Q

a question:

...you know this can't be good

exam:

next Friday
(11.15.2013)

material from exam one to A/D & D/A
(UART--AD/DA)

I2C III

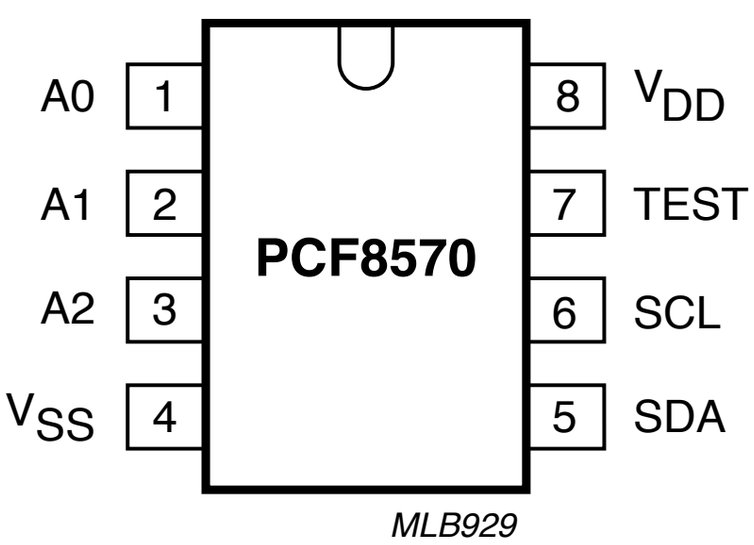
ECE 3710

I told my psychiatrist that everyone hates me. He said I was being ridiculous - everyone hasn't met me yet.

- Rodney Dangerfield

LM3S1968 burst TX example

256x8-bit
RAM
(I2C)



MLB929

to read memory:
1. specify address
(master TX)
2. get byte
(master RX)

to write memory:
1. specify address
(master TX)
2. send byte
(master TX)

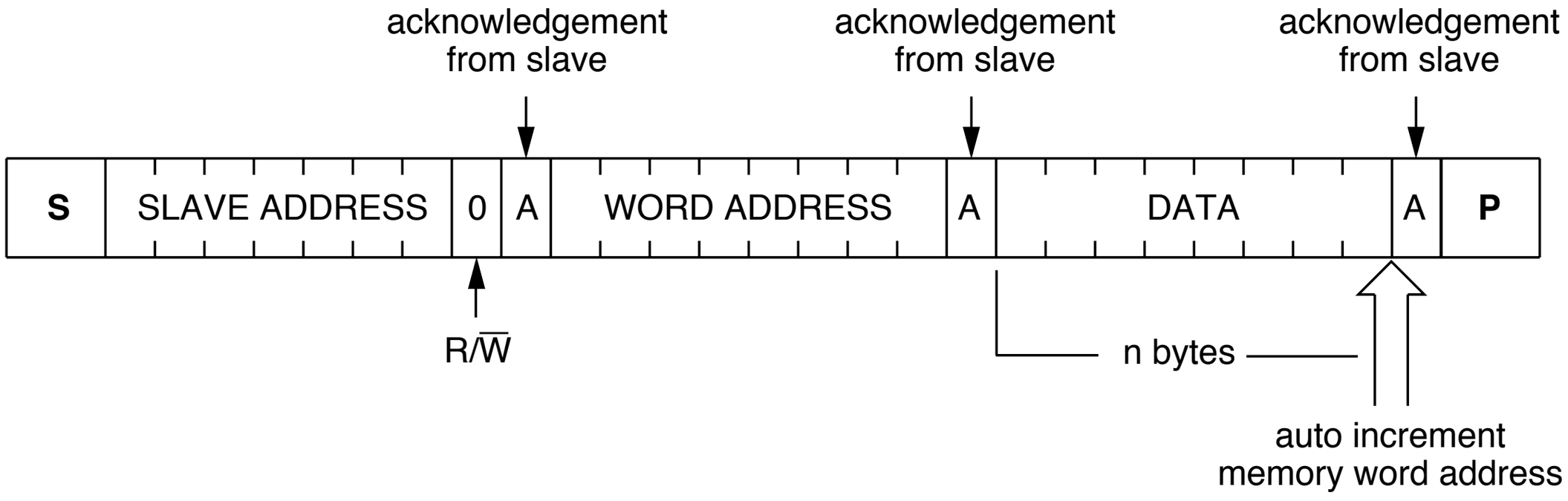
multiple RX/TX need to
read/write to device

LM3S1968 burst TX example

requires burst mode

(master doesn't give up line
or issue STOP/START)

write (master TX/TX):

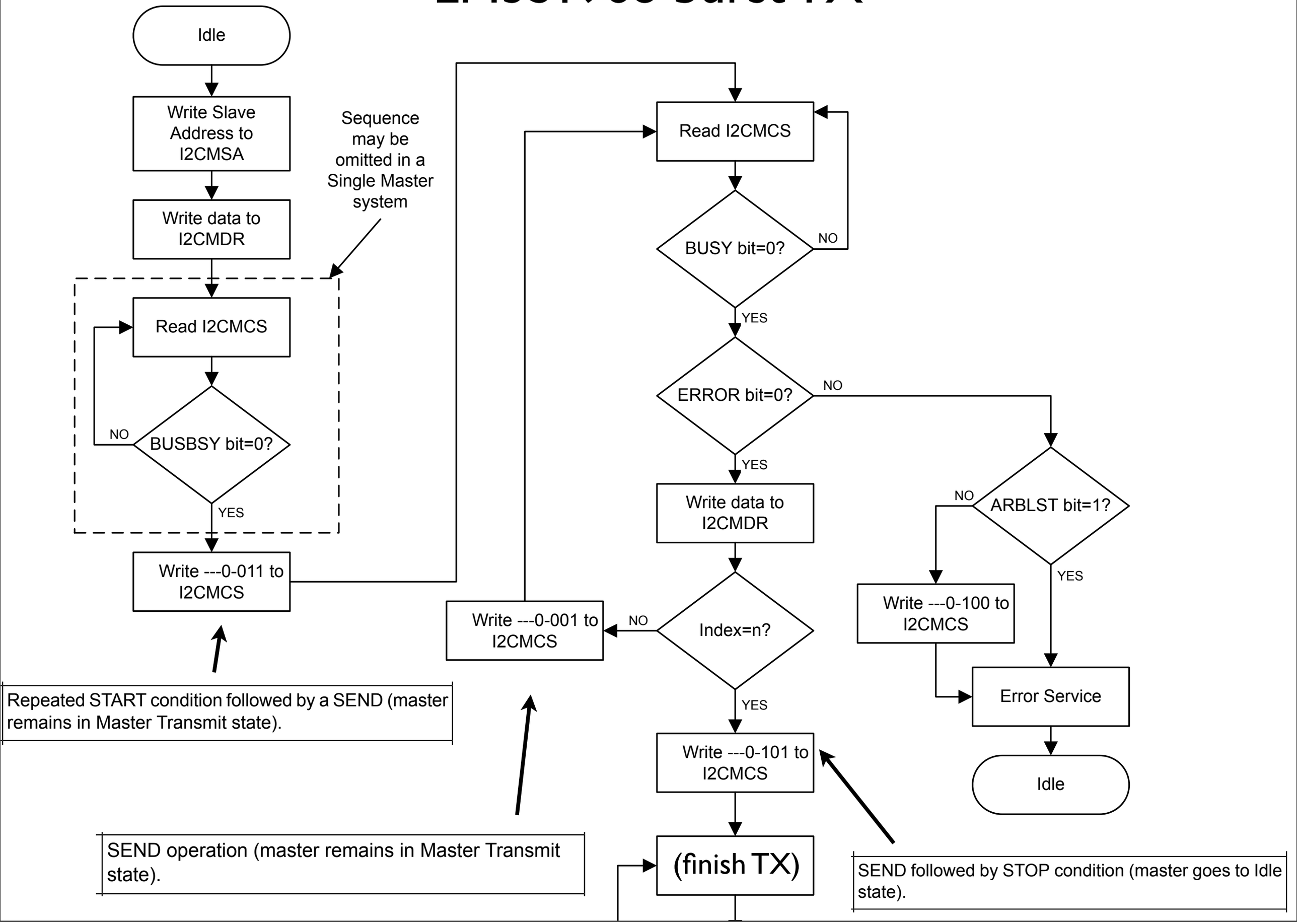


last addr specified +1
remembered by device

(remembers current addr for a special no-addr read)

master could TX another byte
w/o RESTART

LM3S1968 burst TX



gew...

