Carnegie Mellon University Qatar

15-348: Embedded Systems Lecture 8

Fall 2022

Ryan Riley
(based on original slides by Saquib Razak)

1

1

Where are we?

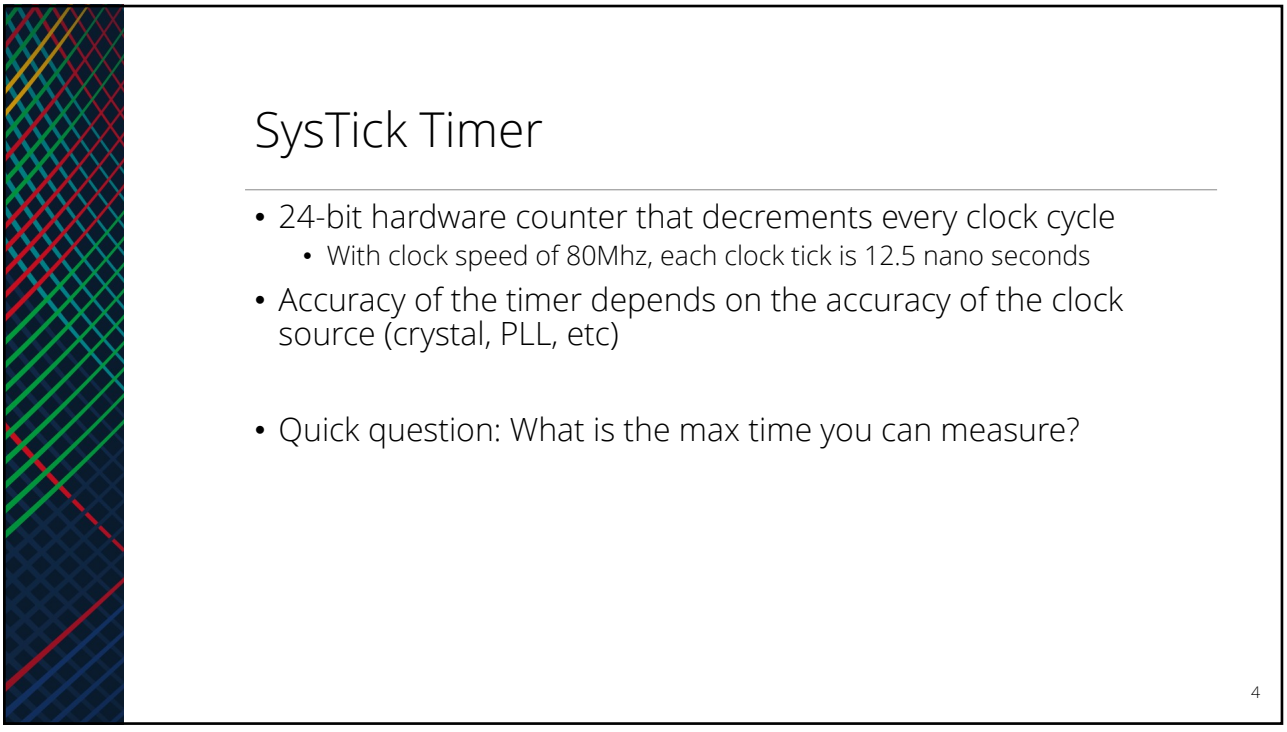
- Recently
  - Assembly
  - State Machines
- Today
  - The SysTick timer
  - Timing and Interfacing
  - Interrupts pre-game show

2

2


SysTick
3

3

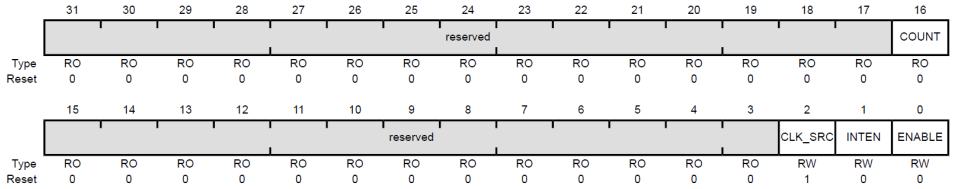

SysTick Timer
• 24-bit hardware counter that decrements every clock cycle
• With clock speed of 80Mhz, each clock tick is 12.5 nano seconds
• Accuracy of the timer depends on the accuracy of the clock source (crystal, PLL, etc)
• Quick question: What is the max time you can measure?
4

