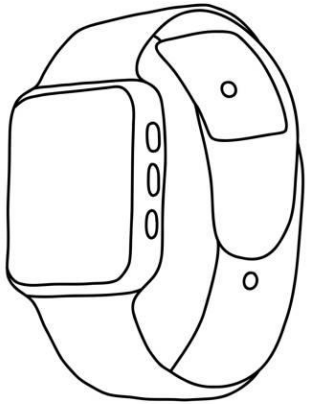
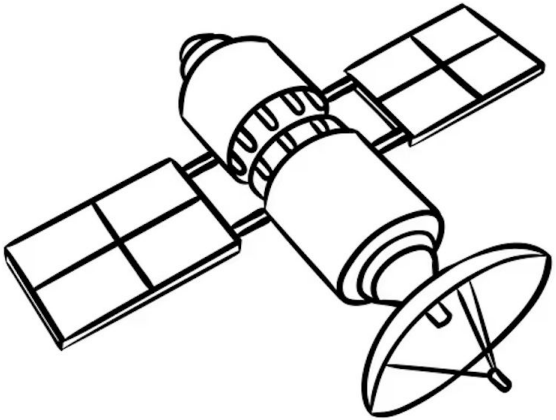
Leveraging Types and Typestate for Peripheral Cost Analysis

Sai Divvela (UMD), Tyler Potyondy (UCSD),
Mohak Vaswani (UCSD), Milijana Surbatovich (UMD)