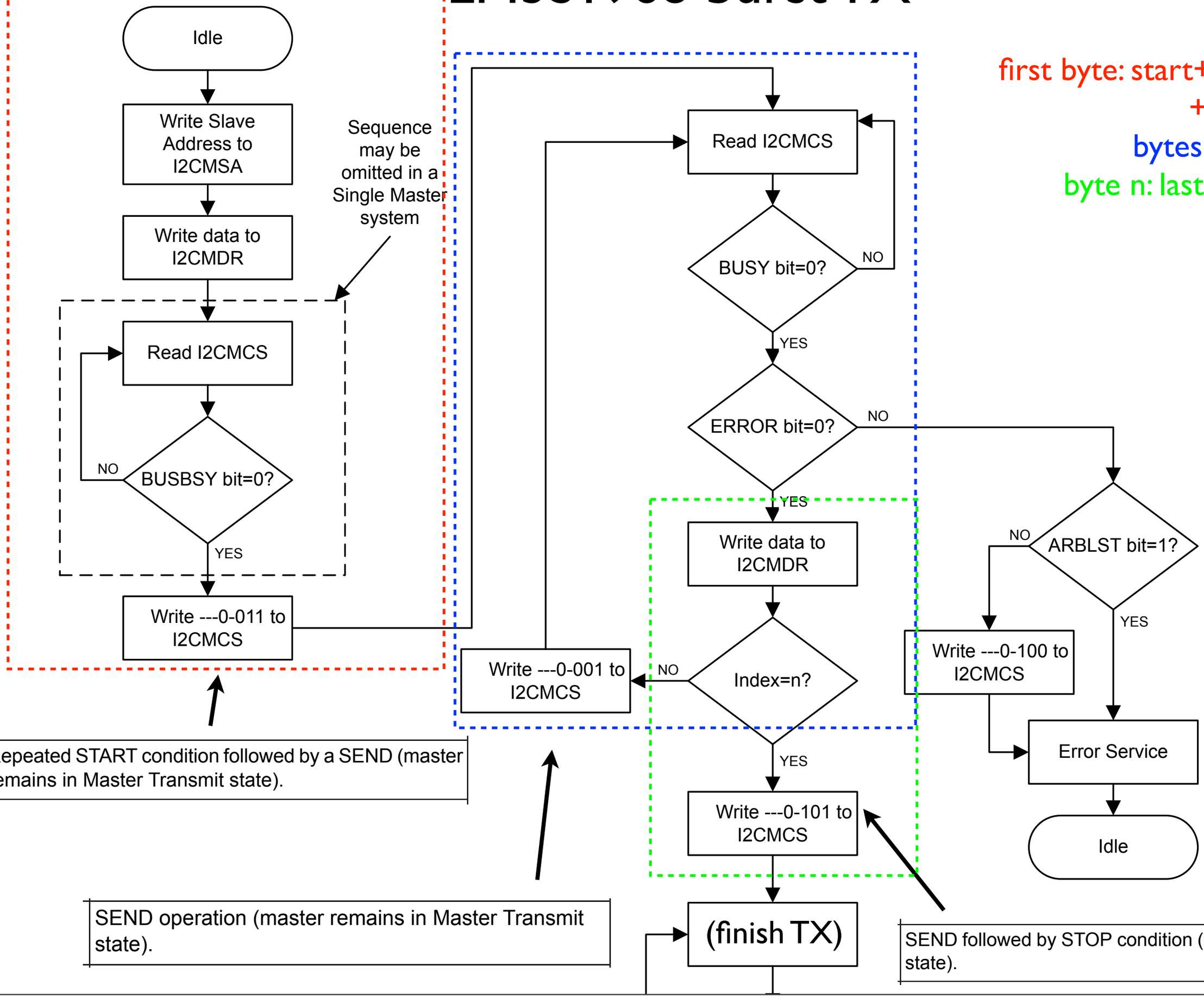
take help wherever you can get it...

LM3S1968 burst TX

first byte: start+slave addr. +mem. addr

bytes 1:n-1: data

byte n: last byte+stop



write to PCF8570-style RAM

(one byte)

this routine performs burst write procedure for writing to PCF8570-style RAM

R10: byte to write

R11: destination of byte

R12: slave addr

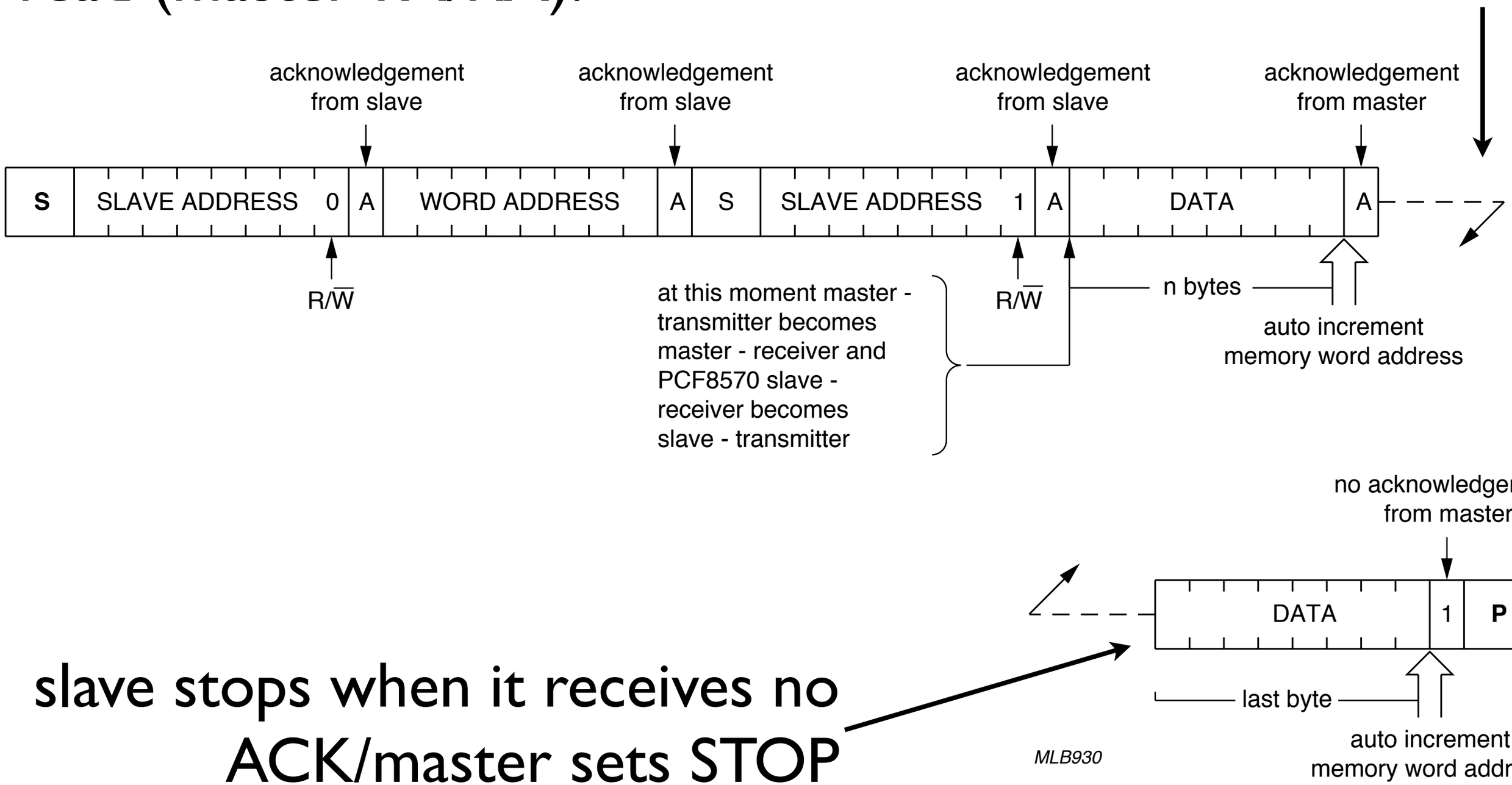
essentially: str R10,[R11]
(where memory is external)

```
I2C_MEM_WR
  ; 0. save return addr
  push {LR}

  ; 1. set slave addr and direction
  ldr R1,=I2C0
  lsl R0,R12,#0x1
  str R0,[R1,#0x0]

  ; 2. tx byte addr first (put data in tx reg)
  str R11,[R1,#0x8]

  ; 3. set master to burst mode
  mov R0,#0x3 ;0x3=0b11 (master remains in tx mode after tx)
  str R0,[R1,#0x4]

  ; 4. poll busy bit (busy=1 then still tx)
  bl I2C0_POLL

  ; 5. now tx byte
  str R10,[R1,#0x8]

  ; 6. set master to single send mode
  mov R0,#0x5 ;0x5=0b101 (master goes idle after tx)
  str R0,[R1,#0x4]

  ; 7. poll busy bit
  bl I2C0_POLL

  ; 8. we're done: restore LR and return to main
  pop {LR}
  bx LR
```

