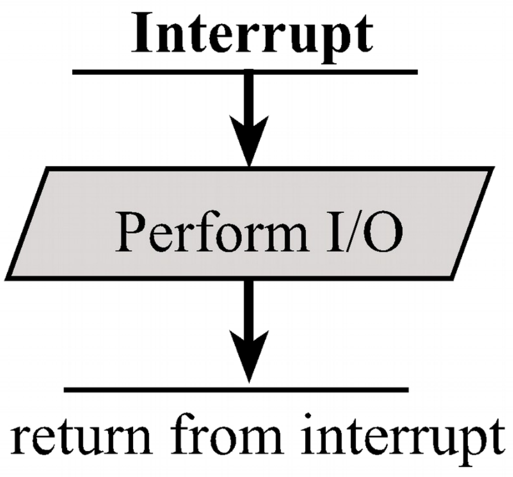
# COMP-462
# Embedded Systems

## Lecture 8: Periodic Timer Interrupts, Digital-to-Analog Conversion, Sound

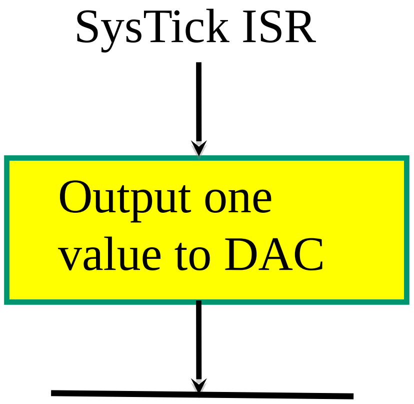
**Interrupts: read Sections 6.1 to 6.4**
**Sound: read Sections 6.5 to 6.7**

# Agenda

- ❑ Recap
  - ❖ PLL
  - ❖ Data structures
  - ❖ FSMs, linked structure
  - ❖ Interrupts

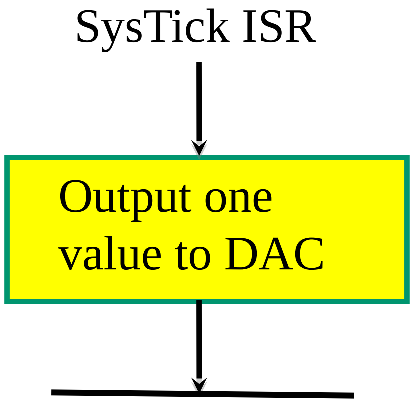


- ❑ Agenda
  - ❖ Periodic Interrupts
  - ❖ Digital to Analog Conversion
  - ❖ Nyquist Theorem
  - ❖ Sound generation



# Interrupts

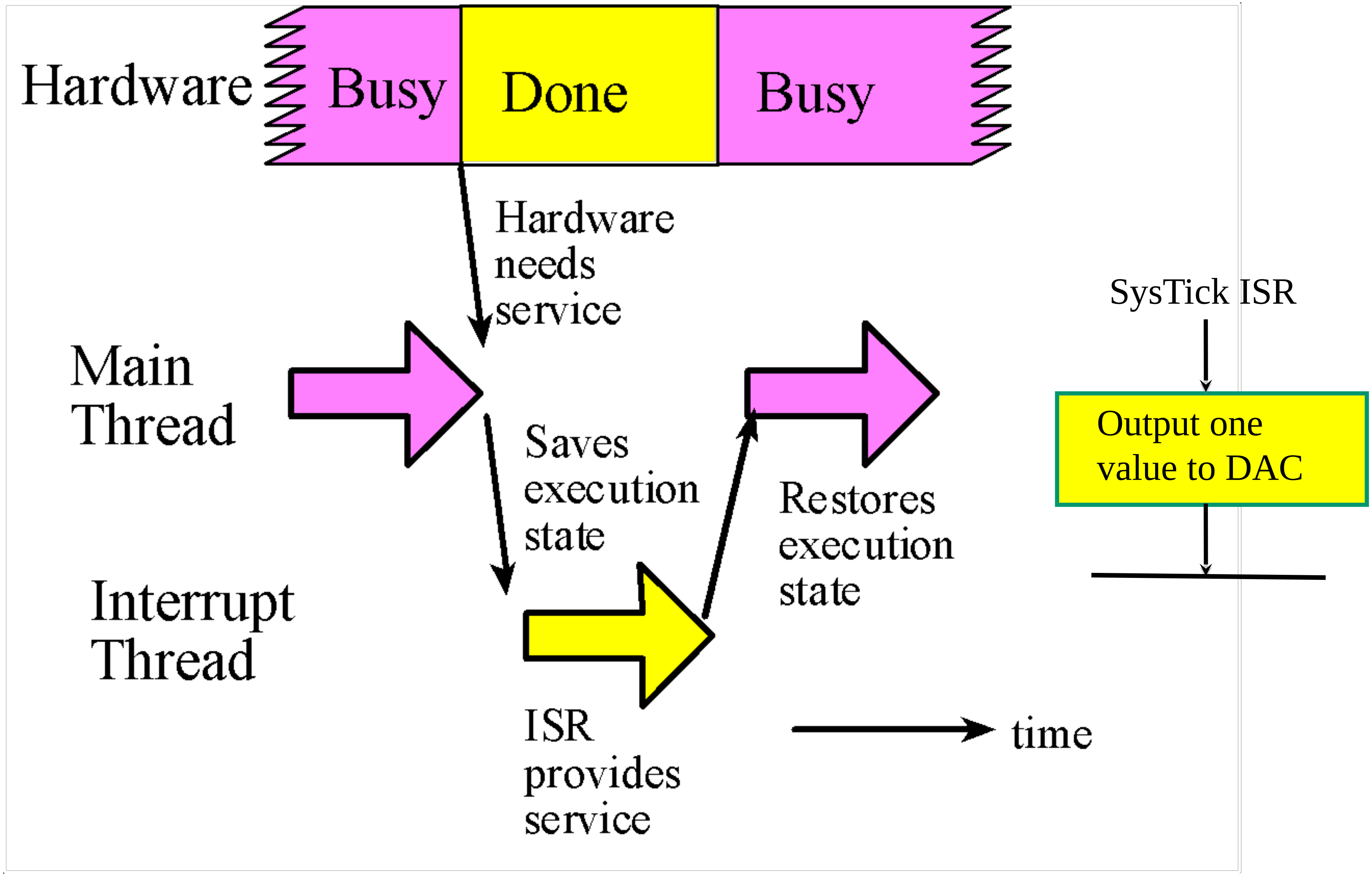
- ❑An **interrupt**
  - ❖Automatic transfer of software
  - ❖In response to a hardware event
- ❑Examples
  - ❖Periodically: output to DAC making sound
  - ❖New input: receive new data
  - ❖Output is idle: send more data



# Interrupt Events

- ❑ Respond to infrequent but important events
  - ❖ Alarm conditions like low battery power
  - ❖ Error conditions
- ❑ I/O synchronization
  - ❖ Trigger interrupt when signal on a port changes
- ❑ Periodic interrupts
  - ❖ Generated by the timer at a regular rate
  - ❖ Systick timer can generate interrupt when it hits zero
  - ❖ Reload value + frequency determine interrupt rate

# What are threads?



# ARM Cortex-M Interrupts

- ❑ **Arm** bit (also called enable)
  - ❖ Separate arm bit for each source
  - ❖ Software initializes arm bit to 1 to allow interrupts

NVIC_ST_CTRL_R, bit 1

- ❑ Trigger **flag**
  - ❖ hardware sets trigger when it wishes to request an interrupt
  - ❖ software clears the trigger to signify processing done

Count flag is in NVIC_ST_CTRL_R, bit 16

- ❑ Interrupt **priority** (0=max to 7=lowest)
  - ❖ Higher priority interrupt can suspend a lo
  - ❖ Lower/equal priority interrupt will wait until higher is done

NVIC_PRI3_R, bits 31-29

- ❑ Interrupt **enable** (I bit)
  - ❖ Global interrupt enable bit, I, in PRIMAS
  - ❖ To enable (I=0), execute **`CPSIE I EnableInterrupts();`**
  - ❖ To disable (I=1), execute **`CPSID I DisableInterrupts();`**

PRIMASK, bit 0

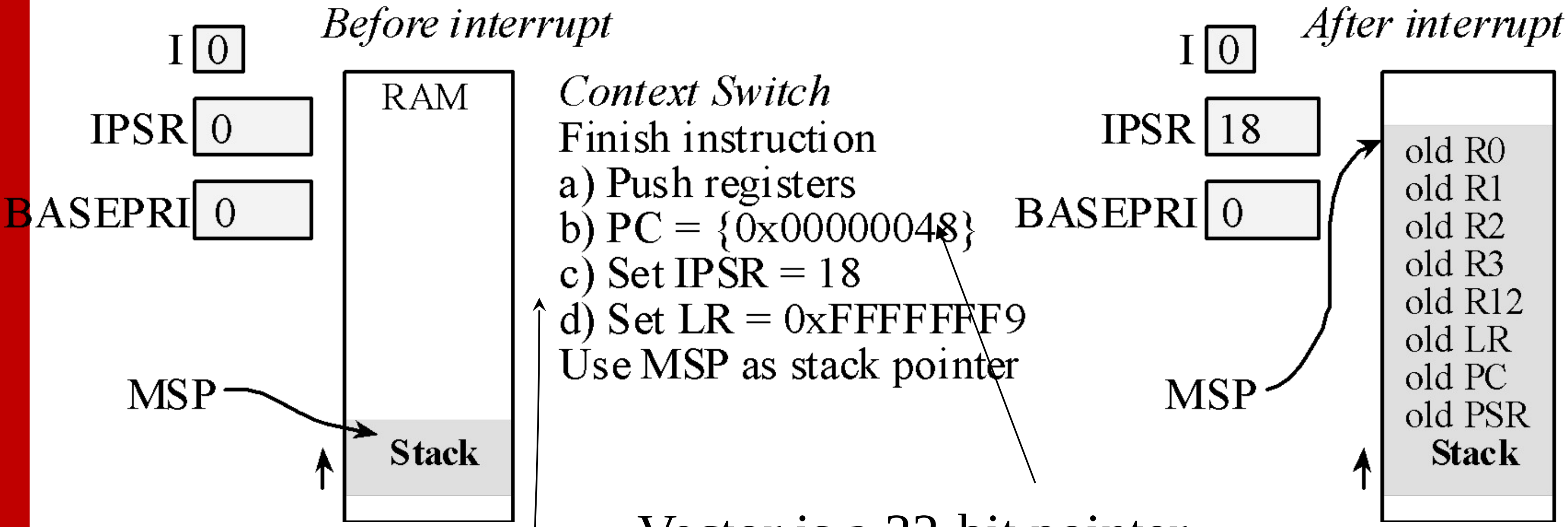
# Interrupt Conditions

- ❑ Four conditions must be true simultaneously for an interrupt to occur:
  1. Arm: control bit for each possible source is set
  2. Enable: interrupts globally enabled (I=0 in PRIMASK)
  3. Level: interrupt level must be less than BASEPRI
  4. Trigger: hardware action sets source-specific flag

```
void SysTick_Init(uint32_t period){
  NVIC_ST_RELOAD_R = period-1;  // reload value
  NVIC_ST_CURRENT_R = 0;          // any write will reload counter and clear count
  NVIC_SYS_PRI3_R =  (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;
  NVIC_ST_CTRL_R = 0x07;
}
int main(void){
  PLL_Init(Bus80MHz);          // bus clock at 80 MHz
  PortF_Init();
  Counts = 0;
  SysTick_Init(80000);         // initialize SysTick timer
  EnableInterrupts();
  while(1){                 // interrupts every 1ms, 500 Hz flash
        PF3 ^= 0x08;           // toggle PF3
  }
}
```

```
void SysTick_Handler(void){
  PF2 ^= 0x04;
  PF2 ^= 0x04;
  Counts = Counts + 1;
  PF2 ^= 0x04;
}
```

