# Micrel KSZ8851 Step-by-Step Programmer's Guide

**Version 1.5**

**8/10/2010**

# Revision History

| Revision | Date | Summary of Changes |
|---|---|---|
| 1.5 | 8/10/2010 | Change section 2, step 7, 11. Disable ICMP checksum because it is only for non-fragment frame).<br>Added section 2, step 14.1 and 14.2. Configure Low/High Watermark to 6KB/4KB available buffer space.<br>Added section 6.2, Special notices for the KSZ8851MLL Register RXQCR "Start DMA Access" Bit. |
| 1.4 | 1/11/2010 | Added section 5 – special notices for 8-bit data bus mode operation. |
| 1.3 | 5/6/2009 | Correct step 6 error in section 4.1.<br>Added step 13.1 in section 2, force link in half duplex if auto-nego is failed. |
| 1.2 | 4/3/2009 | Delete section 2, step 6. KSZ8851 can not do "TxQ Auto-Enqueue".<br>Add step 9.1 in section 3 to do "TxQ Manual-Enqueue" after the frame has written to TxQ.<br>Correct section 3, step 6, section 4.1, step 14, and section 4.2, step 23, write/read frame unit must be in double word alignment. |
| 1.1 | 03/12/2008 | Correct section 4.1, step14, and section 4.2, step 16, RELEASE error frame is at RXQCR not RXFDPR.<br>Correct section 4.1, step8, and section 4.2, step 23, read frame unit must be in double word alignment instead of word alignment.<br>Add KS8851MLL register access section. |
| 1.0 | 03/06/2008 | First release. |
| 0.1 | 11/07/2007 | First draft. |

# Table of Contents

# 1 Overview

This document covers KSZ8851MQL and KSZ8851MLL only. There is a separate programming guide for KSZ8851SNL. Throughout this document, KSZ8851 refers to either KSZ8851MQL or KSZ8851MLL.

This document provides step-by-step procedures detailing the registers and values need to be initialized, steps to transmit data to the device, to receive data from the device for the KSZ8851 series single-port Ethernet controller.

Please refer to KSZ8851 datasheet for detail register information.

In order to set a bit in a register, such as step 13 in Initialization, read the register first and modify the target bit only and write it back.

# 2 KSZ8851 Initialization Steps

| Steps Sequence | Read\write | Register Name[bit] | Value | Description |
|---|---|---|---|---|
| 1 | Read | CIDER [15-0]<br>Offset 0xC0 | 0x8870 | Read the device chip ID, make sure it is correct ID (0x8870 for KSZ8851); otherwise there are some errors on the host bus interface. |
| 2 | Write | MARL[15-0]<br>Offset 0x10 | 0x89AB | Write QMU MAC address (low). MAC address is generally expressed in the form of 01:23:45:67:89:AB. (we use this MAC as an example). |
| 3 | Write | MARM[15-0]<br>Offset 0x12 | 0x4567 | Write QMU MAC address (Medium). MAC address is generally expressed in the form of 01:23:45:67:89:AB. (we use this MAC as an example). |
| 4 | Write | MARH[15-0]<br>Offset 0x14 | 0x0123 | Write QMU MAC address (High). MAC address is generally expressed in the form of 01:23:45:67:89:AB. (we use this MAC as an example). |
| 5 | Write | TXFDPR [15-0]<br>Offset 0x84 | 0x4000 | Enable QMU Transmit Frame Data Pointer Auto Increment. |
| 7 | Write | TXCR [15-0]<br>Offset 0x70 | 0x00EE | Enable QMU Transmit flow control / Transmit padding / Transmit CRC, and IP/TCP/UDP checksum generation. |
| 8 | Write | RXFDPR[15-0]<br>Offset 0x86 | 0x4000 | Enable QMU Receive Frame Data Pointer Auto Increment. |
| 9 | Write | RXFCTR[15-0]<br>Offset 0x9C | 0x0001 | Configure Receive Frame Threshold for one frame. |
| 10 | Write | RXCR1 [15-0]<br>Offset 0x74 | 0x7CE0 | Enable QMU Receive flow control / Receive all broadcast frames /Receive unicast frames, and IP/TCP/UDP checksum verification etc. |
| 11 | Write | RXCR2 [15-0]<br>Offset 0x76 | 0x001C | Enable QMU Receive UDP Lite frame checksum verification, UDP Lite frame checksum generation, IPv6 UDP fragment frame pass, and IPv4/IPv6 UDP UDP checksum field is zero pass. |
| 12 | Write | RXQCR[15-0]<br>Offset 0x82 | 0x0230 | Enable QMU Receive IP Header Two-Byte Offset /Receive Frame Count Threshold/RXQ Auto-Dequeue frame. |
| 13.1 | Write | P1CR[5]<br>Offset 0xF6,<br>bit 5 | 0 | Force link in half duplex if auto-negotiation is failed (e.g. KSZ8851 is connected to the Hub). |
| 13 | Write | P1CR[13]<br>Offset 0xF6,<br>bit 13 | 1 | Restart Port 1 auto-negotiation. |
| 14.1 | Write | FCLWR[15-0]<br>Offset 0xB0, | 0x0600 | Configure Low Watermark to 6KByte available buffer space out of 12KByte. |

