Lecture #12

Time And Counters; Watchdog Timers

18-348 Embedded System Engineering

Philip Koopman

Monday, 22-Feb-2016

Electrical & Computer ENGINEERING

Carnegie Mellon

https://www.youtube.com/watch?v=-5wpm-gesOY

Where Are We Now?

- **Where we've been:**
  - Part 1 of course – lots of general topics that you'll need
  - DON'T FORGET TO:
    - Look at feedback from TAs on your labs
- **Where we're going today:**
  - Time and counters – a bit more nitty-gritty
  - Look for using previous concepts (e.g., fixed point math)
- **Where we're going next:**
  - Test #1 in class Wednesday Feb 24, 2016
  - Interrupts, concurrency, and scheduling
  - Analog and other I/O
  - Test #2 on Wednesday April 20, 2016
  - Final project is more self-directed; a bit more time to work on it

Preview

- **Time of day**
  - Accuracy, drift
  - How computers really measure time
- **Hardware timer operation**
  - Setting up a timer, including frequency calculations
  - Converting a hardware timer to time of day
  - Classic timer mistakes
- **Watchdog timer operation**
  - How and why to use a watchdog timer
  - How not to use a watchdog timer

How Do You Know What Time It Really Is?

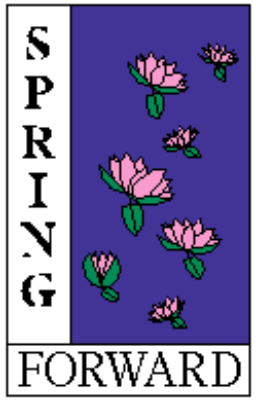
- **www.time.gov**

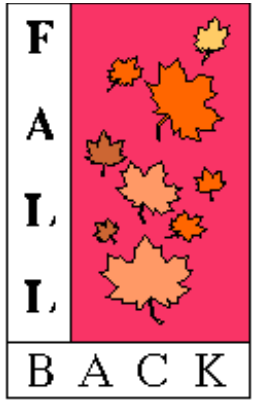
- **Other good sources:**
  - GPS
  - NIST radio broadcast (WWV radio)
  - Cell phone system
  - Internet time servers

  - But you have to know what time zone you're in!
    - (What about mobile systems?)

Daylight Savings Time & Time Zones

- **Daylight savings time switches on particular dates**
  - Which are declared annually by Congress and have been known to change
    - WW II had war-time daylight savings time to save energy
    - "Energy Crisis" in the 70's resulted in year-round daylight savings time
    - Only the Navajo nation within Arizona does DST (not the state; not the Hopi resv.)
  - http://www.energy.ca.gov/daylightsaving.html
    - **Beginning in 2007**, Daylight Saving Time extended:
    - **2 a.m. on the Second Sunday in March to 2 a.m. on the First Sunday of November.**
    - This does not correspond to European dates!





www.time.gov

Users complain iPhone clock bungles time change

Instead of springing forward, some iPhone clocks fell back

Companies: Verizon Communications Inc. Com

Related Quotes

Symbol	Price	Change
VZ	35.85	-0.55

On Sunday March 13, 2011, 7:41 pm EDT

NEW YORK (AP) -- It's hard enough to get your bearings when the time changes twice a year. It's all but impossible when your phone starts playing tricks on you, too.

Users of Apple's iPhone peppered Twitter and blogs with complaints Sunday when their phones bungled the one-hour "spring forward" to daylight savings time that went into effect overnight Saturday.

One user complained of missing church, another of almost missing yoga. One called her iPhone stupid and several just asked for help.

It turns out some users' phones fell back one hour instead of springing forward, making the time displayed on the iPhone two hours off.

This is just the latest clock woe for Apple's chic iPhone. A clock glitch prevented alarms from sounding on New Year's Day, causing slumbering revelers to oversleep. The devices also struggled to adjust to the end of daylight savings time back in November.

The glitch affected iPhone owners who subscribe for phone service through AT&T and Verizon.

Apple, based in Cupertino, Calif., could not be reached for comment Sunday.

TechWeek europe

Enhancing business with technology – in association with eweek.com

Windows Azure Leap-Year Glitch Takes Down G-Cloud

Microsoft says that most services have now returned to normal after a day of chaos

On March 1, 2012 by Steve McCaskill 5

Microsoft has confirmed that a service outage that affected its cloud computing service Microsoft Azure, appears to be caused by a leap year bug.

The Government's G-Cloud CloudStore was among the sites affected by the outage, which Microsoft says has mostly been rectified.

Leap Year Bug

"Yesterday, 28 February, 2012 at 5:45 PM PST Windows Azure operations became aware of an issue impacting the compute service in a number of regions," wrote Bill Laing, corporate vice president of Server and Cloud at Azure in a blog post. "While final root cause analysis is in progress, this issue appears to be due to a time calculation that was incorrect for the leap year."

"Once we discovered the issue we immediately took steps to protect customer services that were already up and running, and began creating a fix for the issue," he explained. "The fix was successfully deployed to most of the Windows Azure sub-regions and we restored Windows Azure service availability to the majority of our customers and services by 2:57AM PST, 29 February."

Laing did concede however that some regions and customers were still experiencing issues and that as a result they may be experiencing a loss of application functionality.

"We are actively working to address these remaining issues," he added. "We sincerely apologise for any inconvenience this has caused. "

Government Issues

The Government's G-Cloud CloudStore, which was launched earlier this month, was taken offline due to the problems.

"Power outage on microsoft azure means #cloudstore is temporarily unavailable. Patch being applied so will update when normal service resumed," said a post on the official G-Cloud twitter account.

