EECS 373
Design of Microprocessor-Based Systems

Robert Dick

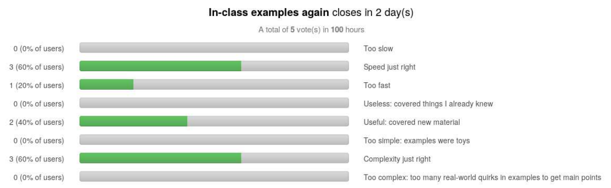
University of Michigan

Lecture 4: Toolchain, ABI, Memory Mapped I/O

18 September 2017

Many slides from Mark Brehob.

In-class examples



In-class examples again closes in 2 day(s)

A total of **5** vote(s) in **100** hours

0 (0% of users)	Too slow
3 (60% of users)	Speed just right
1 (20% of users)	Too fast
0 (0% of users)	Useless: covered things I already knew
2 (40% of users)	Useful: covered new material
0 (0% of users)	Too simple: examples were toys
3 (60% of users)	Complexity just right
0 (0% of users)	Too complex: too many real-world quirks in examples to get main points

Poll results
- Improved
- Maybe still slightly too fast

Actions
- Slow down a little bit more
- Keep narrating
- Keep example complexity/realism trade-off the same

Midterm

20 Sep: Practice/review material posted
24 Sep: Practice/review solutions posted
7pm 27 Sep: Midterm exam 1

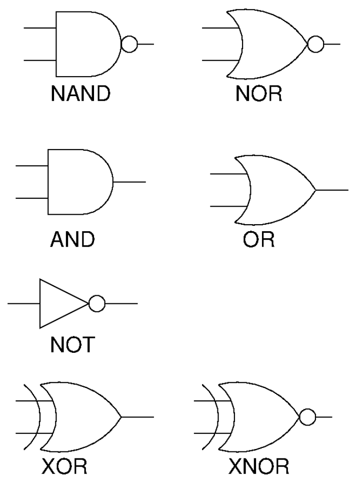
Outline

- Homework 1 review
- Power-on reset
- Pickmin example
- IP register and veneers
- Memory mapped IO

HW1: powerful logic design heuristics

- If you are ever stuck on a design problem involving a combinational network, write the truth table
- If you are ever stuck on a design problem involving a sequential network, draw the state diagram

Gates



De Morgan's Laws and bubble pushing

(ab)' = a' + b'
(a + b)' = a'b'

Example:
a'b' + b'c + ac
((a'b)' + (b'c)' + (ac)')'

Outline

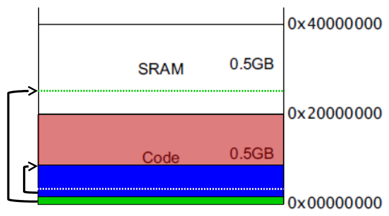
- ~~Homework 1 review~~
- Power-on reset
- Pickmin example
- IP register and veneers
- Memory mapped IO

Power-on reset

- On the ARM Cortex-M3
- SP and PC are loaded from the code (.text) segment
- Initial stack pointer
  - LOC: 0x00000000
  - POR: SP ← mem(0x00000000)
- Interrupt vector table
  - *Initial* base: 0x00000004
  - Vector table is relocatable
  - Entries: 32-bit values
  - Each entry is an address
  - Entry #1: reset vector
    - LOC: 0x0000004
    - POR: PC ← mem(0x00000004)
- Execution begins

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

_start:
     .word     STACK_TOP, start
start:
     movs r0, #10
     ...
```



Register at 0xE000ED08 sets where vector table is.

Outline

- ~~Homework 1 review~~
- ~~Power-on reset~~
- Pickmin example
- IP register and veneers
- Memory mapped IO

IT blocks

- If-then
- Four following instructions can be conditional
- Automatically inserted by assembler
- Must specify condition
- Contained instructions must use that condition
- Longer branches enabled
- Makes status bits available in Thumb mode
- Doesn't generate code in non-Thumb mode

IT blocks

Example

```
IT    EQ
MOVEQ  r0,r1
BEQ    dloop     ; branch at end of IT block is permitted
```

Incorrect example

