Lecture #25

System Resets; Robustness; Power Management

18-348 Embedded System Engineering

Philip Koopman

Wednesday, 18-April-2016

Electrical & Computer ENGINEERING

Carnegie Mellon

Where Are We Now?

- **Where we've been:**
  - Lots and lots of places
- **Where we're going today:**
  - Loose ends: System resets, robustness, power management, etc.
- **Where we're going next:**
  - Bluetooth & CAN
  - Second exam Wed, 22 April 2015
    - Notes sheet in your own handwriting; NO calculator
    - Same rules and procedures as for Exam #1
  - Final projects hand-ins
    - Last day to demo – Wed of final exam week
    - Last day to hand in report – Thursday of final exam week

Preview

- **Special interrupts**
  - External interrupt pin
  - Is SWI maskable?
  - System traps and NMI
- **System resets**
  - Boot Loader
  - How to get clean system resets
  - Multi-tasking watchdog strategy
- **Improving system robustness**
  - Transient vs. permanent faults
  - Timeouts & retries
- **Power management**
  - Power reduction via voltage and clock frequency change
  - Power reduction via sleeping
  - Major factors in power consumption and battery drain
  - Thermal issues

Another Look At Interrupt Masking

- **We've assumed that the interrupt mask always works**
  - *Almost* always true, but not always true
  - Some interrupts shouldn't be masked – for example "System Reset"(!)

Below are "special" interrupts:

Table 7-1. CPU12 Exception Vector Map

Vector Address	Source
$FFFE–$FFFF	System Reset
$FFFC–$FFFD	Clock Monitor Reset
$FFFA–$FFFB	COP Reset
$FFF8–$FFF9	Unimplemented Opcode Trap
$FFF6–$FFF7	Software Interrupt Instruction (SWI)
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ Signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ Signal

IRQ – External Interrupt ReQuest

- **IRQ.L and XIRQ.L**
  - Connected to external pins for requesting interrupts
  - Active low – better noise margin on TTL high voltage (more noise resistant)
  - IRQ.L is maskable
  - XIRQ.L is **non**-maskable (generically, NMI = "Non-Maskable Interrupt")

- **General XIRQ rules:**
  - **Avoid use of non-maskable interrupts for servicing devices!**
  - Problem is they can re-trigger during ISR, causing stack overflow
  - XIRQ mainly useful for waking chip up from low-power "sleep" mode

- **IRQ is external maskable interrupt**
  - Allows interfacing to I/O devices outside the chip
  - All external chips same the same IRQ pin
  - But, it's also a shared interrupt vector …
    … so software has to poll I/O devices to see which one generated the IRQ

Servicing An IRQ

- **If you get an IRQ, how do you know what caused it?**
  - On-chip devices go to a different interrupt vector per device
  - BUT, only one IRQ pin – multiple off-chip devices share the single IRQ vector
- **IRQ servicing done via an oxymoron: "Interrupt Polling"**
  1. Interrupt is received via IRQ
  2. ISR uses I/O commands to read status register from each external device
     - Poll each of them – did you cause this interrupt? How about you? Or You? …
  3. When you find a device that caused the interrupt, execute appropriate ISR
     - Probably done via a subroutine call to appropriate handler from within IRQ ISR
     - IRQ might still be active when you are done due to a different external device!
     - But, that's OK; after RTI it just re-starts the polling loop → go back to step 1
  - Which interrupt do you service?
    - Can do prioritized (poll in same order every time – this is the usual case)
    - Can do round-robin (remember last interrupt, and start polling from there)
    - In effect, the IRQ service routine is running a task switcher to pick next IRQ source to service; **make sure you don't infinite loop if IRQ was a noise glitch**

SWI, Trap & Background Mode

- **SWI is non-maskable**
  - Primary use is for debugging and single-stepping
  - If it were maskable, you couldn't single-step through ISRs!
  - A fine point – this is a big reason why RTI restores interrupt mask via CCR restore instead of just always clearing it with an RTI
    - What would happen if SWI took place during an ISR (for debugging) AND then RTI cleared the interrupt mask instead of restoring it?
- **Unimplemented Opcode Trap**
  - Non-maskable – fatal program error
  - Don't want an ISR to keep executing if it has executed an illegal opcode!
- **BGND instruction (opcode 00 – "Background Mode")**
  - Variant on the above – invokes hardware supported breakpoint debugger
  - If your code goes wild and crashes, often you stop at opcode 00
  - (Why 00? The value 00 is a very common value in data. So you will often enter breakpoint mode if code goes wild and tries to execute data.)

