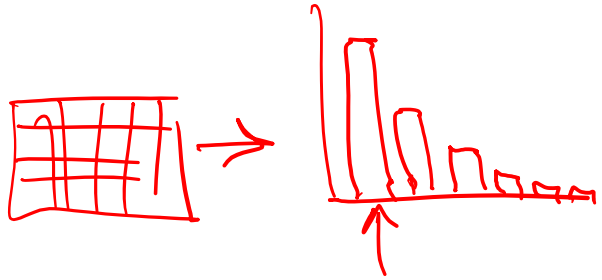
Power and Energy Optimization

Overview



- Process
  1. Start with a power or energy model
  2. Optimize the largest part
  3. Update model based on testing
  4. GOTO 2

The squeaky wheel gets the grease.

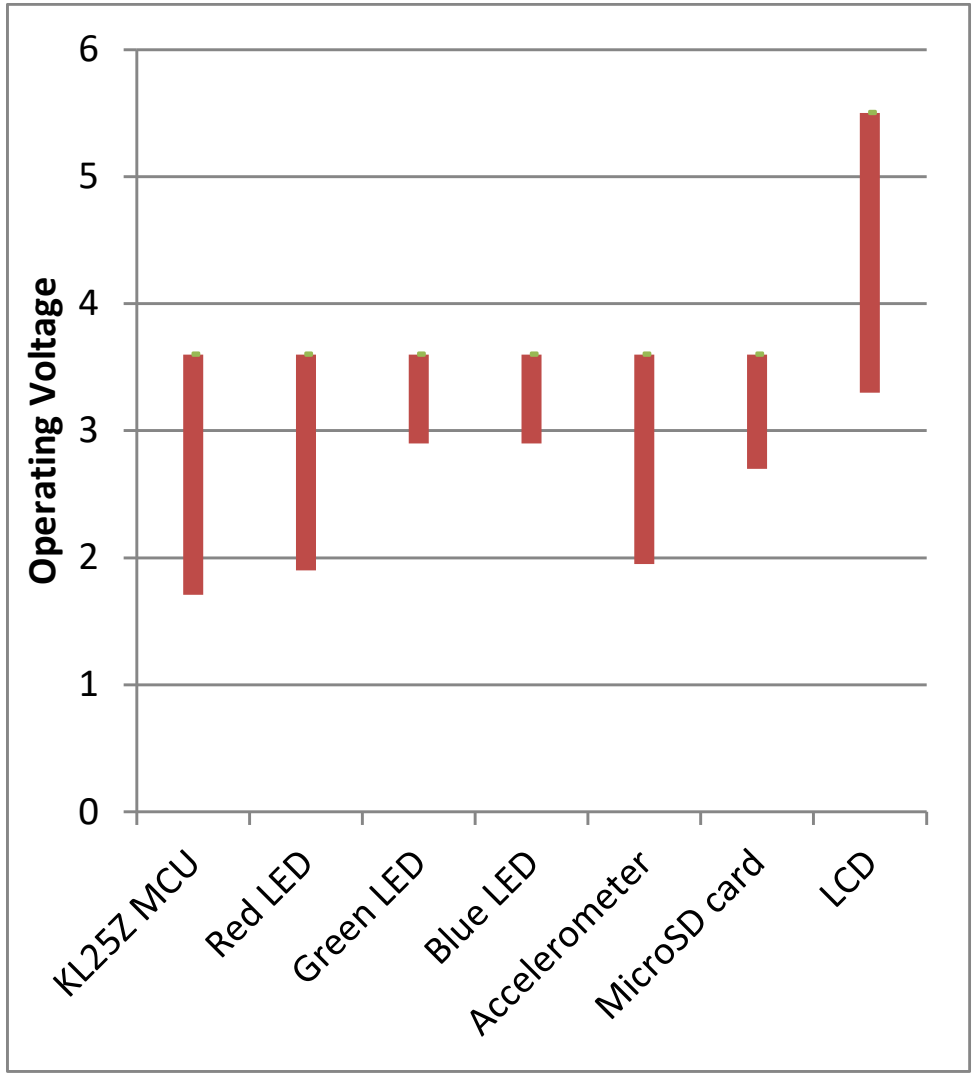
*The **squeakiest** wheel gets the grease **first.***

- What can **you** do to minimize power or energy consumption?
  - Circuit design
    - Choose power-efficient parts
      - Operate at a low voltage
      - Run at **low** frequency if **dynamic** power dominates
      - Turn off processor and other circuits if static power dominates
    - Use low-power modes or shut off parts
  - Program implementation
    - Minimize compute cycles needed
  - Power supply
    - Use an efficient power supply
    - Skip the power supply? Use devices which have a wide operating range
  - Leverage low-power modes of processor and peripherals
    - Group processing together to minimize overhead of switching between active and idle modes
    - Use timers and external events to wake up
    - Use external hardware to reduce CPU on-time

Voltage Scaling

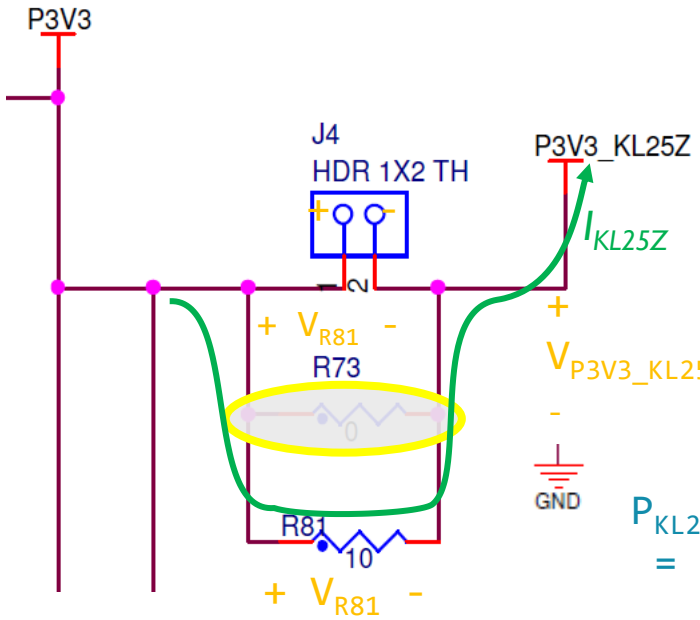
How about Lowering the Operating Voltage?

- P α V^2 for generic digital logic so this can have a big impact
  - Note: we need lower the voltage efficiently, so may want to use switching converter instead of linear regulator
- How low can we go? What stops working as we lower the supply voltage? Examine operating voltage ranges
  - LCD will stop first
  - LEDs will become dim (first blue & green, then red) unless we reduce the series resistor
  - Eventually LEDs will not light at all
- KL25Z MCU
  - 48 MHz at 1.71 V is possible
  - Impact can be significant
  - Reduce power and energy by a factor of ???



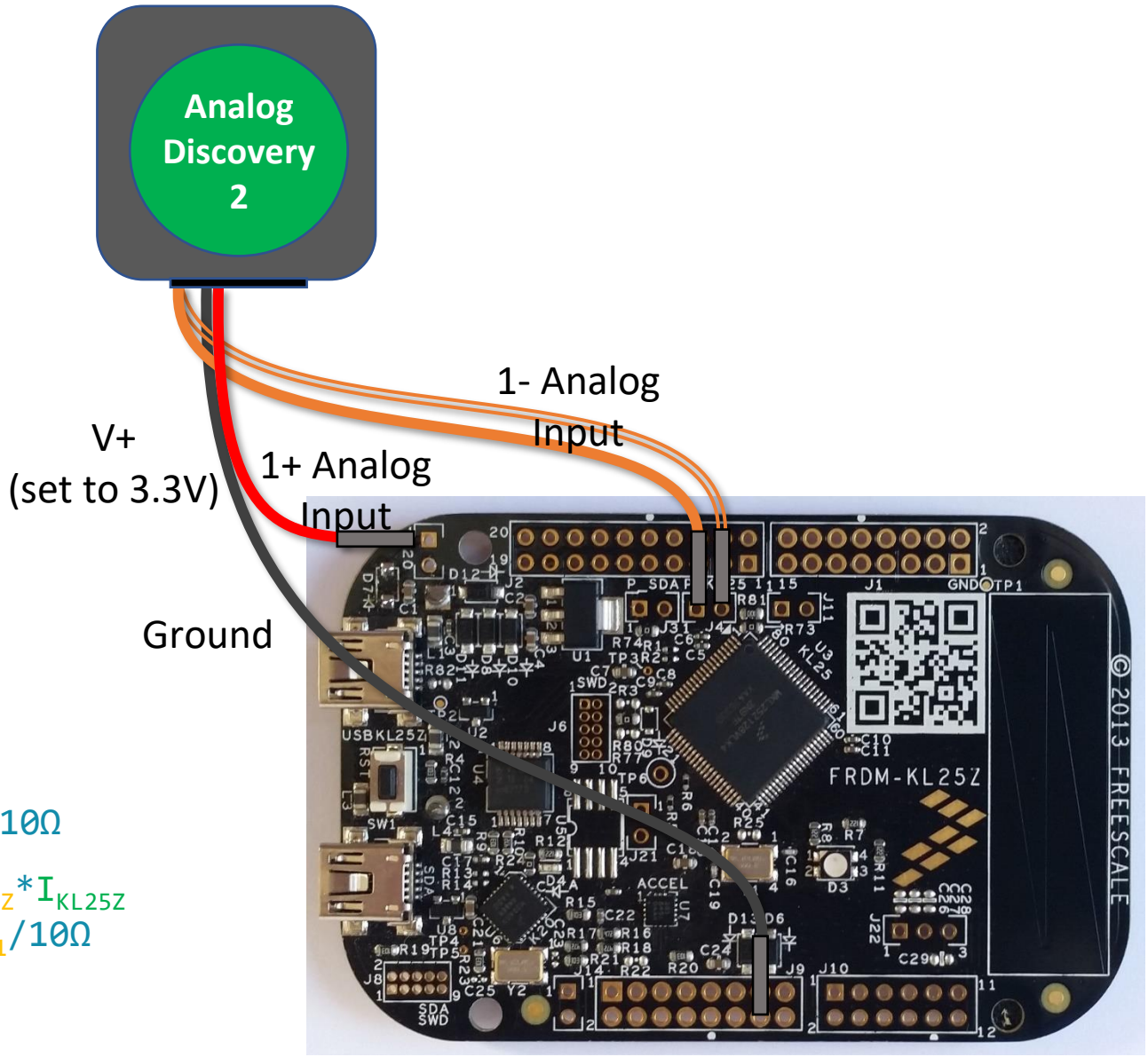
Does the MCU Power Vary Quadratically with Voltage?

- Use AD2 to power P3V3 supply rail at an adjustable voltage
- Monitor MCU current via voltage across R81 (J4) (after removing R73)
- Adjust voltage from 1.8 to 3.6 V
- Power = current * voltage



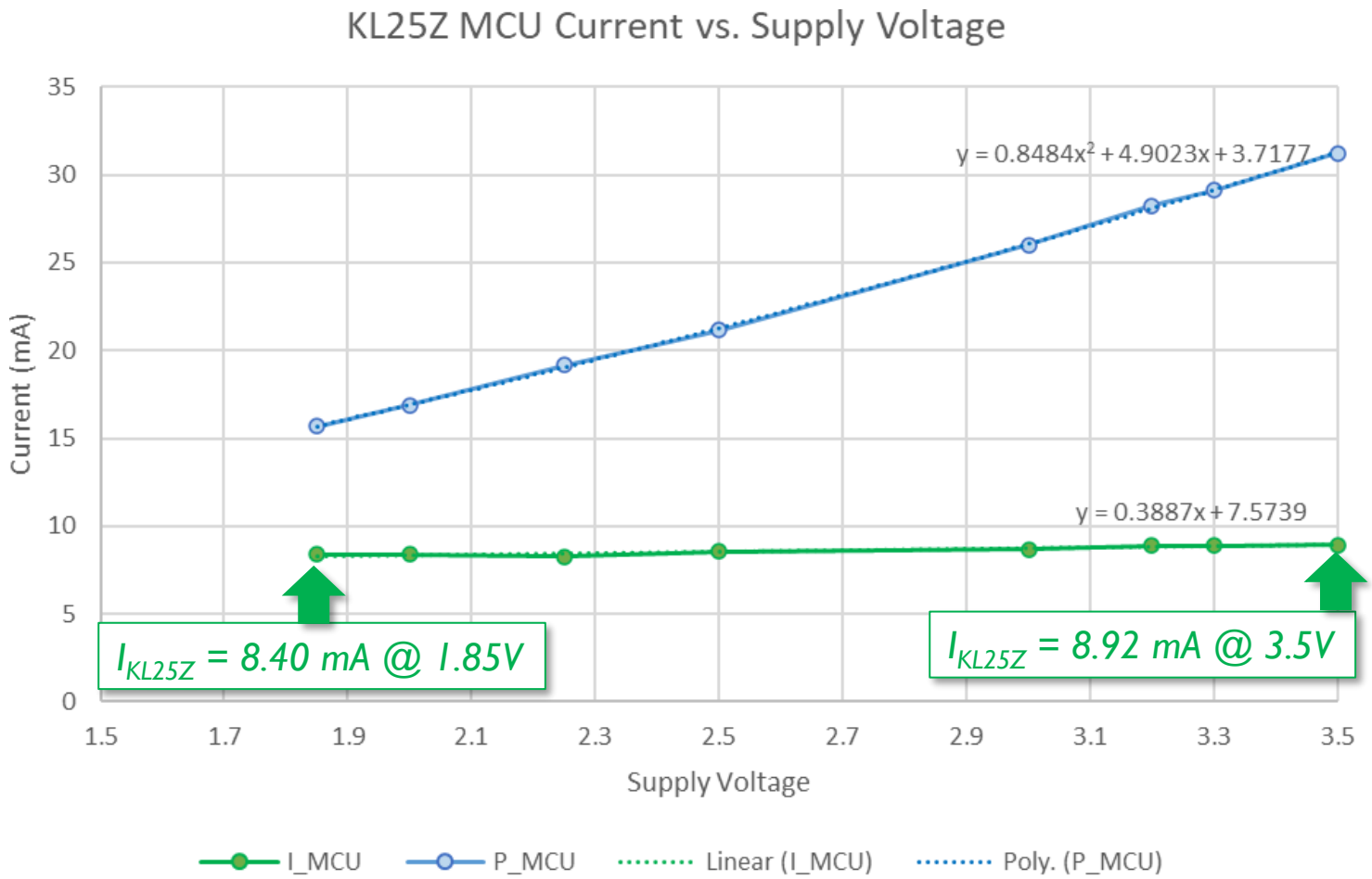
$I_{KL25Z} = V_{R81}/10\Omega$

$P_{KL25Z} = V_{P3V3\_KL25Z} * I_{KL25Z}$
$= V_{P3V3\_KL25Z} * V_{R81}/10\Omega$



Does the MCU Power Vary Quadratically with Voltage?

- Current is almost constant, depends very little on voltage
  - I = 7.57 mA + 0.3887 mA/V
  - MCU has internal voltage regulator!
  - Most MCU circuitry runs at fixed voltage, so current doesn't vary with voltage
- Impact on power
  - Power rises almost linearly, since current is almost constant



Minimum Data Retention Voltage

Table 1. Voltage and current operating requirements (continued)

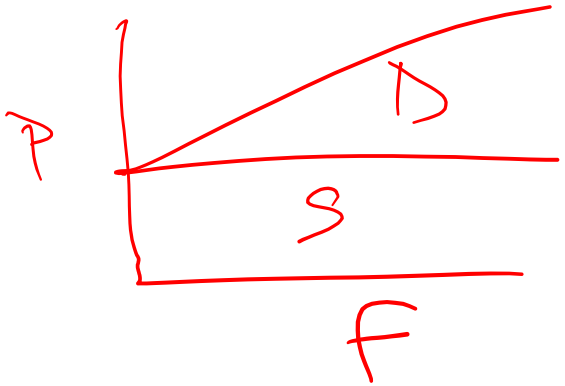
Symbol	Description	Min.	Max.	Unit	Notes
V_{HYS}	Input hysteresis	0.06 × V_{DD}	—	V	
I_{ICDIO}	Digital pin negative DC injection current — single pin • $V_{IN} < V_{SS}$-0.3V	-5	—	mA	1
I_{ICAIO}	Analog[2] pin DC injection current — single pin • $V_{IN} < V_{SS}$-0.3V (Negative current injection) • $V_{IN} > V_{DD}$+0.3V (Positive current injection)	-5 —	— +5	mA	3
I_{ICcont}	Contiguous pin DC injection current —regional limit, includes sum of negative injection currents or sum of positive injection currents of 16 contiguous pins • Negative current injection • Positive current injection	-25 —	— +25	mA	
V_{RAM}	V_{DD} voltage required to retain RAM	1.2	—	V	

- Datasheet says minimum RAM retention voltage = 1.2V
  - *Extra Credit Project - Verify it works!*
- What about processor registers? 1.2V also, or something higher?
- How low can we take the MCU supply voltage and have all RAM and registers retain their values?