# Interrupt Context Switch



*Context Switch*

Finish instruction

a) Push registers

b) PC = {0x00000048}

c) Set IPSR = 18

d) Set LR = 0xFFFFFFF9

Use MSP as stack pointer

Vector is a 32-bit pointer

0x3C holds the address of SysTick ISR

0x48 holds the address of Port C ISR

Interrupt Number specifies which ISR is running

IPSR=18 means GPIO Port C

IPSR=15 means SysTick

# INTERRUPT VECTORS

77 total

| Vector address | Number | IRQ | ISR name in Startup.s | NVIC | Priority bits |
|---|---|---|---|---|---|
| 0x00000038 | 14 | -2 | PendSV_Handler | NVIC_SYS_PRI3_R | 23 – 21 |
| 0x0000003C | 15 | -1 | SysTick_Handler | NVIC_SYS_PRI3_R | 31 – 29 |
| 0x00000040 | 16 | 0 | GPIOPortA_Handler | NVIC_PRI0_R | 7 – 5 |
| 0x00000044 | 17 | 1 | GPIOPortB_Handler | NVIC_PRI0_R | 15 – 13 |
| 0x00000048 | 18 | 2 | GPIOPortC_Handler | NVIC_PRI0_R | 23 – 21 |
| 0x0000004C | 19 | 3 | GPIOPortD_Handler | NVIC_PRI0_R | 31 – 29 |
| 0x00000050 | 20 | 4 | GPIOPortE_Handler | NVIC_PRI1_R | 7 – 5 |
| 0x00000054 | 21 | 5 | UART0_Handler | NVIC_PRI1_R | 15 – 13 |
| 0x00000058 | 22 | 6 | UART1_Handler | NVIC_PRI1_R | 23 – 21 |
| 0x0000005C | 23 | 7 | SSI0_Handler | NVIC_PRI1_R | 31 – 29 |
| 0x00000060 | 24 | 8 | I2C0_Handler | NVIC_PRI2_R | 7 – 5 |
| 0x00000064 | 25 | 9 | PWMFault_Handler | NVIC_PRI2_R | 15 – 13 |
| 0x00000068 | 26 | 10 | PWM0_Handler | NVIC_PRI2_R | 23 – 21 |
| 0x0000006C | 27 | 11 | PWM1_Handler | NVIC_PRI2_R | 31 – 29 |
| 0x00000070 | 28 | 12 | PWM2_Handler | NVIC_PRI3_R | 7 – 5 |
| 0x00000074 | 29 | 13 | Quadrature0_Handler | NVIC_PRI3_R | 15 – 13 |
| 0x00000078 | 30 | 14 | ADC0_Handler | NVIC_PRI3_R | 23 – 21 |
| 0x0000007C | 31 | 15 | ADC1_Handler | NVIC_PRI3_R | 31 – 29 |
| 0x00000080 | 32 | 16 | ADC2_Handler | NVIC_PRI4_R | 7 – 5 |
| 0x00000084 | 33 | 17 | ADC3_Handler | NVIC_PRI4_R | 15 – 13 |
| 0x00000088 | 34 | 18 | WDT_Handler | NVIC_PRI4_R | 23 – 21 |
| 0x0000008C | 35 | 19 | Timer0A_Handler | NVIC_PRI4_R | 31 – 29 |
| 0x00000090 | 36 | 20 | Timer0B_Handler | NVIC_PRI5_R | 7 – 5 |
| 0x00000094 | 37 | 21 | Timer1A_Handler | NVIC_PRI5_R | 15 – 13 |
| 0x00000098 | 38 | 22 | Timer1B_Handler | NVIC_PRI5_R | 23 – 21 |
| 0x0000009C | 39 | 23 | Timer2A_Handler | NVIC_PRI5_R | 31 – 29 |
| 0x000000A0 | 40 | 24 | Timer2B_Handler | NVIC_PRI6_R | 7 – 5 |
| 0x000000A4 | 41 | 25 | Comp0_Handler | NVIC_PRI6_R | 15 – 13 |
| 0x000000A8 | 42 | 26 | Comp1_Handler | NVIC_PRI6_R | 23 – 21 |
| 0x000000AC | 43 | 27 | Comp2_Handler | NVIC_PRI6_R | 31 – 29 |
| 0x000000B0 | 44 | 28 | SysCtl_Handler | NVIC_PRI7_R | 7 – 5 |
| 0x000000B4 | 45 | 29 | FlashCtl_Handler | NVIC_PRI7_R | 15 – 13 |
| 0x000000B8 | 46 | 30 | GPIOPortF_Handler | NVIC_PRI7_R | 23 – 21 |
| 0x000000BC | 47 | 31 | GPIOPortG_Handler | NVIC_PRI7_R | 31 – 29 |
| 0x000000C0 | 48 | 32 | GPIOPortH_Handler | NVIC_PRI8_R | 7 – 5 |
| 0x000000C4 | 49 | 33 | UART2_Handler | NVIC_PRI8_R | 15 – 13 |
| 0x000000C8 | 50 | 34 | SSI1_Handler | NVIC_PRI8_R | 23 – 21 |
| 0x000000CC | 51 | 35 | Timer3A_Handler | NVIC_PRI8_R | 31 – 29 |
| 0x000000D0 | 52 | 36 | Timer3B_Handler | NVIC_PRI9_R | 7 – 5 |
| 0x000000D4 | 53 | 37 | I2C1_Handler | NVIC_PRI9_R | 15 – 13 |
| 0x000000D8 | 54 | 38 | Quadrature1_Handler | NVIC_PRI9_R | 23 – 21 |
| 0x000000DC | 55 | 39 | CAN0_Handler | NVIC_PRI9_R | 31 – 29 |
| 0x000000E0 | 56 | 40 | CAN1_Handler | NVIC_PRI10_R | 7 – 5 |
| 0x000000E4 | 57 | 41 | CAN2_Handler | NVIC_PRI10_R | 15 – 13 |
| 0x000000E8 | 58 | 42 | Ethernet_Handler | NVIC_PRI10_R | 23 – 21 |
| 0x000000EC | 59 | 43 | Hibernate_Handler | NVIC_PRI10_R | 31 – 29 |
| 0x000000F0 | 60 | 44 | USB0_Handler | NVIC_PRI11_R | 7 – 5 |
| 0x000000F4 | 61 | 45 | PWM3_Handler | NVIC_PRI11_R | 15 – 13 |
| 0x000000F8 | 62 | 46 | uDMA_Handler | NVIC_PRI11_R | 23 – 21 |
| 0x000000FC | 63 | 47 | uDMA_Error | NVIC_PRI11_R | 31 – 29 |

# Priority Bits

- **High order three bits of each byte define priority**

| Address | 31 – 29 | 23 – 21 | 15 – 13 | 7 – 5 | Name |
|---|---|---|---|---|---|
| 0xE000E400 | GPIO Port D | GPIO Port C | GPIO Port B | GPIO Port A | NVIC_PRI0_R |
| 0xE000E404 | SSI0, Rx Tx | **UART1, Rx Tx** | UART0, Rx Tx | GPIO Port E | NVIC_PRI1_R |
| 0xE000E408 | PWM Gen 1 | PWM Gen 0 | PWM Fault | I2C0 | NVIC_PRI2_R |
| 0xE000E40C | ADC Seq 1 | ADC Seq 0 | Quad Encoder | PWM Gen 2 | NVIC_PRI3_R |
| 0xE000E410 | Timer 0A | Watchdog | ADC Seq 3 | ADC Seq 2 | NVIC_PRI4_R |
| 0xE000E414 | Timer 2A | Timer 1B | Timer 1A | Timer 0B | NVIC_PRI5_R |
| 0xE000E418 | Comp 2 | Comp 1 | Comp 0 | Timer 2B | NVIC_PRI6_R |
| 0xE000E41C | GPIO Port G | GPIO Port F | Flash Control | System Control | NVIC_PRI7_R |
| 0xE000E420 | Timer 3A | SSI1, Rx Tx | UART2, Rx Tx | GPIO Port H | NVIC_PRI8_R |
| 0xE000E424 | CAN0 | Quad Encoder 1 | I2C1 | Timer 3B | NVIC_PRI9_R |
| 0xE000E428 | Hibernate | Ethernet | CAN2 | CAN1 | NVIC_PRI10_R |
| 0xE000E42C | uDMA Error | uDMA Soft Tfr | PWM Gen 3 | USB0 | NVIC_PRI11_R |
| 0xE000ED20 | **SysTick** | PendSV | -- | Debug | NVIC_SYS_PRI3_R |

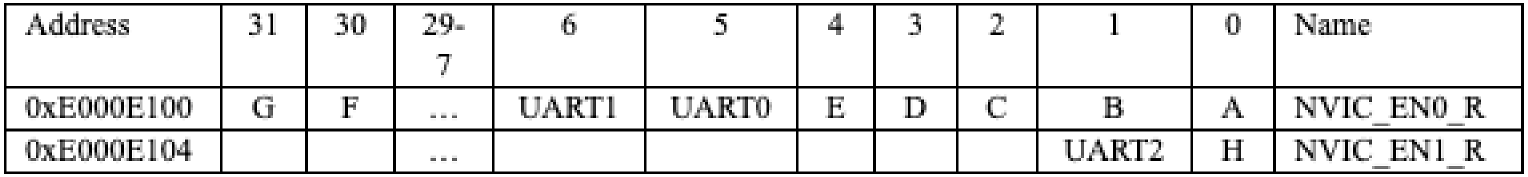
# NVIC Interrupt Enable Registers

❑Arm bit in the device

❑Enable bit in the NVIC –

- ❖A single enable bit for each device
- ❖**NVIC_EN0_R** for IRQ numbers 0 to 31
- ❖**NVIC_EN1_R** for IRQ numbers 32 to 47
- ❖SysTick does not need this enable step
- ❖Write 1 to enable, writing 0 has no effect

```
NVIC_EN0_R = 0x00000002; // IRQ 1
```



| Address | 31 | 30 | 29-7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Name |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0xE000E100 | G | F | … | UART1 | UART0 | E | D | C | B | A | NVIC_EN0_R |
| 0xE000E104 | | | … | | | | | | UART2 | H | NVIC_EN1_R |

# Interrupt Rituals

- ❑Things you must do in every ritual
  - ❖Initialize data structures (counters, pointers)

    Count = 0; // global variable

  - ❖Arm (specify a flag may interrupt)

    NVIC_ST_CTRL_R = 0x07; // bit1 is arm bit

  - ❖Configure NVIC
    - o Enable interrupt (NVIC_EN0_R)
    - o Set priority (e.g., NVIC_PRI1_R)

