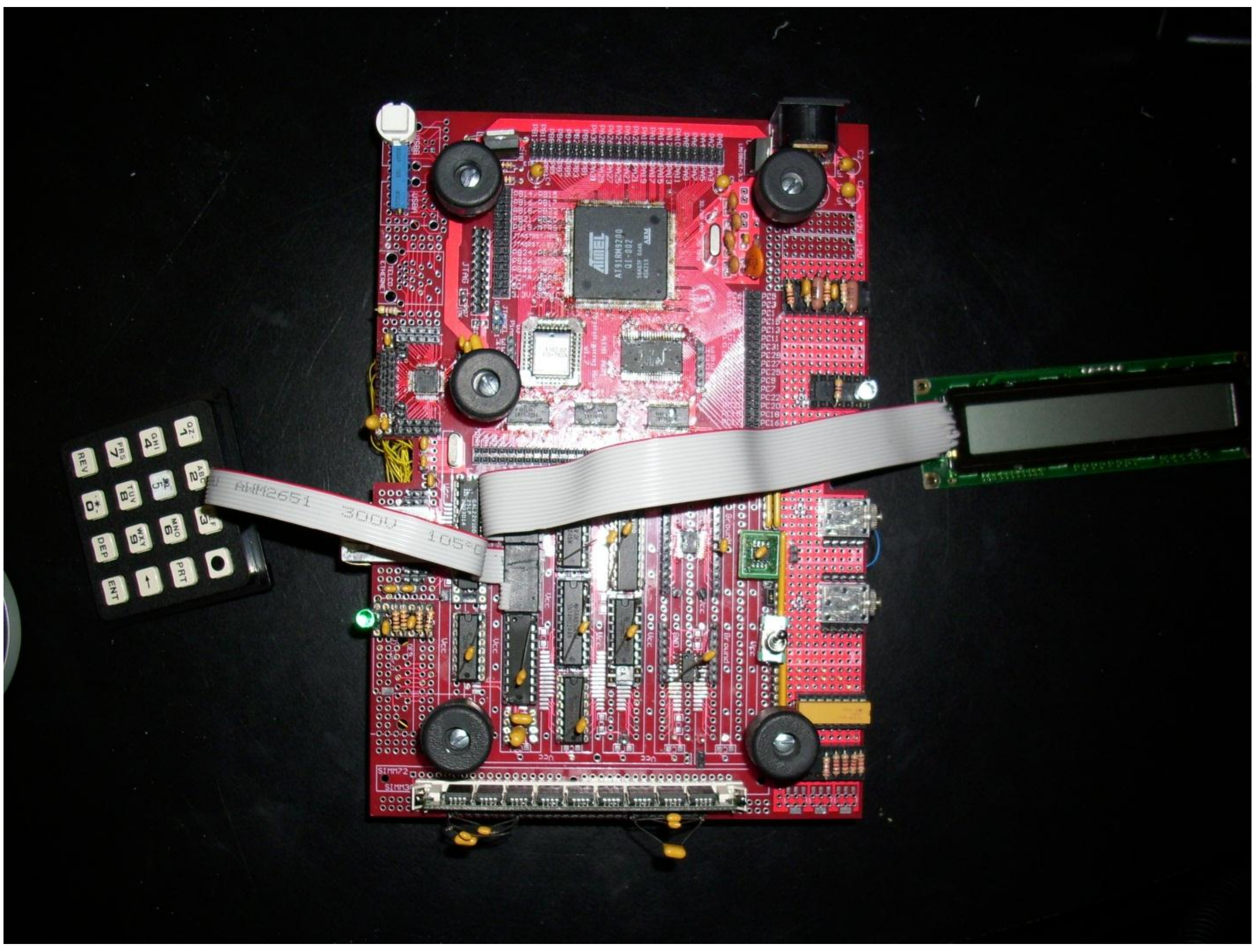
ARM VOIP Telephone Manual and System Information

Josh Fromm

Index

Basic System Description:

The system is a voice over internet protocol (VOIP) telephone that is (theoretically) capable of placing a phone call over the internet to the IP address of a compatible VOIP telephone. Due to the exposed components of the system, the VOIP telephone must be handled carefully.



Birds-eye image of VOIP telephone

User Manual:

The VOIP telephone is initialized by first supplying power through a +5, +-12 Volt transformer. The transformer is plugged in as seen in the following image.

How to attach a power cable

If power is being properly supplied, a blue LED will turn on to indicate the system is powered.

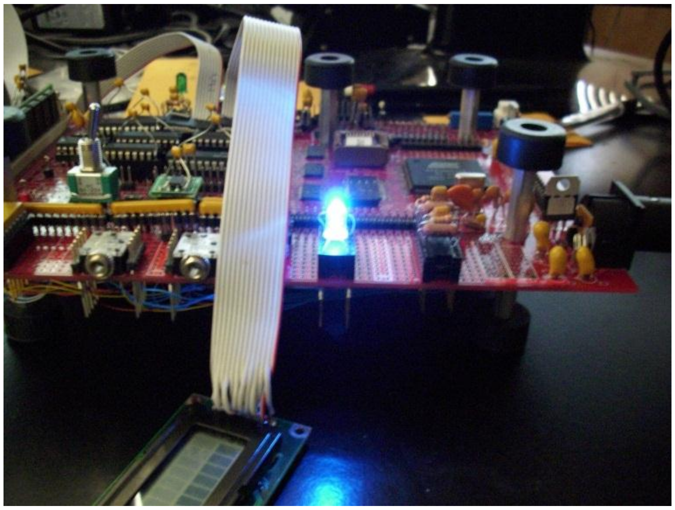


Image of power LED

Once the system is powered, code must be downloaded before the VOIP phone will function. This is done by connecting a wiggler board to JTAG connection area seen below. The wiggler board can then be connected to the parallel port of a computer which can be used to download code through a debugger such as OCD Commander.

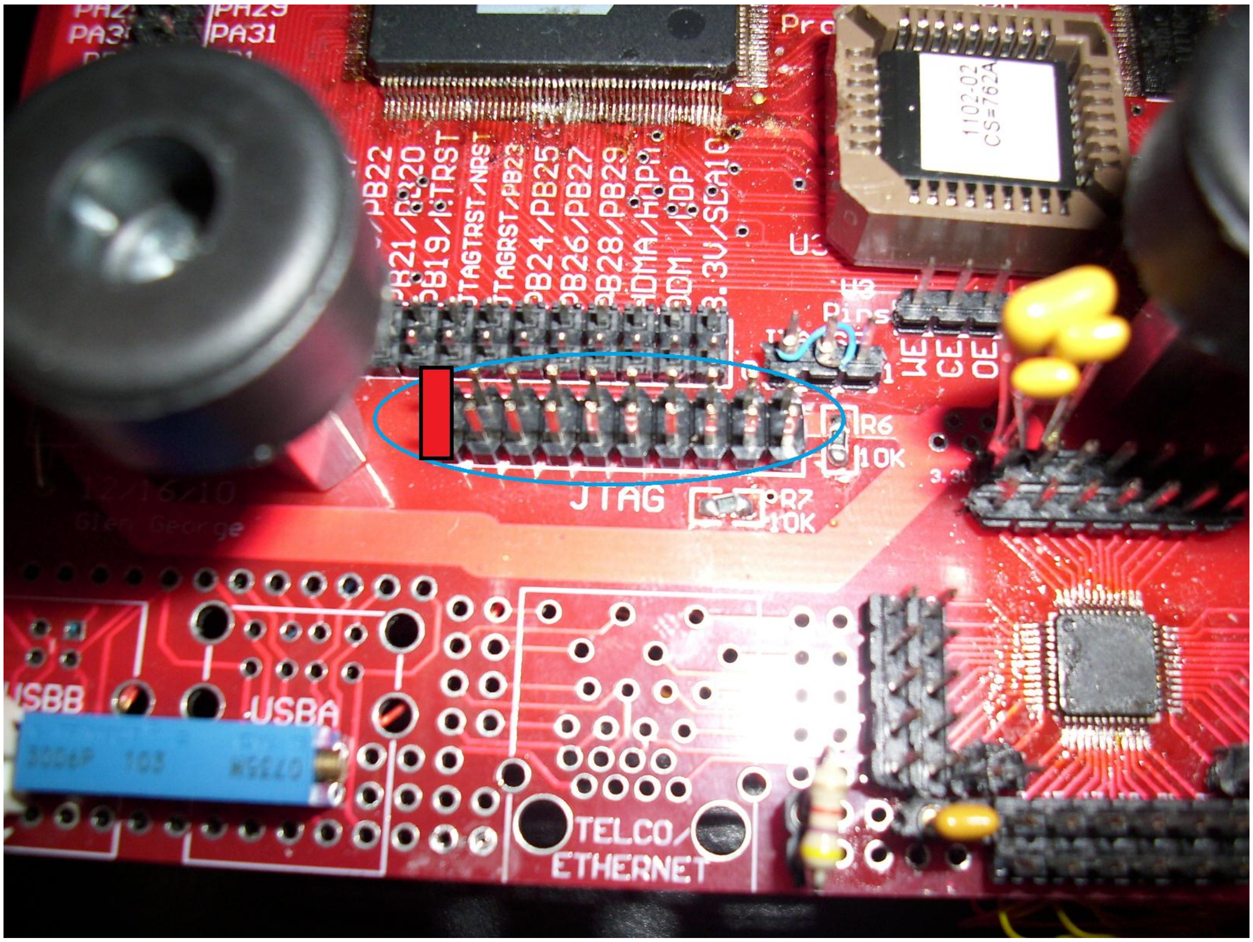


Image showing JTAG connection zone. Note that the red block indicates how to properly attach the cable (the red strip on the connection cable goes on the side with the red block).

Once code is downloaded and running (the display should show the message "Idle"), the vast majority of user interfacing is done through the 16 key keypad and LCD display of the system. The function of each key on the keypad is listed in the following table. It should be noted that the keypad has a shift key which changes the function of each of the keys. Shift is toggled on and off through the same key and the keypad will stay shifted or unshifted until the shift key is pressed again.

Unshifted Key Functions:

Key Description	Key Name	Key Function
QZ-1	One	The number 1
ABC-2	Two	The number 2
DEF-3	Three	The number 3
Black Circle	Escape	Resets any entered value to zero
GHI-4	Four	The number 4
JKL-5	Five	The number 5
MNO-6	Six	The number 6
PRT	Shift	Changes whether the keypad is shifted or not. Shift status determines what the keys do.
PRS-7	Seven	The number 7
TUV-8	Eight	The number 8
WXY-9	Nine	The number 9
Left Arrow	Backspace	Undoes the last entered number

REV	Off hook	Equivalent to picking up a standard telephone. Being off hook causes the display to show the message "Off Hook". Once off hook, a phone call can be placed by entering an IP address using the number keys. Once an IP address is entered, the call goes through once the Enter key is pressed. When a call is being received (which causes the message "Ringing" to be displayed), pressing off hook answers the phone call.
.*_-0	Zero	The number 0
DEP	On Hook	This key returns the phone from an "Off Hook" status to "Idle". Pressing this key is equivalent to hanging up a phone.
ENT	Enter	Enter is used to finalize key functions (Off Hook, Set IP, Set Subnet, Set Gateway, Memory Save, and Memory Recall).

Shifted Key Functions:

Key Description	Key Name	Key Function
QZ-1	Set IP	When pressed, Set IP allows the user to enter a series of numbers of the form xxx.xxx.xxx.xxx where any set of xxx does not exceed 255. The display will show the message “Set IP” with the currently entered IP address displayed below it. The entered value will become the phone’s new IP address (which is what other phones will enter to call the user’s phone). It should be noted that to enter the numbers, the shift key must be pressed. To exit Set IP at any time, simply press the Set IP key again. Once the desired IP address is shown on the display, pressing the enter key finalizes the function.
ABC-2	Set Subnet	Functions the same way as Set IP but instead sets the phone’s subnet. The display will show the message “Set Subnet”
DEF-3	Set Gateway	Functions the same was as Set IP but instead sets the phones Gateway. The display will show the message “Set Gateway”.

Black Circle	Memory Save	When the phone is off hook and an IP address is entered, pressing Memory Save will allow the user to save that IP address for later referencing. When pressed Memory Save requires that the user enter a number between 0 and 15 (16 addresses can be saved in total). After a number between 0 and 15 is entered, the Enter key is pressed to finalize the function. When Memory Recall is pressed and the number entered, the saved IP address will be displayed. Memory Save causes the message “Memory Save” to be displayed.
GHI-4	Memory Recall	When Memory Recall is pressed, the user can enter a number between 0 and 15 to display a previously saved IP address. Once the number is displayed, Enter must be pressed. Memory Recall causes the message “Memory Recall” to be displayed.
PRT	Shift	Shifts the keypad back to the unshifted state.
All Other Keys	Nothing	When shifted, other keys will do nothing.

Besides the display and keypad, the system has several other ways the user can interact with the system. The reset switch allows the user to reboot the system by holding down the switch for around 5 seconds. The mute switch causes both incoming and outgoing sound to be muted. The display contrast dial allows the user to control the contrast of the display. Each of the other user interface options is shown in the following picture.

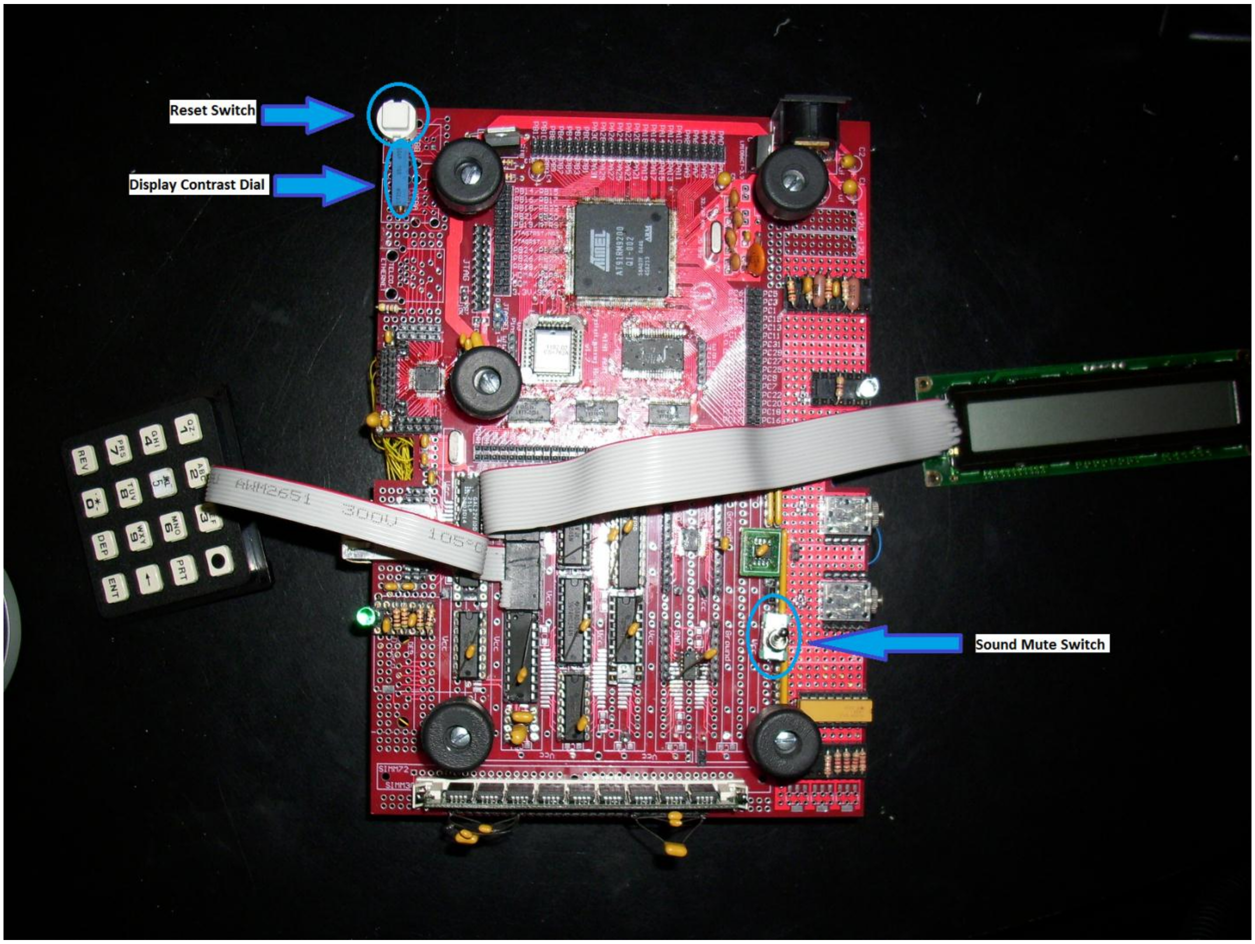
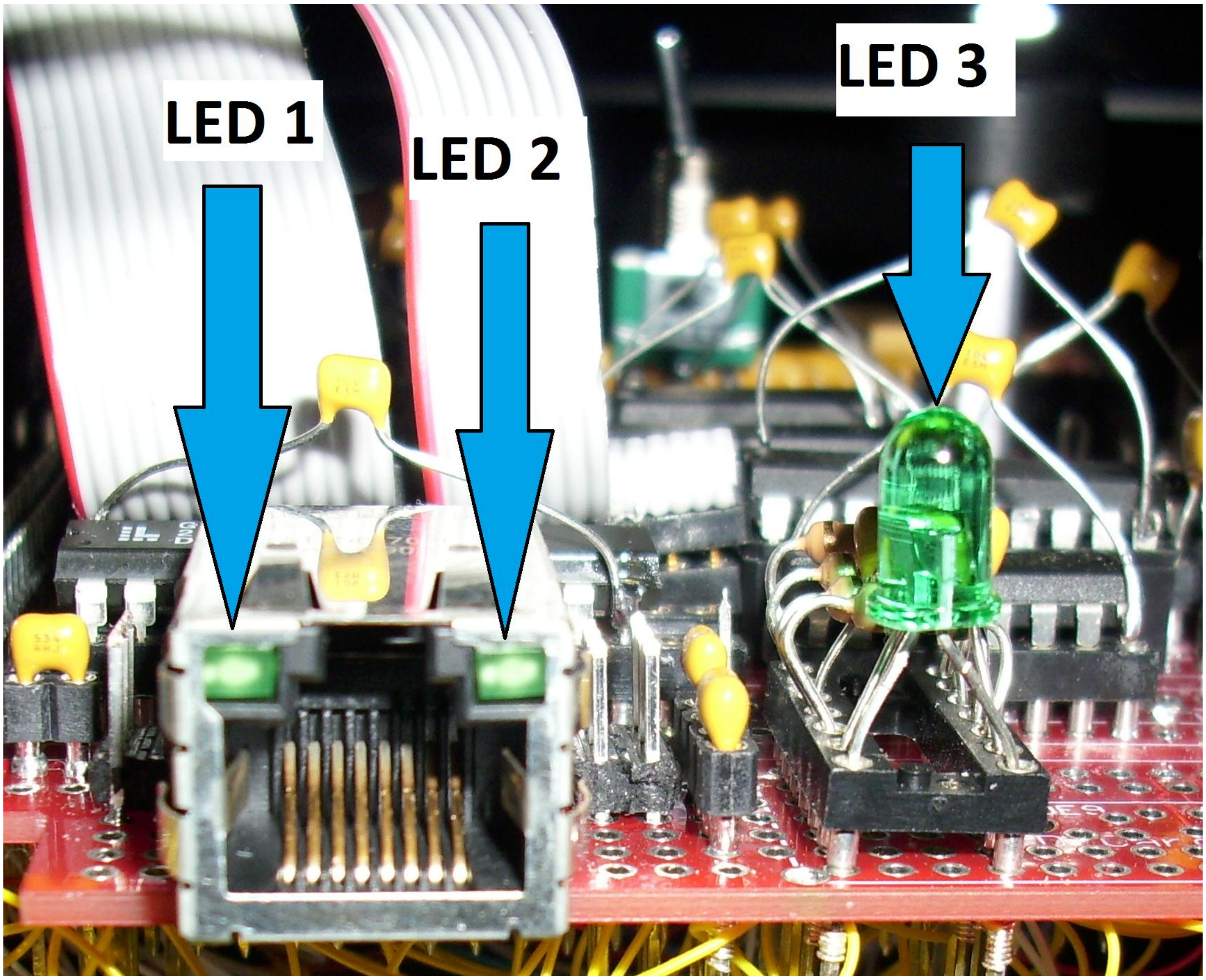
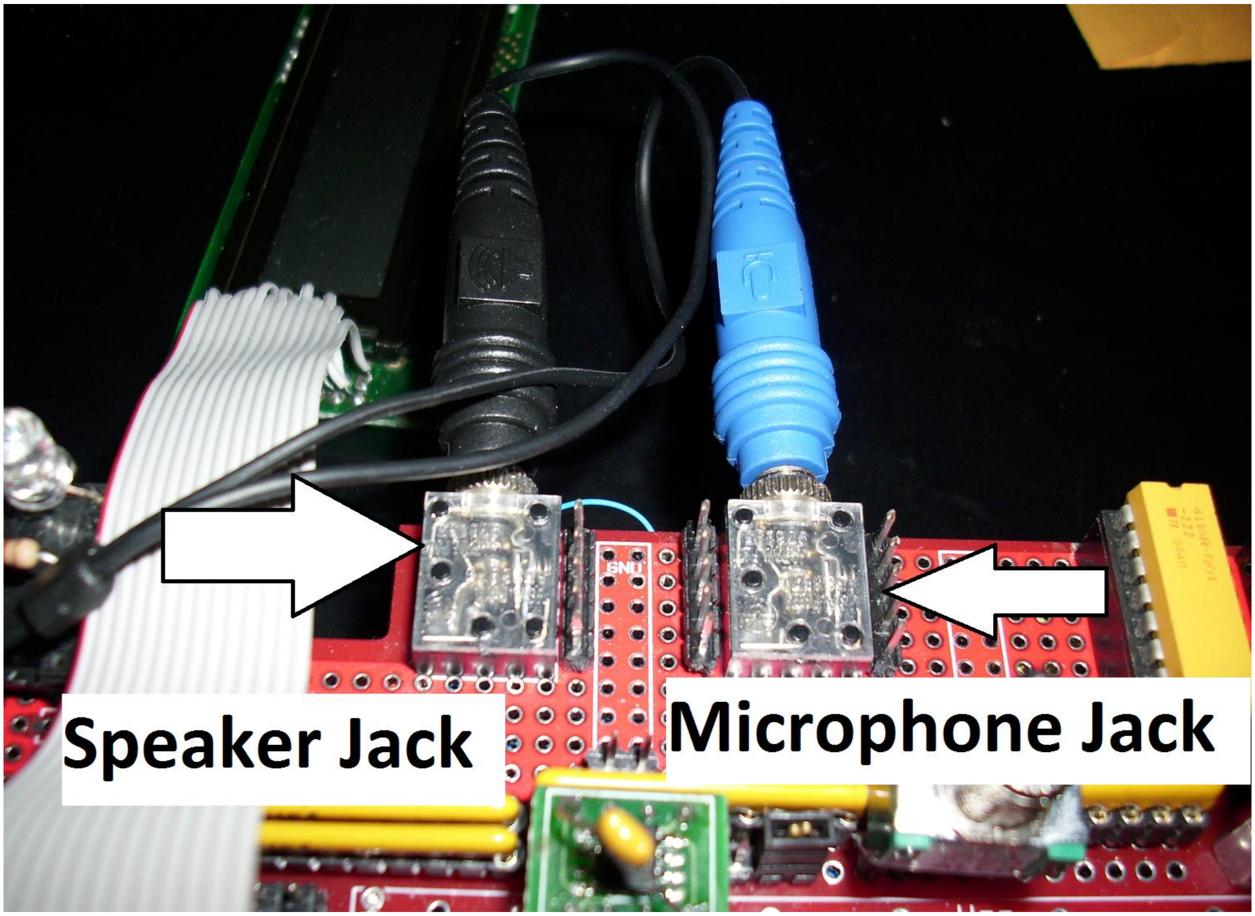


Image showing location of user interface components.

The user also can monitor the status of Ethernet connections through the LEDs on the Ethernet jack. LED 1 indicates Ethernet activity (meaning it should blink when functioning), LED 2 indicates the speed setting of the Ethernet and should be on if no Ethernet cable is plugged in and off if Ethernet is plugged in, LED 3 indicates whether a link is being made and so should be off when no cable is plugged in and on when a cable is plugged in.

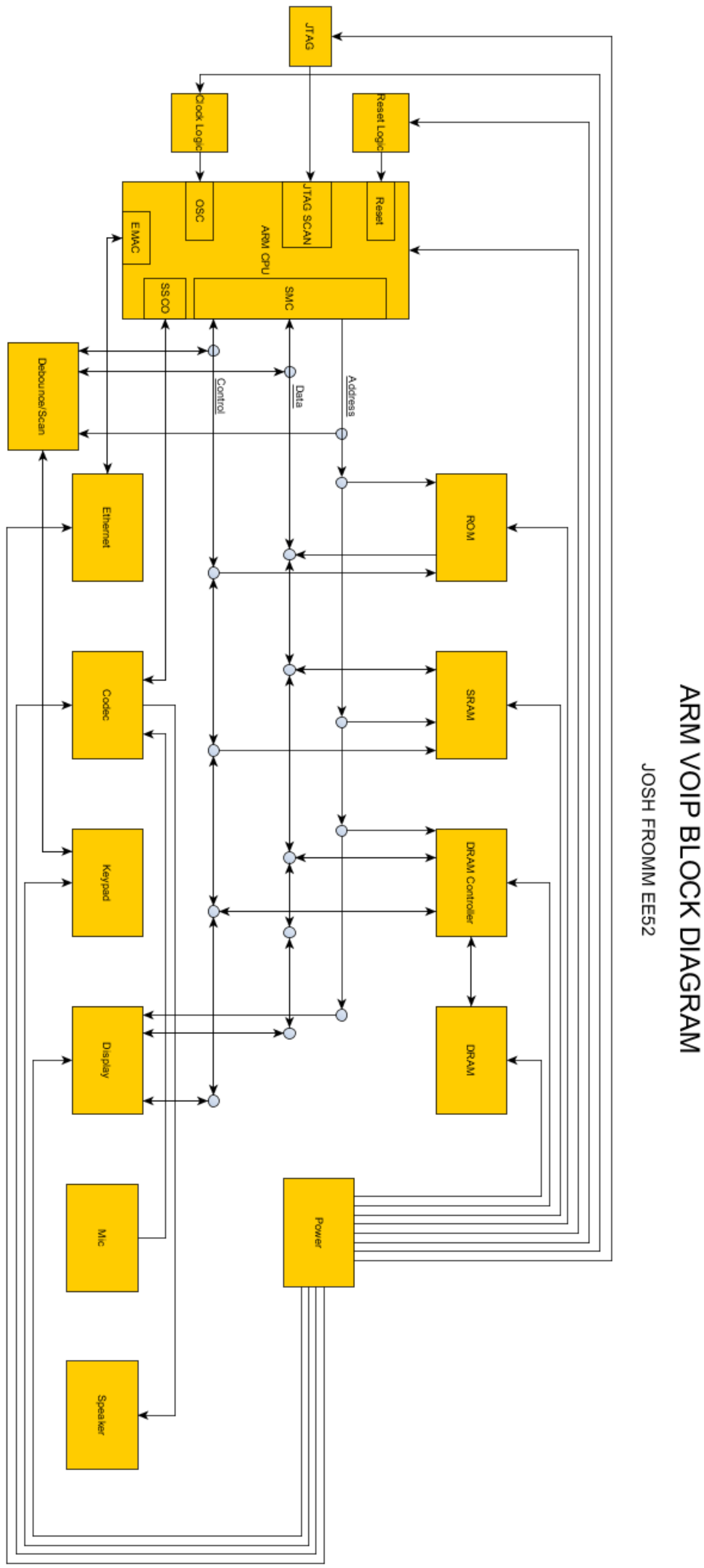


If no speaker or microphone is plugged in, the system will not function properly. Thus, the user must properly plug in both a speaker in microphone to make phone calls. This is done by plugging a speaker into the speaker jack and a microphone into the microphone jack as seen below.


Speaker Jack
Microphone Jack

Overview of System Hardware

The basic interaction of hardware in the VOIP system is outlined in the following block diagram.


ARM VOIP BLOCK DIAGRAM
JOSH FROMM EE52
JTAG
Clock Logic
Reset Logic
ARM CPU
OSC
JTAG SCAN
Reset
EMAC
SSCO
SMC
Address
Data
Control
Debounce/Scan
Ethernet
ROM
Codec
SRAM
Keypad
DRAM Controller
Display
DRAM
Mic
Power
Speaker

Explanation of Each Block:

CPU: The system uses an ARM 920T processor. The CPU is the core of the system; all other blocks interact directly with the CPU. The majority of peripherals (SRAM, ROM, DRAM, display, and keypad) interact with the CPU through the static memory controller (SMC). Other peripherals interact with more specific controllers of the CPU such as Ethernet with the CPU's EMAC controller. The CPU runs code located on the SRAM that dictates how the CPU should interact with other peripherals.

ROM: The read only memory of the system is used to store the code that runs the CPU runs off of. On boot up, assuming a functional boot up sequence, the code located on the ROM is transferred to the SRAM, where the system will read from.

SRAM: The static random access memory of the system is used to store the code used to run the system, the variables accessed by the code, and buffers used to store information (such as incoming and outgoing Ethernet data).

DRAM Controller: The dynamic random access memory controller is used to interact with the system's DRAM. The DRAM controller is needed for the system to properly access memory in DRAM.

DRAM: The system's DRAM is used to store incoming and outgoing audio buffers (which are input and output from the codec).

Keypad: The keypad is used as user interface with the system. The keypad interacts only with a debouncer/scanner chip that then interacts with the CPU.

Display: The display is used to provide information to the user.

Ethernet: The Ethernet block is used to take in and output data over the internet. It interacts with the EMAC controller of the CPU.

Codec: The codec provides sound control for the system. The codec interacts directly with the microphone (input) and speaker (output) of the system and relays information to/from the CPU.

Clock Logic: Clock logic is used to allow a stable phase locked loop running at 150 megahertz. The clock logic interacts with the oscillator controller of the CPU.

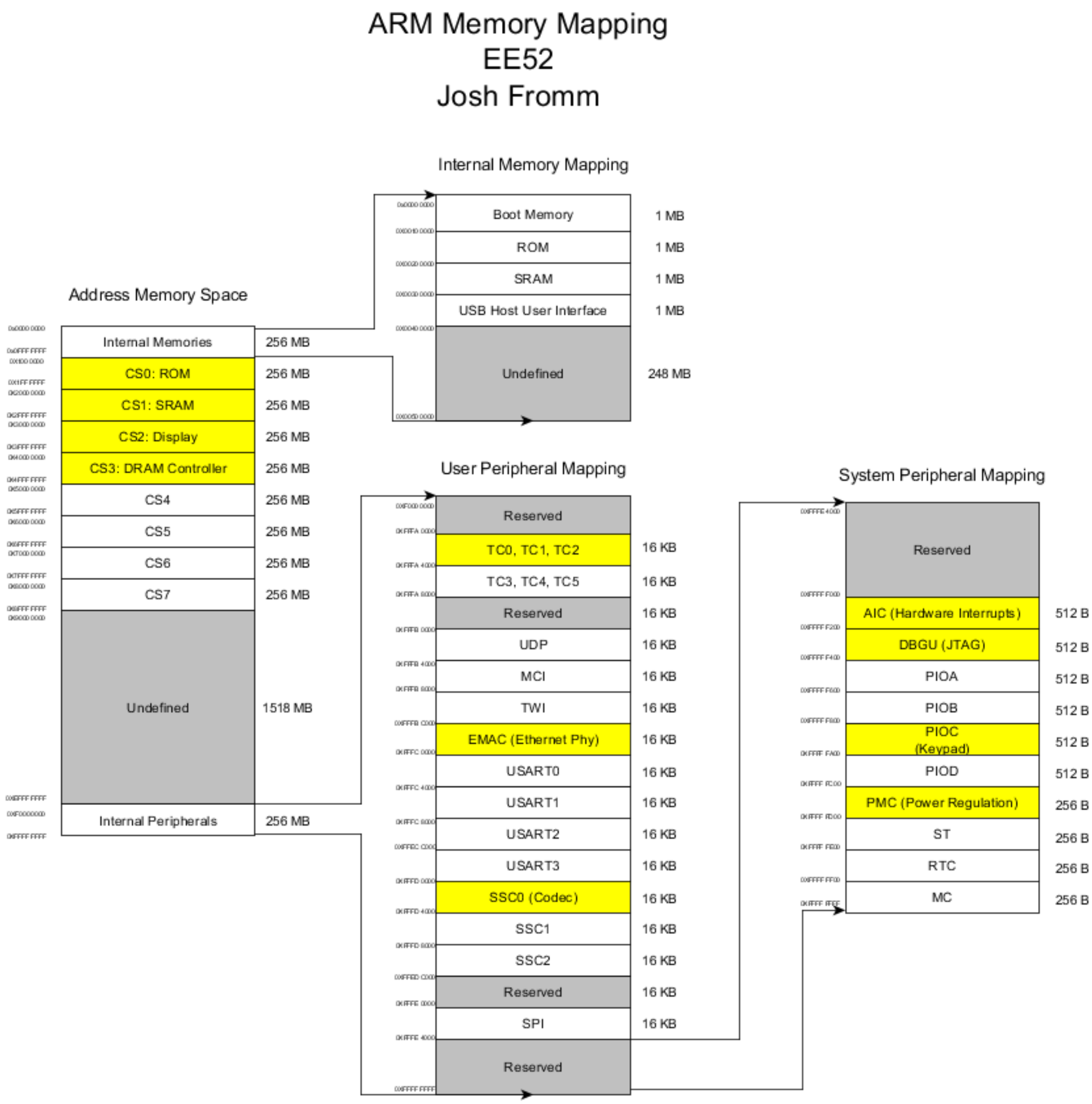
JTAG: The JTAG block allows debuggers to interface with the CPU.

Reset Logic: Reset logic is used to allow the user to reset the system by holding down the reset switch.

Hardware Manual:

This section addresses each of the blocks noted in the previous section with gratuitous detail.

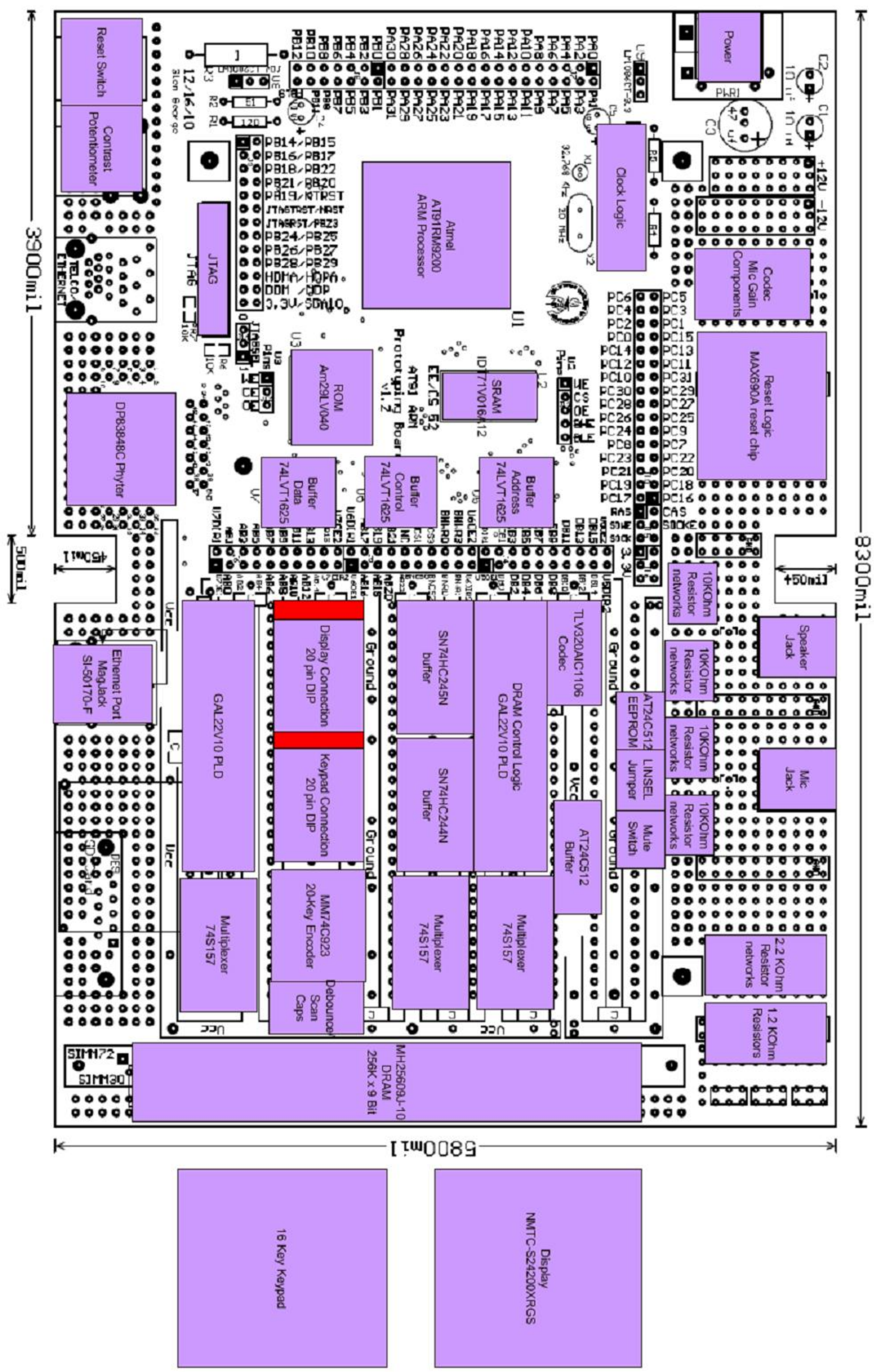
The system's memory map, which shows which chip selects are used to control peripherals, can be seen in the following image. Note that highlighted mappings are those that the VOIP system directly works with.



System Memory Mapping

The next image shows the layout of the board with exact model numbers for each component.

ARM Board Layout
EE52
Josh Fromm



CPU, SRAM, ROM, Clock Logic, and Main Buffers:

This subsection will cover the schematics and specific signals of the CPU, SRAM, ROM, Clock Logic, and buffers used in the VOIP system.

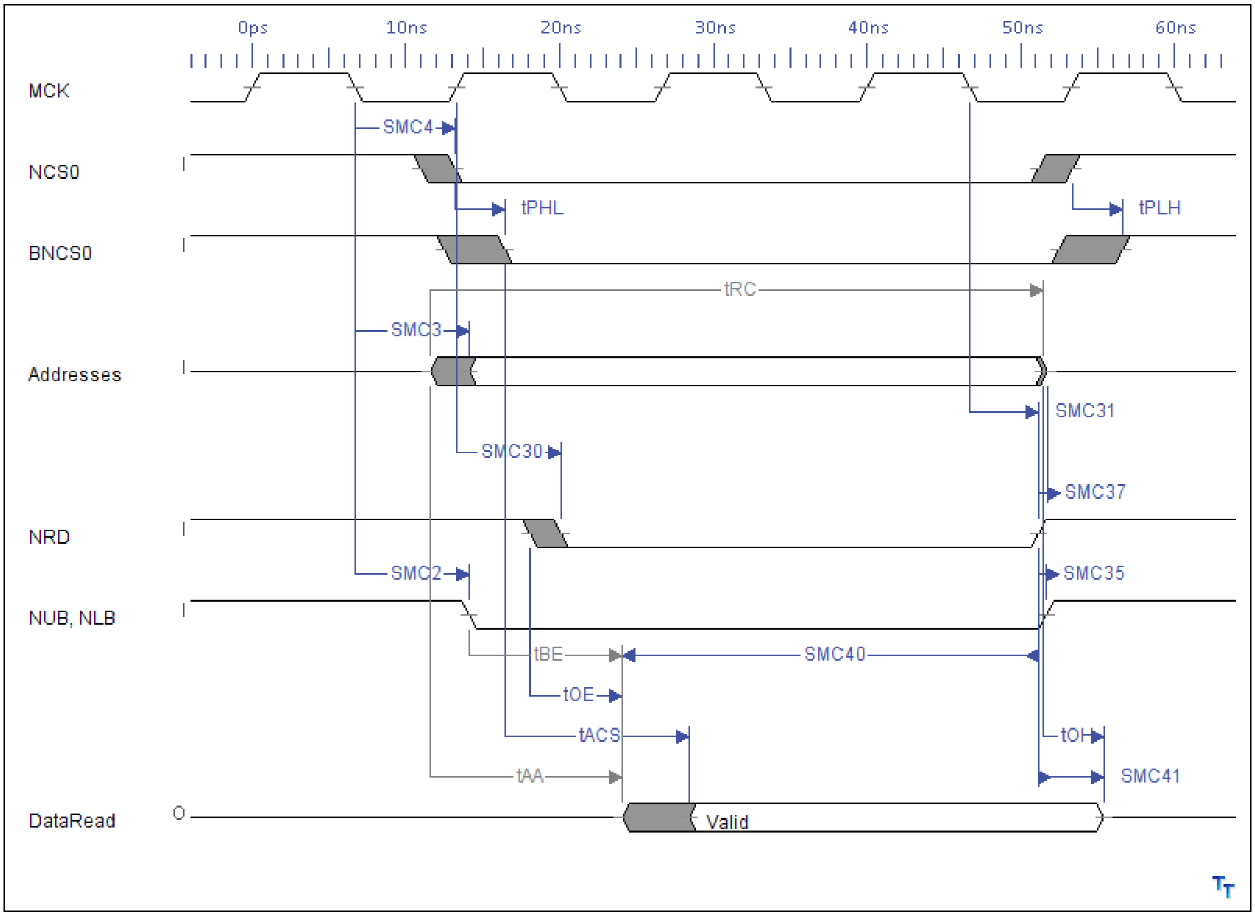
The clock logic of the system, which can be seen on page 1 of the included schematics, allows a PLL frequency of 150 megahertz to be established. This frequency is set as the processor clock speed and the master clock speed, seen in timing diagrams as MCK, is set to be 75 megahertz.

Page 1 of the included schematics (titled ARM Schematic V1) shows the schematic of the CPU, SRAM, ROM, clock logic, and buffers.

SRAM:

The SRAM used by the system is an IDT71V016SA 64K x 16-bit CMOS Static RAM chip.

The interaction between the CPU and SRAM will be focused on first. The SRAM serves as the main memory of the system and stores code, variables, and many buffers. Thus, the CPU must be able to both read and write from the SRAM. Following is a diagram of the signals required for the CPU to perform an SRAM read and an explanation of the timings involved. It should be noted that the CPU is set to perform half word (16 bit) reads with 2 wait states when interacting with the SRAM.



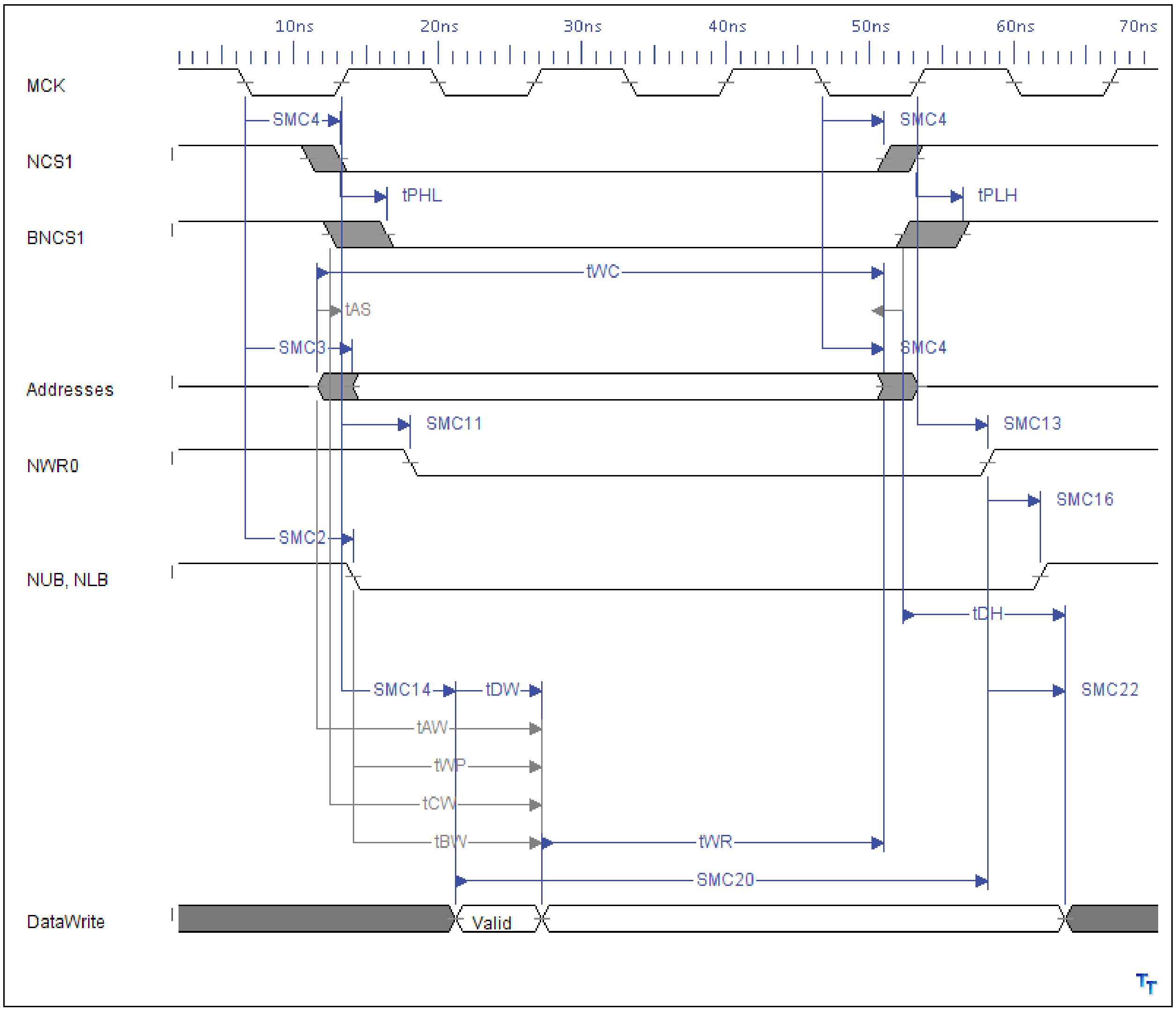
SRAM read signals and timing, where DataRead is valid data.

