MIT Wakeup Call System

6.111 Final Project

Team Members:
Ruby Pai
Surapap Rayanakorn
Audrey Roy

Project TA:
David Milliner

Abstract

The MIT Wakeup Call System provides a wakeup call service for 5-digit MIT campus phone numbers. The system takes wakeup call requests from each user over the phone line, and then it calls them back at the times specified by the user.

An interface to a phone line was implemented with Zarlink's MH88437 data access arrangement and MT8889 DTMF transceiver chips. Together these chips are capable of detecting incoming calls, picking up the phone line, detecting when the other end of the line has been picked up, and dialing and receiving dialed DTMF tones as binary digits. Prerecorded audio messages could also be sent over the phone line.

An audio unit plays back voice messages or music to the phone circuit upon the request of the control unit. Eleven messages and one music piece are stored in ROMs. Once selected, a digitized message is sent to the codec chip LM4550. The codec chip then converts the digitized sample to an audio voltage, which subsequently goes to the phone interface.

A request memory unit performs the storage, timing, and retrieval of wake-up call requests. It receives the data for each request from the phone interface, and this data is stored in a RAM. At the appropriate times, the requests are retrieved, and the phone numbers are sent back to the phone interface for dialing. The requests are stored in sorted priority order, to allow for easy storage and retrieval.

Table of Contents

List of Figures

1 Introduction

A wake-up call system would be very useful to the MIT community. Most students at MIT use an alarm clock, but all too often it is not enough and students sleep through lectures, meetings, and exams. This wake-up call system addresses the problem.

When a user calls, he is prompted to enter his phone number and a four digit PIN number through his phone's keypad. He hears a menu with two options: to request a new call, or to cancel his next upcoming request. If the user chooses to request a call, he will also be prompted to enter the time desired (AM/PM, hour and minute). For both cases an acknowledgement message is played.

At the requested times, the system calls the user. The user picks up the phone, and a wake-up message is played, followed by music.

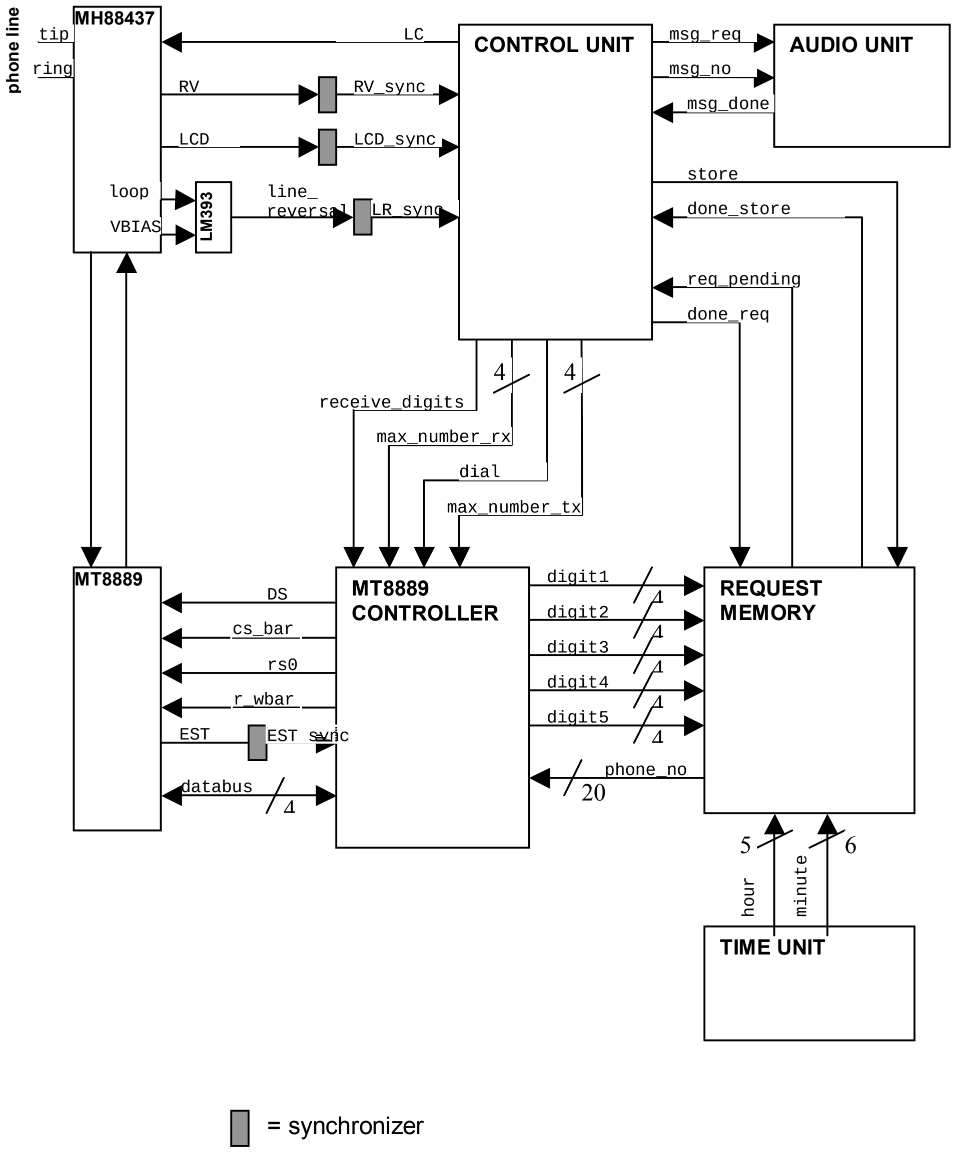


Figure 1: System Block Diagram

2 System Overview

Our system is implemented with the following major components: a phone line interface (external circuitry using a data access arrangement chip and DTMF transceiver chip, with an internal controller for DTMF transceiver), an audio unit for playing system messages over the phone line, a "smart" request memory unit that stores request information and using this information to tell when a wakeup call should be made, and finally a control unit responsible for coordinating all these
components (see Figure 1). In addition to these major components, there are a time unit that can be set on startup and keeps track of real world time and a PIN/phone number look-up table.

