Power and Energy Optimization

Overview

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

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

- 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
  - 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

ARM

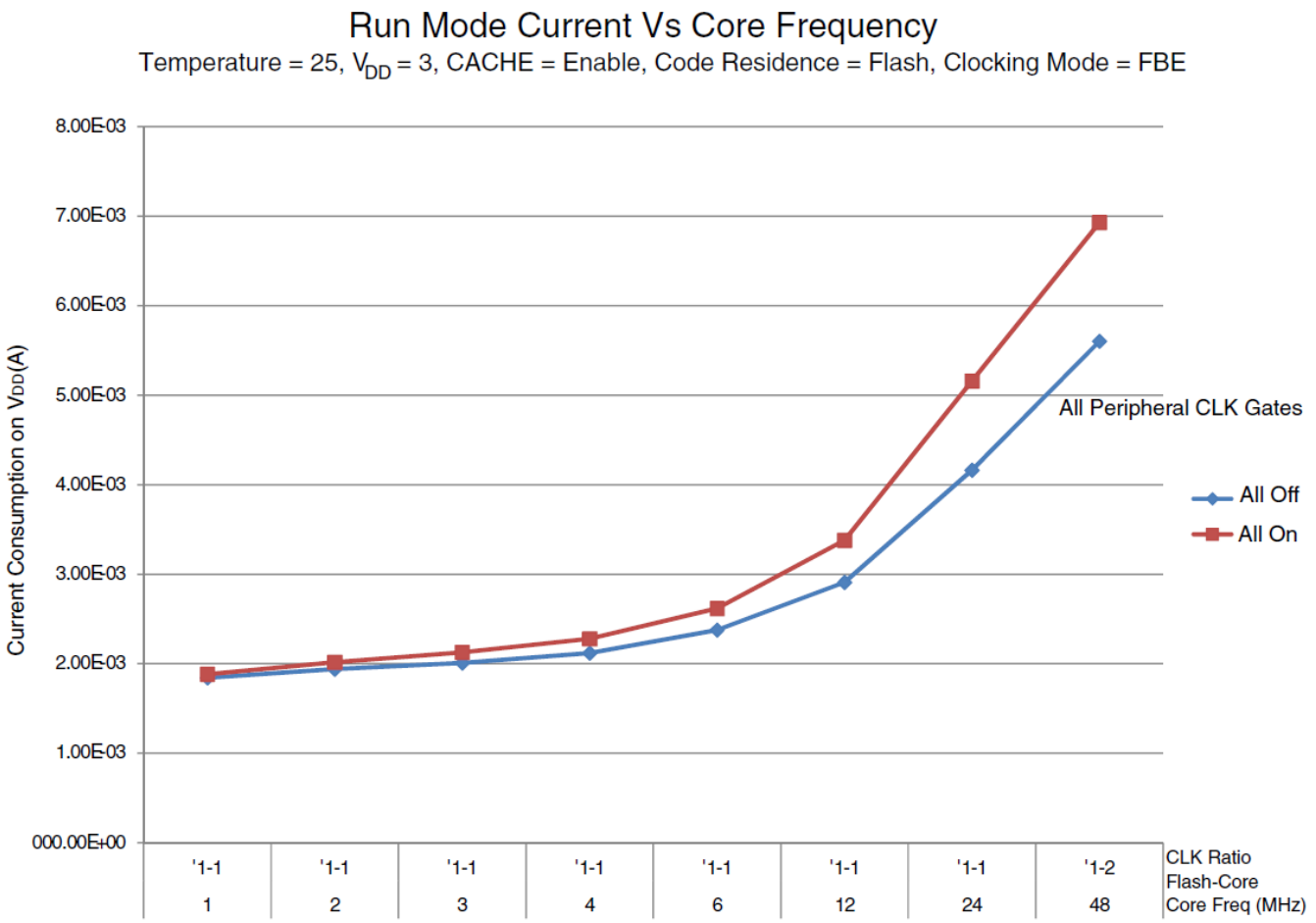
Voltage and Frequency Scaling

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: 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

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

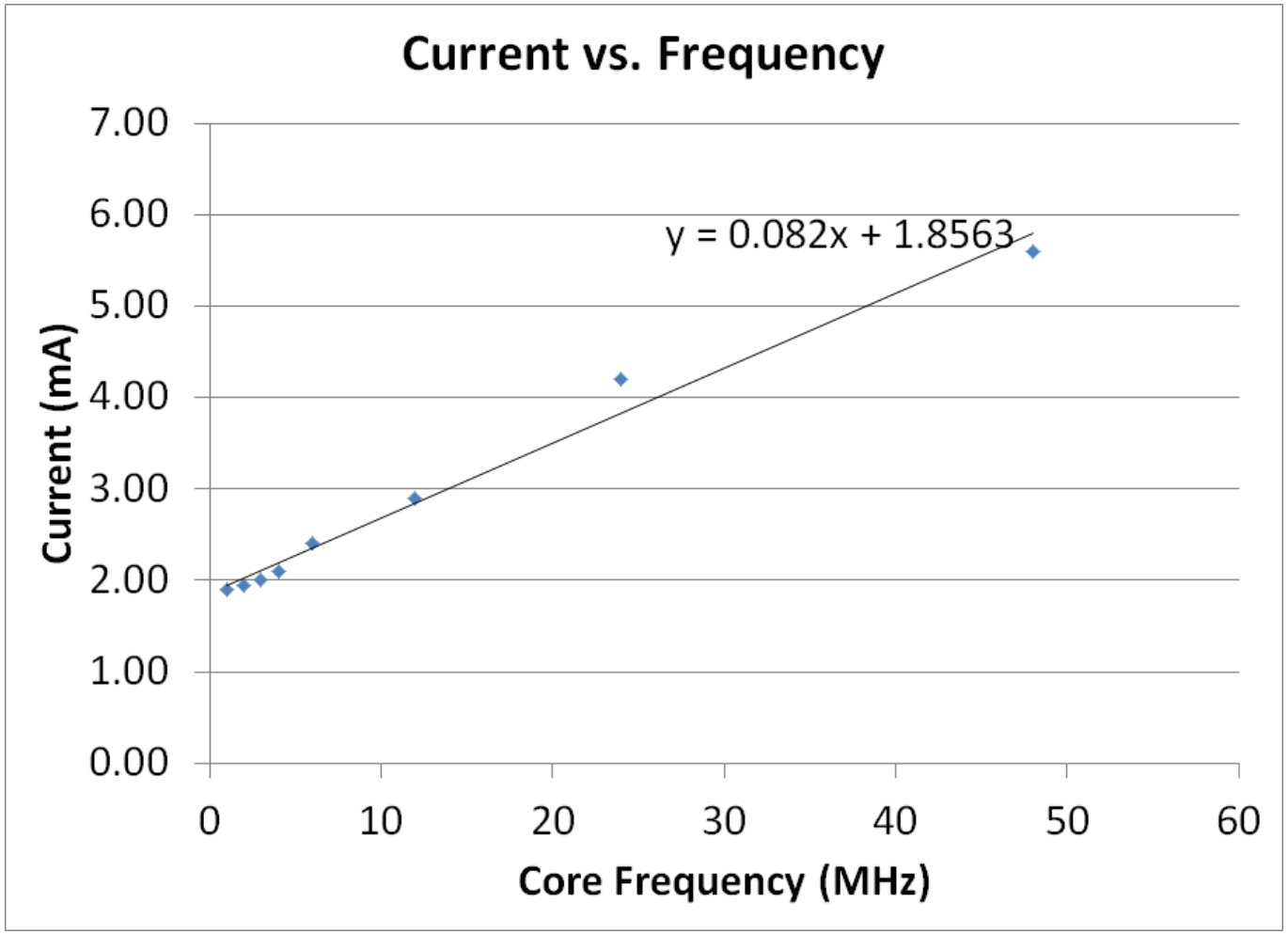
What is the Impact of Clock Speed on Current?



