Memory-mapped External Peripherals III

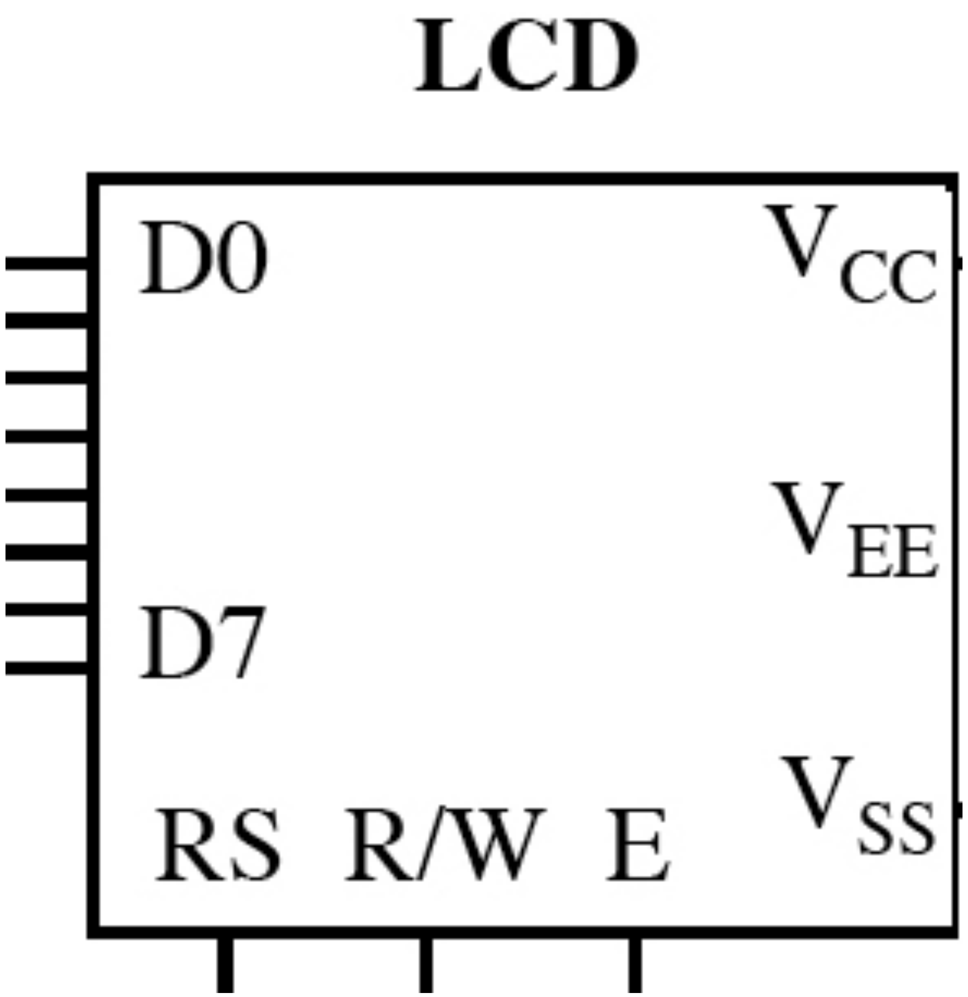
ECE 3710

Just when I discovered the meaning of life, they changed it.

- George Carlin

mm of LCD

(demultiplexed)



command or chars

`D0:D7` => ‘data’

