Lecture 17
Energy Management

CE346 – Microprocessor System Design

Branden Ghena – Spring 2021

Some slides borrowed from:
Josiah Hester (Northwestern), Prabal Dutta (UC Berkeley), Brandon Lucia (Carnegie Mellon)

Administrivia

- Quiz 4 is available
  - Due Friday
- Demo details coming on Wednesday
- ~9 days to complete your project

Today's Goals

- Consider energy and microcontroller systems
  - What are the energy sources?
  - What are the energy sinks?

- Understand sleep mode on a microcontroller

- Discuss software for low-energy operation

Outline

- **Energy Sources**

- Energy Management
  - Sleep
  - Intermittent computing

Energy sources

- Need some way to gain energy to power system

- Choices affect mobility, lifetime, software design, and capabilities
    - Is the device tied to its power source?
    - Must a human manually provide more energy?
    - How do I write software to ensure “progress” is made?
    - What sensors/actuators can the device run?

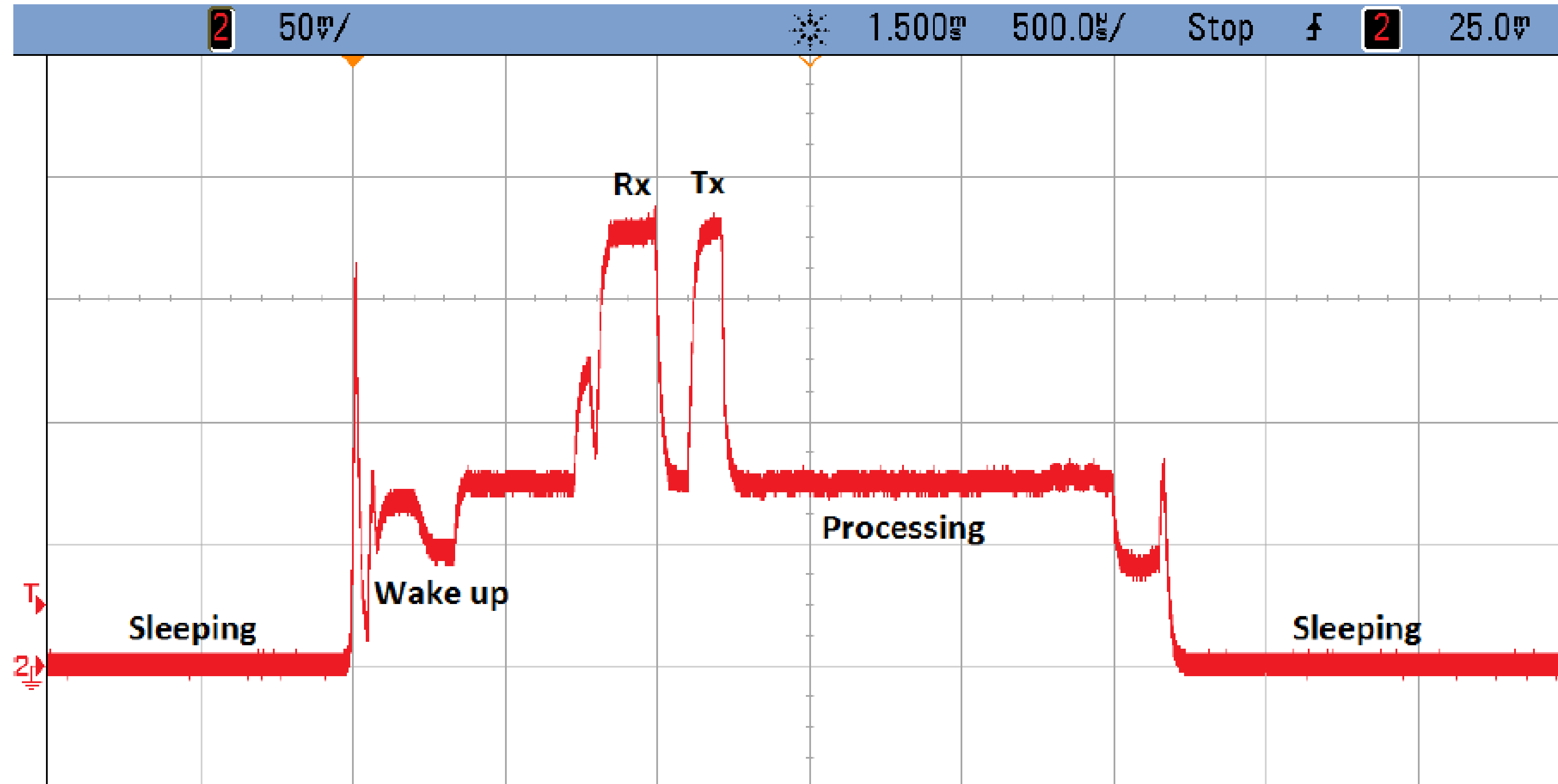
Measuring energy use

- Base equations
  - Power = Current * Voltage (Watts)
  - Energy = Power * Time (Joules)