Signal Name	Signal Description	Signal Time (ns)
SMC 4	MCK Falling to Chip Select Change	6.5
SMC 3	MCK Falling to A1-A25 Valid	7.4
SMC 30	MCK Rising to NRD Active	7.0
SMC 2	MCK Falling to NLB/A0 Valid	7.5
tPHL	Propagation time due to buffer (high to low)	3.3
tPLH	Propagation time due to buffer (low to high)	3.3
tOE	Delay between active read and valid data	6
tACS	Access delay	12
SMC 40	Data Setup Time before NRD	7.5

	High	
tOH	Delay from invalid addresses to invalid data	4
SMC 31	MCK Falling to NRD Inactive	6.8
SMC 37	NRD High to A1-A25 Change	.3
SMC 35	NRD High to NUB Change	.5
SMC 41	Data Hold after NRD High	0

Explanation of SRAM read signals and timings.

Following is a similar explanation of SRAM write timings.



SRAM write signals and timings. Note that DataWrite is valid data.

Signal Name	Signal Description	Signal Time (ns)
SMC 11	MCK Rising to NWR Active	4.8
SMC 14	MCK Rising to D0-D15 Out Valid	7.9
tWC	Write Cycle Minimum Time	12
tDW	Data Valid to End of Write Requirement	9
tWR	Address Hold from End of Write	12
SMC 20	Data Out Valid before NWR High	27
Signal Name	Signal Description	Signal Time (ns)
SMC 13	MCK Rising to NWR Inactive	7.2
SMC 16	NWR High to NLB/A0 Change	3.7
SMC 22	Data Out Valid after NWR High	5.46

Explanation of SRAM write signals and timings.

ROM:

The system uses an AMD Am29LV040B 512K x 8-bit ROM chip.

The CPU reads from the ROM only during its boot up cycle. When booting up, the system will access all the memory in ROM and move it into SRAM. Once running, the system does not access ROM. The schematic for the ROM of the VOIP system can be found along with other schematics in this section on page 1 of the included schematics. Following is a diagram of the signals and timing of a ROM read. It should be noted that when interacting with the ROM, the CPU is set to perform byte reads with 18 wait states.

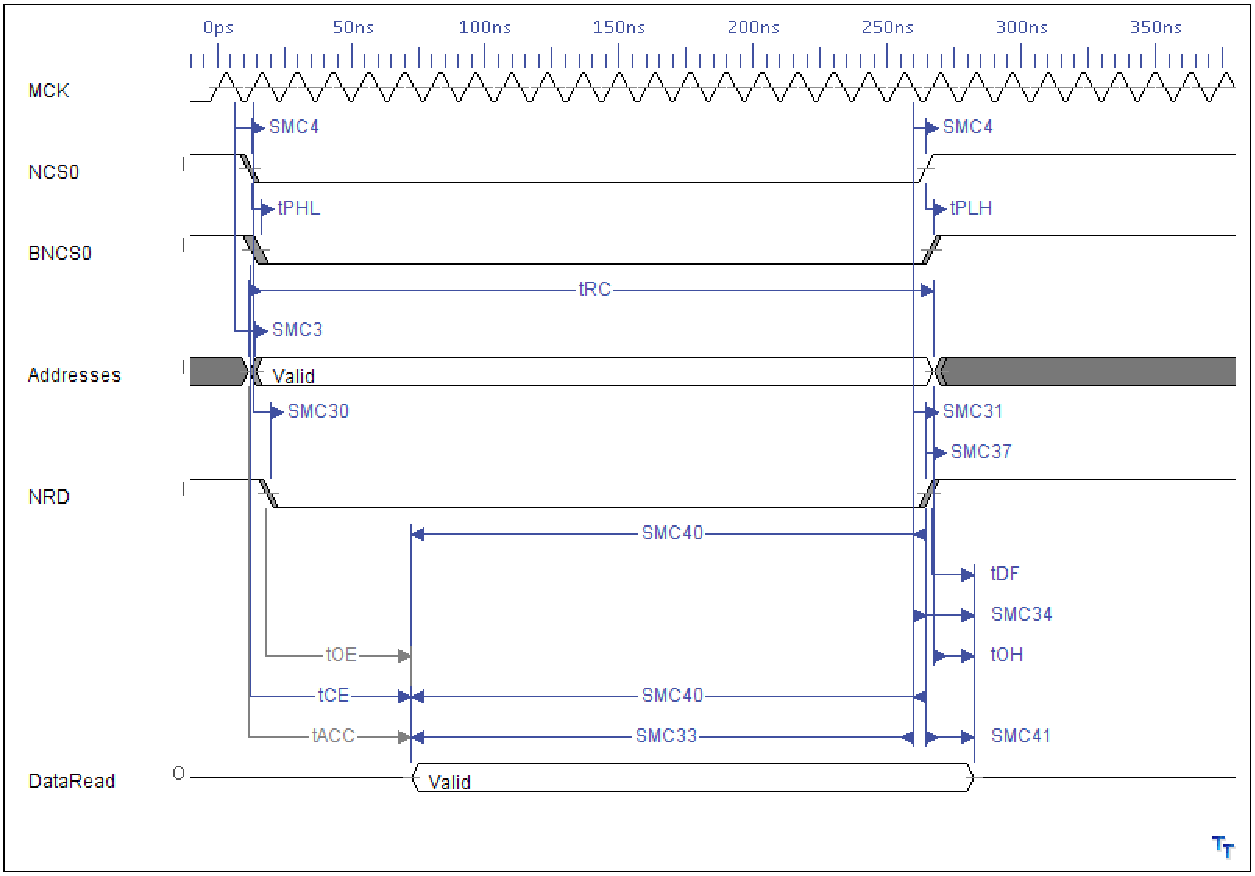


Diagram showing signals and timings of a ROM read. Note that DataRead is valid data.

Signal Name	Signal Description	Signal Time (ns)
tRC	Read Cycle Time	120
tOE	Output Enable to Output Delay	50
tCE	Chip Enable to Output Delay	120
tACC	Address to Output Delay	120
tOH	Output Hold Time from Addresses	0
tPLH	Propagation time due to buffer (low to high)	3.3
tOE	Delay between active read and valid data	6
tACS	Access delay	12
SMC 33	Data Setup Time before NRD High	7.5
SMC 34	D0-D15 in Hold after MCK Falling	1.7

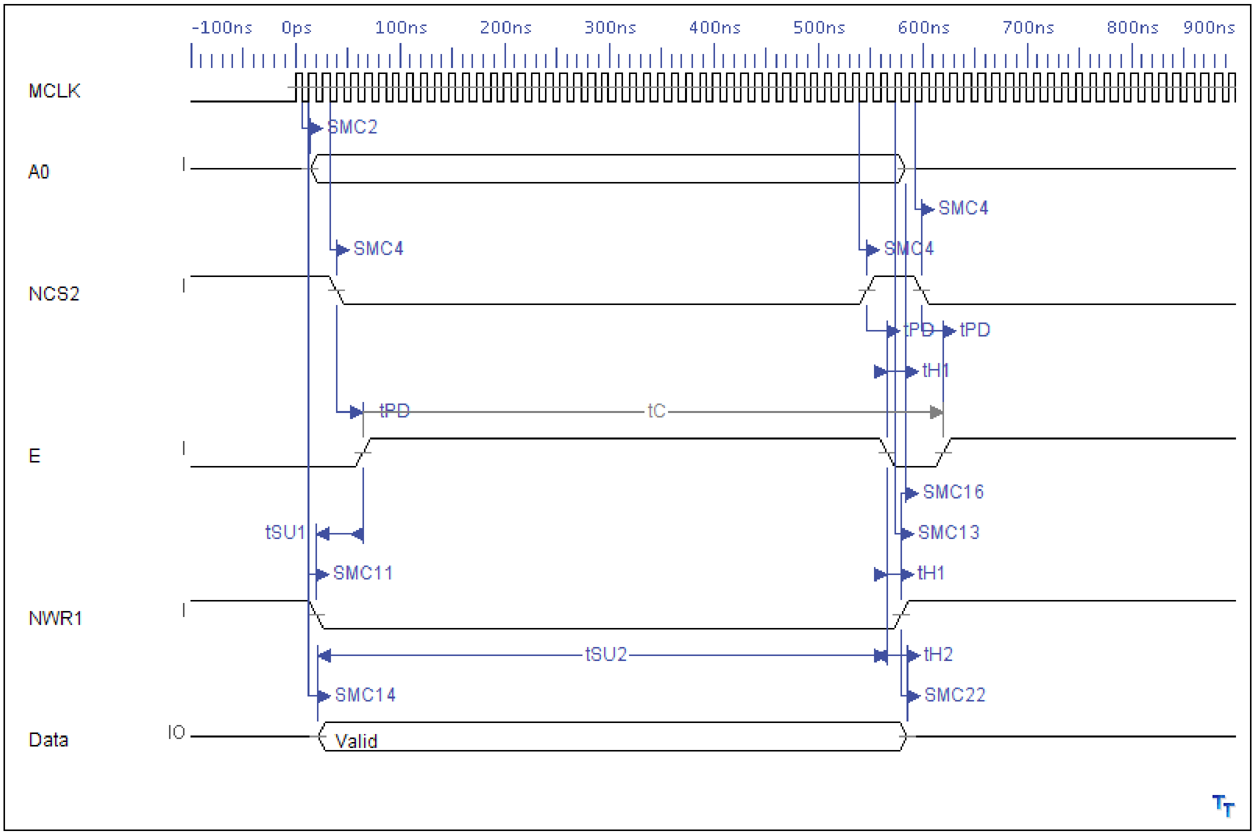
Explanation of ROM read signals and timings.

Display:

This subsection covers the signals and timings associated with the system's display.

The schematic for the display of the system can be found on page 4, titled ARM Display, in the schematics section.

The system uses a Microtips NMTC-S24200XRGHS LCD display module. The display is connected to the board by a 20 pin DIP that a ribbon cable is attached to. The board layout shown at the beginning of this section shows which way the red part of the cable should face once inserted. The display module has its own controller which significantly simplifies CPU interaction with the display. Because the enable signal on the display controller is active high while other enables in the system are active low (most notably NCS2), U1 is used to invert NCS2 and outputs the signal E to the display. It should be noted that when interacting with the display, the CPU uses 42 wait states and 3 address to chip select cycles. Following is a diagram showing the signals and timings that are needed for the CPU to interact with the display module.



Signals and timings used for interaction with display module.

Signal Name	Signal Description	Signal Time (ns)
SMC 2	MCK Falling to NLB/A0 Valid	7.5
tSU1	R/W and RS Setup Time	40
tSU2	Data Setup Time	80
tPD	PLD Propagation Delay	25
tC	Cycle Time	500
tH1	R/W and RS Hold Time	10
tH2	Data Hold Time	10

Explanation of signals and timings of display interaction.

Keypad:

This subsection covers the keypad and encoder chip used by the VOIP system.

The keypad used is modeless (it is made in Hong Kong though) and has a 4x4 key matrix. The encoder used is a Fairchild Semiconductor MM74C923 20-Key Encoder. The schematic for both the encoder and keypad can be found on page 4 (titled ARM Keypad) of the included schematics.

The keypad interacts directly with only the encoder, the encoder handles debouncing key presses as well as the scanning of rows of the keypad and converts key presses into a code which is then sent to the CPU. The CPU receives the data on the pins PIOC26, PIOC27, PIOC28, PIOC29, and PIOC31. PIOC31 is the RDY signal generated by the encoder which indicates that a key press has occurred. Thus, PIOC is set to generate an interrupt whenever a RDY signal change is detected. It should be noted that the state of data being output by the encoder does not change until another key press is detected. Because of the static data output by the encoder, there is no sensitive timing involved when interacting with the encoder. Instead of a timing diagram like those presented in other subsections, a brief description of expected signals will have to suffice.

When no key is being pressed, the encoder (U18) will continuously scan the keypad. While scanning, RDY will be low. The data being output by the keypad during this time will be the key code of the last pressed key. Pressing a key causes some of the output data lines to go low. The keypad is set up so that the QZ-1 key has a key code of 0 while the ENT key has a key code of F. The key codes increment by 1 from left to right on each row.

It should be noted that the relatively large capacitances used to couple the keybounce mask and oscillator pin to ground causes the bounce time on the keypad to be slightly long.

Codec:

This subsection addresses the input and output of audio signals in the VOIP system.

The codec used by the system is a Texus Instruments TLV320AIC1106 PCM Codec. The schematic of the codec (titled ARM Codec) can be found on page 5 of the included schematics.

The codec on this system uses a 13 bit linear filter, which means that the VOIP system can only interact with other systems using the same filter mode.

The codec is set to use Serial Synchronous Controller 0 of the CPU. The expected waveforms of PCMI and PCMO are modulated and sinusoidal (assuming audio data is being transferred). Because the codec expects an input clock frequency of around 2 megahertz, the CPU is set to generate a frequency of the master clock divided by 36 (2.08 megahertz) on the pin TK0. The frame sync signal (TF0) is set to be generated every 256 cycles of the divided master clock.

Following is a diagram of codec signals where TCK is the divided master clock and FSYNC is the frame sync signal.

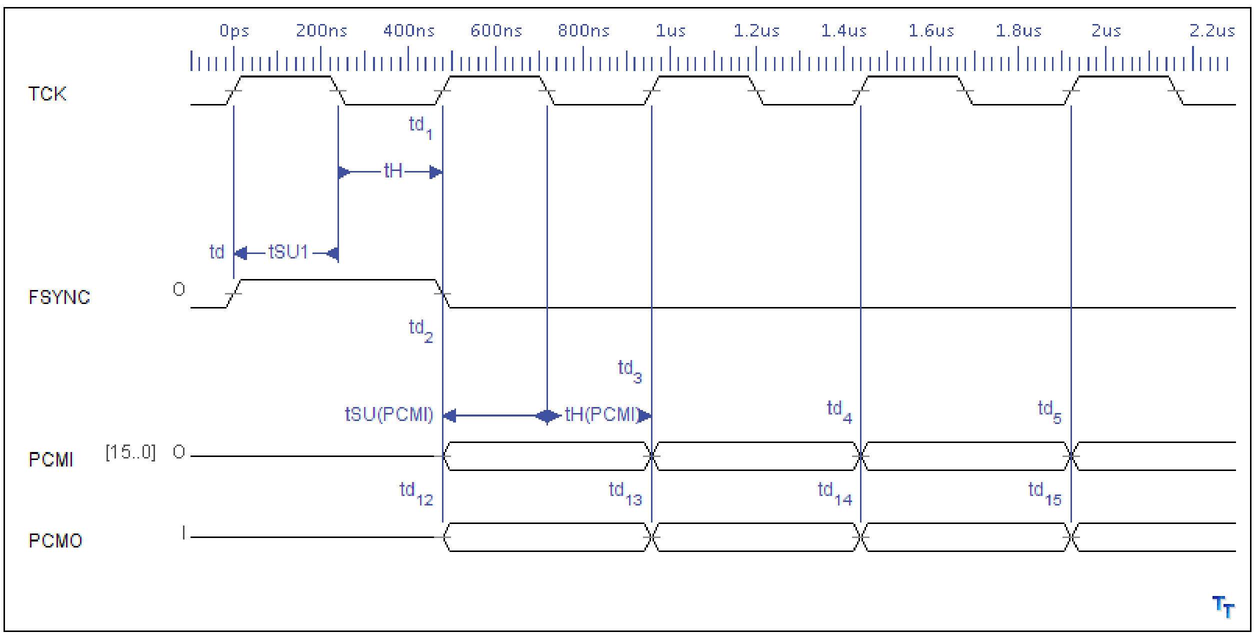


Diagram of codec signals and timings.

Signal Name	Signal Description	Signal Time (ns)
tSU1	Setup time, PCMSYN high before MCLK low.	20
tH	Hold time, PCMSYN high after MCLK low.	20

tSU(PCMI)	Setupt time, PCMI high or low before MCLK low.	20
Signal Name	Signal Description	Signal Time (ns)
tH(PCMI)	Hold time, PCMI high or low after MCLK low.	20
tdx	Synchronization delays.	0

Explanation of signals and timings of codec interaction.

DRAM:

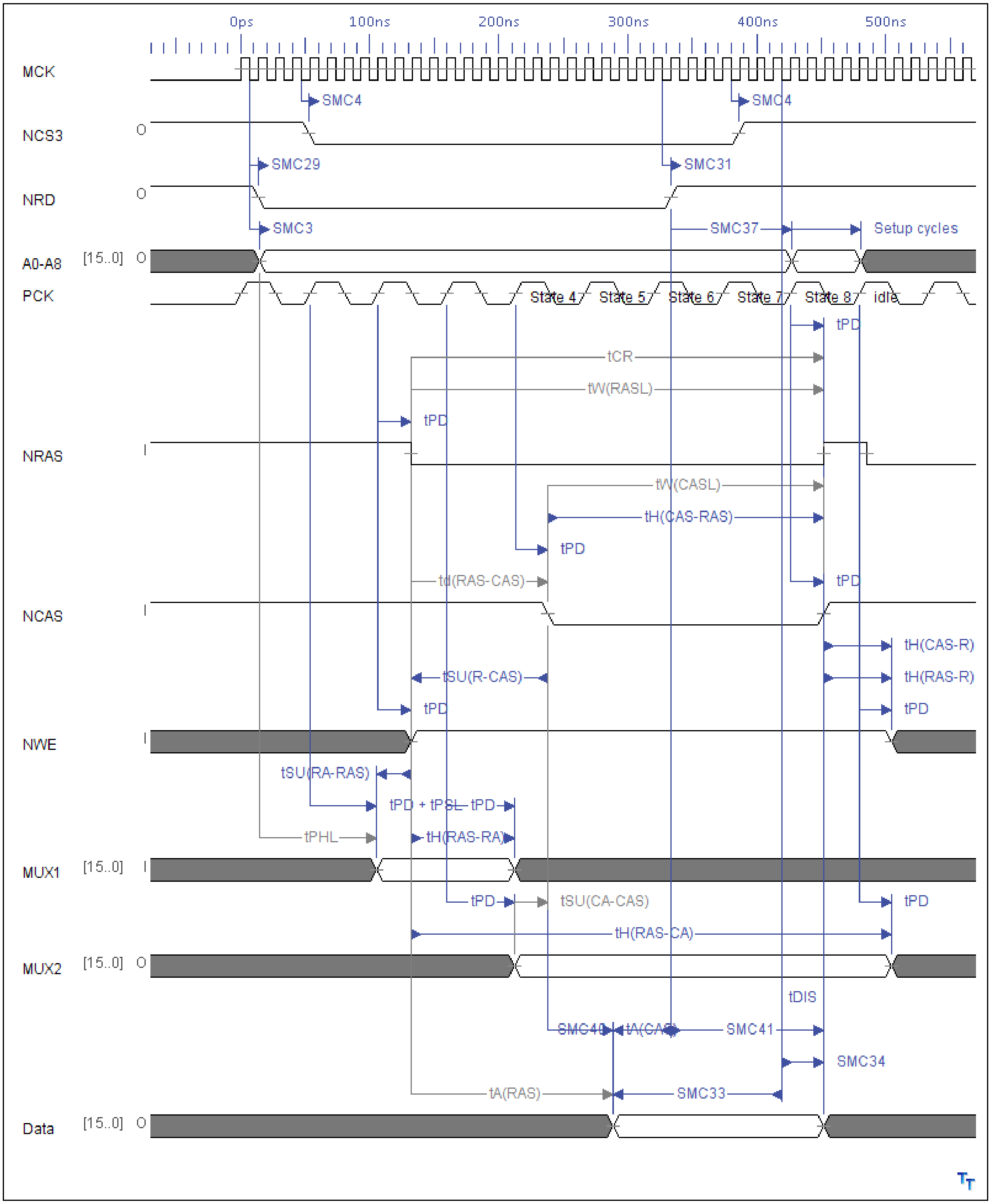
This subsection addresses the dynamic random access memory used by the VOIP system.

The system uses a Mitsubishi MH25609J-10 256K x 9-bit DRAM module. The schematic for DRAM (titled ARM DRAM) can be found on page 7 of the included schematics.

Due to the complexity of interacting with DRAM, a PAL22V10D programmable logic device (U12) is used to generate the necessary signals. U12 is provided with a clock frequency of 18.75 megahertz, which is generated on PIOA4 (set to be a programmable clock). Rather than use a dedicated refresh cycle on the PLD, the system instead reads from every row address the DRAM every 4 milliseconds to prevent stored data from degrading.

Address lines from the CPU are multiplexed by three Texas Instruments 74HC245N multiplexers. The signal controlling which address chunk goes to DRAM is controlled by the PLD and is called MUX1 when the signal is high (for A9-A17 to go through) or MUX2 (for A0-A8 to go through).

The following diagrams show the expected waveforms of signals from the CPU to the PLD and from the PLD to the DRAM module. It should be noted that the CPU is set to have 24 wait states, 7 read/write hold states, 4 read/write setup states, and 3 address to chip select cycles when interacting with chip select 3 (DRAM). Read cycles will be addressed first.



Signal waveforms for a DRAM read.

Signal Name	Signal Description	Signal Time (ns)
tPD	Propagation delay due to PLD	25
SMC 29	MCK Falling to NRD Active	6.8
tSU(RA-RAS)	Setup time from row address to RAS	0
tPSL	Propagation delay due to multiplexers.	27
td(RAS-CAS)	Minimum delay between RAS and CAS signals.	75
tSU(R-CAS)	Setup time of NWE before CAS	0
tH(RAS-RA)	Hold time of row address after RAS low.	20
tH(RAS-CA)	Hold time of column address after RAS low.	100
tCR	Read cycle time.	260
tW(RASL)	RAS low pulse width.	150
tW(CASL)	CAS low pulse width.	75
tH(CAS-RAS)	Hold time of CAS after RAS low.	75
tA(CAS)	Delay between CAS low and data valid.	50
tH(CAS-R)	Hold time from CAS high to NWE invalid.	0
tH(RAS-R)	Hold time from RAS high to NWE invalid.	10
tDIS	Time from CAS high to data invalid.	0

Explanation of signals and timings for a DRAM read.

The following diagram and table show and explain the signals and timings of a DRAM write.

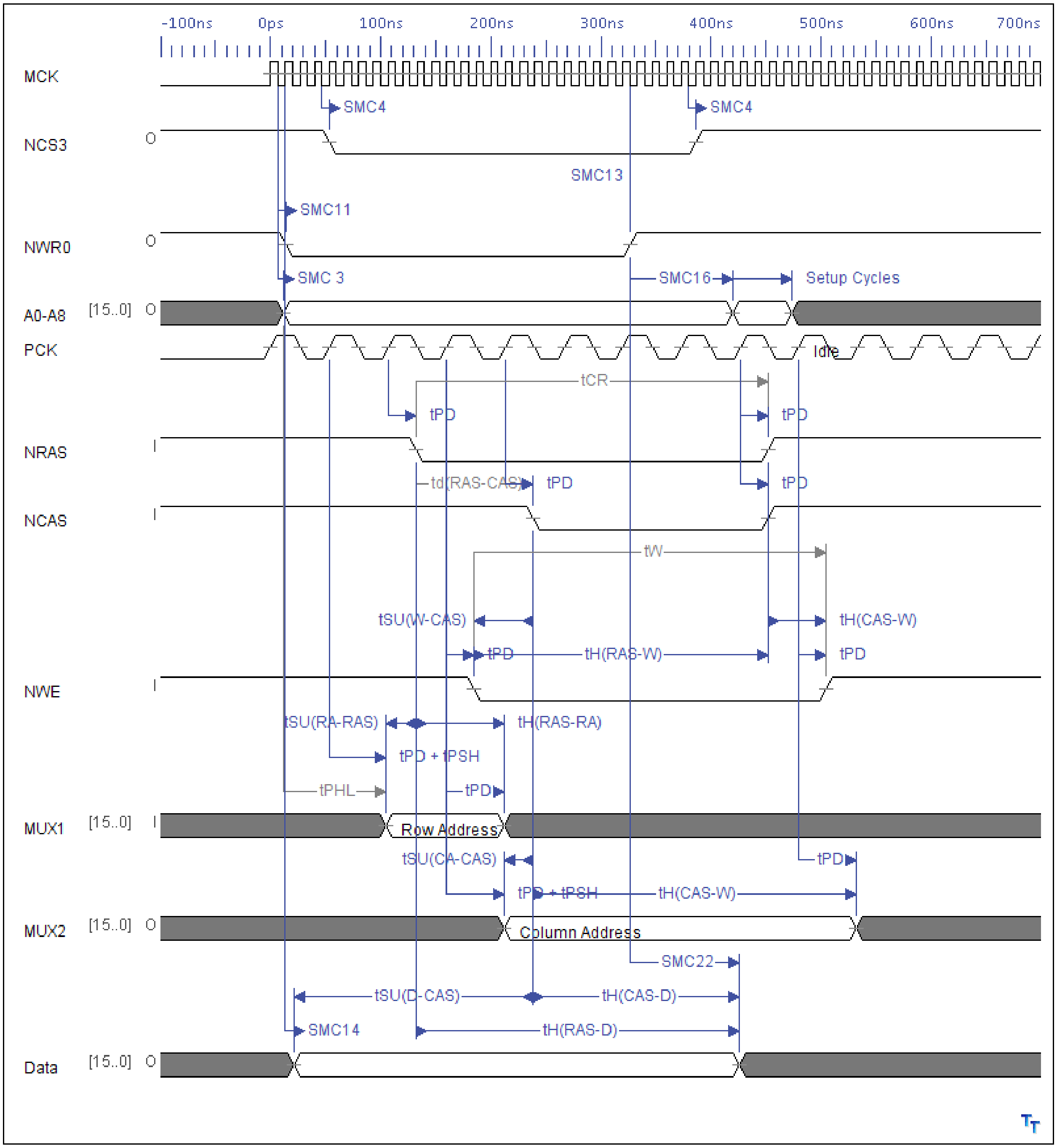


Diagram of the signals and timings used in a DRAM write.

Signal Name	Signal Description	Signal Time (ns)
tCW	Write Cycle Time.	260
tSU(W-CAS)	Write setup time before CAS low.	0
tH(CAS-W)	Write hold time after CAS low.	30
tH(RAS-W)	Write hold time after RAS low.	105
tH(W-RAS)	RAS hold time after write.	45
tH(W-CAS)	CAS hold time after write.	45
tW	Write pulse width.	30
tSU(D-CAS)	Data in setup time before CAS low.	0
tH(CAS-D)	Data in hold time after CAS low.	30
th(RAS-D)	Data in hold time after RAS low.	105

Explanation of signals and timings for a DRAM write.

The following 11 pages are the code (written in ABEL) used to program the DRAM controller. The code sets up the PLD as a Moore state machine that outputs the proper signals and changes state with each rising clock edge.

```
MODULE DramController

TITLE 'DramController'

" Dram Controller       Device          'GAL22V10'

" This GAL implements a DRAM controller that uses state
" bits to generate the timing needed for read and write
" cycles. The device outputs 4 signals, a select for
" multiplexers, an active low write enable, a RAS signal,
" and a CAS signal. It takes NWR0 and NCS3 as inputs.
" This logic is implemented as a Moore state machine.

" Revision History
"       2/18/2012       Josh Fromm      Initial Version


" pins

Clock   PIN     1;                      "in     global clock pin
NWR0    PIN     2;                      "in     read or write data
NCS3    PIN     3;                      "in     DRAM chip select
RAS     PIN     14      ISTYPE 'reg';   "out    RAS signal for DRAM
CAS     PIN     15      ISTYPE 'reg';   "out    CAS signal for DRAM
```

```
SEL     PIN    16     ISTYPE 'reg';   "out    Mux select signal

WE      PIN    17     ISTYPE 'reg';   "out    Write or read signal

StBit0  PIN    18     ISTYPE 'reg';   "out    state bit 0

StBit1  PIN    19     ISTYPE 'reg';  "out     state bit 1

StBit2  PIN   20      ISTYPE 'reg';   "out    state bit 2


" state machine definitions


SBITS   = [ RAS, CAS, SEL, WE, StBit0, StBit1, StBit2 ];        "state bits

                                                                 "state assignment

Start   = [  0,  0,  0, 0,    0,    0,    0 ];     "boot up state

Idle     = [  1,  1,  1, 1,    0,    0,    0 ];     "waiting for dram to be selected

State1 = [  1,  1,  1, 1,    1,    1,    1 ];     "first state of both cycles

State2  = [  0,  1,  1, 1,    0,    0,    0 ];   "second state of both cycles

Write3  = [  0,  1,  0, 0,    0,    0,    0 ];    "third state of write cycle

Write4  = [  0,  0,  0, 0,    1,    0,    1 ];    "fourth state of write cycle

Write5  = [  0,  0,  0, 0,    0,    0,    1 ];    "fifth state of write cycle

Write6  = [  0,  0,  0, 0,    0,    1,    0 ];    "sixth state of write cycle

Write7  = [  0,  0,  0, 0,    0,    1,    1 ];    "seventh state of write cycle

Write8  = [  1,  1,  0, 0,    0,    0,    0 ];    "eighth state of write cycle

Read3  = [  0,  1,  0, 1,    0,    0,    1 ];     "third state of read cycle

Read4  = [  0,  0,  0, 1,    1,    0,    1 ];     "fourth state of read cycle

Read5   = [  0,  0,  0, 1,    0,    0,    1 ];    "fifth state of read cycle

Read6   = [  0,  0,  0, 1,    0,    1,    0 ];   "sixth state of read cycle
```

```
Read7   = [   0,   0,   0,  1,     0,     1,     1 ];    "seventh state of read cycle

Read8   = [   1,   1,   0,  1,     0,     0,     0 ];     "ninth state of read cycle


EQUATIONS


"output enables

"SBITS.OE = 1;


" clocks


SBITS.CLK = Clock;



STATE_DIAGRAM       SBITS


STATE   Start:

          GOTO Idle;


STATE  Idle:

          IF        (!NCS3)           THEN   State1  "if cpu accessing dram go to state 1

     ELSE          Idle


"all other states progress without condition
```

```
STATE  State1:
   GOTO State2;

STATE   State2:
          IF         (!NWR0)          THEN   Write3
  ELSE   Read3

STATE  Write3:
          GOTO Write4;

STATE  Write4:
          GOTO Write5;

STATE  Write5:
          GOTO Write6;

STATE  Write6:
          GOTO Write7;

STATE   Write7:
   GOTO Write8;

STATE   Write8:
   GOTO Idle;
```

```
STATE  Read3:

        GOTO Read4;


STATE  Read4:

        GOTO Read5;


STATE  Read5:

        GOTO Read6;


STATE  Read6:

        GOTO Read7;


STATE   Read7:

  GOTO Read8;


STATE   Read8:

  GOTO Idle;


TEST_VECTORS


"test dram controller state machine


( [ Clock, NCS3, NWR0 ] -> [ RAS, CAS, SEL, WE, StBit0, StBit1, StBit2 ] )
```

```
"first clear out PLD and make sure state is Idle by allowing 30 clock cycles to pass

@repeat 30{

[    .c.,    1,    1 ] -> [ .X., .X., .X.,.X.,     .X.,    .X.,   .X. ];

}


"then check read sequence outputs


[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm that idle state is maintained

[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];

[    .C.,    0,    1 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1

[    .C.,    0,    1 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2

[    .C.,    0,    1 ] -> [   0,   1,   0,  1, 0, 0, 1 ];  "move to read state 3

[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 1, 0, 1 ];  "move to read state 4

[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 0, 1 ];  "move to read state 5

[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 0 ];  "move to read state 6

[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 1 ];  "move to read state 7

[    .C.,    0,    1 ] -> [   1,   1,   0,  1, 0, 0, 0 ];  "move to read state 8

[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state

[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state

"now begin to test write sequence

[    .C.,    0,    0 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1

[    .C.,    0,    0 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2

[    .C.,    0,    0 ] -> [   0,   1,   0,  0, 0, 0, 0 ];  "move to write state 3
```

```
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 1, 0, 1 ];  "move to write state 4
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,    0,    0 ] -> [   1,   1,   0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state


"now test multiple writes
[    .C.,    0,    0 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    0 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    0 ] -> [   0,   1,   0,  0, 0, 0, 0 ];  "move to write state 3
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 1, 0, 1 ];  "move to write state 4
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,    0,    0 ] -> [   1,   1,   0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,    0,    0 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    0 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    0 ] -> [   0,   1,   0,  0, 0, 0, 0 ];  "move to write state 3
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 1, 0, 1 ];  "move to write state 4
```

```
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,    0,    0 ] -> [   1,   1,   0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,    0,    0 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    0 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    0 ] -> [   0,   1,   0,  0, 0, 0, 0 ];  "move to write state 3
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 1, 0, 1 ];  "move to write state 4
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,    0,    0 ] -> [   1,   1,   0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state

"now test multiple reads
[    .C.,    0,    1 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    1 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    1 ] -> [   0,   1,   0,  1, 0, 0, 1 ];  "move to read state 3
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 1, 0, 1 ];  "move to read state 4
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 0, 1 ];  "move to read state 5
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 0 ];  "move to read state 6
```

```
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 1 ];  "move to read state 7
[    .C.,    0,    1 ] -> [   1,   1,   0,  1, 0, 0, 0 ];  "move to read state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,    0,    1 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    1 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    1 ] -> [   0,   1,   0,  1, 0, 0, 1 ];  "move to read state 3
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 1, 0, 1 ];  "move to read state 4
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 0, 1 ];  "move to read state 5
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 0 ];  "move to read state 6
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 1 ];  "move to read state 7
[    .C.,    0,    1 ] -> [   1,   1,   0,  1, 0, 0, 0 ];  "move to read state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,    0,    1 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    1 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    1 ] -> [   0,   1,   0,  1, 0, 0, 1 ];  "move to read state 3
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 1, 0, 1 ];  "move to read state 4
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 0, 1 ];  "move to read state 5
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 0 ];  "move to read state 6
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 1 ];  "move to read state 7
[    .C.,    0,    1 ] -> [   1,   1,   0,  1, 0, 0, 0 ];  "move to read state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
```

```
"now test mixed reads and writes
[    .C.,   0,   0 ] -> [  1,  1,  1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,   0,   0 ] -> [  0,  1,  1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,   0,   0 ] -> [  0,  1,  0,  0, 0, 0, 0 ];  "move to write state 3
[    .C.,   0,   0 ] -> [  0,  0,  0,  0, 1, 0, 1 ];  "move to write state 4
[    .C.,   0,   0 ] -> [  0,  0,  0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,   0,   0 ] -> [  0,  0,  0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,   0,   0 ] -> [  0,  0,  0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,   0,   0 ] -> [  1,  1,  0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,   1,   1 ] -> [  1,  1,  1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,   1,   1 ] -> [  1,  1,  1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,   0,   1 ] -> [  1,  1,  1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,   0,   1 ] -> [  0,  1,  1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,   0,   1 ] -> [  0,  1,  0,  1, 0, 0, 1 ];  "move to read state 3
[    .C.,   0,   1 ] -> [  0,  0,  0,  1, 1, 0, 1 ];  "move to read state 4
[    .C.,   0,   1 ] -> [  0,  0,  0,  1, 0, 0, 1 ];  "move to read state 5
[    .C.,   0,   1 ] -> [  0,  0,  0,  1, 0, 1, 0 ];  "move to read state 6
[    .C.,   0,   1 ] -> [  0,  0,  0,  1, 0, 1, 1 ];  "move to read state 7
[    .C.,   0,   1 ] -> [  1,  1,  0,  1, 0, 0, 0 ];  "move to read state 8
[    .C.,   1,   1 ] -> [  1,  1,  1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,   1,   1 ] -> [  1,  1,  1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,   0,   0 ] -> [  1,  1,  1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,   0,   0 ] -> [  0,  1,  1,  1, 0, 0, 0 ];  "move to state 2
```

```
[    .C.,    0,    0 ] -> [   0,   1,   0,  0, 0, 0, 0 ];  "move to write state 3
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 1, 0, 1 ];  "move to write state 4
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 0, 1 ];  "move to write state 5
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 0 ];  "move to write state 6
[    .C.,    0,    0 ] -> [   0,   0,   0,  0, 0, 1, 1 ];  "move to write state 7
[    .C.,    0,    0 ] -> [   1,   1,   0,  0, 0, 0, 0 ];  "move to write state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state
[    .C.,    0,    1 ] -> [   1,   1,   1,  1, 1, 1, 1 ];  "move to state 1
[    .C.,    0,    1 ] -> [   0,   1,   1,  1, 0, 0, 0 ];  "move to state 2
[    .C.,    0,    1 ] -> [   0,   1,   0,  1, 0, 0, 1 ];  "move to read state 3
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 1, 0, 1 ];  "move to read state 4
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 0, 1 ];  "move to read state 5
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 0 ];  "move to read state 6
[    .C.,    0,    1 ] -> [   0,   0,   0,  1, 0, 1, 1 ];  "move to read state 7
[    .C.,    0,    1 ] -> [   1,   1,   0,  1, 0, 0, 0 ];  "move to read state 8
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "return to idle state
[    .C.,    1,    1 ] -> [   1,   1,   1,  1, 0, 0, 0 ];  "confirm idle state


END
```

Ethernet:

This subsection will address the Ethernet chip and jack used by the VOIP system.

The system uses a National Semiconductor DP83848C PHYTER chip to interact with a MagJack SI-50170-F Ethernet Jack. The PHYTER chip then communicates with the CPU over the CPU's EMAC controller. The schematic for the PHYTER chip and jack (titled ARM Ethernet) can be found on page 6 of the included schematics.

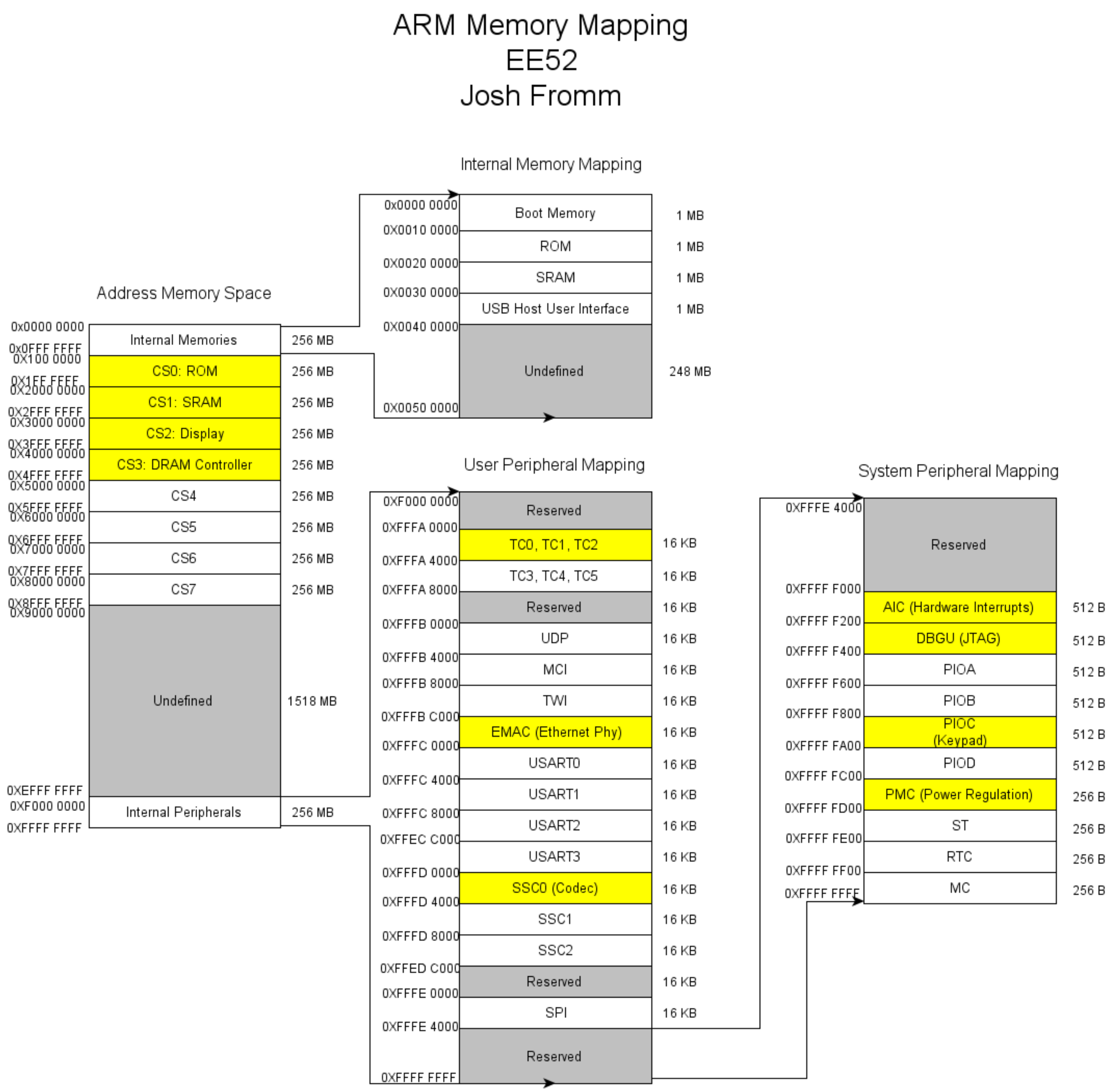
Because the EMAC controller is designed specifically to interact with an Ethernet chip, there is no specific timing needed. The main clock of the Ethernet chip (signal MDC) is set to be the master clock divided by 32 (2.34 megahertz).

To interact properly with the PHYTER chip, a large majority of PIOA is converted to peripheral A and a large majority of PIOB is converted to peripheral B. The signals of the main data lines RXD_0, RXD_1, TXD_0, TXD_1, etcetera, should appear to be very noisy if monitored during use due to the high speed of data transmission.

The PHYTER chip is set to run in advertised mode at 10BASE-T speed with either half or full duplex. The PHYTER also is set to run in MII MAC interface mode.

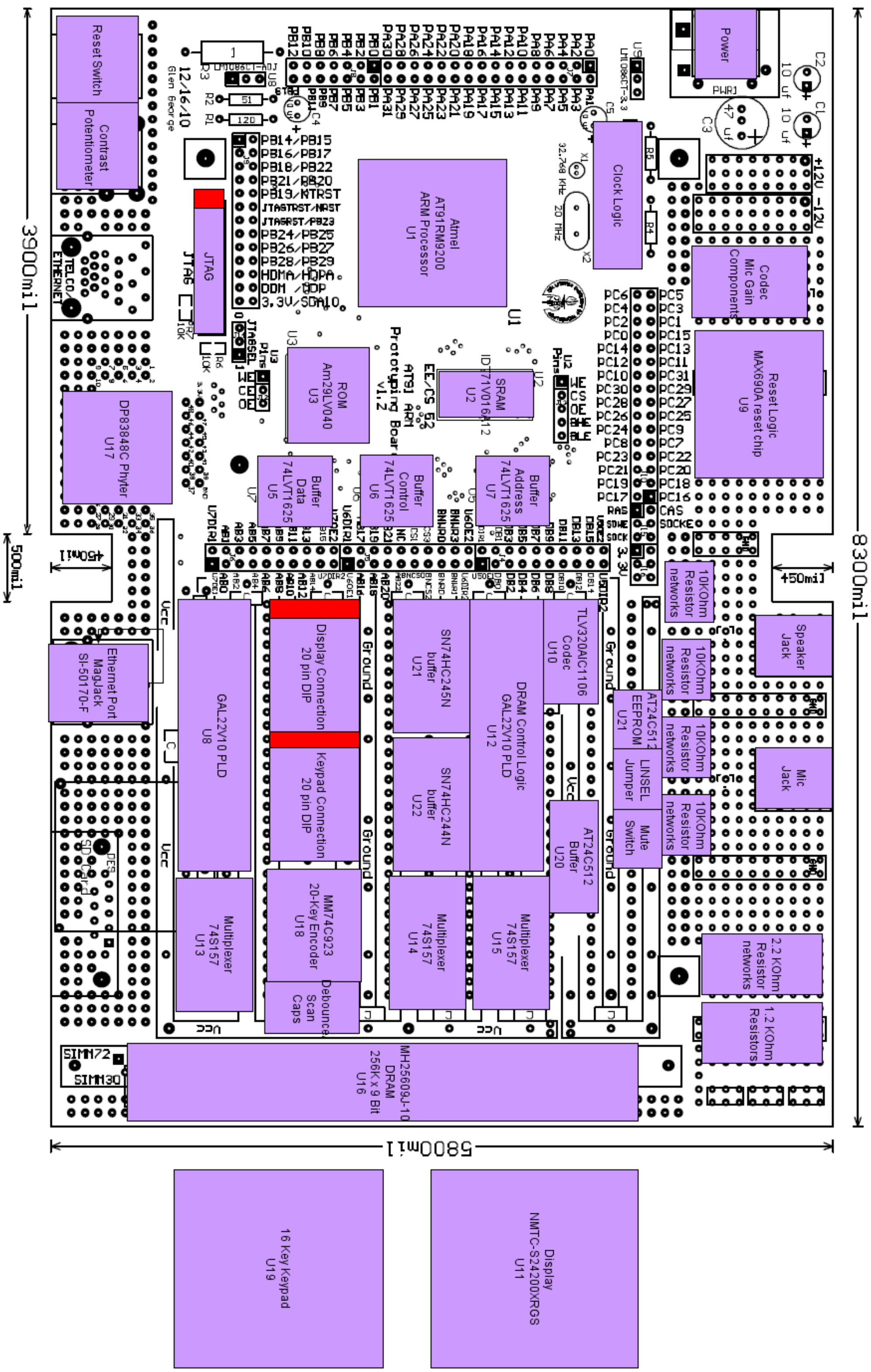
Detailed Description of Hardware:

This section offers a description of the explicit purpose of each chip in the system. Chips will be referenced by the component label assigned to them in the schematics. Following is a memory mapping that better shows addresses of peripherals than the previous memory mapping. Also a board layout with chip labels is provided.



Memory mapping with expanded addresses.

ARM Board Layout
EE52
Josh Fromm



EEPROM, ROM, SRAM, and Buffers:

When the system first powers on or after the CPU receives an active reset signal, the CPU enters its boot-up sequence. In the boot-up sequence, the CPU checks to see whether a suitable EEPROM is connected via two wire interface. It checks by searching for a stored vector on the EEPROM. The system's EEPROM (U22) is connected to the CPU's two wire interface and contains vectors that should cause the code stored on U22 to be executed. The code sets up chip selects 0 and 1 and then copies all the data on U3 to U2. The code then sets the program counter to the lowest address in U2 (0x20000000). The code stored on U3 is the code needed to run the entire board. The data lines between U1 and U2 and the data lines between U1 and U3 are connected directly, as are the address lines. The control lines, however, go through U6. While running, U2 is used as the system's code storage and variable storage.