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

ARM

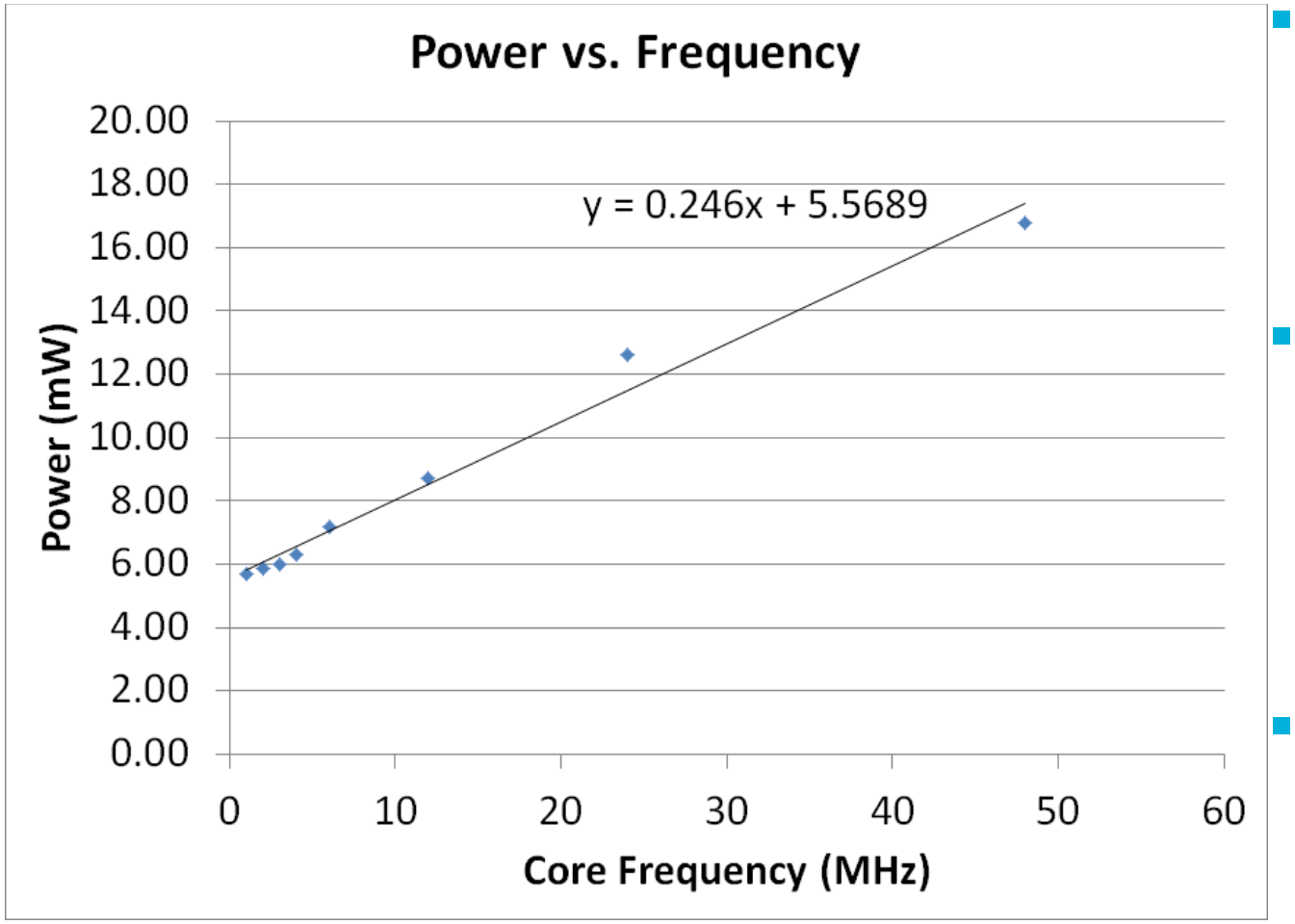
Current with Linear Axes



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

ARM

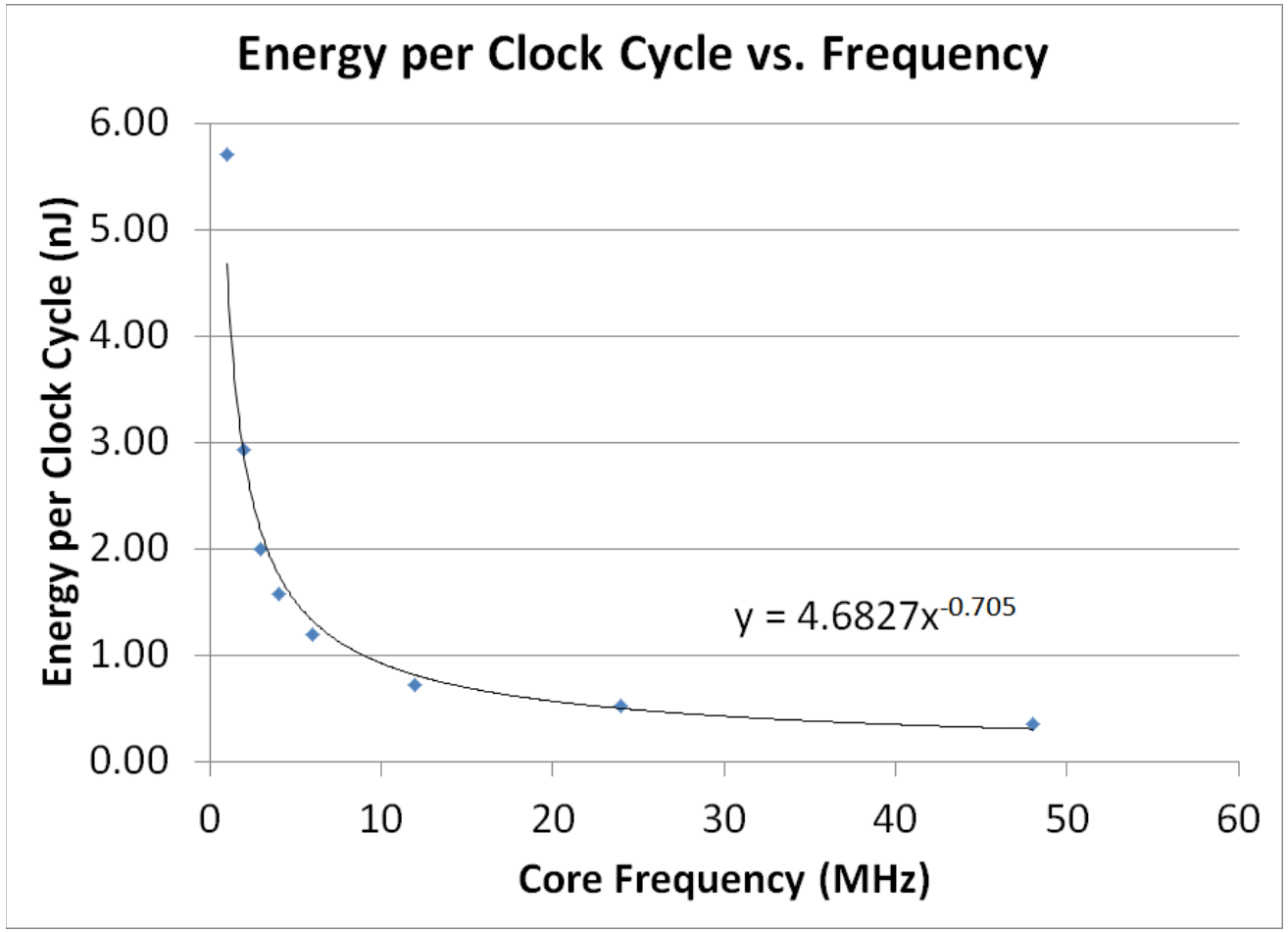
What is the Impact of Clock Speed on Power?



- 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

ARM

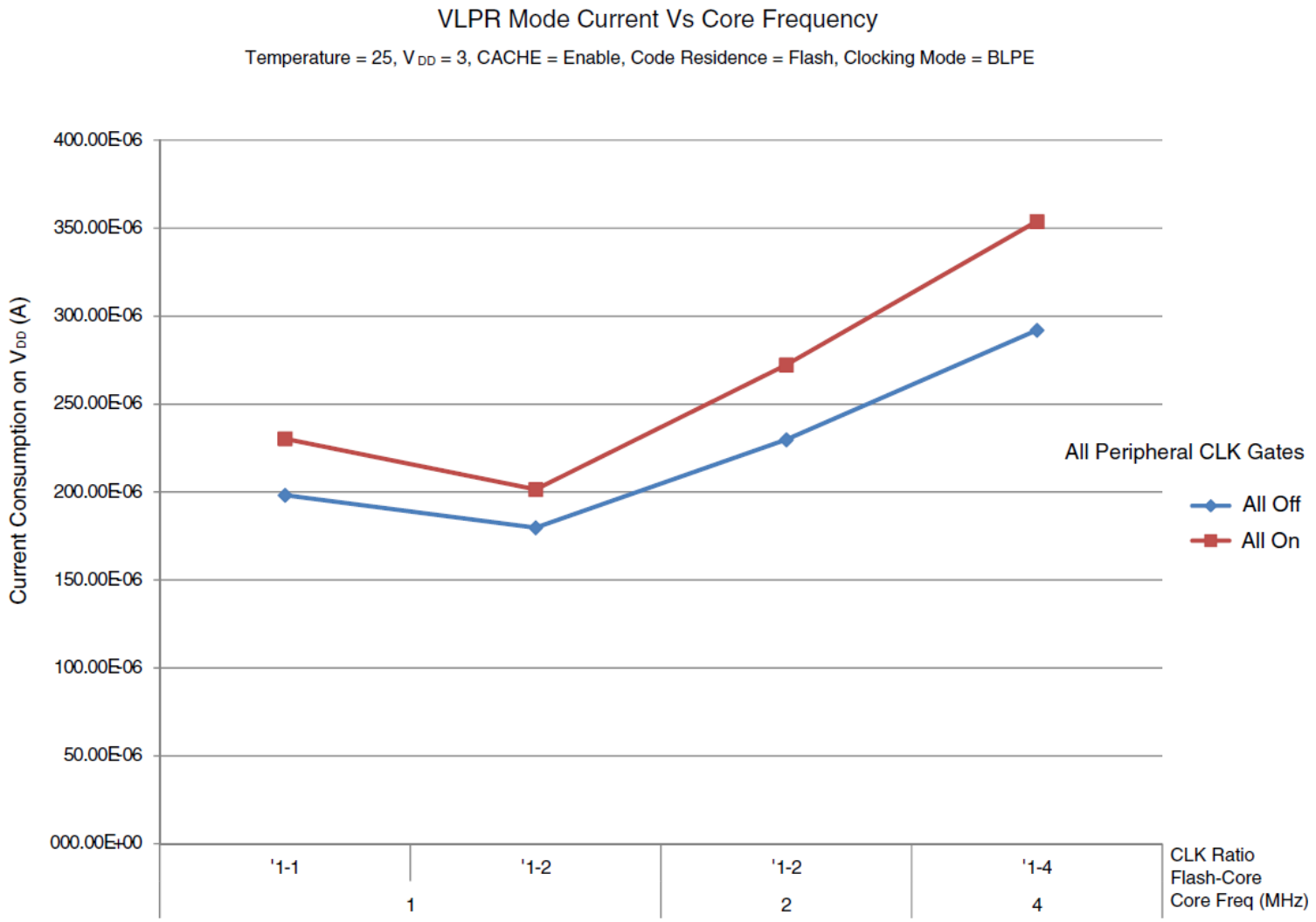
What is the Impact of Clock Speed on Energy?



