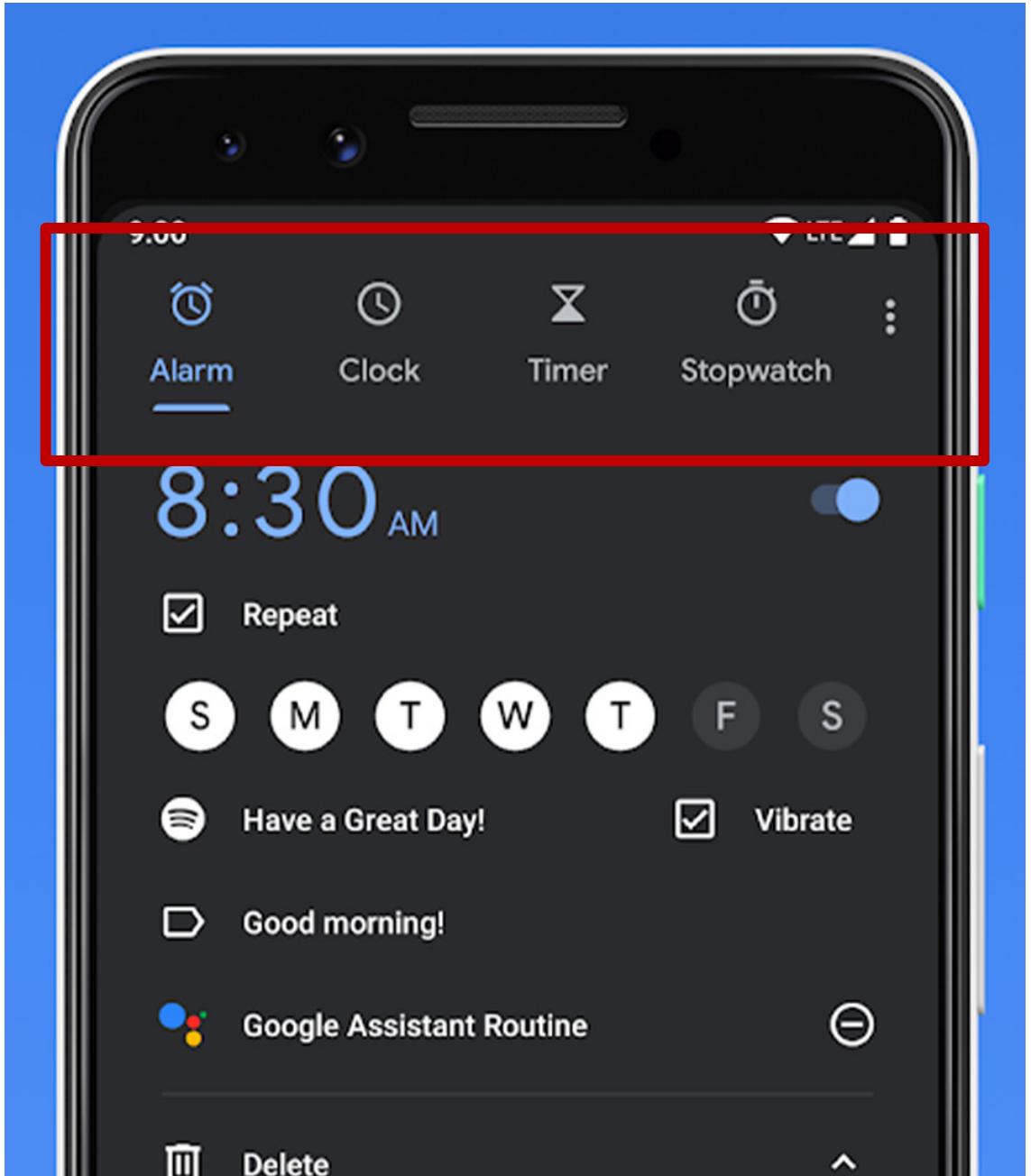
Clocks, Counters & Timers

Android Clock App



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

Motor/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

Methods from android.os.SystemClock

Public Methods	
static long	currentThreadTimeMillis () Returns milliseconds running in the current thread.
static long	elapsedRealtime () Returns milliseconds since boot, including time spent in sleep.
static long	elapsedRealtimeNanos () Returns nanoseconds since boot, including time spent in sleep.
static boolean	setCurrentTimeMillis (long millis) Sets the current wall time, in milliseconds.
static void	sleep (long ms) Waits a given number of milliseconds (of uptimeMillis) before returning.
static long	uptimeMillis () Returns milliseconds since boot, not counting time spent in deep sleep.

Standard C library's <time.h> header file

Library Functions

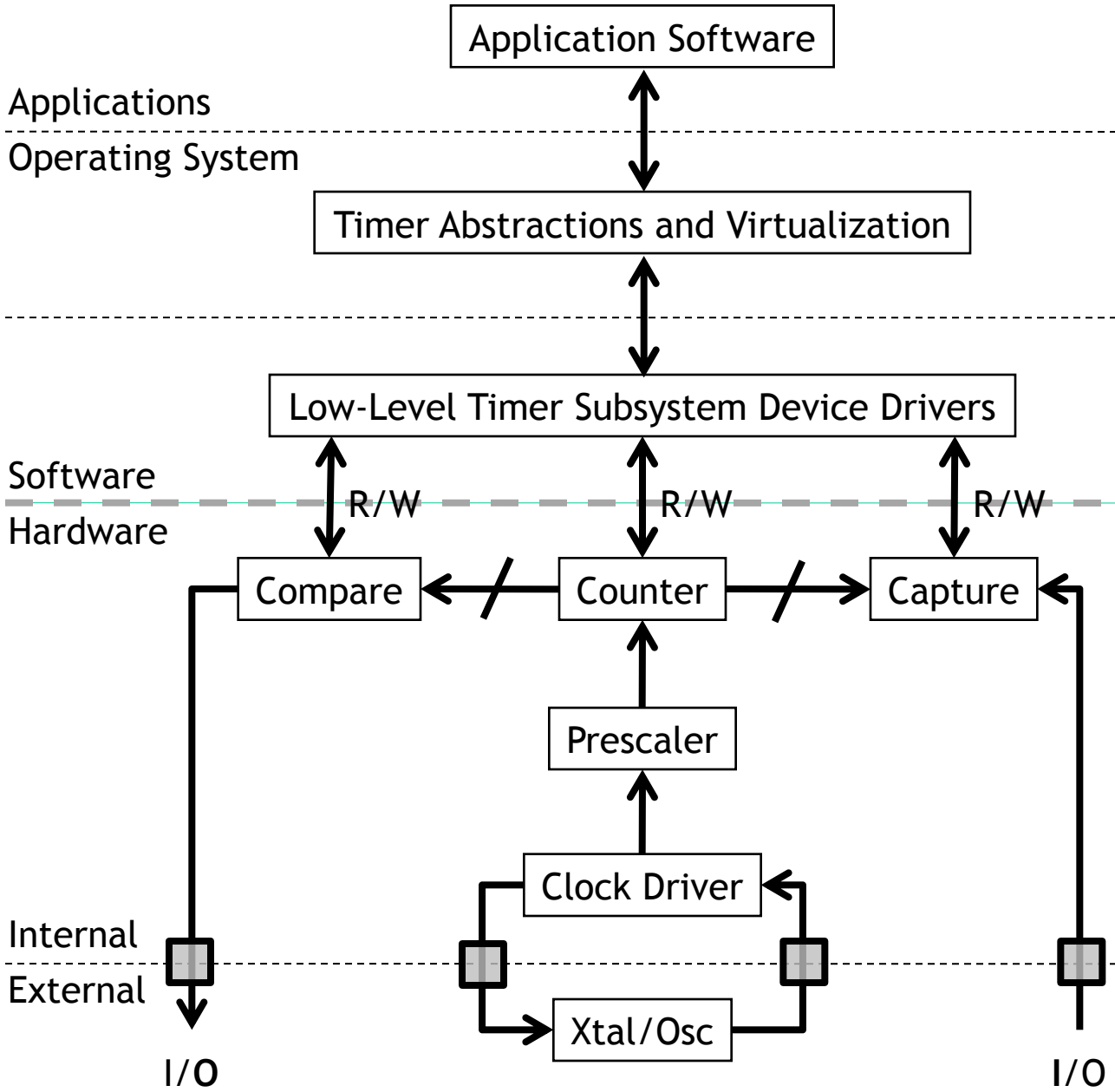
Following are the functions defined in the header time.h:

