```
@Done
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@
@ Display.s
@
@ Description: Contains code for controlling the display component of
@ the EE52 VoIP Project
@
@ Table of Contents:
@   - display_init: Call to initialize the shared variables and peripherals
@                   for interfacing with the C12832 Display
@   - display_memory_addr: Displays the passed memory address on display
@   - display_IP: Displays the passed IP address on display
@   - display_status: Displays the passed status on display
@   - redraw: Redraws the display buffer
@   - queueDisplayCommand: Queues a command to send to the display
@   - setBacklight: Sets the backlight on or off
@   - displayHandler: Handles setting up display DMA stream and also
@                     sending commands to display
@
@
@ Revision History:
@ Name          Comment                 Date
@ Will Werst    Initial version         Some lonely night around 6/10/17
@ Will Werst    Comment                 October 2017

.include    "macro.inc"
.include    "display.inc"
.include    "pio.inc"
.include    "at91rm9200.inc"
.include    "system.inc"
.include    "interfac.inc"


.arm
.text

@ display_init
@
@ Description: Initializes the display
@
@ Operational Description: The PIO pins are initialized,
@                          and then a series of commands
@                          are queued up to initialize the
@                          display. Then, the display handler
@                          is installed and kickstarted by starting
@                          the display output DMA engine.
@
@ Arguments: None
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: None
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: None
@
@ Algorithms: None
```

```
@
@ Data Structures: None
@
@ Limitations: None
@
@ Registers Changed (besides ARM convention r0-r3): None
@
@ Known Bugs: None
@
@ Special notes: None
@
@ Revision History:
@ Name              Comment                Date
@ Will Werst         Initial version      Some lonely night around 6/10/17

.global display_init
display_init:
    mSTARTFNC

    @First, initialize the PIO pins
    mSET_HREG   PMC_PCER,   (1 << 13)
    LDR     r0, =(0xE)          @MOSI, SPCK, NPCS0
    LDR     r1, =PIO_A
    LDR     r2, =(PIO_PERA | PIO_OUTPUT)
    BL      configPIOPin

    LDR     r0, =(1 << DISP_A0) | (1 << DISP_BCKLIGHT) | (1 << DISP_RST)
    LDR     r1, =PIO_A
    LDR     r2, =(PIO_NORM | PIO_OUTPUT)
    BL      configPIOPin

    mSET_HREG PIOA_CODR,    (1 << DISP_RST) @Reset display

    @Need to wait for the display to reset, this loop
    @introduces a delay to allow the display to reset
    LDR r0, =0xFFFF
count:
    SUB r0, #1
    CMP r0, #0
    BNE count
    @Done with delay

    mSET_HREG PIOA_SODR,    (1 << DISP_RST) @Stop reset signal to display

    @Set backlight on
    LDR     r0, =TRUE
    BL      setBacklight

    @Enqueue all of the commands for initializing display now
    LDR     r0, =NHD_RESET
    BL      queueDisplayCommand
    @LDR        r0, =NHD_ON
    @BL     queueDisplayCommand
    @LDR        r0, =0x2F
    @BL     queueDisplayCommand
    @LDR        r0, =0x26
    @BL     queueDisplayCommand
    @LDR        r0, =NHD_RMW
    @BL     queueDisplayCommand
    @LDR        r0, =0xA4
    @BL     queueDisplayCommand
    @LDR        r0, =0x81
    @BL     queueDisplayCommand
    @LDR        r0, =0x2F
    @BL     queueDisplayCommand
    LDR     r0, =0xAE
    BL      queueDisplayCommand
    LDR     r0, =0xA2
    BL      queueDisplayCommand

    LDR     r0, =0xA0
    BL      queueDisplayCommand
    LDR     r0, =0xC8
    BL      queueDisplayCommand

    LDR     r0, =0x22
    BL      queueDisplayCommand
    LDR     r0, =0x2F
    BL      queueDisplayCommand

    LDR     r0, =0x40
    BL      queueDisplayCommand
    LDR     r0, =0xAF
