iEthernet W5200 Datasheet

Version 1.2.6

http://www.wiznet.co.kr

W5200

The W5200 chip is a Hardwired TCP/IP embedded Ethernet controller that enables easier internet connection for embedded systems using SPI (Serial Peripheral Interface). W5200 suits best for users who need Internet connectivity for application that uses a single chip to implement TCP/IP Stack, 10/100 Ethernet MAC and PHY.

The W5200 is composed of a fully hardwired market-proven TCP/IP stack and an integrated Ethernet MAC & PHY. Hardwired TCP/IP stack supports TCP, UDP, IPv4, ICMP, ARP, IGMP, and PPPoE, which has been proven in various applications for many years. W5200 uses a 32Kbytes internal buffer as its data communication memory. By using W5200, users can implement the Ethernet application they need by using a simple socket program instead of handling a complex Ethernet Controller.

SPI (Serial Peripheral Interface) is provided for easy integration with the external MCU. The W5200 SPI supports a high speed SPI capable of communicating over SPI at up to 80MHz.In order to reduce power consumption of the system, W5200 provides WOL (Wake on LAN) and power down mode. To wake up during WOL, W5200 should be received magic packet, which is the Raw Ethernet packet.

Features

- Support Hardwired TCP/IP Protocols : TCP, UDP, ICMP, IPv4 ARP, IGMP, PPPoE, Ethernet
- 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

The W5200 is well suited for many embedded applications, including:

- 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

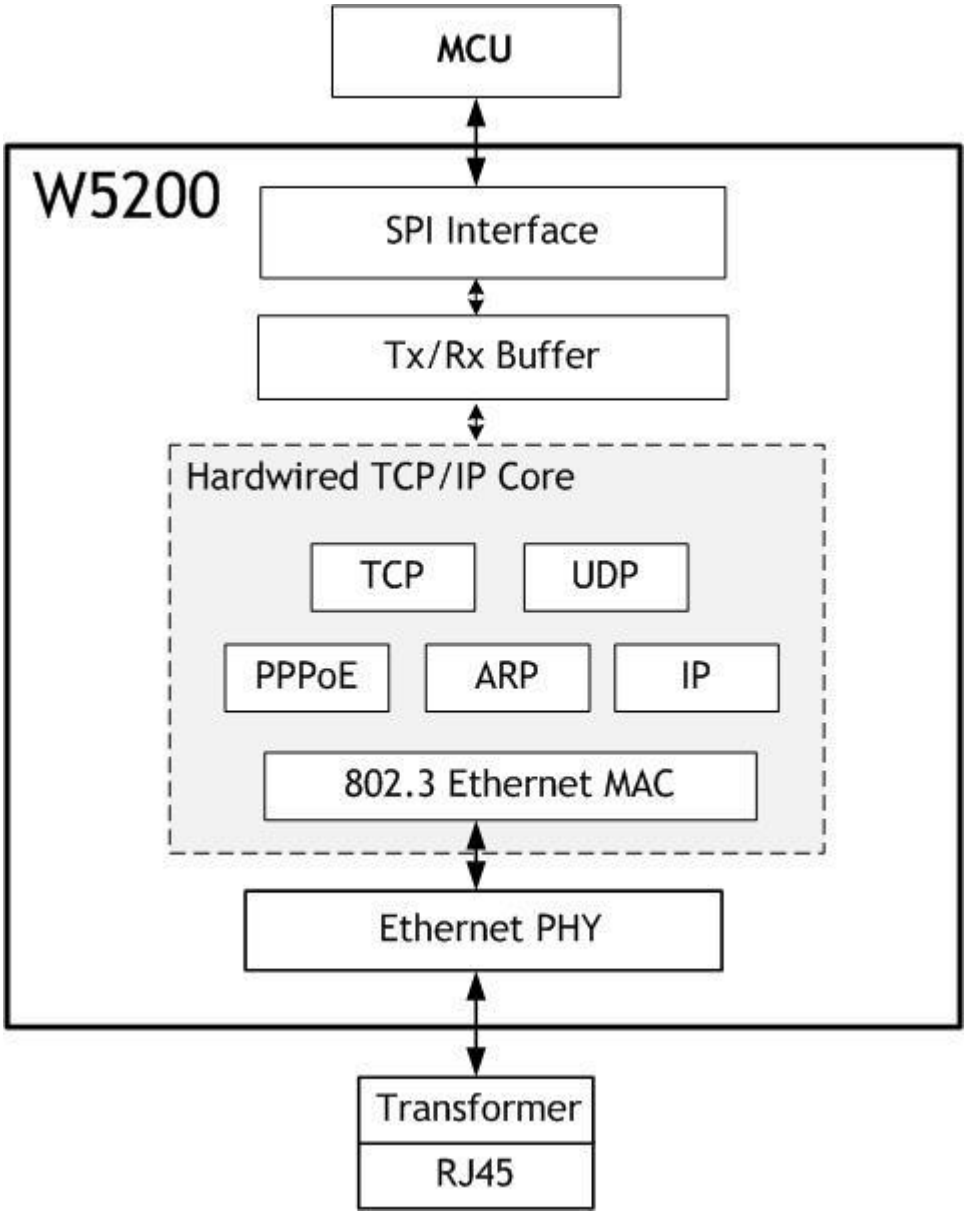


Table of Contents

Table of Figure

1 Pin Assignment

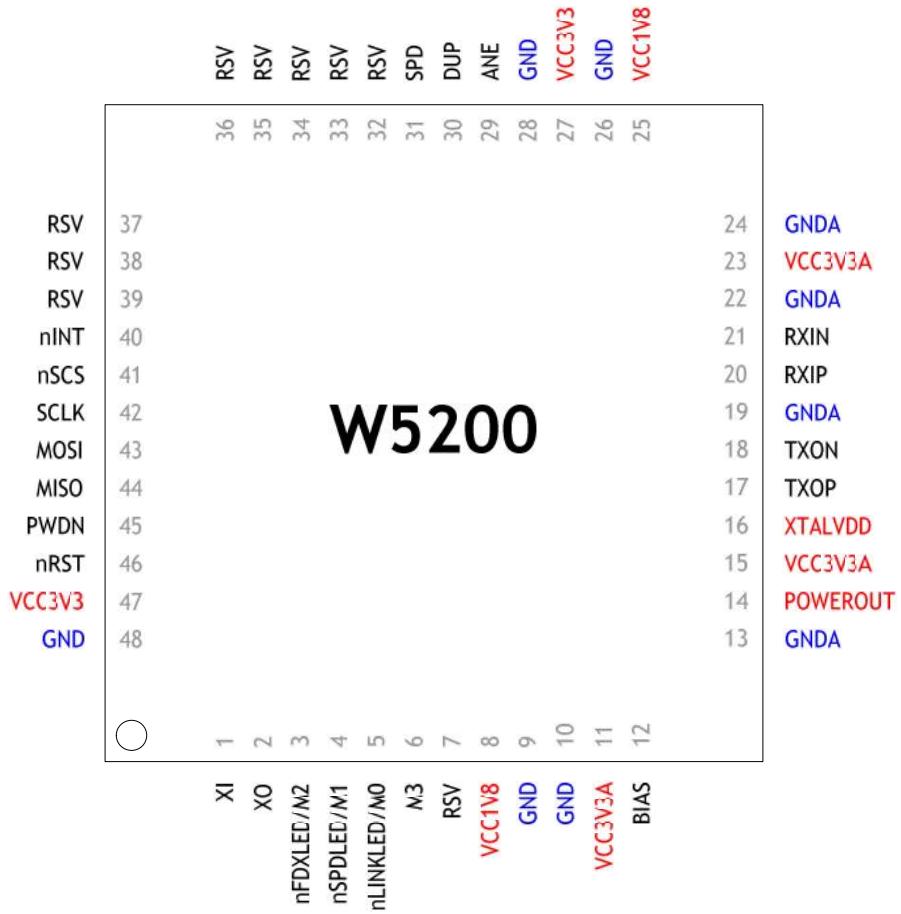


Figure 1 Pin Description W5200

1.1 MCU Interface Signals

Symbol	Type	Pin No	Description
nRST	I	46	RESET (Active LOW) This pin is active Low input to initialize or re-initialize W5200. RESET should be held at least 2us after low assert, and wait for at least 150ms after high de-assert in order for PLL logic to be stable. Refer to RESET timing of “7 Electrical Specification”
nSCS	I	41	SPI SLAVE SELECT (Active LOW) This pin is used to SPI Slave Select signal Pin when using SPI interface.
nINT	O	40	INTERRUPT (Active LOW) This pin indicates that W5200 requires MCU attention after socket connecting, disconnecting, data receiving timeout, and WOL (Wake on LAN). The interrupt is

			cleared by writing IR(Interrupt Register) or Sn_IR (Socket n-th Interrupt Register). All interrupts are maskable. This pin is active low.
SCLK	I	42	SPI CLOCK This pin is used to SPI Clock signal Pin when using SPI interface.
MOSI	I	43	SPI MASTER OUT SLAVE IN This pin is used to SPI MOSI signal pin when using SPI interface.
MISO	O	44	SPI MASTER IN SLAVE OUT This pin is used to SPI MISO signal pin.
PWDN	I	45	POWER DOWN (Active HIGH) This pin is used to power down pin. Low : Normal Mode Enable High : Power Down Mode Enable

1.2 PHY Signals

Symbol	Type	Pin No	Description
RXIP	I	20	RXIP/RXIN Signal Pair The differential data from the media is received on the RXIP/RXIN signal pair.
RXIN	I	21	
TXOP	O	17	TXOP/TXON Signal Pair The differential data is transmitted to the media on the TXOP/TXIN signal pair.
TXON	O	18	
BIAS	O	12	BIAS Register Connect a resistor of 28.7kΩ±1% to the ground. Refer to the “Reference schematic”.
ANE	I	29	Auto Negotiation Mode Enable This pin selects Enable/Disable of Auto Negotiation Mode. Low :Auto Negotiation Mode Disable High : Auto Negotiation Mode Enable
DUP	I	30	Full Duplex Mode Enable This pin selects Enable/Disable of Full Duplex Mode. Low = Half Duplex Mode Enable High = Full Duplex Mode Enable This function activates only during reset period.
SPD	I	31	Speed Mode

			This pin selects 100M/10M Speed Mode. Low = 10M Speed Mode High = 100M Speed Mode This function activates only during reset period.

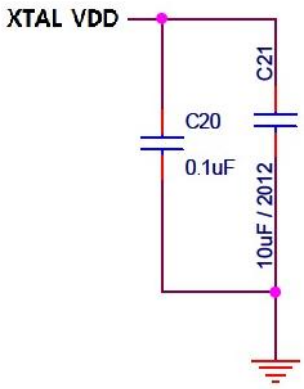
1.3 Miscellaneous Signals

Symbol	Type	Pin No	Description
nFDXLED/M2 nSPDLED/M1 nLINKLED/M0	I	3, 4, 5	W5200 MODE SELECT Normal mode : 111 Other test modes are internal test mode. This function activates only during reset period
M3	I	6	This pin should be pull-up.
RSV	-	7,32,33,34,35,36, 37,38,39	Reserved Pin The pin number 7 should be pull-up. The reserved pins except the pin number 7 should be pull-down or GND.

- Notes: Pull-Up/Down registor = 40KΩ to 100KΩ. Typical value 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 1.8V/200mA power created by internal power regulator, is used for core operation power (VCC1V8). Be sure to connect tantalum capacitor between 1V8O and GND for output frequency compensation, and selectively connect 0.1uF capacitor for high frequency noise decoupling. **Notice: 1V8O is the power for W5200 core operation. It should not be connected to the power of other devices.**

XTALVDD	I	16	 Figure 2 XTAL_VDD Reference Schematic Connect a capacitor of 10.1uF to the ground. ※ Refer to the 'W5200E01-M3 Reference schematic

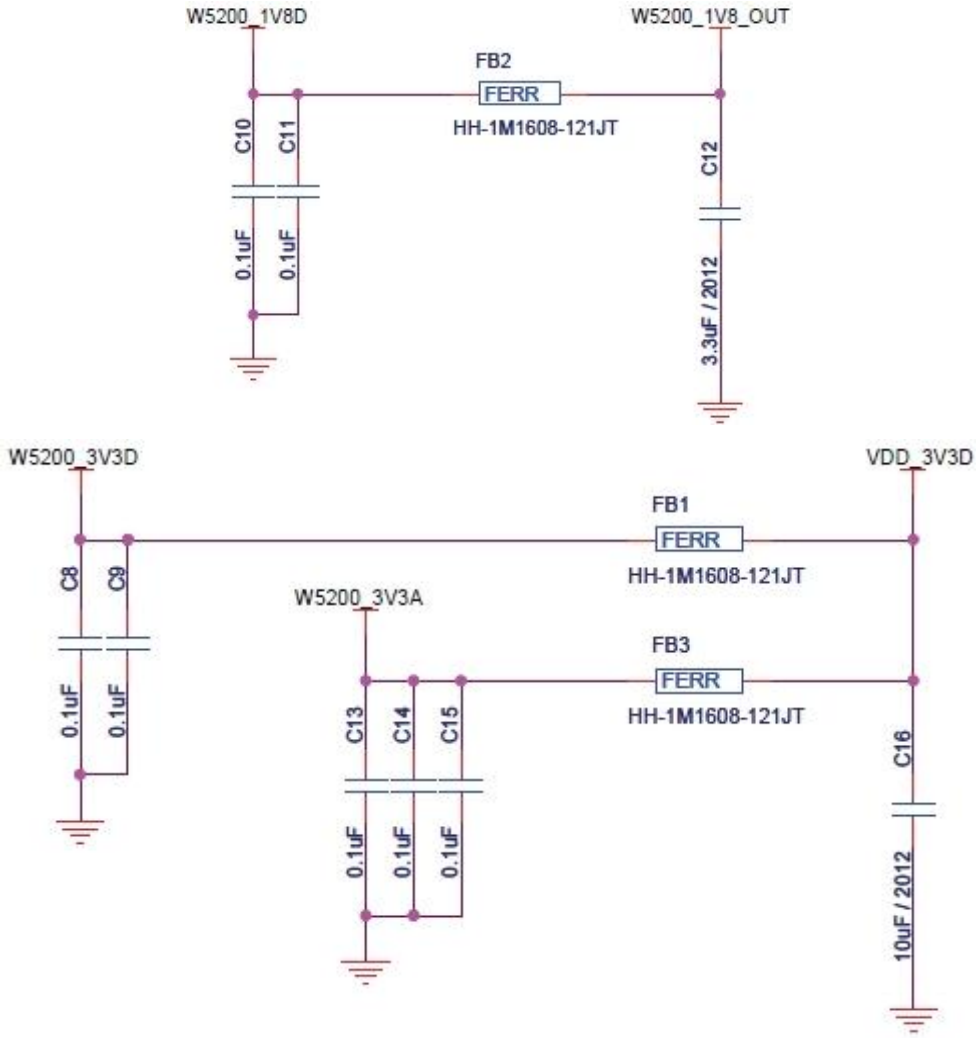


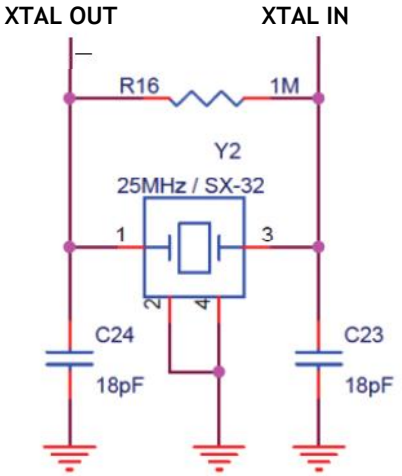
Figure 3 Power Design

Recommend for power design.

