Dynamic Performance Scaling on FreeRTOS

EEM202B Course Project

Gautham Kumar Reddy and Vishesh Dokania
Advisor: Prof. Mani Srivastava

Winter 2017

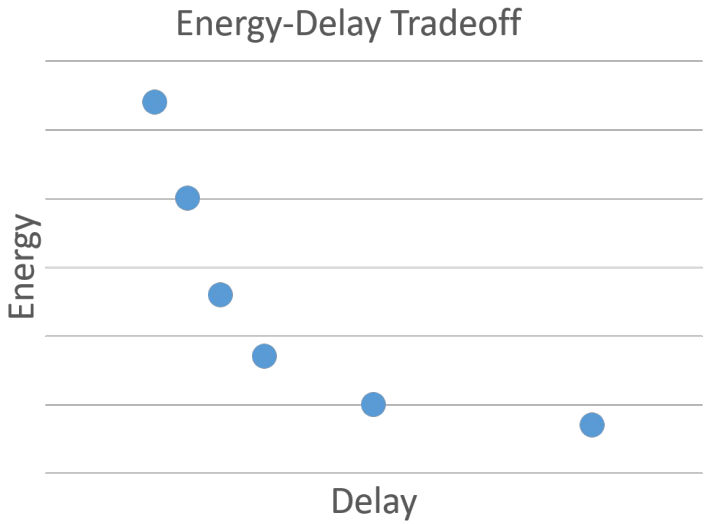
Motivation: DVFS for Power Saving

- Both dynamic + static power depend on supply voltage, operating frequency
- VFS → Impact on system performance

$$P(V,f) = P_{sw} + P_{leak} = k_1 V^2 f + k_2 V e^{k_3 V}$$

$$t = IC \times CPI \times 1/f$$

- Dynamic <volt, freq> tuple at runtime → leverage available slack in performance targets
- Overheads
  - Lower performance - choose right setting to meet task deadlines
  - Voltage/frequency transition latency
  - Decision-making overhead at the kernel/governor level.



Motivation: FreeRTOS

- Open-source, compact and efficient kernel code base (3 C files)
- Available ports for 35+ common microcontrollers
- Offers most useful kernel features
  - Flexible task priorities and notification
  - Queues, semaphores, mutexes
  - Software timers, interrupt nesting, tick and idle hooks.

Scope for Improvement

- Limited available power management schemes
  - Idle task mode - application can switch to low power state
  - Tickless idle mode (on supported architectures)
- No existing API support for DPM or DVFS
  - No implemented governor policies

Board Specifications

STM32L152C Discovery based on STM32I152RCT6

Core: ARM® Cortex®-M3 32-bit CPU

Supply: 1.65 to 3.6V

Frequency: 32 kHz up to 32 MHz

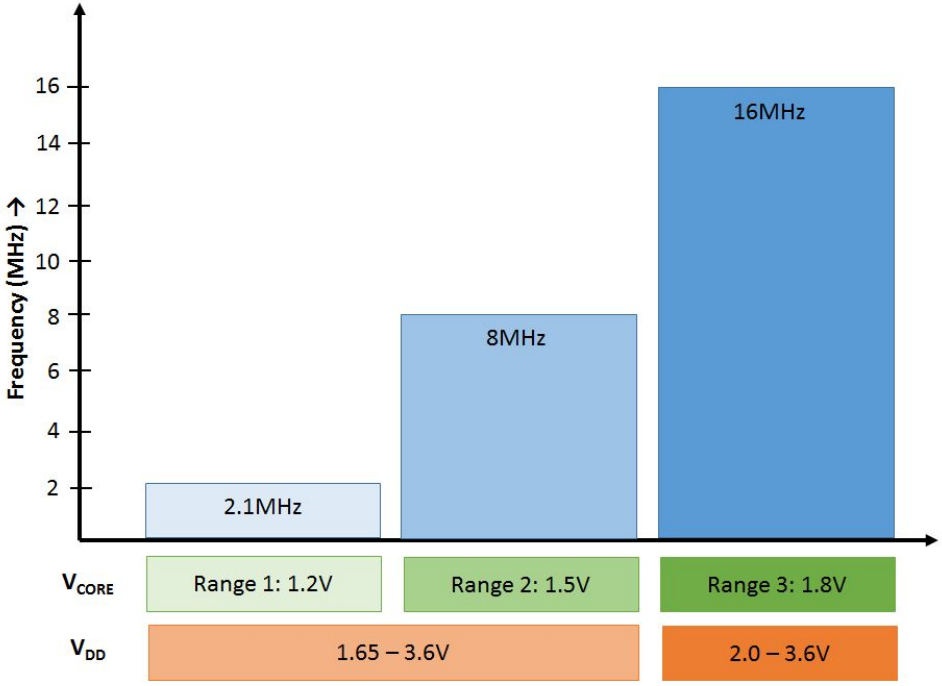
-40°C to 105°C temperature range

256 KB Flash memory with 32 KB RAM

Frequency and Voltage Scaling:

Configurable frequency with multiple sources and prescalers

3 voltage ranges



Implementation

- Getting familiar with FreeRTOS and STM32 architecture
- Isolating SysTick timer from CPU clock
- Generating Run Time Stats for CPU utilization
- Implementing Governor and different policies
- Power measurement and profiling for validation

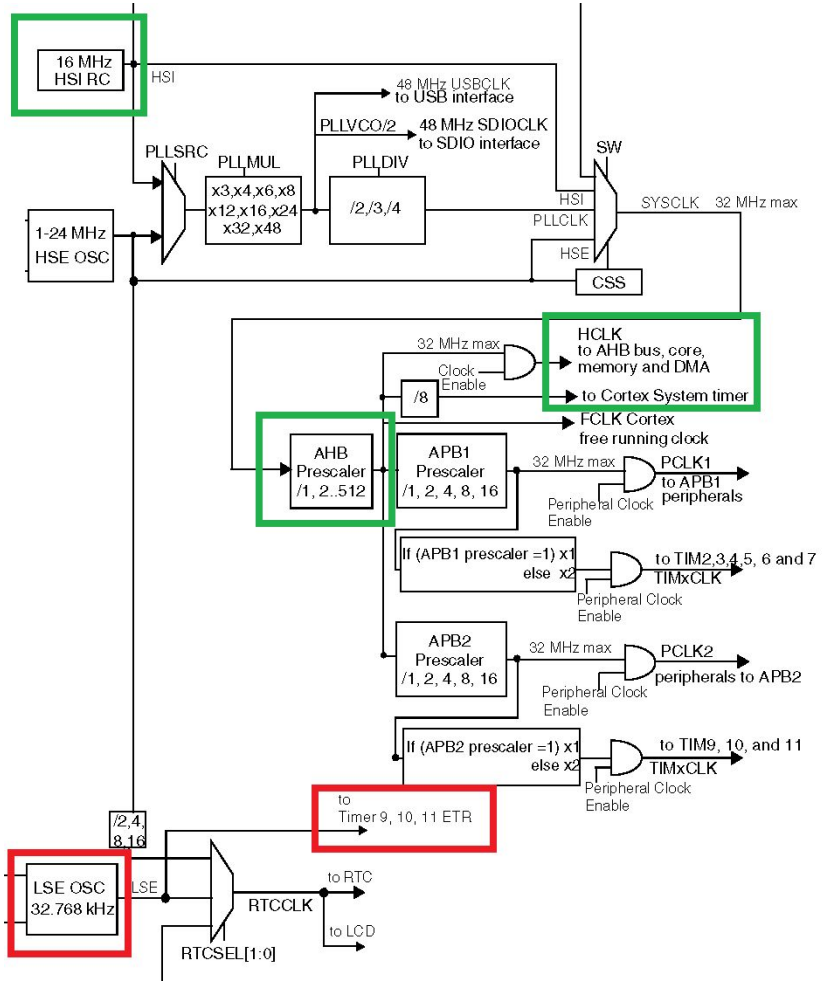
SysTick Timer

Issue :

- Default SysTick timer is derived from CPU frequency.
- SysTick interrupt rate changes with CPU frequency.

Solution :

- Derive SysTick timer from CPU frequency independent source
- TIMER 11 in External mode.
- Clocked from LSE oscillator running at 32.768 KHz
- Modify SysTick source in FreeRTOS.
- Update OS tick on timer interrupt.



Run Time Stats Generation

- Keep track of CPU run time using a Hardware Timer
- 16 bit Timer 11 .
- Configure in External mode
- 32 bit CPU run time.
- Lower two bytes =>counter value
- Higher two bytes =>overflow value
- Update task run time by subtracting CPU run times between task context switched in and switched out
- IDLE task in FreeRTOS

CPU utilization = (PERIOD - IDLE RUN TIME)/PERIOD

Static CPU Governors

PERFORMANCE :

Sets the CPU statically to the highest frequency within the range.