LM3S1968 burst RX example

really only have to give start addr; can read consecutive bytes

read (master TX/RX):



slave stops when it receives no ACK/master sets STOP

read from PCF8570-style RAM

(one byte)

this routine performs read procedure
for PCF8570-style RAM
R10: where we put
R11: addr of byte
R12: slave addr
essentially: ldr R10,[R11]
(where memory is external)

```
I2C_MEM_RD
  ; 0. save return addr
  push {LR}

; 1. tx addr of byte to grab
mov R10,R11 ;TX addr of byte we want
bl I2C0_TX

; 2. rx byte at addr in R11 from slave
bl I2C0_RX

; 3. we're done: restore LR and return to main
pop {LR}
bx LR
```

re-use I2C0_TX:
R10: byte to write
R12: slave addr

re-use I2C0_RX:
R10: where to put byte
R12: slave addr

done w/I2C: just when you think you're getting ahead....

somebody hits you in the head with a fish

DC Motors I

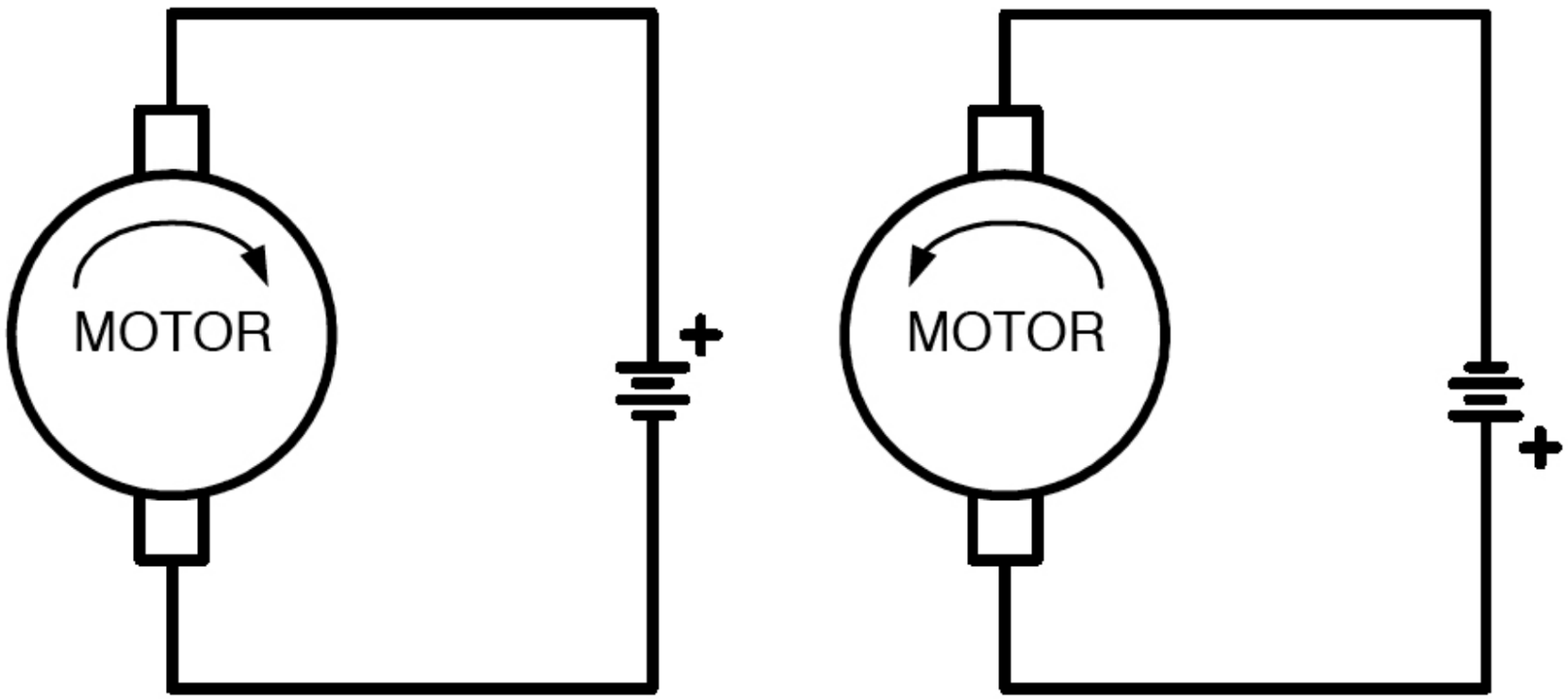
ECE 3710

motor control:

1. speed
2. direction

dc motor

direction:



Clockwise Rotation

Counter-Clockwise Rotation

speed:

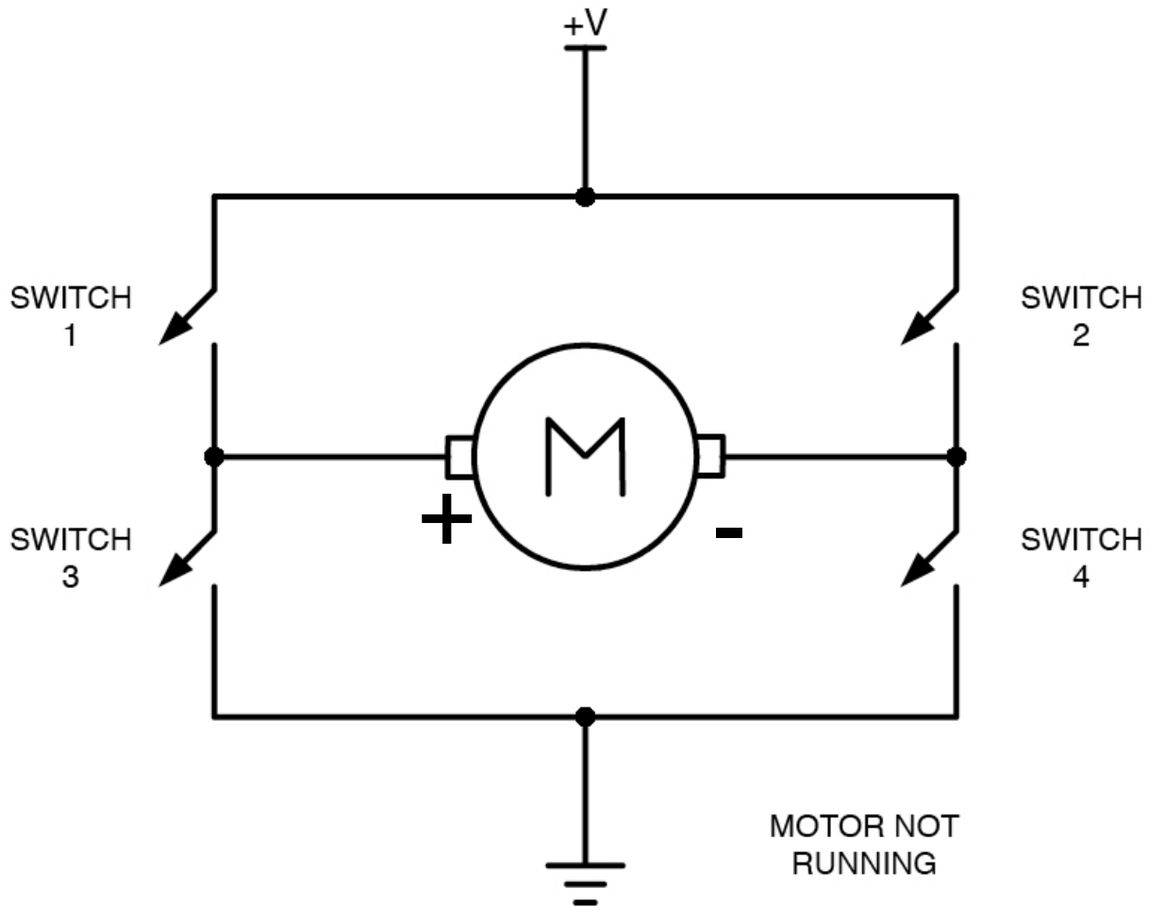
~ Voltage

ideal

(current sourced and load)