    SysTick skips this step

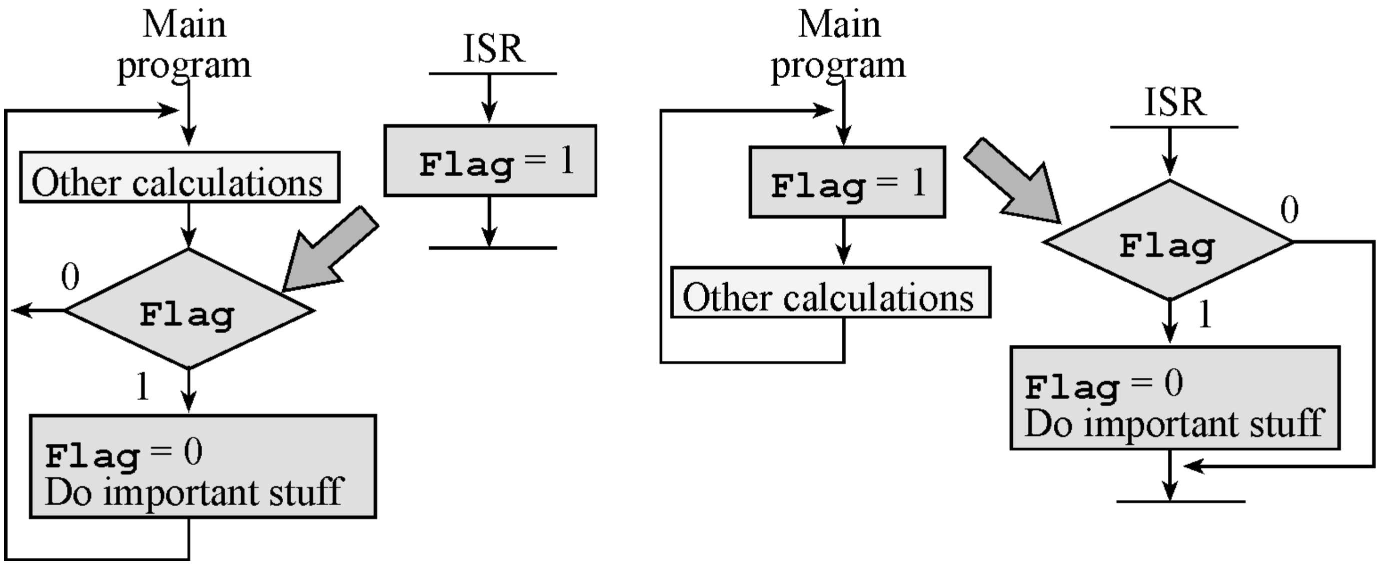
    NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;

  - ❖Enable Interrupts
    - o Assembly code **`CPSIE I`**
    - o C code **`EnableInterrupts();`**

# Interrupt Service Routine (ISR)

- ❑Must do in every interrupt service routine
  - ❖Acknowledge
    - o clear flag that requested the interrupt
    - o SysTick is exception; automatic acknowledge, Processing the SysTick ISR clears count flag
  - ❖Push/pop R4-R11 if used (AAPCS)
  - ❖Communicate via shared global variables

# Synchronization



Use global variable to communicate

- Semaphore
  - One thread sets the flag
  - The other thread waits for, and clears
- Mailbox
- FIFO queue

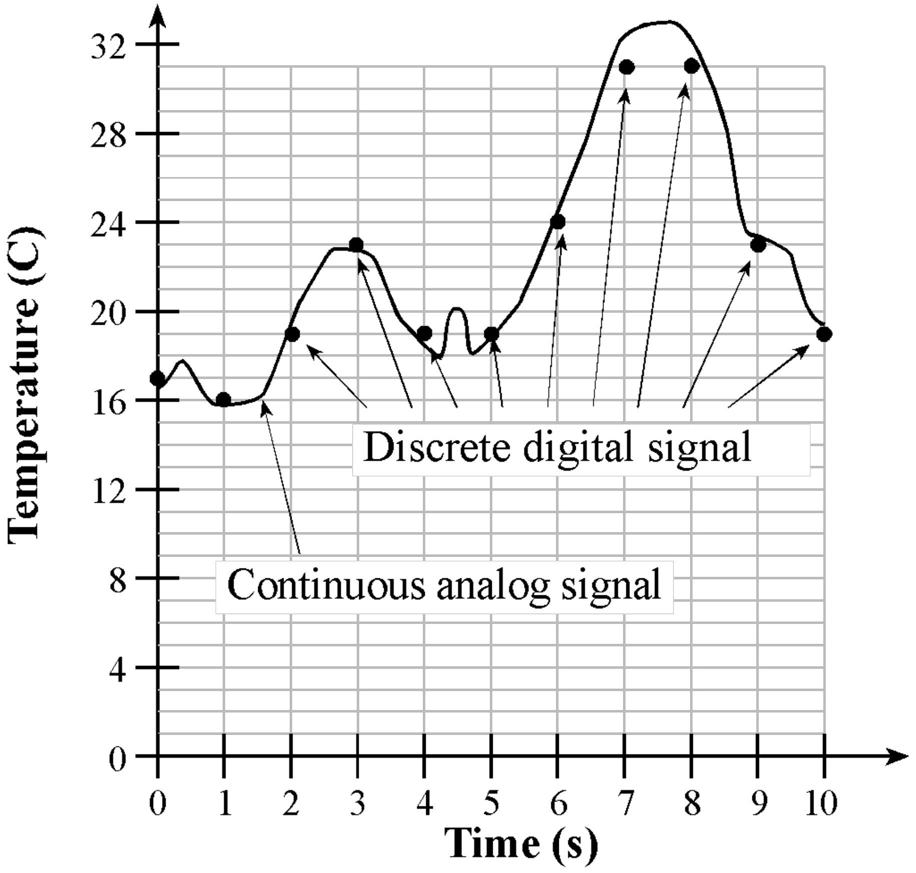
# Periodic Interrupts

- Data acquisition samples ADC
- Signal generation output to DAC
  - Audio player (we use the Systick interrupt to write samples out periodically)
  - Communications
- Digital controller
  - FSM
  - Linear control system