`RS` => D0:D7 is

`0` command

`1` data

`RW` => from LCD

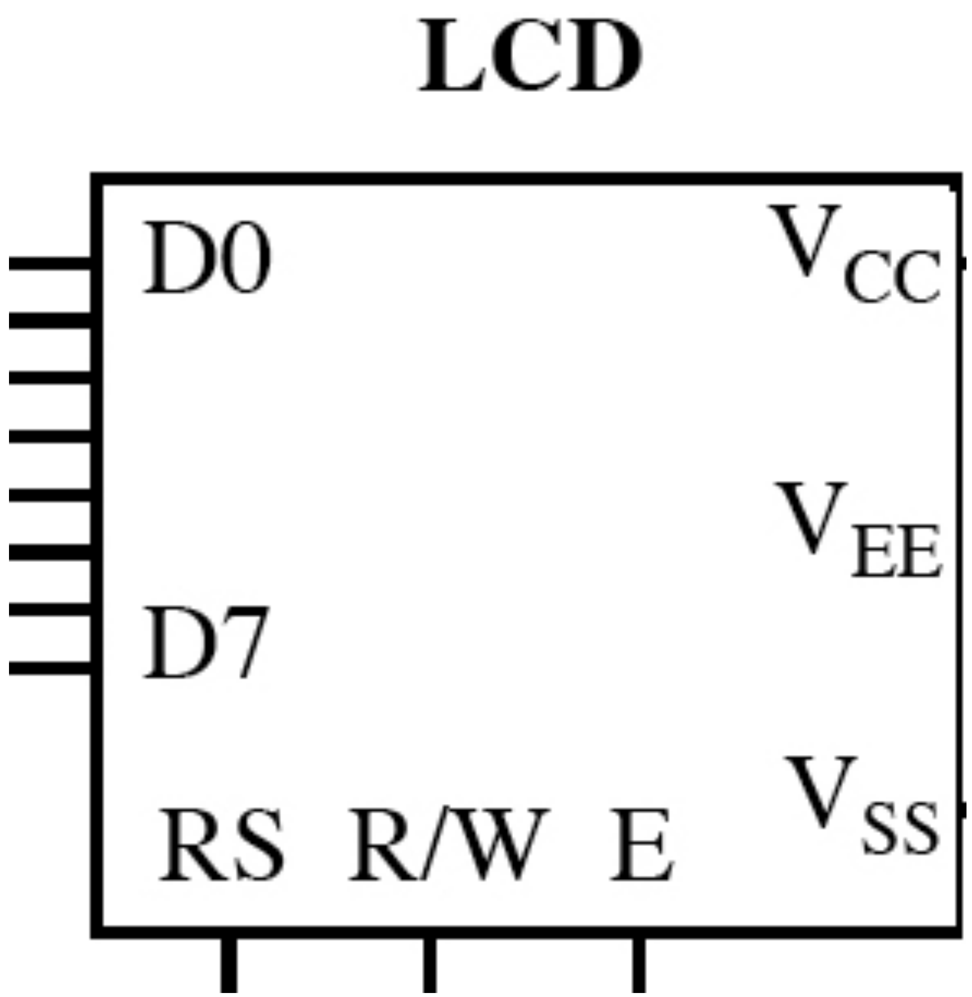
`0` write

`1` read

`E` => CS/CE

these must be set for each communication

to write command



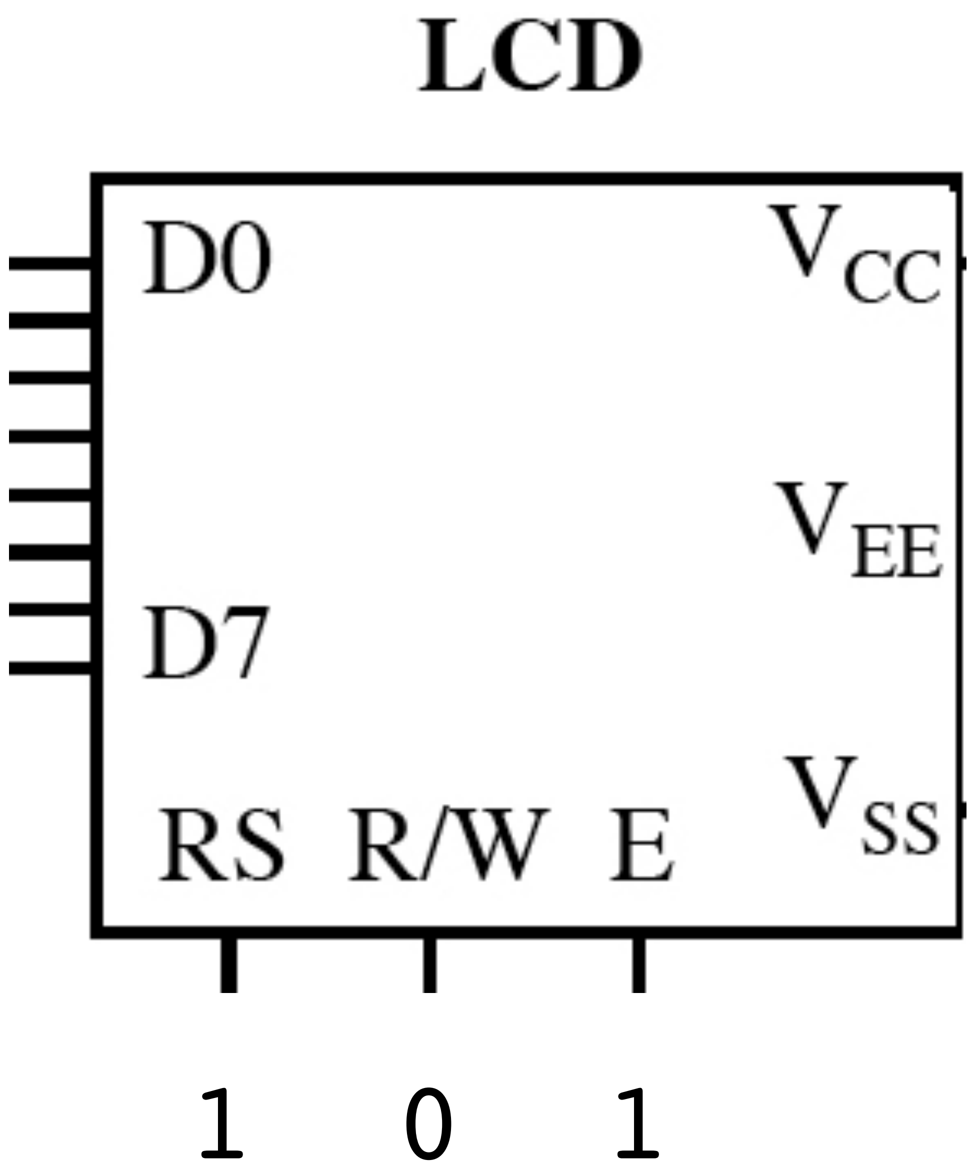
0 0 1

clear display

D[7:0]=0x01

to write data

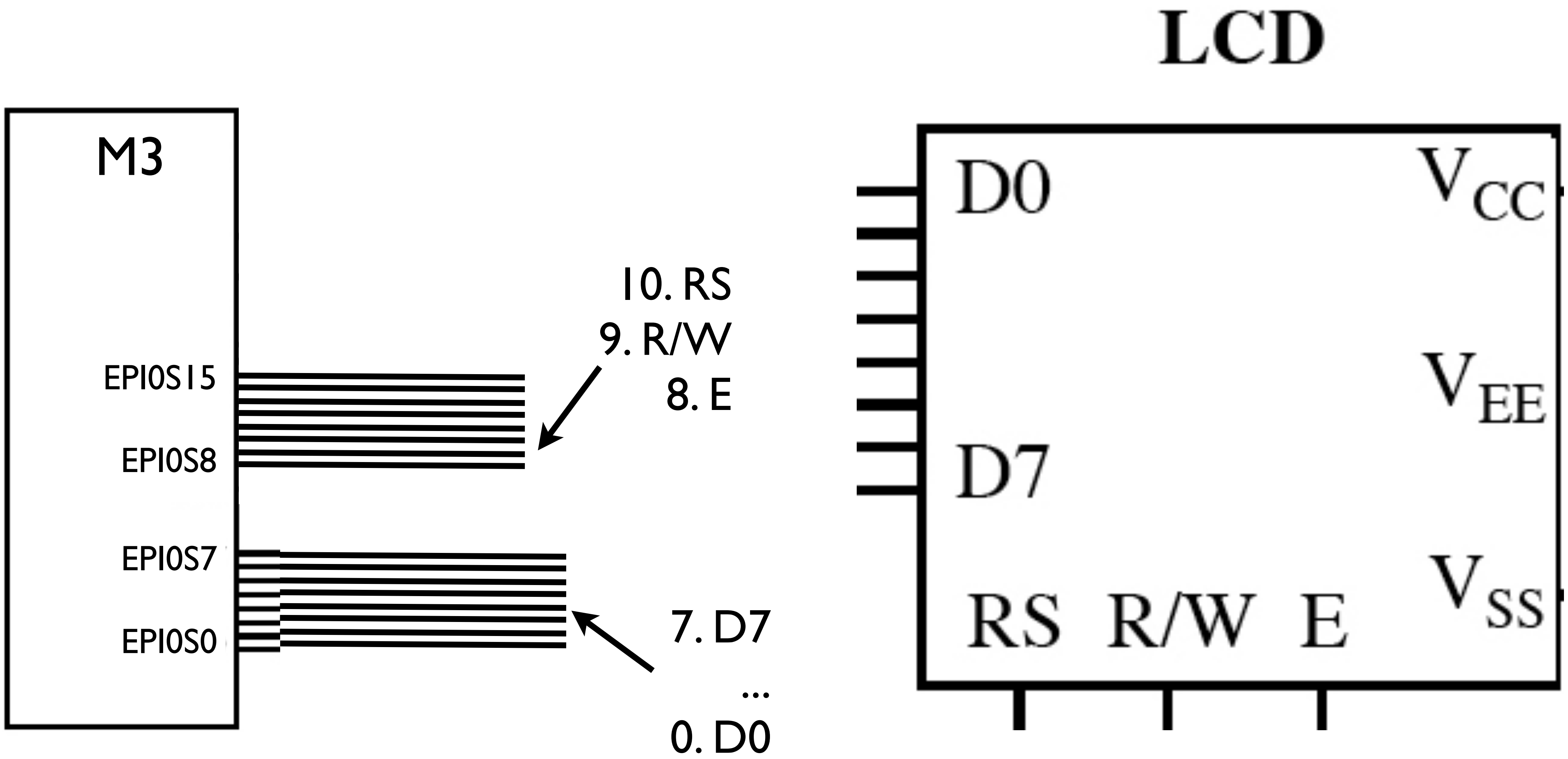
characters



character

`D[7:0]='Z'`