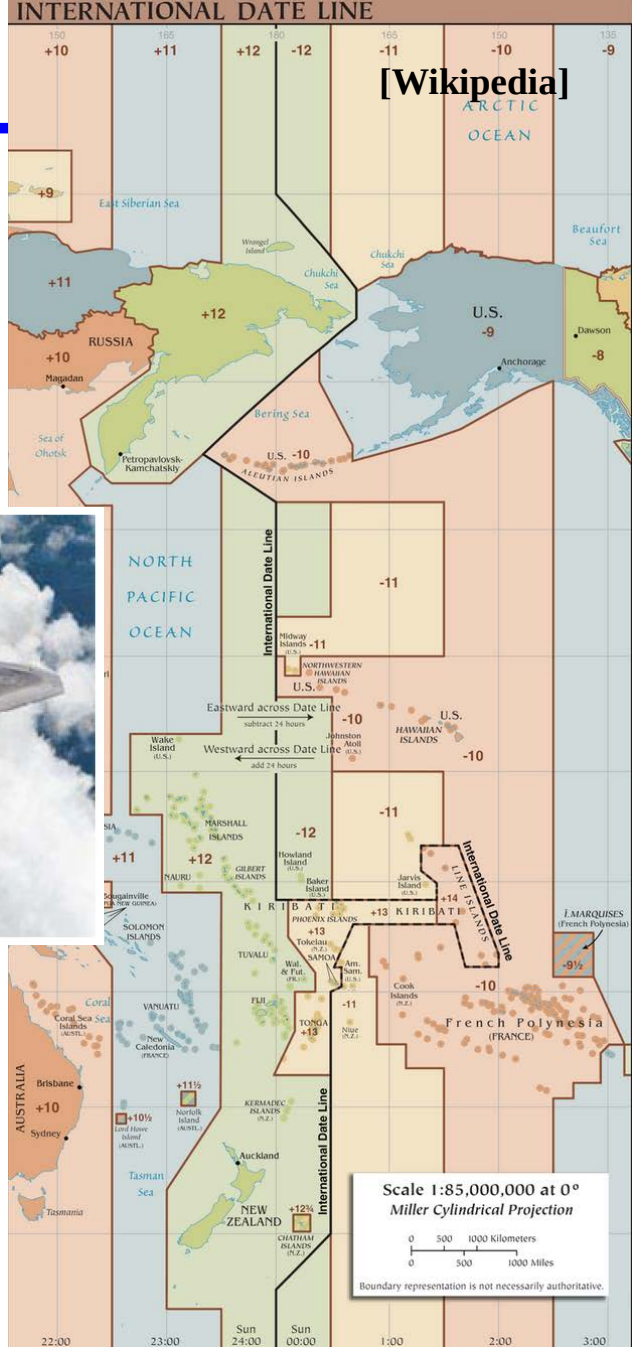
However a second message posted at 3:35pm GMT read: "Update on #cloudstore: microsoft are moving us to a different azure install and are confident we'll be up and running again by 4pm"

This is not the first time that Azure has gone offline. In March 2009, an outage left users unable to access the early test applications. This latest incident is unlikely to inspire confidence in IT managers still recovering from the Amazon Web Services (AWS) outage that occurred last April.

F-22 Raptor Date Line Incident

- **February 2007**
  - A flight of six F-22 Raptor fighters attempts to deploy to Japan
  - $360 million per aircraft
    - (Perhaps $120M RE, rest is NRE)
  - Crossing the International Date
    - No navigation
    - No communications
    - No fuel management
    - Almost everything gone!
    - Escorted to Hawaii by tankers
    - If bad weather, might have caused loss of aircraft
  - Cause: "It was a computer glitch in the millions of lines of code, somebody made an error in a couple lines of the code and everything goes."

[DoD]



[Wikipedia]

2013: NASA Declares End to Deep Impact Comet Mission

http://apod.nasa.gov/apod/image/0505/art1_deepimpact.jpg

Launched in 2005, the spacecraft memorably smashed a copper-jacketed probe into the comet Tempel 1 at 22,800 miles an hour (36,700 kilometers an hour) on July 4 of that year. It then flew through the debris cloud to capture the resultant fireworks, the first close inspection of a comet's interior. (See "'Deep Impact' Comet Revealed by NASA Flyby.")

The $267 million spacecraft later flew by the comet Hartley 2 in 2010, and this year it captured images of comet ISON, which is headed toward a close encounter with the sun in November.

But now the Deep Impact spacecraft appears to be lost.

Mission controllers last radioed the spacecraft on August 8, after which communications were lost, according to a statement from the Jet Propulsion Laboratory in Pasadena, California. After a month of attempts to restore communications through the NASA Deep Space Network, the controllers have declared the mission "lost," concluding that a computer glitch likely doomed the spacecraft.

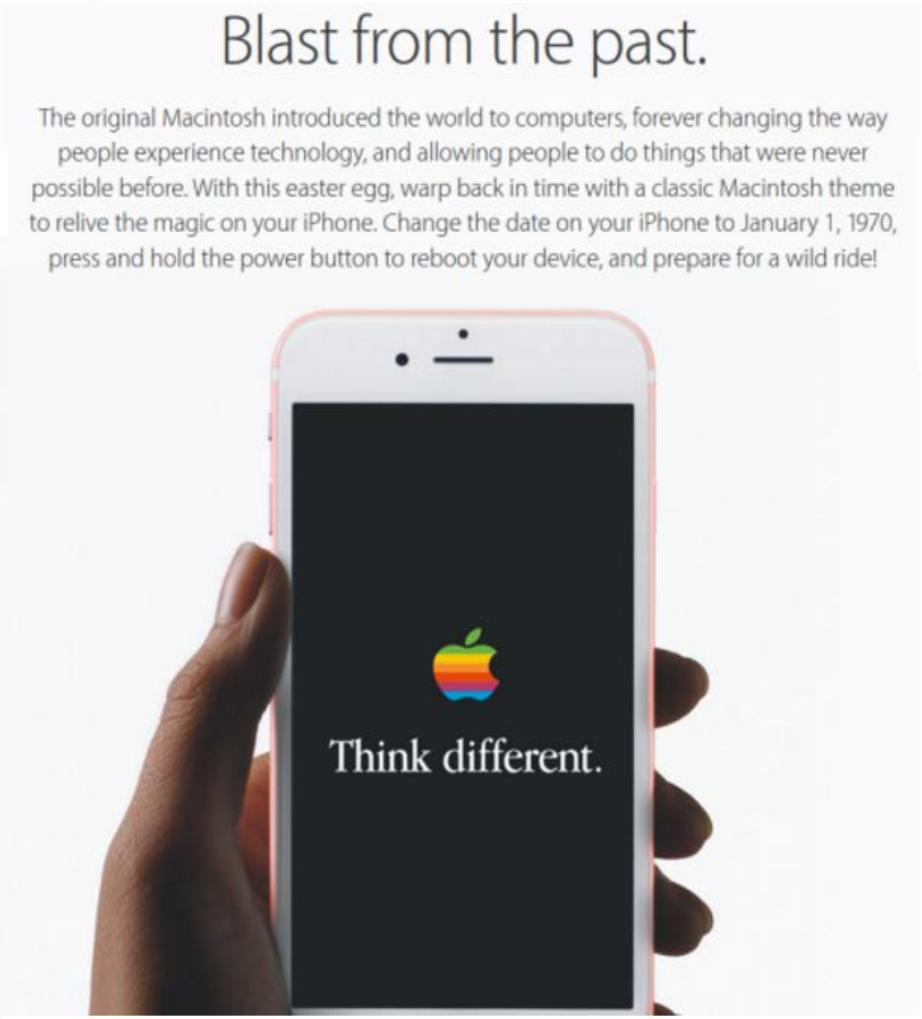
"Basically, it was a Y2K problem, where some software didn't roll over the calendar date correctly," said A'Hearn. The spacecraft's fault-protection software (ironically enough) would have misread any date after August 11, 2013, he said, triggering an endless series of computer reboots aboard Deep Impact.

Dan Vergano
National Geographic
Published September 20, 2013

http://news.nationalgeographic.com/news/2013/09/130920-deep-impact-ends-comet-mission-nasa-jpl/

Do Not Set The Date On Your iPhone To Jan. 1, 1970

By Mary Beth Quirk February 12, 2016



Fakety fake fake fake.