Voltage = 1.8 V

Frequency = 16MHz

POWERSAVE :

Sets the CPU statically to the lowest frequency within the range.

Voltage = 1.2 V

Frequency = 2 MHz

DVFS Power Governor Policies

- CPU utilization
- Window based governor

OnDemand: Halve or double frequency based on Usage Threshold.

```
{
        Usage = getCPUUsage();
        If (Usage > Des_Thr + δ)
                newFreq = currentFreq * 2;
        Else if (Usage < Des_Thr - δ)
                newFreq = currentFreq / 2;

        switchFreq (newFreq);
        AssignLowestVCore (newFreq);
}
```

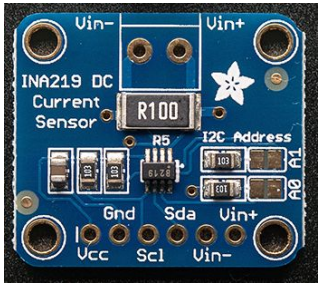
Frequency Oscillations

PAST_MixFreq: Converge to set CPU Usage Threshold. No oscillations.

```
{
        Usage = getCPUUsage();
        newFreq = currentFreq * Current_ExecTime/(Des_Thr *
UsageCheck_Period);
        {Freq1, Freq2} = ClosestAvailFreqs (newFreq);
        //Freq1>Freq2, together sandwich newFreq
        t = (newFreq * Des_Thr * UsageCheck_Period - Des_Thr *
UsageCheck_Period * Freq2) / (Freq1 -  Freq2);
        SetupTimer (TIMX, t);
        switchFreq (Freq1); AssignLowestVCore (Freq1);
}

TimerInterruptHandler (TIMX) {
        switchFreq (Freq2); AssignLowestVCore (Freq2);
}
```