| | | | | |
|---|---|---|---|---|
| 14.2 | Write | FCHWR[15-0]<br>0ffset 0xB2, | 0x0400 | Configure High Watermark to 4KByte available buffer space out of 12KByte. |
| 14 | Write | ISR [15-0]<br>0ffset 0x92, | 0xFFFF | Clear the interrupts status. |
| 15 | Write | IER [15-0]<br>0ffset 0x90, | 0xE000 | Enable Link Change\Transmit\Receive interrupt if your host processor can handle the interrupt, otherwise do not need to do this step. |
| 16 | Write | TXCR [0]<br>0ffset 0x70,<br>bit 0 | 1 | Enable QMU Transmit. |
| 17 | Write | RXCR1 [0]<br>0ffset 0x74,<br>bit 0 | 1 | Enable QMU Receive. |

## 2.1 KSZ8851 Additional Receive Initialization Steps

To minimize host CPU interrupt overhead, the KS8851 also supports generate only one receive interrupt after device RXQ received multiple frames. In order to configure this interrupt scheme, the following addition receives initialization steps need to be set.

| Steps Sequence | Read\write | Register Name[bit] | Value | Description |
|---|---|---|---|---|
| 9 | Write | RXFCTR[15-0]<br>0ffset 0x9C | 0x0004 | Configure Receive Frame Threshold for multiplex frames, e.g. four frames. |
| 9.1 | Write | RXDTTR[15-0]<br>0ffset 0x8C | 0x03E8 | Configure Receive Duration Threshold, e.g. 1ms. Device will still generate receive interrupt if RXQ only received one frame, but device timer already exceeds the threshold set in this register. |
| 12 | Write | RXQCR[15-0]<br>0ffset 0x82 | 0x02B0 | Enable QMU Receive IP Header Two-Byte Offset /Receive Frame Count Threshold/ Receive Duration Timer Threshold /RXQ Auto-Dequeue frame. |

# 3 KSZ8851 Transmit Steps

The host transmit driver must write each frame data to align with double word boundary at end. For example, the driver has to write up to 68 bytes if transmit frame size is 65 bytes.

| Steps Sequence | Read\write | Register Name[bit] | Value | Description |
|---|---|---|---|---|
| 0 | Transmit data frame from the upper layer to KSZ8851 device by a complete packet frame data. For every complete packet frame data transmit to KSZ8851, process the following the steps.<br><br>There are two variables are needed from the upper layer to transmit a data packet frame.<br>(1). Packet data pointer (**pTxData**). It points to the host CPU system memory space contains the complete Ethernet packet data.<br>(2). Packet length (**txPacketLength**). The Ethernet packet data length not includes CRC. | | | |
| 1 | Read | TXMIR [12-0]<br>0ffset 0x78 | >=<br>(**txPacketLength**+4) | Read value from TXMIR to check if QMU TXQ has enough amount of memory for the Ethernet packet data plus 4-byte frame header. Compare the read value with (**txPacketLength**+4), if less than (**txPacketLength**+4), **Exit**. |
| 2 | Write | IER [15-0]<br>0ffset 0x90, | 0000 | Disable all the device interrupts generation. |
| 3 | Write | RXQCR[3]<br>0ffset 0x82<br>bit 3 | 1 | Start[1] QMU DMA transfer operation to write frame data from host CPU to the **TxQ**. |
| 4 | Write | **REG_QDR_DUMMY**[2] | 0x8000 | Write TXIC to the "control word" of the frame header through **'REG_QDR_DUMMY'** dummy address. |
| 5 | Write | **REG_QDR_DUMMY** | **txPacketLength** | Write **txPacketLength** to the "byte count" of the frame header through **'REG_QDR_DUMMY'** dummy address. |
| 6 | `UINT16 *pTxData;`<br>`int lengthInWord=((txPacketLength+3) & ~0x03)>>1;` | | | Write frame data pointer by **pTxData** to the QMU TXQ through **'REG_QDR_DUMMY'** dummy address in WORD until finished the full packet length (**txPacketLength)** in DWORD alignment, but in WORD unit **'lengthInWord'**. |
| 7 | Write | **REG_QDR_DUMMY** | `*pTxData++` | Write 2-**byte**[3] of frame data pointer by |

[1] Once QMU DMA transfer operation is started, host must not access to other device registers.
[2] **REG_QDR_DUMMY** is the dummy address to be accessed to QMU TxQ or RxQ when we start QMU transfer operation which regardless QMU address and byte enable signals with AEN, RDN, or WRN signals.

