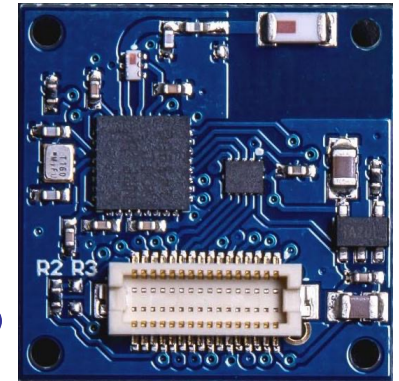
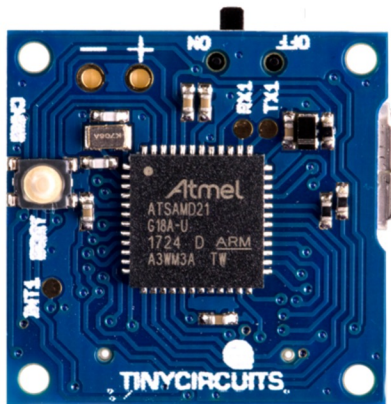


# CSE190 Fall 2023

## Lecture 8

# Debugging Embedded Systems



Wireless Embedded Systems

Aaron Schulman

# Qucik C development note

## **Volatile variables**

```
volatile uint32_t a = 0;
```

```
void TIM2_IRQHandler() {  
    a = 1;  
}
```

```
int main() {  
    if (a == 1) {  
        // Will this code run?  
    }  
}
```

# Hardware is hard.

Who do you trust when things are going wrong?

- Your software?
- Your users?
- Your test equipment?
  - Your test equipment configuration?
- The device designer?
  - The datasheet authors?
- The circuit board designer?
  - The manufacturer and assembler?
- ...
- Physics?

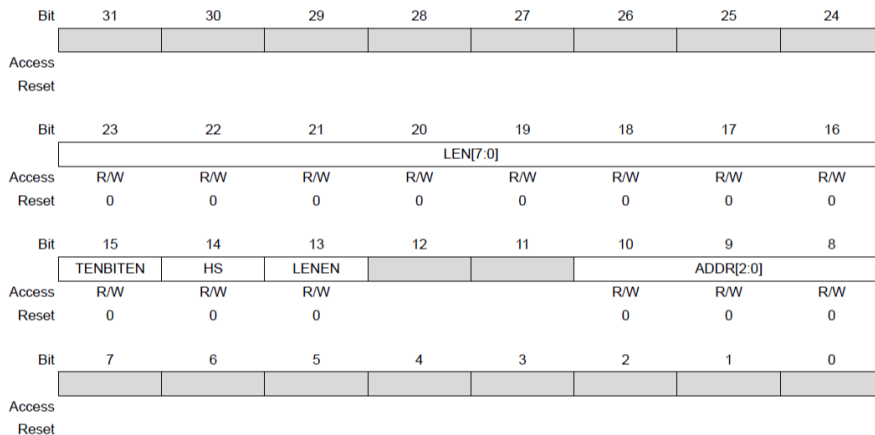
# Data sheet errors are common

Which one of these is correct?  
 SAM21 data sheet in 2017, and 2018 for I2C

2017

28.10.9 Address

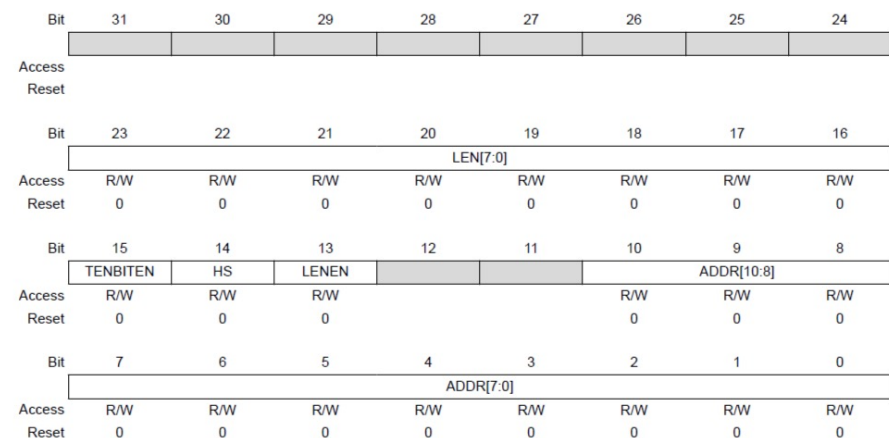
**Name:** ADDR  
**Offset:** 0x24  
**Reset:** 0x0000  
**Property:** Write-Synchronized