4

Using the Timer (NVIC_ST_CTRL_R)

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type RW, reset 0x0000.0004



- Count – 1 means SysTick timer has counted to 0
- CLK_SRC
  - 0 PIOSC divided by 4 (16MHz/4)
  - 1 System clock (80Mhz if `PLLInit()` was called)
- INTEN – Interrupts enabled (later)
- ENABLE – Enable this timer

5

5

Using the Timer (NVIC_ST_CURRENT_R)

- Counts down every clock cycle
- COUNT bit is set in CTRL register when this counter goes from 1 to 0
- Write any value to clear this register and COUNT bit
  - Side note: Reading the CTRL register also clears the COUNT bit
- The next clock cycle after reaching 0 will reload this counter with value in NVIC_ST_RELOAD_R

6

6

Using the Timer (NVIC_ST_RELOAD_R)

- 24-bit value to reload the timer with after it rolls-over from 0

7

7

timer.c

```
void SystickInit(){
    // Disable timer
    NVIC_ST_CTRL_R = 0;
    // The largest possible reload value
    NVIC_ST_RELOAD_R = 0x00FFFFFF;
    // Set the current countdown to 0 (so first
    // cycle after timer starts causes a reload)
    NVIC_ST_CURRENT_R = 0;
    // Enable the timer and set to the source to
    // the system clock
    NVIC_ST_CTRL_R = 0x00000005;
}
```

8

8

timer.c

```
// Wait a specific number of 12.5 ns clock cycles
void SysTick_Wait(uint32_t delay){
     // Set the reload value
     NVIC_ST_RELOAD_R = delay - 1;
     // Set the current countdown to 0 (so first
     // cycle after timer starts causes a reload)
     NVIC_ST_CURRENT_R = 0;
     // Wait until the count bit is set
     while ((NVIC_ST_CTRL_R&0x00010000)==0){}
 }
```

9

9

timer.c

```
void SysTick_Wait10ms(uint32_t delay)
{
    uint32_t i;
    for (i = 0; i < delay; i++)
        SysTick_Wait(800000);
}
void SysTick_Wait1microsec(uint32_t delay)
{
       uint32_t i;
       for (i = 0; i < delay; i++)
           SysTick_Wait(80);
}
```

10

10

Note on Timer:

- The more SysTick_Wait() calls you have, the less accurate your timing will be
  - Overhead of the loop and the function calls
- For 10ms, better to do one call to `Wait_10ms` then 10,000 calls to `Wait_1microsec`

