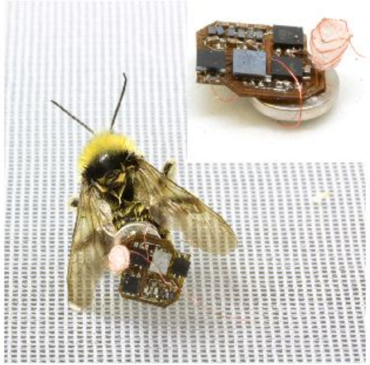
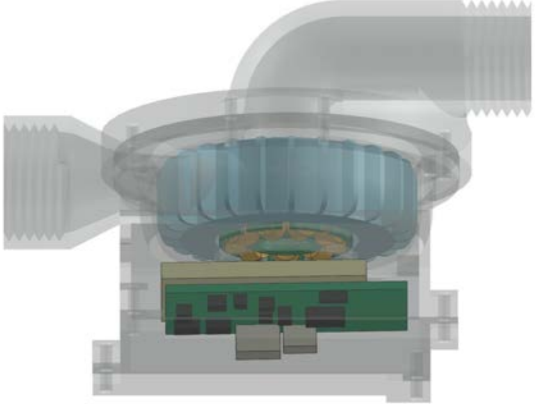
Dynamic Multi-Clock Management for Embedded Systems

Holly Chiang, Hudson Ayers, Daniel Giffin,
Amit Levy, Philip Levis





Deployments are limited by battery lifetime

Sensor energy profile

- Sensor deployment can be a very costly operation in terms of man hours or getting authorization

  Maximize deployment -> extend battery life -> reduce energy usage

- Where do embedded applications spend their energy?
  - Most time spent in deep sleep
  - Most energy spent on brief active periods of I/O and computation

How can we reduce the energy used during active periods?

Clock sources

To support energy efficient applications, modern microcontrollers have multiple clock sources

- Faster clocks use more power but tend to be more energy efficient
- Up to two orders of magnitude difference in power draw depending on clock choice

Clock	**Frequency**	**Current**	**Startup**
RCSYS	113600 Hz	12 μA	38 μs
RC1M	1 MHz	35 μA	-
RCFAST4M	4.3 MHz	90 μA	0.31 μs
RCFAST8M	8.2 MHz	130 μA	0.31 μs
RCFAST12M	12 MHz	180 μA	0.31 μs
OSC0	16 MHz	-	-
RC80M	80 MHz	300 μA	1.72 μs
PLL	48-240 MHz	120-500 μA	30 μs
DFLL	20-150 MHz	122-1919 μA	100 μs

Motivation

- Most applications choose a static clock
  - Lowest power clock that meets application requirements
  - Energy efficient clock
  - Default clock
- Hand coding clock changes is difficult
  - Requires developer to have hardware specific peripheral knowledge
  - Bug prone
  - Not portable
  - Doesn’t work for multiple apps

Power Clocks: dynamic clock management in the kernel

Challenges

- **What** clock to change to
  - Peripherals have wide range of hardware-specific clock requirements
  - I/O vs compute bound peripherals
- **When** to change the clock
  - Ensure most efficient clock is always being used
- **How** to ensure correctness
  - Synchronous vs asynchronous peripherals

	Read (μJ)	Write (μJ)	Read (ms)	Write (ms)
RCSYS	22176	**165**	1235.3	5.2
RC1M	2046	185	147.6	5.6
RCFAST4M	581	198	34.5	5.6
RCFAST8M	393	215	19.0	5.6
RCFAST12M	327	215	13.2	5.4
OSC0	267	234	9.8	5.4
RC80M	221	320	4.7	5.3
PLL	215	320	4.2	5.4
DFLL	**205**	320	3.7	5.4