- Energy = volts * amps * seconds
  - Voltage is *usually* constant for a system
  - Time is how long you are running for / measurement period

  - Current changes based on activities being done
    - Often energy is presented as a current draw
    - Maybe an average current draw
    - With Voltage and Time implicit

Example current trace during wireless communication



Current Consumption versus Time during a single Connection Event

Wired power through USB

- Provides 5v at up to 500 mA (USB 2.0) or 900 mA (USB 3.0)
  - Or power delivery specifications, which can do far more power

- Must be converted to different voltage to use
  - Voltage regulator takes in 5v and spits out 3.3v
  - Has its own maximum current!

- System is limited by the minimum of USB or regulator power
  - Microbit: regulator gives 3.3v at up to 600 mA
    - USB 2.5 Watts, Regulator 1.98 Watts ⇨ System 1.98 Watts
    - This is a max! Stay 15-30% below regulator limit

Thinking about energy

- Batteries often list energy in mA*h (milliamp – hours)
  - Coin cell battery: 3v at 220 mAh
  - 2x AA battery: 3v at 2000 mAh
  - iPhone 11 battery: 3.7v at 3000 mAh

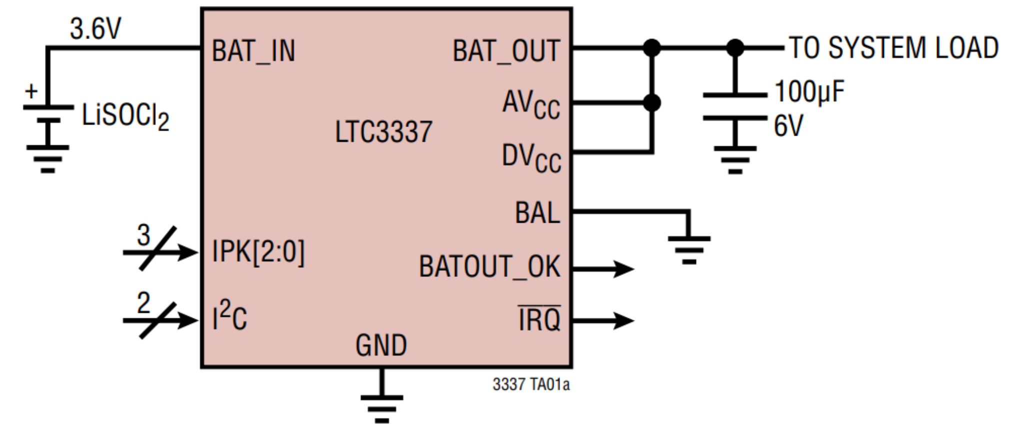
- Also usually limited by regulator
  - Sometimes just directly connected to system
  - We can run at 3v just fine! (3.7v is no good though)

- Voltage can vary with charge
  - But only a little, right before battery is depleted
  - Example: coin cell goes down to ~2.7 volts

How are batteries measured?

- Measuring energy remaining is a difficult problem
  - Many questions to be handled
    - How much did it start with?
    - How much energy has been used?
    - What type of battery is it?
  - Energy is not as constant a quantity as one would hope
    - Pulling out lots at once has an overhead penalty

- Coulomb Counter (aka Battery Fuel Gauge)
  - Designed for a specific battery "chemistry"
  - Monitors charge flowing in each direction
  - I2C interface for reading battery state



- Accuracy is not exact, more of an educated guess

How are batteries managed?