Experimental Setup

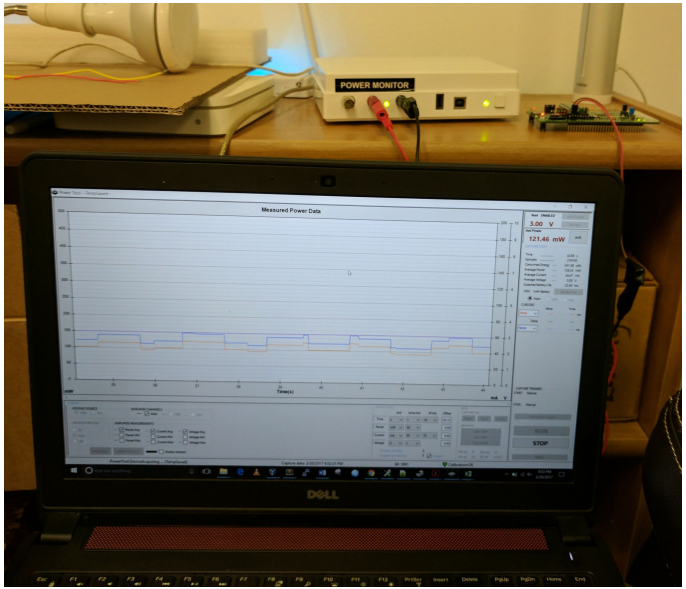


IAR Embedded Workbench IDE for Cortex-M

- Debugger, Live Watch for RunTimeStats

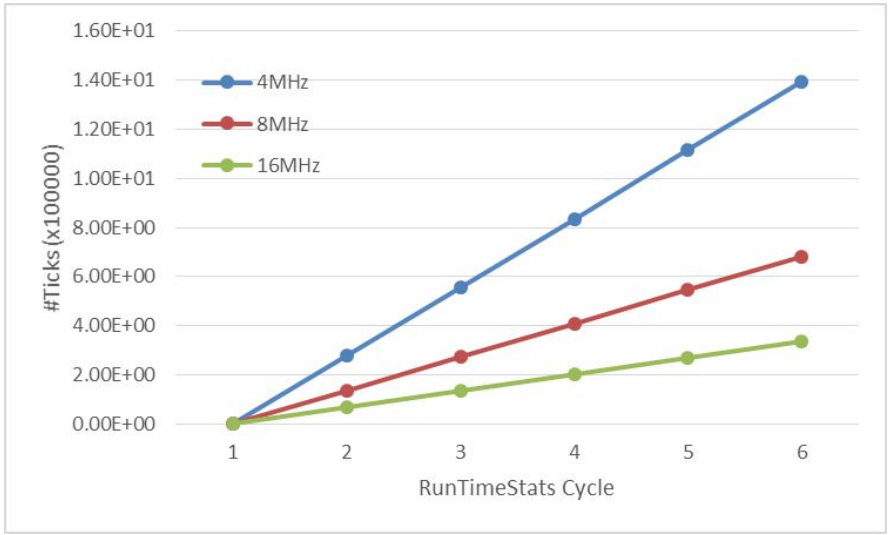
Power Measurement/Profiling

1. In-series Adafruit INA219 current sensing IC, I^2C comm. for sampling
   [M. Gottscho, "A Hardware/Software Framework for Enabling Battery Charging-Aware Systems Research", EEM202A Course Project, Winter '14]
2. Monsoon Power Monitor
   - Integrated power supply + USB sampling channel
   - PowerTool (Windows) for live control and sampling