mm of LCD w/o multiplexing



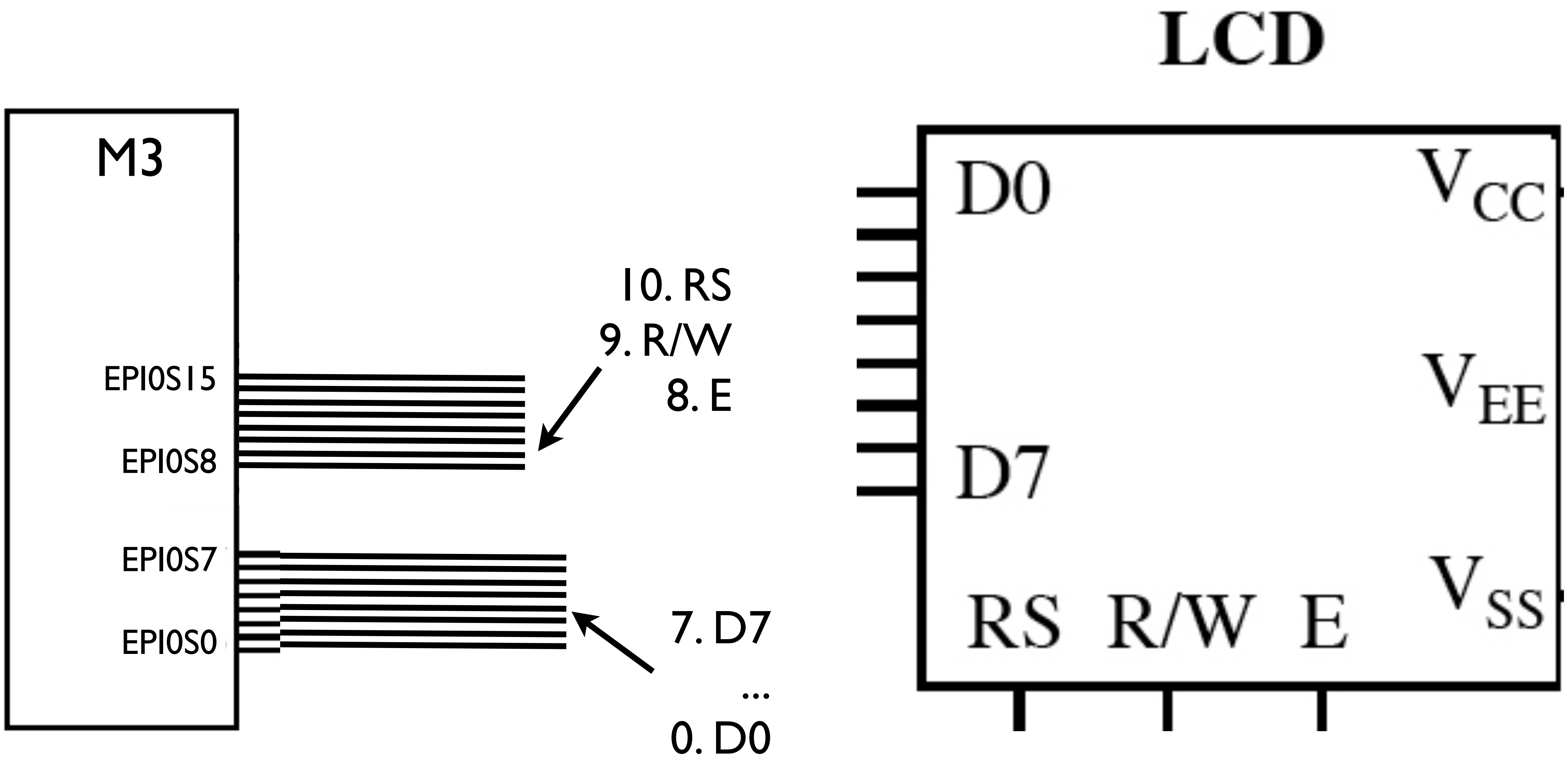
addresses:
(where we write command or char data)

data (char): 0b101 → 0x5

cmd: 0b001 → 0x1

offsets

mm of LCD w/o multiplexing




```
unsigned char LCD_CMD __attribute__((at(0xA0000001)));
unsigned char LCD_DAT __attribute__((at(0xA0000005)));
```

```
LCD_CMD = 0x1; //clear display
LCD_DAT = 'Z'; //display 'Z'
```

