EECS 373

Design of Microprocessor-Based Systems

Robert Dick
University of Michigan

Lecture 13: Review

21 February 2017

Slides inherited from Mark Brehob.

Outline

- Context and review
- Midterm evaluations
- Timers
- Serial buses
- Hazards
- Toolchain

Context and review

- Memory
  - Classes of volatile and nonvolatile memory.
  - Physical operating principles.
  - Internal structures.
  - Strengths and weaknesses.
- PCBs
  - How to design.
  - Techniques for noise immunity.
  - Power distribution.

Outline

- ~~Context and review~~
- Midterm evaluations
- Timers
- Serial buses
- Hazards
- Toolchain

Midterm evaluations: what to improve

- Homework: Some questions ambiguous.
- Lecture: Slower on examples.
- Slides: Avoid full sentences.

Lab 5 deadline, hours during break

- Extension: Lab 5 due 6pm on 6 March.
- 2pm-5pm on 3 or 4 March (Friday or Saturday).
- Decide.

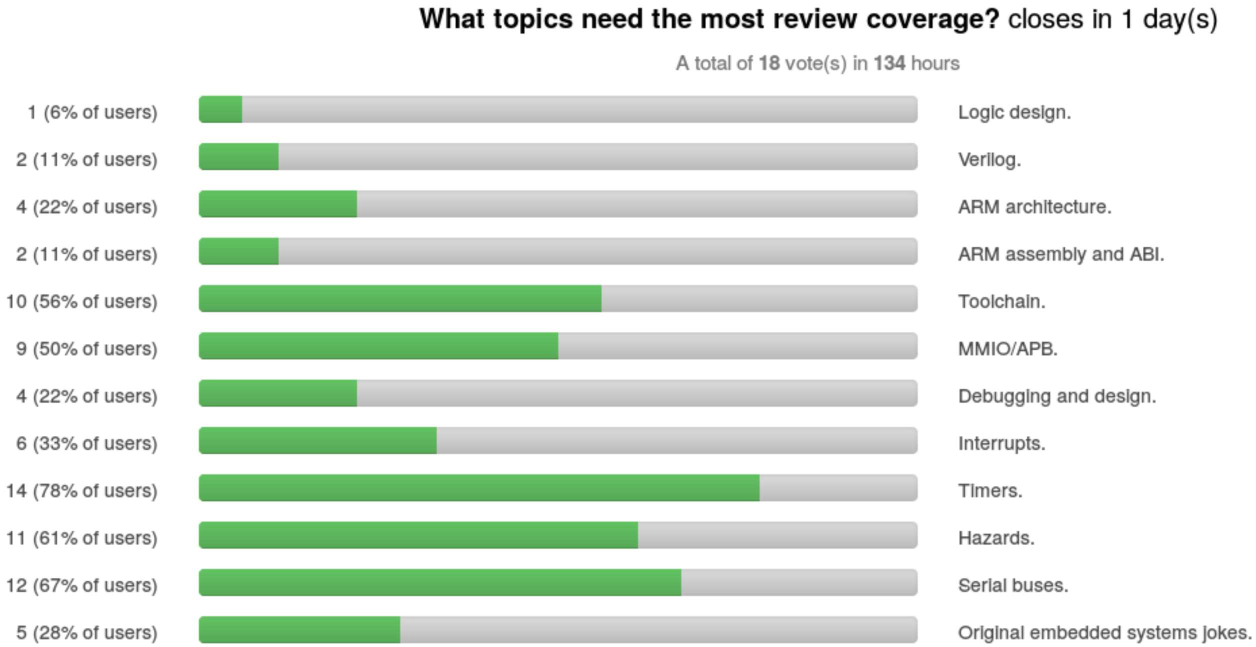
Next topics

- Mechanical.
  - Solenoids.
  - Motors.
  - Linear actuators.
  - H-bridges.
  - Shaft encoders.
- Circuits.
  - Power, energy, temperature, reliability.
  - Power supplies.
  - Voltage regulators.
  - Signal conditioning analog circuits.
- RTOSs.
- AFSM synthesis.

Breakpoints

- Hardware clock keeps running in breakpoints.
- C_DEBUGEN bit in the NVIC Debug Halting Control and Status Register (DHCSR).
- Can only set with debugger.