- Usually a dedicated IC for charging and managing battery packs
  - Recharges battery with appropriate amount of current

  - Monitors issues of battery health
    - Various status monitoring
      - Overcharge, undercharge
      - Overcurrent
      - Overtemperature, undertemperature
    - Will go so far as to cut off the system to protect the battery

- Takeaway: complicated problem, approach with caution!
  - Best to reuse an existing design, if possible

Microbit only uses battery energy in a simple way

- Battery input connects directly to regulator
  - No protection for battery health
  - No battery charging capabilities

- Usually this is fine for simple, low-power systems

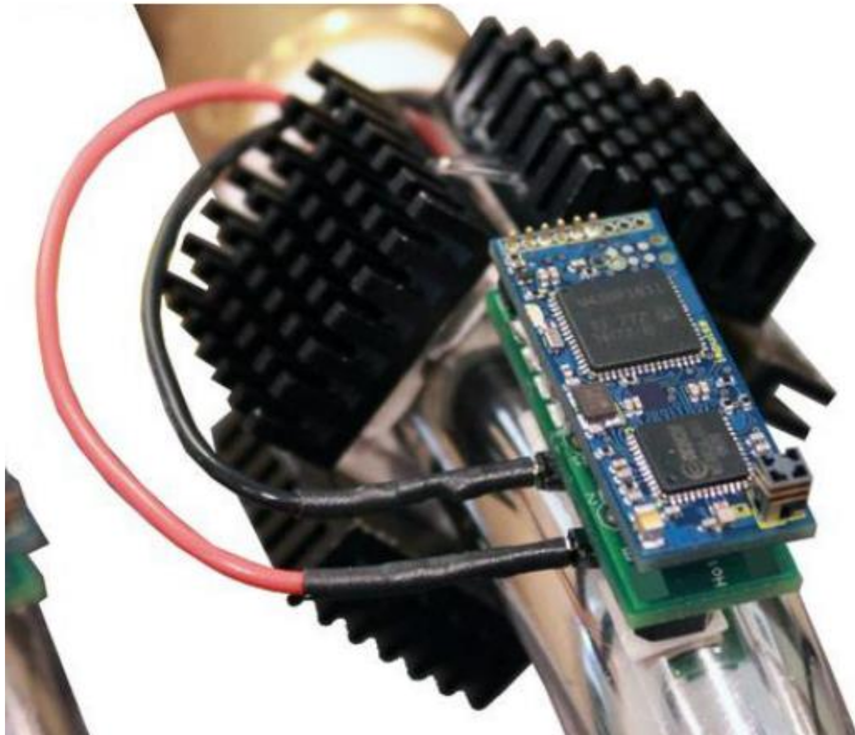
Energy harvesting

- Grab energy from the environment and use that!
  - Could augment with a battery and use energy to recharge
  - Could go entirely batteryless and live on harvested energy alone

- Sources
  - Light (outdoor or indoor. most successful)
  - Airflow (outdoor or air vents)
  - Motion (on human body)
  - Temperature differential (difficult in practice)
  - RF (very low energy source)

Temperature harvesting from hot pipes

- Peltier junctions create a voltage from temperature differential
  - Challenge: needs a large differential for more energy

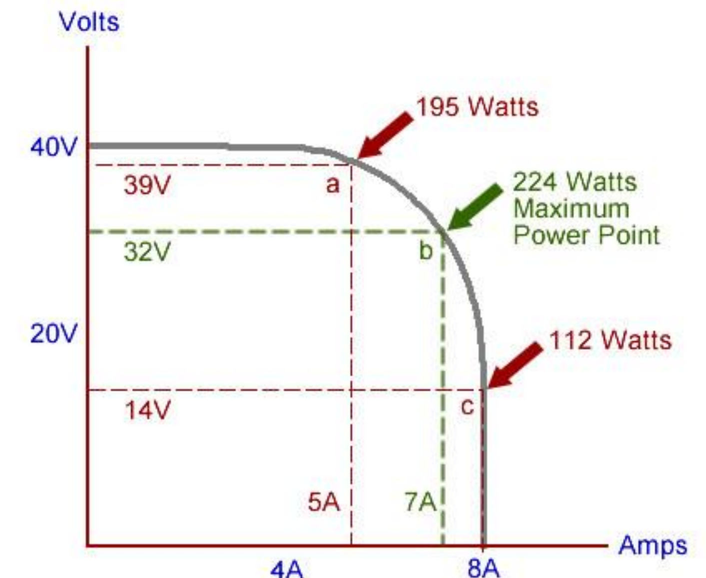


Managing harvested energy

- Often uses an IC to pull in energy and provide to system

- Harvested voltage/current are often very small
  - Signal in millivolts is pretty common
  - Need to accumulate over time to power system
    - Fill up a capacitor

- Need particular load for maximum power
  - ICs often implement Maximum Power Point Tracking (MPPT)
  - Varies load automatically to always harvest the most possible energy