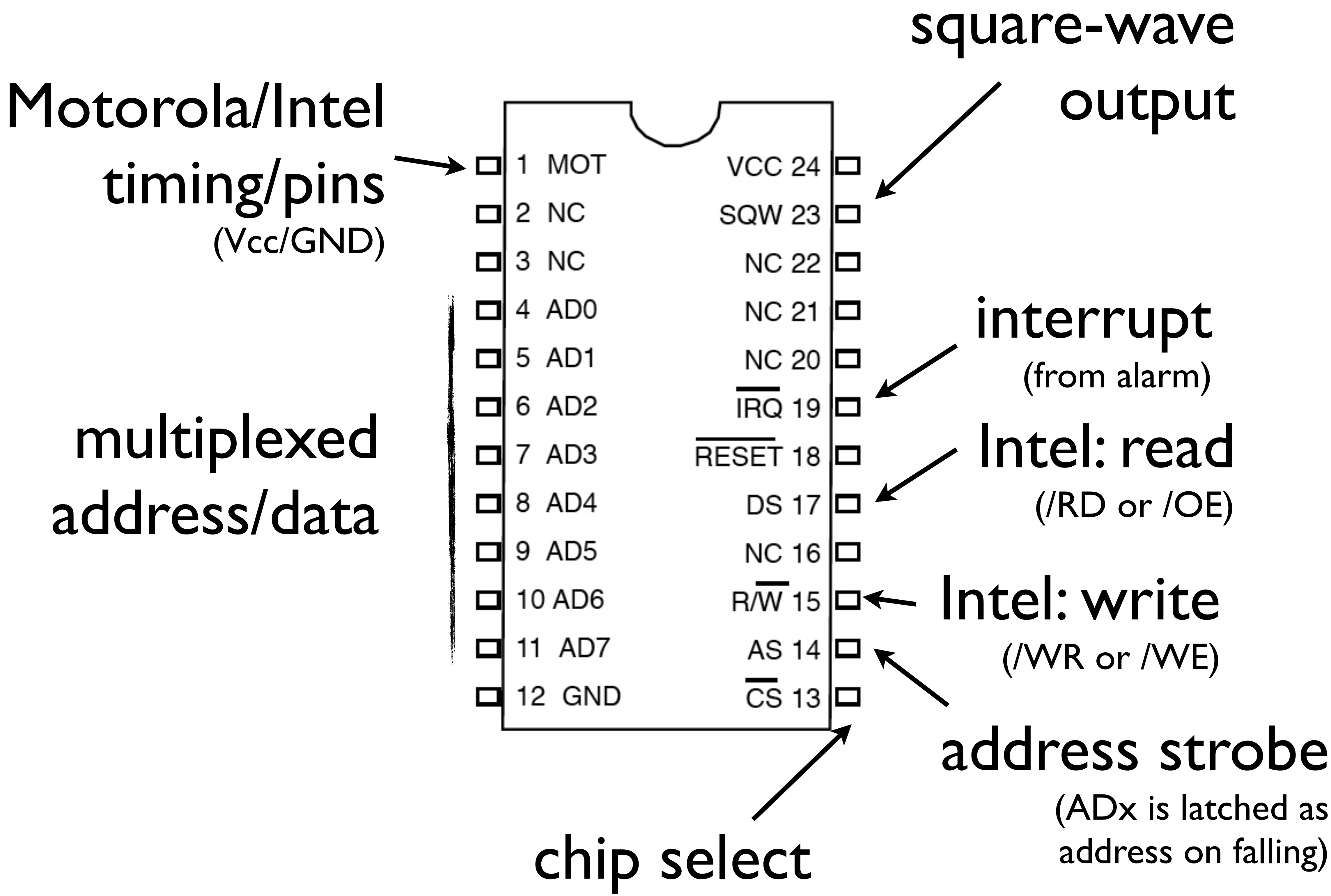
RTC

ECE 3710

Everyone has a photographic memory. Some just don't have film.

- Steven Wright

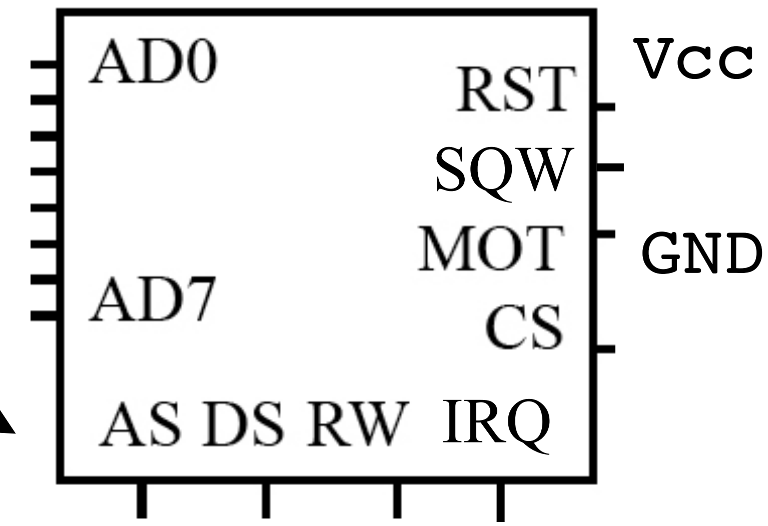
DS12887 RTC

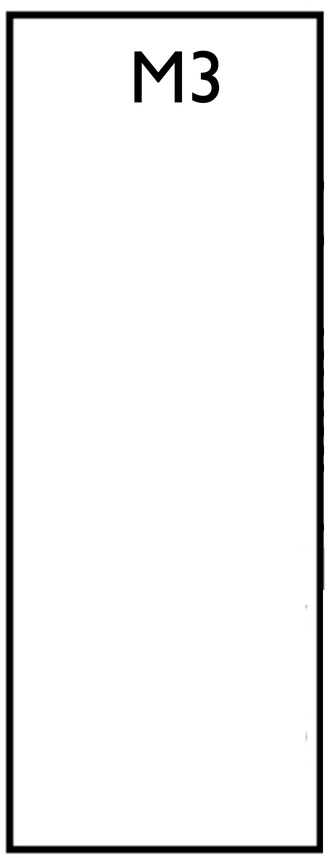


Motorola/Intel
timing/pins
(Vcc/GND)
multiplexed
address/data
1 MOT
2 NC
3 NC
4 AD0
5 AD1
6 AD2
7 AD3
8 AD4
9 AD5
10 AD6
11 AD7
12 GND
VCC 24
SQW 23
NC 22
NC 21
NC 20
IRQ 19
RESET 18
DS 17
NC 16
R/W 15
AS 14
CS 13
square-wave
output
interrupt
(from alarm)
Intel: read
(/RD or /OE)
Intel: write
(/WR or /WE)
address strobe
(ADx is latched as
address on falling)
chip select