All other peripherals have data, control, and address lines from the CPU first pass through U5, U6, and U7 respectively. The buffers are bidirectional, but U6 and U7 have their enable and direction pins tied so that control and addresses are always buffered from the CPU to the peripherals. Because data must be bidirectional, U5 relies on U8 to control the direction and output. U8 is a programmable logic device that monitors chip selects 0 and 1 to determine whether U5 should be enabled or not (it's enabled when the chip selects are inactive) and monitors NRD to determine direction (NRD active implies a read which sets U5 to buffer from peripherals to the CPU). The following several pages contain the code (in ABEL) used to program U8.

```
MODULE MainPLD

TITLE 'Reset and Buffer Logic'

" Device 'GAL22V10D'

" This PLD is used to control whether data buffer U5 is enabled and the direction
" of buffering. It also monitors the JTAG reset and reset lines to produce a signal
" indicating when a debug reset should be performed. Finally, this PLD inverts
" NCS2 to create an active high chip select for the system's display.

" Revision History
"       20 January 2012     Josh Fromm    Initial Version
"       24 January 2012 Josh Fromm Buffer Logic Added
"   26 March   2012 Josh Fromm  Updated Comments
"Pins

JTAGRST             PIN     2;                    "input  One of the signals to be sent into
not nor gate
NRST        PIN     3;                  "input  Other signal to be sent into not nor gate
BJTAGRST    PIN     23    ISTYPE 'com';  "output        Output of not nor gate
NCS0        PIN     4;                  "input  chip select for rom
NCS1        PIN     5;                  "input  chip select for sram
NRD         PIN    6;                   "input  signal indicating data is incoming
```

```
U5DIR           PIN     22      ISTYPE 'com';   "output which direction data buffer should
transmit in

U5OE            PIN     21      ISTYPE 'com';   "output whether data buffer is enabled or not

NCS2     PIN  7            "input to display, should be inverted

E        PIN  20 ISTYPE 'com';   "output inversion of ncs2


equations


!BJTAGRST = !JTAGRST # !NRST;   "If either a JTAG reset or regular reset occurs,

                        "the debugger should be reset.


U5OE = !NCS0 # !NCS1;        "The data buffer should be disabled during an

                        "access of either ROM or SRAM


U5DIR = NRD;            "during a read, data buffer should buffer from

                        "peripherals to the CPU.


E = !NCS2


"output enables

BJTAGRST.OE  = 1;                          "always enable not nor gate

U5OE.OE      = 1;                                "always enable buffer logic outputs

U5DIR.OE = 1;

E.OE = 1;               "always invert ncs2
```

```
TEST_VECTORS ( [ JTAGRST, NRST, NCS0, NCS1, NRD ] -> [ BJTAGRST, U5DIR, U5OE ] )

"run through possible combinations as a test.

[ 0, 0, 0, 0, 0 ] -> [ 0, 0, 1 ];

[ 0, 1, 0, 1, 1 ] -> [ 0, 1, 1 ];

[ 1, 0, 1, 0, 1 ] -> [ 0, 1, 1 ];

[ 1, 1, 1, 1, 1 ] -> [ 1, 1, 0 ];

END
```

Reset Circuit:

The reset circuit of the system, U9, is fairly straight forward. The chip used has several features, however only its watchdog timer was used. When the watchdog pin is grounded for several seconds, the reset pin, which is normally driven high, drops low for a moment. Thus the reset switch is a connection between the watchdog pin and ground. U9 also features an LED between power and U9's power pin. This LED serves to indicate when the board is powered. The reset signal produced by U9 is connected to U20, a 5V tolerant 3.3V buffer. The output of U20 is then connected directly to the systems NRST line. The reset signal generated by U9 is also used as input into U8 to generate the debugger reset signal which is also passed through U20 before going to the NJTAGTRST pin on the CPU.

Keypad and Encoder:

Keypad interaction with other components is very limited. The keypad encoder U18 is the only thing that interacts directly with the keypad U19. U18 scans each column of the keypad at a very fast rate and detects whether a key in that column is being pressed. If a key press is detected, U18 outputs a key code to U1. Because only one column is scanned at a time, simultaneous key presses can only be detected if they are in the same column. The enable pin of U18 is set low by U1 through control of PIOC.

Display:

Due to the sensitivity of the display module and the relatively long distance of the ribbon cable connecting it to the board, all signals sent to or from the display are buffered by 5V buffers U21 and U22. U21 is bidirectional and has its direction line tied to the buffered NRD signal. Because all display data lines pass through U21, data can be both read from and written to the display module. U22 is unidirectional and buffers signals from the CPU to display only. Because the display module doesn't use the systems address bus, A0 was chosen to be the display module's register select. This means that the status of A0 during a read or write determines how the display responds. No other address bit is significant when interacting with the display. The contrast of the display can be controlled by rotating the knob on the contrast potentiometer located next to the system's reset switch.

Codec:

The codec interacts with the CPU through the system's synchronous serial controller 0. Because of this, it does not interact with U5, U6, or U7 at all. Instead it receives all data from the CPU via TD0 and sends out data to the CPU via RD0. U10 takes in data from the system's microphone jack and converts that data into a digital signal which is sent to the CPU. Similarly, when the CPU is outputting audio data, it is sent to the Codec and is converted to an analog signal which is sent out to the system's speaker jack. Data being sent to or received from the codec by the CPU is stored on the DRAM (U16).

DRAM:

The systems DRAM (U16) is used to store large audio buffers that are input and output by the codec (U10). The signals used to control DRAM reads and writes are generated by U12, which monitors NCS3 and NWR0 to determine whether a read cycle or write cycle should be performed. The DRAM uses 3 8:4 multiplexers (U13, U14, and U15) to properly select row and column addresses during a read or write cycle. The data from the DRAM then is buffered by U5 and sent to the CPU for a read cycle or sent from the CPU to be buffered by U5 and written to U16.

Ethernet:

The systems Ethernet chip, U17, interacts only with the CPU through the CPU's EMAC controller. Thus, no additional buffers are needed. U17 receives divided master clock signals on RX_CLK and TX_CLK as well as its own dedicated 25 megahertz clock signal which go to the X1 and X2 pins. When an Ethernet connection is made, data is received on the systems Ethernet port, handled by U17 and sent to the CPU's EMAC controller. Likewise, when the CPU is outputting data, it is first sent to U17 where it is processed and then output through the Ethernet jack.

Software Manual:

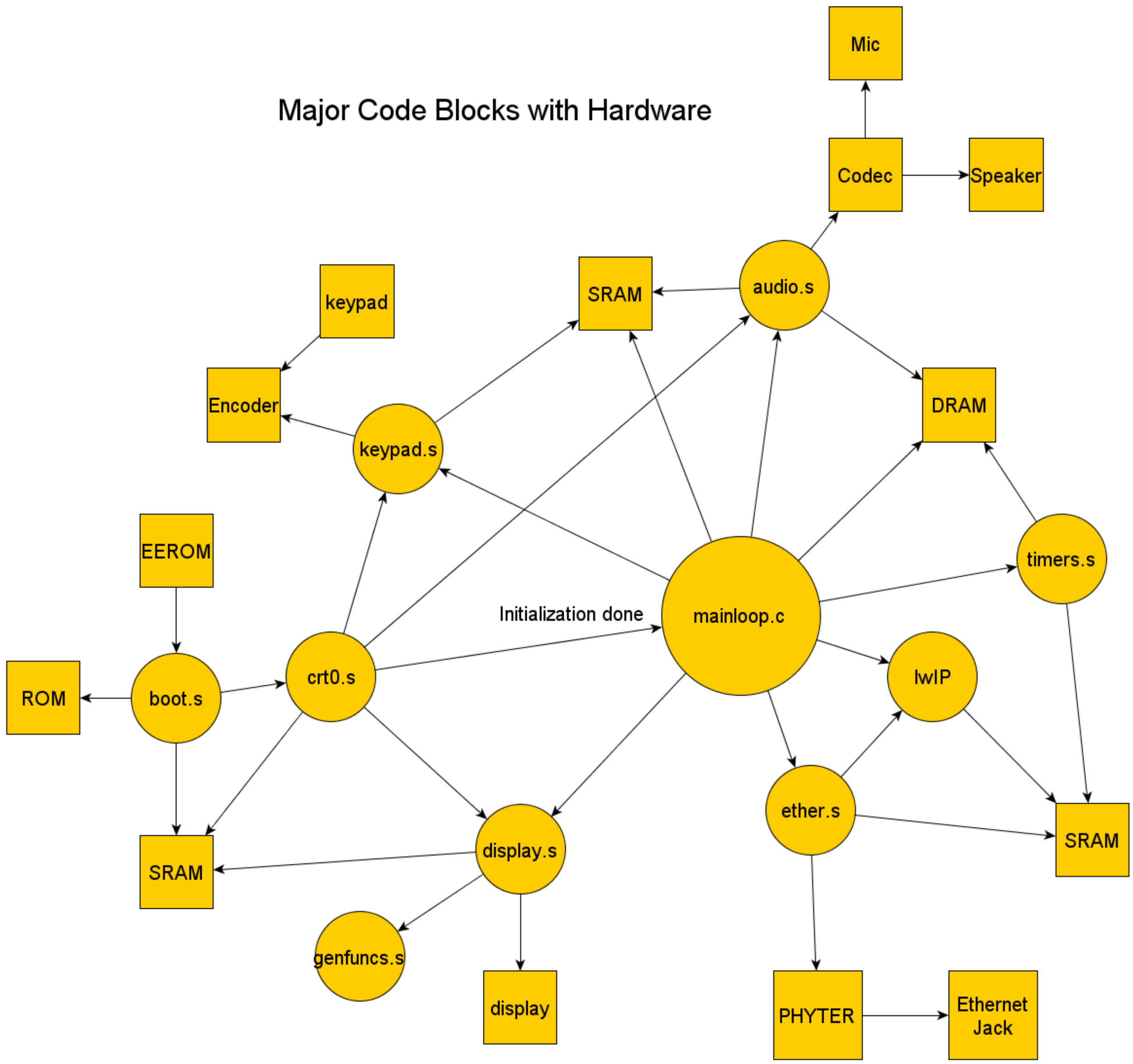
This details how the software run by the system works.

The following image is a diagram of how each of the functions of the software interacts with each other and the hardware.

For more detailed explanations of how software works, see the software section at the end of this manual.

It should be noted that the system uses code developed by the Swedish Institue of Computer Science for a majority of ethernet interaction. The code is called lwIP. lwIP is only briefly covered in this software manual.

Major Code Blocks with Hardware



Explanation of File Purposes:

- Boot.s:

  Boot.s is the code stored on the system's EEPROM. When the system restarts, either through power up or the reset switch, boot.s is executed. Boot.s sets up the chip selects for ROM and SRAM, sets up the main clock (20 megahertz) instead of the default slow clock (32 kilohertz), and copies the contents of the ROM onto SRAM.
- Crt0.s:

  Crt0.s is the system's next layer of initialization code. Crt0.s is run after boot.s moves the system's entire body of code to SRAM. Crt0.s sets up all chip selects, establishes the master clock of the system to be 75 megahertz, calls the initialization functions of the display, keypad, and audio code located in display.s, keypad.s, and audio.s respectively. When finished, crt0.s branches to the code's main loop.
- Mainloop.c:

  Mainloop.c runs the system after initialization. The main loop interacts directly with all peripheral files of the system. Mainloop.c checks if keypad presses are available and handles key codes through accesses to keypad.s. The main loop also displays statuses through interaction with display.s. Audio buffers are both provided and acquired through interaction with the system's DRAM and the code in audio.s. The main loop regulates Ethernet input and output through calls to functions in lwIP and ether.s. The main loop also makes regualre calls to timers.s to determine how much time has passed since a previous call.
- Keypad.s

  Keypad.s is the code used to monitor user input from the system's keypad. Keypad.s uses an interrupt caused by a change in the RDY signal generated by the system's encoder. When an interrupt occurs, the keypad event handler acquires the key code of the most recent key press and sets variable to save the key code and also sets a flag indicating a key press is available. These variables are used to return values to the main loop when the appropriate functions are called.
- Display.s:

  Display.s contains all the code used to interact with the system's display module. Display.s uses calls to genfuncs.s to aid in the algorithm used to convert numbers to their ASCII equivalent. Display.s is capable of displaying a set of statuses that can be changed through slight modification of the code, any decimal number, and any hexadecimal number.

- Timers.s:

  Timers.s uses the system's TC0 (timer counter 0) to trigger an interrupt each millisecond. The interrupt handler keeps track of how many interrupts have occurred since the last call to the elapsed_time function and also refreshes the DRAM through reads to all column addresses.

- Audio.s:

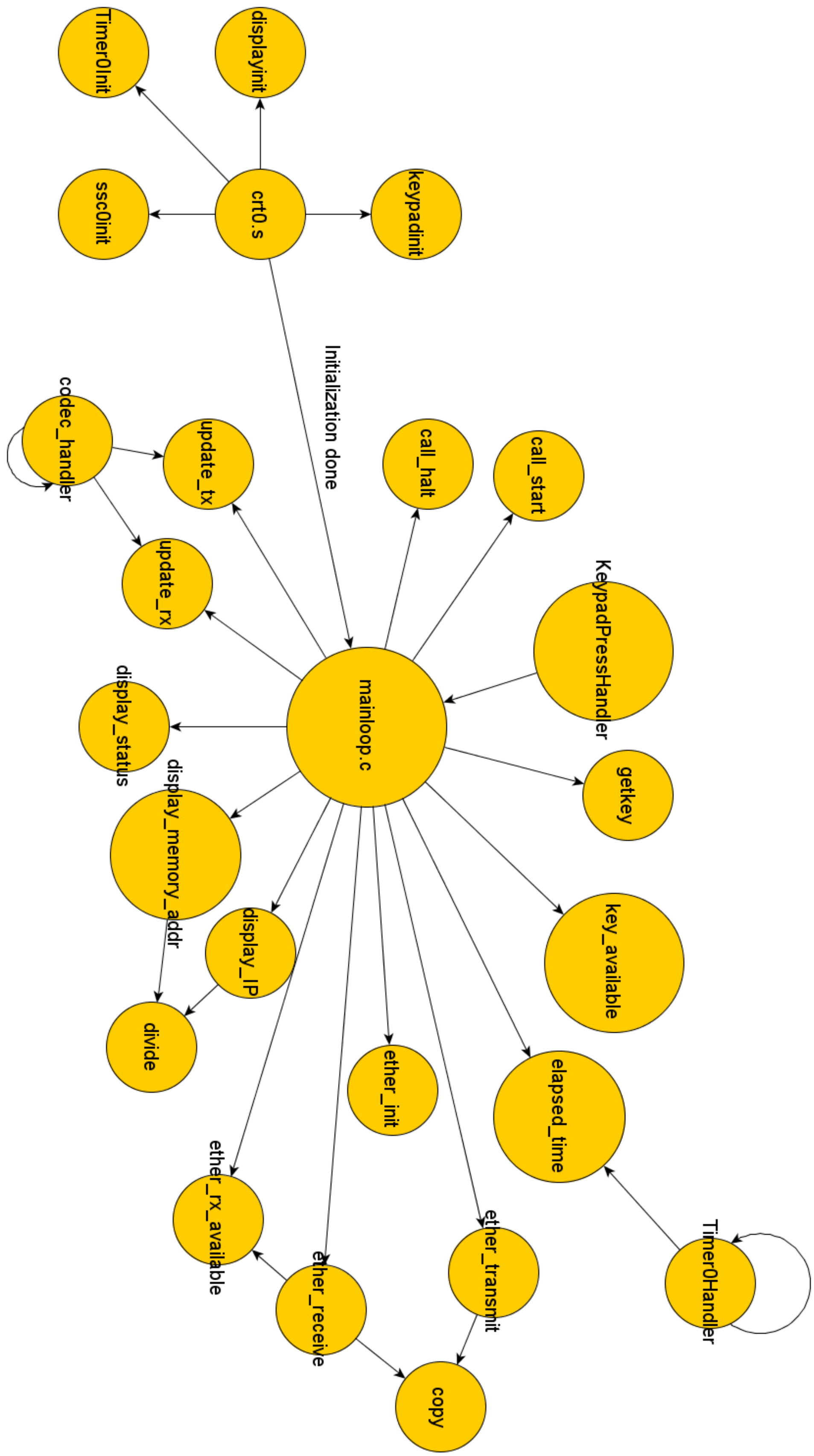
  Audio.s contains the functions needed to set up SSC0 interrupts, an event handler for those interrupts, and functions used by the main loop to update the buffers containing data that is input and output from the codec.
  Audio.s interacts directly with the codec, which then interacts with the system's speaker and microphone jacks.

- Ether.s:

  Ether.s contains functions used to convert between the PBUF data buffers used by the lwIP code and standard buffers. Ether.s interacts with the main loop to receive buffers which are converted to PBUFs and sent out over Ethernet as well as to provide buffers that have been converted from PBUFs. Ether.s also interacts with lwIP to allocate the PBUFs needed when preparing to transmit a buffer.

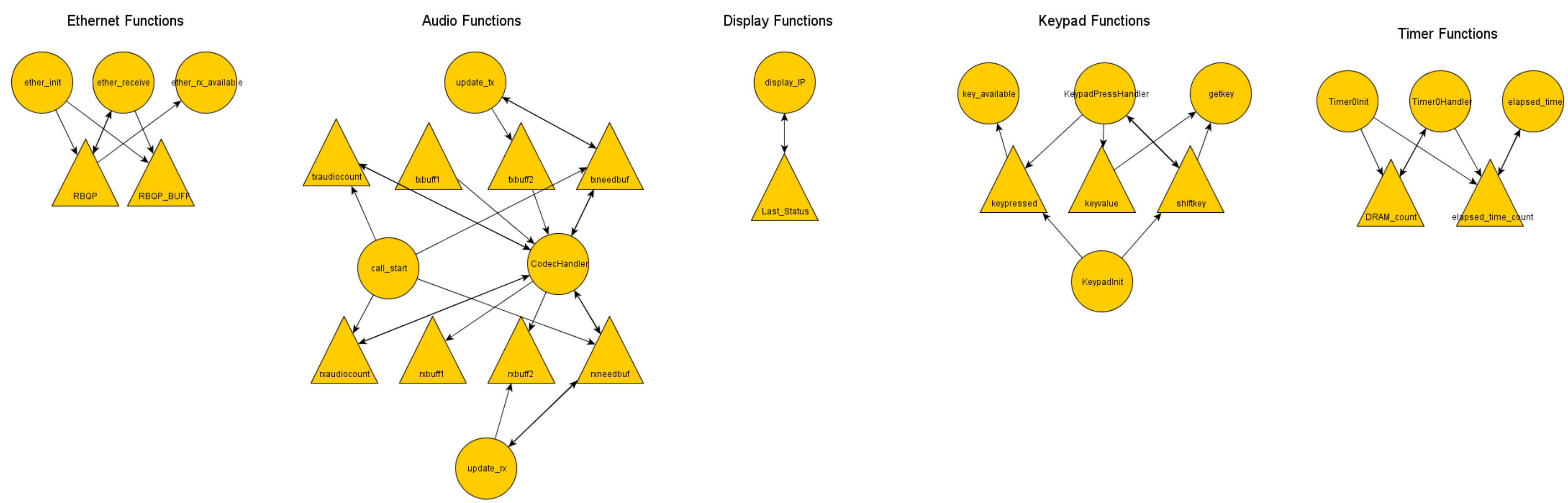
- lwIP:

  lwiP is the code that runs the majority of the system's Ethernet. It interacts with the main loop as well as ether.s. lwIP handles all the abstraction layers of Ethernet except the application layer (handled by main loop) and the lowest layer (handled by ether.s).



Interaction of Software Functions

ARM VOIP VARIABLE SHARING



Explanation of Function Diagrams:

The system is initialized by the file crt0.s. In crt0.s, chip selects are set up, the master clock is established, and the initialization functions of the peripherals are called. Finally, crt0.s branches to the main loop located in mainloop.c. Mainloop.c then proceeds to run the rest of the system. The main loop constantly checks whether a key press is available by calling key_available. If key_available indicates there is a key press, the main loop acquires the key code through getkey. The main loop then handles the key code and outputs the status of the system to the display through calls to display_status, display_memory_addr, and display_IP. The main loop also initializes the systems Ethernet through a call to ether_init. Once Ethernet is initialized, the main loop sends out an ARP packet over internet through calls to lwIP and ether_transmit. The main loop also constantly scans to determine whether Ethernet data has been received by calling ether_rx_available. If a buffer has been received, the main loop calls ether_receive to package that data in a more suitable format. Finally, the mian loop initializes SSC0 interrupts when it enters a calling state through a call to call_start. Once interrupts are initialized, codec_handler will be called during every frame sync interrupt and output and input one half word (16 bits) to/from the codec. During this time, the main loop frequently calls update_tx and update_rx to determine if new audio buffers are needed by the code in audio.s. The main loop also calls elapsed_time to determine when to send out Ethernet acknowledges. Timer0Handler uses the systems timer to be called every millisecond via interrupt. When called, Timer0Handler increments the value returned by elapsed_time by one and refreshes a portion of the systems DRAM by performing reads on a chunk of column addresses. Each call to Timer0Handler results in a change in which chunk of DRAM is refreshed so that every 4 milliseconds, the entire DRAM is refreshed.

System Code:

Note that code is written in assembly language, c language, or make language. Assembly files are of type .s, c files are of type .c, and MakeFile is the only file written in make. Include files in assembly are of file type .inc and include files for c or of file type .h. Also note that the copying of files from notepad ++ to Microsoft Word causes some distortion of comments, it is almost certainly easier to view software files directly from the included .zip containing all system code.

Table of Contents:

25. ethernet.h
26. interface.h
27. ipproc.c
28. keyproc.c
29. keyproc.h
30. mainloop.c
31. memproc.c
32. tcpconn.c
33. tcpconn.h
34. voipdefs.h

LWIP CODE:

35. lwipopts.h

Other LWIP code was unchanged and can be found in the included .zip of all system code

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@                                        @

@ armvoip.inc                                @

@                                        @

@ General VOIP project constants.                     @

@                                        @

@ Revision History:                              @

@                                        @

@   2012/2/6    Josh Fromm  File Created                @

@   2012/2/7    Josh Fromm  Constants added              @

@   2012/2/16   Josh Fromm  Constants added              @

@   2012/3/7    Josh Fromm  DRAM constants added           @

@   2012/3/29   Josh Fromm  Comments updated             @

@                                        @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


.equ IRQ_STACK_SIZE, 0x80      @memory space allocated to IRQ

.equ SVC_STACK_SIZE, 0x80      @memory space allocated to the stack

.equ TOP_STACK, 0x20020000     @last address in SRAM (used as top of the stack)


@ Control Words


.equ MCKSTARTCON, 0x1                @value to write to mck at initialization
```

```
.equ PLLARCON, 0x200EBF02    @ Value to write to PLL A control register to
                             @ set clock value to 75 MHz
.equ MORCON, 0xFF01      @ Value to write to main oscillator register
.equ PRCCLKCON, 0x1      @ Value to write to enable processor clock
.equ MCKCON, 0x00000102  @ Value to write to enable master clock
.equ NCS0CON, 0x0000408C @ Value to write to chip select zero control register

.equ NCS1CON, 0x00003085 @ Value to write to chip select one control register

.equ NCS2CON, 0x000340AA @ Value to write to chip select two control register

.equ NCS3CON, 0x74034098 @ Value to write to chip select three control registers

@ General Constants

.equ LOW_BYTE_MASK, 0xFF @constant used to mask all bits except low byte
.equ LOW_BIT_MASK, 0x1  @constant used to mask all bits except low bit
.equ TRUE,        0xFF @constant used to indicate true
.equ FALSE,       0x0  @constant used to indicate false
.equ NULL,         0x00 @ASCII value for NULL
.equ SRAM_SIZE,        0x20000          @memory space in SRAM
.equ ROM_START, 0x10000000 @first address of ROM
```

```
.equ SRAM_START, 0x20000000 @first address of SRAM

.equ DRAM_START, 0x40000000 @first address of DRAM

.equ WAIT_TIME, 0x1000      @number of cycles for a wait loop

.equ PA4_VAL,   0x10       @value to write to a control register that corresponds

                           @to peripheral 4.

.equ PMC_SCER_PA4, 0x201    @value to write to PMC SCER to enable PCK1

.equ PMC_PCER_PIOA, 0x4      @value to write to PMC_PCER to enable PIOA clocks

.equ PMC_PCER_AUDIO, 0x4008  @value used to provide clocks to peripherals used

                            @in audio functions

.equ PCK1_VAL,      0xE     @value to write to set programmable clock 1 to be

                           @1/4th of the master clock.
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                         @
@ at91rm9200.inc                                          @
@                                                         @
@ General control register address definitions for the Atmel AT91RM9200      @
@                                                         @
@ Revision History:                                       @
@                                                         @
@   2008/04/23  Arthur Chang    Initial Revision                      @
@   2010/02/01  Joseph Schmitz  Modified file I received from Arthur Chang   @
@                               to distribute to students              @
@   2011/02/13  Glen George     Cleaned up commenting, removed definitions   @
@                               not related to the AT91RM9200 chip.      @
@   2012/3/29   Josh Fromm      Relevant constants moved in from other       @
@                               include files.                      @
@                                                         @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


@ Clock Definitions


   .equ   PMC_MCKR,       0xfffffc30        @ Master Clock
```

```
@ System Mode Definitions

   .equ   ARM_MODE_USR,    0x10          @ User Mode
   .equ   ARM_MODE_FIQ,    0x11          @ FIQ Mode
   .equ   ARM_MODE_IRQ,    0x12          @ IRQ Mode
   .equ   ARM_MODE_SVC,    0x13          @ Supervisor Mode
   .equ   ARM_MODE_ABT,    0x17          @ Abort Mode
   .equ   ARM_MODE_UND,    0x1B          @ Undefined Mode
   .equ   ARM_MODE_SYS,    0x1F          @ System Mode
   .equ   I_BIT,           0x80          @ Interrupts disabled
   .equ   F_BIT,           0x40          @ Fast Interrupts disabled


@ Static Memory Controller Definitions

   .equ    SMCBase,       0xFFFFFF70       @ base address

   .equ    SMC_CSR0,      SMCBase + 0x0    @ Chip Select 0 Register
   .equ    SMC_CSR1,      SMCBase + 0x4    @ Chip Select 1 Register
   .equ    SMC_CSR2,      SMCBase + 0x8    @ Chip Select 2 Register
   .equ    SMC_CSR3,      SMCBase + 0xC    @ Chip Select 3 Register
   .equ    SMC_CSR4,      SMCBase + 0x10   @ Chip Select 4 Register
   .equ    SMC_CSR5,      SMCBase + 0x14   @ Chip Select 5 Register
```

```
    .equ    SMC_CSR6,      SMCBase + 0x18     @ Chip Select 6 Register
    .equ    SMC_CSR7,      SMCBase + 0x1C     @ Chip Select 7 Register


@ Parallel I/O A Definintions

    .equ    PIOABase,      0xFFFFF400         @ base address

    .equ    PIOA_PER,      PIOABase           @ PIO Enable
    .equ    PIOA_PDR,      PIOABase + 0x4     @ PIO Disable
    .equ    PIOA_ODR,      PIOABase + 0x14    @ output disable
    .equ    PIOA_PDSR,     PIOABase + 0x3C    @ Pin Data Status
    .equ    PIOA_IMR,      PIOABase + 0x8     @ IRQ Status
    .equ    PIOA_ASR,      PIOABase + 0x70    @ PIO Peripheral A Select
    .equ    PIOA_BSR,      PIOABase + 0x74    @ PIO Peripheral B Select


@ Parallel I/O B Definintions

    .equ    PIOBBase,      0xFFFFF600         @ base address

    .equ    PIOB_PDR,      PIOBBase + 0x4     @ PIO Disable
    .equ    PIOB_ASR,      PIOBBase + 0x70    @ PIO Peripheral A Select
    .equ    PIOB_BSR,      PIOBBase + 0x74    @ PIO Peripheral B Select
```

```
    .equ    PIOB_PER,       PIOBBase
    .equ    PIOB_CODR,      PIOBBase + 0x34
    .equ    PIOB_OER,       PIOBBase + 0x10
    .equ    PIOB_SODR,      PIOBBase + 0x30

@ Parallel I/O C Definintions

    .equ    PIOCBase,       0xFFFFF800        @ base address

    .equ    PIOC_PER,       PIOCBase          @ PIO Enable
    .equ    PIOC_PDR,       PIOCBase + 0x4    @ PIO Disable
    .equ    PIOC_ODR,       PIOCBase + 0x14   @ output disable
    .equ    PIOC_OER,       PIOCBase + 0x10   @ output enable
    .equ    PIOC_CODR,      PIOCBase + 0x34   @ clear output
    .equ    PIOC_PDSR,      PIOCBase + 0x3C   @ Pin Data Status
    .equ    PIOC_IMR,       PIOCBase + 0x8    @ IRQ Status
    .equ    PIOC_IER,       PIOCBase + 0x40   @ interrupt enable
    .equ    PIOC_ASR,       PIOCBase + 0x70   @ PIO Peripheral A Select
    .equ    PIOC_BSR,       PIOCBase + 0x74   @ PIO Peripheral B Select
    .equ    PIOC_SODR,      PIOCBase + 0x30   @ PIO Set Output Data
    .equ    PIOC_ISR,       PIOCBase + 0x4C   @ PIO interrupt status register
```

@ Advanced Interrupt Ccontroller Definitions

```
.equ   AICBase,       0xFFFFF000        @ base address

.equ   AIC_SMR1,      AICBase + 0x4      @ system source mode
.equ   AIC_SVR1,      AICBase + 0x84     @ system source vector
.equ   AIC_SMR4,      AICBase + 0x10     @ PIO C Source Mode
.equ   AIC_SVR4,      AICBase + 0x90     @ PIO C Source Vector
.equ   AIC_SMR14,     AICBase + 0x38     @ SSC0: Source Mode 14
.equ   AIC_SVR14,     AICBase + 0xB8     @ SSC0: Source Vector 14
.equ   AIC_SMR17,     AICBase + 0x44     @ TC0: Source Mode 17
.equ   AIC_SVR17,     AICBase + 0xC4     @ TC0: Source Vector 17
.equ   AIC_SMR18,     AICBase + 0x48     @ TC1: Source Mode 18
.equ   AIC_SVR18,     AICBase + 0xC8     @ TC1: Source Vector 18
.equ   AIC_SMR24,     AICBase + 0x60     @ EMAC: Source Mode 24
.equ   AIC_SVR24,     AICBase + 0xE0     @ EMAC: Source Vector 24
.equ   AIC_SMR25,     AICBase + 0x64     @ IRQ0: Source Mode 25
.equ   AIC_SVR25,     AICBase + 0xE4     @ IRQ0: Source Vector 25
.equ   AIC_SMR26,     AICBase + 0x68     @ IRQ1: Source Mode 26
.equ   AIC_SVR26,     AICBase + 0xE8     @ IRQ1: Source Vector 26
.equ   AIC_ISR,       AICBase + 0x108    @ Interrupt Status
.equ   AIC_IMR,       AICBase + 0x110    @ Interrupt Mask
.equ   AIC_IPR,       AICBase + 0x10C    @ Interrupt Pending
```

```
   .equ   AIC_IECR,     AICBase + 0x120    @ Interrupt Enable Command
   .equ   AIC_IDCR,     AICBase + 0x124    @ Interrupt Disable Command
   .equ   AIC_ICCR,     AICBase + 0x128    @ Interrupt Clear Command
   .equ   AIC_EOICR,    AICBase + 0x130    @ End of Interrupt Command


@ Power Management Controller Definitions

   .equ   PMCBase,      0xFFFFFC00         @ base address

   .equ   PMC_SCER,     PMCBase + 0x0      @ System Clock Enable
   .equ   PMC_PCER,     PMCBase + 0x10     @ Peripheral Clock Enable
   .equ   PMC_MOR,      PMCBase + 0x20     @ Main Oscillator register
   .equ   PMC_PLLBR,    PMCBase + 0x2C     @ PLL B Control
   .equ   PMC_PLLAR,    PMCBase + 0x28     @ PLL A Control
   .equ   PMC_MCKR,     PMCBase + 0x30     @ Master Clock
   .equ   PMC_PCK0,     PMCBase + 0x40     @ Programmable Clock 0
        .equ   PMC_PCK1,          PMCBase + 0x44
   .equ   PMC_SR,       PMCBase + 0x68     @ Status Register


@ Serial Synchronous Controller 0 Definitions

   .equ   SSC0Base,     0xFFFD0000         @ base address
```

```
    .equ   SSC0_CR,       SSC0Base + 0x0     @ Control
    .equ   SSC0_CMR,      SSC0Base + 0x4     @ Clock Mode
    .equ   SSC0_RCMR,     SSC0Base + 0x10    @ Receive Clock Mode
    .equ   SSC0_RFMR,     SSC0Base + 0x14    @ Receive Frame Mode
    .equ   SSC0_TCMR,     SSC0Base + 0x18    @ Transmit Clock Mode
    .equ   SSC0_TFMR,     SSC0Base + 0x1C    @ Transmit Frame Mode
    .equ   SSC0_RHR,      SSC0Base + 0x20    @ Receive Holding
    .equ   SSC0_THR,      SSC0Base + 0x24    @ Transmit Holding
    .equ   SSC0_SR,       SSC0Base + 0x40    @ Status
    .equ   SSC0_IER,      SSC0Base + 0x44    @ Interrupt Enable
    .equ   SSC0_IDR,      SSC0Base + 0x48    @ Interrupt Disable


@ Timer Counter 0 Definitions

    .equ   TC0Base,       0xFFFA0000         @ base address

    .equ   TC0_CCR,       TC0Base + 0x0      @ Channel Control
    .equ   TC0_CMR,       TC0Base + 0x4      @ Channel Mode
    .equ   TC0_RC,        TC0Base + 0x1C     @ Register C
    .equ   TC0_CV,        TC0Base + 0x10     @ Counter Value
    .equ   TC0_SR,        TC0Base + 0x20     @ Status Register
    .equ   TC0_IER,       TC0Base + 0x24     @ Interrupt Enable
```

```
    .equ    TC0_IDR,        TC0Base + 0x28      @ Interrupt Disable

@ Timer Counter 1 Definitions

    .equ    TC1Base,        0xFFFA0040          @ base address

    .equ    TC1_CCR,        TC1Base + 0x0       @ Channel Control
    .equ    TC1_CMR,        TC1Base + 0x4       @ Channel Mode
    .equ    TC1_RC,         TC1Base + 0x1C      @ Register C
    .equ    TC1_CV,         TC1Base + 0x10      @ Counter Value
    .equ    TC1_SR,         TC1Base + 0x20      @ Status Register
    .equ    TC1_IER,        TC1Base + 0x24      @ Interrupt Enable
    .equ    TC1_IDR,        TC1Base + 0x28      @ Interrupt Disable

@ EMAC Definitions

    .equ    EMAC_RSR,       0xFFFBC020          @ Receive Status
    .equ    EMAC_MAN,       0xFFFBC034          @ PHY Maintenance
    .equ    EMAC_HSH,       0xFFFBC090          @ Hash Address High[63:32]
    .equ    EMAC_MCOL,      0xFFFBC048          @ Multiple Collision Frame
    .equ    EMAC_ISR,       0xFFFBC024          @ Interrupt Status Register
    .equ    EMAC_IER,       0xFFFBC028          @ Interrupt Enable
```

```
.equ   EMAC_SA2H,    0xFFFBC0A4     @ Specific Address 2 High, Last 2 bytes
.equ   EMAC_HSL,     0xFFFBC094     @ Hash Address Low[31:0]
.equ   EMAC_LCOL,    0xFFFBC05C     @ Late Collision
.equ   EMAC_OK,      0xFFFBC04C     @ Frames Received OK
.equ   EMAC_CFG,     0xFFFBC004     @ Network Configuration
.equ   EMAC_SA3L,    0xFFFBC0A8     @ Specific Address 3 Low, First 4 bytes
.equ   EMAC_SEQE,    0xFFFBC050     @ Frame Check Sequence Error
.equ   EMAC_ECOL,    0xFFFBC060     @ Excessive Collision
.equ   EMAC_ELR,     0xFFFBC070     @ Excessive Length Error
.equ   EMAC_SR,      0xFFFBC008     @ Network Status
.equ   EMAC_RBQP,    0xFFFBC018     @ Receive Buffer Queue Pointer
.equ   EMAC_CSE,     0xFFFBC064     @ Carrier Sense Error
.equ   EMAC_RJB,     0xFFFBC074     @ Receive Jabber
.equ   EMAC_USF,     0xFFFBC078     @ Undersize Frame
.equ   EMAC_IDR,     0xFFFBC02C     @ Interrupt Disable
.equ   EMAC_SA1L,    0xFFFBC098     @ Specific Address 1 Low, First 4 bytes
.equ   EMAC_IMR,     0xFFFBC030     @ Interrupt Mask
.equ   EMAC_FRA,     0xFFFBC040     @ Frames Transmitted OK
.equ   EMAC_SA3H,    0xFFFBC0AC     @ Specific Address 3 High, Last 2 bytes
.equ   EMAC_SA1H,    0xFFFBC09C     @ Specific Address 1 High, Last 2 bytes
.equ   EMAC_SCOL,    0xFFFBC044     @ Single Collision Frame
.equ   EMAC_ALE,     0xFFFBC054     @ Alignment Error
.equ   EMAC_TAR,     0xFFFBC00C     @ Transmit Address
.equ   EMAC_SA4L,    0xFFFBC0B0     @ Specific Address 4 Low, First 4 bytes
```

```
.equ   EMAC_SA2L,     0xFFFBC0A0        @ Specific Address 2 Low, First 4 bytes

.equ   EMAC_TUE,      0xFFFBC068        @ Transmit Underrun Error

.equ   EMAC_DTE,      0xFFFBC058        @ Deferred Transmission Frame

.equ   EMAC_TCR,      0xFFFBC010        @ Transmit Control

.equ   EMAC_CTL,      0xFFFBC000        @ Network Control

.equ   EMAC_SA4H,     0xFFFBC0B4        @ Specific Address 4 High, Last 2 bytes

.equ   EMAC_CDE,      0xFFFBC06C        @ Code Error

.equ   EMAC_SQEE,     0xFFFBC07C        @ SQE Test Error

.equ   EMAC_TSR,      0xFFFBC014        @ Transmit Status

.equ   EMAC_DRFC,     0xFFFBC080        @ Discarded RX Frame



@ General Definitions


.equ   WORD_SIZE,     0x4               @ size of a word in bytes

.equ   HALFWORD_SIZE, 0x2               @ size of a halfword in bytes

.equ   BYTE_SIZE,     0x1               @ size of a byte in bytes

.equ   BITSPBYTE,     0x8               @ number of bits in a byte

.equ   BITSPHW,       0x10              @ number of bits in a half word
```

```
@ audio.inc
@ This file contains the constants used by the functions that run the VOIP
@ system's audio input and output.
@ Revision History:
@
@   2012/2/24    Josh Fromm  Initial Revision


@ SSC register definitions


    .equ    SSC0_CR_VAL,    0x101       @allow data to be transmitted and received
    .equ    SSC0_CMR_VAL,   0x12        @divides masterclock to get as close to
                                        @2 Mhz as possible
    .equ    SSC0_RCMR_VAL,  0x7F000105  @set period to be one frame sync every
                                        @256 cycles. Data shifted in on falling edge
    .equ    SSC0_RFMR_VAL,  0x8F        @MSB first, data length is half words
    .equ    SSC0_TCMR_VAL,  0x7F000424  @period set to 256 cycles, data shifted out
                                        @on rising edge
    .equ    SSC0_TFMR_VAL,  0x20008F    @data length is half words, MSB first,
                                        @frame sync type is positive pulse
    .equ    SSC0_IER_VAL,   0x400       @value to write to enable frame sync interrupts
    .equ    SSC0_IDR_VAL,   0x400       @value to write to disable frame sync interrupts
    .equ    AIC_SMR14_VAL,  0x07        @ssc0 is given high interrupt priority
    .equ    AIC_IECR14_VAL, 0x4000      @value to write to enable ssc0 interrupts
                                        @on the AIC
```

```
@ General Audio Definitions
    .equ    AUDIO_BUFLEN,   0x200       @length of buffers in bytes
    .equ    SSC0_PINS,      0xF         @bits corresponding to PIOB pins used
                                        @by SSC0
    .equ    PIN_25,         0x2000000   @bit corresponding to PIOC25
    .equ    VOLUME_CON_BITS, 0x3        @bits taken up by volume control in
                                        @audio data output to codec
    .equ    VOLUME_CON_VAL,  0x3        @value ranging between 0 and 7 that
                                        @determines volume of audio data processed
                                        @by the systems codec
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                          @
@ audio.s                                  @
@                                          @
@ Initialization, event handling, and basic functions of VOIP audio.    @
@                                          @
@ Table of Contents:                        @
@   1. ssc0_init: Function called to initialize the registers needed for SSC to@
@      to function.                        @
@   2. call_start: Initializes SSC0 interrupts so that audio data can be input @
@      and output.                         @
@   3. call_halt: Ends SSC0 interrupts so that no audio data is input or    @
@      output.                             @
@   4. CodecHandler: Called during every frame sync interrupt. Inputs and    @
@      outputs a half word of audio data. The function also handles swapping of@
@      filled or empty audio buffers.                @
@   5. update_tx: Checks whether a new buffer of audio data to transmit is    @
@      needed.                             @
@   6. update_rx: Checks if a new buffer to store incoming audio data is needed@
@                                          @
@ Revision History:                        @
@                                          @
@   2012/2/24    Josh Fromm  Initial Revision              @
```

```
@   2012/3/4     Josh Fromm  Code updated to actually work                   @

@   2012/3/29    Josh Fromm  Comments updated                            @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

.include    "at91rm9200.inc"

.include    "armvoip.inc"

.include    "audio.inc"

.text

.arm

.align 2

@@@ Audio initialization

@ ssc0_init()

@

@ Description:      ssc0_init sets up the many registers needed for the CPUs

@                 synchronous serial controller to function properly. ssc0_init

@                 also performs a hardware reset on the system's codec.

