```
@Done
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@
@ System.s
@ Contains code for system-level functions, like timers
@
@
@ Description: Currently only contains code for initializing the system
@              clock.
@
@
@ Table of Contents:
@   - init_system: Call to initialize the timers in the system.
@   - elapsed_time: Returns the number of milliseconds since the last
@                   elapsed_time call.
@   - timerHandler: Handles the timer callbacks, and calling the oneMillisElapsed
@                   function when appropriate
@   - oneMillisElapsed: Called every time 1 ms elapses. This handles
@                       incrementing the millis shared variable, and
@                       anything else that the system needs, like
@                       DRAM refresh.
@
@
@
@
@
@
@ Revision History:
@ Name          Comment                 Date
@ Will Werst    Initial version         Some lonely night around 6/10/17
@ Will Werst    Comment                 October 2017



.include    "at91rm9200.inc"
.include    "macro.inc"
.include    "system.inc"


.text
.arm

@ init_system
@
@ Description: Initializes the code in this file. Call before
@              using anything in this file.
@
@ Operational Description: A timer interrupt is setup, and then
@                          some status registers are read to clear them.
@
@ 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 init_system
init_system:
    mSTARTFNC
    mSET_HREG   AIC_SMR1,   0x00000021      @Initialize the system timer interrupt
    mSET_HREG AIC_SVR1, timerHandler        @Set callback
    mSET_HREG   ST_PIMR,    PIT_INTERVAL    @Set timer period
    mSET_HREG   ST_IER,     0x00000001      @Enable interrupt
    mSET_HREG AIC_IECR,     0x00000002      @Enable interrupt
    LDR     r0,     =ST_SR                  @Read status registers to clear them
    LDR     r0,     [r0]
    LDR     r0,     =RTC_SR                 @Read status registers to clear them
    LDR     r0,     [r0]
    LDR     r0,     =PMC_SR                 @Read status registers to clear them
    LDR     r0,     [r0]
    mRETURNFNC


@ elapsed_time
@
@ Description: Returns the number of milliseconds which have elapsed since the
@              last time this function was called.
@
@ Operational Description: The current millis count is loaded, and
@                          the last_millis count is loaded as well.
@                          The result is generated by subtracting last_millis
@                          from millis, and then last_millis is set to the current
@                          value of millis.
@
@ Arguments: None
@
@ Return values: r0 - number of milliseconds since the last time the function
@                     was called.
@
@ Local variables: None
@
@ Shared variables: millis[R] - the current millis count is read
@                   last_millis[RW] -  the previous millis count is read and updated
@
@ 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 known
@
@ Special notes: None
@
@ Revision History:
@ Name             Comment              Date
@ Will Werst        Initial version     Some lonely night around 6/10/17

@ returns: number of milliseconds since last function call
.global elapsed_time
elapsed_time:
    mSTARTFNC
    mLoadToReg  r0, millis              @load current milliseconds
    mLoadToReg  r1, last_millis         @load last milliseconds
    mStoreFromReg   r0, r2, last_millis @store current milliseconds into last milliseconds
    SUB     r0, r0, r1                  @get difference

    mRETURNFNC


@ timerHandler
@
@ Description: Handles incrementing the timer_counter.
@              This function is called from an interrupt.
@
@ Operational Description: The timer_counter variable is
@                   incremented by the number of milli-ticks of slow clock
@                   per interrupt, and then any full milliseconds are
@                   removed from timer_counter and the oneMillisElapsed
@                   function is called for each one of these full
@                   milliseconds.
@
@ Arguments: None
@
@ Return values: None
@
@ Local variables: None
@
@ Shared variables: timer_counter[RW] - This counter
@                       contains the number of milli-ticks
@                       of the slow clock. It is updated by method.
@
@ 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

@ variables:
@   r1 - timer_counter
timerHandler:
    mSTARTINT
    LDR     r0,     =ST_SR              @Read status register to clear it
    LDR     r0,     [r0]
    LDR     r0,     =RTC_SR             @Read RTC register to clear it
    LDR     r0,     [r0]
    LDR     r0,     =PMC_SR             @Read power management controller
    LDR     r0,     [r0]                @status register to clear it
    LDR     r0,     =(PIT_INTERVAL*MILLIS_IN_SEC) @Load number of milli-ticks
                                        @of slow clock between interrupts.
    mLoadToReg  r1, timer_counter       @And add the number of milli-ticks
    ADD     r1, r1, r0                  @to the current count of milli-ticks
transferFullMillisCounts:
    CMP     r1, #SLCK_CNT_SEC           @Check to see if a millisecond has elapsed
    BLO     endTimerHandler             @No full milliseconds to transfer
    PUSH    {r1}
    BL      oneMillisElapsed            @Call the function to increment millis
                                        @and also take any action that should
                                        @be done when a millisecond has elapsed
    POP     {r1}
    SUB     r1, #SLCK_CNT_SEC           @Take off the number of counts in timer_counter
                                        @associated with one millisecond elapsing
    B       transferFullMillisCounts    @Go back and check if more milliseconds to transfer
endTimerHandler:
    mStoreFromReg   r1, r0, timer_counter @Save the remainder milli-ticks
    mRETURNINT


@ oneMillisElapsed
@
@ Description: This function should be called by the timerHandler
@              every time one millisecond elapses.
@
@ Operational Description: The millis variable is incremented, and then
@                          any other actions that should be taken every
@                          millisecond are done. Currently this is only
@                          a DRAM refresh.
@
@ 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


@ oneMillisElapsed
@ Description: Function used to increment millisecond, and also do any other
@               actions such as refresh DRAM
@

oneMillisElapsed:
    mSTARTFNC
    mLoadToReg  r0, millis          @Load millis
    ADD     r0, #1                  @Increment millis
    mStoreFromReg   r0, r2, millis  @save millis

    @Refresh DRAM, need to refresh 1024 rows every 16 ms
    LDR     r0,     =((DRAM_NUM_ROWS / DRAM_MILLIS_PER_REF)) @Figure out how many rows to refresh
    LDR     r1,     =DRAM_START     @Load start address of DRAM
    mLoadToReg  r2,     dramRefreshRow  @Load the row that was refreshed last
dramRefresh:
    LDRB    r3, [r1, r2]            @Refresh row
    ADD     r2,     #1              @Go to next row
    CMP     r2,     #DRAM_NUM_ROWS  @Check if gone through all rows
    LDRHS   r2,     =0              @If so, go back to row 0
    SUBS    r0,     #1              @Decrement row counter
    BHI     dramRefresh             @If still rows to refresh, continue refreshing
    @B      endDRAMRefresh          @Done refreshing

endDRAMRefresh:
    mStoreFromReg   r2, r0, dramRefreshRow @Save row that was worked on
    @B  endOneMillisElapsed
endOneMillisElapsed:
    mRETURNFNC                      @Done, return



.data

millis:                             @Stores milliseconds elapsed since start
    .word 0x00000000
last_millis:                        @Stores milliseconds since last elapsed_time call
    .word 0x00000000
timer_counter:                      @Counter for converting slow clock ticks to milliseconds
    .word 0x00000000
dramRefreshRow:                     @Counter for which row to refresh next
    .word 0x00000000

.end
```