

# A/D & D/A I

ECE 3710

My friend has a baby. I'm  
recording all the noises he  
makes so later I can ask him  
what he meant.

- Steven Wright

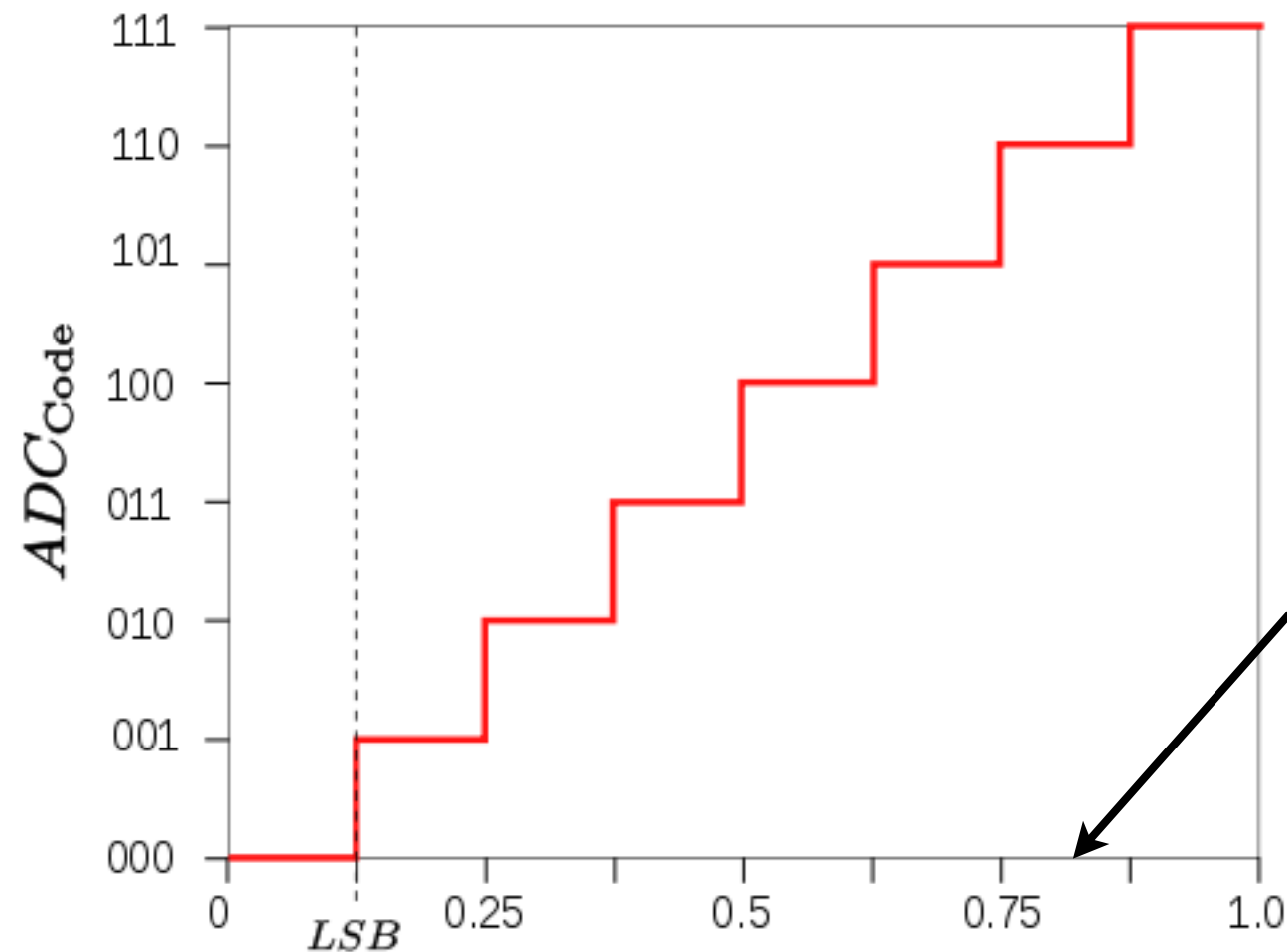
analogue-to-digital converter (ADC):  
converts analogue value (voltage or current)  
to digital value (bit)