1. **Locate decoupling capacitor as close as possible to W5200.**
2. **Use ground plane as wide as possible.**

3. **If ground plane width is adequate, having a separate analog ground plane and digital ground plane is good practice.**
4. **If ground plane is not wide, design analog and digital ground planes as a single ground plane, rather than separate them.**

1.5 Clock Signals

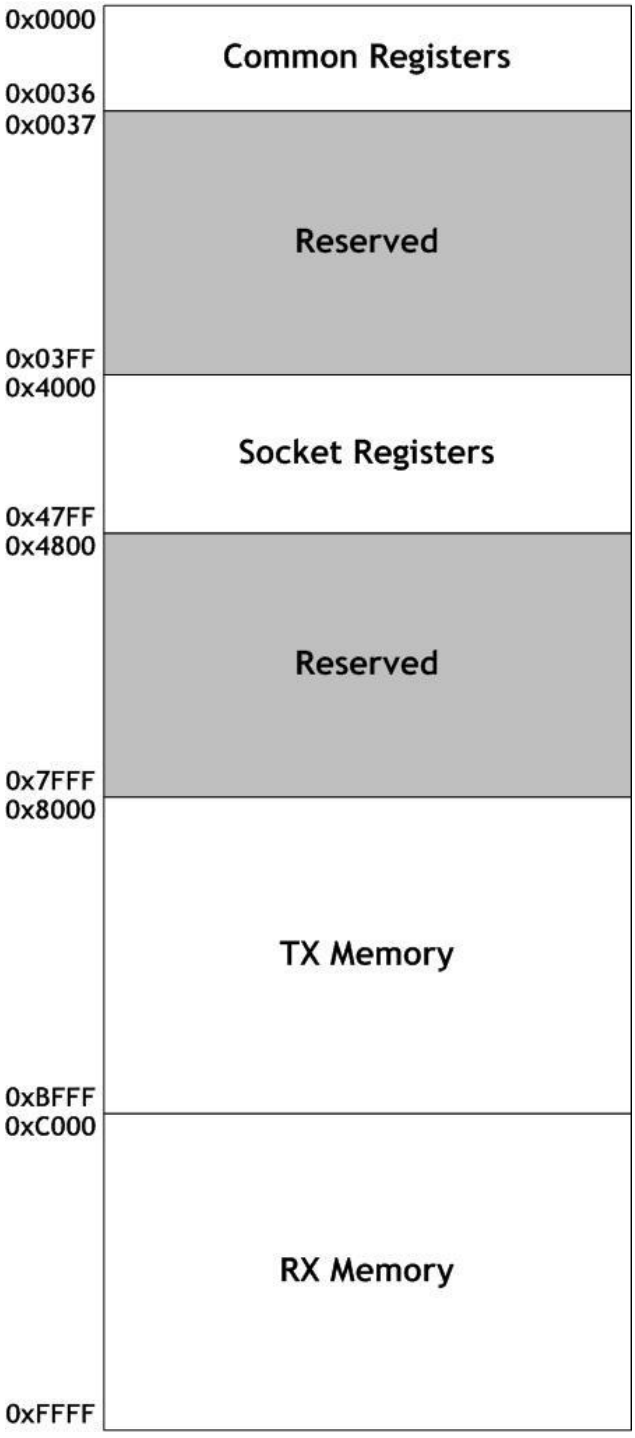
Symbol	Type	Pin No	Description
XI	I	1	25MHz crystal input/output. A 25MHz crystal and Oscillator is used to connect these pins.
XO	O	2	 Figure 4 Crystal Reference Schematic

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

W5200 is composed of Common Register, Socket Register, TX Memory, and RX Memory as shown below.



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	Reserved
0x0014	
0x0015	Interrupt (IR)
0x0016	Socket Interrupt Mask (IMR2)
0x0017	Retry Time (RTR0)
0x0018	(RTR1)
0x0019	Retry Count (RCR)
0x001A	Reserved
0x001B	

Address	Register
0x001C 0x001D	Authentication Type in PPPoE (PATR0) (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 0x0031	Interrupt Low Level Timer (INTLEVEL0) (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	Socket n SourcePort (SN_PORT0)
0x4n05	(SN_PORT1)
0x4n06	Socket n Destination Hardware Address (Sn_DHAR0)
0x4n07	(Sn_DHAR1)
0x4n08	(Sn_DHAR2)
0x4n09	(Sn_DHAR3)
0x4n0A	(Sn_DHAR4)
0x4n0B	(Sn_DHAR5)
0x4n0C	Socket 0 Destination IP Address (Sn_DIPR0)
0x4n0D	(Sn_DIPR1)
0x4n0E	(Sn_DIPR2)
0x4n0F	(Sn_DIPR3)
0x4n10	Socket 0 Destination Port (Sn_DPORT0)
0x4n11	(Sn_DPORT1)
0x4n12	Socket 0 Maximum Segment Size (Sn_MSSR0)
0x4n13	(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	Socket 0 TX Free Size (Sn_TX_FSR0)
0x4n21	(Sn_TX_FSR1)
0x4n22	Socket 0 TX Read Pointer (Sn_TX_RD0)
0x4n23	(Sn_TX_RD1)
0x4n24	Socket 0 TX Write Pointer (Sn_TX_WR0)
0x4n25	(Sn_TX_WR1)
0x4n26	Socket 0 RX Received Size (Sn_RX_RSR0)
0x4n27	(Sn_RX_RSR1)
0x4n28	Socket 0 RX Read Pointer (Sn_RX_RD0)
0x4n29	(Sn_RX_RD1)
0x4n2A	Socket 0 RX Write Pointer (Sn_RX_WR0)
0x4n2B	(Sn_RX_WR1)
0x4n2C	Socket Interrupt Mask (Sn_IMR)
0x4n2D	Fragment Offset in IP header (Sn_FRAG0)
0x4n2E	(Sn_FRAG1)
0x4n30 ~ 0x4nFF	Reserved

4 Register Descriptions

4.1 Common Registers

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

This register is used for S/W reset, ping block mode and PPPoE mode.

7	6	5	4	3	2	1	0
RST			PB	PPPoE			

Bit	Symbol	Description
7	RST	**S/W Reset** If this bit is '1', internal register will be initialized. It will be automatically cleared after reset.
6	Reserved	Reserved
5	Reserved	Reserved
4	PB	**Ping Block Mode** 0 : Disable Ping block 1 : Enable Ping block If the bit is set as '1', there is no response to the ping request.
3	PPPoE	**PPPoE Mode** 0 : DisablePPPoE mode 1 : EnablePPPoE mode If you use ADSL without router or etc, you should set the bit as '1' to connect to ADSL Server. For more detail, refer to the application note, *"How to connect ADSL"*.
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

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

This Register sets up the default gateway address.

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]

This register sets up the subnet mask address.

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]

This register sets up the Source Hardware address.

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]

This register sets up the Source IP address.

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]

This register is accessed by the host processor to know the cause of interrupt. Any interruption can be masked in the Interrupt Mask Register (IMR). The nINT signal retain low as long as any masked signal is set, and will not go high until all masked bits in this Register have been cleared.

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

Bit	Symbol	Description
7	CONFLICT	**IP Conflict** It is set as '1' when there is ARP request with same IP address as Source IP address. This bit is cleared to '0' by writing '1' to this bit.
6	Reserved	Reserved
5	PPPoE	**PPPoE Connection Close** In the Point-to-Point Protocol over Ethernet (PPPoE) Mode, if the PPPoE connection is closed, '1' is set. This bit will be cleared to '0' by writing '1' to this bit.
4	Reserved	Reserved
3	Reserved	Reserved
2	Reserved	Reserved
1	Reserved	Reserved
0	Reserved	Reserved

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

The Interrupt Mask Register is used to mask interrupts. Each interrupt mask bit corresponds to a bit in the Interrupt Register (IR). If an interrupt mask bit is set, an interrupt will be issued whenever the corresponding bit in the IR is set. If any bit in the IMR is set as '0', an interrupt will not occur though the bit in the IR is set.

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]

It configures the retransmission timeout-period. The standard unit of RTR is 100us. RTR is initialized with 2000(0x07D0) and has 200ms timeout-period.

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

0x0017	0x0018
0x0F	0xA0

Re-transmission will occur if there is no response from the remote peer to the commands of CONNECT, DISCON, CLOSE, SEND, SEND_MAC and SEND_KEEP, or the response is delayed.

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

It configures the number of retransmission times. When retransmission occurs as many as 'RCR+1' times, Timeout interrupt is set ('TIMEOUT' bit of Sn_IR is set as '1').

In case of using TCP communication, the value of Sn_SR (Socket n-th Status Register) is changed to 'SOCK_CLOSED' and Sn_IR(Socket n-th Status Register) (TIMEOUT) turns into '1'. In case of not using TCP communication, only Sn_IR(TIMEOUT) turns into '1'.

Ex) RCR = 0x0007

0x0019
0x07

The timeout of W5200 can be configurable with RTR and RCR. W5200's timeout has Address Resolution Protocol (ARP) and TCP retransmission timeout.

At the ARP (Refer to RFC 826, http://www.ietf.org/rfc.html) retransmission timeout, W5200 automatically sends ARP-request to the peer's IP address in order to acquire MAC address information (used for communication of IP, UDP, or TCP). As waiting for ARP-response from the peer, if there is no response during the time set in RTR, Timeout occurs and ARP-request is re-transmitted. It is repeated as many as 'RCR + 1' times.

Even after ARP-request retransmissions are repeated 'RCR + 1' times, if there is no ARP-response, the final timeout occurs and Sn_IR(TIMEOUT) becomes '1'.

The value of final timeout (ARP_{TO}) of ARP-request is as below.

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

At the TCP packet retransmission timeout, W5200 transmits TCP packets (SYN, FIN, RST, DATA packets) and waits for the acknowledgement (ACK) during the time set in RTR and RCR. If there is no ACK from the peer, Timeout occurs and TCP packets (sent earlier) are retransmitted. The retransmissions are repeated as many as 'RCR + 1' times. Even after TCP packet retransmissions are repeated 'RCR +1' times, if there is no ACK from the peer, final timeout occurs and Sn_SR is changed to 'SOCK_CLOSED" at the same time with Sn_IR(TIMEOUT) = '1'

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

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

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

The value of final timeout (TCP_{TO}) of TCP packet retransmission can be calculated as below,

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

This register notifies authentication method that has been agreed at the connection with PPPoE Server. W5200 supports two types of Authentication method - PAP and CHAP.

Value	Authentication Type
0xC023	PAP
0xC223	CHAP

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

This register notifies authentication algorithm in PPPoE mode. For detailed information, please refer to PPPoE application note.

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

This register is the W5200 chip version register.

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

This register indicates the duration for sending LCP Echo Request. Value 1 is about 25ms.

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]

This register is used in Magic number option during LCP negotiation. Refer to the application note, *"How to connect ADSL"*.

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

It sets Interrupt Assert wait time (I_{AWT}). It configures nINT Low Assert waiting time until the next interrupt.

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

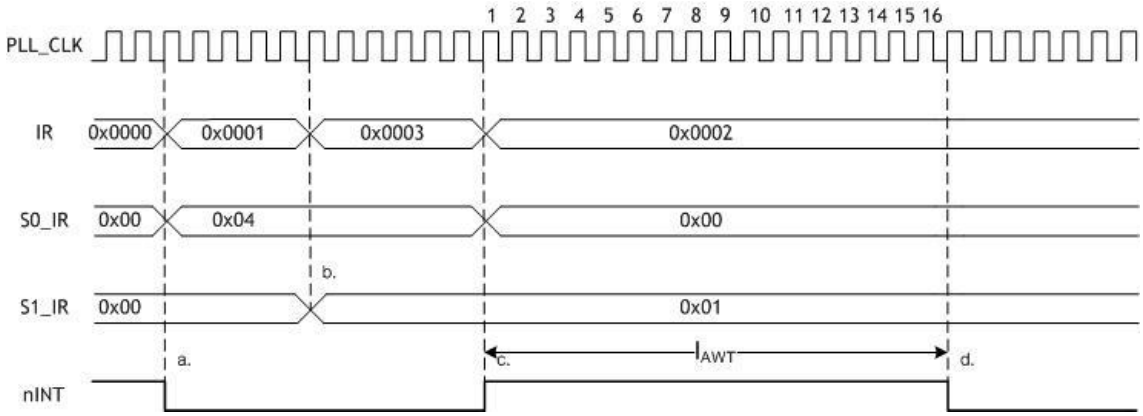


Figure 5 INTLEVEL Timing

a. At SOCKET 0, Receive Interrupt occurs (S0_IR(3) = '1') and corresponding IR2 bit is set as '1' (IR(S0_IR) = '1'). nINT signal is asserted low.

b. At SOCKET 1, Connected Interrupt occurs (S1_IR(0) = '1') and corresponding IR2 bit set as '1' (IR(S1_IR) = '1').

c. The Host clears S0_IR1 (S0_IR = 0x00) and corresponding IR bit is automatically cleared (IR(S0_IR) = '0'). nINT signal becomes High.

d. S0_IR is cleared. As IR2 is not 0x00, nINT should be asserted low right after 1PLL_CLK. However, as INTLEVEL is 0x000F, the interrupt about IR is processed after I_{AWT}(16 PLL_CLK).

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

IR2 is the Register to notify W5200 SOCKET interrupt to the Host. If any interrupt occurs, the related bit of IR2 is set as '1'. When related Mask Bit is '1', nINT signal is asserted low. nINT keeps low until all bits of Sn_IR becomes '0'. If all bits of Sn_IR become '0', it becomes high automatically.

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	When an interrupt occurs at SOCKET 7-th, it becomes '1'. This interrupt information is applied to S7_IR. This bit is automatically cleared when S7_IR is cleared to 0x00 by host.
6	S6_INT	When an interrupt occurs at SOCKET 6-th, it becomes '1'. This interrupt information is applied to S6_IR. This bit is automatically cleared when S6_IR is cleared to 0x00 by host.
5	S5_INT	When an interrupt occurs at SOCKET 5-th, it becomes '1'. This interrupt information is applied to S5_IR. This bit is automatically cleared when S5_IR is cleared to 0x00 by host.
4	S4_INT	When an interrupt occurs at SOCKET 4-th, it becomes '1'. This interrupt information is applied to S4_IR. This bit is automatically cleared when S4_IR is cleared to 0x00 by host.
3	S3_INT	When an interrupt occurs at SOCKET 3-th, it becomes '1'. This interrupt information is applied to S3_IR. This bit is automatically cleared when S3_IR is cleared to 0x00 by host.
2	S2_INT	When an interrupt occurs at SOCKET 2-th, it becomes '1'. This interrupt information is applied to S2_IR. This bit is automatically cleared when S2_IR is cleared to 0x00 by host.
1	S1_INT	When an interrupt occurs at SOCKET 1-th, it becomes '1'. This interrupt information is applied to S1_IR. This bit is automatically cleared when S1_IR is cleared to 0x00 by host.
0	S0_INT	When an interrupt occurs at SOCKET 0-th, it becomes '0'. This interrupt information is applied to S0_IR. This bit is automatically cleared when S0_IR is cleared to 0x00 by host.

PHYSTATUS(W5200 PHY status Register)[R/W][0x0035][0x00]

PHYSTATUS is the Register to indicate W5200 status of PHY.

Bit	Symbol	Description
7	Reserved	Reserved
6	Reserved	Reserved
5	LINK	**Link Status Register[Read Only]** This register indicates Link status. 0 : Link down 1 : Link Up
4	Reserved	Reserved
3	POWERDOWN	**Power down mode of PHY[Read Only]** This register indicates status of Power down mode 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]

The Interrupt Mask Register is used to mask interrupts. Each interrupt mask bit corresponds to a bit in the Interrupt Register2 (IR2). If an interrupt mask bit is set, an interruption will be issued whenever the corresponding bit in the IR2 is set. If any bit in the IMR is set as '0' an interrupt will not occur though the bit.