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

ARM

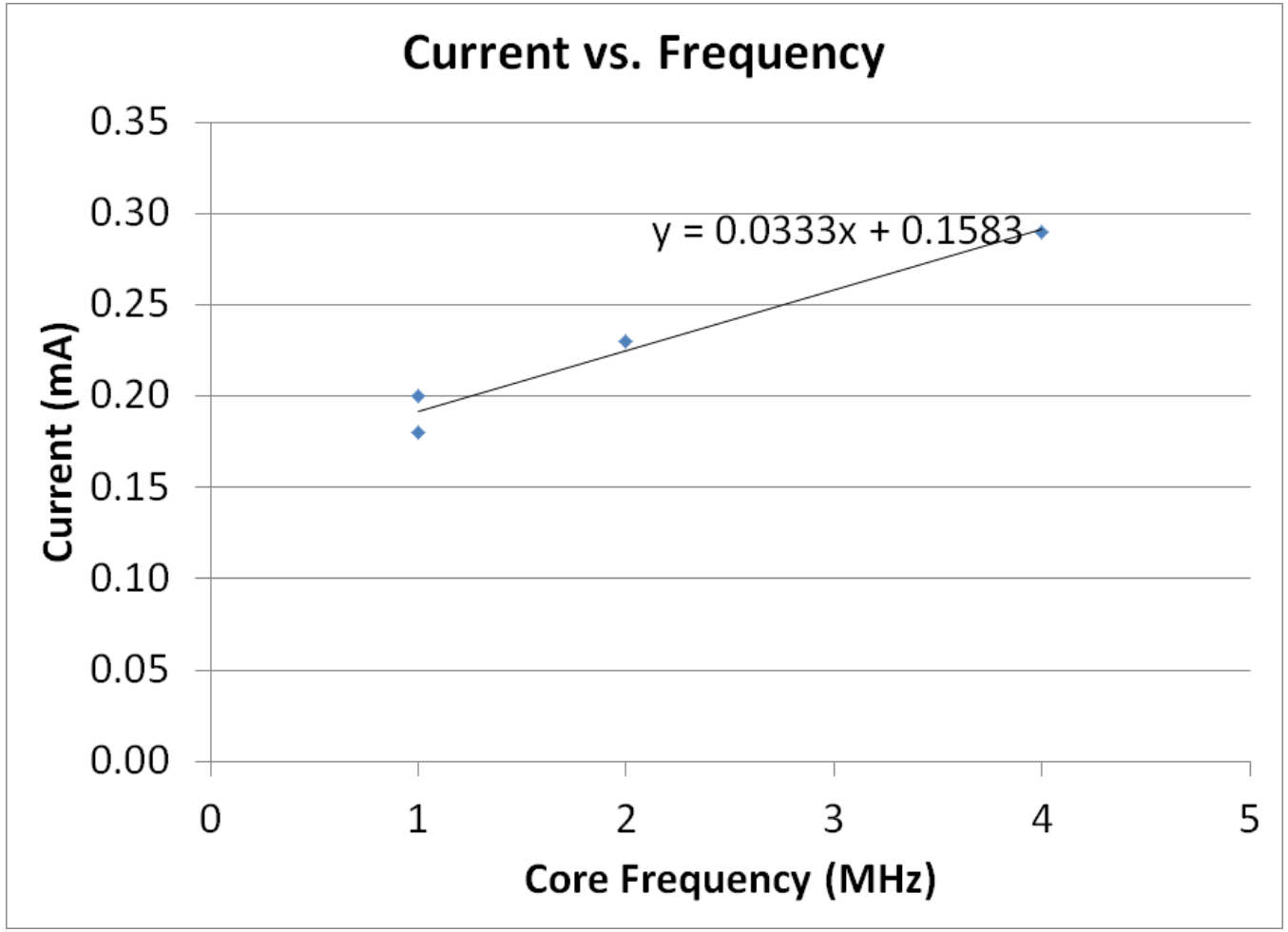
Current vs. Clock Speed – VLPR Mode



- 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

ARM

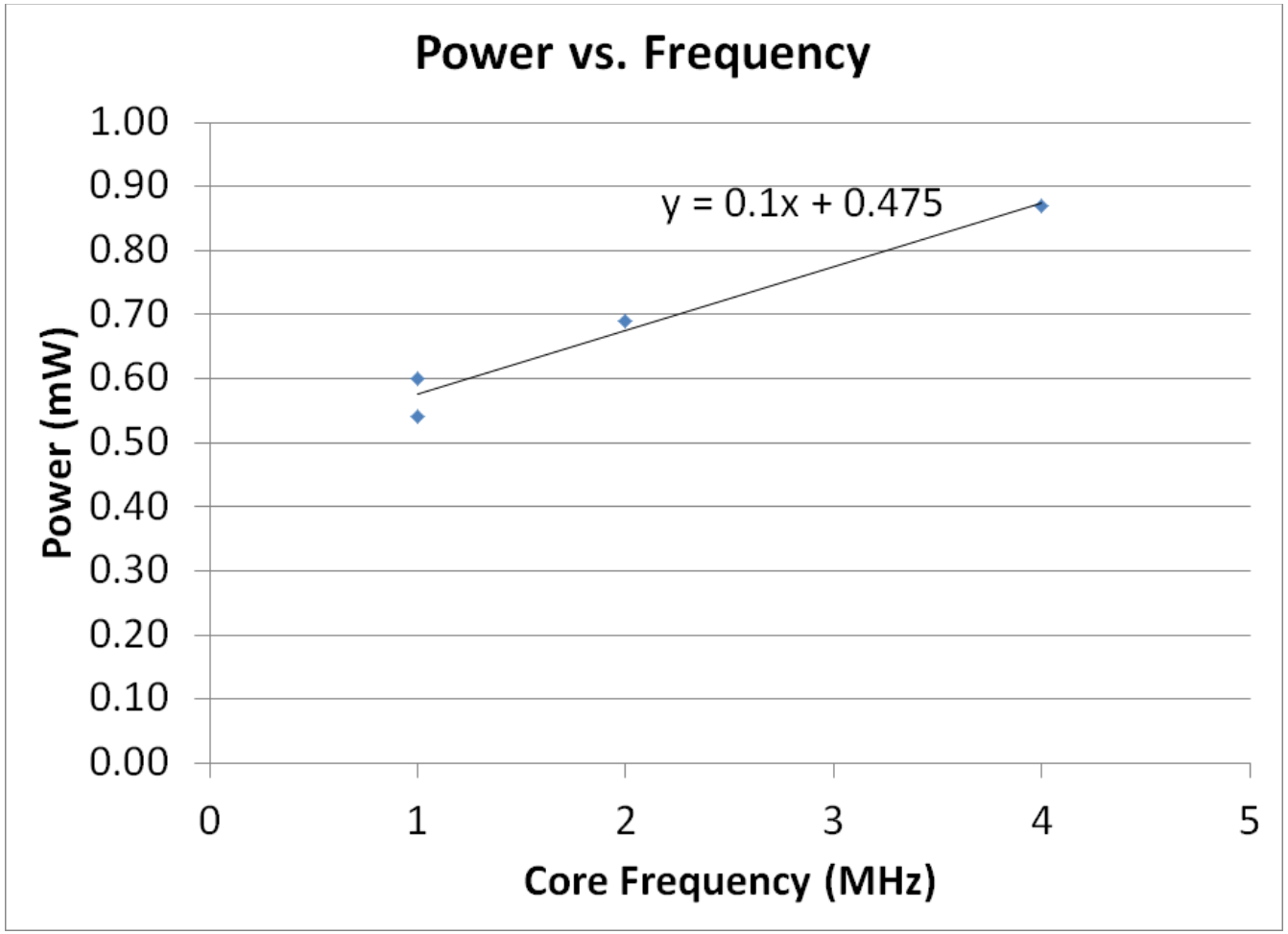
Current with Linear Axes



- Current linearly rises with core clock frequency

ARM

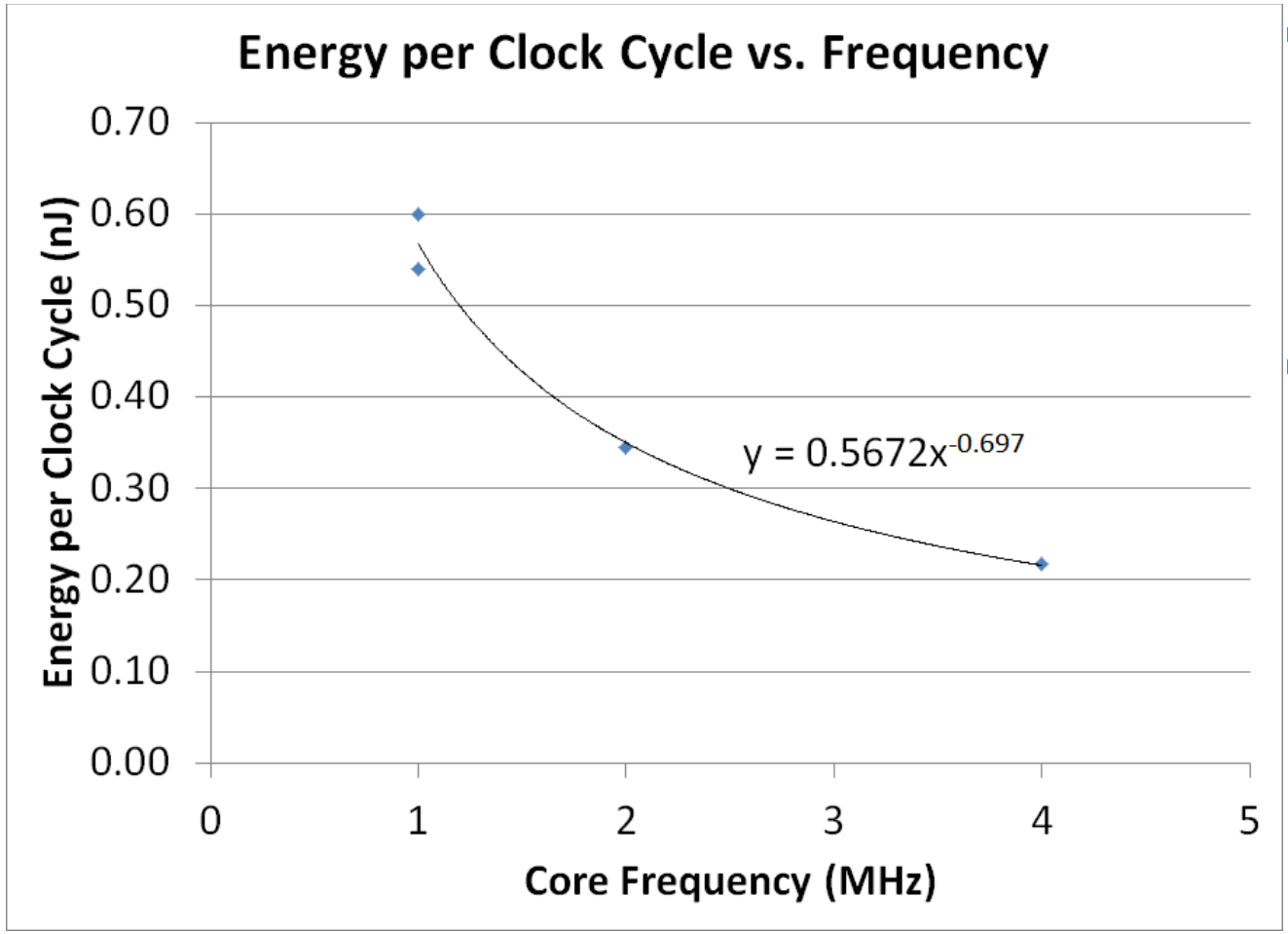
What is the Impact of Clock Speed on Power?



- 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

ARM

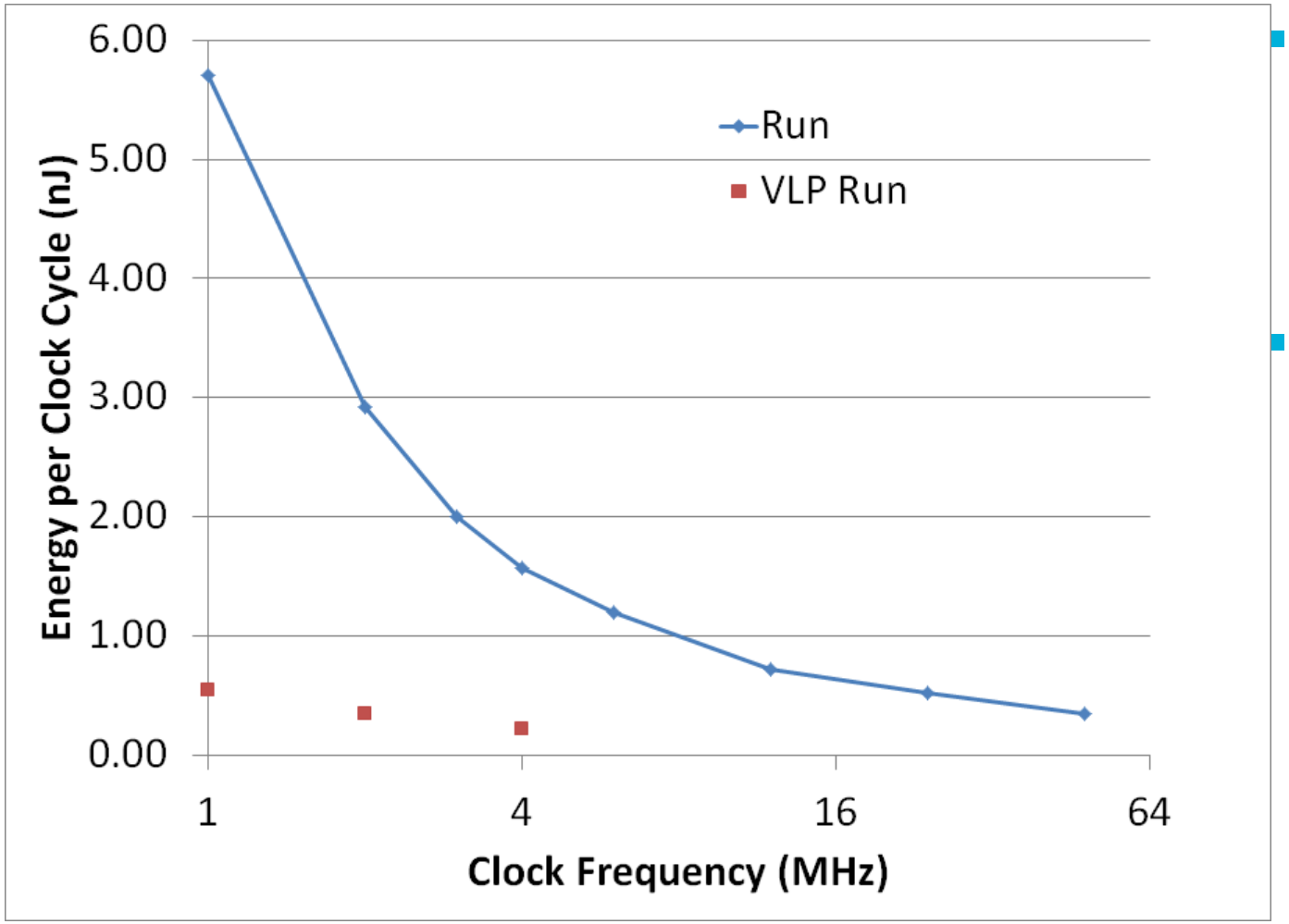
What is the Impact of Clock Speed on Energy?