bit pattern represents analogue value

# 3-bit ADC (voltage)

$2^3=8$  levels



step size  
 $= 1/2^3 V$   
 $= 0.125 V$

min input V


input Voltage

max input V

note:

any V between 0 V and 1 V  
is rounded to one of the 8 values

e.g. if  $0.5 \leq V_i < 0.625$ ,  
code = 100

resolution  n-bit ADC

$2^n$  codes w/  $V_{\text{max}}$  and  $V_{\text{min}}$



step size:

$$\frac{(V_{\text{max}} - V_{\text{min}})}{2^n}$$

full scale voltage ( $V_{\text{FS}}$ )

also:

sample rate how many conversions per  
second

(samples per second)

# code to voltage

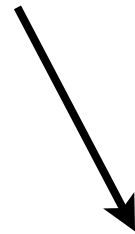
assume:

1.  $V_{\min} = 0\text{ V}$

2.  $n=3$

3.  $000 = 0\text{ V}$

if code =  $0b101 = 5$



$$\text{voltage} = 5 * (V_{\max}) / 2^3$$

# note on step size

(definition depends on ADC)



$$0b111 = 7 * 0.125 V = 0.875 V \quad 0b111 = 7 * 0.1429 V = 1 V$$

n-bit ADC:

1. code =  $\text{floor}(V_i * 2^n / (V_{\text{max}} - V_{\text{min}}))$

2. code =  $\text{floor}(V_i * (2^n - 1) / (V_{\text{max}} - V_{\text{min}}))$

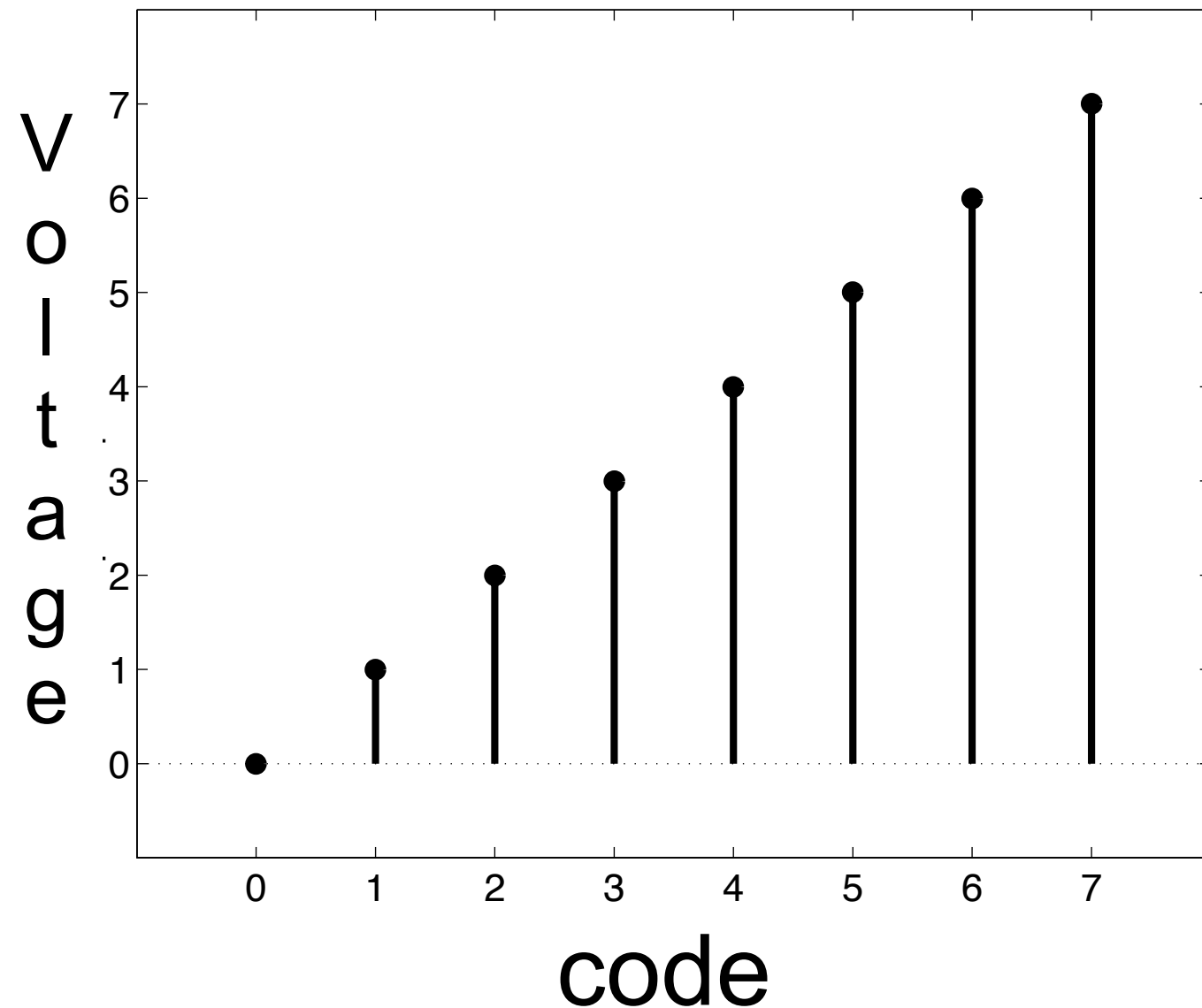
# DAC: digital-to-analogue converter

ADC: voltage  $\longrightarrow$  code

DAC: code  $\longrightarrow$  voltage



# 3-bit DAC



$$V_{\max} = 7V$$
$$V_{\min} = 0V$$
$$\text{resolution} = \frac{(7-0)}{(2^3-1)}$$

(step size/increment)

n-bit DAC:

$$1. V_i = \text{code} * (V_{\max} - V_{\min}) / 2^n$$
$$2. V_i = \text{code} * (V_{\max} - V_{\min}) / (2^n - 1)$$

# i:ADC and o:DAC

if resolution,  $V_{\text{max}}$ , and  $V_{\text{min}}$   
are equal, then:

$$\begin{aligned} \text{code} &= \text{ADC}(V_i) \\ V_o &= \text{DAC}(\text{code}) \\ &= V_i \end{aligned}$$

code equals same voltage for both

## ADC basics:

1. tell ADC to convert input (voltage2code)
2. wait for conversion to complete
3. get code (and do with it what you will)