How Do You Reset Your System?

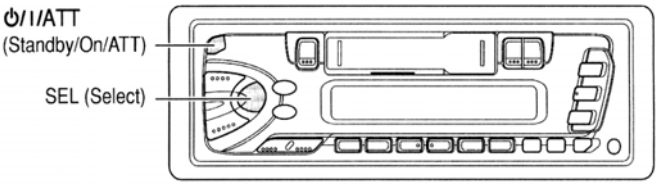
- **Generally, it is important to have a way to reset the system!**
  - Debate – do you have to unscrew the cover plate to reset your system?
    - A reset button on a thermostat tells the customer you don't trust your own software
    - No reset button on a thermostat ticks off the customer when they disassemble to reset it (assuming it is running on remote power and not batteries)
- **Common reset methods:**
  - Hardware reset button connected to reset pin
  - Soft reset button(s) (what if the software that reads the button crashes?)
    - PCs typically have an embedded microcontroller running the power system
    - Assume that embedded micro doesn't crash; it can reboot the Intel main CPU
  - Remote reset for off-site maintenance support
    - New cable TV/set-top boxes can be reset remotely when you call in for help
  - Cycle power; remove batteries; etc.
- **Good news: Microsoft has trained people to reboot if there is a crash**
  - Bad news: if people reboot instead of complaining, you won't know about problems that might get worse in an unusual situation!

JVC

CASSETTE RECEIVER KS-F150

How to reset your unit

Press and hold both the SEL (Select) and ⏻/I/ATT (Standby/On/ATT) buttons at the same time for several seconds.
This will reset the built-in microcomputer.
NOTE: Your preset adjustments — such as preset channels or sound adjustments — will also be erased.



(This is for real! Manual says resetting is normal and does not indicate a defect.)

Reset Interrupts

- **Interrupts that are designed to reset the system**
  - "System Reset" –"Clock Monitor Reset" – "COP Reset"
  - NOT maskable
  - They do have interrupt vectors, but make sure you don't abuse this!

- **Reset.L – external pin active low**
  - Commonly used Power On Reset circuit – Note the Schmitt Trigger!
    - But can miss very small power glitches or rapidly cycling power on/off
      - » Reset based on un-filtered power
      - » Make sure you have a big power filter cap to ride through small glitches
    - Get a good analog designer for this item if this is a worry for you…
      Or use a CPU (such as the course CPU) that has this on-chip!

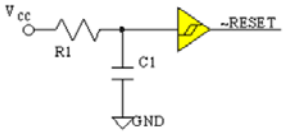
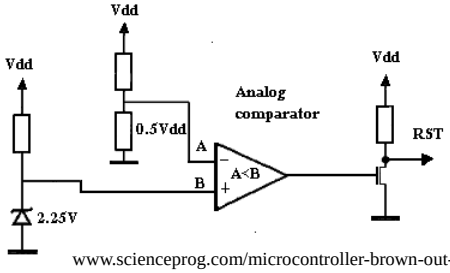


Figure 1— This power-on-reset circuit is simple but flawed.

http://www.sigcon.com/Pubs/edn/por.htm

Brown-Out Detection

- **What if power supply voltage dips?**
  - Could happen due to marginal battery, sudden high current draw
  - CPU operation will not be reliable if voltage is below specification!
- **Hold-up cap can help a little**
  - Big capacitor can smooth bumps in power supply
  - But, capacitor won't help if battery voltage is marginal but too low!
- **Solution: always enable brown-out detection!**
  - Brown-out means slightly lower than minimum specified voltage
  - Example brown-out detection circuit:

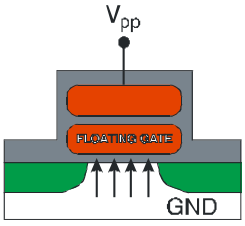


www.scienceprog.com/microcontroller-brown-out-detection/

Course CPU has "Low Voltage Detect" (LVD) and "Low Voltage Reset" (LVR) which handle this

Avoiding EEPROM Corruption

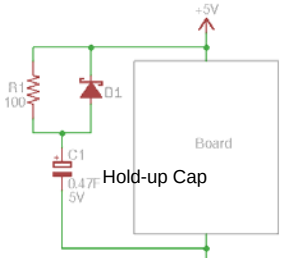
- **Remember flash/EEPROM memory writing?**
  - Repeated high voltage pulses to shoot electrons onto a glass-insulated gate
  - If you have too low a voltage you get a weak write or just garbage data
  - Flash memory has a similar problem



Program

- **Ways to ensure EEPROM isn't corrupted**
  - Read it back after you write a value
  - Use "finished" hardware line instead of assuming fixed time to write
  - Use brownout protection – EEPROM minimum voltage > MCU minimum!
  - Use a hold-up capacitor for loss of power mid-write
  - Don't ever use address zero of EEPROM (serial EEPROM default power-up address)
  - Watch out for wearout – limited number of writes
  - Consider error coding (CRC) on data written to detect wearout and weak writes

http://betterembsw.blogspot.com/2011/11/avoiding-eeprom-corruption-problems.html



ruggedcircuits.com/html/circuit__13.html

Boot Loader – What Happens At Power On?

- **In small systems entire program is in flash memory**
  - Booting is simply init to configure all the I/O etc. and start operation
- **In bigger embedded system with an RTOS at power on…**
  - There is no operating system loaded, no file system, lots of stuff missing
  - Need to load RTOS etc. from a disk or serial flash memory
- **Booting (booting up; bootstrapping)**
  - From "to pull yourself up by the bootstraps"
  - Load some initial program just smart enough to load other programs
    - Usually it is already in flash memory waiting to run
    - For example, smart enough to read sector 0 of hard disk or bulk flash …
      … sector 0 of hard disk knows how to read OS image from hard disk
      … OS image from hard disk knows how to start user programs
      … etc.

http://blogs.independent.co.uk/wp-content/uploads/2011/05/20a-02.jpg

Good Practices For System Booting

- **Validate image using secure digital signature (avoid Trojan Horse updates)**
- **Do a system self-test**
  - RAM test (can you read/write ones and zeros to all bytes?)
  - Flash integrity check
    - Compute CRC across flash and check for match to stored CRC value
  - Do timers seem to be timing? A/Ds converting? Etc?
  - Do external pins seem to be working (can you talk to all external devices?)
  - Is system you are controlling healthy and ready to run?
  - It's hard to be thorough, but try to do the basics (use a vendor library if available)
- **Put system in known, defined state**
  - Put all I/O in known, defined state, even if you aren't using it
  - Turn off devices you aren't using to save power
  - Start watchdog timer
  - Make sure all outputs are in a ***safe*** state ("safe" is system dependent)
    - Ideally, HW reset automatically makes all outputs safe!
  - Etc.
  - Some designers periodically reset hardware to known state in case transient fault flips direction of I/O pin, etc.