API

ClockManager

set_max_frequency

set_min_frequency

set_clocklist

set_need_lock

enable_clock

disable_clock

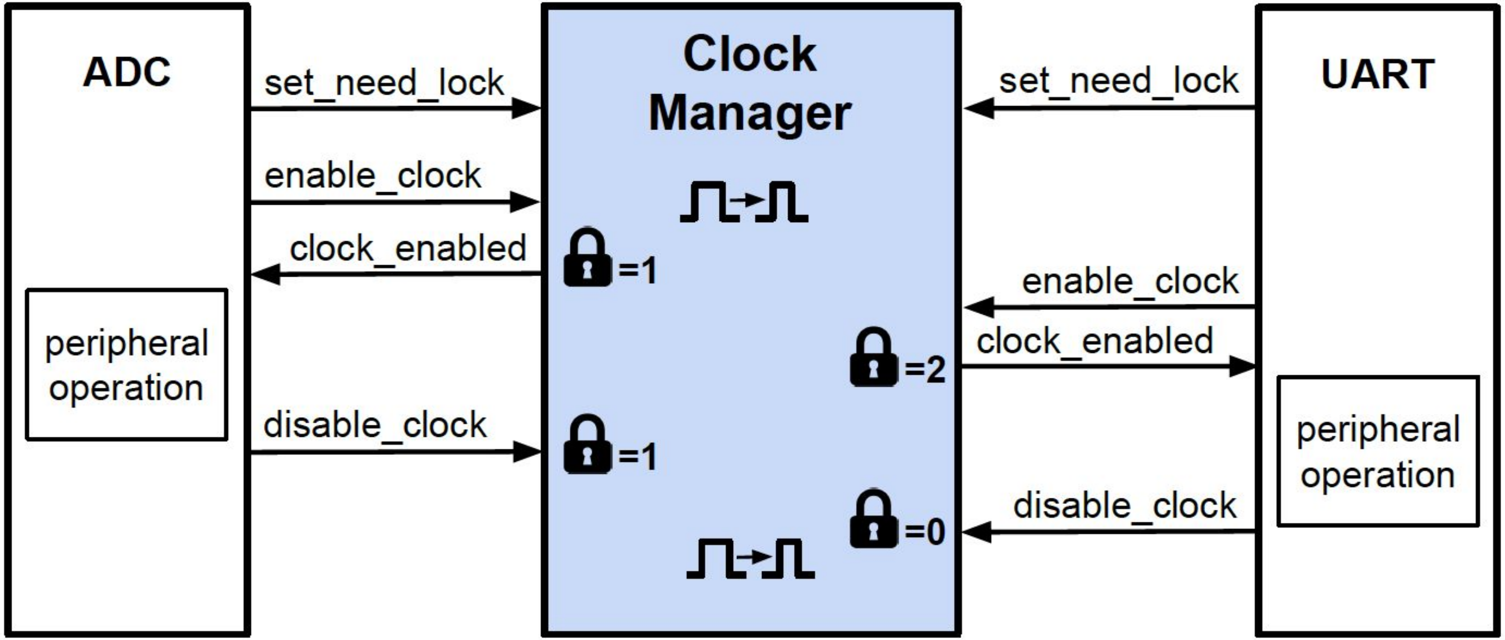
ClockClient

clock_enabled

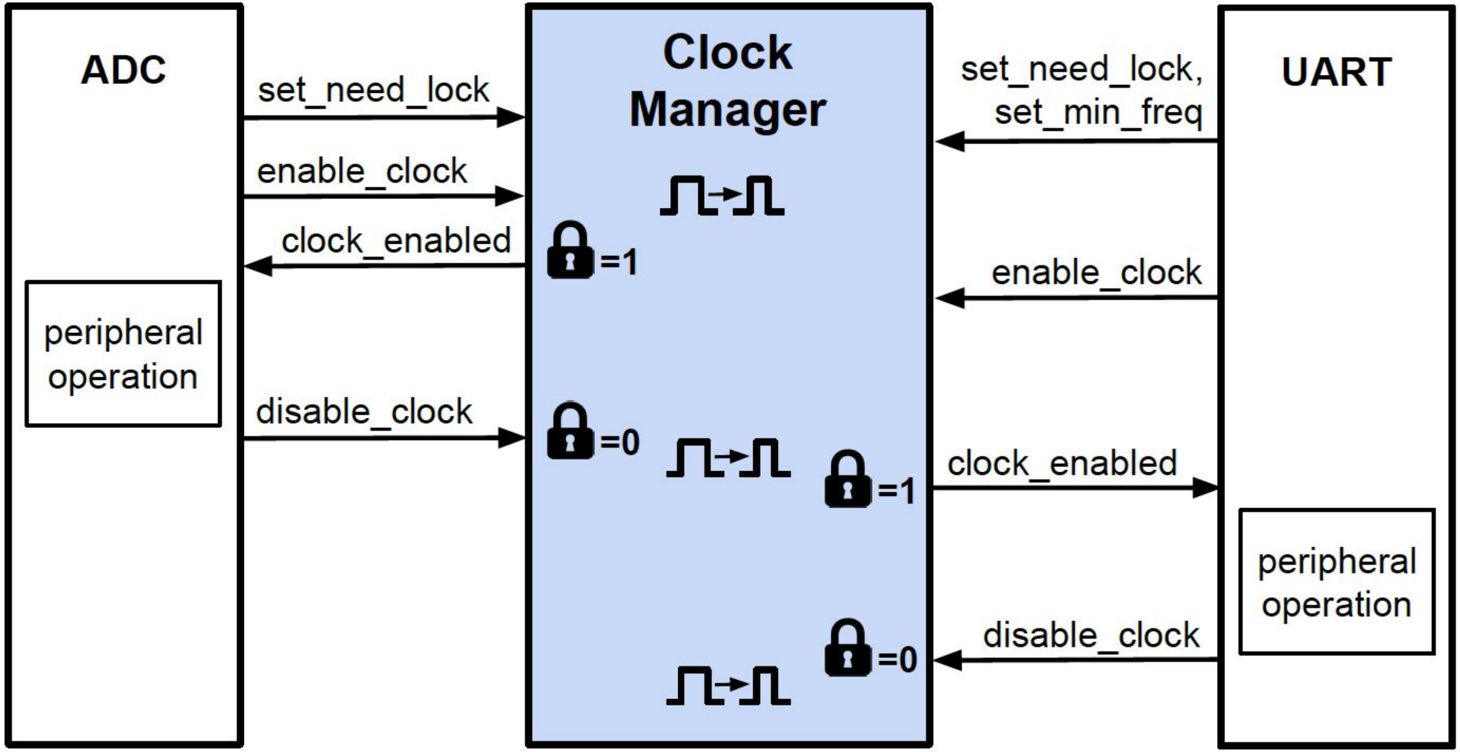
```
sample(freq):
    //setup ADC
    //start sampling

stop_sampling():
    //disable ADC
```

```
sample(freq):
    CM.set_min_freq(freq*32)
    CM.set_need_lock()
    CM.enable_clock()

clock_enabled():
    //setup ADC
    //start sampling

stop_sampling():
    //disable ADC
    CM.disable_clock()
```