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

ARM

Comparing Energy for Run and VLP Run



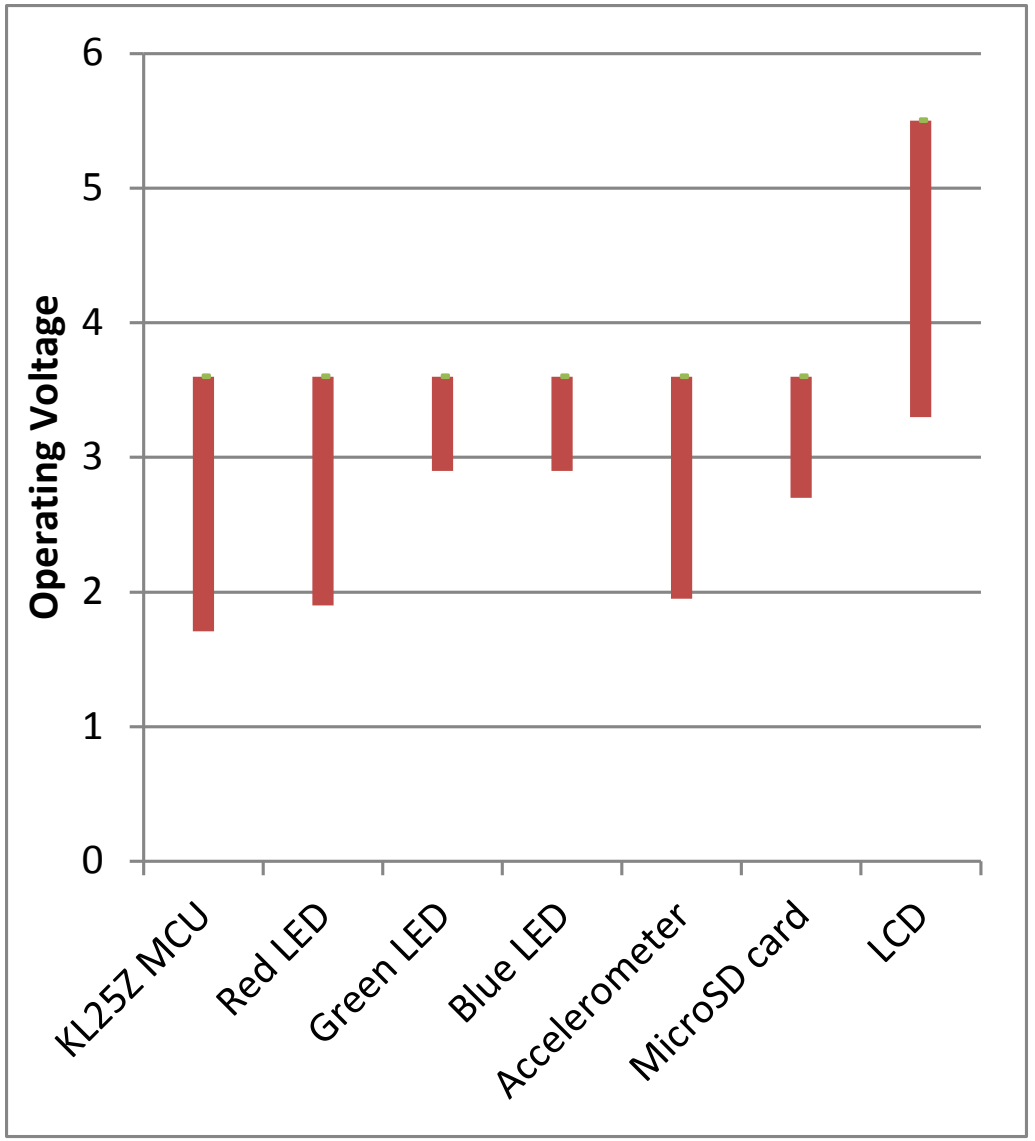
- Very low power also gives very low energy!
- 4 MHZ VLP Run mode more efficient than 48 MHz Run mode

ARM

Evaluating Voltage Scaling for the KL25

How about Lowering the Operating Voltage?

- P α V^2 so this will have a **BIG** impact
- How low can we go? What stops working as we lower the supply voltage?
  - LCD will stop first
  - LEDs will become dim unless we reduce the series resistor
  - Eventually LEDs will not light at all
- MCU
  - Is 48 MHz at 1.71 V possible?
  - Impact would be significant
  - Reduce power and energy by a factor of $3^2/1.71^2 = 3.08$



ARM

Does the Power Fall Quadratically with Voltage?

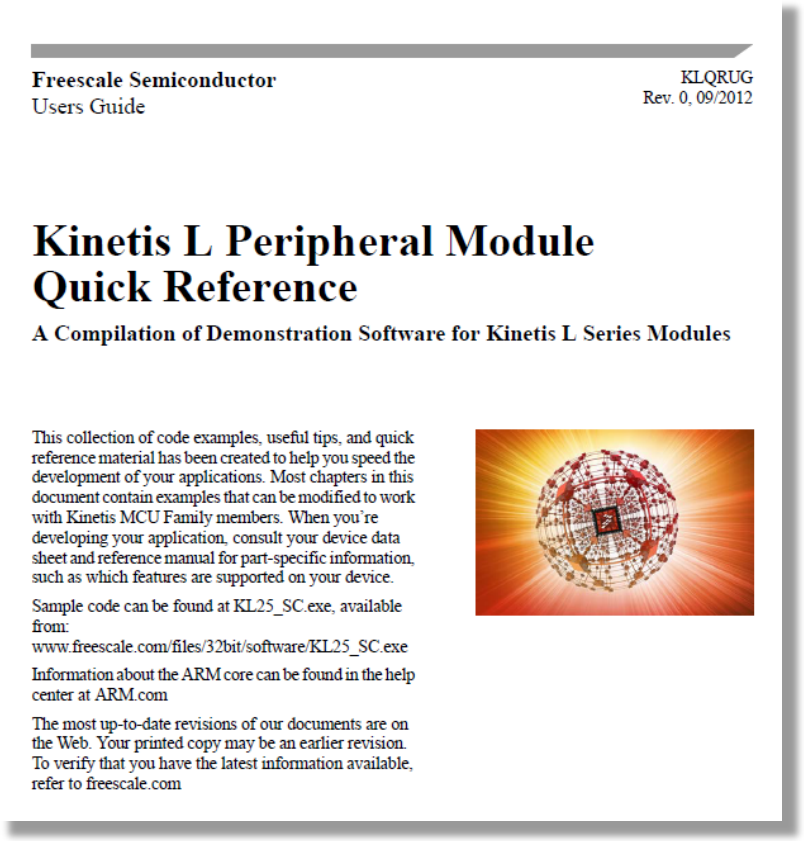
- Measure current and voltage (5V0 rail) with VMin Demo
- Be sure to keep debugger in reset (short JP15 pins 1 & 2)

5V0 Rail Voltage	Start Current	Start Power	All Done Current	All Done Power
5				
4				
3				
2				
(minimum)				

Power Management Strategies

Application Notes

- AN4503
- KLQRUG



Freescale Semiconductor
Users Guide

KLQRUG
Rev. 0, 09/2012

Kinetis L Peripheral Module Quick Reference

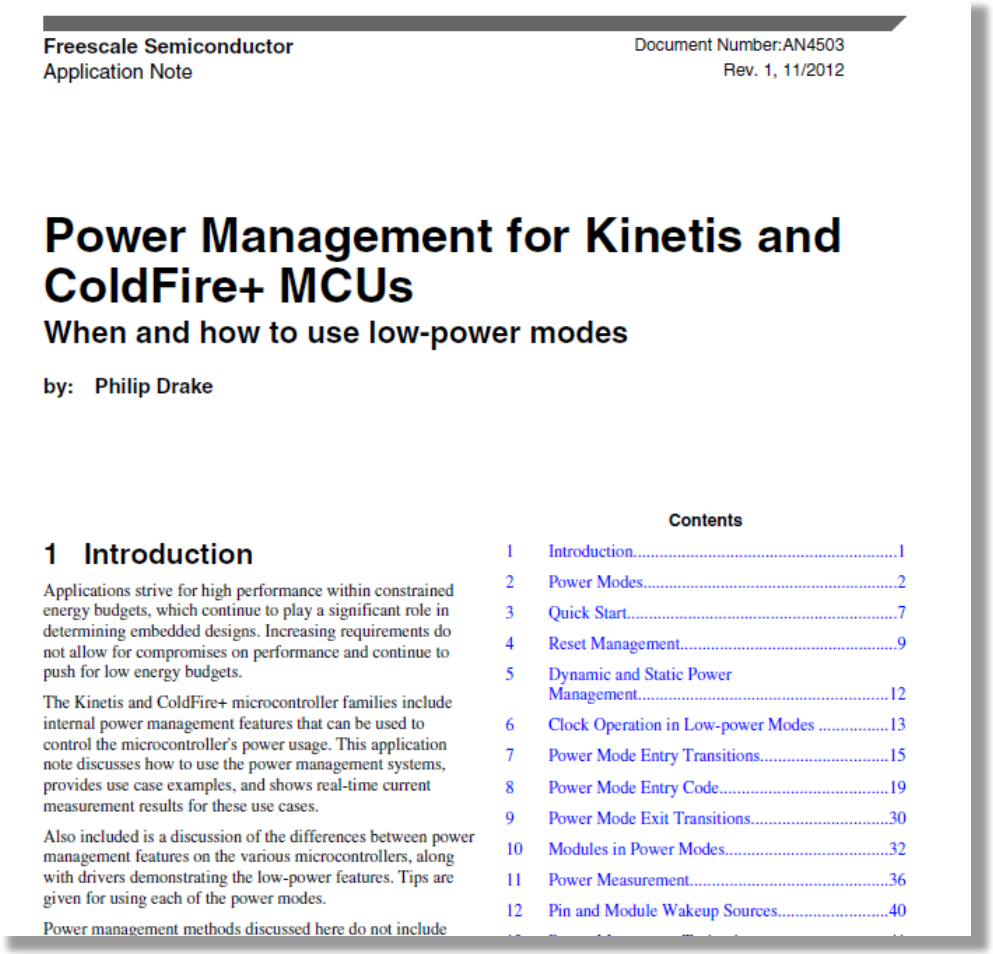
A Compilation of Demonstration Software for Kinetis L Series Modules

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:
www.freescale.com/files/32bit/software/KL25_SC.exe

Information about the ARM core can be found in the help center at ARM.com

The most up-to-date revisions of our documents are on the Web. Your printed copy may be an earlier revision. To verify that you have the latest information available, refer to freescale.com



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.

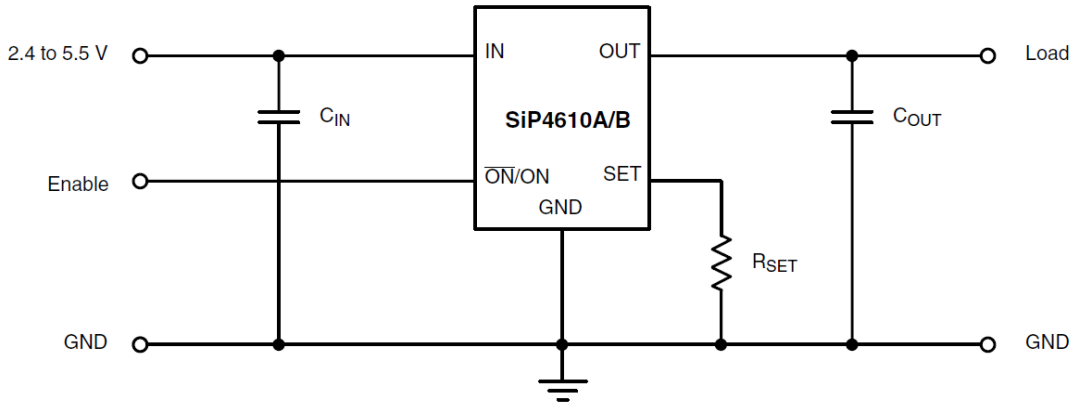
Also included is a discussion of the differences between power management features on the various microcontrollers, along with drivers demonstrating the low-power features. Tips are given for using each of the power modes.