dc motor: direction

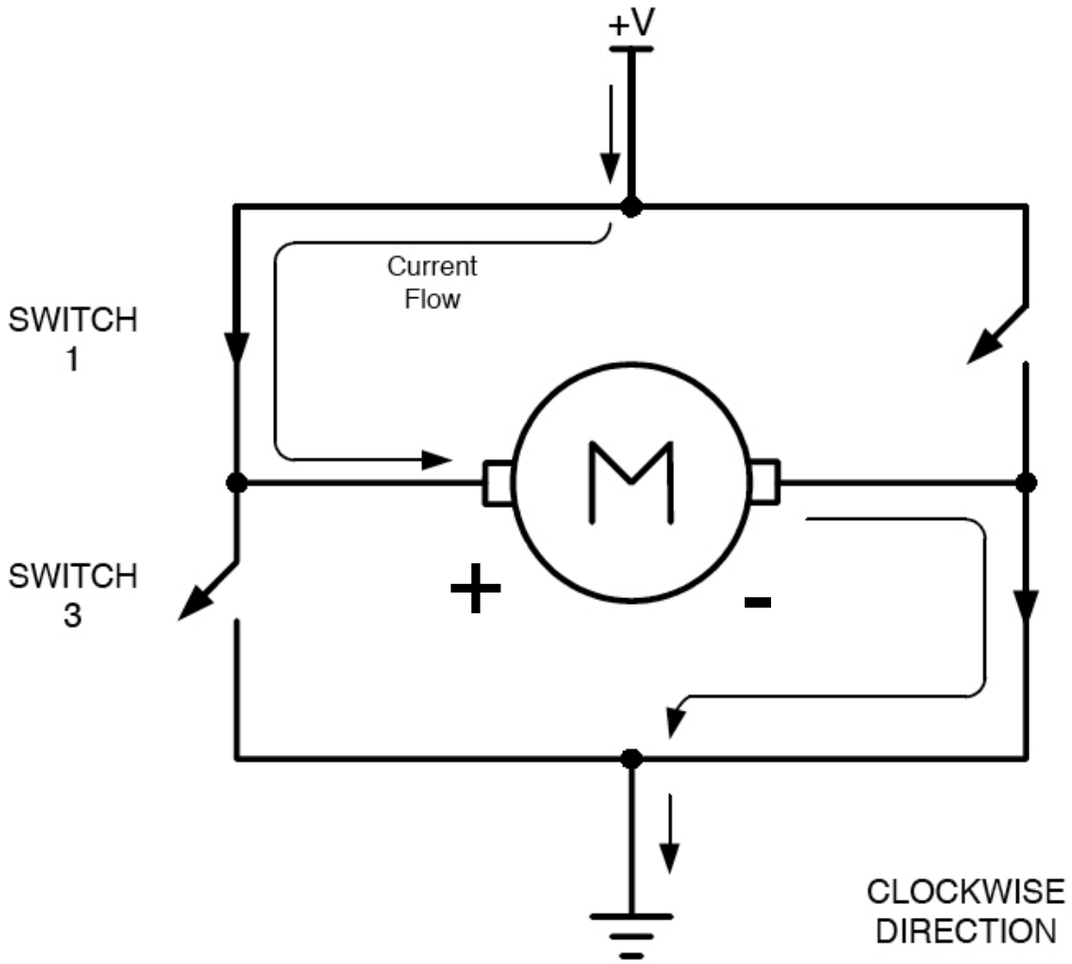
h-bridge



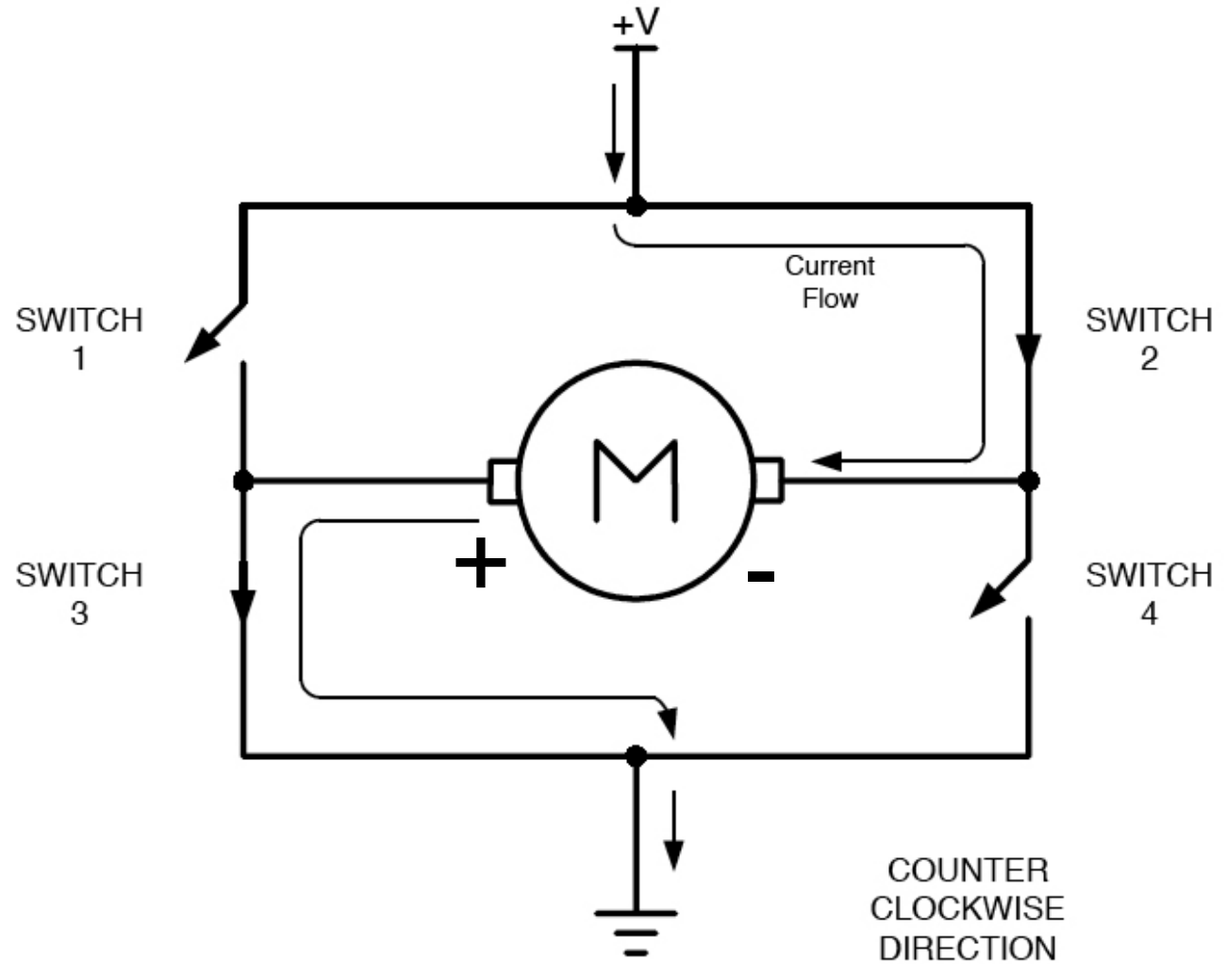
Q: how to set polarity?

don't want to have multiple supplies

dc motor: direction

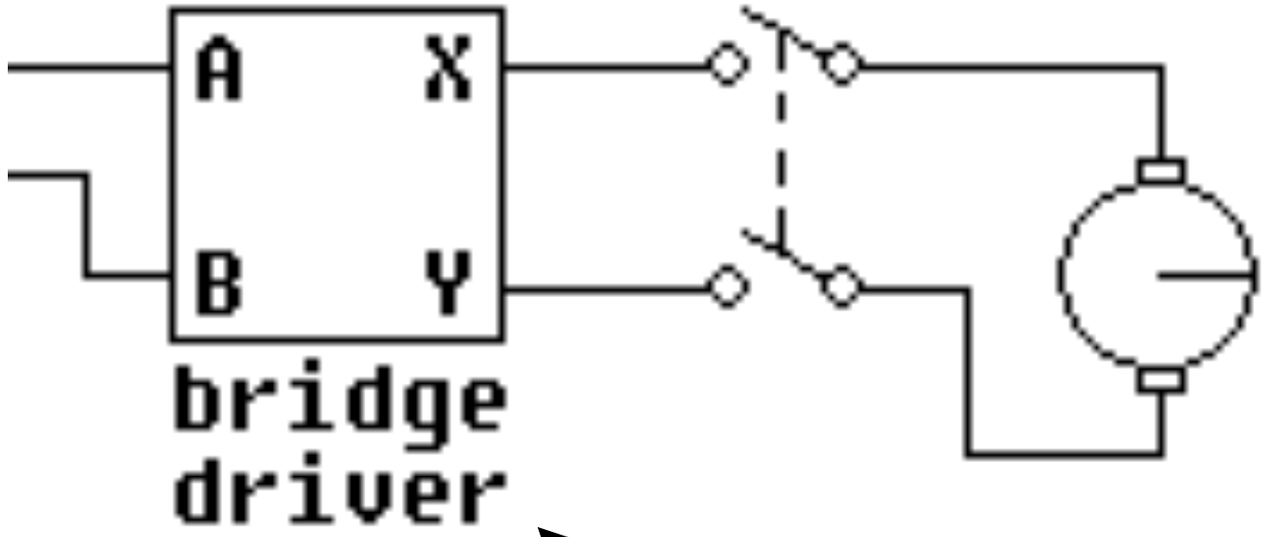


S1	S2	S3	S4
off	0	0	0	0
CW	1	0	0	1
CCW	0	1	1	0
invalid	1	1	1	1



dc motor: direction

uC needn't drive h-bridge switches directly:
(waste o' pins)