Demo **`PeriodicSystickInts`** starter C code

# Digital Representation of Analog Signals

**Digitization**: Amplitude and time quantization



# Conversion from Digital to Analog

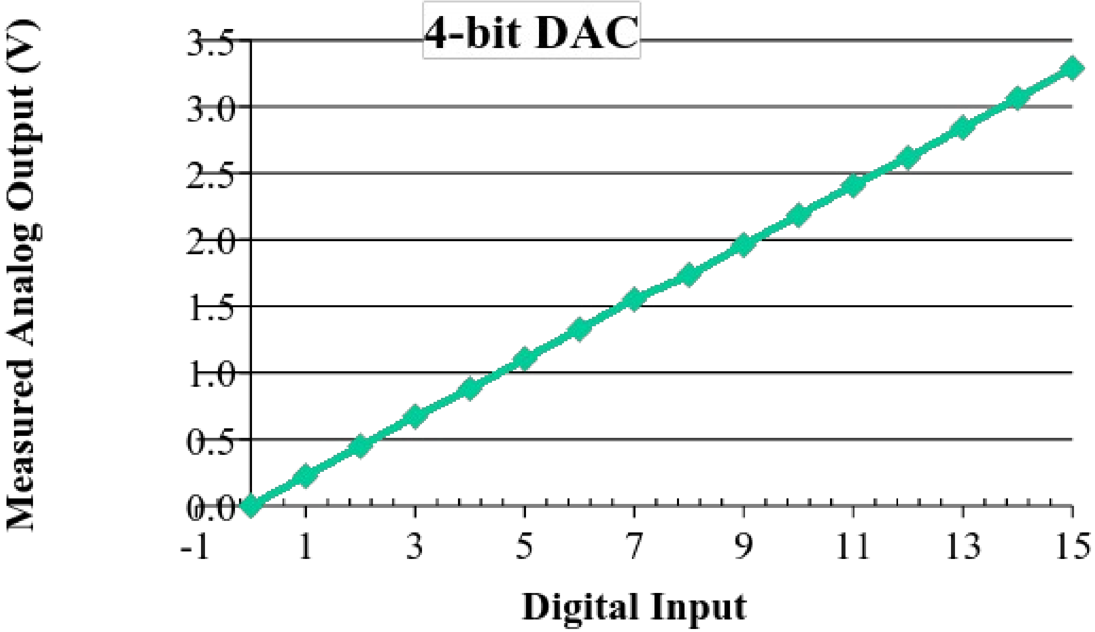
❑Range
  - ❖0 to 3.3V

❑Resolution
  - ❖3.3V/15=0.22V

❑Precision
  - ❖4 bits
  - ❖16 alternatives

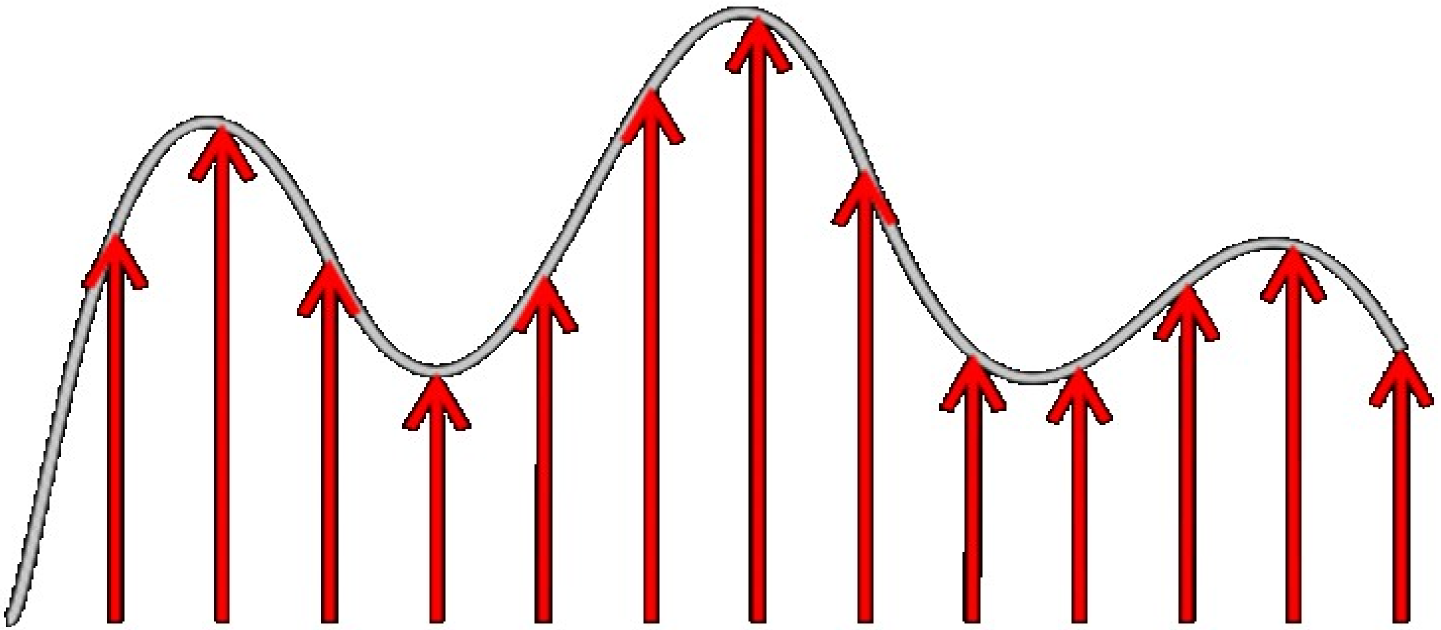
❑Speed

❑Monotonic



# Digital ↔ Analog Conversion

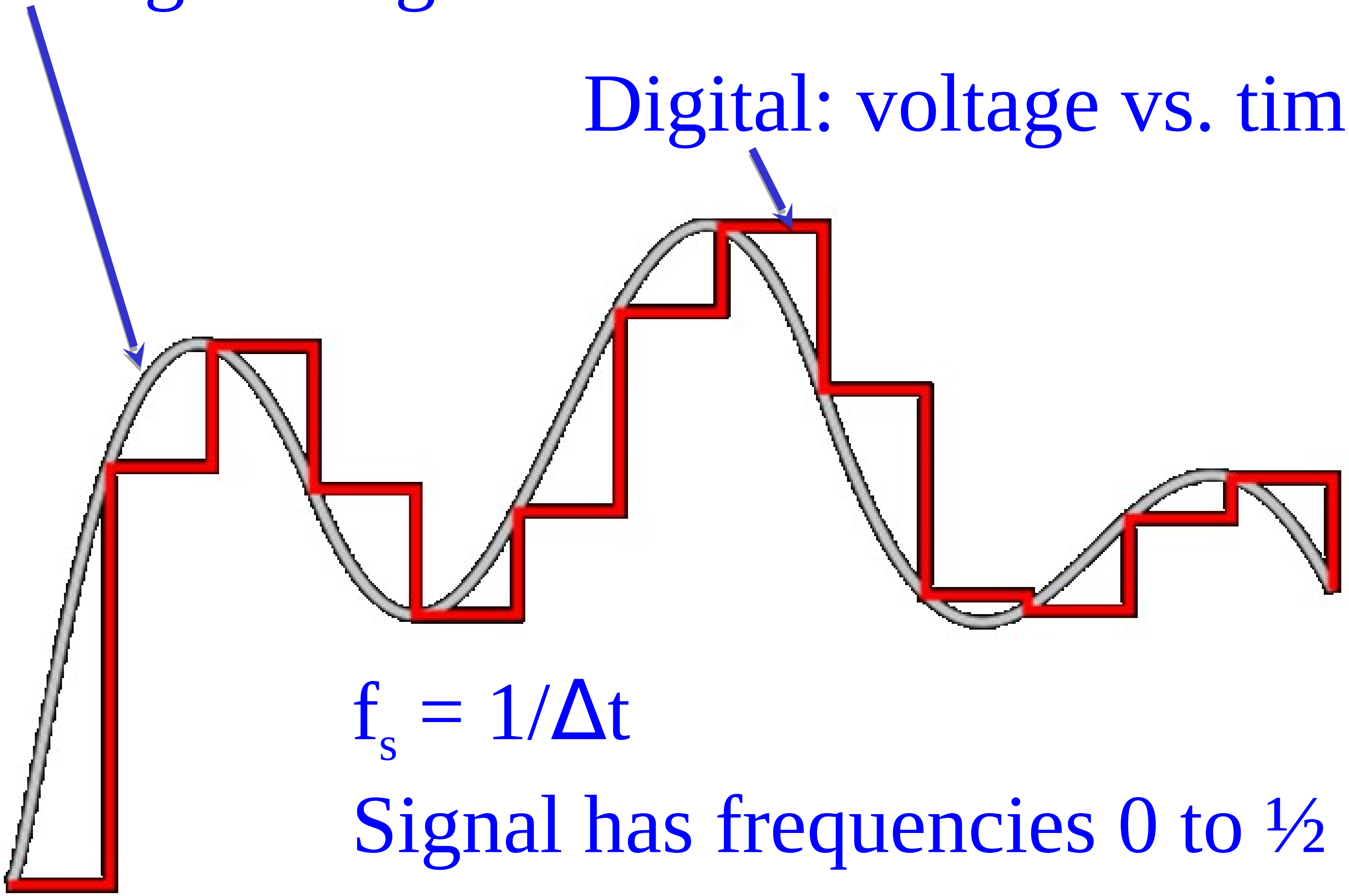
Sampled at a fixed time, Δt



$f_s$ = 1/Δt

Signal has frequencies 0 to ½ $f_s$

# Digital ↔ Analog Conversion



Analog: voltage vs. time

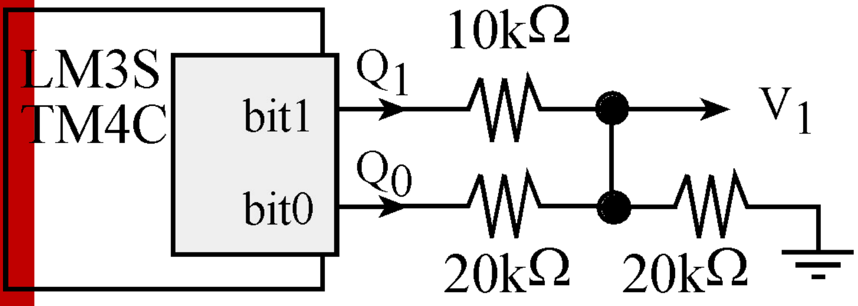
Digital: voltage vs. time

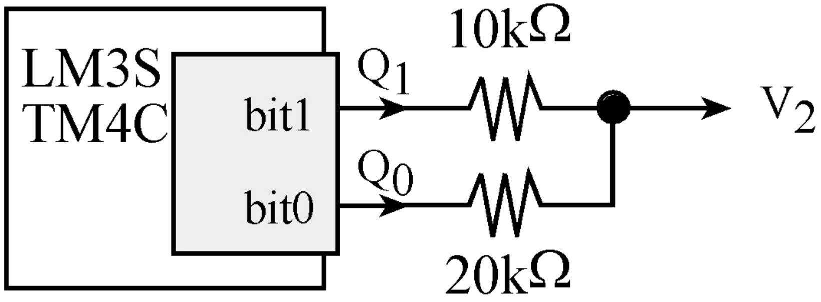
$f_s = 1/\Delta t$

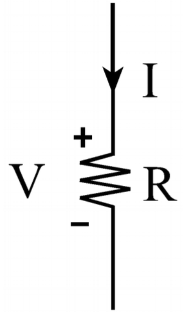
Signal has frequencies 0 to ½ $f_s$

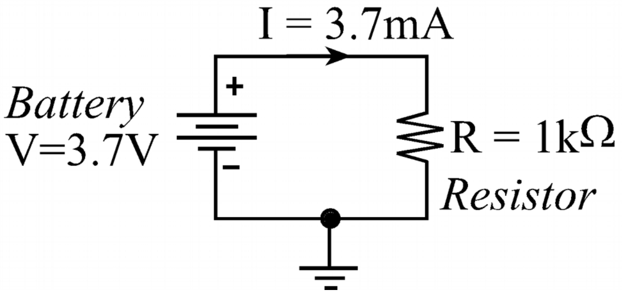
# Digital-to-Analog Converter (DAC)

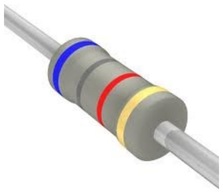
- Binary Weighted DAC
  - One resistor for each bit of output
  - Resistor values in powers of 2





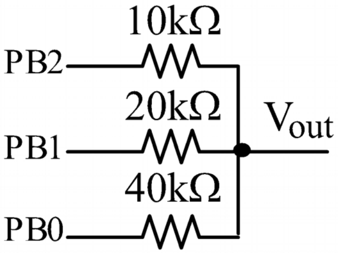




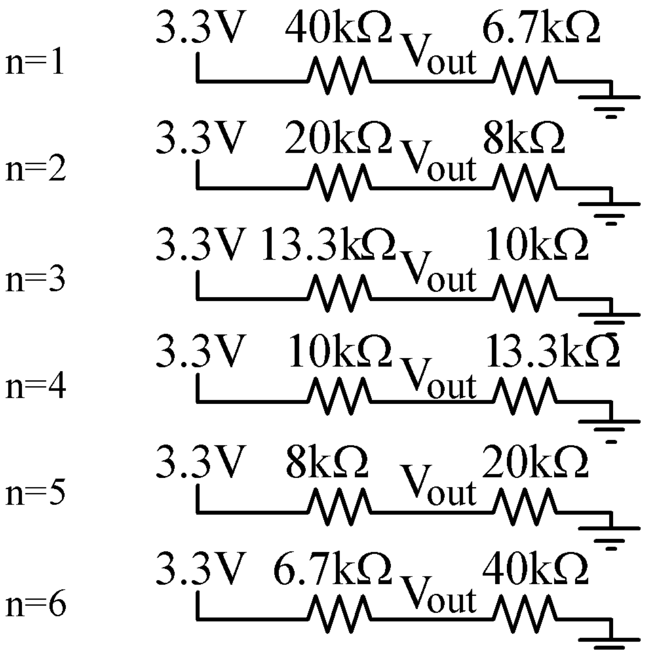


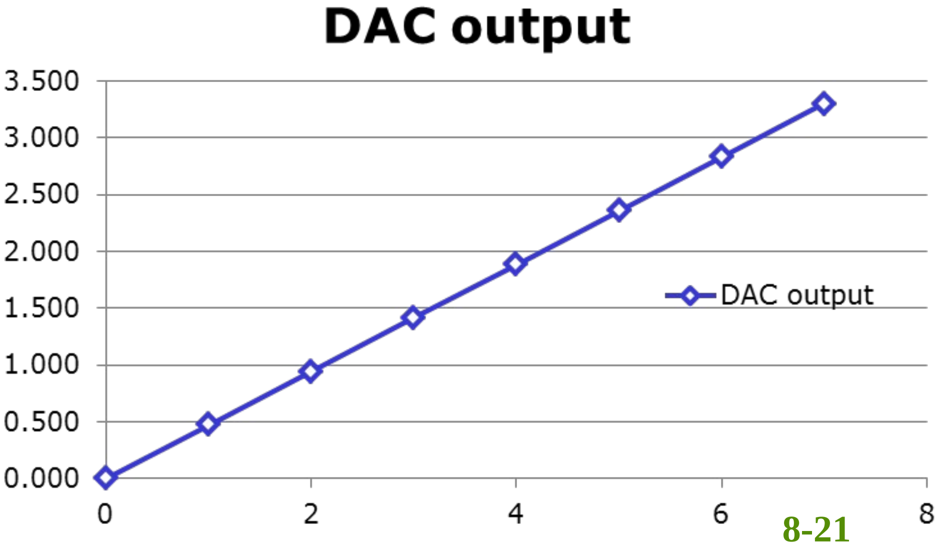
|  |  |  |
|---|---|---|
| R2 | 10 | kΩ |
| R1 | 20 | kΩ |
| R0 | 40 | kΩ |