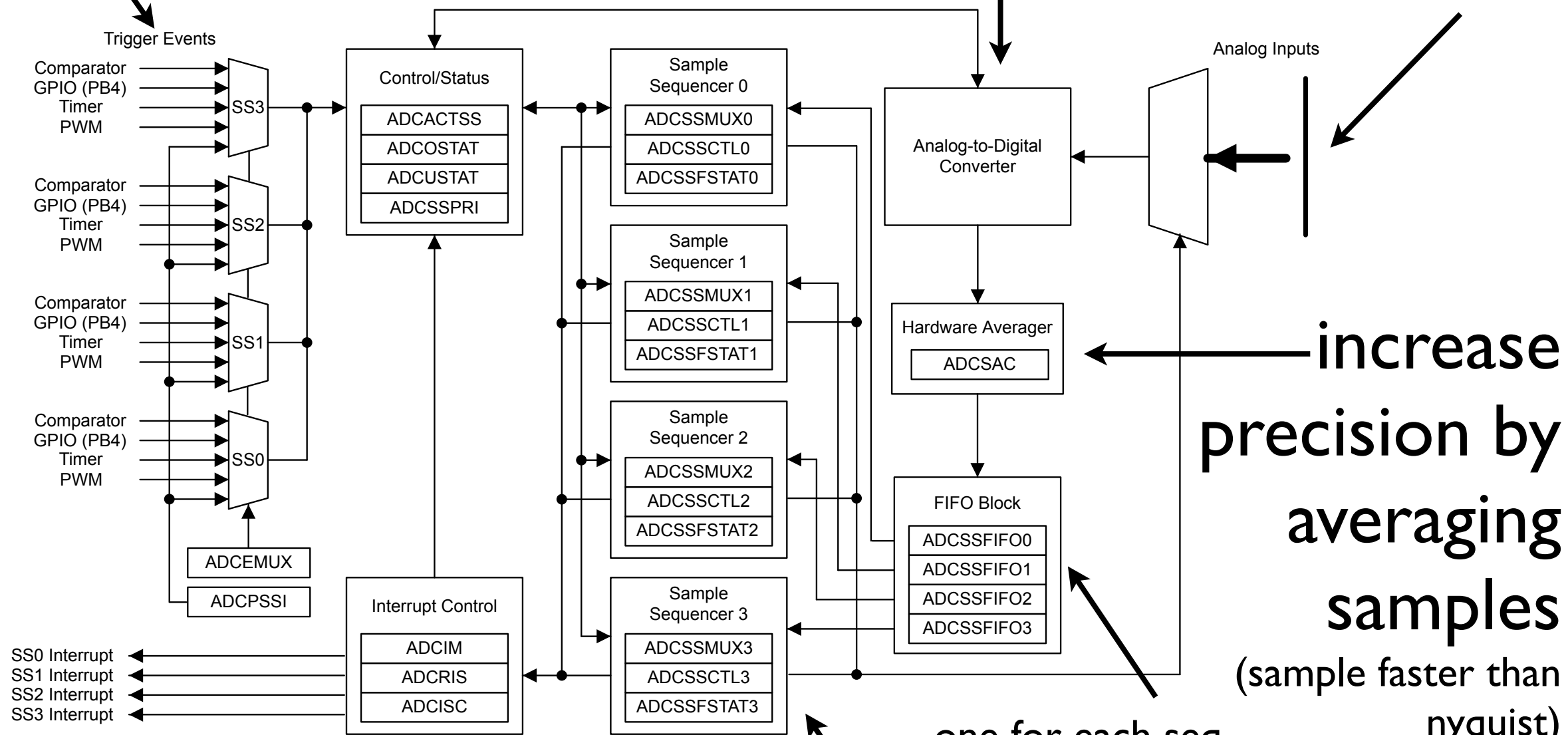
(somewhat typical)

# what causes ADC to begin conversion

# a single A/D

(we switch between inputs)

# ADC[0--7]



each sequencer has  
own IRQ

multiple sequencers do this with  
different priorities

one sequencer can: sample inputs  
(different or the same) **serially**

# LM3S1968 ADC config

- 1. enable clock for ADC**
2. set max sample rate
3. set priority of sequencers  
(sample multiple signals sequentially)
- 4. disable ADC while configuring**
- 5. set trigger event**
- 6. select ADC channel source**
7. select end of sample sequence
8. enable/disable interrupt for sample sequence
- 9. enable/disable interrupt for sequencer**
- 10. enable ADC**
- 11. enable interrupts for ADC{X}**

bolded entries are typical for generic ADC config  
(i.e. you can ignore sequencer stuff)

# LM3S1968 ADC config



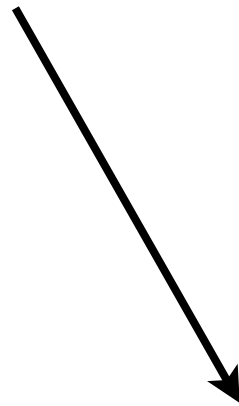
single input w/ADC0 using SEQUENCER0

(i.e. nothing fancy, just give me the voltage of a single input)



initiate conversion periodically

(when timer expires)



Q: how should we be notified that  
conversion has finished?

