Review

- Assembler directives
  - Some assembly doesn’t generate instructions
- APB: wait states and errors
- Volatile keyword
  - Important for correctness
  - Easy to miss
- Pointers and function pointers
- Vector tables
- Genericity
  - Often dangerous (void *)
- Weak references
  - Call only if appropriate function linked in
- Interrupts: a little more today
Outline

- Interrupts review
- Timers
Timer tie-in: higher-level APIs atop interrupts

- Interrupt vector / jump table used to indicate ISR for each interrupt.
- Callback
  - Similar concept at a higher level.
  - Pass a function pointer (a callback) into another function.
  - Function registers callback for later execution.
  - E.g., pass function to execute at a particular time.
- int execute_when(void (*callback)(int when), int when);
Sharing data with ISR

• What if an ISR/program shared data structure requires multiple instructions to modify?
• E.g., deleting an element from a linked list.
• Program
  Get pointer to relevant list element.
  [What if interrupt happens here?]
  Read and write data in list element.
• ISR
  Delete list element.
Sharing data with ISR

- Solution: make atomic operations atomic.
- Disable interrupts.
  - Just those that care about inconsistent state.
  - Keep short.
Debugging ISRs

- Set a breakpoint at interrupt handler.
- Is it ever called?

- Examine NVIC registers.
- Are they set correctly?

- Use oscilloscope to look at interrupt signal.

- Default interrupt vector table traps to infinite loop.
Outline

- Interrupts review
- Timers
Timers

- Why they matter.
- Avoid pitfalls of loop-based delays.
  - Waste power.
  - Prevent other useful work from being done.
- Why they are complex?
  - Span HW/SW boundary.
iPhone Clock App

- World Clock – display real time in multiple time zones
- Alarm – alarm at certain (later) time(s).
- Stopwatch – measure elapsed time of an event.
- Timer – count down time and notify when count becomes zero.
Motor and light Control

- Servo motors – PWM signal provides control signal.
- DC motors – PWM signals control power delivery.
- RGB LEDs – PWM signals allow dimming through current-mode control.
Pulse width modulation (PWM)

- Given fixed period oscillating signal,
- vary high % to transmit analog value.
- Show using pen and paper.
Methods from Android SystemClock

<table>
<thead>
<tr>
<th>Public Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long</td>
<td><strong>currentThreadTimeMillis()</strong>&lt;br&gt;Returns milliseconds running in the current thread.</td>
</tr>
<tr>
<td>static long</td>
<td><strong>elapsedRealtime()</strong>&lt;br&gt;Returns milliseconds since boot, including time spent in sleep.</td>
</tr>
<tr>
<td>static long</td>
<td><strong>elapsedRealtimeNanos()</strong>&lt;br&gt;Returns nanoseconds since boot, including time spent in sleep.</td>
</tr>
<tr>
<td>static boolean</td>
<td><strong>setCurrentTimeMillis</strong>&lt;br&gt;Sets the current wall time, in milliseconds.</td>
</tr>
<tr>
<td>static void</td>
<td><strong>sleep</strong>&lt;br&gt;Waits a given number of milliseconds (of uptimeMillis) before returning.</td>
</tr>
<tr>
<td>static long</td>
<td><strong>uptimeMillis()</strong>&lt;br&gt;Returns milliseconds since boot, not counting time spent in deep sleep.</td>
</tr>
</tbody>
</table>
# Standard C library's `<time.h>` header file

## Library Functions

Following are the functions defined in the header file `time.h`:

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Function &amp; Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>char *asctime(const struct tm *timeptr)</code>&lt;br&gt;Returns a pointer to a string which represents the day and time of the structure timeptr.</td>
</tr>
<tr>
<td>2</td>
<td><code>clock_t clock(void)</code>&lt;br&gt;Returns the processor clock time used since the beginning of an implementation-defined era (normally the beginning of the program).</td>
</tr>
<tr>
<td>3</td>
<td><code>char *ctime(const time_t *timer)</code>&lt;br&gt;Returns a string representing the localtime based on the argument timer.</td>
</tr>
<tr>
<td>4</td>
<td><code>double difftime(time_t time1, time_t time2)</code>&lt;br&gt;Returns the difference of seconds between time1 and time2 (time1-time2).</td>
</tr>
<tr>
<td>5</td>
<td><code>struct tm *gmtime(const time_t *timer)</code>&lt;br&gt;The value of timer is broken up into the structure tm and expressed in Coordinated Universal Time (UTC) also known as Greenwich Mean Time (GMT).</td>
</tr>
<tr>
<td>6</td>
<td><code>struct tm *localtime(const time_t *timer)</code>&lt;br&gt;The value of timer is broken up into the structure tm and expressed in the local time zone.</td>
</tr>
<tr>
<td>7</td>
<td><code>time_t mktime(struct tm *timeptr)</code>&lt;br&gt;Converts the structure pointed to by timeptr into a time_t value according to the local time zone.</td>
</tr>
<tr>
<td>8</td>
<td><code>size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr)</code>&lt;br&gt;Formats the time represented in the structure timeptr according to the formatting rules defined in format and stored into str.</td>
</tr>
<tr>
<td>9</td>
<td><code>time_t time(time_t *timer)</code>&lt;br&gt;Calculates the current calender time and encodes it into time_t format.</td>
</tr>
</tbody>
</table>
Standard C library’s `<time.h>` header file: struct tm

```c
struct tm {
    int tm_sec;    /* seconds, range 0 to 59 */
    int tm_min;    /* minutes, range 0 to 59 */
    int tm_hour;   /* hours, range 0 to 23 */
    int tm_mday;   /* day of the month, range 1 to 31 */
    int tm_mon;    /* month, range 0 to 11 */
    int tm_year;   /* The number of years since 1900 */
    int tm_wday;   /* day of the week, range 0 to 6 */
    int tm_yday;   /* day in the year, range 0 to 365 */
    int tm_isdst;  /* daylight saving time */
};
```
Anatomy of a timer system

