Micrel KSZ8851SNL Step-by-Step Programmer's Guide

Version 1.6

10/09/2012

Revision History

Revision	Date	Summary of Changes
1.6	10/09/2012	Enhance section 2.2. Change section 3, step 1 for correct Rev A2 and Rev A3 chip ID. Delete section 4, step 4, 5, 1; section 5.1, step 11, 12, 20; section 5.2, step 12, 16, 21, 22. These belong to section 2 SPI interface. Change section 4, step 13; section 5.1, step 24; section 5.2, step 35; interrupt mask from '0xEB00' to '0xE000'. Change section 5.1, step 6, 7 descriptions. Add section 5.3 and 5.4.
1.5	8/10/2010	Change section 3, step 6, 10. Disable ICMP checksum because it is only for non-fragment frame). Added section 3, step 14.1 and 14.2. Configure Low/High Watermark to 6KB/4KB available buffer space.
1.4	2/12/2010	Delete section 5.1, step 16, and section 5.2, step 27.
1.3	5/26/2009	Added section 2.3.3 and 2.3.4, example for how the KSZ8851 SNL driver read/write frame data from/to RXQ/TXQ with SPI controller only can transfer 4-byte data per SPI transaction cycle.
1.2	5/6/2009	Correct step 6 error in section 5.1. Added step 13.1 in section 3, force link in half duplex if auto-nego is failed.
1.1	4/3/2009	Add step 12.1 in section 4 to do "TxQ Manual-Enqueue" after the frame has written to TxQ.
1.0	03/20/2009	First release.

Table of Contents

1 Overview

This document provides step-by-step programming procedure detailing the registers and values need to be initialized, how to transmit data to the device, and how to receive data from KSZ8851SNL single-port Ethernet controller connected to the host processor SPI master controller.

Please refer to KSZ8851SNL 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 KSZ8851SNL SPI Interface

The KSZ8851SNL supports SPI interface in the slave mode. In this mode, an external SPI master controller supplies the operating serial clock (SCLK), chip select (CSN) and serial input data (SI) which is sampled on the rising edge of SCLK to KSZ8851SNL device. Serial output data (SO) is driven by KSZ8851SNL on the rising edge of SCLK to external SPI master device. The falling edge of CSN starts the SPI operation and the rising edge of CSN ends the SPI operation. The SCLK stays low when SPI operation is idle. Figure 2 shows the SPI interface connection for KSZ8851SNL.

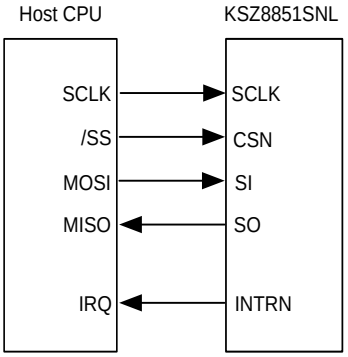

Figure 2. SPI Interface to KSZ8851SNL

2.1 SPI Master Controller Configuration

The SPI master initialization routine must configure SPI master controller with following modes required by KSZ8851SNL:

√ 8 bits per SPI data transfers if possible.

√ Data are transferred with the MSB first.

√ SPI serial clock (SCLK) - KSZ8851SNL can handle up to 50 MHz.

√ Chip select (CSN) must remain active low during each SPI operation cycle.

√ Uses SPI mode 0 (CPOL=0, CPHA=0) as Figure 2-1.

- Clock Polarity (CPOL=0) define SCLK active state is at logic level high.
- Clock Phase (CPHA=0) define data starts on the leading edge of SCLK (from low to high transition)

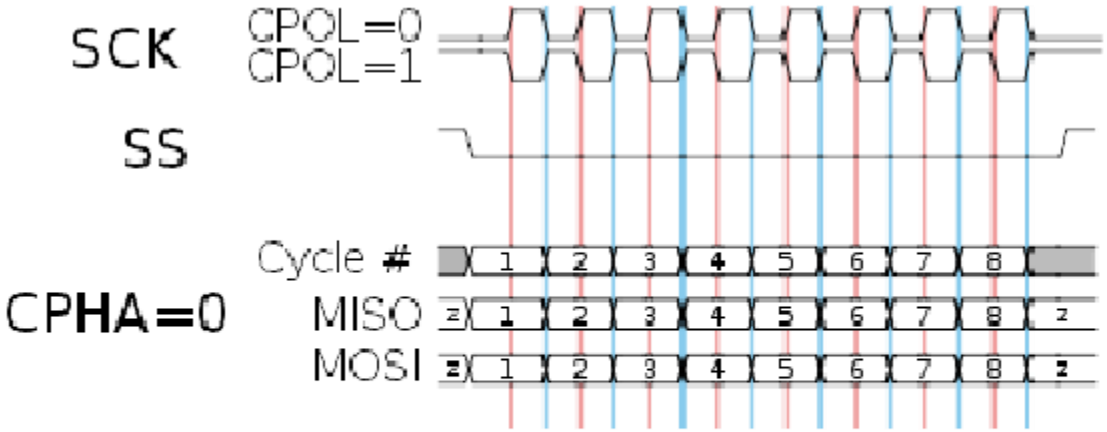

Figure 2-1. SPI Mode 0 Timing Diagram.