S.N.	Function & Description
1	char *asctime(const struct tm *timeptr) Returns a pointer to a string which represents the day and time of the structure timeptr.
2	clock_t clock(void) Returns the processor clock time used since the beginning of an implementation-defined era (normally the beginning of the program).
3	char *ctime(const time_t *timer) Returns a string representing the localtime based on the argument timer.
4	double difftime(time_t time1, time_t time2) Returns the difference of seconds between time1 and time2 (time1-time2).
5	struct tm *gmtime(const time_t *timer) 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).
6	struct tm *localtime(const time_t *timer) The value of timer is broken up into the structure tm and expressed in the local time zone.
7	time_t mktime(struct tm *timeptr) Converts the structure pointed to by timeptr into a time_t value according to the local time zone.
8	size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr) Formats the time represented in the structure timeptr according to the formatting rules defined in format and stored into str.
9	time_t time(time_t *timer) Calculates the current calender time and encodes it into time_t format.

Standard C library's <time.h> header file: struct tm

```
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



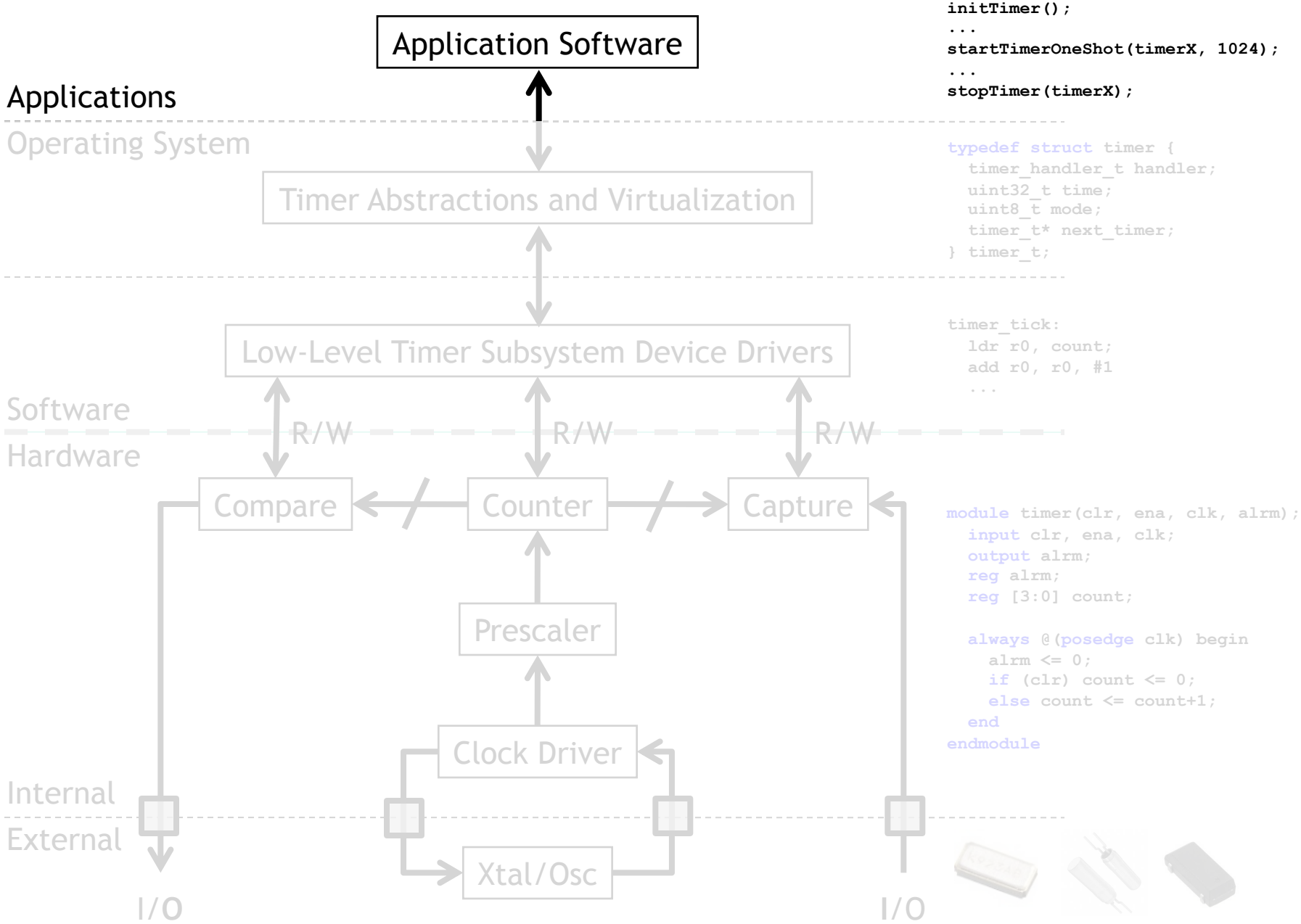

```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
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
  ...
```

```
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




```
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
  ...
```

```
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
```

What do we really want from our timing subsystem?

- 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 – count down time and notify when count = 0
  - Could invoke some code (e.g. a handler)
  - Could take some action (e.g. set/clear an I/O line)

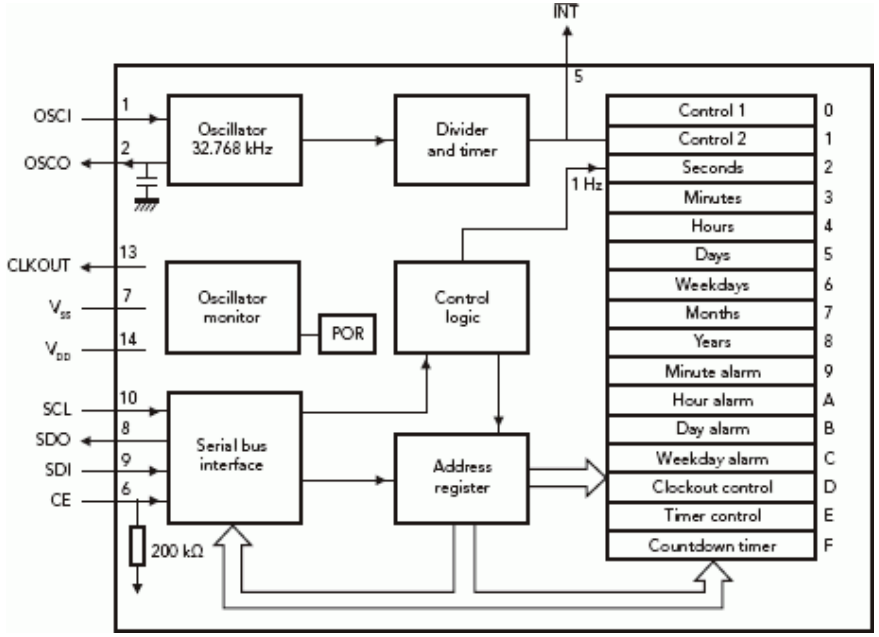
What do we really want from our timing subsystem?