Outline

- Energy Sources
- **Energy Management**
  - **Sleep**
  - Intermittent computing

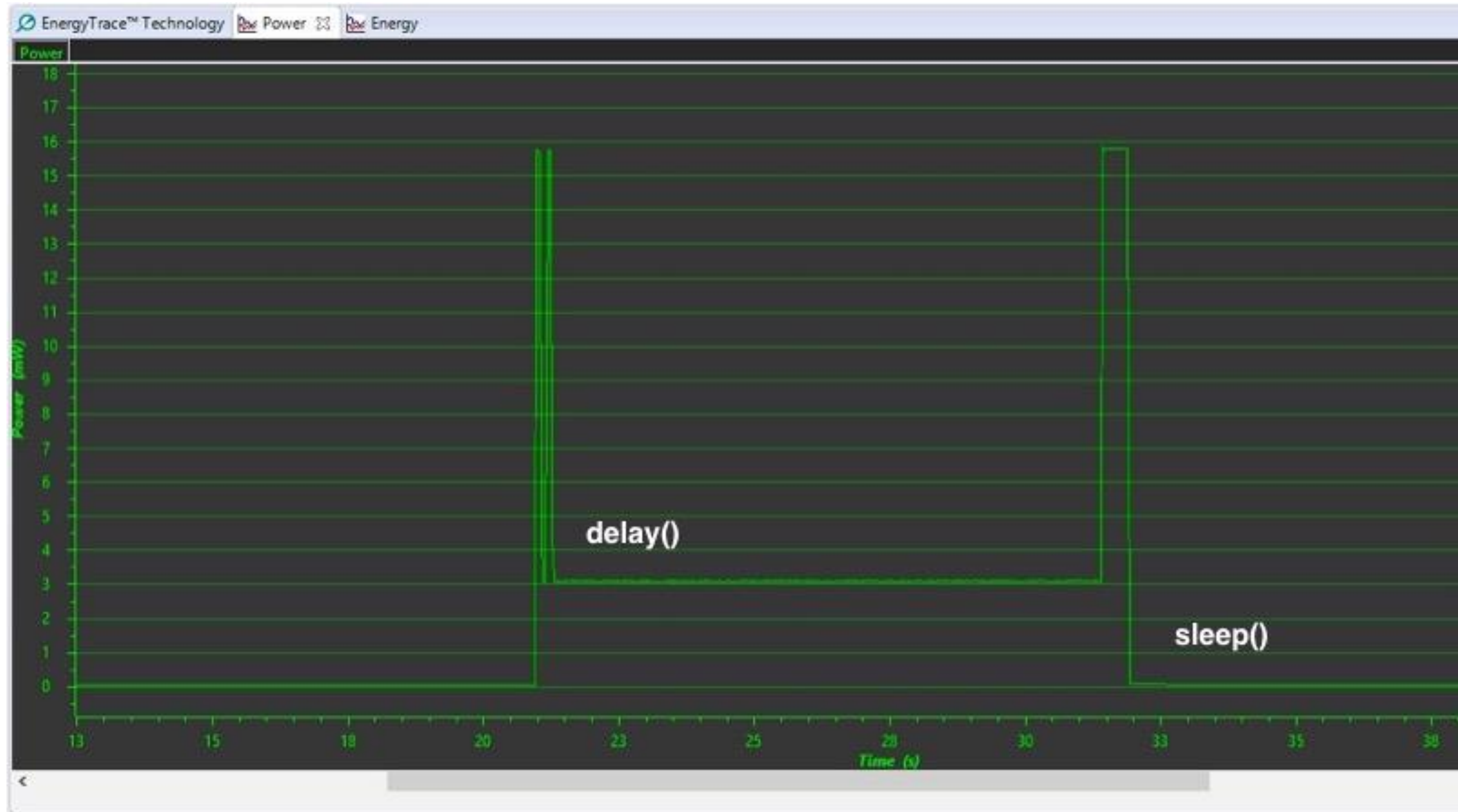
Thinking about energy

- Battery energy
  - Coin cell battery: 3v at 220 mAh
  - 2x AA battery: 3v at 2000 mAh
  - iPhone 11 battery: 3.7v at 3000 mAh

- nRF52833 active current: 5.6 mA (at 3v)
  - Coin cell: 40 hours -> ~2 days
  - 2x AA: 360 hours -> ~15 days
  - iPhone 11: 535 hours -> ~22 days

- So how does any of this work???

