R1610/R1620-B Fast Ethernet MAC Programming Draft

Version 1.00
October 14, 2002

Content Index:

1. How to initialize RDC MAC

This chapter, we describe some necessary operation.

1-1 How to reset MAC controller

Software reset is a necessary action to make sure your chip works on the default state. Software reset sequence is 1). Set MSCTL register bit0. 2). Wait MSCTL bit0 clear. 3) Reset complete.

<Example>

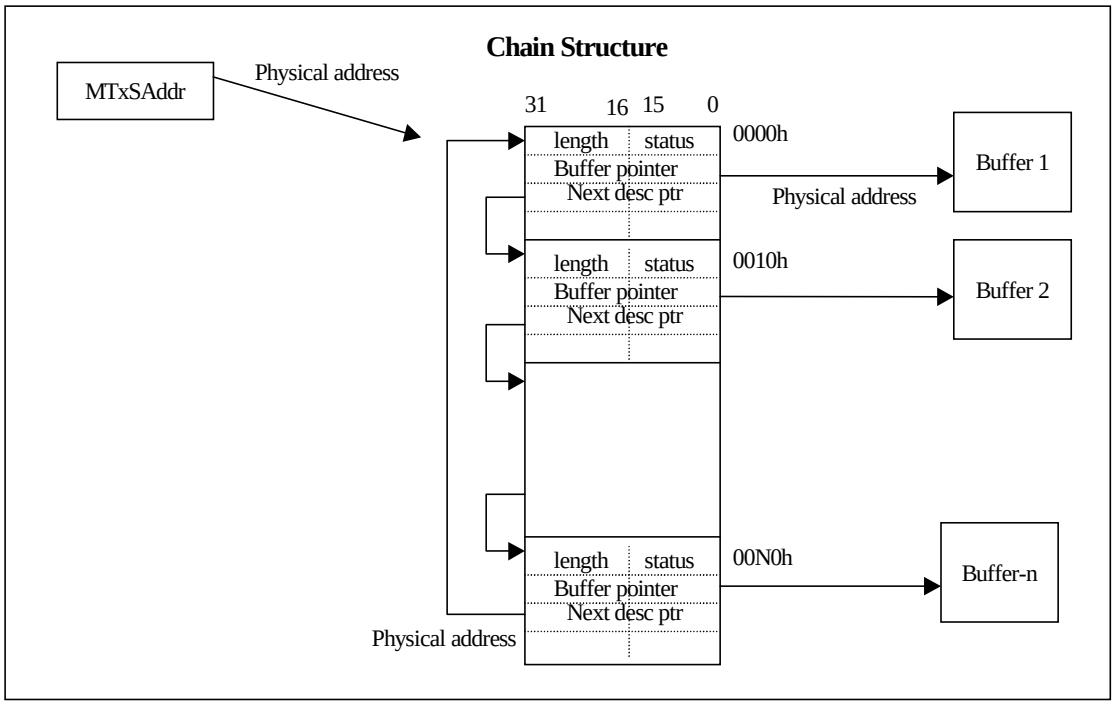
```
; Issue RESET command
      mov       ax, 0001h
      mov       dx, MSCTL
      out       dx, ax

; Wait RESET complete
reset_wait:
      in        ax, dx
      test      ax, 0001h
      jnz       reset_wait

; RESET complete
```