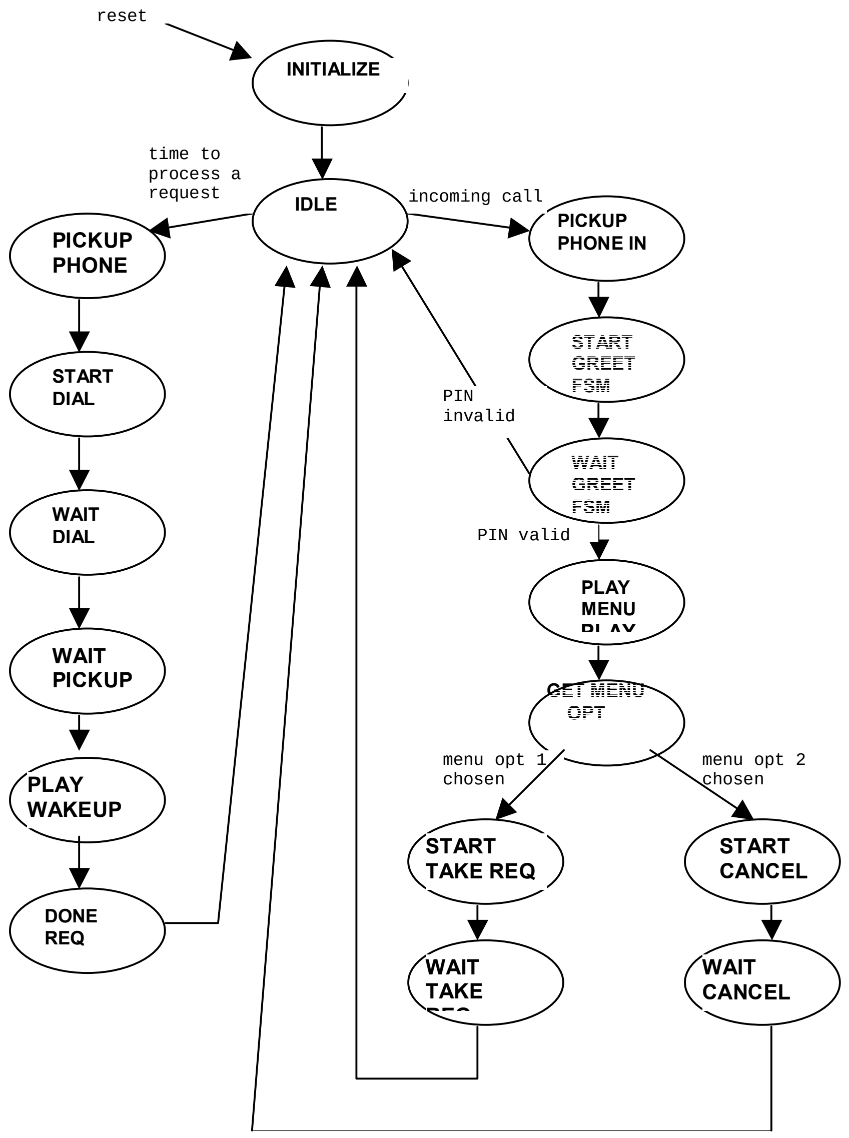


Figure 2: Control Unit Major FSM High-Level Transition Diagram

A high level FSM diagram for the major FSM of the control unit is shown in Figure 2. This traces the events that happen for the two scenarios in our system, taking an incoming call and making an outgoing wakeup call. The control unit contains three minor FSMs: a greet FSM that asks for and receives the user's phone number and PIN, and checks if it is valid; a minor FSM for taking requests, which asks for and gets the time of the request, causes the information to be stored and plays an acknowledgement message; and a minor FSM for canceling requests

3 Design

The most important system components discussed above were partitioned among group members in the following way: the control unit and phone line interface, the audio unit and the request memory unit.

3.1 Phone Line Interface

The phone line interface was implemented with two chips, the MH88437 data access arrangement and MT8889 DTMF transceiver chips. The MH88437 connects directly to tip and ring of the phone line. It detects when the line is ringing by pulsing RV high. The line can be picked up by asserting LC. After LC is asserted, the line is off hook the MH88437 asserts LCD. When the other end of the line is picked up, LR (line_reversal), derived from LOOP and VBIAS, goes high.

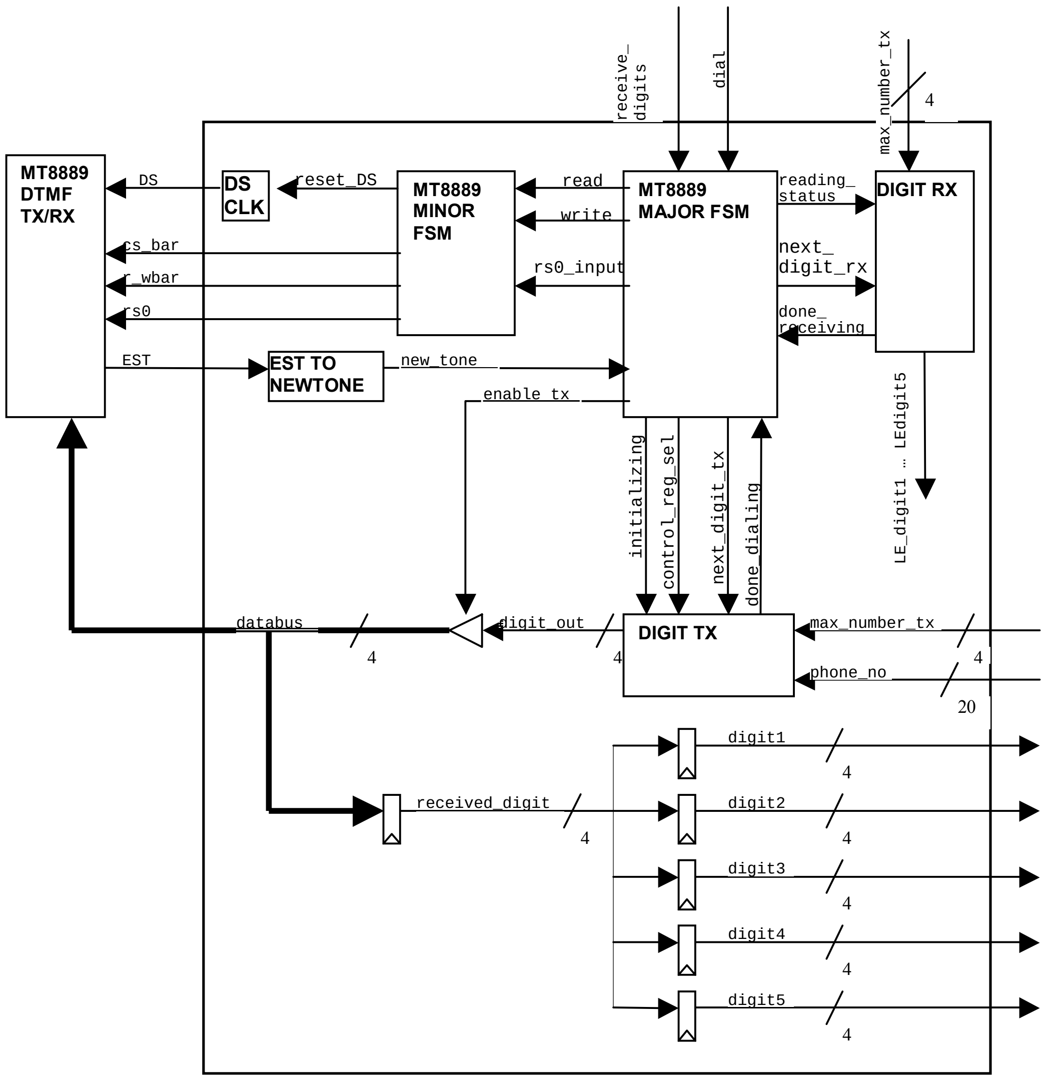


Figure 3: MT8889 Controller Block Diagram

The MH88437 control signals are straightforward and fit with the high-level flow of events that occur in either scenario of the system (incoming and outgoing calls)**.** For these reasons, the MH88437 was controlled directly by the major FSM of the control unit.

