Wireless MP3 Remote Jukebox with Atmel AT90S2313-4PC

WE'VE GONE FROM RECORDS AND 8-TRACKS TO CDS AND MP3S. CONVERTING TO MP3S HAS BECOME POPULAR, AND HOW TO STORE THE FILES IS AN INTERESTING TOPIC THESE DAYS. TUNE IN, BECAUSE THIS PROJECT COVERS A STORAGE SCHEME AND EXTENDS YOUR CURRENT MP3 LISTENING AREA.

Reprinted from
Circuit Cellar #134

By: Brian Miller

Regardless of the success of various music file sharing services, it's a safe bet that many computer users are converting their record and CD collections to MP3 files and storing them on the ridiculously large hard drives available in modern computers. Tiny MP3 players compete with CD and tape walkmans for portable use. With many computers powered up continuously for Internet access, it occurred to me that it would be nice if my computer could act as the "server" for a wireless MP3 jukebox.

Let me explain the concept in more detail. The idea is to allow you to listen to MP3 music files at any location within the home where there is an FM receiver (which could be a walkman, for example). This is made possible by feeding the server computer's audio output to a low-power FM broadcaster. A portable unit displays the contents of the server's MP3 file folders to allow you to browse through your music collection.

For convenience, you can scroll through up to four different folders containing lists of songs. The functions of picking a particular song, starting, stopping, and skipping a song are accomplished using a universal IR remote control. These commands are then forwarded to the MP3 server computer via a 433-MHz wireless link. Photo 1 shows all the components of the system.

For the server end, I designed two modules, a 433 MHz receiver and FM broadcaster. The receiver picks up commands sent by the remote control unit and feeds them to the server computer via a serial port. A dedicated PC application running in the background receives these commands and dispatches them to the Windows Media Player, which then plays the requested selection. The second module, the FM broadcaster, gets audio from the sound card output of the computer and broadcasts it.

To reduce cost and simplify the design, the 433 MHz wireless link operates in one direction only. That is, after you have selected a function, that command is sent to the MP3 server computer via the wireless link. If that transmission never makes it to the server, you will hear that nothing has happened and can issue the command again. However, to ensure that spurious commands don't disrupt operation, a dedicated decoder/encoder chipset is used. This performs all necessary functions to ensure that only legitimate packets are passed to the MP3 server.

For the remote unit to be able to display the contents of the MP3 folders on the server, it must have the contents of those folders downloaded to it prior to use.

Photo 1: This photo shows the various parts of the system. The mini-box to the left houses the 433-MHz receiver. The case with the large LCD is the MP3 remote control unit. Sitting above it are the RCA remote control and the FM broadcaster module. In normal use, only the FM broadcaster and receiver are located with the host computer, the remote controller is placed wherever you want to listen to the MP3 music.

This is done via the serial port on the MP3 server computer using the same application software that passes the incoming wireless commands to the Microsoft Media Player. The firmware in the remote unit can handle up to four different music folders.

To avoid the need to constantly update the remote unit's flash memory, it's recommended that four stable folders are chosen for remote playing, with other folders used for newly downloaded music or selections that are always changing. The remote unit contains a 32K x 8 flash memory, which can hold up to 200 song titles per folder (800 total). Because flash memory is nonvolatile, it will hold this song database even when the unit is off, which is important because the remote unit is battery-operated.

Remote Control Unit

The heart of the system is the remote control unit shown in Figure 1. It is built around an Atmel AT90S2313-4PC which has 15 I/O lines (12 of which are used) as well as an internal hardware UART. Because the unit requires so little power to operate, I chose to power it with four AA cells followed by an LM2936CZ5 low-dropout regulator.

It became obvious early on that the song display would not be useful unless it was able to display the complete song title and/or artist's name. Because there can be a lot of songs to scroll through, I also decided that four songs should be displayed simultaneously. Therefore, I chose a 4 x 40 LCD panel. This unit has a different wiring scheme than most common LCDs. It contains two HD44780 controller LSIs, one for the top two lines and another for the bottom two. The LCD

connection to the micro is made through seven lines of port B using the common 4-bit data interface. The control lines consist of an RS line and two ENABLE lines, one for each controller LSI. The LCD is sent commands only, no status is read back, so the R/*W line is tied low.

Rather than putting a keypad or a lot of switches on the front panel, I chose instead to install an IR detector module and use a universal IR remote control for the user interface. The IR remote is a commonly available RCA model CRCU410 set to emulate a Quasar TV (code 054). I chose this because it's a simple IR code that is easy to decode in software. The keys that are decoded and their functions are shown in Table 1.