While it might be tempting to take a "wild ride" into the past, do not set the date on your iPhone to Jan. 1, 1970, despite what a hoax image circulating recently says. That is, unless your idea of a wild ride is having a phone you can't use anymore.

Note: Unix epoch is 00:00:00 UTC on 1 January 1970.

ISO 8601 date format: 1970-01-01T00:00:00Z.

Problems With Time in the Real World

- **Coordinated Universal Time (UTC; world time standard)**
  - Is not a continuous function due to leap seconds
    (and is only monotonic by putting 61 seconds in a minute just before midnight)
  - Leap year also causes discontinuities, although they're more predictable

- **Time zones**
  - Not just on hourly boundaries – Venezuela is UTC/GMT -4:30 hours; no DST
  - TV auto-time-set might sync to channel from wrong time zone via cable feed

- **DST changeover date changes fairly often**
  - With little warning compared to a 10-20 year embedded system lifetime

- **"Y2K"**
  - "99" ➔ "00" on Jan 1, 2000 (there were many failures, but world did not end)
  - The GPS 1024 week time rollover (a ship got lost at sea…)
  - And Unix rollover problem (January 19, 2038 03:15:07 GMT)

Internationalization

- **The Moral Of The Time Stories:**
  - Keep time in GMT or UTC, not local time
  - Keeping time is tricky (rollover, time zones, etc.); ***kids don't try this at home***
  - And … it's more than just time keeping

- **What day is 02/03/16?**
  - In the US: Feb 3, 2016
  - In Europe: 2 March 2016
  - Don't forget: AM / PM vs. 24 hour clock

- **Other internationalization issues:**
  - English vs. Metric (F vs. C; ft vs. meters; speed limit in mph + distance in km)
  - Many, many complications on translation
    - Singular v. plural; Gender
    - Currency signs & conversion
    - ASCII vs. 16-bit Unicode
    - …

Time and Computers

- **Computers are digital (and therefore discrete) devices**
  - Can count up things (for example, seconds)
  - But, can't actually represent exact analog values

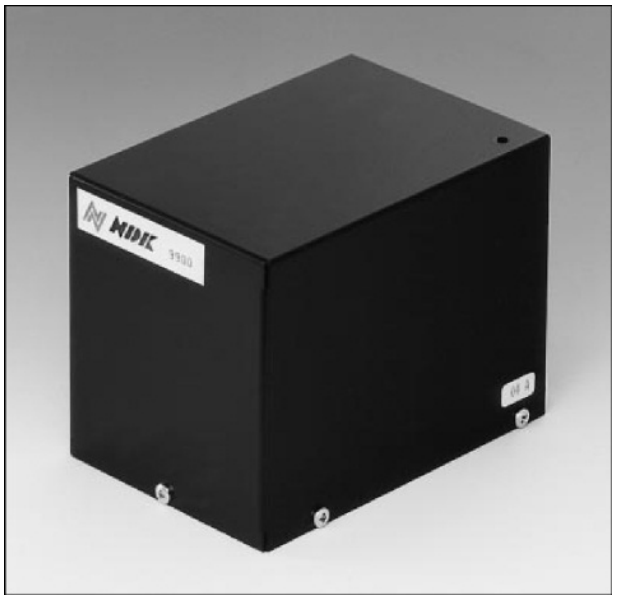
- **Time is an analog value**
  - Time flows smoothly as far as we're concerned, not in big chunks
  - How do we get from a smooth, continuous flow to a countable number?

- **Basic source of timing information – the system clock**
  - A clock provides discrete time chunks at some operating speed
  - Not only cycles the CPU registers, but also providing a basis for counting time!

  - Basis of time in computers is – no surprise – counting "clock" cycles

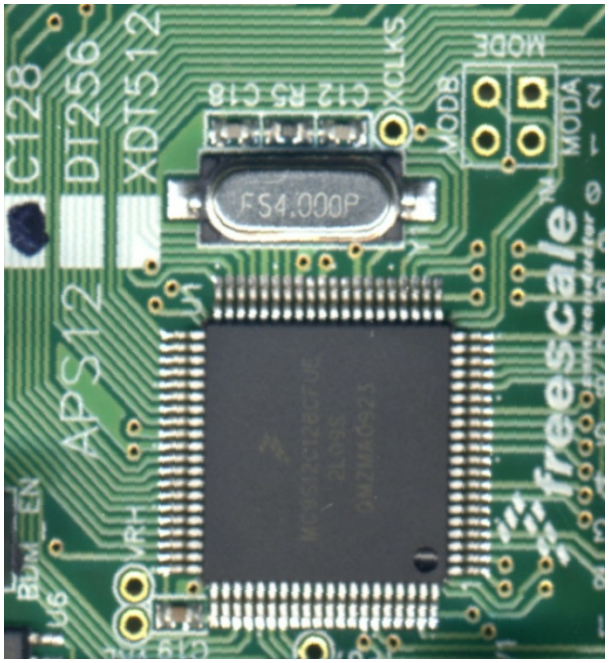
Physical Clock – What's The Basis For Time?

- **Typical source: oscillator circuit, perhaps augmented with GPS time signal**
  - R/C timing circuit; somewhat stable (e.g., 1% resistor gives a lot of drift!)
  - Commodity crystal oscillator; perhaps 10^{-6} /sec stability (14-pin DIP size)
  - Oven-controlled for wireless communications; perhaps 10^{-11} /sec stability
  - Micro-rubidium atomic oscillator
    - perhaps 10^{-11} /*month* stability
    - 0.7 kg weight
    - 0.3 liter volume



Can You Run Faster Than The Oscillator?

- **Course board uses a 4 MHz crystal oscillator**
  - Divided to 2 MHz to get accurate 50% duty cycle
  - Specs from module documentation are: 4 MHz +/- 30 ppm
    - 30 ppm = 3 * 10^{-5} = 0.003% => +/- 120 Hz



- **We want to run at 8 MHz**
  - CPU can handle up to 25 MHz
  - Old modules ran at 8 MHz and this avoids many potential bugs in course software infrastructure
  - Any guesses as to why new module is at 2 MHz?

- **Running faster than the oscillator:**
  - Turn on the PLL (Phase Locked Loop)
  - Set PLL multiplier
  - Hardware automatically generates faster clock that tracks input oscillator edges
  - What is drift rate of this faster oscillator?

Simple Real-World Drift Example

- **A gizmo has a crystal oscillator running at 32,768 Hz + 0.002%**
  - 32,768 is a standard watch crystal frequency (15-bit divider gives you 1 Hz)
    - (.002% is a $2*10^{-5}$ drift rate)
  - The product specification requires accuracy of 2 seconds/day
  - Assume perfect software counting of oscillator clock cycles
  - Will the oscillator meet the specification?

(.00002 sec/sec drift rate * (60 sec * 60 min * 24 hr)

= 1.728 sec drift per day (*so it meets the spec.*)

  - How far will it drift over a 2-year battery life?

