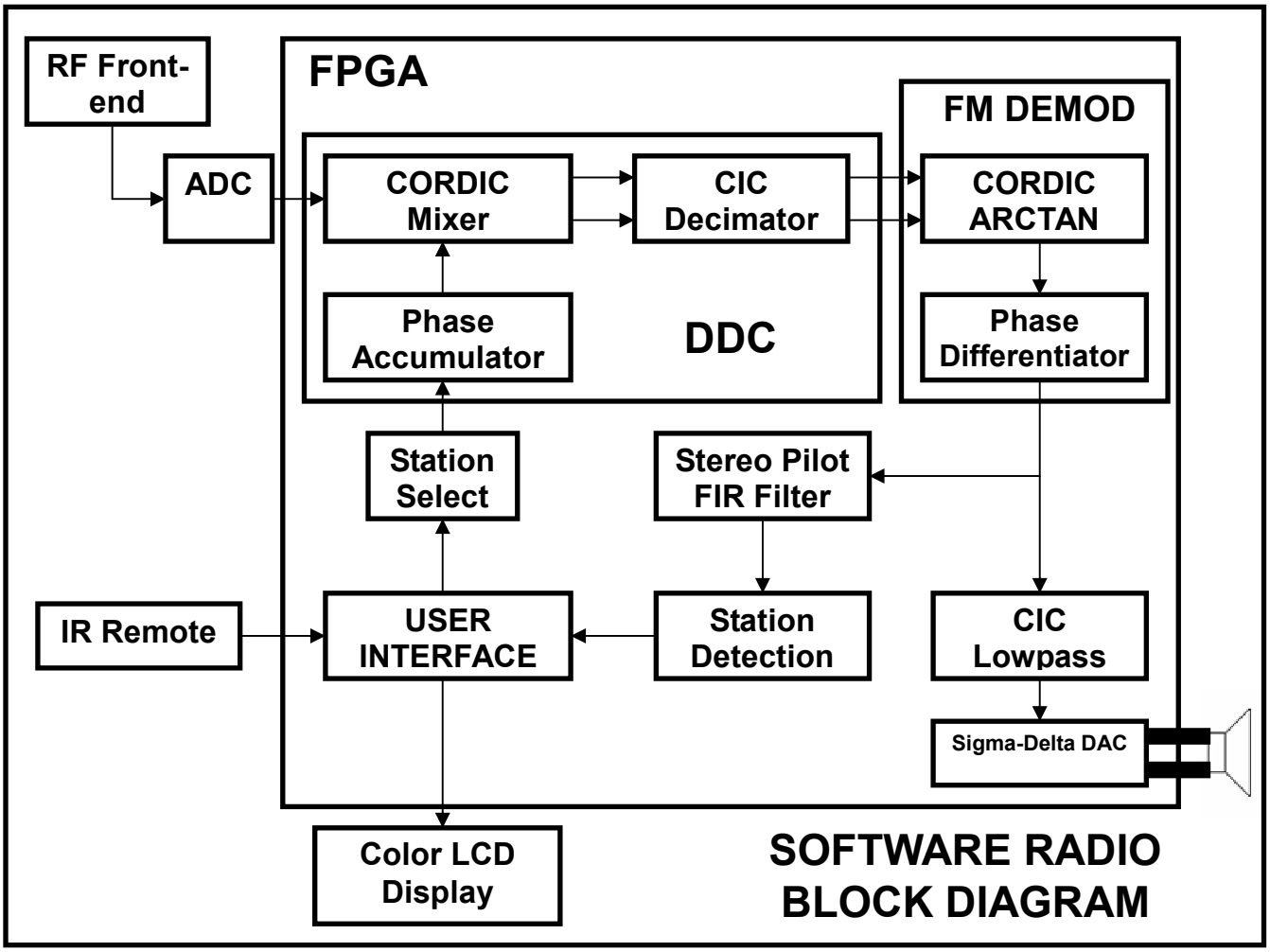
Software Defined Radio FPGA Project

Asif Khan
Nnoduka Eruchalu

(Last Revision: June 25, 2010)

TABLE OF CONTENTS:

SYSTEM BLOCK DIAGRAM



Associated VHDL Source Files:

FPGA Top Block	-	**TOPLEVEL.VHD**
CORDIC	-	**CORDIC.VHD**
CIC	-	**CIC.VHD**
Phase Accumulator	-	**PHASE_ACC.VHD**
Digital Down Converter	-	**DDC_MIXER.VHD**
FM Demodulation	-	**FMDEMOD.VHD**
FIR Filter	-	**FIR.VHD**
Station Detection	-	**STATION_DETECT.VHD**
Sigma-Delta DAC	-	**DAC.VHD**
IR Remote	-	**IR_DECODE.VHD**
Color LCD	-	**LCDCNTRL.PSM**
User Interface	-	**UI.VHD**

Functional Specification

Description:
The SDR project is an FPGA-based system which is capable of receiving FM radio signals and demodulating them. The user will be able to select a station to listen to using an LCD and IR remote controller interface. The radio can also seek and jump to the next available radio station above or below the current station's frequency.