Nonvolatile memory storage of the song lists is provided by a serial flash memory EEPROM. I used Microchip's 24LC256I/P8EA because it's commonly available. This chip is an I2C device, therefore it needs only a two-wire interface to the '2313 micro. Unfortunately, the '2313 doesn't contain a hardware I²C port, so a bit-banged software routine must be used instead. However, fortunately Atmel provides an app note describing I²C read/write routines for the '2313 when used as the master device.

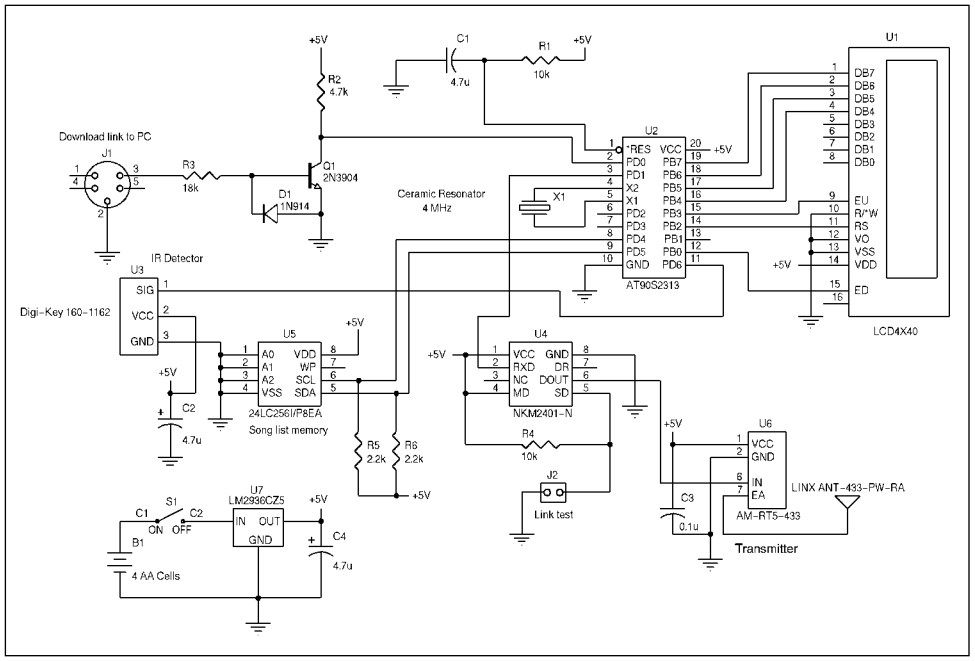

Figure 1: The MP3 remote controller is run by a AVR device.

Note that 2.2 kW pull-up resistors are required on both the SDA and SCL lines per the I²C specifications. The 24LC256I can assume up to eight different I²C addresses allowing for flash memory expansion to 256 KB, depending on the state of the A0 through A2 pins. Because you're using only one device, all three address lines are tied low.

The datasheet for this memory device states a 5-ms flash memory write cycle time. The flash memory is written to only during a data download, at which time the data comes in via the receive section of the UART of the '2313. My download protocol is strictly one-way from the server PC to the remote unit, using no handshaking. Therefore, a data rate of 1200 bps was chosen, which places the incoming data characters 8.3 ms apart. This period allows sufficient time to send the data to the I2C flash memory, even using software I²C routines, and have 5 ms to spare for the actual EEPROM write.

I didn't use a full-blown RS-232 interface like a MAX232. Instead, I converted RS-232 levels of the host computer to TTL with a single 2N3904 NPN transistor and a few passive components.

I chose an Abacom AM-RT5-433 for the wireless transmitter module. Conveniently, the AM-RT5-433 is contained in a small SIP package that's easy to mount. Abacom was kind enough to send me a sample of both the transmitter and receiver modules. These inexpensive modules are meant to cover a distance of 100 feet or so, using simple carrier on/off modulation for data transmission.

My other experience with small wireless modules was with the more costly Linx HP-II series (900 MHz). These are FSK units, the transmitter can be fed directly from a UART, and the receiver is a squelched unit that feeds a UART directly.

Luckily Abacom's technical staff pointed out that the low-cost units I had chosen can't be directly interfaced to a UART port. However, Abacom has designed a combo chip called the NKM2401-N, which can function as either a data encoder or decoder depending on the wiring of the mode pin. For this project, I use one of these devices at each end of the wireless data link.

In the remote unit, the NKM2401 accepts an 8-byte packet from the UART of the '2313 (at 2400 bps), adds sync, pre-/post-amble bytes and CRC, and performs Manchester encoding on the resulting data. Because the NKM2401 requires an 8-byte packet and my commands are only 2 bytes long, I add my own sync and filler bytes to make up an 8-byte packet. The MK2401's data output is connected directly to the data input of the AM-RT5-433 transmitter. For an antenna, I use a 1/4 wave whip.