Poll



What topics need the most review coverage? closes in 1 day(s)

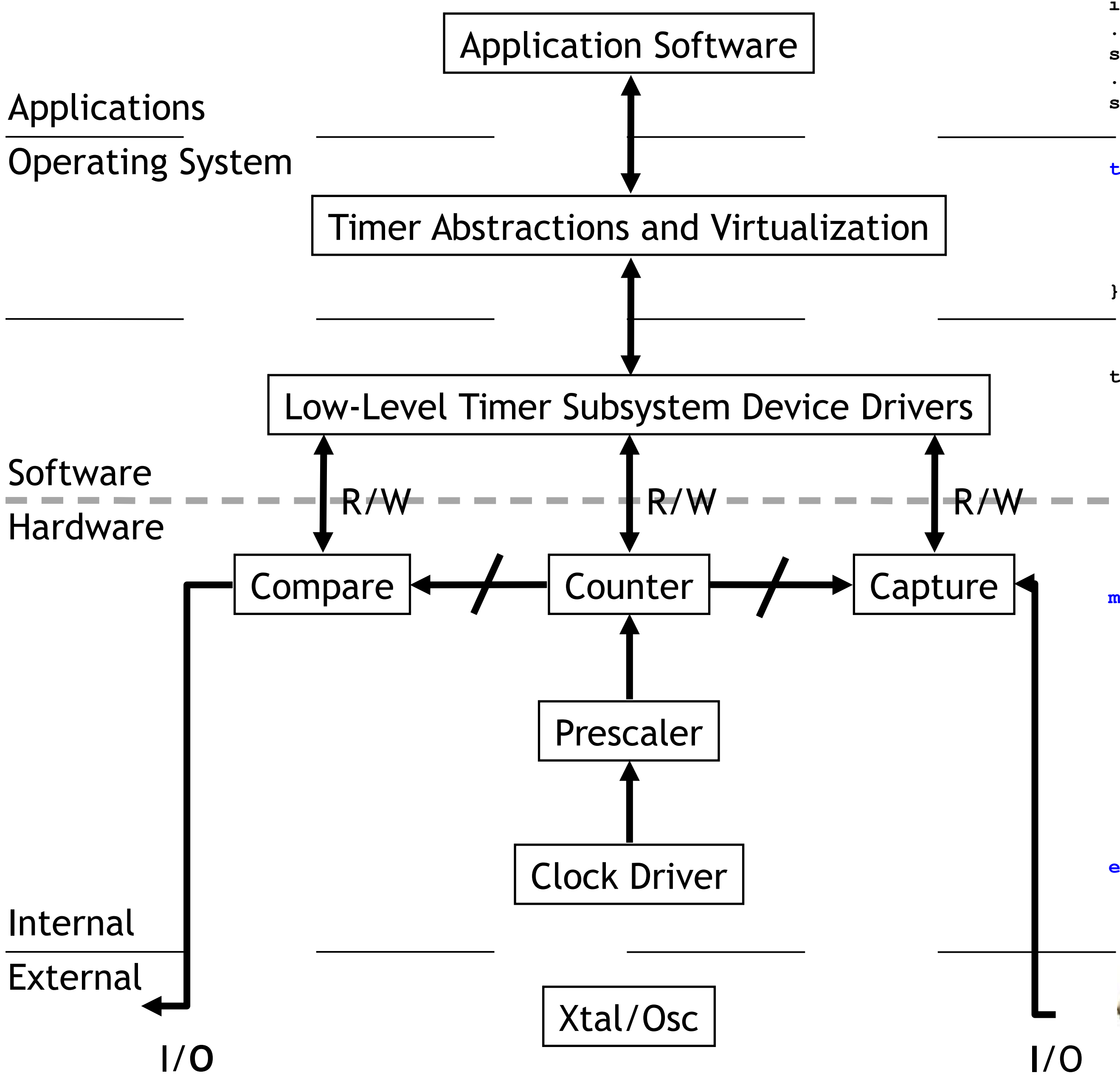
A total of **18** vote(s) in **134** hours

1 (6% of users)	Logic design.
2 (11% of users)	Verilog.
4 (22% of users)	ARM architecture.
2 (11% of users)	ARM assembly and ABI.
10 (56% of users)	Toolchain.
9 (50% of users)	MMIO/APB.
4 (22% of users)	Debugging and design.
6 (33% of users)	Interrupts.
14 (78% of users)	Timers.
11 (61% of users)	Hazards.
12 (67% of users)	Serial buses.
5 (28% of users)	Original embedded systems jokes.

