX_RSR0 != 0x0000) goto Receiving Process stage;
}
```

First method는 매 수신 DATA packet 마다 Sn_IR(RECV)이 '1'로 설정된다. Host가 이전에 수신한 DATA packet의 Sn_IR(RECV)를 미처 처리 못하고 W5200이 다음 DATA packet을 수신할 경우, 이전 Sn_IR(RECV)에 중복 설정되어 Host는 그 다음의 수신 DATA packet에 대한 Sn_IR(RECV)를 인지할 수가 없게 된다. 따라서 Host가 각 Sn_IR(RECV)에 대한 DATA packet을 완벽하게 처리하지 못한다면 이 방법은 권장되지 않는다.

ESTABLISHMENT : Receiving process

이 과정에서는 내부 RX memory에 수신된 TCP 데이터를 처리한다. TCP mode에서 상대방이 전송한 Data 크기가 Socket n의 RX memory free size보다 클 경우 W5200은 그 data를 수신할 수 없으며, RX memory free size가 전송한 데이터 크기보다 클 때까지 connection을 유지한 체 기다린다.

```
{
/* first, get the received size */
    len = Sn_RX_RSR;       // len is received size
    /* calculate offset address */
    src_mask = Sn_RX_RD&gSn_RX_MASK;       // src_mask is offset address
    /* calculate start address(physical address) */
    src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address

    /* if overflow SOCKET RX memory */
    If((src_mask + len) > (gSn_RX_MASK + 1))
    {
       /* copy upper_size bytes of source_ptr to destination_address */
       upper_size = (gSn_RX_MASK + 1) - src_mask;
       memcpy(src_ptr, dst_ptr, upper_size);
       /* update destination_ptr */
       dst_address += upper_size;
       /* copy left_size bytes of gSn_RX_BASE to destination_address */
```

```
        left_size = len - upper_size;
        memcpy(gSn_RX_BASE, dst_address, left_size);
    }
    else
    {
        copy len bytes of source_ptr to destination_address */
        memcpy(src_ptr, dst_ptr, len);
    }
    /* increase Sn_RX_RD as length of len */
    Sn_RX_RD += len;
    /* set RECV command */
    Sn_CR = RECV;
}
```

ESTABLISHMENT: Check send data / Send process

전송할 data 크기는 할당된 Socket n의 Internal TX memory보다 클 수 없으며, 전송할 data 크기가 설정된 MSS보다 클 경우 MSS 단위로 나뉘어져 전송된다. 다음 data를 전송하기 위해선 반드시 이전의 SEND command가 완료되었는지 확인해야 한다. 이전 SEND command 완료 전에 다시 SEND command를 수행할 경우 오류가 발생할 수 있다. Data의 크기가 클수록 SEND command 완료 시간도 길어지므로, 전송 Data를 적정한 크기로 나누어 전송하는 것이 유리하다.

SEND command 수행 후 Data의 전송의 완료를 확인하기 위해서는 송신하려는 Data length와 실제 송신된 Data Length가 같은지 반드시 확인 해야 한다. 실제 송신된 Data Length는 SEND command 수행 전후의 Sn_TX_RD register 값의 차이로 계산 할 수 있다. 만약, 실제 송신된 Data Length의 값이 송신하려는 Data length 보다 작을 경우 SEND command를 이용하여 남겨진 Data를 모두 송신해야 한다. 따라서, 남겨진 Data기 모두 전송되어 실제 송신된 Data Length의 합의 값이 송신하려는 Data length와 같아야 send process가 정상적으로 완료된다. 아래에는 Send process의 간단한 예를 보여준다.

Ex) 보내려는 Data Length= 10,

1) Data Length를 이용하여 SEND Command 를 수행한다.
2) 실제 전송된 Data Length 값을 계산한다.
   만약에 실제 전송된 Data Length 7 (=Sn_TX_RD_after_SEND-Sn_TX_RD_befor_SEND) 이라면, 남겨진 Data Length는 3이된다.
3) 실제 전송된 Data Length 값의 합이 보내려는 Data Length 값과 일치 할 때까지 SEND Command를 반복하여 수행한다.
   Note: Return값의 합이 보내려는 Data Length 값과 일치 할 때까지 Data copy를 Data copy를 수행해서는 안 된다.

```
{
    /* first, get the free TX memory size */
FREESIZE:
```

```
    freesize = Sn_TX_FSR;
    if (freesize<len) goto FREESIZE;     // len is send size

    Sn_TX_RD
    /* calculate offset address */
    dst_mask= Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
    /* calculate start address(physical address) */
    dst_ptr = gSn_TX_BASE + dst_mask;       // destination_address is physical start address
    /* if overflow SOCKETTX memory */
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {
        /* copy upper_size bytes of source_addr to destination_address */
        upper_size = (gSn_TX_MASK + 1) - dst_mask;
        memcpy(src_addr, dst_ptr, upper_size);
        /* update source_addr*/
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = len - upper_size;
        memecpy(source_addr, gSn_TX_BASE, left_size);
    }
    else
    {
        /* copy len bytes of source_addr to destination_address */
        memcpy(source_addr, dst_ptr, len);
    }
/* increase Sn_TX_WR as length of len */
    Sn_TX_WR0 += send_size;
/* set SEND command */
Sn_CR = SEND;
/* return real packet size */
return ( read_ptr_after_send - read_ptr_befor_send )
/* if return value is not equal len (len is send size),
 retry send left data without copying data*/
}
```

ESTABLISHMENT : Check disconnect-request(FIN packet)

상대방으로부터 disconnect-request(FIN packet)를 수신했는지 확인한다. FIN packet 수신은 다음과 같이 확인할 수 있다.

```
First method :
{
    if (Sn_IR(DISCON) == '1') Sn_IR(DISCON)='1'; goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second method :
{
    if (Sn_SR == SOCK_CLOSE_WAIT) goto DISCONNECT stage;
}
```

ESTABLISHMENT : Check disconnect / disconnecting process

더 이상 상대방과의 data communication이 필요가 없는 경우나 상대방으로부터 FIN packet을 수신했을 경우는 connection SOCKET을 disconnect한다.

```
{
/* set DISCON command */
Sn_CR = DISCON;
}
```

ESTABLISHMENT : Check closed

DISCON이나 CLOSE command에 의해 Socket n이 Disconnect 혹은 close 되었는지 확인한다.

```
First method :
{
    if (Sn_IR(DISCON) == '1') goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second method :
{
    if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

ESTABLISHMENT: Timeout

Timeout은 connect-request(SYN packet)나 그것에 대한 응답(SYN/ACK packet), DATA packet이나 그것의 응답(DATA/ACK packet), disconnect-request(FIN packet)나 그것의 응답(FIN/ACK packet)등, 모든 TCP packet을 전송할 때 발생할 수 있다. RTR과 RCR에 설정된 Timeout 시간 동안 위 packet들을 전송하지 못하면 TCP final timeout(TCP_{TO})이 발생하게 되고 Sn_SR은 SOCK_CLOSED로 전이한다. TCP_{TO}의 확인은 다음과 같이 할 수 있다.

```
First method :
{
    if (Sn_IR(TIMEOUT bit) == '1') Sn_IR(TIMEOUT)='1'; goto CLOSED stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second method :
{
    if (Sn_SR == SOCK_CLOSED) goto CLOSED stage;
}
```

SOCKET Close

Disconnect-process에 의해 이미 disconnection된 SOCKET이나 TCP_{TO}에 의해 Close된 SOCKET을 완전히 close하거나, Host가 disconnect-process없이 필요에 의해 SOCKET을 close할 경우 사용할 수 있다.

```
{
    /* clear the remained interrupts of SOCKET n*/
Sn_IR = 0xFF;
    IR(n) = '1';
    /* set CLOSE command */
Sn_CR = CLOSE;
}
```

5.2.1.2 TCP CLIENT

TCP client는 CONNECT state를 제외한 모든 state가 TCP SEVER와 동일하다. 자세한 내용은 '52.1.1 TCP SERVER'를 참조하기 바란다.