Power management methods discussed here do not include

Contents

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 have accept command over serial bus (SPI, I2C)
    - Accelerometer MMA8451Q



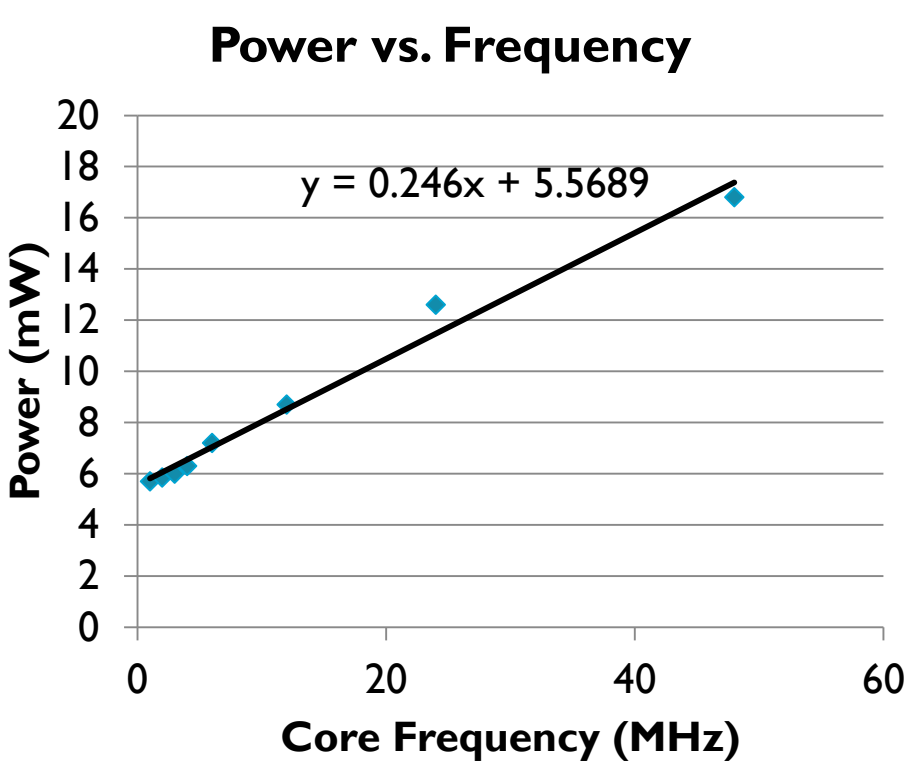
- Otherwise have to switch off power
  - Low-current devices (<10 mA)
    - Supply power with MCU's digital output pin(s)
  - Higher-current devices
    - Switch power with a transistor or high-side supply switch driven by an MCU digital output
    - Example: Vishay SiP4610 Protected 1 Amp High-Side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

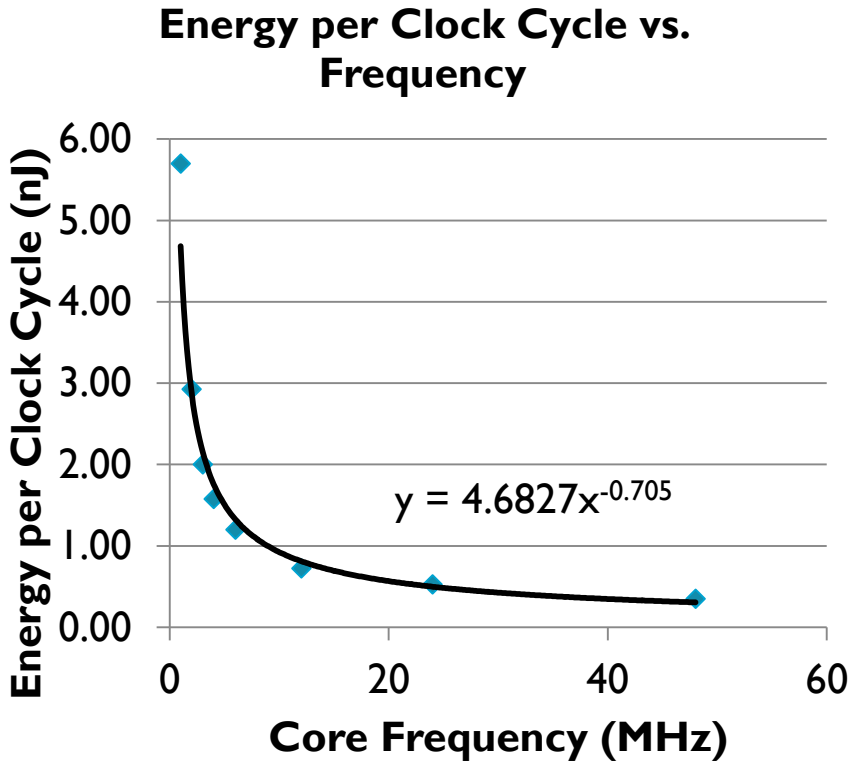
ARM

Power Management Options for MCUs

- Clock scaling
  - Slow down the CPU clock if there's not much 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

Clock Scaling – Slow Down



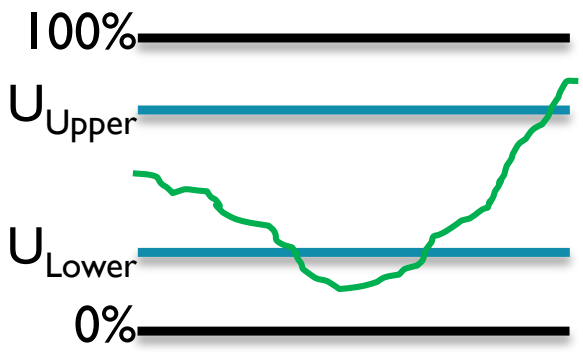


- Reducing clock rate cuts dynamic power used
  - Eventually reaches limit of about 6 mW - static (leakage) power
- Fewer cycles spent waiting for the next interrupt or timer event
- Actual energy cost per cycle increases

ARM

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

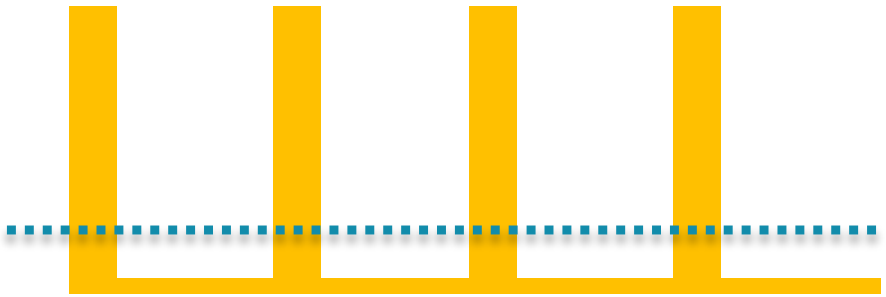


- Real-time systems have deadlines
  - Goal: minimize energy AND meet all deadlines
  - Can then set speed for each task *statically*…
    - Pick the frequency (hence voltage) per task which minimizes the energy based on the task's ***worst-case execution time*** and ***deadline***.
    - One fixed clock 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

ARM

Sleep While Idle?

- Factors
  - C 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

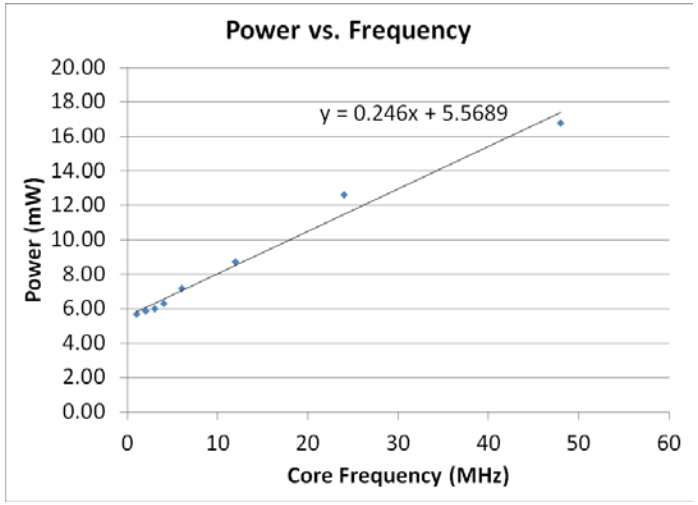


- Times and Duty Cycles
  - $T_{Active} = C/f + T_{const}$
  - $D_{Active} = T_{Active}/1$ sec
  - $T_{Sleep} = 1 - Tactive$
  - $D_{Sleep} = T_{Sleep}/1$ sec

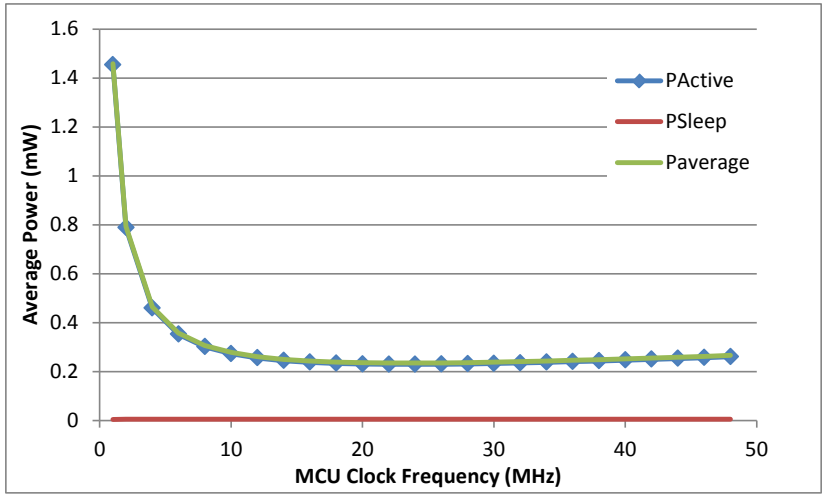
- Average MCU Power
  - $P_{MCU} = T_{Active} * P_{Active} + T_{Sleep} * P_{Sleep}$
  - $P_{MCU} = P_{Active} * (C/f + T_{const}) + P_{Sleep} * (1 - C/f - T_{const})$

Use Both Sleep and Clock Scaling

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



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



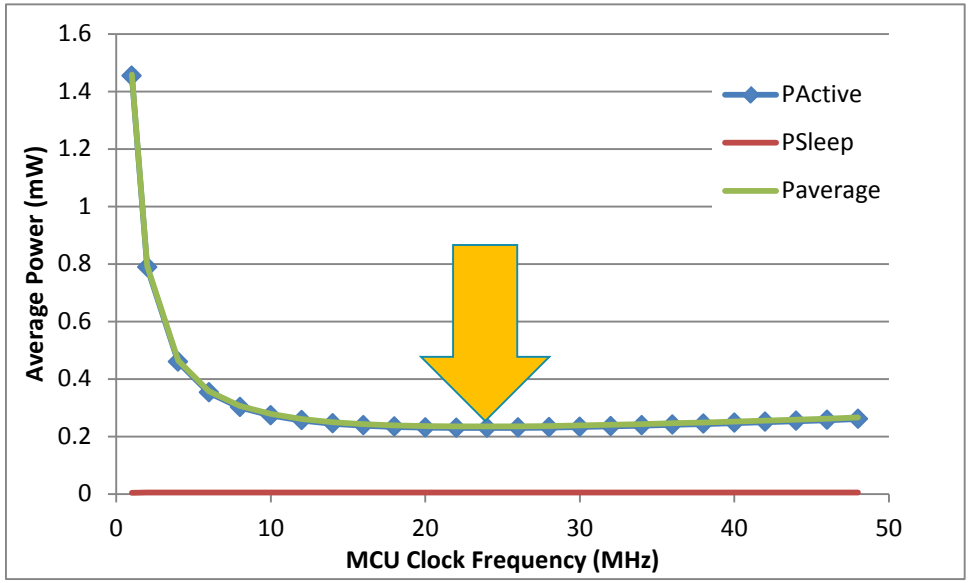
- Combine equations
  - $P_{MCU} = (P_{Active,Dynamic}*f + P_{Active,Static})(C/f + T_{const}) + P_{Sleep}*(1 - C/f - T_{const})$

ARM

What is the Optimal Frequency?

