SigmaStar Camera I2C User Guide

Version 0.1

REVISION HISTORY

Revision No.	Description	Date
0.1	• Initial release	12/18/2019

TABLE OF CONTENTS

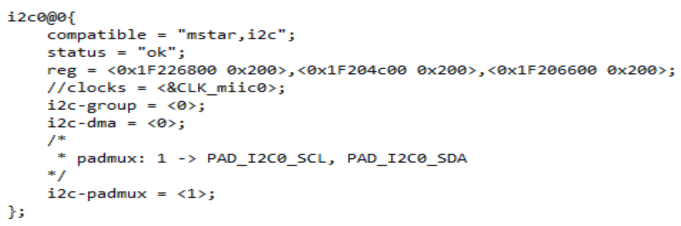
1. OVERVIEW

1.1. General Description

I2C Group	SCL	SDA	DEV
HW I2C group0	PAD_I2C0_SCL	PAD_I2C0_SDA	/dev/i2c-0
HW I2C group1	PAD_I2C1_SCL	PAD_I2C1_SDA	/dev/i2c-1,

As illustrated in the above table, there are two groups of HW I2C. The first group corresponds to PAD_I2C0_SCL/PAD_I2C0_SDA and the device node /dev/i2c-0. The second group corresponds to PAD_I2C1_SCL/PAD_I2C1_SDA and the device node /dev/i2c-1.

If not all of the two groups of I2C are used, refer to the definition "#define HAL_HWI2C_PORTS 2" in mhal_iic.h and the I2C device node in infinity6b0.dtsi. Normally, the definition in i2c-group = <0> denotes the corresponding device node.


```
i2c0@0{
    compatible = "mstar,i2c";
    status = "ok";
    reg = <0x1F226800 0x200>,<0x1F204c00 0x200>,<0x1F206600 0x200>;
    //clocks = <&CLK_miic0>;
    i2c-group = <0>;
    i2c-dma = <0>;
    /*
     * padmux: 1 -> PAD_I2C0_SCL, PAD_I2C0_SDA
    */
    i2c-padmux = <1>;
};
```

The example illustrated below is given based on the first I2C group. For usage of the remaining groups, refer to the first group directly.

2. USING I2C

2.1. I2C Read/Write

Below is an example of I2C read/write via standard ms_i2c_xfer function.

```
#include <stdio.h>
#include <linux/types.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <assert.h>
#include <string.h>
#include <linux/i2c.h>
#include <linux/i2c-dev.h>

#define FILE_NAME "/dev/i2c-0"

static int i2c_write(int fd,unsigned char slave_addr, unsigned char reg_addr, unsigned char value)
{
    unsigned char outbuf[2];
    struct i2c_rdwr_ioctl_data packets;
    struct i2c_msg messages[1];

    messages[0].addr   = slave_addr;
    messages[0].flags = 0;
    messages[0].len    = sizeof(outbuf);
    messages[0].buf    = outbuf;

    /* The first byte indicates which register we ‘ll write */
    outbuf[0] = reg_addr;

    /*
     * The second byte indicates the value to write.   Note that for many
     * devices, we can write multiple, sequential registers at once by
     * simply making outbuf bigger.
     */
```

```
        outbuf[1] = value;

        /* Transfer the i2c packets to the kernel and verify it worked */
        packets.msgs    = messages;
        packets.nmsgs = 1;
        if(ioctl(fd, I2C_RDWR, &packets) < 0)
        {
            perror("Unable to send data");
            return 1;
        }

        return 0;
    }

    static int i2c_read(int fd, unsigned char slave_addr, unsigned char reg_addr, unsigned char *value)
    {
        unsigned char inbuf, outbuf;
        struct i2c_rdwr_ioctl_data packets;
        struct i2c_msg messages[2];

        /*
         * In order to read a register, we first do a "dummy write" by writing
         * 0 bytes to the register we want to read from.   This is similar to
         * the packet in set_i2c_register, except it ‘s 1 byte rather than 2.
         */
        outbuf = reg_addr;
        messages[0].addr   = slave_addr;
        messages[0].flags = 0;
        messages[0].len     = sizeof(outbuf);
        messages[0].buf     = &outbuf;

        /* The data will get returned in this structure */
        messages[1].addr   = slave_addr;
        messages[1].flags = I2C_M_RD/* | I2C_M_NOSTART*/;
        messages[1].len     = sizeof(inbuf);
        messages[1].buf     = &inbuf;

        /* Send the request to the kernel and get the result back */
        packets.msgs          = messages;
        packets.nmsgs        = 2;
        if(ioctl(fd, I2C_RDWR, &packets) < 0)
        {
            perror("Unable to send data");
            return 1;
```

```
	}
	*value = inbuf;

	return 0;
}

int main(int argc, char **argv)
{
	int fd;
	unsigned int slave_addr=0, reg_addr=0, value = 0;

	if (argc < 4){
		printf("Usage:\n%s r[w] start_addr reg_addr [value]\n",argv[0]);
		return 0;
	}

	fd = open(FILE_NAME, O_RDWR);
	if (!fd)
	{
		printf("can not open file %s\n", FILE_NAME);
		return 0;
	}

	sscanf(argv[2], "%x", &slave_addr);
	sscanf(argv[3], "%x", &reg_addr);

	if(!strcmp(argv[1],"r"))
	{
		i2c_read(fd, slave_addr, reg_addr, (unsigned char*)&value);
	}
	else if(argc>4&&!strcmp(argv[1],"w"))
	{
		sscanf(argv[4], "%x", &value);
		i2c_write(fd, slave_addr, reg_addr, value);
	}

	close(fd);
	return 0;
}
```