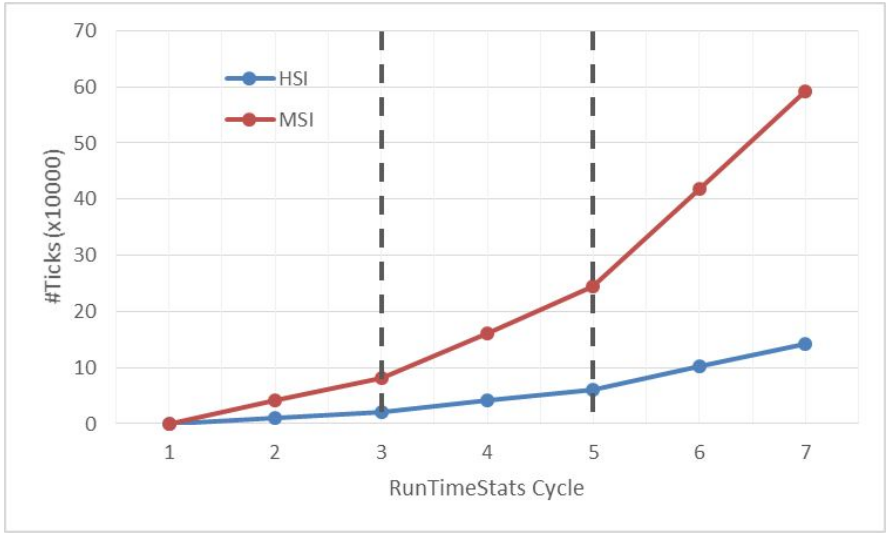


Frequency Scaling vs Task Runtime

Static Frequency Scaling



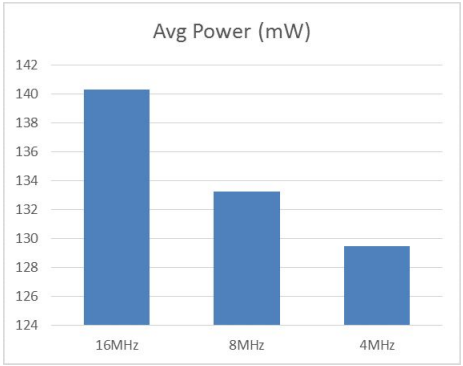
Dynamic Frequency Scaling

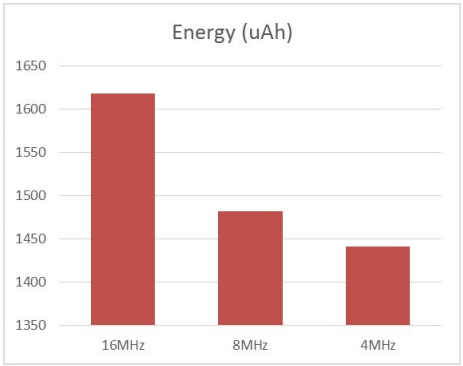


Frequency/Voltage Scaling vs Power/Energy

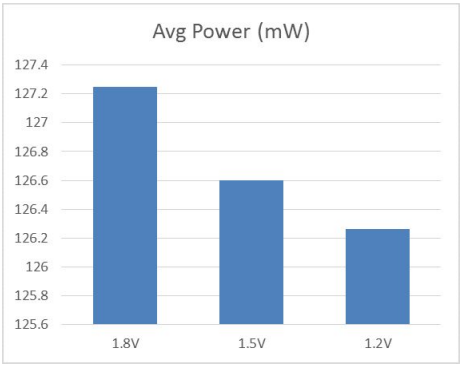
Frequency Scaling

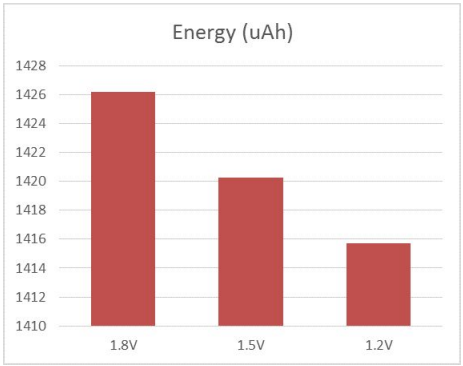
- Vcore = 1.8V, statically scaled frequency
- Accumulated energy over 120 seconds assuming 1000uAh battery
- Scaled frequency applies to core + most peripherals
- Significant power/energy benefits





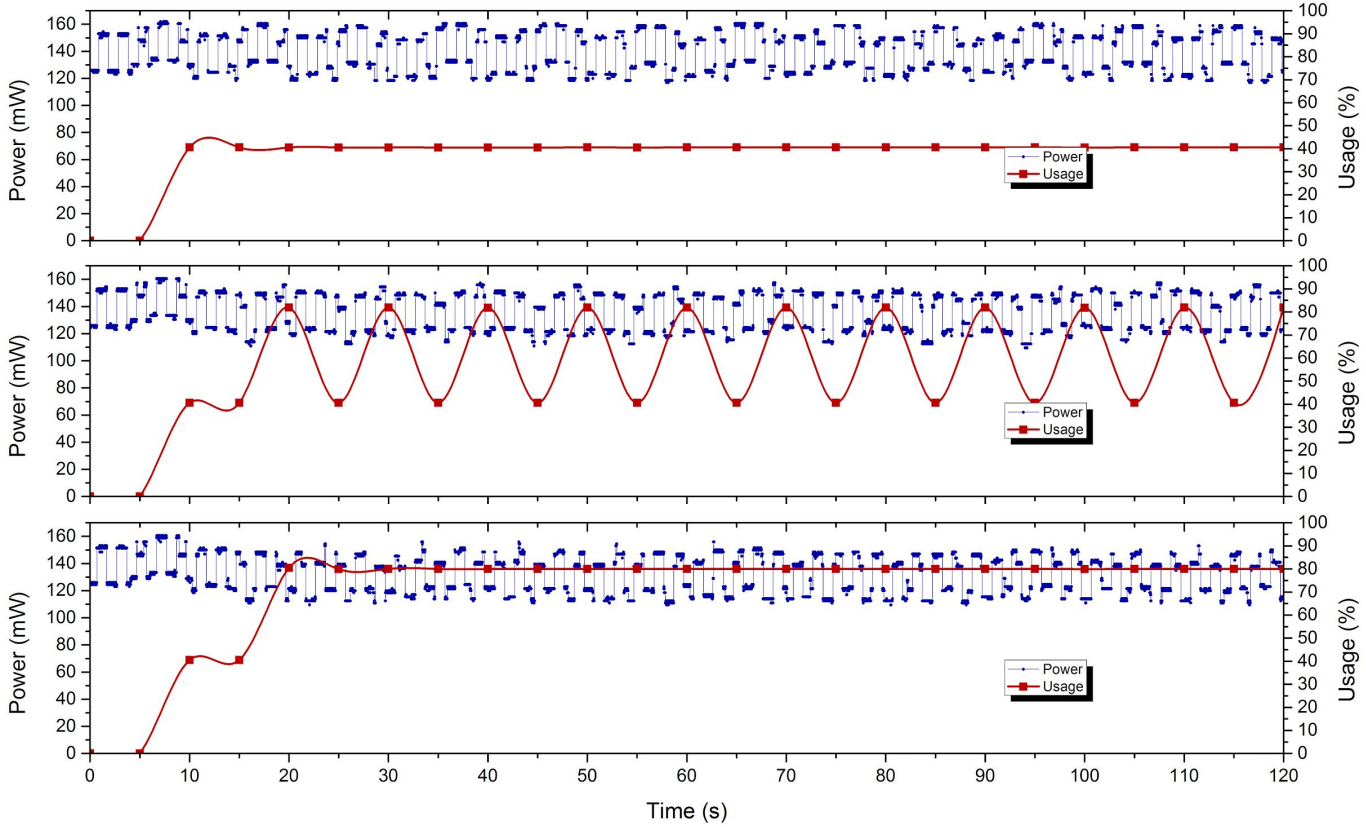
Core Voltage Scaling





- Freq = 1MHz to permit all Vcore ranges, statically scaled Vcore
- Accumulated energy over 120 seconds assuming 1000uAh battery
- Scaled Voltage only applies to the core, peripherals separately fed by constant VDD
- Limited power/energy benefits

