Ubicom32 SLIC Block Programming

Application Note

Ubicom32 processors such as IP7000 have a SLIC virtual hardware block, henceforth referred to as SLIC. The virtual SLIC hardware block behaves similar to other hardware blocks found on other processors. The SLIC block is accessible through a memory mapped interface. It has number of configurable parameters, such as buffer sizes, number of channels, etc.

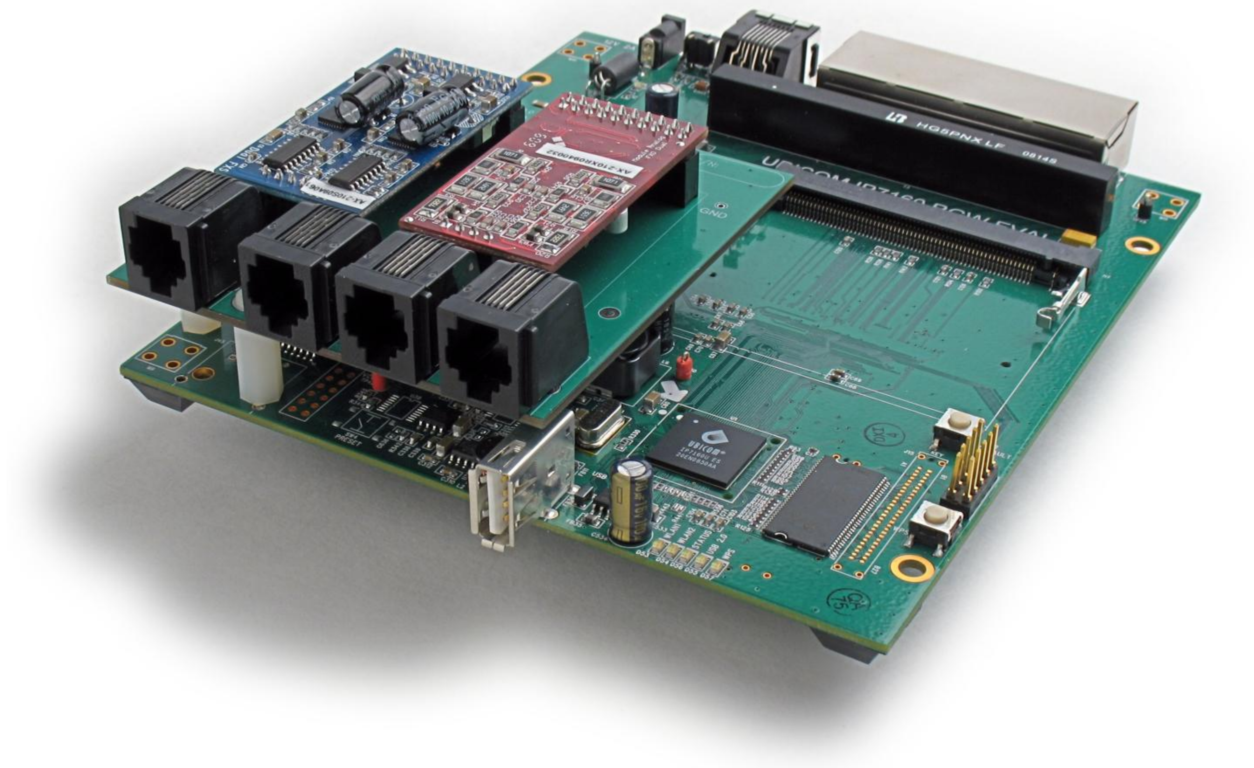

Figure1. IP7160 RGW Evaluation board with SLIC module.

Contents

1. Timing Diagram

Figure 2 shows the output of the SLIC. The active edge of PCLK is the falling edge.