@

@ Operation:       Moves many control words into appropriate registers to set up

@                 the SSC and uses a wait loop to allow for an appropriate
```

```
@               hardware reset.
@
@ Arguments:       None.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and control words.
@               r1 - contains addresses and control words.
@ Shared Variables: None.
@ Global Variables: None.
@ Input:           None.
@ Output:          None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;       2.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
```

```
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated


.global ssc0_init
ssc0_init:

    PUSH {r0, r1}          @save used registers

    LDR r0, =PMC_PCER      @enable ssc0 clock and piob clock
    LDR r1, =PMC_PCER_AUDIO
    STR r1, [r0]

    LDR r0, =PIOB_PDR      @disable the pins needed by SSC0
    LDR r1, =SSC0_PINS
    STR r1, [r0]

    LDR r0, =PIOB_ASR      @shift control of those pins to peripheral A
    STR r1, [r0]

    LDR r0, =SSC0_CMR      @set up clock mode register
    LDR r1, =SSC0_CMR_VAL
    STR r1, [r0]
```

```
LDR r0, =SSC0_RCMR      @set up receive clock mode register

LDR r1, =SSC0_RCMR_VAL

STR r1, [r0]


LDR r0, =SSC0_RFMR      @set up receive frame mode register

LDR r1, =SSC0_RFMR_VAL

STR r1, [r0]


LDR r0, =SSC0_TCMR      @set up transmit clock mode register

LDR r1, =SSC0_TCMR_VAL

STR r1, [r0]


LDR r0, =SSC0_TFMR      @set up transmit frame mode register

LDR r1, =SSC0_TFMR_VAL

STR r1, [r0]


LDR r0, =AIC_SMR14      @set priority for ssc0 interrupts

LDR r1, =AIC_SMR14_VAL

STR r1, [r0]


LDR r0, =AIC_SVR14      @cause ssc0 interrupts to jump to CodecHandler

LDR r1, =CodecHandler

STR r1, [r0]
```

```
LDR r0, =AIC_IECR       @set up ssc0 interrupts to trigger AIC interrputs

LDR r1, =AIC_IECR14_VAL

STR r1, [r0]


LDR r0, =SSC0_CR        @enable data to be transmitted and received over ssc0

LDR r1, =SSC0_CR_VAL

STR r1, [r0]


@perform a codec reset


LDR r0, =PIOC_PER       @enable the reset pin

LDR r1, =PIN_25

STR r1, [r0]


LDR r0, =PIOC_OER       @allow reset pin to output

STR r1, [r0]


LDR r0, =PIOC_CODR      @force reset pin low to indicate hardware reset

STR r1, [r0]


@now wait for a while to make sure the reset works


LDR r1, =WAIT_TIME      @set up counter for wait loop
```

```
    LDR r0, =0x0          @when counter is zero, wait loop can end
ssc0_init_loop:
    ADD r0, r0, #0x1      @increment counter by 1
    CMP r0, r1            @if counter is zero, wait loop is over
    BNE ssc0_init_loop    @otherwise keep waiting

    LDR r0, =PIOC_SODR    @set reset pin back to end reset
    LDR r1, =PIN_25
    STR r1, [r0]

    POP {r0, r1}          @restore registers and return

    BX LR



@ call_start(rx_buffer_addr)
@
@ Description:    call_start initializes the shared variables used by audio
@                 functions and also starts ssc0 interrupts so that audio can
@                 be input and output.
@
@ Operation:      Sets shared variables to their in initial value and writes
@                 to the SSC0 interrupt enable register to trigger interrupts
@                 on frame sync signals.
```

```
@
@ Arguments:        r0 - address of first receive buffer.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@ Shared Variables: rxbuff1, txaudiocount, rxaudiocount, txneedbuf, rxneedbuf.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  rxbuff1.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
```

```
@               3/7/2012    Josh Fromm  Updated to function better
@               3/29/2012   Josh Fromm  Comments updated


.global call_start
call_start:    @sets up registers and variables needed for codec and SSC to run
   PUSH {r0, r1}

   LDR r1, =rxbuff1       @store passed receive buffer address
   STR r0, [r1]

   LDR r0, =txaudiocount   @set transmit buffer to be full (prevents sending
                           @until a new transmit buffer is passed)
   LDR r1, =AUDIO_BUFLEN
   STR r1, [r0]

   LDR r0, =rxaudiocount   @set receive buffer to empty
   LDR r1, =0x0
   STR r1, [r0]

   LDR r0, =txneedbuf      @indicate a new transmit buffer is needed
   LDR r1, =TRUE
   STR r1, [r0]
```

```
    LDR r0, =rxneedbuf      @indicate a new receive buffer is needed
    LDR r1, =TRUE
    STR r1, [r0]

    LDR r0, =SSC0_IER       @enable frame sync interrupts to occur.
    LDR r1, =SSC0_IER_VAL
    STR r1, [r0]

    POP {r0, r1}            @restore registers and return
    BX LR



@ call_halt()
@
@ Description:      call_halt disables frame sync interrupts to that no more
@                   audio data is input or output.
@
@ Operation:        Writes to the SSC0 interrupt disable register to turn off
@                   frame sync interrupts.
@
@ Arguments:        None.
@
@ Return Values:    None.
@
```

```
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@ Shared Variables: None.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated


.global call_halt
call_halt:                  @end the current call
```

```
        PUSH {r0, r1}           @save used registers


        LDR r0, =SSC0_IDR       @end ssc interrupts
        LDR r1, =SSC0_IDR_VAL
        STR r1, [r0]


        POP {r0, r1}            @restore registers and return
        BX LR


@ CodecHandler()
@
@ Description:      CodecHandler is the event handler called when a frame sync
@                   interrupt on SSC0 occurs. Codec handler's basic purpose is
@                   to read the half word in SSC0's receive holding register
@                   and move that half word into a buffer of other received values.
@                   CodecHandler then takes the next half word in an output buffer
@                   and writes it to SSC0's transmit holding register. SSC0 also
@                   checks whether either of these buffers is filled and handles
@                   such exceptions accordingly.
@
@ Operation:        CodecHandler processes an SSC0 receive first, then transmits
@                   data; however, input and output function essentially the
@                   same way. CodecHandler first checks whether the receive or
@                   transmit buffer is full or empty respectively. If so,
```

```
@               CodecHandler switches the active buffer to the back up buffer
@               and indicates a new buffer is needed. If there is no
@               active buffer, received data is dropped or a 0 is transmitted.
@               If data does not need to be dropped, CodecHandler inputs and
@               outputs data normally and increments counters indicating how
@               how many bytes in the current buffer have been input or output.
@
@
@ Arguments:       None.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - varies by process, often used as address storage
@               r1 - varies by process, often used as control word or data
@               r2 - values loaded from addresses
@ Shared Variables: rxaudiocount, txaudiocount, rxneedbuf, txneedbuf, txbuff1,
@               txbuff2, rxbuff1, rxbuff2.
@ Global Variables: None.
@ Input:           Audio data from codec moved to buffer.
@ Output:          Outputs audio data from buffer to codec.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;       2.
```

```
@
@ Algorithms:      None.
@
@ Data Structures:  buffers located at rxbuff1, rxbuff2, txbuff1, txbuff2.
@
@ Known Bugs:      None.
@ Limitations:     None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated



.global CodecHandler
CodecHandler:            @input or output the next halfword of data
    SUB  LR, LR, #4      @first correct systems pipeline
    STMFD SP!, {LR}
    PUSH {r0, r1, r2}

RxFullCheck:             @if buffers are full, swap them or drop data
    LDR r0, =rxaudiocount
    LDR r1, [r0]
    CMP r1, #AUDIO_BUFLEN   @compare to the total size of a buffer
    BNE Handle_Rx           @otherwise, read and write next byte
```

```
@       BEQ RxBuff_Swap


RxBuff_Swap:                @active buffer is full and must be swapped with backup
        LDR r0, =rxneedbuf        @if this is the second buffer, data must be dropped
        LDR r1, [r0]
        CMP r1, #TRUE       @if neebuf is true then a backup buffer isnt provided
        BEQ RxDrop

        @otherwise switch buffers normally

        LDR r0, =rxaudiocount     @first reset count of bytes written to buffer
        LDR r1, =0x0
        STR r1, [r0]

        LDR r0, =rxbuff1       @set rxbuff1 to contain the address of the next
                        @buffer
        LDR r1, =rxbuff2
        LDR r2, [r1]
        STR r2, [r0]

        LDR r0, =rxneedbuf      @indicate that rx needs a new buffer
        LDR r1, =TRUE
        STR r1, [r0]
        B   Handle_Rx          @then proceed to read and output data
```

```
RxDrop:                 @ran out of buffer space, must drop data
     LDR  r0, =SSC0_RHR  @read from receive register and do nothing with data
     LDRH r1, [r0]

     B   TxFullCheck     @then go on to handle tx

Handle_Rx:              @add next received byte to buffer
    LDR r0, =rxbuff1     @first determine where next received byte will be stored
    LDR r2, [r0]        @value of base address of buffer loaded
    LDR r0, =rxaudiocount  @base address offset by number of bytes already stored
    LDR r1, [r0]
    ADD r2, r2, r1      @once base and offset are added, r2 contains where
                  @to store next read byte

    LDR r0, =SSC0_RHR    @now read next receive half word
    LDRH r1, [r0]
    STRH r1, [r2]        @and store that value in the buffer

    LDR r0, =rxaudiocount    @update count of bytes written to buffer
    LDR r1, [r0]
    ADD r1, r1, #0x2
    STR r1, [r0]
```

```
	@then continue to handle the transmit


TxFullCheck:		@first determine if current transmit buffer has been
			@completely transmitted
	LDR r0, =txaudiocount
	LDR r1, [r0]
	CMP r1, #AUDIO_BUFLEN	@compare with total size of audio buffer
	BNE Handle_Tx		@if buffer isnt full, input normally
@	BEQ TxBuff_Swap		@if it is full, swap to backup buffer


TxBuff_Swap:		@switch from buffer 1 to buffer 2
	LDR r0, =txneedbuf
	LDR r1, [r0]
	CMP r1, #TRUE
	BEQ TxDrop		@if already in buffer 2, drop data


	@otherwise swap buffers normally


	LDR r0, =txaudiocount	@first reset number of bytes stored in buffer
	LDR r1, =0x0
	STR r1, [r0]


	LDR r0, =txbuff1	@set txbuff1 to contain the address of the next
				@buffer
```

```
    LDR r1, =txbuff2
    LDR r2, [r1]
    STR r2, [r0]

    LDR r0, =txneedbuf      @indicate that tx needs a new buffer
    LDR r1, =TRUE
    STR r1, [r0]
    B   Handle_Tx           @then proceed to read and output data

TxDrop:                     @out of data to transmit, must send a blank
    LDR r0, =SSC0_THR
    LDR r1, =0x0
    STRH r1, [r0]       @output a blank half word

    B CodecHandlerDone  @once sent function is done

Handle_Tx:                  @first output next halfword in buffer
    LDR r0, =txbuff1     @load base address of current buffer
    LDR r2, [r0]        @address of buffer loaded

    LDR r0, =txaudiocount   @determine which halfword in the buffer should be
                    @output
    LDR r1, [r0]
```

```
    ADD r2, r2, r1        @base address of buffer shifted by the number of
                          @bytes already read from tx buffer
    LDRH r0, [r2]         @load the next halfword to output

    LSL r0, #VOLUME_CON_BITS        @set volume control
    ADD r0, r0, #VOLUME_CON_VAL

    LDR r1, =SSC0_THR
    STRH r0, [r1]         @output next half word in buffer

    LDR r0, =txaudiocount   @update number of bytes read from buffer
    LDR r1, [r0]
    ADD r1, r1, #0x2
    STR r1, [r0]

    @function is now done

CodecHandlerDone:

    LDR r0, =SSC0_SR    @read ssc0 status register to reset it
    LDR r1, [r0]

    LDR r0, =AIC_EOICR   @signal end of interrupt
    LDR r1, =TRUE
```

```
    STR r1, [r0]

    POP {r0, r1, r2}    @restore registers and return
    LDMFD SP!, {PC}^



@ update_tx(new_buffer_addr)
@
@ Description:      update_tx checks whether a new buffer of outgoing audio
@                   data is needed. A buffer is needed when one or both of
@                   the stored buffers has been emptied of audio data. If this is the
@                   case, the function accepts the passed buffer address and
@                   returns true. If no buffer is needed the
@                   function returns false.
@
@ Operation:        update_tx simply checks the statuses
@                   of shared variables to determine whether a new buffer is
@                   needed, then updates shared variables to indicate a new buffer
@                   has been accepted, or does nothing to indicate no new buffer
@                   has been accepted.
@
@ Arguments:        r0 - address of available transmit buffer.
@
@ Return Values:    r0 - TRUE if passed buffer was accepted, FALSE if not.
```

@

@ Local Variables: r0 - contains addresses and variable values

@   r1 - contains addresses and variable values

@   r2 - contains addresses and variable values

@ Shared Variables: txneedbuf, txbuff2.

@ Global Variables: None.

@ Input: None.

@ Output: None.

@ Error Handling: None.

@

@ Registers Changed: r0.

@ Stack Depth;   2.

@

@ Algorithms: None.

@

@ Data Structures: buffer stored at address in txbuff2.

@

@ Known Bugs: None.

@ Limitations: None.

@

@ Revision Hisotry: 2/28/2012   Josh Fromm Outline Created

@   3/7/2012   Josh Fromm Updated to function better

@   3/29/2012   Josh Fromm Comments updated

```
.global update_tx
update_tx:
      PUSH {r1, r2}      @save used registers

      LDR r1, =txneedbuf  @determine if a new buffer is needed
      LDR r2, [r1]
      CMP r2, #TRUE
      BNE update_tx_nobuf @if no buffer is needed, return false
@      BEQ new_tx_buff     @if a buffer is needed, move to buffer handling
                  @process

new_tx_buff:             @accept the new buffer

      LDR r1, =txbuff2    @handle by adding passed buffer
                  @to buffer two spot
      STR r0, [r1]

      LDR r1, =txneedbuf  @indicate a buffer is no longer needed
      LDR r2, =FALSE
      STR r2, [r1]

      LDR r0, =TRUE      @indicate update was needed
```

```
        B update_tx_done    @function is done



update_tx_nobuf:            @a new buffer is not needed
        LDR r0, =FALSE      @set return value to indicate an update was un needed
@       B update_tx_done    @function is done

update_tx_done:             @pop registers and return

        POP {r1, r2}
        BX LR



@ update_rx(new_buffer_addr)
@
@ Description:      update_rx checks whether a new buffer to store incoming audio
@                   data in needed. A buffer is needed when one or both of
@                   the stored buffers has been filled with audio data. If this is the
@                   case, the function accepts the passed buffer address and
@                   returns true. If no buffer is needed the
@                   function returns false.
@
@ Operation:        Iupdate_tx checks the statuses of shared variables to
@                   determine whether a new buffer is needed, then updates
```

```
@               shared variables to indicate a new buffer has been accepted,
@               or does nothing to indicate no new buffer has been accepted.
@
@ Arguments:       r0 - address of available receive buffer.
@
@ Return Values:   r0 - TRUE if passed buffer was accepted, FALSE if not.
@
@ Local Variables: r0 - contains addresses and variable values
@               r1 - contains addresses and variable values
@               r2 - contains addresses and variable values
@ Shared Variables: rxneedbuf, rxbuff2.
@ Global Variables: None.
@ Input:          None.
@ Output:         None.
@ Error Handling:  None.
@
@ Registers Changed: r0.
@ Stack Depth;      2.
@
@ Algorithms:      None.
@
@ Data Structures:  buffer stored at address in rxbuff2.
@
@ Known Bugs:      None.
```

```
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                 3/7/2012     Josh Fromm  Updated to function better
@                 3/29/2012    Josh Fromm  Comments updated

.global update_rx
update_rx:
    PUSH {r1, r2}          @save used registers
    LDR r1, =rxneedbuf      @check if a new buffer is needed
    LDR r2, [r1]
    CMP r2, #TRUE
    BNE update_rx_nobuf     @if not, end function
@      BEQ new_rx_buff        @otherwise get the new buffer

new_rx_buff:
    LDR r1, =rxbuff2        @store passed pointer in spot allocated to
                            @back up buffer
    STR r0, [r1]

    LDR r1, =rxneedbuf      @indicate a buffer is no longer needed
    LDR r2, =FALSE
    STR r2, [r1]
```

```
    LDR r0, =TRUE        @set output to show that the buffer was taken

    B update_rx_done     @function can end

update_rx_nobuf:         @no update is needed
    LDR r0, =FALSE       @set return value to show no update was needed
@   B update_rx_done

update_rx_done:          @restore registers and return
    POP {r1, r2}
    BX LR



@ The data segment

.data
txbuff1:    @contains address of active transmit buffer
    .word  '?'
txbuff2:    @contains address of backup transmit buffer
    .word  '?'
rxbuff1:    @contains address of active receive buffer
    .word  '?'
rxbuff2:    @contains address of backup receive buffer
```

```
    .word   '?'
txaudiocount:   @number of bytes transmitted from current transmit buffer
    .word   '?'
rxaudiocount:   @number of bytes moved into current receive buffer
    .word   '?'
txneedbuf:      @whether a new transmite buffer is needed or not
    .word   '?'
rxneedbuf:      @whether a new receive buffer is needed or not
    .word   '?'

.end
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                   @
@ audio.s                                             @
@                                                   @
@ Initialization, event handling, and basic functions of VOIP audio.          @
@                                                   @
@ Table of Contents:                                        @
@   1. ssc0_init: Function called to initialize the registers needed for SSC to@
@      to function.                                    @
@   2. call_start: Initializes SSC0 interrupts so that audio data can be input @
@      and output.                                     @
@   3. call_halt: Ends SSC0 interrupts so that no audio data is input or       @
@      output.                                         @
@   4. CodecHandler: Called during every frame sync interrupt. Inputs and      @
@      outputs a half word of audio data. The function also handles swapping of@
@      filled or empty audio buffers.                            @
@   5. update_tx: Checks whether a new buffer of audio data to transmit is     @
@      needed.                                         @
@   6. update_rx: Checks if a new buffer to store incoming audio data is needed@
@                                                   @
@ Revision History:                                        @
@                                                   @
@   2012/2/24    Josh Fromm  Initial Revision                          @
```

```
@   2012/3/4     Josh Fromm  Code updated to actually work                   @

@   2012/3/29    Josh Fromm  Comments updated                              @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


.include    "at91rm9200.inc"

.include    "armvoip.inc"

.include    "audio.inc"


.text

.arm

.align 2


@@@ Audio initialization




@ ssc0_init()

@

@ Description:      ssc0_init sets up the many registers needed for the CPUs

@                   synchronous serial controller to function properly. ssc0_init

@                   also performs a hardware reset on the system's codec.

@

@ Operation:        Moves many control words into appropriate registers to set up

@                   the SSC and uses a wait loop to allow for an appropriate
```

@ hardware reset.

@

@ Arguments: None.

@

@ Return Values: None.

@

@ Local Variables: r0 - contains addresses and control words.

@ r1 - contains addresses and control words.

@ Shared Variables: None.

@ Global Variables: None.

@ Input: None.

@ Output: None.

@ Error Handling: None.

@

@ Registers Changed: None.

@ Stack Depth; 2.

@

@ Algorithms: None.

@

@ Data Structures: None.

@

@ Known Bugs: None.

@ Limitations: None.

@

```
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                  3/7/2012     Josh Fromm  Updated to function better
@                  3/29/2012    Josh Fromm  Comments updated


.global ssc0_init
ssc0_init:

   PUSH {r0, r1}          @save used registers

   LDR r0, =PMC_PCER       @enable ssc0 clock and piob clock
   LDR r1, =PMC_PCER_AUDIO
   STR r1, [r0]

   LDR r0, =PIOB_PDR       @disable the pins needed by SSC0
   LDR r1, =SSC0_PINS
   STR r1, [r0]

   LDR r0, =PIOB_ASR       @shift control of those pins to peripheral A
   STR r1, [r0]

   LDR r0, =SSC0_CMR       @set up clock mode register
   LDR r1, =SSC0_CMR_VAL
   STR r1, [r0]
```

```
LDR r0, =SSC0_RCMR      @set up receive clock mode register
LDR r1, =SSC0_RCMR_VAL
STR r1, [r0]

LDR r0, =SSC0_RFMR      @set up receive frame mode register
LDR r1, =SSC0_RFMR_VAL
STR r1, [r0]

LDR r0, =SSC0_TCMR      @set up transmit clock mode register
LDR r1, =SSC0_TCMR_VAL
STR r1, [r0]

LDR r0, =SSC0_TFMR      @set up transmit frame mode register
LDR r1, =SSC0_TFMR_VAL
STR r1, [r0]

LDR r0, =AIC_SMR14      @set priority for ssc0 interrupts
LDR r1, =AIC_SMR14_VAL
STR r1, [r0]

LDR r0, =AIC_SVR14      @cause ssc0 interrupts to jump to CodecHandler
LDR r1, =CodecHandler
STR r1, [r0]
```

```
LDR r0, =AIC_IECR       @set up ssc0 interrupts to trigger AIC interrputs

LDR r1, =AIC_IECR14_VAL

STR r1, [r0]


LDR r0, =SSC0_CR        @enable data to be transmitted and received over ssc0

LDR r1, =SSC0_CR_VAL

STR r1, [r0]


@perform a codec reset


LDR r0, =PIOC_PER       @enable the reset pin

LDR r1, =PIN_25

STR r1, [r0]


LDR r0, =PIOC_OER       @allow reset pin to output

STR r1, [r0]


LDR r0, =PIOC_CODR      @force reset pin low to indicate hardware reset

STR r1, [r0]


@now wait for a while to make sure the reset works


LDR r1, =WAIT_TIME      @set up counter for wait loop
```

```
    LDR r0, =0x0            @when counter is zero, wait loop can end
ssc0_init_loop:
    ADD r0, r0, #0x1        @increment counter by 1
    CMP r0, r1              @if counter is zero, wait loop is over
    BNE ssc0_init_loop      @otherwise keep waiting

    LDR r0, =PIOC_SODR      @set reset pin back to end reset
    LDR r1, =PIN_25
    STR r1, [r0]

    POP {r0, r1}            @restore registers and return

    BX LR



@ call_start(rx_buffer_addr)
@
@ Description:      call_start initializes the shared variables used by audio
@                   functions and also starts ssc0 interrupts so that audio can
@                   be input and output.
@
@ Operation:        Sets shared variables to their in initial value and writes
@                   to the SSC0 interrupt enable register to trigger interrupts
@                   on frame sync signals.
```

```
@
@ Arguments:        r0 - address of first receive buffer.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@ Shared Variables: rxbuff1, txaudiocount, rxaudiocount, txneedbuf, rxneedbuf.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  rxbuff1.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
```

```
@               3/7/2012     Josh Fromm  Updated to function better

@               3/29/2012    Josh Fromm  Comments updated



.global call_start

call_start:     @sets up registers and variables needed for codec and SSC to run

   PUSH {r0, r1}


   LDR r1, =rxbuff1        @store passed receive buffer address

   STR r0, [r1]


   LDR r0, =txaudiocount   @set transmit buffer to be full (prevents sending

                           @until a new transmit buffer is passed)

   LDR r1, =AUDIO_BUFLEN

   STR r1, [r0]


   LDR r0, =rxaudiocount   @set receive buffer to empty

   LDR r1, =0x0

   STR r1, [r0]


   LDR r0, =txneedbuf      @indicate a new transmit buffer is needed

   LDR r1, =TRUE

   STR r1, [r0]
```

```
    LDR r0, =rxneedbuf      @indicate a new receive buffer is needed
    LDR r1, =TRUE
    STR r1, [r0]

    LDR r0, =SSC0_IER       @enable frame sync interrupts to occur.
    LDR r1, =SSC0_IER_VAL
    STR r1, [r0]

    POP {r0, r1}            @restore registers and return
    BX LR



@ call_halt()
@
@ Description:      call_halt disables frame sync interrupts to that no more
@                   audio data is input or output.
@
@ Operation:        Writes to the SSC0 interrupt disable register to turn off
@                   frame sync interrupts.
@
@ Arguments:        None.
@
@ Return Values:    None.
@
```

```
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@ Shared Variables: None.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated


.global call_halt
call_halt:                  @end the current call
```

```
        PUSH {r0, r1}           @save used registers

        LDR r0, =SSC0_IDR       @end ssc interrupts
        LDR r1, =SSC0_IDR_VAL
        STR r1, [r0]

        POP {r0, r1}            @restore registers and return
        BX LR

@ CodecHandler()
@
@ Description:      CodecHandler is the event handler called when a frame sync
@                   interrupt on SSC0 occurs. Codec handler's basic purpose is
@                   to read the half word in SSC0's receive holding register
@                   and move that half word into a buffer of other received values.
@                   CodecHandler then takes the next half word in an output buffer
@                   and writes it to SSC0's transmit holding register. SSC0 also
@                   checks whether either of these buffers is filled and handles
@                   such exceptions accordingly.
@
@ Operation:        CodecHandler processes an SSC0 receive first, then transmits
@                   data; however, input and output function essentially the
@                   same way. CodecHandler first checks whether the receive or
@                   transmit buffer is full or empty respectively. If so,
```

```
@               CodecHandler switches the active buffer to the back up buffer
@               and indicates a new buffer is needed. If there is no
@               active buffer, received data is dropped or a 0 is transmitted.
@               If data does not need to be dropped, CodecHandler inputs and
@               outputs data normally and increments counters indicating how
@               how many bytes in the current buffer have been input or output.
@
@
@ Arguments:       None.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - varies by process, often used as address storage
@               r1 - varies by process, often used as control word or data
@               r2 - values loaded from addresses
@ Shared Variables: rxaudiocount, txaudiocount, rxneedbuf, txneedbuf, txbuff1,
@               txbuff2, rxbuff1, rxbuff2.
@ Global Variables: None.
@ Input:           Audio data from codec moved to buffer.
@ Output:          Outputs audio data from buffer to codec.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
```

```
@
@ Algorithms:       None.
@
@ Data Structures:  buffers located at rxbuff1, rxbuff2, txbuff1, txbuff2.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated



.global CodecHandler
CodecHandler:               @input or output the next halfword of data
    SUB  LR, LR, #4      @first correct systems pipeline
    STMFD SP!, {LR}
    PUSH {r0, r1, r2}

RxFullCheck:                @if buffers are full, swap them or drop data
    LDR r0, =rxaudiocount
    LDR r1, [r0]
    CMP r1, #AUDIO_BUFLEN   @compare to the total size of a buffer
    BNE Handle_Rx           @otherwise, read and write next byte
```

```
@       BEQ RxBuff_Swap


RxBuff_Swap:                @active buffer is full and must be swapped with backup

        LDR r0, =rxneedbuf        @if this is the second buffer, data must be dropped

        LDR r1, [r0]

        CMP r1, #TRUE        @if neebuf is true then a backup buffer isnt provided

        BEQ RxDrop


        @otherwise switch buffers normally


        LDR r0, =rxaudiocount     @first reset count of bytes written to buffer

        LDR r1, =0x0

        STR r1, [r0]


        LDR r0, =rxbuff1        @set rxbuff1 to contain the address of the next

                        @buffer

        LDR r1, =rxbuff2

        LDR r2, [r1]

        STR r2, [r0]


        LDR r0, =rxneedbuf      @indicate that rx needs a new buffer

        LDR r1, =TRUE

        STR r1, [r0]

        B   Handle_Rx          @then proceed to read and output data
```

```
RxDrop:                  @ran out of buffer space, must drop data
    LDR  r0, =SSC0_RHR  @read from receive register and do nothing with data
    LDRH r1, [r0]

    B   TxFullCheck     @then go on to handle tx

Handle_Rx:               @add next received byte to buffer
   LDR r0, =rxbuff1     @first determine where next received byte will be stored
   LDR r2, [r0]         @value of base address of buffer loaded
   LDR r0, =rxaudiocount  @base address offset by number of bytes already stored
   LDR r1, [r0]
   ADD r2, r2, r1       @once base and offset are added, r2 contains where
                   @to store next read byte

   LDR r0, =SSC0_RHR    @now read next receive half word
   LDRH r1, [r0]
   STRH r1, [r2]        @and store that value in the buffer

   LDR r0, =rxaudiocount    @update count of bytes written to buffer
   LDR r1, [r0]
   ADD r1, r1, #0x2
   STR r1, [r0]
```

```
    @then continue to handle the transmit

TxFullCheck:        @first determine if current transmit buffer has been
                    @completely transmitted
    LDR r0, =txaudiocount
    LDR r1, [r0]
    CMP r1, #AUDIO_BUFLEN   @compare with total size of audio buffer
    BNE Handle_Tx       @if buffer isnt full, input normally
@   BEQ TxBuff_Swap     @if it is full, swap to backup buffer

TxBuff_Swap:        @switch from buffer 1 to buffer 2
    LDR r0, =txneedbuf
    LDR r1, [r0]
    CMP r1, #TRUE
    BEQ TxDrop      @if already in buffer 2, drop data

    @otherwise swap buffers normally

    LDR r0, =txaudiocount   @first reset number of bytes stored in buffer
    LDR r1, =0x0
    STR r1, [r0]

    LDR r0, =txbuff1        @set txbuff1 to contain the address of the next
                    @buffer
```

```
    LDR r1, =txbuff2
    LDR r2, [r1]
    STR r2, [r0]

    LDR r0, =txneedbuf      @indicate that tx needs a new buffer
    LDR r1, =TRUE
    STR r1, [r0]
    B   Handle_Tx           @then proceed to read and output data

TxDrop:                 @out of data to transmit, must send a blank
    LDR r0, =SSC0_THR
    LDR r1, =0x0
    STRH r1, [r0]       @output a blank half word

    B CodecHandlerDone  @once sent function is done

Handle_Tx:              @first output next halfword in buffer
    LDR r0, =txbuff1     @load base address of current buffer
    LDR r2, [r0]         @address of buffer loaded

    LDR r0, =txaudiocount   @determine which halfword in the buffer should be
                @output
    LDR r1, [r0]
```

```
    ADD r2, r2, r1          @base address of buffer shifted by the number of
                            @bytes already read from tx buffer
    LDRH r0, [r2]           @load the next halfword to output

    LSL r0, #VOLUME_CON_BITS          @set volume control
    ADD r0, r0, #VOLUME_CON_VAL

    LDR r1, =SSC0_THR
    STRH r0, [r1]           @output next half word in buffer

    LDR r0, =txaudiocount   @update number of bytes read from buffer
    LDR r1, [r0]
    ADD r1, r1, #0x2
    STR r1, [r0]

    @function is now done

CodecHandlerDone:

    LDR r0, =SSC0_SR    @read ssc0 status register to reset it
    LDR r1, [r0]

    LDR r0, =AIC_EOICR   @signal end of interrupt
    LDR r1, =TRUE
```

```
        STR r1, [r0]


        POP {r0, r1, r2}    @restore registers and return

        LDMFD SP!, {PC}^



@ update_tx(new_buffer_addr)
@
@ Description:      update_tx checks whether a new buffer of outgoing audio
@                   data is needed. A buffer is needed when one or both of
@                   the stored buffers has been emptied of audio data. If this is the
@                   case, the function accepts the passed buffer address and
@                   returns true. If no buffer is needed the
@                   function returns false.
@
@ Operation:        update_tx simply checks the statuses
@                   of shared variables to determine whether a new buffer is
@                   needed, then updates shared variables to indicate a new buffer
@                   has been accepted, or does nothing to indicate no new buffer
@                   has been accepted.
@
@ Arguments:        r0 - address of available transmit buffer.
@
@ Return Values:    r0 - TRUE if passed buffer was accepted, FALSE if not.
```

```
@
@ Local Variables:  r0 - contains addresses and variable values
@                   r1 - contains addresses and variable values
@                   r2 - contains addresses and variable values
@ Shared Variables: txneedbuf, txbuff2.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: r0.
@ Stack Depth;       2.
@
@ Algorithms:        None.
@
@ Data Structures:  buffer stored at address in txbuff2.
@
@ Known Bugs:        None.
@ Limitations:       None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                   3/7/2012     Josh Fromm  Updated to function better
@                   3/29/2012    Josh Fromm  Comments updated
```

```
.global update_tx
update_tx:
    PUSH {r1, r2}     @save used registers

    LDR r1, =txneedbuf  @determine if a new buffer is needed
    LDR r2, [r1]
    CMP r2, #TRUE
    BNE update_tx_nobuf @if no buffer is needed, return false
@     BEQ new_tx_buff     @if a buffer is needed, move to buffer handling
                  @process

new_tx_buff:            @accept the new buffer

    LDR r1, =txbuff2    @handle by adding passed buffer
                  @to buffer two spot
    STR r0, [r1]

    LDR r1, =txneedbuf  @indicate a buffer is no longer needed
    LDR r2, =FALSE
    STR r2, [r1]

    LDR r0, =TRUE       @indicate update was needed
```

```
    B update_tx_done    @function is done


update_tx_nobuf:           @a new buffer is not needed
    LDR r0, =FALSE      @set return value to indicate an update was un needed
@       B update_tx_done    @function is done

update_tx_done:            @pop registers and return

    POP {r1, r2}
    BX LR



@ update_rx(new_buffer_addr)
@
@ Description:      update_rx checks whether a new buffer to store incoming audio
@                   data in needed. A buffer is needed when one or both of
@                   the stored buffers has been filled with audio data. If this is the
@                   case, the function accepts the passed buffer address and
@                   returns true. If no buffer is needed the
@                   function returns false.
@
@ Operation:        Iupdate_tx checks the statuses of shared variables to
@                   determine whether a new buffer is needed, then updates
```

```
@               shared variables to indicate a new buffer has been accepted,
@               or does nothing to indicate no new buffer has been accepted.
@
@ Arguments:       r0 - address of available receive buffer.
@
@ Return Values:    r0 - TRUE if passed buffer was accepted, FALSE if not.
@
@ Local Variables:  r0 - contains addresses and variable values
@               r1 - contains addresses and variable values
@               r2 - contains addresses and variable values
@ Shared Variables: rxneedbuf, rxbuff2.
@ Global Variables: None.
@ Input:           None.
@ Output:          None.
@ Error Handling:   None.
@
@ Registers Changed: r0.
@ Stack Depth;      2.
@
@ Algorithms:      None.
@
@ Data Structures:  buffer stored at address in rxbuff2.
@
@ Known Bugs:      None.
```

```
@ Limitations:      None.
@
@ Revision Hisotry: 2/28/2012    Josh Fromm  Outline Created
@                 3/7/2012     Josh Fromm  Updated to function better
@                 3/29/2012    Josh Fromm  Comments updated

.global update_rx
update_rx:
    PUSH {r1, r2}          @save used registers
    LDR r1, =rxneedbuf      @check if a new buffer is needed
    LDR r2, [r1]
    CMP r2, #TRUE
    BNE update_rx_nobuf     @if not, end function
@      BEQ new_rx_buff        @otherwise get the new buffer

new_rx_buff:
    LDR r1, =rxbuff2       @store passed pointer in spot allocated to
                          @back up buffer
    STR r0, [r1]

    LDR r1, =rxneedbuf      @indicate a buffer is no longer needed
    LDR r2, =FALSE
    STR r2, [r1]
```

```
	LDR r0, =TRUE		@set output to show that the buffer was taken

	B update_rx_done	@function can end

update_rx_nobuf:		@no update is needed
	LDR r0, =FALSE		@set return value to show no update was needed
@	B update_rx_done

update_rx_done:			@restore registers and return
	POP {r1, r2}
	BX LR



@ The data segment

.data
txbuff1:	@contains address of active transmit buffer
	.word	'?'
txbuff2:	@contains address of backup transmit buffer
	.word	'?'
rxbuff1:	@contains address of active receive buffer
	.word	'?'
rxbuff2:	@contains address of backup receive buffer
```

```
    .word   '?'
txaudiocount:   @number of bytes transmitted from current transmit buffer
    .word   '?'
rxaudiocount:   @number of bytes moved into current receive buffer
    .word   '?'
txneedbuf:      @whether a new transmite buffer is needed or not
    .word   '?'
rxneedbuf:      @whether a new receive buffer is needed or not
    .word   '?'

.end
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@                                           @

@ crt0.s                                     @

@                                           @

@ Initialization file for EE52 ARM VoIP phone project.  It sets up the IRQ     @

@ vector table, initializes the stacks for both the IRQ and System modes, sets @

@ up the Master Clock, all of the chip selects for external memories, and      @

@ will eventually call all of the intialization functions for each hardware    @

@ block.                                     @

@                                           @

@ Revision History:                                @

@                                           @

@   2008/02/02  Joseph Schmitz  Modified code from Arthur Chang to make it     @

@                     available to the students.                @

@                                           @

@   2011/01/27  Joseph Schmitz  Split from crt0.s to boot.s                 @

@                                           @

@   2011/01/31  Joseph Schmitz  Removed unused comments.                   @

@                                           @

@   2012/03/29  Josh Fromm      Added code to initialize VOIP system.         @

@                                           @

@                                           @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
.include    "at91rm9200.inc"

.include    "armvoip.inc"


.text

.arm


.global low_level_init

low_level_init:

.global _start

_start:


@@@ Stack and IRQ Initialization
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@


   LDR          r0, =TOP_STACK


   MSR          cpsr_c, #ARM_MODE_IRQ | I_BIT | F_BIT

   MOV          sp, r0

   SUB          r0, r0, #IRQ_STACK_SIZE


   MSR          cpsr_c, #ARM_MODE_SVC | I_BIT | F_BIT

   MOV          sp, r0

   SUB          r0, r0, #SVC_STACK_SIZE
```

```
    MSR         cpsr_c, #ARM_MODE_USR | F_BIT

    MOV         sp, r0


@@@ Future Code for Peripheral Initialization
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


    @ Eventually all the rest of your code will end up here.


@@@ Clock Initialization
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@


ClockMainInitStart:

    LDR r0, =MCKSTARTCON

    LDR r1, =PMC_MCKR

    STR r0, [r1]   @ first set clock to master clock from slow clock

ClockMainInitStab:

    LDR r0, =PMC_SR

    LDR r1, [r0]

    LDR r2, =LOW_BIT_MASK

    AND r0, r1, r2              @determine whether clock has

                           @signal has stabilized

    CMP r0, r2                  @check if stable

    BNE ClockMainInitStab               @if not stable, then keep
```

```
                    @looping
  @if low bit is set, then clock has stabilized and boot can continue




ClockInitStart:
@ Set PLLA to be masterclock with frequency 75 MHz

Proc_Clock:
  LDR r0, =PMC_SCER      @enable processor clock
  LDR r1, =PRCCLKCON
  STR r1, [r0]

Main_Osc:
  LDR r0, =PMC_MOR       @install the main oscillator
  LDR r1, =MORCON
  STR r1, [r0]

  LDR r1, =WAIT_TIME     @set wait loop counter
WaitLoop1:               @wait a while to let clock stabilize
  SUB r1, r1, #0x1       @subtract 1 from the wait loop counter
  CMP r1, #0x0           @once counter is zero, code can continue
  BNE WaitLoop1          @if the counter is not zero, keep waiting
```

```
PLLAR:
   LDR r0, =PMC_PLLAR      @initialize PLLA
   LDR r1, =PLLARCON
   STR r1, [r0]

   LDR r1, =WAIT_TIME
WaitLoop2:                 @wait a while to let clock stabilize
   SUB r1, r1, #0x1        @subtract 1 from the wait loop counter
   CMP r1, #0x0            @once counter is zero, code can continue
   BNE WaitLoop1           @if the counter is not zero, keep waiting

MCKR:                      @set the master clock to be PLLA divided by 2
   LDR r0, =PMC_MCKR
   LDR r1, =MCKCON
   STR r1, [r0]

ClockInitStab:             @check to make sure master clock has stabilized
   LDR r0, =PMC_SR
   LDR r1, [r0]
   LDR r2, =LOW_BIT_MASK
   AND r0, r1, r2                 @determine whether clock has
                           @signal has stabilized
   CMP r0, r2                     @check if stable
```

```
   BNE ClockInitStab           @if not stable, then keep
                               @looping
   @if low bit is set, then clock has stabilized and boot can continue

DRAM_Clock_Setup:          @set up PA4 to be the frequency used by the
                           @DRAM controller
   LDR r0, =PA4_VAL        @first relinquish control of PA4 to peripheral B
   LDR r1, =PIOA_BSR
   STR r0, [r1]

        LDR r0, =PA4_VAL        @disable PA4 so it can output a clock
        LDR r1, =PIOA_PDR
        STR r0, [r1]

        LDR r0, =PMC_SCER_PA4      @enable TCK1
        LDR r1, =PMC_SCER
        STR r0, [r1]

        LDR r0, =PMC_PCER_PIOA     @provide peripheral clock to PIOA
        LDR r1, =PMC_PCER
        STR r0, [r1]

   LDR r0, =PCK1_VAL       @load control word of PCK1, low bit sets
                           @output clock to equal master clock
```

```
   LDR r1, =PMC_PCK1         @load address of PCK1 control register

   STR r0, [r1]              @store the control word to set up PCK1


@@@ CS Initialization
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


NCSInit:

   LDR r0, =NCS0CON

   LDR r1, =SMC_CSR0

   STR r0, [r1]             @ set up chip select zero with control word

   LDR r0, =NCS1CON

   LDR r1, =SMC_CSR1

   STR r0, [r1]             @ set up chip select one with control word

   LDR r0, =NCS2CON

   LDR r1, =SMC_CSR2

   STR r0, [r1]    @ set up chip select two with control word

        LDR r0, =NCS3CON

        LDR r1, =SMC_CSR3

        STR r0, [r1]                              @set up chip select three with control word



@@@ Loop to catch execution if main returns (or you have no main)
@@@@@@@@@@@@@@
```

```
   @ You will eventually call Glen's main function here.
   BL     KeypadInit
   BL     displayInit
   BL     Timer0Init
   BL     ssc0_init
   B      main

LowLevelInitEndLoop:
   B LowLevelInitEndLoop

.end
```

```
@ display.inc
@ This file contains the constants used by the functions that run the VOIP
@ system's display output.
@ Revision History:
@
@   2012/2/5    Josh Fromm  Initial Revision

.equ    MEMORY_SIZE,    0x80        @im not sure what this does but im scared
                                    @to take it out
.equ    ASCII_ZERO,     0x30        @ASCII value for 0
.equ    ASCII_ELLIPSE,    0x2E        @ASCII value for .
.equ    ASCII_NULL,     0x0         @ASCII value for null
.equ    ASCII_A,        0x41        @ASCII value for A
.equ    DISPLAY_WRITE,  0x30000001  @address used when writing to display

.equ DISP_CS, 0x30000000  @ Register written to during
                          @ display initialization

.equ    NibShiftsPerWord, 0x8       @nibbles per word
.equ    MAX_10, 0x64                @maximum value that can be used in one chunk
                                    @of an IP address
.equ    Num_Shifts, 0x18    @initial number of shifts needed to extract
                            @a chunk of an IP address
.equ    MAX_16, 0x10000     @maximum hex value that can be used in display
```

```
                  @memory address
.equu    INIT_WAIT, 0x8000   @number of cycles to wait during display initialization
.equ    CLEAR_DISPLAY, 0x1  @value to write to clear the display
.equ    BOTTOM_ROW, 0xC0    @value to write to set display to bottom row
.equ    DISPLAY_RDY_BIT, 0x80    @bit indicating whether display is ready or not
.equ    DISPLAY_RDY, 0x0    @value of display ready bit indicating display is ready
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                        @
@ display.s                              @
@                                        @
@ Initialization, and character output for display on VOIP system.    @
@                                        @
@ Table of Contents:                     @
@   1. displayInit: Function used to initialize systems display.     @
@   2. display_IP: Function used to display the decimal equivalent of the   @
@      passed value.                     @
@   3. display_memory_addr: Function used to display passed value in hex.   @
@   4. display_status: Function used to display one of the system's statuses.  @
@   5. DispBusyCheck: Holds until display is ready to accept another command.  @
@                                        @
@ Revision History:                      @
@                                        @
@   2012/2/5    Josh Fromm  Initial Revision             @
@   2012/2/7    Josh Fromm  Code updated to work better          @
@   2012/3/29   Josh Fromm  Comments updated             @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@



.include    "at91rm9200.inc"
```

```
.include    "armvoip.inc"

.include    "display.inc"


.text

.arm

.align 2


@@@ display initialization

@initializes the system's display according to the steps listed in the optrex manual




@ displayInit()

@

@ Description:      displayInit runs the necesarry procedure for the system's

@                   display module to function properly.

@

@ Operation:        displayInit writes various values to the display module and

@                   employs wait loops in between to meet the requirements for

@                   initializing the display. Note that the values written to

@                   the display in this function are defined in the Optrex

@                   manual and should not be changed.

@

@ Arguments:        None.

@
```

```
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@                   r2 - contains addresses and control words.
@ Shared Variables: Last_Status.
@ Global Variables: None.
@ Input:            None.
@ Output:           clears display module and displays a blinking cursor.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      4.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm   Initial Revision
@                   2/7/2012    Josh Fromm  Updated to function better
@                   3/29/2012   Josh Fromm  Comments updated
```

```
.global displayInit
displayInit:

    PUSH    {r0, r1, r2, LR}   @save all used registers

    LDR     r0, =DISP_CS       @begin initialization of display
    LDRB    r1, =0x30          @write first value to display
    STRB    r1, [r0]

    LDR     r2, = 0x0   @set counter to 0, this is needed to allow enough time
                    @between display initialization instructions
DispWaitLoop1:
    ADD     r2, #1      @increment counter
    CMP     r2, #INIT_WAIT
    BNE     DispWaitLoop1   @loop until sufficient cycles have passed

    STRB    r1, [r0]       @write same value to display again
    LDR     r2, =0x0    @set counter to 0, this is needed to allow enough time
                    @between display initialization instructions
DispWaitLoop2:         @wait for a while again
    ADD     r2, #1
    CMP     r2, #INIT_WAIT
```

```
BNE     DispWaitLoop2   @loop until sufficient cycles have passed

STRB     r1, [r0]       @write same value to display for a third time

@other instructions do not require a wait time and can be run in succession.

