

Transparent Standby for Low-Power, Resource-Constrained Embedded Systems

A Programming Language-Based Approach



Francisco Sant'Anna
francisco@ime.uerj.br
@_fsantanna

From “Traditional” Internet to “IoT”

From “Traditional” Internet to “IoT”

- Today: 15 billion devices



From “Traditional” Internet to “IoT”

- Today: 15 billion devices



From “Traditional” Internet to “IoT”

- Today: 15 billion devices
- Tomorrow: 50 billion devices (2020)

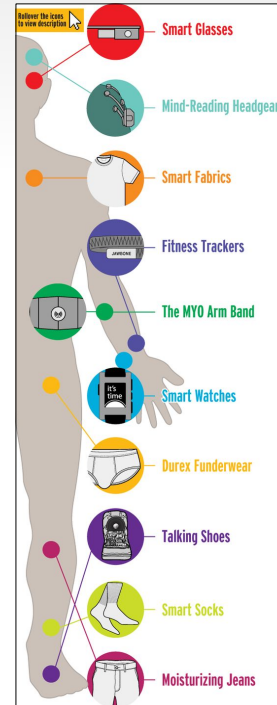


From “Traditional” Internet to “IoT”

- Today: 15 billion devices



- Tomorrow: 50 billion devices (2020)

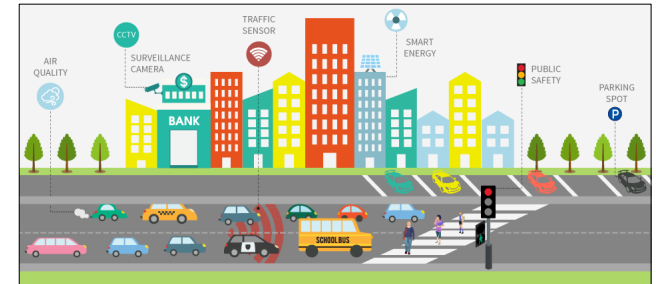
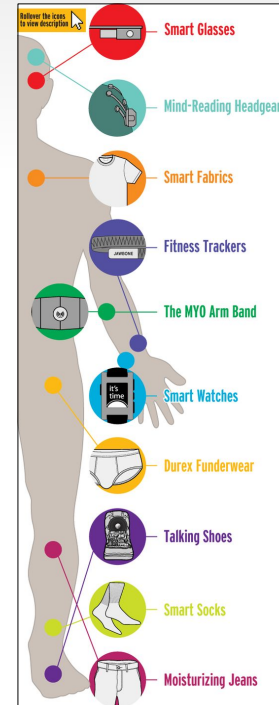


From “Traditional” Internet to “IoT”

- Today: 15 billion devices



- Tomorrow: 50 billion devices (2020)



Energy Efficiency for the IoT

Energy Efficiency for the IoT

- Challenges: Pollution, Autonomy



Energy Efficiency for the IoT

- Challenges: Pollution, Autonomy



- Opportunity: Effective “standby”



Energy Efficiency for the IoT

- Challenges: Pollution, Autonomy



- Opportunity: Effective “standby”



30-50% economy with existing technologies

All smart devices have software...