Voltage and Frequency Scaling

Voltage and Frequency Scaling

- Ideas
  - Reduce frequency to ideal frequency for each task
  - Reduce voltage to minimum needed for that clock frequency
  - $P = S_P V_{CC}^2 + C_P V_{CC}^2 f_{Clock}$

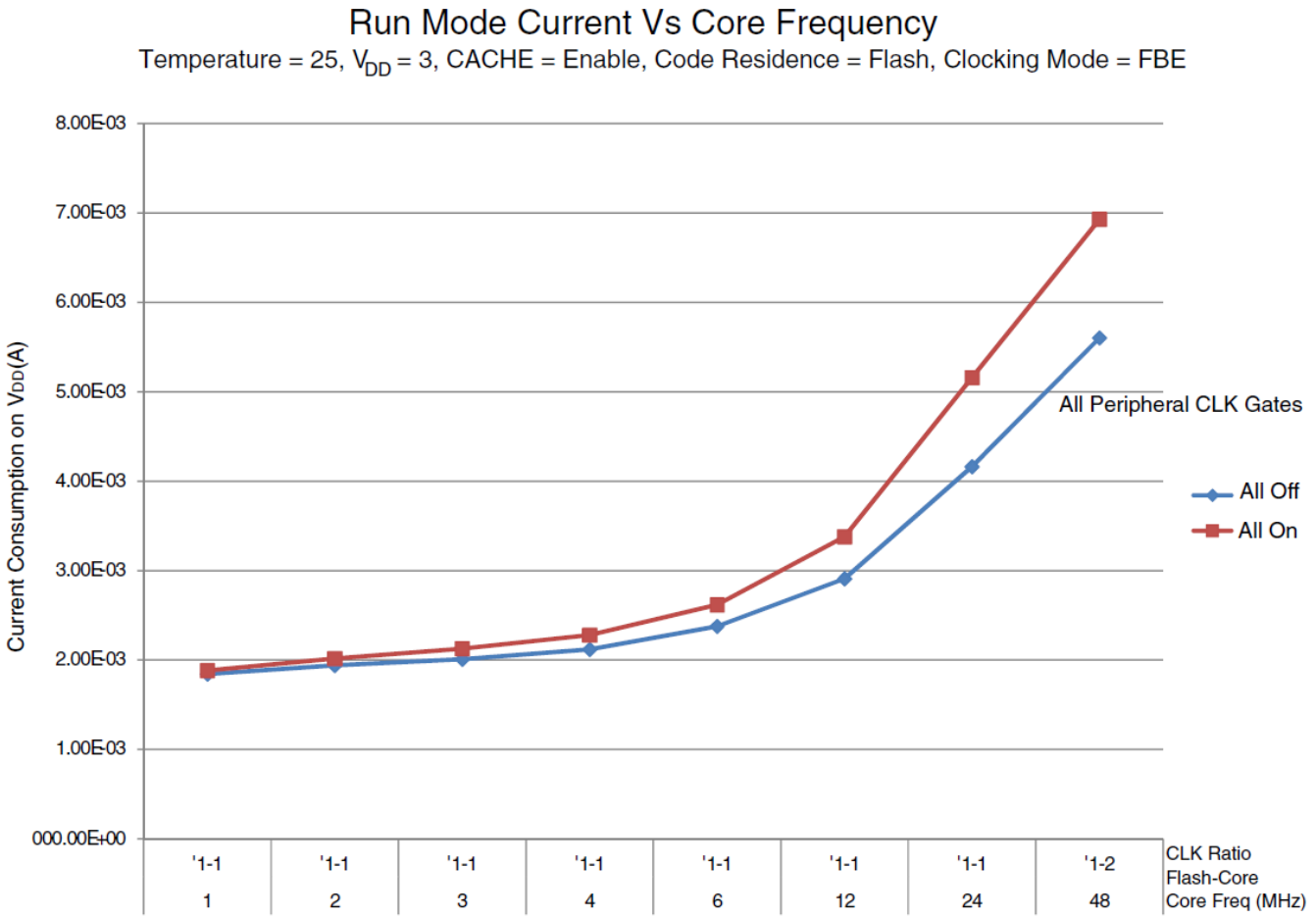


- Circuitry
  - Clock divider/synthesizer
  - Variable voltage supply
    - Efficiency: may need a DC-to-DC converter (switch-mode power supply) to efficiently convert supply voltage.
    - Voltage transition rate: need to be able to quickly change output voltage
    - Transition energy dissipation: want to waste little energy when changing the voltage

PLL CLKDIV
FLL

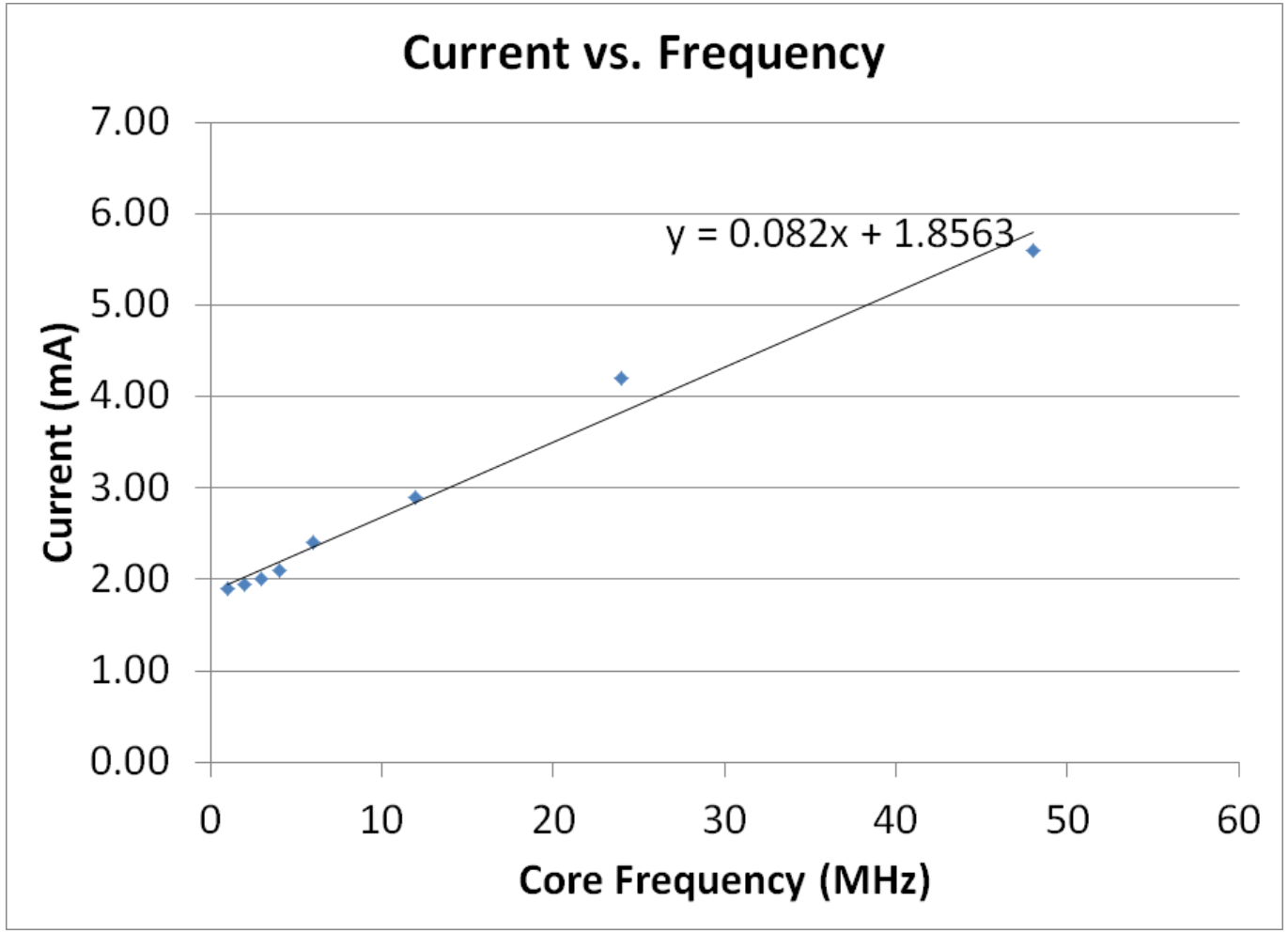
Evaluating Frequency Scaling for the KL25 with Run and Very Low Power Run Modes

What is the Impact of Clock Speed on Current?



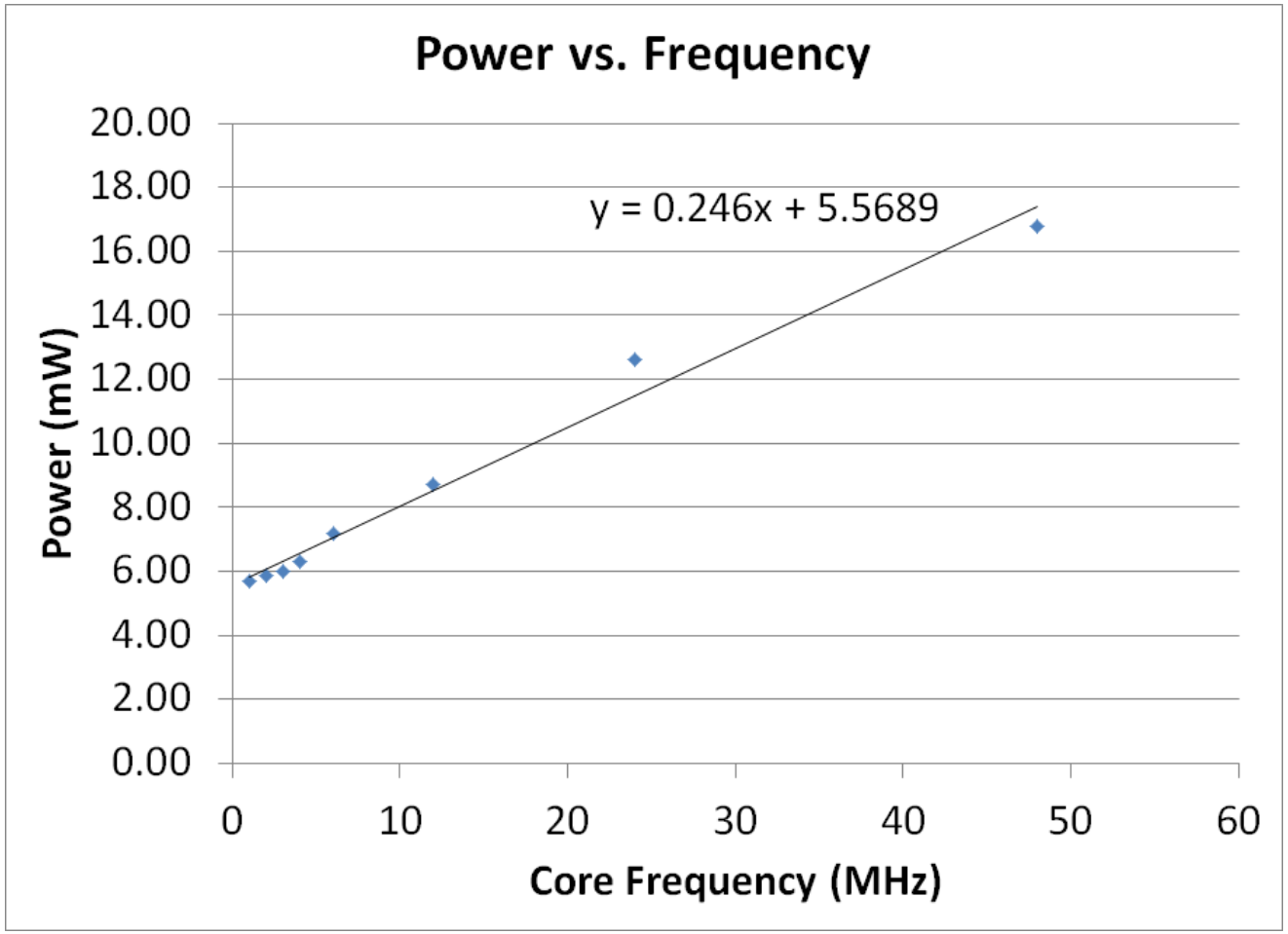
- Current rises with core clock frequency
  - What relationship?
- Offset due to static current consumption
- Impact of enabling peripheral clocks

Current with Linear Axes



- Assumes all peripherals are turned off
- Current linearly rises with core clock frequency

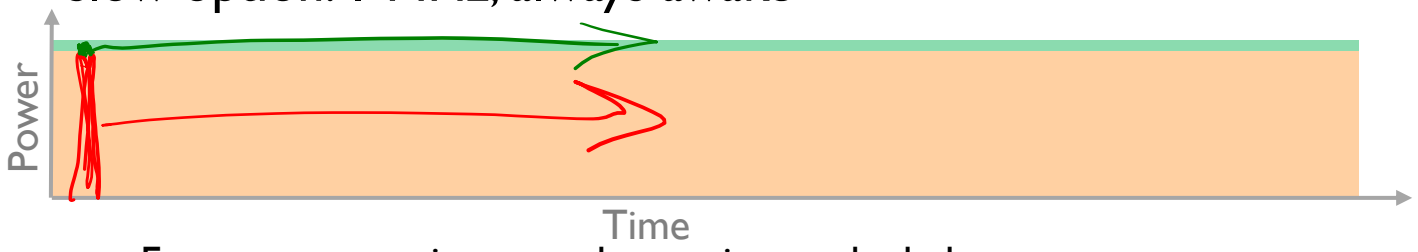
What is the Impact of Clock Speed on Power?



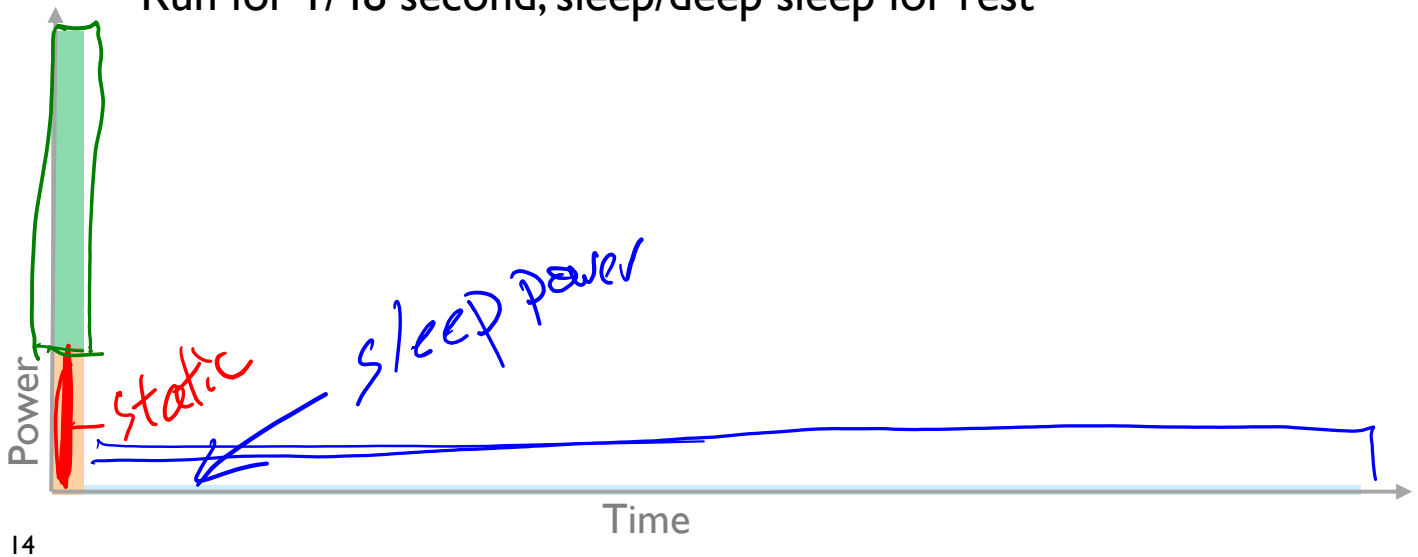
- Power rises linearly with core clock frequency
- Power is proportional to voltage*current, and voltage is fixed here (3.0 V)
- 1 MHz is lowest power point
  - Compare: https://ambiq.com/mcu-soc/

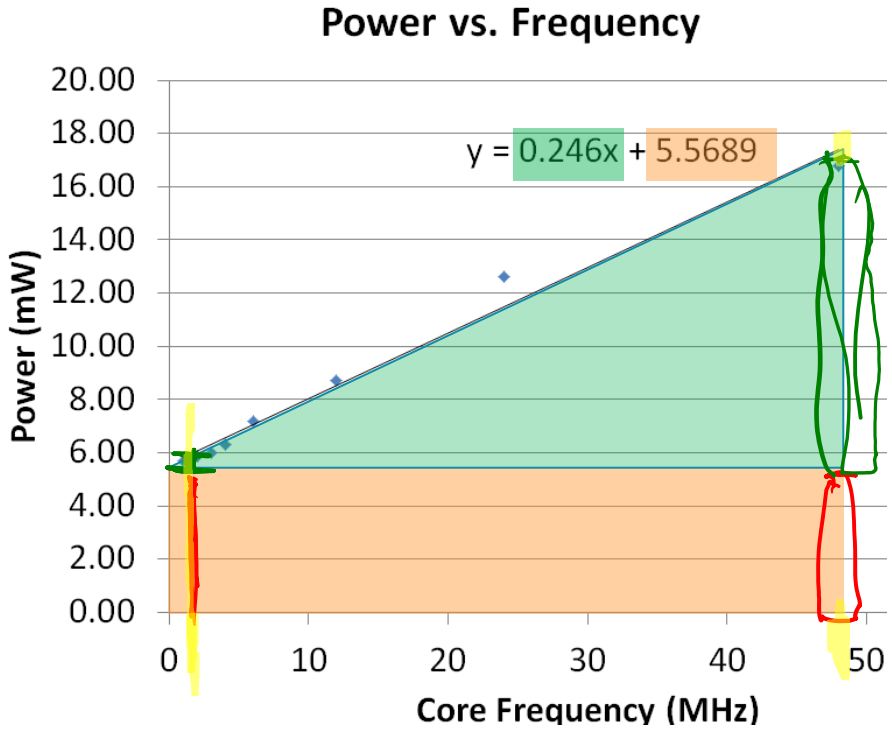
Comparing Power and Energy

- MCU Power = 0.246 mW/MHz + 5.5689 mW
- Example: Need 1,000,000 cycles of processing per second
- Slow option: 1 MHz, always awake