Compatible clock requirements



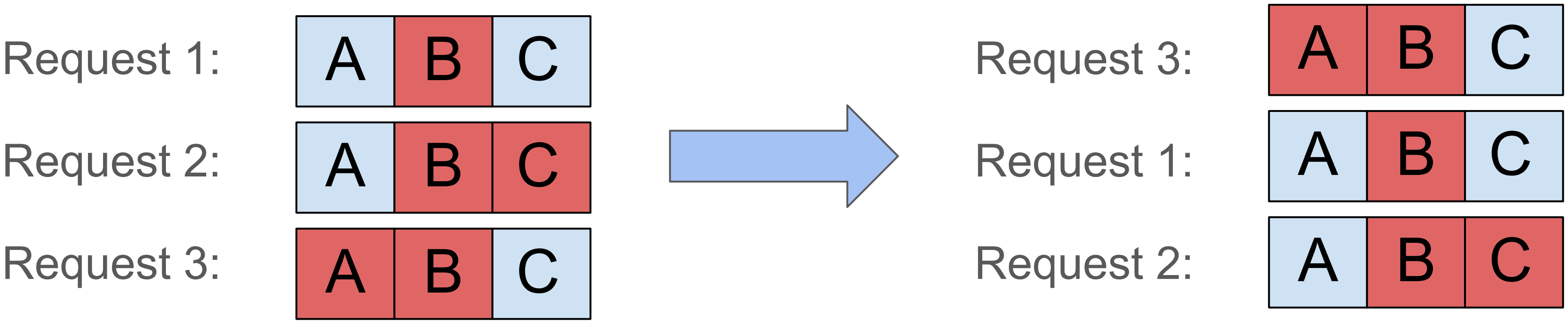
Incompatible clock requirements



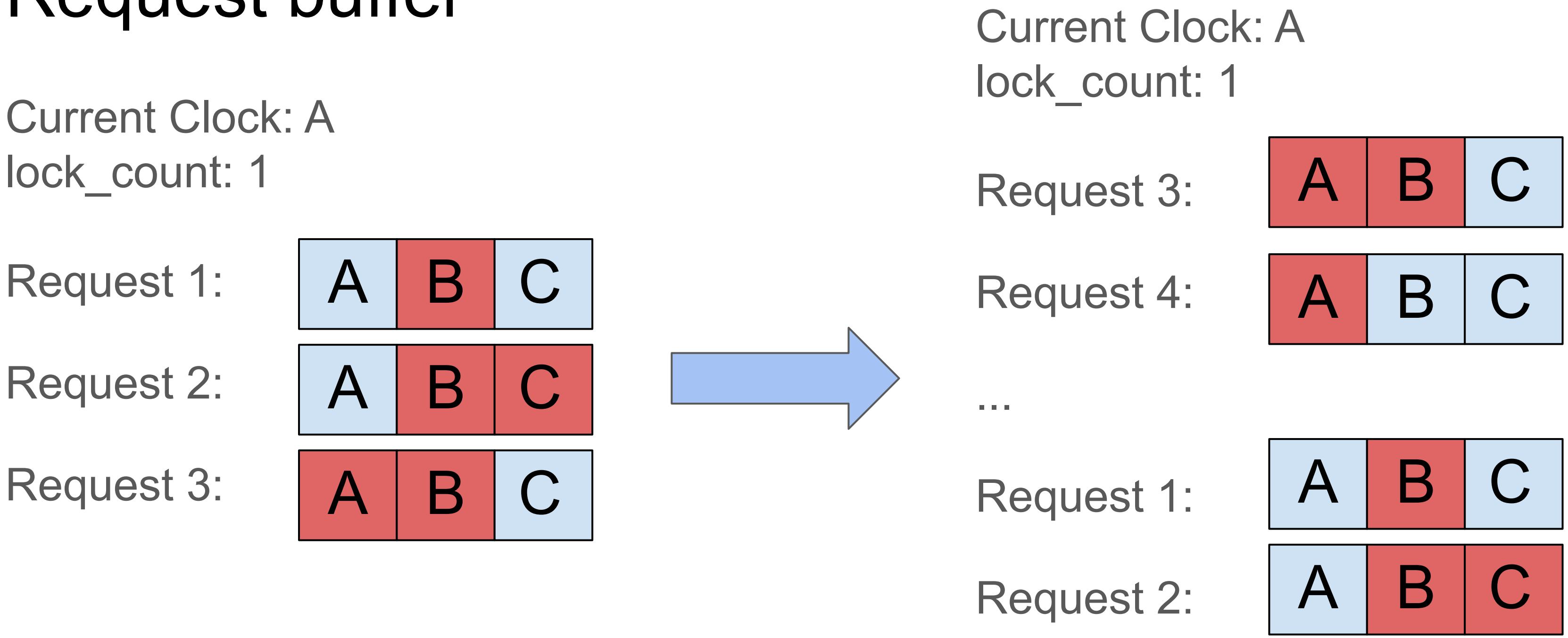
Request buffer

Current Clock: A
lock_count: 1

Current Clock: A
lock_count: 1



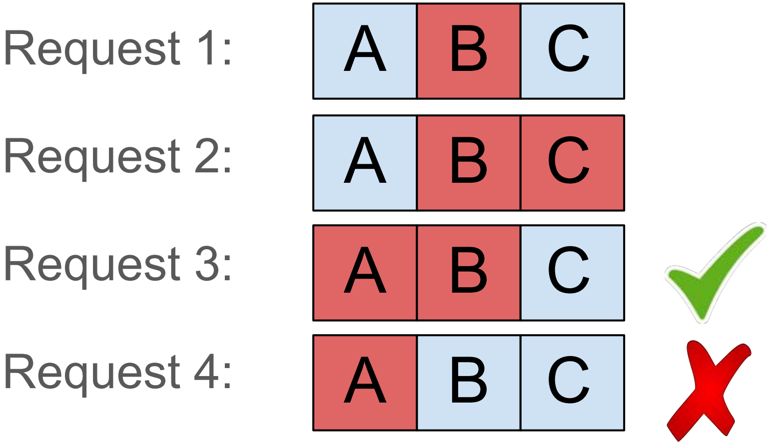
Request buffer



Will extend the starvation of a proceeding request

Request buffer

Current Clock: A
lock_count: 1



If head of request queue is blocked, a new request will run only if:

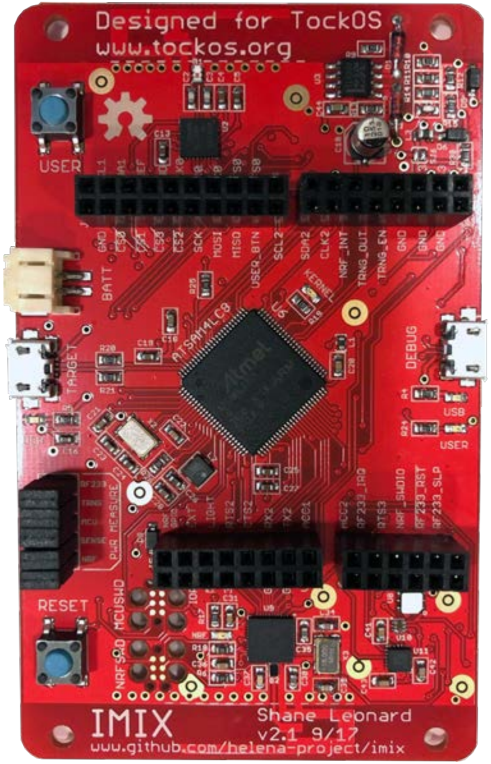
1. Request does not require a lock
2. Must exist a clock that satisfies both the requesting client and all proceeding clients in the queue

Limitations

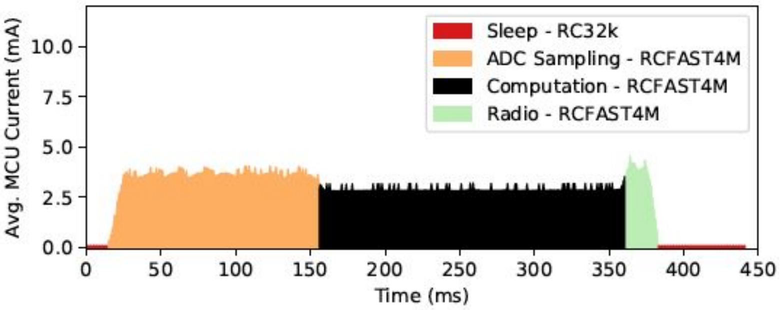
- Only works if peripherals have limited runtime
  - Energy-limited applications keep peripheral operations short
- Timing delay caused by locking
  - Peripheral operations are already asynchronous
- Does not perform well on applications that work best with static clock
  - Adds inefficiency due to clock change overhead