2.2 Register Access

To access KSZ8851SNL registers, it always requires two phases, command phase (**CMD**) and data phase (**DATA**) to complete the SPI cycle.

- Command phase includes two bytes consisting of 'opcode', 'byte enable', and 'register address'.
- Data phase can be 1, 2, or 4 bytes long specified in command phase 'byte enable' to access specified byte location in the internal 32-bit boundary of registers.
- The 'register address' field in the command phase consists of only A[7:2] to access register location in DWORD boundary.

SPI Operation	Command Phase (CMD) (to SI pin)				Data Phase (DATA) (from SO or to SI pins)
	Byte 0 [7:0]			Byte 1 [7:0]	
	Opcode	Byte enable	Register Address	Don't care bits	
Register Read	0 0	B3 B2 B1 B0	A7 A6 A5 A4 A3 A2	X X X X	1 to 4 Bytes (read data on SO pin)
Register Write	0 1	B3 B2 B1 B0	A7 A6 A5 A4 A3 A2	X X X X	1 to 4 Bytes (write data on SI pin)

Table 2-2. SPI Operation for Registers Access

- The 'byte enable' field in the command phase B[3:0] specifies the bytes to be accessed. When B0 is '1', SPI master controller will access LSB byte from KSZ8851SNL register. When B3 is '1', SPI master control will access MSB byte from KSZ8851SNL register. The program can specify to read a byte, a word, or a long word at a time.

  (1). to read a BYTE at a time:

A1	A0	B3	B2	B1	B0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

  (2). to read a WORD at a time:

A1	A0	B3	B2	B1	B0
0	0	0	0	1	1
1	0	1	1	0	0

(3). to read a DWORD at a time:

A1	A0	B3	B2	B1	B0
0	0	1	1	1	1

- While data is transferred in the MSB first mode in the SPI cycle, byte0 is the first byte to appear and the byte 3 is the last byte for the data phase.
- **Chip select (CSN) must remain active low during each SPI register read or write cycle.**
- Number of SCLK for register access as following:
  Register BYTE access: CMD(16bits) + DATA(8bits)
  Register WORD access: CMD(16bits) + DATA(16bits)
  Register DWORD access: CMD(16bits) + DATA(32bits)

2.2.1 Register Reading Examples

In this section, examples show how KS8851SNL driver (SPI master) reads KSZ8851SNL (SPI slave) registers using 8 bit SPI data transfer.

Example 1: read 2-byte from register 0xC0 – read chip ID.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x0F	CMD Byte 0 (B [1:0] is enabled).
3	Write	SI	0x00	CMD Byte 1.
4	Read	SO	0x72	DATA Byte 0.
5	Read	SO	0x88	DATA Byte 1.
6	Set	CSN	Deactivate	Stop SPI operation.

Example 2: read 4-byte from register 0x10 – read MAC address value.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x3C	CMD Byte 0 (B [3:0] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Read	SO	0x11	DATA Byte 0.
5	Read	SO	0x95	DATA Byte 1.
6	Read	SO	0x86	DATA Byte 2.
7	Read	SO	0xA1	DATA Byte 3.
8	Set	CSN	Deactivate	Stop SPI operation.

Example 3: read 1-byte from register 0x10.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x04	CMD Byte 0 (B [0] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Read	SO	0x11	DATA Byte 0.

5	Set	CSN	Deactivate	Stop SPI operation.

Example 4: read 1-byte from register 0x11.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x08	CMD Byte 0 (B [1] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Read	SO	0x95	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 5: read 1-byte from register 0x12.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x10	CMD Byte 0 (B [2] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Read	SO	0x86	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 6: read 1-byte from register 0x13.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x20	CMD Byte 0 (B [3] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Read	SO	0xA1	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

2.2.2 Register Writing Examples

In this section, examples show how KS8851SNL driver (SPI master) write value to the KSZ8851SNL (SPI slave) registers using 8 bit SPI data transfer.

Example 1: write 2-byte value (0x1234) to register 0x10.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x4C	CMD Byte 0 (B [1:0] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0x34	DATA Byte 0.
5	Write	SI	0x12	DATA Byte 1.
6	Set	CSN	Deactivate	Stop SPI operation.

Example 2: write 2-byte value (0x5678) to register 0x12.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x70	CMD Byte 0 (B [3:2] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0x78	DATA Byte 0.
5	Write	SI	0x56	DATA Byte 1.
6	Set	CSN	Deactivate	Stop SPI operation.