Governor Policies - Power/CPU Usage Profiling

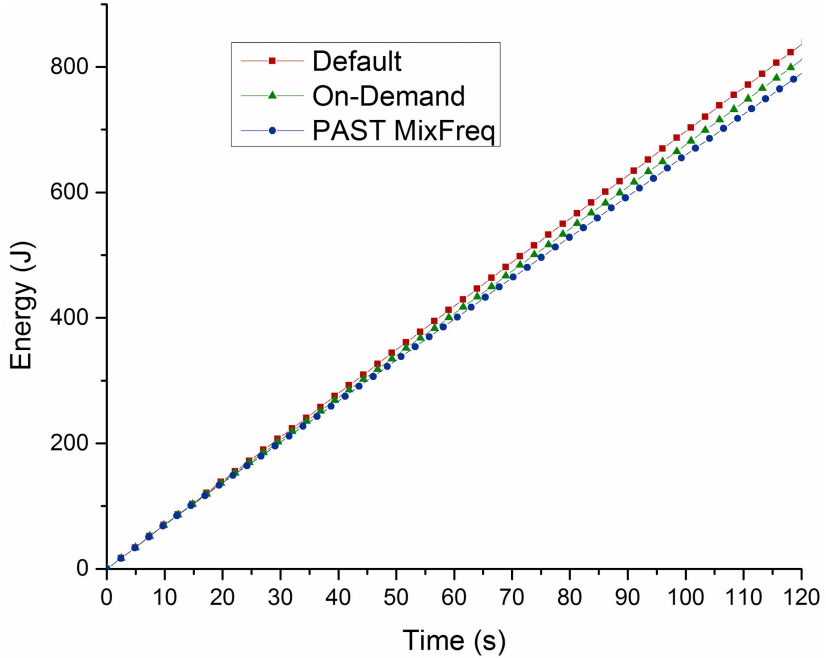


Default - No Governor
3 running tasks @ 16MHz
Default CPU usage
constant - 40%

On-Demand Governor
Oscillating CPU usage
Frequency flips between
16 and 8MHz

PAST MixFreq Governor
Configurable desired Usage
Converges within few cycles
Max. power savings

Governor Policies - Power/Energy Comparison



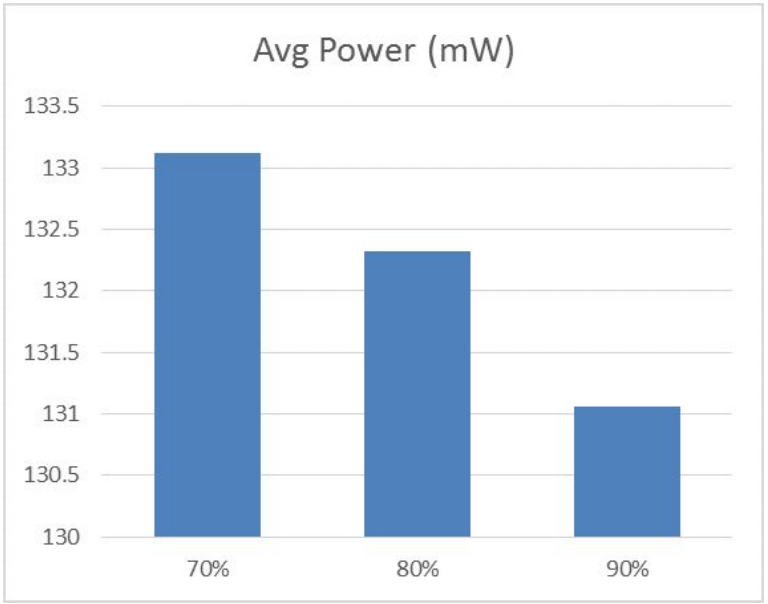
Governor	Default (None)	On-Demand	PAST MixFreq
Average Power (uW)	140.3	136.12	132.32

