Characterization Of The RFX400 For Use In Software Defined Radio

ΒY

Edward Adams

Senior Project

COMPUTER ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2010

Table of Contents

List of	f Figures ii
List of	f Equations ii
Ackno	owledgementsiii
Abstra	activ
I.	Introduction 1
II.	Background 2
	Software Defined Radio2
	Heterodyne Up-/Down-Conversion2
	Fractional-N Synthesizer3
III.	Requirements5
IV.	Design 6
	SPI Interface to Fractional-N Synthesizer
	Analog Interfacing to Board Transmit6
	Analog Interfacing to Board Receive9
	Voltage Rails for Op-Amps9
V.	Construction 11
VI.	Testing 12
VII.	Conclusions and Recommendations13
VIII.	Bibliography14
IX.	Appendices15
	A. ATMega8 Frequency Synthesizer Code Listing
	B. Nexsys Frequency Synthesizer Code Listing

List of Figures

2
2
4
6
7
8
9

List of Equations

Equation 1: Genera	alized modulated waveform		3
Equation 2: Hetero	dyne down-conversion	3	3

Acknowledgements

I would like to take the time to thank my parents for always encouraging and trusting my curiosities, as well as providing the financial and moral support without which I would not be writing this today. I would also like to thank Dr. Oliver for taking a risk on this project and his understanding when I was unable to dedicate the time that this project deserved. Last, but certainly not least, I would like to offer my gratitude to Dr. Michael Connor: it's been years, but you still inspire me to learn and teach others.

Abstract

While software defined radio is an extremely flexible technology that is fairly rarely used, as a single purpose radio is much cheaper, it provides an excellent means of learning RF and communications skills through the lens of digital electronics, and it would behoove a cal poly student interested in these areas to take on a project in software defined radio. This project aims to lower the barrier to entry for future students to undertake such a project. The use of an up-converter/down-converter is documented for future students to interface with.

I. Introduction

At the time of writing, the predecessors to fourth-generation (4G) cellular networks are beginning to be deployed, and along with this deployment, there is a considerable amount of hype among gadget enthusiasts. It is a significant marketing advantage then to have your phone branded as being 4G compatible.

There are, at the moment, a couple competing candidate standards out there. One candidate standard, 3GPP LTE Advanced¹, has been endorsed by multiple carriers in the US and Europe², and its predecessor, 3GPP LTE, has already been deployed, marketed as 4G. Another candidate, Mobile WiMAX 802.16m³, is under development by IEEE, and its predecessor, 802.16e-2005 will be deployed by Sprint Nextel⁴, also marketed as 4G. Furthermore, UMB, an extension of the EV-DO standard was a candidate until late 2008.

As a handset designer, which standard do you design your cellular radio for? If a standards war breaks out, you don't want to throw your lot in with Betamax. Furthermore, these standards are moving targets — you may release a month too soon to have incorporated an upgrade that doubles range. It would be a great asset to be able to upgrade the radios with a patch, much the way that software designers are able to push new features to a phone without ever having to bring it in for servicing.

Software defined radio does just that, and this project is designed to open the door for future students to gain experience in the medium. By focusing on the 70cm amateur radio band, this project will allow students to have easy access to licensed spectrum with readily available equipment.

¹ Stefan Parkvall, Erik Dahlman, Anders Furuskär, Ylva Jading, Magnus Olsson, Stefan Wänstedt, Kambiz Zangi, "LTE-Advanced – Evolving LTE towards IMT-Advanced", VTC 2008

<http://www.ericsson.com/res/thecompany/docs/journal_conference_papers/wireless_access/VTC08
F_jading.pdf>

² http://en.wikipedia.org/wiki/4G#LTE

³ Draft IEEE 802.16m System Description Document, 2008-04-20

<http://www.ieee802.org/16/tgm/docs/80216m-08_003r1.pdf>

⁴ "Sprint announces seven new WiMAX markets, says 'Let AT&T and Verizon yak about maps and 3G coverage'". http://www.engadget.com/2010/03/23/sprint-announces-seven-new-wimax-markets-says-let-atandt-and-ver/> Engadget. 2010-03-23. Retrieved 2010-04-08.