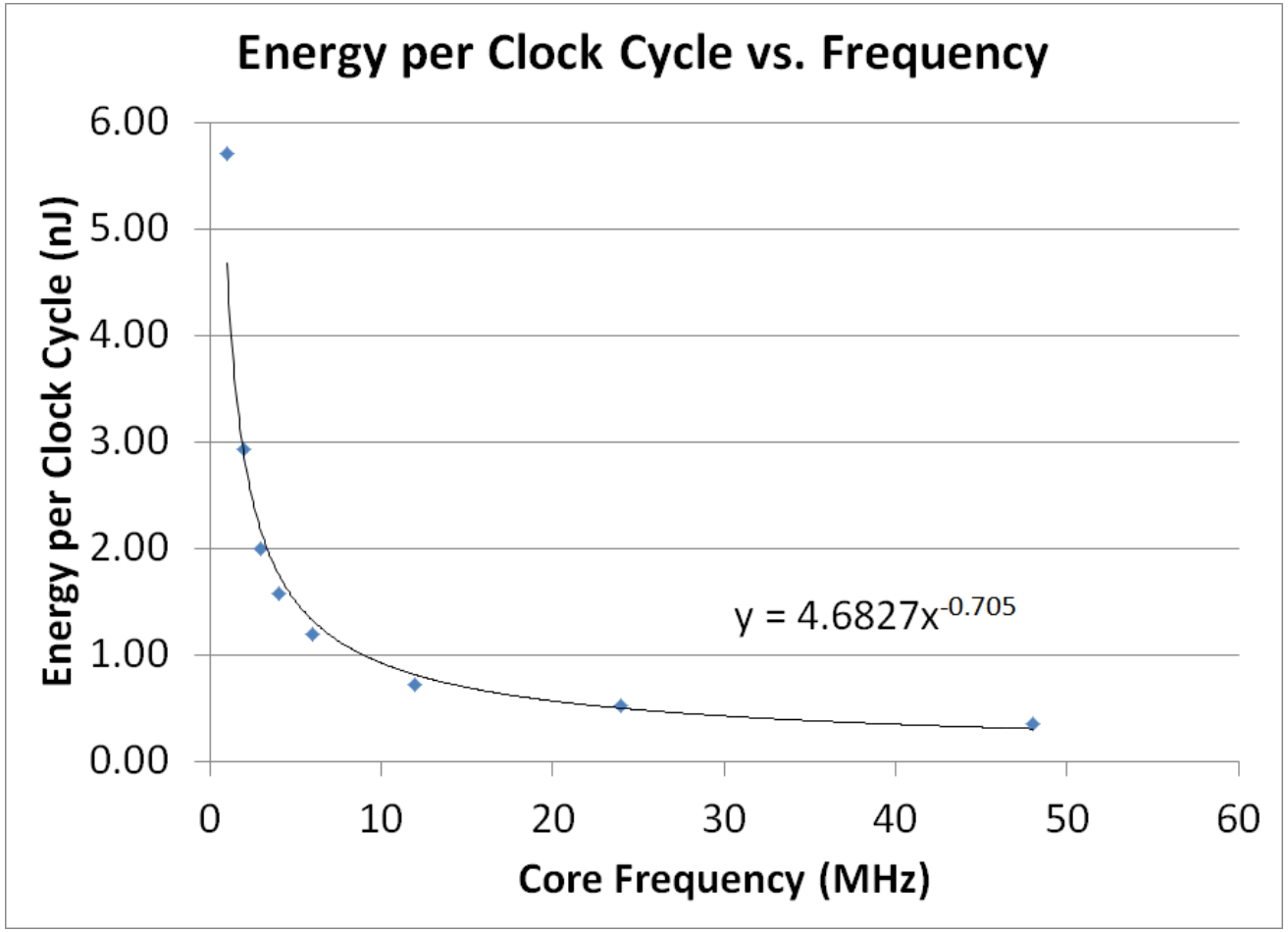


  - Energy = power integrated over time = shaded area
- Fast option: 48 MHz. Can sleep when not needed.
  - Run for 1/48 second, sleep/deep sleep for rest



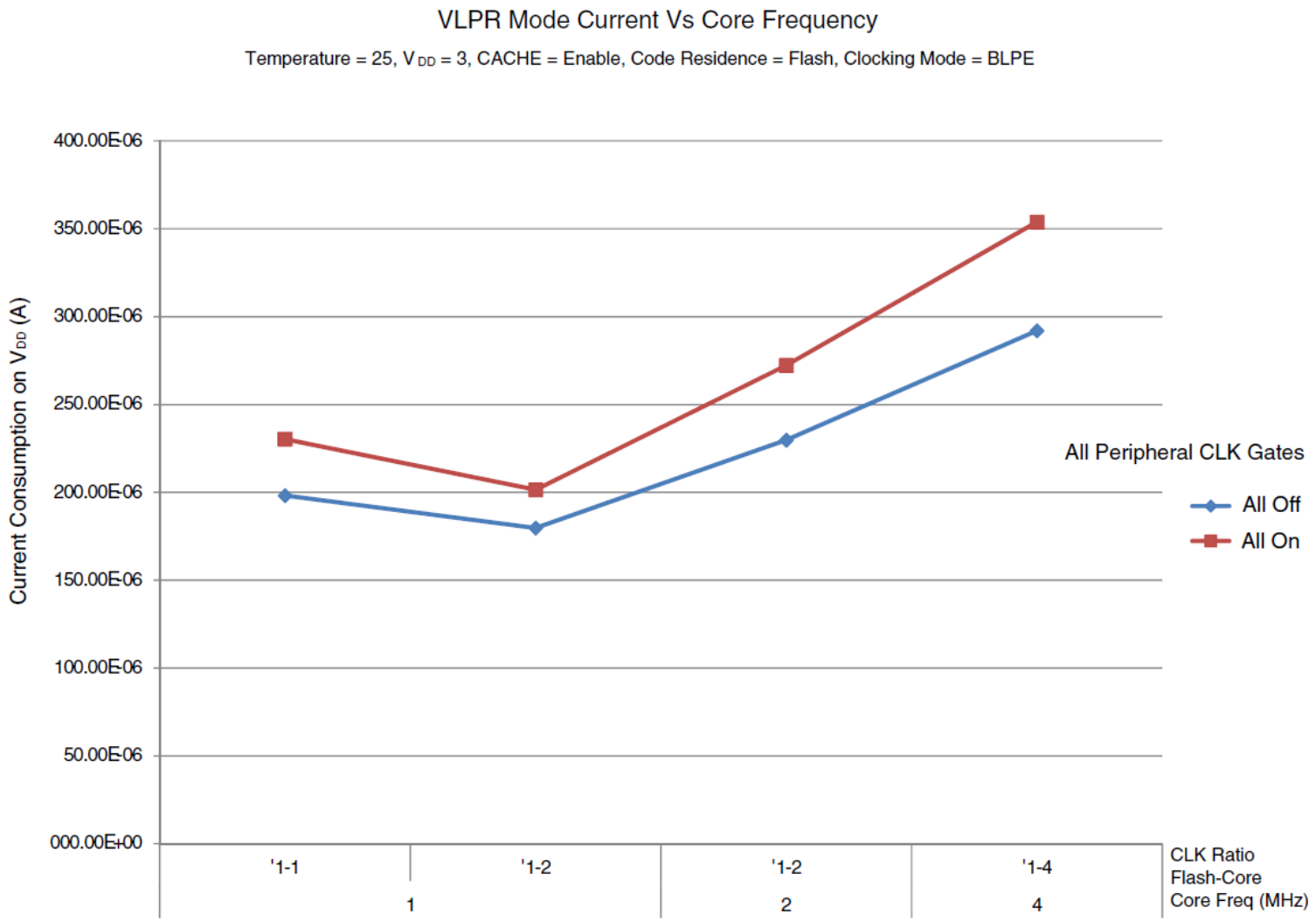


What is the Impact of Clock Speed on Energy?



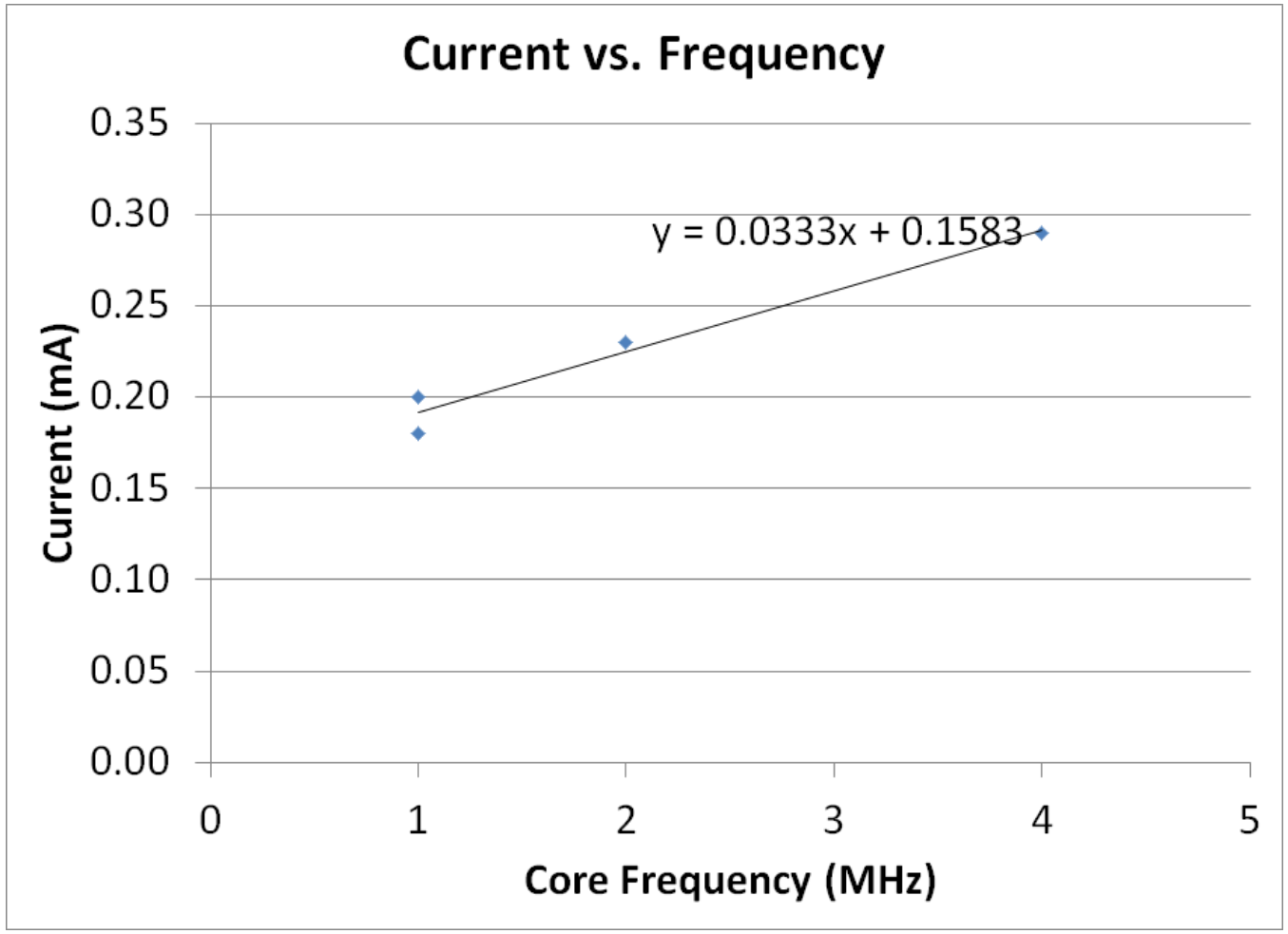
- Energy per clock cycle is power divided by clock frequency
- 48 MHz is lowest-energy point

Current vs. Clock Speed – VLPR Mode



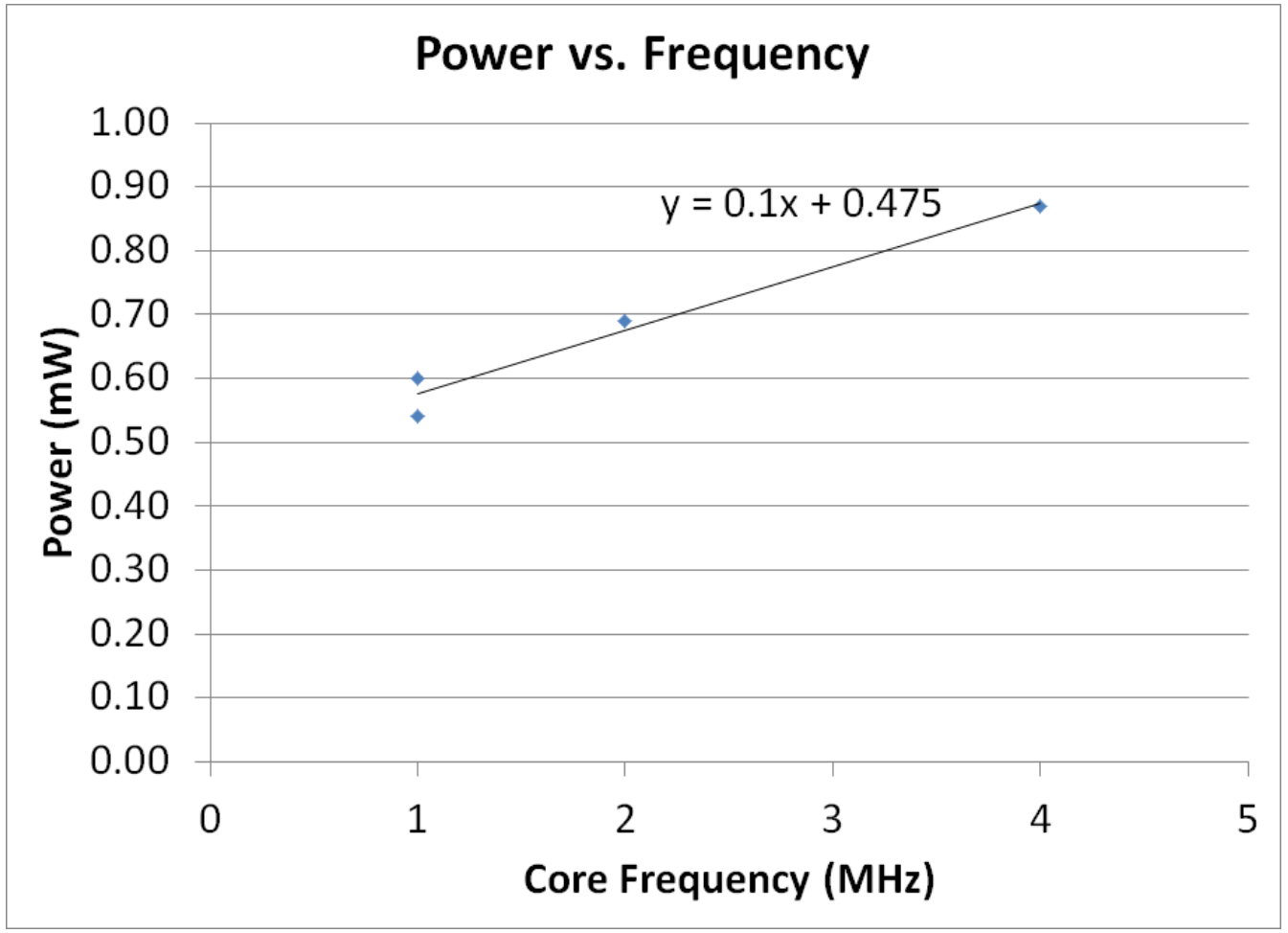
- Now consider VLPR – very low power run mode
  - Maximum core frequency is 4 MHz
- Still have static current offset
- Can see impact of changing core speed, bus speed, and providing clock to peripherals

VLPR Current with Linear Axes



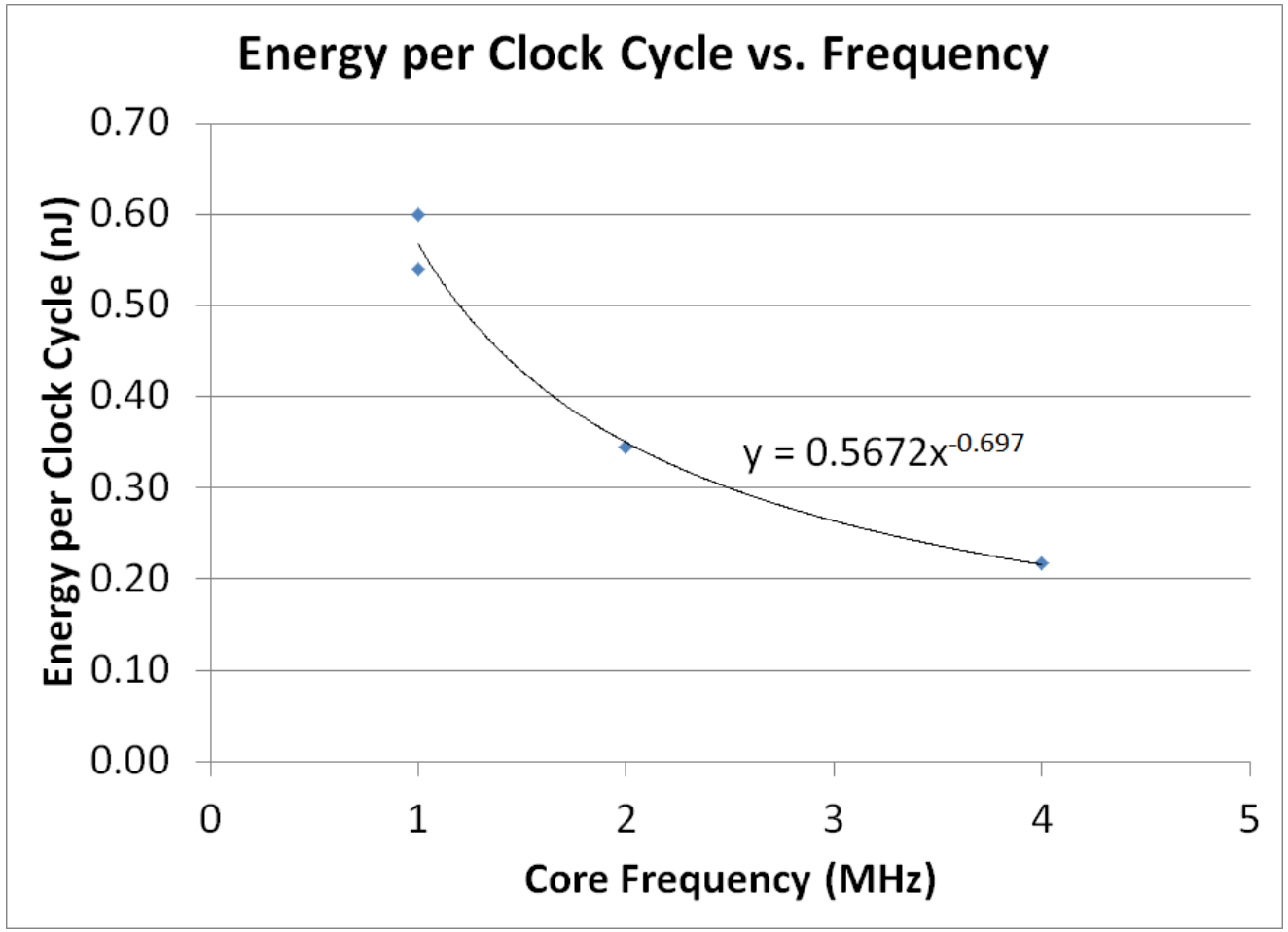
- Current linearly rises with core clock frequency

VLPR: What is the Impact of Clock Speed on Power?