Inputs:
Below is a table with the input devices and their descriptions

Input Device	Description
IR Remote Control	This is a multiple-button remote controller used for station select (up/down), station entering (via the number pad), volume mute and station seeking.

Outputs:
Below is a table with the output devices and their descriptions

Output Device	Description
Display	This is a Color TFT LCD. It will display the currently playing station frequency.

User Interface:
The user is able to input commands via the IR Remote Control and is provided visual feedback via the LCD. Audible feedback (button press tones, radio broadcasts, etc.) is provided via the speaker output.

Error Handling:
None necessary.

Algorithms:
NCO based mixing, CIC filtering, FIR bandpass filtering, FM Quadrature Demodulation, Sigma-Delta DAC, time multiplexing FIR filter, CORDIC, Self-designed Heuristic Station Detection Algorithm

Data Structures:
None.

Limitations:
Unlimited potential

Known Bugs:
Not Applicable.

SYSTEM OVERVIEW

Introduction

This project is a Software Defined Radio implemented in an FPGA. The FPGA of choice is a Spartan 3E with 1200K gates. The external peripherals are discussed in the "User Manual" section and they are the IR remote controller and the TFT LCD. This section of the documentation discusses the internals of the FPGA. It describes the separate blocks that can be found in the block diagram and are implemented in toplevel.vhd, the vhdl file which puts the project's code blocks together.

The major blocks of the system are:

- RF Front End
- The ADC
- The DDC block (the NCO-based mixer)
- The CIC lowpass filter and Decimator
- The FM demodulator
- The CIC lowpass filter and Decimator (pre-DAC)
- The sigma delta DAC
- FIR bandpass filter
- Station Detector
- The User Interface

RF Front End

The RF Front End should have a bandpass filter that passes only the FM range (87.5MHz to 108MHz) As it turns out we do not need this bandpass filter. So really all that our RF front end consists of is the FM antenna.

The ADC

The actual ADC block is a 12-bit AC-coupled ADC board with a peak-to-peak input voltage of 2Vpp. This ADC board is the DSPBrik DA4012. The ADC is clocked at 43.3333MHz and this is explained below.

The FM band is 87.5MHz to 108MHz. By the sampling theorem, we will require an ADC that can be clocked at least 216MHz, and even though we could get an ADC that can handle such a high frequency, we would still need to find an FPGA that can handle such clock frequencies. This is rather impractical so we decided to use the undersampling theorem to our advantage.
With the undersampling theorem we can sample the FM band signals with a sampling frequency Fs in the range 43.2MHz < Fs < 44MHz. This frequency range can definitely be achieved in the FPGA. Using one of the FPGA's DCM blocks we were able to achieve a sampling frequency of 130/3 MHz (43 and a third). The FPGA already has an input clock of 50MHz so all that was needed was a multiplication by 13/15 to achieve 130/3MHz.

So with this frequency, the FPGA and the ADC board could be clocked at the same frequency.

It turned out to be rather difficult to clock the ADC board from the FPGA board because of the MCX connector for the external clock input on the ADC board. So we decided to use channel 4 of the ADC board, set it in independent mode and clock it externally with a high frequency (RF) function generator set at 43.3333MHz. Note that we could not set it to exactly 130/3MHz so the ADC board and the FPGA boards are now running at slightly different frequencies.

The ADC also comes with clock output lines for each ADC channel. These clock output lines are LVDS outputs which are at the same frequency as the input signal from the function generator. The ADC's data outputs are valid on the falling edge of these clock out lines. So these clock out lines were sent into the FPGA. These LVDS clock lines from the FPGA are converted to a single clock signal in the FPGA and the FPGA latches the ADC data on the falling edge of this clock line derived from the two LVDS lines clock lines the ADC.

When the ADC data had been latched, we could go on to implement the ever-recommended synchronizing with the FPGA system clock. This just meant passing the latched ADC data through two DFFs. With this latched and synchronized ADC data, we can now do the real processing that defines the software aspect of this software defined radio.

The DDC block (the NCO-based mixer)

With our use of the sampling theorem, the FM band of 87.5 to 108MHz has been shifted down to start from 0.8334MHz. Now what gets tricky is working with the users station choice. As the user changes radio station the frequency of that station of course changes as well and the entire system would need to account for this. To make the FM signal processing easier we decided to make the later stages independent of the chosen station by having them assume that the current station is already at the base of 0 frequency. To do this there has to be a preliminary block which detects the station of choice and down converts it to the base of 0 frequency. This is the Digital Down Conversion (DDC) Block. The DDC block is in theory comprised of a mixer, a low-pass filter and a decimator. For now we will discuss the mixers. The NCO-based mixers multiply the incoming signal by cos(2*pi*Fc*t) and sin(2*pi*Fc*t) where Fc is the offset of the station of choice from the base frequency of 87.5MHz plus the 0.8334MHz offset (coming from the sampling). These multiplications come easy with the use of a CORDIC as we have done. So for example, if the user wants a station at 97.1MHz, then
Fc = (97.1-87.5) + 0.8334 = 1.43334MHz

Mixing by multiplying with cosines and sines at frequency Fc results in the station of choice appearing at frequencies of 0 and 2*Fc (which is why we need the low pass filter, to get rid of this high frequency component).