The MT8889 DTMF transceiver converts binary 4-bit digits to DTMF for dialing and DTMF to binary for interpreting digits being dialed at the other end of the line. Because the MT8889 was designed to interface to many different specific microprocessors, and had different operating modes, controlling it with a FPGA was more involved and was therefore implemented in a separate control unit. (Our system emulated Motorola control in DTMF burst mode with interrupt request enabled).

The MT8889 has five internal 4-bit registers, and two operations, read and write. The transmit data register must be written to transmit a DTMF tone. When there is an incoming DTMF signal, the receive register can be read to get the dialed digit. The remaining three registers are for control: control registers A and B must be written on power up to initialize the chip and select options for operating mode. The status register can be read during operation; basically it tells whether a tone has been sent or received.

The MT8889 control unit has the following important components: a major FSM, a minor FSM, digit_tx and digit_rx modules (see figure 3).

On receiving a read or write command from the major FSM, the minor FSM sets up the control signals (DS, cs_bar, r_wbar, and rs0) for an individual read or write operation to the MT8889 (see selected Verilog code in wakeupcode.pfd).

Digit_rx counts the number of digits that has been received, and based on the current digit number causes the appropriate digit register to be loaded. Digit_tx counts the number of digits that has been dialed and selects the appropriate bits of the phone number based on that count. The major FSM interacts with digit_rx and digit_tx by asserting the respective next_digit signals after a digit has been received or dialed. Digit_tx and digit_rx assert their respective done signals to the major FSM when their counts have reached the number specified by the top level control unit.

Whenever the MT8889 receives a new dialed digit, new_tone, derived from MT8889 EST output, will pulse high. When receiving digits, the major FSM waits for the new_tone pulse, waits for the maximum amount of time it takes for a tone to be converted, then tells the minor FSM to read the MT8889 receive register. After performing the read, the minor FSM causes the read data to be registered as digit_received while it is guaranteed to be valid. After the minor FSM finishes, the major FSM causes received_digit to be loaded into the appropriate digit register by asserting enable_load to digit_rx. After the received digit has been loaded, the major FSM asserts next_digit to digit_rx, then waits to see if digit_rx returns a done signal.

A similar sequence of events occurs for dialing.

3.2 Audio Unit and Audio-Phone Interface

3.2.1 Overview

The audio unit plays back voice messages to the phone circuit upon the request of the control unit. It essentially consists of AUDIO_FSM and AC97 CONTROLLER (called "digloop" in the code in the appendix). The AUDIO_FSM controls the addressing of the twelve ROMs, and when to load a new sample of the voice message to the AC97 CONTROLER. AC97 CONTROLLER configures the gain, sample rate, and D/A conversion of the codec chip LM4550. All the components in the audio unit is clocked by the system global clock and reset by the global synchronized reset signal.

Overall behavior of the audio unit in the playback mode is as follows. AUDIO_FSM waits for signal `msg_req`, and `msg_no` from the CONTROL UNIT. Signal `msg_req` notifies AUDIO_FSM to start playing a message from a ROM. The ROM is selected by signal `msg_no.` AUDIO_FSM plays the message by outputting the address of the ROM from the first line to the last line. The frequency of incrementing the address, i.e. the sampling rate is controlled by DIVCOUNTER. In this design, the addressing is operated at 8kHz.

Once supplied the address, the ROM gives out an 8-bit audio signal `audio_int,` which will be selected my the signal `msg_no` to the multiplexer. Next, the 8-bit audio signal goes to AC97_REG which registers the signal to ensure that the output to AC97 CONTROLLER (called "digloop" in the code) is free of glitches. The codec chip LM4550 expects that the audio input is 20-bit wide in the twos-complement format, so AC97_REG also appends twelve 0's to the 8-bit audio signal before passing it to AC97 CONTROLLER. The new input audio signal is loaded to be the output when asserted by `LE_audioreg.`

AC97 CONTROLLER configures the codec chip to take in the 20-bit audio signal at 48 kHz rate. Therefore, one sample from AC97 REG is used for 8 times, i.e. AC97 CONTROLLER upsamples the audio signal from AC97

REG. The codec chip LM4550 takes in the audio information bit by bit, converts it to an analog voltage output at 0V bias. Then, this analog audio voltage goes through a bias shifting circuit, which centers the audio signal around 2V before passing it to the phone interface.

Please refer to the verilog code in the appendix for more detailed implementation of the overall audio unit.