    BL      queueDisplayCommand

    LDR     r0, =0x81
    BL      queueDisplayCommand
    LDR     r0, =0x17
    BL      queueDisplayCommand

    LDR     r0, =0xA6
    BL      queueDisplayCommand

    @Setup SPI
    mSET_HREG   SPI_CR, SPI_CR_RESET
    mSET_HREG   SPI_MR, SPI_MR_VAL
    mSET_HREG   SPI_CSR0, SPI_CSR0_VAL

    @Install interrupt for when DMA finished to display
    mSET_HREG   AIC_SVR13, displayHandler
    mSET_HREG   AIC_SMR13, AIC_SMR13_VAL
    mSET_HREG   AIC_IECR, (1 << 13)

    @Finish setting up SPI and enable DMA interrupt
    mSET_HREG   SPI_CR, SPI_CR_EN
    mSET_HREG   SPI_PTCR,   SPI_DMA_ENABLE  @Enable the DMA controller
    mSET_HREG   SPI_IER, SPI_IER_VAL
    mRETURNFNC

@ display_memory_addr
@
@ Description: Displays the passed memory address on the display.
@
@ Operational Description: registers 4-7 are pushed to stack so they
@                          can be used. The string is composed by looking
@                          up each character in a hexToAscii table.
@                          After the string is finished being written,
@                          the buffers are redrawn using the redraw
@                          function, and then the method returns.
@                          The display is redrawn the next time the DMA
@                          engine writes the buffer to the display
@
@
@ Arguments: r0 - unsigned int of address to display
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: TextMessageR2[W] - The message displayed on 2nd line
@                                      is replaced with the memory address
@                                       string representation generated
@                                       by this function.
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: None
@
@ Limitations: None
@
@ Registers Changed (besides ARM convention r0-r3): None
@
@ Known Bugs: None
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment               Date
@ Will Werst        Initial version      Some lonely night around 6/10/17

.global display_memory_addr
display_memory_addr:
    mSTARTFNC

    PUSH    {r4-r7}
    LDR r5, =TextMessageR2          @String to write to
    LDR r6, =0                      @ASCII string position


    MOV r7, r0                      @Store memory address to use later
    @Get tens digit
    LDR r1, =10
    MOV r0, r7
    BL  divide
    LDR r1, =hexToASCII
    LDRB    r0, [r1, r0]
    STRB    r0, [r5, r6]
    ADD     r6, #1

    @Remove tens digit value from memory address
    LDR r1, =10
    MOV r0, r7
    BL  mod
    MOV r7, r0                      @r7 now contains original address with tens digit 0

    @Get ones digit
    LDR r1, =hexToASCII
    LDRB    r0, [r1, r0]
    STRB    r0, [r5, r6]
    ADD     r6, #1

    @Add null termination to string
    LDRB    r0, =ASCII_NULL
    STRB    r0, [r5, r6]


    BL redraw                       @Redraw the buffer

    POP {r4-r7}
    mRETURNFNC

@ display_IP
@
@ Description:
@
@ Operational Description: registers 4-7 are pushed to stack so they
@                          can be used. The octets are then extracted from
@                          the input IP address to be displayed, and each
@                          octet is pushed to the stack. Then, each octet
@                          is popped off the stack in a loop and added to
@                          the 2nd row on display string.
@                          Finally, the redraw method is used to update
@                          the display buffers.
@
@ Arguments: None
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: TextMessageR2[W] - The message displayed on 2nd line
@                                      is replaced with the IP address
@                                      string representation generated
@                                      by this function.