These multiplications with cosine and sine generate the Real/In-phase (I) and Imaginary/Quadrature (Q) components that will be used later in the FM signal processing.

CIC Lowpass filter and Decimator

As mentioned when discussing the mixers, there is a high frequency component generated after mixing and this needs to be removed. This will be done by using a lowpass filter. Also at this stage it will be ideal to decimate the data. Decimating the data means the next few stages will get data at a lower rate

and thus we will need fewer resources as we can do more resource sharing/ reuse. We decided to avoid MULTIPLIERS in this project so decided on using a Cascaded Integrator-Comb (CIC) filter at this stage. This uses only adds and subtracts to lowpass filter and decimate data simultaneously.

The basic building blocks of a CIC filter are an integrator and a comb.

An integrator is a single-pole IIR filter with a unity feedback coefficient as seen in its formula:

$y[n] = y[n-1] + x[n]$.

A comb filter running at the high sampling system clock of 43.3333MHz with a rate change R is an odd-symetric FIR filter described by

$y[n] = x[n] – x[n-RM]$

where M is called the differential delay, a positive integer usually limited to 1 or 2.

The CIC filter then uses these blocks and throws in down-sampler to implement the rate change R.

The input signal is fed through a series of Integrators, then a down-sampler, followed by a a series of comb sections. NOTE that the number of comb sections has to be equal to the number of integrator sections.

The CIC filter is known to have a linear phase response and most importantly has NO MULTIPLIERS. The transfer function of the CIC filter is:

$$H(z) = [\sum_{k=0}^{RM-1} z^{-k}]^N$$

where R = the decimation rate
M = the number of samples per stage (we fixed this at 1)
N = the number of stages in the filter (both the comb and integrator sections have the same number of stages!)

The FM Demodulator

With the CIC filters described in the last section are used to lowpass and decimate the I and Q outputs of the NCO mixers stage. The lowpassed and decimated I and Q signals can now be FM Demodulated in this block.

At this stage the message coming into the Quadrature demodulator block is proportional to the instantaneous frequency of the I and Q signals. So the instantaneous frequency has to be estimated somehow. Ideally this is done by differentiating the phase of the signals at every instant. This phase differentiation will be estimated by determining the angle between adjacent samples. To do this the CORDIC will be brought back to use. The CORDIC can do ArcTan calculations and will be used to determine the phase of each sample based on the Is and Qs. The trick lies in detecting the quadrant of the sample by looking at the signs of its I and Q values, then putting this sample in a quadrant that the CORDIC can work with. Of course the output of the CORDIC's ArcTan calculation will need to be placed in the right quadrant depending on the original I and Q values.

To find the angle between adjacent samples we save the result of the last calculation in a register so as to be able to subtract it from the result of the current calculation.

Of course the FM demodulator needs to account for the fact that it receives decimated data which is done by only latching new inputs on the rising edge of the decimated clock, which is really just a decimated clock enable signal. The quadrature demodulated data is now ready for output via a DAC.

The CIC lowpass filter and Decimator (pre-DAC)

In the last section it was said that the demodulated data was ready for output via a DAC. Well that is not quite accurate because it turns out that there are still some high frequency components in the demodulated data. This needs to be lowpass filtered for the best quality sound output. An FIR lowpass filter would have been best for this, but the FPGA has more than enough space for a CIC filter with a sharp roll-off. At the same time this decimates the sound output before sending it to the DAC. This is helpful for the DAC which comes with an analog low pass filter of 20KHz. So it wouldn't make too much sense to send in data at a rate much higher than 40KHz.

The Sigma Delta DAC

Now that the FM signal has been demodulated and low pass filtered it needs to be converted to audio. This means that we will need a Digital to Analog Converter (DAC) block. This is going to be implemented in the FPGA and it will be a Sigma Delta DAC.

The setup of this Sigma Delta DAC follows the standard setup. The demodulated digital signal input goes into an integrator which has a transfer function of 1/(z-1). The output of the integerator goes through a 1-bit quantizer which is just a (signed) comparator. In the signed comparator if the input digital number >= 0, the output = 1 else the output = 0. The output of this quantizer is then sent to the 1-Bit DAC which is just a 1-bit DFF with an inverter (to account for the inverting op amp config of the active LPF on the analog side). The output of this quantizer is also sent through a feedback loop with a signed 1-Bit DDC. The signed 1-Bit DDC gives out an n-bit number and turns 0 -> 10..00 and 1 -> 01..11. The output of the 1-Bit DDC is summed with the DAC's digital input before that goes into the integrator.

It was hinted at earlier that there is an analog section following this DAC.

The output of the 1-Bit DAC goes out of the FPGA to an active low pass filter. The lowpass filter is designed for the audible range (<20KHz) and it is an active lowpass filter so as to amplify the signal as appropriate. The output of the 1-Bit DAC is really a signed signal so we have to level shift the signal by 1.66V so as to be able to use only the voltages which drive the FPGA, 0 and 3.3V.