1-2 How to initialize TX descriptor chain

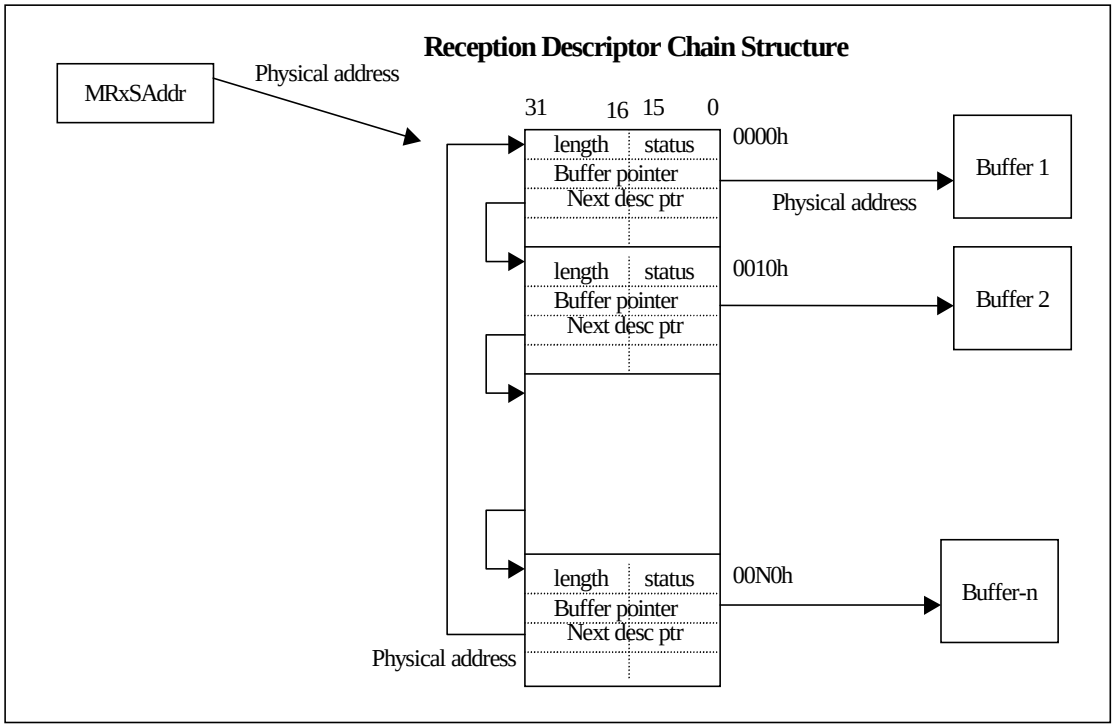
For data transmission, driver needs to prepare a descriptor chain structure like as figure. Buffer is not necessary. If you want to hook your sent buffer directly, you don't need to allocate buffer and initialize <buffer pointer>. <status> OWN_BIT(bit 15) must be zero because you have no data to send so far. About descriptor format and MTxSAddr register, please see R1620 data sheet.



Descriptor must be double word aligned. TX buffer can be at any byte address. MTxSAddr, Buffer_pointer, Next_desc_ptr must be the physical address.

1-3 How to initialize RX descriptor chain

For data reception, driver needs to prepare a descriptor chain structure like as figure. Buffer is necessary. But you can dynamically allocate. Or like as figure, statically allocate buffer. <status> OWN_BIT(bit 15) can be set if you allocate buffer. You must specify the buffer size into register MRBS. About descriptor format, MRxSAddr and MRBS register, please see R1620 data sheet.



RX Descriptor and buffer must be double word aligned.
MRxSAddr, Buffer_pointer, Next_desc_ptr must be the physical address.

1-4 How to initialize MAC register

There are some registers must be initialized before driver activates MAC. 1) Register MTxSHAddr:MTxSLAddr must point to the start address of the transmission descriptor chain. 2) Register MRxSHAddr:MRxSLAddr must point to the start address of the reception descriptor chain. 3) Decide register MCTL <full duplex bit>. You must depend on PHY's status. If PHY is on full duplex mode, this bit must set. Others, this bit must clear. 4) If your MAC address and multi-cast address are less than or equal 4 groups, you can directly use register MIDxx. If over 4 groups, you must set hash table by register MHMARxx and set MCTL bit to enable hash function. 5) If you want to use the interrupt driven to receive packets, please register your interrupt handler before you turn on register MintMsk. If you use the polling method for transmission and reception, you need to turn off MintMsk. 6) Decide MAC operation mode by register MCTL. Please see R1620 data sheet and set the function bit that you want to use. 7) Set MCTL bit0 RCV_EN and bit12 XMT_EN to activate MAC transmission and reception.