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

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

This register sets up socket option or protocol type for each socket.

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 It is applied only in case of UDP. For using multicasting, write multicast group address to Socket n-th Destination IP and multicast group port number to Socket n-th Destination Port Register, before OPEN command.
6	MF	MAC Filter 0 : Disable MAC filter 1 : Enable MAC filter It is used in MACRAW (P3~P0: “0100”). When this bit is set as ‘1’, W5200 can receive packet that is belong in itself or broadcasting. When this bit is set as ‘0’, W5200 can receive all packets on Ethernet. When using the hybrid TCP/IP stack, it is recommended to be set as ‘1’ for reducing the receiving overhead of host.
5	ND/MC	Use No Delayed ACK 0 : Disable No Delayed ACK option 1 : Enable No Delayed ACK option, This only applies to TCP case (P3-P0 : “0001”) If this bit is set as ‘1’, ACK packet is immediately transmitted after receiving data packet from a peer. If this bit is cleared, ACK packet is transmitted according to internal timeout mechanism. Multicast 0 : using IGMP version 2 1 : using IGMP version 1 This bit is valid when MULTI bit is enabled and UDP mode is used (P3-P0 : “0010”). In addition, multicast can be used to send out the version number in IGMP messages such as Join/Leave/Report to multicast-group

[1] *n* is Socket n-thumber (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]

<table>
<tr><td>4</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>3</td><td>P3</td><td rowspan="4">Protocol
Sets up corresponding socket as TCP, UDP, or IP RAW mode
<table>
<tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>Closed</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>TCP</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>UDP</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>IPRAW</td></tr>
</table></td></tr>
<tr><td>2</td><td>P2</td></tr>
<tr><td>1</td><td>P1</td></tr>
<tr><td>0</td><td>P0</td><td>* In case of socket 0, MACRAW and PPPoE mode exist.
<table>
<tr><th>P3</th><th>P2</th><th>P1</th><th>P0</th><th>Meaning</th></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>MACRAW</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>PPPoE</td></tr>
</table>
S0_MR_MACRAW and S0_MR_PPPoE are valid only in SOCKET 0.
S0_MR_PPPoE is temporarily used for PPPoE server connection/Termination. After connection is established, it can be utilized as another protocol.</td></tr>
</table>

S*n*_CR (Socket n-th Command Register) [R/W] [0x4001+0x0n00] [0x00]

This is used to set the command for Socket n-th such as OPEN, CLOSE, CONNECT, LISTEN, SEND, and RECEIVE. After W5200 identifies the command, the Sn_CR register is automatically cleared to 0x00. Even though Sn_CR is cleared to 0x00, the command is still being processed. To verify whether the command is completed or not, please check the Sn_IR or Sn_SR registers.

<table>
<tr><th>Value</th><th>Symbol</th><th>Description</th></tr>
<tr><td>0x01</td><td>OPEN</td><td>Socket n-th is initialized and opened according to the protocol selected in Sn_MR (P3:P0). The table below shows the value of Sn_SR corresponding to Sn_MR
<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>This is valid only in TCP mode (Sn_MR(P3:P0) = Sn_MR_TCP). In this mode, the Socket n-th is configured as a TCP server which is waiting for connection-request (SYN packet) from any “TCP CLIENT”. The Sn_SR register changes the state from SOCK_INIT to SOCKET_LISTEN.
When a client’s connection request is successfully established, the Sn_SR changes from SOCK_LISTEN to SOCK_ESTABLISHED and the Sn_IR(0) becomes ‘1’. On the other hand, Sn_IR(3) is set as ‘1’ and Sn_SR changes to SOCK_CLOSED during a connection failure(SYN/ACK packet failed to transfer)
cf> If the destination port of the TCP Client does not exist during a connection request, W5200 will transmit a RST packet and Sn_SR is unchanged.</td></tr>
<tr><td>0x04</td><td>CONNECT</td><td>This mode is only valid in TCP mode and operates the Socket n-th as a TCP client.A connect-request (SYN packet) is sent to the TCP server by connecting to the IP address and port stored in destination address and port registers (Sn_DIPR0 and Sn_DPORT0)
When a client’s connection request is successfully established, the Sn_SR register is changed to SOCK_ESTABLISHED and the Sn_IR(0) becomes ‘1’.In the following cases, the connect-request fails When a ARPTO occurs (Sn_IR(s)=‘1’) because the Destination Hardware Address is not acquired through the ARP process</td></tr>
</table>

		When a SYN/ACK packet is not received and TCPTO(Sn_IR(3)) is'1' When a RST packet is received instead of a SYN/ACK packet Above three cases, Sn_SR is changed to SOCK_CLOSED.
0x08	DISCON	Only valid in TCP mode Regardless of "TCP SERVER" or "TCP CLIENT", this disconnect command processes the **Active close**: it transmits disconnect-request(FIN packet) to the connected peer **Passive close**: When FIN packet is received from peer, a FIN packet is replied back to the peer when FIN/ACK packet is received, Sn_SR is changed to SOCK_CLOSED. When a disconnect request is not received, TCPTO occurs (Sn_IR(3)='1') and Sn_SR is changed to SOCK_CLOSED. cf> If CLOSE is used instead of DISCON, only Sn_SR is changed to SOCK_CLOSED without disconnect-process(disconnect-request). If a RST packet is received from a peer during communication, Sn_SR is unconditionally changed to SOCK_CLOSED.
0x10	CLOSE	Closes Socket n-th. Sn_SR is changed to SOCK_CLOSED.
0x20	SEND	SEND transmits all the data buffered in the TX memory. For more details, please refer to Socket n-th TX Free Size Register (Sn_TX_FSR0), Socket n-th TX Write Pointer Register(Sn_TX_WR0), and Socket n-th TX Read Pointer Register(Sn_TX_RD0).
0x21	SEND_MAC	Used in UDP mode only The basic operation is same as SEND. Normally SEND operation needs Destination Hardware Address which can be retrieved by the ARP (Address Resolution Protocol) process. SEND_MAC uses Socket n-th Destination Hardware Address(Sn_DHAR0) that is chosen by the user without going through the ARP process.
0x22	SEND_KEEP	Used in TCP mode It checks the connection status by sending 1byte data. If the connection has no response from peers or is terminated, the Timeout interrupt will occur.
0x40	RECV	RECV processes the data received by using a RX read pointer register(Sn_RX_RD). For more detail, please refer to 5.2.1.1 SERVER mode Receiving Process with Socket n-th RX Received Size Register (Sn_RX_RSR0), Socket n-th RX Write Pointer Register(Sn_RX_WR), and Socket n-th RX Read Pointer

		Register(Sn_RX_RD).

Below commands are only valid for SOCKET 0 and S0_MR(P3:P0) = S0_MR_PPPoE. For more detail refer to the “How to use ADSL”.

Value	Symbol	Description
0x23	PCON	PPPoE connection begins by transmitting PPPoE discovery packet
0x24	PDISCON	Closes PPPoE connection
0x25	PCR	In each phase, it transmits REQ message.
0x26	PCN	In each phase, it transmits NAK message.
0x27	PCJ	In each phase, it transmits REJECT message.

S*n*_IR (Socket *n-th* Interrupt Register) [R] [0x4002+0x0n00] [0x00]

Sn_IR register provides information such as the type of interrupt (establishment, termination, receiving data, timeout) used in Socket n-th. When an interrupt occurs and the mask bit of Sn_IMR is '1', the interrupt bit of Sn_IR becomes '1'.

In order to clear the Sn_IR bit, the host should write the bit as '1'. When all the bits of Sn_IR is cleared ('0'), IR(*n*) is automatically cleared.

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' and 'S0_MR(P3:P0)=S0_MR_PPPoE' PPP Receive Interrupts when the option which is not supported is received
6	PFAIL	**Sn_IR(PFAIL) Interrupt Mask** Valid only in case of 'SOCKET=0' & 'S0_MR(P3:P0)=S0_MR_PPPoE' PPP Fail Interrupts when PAP Authentication is failed
5	PNEXT	**Sn_IR(PNEXT) Interrupt Mask** Valid only in case of 'SOCKET=0' & 'S0_MR(P3:P0)=S0_MR_PPPoE' PPP Next Phase Interrupts when the phase is changed during ADSL connection process
4	SEND_OK	**Sn_IR(SENDOK) Interrupt Mask** SEND OK Interrupts when the SEND command is completed
3	TIMEOUT	**Sn_IR(TIMEOUT) Interrupt Mask** TIMEOUT Interrupts when ARP_{TO} or TCP_{TO} occurs
2	RECV	**Sn_IR(RECV) Interrupt Mask** Receive Interrupts whenever data packet is received from a peer
1	DISCON	**Sn_IR(DISCON) Interrupt Mask** Disconnect Interrupts when FIN of FIN/ACK packet is received from a peer
0	CON	**Sn_IR(CON) Interrupt Mask** Connect Interrupts when a connection is established with a peer

S*n*_SR (Socket n-th Status Register) [R] [0x4003+0x0n00] [0x00]

This register provides the status of Socket *n-th*. SOCKET status are changed when using the Sn_CR register or during packet transmission/reception. The table below describes the different states of Socket n-th.

Value	Symbol	Description
0x00	SOCK_CLOSED	It is the status that resource of SOCKETn is released. When DISCON or CLOSE command is performed, or ARP_{TO}, or TCP_{TO} occurs, it is changed to SOCK_CLOSED regardless of previous value.
0x13	SOCK_INIT	It is shown in case that S*n*_MR is set as TCP and OPEN commands are given to S*n*_CR. It is changed to SOCK_INIT when Sn_MR (P3:P0) is Sn_MR_TCP and OPEN command is performed. It is the initial step of TCP connection establishment. It is possible to perform LISTEN command at the "TCP SERVER" mode and CONNECT command at the "TCP CLIENT". It is the status that SOCKETn operates as "TCP SERVER" and waits for connect-request (SYN packet) from "TCP CLIENT".
0x14	SOCK_LISTEN	Socket n-th operates in TCP Server Mode and waits for a connection-request (SYN packet) from a “TCP CLIENT”. When the LISTEN command is used, the stage changes to SOCK_LISTEN Once the connection is established, the SOCKET state changes from SOCK_LISTEN to SOCK_ESTABLISHED; however, if the connection fails, TCP_{TO} occurs (Sn_IR(TIME_OUT) = ‘1’) and the state changes to SOCK_CLOSED.
0x17	SOCK_ESTABLISHED	It is shown in case that connection is established. It is changed to SOCK_ESTABLISHED when SYN packet from “TCP CLIENT” is successfully processed at the SOCK_LISTEN, or CONNECTS command is successfully performed. At this status, DATA packet can be transferred, that is, SEND or RECV command can be performed.
0x1C	SOCK_CLOSE_WAIT	It is the status that disconnect-request (FIN packet) is received from the peer As TCP connection is half-closed, it is possible to transfer data packet. In order to complete the TCP disconnection, DISCON command should be performed. For SOCKETn close without disconnection-process, CLOSE command should be just performed.

0x22	SOCK_UDP	It is the status that SOCKETn is open as UDP mode. It is changed to SOCK_UDP when Sn_MR(P3:P0) is Sn_MR_UDP and OPEN command is performed. DATA packet can be transferred without connection that is necessary to TCP mode SOCKET.
0x32	SOCK_IPRAW	The socket is opened in IPRAW mode. The SOCKET status is change to SOCK_IPRAW when Sn_MR (P3:P0) is Sn_MR_IPRAW and OPEN command is used. IP Packet can be transferred without a connection similar to the UDP mode.
0x42	SOCK_MACRAW	It is changed to SOCK_MACRAW in case of S0_CR=OPEN and S0_MR (P3:P0)=S0_MR_MACRAW. MACRAW packet (Ethernet frame) can be transferred similar to UDP mode.
0x5F	SOCK_PPPOE	It is the status that SOCKET0 is open as PPPoE mode. It is changed to SOCK_PPPoE in case of S0_CR=OPEN and S0_MR (P3:P0)=S0_MR_PPPoE . It is temporarily used at the PPPoE connection.

Below is shown during changing the status.

Value	Symbol	Description
0x15	SOCK_SYNSENT	This status indicates that a connect-request (SYN packet) is sent to a "TCP SERVER". This status shows changing process from SOCK_INIT to SOCK_ESTABLISHED by CONNECT command. At this status, if connect-accept (SYN/ACK packet) is received from "TCP SERVER", it is automatically changed to SOCK_ ESTBLISHED. If SYN/ACK packet is not received from the "TCP SERVER" before TCPTO occurs (Sn_IR(TIMEOUT)='1'), it is changed to SOCK_CLOSED.
0x16	SOCK_SYNRECV	This status indicate that a connect-request(SYN packet) is received from a "TCP CLIENT". It is automatically changed to SOCK_ESTABLISHED when W5200 successfully transmits connect-accept (SYN/ACK packet) to the "TCP CLIENT". If it is failed, TCPTO occurs (Sn_IR(TIMEOUT)='1'), and it is changed to SOCK_CLOSED
0x18	SOCK_FIN_WAIT	These status shows that Socket n-th is closed. It is observed in the disconnect-process of active close or passive close. It is changed to SOCK_CLOSED, when disconnect-process is successfully finished or TCPTO occurs (Sn_IR (TIMEOUT) ='1').
0x1A	SOCK_CLOSING	
0X1B	SOCK_TIME_WAIT	
0X1D	SOCK_LAST_ACK	
0x01	SOCK_ARP	This status indicates that ARP-request is transmitted in order to acquire destination hardware address. This status

		is observed when SEND command is performed at the SOCK_UDP or SOCK_IPRAW, or CONNECT command is performed at the SOCK_INIT. If hardware address is successfully acquired from destination (when ARP-response is received), it is changed to SOCK_UDP, SOCK_IPRAW or SOCK_SYNSENT. If it's failed and ARPTO occurs (Sn_IR(TIMEOUT)='1'), in case of UDP or IPRAW mode it goes back to the previous status(the SOCK_UDP or SOCK_IPRAW), in case of TCP mode it goes to the SOCK_CLOSED. cf> ARP-process operates at the SOCK_UDP or SOCK_IPRAW when the previous and current values of Sn_DIPR are different. If the previous and current values of Sn_DIPR are same, ARP-process doesn't operate because the destination hardware address is already acquired.