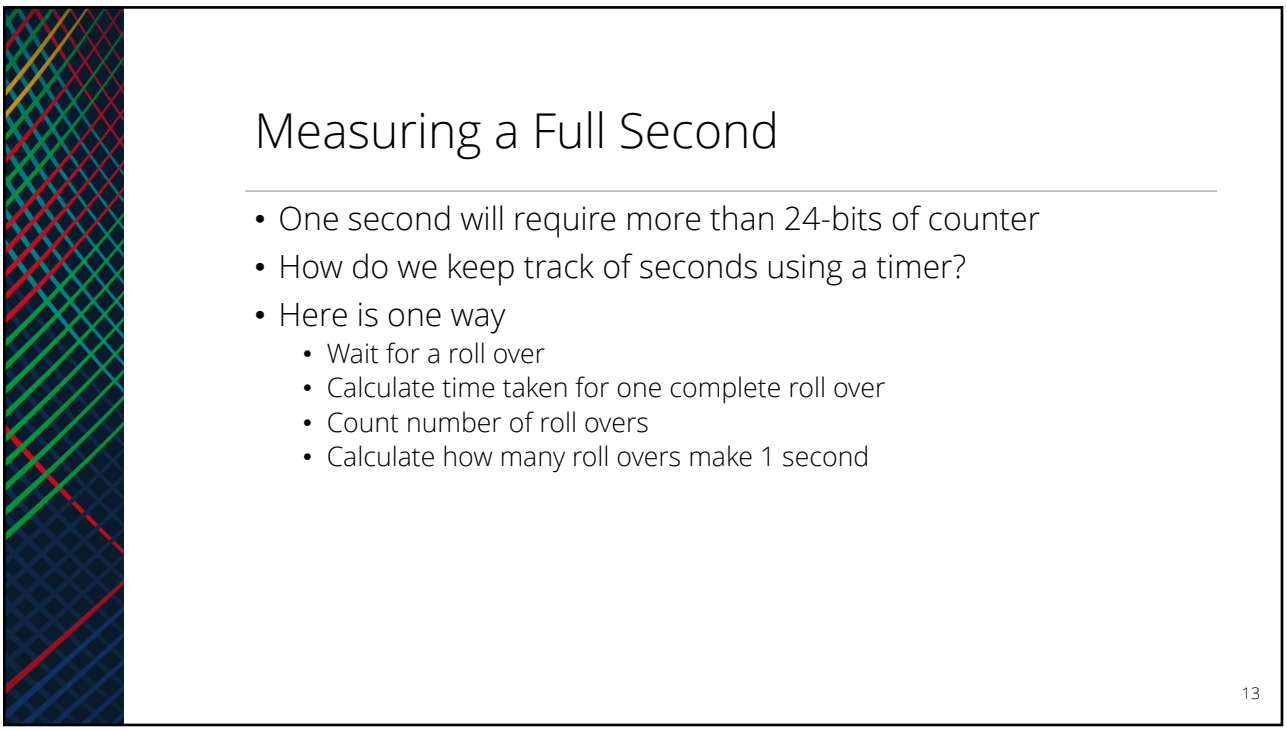
11

11

Longer Times With the Timer

12

12

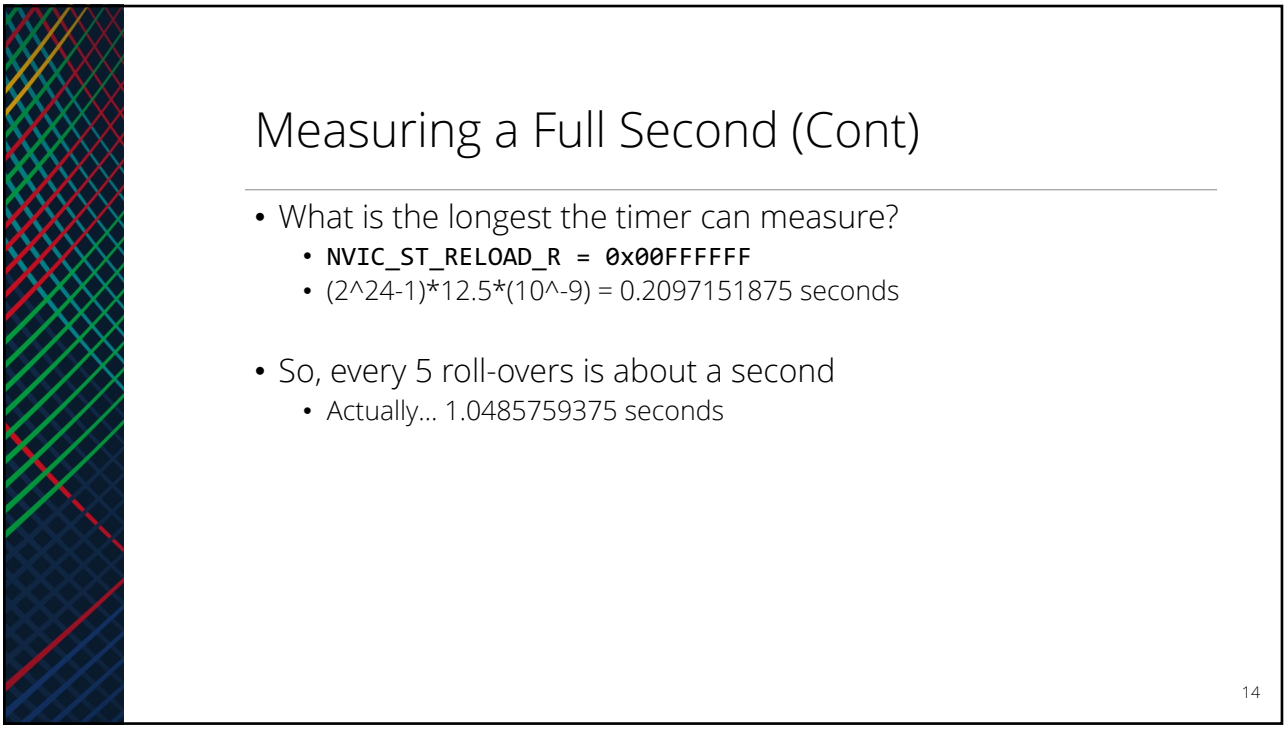


Measuring a Full Second

- One second will require more than 24-bits of counter
- How do we keep track of seconds using a timer?
- Here is one way
  - Wait for a roll over
  - Calculate time taken for one complete roll over
  - Count number of roll overs
  - Calculate how many roll overs make 1 second

13

13



Measuring a Full Second (Cont)