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

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

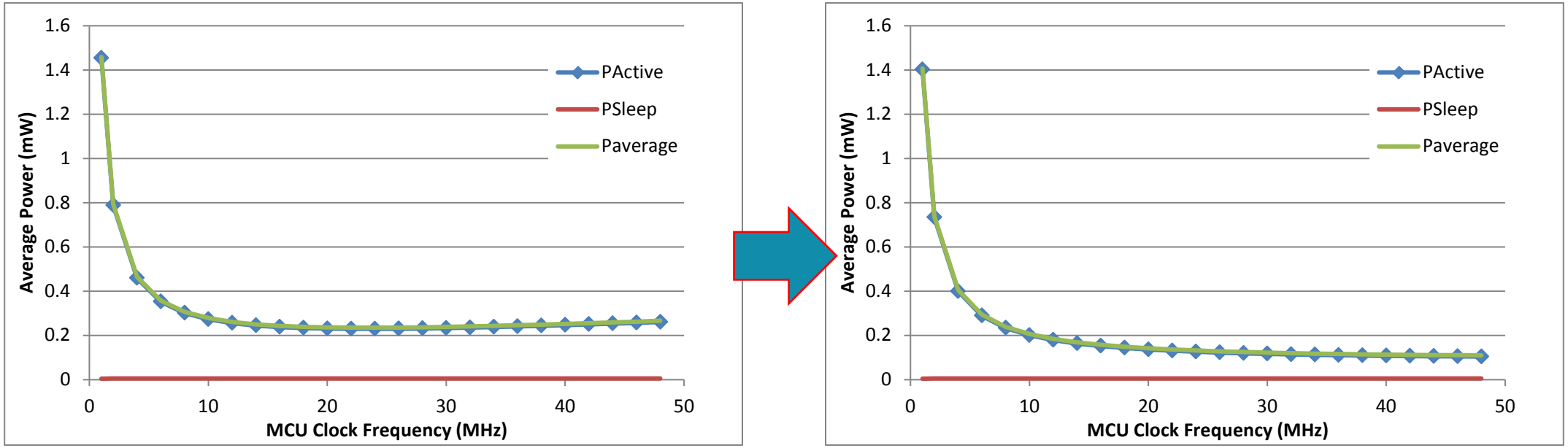


- Optimal frequency rises with
  - More computation
  - Increasing difference between active static power and sleep power
- Optimal frequency falls with
  - Higher active dynamic power
  - Longer wakeup overhead

ARM

Factors Affecting Optimal Frequency

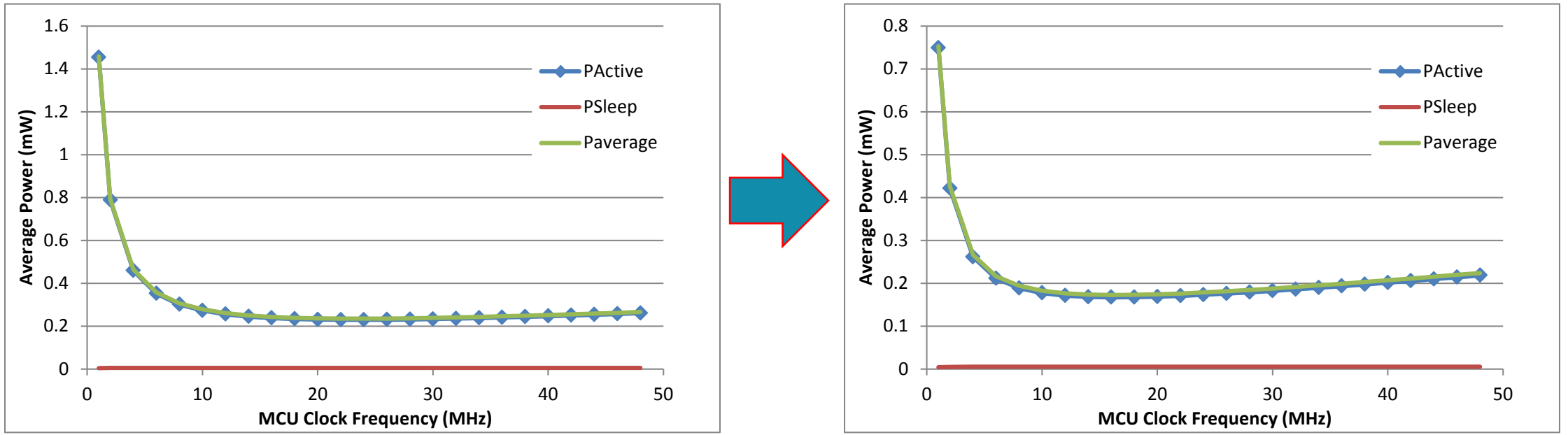
- Less wake-up overhead? ($T_{const}\downarrow$)
  - Higher frequency is better
  - Dropping T_{const} from 10 ms to 5 ms raises f_{opt} from 24 MHz to 32 MHz



ARM

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



ARM

Factors Affecting Optimal Frequency

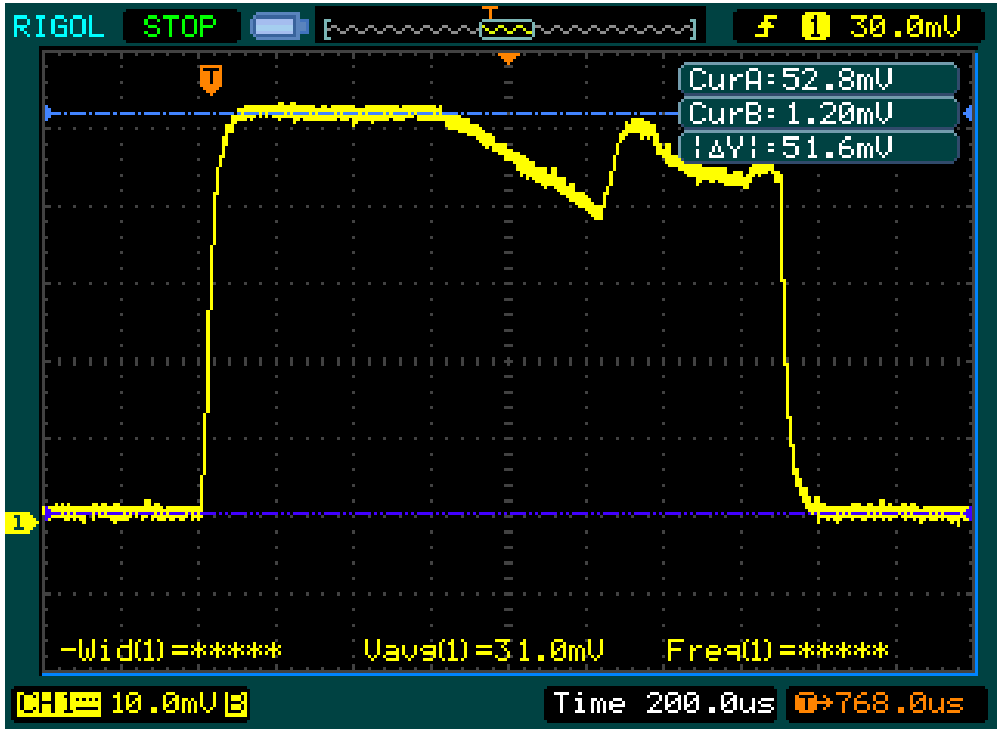
- Less dynamic active power dependence on frequency? ($P_{Active,Dynamic}$ ↓)
  - Higher frequency is better
  - Dropping $P_{Active,Dynamic}$ from 0.246 mW/MHz to 0.123 mW/MHz changes f_{opt} from 24 MHz to 32 MHz

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

Power over Time Analysis

Power Profiling

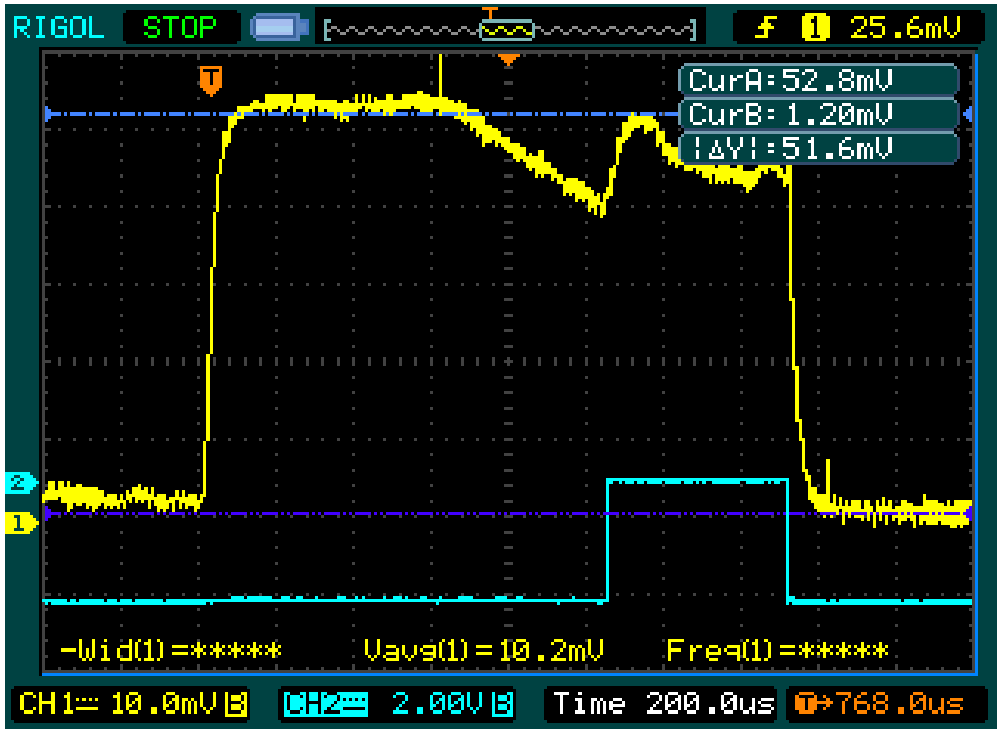
- How much power does MCU take when waking up, doing 0.4 ms of computation and then sleeping?
  - Measure current to MCU via J4, power is proportional to current here (fixed voltage)
- MCU draws large current for 1.5 ms
  - Expected only 0.4 ms
  - Why so much longer?
- Let's find out – add an output bit to indicate when processor is awake (1), when it is asleep (0)
  - Set to 1 on reset, entry to ISR, immediately after WFI instruction
  - Clear to 0 immediately before WFI