FIR bandpass filter

A big feature of this software defined radio is the station seeking, and for this to happen a station detection block is essential. This will be discussed next, but the important input of the station detect

block is a “pilot” signal and that is what this bandpass filter generates. Radio stations which output stereo sound have a 19KHz pilot signal. It turns out that all the FM radio stations we could detect were outputting stereo signals so they all had this 19KHz pilot signal. Thus we decided that we would extract this 19KHz pilot signal as a means of detecting stations not just stereo. This 19KHz bandpass filter uses just 1 of the inbuilt hardware multipliers on the Xilinx FPGA. This is a 104-tap filter which works fine because the actual decimation rate on the first CIC filter is 110, so 104 multiplies can fit comfortably in the lag time of 110 clocks.

The input to this bandpass filter is the demodulated output and the output of this bandpass filter is a 19KHz signal for stations with stereo and static otherwise. This then goes into the station detect block.

The Station Detector

This block detects the presence of an FM radio station when given the 19 kHz pilot tone (BPF 15-23 KHZ) for the demodulated FM signal. This is done using a heuristic, determined via trial and error, that seems to work very well in practice.

The mechanism used to detect a station is by looking at the amplitude of the band pass filtered pilot tone. Essentially the entity looks at the average of the signal and determines there is a station if this average is BELOW a certain threshold. This works because when band-pass filtering 15-23 kHz on a station, the only signal present is at 19 kHz. When this is done on a non-station, there will be noise in this band, which increases the average amplitude detected.

The exact algorithm used is as follows:

Sample 32 positive-sloped zero crossings at 19 kHz. For each period of the 19 kHz wave, the maximum amplitude is recorded. These amplitudes are averaged over the 32 samples. If this average is greater than a certain threshold, we assume it is likely that a station exists at this frequency. We repeat the entire process for 32 trials and determine a station is present if we get a positive indicator for a certain hard-coded number of trials.

With this the station detector can interface with the UI and tell it if the current frequency is a radio station. This block is essential to the seeking feature of the User Interface.

The User Interface

This is discussed thoroughly in the “User Manual” Section.

CHALLENGES FACED

Some of the challenges faced in this project were:

Getting our ADC to work.

We originally bought an NXP ADC, and after a week of wrestling with it we decided to use the provided DSPBrik ADC. We ended up getting sporadic data from our own chip, but it seemed there was some issue with the timing that prevented the data from coming in at a reliable rate. Even if we were to have gotten this to work, we would have had to level shift the signal to the common mode ADC output level of the chip, and it seemed we were having issues with the high-speed op-amps that we had purchased.

Only detecting every other station

It turns out that the FM stations in the Pasadena area are actually spaced apart by 0.4MHz being as there are not so many of them. Our SDR only picked up every other station, so stations were spaced apart by 0.8MHz on our system. Also for the strong stations we could hear them bleeding over to where the neighboring stations should have been. This was another feature we did not have time to look into carefully. We suspect that this has something to do with the CIC filter not having a sharp enough cutoff, so that strong neighboring stations are able to overpower weak stations, even if we are not at the strong station's base frequency. Due to time constraints, we were unable to test adding in a FIR-based CIC-compensation filter which would have improved the cut-off at 100 kHz.

A buzzing sound over our output

While our signal quality was decent we kept on hearing a buzzing sound over our output. Again this was another problem that the limited time did not let us attend to. However, since this did seem like a low frequency buzz, and the CIC filter exhibits passband droop, it is possible that the uncompensated CIC filter was at least partly responsible for the overly emphasized low frequencies of our audio output. This too could perhaps been improved with an FIR compensation block, but was not implemented due to time constraints. Another theory is that perhaps high frequency components of our demodulated signal get aliased to lower frequencies because we don't low pass filter our demodulated output with anything more than another CIC, and the DAC samples at a low frequency.

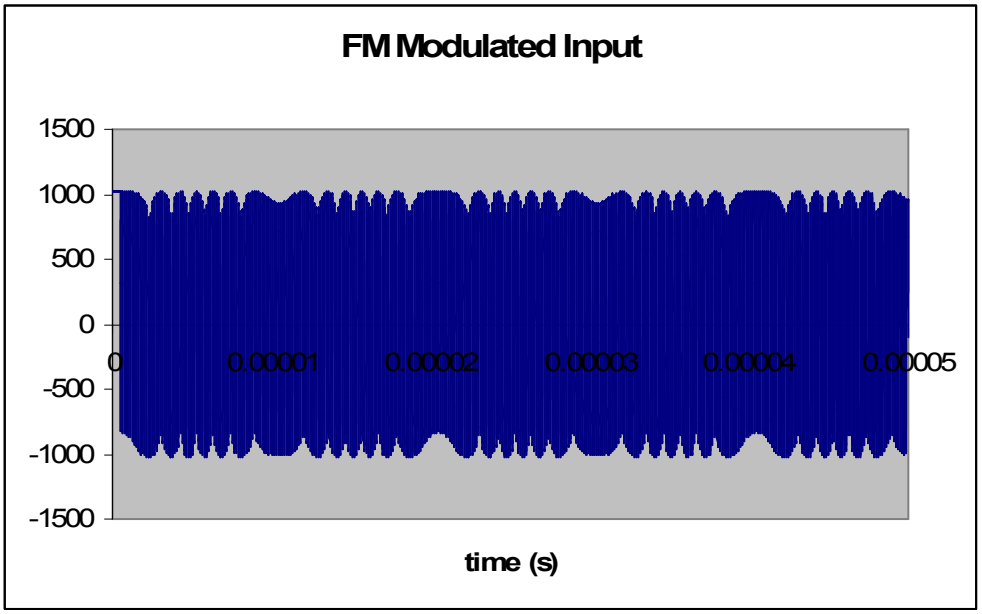
Seek function skipped some valid stations