1.728 sec/day * (365.25 days * 2 years) = 21 minutes drift over 2 years

- **Observations:**
  - 10^{-6} or 10^{-7} is probably desirable for consumer products that keep time
    - Is the course computer good enough to be a clock?
  - There are a lot of seconds in a year (31.6 million ~= 2^{25} of them)

Counting The Clocks

- **Time is an integer count of some number of clock "ticks"**
  - One year @ 2 MHz takes about 47 bits to represent as an integer – too big to be useful for most embedded applications
  - But, most applications don't need time to the nearest 1/2,000,000 second
  - So, we want time with a bigger granularity than this

- **Thus, the concept of the timer**
  - Increment a "timer" once every N CPU clocks (this is a clock "tick")
    - Potentially, tell the CPU to update its software-maintained clock on every timer increment; maybe a 32-bit integer
  - Example: Original IBM PC updated time of day 18.2 times/second
    - Windows Forms timer is still that speed (55 msec)
  - Many Unix systems have base timers that run at 30 or 60 times/second
    - Why this frequency?

Course Chip Timing Support

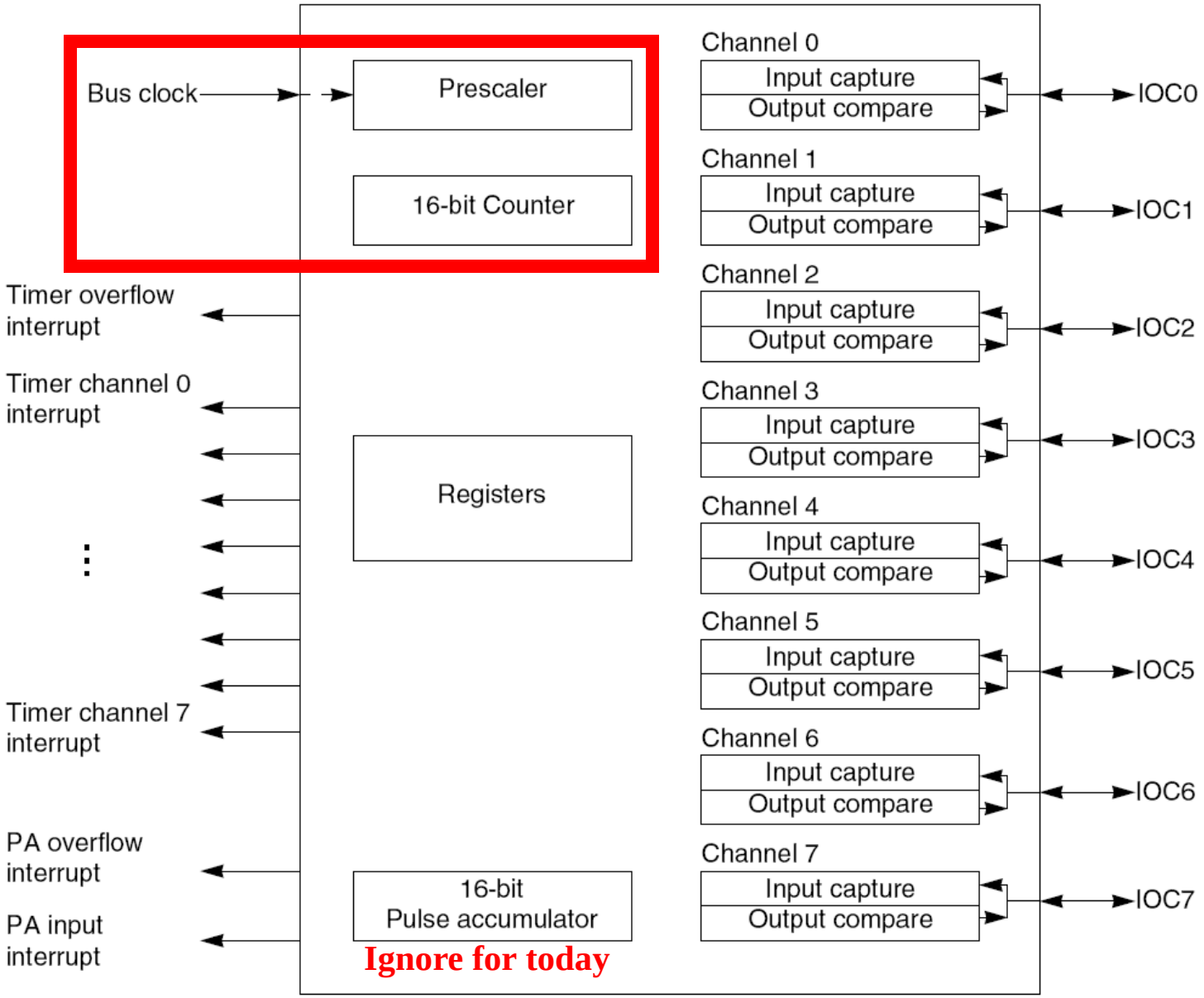


Figure 15-1. TIM16B8CV1 Block Diagram

[Freescale]

Hardware Timer Operation

- **"Channels" and "IOC" items are for pulse inputs/outputs**
  - Not relevant to this lecture
- **Prescaler**
  - Divide system clock by an integer value as input to timer
    - System clock is 8 MHz for course HW; defaults to some other speed in simulator
  - PR[2:0] controls prescale amount
    - Divide bus clock by: 1, 2, 4, 8, 16, 32, 64, 128
- **16-bit Counter -- TCNT**
  - "up" counter – always increments
  - Clocked by prescaled bus – one increment every 1, 2, 4, 8, … , 128 bus clocks

15.3.2.11 Timer System Control Register 2 (TSCR2)

Module Base + 0x000D

	7	6	5	4	3	2	1	0
R	TOI	0	0	0	TCRE	PR2	PR1	PR0
W								
Reset	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Figure 15-19. Timer System Control Register 2 (TSCR2)

[Freescale]

Reading The Hardware Timer

```
// set TN = 1   Timer Enable   TSCR1 bit 7
TSCR1 |= 0x80;
// set  PR[2:0]  Timer prescale in bottom 3 bits of TSCR2
TSCR2 = (TSCR2 & 0xF8) | 0x04;   // 0x04  bus clock / 16
for(;;) {  timer_val = TCNT;
} /* update timer_val forever */
```

Table 15-14. Timer Clock Selection

PR2	PR1	PR0	Timer Clock
0	0	0	Bus Clock / 1
0	0	1	Bus Clock / 2
0	1	0	Bus Clock / 4
0	1	1	Bus Clock / 8
1	0	0	Bus Clock / 16
1	0	1	Bus Clock / 32
1	1	0	Bus Clock / 64
1	1	1	Bus Clock / 128

How Can We Use This To Measure Time?

- **Every time TCNT rolls over to zero, increment a software time counter**
  - This is really inefficient!!! – but demonstrates the general idea

```
int time_count = 0;
// set TN = 1   Timer Enable   TSCR1 bit 7
TSCR1 |= 0x80;
// set  PR[2:0]  Timer prescale in bottom 3 bits of TSCR2
TSCR2 = (TSCR2 & 0xF8) | 0x07;   // 0x07  bus clock / 128

// Only works if loop is faster than timer increments!
for(;;)
{ // increment time_count whenever TCNT reaches zero
  if (TCNT == 0)
  { time_count++;
    while (TCNT == 0); /*wait for TCNT to change again*/
  }
}
```

How Fast Does That Time Counter Increment?

