

Admin details:

Steven Leeb
sbleeb@mit.edu

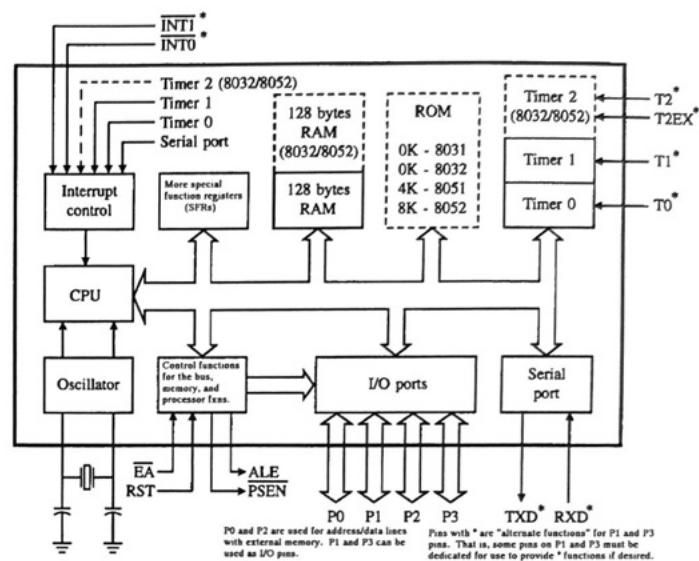
Office Hours: Anytime!

Please see handouts for course details.

Introduction: Why and where embedded control and microcontrollers?
Why the 8051 family?

Hardware: Books, handouts, lab kit in the mail.
R31JP microcontroller board.
PC interface, cross-assembler.
Lab familiarization.

Inside the Microcontroller:



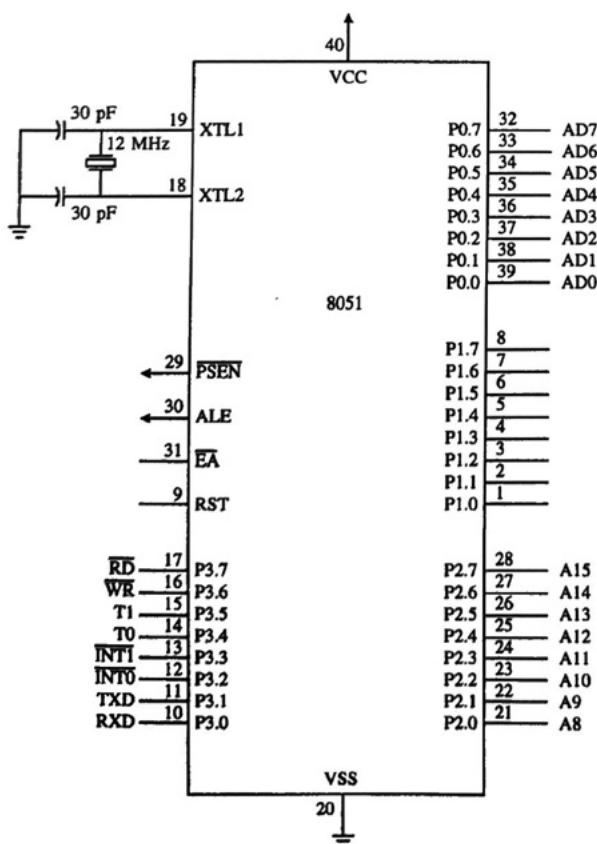
8051 Microcontroller – What is it?

(Figures in this lecture adopted from Yeralan, et.al., "The 8051 Microcontroller" by MacKenzie, and the Intel MCS-51 Microcontroller Family User's Manual.)

1

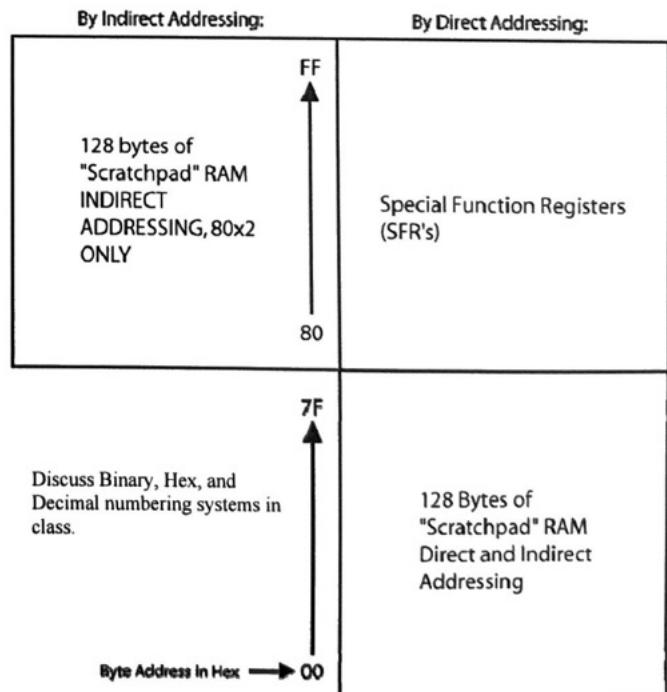
2

In "CHIP" form:



3

Internal Memory Map:



4

Base "Scratchpad" RAM (first 128 bytes): (in lecture, discuss stack)

Byte address	Bit address
7F	General purpose RAM
30	
2F	7F 7E 7D 7C 7B 7A 79 78
2E	77 76 75 74 73 72 71 70
2D	6F 6E 6D 6C 6B 6A 69 68
2C	67 66 65 64 63 62 61 60
2B	5F 5E 5D 5C 5B 5A 59 58
2A	57 56 55 54 53 52 51 50
29	4F 4E 4D 4C 4B 4A 49 48
28	47 46 45 44 43 42 41 40
27	3F 3E 3D 3C 3B 3A 39 38
26	37 36 35 34 33 32 31 30
25	2F 2E 2D 2C 2B 2A 29 28
24	27 26 25 24 23 22 21 20
23	1F 1E 1D 1C 1B 1A 19 18
22	17 16 15 14 13 12 11 10
21	0F 0E 0D 0C 0B 0A 09 08
20	07 06 05 04 03 02 01 00
1F	Bank 3
18	Bank 2
10	Bank 1
0F	Default register bank for R0-R7
08	
07	
00	
RAM	

Second 128 Bytes, directly addressable. The "SFR's":

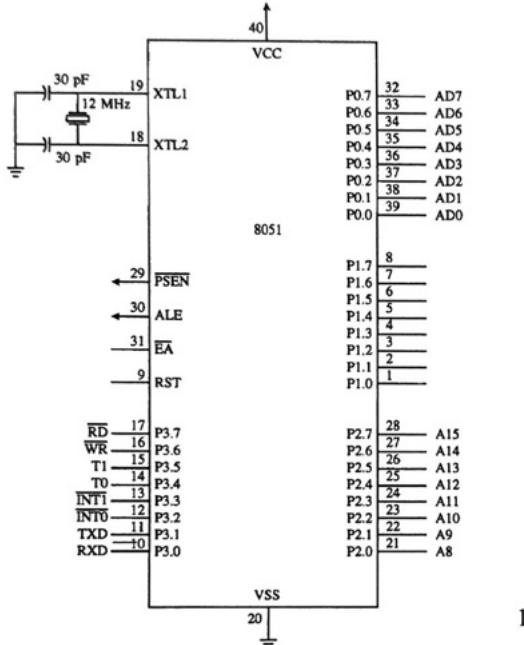
Byte address	Bit address	
FF	F7 F6 F5 F4 F3 F2 F1 F0	B
F0	E7 E6 E5 E4 E3 E2 E1 E0	ACC
E0	D7 D6 D5 D4 D3 D2 - D0	PSW
D0	- - - BC BB BA B9 B8	IP
B8	B7 B6 B5 B4 B3 B2 B1 B0	P3
B0	AF - - AC AB AA A9 A8	IE
A8	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit addressable	SBUF
98	9F 9E 9D 9C 9B 9A 99 98	SCON
90	97 96 95 94 93 92 91 90	P1
8D	not bit addressable	TH1
8C	not bit addressable	TH0
8B	not bit addressable	TL1
8A	not bit addressable	TL0
89	not bit addressable	TMOD
88	8F 8E 8D 8C 8B 8A 89 88	TCON
87	not bit addressable	PCON
83	not bit addressable	DPH
82	not bit addressable	DPL
81	not bit addressable	SP
80	87 86 85 84 83 82 81 80	PO

SPECIAL FUNCTION REGISTERS

Our first "System Study": The R31JP development board.
What might you want in a microcontroller development? Perhaps:

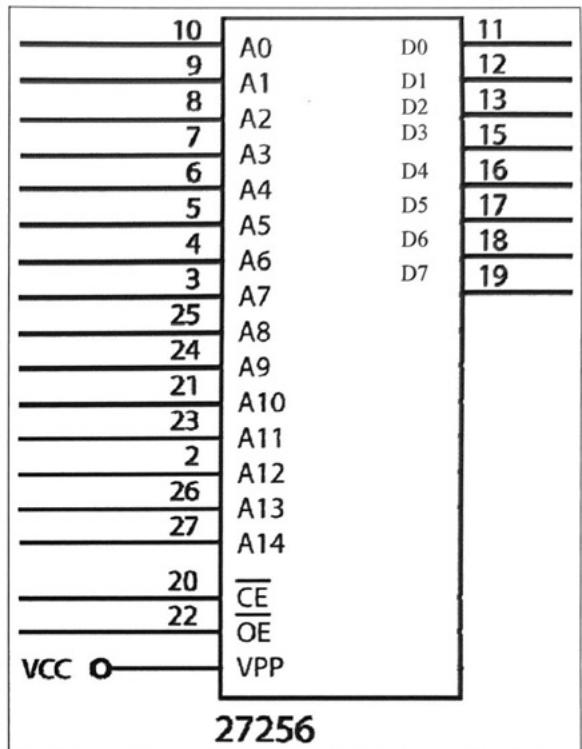
- Easy to change test code
- Plenty of memory for code and data storage
- Requires little specialized/expensive hardware to get started using the system
- Easy interface to other hardware – remember, we're making embedded systems!

The simplest possible development system in term of chip count



1

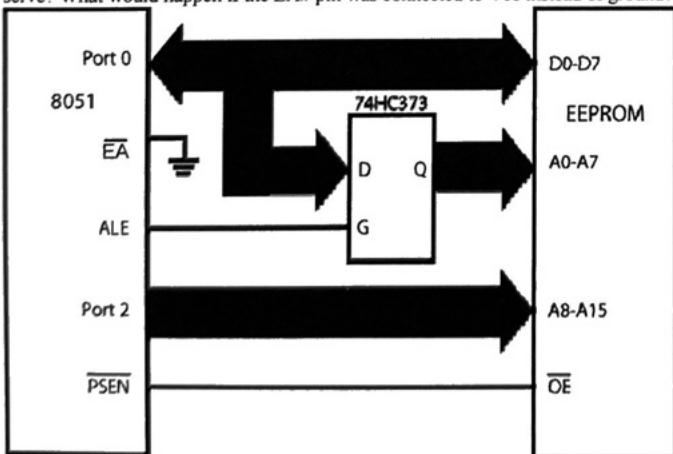
And a typical ROM chip:



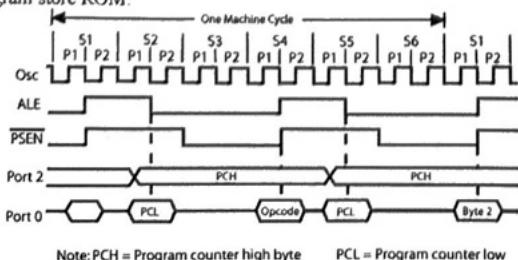
2

The lines CE# and OE# are again active LOW for this ROM. They perform similar functions as for the RAM chip. Why is there no WR# line?

Here is a simple connection scheme to allow a ROM or EPROM or FLASH/EEPROM to serve as "program store for the 8051. What must be done with the CE# connection on the ROM? What purpose does the 74373 transparent latch serve? What would happen if the EA# pin was connected to Vcc instead of ground?



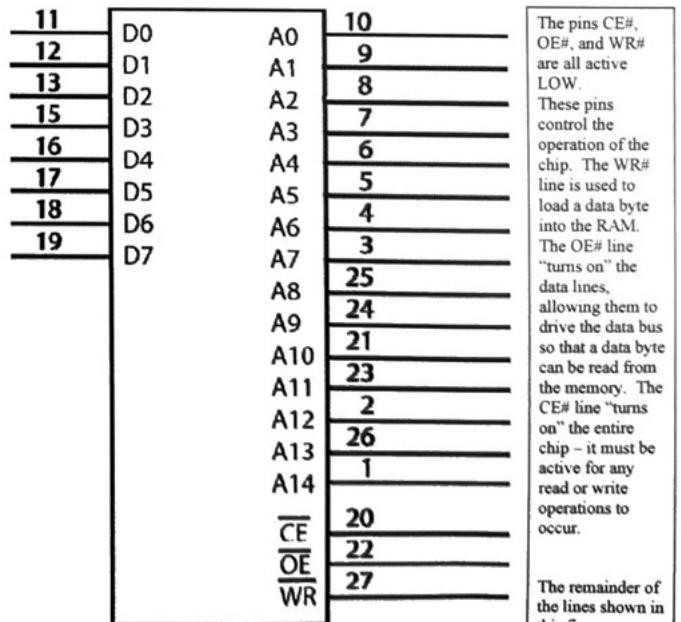
And here is the timing diagram that describes how the 8051 reads op-codes from the program store ROM:



Note: PCH = Program counter high byte

PCL = Program counter low

Typical RAM Chip:



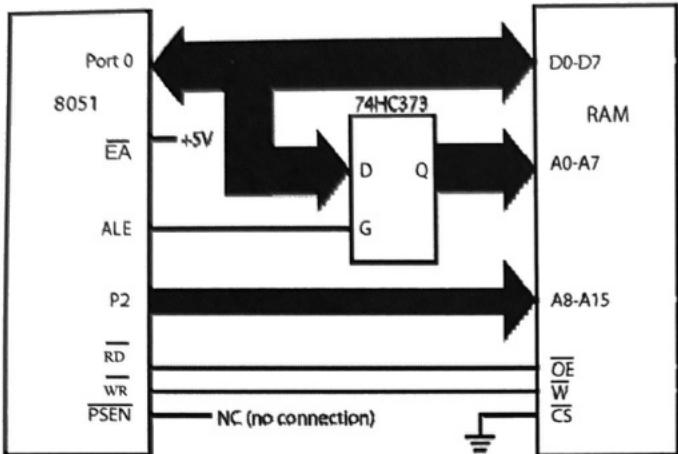
The pins CE#, OE#, and WR# are all active LOW. These pins control the operation of the chip. The WR# line is used to load a data byte into the RAM. The OE# line "turns on" the data lines, allowing them to drive the data bus so that a data byte can be read from the memory. The CE# line "turns on" the entire chip – it must be active for any read or write operations to occur.

The remainder of the lines shown in this figure constitute the address and data bus.

Some lines are missing!
What are they?

(Gnd, Vcc)

If you don't need external program store, you might still need more scratchpad RAM than is available on the 8051. In this case, the program store might be inside the 8051 on-board ROM, and we might connect an external RAM to provide more "variable" space:



This connection provides up to 64K of external scratch RAM. This RAM cannot hold a program, only data! Notice:

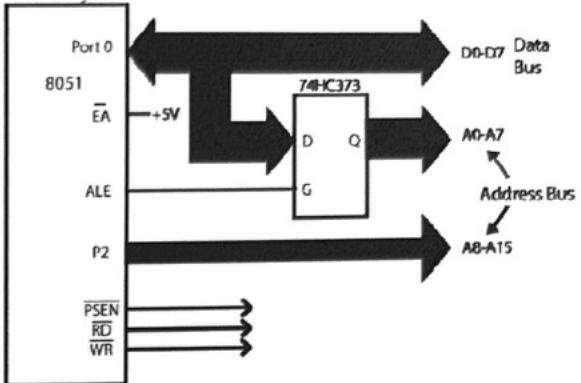
- PSEN# is disconnected
- EA# is HIGH – where is the program stored?
- CS# on the RAM chip is the same as CE# - you'll see both on data sheets
- See the INTEL MCS51 manual for timing diagrams that explain how data is read from or written to the RAM by the 8051.

Now, suppose you wanted to have both external program store and additional, external scratch RAM:

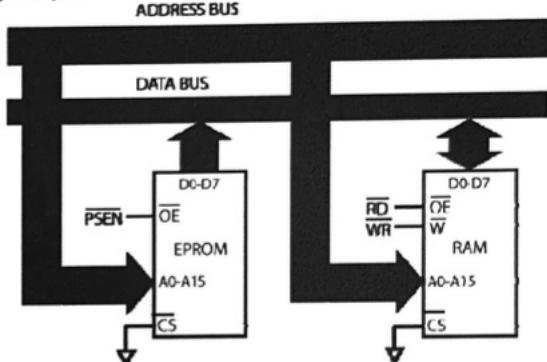
5

HARVARD Architecture: Provides 64K of program store, 64K of data memory.

Processor Subsystem:



Memory Subsystem:

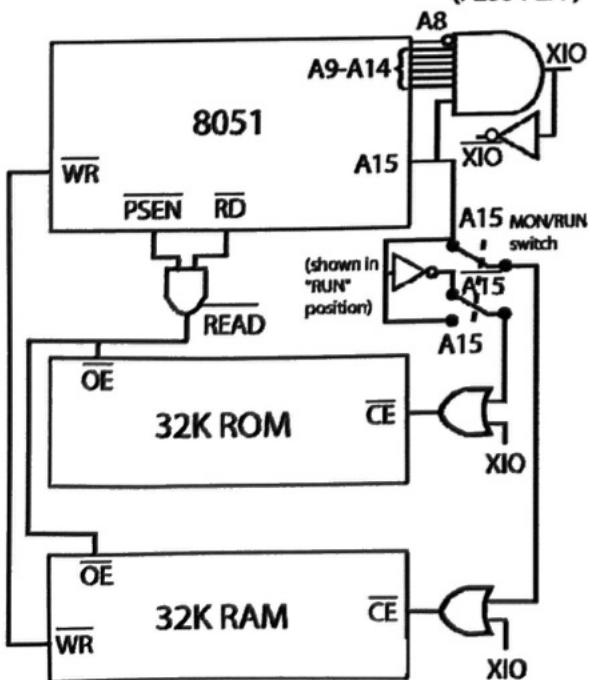


6

LINEAR Architecture with memory-mapped I/O:

Only the control signals are shown. The address bus and data bus must also be connected to the RAM, ROM, and 8051, but have been removed in the figure for clarity. Provides a 64K linear memory space consisting of 32K RAM and 32K ROM. Either the RAM or ROM can fill the first 32K (0000h – 7FFFh) by flipping the DPDT "MON/RUN" switch. XIO# selects memory-mapped I/O devices.

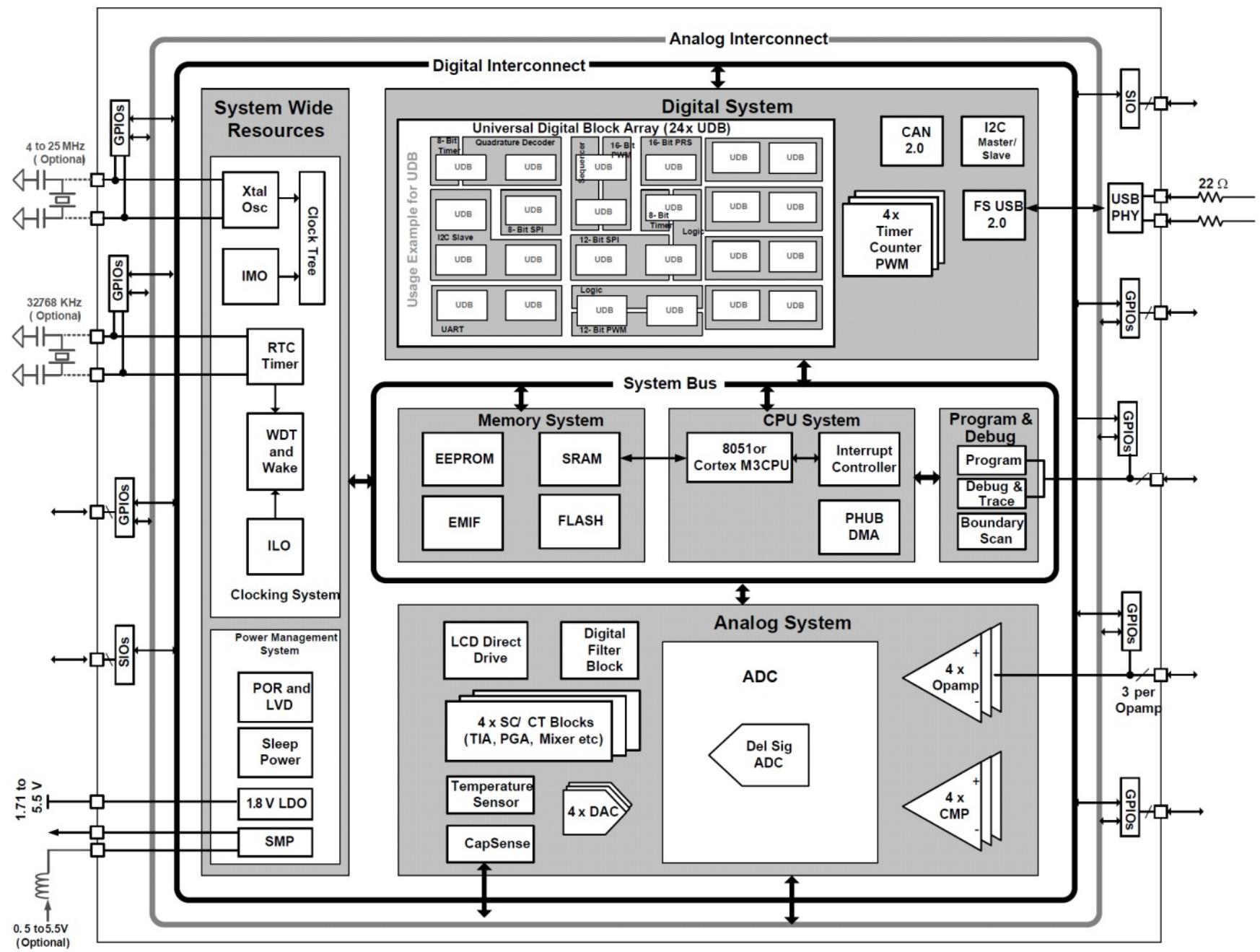
(FE00-FEFF)

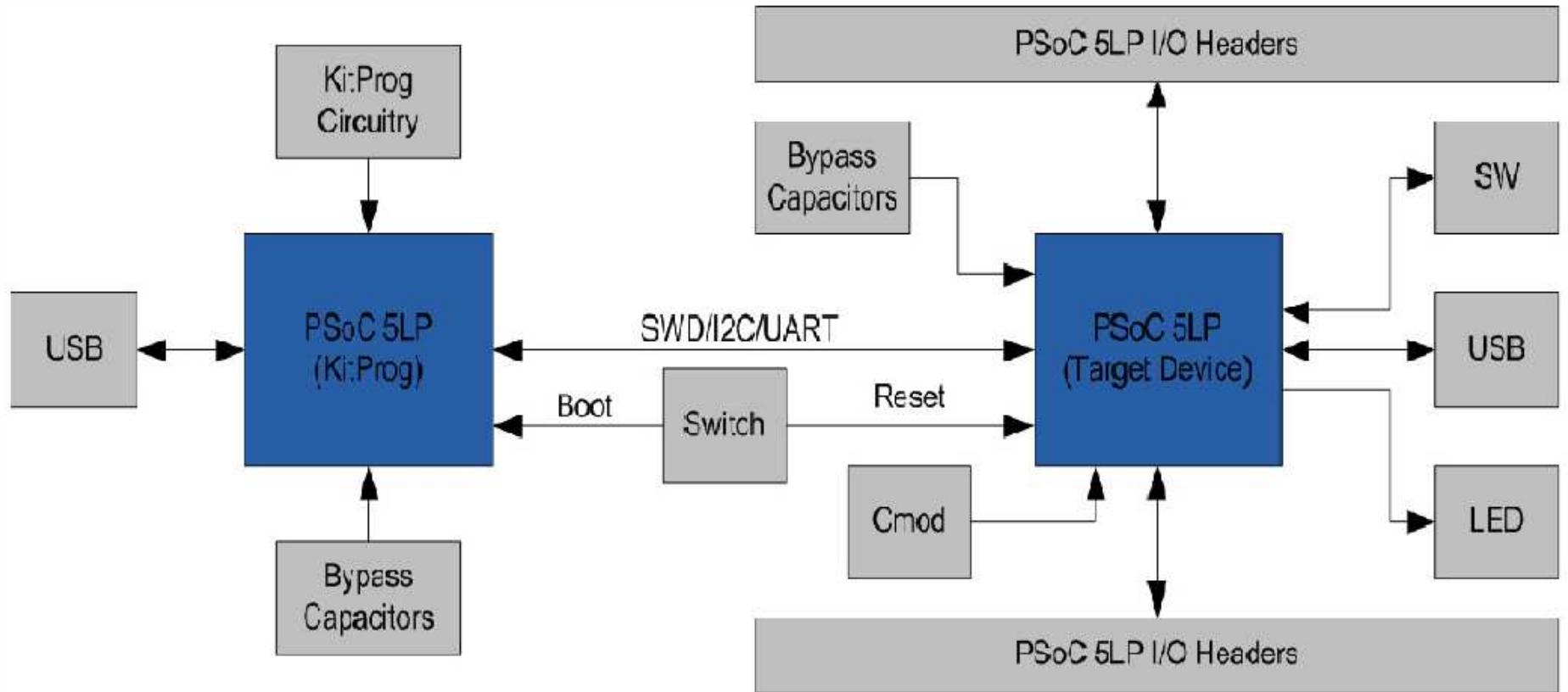


7

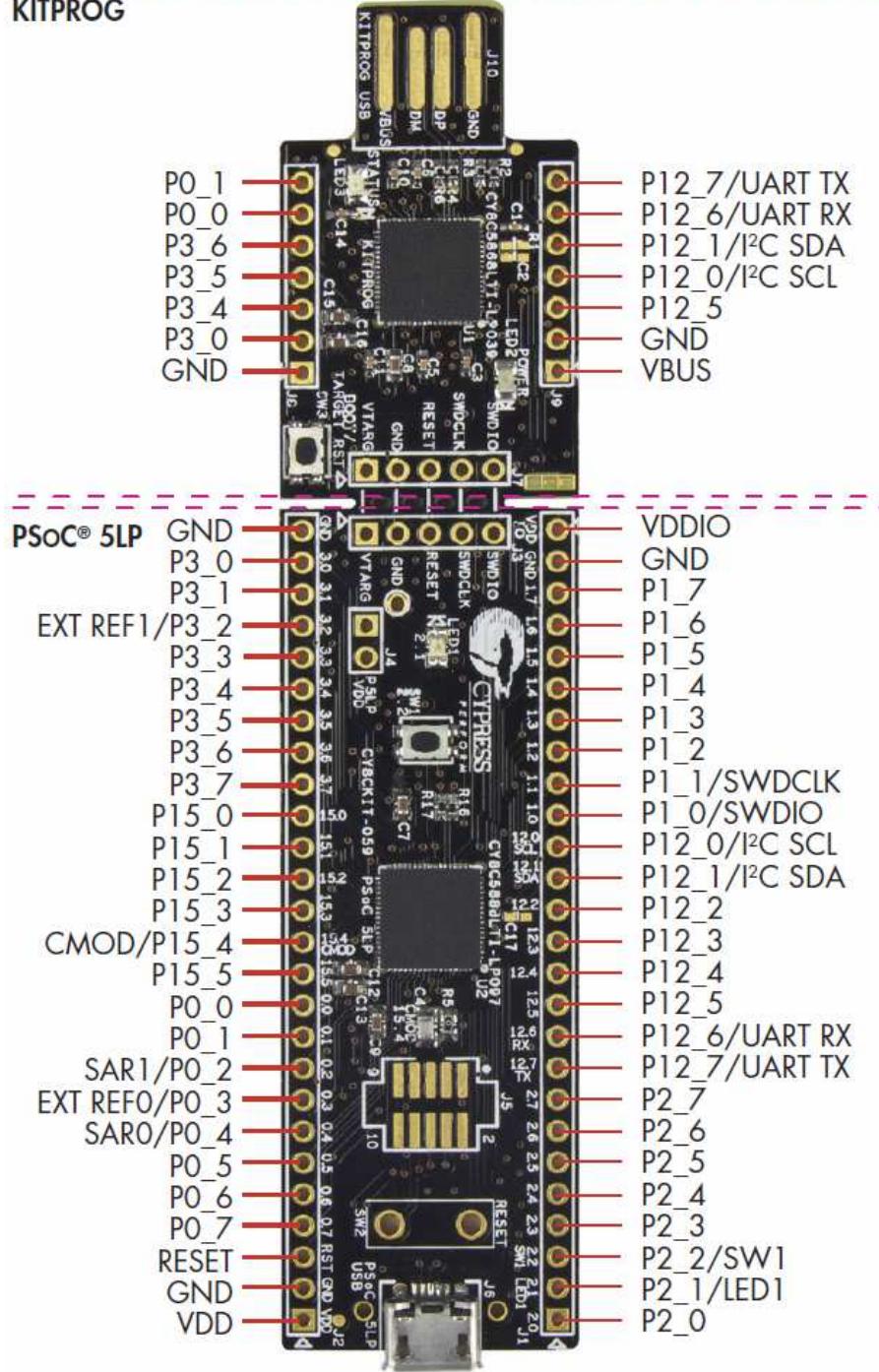
Welcome!

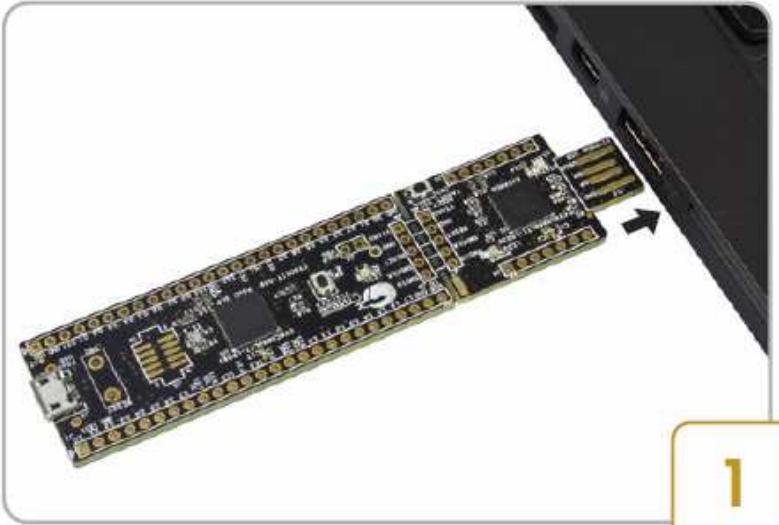
Cypress PSoC:





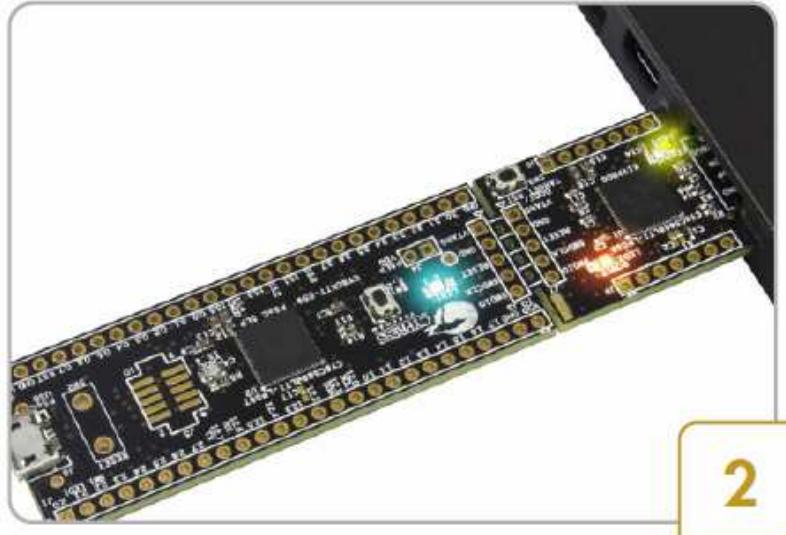
KITPROG





1

- Connect the board to your computer using the USB connector



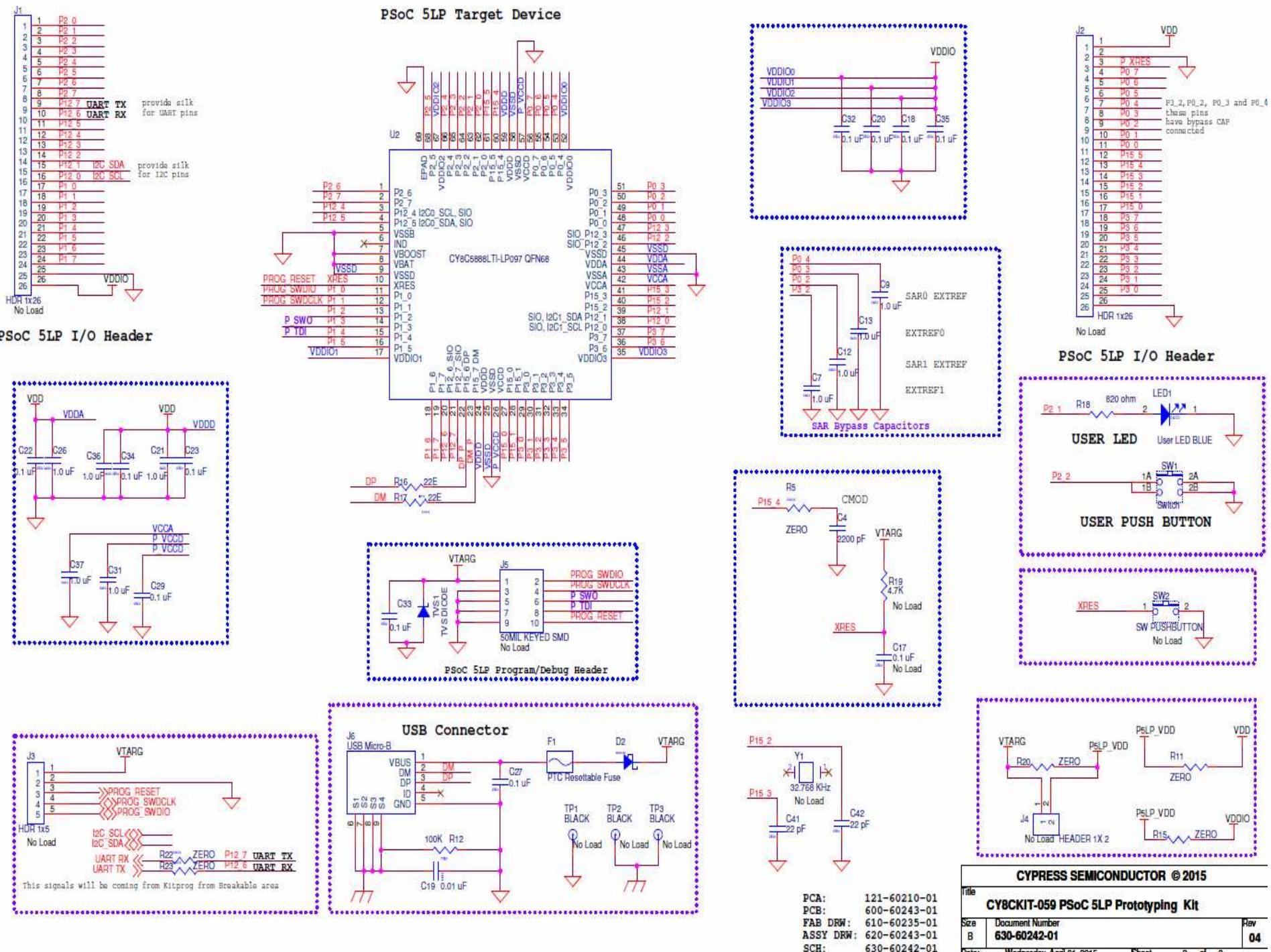
2

- Amber LED indicates power on
- Green LED indicates status
- Blue LED on the board blinks

