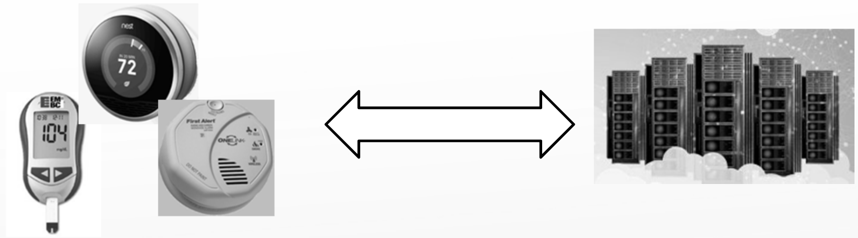


MULTICORE PROGRAMMING IN APPLICATIONS RANGING FROM IOT EDGE NODES TO THE CLOUD

Markus Levy

EEMBC President

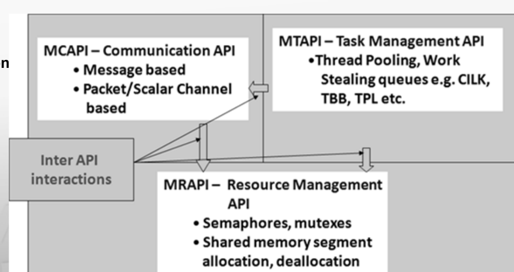
Multicore Association President

MPSoc July 13-17

MCA API ABSTRACTIONS; MXAPI TAXONOMY

- Heterogeneous and resource-constrained embedded systems
- Allows implementation of existing programming interfaces, such as OpenMP, OpenCL
- C-API and pure library (no language extensions), enhances portability
- Commercial and open source implementations available
- Support by tools vendors

API and semantic for communication and synchronization between processing cores



Creating low-level tasks that can delegate their execution to HW or SW nodes being part of a system consisting of homogeneous or heterogeneous processors

Application-level management of shared resources

An Industry Standard = Many Years to Wide-Spread Adoption

- MCAPI effort started in 2005
- Version 1.0 released in 2009
- Version 2.0 release in 2011
- Version 3.0 starting 2014 (complete in 2015?)
  - Zero copy messaging
  - Subsets for messaging, channels, IoT, etc.
  - Safety critical support

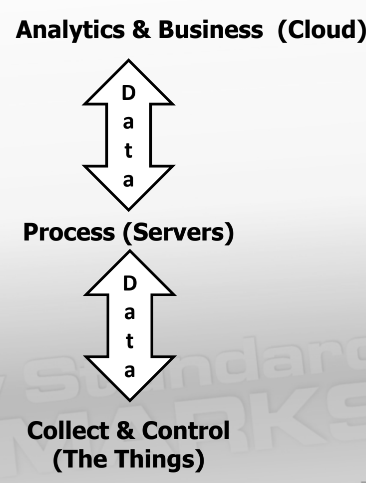
Why Developers are Using MCAPI

- Functionality aligns with design requirements
- Messaging scales
  - Local and remote nodes
  - Transport independent communications
  - AMP and SMP
- Application portability
  - Separates application from platform and architecture
  - Heterogeneous and homogeneous ISA, OS and transport
- Internet of Things. Really?

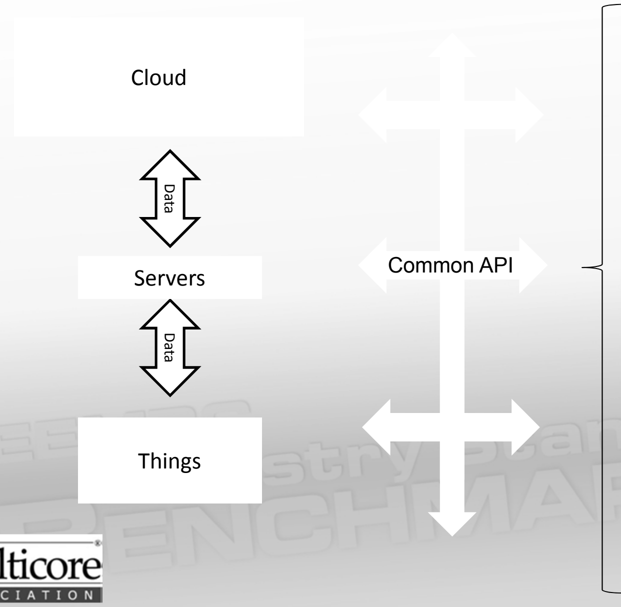
APPLYING MULTICORE DEVELOPMENT TECHNIQUES TO THE IOT

IoT is 'Widely Distributed' Multicore

- Simplify data sharing and synchronization for IoT applications, between any type of core, any transport, any OS.