- **Analytically:**
  - 8 MHz module
  - 16-bit counter rolls over every 65536 counts
  - Prescale by 128, so roll over happens 128 times slower

$$time = 65536 * 128 / 8{,}000{,}000 = 1.048576 \text{ seconds}$$

  - (How long for 2 MHz Module?)

- **Experimentally (via simulator set for 8 MHz):**
  - Set breakpoint at: time_count++;
  - First breakpoint: 8,372,411
  - Second breakpoint: 16,761,026
  - Elapsed time: 8388615 clocks = 1.048577 seconds
    - (accurate within less than time to execute the loop testing for zero)

Accuracy

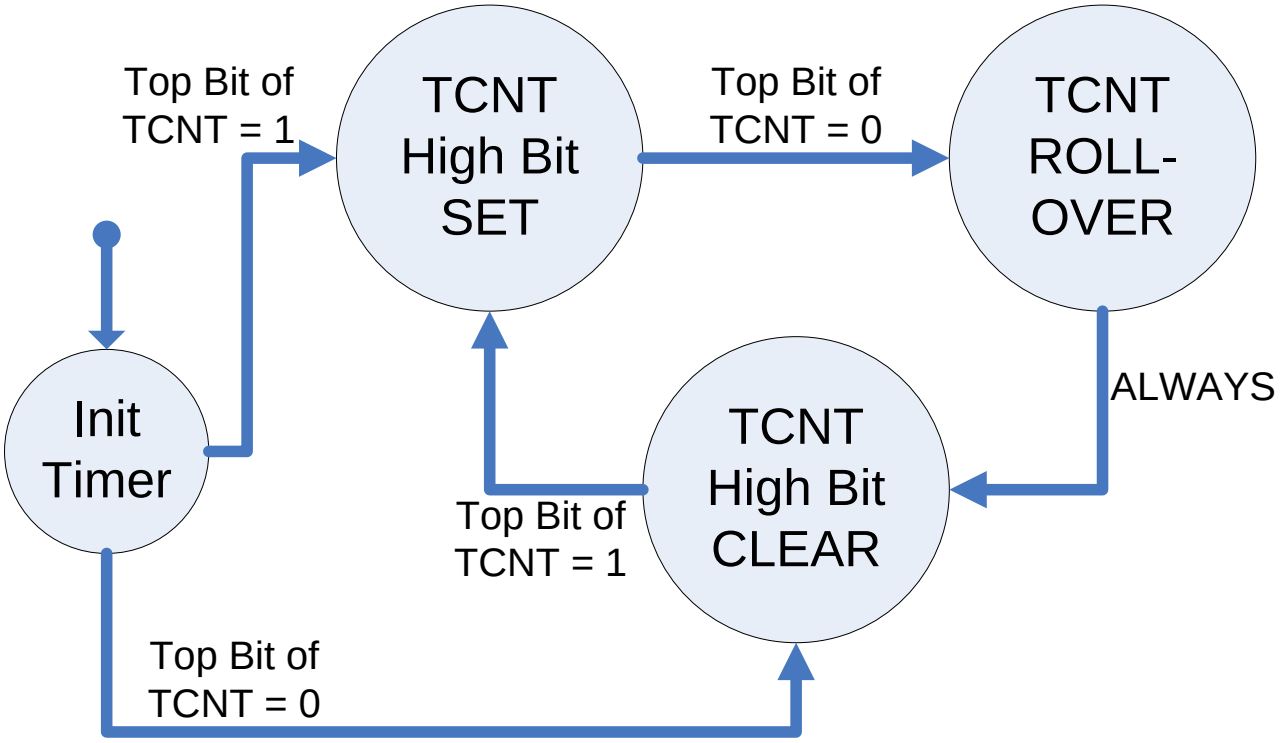
- **What if we wanted to display time with seconds?**
  - This hardware won't make that easy!
  - Can't get exactly 1 second tick values from hardware
  - Can do better by updating a lot more frequently than every second

- **For example, to display time in seconds…**
  - Find a divider value for which TCNT rolls over every 0.025 to 0.10 seconds
    - This is how the IBM PC got 0.055 second ticks – it was an "easy" divider value
  - Update a software counter on every TCNT rollover
  - Whenever that software counter exceeds 1 second of value, update the seconds count

  - This still won't display exact seconds….
    - Accurate to within TCNT rollover period plus sampling jitter
    - But for a clock the human eye can only "see" about 0.05 to 0.1 seconds anyway

Design To Track Seconds

- **Keep state machine to track rollover**
  - Only needs to sample TCNT a few times per rollover to avoid missing one
  - Accuracy improved with sampling speed
  - Do other application stuff in both “TCNT high bit set” and “TCNT high bit clear” states
  - But how do we handle the rollover?



Design Example – Don't Lose Fractions

- **Assume bus clock divide by 64; 25 MHz board**
  - 65536 * 64 / 25,000,000 ➔ TCNT rollover every 0.167772 seconds
  - (Need to sample TCNT every 0.08 seconds to catch the rollover event)

  - If we want to keep seconds, then increment seconds every 5 or 6 rollovers

- **How do we track fractional seconds without floating point?**
  - Answer: 16.16 fixed point! – a 32-bit fixed point integer
    - unsigned long current_time;
    - Top 16 bits are integer seconds
    - Bottom 16 bits are fractional seconds
      (each integer "count" = 1/65536 seconds = 0.00001525878906 seconds)
  - For each TCNT rollover, add 0.166772 / 0.00001525878906
    = 10930 fractional seconds

  - TCNT rollover becomes: current_time += 10930;
  - Seconds are in: (current_time >> 16) & 0xFFFF;

Time Accuracy Calculation

- **An approximation makes life easy, but how far off is it?**

- **In 10,000 seconds, TCNT will roll over:**
  - 10,000 * 25,000,000 / (65536 * 64) = 59,605 times
  - That's 10930 fractional seconds added to the 32-bit time counter
    10930 * 59,605 = 651,482,650 ➔ $26D4 D61A
  - Top 16 bits are $26D5 (rounded) ➔ 9941 (instead of 10,000)

  - Accuracy is 9941/10,000 ➔ 99.41% (0.59% error due to timer interval)
  - How could we be better?

- **Is this good enough?**
  - Crystal Oscillator is 4 MHz +/- 0.003%, which is insignificant for this purpose
  - Error is: 0.59% * 31536000 seconds/year = 51.7 hours per year; 8.5 minutes/day
  - **NOTE:** Our time counter rolls over every 64K seconds = 18.2 hours
    - What this really means is you want 32.32 fixed point time for longer operation

Why Are Timers Such A Big Deal?

- **No more counting NOPs in loops**
  - NOP-delay loops are a pain to build and get right
  - And they break every time you change the oscillator speed or CPU clocks/instr!

