The following is a rough translation by Craig Van Degrift (craig@yosemitefoothills.com) of an article in Japanese written by Shigeki Hosaka that appeared in July 8, 2015 on http://netbuffalo.doorblog.jp/archives/5064702.html. I believe he is the first person to find the solution of how to make the Raspberry Pi 2 transmit FM sounds. Earlier, articles had appeared around mid-2012 describing how to do this with a Raspberry Pi model B, but the Model B 2 was sufficiently different to stop hacker success until, I believe, the author translated here showed how. - Craig

How to enjoy a mini-FM station with frequency modulation of a RPi – Raspberry Pi 2

It is an irresistible hack for a boy interested in science to make and play with a mini-FM station even though it is a big technical problem. It is a hack that is exciting and can make a FM station using only a Raspberry Pi 2 and a jumper wire.

Sending a FM (frequency modulation) wave using a Raspberry Pi

If I remember correctly, this hack was first made public in "Turning the Raspberry Pi into an FM Transmitter" (http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter). Thoughts about this and similar projects growing like bamboo sprouts after a rain were based on this, PiFmRds (ChristopheJacquet, https://github.com/ChristopheJacquet/PiFmRds) and others. For me, pifm would have been born in about 2012, except that I was without the money for an antenna and so I could not give it a try.

Afterwards, while recalling memories of various blogs and the July, 2014, issue of "Transistor Technology" here in Japan, I began to imagine about "sometime I would try the Raspberry Pi 2 challenge."

I had come across a built-in feature of the SoC (System on Chip) made by Broadcom that, if without errors and with little detail about high frequencies, was an outline for pifm. There existed in the Raspberry Pi digital I/O unit (GPIO) a connector (GPIO4) to the output circuit that was a reference signal for the input phase signal of the Phase-locked loop (PLL) circuit. This PLL had a frequency set by the 500 MHz system clock. With it as the core, it would be possible to produce FM using voice data or similar frequencies as input to the frequency divider and produce stable output frequencies to the GPIO4 pin of the Raspberry Pi!

The signal diagram shows the 500 MHz clock signal being divided by an integer and fed as a reference signal to a phase comparator (PC) that has the fed-back signal from a voltage-controlled oscillator (VCO) as its other input. The output of the phase comparator is run through a low-pass filter and used to control the frequency of the VCO. The output of the VCO is connected to pin GPIO 4.

With that circuit, I thought it was possible to realize my knowledge, curiosity, and heart resources of a boy with my Raspberry Pi 2. Still, regrettably, it was not simply done. The biggest problem was that the circuits that surround the GPIO (This is a peripheral device) are mapped in registers above physical memory. This region changed from 0x20000000 (Raspberry Pi) to 0x3F000000 (Raspberry Pi 2) depending on the SoC.

Actually, other than this there was another problem. Even after the simple `pifm.c` and similar derived code was edited, the change of the peripheral device address from 0x20000000 to 0x3F000000 did not make it work. With that everyone used trial and error efforts, but it was a big problem and they became disheartened.

`pi2fm` Build and Wav File Streaming

Well, even if I think it is not working well, we now have a new starting day. I had become interested in an article "sugee" that I found about the Raspberry Pi 2 on Yochi-san's site (http://jm7muu.com/index.php?RaspberryPi2%2FFM%E9%80%81%E4%BF%A1%E6%A9%9F2, link page created by Yoshihiko.Honda 2015/06/09 must not be the original "sugee" page.) that got me excited. I was not too interested since usually Yochi's page has code that sends and receives data via the standard pipe input method whereas I was interested in movement of a frequency about fixed central value. So, I took a derivation of the simplest code of dpiponi's pifm.c (https://github.com/dpiponi/pifm/blob/master/pifm.c) code as a base and wrote the pifm code I desired by merging the two codes.