I was hoping to be able to do without the encoded NKM2401 in the remote unit, instead depending on some '2313 firmware routines to perform the same functions. The remote unit's firmware is written in assembly language and uses only about one-third of the AT90S2313's 2 KB flash memory. There would have been plenty of room for the necessary routines there. However, at the receive end, I didn't plan to use a microcontroller, so I had to use an NKM2401 for decoding there. I was unable to convince Abacom to give me the exact protocol it uses for communication, which is understandable. Therefore, I wasn't able to write routines to do the encoding of the packets. Given more time, I could have captured the datastream using a digital scope or a program running on a micro and reverse-engineered it, but it wasn't in the cards.

Before moving on, let me mention a few miscellaneous details about the remote unit. The AT90S2313 operates at 4 MHz using a ceramic resonator. This is accurate enough for the slow serial data communications rate used. I had to set the UART to receive at 1200 bps (during data download) but transmit at 2400 bps (sending commands via NKM2401 and transmitter). The two rates are necessary because the NKM2401 works only at 2400 bps, and 1200 bps is the highest speed that can be used for downloads (considering the write cycle time of the serial flash EEPROM).

Lastly, there is a jumper labeled J1 Link Test on the remote control unit. When in place, the NMK2401-N sends out a constant "ABACOM" message, which can be used to check the wireless link.

Wireless Receiver Module

The job of the wireless receiver is to pick up the 433-MHz signal transmitted by the MP3 remote control unit and convert that signal to RS-232 level data for the server PC.

The Abacom AM-HRR3-433 receiver is shown in Figure 2. The receiver module is connected to the same type of 1/4 wave whip antenna as the remote transmitter.

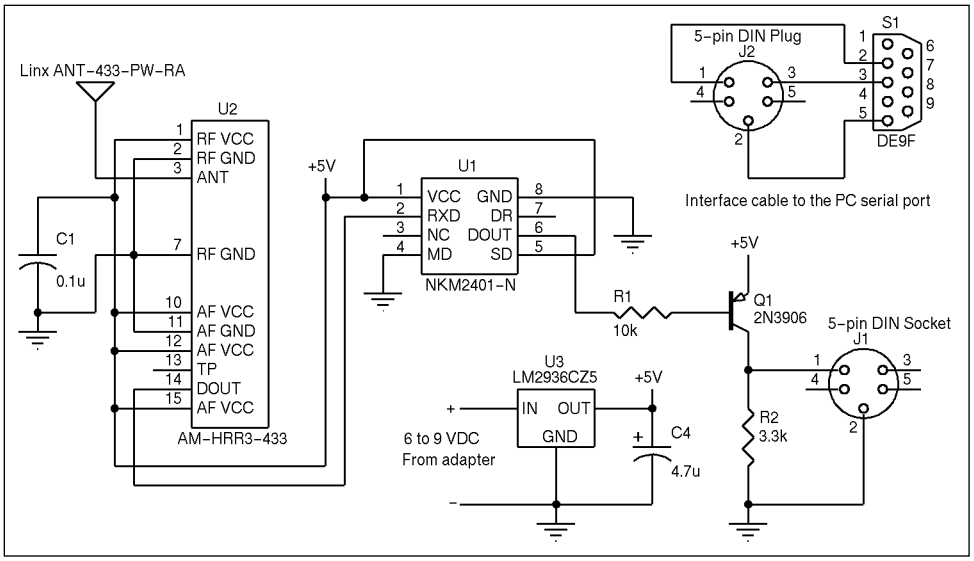

Figure 2: The 433-MHz receiver receives the wireless commands sent by the remote controller and passes them to the server computer for execution.

The output of the receiver module is full of spikes and noise in the absence of an actual signal coming in from the transmitter. I monitored it using an oscilloscope, and I live in a "quiet" RF rural area! For that reason, using Abacom's NKM2401-N decoder chip was a necessity. In the receiver, the NKM2401-N has its mode pin (pin 4) tied to ground to place it in Decode mode. A simple PNP transistor inverter is used to provide pseudo-RS-232 level signals to the server PC. An LM2936CZ5 low-dropout regulator is used to provide the 5 VDC needed for the receiver.

Most of the time this receiver module is left connected to a serial port on the MP3 server computer. However, from time to time, the remote control unit must be connected to the PC in its place (i.e., when the song lists are being downloaded to it). For that reason, I made up a short cable to connect the DB9 male socket on the PC to a five-pin DIN plug. Both the receiver and remote control unit use matching five-pin DIN sockets, and you simply hook up the proper unit as needed.