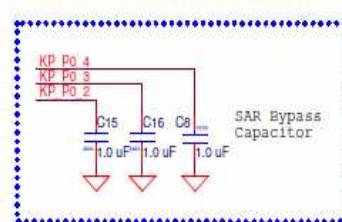
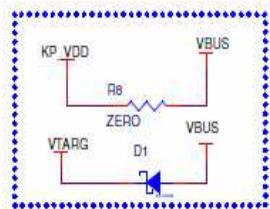
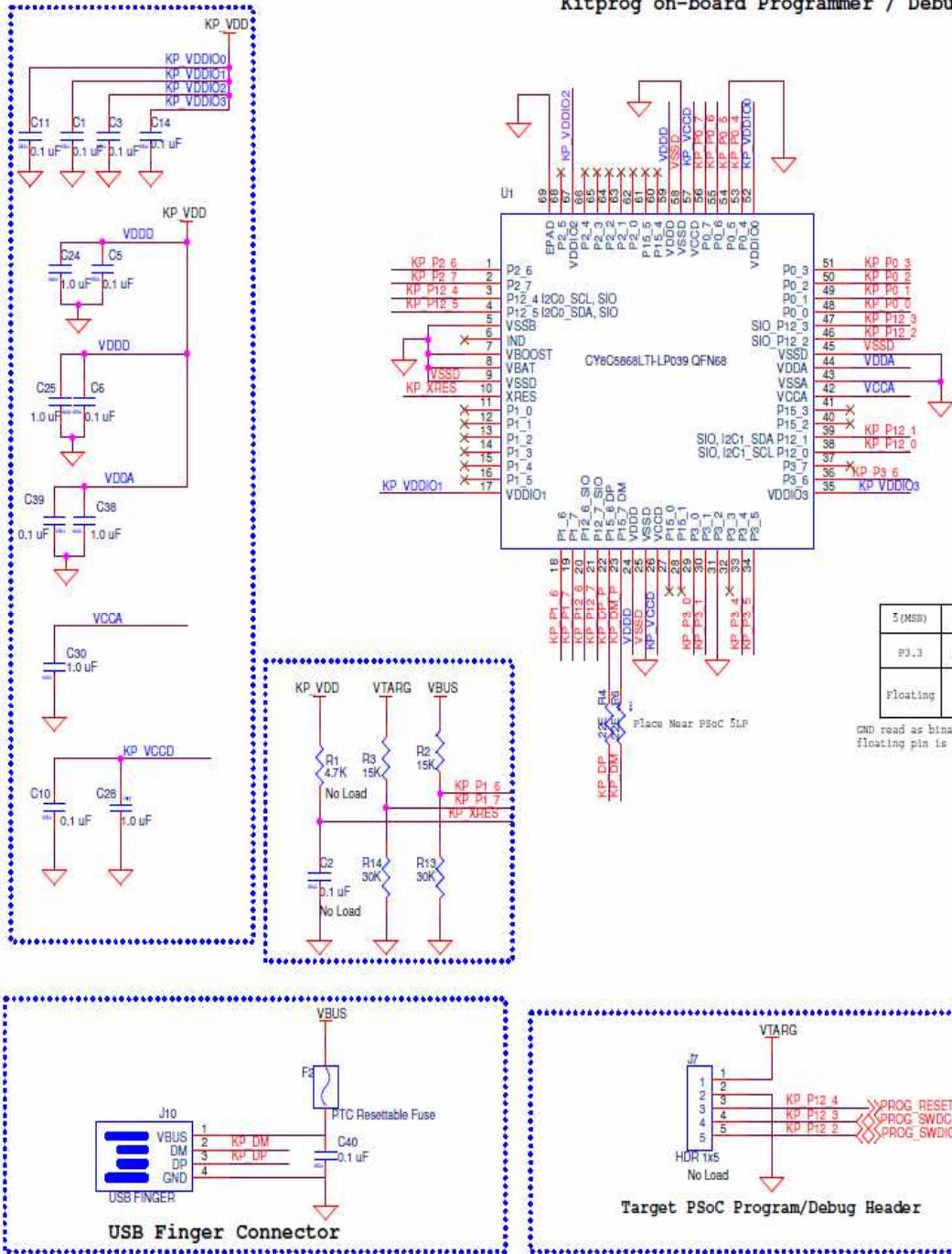
PSoC® 5LP: 32-BIT ARM® CORTEX™-M3 PSOC
Designed for High Performance,
Optimized for Low Power
Now with a Faster CPU at 80 MHz

3

For more information on the kit, please go to the following web page:
www.cypress.com/CY8CKIT-059

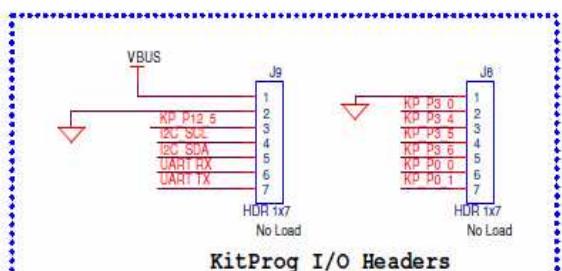
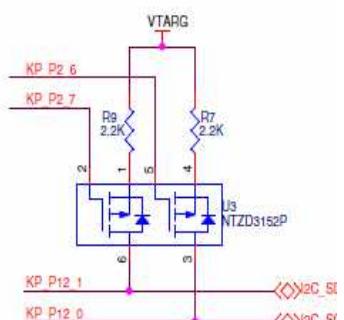
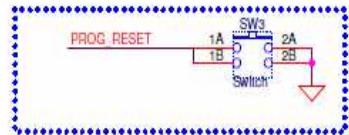
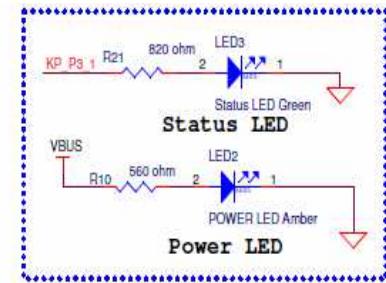


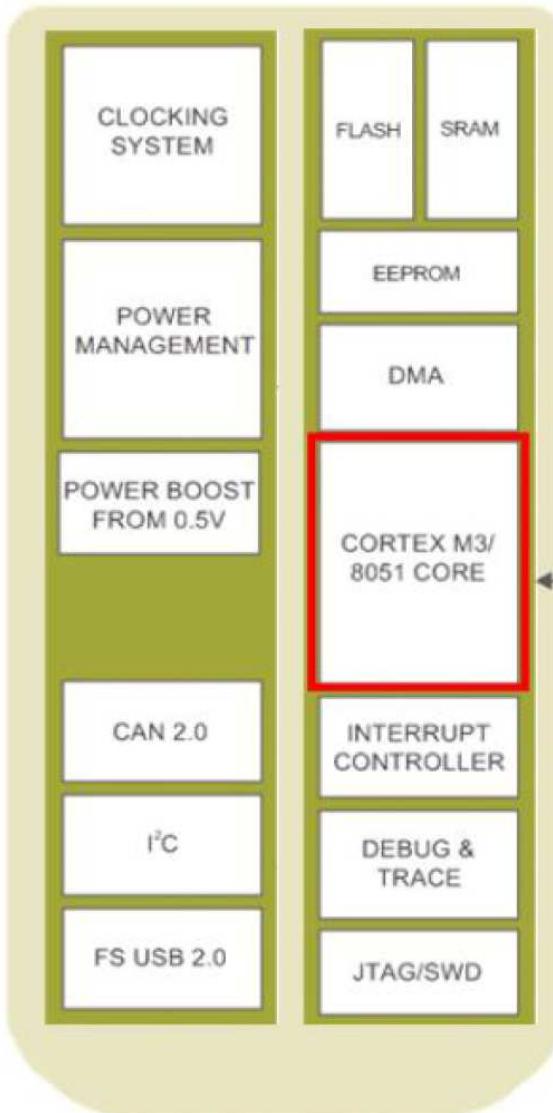
Kitprog on-board Programmer / Debugger



Pins	5 (MSB)	4	3	2	1	0 (LSB)
P3_3	P3_2	P0_7	P0_6	P0_5	P0_4	
Floating	GND	Floating	Floating	GND	Floating	

GND read as binary "1"
floating pin is read as binary "0"





ARM Cortex-M3

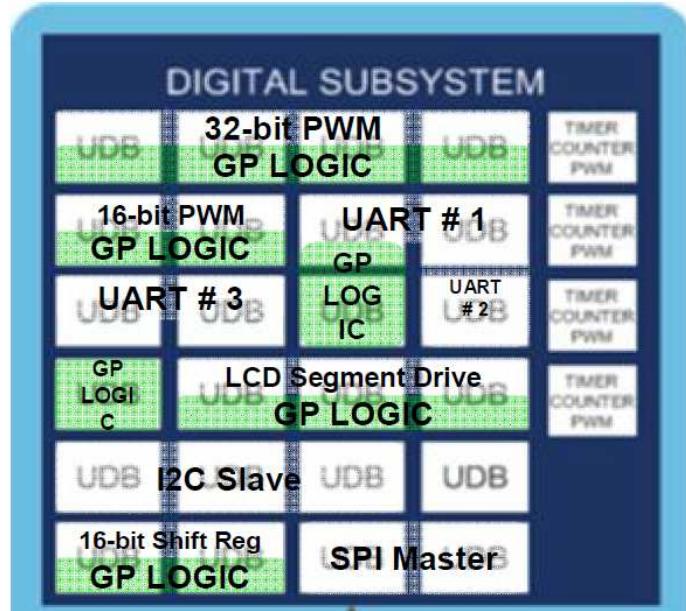
- Industry's leading embedded CPU company
- Broad support for middleware and applications
- Up to 67 MHz; 83 DMIPS
- Enhanced v7 ARM architecture
- Thumb2 Instruction Set
- 16- and 32-bit Instructions (no mode switching)
- 32-bit ALU; Hardware multiply and divide
- Single cycle 3-stage pipeline; Harvard architecture

8051

- Broad base of existing code and support
- Up to 67 MHz; 33 MIPS
- Single cycle instruction set

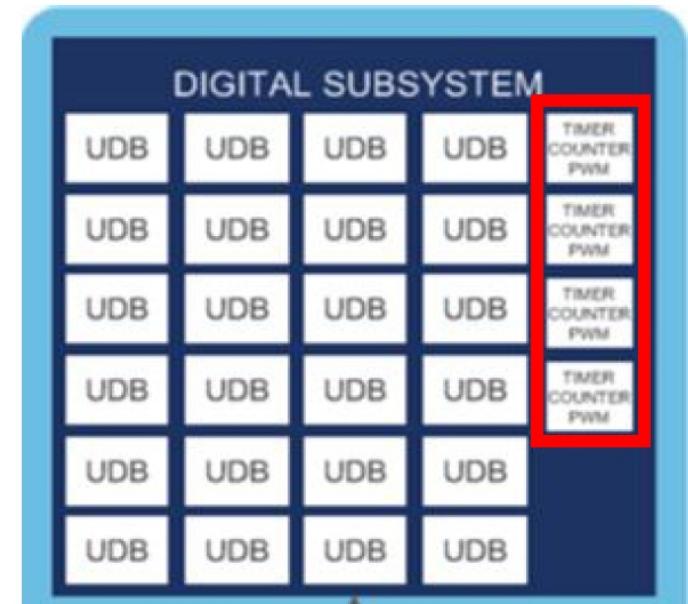
Universal Digital Block Arrays (UDBs)

- Flexibility of a PLD integrated with a CPU
- Provides hardware capability to implement components from a rich library of pre-built, documented and characterized components in PSoC Creator
- PSoC Creator will synthesize, place and route components automatically as well as provide static timing analysis
- Fine configuration granularity enables high silicon utilization
- DSI routing mesh allows any function in the UDBs to communicate with any other on-chip function/GPIO pin with 8- to 32-bit data buses



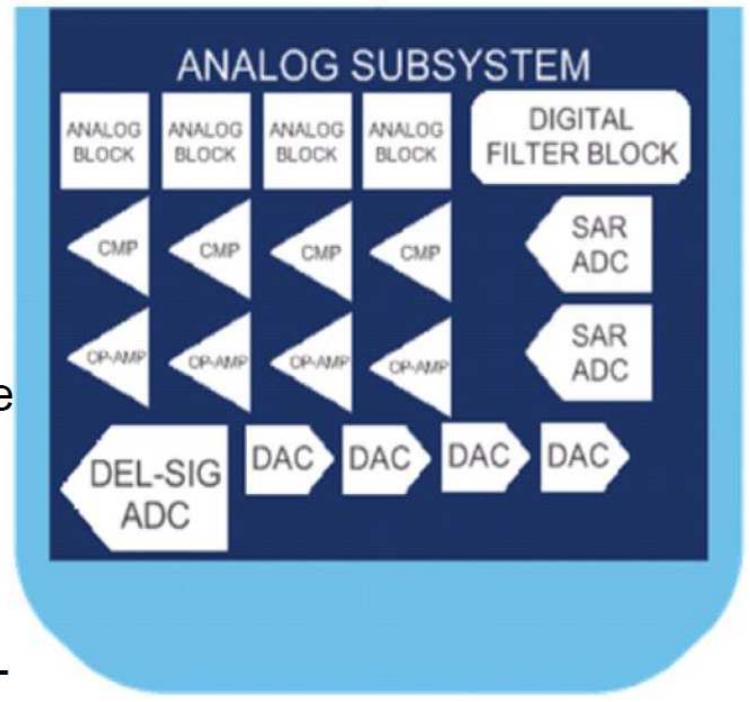
Organized 8/16-bit Timer/Counter/PWM Blocks

- Provides nearly all of the features of a UDB based timer, counter or PWM
- PSoC Creator provides easy access to these flexible blocks
- Each block may be configured as either a full featured 8-bit Timer, Counter or PWM. Two blocks may be combined to make it 16-bit
- Programmable options
 - Clock, enable, reset, capture, kill from any pin or digital signal on chip
 - Independent control of terminal count, interrupt, compare, reset, enable, capture and kill synchronization
- Plus
 - Configurable to measure pulse-widths or periods
 - Buffered PWM with dead band and kill



Configurable Analog System

- Flexible Routing: All GPIO are Analog Input/Output
- +/- 0.1% Internal Reference Voltage
- Delta-Sigma ADC: Up to 20-bit resolution
 - 16-bit at 48 ksps or 12-bit at 192 ksps
- SAR ADC: 12-bit at 700 ksps
- DAC's: 8-bit resolution, current and voltage mode
- Low Power Comparators
- Opamps (25 mA output buffers)
- Programmable Analog Blocks
 - Configurable PGA (up to X50), Mixer, Trans-Impedance Amplifier, Sample and Hold
- Digital Filter Block: Implement HW IIR and FIR filters
- CapSense Touch Sensing enabled

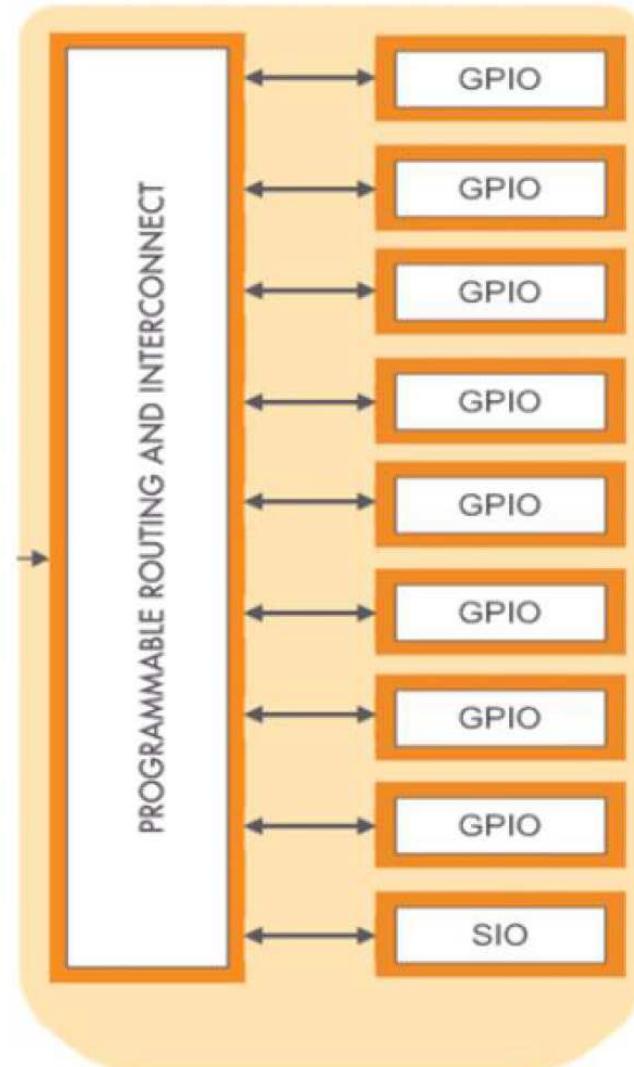


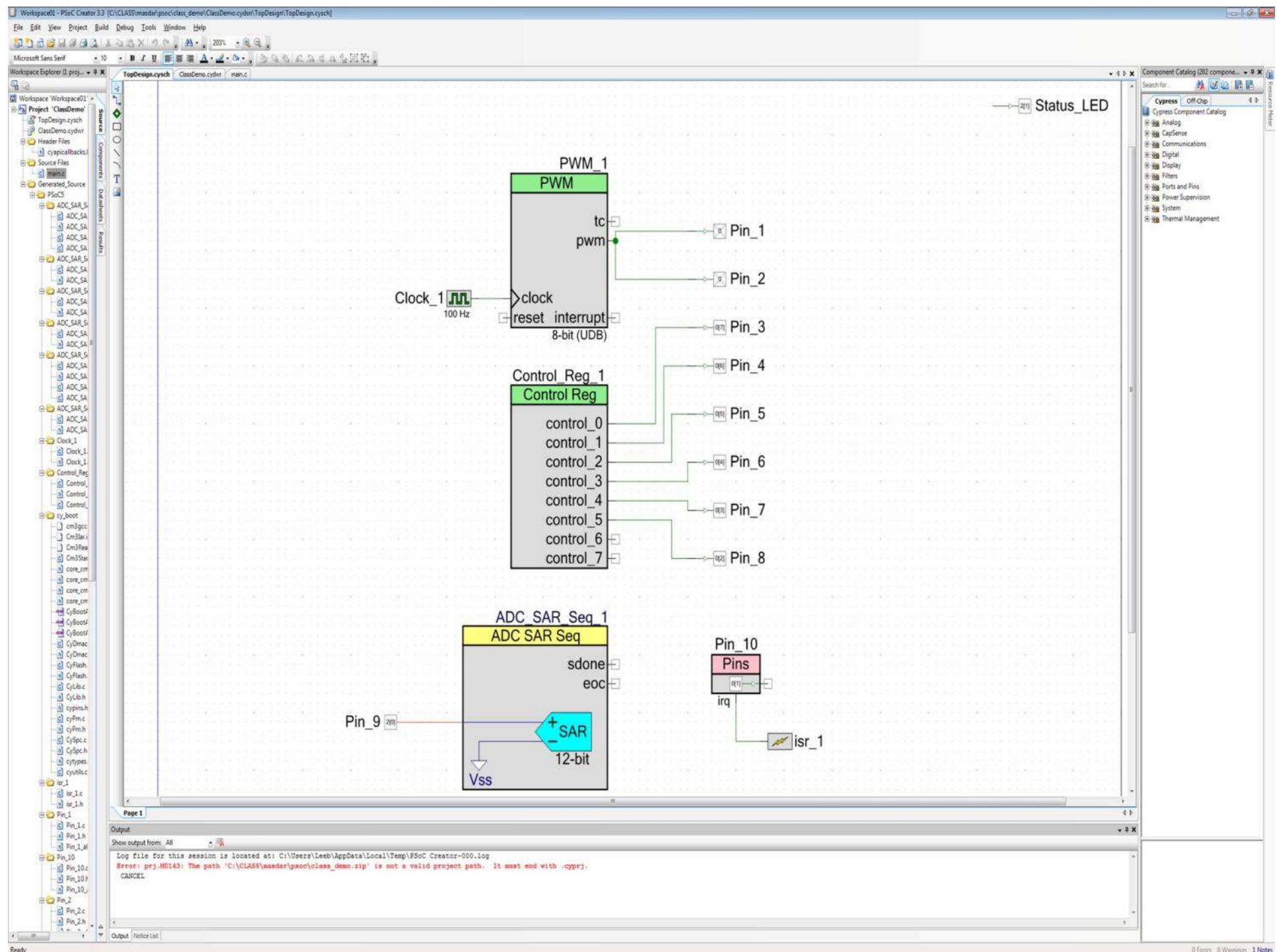
Input / Output System

- Three types of I/O
 - GPIO, SIO, USBIO
- Any GPIO to any peripheral routing
- Wakeup from sleep on analog, digital or I2C events
- Programmable slew rate reduces power and noise
- Eight different configurable drive modes
- Programmable input threshold capability for SIO
- Automatic and custom/lock-able routing in PSoC Creator

Four separate I/O voltage domains

- Interface with multiple devices using one PSoC 3 / PSoC 5 device





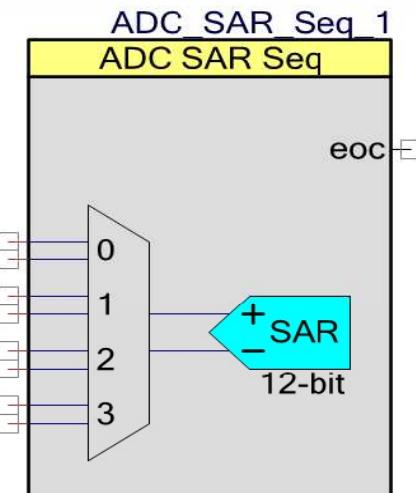
Sequencing Successive Approximation ADC (ADC_SAR_Seq)

2.0

Features

- Supports PSoC 5LP devices
- Selectable resolution (8, 10 or 12 bit) and sample rate (up to 1 Msps)
- Scans up to 64 single ended or 32 differential channels automatically, or just a single input

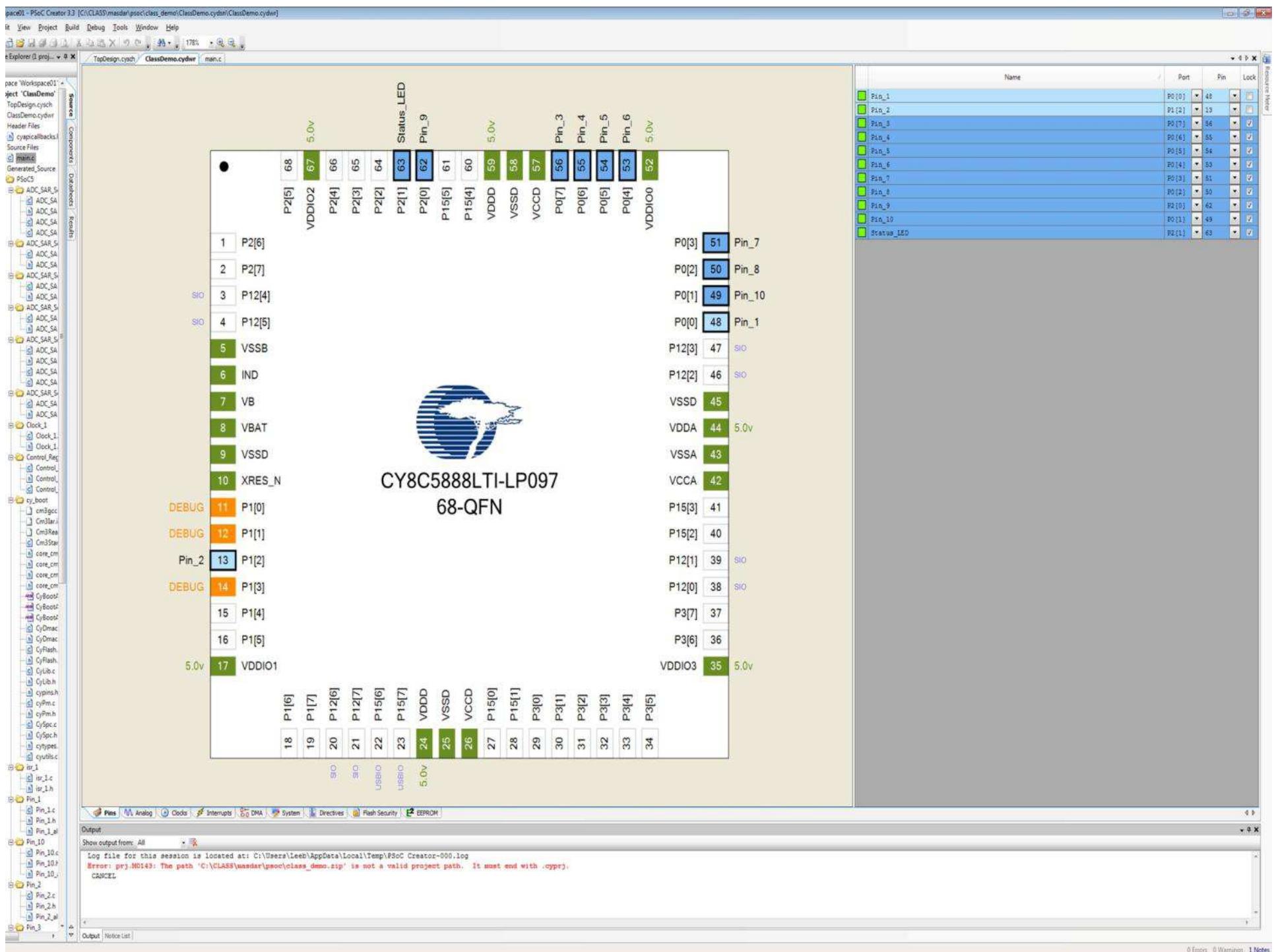
Note Only the GPIOs can be connected to the channels inputs. The actual maximum number of input channels depends on the number of routable analog GPIOs that are available on a specific PSoC part and package.



General Description

The Sequencing SAR ADC component enables makes it possible for you to configure and then use the different operational modes of the SAR ADC on PSoC 5LP. You also have schematic level and firmware level support for seamless use of the Sequencing SAR ADC in PSoC Creator designs and projects. You are able to configure multiple analog channels that are automatically scanned with the results placed in individual SRAM locations.

When to Use the ADC_SAR_Seq



Workspace01 - PSoC Creator 3.3 [C:\CLASS\masdar\psoc\class_demo\ClassDemo.cydsn\main.c]

File Edit View Project Build Debug Tools Window Help

Workspace Explorer (1 proj...)

Project 'ClassDemo'

- TopDesign.cydwr
- ClassDemo.cydwr
- Header Files
- cypiccallbacks.h
- Source Files
- main.c
- Generated Source
- PSoC5
 - ADC_SAR_S
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SAR_S
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SAR_S
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SAR_S
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - ADC_SA
 - Clock_1
 - Clock_1
 - Clock_1
 - Control_Reg
 - Control_Reg
 - Control_Reg
 - Control_Reg
 - cy_boot
 - cm3gcc
 - Cm3lari
 - Cm3Rea
 - Cm3Star
 - core_cm
 - core_cm
 - core_cm
 - CyBoot2
 - CyBoot2
 - CyBoot2
 - CyDmac
 - CyFlash
 - CyFlash
 - CyLibc
 - CyLibh
 - cypins.h
 - cyPm.c
 - cyPm.h
 - CySpec.c
 - CySpec.h
 - cytypes.h
 - cyutils.c
 - isr_1
 - isr_1.c
 - isr_1.h
 - Pin_1
 - Pin_1.c
 - Pin_1.h
 - Pin_1_all
 - Pin_10
 - Pin_10.c
 - Pin_10.h
 - Pin_10_j
 - Pin_10_s

Components

Dependencies

Ready

TopDesign.cydwr ClassDemo.cydwr main.c

```
1 /* * Copyright YOUR COMPANY, THE YEAR
2 * All Rights Reserved
3 * UNPUBLISHED, LICENSED SOFTWARE.
4 *
5 * CONFIDENTIAL AND PROPRIETARY INFORMATION
6 * WHICH IS THE PROPERTY OF your company.
7 */
8 */
9
10 /*
11 */
12 #include <project.h>
13
14 void isr_1_interrupt(void);
15 int flag = 1;
16
17 int main()
18 {
19     CyGlobalIntEnable; /* Enable global interrupts. */
20
21     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
22     int result;
23     int addstuff = 0;
24
25     isr_1_StartEx(isr_1_interrupt);
26     PWM_1_Start();
27     ADC_SAR_Seq_1_Start();
28     ADC_SAR_Seq_1_StartConvert();
29
30     for(;;)
31     {
32         /* Place your application code here. */
33         Control_Reg_1_Write(1+flag+addstuff);
34         CyDelay(250);
35         Control_Reg_1_Write(2+addstuff);
36         CyDelay(250);
37         result = ADC_SAR_Seq_1_GetResult16(0);
38         if (result > 1800) addstuff = 32;
39         else if (result > 1500) addstuff = 16;
40         else if (result > 1000) addstuff = 8;
41         else if (result > 500) addstuff = 4;
42         else addstuff = 0;
43     }
44 }
45 */
46
47 CY_ISR(isr_1_interrupt) {
48     if (flag == 0) flag = 1;
49     else flag = 0;
50     Pin_10_ClearInterrupt();
51 }
52
53 /* {} END OF FILE */
54
```

Output

Show output from: All

Log file for this session is located at: C:\Users\Leeb\AppData\Local\Temp\PSoC Creator-000.log

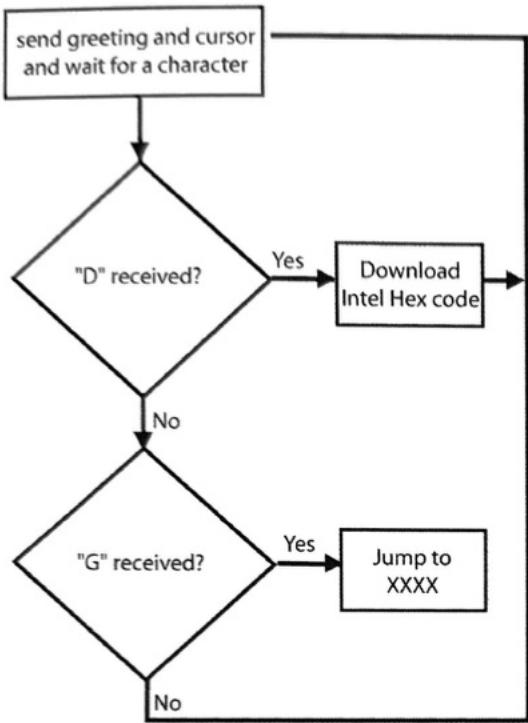
Error: prj.M0143: The path 'C:\CLASS\masdar\psoc\class_demo.zip' is not a valid project path. It must end with .cyprj.

CANCEL

```
#include <project.h>
void isr_1_interrupt(void);
int flag = 1;
int main()
{
    CyGlobalIntEnable; /* Enable global interrupts. */
    int result;
    int addstuff = 0;

    isr_1_StartEx(isr_1_interrupt);
    PWM_1_Start();
    ADC_SAR_Seq_1_Start();
    ADC_SAR_Seq_1_StartConvert();
    for(;;)
    {
        Control_Reg_1_Write(1+flag+addstuff);
        CyDelay(250);
        Control_Reg_1_Write(2+addstuff);
        CyDelay(250);
        result = ADC_SAR_Seq_1_GetResult16(0);
        if (result > 1800) addstuff = 32;
        else if (result > 1500) addstuff = 16;
        else if (result > 1000) addstuff = 8;
        else if (result > 500) addstuff = 4;
        else addstuff = 0;
    }
}
CY_ISR(isr_1_interrupt) {
    if (flag == 0) flag = 1;
    else flag = 0;
    Pin_10_ClearInterrupt();
}
```

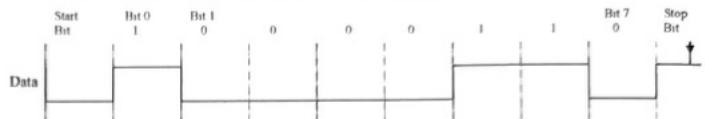
Previously, we've examined the idea of operating the 8051 processor with a common linear memory space for code and data. We need an "operating system":



1

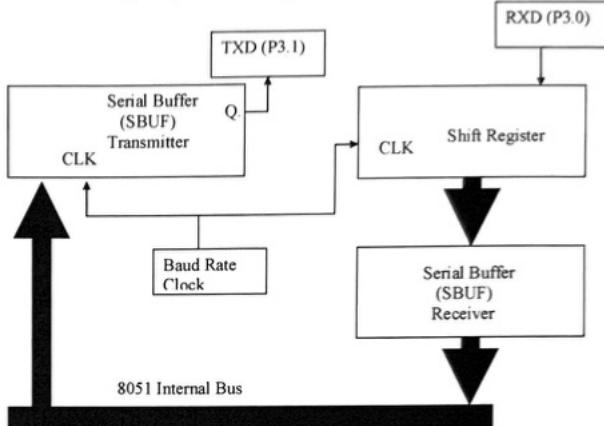
To develop this OS, we need a scheme for communicating with a personal computer/terminal.

RS232: (with asynchronous 8 bit serial data transmission)



For this pattern, the transmitted data is 61h. Is bit 1 the LSB or the MSB?

The 8051 family incorporates a serial port:



2

Data is transmitted and received using the serial port through the SBUF SFR. The behavior of the serial port is controlled or programmed through the SCON SFR:

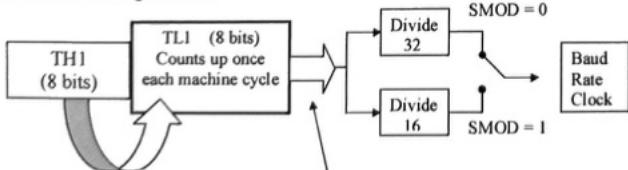
SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

We'll want to transmit 7 bit ASCII codes back and forth from the PC. So, we'll need to configure the serial port:

SM0	SM1	0 0 Mode 0: Shift register, fixed clock
0	1	Mode 1: 8-bit UART, Baud rate set by Timer 1
1	0	Mode 2: 9-bit UART, Fixed baud rate (set by oscillator frequency)
1	1	Mode 3: 9-bit UART, Variable baud rate set by Timer 1

For ASCII transmissions without error correction, Mode 1 will be fine.

Set the baud rate using Timer 1:



When TL1 overflows, "auto re-load" the 8 bit TL1 counter register with the byte in TH1.

One pulse here every time TL1 overflows.
(FFh to 00h transition)

What's the baud rate?

So, for instance, pick TH1 = -3, i.e., TH1 = FDh, for a baud rate of 9600 with an 11.0592 crystal.

$$\text{Baud Rate} = \frac{\text{Clock Crystal} * 2^8 (\text{SMOD})}{12 * 32 * (256 - \text{TH1})}$$

Typically can tolerate up to 4% error in baud rate.

TMOD Register

Time/Counter 1				Timer/Counter 0			
GATE#	C/T#	M1	M0	GATE#	C/T#	M1	M0

M0 and m1 (bits 0 and 1 of TMOD, respectively): *Mode Select*

The 2-bit field (M1, M0) selects one of four modes. Mode 0 is a 13-bit counter. An interrupt is generated when the counter overflows. Thus it takes 2^{13} or 8192 input pulses to generate the next interrupt. Mode 1, similar to Mode 0, implements a 16-bit counter. It takes 2^{16} or 65,526 input pulses to generate the next interrupt. Mode 2 operates in an 8-bit reload fashion. TL1 serves as an 8-bit timer/counter. When the counter overflows, the number stored in TH1 is copied into TL1 and the count continues. An interrupt is generated each time the counter overflows and a reload is performed. In Mode 3, Timer 1 is inactive and simply holds its count. Timer 0 control bits and generates a Timer 0 interrupt at overflow. TH0 operates as a time driven by the system clock, prescaled by 12, and causes a Timer 1 interrupt at overflow.

C/T# (bit 2): *Counter/Timer Select*

When set, the timer/counter operates as a counter. The count increment is caused by external pulses through the T0 pin—the alternative function of P3.4. When cleared the timer/counter operates as a timer. The count increment is caused by every 12th system clock pulse. That is, the input to the counter is the system clock prescaled by 12.

GATE# (bit 3): *Gate*

Provided that TR0 of TCON (see next page) is set, clearing GATE enables (starts) counter/timer operation. If GATE is set, again provided that TR0 is set, the timer/counter operation is enabled if INT0# is high.

TCON Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

IT0 (bit 0): *Interrupt 0 Type*

External interrupt 0 is received through bit 2 of Port 3 as an alternative function assignment. Setting IT0 causes the INT0# to be recognized at the falling edge of the input signal. Clearing IT0 causes an interrupt to be generated when the external signal is low. If the external signal stays low, INT0 will be generated over and over if IT0 is cleared but will not be generated if IT0 is set. IT0 is completely under software control.

IE0 (bit 1): *Interrupt 0 Edge Flag*

Set by hardware when an external interrupt edge is detected. Cleared when a Return From Interrupt (RETI) instruction is executed.

IT1 (bit 2): *Interrupt 1 Type*

IT1 is associated with External Interrupt 1. Its function is similar to that of IT0 as described above.

IE1 (bit 3): *Interrupt 1 Edge Flag*

IE1 is associated with External Interrupt 1. Its function is similar to that of IE0 as described above.

TR0 (bit 4): *Timer 0 Run Control Bit*

Timer/Counter 0 is disabled when TR0 is cleared. Setting TR0 is necessary but not sufficient to enable Timer/Counter 0 (see GATE# AND INT0#). TR0 is completely under software control.

TF0 (bit 5): *Timer 0 Overflow Flag*

TF0 is set by hardware when Timer/Counter 0 overflows. TF0 is cleared by hardware when the processor branches to the associate interrupt service routine. TF0, along with IE0, may be used to software to determine the state of the timer.

TR1 (bit 6): *Timer 1 Run Control Bit*

TR1 is associated with Timer/Counter 1. Its function is similar to that of TR0 described above.

TF1 (bit 7): *Timer 0 Overflow Flag*

TF1 is associated with Timer/Counter 1. Its function is similar to that of TF0 described above.