LDRB     r1, =0x08
STRB     r1, [r0]
BL      DispBusyCheck   @make sure display is ready for another value
LDRB     r1, =0x01
STRB     r1, [r0]
BL      DispBusyCheck   @make sure display is ready for another value
LDRB     r1, =0x06
STRB     r1, [r0]
BL      DispBusyCheck   @make sure display is ready for another value
LDRB     r1, =0x0F
STRB     r1, [r0]
BL      DispBusyCheck   @make sure display is ready for another value
LDRB    r1, =0x3F       @activate display with blinking and cursor location
STRB    r1, [r0]

@   Display is now initialized running

LDR r0, =Last_Status    @initialize last status register
```

```
   LDR r1, =0xFF          @set last status to a value that can not be reached
   STR r1, [r0]

   POP    {r0, r1, r2, LR}    @restore used registers and return

   BX LR                @after initialization jump to main loop

@display_IP
@Description:    This function takes a 32 bit input in r0 and displays the
@                equivalent IP address on the VOIP systesms display. The
@                format of the output display is 4 sets of three digit values
@                seperated by ellipses (as in a standard IP address).
@Operation:      The function contains a loop that shifts the input value
@                so that the byte currently being output to display is the
@                lowest byte in the register being operated on. Then, all
@                other bits in the register are masked. Next, the function
@                divides the register by the highest possible value of 10
@                (1000 initially). The remainder of this division is converted
@                to ASCII and output to the display. Then, the register
@                containing the power of 10 is divided by 10 and the division
@                repeats itself until the last digit is output. Once this is
@                done, a count decrements to cause the shift to move a different
@                byte to the low byte of the operation register. The procedure
@                then loops and should repeat itself exactly 4 times. After
```

@ each byte, an ellipse is output to the display.

@Arguments: r0 - Value to output as an IP address to the display.

@Return Values: None.

@Local Variables: r0 - division remainder

@ r1 - division quotient

@ r2 - shift counter

@ r3 - IP address initial value

@ r4 - Power of 10

@ r5 - individual character byte value

@Shared Variables: None.

@Global Variables: None.

@Input: None.

@Output: Displays an IP address on the bottom row of the display module.

@Error Handling: None.

@Registers Used: None.

@Stack Depth: 7 words.

@Algorithms: Repeatedly divide by powers of 10 to get digits and

@ remainders needed for next division.

@

@Data Structures: Arrays (for output string).

@

@Known Bugs: None.

@Limitations: None.

@ Revision Hisotry: 2/5/2012 Josh Fromm Initial Revision

```
@               2/7/2012    Josh Fromm  Updated to function better

@               3/29/2012   Josh Fromm  Comments updated


.global display_IP

display_IP:

display_IP_Init:

   PUSH {r0, r1, r2, r3, r4, r5, LR}


   MOV r3, r0         @save the passed IP address

   LDR r2, =Num_Shifts @load the inital number of shifts needed


   LDR r0, =DISP_CS    @first change display address to be bottom row

   LDR r1, =BOTTOM_ROW

   STRB r1, [r0]


display_IP_Shift:

   MOV r5, r3      @reset r5 to be the initial IP value

   LSR r5, r2      @shift the IP value by the number of bits remaining in the

                   @the shift register

   AND r5, r5, #LOW_BYTE_MASK  @mask all bits except low byte


   LDR r4, =MAX_10 @prepare for display loop by setting power of 10

display_IP_Main:

   MOV r1, r4      @first set denominator to be the current power of 10
```

```
    MOV r0, r5      @set numerator to be binary value of a byte
    BL Divide       @divide numerator by denominator

    MOV r5, r0      @store remainder as new byte value

    BL DispBusyCheck
    LDR r0, =DISPLAY_WRITE  @output quotient of division
    ADD r1, r1, #ASCII_ZERO @convert value to ascii
    STRB r1, [r0]           @output value

    MOV r0, r4      @set numerator to be current power of 10
    LDR r1, =0xA    @set denominator to be 10
    BL Divide       @divide to find new power of 10

    CMP r1, #0x0    @if division returned zero, then we are done with the function
    BEQ display_Main_Done

    MOV r4, r1      @otherwise store that value and repeat
    B display_IP_Main

display_Main_Done:
    CMP r2, #0x0       @determine if this was the last byte, if so no ellipse is
                   @needed
    BEQ display_IP_Done @jump to finished
```

```
    SUB r2, r2, #BITSPBYTE  @then subtract the number of bits in a byte from
                            @number of shifts remaining
    BL DispBusyCheck        @check to make sure display is ready

    LDR r0, =DISPLAY_WRITE  @otherwise write an ellipse to the display
    LDRB r1, =ASCII_ELLIPSE
    STRB r1, [r0]

    B display_IP_Shift      @then begin to output next byte packet of IP

display_IP_Done:        @all characters have been displayed, pop registers and
                        @return.
    LDR r0, =Last_Status    @reset last status
    LDR r1, =0xFF
    STR r1, [r0]
    POP {r0, r1, r2, r3, r4, r5, LR}
    BX LR



@display_memory_addr
@Description:     This function takes a 32 bit input in r0 and displays the
@                 equivalent memory address in hex on the VOIP systems display.
@Operation:       The function divides the register by the highest possible value of 16
@                 (256 initially). The remainder of this division is converted
```

```
@               to ASCII and output to the display. Then, the register
@               containing the power of 16 is divided by 16 and the division
@               repeats itself until the last digit is output.
@Arguments:        r0 - Value to output as an IP address to the display.
@Local Variables:   r0 - division remainder
@               r1 - division quotient
@               r2 - power of 16
@               r3 - remainder of address
@Shared Variables:  None.
@Global Variables:  None.
@Input:           None.
@Output:           Displays a memory address on the bottom row of the display module.
@Error Handling:    None.
@Registers Used:    None.
@Stack Depth:      5 words.
@Algorithms:             Repeatedly divide by powers of 16 to get digits and
@                                remainders needed for next division.
@
@Data Structures:      Arrays (for output string).
@
@Known Bugs:                       None.
@Limitations:             None.
@ Revision Hisotry: 2/5/2012    Josh Fromm   Initial Revision
@               2/7/2012     Josh Fromm  Updated to function better
```

```
@               3/29/2012    Josh Fromm  Comments updated

.global display_memory_addr
display_memory_addr:
display_addr_Init:
   PUSH {r0, r1, r2, r3, LR}
   MOV r3, r0         @save passed address
   LDR r0, =DISP_CS    @first change display address to be bottom row
   LDR r1, =BOTTOM_ROW
   STRB r1, [r0]

   LDR r2, =MAX_16 @prepare for display loop by setting power of 16
display_addr_Main:
   MOV r1, r2      @first set denominator to be the current power of 16
   MOV r0, r3      @set numerator to be binary value of address
   BL Divide       @divide numerator by denominator

   MOV r3, r0      @store remainder as remaining address value
   CMP r1, #0xA   @if quotient is greater than 10, conversion to ASCII is
              @different
   BGE display_addr_hex    @if remainder is greater than 10, go to special
                    @protocol

                    @otherwise proceed normally
```

```
    LDR r0, =DISPLAY_WRITE  @output quotient of division
    ADD r1, r1, #ASCII_ZERO @convert value to ascii
    BL DispBusyCheck        @block until display is ready
    STRB r1, [r0]           @output value
    B display_addr_power_update

display_addr_hex:
    LDR r0, =DISPLAY_WRITE      @load register that must be written to for output
    SUB r1, r1, #0xA            @subtract by 10 to find ascii offset from 'A'
    ADD r1, r1, #ASCII_A        @find ascii value of address character
    STRB r1, [r0]               @output address
    B display_addr_power_update @now go on to update power of 16

display_addr_power_update:
    MOV r0, r2      @set numerator to be current power of 16
    LDR r1, =0x10   @set denominator to be 16
    BL Divide       @divide to find new power of 16

    CMP r1, #0x0    @if division returned zero, then we are done with the function
    BEQ display_addr_Done

    MOV r2, r1      @otherwise store that value and repeat
    B display_addr_Main
```

```
display_addr_Done:      @all characters have been displayed, pop registers and
                        @return.
    LDR r0, =Last_Status    @reset last status
    LDR r1, =0xFF
    STR r1, [r0]
    POP {r0, r1, r2, r3, LR}
    BX LR



@ display_status(status_value)
@
@ Description:      display_status takes a value between 0 and 10 and outputs
@                   a corresponding status. The list of statuses is fixed and
@                   can be seen in DispStringArray.
@
@ Operation:        To help stabilize the display, display status only outputs
@                   to the display if the passed status isn't currently being
@                   displayed, this is done by checking the value stored in
@                   Last_Status. If that value is the same as the passed value,
@                   display_status returns. Otherwise, display status takes
@                   the base address of DispStringArray and adds on the passed
@                   value times 16 to find the start address of the correct status.
@                   display_status then iterates through the ascii characters
```

```
@               and displays them until a null is encountered.
@
@ Arguments:       Value of status to display.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and control words.
@               r1 - contains addresses and control words.
@               r2 - contains addresses and control words.
@               r3 - contains addresses and control words.
@ Shared Variables: Last_Status.
@ Global Variables: None.
@ Input:          None.
@ Output:          Displays a status on the top row of the display module.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      5.
@
@ Algorithms:      None.
@
@ Data Structures:  DispStringArray.
@
@ Known Bugs:      None.
```

```
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm   Initial Revision
@                   2/7/2012    Josh Fromm  Updated to function better
@                   3/29/2012   Josh Fromm  Comments updated

.global display_status
display_status:
    PUSH {r0, r1, r2, r3, LR}         @save used registers



    MOV r2, r0                  @save input

    LDR r0, =Last_Status       @then determine if status needs to be updated
    LDR r1, [r0]
    CMP r2, r1
    BEQ display_status_done     @if not, function is over

    STR r2, [r0]                @then update last status

    BL DispBusyCheck            @check if display is ready

    LDR r0, =DISP_CS            @if so clear the display
    LDRB r1, =CLEAR_DISPLAY
```

```
    STRB r1, [r0]

    ADR r1, DispStringArray     @next, determine which status is to be displayed
    LSL r2, #0x4                @convert offset from byte offset to table offset
    ADD r1, r1, r2              @find address string begins at
    LDR r2, =DISPLAY_WRITE      @address of register to be written to output

display_status_loop:
    LDRH r0, [r1]               @load a half word from the string to display
    MOV r3, r0
    AND r3, r3, #LOW_BYTE_MASK
    CMP r3, #ASCII_NULL         @if low byte is ascii null, string is over
    BEQ display_status_done     @jump to end of function if string done
    BL DispBusyCheck            @block until display is ready
    STRB r3, [r2]               @otherwise output to display

    LSR r0, #0x8                @check if byte following the one just output is
                                @ a null.
    CMP r0, #ASCII_NULL
    BEQ display_status_done     @if so, function is done
    BL DispBusyCheck            @if not, wait to make sure display is ready for
                                @for another character
    STRB r0, [r2]               @then display that character
```

```
    ADD r1, r1, #HALFWORD_SIZE      @increment byte to be displayed by 1
    B display_status_loop      @and repeat

display_status_done:          @restore registers and return
    POP {r0, r1, r2, r3, LR}
    BX LR



@ DispBusyCheck()
@
@ Description:     DispBusyCheck reads from the display modules status register
@                  to determine if enough time has passed for the display to
@                  accept another command. The function will hold until the
@                  display is ready.
@
@ Operation:       DispBusyCheck reads from the display modules status register
@                  and checks the status of the bit indicating whether the display
@                  is ready for another command. If the display is ready, the
@                  function returns, if the display is not ready, the function
@                  loops back and repeats the checking process.
@
@ Arguments:       None.
@
@ Return Values:   None.
```

```
@
@ Local Variables:  r0 - contains addresses and control words.
@                   r1 - contains addresses and control words.
@ Shared Variables: None.
@ Global Variables: None.
@ Input:            Status vale from display module.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  DispStringArray.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm   Initial Revision
@                   2/7/2012    Josh Fromm  Updated to function better
@                   3/29/2012   Josh Fromm  Comments updated


@DispBusyCheck:
```

```
DispBusyCheck:
   PUSH {r0, r1}   @save used registers


DispBusyLoop:
   LDR r1, =DISP_CS    @load the value stored in the status register of display
   LDRB r0, [r1]
   AND r0, r0, #DISPLAY_RDY_BIT   @extract the status bit
   CMP r0, #DISPLAY_RDY          @check if display is ready
   BNE DispBusyLoop              @if not, repeat loop


   POP {r0, r1}                  @otherwise restore registers and return
   BX LR


DispStringArray:      @array of all possible statuses that can be displayed
                 @on VOIP system. Statuses are in order of their corresponding
                 @value, i.e. Idle has a value of 0, Off hook has a value
                 @ of 1, etc.
   .asciz "Idle          "
   .asciz "Off Hook      "
   .asciz "Ringing       "
   .asciz "Connecting    "
   .asciz "Connected     "
   .asciz "Set IP        "
```

```
    .asciz "Set Subnet     "
    .asciz "Set Gateway    "
    .asciz "Memory Save    "
    .asciz "Memory Recall  "
    .asciz "Recalled       "

@ The Data Segment
.data

Last_Status:    @value of the status that was last displayed
    .word '?'

.end
```

```
@ether.inc

@Constants used by functions that run the system's ethernet.

@ Revision History:

@

@   2012/3/3    Josh Fromm  Initial Revision


    .equ    PBUF_POOL,  0x3        @value used to indicate a PBUF_POOl setting

    .equ    PBUF_RAW,   0x3        @value used to indicate a PBUF_RAW setting


    .equ    RBQP_BUFFER_SIZE, 0x5EE @maximum size of an RBQP buffer

    .equ    RBQP_ALLOC, 0x600      @space given for an RBQP buffer

    .equ    NUM_RBQP_BUFFERS, 0x5   @number of RBQP buffers used


    .equ    TRANSFER_SIZE, 0x5EE    @maximum number of bytes that can be transferred


        .equ     MAC_ADDR_L, 0x87654321  @lower word of MAC address

        .equ     MAC_ADDR_H, 0xEA09      @higher word of MAC address


    .equ    WRAP_BIT, 0x2          @bit corresponding to wrap bit in RBQP

    .equ    PMC_PCER_ETHER, 0x100000C   @PCER value to write for ethernet

    .equ    ETHER_PINS, 0xFFFFFFE0      @PIOA/B pins not used by other peripherals

    .equ    PERPHA_PINS, 0x0001FF80     @Ethernet pins in PIOA

    .equ    PERPHB_PINS, 0x080FF000     @Ethernet pins in PIOB

    .equ    EMAC_CTL_VAL, 0xC       @enable transmitting and receiving over ethernet
```

```
.equ    EMAC_CFG_VAL, 0x810     @set speed to 10BASET, half duplex

.equ    IDLE_BIT,     0x8       @bit corresponding to transmit idle status

.equ    OWNERSHIP_MASK, 0xFFFFFFFE @all bits but the ownership bit
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                          @
@ ether.s                                                  @
@                                                          @
@ File Description:                                        @
@ Initialization, event handling, and basic functions of VOIP emac interface.  @
@                                                          @
@ Table of Contents:                                       @
@ 1. ether_init: initializes buffers, shared variables, registers, and        @
@ interrupt handlers needed for interacting with EMAC.                        @
@ 2. etherDMA_handler: Sets needed shared variables to indicate a finished     @
@ receive or transfer.                                     @
@ 3. ether_transmit: transmits a pbuf chain over ethernet                     @
@ 4. ether_rx_available: indicates whether received ethernet data is available @
@ 5. ether_ISR: determines which receive buffer should be serviced next.      @
@ 6. ether_receive: indicates start address of a received data packet.        @
@                                                          @
@ Revision History:                                        @
@                                                          @
@   2012/3/3    Josh Fromm  Initial Revision                            @
@   2012/3/7    Josh Fromm  Code updated so that functions work             @
@   2012/3/29   Josh Fromm  Commenting updated                          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
.include    "at91rm9200.inc"

.include    "armvoip.inc"

.include    "ether.inc"


.text

.arm

.align 2


@ ether_init

@

@ Description:     ether_init initializes the shared variables needed by other

@                  ethernet functions, initializes buffers used

@                  for DMA transfers, sets up DMA interrupt handlers,

@                  initializes the ethernet protocol, and creates the receive

@                  buffer status register RBQP. Note that RBQP contains the

@                  receive buffers.

@

@ Operation:       Moves many control words into appropriate registers and

@                  creates the RBQP buffer by iterating through its chunks and

@                  storing the address of where in RBQP_BUFF data should be

@                  stored, incrementing by a word, then storing the maximum

@                  size of data that can be stored in that buffer. When the last
```

```
@                chunk is reached, the wrap bit is set.
@
@ Arguments:       None.
@
@ Return Values:    r0 - TRUE if intialized with no errors, FALSE otherwise.
@
@ Local Variables:  r0 - contains addresses and control words.
@                r1 - contains addresses and control words.
@                r2 - counter needed for buffer set up.
@ Shared Variables: RBQP, RBQP_BUFF.
@ Global Variables: None.
@ Input:           None.
@ Output:          None.
@ Error Handling:   Checks for ethernet errors and returns FALSE if one occurs.
@
@ Registers Changed: r0.
@ Stack Depth;       2.
@
@ Algorithms:      None.
@
@ Data Structures:  RBQP, RBQP_BUFF.
@
@ Known Bugs:      None.
@ Limitations:     None.
```

```
@
@ Revision Hisotry: 3/3/2012    Josh Fromm  Outline Created
@                  3/29/2012   Josh Fromm  Comments Updated


.global ether_init
ether_init:
   PUSH {r0, r1, r2, r3, r4, r5}   @save used registers

   LDR r2, =0x0    @set counter to zero
build_RBQP:
   LDR r0, =RBQP   @load address of first chunk of the RBQP
   MOV r4, r2      @get the current counter value into r4
   LSL r4, #0x3    @each chunk consists of 2 words so need to shift over by 8 bytes
               @for each finished chunk
   ADD r0, r0, r4  @add this to base of RBQP to arrive at beginning of current chunk

   LDR r4, =RBQP_ALLOC   @determine start address of current buffer
   MUL r3, r2, r4        @multiply the current counter value by the size of a
                    @buffer
   LDR r5, =RBQP_BUFF    @load the start address of the RBQP buffer space
   ADD r3, r3, r5        @add the offset calculated to the start address

   STR r3, [r0]         @put address into buffer
```

```
    ADD r0, #0x4        @move one word over
    LDR r4, =RBQP_BUFFER_SIZE
    STR r4, [r0]        @store length

    ADD r2, r2, #0x1     @increment counter by 1
    CMP r2, #NUM_RBQP_BUFFERS      @if all buffers have been built, we can continue
    BNE build_RBQP      @if not, keep building buffers

    SUB r0, r0, #0x4    @if code reaches this point, the last chunk is being built
                @so we must move back one word and set bit 1 to 1 to
                @indicate wrap.
    ADD r3, r3, #WRAP_BIT    @set the wrap bit to 1
    STR r3, [r0]

init_ether_regs:      @set up ethernet control registers and convert pio a and
                @b pins to peripheral PID pins

    LDR r0, =PMC_PCER      @Supply clocks to needed peripherals
    LDR r1, =PMC_PCER_ETHER
    STR r1, [r0]

    LDR r0, =PIOA_PDR      @disable most PIOA pins
    LDR r1, =ETHER_PINS
```

```
STR r1, [r0]

LDR r0, =PIOB_PDR      @disable most PIOB pins
STR r1, [r0]

LDR r0, =PIOA_ASR      @convert needed PIOA pins to peripheral A
LDR r1, =PERPHA_PINS
STR r1, [r0]

LDR r0, =PIOB_BSR      @convert needed PIOB pins to peripheral B
LDR r1, =PERPHB_PINS
STR r1, [r0]

        LDR r0, =EMAC_HSH      @set the high word of the MAC address
        LDR r1, =MAC_ADDR_H
        STR r1, [r0]

        LDR r0, =EMAC_HSL      @set the low word of the MAC address
        LDR r1, =MAC_ADDR_L
        STR r1, [r0]

LDR r0, =EMAC_CTL      @set the control register for EMAC
LDR r1, =EMAC_CTL_VAL
STR r1, [r0]
```

```
    LDR r0, =EMAC_CFG       @set the configuration register for EMAC
    LDR r1, =EMAC_CFG_VAL
    STR r1, [r0]

    LDR r0, =EMAC_RBQP      @store the address of the RBQP buffer to allow for
                            @EMAC DMA
    LDR r1, =RBQP
    STR r1, [r0]

    POP {r0, r1, r2, r3, r4, r5}    @restore registers and return
    BX LR



@ ether_transmit
@
@ Description:    ether_transmit outputs the the pbuf chain located at the
@                 passed address over ethernet through a DMA write sequence.
@
@ Operation:      ether_transmit functions by first obtaining the information
@                 about the current pbuf packet (size, next, and payload) and then
@                 transfers all the data in that pbuf to the transmit buffer.
@                 the function then moves to the next pbuf and repeats until
```

```
@               a pointer to NULL is found. Once finished, the function
@               initiates a DMA to transfer the data in the buffer to the
@               ethernet output register. If the amount of data in the passed
@               pbuf structure exceeds the size of the transmit buffer,
@               the function returns  FALSE to indicate and error. Otherwise,
@               the function returns TRUE to indicate success.
@
@ Arguments:      r0 - Address of first pbuf structure to output.
@
@ Return Values:   r0 - TRUE if output with no errors, FALSE otherwise.
@
@ Local Variables:  r0 - contains addresses and control words.
@               r1 - contains addresses and control words.
@               r2 - various uses.
@               r3 - needed to store pbuf information.
@               r4 - needed to store pbuf information.
@               r5 - needed to store pbuf information.
@               r6 - used for function calls
@               r7 - used for function calls
@               r8 - bit mask
@ Shared Variables: ether_tx_buff.
@ Global Variables: None.
@ Input:          None.
@ Output:         EMAC.
```

```
@ Error Handling:   Checks for ethernet errors or overflow and returns FALSE
@                   if one occurs.
@
@ Registers Changed: r0.
@ Stack Depth;      9.
@
@ Algorithms:       None.
@
@ Data Structures:  ether_tx_buff.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 3/3/2012    Josh Fromm  Outline Created
@                   3/29/2012   Josh Fromm  Comments updated


.global ether_transmit
ether_transmit:
    PUSH {r1, r2, r3, r4, r5, r6, r7, r8, LR}
    LDR r5, =Transfer_Buff


    MOV r1, r0          @first determine if the passed pbuf struct contains
                        @too much data
```

```
   ADD r0, r0, #0x8
   LDR r3, [r0]
   LDR r8, =0xFFFF
   AND r3, r3, r8      @get value of sum of all payloads, this value will be
                  @saved for later use
   LDR r2, =TRANSFER_SIZE
   CMP r3, r2
   BGT ether_transmit_error    @if theres too much data, indicate an error
   MOV r0, r1
@   BLE ether_transmit_main     @otherwise continue normally


ether_transmit_main:
   LDR r2, [r0]       @set r2 to be the pointer to the next pbuf packet
   ADD r0, r0, #WORD_SIZE    @increment to next word in pbuf structure
   LDR r6, [r0]       @set r6 to by payload address
   ADD r0, r0, #WORD_SIZE
   LDR r7, [r0]       @set r7 to be size of current payload
   LSR r7, #BITSPHW    @dont care about cumulative size of payloads, only
                  @interested in size of current payload


   BL copy        @copy specified number of bytes to the transmit buffer


   ADD r5, r5, r7  @increment receive buffer to point to the start of where
               @the next payload will be dumped
```

```
   CMP r2, #NULL   @if next pointer is null, all data has been transferred

   BEQ transmit_blocking @if this is the case then we move to sending out

               @the data


   MOV r0, r2      @otherwise set the current pbuf pointer to next

   B ether_transmit_main   @then loop back and repeat


transmit_blocking:


   LDR r0, =EMAC_TSR   @check status of transmit

   LDR r1, [r0]       @load transmit status value


   AND r1, r1, #IDLE_BIT    @isolate idle bit

   CMP r1, #IDLE_BIT       @check if that bit is set

   BNE transmit_blocking @if not, keep looping until it is


   LDR r0, =TRUE      @once transmit is done, indicate success


   BEQ transmit_ether_data @function can now output


transmit_ether_data:

   LDR r0, =EMAC_TAR

   LDR r1, =Transfer_Buff

   STR r1, [r0]
```

```
   LDR r0, =EMAC_TCR    @set length to be transmitted to total length of all
                   @pbuf payloads
   STR r3, [r0]
   B ether_transmit_done @now function is finished



ether_transmit_error:
   LDR r0, =FALSE      @indicate an error ocurred
@   B ether_transmit_done   @function can return


ether_transmit_done:    @restore used registers and return


   POP {r1, r2, r3, r4, r5, r6, r7, r8, LR}   @restore used registers and return
   BX LR


@ ether_rx_available:
@
@ Description:      ether_rx_available returns the current status of
@                ether_rx_flag. If a buffer is available, the function calls
@                ether_ISR to determine which buffer should be serviced next
@                by ether_receive.
@
@ Arguments:       None.
```

```
@
@ Return Values:    r0 - TRUE if theres a filled receive buffer, FALSE otherwise.
@                   r3 - address of filled RBQP chunk
@ Local Variables:  r0 - variable storage
@                   r1 - various things
@                   r2 - addresses
@                   r3 - counter
@                   r4 - value storage
@ Shared Variables: ether_rx_flag.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: r0, r3.
@ Stack Depth;      0.
@
@ Algorithms:       None.
@
@ Data Structures:  RBQP.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
```

```
@ Revision Hisotry: 3/3/2012    Josh Fromm  Outline Created
@                   3/29/2012   Josh Fromm  Comments Updated


.global ether_rx_available


ether_rx_available:
    PUSH {r1, r2, r4}   @save used registers


    LDR r3, =0x0        @set counter value to 0
RBQP_scan:
    LDR r0, =RBQP       @load address of RBQP
    MOV r1, r3          @load current counter value
    LSL r1, #0x3        @each chunk takes up 2 words = 8 bytes, so we must
                        @multiply by 8
    ADD r0, r0, r1      @add calculated to offset of RBQP
    LDR r2, [r0]        @load value of RBQP at that address
    MOV r4, r2          @save that value


    AND r4, r4, #0x1    @check if software owns this buffer
    CMP r4, #0x1
    BEQ buffer_found    @if so, we found a filled buffer


    AND r2, r2, #0x2    @check if this is the last buffer
```

```
   CMP r2, #0x2
   BEQ no_buffer       @if this was the last buffer, there is no available data

   ADD r3, r3, #0x1    @increment counter

   B RBQP_scan         @repeat loop

buffer_found:

   MOV r3, r0          @set r3 to be the address of the filled buffer to return

   LDR r0, =TRUE       @indicate a buffer was found
   B ether_rx_available_done

no_buffer:
   LDR r0, =FALSE      @indicate no buffer was found
@  B ether_rx_available_done

ether_rx_available_done:
   POP {r1, r2, r4}    @restore used registers
   BX LR               @return r0 = buffer received, r3 = payload address, and
                       @ r4 = size of payload
```

```
@ ether_receive
@
@ Description:      This function moves the data stored in the filled buffer
@               indicated by filled_buffer_num into a pbuf. The function
@               returns the start address of that pbuf chain. If no buffer
@               is filled, the function returns NULL.
@
@ Operation:        ether_receive operates by first checking if any buffers are
@               filled, if not the function returns NULL. The function
@               then calls pbuf_alloc to obtain a pbuf structure which can
@               be filled. The fucntion then iterates through the pbuf until
@               all the data in the receive buffer has been transferred.
@               the function then returns the pointer to the now filled
@               pbuf structure.
@
@ Arguments:        None.
@
@ Return Values:    r0 -  pointer to filled pbuf or NULL.
@
@ Local Variables:  r0 - general storage.
@               r1 - general storage.
```

```
@		r2 - counter 1.
@		r3 - counter 2.
@		r4 - counter 3.
@		r5 - pbuf start address
@		r6 - value storage
@		r7 - values loaded from variables
@ Shared Variables: RBQP, filled_buffer_num.
@ Global Variables: None.
@ Input:		EMAC.
@ Output:		None.
@ Error Handling:	None.
@
@ Registers Changed: r0.
@ Stack Depth;	8.
@
@ Algorithms:	None.
@
@ Data Structures:  RBQP.
@
@ Known Bugs:	None.
@ Limitations:	None.
@
@ Revision Hisotry: 3/3/2012	Josh Fromm  Outline Created
@		3/29/2012   Josh Fromm  Comments Updated
```

```
.global ether_receive
ether_receive:
    PUSH {r1, r2, r3, r4, r5, r6, r7, LR}

    BL ether_rx_available       @determine if there is a filled pbuf
                                @address of filled buffer now in r3
    CMP r0, #TRUE               @if not, indicate theres none
    BNE ether_receive_no_data

    @if there is a filled payload, extract the data from its RBQP chunk

    LDR r2, [r3]
    PUSH {r2, r3}         @save unaltered first word of RBQP and chunk address
                          @for later to allow reset of ownership and size
    LDR r5, =0xFFFFFFFC
    AND r2, r2, r5     @extract address of payload

    ADD r3, r3, #0x4    @go to next word to extract size
    LDR r4, [r3]
    LDR r1, =0x7FF
    AND r4, r4, r1      @extract the length of payload

    MOV r6, r2                 @save the address of the payload in r6
```

```
                    @for later to allow reset of ownership and size

@   BEQ receive_get_pbuf       @otherwise acquire a pbuf

receive_get_pbuf:

    LDR r0, =PBUF_RAW
    MOV r1, r4      @set size to be the value found in ether_rx_available
    LDR r2, =PBUF_POOL

    BL pbuf_alloc

@now begin to transfer data to the pbuf at r0
    @ current pbuf pointer address stored in r0
    LDR r2, =0x0       @initialize counter
    MOV r1, r0         @save pointer to beginning of pbuf (for return value)
                    @r1 will be used as a pointer to a variable pbuf instead
receive_main:
    LDR r3, [r1]       @set r3 to be the pointer to the next pbuf packet
    ADD r1, r1, #0x4   @increment to next word in pbuf structure
    LDR r5, [r1]       @set r6 to be payload address
    ADD r1, r1, #0x4
    LDR r7, [r1]       @set r7 to be size of current payload
```

```
LSR r7, #BITSPHW    @dont care about cumulative size of payloads, only
                    @interested in size of current payload

BL copy

ADD r6, r6, r7      @increment address of payload so next pbuf will be filled
                    @correctly

MOV r1, r3          @update current pbuf pointer to the next pointer

ADD r2, r2, r7      @keep track of how much data has been transferred
CMP r2, r4               @if all the data in the buffer is transferred
                         @function is done

BNE receive_main    @if not, continue transferring

POP {r2, r3}        @restore address of payload
                    @restore address of RBQP chunk addresss

LDR r4, =OWNERSHIP_MASK  @filter out the ownership bit only (leave wrap bit)
AND r2, r2, r4

ADD r3, r3, #WORD_SIZE       @move to size word
```

```
    LDR r1, =RBQP_BUFFER_SIZE   @reset size of RBQP chunk
    STR r1, [r3]

    SUB r3, r3, #WORD_SIZE          @move back one word

    STR r2, [r3]              @reset ownership of RBQP chunk

    B ether_receive_done

ether_receive_no_data:
    LDR r0, =NULL       @indicate there was no data to transfer
@   B ether_receive_done    @function is done

ether_receive_done:     @restore used registers and return
    POP {r1, r2, r3, r4, r5, r6, r7, LR}
    BX LR



.data


.align 10
RBQP:       @bufffer information space used by EMAC receive DMA
    .skip NUM_RBQP_BUFFERS*2*WORD_SIZE
```

```
.align 4

Transfer_Buff:  @buffer where outgoing data is stored prior to transmit

   .skip TRANSFER_SIZE


.align 4

RBQP_BUFF:      @buffer where received data is moved

   .skip NUM_RBQP_BUFFERS*RBQP_ALLOC


.end
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                         @
@ Genfuncs                                @
@                                         @
@ A file containing functions that are meant to be used by multiple other      @
@ files.                                  @
@                                         @
@ Table of Contents:                       @
@   1. Divide: Divides one argument by the other and returns a quotient value  @
@      and a remainder.                    @
@   2. Copy: copies a specified amount of memory from one passed memory address@
@      to another passed memory address.                @
@                                         @
@ Revision History:                        @
@                                         @
@   2012/2/24    Josh Fromm  Initial Revision                @
@   2012/3/4     Josh Fromm  Code updated to actually work          @
@   2012/3/3     Josh Fromm  Copy function added              @
@   2012/3/29    Josh Fromm  Comments updated                @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


@ Divide(numerator, denominator)
@
```

```
@ Description:      Divide takes a value to be divided and a value to divide by
@                   as input and returns a remainder of the division and the
@                   number of times the denominator goes into the numerator.
@
@ Operation:        Divide repeatedly subtracts the denominator from the
@                   numerator, checking each iteration to see if the the
@                   denominator exceeds the numerator. The number of subtractions
@                   that take place are stored in r2. Once the denominator exceeds
@                   the numerator, the function returns the remaining numerator
@                   as the divisions remainder and the count of how many subtractions
@                   took place as the quotient value.
@
@ Arguments:        r0 - value to be divided.
@                   r1 - value to divide by.
@
@ Return Values:    r0 - remainder.
@                   r1 - quotient value.
@
@ Local Variables:  r0 - remainder of each subtraction step.
@                   r1 - value to divide by, quotient.
@                   r2 - counter.
@ Shared Variables: None.
@ Global Variables: None.
@ Input:            None.
```

```
@ Output:          None.
@ Error Handling:   None.
@
@ Registers Changed: r1, r2.
@ Stack Depth;      1.
@
@ Algorithms:      None.
@
@ Data Structures:  None.
@
@ Known Bugs:      None.
@ Limitations:     None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm   Initial Revision
@                 2/7/2012     Josh Fromm  Updated to function better
@                 3/29/2012    Josh Fromm  Comments updated


.global Divide
Divide:
DivInit:

    PUSH {r2}      @Save used registers
    LDR r2, =0x0    @set initial counter value to zero
```

```
DivMain:        @loop used to calculate quotient and remainder

   CMP r0, r1      @determine if subtracting another divisor would cause.
             @numerator to become negative.
   BLT DivDone     @if so, value in r0 is the remainder of the division.

   SUB r0, r0, r1  @otherwise subtract the divisor from remaining value
   ADD r2, r2, #0x1    @and increase counter to indicate another division

   B DivMain       @loop back to divide step to repeat

DivDone:        @division is finsihed

   MOV r1, r2      @move number of times division occurred into output
   POP {r2}        @restore used registers
   BX LR           @return


@ copy
@
@ Description:     Copy moves all the elements of one buffer to another
@               buffer of equal or greater length
@
```

```
@ Operation:        Copy is passed a size in bytes, a pointer to a receive buffer
@                   and a pointer to a transmit buffer. Copy uses a
@                   counter to move the number of bytes indicated in size
@                   from the transfer buffer into the receive buffer.
@
@ Arguments:        r5 - Address of receive buffer.
@                   r6 - Address of transfer buffer.
@                   r7 - number of bytes to be copied
@
@ Return Values:    None.
@
@ Local Variables:  r0 - counter
@                   r1 - value at current transfer address
@                   r5 - address of current receive byte
@                   r6 - address of current transfer byte
@ Shared Variables: None.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      5.
@
```

```
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 3/3/2012    Josh Fromm  Outline Created


copy:
   PUSH {r0, r1, r5, r6, r7}   @save used registers


LDR r0, =0x0
copy_loop:                  @loop through each byte and move it
   LDRB r1, [r6]            @move a byte to receive buffer
   STRB r1, [r5]
   ADD r5, r5, #0x1          @increment byte position in the buffers
   ADD r6, r6, #0x1
   ADD r0, r0, #0x1          @keep track of how many moves have been done
   CMP r0, r7               @if we've moved the expected number of bytes
                        @function is done
   BNE copy_loop             @otherwise keep moving bytes
@   BEQ copy_done
```

```
copy_done:
   POP {r0, r1, r5, r6, r7}    @restore used registers and return
   BX LR

.end
```

```
@ keypad.inc
@ This file contains the constants used by the functions that run the VOIP
@ system's keypad input.
@
@ Revision History:
@
@   2012/2/5    Josh Fromm  Initial Revision


@ Pin Definitions


   .equ    KeyDat0,       0x04000000  @bit corresponding to data line 0 of keypad
   .equ    KeyDat1,       0x08000000  @bit corresponding to data line 1 of keypad
   .equ    KeyDat2,       0x10000000  @bit corresponding to data line 2 of keypad
   .equ    KeyDat3,       0x20000000  @bit corresponding to data line 3 of keypad
   .equ    KeyDatAll,     0x3C000000  @bit corresponding to all data lines of keypad
   .equ    KeyOE,         0x40000000  @bit corresponding to enable line of keypad
   .equ    KeyRDY,        0x80000000  @bit corresponding to ready line of keypad
   .equ    ALL_PINS,      0xFFFFFFFF  @all PIOC bits


@ PIO Control Words


   .equ    Key_Enable,    0xFC000000        @enables keypad pio pins
   .equ    PID_4_Set,     0x00000010        @used to initialize clock for
                                  @PID 4
```

```
    .equ    SMR_SET,        0x00000060
    .equ    AIC_IECR_KEYS,  0x11            @control word to enable PIOC
                                            @interrupts

@ General Definitions

    .equ    Invalid_Key,    0xFF
    .equ    Datshiftval,    0x1A            @amount read data must be
                                            @shifted to give correct key
                                            @code
    .equ    SHIFT_KEYCODE,  0x7
    .equ    SHIFT_VALUE,    0x10            @value to be added to shifted keys
                                            @equal to the number of keys on
                                            @the keypad
    .equ    SYS_BIT,        0x2             @bit corresponding to system
                                            @interrupts
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                                                                              @
@ keypad.s                                                                     @
@                                                                              @
@ Initialization, event handling, and functions of VOIP keypad.                @
@                                                                              @
@ Table of Contents:                                                           @
@   1. KeypadInit: Sets ups registers and shared variables needed for keypad   @
@      functions to run. Also installs the keypad event handler                @
@   2. KeypadPressHandler: Called whenever user presses a key. Stores the key  @
@      code of the key press or shifts the keypad.                             @
@   3. call_halt: Ends SSC0 interrupts so that no audio data is input or       @
@      output.                                                                 @
@   4. key_available: Checks whether there is an unhandled key press or not.   @
@   5. getkey: returns the key code of the most recent key press.              @
@                                                                              @
@ Revision History:                                                            @
@                                                                              @
@   2012/2/5    Josh Fromm  Initial Revision                                   @
@   2012/3/29    Josh Fromm  Comments updated                                  @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


.include    "at91rm9200.inc"
```

```
.include    "armvoip.inc"

.include    "keypad.inc"


.text

.arm

.align 2


@@@ Keypad initialization



@ KeypadInit()

@

@ Description:      KeypadInit sets up the registers needed to run the rest of

@                   the keypad functions. KeypadInit also installs

@                   KeypadPressHandler as the interrupt handler for PIOC

@                   interrupts.

@

@ Operation:        Moves many control words into appropriate registers.

@

@ Arguments:        None.

@

@ Return Values:    None.

@

@ Local Variables:  r0 - contains addresses and control words.
```

```
@               r1 - contains addresses and control words.
@ Shared Variables: keypressed, shiftkey.
@ Global Variables: None.
@ Input:          None.
@ Output:         None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm  Outline Created
@               3/29/2012    Josh Fromm  Comments updated



.global KeypadInit
KeypadInit:
```

```
@ KeypadInit sets up the registers needed to run the keypad functions

    PUSH {r0, r1}          @save used registers

    LDR r0, =PIOC_PER      @enable PIOC pins
    LDR r1, =ALL_PINS
    STR r1, [r0]

    LDR r0, =PIOC_ODR      @Disable output on all PIOC Lines
    LDR r1, =ALL_PINS
    STR r1, [r0]

    LDR r0, =PMC_PCER      @Enable clock for PIOC
    LDR r1, =PID_4_Set
    STR r1, [r0]

    LDR r0, =PIOC_IER      @set key ready pin to be an interrupt
    LDR r1, =KeyRDY
    STR r1, [r0]

    LDR r0, =PIOC_SODR     @allow keypad data lines to be read by CPU
    LDR r1, =KeyDatAll
    STR r1, [r0]
```

```
LDR r0, =PIOC_OER      @Enable output for output enable pin
LDR r1, =KeyOE
STR r1, [r0]

LDR r0, =PIOC_CODR     @Set output enable to low (active) for keypad
STR r1, [r0]

LDR r0, =AIC_SMR4      @set priority for PIOC interrupts
LDR r1, =SMR_SET
STR r1, [r0]

LDR r0, =AIC_SVR4      @set pio C interrupt to go to keypad event handler
LDR r1, =KeypadPressHandler
STR r1, [r0]

LDR r0, =AIC_IECR      @enable PIO C interrupts
LDR r1, =AIC_IECR_KEYS
STR r1, [r0]

LDR r0, =AIC_IDCR      @disable system interrupts. These were causing some
                       @problems by triggering for seemingly no reason.
LDR r1, =SYS_BIT
STR r1, [r0]
```

```
LDR r0, =keypressed     @at first there are no available keys to handle, so
                        @the variable indicating available keys is false
LDR r1, =FALSE
STR r1, [r0]

LDR r0, =shiftkey       @keypad should be unshifted at first
LDR r1, =FALSE
STR r1, [r0]

LDR r0, =PIOC_ISR   @read the value of the interrupt status register
                    @to allow future interrupts
LDR r1, [r0]
LDR r1, [r0]

LDR r0, =AIC_EOICR   @signal end of interrupt to refresh any standing
                     @interrupt
LDR r1, =TRUE
STR r1, [r0]

POP {r0, r1}          @restore used registers
BX LR                 @after initialization jump to main loop
```

```
@ KeypadPressHandler()
@
@ Description:     KeypadPressHandler is called whenever a change on the RDY
@                  line is detected (when a key is presssed). KeypadPressHandler
@                  saves the key code of the msot recent key press and indicates
@                  that a key is available. KeypadPressHandler also checks
@                  if the pressed key is the shift key. If so, it does not save
@                  a key code or indicate a key is available. Instead, it causes
@                  all future key presses to yield higher key codes until the
@                  shift key is pressed again. This allows the keypad to be able
@                  to have about twice as many distinct keys as normal.
@
@ Operation:       KeypadPressHandler reads the value of the keypad data lines
@                  and (if the read value isn't the shift key) stores the value
@                  in the shared variable keyvalue. KeypadPressHandler then
@                  sets keypressed to TRUE to indicate a key is available for
@                  handling. If the value read from keypad data indicates the
@                  shift key has been pressed, KeypadPressHandler sets the
@                  shared variable shiftkey to TRUE to indicate future key
@                  presses should have the number of keys on the keypad
@                  added to their key code to yield the key codes of the
@                  shifted keypad.
```

```
@
@ Arguments:        None.
@
@ Return Values:    None.
@
@ Local Variables:  r0 - contains addresses and values.
@                   r1 - contains addresses and values.
@ Shared Variables: keypressed, shiftkey, keyvalue.
@ Global Variables: None.
@ Input:            Keypad presses.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: None.
@ Stack Depth;      2.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm  Outline Created
```

```
@               3/29/2012    Josh Fromm  Comments updated



KeypadPressHandler:

    SUB  LR, LR, #4     @first correct systems pipeline
    STMFD SP!, {LR}
    PUSH {r0, r1}       @save all used registers



    LDR r1, =PIOC_PDSR     @load current value of pio data pins
    LDR r0, [r1]
    LDR r1, =KeyRDY        @determine if current interrupt is due to key
                @press or key release
    AND r0, r1, r0

    CMP r0, #KeyRDY        @if keypress is invalid then keep looping until
                @a valid key code is read
    BNE KeypadPressHandlerDone

keycode_get:
    LDR r1, =PIOC_PDSR     @load current value of pio data pins
    LDR r0, [r1]
    LDR r1, =KeyDatAll     @mask all bits that aren't from keypad data
```

```
        AND r0, r1, r0


        LSR r0, #Datshiftval    @shift read value to lowest 4 bits, this gives
                        @final keycode.
        CMP r0, #SHIFT_KEYCODE  @if the keypress was a shift key, must treat
                        @specially
        BEQ have_shift_key
@       BNE no_shift_key


no_shift_key:   @current keypress is not the shift key, treat normally
        LDR     r1, =keyvalue    @save acquired keycode
        STR     r0, [r1]


        LDR     r0, =keypressed  @set keypressed to true to indicate a key was pressed
        LDR     r1, =TRUE
        STR     r1, [r0]


        B       KeypadPressHandlerDone  @function can return


have_shift_key: @current keypress is shift key, must switch status of shift
                @without updating current keycode or key pressed status
        LDR     r0, =shiftkey       @load value of shift key
        LDR     r1, [r0]
        EOR     r1, r1, #TRUE       @invert and store that value
```

```
        STR     r1, [r0]
@       B       KeypadPressHandlerDone  @interrupt can end

KeypadPressHandlerDone:

        LDR r0, =PIOC_ISR   @read the value of the interrupt status register
                            @to allow future interrupts
        LDR r1, [r0]
        LDR r1, [r0]

        LDR r0, =AIC_EOICR   @signal end of interrupt
        LDR r1, =TRUE
        STR r1, [r0]

        POP {r0, r1}    @restore registers
        LDMFD SP!, {PC}^



@ key_available()
@
@ Description:      This function returns the value sotred in keypressed. This
@                   indicates whether a key has been pressed. The value is
@                   returned in r0.
```

@

@ Operation: Loads and returns the value in keypressed.

@

@ Arguments: None.

@

@ Return Values: r0 - TRUE if a key press is available, FALSE if not.

@

@ Local Variables: r0 - value of keypressed.

@ r1 - address of keypressed.

@ Shared Variables: keypressed.

@ Global Variables: None.

@ Input: None.

@ Output: None.

@ Error Handling: None.

@

@ Registers Changed: r0.

@ Stack Depth; 1.

@

@ Algorithms: None.

@

@ Data Structures: None.

@

@ Known Bugs: None.

@ Limitations: None.

```
@
@ Revision Hisotry: 2/5/2012    Josh Fromm  Outline Created
@                3/29/2012    Josh Fromm  Comments updated

.global key_available
key_available:

    PUSH    {r1}        @save used register
    LDR     r1, =keypressed
    LDR     r0, [r1]    @load keypress value in to return register

    POP     {r1}        @restore used register
    BX LR               @return



@ getkey()
@
@ Description:    getkey returns the value of the most recent key press.
@                 if the key code of the most recent key press is invalid,
@                 getkey blocks until a valid key code is available.
@
@ Operation:      getkey loads the value stored in keyvalue and checks if the
@                 keypad is in its shifted state. If the keypad is shifted,
@                 getkey adds the shift value onto the key code to yield the
```

```
@               shifted key code of the pressed key. If the keypad is not
@               shifted, getkey simply returns the value of keyvalue. Before
@               returning a value, getkey checks if the value in r0 is invalid,
@               if it is, getkey loops back to the beginning of the function
@               and attempts to get a valid key code.
@
@ Arguments:        None.
@
@ Return Values:    r0 - key code of most recently pressed key.
@
@ Local Variables:  r0 - register addresses and loaded values, key code.
@                   r1 - register addreses and values.
@ Shared Variables: keyvalue, shiftkey.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@
@ Registers Changed: r0.
@ Stack Depth;      1.
@
@ Algorithms:       None.
@
@ Data Structures:  None.
```

```
@
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision Hisotry: 2/5/2012    Josh Fromm  Outline Created
@                   3/29/2012   Josh Fromm  Comments updated


.global getkey
getkey:

    PUSH {r1}       @save register

    LDR r0, =keypressed     @first indicate key press is being processed
    LDR r1, =FALSE
    STR r1, [r0]

invalidkeyloop:     @determine if shift key is pressed and handle accordingly
    LDR r1, =shiftkey       @load value of shiftkey variable
    LDR r0, [r1]
    CMP r0, #TRUE           @if value is true, keycode must be shifted
    BEQ getkey_shifted      @if true, must add on shift value
@       BNE getkey_unshifted    @otherwise handle normally
```

```
getkey_unshifted:            @keycode does not need to be shifted
       LDR r1, =keyvalue      @load value of key code into return register
       LDR r0, [r1]

       B getkeyend            @function is done

getkey_shifted:              @keycode must be shifted
       LDR r1, =keyvalue      @load value of of key code into return register
       LDR r0, [r1]
       ADD r0, r0, #SHIFT_VALUE    @add shift value to keycode variable
@      B getkeyend

getkeyend:                   @if key code is valid then we can return

       CMP r0, #Invalid_Key    @if keypress is invalid then keep looping until
                            @a valid key code is read
       BEQ invalidkeyloop
       POP {r1}             @restore used register and return
       BX LR

@The Data Segment
.data
```

```
keypressed:     @variable indicating whether a key press is available

   .word '?'


keyvalue:       @variable containing the key code of the most recent key press

   .word '?'


shiftkey:       @variable indicating whether the keypad is in a shifted state

   .word '?'


.end
```

```
@ timers.inc
@ Constants used to run the timer functions in timers.s
@
@ Revision History:
@
@   2012/3/6    Josh Fromm  Initial Revision


    .equ    DRAM_REFRESH_RATE,  0x3 @refresh DRAM every 4 milliseconds
    .equ    DRAM_CHUNK,         0x40 @number of reads that must take place to
                                @refresh 1/4th of the DRAM
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@                                                           @

@ timers.s                                                  @

@                                                           @

@ Initialization, event handling, and basic functions of VOIP timers.         @

@                                                           @

@ Table of Contents:                                        @

@   1. Timer0Init: Initializes TC0 and causes it to trigger interrupts every   @

@      millisecond.                                         @

@   2. Timer0Handler: Increments the counter used by elapsed_time to determine @

@      how many milliseconds have passed. Also reads from 1/4th of the columns @

@      in the system's DRAM to refresh them.                @

@   3. elapsed_time: Returns the number of milliseconds since elapsed_time     @

@      was last called.                                     @

@                                                           @

@ Revision History:                                         @

@                                                           @

@   2012/3/6    Josh Fromm  Initial Revision                @

@   2012/3/29   Josh Fromm  Comments Updated                @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


.include    "at91rm9200.inc"

.include    "armvoip.inc"
```

```
.include    "timers.inc"


.text

.arm

.align 2


@ TimerOInit

@ Description:     This function sets up timer 0 of the system to cause an

@                  interrupt every millisecond. The function also installs

@                  the timer 0 event handler that is called when a timer

@                  0 interrupt is detected. Shared variables used by timer

@                  functions are also set up.

@

@ Operation:       TimerOInit sets up timer and AIC registers with the needed

@                  control words.

@

@ Arguments:       None.

@ Return Values:   None.

@ Local Variables: r0 - addresses

@                  r1 - control words

@ Shared Variables: DRAM_count, elapsed_time_count.

@ Global Variables: None.

@ Input:           None.