DATA SHARING ACROSS THE IOT



Enterprise
Fibre Channel
Googlexxx
IBMxxx
Infiniband
MSxxx
OFED
SNA
TCP/IP
USB

Application Server
ATM
CDMA
GSM
Infiniband
LTE
PCIx
Serial
Sockets
SRIO
UDP
UMTS
USB
WiGig

Field Devices
6LoWPan
ANT
Bluetooth
DASH7
Firewire
ISA100
Nest
WIFI
Wireless HART
Wireless M-Bus
Z-Wave
Zigbee
Zigbee-IP

WHY MCAPI?

- Robust API that enables communications across compute elements (cores, hardware accelerators, etc.)
- Allows the implementation to communicate within same chip or for inter-chip communications
- Can use the same API to share data (communicate) across many transports.
- An application node may be located on a core in the same chip or an IoT node in the infrastructure.

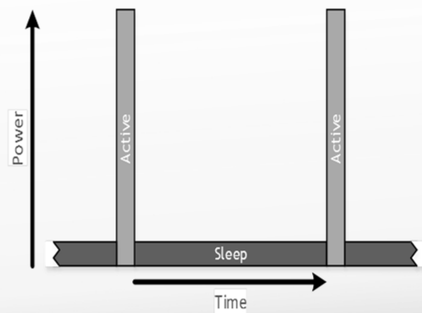
How MCAPI?

- MCAPI is the “tip of the iceberg”, providing the API to the application.
- Consider the API as the steering wheel and accelerator pedal.
  - The driver only sees the steering wheel and accelerator pedal.
  - The steering implementation may be mechanical or electrical (drive by wire). Same steering wheel API and same driver.
- The “invisible” implementation (middleware) makes it all work with the many protocols and interconnects.
- MCAPI enabled application nodes can communicate across the supported transports with the same MCAPI functions and with the same application source code.
  - Commercial tools, such as Poly-Platform can select and configure the transport
  - Supports shared memory, closely distributed wire transports (e.g. RapidIO) and widely distributed transports (e.g. USB, Infiniband).

IoT Benchmark Strategy to Quantify IoT ‘Sleepy Node’ Energy Efficiency

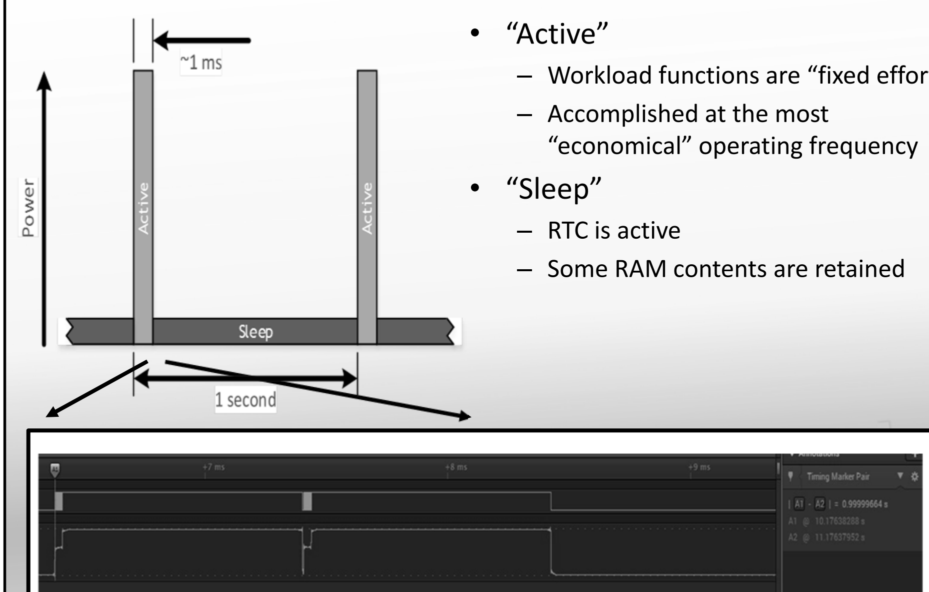
IOT EDGE NODES CONTINUED

A Simple Sleepy Node Energy Profile



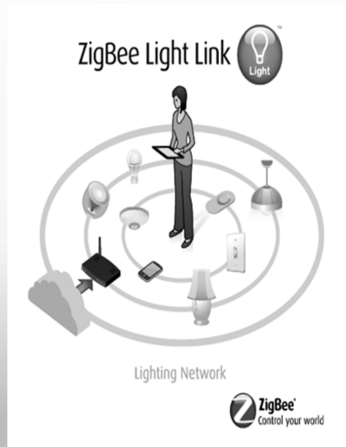
- Sleepy Node has 2 power states
  - "Active" – sensing, computing, sharing
  - "Sleep" – timekeeping, waiting for events
- Energy is the integration of power over time
- "Duty Cycle" is the ratio of "Active" time over the period

Case Study – ULPBench-CP



- "Active"
  - Workload functions are "fixed effort"
  - Accomplished at the most "economical" operating frequency
- "Sleep"
  - RTC is active
  - Some RAM contents are retained