| | | | | |
|---|---|---|---|---|
| | | | | pTxData to the QMU TXQ through **'REG_QDR_DUMMY'** dummy address. Increase **`pTxData`** pointer by 2. |
| 8 | **lengthInWord** --;<br>if (**lengthInWord** > 0 )<br>goto Step 7;<br>else<br>goto Step 9; | | | Subtract **lengthInWord** by 1. |
| 9 | Write | RXQCR[3]<br>0ffset 0x82<br>bit 3 | 0 | Stop QMU DMA transfer operation. |
| 9.1 | Write | TXQCR[0]<br>0ffset 0x80<br>bit 0 | 1 | TxQ Manual-Enqueue. |
| 10 | Write | IER [15-0]<br>0ffset 0x90, | 0xEB00 | Enable the device interrupts again.<br>**Exit.** |

[3] If it is KSZ8851-32, 32bit bus interface, you should write **4-byte** of frame date pointer by pTxData to the QMU TXQ through **'REG_QDR_DUMMY'** dummy address. Increase **pTxData** pointer by 4 and Subtract **txPacketLength** by 4.
If it is KSZ8851-8, 8bit bus interface, you should write **1-byte** of frame date pointer by pTxData to the QMU TXQ through **'REG_QDR_DUMMY'** dummy address. Increase **pTxData** pointer by 1 and Subtract **txPacketLength** by 1.

# 4 KSZ8851 Receive Steps

There are two methods of receiving frames from QMU RXQ. First one just reads single frame from RXQ per DMA transfer operation. Second one with have better performance by reading multiplex frames per DMA transfer operation. The following sections describe receiving steps on these two different methods.

The host receive driver must read each frame data to align with double word boundary at end. For example, the driver has to read up to 68 bytes if received frame size is 65 bytes.

## 4.1 KSZ8851 Receive Single Frame per DMA

Host driver reads single frame from RXQ per DMA transfer operation.

| Steps Sequence | Read\write | Register Name[bit] | Value | Description |
|---|---|---|---|---|
| 0 | There are two methods to receive a complete Ethernet packet from KSZ8851 device to upper layer either as a result of polling or servicing an interrupt.<br><br>(1). By polling, set a timer routine to periodically execute step 1.<br>(2). By servicing an interrupt, when interrupt occurs, execute step 1.<br><br>Allocate a system memory space (address by **`pRxData`**) which is big enough to hold an Ethernet packet frame for each received frame from QMU RXQ. | | | |
| 1 | Read | ISR [13]<br>0ffset 0x92,<br>bit 13 | 1 | Read value from ISR to check if RXIS 'Receive Interrupt' is set. If not set, **Exit.** |
| 2 | Write | IER [15-0]<br>0ffset 0x90, | 0000 | Disable all the device interrupts generation. |
| 3 | Write | ISR [13]<br>0ffset 0x92,<br>bit 13 | 1 | Acknowledge (clear) RXIS Receive Interrupt bit. |
| 4 | Read | RXFCTR[15-8]<br>0ffset 0x9C | **rxFrameCount** | Read current total amount of received frame count from RXFCTR, and save in '**rxFrameCount**'. |
| **5** | if (**rxFrameCount** > 0 )<br>goto Step 6;<br>else<br>goto step 20; | | | Loop reading all frames from RXQ.<br>If **rxFrameCount** <= 0, **goto step 20** |
| 6 | Read | RXFHSR [15-0]<br>0ffset 0x7C | **rxStatus** | Read received frame status from RXFHSR to check if this is a good frame.<br><br>if **rxStatus**'s bit_15 is 0, or |

| | | | | |
|---|---|---|---|---|
| | | | | if **rxStatus**'s bit_0, bit_1, bit_2, bit_4, bit_10, bit_11, bit_12, bit_13 are 1, received a error frame, **goto step 8,**<br>else received a good frame, **goto step 7.** |
| 7 | Read | RXFHBCR [11-0]<br>0ffset 0x7E | **rxPacketLength** | Read received frame byte size from RXFHBCR to get this received frame length (4 byte CRC is included, and ), And store into **rxPacketLength** variable.<br><br>if **rxPacketLength** <= 0, **goto step 8;**<br>else **goto step 9**; |
| 8 | Write | RXQCR [0]<br>0ffset 0x82<br>bit 0 | 1 | Issue the RELEASE error frame command for the QMU to release the current error frame from RXQ.<br>**goto step 19**; |
| 9 | Write | RXFDPR[15-0]<br>0ffset 0x86 | 0x4000 | Reset QMU RXQ frame pointer to zero. |
| 10 | Write | RXQCR[3]<br>0ffset 0x82<br>bit 3 | 1 | Start QMU DMA transfer operation to read frame data from the RXQ to host CPU. |
| 11 | Read | **REG_QDR_DUMMY** | **pDummy** | Dummy read 2-byte if it is 16-bit data bus interface, read 4-byte if it is 32-bit data bus interface, or read 1-byte if it is 8-bit data bus interface from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 12 | Read | **REG_QDR_DUMMY** | **pDummy** | Read out 2-byte 'Status Word' of frame header from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 13 | Read | **REG_QDR_DUMMY** | **pDummy** | Read out 2-byte 'Byte Count' of frame header from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 14 | `UINT16 *pRxData;`<br>int **lengthInWord**=((**rxPacketLength** +3) & ~0x03)>> 1; | | | Read frame data to system memory pointer by **`pRxData`** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address in DWORD alignment until finished the full packet length (**rxPacketLength)** in WORD unit **'lengthInWord'**. |
| 15 | Read | **REG_QDR_DUMMY** | **`*pRxData ++`** | Read 2-byte[4] of frame data to system |

