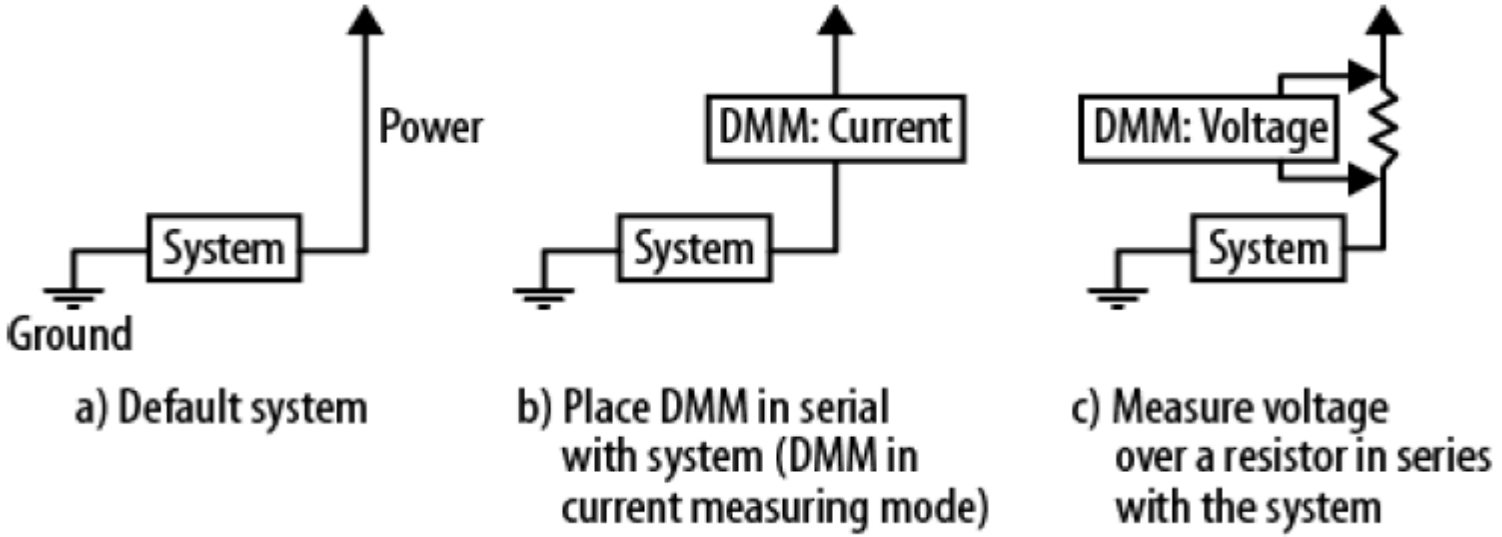
Reducing Power

Chapter 10

Power Consumption

- Power = Voltage x Current
  - Unit
    - Voltage: volt
    - Current: ampere
    - Power: watt
- Energy = Power x Time
  - Unit
    - Power: watt
    - Time: second
    - Energy: joul
- Problems/issues:
  - Heat: determined by power
  - Battery: determined by energy

Power Measurement



a) Default system

b) Place DMM in serial with system (DMM in current measuring mode)

c) Measure voltage over a resistor in series with the system

Expected current	Resistor in circuit	Voltage measured on DMM	Note
12 mA	1 Ω	0.012 V	Easily measured on most DMMs.
23 µA	1 Ω	0.000023 V	Impossible to read on most DMMs.
34 µA	10 Ω	0.00034 V	A very good DMM might be able to read this.
45 µA	100 Ω	0.0045 V	Once again, within reach of most DMMs, but resistor is getting to be too large.

Turn Off Components

- When components are not needed.
- Turn off peripherals
  - Leds off
  - Sensors off
- Turn off unused IO pins
  - Set IO pins to input with internal pulldown
  - Set IO pins to tristate
- Turn off processor subsystems
  - SPI off
  - USART off

Slow Down

- Clock switching
  - Fast and slow clock sources
- Frequency scaling
  - Slow clock means smaller power (not less energy).
  - DVS: dynamic voltage scaling to scale frequency
  - Assume all tasks are known with worst execution time Ci and period Pi under maximum frequency f_M.
  - Tasks are schedulable at f_M without DVS if $\sum \frac{C_i}{P_i} < 1$
  - Tasks are schedulable at f_s with DVS if $\sum \frac{C_i}{P_i} < \frac{f_s}{f_M}$