II. Background

Software Defined Radio

Software defined radio (SDR), in its theoretically ideal form consists of a digital to analog converter (DAC) and/or an analog to digital converter (ADC) hooked up directly to antennas as in Figure 1. The software controlled DAC and ADC would in this case directly generate the waveforms to be transmitted and directly read the waveforms received.



Figure 1: Idealized SDR

This design turns out to be impractical due

to, among other things, limitations in ADC and DAC technology. These devices cannot operate fast enough and with enough precision to operate in the UHF band we are interested in. Instead, for this project we use the RFX400 daughterboard for the Universal Software Radio Peripheral (USRP), part of the GNU Radio Project.



Heterodyne Up-/Down-Conversion

The RFX400 has multiple DACs and ADCs operating at baseband and mixes the signals with a sinusoid at a center frequency. This is known as heterodyne up-conversion and down-conversion, and is shown in simplified form in Figure 2. The generalized form of a modulated RF waveform is shown in

Equation 1. Note that both amplitude and

phase information can be encoded in the amplitude of in-phase and

quadrature waveforms. Heterodyne converters work by adding and removing the sine/cosine terms attached to the baseband signals I(t) and Q(t).

$$S(t) = A(t)\cos(\omega_0 t + \varphi(t)) = I(t)\cos(\omega_0 t) + Q(t)\sin(\omega_0 t)$$

Equation 1: Generalized modulated waveform

The up-converter on the transmit path is fairly straightforward. After passing through low-pass filters to remove aliasing from the DACs, the I and Q components are mixed with the cosine and sin components and then summed. Removing the cosine and sin components turns out to be less intuitive. The input is mixed with the cosine and sine components, and due to the double angle identities, the baseband signals appear summed with signals centered at twice the original center frequency. See Equation 2. These signals are sent through a low-pass filter to remove the double-frequency components and then through to the ADC.

In-phase	Quadrature	
$\cos(\omega_0 t) \times S(t)$	$\sin(\omega_0 t) \times S(t)$	
$\cos(\omega_0 t) \times (I(t)\cos(\omega_0 t) + Q(t)\sin(\omega_0 t))$	$\sin(\omega_0 t) \times (I(t)\cos(\omega_0 t) + Q(t)\sin(\omega_0 t))$	
$I(t)\cos^2(\omega_0 t) + Q(t)\cos(\omega_0 t)\sin(\omega_0 t)$	$I(t)\cos(\omega_0 t)\sin(\omega_0 t) + Q(t)\sin^2(\omega_0 t)$	
$\frac{1}{2}(I(t)(1 + \cos(2\omega_0 t)) + Q(t)\sin(2\omega_0 t))$	$\frac{1}{2}(I(t)\sin(2\omega_0 t) + Q(t)(1-\cos(2\omega_0 t)))$	
$\frac{1}{2}I(t) + \frac{1}{2}(I(t)\cos(2\omega_0 t) + Q(t)\sin(2\omega_0 t))$	$\frac{1}{2}Q(t) + \frac{1}{2}(I(t)\sin(2\omega_0 t) - Q(t)\cos(2\omega_0 t))$	

Equation 2: Heterodyne down-conversion

Fractional-N Synthesizer

The oscillator used to generate the mixed-in sine and cosine is a fractional-N synthesizer: a device that counts zero crossings of a reference clock and a voltage controlled oscillator (VCO) to maintain a programmable constant ratio between the VCO and the reference oscillator. See Figure 3.



Figure 3: Simplified fractional-N synthesizer

On startup, the N- and R-latches are programmed with the desired clock ratio and their respective counters begin to count up. As soon as a counter reaches its latch value, it resets. the first counter to reset sets its flip flop, turning on its current source in the charge pump. Once the second counter resets, its flip flop is momentarily set, causing both flip flops to reset, turning the charge pump off.

The charge pump is fed back to a shunt capacitor and the control voltage of the VCO such that the charge pump provides negative feedback on the VCO, pulling its frequency down should the N-counter overflow first, and pulling its frequency up should the R-counter overflow first.

III. Requirements

The purpose of this project is to open the way for other students to come in and make their contribution to a larger SDR project. As such the primary goal is to create quality, easily accessible documentation for students to work from.

In order to understand the interface I am to perform a loopback test: the transmit port will be connected to the receive port, the synthesizer will be programmed, and voltage will be applied the transmit inputs. If all goes well, the input signal will show up on the receive outputs.