[4] If it is KSZ8851-32, 32bit bus interface, you should read **4-byte** of frame data to system memory pointer by **pRxData** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. Increase **pRxData** pointer by 4 and Subtract **rxPacketLength** by 4.
If it is KSZ8851-8, 8bit bus interface, you should read **1-byte** of frame data to system memory pointer by **pRxData** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. Increase **pRxData** pointer by 1 and Subtract **rxPacketLength** by 1.

| | | | | |
|---|---|---|---|---|
| | | | | memory pointer by **pRxData** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address.<br>Increase **`pRxData`** pointer by 2. |
| 16 | **lengthInWord** --;<br>if (**lengthInWord** > 0 )<br>goto Step 15;<br>else<br>goto Step 17; | | | Subtract **lengthInWord** by 1. |
| 17 | Write | RXQCR[3]<br>0ffset 0x82<br>bit 3 | 0 | Stop QMU DMA transfer operation. |
| 18 | Pass this received frame to the upper layer protocol stack.<br><br>Because "Receive IP Header Two-Byte Offset" feature is enabled, there are two extra bytes before the valid frame data, and two extra bytes count is included in the frame header 'Byte Count' (RXFHBCR). Also, another 4-byte CRC length is included in the frame header 'Byte Count' (RXFHBCR).<br><br>In order to pass the correct received frame (not include CRC) pointer by **pRxData** and received frame length '**rxPacketLength**' to the upper layer protocol stack, the driver need to do:<br>(1). Increase data pointer **`pRxData`** by 2-byte to the beginning of Ethernet packet data ,<br>**`pRxData += 2;`**<br>(2). Minus 2 extra bytes from '**rxPacketLength**' to the upper layer.<br>**rxPacketLength** -= 2;<br>(3). Minus 4-byte CRC length from '**rxPacketLength**' to the upper layer.<br>**rxPacketLength** -= 4;<br>(4). Pass received frame to upper layer protocol stack.<br>toUpperLayer (**`pRxData`**, **rxPacketLength** ); | | | |
| 19 | **rxFrameCount = rxFrameCount – 1;**<br>goto step 5 . | | | Finished reading one frame, subtract **rxFrameCount** by 1.<br>Loop again. |
| 20 | Write | IER [15-0]<br>0ffset 0x90 | 0xEB00 | Enable the device interrupts again.<br>Exit. |

.

## 4.2 KSZ8851 Receive Multiple Frames per DMA

Host driver reads multiple frames from RXQ per DMA transfer operation.

| Steps Sequence | Read\write | Register Name[bit] | Value | Description |
|---|---|---|---|---|
| 0 | There are two methods to receive a complete Ethernet packet from KSZ8851 device to upper layer either as a result of polling or servicing an interrupt.<br><br>(1). By polling, set a timer routine to periodically execute step 1.<br>(2). By servicing an interrupt, when interrupt occurs, execute step 1.<br><br>Since we need to record received multiplex frames header information (status and frame length) before read the multiplex frames from QMU RXQ in one DMA transfer operation, we need a array or link list structure that has two variable to store each received frame status '**rxStatus**', and frame length '**rxLength**'.<br>Eg, the sample array structure to store received multiplex frame header information:<br>`typedef struct {`<br>`USHORT  rxStatus;`<br>`USHORT  rxLength;`<br>`} FR_HEADER_INFO;`<br>FR_HEADER_INFO rxFrameHeader[ MAX_FRAMES_IN_RXQ ];<br><br>Allocate a system memory space (address by **`pRxData`**) which is big enough to hold an Ethernet packet frame for each received frame from QMU RXQ. | | | |
| 1 | Read | ISR [13]<br>0ffset 0x92,<br>bit 13 | 1 | Read value from ISR to check if RXIS 'Receive Interrupt' is set. If not set, **Exit.** |
| 2 | Write | IER [15-0]<br>0ffset 0x90, | 0000 | Disable all the device interrupts generation. |
| 3 | Write | ISR [13]<br>0ffset 0x92,<br>bit 13 | 1 | Acknowledge (clear) RXIS Receive Interrupt bit. |
| 4 | Read | RXFCTR[15-8]<br>0ffset 0x9C | **rxFrameCount** | Read current total amount of received frame count from RXFCTR, and save in '**rxFrameCount**'. |
| 5 | int i = 0;<br>if (**rxFrameCount** > 0 )<br>goto Step 6;<br>else<br>goto step 9; | | | Loop reading all frames header information from RXGHSR and RXFHBCR.<br>If **rxFrameCount** <= 0, **goto step 9.** |
| 6 | Read | RXFHSR [15-0]<br>0ffset 0x7C | **rxFrameHeader[i].rxStatus** | Read received frame status from RXFHSR to '**rxStatus**' array variable. |
| 7 | Read | RXFHBCR [11-0]<br>0ffset 0x7E | **rxFrameHeader[i].rxLength** | Read received frame byte size from RXFHBCR to **'rxLength'** array variable. |

