YM2413 FM Synthesizer Driver Interface
Group: BareMetalMusic (“Project Group”)
Members: Ricky Ren, Daniel Greenwood, Irfandi Riawan
Course: CMPT433 – Embedded Systems

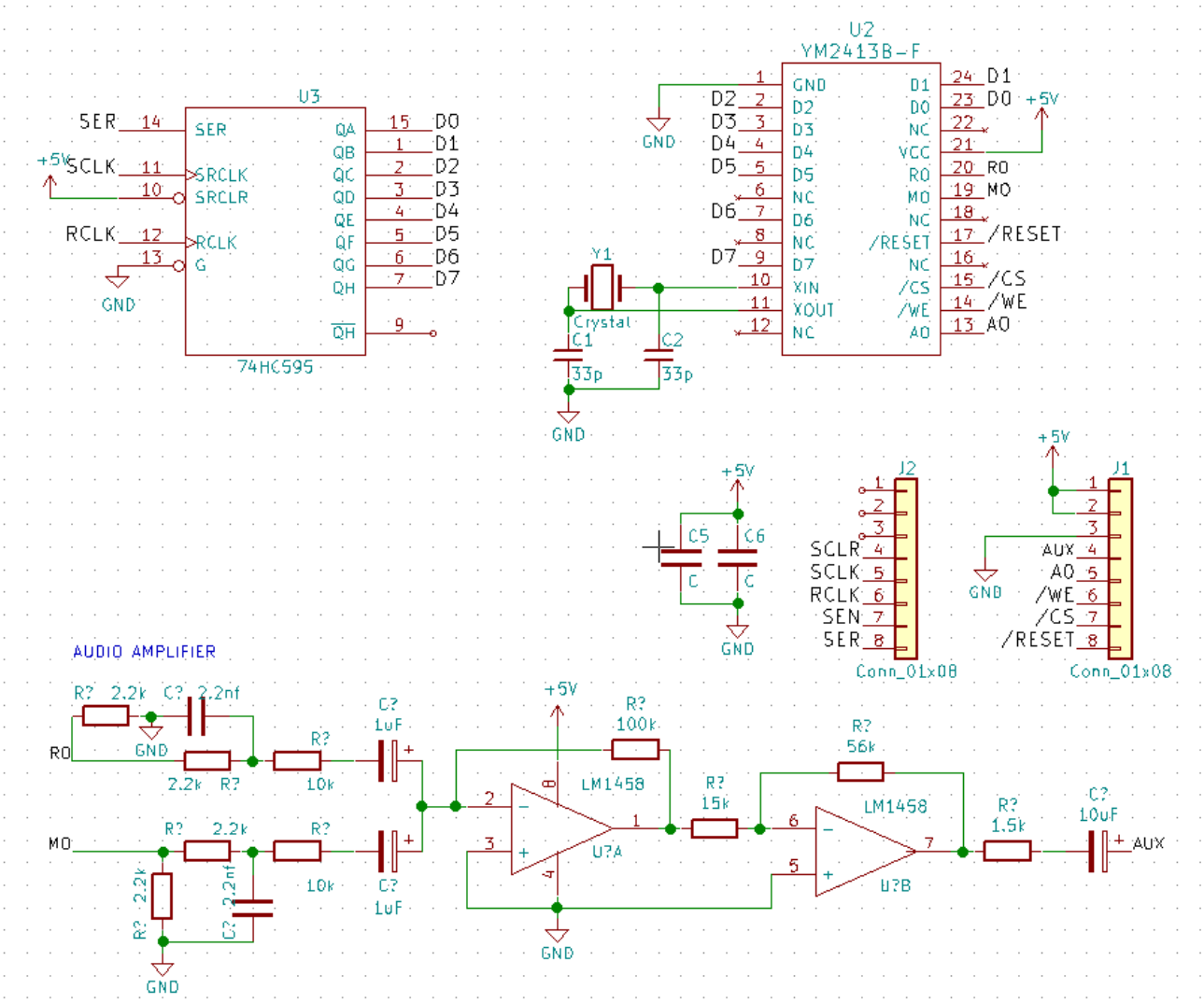
- This guide outlines controlling an FM synthesizer IC through a shift register and GPIO kernel driver.
- Follow Dr. Fraser’s kernel driver guide in order to set up the appropriate build environment and install the kernel module.
- Read the following application manual for more information about the YM2413:
  - http://www.smspower.org/maxim/Documents/YM2413ApplicationManual
- Read the following datasheet for more information about the HC595 shift register:
  - http://www.ti.com/lit/ds/symlink/sn74hc595.pdf

Hardware:

Connect the hardware to the Beaglebone's GPIO as follows:

GPIO	YM2413B	HC595	Audio Amp	Description
66	CS			!Chip Select
67	WE			!Write Enable
68	Ao			Address/Data
69	RS			!Reset
79		SCK		Serial Clock
80		RCK		Register Clock
81		SER		Serial Data
GND	GND	GND	GND	Ground
VPP[1]	VPP	VPP	VPP	5V (+/- 0.5V)
	RO		RO	Rhythm Output
	MO		MO	Modulator Output

1 An external 5V DC supply may be used instead and will reduce signal noise; in that case, only connect the ground of the supply to the beaglebone's ground.