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: None
@
@ Limitations: None
@
@ Registers Changed (besides ARM convention r0-r3): None
@
@ Known Bugs: None
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment               Date
@ Will Werst        Initial version     Some lonely night around 6/10/17

.global display_IP
display_IP:
    mSTARTFNC

    PUSH {r4-r7}
    @Get 4th octet
    AND r1, r0, #0xFF
    LSR r0, #8
    PUSH {r1}

    @Get 3rd octet
    AND r1, r0, #0xFF
    LSR r0, #8
    PUSH {r1}

    @Get 2nd octet
    AND r1, r0, #0xFF
    LSR r0, #8
    PUSH {r1}

    @Get 1st octet
    AND r1, r0, #0xFF
    LSR r0, #8
    PUSH {r1}

    LDR r4, =NUM_OCTETS_IP
    LDR r5, =TextMessageR2          @String to write to
    LDR r6, =0                      @ASCII string position
OctetToASCII:
    POP {r0}                        @Get next octet
    MOV r7, r0                      @Store octet so can operate on it and recover later


    @Get hundreds digit
    LDR r1, =100
    MOV r0, r7
    BL  divide
    LDR r1, =hexToASCII
    LDRB    r0, [r1, r0]
    STRB    r0, [r5, r6]
    ADD     r6, #1

    @Remove hundreds digit value from octet
    LDR r1, =100
    MOV r0, r7
    BL  mod
    MOV r7, r0                      @r7 now contains original octet with hundreds digit 0

    @Get tens digit
    LDR r1, =10
    MOV r0, r7
    BL  divide
    LDR r1, =hexToASCII
    LDRB    r0, [r1, r0]
    STRB    r0, [r5, r6]
    ADD     r6, #1

    @Remove tens digit value from octet
    LDR r1, =10
    MOV r0, r7
    BL  mod
    MOV r7, r0                      @r7 now contains original octet with hundreds and tens
    digit 0

    @Get ones digit
    LDR r1, =hexToASCII
    LDRB    r0, [r1, r0]
    STRB    r0, [r5, r6]
    ADD     r6, #1

    @Add '.' character to terminate string
    LDRB    r0, =ASCII_DOT
    STRB    r0, [r5, r6]
    ADD     r6, #1

    SUB r4, #1
    CMP r4, #0
    BHI OctetToASCII

    @Add null termination to string
    LDRB    r0, =ASCII_NULL
    SUB     r6, #1                  @Left a '.' character at end, will overwrite with null.
    STRB    r0, [r5, r6]


    BL redraw

    POP {r4-r7}
    mRETURNFNC


@ display_status
@
@ Description: Displays the passed display status
@
@ Operational Description: The string corresponding to the passed display status
@                          is looked up and rendered to the display buffer. If
@                          the status is STATUS_IDLE, the rest of the display
@                          is cleared.
@
@ Arguments: r0 - status to display
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: TextMessageR1[W] - The message displayed on 1nd line
@                                      is replaced with the status
@                                      string representation generated
@                                      by this function.
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: None
@
@ Limitations: None
@
@ Registers Changed (besides ARM convention r0-r3): None
@
@ Known Bugs: None
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment              Date
@ Will Werst        Initial version     Some lonely night around 6/10/17

.global display_status
display_status:
    mSTARTFNC
    CMP     r0, #STATUS_IDLE            @Check if status is idle
    LDREQ     r1, =TextMessageR2        @And if so, clear the display
    LDREQ     r2, =ASCII_NULL
    @Clear the 2nd row of display
    STREQB    r2, [r1]

    @Load the status message to TextMessageR1
    LDR     r1, =StatusMessages
    LDR     r1, [r1, r0, LSL #2]
    mSTOREFROMREG r1, r0, TextMessageR1
    @Redraw the display
    BL  redraw
    mRETURNFNC


@ redraw
@
@ Description: The display buffer is redrawn. Call
@              after changing TextMessageR1 or TextMessageR2.