EXAMPLE: Let's set the serial port to operate at 2400 baud, 8 bits, using Timer 1 to provide the baud rate clock. Assume 11.0592 crystal. We need to set up a total of 4 registers!

Baud rate generation:

TMOD: set to #0010000b = #20h
TCON: set to #0100000b = #40h

After computation,

TH1: set to #-12 = #F4h (assuming SMOD = 0)

Finally, fire up the serial port:

SCON: set to #0101000b = #50h

How might we actually set these registers in assembly language?
Here's a code segment:

Init:

```
MOV TMOD, #20h
MOV TCON, #40h
MOV TH1, #-12
MOV SCON, #50h
RET
```

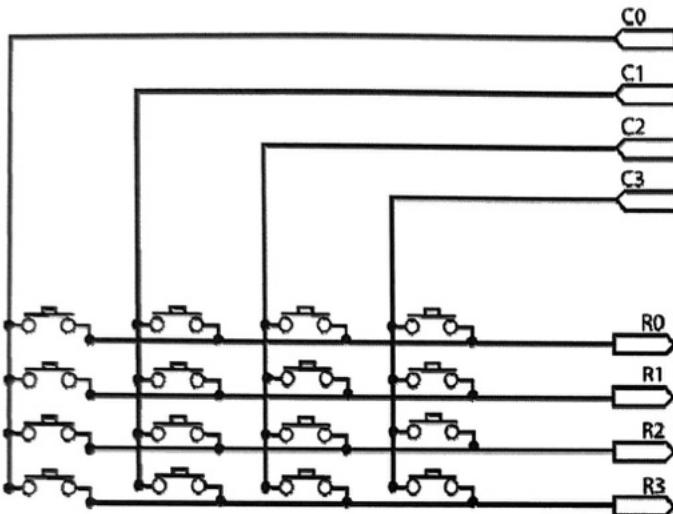
In your MINMON handout, you'll find an initialization subroutine for 9600 baud:

```
;-----
; subroutine init
; this routine initializes the hardware
;-----
init:
; set up serial port with a 11.0592 MHz crystal,
; use timer 1 for 9600 baud serial communications
    mov tmod, #20h          ; set timer 1 for auto reload - mode 2
    mov tcon, #41h           ; run counter 1 and set edge trig ints
    mov th1, #0fdh           ; set 9600 baud with xtal=11.0592mhz
    mov scon, #50h           ; set serial control reg for 8 bit data
                           ; and mode 1
ret
```

Adding peripheral and interface chips to your microcontroller. Why bother?

- Add new functionality to the system
- Relieve the micro of burdensome tasks
- Alter signal types (digital to analog, etc.)
- Alter signal levels (voltage, current, power)

EXAMPLE: Adding a "MATRIX" keypad to your kit



We could use port 1 on the microcontroller to "read" the keypad!

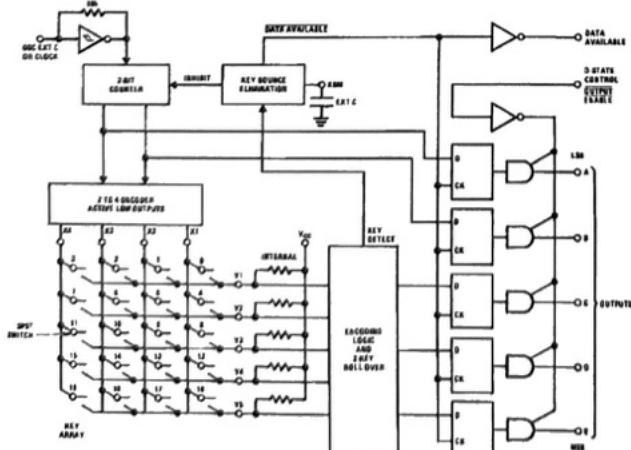
1

2

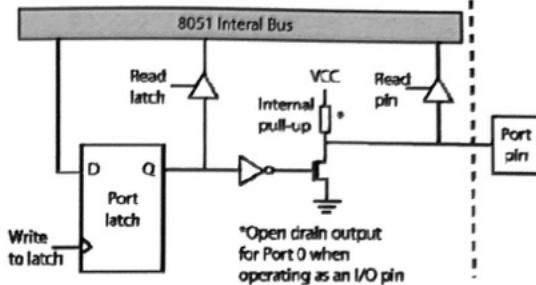
This task is complicated by switch "bounce":



Reading the keypad is a burdensome task. Get help! 74C922:

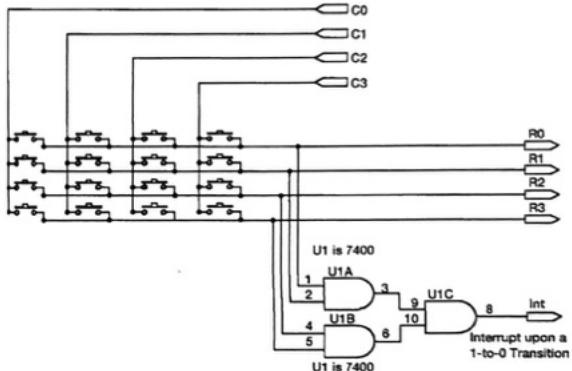


Here is the internal diagram of a port 1 pin:



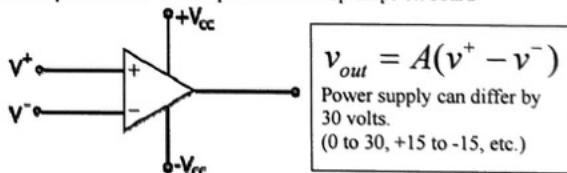
First, write a "1" to this pin to set it up for use as an input port.
Discuss approach for reading a keypress.

Could add some additional logic to try to minimize the micro's burden:

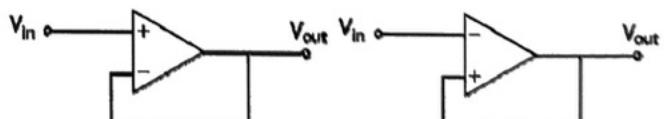


Signal Conditioning with OP-AMPS:

Example: LM358 → an 8 pin IC with 2 op-amps on board



OP-AMPS are generally configured with feedback. Consider these two possibilities:



$$V_{out} = A(v_{in} - v_{out})$$

let $A \rightarrow \infty$

$$0 = v_{in} - v_{out}$$

$$V_{out} = V_{in}$$

STABLE
NEGATIVE FEEDBACK

USE THIS CIRCUIT!
⊗ A BUFFER

$$V_{out} = A(v_{out} - v_{in})$$

let $A \rightarrow \infty$

$$0 = v_{out} - v_{in}$$

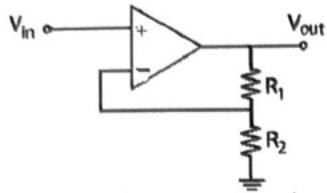
$$V_{out} = V_{in}$$

UNSTABLE
POSITIVE FEEDBACK

WILL NOT WORK!!!

A short catalog of other useful circuits:

- Non-Inverting Amplifier



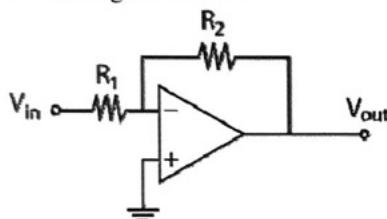
$$v_{out} = A \left(v_{in} - \frac{R_2}{R_1 + R_2} \cdot v_{out} \right)$$

let $A \rightarrow \infty$

$$v_{in} = \frac{R_2}{R_1 + R_2} \cdot v_{out} \text{ or } v_{out} = \frac{R_1 + R_2}{R_2} \cdot v_{in}$$

Can make a “variable gain” with a POT
Can limit gain with series resistors.

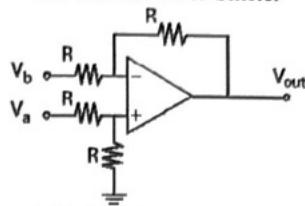
- Inverting Gain Block



$$v_{out} = -\frac{R_2}{R_1} v_{in}$$

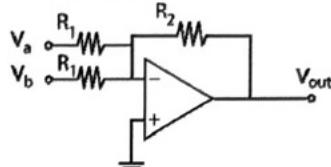
5

- Subtractor/ Level-Shifter



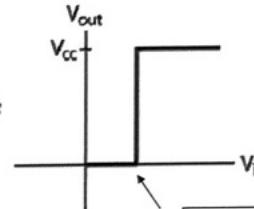
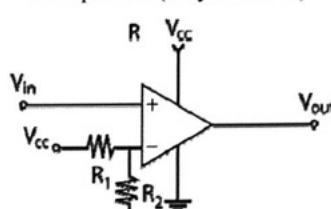
$$v_{out} = (v_a - v_b)$$

- Initial Adder



$$v_{out} = -\frac{R_2}{R_1} (v_a + v_b)$$

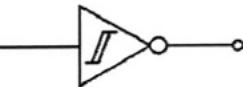
- Comparator (very crude ☺)



For critical apps, use a real comparator, eg. LM311.

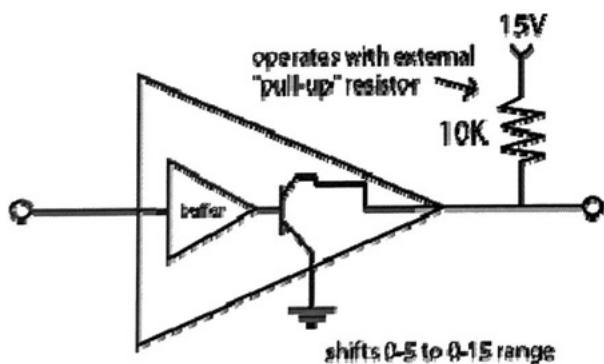
$$V_{cc} \times \frac{R_2}{R_1 + R_2}$$

- Schmitt Trigger Inverter
for cleaning up 0–5 V signals

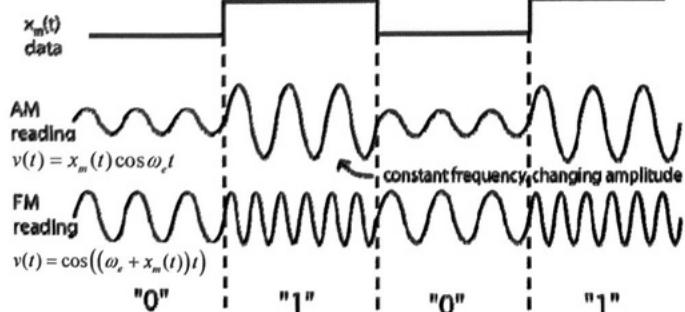


6

- Open collector Buffer → for “level shifting” e.g. 7407:



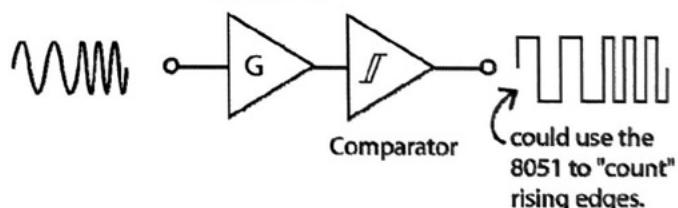
EXAMPLE → COMMUNICATION SYSTEM RECEIVER
Comm Signal:



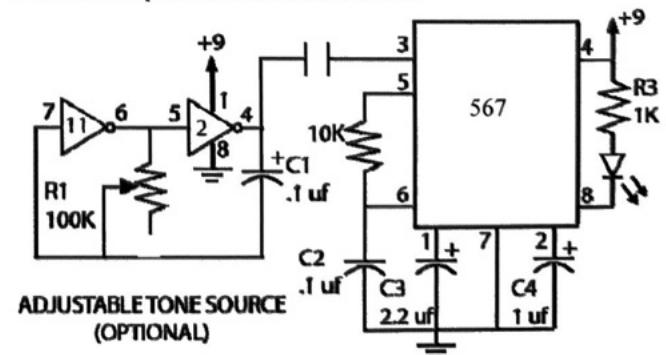
How do we “decode” frequencies (in an FM input signal) to recover $x_m(t)$.

FM’s neat → use huge input gain and a comparator or saturating op-amp

Gain Block



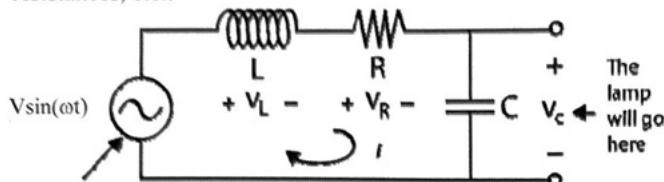
Or, Use a chip like the 567 tone decoder!



7

8

Ballast circuit with NO lamp, but some parasitic R from wiring resistances, etc.:



This is a square wave in our circuit. We'll "approximate" the square wave by its fundamental for now.

Question: What's V_c , the capacitor (& lamp) voltage?

A little 8.02:

$$V \sin \omega t = V_L + V_R + V_C$$

or

$$V \sin \omega t = L \frac{di}{dt} + Ri + V_C \quad \text{but } i = C \frac{dV_C}{dt}$$

so

$$V \sin \omega t = LC \frac{d^2 V_C}{dt^2} + RC \frac{dV_C}{dt} + V_C$$

What's V_C ? Guess a particular solution

Try:

$$V_C = A \sin(\omega t - \phi)$$

Substitute it in and see what happens:

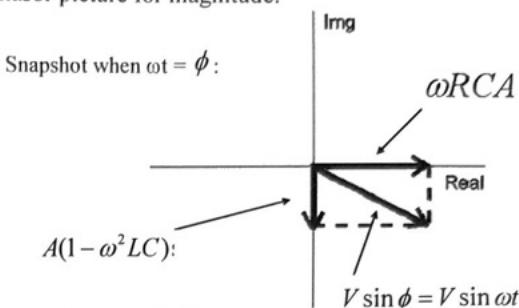
$$V \sin \omega t = -\omega^2 L C A \sin(\omega t - \phi) + \omega R C A \cos(\omega t - \phi) + A \sin(\omega t - \phi)$$

1

or

$$V \sin \omega t = \omega R C A \cos(\omega t - \phi) + A(1 - \omega^2 L C) \sin(\omega t - \phi)$$

Phasor picture for magnitude:



so, for magnitude:

$$V = A \sqrt{(\omega R C)^2 + (1 - \omega^2 L C)^2}$$

$$\text{or } A = \frac{V}{\sqrt{(\omega R C)^2 + (1 - \omega^2 L C)^2}}$$

(explore what happens at $\omega = \frac{1}{\sqrt{LC}}$)

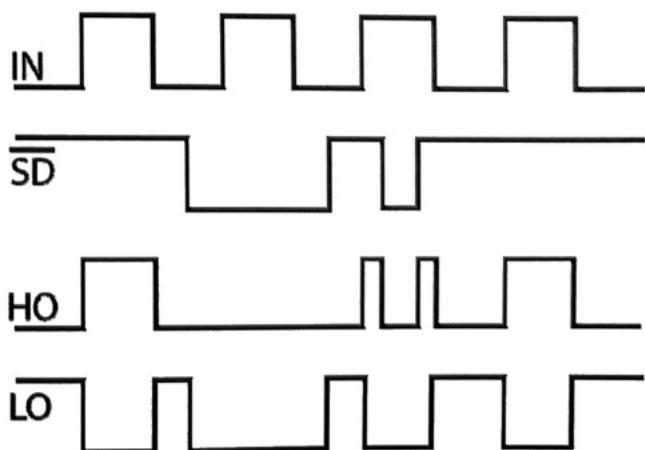
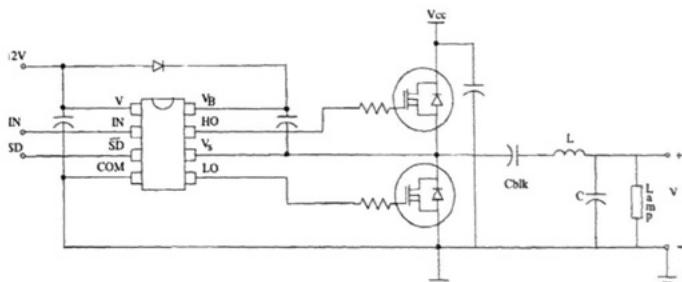
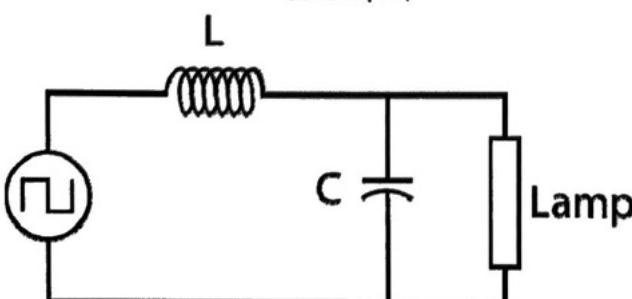
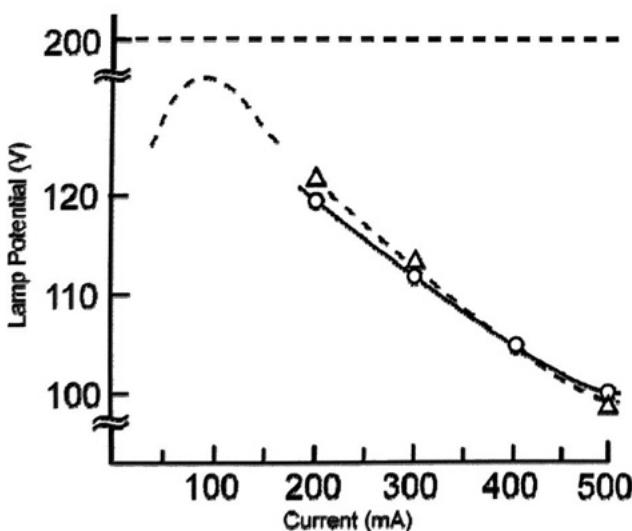
For completeness, evaluate at $\omega t = 0$ to eliminate ϕ :

$$[-\sin(-\phi)]A(1 - \omega^2 L C) = \omega R C A \cos \phi$$

$$\tan \phi = \frac{\omega R C}{(1 - \omega^2 L C)}$$

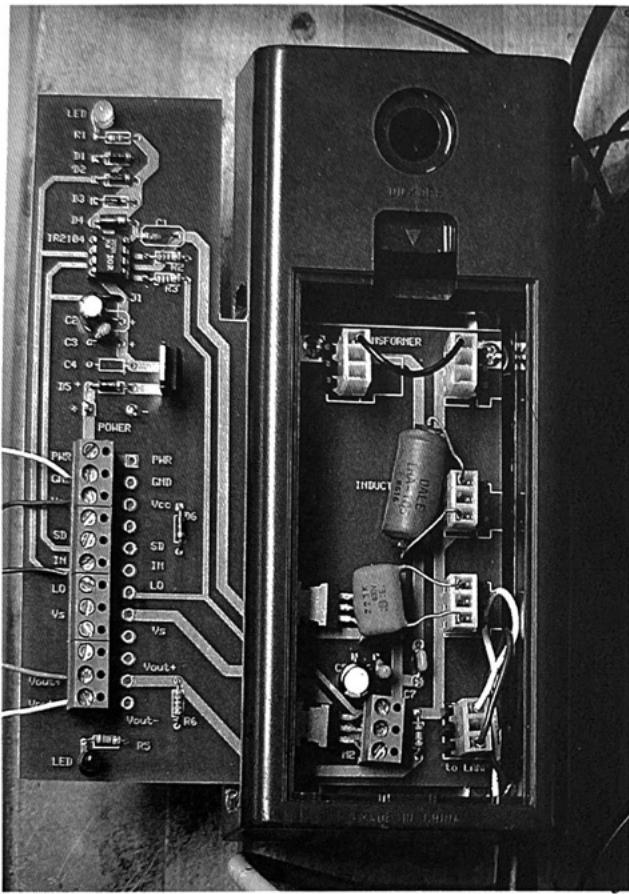
$$\phi = \tan^{-1} \frac{\omega R C}{(1 - \omega^2 L C)}$$

2

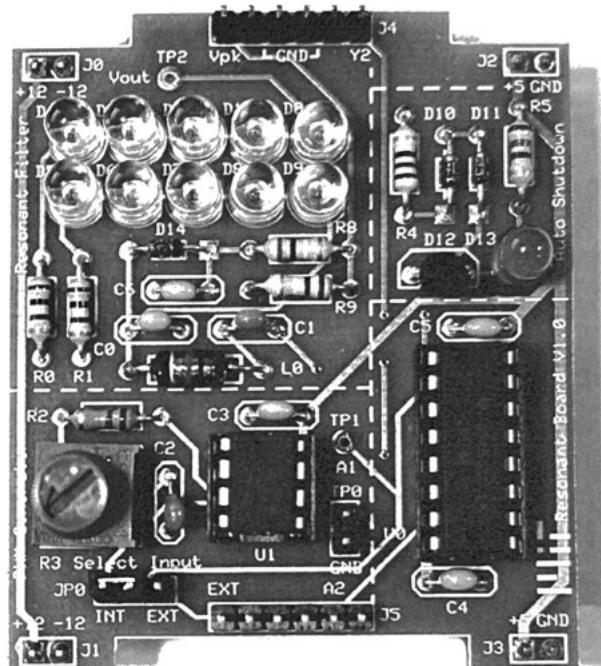


3

4

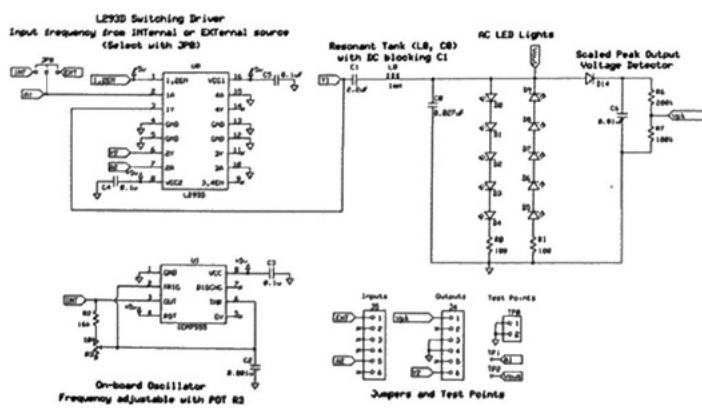


Resonant converter with LED lamp:



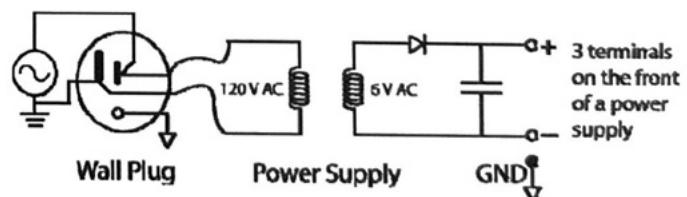
6

Schematic of Resonant Converter with LED lamp:

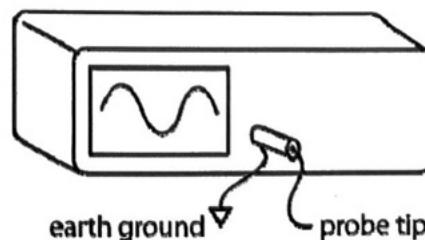


Electrical test equipment:

Example: Power supply, internal construction. Very simplified ☺:



Oscilloscope:



With our lab power supplies, it is your choice whether to ground the output or leave it floating.

Remember, however, an oscilloscope ground alligator clip is always corrected to earth ground on a healthy scope.

6.115 Mini-Quiz (Don't worry, it's not graded ☺)

Please check out these two pieces of code and answer the two questions below:

Code I

```
.ORG 0000h  
  
main:  
    mov P1, #5  
    ljmp main
```

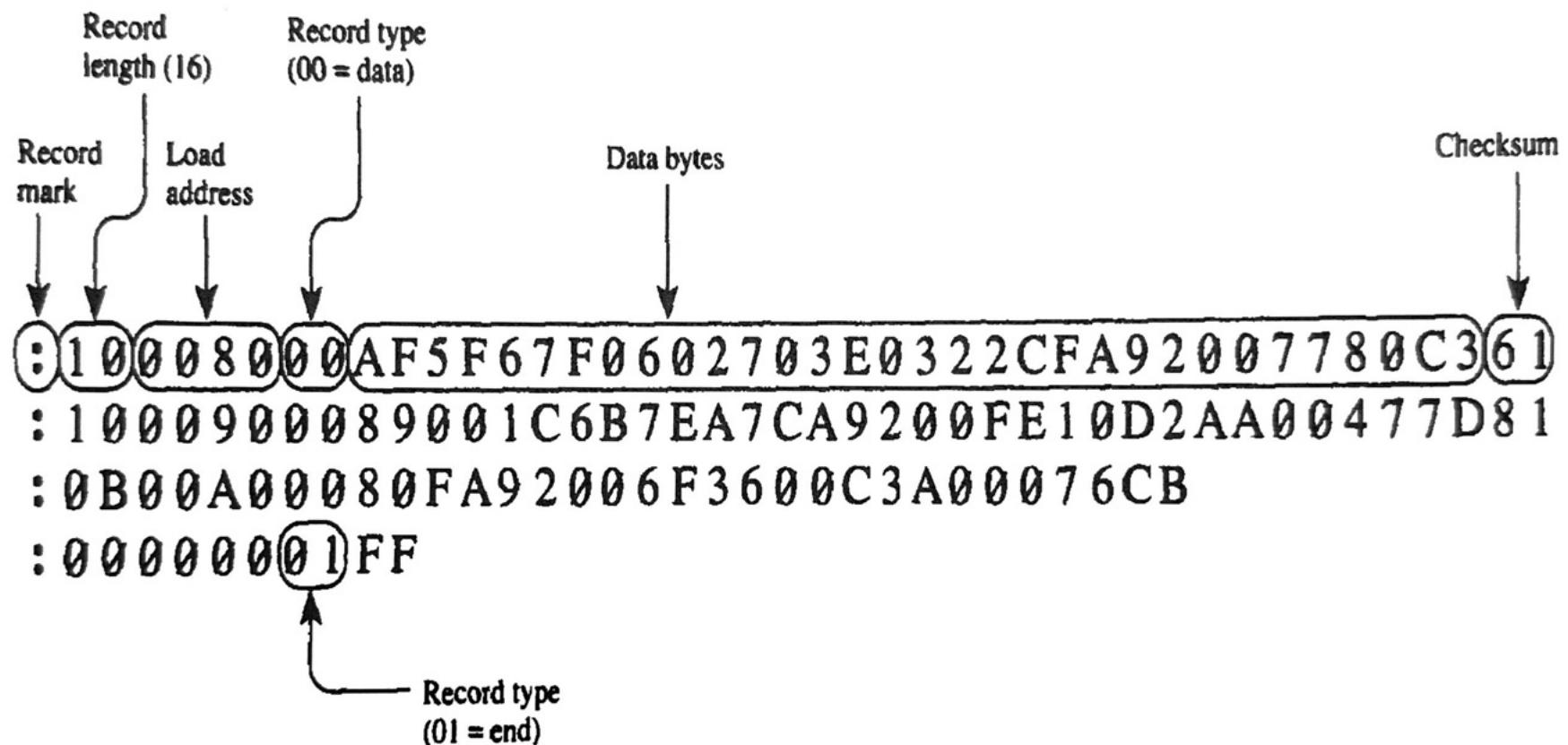
Code II

```
.ORG 8000h  
  
main:  
    mov P1, #5  
    ljmp main
```

Question 1: Suppose that we assembled both of these codes with RASM and downloaded them to R31JP using our normal MINMON with hyperterminal running on lab PCs. Which codes (I, II, or both) could be run using the approach of holding down the reset button, flipping the MON/RUN switch, and releasing the reset button?

Question 2: How would your answer change if, instead of using ljmp, we used sjmp in Code I and Code II?

INTEL HEX FILE:



Check out the “lst” file you get with: `as31 -l myfile.asm`


```

;=====
; subroutine init
; this routine initializes the hardware
;=====
init:
; set up serial port with a 11.0592 MHz crystal,
; use timer 1 for 9600 baud serial communications
    mov tmod, #20h      ; set timer 1 for auto reload - mode 2
    mov tcon, #41h      ; run timer 1 and set edge trig ints
    mov th1, #0fdh      ; set 9600 baud with xtal=11.059mhz
    mov scon, #50h      ; set serial control reg for 8 bit data
                        ; and mode 1
    ret
;=====
; monitor jump table
;=====
jumtab:
→ dw badcmd          ; command '@' 00      low at 3, high at 2
dw badcmd          ; command 'a' 01
dw badcmd          ; command 'b' 02
dw badcmd          ; command 'c' 03
dw badcmd          ; command 'd' 04 used
dw badcmd          ; command 'e' 05
dw badcmd          ; command 'f' 06
dw goaddr          ; command 'g' 07 used
dw badcmd          ; command 'h' 08
dw badcmd          ; command 'i' 09
dw badcmd          ; command 'j' 0a
dw badcmd          ; command 'k' 0b
dw badcmd          ; command 'l' 0c
dw badcmd          ; command 'm' 0d
dw badcmd          ; command 'n' 0e
dw badcmd          ; command 'o' 0f
dw badcmd          ; command 'p' 10
dw badcmd          ; command 'q' 11
dw badcmd          ; command 'r' 12
dw badcmd          ; command 's' 13
dw badcmd          ; command 't' 14
dw badcmd          ; command 'u' 15
dw badcmd          ; command 'v' 16
dw badcmd          ; command 'w' 17
dw badcmd          ; command 'x' 18
dw badcmd          ; command 'y' 19
dw badcmd          ; command 'z' 1a

```

db but →
 for
 2-byte
 objects
 (db is for)
 .1 byte

```

; monitor command routines
; goaddr 'g' - this routine branches to the 4 hex digit address which follows
;=====
goaddr:
    lcall getbyt          ; get address high byte
    mov    r7, a           ; save in R7
    lcall prthex
    lcall getbyt          ; get address low byte
    push   acc             ; push lsb of jump address
    lcall prthex
    lcall crlf
    mov    a, r7            ; recall address high byte
    push   acc             ; push msb of jump address
    ret                  ; do jump by doing a ret
;=====
; downld 'd' - this command reads in an Intel hex file from the serial port and stores it in external memory.
;=====
downld:
    lcall crlf
    mov    a, #'>'         ; acknowledge by a '>'
    lcall sndchr
dl:
    lcall getchr          ; read in ':'
    cjne  a, #':', dl
    lcall getbytx          ; get hex length byte
    jz    enddl            ; if length=0 then return
    mov    r0, a             ; save length in r0
    lcall getbytx          ; get msb of address
    setb  acc.7            ; make sure it is in RAM
    mov    dph, a             ; save in dph
    lcall getbytx          ; get lsb of address
    mov    dpl, a             ; save in dpl
    lcall getbytx          ; read in special purpose byte (ignore)
dloop:
    lcall getbytx          ; read in data byte
    movx  @dptr, a           ; save in ext mem
    inc   dptr              ; bump mem pointer
    djnz  r0, dloop          ; repeat for all data bytes in record
    lcall getbytx          ; read in checksum
    mov    a, '#.'
    lcall sndchr            ; handshake '.'
    sjmp  dl                ; read in next record
enddl:
    lcall getbytx          ; read in remainder of the
    lcall getbytx            ; termination record
    lcall getbytx
    lcall getbytx
    mov    a, '#.'
    lcall sndchr            ; handshake '..'
    ljmp  endloop            ; return
getbytx:
    lcall getbyt
    jb    errorf, gb_err
    ret
gb_err:
    ljmp  badpar

```

```

;***** monitor support routines *****
;***** badcmd:
badcmd:
    lcall print
    db 0dh, 0ah, " bad command ", 0h
    ljmp endloop
;***** badpar:
badpar:
    lcall print
    db 0dh, 0ah, " bad parameter ", 0h
    ljmp endloop
;===== subroutine getbyt
; this routine reads in an 2 digit ascii hex number from the
; serial port. the result is returned in the acc.
;=====
getbyt:
    lcall getchr      ; get msb ascii chr
    lcall ascbin      ; conv it to binary
    swap a           ; move to most sig half of acc
    mov b, a         ; save in b
    lcall getchr      ; get lsb ascii chr
    lcall ascbin      ; conv it to binary
    orl a, b        ; combine two halves
    ret
;===== subroutine getcmd
; this routine gets the command line. currently only a
; single-letter command is read - all command line parameters
; must be parsed by the individual routines.
;
;=====
getcmd:
    lcall getchr      ; get the single-letter command
    clr acc.5        ; make UPPER case
    lcall sndchr      ; echo command
    clr C            ; clear the carry flag
    subb a, #'@'      ; convert to command number
    jnc cmdok1       ; letter command must be above '@'
    lcall badpar
cmdok1:
    push acc          ; save command number
    subb a, #1Bh      ; command number must be 1Ah or less
    jc cmdok2         ; no need to pop acc since badpar
    lcall badpar      ; initializes the system
cmdok2:
    pop acc           ; recall command number
    ret

```

they're offset by 32
 checks if negative
 via carry flag
 wants
 between
 @ & beyond
 bound of Z
 uppercase

```

;=====
; subroutine nway
; this routine branches (jumps) to the appropriate monitor
; routine. the routine number is in r2
;=====

nway:
    mov    dptr, #jumtab      ;point dptr at beginning of jump table
    mov    a, r2                ;load acc with monitor routine number
    rl     a                  ;multiply by two.
    inc    a                  ;load first vector onto stack
    movc   a, @a+dptr        ;           "           "
    push   acc                ;           "           "
    mov    a, r2                ;load acc with monitor routine number
    rl     a                  ;multiply by two
    movc   a, @a+dptr        ;load second vector onto stack
    push   acc                ;           "           "
    ret                 ;jump to start of monitor routine

    ← load pointer
    of a █ subroutine
    rotate-left

;=====
; general purpose routines
;=====

; subroutine sndchr
; this routine takes the chr in the acc and sends it out the
; serial port.
;=====

sndchr:
    clr    scon.1            ; clear the tx complete flag
    mov    sbuf,a             ; put chr in sbuf
txloop:
    jnb    scon.1, txloop    ; wait till chr is sent
    ret

;=====
; subroutine getchr
; this routine reads in a chr from the serial port and saves it
; in the accumulator.
;=====

getchr:
    jnb    ri, getchr        ; wait till character received
    mov    a, sbuf             ; get character
    anl    a, #7fh            ; mask off 8th bit
    clr    ri                 ; clear serial status bit
    ret

```

```

=====
; subroutine print
; print takes the string immediately following the call and
; sends it out the serial port. the string must be terminated
; with a null. this routine will ret to the instruction
; immediately following the string.
=====
print:
    pop dph          ; put return address in dptr
    pop dpl          ← little endian, high byte first
    prtstr:
        clr a          ; set offset = 0
        movc a, @a+dptr ; get chr from code memory
        cjne a, #0h, mchrok ; if termination chr, then return
        sjmp prtdone
    mchrok:
        lcall sndchr      ; send character
        inc dptr          ; point at next character
        sjmp prtstr       ; loop till end of string
    prtdone:
        mov a, #1h          ; point to instruction after string
        jmp @a+dptr        ; return
=====
; subroutine crlf
; crlf sends a carriage return line feed out the serial port
=====
crlf:
    mov a, #0ah         ; print lf
    lcall sndchr
    cret:
    mov a, #0dh         ; print cr
    lcall sndchr
    ret
=====
; subroutine prthex
; this routine takes the contents of the acc and prints it out
; as a 2 digit ascii hex number.
=====
prthex:
    push acc
    lcall binasc        ; convert acc to ascii
    lcall sndchr        ; print first ascii hex digit
    mov a, r2            ; get second ascii hex digit
    lcall sndchr        ; print it
    pop acc
    ret

```

$$dptr = \underline{dph} \quad \underline{dpl}$$

works
w1 movx ←
too, program
and exit.
memory are
all the game

← doesn't use
lcall to escape

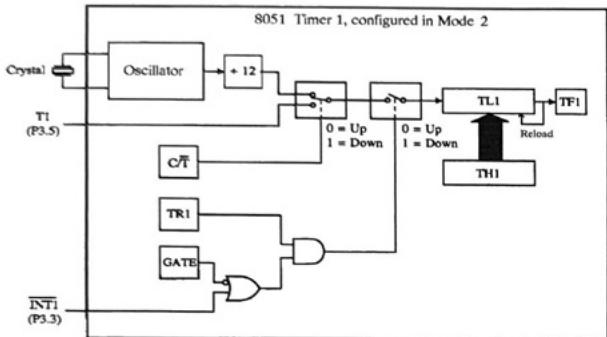
```

;=====
; subroutine binasc
; binasc takes the contents of the accumulator and converts it
; into two ascii hex numbers.  the result is returned in the
; accumulator and r2.
;=====
binasc:
    mov    r2, a          ; save in r2
    anl    a, #0fh         ; convert least sig digit.
    add    a, #0f6h        ; adjust it
    jnc    noadj1          ; if a-f then readjust
    add    a, #07h
noadj1:
    add    a, #3ah         ; make ascii
    xch    a, r2           ; put result in reg 2
    swap   a               ; convert most sig digit
    anl    a, #0fh         ; look at least sig half of acc
    add    a, #0f6h        ; adjust it
    jnc    noadj2          ; if a-f then re-adjust
    add    a, #07h
noadj2:
    add    a, #3ah         ; make ascii
    ret
;=====
; subroutine ascbin
; this routine takes the ascii character passed to it in the
; acc and converts it to a 4 bit binary number which is returned
; in the acc.
;=====
ascbin:
    clr    errorf
    add    a, #0d0h        ; if chr < 30 then error
    jnc    notnum
    clr    c               ; check if chr is 0-9
    add    a, #0f6h        ; adjust it
    jc     hextry          ; jmp if chr not 0-9
    add    a, #0ah          ; if it is then adjust it
    ret
hextry:
    clr    acc.5           ; convert to upper
    clr    c               ; check if chr is a-f
    add    a, #0f9h        ; adjust it
    jnc    notnum          ; if not a-f then error
    clr    c               ; see if char is 46 or less.
    add    a, #0faf          ; adjust acc
    jc     notnum          ; if carry then not hex
    anl    a, #0fh          ; clear unused bits
    ret
notnum:
    setb   errorf          ; if not a valid digit
    ljmp   endloop
;=====
; mon_return is not a subroutine.  It simply jumps to address 0 which resets the
; system and invokes the monitor program.
;=====
mon_return:
    ljmp   0 ; end of MINMON

```

How can we make some precisely timed square waves using R31JP?