ARM

The Plot Thickens

- What is happening before our ISR starts running?
- It is taking a very long time (about 1 ms) compared to the ISR duration, leading to a lot of extra power consumption
- How can we wake up faster?
  - *Extra credit for the solution!*



ARM

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)

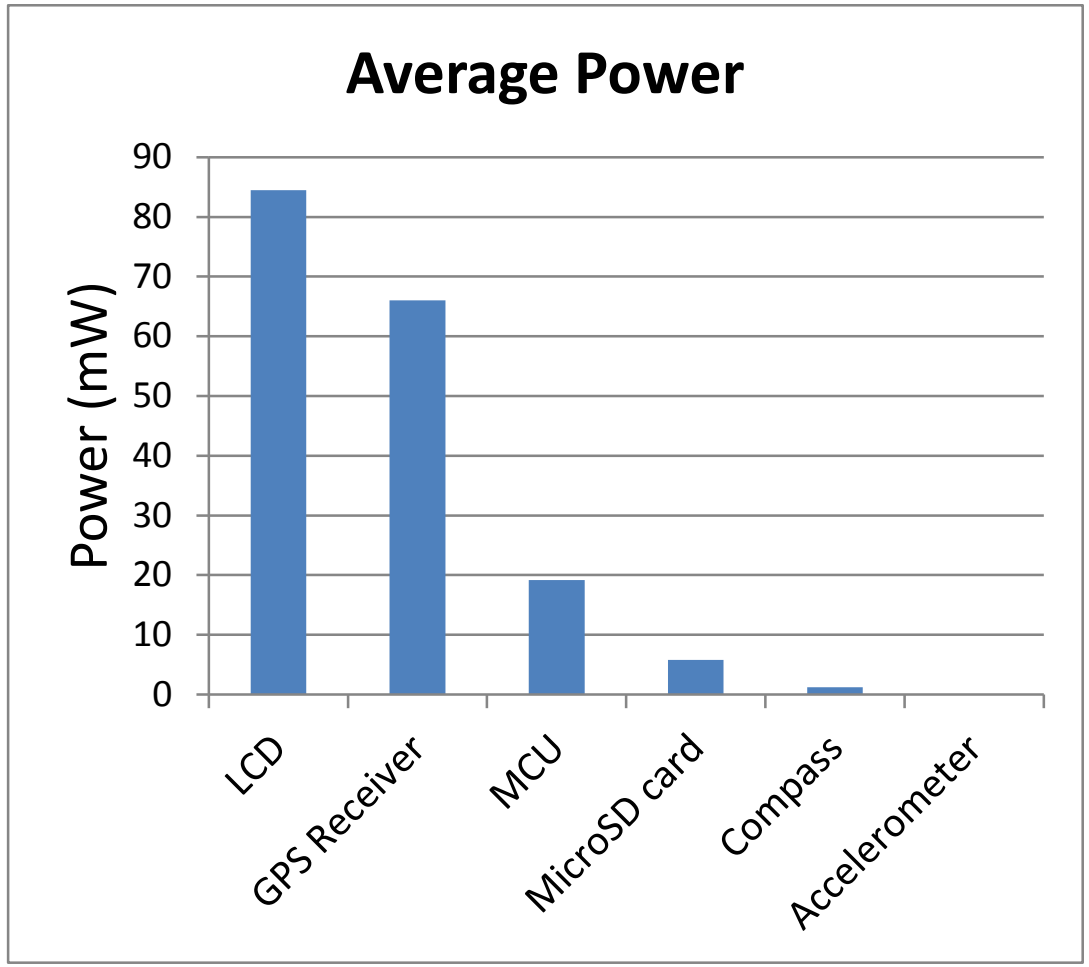
ARM

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



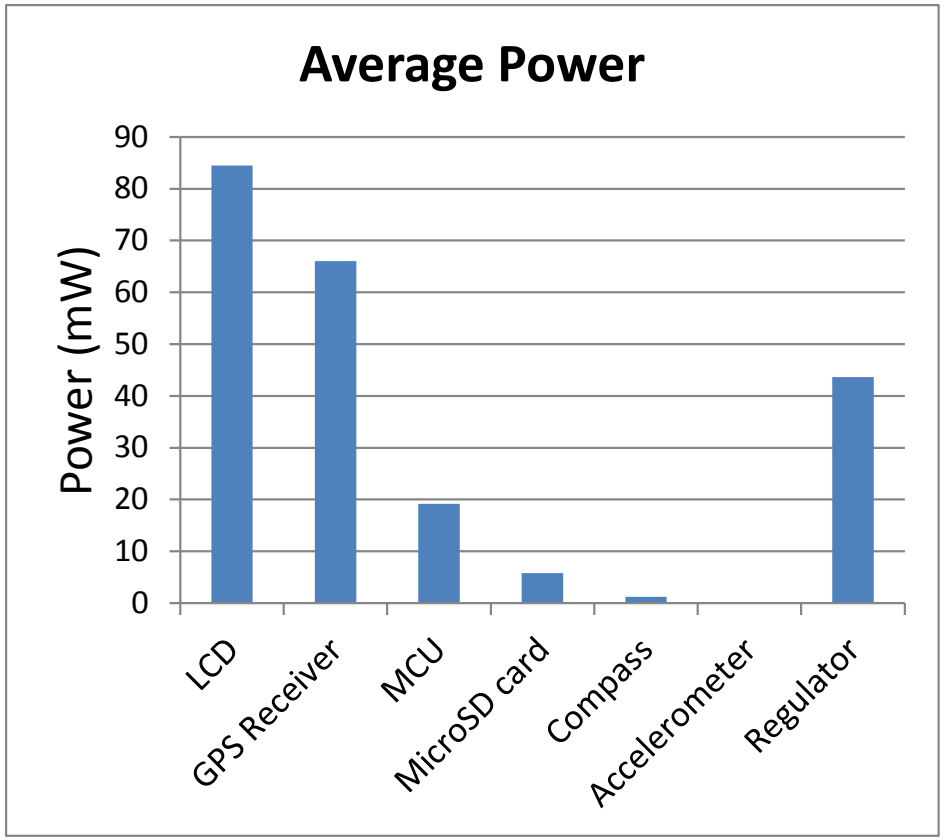
ARM

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)



ARM

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

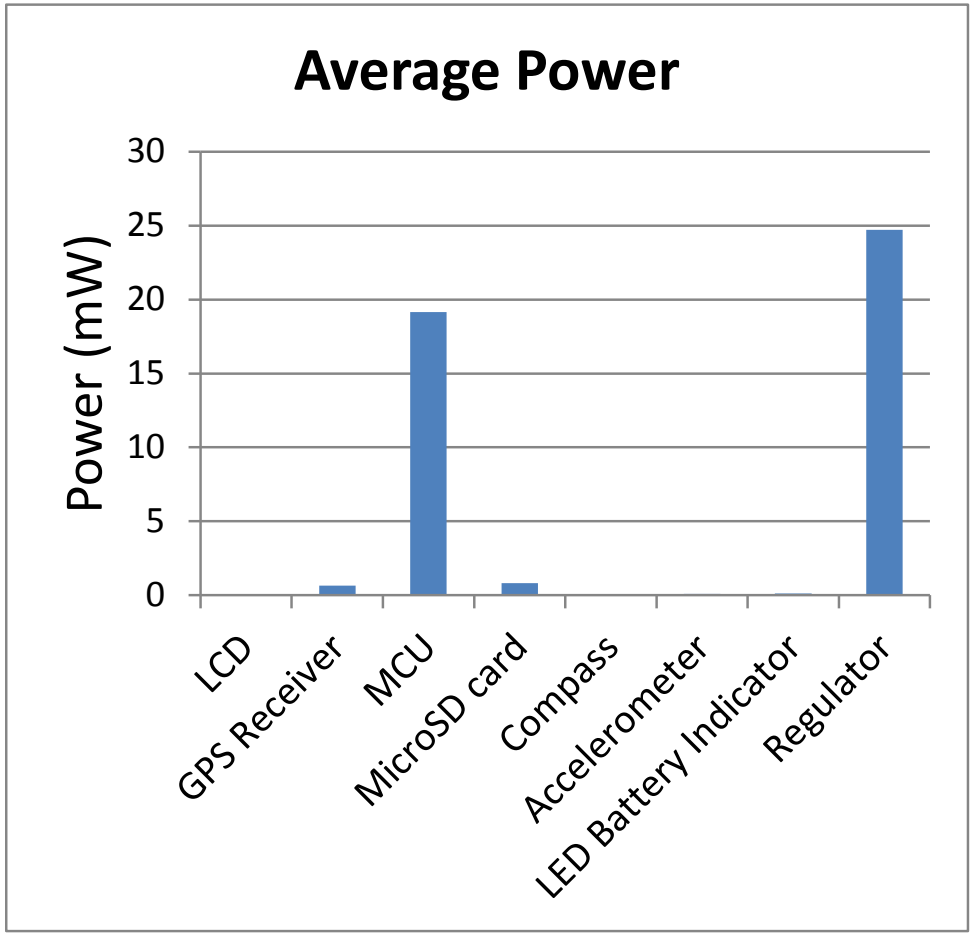
	Standby	
Device	Current (mA)	Power (mW)
LCD	0.000	0.000
GPS Receiver	0.200	0.660
MCU	n/a	n/a
MicroSD card	0.250	0.825
Compass	0.002	0.007
Accelerometer	n/a	n/a
LED Battery Indicator	0	0.000

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

ARM

Standby – Linear Voltage Regulator

- Results
  - Average power is 45.5 mW, was 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)?

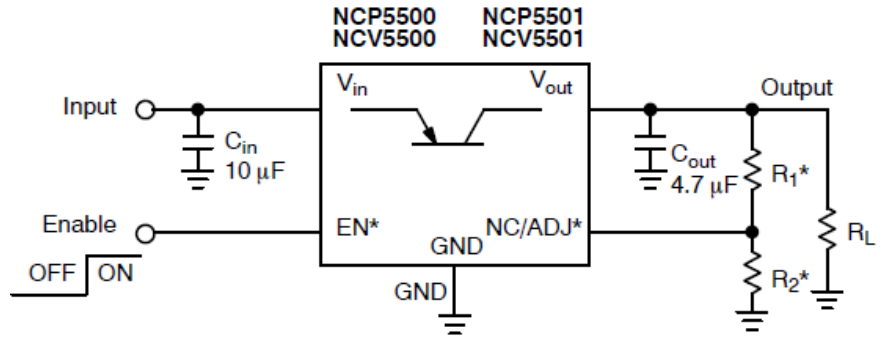
  - Can we even get rid of the regulator?
    - All components must be able to handle maximum battery voltage (4.3 V for Li)