Example 3: write 1-byte value (0xAB) to register 0x10.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x44	CMD Byte 0 (B [0] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0xAB	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 4: write 1-byte value (0xCD) to register 0x11.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x48	CMD Byte 0 (B [1] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0xCD	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 5: write 1-byte value (0xEF) to register 0x12.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x50	CMD Byte 0 (B [2] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0xEF	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 6: write 1-byte value (0x56) to register 0x13.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x60	CMD Byte 0 (B [3] is enabled).
3	Write	SI	0x40	CMD Byte 1.
4	Write	SI	0x56	DATA Byte 0.
5	Set	CSN	Deactivate	Stop SPI operation.

Example 7: write 4-byte value (0x12345678) to register 0x38.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x7C	CMD Byte 0 (B [3:0] is enabled).
3	Write	SI	0xE0	CMD Byte 1.
4	Write	SI	0x78	DATA Byte 0.

5	Write	SI	0x56	DATA Byte 1.
6	Write	SI	0x34	DATA Byte 2.
7	Write	SI	0x12	DATA Byte 3.
8	Set	CSN	Deactivate	Stop SPI operation.

2.3 QMU Access Examples

To access KSZ8851SNL QMU, it always requires two phases in SPI cycle - command phase (**CMD**) and data phase (**DATA**).

- Command phase is one bytes long including only 'opcode'.
- Data phase is limited up to 6Kbytes for TXQ access, or 12Kbytes for RXQ access.
- "TXQ Write" CMD is required for each frame writing to TXQ if SPI master controller could finish writing the complete frame in one SPI cycle.
- "TXQ Write" CMD is need for each data burst writing to TXQ if SPI master controller could not finish writing the complete frame in one SPI cycle.
- "RXQ Read" CMD is needed for each frame reading from RXQ if SPI master controller could read a single frame data in one SPI CSN activate.
- "RXQ Read" CMD is need for each data reading burst from RXQ if SPI master controller could not finish reading the complete frame in one SPI cycle.
- Number of DATA bytes writing to TXQ must be in DWORD alignment.
- Number of DATA bytes reading from RXQ must be in DWORD alignment.
- The Start DMA Access, bit 3 in RXQCR register, must set to '1' before "RXQ Read" CMD or "TXQ Write" CMD (before CSN activation). And clear to '0' after "RXQ Read" CMD or "TXQ Write" CMD cycle finishes (after CSN deactivate).
- **Chip select (CSN) must remain active low during each SPI RXQ read or TXQ write cycle.**
- Number of SCLK for register access as following:
  Write frame to TXQ: CMD(8bits) + Frame Header(32bits) + Frame Data(8bits*N)
  Read frame from RXQ: CMD(8bits) + Dummy(32bits) + Frame Status(32bits) + Frame Data(8bits*N)

SPI Operation	Command Phase(CMD) (to SI pin)		Data Phase (DATA) (from SO or to SI pins)
	Byte 0 [7:0]		
	Opcode	Don't care bits	
RXQ Read (12 KByte)	1 0	X X X X X X	1 to 12 KBytes (Reading data on SO pin)
TXQ Write (6 KByte)	1 1	X X X X X X	1 to 6 KBytes (Writing data on SI pin)

Table 2-3. SPI Operation for TXQ/RXQ QMU Access

2.3.1 Writing To TXQ Example 1

In this section, an example shows how KS8851SNL driver (SPI master) writes a **61 byte** frame data in **one SPI transaction cycle** to KSZ8851SNL (SPI slave) TXQ by using 8 bit SPI data transfers.

The Start DMA Access, bit 3 of RXQCR register, is set to '1' before "TXQ Write" CMD (before CSN activate). And it is cleared to '0' after "TXQ Write" CMD (after CSN deactivate).

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0xC0	TXQ Write CMD.
3	Write	SI	0x00	Frame header 'Control Word'.
4	Write	SI	0x80	
5	Write	SI	0x3D	Frame header 'Byte Count' is 61 bytes.
6	Write	SI	0x00	
7	Write	SI	0xFF	Frame byte 1 (first byte of frame packet)
8	Write	SI	0xFF	Frame byte 2.
9	…			Frame byte 3 – byte 60.
68	Write	SI	0x31	Frame byte 61 (last byte of frame packet)
69	Write	SI	Dummy data	Dummy write byte 62-64 to make number of DATA bytes write to TXQ must be in DWORD alignment.
70	Write	SI	Dummy data	
71	Write	SI	Dummy data	
72	Set	CSN	Deactivate	Stop SPI operation.

2.3.2 Reading From RXQ Example 1

In this section, an example shows how KS8851SNL driver (SPI master) read a **65 byte** (including CRC) frame from KSZ8851SNL (SPI slave) RXQ by using 8 bit SPI data transfers assuming SPI master controller could read a single frame data in **one SPI transaction cycle.**

Assuming that 'RX IP Header Two-Byte Offset Enable' ,bit 9 of RXQCR register is set to '1', there will be two extra bytes count that is included in the frame header 'Byte Count'.

The Start DMA Access ,bit 3 of RXQCR register, is set to '1' before "RXQ Read" CMD (before CSN activate). And it is cleared to '0' after "RXQ Read" CMD (after CSN deactivate).