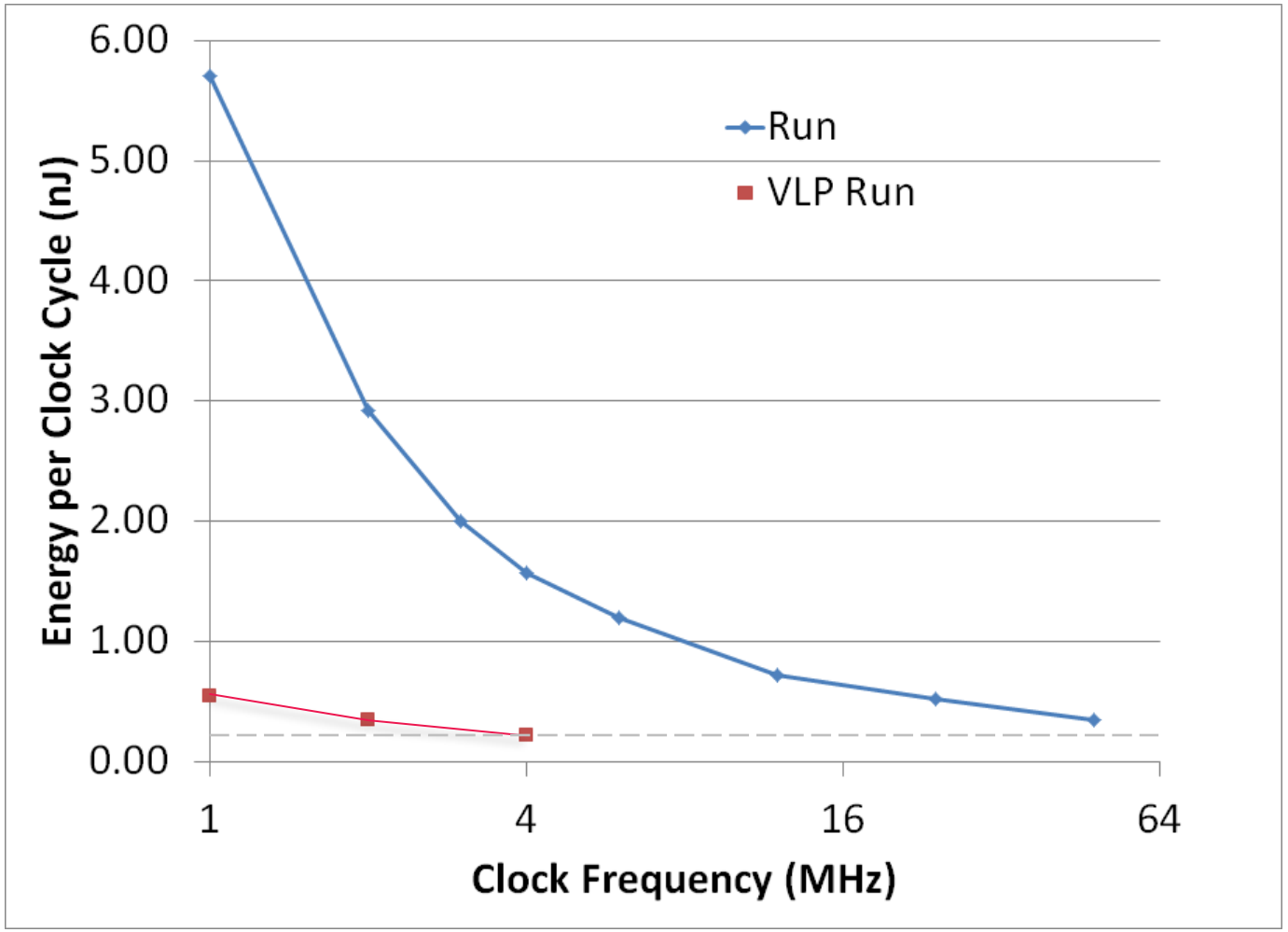
- Power rises linearly with core clock frequency
- Power is proportional to voltage*current, and voltage is fixed here (3.0 V)
- 1 MHz is lowest power point

VLPR: What is the Impact of Clock Speed on Energy?



- Energy per clock cycle is power divided by clock frequency
- 4 MHz is lowest-energy point

Comparing Energy for Run and VLP Run

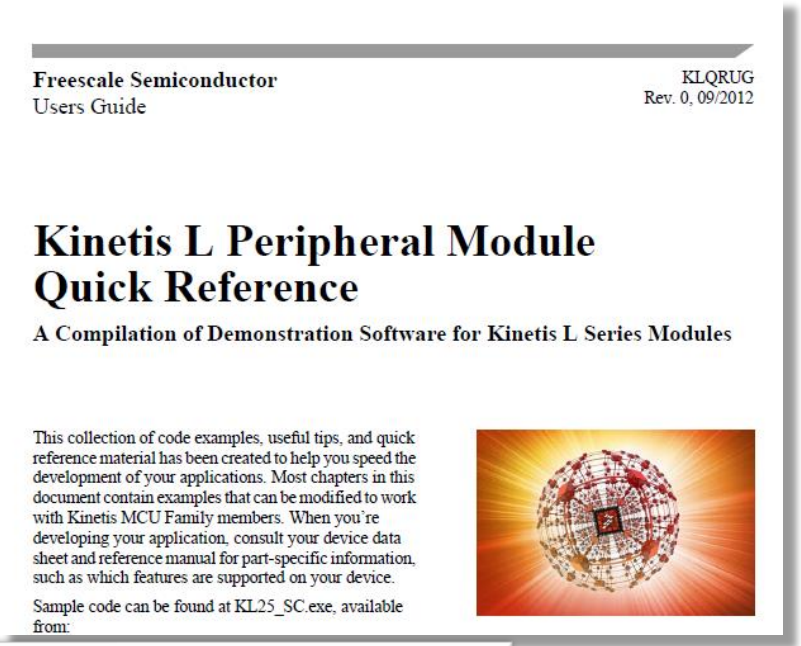


- Very low power also gives very low energy!
- 4 MHZ VLP Run mode more energy-efficient than 48 MHz Run mode

Power Management Strategies

Application Notes

- NXP/Freescale
  - AN4503
  - KLQRUG
- Microchip
  - AN1416



Freescale Semiconductor
Users Guide

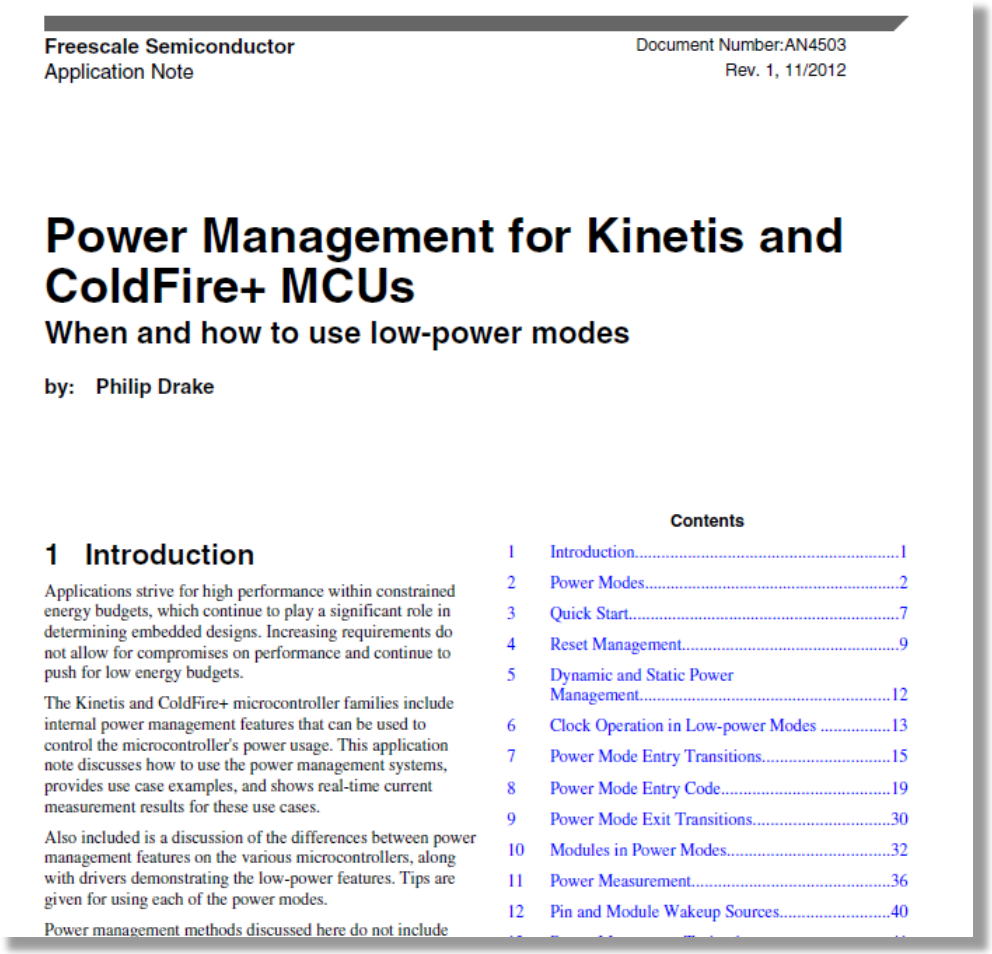
KLQRUG
Rev. 0, 09/2012

Kinetis L Peripheral Module Quick Reference

A Compilation of Demonstration Software for Kinetis L Series Modules

This collection of code examples, useful tips, and quick reference material has been created to help you speed the development of your applications. Most chapters in this document contain examples that can be modified to work with Kinetis MCU Family members. When you're developing your application, consult your device data sheet and reference manual for part-specific information, such as which features are supported on your device.

Sample code can be found at KL25_SC.exe, available from:



Freescale Semiconductor
Application Note

Document Number:AN4503
Rev. 1, 11/2012

Power Management for Kinetis and ColdFire+ MCUs

When and how to use low-power modes

by: Philip Drake

1 Introduction

Applications strive for high performance within constrained energy budgets, which continue to play a significant role in determining embedded designs. Increasing requirements do not allow for compromises on performance and continue to push for low energy budgets.

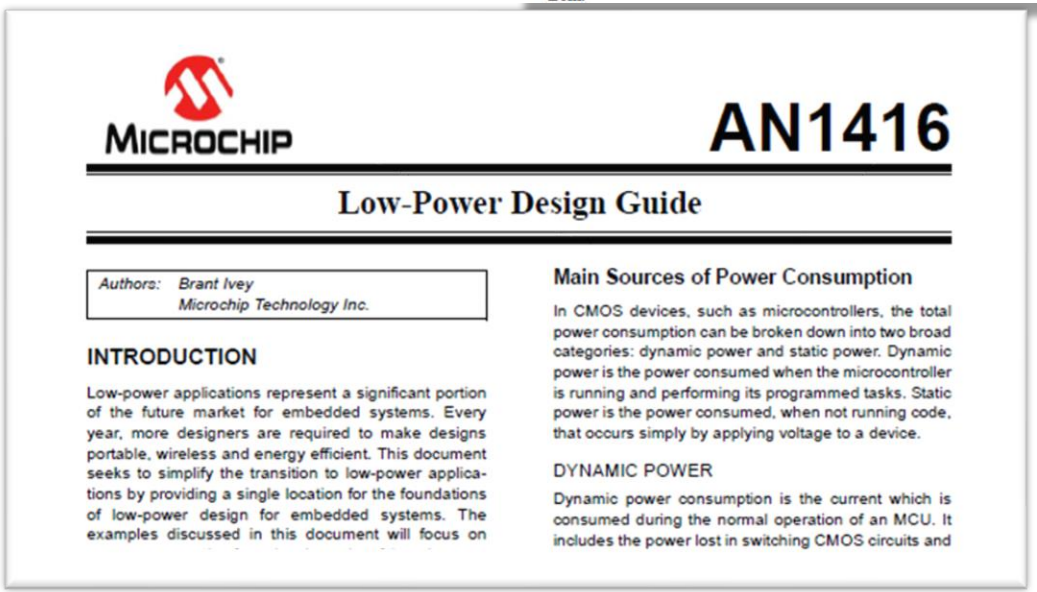
The Kinetis and ColdFire+ microcontroller families include internal power management features that can be used to control the microcontroller's power usage. This application note discusses how to use the power management systems, provides use case examples, and shows real-time current measurement results for these use cases.

Also included is a discussion of the differences between power management features on the various microcontrollers, along with drivers demonstrating the low-power features. Tips are given for using each of the power modes.

Power management methods discussed here do not include

Contents

1 Introduction....1
2 Power Modes....2
3 Quick Start....7
4 Reset Management....9
5 Dynamic and Static Power Management....12
6 Clock Operation in Low-power Modes....13
7 Power Mode Entry Transitions....15
8 Power Mode Entry Code....19
9 Power Mode Exit Transitions....30
10 Modules in Power Modes....32
11 Power Measurement....36
12 Pin and Module Wakeup Sources....40



MICROCHIP

AN1416

Low-Power Design Guide

Authors: Brant Ivey
Microchip Technology Inc.

INTRODUCTION

Low-power applications represent a significant portion of the future market for embedded systems. Every year, more designers are required to make designs portable, wireless and energy efficient. This document seeks to simplify the transition to low-power applications by providing a single location for the foundations of low-power design for embedded systems. The examples discussed in this document will focus on

Main Sources of Power Consumption

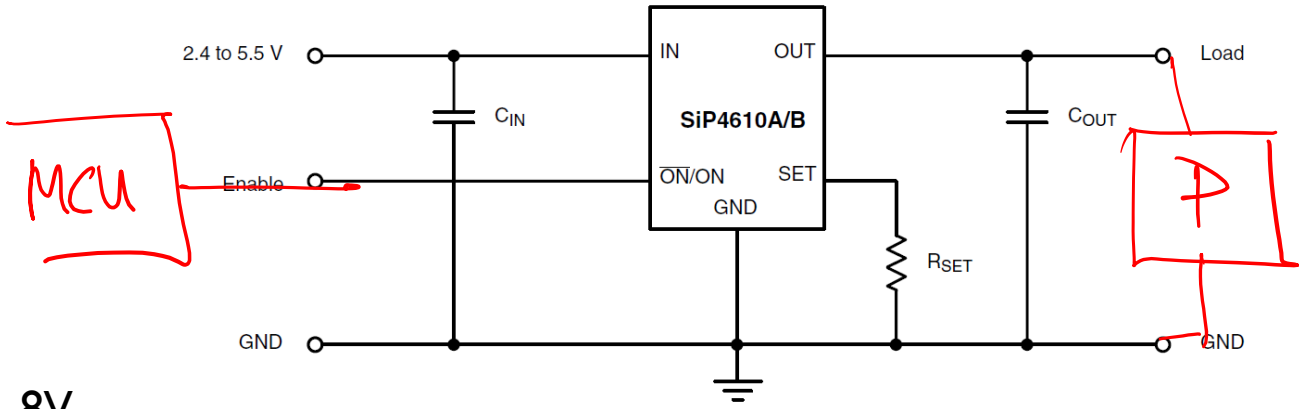
In CMOS devices, such as microcontrollers, the total power consumption can be broken down into two broad categories: dynamic power and static power. Dynamic power is the power consumed when the microcontroller is running and performing its programmed tasks. Static power is the power consumed, when not running code, that occurs simply by applying voltage to a device.