Example: IEC60730/60335 Self-Test

- **Recent standard used with UL-1998 to self-test critical components**
  - For example, CPU-actuated circuit breaker

Group	Test Components
Microcontroller specific	CPU registers
	CPU program counter
	Clock
	Volatile and non-volatile memories
	Internal addressing (and external memory of any)
	Internal data path
Application specific	Interrupt handling
	External communication
	Timing
	I/O peripherals
	Analog ADC and DAC
	Analog multiplexer

Component	Error	Method	Definitions as per Annex H of IEC 60730	In Self-Test Library
1. CPU 1.1 Register	Stuck at	Static memory test	H. 2.19.6	YES
1.3 Program counter	Stuck at	Logical monitoring of the program sequence	H.2.18.10.2	YES
2. Interrupt	No interrupt or too frequent interrupts	Time-slot monitoring	H.2.18.10.4	YES
3. Clock	Wrong frequency	Frequency monitor	H.2.18.10.1	YES
4. Memory 4.1 Invariable memory	All single-bit faults	Word protection with multi-bit redundancy	H.2.19.8.1	YES
4.2 Variable memory	DC fault	Static memory test	H.2.19.6	YES
4.3 Address (Related to 4.1 and 4.2)				
5. Internal data path (Only with external memory)	-	-	-	NO*
6. External communication (Not involved in STL)	Hamming distance 3	Word protection with multi-bit redundancy	H.2.19.4.2	NO*
7. Input/output periphery 7.1 Digital I/O	Function error	Input comparison Output verification	H.2.18.8 H.2.18.12	YES YES
7.2 A/D	Function error	Input comparison	H.2.18.8	YES

**Involves tests external to MCUs See main text.*

http://www.eetimes.com/design/microcontroller-mcu/4218143/Guidelines-for-obtaining-IEC-UL-60730-certification-for-self-test-library-implementations

How Much Do You Reboot?

- **Small system – reset pin may do hard reset of entire system**
- **Large system – may have several levels of reboot**
  - Reset some I/O mechanism and related driver
  - Kill a particular task; application has to deal with it
  - Kill and automatically restart a task
  - Kill and automatically restart the RTOS
  - How hard you reboot the system depends on how wrong things have gone
    - And how long you can wait before the plant becomes unstable or dangerous
- **Software rejuvenation**
  - Idea that you periodically reboot system to clean out accumulated errors
    - Maybe you kill and restart tasks; maybe the entire system
  - Works for software defects such as memory leaks
  - But, doesn't solve downright incorrect software that isn't fixed by rebooting

Netrino

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…"

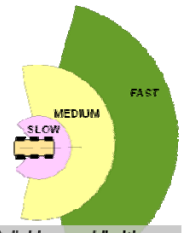
APD ("son-of-crusher")

APD Safety System

Objective: Enforce and control safe standoff distance between APD and nearby personnel.

Approach:

- Provide fail-safe braking mechanisms with well-modeled stopping distance.
- Incorporate Safety Monitor for redundant, high-reliability means of restraining vehicle speed.
- Identify and mitigate risks that could lead to failures of braking and speed-limiting.

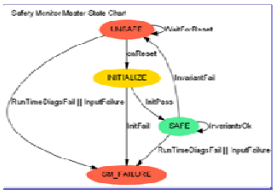


Reliable speed limiting allows safe standoff distances to be decreased

Techniques:

- Identifying hazards that lead to safety mishaps.
- Modeling of correlation between latent hazards with rich instrumentation.
- Firewalling safety-criticality to a subset of vehicle components.
- Developing & testing fault-resistant software for speed limiting.
- V&V testing traced to safety requirements.

Careful analysis of mishaps drives safety system design



Safety Monitor ensures that safety invariants are maintained

TECHNOLOGY DRIVEN. WARFIGHTER FOCUSED.