Outline

- ~~Context and review~~
- ~~Midterm evaluations~~
- Timers
- Serial buses
- Hazards
- Toolchain

Anatomy of a timer system



```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```

Example timer problem

- Design a PWM LED controller.
- Crystal: 500 MHz.
- Divider: 12-bit.
- HW counter
  - 32-bit.
  - Count-down only.
  - One-shot only.
- LED
  - Brightness highly non-linear in V.
  - 100 Ohm series R.
  - 100 nF C to ground.
  - Phosphor decay halflife: 10 ms.

Outline

- ~~Context and review~~
- ~~Midterm evaluations~~
- ~~Timers~~
- Serial buses
- Hazards
- Toolchain

Serial buses

- Tristate vs. open collector.
    - Advantages and disadvantages.
    - How they work.
- Addressing methods.
- Full/half-duplex.
- Signaling.
- Timing diagrams.
- Stall cycles.
- Master/slave.
- Read/write.

Outline

- ~~Context and review~~
- ~~Midterm evaluations~~
- ~~Timers~~
- ~~Serial buses~~
- Hazards
- Toolchain

Hazards

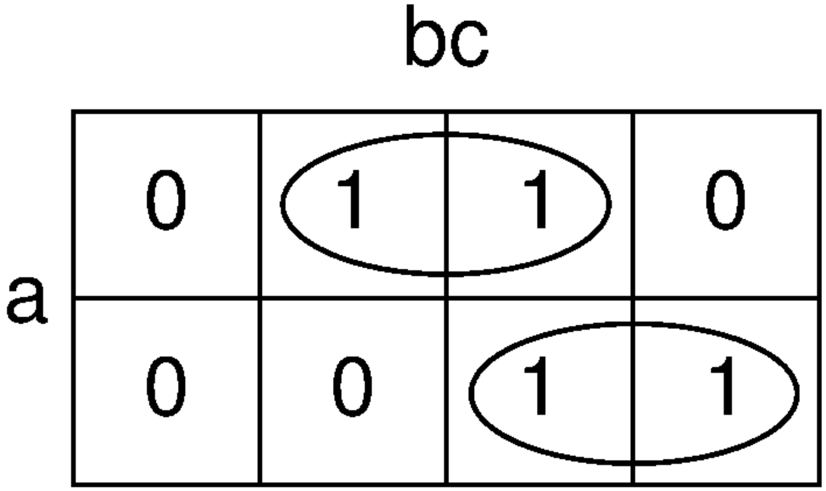
- Race between variable transitions.
- May, but not must, produce a glitch.
- Glitch
  - Static glitch: transient pulse of incorrect value when output should be stable.
  - Dynamic glitch: transient pulse of incorrect value when output should be changing.
- Consider a minimal implementation of