```
IT     NE
ADD    r0,r0,r1  ; syntax error: no condition code used
```

Outline

- ~~Homework 1 review~~
- ~~Power-on reset~~
- ~~Pickmin example~~
- IP register and veneers
- Memory mapped IO

IP register and veneers

- How far can one branch?
- With bl, even numbers in -16777216 to 16777214 range
- That’s 24 bits (25 due to even restriction)
- 24 < 32
- Need lilypads to hop on, i.e., veneers
- Linker-generated glue to handle far calls
- Allowed (but not required) to use r12

Outline

- ~~Homework 1 review~~
- ~~Power-on reset~~
- ~~Pickmin example~~
- ~~IP register and veneers~~
- Memory mapped IO

Old-style I/O

- Special instructions for reading/writing peripherals
- Wasteful: Already have read/write instructions

Memory-mapped I/O

- Put peripherals at memory addresses
- Turn LED turn on by writing 1 to address 5
- Read button state (active-high) at address 4
- Use a bus on which the peripheral sits

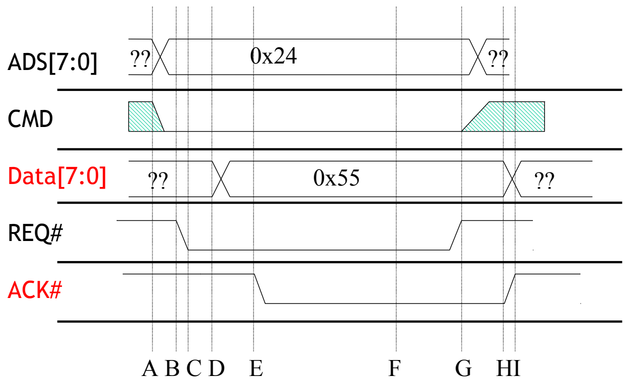
Bus access example

- Discuss a basic bus protocol
  - Asynchronous (no clock)
  - Initiator and Target
  - REQ#, ACK#, Data[7:0], ADS[7:0], CMD
    - CMD=0 is read, CMD=1 is write.
    - REQ# low means initiator is requesting something.
    - ACK# low means target has done its job.

A read transaction

- Initiator wants to read location 0x24
  - Initiator sets ADS=0x24, CMD=0
  - Initiator *then* sets REQ# to low
    - Delay first
  - Target sees read request.
  - Target drives data onto data bus.
  - Target *then* sets ACK# to low.
  - Initiator grabs the data from the data bus.
  - Initiator sets REQ# to high, stops driving ADS and CMD
  - Target stops driving data, sets ACK# to high terminating the transaction

Read transaction



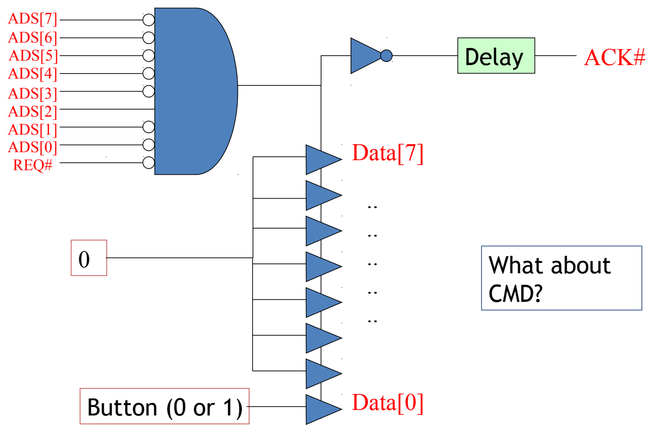
Write transaction

(write 0xF4 to location 0x31)

- Initiator sets ADS=0x31, CMD=1, Data=0xF4
- Initiator *then* sets REQ# to low.
- Target sees write request.
- Target reads data from data bus. (Just has to store in a register, need not write all the way to memory!)
- Target *then* sets ACK# to low.
- Initiator sets REQ# to high & stops driving other lines.
- Target sets ACK# to high terminating the transaction

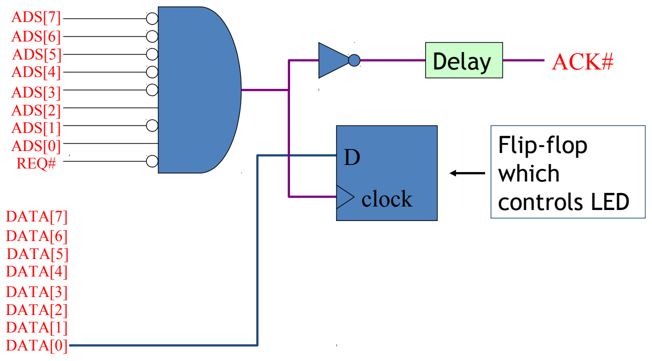
The push-button

(if ADS=0x04 write 0 or 1 depending on button)



The LED

(1 bit reg written by LSB of address 0x05)



What does this look like from software perspective?

```
volatile char * button_ads = (char *)(0x24);

char read_button(void) {
       return *button_ads;
}
```

```
.equ BUTTON_ADS, 0x24

movw r0, #:lower16:BUTTON_ADS
movt r0, #:upper16:BUTTON_ADS
ldr r1, [r0, #0]
```