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x80	RXQ Read CMD.
3	Read	SO	Dummy	Read out dummy 4 bytes.
4	Read	SO	Dummy	
5	Read	SO	Dummy	
6	Read	SO	Dummy	
7	Read	SO	0xC8	Frame header 'Status Word'.
8	Read	SO	0x81	
9	Read	SO	0x43	Frame header 'Byte Count' is 67 bytes, which is 2 bytes of dummy data (due to 'IP Header Two-Byte Offset Enable') + 61 bytes of frame data + 4 bytes of CRC.
10	Read	SO	0x00	
11	Read	SO	Dummy	Read out dummy 2 bytes due to 'IP Header Two-Byte Offset Enable'.
12	Read	SO	Dummy	
13	Read	SO	0xFF	Frame byte 1 (first byte of frame packet)
14	Read	SO	0xFF	Frame byte 2.
15	…			Frame byte 3 – byte 60.
68	Read	SO	0x31	Frame byte 61 (last byte of frame packet)
69	Read	SO	0x59	Frame CRC
70	Read	SO	0xAE	
71	Read	SO	0x5C	
72	Read	SO	0x38	
73	Read	SO	Dummy data	Dummy read 66-68 to make number of DATA bytes read from RXQ must be in DWORD alignment.
74	Read	SO	Dummy data	
75	Read	SO	Dummy data	
76	Set	CSN	Deactivate	Stop SPI operation.

2.3.3 Writing To TXQ Example 2

In this section, an example shows how KS8851SNL driver (SPI master) writes a **61 byte** frame data to KSZ8851SNL (SPI slave) TXQ by using 8 bit SPI data transfers, assuming SPI master controller writes **4-byte data per SPI transaction cycle.**

The Start DMA Access, bit 3 of RXQCR register, is set to '1' before step 1. And it is cleared to '0' after step 22.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0xC0	TXQ Write CMD.
3	Write	SI	0x00	Frame header 'Control Word'.
4	Write	SI	0x80	
5	Write	SI	0x3D	Frame header 'Byte Count' is 61 bytes.
6	Write	SI	0x00	
7	Set	CSN	Deactivate	Stop SPI operation.
8	Set	CSN	Activate	Starting SPI operation.
9	Write	SI	0xC0	TXQ Write CMD.
10	Write	SI	0xFF	Frame byte 1 (first byte of frame packet)
11	Write	SI	0xFF	Frame byte 2.
12	Write	SI	0xFF	Frame byte 3.
13	Write	SI	0xFF	Frame byte 4.
14	Set	CSN	Deactivate	Stop SPI operation.
15	…			Repeat for the reset of Frame data.
16	Set	CSN	Activate	Starting SPI operation.
17	Write	SI	0xC0	TXQ Write CMD.
18	Write	SI	0x31	Frame byte 61 (last byte of frame packet)
19	Write	SI	Dummy data	Dummy write byte 62-64 to make number of DATA bytes write to TXQ must be in DWORD alignment.
20	Write	SI	Dummy data	

21	Write	SI	Dummy data	
22	Set	CSN	Deactivate	Stop SPI operation.

2.3.4 Reading From RXQ Example 2

In this section, an example shows how KS8851SNL driver (SPI master) read a **65 byte** (including CRC) frame from KSZ8851SNL (SPI slave) RXQ by using 8 bit SPI data transfers, assuming SPI master controller reads **4-byte data per SPI transaction cycle.**

Assuming that 'RX IP Header Two-Byte Offset Enable', bit 9 of RXQCR register is set to '1', there will be two extra bytes count that is included in the frame header 'Byte Count'.

The Start DMA Access , bit 3 of RXQCR register, is set to '1' before step 1. And it is cleared to '0' after step 29.

Steps Sequence	Operation	Pin Name	Value	Description
1	Set	CSN	Activate	Starting SPI operation.
2	Write	SI	0x80	RXQ Read CMD.
3	Read	SO	Dummy	Read out dummy 4 bytes.
4	Read	SO	Dummy	
5	Read	SO	Dummy	
6	Read	SO	Dummy	
7	Set	CSN	Deactivate	Stop SPI operation.
8	Set	CSN	Activate	Starting SPI operation.
9	Write	SI	0x80	RXQ Read CMD.
10	Read	SO	0xC8	Frame header 'Status Word'.
11	Read	SO	0x81	
12	Read	SO	0x43	Frame header 'Byte Count' is 67 bytes, which is 2 bytes of dummy data (due to 'IP Header Two-Byte Offset Enable') + 61 bytes of frame data + 4 bytes of CRC.
13	Read	SO	0x00	
14	Set	CSN	Deactivate	Stop SPI operation.
15	Set	CSN	Activate	Starting SPI operation.
16	Write	SI	0x80	RXQ Read CMD.