Note: the crystal oscillator should be about 3.579545 MHz, as specified in the application manual.

Code:

Driver side C code:

Define the GPIO pins and device file and included libraries:

```
#include <linux/module.h>
#include <linux/miscdevice.h>
#include <linux/fs.h>
#include <linux/delay.h>
#include <asm/uaccess.h>
#include <linux/slab.h>
#include <linux/init.h>
#include <linux/proc_fs.h>
#include <linux/mutex.h>
#include <asm/string.h>
#include <linux/gpio.h>
#include <linux/delay.h>

#define DEVICE_FILE "hc595"
#define gpio_cs 66
#define gpio_we 67
#define gpio_a0 68
#define gpio_rs 69
#define gpio_sck 79
#define gpio_rck 80
#define gpio_ser 81
```

Code to reset and set the shift register:

```
static void reset(void){
    gpio_set_value(gpio_rs,0);
    msleep(1);
    gpio_set_value(gpio_rs,1);
}
static void push_byte(char byte){
    int i;
    printk(KERN_INFO "hc595: pushing %d\n",byte);
    //set the register clock to zero
    gpio_set_value(gpio_rck,0);
    for(i=7;i>=0;i--){
        //shift in each bit, which is latched on the postitive edge of the serial clock
        gpio_set_value(gpio_sck,0);
        gpio_set_value(gpio_ser,byte>>i);
        gpio_set_value(gpio_sck,1);
    }
    //output registers are latched on the positive edge
    gpio_set_value(gpio_rck,1);
}
```

Code to set the ym2413's registers:

```
static void ym_write(char addr,char data){
    //disable the ym2413 while we set the data/address bus
    gpio_set_value(gpio_cs,1);
    gpio_set_value(gpio_we,1);

    //set the ym2413 to address latch mode
    gpio_set_value(gpio_a0,0);
    //set the shift register to the desired ym2413 register address
    push_byte(addr);
    //latch in the address by driving the write enable and chip select low, then
high
    gpio_set_value(gpio_we,0);
    gpio_set_value(gpio_cs,0);
    gpio_set_value(gpio_we,1);
    gpio_set_value(gpio_cs,1);

    //set the ym2413 to data latch mode
    gpio_set_value(gpio_a0,1);
    //set the desired contents of the ym2413 register
    push_byte(data);
    //latch in the data by driving the write enable and chip select low, then high
    gpio_set_value(gpio_we,0);
    gpio_set_value(gpio_cs,0);
    gpio_set_value(gpio_we,1);
    gpio_set_value(gpio_cs,1);
}
```

Initialization code:

```
#define GPIO_SETUP(a,b)\
    gpio_request(a,"sysfs");\
    gpio_direction_output(a,b);\
    gpio_export(a,false);
#define GPIO_TEARDOWN(a,b)\
    gpio_set_value(a,b);\
    gpio_unexport(a);\
    gpio_free(a);
static int __init hc595_init(void){
    int ret;
    printk(KERN_INFO "hc595: init\n");
    ret=misc_register(&hc595_device);
    //Put the ym2413 into an inhibited state and clear the shift register
    GPIO_SETUP(gpio_cs,1)
    GPIO_SETUP(gpio_we,1)
    GPIO_SETUP(gpio_a0,0)
    GPIO_SETUP(gpio_rs,1)
```

```
    GPIO_SETUP(gpio_ser,0)
    GPIO_SETUP(gpio_sck,0)
    GPIO_SETUP(gpio_rck,0)
    reset();
    return ret;
}
static void __exit hc595_exit(void){
    printk(KERN_INFO "hc595: exit\n");
    misc_deregister(&hc595_device);
    GPIO_TEARDOWN(gpio_cs,1)
    GPIO_TEARDOWN(gpio_we,1)
    GPIO_TEARDOWN(gpio_a0,0)
    GPIO_TEARDOWN(gpio_rs,1)
    GPIO_TEARDOWN(gpio_ser,0)
    GPIO_TEARDOWN(gpio_sck,0)
    GPIO_TEARDOWN(gpio_rck,0)
}
module_init(hc595_init);
module_exit(hc595_exit);
MODULE_AUTHOR("Daniel Greenwood");
MODULE_DESCRIPTION("HC595 GPIO driver");
MODULE_LICENSE("GPL");
```

User side C code:

Constants and Register offsets:

```
//YM2413 Register Address Map
//See application manual for a handy chart
//Tone Registers
#define YM_REG_MOD_CTRL     0x0
#define YM_REG_CAR_CTRL     0x1
#define YM_REG_KEYSCALE_0   0x2
#define YM_REG_KEYSCALE_1   0x3
#define YM_REG_MOD_INDEX    0x2
#define YM_REG_DISTORT      0x3
#define YM_REG_ENV_ATTACK   0x4
#define YM_REG_ENV_DECAY    0x5
#define YM_REG_ENV_SUSTAIN  0x6
#define YM_REG_ENV_RELEASE  0x7
//Rhythm Control
#define YM_REG_RHYTHM       0xE
//Test Data
#define YM_REG_TEST         0xF
//F-Number
#define YM_REG_F_LSB_0      0x10
#define YM_REG_F_LSB_1      0x11
#define YM_REG_F_LSB_2      0x12
```

```
#define YM_REG_F_LSB_3      0x13
#define YM_REG_F_LSB_4      0x14
#define YM_REG_F_LSB_5      0x15
#define YM_REG_F_LSB_6      0x16
#define YM_REG_F_LSB_7      0x17
#define YM_REG_F_LSB_8      0x18
//F-Number, Octave, Key/Sustain on/off
#define YM_REG_F_MSB_0      0x20
#define YM_REG_F_MSB_1      0x21
#define YM_REG_F_MSB_2      0x22
#define YM_REG_F_MSB_3      0x23
#define YM_REG_F_MSB_4      0x24
#define YM_REG_F_MSB_5      0x25
#define YM_REG_F_MSB_6      0x26
#define YM_REG_F_MSB_7      0x27
#define YM_REG_F_MSB_8      0x28
//Instrument Selection and Volume
#define YM_REG_INST_0       0x30
#define YM_REG_INST_1       0x31
#define YM_REG_INST_2       0x32
#define YM_REG_INST_3       0x33
#define YM_REG_INST_4       0x34
#define YM_REG_INST_5       0x35
#define YM_REG_INST_6       0x36
#define YM_REG_INST_7       0x37
#define YM_REG_INST_8       0x38

//bitmasks
#define LSB 0b11111111
#define MSB 0b100000000
#define VOLUME_MASK 0b00001111

#define hc595_device_file "/dev/hc595"

//supported voices and rhythm instruments
typedef enum{
    ORIGINAL=0,
    VIOLIN,
    GUITAR,
    PIANO,
    FLUTE,
    CLARINET,
    OBOE,
    TRUMPET,
    ORGAN,
    HORN,
    SYNTH,
    HARPSICORD,
```

```
    VIBRAPHONE,
    SYNTH_BASS,
    ACCOUSTIC_BASS,
    ELECTRIC_GUITAR,
    BASS_DRUM=0,
    SNARE_DRUM,
    TOM,
    CYMBAL,
    HAT
}VOICES;
```

Constants for the frequency number the YM2413 uses to generate musical notes:

```
const int F_NUMBER[]={
    181,
    192,
    204,
    216,
    229,
    242,
    257,
    272,
    288,
    305,
    323,
    343
};
const float NOTE_FREQ[]={
    261.6256,
    277.1826,
    293.6648,
    311.1270,
    329.6276,
    349.2282,
    369.9944,
    391.9954,
    415.3047,
    440.0,
    466.1638,
    493.8833
};
//Exercise left for the reader:
//Based on the application manual, write a function to generate the fnumber
```

Write an address/data pair to the driver:

```
void ym_write_value(char addr, char data){
        FILE *fp=fopen(hc595_device_file,"w");
    if(fp!=NULL){
```

```
        //setvbuf(fp,NULL,_IONBF,0);
        fprintf(fp,"%c%c",addr,data);
        fclose(fp);
    }else{
        printf("write failure\n");
    }
    return;
}
```

Set the note volume and key on:

```
//voice mode for pitched instruments
void ym_set_voice_volume(int channel, VOICES voice, int volume){
    ym_write_value(YM_REG_INST_0+channel,voice<<4|
(volume&VOLUME_MASK));
}
void ym_set_key(int channel, NOTES note, int octave,
        int velocity, int on_off, int sustain){
    //calculate f-num lsb and msb
    int f=F_NUMBER[note];
    ym_write_value(YM_REG_F_LSB_0+channel,f&LSB);
    ym_write_value(YM_REG_F_MSB_0+channel,
        sustain<<5|on_off<<4|octave<<1|(f&MSB));
}
//rhythm mode for percussion
void ym_set_rhythm_volume(VOICES voice,int volume){

    ym_write_value(YM_REG_INST_6,volume);
}
void ym_set_rhythm_mode(int on_off){
    ym_write_value(YM_REG_RHYTHM,(on_off&0x1)<<5);
}
```