To summarize, the requirements for this project are:

- 1. Investigate RFX400 daughterboard interface.
- 2. Perform loopback test.
- 3. Provide documentation for future projects wishing to use this board.

IV. Design

SPI Interface to Fractional-N Synthesizer

The synthesizer used on the RFX400 is controlled through three 24-bit registers accessible through a three-wire SPI bus. These registers are the N-counter register, the R-counter register, and the control register. These registers must be programmed in order on startup, and they have timing constraints.

Programming the synthesizer was originally accomplished through a simple HDL design the Nexsys board. Events were sequenced through an up counter and the register contents were held in a 72-bit shift register. Making changes to the contents of the synthesizer registers was an arduous process of counting bits, recompiling, and reprogramming the board.

At the start of spring quarter I came to possess an STK500, a development board for Atmel's AVR line of microcontrollers. I immediately used an ATMega8 Microcontroller to implement a synthesizer programmer that was much easier to use. I highly recommend using some form of processor (e.g. AVR, MicroBlaze, PicoBlaze, etc.) for synthesizer control to anyone looking to continue this project.

Analog Interfacing to Board Transmit

The inputs to the transmission modulator are differential lines balanced about .7V with a voltage swing of $0.6V_{PP}$. Since this is to be used on a testbench, we will be using a single-ended source. There are a number of ways to convert a single-ended source to a differential output with nonzero bias.

The simplest method is to use a 1:1 transformer, hooking the input up to a $0.6V_{PP}$ source connecting the center tap of the output to .7V, as in Figure 4. The advantage here is simplicity: a single component produces a differential output and biases the lines appropriately. The drawback here is that a transformer can only AC couple the input and output: it prevents holding a signal at any one level.







Figure 5: Fully differential op-amp circuit for RFX400 transmit circuit

Another method of producing the proper output is to set a function generator to directly output the positive side of the differential line and use an op amp to invert that about V_{bias} for the negative side. This introduces more complexity, as it may require a negative power rail and uses a number of components, but the extra complexity is balanced by the ability to DC couple the line. Another drawback is that the op-amp may introduce some phase shifting on one line that isn't present on the other, distorting the output, but that would be passable for qualitative test purposes.

Fully differential op-amps are designed to drive differential loads and provide a means to set their output bias, so they seem like an optimal means of driving our output⁵. Figure 5 is an example circuit for providing differential loads to the RFX400. I and Q should have a voltage swing of

⁵ Fully Differential Amplifiers

<http://www.ti.com/sc/docs/apps/msp/journal/aug2000/aug_08.pdf>

0.6 $V_{\mbox{\tiny PP}}$ and I_GND and Q_GND should be set to 1/2 the dynamic range of I and Q respectively.

Differential op-amps are, however, an uncommon part, and not available as a DIP package. This being the case, I was unable to prototype the circuit on a breadboard. I provide a schematic because it will be useful to any future students wishing to go further with this project.

The design I chose to implement for generating the output signal is one that is tenable for a final design. Rather than using a fully differential amplifier, two single-ended op-amps are configured as differential follower circuits with V_{bias} replacing the ground input. I used some LM324 op-amps I had lying around, as bandwidth is not a problem at this point. The outputs of these are fed into voltage followers for hi-impedance output. This circuit provides DC coupling, and due to both sides having nearly identical circuitry, any phase delay is applied equally, ensuring that the output differential voltage is correct. Figure 6 shows the circuit as implemented.



Figure 6: Single-ended op-amp circuit for RFX400 transmit circuit

Analog Interfacing to Board Receive

Output from board receive is another pair of differential lines. Since our output should be single-ended, it isn't necessary to use a fully differential

op-amp. Instead, two differential follower op-amp circuits are used to convert the I and Q differential outputs to groundreferenced, singleended outputs. The outputs of the RFX400 requires at least $2k\Omega$ line resistance, so resistor values of 1.5k Ω were used to provide approximately $3k\Omega$ line resistance. Again, bandwidth is not a concern at



Figure 7: Differential follower circuit for RFX400 receiver circuit

this point, so an LM324 was used to implement this circuit. See Figure 7.