- **Lets processor do other useful work while keeping time**
  - Can check timer once in a while to see if top bit of TCNT rolled over
  - Combined with "interrupts" (next lecture), processor doesn't have to check time periodically – is just notified on every rollover of TCNT

- **Time values independent of software execution**
  - Not sensitive to variations in instruction timing
  - Still works if software inside loop has multiple "if/else" paths… because it is not based on how long software takes to run
  - Still works at different clock speed (need to adjust the prescale value)

- **BUT, it's a bit of work getting accurate time-of-day values**
  - Have to take into account exactly how often HW timer ticks and rolls over!

Classic Timer Mistakes – "Nanosecond" Time

- [http://www.gnu.org/software/libc/manual/html_node/Elapsed-Time.html#Elapsed-Time]
- **– Data Type: struct timespec**
  The struct timespec structure represents an elapsed time. It is declared in time.h and has the following members:
- **long int tv_sec**
  - **This represents the number of whole seconds of elapsed time.**
- **long int tv_nsec**
  - **This is the rest of the elapsed time (a fraction of a second), represented as the number of nanoseconds. It is always less than one billion.**

- **This value reports time in nanoseconds**
  - That means it is a number of nanoseconds
  - That does NOT mean it is the ***nearest*** nanosecond
  - The underlying hardware has a timer that only increments once in a while!

  - Classic mistake is to ignore quantization error in the timers

Classic Timer Mistakes – Non-Atomic Access

The 16-bit main timer is an up counter.

A full access for the counter register should take place in one clock cycle. A separate read/write for high byte and low byte will give a different result than accessing them as a word.

Address Offset	Use
0x0000	Timer Input Capture/Output Compare Select (TIOS)
0x0001	Timer Compare Force Register (CFORC)
0x0002	Output Compare 7 Mask Register (OC7M)
0x0003	Output Compare 7 Data Register (OC7D)
0x0004	Timer Count Register (TCNT(hi))
0x0005	Timer Count Register (TCNT(lo))

[Freescale]

- **What happens if you use two 8-bit reads (LDAA) instead of LDD?**
  - 16-bit fetch locks the value as it is being read; gives correct result
  - Timer hardware might increment between two byte-sized reads
  - AND, that increment might include a carry from low 8 to high 8 bits

  - $**03**FF → $04**00** read hi then lo gives $03 … $00 => $0300!
  - This is an absolutely classic timer bug – don't let it happen to you!!!

Classic Timer Mistakes – Rollover

http://www.nytimes.com/2015/05/01/business/faa-orders-fix-for-possible-power-loss-in-boeing-787.html

DEPARTMENT OF TRANSPORTATION
Federal Aviation Administration

Airworthiness Directives; The Boeing Company Airplanes
AGENCY: Federal Aviation Administration (FAA), DOT.
ACTION: Final rule; request for comments.

SUMMARY: We are adopting a new airworthiness directive (AD) for all The Boeing Company Model 787 airplanes. This AD requires a repetitive maintenance task for electrical power deactivation on Model 787 airplanes. This AD was prompted by the determination that a Model 787 airplane that has been powered continuously for 248 days can lose all alternating current (AC) electrical power due to the generator control units (GCUs) simultaneously going into failsafe mode. This condition is caused by a software counter internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane.

DATES: This AD is effective May 1, 2015.

- **Eventually integer timers roll over**
  - Assume time kept in 100ths of a second as a **signed 32-bit integer** (wrong type!)
  - 0x7FFFFFF = 2147483647 / (24 * 60 * 60 * 100) = **248.55 days to overflow**
  - (Note: unsigned int would roll over after 497 days)

http://rgl.faa.gov/Regulatory_and_Guidance_Library/rgad.nsf/0/584c7ee3b270fa3086257e38004d0f3e/$FILE/2015-09-07.pdf

More Consequences of Bad Code

Product recalls

Dive Computer	External Defibrillator
 •Recalled due to incorrect dive time •Consequences could be deadly	 •Unexpected shutdown •At least 1 unsaved life

How safe do you feel?

- "No problem--we have a fix for that! Just reflash your pacemaker…"

Watchdog Timers – Detecting Software "Hangs"

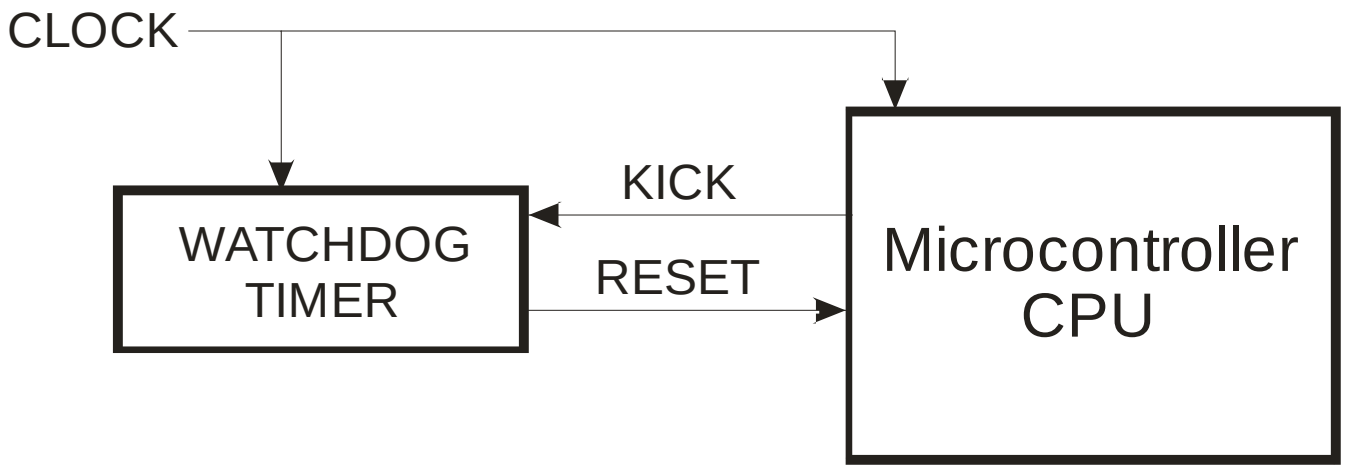
- **A common symptom of software problem – system hang**
  - Could be an infinite loop
  - Could be continually chasing a "wild" pointer around
  - Could be corrupted data
  - … but often systems "lock up" or "hang"

- **Good general-purpose remedy – reboot system if it hangs**
  - But, there is no person around to press "ctl-alt-delete"
  - So, let the watchdog timer do it instead
  - BUT realize this ***doesn't*** solve ***all*** problems
    - just some that are nice to address

- **Basic watchdog idea:**
  - Have a hardware timer running all the time (count-down timer)
  - When timer reaches zero, it resets the system
  - Software periodically "kicks" (or "pets") the watchdog, restarting the count
  - If software has "kicked" the watchdog often enough, no reset takes place