- What is the longest the timer can measure?
  - `NVIC_ST_RELOAD_R = 0x00FFFFFF`
  - (2^24-1)*12.5*(10^-9) = 0.2097151875 seconds
- So, every 5 roll-overs is about a second
  - Actually… 1.0485759375 seconds

14

14

How Much Error Is That?

- What fraction do we lose every second?
  - 0.048576 seconds
- How much do we lose every hour?
  - 175 seconds
- Everyday?
  - 70 minutes

15

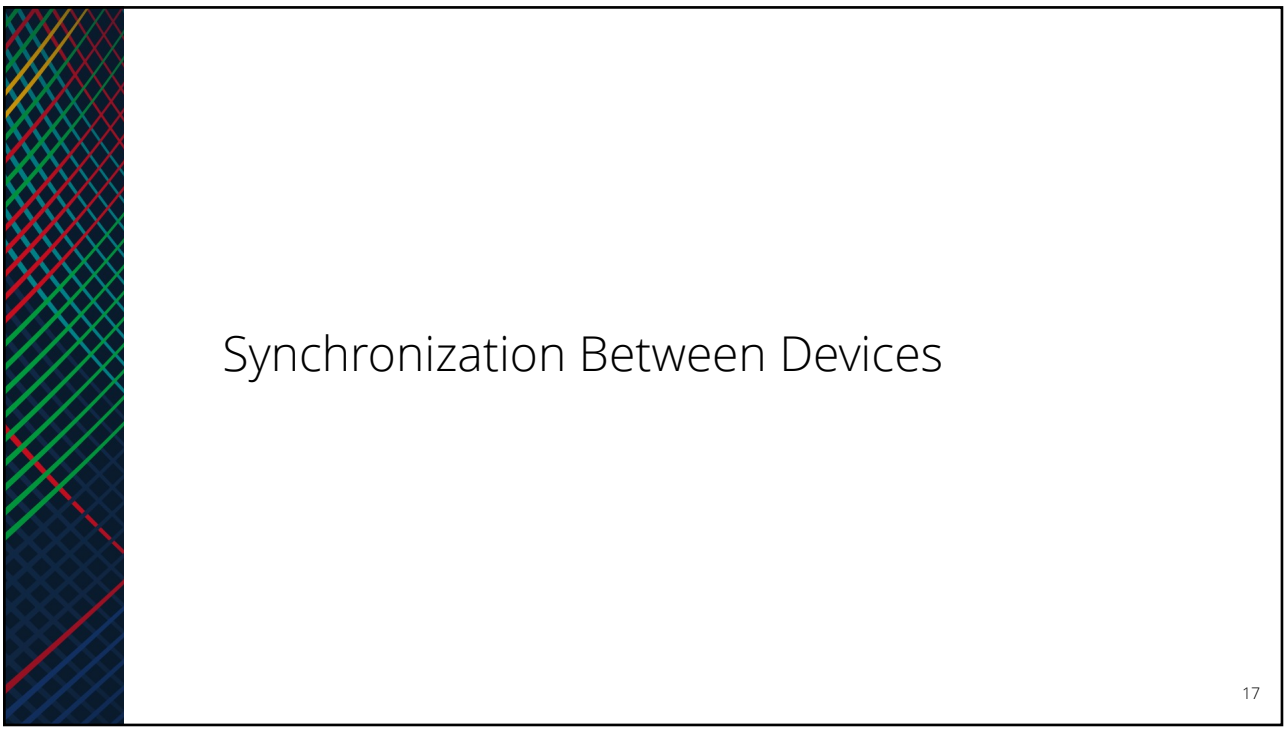
15

Can We Do Better?