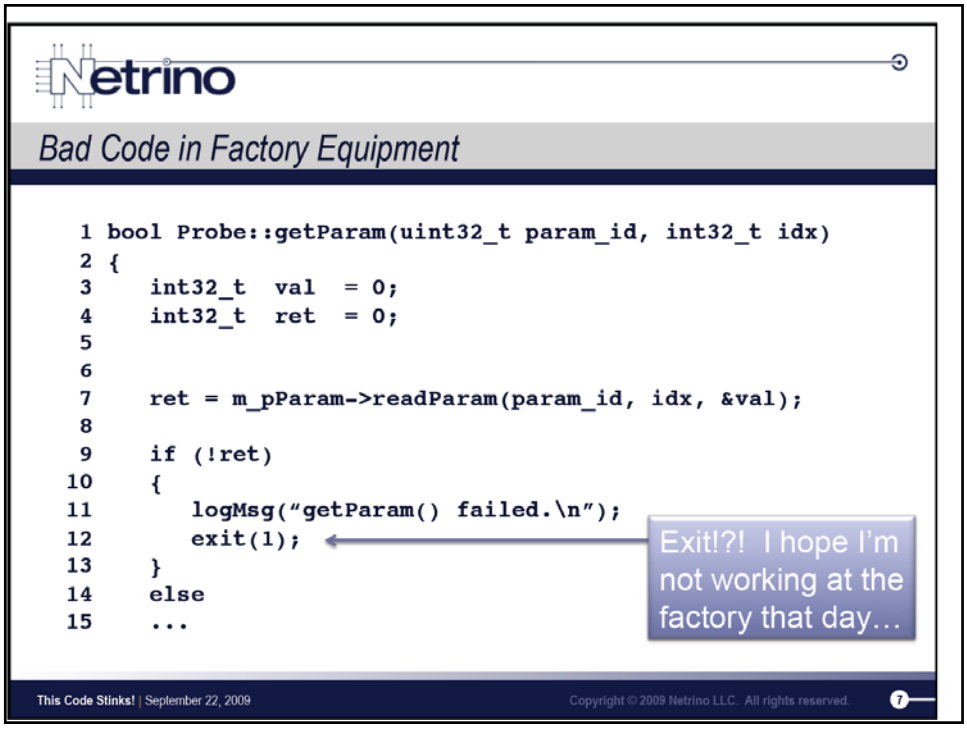
Robustness Resets

- **COP reset, Clock monitor reset, unimplemented opcode reset**
  - Are all there to help reboot system when a problem occurs
- **Simple approach:**
  - When reset occurs, simply do a cold start of the system
  - Sometimes this works, but sometimes isn't enough!
- **Problem: "yoyo" mode**
  - Reboot only works if the error condition clears itself
  - If error condition persists, system will keep rebooting
  - Safety vulnerability in yoyo mode
    - What if a "dead" system is safe?
    - And a "live" system is safe?
    - But control loops are left running open-loop during boot?
    - And, system continually reboots?

[Wikipedia]

Useful Reset Strategies/Patterns

- **Reset a limited number of times**
  - Permit system to reset a limited number of times within a time interval
  - Accomplish by setting a flag for "I was just reset" in EEPROM
  - Clear the "I was just reset" flag only after a time delay
  - If reset again within time delay, turn off instead of resetting
    - (prevents yoyo mode)
  - Alternate: count number of resets total in EEPROM, suicide when too many
- **Never reboot**
  - Go into shutdown and require maintenance call to restart
  - Can be prudent for safety-critical systems, but service calls are expensive!
  - Alternate: reset to a very simple "limp along" mode until service call
- **Important design guidance**
  - Always log, track, or indicate resets so you know they happened!
  - Ask yourself what happens to system during reset process



Reset In Reactive Systems

- **Simply resetting software usually *isn't enough***
  - The system is in some physical state
  - The physical state isn't changed by the fact the CPU reset!
  - What if your car's power steering controller resets to "off" at 100 kph?

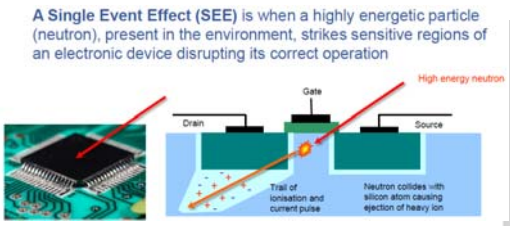
- **General strategies to get CPU back in synch with physical system:**

1. **Software collects system state after reset**
   - Query every sensor in system, figure out what is going on
   - Re-prime control loops so ID parts of PID don't bump system outputs

2. **Software forces system into known state after reset**
   - Do a "system shutdown" of the plant and restart when software resets
   - Can be safer, but also creates system down time!
     - Do you want an entire petro refinery to shut down, ever? (No, you don't.)