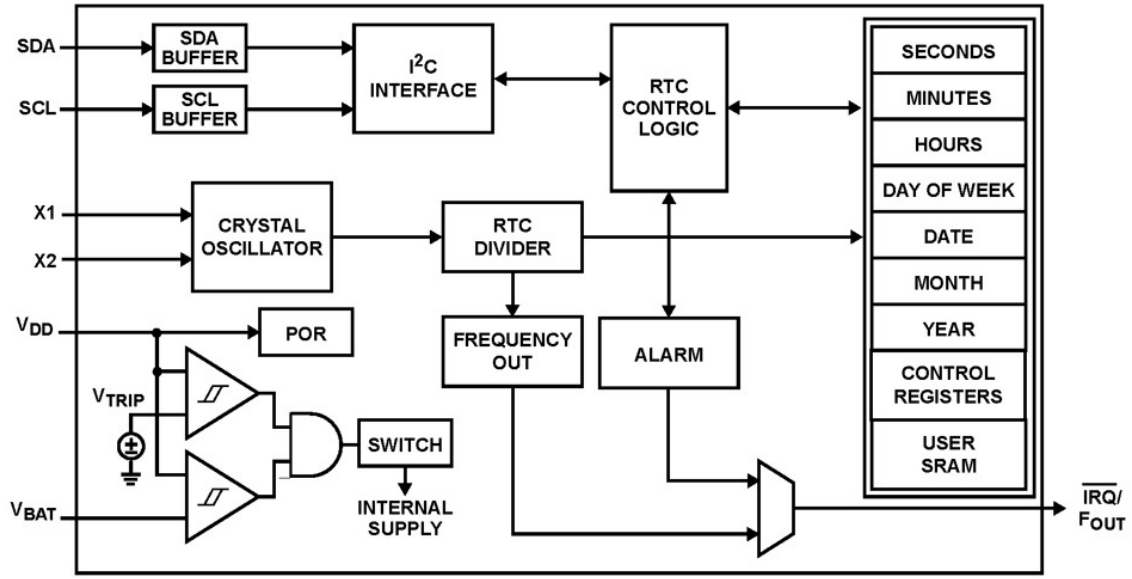
- Wall clock
  - datetime_t getDateTime()
- Alarm
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: measure (elapsed) time of an event
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

```
LDR R1, [R0, #0]          % R0=timer address
```

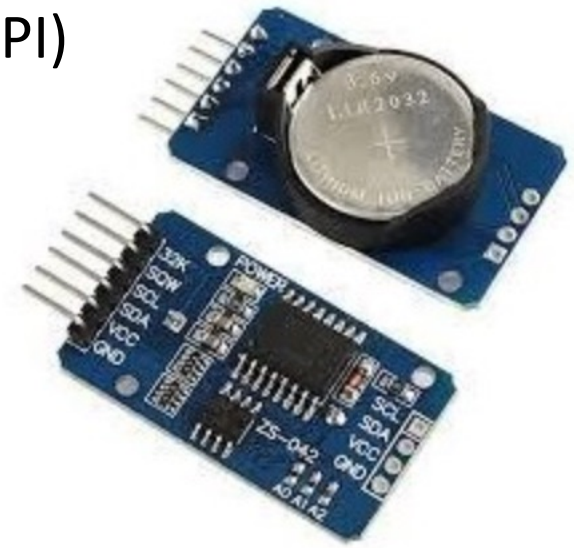
- Timer – count down time and notify when count = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

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)



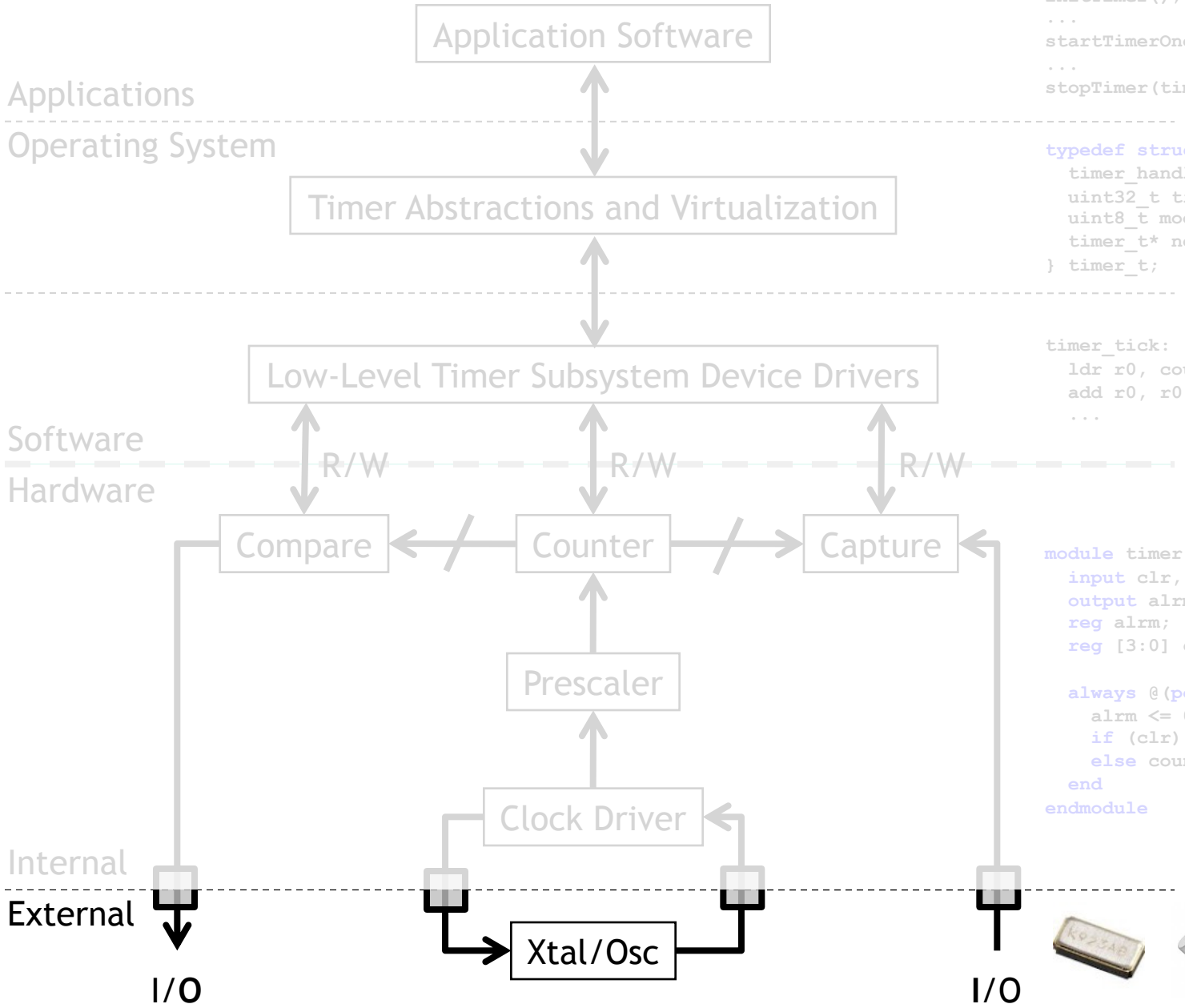
What do we really want from our timing subsystem?