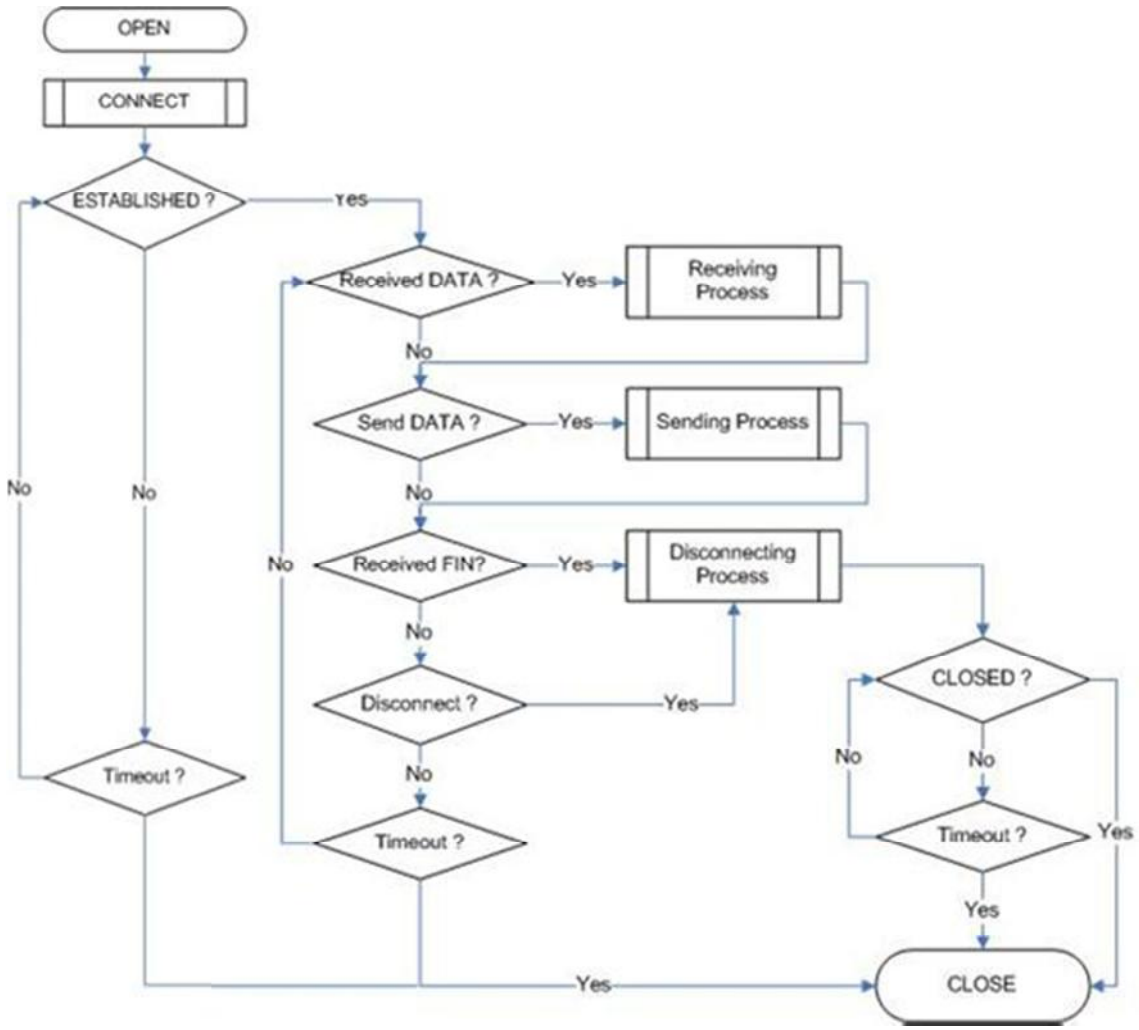


Figure 10 TCP CLIENT Operation Flow

CONNECT

'TCP SERVER'에게 connect-request (SYN packet)를 전송한다. 'TCP SERVER'와의 Connection SOCKET 형성 과정에서 ARP_{TO}, TCP_{TO}와 같은 Timeout이 발생할 수 있다.

```
{
   Sn_DIPR0 = server_ip;              /* set TCP SERVER IP address*/
   Sn_DPORT0 = server_port;           /* set TCP SERVER listen port number*/
   Sn_CR = CONNECT;                   /* set CONNECT command */
}
```

5.2.2 UDP

UDP는 Connection-less protocol이다. UDP는 TCP와 달리 Connection을 형성하지 않고 data를 송수신한다. TCP는 신뢰성 있는 data 통신을 보장하는 반면, UDP는 data 통신의 신뢰성을 보장하지 않는 datagram 통신을 하는 protocol이다. UDP는 connection을 사용하지 않기 때문에 자신의 IP address와 Port number를 알고 있는 많은 상대방과의 통신이 허락된다. 이와 같은 datagram 통신은 하나의 SOCKET을 이용하여 많은 상대방과 통신을 할 수 있는 이점이 있는 반면, 전송된 data의 손실이나 원치 않는 상대방으로부터의 data 수신과 같은 여러 문제가 발생할 수 있다. 이와 같은 문제를 해결하고 신뢰성을 보장하기 위해서, Host가 직접 손실된 data를 재전송하거나, 원치 않는 상대방으로부터의 수신 data를 무시해야 한다. UDP 통신은 unicast, broadcast, multicast 통신을 지원하며, 다음과 같은 통신 Flow를 따른다.

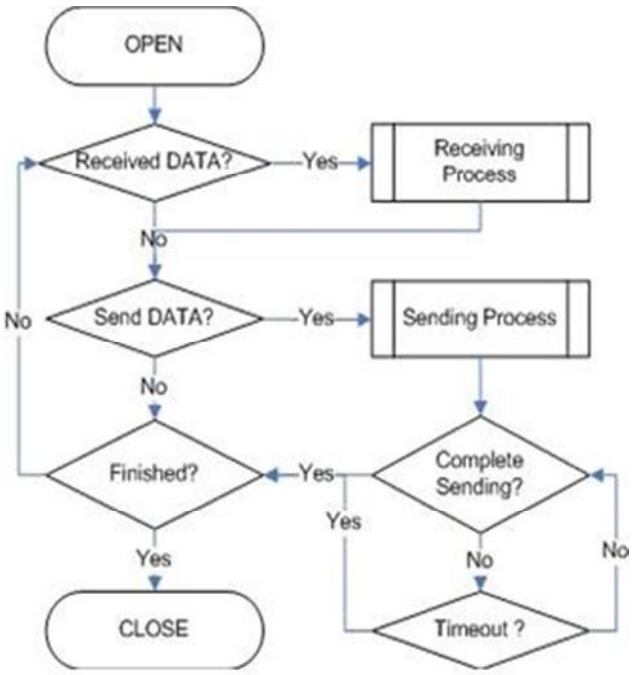


Figure 11 UDP Operation Flow

5.2.2.1 Unicast and Broadcast

Unicast 통신은 가장 일반적인 UDP 통신으로, 한번에 하나의 상대방에게 Data를 전송한다. 반면, Broadcast 통신은 Broadcasting IP address을 이용하여 한번의 통신으로 수신 가능한 모든 상대방에게 Data를 전달한다. 예로 A, B, C 에게 Data를 전송할 경우, Unicast 통신은 A, B, C 각각에 대해서 한번씩 Data를 전송을 한다. 이 때 A, B, C에 대한 Destination hardware address를 획득하는 과정(ARP-process)에서 ARP_{TO}가 발생할 수 있으며, ARP_{TO}가 발생한 상대방에게는 Data를 전송할 수가 없다.

Broadcast 통신은 Broadcasting IP address로 한번의 Data 전송을 통하여 A, B, C 모두에게 동시에 Data를 전달한다. 이 때 A, B, C에 대한 Destination hardware address를 획득할 필요가 없으며, ARP_{TO} 역시 발생하지 않는다

Note: Broadcast IP

=> HOST IP와 Subnet Mask의 보수를 OR 연산하면 Broadcast IP가 생성된다.

ex> IP: 222.98.173.123, Subnet Mask: 255.255.255.0 이면 broadcast IP: 222.98.173.255

Description	Decimal	Binary
HOST IP	222.098.173.123	11011110.01100010.10101101.01111011
Bit Complement Subnet mask	000.000.000.255	00000000.00000000.00000000.11111111
Bitwise OR	-	-
Broadcast IP	222.098.173.255	11011110.01100010.10101101.11111111

SOCKET Initialization

UDP data communication을 위해 SOCKET initialization 과정이 필요하다. 이는 SOCKET을 open하는 일이다. SOCKET open 과정은 W5200의 8개의 SOCKET 중 하나를 선택하고, 선택된 SOCKET의 protocol mode(Sn_MR(P3:P0))와 상대방과의 통신에 사용할 source port number인 Sn_PORT0을 설정한 후, OPEN command를 수행한다. OPEN command 이후 Sn_SR이 SOCK_UDP으로 변경되면 SOCKET initialization 과정은 완료된다.

```
{
START:
  Sn_MR = 0x02;                 /* sets UDP mode */
  Sn_PORT0 = source_port;       /* sets source port number */
  Sn_CR = OPEN;                 /* sets OPEN command */
  /* wait until Sn_SR is changed to SOCK_UDP */
  if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

Check received data

상대방으로부터의 UDP data 수신을 확인한다. TCP 통신과 동일한 방법으로 확인이 가능하다. 물론 TCP와 같은 이유로 Second method를 권장한다. “5.2.1.1 TCP SERVER”의 해당 절을 참조하라.

```
First method :
{
    if (Sn_IR(RECV) == ‘1’) Sn_IR(RECV) = ‘1’; goto Receiving Process stage;
    /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to IR, IMR
       Sn_IMR and Sn_IR. */
}
```

```
Second Method :
{
    if (Sn_RX_RSR0 != 0x0000) goto Receiving Process stage;
}
```

Receiving process

이 과정에서는 Internal RX memory에 수신된 UDP Data를 처리한다. 수신된 UDP data의 구조는 아래와 같다.

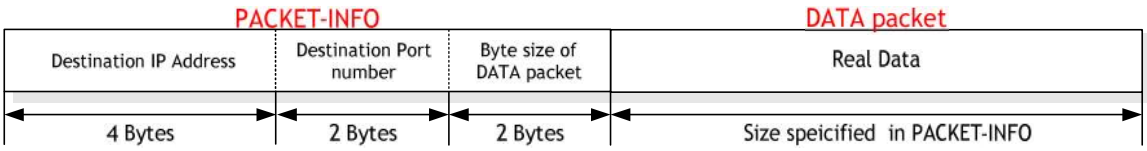


Figure 12 The Received UDP data Format

수신된 UDP data는 8bytes의 PACKET-INFO와 data packet으로 이루어지며, PACKET-INFO는 송신자의 정보(IP address, Port number)와 data packet의 길이가 포함된다. UDP는 많은 송신자로부터 UDP data를 수신할 수 가 있다. 송신자의 구분은 PACKET-INFO의 송신자 정보를 통해 확인할 수 있다. 송신자가 Broadcasting IP address를 이용하여 broadcast한 경우도 수신된다. Host는 PACKET-INFO의 송신자 정보를 분석하여 필요 없는 수신 data packet은 무시해야 한다. 송신자의 data 크기가 SOCKET의 Internal RX memory free size보다 클 경우 그 data는 수신할 수 없으며, 또한 fragment된 data 역시 수신할 수 없다.