3. **Ignore the reset problem**
   - Only works sometimes … usually risky!
   - This is a bad strategy (even if people do it)

Bit Flips Due To Hardware Faults



Radiation strike causing transistor disruption (Gorini 2012).

(Constantinescu 2003)

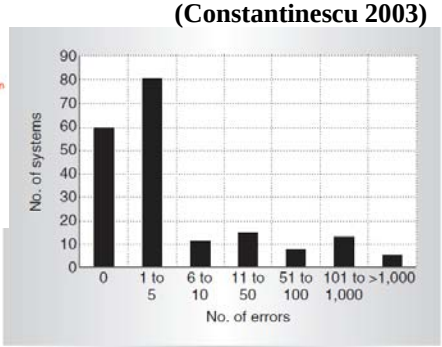


Figure 2. Histogram of the number of memory single-bit errors reported by 193 systems over 16 months.

- **Hardware bits flip values due to radiation strikes ("soft errors")**
  - Affects memory, control logic, CPU registers – everything on a chip
- **Common techniques only help with memory, not CPU:**
  - Mirroring (store two copies of data in memory)
  - Error Detection & Correction Codes
- **Software can also produce transient faults**
  - E.g., memory corruption due to wild pointers

Dealing With Transient Faults

- **Most faults are transient**
  - They occur randomly
  - They don't persist – they clear all by themselves
    - Transient faults are often 10x to 100x more frequent than permanent faults
  - Hardware transients – radiation causes bit upsets; power supply noise; lightning
  - Software transients – bad pointers, occasionally missed deadlines, timing jitter
  - Phantom interrupt – noise on interrupt line but no interrupt really there
- **Robustness for transient faults – you might use all of these in a single system**
  - First strategy – try again!
  - Second strategy – timeout
  - Third strategy – reboot system to try again
    - Perhaps back off one notch on watchdog time interval? (depends on system)
  - Fourth strategy – realize when trying again isn't working
    - Some faults are *permanent*
  - *Important note:* it is difficult to know a robust system is in trouble unless you make special effort to log and record problems!

Timeouts & Error Checking

Timeout is a good way to balance transient fault tolerance with detection of permanent faults

- **Code that is just waiting for an error to cause problems:**

```
serial_byte_out(b);   // what if there is an error?
```

- **Non-robust code**

```
if(!serial_byte_out(b)) {log_error(…;)}
```

- **Code robust to transient faults, but vulnerable to permanent faults**

```
while(!serial_byte_out(b)) {log_error(…);}
```

- **Code that handles both transient and permanent faults**

```
retry=0;
while(!serial_byte_out(b))
{ log_error(…);
  if (retry++ > 100) {permanent_fault(…); break;}
}
```

Remember This? Multitasking Watchdogs

- **Consider a preemptive tasking system**
  - Assume there is a watchdog timer (a COP timer)
  - kick() restarts the watchdog time at initial value

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

  - What's wrong with the above approach?