The Abacom receiver/transmitter modules, when used with NKM2401-N devices, are reliable. The wireless link for the command transmission was one of the smoother aspects of the project. The only complication I ran into was that I wasn't able to place the 433 MHz receiver and FM broadcaster modules in the same case. When the FM broadcast transmitter was placed next to the receiver, its RF output slightly desensitized the receiver. The result was that the wireless link would work for a distance of only about 20 feet, which was too short for my purpose. However, when I placed the FM broadcaster in its own case and moved it several feet away from the 433 MHz receiver, the problem disappeared and the range increased to about 50 feet (keep in mind, that's still within the house).

FM Broadcaster

When I originally conceived this project, I anticipated some challenging design or programming problems. I assumed getting a small FM broadcaster module would be easy, so I tackled that job last. However, of course, Murphy's Law dictated that this would turn out to be the most frustrating and time-consuming part of the project!

I had heard rumors that stereo FM transmitter kits based on the Rohm BA1404 IC were too unstable to be useful. Not easily deterred, I bought such a kit anyway. Alas, the rumors were true—its frequency stability is too poor to work with modern, digitally-tuned FM receivers. Even when I replaced the cheap RF tuning components with high-quality parts, the problem remained. In fairness to Rohm, I expect that this IC was designed before the advent of digital FM receivers. Older analog FM receivers had an automatic frequency control circuit that likely would have tracked the frequency changes that occur with this transmitter kit.

I found a PLL-stabilized FM broadcaster kit, but it cost about $200, which was too steep for this project. A number of years ago, I built several PLL-based frequency generators in the 10 to 400 MHz range, so I thought I'd try to roll my own FM broadcaster.

That's when the trouble began. The PLL chips that I had used in the past were no longer available. Most of the currently available PLLs are meant for cell phones and the like and won't work reliably below 100 MHz. I found some that were targeted for the FM broadcast band, but they were in such tiny packages that I couldn't solder anything to them.

At this point, I decided to try something different. Because I needed a microcontroller to load the PLL chip anyway, why not try to let the microcontroller measure and control the oscillator frequency and dispense with the PLL chip entirely? What I had in mind could be considered an automatic frequency stabilizer.

The basic concept is demonstrated in Figure 3. The oscillator frequency is determined primarily by the values of the inductor and variable capacitor. For this design, I chose an overall tuning range of roughly 88 to 92 MHz for two reasons. First, there are fewer commercial FM stations at the bottom of the band. More importantly, 96 MHz is the highest frequency that can be measured using this circuit.

Fine-tuning of the oscillator, both for stabilization and FM modulation purposes, is handled by a varactor diode. The bias on the varactor sets its capacitance. This bias has two components. A DC voltage level is applied by a 12-bit DAC, and an AC signal is superim-

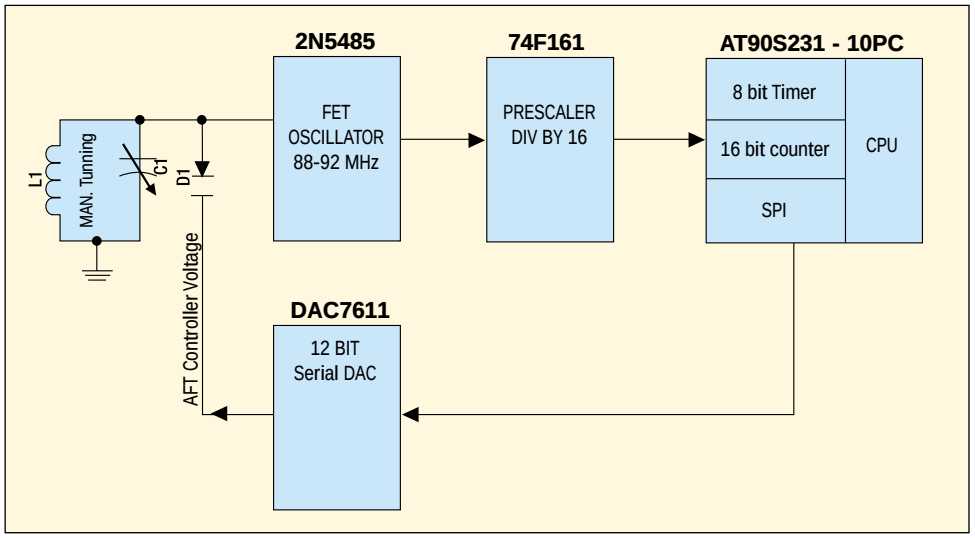

Figure 3: The block diagram shown here outlines the FM broadcaster.