```
{
    /* calculate offset address */
    src_mask = Sn_RX_RD &g Sn_RX_MASK;       // src_mask is offset address
    /* calculate start address(physical address) */
    src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address

    /* read head information (8 bytes) */
    header_size = 8;
    /* if overflow SOCKET RX memory */
    if ( (src_mask + header_size) > (gSn_RX_MASK + 1) )
    {
        /* copy upper_size bytes of src_ptr to header_addr*/
        upper_size = (gSn_RX_MASK + 1) - src_mask;
        memcpy(src_ptr, header, upper_size);
        /* update header_addr*/
        header_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to header_address */
        left_size = header_size - upper_size;
        memcpy(gSn_RX_BASE, header, left_size);
        /* update src_mask */
        src_mask = left_size;
    }
    else
    {
        /* copy header_size bytes of get_start_address to header_address */
        memcpy(src_ptr, header, header_size);
        /* update src_mask */
```

```
        src_mask += header_size;
    }
    /* update src_ptr */
    src_ptr = gSn_RX_BASE + src_mask;

    /* save remote peer information & received data size */
    peer_ip = header[0 to 3];
    peer_port = header[4 to 5];
    get_size = header[6 to 7];

    /* if overflow SOCKET RX memory */
    if ( (src_mask + get_size) > (gSn_RX_MASK + 1) )
    {
        /* copy upper_size bytes of src_ptr to destination_address */
        upper_size = (gSn_RX_MASK + 1) - src_mask;
        memcpy(src_ptr, destination_addr, upper_size);
        /* update destination_addr*/
        destination_addr += upper_size;
        /* copy left_size bytes of gSn_RX_BASE to destination_address */
        left_size = get_size - upper_size;
        memcpy(gSn_RX_BASE, destination_addr, left_size);
    }
    else
    {
        /* copy len bytes of src_ptr to destination_address */
        memcpy(src_ptr, destination_addr, get_size);
    }
/* increase Sn_RX_RD as length of len+ header_size */
    Sn_RX_RD = Sn_RX_RD + header_size + get_size;
/* set RECV command */
Sn_CR = RECV;
}
```

Check send data / sending process

전송할 data크기는 할당된 SOCKET의 internal TX memory보다 클 수 없으며, 전송할 data 크기가 MTU 보다 클 경우 MTU 단위로 자동으로 나누어져 전송된다. Broadcast할 경우에는 Sn_DIPR0을 Broadcasting IP address로 설정한다.

```
{
    /* first, get the free TX memory size */
FREESIZE:
    freesize = Sn_TX_FSR0;
    if (freesize<len) goto FREESIZE;       // len is send size

    /* Write the value of remote_ip, remote_port to the SOCKET n Destination IP Address
      Register(Sn_DIPR), SOCKET n Destination Port Register(Sn_DPORT). */
    Sn_DIPR0 = remote_ip;
    Sn_DPORT0 = remote_port;

    /* calculate offset address */
    dst_mask = Sn_TX_WR0 & gSn_TX_MASK;       // dst_mask is offset address
    /* calculate start address(physical address) */
    dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address

    /* if overflow SOCKETTX memory */
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {
        /* copy upper_size bytes of source_address to dst_ptr */
        upper_size = (gSn_TX_MASK + 1) - dst_mask;
        memcpy(src_ptr, destination_addr, upper_size);

        /* update source_address*/
        source_address += upper_size;
        /* copy left_size bytes of source_address to gSn_TX_BASE */
        left_size = send_size - upper_size;
        memcpy(src_ptr, destination_addr, left_size);
    else
    {
        /* copy len bytes of source_address to dst_ptr */
        memcpy(src_ptr, destination_addr, len);
    }
/* increase Sn_TX_WR0 as length of len */
    Sn_TX_WR0 += len;
```

```
    /* set SEND command */
Sn_CR = SEND;
}
```

Check complete sending / Timeout

다음 Data를 전송하기 위해선 반드시 이전 SEND command가 완료되었는지 확인해야 한다. Data의 크기가 클수록 SEND command 완료 시간도 길어지므로, 전송 Data를 적정한 크기로 나누어 전송하는 것이 유리하다. UDP data 전송 시 ARP_{TO}가 발생할 수 있고, ARP_{TO}가 발생할 경우 UDP data 전송은 실패한다.

```
First method :
{
  /* check SEND command completion */
  while(Sn_IR(SENDOK)==‘0’)   /* wait interrupt of SEND completion */
  {
    /* check ARPTO */
    if (Sn_IR(TIMEOUT)==‘1’) Sn_IR(TIMEOUT)=‘1’; goto Next stage;
  }
  Sn_IR(SENDOK) = ‘1’; /* clear previous interrupt of SEND completion */
}
```

```
Second method :
{
  If (Sn_CR == 0x00) transmission is completed.
  If (Sn_IR(TIMEOUT bit) == ‘1’) goto next stage;
  /* In this case, if the interrupt of SOCKET n is activated, interrupt occurs. Refer to Interrupt Register(IR), Interrupt Mask Register (IMR) and SOCKET n Interrupt Register (Sn_IR). */
}
```

Check Finished / SOCKET close

통신이 모두 끝난 경우 Socket *n*을 close한다.

```
{
    /* clear remained interrupts */
Sn_IR = 0x00FF;
    IR(n) = ‘1’;
    /* set CLOSE command */
Sn_CR = CLOSE;
}
```

5.2.2.2 Multicast

Broadcast 통신이 불특정 다수와 통신하는 반면, multicast 통신은 특정 multicast-group에 등록된 다수와 통신을 한다. A, B, C가 특정 multicast-group에 등록되어 있고, A가 등록된 multicast-group으로 data를 전송할 경우 B, C 역시 A의 전송 data를 수신할 수 있다. multicast 통신을 하기 위해선 IGMP protocol을 이용하여 multicast-group에 등록하여야 한다. multicast-group은 group hardware address, Group IP address, group port number로 구분된다. Group hardware address와 IP address는 이미 지정되어 있는 address를 사용하고, group port number는 임의로 사용할 수 있다.

Group hardware address는 지정 범위 ('01:00:5e:00:00:00'에서부터 '01:00:5e:7f:ff:ff') 내에서 선택되며, Group IP address는 D-class IP address 범위 ("224.0.0.0"에서 '239.255.255.255'까지, http://www.iana.org/assignments/multicast-addresses참조)내에서 선택된다. 이때 6bytes의 group hardware address와 4bytes의 IP address의 하위 23Bit는 같도록 선택해야 한다. 예로, Group IP address를 '224.1.1.11'로 선택할 경우 group hardware address는 '01:00:5e:01:01:0b'로 선택된다. 'RFC1112' 참조 (http://www.ietf.org/rfc.html).

W5200에서는 multicast-group 등록에 필요한 IGMP 처리는 내부적으로 (Automatically) 이루어진다. SOCKET n을 multicast mode로 open할 경우 IGMP의 'Join' message, close할 경우 'Leave' message가 내부적으로 전송된다. SOCKET open 이후 통신 시 주기적으로 'Report' message가 내부적으로 전송된다.

W5200은 IGMP version 1과 version 2만을 지원하며 상위 version을 사용하고자 한다면, IPRAW mode SOCKET을 이용하여 host가 직접 IGMP를 처리해야 한다.

SOCKET Initialization

Multicast 통신을 위해 8개의 SOCKET 중 하나를 선택하고, Sn_DHAR0을 multicast-group hardware address로 Sn_DIPR0을 multicast-group IP address로 설정한다. Sn_PORT0과 Sn_DPORT0을 multicast-group port number로 설정한다. Sn_MR(P3:P0)를 UDP로 Sn_MR (MULTI)를 '1'로 설정한 후 OPEN command를 수행한다. OPEN command 이후 Sn_SR이 SOCK_UDP으로 변경되면 SOCKET initialization 과정은 완료된다.

```
{
START:
    /* set Multicast-Group information */
    Sn_DHAR0 = 0x01;        /* set Multicast-Group H/W address(01:00:5e:01:01:0b) */
    Sn_DHAR1 = 0x00;
    Sn_DHAR2 = 0x5E;
    Sn_DHAR3 = 0x01;
    Sn_DHAR4 = 0x01;
    Sn_DHAR5 = 0x0B;
    Sn_DIPR0 = 211;         /* set Multicast-Group IP address(211.1.1.11) */
    Sn_DIPR1 = 1;
    Sn_DIPR2 = 1;
    Sn_DIRP3 = 11;
    Sn_DPORT0 = 0x0BB8;   /* set Multicast-GroupPort number(3000) */
```

```
    Sn_PORT0 = 0x0BB8;     /* set SourcePort number(3000) */
    Sn_MR = 0x02 | 0x80; /* set UDP mode & Multicast on SOCKET n Mode Register */

    Sn_CR = OPEN;            /* set OPEN command */

    /* wait until Sn_SR is changed to SOCK_UDP */
    if (Sn_SR != SOCK_UDP) Sn_CR = CLOSE; goto START;
}
```

Check received data

"5.2.2.1 Unicast & Broadcast." 참조

Receiving process

"5.2.2.1 Unicast & Broadcast." 참조

Check send data / Sending Process

SOCKET initialization에서 이미 multicast-group에 대한 정보를 설정하였으므로, unicast통신처럼 상대방의 IP address와 port number를 설정할 필요가 없다. 따라서 전송할 data를 internal TX memory로 copy 한 후 SEND command를 수행한다.

```
{
    /* first, get the free TX memory size */
FREESIZE:
    freesize = Sn_TX_FSR;
    if (freesize<len) goto FREESIZE;       // len is send size

    /* calculate offset address */
    dst_mask = Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
    /* calculate start address(physical address) */
    dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address
    /* if overflow SOCKETTX memory */
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {
        /* copy upper_size bytes of source_addr to destination_address */
        upper_size = (gSn_TX_MASK + 1) - dst_mask;
        wizmemcpy((0x000000 + source_addr), (0xFE0000 + dst_ptr), upper_size);
        /* update source_addr*/
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = len - upper_size;
```

```
        wizmemcpy( source_addr, gSn_TX_BASE, left_size);
    }
    else
    {
        /* copy len bytes of source_addr to dst_ptr */
        wizmemcpy( source_addr, dst_ptr, len);
    }
/* increase Sn_TX_WR as length of len */
    Sn_TX_WR0 += send_size;
/* set SEND command */
Sn_CR = SEND;
}
```

Check complete sending / Timeout

Data 통신에 필요한 모든 Protocol 처리는 Host가 관장하므로 Timeout은 발생하지 않는다.