Better 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

```
uint16 watch_flag = 0;
void Alive(uint16 x)
{ DisableInterrupts();        // Why do we need to do this?
  watch_flag |= x;
  EnableInterrupts();
}  // set task's "I'm Alive" bit

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

Low Power Basics

- **CPU power = dynamic + static**
  - Dynamic power consumed every time a CMOS gate turns on or off
  - Static power is leakage current when gate is on but doesn't switch
- **Dynamic power = C * V^2 * f ; Static power = Leakage * #gates**
  - C = capacitance being switched (roughly, based on size of gates + IC wires)
  - V = operating voltage
  - f = switching frequency
- **Power proportional to V^2**
  - All things being equal, 2.5V consumes 25% as much power as 5V power
- **Power proportional to 1/f**
  - Slower frequency linearly reduces power
- **To reduce power:**
  1. Lower voltage
  2. Reduce frequency
  3. Don't clock unused portions of chip to reduce C
  4. Turn off power to unused portion of chip to reduce number of gates with leakage

Course CPU: Low Frequency Operation

- **Course CPU runs down to 0.25 MHz bus frequency**
  - How much power does that save compared to 25 MHz?

Table A-4. Operating Conditions

Rating	Symbol	Min	Typ	Max	Unit
I/O, Regulator and Analog Supply Voltage	V_{DD5}	2.97	5	5.5	V
Digital Logic Supply Voltage (1)	V_{DD}	2.35	2.5	2.75	V
PLL Supply Voltage 1	V_{DDPLL}	2.35	2.5	2.75	V
Voltage Difference V_{DDX} to V_{DDA}	Δ_{VDDX}	–0.1	0	0.1	V
Voltage Difference V_{SSX} to V_{SSR} and V_{SSA}	Δ_{VSSX}	–0.1	0	0.1	V
Bus Frequency	f_{bus} (2)	0.25	—	25	MHz
Operating Junction Temperature Range	T_J	–40	—	140	°C

1. The device contains an internal voltage regulator to generate the logic and PLL supply out of the I/O supply. The operating conditions apply when this regulator is disabled and the device is powered from an external source.
   Using an external regulator, with the internal voltage regulator disabled, an external LVR must be provided.
2. Some blocks e.g. ATD (conversion) and NVMs (program/erase) require higher bus frequencies for proper operation.

Course CPU: Low Voltage Operation

- **External voltage range is 2.97V to 5.5V**
  - Internal voltage regulator supplies 2.5V to chip
  - You can supply 2.5V direct to V_{DD} to reduce 5V→2.5V step-down power losses
  - Analog conversion assume 5V interface levels
    - LVI triggers when they are impaired due to low voltage
    - Internal digital parts of CPU work just fine down to V

A.7.1 Voltage Regulator Operating Conditions

Table A-23. Voltage Regulator Electrical Parameters

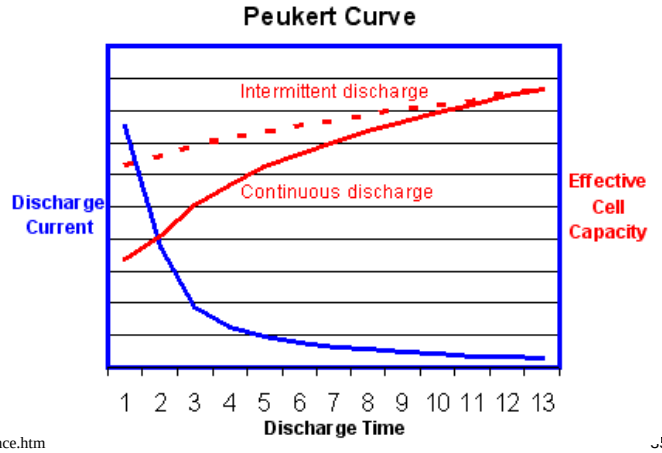
Num	C	Characteristic	Symbol	Min	Typ	Max	Unit
1	P	Input Voltages	$V_{VDDR, A}$	2.97	—	5.5	V
3	P	Output Voltage Core Full Performance Mode	V_{DD}	2.35	2.5	2.75	V
4	P	Low Voltage Interrupt[1] Assert Level (xL45J mask set) Assert Level (other mask sets) Deassert Level (xL45J mask set) Deassert Level (other mask sets)	 V_{LVIA} V_{LVIA} V_{LVID} V_{LVID}	 4.30 4.00 4.42 4.15	 4.53 4.37 4.65 4.52	 4.77 4.66 4.89 4.77	 V V V V
5	P	Low Voltage Reset[2], [3] Assert Level (xL45J mask set) Assert Level (other mask sets)	V_{LVRA}	 2.25 2.25	 2.3 2.35	 — —	V
7	C	Power-on Reset[4] Assert Level Deassert Level	V_{PORA} V_{PORD}	0.97 —	— —	— 2.05	V V

Course CPU: Stopping The Clock To Save Power

- **Stop instruction: "STOP"**
  - Pushes CPU information onto stack (same format as any interrupt)
  - Full stop: halts all clocks; minimum power consumption mode
  - Pseudo-stop: COP (watchdog) keeps running (configure chip for Full or Pseudo)
  - BTW: "S" flag in CCR is "ignore STOP opcodes" defaults to "1"
    - STOP circumvents watchdog!