While trying to seek, the SDR would skip over some perfectly valid stations and the skips/jumps were unpredictable, they went could be anything from <1MHz to over 5MHz skips, This is definitely a bug that could have been fixed had we had a couple more hours to spare. But with demo time closing in we had to round up the project. However, it is important to note that our station detection module is able to detect stations perfectly, so this bug is not an issue of detection, but rather an issue of miscommunication between the UI code and the station detection code.

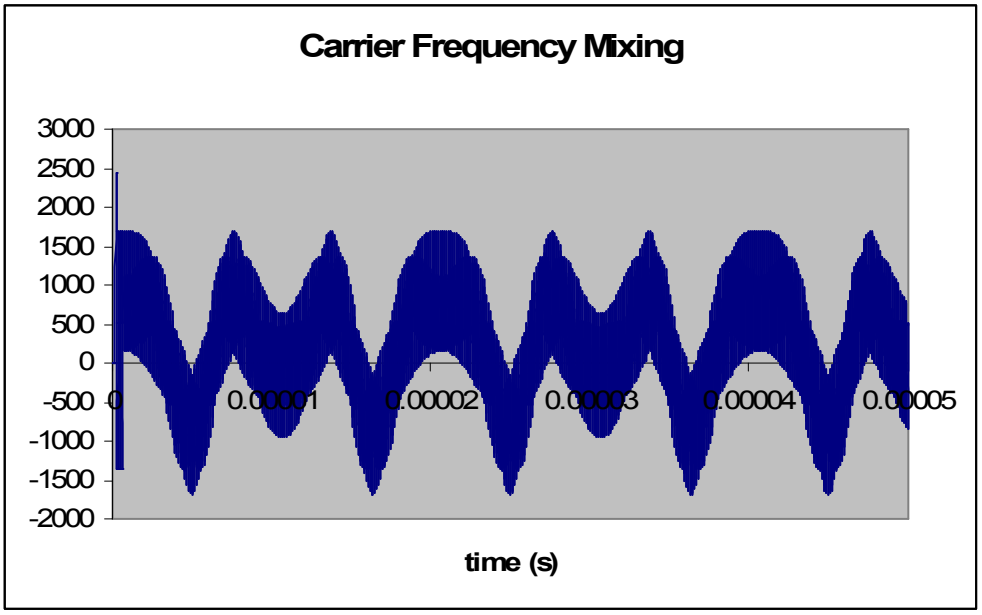
SYSTEM SIMULATIONS

As part of the testing phase of the project, a 10 kHz sine wave was modulated over a 10 Mhz carrier in MATLAB, creating an input dataset for our system. We used this waveform to test the outputs of various stages of our radio. The graphs from testing are provided here for informational purposes. Before this testing occurred, many of the individual components were separately tested using their own test benches. Also note that there may be an anomaly at around 10 us – for test benching purposes, we had to execute a reset at this point in order for the simulations to work.

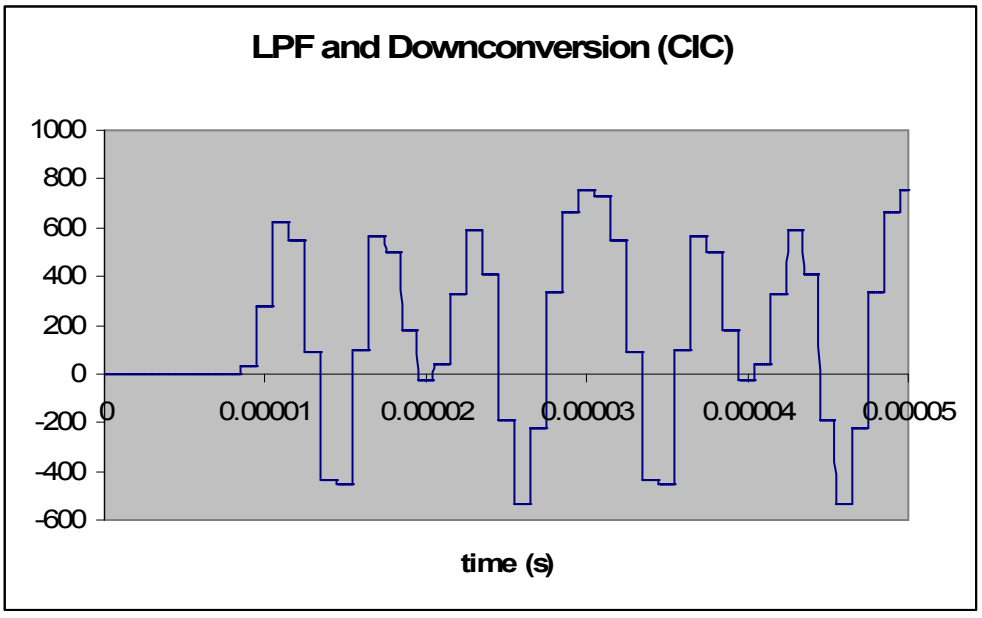
First is a graph of the input to the system: A 10 kHz sine wave with a 200 kHz modulation factor, modulated over a 10 Mhz carrier frequency.