# 3 bit DAC



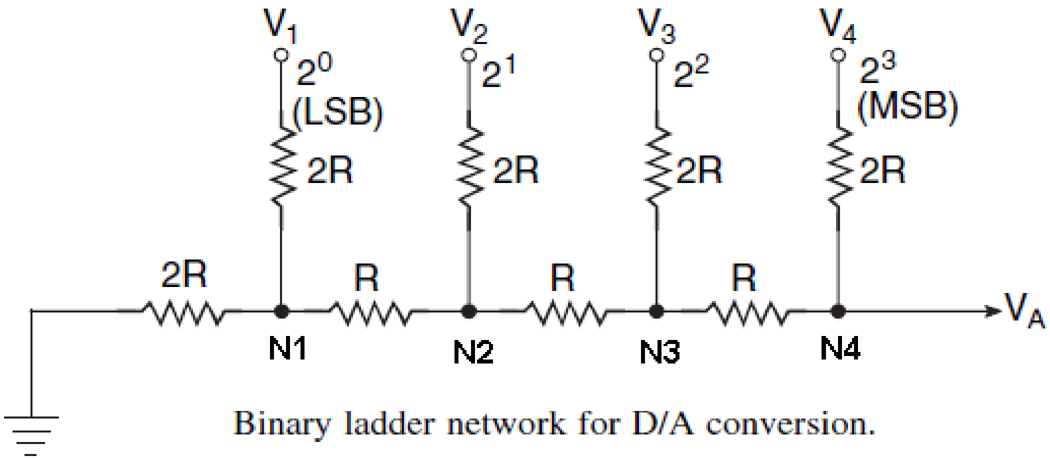
| n | PB2 | PB1 | PB0 |  | kohm | equation | Vout (V) |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 |  |  |  | 0.000 |
| 1 | 0 | 0 | 3.3 | R2\|\|R1 | 6.67 | 3.3*(R1\|\|R2)/(R0+R1\|\|R2) | 0.471 |
| 2 | 0 | 3.3 | 0 | R2\|\|R0 | 8.00 | 3.3*(R2\|\|R0)/(R1+R2\|\|R0) | 0.943 |
| 3 | 0 | 3.3 | 3.3 | R1\|\|R0 | 13.33 | 3.3*R2/(R2+R1\|\|R0) | 1.414 |
| 4 | 3.3 | 0 | 0 | R1\|\|R0 | 13.33 | 3.3*(R1\|\|R0)/(R2+R1\|\|R0) | 1.886 |
| 5 | 3.3 | 0 | 3.3 | R2\|\|R0 | 8.00 | 3.3*R1/(R1+R2\|\|R0) | 2.357 |
| 6 | 3.3 | 3.3 | 0 | R2\|\|R1 | 6.67 | 3.3*R0/(R0+R2\|\|R1) | 2.829 |
| 7 | 3.3 | 3.3 | 3.3 |  |  |  | 3.300 |





# Other Types of DACs

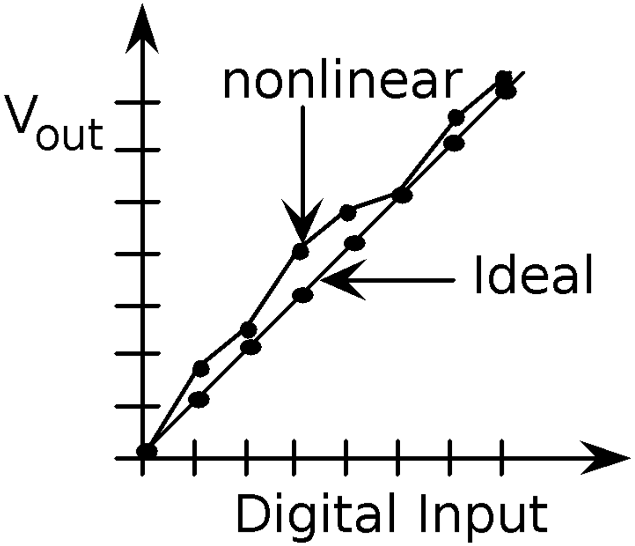
- ❑ R-2R Ladder DAC
  - ❖ Binary weighted cascading ladder
  - ❖ Improved precision owing to ability to select resistors of equal value

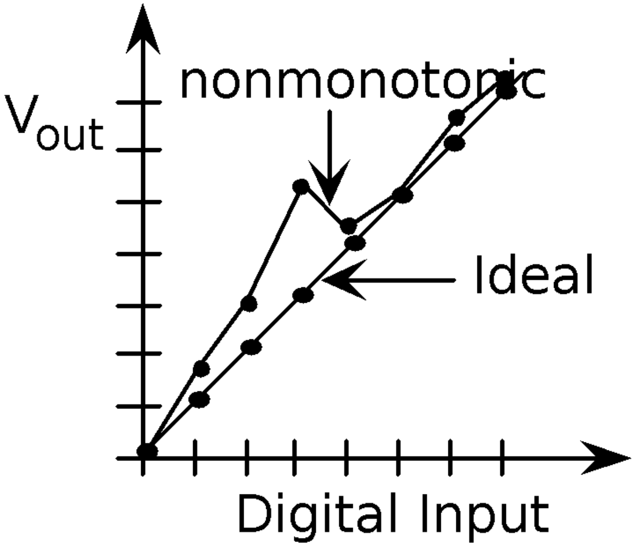


Binary ladder network for D/A conversion.

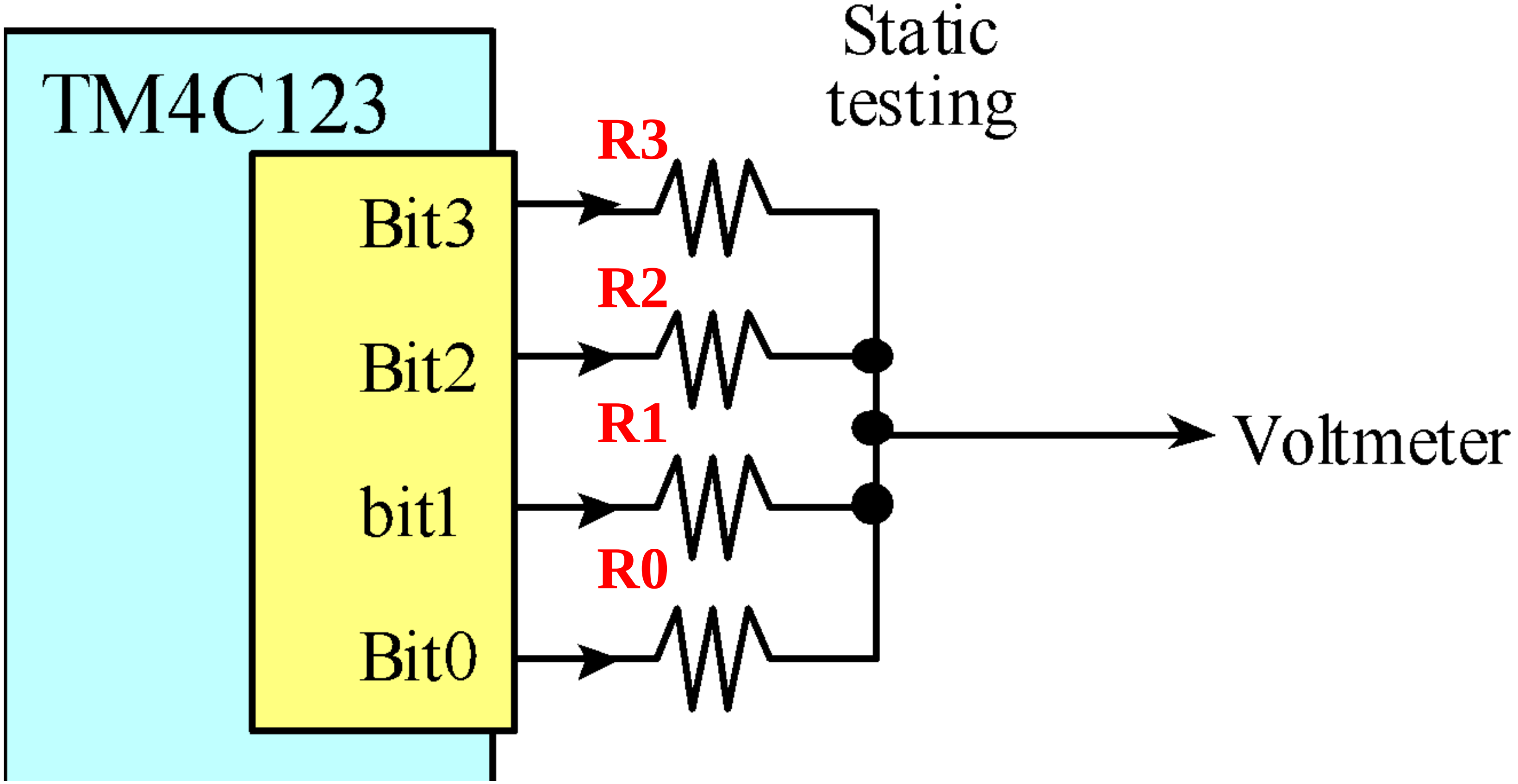
# DAC Performance

- ❑Resolution, range, precision
- ❑Maximum sampling frequency
- ❑Monotonicity
  - ❖Input increase causes output increase (always)