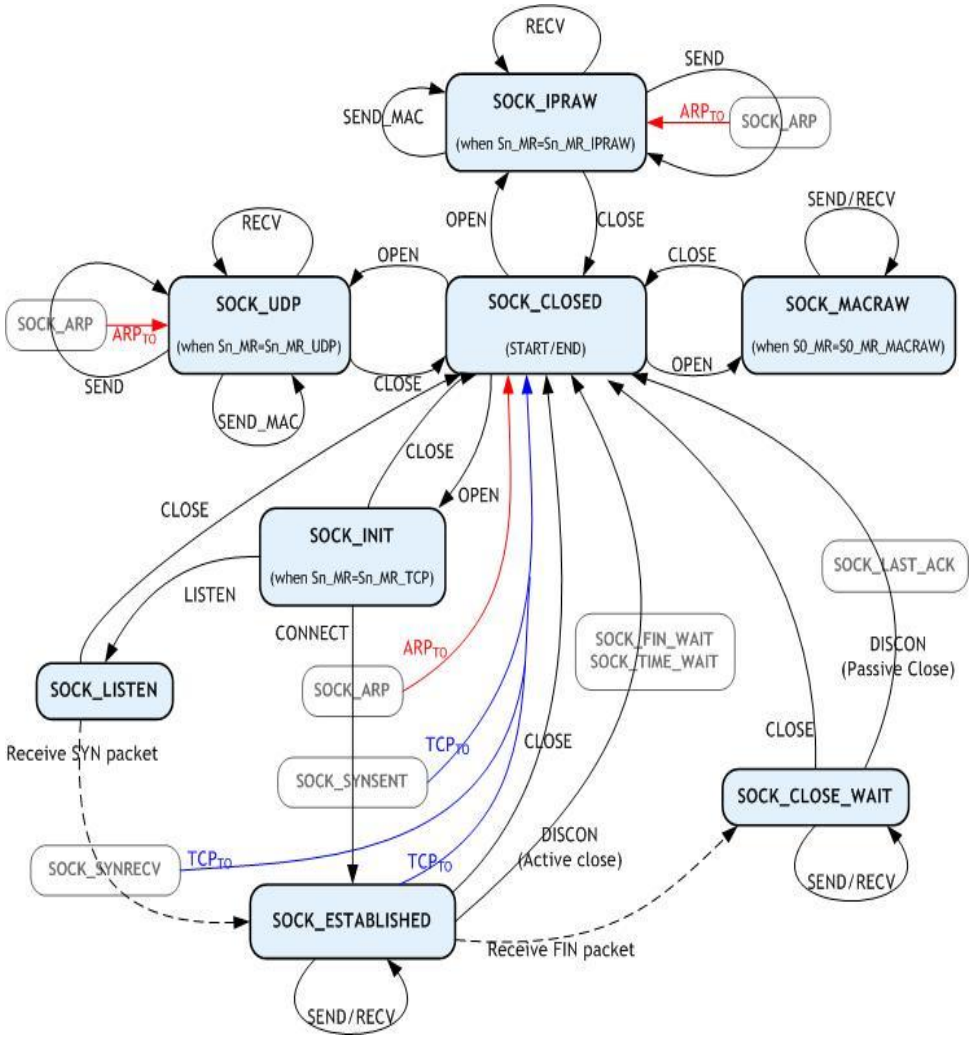


Figure 6 Socket Status Transition

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

This register sets the Source Port number for each Socket when using TCP or UDP mode, and the set-up needs to be made before executing the OPEN command.

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

0x4004	0x4005
0x13	0x88

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

It sets or is set as destination hardware address of Socket *n-th*. Also, if SOCKET 0 is used for PPPoE mode, S0_DHAR sets as PPPoE server hardware an address that is already known.

When using SEND_MAC command at the UDP or IPRAW mode, it sets the destination hardware address of Socket n-th. At the TCP, UDP and IPRAW mode, Sn_DHAR is set as destination hardware address that is acquired by ARP-process of CONNECT or SEND command. The host can acquire the destination hardware address through Sn_DHAR after successfully performing CONNET or SEND command.

When using PPPoE-process of W5200, PPPoE server hardware address is not required to be set. However, even if PPPoE-process of W5200 is not used, but implemented by yourself with MACRAW mode, in order to transmit or receive the PPPoE packet, PPPoE server hardware address(acquired by your PPPoE-process), PPPoE server IP address, and PPP session ID should be set, and MR(PPPoE) also should be set as '1'.

S0_DHAR sets the PPPoE server hardware address before the OPEN command. PPPoE server hardware address which is set by S0_DHAR is applied to PDHAR after performing the OPEN command. The configured PPPoE information is internally valid even after the CLOSE command.

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-th Destination IP Address Register)[R/W][0x400C+0x0n00 0x400F+0x0n00] [0x00000000]

It sets or is set as destination IP address of Socket n-th. If SOCKET0 is used as PPPoE mode, S0_DIPR sets PPPoE server IP address that is already known. It is valid only in TCP, UDP, IPRAW or PPPoE mode, but ignored in MACRAW mode. At the TCP mode, when operating as "TCP CLIENT" it sets as IP address of "TCP SERVER" before performing CONNECT command and when operating as "TCP SERVER", it is internally set as IP address of "TCP CLIENT" after successfully establishing connection.

At the UDP or IPRAW mode, Sn_DIPR sets as destination IP address to be used for transmitting UDP or IPRAW DATA packet before performing SEND or SEND_MAC command.

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-th Destination Port Register)[R/W][0x4010+0x0n00-0x4011+0x0n00] [0x00]

The destination port number is set in the Sn_DPORT of Socket n-th. If SOCKET 0 is used as PPPoE mode, S0_DPORT0 sets PPP session ID that is already known. It is valid only in TCP, UDP or PPPoE mode, and ignored in other modes.

At the TCP mode, when operating as "TCP CLIENT", it listens for the port number of the "TCP SERVER" before performing the CONNECT command.

At the UDP mode, the destination port number is set in the Sn_DPORT to be used for transmitting UDP DATA packets before performing SEND or SEND_MAC command.

At the PPPoE mode, the PPP session ID that is already known is set in the S0_DPORT. PPP session ID (set by S0_DPORT0) is applied to PSIDR after performing the OPEN command.

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

0x4010	0x4011
0x13	0x88

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

This register is used for MSS (Maximum Segment Size) of TCP, and the register displays MSS set by the other party when TCP is activated in Passive Mode. It just supports TCP or UDP mode. When using PPPoE (MR(PPPoE)='1'), the MTU of TCP or UDP mode is assigned in the range of MTU of PPPoE.

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			

At the IPRAW or MACRAW, MTU is not processed internally, but default MTU is used. Therefore, when transmitting the data bigger than default MTU, the host should manually divide the data into the unit of default MTU.

At the TCP or UDP mode, if transmitting data is bigger than MTU, W5200 automatically divides

the data into the unit of MTU. MTU is called as MSS at the TCP mode. By selecting from Host-Written-Value and peer's MSS, MSS is set as smaller value through TCP connection process.

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

0x4012	0x4013
0x05	0xB4

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

It is a 1 byte register that sets the protocol number field of the IP header at the IP layer. It is valid only in IPRAW mode, and ignored in other modes. Sn_PROTO is set before OPEN command. When Socket n-th is opened in IPRAW mode, it transmits and receives the data of the protocol number set in Sn_PROTO. Sn_PROTO can be assigned in the range of 0x00 ~ 0xFF, but W5200 does not support TCP(0x06) and UDP(0x11) protocol number

Protocol number is defined in IANA(Internet assigned numbers authority). For the detail, refer to online document (http://www.iana.org/assignments/protocol-numbers).

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

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

It sets the TOS(Type of Service) field of the IP header at the IP layer. It should be set before the OPEN command. Refer to http://www.iana.org/assignments/ip-parameters.

S*n*_TTL (Socket n-th IP Time To Live Register) [R/W] [0x4016+0x0n00] [0x80]

It sets the TTL(Time To Live) field of the IP header at the IP layer. It should be set before the OPEN command. Refer to http://www.iana.org/assignments/ip-parameters.

S*n*_RXMEM_SIZE(Socket n-th RX Memory Size Register) [R/W] [0x401E+0x0n00] [0x02]

It configures the internal RX Memory size of each SOCKET. RX Memory size of each SOCKET is configurable in the size of 0, 1, 2, 4, 8, and 16 Kbytes. 2Kbytes is assigned when reset. $Sn\_RXMEM\_SIZE_{SUM}$(sum of Sn_RXMEM_SIZE) of each SOCKET should be 16KB.

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

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

0x401E	0x411E
0x08	0x02

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

0x421E	0x431E
0x01	0x01

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

0x441E	0x451E
0x01	0x01

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

0x461E	0x471E
0x01	0x01

S*n*_TXMEM_SIZE(Socket n-th TX Memory size Register) [R/W][0x401F+0x0n00] [0x02]

It configures the internal TX Memory size of each SOCKET. TX Memory size of each SOCKET is configurable in the size of 0, 1, 2, 4, 8, and 16Kbytes. 2Kbytes is assigned when reset. $Sn\_TXMEM\_SIZE_{SUM}$(summation of Sn_TXMEM_SIZE) of each SOCKET should be 16KB.

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

0x401F	0x411F
0x04	0x01

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

0x421F	0x431F
0x02	0x01

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

0x441F	0x451F
0x02	0x02

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

0x461F	0x471F
0x02	0x02

S*n*_TX_FSR (Socket n-th TX Free Size Register) [R] [0x4020+0x0n00-0x4021+0x0n00] [0x0800]

It notifies the available size of the internal TX memory (the byte size of transmittable data) of Socket n-th. The host can't write data as a size bigger than Sn_TX_FSR. Therefore, be sure to check Sn_TX_FSR before transmitting data, and if your data size is smaller than or the same as Sn_TX_FSR, transmit the data with SEND or SEND_MAC command after copying the data.

At the TCP mode, if the peer checks the transmitted DATA packet (if DATA/ACK packet is received from the peer), Sn_TX_FSR is automatically increased by the size of that transmitted DATA packet. At the other modes, when Sn_IR(SENDOK) is '1', Sn_TX_FSR is automatically increased by the size of the transmitted data.*When checking this register, user should read upper byte(0x4020, 0x4120, 0x4220, 0x4320, 0x4420, 0x4520, 0x4620, 0x4720) first and lower byte(0x4021, 0x4121, 0x4221, 0x4321, 0x4421, 0x4521, 0x4621, 0x4721) later to get the correct value.*

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

0x4020	0x4021
0x08	0x00

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

This register shows the address of the last transmission finishing in the TX memory. With the SEND command of Socket n-th Command Register, it transmits data from the current Sn_TX_RD to the Sn_TX_WR and automatically updates after transmission is finished. Therefore, after transmission is finished, Sn_TX_RD and Sn_TX_WR will have the same value. When reading this register, user should read upper byte (0x4022, 0x4122, 0x4222, 0x4322, 0x4422, 0x4522, 0x4622, 0x4722) first, and lower byte (0x4023, 0x4123, 0x4223, 0x4323, 0x4423, 0x4523, 0x4623, 0x4723) later to get the correct value.

S*n*_TX_WR (Socket n-th TX Write Pointer Register) [R/W] [0x4024+0x0n00-0x4025+0x0n00] [0x0000]

This register offers the location information to write the transmission data. When reading this register, user should read upper byte (0x4024, 0x4124, 0x4224, 0x4324, 0x4424, 0x4524, 0x4624, 0x4724) first, and lower byte (0x4025, 0x4125, 0x4225, 0x4325, 0x4425, 0x4525, 0x4625, 0x4725) later to get the correct value.

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

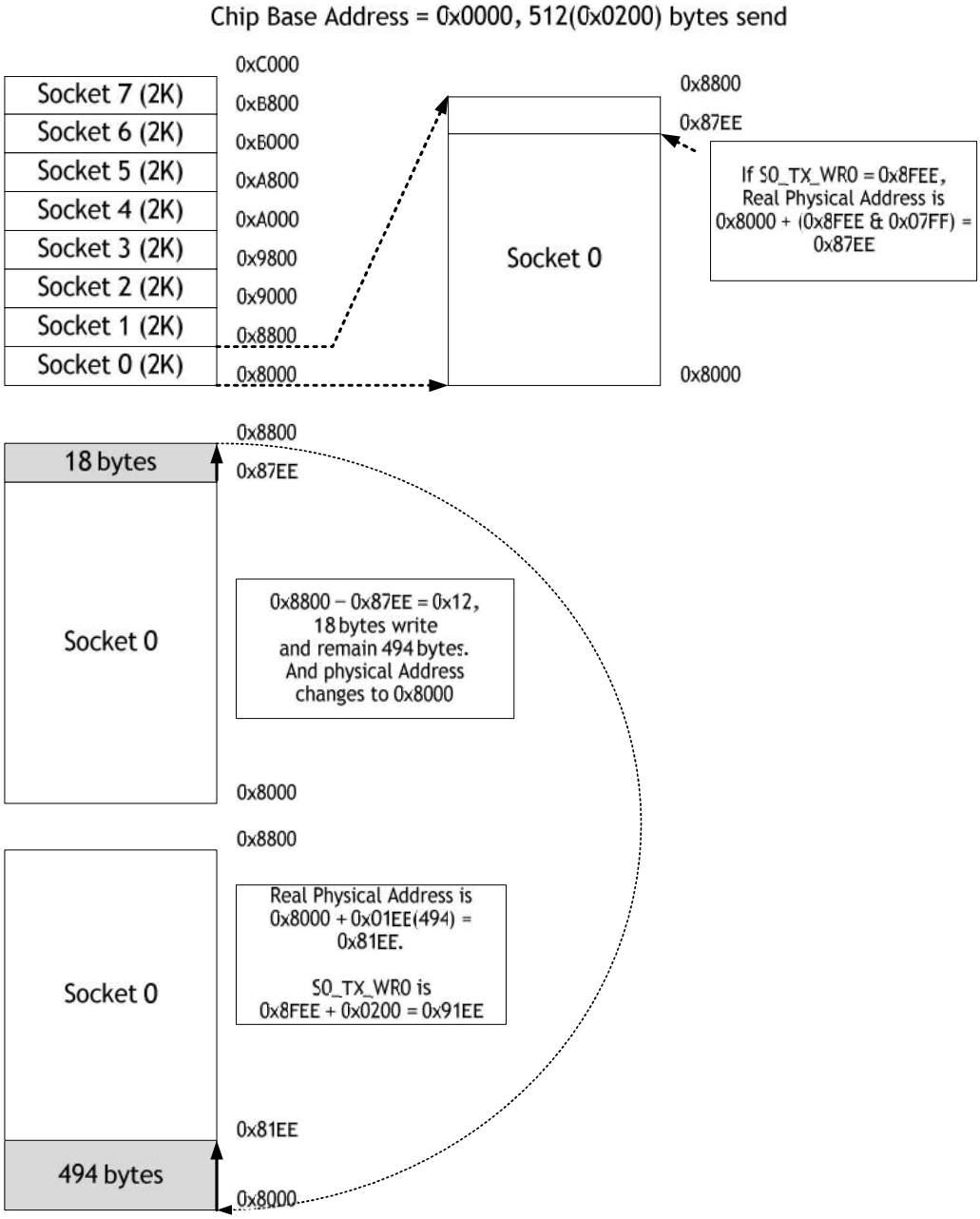


Figure 7 Physical Address Calculation

But this value itself is not the physical address to read. So, the physical address should be calculated as follow.

1. Socket n-th TX Base Address (hereafter we'll call *gSn_TX_BASE)* and Socket n-th TX Mask Address (hereafter we'll call *gSn_TX_MASK)* are calculated on TMSR value. *Refer to the psedo code of the Initialization if detail is needed.*
2. The bitwise-AND operation of two values, Sn_TX_WR and *gSn_TX_MASK* give result to the offset address(hereafter we'll call *get_offset*) in TX memory range of the socket.
3. Two values *get_offset* and *gSn_TX_BASE* are added together to give result tothe physical address(hereafter, we'll call *get_start_address*).

Now, write the transmission data to *get_start_address* as large as you want. (* There's a case that it exceeds the TX memory upper-bound of the socket while writing. In this case, write

the transmission data to the upper-bound, and change the physical address to the *gSn_TX_BASE*. Next, write the rest of the transmission data.) After that, be sure to increase the Sn_TX_WR value as much as the data size that indicates the size of writing data. Finally, give SEND command to Sn_CR(Socket n-th Command Register).

Refer to the psedo code of the transmission part on TCP Server mode if detail is needed.

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

It informs the user of the byte size of the received data in Internal RX Memory of Socket n-th. As this value is internally calculated with the values of Sn_RX_RD and Sn_RX_WR, it is automatically changed by RECV command of Socket n-th Command Register (Sn_CR) and receives data from the remote peer. *When reading this register, user should read upper byte*(0x4026, 0x4126, 0x4226, 0x4326, 0x4426, 0x4526, 0x4626, 0x4726*) first, and lower byte*(0x4027, 0x4127, 0x4227, 0x4327, 0x4427, 0x4527, 0x4627, 0x4727) *later to get the correct value*.

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-th RX Read Pointer Register) [R/W] [0x4028+0x0n00-0x4028+0x0n00] [0x0000]

This register offers the location information to read the receiving data. When reading this register, user should read upper byte (0x4028, 0x4128, 0x4228, 0x4328, 0x4428, 0x4528, 0x4628, 0x4728) first, and lower byte (0x4029, 0x4129, 0x4229, 0x4329, 0x4429, 0x4529, 0x4629, 0x4729) later to get the correct value. It has a random value as its initial value.