Slow Down (Example)

- Four events under Fmax=20MHz
  - C1=10ms, P1=200ms
  - C2=5ms, P2=20ms
  - C3=1ms, P3=50ms
  - C4=4ms, P4=100ms
- Schedulable?
  - sum(Ci/Pi) = 0.36 < 1
- Lower frequency?
  - Fmax=20MHz
  - Fmin=7.2MHz

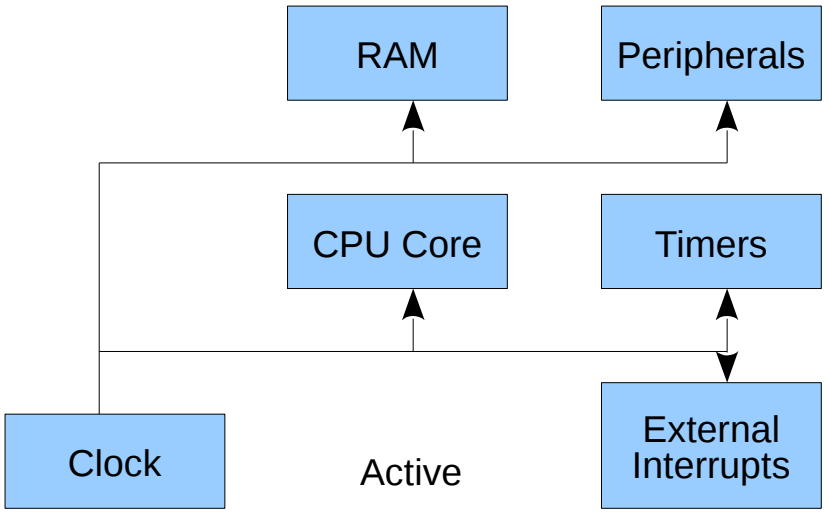
Sleep

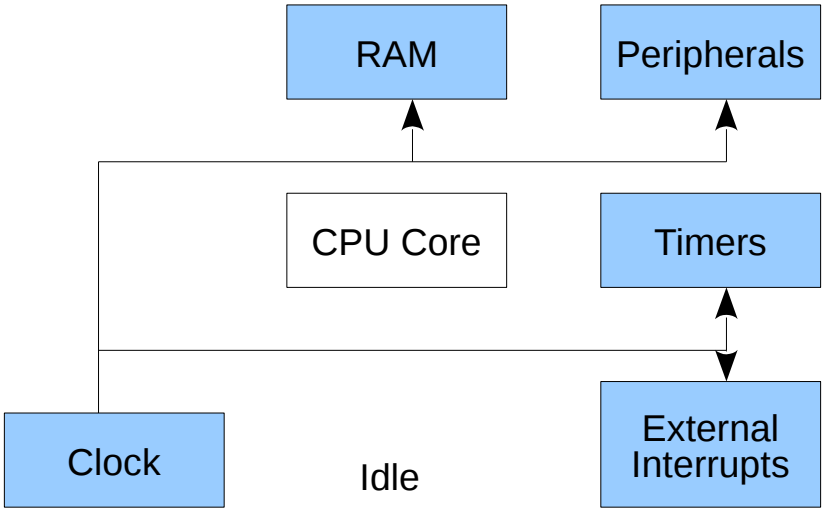
- Common sleep configuration
  - Slow down
    - Slow the clock down to hundreds of hertz.
  - Idle or sleep
    - Turn off the processor core but keeps enabled timers, peripherals, and RAM alive.
  - Deep sleep or light hibernation
    - In addition to the processor core being disabled, some (possibly all) on-chip peripherals can be configured to turn off.
    - Subsystem that generates the interrupt is on to wake up the processor.