connecting to: shared, multiplexed bus

we do our
own latching



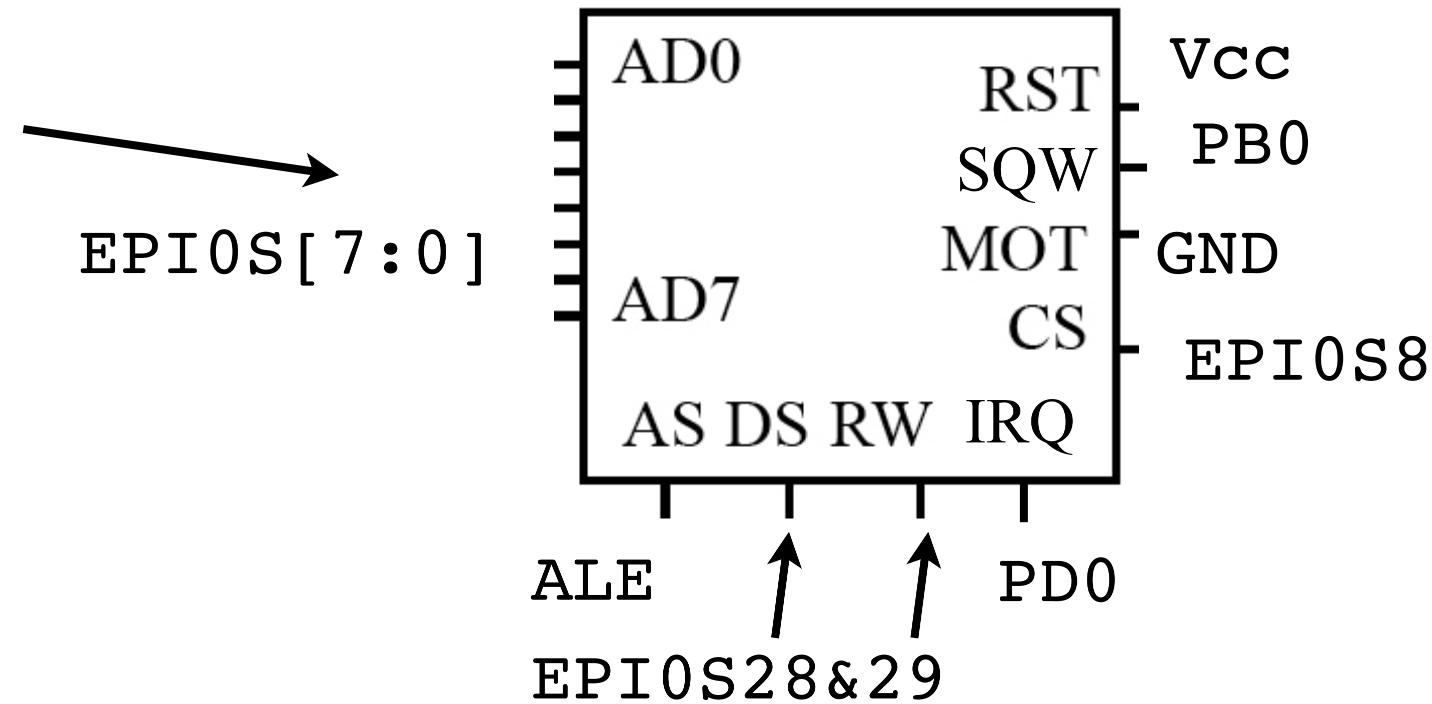


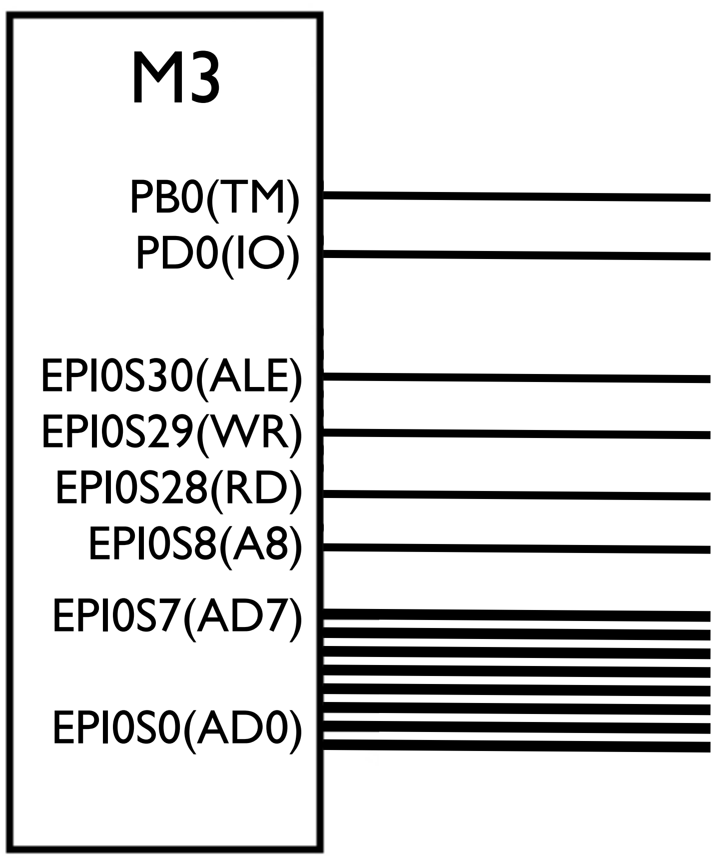
want:

1. MM
2. ext. int.
3. ext. timer

connecting to: shared, multiplexed bus

we do our own latching

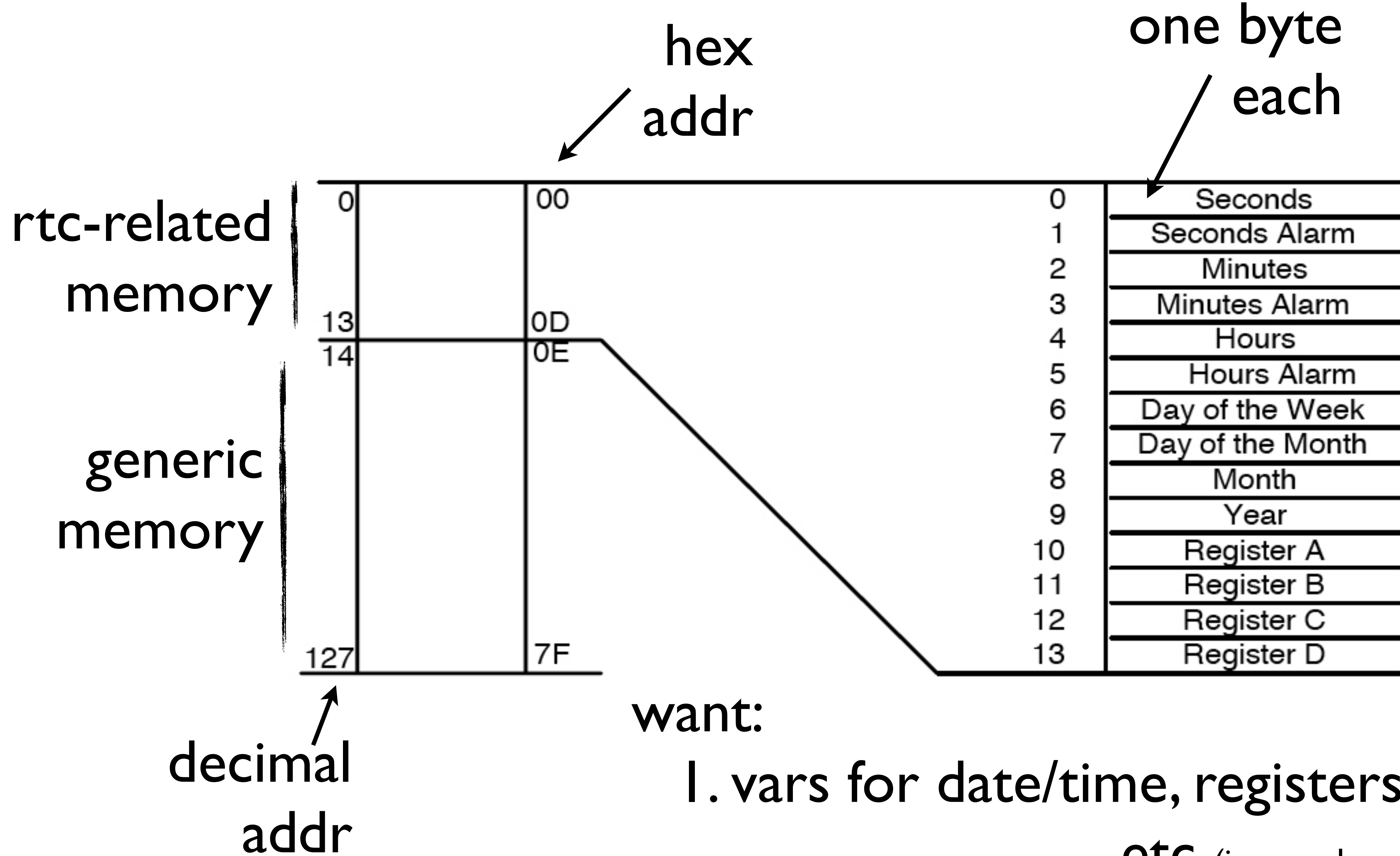




want:

1. MM
2. ext. int.
3. ext. timer

device addresses: 2^7



want:

1. vars for date/time, registers, etc (ignore alarm)
2. array for generic memory

device addresses

(CS connected to AD8)

host bus mode:
16-bit addr
8-bit data

memory map:

0b0000000100000000 --
0b0000000101111111

note: all 0's after 8th bit are don't cares

assume other devices will use bus

just for fun

(arbitrary access to device)

```
unsigned char SEC __attribute__((at(0xA0000100)));
unsigned char MIN __attribute__((at(0xA0000102)));
unsigned char HOUR __attribute__((at(0xA0000104)));
                                  ...
unsigned char D __attribute__((at(0xA000010D)));
unsigned char *RTC = (unsigned char *) 0xA0000100;
unsigned char *RTC_MEM = (unsigned char *) 0xA000010E;
```

e.g. to access hour byte:

1. HOUR=X or X=HOUR
2. RTC[4]=X or X=RTC[4]

a note on storage format

time/date stored as:

1. BCD
2. hex

12:59:35

0x0C:0x3B:0x23

binary-coded decimal (BCD)

in binary:

represent 21 by 2 and 1

0b00010101

0b0010

0b0001

regular binary representation

packed bcd:

0b00100001

a note on storage format

time/date stored as:

1. BCD
2. hex

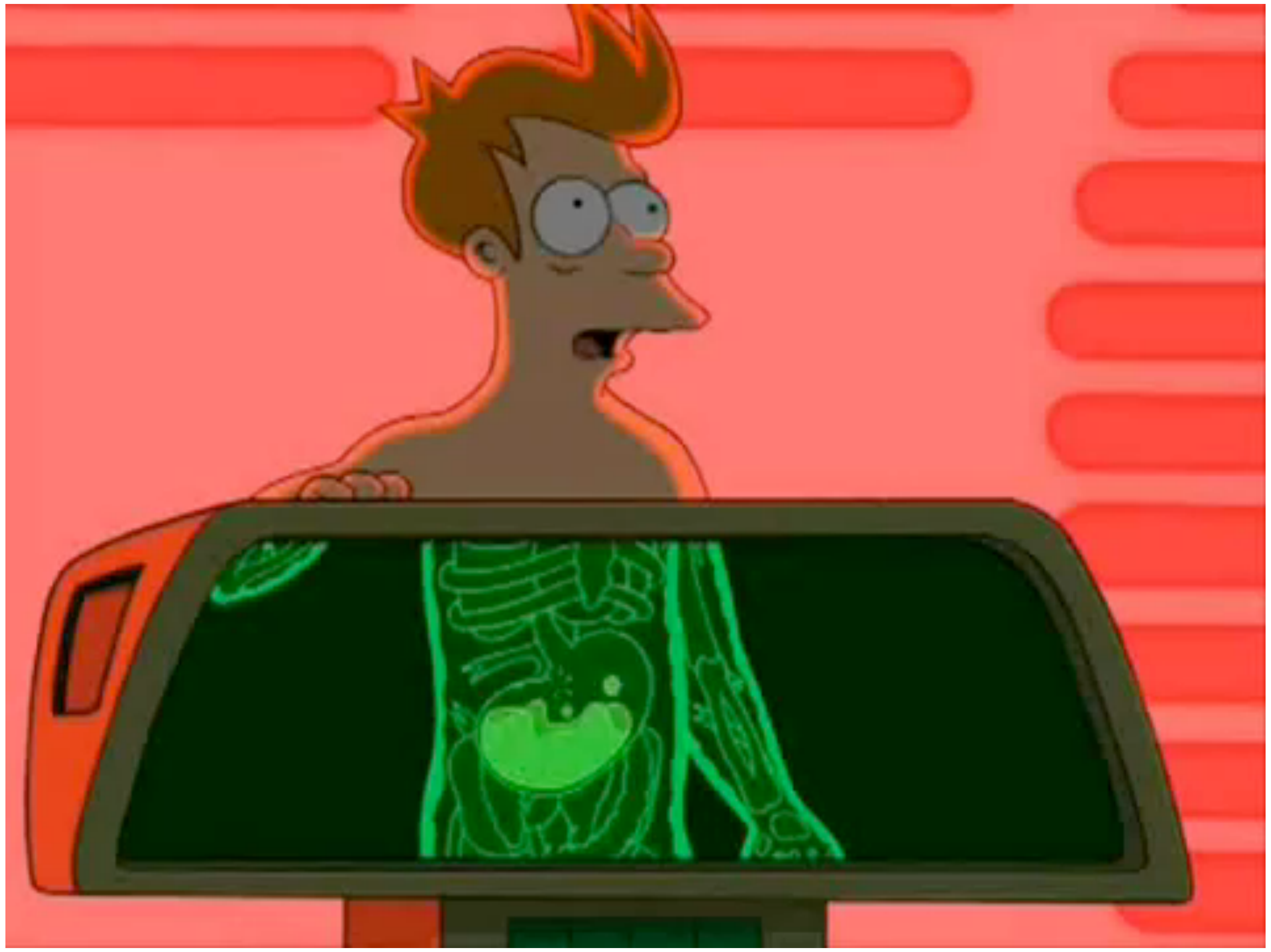
12:59:35

0b00010010:0b01011001:0b00110101

financial 'engineers'

bcd is default for this rtc ic

(useful if errors in floating point representation are intolerable) the idea of a financial ‘engineer’:



engineering: something should only blow up
when designed to...also, laws

a note on storage format

time/date stored as:

1. BCD
2. hex

12:59:35

0b00010010:0b01011001:0b00110101

0x12:0x59:0x35

ah, bcd is useful...

rtc: things to do

setup:

1. setup
2. setting time
3. getting time
4. storing data

rtc: setup

rtc: a sophisticated timer

ergo, an oscillator

register `A`:

UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP Update in progress. This is a read-only bit.

DV2	**DV1**	**DV0**	
0	1	0	will turn the oscillator on

RS3 RS2 RS1 RS0
Provides 14 different frequencies at the SQW pin. See Section 16.3 and the DS12887 data sheet.

Q: turn on rtc

rtc: setup

```
unsigned char SEC __attribute__((at(0xA0000100)));
unsigned char MIN __attribute__((at(0xA0000102)));
unsigned char HOUR __attribute__((at(0xA0000104)));
                                  ...
unsigned char D __attribute__((at(0xA000010D)));
unsigned char *RTC = (unsigned char *) 0xA0000100;
unsigned char *RTC_MEM = (unsigned char *) 0xA000010E;

void main()
{
  A = 0b00100000; //turn rtc on
}
```

great, it's oscillating

rtc: setting time

register B:

make it count, man

SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET SET = 0: Clock is counting once per second and time and dates are updated
SET = 1: Update is inhibited (during the initialization we must make SET = 1)

PIE Periodic Interrupt Enable. See Section 16.3.

AIE Alarm Interrupt Enable. The AIE = 1 will allow the IRQ to be asserted, when all three bytes of time (yy:mm:dd) are the same as the alarm bytes. See Section 16.3.