@
@ Operational Description: The display is cleared using the clear_displaybuffer
@                          C method in displayrenderer.c. Then, the 1st and 2nd
@                          row are rendered using the render_displaybuffer
@                          C method in displayrenderer.c
@
@ Arguments: None
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: None
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: None
@
@ Limitations: None
@
@ Registers Changed (besides ARM convention r0-r3): None
@
@ Known Bugs: None
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment              Date
@ Will Werst        Initial version     Some lonely night around 6/10/17

.global redraw
redraw:
    mSTARTFNC
    LDR r0, =DispBuffer
    LDR r1, =(NUM_COLS*NUM_PAGES)
    BL  clear_displaybuffer

    LDR r0, =TextMessageR1
    LDR r0, [r0]
    LDR r1, =DispBuffer
    LDR r2, =NUM_COLS
    BL render_displaybuffer

    LDR r0, =TextMessageR2
    LDR r1, =(DispBuffer+NUM_COLS)
    LDR r2, =NUM_COLS
    BL render_displaybuffer
    mRETURNFNC


@ queueDisplayCommand
@
@ Description: queues the command passed in r0 to be sent to the display.
@
@ Operational Description: Interrupts are disabled since this accesses
@                          the command queue which is also accessed in
@                          the displayHandler. The command is enqueued if
@                          there is space. If the command was enqueued
@                          successfully, the method exits with TRUE, else FALSE.
@
@
@ Arguments: r0 - command to send over SPI
@
@ Return values: r0 - TRUE if added successfully, FALSE if not added.
@
@ Local variables: None
@
@ Shared variables: CommandQueueSize[RW] - read to check queue size,
@                                         and incremented after adding queue element.
@                   ActiveCommandQueue[W] - command is enqueued.
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: None
@
@ Error Handling: If queue is full, the method exits with FALSE
@
@ Algorithms: None
@
@ Data Structures: Queue
@
@ Limitations: None known
@
@ Registers Changed (other than r0-r3): None
@
@ Known Bugs: None known
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment               Date
@ Will Werst        Initial version     6/22/2017

queueDisplayCommand:
    mSTARTFNC
    mSTARTCRITCODE                      @Enter critical code, r7 used by this macro
    mLOADTOREG  r1, CommandQueueSize
    CMP r1, #COM_QUEUE_LENGTH
    BLO addCommandToQ
    @B  CommandQFull
CommandQFull:
    LDR r0, =FALSE
    B   sDCEndCritCode
addCommandToQ:
    LDR r2, =ActiveCommandQueue         @Load pointer to activeCommandQueue
    LDR r2, [r2]                        @Load the queue pointed to by activeCommandQueue
    STRB r0, [r2,r1]
    ADD r1, #1
    mSTOREFROMREG r1, r0, CommandQueueSize
    LDR r0, =TRUE
sDCEndCritCode:
    mENDCRITCODE                        @Exit critical code, r7 used by this macro

    mRETURNFNC

@ setBacklight
@
@ Description: Enables or disables the backlight.
@
@ Operational Description: The backlight is set on or off.
@
@ Arguments: r0 - TRUE if backlight should be enabled, otherwise backlight disabled.
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: None
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: Backlight of display
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: None
@
@ Limitations: None known
@
@ Registers Changed (besides r0-r3): None
@
@ Known Bugs: None known
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment              Date
@ Will Werst        Initial version     6/22/2017

setBacklight:
    mSTARTFNC
    CMP     r0, #TRUE           @Check which action to take
    BEQ     backlightOn         @if true, enable backlight
    BNE     backlightOff        @else, disable backlight
backlightOn:
    mSET_HREG   PIOA_SODR,  (1 << DISP_BCKLIGHT)
    B       endSetBacklight
backlightOff:
    mSET_HREG   PIOA_CODR,  (1 << DISP_BCKLIGHT)
    B       endSetBacklight
endSetBacklight:
    mRETURNFNC


@ displayHandler
@
@ Description: Handles sending commands to the display
@
@ Operational Description: This interrupt routine handles a state machine that
@                            updates the display. This state machine has two
@                            states: Commands, and Data. The different states
@                            are handled as follows:
@                            Commands: The ActiveCommandQueue is switched to the other queue.