DYNAMIC POWER

Dynamic power consumption is the current which is consumed during the normal operation of an MCU. It includes the power lost in switching CMOS circuits and

Power Management for Peripherals

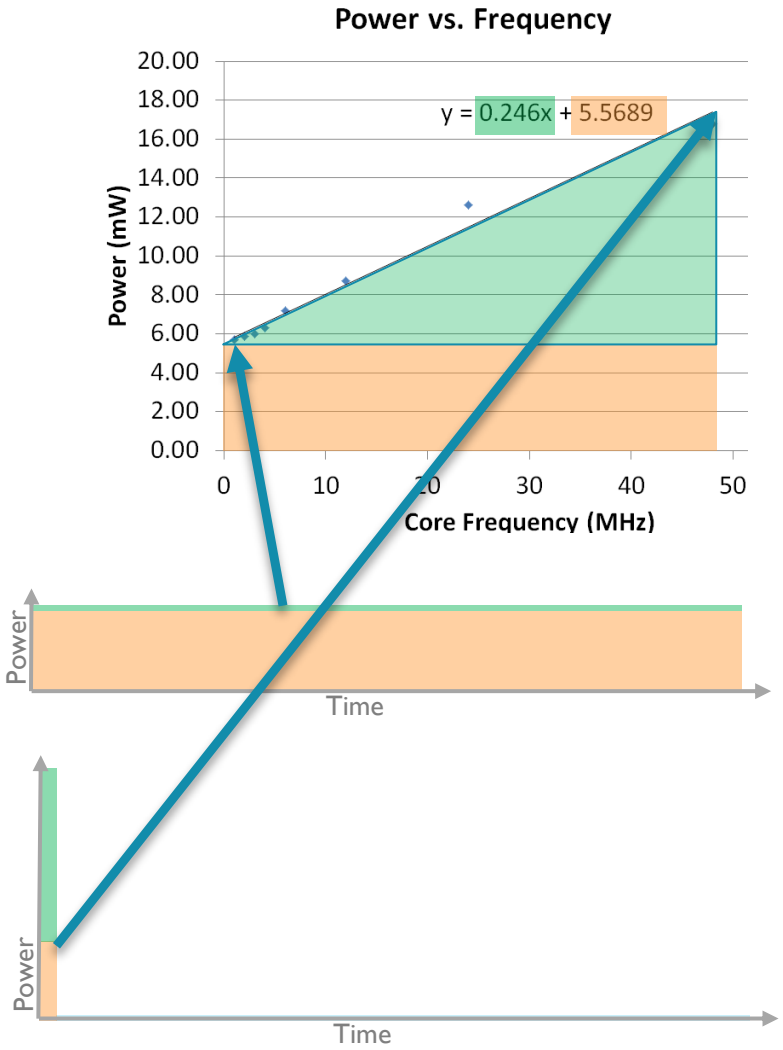
- Pick low-power peripherals
- Switch them off when not needed
  - Some have standby or power-down modes controlled with a pin
    - Example: Nat. Semiconductor LMV982 Dual 1.8V Op-Amps with Shutdown
      - V_{CC} = 1.8 to 5.0 V
      - Supply current 200 uA
      - Shutdown current <5 uA
      - 19 us turn-on time from shutdown
  - Some accept standby or sleep command
    - Accelerometer MMA8451Q via I2C
    - LCD Controller ST7789S via parallel bus



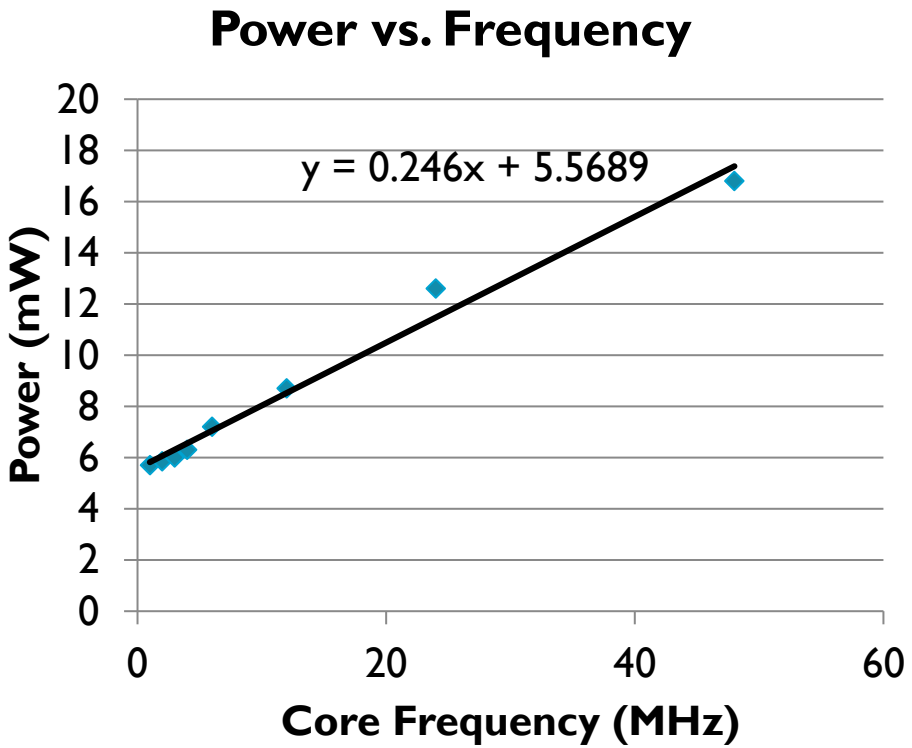
- Otherwise have to switch off power
  - Low-current devices (<10 mA)
    - Supply power with MCU's digital output pin(s)
  - Higher-current devices
    - Switch power with a transistor or high-side supply switch driven by an MCU digital output
    - Example: Vishay SiP4610 Protected 1 Amp High-Side Load switch
      - V_{CC} = 2.4 to 5.5 V
      - 1 amp output
      - Quiescent current 9 uA
      - Shutdown current < 1 uA

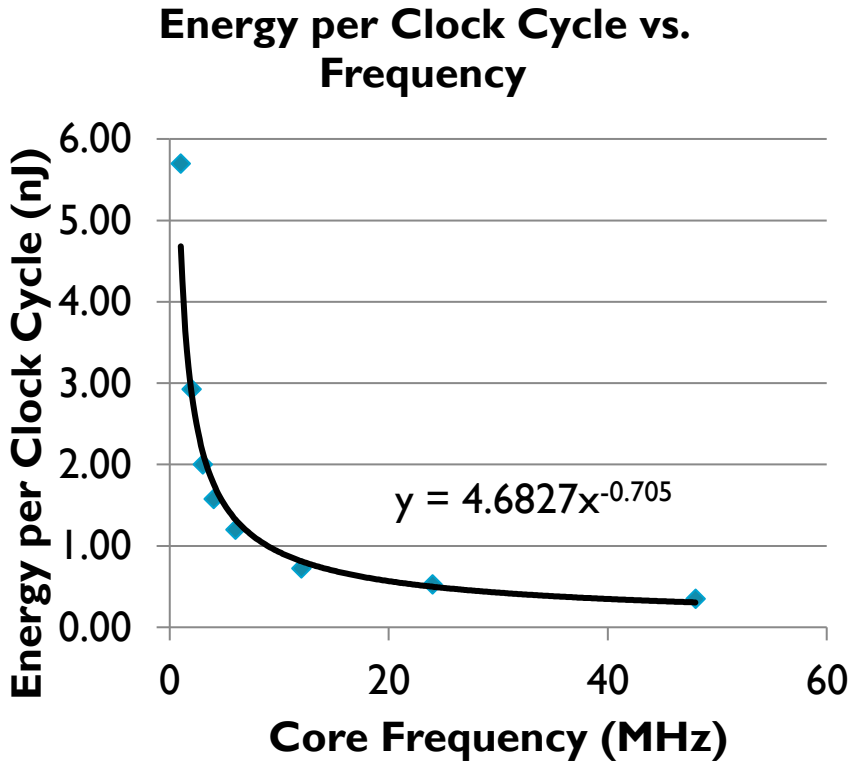
Power Management Options for MCUs

- Clock scaling
  - Slow down the CPU clock if there's little to do
- Sleep
  - Turn off the processor if there's nothing to do
    - Many MCUs have sleep modes which halt instruction execution and shut off unused portions of the MCU
    - Can have significant power savings: factor of 1000 or more
  - Disadvantage: waking up
    - Need a mechanism such as timer or interrupt
      - Events you care about must be connected to trigger wake-up
    - Depth of sleep mode may lead to long wake-up time, reducing system performance and reducing viable "nap windows"
- Both clock scaling and sleep
  - Many variables!
  - Not immediately obvious which settings are best



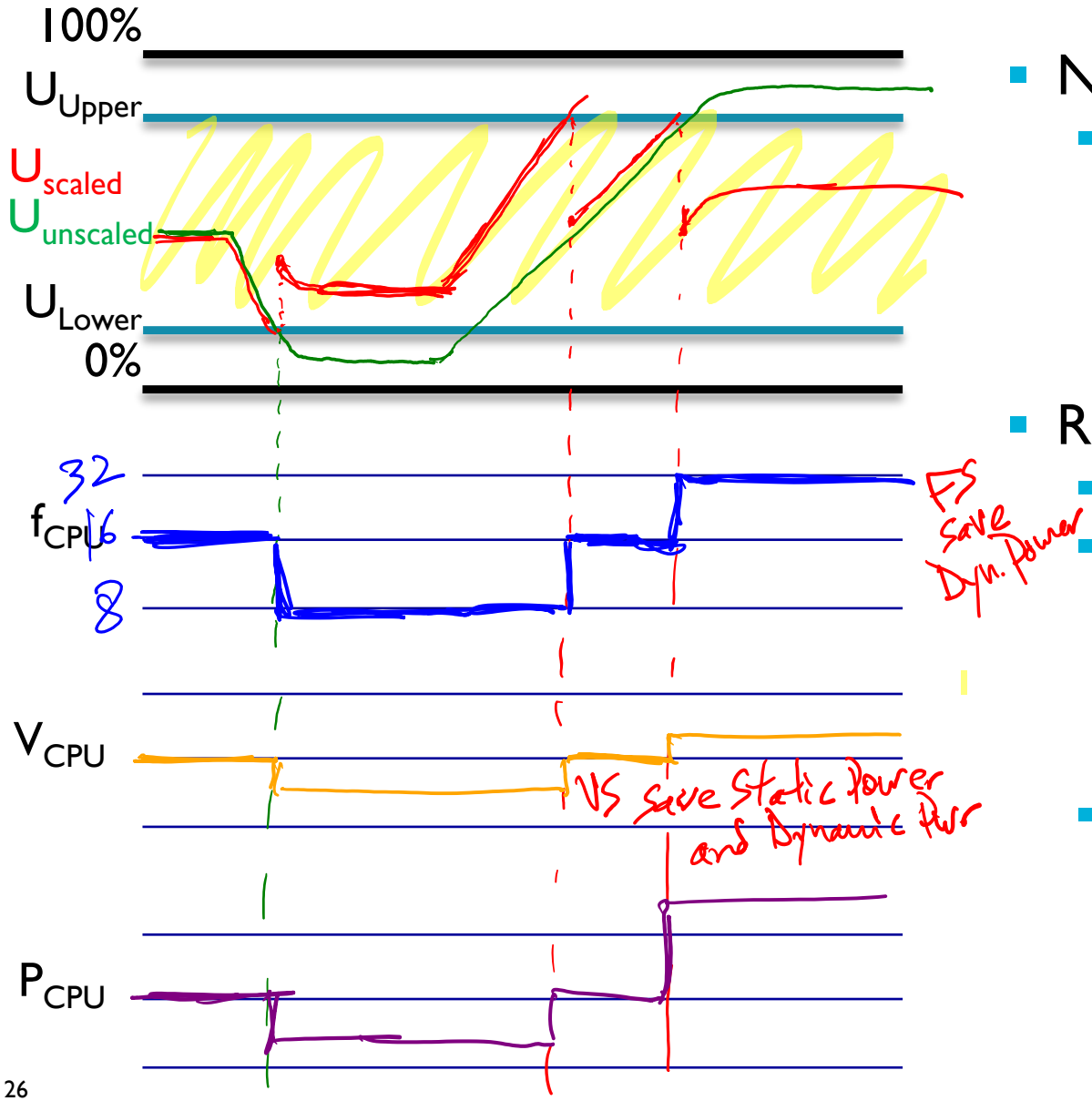
Clock Scaling – Slow Down





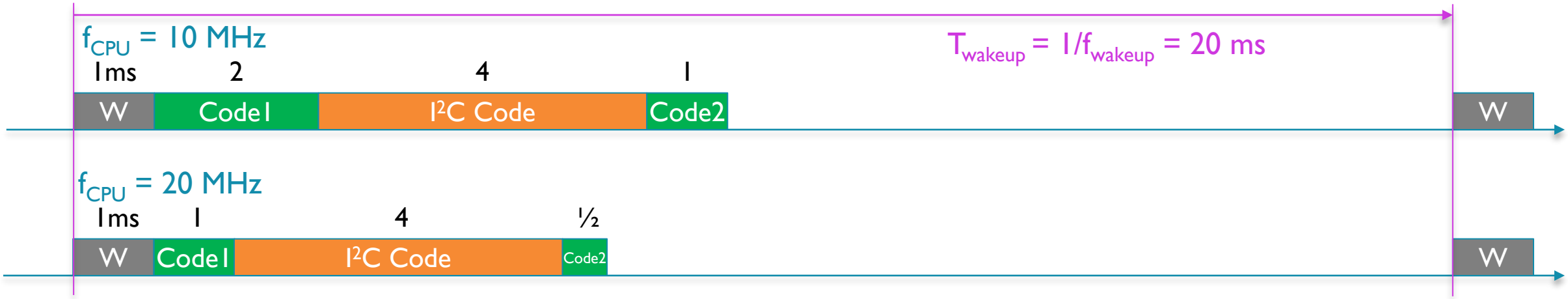
- Slowing clock cuts dynamic power used
  - Eventually reaches limit of about 6 mW - static (leakage) power
- Fewer cycles spent waiting for the next interrupt or timer event
- Actual energy cost per cycle increases

Slow Down … But To Which Frequency?



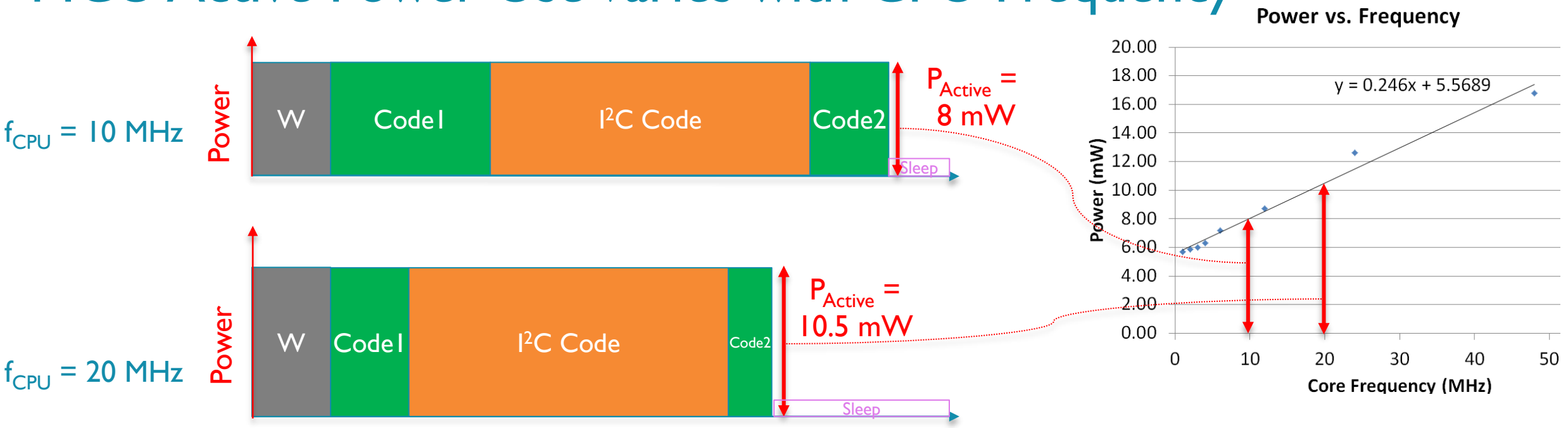
- Non-real-time systems
  - Use a ***governor*** *(feedback system)* to set the speed, trying to keep processor utilization between upper and lower bounds
    - If $U>U_{upper}$, raise CPU speed and voltage
    - If $U< U_{lower}$, lower CPU speed and voltage
- Real-time systems have deadlines
  - Goal: minimize energy AND meet all deadlines
  - Can then set speed for each task *statically*…
    - Pick the frequency (hence voltage) per task which minimizes the energy based on the task's ***worst-case execution time*** and ***deadline***.
    - One fixed clock frequency per task
  - Or can set speed for each task *dynamically*
    - Pick the frequency (hence voltage) per task which minimizes the energy based on the task's ***past behavior, current progress, etc.*** and ***deadline***
    - Clock frequency varies as task behavior changes

MCU Active Time falls with CPU Clock Frequency



- Time for many MCU activities depends directly on CPU clock frequency f_{CPU}
  - Code1 needs 20,000 cycles
  - Code2 needs 10,000 cycles
  - If code i needs C_i clock cycles, computation time C_i/f_{CPU}
- Others aren't proportional to f_{CPU}
  - Wake from sleep: Depends on oscillator (and maybe its wake time)
  - I²C communication: Data rate determines time waiting for next byte
  - Using ADC: Conversion clock frequency determines time waiting for conversion to complete
  - May be able to sleep during some of these?
- Time used per wake-up at f_{CPU}
  - $T_{Active} = C/f_{CPU} + T_{const}$
  - f_{CPU} = 10 MHz: T_{Active} = 1 + 2 + 4 + 1 ms = 8 ms
  - f_{CPU} = 20 MHz: T_{Active} = 1 + 1 + 4 + ½ ms = 6.5 ms
- Active, sleep times per 1 second
  - $T_{Active} = f_{Wakeup}*(C/f_{CPU} + T_{const})$
  - $T_{Sleep} = 1 - T_{active}$

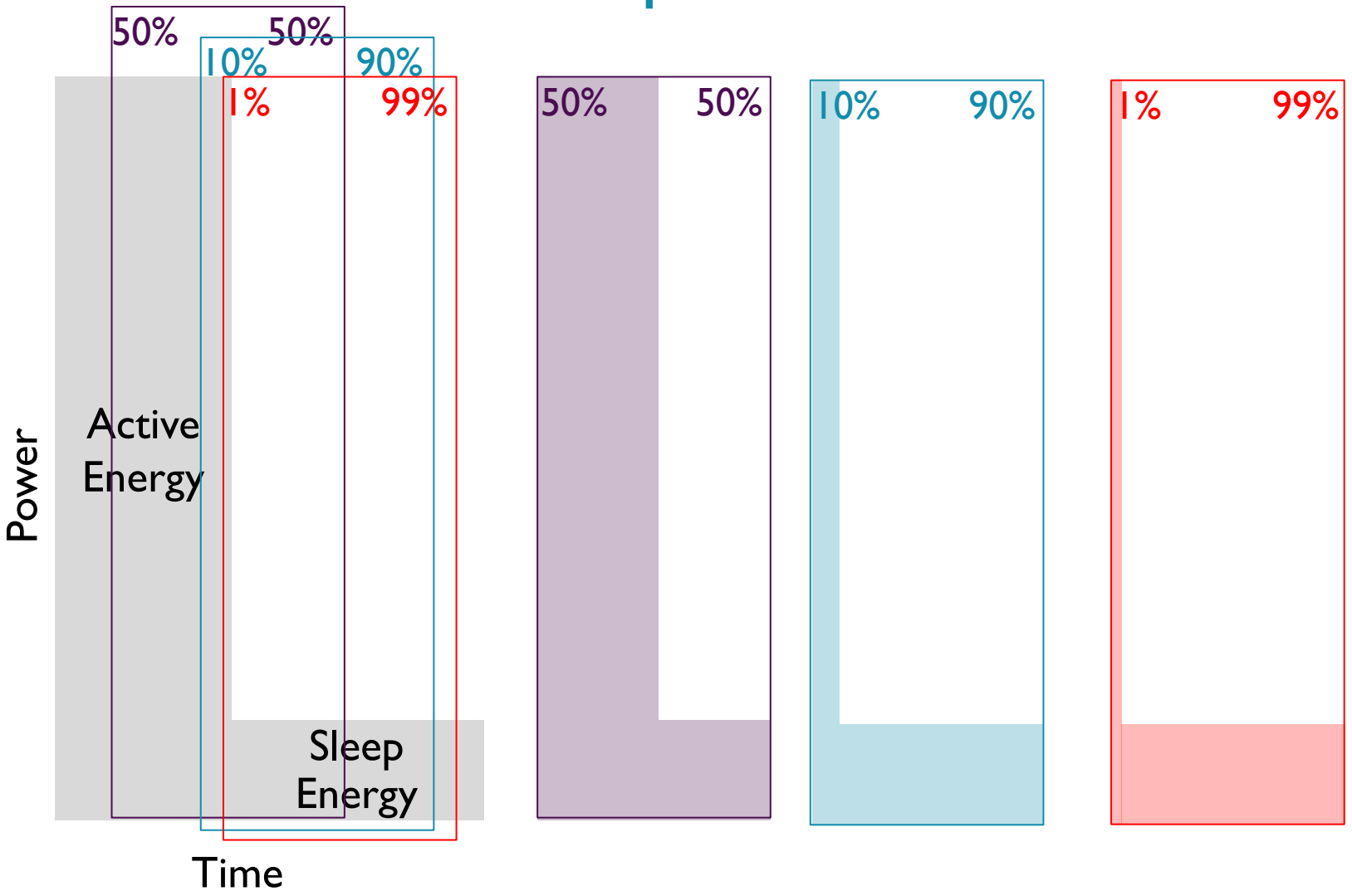
MCU Active Power Use Varies with CPU Frequency



- Speed up f_{CPU} from 10 MHz to 20 MHz?
  - Finish 1.5 ms earlier, but use more power while active.
  - Is average power higher or lower? Depends
- Average MCU Power (Proportional to Energy)
  - $P_{MCU} = D_{Active} * P_{Active} + D_{Sleep} * P_{Sleep}$
  - $P_{MCU} = P_{Active} * (C/f_{CPU} + T_{const}) + P_{Sleep} * (1 - C/f_{CPU} - T_{const})$

Which Energy Dominates? Active or Sleep?