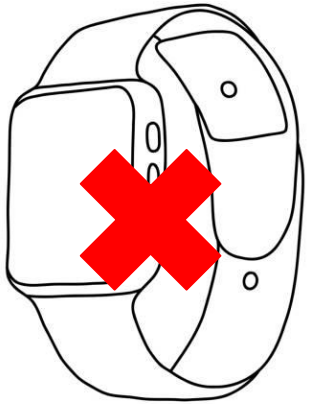
Embedded Systems are Everywhere

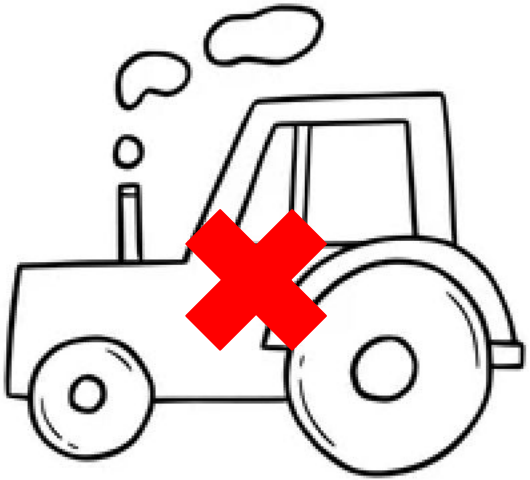


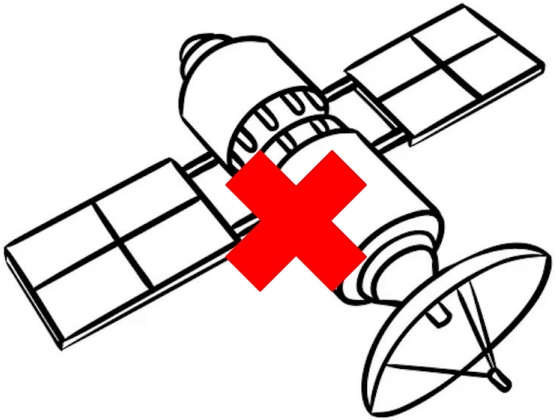


Peripheral Usage

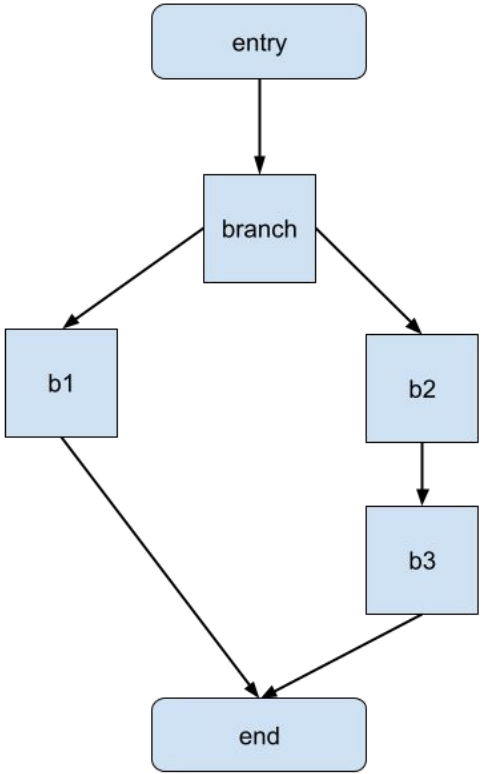
The need for energy efficiency



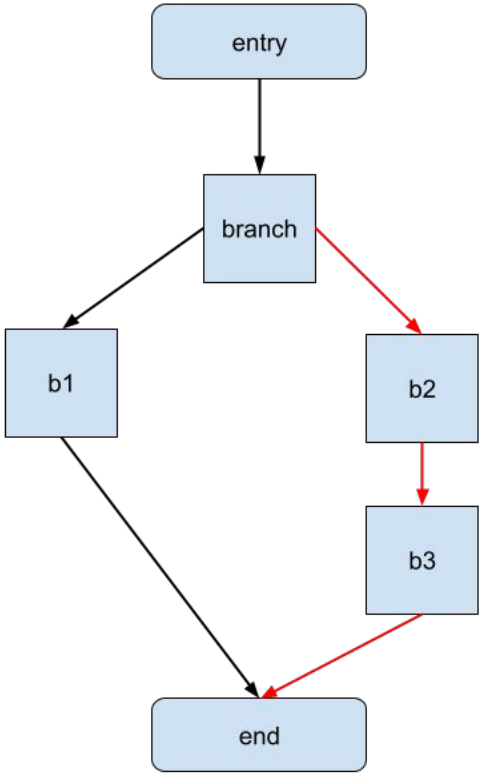




Worst Case Energy Consumption

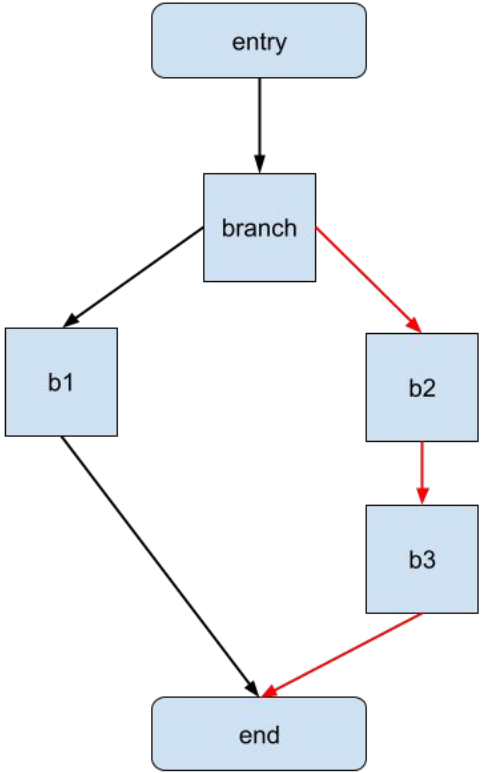


Worst Case Energy Consumption

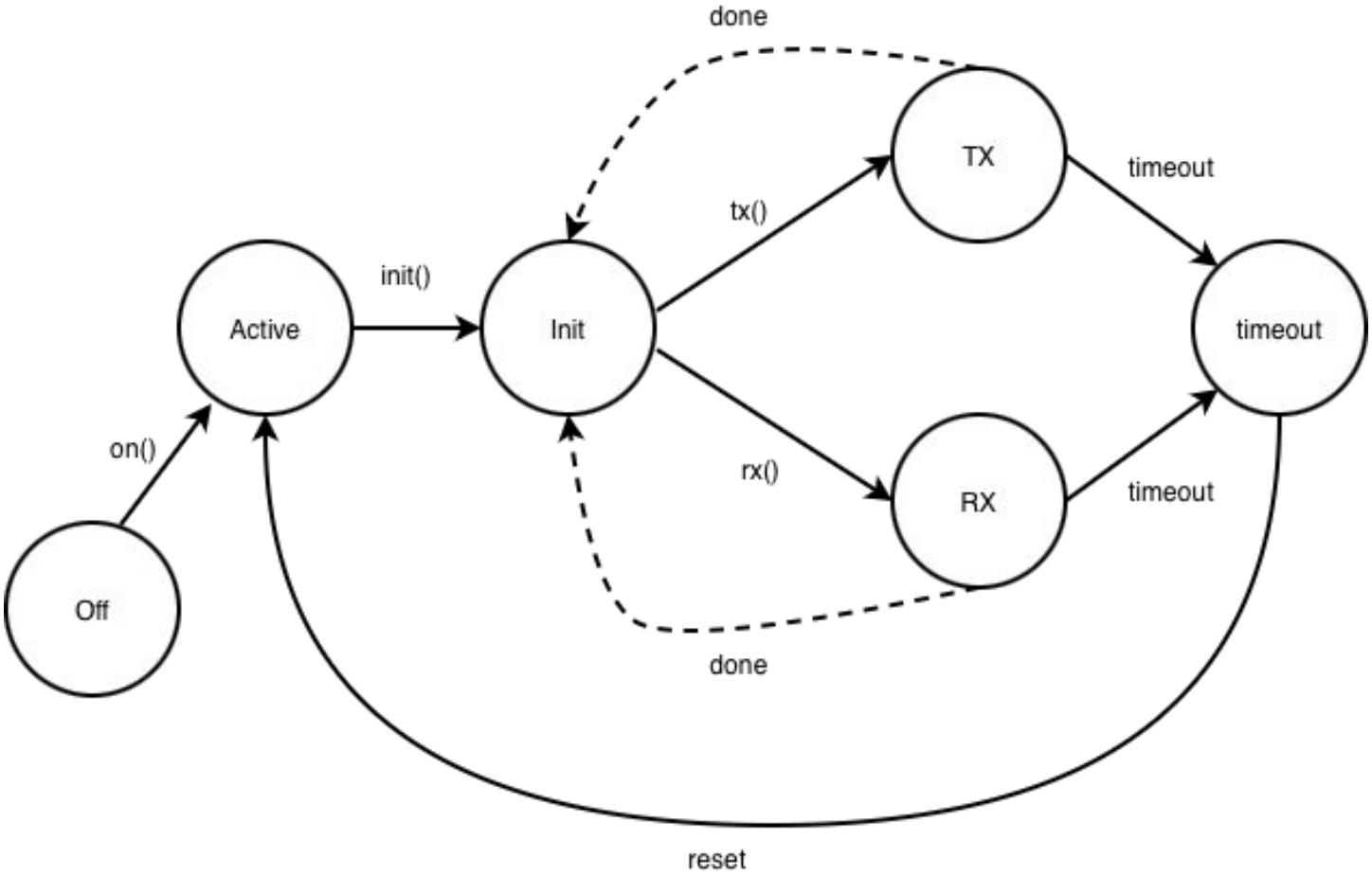


Calculating Worst Case Energy Consumption

1. Measurement-based strategies
2. Static analysis based on path enumeration



The issue with Peripherals



Energy Analysis Approaches

Measurement-based approaches that can be difficult to incorporate

vs

Static Analyses that may not capture all costs

Can we have a tool that does both?

Cost Categories

1. Main Board Execution
2. Peripheral Operations
3. Peripherals active in the **background**

Cost Categories

1. Main Board Execution
2. Peripheral Operations
3. Peripherals active in the background

Code	I2C State
`transition(i2C, Tx);`	? $\rightarrow$ Tx
`i2C.transmit();`	Tx
`some_other_func();`	Tx
`i2C.stop();`	Tx $\rightarrow$ Stop

Our approach: Combining typestate and AARA

Typestate helps to track peripheral state precisely

Automatic Amortized Resource Analysis (AARA) enables robust cost bounds

We report costs **symbolically** to aid programmer understanding

Challenges

Major Challenges:

- Tracking Peripheral state effectively
- Providing accurate cost analysis
- Supporting advanced features such as asynchronous peripherals

Types of Cost Expressions

- C - a fixed cost of executing an expression
- $C(p : st)$ - the cost depends on a peripheral p being in state st
- $C(p : (st, st'))$ - the cost depends on a peripheral p transitioning from state st to state st'

A running example

```
fn log_data(temp) -> u64 {
 transition(temp, Measure);
 let res = temp.measure();
 let t = 0;
 if(res <= 65) {
   t = 1;
 }
 transition(temp, Idle);
 return t;
}
```

A running example

```
fn log_data(temp) -> u64 {
 transition(temp, Measure);
 let res = temp.measure();
 let t = 0;
 if(res <= 65) {
   t = 1;
 }
 transition(temp, Idle);
 return t;
}
```

C(temp: (???, Measure))

C_m + C(temp: Measure)

C_v + C(temp: Measure)

max(C_v, 0) + C(temp: Measure)

C(temp: (Measure, Idle))

A running example

```
fn log_data(temp) -> u64 {
 transition(temp, Measure);
 let res = temp.measure();
 let t = 0;
 if(res <= 65) {
   t = 1;
 }
 transition(temp, Idle);
 return t;
}
```

C(temp: (???, Measure))

C_m + C(temp: Measure)

C_v + C(temp: Measure)

max(C_v, 0) + C(temp: Measure)

C(temp: (Measure, Idle))

Peripheral State Tracking

Type system does not know state of `temp` until `log_data()` is called

```
fn main() {
 transition(temp, Idle);
 let t = log_data(temp);
}
```

```
fn main() {
 transition(temp, Idle);
 transition(temp, Measure);
 let t = log_data(temp);
}
```

Peripheral State Tracking

Type system does not know state of `temp` until `log_data()` is called

```
fn main() {
 transition(temp, Idle);
 let t = log_data(temp);
}
```

$$C(\text{temp}: (\text{Off}, \text{Idle}))$$

$$C_1[?? = \text{Idle}]$$

Peripheral State Tracking

Type system does not know state of `temp` until `log_data()` is called

```
fn main() {
 transition(temp, Idle);
 transition(temp, Measure);
 let t = log_data(temp);
}
```

$C(\text{temp: (Off, Idle)})$

$C(\text{temp: (Idle, Measure)})$

$C_1[?? = \text{Measure}]$

State Inference

Both of these examples are valid!
The usage determines the cost and the peripheral state

```
fn main() {
 transition(temp, Idle);
 let t = log_data(temp);
}
```

```
fn main() {
 transition(temp, Idle);
 transition(temp, Measure);
 let t = log_data(temp);
}
```

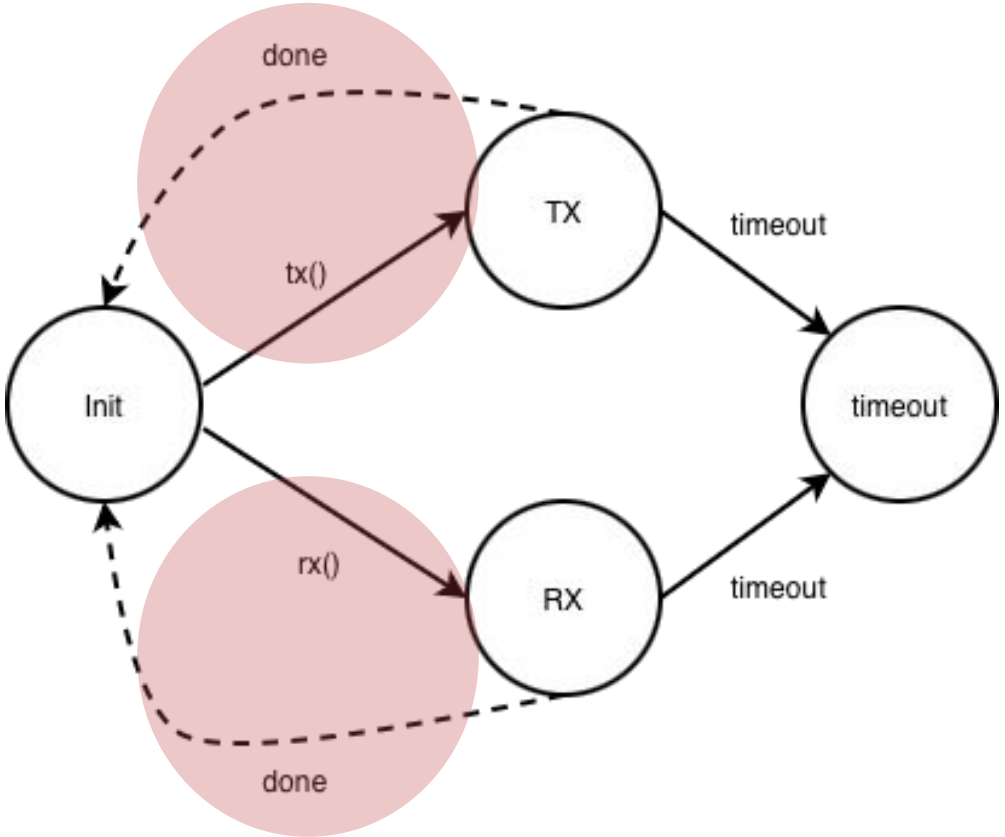
We need to **infer** the state of each peripheral