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

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

0x4028	0x4029
0x08	0x00

But this value itself is not the physical address to read. So, the physical address should be calculated as follow.

1. Socket n-th RX Base Address (hereafter we'll call *gSn_RX_BASE)* and Socket n-th RX Mask Address (hereafter we'll call *gSn_RX_MASK)* are calculated on RMSR value. *Refer to the pseudo code of the 5.1 Initialization if the detail is needed.*
2. The bitwise-AND operation of two values, Sn_RX_RD and *gSn_RX_MASK* give result the offset address(hereafter we'll call *get_offset)*, in RX memory range of the socket.
3. Two values *get_offset* and *gSn_RX_BASE* are added together to give result the

physical address(hereafter, we'll call *get_start_address)*.

Sn_RX_WR (Socket n-th RX Write Pointer Register)[R/W][(0xFE402A + 0xn00) - (0xFE402B + 0xn00)][0x0000]

This register offers the location information to write the receive data. When reading this register, the user should read upper bytes (0x402A, 0x412A, 0x422A, 0x432A, 0x442A, 0x452A, 0x462A, 0x472A) first and lower bytes (0x402B, 0x412B, 0x422B, 0x432B, 0x442B, 0x452B, 0x462B, 0x472B) later to get the correct value.

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

0x402A	0x402B
0x08	0x00

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

It configures the interrupt of Socket n-th so as to notify to the host. Interrupt mask bit of Sn_IMR corresponds to interrupt bit of Sn_IR. If interrupt occurs in any SOCKET and the bit is set as '1', its corresponding bit of Sn_IR is set as '1'. When the bits of Sn_IMR and Sn_IR are '1', IR(*n*) becomes '1'. At this time, if IMR(*n*) is '1', the interrupt is issued to the host. ('nINT' signal is asserted low)

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-th Fragment Register)[R/W][0x402D+0x0n00-0x402E+ 0x0n100][0x4000]

It sets the Fragment field of the IP header at the IP layer. W5200 does not support the packet fragment at the IP layer. Even though Sn_FRAG is configured, IP data is not fragmented, and not recommended either. It should be configured before performing OPEN command.

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

0x402D	0x402E
0x40	0x00

5 Functional Descriptions

By setting some register and memory operation, W5200 provides internet connectivity.

This chapter describes how it can be operated.

5.1 Initialization

Basic Setting

For the W5200 operation, select and utilize appropriate registers shown below.

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

For more information of above registers, refer to the “Register Descriptions.”

Setting network information

Basic network information setting for communication:

It must be set the basic network information.

① SHAR(Source Hardware Address Register)

It is prescribed that the source hardware addresses, which is set by SHAR, use unique hardware addresses (Ethernet MAC address) in the Ethernet MAC layer. The IEEE manages the MAC address allocation. The manufacturer which produces the network device allocates the MAC address to product.

Details on MAC address allocation refer to the website as below.

http://www.ieee.org/, http://standards.ieee.org/regauth/oui/index.shtml

② GAR(Gateway Address Register)

③ SUBR(Subnet Mask Register)

④ SIPR(Source IP Address Register)

Set socket memory information

This stage sets the socket tx/rx memory information. The base address and mask address of each socket are fixed and saved in this stage.

```
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 8 Allocation Internal TX/RX memory of Socket n-th

5.2 Data Communications

After the initialization process, W5200 can transmit and receive the data with others by 'open' the SOCKET of TCP, UDP, IPRAW, and MACRAW mode. The W5200 supports the independently and simultaneously usable 8 SOCKETS. In this section, the communication method for each mode will be introduced.

5.2.1 TCP

The TCP is a connection-oriented protocol. The TCP make the connection SOCKET by using its own IP address, port number and destination IP address, port number. Then transmits and receives the data by using this SOCKET.

Methods of making the connection to SOCKET are "TCP SERVER" and "TCP CLIENT". It is divided by transmitting the connect-request (SYN packet).

The "TCP SERVER" listens to the connect-request from the "TCP CLIENT", and makes connection SOCKET by accepting the transmitted connect-request (Passive-open).

The "TCP CLIENT" transmits the connect-request first to "TCP SERVER" to make the connection (Active-open).

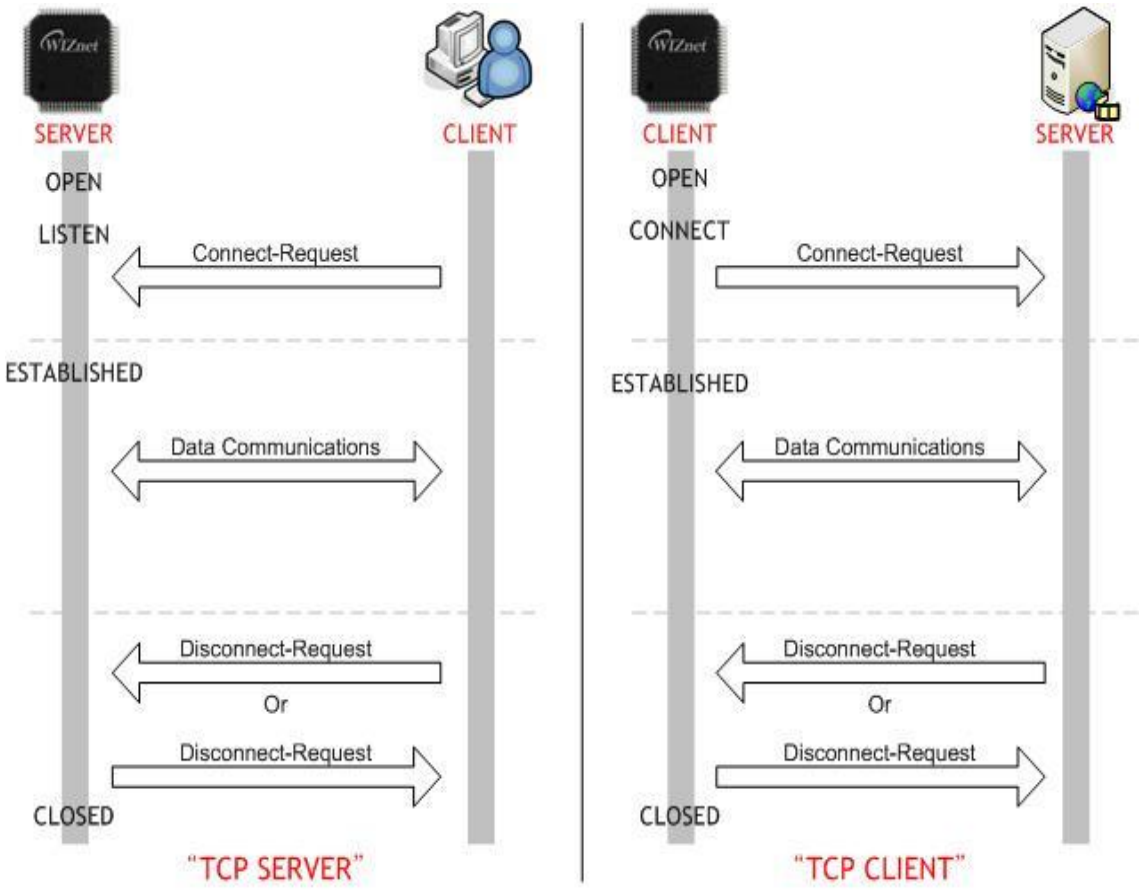


Figure 9 TCP SERVER and TCP CLIENT

5.2.1.1 TCP SERVER

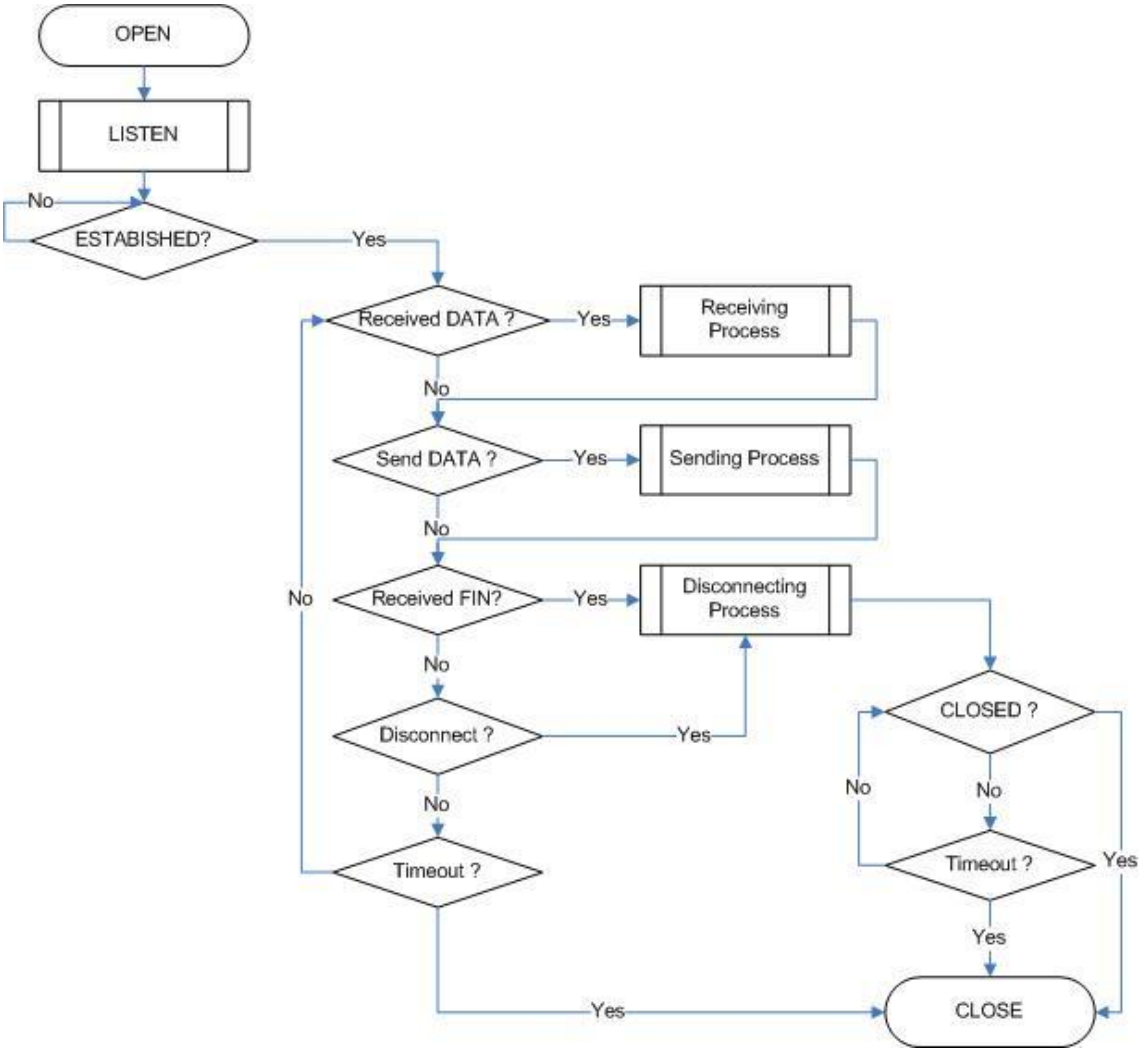


Figure 10 TCP SERVER Operation Flow

SOCKET Initialization

SOCKET initialization is required for TCP data communication. The initialization is opening the SOCKET. The SOCKET opening process selects one SOCKET from 8 SOCKETS of the W5200, and sets the protocol mode (Sn_MR) and Sn_PORT0 which is source port number (Listen port number in "TCP SERVER") in the selected SOCKET, and then executes OPEN command. After the OPEN command, if the status of Sn_SR is changed to SOCK_INIT, the SOCKET initialization process is completed.

The SOCKET initialization process is identically applied in "TCP SERVER" and "TCP CLIENT."The Initialization process of Socket n-th in TCP mode is shown below.

```
{
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

Run as "TCP SERVER" by LISTEN command.

```
{
/* 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

When the status of Sn_SR is SOCK_LISTEN, if it receives a SYN packet, the status of Sn_SR is changed to SOCK_SYNRECV and transmits the SYN/ACK packet. After that, the Socket n-th makes a connection. After it makes the connection of Socket n-th, it enables the data communication. There are two methods to confirm the connection of Socket n-th.

```
First method :
{
 if (Sn_IR(CON) == '1')   Sn_IR(CON) = '1'; goto ESTABLISHED stage;
 /* In this case, if the interrupt of Socket n-th 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

Confirm the reception of the TCP data.

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

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

The First method: set the Sn_IR(RECV) to '1' whenever you receive a DATA packet. If the host receives the next DATA packet without setting the Sn_IR(RECV) as '1' in the prior DATA packet, it cannot recognize the Sn_IR(RECV) of the next DATA packet. This is due to the prior Sn_IR(RECV) and next Sn_IR(RECV) being overlapped. So this method is not recommended if the host cannot perfectly process the DATA packets of each Sn_IR(RECV).

ESTABLISHMENT : Receiving process

In this process, it processes the TCP data which was received in the Internal RX memory. At the TCP mode, the W5200 cannot receive the data if the size of received data is larger than the RX memory free size of Socket n-th. If the prior stated condition is happened, the W5200 holds on to the connection (pauses), and waits until the RX memory's free size is larger than the size of the received data.

```
{
/* 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

The size of the transmit data cannot be larger than assigned internal TX memory of Socket n-th. If the size of transmit data is larger than configured MSS, it is divided by size of MSS and transmits. To transmit the next data, user must check the completion of prior SEND command. An error may occur if the SEND command executes before completion of prior SEND command. The larger the data size, the more time to complete the SEND command. So the user should properly divide the data to transmit.

To check the completion of the SEND command, it should be check that the send data length is equal with the actual sent data length. The actual sent data length is calculated by the difference of the Sn_TX_RD before and after performing the SEND command. If the actual sent data is less than the send data length, the SEND command is retried for sending the left data. The send process is therefore completed the SENDwhen the sum of the actual sent data is equal the send data length. A simple example of the send process is as below

Ex) Send Data Length Size= 10,

1) **Execute SEND Command with send data length**
2) **Calculate the actual sent data length**

   If the actual sent data length is 7 (= Sn_TX_RD_after_SEND-Sn_TX_RD_befor_SEND), the left Data length= 3
3) **Retry SEND Command until the sum of the actual sent data length is same the send data length.**

Note: Don't copy data until the sum of the actual sent data length is the send data length.

```
{
  /* 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;      // 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)

Check if the Disconnect-request(FIN packet) has been received. User can confirm the reception of FIN packet as below.

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

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

ESTABLISHMENT : Check disconnect / disconnecting process

When the user does not need data communication with others, or receives a FIN packet, disconnect the connection SOCKET.

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

ESTABLISHMENT : Check closed

Confirm that the Socket n-th is disconnected or closed by DISCON or close command.

```
First method :
{
 if (Sn_IR(DISCON) == '1') goto CLOSED stage;
 /* In this case, if the interrupt of Socket n-th 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

The timeout can occur by Connect-request(SYN packet) or its response(SYN/ACK packet), the DATA packet or its response(DATA/ACK packet), the Disconnect-request(FIN packet) or its response(FIN/ACK packet) and transmission all TCP packet. If it cannot transmit the above packets within 'timeout' which is configured at RTR and RCR, the TCP final timeout(TCP_{TO}) occurs and the state of Sn_SR is set to SOCK_CLOSED. Confirming method of the TCP_{TO} is as below:

```
First method :
{
 if (Sn_IR(TIMEOUT bit) == '1') Sn_IR(TIMEOUT)='1'; goto CLOSED stage;
 /* In this case, if the interrupt of Socket n-th 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

It can be used to close the Socket n-th, which disconnected by disconnect-process, or closed by TCP_{TO} or closed by host's need without disconnect-process.