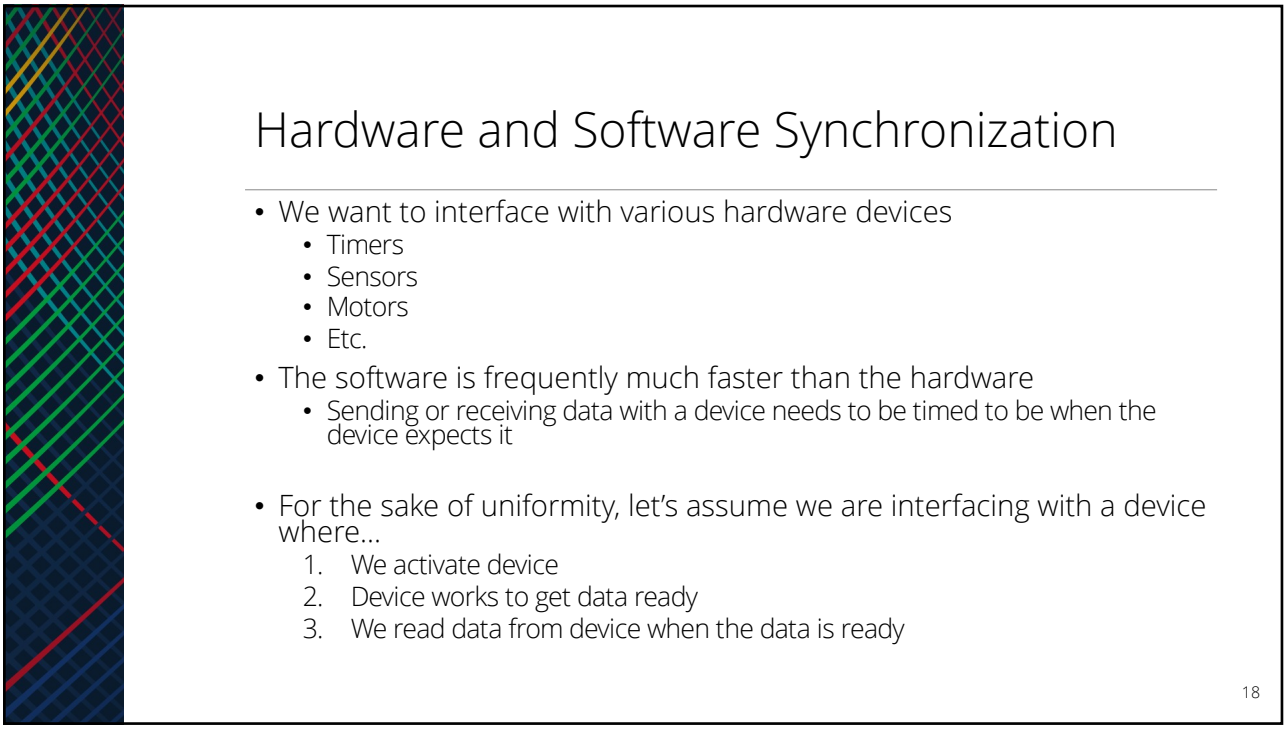
- Option 1:
  - Keep track of the error and correct for it
  - Complicated, but doable
- Option 2:
  - Pick a better RELOAD value
  - 0x00F42400 is exactly 0.2 seconds

16

16


Synchronization Between Devices
17

17


Hardware and Software Synchronization
• We want to interface with various hardware devices
• Timers
• Sensors
• Motors
• Etc.
• The software is frequently much faster than the hardware
• Sending or receiving data with a device needs to be timed to be when the device expects it
• For the sake of uniformity, let's assume we are interfacing with a device where…
1. We activate device
2. Device works to get data ready
3. We read data from device when the data is ready
18