  - f(a, b, c) = a'b'c + a'bc + abc + abc'

Hazards: what can go wrong

- Hazardous clock signal used to drive special-purpose timer counter.
- Timer used for motor PWM.
- Two second watchdog timer on other counter.
- Duty cycle constrained. Prevent arm swinging out of safe zone in 2 seconds.
- Tested: safe.
- One day, T decreases by 10 degrees F.
- Code hangs, robot arm hits co-worker.

Hazards

- Consider a minimal implementation of
  - f(a, b, c) = a'b'c + a'bc + abc + abc'

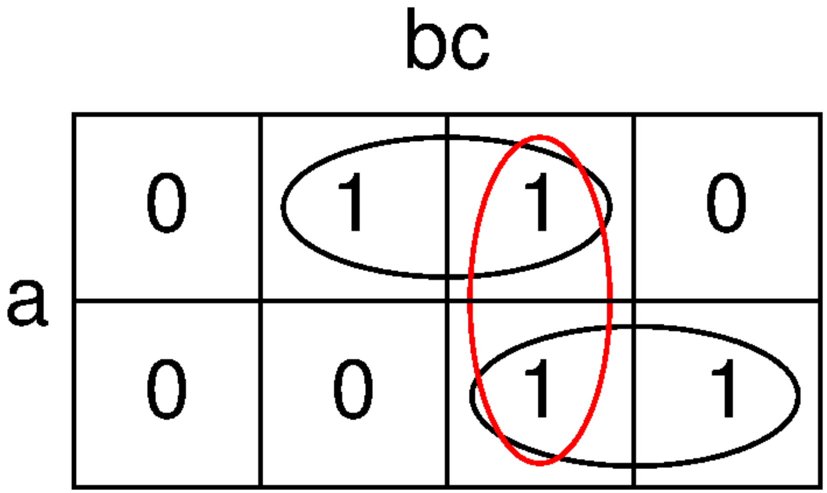


0	1	1	0
0	0	1	1

- f(a, b, c) = a'c + ab
- What if b=1, c=1?

Hazards

- How to eliminate
  - Limit logic to two levels
  - Cover all transitions

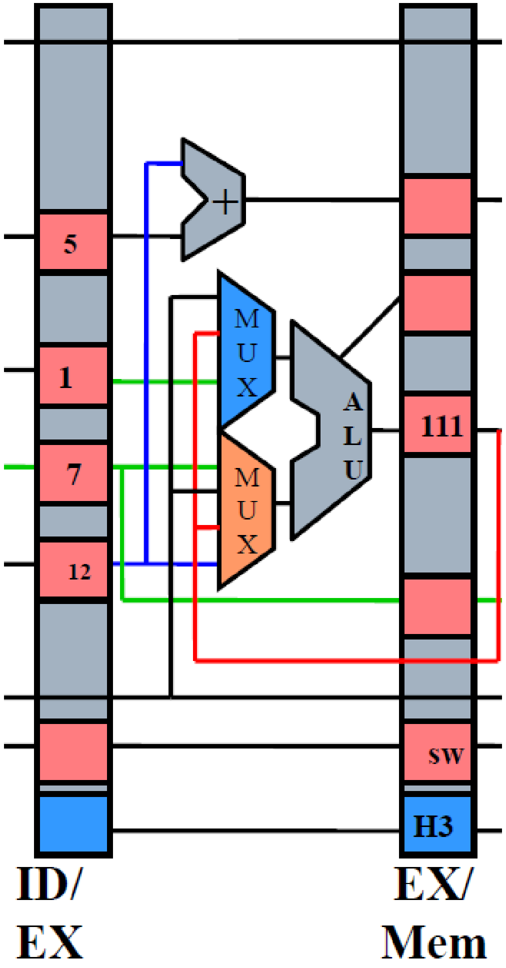


0	1	1	0
0	0	1	1

- f(a, b, c) = a'c + ab + bc
- What if b=1, c=1?

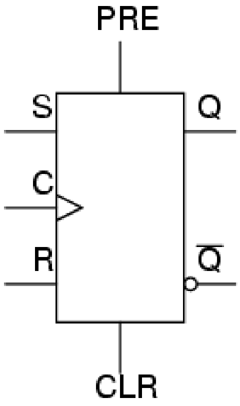
Effect of hazards

- Hazards can often be ignored in synchronous systems.
- Only sampling on clock edges.
- Make clocks slow enough for glitching to finish before next edge.
- Still wastes power.
- Causes major problems in asynchronous systems.
  - Different design style required.

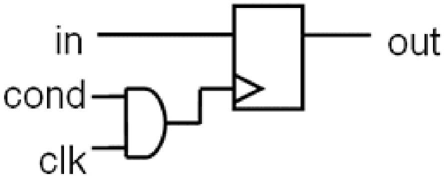


When hazards need special attention

- Asynchronous resets
  - Can use a flip-flop on the input.