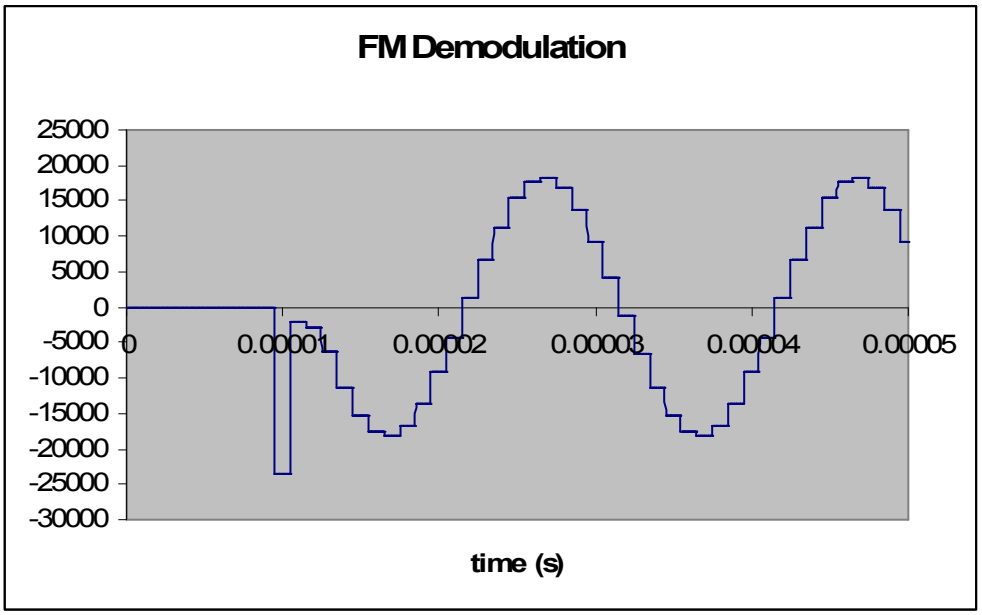
Next this is mixed in with 10 Mhz from our NCO in order to bring the signal down to baseband (high frequency components still present).



This is then decimated and lowpass filtered through the CIC filter.



The FM demodulator finally takes this data from the CIC, extracts the phase angle, and differentiates it to get the demodulated data. As we can see, we get a 10 kHz sine wave, as hoped (minus the initial latency period, of course)!