```
{
  /* check SEND command completion */
  while(S0_IR(SENDOK)=='0');    /* wait interrupt of SEND completion */
  S0_IR(SENDOK) = '1';          /* clear previous interrupt of SEND completion */
}
```

Check finished / SOCKET close

"5.2.2.1 Unicast & Broadcast." 참조

5.2.3 IPRAW

IPRAW는 TCP와 UDP의 하위 protocol 계층인 IP layer를 이용한 Data 통신이다. IPRAW는 protocol number에 따라 ICMP(0x01), IGMP(0x02)와 같은 IP layer의 protocol을 지원한다. ICMP의 ping이나 IGMP v1/v2는 W5200에서 Hardware logic으로 이미 구현되어있다. 하지만 필요에 따라 Host는 Socket n을 IPRAW mode로 open하여 이를 직접 구현하여 처리할 수 있다. IPRAW mode SOCKET을 사용할 경우, 어떤 protocol을 사용할지 반드시 IP header의 protocol number field를 설정하여야 한다. Protocol number는 IANA에 의해 이미 정의되어 있다(http://www.iana.org/assignments/protocol-numbers 참조). Protocol number는 SOCKET Open 이전에 Sn_PROTO에 반드시 설정한다. W5200은 IPRAW mode에서 TCP(0x06)나 UDP(0x11) protocol number는 지원하지 않는다. IPRAW mode SOCKET의 통신은 지정된 protocol number만의 통신을 허용한다. ICMP로 설정된 SOCKET은 IGMP와 같이 설정되지 않은 그 외의 Protocol Data를 수신할 수 없다.

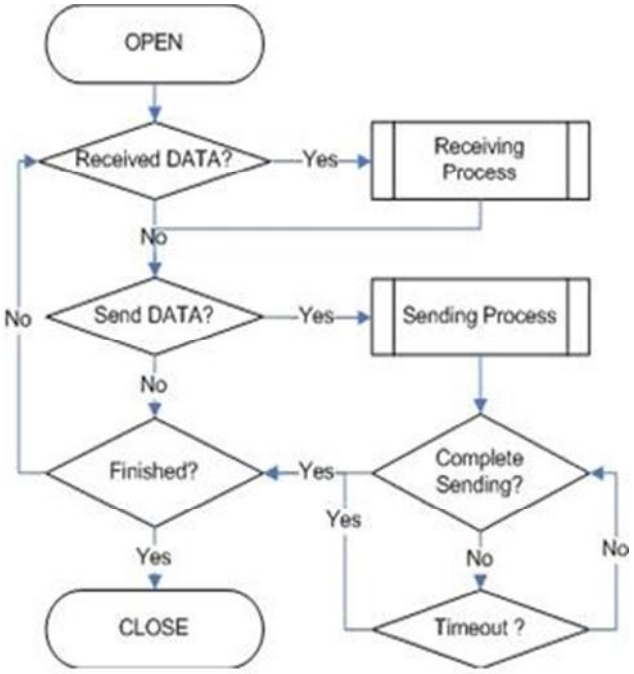


Figure 13 IPRAW Operation Flow

SOCKET Initialization

SOCKET을 선택하고 Protocol number를 설정한다. Sn_MR (P3:P0)를 IPRAW mode로 설정하고 OPEN command를 수행한다. OPEN command 이후 Sn_SR이 SOCK_IPRAW로 변경되면 SOCKET initialization 과정은 완료된다.

```
{
START:
/* sets Protocol number */
    /* The protocol number is used in Protocol Field of IP Header. */
    Sn_PROTO = protocol_num;
    /* sets IP raw mode */
```

```
    Sn_MR = 0x03;
/* sets OPEN command */
    Sn_CR = OPEN;
    /* wait until Sn_SR is changed to SOCK_IPRAW */
    if (Sn_SR != SOCK_IPRAW) Sn_CR = CLOSE; goto START;
}
```

Check received data

"5.2.2.1 Unicast & Broadcast." 참조

Receiving process

Internal RX Memory에 수신된 IPRAW Data를 처리한다. 수신된 IPRAW Data의 구조는 아래의 그림과 같다.

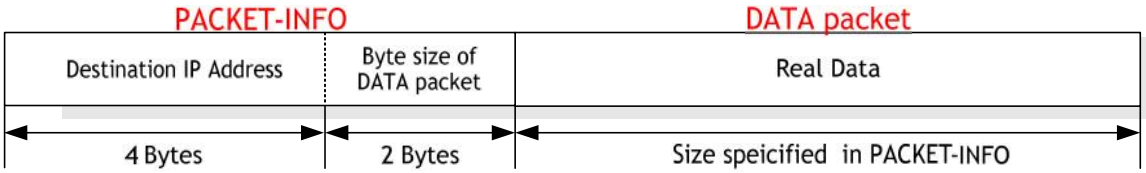


Figure 14 The receive IPRAW data Format

IPRAW data는 6 bytes의 PACKET-INFO와 data packet으로 이루어지며, PACKET-INFO는 송신자의 정보(IP address)와 data packet의 길이가 포함된다. IPRAW mode의 data 수신은 UDP의 PACKET-INFO에서 송신자의 port number 처리를 제외하고는 UDP data 수신과 모두 동일하다. '5.2.2.1 Unicast & Broadcast' 참조. 송신자의 Data 크기가 SOCKET n의 RX memory free size보다 클 경우 그 data는 수신할 수 없으며, 또한 fragmented data 역시 수신할 수 없다.

Checks send data / Sending process

전송할 data 크기는 할당된 SOCKET n의 internal TX memory보다 클 수 없고, default MTU보다 클 수 없다. IPRAW data 전송은 UDP data 전송에서 destination port number를 설정하는 것을 제외하고 모두 동일하다. 자세한 사항은 '5.2.2.1 Unicast & Broadcast'를 참조하기 바란다

Complete sending / Timeout

UDP와 동일, '5.2.2 UDP'참조.

Check finished / SOCKET closed

UDP와 동일, '5.2.2 UDP'참조

5.2.4 MACRAW

MACRAW 통신은 Ethernet MAC을 기반으로 그 상위 protocol을 host가 목적에 맞게 유연하게 사용할 수 있도록 하는 통신방법이다.

MACRAW mode는 오직 SOCKET 0만 사용 가능하다. SOCKET 0를 MACRAW로 사용할 경우 SOCKET 1에서 7

까지는 hardwired TCP/IP stack을 그대로 사용할 뿐만 아니라, SOCKET 0를 마치 NIC(Network Interface Controller)처럼 사용할 수 있어 software TCP/IP stack을 구현할 수 있다. 이와 같이 W5200은 hardwired TCP/IP와 software TCP/IP를 모두 구현할 수 있는 hybrid TCP/IP stack을 지원한다. W5200이 지원하는 8개의 SOCKET보다 더 많은 SOCKET들이 요구될 경우, 높은 성능을 요구하는 SOCKET들은 hardwired TCP/IP stack으로 구현하고, 그 외는 MACRAW mode를 이용하여 software TCP/IP로 구현하여 SOCKET 수의 한계를 극복할 수 있다. MACRAW mode의 SOCKET 0는 SOCKET 1에서 7까지 사용되고 있는 protocol들을 제외한 모든 protocol들을 처리할 수 있다. MACRAW 통신은 아무런 처리 없이 순수 Ethernet packet만의 통신이므로 MACRAW 설계자는 이러한 protocol을 분석하고 처리할 수 있는 software TCP/IP stack를 직접 구현해야 한다. MACRAW data는 Ethernet MAC을 기반으로 하기 때문에 6bytes의 source hardware address, 6bytes의 destination hardware address, 2 bytes의 Ethernet type 총 14bytes을 기본으로 포함해야 한다

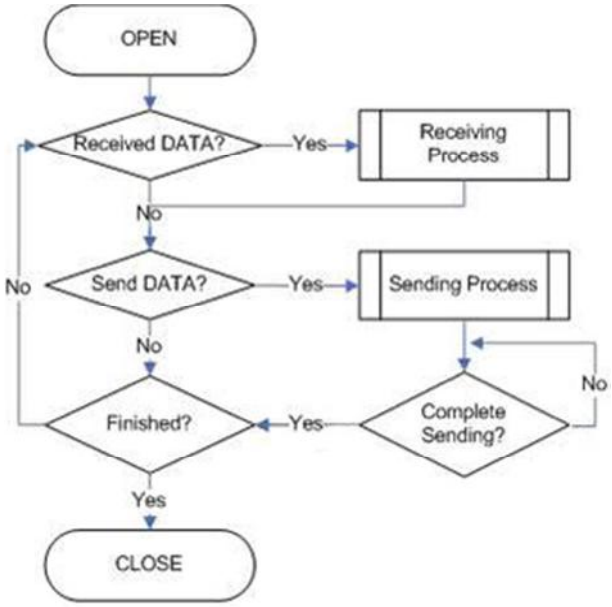


Figure 15 MACRAW Operation Flow

SOCKET Initialization

SOCKET을 선택하고 Sn_MR(P3:P0)를 MACRAW mode로 설정한 후 OPEN command를 수행한다. OPEN command 이후 Sn_SR이 SOCK_MACRAW로 변경되면 SOCKET initialization 과정은 완료된다. 이때 통신에 필요한 모든 정보 (Source hardware address, Source IP address, Source port number, Destination hardware address, Destination IP address, Destination port number, 각종 Protocol header, ETC)는 MACRAW Data의 일부분이므로 이와 관련된 별도의 register 설정은 필요 없다.