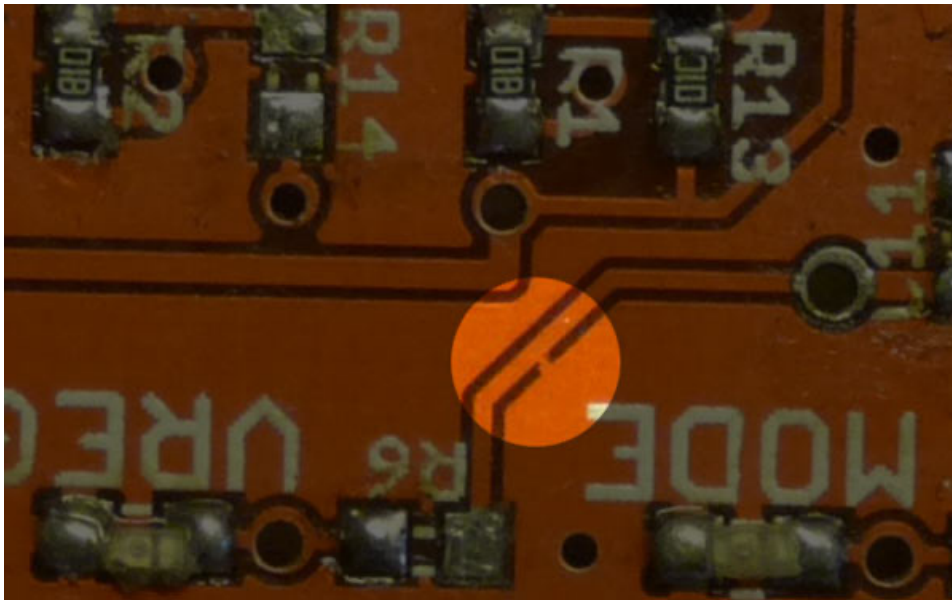
2018

28.10.9 Address

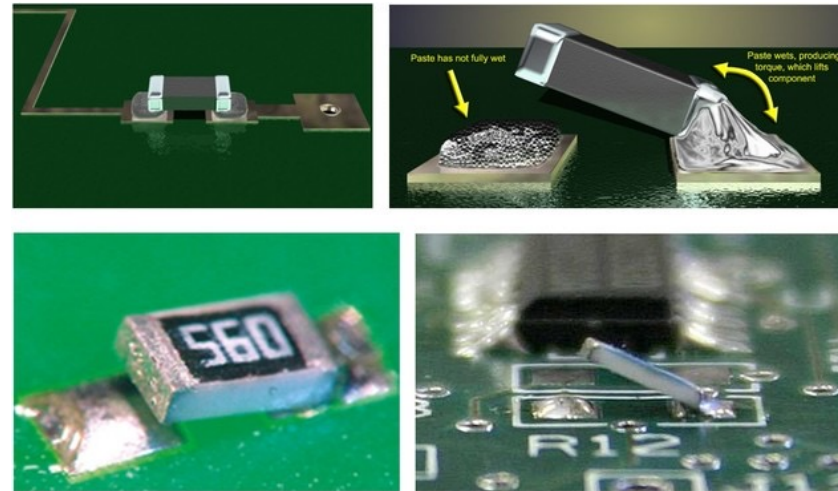
**Name:** ADDR  
**Offset:** 0x24  
**Reset:** 0x0000  
**Property:** Write-Synchronized



# Manufacturing errors are common



Circuit board manufacturing fault (signal shorted to GND)



PCB assembly fault ("tombstoning") resistors

# When all else fails, the bootloader can save the day!

**When your software has a bug in it that causes the processor to hang or crash, you can't do anything!**

- Programming over USB requires the processor to be responsive (a reset message is sent over USB at the beginning of the programming)
- The bootloader is a small bit of code run at boot time that doesn't get overwritten
  - It has just enough code in it to reflash over USB
  - Normally it just jumps into your code, but you can force it to wait for a program flash switching the power on while you hold the reset button.

# Even one LED can be critical for debugging

## **Think of an LED as a breakpoint (very efficient!)**

- Turn it on before a line of code
  - Lets you know you reached that line
- Turn it off after a line of code
  - Lets you know you exited that code segment
- Toggle it each time a repeated event is called
  - This can show you visually how often the event happens

# Need some way to introspect code: printf() is essential

## Think of prints like tracing + inspection (not efficient)

- Print variables you need to inspect
  - Register states (flags, sensor readings)
- Print when a specific function is run
  - Lets you know the order functions are run
  - Can you use printf for interrupt handlers?
  - `printf("Currently inside function: %s\n", __func__);`



# Hardware debugger pro/cons

**Ok for development, impossible after deployment**

- Excellent tool for building/testing new drivers
- Lets you pause/inspect memory of CPU
- Does not pause peripherals, they keep running
  - MMIO inspection will be current peripheral state