<Example>
Map MAC address 00-01-02-03-04-05 to MAC0 MID0

```
        mov     dx, 0FD68h      ; MID0 low byte offset
        mov     ax, 0100h
        out     dx, ax
        mov     dx, 0FD6Ah      ; MID0 middle byte offset
        mov     ax, 0302h
        out     dx, ax
        mov     dx, 0FD6Ch      ; MID0 high byte offset
        mov     ax, 0504h
        out     dx, ax
```

2. Transmit packets

Transmit a packet include two part. One is prepare data and send. Another is sent status check.

2-1 How to transmit packet

1) Get the first free transmission descriptor
2) If there is a buffer hook on this descriptor, you can copy the send data into this buffer. If not, you can directly hook your send data buffer into descriptor <DaPtH:DaPtL>. But you must be care, <DaPth:DaPtL> must be the physical address.
3) Set the send data length into descriptor <Length>. If your send data length is less than 60 byte, please put 60 into this field because MAC doesn't do the padding action.
4) If want to disable CRC padding, please let <Status> bit13 DISCRC set. If DISCRC is 0, MAC will automatically append CRC at the packet end.
5) Set <Status> OWN_BIT. OWN_BIT is set, it means this descriptor belongs to MAC. OWN_BIT is clear, it means this descriptor belongs to host.
6) Issue transmission-polling command. Out register XMTPC to signal MAC to send. Please note MCTL bit 12 must be set, otherwise MAC do not send.

2-2 how knows the transmitted status

Driver how to know MAC sent status? MAC will clear descriptor <status> OWN_BIT after transmit complete and fit the sent status into <status> field. After complete above action, MAC will fetch the next descriptor and check <status> OWN_BIT. If OWN_BIT is set, MAC will send it. If OWN_BIT is clear, MAC will be idle state to wait next polling command.

So you can use the polling method to the sent descriptor <status> OWN_BIT. If this bit is 0, it means this data was sent completely. Then driver can check <status> to know the sent status.

If you want to use the interrupt driven method to check send status, you can register an interrupt handler and turn on register MintMsk <XPF_EN>. MAC will generate an interrupt and set register MintSt <XPKT_FINISH> after sent completely.

3. Receive packets

MAC will clear the reception descriptor <status> OWN_BIT after received completely. Now the received data is in the address that the descriptor <DaPtL> point. The received data length is in the descriptor <Length> field. The received data status is in the descriptor <Status> field. MAC will fetch the next descriptor after complete this reception. If next descriptor <status> OWN_BIT is 1, MAC will receive the next packet. If next descriptor <status> OWN_BIT is 0, MAC will enters the waiting state. Next packet coming will trigger MAC to check descriptor available again.

3-1 How to receive packet

The simple method is the polling method. Driver keep to check the reception descriptor <status> OWN_BIT, until OWN_BIT is 0. It means a packet received already.
Another method, driver can register an interrupt and turn on register MintMsk <RPF_EN>. MAC will generate an interrupt and set MintSt <RPKT_FINISH> after receive a packet.

3-2 How to re-use reception descriptor

Driver must take care of the data of received descriptor. Driver can copy the data from the received buffer to driver private buffer. Then set descriptor <status> OWN_BIT to 1 to let this descriptor can receive data again.

Another method, driver can directly take off this buffer from the descriptor. Then allocated a new buffer and hook this new buffer into the descriptor. Finally, set descriptor <status> OWN_BIT to 1.