```
{
START:
    /* sets MAC raw mode */
    S0_MR = 0x04;
/* sets OPEN command */
    S0_CR = OPEN;
    /* wait until Sn_SR is changed to SOCK_MACRAW */
    if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
}
```

Check received data

'5.2.2.1 Unicast & Broadcast' 참조

Receiving process

SOCKET 0의 internal RX memory에 수신된 MACRAW data를 처리한다. MACRAW data의 구조는 Figure 17과 같다.

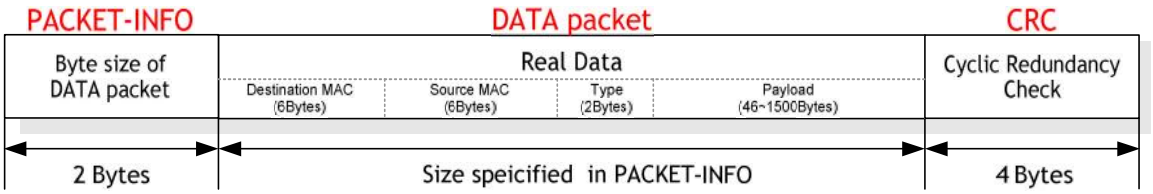


Figure 16 The received MACRAW data Format

MACRAW data는 2 bytes의 PACKET-INFO, data packet, 4bytes의 CRC로 이루어진다. PACKET-INFO는 data packet의 길이이며, data packet은 6bytes destination MAC address, 6bytes source MAC address, 2bytes type, 46~1500 bytes payload로 이루어진다. Data packet의 Payload는 Type에 따라 ARP, IP와 같은 Internet protocol로 이루어진다. Type에 관한 정보는 http://www.iana.org/assignments/ethernet-numbers를 참조하기 바란다.

```
{
    /* calculate offset address */
    src_mask = Sn_RX_RD & gSn_RX_MASK;       // src_mask is offset address
    /* calculate start address(physical address) */
    src_ptr = gSn_RX_BASE + src_mask;       // src_ptr is physical start address
```

```
    /* get the received size */
    len = get_byte_Size_Of_Data_packet // get byte size of DATA packet from Packet-INFO

    /* if overflow SOCKET RX memory */
    If((src_mask + len) > (gSn_RX_MASK + 1))
    {
    /* copy upper_size bytes of get_start_address to destination_address */
    upper_size = (gSn_RX_MASK + 1) - src_mask;
       memcpy(src_ptr, dst_addr, upper_size);
    /* update destination_address */
    dst_addr += upper_size;
    /* copy left_size bytes of gSn_RX_BASE to destination_address */
    left_size = len - upper_size;
     memcpy(src_ptr, dst_addr, left_size);
    }
    else
    {
    /* copy len bytes of src_ptr to destination_address */
    memcpy(src_ptr, dst_addr, len);
    }
    /*   increase Sn_RX_RD as length of len */
    Sn_RX_RD += len;
    /* extract 4 bytes CRC from internal RX memory and then ignore it */
    memcpy(src_ptr, dst_addr, len);
    /* set RECV command */
    Sn_CR = RECV;
}
```

<Notice>

Internal RX memory의 free size가 W5200이 수신해야 할 MACRAW data의 크기보다 작을 경우, 수신되어서는 안 되는 MACRAW data의 PACKET-INFO와 data packet의 일부가 internal RX memory에 저장되는 문제가 간혹 발생할 수 있다. 이는 상기 sample code에서 PACKET-INFO 분석의 오류를 야기시켜 올바른 MACRAW data 수신 처리를 할 수 없게 된다. 이 문제는 internal RX memory가 Full에 가까울수록 발생할 확률이 높아진다. 이 문제는 MACRAW data의 소실을 어느 정도 감안한다면 아래와 같이 해결할 수 있다.

- Internal RX memory의 처리를 최대한 빨리 하여 Full에 도달하는 것을 방지한다.
- SOCKET Initialization 과정의 Sample code에서 S0_MR의 MF(MAC Filter) Bit를 설정하여 자신에 해당하는 MACRAW data만을 수신하도록 하여 수신부하를 줄인다.

```
{
```

```
START:
/* sets MAC raw mode with enabling MAC filter */
S0_MR = 0x44;
/* sets OPEN command */
S0_CR = OPEN;
/* wait until Sn_SR is changed to SOCK_MACRAW */
if (Sn_SR != SOCK_MACRAW) S0_CR = CLOSE; goto START;
}
```

▪Internal RX memory의 free size가 1528 - default MTU(1514)+PACKET-INFO(2) + data packet(8) + CRC(4) - 보다 작을 경우 SOCKET0을 close한 후 지금까지 수신한 모든 MACRAW data를 처리하고 다시 SOCKET 0를 open하여 정상 처리한다. 이때 SOCKET 0 close이후 수신되는 MACRAW data는 손실될 수도 있다.

```
{
  /* check the free size of internal RX memory */
  if((Sn_RXMEM_SIZE(0) * 1024) - Sn_RX_RSR0(0) < 1528)
  {
      recved_size = Sn_RX_RSR0(0);          /* backup Sn_RX_RSR */
      Sn_CR0 = CLOSE;                /* SOCKET Closed */
      while(Sn_SR != SOCK_CLOSED);          /* wait until SOCKET is closed */
      /* process all data remained in internal RX memory */
    while(recved_size> 0)
     {/* calculate offset address */
          src_mask = Sn_RX_RD&gSn_RX_MASK;     // src_mask is offset address
          /* calculate start address(physical address) */
          src_ptr = gSn_RX_BASE + src_mask;    // src_ptr is physical start address
          /* if overflow SOCKET RX memory */
          If((src_mask + len) > (gSn_RX_MASK + 1))
          {
          /* copy upper_size bytes of get_start_address to destination_address */
            upper_size = (gSn_RX_MASK + 1) - src_mask;
            memcpy(src_ptr, dst_addr, upper_size);
            /* update destination_address */
            dst_address += upper_size;
            /* copy left_size bytes of gSn_RX_BASE to destination_address */
            left_size = len - upper_size;
            memcpy(src_ptr, dst_addr, left_size);
          }
          else
          {   /* copy len bytes of src_ptr to destination_address */
```

```
            memcpy(src_ptr, dst_addr, len);

        }
        /* increase Sn_RX_RD as length of len */
        Sn_RX_RD += len;
        /* extract 4 bytes CRC from internal RX memory and then ignore it */
        memcpy(src_ptr, dst_addr, len);
        /* calculate the size of remained data in internal RX memory*/
recved_size = recved_size - 2 - len - 4;
      }
     /* Reopen the SOCKET */
    /* sets MAC raw mode with enabling MAC filter */
      S0_MR = 0x44; /* or S0_MR = 0x04 */
  /* sets OPEN command */
      S0_CR = OPEN;
      /* wait until Sn_SR is changed to SOCK_MACRAW */
      while (Sn_SR != SOCK_MACRAW);
  }
  else /* process normally the DATA packet from internal RX memory */
  {/* This block is same as the code of "Receiving process" stage*/
  }
}
```

Check send data / sending process

전송할 Data 크기는 할당된 SOCKET 0의 internal TX memory보다 클 수 없고, 또한 default MTU보다 클 수 없다. Host는 'Receiving process'과정과 같이 data packet 형식과 동일한 MACRAW data를 생성하고 그 data를 전송한다. 이 때 생성된 data의 크기가 60 bytes 미만일 경우 실제 Ethernet으로 전송되는 packet은 내부적으로 60bytes가 되도록 'Zero padding'하여 전송한다.

```
{
    /* first, get the free TX memory size */
FREESIZE:
    freesize = S0_TX_FSR;
    if (freesize<send_size) goto FREESIZE;
    /* calculate offset address */
    dst_mask = Sn_TX_WR0 &gSn_TX_MASK;       // dst_mask is offset address
    /* calculate start address(physical address) */
    dst_ptr = gSn_TX_BASE + dst_mask;       // dst_ptr is physical start address

    /* if overflow SOCKETTX memory */
```

```
    if ( (dst_mask + len) > (gSn_TX_MASK + 1) )
    {/* copy upper_size bytes of source_addr to destination_address */
        upper_size = (gSn_TX_MASK + 1) - dst_mask;
        memcpy(src_ptr, dst_addr, upper_size);
        /* update source_addr*/
        source_addr += upper_size;
        /* copy left_size bytes of source_addr to gSn_TX_BASE */
        left_size = len - upper_size;
        memcpy(src_ptr, dst_addr, left_size);
    }
    else
    {/* copy len bytes of source_addr to destination_address */
        memcpy(src_ptr, dst_addr, len);
    }
/* increase Sn_TX_WR as length of len */
    Sn_TX_WR += send_size;

/* set SEND command */
    S0_CR = SEND;
}
```

Check complete sending

Data 통신에 필요한 모든 protocol 처리는 host가 관리하기 때문에 timeout은 발생하지 않지 않는다.