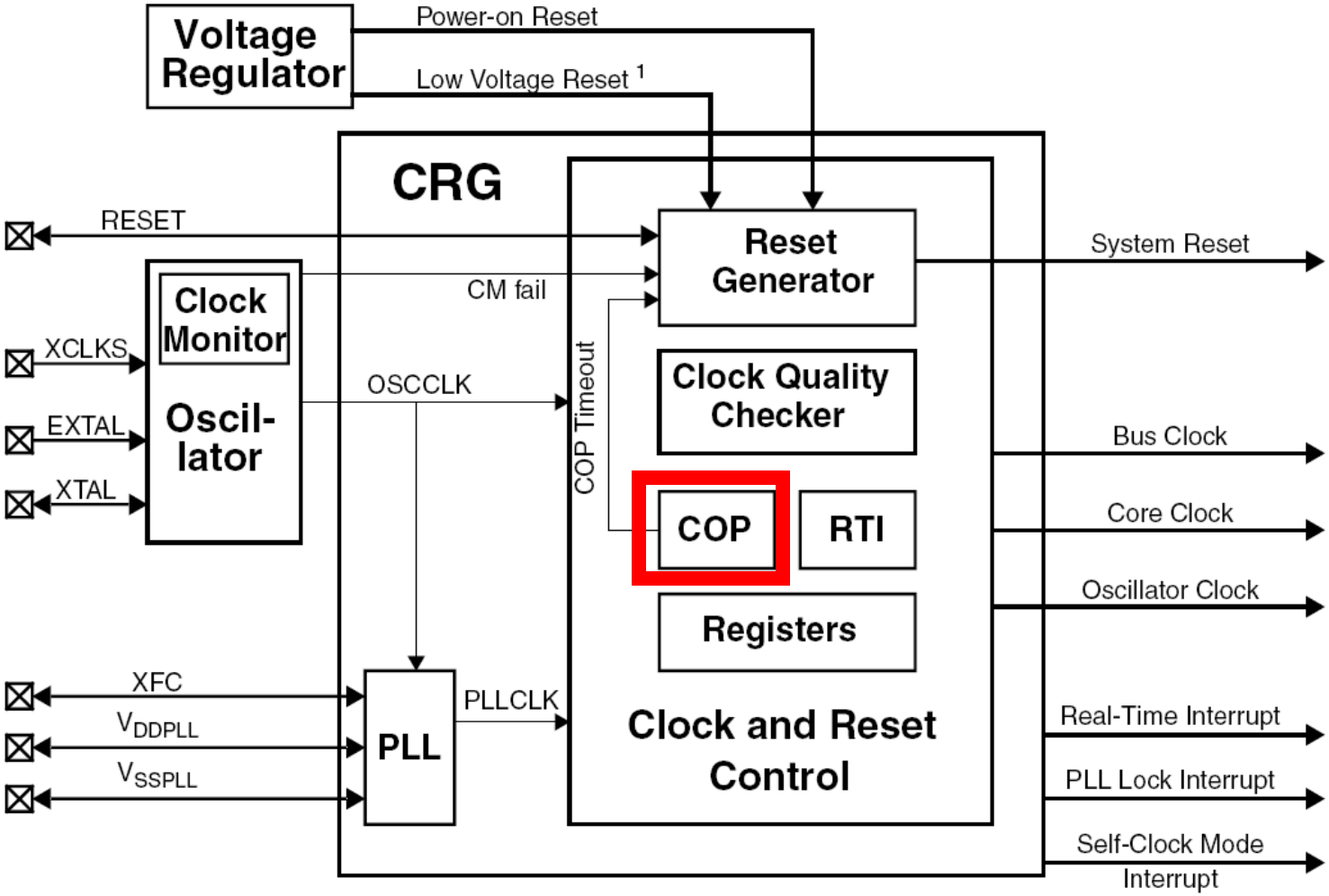
Watchdog General Block Diagram

- **System reset starts the watchdog initially**
  - Clock is used to count-down the watchdog timer
  - Kick restarts the watchdog
  - Watchdog resets CPU when it reaches zero



Course MCU Watchdog ➔ "COP"

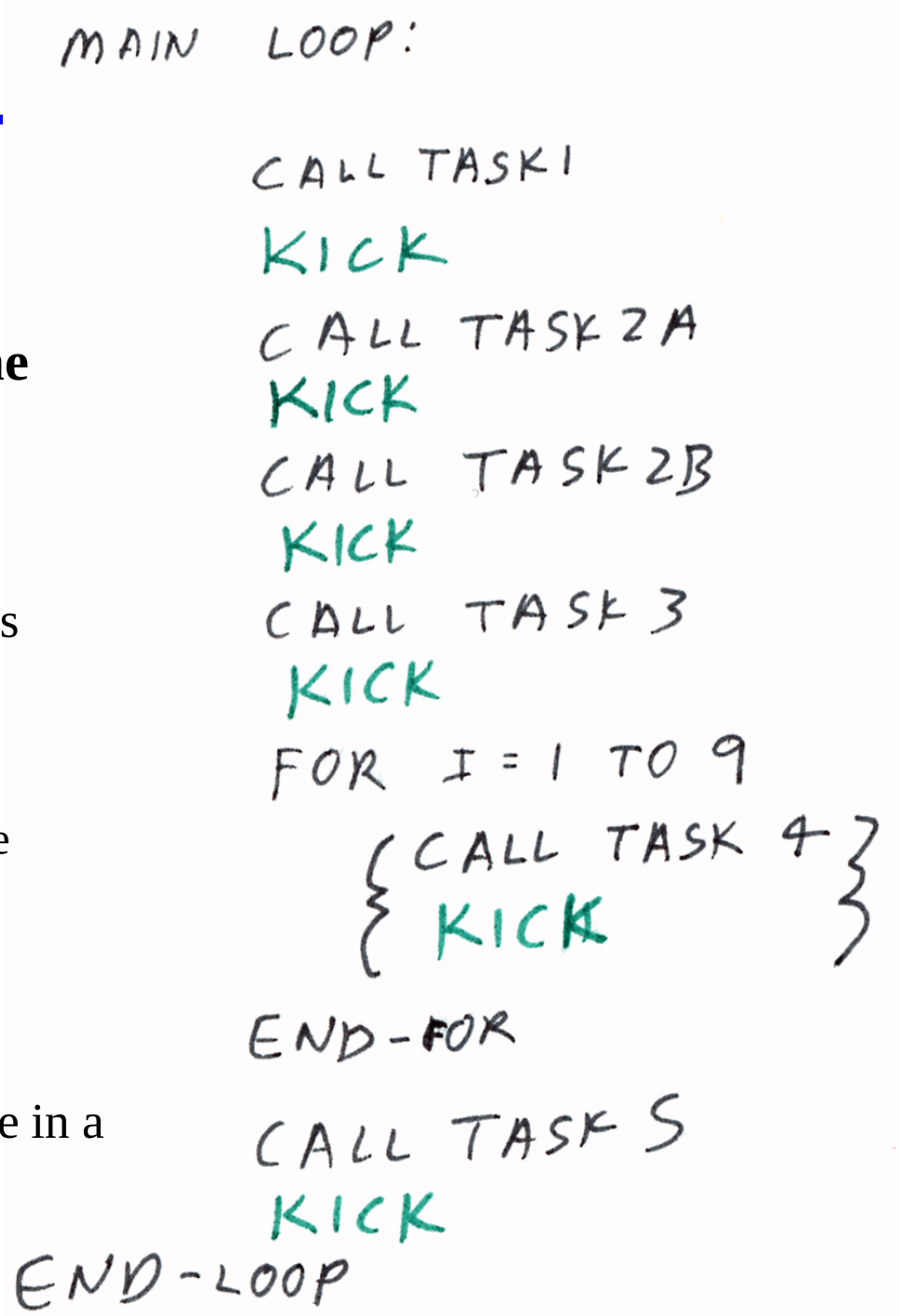
- **See chapter 9 of data sheet – "Clocks and Reset Generator" (CRGV4)**
  - **COP** = "Computer Operating Properly" ➔ Freescale name for watchdog