- Hardware and Firmware Issues in Using Ultra-Low Power MCUs, Jack Ganssle
- Consider fraction of time spent in sleep mode
  - 50% asleep
  - 90% asleep
  - 99 % asleep
- The more the MCU sleeps, the more sleep power matters
- **Average** MCU Power (proportional to energy)
  - $P_{MCU} = P_{Active} * D_{Active} + P_{Sleep} * D_{Sleep}$
  - $P_{MCU} = P_{Active}*(C/f_{CPU} + T_{const}) + P_{Sleep}*(1 - C/f_{CPU} - T_{const})$



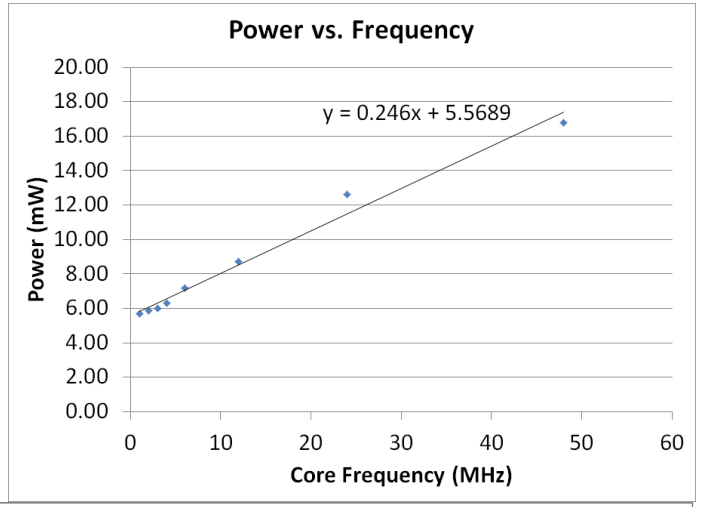
Energy = power (gray) within box

The Power of Sleep

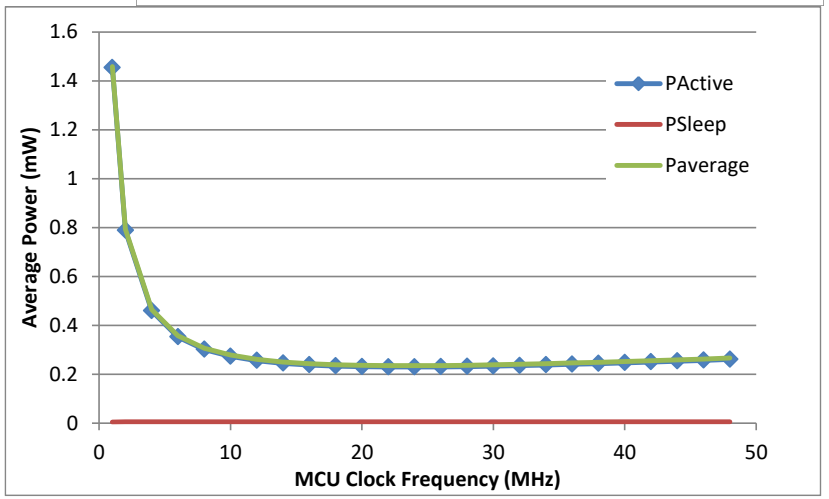
- The more the MCU sleeps, the more sleep power matters
- Optimize active or sleep energy (average power)?
  - Where is the cross-over point (active energy = sleep energy)?
  - Example: CPU has D = 1/10
  - $P_{MCU} = 1/10 * P_{Active} + 9/10 * P_{Sleep}$
  - Examine point where active and sleep energies (ave. powers) are equal
    - $0.1 * P_{Active} = 0.9 * P_{Sleep}$
    - Rewrite as $P_{Active}/P_{Sleep} = 0.9/0.1 = 9$
  - Sleep power will dominate energy use (average power) if $P_{Sleep} > P_{Active}/9$
  - D = 0.001? Cross-over point is $P_{Sleep} = P_{Active}/999$

Use Both Sleep and Clock Scaling

- What average power will be used at a given MCU frequency f_{CPU}?
  - Weighted sum of power used in active and standby
  - $P_{MCU} = P_{Active}*(C/f_{CPU} + T_{const}) + P_{Sleep}*(1 - C/f_{CPU} - T_{const})$

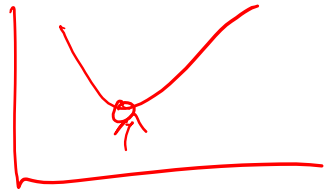


- Use linear model for active MCU power (mW) vs. frequency (MHz)
  - $P = P_{Active,Dynamic}*f_{CPU} + P_{Active,Static}$
  - $P = 0.246\ mW/MHz*f_{CPU} + 5.5689\ mW$



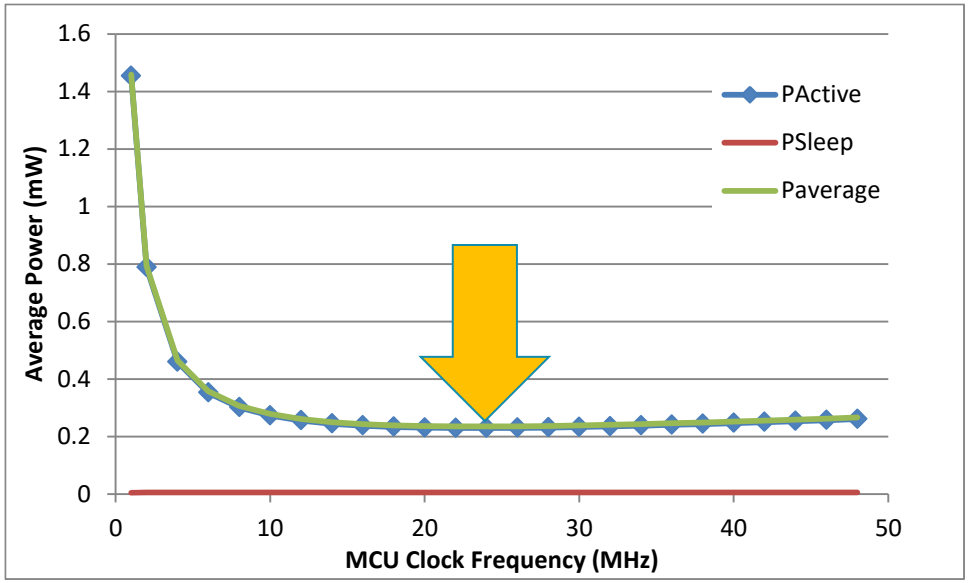
- Combine equations
  - $P_{MCU} = (P_{Active,Dynamic}*f_{CPU} + P_{Active,Static})(C/f_{CPU} + T_{const}) + P_{Sleep}*(1 - C/f_{CPU} - T_{const})$
  - Reducing f_{CPU} increases time which CPU must remain active, since must perform C cycles of computation

What is the Optimal Frequency?



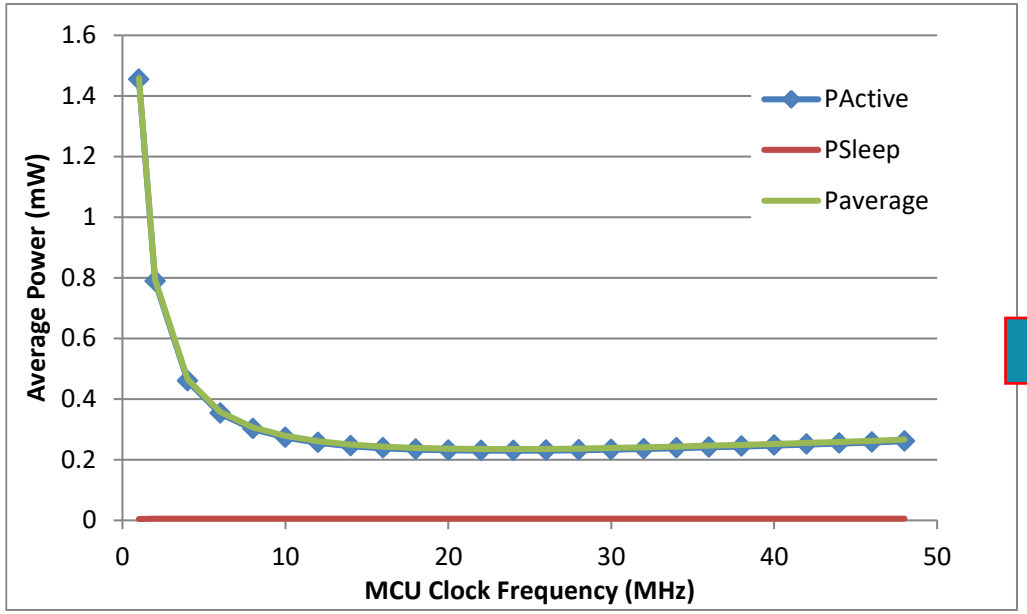
- Solve for frequency f_{opt} with minimum average power
  - Differentiate ave. power equation with respect to frequency
  - Solve for f_{opt} when derivative is 0 (minimum value of power)

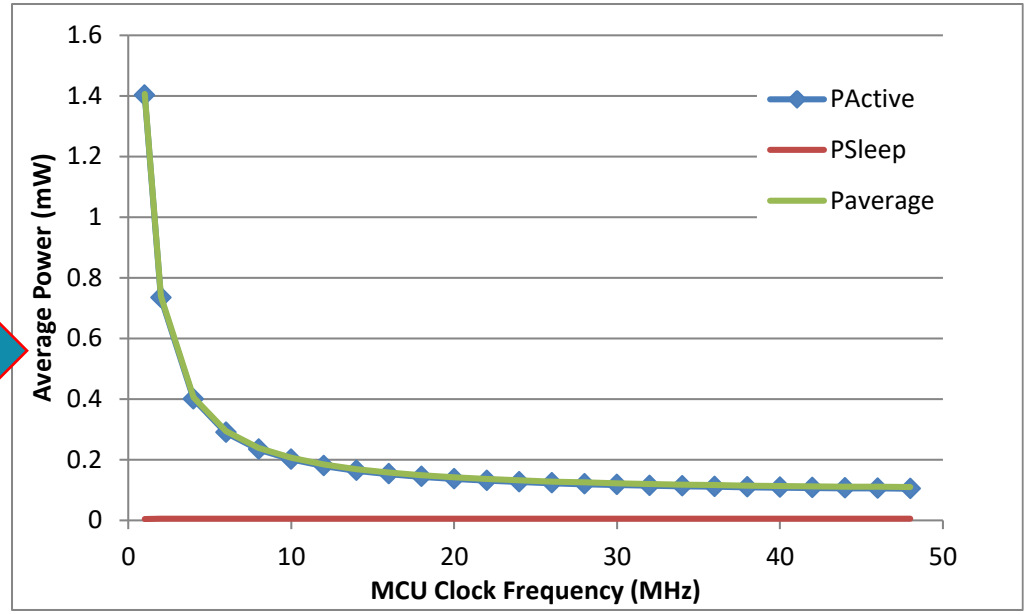
$$f_{opt} = \sqrt{\frac{C(P_{Active,Static} - P_{Sleep})}{P_{Active,Dynamic} T_{Const}}}$$



- Optimal frequency rises with
  - More computation
  - Increasing difference between active static power and sleep power
- Optimal frequency falls with
  - Higher active dynamic power
  - Longer wakeup overhead, ADC conversion times, communication times
    - Note: may be able to sleep for some of ADC, comm. times (if long enough)

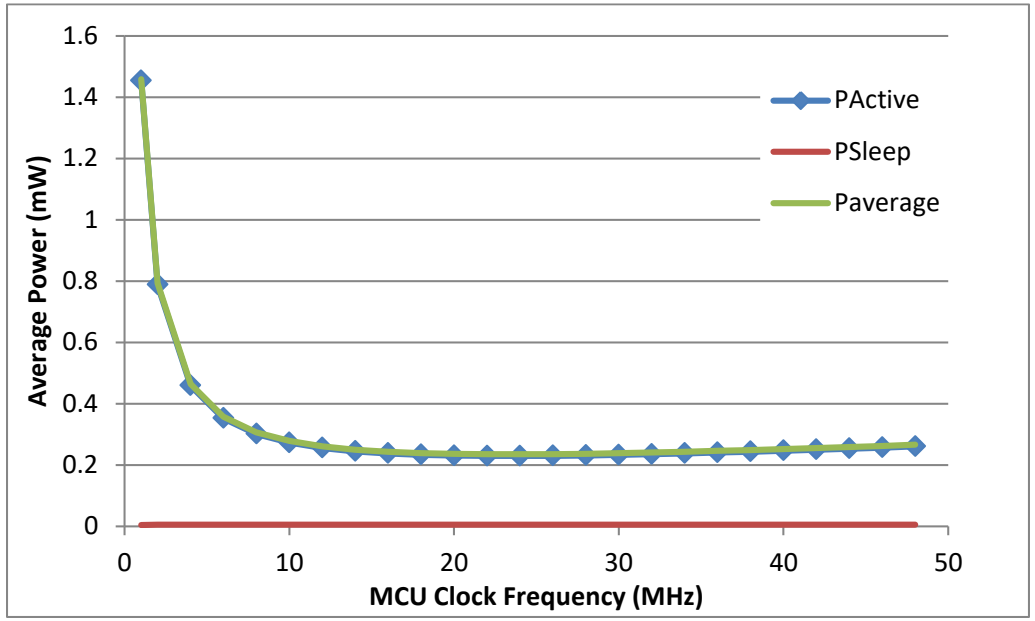
Factors Affecting Optimal Frequency

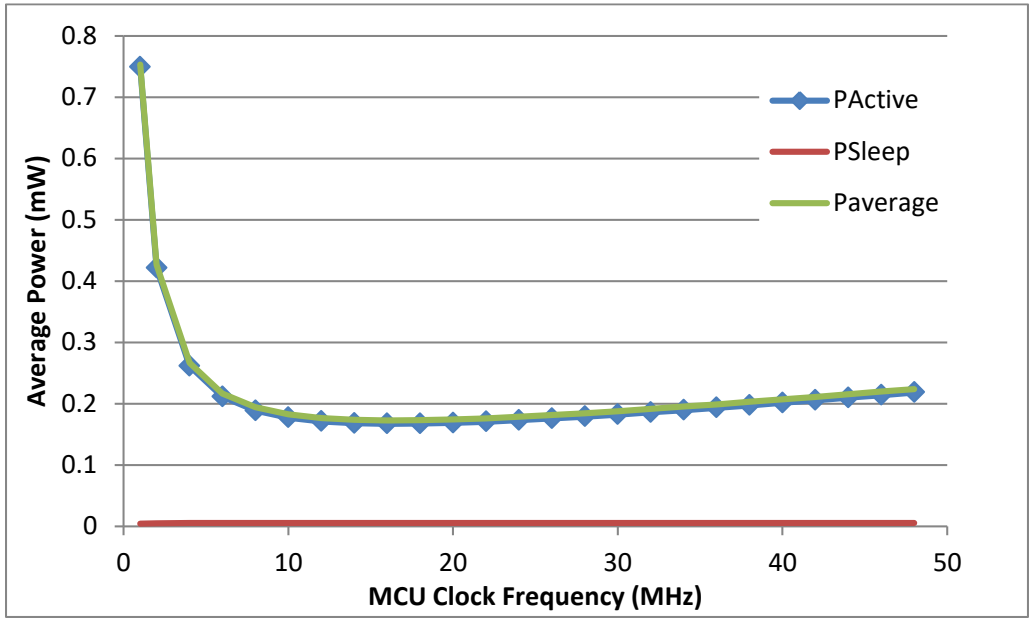




- Less wake-up overhead? ($T_{const}\downarrow$)
  - Higher frequency is better
  - Dropping T_{const} from 10 ms to 5 ms raises f_{opt} from 24 MHz to 32 MHz