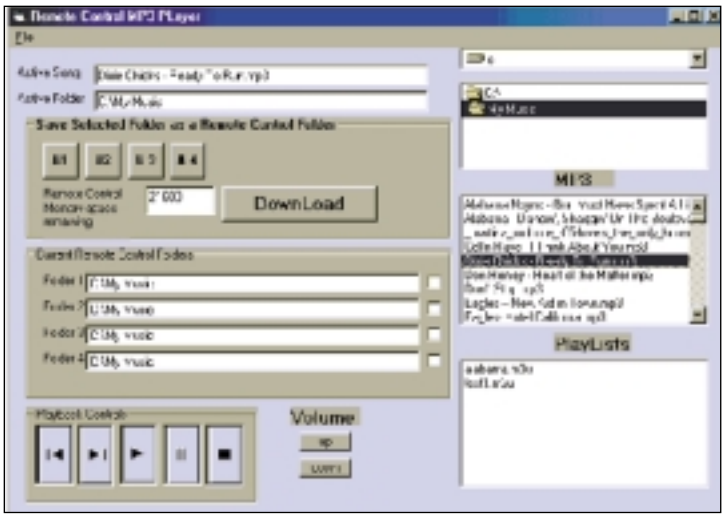

Photo 2: This is the PC application program that controls the playing of MP3 files through the Media Player, from wireless commands received from the MP3 remote controller. The code's other main function is to allow you to choose up to four music folders and then download the song titles (file names) to the flash memory in the remote controller.

posed on it to perform the frequency modulation. The DAC output voltage is initially set to a middle range (2 V) and you manually tune the oscillator to the desired frequency by adjusting the variable (trimmer) capacitor. Thereafter, the microcontroller will adjust the DAC voltage up and down slightly to maintain the proper frequency.

For the microcontroller to measure the oscillator frequency, it must first be prescaled by a factor of 16. This is done using a common 74F161 4-bit divider chip. The prescaler output is in the 5 to 6 MHz range, which can be directly counted by the microcontroller using its 16-bit counter/timer.

To determine the oscillator frequency, clear the 16-bit timer and then read it after a fixed interval. This fixed interval is provided for by another counter/timer in the microcontroller, which is programmed to provide a periodic interrupt every 5.461 ms. Using this arrangement, the expected value in the 16-bit counter equals:

$$\text{counter/timer Value} = \left(\frac{f_{osc}}{16}\right) x\,(5{,}461x10^{-3})$$

In operation, the microcontroller reads the count in the 16-bit timer and compares it to a fixed constant derived from this equation using the FM broadcast frequency chosen by you. If the oscillator frequency is too low, it bumps up the DAC value by one and tries again. Conversely, if the oscillator frequency is too high, it decreases the DAC value by one. This process is repeated until the oscillator frequency falls within a narrow band around your set point frequency.

The circuit described here normally would be hunting constantly for two reasons. First, there always would be a one-count bobble in the counter/timer value because of variations in the position of the sampling interval with respect to the oscillator signal. Secondly, because the oscillator is being frequency-modulated by an audio signal, its frequency will vary from this modulation voltage.

This hunting is undesirable because it produces artifacts in the music heard through the FM receiver. To prevent this, as soon as the microcontroller has tuned the oscillator so that the 16-bit counter/timer is within two counts of ideal (an accuracy of about 6 kHz), it will "go to sleep" for 10 minutes, after which time it will check the frequency again. Unless the room temperature changes significantly, the frequency of the oscillator will require little correction, and this circuit will apply it as necessary.

AVR AFC

While I describe how I implemented the automatic frequency-controlled FM broadcaster in detail, refer to Figure 4 on page 35for a visual representation.

To begin, I needed a microcontroller that could count pulses at a 6 MHz rate, also containing another timer to generate the periodic interrupt needed to read and clear this counter. I chose the Atmel AT90S2313-10PC because it has the required functionality. However, I have to run the device at 12 MHz instead of its rated 10 MHz to achieve the 6 MHz counting rate. By the way, I have not experienced any problems "over-clocking" the '2313 by this modest amount.

The setting of the FM oscillator frequency will be performed only once, when the unit is first set up and a clear channel is found on the FM dial. Therefore, to make things simple, the desired broadcast frequency is entered into the program code as a constant at the beginning of the program. The program is then compiled and downloaded into the '2313, resulting in a fixed-frequency FM transmitter.

Ten years ago, you could find prescaler chips readily available that would divide by 256 at frequencies up to 1 GHz, but these are no longer manufactured. So, I used a 74F161 four-stage counter to implement a divide by 16 prescaler. It handles frequencies greater than 100 MHz, costs less than $1, and is readily available.