[Freescale]

When To Kick

- **Kick periodically**
  - Often enough to avoid reset
- **Kick only when doing so means the system is really alive**
  - Between major subroutine calls
  - Only in the ***main*** program loop
  - NEVER within individual task loops
    - Except if you are sure they will terminate (e.g., fixed integer loop bounds)
    - And even then, probably only in the *main* program loop
- **These are basic rules**
  - Advanced topic: with multitasking system, every task should participate in a consensus-based watchdog reset operation



Watchdog Timer Select

- **Set watchdog so that it is fast enough to catch problems quickly**
  - But not so fast you miss it
  - Requires estimate of program execution speed between kicks

2:0 CR[2:0]	**COP Watchdog Timer Rate Select** — These bits select the COP time-out rate (see Table 9-9). The COP time-out period is OSCCLK period divided by CR[2:0] value. Writing a nonzero value to CR[2:0] enables the COP counter and starts the time-out period. A COP counter time-out causes a system reset. This can be avoided by periodically (before time-out) reinitializing the COP counter via the ARMCOP register.

Table 9-9. COP Watchdog Rates(1)

CR2	CR1	CR0	OSCCLK Cycles to Time Out
0	0	0	COP disabled
0	0	1	2^{14}
0	1	0	2^{16}
0	1	1	2^{18}
1	0	0	2^{20}
1	0	1	2^{22}
1	1	0	2^{23}
1	1	1	2^{24}

1. OSCCLK cycles are referenced from the previous COP time-out reset (writing 0x0055/0x00AA to the ARMCOP register)

Petting The Watchdog (Kicking the COP)

9.3.2.12 CRG COP Timer Arm/Reset Register (ARMCOP)

This register is used to restart the COP time-out period.

Module Base + 0x000B

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reset	0	0	0	0	0	0	0	0

Figure 9-15. ARMCOP Register Diagram

Read: always reads 0x0000

Write: anytime

When the COP is disabled (CR[2:0] = "000") writing to this register has no effect.

When the COP is enabled by setting CR[2:0] nonzero, the following applies:

Writing any value other than 0x0055 or 0x00AA causes a COP reset. To restart the COP time-out period you must write 0x0055 followed by a write of 0x00AA. Other instructions may be executed between these writes but the sequence (0x0055, 0x00AA) must be completed prior to COP end of time-out period to avoid a COP reset. Sequences of 0x0055 writes or sequences of 0x00AA writes

NOTE – multi-operation "kick" to reduce chance of random code kicking it

Bad Watchdog Use

- **Kicking inside a single task loop**
  - OK, so that loop is alive, but what about other tasks?
- **Kicking in a great many places in the code**
  - Only kick in the main loop; as few places as possible
  - What if you make a mistake and kick inside a loop?
- **Hooking up a timer interrupt to kick the watchdog**
  - Every time timer rolls over, kick the watchdog
  - Only proves the timer is working, not the main tasks!
  - (There are very special exceptions for multitasking)
- **Watchdog can be defeated by software**
  - HW should prevent watchdog turning off once on
  - HW should prevent masking/disabling the watchdog reset once enabled
  - Watchdog should require sequence of values to “kick”
  - Some systems forget to turn on watchdog

Watchdog Margin

- **Let's say you set the watchdog where you think it should be**
  - You compute expected task execution time
  - In the lab, you never see a watchdog trip
    - Hopefully you don't blame one on something else – make sure they are unmistakable!
  - In the field, the watchdog trips – what happened?
    - Well, obviously something you didn't test
    - ***Maybe*** you set the watchdog too close to the edge!

- **Testing watchdog margin**
  - Change the watchdog divider until it trips
    - Does it trip where you expect? (If not, you don't understand something)
  - Add some time-wasting nop-loops in your code
    - Does it trip where you expect? (If not, you don't understand something)

Multi-Tasking Watchdog

- **Consider a preemptive tasking system**
  - (We'll talk more about preemption later – we just mean "multi-tasking" here)
  - Assume there is a watchdog timer (a COP timer)
  - kick() restarts the watchdog time at initial value

```
void task0(void) { .. Do stuff..; kick(); …more… ;}
void task1(void) { .. Do stuff..; kick(); …more… ;}
void task2(void) { .. Do stuff..; kick(); …more… ;}
void task3(void) { .. Do stuff..; kick(); …more… ;}
```

  - What's wrong with the above approach?

  - (Murphy00 supplemental reading also talks about this issue)

Effective Multi-Tasking Watchdog Approach

```
void task0(void) { .. Do stuff..; Alive(0x1); }
void task1(void) { .. Do stuff..; Alive(0x2); }
void task2(void) { .. Do stuff..; Alive(0x4); }
void task3(void) { .. Do stuff..; Alive(0x8); }
```

- **Main idea – each task sets a bit indicating it has run**
  - Separate watchdog monitor task kicks watchdog only when every task has reported in
  - Needs to be modified to account for task periods, but this is the basic idea

```
static uint16 watch_flag = 0;
void Alive(uint16 x)
{ SEI();                  // disable interrupts
  watch_flag |= x;
  CLI();                  // enable interrupts
}  // set task's "I'm Alive" bit

void taskw(void)   // run periodically
{ if (watch_flag == 0x0F) // if all tasks alive
  { kick();               // kick watchdog
    watch_flag = 0;       // erase flags
}}
```

Review

- **Time of day**
  - Accuracy – time measurement and quantization
  - Drift – due to oscillator speed ***AND*** software inaccuracies
  - Converting a hardware timer to time of day

- **Hardware timer operation**
  - Setting up a timer, including frequency calculations
  - Classic timer mistakes

- **Watchdog timer operation**
  - Setting up the watchdog, including frequency calculations
  - How to ensure a watchdog timer is set properly
  - Rules for good and bad watchdog use
  - Multi-tasking watchdog

Lab Skills

- **Counter/timer**
  - Be able to set, read, and generate time of day from a hardware timer
- **Watchdog timer**
  - Be able to set up and measure effects of watchdog timer