to uC outputs

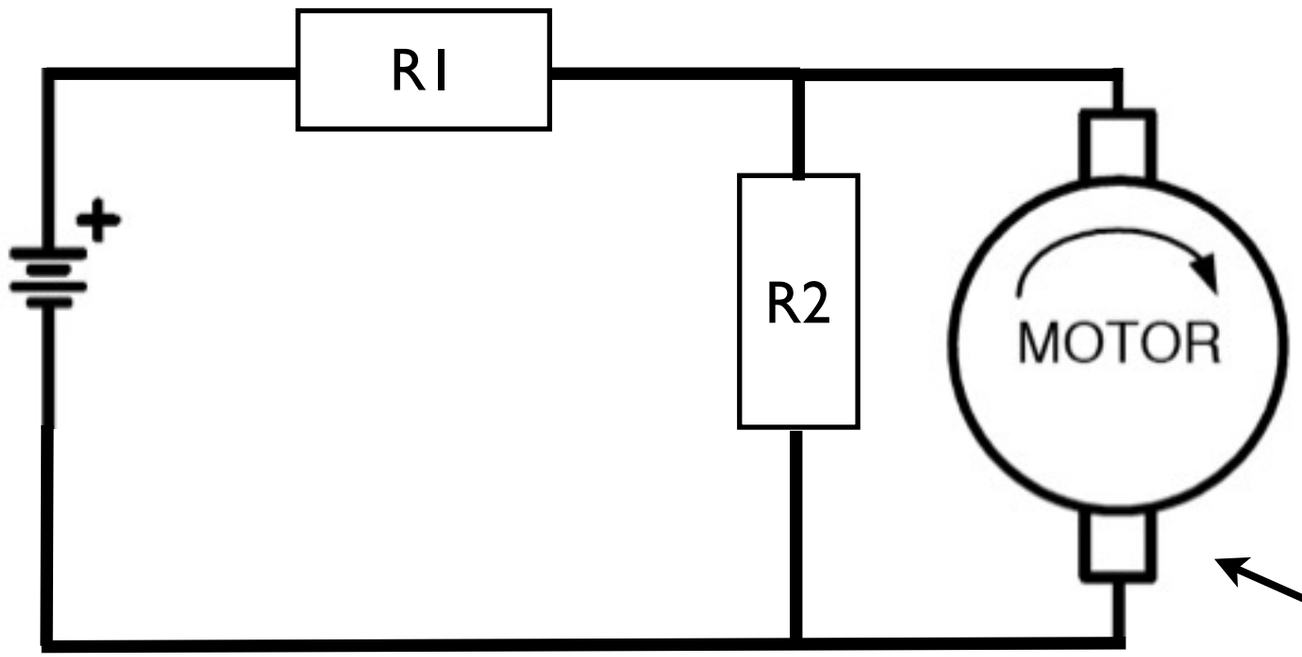
e.g. TC4428

A	B	motor
0	0	stop
0	1	forward
1	0	reverse
1	1	stop

dc motor: speed

if speed ~ Voltage ⟶ to slow down

decrease voltage



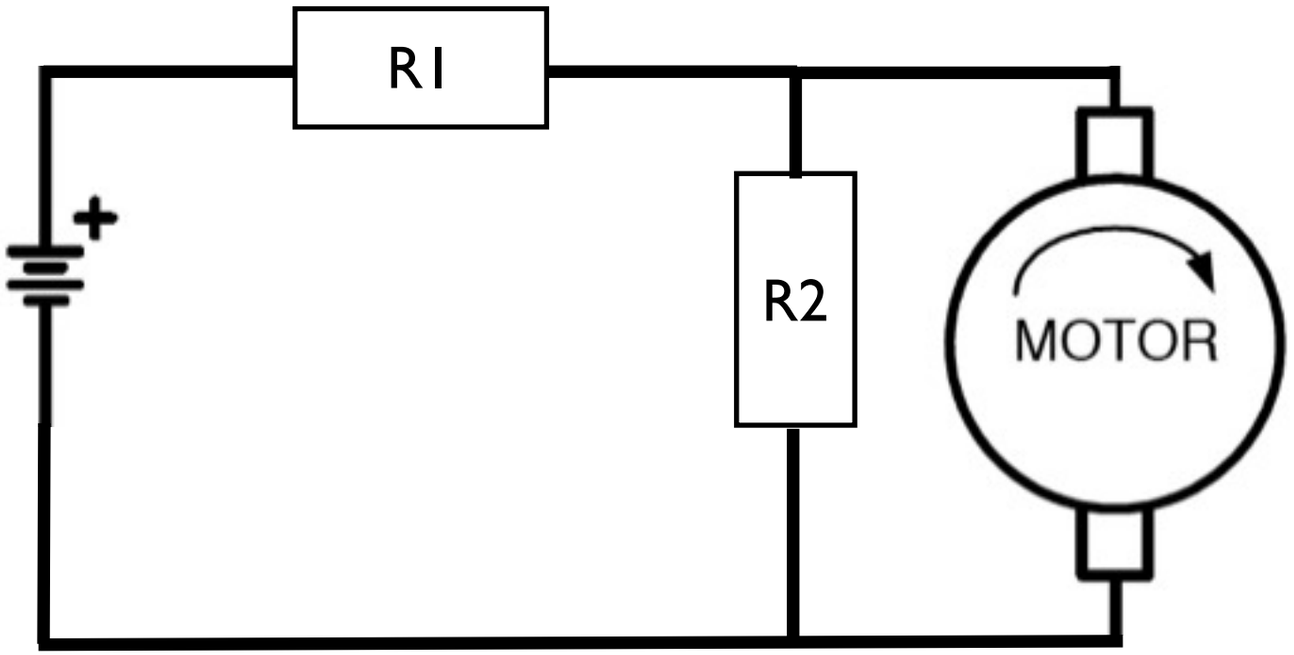
common configuration for computer fan control

Q: why not optimal?

the lowly voltage divider...

dc motor: speed

if speed ~ Voltage ⟶ to slow down



decrease voltage

Q: why not optimal?

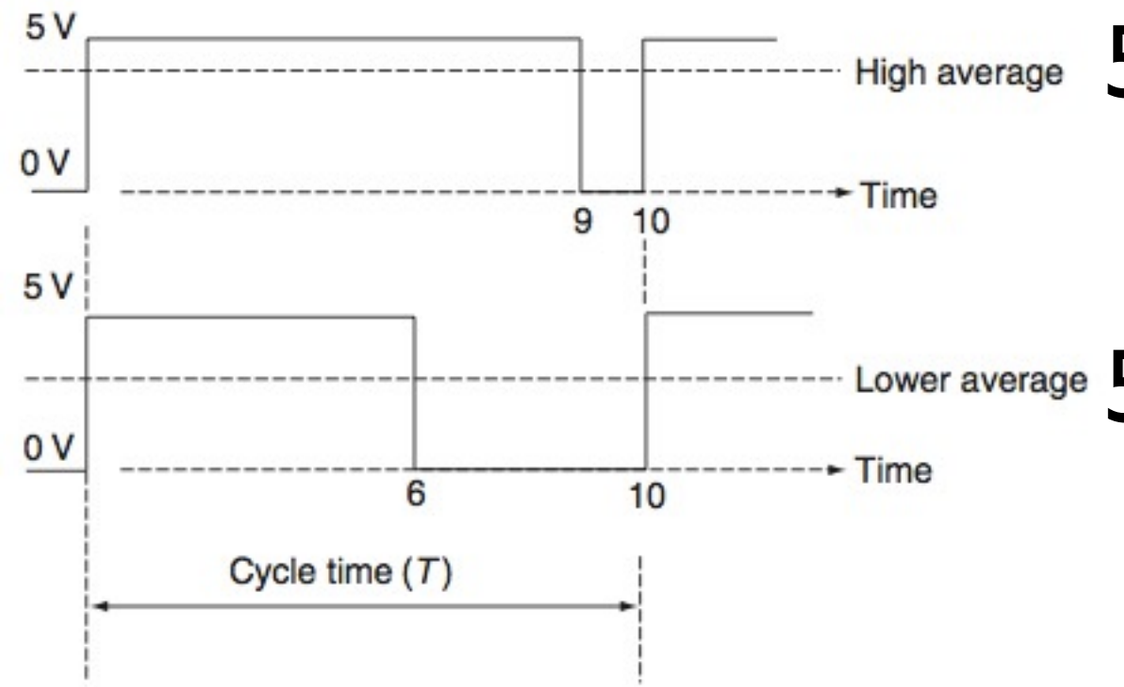
A:

1. no variable speed (w/o variable resistor)

2. wasted power

dc motor: speed

pulse-width modulation:



'on' time

↓

5 V * (9/10) = 4.5 V

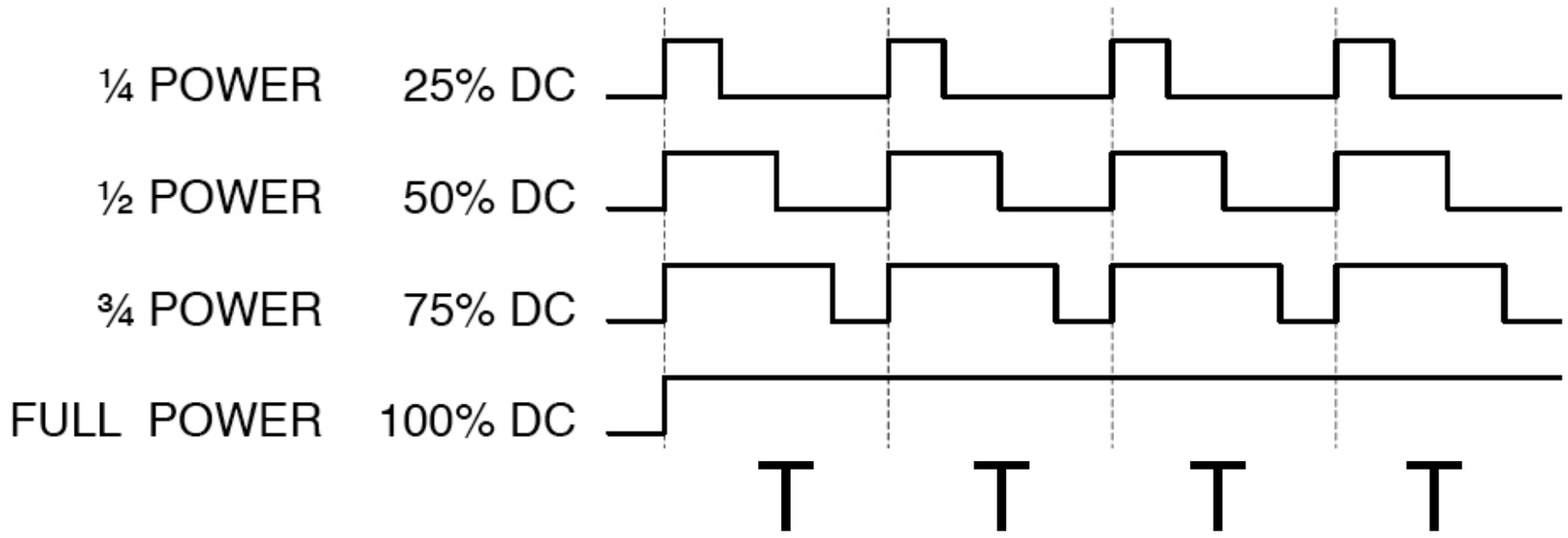
5 V * (6/10) = 3.0 V

↑

aka duty cycle

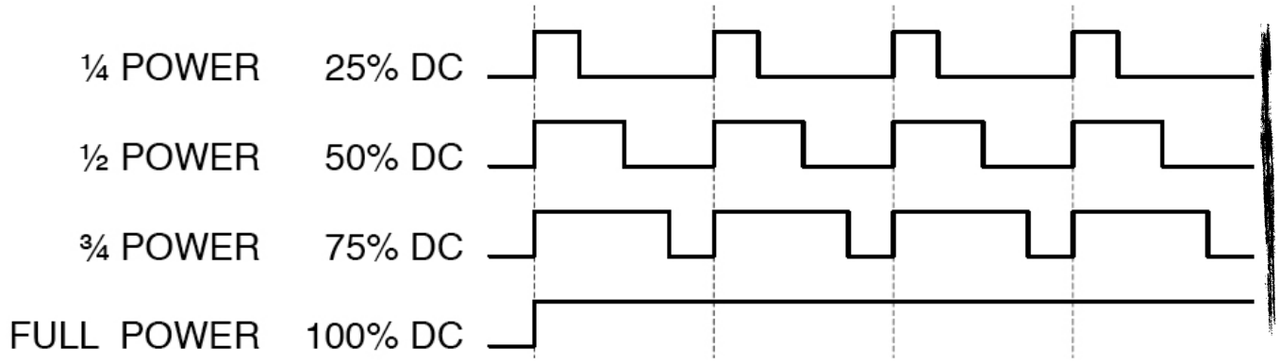
dc motor: speed

if T << 1 ⟵ motor just sees average

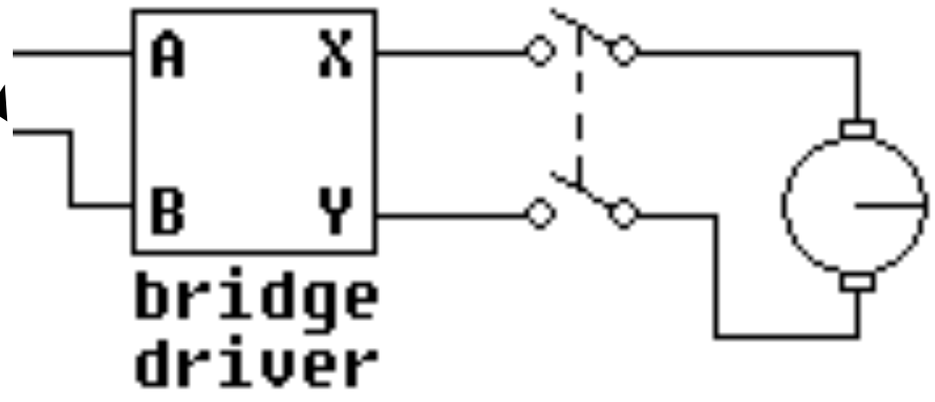


dc motor: speed

(PWM)



PWM signals serve as input to driver



A	B	motor
0	0	stop
0	1	forward
1	0	reverse
1	1	stop

note: signal can't change too fast for driver

(look at driver switching speed)

really, almost any
uC can do PWM

how-to generate PWM output:

1. GPIO + timer
2. dedicated module

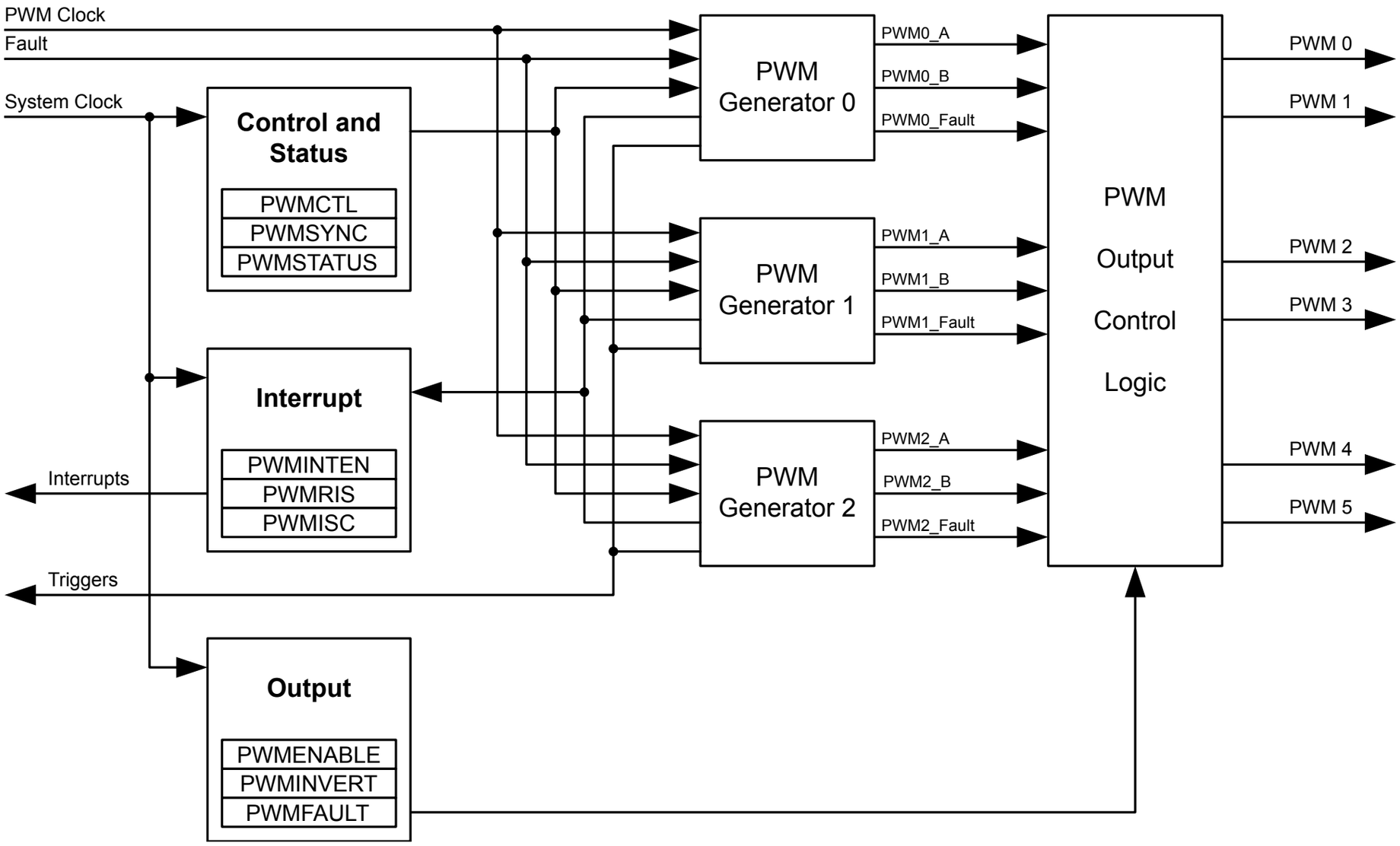
a. makes complex,
parallel PWM output easier
(especially for stepper motors)
b. frees up timer

Q: which method do we focus on?