Factors Affecting Optimal Frequency





- Less static active power? ($P_{Active,Static}\downarrow$)
  - Lower frequency is better
  - Dropping $P_{Active,Static}$ from 5.5689 mW to 2.75 mW changes f_{opt} from 24 MHz to 16 MHz

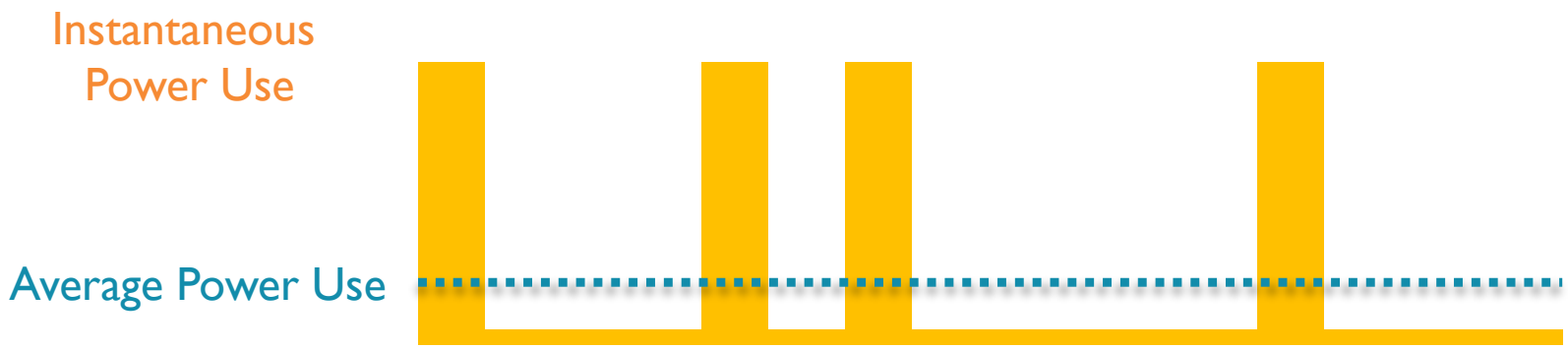
Factors Affecting Optimal Frequency

- Less dynamic active power dependence on frequency? ($P_{Active,Dynamic}$ ↓)
  - Higher frequency is better
  - Dropping $P_{Active,Dynamic}$ from 0.246 mW/MHz to 0.123 mW/MHz changes f_{opt} from 24 MHz to 32 MHz

- Less standby power? (P_{Sleep}↓)
  - Dropping P_{Sleep} from 5.7 uW to 2.85 uW doesn't change f_{opt} significantly

Power Management and Software Task Schedulers

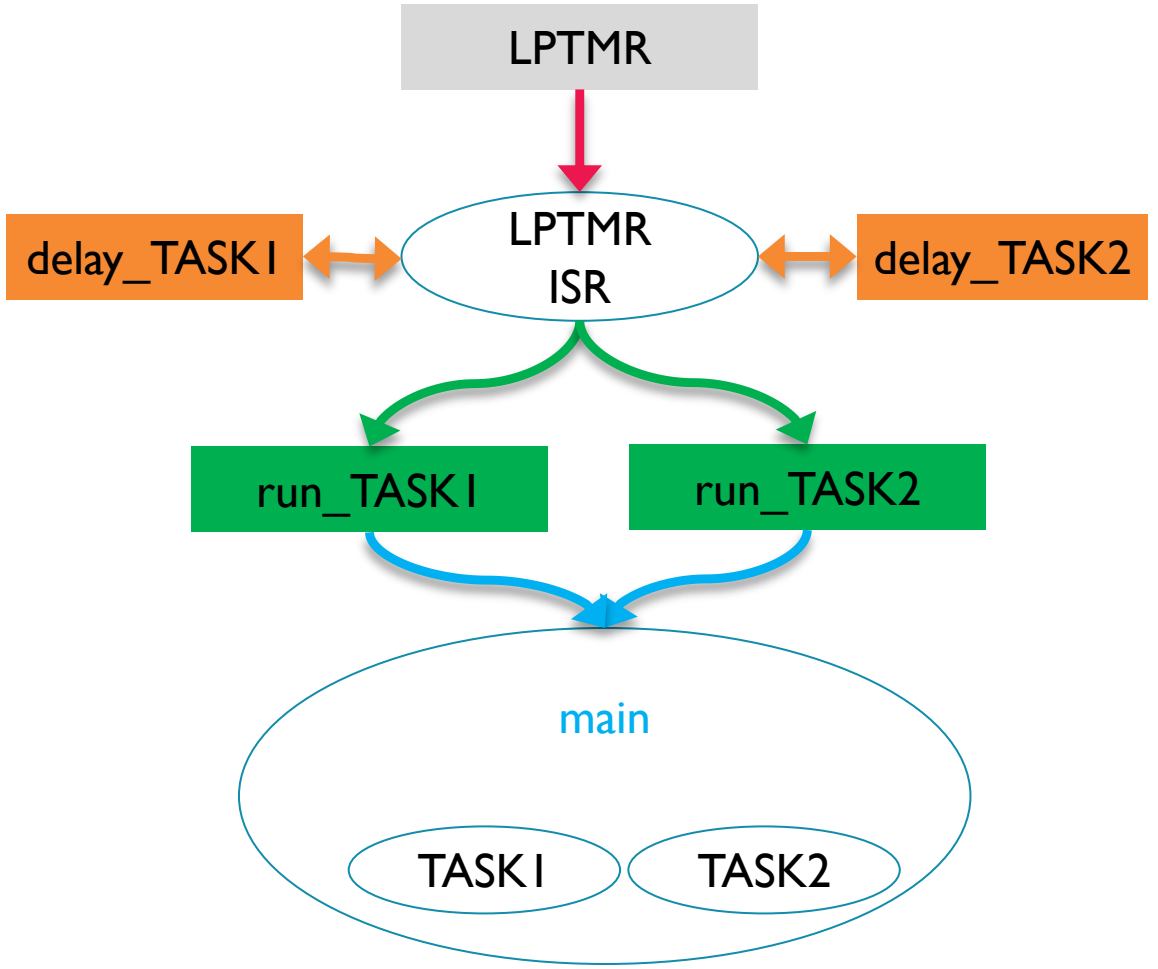
Basic Concepts



- Put CPU to sleep when there is no work to do
- Wake up CPU with interrupt from…
  - Event occurring: ADC conversion completing, I^2C transmission completing, switch press
  - Time delay completing: Timer interrupt
- How do we know there's no work to do?
  - End of loop or function (in simple systems)
  - Explicit busy-wait delay operations (e.g. delay for 10 ms)
  - Scheduler executes idle thread

No Scheduler or OS?

- Use low-power timer (LPTMR) to generate periodic interrupt request
  - LPTMR operates in ***all*** low-power modes
- LPTMR ISR (IRQ handler) becomes part of a simple scheduler – tick management code
  - Decrements each delay counter
  - If delay counter reaches 0,
    - Release task: set task's run flag
    - Reload delay counter with period value
- Main loop becomes rest of simple scheduler – task dispatcher
  - If a task's run flag is set, then clear the flag and execute that task
  - Go back to sleep



Code for Timer-Interrupt-Driven Approach

```
volatile uint8_t run_Read_Accel=0;
volatile uint8t run_Update_LEDs=0;
volatile int delay_Read_Accel =
PERIOD_READ_ACCEL;
volatile int delay_Update_LEDs =
PERIOD_UPDATE_LEDS;

void LPTimer_IRQHandler(void) {
  …
  delay_Read_Accel--;
  if (delay_Read_Accel == 0) {
    run_Read_Accel = 1;
    delay_Read_Accel=PERIOD_READ_ACCEL;
  }
  delay_Update_LEDs--;
  if (delay_Update_LEDs == 0) {
    run_Update_LEDs = 1;
    delay_Update_LEDs=PERIOD_UPDATE_LEDS;
  }
}
```

```
void main (void) {
  …
  while (1) {
    if (run_Read_Accel) {
      run_Read_Accel = 0;
      Read_Accel();
    }
    if (run_Update_LEDs) {
      run_Update_LEDs = 0;
      Update_LEDs();
    }
    __wfi(); // go to sleep
}
```

CPU Activity Timeline

Run-to-Completion Scheduler

- Schedulers typically built on periodic timer tick generated by hardware timer expiring
  - ISR runs scheduler's tick update code, which updates delay counters, releases appropriate tasks (marks tasks as **ready** to run)
  - Scheduler's dispatcher code actually starts and stops tasks running
- Modify scheduler's dispatcher to put processor to sleep when no tasks are ready to run
- RTC scheduler
  - Has loop which runs tasks in priority order
  - Loop gets to bottom only if no tasks are ready to run
  - Go to sleep there.
  - CPU will wake up with next timer tick or other interrupt

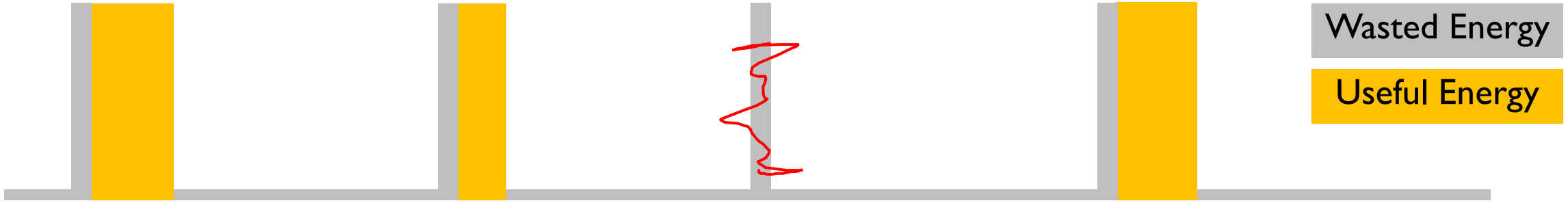
```
void Run_RTC_Scheduler(void)
{
  …
  /* Dispatcher loops forever */
  while (1) {
    /* Check each task */
    for (i=0 ; i<MAX_TASKS ; i++) {
      /* Run task if ready*/
      if (…){
        GBL_task_list[i].task();
        …
       break;
      }
    }
    // no tasks ready to run
    __wfi(); // go to sleep
  }
}
```

RTX v5 (Preemptive Scheduler)

```
#include "RTE_Components.h"
#include CMSIS_device_header /* Device definitions */
void osRtxIdleThread (void) {
      /* The idle demon is a system thread, running when no other */
      /* thread is ready to run. */
      for (;;) {
            __WFI(); /* Enter sleep mode */
      }
}
```

- (We are using RTX v5 with a CMSIS-RTOS2 interface)
- Scheduler runs idle thread when there is no work -> Put CPU into a sleep mode there
- Use WFI (Cortex-M0+) or WFE (Cortex-M3 and others)
- CPU will wake up with next timer tick or other interrupt
- Details: http://www.keil.com/pack/doc/CMSIS/RTOS2/html/theory_of_operation.html#lowPower

Tickless Idle



- What if timer tick wakes up CPU and scheduler finds no work to do?
  - CPU goes back to sleep
  - Waste of time to wake up for a timer tick which won't cause anything to run
- How about delaying timer tick until the next scheduled event?
  - Will eliminate multiple wake-ups
  - Other interrupts (e.g. UART reception) will still wake up CPU -> correct operation
- To make it work, we need to…
  - Ask scheduler for how many ticks we can sleep
  - Reconfigure tick timer period, or use different timer
  - On wake-up, tell scheduler how long we slept
    - Easy if awakened by timer
    - More complex if awakened by non-timer interrupt

Tickless Idle for RTX v5

- RTX uses SysTick timer to generate periodic interrupts (scheduler ticks)
- Idle thread decides whether to go to sleep
- Go to sleep?
  - Call **osKernelSuspend** to determine delay (in ticks) until next scheduled processing
    - If delay == 0, then stay awake
- Going to sleep
  - Set wake-up timer for that delay
  - Disable interrupts from SysTick
  - Go to sleep
- Waking up
  - Caused by wake-up timer or other interrupt
  - Determine how much time we slept
    - If not caused by wake-up timer, then not likely to be the delay we set above
  - Start scheduler again by calling **osKernelResume**, telling it how long we slept
- Consider timing error from …
  - Switching between clocks if not synchronized
  - Sampling effects of clocks
- More at https://www.keil.com/pack/doc/CMSIS/RTOS2/html/theory_of_operation.html#TickLess

Design Example with Analysis

System Overview – Drift Meter

- Drift Meter – find difference between vehicle heading and actual track to find effects of crosswinds, skidding, water currents
- Two operating modes
  - Active: all devices operating
  - Standby: system mostly off, use accelerometer to determine orientation and power state
- Approach – Analyze consumption, then start with addressing the worst offender
- Consider major operating modes: active, sleep (“off”)

Create Power Model

		Active		Standby			
Device	Voltage (V)	Current (mA)	Power (mW)	Current (mA)	Power (mW)	Duty Cycle	Average Power (mW)
LCD	3.300	25.600	84.480			100%	84.480
GPS Receiver	3.300	20.000	66.000			100%	66.000
MCU	3.300	5.800	19.140			100%	19.140
MicroSD card	3.300	150.000	495.000	0.250	0.825	1%	5.767
Compass	3.300	0.360	1.188	0.002		100%	1.188
Accelerometer	3.300	0.024	0.079			100%	0.079

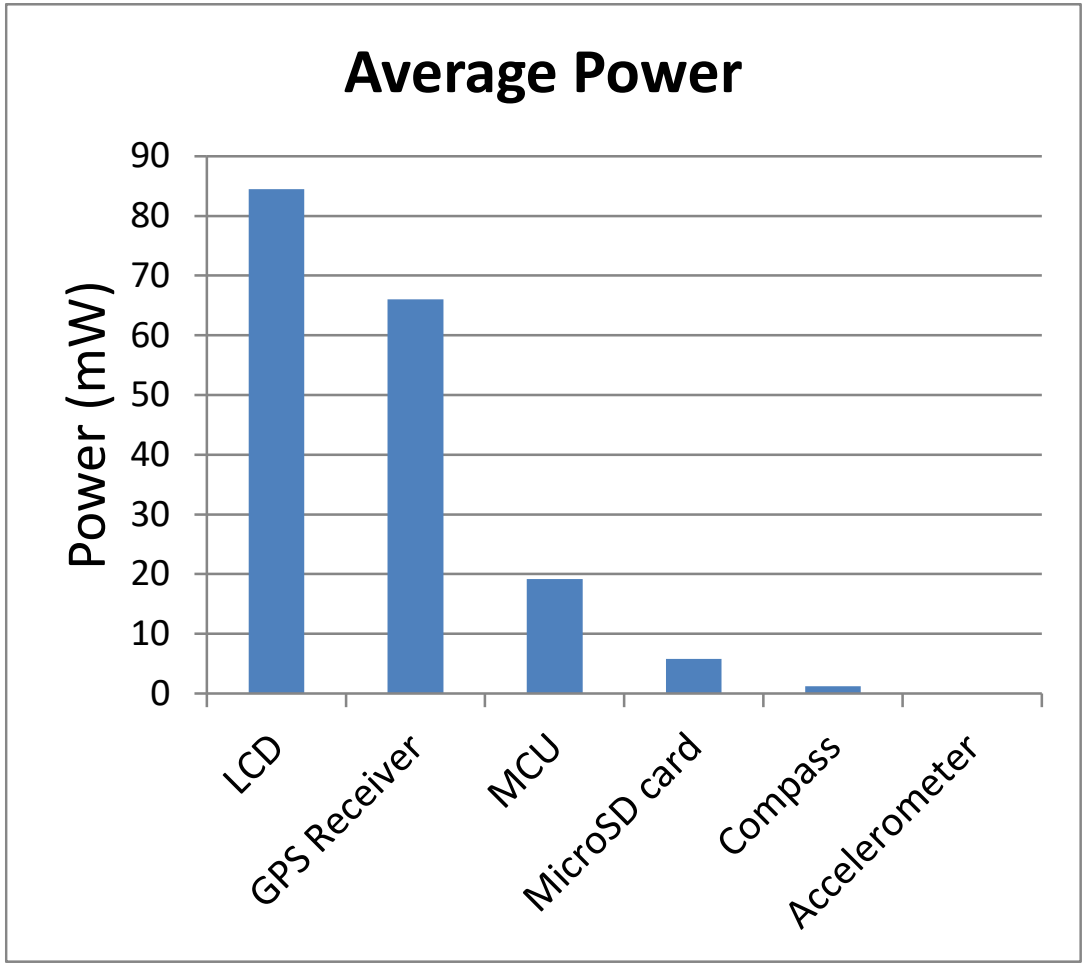
- Start with basic system without premature optimization
  - No MCU power management - MCU is always on, running at 48 MHz
  - Minimize number of voltage domains - all devices run at 3.3 V
- Duty cycle
  - Must consider for microSD card due to large variation in current
    - Max I_{write} @ 3.6 V = 150 mA
    - Max $I_{standby}$ @ 3.0 V = 0.25 mA
  - Start off by assuming 1% duty cycle (1% in write, 99% in standby)

Battery Characteristics

- Use a Lithium-Ion rechargeable cell (e.g. from phone)
- Nominal 3.7V, rated at 1 Amp-Hour
  - Starts at about 4.2V
  - Cuts off about 3.0V
- Total Power
  - 3.7V * 1 AH = 3.7WH
- Total Energy
  - 3.7V * 1 AH * 3600 sec/H = 13.32kJ

Active Power – 100% Efficient Supply

- Total power of 176.7 mW
- Dominated by LCD and GPS receiver
- Battery life = 3.7 WH/0.1767 W = 20.93 H
- OK?
  - We are done!
- Not OK?
  - Increase storage
    - Use a larger battery
  - Reduce consumption
    - Find a more efficient LCD, GPS receiver