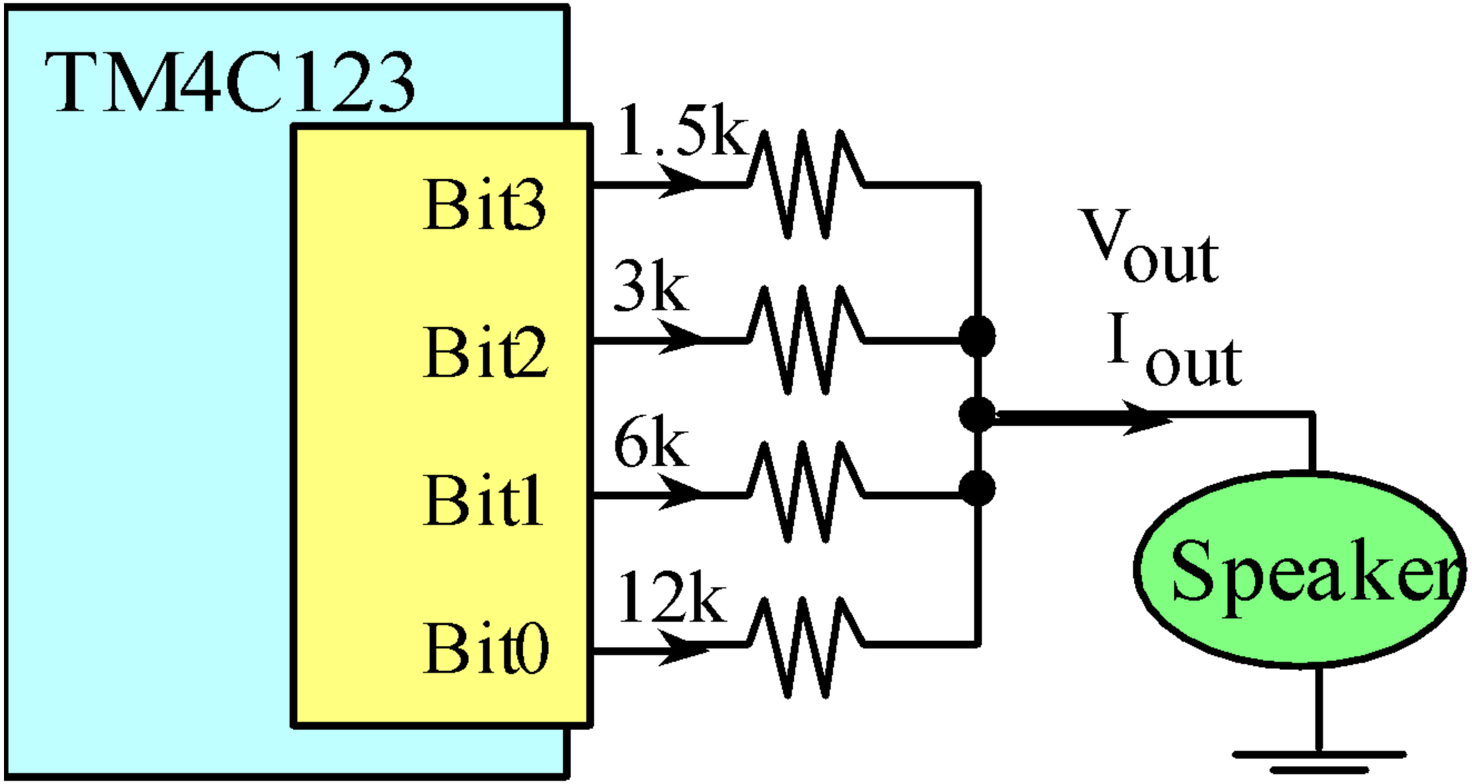




# Resistor Network for 4-bit DAC

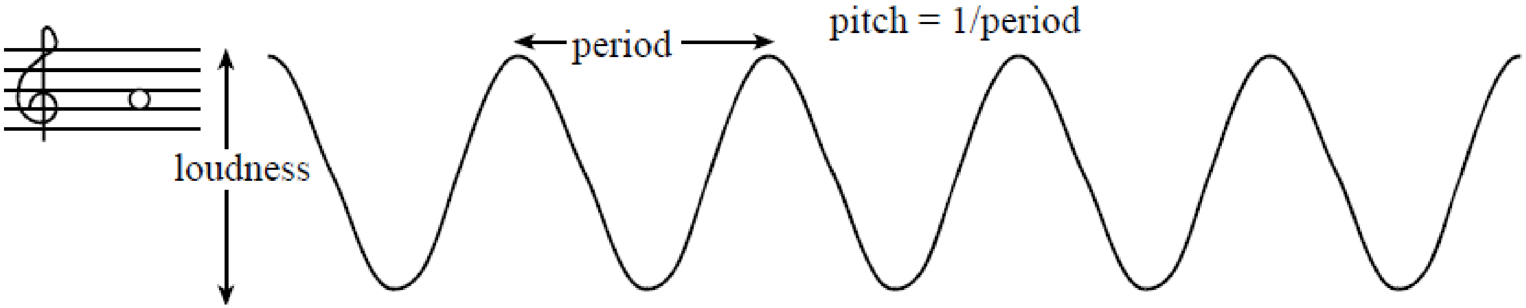


# Dynamic testing



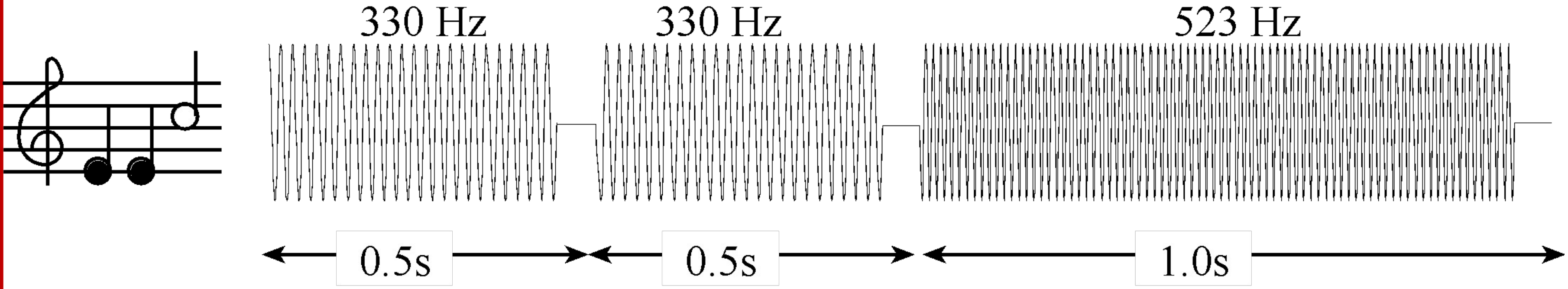
# Sound

- ❑Loudness and pitch
  - ❖ Controlled by amplitude and frequency



- ❖Humans can hear from about 25 to 20,000 Hz.
- ❖Middle A is 440 Hz
- ❖Other notes on a keyboard are determined
  - o 440 * $2^{N/12}$, where *N* is no. of notes from middle A
- ❖Middle C is 261.6 Hz.
- ❖Music contains multiple harmonics

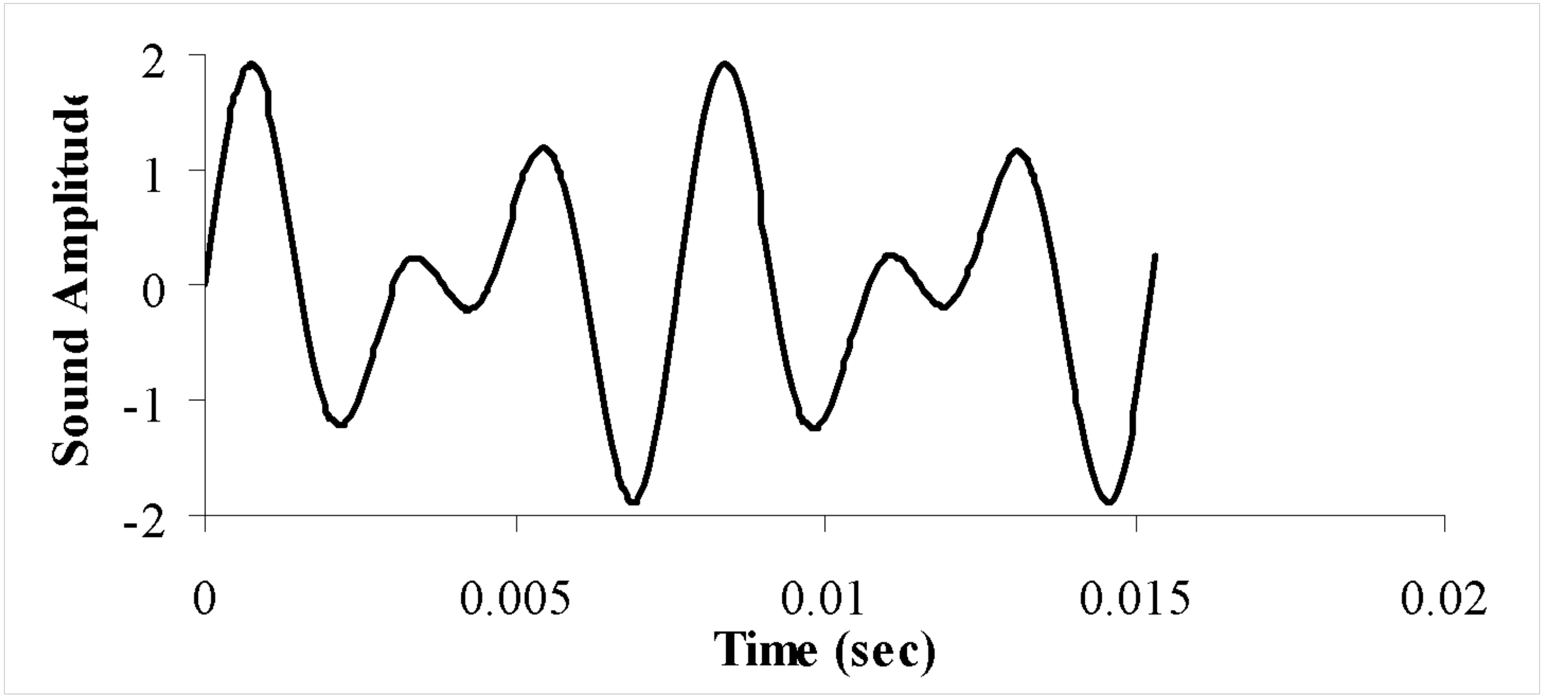
# Tempo



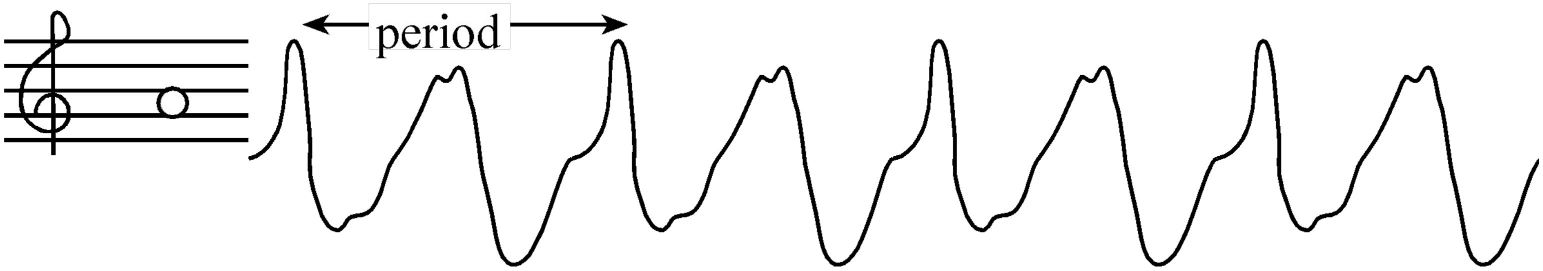
Tempo defines note duration
Quarter note = 1 beat
120 beats/min => ½ s duration

# Chord

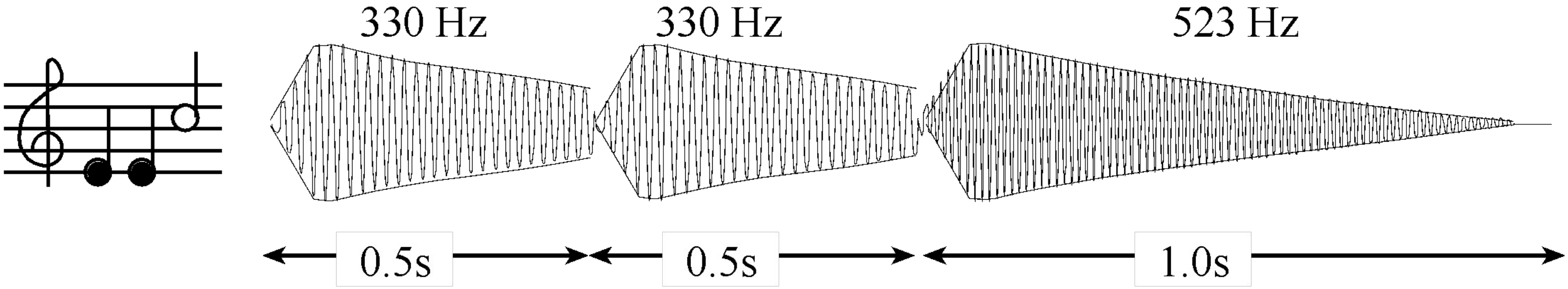
- ❑ Two notes at the same time
  - ❖ Superimposed waveforms
  - ❖ 262 Hz (low C) and a 392 Hz (G)



# Instrument Characteristics



Waveform *shape* of a trumpet sound



Plucked string signal with *envelope*

# Synthesizing Digital Music

❑ Nyquist’s Sampling Theorem

- ❖ We can reproduce any bandlimited signal from its samples if we sample correctly and at a frequency, $f_s$, that is at least twice the highest frequency component of the signal, $f_{max}$.