| | | | | |
|---|---|---|---|---|
| 8 | **rxFrameCount = rxFrameCount – 1;**<br>**i +=1;**<br>goto step 5 . | | | Finished store one frame header information, subtract **rxFrameCount** by 1,<br>Increase array index by 1.<br>Loop again. |
| 9 | **rxFrameCount = i;**<br>i=0; | | | Restore total amount of received frame count '**rxFrameCount**' again. |
| 10 | Write | RXFDPR[15-0]<br>Offset 0x86 | 0x4000 | Reset QMU RXQ frame pointer to zero. |
| 11 | Write | RXQCR[3]<br>Offset 0x82<br>bit 3 | 1 | Start QMU DMA transfer operation to read frame data from the RXQ to host CPU. |
| 12 | Read | **REG_QDR_DUMMY** | **pDummy** | Dummy read 2-byte if it is 16-bit data bus interface, read 4-byte if it is 32-bit data bus interface, or read 1-byte if it is 8-bit data bus interface from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 13 | if (**rxFrameCount** > 0 )<br>goto Step 14;<br>else<br>goto step 28; | | | Loop reading all frames from RXQ.<br>If **rxFrameCount** <= 0, **goto step 28.** |
| 14 | #define **RX_ERRORS** 0x3C17<br>if ( (**rxFrameHeader[i]. rxStatus** & RX_ERRORS ) \|\|<br>(**rxFrameHeader[i]. rxLength** <= 0 ))<br>error frame, goto step 15;<br>else<br>good frame, goto step 21; | | | Check received frame status **'rxFrameHeader[i]. rxStatus'** to see if this is a good frame, and received frame length '**rxFrameHeader[i]. rxLength'.** |
| 15 | Write | RXQCR[3]<br>Offset 0x82<br>bit 3 | 0 | This is an error frame. Stop QMU DMA transfer operation. |
| 16 | Write | RXQCR[0]<br>Offset 0x82<br>bit 0 | 1 | Issue the RELEASE error frame command for the QMU to release the current error frame from RXQ. |
| 17 | Write | RXFDPR[15-0]<br>Offset 0x86 | 0x4000 | Reset QMU RXQ frame pointer to zero. |
| 18 | Write | RXQCR[3]<br>Offset 0x82<br>bit 3 | 1 | Then, Start the DMA transfer operation again for the next frame. |
| 19 | Read | **REG_QDR_DUMMY** | **pDummy** | Dummy read 2-byte if it is 16-bit data bus interface, read 4-byte if it is 32-bit data bus interface, or read 1-byte if it is 8-bit data bus interface from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 20 | goto step 27; | | | Go for processing the next frame. |
| 21 | Read | **REG_QDR_DUMMY** | **pDummy** | Read out 2-byte 'Status Word' of frame header from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |

| | | | | |
|---|---|---|---|---|
| 22 | Read | **REG_QDR_DUMMY** | **pDummy** | Read out 2-byte 'Byte Count' of frame header from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. |
| 23 | `UINT16 *pRxData;`<br>int **lengthInWord;**<br><br>**lengthInWord** =(( **rxFrameHeader[i]. rxLength** +3) & ~0x03 ) >> 1; | | | Read frame data to system memory pointer by **pRxData** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address in DWORD alignment until finished the full packet length **'rxFrameHeader[i]. rxLength'** in WORD unit '**lengthInWord**. |
| 24 | Read | **REG_QDR_DUMMY** | `*pRxData ++` | Read 2-byte[5] of frame data to system memory pointer by pRxData from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. Increase `pRxData` pointer by 2. |
| 25 | **lengthInWord** --;<br>if (**lengthInWord** > 0 )<br>goto Step 24;<br>else<br>goto Step 26; | | | Subtract **lengthInWord** by 1. |
| 26 | Pass this received frame to the upper layer protocol stack.<br><br>Because "Receive IP Header Two-Byte Offset" feature is enabled, there are two extra bytes before the valid frame data, and two extra bytes count is included in the frame header 'Byte Count' (RXFHBCR). Also, another 4-byte CRC length is included in the frame header 'Byte Count' (RXFHBCR).<br><br>In order to pass the correct received frame (not include CRC) pointer by **pRxData** and received frame length '**rxPacketLength**' to the upper layer protocol stack, the driver need to do:<br>(1). Increase data pointer `pRxData` by 2-byte to the beginning of Ethernet packet data ,<br>`pRxData += 2;`<br>(2). Minus 2 extra bytes from '**rxPacketLength**' to the upper layer.<br>**rxLength** -= 2;<br>(3). Minus 4-byte CRC length from '**rxPacketLength**' to the upper layer.<br>**rxLength** -= 4;<br>(4). Pass received frame to upper layer protocol stack.<br>toUpperLayer (`pRxData`, **rxFrameHeader[i]. rxLength**); | | | |
| 27 | **rxFrameCount = rxFrameCount – 1;**<br>**i +=1;**<br>goto step 13 . | | | Finished reading one frame,<br>subtract **rxFrameCount** by 1.<br>Increase array index by 1.<br>Loop again. |
| 28 | Write | RXQCR[3]<br>0ffset 0x82<br>bit 3 | 0 | Stop QMU DMA transfer operation. |
| 29 | Write | IER [15-0]<br>0ffset 0x90 | 0xEB00 | Enable the device interrupts again.<br>Exit. |

[5] If it is KSZ8851-32, 32bit bus interface, you could reade **4-byte** of frame data to system memory pointer by **pRxData** from the QMU RXQ through **'REG_QDR_DUMMY'** dummy address. Increase **pRxData** pointer by 4 and Subtract **rxLength** by 4.

# 5 KSZ8851MLL Host Bus Interface (BIU)

The KSZ8851MLL BIU host interface is an indirect access data bus interface. The Data Bus SD[15:0] specifies the address or data depending on the CMD control signal.

The Command (CMD) determines whether SD[15:0] is the address or data bus by following table.

<table>
<tr><th>CMD</th><th>SD</th><th colspan="2">Function (16-bit Bus)<br>CMD pin can be connecting to host address line A1.</th></tr>
<tr><td rowspan="5">CMD=1</td><td rowspan="5">SD[15:0]</td><td colspan="2">When command input is high, the access of shared data bus is for Address and Byte Enable.</td></tr>
<tr><td>SD[1:0]</td><td>Don't care</td></tr>
<tr><td>SD[7:2]</td><td>A[7:2]</td></tr>
<tr><td>SD[8:11]</td><td>Don't care</td></tr>
<tr><td>SD[15:12]</td><td>BE3/BE2/BE1/BE0<br><br>BE3:0x8000<br>BE2:0x4000<br>BE1:0x2000<br>BE0:0x1000</td></tr>
<tr><td rowspan="2">CMD=0</td><td rowspan="2">SD[15:0]</td><td colspan="2">When command input is low, the access of shared data bus is for Data.</td></tr>
<tr><td>SD[15:0]</td><td>D[15:0]</td></tr>
</table>

<table>
<tr><th>CMD</th><th>SD</th><th colspan="2">Function (8-bit Bus)<br>CMD pin can be connect to host address line A0,<br>SD[15:8] tied to low.</th></tr>
<tr><td rowspan="2">CMD=1</td><td rowspan="2">SD[7:0]</td><td colspan="2">When command input is high, the access of shared data bus is for Address.</td></tr>
<tr><td>SD[7:0]</td><td>A[7:2]</td></tr>
<tr><td rowspan="2">CMD=0</td><td rowspan="2">SD[7:0]</td><td colspan="2">When command input is low, the access of shared data bus is for Data.</td></tr>
<tr><td>SD[7:0]</td><td>D[7:0]</td></tr>
</table>

**Table 5. KSZ8851MLL Shared Data Bus Operation for Register Access**

The following sections describe how to access KSZ8851MLL registers and QMU in 16-bit bus interface.

## 5.1 Register Access

To access KSZ8851MLL registers, it always needs two steps to set value to SD bus, first step is write the address/BEn data to SD bus with CMD high, second step is read/write data from/to SD bus with CMD is low.