Implementation

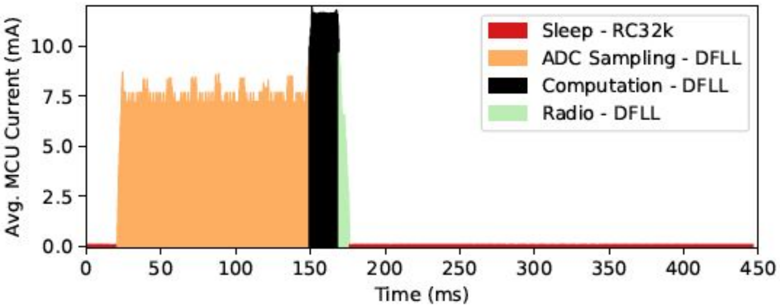
- Tock - secure embedded OS written in Rust
  - Allows multiple concurrent applications to run on a single microcontroller
- imix development board
  - SAM4L Cortex M4 (64kB RAM, 512 kB flash)
  - BLE and 802.15.4 radios, variety of sensors and I/O buses
  - Jumpers to measure power for each subsystem



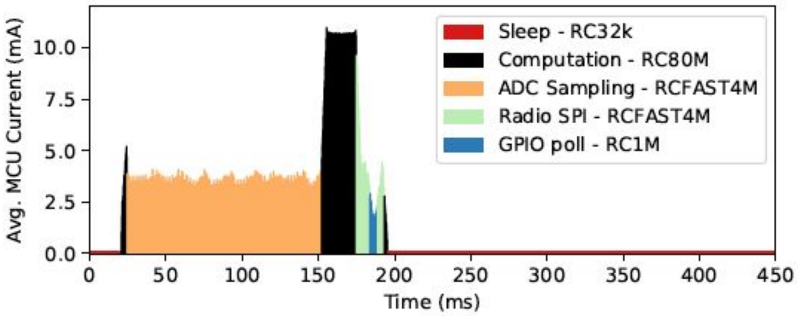
ADC-Radio application



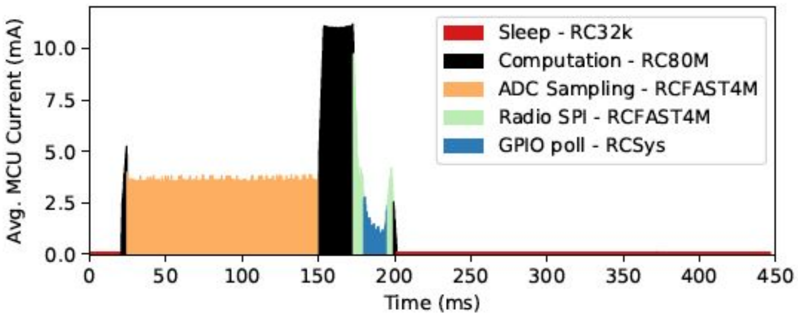
(a) Static RCFAST (4 MHz)