1. polling
2. interrupts

# should we use interrupts or polling?

whoever says 'interrupts':



## ADC ISR:

1. have SysTick ISR initiate conversion
2. ADC ISR moves code to R4

ADC details:

$n=10$

$V_{\min}=0V$

$V_{\max}=3V$

$V_{\text{inc}} = 3/(2^n - 1)$



# single input w/ADC0 using SEQUENCER0

(i.e. nothing fancy, just give me the voltage of a single input)

**; 1. activate clock for adc0 peripheral**

```
LDR R1,=SYSCTL
```

```
MOV R0,#0x10000 ;ADC is bit 16 of RCGC0 (offset 0x100)
```

```
STR R0,[R1,#0x100]
```

```
NOP ;system clock takes a while...
```

```
NOP ;to get going...
```

**; 2. specify maximum sampling rate (r-m-w cycle)**

**;default is 125 KS/s (enough)**

**; 3. set sequencer priorities**

**;default is sequencer zero has highest priority**

**; (only sampling on one channel)**

**; 4. disable adc0 while configuring it**

```
ldr R1,=ADC
```

```
mov R0,#0
```

```
str R0,[R1,#0x0]
```

**; 5. set trigger event**

```
mov R0,#0 ;software trigger for sequencer zero
```

```
str R0,[R1,#0x14]
```

# single input w/ADC0 using SEQUENCER0

(i.e. nothing fancy, just give me the voltage of a single input)

**; 6. set source for sequencer (select channel)**

```
mov R0,#0 ;select adc0
```

```
str R0,[R1,#0x0]
```

**; 7. select end of sample sequence (we only want one sample in sequence)**

**; 8. set interrupt for first sample (same register as step seven)**

**; (enable interrupt, first sample is end of sequence, single input)**

```
mov R0,#0x6 ;0x6=0b110
```

```
str R0,[R1,#0x44]
```

**; 9. enable interrupt for first sequencer**

```
mov R0,#1
```

```
str R0,[R1,#0x8]
```

**; 10. enable adc**

```
mov R0,#1
```

```
str R0,[R1,#0x0]
```

**; 11. enable interrupts for adc0 (adc0 is bit 14)**

```
ldr R1,=M3CP
```

```
mov R0,#0x4000
```

```
str R0,[R1,#0x100]
```

# single input w/ADC0 using SEQUENCER0

(i.e. nothing fancy, just give me the voltage of a single input)

## SysTick\_Handler

```
; trigger conversion
```

```
ldr R1,=ADC
```

```
mov R0,#1
```

```
str R0,[R1,#0x28]
```

```
bx LR
```

## ADC0\_Handler

```
; acknowledge interrupt
```

```
ldr R1,=ADC
```

```
mov R0,#1
```

```
str R0,[R1,#0xC]
```

```
; copy adc code to R4
```

```
ldr R4,[R1,#0x48]
```

```
bx LR
```

for accurate periodic sampling:

1. use timer directly to initiate conversion

(no need for timer ISR)

2. use main oscillator/PLL

(internal oscillator too inaccurate)