    - Should have used synchronous reset.
  - Clocks
    - Hazards can produce spurious clock edges.



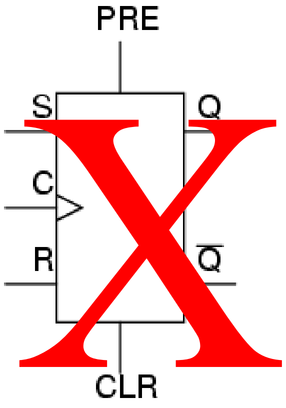
Traditionally, CLR is used to indicate async reset. "R" or "reset" for sync. reset.

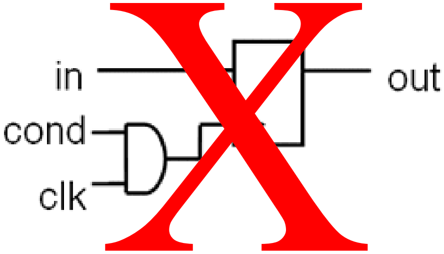


If clk is high and cond glitches, you get extra edges!

Simple design rules

- Don't use asynchronous resets unless you understand the implications fully.
- Don't drive a clock with logic containing hazards.
- Hazard-free guarantee.
    - Only two levels.
    - Cover transitions.
        - Literal or complement, not both.

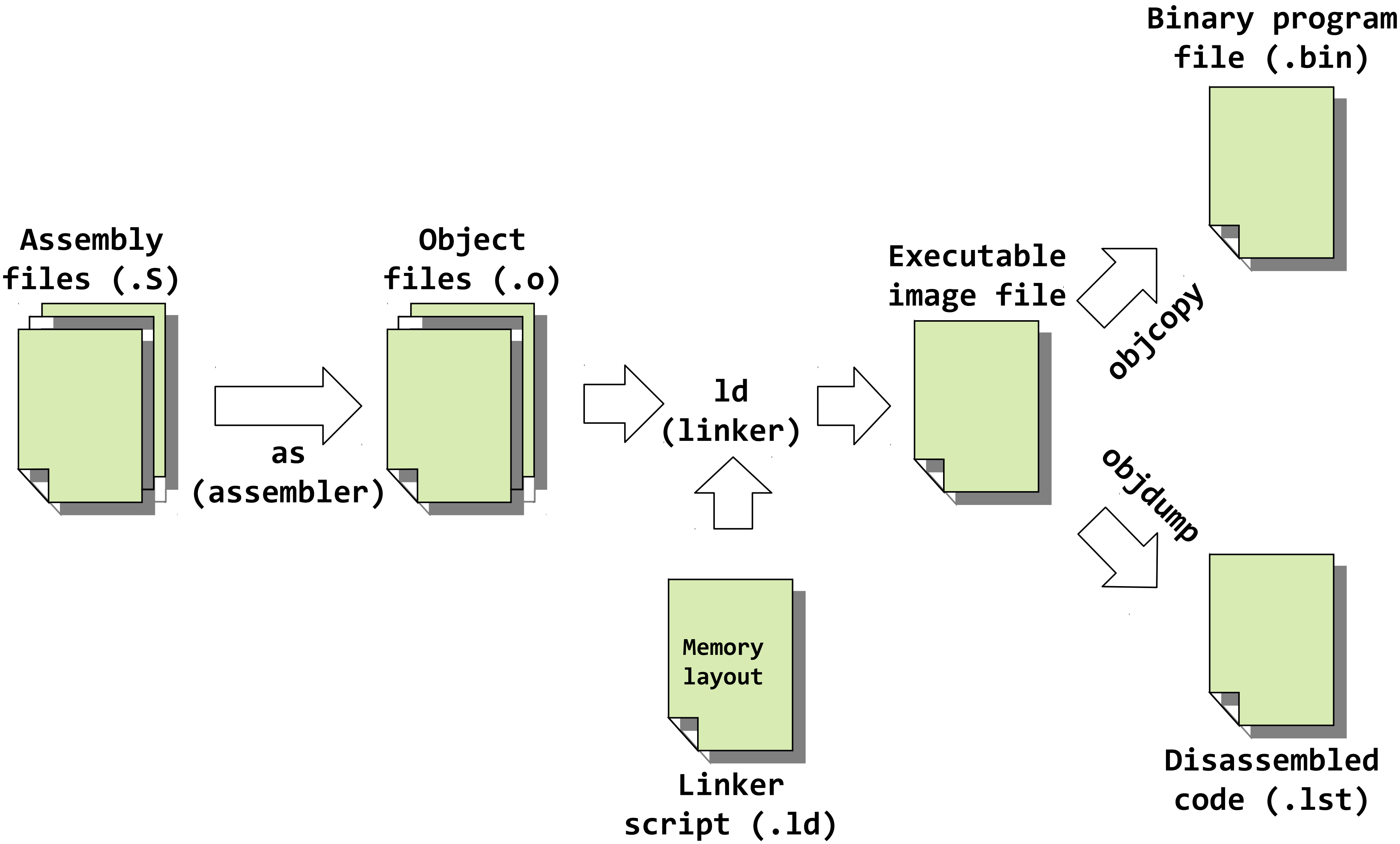




Outline

- ~~Context and review~~
- ~~Midterm evaluations~~
- ~~Timers~~
- ~~Serial buses~~
- ~~Hazards~~
- Toolchain

How does an assembly language program get turned into a executable program image?



What are the real GNU executable names for the ARM?

- Just add the prefix “arm-none-eabi-” prefix
- Assembler (as)
  - arm-none-eabi-as
- Linker (ld)
  - arm-none-eabi-ld
- Object copy (objcopy)
  - arm-none-eabi-objcopy
- Object dump (objdump)
  - arm-none-eabi-objdump
- Symbol table examiner (nl)
  - arm-non-eabi-nm
- C Compiler (gcc)
  - arm-none-eabi-gcc
- C++ Compiler (g++)
  - arm-none-eabi-g++

A simple (hardcoded) Makefile example