```

```
The command to
@                                        switch to the next page of data is enqueued if possible.
@                                        If there is space for this command, then the state
transitions
@                                        to Data next, and the current page is updated to the
next page, else the state stays at Commands.
@                                        The display is transitioned to command mode (DISP_A0
pin is set to 0 in PIO). Then, the DMA controller
@                                        is set to transmit all commands in the now-inactive
command queue.
@                            Data: The DMA controller is setup to transmit the next page of data
from DispBuffer.
@
@
@
@ Arguments: None
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: ActiveCommandQueue[RW] - Read from to send commands, and toggled back and
forth between
@                                             queue 1 and queue 2.
@
@                   displayHandlerState[RW] - Checked to determine whether sending data or
commands.
@                   displayCurPage[RW] - Checked to determine which page in display buffer to
send next.
@                   displayBuffer[R] - Data is read from here to send to display
@
@ Global Variables: None
@
@ Inputs: None
@
@ Outputs: SPI to display
@
@ Error Handling: None
@
@ Algorithms: None
@
@ Data Structures: Queues
@
@ Limitations: None
@
@ Registers Changed: None
@
@ Known Bugs: None known
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment              Date
@ Will Werst        Initial version     6/22/2017

displayHandler:
    mSTARTINT
    mSET_HREG   SPI_PTCR,   SPI_DMA_DISABLE @Disable the DMA controller
    mSET_HREG   SPI_CR, SPI_CR_DIS
waitSPIDIS:
    mLOADTOREG  r0, SPI_SR
    TST r0, #0x10000
    BNE waitSPIDIS
    mLOADTOREG  r0, displayHandlerState
    CMP r0, #STATE_COMMANDS
    BEQ stateCommand
    CMP r0, #STATE_DATA
    BEQ stateData
    B   endDisplayHandler                     @Should never hit this state
stateCommand:
    LDR r0, =(NHD_COLL_PREF | 0x0)
    BL  queueDisplayCommand
    LDR r0, =(NHD_COLU_PREF | 0x0)
    BL  queueDisplayCommand
    mLOADTOREG  r0, displayCurPage            @Add the page address set command
    ORR r0, r0, #NHD_PAGE_PREF
    BL  queueDisplayCommand
    CMP r0, #TRUE                             @Check if command added
    LDREQ   r0, =STATE_DATA                   @If it was, transition states
    LDREQ   r1, =displayHandlerState          @Else, the state is left as STATE_COMMANDS
    STREQ   r0, [r1]
    LDR r0, =ActiveCommandQueue               @Load the address of activeCommandQueue pointer
    LDR r1, [r0]                              @Dereference activeCommandQueue to get
                                              @pointer to active queue's start.
    PUSH {r1}                                 @Save pointer to current active queue for later.
    LDR r2, =CommandQueue1                    @Load pointer to CommandQeueu1 to
                                              @compare to active command queue
    CMP r1, r2                                @Compare the queues
    LDREQ   r2, =CommandQueue2                @If active queue is 1, switch to queue 2.
