Lecture 07
Driver Design

CE346 – Microprocessor System Design

Branden Ghena – Fall 2021

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley)

Administriva

- Project Proposals due today!
  - A few are in so far and they look great and I’m super excited!!!!!

Today’s Goals

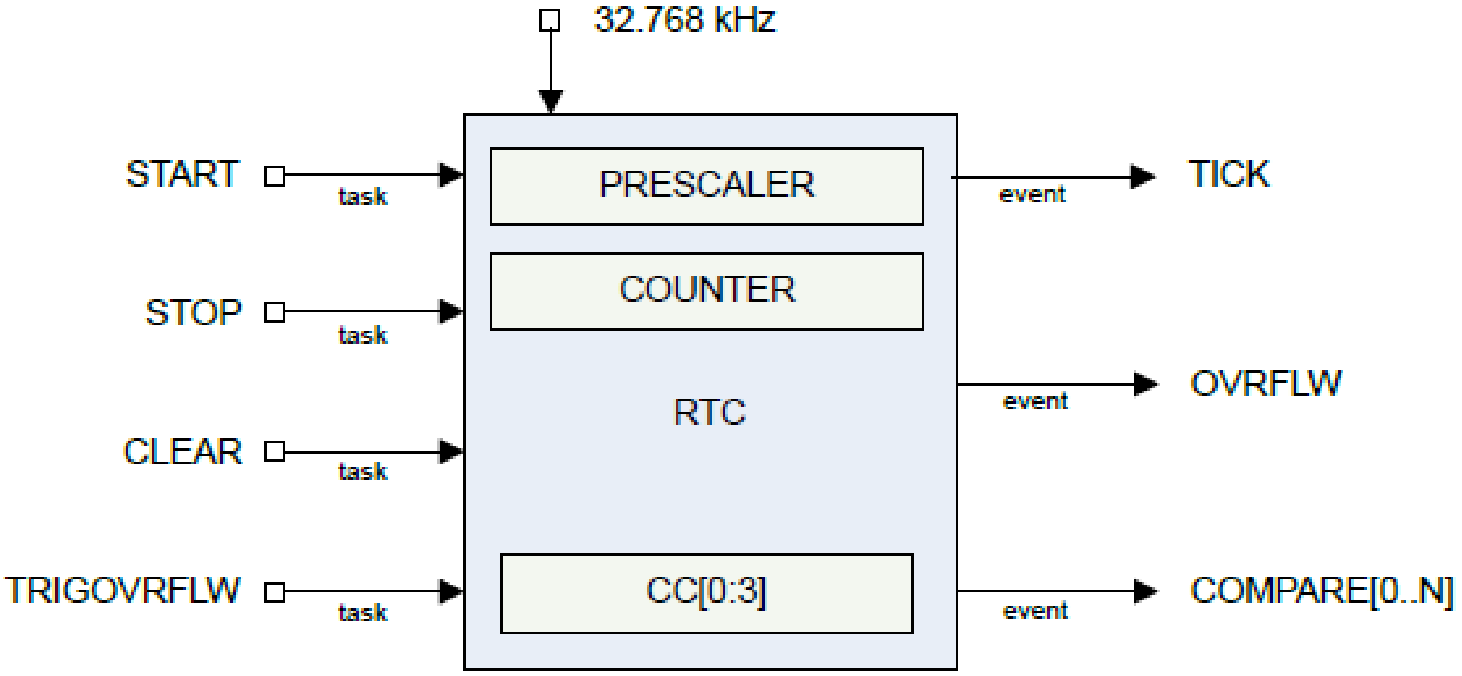
- Finish up Timer-like peripherals
- Explore another aspect of device driver design
  - Non-blocking vs Blocking interfaces
- Discuss how interrupts interact with these
  - Event-loop as a partial alternative

Outline

- **Real-time Counter**
- Watchdog Timer
- Driver Interfaces (Blocking and Non-Blocking)
- Event Loop

Real-time Counter

- Low-power (32 kHz) version of Timer
  - Only a 24-bit internal Counter



- Note: abbreviated RTC, but that already means something else (Real-Time Clock)

Differences between Real-Time Counter and Timer

- Runs off of LFCLK instead of HFCLK
  - With smaller prescaler value (4096 vs 32768)
- 24-bit counter vs 32-bit counter for Timer
- Can read the Counter value directly
  - No need for Capture task
- Otherwise extremely similar. Just a low-power version of Timer

Time resolution for Real-Time Counter

$$f_{\text{TIMER}} = \frac{32\ \text{KHz}}{Prescaler+1}$$

- Resolution
  - Minimum: 30.517 µs, overflows in 512 seconds (24-bit Counter)
  - Maximum: 125 ms, overflows in 582 hours

- Not as precise as the Timer (62.5 ns best precision)
  - Possible design: use both
    - Real-Time Counter for most of the waiting
    - Chained into Timer for precise remaining amount of time

Outline

- Real-time Counter
- **Watchdog Timer**
- Driver Interfaces (Blocking and Non-Blocking)
- Event Loop

Reliable systems

- What’s the most common way to solve computer problems?
  - Turn it off and turn it on again.