The receive side of the RFX400 includes a gain setting — a pin whose voltage can vary from 0.2V for maximum gain to 1.2V for minimum gain. Current draw from this pin should be negligible, so a voltage divider may be used to set the voltage on this pin. Using a $1k\Omega$ potentiometer for a 1V swing gives a current of 1mA through the voltage divider. 200Ω to ground gives us a 0.2V offset and $4.8K\Omega$ to the +6V rail creates a total of $6k\Omega$ from 6V to ground and gives us our 1mA voltage divider current. Having only 5% tolerance resistors, 220Ω and $4.7k\Omega$ resistors were used instead.

Voltage Rails for Op-Amps

Since non-ideal (i.e. real) op-amps cannot output rail-to-rail, the voltage supplies for the op-amps used in these circuits will need to reach beyond the 0V and +6V rails used elsewhere in the circuit. With a 12V input, the 0V and +6V rails can be generated by offsetting by 3V each, turning the negative and positive input rails into -3V and +9V respectively.

This was accomplished by first generating a 3V reference with respect to the negative rail using a 3.3V regulator, a voltage divider, and a voltage follower. This is fed into a voltage follower with a PNP bipolar transistor for greater current capacity. The output of this is our 0V rail. The 3V reference is also fed into a differential follower which subtracts the 3V from the 12V input to give the reference for our +6V rail. The +6V reference is fed into a voltage follower with a NPN bipolar transistor for greater current capacity, giving us our +6V rail.

V. Construction

The circuits described in the previous section were constructed on a breadboard, power was routed from the voltage regulation circuit to the biasing circuitry and the SPI lines from the synthesizer programmer were brought onboard.

The RFX400 daughterboard connects with its motherboard via a pair of 64-pin PCI mezzanine connectors (PMC). In order to connect the RFX400 to the test setup, I bought mating PMC risers and soldered wire-wrap wire to the pins, providing power, ground, logic, and analog breakouts. The SPI lines, Tx and Rx differential pairs, and gain pin were soldered to breadboard patch wire and connected to their appropriate inputs. the remaining logic pins were pulled high or low through resistors to power and ground. Power and ground were attached to the +6V and 0V rails respectively.

VI. Testing

When first tested, the RFX400's frequency synthesizer would not lock. Using the "muxout" pin of the synthesizer, I scoped the N-counter and Rcounter overflow signals and found that the N-counter was overflowing at a much greater frequency than the R-counter. Misinterpreting these results, I attempted to cause the synthesizer to lock at lower frequencies. This did not work, it turned out that the VCO in the synthesizer was at as low a frequency as it could manage.

The diagram in Figure 3 is greatly simplified and does not show all the features available to be controlled through the SPI interface. One of these features is a pair of optional devide-by-2 stages on the output of the VCO, one leading to the output, the other leading to the R-counter. The documentation for the RFX400 states that it uses one of these, and seemed to imply it was the one leading to the R-counter. It wasn't.

Eventually I got through to the designer of the board, and he pointed me to the code that he had written to program the frequency synthesizer and let me know that it was the output divide-by-2 stage that was enabled; I would have to lock the synthesizer at twice my desired frequency.

Once I implemented the changes that resulted from talking with the board's designer, I got the board to lock, but did not have enough time to get any further.

VII. Conclusions and Recommendations

At the time of this draft, the loopback test has not been successfully completed, but the greatest hurdle, controlling the frequency synthesizer, has been overcome. What remains is testing of the analog biasing circuitry I have proposed, designing the ADC/DAC circuitry, and interfacing all the digital outputs with the Nexsys board through the Hirose FX2 connector.

Despite my frustrations, I believe that the RFX400 is an excellent route to a custom SDR project at Cal Poly. This board effectively blackboxes the portions of a radio that are the most difficult for digital engineers that are interested in radio applications such as myself.

Meanwhile, the work on the project remains ongoing and I would be more than willing to be of assistance to anybody who wishes to pick up where I leave off. That's right, you there, reading the microfiche, can contact me (If I'm still alive at the time!) at edward.casey.adams@gmail.com and I will be glad to help.