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

5.2.1.2 TCP CLIENT

It is same as TCP server except 'CONNECT' state. User can refer to the "5.2.1.1 TCP SERVER".

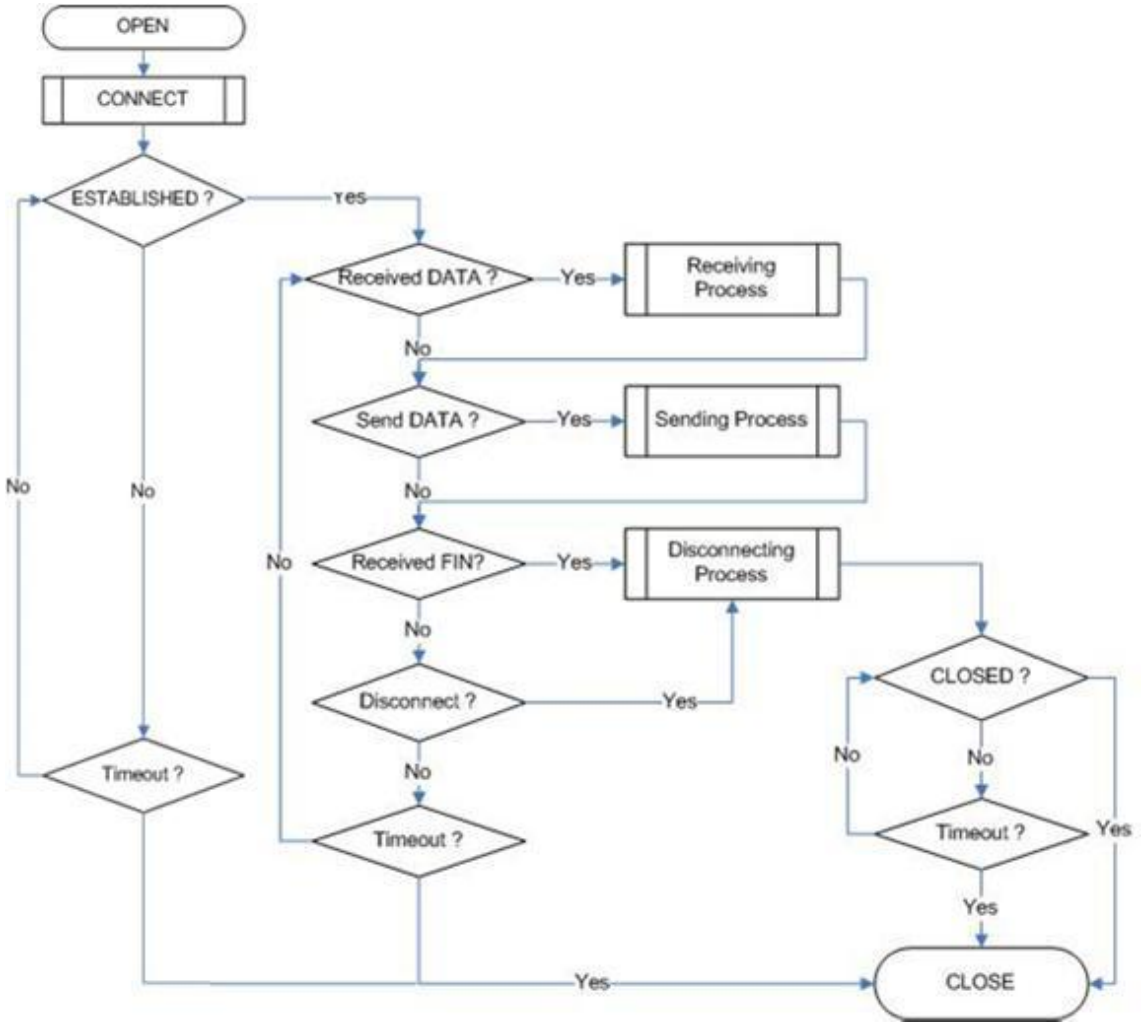


Figure 11 TCP CLIENT Operation Flow

CONNECT

Transmit the connect-request (SYN packet) to "TCP SERVER". It may occurs the timeout such as ARP_{TO}, TCP_{TO} when make the "connection SOCKET" with "TCP SERVER"

```
{
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

The UDP is a Connection-less protocol. It communicates without "connection SOCKET." The TCP protocol guarantees reliable data communication, but the UDP protocol uses datagram communication which has no guarantees of data communication. Because the UDP does not use "connection SOCKET," it can communicate with many other devices with the known host IP address and port number. This is a great advantage; communication with many others by using just one SOCKET, but also it has many problems such as loss of transmitted data, unwanted data received from others, etc. To avoid these problems and guarantee reliability, the host retransmits damaged data or ignores the unwanted data which is received from others. The UDP protocol supports unicast, broadcast, and multicast communication. It follows the below communication flow.

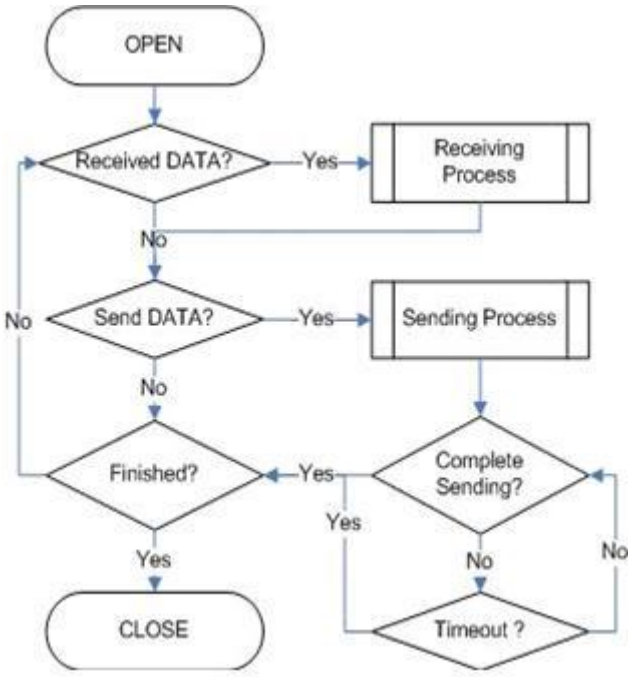


Figure 12 UDP Operation Flow

5.2.2.1 Unicast and Broadcast

The unicast is one method of UDP communication. It transmits data to one destination at one time. On the other hand, the broadcast communication transmits data to all receivable destinations by using 'broadcast IP address (255.255.255.255)'. For example, suppose that the user transmits data to destination A, B, and C. The unicast communication transmits each destination A, B, and C at each time. At this time, the ARP_{TO} can also occur when the user gets the destination hardware address of destinations A, B and C. User cannot transmit data to destinations which have ARP_{TO}.

The broadcast communication can simultaneously transmit data to destination A, B and C at one time by using "255.255.255.255" or "local address | (~subnet address)"

IP address. At this time, there is no need to get the destination hardware address about destination A, B and C, and also $ARP_{TO}$is not occurred.

Note: Broadcast IP

=> The Broadcast IP address can be obtained by performing a bit-wise logical OR operation between the bit complement of the subnet mask and the host's IP address.

ex> If IP:"222.98.173.123" and the subnet mask:"255.255.255.0", broadcast IP is "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

For the UDP data communication, SOCKET initialization is required; it opens the SOCKET. The SOCKET open process is as followed. At first, choose one SOCKET among the 8 SOCKETS of W5200, then set the protocol mode (Sn_MR(P3:P0)) of the chosen SOCKET and set the source port number Sn_PORT0 for communication. Finally, execute the OPEN command. After the OPEN command, the state of Sn_SR is changed to SOCK_UDP. Then the SOCKET initialization is complete.

```
{
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

Check the reception of UDP data from destination. User can also check for received data via TCP communication. It is strongly recommended to use the second method because of the same reasoning from TCP. Please refer to the "5.2.1.1 TCP SERVER".

```
First method :
{
  if (Sn_IR(RECV) == '1') Sn_IR(RECV) = '1'; goto Receiving Process stage;
```

```
    /* In this case, if the interrupt of Socket n-th 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

Process the received UDP data in Internal RX memory.

The structure of received UDP data is as below.

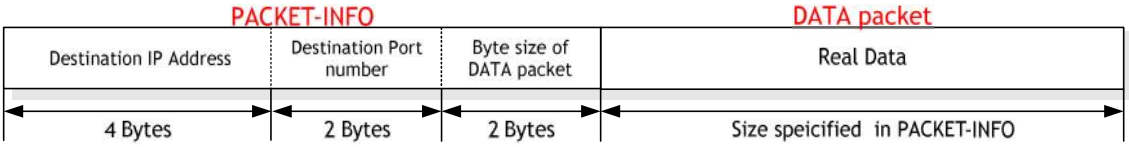


Figure 13 The Received UDP data Format

The received UDP data consists of 8bytes PACKET-INFO, and DATA packet. The PACKET-INFO contains transmitter's information (IP address, Port number) and the length of DATA packet. The UDP can receive UDP data from many others. User can classify the transmitter by transmitter's information of PACKET-INFO. It also receives broadcast SOCKET by using "255.255.255.255" IP address. So the host should ignore unwanted reception by analysis of transmitter's information.

If the DATA size of Socket n-th is larger than Internal RX memory free size, user cannot receive that DATA and also cannot receive fragmented DATA.

```
{
  /* 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

The size of DATA that the user wants to transmit cannot be larger than Internal TX memory. If it is larger than MTU, it is automatically divided by MTU unit and transmitted. The Sn_DIPR0 is set “255.255.255.255” when user wants to broadcast.

```
{
  /* 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-th Destination IP Address
     Register(Sn_DIPR), Socket n-th 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

To transmit the next data, user must check that the prior SEND command is completed. The larger the data size, the more time to complete the SEND command. Therefore, the user must properly divide the data to transmit. The ARP_{TO} can occur when user transmits UDP data. If ARP_{TO} occurs, the UDP data transmission has failed.

```
 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-th is activated, interrupt occurs. Refer to Interrupt Register(IR), Interrupt Mask Register (IMR) and Socket n-th Interrupt Register (Sn_IR). */
 }
```

Check Finished / SOCKET close

If user doesn't need the communication any more, close the Socket n-th.

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

5.2.2.2 Multicast

The broadcast communication communicates with many and unspecified others. But the multicast communication communicates with many specified others who registered at a multicast-group. Suppose that A, B, and C are registered at a specified multicast-group. If user transmits data to multicast-group (contains A), B and C also receive the DATA for A. To use multicast communication, the destination list registers to multicast-group by using IGMP protocol. The multicast-group consists of 'Group hardware address,' 'Group IP address,' and 'Group port number.' User cannot change the 'Group hardware address' and 'Group IP address.' But the 'Group port number' can be changed.

The 'Group hardware address' is selected at the assigned range (From "01:00:5e:00:00:00"to "01:00:5e:7f:ff:ff") and the 'Group IP address' is selected in D-class IP address (From "224.0.0.0" to "239.255.255.255", please refer to the website; http://www.iana.org/assignments/multicast-addresses).

When selecting, the upper 23bit of 6bytes 'Group hardware address' and the 4bytes 'Group IP address' must be the same. For example, if the user selects the 'Group IP address' to "244.1.1.11," the 'Group hardware address' is selected to "01:00:5e:01:01:0b." Please refer to the "RFC1112" (http://www.ietf.org/rfc.html).

In the W5200, IGMP processing to register the multicast-group is internally (automatically) processed. When the user opens the Socket n-th with multicast mode, the "Join" message is internally transmitted. If the user closes it, the "Leave" message is internally transmitted. After the SOCKET opens, the "Report" message is periodically and internally transmitted when the user communicates.

The W5200 support IGMP version 1 and version 2 only. If user wants use an updated version, the host processes IGMP directly by using the IPRAW mode SOCKET.

SOCKET Initialization

Choose one SOCKET for multicast communication among 8 SOCKETS of W5200. Set the Sn_DHAR0 to 'Multicast-group hardware address' and set the Sn_DIPR0 to 'Multicast-group IP address.' Then set the Sn_PORT0 and Sn_DPORT0 to 'Multicast-group port number.' Set the Sn_MR(P3:P0) to UDP and set the Sn_MR(MULTI) to '1.' Finally, execute OPEN command. If the state of Sn_SR is changed to SOCK_UDP after the OPEN command, the SOCKET initialization is completed.

```
{
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-th 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

Refer to the "5.2.2.1 Unicast & Broadcast."

Receiving process

Refer to the "5.2.2.1 Unicast & Broadcast."

Check send data / Sending Process

Since the user sets the information about multicast-group at SOCKET initialization, user does not need to set IP address and port number for destination any more. Therefore, copy the transmission data to internal TX memory and executes SEND command.

```
 {
  /* 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

Since the host manages all protocol process for data communication, timeout cannot occur.

```
{
/* 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

Refer to the "5.2.2.1 Unicast & Broadcast."

5.2.3 IPRAW

IPRAW is data communication using TCP, UDP, and IP layers, which are the lower protocol layers. IPRAW supports IP layer protocol such as ICMP (0x01) and IGMP (0x02) according to the protocol number. The 'ping' of ICMP or IGMP v1/v2 is already included in W5200 by hardware logic. But if the user needs, the host can directly process the IPRAW by opening the Socket n-th to IPRAW. In the case of using IPRAW mode, user must set the protocol number field of the IP header to what the user wants to use. The protocol number is defined by IANA. Refer to the web (http://www.iana.org/assignments/protocol-numbers). The protocol number must be configured to Sn_PROTO before 'SOCKET open.' In IPRAW mode, W5200 does not support TCP (0x06) or UDP (0x11) protocol number. The SOCKET communication of IPRAW mode only allows the communication of an assigned protocol number. The ICMP SOCKET cannot receive unassigned protocol data except assigned protocol data such as IGMP.