Application Software

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Applications

Operating System

Software

Hardware

Compare → Counter → Capture

Prescaler

Clock Driver

Xtal/Osc

I/O

Internal

External

Applications

Input

Output

Hardware

Software

Internal

External

I/O

module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;
  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
Anatomy of a timer system

Application Software

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Compare

Counter

Capture

Prescaler

Clock Driver

Xtal/Osc

Applications

Operating System

Software

Hardware

I/O

I/O

R/W

R/W

R/W

typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;

module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;
  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule

timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);

timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
Timer requirements

- Wall clock date & time
  - Date: Month, Day, Year
  - Time: HH:MM:SS:mmm
  - Provided by a “real-time clock” or RTC
- Alarm: do something (call code) at certain time later
  - Later could be a delay from now (e.g., Δt)
  - Later could be actual time (e.g., today at 3pm)
- Stopwatch: measure (elapsed) time of an event
  - Instead of pushbuttons, could be function calls or
  - Hardware signals outside the processor
Timer requirements

- Wall clock
  - `datetime_t getDateTime()`
- Alarm
  - `void alarm(callback, delta)`
  - `void alarm(callback, datetime_t)`
- Stopwatch: measure (elapsed) time of an event
  - `t1 = now(); ... ; t2 = now(); dt = difftime(t2, t1);`
- `GPIO_INT_ISR:`
  - `LDR R1, [R0, #0]`  \( R0=\text{timer address} \)
Wall Clock from a Real-Time Clock (RTC)

- Often a separate module
- Built with registers for
  - Years, Months, Days
  - Hours, Mins, Seconds
- Alarms: hour, min, day
- Accessed via
  - Memory-mapped I/O
  - Serial bus (I2C, SPI)
Timer requirements

• Wall clock
  • `datetime_t getDateTime()`

• Alarm
  • `void alarm(callback, delta)`
  • `void alarm(callback, datetime_t)`

• Stopwatch: measure (elapsed) time of an event
  • `t1 = now(); ... ; t2 = now(); dt = difftime(t2, t1);`

• GPIO_INT_ISR:
  • `LDR R1, [R0, #0]`  % R0=timer address
Anatomy of a timer system

Applications

Operating System

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Software

Hardware

Compare

Counter

Capture

Prescaler

Clock Driver

Xtal/Osc

Applications

Operating System

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Software

Hardware

Compare

Counter

Capture

Prescaler

Clock Driver

Xtal/Osc

typedef struct timer {
    timer_handler_t handler;
    uint32_t time;
    uint8_t mode;
    timer_t* next_timer;
} timer_t;

timer_tick:
    ldr r0, count;
    add r0, r0, #1
    ...

module timer(clr, ena, clk, alrm);
    input clr, ena, clk;
    output alrm;
    reg alrm;
    reg [3:0] count;

    always @(posedge clk) begin
        alrm <= 0;
        if (clr) count <= 0;
        else count <= count+1;
    end
endmodule
Oscillators – RC
Oscillators – Crystal

![Figure 1: Fundamental Mode Isolated Pierce-Gate Oscillator](image)
Anatomy of a timer system

Applications
Operating System

Timer Abstractions and Virtualization

Low-Level Timer Subsystem Device Drivers

Application Software

Compare
Counter
Capture

Prescaler
Clock Driver

Xtal/Osc

Software
Hardware

I/O

module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;
  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule

timer_t timerX;
initTimer();
... startTimerOneShot(timerX, 1024);
... stopTimer(timerX);

typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
timer_tick:
  ldr r0, count;
  add r0, r0, #1
...
Timer requirements

- Wall clock
  - `datetime_t getDateTime()`
- Alarm
  - `void alarm(callback, delta)`
  - `void alarm(callback, datetime_t)`
- Stopwatch: measure (elapsed) time of an event
  - `t1 = now();`
  - `{slow code} ;`
  - `t2 = now();`
  - `dt = difftime(t2, t1);`
  - `GPIO_INT_ISR:`
    - `LDR R1, [R0, #0] % R0=timer address`
There are two basic activities one wants timers for:

• Measure how long something takes
  – “Capture”

• Have something happen once or every X time period
  – “Compare”
Example # 1: Capture

• Fan
  • Measure spin speed.
  • Option 1
    • Interrupt every rotation.
    • Slow to process interrupt.
    • Need to determine time within it.
  • Option 2
    • Have timer store interval.
    • Restart self.
    • Generate interrupt.
• Relevant to ABS.
Example #2: Compare

- Driving a DC motor.
  - Motors turn at a speed determined by current.
  - Doing this in analog can be hard.
    - Need analog output.
    - Linear amplification (op-amp?).
  - PWM easier.
    - Linearity unimportant (FET or BJT).
    - Control duty cycle.
    - Make sure frequency is high enough.
    - Can even make analog after FET.
Servo motor control: class exercise

- Assume 1 MHz clock.
- Design “high-level” circuit to
  - Generate 1.52 ms pulse
  - Every 6 ms
  - Repeat
- How would we generalize this?
Virtual timers

- What if more timers needed?
- Use HW timers as a foundation for SW timers.
- List events.
- Set timer for earliest one.
- Repeat.
Problems?

- Only works for “compare” timer uses.
- Slows timer ISR.
Implementation Issues

- Shared user-space/ISR data structure.
  - Insertion can be in user code.
  - Deletion happens in ISR.
    - We need critical section (disable interrupt)
- One-shot and repeating useful.
- Simultaneous events.
  - Pick an order, do both.
Implementation Issues (continued)

• What data structure?
  - Data needs be sorted.
    • Inserting one thing at a time.
  - Always pop from same end.
    • Fast.
  - Add in sorted order.
typedef struct timer {
    timer_handler_t handler;
    uint32_t time;
    uint8_t mode;
    timer_t* next_timer;
} timer_t;

timer_t* current_timer;

void initTimer() {
    setupHardwareTimer();
    initLinkedList();
    current_timer = NULL;
}

error_t startTimerOneShot(timer_handler_t handler, uint32_t t) {
    // add handler to linked list and sort it by time
    // if this is first element, start hardware timer
}

error_t startTimerContinuous(timer_handler_t handler, uint32_t dt) {
    // add handler to linked list for (now+dt), set mode to continuous
    // if this is first element, start hardware timer
}

error_t stopTimer(timer_handler_t handler) {
    // find element for handler and remove it from list
}
Timer

Program

//Setup Timer

Hardware Timer

Count: -1

Interrupt

ISR

//Do Something
Virtual Timer

Virtual Event Queue

Virtual Timer Code

//Update Event Queue
//Adjust Timer if needed

Hardware Timer

Count: -1

Interrupt

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Virtual ISR

//Figure out source
bl
bl
bl
bl

//Insert new event?
//Set new time

Program 1-Handler
//Do Something

Program 2-Handler
//Do Something

Program 3-Handler
//Do Something

Program 4-Handler
//Do Something

Program 1-Handler
//Do Something

Program 2-Handler
//Do Something

Program 3-Handler
//Do Something

Program 4-Handler
//Do Something
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr
Null

Hardware Timer
Count: -1
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr
Time: 3
Delta: -1
Mode: One

Hardware Timer
Count: 3
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

Time: 3
Delta: -1
Mode: One

Hardware Timer
Count: 3
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr
Time: 3
Delta: -1
Mode: One

Hardware Timer
Count: 3

Time: 5
Delta: -1
Mode: One

Time: 2
Delta: 2
Mode: Delta

Hdlr Ptr
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Event Queue

- **Program 1**
  - //Setup Timer @3
  - //Setup Timer @5

- **Program 2**
  - //Setup Timer @D2

- **Program 3**
  - //Setup Timer @D7

- **Program 4**
  - //Setup Timer @4

- **Hardware Timer**
  - Count: 0
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Head Ptr

Hardware Timer
Count: 0

Virtual ISR
//Figure out source
//Remove head
//Insert new event?
//Set new time

Time: 1
Delta: -1
Mode: One

Time: 2
Delta: -1
Mode: One

Time: 2
Delta: 2
Mode: Delta

Time: 3
Delta: -1
Mode: One

Time: 5
Delta: 7
Mode: Delta
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4
Event Queue - Caveats

- No new event added for one-shot
- Handler may need to loop to handle multiple events at same time
Caveats

• Previous slides assumed scheduling of events all when timer was first set.

• What if we need to schedule an event and we have already expired some of the timer?
  • Need to update the entire virtual time queue before inserting new event.
Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Event Queue

Some of clock already expired, so need to update queue first
---Subtract difference from all elements in list (2 in this case)
Event Queue

Program 1
//Setup Timer @3
//Setup Timer @5

Program 2
//Setup Timer @D2

Program 3
//Setup Timer @D7

Program 4
//Setup Timer @4

Hardware Timer
Count: 1
Done.