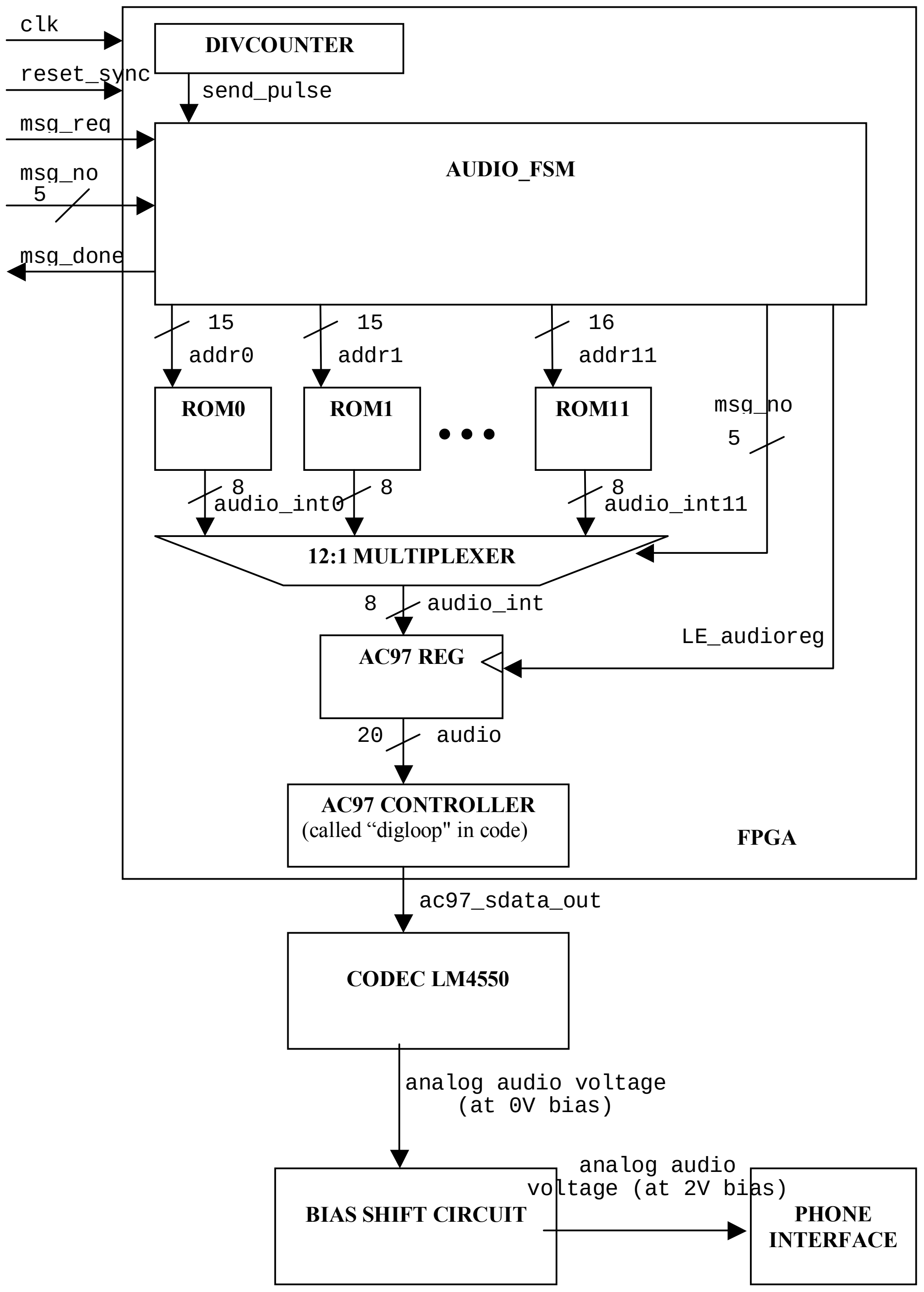


Figure 4: Audio Unit Block Diagram

3.2.2 Audio FSM

AUDIO FSM starts in the IDLE state upon being reset. In this state, all the address variables, signals `msg_done, count_int, and LE_audioreg` are initialized to be zero. One clock cycle later, AUDIO FSM moves to state WAIT_REQ, where it waits for the start signal `msg_req` and the signal `msg_no` specifying which message, i.e. which ROM to be selected. Once the signal `msg_req` is asserted high, AUDIO FSM goes to the next state, WAIT_SEND_ADDR, and waits for signal `send_pulse` which is high every 3,375 clock cycles of the 27 MHz clock. In other words, the signal `send_pulse` is asserted high 8kHz.

When the signal `send_pulse` becomes high, AUDIO FSM makes signal `LE_audioreg` high to load the most recent audio sample to the output of AC97 REG before moving to the next state, UPDATE_ADDR. In this state, AUDIO FSM makes signal `LE_audioreg` low again to disable the loading of AC97 REG before changing the address of the ROM in the next state, UPDATE_ADDR_DELAY.

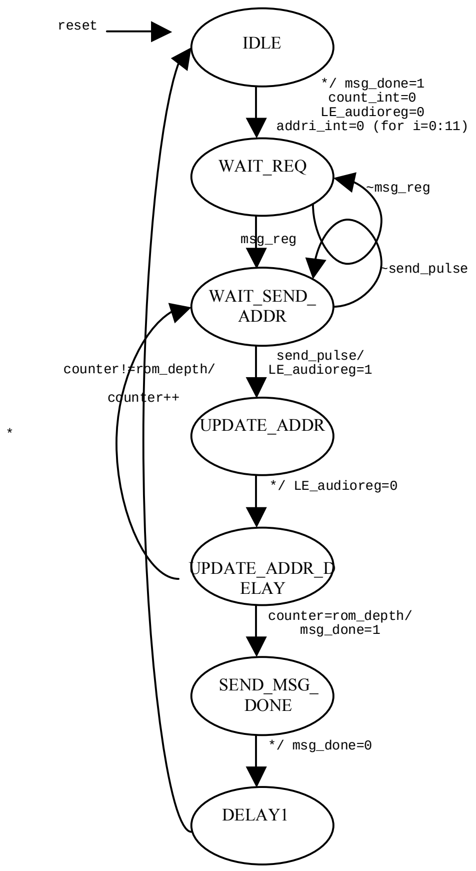


Figure 5: Audio FSM Transition Diagram

In state UPDATE_ADDR_DELAY, AUDIO FSM checks if the last address of the ROM has been read, i.e. if the last sample in the ROM is taken, by comparing the address to the counter. (The depth of each ROM is different because of the variation in message length). If the ROM is finished, AUDIO FSM moves to the next state, SEND_MSG_DONE in which it pulses signal `msg_done` high to tell CONTROL UNIT that the requested message has been played. Otherwise, AUDIO FSM goes back to to WAIT_SEND_ADDR to read the next sample in the ROM until it finishes the last sample.

3.2.3 AC97 Controller and Codec LM4550 Configuration

The codec chip and AC97 link interface is operated by its own clock, i.e. ac97_bit_clock at 12.288 MHz. The interface between the FPGA and the codec chip is in AC'97 format. AC97 CONTROLLER on the FPGA communicates with the codec chip serially, i.e. data to and from the codec is sent one bit for every cycle of ac97_bit_clock. All the properties of the codec chip: input/output volume, ADC sources, and PCM DAC rate are specified by setting the values of the corresponding registers in the chip. The path of the audio signal in the playback mode is shown by the blue dashed arrow the codec diagram.