Sleep mode power draw



Microcontroller sleep modes

- Sleep mode
  - Processor stops running
  - Most peripherals are disabled
  - Continues until an interrupt occurs and wakes the microcontroller
    - Usually a timer or GPIO input

- nRF52833 sleep mode current: 1.8 μA (GPIO port event only)
  - Coin cell: 122222 hours -> ~5000 days -> ~14 years

- Low-power systems shoot for less than 1% duty cycle
  - Average current of ~100 μA or less
  - Warning: other stuff on the board counts!!
    - LEDs are 1-10 mA each… Power is not a concern of the Microbit

Microbit current draw (microcontroller)

- Active CPU
  - 5.6 mA (executing from Flash)
  - 1.8 μA (sleep mode with RAM retention)

- Transmitting RF packet
  - 15.5 mA (+8 dBm)

- Other peripherals
  - SAADC: 1.37 mA
  - Timer: 729 μA (for any Timer peripheral)
  - I2C: 6.6 mA (pull-down resistors when transmitting 0 bit)
  - Everything else is handfuls of μA

Microbit current draw (non-microcontroller)

- KL27 (JTAG interface microcontroller)
  - 2 μA sleep, 8 mA active
- Speaker
  - 0-27.5 mA (changes with input signal)
- Microphone
  - 0-120 μA (activated with GPIO pin)
- Accelerometer/Magnetometer
  - 2-212 μA (depends on sensing rates, 200 is magnetometer)
- LEDs
  - 0-230 mA (can be activated individually)
- Everything else
  - 0-1 mA (mostly due to pull-up resistors)

Max and min current for Microbit

- Maximum current: 280 mA at 3.3 volts (~1 W)
  - With *everything* active
  - Well within limits of regulator

- Minimum current
  - ~15 mA (always-on power LED)

  - If you removed the power LED:
  - <100 μA (with everything off)

nRF52 sleep mode

- Triggered with assembly instruction
  - WFI (Wait For Interrupt) or WFE (Wait For Event)

- Stops processor until woken by interrupt, exception, or event

- On nRF52 automatically disables high frequency clock if unneeded

```
__attribute__((always_inline)) __STATIC_INLINE void __WFI(void)
{
  __ASM volatile ("wfi");
}
```

Software stops when the processor does, but peripherals continue

- Problem: when the processor is off, no code is running

- Solutions
  - Peripherals can wake it up again
    - Can probably go for milliseconds to minutes without any actions
    - Timer interrupt can wake processor to do things

  - Have hardware handle some parts in the background without the processor's involvement
    - DMA
    - PPI

Controlling peripherals while processor sleeps

- DMA (Direct Memory Access)
  - Set up a pointer to memory and a length
  - Peripheral can load/store memory without processor's involvement
  - Usually use completion interrupt to wake processor

- PPI (Programmable Peripheral Interconnect)
  - Any Event can be tied to any Task within the nRF52
  - Allows for complicated actions to be chained together