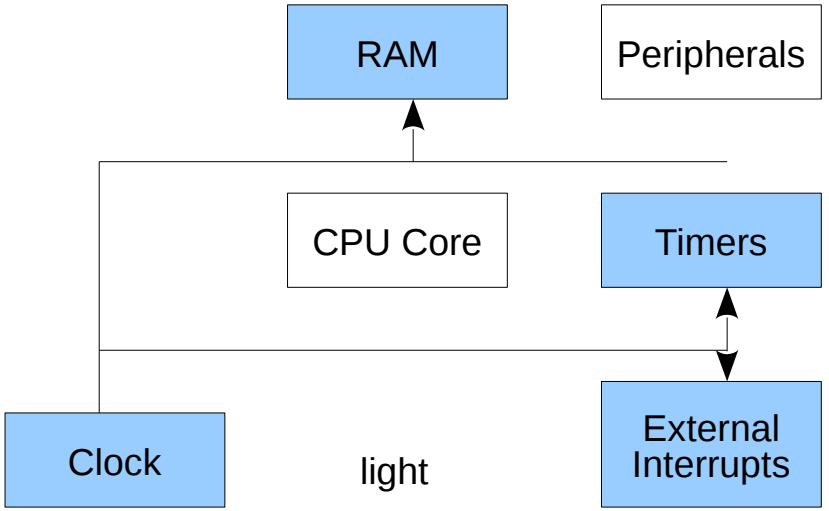
Sleep

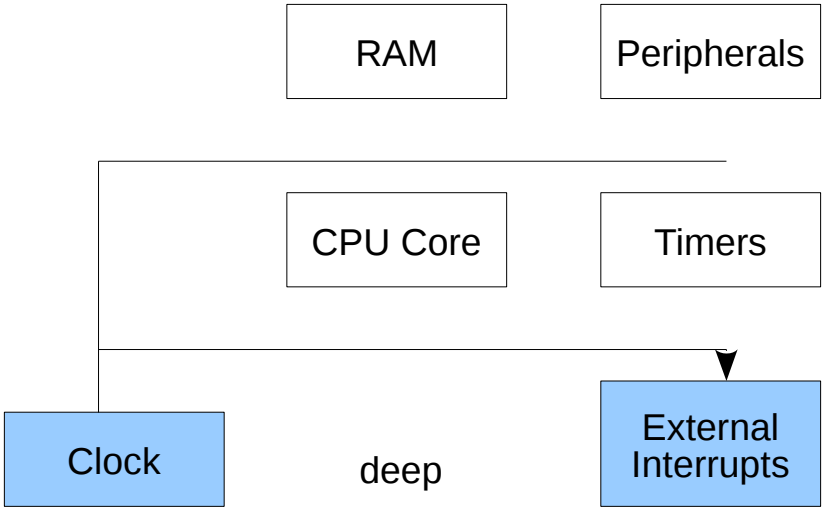
- Common sleep configuration
  - Deep hibernation or power down
    - The processor chooses which on-chip peripherals to be turned off (usually almost all of them). RAM is usually left in an unstable state.
    - Processor registers are retained, so the system doesn’t need initialization on boot.
    - A wakeup pin or a small subset of interrupts can restart the processor.
  - Power off
    - The processor does not retain any memory, and starts from a completely clean state.

Sleep


RAM
Peripherals
CPU Core
Timers
Clock
Active
External Interrupts


RAM
Peripherals
CPU Core
Timers
Clock
Idle
External Interrupts


RAM
Peripherals
CPU Core
Timers
Clock
light
External Interrupts


RAM
Peripherals
CPU Core
Timers
Clock
deep
External Interrupts

Sleep

- Issues with sleep
  - Wake up latency
  - Wake up mechanisms
- Deeper sleep modes place the processor into a lower-power state but take longer to wake from, increasing system latency.
- The wake-up interrupts could be buttons, timers, other chips (e.g., ADC conversion complete), or traffic on a communications bus.

Power Saving: Examples

- Atmega128
  - Idle
  - ADC noise reduction
  - Power-down
  - Power-save
  - Standby
  - Extended standby
- Pic24ep
  - Clock switching
  - Idle
  - Sleep
  - Doze
  - Peripheral disable

ATMega128

- MCUCR register (page 44)
  - Sleep enable: SE
  - Sleep mode: SM2..0
- #include <avr/sleep.h>
  - http://www.nongnu.org/avr-libc/user-manual/group__avr__sleep.h
  - Sleep mode macros
    - avrgcc\avr8-gnu-toolchain\avr\include\avr\sleep.h
  - Sleep functions