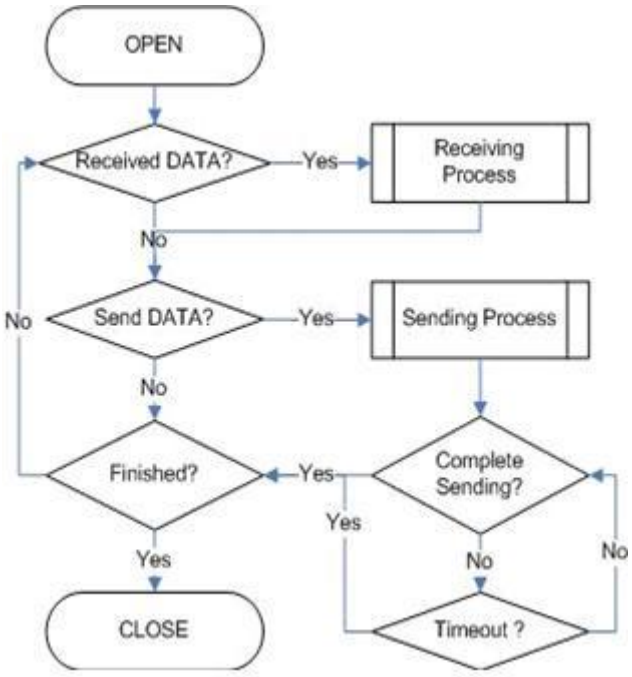


Figure 14 IPRAW Operation Flow

SOCKET Initialization

Select the SOCKET and set the protocol number. Then set the Sn_MR(P3:P0) to IPRAW mode and execute 'OPEN' command. If the Sn_SR is changed to SOCK_IPRAW after the 'OPEN' command, the SOCKET initialization is completed.

```
{
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

Refer to the "5.2.2.1 Unicast & Broadcast."

Receiving process

Process the IPRAW data which is received in internal RX memory. The structure of received IPRAW data is as below.

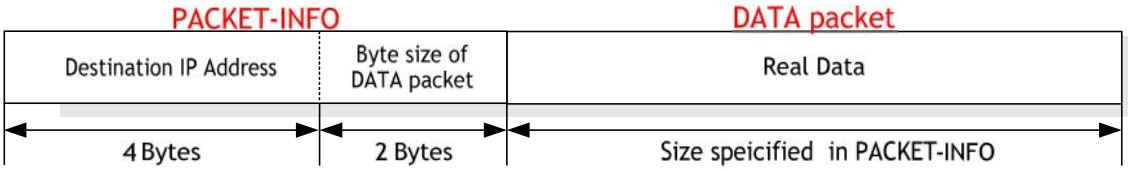


Figure 15 The receive IPRAW data Format

The IPRAW data consists 6 bytes PACKET-INFO and DATA packet. The PACKET-INFO contains information about the transmitter (IP address) and the length of the DATA-packet. The data reception of IPRAW is the same as UDP data reception except processing the port number of the transmitter in UDP PACKET-INFO. Refer to the "5.2.2.1 Unicast & Broadcast." If the transmitted DATA size is larger than RX memory free size of Socket n-th, user cannot receive that DATA and also cannot receive fragmented DATA.

Checks send data / Sending process

The size of DATA which user wants to transmit cannot be larger than Internal TX memory and default MTU. The transmission of IPRAW data is the same as transmission of UDP data except setting 'Destination port number'. Refer to the "5.2.2.1 Unicast & Broadcast."

Complete sending / Timeout

Same as UDP, please refer to the "5.2.2 UDP."

Check finished / SOCKET closed

Same as UDP, please refer to the "5.2.2 UDP."

5.2.4 MACRAW

The MACRAW communication is based on Ethernet MAC, and it can flexibly use upper layer protocol to suit the host's needs.

The MACRAW mode can only be used with a SOCKET. If the user uses the SOCKET in MACRAW mode, not only can it use the SOCKET1~7 in the 'Hardwired TCP/IP stack', but it can also be used as a NIC (Network Interface Controller). Therefore, any SOCKET1~7 can be used with 'Software TCP/IP stack'. Since the W5200 supports 'Hardwired TCP/IP stack' and 'Software TCP/IP stack', it calls 'Hybrid TCP/IP stack'. If user wants more SOCKETs beyond the supported 8 SOCKETS, the SOCKET in which the user wants high performance should be utilizing the ''Hardwired TCP/IP stack', and the others should be using 'Software TCP/IP stack' by MACRAW mode. So it overcomes the limited capacity of 8 SOCKETS. The SOCKET of MACRAW mode can process all protocols except using in SOCKET1~7. Since the MACRAW communication is pure Ethernet packet communication (there is no other processing), the MACRAW designer should use the 'Software TCP/IP stack' to process the protocol. The MACRAW data should basically contain the 6bytes of 'Source hardware address', 6bytes of 'destination hardware address' and 2bytes of 'Ethernet type' because it is based on Ethernet MAC.

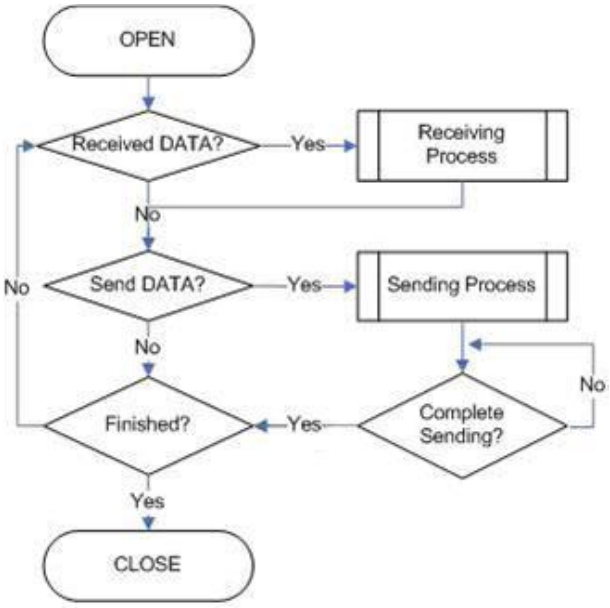


Figure 16 MACRAW Operation Flow

SOCKET Initialization

Select the SOCKET and set the SN_MR(P3:P0) to MACRAW mode. Then execute the 'OPEN' command. After the 'OPEN' command, if the Sn_SR is successfully changed to 'SOCK_MACRAW', the SOCKET initialization is completed. Since all information about communication (Source hardware address, Source IP address, Source port number, Destination hardware address, Destination IP address, Destination port number, Protocol header, etc.) is in the 'MACRAW data', there is no more register setting.

```
{
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

Refer to the "5.2.2.1 Unicast & Broadcast."

Receiving process

Process the MACRAW data of the SOCKET which received it in internal RX memory.

The structure of the MACRAW data is as below:

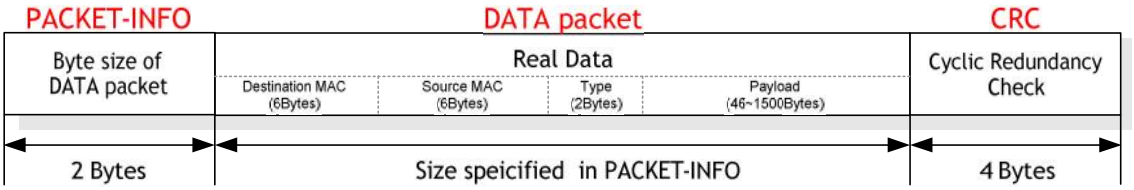


Figure 17 The received MACRAW data Format

The MACRAW data consists of 'PACKET-INFO,' 'DATA packet' and 4bytes CRC. The 'PACKET-INFO' is the length of the DATA packet. The 'DATA packet' consists of 6bytes 'Destination MAC address,' 6bytes 'Source MAC address' and 2bytes 'Type,' 46~1500 bytes 'Payload.' The 'Payload' of DATA packet consists of Internet protocol such as ARP, IP according to the 'Type.' The details of 'Type' please refer to the web:

(http://www.iana.org/assignments/ethernet-numbers)

```
{
 /* 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>

If the free size of the internal RX memory is smaller than the MACRAW data, a problem may occasionally occur where some parts of that PACKET-INFO and DATA packet are stored to the internal RX memory. Since the problem occurs as an analysis error for PACKET-INFO, it cannot process the MACRAW data correctly. The closer the internal RX memory is to being full, the higher the probability is for an error to occur. This problem can be resolved if user allows some loss of the MACRAW data.

The solution is as follows:

- Process the internal RX memory as fast as possible to prevent that it closes to full.
- Reduce the receiving load by reception only its MACRAW data by setting the MF (MAC Filter) bit of S0_MR in sample code of SOCKET initialization.

```
{
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;
}
```

▪If the free size of the internal RX memory is smaller than '1528 - Default MTU(1514)+PACKET INFO(2) + DATA packet(8) + CRC(4)', close the SOCKET and process all received data. Then reopen the SOCKET. After closing the SOCKET, the received MACRAW data from closing time can be lost.

```
{
/* 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

The size of the data which the user wants to transmit cannot be larger than the internal TX memory and default MTU. The host generates the MACRAW data in the same format as the "Receiving process" data packet, and transmits it. At this time, if the size of the generated data is smaller than 60bytes, the transmitted Ethernet packet internally fills to 60bytes by "Zero padding" and then it is transmitted.

```
 {
   /* 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

Since the host manages all protocol processors to communicate, the timeout can notoccur.

```
{
/* 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

Refer to the "5.2.2.1 Unicast & Broadcast."

6 External Interface

For the communication with MCU, W5200 provides SPI I/F modes. For the communication with Ethernet PHY, MII is used.

6.1 SPI (Serial Peripheral Interface) mode

Serial Peripheral Interface Mode uses only four pins for data communication.

Four pins are nSCS, SCLK, MOSI, and MISO.

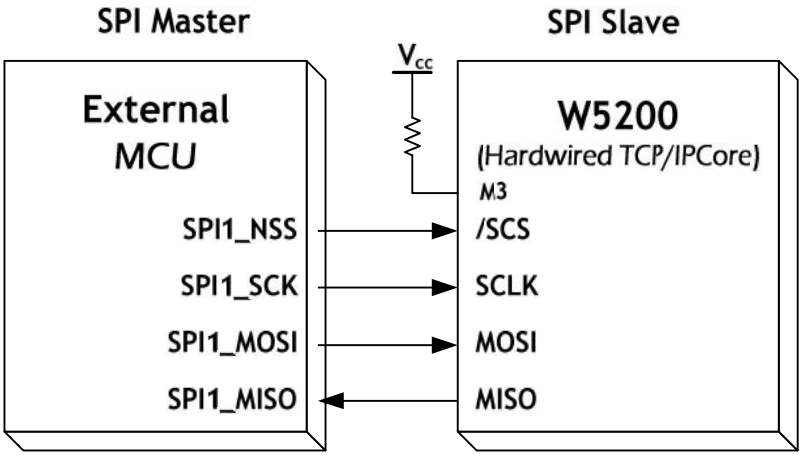


Figure 18 SPI Interface

6.2 Device Operations

W5200 is controlled by a set of instruction that is sent from a external host , commonly referred to as the SPI Master. The SPI Master communicates with W5200 via the SPI bus, which is composed of four signal lines: Slave Chip Select (nSCS), Serial Clock (SCLK), MOSI (Master Out Slave In) and MISO (Master In Slave Out).

The SPI protocol defines four modes for its operation (Mode 0-3). Each mode differs according to the SCLK polarity and phase - how the polarity and phase control the flow of data on the SPI bus. The W5200 operates as SPI Slave device and supports the most common modes - SPI Mode 0 and 3.

The only difference between SPI Mode 0 and 3 is the polarity of the SCLK signal at the inactive state. With SPI Mode 0 and 3, data is always latched in on the rising edge of SCLK and always output on the falling edge of SCLK.

6.3 Process of using general SPI Master device

1. **Configure Input/Output direction on SPI Master Device pins.**
2. **Configure nSCS as 'High' on inactive**
3. **Write target address for transmission on SPDR register (SPI Data Register).**
4. **Write OP code and data length for transmission on SPDR register.**
5. **Write desired data for transmission on SPDR register.**
6. **Configure nSCS as 'Low' (data transfer start)**
7. **Wait for reception complete**
8. **If all data transmission ends, configure nSCS as 'High'**

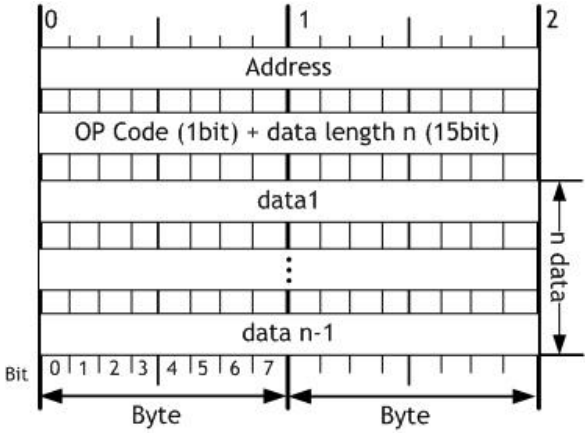


Figure 19 W5200 SPI Frame Format

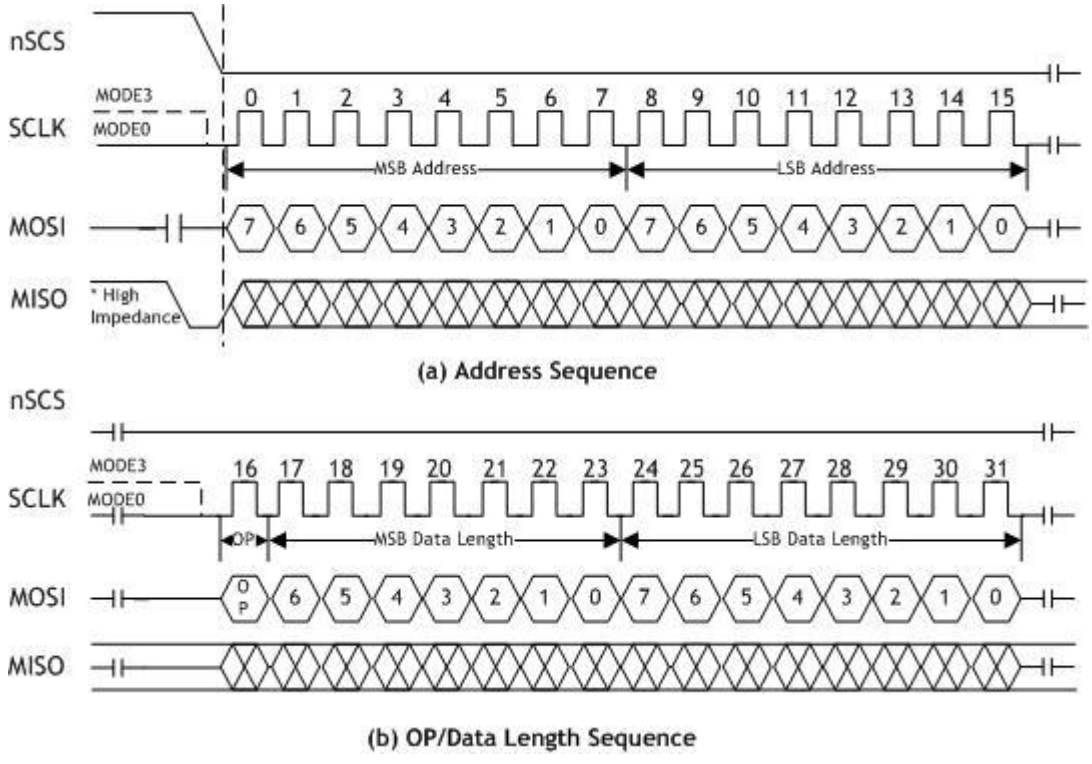


Figure 20 Address and OP/DATA Length Sequence Diagram

READ Processing

The READ Processing Sequence Diagram is shown in Figure 20. The READ processing is entered by driving nSCS low, followed by the Address, the OP code, the Data Length and the Dummy data byte on MOSI. And then W5200 read the data byte on MISO. The Address, the OP/Data Length Sequence Diagram and the Data are shown in Figure 19. The OP code (OP) is defined type of the READ OP and WIRTE OP. On OP = 0, the read operation is selected. Otherwise, On OP = 1, the write operation is selected.

In W5200 SPI mode, the Byte READ processing and the burst READ processing are provided. The Byte READ processing takes 4 instructions which is consist of the 16-bit Address, the 1-bit OP code(0x0), the 15-bit Data length and 8-bit Data. Otherwise, The Burst READ processing only takes the Data instruction after the setting of the burst read processing. To distinguish between the Byte READ and the burst READ processing, the Data length is used. If the Data length is '1,' the Byte READ processing is operated. Otherwise, the Burst READ Processing is operated when the Data length is more than two. The MISO pin should be selected by driving MISO low after the falling edge of the nSCS.