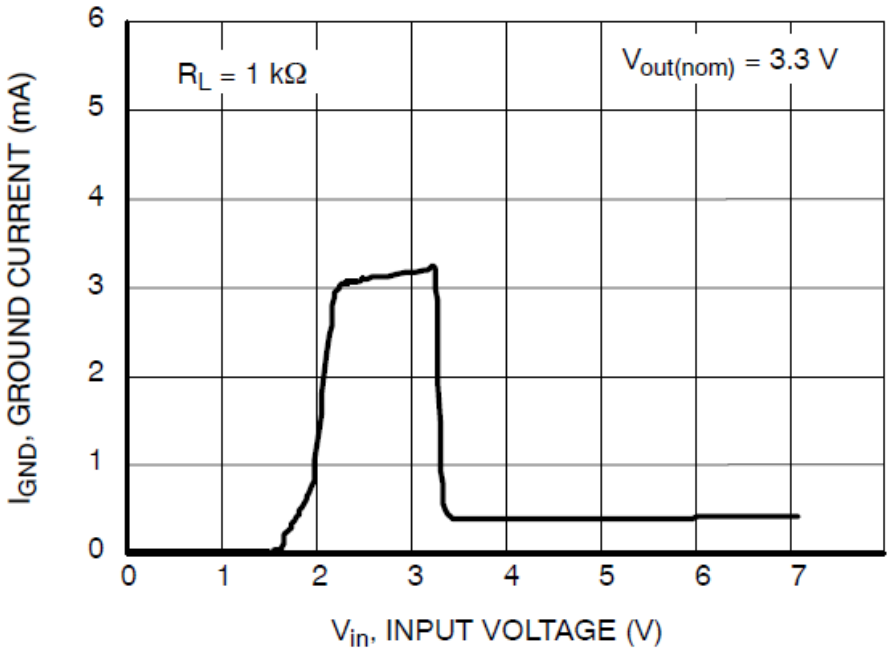
ARM

Improving the Linear Voltage Regulator

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

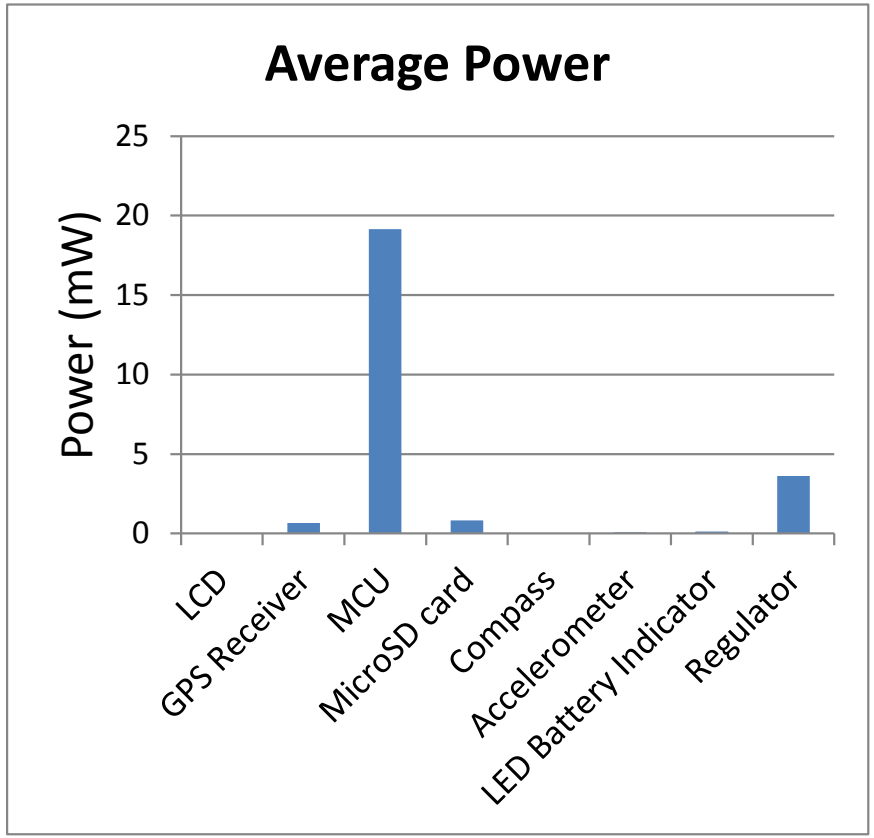
- Example: On Semiconductor NCP5500 500 mA LDO Voltage Regulator
  - Datasheet: NCP5500/D, April 2013, Rev. 12, http://onsemi.com
  - Low quiescent current: 300 uA
  - I_{out} up to 500 mA
  - What's going on between 2 and 3.3 V?
    - *Extra credit*





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



ARM

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

- 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

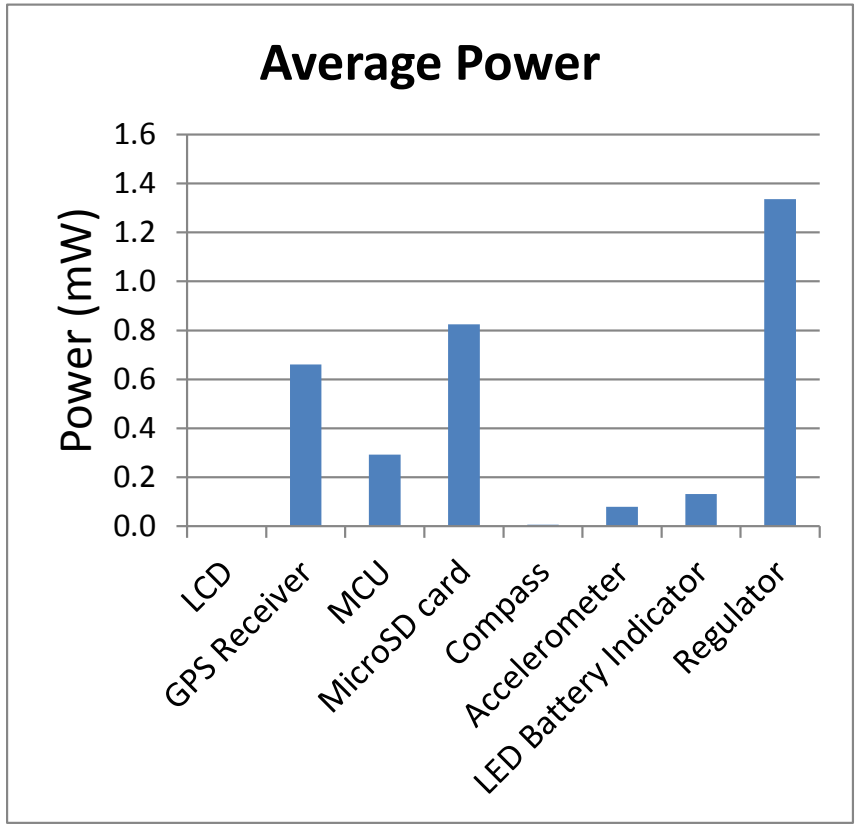
How Many Compute Cycles per Second?

- Derive rough estimate based on similar system (energy measurement lab)
  - Wakes up at 10 Hz
  - Measures voltage with ADC, does floating point computation
  - Duty cycle is about 500 uS/100 ms = 0.5%
- Consider major differences – use accelerometer to determine whether to wake up
  - Polling? Need to consider duration of I^2C Communication with accelerometer at 400 kHz
  - Interrupt-based triggering? Don't need to consider additional computation - use this approach
- 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

Sleep While Idle?

- Factors
  - 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

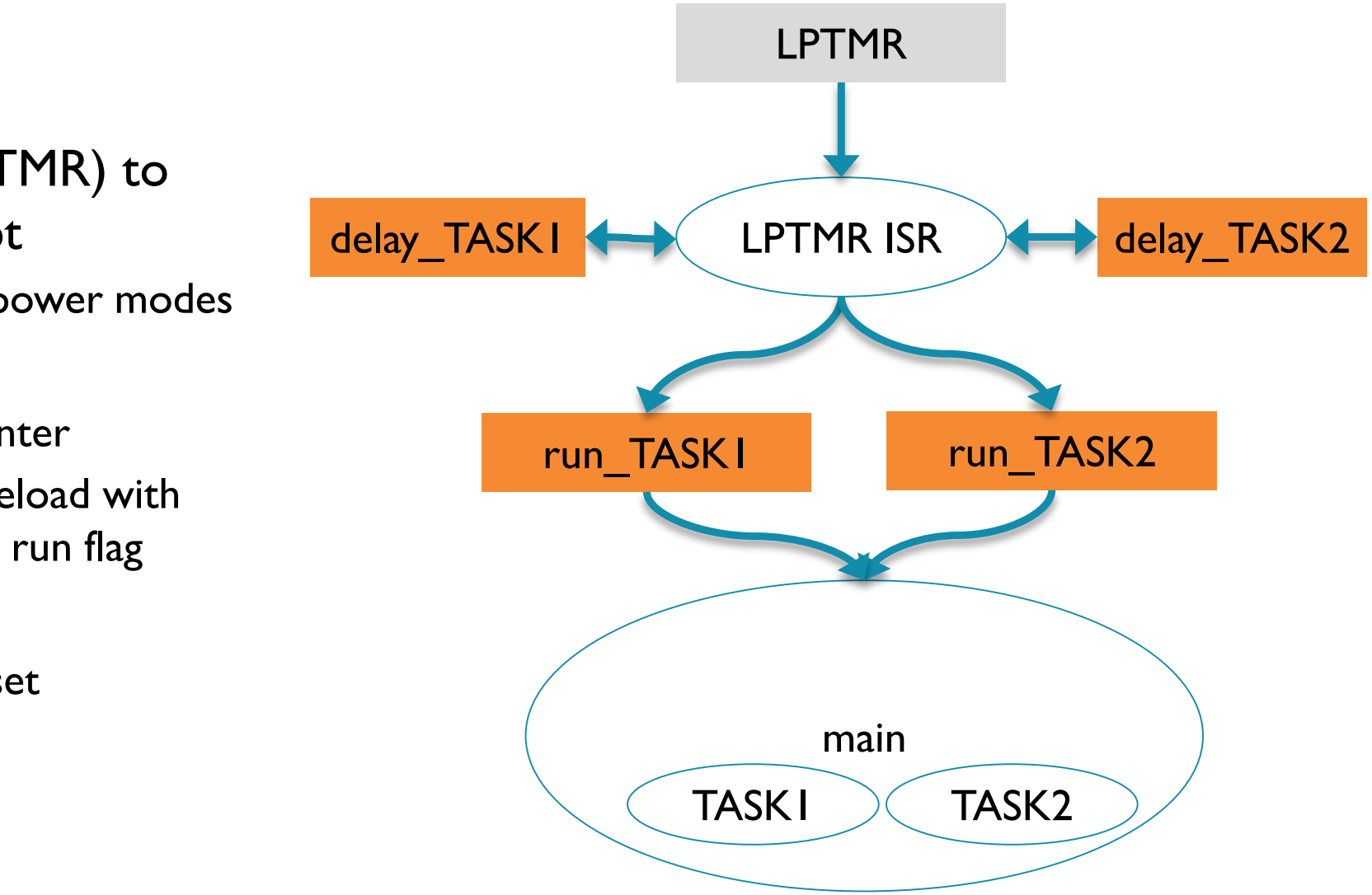


ARM

Power Management and Software Task Schedulers

No Scheduler?

- Use low-power timer (LPTMR) to generate periodic interrupt
  - LPTMR operates in all low-power modes
- LPTMR ISR
  - Decrements each delay counter
  - If delay counter reaches 0, reload with original value and sets task's run flag
- Main loop
  - Executes code if run flag is set
  - Goes back to sleep



ARM

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
}
```

Scheduler Support for Low-Power Modes

- Modify scheduler to put processor into stop or halt mode when no tasks are ready to run

- "Tickless Idle"
  - Schedulers often use a timer tick to update scheduling tables
  - This timer tick will wake up processor from low power mode
  - If next task won't need to run until N ticks from now, disable the timer for the next N timer periods

```
void Run_RTC_Scheduler(void)
{
  …
  /* Loop 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
    __halt(); // or __stop();
  }
}
```

Summary

- What can **you** do to minimize power consumption?

- Circuit design
  - Choose power-efficient parts
    - Operate at a low voltage
    - Operate at a low frequency if dynamic power dominates
    - Turn off processor and other circuits if static power dominates
  - Use low-power modes or shut off parts
- Power supply
  - Use an efficient power supply
  - ***Maybe*** 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 processor
  - Possibly use external hardware to reduce CPU on-time