VIII. Bibliography

- Parkvall, S.; Dahlman, E.; Furuskar, A.; Jading, Y.; Olsson, M.; Wanstedt, S.; Zangi, K.; , "LTE-Advanced - Evolving LTE towards IMT-Advanced," *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th* , vol., no., pp.1-5, 21-24 Sept. 2008 doi: 10.1109/VETECF.2008.313 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4 657145&isnumber=4656832
- June, L.; , "Sprint Announces Seven New WiMAX Markets, Says 'Let AT&T and Verizon Yak about Maps and 3G Coverage'" *Engadget*, 23 Mar. 2010, Web, 11 June 2010 URL: http://www.engadget.com/2010/03/23/sprint-announcesseven-new-wimax-markets-says-let-atandt-and-ver/

Karki, J; , "Fully Differential Amplifiers." Analog Applications Journal (2000): 38-41, High-Performance Analog - Analog Application Journal - Texas Instruments, Texas Instruments, Aug. 2000, Web, 11 June 2010 URL: http://focus.ti.com/general/docs/gencontent.tsp?contentId=2 9569

IX. Appendices

A. ATMega8 Frequency Synthesizer Code Listing

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
char R reg[3] = { 0x34, 0x0A, 0x02 };
char C_{reg[3]} = \{ 0x0F, 0xC9, 0x84 \};
char N reg[3] = { 0x4F, 0xAE, 0x02 };
void init(void)
{
    DDRD = 0 \times FC;
    PORTD = 0x03;
    DDRB = 0xFF;
    SPCR = BV(SPE) | BV(MSTR);
    SPSR = BV(SPI2X);
   PORTB = BV(PB1);
}
int main(void)
{
    uint8 t i;
    //volatile uint16 t j;
    init();
    while(1)
    {
        while ((PIND \& 0 \times 01) == 0)
        {
        ;
        }
        while ((PIND&0x01) != 0)
        {
         ;
        }
        PORTB = PINB & ~ BV(PB1);
        for (i = 0; i < \overline{3}; i++)
        {
            SPDR = R reg[i];
            while(!(SPSR & _BV(SPIF)))
            {
                 ;
```

```
}
    }
    PORTB = PINB | BV(PB1);
    PORTB = PINB & ~ BV(PB1);
    for (i = 0; i < \overline{3}; i++)
    {
        SPDR = C reg[i];
        while(!(SPSR & BV(SPIF)))
        {
             ;
         }
    }
    PORTB = PINB | _BV(PB1);
    while ((PIND&0x02) == 0)
    {
     ;
    }
    while ((PIND \& 0 \times 02) != 0)
    {
       ;
    }
    //wait >10ms
    //for (j = 0; j < 400000; j++)
    //{
    // ;
    //}
    PORTB = PINB & ~ BV(PB1);
    for (i = 0; i < \overline{3}; i++)
    {
        SPDR = N reg[i];
        while(!(SPSR & _BV(SPIF)))
         {
            ;
         }
    }
    PORTB = PINB | BV(PB1);
}
return 0;
```

}

B. Nexsys Frequency Synthesizer Code Listing

```
`timescale 1ns / 1ps
module vco(clk, sck, sda, sen);
parameter word len = 24;
parameter R latch =
24'b00 01 0 1 00 00 0000 1010 0000 01;
parameter C latch =
24'b01 00 000 000 00 0 0 0 1 001 0 00 00;
parameter N latch =
24'b1 0 0 0 1111 1010 1110 0 0 0000 10;
parameter data x = {R latch, C latch, N latch};
input clk;
output sck;
output sda;
output sen;
reg
     sck;
wire sda;
reg sen;
     [31:0] counter;
req
reg
      [3*word len-1:0] data;
assign sda = data[3*word len-1];
initial
begin
     data = data x;
     counter = 0;
     sck = 0;
     sen = 0;
end
always @(posedge clk)
begin
     if (counter < 32'hFFFFFFF)
     begin
          counter = counter + 1;
     end
     if (sck)
     begin
           sck = 1'b0;
           data = data << 1;</pre>
```

```
end
     else if ( (counter == 49) || (counter == 99) || (counter
== 1000149) )
     begin
          sen = 1'b1;
     end
     else if (sen)
     begin
          sen = 1'b0;
     end
     else if (counter < 99)
     begin
         sck = 1'b1;
     end
     else if ( (counter >= 1000100) && (counter < 1000148) )
      sck = 1'b1;
```

```
end
```

endmodule