WHERE ULPBENCH CAN BE EXTENDED FOR IOT

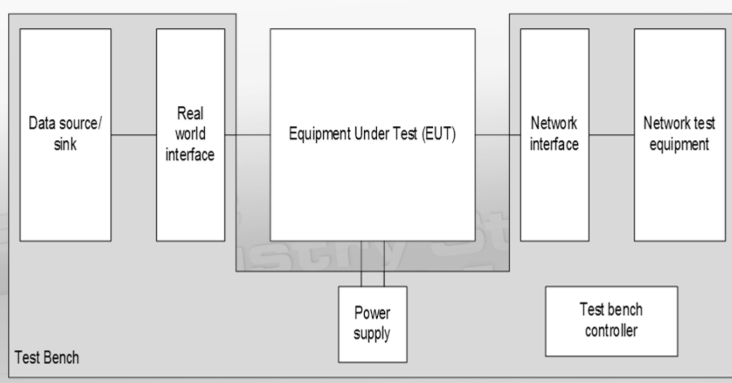


Application Focus!

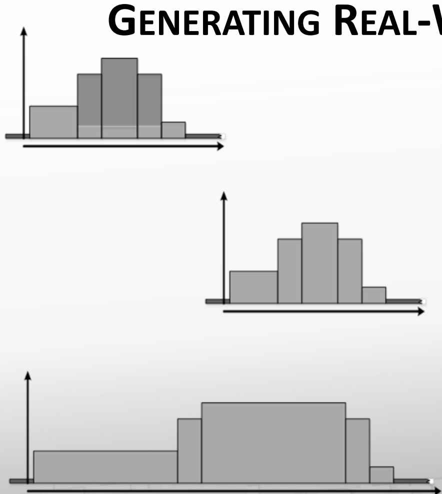
- Consider energy needed for communication, especially wireless
  - Wireless radio energy
  - Security related compute
- Consider sensing
- How to derive a standard applicable across IoT applications?

BENCHMARK MEASUREMENT IMPLEMENTATION EXAMPLE

- The device under test
- The ULPBench energy measurement hardware
- A scriptable coordinator that is compatible with the wireless protocol



Generating Real-World Workloads



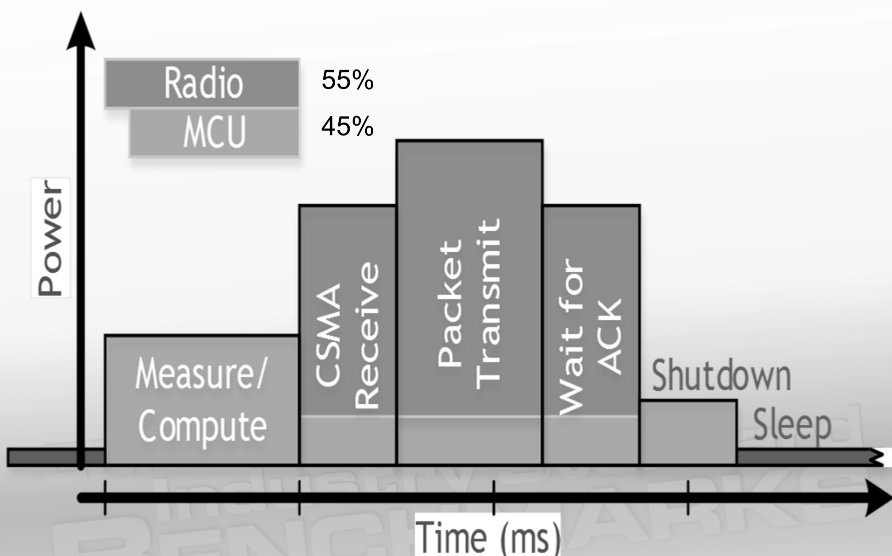
– Workload types are representative based on use cases such as
  - Light Switch/dimmer
  - Wearable activity device

– Radio and MCU both contribute meaningfully to the energy use

– Various packet types consume different amounts of energy

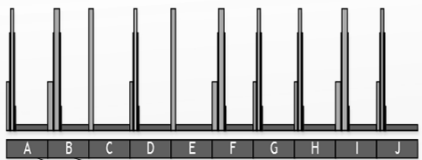
– Security adds to energy cost by increasing CPU computation and RF packet length

ZigBee Data Poll Energy Profile – MCU / Radio split



ANALYZING THE RESULTS

Characteristic Transactions



Frequency of Transactions by Application Type

Applications	A	B	C	D	…	J
Zigbee Light Switch	1400	37	0	10	…	0
Health Monitor	0	3600	0	0	…	0
…					…	

- Applied to ZigBee Light Switch
  - Energy = A*1400 + B*37 + C*0 + …
- Applied to Health Monitor
  - Energy = A*0 + B*3600 + …

WRAP UP

- Why MCA? Why MCAPI?
- Shift to the IoT
- Meeting the battery life requirements for IoT
- Questions?