I built my own VCO around a 2N5485 FET device. The resonant frequency of the VCO is largely determined by the values of L1 and C12. The latter is a trimmer capacitor that is adjusted manually when the unit is first powered up to achieve roughly the desired frequency. This tuning is performed with jumper J1 in place, which causes the microcontroller to set the DAC to mid-scale.

Thereafter, the jumper is removed. And when the unit is powered up again, the automatic frequency stabilization circuit starts to operate and the oscillator is fine-tuned to the desired frequency by D1, the varactor diode. The 74F161 prescaler requires several volts of signal for proper clocking. The FET oscillator can provide this (most other oscillator configurations are not capable).

The RF output from the oscillator is taken from a tap on L1 to reduce loading effects. This is capacitor-coupled to the clock input of the first stage of the 74F161 prescaler. Potentiometer R7 is adjusted to provide the proper bias on the clock input pin, allowing the oscillator signal to properly trigger the input divider stage. It should be set for around 2 to 2.5 V, but is best set up by using an oscilloscope and looking for a clean 5 to 6 MHz waveform on pin 11 of the 74F161.

The 12-bit DAC (U4) that controls the VCO's fine-tuning is a TI DAC7611, with an SPI interface. Although the '2313 doesn't have a user SPI port (its SPI port works strictly for flash memory programming), it's simple to bit bang the SPI data out to the DAC using I/O lines PB1 through PB4.

Shown just below the '2313 in Figure 4 is jumper J1, which is connected to port line PD6. At reset, the microcontroller checks the state of this line; if the jumper is in place, it merely sets the DAC to mid-scale (2.048 V) and waits. This allows you to set the mechanical trimmer capacitor (C12) for an oscillator frequency as close as possible to the desired FM broadcast frequency. In North America, all FM broadcasting is done at odd multiples of 100 kHz, so pick a frequency accordingly.

Having accomplished this, next you remove the jumper and powerup the unit again. The device should home in on the desired frequency within a few seconds by repeatedly adjusting the DAC voltage and measuring the resulting frequency via the prescaler. This is the normal operating mode when subsequently being used as an FM broadcaster module.

Line-level stereo audio from the host computer's sound card line out is first run through a pre-emphasis network (for each channel), and then mixed down to a monaural signal. This low-level AC signal is superimposed on the DC control voltage of the DAC and used to frequency-modulate the oscillator.

The RC values in the pre-emphasis network were

picked by monitoring the output of a stereo receiver and aiming at a flat-frequency response. The values shown in Figure 4 come pretty close. Don't expect the chosen component values to provide a time constant of 75 µs, which is the standard pre-emphasis used by broadcasters. Consider that there are many other factors in the VCO that affect the modulation characteristics. My values provide an overall flat-frequency response as measured with a high-quality FM receiver.

Note that no antenna is illustrated in Figure 4. If the project is placed in a plastic enclosure, it will radiate enough signal to cover a 50¢ radius. This works out well, because government regulations prevent using a transmitter capable of covering a much greater range than this.

I built the VCO section of the circuit (shown within the dashed lines in Figure 4) on a small, single-sided PC board with dimensions of about 1.5" x 1". The remaining part of the circuit was hand-wired on a Simm-Stick protocard. The VCO PCB is designed like a SIP package and mounts vertically on the Simm-Stick.