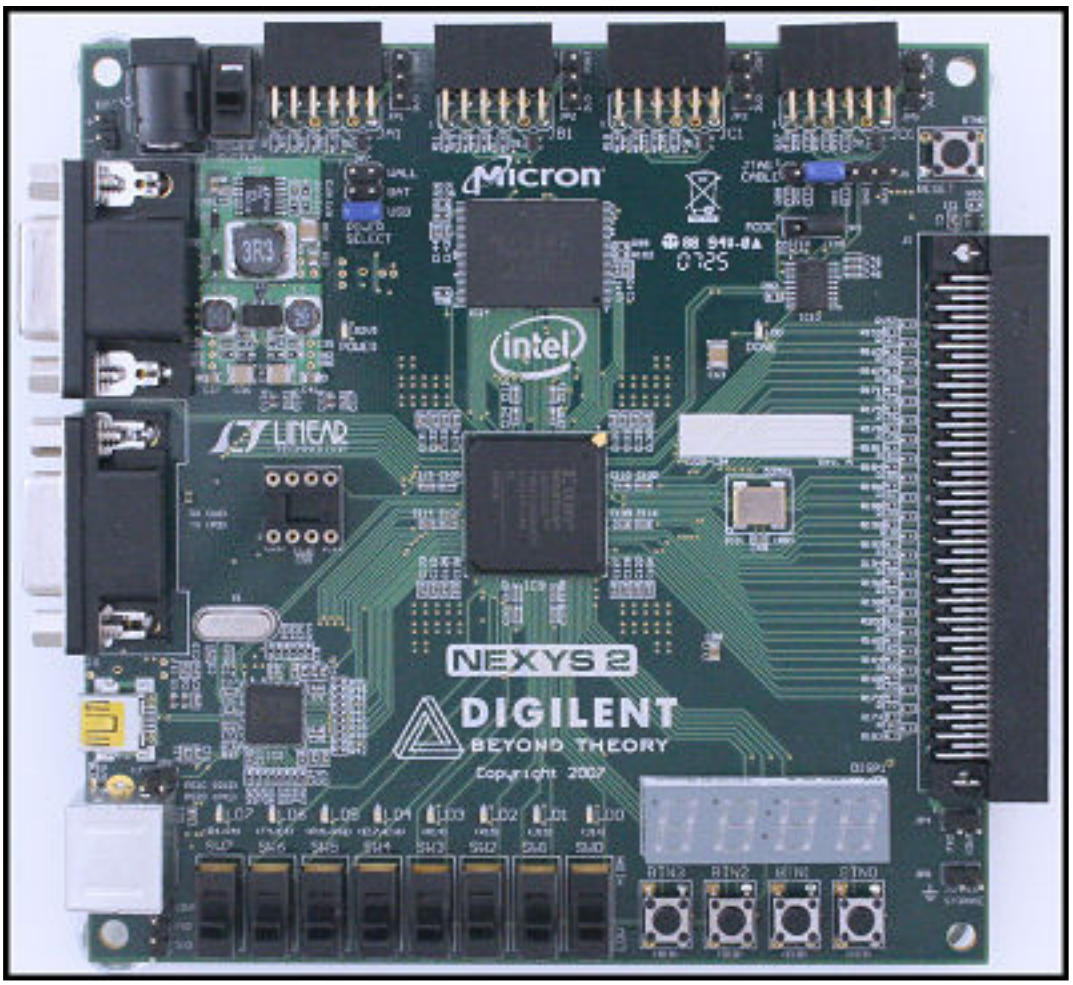


USER MANUAL

Components of the Software Defined Radio

Nexys2 Board

The Digilent Nexys2 Board is a Spartan 3E 1200K development board. It comes with enough I/O ports to serve the external peripherals used in this project. Below is a picture of the Nexys2 Board.



Infrared Remote Controller

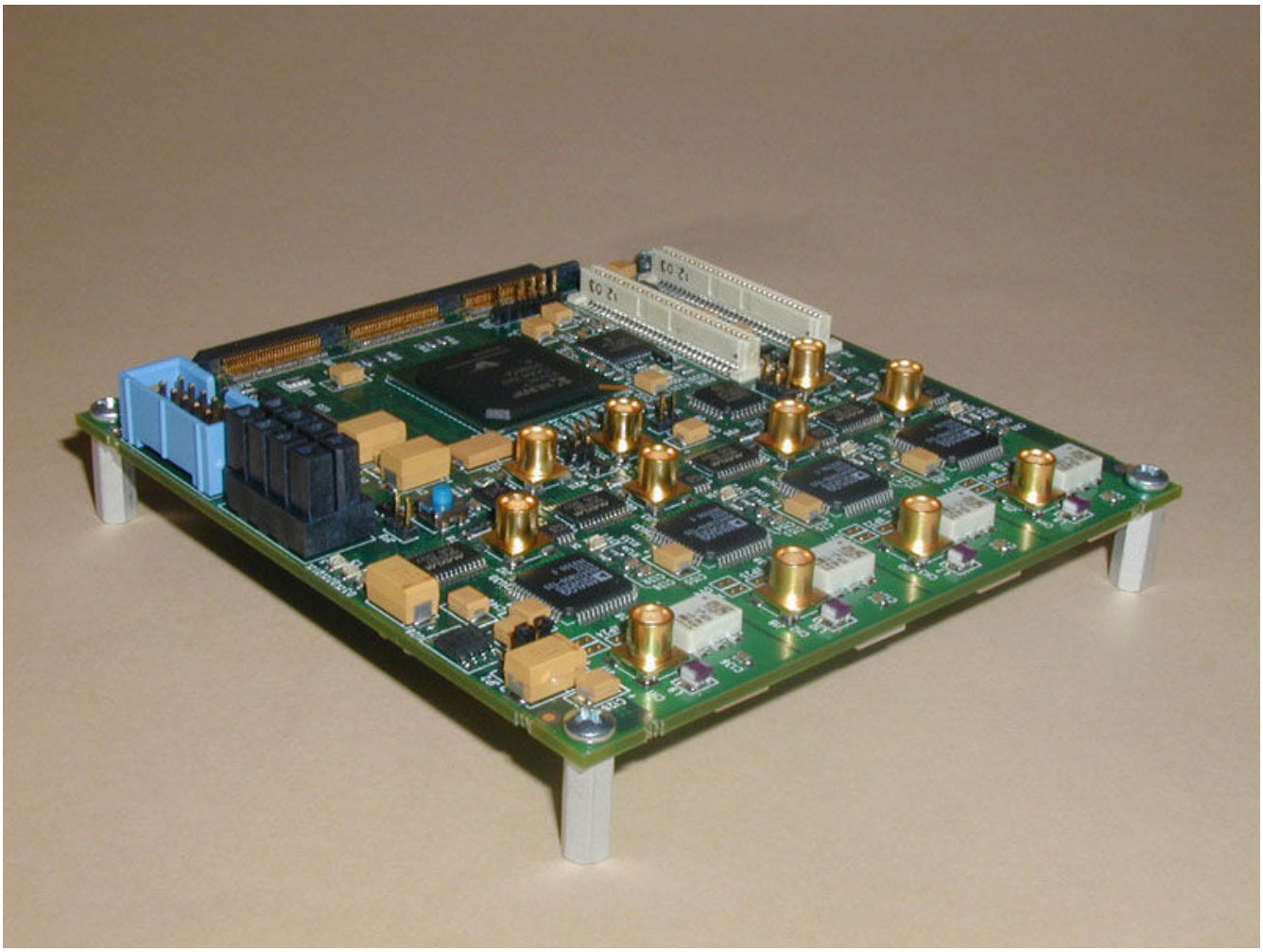
It only made sense that the user be able to control the project from a distance, so this multi-purpose infrared remote controller was chosen. The remote controller comes with channel up/down buttons (used for station up/down), up/down arrow buttons (used for station seeking), a number pad (used for inputting station frequencies of choice). Below is a picture of the infrared remote controller.