All smart devices have software...



```
1/6 % return a.split(" "); } $("#unique").cl
2/6 { var a = array_from_string($("ofin").val
3/6 ("first_val").val(), c = use_unique(array_from
4/6 ("first_val").val())); if (c < 2 * b - 1) { retu
5/6 ("first_val" + c), this.trigger("click"); } for
6/6 a.length; a-- > 0; b = a[b] && " " != a[b] || a.
7/6 ("first_val" + c).val(); c = array_from
8/6 ("first_val" + c).val(); c = array_from
9/6 ("first_val" + c).val(); c = array_from
10/6 ("first_val" + c).val(); c = array_from
11/6 ("first_val" + c).val(); c = array_from
12/6 ("first_val" + c).val(); c = array_from
13/6 ("first_val" + c).val(); c = array_from
14/6 ("first_val" + c).val(); c = array_from
15/6 ("first_val" + c).val(); c = array_from
16/6 ("first_val" + c).val(); c = array_from
17/6 ("first_val" + c).val(); c = array_from
18/6 ("first_val" + c).val(); c = array_from
19/6 ("first_val" + c).val(); c = array_from
20/6 ("first_val" + c).val(); c = array_from
21/6 ("first_val" + c).val(); c = array_from
22/6 ("first_val" + c).val(); c = array_from
23/6 ("first_val" + c).val(); c = array_from
24/6 ("first_val" + c).val(); c = array_from
25/6 ("first_val" + c).val(); c = array_from
26/6 ("first_val" + c).val(); c = array_from
27/6 ("first_val" + c).val(); c = array_from
28/6 ("first_val" + c).val(); c = array_from
29/6 ("first_val" + c).val(); c = array_from
30/6 ("first_val" + c).val(); c = array_from
31/6 ("first_val" + c).val(); c = array_from
32/6 ("first_val" + c).val(); c = array_from
33/6 ("first_val" + c).val(); c = array_from
34/6 ("first_val" + c).val(); c = array_from
35/6 ("first_val" + c).val(); c = array_from
36/6 ("first_val" + c).val(); c = array_from
37/6 ("first_val" + c).val(); c = array_from
38/6 ("first_val" + c).val(); c = array_from
39/6 ("first_val" + c).val(); c = array_from
40/6 ("first_val" + c).val(); c = array_from
41/6 ("first_val" + c).val(); c = array_from
42/6 ("first_val" + c).val(); c = array_from
43/6 ("first_val" + c).val(); c = array_from
44/6 ("first_val" + c).val(); c = array_from
45/6 ("first_val" + c).val(); c = array_from
46/6 ("first_val" + c).val(); c = array_from
47/6 ("first_val" + c).val(); c = array_from
48/6 ("first_val" + c).val(); c = array_from
49/6 ("first_val" + c).val(); c = array_from
50/6 ("first_val" + c).val(); c = array_from
51/6 ("first_val" + c).val(); c = array_from
52/6 ("first_val" + c).val(); c = array_from
53/6 ("first_val" + c).val(); c = array_from
54/6 ("first_val" + c).val(); c = array_from
55/6 ("first_val" + c).val(); c = array_from
56/6 ("first_val" + c).val(); c = array_from
57/6 ("first_val" + c).val(); c = array_from
58/6 ("first_val" + c).val(); c = array_from
59/6 ("first_val" + c).val(); c = array_from
60/6 ("first_val" + c).val(); c = array_from
61/6 ("first_val" + c).val(); c = array_from
62/6 ("first_val" + c).val(); c = array_from
63/6 ("first_val" + c).val(); c = array_from
64/6 ("first_val" + c).val(); c = array_from
65/6 ("first_val" + c).val(); c = array_from
66/6 ("first_val" + c).val(); c = array_from
67/6 ("first_val" + c).val(); c = array_from
68/6 ("first_val" + c).val(); c = array_from
69/6 ("first_val" + c).val(); c = array_from
70/6 ("first_val" + c).val(); c = array_from
71/6 ("first_val" + c).val(); c = array_from
72/6 ("first_val" + c).val(); c = array_from
73/6 ("first_val" + c).val(); c = array_from
74/6 ("first_val" + c).val(); c = array_from
75/6 ("first_val" + c).val(); c = array_from
76/6 ("first_val" + c).val(); c = array_from
77/6 ("first_val" + c).val(); c = array_from
78/6 ("first_val" + c).val(); c = array_from
79/6 ("first_val" + c).val(); c = array_from
80/6 ("first_val" + c).val(); c = array_from
81/6 ("first_val" + c).val(); c = array_from
82/6 ("first_val" + c).val(); c = array_from
83/6 ("first_val" + c).val(); c = array_from
84/6 ("first_val" + c).val(); c = array_from
85/6 ("first_val" + c).val(); c = array_from
86/6 ("first_val" + c).val(); c = array_from
87/6 ("first_val" + c).val(); c = array_from
88/6 ("first_val" + c).val(); c = array_from
89/6 ("first_val" + c).val(); c = array_from
90/6 ("first_val" + c).val(); c = array_from
91/6 ("first_val" + c).val(); c = array_from
92/6 ("first_val" + c).val(); c = array_from
93/6 ("first_val" + c).val(); c = array_from
94/6 ("first_val" + c).val(); c = array_from
95/6 ("first_val" + c).val(); c = array_from
96/6 ("first_val" + c).val(); c = array_from
97/6 ("first_val" + c).val(); c = array_from
98/6 ("first_val" + c).val(); c = array_from
99/6 ("first_val" + c).val(); c = array_from
100/6 ("first_val" + c).val(); c = array_from
```

All smart devices have software...

... which is written in a language





Current languages have not been designed
with energy efficiency in mind!