- 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
```

- Timer – count down time and notify when count = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

Anatomy of a timer system




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

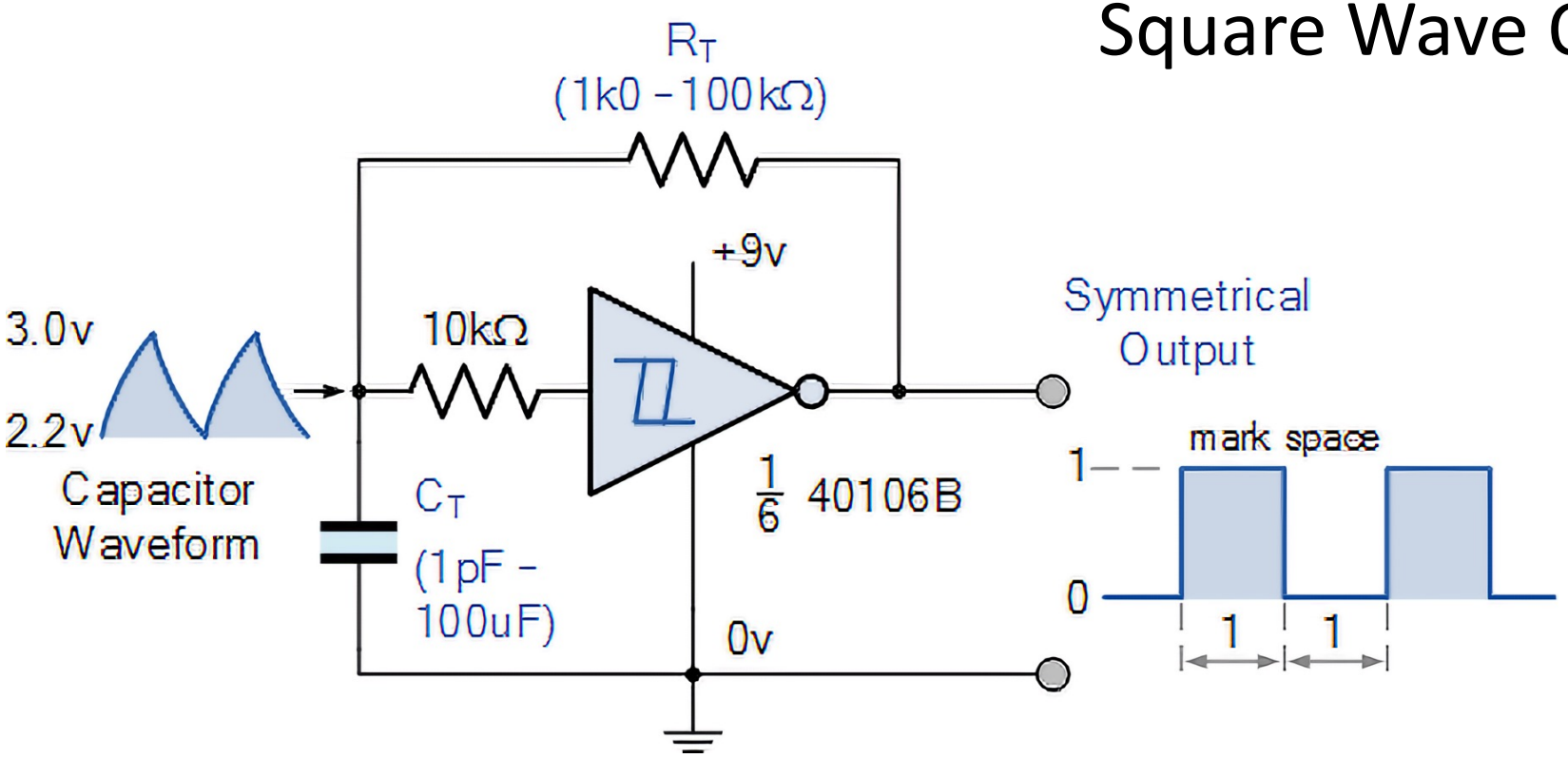
```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