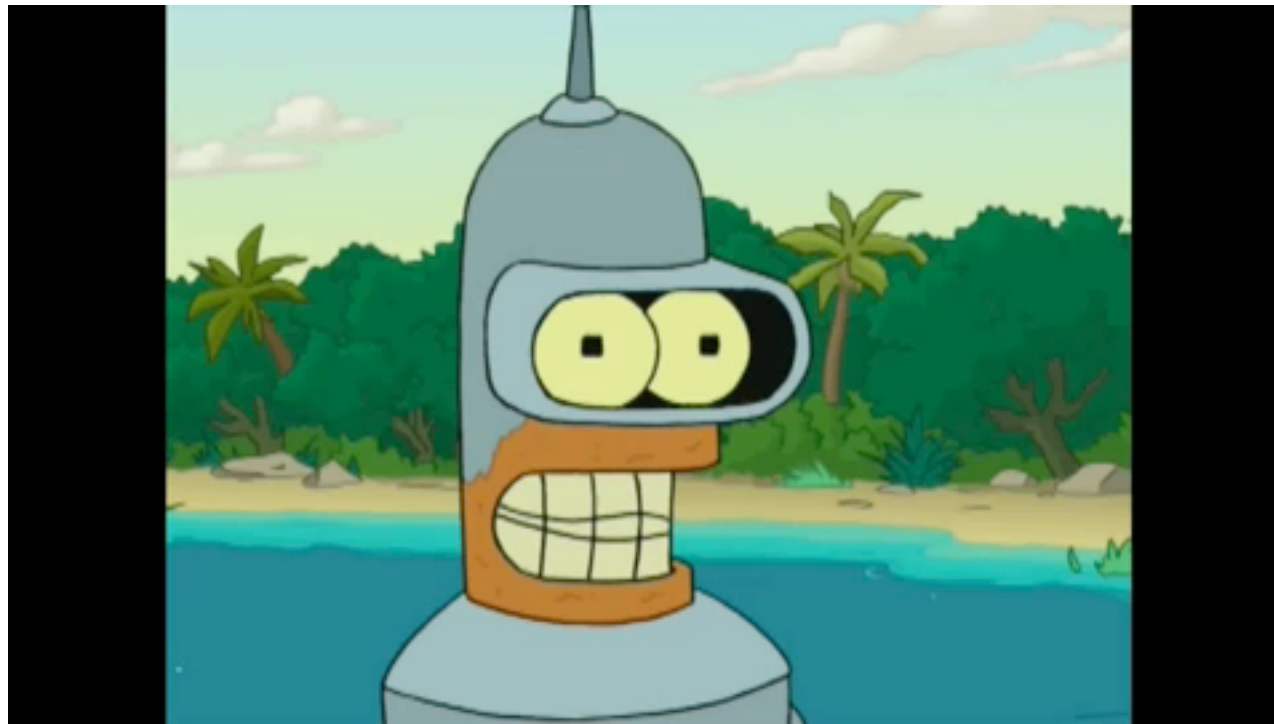
also, can do continuous sampling

(limited sample rates, though)

# LM3S1968 DAC

**note: LM3S1968 has no built-in DAC**

no DACs on any M3/4 uC,TI?