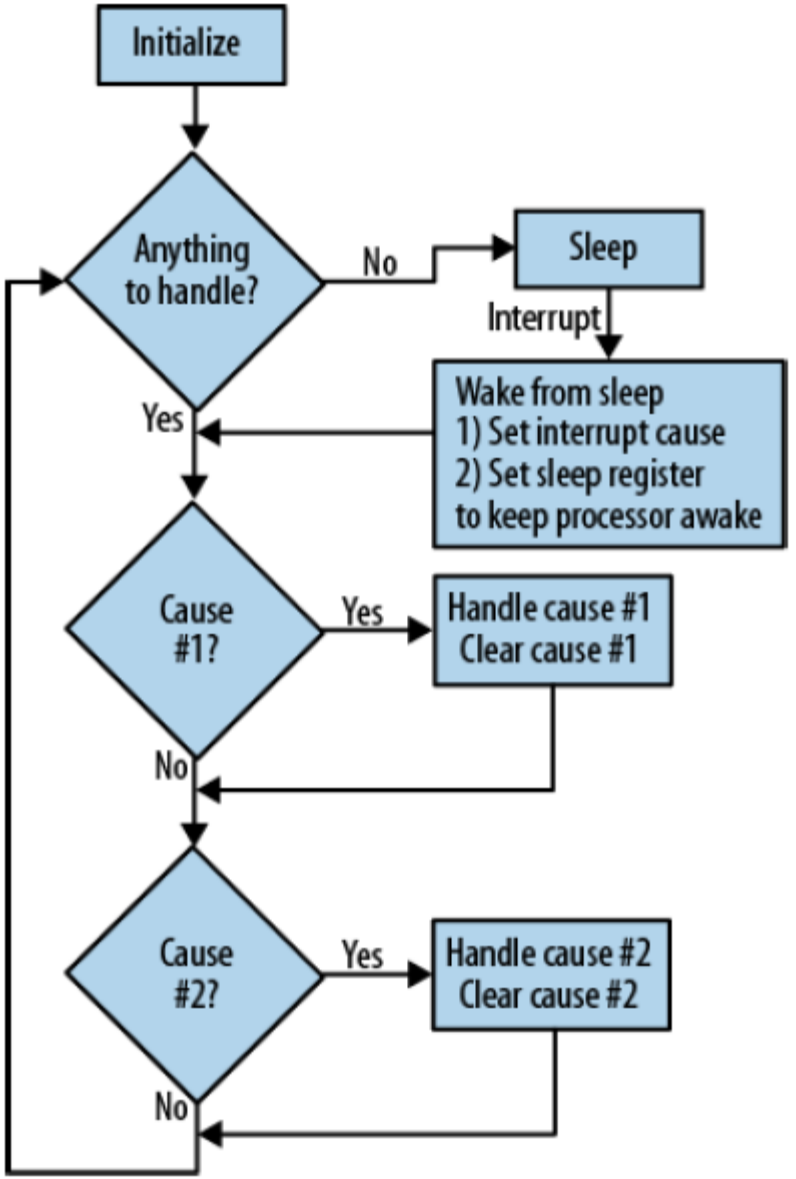
Table 17. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

PIC24E

- Clock switching
  - Clock frequency (prescaling and postscaling) in clock control registers
- Instruction-based
  - Idle and sleep bits in reset Control register
  - Idle()
  - Sleep()
- Doze
  - Doze bits in clock control registers
- Peripheral disable
  - Peripheral module disable bits in peripheral module control registers

Interrupt-based Code Flow



Interrupt-based Code Flow

```
volatile uint16_t gInterruptCause = 0; // Bitmasked flags that describe
                                       // which interrupt has occurred

void main(){
    uint16_t cause;
    Initialize();
    while (1) {
        cause = gInterruptCause; // atomic interrupt-safe read of gInterruptCause
        while (cause) {
            // each handler will clear the relevant bit in gInterruptCause
            if (cause & 0x01) { HandleSlowTimer();}
            if (cause & 0x02) { HandleADCConversion();}
            FeedWatchdog(); // while awake, make sure the watchdog stays happy
            cause = gInterruptCause;
        }
        // need to read the cause register again without allowing any new interrupts:
        InterruptsOff();
        if (gInterruptCause == 0) {
            InterruptsOn();
            GoToSleep(); // an interrupt will cause a wake-up and run the while loop
         }
        InterruptsOn();
    }
} // end main
```

Sleep + Scheduler

```
int main() {

    setSleep();
    initQueue(&queue);

    while (1) {

        // queue loop
        while (!isEmpty(&queue)) {
            // process the events in queue
            uint8_t event = pop(&queue);
            handleEvent(event);
        }

        sleep();
    }
}
```