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

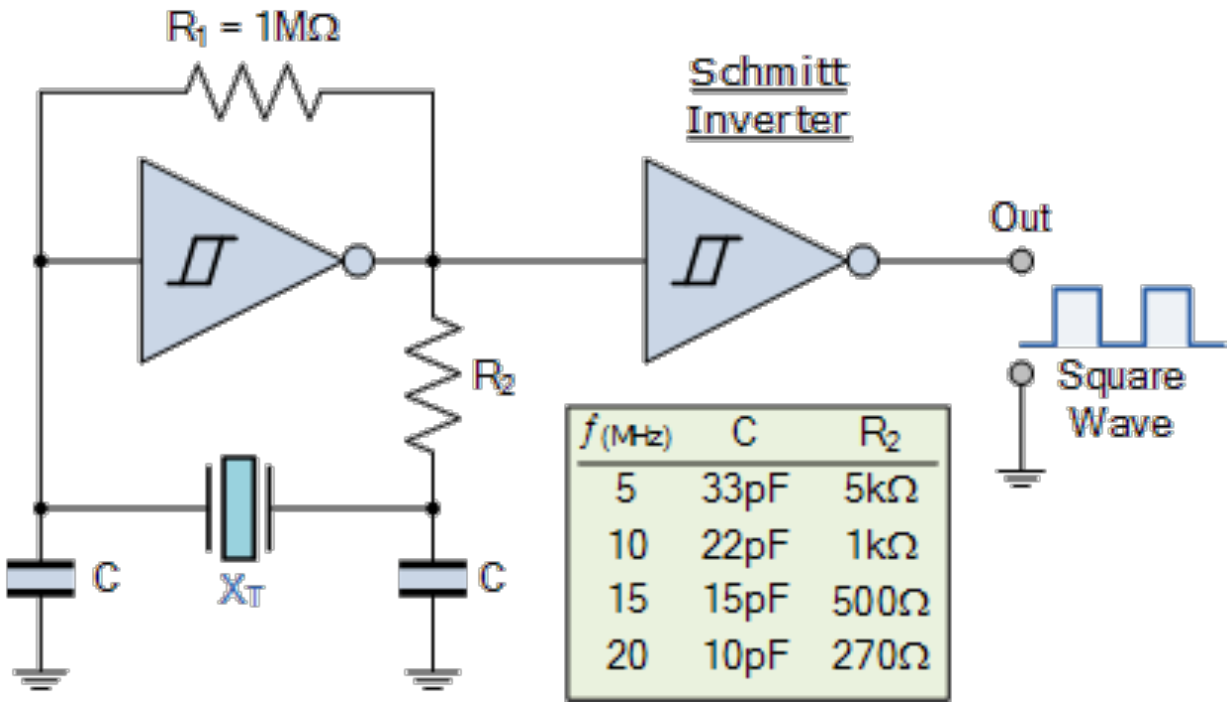
```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

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

Oscillators - RC

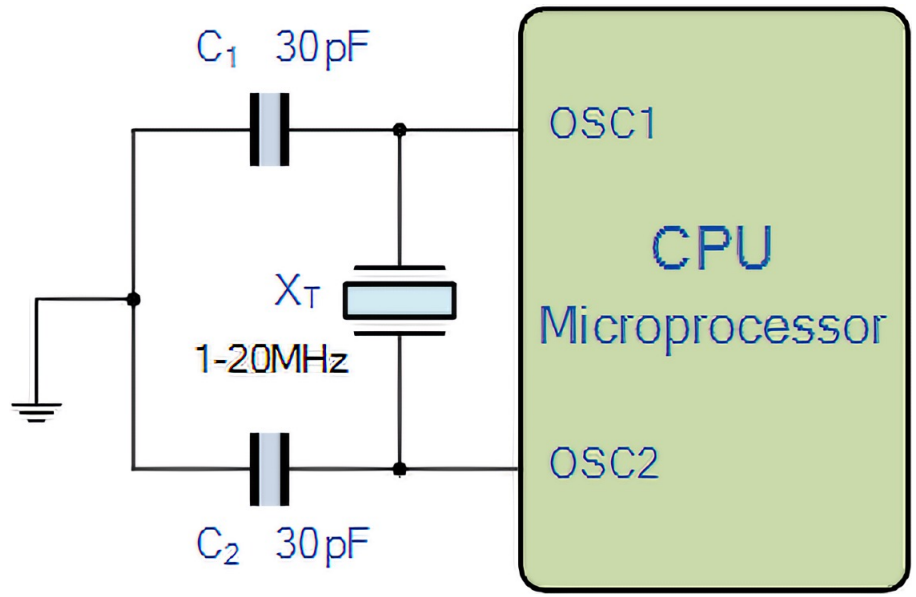
Square Wave Oscillator



Oscillators - Crystal

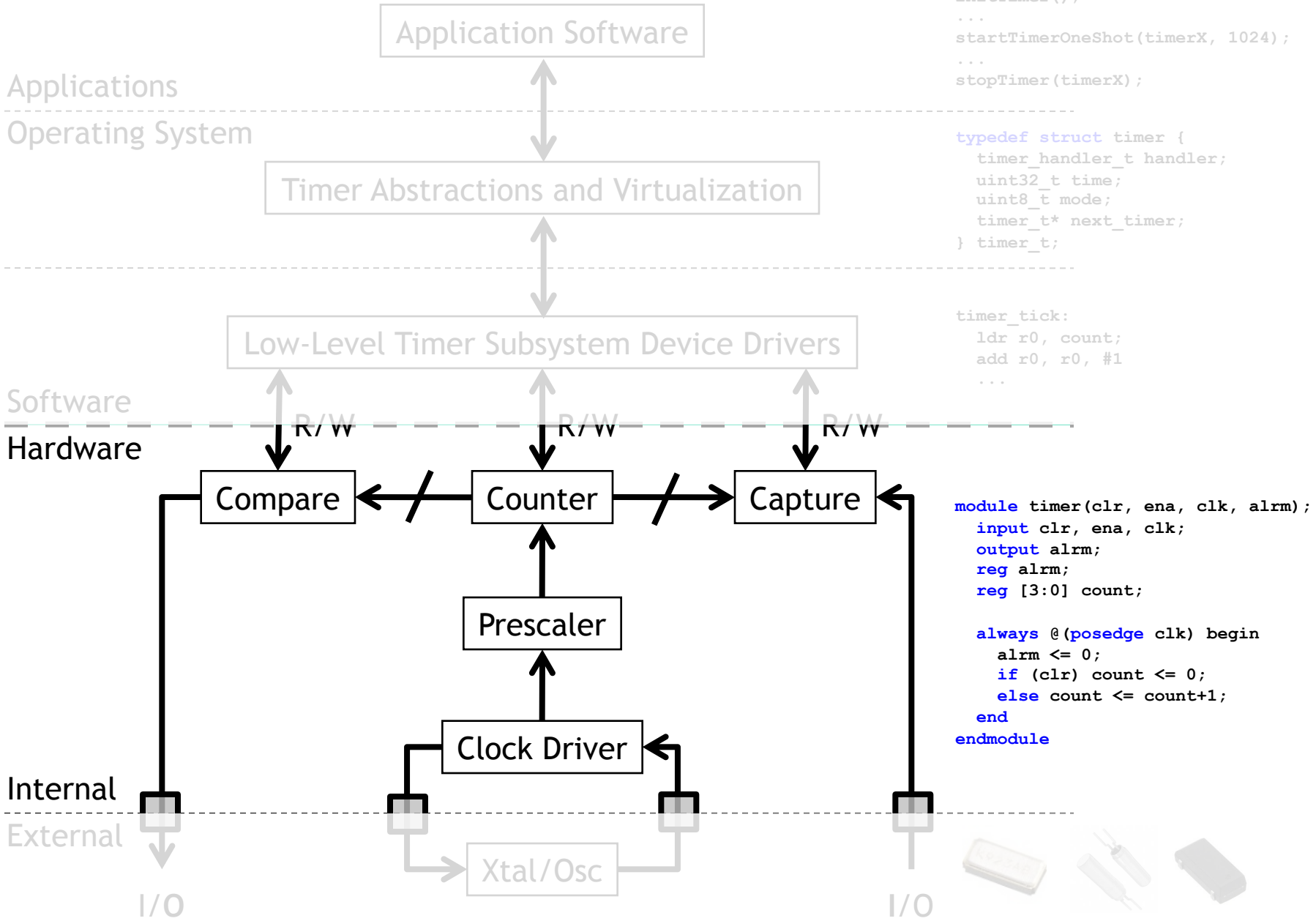


f (MHz)	C	R2
5	33pF	5kΩ
10	22pF	1kΩ
15	15pF	500Ω
20	10pF	270Ω



Pierce Oscillator

Anatomy of a timer system




```
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
  ...
```

```
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
```

What do we really want from our timing subsystem?

- 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
```

- Timer – count down time and notify when count = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

Why should we care?

- 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
  - Say you have a fan spinning and you want to know how fast it is spinning. One way to do that is to have it throw an interrupt every time it completes a rotation.
    - Right idea, but might take a while to process the interrupt, heavily loaded system might see slower fan than actually exists.
    - This could be bad.
  - Solution? Have the timer note *immediately* how long it took and then generate the interrupt. Also restart timer immediately.
- Same issue would exist in a car when measuring speed of a wheel turning (for speedometer or anti-lock brakes).