nRF52 Tasks and Events

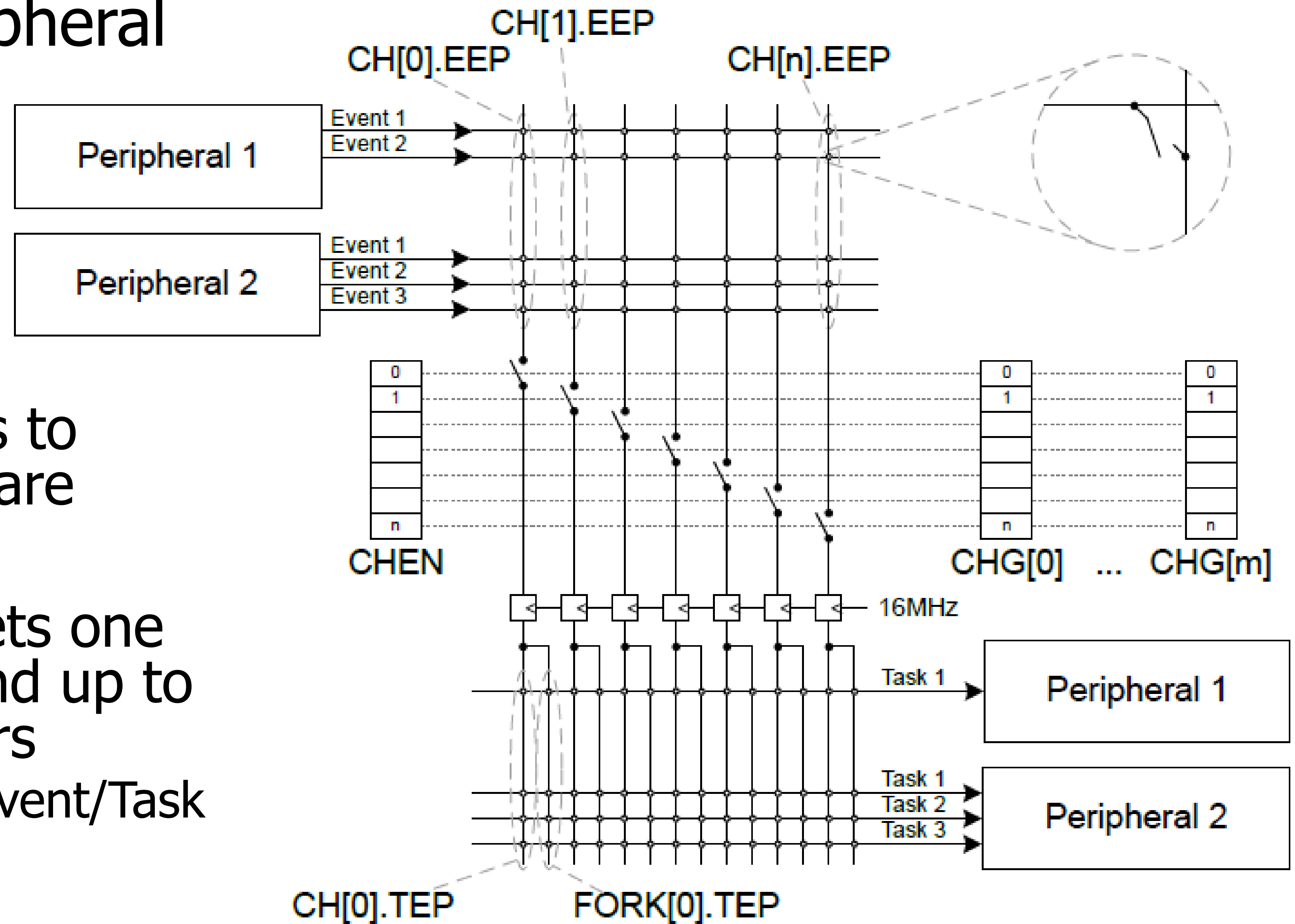
- Tasks are used to perform some operation
  - Often written to by software
- Events change value when some change in status occurs
  - Often used to trigger interrupts
- PPI peripheral can connect any TASK to any EVENT

Example: Timer peripheral

Register	Offset	Description
TASKS_START	0x000	Start Timer
TASKS_STOP	0x004	Stop Timer
TASKS_COUNT	0x008	Increment Timer (Counter mode only)
TASKS_CLEAR	0x00C	Clear time
TASKS_SHUTDOWN	0x010	Shut down timer
TASKS_CAPTURE[0]	0x040	Capture Timer value to CC[0] register
TASKS_CAPTURE[1]	0x044	Capture Timer value to CC[1] register
TASKS_CAPTURE[2]	0x048	Capture Timer value to CC[2] register
TASKS_CAPTURE[3]	0x04C	Capture Timer value to CC[3] register
TASKS_CAPTURE[4]	0x050	Capture Timer value to CC[4] register
TASKS_CAPTURE[5]	0x054	Capture Timer value to CC[5] register
EVENTS_COMPARE[0]	0x140	Compare event on CC[0] match
EVENTS_COMPARE[1]	0x144	Compare event on CC[1] match
EVENTS_COMPARE[2]	0x148	Compare event on CC[2] match
EVENTS_COMPARE[3]	0x14C	Compare event on CC[3] match
EVENTS_COMPARE[4]	0x150	Compare event on CC[4] match
EVENTS_COMPARE[5]	0x154	Compare event on CC[5] match

nRF52 PPI peripheral

- Connects Events to Tasks via hardware
- Each channel gets one Event pointer and up to two Task pointers
  - Must point to Event/Task registers



Example PPI use case

- Automatic high-speed ADC sampling

- Software configures and sleeps
  - ADC (buffer and enable)
  - Timer (prescaler, compare value, short from compare to clear, and start)

- PPI: When Timer fires (EVENTS_COMPARE[0]),
  - Sample ADC (TASKS_SAMPLE)

- PPI: When ADC buffer full (EVENTS_END),
  - Stop Timer (TASKS_STOP)
  - Fork: wake processor (via software interrupt from EGU)