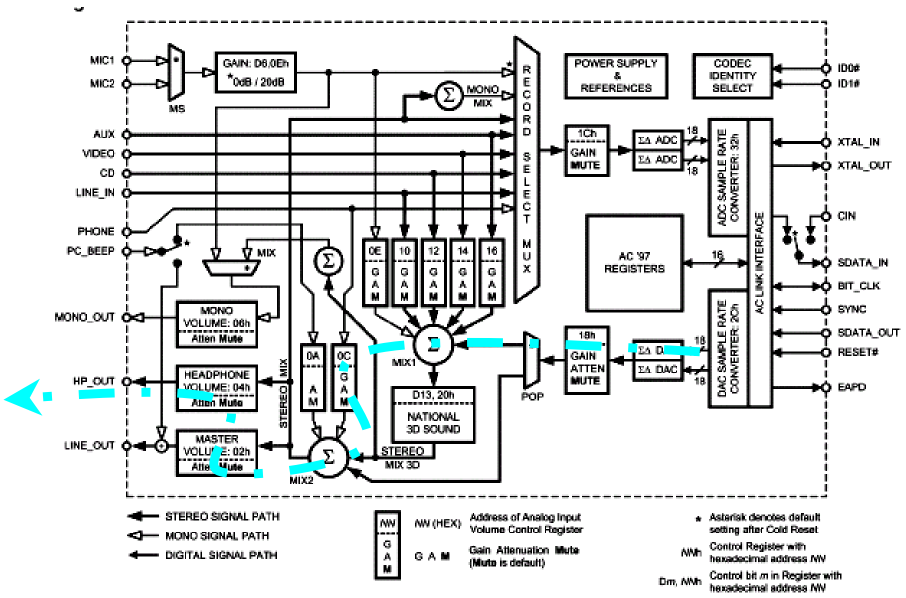

MIC1
MIC2
GAIN: D6,0Eh *0dB / 20dB
MS
MONO MIX
RECORD SELECT MUX
POWER SUPPLY & REFERENCES
CODEC IDENTITY SELECT
ID0#
ID1#
AUX
VIDEO
CD
LINE_IN
1Ch GAIN MUTE
ΣΔ ADC
ΣΔ ADC
18
ADC SAMPLE RATE CONVERTER: 32h
XTAL_IN
XTAL_OUT
CIN
PHONE
PC_BEEP
MIX
0E
10
12
14
16
GAM
AC '97 REGISTERS
16
AC LINK INTERFACE
SDATA_IN
BIT_CLK
SYNC
SDATA_OUT
RESET#
EAPD
MONO_OUT
MONO VOLUME: 06h Atten Mute
HP_OUT
HEADPHONE VOLUME: 04h Atten Mute
STEREO MIX
0A
0C
MIX1
POP
18h GAIN ATTEN MUTE
ΣΔ DAC
ΣΔ DAC
DAC SAMPLE RATE CONVERTER: 2Ch
D13, 20h NATIONAL 3D SOUND
LINE_OUT
MASTER VOLUME: 02h Atten Mute
MIX2
STEREO MIX 3D
STEREO SIGNAL PATH
MONO SIGNAL PATH
DIGITAL SIGNAL PATH
NN (HEX) Address of Analog Input Volume Control Register
G A M Gain Attenuation Mute (Mute is default)
* Asterisk denotes default setting after Cold Reset
NNh Control Register with hexadecimal address NN
Dm, NNh Control bit m in Register with hexadecimal address NN

Figure 6: Codec LM4550 Chip (taken from National Semiconductor datasheet)

In every cycle of ac97_bit_clock, one-bit `sdata_in` is sent from the codec chip to AC97 CONTROLLER, and one-bit `sdata_out` is sent from AC97 CONTROLLER to the codec chip.

The communication between AC97 CONTROLLER and the codec chip is specified in frames. Each frame consisting of 256 bits is started by a high `sync` pulse. In a frame, a register value in the codec chip can be set by declaring which slots are valid in the tag field and declaring appropriate values for each bit in the corresponding slots.

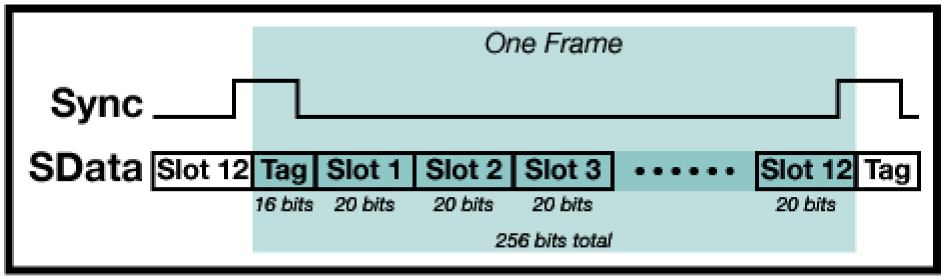


Figure 7: AC'97 Frames (taken from Nathan Ickes' Audio Tutorial)

In implementing the project, configuring the codec chip takes a substantial amount of time. There are some subtle issues that are not documented in the datasheet. For example, the signal `sdata_out` and `sync` must be low during the reset period of the codec, i.e. when `audio_reset_b` is low.

3.2.4 ROMs and Message Recording

We use Coregen in the Xilinx ISE to generate the ROMs. All ROMs are made input-registered so that there is no glitch in the address input.

Voice messages are first recorded in .wav format. Then, each message is digitized in MATLAB at sampling rate 8 kHz. (Please refer to the MATLAB code in the appendix for details). The result of the digitization is in decimal between -1 and 1. The digitized data is multiplied by 128 and then converted to twos-complement format since the codec chip expects the audio input to it in this format. Then, the ROMs are initialized with the audio data in twos-complement format in coe files.

3.2.5 Bias Shifting Circuit

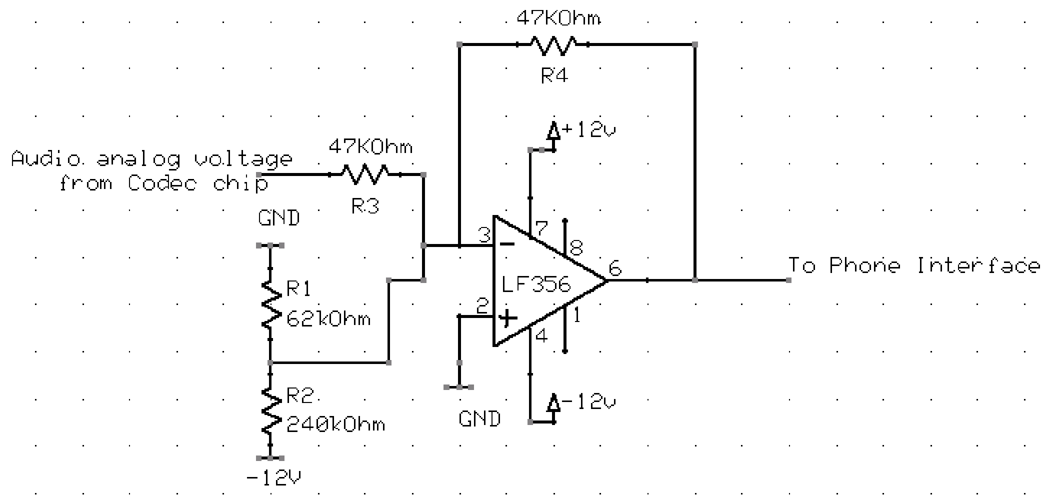


Figure 8: Op-Amp Adder with -2V DC and Audio Analog Voltage from Codec as the inputs

The phone interface requires that the audio signal input be biased at 2V, but the audio output from the codec chip is centered at 0V. Therefore, it is crucial to shift the audio signal bias by the adder shown in the figure. The adder has a gain of 1.

The adder takes two inputs at the negative terminal. The first input is the audio analog voltage from the codec chip. The other input is a DC voltage at -2V. Since the configuration is negative feedback, the output is the sum of the flipped two inputs. So the audio output to the phone interface is centered at 2V.

3.3 Phone/PIN Look-Up Table

To be able to use the system, a user must register her phone number with an operator in person. The user then is manually assigned a unique PIN number, and the PIN and phone number are stored as a pair in a look-up table.

Every time the user calls the system, she must enter in her PIN number before her requests can be accepted. This feature prevents people from abusing the system.

The phone/PIN look-up table consists of two parts: a content-addressable memory (CAM) containing 16-bit PIN numbers, and a ROM containing 20-bit phone numbers. Both were created with Xilinx Core Generator.

In a lookup operation, the PIN is passed as input to the CAM. Two clock cycles later, the CAM outputs the address of the location containing the matching PIN number. This address is sent as the input to the ROM, and after one clock cycle the corresponding phone number is valid at the ROM output.

3.4 Request Memory

3.4.1 Overview

The *request memory unit* performs the storage, timing, and retrieval of wake-up call requests.

Given a `store_ctrl` signal, a phone number, and the wake-up time, this unit stores the request in memory until it is ready to retrieved.

When the system time matches the time of the next upcoming request, a `request_pending` signal is sent along with the corresponding phone number. These signals remain valid until they are reset by the control unit.