4. How to access PHY register

R1620 MAC supports the useful registers to read or write PHY registers. User only needs to specify which PHY registers those user wants to read or write and issues a read or write command. R1620 MAC will do the all detail steps.
MMDIO register offset 20h, MII command register
MMIIRDATA register, offset 24h, data read from PHY
MMIIWDATA register, offset 28h, data write to PHY
About above register description, please refer R1620 data sheet.

4-1 How to read PHY register

1). Put the PHY register address that you want to read into MMDIO bit4:0, Put PHY address into MMDIO bit12:8. Set MII read command on MMDIO bit13. 2). Wait read complete after MMDIO bit13 is 0. 3). Now PHY data is on MMIIRDATA register.

<Example>
Assume PHY address is 01h, read PHY register 02h

```
; Set PHY address, register address and issue read command
      mov       ax, 2102h
      mov       dx, MMDIO
      out       dx, ax

; Wait read command complete
read_wait:
      in        ax, dx
      test      ax, 2000h
      jnz       read_wait

; PHY register data on MMIIRDATA
      mov       dx, MMIIRDATA
      in        ax, dx
```

4-2 How to write PHY register

1). Put the written data into MMIIWDATA. 2). Put the PHY register address that you want to write into MMDIO bit4:0, Put PHY address into MMDIO bit12:8. Set MII write command on MMDIO bit14. 3). Wait the write command complete after MMDIO bit14 is 0.

<Example>

Assume PHY address is 01h, write 0061h to PHY register 04h

```
; Put the written data to MMIIWDATA
        mov     ax, 0061h
        mov     dx, MMIIWDATA
        out     dx, ax
; Set PHY address, register address and issue write command
        mov     ax, 4104h
        mov     dx, MMDIO
        out     dx, ax

; Wait write command complete
write_wait:
        in      ax, dx
        test    ax, 4000h
        jnz     write_wait
```

5. How to use hash function for address filter

R1620 has 4 set address. MID 0,1,2,3, every one can be a multi-cast address or uni-cast address. If your address is over this, you can use hash function for multi-cast address.
R1620 MAC supports 4 hash table registers and each register is 16 bits. So the hash table is 64-bits width. It needs 6 bit to index this hash table.

5-1 How to enable hash function

Please note hash function is only for multi-cast address. You just need to set MCTL FC_EN bit to enable the function. This function is disabled after reset.

<For example>

```
mov     dx, MCTL
in      ax, dx
or      ax, 0100h
out     dx, ax
```

5-2 How to calculate the hash table index from your multi-cast address

You can use the following program to get a 6-bit hash index from a multi-cast address. Then use this hash index to map hash table register.

Hash index algorithm:

Buffer: 6 byte multi-cast address

```
{
   unsigned long int  Crc, Carry;
   unsigned int       i, j, Hash_Index;
   unsigned char      CurByte;

   Crc = 0xffffffffUL;

   for (i = 0; i < 6; i++) {
      CurByte = Buffer[i];

      for (j = 0; j < 8; j++) {
         Carry = ((Crc & 0x80000000UL) ? 1 : 0) ^ (CurByte & 0x01);
         Crc <<= 1;
         CurByte >>= 1;

         if (Carry) {
            Crc = (Crc ^ 0x04c11db6UL) | Carry;
         }
      }
   }

   Hash_Index = (unsigned int) ( (Crc>>26) & 0x3F
}
```

5-3 How to map Hash register

Hash index 00 maps to Hash Table word1 bit 00
Hash index 15 maps to Hash Table word1 bit 15
Hash index 16 maps to Hash Table word2 bit 00
Hash index 31 maps to Hash Table word2 bit 15
Hash index 32 maps to Hash Table word3 bit 00
Hash index 47 maps to Hash Table word3 bit 15
Hash index 48 maps to Hash Table word4 bit 00
Hash index 63 maps to Hash Table word4 bit 15

6. Flow control function

R1620 MAC supports 802.3X flow-control function for full duplex mode.