Class happy quiz: think about some short programs that make fast square waves. Today: How might we use timers to help? Our familiar tool, a timer in auto-reload mode. Very helpful for our serial port work.

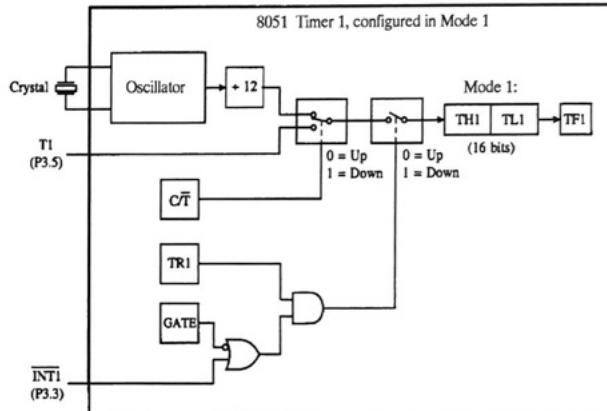
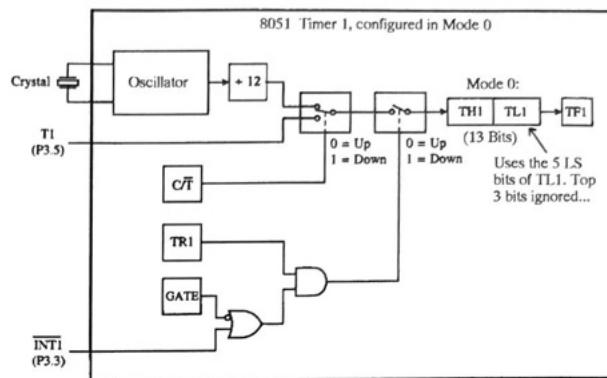


Here's a program for a 10kHz square wave on the P1.0: (assume a 12MHz crystal)

```
MOV TMOD, #02H
MOV TH0, #0Ceh ; 50 counts
SETB TR0
LOOP:
    JNB TF0, LOOP
    CLR TF0
    CPL P1.0
    SJMP LOOP
```

Will this give a frequency of exactly 10kHz?

1



2

TMOD Register

Timer/Counter 1				Timer/Counter 0			
GATE#	C/T#	M1	M0	GATE#	C/T#	M1	M0

M0 and M1 (bits 0 and 1 of TMOD, respectively): Mode Select

The 2-bit field (M1,M0) selects one of four modes. Mode 0 is a 13-bit counter. An interrupt is generated when the counter overflows. Thus it takes 2^{13} or 8192 input pulses to generate the next interrupt. Mode 1, similar to Mode 0, implements a 16-bit counter. It takes 2^{16} or 65,536 input pulses to generate the next interrupt. Mode 2 operates in an 8-bit reload fashion. TL1 serves as an 8-bit timer/counter. When the counter overflows, the number stored in TH1 is copied into TL1 and the count continues. An interrupt is generated each time the counter overflows and a reload is performed. In Mode 3, Timer 1 is inactive and simply holds its count. Timer 0 operates as two separate 8-bit timers. TL0 is controlled by Timer 0 control bits and generates a Timer 0 interrupt at overflow. TH0 operates as a timer driven by the system clock, prescaled by 12, and causes a Timer 1 interrupt at overflow.

C/T# (bit 2): Counter/Timer Select

When set, the timer/counter operates as a counter. The count increment is caused by external pulses through the T0 pin—the alternative function of P3.4. When cleared the timer/counter operates as a timer. The count increment is caused by every 12th system clock pulse. That is, the input to the counter is the system clock prescaled by 12.

GATE# (bit 3): Gate

Provided that TR0 of TCON (see next page) is set, clearing GATE enables (starts) counter/timer operation. If GATE is set, again provided that TR0 is set, the timer/counter operation is enabled if INT0# is high.

TCON Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
-----	-----	-----	-----	-----	-----	-----	-----

IT0 (bit 0): Interrupt 0 Type

External interrupt 0 is received through bit 2 of Port 3 as an alternative function assignment. Setting IT0 causes the INT0# to be recognized at the falling edge of the input signal. Clearing IT0 causes an interrupt to be generated when the external signal is low. If the external signal stays low, INT0 will be generated over and over if IT0 is cleared but will not be generated if IT0 is set. IT0 is completely under software control.

IE0 (bit 1): Interrupt 0 Edge Flag

Set by hardware when an external interrupt edge is detected. Cleared when a Return From Interrupt (RETI) instruction is executed.

IT1 (bit 2): Interrupt 1 Type

IT1 is associated with External Interrupt 1. Its function is similar to that of IT0 as described above.

IE1 (bit 3): Interrupt 1 Edge Flag

IE1 is associated with External Interrupt 1. Its function is similar to that of IE0 as described above.

TR0 (bit 4): Timer 0 Run Control Bit

Timer/Counter 0 is disabled when TR0 is cleared. Setting TR0 is necessary but not sufficient to enable Timer/Counter 0 (see GATE# and INT0#). TR0 is completely under software control.

TF0 (bit 5): Timer 0 Overflow Flag

TF0 is set by hardware when Timer/Counter 0 overflows. TF0 is cleared by hardware when the processor branches to the associated interrupt service routine. TF0, along with IE0, may be used by software to determine the state of the timer.

TR1 (bit 6): Timer 1 Run Control Bit

TR1 is associated with Timer/Counter 1. Its function is similar to that of TR0 described above.

TF1 (bit 7): Timer 0 Overflow Flag

TF1 is associated with Timer/Counter 1. Its function is similar to that of TF0 described above.

Techniques for programming timed intervals:
(assume 12Mhz clock)

TECHNIQUE	INTERVAL in microsecs
Software tuning (no timers)	~10
8 bit auto-reload timer	256
16 bit timer	65,536
8 or 16 bit timer with software loops	Very long

How about a 1kHz square wave on P1.0? 500usec half period, longer than the 256 usec available with mode 2.

Try mode 1 (16 bit timer): (No autoreload, so timing is imperfect...)

```
MOV TMOD #01h
LOOP:
    MOV TH0, #0Feh ; -500 high byte
    MOV TL0, #0Ch   ; -500 low byte
    SETB TR0
    WAIT:
        JNB TF0, WAIT
        CLR TR0
        CLR TF0
        CPL P1.0
        SJMP LOOP
```

5

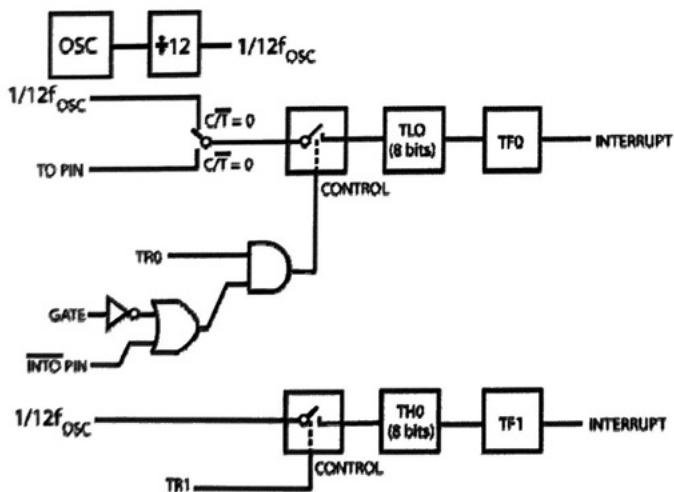
D8h and F0h are
the high and low
bytes for -10000

```
loop:
    cpl P1.0
    lcall delay
    sjmp loop

delay:
    mov R7, #100
again:
    mov TH0, #0D8h
    mov TL0, #0F0h
    setb TR0
    wait2: jnb TF0, wait2
    clr TF0
    clr TR0
    djnz R7, again
ret
```

6

And, finally, here's MODE 3 (for timer 0):



A brief introduction to interrupts - another way to a 10kHz square wave:

```
.org 0
ljmp MAIN
.org 000Bh
T0ISR:
    cpl P1.0
    reti
.ORG 0030h
MAIN:
    mov TMOD, #02h
    mov TH0, #0Ceh ;50 counts
    setb TR0
    mov IE, #82h
loop: sjmp loop
```

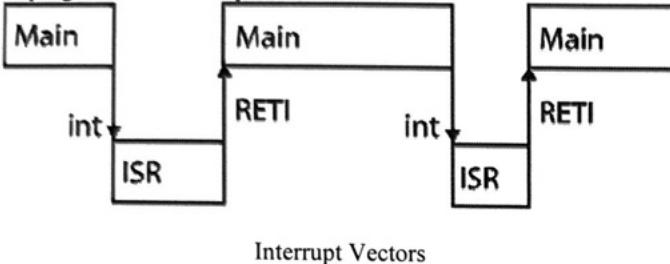
7

8

A program without interrupts:



A program with interrupts:



INTERRUPT	FLAG	VECTOR ADDR.
RESET	RST	0000h
External 0	IE0, TCON.1	0003h
Timer 0	TF0, TCON.5	000Bh
External 1	IE1, TCON.3	0013h
Timer 1	TF1, TCON.7	001Bh
Serial Port	R1 (SCON.0) or T1 (SCON.1)	0023h
Timer 2 (8052)	TF@ or EXF2	002Bh

1

2

Transmit the graphic (non-control) ascii characters:

```
.ORG 0 ; 12 Mhz!!           .ORG 200h
LJMP MAIN                   SPISR:
                            CJNE A, #7Fh, skip
ORG 023h                   MOV A, #20h
LJMP SPISR                 SKIP:
                            MOV SBUF, A
.ORG 030h                   INC A
MAIN:                      CLR TI
                            RETI
MOV TMOD, #20h
MOV TH1, #0E6h
SETB TR1
MOV SCON, #42h
MOV A, #20h
MOV IE, #90h
LOOP:
SJMP LOOP
```

Again, assuming a 12Mhz crystal,
and a setup for 1200 baud serial
transmission.)

Controller for a furnace:

```
.ORG 0
LJMP MAIN
EX0ISR:
    CLR P1.7
    RETI
.ORG 013H
EX1ISR:
    SETB P1.7
    RETI
```

HOT# → INT0# → 8051 → P1.7 → Furnace on

COLD# → INT1# → 8051

3

A brief introduction to interrupts – another way to a 10kHz square wave:

```
.org 0
ljmp MAIN
.org 000Bh
T0ISR:
    cpl P1.0
    reti
.ORG 0030h
MAIN:
    mov TMOD, #02h
    mov TH0, #0Ceh ;50 counts
    setb TR0
    mov IE, #82h
    loop: sjmp loop
```

Two Square Waves! 7kHz on P1.7 and 500Hz on P1.6 (approx).

```
.ORG 0
LJMP MAIN
ORG 0Bh
LJMP T0ISR
ORG 1Bh
LJMP T1ISR
.ORG 030h
MAIN:
    MOV TMOD, #12h
    MOV TH0, #0B9h
    SETB TR0
    SETB TF1
    MOV IE, #8Ah
    LOOP:
        SJMP LOOP
```

```
.ORG 200h
T0ISR:
    CPL P1.7
    RETI
T1ISR:
    CLR TR1
    MOV TH1, #0FCh
    MOV TL1, #18h
    SETB TR1
    CPL P1.6
    RETI
```

(FC18 is hex for 1000, i.e., the
T1ISR counts out 1ms.)
ASSUME 12MHz crystal!!!

Door buzzer: door opens, sounds alarm 400Hz tone for 1 sec.

```
.ORG 0
EX0ISR:
    MOV R7, #20
LJMP MAIN
LJMP EX0ISR
ORG 00Bh:t0, 1sec interval
LJMP T0ISR
.ORG 01Bh:t0,400hz interval
LJMP T1ISR
.ORG 0030h
MAIN:
    SETB ITO
    MOV TMOD #11H
    MOV IE, #81h
    LOOP:
        SJMP LOOP
```

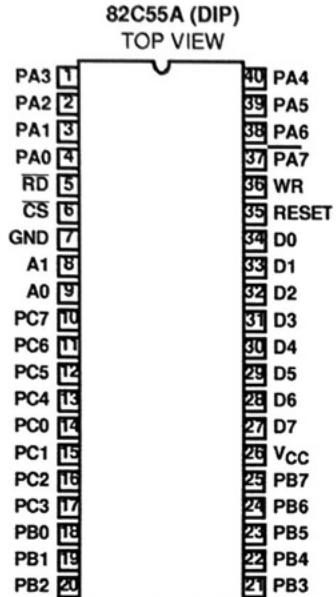
```
TOISR:          T1ISR:
    CLR TR0    CLR TR1
    DJNZ R7, SKIP    MOV TH1, #0FBh
    CLR ET0    MOV TL1, #01eh
    CLR ET1    CPL P1.7
    LJMP EXIT    SETB TR1
SKIP:          RETI
;3cb0h counts 0.05s
    MOV TH0, #3Ch
    MOV TLO, #0B0h
    SETB TR0
EXIT:
    RETI
```

4

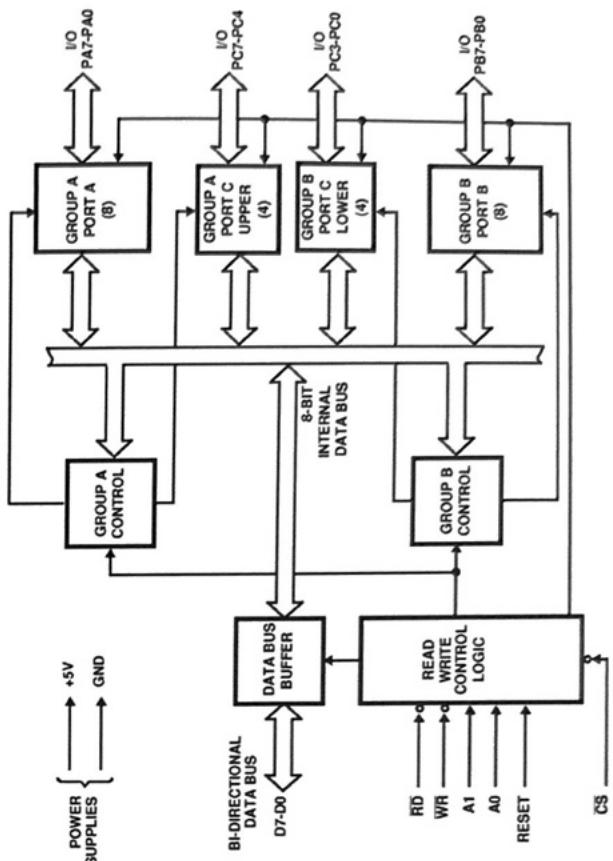
In Lab 2, you worked with the 8254 counter/timer, which expanded the capabilities of the R31JP system. The 8254 is one member in the "8000" series of peripheral chips made by Intel. We'll look at some of these and other types of peripherals, today.



82C55A CHMOS PROGRAMMABLE PERIPHERAL INTERFACE

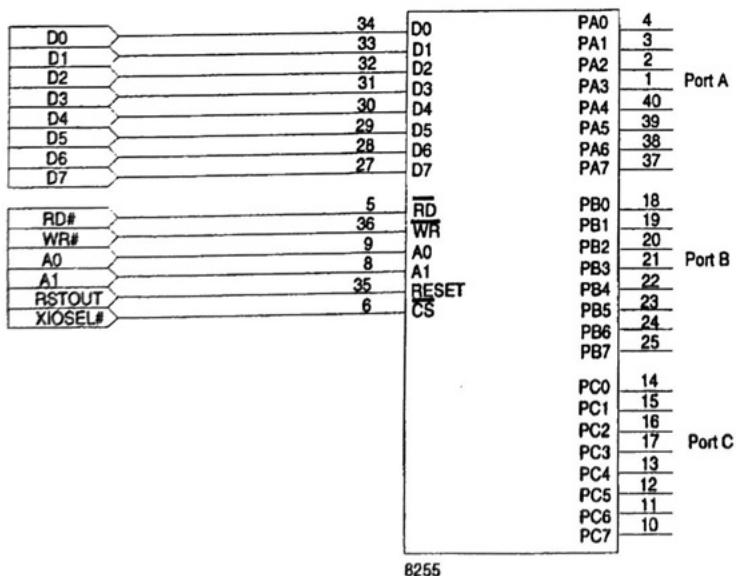


1

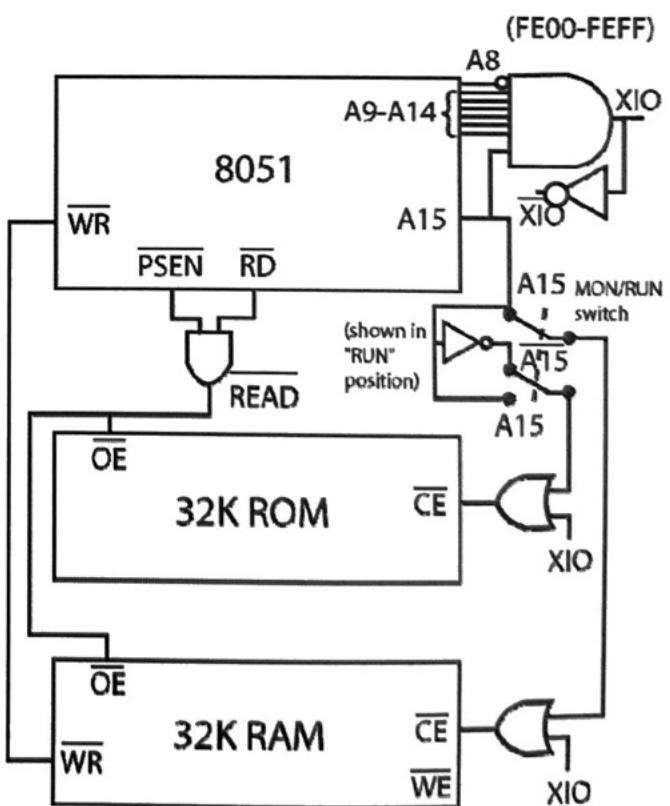


2

Here's the 8255 connection to the R31JP shown in your R31JP manual:



3

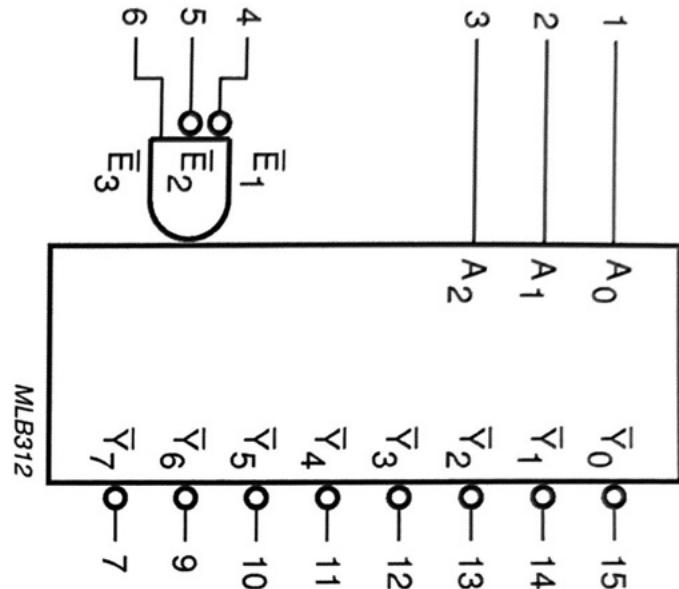


4

Table I. AD558 Control Logic Truth Table

Input Data	\overline{CE}	\overline{CS}	DAC Data	Latch Condition
0	0	0	0	"Transparent"
1	0	0	1	"Transparent"
0	g	0	0	Latching
1	g	0	1	Latching
0	0	g	0	Latching
1	0	g	1	Latching
X	1	X	Previous Data	Latched
X	X	1	Previous Data	Latched

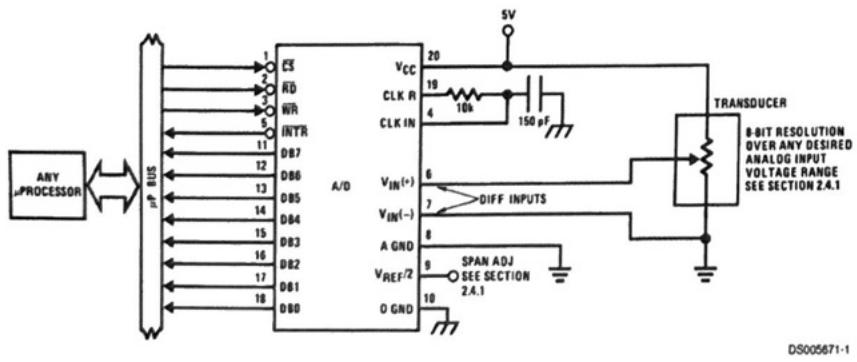
74C138



5



ADC0801/ADC0802/ADC0803/ADC0804/ADC0805 8-Bit μ P Compatible A/D Converters



8

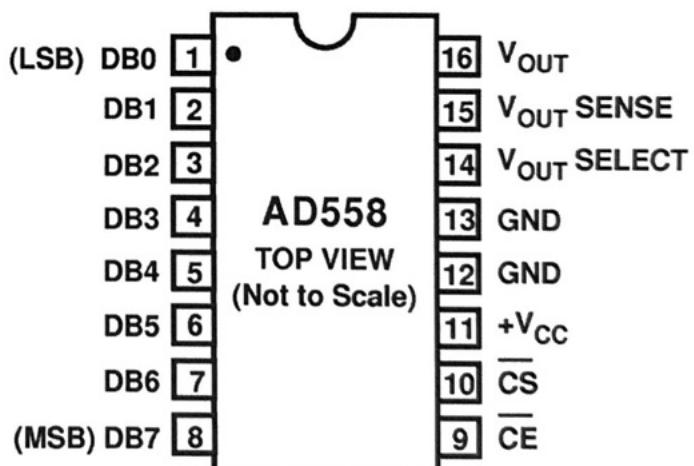


Figure 1a. AD558 Pin Configuration (DIP)

6

8254-Style Frequency Calculation:

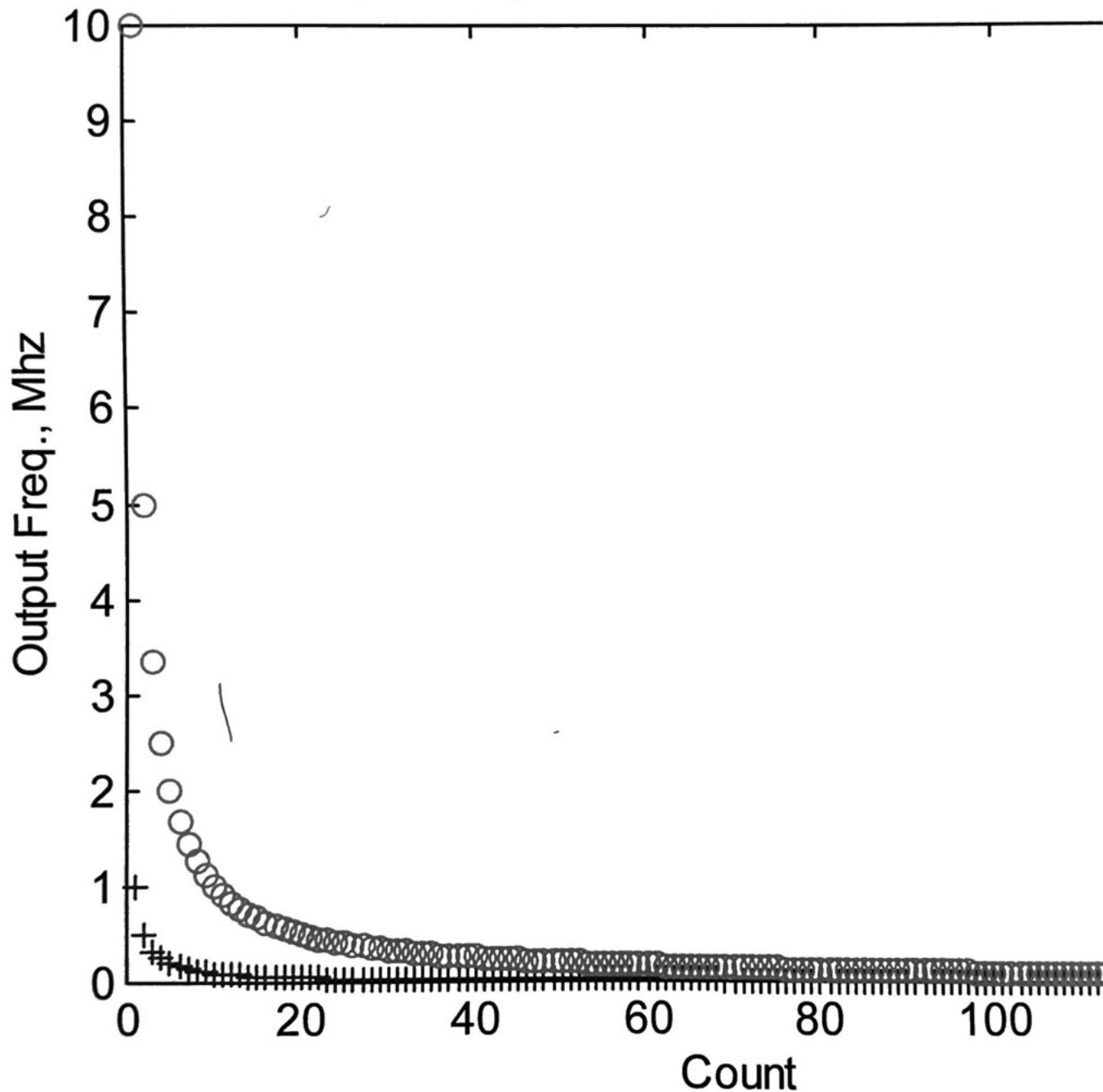
$$\text{Output Frequency} = \frac{\text{Clock Frequency}}{\text{Counts}}$$

$$\text{Maximum Output Frequency} = \frac{\text{Clock Frequency}}{\text{Minimum Count}}$$

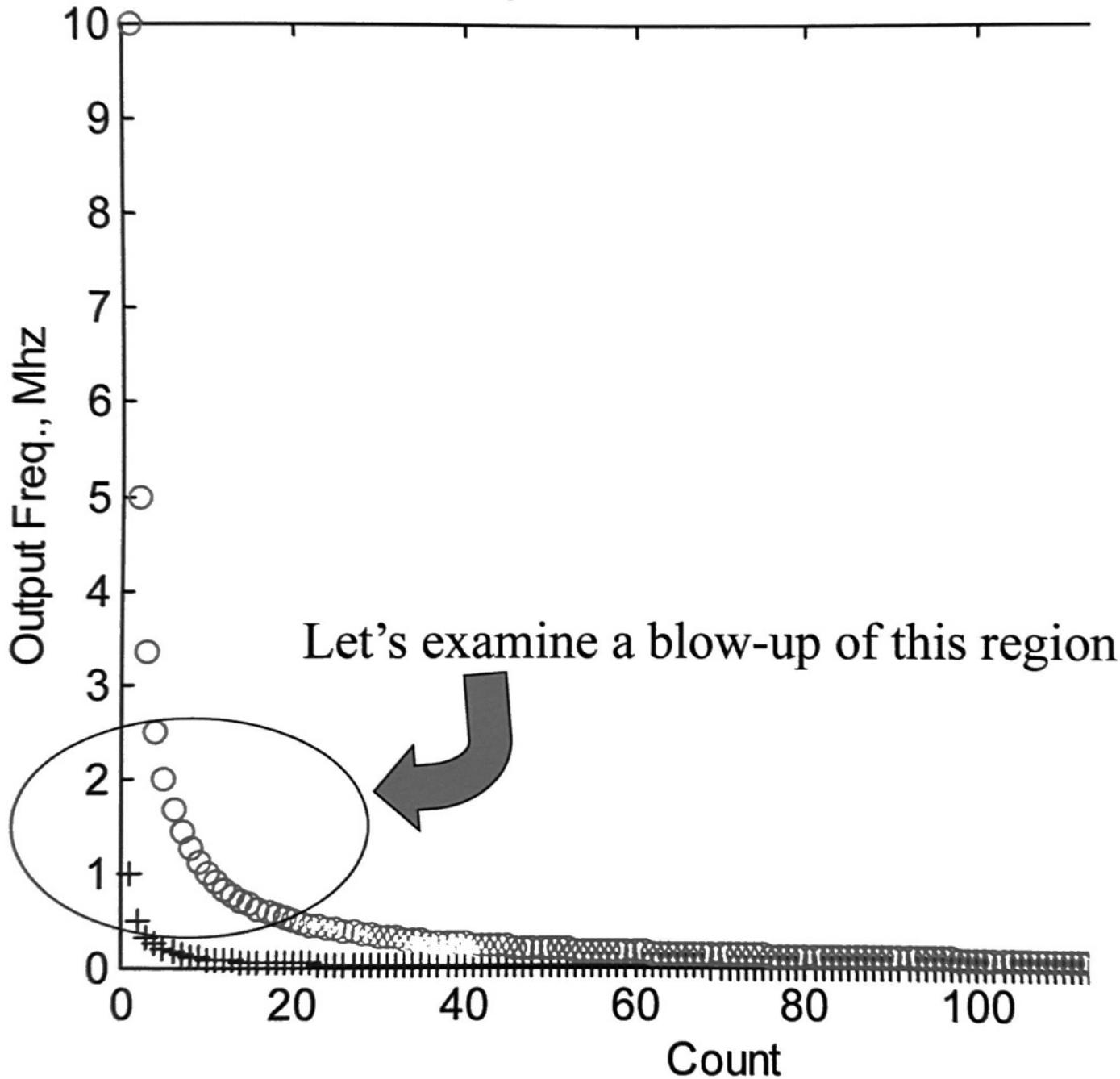
↑
Count RANGE
↓

$$\text{Minimum Output Frequency} = \frac{\text{Clock Frequency}}{\text{Maximum Count}}$$