Q: which method do we focus on?

A: dedicated module

LM3S1968 PWM modules



1. three modules, two pwm gen. per
2. one 16-bit timer module
3. count: down and up-then-down
4. two comparators per module

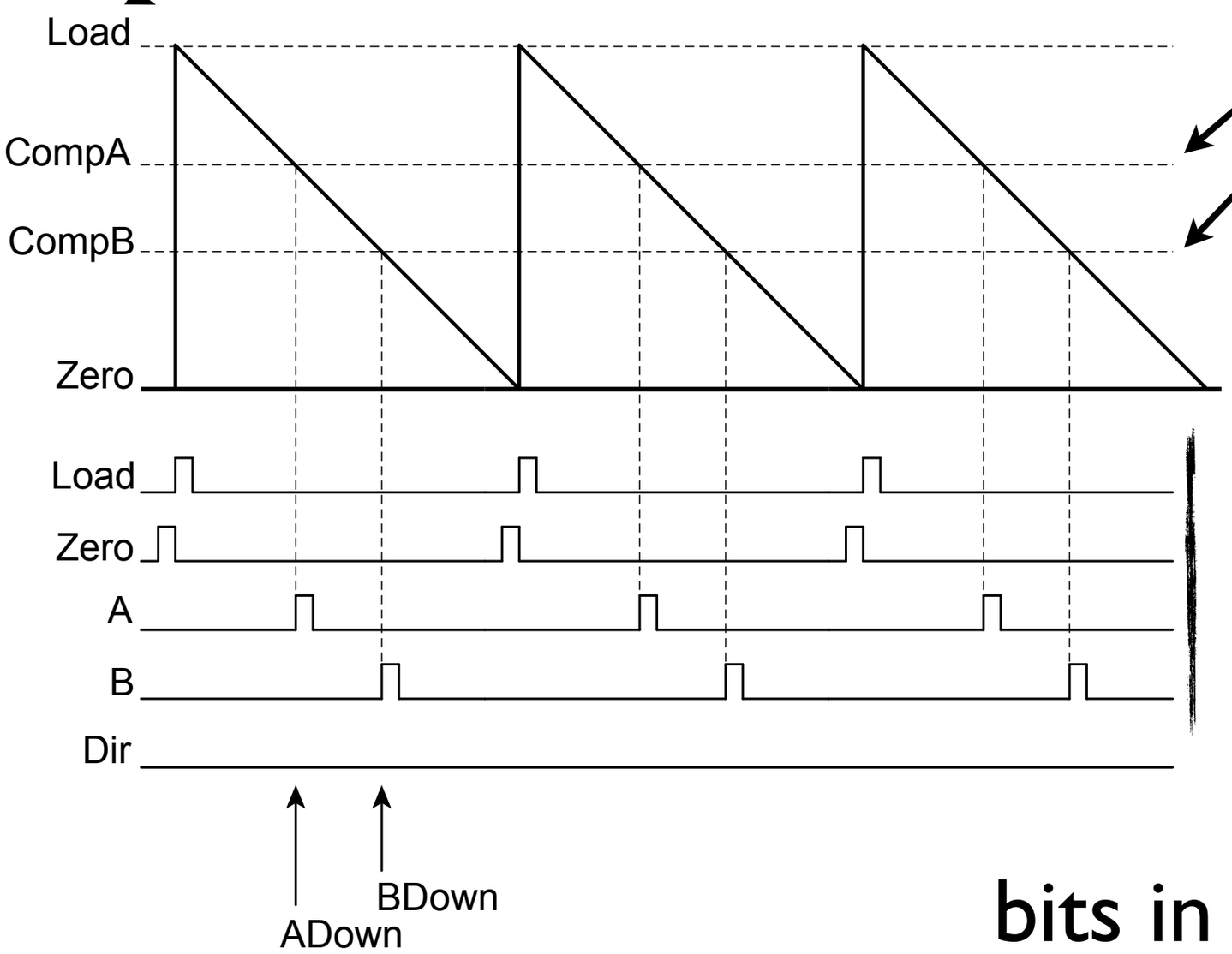
(event happens when count=comparator value)

LM3S1968 PWM countdown mode

initial timer value

comparator values

(also 16-bit)



when COUNT=XX

one-clock cycle high

(trigger some action;
e.g. interrupt)

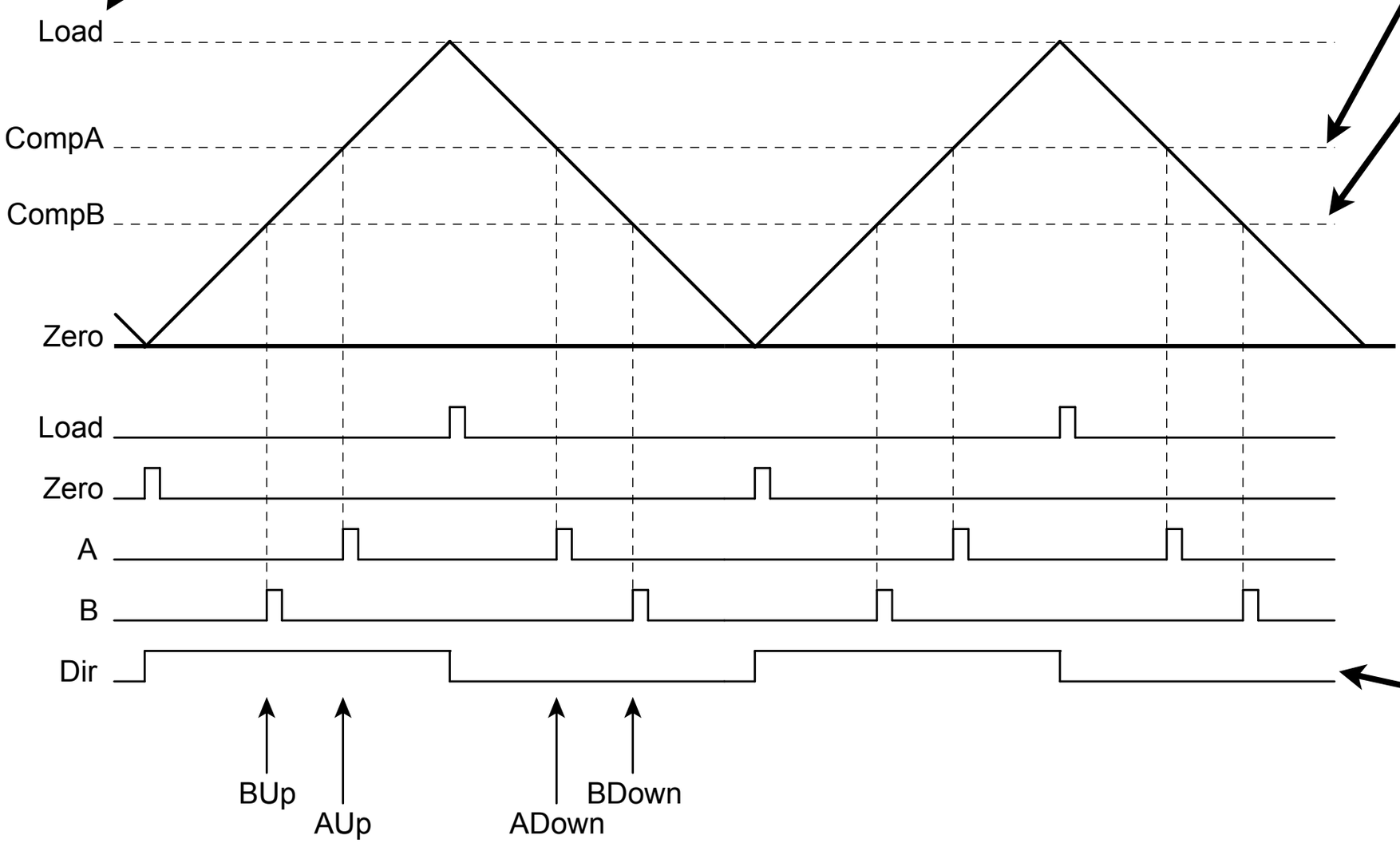
bits in `PWMxRIS`

interrupts: zero,load,match a down, match b down

LM3S1968 PWM count up-then-down mode

initial timer value

comparator values (also 16-bit)



can figure out if pwm timer counting up or down

interrupts: zero,load,match a up/down, match b up/down

LM3S1968 PWM events

what should happen when COUNT=XX

bits in `PWMxGENy`:

0x0	Do nothing.
0x1	Invert the output signal.
0x2	Set the output signal to 0.
0x3	Set the output signal to 1.