@ Output:          None.
```

```
@ Error Handling:   None.
@ Registers Changed: None.
@ Stack Depth:      2 Words.
@ Algorithms:       None.
@ Data Structures:  None.
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision History: 3/6/2012    Josh Fromm  Initial Revision
@                   3/29/2012   Josh Fromm  Comments Updated


.global Timer0Init
Timer0Init:

    PUSH {r0, r1}       @save used registers

    LDR r0, =DRAM_START @first write to beginning of dram, this helps stabilize
                        @the dram for some reason
    LDRB r1, =0x12
    STRB r1, [r0]

    LDR r0, =PMC_PCER   @enable tc0 peripheral clock
    LDR r1, =0x20000
```

```
STR r1, [r0]

LDR r0, =AIC_SMR17  @set timer 0 interrupt mode
LDR r1, =0x60
STR r1, [r0]

LDR r0, =AIC_SVR17  @set timer 0 interrupt to cause event handler to be called
LDR r1, =Timer0Handler
STR r1, [r0]

LDR r0, =AIC_IECR   @enable PID17 interrupts
LDR r1, =0x20000
STR r1, [r0]

LDR r0, =AIC_IDCR   @disable sytem interrupts to prevent problems
LDR r1, =0x2
STR r1, [r0]

LDR r0, =TC0_CCR    @disable timer 0 so it can be set
LDR r1, =0x1
STR r1, [r0]

LDR r0, =TC0_CMR    @cause clock to reset at compare with RC
LDR r1, =0xC000
```

```
STR r1, [r0]

LDR r0, =TC0_IER     @set interrupt to generate on RC compare
LDR r1, =0x10
STR r1, [r0]

LDR r0, =TC0_RC     @set RC to cause an interrupt every millisecond for
                  @75 Mhz clock speed
LDR r1, =0x927C
STR r1, [r0]

LDR r0, =TC0_CCR    @initiate clock 0
LDR r1, =0x5
STR r1, [r0]

LDR r0, =DRAM_count  @set timer function counters to zero
LDR r1, =0x0
STR r1, [r0]
LDR r0, =elapsed_time_count @initial elapsed time is 0 milliseconds
STR r1, [r0]

POP {r0, r1}       @restore registers and return
BX LR
```

```
@ Timer0Handler
@ Description:      This function is called every millisecond through a timer
@                   interrupt. When called it increments the counters used to
@                   keep track of how many milliseconds have passed since
@                   elapsed_time has been called and how many milliseconds have
@                   passed since DRAM was used to refresh the systems DRAM.
@                   Each of these counts is incremented by one when the function
@                   is called. The function also reads from 1/4th of the DRAM
@                   to keep that section refreshed.
@
@ Operation:        Timer0Handler first increments elapsed_time_count by 1. Next,
@                   Timer0Handler determines which address of DRAM to begin reading
@                   at by multiplying the value in DRAM_count by the number of
@                   columns that are read each millisecond. Timer0Handler then
@                   reads from the calculated addresses of DRAM. Finally,
@                   Timer0Handler checks if the value in DRAM_count has reached
@                   its maximum value and sets it back to 0 if it has.
@
@ Arguments:        None.
@ Return Values:    None.
@ Local Variables:  r0 - addresses
@                   r1 - control words
@ Shared Variables: DRAM_count, elapsed_time_count.
```

```
@ Global Variables: None.

@ Input:           None.

@ Output:          None.

@ Error Handling:   None.

@ Registers Changed: None.

@ Stack Depth:      2 Words.

@ Algorithms:       None.

@ Data Structures:  None.

@ Known Bugs:       None.

@ Limitations:      None.

@

@ Revision History: 3/6/2012    Josh Fromm  Initial Revision



Timer0Handler:


    SUB LR, LR, #4      @first correct systems pipeline

    STMFD SP!, {LR}     @save link register and other used registers

    PUSH {r0, r1, r2}


    LDR r0, =elapsed_time_count   @increment the count of elapsed milliseconds since

                                  @last elapsed_time function call

    LDR r1, [r0]

    ADD r1, r1, #0x1
```

```
    STR r1, [r0]

    LDR r0, =DRAM_count   @get current refresh number
    LDR r1, [r0]

    LDR r0, =DRAM_CHUNK
    MUL r1, r1, r0        @multiply number of reads per iteration by the refresh
                    @number of this cycle to get start address to read from
    ADD r1, r1, #DRAM_START
    LDR r2, =0x0          @set counter to zero


DRAMRefresh:          @set up a DRAM read of 1/4th of the columns in DRAM

    LDR r0, =DRAM_CHUNK
    CMP r2, r0        @check if 1/4th of DRAM has been read from yet
    BEQ DRAMCountUpdate @if so, we're done refreshing the DRAM

    LDRB r0, [r1]     @otherwise, read a byte from the current column
    ADD r1, r1, #0x1   @increment column address by 1 for next read

    ADD r2, r2, #0x1   @increment counter to keep track of reads
    B DRAMRefresh      @continue refreshing DRAM
```

```
DRAMCountUpdate:        @if all columns have been refreshed, restart refresh cycle


    LDR r0, =DRAM_count @check if the value in DRAM_count is at its maximum value

    LDR r1, [r0]

    CMP r1, #DRAM_REFRESH_RATE

    BNE DRAMCountInc   @increment DRAM if cycle shouldnt be reset

    LDR r1, =0x0       @if DRAM_count is maximum value, set it back to 0

    STR r1, [r0]

    B Timer0HandlerDone


DRAMCountInc:   @increment DRAM_count by 1

    LDR r0, =DRAM_count @load value in DRAM_count

    LDR r1, [r0]

    ADD r1, r1, #0x1    @add 1

    STR r1, [r0]        @store new value

@   B Timer0HandlerDone @function can finish


Timer0HandlerDone:


    LDR r0, =TC0_SR     @read timer 0 status register to reset interrupt status

    LDR r1, [r0]


    LDR r0, =AIC_EOICR  @signal end of interrupt

    LDR r1, =TRUE
```

```
    STR r1, [r0]


    POP {r0, r1, r2}    @restore used registers
    LDMFD SP!, {PC}^    @then return



@ elapsed_time
@ Description:      This function returns the number of milliseconds since it
@                   was last called.
@ Operation:        Each millisecond, Timer0Handler increments elapsed_time_count.
@                   This means that elapsed_time simply needs to return the
@                   value in that shared value and then reset it.
@
@ Arguments:        None.
@ Return Values:    r0 - milliseconds since function was last called
@ Local Variables:  r0 - addresses
@                   r1 - general purpose
@                   r2 - general purpose
@ Shared Variables: elapsed_time_count.
@ Global Variables: None.
@ Input:            None.
@ Output:           None.
@ Error Handling:   None.
@ Registers Changed: None.
```

```
@ Stack Depth:      2 Words.

@ Algorithms:       None.

@ Data Structures:  None.

@ Known Bugs:       None.

@ Limitations:      None.

@

@ Revision History: 3/6/2012    Josh Fromm  Initial Revision


.global elapsed_time

elapsed_time:


    PUSH {r1, r2}   @save used registers


    LDR r1, =elapsed_time_count    @first get value of elapsed_time_count

    LDR r0, [r1]


    LDR r2, =0x0                   @then reset elapsed_time_count

    STR r2, [r1]


    POP {r1, r2}                   @restore registers and return

    BX LR



@The Data Segment
```

```
.data

DRAM_count:        @current refresh cycle number of DRAM
   .word '?'

elapsed_time_count: @number of milliseconds since elapsed_time was last called
   .word '?'

.end
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@                                               @

@ timers.s                                          @

@                                               @

@ Initialization, event handling, and basic functions of VOIP timers.       @

@                                               @

@ Table of Contents:                                    @

@   1. Timer0Init: Initializes TC0 and causes it to trigger interrupts every   @

@      millisecond.                                 @

@   2. Timer0Handler: Increments the counter used by elapsed_time to determine @

@      how many milliseconds have passed. Also reads from 1/4th of the columns @

@      in the system's DRAM to refresh them.                    @

@   3. elapsed_time: Returns the number of milliseconds since elapsed_time     @

@      was last called.                             @

@                                               @

@ Revision History:                                 @

@                                               @

@   2012/3/6    Josh Fromm  Initial Revision                    @

@   2012/3/29   Josh Fromm  Comments Updated                    @

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


.include    "at91rm9200.inc"

.include    "armvoip.inc"
```

```
.include    "timers.inc"


.text

.arm

.align 2


@ TimerOInit

@ Description:      This function sets up timer 0 of the system to cause an

@                   interrupt every millisecond. The function also installs

@                   the timer 0 event handler that is called when a timer

@                   0 interrupt is detected. Shared variables used by timer

@                   functions are also set up.

@

@ Operation:        TimerOInit sets up timer and AIC registers with the needed

@                   control words.

@

@ Arguments:        None.

@ Return Values:    None.

@ Local Variables:  r0 - addresses

@                   r1 - control words

@ Shared Variables: DRAM_count, elapsed_time_count.

@ Global Variables: None.

@ Input:            None.

@ Output:           None.
```

```
@ Error Handling:   None.
@ Registers Changed: None.
@ Stack Depth:      2 Words.
@ Algorithms:       None.
@ Data Structures:  None.
@ Known Bugs:       None.
@ Limitations:      None.
@
@ Revision History: 3/6/2012    Josh Fromm  Initial Revision
@                   3/29/2012   Josh Fromm  Comments Updated


.global Timer0Init
Timer0Init:

    PUSH {r0, r1}       @save used registers

    LDR r0, =DRAM_START @first write to beginning of dram, this helps stabilize
                        @the dram for some reason
    LDRB r1, =0x12
    STRB r1, [r0]

    LDR r0, =PMC_PCER   @enable tc0 peripheral clock
    LDR r1, =0x20000
```

```
STR r1, [r0]

LDR r0, =AIC_SMR17  @set timer 0 interrupt mode
LDR r1, =0x60
STR r1, [r0]

LDR r0, =AIC_SVR17  @set timer 0 interrupt to cause event handler to be called
LDR r1, =Timer0Handler
STR r1, [r0]

LDR r0, =AIC_IECR   @enable PID17 interrupts
LDR r1, =0x20000
STR r1, [r0]

LDR r0, =AIC_IDCR   @disable sytem interrupts to prevent problems
LDR r1, =0x2
STR r1, [r0]

LDR r0, =TC0_CCR    @disable timer 0 so it can be set
LDR r1, =0x1
STR r1, [r0]

LDR r0, =TC0_CMR    @cause clock to reset at compare with RC
LDR r1, =0xC000
```

```
STR r1, [r0]

LDR r0, =TC0_IER     @set interrupt to generate on RC compare
LDR r1, =0x10
STR r1, [r0]

LDR r0, =TC0_RC     @set RC to cause an interrupt every millisecond for
                    @75 Mhz clock speed
LDR r1, =0x927C
STR r1, [r0]

LDR r0, =TC0_CCR    @initiate clock 0
LDR r1, =0x5
STR r1, [r0]

LDR r0, =DRAM_count  @set timer function counters to zero
LDR r1, =0x0
STR r1, [r0]
LDR r0, =elapsed_time_count @initial elapsed time is 0 milliseconds
STR r1, [r0]

POP {r0, r1}       @restore registers and return
BX LR
```

```
@ Timer0Handler
@ Description:    This function is called every millisecond through a timer
@                 interrupt. When called it increments the counters used to
@                 keep track of how many milliseconds have passed since
@                 elapsed_time has been called and how many milliseconds have
@                 passed since DRAM was used to refresh the systems DRAM.
@                 Each of these counts is incremented by one when the function
@                 is called. The function also reads from 1/4th of the DRAM
@                 to keep that section refreshed.
@
@ Operation:      Timer0Handler first increments elapsed_time_count by 1. Next,
@                 Timer0Handler determines which address of DRAM to begin reading
@                 at by multiplying the value in DRAM_count by the number of
@                 columns that are read each millisecond. Timer0Handler then
@                 reads from the calculated addresses of DRAM. Finally,
@                 Timer0Handler checks if the value in DRAM_count has reached
@                 its maximum value and sets it back to 0 if it has.
@
@ Arguments:      None.
@ Return Values:  None.
@ Local Variables: r0 - addresses
@                 r1 - control words
@ Shared Variables: DRAM_count, elapsed_time_count.
```

```
@ Global Variables: None.

@ Input:            None.

@ Output:           None.

@ Error Handling:   None.

@ Registers Changed: None.

@ Stack Depth:      2 Words.

@ Algorithms:       None.

@ Data Structures:  None.

@ Known Bugs:       None.

@ Limitations:      None.

@

@ Revision History: 3/6/2012    Josh Fromm  Initial Revision



Timer0Handler:


    SUB LR, LR, #4      @first correct systems pipeline

    STMFD SP!, {LR}     @save link register and other used registers

    PUSH {r0, r1, r2}


    LDR r0, =elapsed_time_count   @increment the count of elapsed milliseconds since

                                  @last elapsed_time function call

    LDR r1, [r0]

    ADD r1, r1, #0x1
```

```
    STR r1, [r0]

    LDR r0, =DRAM_count   @get current refresh number
    LDR r1, [r0]

    LDR r0, =DRAM_CHUNK
    MUL r1, r1, r0         @multiply number of reads per iteration by the refresh
                        @number of this cycle to get start address to read from
    ADD r1, r1, #DRAM_START
    LDR r2, =0x0           @set counter to zero


DRAMRefresh:           @set up a DRAM read of 1/4th of the columns in DRAM

    LDR r0, =DRAM_CHUNK
    CMP r2, r0         @check if 1/4th of DRAM has been read from yet
    BEQ DRAMCountUpdate @if so, we're done refreshing the DRAM

    LDRB r0, [r1]      @otherwise, read a byte from the current column
    ADD r1, r1, #0x1    @increment column address by 1 for next read

    ADD r2, r2, #0x1    @increment counter to keep track of reads
    B DRAMRefresh       @continue refreshing DRAM
```

```
DRAMCountUpdate:        @if all columns have been refreshed, restart refresh cycle

    LDR r0, =DRAM_count @check if the value in DRAM_count is at its maximum value
    LDR r1, [r0]
    CMP r1, #DRAM_REFRESH_RATE
    BNE DRAMCountInc   @increment DRAM if cycle shouldnt be reset
    LDR r1, =0x0       @if DRAM_count is maximum value, set it back to 0
    STR r1, [r0]
    B Timer0HandlerDone

DRAMCountInc:   @increment DRAM_count by 1
    LDR r0, =DRAM_count @load value in DRAM_count
    LDR r1, [r0]
    ADD r1, r1, #0x1    @add 1
    STR r1, [r0]        @store new value
@   B Timer0HandlerDone @function can finish

Timer0HandlerDone:

    LDR r0, =TC0_SR     @read timer 0 status register to reset interrupt status
    LDR r1, [r0]

    LDR r0, =AIC_EOICR  @signal end of interrupt
    LDR r1, =TRUE
```

```
    STR r1, [r0]

    POP {r0, r1, r2}    @restore used registers
    LDMFD SP!, {PC}^    @then return


@ elapsed_time
@ Description:     This function returns the number of milliseconds since it
@                  was last called.
@ Operation:       Each millisecond, Timer0Handler increments elapsed_time_count.
@                  This means that elapsed_time simply needs to return the
@                  value in that shared value and then reset it.
@
@ Arguments:       None.
@ Return Values:   r0 - milliseconds since function was last called
@ Local Variables: r0 - addresses
@                  r1 - general purpose
@                  r2 - general purpose
@ Shared Variables: elapsed_time_count.
@ Global Variables: None.
@ Input:           None.
@ Output:          None.
@ Error Handling:  None.
@ Registers Changed: None.
```

```
@ Stack Depth:      2 Words.

@ Algorithms:       None.

@ Data Structures:  None.

@ Known Bugs:       None.

@ Limitations:      None.

@

@ Revision History: 3/6/2012    Josh Fromm  Initial Revision


.global elapsed_time

elapsed_time:


    PUSH {r1, r2}   @save used registers


    LDR r1, =elapsed_time_count    @first get value of elapsed_time_count

    LDR r0, [r1]


    LDR r2, =0x0                  @then reset elapsed_time_count

    STR r2, [r1]


    POP {r1, r2}                  @restore registers and return

    BX LR



@The Data Segment
```

```
.data

DRAM_count:        @current refresh cycle number of DRAM
   .word '?'

elapsed_time_count: @number of milliseconds since elapsed_time was last called
   .word '?'

.end
```

```
/****************************************************************************/
/*                                                                          */
/*                               BUFFERS.H                                  */
/*                          General Definitions                             */
/*                              Include File                                */
/*                         VoIP Telphone Project                            */
/*                                EE/CS 52                                  */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constant and structure definitions and function
   declarations for the buffer management functions for the VoIP Telephone
   defined in buffers.c.


   Revision History:
      6/6/06   Glen George       Initial revision.
      5/26/08  Glen George       Updated return types of buffer functions.
      2/28/11  Glen George       Changed buffers to be short ints (16-bits)
                                 instead of unsigned chars (8-bits).
      3/9/11   Glen George       Added prototypes for get_xmit_next_ptr() and
                                 get_rcv_next_ptr().
*/
```

```
#ifndef  I__BUFFERS_H__
   #define  I__BUFFERS_H__

/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"

/* constants */

/* number of transmit and receive buffers */
#define  NUM_RX_BUFFERS  20        /* must be at least 4, more is better */
#define  NUM_TX_BUFFERS  20        /* must be at least 4, more is better */
```

```
/* structures, unions, and typedefs */

   /* none */




/* function declarations */


/* initialization functions */

void  init_buffers(void);          /* initialize the buffer system */

void  reset_rx_buffer(void);    /* reset the receive buffer to empty */

void  reset_tx_buffer(void);    /* reset the transmit buffer to empty */


/* status functions */

int  rx_available(void);           /* there is a buffer available for mic data */

int  xmit_available(void);        /* there is a buffer ready to send over ethernet */

int  tx_available(void);           /* there is a buffer available to play */

int  rcv_available(void);          /* there is a buffer ready to fill from ethernet */


/* blocking functions for getting buffers */

short int  *get_rx_buffer(void);          /* get a buffer to fill with mic data */

short int  *get_rx_next_ptr(void);      /* get the pointer to next buffer w/o allocating it */

short int  *get_xmit_buffer(void);      /* get a buffer to send over ethernet */
```

```
short int  *get_xmit_next_ptr(void);  /* get the pointer to next ethernet buffer w/o allocating it
*/

short int  *get_tx_buffer(void);          /* get a buffer to send to the speaker */

short int  *get_tx_next_ptr(void);      /* get the pointer to next buffer w/o allocating it */

short int  *get_rcv_buffer(void);        /* get a buffer to fill with ethernet data */

short int  *get_rcv_next_ptr(void);    /* get the buffer to fill with ethernet w/o allocating it */



#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                                CALLPROC                                  */
/*                        Call Processing Functions                         */
/*                         VoIP Telephone Project                           */
/*                                EE/CS 52                                  */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the key processing functions for initiating and ending
   calls for the VoIP Telephone Project.  These functions are called by the
   main loop of the system.  The functions included are:
      do_answer - handle picking up an incoming call
      do_call   - initiate an outgoing call
      end_call  - end a call

   The local functions included are:
      none

   The global variable definitions included are:
      none
```

```
   Revision History
      6/3/06   Glen George      Initial revision.
      3/8/11   Glen George      Updated comments.
*/



/* library include files */
  /* none */


/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "keyproc.h"
#include  "error.h"
#include  "callutil.h"




/*
   do_answer

   Description:      This function handles answering a phone call.  It is
```

assumed that there is an incoming call and this function
is called when the phone goes "off hook".

Operation: The function initiates the call by calling the function connect_incoming() and then returns the status STAT_CONNECTED.

Arguments: cur_status (enum status) - the current system status (ignored).
key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_CONNECTED).

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

Author: Glen George

```
   Last Modified:    June 3, 2006

*/

enum status  do_answer(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* connect to the incoming call */
    connect_incoming();


    /* and return that we are connected now */
    return  STAT_CONNECTED;

}




/*
```

do_call

Description: This function handles the <Send> key to start an outgoing call.

Operation: It just starts the call by calling initiate_outgoing() and then returns the status STAT_CONNECTING.

Arguments: cur_status (enum status) - the current system status (ignored).

key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_CONNECTING).

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

```
  Author:          Glen George
  Last Modified:   June 3, 2006

*/

enum status  do_call(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* start the outgoing call */
    initiate_outgoing();


    /* and return the new status - STAT_CONNECTING */
    return  STAT_CONNECTING;

}
```

/*

end_call

Description: This function handles the end of a call, when the user hangs up.

Operation: It disconnects the call by calling disconnect_call() and returns the status STAT_IDLE.

Arguments: cur_status (enum status) - the current system status (ignored).

key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_IDLE).

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

```
   Author:          Glen George
   Last Modified:   June 3, 2006

*/

enum status  end_call(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* disconnect the call */
    disconnect_call();



    /* and return with the new status - idle */
    return  STAT_IDLE;

}
```

```
/****************************************************************************/
/*                                                                          */
/*                              CALLUTIL                                    */
/*                      Calling Utility Functions                           */
/*                       VoIP Telephone Project                             */
/*                               EE/CS 52                                   */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the utility functions for dealing with calls for the
   VoIP Telephone Project.  The IP number of the "other end" of the call is
   also defined in this file (locally).  The functions included are:
      call_connected    - has the other end connected with us
      connect_incoming  - connect to an incoming call
      disconnect_call   - end a call
      get_calling_IP    - get the calling IP address (accessor)
      get_calling_name  - get the calling name (accessor)
      incoming_call     - is there an incoming call
      initiate_outgoing - initiate an outgoing call
      process_call      - process a continuing call
      set_calling_IP    - set the calling IP address (mutator)
      set_calling_name  - set the calling name (mutator)
      start_call        - start an outgoing call
```

The local functions included are:

ring_fill - fill buffer with a ring tone

busy_fill - fill buffer with a busy tone

sine_wave - get the value of the sine function

The locally global variable definitions included are:

calling_IP - the IP number of the other party

calling_name - the name of the other party

last_status - the lassed status of call (to find changes in status)

ring_busy_timer - timer for timing ring and busy tones

Revision History

6/3/06 Glen George Initial revision (only dummy versions of the functions).

6/6/06 Glen George Started filling in real functions.

6/8/06 Glen George Continued filling in real functions.

2/28/11 Glen George Added call to call_halt() in disconnect_call.

3/10/11 Glen George Updated code to use TCP functions to implement actual calling (major changes).

3/16/11 Glen George Fixed some bugs in the ring and busy tone generation and changed the ring tone to 500 Hz modulated by 20 Hz.

```
*/



/* library include files */
  /* none */

/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "callutil.h"
#include  "buffers.h"
#include  "tcpconn.h"
#include  "error.h"




/* local function declarations */
static void  ring_fill(short int *p, int size);      /* fill buffer w/ring tone */
static void  busy_fill(short int *p, int size); /* fill buffer w/busy tone */
static int   sine_wave(long int angle);             /* compute sine function */
```

```
/* locally global variables */


/* IP address of the "other end" */
static unsigned long int  calling_IP;


/* name of the "other end" */
static char  calling_name[MAX_NAME_LEN];


/* the lagged status of the call (used to check for changes) */
static enum tcp_conn_status  last_status;


/* ring/busy timer */
static unsigned long int  ring_busy_timer;




/* mutators/accessors */



/*
  get_calling_IP
```

Description: This function returns the IP address of the "other end" of the call.

Operation: The value of the shared variable calling_IP is returned.

Arguments: None.

Return Value: (unsigned long int) - the IP address of the "other end" of the call, either who is calling or who was called.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: calling_IP (accessed) - value to return.

Author: Glen George

Last Modified: June 3, 2006

*/

```
unsigned long int  get_calling_IP()
{
    /* variables */
      /* none */



    /* just return the value of calling_IP */
    return  calling_IP;

}



/*
   get_calling_name

   Description:      This function returns the name of the "other end" of the
                     call.

   Operation:        The value of the shared variable calling_name is
                     returned.
```

```
   Arguments:        None.
   Return Value:     (const char *) - the name of the "other end" of the call,
                     either who is calling or who was called.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: calling_name (accessed) - value to return.

   Author:           Glen George
   Last Modified:    March 8, 2011

*/

const char  *get_calling_name()
{
    /* variables */
      /* none */
```

```
    /* just return a pointer to calling_name */
    return  calling_name;

}




/*
   set_calling_IP

   Description:      This function sets the IP address of the "other end" of
                     the call.

   Operation:        The shared variable calling_IP is set to the passed
               value.

   Arguments:        ip (unsigned long int) - the new value of the IP address
                              of the "other end" of the call,
                                             either who is calling or who was
                                             called.
```

```
   Return Value:     None.


   Input:           None.

   Output:          None.


   Error Handling:   None.


   Algorithms:      None.

   Data Structures:  None.


   Shared Variables: calling_IP (changed) - changed to passed value.


   Author:          Glen George

   Last Modified:    June 3, 2006


*/


void  set_calling_IP(unsigned long int ip)

{

    /* variables */

      /* none */
```

```
    /* set the calling IP address */
    calling_IP = ip;


    /* done - return */
    return;

}


/*
   set_calling_name

   Description:      This function sets the name of the "other end" of the
                     call to a copy of the passed value.

   Operation:        The shared variable calling_name is set to the passed
                value.  The passed string is copied character by
                     character with care taken to not overwrite the buffer.

   Arguments:        name (const char *) - pointer to the new value for the
                            name of the "other end" of the
```

```
                                     call, either who is calling or who
                                     was called.
   Return Value:     None.


   Input:          None.
   Output:         None.


   Error Handling:   If the passed string is longer than the buffer for the
                  name, it is truncated.


   Algorithms:      None.
   Data Structures:  None.


   Shared Variables: calling_name (changed) - changed to passed value.


   Author:          Glen George
   Last Modified:    March 8, 2011

*/


void  set_calling_name(const char *name)
{
    /* variables */
    int  i;            /* general loop index */
```

```
    /* copy the passed string up to the end of the string or maximum length */
    for (i = 0; ((i < (MAX_NAME_LEN - 1)) && (name[i] != '\0')); i++)
        calling_name[i] = name[i];

    /* <null> terminate the string */
    calling_name[i] = '\0';


    /* done copying the name, return */
    return;

}



/* status functions */


/*
   incoming_call
```

Description: This function determines whether or not there is an incoming call. If someone is trying to connect with this phone, TRUE is returned.

Operation: The result of calling have_tcp_connection() is returned.

Arguments: None.

Return Value: (char) - TRUE is someone is trying to connect with us, FALSE otherwise.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

Author: Glen George

Last Modified: March 8, 2011

```
*/

char  incoming_call()
{
   /* variables */
     /* none */



   /* return whether or not there is a call */
   return  have_tcp_connection();

}



/*
   call_connected

   Description:      This function determines whether or not the "other end"
                  has connected with us.  If the connection has been
                         established TRUE is returned.
```

```
  Operation:       The connection status is queried and TRUE is returned if
                   there is a connection.

  Arguments:       None.
  Return Value:    (char) -  TRUE if the connection has been established,
                             FALSE otherwise.

  Input:           None.
  Output:          None.

  Error Handling:  None.

  Algorithms:      None.
  Data Structures: None.

  Shared Variables: None.

  Author:          Glen George
  Last Modified:   March 9, 2011

*/

char  call_connected()
{
```

```
    /* variables */
      /* none */



    /* return whether or not have a connection */
    return  (tcp_connection_status() != CALL_NO_CONNECTION);

}




/* call management functions */



/*
   initiate_outgoing

   Description:      This function initiates an outgoing call to the currently
                     set IP address.

   Operation:        An attempt is made to connect to calling_IP.
```

```
   Arguments:       None.
   Return Value:    None.

   Input:           None.
   Output:          None.

   Error Handling:  None.

   Algorithms:      None.
   Data Structures: None.

   Shared Variables: calling_IP (accessed) - IP to connect to.

   Author:          Glen George
   Last Modified:   March 9, 2011

*/

void  initiate_outgoing()
{
    /* variables */
      /* none */
```

```
    /* start trying to get a connection and return */
    tcp_connection_connect(calling_IP);
    return;

}
```

```
/*
   start_call

   Description:      This function starts an outgoing call.

   Operation:        The buffers are initialized, the call state is set to
                     connected, the ring and busy tones are started and the
                     call is started.

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.
```

```
   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: last_status (changed)     - set to CALL_NO_CONNECTION.
                ring_busy_timer (changed) - set to 0.

   Author:          Glen George
   Last Modified:    March 9, 2011

*/

void  start_call()
{
    /* variables */
      /* none */



    /* reset the buffers for a new call */
    reset_rx_buffer();
    reset_tx_buffer();
```

```
    /* start the audio portion of the call */
    /*    Note: buffer had better be available now */
    call_start(get_rx_buffer());



    /* the prior call status is that the call is not connected yet */
    last_status = CALL_NO_CONNECTION;



    /* the ring and busy timer starts over for those tones */
    ring_busy_timer = 0;



    /* done setting up the call, return */
    return;

}



/*
```

connect_incoming

Description: This function connects to an incoming call.

Operation: It calls the TCP connection function to answer the call.
It then initializes the buffer and audio code to start
the call.

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: last_status (changed) - set to CALL_NO_CONNECTION.

Author: Glen George

Last Modified: March 9, 2011

```
*/

void  connect_incoming()
{
   /* variables */
    /* none */



   /* answer the call */
   tcp_connection_answer();


   /* reset the buffers for a new call */
   reset_rx_buffer();
   reset_tx_buffer();


   /* start the audio portion of the call */
   /*    Note: buffer had better be available now */
   call_start(get_rx_buffer());


   /* the prior call status is that the call is not connected yet */
```

```
    last_status = CALL_NO_CONNECTION;



    /* done setting up the call, return */
    return;

}




/*
   process_call

   Description:      This function processes a continuing call.  If there are
                  buffers to play and the update function is ready, a
                        buffer is passed.  If there are buffers to fill and the
                        update function is ready, a buffer is passed.  The
                        ethernet interface is also checked for buffers.

   Arguments:       None.
   Return Value:    None.

   Input:           None.
```

```
   Output:         None.


   Error Handling:   None.


   Algorithms:      None.
   Data Structures:  None.


   Shared Variables: None.


   Author:         Glen George
   Last Modified:    March 9, 2011

*/


void  process_call()
{
    /* variables */
    enum tcp_conn_status  cur_status;          /* current connection state */




    /* get the current connection state */
    cur_status = tcp_connection_status();
```

```
/* if the current connection state has changed, need to reset the */
/*    the receive buffers, they may have old tone data in them */
if (cur_status != last_status)
    reset_tx_buffer();


/* always update the last status */
last_status = cur_status;



/* now check the status of the connection */
switch  (cur_status)  {

    case CALL_RINGING:        /* connection is ringing on the other end */

                              /* generate a buffer of ringing tone if */
                              /*    there is room for it */
                              if (rcv_available())  {
                                  /* have a buffer, get and fill it */
                                  ring_fill(get_rcv_buffer(), AUDIO_BUFLEN);
                              }

                              /* if there is a buffer to transmit, need */
                    /*    to discard it */
                              if (xmit_available())  {
```

```
                /* have a buffer, dump it */
                get_xmit_buffer();
            }
            break;

case CALL_BUSY:     /* connection is busy on the other end */

            /* generate a buffer of busy signal if */
            /*    there is room for it */
            if (rcv_available())  {
                /* have a buffer, get and fill it */
                busy_fill(get_rcv_buffer(), AUDIO_BUFLEN);
            }

            /* if there is a buffer to transmit, need */
        /*    to discard it */
            if (xmit_available())  {
                /* have a buffer, dump it */
                get_xmit_buffer();
            }
            break;

case CALL_CONNECTED: /* are talking on the connection */
```

```
                    /* check if there is a buffer to transmit */
                      if (xmit_available())  {
                        /* buffer is available - try sending it */
                        if (tcp_connection_tx(get_xmit_next_ptr(), AUDIO_BUFLEN))
                          /* actually sent it, let buffer functions know */
                          get_xmit_buffer();
                      }

                      /* check if have room for a received buffer */
                      if (rcv_available())  {
                        /* have room - try to get a buffer */
                        if (tcp_connection_rx(get_rcv_next_ptr(), AUDIO_BUFLEN))
                          /* got the buffer, so allocate it */
                             get_rcv_buffer();
                      }
                      break;

    default:          /* some other status, must have aborted call */

                      /* if there is a buffer to transmit, need */
              /*    to discard it */
                      if (xmit_available())  {
                        /* have a buffer, dump it */
                        get_xmit_buffer();
```

```
            }
            break;
}

/* try to play and receive any buffers */

/* check if a receive buffer is available for recording into */
if (rx_available())  {

  /* have a receive buffer, see if update is ready */
    if (update_rx(get_rx_next_ptr()))
      /* it wanted the buffer - actually allocate it */
      get_rx_buffer();
}

/* check if a transmit buffer is available for playing */
if (tx_available())  {

  /* have a transmit buffer, see if update is ready */
    if (update_tx(get_tx_next_ptr()))
      /* it wanted the buffer - actually allocate it */
      get_tx_buffer();
}
```

```
    /* all done processing the call for now - return */

    return;

}




/*
   disconnect_call


   Description:      This function ends a call.


   Operation:        The TCP connection is closed and the audio I/O is halted.


   Arguments:        None.
   Return Value:     None.


   Input:            None.
   Output:           None.


   Error Handling:   None.
```

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    March 9, 2011

*/

void  disconnect_call()
{
    /* variables */
      /* none */



    /* close the TCP connection */
    tcp_connection_close();

    /* and halt the call */
    call_halt();
```

```
    /* done ending the call - return */
    return;

}




/* ring and busy tone generators */



/*
   ring_fill

   Description:      This function fills the passed buffer of the passed size
                     with a ring tone.  A ring tone is a 500 Hz signal
                     modulated by a 20 Hz signal at -13 dBm.  The tone is on
                           for 2 seconds and off for 4 seconds.

   Operation:        A static shared variable is used to keep track of the
                     position in the waveform pattern when the function is
                           called.  The function computes the the waveform pattern
                           and writes it to the buffer.
```

Arguments: p (short int *) - pointer to buffer to be filled with

data.

size (int)   - size of the passed buffer.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: ring_busy_timer (changed) - updated on each call.

Author: Glen George

Last Modified: March 16, 2011

*/

```
static void  ring_fill(short int *p, int size)
{
   /* variables */
```

```
long int  sig_500;              /* value of the 420 Hz tone signal */

long int  sig_20;               /* value of the 40 Hz modulating signal */


long int  angle;                /* angle to find the sine for */


int       i;                /* general loop index */




/* fill the buffer with data */

for (i = 0; i < size; i++)  {


    /* check if past the end of the tone + silent portion */

        if (++ring_busy_timer >= (RING_TONE_TIME + RING_SILENT_TIME))

            /* timer has wrapped, reset it */

            ring_busy_timer = 0;


        /* get the angle for both waveforms (with a factor of 25) */

        /* calculation is complicated to keep it from overflowing */

        angle = (((500 * ring_busy_timer) / SAMPLE_RATE) * SINE_RESOLUTION) +

                    (((500 * ring_busy_timer) % SAMPLE_RATE) * SINE_RESOLUTION) /
SAMPLE_RATE;


        /* get the 500 Hz sine wave */
```

```
        sig_500 = sine_wave(angle);


        /* now get the 20 Hz modulation wave (25 = 500 Hz / 20 Hz) */
        sig_20 = sine_wave(angle / 25);
        /* the modulation signal should always be positive */
        if (sig_20 < 0)
          sig_20 = -sig_20;



        /* should the tone be output or should it be silent */
        if (ring_busy_timer < RING_TONE_TIME)
          /* outputting the tone (make it -13 dBm) */
          p[i] = (sig_500 * sig_20) / (4096 * 4);
        else
          /* output no tone - store a 0 */
          p[i] = 0;
  }



  /* all done filling the buffer - return */
  return;

}
```

```
/*
  busy_fill


  Description:      This function fills the passed buffer of the passed size
                    with a busy tone.  A busy tone is a 600 Hz signal
                    modulated by a 120 Hz signal at -13 dBm.  The tone is on
                            for 0.5 seconds and off for 0.5 seconds.


  Operation:        A static shared variable is used to keep track of the
                    position in the waveform pattern when the function is
                            called.  The function computes the the waveform pattern
                            and writes it to the buffer.


  Arguments:        p (short int *) - pointer to buffer to be filled with
                                      data.
                    size (int)      - size of the passed buffer.
  Return Value:     None.


  Input:            None.
  Output:           None.
```

```
   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: ring_busy_timer (changed) - updated on each call.

   Author:           Glen George
   Last Modified:    March 16, 2011

*/

static void  busy_fill(short int *p, int size)
{
    /* variables */
    long int  sig_600;              /* value of the 600 Hz tone signal */
    long int  sig_120;              /* value of the 120 Hz modulating signal */

    long int  angle;                /* angle to find the sine for */

    int       i;                    /* general loop index */
```

```
/* fill the buffer with data */
for (i = 0; i < size; i++)  {

   /* check if past the end of the tone + silent portion */
       if (++ring_busy_timer >= (BUSY_TONE_TIME + BUSY_SILENT_TIME))
          /* timer has wrapped, reset it */
          ring_busy_timer = 0;

       /* get the angle for both waveforms (with a factor of 5) */
       /* calculation is complicated to keep it from overflowing */
       angle = (((600 * ring_busy_timer) / SAMPLE_RATE) * SINE_RESOLUTION) +
         (((600 * ring_busy_timer) % SAMPLE_RATE) * SINE_RESOLUTION) / SAMPLE_RATE;

       /* get the 600 Hz sine wave */
       sig_600 = sine_wave(angle);

       /* now get the 120 Hz modulation wave (5 = 600 Hz / 120 Hz) */
       sig_120 = sine_wave(angle / 5);
       /* the modulation signal should always be positive */
       if (sig_120 < 0)
          sig_120 = -sig_120;


       /* should the tone be output or should it be silent */
```

```
        if (ring_busy_timer < BUSY_TONE_TIME)
            /* outputting the tone (make it -13 dBm) */
            p[i] = (sig_600 * sig_120) / (4096 * 4);
        else
            /* output no tone - store a 0 */
            p[i] = 0;
    }


    /* all done filling the buffer - return */
    return;

}




/*
   sine_wave

   Description:      This function returns the value of a sine wave for the
                     passed angle (units of 360/1024 degrees).  The returned
                     value is 13-bits.
```

Operation: A table is used to lookup the sine wave. The table only
covers one quadrant of the sine wave so the argument is
first checked to see which quadrant it is in.

Arguments: angle (long int) - angle (units of 360/SIN_RESOLUTION
degrees) for which to find the value
of the sine function.

Return Value: (int) - value of the sine function for the passed angle
with the maximum amplitude being +/- 4096.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

Author: Glen George

Last Modified: March 16, 2011

*/

```
static int  sine_wave(long int angle)
{
   /* variables */

   /* samples of one fourth of a cycle of a sine wave */
   static const short int  sin_wave[SINE_RESOLUTION / 4] =  {
         0,     25,     50,     75,    100,    126,    151,    176,
       201,    226,    251,    276,    301,    326,    351,    376,
       401,    426,    451,    476,    501,    526,    551,    576,
       601,    626,    651,    675,    700,    725,    750,    774,
       799,    824,    848,    873,    897,    922,    946,    971,
       995,   1019,   1044,   1068,   1092,   1116,   1141,   1165,
      1189,   1213,   1237,   1261,   1285,   1308,   1332,   1356,
      1380,   1403,   1427,   1450,   1474,   1497,   1521,   1544,
      1567,   1590,   1613,   1636,   1659,   1682,   1705,   1728,
      1751,   1773,   1796,   1819,   1841,   1864,   1886,   1908,
      1930,   1952,   1975,   1996,   2018,   2040,   2062,   2084,
      2105,   2127,   2148,   2170,   2191,   2212,   2233,   2254,
      2275,   2296,   2317,   2337,   2358,   2378,   2399,   2419,
      2439,   2459,   2480,   2499,   2519,   2539,   2559,   2578,
      2598,   2617,   2636,   2656,   2675,   2694,   2713,   2731,
      2750,   2769,   2787,   2805,   2824,   2842,   2860,   2878,
      2896,   2913,   2931,   2948,   2966,   2983,   3000,   3017,
```

```
    3034, 3051, 3068, 3084, 3101, 3117, 3133, 3149,
    3165, 3181, 3197, 3213, 3228, 3244, 3259, 3274,
    3289, 3304, 3319, 3333, 3348, 3362, 3377, 3391,
    3405, 3419, 3433, 3446, 3460, 3473, 3486, 3499,
    3512, 3525, 3538, 3551, 3563, 3575, 3587, 3600,
    3611, 3623, 3635, 3646, 3658, 3669, 3680, 3691,
    3702, 3712, 3723, 3733, 3744, 3754, 3764, 3774,
    3783, 3793, 3802, 3811, 3821, 3830, 3838, 3847,
    3856, 3864, 3872, 3880, 3888, 3896, 3904, 3911,
    3919, 3926, 3933, 3940, 3947, 3953, 3960, 3966,
    3972, 3978, 3984, 3990, 3995, 4001, 4006, 4011,
    4016, 4021, 4026, 4030, 4035, 4039, 4043, 4047,
    4051, 4054, 4058, 4061, 4064, 4067, 4070, 4073,
    4075, 4078, 4080, 4082, 4084, 4086, 4087, 4089,
    4090, 4091, 4092, 4093, 4094, 4094, 4095, 4095
};

int  sign;          /* sign of the sine wave, based on quadrant */

int  index;         /* index into the table */



/* reduce the angle to one cycle */
```

```
angle %= SINE_RESOLUTION;
/* and make sure it is positive */
if (angle < 0)
   angle += SINE_RESOLUTION;

/* find the sign of the sine wave */
if (angle < (SINE_RESOLUTION / 2))
   /* first two quadrants are positive */
       sign = +1;
else
   /* third and fourth quadrants are negative */
       sign = -1;



/* lastly get the table index */
/* note: there is only one quadrant in the table so the index is a */
/*      function of which quadrant we are in */
if (((angle / (SINE_RESOLUTION / 4)) & 0x01) == 0)
   /* in quadrant I or III, go through table in normal order */
       index = angle % (SINE_RESOLUTION / 4);
else
   /* in quadrant II or IV, go through table in reverse order */
       index = (SINE_RESOLUTION / 4) - angle % (SINE_RESOLUTION / 4) - 1;
```

```
   /* now return the sine function (only have one quadrant of function */

   return  (sign * sin_wave[index]);


}
```

```
/****************************************************************************/
/*                                                    */
/*                        CALLUTIL.H                        */
/*                   Calling Utility Functions                   */
/*                        Include File                        */
/*                   VoIP Telephone Project                   */
/*                         EE/CS 52                         */
/*                                                    */
/****************************************************************************/


/*
   This file contains the constants and function prototypes for the calling
   utility functions for the VoIP Telephone Project which are defined in
   callutil.c and memproc.c.



   Revision History
      6/3/06   Glen George      Initial revision.
      3/9/11   Glen George      Made numerous changes to fully support making
                                actual phone calls.
*/
```

```
#ifndef  I__CALLUTIL_H__
   #define  I__CALLUTIL_H__



/* library include files */
  /* none */


/* local include files */
  /* none */




/* constants */


/* resolution of sine calculation (points per 360 degrees) */
#define  SINE_RESOLUTION   1024


/* length of the ring tone in sample rate units (2 seconds) */
#define  RING_TONE_TIME    (2 * SAMPLE_RATE)


/* length of the ring silent period in sample rate units (4 seconds) */
#define  RING_SILENT_TIME  (4 * SAMPLE_RATE)
```

```
/* length of the busy tone period in sample rate units (0.5 seconds) */
#define  BUSY_TONE_TIME    (SAMPLE_RATE / 2)

/* length of the busy silent time in sample rate units (0.5 seconds) */
#define  BUSY_SILENT_TIME  (SAMPLE_RATE / 2)




/* structures, unions, and typedefs */
   /* none */




/* function declarations */

/* call status functions */
char  incoming_call(void);      /* there is an incoming call */
char  call_connected(void);    /* other end has connected with us */

/* call management functions */
void  initiate_outgoing(void); /* initiate an outgoing call */
```

```
void  start_call(void);             /* start an outgoing call */
void  connect_incoming(void);           /* connect to an incoming call */
void  process_call(void);         /* process a continuing call */
void  disconnect_call(void);    /* end a call */

/* IP accessor/mutator functions */
unsigned long int   get_calling_IP(void);
const char        *get_calling_name(void);
void              set_calling_IP(unsigned long int ip);
void              set_calling_name(const char *name);

/* memory functions */
void  init_memory(void);        /* initialize the IP address memory system */

#endif
```

```
/****************************************************************************/
/*                                                        */
/*                         ERROR                         */
/*                  Error Processing Functions                  */
/*                   VoIP Telephone Project                   */
/*                         EE/CS 52                         */
/*                                                        */
/****************************************************************************/


/*
  This file contains the erro processing functions for the VoIP Telephone
  Project.  These functions are called whenever an error occurs.  Currently
  they do nothing, but they exist to allow error handling to be added to the
  system.  The functions included are:
    process_error - process the passed error code

  The local functions included are:
    none

  The global variable definitions included are:
    none


  Revision History
```

```
     6/3/06   Glen George       Initial revision.
*/




/* library include files */
  /* none */


/* local include files */
#include  "error.h"




/*
   process_error


   Description:      This function processes the passed error.  The error is
                     indicated by the value of the passed enumerated type.
                     Currently the function does nothing, but is here to allow
                     for error handling to be added in the future.


   Arguments:        e (enum error_type) - type of the error that occurred.
   Return Value:     None.
```

```
   Input:          None.
   Output:         None.

   Error Handling:   None.

   Algorithms:      None.
   Data Structures:  None.

   Shared Variables: None.

   Author:          Glen George
   Last Modified:   May 31, 2006

*/

void  process_error(enum error_type e)
{
    /* variables */
      /* none */

    /* do nothing for now, just return */
    return;
```

```
}
```

```
/****************************************************************************/
/*                                                    */
/*                      ERROR.H                       */
/*                  Error Processing                  */
/*                    Include File                    */
/*               VoIP Telphone Project                */
/*                     EE/CS  52                      */
/*                                                    */
/****************************************************************************/


/*
   This file contains the error processing definitions for the VoIP Telephone
   Project.  This includes constant and enum definitions along with function
   declarations for the error processing functions defined in error.c.



   Revision History:
      6/3/06   Glen George      Initial revision.
      6/8/06   Glen George      Added error value (UNKNOWN_ETHER_NAME).
      3/10/11  Glen George      Added a number of error values related to
                                TCP communication.
*/
```

```
#ifndef  I__ERROR_H__
   #define  I__ERROR_H__



/* library include files */
  /* none */

/* local include files */
  /* none */




/* constants */

  /* none */




/* structures, unions, and typedefs */

/* error types */
```

```
enum error_type  {
   UNKNOWN_KEYCODE_INIT,          /* unknown keycode in memory or IP initialization */
   IP_ADDRESS_OVERFLOW,   /* overflow in entered IP address */
   IP_ADDRESS_UNDERFLOW,          /* underflow in entered IP address */
   MEMORY_ADDRESS_OVERFLOW,  /* overflow in entered memory location */
   MEMORY_ADDRESS_UNDERFLOW,          /* underflow in entered memory location */
   BAD_MEMORY_ADDRESS,            /* bad memory address on save/recall */
   UNKNOWN_ETHER_NAME,          /* unknown ethernet interface name */
   UNKNOWN_TCP_STATUS,            /* unknown TCP status returned */
   MULTIPLE_CONNECTIONS, /* multiple connections attempted */
   NETERR_NOLISTEN,          /* can't setup listener */
   NETERR_NOBIND,            /* can't bind the port */
   NETERR_SEND,              /* error sending a packet */
   NETERR_CLOSE,             /* error closing a connection */
   NETERR_NOCONNECT,                /* error setting up a connection */
   NETERR_UNKNOWN_PACKET,       /* unknown packet type received */
   NETERR_GENERAL            /* general networking error */
};




/* function declarations */
```

```
void  process_error(enum error_type);         /* process an error */


#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                              ETHERNET                                    */
/*                    Ethernet Interface Functions                          */
/*                       VoIP Telephone Project                             */
/*                              EE/CS 52                                    */
/*                                                                          */
/****************************************************************************/


/*
  This file contains the ethernet interface low-level driver functions for
  the VoIP Telephone Project.  These functions are used to interface the
  VoIP code with the lwIP code.  This file should only be compiled into the
  project if lwIP is being used.  The functions included are:
    ethernetif_init   - initialize the ethernet interface
    ethernetif_input  - get input from the ethernet interface
    ethernetif_output - send output to the ethernet interface
    init_networking   - initialize the ethernet interface and library

  The local functions included are:
    none

  The global variable definitions included are:
    et0 - the network interface to use for the phone
```

```
   Revision History
     6/11/09  Glen George       Initial revision (based on lwIP ethernetif.c
                                file).
     6/13/09  Glen George       Fixed polarity problems with return values
                                for ether_init and ether_transmit.
     3/8/11   Glen George       Get the MAC address from interfac.h instead
                                of hard coding it.
*/



/* library include files */
#include  "lwip/opt.h"
#include  "lwip/ip_addr.h"
#include  "lwip/mem.h"
#include  "lwip/memp.h"
#include  "lwip/pbuf.h"
#include  "lwip/netif.h"
#include  "netif/etharp.h"
#include  "lwip/stats.h"
#include  "ipv4/lwip/ip.h"
#include  "lwip/err.h"
```

```
#include  "lwip/tcp.h"
#include  "lwip/raw.h"

/* local include files */
#include  "voipdefs.h"
#include  "ethernet.h"
#include  "interfac.h"




/* locally global variables */
static struct netif  et0;/* the network interface */




/*
   init_ethernet

   Description:      This function takes care of all of the initialization for
                     the ethernet interface.  It first sets up the lwIP code.
                     Then it initializes the ethernet hardware.
```

```
  Operation:       The lwIP library is initialized.  Then the network
                interface is initialized with default addresses.  And
                        finally the hardware is setup.


  Arguments:       None.
  Return Value:    None.


  Input:           None.
  Output:          None.


  Error Handling:  None.


  Algorithms:      None.
  Data Structures: None.


  Shared Variables: et0 - the network interface for the system.


  Author:          Glen George
  Last Modified:   June 9, 2009

*/


void  init_networking()
{
```

```
  /* variables */
  struct ip_addr  gateway;            /* default gateway */
  struct ip_addr  netmask;            /* default network mask */
  struct ip_addr  default_addr;       /* default IP address */



  /* setup the lwIP library */
/*  mem_init();
  memp_init();
  pbuf_init();
  netif_init();
  etharp_init();
  stats_init();
*/

  stats_init();
  mem_init();
  memp_init();
  pbuf_init();
  netif_init();
  ip_init();
  etharp_init();
  raw_init();
```

```
    /* setup the default gateway, net mask, and IP address */
    IP4_ADDR(&gateway, 192, 168, 1, 1);
    IP4_ADDR(&netmask, 255, 255, 255, 0);
    IP4_ADDR(&default_addr, 192, 168, 1, 21);


    /* tell lwIP about the network interface */
    netif_add(&et0, &default_addr, &netmask, &gateway, NULL, ethernetif_init, ip_input);

    /* set et0 as the default interface and set up the interface */
    netif_set_default(&et0);
    netif_set_up(&et0);


    /* done initializing the ethernet interface, return */
    return;

}
```

/*

ethernetif_init

Description: This function handles initialization of the ethernet interface itself. It is meant to be passed as a parameter to netif_add().