6- 1 How to enable flow control function

Enable flow control function, you just need to set MCTL FC_EN bit. And this function is only work on full duplex mode. If you turn on this bit on half duplex mode, it does not work.

<For example>

```
mov     dx, MCTL
in      ax, dx
or      ax, 0200h
out     dx, ax
```

6-2 MAC how to handle the received pause frame

When MAC receive a pause frame with time not zero, MAC will temporary stop to send after current transmitting completely. Until timeout or receive a pause frame with time 00, MAC will keep to send.

6-3 MAC how to send pause frame and how to stop

MRDCR(1Ah) bit7..0 is RXDESCPAN, RX descriptor available count for flow-control. Bit15..8 is RXPT, RX descriptor threshold. Please see data sheet for details.

MAC will automatically send the pause frame when RXDESCPAN is less or equal RXPT. MAC automatically decreases RXDESCPAN after received a packet. If RXDESCPAN equals to RXPT, MAC will set MSCTL TPF bit then send a pause frame with time FFFFh. Please note transmission must be enabled (MCTL XMT_EN is set), otherwise MAC can't send the pause frame. MAC will keep sending the pause frame after timeout. Driver need to signal MAC to increase RX descriptor count after RX descriptor is available. MAC will stop to send PF and clear TPF flag when RXDESCPAN is larger than RXPT.

MRDCR register is a special register. You can specify RXDESPAN and RXPT into this register when RX_EN is 0. If RX_EN is 1 (RX machine activates), MAC will automatically increase 1 to RXDESCPAN when driver does an I/O write to this register. Any time, driver does an I/O read to this register to get the current value.

<For example>

```
; One RX descriptor is available again & Signal MAC to increase RXDESCPAN
mov     RX_descripotr_OWN, 1

mov     dx, MRDCR
out     dx, ax
```

7. How to reduce interrupt amount

7-1 Host interrupt

R1620 MAC 1 and 2 share the interrupt 4. You can check the interrupt status register (FF30h) to identify this interrupt caused by MAC1 or MAC2. If ISR bit8 is set, it means this interrupt cause by MAC1. If ISR bit9 is set, it means this interrupt cause by MAC2. Those two bits, only one can set because one time service an interrupt request. If bit4 is set, it means MAC1 has an interrupt request. If bit5 is set, it means MAC2 has an interrupt request. Those two bits can set both when MAC1/2 have the interrupt request at the same time. Bit 5..4 response the MAC current interrupt request status. Bit 9..8 response the MAC interrupts in service.

7-2 MAC interrupt

When packet transmit and receive too fast, it will generate many interrupts. If interrupts are too many, OS has no time to service other process because OS always service interrupts. When this condition happens, R1620 MAC has a good method to reduce transmit and receive interrupt amount.

There are two interrupt control registers. One is for transmission and another is for reception.

How to use the interrupt control register to reduce the interrupt amount. There are two fields ,INTC and TIMER, in this control register.

If INTC is 0, it means disable interrupt control function. MAC will generate interrupt after a transmission or reception complete.

If INTC is not zero, it means MAC will generate an interrupt after transmit or receive packets reach INTC.

For example, TXINTC is 6. MAC will generate an interrupt after transmit 6 packets. In this case, we reduce 5 interrupts. OS get more time to service other process. Do not waste too much time on content switch.

TIMER is another way to generate interrupt. In upper case, if MAC only send

3 packets, there is no data to send a long time. MAC will generate an interrupt after TIMER is timeout. TIMER is based on TX clock or RX clock. Driver can specify the timeout time by program TIMER field.

Timer activated by packet transmission or reception. So if no packet transmits or receives, the timer doesn't work and doesn't generate interrupt.