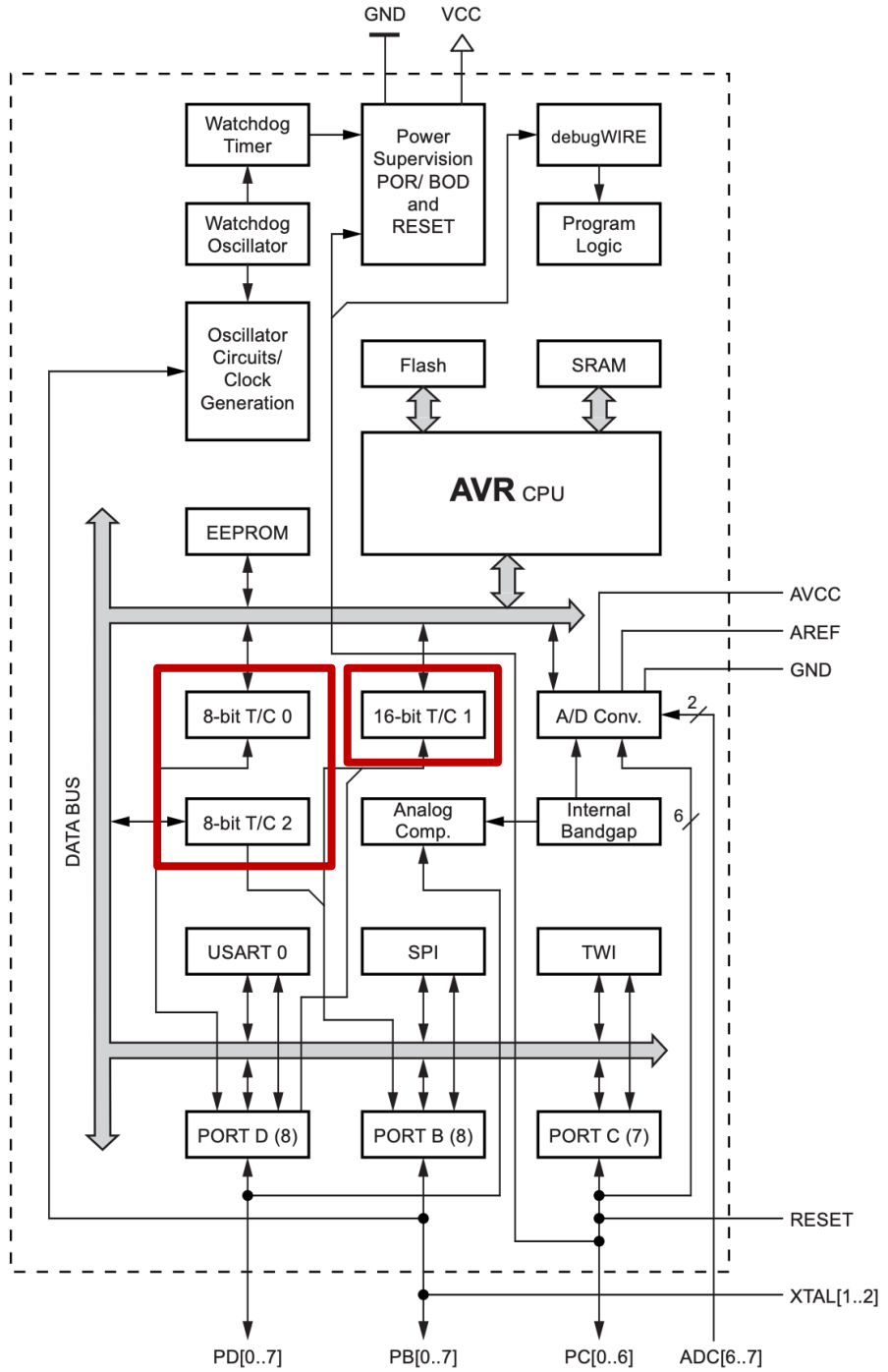
Example # 2: Compare

- Driving a DC motor via PWM.
  - Motors turn at a speed determined by the voltage applied.
    - Doing this in analog can be hard.
      - Need to get analog out of our processor
      - Need to amplify signal in a linear way (op-amp?)
    - Generally prefer just switching between "Max" and "Off" quickly.
      - Average is good enough.
      - Now don't need linear amplifier—just "on" and "off". (transistor)
  - Need a signal with a certain duty cycle and frequency.
    - That is % of time high.

ATMEGA328P Timer System

Timer Resources

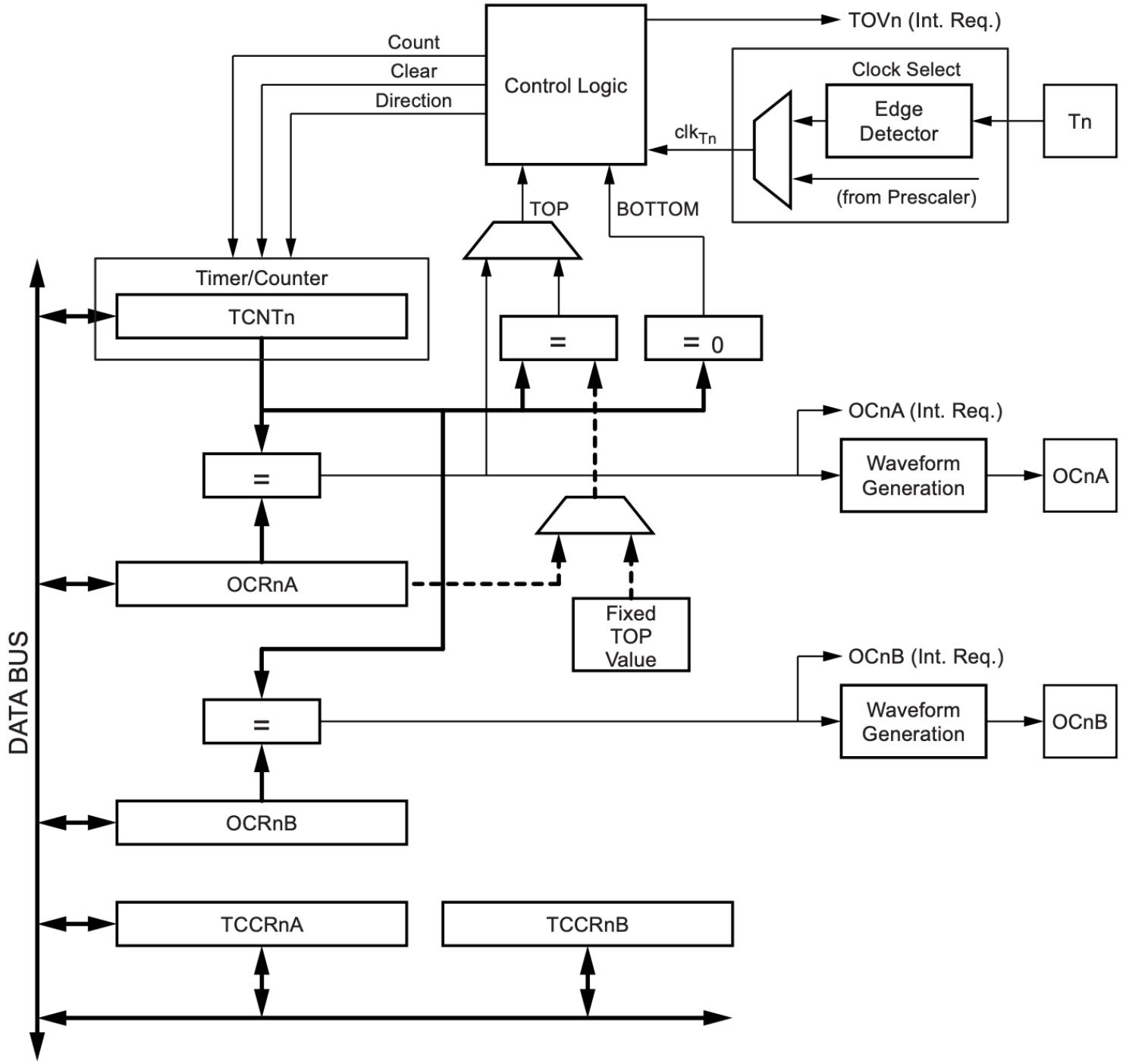
- 8-bit Timer/Counter 0
- 16-bit Timer/Counter 1
- 8-bit Timer/Counter 2