Operation: The passed network interface is initialized and then the hardware is initialized.

Arguments: etx (struct netif *) - pointer to the network interface whose hardware is to be initialized.

Return Value: (err_t) - ERR_OK if the interface is initialized, or ERR_IF if there is an error initializing the hardware.

Input: None.

Output: None.

Error Handling: If there is an error initializing the hardware, ERR_IF is returned.

Algorithms: None.

Data Structures: None.

```
  Shared Variables: None.

  Author:          Glen George
  Last Modified:   March 8, 2011

*/

err_t  ethernetif_init(struct netif *etx)
{
   /* variables */
   char  err;                /* whether or not there is an error */



   /* make sure the argument is reasonable */
   LWIP_ASSERT("etx != NULL", (etx != NULL));


   /* setup the interface host name */
   #if LWIP_NETIF_HOSTNAME
      etx->hostname = "lwip";
   #endif
```

```
/* no state */
etx->state = NULL;

/* set the interface name */
etx->name[0] = 'e';
etx->name[1] = 't';

/* use etharp_output() to do output (saves a function call) */
etx->output = etharp_output;
etx->linkoutput = ethernetif_output;



/* set MAC hardware address length */
etx->hwaddr_len = ETHARP_HWADDR_LEN;

/* and set MAC hardware address itself */
etx->hwaddr[0] = (MAC_ADDR_L        & 0xFF);
etx->hwaddr[1] = ((MAC_ADDR_L >> 8)  & 0xFF);
etx->hwaddr[2] = ((MAC_ADDR_L >> 16) & 0xFF);
etx->hwaddr[3] = ((MAC_ADDR_L >> 24) & 0xFF);
etx->hwaddr[4] = (MAC_ADDR_H        & 0xFF);
etx->hwaddr[5] = ((MAC_ADDR_H >> 8) & 0xFF);
```

```
    /* set the maximum transfer unit */
    etx->mtu = 1500;

    /* device capabilities */
    /* can broadcast, can do ARP, and there is an active link */
    etx->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP |
            NETIF_FLAG_LINK_UP;


    /* initialize the hardware */
    err = !ether_init();


    /* return whether or not there was an error */
    if (err)
      /* had an error - return ERR_IF */
      return ERR_IF;
    else
      /* no error - return ERR_OK */
      return ERR_OK;

}
```

```
/*
  ethernetif_input

  Description:      This function gets input from the passed network
                    interface and processes it.  Only ARP and IP packets are
                          processed.  For efficiency it should only be called when
                    packets are available.

  Operation:        The passed network interface is initialized and then the
                    hardware is initialized.

  Arguments:        None.
  Return Value:     None.

  Input:            None.
  Output:           None.

  Error Handling:   If there is no packet available, the function returns.

  Algorithms:       None.
  Data Structures:  None.
```

```
   Shared Variables: None.


   Author:          Glen George
   Last Modified:   June 9, 2009

*/


void  ethernetif_input()
{
    /* variables */
    struct pbuf     *p;                 /* pointer to a received packet */

    struct eth_hdr  *ethhdr;     /* pointer to the packet header */



    /* get the received packet (may be a buffer chain) */
    p = ether_receive();



    /* only process the packet if there was one */
    if (p != NULL)  {

        /* have a packet - figure out what it is from the header */
        ethhdr = p->payload;
```

```
    /* what kind of packet was this */
  switch (htons(ethhdr->type))  {

    case ETHTYPE_IP:        /* a standard IP packet */
                            /* update the ARP table if we can */
                            etharp_ip_input(&et0, p);
                            /* remove the ethernet header (to get to IP) */
                            pbuf_header(p, -((s16_t) sizeof(struct eth_hdr)));
                            /* now pass the packet to IP processing which will free it */
                            ip_input(p, &et0);
                            break;

    case ETHTYPE_ARP:       /* an ARP packet - process it */
                etharp_arp_input(&et0, (struct eth_addr *) &(et0.hwaddr), p);
                            /* buffer is freed by etharp_arp_input() */
                            break;

        default:            /* some other kind of packet */
                            /* ignore it - just free the memory */
                            pbuf_free(p);
                            p = NULL;       /* no packet */
                            break;
  }
```

```
    }



    /* done processing the input packet (if there was one) - return */

    return;

}





/*

   ethernetif_output


   Description:      This function outputs the passed packet over the ethernet

                  interface and then frees the memory for the packet.  Note

                  that the passed packet could be in a pbuf chain, not in a

                          single pbuf.


   Operation:       The passed packet is output via the low-level output

                  function and then the packet is freed.


   Arguments:       etx (struct netif *) - pointer to the network interface

                                    over which the packet is to be
```

```
                                            output (ignored).
                b (struct pbuf *)    - pointer to the packet (chain) to
                                  output.
  Return Value:     (err_t) - ERR_OK if the packet is successfully output, or
                  ERR_IF if there is an error outputing the packet.

  Input:           None.
  Output:          None.

  Error Handling:   If there is an error outputing the packet, ERR_IF is
                returned and the passed buffer is still freed.

  Algorithms:       None.
  Data Structures:  None.

  Shared Variables: None.

  Author:           Glen George
  Last Modified:    June 9, 2009

*/

err_t  ethernetif_output(struct netif *etx, struct pbuf *b)
{
```

```
    /* variables */
    char  err;                  /* whether or not there was an error */



    /* send the packet watching for an error */
    err = !ether_transmit(b);


    /* error or not, free the packet (if there was one) */
    if (b != NULL)
        pbuf_free(b);



    /* return whether or not there was an error */
    if (err)
        /* had an error - return ERR_IF */
        return ERR_IF;
    else
        /* no error - return ERR_OK */
        return ERR_OK;

}
```

```
/****************************************************************************/
/*                                                                          */
/*                             ETHERNET.H                                   */
/*                    Ethernet Interface Functions                          */
/*                            Include File                                  */
/*                       VoIP Telephone Project                             */
/*                              EE/CS 52                                    */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the constant and structure definitions and the function
   declarations for the ethernet interface low-level driver functions for the
   VoIP Telephone Project.


   Revision History:
      6/11/09  Glen George      Initial revision.
      6/12/09  Glen George      Fixed directory for netif.h.
      3/10/11  Glen George      Fixed value of TCP_TIMEOUT.
*/
```

```
/* make sure the file isn't already included */
#ifndef  I__ETHERNET_H__
   #define  I__ETHERNET_H__



/* library include files */
#include  "lwip/pbuf.h"
#include  "lwip/netif.h"
#include  "lwip/err.h"

/* local include files */
   /* none */



/* constants */

#define  TCP_TIMEOUT   250  /* TCP timeout in ms */
```

```
/* structures, unions, and typedefs */

   /* none */




/* function declarations */


void   ethernetif_input(void);            /* driver input */

err_t  ethernetif_init(struct netif *);  /* driver initialization */

err_t  ethernetif_output(struct netif *, struct pbuf *);  /* driver output */

void   init_networking(void);             /* networking system initialization */



#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                              INTERFAC.H                                  */
/*                         Interface Definitions                            */
/*                             Include File                                 */
/*                        VoIP Telephone Project                            */
/*                               EE/CS 52                                   */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constants for interfacing between the C code and
   the assembly code/hardware.  This is a sample interface file to allow test
   compilation and linking of the code.


   Revision History:
      6/3/06   Glen George      Initial revision.
      6/7/06   Glen George      Added ETHER_INTF definition.
      3/8/11   Glen George      Added MAC_ADDR_H and MAC_ADDR_L definitions.
*/
```

```c
#ifndef  I__INTERFAC_H__
   #define  I__INTERFAC_H__



/* library include files */
  /* none */

/* local include files */
  /* none */




#define  DRAM_START      0x40000000
#define  MEMORY_SIZE     32
#define  SAMPLE_RATE     8000

#define  MAC_ADDR_H      0xEA09
#define  MAC_ADDR_L      0x87654321

#define  ETHER_INTF      "et0"

#define  KEY_0           13
#define  KEY_1           0
```

```
#define  KEY_2         1
#define  KEY_3         2
#define  KEY_4         4
#define  KEY_5         5
#define  KEY_6         6
#define  KEY_7         8
#define  KEY_8         9
#define  KEY_9         10
#define  KEY_ESC        3
#define  KEY_BACKSPACE   11
#define  KEY_SEND       15
#define  KEY_OFFHOOK     12
#define  KEY_ONHOOK     14
#define  KEY_SET_IP     16
#define  KEY_SET_SUBNET  17
#define  KEY_SET_GATEWAY 18
#define  KEY_MEM_SAVE    19
#define  KEY_MEM_RECALL  20
#define  KEY_ILLEGAL    31

#define  STATUS_IDLE        0
#define  STATUS_OFFHOOK      1
#define  STATUS_RINGING      2
#define  STATUS_CONNECTING   3
```

```
#define  STATUS_CONNECTED    4
#define  STATUS_SET_IP       5
#define  STATUS_SET_SUBNET   6
#define  STATUS_SET_GATEWAY  7
#define  STATUS_MEM_SAVE     8
#define  STATUS_MEM_RECALL   9
#define  STATUS_RECALLED     10
#define  STATUS_ILLEGAL      11

#define  AUDIO_BUFLEN    256


#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                              IPPROC                                      */
/*                IP Address Key Processing Functions                       */
/*                       VoIP Telephone Project                             */
/*                              EE/CS 52                                    */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the key processing functions for IP addresses for the
   VoIP Telephone Project.  These functions are called by the main loop of
   the system.  The functions included are:
      add_IPDigit   - add a decimal digit to the input IP address
      clear_IPAddr  - clear the input IP address
      del_IPDigit   - delete the last input IP address decimal digit
      set_gateway   - set the gateway address
      set_IP        - set the IP address
      set_subnet    - set the subnet mask
      start_IPEntry - start entering an IP address

   The local functions included are:
      clr_IP_address - clears the IP address
```

The locally global (shared) variable definitions included are:

IP_address - the current value of the IP address

IP_digit_cnt - number of decimal digits input to the IP address

Revision History

6/3/06 Glen George Initial revision.

6/6/06 Glen George Changed start_IPEntry to output the current ethernet address (all 0's).

6/7/06 Glen George Changed clear_IPAddr to handle clearing a recalled address correctly.

6/7/06 Glen George Updated del_IPDigit to work better with a recalled address (i.e. don't clear it out).

6/8/06 Glen George Added code to actually set the IP address, gateway, and subnet mask.

5/26/08 Glen George Only do actual IP calls if NO_LWIP is not defined.

6/13/08 Glen George Fixed a number of compiler errors.

3/10/11 Glen George Updated code so any time the ethernet interface is changed (changing the gateway for example) the connection is restarted.

*/

```
/* library include files */
#ifndef  NO_LWIP                  /* don't include files if not using LWIP */
  #include  "lwip/netif.h"
  #include  "lwip/ip_addr.h"
#endif


/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "keyproc.h"
#include  "error.h"
#include  "callutil.h"
#include  "tcpconn.h"




/* local function declarations */
static void  clr_IP_address(void);
```

```
/* locally global (shared) variables */
static unsigned long int  IP_address;  /* the current IP address */
static int                IP_digit_cnt;          /* the number of IP address digits input */




/*
  start_IPEntry

  Description:      This function handles the start of entering an IP
                    address.  It first calls a function to clear the IP
                    address, then sets the new system status based on the
                    passed key value and returns that status.

  Arguments:        cur_status (enum status) - the current system status.
                    key_value (int)          - value of the input key.
  Return Value:     (enum status) - the new status.

  Input:            None.
  Output:           None.

  Error Handling:   If there is an unexpected passed key value, the error
                    handler is called with the error UNKNOWN_KEYCODE_INIT.
```

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:           Glen George
   Last Modified:    June 6, 2006

*/

enum status  start_IPEntry(enum status cur_status, int key_value)
{
    /* variables */
    enum status  new_status;              /* new status to return */



    /* clear the IP address */
    clr_IP_address();



    /* figure out the new system status */
    switch (key_value)  {
```

```
case  KEYCODE_OFFHOOK:        /* Off-Hook key was seen */
                              /* new status is off-hook */
                              new_status = STAT_OFFHOOK;
                              break;

case  KEYCODE_SETIP:    /* <Set IP> key was seen */
                              /* new status is setting IP address */
                              new_status = STAT_SETIP;
                              break;

case  KEYCODE_SETSUBNET:/* <Set Subnet> key was seen */
                              /* new status is setting subnet mask */
                              new_status = STAT_SETSUBNET;
                              break;

case  KEYCODE_SET_GW:         /* <Set Gateway> key was seen */
                              /* new status is setting gateway address */
                              new_status = STAT_SET_GW;
                              break;

default:            /* some other key was seen */
                              /* generate an error and leave status unchanged */
                              process_error(UNKNOWN_KEYCODE_INIT);
```

```
                    new_status = cur_status;
                    break;
    }


    /* if got a valid key (changed status) then output the current IP address */
    if (new_status != cur_status)
        display_IP(IP_address);


    /* and return the new status */
    return  new_status;

}




/*
   clear_IPAddr

   Description:      This function handles the clear key when an IP address is
                     being input.  It clears the IP address, displays the now
                     cleared IP address, and returns with the system status it
```

was passed, unless it was the recalled state. If the
current status was the recall state then status reverts
to the idle state.

Arguments: cur_status (enum status) - the current system status.
key_value (int) - value of the key that was
input (ignored).
Return Value: (enum status) - the new status, same as current status or
the idle state if it was in the recall state.

Input: None.
Output: None.

Error Handling: None.

Algorithms: None.
Data Structures: None.

Shared Variables: None.

Author: Glen George
Last Modified: June 7, 2006

*/

```
enum status  clear_IPAddr(enum status cur_status, int key_value)
{
  /* variables */
    /* none */




  /* clear the IP address and digit count */
  clr_IP_address();


  /* update the calling IP address */
  set_calling_IP(IP_address);


  /* display the new IP address */
  display_IP(IP_address);



  /* was there a recalled IP address on the screen */
  if (cur_status == STAT_RECALLED)
      /* if so, need to switch to idle state since don't have that address anymore */
          cur_status = STAT_IDLE;
```

```
    /* and return the possibly new status */
    return  cur_status;

}




/*
   add_IPDigit

   Description:      This function handles a digit key when an IP address is
                  being input.  It adds the passed key value to the current
                        value of the IP address.  If there is an overflow (more
                        than 32-bits or a byte value greater than 255) the error
                  handler is called and the key is ignored.  The function
                        returns with the system status it was passed.

   Arguments:        cur_status (enum status) - the current system status.
                  key_value (int)         - value of the input key.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.
```

```
   Error Handling:   If the input causes the IP address to have more than 32
                 bits or if a byte in the address would be greater than
                       256, the error handler is called with the error
                       IP_ADDRESS_OVERFLOW and the key is ignored.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: IP_address (changed)   - updated to new address based on
                               the passed key value.
                 IP_digit_cnt (changed) - updated to reflect which decimal
                                     digit of the IP address will be
                               input next.

   Author:           Glen George
   Last Modified:    June 3, 2006

*/

enum status  add_IPDigit(enum status cur_status, int key_value)
{
    /* variables */
    unsigned long int  current_byte;     /* current byte of the IP address */
```

```
/* check if there is room for the digit */
if (IP_digit_cnt < NUM_IP_DIGITS)  {

   /* have room for the digit - add it in to the current byte */
       /* get the current byte's value so far */
       current_byte = IP_address & (0xFF000000L >> (8 * (IP_digit_cnt / 3)));
       /* shift it down to an 8-bit value */
       current_byte >>= (8 * (3 - (IP_digit_cnt / 3)));

       /* add in the new digit (if possible) */
       if (((10 * current_byte) + key_value) < 256)  {

          /* it fits - add it in */
          current_byte = (10 * current_byte) + key_value;

          /* compute the new IP address value */
          /* mask off old bits for the current byte */
          IP_address &= ~(0xFF000000L >> (8 * (IP_digit_cnt / 3)));
          /* bring in the new value for the byte */
          IP_address  |= (current_byte << (8 * (3 - (IP_digit_cnt / 3))));
```

```
            /* update the digit count */
            IP_digit_cnt++;
          }
          else  {

            /* doesn't fit - call the error handler */
            process_error(IP_ADDRESS_OVERFLOW);
        }
      }
      else  {

        /* too many digits in the IP address - call the error handler */
          process_error(IP_ADDRESS_OVERFLOW);
      }


      /* update the calling IP address */
      set_calling_IP(IP_address);

      /* display the new IP address */
      display_IP(IP_address);


      /* all done adding the digit, return the passed status */
```

```
    return  cur_status;

}




/*
   del_IPDigit


   Description:      This function handles a backspace key when an IP address
                  is being input.  It removes the last input decimal digit
                  from the current value of the IP address.  If there are
                  no digits in the IP address, the error handler is called
                  and the key is ignored.  The function returns with the
                  system status it was passed.


   Arguments:        cur_status (enum status) - the current system status.
                  key_value (int)         - value of the key that was
                                          input (ignored).
   Return Value:     (enum status) - the new status (same as current status).


   Input:            None.
   Output:           None.
```

```
   Error Handling:   If there are no digits in the IP address to delete, the
                     key is ignored and the error handler is called with the
                     error IP_ADDRESS_UNDERFLOW.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: IP_address (changed)   - updated to new address by
                                  deleting the lowest decimal
                                                 digit.
                 IP_digit_cnt (changed) - updated to reflect which decimal
                                          digit of the IP address will be
                                  input next.

   Author:           Glen George
   Last Modified:    June 7, 2006

*/

enum status  del_IPDigit(enum status cur_status, int key_value)
{
    /* variables */
    unsigned long int  current_byte;     /* last input byte of the IP address */
```

```
/* are there any digits to delete? */
if (IP_digit_cnt > 0)  {

   /* looks like there should be a digit to delete */
   /* get the value of the last input byte */
   current_byte = IP_address & (0xFF000000L >> (8 * ((IP_digit_cnt - 1) / 3)));
       /* shift it down to an 8-bit value */
       current_byte >>= (8 * (3 - ((IP_digit_cnt - 1) / 3)));

       /* remove the last digit - just divide by 10 */
       current_byte /= 10;

       /* compute the new IP address value */
       /* mask off old bits for the current byte */
       IP_address &= ~(0xFF000000L >> (8 * ((IP_digit_cnt - 1) / 3)));
       /* bring in the new value for the byte */
       IP_address  |= (current_byte << (8 * (3 - ((IP_digit_cnt - 1) / 3))));

       /* update the digit count - there's one less digit now */
       IP_digit_cnt--;
```

```
        /* update the calling IP address */
        set_calling_IP(IP_address);


        /* display the new IP address */
        display_IP(IP_address);
    }
    else  {


        /* no digits to delete - call the error handler */
            process_error(IP_ADDRESS_UNDERFLOW);
    }



    /* all done deleting the digit, return the passed status */
    return  cur_status;

}





/*
  set_IP
```

Description: This function handles setting the IP address to the address entered thus far. The status is always returned as idle.

Operation: The TCP/IP stack functions are setup with the current value of the IP address and then the connection is restarted.

Arguments: cur_status (enum status) - the current system status (ignored).

key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_IDLE).

Input: None.

Output: None.

Error Handling: If there is an error getting the named interface (given by ETHER_INTF), the error handler is called with the error UNKNOWN_ETHER_NAME.

Algorithms: None.

Data Structures: None.

```
   Shared Variables: None.

   Author:          Glen George
   Last Modified:   March 10, 2011

*/

enum status  set_IP(enum status cur_status, int key_value)
{
    /* variables */
#ifndef  NO_LWIP                  /* empty function if there is no LWIP code */
    struct netif   *net_intf;                  /* network interface to access */
    struct ip_addr   ip;                       /* the new IP address */



    /* create the IP address in network order */
    ip.addr = htonl(get_calling_IP());


    /* now get the network interface for our device */
    net_intf = netif_find(ETHER_INTF);

    /* check if got an interface */
```

```
    if (net_intf != NULL)
        /* got the interface, set the IP address */
            netif_set_ipaddr(net_intf, &ip);
    else
        /* no interface, report the error */
            process_error(UNKNOWN_ETHER_NAME);
#endif


    /* need to restart the connection */
    tcp_connection_restart();


    /* done so return with idle as the status */
    return  STAT_IDLE;

}




/*
   set_gateway
```

Description: This function handles setting the gateway address to the address entered thus far. The status is always returned as idle.

Operation: The function calls the TCP/IP stack functions to set the gateway to the just entered IP address. Then the TCP connection is restarted.

Arguments: cur_status (enum status) - the current system status (ignored).

key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_IDLE).

Input: None.

Output: None.

Error Handling: If there is an error getting the named interface (given by ETHER_INTF), the error handler is called with the error UNKNOWN_ETHER_NAME.

Algorithms: None.

Data Structures: None.

```
   Shared Variables: None.


   Author:          Glen George
   Last Modified:    March 10, 2011

*/

enum status  set_gateway(enum status cur_status, int key_value)
{
    /* variables */
#ifndef  NO_LWIP                   /* empty function if there is no LWIP code */
    struct netif    *net_intf;                   /* network interface to access */
    struct ip_addr   gw;            /* the new gateway address */




    /* create the gateway address in network order */
    gw.addr = htonl(get_calling_IP());



    /* now get the network interface for our device */
    net_intf = netif_find(ETHER_INTF);


    /* check if got an interface */
```

```
    if (net_intf != NULL)
        /* got the interface, set the gateway address */
            netif_set_gw(net_intf, &gw);
    else
        /* no interface, report the error */
            process_error(UNKNOWN_ETHER_NAME);
#endif


    /* need to restart the connection */
    tcp_connection_restart();


    /* done so return with idle as the status */
    return  STAT_IDLE;

}




/*
   set_subnet
```

Description: This function handles setting the subnet mask to the address entered thus far. The status is always returned as idle.

Operation: The function calls the TCP/IP stack functions to set the subnet mask to the just entered IP address. Then the TCP connection is restarted.

Arguments: cur_status (enum status) - the current system status (ignored).

key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (always STAT_IDLE).

Input: None.

Output: None.

Error Handling: If there is an error getting the named interface (given by ETHER_INTF), the error handler is called with the error UNKNOWN_ETHER_NAME.

Algorithms: None.

Data Structures: None.

```
   Shared Variables: None.

   Author:          Glen George
   Last Modified:    March 10, 2011

*/

enum status  set_subnet(enum status cur_status, int key_value)
{
    /* variables */
#ifndef  NO_LWIP                  /* empty function if there is no LWIP code */
    struct netif    *net_intf;                 /* network interface to access */
    struct ip_addr   mask;                     /* the new subnet mask */



    /* create the subnet mask in network order */
    mask.addr = htonl(get_calling_IP());



    /* now get the network interface for our device */
    net_intf = netif_find(ETHER_INTF);

    /* check if got an interface */
```

```
    if (net_intf != NULL)
        /* got the interface, set the subnet mask */
            netif_set_netmask(net_intf, &mask);
    else
        /* no interface, report the error */
            process_error(UNKNOWN_ETHER_NAME);
#endif


    /* need to restart the connection */
    tcp_connection_restart();


    /* done so return with idle as the status */
    return  STAT_IDLE;

}




/* local helper functions */
```

```
/*
   clr_IP_address

   Description:      This function clears the IP address and digit count,

   Arguments:        None.
   Return Value:     None.

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: IP_address (changed)   - reset to 0.
                     IP_digit_cnt (changed) - reset to 0.

   Author:           Glen George
   Last Modified:    June 3, 2006

*/
```

```
static void  clr_IP_address()
{
   /* variables */
     /* none */



   /* clear the IP address and digit count */
   IP_address = 0;
   IP_digit_cnt = 0;



   /* and return */
   return;

}
```

```
/****************************************************************************/
/*                                                                          */
/*                              KEYPROC                                     */
/*                  Miscellaneous Key Processing Functions                  */
/*                         VoIP Telephone Project                           */
/*                              EE/CS  52                                   */
/*                                                                          */
/****************************************************************************/


/*
  This file contains the general key processing functions for the VoIP
  Telephone Project.  These functions are called by the main loop of the
  system.  The functions included are:
    no_action   - nothing to do
    reset_input - reset the input state

  The local functions included are:
    none

  The global variable definitions included are:
    none


  Revision History
```

```
     6/3/06   Glen George       Initial revision.
*/




/* library include files */
  /* none */


/* local include files */
#include  "voipdefs.h"
#include  "keyproc.h"
#include  "callutil.h"




/*
   no_action


   Description:      This function handles a key when there is nothing to be
                     done.  It just returns.  It is needed to fill in the key
                     processing table.


   Arguments:        cur_status (enum status) - the current system status.
```

```
                key_value (int)          - value of the key that was
                                           input (ignored).
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   None.

   Algorithms:       None.
   Data Structures:  None.

   Global Variables: None.

   Author:           Glen George
   Last Modified:    June 3, 2006

*/

enum status  no_action(enum status cur_status, int key_value)
{
    /* variables */
      /* none */
```

```
    /* return the current status */

    return  cur_status;

}




/*
   reset_input

   Description:      This function handles keys which cause the input state to
                 be reset.  The function resets the calling IP address and
                 returns idle as the new state.

   Arguments:       cur_status (enum status) - the current system status.
                 key_value (int)         - value of the key that was
                                          input (ignored).
   Return Value:     (enum status) - the new status (always STAT_IDLE).

   Input:           None.
   Output:          None.
```

```
   Error Handling:   None.


   Algorithms:       None.
   Data Structures:  None.


   Global Variables: None.


   Author:           Glen George
   Last Modified:    June 3, 2006

*/


enum status  reset_input(enum status cur_status, int key_value)
{
    /* variables */
      /* none */




    /* reset the calling IP address */
    set_calling_IP(0L);
```

```
    /* now return with the idle state */

    return  STAT_IDLE;


}
```

```
/****************************************************************************/
/*                                              */
/*                    KEYPROC.H                  */
/*              Key Processing Functions           */
/*                   Include File                */
/*              VoIP Telphone Project            */
/*                   EE/CS  52                  */
/*                                              */
/****************************************************************************/


/*
   This file contains the constants and function prototypes for the key
   processing functions defined in ipproc.c, callproc.c, memproc.c, and
   keyproc.c.


   Revision History:
      6/3/06   Glen George      Initial revision.
*/



#ifndef  I__KEYPROC_H__
    #define  I__KEYPROC_H__
```

```
/* library include files */
  /* none */

/* local include files */
#include  "voipdefs.h"

/* constants */
   /* none */

/* structures, unions, and typedefs */
   /* none */

/* function declarations */
```

```
enum status  no_action(enum status, int);    /* nothing to do */
enum status  reset_input(enum status, int);  /* reset input state */

enum status  start_IPEntry(enum status, int);          /* start entering IP address */
enum status  clear_IPAddr(enum status, int);           /* clear the IP address */
enum status  add_IPDigit(enum status, int);  /* add a digit to the IP address */
enum status  del_IPDigit(enum status, int);   /* delete a digit from the IP address */
enum status  set_IP(enum status, int);                 /* set the IP address */
enum status  set_gateway(enum status, int);            /* set the gateway address */
enum status  set_subnet(enum status, int);   /* set the subnet mask */

enum status  start_memLoc(enum status, int);        /* starting entering a memory location */
enum status  clear_memLoc(enum status, int);        /* clear the memory address */
enum status  add_memDigit(enum status, int);        /* add a digit to the memory address */
enum status  del_memDigit(enum status, int);        /* delete a digit from the memory address
*/
enum status  recall_mem(enum status, int); /* recall an address from memory */
enum status  save_mem(enum status, int);   /* save an address to memory */

enum status  do_answer(enum status, int);  /* answer an incoming call */
enum status  do_call(enum status, int);                /* start an outgoing call */
enum status  end_call(enum status, int);      /* end a call */
```

```
#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                              MAINLOOP                                    */
/*                          Main Program Loop                               */
/*                       VoIP Telephone Project                             */
/*                              EE/CS 52                                    */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the main processing loop (background) for the VoIP
   Telephone Project.  The only global function included is:
      main - background processing loop

   The local functions included are:
      key_lookup - get a key and look up its keycode

   The locally global variable definitions included are:
      none


   Revision History
      6/3/06   Glen George      Initial revision.
      6/7/06   Glen George      Added initialization of the rx/tx buffers.
```

```
     6/7/06   Glen George      Added displaying incoming call IP address.
     6/7/06   Glen George      Improved processing of memory recall.
     6/9/09   Glen George      Added support for LWIP.
     3/4/11   Glen George      Added static to key_lookup declaration to
                               avoid compiler errors/warnings.
     3/8/11   Glen George      Changed TCP timer to run at a fixed rate
                               instead of a fixed interval.
*/



/* library include files */
#ifndef  NO_LWIP                /* don't include files if not using LWIP */
  #include  "lwip/tcp.h"
#endif

/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "keyproc.h"
#include  "callutil.h"
#include  "buffers.h"
#include  "tcpconn.h"
```

```
#ifndef  NO_LWIP                    /* don't include files if not using LWIP */
  #include  "ethernet.h"
#endif




/* local function declarations */
static enum keycode  key_lookup(int);          /* translate key values into keycodes */




/*
   main

   Description:      This procedure is the main program loop for the VoIP
                     Telephone Recorder.

   Operation:        The function loops getting keys from the keypad,
                     processing those keys as is appropriate.  It also handles
                     updating the display and setting up the buffers for audio
                     input and output.  The loop is table driven.
```

```
   Arguments:       None.
   Return Value:    (int) - return code, always 0 (never returns).

   Input:           Keys from the keypad.
   Output:          Status information to the display.

   Error Handling:  Invalid input is ignored.

   Algorithms:      The function is table-driven.  The processing routines
                    for each input are given in tables which are selected
                    based on the context (state) the program is operating in.
   Data Structures: None.

   Global Variables: None.

   Author:          Glen George
   Last Modified:   March 8, 2009

*/

int  main()
{
    /* variables */
    enum keycode  key;                  /* an input key */
```

```
int        timeout;              /* ethernet timing information */

enum status   cur_status = STAT_IDLE;   /* current program status */
enum status   prev_status = STAT_IDLE;  /* previous program status */

/* array of status type translations (from enum status to #defines) */
/* note: the array must match the enum definition order exactly */
const static unsigned int  xlat_stat[] =
   {  STATUS_IDLE,             /* system idle */
        STATUS_OFFHOOK,  /* phone is off hook */
        STATUS_RINGING,   /* incoming call */
        STATUS_CONNECTING,       /* attempting to connect */
        STATUS_CONNECTED,        /* connected to remote phone */
        STATUS_SET_IP,       /* setting the IP address */
        STATUS_SET_SUBNET,        /* setting the subnet address */
        STATUS_SET_GATEWAY,     /* setting the gateway address */
        STATUS_MEM_SAVE,          /* saving an address to memory */
        STATUS_MEM_RECALL,       /* recalling an address from memory */
        STATUS_RECALLED   /* just recalled an address from memory */
      };

/* array of key type translations (from enum keycode to enum keytype) */
/* note: the array must match the enum definition order exactly */
```

```
const static enum keytype  key_type[] =
  {  KEYTYPE_DIGIT,       /* <0>            */
     KEYTYPE_DIGIT,       /* <1>            */
        KEYTYPE_DIGIT,       /* <2>            */
        KEYTYPE_DIGIT,       /* <3>            */
        KEYTYPE_DIGIT,       /* <4>            */
        KEYTYPE_DIGIT,       /* <5>            */
        KEYTYPE_DIGIT,       /* <6>            */
        KEYTYPE_DIGIT,       /* <7>            */
        KEYTYPE_DIGIT,       /* <8>            */
        KEYTYPE_DIGIT,       /* <9>            */
        KEYTYPE_ESC,         /* <ESC>          */
        KEYTYPE_BS,          /* <Backspace>     */
        KEYTYPE_SEND,        /* <Send>          */
        KEYTYPE_OFFHOOK,     /* Off-Hook        */
        KEYTYPE_ONHOOK,      /* On-Hook         */
        KEYTYPE_SETIP,       /* <Set IP>        */
        KEYTYPE_SETSUBNET,   /* <Set Subnet>    */
        KEYTYPE_SET_GW,      /* <Set Gateway>   */
        KEYTYPE_MEMSAVE,     /* <Memory Save>   */
        KEYTYPE_MEMRECALL,   /* <Memory Recall> */
        KEYTYPE_UNKNOWN      /* other keys      */
     };
```

```
/* array of key value translations (from enum keycode to int) */
/* note: the array must match the enum definition order exactly */
const static int  key_value[] =
  {  0,              /* <0>           */
     1,              /* <1>           */
        2,              /* <2>           */
        3,              /* <3>           */
        4,              /* <4>           */
        5,              /* <5>           */
        6,              /* <6>           */
        7,              /* <7>           */
        8,              /* <8>           */
        9,              /* <9>           */
        KEYCODE_ESC,       /* <ESC>          */
        KEYCODE_BS,        /* <Backspace>     */
        KEYCODE_SEND,      /* <Send>         */
        KEYCODE_OFFHOOK,    /* Off-Hook key    */
        KEYCODE_ONHOOK,     /* Off-Hook key    */
        KEYCODE_SETIP,      /* <Set IP>       */
        KEYCODE_SETSUBNET,  /* <Set Subnet>    */
        KEYCODE_SET_GW,     /* <Set Gateway>   */
        KEYCODE_MEMSAVE,    /* <Memory Save>   */
        KEYCODE_MEMRECALL,  /* <Memory Recall> */
        0               /* other keys      */
```

```
        };

  /* whether or not numeric keys have priority */
  static const int  numeric_priority[] =
        {
     /*              Current System Status              */
     /* idle     off-hook    ringing     connecting  connected   */
        FALSE,    TRUE,       FALSE,       FALSE,      FALSE,
     /* set IP    set subnet  set gateway  mem save    mem recall  */
        TRUE,     TRUE,       TRUE,        TRUE,       TRUE,
     /* recalled                                   */
        FALSE
      };

  /* key processing functions (one for each system status type and key) */
  static enum status  (* const process_key[NUM_KEYTYPES][NUM_STATUS])(enum status, int)
=
    /*                    Current System Status                              */
    /* idle        off-hook      ringing      connecting    connected          key        */
    /* set IP       set subnet      set gateway    mem save      mem recall          type       */
    /* recalled                                                               */
   { {  no_action,    add_IPDigit,    no_action,     no_action,     no_action,       /* Digit        */
        add_IPDigit,   add_IPDigit,    add_IPDigit,   add_memDigit,  add_memDigit,
        no_action                                                 },
```

```
{ no_action,    clear_IPAddr,  no_action,    no_action,    no_action,      /* Escape       */
  clear_IPAddr,  clear_IPAddr,   clear_IPAddr,  clear_memLoc,  clear_memLoc,
  clear_IPAddr                                             },
{ no_action,    del_IPDigit,   no_action,    no_action,    no_action,      /* Backspace    */
  del_IPDigit,   del_IPDigit,    del_IPDigit,   del_memDigit,  del_memDigit,
  no_action                                                },
{ no_action,    do_call,       do_answer,    end_call,     end_call,       /* Send/Enter   */
  set_IP,       set_subnet,    set_gateway,  save_mem,     recall_mem,
  do_call                                                  },
{ start_IPEntry, no_action,    do_answer,    no_action,    no_action,      /* Off-Hook     */
  no_action,    no_action,     no_action,    no_action,    no_action,
  start_IPEntry                                            },
{ no_action,    end_call,      no_action,    end_call,     end_call,       /* On-Hook      */
  no_action,    no_action,     no_action,    no_action,    no_action,
  no_action                                                },
{ start_IPEntry, no_action,    start_IPEntry, no_action,   no_action,      /* Set IP       */
  reset_input,  no_action,     no_action,    no_action,    no_action,
  set_IP                                                   },
{ start_IPEntry, no_action,    start_IPEntry, no_action,   no_action,      /* Set Subnet   */
  no_action,    reset_input,   no_action,    no_action,    no_action,
  set_subnet                                               },
{ start_IPEntry, no_action,    start_IPEntry, no_action,   no_action,      /* Set Gateway
*/
  no_action,    no_action,     reset_input,  no_action,    no_action,
```

```
        set_gateway                                    },
      { start_memLoc,  start_memLoc,   start_memLoc,  start_memLoc,  start_memLoc,    /*
Memory Save   */
        no_action,    no_action,     no_action,    reset_input,   no_action,
        start_memLoc                                   },
      { start_memLoc,  start_memLoc,   start_memLoc,  no_action,    no_action,      /* Memory
Recall */
        no_action,    no_action,     no_action,    no_action,    reset_input,
        start_memLoc                                   },
      { no_action,    no_action,     no_action,    no_action,    no_action,      /* unknown key   */
        no_action,    no_action,     no_action,    no_action,    no_action,
        no_action                                      } };



    /* first initialize everything */
    init_buffers();                  /* initialize the rx/tx buffers */
    init_memory();                   /* initialize saved IPs */

      /* initialize the lwIP code if it is being used */
    #ifndef  NO_LWIP
      init_networking();
    #endif
```

```
tcp_connection_init();        /* initialize the TCP connection */

/* reset timing - reset elapsed timer and no time yet */
elapsed_time();
timeout = 0;


/* display the initial status */
display_status(xlat_stat[cur_status]);


/* infinite loop processing input */
while(TRUE)  {

   /* handle networking status changes */
      if ((cur_status == STAT_IDLE) && incoming_call())  {
         /* have an incoming call - change to ringing state */
         cur_status = STAT_RINGING;
         /* and display the incoming call IP Address */
         display_IP(get_calling_IP());
      }
      else if ((cur_status == STAT_RINGING) && !incoming_call())  {
         /* no more incoming call - stop ringing */
         cur_status = STAT_IDLE;
```

```
}
else if ((cur_status == STAT_CONNECTING) && call_connected())  {
   /* have connected to the remote phone */
   cur_status = STAT_CONNECTED;
   /* need to start the call */
   start_call();
}
else if (cur_status == STAT_CONNECTED)  {
   /* we are connected, continue processing the call */
   process_call();
}



/* now check for keypad input */
if (key_available())  {

   /* have keypad input - get the key */
   key = key_lookup(numeric_priority[cur_status]);

   /* execute processing routine for that key */
   cur_status = process_key[key_type[key]][cur_status](cur_status, key_value[key]);
}
```

```
/* finally, if the status has changed - display the new status */
if (cur_status != prev_status)  {

   /* status has changed - update the status display */
       display_status(xlat_stat[cur_status]);
}

/* always remember the current status for next loop iteration */
prev_status = cur_status;



    /* see if we need to generate a TCP timer event */
    timeout += elapsed_time();
    /* only worry about generating timer event if using LwIP */
#ifndef  NO_LWIP
       if (timeout >= TCP_TIMEOUT)  {

      /* have a timeout - tell the TCP code */
      tcp_tmr();

          /* have taken care of a timeout interval now */
          timeout -= TCP_TIMEOUT;
   }
#endif
```

```
        /* check if need to process a packet */
        if (ether_rx_available())  {

            /* there is ethernet data available - try to process it */
            /*    only process it if using lwIP code */
        #ifndef  NO_LWIP
            ethernetif_input();
        #endif
      }
   }

   /* done with main (never should get here), return 0 */
   return  0;

}

/*
  key_lookup
```

Description: This function gets a key from the keypad and translates the raw keycode to an enumerated keycode for the main loop. The keycode precedence is a function of the passed argument. If the argument is true, numeric keys have precedence. This allows the numeric keys to have more than one meaning.

Operation: The function calls getkey and then converts the returned raw keycode to an enumerated keycode for the main loop by using one of two translation tables. Which table to use is determined by the passed argument.

Arguments: numeric (int) - true to indicate numeric keys have precedence.

Return Value: (enum keycode) - type of the key input on keypad.

Input: Keys from the keypad.

Output: None.

Error Handling: Invalid keys are returned as KEYCODE_ILLEGAL.

Algorithms: The function uses arrays to lookup the key types.

Data Structures: Arrays of key types versus key codes.

```
   Global Variables: None.