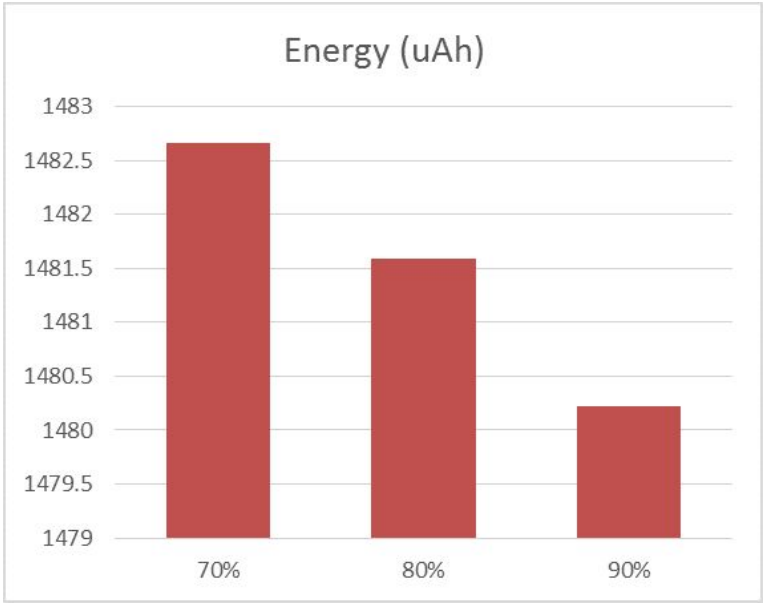
- **Total energy** consumed over 120 seconds from profiles shown earlier
- Governor period ~ Tasks Period

- **On-Demand Governor** sub-optimal
  - Cannot make full use of available gap in CPU usage
  - Can only double or halve frequency
  - Oscillating frequency adds overhead

- **PAST MixFreq Governor** offers significant benefit
  - Fine-tuned CPU usage
  - Can snap to any frequency combination
  - Only switches frequency where needed

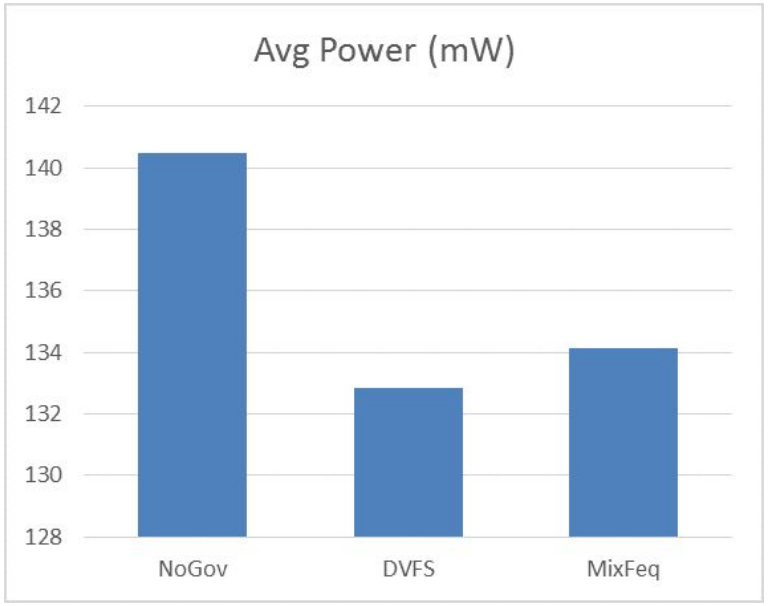
Past MixFreq Policy - Desired Usage Tuning