Given a `cancel_ctrl` signal and a phone number, this unit cancels the next upcoming request that matches that particular phone number in memory. The next upcoming wake-up call request is the highest priority request for that phone number.

3.4.2 Design

The overall structure of the *request memory* unit is shown in Figure 9.

The *memory controller* is the major FSM of this unit, performing store operations, cancel operations, and the timing of wake-up call requests on the RAM.

The *shifting* unit is a minor FSM for the memory controller. It shifts multiple rows up or down in the RAM. A multiplexer determines whether the RAM inputs are set by this unit or the memory controller.

The *time* unit keeps track of the current system time. The event compare unit compares two times to see which comes first or if they are equal.

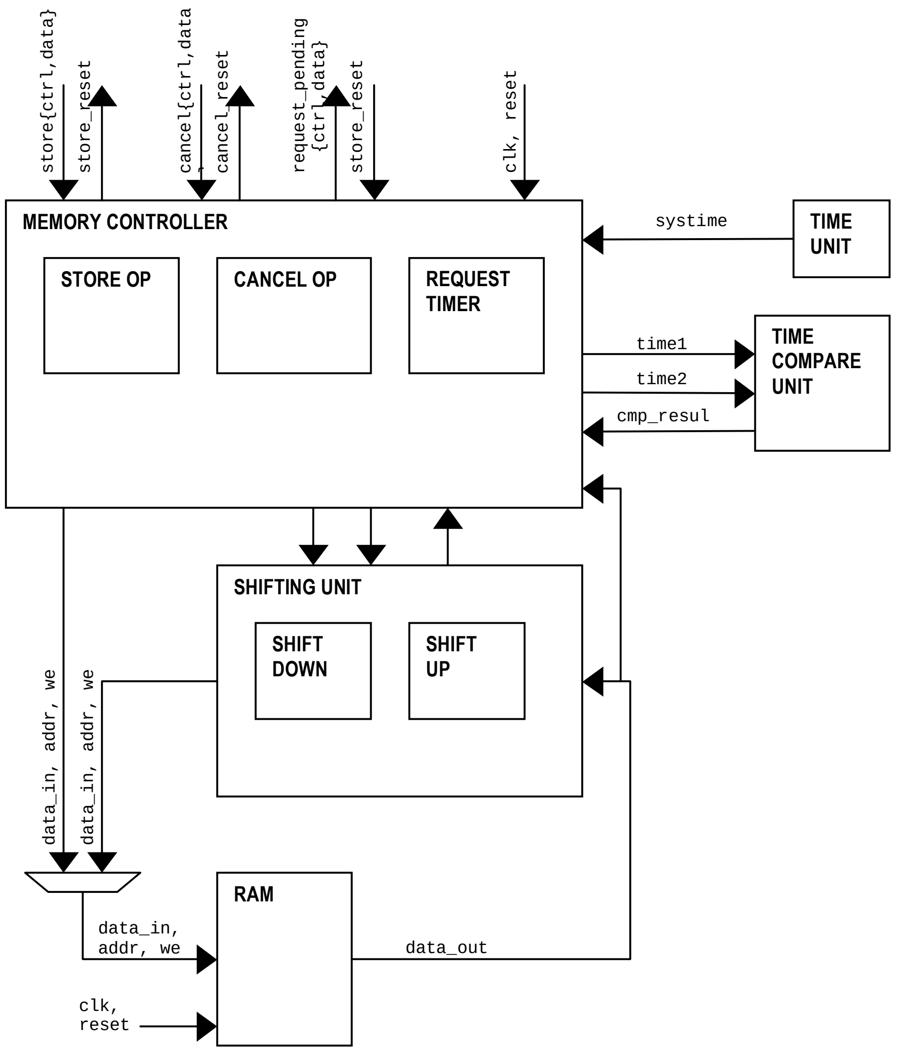


Figure 9: Request Memory Unit Block Diagram

3.4.2.1 Store Operation

After a user requests a wake-up call, the phone interface outputs data in the form of a time and a 5-digit MIT phone number. The *request memory* unit receives this data along with a `store_ctrl` signal from the main control unit.

The *request memory* unit stores this data in a RAM, sorted by time so that the next upcoming wake-up call is in the first memory location.

A simplified FSM for this operation is shown in Figure 10.

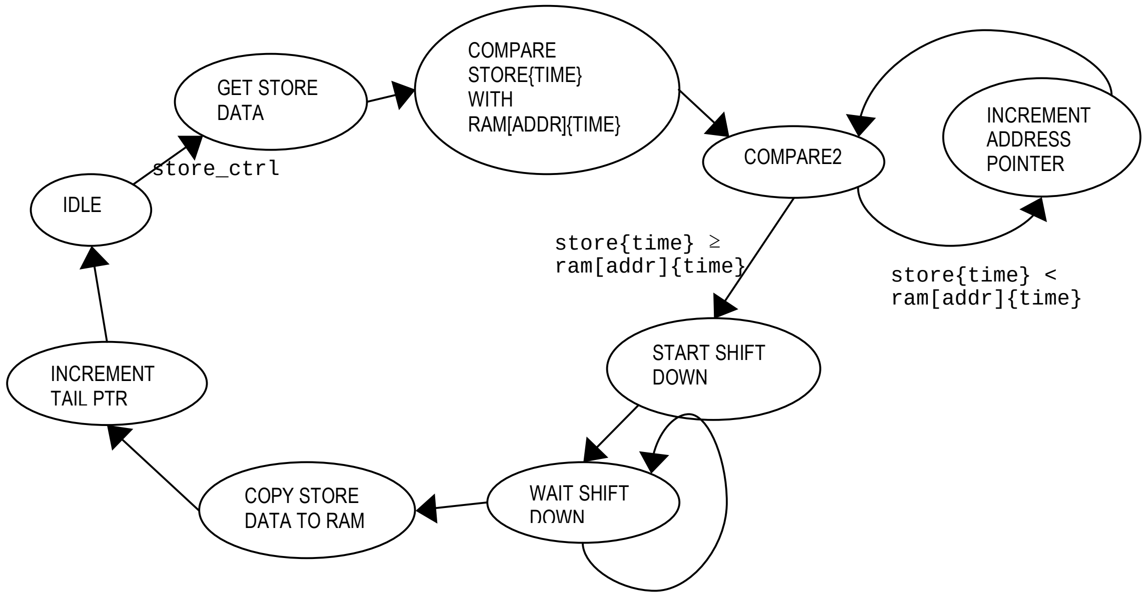
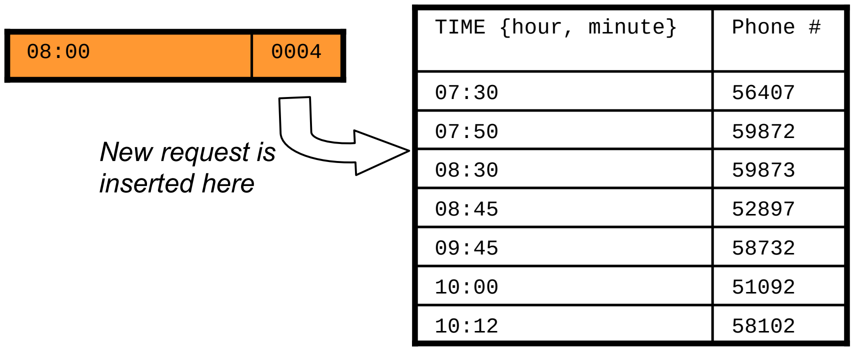