  - **Why?**

Reliable systems

- What’s the most common way to solve computer problems?
  - Turn it off and turn it on again.

  - **Why?**

  - Resets “state” to original values, which are likely good
    - Startup is often well-tested

    - It’s long-running code interacting in unexpected ways that leaves systems in a broken state

Watchdog timer (WDT)

- Focused on failures where the system “hangs” forever
  - Maybe software, maybe hardware!

- Can’t know for certain the system is hung, but can know practically
  - Select a timeout that is the maximum amount of time you expect the system to ever go without looping in main()
  - Multiply it by 2-10
  - Set a watchdog timer to that value

- If watchdog timer ever expires, it resets the system (in hardware)

Watchdog configuration

$$\text{timeout (seconds)} = \frac{Counter\ Reload\ Value + 1}{32768}$$

- Configure watchdog
    - Can choose whether to count down during Sleep mode or Debug mode
- Set a Counter Reload Value (CRV, 32-bits)
- Start the watchdog timer
    - Loads internal Counter to CRV value
    - Starts counting down at 32 kHz

Running applications with a watchdog timer

- Need to periodically reset the watchdog to keep it from expiring
  - Known as “feeding” the watchdog or “kicking” the watchdog

- Reload Request register
  - Must write sequence 0x6E524635 to reload watchdog
  - Incredibly unlikely to happen by accident

- While running, watchdog is protected from modification
  - Configure once, run forever (at least until a reboot)
  - Only option is to make periodic Reload Requests

- Default off on the nRF52833

Break + Open Question

- MSP430 microcontrollers start with the watchdog on by default

- What are the pros and cons of this choice?

Outline

- Real-time Counter
- Watchdog Timer
- **Driver Interfaces (Blocking and Non-Blocking)**
- Event Loop

Callback functions

- `timer_start(duration, my_timer_handler, context);`

- Driver interfaces often provide a callback mechanism
  - Caller provides a function which should be executed when complete

- "Context" is often provided as well (void*)
  - Ability for caller to pass an argument for the callback function
  - Often a pointer to a position in a structure or a shared variable to modify

Function pointers in C

- Harder than in Javascript or C++. Can’t define anonymous function inline
  - Instead create a pointer to an existing function in your code

```
void myfun (int a) {
    // do something here
}

void main() {
    void (*fun_ptr)(int) = &myfun;
    fun_ptr(10); // dereference happens automatically
}
```

Callbacks usually run in an interrupt mode

- If the interrupt handler calls the callback, the callback will be within that same interrupt mode

- Be careful which variables you modify!!
  - Same concurrency problems mentioned before

- Starts to get pretty annoying
  - Embedded systems deal with concurrency issues just like OS

Blocking function calls

- Alternative option: blocking calls
  - Do not return until request is complete

```
void myfun (void* context) {
    *(boolean*)context = true; // context is the flag pointer
}

void timer_start_blocking(duration) {
    boolean flag = false;
    //         duration, pointer, context
    timer_start(duration, &myfun, &flag);
    while (!flag) { }
}
```

Temp driver example

nu-microbit-base/software/apps/temp_driver/

Outline

- Real-time Counter
- Watchdog Timer
- Driver Interfaces (Blocking and Non-Blocking)
- **Event Loop**

Interrupts are frustrating

- We do not want to block on every call
- We also do not want to deal with concurrency issues
- Alternative: one main event loop
  - Polls necessary sensors
  - Iterates through state machine and determine actions
  - Runs at a certain frequency

Event loop

- Rather than polling a single driver, poll all of them
  - Each time through the loop check all relevant inputs
  - Respond to events that are necessary
  - Sleep until ready to start again

```
while (1) {
    time start = get_time();
    boolean result = check_timer();
    if (result) { check_gps(); }
    adjust_throttle();
    sleep(1ms – (get_time() – start));
}
```

Top-half / Bottom-half handler design

- Top half
  - Implements interface that higher layers require
  - Performs logic to start device requests
  - Wait for I/O to be completed
    - Synchronously (blocking) or asynchronously (return to event loop)
  - Handle responses from the device when complete

- Bottom half
  - Interrupt handler
    - Continues next transaction
    - Or signals for top half to continue (often with shared variable)

Temperature event-loop example

nu-microbit-base/software/apps/temp_event_loop/

Outline

- Real-time Counter
- Watchdog Timer
- Driver Interfaces (Blocking and Non-Blocking)
- Event Loop