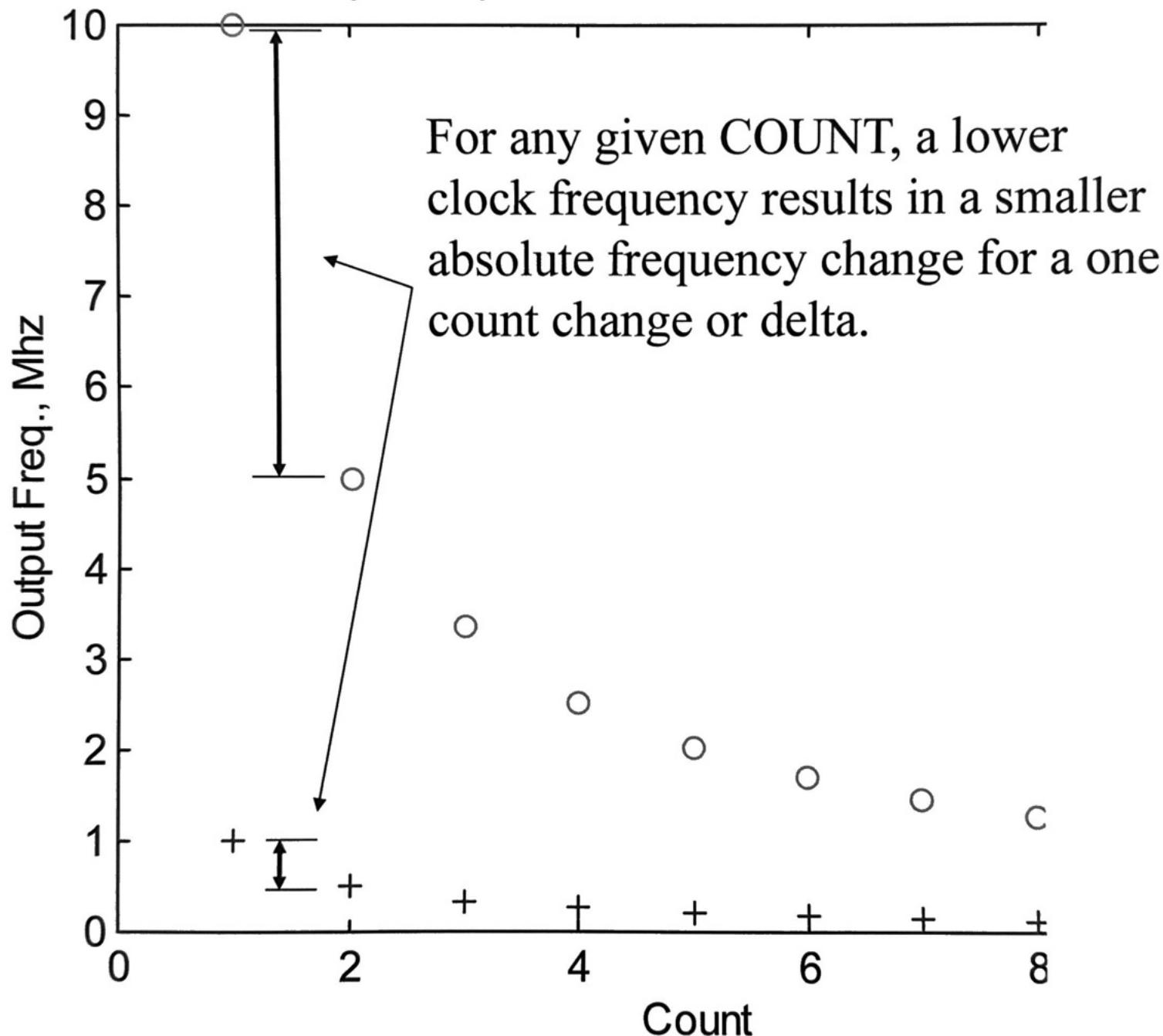
Output Frequencies, 10 Mhz and 1 Mhz Cl

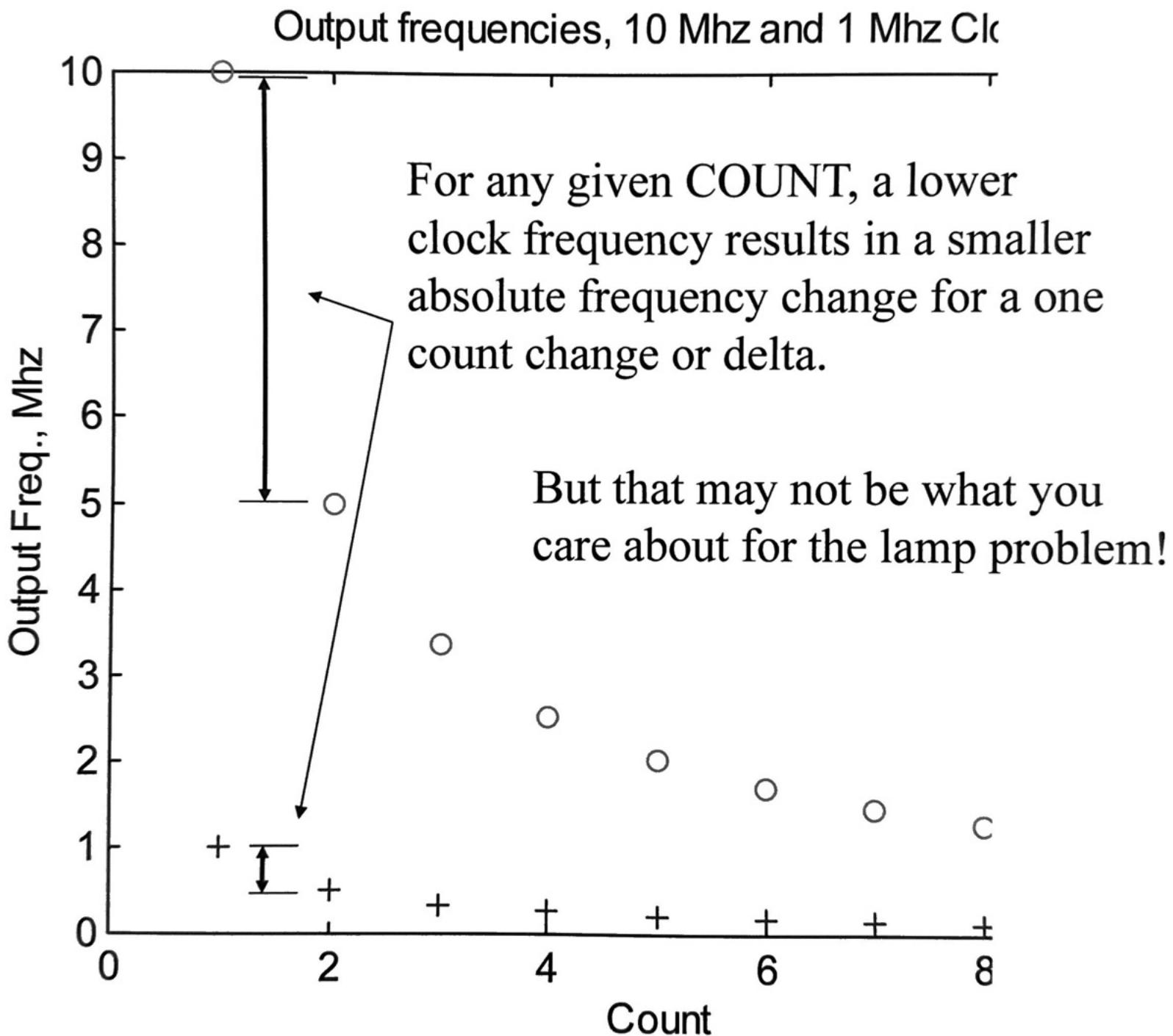


Output Frequencies, 10 Mhz and 1 Mhz CI

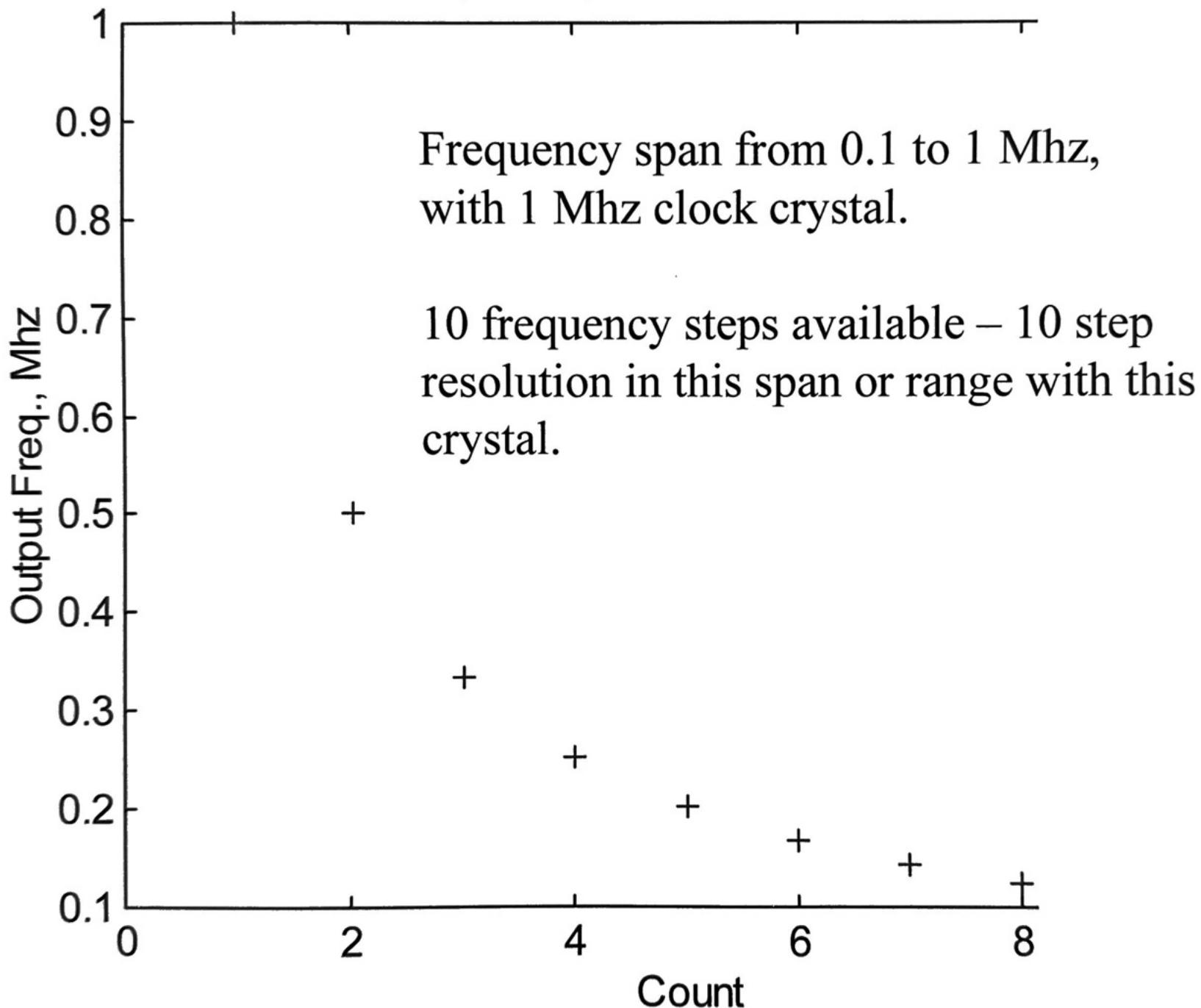


Output frequencies, 10 Mhz and 1 Mhz Clk

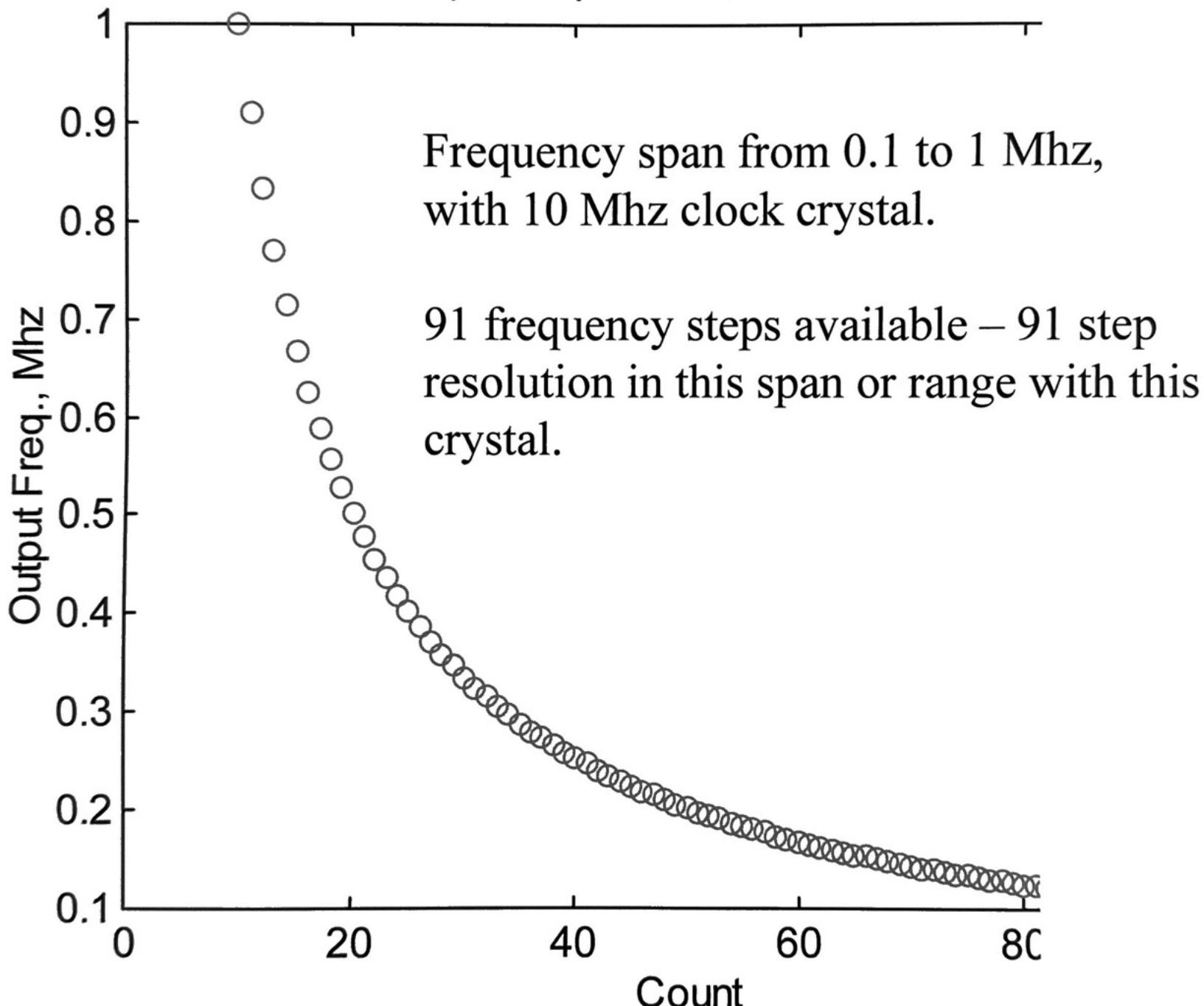




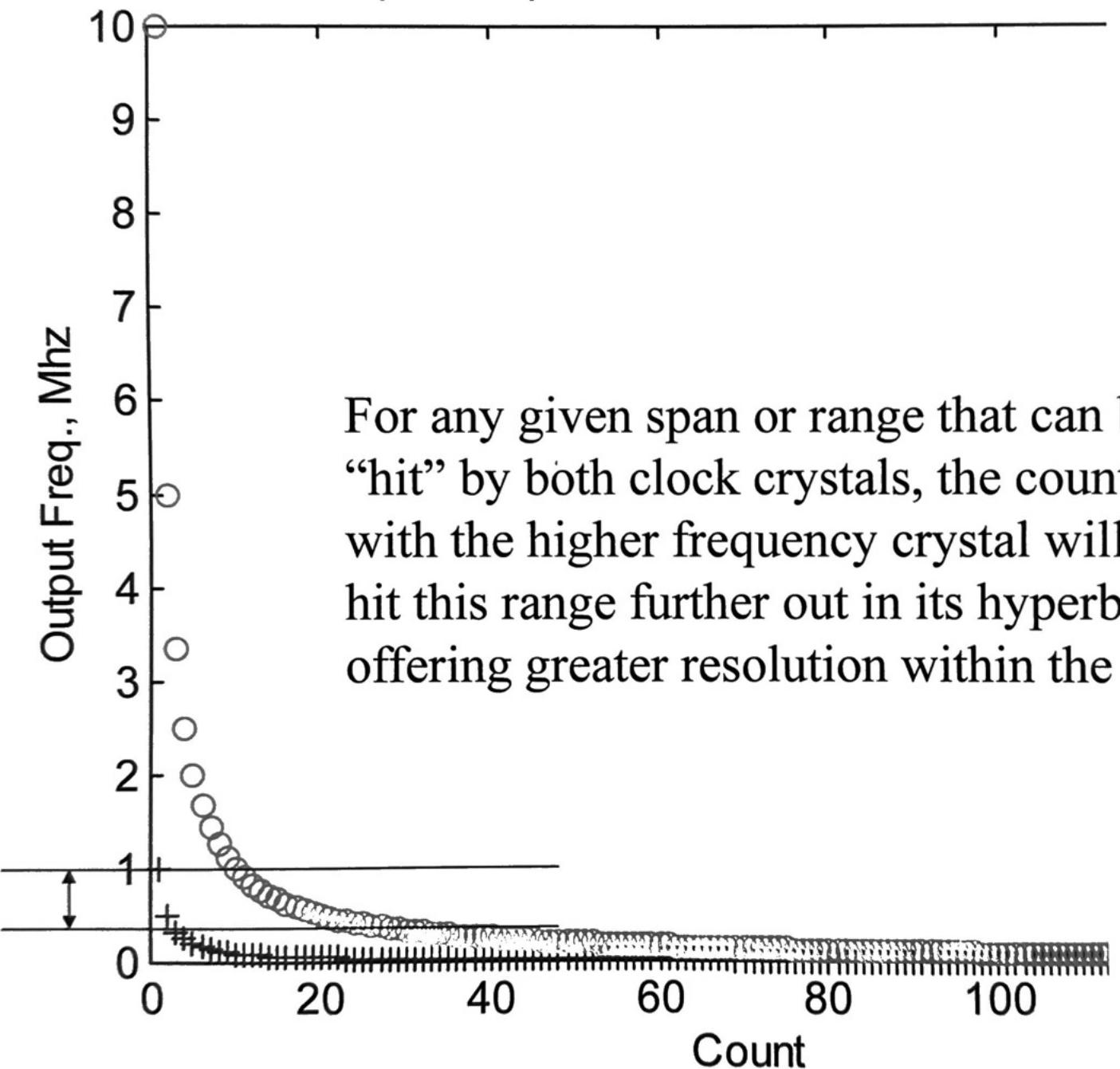
Output frequencies, 1 Mhz Clock



Output frequencies, 10 Mhz Clock



Output Frequencies, 10 Mhz and 1 Mhz Cl





DACPORT Low Cost, Complete μP-Compatible 8-Bit DAC

AD558*

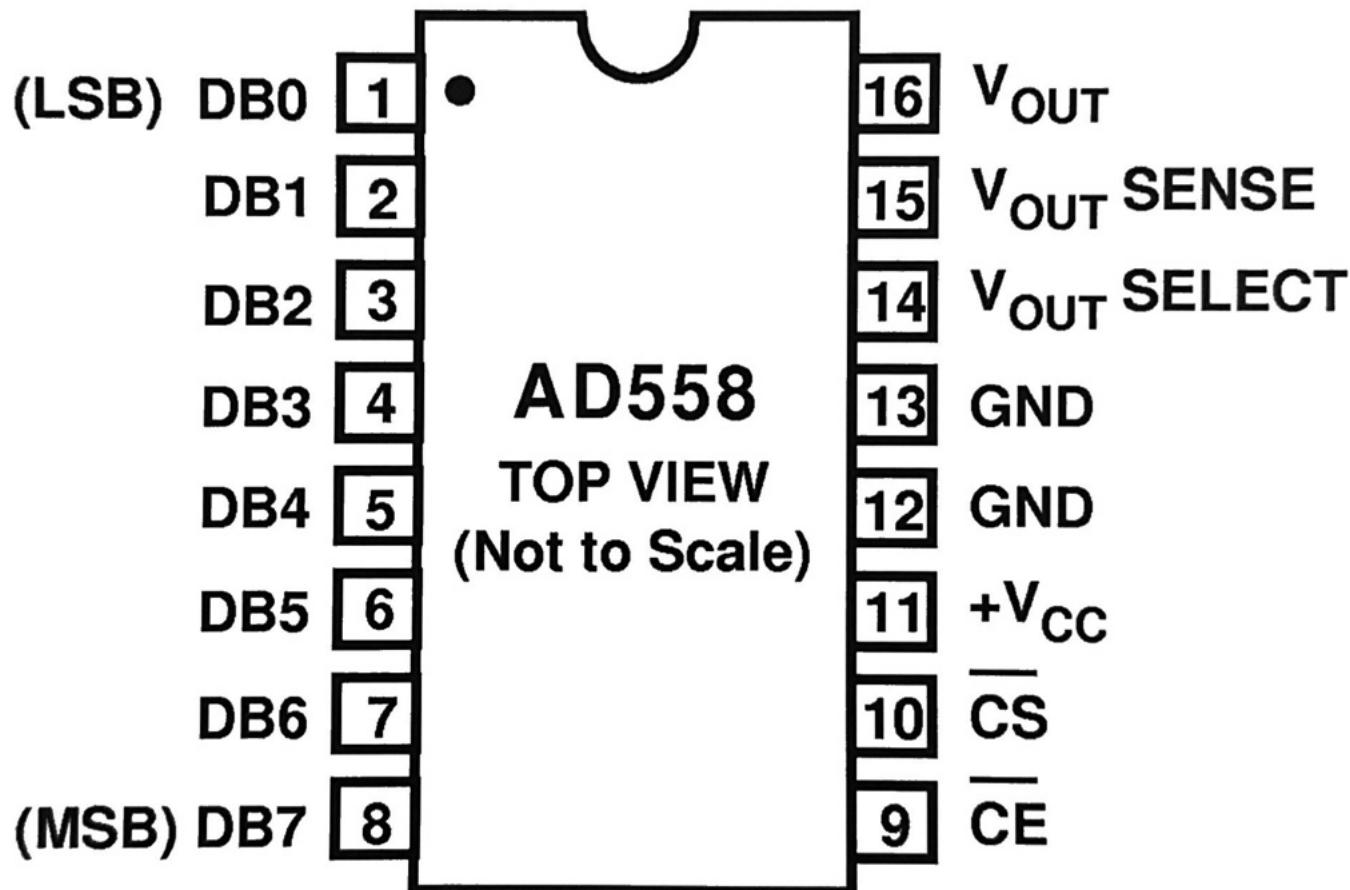


Figure 1a. AD558 Pin Configuration (DIP)

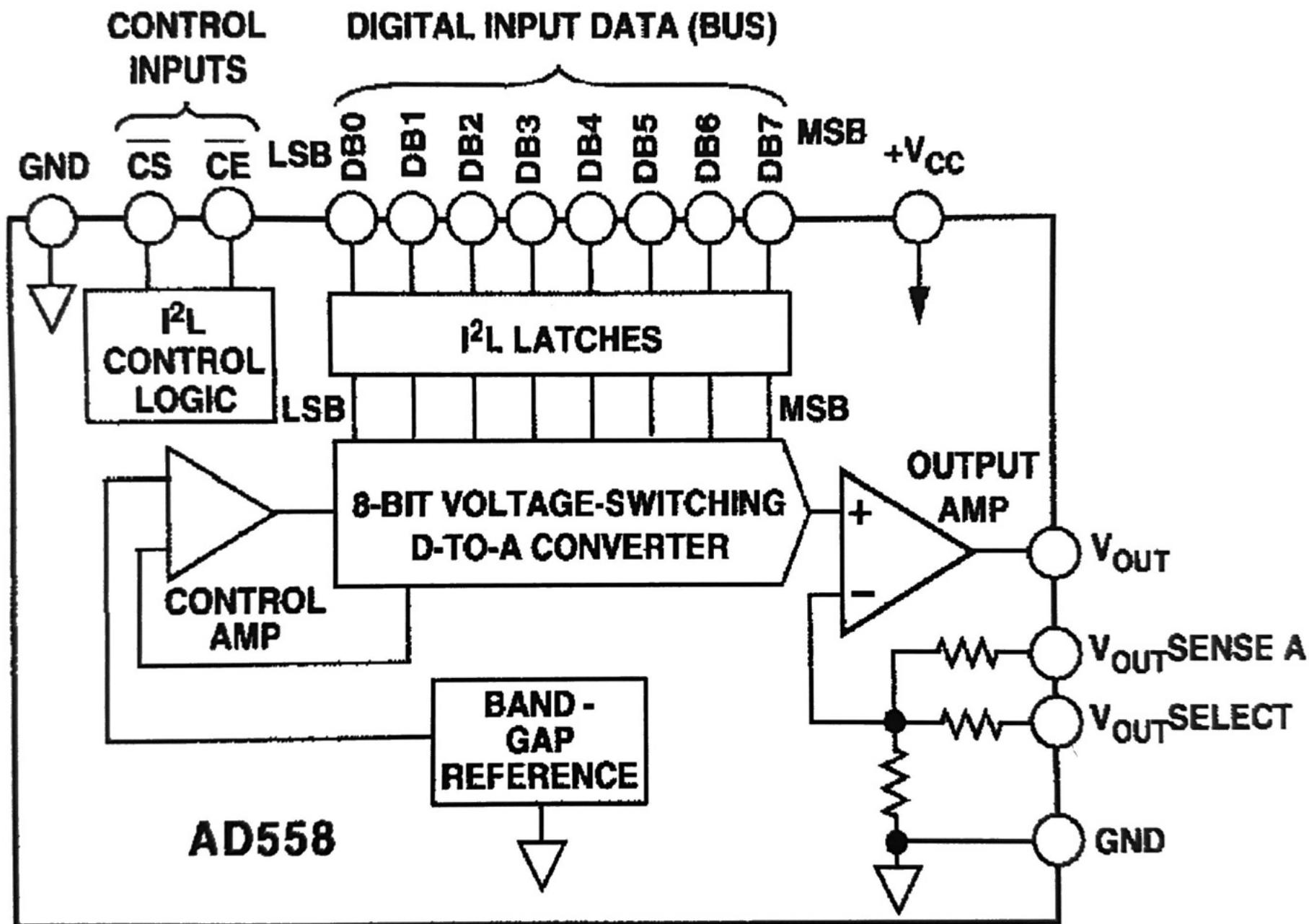


Table I. AD558 Control Logic Truth Table

Input Data	\overline{CE}	\overline{CS}	DAC Data	Latch Condition
0	0	0	0	“Transparent”
1	0	0	1	“Transparent”
0	g	0	0	Latching
1	g	0	1	Latching
0	0	g	0	Latching
1	0	g	1	Latching
X	1	X	Previous Data	Latched
X	X	1	Previous Data	Latched

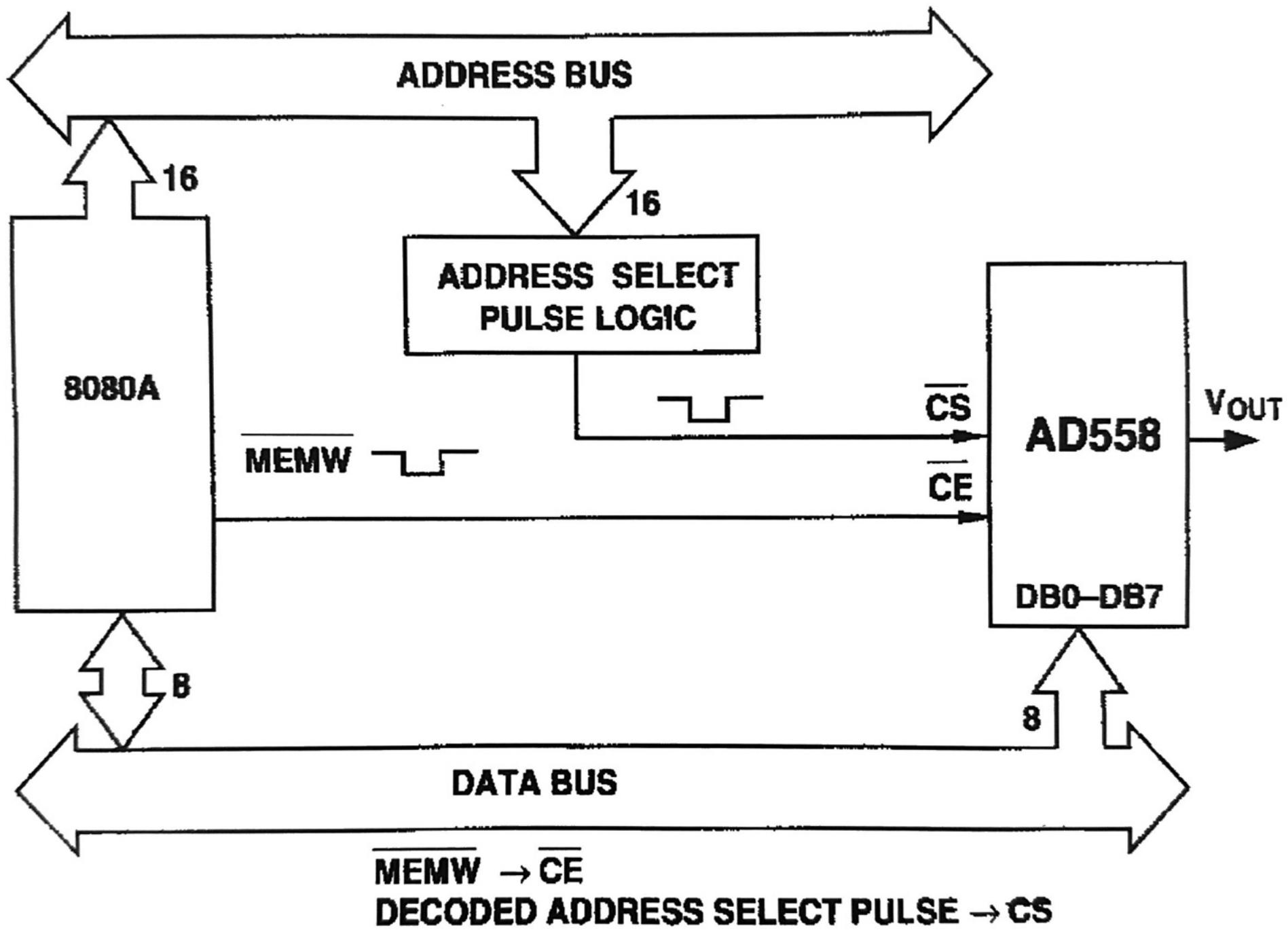


Table Entry Number:

	0	1	2	3	4	5	6	7
Analog Sinewave:	0	0.707	1	0.707	0	-0.707	-1	-0.707
Offset Analog:	1	1.707	2	1.707	1	0.293	0	0.293
Times 255/2, converted to hex...	7Fh	DAh	FFh	DAh	7Fh	25h	00h	25h

- Eight (8) entries in the sinewave data table
- Limit timer interrupts to an advance every 1/8th second
- Determine the counts/advance of the table counter:
- First, try a 3-bit table counter to make a 1 Hz sinewave:

$$2^3 \frac{\text{counts}}{\text{period}} \bullet 1 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 1 \frac{\text{count}}{\text{advance}}$$

- Eight (8) entries in the sinewave data table
- Limit timer interrupts to an advance every 1/8th second
- Determine the counts/advance of the table counter:
- First, try a 3-bit table counter to make a 1 Hz sinewave:

$$2^3 \frac{\text{counts}}{\text{period}} \bullet 1 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 1 \frac{\text{count}}{\text{advance}}$$

- Now, try a 3-bit table counter to make a **1.5 Hz** sinewave:

$$2^3 \frac{\text{counts}}{\text{period}} \bullet 1.5 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 1 \frac{\text{count}}{\text{advance}}$$

(with integer rounding)

- Eight (8) entries in the sinewave data table
- Limit timer interrupts to an advance every 1/8th second
- Determine the counts/advance of the table counter:
- First, try a 3-bit table counter to make a 1 Hz sinewave:

$$2^3 \frac{\text{counts}}{\text{period}} \bullet 1 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 1 \frac{\text{count}}{\text{advance}}$$

- Now, try a 3-bit table counter to make a **1.5 Hz** sinewave:

$$2^3 \frac{\text{counts}}{\text{period}} \bullet 1.5 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 1 \frac{\text{count}}{\text{advance}}$$

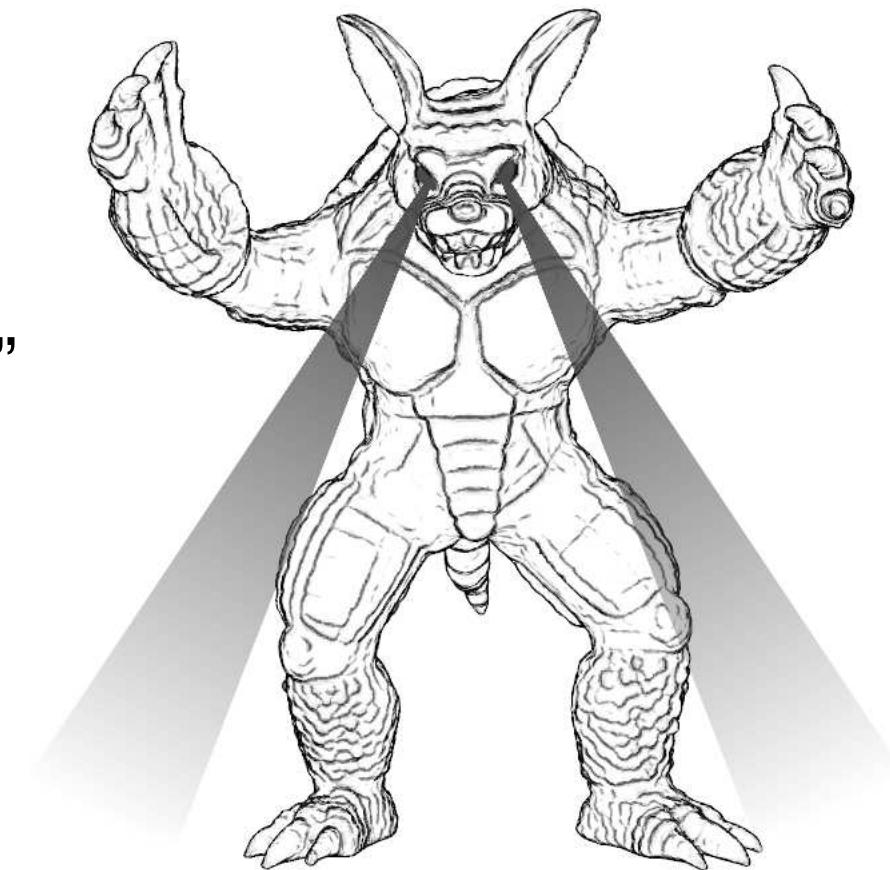
(with integer rounding)

- Now, try a **4-bit** table counter to make a **1.5 Hz** sinewave:

$$2^4 \frac{\text{counts}}{\text{period}} \bullet 1.5 \frac{\text{period}}{\text{second}} \bullet \frac{1}{8} \frac{\text{seconds}}{\text{advance}} = 3 \frac{\text{count}}{\text{advance}}$$

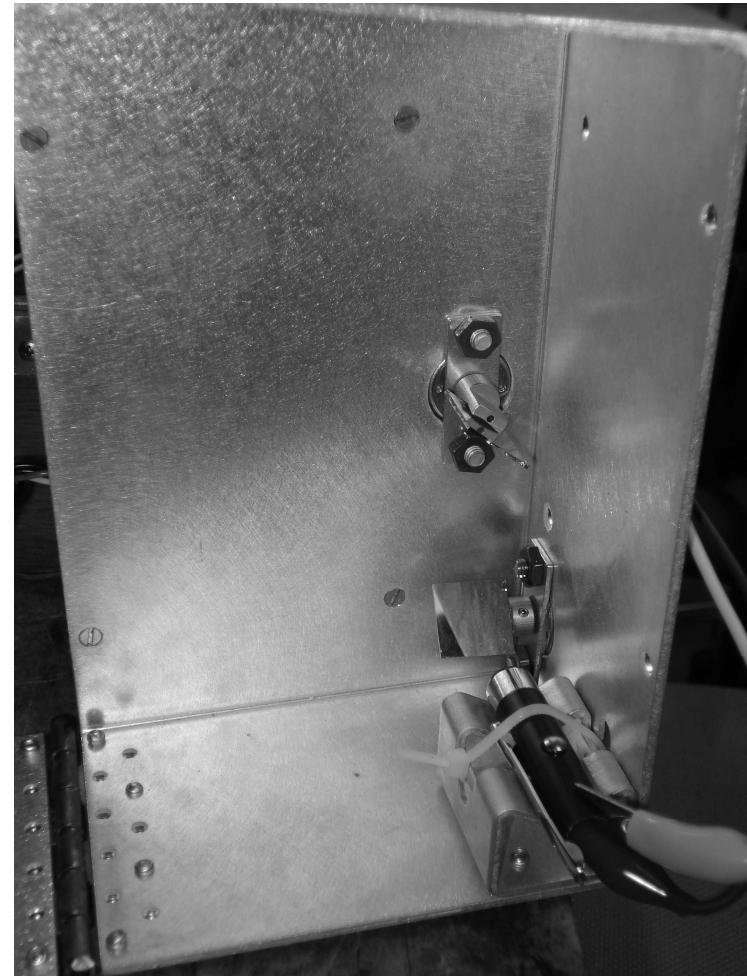
Lazerdillo

“To Project and Serve...”



The Lazerdillo

- Simple laser projector
- R31JP interface
- x, y position controls
- on/off toggle for laser

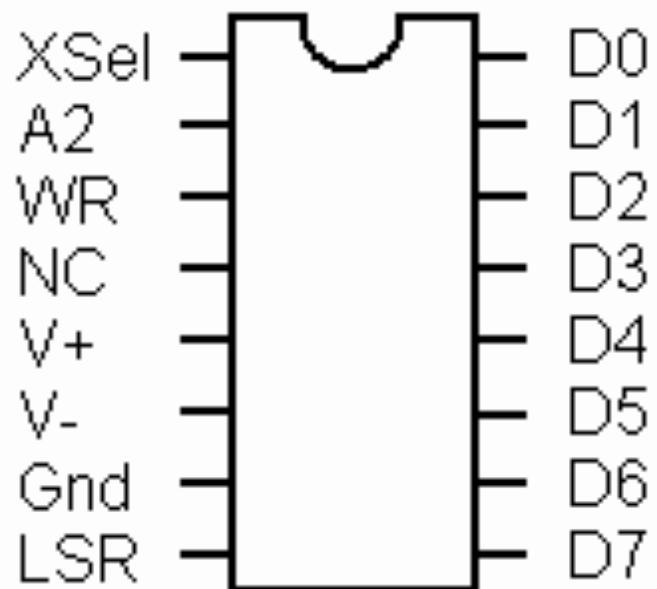


Operation

- Two steppers control the X and Y position
- X and Y DACs are 8 bits
- Buffered voltage signals control stepper motors

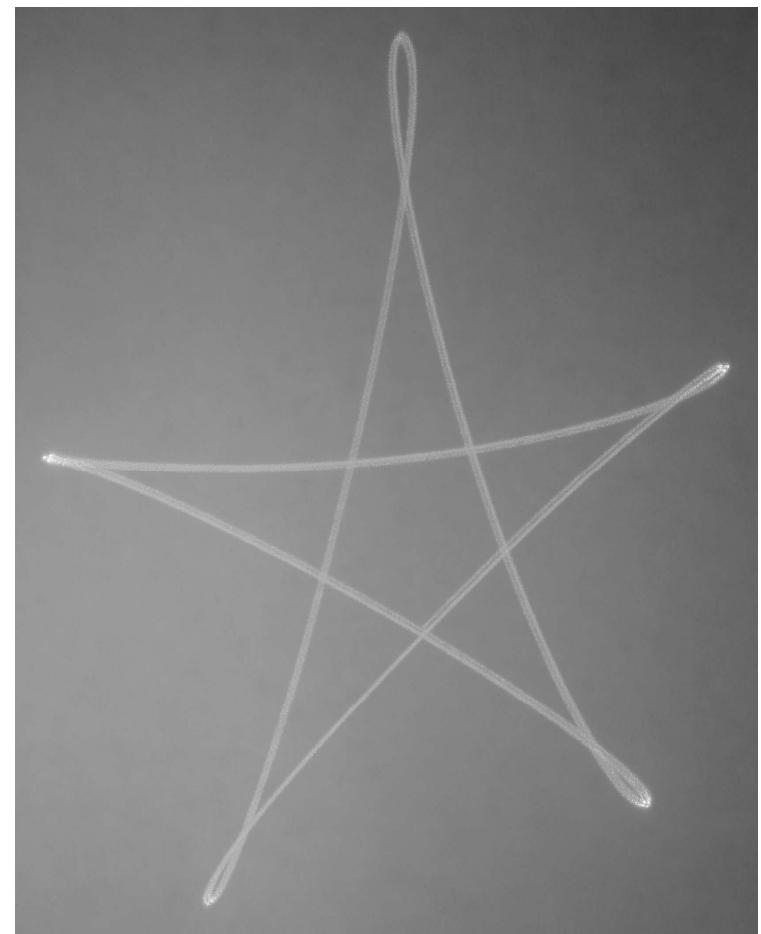
Pin-Out

- A “microcontroller-compatible” peripheral
- A2 selects X or Y DACs
- XSel selects the Lazerdillo
- WR triggers the write to the selected DAC
- LSR triggers the laser



Drawing Objects with the Lazerdillo

Figure generated by sending the laser dot through a sequence of (x,y) points.



Inertial Limitation

- Bandwidth limited.
- Lazerdillo can handle sinusoidal signals up to about 80 Hz.