IDLE
store_ctrl
GET STORE DATA
COMPARE STORE{TIME} WITH RAM[ADDR]{TIME}
COMPARE2
INCREMENT ADDRESS POINTER
store{time} < ram[addr]{time}
store{time} ≥ ram[addr]{time}
START SHIFT DOWN
WAIT SHIFT DOWN
COPY STORE DATA TO RAM
INCREMENT TAIL PTR

Figure 10: Memory Controller, Store FSM

Note that in the 'compare' states, the *time compare* unit is used to compare the two time values. Separating this comparison makes it more extensible, because the month, day, and day of week can be added to the event compare unit without requiring major changes to the *memory controller* code.

This FSM inserts the new request in the appropriate location, sorted by time, as shown in Figure 11.



TIME {hour, minute}	Phone #
07:30	56407
07:50	59872
08:30	59873
08:45	52897
09:45	58732
10:00	51092
10:12	58102

Figure 11: What Occurs During a Store Operation.

3.4.2.2 Request Timer

At the start of every minute, the *request memory* unit looks at the request data in the first memory location. The first entry in the RAM is the next upcoming wake-up call request to be processed. The FSM for this operation is shown in Figure 12.

If the time of that request matches the current system time, a `request_pending` message is sent to the control unit, along with the phone number. The topmost request data is deleted from the RAM, and the remaining requests are shifted up in memory.

Once the control unit sends a `request_reset` pulse, the process repeats for every new first entry in RAM until there are no more requests. Then the request timer operation is inactive until the start of the next minute.

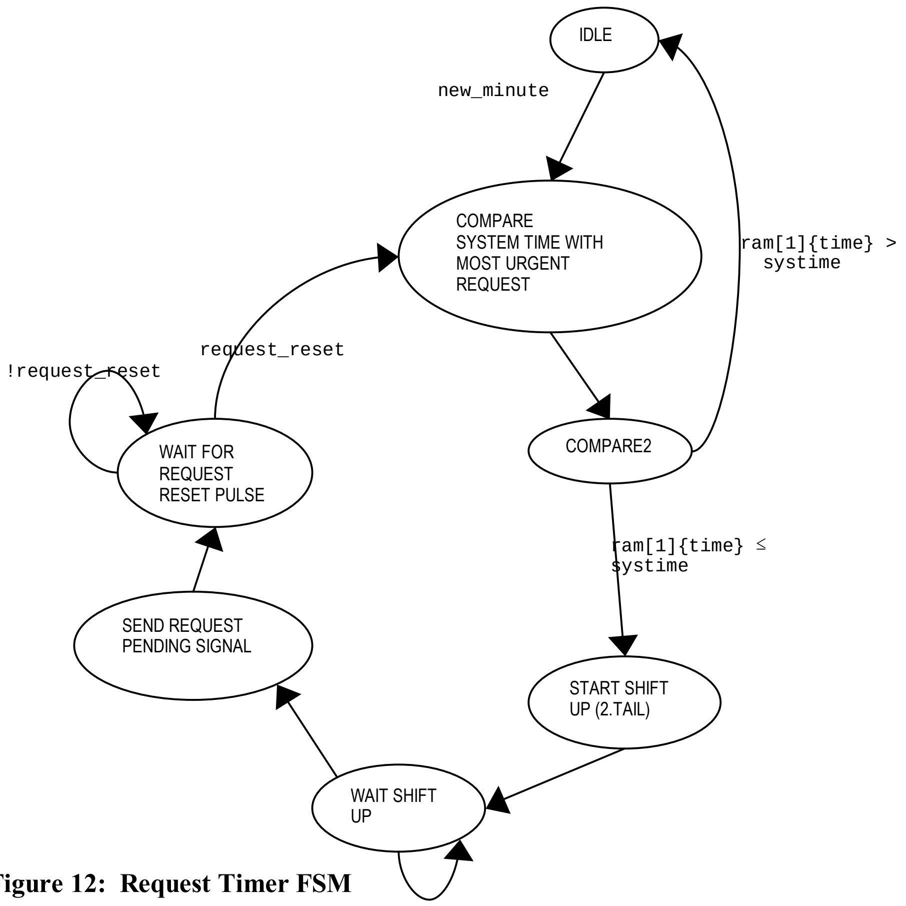


Figure 12: Request Timer FSM

3.4.2.3 Cancel Operation

To start a 'cancel' operation, the control unit sends the *request memory* unit a `cancel_ctrl` signal along with the user's phone number. The user's next upcoming request is deleted from memory.

The FSM (shown in Figure 13) is similar to that of the store operation. The difference is that the user's phone number is compared with each memory entry's phone number until there is a match, and then all the rows following the matching row are shifted up by one address.

3.4.2.4 Shifting Unit

The store operation, cancel operation, and the request timer require multiple rows of memory to be shifted up or down. A separate *shifting* unit was created to handle these shifts, simplifying the memory controller's operation and allowing code to be reused.

A `shiftdown` operation shifts rows `a` to `b` down by 1 address. A `shiftup` operation shifts rows `a` to `b` up by 1 address. Both operations output a done signal when they are finished. Because the *shifting* unit has no knowledge of the memory controller's intended operation, it does not increment or decrement the memory controller's tail pointer.

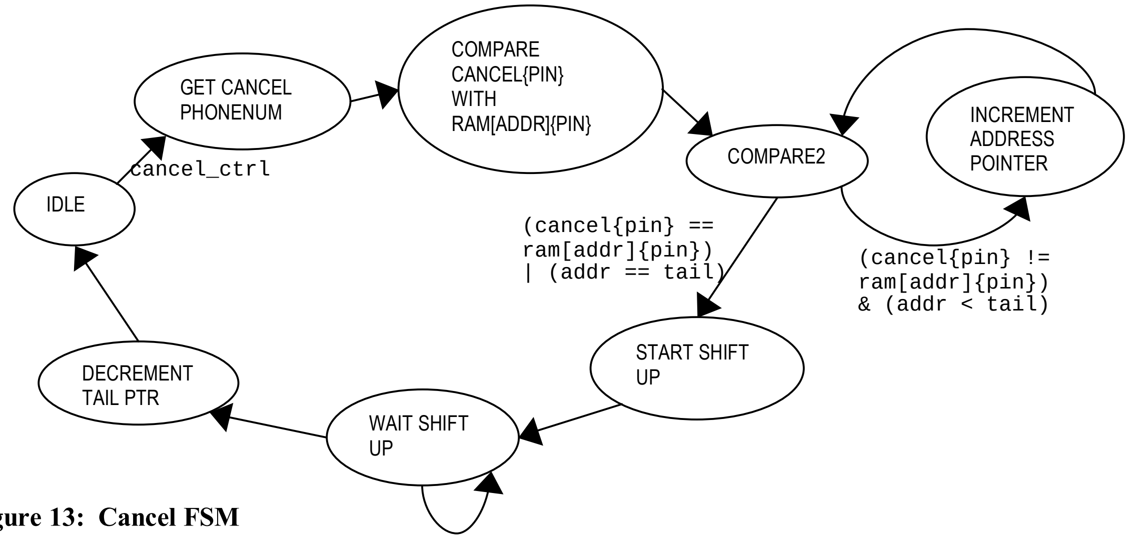

GET CANCEL PHONENUM
COMPARE CANCEL{PIN} WITH RAM[ADDR]{PIN}
COMPARE2
INCREMENT ADDRESS POINTER
IDLE
cancel_ctrl
(cancel{pin} == ram[addr]{pin}) | (addr == tail)
(cancel{pin} != ram[addr]{pin}) & (addr < tail)
START SHIFT UP
DECREMENT TAIL PTR
WAIT SHIFT UP