Outline

- Energy Sources
- **Energy Management**
  - Sleep
  - **Intermittent computing**

Reducing energy consumption even further

- If sleep isn't enough, you can power things off completely
  - Transistor can be used to turn off the sensor

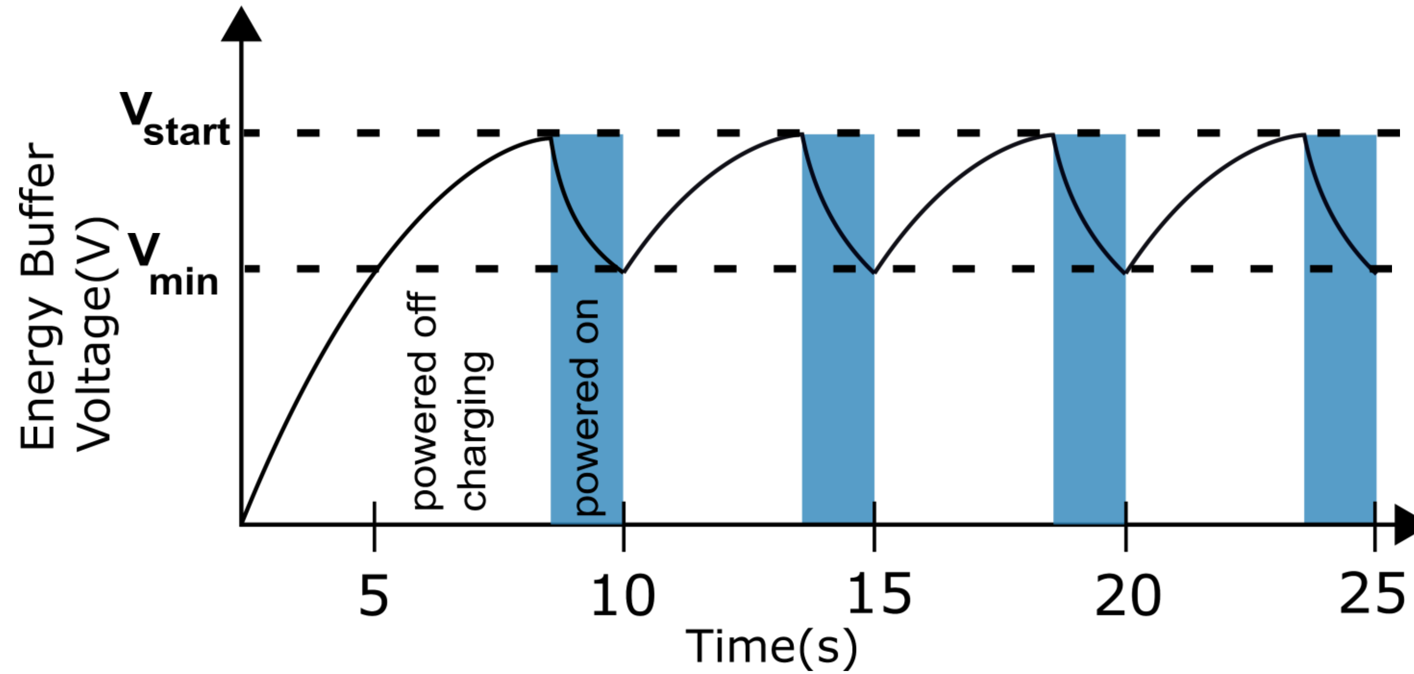
Humidity

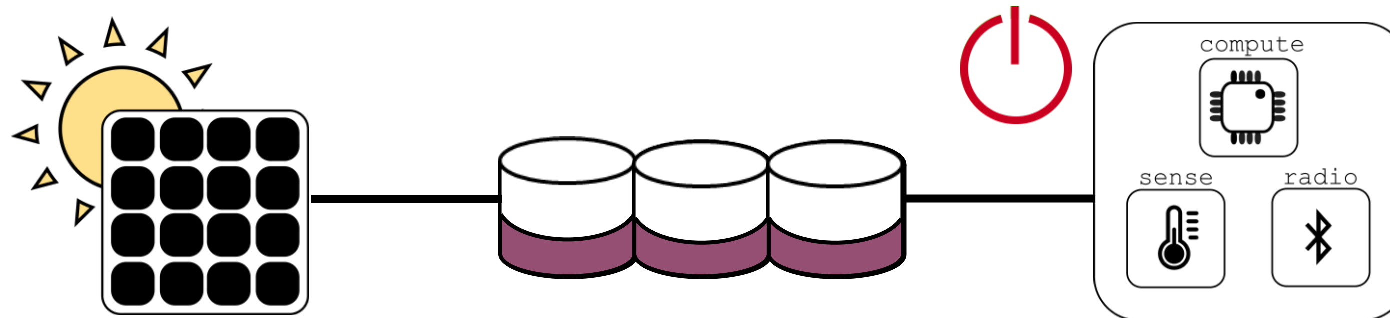
VCC
R26
100M
HUM_EN/4.10H
Q8.2
C20
0.1uF
GND
HUM_EN/1.8H
U11
VCC SCL
GND SDA
SI7021-A20-GM1
GND
Q2.1
Q2.2
SCL/4.10F
SDA/4.10F