Optical Limitation

- Lazerdillo works within and around the response and blending limitations of the eye
- Movie theaters run at 24 frames per second
- Fusion fails below 20 frames per second

Linear Limitation

- 20 degree range of motion in x and y axes
- Voltage/Position is not linear near the edges of its range
- Rough linearity if position bytes kept between #40h and #C0h

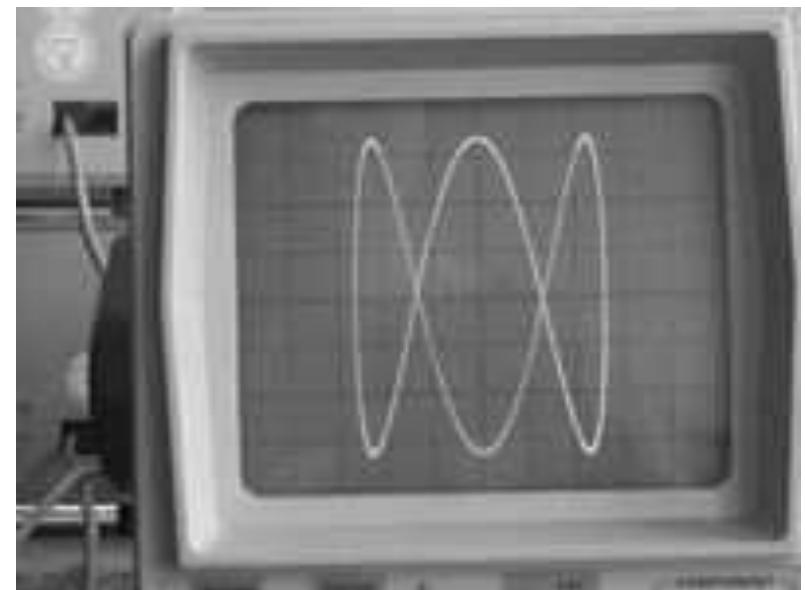
Lissajous Figures

- Use Lazerdillo to generate Lissajous figures

- Defined by:

$$x = A \sin 2\pi a t$$

$$y = B \sin 2\pi (bt + \phi)$$



Interpretation of Variables

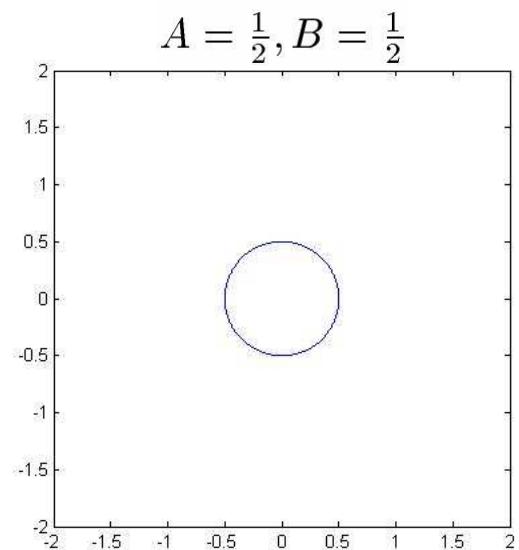
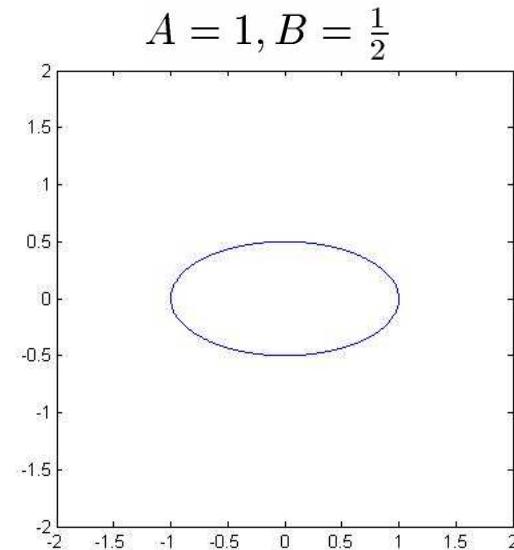
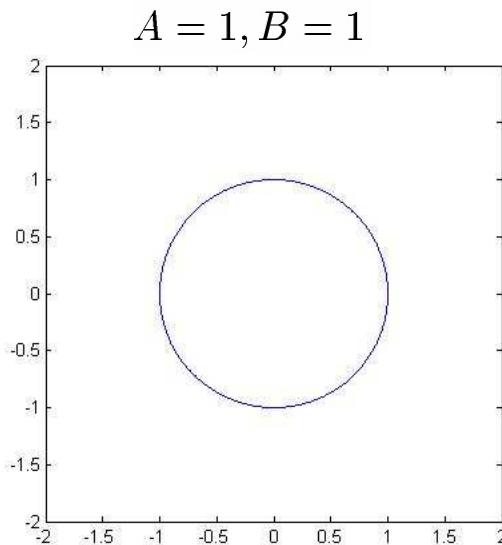
$$x = A \sin 2\pi a t \quad y = B \sin 2\pi (bt + \phi)$$

- A and B represent amplitude
- Variable ϕ represents phase difference
- a and b represent oscillation frequencies

Amplitude

$$x = A \sin 2\pi at \quad y = B \sin 2\pi(bt + \phi)$$

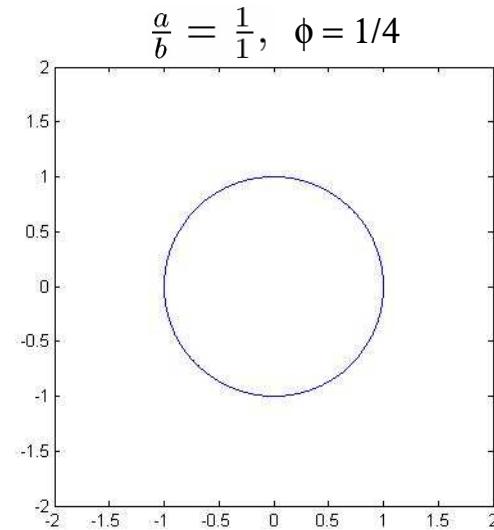
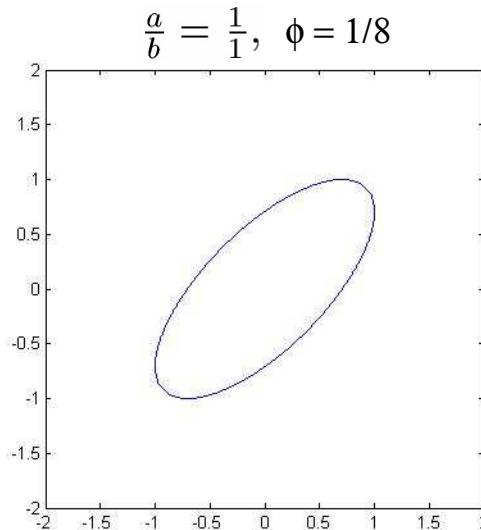
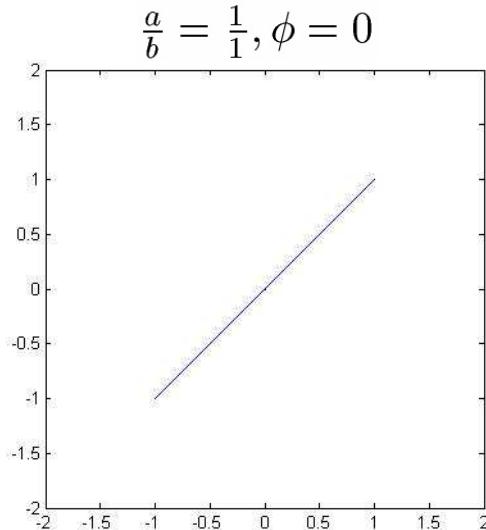
- The effect of varying just amplitude variables A and B :



Phase

$$x = A \sin 2\pi at \quad y = B \sin 2\pi(bt + \phi)$$

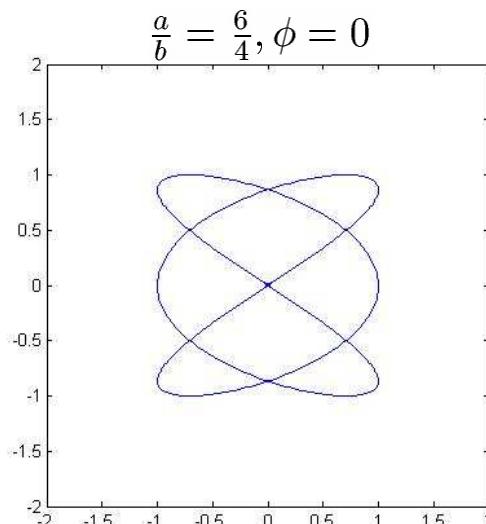
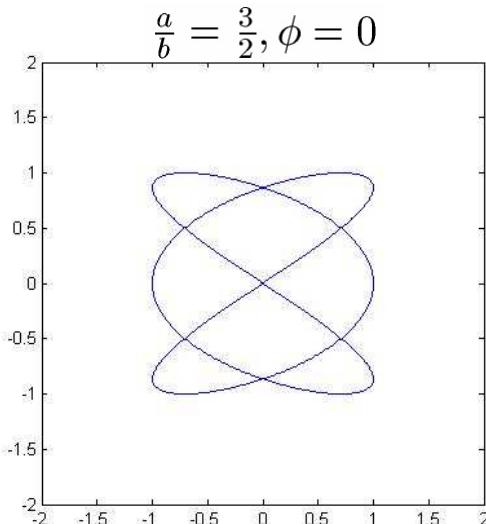
- The phase angle ϕ influences the orientation and the shape of the figure
- Consider the figures below generated with $a/b=1$ and different values of ϕ



Frequency

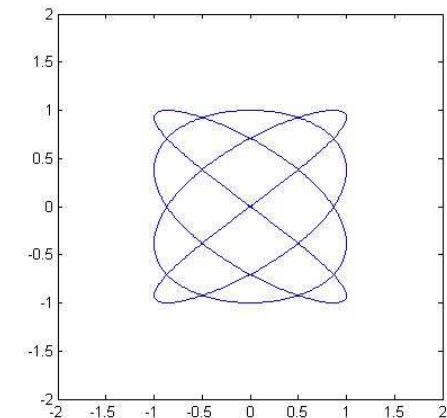
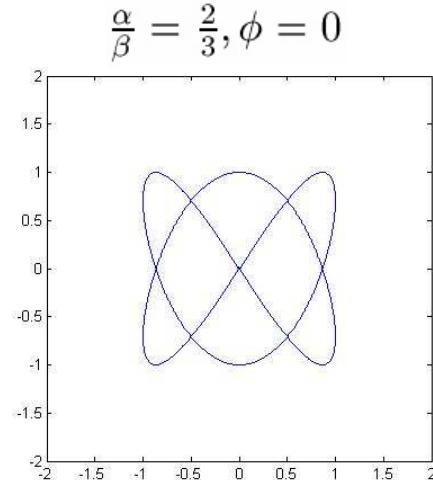
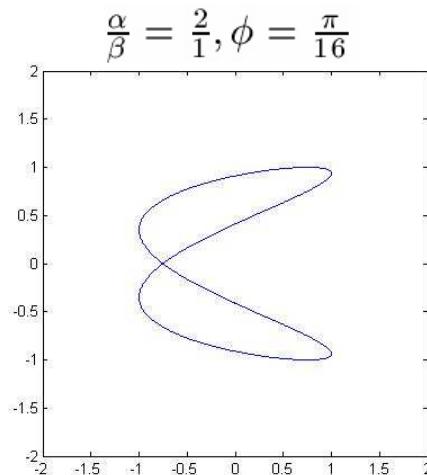
$$x = A \sin 2\pi a t \quad y = B \sin 2\pi (bt + \phi)$$

- Frequencies a and b control oscillation independently
- The ratio of frequencies a/b defines the structure
- We define a/b to be the absolute frequency ratio, and a/β to be the frequency ratio in simplest terms

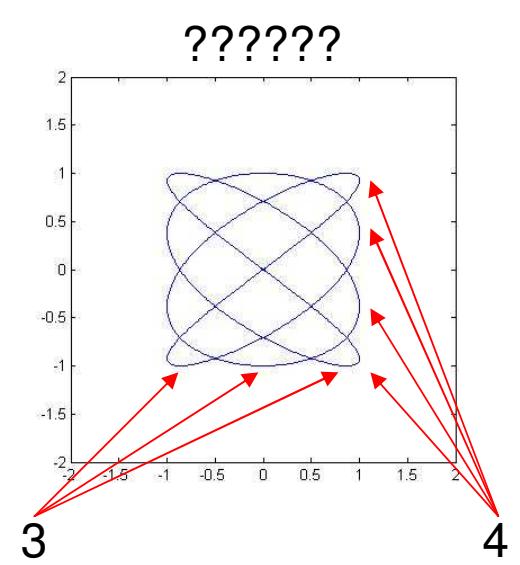
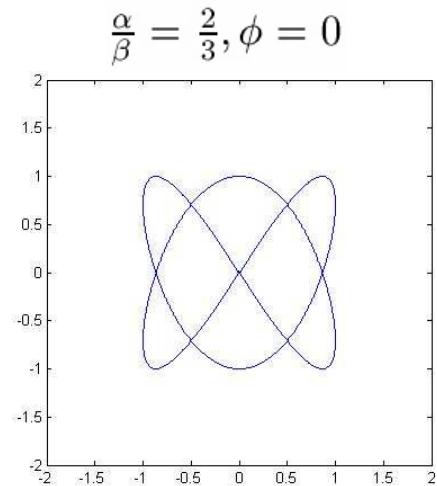
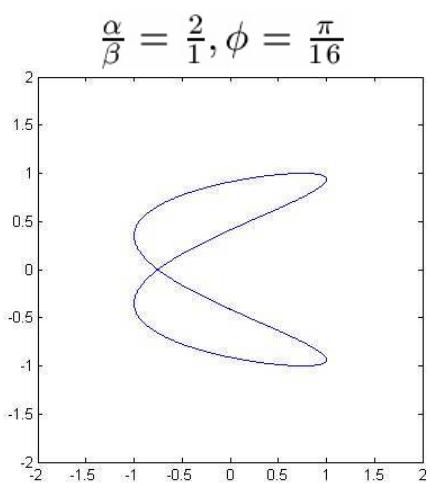


Frequency Ratio

- α and β : oscillations about their respective axes per frame
- α/β : ratio of oscillations about the x axis versus y axis
- Frequency ratio also sets maxima along the x axis versus y
- The frequency ratio indicates the complexity of the figure

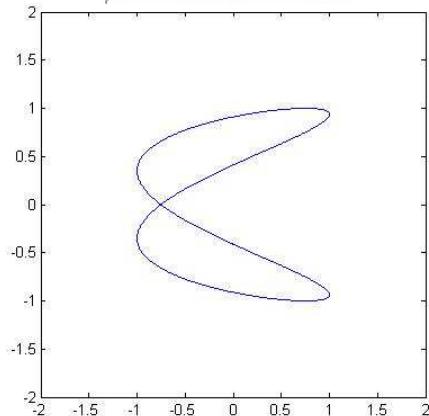


Determine mystery ratio:

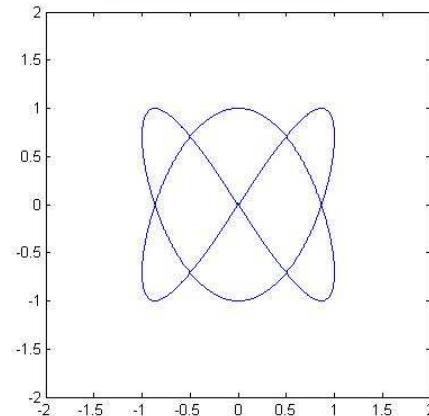


Determine mystery ratio:

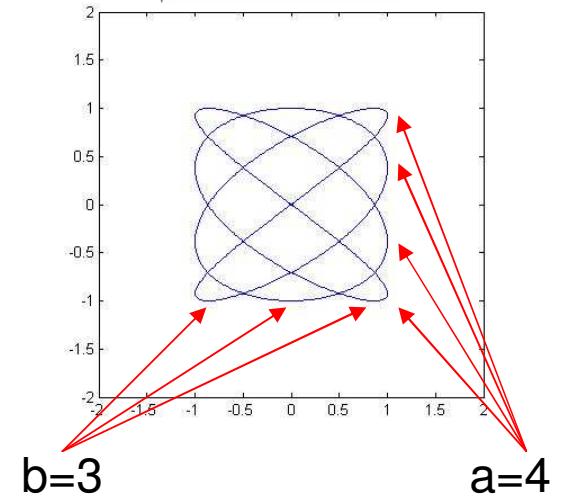
$$\frac{\alpha}{\beta} = \frac{2}{1}, \phi = \frac{\pi}{16}$$



$$\frac{\alpha}{\beta} = \frac{2}{3}, \phi = 0$$



$$\frac{\alpha}{\beta} = \frac{4}{3}, \phi = 0$$



Period

$$x = A \sin 2\pi a t \quad y = B \sin 2\pi (bt + \phi)$$

- Consider $\phi = 0$.
- Frequency ratio does not fully constrain the picture
- Ratio 2/2 will be drawn in half the time as 1/1
- Period T: time it takes to return to the starting point

Calculating the Period

- Period:

$$T = \frac{\alpha}{a} = \frac{\beta}{b}$$

- Examining units:

$$\frac{\alpha \frac{cycles}{frame}}{a \frac{cycles}{second}} = T \frac{seconds}{frame}$$

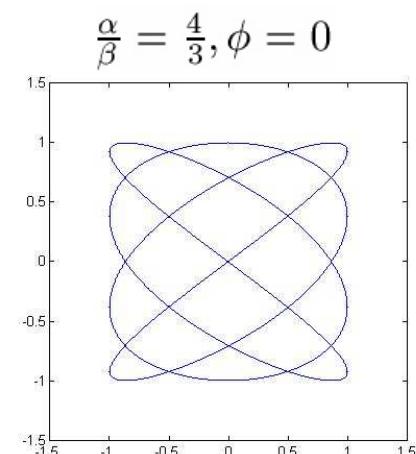
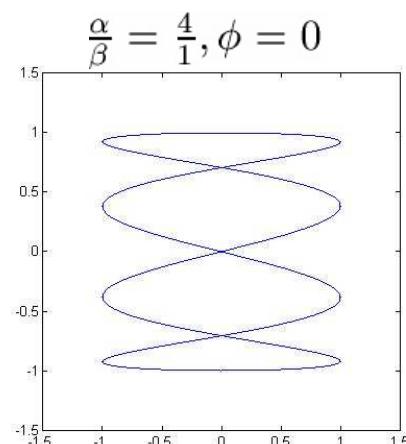
- To meet the optical refresh rate limitation:

$$\frac{1}{T} > 20Hz$$

Limits of the Lazerdillo

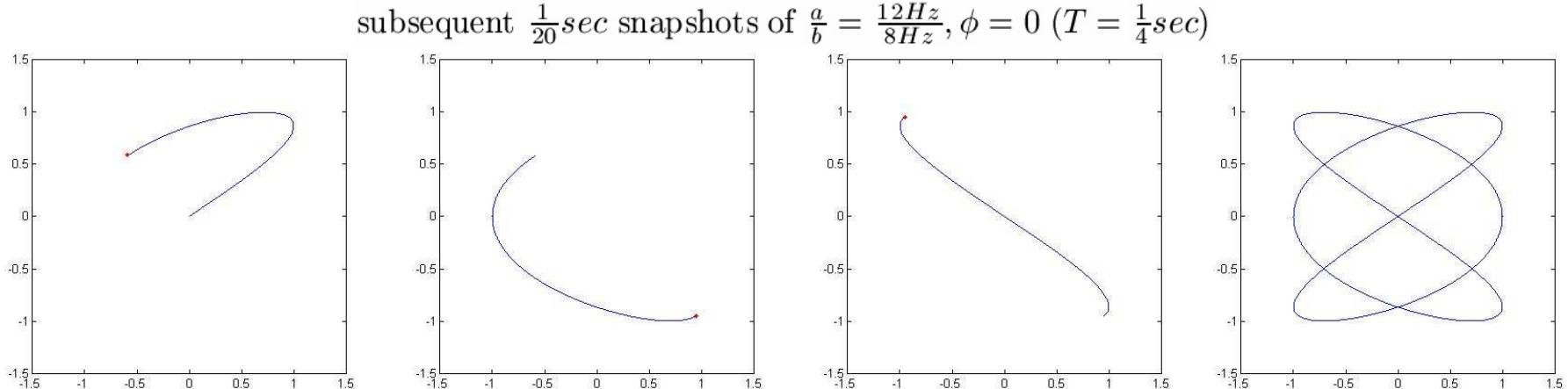
- Lazerdillo limits the figures that can be displayed
- Neither α nor β in the simplified frequency ratio can be above 4

$$T = \frac{\alpha}{a} = \frac{4 \frac{\text{cycles}}{\text{frame}}}{80 \text{Hz}}$$
$$= \frac{1}{20} \text{second}$$



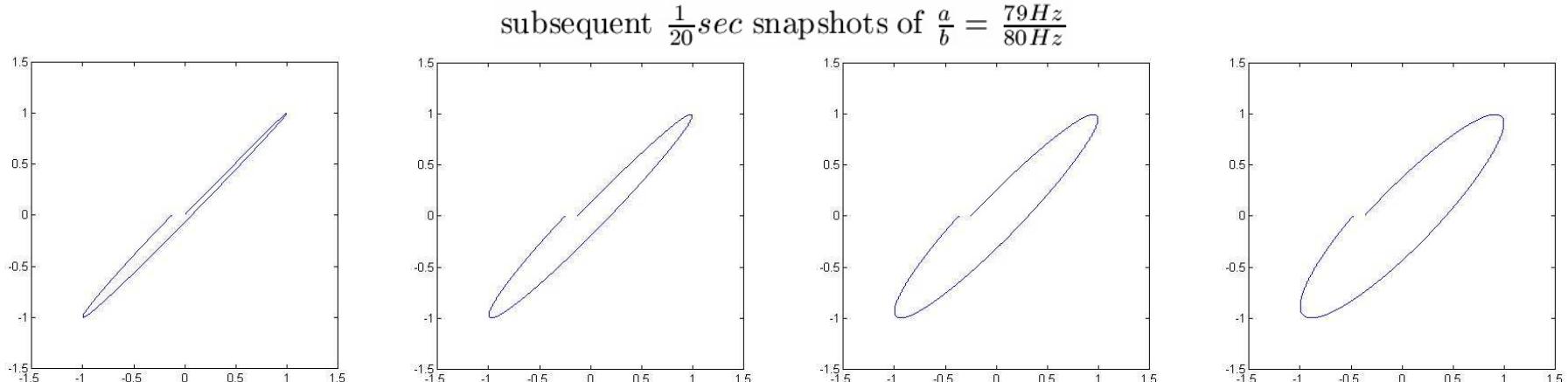
Fusion Failure

- For large T, “frames” are not full figures
- Snapshots below for $T \gg 1/20$ sec
- Full figure shown for comparison

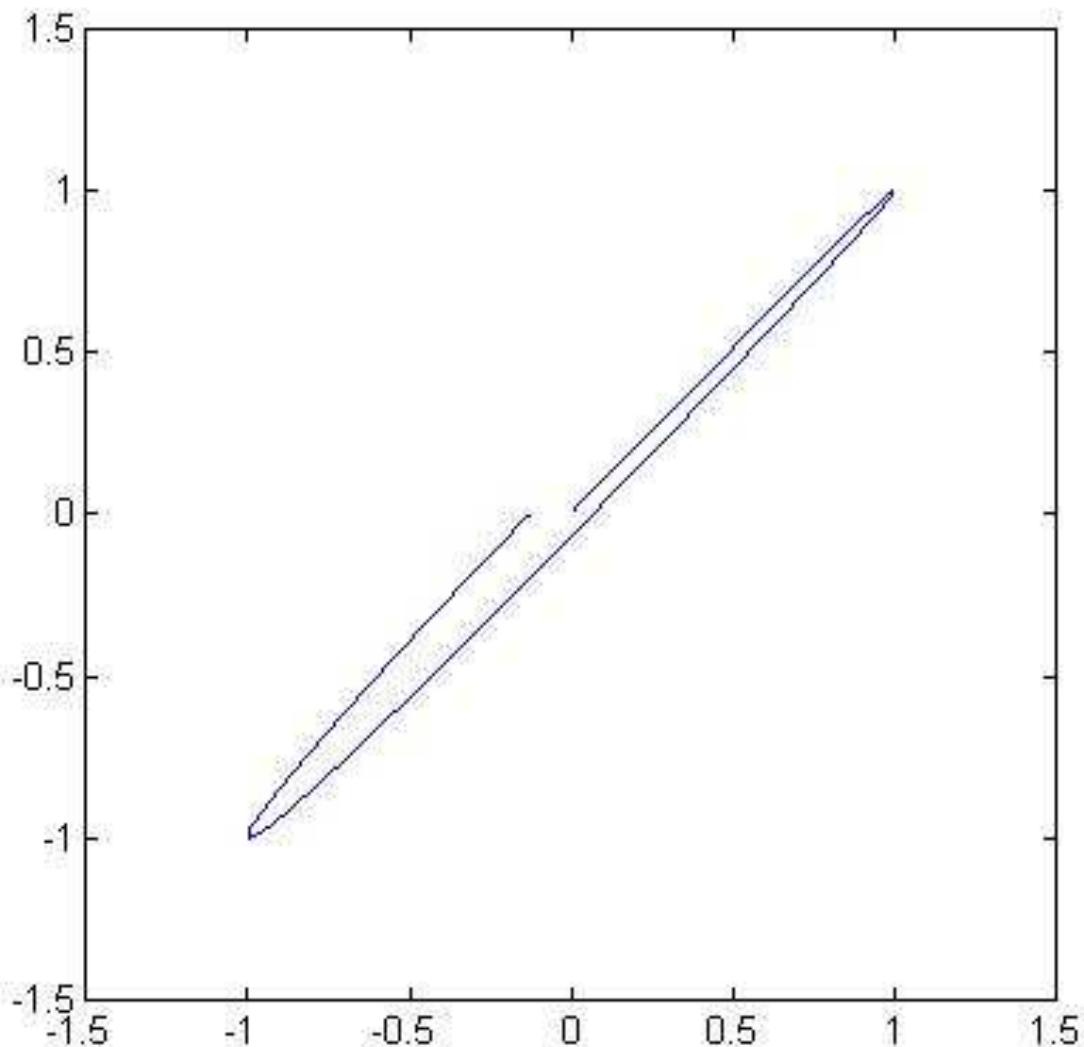


Rotation

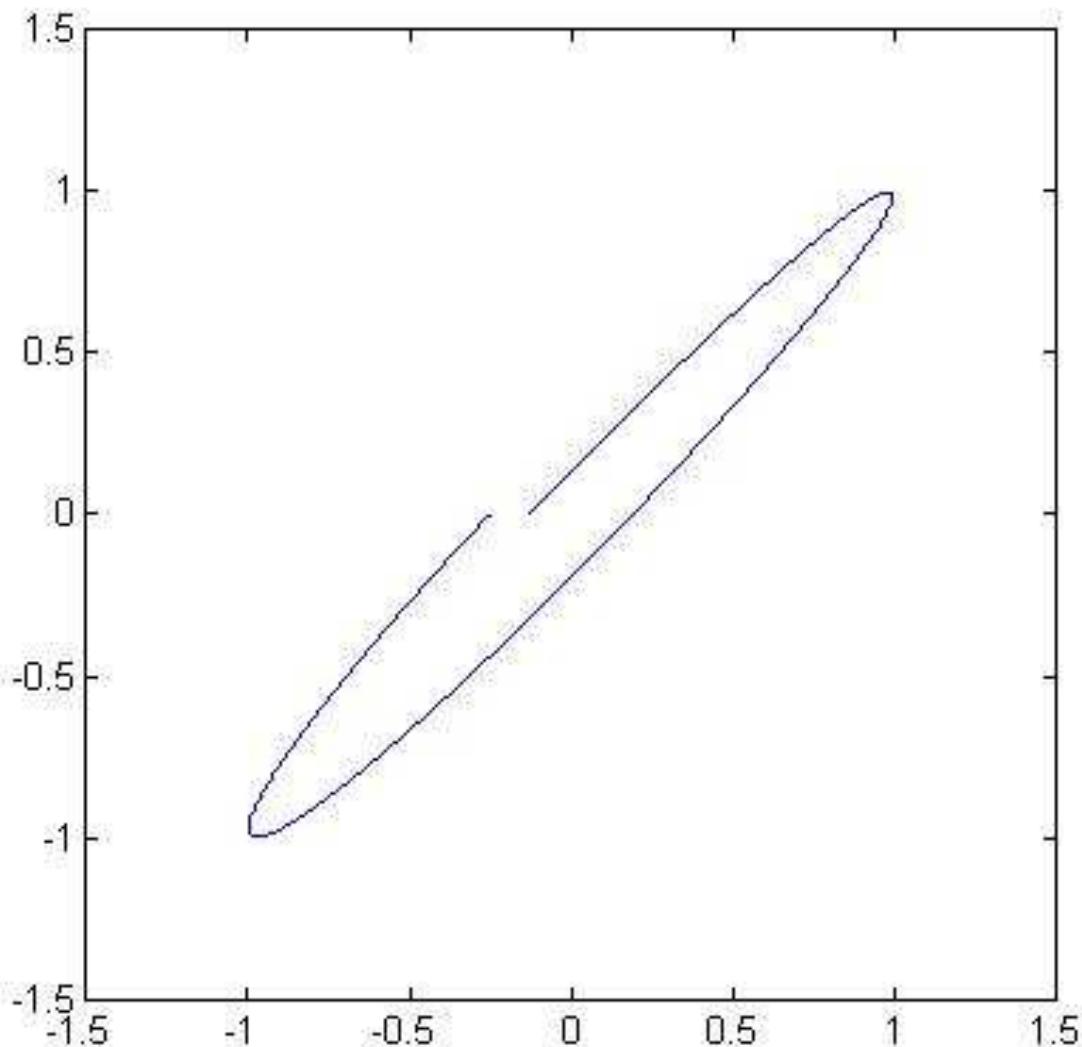
- Below: successive images from Lissajous figure with ratio 79/80 ($T = 1$).
- The frames move smoothly and the figure appears to rotate!