8. Application Notes

8-1 Difference between R1620-B and R88xx

1. Clock pre-scalar register (FFE2h) for SDRAM re-fresh time
   At 75Mhz-host clock, suggestion value is 03E8h
   Re-fresh time is 13.3us

2. RCU register (FFE4h) for SDRAM re-fresh function
   Enable SDRAM re-fresh function
   out FFE4h, 8000h

3. Arbiter Control register (FEF0h) for bus utilization
   Suggestion value is A0h to enable R/W re-order and post-write.
   out FEF0h, A0h

4. Serial Port Control (16550)
   R1620-B UART with 16 byte FIFO.
   <For example>
   Host clock 7.5Mhz, Baud Rate 19200
   Divisor = (host clock) / 16 / (baud rate) = 24.4
   out FF80h, 24 ; DLL, 16550A Divisor Latch low
   out FF82h, 00 ; DLM, 16550A Divisor Latch high
   PS. Please note, LCR(FF86h) bit7 (DLAB) must be set before program DLL and DLM.

5. LMCS block size: 1M byte minus UMCS block size.
   <For example>
   UMCS is F03Bh, UMCS block size is 64K byte
   LMCS block size is 960K byte

6. Timer 0, 1, 2 and Watchdog timer
   The timer source is the host clock divided 2.
   <For example>
   The host clock is 100Mhz, The timer clock source is 50Mhz.

7. UMCS and PCS support up to 15 wait-state
   Please see R1620 specification for details.

8. Processor Release Level Register (PRLR, FFF4h)
   R1620-B PRLR is 10D9h

8-2 Difference between R1620-B and R1620-A

1. MCR0 (MAC Control Register0) bit13
   R1620-B: MCR0 bit13 reserved
   R1620-A: MCR0 bit13 CRC append when set

2. TX Descriptor DTST (TX Status and packet control) bit13
   R1620-B: DTST bit13 Disable append CRC field when set
   R1620-A: DTST bit13 reserved

3. UMCSR (Upper Memory Chip Select Register FFA0h) wait-state
   R1620-B: Support up to 15 wait-state, UMCSR bit3 wait state extends
   R1620-A: Support up to 3 wait-state, UMCSR bit3 reserved

4. PCSR (Peripheral Chip Select Register FFA4h) wait-state
   R1620-B: Support up to 15 wait-state, PCSR bit3 wait state extends
   R1620-A: Support up to 3 wait-state, PCSR bit3 reserved

8-3 Programming Notes

1. Be sure you don't use TX/RX interrupt control register (0Ch, 10h). If you turn on this function, it will let R1620 lose interrupt. So please set those two registers to zero.

2. Keeps ACR (Auxiliary Configuration Register FFF2h) bit7 (USIZ) and bit2 (LSIZ) when write ACR.
   <For example, Change IO size to 8 bits>

```
mov dx, 0FFF2h
in  ax, dx
and ax, 0084h     ; Save bit7 and bit2
or  ax, 0001h     ; Let IOSIZ be 8 bits
out dx, ax
```

3. After watchdog timeout, system will be reset and R1620 will re-latch AD15-0 into the RESCON register. Unfortunately, sometimes it latches the wrong data in RESCON register. Program can check WTCR (Watchdog Timer Control Register) bit13 to avoid this problem. When system is a cold boot, WTCR bit13 is "0" and program can process RESCON register. When system is re-start by the watchdog timeout, WTCR bit13 is "1" and program can skip RESCON check.

4. TX and RX descriptor must be aligned at D-word boundary (4 byte). RX buffer must be aligned at D-word too. TX buffer can be at any byte boundary.

5. Register FFE2h (Clock Pre-scalar Register) and FFE4h (Enable RCU Register) can't read. It can write to configure but the read back value is no meaning.

6. Register FEF0h (SDRAM Arbiter Control Register), FEF2h (SDRAM Mode Set Register), FEF4h (SDRAM Control Register) and FEF6h (SDRAM Timing Parameter Register) high byte can't read back when FFF2h bit0 is "1" (Auxiliary Configuration Register IOSIZ is 8 bits mode).