Asynchronicity?

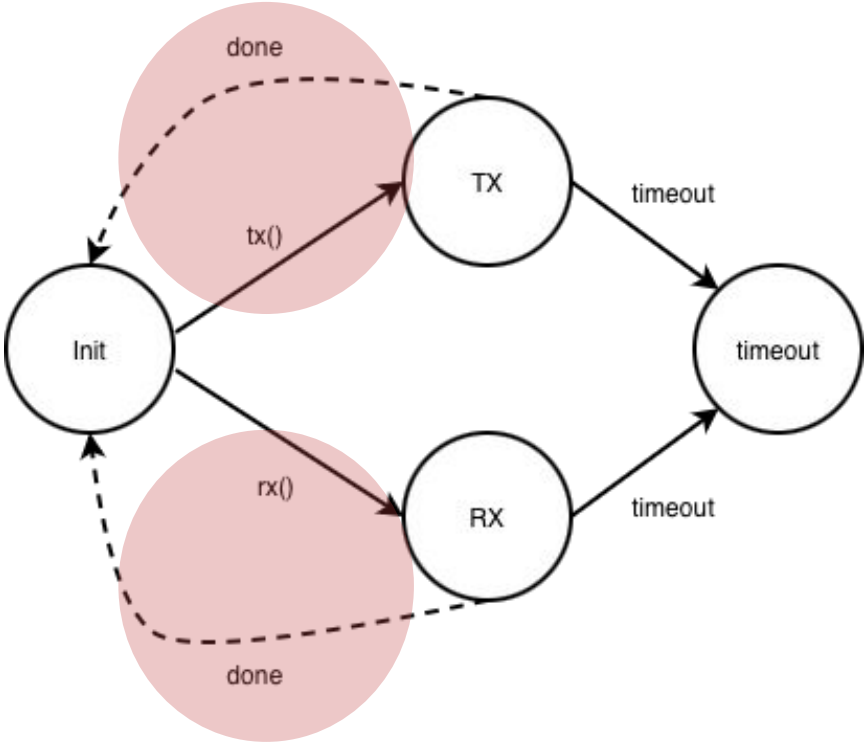
How do we deal with asynchronous peripherals?

We no longer have a 1-1 correspondence between hardware and software

Interrupt driven transitions



Interrupt driven transitions



We aim to use an enhancement of typestate to support this

Soundness

Soundness is inspired by soundness theorem for AARA systems

Type Soundness (informal)

Given a program P, with an inferred cost C, then executing P gives a cost C' and $C \geq C'$

Next Steps

This is still work in progress! Next steps include:

1. Flesh out support for asynchronous peripherals
2. Finish implementing this analysis
3. Integrate with real drivers
4. Prove type soundness

- Want energy efficient embedded systems
- Typestate + AARA provides accurate cost for stateful peripherals
- Report costs symbolically for programmer understanding

Thank you!