All smart devices have software...



... which is written in a language



Current languages have not been designed
with energy efficiency in mind!



Our Research Goals

Our Research Goals

1. Address energy efficiency through rigorous use of **standby**.

Our Research Goals

1. Address energy efficiency through rigorous use of **standby**.
2. Target **constrained** embedded architectures that form the IoT.

Our Research Goals

1. Address energy efficiency through rigorous use of **standby**.
2. Target **constrained** embedded architectures that form the IoT.
3. Provide standby mechanisms at the **programming language** level that scale to all applications.

Our Research Goals

1. Address energy efficiency through rigorous use of **standby**.
2. Target **constrained** embedded architectures that form the IoT.
3. Provide standby mechanisms at the **programming language** level that scale to all applications.
4. Support **transparent**/non-intrusive standby mechanisms that reduce barriers of adoption.

Research Non-Goals

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)
 - *Don't take advantage of standby modes (goal 1)*

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)
 - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
 - Low-power listening
 - Small periods of duty cycles

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)
 - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
 - Low-power listening
 - Small periods of duty cycles
 - *Only for networked parts, not automatic (goals 3 & 4)*

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)
 - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
 - Low-power listening
 - Small periods of duty cycles
 - *Only for networked parts, not automatic (goals 3 & 4)*
- Complex Hardware Architectures
 - Microprocessors, MMU, OS-based, Smartphones

Research Non-Goals

- Adaptive Computing
 - QoS (e.g., resolution, frame rate, accuracy)
 - Behavior (e.g., switch UI, disable functionalities)
 - *Don't take advantage of standby modes (goal 1)*
- Energy-Aware Network Protocols
 - Low-power listening
 - Small periods of duty cycles
 - *Only for networked parts, not automatic (goals 3 & 4)*
- Complex Hardware Architectures
 - Microprocessors, MMU, OS-based, Smartphones
 - *Not constrained embedded platforms (goal 2)*

General Approach

(standby, constrained, programming language, transparent)

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
- Infer deepest sleeping mode

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
- Infer deepest sleeping mode
- Put device to sleep

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
- Infer deepest sleeping mode
- Put device to sleep
- Only awake from interrupts

A Motivating Example

A Motivating Example

Every second, read sensor value and broadcast it.

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

- Céu

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

- Céu

```
loop do  
    await 1s;  
    int v = await AnalogRead();  
    await RadioWrite(v);  
end
```

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

- Céu

```
loop do  
    await 1s;  
    int v = await AnalogRead();  
    await RadioWrite(v);  
end
```

A Motivating Example

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

- Céu

```
loop do  
    await 1s;  
    int v = await AnalogRead();  
    await RadioWrite(v);  
end
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Let's add concurrency...

Every second, read sensor value and broadcast it.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
while (1) {  
  delay(1000);  
  int v = analogRead();  
  radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

```
while (1) {
    delay(1000);
    int v = analogRead();
    radioWrite(v);
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();  
while (1) {  
    if (radioAvail())  
        break;  
    uint32_t cur = millis();  
    if (cur > prv+1000) {  
        prv = cur;  
        int v = analogRead();  
        radioWrite(v);  
    }  
}
```

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();  
while (1) {  
    if (radioAvail())  
        break;  
    uint32_t cur = millis();  
    if (cur > prv+1000) {  
        prv = cur;  
        int v = analogRead();  
        radioWrite(v);  
    }  
}
```

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

```
while (1) {
    delay(1000);
    int v = analogRead();
    radioWrite(v);
}
```


Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();  
while (1) {  
    if (radioAvail())  
        break;  
    uint32_t cur = millis();  
    if (cur > prv+1000) {  
        prv = cur;  
        int v = analogRead();  
        radioWrite(v);  
    }  
}
```

```
while (1) {  
    delay(1000);  
    int v = analogRead();  
    radioWrite(v);  
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

```
par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end
```

Let's add concurrency...

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

```
par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end
```

Standby Considerations

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

```
par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end
```

Standby Considerations

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

```
par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end
```

Standby Considerations

Every second, read sensor value and broadcast it.

Stop immediately as soon as a message arrives.