18

Five Approaches to Synchronization

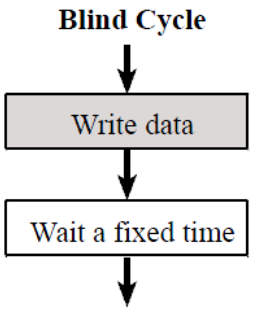
1. Blind Cycle counting
2. Busy Waiting
3. Interrupts
4. Periodic Polling
5. Direct Memory Access

19

19

1. Blind Cycle Counting

1. Activate device
2. Wait a fixed amount of time
3. Read results

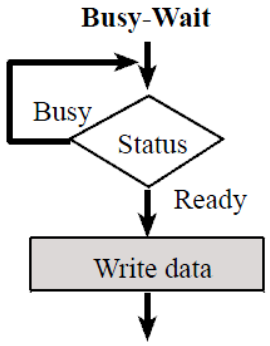


20

20

2. Busy-Waiting (Gadfly)

1. Activate Device
2. Loop: Ask device if it is finished yet
3. Read results



21

21

3. Interrupts

1. Activate device
2. Do other things until I am interrupted
3. Read results

We'll come back to this later

22

22

4. Periodic Polling

- A mix of busy-waiting and interrupts
- Do other things, but periodically check with the device to see if it is done