    - If you enable STOP, decide if you should switch to Pseudo-Stop enabled as well
- **Wait instruction: "WAI"**
  - Pushes CPU information on stack (same format as any interrupt)
  - System clock still runs, but nothing happens
  - Less power than NOP wait loop (no transistors switching in CPU); more than STOP
- **Both STOP and WAI restart on reset or interrupt**
  - STOP takes longer to restart because have to wait for oscillator stability
  - Usually use unmasked interrupt to avoid masking the wakeup function
- **Reaction of other functional blocks (e.g., timers) to STOP/WAI varies**
  - Read data sheet carefully to find out all the details (especially Ch. 9 on clocking)
  - You can turn off many blocks to save power via software (not automatic in HW)

Minimize Current To Maximize Battery Power

- **Battery capacity depends on charge/discharge rates**
  - Slow charge & discharge gives better capacity ("1c" = 1 hour discharge rate)
  - Effective battery capacity goes down with increasing current
    - This means that battery life extension is ***more than linear*** with power reduction
  - (Note: batteries self-discharge over time, which limits life as well)



http://www.mpoweruk.com/performance.htm

Thermal Issues: Power ➔ Heat

- **Embedded systems may have thermal constraints:**
  - No fan
    - Too much weight, power, noise
  - No air exchange outlet
    - How do you keep mud out of the air vent?
  - Temperature limits –temperature of exposed surface matters for human touch
    - Pain at perhaps 106-108 degrees F
    - Human skin starts scalding at about 110 degrees – hours of exposure to get burn
    - Risk gets more acute at 120 degrees – a few minutes exposure to burn in hot water
- **With a sealed unit, heat transfer is proportional to:**
  - Temperature difference between case temp and outside temp
  - Surface area of case
  - Heat transfer efficiency of case (Is it insulated? Is wind blowing across it?)
  - For wearable computers, this can be a huge power limitation
    - Part of computer might be insulated or against a 98.6F human body
    - One solution: put a block of soft wax inside and melt wax during operation!

Lower Power System Summary

- **Static techniques**
  - Minimize voltage –savings proportional to V^2
    - Separate digital power supply at lower voltage than analog power supply
  - Minimize frequency – savings proportional to f
  - Disable clock to portions of chip not being used – savings proportional to C (# gates)
  - Power down portions of chip not being used – savings proportional to C (# gates)
  - Minimize current drawn – battery life gain proportional to $I^{\sim 1.3}$
- **Dynamic techniques**
  - Go to sleep or minimize current when not busy (STOP, WAI)
    - Definitely go to sleep if you are computing for 1 second out of every 10 minutes
    - But, might be better to go slow continuously rather than very fast in bursts
      - » Depends on complex tradeoffs if "off" times aren't extremely long due to non-linear Peukert curve
  - Dynamic voltage & frequency scaling
    - Increase voltage just enough to allow faster frequency to meet deadlines
  - Turn off parts of chip if not being used, even if CPU is running
- **Low power extends battery life AND reduces heat problems**

Review

- **Special interrupts**
  - External interrupt pin
  - Is SWI maskable?
  - System traps and NMI
- **System resets**
  - Boot Loader
  - How to get clean system resets
  - Multi-tasking watchdog strategy
- **Improving system robustness**
  - Transient vs. permanent faults
  - Timeouts & retries
- **Power management**
  - Power reduction via voltage and clock frequency change
  - Power reduction via sleeping
  - Major factors in power consumption and battery drain
  - Thermal issues