17	Read	SO	Dummy	Read out dummy 2 bytes due to 'IP Header Two-Byte Offset Enable'.
18	Read	SO	Dummy	
19	Read	SO	0xFF	Frame byte 1 (first byte of frame packet)
20	Read	SO	0xFF	Frame byte 2.
21	Set	CSN	Deactivate	Stop SPI operation.
22	…			Repeat for the reset of Frame data.
23	Set	CSN	Activate	Starting SPI operation.
24	Write	SI	0x80	RXQ Read CMD.
25	Read	SO	0x38	Last byte of frame CRC
26	Read	SO	Dummy data	Dummy read 66-68 to make number of DATA bytes read from RXQ must be in DWORD alignment.
27	Read	SO	Dummy data	
28	Read	SO	Dummy data	
29	Set	CSN	Deactivate	Stop SPI operation.

3 KSZ8851SNL Initialization Steps

Steps Sequence	Read\write	Register Name[bit]	Value	Description
1	Read	CIDER [15-0] 0ffset 0xC0	0x8872	Read the device chip ID, make sure it is correct ID 0x8872; otherwise there are some errors on the host bus interface.
2	Write	MARL[15-0] 0ffset 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] 0ffset 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] 0ffset 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] 0ffset 0x84	0x4000	Enable QMU Transmit Frame Data Pointer Auto Increment.
6	Write	TXCR [15-0] 0ffset 0x70	0x00EE	Enable QMU Transmit flow control / Transmit padding / Transmit CRC, and IP/TCP/UDP checksum generation.
7	Write	RXFDPR[15-0] 0ffset 0x86	0x4000	Enable QMU Receive Frame Data Pointer Auto Increment.
8	Write	RXFCTR[15-0] 0ffset 0x9C	0x0001	Configure Receive Frame Threshold for one frame.
9	Write	RXCR1 [15-0] 0ffset 0x74	0x7CE0	Enable QMU Receive flow control / Receive all broadcast frames /Receive unicast frames, and IP/TCP/UDP checksum verification etc.
10	Write	RXCR2 [15-0] 0ffset 0x76	0x009C	Enable QMU Receive UDP Lite frame checksum verification, UDP Lite frame checksum generation, IPv4/IPv6 UDP fragment frame pass, IPv4/IPv6 UDP UDP checksum field is zero pass, and single frame data burst if SPI master controller could read a single frame data in one SPI CSN activate[1].
11	Write	RXQCR[15-0] 0ffset 0x82	0x0230	Enable QMU Receive IP Header Two-Byte Offset /Receive Frame Count Threshold/RXQ Auto-Dequeue frame.
12	Write	OBCR[5-3]] 0ffset 0x20		Adjusts SPI Data Output (SO) Delay according to SPI master controller configuration.

[1] If SPI master controller could not read a single frame data in one SPI operation cycle, e.g. only could read 4 bytes data per SPI transaction cycle, then set RXCR2 bit 7-5 to '0'.

13.1	Write	P1CR[5] 0ffset 0xF6, 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] 0ffset 0xF6, bit 13	1	Restart Port 1 auto-negotiation.
14.1	Write	FCLWR[15-0] 0ffset 0xB0,	0x0600	Configure Low Watermark to 6KByte available buffer space out of 12KByte.
14.2	Write	FCHWR[15-0] 0ffset 0xB2,	0x0400	Configure High Watermark to 4KByte available buffer space out of 12KByte.
14	Write	ISR [15-0] 0ffset 0x92,	0xFFFF	Clear the interrupts status.
15	Write	IER [15-0] 0ffset 0x90,	0xE000	Enable Link Change\Transmit\Receive if your host processor can handle the interrupt, otherwise do not need to do this step.
16	Write	TXCR [0] 0ffset 0x70, bit 0	1	Enable QMU Transmit.
17	Write	RXCR1 [0] 0ffset 0x74, bit 0	1	Enable QMU Receive.

3.1 KSZ8851 Additional Receive Initialization Steps

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

Steps Sequence	Read\write	Register Name[bit]	Value	Description
8	Write	RXFCTR[15-0] 0ffset 0x9C	0x0004	Configure Receive Frame Threshold for multiplex frames, e.g. four frames.
8.1	Write	RXDTTR[15-0] 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.
11	Write	RXQCR[15-0] 0ffset 0x82	0x02B0	Enable QMU Receive IP Header Two-Byte Offset /Receive Frame Count Threshold/ Receive Duration Timer Threshold /RXQ Auto-Dequeue frame.

4 KSZ8851SNL 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. There are two variables are needed from the upper layer to transmit a data packet frame. (1). Packet data pointer (**pTxData**). It points to the host CPU system memory space contains the complete Ethernet packet data. (2). Packet length (**txPacketLength**). The Ethernet packet data length not includes CRC.			
1	Read	TXMIR [12-0] 0ffset 0x78	>= (**txPacketLength**+ +4+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, plus 4-byte for DWORD alignment. Compare the read value with (**txPacketLength**+4+4), if less than (**txPacketLength**+4+4), **Exit**.
2	Write	IER [15-0] 0ffset 0x90,	0000	Disable all the device interrupts generation.
3	Write	RXQCR[3] 0ffset 0x82 bit 3	1	Start[2] QMU DMA transfer operation to write frame data from host CPU to the **TxQ**.
6	Write		0x8000	Write TXIC to the "control word" of the frame header.
7	Write		**txPacketLength**	Write **txPacketLength** to the "byte count" of the frame header.