```
{
  /* check SEND command completion */
  while(S0_IR(SENDOK)==‘0’);    /* wait interrupt of SEND completion */
  S0_IR(SENDOK) = ‘1’;          /* clear previous interrupt of SEND completion */
}
```

Check finished / SOCKET close

‘5.2.2.1 Unicast & Broadcast’ 참조

6 External Interface

W5200은 외부 MCU와의 통신을 위해 SPI interface를 제공하고 있다.

6.1 SPI (Serial Peripheral Interface) mode

Serial Peripheral Interface Mode는 Data 통신을 위해 아래의 그림과 같이 4핀이 필요하다.

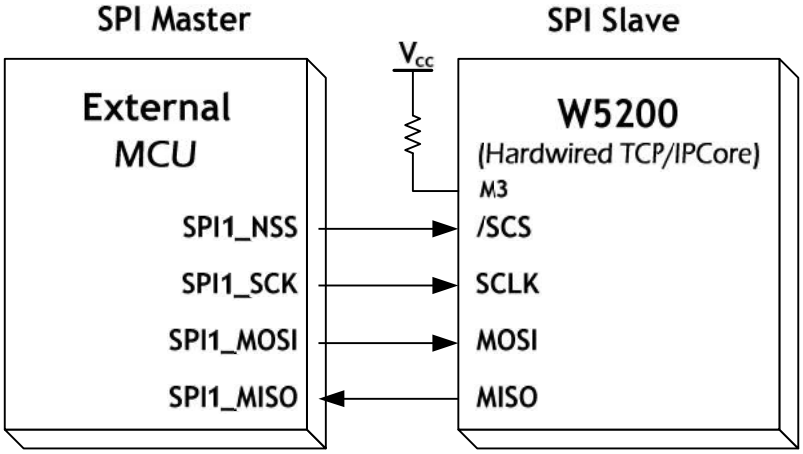


Figure 17 SPI Interface

6.2 Device Operations

W5200은 External Host로부터 보내진 Instruction Set에 의해 제어된다. W5200은 SPI Slave로 동작하며 External Host는 SPI Master로 동작하게 된다. SPI Masters는 W5200와 SPI bus인 Slave Chip Select (nSCS), Serial Clock (SCLK), MOSI (Master Out Slave In), MISO (Master In Slave Out)을 이용하여 통신한다.

SPI Protocol은 Mode 0-3, 4개의 모드로 정의 되어 있다. 각 모드는 SPI Clock의 극성과 위상에 따라 Data을 처리하는 방법이 달라진다. W5200은 SPI Slave모드로 동작하며 SPI mode 0 과 3을 지원한다. SPI Mode 0과 3의 inactive일 때 SCLK의 극성이 다르며 active시 동일하게 동작한다. SPI Mode가 0과 3일 경우 Data는 항상 SCLK의 Rising edge때 latch되며, Data의 출력은 Falling edge 일 때 이루어 진다.

6.3 Process of using general SPI Master device

1. Inactive일 경우 nSCS를 'High'로 설정한다.
2. SPI Master 디바이스의 SPI 관련 레지스터들을 설정한다.
3. nSCS를 'Low'로 설정한다. (data transfer start)
4. SPI Master 디바이스의 SPDR register에 전송할 Address를 Write한다.
5. SPI Master 디바이스의 SPDR register에 전송할 OP code / data length를 Write한다.
6. SPI Master 디바이스의 SPDR register (SPI Data Register)에 전송할 Data를 Write한다.
7. 수신이 완료될 때까지 기다린다.
8. 만약 모든 Data의 전송이 완료되면 nSCS를 'High'로 설정한다.

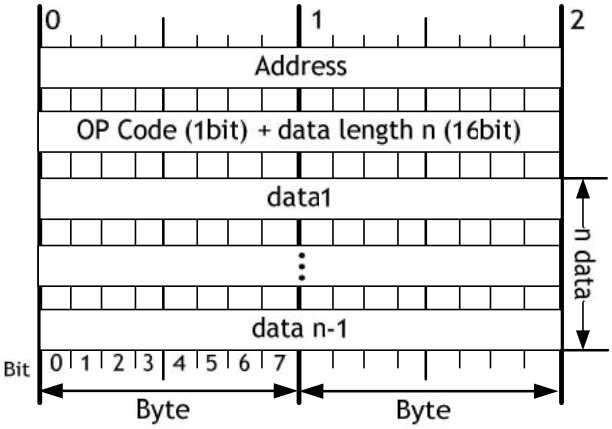


Figure 18 W5200 SPI Frame Format

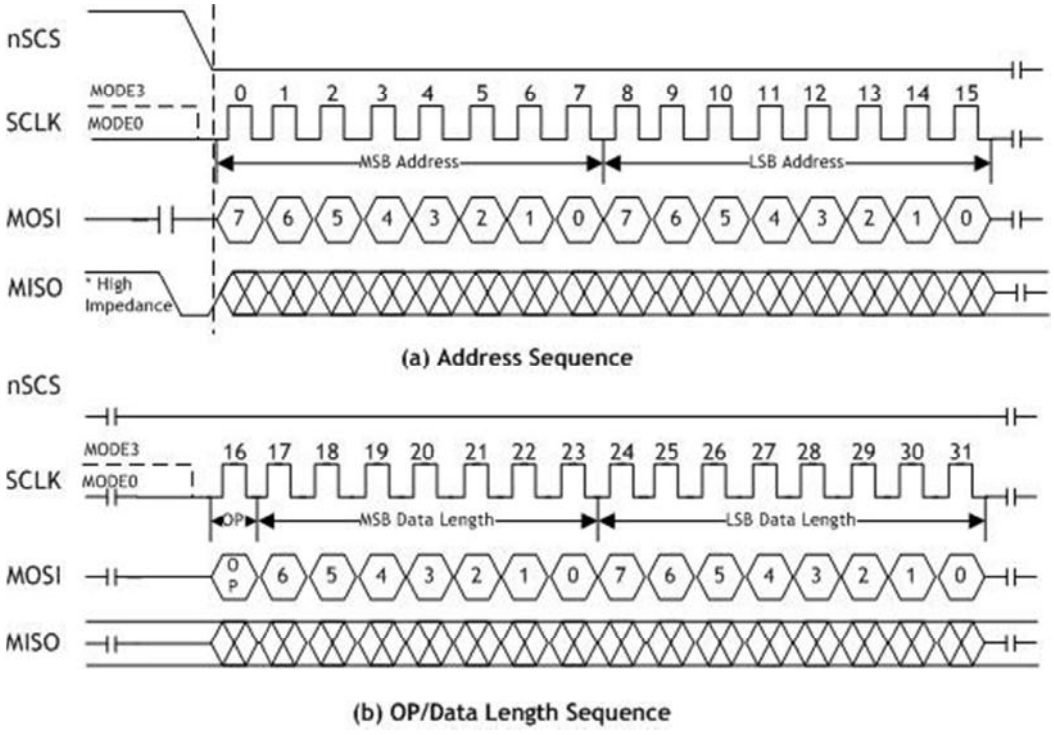


Figure 19 Address and OP/DATA Length Sequence Diagram

READ Processing

READ Processing Sequence Diagram은 그림 20과 같다. READ processing을 시작하기 위해 nSCS를 'low'로 driving 한 뒤, Address, OP Code, Data Length, Data를 차례대로 MOSI에 입력한다. Address, OP code, Data Length와 Data관련 자세한 Sequences는 그림 19을 참고한다. OP Code (OP)에는 READ OP와 WRITE OP이렇게 두 종류가 있다. OP 가 0일 때 read operation이 선택되며, OP 가 1일 경우 write operation이 선택된다.

W5200의 SPI는 byte 단위의 byte READ processing과 복수의 Data를 처리하는 burst READ processing이 제공된다. byte READ processing일 경우, 16-Bit Address, 1-Bit OP code(0x0), 15-Bit Data Length와 8-Bit Data이렇게 4 instruction이 필요하다. 반면에 Burst READ processing은 Data instruction만으로 수행된다. Data length로 byte READ processing과 Burst READ processing이 구분된다. 만약, Data length가 1일 경우 byte READ processing이 수행되며, Data length가 2 이상일 경우 Burst READ processing이 수행된다. MISO 핀은 반드시 nSCS의 Falling edge 이후 'low'로 Driving되어야 한다.

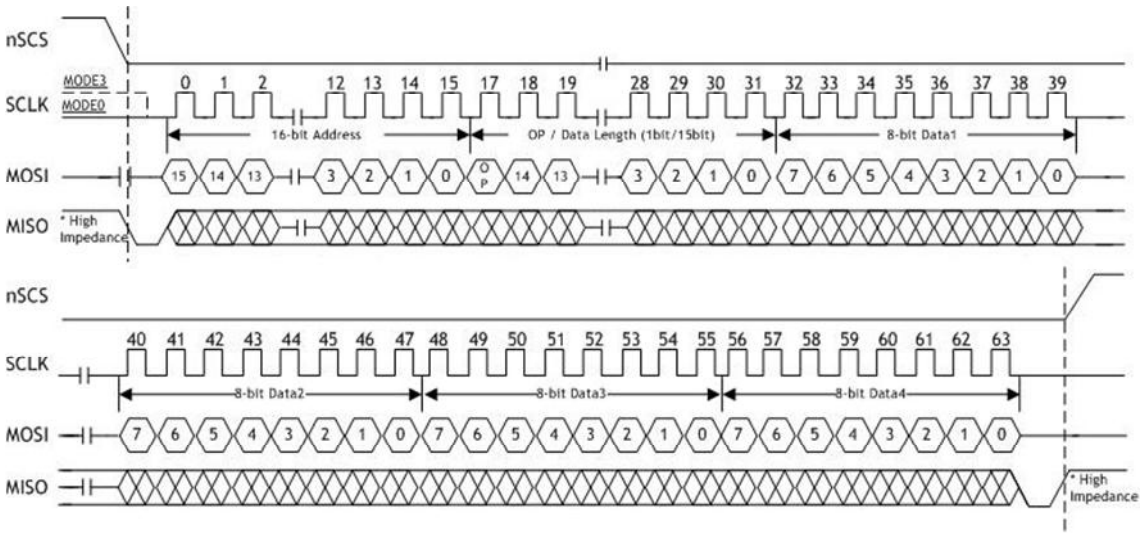


Figure 20 READ Sequence

```
/* Pseudo Code for Read data of 8bit per packet */
#define data_read_command      0x00
uint16 addr;      // Address : 16bits
int16 data_len;    // Data length :15bits
uint8 data_buf[];   // Array for data
SpiSendData();    // Send data from MCU to W5200
SpiRecvData();    // Receive data from W5200 to MCU

{
ISR_DISABLE();    // Interrupt Service Routine disable
CSoff();    // CS=0, SPI start
```

```
// SpiSendData
SpiSendData(((addr+idx) & 0xFF00) >> 8);     // Address byte 1
SpiSendData((addr+idx) & 0x00FF);            // Address byte 2

// Data write command + Data length upper 7bits
SpiSendData((data_read_command| ((data_len& 0x7F00) >> 8)));
// Data length bottom 8bits
SpiSendData((data_len& 0x00FF));