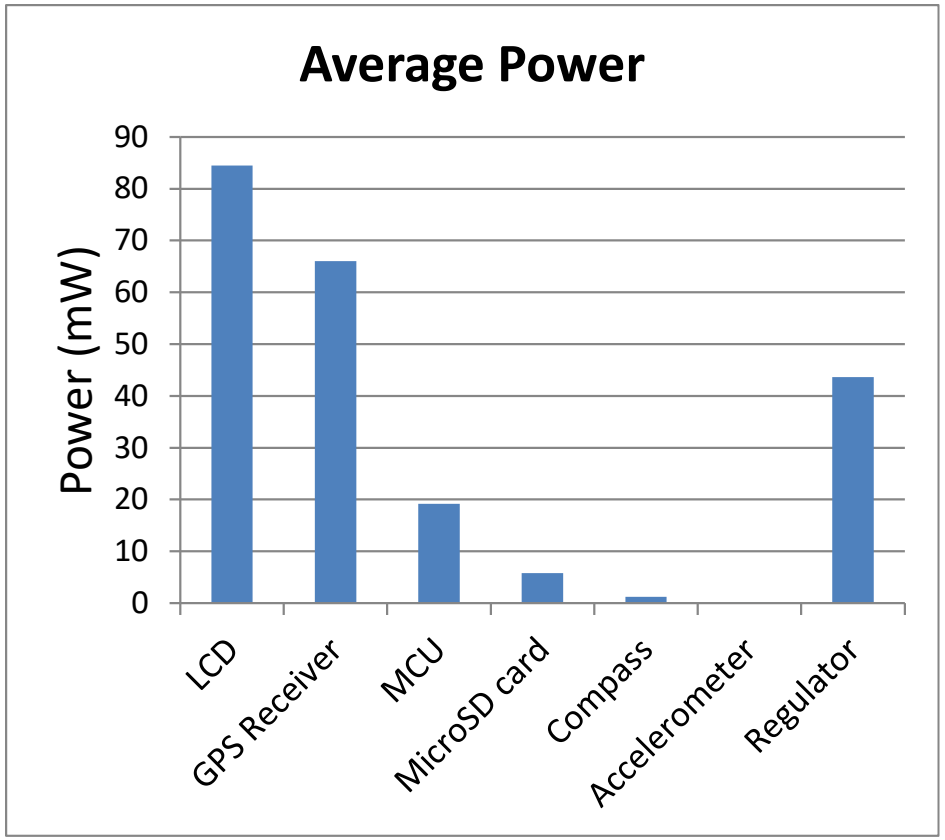


Power Conversion Efficiency

- Is voltage regulation needed?
  - Cell output voltage varies from 3 to 4.2 V
    - Exceeds maximum voltage for some components (SD, etc.)
- Need to regulate or at least limit voltage
  - Linear regulator – wastes more power if there is a large voltage drop
    - Freedom KL25Z uses NCP1117
    - Select regulator with small quiescent current, small dropout voltage
  - Switching regulator – more efficient
    - More complex as well

Active Power – Linear Voltage Regulator

- Add in linear regulator power
  - $P_{VReg} = (V_{in}-V_{out})*(I_{out}) + V_{in}*I_{Quiescent}$
  - 43.6 mW
- Total power of 220.3 mW
  - Still dominated by LCD and GPS receiver
  - Battery life = 3.7 WH/0.2203 W = 16.8 H
- Is this OK?
  - Yes
    - We are done!
  - No
    - Increase storage
      - Use a larger battery
    - Reduce consumption
      - Use lower power LCD (e.g. memory LCD, e-paper, e-ink), GPS receiver, regulator (power converter)



Standby Power – Linear Voltage Regulator

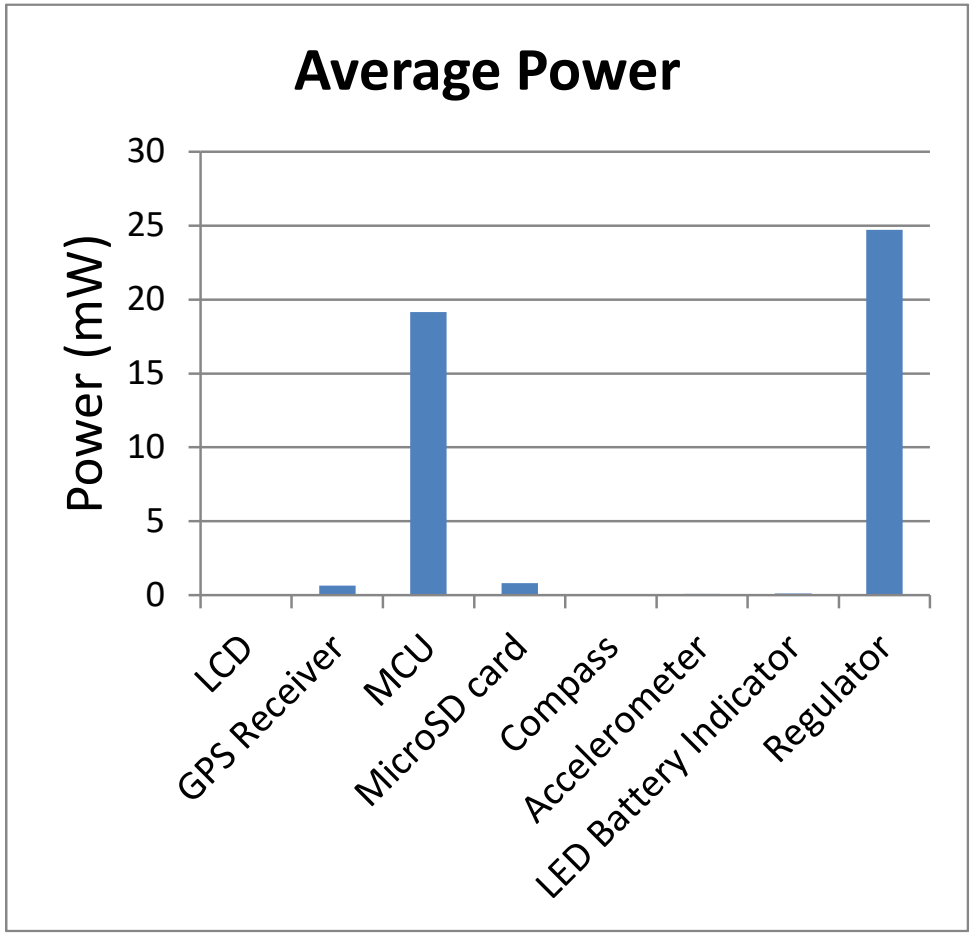
- Disable everything but …
  - MCU
  - Accelerometer
  - LED battery indicator (1%)
- Request standby mode
  - With message
    - Accelerometer – I2C
    - Compass – I2C
    - GPS receiver – UART
    - microSD card – SPI – GO_IDLE_STATE command (0)
  - With logic level signal
    - MicroSD card – chip select - /CS

	Standby	
Device	Current (mA)	Power (mW)
LCD	0.000	0.000
GPS Receiver	0.200	0.660
MCU	n/a	n/a
MicroSD card	0.250	0.825
Compass	0.002	0.007
Accelerometer	n/a	n/a
LED Battery Indicator	0	0.000

- Brute force methods
  - Hold device in reset state
  - Shut off power
    - Needed for this LCD module

Standby – Linear Voltage Regulator

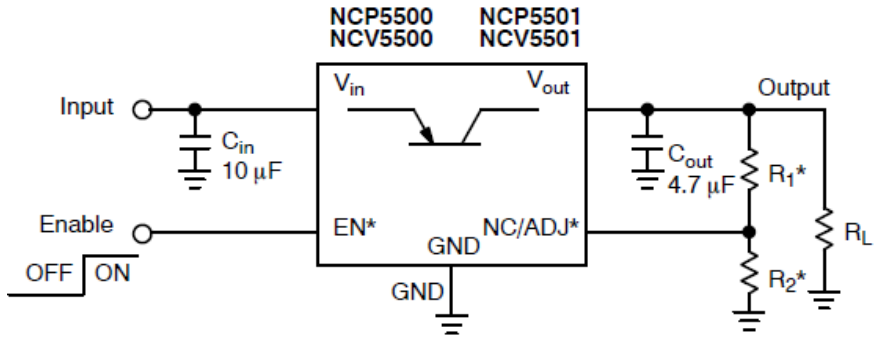
- Results
  - Average power is 45.5 mW (220.3 mW)
  - Battery life = 3.7 WH/45.5 mW = 81.2 H
  - Regulator and MCU now dominate power
- What can we do now?
- Reduce regulator power
  - Can we find a more efficient regulator (with less quiescent current)?
  - Can we even get rid of the regulator?
    - All components must be able to handle maximum battery voltage (4.3 V for Li)

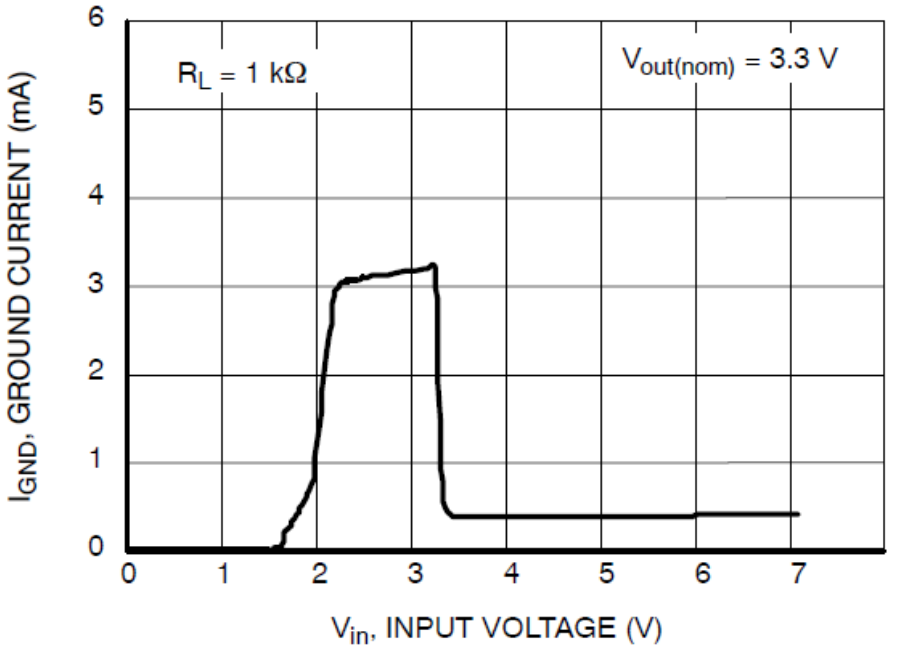


Improving the Linear Voltage Regulator

- Select regulator with …
  - Small quiescent current
  - Small dropout voltage
  - Adjustable output voltage
  - Shutdown control signal

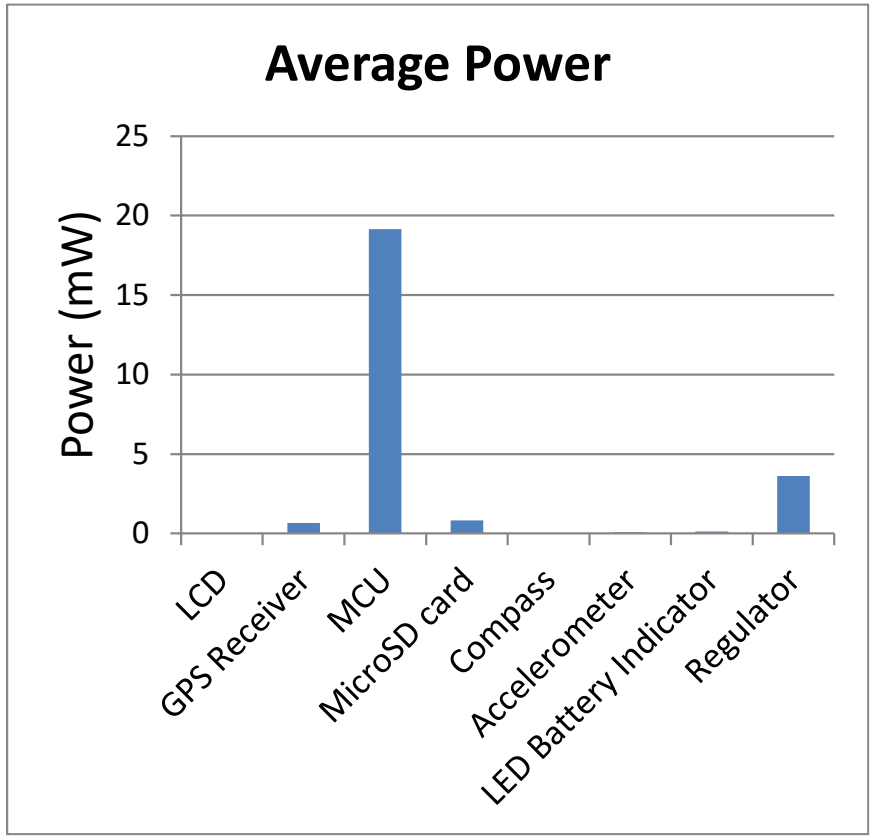
- Example: On Semiconductor NCP5500 500 mA LDO Voltage Regulator
  - Datasheet: NCP5500/D, April 2013, Rev. 12, http://onsemi.com
  - Low quiescent current: 300 uA
  - I_{out} up to 500 mA
  - What's going on between 2 and 3.3 V?
    - *Extra credit*





Resulting Power Use

- Good improvement from replacing voltage regulator!
  - 24.5 mW average power (was 45.4 mW)
  - 151.2 hour battery life in sleep mode
- Now focus on the MCU
- We could lower the operating voltage
  - But we need efficient power conversion (not a linear regulator)



- Do we really need 48,000,000 cycles of computation per second while the device is asleep?
  - If not, we can lower the clock speed or use a sleep mode

How Many Compute Cycles per Second?

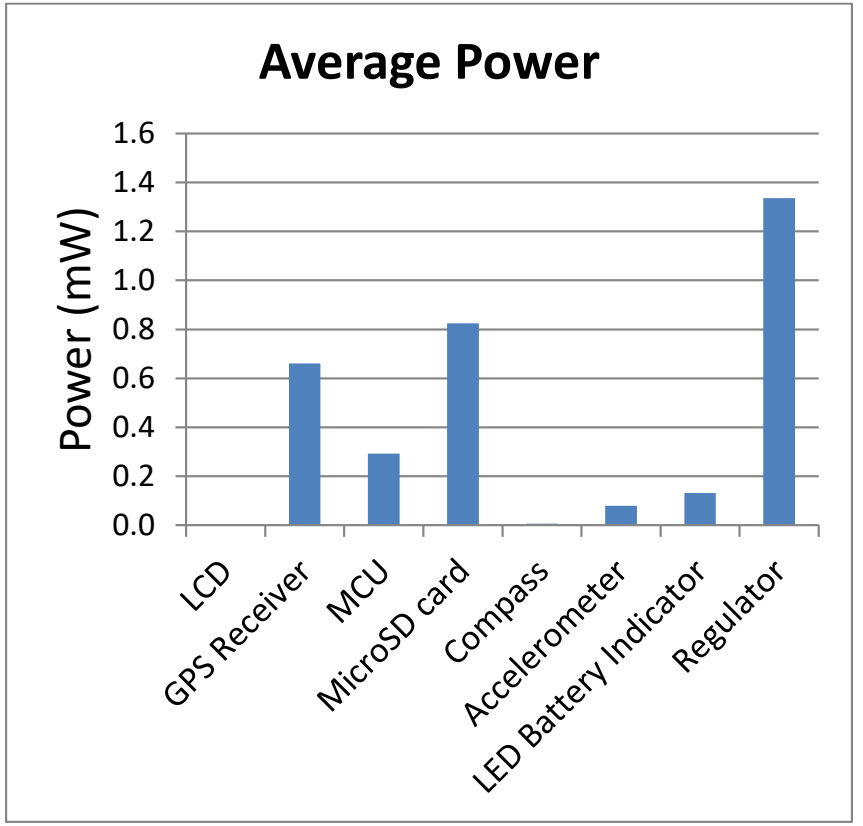
- We don't know, since we haven't written the code yet!

- Need to estimate it instead
  - Analyze activity, break down into smaller sub-activities
  - Estimate computation needed per sub-activity
  - Sum up computation costs – may need to scale if running at different frequencies

Sleep While Idle?

- Factors
  - Estimated C = 240 000 cycles of computation needed per second
  - T_{const} needed per second for constant-time activities (independent of MCU clock rate)
    - E.g. 1 ms per wakeup @ 10 Hz
  - f is MCU clock speed
- Use Low-Leakage Stop mode
  - 1.9 uA at 3 V = 5.7 uW
- Results
  - Average MCU power falls to 0.293 mW (from 24.5 mW)
  - Average system power is 1.336 mW
  - Battery life = 46.3 days

C	240000	cycles
PActive	19.14	mW
PSleep	0.0057	mW
Tconst	0.01	s



Review

- Process
  1. Start with a power or energy model
  2. Optimize the largest part
  3. Update model
  4. GOTO 2

- What can **you** do to minimize power consumption?
  - Circuit design
    - Choose power-efficient parts
      - Operate at a low voltage
      - Run at **low** frequency if **dynamic** power dominates
      - Turn off processor and other circuits if static power dominates
    - Use low-power modes or shut off parts
  - Program implementation
    - Minimize compute cycles needed

*The **squeakiest** wheel gets the grease **first***

- Power supply
  - Use an efficient power supply
  - Skip the power supply? Use devices which have a wide operating range
- Leverage low-power modes of processor and peripherals
  - Group processing together to minimize overhead of switching between active and idle modes
  - Use timers and external events to wake up
  - Use external hardware to reduce CPU on-time

Appendix

Estimating Compute Cycles per Second – Energy Lab

- What does the CPU do while device is in sleep mode?
  - Indicate battery voltage
    - Read voltage with ADC, flash LED accordingly
  - Decide based on orientation whether to wake up or stay asleep
    - Use accelerometer to measure orientation
- Derive rough estimate
  - Wakes up at 10 Hz
  - Measures voltage with ADC, does floating point computation
  - Duty cycle is about 500 uS/100 ms = 0.5%
- Only need about C = 48 MHz * 0.5% = 240 000 cycles per second of processing
- What to do about the remaining free cycles?
  - Could slow down processor clock
  - Could use a sleep mode
  - Could do both