If KSZ8851MLL is configured as little-endian mode, the second step's data format is in which the least signification byte (LSB) is at the 0 address end as following:

| | MSB | LSB |
|---|---|---|
| Word | Data 15 - 8 | Data 7 – 0 |

And the first step's BEn to access internal 32-bit alignment registers as following:

| Operation | | Data Bus | | | | BEn | | | |
|---|---|---|---|---|---|---|---|---|---|
| Access Size | Address No. | D31-D24 | D23-D16 | D15-D8 | D7-D0 | BE3 | BE2 | BE1 | BE0 |
| Byte | 4n | — | — | — | Data 7-0 | | | | Asserted |
| | 4n+1 | — | — | Data 7-0 | — | | | Asserted | |
| | 4n+2 | — | Data 7-0 | — | — | | Asserted | | |
| | 4n+3 | Data 7-0 | — | — | — | Asserted | | | |
| Word | 4n | — | — | Data 15-8 | Data 7-0 | | | Asserted | Asserted |
| | 4n+2 | Data 15-8 | Data 7-0 | — | — | Asserted | Asserted | | |

### 5.1.1 Read From Registers

While CMD pin is connected to host address line **A1**, along with Chip Select Enable (CS) and Read (RDN) signals, the driver read data from registers in following steps:

1. Write address command - register offset value along with BEn to SD[15:0].
2. Read register value from SD[15:0].

Example 1: read 2-byte from register 0xC0 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x30C0 | Write value 0x30C0 (register offset 0xC0 with BE1/BE0), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | 0x8870 | Read value (will be chip ID 0x8870), |
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

| | | | | |
|---|---|---|---|---|
| | | | | |

Example 2: read 2-byte from register 0x12 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0xC012 | Write value 0xC012 (register offset 0x12 with BE3/BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | valus | Read value, |
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 3: read 1-byte from register 0x10 at external IO base address 0x0300.
Assume register 0x10 to register 13 contains value 0x12345678.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x1010 | Write value 0x1010 (register offset 0x10 with BE0), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | 0x1078 | Read 2-byte value 0x1078, D15-8 is invalid, only D7-0 is valid, |
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 4: read 1-byte from register 0x11 at external IO base address 0x0300.
Assume register 0x10 to register 13 contains value 0x12345678.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x2011 | Write value 0x2011 (register offset 0x11 with BE1), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | 0x5611 | Read 2-byte value 0x5611, only D15-8 is valid, D7-0 is invalid, |
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 5: read 1-byte from register 0x12 at external IO base address 0x0300.
Assume register 0x10 to register 13 contains value 0x12345678.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x4012 | Write value 0x4012 (register offset 0x12 with BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | 0x4034 | Read 2-byte value 0x4034, D15-8 is invalid, only D7-0 is valid, |

| | | | | |
|---|---|---|---|---|
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 6: read 1-byte from register 0x13 at external IO base address 0x0300.
Assume register 0x10 to register 13 contains value 0x12345678.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x8013 | Write value 0x8013 (register offset 0x13 with BE3), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Read | | 0x1213 | Read 2-byte value 0x1213, only D15-8 is valid, D7-0 is invalid, |
| 4 | From | 0x0300 | | From address 0x0300 (the address line A1 is low⇒CMD is low). |

### 5.1.2 Write To Registers

Assuming CMD pin is connected to host address line A1, along with Chip Select Enable (CS) and Write (WRN) signals, the driver write data to registers in following steps:

1. Write address command - register offset value along with BEn to SD[15:0].
2. Write value to SD[15:0].

Example 1: write 2-byte value (0x1234) to register 0x10 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x3010 | Write value 0x3010 (register offset 0x10 with BE1/BE0), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x1234 | Write 2-byte value 0x1234, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 2: write 2-byte value (0x5678) to register 0x12 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0xC012 | Write value 0xC012 (register offset 0x12 with BE3/BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x5678 | Write 2-byte value 0x5678, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 3: write 1-byte value (0xAB) to register 0x10 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x1010 | Write value 0x1010 (register offset 0x10 with BE0), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x00AB | Write 2-byte value 0x00AB, D15-8 don't care, only D7-0 is valid, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 4: write 1-byte value (0xCD) to register 0x11 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x2011 | Write value 0x2011 (register offset 0x11 with BE1), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0xCD00 | Write 2-byte value 0xCD00, only D15-8 is valid, D7-0 don't care, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 5: write 1-byte value (0xEF) to register 0x12 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x4012 | Write value 0x4012 (register offset 0x12 with BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x00EF | Write 2-byte value 0x00EF, D15-8 don't care, only D7-0 is valid, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example 6: write 1-byte value (0x56) to register 0x13 at external IO base address 0x0300.

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | 0x8013 | Write value 0x8013 (register offset 0x13 with BE3), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x5600 | Write 2-byte value 0x5600, only D15-8 is valid, D7-0 don't care, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

## 5.2 QMU Access

To access KSZ8851MLL QMU RXQ/TXQ, it only needs one step to read/write data from/to SD bus with CMD is low.

### 5.2.1 Read From RXQ

The KSZ8851MLL allows a transfer operation from the host CPU to read frame data from QMU RXQ frame buffer with Chip Select Enable (CS), Read (RDN) while CMD pin (**A1**) is always low after RXQCR bit 3 ("Start DMA Access") is set, which start the QMU transfer operation.