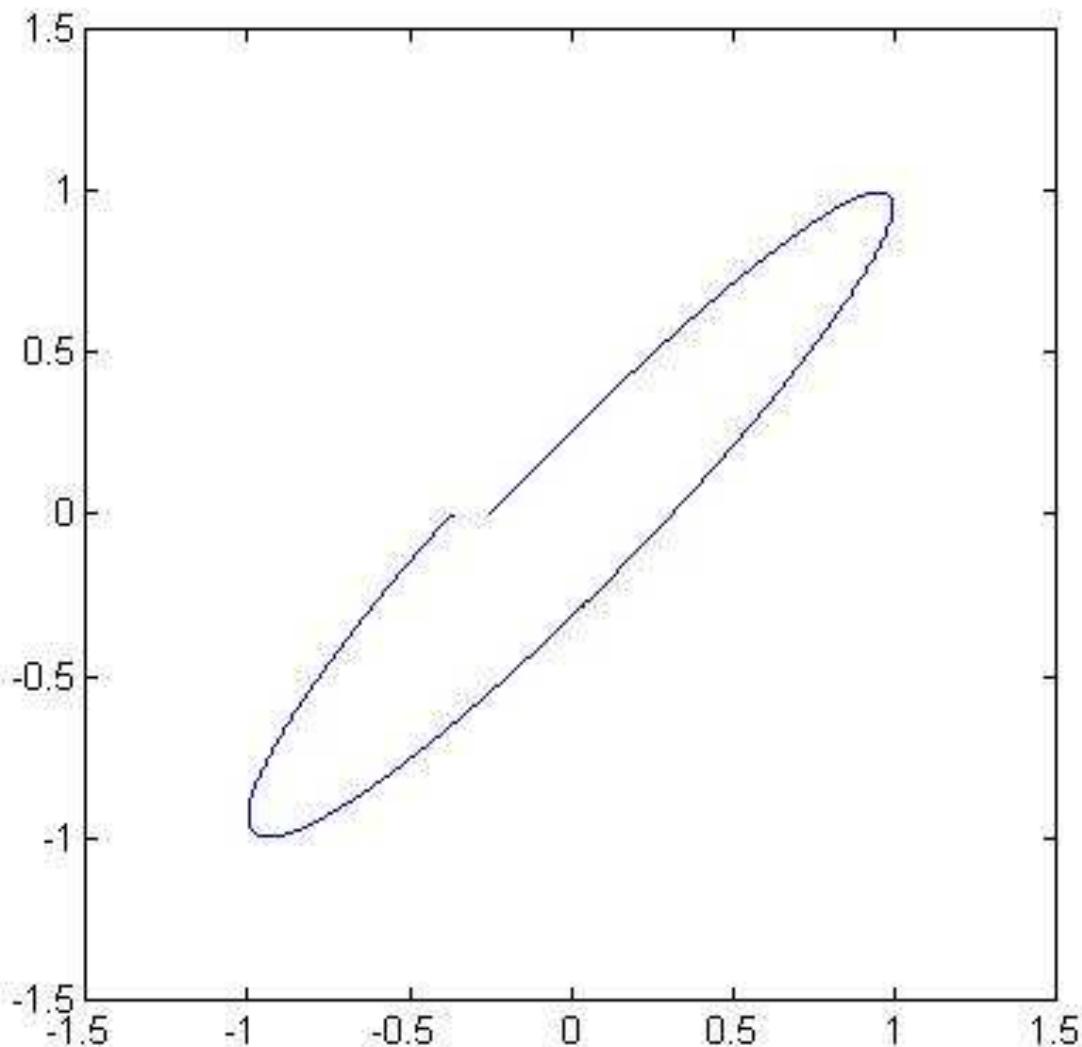
Rotation Example



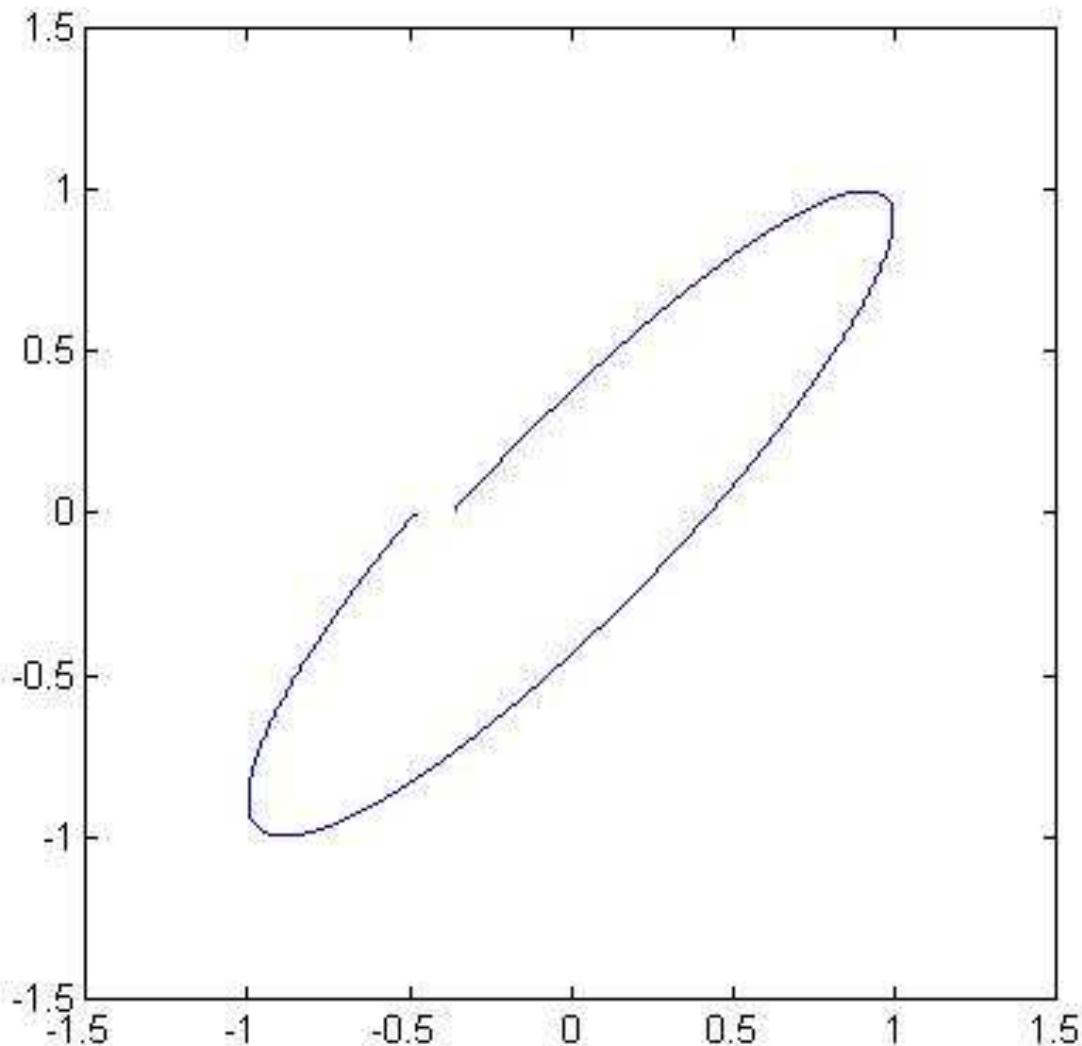
Rotation Example



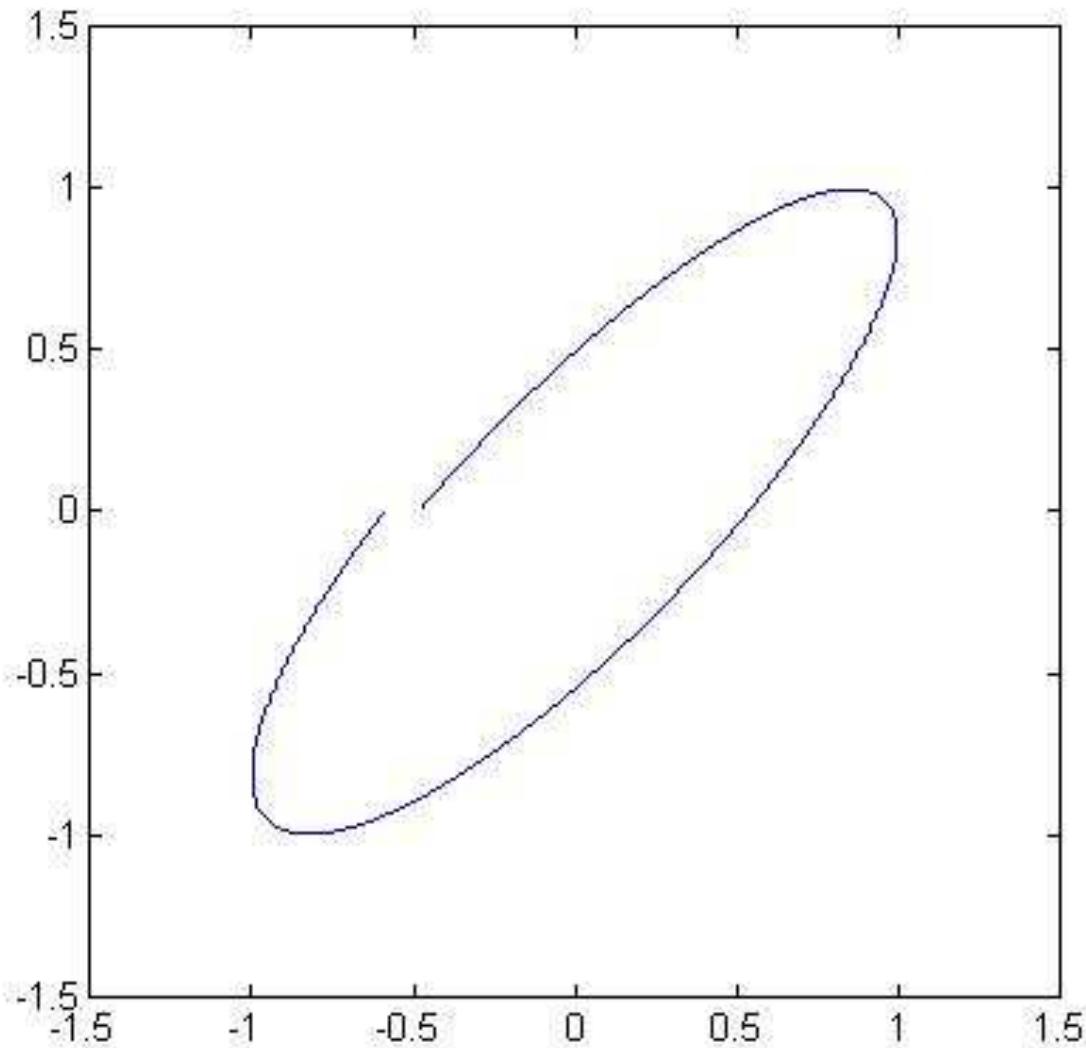
Rotation Example



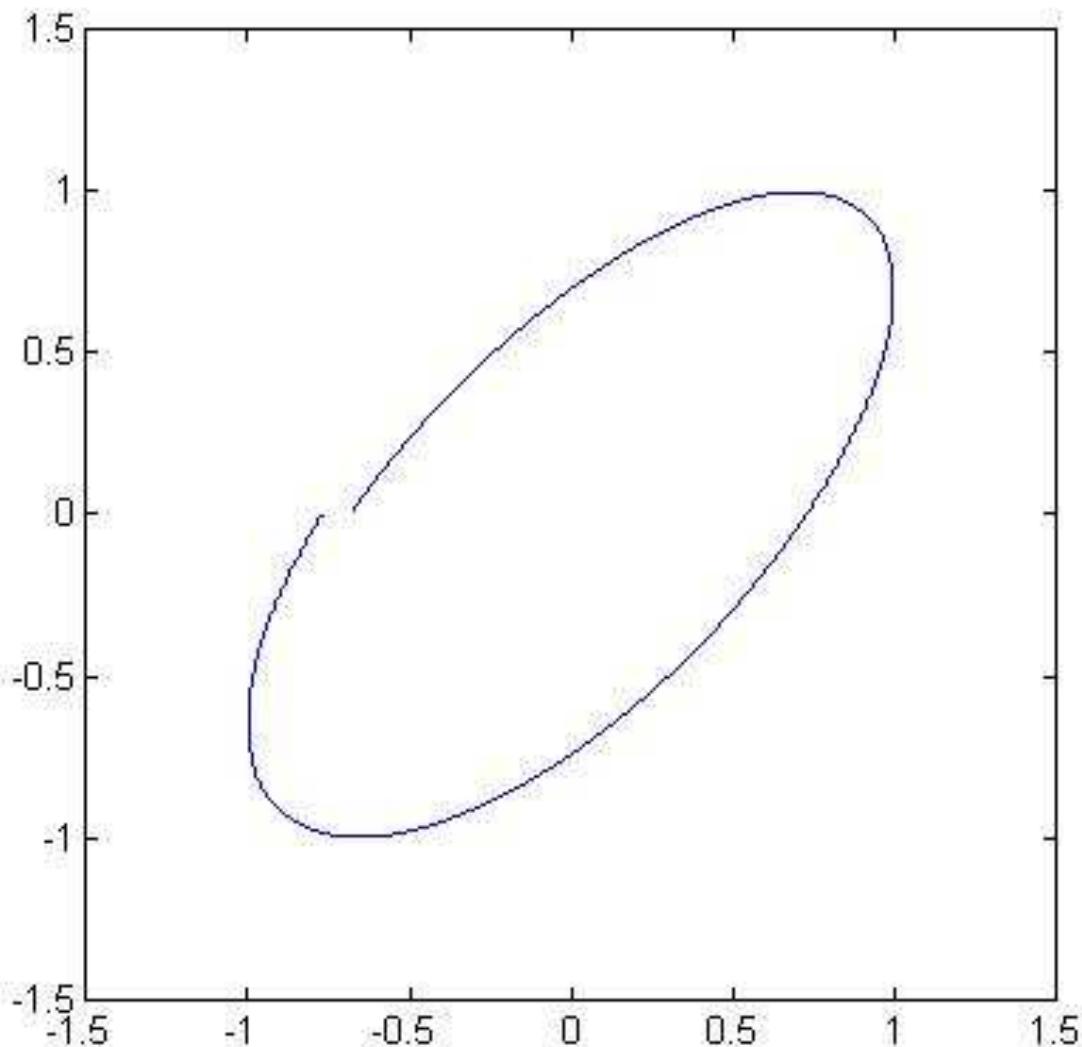
Rotation Example



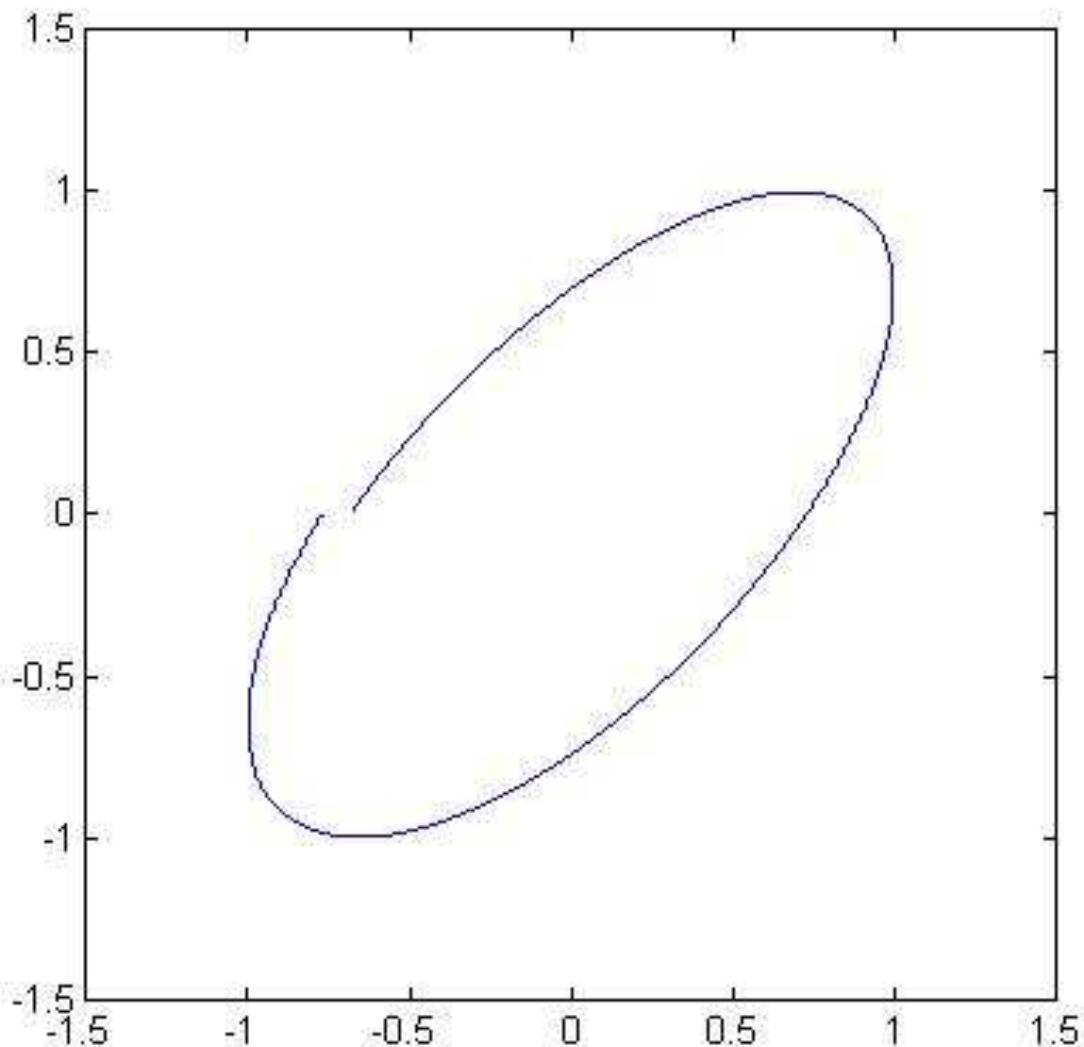
Rotation Example



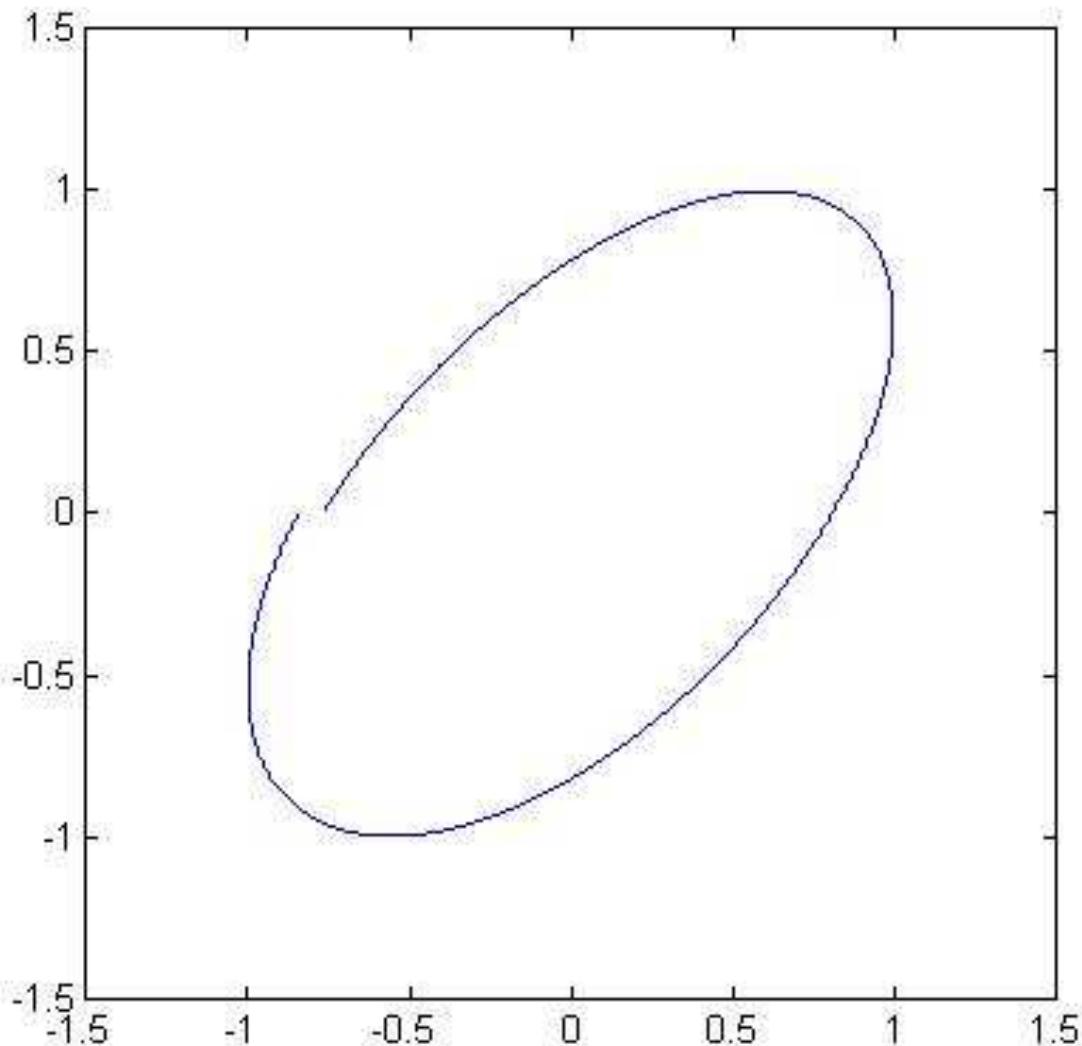
Rotation Example



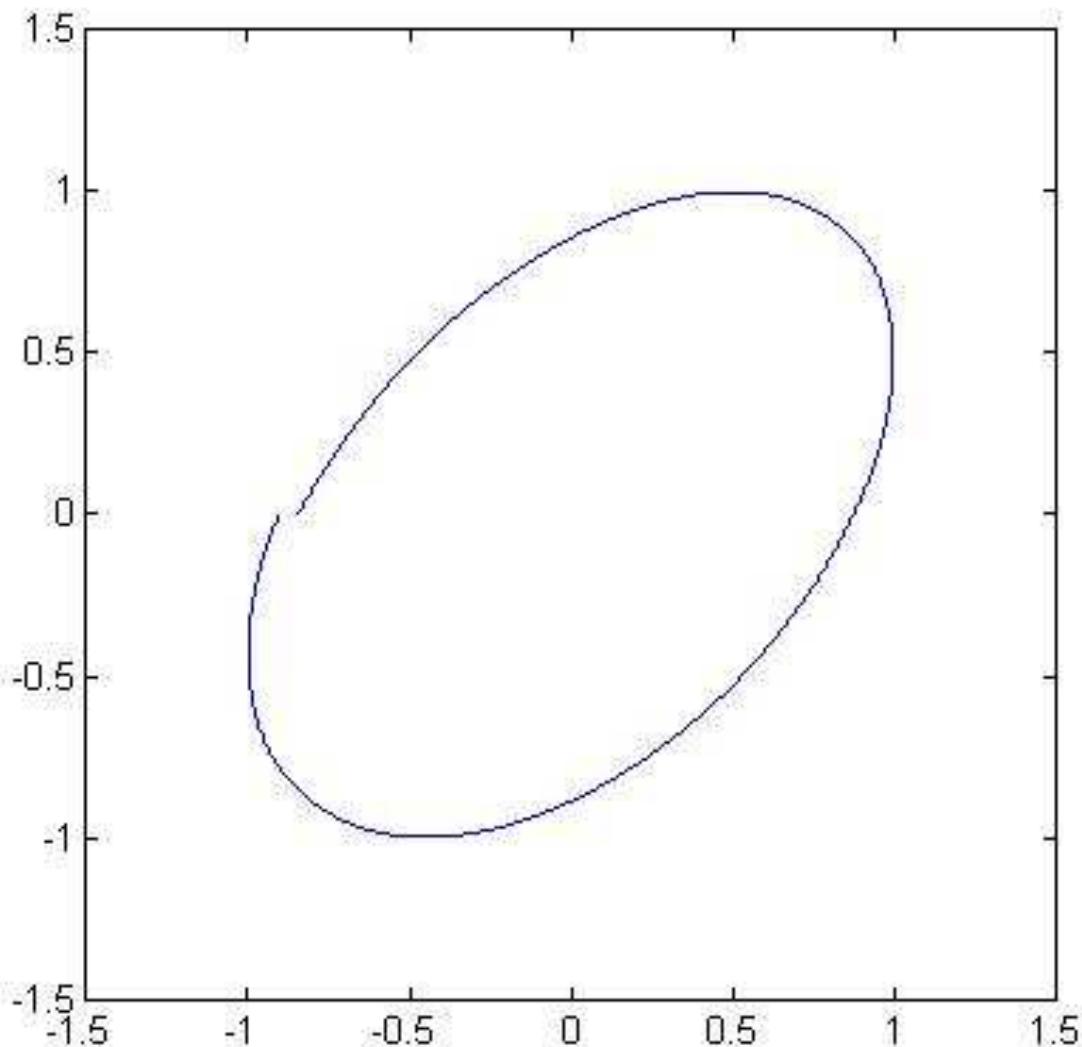
Rotation Example



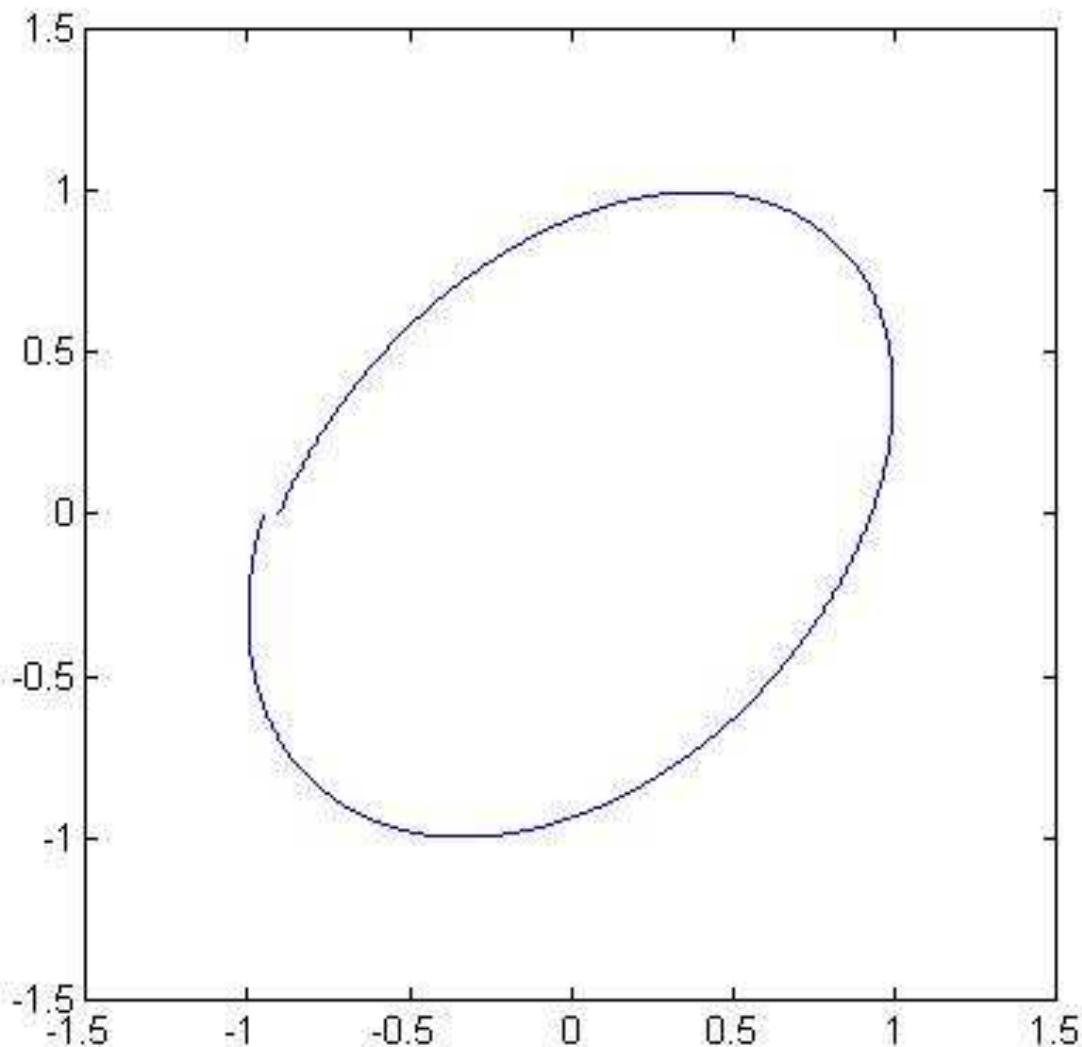
Rotation Example



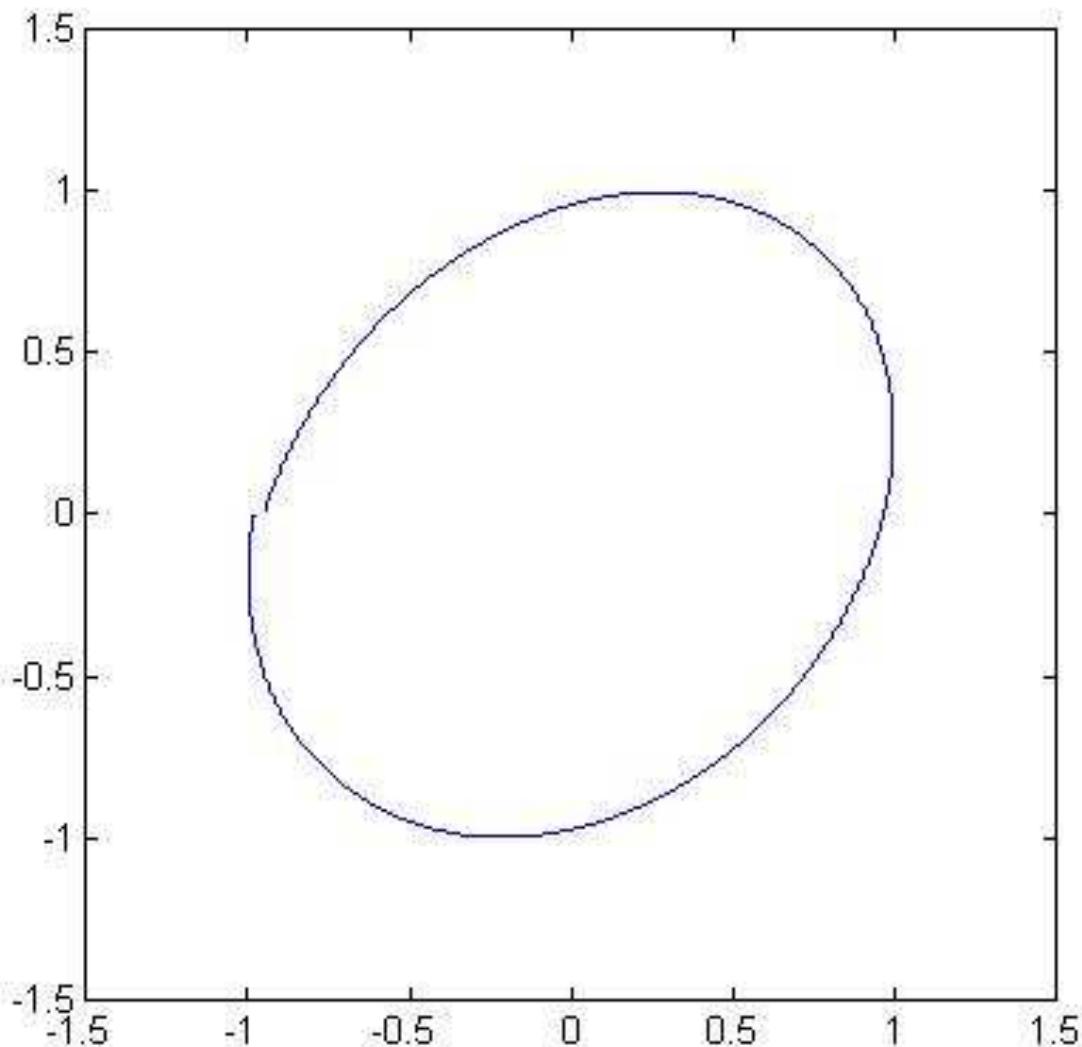
Rotation Example



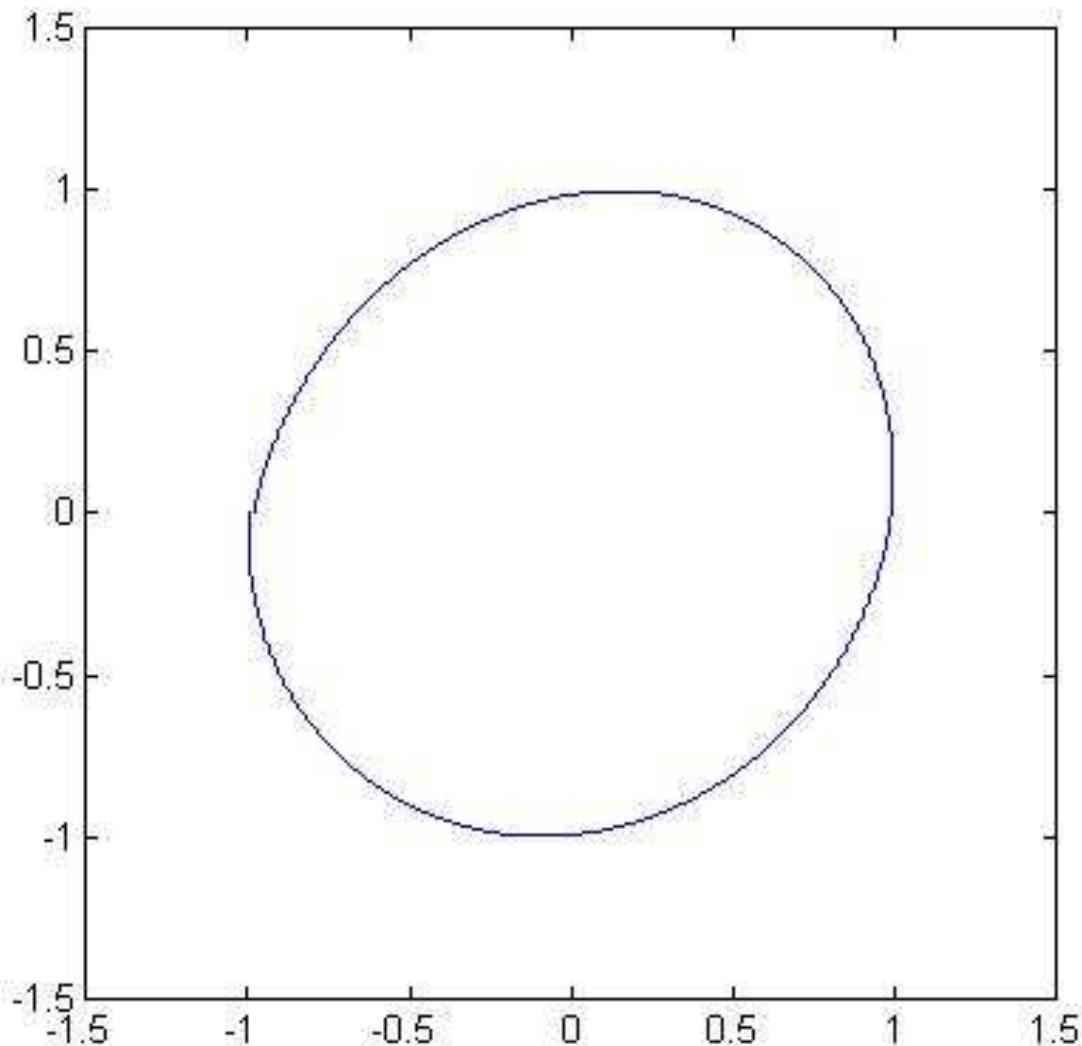
Rotation Example



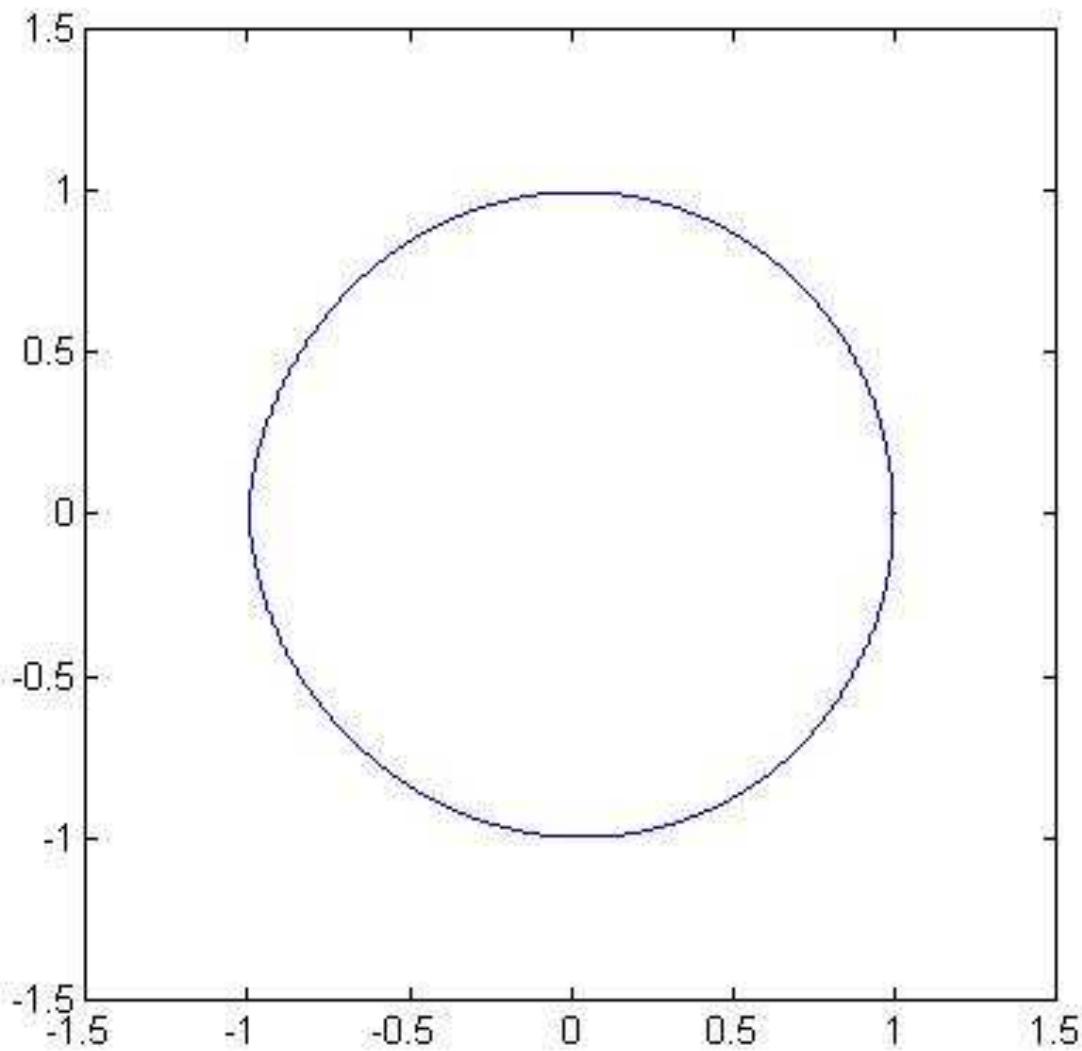
Rotation Example



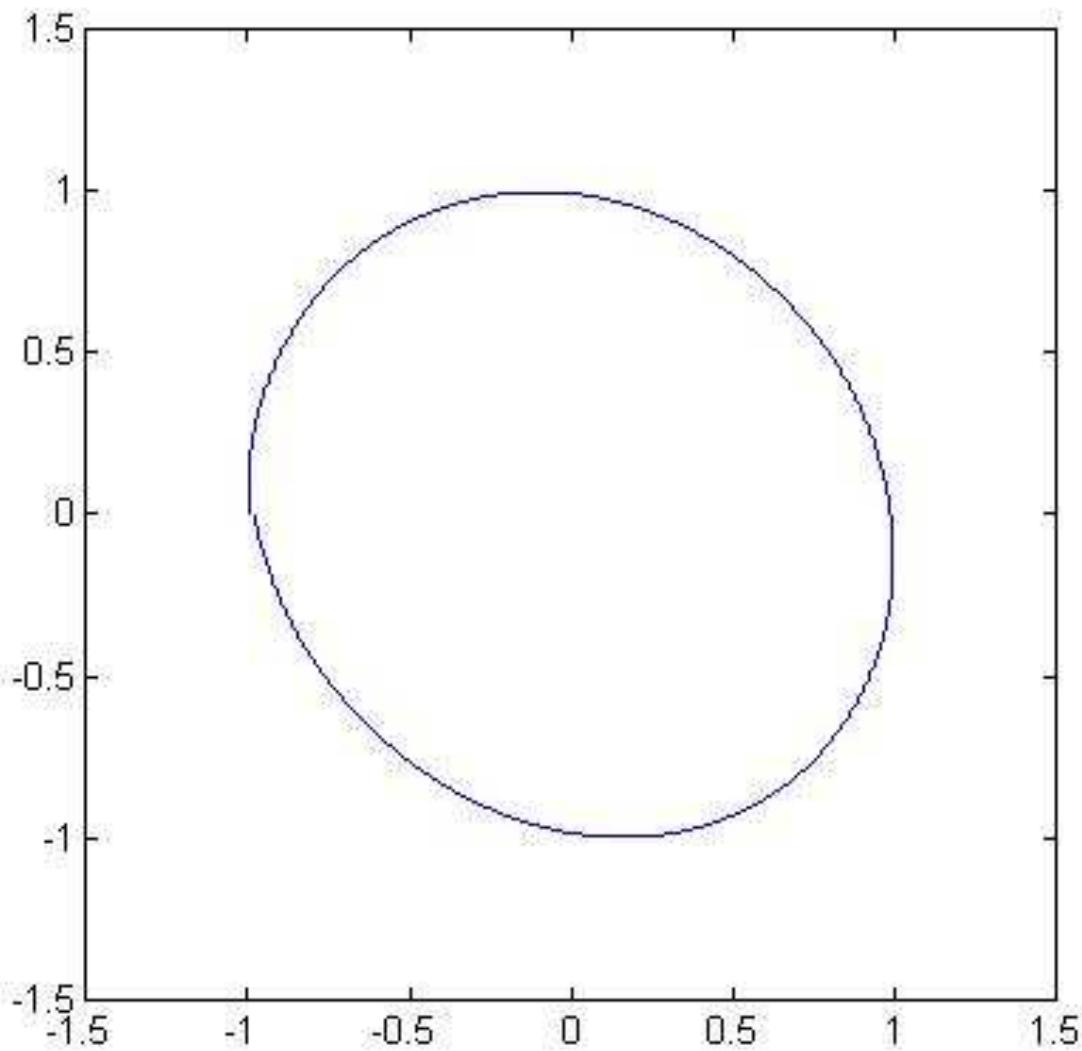
Rotation Example



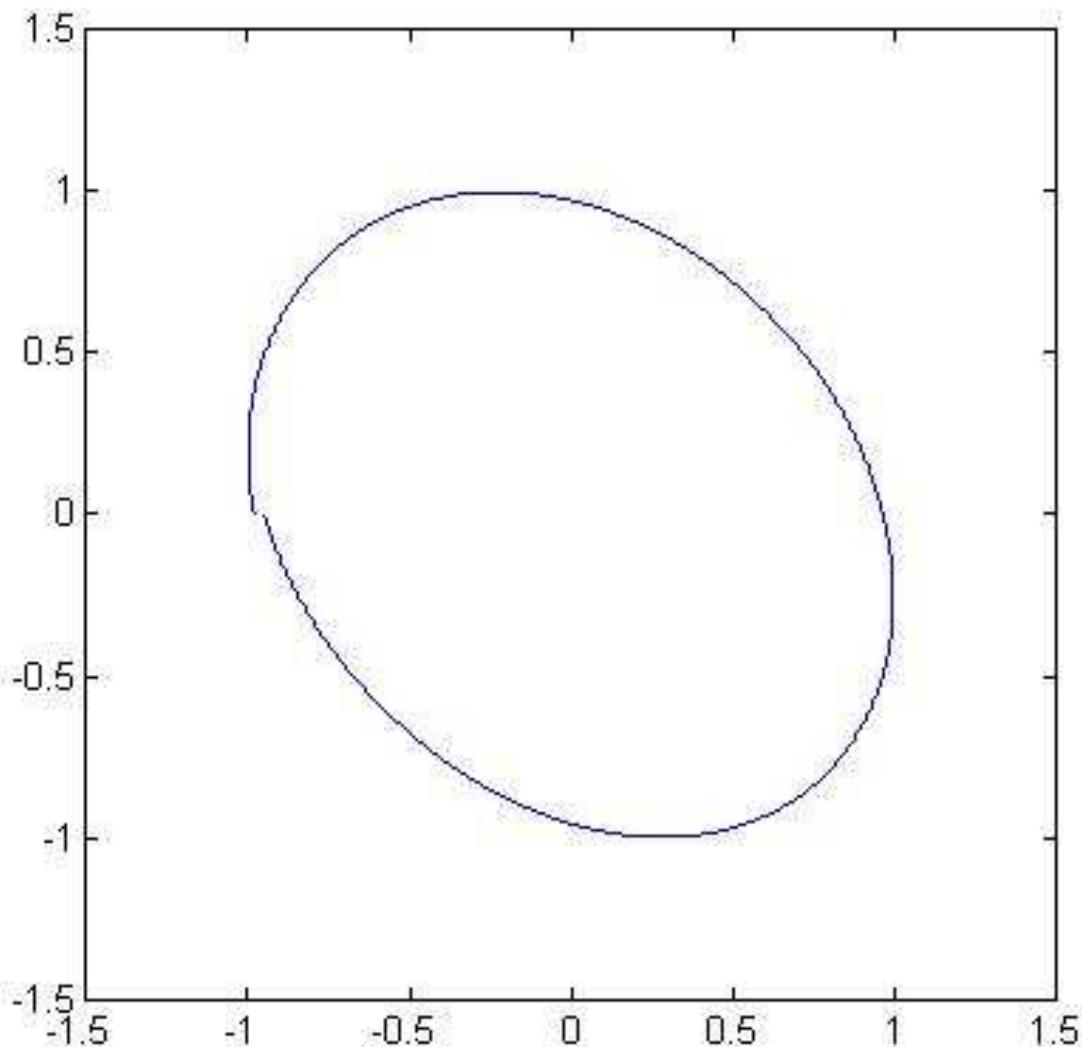
Rotation Example



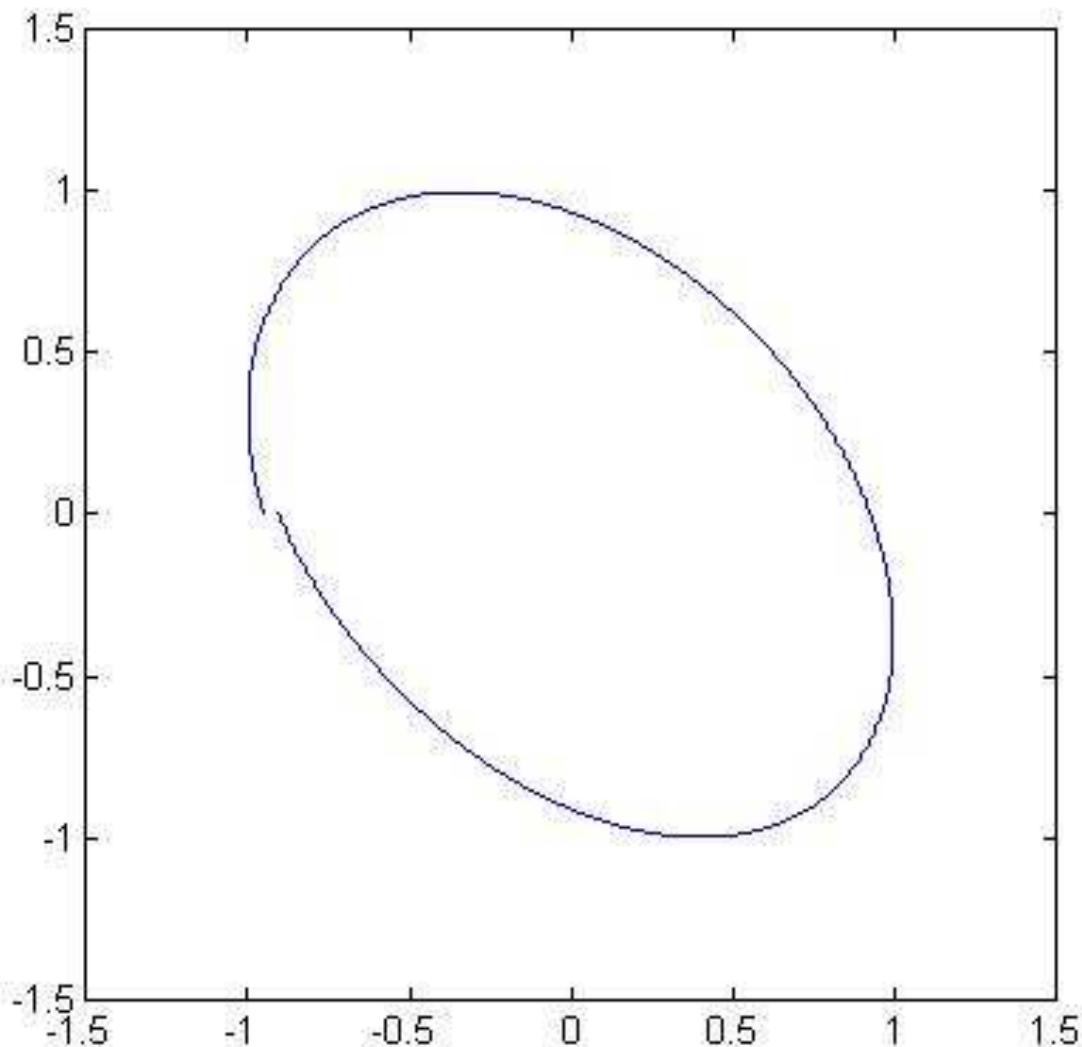