8	`UINT8 *pTxData;`
	`int lengthInDWord=((txPacketLength+3)>>2);`
	`int lengthInByte=(lengthInDWord *4);`
				Write frame data pointer by **pTxData** to the QMU TXQ in BYTE until finished the full packet length (**txPacketLength)** in DWORD alignment '**lengthInByte**'.

9	Write		`*pTxData++`	Write 1-byte of frame data pointer by pTxData to the QMU TXQ. Increase **pTxData** pointer by 1.

[2] Once QMU DMA transfer operation is started, host must not access to other device registers.

10	**lengthInByte --;** if **(lengthInByte** > 0) goto Step 9; else goto Step12;			Subtract **lengthInByte** by 1.
12	Write	RXQCR[3] 0ffset 0x82 bit 3	0	Stop QMU DMA transfer operation.
12.1	Write	TXQCR[0] 0ffset 0x80 bit 0	1	TxQ Manual-Enqueue.
13	Write	IER [15-0] 0ffset 0x90,	0xE000	Enable the device interrupts again. **Exit.**

5 KSZ8851SNL Receive Steps

There are two methods of receiving frames from QMU RXQ – One frame at a time or multiple frames at a time. 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.

5.1 KSZ8851SNL Receive Single Frame per transfer

The 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 frame from KSZ8851 device to upper layer either as a result of polling or servicing an interrupt. (1). By polling, set a timer routine to periodically execute step 1. (2). By servicing an interrupt, when interrupt occurs, execute step 1. 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] 0ffset 0x92, bit 13	1	Read value from ISR to check if RXIS 'Receive Interrupt' is set. If not set, **Exit.**
2	Write	IER [15-0] 0ffset 0x90,	0000	Disable all the device interrupts generation.
3	Write	ISR [13] 0ffset 0x92, bit 13	1	Acknowledge (clear) RXIS Receive Interrupt bit.
4	Read	RXFCTR[15-8] 0ffset 0x9C	**rxFrameCount**	Read current total amount of received frame count from RXFCTR, and save in '**rxFrameCount**'.
5	if (**rxFrameCount** > 0) goto Step 6; else goto step 24;			Repeatedly reading all frames from RXQ. If **rxFrameCount** <= 0, **goto step 24**
6	Read	RXFHSR [15-0] 0ffset 0x7C	**rxStatus**	Read received frame status from RXFHSR to check if this is a good frame.
7	Read	RXFHBCR [11-0] 0ffset 0x7E	**rxPacketLength**	Read received frame byte size from RXFHBCR to get this received frame length (4-byte CRC, and extra 2-byte due to 'IP Header Two-Byte Offset Enable'

				are included), and store into **rxPacketLength** variable. 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 an error frame, **goto step 8,** Else received a good frame, **goto step 9.** if **rxPacketLength** <= 0**, goto step 8;** else **goto step 9**;
8	Write	RXQCR [0] 0ffset 0x82 bit 0	1	Issue the RELEASE error frame command for the QMU to release the current error frame from RXQ. **goto step 23**;
9	Write	RXFDPR[15-0] 0ffset 0x86	0x4000	Reset QMU RXQ frame pointer to zero.
10	Write	RXQCR[3] 0ffset 0x82 bit 3	1	Start QMU DMA transfer operation to read frame data from the RXQ to host CPU.
13	Read		**pDummy**	Must read out dummy 4-byte.
14	Read		**pDummy**	Read out 2-byte 'Status Word' of frame header from the QMU RXQ.
15	Read		**pDummy**	Read out 2-byte 'Byte Count' of frame header from the QMU RXQ.

17	`UINT8 *`**`pRxData;`**
	int **lengthInDWord**=((**rxPacketLength** +3)>> 2);
	int **lengthInByte**=(**lengthInDWord** * 4);
				Read frame data to system memory pointer by **`pRxData`** from the QMU RXQ in BYTE until finished the full packet length (**rxPacketLength)** in DWORD alignment '**lengthInByte**.

18	Read		`*`**`pRxData ++`**	Read 1-byte of frame data to system memory pointer by **pRxData** from the QMU RXQ. Increase **`pRxData`** pointer by 1.
19	**lengthInByte** --; if (**lengthInByte** > 0) goto Step 18; else goto Step 20;			Subtract **lengthInByte** by 1.
21	Write	RXQCR[3] 0ffset 0x82 bit 3	0	Stop QMU DMA transfer operation.