UIE See the DS12887 data sheet

SQWE Square wave enable: See Section 16.3

DM Data mode. DM = 0: BCD data format and DM = 1: Binary (hex) data format

24/12 1 for 24-hour mode and 0 for 12-hour mode

DSE Daylight Saving Enable. If 1, enables the daylight saving. (The first Sunday in April and the last Sunday of October)

hey, let me set it, man

Q: set 11:19:10
(mod 60 epoch)

rtc: setting time

```
unsigned char SEC __attribute__((at(0xA0000100)));
unsigned char MIN __attribute__((at(0xA0000102)));
unsigned char HOUR __attribute__((at(0xA0000104)));
                                  ...
unsigned char D __attribute__((at(0xA000010D)));
unsigned char *RTC = (unsigned char *) 0xA0000100;
unsigned char *RTC_MEM = (unsigned char *) 0xA000010E;

void main()
{
  A = 0x20; //0b00100000: turn rtc on
  B = 0x81; //0b10000001: let’s SET rtc time (lousy daylight savings...)
  /* time to set: 11:19:10 */
  HOUR = 0x11; //w00t, bcd!
  MIN = 0x19;
  SEC = 10;
  B = B & 0x7F; //only disable SET...now we’re counting
}
```

rtc: getting time

Q: store current time in RTC memory (start at 0x0E)

rtc: getting time

```
unsigned char SEC __attribute__((at(0xA0000100)));
unsigned char MIN __attribute__((at(0xA0000102)));
unsigned char HOUR __attribute__((at(0xA0000104)));
                                   ...
unsigned char D __attribute__((at(0xA000010D)));
unsigned char *RTC = (unsigned char *) 0xA0000100;
unsigned char *RTC_MEM = (unsigned char *) 0xA000010E;

void main()
{
  A = 0x20; //0b00100000: turn rtc on
  B = 0x81; //0b10000001: let’s SET rtc time (lousy daylight savings...)
  /* time to set: 11:19:10 */
  HOUR = 0x11; //w00t, bcd!
  MIN = 0x19;
  SEC = 10;
  B = B & 0x7F; //only disable SET...now we’re counting

  while(COWS != HOME);

  RTC_MEM[0]=HOUR;
  RTC_MEM[1]=MIN;
  RTC_MEM[2]=SEC;
}
```

success!

Q: done, right?

A: let's try one more example

(complicated: MM+interrupts)

rtc: producing square wave

wave appears on SQW pin:

1. enable
2. select frequency

SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET SET = 0: Clock is counting once per second, and time and dates are updated.
SET = 1: Update is inhibited (during the initialization we must make SET = 1).

PIE Periodic interrupt enable. If PIE = 1, upon generation of the periodic-interrupt, the IRQ pin of the DS12887 is asserted low. Therefore, IRQ becomes a hardware version of the PI bit in register C if we do not want to poll the PI bit. The rate of the periodic-interrupt is dictated by RS0 - RS3 of register A. Remember that PIE allows the generation of a hardware interrupt version of bit PI in register C and has no effect on the periodic-interrupt generation. In other words, the PIE will simply direct the PI bit of register C into the IRQ output pin.

AIE Alarm interrupt enable. If AIE = 1, the IRQ pin will be asserted low when all three bytes of the real time (hh:mm:ss) are the same as the alarm bytes of hh:mm:ss. Also, if AIE = 1, the cases of once-per-second, once-per-minute, and once-per-hour will assert low the IRQ pin. Remember that AIE allows the generation of the hardware interrupt version of the AI bit in register C and has no effect on AI generation. In other words, the AIE will simply direct the AI bit of register C into the IRQ output pin.

UIE See the DS12887 data sheet.

SQWE Square wave enable: If SQWE = 1, the square-wave frequency generated by the RS0 - RS3 options of register A will show up on the SQW output pin of the DS12877 chip.

DM Data Mode. DM = 0: BCD data format and DM = 1:binary (hex) data format

24/12 1 for 24-hour mode and 0 for 12-hour mode

DSE Daylight saving enable

register B

rtc: producing square wave

wave appears on SQW pin:

1. enable
2. select frequency

UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP Update in progress. This is a read-only bit.

DV2	DV1	DV0	
0	1	0	will turn the oscillator on

RS3	RS2	RS1	RS0	Tpi PERIODIC INTERRUPT RATE	SQW Output Freq.
0	0	0	0	None	None
0	0	0	1	3.9062 ms	256 Hz
0	0	1	0	7.812 ms	128 Hz
0	0	1	1	122.070 µs	8.192 kHz
0	1	0	0	244.141 µs	4.096 kHz
0	1	0	1	488.281 µs	2.048 kHz
0	1	1	0	976.5625 µs	1.024 kHz
0	1	1	1	1.953125 ms	512 Hz
1	0	0	0	3.90625 ms	256 Hz
1	0	0	1	7.8125 ms	128 Hz
1	0	1	0	15.625 ms	64 Hz
1	0	1	1	31.25 ms	32 Hz
1	1	0	0	62.5 ms	16 Hz
1	1	0	1	125 ms	8 Hz
1	1	1	0	250 ms	4 Hz
1	1	1	1	500 ms	2 Hz

register A

Q: stop watch (minutes+seconds)