❑ Where do we get the samples?

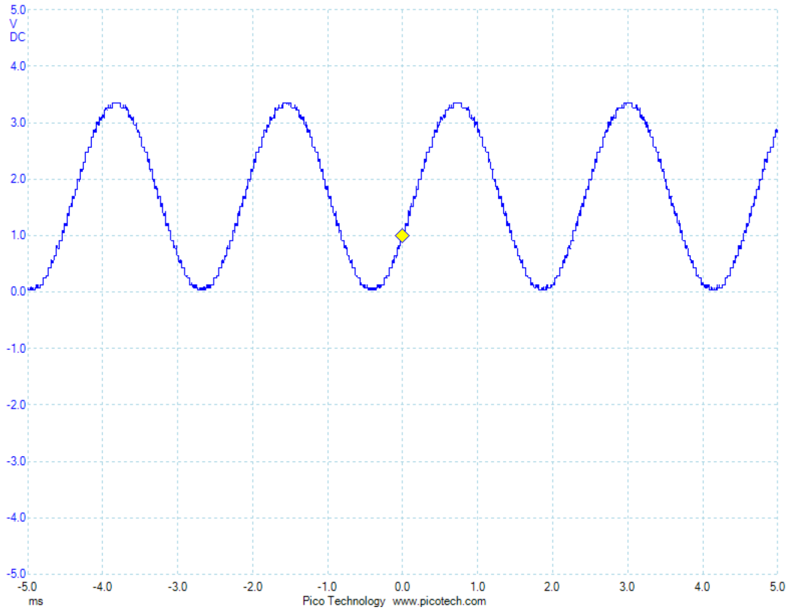
- ❖ We could sample a series of musical tones
- ❖ *We can **compute** the samples*

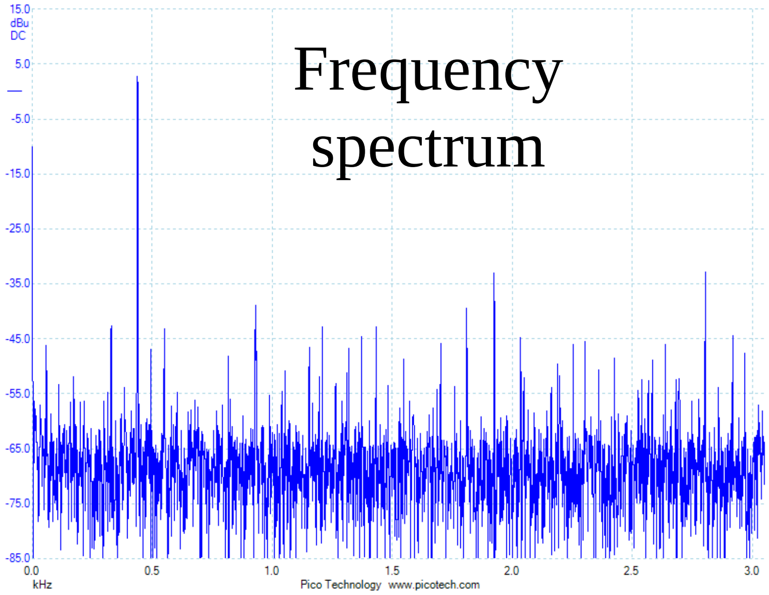
# Synthesizing Digital Music (cont.)

- What is a musical tone?
  - A sinusoid of a particular frequency
  - Notes vary by twelfth root of 2 ~ 1.059
- What would the samples be?
  - Fixed point numbers
- How do we generate a sinusoid?
  - Output appropriate digital values via a resistor network that effectively produces an *pseudo*-analog signal
- What about frequency?
  - Employ a programmable timer to tell us when to output the next value

# Synthesizing Digital Music (cont.)

- ❑440 Hz sine wave generated by 6-bit DAC




Frequency spectrum

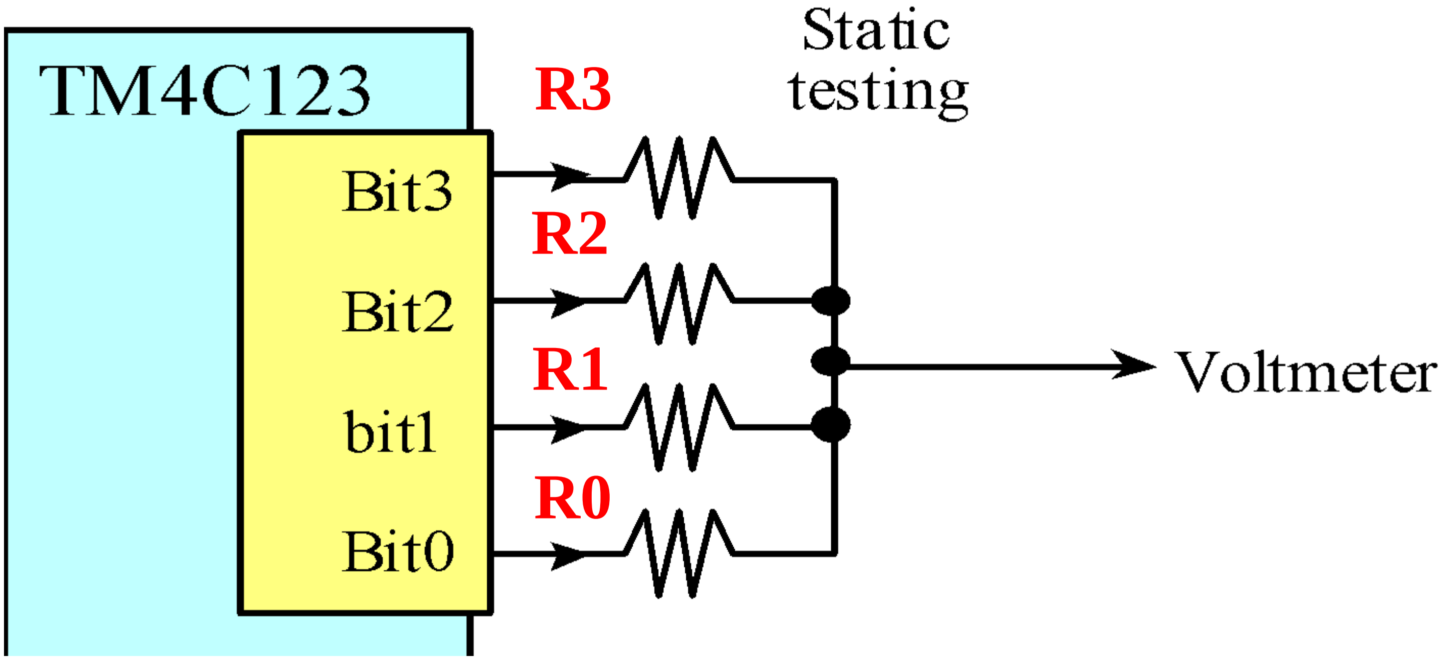
# Music Generation – Lab 6

- Objectives
  - Employ TM4C to generate *appropriately scaled* digital outputs at a specified *frequency*
    - Three frequencies are required
    - Frequencies are to be determined by switch settings
  - Four digital outputs are inputs to a resistor network that serves as a *digital-to-analog converter* (DAC)
    - Four output bits => 16 levels
    - Six output bits => 64 levels (optional)

# Music Generation (cont.)

❑DAC hardware

- ❖Employ least significant four bits of a GPIO port
- ❖Arrange resistor network in 1, 2, 4, 8 sequence
  - o Each port bit can assume digital levels of 0 and 3.3 V
  - o Ports are *current limited* – max 8 mA



# Music Generation (cont.)

- DAC software
  - Interactions via *device drivers*
  - Two device driver functions required

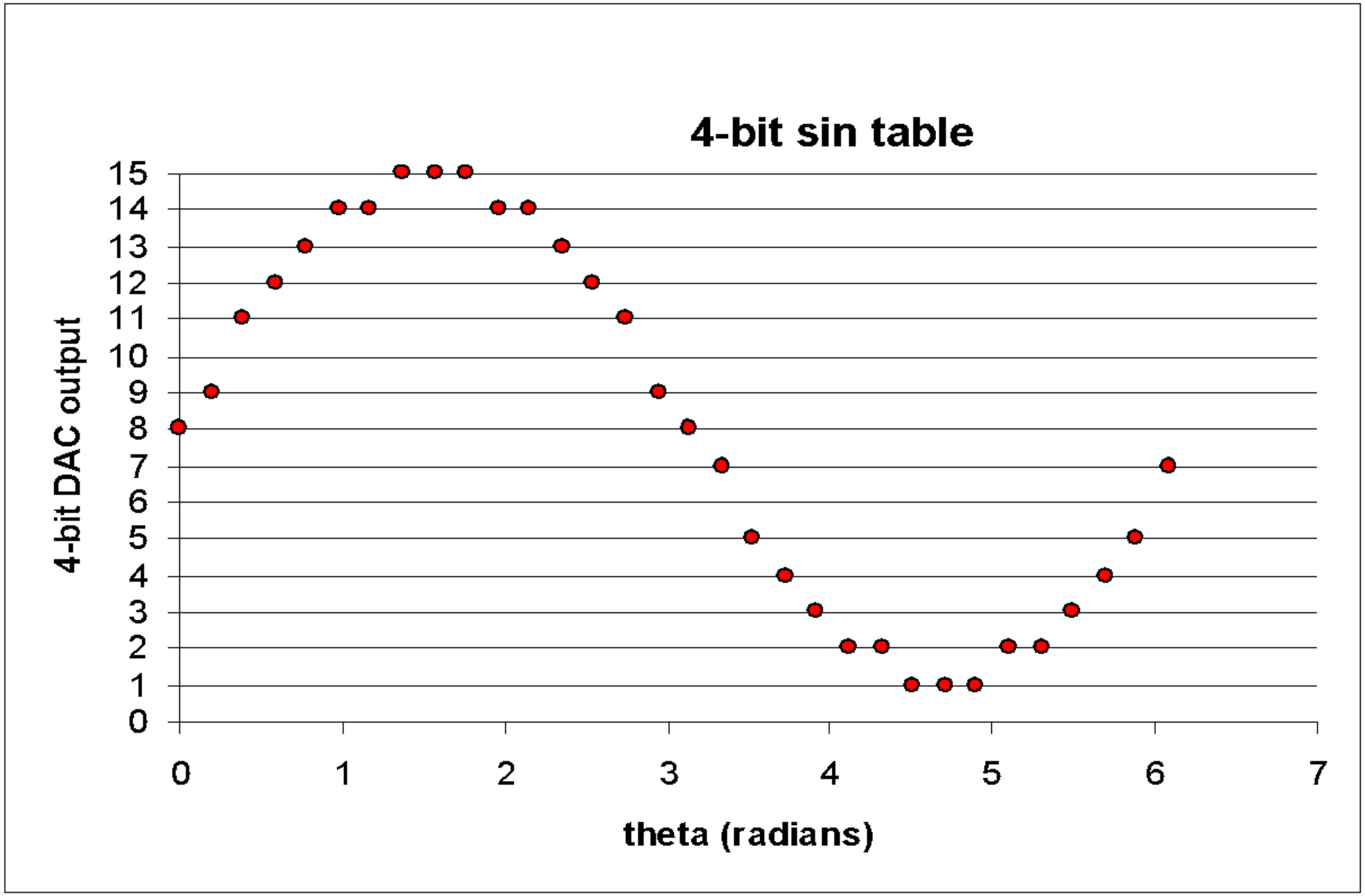
```
void DAC_Init(void);                 // initializes the device
void DAC_Out(uint8_t data);          // transfers data to device
```

(Device driver provides the *functions* associated with the device but hides the detailed actions necessary to implement the functions.)