Like section 4.1 steps 15 (external IO base address 0x0300),

| | | | | |
|---|---|---|---|---|
| 15 | Read | 0x0300 | `*pRxData ++` | Read 2-byte of frame data to system memory pointer by **pRxData** from the QMU RXQ through **'0x0300'** address (the address line A1 is low⇒CMD is low).<br>Increase **`pRxData`** pointer by 2. |

### 5.2.2 Write To TXQ

The KSZ8851MLL allows a transfer operation from the host CPU to write frame data to QMU TXQ frame buffer with Chip Select Enable (CS), Write (WRN) signals while CMD pin (**A1**) is always low after RXQCR bit 3 ("Start DMA Access") is set, which start the QMU transfer operation.

Like section 3, steps 7 (external IO base address 0x0300),

| | | | | |
|---|---|---|---|---|
| 7 | Write | 0x0300 | `*pTxData++` | Write 2-**byte** of frame data pointer by pTxData to the QMU TXQ through **'0x0300'** address (the address line A1 is low⇒CMD is low).<br>Increase **`pTxData`** pointer by 2. |

# 6 Special Notices

There are some special notices for the
- KSZ8851 8-Bit data bus driver
- Set/reset KSZ8851MLL register RXQCR bit 3 - "Start DMA Access".

## 6.1 KSZ8851 8-Bit Data Bus Mode Operation

If KSZ8851 is connected to the host processor using 8-bit data bus, all registers are accessed low byte first with only one exception – RXQCR register.

When the bit 3 of RXQCR register (SDA – Start DMA Access) is set, the QMU access starts immediately and any KSZ8851 access afterward is related to QMU. So the high byte of RXQCR need to be written first before SDA is set.

The SDA is set only at four steps and there are:

- Section 3 Transmit step 3
- Section 4.1 Receive Single Frame step 10
- Section 4.2 Receive Multiple Frames step 11 and step 18

<u>*QMU Access*</u>

The frame format for the transmit queue and receive queue are shown in the following tables in the 8-bit format. The TXQ will be written and RXQ will be read in the 8-bit operation.

- Transmit Queue (TXQ) Frame Format

| Packet Memory Address Offset | Bit 7 Bit 0 |
|---|---|
| 0 | Low byte of 'Control Word' - Transmit Frame ID |
| 1 | High byte of 'Control Word'. E.g. 0x80 to set the transmit interrupt. |
| 2 | Low byte of 'Byte Count'. E.g. (length & 0xff) |
| 3 | High byte of 'Byte Count'. E.g. (length >> 8) |
| 4 - up | Transmit Packet Data<br>(maximum size is 2000) |

- Receive Queue (RXQ) Frame Format

| Packet Memory Address Offset | Bit 7 Bit 0 |
|---|---|
| 0 | Low byte of 'Status Word' - Same as register RXFHSR bit 7 – 0. |
| 1 | High byte of 'Status Word' - Same as register RXFHSR bit 15 – 8. |
| 2 | Low byte of 'Byte Count'. E.g. (length = low byte) |
| 3 | High byte of 'Byte Count'. E.g. (length \|= (high byte << 8) |
| 4 - up | Receive Packet Data<br>(maximum size is 2000) |

## 6.2 KSZ8851MLL Set/Reset Register RXQCR "Start DMA Access" Bit

As description in section 5.1, it requested two steps to access KSZ8851MLL registers:

1. First phase, write register offset address/BEn with CMD high to SD bus.
2. Second phase, read/write value with CMD low from/to SD bus.

For the first phase, as show on Table 5, register offset address A1/A0 is Don't care option. **But, register offset address A1/A0 MUST included in the first phase for the register RXQCR to Start_DMA or Stop_DMA** during the frame data read\write to KSZ8851MLL RxQ\TxQ, like in the previous section:

- Section 3 Transmit step 3, 9.
- Section 4.1 Receive Single Frame step 10, 17
- Section 4.2 Receive Multiple Frames step 11, 18, and 28.

Example: write 0x38 (**Start_DMA**) to register 0x82 (**RXQCR)** at external IO base address 0x0300:

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | **0xC082** | Write value 0xC012 (register offset 0x82 with BE3/BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x0038 | Write 2-byte value 0x0038, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |

Example: write 0x30 (**Stop_DMA**) to register 0x82 (**RXQCR)** at external IO base address 0x0300:

| Steps Sequence | Operation | Address | Value | Description |
|---|---|---|---|---|
| 1 | Write | | **0xC082** | Write value 0xC012 (register offset 0x82 with BE3/BE2), |
| 2 | To | 0x0302 | | To address 0x0302 (the address line A1 is high⇒CMD is high). |
| 3 | Write | | 0x0030 | Write 2-byte value 0x0030, |
| 4 | To | 0x0300 | | To address 0x0300 (the address line A1 is low⇒CMD is low). |