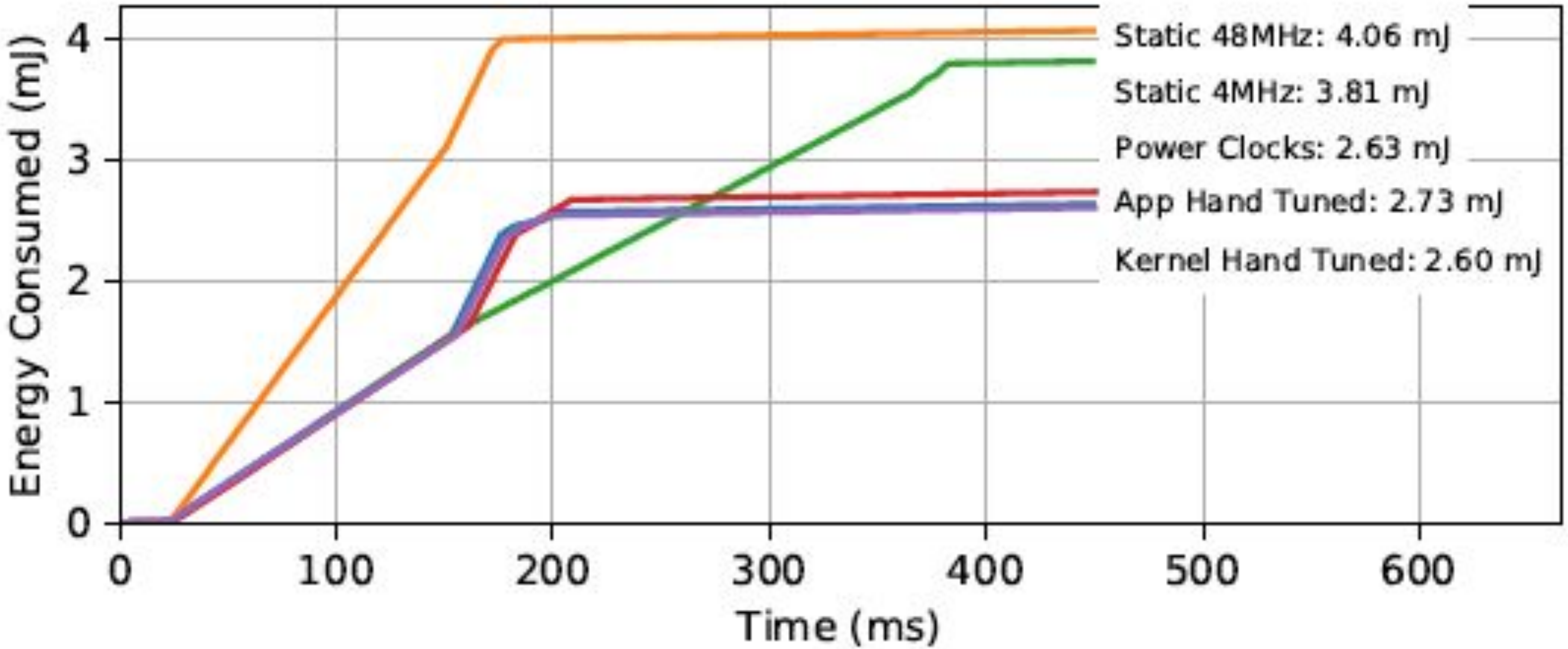
(b) Static DFLL (48 MHz)



(c) Hand Tuned



(d) Power Clocks



Energy Consumed (mJ)
0
1
2
3
4
Time (ms)
0
100
200
300
400
500
600
Static 48MHz: 4.06 mJ
Static 4MHz: 3.81 mJ
Power Clocks: 2.63 mJ
App Hand Tuned: 2.73 mJ
Kernel Hand Tuned: 2.60 mJ

Code overhead

	ROM	diff	RAM	diff
Tock	131352	-	57344	-
+ClockManager	131404	52	57344	0
+ADC	134076	2672	57344	0
+Flash	134540	464	57348	4
+I2C	134736	196	57348	0
+SPI	134956	220	57348	0
+USART	135208	252	57348	0
Total		3856		4

CPU cycle overhead

Function	Changes clock	Lock	CPU Cycles
enable_clock			113-189
		Yes	110-217
	Yes		722
	Yes	Yes	717
disable_clock			276
		Yes	123
	Yes		749
	Yes	Yes	587

Conclusion

- Embedded applications can significantly extend their deployment lifetimes by dynamically changing the clock in response to application workloads
- Changing the clock manually places the engineering burden on the developer
  - Requires hardware-specific knowledge, bug prone, not portable, doesn’t work when running multiple apps
- Power clocks automatically manages clock changes in the kernel
  - No application changes necessary
  - 31% less energy than an optimal static clock
  - Minimal code and CPU cycle overheads