PWM0 Generator A Control (PWM0GENA)

Base 0x4002.8000
Offset 0x060
Type R/W, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ActCmpBD		ActCmpBU		ActCmpAD		ActCmpAU		ActLoad		ActZero	
Type	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

COUNT=

CompB, down

CompB, up

CompA, down

CompA, up

LOAD

ZERO

match while counting

note: each generator (A/B) can use both comparators

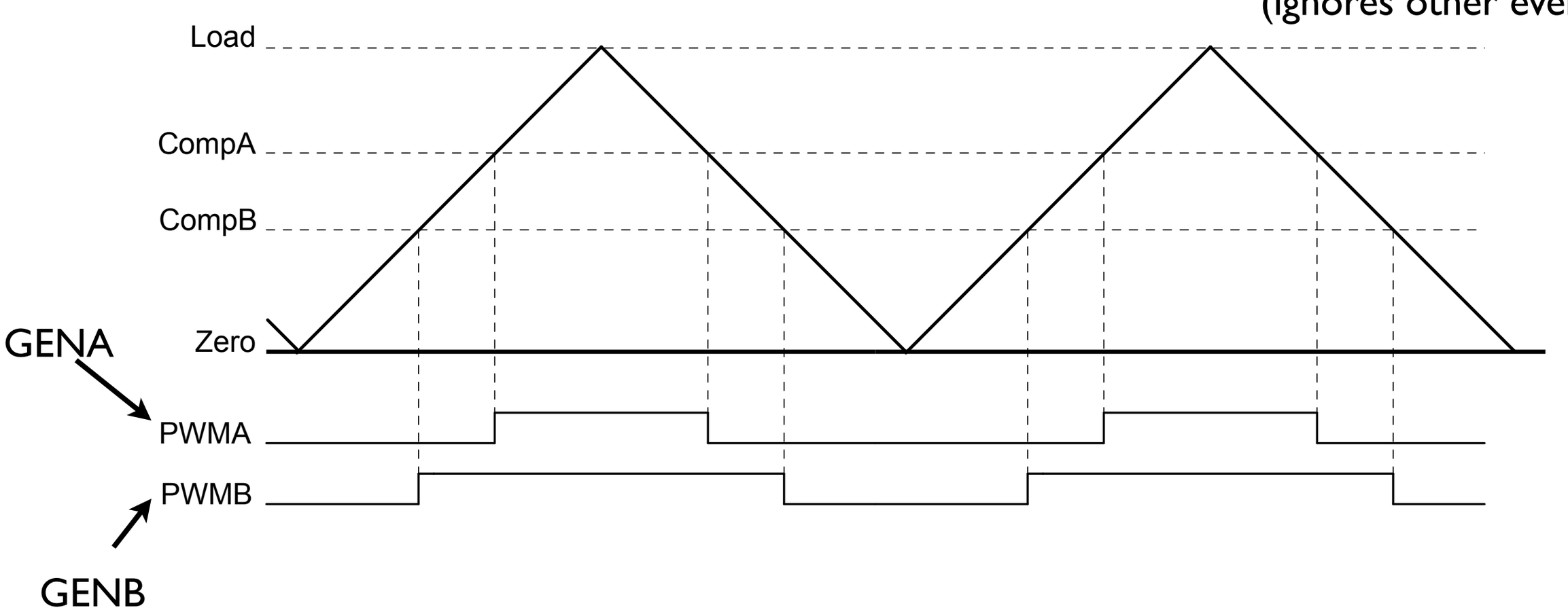
LM3S1968 PWM events

PWMA set to invert outputs when COUNT=CompAU/D

(ignores other events)

PWMB set to invert outputs when COUNT=CompBU/D

(ignores other events)



LM3S1968 PWM0, GENA configuration

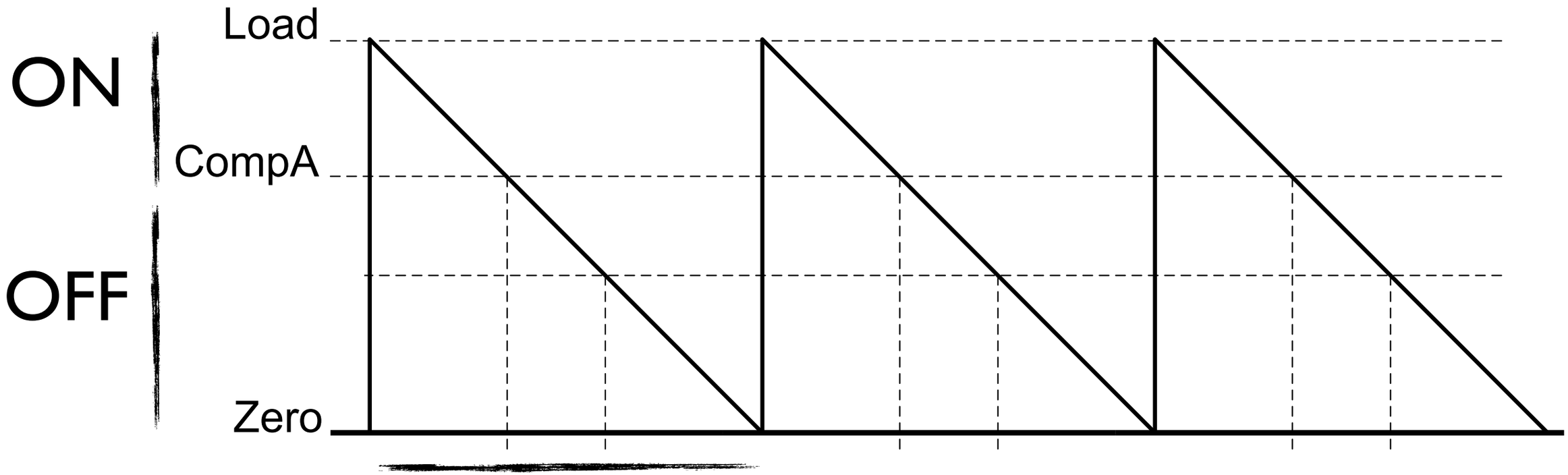
1a. enable GPIO clock
(RCGC2, p227)
1b. enable PWM peripheral clock:
(RCGC0, p212)
2. pin config: alt func, and digital enable
(AFSEL, p307; DEN, p316)
3. disable PWM0, set count mode, updates
(CTL, p580)
4. set trigger actions
(GENA, p591)
5. set init pwm0 timer value (LOAD)
(LOAD, p587)
6. set CompA/B value
(CMPA/B, p589/90)
7. enable PWM0 timer and PWM output
(CTL, p580 and ENABLE, p589)

note: each generator in a module uses the same timer value and comparator values

LM3S1968 PWM0, GENA configuration

ex: 25% DC w/25 KHz period:

(PWMClk = SysClk = 12 MHz)



T=1/25e3 = 40 us

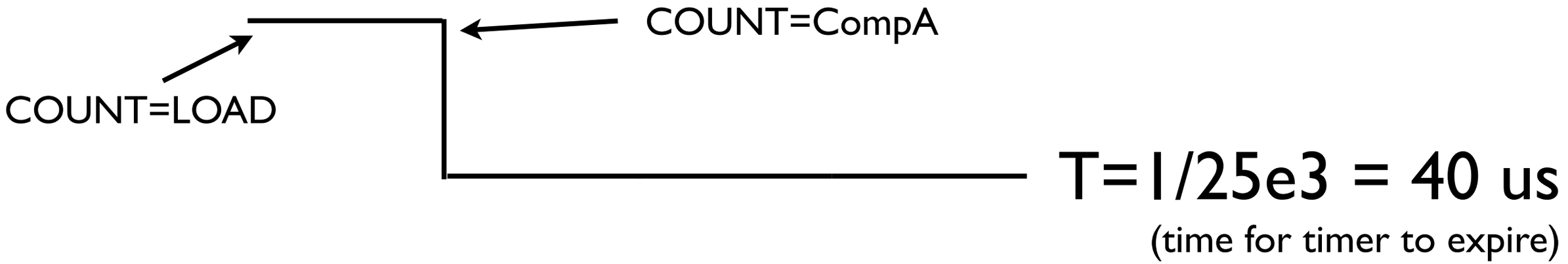
(time for timer to expire)

note: only need a single comparator

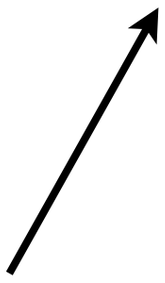
LM3S1968 PWM0, GENA configuration

ex: 25% DC w/25 KHz period:

(PWMClk = SysClk = 12 MHz)



bits in `PWMxGENy`:



0x0	Do nothing.
0x1	Invert the output signal.
0x2	Set the output signal to 0.
0x3	Set the output signal to 1.

when:

a. COUNT=LOAD, output = 1

b. COUNT= CompA, output = 0

LM3S1968 PWM0, GENA configuration

timer and comparator values:

clock cycles to get T=40 us

$$\text{TICKS} = \frac{1/f}{\frac{1}{\text{PWMClk}}} = \frac{1/25e3}{1/12e6} = 480$$

remember, just 16-bit timer

LOAD = TICKS - 1 = 479

CompA = TICKS*(1-DC%) - 1
= 480*(1 - 0.25) - 1 = 359

counting down, want to be off for 1-DC% of time