8-bit Timers

Modes of operation

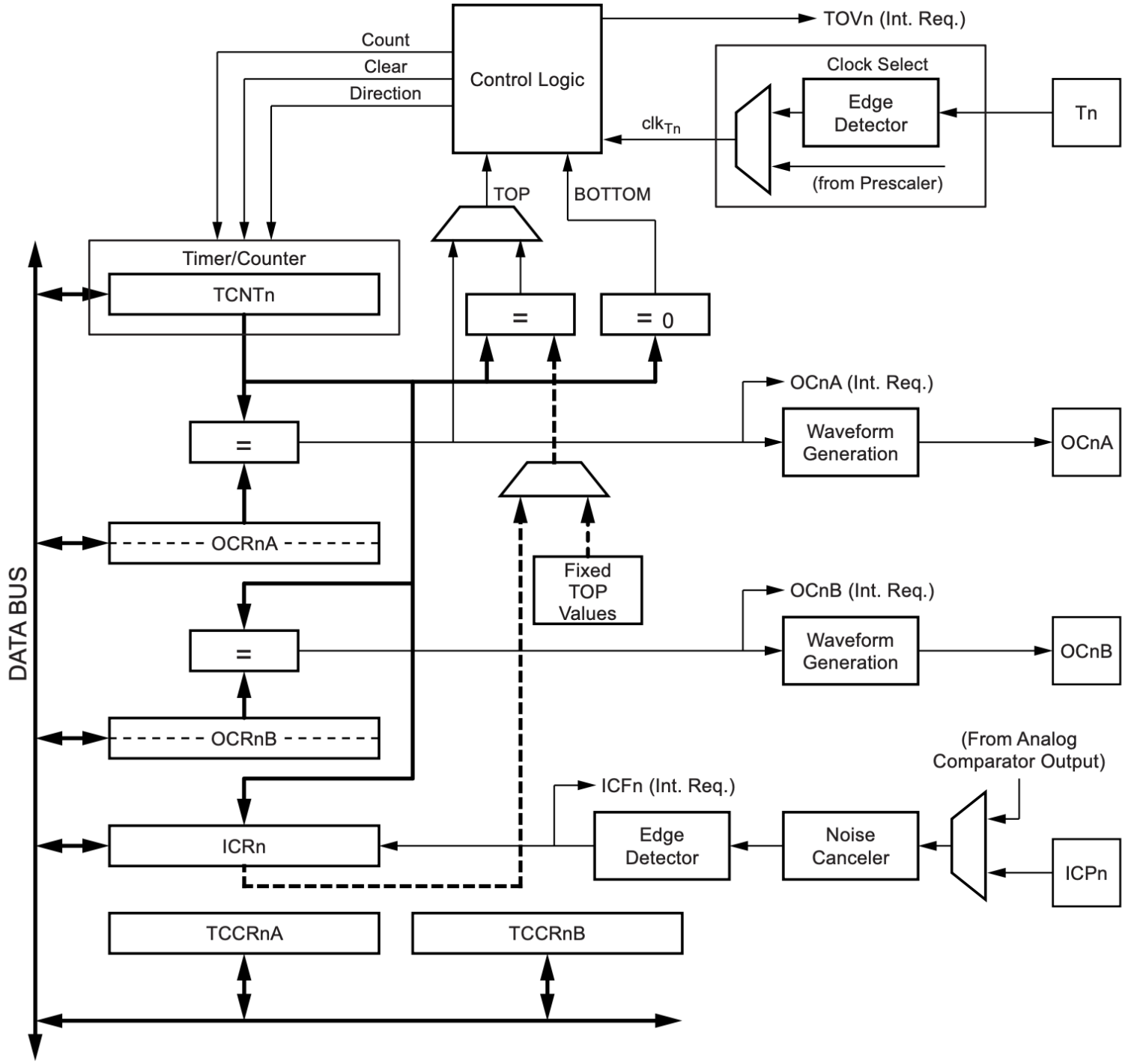
- Overflow (reset at 0xFF)
- Capture (CTC)
- Fast PWM
- Phase-correct PWM



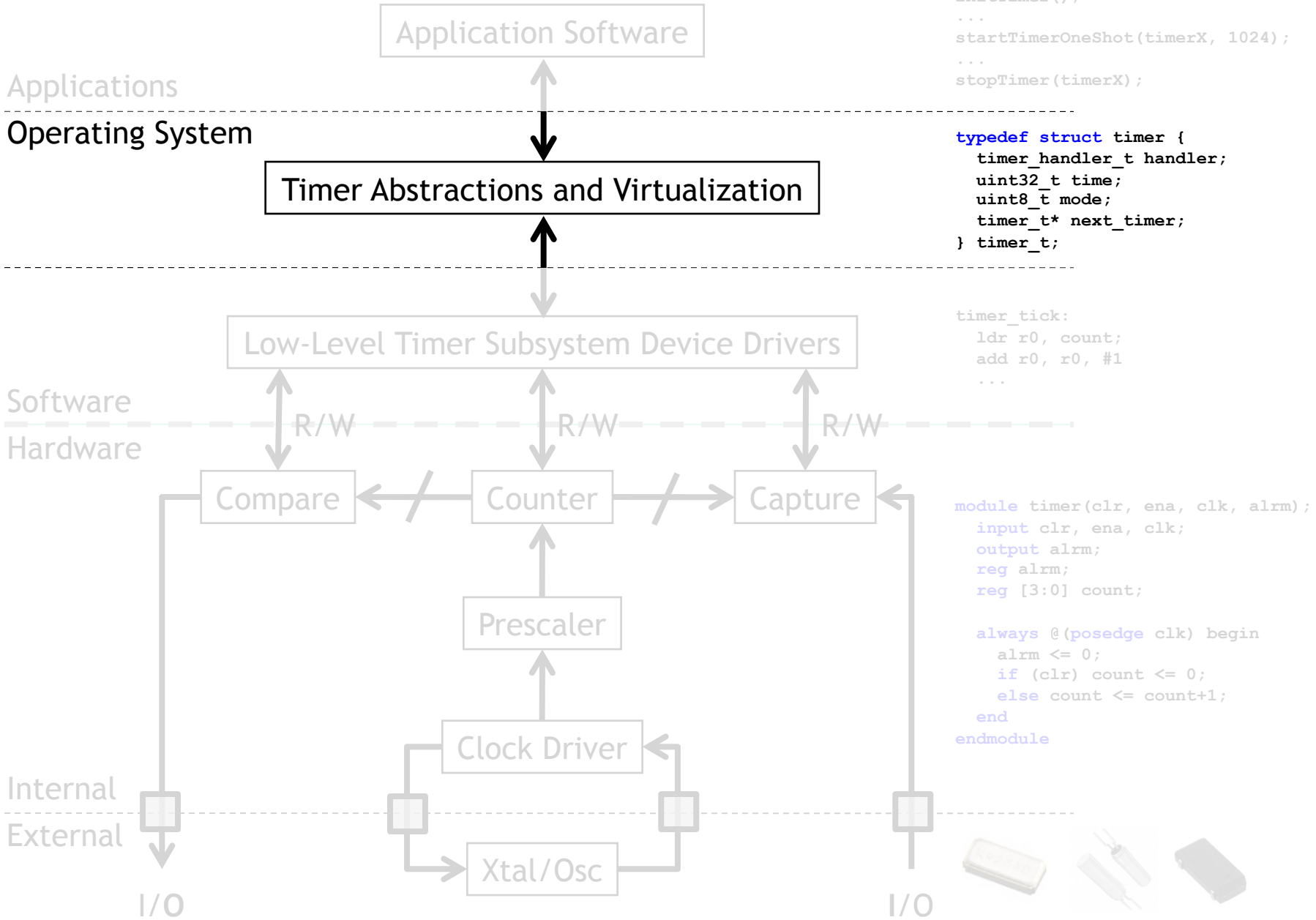
16-bit Timer

Modes of operation

- Overflow (reset at 0xFFFF)
- Capture
  - CTC
  - ICR
- Fast PWM
  - 5 modes
  - 8 to 16-bit resolution
- Phase-correct PWM
  - 7 modes
  - 8 to 16-bit resolution



Anatomy of a timer system




```
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
  ...
```

```
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
```

Virtual Timers

- You never have enough timers.
  - Never.
- So what are we going to do about it?
  - How about we handle in software?

Virtual Timers

- Simple idea.
  - Maybe we have 10 events we might want to generate.
    - Just make a list of them and set the timer to go off for the *first* one.
      - Do that first task, change the timer to interrupt for the next task.

Problems?