// Read data:On data_len> 1, Burst Read Processing Mode.
for(int idx = 0; idx<data_len; idx++)
{
    SpiSendData(0);   // Dummy data
    data_buf[idx] = SpiRecvData(idx);   // Read data
}
CSon();   // CS=1, SPI end
ISR_ENABLE();   // Interrupt Service Routine disable
}
```

WRITE Processing

Figure 21은 WRITE Processing Sequence Diagram을 보여준다. READ processing을 시작하기 위해 nSCS를 'low'로 driving 한 뒤, Address, OP Code, Data Length, Data를 차례대로 MISO에 입력한다.

W5200의 SPI는 byte 단위의 byte WRITE processing과 복수의 Data를 처리하는 burst WRITE processing이 제공된다. byte WRITE processing일 경우, 16-Bit Address, 1-Bit OP code (0x1), 15-Bit Data Length와 8-Bit Data이렇게 4 instruction로 구성된다. 반면에 Burst WRITE processing은 Data instruction만으로 수행된다. Data length로 byte READ processing과 Burst READ processing이 구분된다. 만약, Data length가 1일 경우 byte READ processing이 수행되며, Data length가 2 이상일 경우 Burst READ processing이 수행된다. MOSI 핀은 반드시 nSCS의 Falling edge 이후 'low'로 Driving되어야 한다

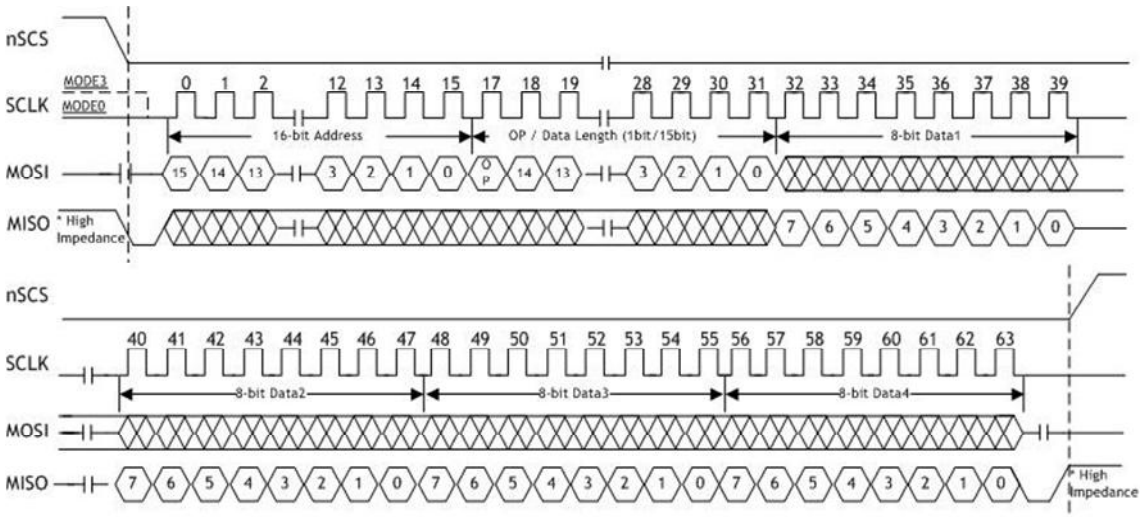


Figure 21 Write Sequence

```
/* Pseudo Code for Write data of 8bit per packet */
#define data_write_command      0x80
uint16 addr;      // Address : 16bits
int16 data_len;      // Data length :15bits
uint8 data_buf[];// Array for data

{
SpiSendData();    // Send data from MCU to W5200

ISR_DISABLE();   // Interrupt Service Routine disable
CSoff();   // CS=0, SPI start

SpiSendData(((addr+idx) & 0xFF00) >> 8);   // Address byte 1
SpiSendData((addr+idx) & 0x00FF);   // Address byte 2
```

```
// Data write command + data length upper 7bits
SpiSendData((data_write_command | ((data_len& 0x7F00) >> 8)));
// Data length bottom 8bits
SpiSendData((data_len& 0x00FF));

// Write data: On data_len> 1, Burst Write Processing Mode.
for(int idx = 0; idx<data_len; idx++)
    SpiSendData(data_buf[idx]);