22	Pass this received frame to the upper layer protocol stack. 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). 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: (1). Increase data pointer **`pRxData`** by 2-byte to the beginning of Ethernet packet data , **`pRxData += 2;`** (2). Minus 2 extra bytes from '**rxPacketLength**' to the upper layer. **rxPacketLength** -= 2; (3). Minus 4-byte CRC length from '**rxPacketLength**' to the upper layer. **rxPacketLength** -= 4; (4). Pass received frame to upper layer protocol stack. toUpperLayer (**`pRxData`**, **rxPacketLength**);			
23	**rxFrameCount = rxFrameCount – 1;** goto step 5 .			Finished reading one frame, subtract **rxFrameCount** by 1. Loop again.
24	Write	IER [15-0] 0ffset 0x90	0xE000	Enable the device interrupts again. Exit.

5.2 KSZ8851SNL Receive Multiple Frames per Transfer

The 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.
	
	(1). By polling, set a timer routine to periodically execute step 1.
	(2). By servicing an interrupt, when interrupt occurs, execute step 1.
	
	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'**.
	Eg, the sample array structure to store received multiplex frame header information:
	`typedef struct {`
	`  USHORT  rxStatus;`
	`  USHORT  rxLength;`
	`} FR_HEADER_INFO;`
	FR_HEADER_INFO rxFrameHeader[MAX_FRAMES_IN_RXQ];
	
	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] 0ffset 0x92, bit 13	1	Read value from ISR to check if RXIS 'Receive Interrupt' is set. If not set, **Exit.**
2	Write	IER [15-0] 0ffset 0x90,	0000	Disable all the device interrupts generation.
3	Write	ISR [13] 0ffset 0x92, bit 13	1	Acknowledge (clear) RXIS Receive Interrupt bit.
4	Read	RXFCTR[15-8] 0ffset 0x9C	**rxFrameCount**	Read current total amount of received frame count from RXFCTR, and save in '**rxFrameCount**'.

5	int i = 0;
	if (**rxFrameCount** > 0) goto Step 6;
	else goto step 9;
				Repeatedly reading all frames header information from RXGHSR and RXFHBCR.
				If **rxFrameCount** <= 0, **goto step 9.**

6	Read	RXFHSR [15-0] 0ffset 0x7C	**rxFrameHeader[i].rxStatus**	Read received frame status from RXFHSR to '**rxStatus**' array variable.
7	Read	RXFHBCR [11-0] 0ffset 0x7E	**rxFrameHeader[i].rxLength**	Read received frame byte size from RXFHBCR to **'rxLength'** array variable.

8	**rxFrameCount = rxFrameCount – 1;** **i +=1;** goto step 5 .			Finished store one frame header information, subtract **rxFrameCount** by 1, Increase array index by 1. Loop again.
9	**rxFrameCount = i;** i=0;			Restore total amount of received frame count '**rxFrameCount**' again.
10	Write	RXFDPR[15-0] 0ffset 0x86	0x4000	Reset QMU RXQ frame pointer to zero.
11	Write	RXQCR[3] 0ffset 0x82 bit 3	1	Start QMU DMA transfer operation to read frame data from the RXQ to host CPU.
13	Read		**pDummy**	Must read out dummy 4-byte.
14	if (**rxFrameCount** > 0) goto Step 15; else goto step 34;			Loop reading all frames from RXQ. If **rxFrameCount** <= 0, **goto step 34.**

15	#define **RX_ERRORS** 0x3C17
	if ((**rxFrameHeader[i]. rxStatus** & RX_ERRORS) ‖
	(**rxFrameHeader[i]. rxLength** <= 0))
	error frame, goto step 16;
	else
	good frame, goto step 25;
				Check received frame status **'rxFrameHeader[i]. rxStatus'** to see if this is a good frame, and received frame length '**rxFrameHeader[i]. rxLength'.**

17	Write	RXQCR[3] 0ffset 0x82 bit 3	0	This is an error frame. Stop QMU DMA transfer operation.
18	Write	RXQCR[0] 0ffset 0x82 bit 0	1	Issue the RELEASE error frame command for the QMU to release the current error frame from RXQ.
19	Write	RXFDPR[15-0] 0ffset 0x86	0x4000	Reset QMU RXQ frame pointer to zero.
20	Write	RXQCR[3] 0ffset 0x82 bit 3	1	Then, Start the DMA transfer operation again for the next frame.
23	Read		**pDummy**	Must read out dummy 4-byte from QMU RXQ.
24	goto step 33;			Go for processing the next frame.
25	Read		**pDummy**	Read out 2-byte 'Status Word' of frame header from the QMU RXQ.
26	Read		**pDummy**	Read out 2-byte 'Byte Count' of frame header from the QMU RXQ.

28	`UINT8 *pRxData;`
	int **lengthInDWord** =
	((**rxFrameHeader[i]. rxLength** +3)>>2);
	int **lengthInByte**=(**lengthInDWord** * 4);
				Read frame data to system memory pointer by **pRxData** from the QMU RXQ in BYTE until finished the full packet length '**rxFrameHeader[i]. rxLength'** in DWORD alignment '**lengthInByte**'.