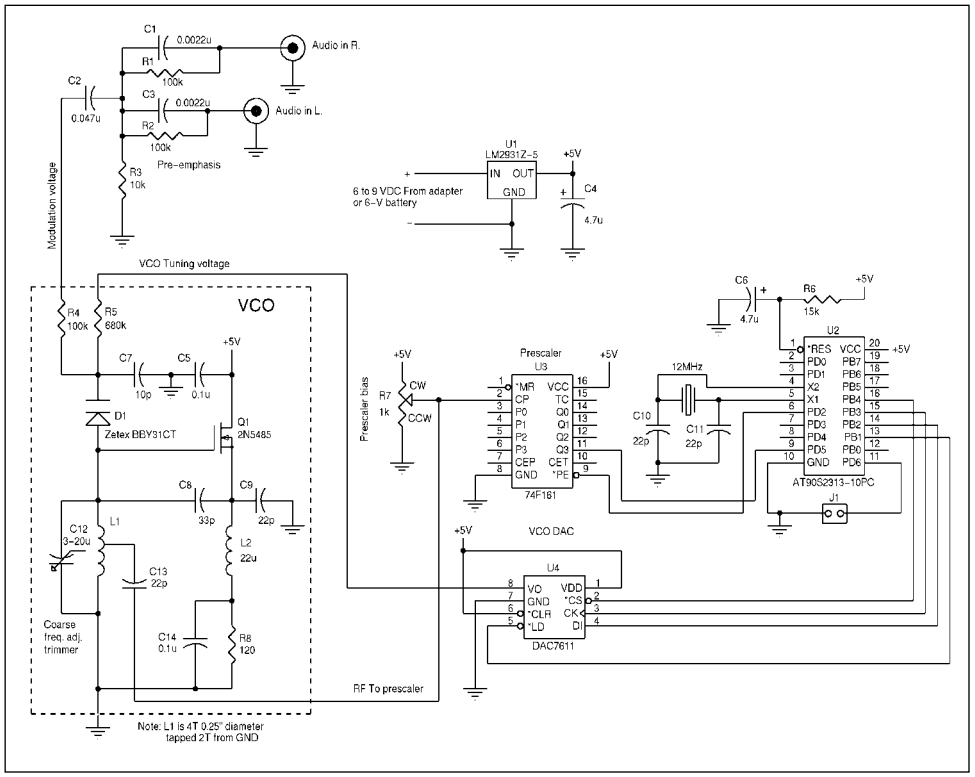

Figure 4: The FM broadcaster unit is automatically fine-tuned.

Software and Firmware

The application software consists of a server application running on a computer and client firmware running on the remote controller. At the PC end, the server software is written in Visual Basic 6.

The remote controller firmware is written in AVR assembly language. The FM broadcaster is frequency stabilized by another AT90S2313. The program required for this is trivial, so I used the BASCOM-AVR compiler for that application.

Client Application Software

The client application running on the PC has two main functions. Most of the time it is polling the COM1 serial port looking for commands that have been sent to it by the remote controller. Its other main function is to allow you to browse through your directory structure and designate up to four folders as jukebox folders.

The file names in these folders are then converted into a database record and sent to the remote controller using the transmit section of the COM1 serial port. This download is performed only once unless the contents of the folders change, because this data is stored in the song list flash memory within the remote controller (Photo 2).

Let's look at the first function in more detail. As mentioned earlier, the connection between the computer and remote controller is a 433 MHz wireless link. At the PC end, the 433 MHz receiver takes the RF signal and converts it into serial data at 2400 bps, which is fed into the COM1 port. All data formatting and error checking are performed in the hardware using the NKM2401 encoder/decoder chips. This method ensures that any command received by the client application will be legitimate.

The NKM2401 uses 8-byte packets. All commands sent by the remote controller consist of a 2-byte sync pattern (0xAA, 0x55) followed by a 16-bit command word and four dummy bytes. Two bits of the command word are used to designate the four commands: Play (Stop), Play Next, Play Last, and Pause. The other 14 bits are used to select both the active folder and offset of the song's file name within that folder.

Earlier, when you picked out the folder for use by this application, that folder was parsed and a fixed record length database was generated, a copy of which is maintained both by this application (in a file) and in the remote controller (in the song list flash memory). Doing it this way allows you to see all of the song names on the remote LCD readout. In addition, it means you can select songs to play by merely sending a number corresponding to the song's position in the database file.

After the client program knows what song to play, how does it actually get the computer to play that particular song? If the songs were in WAV format, it would be easy, as Visual Basic has built-in support for multimedia functions including the playback of WAV files. However, it does not support the playing of MP3 files, at least not the version that I have. I looked around for a shareware/freeware ActiveX control that plays MP3 files but wasn't able to find one that was free or reasonably priced.

I took a different tack at this point. I had just downloaded the Windows Media Player V.7 that Microsoft distributes for free. This multi-purpose program handles MP3 files nicely and has all the bells and whistles people want. Like many Windows programs, it allows for keyboard shortcuts, an advantageous capability. My idea then was to run my MP3 Jukebox client application and the Windows Media Player concurrently and let my client application send keyboard strokes to the Media Player to control it.

This is done using a couple of Visual Basic commands. The Shell command transfers program flow over to the Media Player, and the Sendkeys command sends the proper codes to the Media Player, causing it to select the desired song and play it or do some other functions. To check that the client program is actually communicating with the Windows Media Player, I added the capability of picking a specific song from a folder and playing it (without a command coming in from the remote control). As Photo 2 demonstrates, you may choose the song to be played using the drive, folder, and file windows on the right, and the transport controls located along the bottom will play or stop it.

The second function, used occasionally, chooses the desired music folders (up to four) and downloads the

contents into the remote controller's flash memory. You do this by picking out a specific folder and then clicking on one of the four numbered buttons located to the left of the form. A window keeps track of the space remaining in the song list flash memory. Each folder can contain almost 200 songs before its flash memory allocation is exceeded.