```
all:
	arm-none-eabi-as -mcpu=cortex-m3 -mthumb example1.s -o example1.o
	arm-none-eabi-ld -Ttext 0x0 -o example1.out example1.o
	arm-none-eabi-objcopy -Obinary example1.out example1.bin
	arm-none-eabi-objdump -S example1.out > example1.lst
```

What information does the disassembled file provide?

```
all:
    arm-none-eabi-as -mcpu=cortex-m3 -mthumb example1.s -o example1.o
    arm-none-eabi-ld -Ttext 0x0 -o example1.out example1.o
    arm-none-eabi-objcopy -Obinary example1.out example1.bin
    arm-none-eabi-objdump -S example1.out > example1.lst
```

```
     .equ  STACK_TOP, 0x20000800
     .text
     .syntax    unified
     .thumb
     .global    _start
     .type start, %function

_start:
     .word STACK_TOP, start
start:
     movs r0, #10
     movs r1, #0
loop:
     adds r1, r0
     subs r0, #1
     bne  loop
deadloop:
     b    deadloop
     .end
```

```
example1.out:     file format elf32-littlearm

Disassembly of section .text:

00000000 <_start>:
   0: 20000800    .word 0x20000800
   4: 00000000    .word 0x00000000

00000008 <start>:
   8: 200a        movs  r0, #10
   a: 2100        movs  r1, #0

0000000c <loop>:
   c: 1809        adds  r1, r1, r0
   e: 3801        subs  r0, #1
  10: d1fc        bne.n c <loop>