Figure 13: Cancel FSM

3.4.2.5 Time Unit

The *time* unit keeps track of the current system time. It divides the 27 MHz clock down behaviorally to create a signal that pulses at 1 Hz.

Every 60 seconds, minute is incremented. Every 60 minutes, `hour` is incremented and `minute` is reset to 0. Every 24 hours, `hour` is reset to 0. As with other time values in this implementation, the system time has two parameters: `hour[4:0]` and `minute[5:0]`.

The time unit also tracks the month, day, and the day of the week. These parameters were not used in the basic system, but they are useful for future versions of the system.

3.4.2.6 Time Compare Unit

Given `time1` and `time2`, the time compare unit returns 0 if `time1` is before `time2`, 1 if the times are equal, and 2 if `time1` is after `time2`.

3.4.2.7 RAM

The RAM, shown in Figure 14, is a 31 by 256 single-port block memory. It was created with the Xilinx Core Generator and has a latency of one clock cycle.

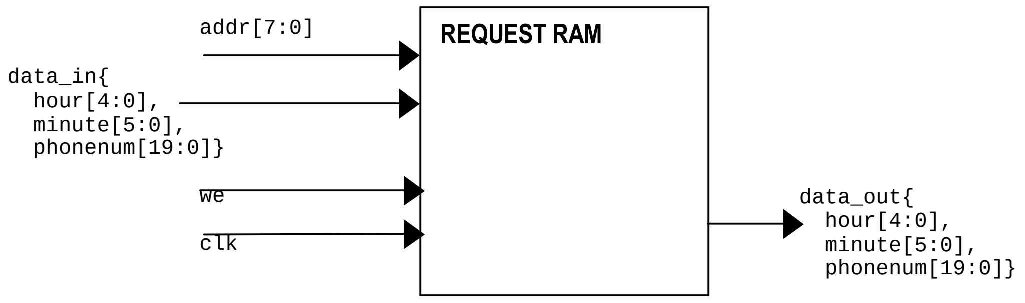

addr[7:0]
REQUEST RAM
data_in{ hour[4:0], minute[5:0], phonenum[19:0]}
we
clk
data_out{ hour[4:0], minute[5:0], phonenum[19:0]}

Figure 14: RAM for storing wake-up call requests information

Each row contains a wake-up call request with the parameters `time(hour[4:0], minute[5:0]}`, and `phonenum[19:0]`. The requests are sorted by time, starting with the earliest request in address 1.

Address 0 is unused, so that the *memory controller* unit can set its tail pointer at 0 when there are no requests stored in memory.

4 Methodology and Testing

Our group spent a lot of time in planning the system and designing on paper. Our design process was as follows: define system components, delegate system components, design interfaces between control unit and components, individual design of components, implement and test in hardware components separately,
integrate components one at a time. Control logic was designed, coded and simulated entirely beforehand (which showed possible problems with interfaces, missing logic, etc), but in implementation added piece by piece as each component was integrated.

Our testing philosophy is that we test each module, one by one, to ensure that a module works on its own. Then, we group modules into subcomponents and test each subcomponent separately to ensure that each subsystem works.

4.1 Phone Line Interface Unit

First the MH88437 circuit was tested by connecting it to a phone line, calling the phone number, asserting control signals manually and probing the chip's outputs. After ensuring that the chip was functional, the MT8889 was tested. Since the MT8889 control signals were significantly more complicated, its controller was written and tested for just receiving one digit before moving on to multiple digits.

4.2 Audio Unit

For the audio unit, each module is verified in the simulation on its own. Then, three subsystems are created from those modules: audio signal retrieving block, AC97 CONTROLLER with the codec, and the bias shift circuit.

In testing the audio retrieving block, we first verify a set consisting of AUDIO FSM and AC97 REG with one ROM. After verifying that it works on the lab kit, ten more ROMs are added with some corresponding modification in other modules.

In testing AC97 CONTROLLER and the codec chip, the analog loopback testing of the chip is the first step. Then, the digital loopthrough test is performed. Finally, the codec chip is customized to accommodate the playback from the ROMs.

In testing the bias shift module, the op-amp circuit as shown in the figure is built. It is tested by an output from a function generate to verify that it adds the two inputs correctly.

After verifying that these three subsystems work correctly in hardware, they are integrated to be the audio unit.

4.3 Request Memory Unit

All components of the *Request Memory* unit were implemented and simulated using 'Simulate Behavioral Model' in the Xilinx ISE. This was a mistake because the unit did not behave as expected once it was implemented in hardware.

The RAM was then tested individually with a logic analyzer, using a simple test module that would output the contents of the first 16 memory locations. For testing purposes, the RAM contents were initialized to contain data at startup. Because of various issues with the Core Generator, this part of the process took over 2 weeks to complete.

Once the RAM was functioning properly in hardware, each remaining part was tested individually with the logic analyzer. At the time of the deadline, the *request memory* unit was completely functioning in hardware. The logic for the interface to the control unit was also functioning properly in hardware, but time did not permit for full integration of the request memory unit into the rest of the wake-up call system.

5 Conclusion

The basic wake-up call system described in this paper meets some of the needs of the MIT community. Using just a touch-tone phone, anyone with a 5-digit MIT phone number can call the system to make a wake-up call request. Simple audio menus guide the user to enter numbers at the appropriate times, to store new requests or cancel a request. The system will then call them back at the times that they have requested.

The system has been designed so that new features can be added without requiring major modifications to the system structure. By adding a new audio greeting, receiving additional digits, modifying the time compare unit, and increasing the size of the request memory, a 365-day wakeup call system can be implemented. With a few more modifications, users could even request a wake-up call that repeats daily or on certain days of the week.

Another modification would be to allow users to store messages for themselves. To implement this feature, large amounts of external memory must be added to the system, and there is considerable work involved in interfacing and managing the additional memory.

Finally, there are interesting possibilities with allowing users to store messages for one another. Permission lists can be implemented, allowing users to request wake-up calls for other users or for particular groups of people. A professor would be able to send wake-up calls to his students before an exam, or friends could wake up to each other's voices in the m