1. stop/start and reset push buttons

a. use interrupt

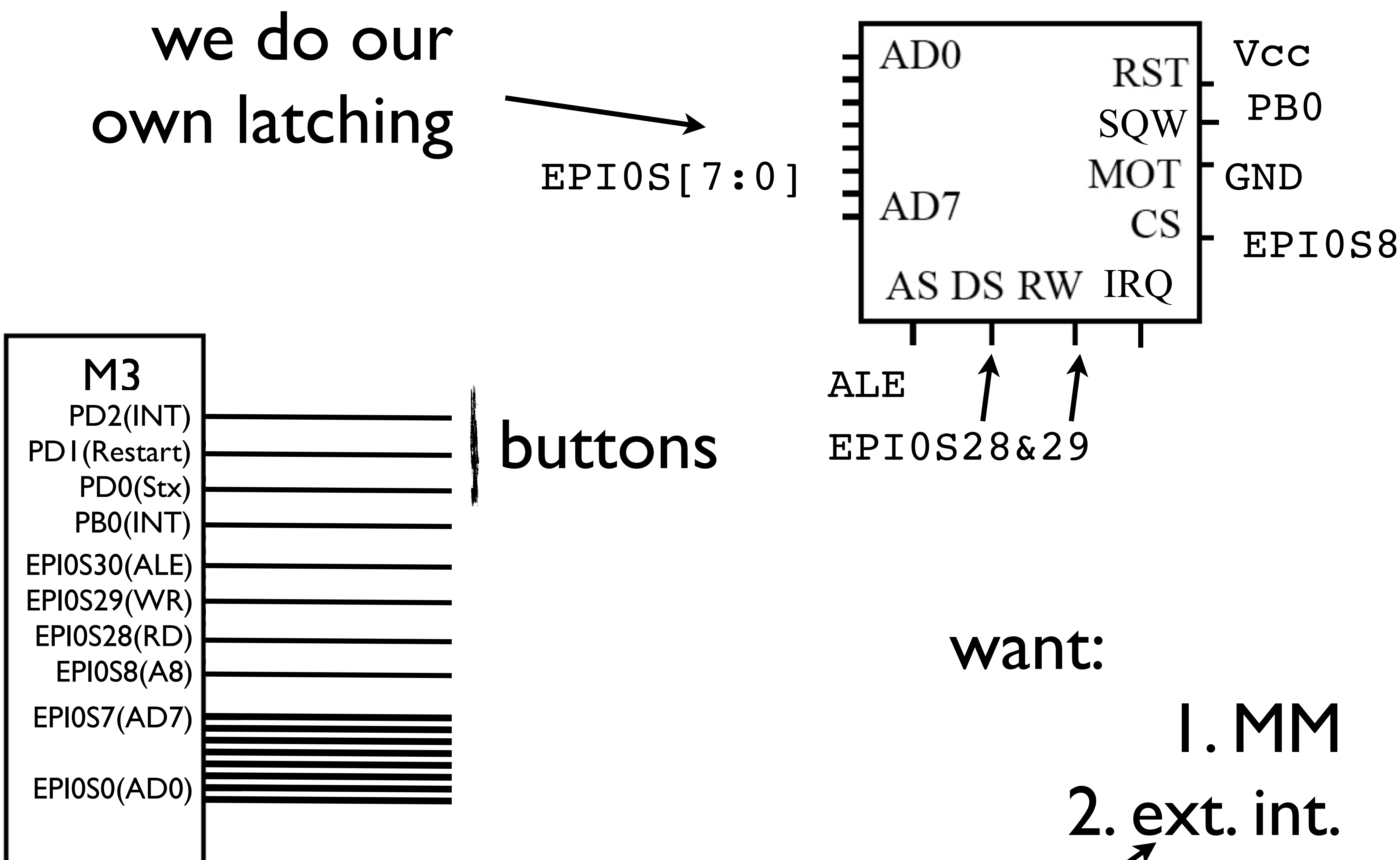
b. debouncing ← 30 ms

2. display accurate to half second

a. use interrupt

requirements

stopwatch: shared, multiplexed bus



want:

1. MM
2. ext. int.

not due to alarm, though

Q: stop watch (minutes+seconds)

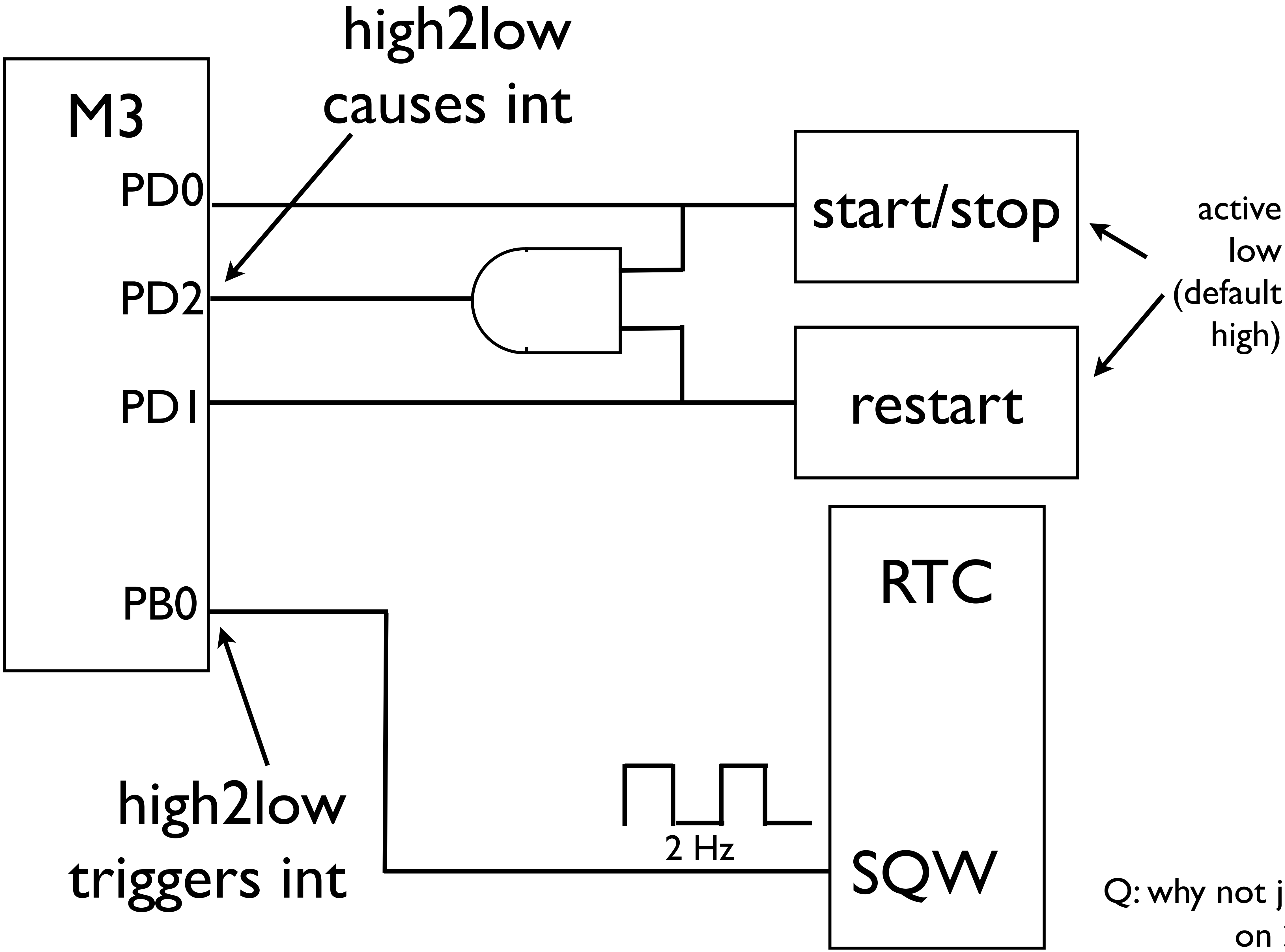
1. stop/start and reset push buttons
   a. active-low
   b. connected to `PD0--1`
   c. AND'd to `PD2`

ext. int.

2. display accurate to half second
   a. SQW
   b. PB0

ext. int.

stopwatch: connections



Q: why not just level on `PD0/1`?

rtc: stop watch

```
unsigned char SEC __attribute__((at(0xA0000100)));
unsigned char MIN __attribute__((at(0xA0000102)));
unsigned char HOUR __attribute__((at(0xA0000104)));
                                  ...
unsigned char D __attribute__((at(0xA000010D)));
unsigned char *RTC = (unsigned char *) 0xA0000100;
unsigned char *RTC_MEM = (unsigned char *) 0xA000010E;

void GPIOPortB_Handler(void); //update display
void GPIOPortD_Handler(void); //button press
void SysTick_Handler(void); //debouncing
```

idea:

1. button press: disable ext. int. one so bounces don't cause interrupt (buttons stabilised, too)
2. debounce: figure out which one and start/stop or reset rtc; re-enable ext. int.
3. update: get min and sec from rtc; send to display