Ambient Light

VCC
R24
100M
LT_EN/1.6K
Q7.2
C25
1uF
R9
10k
GND
LT_INT/4.10J
LT_EN/4.10H
U8
VCC INT
A0 SCL
GND SDA
MAX44009
GND
Q3.1
Q3.2
SCL/4.10F
SDA/4.10F

Energy harvesting can lead to intermittent computing





Disabling the microcontroller

- Even 2 μA sleep current might be too much for energy harvesting
  - Can turn off microcontroller periodically
  - Enable it again once VCC returns

- Problem: how do you write software to deal with intermittency?
  - Run-to-completion with relatively quick code
    - Initialize, sample sensor, send packet, turn off again

  - Code checkpointing
    - Save state from code and restore when power resumes
    - Might be as little as which state the system is in, plus some samples
    - Might be as much as saving entire stack state
    - Needs low-energy, nonvolatile storage (FRAM or MRAM help!)

Programs may not finish

```
int process() {
    count++;
    buf[count] = accel();
    avg = sum(buf)/count;
    transmit(avg);
}
```

```
count++
buf[count] = accel()
```

Power fail

Programs may not finish

```
int process() {
   count++;
   buf[count] = accel();
   avg = sum(buf)/count;
   transmit(avg);
}
```

```
count++
buf[count] = accel()
```
Power fail

```
count++;
buf[count] = accel()
```
Power fail

.
.
.

Programs may not finish

```
int process() {
    count++;
    buf[count] = accel();
    avg = sum(buf)/count;
    transmit(avg);
}
```

```
count++
buf[count] = accel()
```
Power fail

```
count++;
buf[count] = accel()
```
Power fail

.
.
.

Need to latch execution state periodically!

Checkpointing enables progress

```
int process() {
   count++;
   buf[count] = accel();
   avg = sum(buf)/count;
   transmit(avg);
}
```

Execute with checkpoints

```
count++
buf[count] = accel()
```
Power fail

```
count++;
buf[count] = accel()
```
Power fail

.
.
.

Execution Time

```
count++
```
Checkpoint
```
buf[count] = accel()
```
Power fail

Need to latch execution state periodically!

Checkpointing enables progress

```
int process() {
   count++;
   buf[count] = accel();
   avg = sum(buf)/count;
   transmit(avg);
}
```

Execute with checkpoints

```
count++
buf[count] = accel()
```
Power fail

```
count++;
buf[count] = accel()
```
Power fail

. . .

Need to latch execution state periodically!

Execution Time

```
count++
```
Checkpoint
```
buf[count] = accel()
```
Power fail

```
buf[count] = accel()
avg = sum(buf)/count
```
Checkpoint
```
transmit-
```
Power fail

Checkpointing enables progress

```
int process() {
   count++;
   buf[count] = accel();
   avg = sum(buf)/count;
   transmit(avg);
}
```

Execute with checkpoints

Execution Time

```
count++
buf[count] = accel()
```
Power fail

```
count++;
buf[count] = accel()
```
Power fail

.
.
.

Need to latch execution state periodically!

```
count++
```
Checkpoint
```
buf[count] = accel()
```
Power fail

```
buf[count] = accel()
avg = sum(buf)/count
```
Checkpoint
```
transmit-
```
Power fail

```
transmit(avg)
```

Checkpointing goals

- Have the compiler automatically insert checkpoints as needed
  - Human doesn’t have to think about them when programming

- Limit checkpointing overhead while maximizing forward progress
  - Checkpointing will take time to perform, so want to do it rarely
  - Rarer checkpoints mean more progress is lost in average outage
  - Need to compromise on the two based on available energy

Outline

- Energy Sources
- Energy Management
  - Sleep
  - Intermittent computing