# Music Generation (cont.)

- Interpretation of data
  - *Note* has three parameters
    - Amplitude (loudness)
    - Frequency (pitch)
    - Duration
  - Amplitude is a digitally approximated sinusoid
    - Sinusoid varies between 0 and 3.3 volts
  - Frequency is selected by switches
    - Four states – stop, note_1, note_2, and note_3
  - Duration is period switch(es) activated

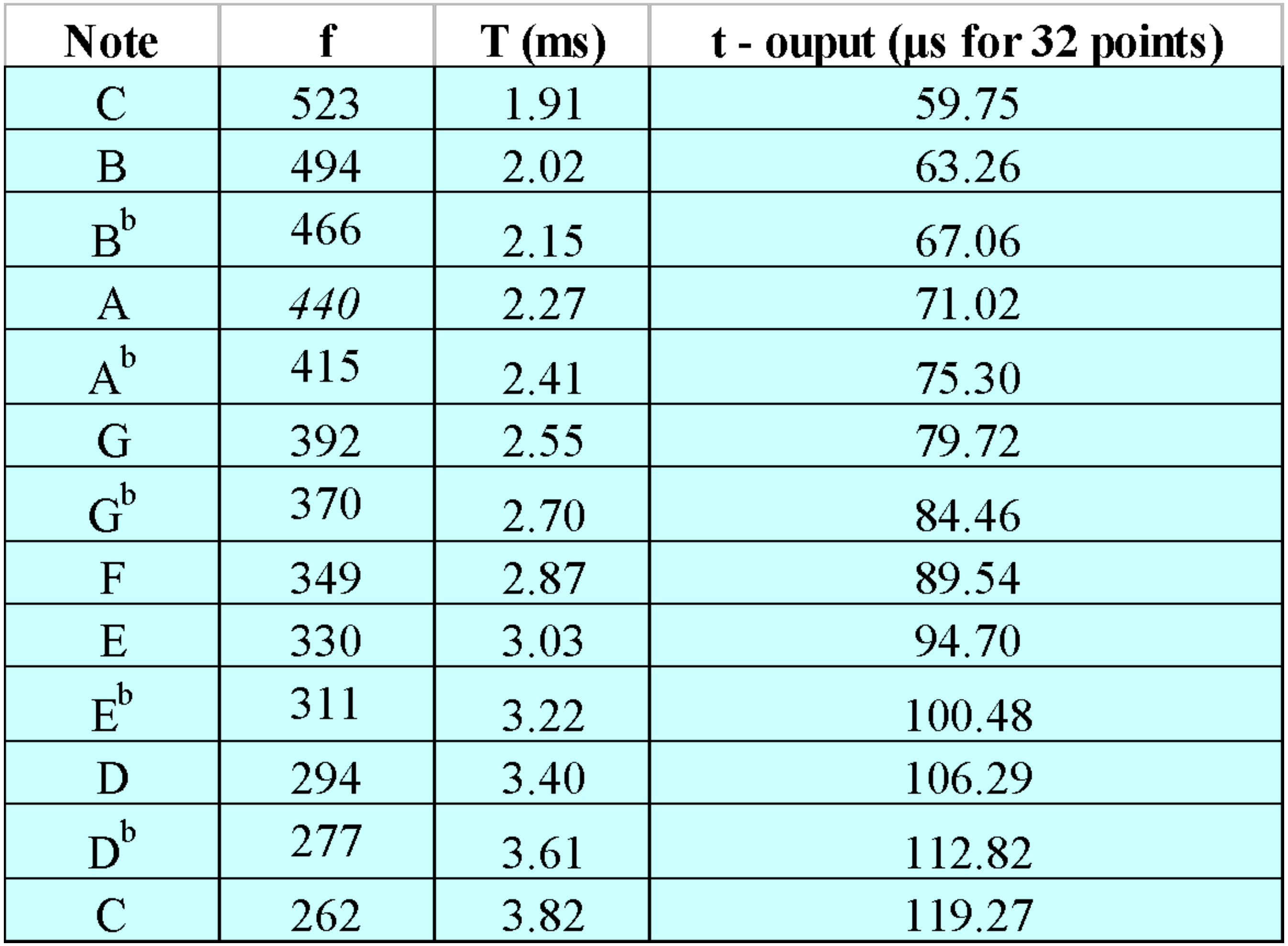
# 4-bit Sinusoid Table



```
SinTab   8,9,11,12,13,14,14,15,15,15,14
         14,13,12,11,9,8,7,5,4,3,2
         2,1,1,1,2,2,3,4,5,7
```

32 value sinusoid

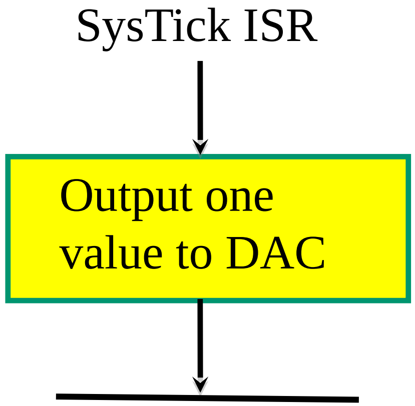
# Musical Notes



| Note | f | T (ms) | t - ouput (µs for 32 points) |
|---|---|---|---|
| C | 523 | 1.91 | 59.75 |
| B | 494 | 2.02 | 63.26 |
| $B^b$ | 466 | 2.15 | 67.06 |
| A | *440* | 2.27 | 71.02 |
| $A^b$ | 415 | 2.41 | 75.30 |
| G | 392 | 2.55 | 79.72 |
| $G^b$ | 370 | 2.70 | 84.46 |
| F | 349 | 2.87 | 89.54 |
| E | 330 | 3.03 | 94.70 |
| $E^b$ | 311 | 3.22 | 100.48 |
| D | 294 | 3.40 | 106.29 |
| $D^b$ | 277 | 3.61 | 112.82 |
| C | 262 | 3.82 | 119.27 |

# Tone Generation

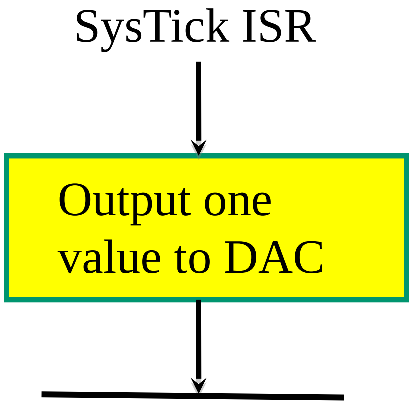
```
uint32_t I;
// 4-bit 32-element sine wave
const uint8_t wave[32]= {
  8,9,11,12,13,14,14,15,15,15,14
  14,13,12,11,9,8,7,5,4,3,2
  2,1,1,1,2,2,3,4,5,7};
```



- For a 440Hz tone
  - Assume a bus clock frequency of 50 MHz
    - SysTick count every 20ns
  - Each cycle of the 440 Hz sinusoid requires:
    - ($50*10^6$ counts/s)/440 Hz = 113636.36 SysTick counts
  - Each cycle consists of 32 values each of duration:
    - 113636.36 interrupt counts/32 values = 3551 SysTick counts/value
    - DAC values change every 71.02 us

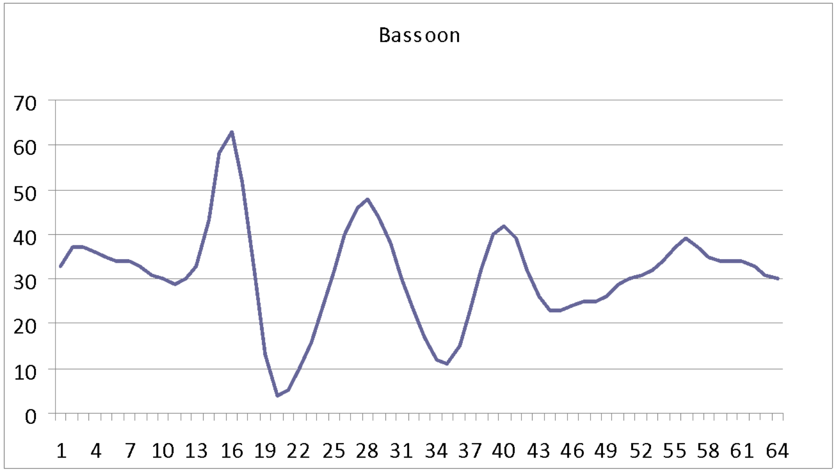
# Lab 6 ISR

- ❑Each Systick interrupt
  - ❖Output one value from the array to DAC
  - ❖Increment index to array (wrap back to zero)
- ❑In main program
  - ❖If a switch is pressed set SysTick period (arm)
  - ❖If no switches are pressed then disarm

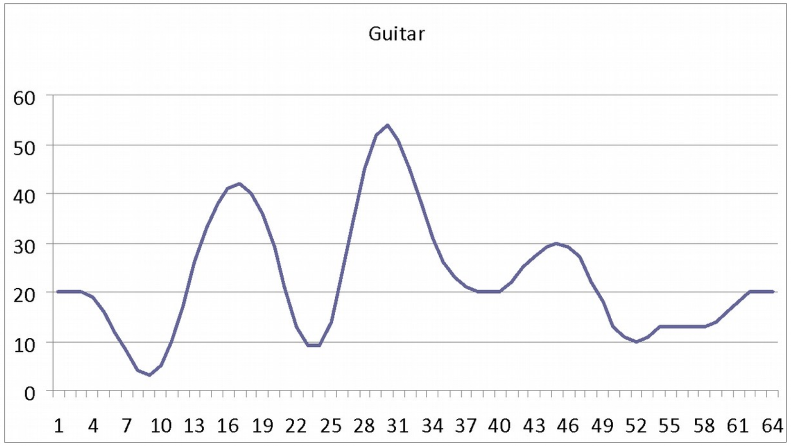


# Other Instruments

```
// 6-bit 64-element bassoon wave
const uint8_t Bassoon[64] = {
  33,37,37,36,35,34,34,33,31,30,29,
  30,33,43,58,63,52,31,13,4,5,10,16,
  23,32,40,46,48,44,38,30,23,17,12,11,
  15,23,32,40,42,39,32,26,23,23,24,25,
  25,26,29,30,31,32,34,37,39,37,35,34,
  34,34,33,31,30};
```



```
// 6-bit 64-element guitar wave
const uint8_t Guitar[64] = {
  20,20,20,19,16,12,8,4,3,5,10,17,
  26,33,38,41,42,40,36,29,21,13,9,
  9,14,23,34,45,52,54,51,45,38,31,
  26,23,21,20,20,20,22,25,27,29,
  30,29,27,22,18,13,11,10,11,13,13,
  13,13,13,14,16,18,20,20,20};
```


Guitar