7. MAC descriptor sixth word (next descriptor high word) can't cross the page

boundary. If let the sixth word at the next page, MAC will crash. A page is 256 word. So please don't let descriptor start address at 001F6h, 003F6h, 005F6h…… FFDF6h, FFFF6h.

8. Transmit data byte count can't be one or two bytes. Else MAC TX hangs. DTLEN field of TX descriptor can't be 1 or 2.

9. When access MAC register offset 24h (MDIO read data register), it will trigger FF24h(Poll Register) too. Sometime this action will cause interrupt can't happen. Suggestion one is don't access register FD24h/FE24h in the interrupt handler.

10. System will hang when the next burst read cycle is the page hit, the start address is the last word of this page (XX1FFh, XX3FFh…) and the burst length is over 1.Suggestion is let TX and RX buffer and descriptor must be aligned the double word boundary.

11. OBL (over buffer length) status of RX descriptor sometimes doesn't correctly response. Suggestion is ignored OBL status.

9. Performance Tips

Here are some tips for enhance R1620 performance.

1. SDRAM timing control register (FEF6h)

This register value is depending on your SDRAM module. So please be careful to try it. If found, SDRAM data is incorrect, please get back the previous setting.

Default: F933h

Modification1: F722h

Description: It can save 4 clocks. Bit15-12:Self-refresh exit time, Bit11-8:Min Row Cycle time, Bit7-4:Min Pre-charge time, Bit3-0:Row to Column delay time.

Modification2: F522h

Description: It can save 6 clocks.

2. SDRAM control register (FEF0h)

Default: 0020h

Modification1: 00A1h

Description: bit7 R/W re-order, enhance CPU read performance
bit5 post-write, enhance CPU write performance
bit4-0: re-fresh priority counter, post the re-fresh operation when bus is busy.

Modification2: 00E1h

Description: bit7 R/W re-order, enhance CPU read performance
bit5 post-write, enhance CPU write performance
bit4-0: re-fresh priority counter, post the re-fresh operation when bus is busy.
bit6: instruction fetch burst, reduce SDRAM loading and enhance CPU code fetch.
But this function has bug at some special condition. If your program can't work properly, please turn off it.

3. Transmit descriptor count

Every TX descriptor is only 12byte(depend on your program declaration). Please let it more, for example 16 TX descriptors.

When program want to send packet, it can get a free TX descriptor every time. So program don't need to put the TX packet into QUEUE and send it in interrupt handler (Program need to get TX packets from QUEUE).

4. MAC Bus Control Register (FD08h/FE08h)

Default: 1F1Ah

Modification1: 102Bh

Description: FIFO transfer length from 16 bytes to 32 bytes. RX FIFO data threshold from 16 bytes to 32 bytes. This modification let SDRAM bus usage more efficiency.

Note: This setting is only for R1620-B. R1620-A please uses the default value.

Index

Errata:

V0.03	11/04	Append an example: how to clear TPF. P-14.
	11/06	Change data sheet to V1.07A. Flow-control function description rewrite.
V0.04	11/19	Difference between R1620 and R88xx.
V0.05	12/07	Append shadow function description
	12/20/01	Append a content index
V0.06	03/05/02	For R1620-A
V0.07	04/09/02	For R1620-B
V0.08	04/15/02	Addition performance tips.
V0.08	04/27/02	Addition difference between R88xx.
V0.09	05/08/02	Addition difference between R88xx and a difference to R1620-A
V0.091	05/21/02	Addition programming notes.
V0.092	05/27/02	Append programming notes items.
V0.093	06/25/02	Append programming notes items.
V0.094	07/02/02	Append programming notes items.
V0.096	07/25/02	Modify 8-3-2 AUR bit7,2 description Append programming notes item.
V0.097	08/01/02	Append programming notes item.
V0.098	08/06/02	Append programming notes item.
V0.100	10/14/02	Append programming notes item.