Newhaven Displays TFT LCD

With a 2.4” color graphic TFT LCD we had flexibility in how our station info was displayed. We could choose the color and size of the numbers as well as the background. This also gave us a chance to use something other than the standard 2-line character LCD that appears in most EE projects. Below is a picture of the LCD:

DSPBrik DA4012 ADC

This is the ADC board. Below is a picture



Other Components

The other peripherals that were on this project were the FM antenna, IR receiver unit, the audio jack and of course the audio speakers.

Controlling and Operating the SDR

This section deals with the inputs, outputs, and user interface of the SDR project.

Inputs

Below is a table with the input device and its description

Input Device	Description
IR Remote Control	This is a 29 button remote control which will be used to select the station being listened to. A more detailed description of the IR Remote Control is given later.

Outputs

Below is a table with the output devices and their descriptions

Output Device	Description
TFT LCD Display	It is a color graphics LCD. It shows the current station being listened to. A more detailed description of the LCD Display is given later.
Audio Jack	Connected to speakers to output the radio station's demodulated signal.

User Interface

The user interface's important components are the 29-button IR remote control and the display. Brief introductions have been given in the Inputs and Outputs sections. Below they will be covered in more detail.

The IR Remote Control:

It should be noted that on this remote control, only one button should be pressed at a time.

Simultaneous multiple button presses are really not possible. The remote controller will decide on what button was pressed first and will send out the corresponding signals. This will thus lead to

unpredictable results. This will be undesirable so avoid it.

The IR remote control's button presses have been debounced in the FPGA so the user does not have to worry about unintentional autorepeats happening.

The unused buttons on the IR remote control only give a tone when pressed, but they serve no function. Below is a table describing the used buttons and their functions. These used buttons do also give tones when pressed.

IR remote control button	**Description**
Channel Up	Used to increment the current station frequency by 0.2MHz. This saturates at 107.9MHz
Channel Down	Used to decrement the current station frequency by 0.2MHz. This saturates at 88.1MHz.
Up Arrow	Seeks the next station at a frequency above the current station frequency. Also saturates at 107.9MHz
Down Arrow	Seeks the next station at a frequency below the current station frequency. Also saturates at 88.1MHz.
Number Pad	The user uses this to select the station of choice. This will only allow the user to input valid stations. The valid stations are stations from 88.1MHz to 107.9MHz (inclusive) and only stations spaced apart by 0.2MHz. The way this works is detailed below: Only accept an input for the first digit if it is 1, 8, or 9. If it is an 8 or 9 then place it in the second digit and move the cursor to the 3rd digit. If it turns out that the user puts in a 1 for the first digit, then fill in a 0 automatically for the second digit and jump to the third digit. The 3rd digit can be any digit if the 2nd digit is 9. If the 2nd digit is 8, then the third digit can only be 8 or 9. If the 1st digit is 1, then the third digit is less than or equal to 7 and the 4th digit is the 1st digit after the decimal point and it can only be an odd number. This is because there are no stations at frequencies with even numbers after the decimal point. Note that with this system, the user only has to press 3 numbers to get to the station of choice. The first number being 1,2,8,9 and the third number being an odd number. The middle number depends on what the first number is. To get out of this number input mode while the user has started punching in numbers, all the user has to do is press a non-number pad button on the IR remote controller.

The LCD Display

This is a color and graphics TFT LCD.

It is setup to use a black background with white text. There are only 3 things being displayed on the screen.

The first is the current station being displayed. This appears in the middle of the screen and is of a font size which takes up a good chunk of the LCD display. It makes sense for the current station to be big and bold.

The second displayed string is the names of the people who busted out this project

“ASIF AND NODDY”.

The third thing that is displayed is the number input cursor. This only appears when the user is in number pad input mode. When the user presses a valid button on the number pad, the cursor appears over the third digit showing that a new number can be entered here. On the input of a valid number the cursor moves to the final digit and awaits any odd number. After this the cursor disappears again as the station is changed to the newly input value. This helps the user not be confused by the three-digit input system, which has been designed for their convenience.