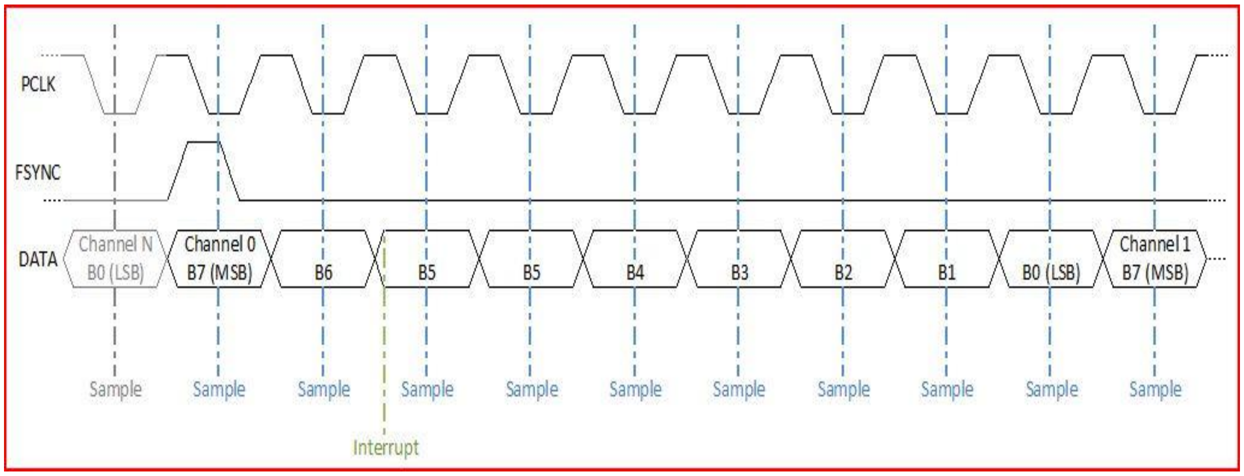

Figure 2. Timing of UBICOM32 SLIC block.

2. Registers

The SLIC block is controlled and used through the following 32 bit registers. The addresses of the registers are offsets from a base address, and the read/write operations are indicated through R/W respectively. Please refer to the section of SLIC usage to determine the base address. ***Note that UBICOM32 processors are big-endian.***

Bit 31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0

Figure 3. Ubicom32 Register

2.1. Reload - 0x0 – RW

Bit 0 - Set to 1 to reload the parameters and restart the hardware block.

Bit 1:31 - Reserved (not used).

2.2. Input Buffer - 0x4 – R

Bit 0:31 - Pointer to input buffer. Address of the Memory mapped area of the input buffer of SLIC.

2.3. Output Buffer - 0x8 – R

Bit 0:31 - Pointer to output buffer. Address of the memory mapped area of the output buffer of SLIC.

2.4. Buffer Size - 0xC – RW

Bit 0:31 – Current buffer size. Set the reload register to use the new buffer size.

The buffers of SLIC act like ping-pong buffers. For a single channel, the buffer size is twice the size of the input buffer that the driver intends to use. For an N-channel, this buffer size is N * 2 * `driver_input_read_size`.

Note:

1. Must be a power of 2.
2. Must be evenly divisible by the (number-of-channels * 2)

Example:

If the telephony adapter driver has 4 channels and wants to get 16 samples every interrupt for each channel, the buffer size is (16 * 8) * 2 * 4 = 1024 bytes.

2.5. Cycle - 0x10 – R

Bit 0:31 – Counter that increases every time half of the buffer is consumed. Since the buffers of SLIC are ping-pong in nature, the telephony adapter needs to read this counter and determine which half of the buffer it needs to read or write.

2.6. Version - 0x14 – R

Bit 0:31 – Version of the SLIC block. The telephony adapter driver can check this field to do a sanity check during initialization to see if the driver and hardware block are in sync with respect to the SLIC register space.

2.7. Max Buffer Size - 0x20 – R

Bit 0:31 – Maximum buffer size. The telephony interface driver can use this field to check if the buffer size is in less than or equal to the maximum buffer size allowed by the SLIC block. The maximum buffer size can be changed at project configuration time.

3. Buffer Usage and Interrupt Generation

The active edge of PCLK is the falling edge. To make PCLK rising edge, change the initial state of PCLK from HIGH to LOW.

The pointer to data to send is stored in Output Buffer Pointer. The channels are stored in memory as follows <0><1>...<n–1><n><0>.... Output data is changed after the rising edge of PCLK.

The pointer to the location to put received data is stored in Input Buffer pointer. The channels are stored in memory as follows <0><1>...<n–1><n><0>.... Input data is sampled after the falling edge of PCLK. The first sample of channel 0 after the HRT is (re)started will be indeterminate.

An interrupt is issued when the HRT has consumed 1/2 of the buffer.

Note: The last received sample will not be valid until approximately 6 bit times after the interrupt was signaled (about 11.7us at 512 kHz PCLK; see above.)

4. SLIC Module Usage

This section describes the usage of the SLIC module (shown in Figure 1, on the front page).