   Author:          Glen George
   Last Modified:   June 3, 2006

*/


static  enum keycode  key_lookup(int numeric)
{
    /* variables */

    const static enum keycode  nnkeycodes[] = /* array of keycodes for non-numeric precedence
*/
        {                               /* order must match keys array exactly */
            KEYCODE_ESC,        /* <ESC>          */
            KEYCODE_BS,         /* <Backspace>    */
            KEYCODE_SEND,       /* <Send>         */
            KEYCODE_OFFHOOK,    /* Off-Hook       */
            KEYCODE_ONHOOK,     /* On-Hook        */
            KEYCODE_SETIP,      /* <Set IP>       */
            KEYCODE_SETSUBNET,  /* <Set Subnet>   */
            KEYCODE_SET_GW,     /* <Set Gateway>  */
            KEYCODE_MEMSAVE,    /* <Memory Save>  */
```

```
        KEYCODE_MEMRECALL,  /* <Memory Recall> */
    KEYCODE_0,         /* <0>            */
        KEYCODE_1,         /* <1>            */
        KEYCODE_2,         /* <2>            */
        KEYCODE_3,         /* <3>            */
        KEYCODE_4,         /* <4>            */
        KEYCODE_5,         /* <5>            */
        KEYCODE_6,         /* <6>            */
        KEYCODE_7,         /* <7>            */
        KEYCODE_8,         /* <8>            */
        KEYCODE_9,         /* <9>            */
        KEYCODE_ILLEGAL     /* entry needed for illegal codes */
  };

const static enum keycode  nkeycodes[] = /* array of keycodes for numeric precedence */
  {                               /* order must match nkeys array exactly */
    KEYCODE_0,         /* <0>            */
        KEYCODE_1,         /* <1>            */
        KEYCODE_2,         /* <2>            */
        KEYCODE_3,         /* <3>            */
        KEYCODE_4,         /* <4>            */
        KEYCODE_5,         /* <5>            */
        KEYCODE_6,         /* <6>            */
        KEYCODE_7,         /* <7>            */
```

```
        KEYCODE_8,        /* <8>           */
        KEYCODE_9,        /* <9>           */
        KEYCODE_ESC,      /* <ESC>         */
        KEYCODE_BS,       /* <Backspace>   */
        KEYCODE_SEND,     /* <Send>        */
        KEYCODE_OFFHOOK,  /* Off-Hook      */
        KEYCODE_ONHOOK,   /* On-Hook       */
        KEYCODE_SETIP,    /* <Set IP>      */
        KEYCODE_SETSUBNET, /* <Set Subnet>  */
        KEYCODE_SET_GW,   /* <Set Gateway>  */
        KEYCODE_MEMSAVE,  /* <Memory Save>  */
        KEYCODE_MEMRECALL, /* <Memory Recall> */
        KEYCODE_ILLEGAL   /* entry needed for illegal codes */
    };

const enum keycode  *keycodes;   /* pointer to appropriate keycode array */

const static int  nnkeys[] = /* array of key values for non-numeric precedence */
    {                        /* order must match keycodes array exactly */
      KEY_ESC,               /* <ESC>         */
      KEY_BACKSPACE,         /* <Backspace>   */
      KEY_SEND,              /* <Send>        */
      KEY_OFFHOOK,           /* Off-Hook      */
      KEY_ONHOOK,            /* On-Hook       */
```

```
    KEY_SET_IP,          /* <Set IP>       */
    KEY_SET_SUBNET,      /* <Set Subnet>    */
    KEY_SET_GATEWAY,     /* <Set Gateway>   */
    KEY_MEM_SAVE,        /* <Memory Save>   */
    KEY_MEM_RECALL,      /* <Memory Recall> */
    KEY_0,       /* <0>          */
    KEY_1,       /* <1>          */
    KEY_2,       /* <2>          */
    KEY_3,       /* <3>          */
    KEY_4,       /* <4>          */
    KEY_5,       /* <5>          */
    KEY_6,       /* <6>          */
    KEY_7,       /* <7>          */
    KEY_8,       /* <8>          */
    KEY_9        /* <9>          */
  };

const static int  nkeys[] =  /* array of key values for numeric precedence */
  {                          /* order must match keycodes array exactly */
    KEY_0,       /* <0>          */
    KEY_1,       /* <1>          */
    KEY_2,       /* <2>          */
    KEY_3,       /* <3>          */
    KEY_4,       /* <4>          */
```

```
    KEY_5,        /* <5>          */
    KEY_6,        /* <6>          */
    KEY_7,        /* <7>          */
    KEY_8,        /* <8>          */
    KEY_9,        /* <9>          */
    KEY_ESC,              /* <ESC>        */
    KEY_BACKSPACE,     /* <Backspace>    */
    KEY_SEND,             /* <Send>        */
    KEY_OFFHOOK,       /* Off-Hook       */
    KEY_ONHOOK,        /* On-Hook        */
    KEY_SET_IP,           /* <Set IP>      */
    KEY_SET_SUBNET,     /* <Set Subnet>    */
    KEY_SET_GATEWAY,    /* <Set Gateway>   */
    KEY_MEM_SAVE,       /* <Memory Save>   */
    KEY_MEM_RECALL      /* <Memory Recall> */
   };

const int  *keys;      /* pointer to appropriate key array */



int  key;                  /* an input key */


int  i;           /* general loop index */
```

```
/* figure out which array to use */
if (numeric)  {
   /* should be using the numeric precedence arrays */
        keycodes = nkeycodes;
        keys = nkeys;
}
else  {
   /* should be using the non-numeric precedence arrays */
        keycodes = nnkeycodes;
        keys = nnkeys;
}


/* get a key */
key = getkey();


/* lookup key in the appropriate keys array */
/* note that nnkeys[] and nkeys[] should be the same size so can */
/*    use either in the comparison */
for (i = 0; ((i < (sizeof(nkeys)/sizeof(int))) && (key != keys[i])); i++);
```

```
    /* return the appropriate key type */
    return  keycodes[i];

}
```

```
/****************************************************************************/
/*                                                   */
/*                        MEMPROC                          */
/*                 Memory Processing Functions                    */
/*                   VoIP Telephone Project                      */
/*                        EE/CS  52                          */
/*                                                   */
/****************************************************************************/


/*
  This file contains the key processing and general functions for memory
  operations for the VoIP Telephone Project.  These functions are called by
  the main loop of the system.  The functions included are:
    add_memDigit - add a digit to the memory location number
    clear_memLoc - clear the memory location number
    del_memDigit - delete a digit from the memory location number
    init_memory  - initialize the memory system
    recall_mem   - recall the IP address from the current memory location
    save_mem     - save the current IP address to current memory location
    start_memLoc - start entering a memory location number

  The local functions included are:
    none
```

```
   The global variable definitions included are:
      memloc     - current memory location
      saved_IPs  - array of saved IP memory addresses
      old_status - status of system before a save or recall operation


   Revision History
      6/3/06   Glen George       Initial revision.
      6/7/06   Glen George       Changed start_memLoc, recall_mem, and
                                    save_mem to remember the state the phone was
                                    in before the IP address was saved or
                                    recalled and restore that state if necessary.
*/




/* library include files */
  /* none */


/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "keyproc.h"
#include  "callutil.h"
#include  "error.h"
```

```
/* locally global variables */
static unsigned int       memloc;                              /* the current memory location */
static unsigned long int  saved_IPs[MEMORY_SIZE + 1];       /* array of saved IP addresses */

static enum status        old_status;                          /* status before a memory operation */




/*
  init_memory

  Description:      This function initializes the memory system.  It first
                    checks if location 0 has the "magic" value in it.  If so
                    the system is already initialized and there is nothing
                    to do.  If the "magic" value is not at location 0 then it
                            is written there and the other saved memory values are
                            set to 0.  This is done so the memory is maintained
                            between resets, as long as power doesn't fail.
```

```
   Arguments:       None.
   Return Value:    None.

   Input:           None.
   Output:          None.

   Error Handling:  None.

   Algorithms:      None.
   Data Structures: None.

   Shared Variables: saved_IPs (changed) - initialized to all 0's.

   Author:          Glen George
   Last Modified:   June 3, 2006

*/

void  init_memory()
{
    /* variables */
    int  i;             /* general loop index */
```

```
    /* check if memory is already initialized */
    if (saved_IPs[0] != MAGIC_IP)  {

        /* memory is not initialized - need to initialize it */
            /* first save the "magic" value */
            saved_IPs[0] = MAGIC_IP;

            /* now set all other IPs to 0 */
            for (i = 1; i <= MEMORY_SIZE; i++)
                saved_IPs[i] = 0;
    }


    /* done with memory initialization - return */
    return;

}




/*
   start_memLoc
```

Description: This function handles the start of entering a memory location number. It first clears the current memory location, then sets the new system status based on the passed key value and returns that status.

Arguments: cur_status (enum status) - the current system status.
key_value (int) - value of the input key.

Return Value: (enum status) - the new status.

Input: None.

Output: None.

Error Handling: If there is an unexpected passed key value, the error handler is called with the error UNKNOWN_KEYCODE_INIT.

Algorithms: None.

Data Structures: None.

Shared Variables: memloc (changed) - set to 0.

Author: Glen George

Last Modified: June 7, 2006

```
*/

enum status  start_memLoc(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* clear the memory location */
    memloc = 0;

    /* and display it */
    display_memory_addr(memloc);


    /* save the current status so can go back to it after saving/restoring */
    old_status = cur_status;


    /* figure out the new system status */
    switch (key_value)  {

        case  KEYCODE_MEMSAVE:         /* <Memory Save> key was seen */
```

```
                                        /* new status is saving an IP to memory */
                                        cur_status = STAT_MEMSAVE;
                                        break;

        case  KEYCODE_MEMRECALL:/* <Memory Recall> key was seen */
                                        /* new status is recalling an IP from memory */
                                        cur_status = STAT_MEMRECALL;
                                        break;

        default:                /* some other key was seen */
                                        /* generate an error and leave status unchanged */
                                        process_error(UNKNOWN_KEYCODE_INIT);
                                        break;
    }


    /* and return the new status */
    return  cur_status;

}
```

```
/*
  clear_memLoc


  Description:      This function handles the clear key when a memory
                location number is being input.  It clears the current
                        memory location number, displays the now cleared number,
                        and returns with the system status it was passed.


  Arguments:       cur_status (enum status) - the current system status.
                key_value (int)         - value of the key that was
                                    input (ignored).
  Return Value:     (enum status) - the new status (same as current status).


  Input:           None.
  Output:          None.


  Error Handling:   None.


  Algorithms:      None.
  Data Structures:  None.


  Shared Variables: memloc (changed) - set to 0.


  Author:          Glen George
```

```
   Last Modified:    June 3, 2006

*/

enum status  clear_memLoc(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* clear the current memory location number */
    memloc = 0;

    /* display the new memory address */
    display_memory_addr(memloc);


    /* and return the current status */
    return  cur_status;

}
```

```
/*
   add_memDigit

   Description:      This function handles a digit key when a memory location
                     number is being input.  It adds the passed key value to
                     the current value of the memory location number.  If
                     there is an overflow (value larger than MEMORY_SIZE), the
                     error handler is called and the key is ignored.  The
                     function returns with the system status it was passed.

   Arguments:        cur_status (enum status) - the current system status.
                     key_value (int)          - value of the input key.
   Return Value:     (enum status) - the new status (same as current status).

   Input:            None.
   Output:           None.

   Error Handling:   If the input causes the memory location number to be to
                     large (greater than MEMORY_SIZE), the error handler is
                     called with the error MEMORY_ADDRESS_OVERFLOW and the key
                     is ignored.
```

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: memloc (changed) - updated to new memory location number
                                       based on the passed key value.

   Limitations:      Does not work for memory sizes greater than 6553.

   Author:           Glen George
   Last Modified:    June 3, 2006

*/

enum status  add_memDigit(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* check if there is room for the digit */
    if (((10 * memloc) + key_value) <= MEMORY_SIZE)  {

        /* have room for the digit - add it in to the current location */
```

```
            memloc = (10 * memloc) + key_value;
        }
        else  {

            /* too big of a value - call the error handler */
            process_error(MEMORY_ADDRESS_OVERFLOW);
        }



        /* display the new memory location number */
        display_memory_addr(memloc);



        /* all done adding the digit, return the passed status */
        return  cur_status;

}





/*
   del_memDigit
```

Description: This function handles a backspace key when a memory location number is being input. It removes the last input digit from the current value of the memory location number. If the current location is zero, it is assumed that there are no more digits in the location number and the error handler is called and the key is ignored. The function returns with the system status it was passed.

Arguments: cur_status (enum status) - the current system status.
key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status (same as current status).

Input: None.

Output: None.

Error Handling: If the current location number is zero then there are no digits to delete in the location number and the key is ignored and the error handler is called with the error MEMORY_ADDRESS_UNDERFLOW.

Algorithms: None.

Data Structures: None.

```
   Shared Variables: memloc (changed) - updated to the new memory location
                                 number by deleting the lowest digit.

   Author:          Glen George
   Last Modified:    June 3, 2006

*/

enum status  del_memDigit(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* are there any digits to delete? */
    if (memloc != 0)  {

        /* looks like there should be a digit to delete */
            /* remove the last digit - just divide by 10 */
            memloc /= 10;
    }
    else  {
```

```
        /* no digits to delete - call the error handler */
            process_error(MEMORY_ADDRESS_UNDERFLOW);
    }


    /* display the new memory location number */
    display_memory_addr(memloc);


    /* all done deleting the digit, return the passed status */
    return  cur_status;

}




/*
  save_mem

  Description:      This function handles the <Send/Enter> key when a memory
                location is being saved.  If the memory location number
                        is valid, it saves the current IP address (even if it is
                        zero) at that memory location.  If the memory location
```

number is out of range (0 or greater than MEMORY_SIZE), the error handler is called. The function always returns with the status it had when the save state was entered if there was no error and with the passed state if there was an error.

Arguments: cur_status (enum status) - the current system status.
key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status.

Input: None.

Output: None.

Error Handling: If the current location number is zero or greater than MEMORY_SIZE the key is ignored and the error handler is called with the error BAD_MEMORY_ADDRESS.

Algorithms: None.

Data Structures: None.

Shared Variables: memloc (accessed) - used to index the saved_IPs array.
saved_IPs (changed) - current IP address is written to this array.

```
   Author:          Glen George
   Last Modified:   June 7, 2006

*/

enum status  save_mem(enum status cur_status, int key_value)
{
    /* variables */
      /* none */



    /* is the memory location number valid */
    if ((memloc > 0) && (memloc <= MEMORY_SIZE))  {

        /* valid memory location number - save the IP address */
            saved_IPs[memloc] = get_calling_IP();

            /* and restore the system status */
            cur_status = old_status;

            /* restore the old IP address too */
            display_IP(get_calling_IP());
```

```
    }
    else  {

        /* bad memory location number - call the error handler */
            process_error(BAD_MEMORY_ADDRESS);
    }


    /* all done saving an address, return the possibly new status */
    return  cur_status;

}




/*
   recall_mem

   Description:      This function handles the <Send/Enter> key when a memory
                  location is being recalled.  If the memory location
                        number is valid, it sets the current IP address to the
                        address saved at that memory location.  If the memory
                        location number is out of range (0 or greater than
```

MEMORY_SIZE), the error handler is called. The function returns with the STAT_RECALLED state if there was no no error and it wasn't in the off-hook state before. It returns with the passed state if there was an error and with the off-hook state if that's the state it was in before.

Arguments: cur_status (enum status) - the current system status.
key_value (int) - value of the key that was input (ignored).

Return Value: (enum status) - the new status.

Input: None.

Output: None.

Error Handling: If the current location number is zero or greater than MEMORY_SIZE the key is ignored and the error handler is called with the error BAD_MEMORY_ADDRESS.

Algorithms: None.

Data Structures: None.

Shared Variables: memloc (accessed) - used to index the saved_IPs array.
saved_IPs (accessed) - recalled IP address is read from

```
                               this array.


   Author:          Glen George
   Last Modified:    June 7, 2006


*/


enum status  recall_mem(enum status cur_status, int key_value)
{
    /* variables */
      /* none */




    /* is the memory location number valid */
    if ((memloc > 0) && (memloc <= MEMORY_SIZE))  {


        /* valid memory location number - get the saved IP address */
            set_calling_IP(saved_IPs[memloc]);


            /* display the recalled IP address */
            display_IP(saved_IPs[memloc]);


            /* and change to the recalled memory state if we weren't off-hook */
```

```
        if (old_status != STAT_OFFHOOK)
            cur_status = STAT_RECALLED;
        else
            /* were off hook, so go back to that state */
            cur_status = STAT_OFFHOOK;
    }
    else  {

      /* bad memory location number - call the error handler */
        process_error(BAD_MEMORY_ADDRESS);
    }



    /* all done recalling memory, return the possibly new status */
    return  cur_status;

}
```

```
/****************************************************************************/
/*                                                                          */
/*                        TCPCONN                                           */
/*                  TCP Interface Functions                                 */
/*                   VoIP Telephone Project                                 */
/*                        EE/CS 52                                          */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the functions for managing the TCP interface for the
   VoIP Telephone Project.  The functions included are:
      have_tcp_connection    - have an incoming connection
      tcp_connection_answer  - answer an incoming connection
      tcp_connection_close   - close the connection
      tcp_connection_connect - connect to an IP address
      tcp_connection_init    - initialize the connection
      tcp_connection_restart - restart the TCP connections
      tcp_connection_rx      - attempt to receive TCP data
      tcp_connection_status  - get the connection status
      tcp_connection_tx      - attempt to transmit TCP data

   The local functions included are:
      accept_connection - accept a TCP connection being made (callback)
```

```
      busy_sent        - the busy packet has been sent (callback)
      check_for_close  - check if the connection is closed and handle it
      error_handler    - handle errors for TCP connections (callback)
      handle_connection - handle a TCP connection being made (callback)
      receive_data     - receive TCP data (callback)

   The locally global (shared) variable definitions included are:
      call_pcb    - protocol control block for a connected call
      cur_status  - status of the TCP connection
      have_rx_data - flag indicating received data is available
      listener    - the listener for incoming connections
      rx_data     - the data from a received data packet


   Revision History
      3/10/11  Glen George      Initial revision.
*/



/* library include files */
#include  "lwip/opt.h"
#include  "lwip/ip_addr.h"
#include  "lwip/pbuf.h"
```

```
#include  "ipv4/lwip/ip.h"
#include  "lwip/err.h"
#include  "lwip/tcp.h"

/* local include files */
#include  "interfac.h"
#include  "voipdefs.h"
#include  "tcpconn.h"
#include  "callutil.h"
#include  "error.h"



/* local function declarations */
static err_t  accept_connection(void *, struct tcp_pcb *, err_t);
static err_t  busy_sent(void *, struct tcp_pcb *, u16_t);
static err_t  generic_sent(void *, struct tcp_pcb *, u16_t);
static void   check_for_close(void);
static void   error_handler(void *, err_t);
static err_t  handle_connection(void *, struct tcp_pcb *, err_t);
static err_t  receive_data(void *, struct tcp_pcb *, struct pbuf *, err_t);
```

```
/* locally global (shared) variables */

/* status of the TCP connection */
static enum tcp_conn_status  cur_status;

/* protocol control block for a connected call */
static struct tcp_pcb  *call_pcb;

/* the listener for incoming connections */
static struct tcp_pcb  *listener;

/* a received data packet */
static short int  rx_data[AUDIO_BUFLEN];
/* flag indicating data is available */
static int       have_rx_data;


/* status functions */
```

/*

have_tcp_connection

Description: This function returns TRUE if there is a valid TCP connection.

Operation: The connection is first checked to see if it is closed. Then TRUE is returned if the shared variable call_pcb is not NULL and FALSE is returned if it is NULL.

Arguments: None.

Return Value: (char) - TRUE is returned if there is a valid TCP connection and FALSE is returned otherwise.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: call_pcb (accessed) - checked for NULL.

```
  Author:          Glen George
  Last Modified:    March 10, 2011

*/

char  have_tcp_connection()
{
    /* variables */
      /* none */



    /* check if the connection is closed */
    check_for_close();


    /* there is a connection if call_pcb is not NULL */
    return  (call_pcb != NULL);

}
```

/*

tcp_connection_status

Description: This function returns the current status of the TCP connection.

Operation: First the connection is checked to see if it has been closed. Then the value of the shared variable cur_status is returned.

Arguments: None.

Return Value: (enum tcp_conn_status) - current status of the TCP connection.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: cur_status (accessed) - value to return.

```
   Author:          Glen George
   Last Modified:    March 10, 2011

*/

enum tcp_conn_status  tcp_connection_status()
{
    /* variables */
      /* none */



    /* check if the connection has been closed */
    check_for_close();

    /* and just return the value of cur_status */
    return  cur_status;

}




/* connection functions */
```

/*

tcp_connection_init

Description: This function initializes the TCP connection software.

Operation: The shared variables are initialized and the LwIP TCP
interface is initialized. Then the interface is
restarted.

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: call_pcb (changed) - set to NULL.
cur_status (changed) - set to CALL_NO_CONNECTION.

```
                 listener (changed)   - initialized.


   Author:          Glen George
   Last Modified:    March 10, 2011


*/


void  tcp_connection_init()
{
    /* variables */
      /* none */




    /* currently there is no connection */
    cur_status = CALL_NO_CONNECTION;
    call_pcb = NULL;


    /* no listener either */
    listener = NULL;



    /* initialize the LwIP code */
    tcp_init();
```

```
    /* just restart the connection */
    tcp_connection_restart();



    /* done with initialization, return */
    return;

}




/*
   tcp_connection_restart

   Description:      This function restarts the TCP connection.  This is
                     useful when the link has changed.  For example, when the
                     IP address is changed.

   Operation:        Any open connections are closed.  Then the status is
                     reset and a listener is setup to listen for incoming
                     calls.
```

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: If there is an error setting up the listener the
                  process_error function is called with either a
                  NETERR_NOLISTEN or a NETERR_NOBIND error code, depending
                  on the exact error.

Algorithms: None.

Data Structures: None.

Shared Variables: call_pcb (changed)   - set to NULL.
              cur_status (changed)   - set to CALL_NO_CONNECTION.
              have_rx_data (changed) - set to FALSE.
              listener (changed)   - initialized.

Author: Glen George

Last Modified: March 10, 2011

*/

```
void  tcp_connection_restart()
{
   /* variables */
   struct tcp_pcb  *new_listener;       /* new listener on a port */




   /* check if there is a call in progress */
   if (call_pcb != NULL)
      /* have a call in progress, need to close it */
      tcp_abort(call_pcb);

   /* check if currently listening */
   if (listener != NULL)
      /* are listening, need to close the port */
      tcp_abort(listener);



   /* now there is no connection */
   cur_status = CALL_NO_CONNECTION;
   call_pcb = NULL;

   /* and no data available */
```

```
have_rx_data = FALSE;


/* create a protocol control block for listening for an incoming connection */
listener = tcp_new();

/* setup to listen on port CALL_LISTEN_PORT using the set IP address */
if (tcp_bind(listener, IP_ADDR_ANY, CALL_LISTEN_PORT) == ERR_OK)  {

   /* bound the port, now listen for a connection */
   new_listener = tcp_listen(listener);
   /* if have a listener, update the pointer (old pointer is bad now) */
   if (new_listener != NULL)  {

          /* have a listener - set it up */
      listener = new_listener;

      /* setup the callback function for accepting connections */
      tcp_accept(listener, accept_connection);

          /* no argument in callbacks */
          tcp_arg(listener, NULL);
      /* need a callback for the error handler */
      tcp_err(listener, error_handler);
```

```
        }
        else {

            /* error getting a listener, report it */
            process_error(NETERR_NOLISTEN);
        }
    }
    else {

        /* error binding to the port, report it */
        process_error(NETERR_NOBIND);
    }


    /* done restarting the interface, return */
    return;

}



/*
   tcp_connection_connect
```

Description: This function tries to connect to a remote device whose IP address is passed.

Operation: If there is currently no connection a connection is started.

Arguments: ip (unsigned long int) - the IP address to connect to.

Return Value: None.

Input: None.

Output: None.

Error Handling: If there is already a connection, the process_error function is called with the error MULTIPLE_CONNECTIONS. If there is an error setting up the connection the process_error function is called with the error NETERR_NOCONNECT.

Algorithms: None.

Data Structures: None.

Shared Variables: call_pcb (accessed) - checked to see if there is already a connection.

```
  Author:          Glen George
  Last Modified:    March 10, 2011

*/

void  tcp_connection_connect(unsigned long int ip)
{
    /* variables */
    struct tcp_pcb  *outgoing;             /* outgoing TCP connection */
    struct ip_addr   ipaddr;               /* IP address to connect to */



    /* check if there is already a connection */
    if (call_pcb == NULL)  {

        /* no connection yet, try to start one */
        /* create the protocol control block for the outgoing connection */
        outgoing = tcp_new();

            /* no argument in callbacks */
            tcp_arg(outgoing, NULL);
            /* need a callback for the error handler */
```

```
        tcp_err(outgoing, error_handler);


        /* setup the IP address structure */
        ipaddr.addr = htonl(ip);


      /* attempt to connect to the remote phone */
      if (tcp_connect(outgoing, &ipaddr, CALL_LISTEN_PORT, handle_connection) != ERR_OK)
        /* error setting up the connection, report it */
        process_error(NETERR_NOCONNECT);
    }
    else  {


      /* already have a connection - that's an error */
        process_error(MULTIPLE_CONNECTIONS);
    }



    /* done setting up the connection, return */
    return;


}
```

/*

tcp_connection_answer

Description: This function answers an incoming call whose connection was already established.

Operation: An answer packet is sent over the connection and the connection is setup to receive data.

Arguments: None.

Return Value: None.

Input: None.

Output: None.

Error Handling: If there is an error sending the packet, the process_error function is called with the error NETERR_SEND.

Algorithms: None.

Data Structures: None.

Shared Variables: call_pcb (accessed) - used to send the answer packet.

```
                cur_status (changed)   - updated to reflect the new
                                   connection status (connected).
                have_rx_data (changed) - set to FALSE.

   Author:          Glen George
   Last Modified:    March 10, 2011

*/

void  tcp_connection_answer(void)
{
    /* variables */
      /* none */




    /* setup the function to receive any data for the call */
    tcp_recv(call_pcb, receive_data);

    /* no data yet */
    have_rx_data = FALSE;



    /* send an answer packet */
```

```
    if (tcp_write(call_pcb, "A", 1, 1) != ERR_OK)
        /* error sending the packet - report it */
        process_error(NETERR_SEND);

    /* and the connection is now established */
    cur_status = CALL_CONNECTED;



    /* done answering the call, return */
    return;

}





/*
   tcp_connection_close

   Description:      This function closes the current connection.

   Operation:        If there is a connection, it is closed and the shared
                     variable call_pcb is set to NULL and the connection
                     status is set to no connection.
```

```
   Arguments:       None.
   Return Value:    None.

   Input:           None.
   Output:          None.

   Error Handling:  If there is an error closing the connection, the
                    process_error function is called with the error
                        NETERR_CLOSE.

   Algorithms:      None.
   Data Structures: None.

   Shared Variables: call_pcb (changed)     - set to NULL.
                     cur_status (changed)   - set to CALL_NO_CONNECTION.
                     have_rx_data (changed) - set to FALSE.

   Author:          Glen George
   Last Modified:   March 10, 2011

*/

void  tcp_connection_close()
```

```
{
    /* variables */
      /* none */



    /* if there is a connection, close it, watching for errors */
    if ((call_pcb != NULL) && (tcp_close(call_pcb) != ERR_OK))
        /* error closing the connection - report it */
        process_error(NETERR_CLOSE);


    /* there is no longer a connection */
    cur_status = CALL_NO_CONNECTION;
    call_pcb = NULL;
    /* and no received data */
    have_rx_data = FALSE;


    /* done closing the connection, return */
    return;

}
```

/*

tcp_connection_rx

Description: This function attempts to get received data and returns it in the passed buffer.

Operation: If there is received data it is copied to the passed buffer up to the total data available and the buffer length and TRUE is returned. Otherwise FALSE is returned.

Arguments: buf (short int *) - pointer to the buffer for the received data.

len (int) - length of the buffer.

Return Value: (char) - TRUE if there was data and the buffer was filled and FALSE otherwise.

Input: None.

Output: None.

Error Handling: None.

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: have_rx_data (changed) - checked and set to FALSE.
                     rx_data (accessed)     - copied to the passed buffer.

   Author:           Glen George
   Last Modified:    March 10, 2011

*/

char  tcp_connection_rx(short int *buf, int len)
{
    /* variables */
    char  status;              /* return status */

    int   i;                   /* general loop index */



    /* check if data is available */
    if (have_rx_data)  {
```

```
        /* have data, copy it into the buffer */
            for (i = 0; ((i < len) && (i < AUDIO_BUFLEN)); i++)
                buf[i] = rx_data[i];

        /* no longer have data (already copied) */
            have_rx_data = FALSE;

        /* and the data was successfully copied */
            status = TRUE;
    }
    else  {

        /* no data to copy, return FALSE */
            status = FALSE;
    }

    /* return whether or not any data was actually received */
    return  status;

}
```

```
/*
  tcp_connection_tx


  Description:     This function attempts to send the passed data over the
                 TCP connection.


  Operation:      A packet is created for the passed data and it is sent
                   over the connection.  If there is an error sending the
                   packet, the error handler is called and FALSE is
                   returned.


  Arguments:      buf (short int *) - pointer to the buffer with the data
                          to be sent (16-bit values).
                 len (int)        - length of the buffer (number of
                                                16-bit values).
  Return Value:    (char) - TRUE if the passed data was successfully sent
                   and FALSE otherwise.


  Input:          None.
  Output:          None.


  Error Handling:   If there is an error sending the packet, FALSE is
                 returned and the process_error function is called with
```

```
                the error NETERR_SEND.


   Algorithms:       None.

   Data Structures:  None.


   Shared Variables: call_pcb (accessed) - used to send the data packet.


   Author:          Glen George

   Last Modified:    March 10, 2011


*/


char  tcp_connection_tx(short int *buf, int len)

{

    /* variables */

    char  packet[AUDIO_BUFLEN * 2 + 1];                  /* packet to send */


    char  status;                                /* return status */


    int   i;                                           /* general loop index */





    /* create the packet */
```

```
/* packet type is data (D) */
packet[0] = 'D';

/* copy the data from the passed buffer */
for (i = 0; ((i < len) && (i < AUDIO_BUFLEN)); i++)  {
   /* break the passed 16-bit values into bytes */
        packet[2 * i + 1] = (buf[i] >> 8) & 0xFF;
        packet[2 * i + 2] = buf[i] & 0xFF ;
}



/* now send the packet, watching for errors */
if (tcp_write(call_pcb, packet, sizeof(packet), 1) != ERR_OK)  {
   /* error sending the packet - report it */
   process_error(NETERR_SEND);
        /* remember that there was an error */
        status = FALSE;
}
else  {

   /* no error */
        status = TRUE;
}
```

```
    /* done sending the data, return whether we were successful or not */
    return  status;

}




/* utility functions */


/*
   check_for_close

   Description:      This function checks if the call connection has been
                     closed (most likely by the remote system) and, if so,
                     cleans up.

   Operation:        The state of the TCP connection is checked and if it is
                     closed, the connection is closed at this end and the
                     shared variables are reset to no call.
```

```
   Arguments:       None.
   Return Value:    None.

   Input:           None.
   Output:          None.

   Error Handling:  If there is an error closing the connection, the
                    process_error function is called with the error
                        NETERR_CLOSE.

   Algorithms:      None.
   Data Structures: None.

   Shared Variables: call_pcb (changed)     - checked and possibly set to
                                  NULL.
                    cur_status (changed)   - possibly reset to no connection.
                    have_rx_data (changed) - set to FALSE.

   Author:          Glen George
   Last Modified:   March 10, 2011

*/
void  check_for_close()
{
```

```
/* variables */
  /* none */




/* check if there is a connection and it is now closed */
if ((call_pcb != NULL) && (call_pcb->state == CLOSED))  {

    /* have a closed connection, close it on this end too */
    if (tcp_close(call_pcb) != ERR_OK)
        /* error closing the connection - report it */
        process_error(NETERR_CLOSE);

    /* there is no longer a connection */
    cur_status = CALL_NO_CONNECTION;
    call_pcb = NULL;
    /* and no data */
    have_rx_data = FALSE;
}



/* done checking if the connection is closed, return */
return;
```

```
}




/* callback functions */


/*
   handle_connection

   Description:      This function handles a connection being successfully
                     made.

   Operation:        The status is changed to reflect we are trying to connect
                     (actual call, not TCP connection) and the receive data
                     handler is setup for the connection.

   Arguments:        arg (void *)          - not used.
                     conn (struct tcp_pcb *) - protocol control block for the
                                 connection.
                     err (err_t)           - error code for the connection.
   Return Value:     (err_t) - error status, the passed error code.
```

```
   Input:          None.
   Output:         None.

   Error Handling:   None.

   Algorithms:      None.
   Data Structures:  None.

   Shared Variables: call_pcb (changed)     - set to the passed connection.
                cur_status (changed)   - changed to indicate trying to
                                establish a call.
                have_rx_data (changed) - set to FALSE.

   Author:          Glen George
   Last Modified:    March 10, 2011

*/

static err_t  handle_connection(void *arg, struct tcp_pcb *conn, err_t err)
{
    /* variables */
      /* none */
```

```
/* have a connection, this is the calling protocol control block */
call_pcb = conn;

/* setup the callbacks */

/* no argument in callbacks */
tcp_arg(conn, NULL);
/* need a callback for the error handler */
tcp_err(conn, error_handler);
/* and setup the data processing callback */
tcp_recv(conn, receive_data);


/* the status is now that we are trying to establish a call */
cur_status = CALL_CONNECTING;

/* no data yet */
have_rx_data = FALSE;


/* return the passed error code */
return  err;
```

```
}




/*
  accept_connection

  Description:      This function handles accepting a connection.

  Operation:        If there is currently no call in progress the connection
                    is accepted and a ringing packet is sent.  If there is a
                    call in progress the connection is accepted and the busy
                    packet is sent.

  Arguments:        arg (void *)          - not used.
                    conn (struct tcp_pcb *) - protocol control block for the
                                connection.
                    err (err_t)           - error code for the connection.
  Return Value:     (err_t) - error status, always the passed error code,

  Input:            None.
  Output:           None.
```

```
   Error Handling:   If there is an error sending the busy or ringing packet,
                     the process_error function is called with the error
                           NETERR_SEND.

   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: call_pcb (changed)   - checked if NULL and set to the
                                 incoming connection if so.
                     cur_status (changed) - changed to CALL_CONNECTING if are
                                 trying to establish a connection.

   Author:           Glen George
   Last Modified:    March 10, 2011

*/
static err_t  accept_connection(void *arg, struct tcp_pcb *conn, err_t err)
{
    /* variables */
      /* none */




    /* accept the incoming connection */
```

```
tcp_accepted(conn);

/* no argument in callbacks */
tcp_arg(conn, NULL);
/* need a callback for the error handler */
tcp_err(conn, error_handler);


/* is there already a call in progress */
if (call_pcb == NULL)  {

   /* no call in progress - remember the incoming connection */
   call_pcb = conn;
       /* set the calling IP number */
       set_calling_IP(ntohl(conn->remote_ip.addr));
tcp_sent(conn, generic_sent);

       /* send a ringing packet */
       if (tcp_write(conn, "R", 1, 1) != ERR_OK)
          /* error sending the packet - report it */
          process_error(NETERR_SEND);

   /* and the status is now that we are trying to connect */
       cur_status = CALL_CONNECTING;
```

```
    }
    else  {

        /* already have a call in progress - send back a busy signal */
            /* need callback for data sent */
            tcp_sent(conn, busy_sent);

            /* send the busy packet */
            if (tcp_write(conn, "B", 1, 1) != ERR_OK)
                /* error sending the packet - report it */
                process_error(NETERR_SEND);
    }


    /* nothing else to do, return with the passed error code */
    return  err;

}




/*
   receive_data
```

Description: This function handles receiving data over a TCP connection.

Operation: If there is currently no call in progress the connection is accepted and a ringing packet is sent. If there is a call in progress the connection is accepted and the busy packet is sent.

Arguments: arg (void *) - not used.

conn (struct tcp_pcb *) - protocol control block for the connection.

packet (struct pbuf *) - pointer to the packet holding the data received.

err (err_t) - error code for the connection.

Return Value: (err_t) - error status, always the passed error code,

Input: None.

Output: None.

Error Handling: If the packet type is unknown, the process_error function is called with the error NETERR_UNKNOWN_PACKET.

Algorithms: None.

```
   Data Structures:  None.

   Shared Variables: cur_status (changed)   - possibly updated based on actual
                                              packets received.
                     have_rx_data (changed) - set to TRUE if it is a data
                                              packet.
                     rx_data (changed)      - filled with packet data if it is
                                              a data packet.

   Author:          Glen George
   Last Modified:   March 10, 2011

*/

static err_t  receive_data(void *arg, struct tcp_pcb *conn,
                           struct pbuf *packet, err_t err)
{
    /* variables */
    int  i;             /* general loop index */



    /* figure out what kind of packet this was */
    switch(((char *)(packet->payload))[0])  {
```

```
case 'A':   /* answer packet */
            /* the other side is answering, change status */
            cur_status = CALL_CONNECTED;
            break;

    case 'R':   /* ringing packet */
            /* the phone is ringing on the other end */
            cur_status = CALL_RINGING;
            break;

    case 'B':   /* busy packet */
            /* the phone is busy on the other end */
            cur_status = CALL_BUSY;
            break;

case 'D':   /* data packet */
            /* have data, copy it into the buffer */
            for (i = 0; i < AUDIO_BUFLEN; i++)  {
               /* copy a word into the data buffer */
                   rx_data[i] = ((((unsigned char *)(packet->payload))[2 * i + 1]) << 8) |
                             ((((unsigned char *)(packet->payload))[2 * i + 2]) & 0xFF);
            }
            /* have data now */
            have_rx_data = TRUE;
```

```
                break;

        default:    /* unknown packet */
                /* report an error, this shouldn't happen */
                process_error(NETERR_UNKNOWN_PACKET);
                break;
    }


    /* acknowledge that we've gotten the payload */
    tcp_recved(conn, packet->len);


    /* done with the packet, release it */
    pbuf_free(packet);


    /* have processed the received data, return with the passed error code */
    return  err;

}
```

```
/*
  busy_sent

  Description:      This function handles the busy packet being successfully
                    sent.

  Operation:        The connection is closed while watching for an error.

  Arguments:        arg (void *)           - not used.
                    conn (struct tcp_pcb *) - protocol control block for the
                                 connection over which the busy
                                             packet was sent.
                    len (u16_t)            - number of bytes sent (ignored).
  Return Value:     (err_t) - error code for handling the data being sent,
                    always ERR_OK.

  Input:            None.
  Output:           None.

  Error Handling:   If there is an error closing the connection, the
                    process_error function is called with the error
                    NETERR_CLOSE.
```

```
   Algorithms:       None.

   Data Structures:  None.


   Shared Variables: None.


   Author:           Glen George

   Last Modified:    March 10, 2011


*/

static err_t  busy_sent(void *arg, struct tcp_pcb *conn, u16_t len)

{

    /* variables */

      /* none */




    /* the busy packet has been sent, close the connection */

    if (tcp_close(conn) != ERR_OK)

        /* error closing the connection - report it */

            process_error(NETERR_CLOSE);



    /* nothing else to do, return with no error */

    return  ERR_OK;
```

```
}



/*
   generic_sent

   Description:     This function handles the busy packet being successfully
                    sent for states other than busy.

   Operation:       Used only to help LWIP run smoothly, returns immediately.

   Arguments:       arg (void *)           - not used.
                    conn (struct tcp_pcb *) - protocol control block for the
                                              connection over which the busy
                                              packet was sent.
                    len (u16_t)            - number of bytes sent (ignored).
   Return Value:    None.

   Input:           None.
   Output:          None.

   Error Handling:  None.
```

```
   Algorithms:       None.
   Data Structures:  None.

   Shared Variables: None.

   Author:          Josh Fromm
   Last Modified:    March 16, 2012

*/
static err_t  generic_sent(void *arg, struct tcp_pcb *conn, u16_t len)
{
    /* variables */
      /* none */



    /* return nothing */
    return ERR_OK;

}



/*
   error_handler
```

Description: This function handles any errors from a TCP connection.

Operation: The process_error function is called with NETERR_GENERAL.

Arguments: arg (void *) - not used.

err (err_t) - error code for the error (ignored).

Return Value: None.

Input: None.

Output: None.

Error Handling: None.

Algorithms: None.

Data Structures: None.

Shared Variables: None.

Author: Glen George

Last Modified: March 10, 2011

*/

static void error_handler(void *arg, err_t err)

```
{
   /* variables */
     /* none */



   /* inform the system there was a general network error */
   process_error(NETERR_GENERAL);



   /* nothing else to do, return */
   return;

}
```

```
/****************************************************************************/
/*                                                                          */
/*                              TCPCONN.H                                   */
/*                        TCP Interface Functions                           */
/*                             Include File                                 */
/*                         VoIP Telephone Project                           */
/*                               EE/CS 52                                   */
/*                                                                          */
/****************************************************************************/

/*
   This file contains the constants, structures, and function prototypes for
   the TCP interface functions for the VoIP Telephone Project which are
   defined in tcpconn.c.


   Revision History
      3/10/11  Glen George       Initial revision.
*/



#ifndef  I__TCPCONN_H__
```

```
   #define  I__TCPCONN_H__



/* library include files */
  /* none */

/* local include files */
  /* none */




/* constants */
   /* none */





/* structures, unions, and typedefs */

/* status of the TCP connection */
enum tcp_conn_status  {
   CALL_NO_CONNECTION,              /* no connecton */
   CALL_CONNECTING,                 /* trying to setup a full connection */
```

```
   CALL_RINGING,          /* have connection, it's ringing */
   CALL_BUSY,             /* have connection, it's busy */
   CALL_CONNECTED         /* have connection, can talk */
};




/* function declarations */


/* status functions */
char            have_tcp_connection(void);   /* have incoming connection */
enum tcp_conn_status  tcp_connection_status(void);       /* get status */



/* connection functions */
void  tcp_connection_answer(void);          /* answer an incoming connection */
void  tcp_connection_close(void);           /* close the connection */
void  tcp_connection_connect(unsigned long int);/* connect to an IP address */
void  tcp_connection_init(void);            /* initialize connection */
void  tcp_connection_restart(void);         /* restart a connection */



/* receive/transmit functions */
```

```
char  tcp_connection_rx(short int *, int);      /* try to receive data */

char  tcp_connection_tx(short int *, int);      /* try to transmit data */



#endif
```

```
/****************************************************************************/
/*                                                                          */
/*                              VOIPDEFS.H                                  */
/*                          General Definitions                             */
/*                             Include File                                 */
/*                         VoIP Telphone Project                            */
/*                               EE/CS  52                                  */
/*                                                                          */
/****************************************************************************/


/*
   This file contains the general definitions for the VoIP Telephone.  This
   includes constant and structure definitions along with the function
   declarations for the assembly language functions.


   Revision History:
      6/3/06   Glen George      Initial revision.
      6/5/06   Glen George      Removed the "far" keyword from the pointers
                                in the assembly language function
                                declarations.
      6/6/06   Glen George      Removed buffer structure definition.
      6/6/06   Glen George      Fixed missing declaration of update_tx.
      6/9/09   Glen George      Added declarations for networking functions.
```

```
     6/12/09  Glen George       Fixed minor compiler error.
     2/28/11  Glen George       Updated prototypes for call_start, update_rx,
                                and update_tx to match new specification.
     3/9/11   Glen George       Added some networking constants and removed
                                the audio constants, they are no longer used.
*/



#ifndef  I__VOIPDEFS_H__
   #define  I__VOIPDEFS_H__



/* library include files */
#ifndef  NO_LWIP                  /* don't include files if not using LWIP */
  #include  "lwip/pbuf.h"
#endif


/* local include files */
#include  "interfac.h"
```

```
/* constants */


/* general constants */
#define  FALSE      0
#define  TRUE       !FALSE
#define  NULL       (void *) 0



/* IP parameters */
#define  NUM_IP_DIGITS   12 /* number of decimal digits in an IP address */
#define  MAGIC_IP       0x00FF55AA  /* IP address that should not occur */
#define  CALL_LISTEN_PORT  0x4747 /* port to listen for connections */



/* miscellaneous constants */
#define  MAX_NAME_LEN  40           /* maximum length of a caller ID name */





/* structures, unions, and typedefs */


/* status types */
enum status  {  STAT_IDLE,               /* system idle */
```

```
        STAT_OFFHOOK,           /* phone is off hook */
        STAT_RINGING,           /* incoming call */
        STAT_CONNECTING,   /* attempting to connect */
        STAT_CONNECTED,         /* connected to remote phone */
        STAT_SETIP,        /* setting the IP address */
        STAT_SETSUBNET,         /* setting the subnet address */
        STAT_SET_GW,            /* setting the gateway address */
        STAT_MEMSAVE,           /* saving an address to memory */
        STAT_MEMRECALL,         /* recalling an address from memory */
        STAT_RECALLED,          /* just recalled an address from memory */
        NUM_STATUS         /* number of status types */
    };

/* key codes */
enum keycode  {  KEYCODE_0,     /* <0>      */
      KEYCODE_1,     /* <1>      */
      KEYCODE_2,     /* <2>      */
      KEYCODE_3,     /* <3>      */
      KEYCODE_4,     /* <4>      */
      KEYCODE_5,     /* <5>      */
      KEYCODE_6,     /* <6>      */
      KEYCODE_7,     /* <7>      */
      KEYCODE_8,     /* <8>      */
      KEYCODE_9,     /* <9>      */
```

```
        KEYCODE_ESC,       /* <ESC>          */
        KEYCODE_BS,        /* <Backspace>    */
        KEYCODE_SEND,      /* <Send>         */
        KEYCODE_OFFHOOK,   /* Off-Hook       */
        KEYCODE_ONHOOK,    /* On-Hook        */
        KEYCODE_SETIP,     /* <Set IP>       */
        KEYCODE_SETSUBNET, /* <Set Subnet>   */
        KEYCODE_SET_GW,    /* <Set Gateway>  */
        KEYCODE_MEMSAVE,   /* <Memory Save>  */
          KEYCODE_MEMRECALL, /* <Memory Recall> */
          KEYCODE_ILLEGAL,   /* other keys     */
            NUM_KEYCODES       /* number of key codes */
      };

/* key types */
enum keytype  {  KEYTYPE_DIGIT,     /* <0> <1> <2> <3> <4> */
                 /* <5> <6> <7> <8> <9> */
        KEYTYPE_ESC,       /* <ESC>          */
        KEYTYPE_BS,        /* <Backspace>    */
        KEYTYPE_SEND,      /* <Send>         */
        KEYTYPE_OFFHOOK,   /* Off-Hook       */
        KEYTYPE_ONHOOK,    /* On-Hook        */
        KEYTYPE_SETIP,     /* <Set IP>       */
        KEYTYPE_SETSUBNET, /* <Set Subnet>   */
```

```
            KEYTYPE_SET_GW,     /* <Set Gateway>       */
            KEYTYPE_MEMSAVE,    /* <Memory Save>       */
                KEYTYPE_MEMRECALL,  /* <Memory Recall>     */
                KEYTYPE_UNKNOWN,    /* unknown key type    */
                    NUM_KEYTYPES        /* number of key types */
          };

/* declare the ethernet buffer if not using LWIP code */
#ifdef  NO_LWIP
   struct pbuf  { struct pbuf *next; };
#endif




/* function declarations */


/* update needed functions */
unsigned char  update_rx(short int *);  /* record data update */
unsigned char  update_tx(short int *);  /* play data update */


/* keypad functions */
unsigned char  key_available(void);     /* key is available */
int            getkey(void);            /* get a key */
```

```
/* display functions  */
void  display_IP(unsigned long int);       /* display the track time */
void  display_memory_addr(unsigned int);    /* display the track number */
void  display_status(unsigned int);        /* display the system status */

/* audio functions */
void  call_start(short int *);         /* start playing */
void  call_halt(void);               /* halt play or record */

/* timing function */
int  elapsed_time(void);

/* networking functions */
char        ether_init(void);
char        ether_transmit(struct pbuf *);
char        ether_rx_available(void);
struct pbuf  *ether_receive(void);


#endif
```