                                              @Else, queue 2 is active and want to switch to
                                              queue 1
    STR     r2, [r0]                          @switch to the new queue

    @Setup DMA for sending commands
    mSET_HREG   PIOA_CODR,  (1 << DISP_A0)  @Clear A0 (sending commands)
    POP {r1}                                  @Get the queue of commands to send
    mSTOREFROMREG   r1, r0, SPI_TPR     @Set pointer to now inactive command queue
    mLOADTOREG  r1, CommandQueueSize          @Set count of bytes to send to command queue size
    mSTOREFROMREG   r1, r0, SPI_TCR

    LDR r1, =CommandQueueSize                 @clear command queue size since now using different
    queue
    LDR r0, =0
    STR r0, [r1]
    B   endDisplayHandler
stateData:

    @Setup DMA for sending data
    mSET_HREG   PIOA_SODR,  (1 << DISP_A0)  @Set A0 (sending data)
    mLOADTOREG r0, displayCurPage             @Calculate the pointer to the current
                                              @buffer page

    LDR r1, =NUM_COLS
    MUL r0, r0, r1
    LDR r1, =DispBuffer
    ADD r0, r0, r1
    mSTOREFROMREG   r0, r2, SPI_TPR           @Set pointer to display buffer
    LDR r1, =NUM_COLS
    mSTOREFROMREG   r1, r2, SPI_TCR           @Set length of DMA

    LDR r0, =STATE_COMMANDS                   @Transition states
    LDR r1, =displayHandlerState
    STR r0, [r1]
    mLOADTOREG r0, displayCurPage             @Calculate the pointer to the current
                                              @buffer page
    ADD r0, #1
    CMP r0, #NUM_PAGES
    LDREQ   r0, =0
    mSTOREFROMREG   r0, r1, displayCurPage
    @B endDisplayHandler
endDisplayHandler:
    mSET_HREG   SPI_CR, SPI_CR_EN
    mSET_HREG   SPI_PTCR,   SPI_DMA_ENABLE  @Enable the DMA controller
    mRETURNINT


.data
.balign 4

@Pointer to null-terminated ASCII string
displayHandlerState:
    .word STATE_COMMANDS
displayUpdated:
    .word TRUE
displayCurPage:
    .word 0

hexToASCII:                 @Table to map hex bytes to ASCII characters
    .byte   0x30
    .byte   0x31
    .byte   0x32
    .byte   0x33
    .byte   0x34
    .byte   0x35
    .byte   0x36
    .byte   0x37
    .byte   0x38
    .byte   0x39
    .byte   0x41
    .byte   0x42
    .byte   0x43
    .byte   0x44
    .byte   0x45
    .byte   0x46

.balign 4
TextMessageR1:
    .word   St_IDLE_MES
TextMessageR2:
    .skip   22              @Max 21 characters on display plus null termination
.balign 4
StatusMessages:
    .word   St_IDLE_MES
    .word   St_OFFHOOK_MES
    .word   St_RINGING_MES
    .word   St_CONNECTING_MES
    .word   St_CONNECTED_MES
    .word   St_SET_IP_MES
    .word   St_SET_SUBNET_MES
    .word   St_SET_GATEWAY_MES
    .word   St_MEM_SAVE_MES
    .word   St_MEM_RECALL_MES
    .word   St_RECALLED_MES
    .word   St_ILLEGAL_MES

St_IDLE_MES:
    .asciz  "Idle"
St_OFFHOOK_MES:
    .asciz  "Off Hook"
St_RINGING_MES:
    .asciz  "Ringing"
St_CONNECTING_MES:
    .asciz  "Connecting"
St_CONNECTED_MES:
    .asciz  "Connected"
St_SET_IP_MES:
    .asciz  "Set IP"
St_SET_SUBNET_MES:
    .asciz  "Set Subnet"
St_SET_GATEWAY_MES:
    .asciz  "Set Gateway"
St_MEM_SAVE_MES:
    .asciz  "Memory Save"
St_MEM_RECALL_MES:
    .asciz  "Memory Recall"
St_RECALLED_MES:
    .asciz  "Recalled Message"
St_ILLEGAL_MES:
    .asciz  "Illegal Message"


CommandQueueSize:
    .word 0x00000000
ActiveCommandQueue:
    .word CommandQueue1



CommandQueue1:
    .skip   (COM_QUEUE_LENGTH)
CommandQueue2:
    .skip   (COM_QUEUE_LENGTH)
DispBuffer:
    .skip (NUM_COLS*NUM_PAGES)
.balign 4
.end
```