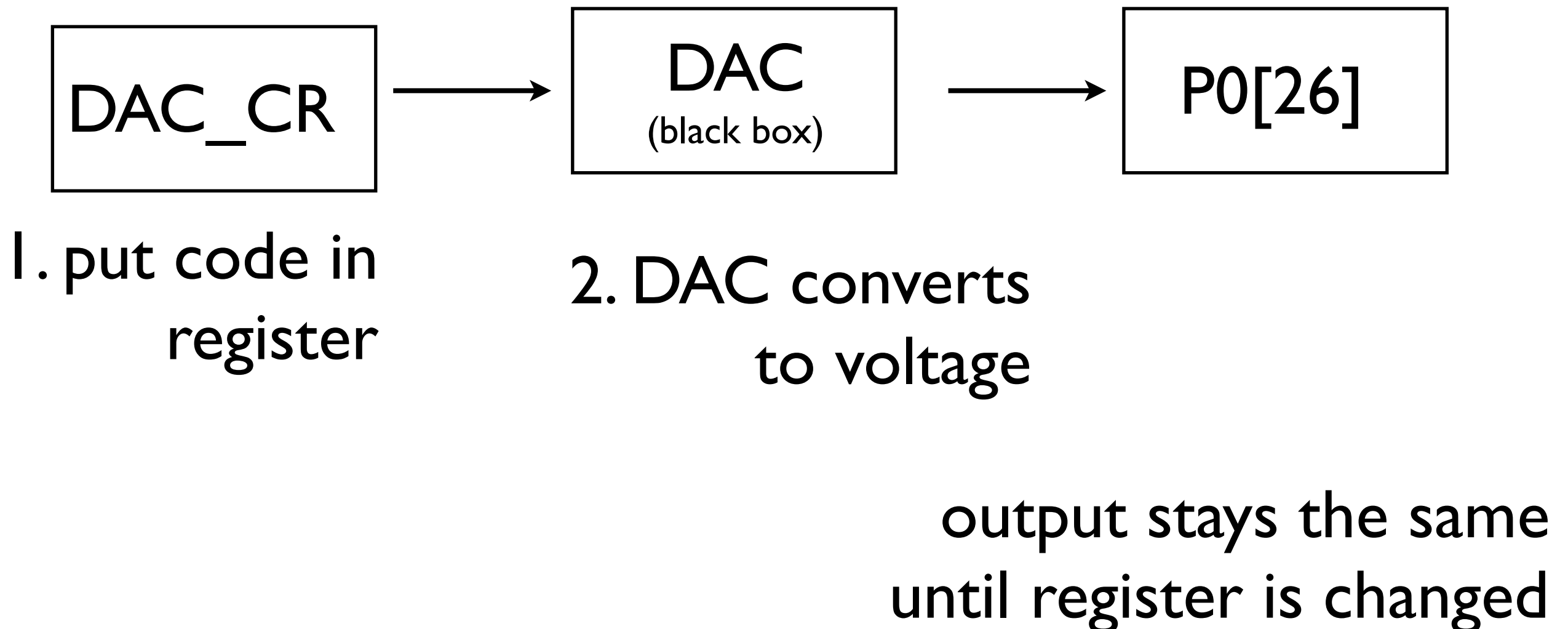


# NXP I778 uC DAC

is simulated  
in Keil uVision

how to think about DAC:  
(for our purposes)

3. voltage appears  
on P0, pin 26



# NXP I778 uC DAC

10-bit  
DAC code  
stored in  
bits 15:6 of  
register

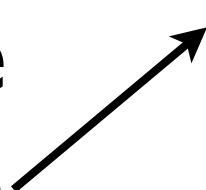


Table 681: D/A Converter Register (CR - address 0x4008 C000) bit description

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved. Read value is undefined, only zero should be written.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the DAC_OUT pin (with respect to V <sub>SSA</sub> ) is $VALUE \times ((V_{REFP} - V_{REFN})/1024) + V_{REFN}$ .	0
16	BIAS		Settling time The settling times noted in the description of the BIAS bit are valid for a capacitance load on the DAC_OUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time. One or more graphs of load impedance vs. settling time will be included in the final data sheet.	0
		0	The settling time of the DAC is 1 $\mu$ s max, and the maximum current is 700 $\mu$ A. This allows a maximum update rate of 1 MHz.	
		1	The settling time of the DAC is 2.5 $\mu$ s and the maximum current is 350 $\mu$ A. This allows a maximum update rate of 400 kHz.	
31:17	-		Reserved. Read value is undefined, only zero should be written.	NA

DAC details:

n=10

V<sub>min</sub>=0V

V<sub>max</sub>=3.3V

V<sub>inc</sub> =  $3.3/(2^n)$



## ex: DAC ramp

```
int main(void)
```

```
{
```

```
    // simulate an 8-bit dac ← bits 1:0 of 10-bit DAC=0  
    unsigned char code = 0;
```

```
    DACInit();
```

```
    // want an infinite loop
```

```
    for(code=0;code<256;code++)
```

```
        DAC_CR = ((short)code<<8);
```

```
}
```

cast code as short so we can  
shift MSB of code to bit 16 of DAC\_CR