23

23

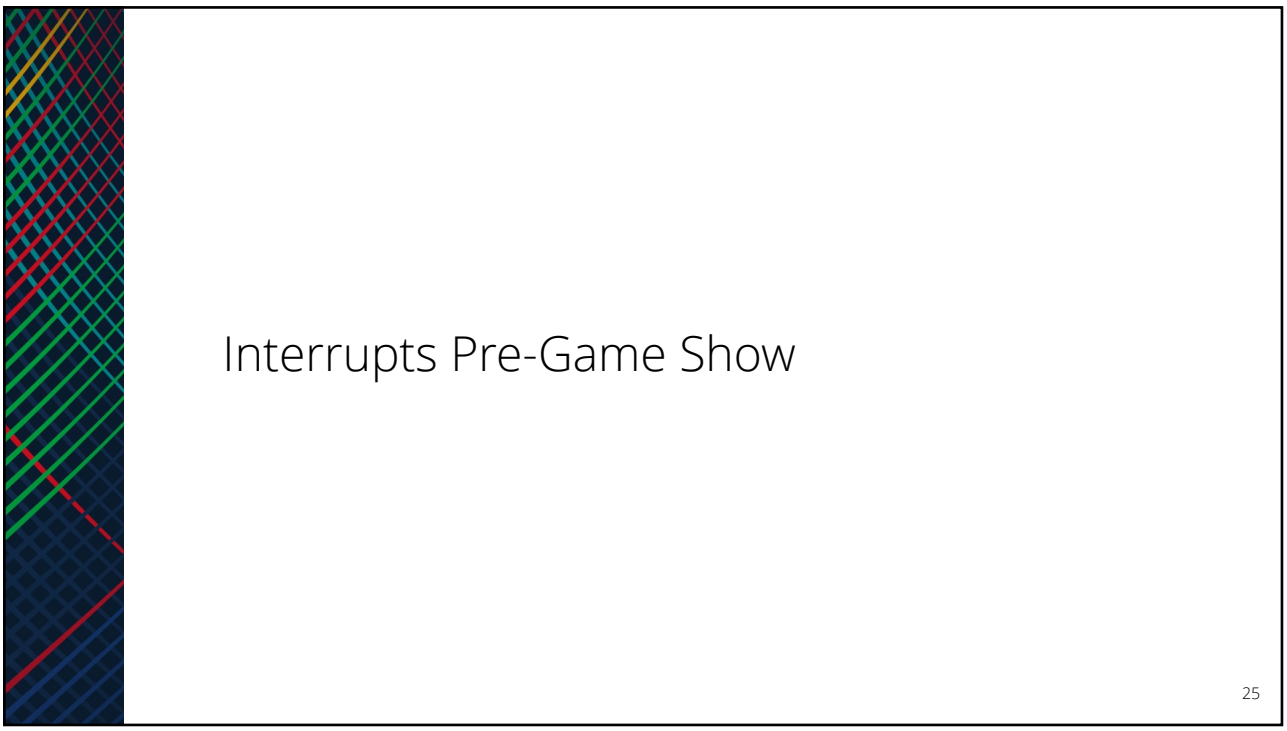
5. Direct Memory Access

1. Activate device
2. Device writes its results directly to RAM

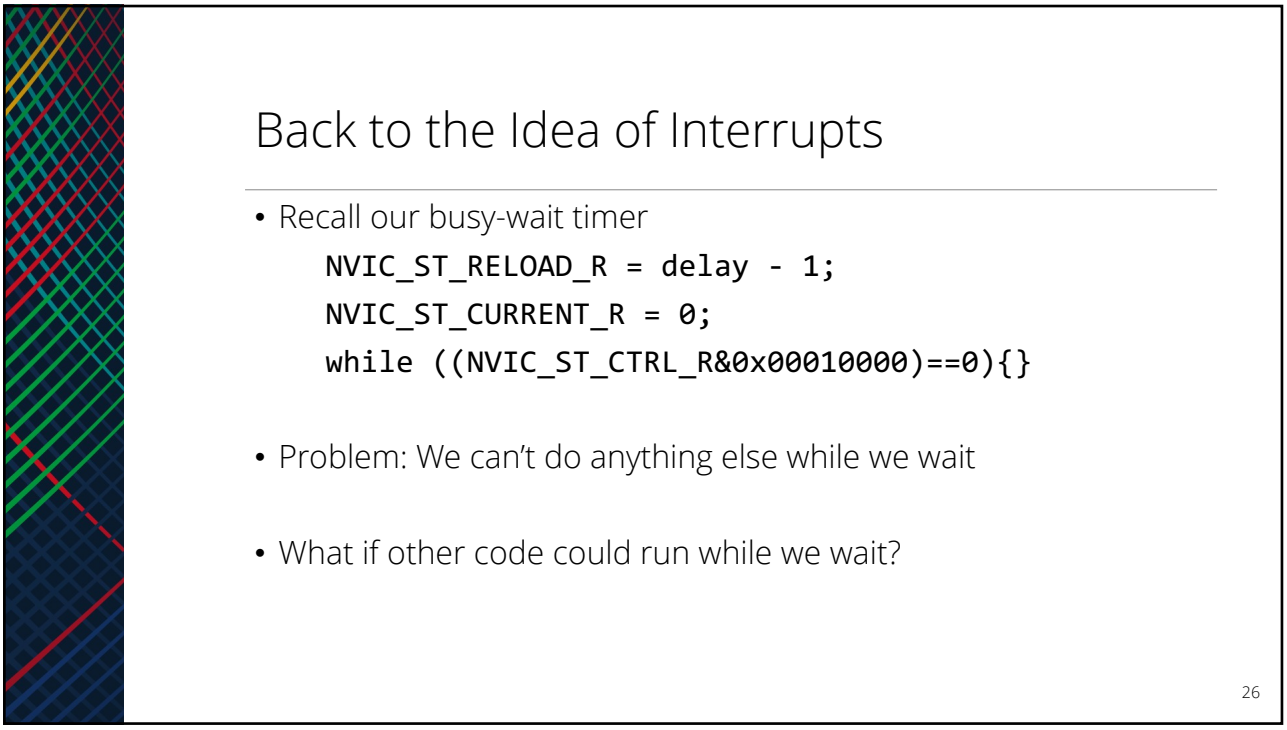
We won't do this in this course, but your computer does this frequently

24

24


Interrupts Pre-Game Show
25

25


Back to the Idea of Interrupts
• Recall our busy-wait timer
NVIC_ST_RELOAD_R = delay - 1;
NVIC_ST_CURRENT_R = 0;
while ((NVIC_ST_CTRL_R&0x00010000)==0){}
• Problem: We can't do anything else while we wait
• What if other code could run while we wait?
26

26

High-Level Idea

1. Write a function called an Interrupt Service Routine (ISR)
2. Inform the CPU that it should call the ISR automatically whenever the timer count rolls over
3. Start the timer, configure it with a RELOAD value of, say 1ms

How can we use this to improve our state machines?

27

27