- Arduino

```
uint32_t prv = millis();
while (1) {
    if (radioAvail())
        break;
    uint32_t cur = millis();
    if (cur > prv+1000) {
        prv = cur;
        int v = analogRead();
        radioWrite(v);
    }
}
```

- Céu

```
par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end
```


Céu in a Nutshell

Céu in a Nutshell

- Reactive
 - environment in control: *events*

Céu in a Nutshell

- Reactive
 - environment in control: *events*
- Imperative
 - sequences, loops, assignments

Céu in a Nutshell

- Reactive
 - environment in control: *events*
- Imperative
 - sequences, loops, assignments
- Concurrent
 - multiple lines of execution: *trails*

Céu in a Nutshell

- Reactive
 - environment in control: *events*
- Imperative
 - sequences, loops, assignments
- Concurrent
 - multiple lines of execution: *trails*
- Synchronous (based on Esterel)
 - trails synchronize at each external event

Céu in a Nutshell

- Reactive
 - environment in control: *events*
- Imperative
 - sequences, loops, assignments
- Concurrent
 - multiple lines of execution: *trails*
- Synchronous (based on Esterel)
 - trails synchronize at each external event
 - **trails are always awaiting**

Céu in a Nutshell

- Reactive
 - environment in control: *events*
- Imperative
 - sequences, loops, assignments
- Concurrent
 - multiple lines of execution: *trails*
- Synchronous (based on Esterel)
 - trails synchronize at each external event
 - **trails are always awaiting**
- Deterministic
 - always yields the same outcome for a given timeline

In Practice...

In Practice...

- Hardware infrastructure

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
 - Rewrite existing IoT applications in Céu

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
 - Rewrite existing IoT applications in Céu
 - Time to rewrite, Coding “aesthetics”, Energy consumption

In Practice...

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
 - Rewrite existing IoT applications in Céu
 - Time to rewrite, Coding “aesthetics”, Energy consumption


```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```



```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
  ADMUX = 0x40|(A0&0x07);
  bitSet(ADCSRA, ADIE);
  bitSet(ADCSRA, ADSC);
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
  ADMUX = 0x40|(A0&0x07);
  bitSet(ADCSRA, ADIE);
  bitSet(ADCSRA, ADSC);
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
  ADMUX = 0x40|(A0&0x07);
  bitSet(ADCSRA, ADIE);
  bitSet(ADCSRA, ADSC);
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

```
output void ADC_REQUEST do
  ADMUX = 0x40|(A0&0x07);
  bitSet(ADCSRA, ADIE);
  bitSet(ADCSRA, ADSC);
end
```

```
input ADC_vect_num do
  bitClear(ADCSRA, ADIE);
  emit ADC_DONE(ADC);
end
```

```
code AnalogRead (void) -> int do
  PM_SET(PM_ADC, 1);
  do finalize with
    PM_SET(PM_ADC, 0);
  end
  emit ADC_REQUEST;
  int value = await ADC_DONE;
  escape value;
end
```

```

par/or do
    await RadioAvail();
with
    loop do
        await 1s;
        int v = await AnalogRead();
        await RadioWrite(v);
    end
end

```

```

void pm_sleep (void) {
    if (PM_GET(PM_TIMER1)) {
        sleep_1(<...>)
    } else if (PM_GET(PM_ADC)) {
        sleep_2(<...>);
    } else {
        sleep_3(<...>);
    }
}

```

```

output void ADC_REQUEST do
    ADMUX = 0x40|(A0&0x07);
    bitSet(ADCSRA, ADIE);
    bitSet(ADCSRA, ADSC);
end

```

```

input ADC_vect_num do
    bitClear(ADCSRA, ADIE);
    emit ADC_DONE(ADC);
end

```

```

code AnalogRead (void) -> int do
    PM_SET(PM_ADC, 1);
    do finalize with
        PM_SET(PM_ADC, 0);
    end
    emit ADC_REQUEST;
    int value = await ADC_DONE;
    escape value;
end

```

```
par/or do
```

```
    await RadioAvail();
```

```
with
```

```
    loop do
```

```
        await 1s;
```

```
        int v = await AnalogRead();
```

```
        await RadioWrite(v);
```

```
    end
```

```
end
```

```
void pm_sleep (void) {
```

```
    if (PM_GET(PM_TIMER1)) {
```

```
        sleep_1(<...>)
```

```
    } else if (PM_GET(PM_ADC)) {
```

```
        sleep_2(<...>);
```

```
    } else {
```

```
        sleep_3(<...>);
```

```
    }
```

```
}
```

```
}
```

```
output void ADC_REQUEST do
```

```
    ADMUX = 0x40|(A0&0x07);
```

```
    bitSet(ADCSRA, ADIE);
```

```
    bitSet(ADCSRA, ADSC);
```

```
end
```

```
input ADC_vect_num do
```

```
    bitClear(ADCSRA, ADIE);
```

```
    emit ADC_DONE(ADC);
```

```
end
```

```
code AnalogRead (void) -> int do
```

```
    PM_SET(PM_ADC, 1);
```

```
    do finalize with
```

```
        PM_SET(PM_ADC, 0);
```

```
    end
```

```
    emit ADC_REQUEST;
```

```
    int value = await ADC_DONE;
```

```
    escape value;
```

```
end
```