- Only works for “compare” timer uses.
- Will result in slower ISR response time
  - May not care, could just schedule sooner…

Implementation Issues

- Shared user-space/ISR data structure.
  - Insertion happens at least some of the time in user code.
  - Deletion happens in ISR.
    - We need critical section (disable interrupt)
- How do we deal with our modulo counter?
  - That is, the timer wraps around.
  - Why is that an issue?
- What functionality would be nice?
  - Generally one-shot vs. repeating events
  - Might be other things desired though
- What if two events are to happen at the same time?
  - Pick an order, do both…

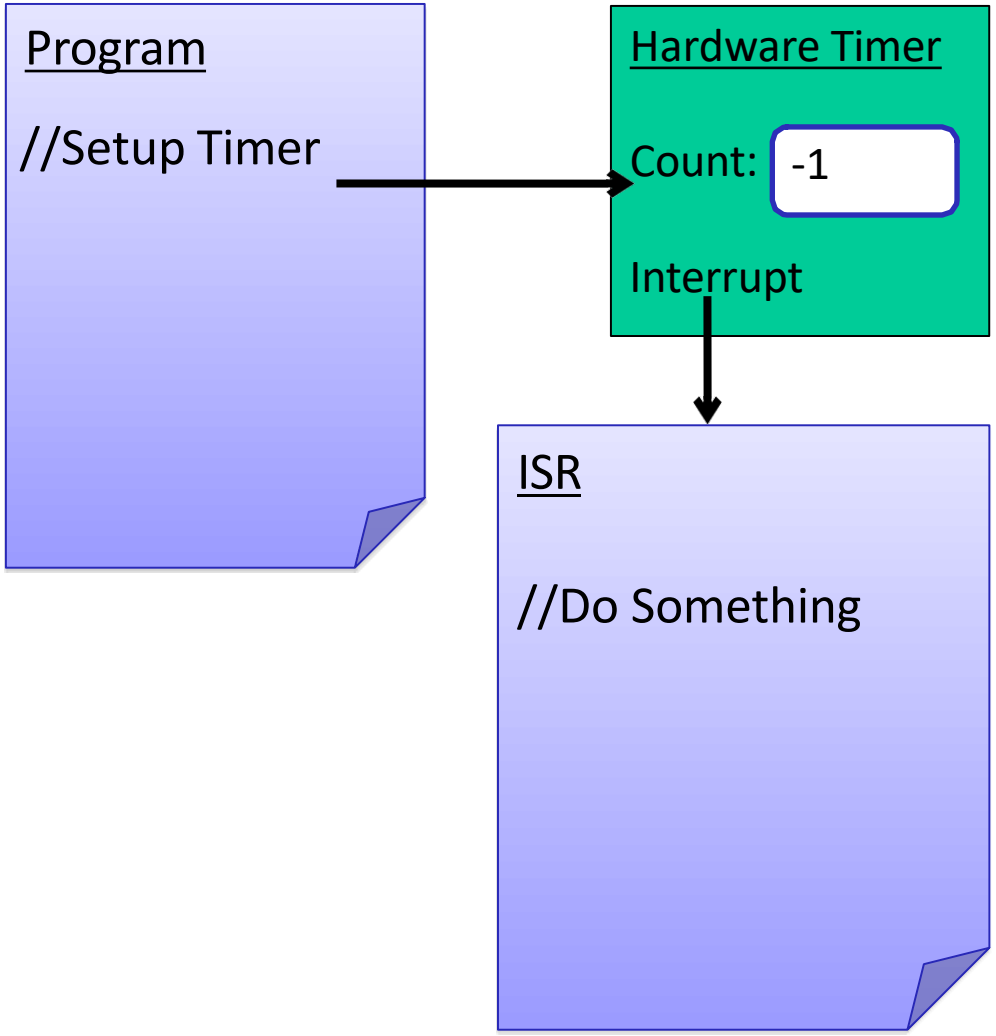
Implementation Issues (continued)

- What data structure?
  - Data needs be sorted
    - Inserting one thing at a time
  - We always pop from one end
  - But we add in sorted order.

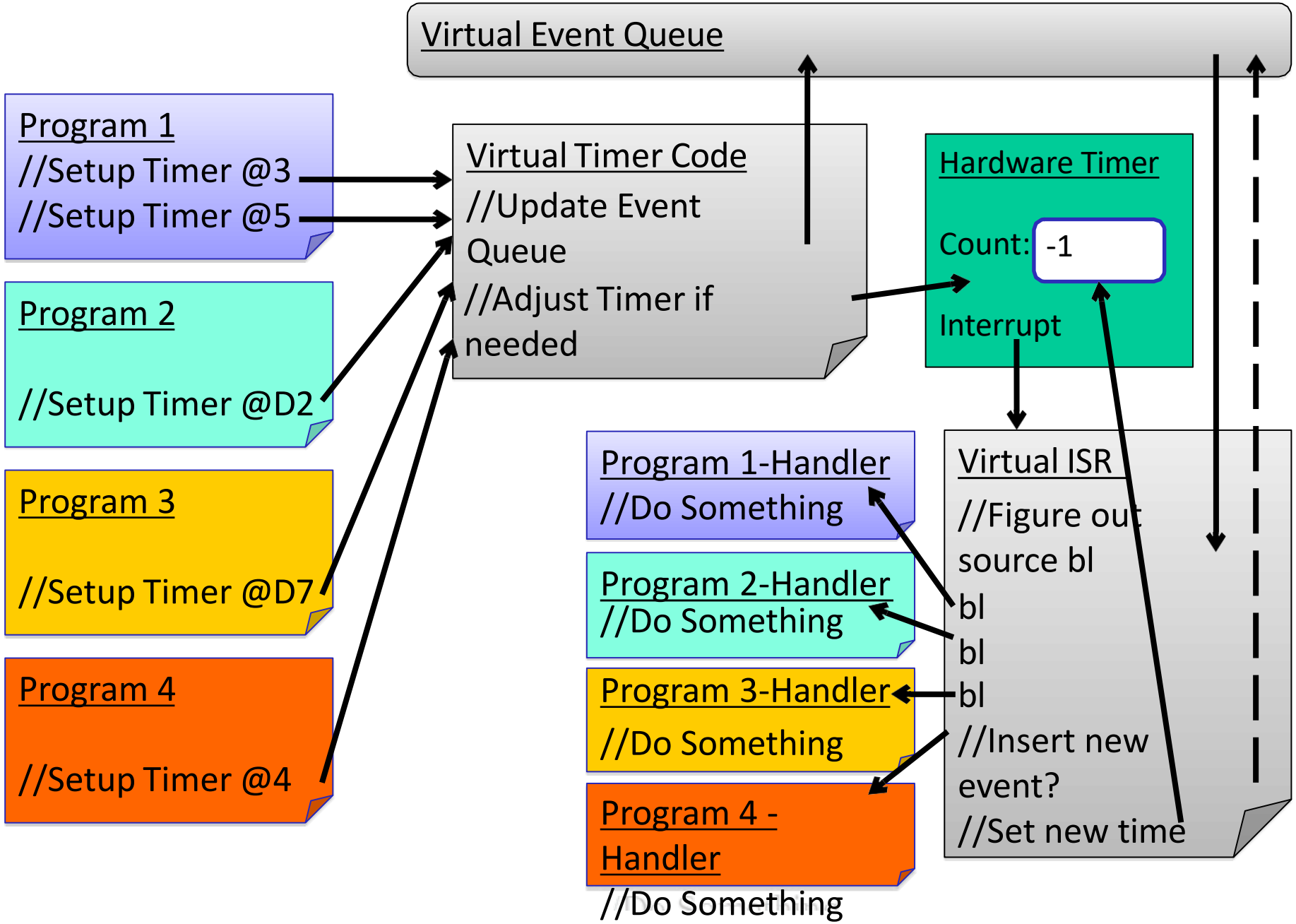
Data structures

```
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
}
```

HW Timer


Program
//Setup Timer
Hardware Timer
Count:
-1
Interrupt
ISR
//Do Something

Virtual Timer



Acknowledgements

- These slides contain materials from Prabal Dutta, Mark Brehob and Thomas Schmid (UMich)