It is named pi2fm.c with the contents (code) as follows:

```
// FM Transmitter for Raspberry Pi 2.
// modified from code at https://github.com/dpiponi/pifm by netbufalo.
//
// page numbers in comments refer to
// http://www.raspberrypi.org/wp-content/uploads/2012/02/BCM2835-ARM-Peripherals.pdf
//
// build:
//   $ gcc -lm -std=c99 -g pi2fm.c -o pi2fm
//
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <math.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>
volatile unsigned char *allof7e;
#define BCM2836_PERI_BASE        0x3F000000 // register physical address.
#define GPIO_BASE (BCM2836_PERI_BASE + 0x200000) // GPIO offset (0x200000).
#define CM_GP0CTL (0x7e101070) // p.107
#define GPFSEL0   (0x7E200000) // p.90
#define CM_GP0DIV (0x7e101074) // p.108
#define ACCESS(offset, type) (*(volatile type*)(offset+(int)allof7e-0x7e000000))
void setup_fm(int state) {
    int mem_fd = open("/dev/mem", O_RDWR|O_SYNC);
    if (mem_fd < 0) {
        printf("can't open /dev/mem\n");
        exit(-1);
    }
    allof7e = (unsigned char *)mmap(
                  NULL,
                  0x01000000,  // len
                  PROT_READ|PROT_WRITE,
                  MAP_SHARED,
                  mem_fd,
                  BCM2836_PERI_BASE // base
              );
    if (allof7e == (unsigned char *)-1) {
        exit(-1);
    }
    // set up GPIO 4 to pulse regularly at a given period.
    struct GPFSEL0_T {
        char FSEL0 : 3;
        char FSEL1 : 3;
        char FSEL2 : 3;
        char FSEL3 : 3;
        char FSEL4 : 3;
        char FSEL5 : 3;
        char FSEL6 : 3;
        char FSEL7 : 3;
        char FSEL8 : 3;
        char FSEL9 : 3;
        char RESERVED : 2;
    };
    // note sure why i can't use next line in place of following three.
    // this is a pure C issue, not a hardware issue.
    //ACCESS(GPFSEL0, struct GPFSEL0_T).FSEL4 = 4; // alternative function 0 (see p.92)
    int tmp = ACCESS(GPFSEL0, int);
    tmp = (tmp | (1<<14)) & ~ ((1<<12) | (1<<13));
    ACCESS(GPFSEL0, int) = tmp;
    struct GPCTL {
        char SRC         : 4;
        char ENAB        : 1;
        char KILL        : 1;
        char             : 1;
        char BUSY        : 1;
        char FLIP        : 1;
```

```c
        char MASH        : 2;
        unsigned int     : 13;
        char PASSWD      : 8;
    };
    char clock_src_plld = 6; // p.107
    ACCESS(CM_GP0CTL, struct GPCTL) = (struct GPCTL) {clock_src_plld, state, 0, 0, 0, state, 0x5a };
}
void shutdown_fm() {
    static int shutdown = 0;
    if (!shutdown) {
        shutdown = 1;
        printf("\nShutting down\n");
        setup_fm(0);
        exit(0);
    }
}
void modulate(int period) {
    struct CM_GP0DIV_T {
        unsigned int DIV : 24;
        char PASSWD : 8;
    };
    ACCESS(CM_GP0DIV, struct CM_GP0DIV_T) = (struct CM_GP0DIV_T) { period, 0x5a };
}
// set square wave period. See p. 105 and 108
// although DIV is 24 bit the period can only be set to an accuracy of 12 bits.
// the first 12 bits control the pulse length in units of 1/500MHz.
// the next 12 bits are used to dither the period so it averages at the chosen 24 bit value.
// the resulting quare wave is then filtered using MASH.
// see p.105 and http://en.wikipedia.org/wiki/MASH_(modulator)#Decimation_structures
// the 0x5a is a "password"
void playWav(char *filename, int mod, float bandwidth) {
    int fp = STDIN_FILENO;
    if (filename[0]!='-') fp = open(filename, 'r');
    lseek(fp, 22, SEEK_SET); // Skip 44 bytes wave header.
    int len = 512;
    short *data = (short *)malloc(len);
    printf("now broadcasting: %s ...\n", filename);
    int speed = 270; // you can play faster by decreasing this value.
    unsigned int lfsr = 1;
    int readBytes;
    while (readBytes = read(fp, data, len)) {
        for (int j = 0; j<readBytes/2; j++) {
            // compute modulated carrier period.
            float dval = (float)(data[j])/65536.0 * bandwidth;
            int intval = (int)(floor(dval));
            float frac = dval - (float)intval;
            unsigned int fracval = (unsigned int)(frac*((float)(1<<16))*((float)(1<<16)));
            for (int i=0; i<speed; i++) {
                lfsr = (lfsr >> 1) ^ (-(lfsr & 1u) & 0xD0000001u); // Galois LFSR
                modulate(intval + (fracval>lfsr?1:0) + mod);
            }
        }
    }
}
int main(int argc, char **argv) {
    if (argc>1) {
        signal(SIGTERM, &shutdown_fm);
        signal(SIGINT, &shutdown_fm);
        atexit(&shutdown_fm);
        setup_fm(1);
        float freq_out = argc>2?atof(argv[2]):77.7; // center freq
        float bandwidth = argc>3?atof(argv[3]):8; // a.k.a volume
        int freq_pi = 500; // 500 MHz (RPi core_freq?).
        int mod = (freq_pi/freq_out)*4096; // divisor * PAGE_SIZE
        modulate(mod); // initialize carrier.
        printf("starting...\n -> carrier freq: %3.1f MHz\n -> band width: %3.1f\n", freq_out,
bandwidth);
        playWav(argv[1], mod, bandwidth);
    } else {
         fprintf(stderr,
                "usage: %s wavfile.wav [freq] [A.K.A volume]\n\n"
                "where wavfile is 16 bit 22.050 kHz Mono.\n"
                "set wavfile to '-' to use stdin.\n"
                "band width will default to 16 if not specified. it should only be lowered!", argv[0]);
    }
    return 0;
}
```

Simply put, the FM wave is created and put out using the 500 MHz system clock, a fixed frequency as an argument (default 77.7 MHz [in US a bad idea! Use an empty channel in he US FM band.]), and a Wav data (sound/voice) from a file or from standard input from a file or from standard input. In response, it will produce a modulated (1/R) FM wave.

Thus, let's copy and download (pi2fm.tgz, https://dl.dropboxusercontent.com/u/5966412/rpi/fm_transmitter/pi2fm.tgz) and build pi2fm.c as shown in the following command. I am using the Raspbian system.

```
$ gcc -lm -std=c99 -g pi2fm.c -o pi2fm
```

After the build has finished, attach a jumper wire to GPIO 4 of the Raspberry Pi 2 (although it can work without an antenna).

With the following command, please replay a wave. Surely, from your receiver sound will come flowing out. (The frequency at the end is only an example.)

```
$ sudo ./pi2fm sound.wav 77.7
```

Note: The original article continues showing how to use pipes to send music to `pi2fm` and how to transform `mp3` files into a `wav` format suitable for `pi2fm`. He also discusses the legal limitations placed on private radio broadcasting. There are also a couple of comments about using DMA (direct memory access).