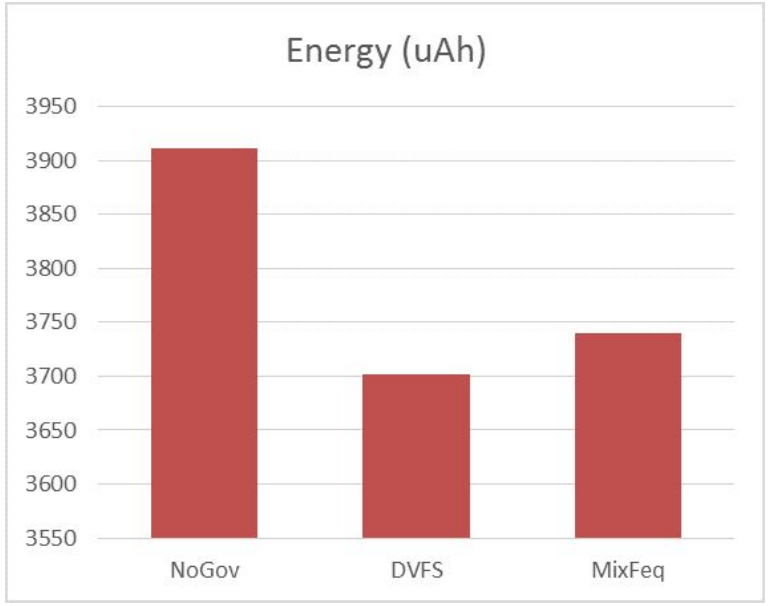




- **PAST MixFreq Governor** can be tuned to achieve **Desired Usage %**
  - Higher usage leverages available Idle period between task and deadline
  - Task computation can be performed at lower frequency and voltage

Policy Comparison - Small Usage Check Period





- **Small CPU Usage check period** - every second compared to task period of 15 seconds
- OnDemand has better characteristics as it can adapt based on CPU usage

Limitations

- Time critical tasks might miss deadlines
- Run Time Stats/CPU Usage as separate task
  - Can be a software timer task
  - FreeRTOS Limitation: Cannot be run in an interrupt context
- Performance related to governor period
- OnDemand mode can lead to CPU Usage oscillations
  - High frequency switching overhead
- Usage convergence difficult in PAST_MixFreq mode if governor period is small compared to task period
- Simple task schedule - identical arrival times and periods
  - Requires further exploration for aperiodic tasks
- ~12 OS ticks for Governor

Future Work

- Co-implementation with Dynamic Power Management policies
  - Multiple Sleep, Stop and Standby states available
  - Predict available Idle period, Sleep if >threshold
  - Manage peripheral power states for additional power benefits

- Simple learning algorithm to predict CPU usage
- Per Task slowdown
  - Identify task critical to deadline
  - Leverage idle times for tasks that do not affect overall schedulability
- Tune the governor window period according to the performance target.
- Vdd scaling

References

- http://www.freertos.org/low-power-tickless-rtos.html
- http://www.freertos.org/rtos-run-time-stats.html
- STM32 RM0038 Reference manual
- R. Barry, “Mastering the FreeRTOS™ Real Time Kernel”, 2016.
- M. Weiser et al., “Scheduling for reduced CPU energy”, USENIX Conf. on OSDI, 1994.
- M. Martonosi, S. Malik, and F. Xie, “”Efficient behavior-driven runtime dynamic voltage scaling policies”, IEEE/ACM/IFIP Int. Conf. CODES+ISSS, Sept. 2005.
- N.P.K. Gorti, “Application aware performance, power consumption, and reliability tradeoff”, PhD Thesis, Iowa State University, 2014.
- K. Bhatti, C. Belleudy, and M. Auguin, “Power Management in Real Time Embedded Systems through Online and Adaptive Interplay of DPM and DVFS Policies”, IEEE/IFIP Int. Conf. EUC, Dec. 2010.
- V. Spiliopoulos, S. Kaxiras, and G. Keramidas, “Green governors: A framework for Continuously Adaptive DVFS”, Int. Green Computing Conference and Workshops (IGCC), 2011.

Thank You!

Backup: Original PAST Policy

Speed Setting Algorithm (PAST)

`run_cycles` is the number of non-idle CPU cycles in the last interval.

`idle_cycles` is the idle CPU cycles, split between hard and soft idle time.

`excess_cycles` is the cycles left over from the previous interval because we ran too slow. All these cycles are measured in time units.

```
idle_cycles = hard_idle + soft_idle;
run_cycles += excess_cycles;
run_percent = run_cycles /
   (idle_cycles + run_cycles);

next_excess = run_cycles -
   speed * (run_cycles + soft_idle)
IF excess_cycles < 0. THEN
   excess_cycles = 0.
```

```
energy = (run_cycles - excess_cycles) *
   speed * speed;

IF excess_cycles > idle_cycles THEN
   newspeed = 1.0;
ELSEIF run_percent > 0.7 THEN
   newspeed = speed + 0.2;
ELSEIF run_percent < 0.5 THEN
   newspeed = speed -
      (0.6 - run_percent);

IF newspeed > 1.0 THEN
   newspeed = 1.0;
IF newspeed < min_speed THEN
   newspeed = min_speed;

speed = newspeed;
excess_cycles = next_excess;
```

[M. Weiser et al., “Scheduling for reduced CPU energy”, USENIX Conf. on OSDI, 1994]