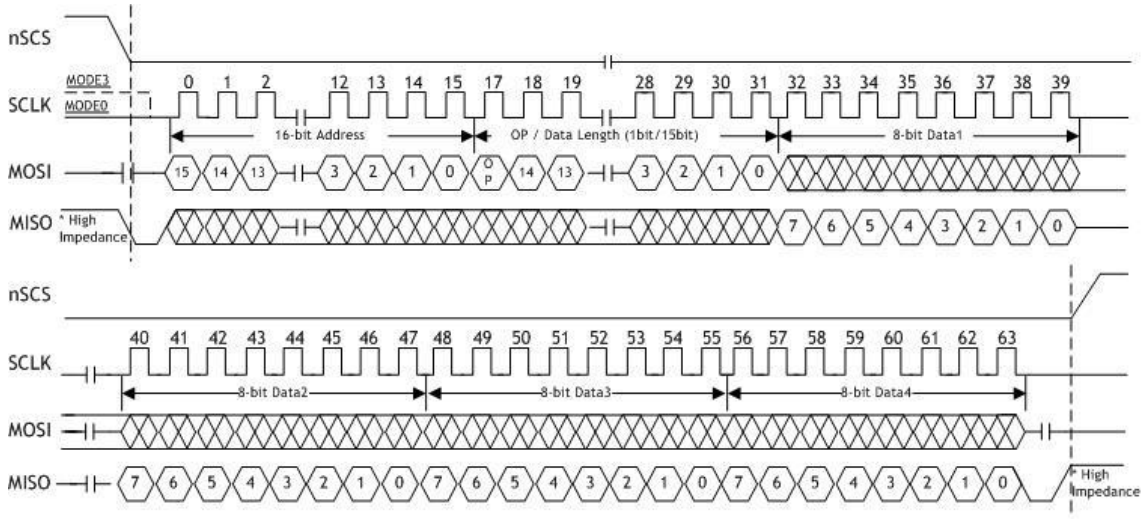


Figure 21 READ Sequence

```
/* 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

The WRITE Processing Sequence Diagram is shown in Figure 21.The WRITE processing is entered by driving nSCS low, followed by the Address, the OP code, the Data Length, and the Data byte on MOSI.

In W5200 SPI mode, the Byte WRITE processing and the Burst WRITE processing are provided. The Byte WRITE processing takes 4 instructions which is consist of the 16-bit Address, the 1-bit OP code(0x1), the 15-bit Data length and 8-bit Data. Otherwise, The Burst WRITE processing only takes the Data instruction after the setting of the Burst WRITE processing. To distinguish between the Byte WRITE and the Burst WRITE processing, the Data length is used. If the Data length is '1,' the Byte WRITE processing is operated. Otherwise, the Burst WRITE Processing is operated when the Data length is more than two. The MOSI pin should be selected by driving MOSI low after the falling edge of the nSCS.

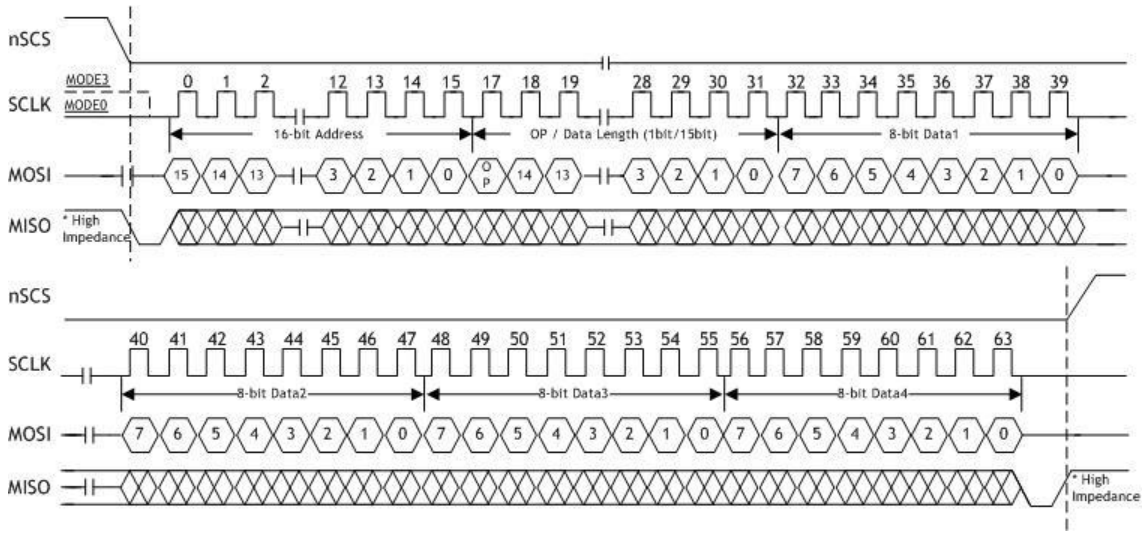


Figure 22 Write Sequence

```
/* 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
VDD	DC Supply voltage	Junction temperature is from -55°C to 125°C	2.97		3.63	V
VIH	High level input voltage		2.0		5.5	V
VIL	Low level input voltage		- 0.3		0.8	V
VOH	High level output voltage	IOH = 4 ~8 mA	2.4			V
VOL	Low level output voltage	IOL = 4 ~8mA			0.4	V
II	Input Current	VIN = VDD			±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
Loss 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

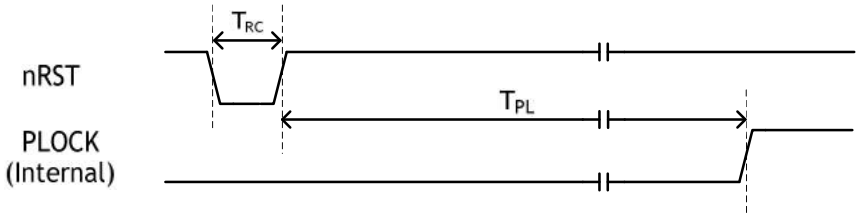


Figure 23 Reset Timing

Symbol	Description	Min	Max
TRC	Reset Cycle Time	2 us	-
TPL	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

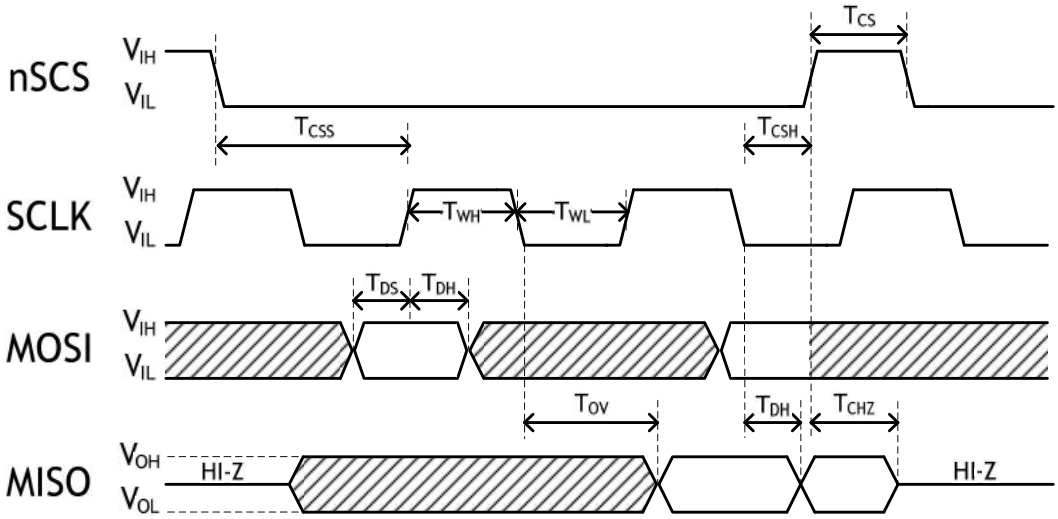


Figure 24 SPI Timing

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

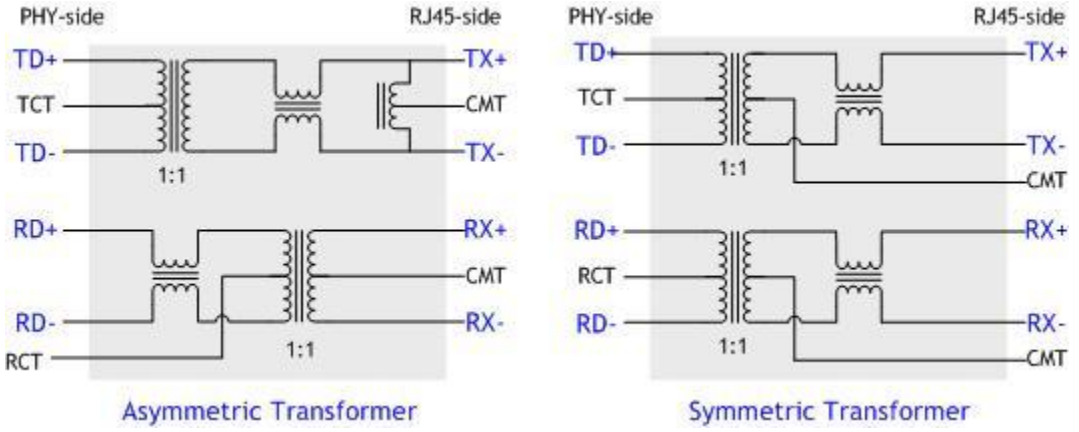


Figure 25 Transformer Type

In case of using internal PHY mode, be sure to use symmetric transformer in order to support Auto MDI/MDIX(Crossover).

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}) – Temperature Max (Ts_{max}) – Time (ts_{min} to ts_{max})	 150 °C 200 °C 60-120 seconds
Time maintained above: – Temperature (TL) – Time (tL)	 217 °C 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.

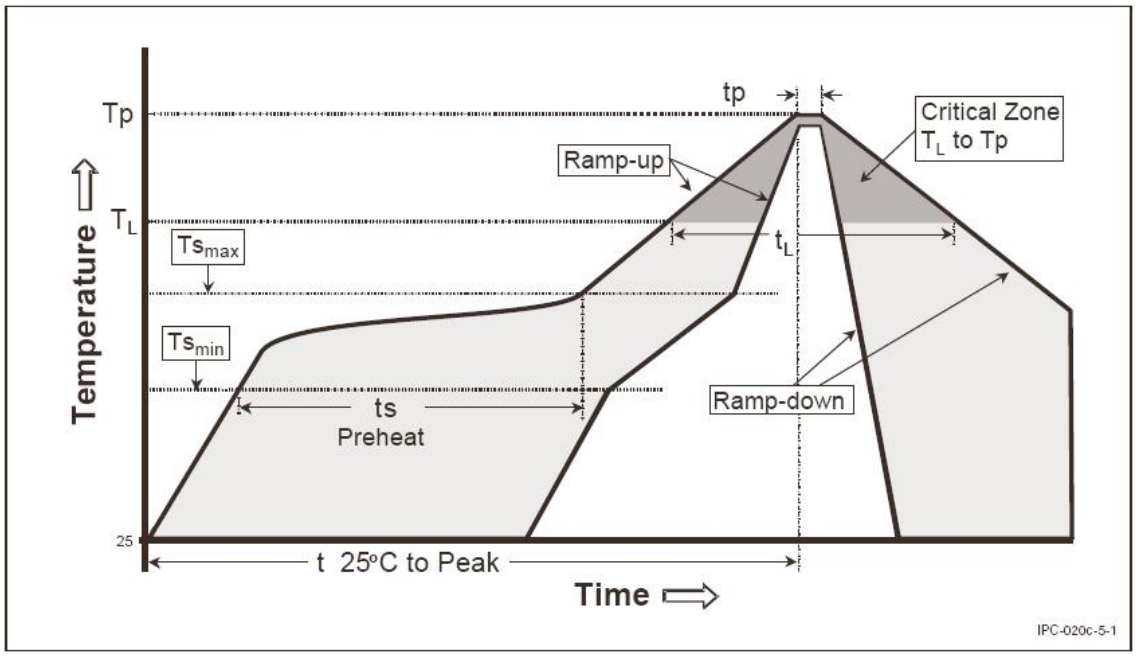
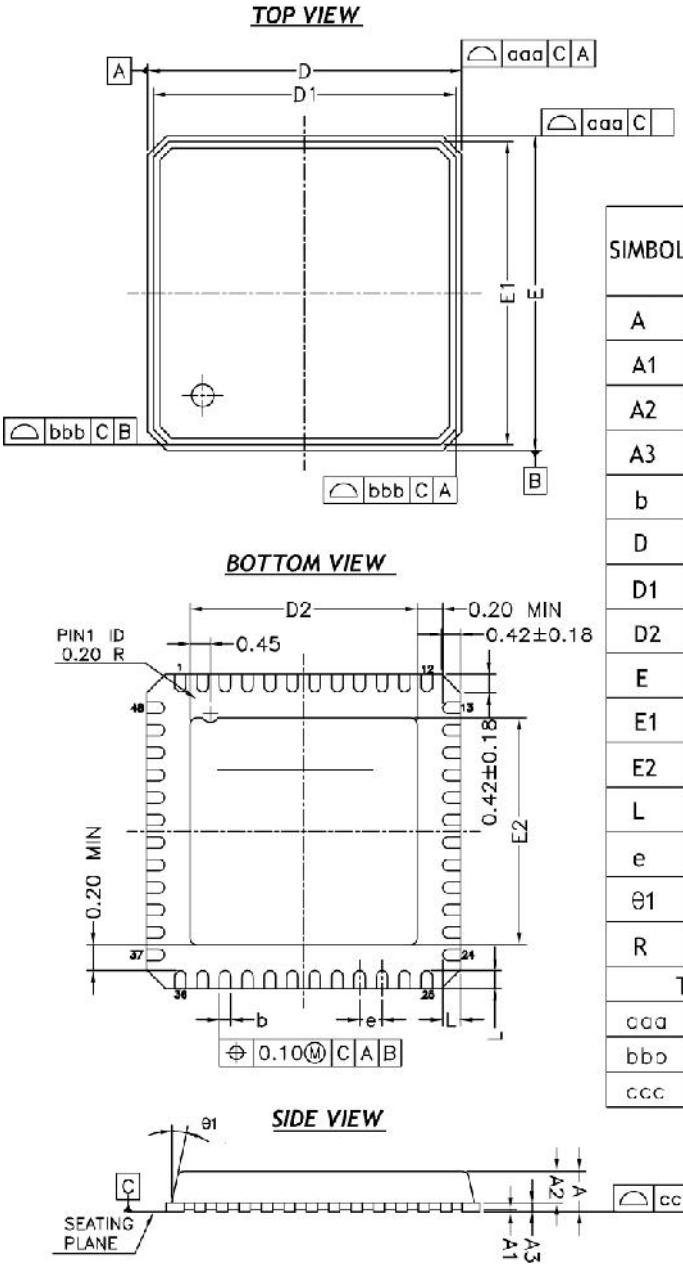


Figure 26 IR Reflow Temperature

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 27 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. 1.0.0	Mar2011	Released with W5200 Launching
Ver. 1.1.0	13MAR2011	Changed IMR address (0x16 to 0x36) (P.14, P.18) Changed IMR2 address (0x36 to 0x16) (P.14, P.22)
Ver. 1.2.0	22APR2011	Fixed the description of RSV at 1.3 Miscellaneous Signals (P.10) Fixed the values of typical at 7.3 power dissipation (P.77) Added the values of maximum at 7.3 power dissipation (P.77) Fixed the description of RSV at 1.3 Miscellaneous Signals (removed PIN 31, P.10)
Ver. 1.2.1	2AUG2011	Fixed the description of READ processing at 6.3 Processing of using general SPI Master device (P.73)
Ver. 1.2.2	25NOV2011	Fixed the Block Diagram (P.4)
Ver. 1.2.3	3FEB2012	Added the Figure of XTAL_VDD at 1.4 Power Supply Signals (P.11)
Ver. 1.2.4	16FEB2012	Fixed the Pin names and sequence diagrams of READ processing and WRITE processing at 6.3 Processing of using general SPI Master device (P.73, 75)
Ver. 1.2.5	12APR2012	Corrects some miss phrase and words. Corrects the figure of SPI frame format (P.72)
Ver. 1.2.6	4JUN2012	Corrects some miss phrase and words. Corrects the table and figure of RX, TX memory size format (P.35,41)

Sales & Distribution: sales@wiznet.co.kr
For more information, visit our website at http://www.wiznet.co.kr