To simplify the firmware in the remote controller, I make the assumption that all four folders will be used, and therefore downloaded. If you have fewer folders, the remaining folders should be designated as duplicates of the desired folder(s).

Before pushing the Download button, the remote controller must be plugged into the COM1 port of the PC, temporarily displacing the 433 MHz receiver. I use the custom cable described earlier, which stays plugged into the PC. The other end of the cable is a five-pin DIN plug, which fits sockets on either the receiver or the remote controller. The download time is dictated by the write timing of the flash memory in the remote controller. Download time is about 4 minutes if all four directories contain the maximum of about 200 songs. The download progress is indicated by check boxes appearing next to the client program's folder list as well as messages on the remote controller's LCD screen.

AVR Firmware

I already outlined the automatic frequency control of the FM broadcaster. The program that accomplishes this is simple, and therefore was written in Basic and compiled using the BASCOM-AVR compiler. For more details, download the program listing contained in the file MP3FM.bas, which is available on Circuit Cellar's web site as well as my personal web site.

The remote controller firmware was much more involved. I first tried to write it in Basic using the BASCOM-AVR compiler, but the program code generated would not fit into the flash memory of the '2313. Using assembly language, I accomplished the same thing in less than half of the flash memory space.

As with the PC client software, the remote controller performs two functions, one of which happens infrequently. When turned on, it displays the first four songs in folder one. It then goes into a polling loop to detect IR commands sent by the RCA universal remote and received by the IR receiver module. This IR signal comes into port D6 of the '2313, which is the INPUT CAPTURE pin.

The IR codes are deciphered by using the input capture feature of timer1, a 16-bit timer/counter. I chose a simple IR command structure (RCA code 54, Quasar TV) to make my job easier. This command structure has a fixed-length start pulse at the beginning of each command sent, followed by 8 bits of data. The data bits are represented by two different intervals between pulses. After recognizing the fixed length start pulse, you have to do only two things. You must capture the timing of the eight subsequent pulses and, from the interval separating them, assign the proper bit values. As mentioned earlier, the remote control constantly polls the IR receiver for commands and then performs the proper function. Many keys allow for navigation only through the song lists and moving from one folder to another. Do this by adjusting pointers into the storage flash chip, reading the song names, and transferring these ASCII characters to the LCD screen.

Key	Function
One through four	Play one of the four songs currently shown on display
Five	Skip to next song
Six	Jump back to previous song
Seven	Stop/start again
Enter	Prepare for song download from host PC
CH+	Scroll to next screen of four songs
CH–	Scroll to previous screen
Left VOL	Select previous folder (four total)
Right VOL	Select next folder (four total)

Table 1: I defined the buttons of the RCA universal IR remote control to function properly with the MP3 remote controller. The remote control must be set up to emulate a Quasar TV remote for this application (code 54).

The keys used for Play, Play Next, Play Last, and Pause actually send out the proper command via the UART transmit port of the '2313. Again, the NKM2401 uses 8-byte packets, so sync and filler bytes are added to the 16-bit command word as needed.

There is a key dedicated to the download function. When pressed, it shifts program execution to a routine that accepts characters coming into the UART receive port of the '2313. The database generated by the PC is transferred into the song flash memory using this function.

The Atmel 24C256 flash memory chip is an I²C device with a 32K x 8 storage capacity. Its 5 ms write timing isn't a problem, because the data coming in from the server computer is sent at 1200 bps, which is an 8.3 ms per character rate. The '2313 doesn't have a dedicated I²C port so this function must be done in firmware. The master I²C routines were taken directly from Atmel's application note and work fine.

The LCD is a 4 x 40 unit that uses the ubiquitous Hitachi HD44780 controller. Actually it contains two controller LSIs, with a common data/control interface in addition to two enable lines. I had to rewrite my trusty 4-bit LCD driver routines to handle the fact that lines one and two use controller one and lines three and four use controller two.

All Played Out

I found this project to be very interesting, partly because of the wireless aspect. The Abacom receiver/transmitter modules coupled with the company's encoder and decoder devices work well. Moreover, the user-friendly flash programming capability of the AVR device made writing the assembly language firmware nearly painless for me.

However, there was one disappointment. I didn't anticipate spending so much time coming up with a satisfactory FM broadcaster module. Although too late for this project, I recently came across the Rohm BH1416F Wireless Audio Link IC, which contains a complete PLL-stabilized FM transmitter and FM stereo modulator in a SOP22 package. I bought a few of these to try out later. If you're not a big music fan, maybe you'll be able to apply some of the ideas mentioned here for other useful remote control concepts.