General Approach

(standby, constrained, programming language, transparent)

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution

```
par/or do
  await RadioAval();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```


General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
 - Céu has a semantics amenable to analysis
- Put device to sleep

```
par/or do
  await RadioAval();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
 - Céu has a semantics amenable to analysis
- Put device to sleep

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
 - Céu has a semantics amenable to analysis
- Put device to sleep
 - Céu has an energy-aware runtime
- Only awake from interrupts

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

General Approach

(standby, constrained, programming language, transparent)

- Enforce idle states of execution
 - Céu enforces a reactive model of execution
- Infer deepest sleeping mode
 - Céu has a semantics amenable to analysis
- Put device to sleep
 - Céu has an energy-aware runtime
- Only awake from interrupts
 - Céu provides interrupt service routines (ISRs)

```
par/or do
  await RadioAvail();
with
  loop do
    await 1s;
    int v = await AnalogRead();
    await RadioWrite(v);
  end
end
```

Transparent Standby for Low-Power, Resource-Constrained Embedded Systems

A Programming Language-Based Approach



Francisco Sant'Anna
francisco@ime.uerj.br
@_fsantanna

Standby Efficiency for IoT

Standby Efficiency for IoT

- 15 billion “traditional” network-connected devices in 2015 (e.g., mobile phones & smart TVs).

Standby Efficiency for IoT

- 15 billion “traditional” network-connected devices in 2015 (e.g., mobile phones & smart TVs).
- 75 billion by 2025 with the IoT (e.g., smart bulbs & fitness wearables). [2016]

Standby Efficiency for IoT

- 15 billion “traditional” network-connected devices in 2015 (e.g., mobile phones & smart TVs).
- 75 billion by 2025 with the IoT (e.g., smart bulbs & fitness wearables). [2016]
- **Most energy consumed in standby mode.**

Standby Efficiency for IoT

- 15 billion “traditional” network-connected devices in 2015 (e.g., mobile phones & smart TVs).
- 75 billion by 2025 with the IoT (e.g., smart bulbs & fitness wearables). [2016]
- **Most energy consumed in standby mode.**
- Network standby is one of the six fronts on IEA/G20's Energy Efficiency Action Plan
 - <https://www.iea-4e.org/projects/g20>

Related Publications

Related Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
 - *Safe System-level Concurrency on Resource-Constrained Nodes*

Related Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
 - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
 - *The Design and Implementation of the Synchronous Language Céu*

Related Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
 - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
 - *The Design and Implementation of the Synchronous Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
 - *A Memory-Bounded, Deterministic and Terminating Semantics
for the Synchronous Programming Language Céu*

Related Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
 - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
 - *The Design and Implementation of the Synchronous Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
 - *A Memory-Bounded, Deterministic and Terminating Semantics
for the Synchronous Programming Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
 - *WIP: Transparent Standby for Low-Power, Resource-Constrained*

Related Publications

- SenSys'13, ACM Embedded Networked Sensor Systems
 - *Safe System-level Concurrency on Resource-Constrained Nodes*
- TECS'17, ACM Transactions on Embedded Computing Systems
 - *The Design and Implementation of the Synchronous Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
 - *A Memory-Bounded, Deterministic and Terminating Semantics
for the Synchronous Programming Language Céu*
- LCTES'18, ACM Conference on Languages, Compilers, and Tools for Embedded Systems
 - *WIP: Transparent Standby for Low-Power, Resource-Constrained*

Cooperation Opportunities

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
 - Rewrite existing IoT applications in Céu
 - Time to rewrite

Cooperation Opportunities

- Hardware infrastructure
 - Off-the-shelf Arduinos (ATMega328, Cortex-M0)
- Software infrastructure
 - Implement an energy-aware runtime for Céu
 - Rewrite device drivers in Céu (timers, ADC, Radio)
- Applications
 - Rewrite existing IoT applications in Céu
 - Time to rewrite