29	Read		*`pRxData ++`	Read 1-byte of frame data to system memory pointer by pRxData from the QMU RXQ. Increase `pRxData` pointer by 1.
30	**lengthInByte** --; if (**lengthInByte** > 0) goto Step 29; else goto Step 31;			Subtract **lengthInByte** by 1.
31	Set	**CSN**	Deactivate	Set SPI mater controller CSN deactivates to stop SPI operation.
32	Pass this received frame to the upper layer protocol stack. 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). 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: (1). Increase data pointer `pRxData` by 2-byte to the beginning of Ethernet packet data , `pRxData += 2;` (2). Minus 2 extra bytes from '**rxPacketLength**' to the upper layer. **rxLength** -= 2; (3). Minus 4-byte CRC length from '**rxPacketLength**' to the upper layer. **rxLength** -= 4; (4). Pass received frame to upper layer protocol stack. toUpperLayer (`pRxData`, **rxFrameHeader[i]. rxLength**);			
33	**rxFrameCount = rxFrameCount – 1;** **i +=1;** goto step 14 .			Finished reading one frame, subtract **rxFrameCount** by 1. Increase array index by 1. Loop again.
34	Write	RXQCR[3] 0ffset 0x82 bit 3	0	Stop QMU DMA transfer operation.
35	Write	IER [15-0] 0ffset 0x90	0xE000	Enable the device interrupts again. Exit.

5.3 KSZ8851 Receiver Interaction with OS Device Driver

This table shows what is 'frame count' should be from register RXFCTR when the device generates RXIE interrupt event after it received first frame with continuous frames injecting to the device from Network. The example is base on 'Receive Frame Count Threshold' is set to 1 in register RXFCTR when bit 5 set to 1 in register RXQCR.

Time	Sender (PC)	Receiver (KSZ8851)	OS (KSZ8851 Device Driver)
0	Sends frame A	receives A, Generate RXIE interrupt to Host.	OS receive RXIE interrupt, schedule to call driver ISR
1	sends frame B	receives B	Driver ISR is called, ISR do: 1. disable device interrupt, 2. clear RXIE from register ISR, 3. read 'frame count' (it should be 2 or 3 frames dependent on how fast ISR is scheduled, we assume 2 for now) 4. Read frame A, and B. 5. enable device interrupt. Note: while driver is doing above steps, the device is continuous receiving frame C, D….
2	sends frame C	receives C	
3	sends frame D	receives D Generate RXIE interrupt to Host.	OS receive RXIE interrupt, schedule to call driver ISR
4	sends frame E	receives E	Driver ISR is called, ISR do: 1. disable device interrupt, 2. clear RXIE from register ISR, 3. read 'frame count' (it should be 2 or 3 frames dependent on how fast ISR is scheduled, we assume 3 for now) 4. Read frame C, D, and E. 5. enable device interrupt. Note: while driver is doing above steps, the device is continuous receiving frame …
	…	…	

The table below shows how the device updates 'frame count' in register RXFCTR, received 'frame status' in register RXFHSR, and received 'byte count' in register RXFHBCR.

Time	Sender (PC)	KSZ8851 Device Action	Trigger	KSZ8851 Driver Action
0	Sends frame A, B, C, …	Generate RXIE interrupt to Host.		
1		Update 'frame count' in register RXFCTR	←	Write '1' in RXIE bit on register ISR
2		Update 1st received 'frame status' in register RXFHSR, and 'byte count' in register RXFHBCR	←	Read 'frame count' in register RXFCTR. Assumes it is 3 frames.
3		Update 2nd received 'frame status' in	←	Read 1st received 'frame status' from

		register RXFHSR, and 'byte count' in register RXFHBCR		register RXFHSR, then 'byte count' from register RXFHBCR.
4		Update 3rd received 'frame status' in register RXFHSR, and 'byte count' in register RXFHBCR	←	Read 2nd received 'frame status' from register RXFHSR, then 'byte count' from register RXFHBCR.
5				Read 3rd received 'frame status' from register RXFHSR, then 'byte count' from register RXFHBCR.

5.4 KSZ8851 ISR

If Host operation system (OS) could handle interrupts generated by the device, then the driver should create an Interrupt Server Routine (ISR). This section provides basically guide line of how to write an Interrupt Server Routine (ISR) for KSZ8851MLL.

Steps Sequence	Read\write	Register Name[bit]	Value	Description
1	Write	IER [15-0] 0ffset 0x90,	0000	Disable all the device interrupts generation.
2	Read	ISR [15:0] 0ffset 0x92,		Read interrupt event from register ISR; Acknowledge (write '1') to register ISR to clear Interrupt event; Process every interrupt event.
3	Write	ISR [15:0] 0ffset 0x92,	'1'	
4	Write	IER [15-0] 0ffset 0x90,	0xE000	Enable the device interrupts again.