CSon();// CS=1, SPI end
IINCHIP_ISR_ENABLE();// Interrupt Service Routine disable
}
```

7 Electrical Specifications

7.1 Absolute Maximum Ratings

Symbol	Parameter	Rating	Unit
V_{DD}	DC Supply voltage	-0.5 to 3.63	V
V_{IN}	DC input voltage	-0.5 to 5.5 (5V tolerant)	V
I_{IN}	DC input current	±5	mA
T_{OP}	Operating temperature	-40 to 85	°C
T_{STG}	Storage temperature	-55 to 125	°C

***COMMENT**: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage.

7.2 DC Characteristics

Symbol	Parameter	Test Condition	Min	Typ	Max	Unit
V_{DD}	DC Supply voltage	Junction temperature is from -55°C to 125°C	2.97		3.63	V
V_{IH}	High level input voltage		2.0		5.5	V
V_{IL}	Low level input voltage		- 0.3		0.8	V
V_{OH}	High level output voltage	I_{OH} = 4 ~8 mA	2.4			V
V_{OL}	Low level output voltage	I_{OL} = 4 ~8mA			0.4	V
I_I	Input Current	$V_{IN} = V_{DD}$			±5	μA

7.3 POWER DISSIPATION(Vcc 3.3V Temperature 25°C)

Condition	Min	Typ	Max	Unit
100M Link	-	160	175	mA
10M Link	-	110	125	mA
Un-Link	-	125	140	mA
100M Transmitting	-	160	175	mA
10M Transmitting	-	110	125	mA
Power Down mode	-	2	4	mA

7.4 AC Characteristics

7.4.1 Reset Timing

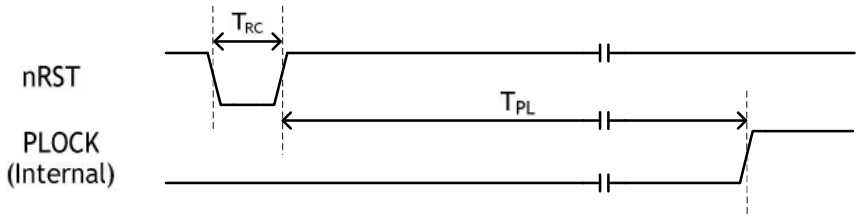


Figure 22 Reset Timing

Symbol	Description	Min	Max
T_{RC}	Reset Cycle Time	2 us	-
T_{PL}	nRST internal PLOCK	-	150 ms

7.4.2 Crystal Characteristics

Parameter	Range
Frequency	25 MHz
Frequency Tolerance (at 25℃)	±30 ppm
Shunt Capacitance	7pF Max
Drive Level	59.12uW/MHz
Load Capacitance	27pF
Aging (at 25℃)	±3ppm / year Max

7.4.3 SPI Timing

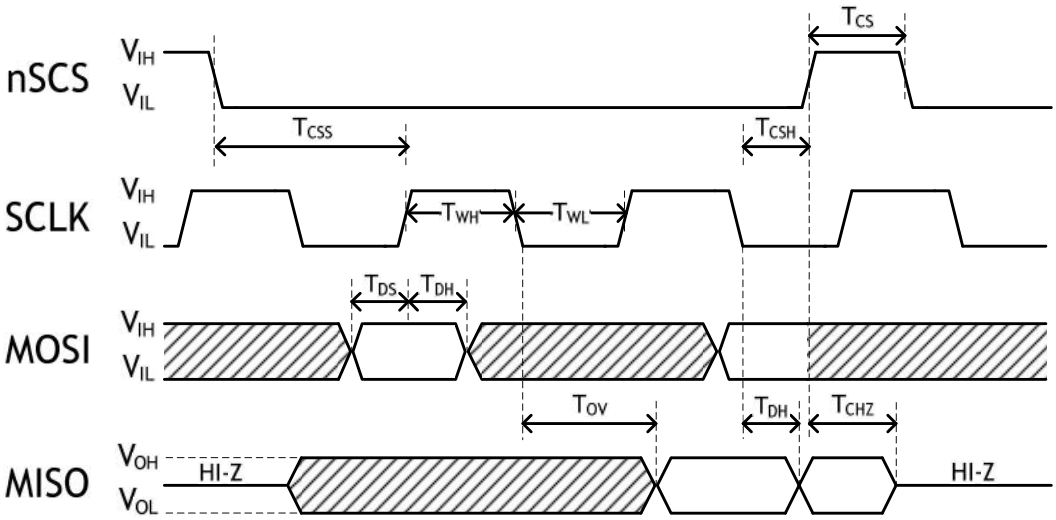


Figure 23 SPI Timing

Symbol	Description	Min	Max	Units
F_{SCK}	SCK Clock Frequency		80	MHz
T_{WH}	SCK High Time	6		ns
T_{WL}	SCK Low Time	6		ns
T_{CS}	nSCS High Time	5		ns
T_{CSS}	nSCS Hold Time	5	-	ns
T_{CSH}	nSCS Hold Time	5		ns
T_{DS}	Data In Setup Time	3		ns
T_{DH}	Data In Hold Time	3		ns
T_{OV}	Output Valid Time		5	ns
T_{OH}	Output Hold Time	0		ns
T_{CHZ}	nSCS High to Output Hi-Z		5	ns

7.4.4 Transformer Characteristics

Parameter	Transmit End	Receive End
Turn Ratio	1:1	1:1
Inductance	350 uH	350 uH

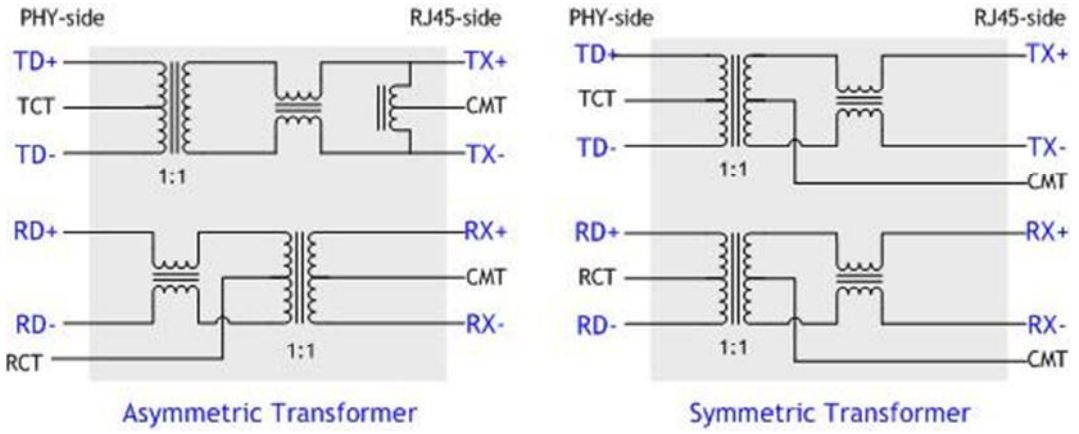


Figure 24 Transformer Type

내부 PHY모드를 사용하는 경우, Auto MDI/MDIX (Crossover)를 위해서 symmetric transformer를 사용해야 한다.

8 IR Reflow Temperature Profile (Lead-Free)

Moisture Sensitivity Level : 3

Dry Pack Required: Yes

Average Ramp-Up Rate (Ts_{max} to Tp)	3° C/second max.
Preheat	
- Temperature Min (Ts_{min})	150 °C
- Temperature Max (Ts_{max})	200 °C
- Time (ts_{min} to ts_{max})	60-120 seconds
Time maintained above:	
- Temperature (TL)	217 °C
- Time (tL)	60-150 seconds
Peak/Classification Temperature (Tp)	265 + 0/-5°C
Time within 5 °C of actual Peak Temperature (tp)	30 seconds
Ramp-Down Rate	6 °C/second max.
Time 25 °C to Peak Temperature	8 minutes max.

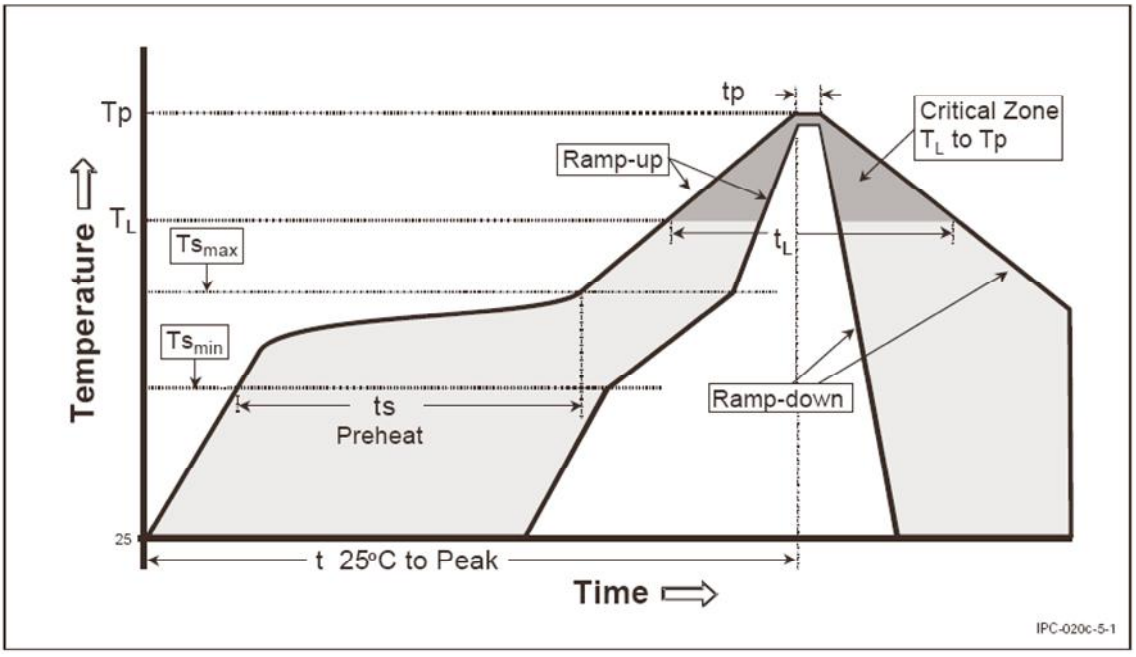
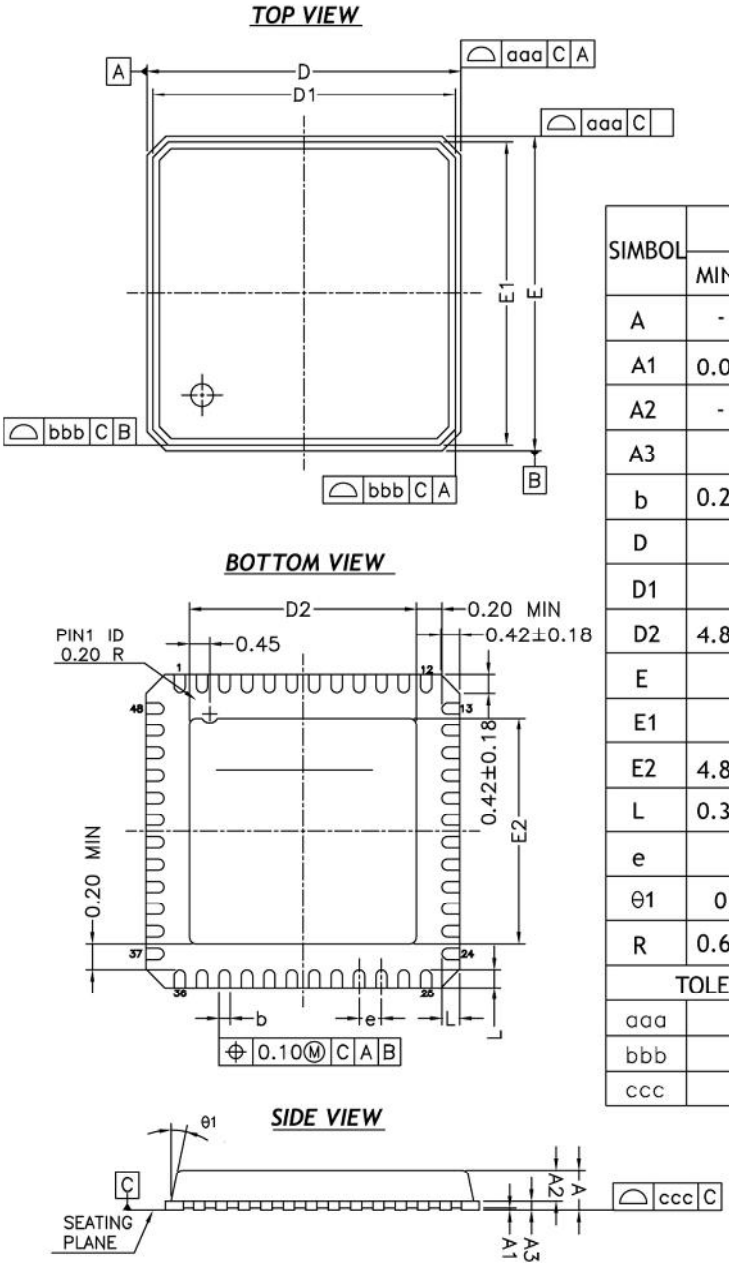


Figure 25 IR Reflow Temperature

9 Package Descriptions



SIMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	-	-	0.90	-	-	0.035
A1	0.00	0.01	0.05	0.00	0.0004	0.02
A2	-	0.65	0.70	-	0.26	0.028
A3	0.20 REF.			0.008 REF.		
b	0.20	0.25	0.32	0.008	0.010	0.013
D	7.00 bsc			0.276 bsc		
D1	6.75 bsc			0.266 bsc		
D2	4.86	5.06	5.26	0.191	0.199	0.207
E	7.00 bsc			0.276 bsc		
E1	6.75 bsc			0.266 bsc		
E2	4.86	5.06	5.26	0.191	0.199	0.207
L	0.30	0.40	0.50	0.012	0.016	0.020
e	0.50 bsc			0.020bsc		
θ1	0°	-	12°	0°	-	12°
R	0.65	-	-	0.028	-	-
TOLERANCES OF FORM AND POSITION						
aaa	0.10			0.004		
bbb	0.10			0.004		
ccc	0.05			0.002		

Figure 26 Package Dimensions

Note:

1. All dimensions are in millimeters.
2. Die thickness allowable is 0.0304 mm MAXMUM(0.012 Inches MAXIMUM)
3. Dimension & tolerances conform to same Y14.5M. -1994.
4. Dimension applies to plated terminal and is measured between 0.20 and 0.25mm from terminal tip.
5. The pin #1 identifier must be placed on the top surface of the package by using indentation mark or other feature of package body.
6. Exact shape and size of this feature is optional.

7. Package warpage max 0.08 mm.
8. Applied for exposed pad and terminals. Exclude embedding part of exposed pad from measuring
9. Applied only to terminals
10. Package corners unless otherwise specified are R0.175+/- 0.025mm

Document History Information

Version	Date	Descriptions
Ver. 1.0	MAR2011	Released with W5200 Launching
Ver. 1.1	13APR2011	Changed IMR address (0x16 to 0x36) - (P.14, P.18) Changed IMR2 address (0x36 to 0x16) - (P.14, P.22)
Ver. 1.2	22ARP2011	Fixed the description of RSV at 1.3 Miscellaneous Signals (P.10) Fixed the values of typical at 7.3 power dissipation (P.75) Added the values of maximum at 7.3 power dissipation (P.75) Fixed the description of RSV at 1.3 Miscellaneous Signals (removed PIN 31, P.10)
Ver. 1.21	2AUG2011	Fixed the description of READ processing at 6.3 Processing of using general SPI Master device (P.70)