00000012 <deadloop>:
  12: e7fe        b.n   12 <deadloop>
```

Elements of assembly language program?

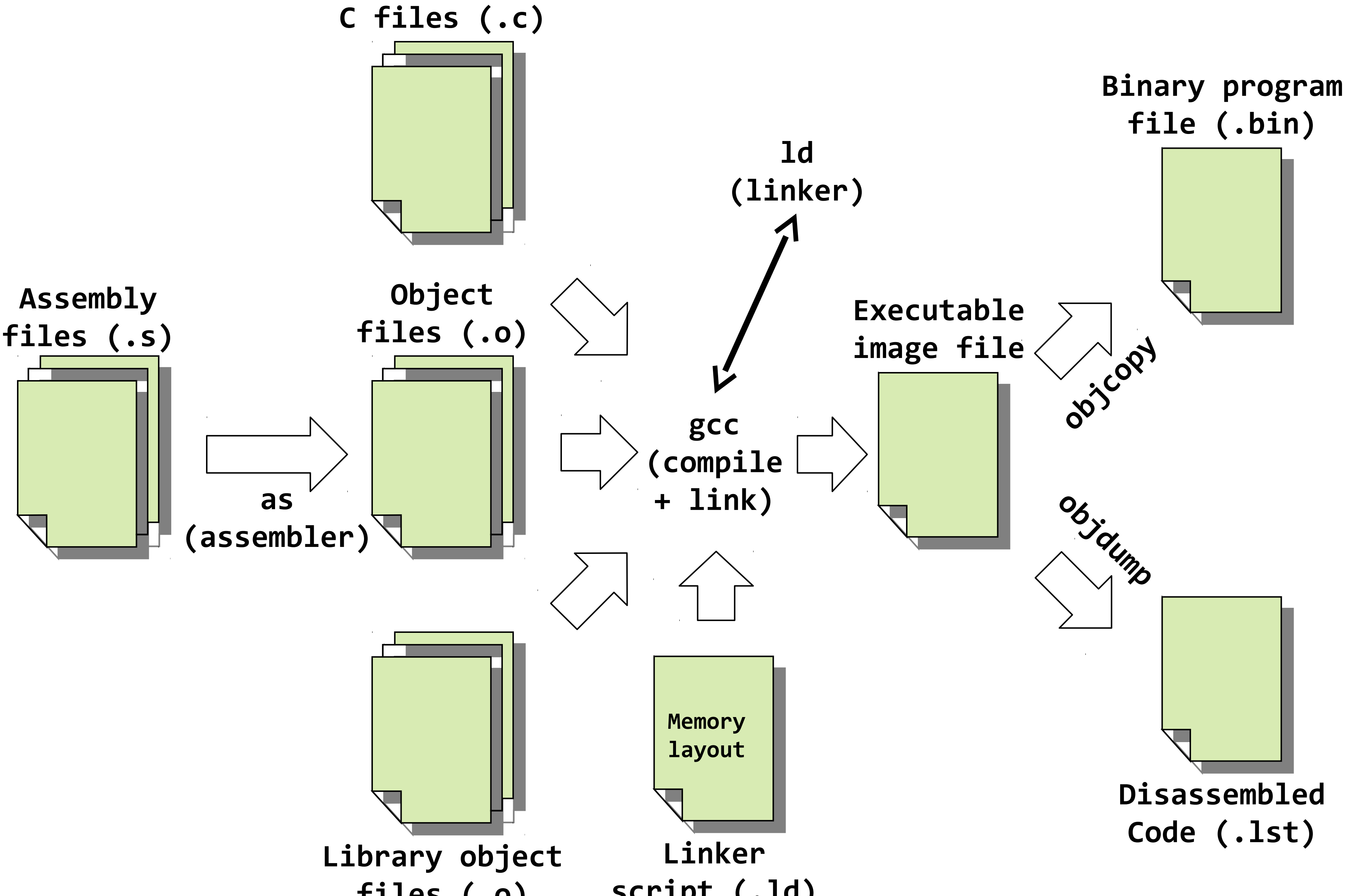
```
    .equSTACK_TOP, 0x20000800    /* Equates symbol to value */
    .text                        /* Tells AS to assemble region */
    .syntax unified              /* Means language is ARM UAL */
    .thumb                       /* Means ARM ISA is Thumb */
    .global _start               /* .global exposes symbol */
                                 /* _start label is the beginning */
                                 /* ...of the program region */
    .type   start, %function     /* Specifies start is a function */
                                 /* start label is reset handler */
_start:
    .word   STACK_TOP, start     /* Inserts word 0x20000800 */
                                 /* Inserts word (start) */
start:
    movs r0, #10                 /* We’ve seen the rest ... */
    movs r1, #0
loop:
    adds r1, r0
    subs r0, #1
    bne  loop
deadloop:
    b    deadloop
    .end
```

How are assembly files assembled?

- $ arm-none-eabi-as
  - Useful options
    - -mcpu
    - -mthumb
    - -o

```
$ arm-none-eabi-as -mcpu=cortex-m3 -mthumb example1.s -o example1.o
```

How does a mixed C/Assembly program get turned into a executable program image?


C files (.c)
Assembly files (.s)
as (assembler)
Object files (.o)
Library object files (.o)
ld (linker)
gcc (compile + link)
Memory layout
Linker script (.ld)
Executable image file
objcopy
Binary program file (.bin)
objdump
Disassembled Code (.lst)