4.1. Base Address of SLIC

The SLIC block is accessible through a memory mapped interface. The SLIC registers are accessible using a device tree (**devtree**) structure which can be obtained as described.

```
pcm_node = (struct pcm_tio_node *)devtree_find_node("pcmtio");
if (pcm_node) {
    ip7160rgw_ubicom32fx_platform_data.pcm_node = pcm_node;
    ip7160rgw_ubicom32fx_platform_data.pin_reset = GPIO_RI_5;
    ip7160rgw_ubicom32fx_platform_data.irq_fx = gpio_to_irq(GPIO_RA_5); <<
    interrupt
    printk(KERN_INFO "%s: registering SPI resources bus 1\n", __FUNCTION__);
    spi_register_board_info(ip7160rgw_spi_1_board_info,
    ARRAY_SIZE(ip7160rgw_spi_1_board_info));
}
```

Folder **Ubicom-distro/linux-2.6/arch/ubicom32/include/asm** has the devtree and the corresponding nodes defined for the hardware blocks, for example **pcmtio.h**:

```
struct pcm_tio_node {
    struct devtree_node   dn;
    u32_t                 version;
    struct pcm_tio_regs   *regs;
};
```

```
struct pcm_tio_regs {
    u32_t           reload;
    void      *input_buf;
    void      *output_buf;
    u32_t           buffer_size;
    u32_t           cycle;
    u32_t           version;
    u32_t           channels;
    u32_t           max_buffer_size;
};
```

4.2. Example of SLIC Buffer Usage

```
static inline void ubicom32fx_receiveprep(struct ubicom32fx *ubicom32fx,
  unsigned char ints)
{
  volatile unsigned int *readchunk;
  int x;

  if (ints & 0x08)
    readchunk = ubicom32fx->readchunk + DAHDI_CHUNKSIZE;
  else
    /* Read is at interrupt address.  Valid data is available at normal
  offset */
    readchunk = ubicom32fx->readchunk;
  for (x=0;x<DAHDI_CHUNKSIZE;x++) {
    if (ubicom32fx->cardflag & (1 << 3))
      ubicom32fx->chans[3]->readchunk[x] = (readchunk[x]) & 0xff;
    if (ubicom32fx->cardflag & (1 << 2))
      ubicom32fx->chans[2]->readchunk[x] = (readchunk[x] >> 8) & 0xff;
    if (ubicom32fx->cardflag & (1 << 1))
      ubicom32fx->chans[1]->readchunk[x] = (readchunk[x] >> 16) & 0xff;
    if (ubicom32fx->cardflag & (1 << 0))
      ubicom32fx->chans[0]->readchunk[x] = (readchunk[x] >> 24) & 0xff;
  }
…..
}
```

4.3. Pin Definitions on RGW Eval Board

Below are the pin mappings on the RGW eval board. Note that these are given as an example and should not be taken to work for all boards.

Signal	Pin	Defined in ...
FSYN	I0	ubicom-dstro/ultra/pkg/ipPCMTIO/pkginfo/ipPCMTIO.pkg
DTX	I1	
DRX	I2	
PCLCK	I3	
SDO	I13	ubicom-distro / linux-2.6.x/arch/ubicom32/mach-ip7k/board-ip*.c
SD1	I11	
SCLCK	I14	
RESET	I5 0	ubicom-distro / linux-2.6.x/arch/ubicom32/mach-ip7k/board-ip*.c
CS1	I7	ubicom-distro / linux-2.6.x/arch/ubicom32/mach-ip7k/board-ip*.c
CS2	I10	
CS3	I9	
CS4	I8	

4.4. Buffer Interrupt from SLIC

The buffer interrupt is a software interrupt that comes from the SLIC through memory mapped interface in the **dev_tree_node** found in **devtree.h**.

```
struct devtree_node {
struct devtree_node *next;
unsigned char sendirq;
unsigned char recvirq;
char name[DEVTREE_MAX_NAME];
unsigned int magic;
};
```

Application Note

Ubicom®, Inc.

Ubicom develops networking and media processor solutions that address the unique demands of real-time interactive applications and multimedia content delivery in the digital home. The company provides optimized system-level solutions for a wide range of products including wireless routers, access points, bridges, VoIP gateways, networked digital photo frames, and streaming media players.

UBICOM

195 Baypointe Parkway
San Jose, CA 94134

Tel	408.433.3300
Fax	408.433.3339
Email	sales@ubicom.com
Web	www.ubicom.com