Rotation Example



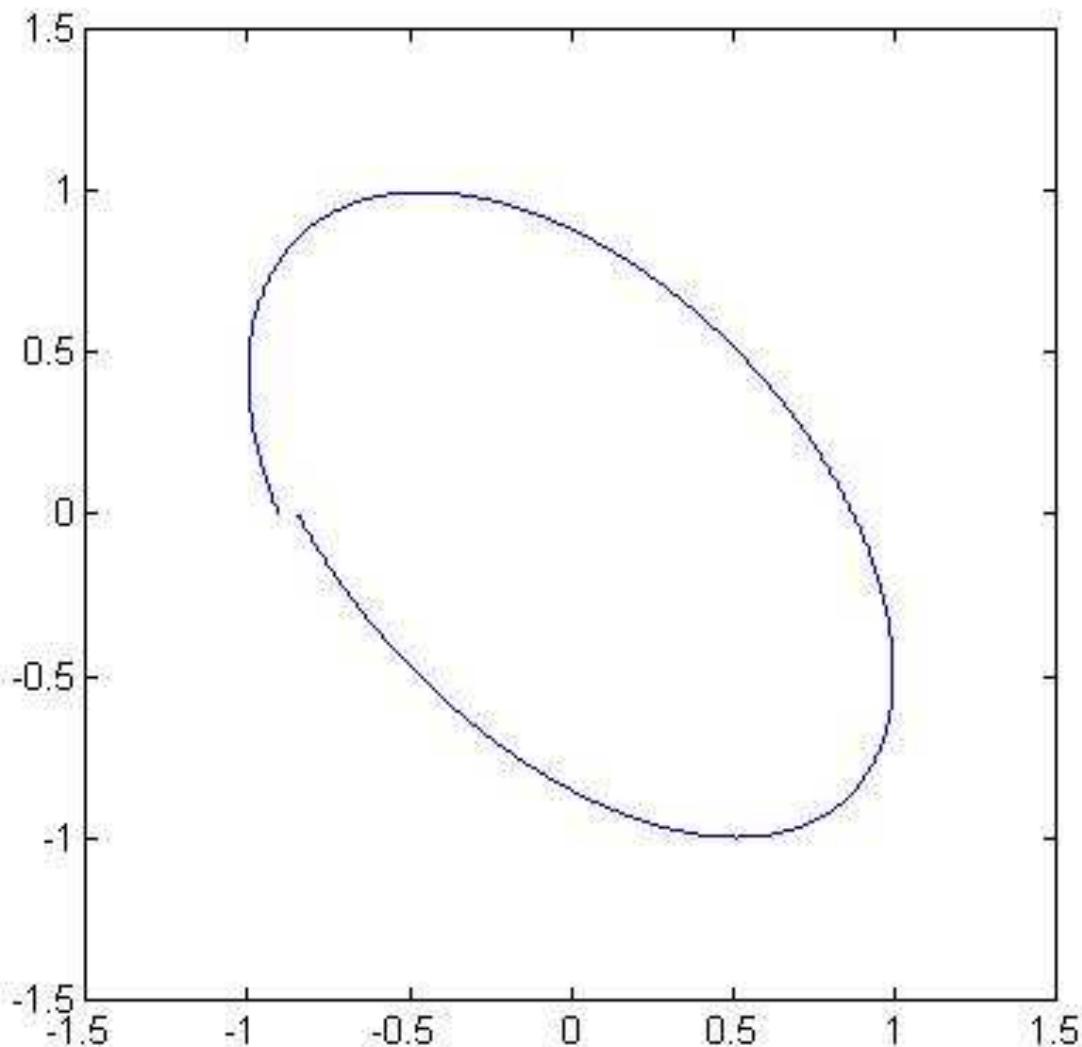
Rotation Example



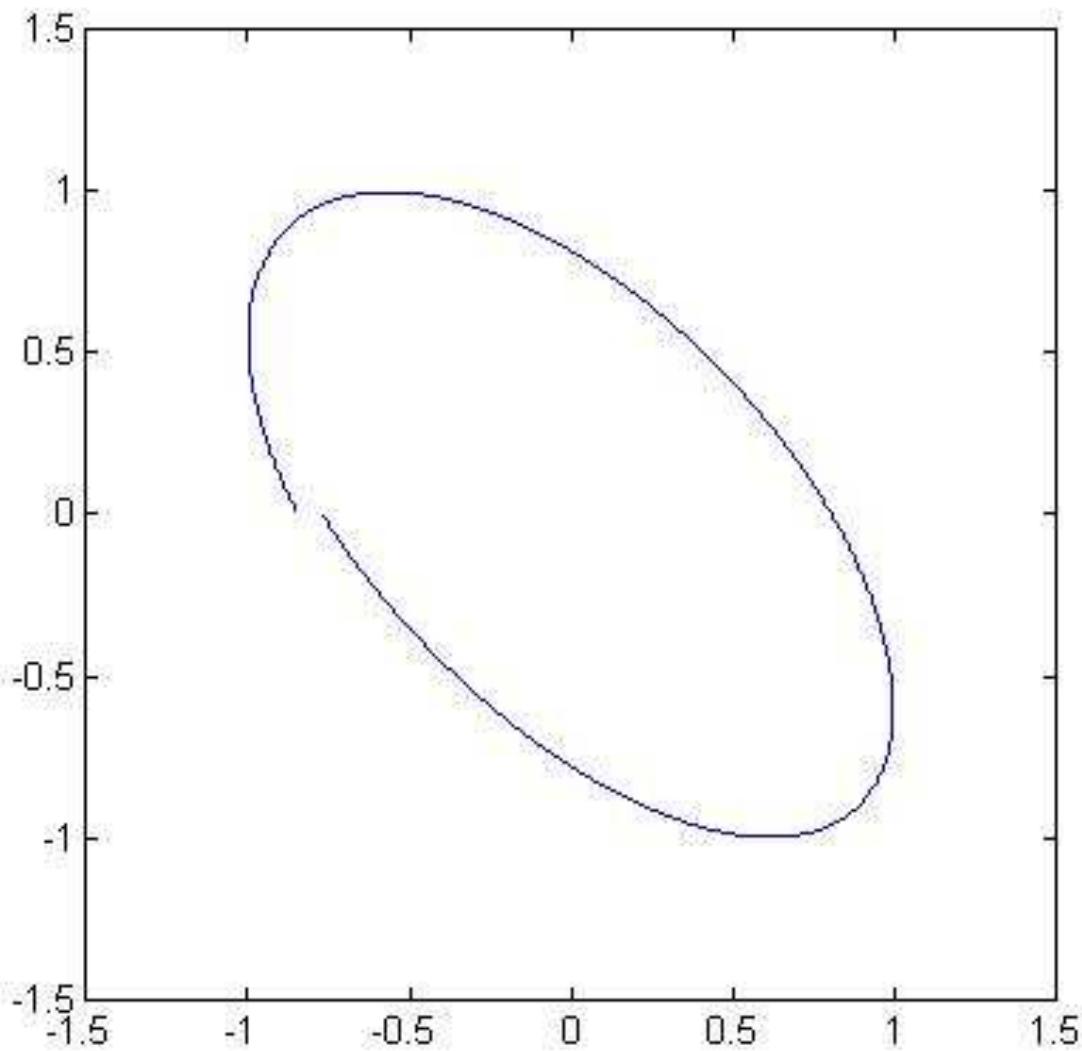
Rotation Example



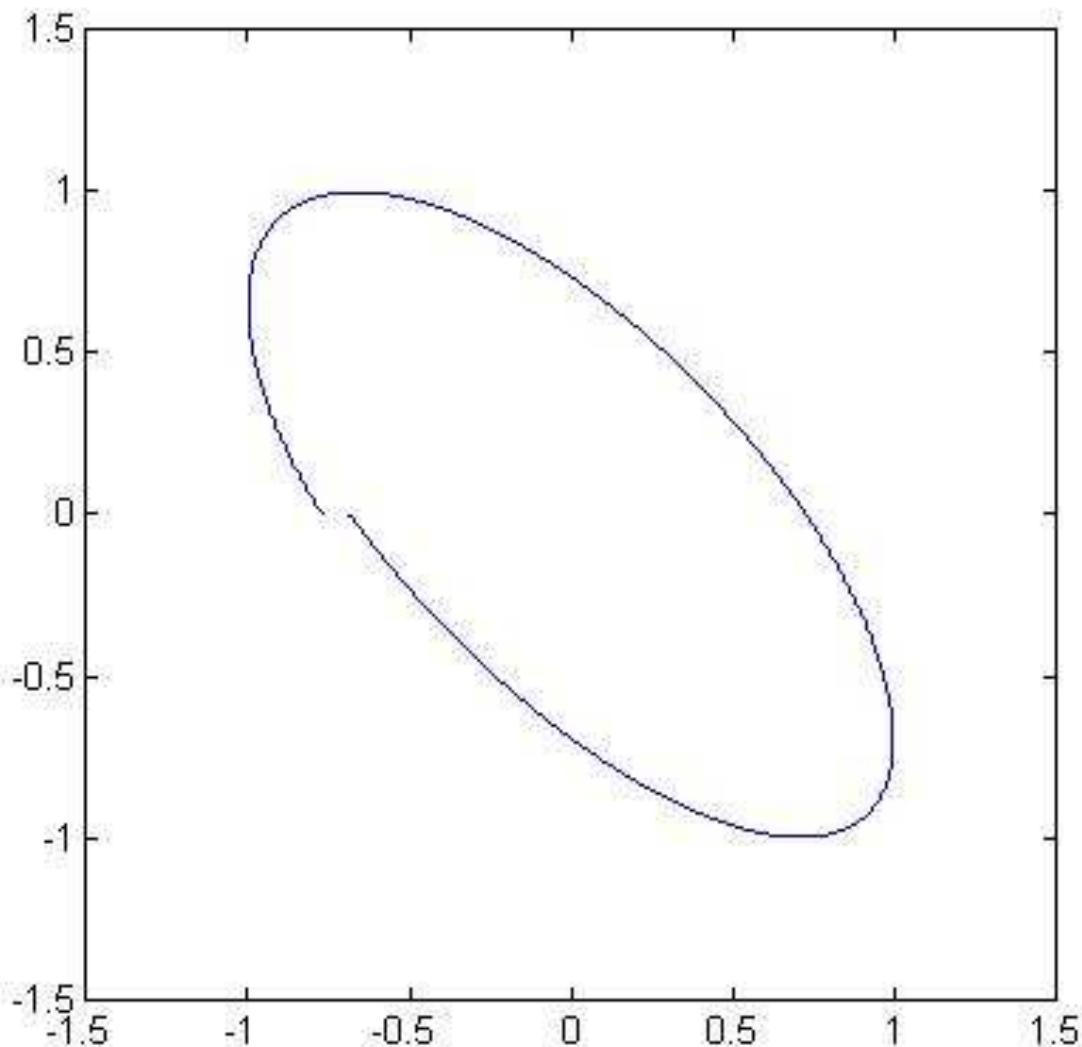
Rotation Example



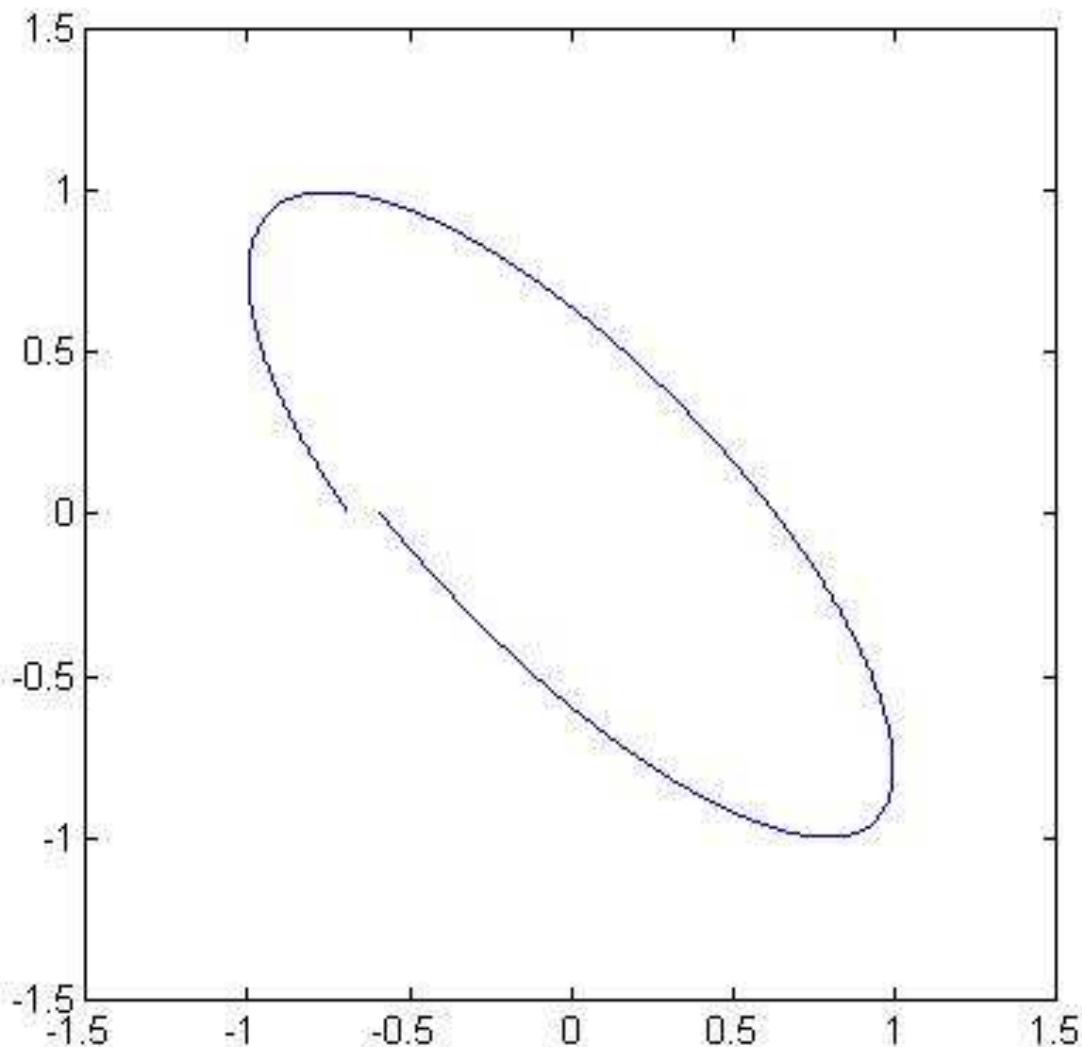
Rotation Example



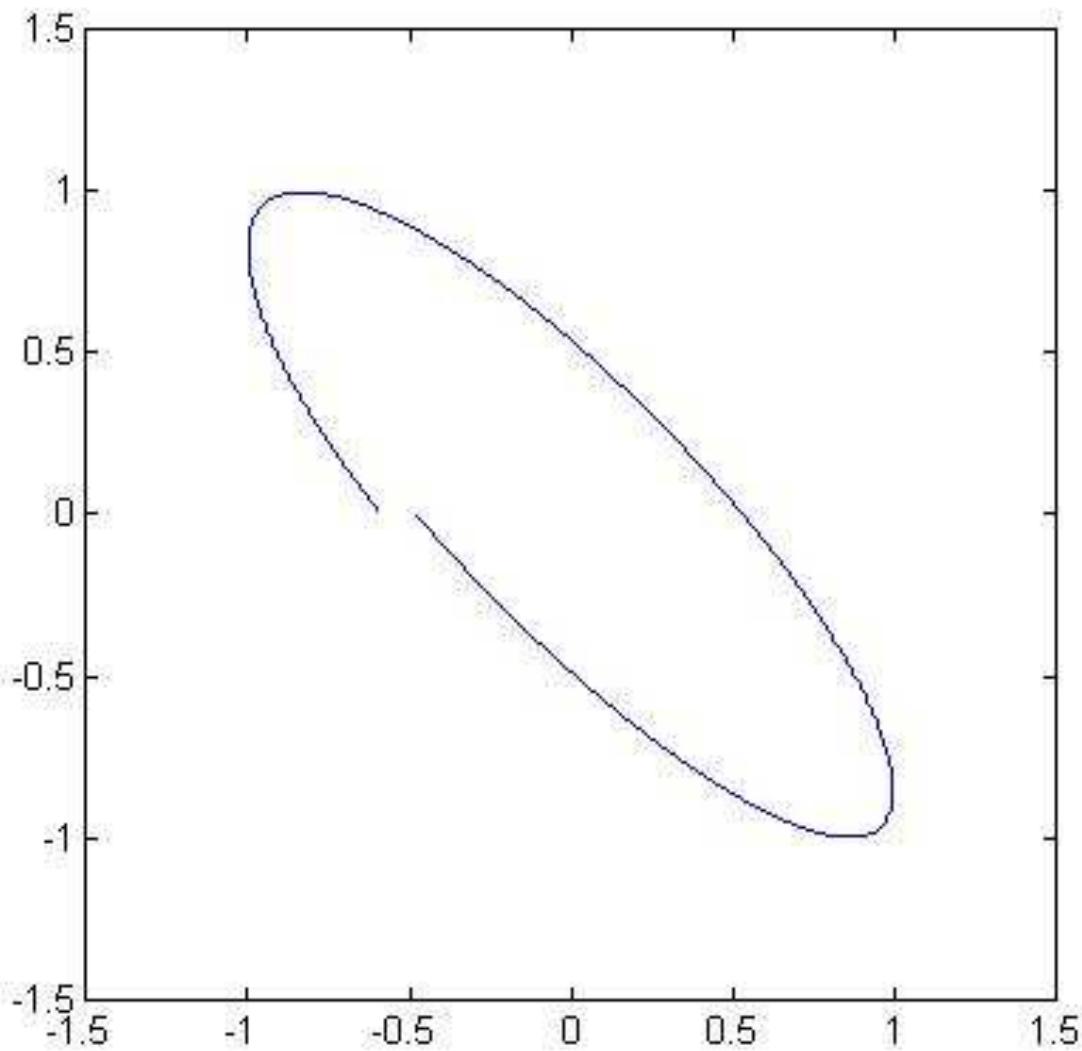
Rotation Example



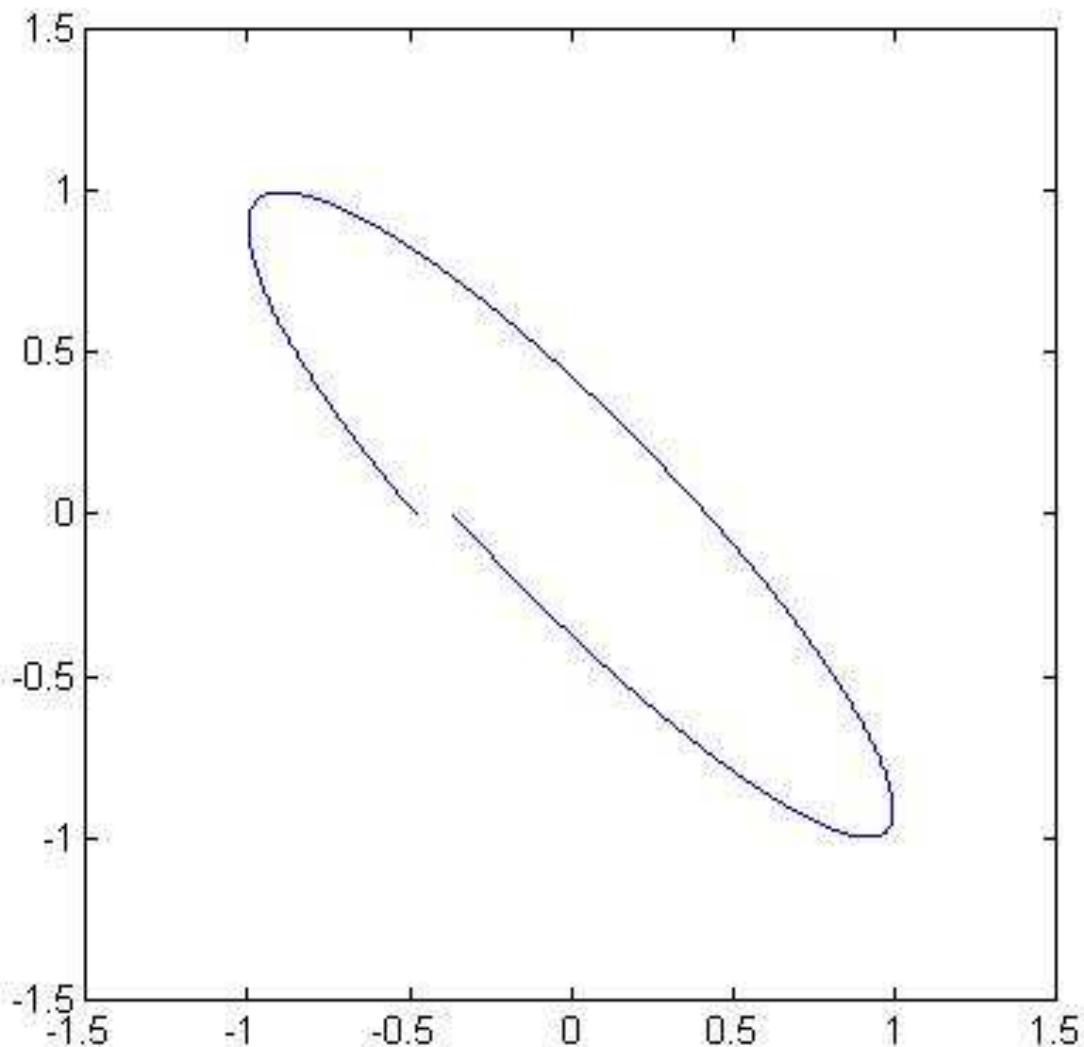
Rotation Example



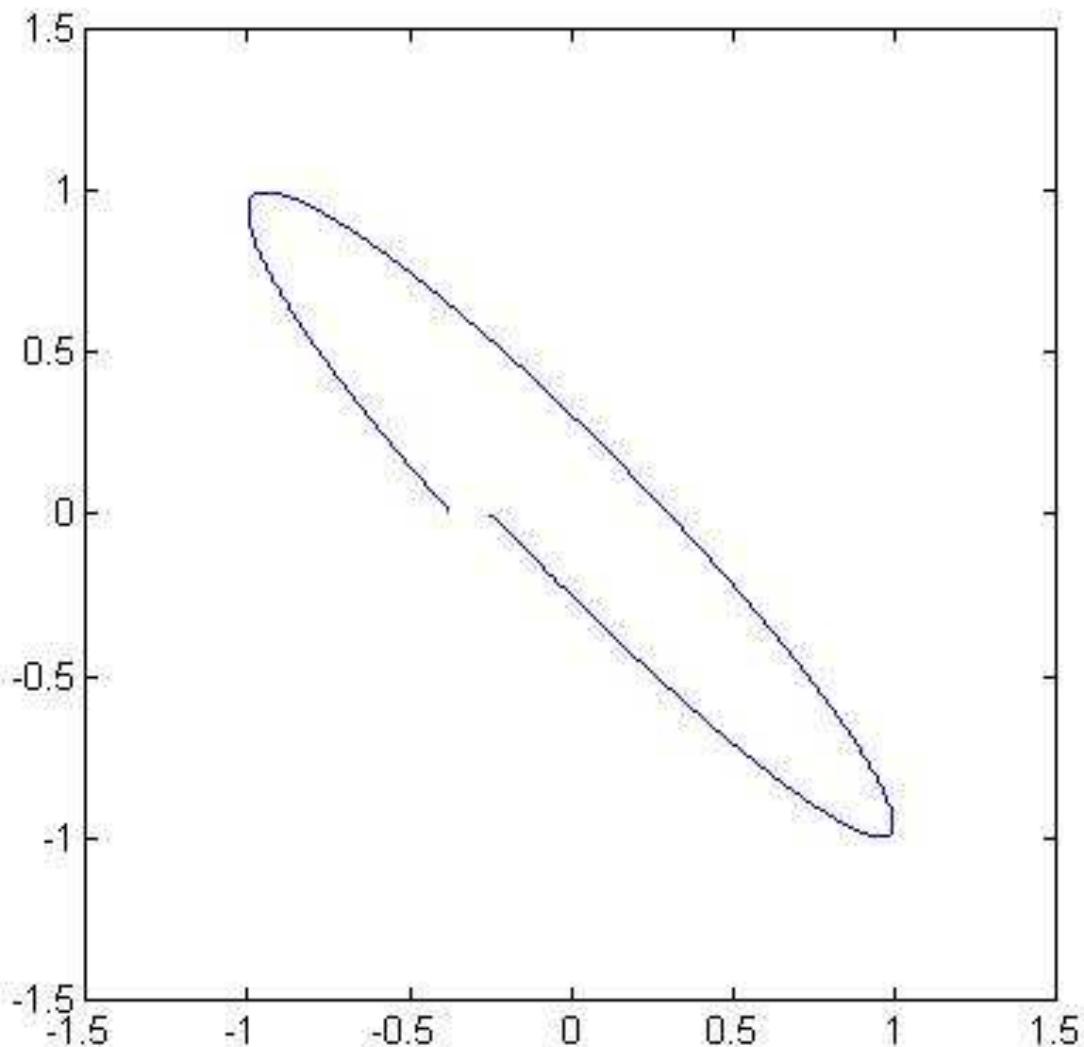
Rotation Example



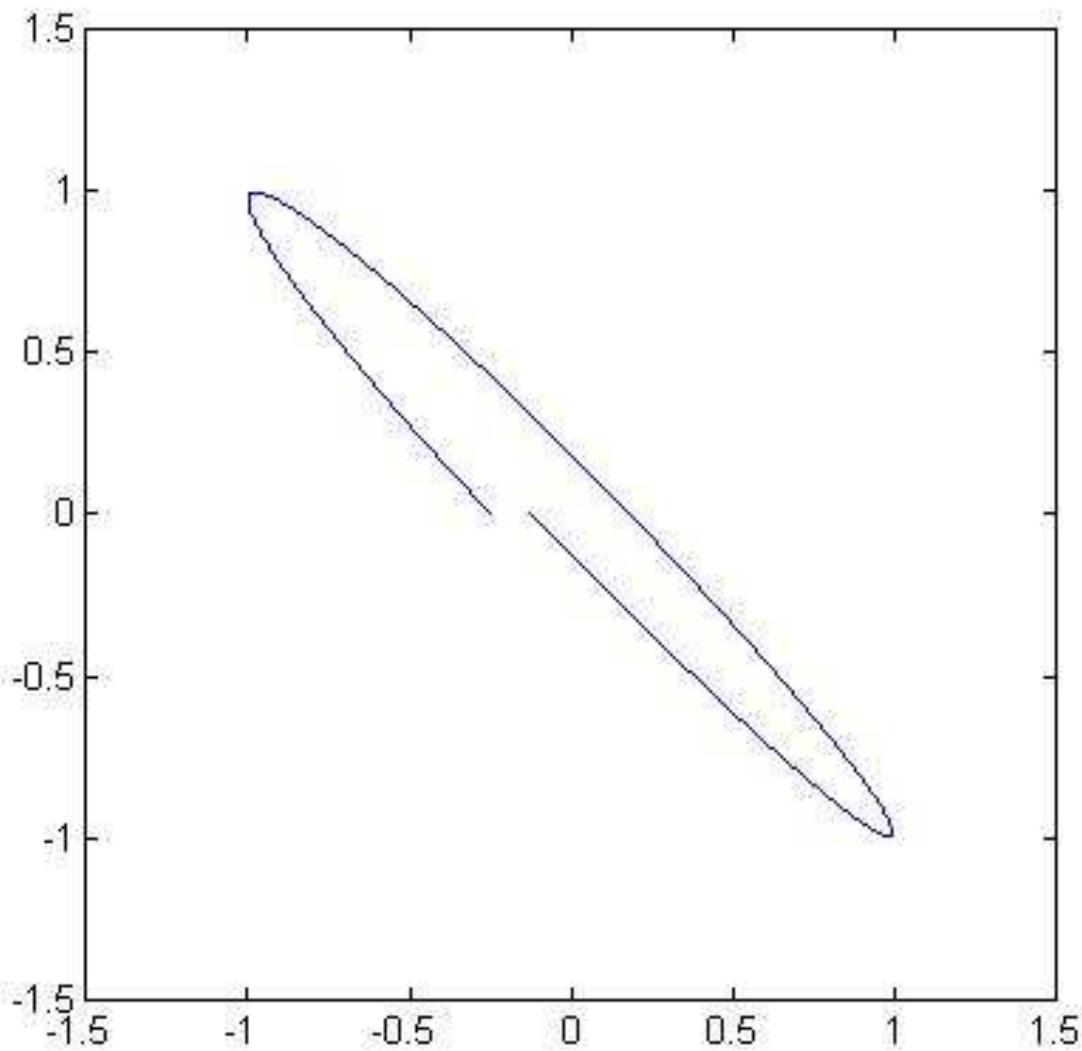
Rotation Example



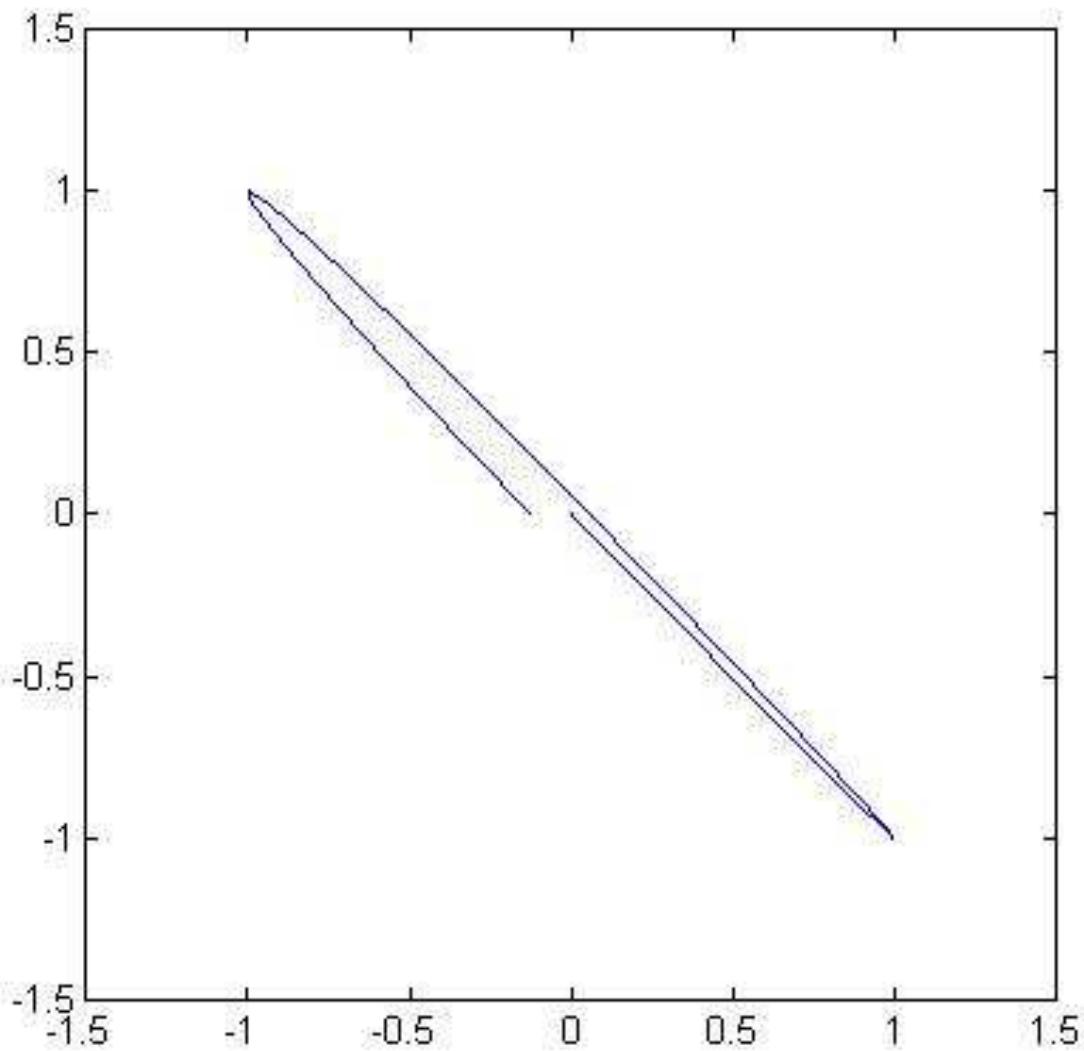
Rotation Example



Rotation Example



Rotation Example



Rotation: Another Look

- Why does rotation occur?
- If we look at the equations of the figure in the previous example:

$$x = \sin 2\pi 79t$$

$$y = \sin 2\pi 80t$$

$$x = \sin 2\pi 79t$$

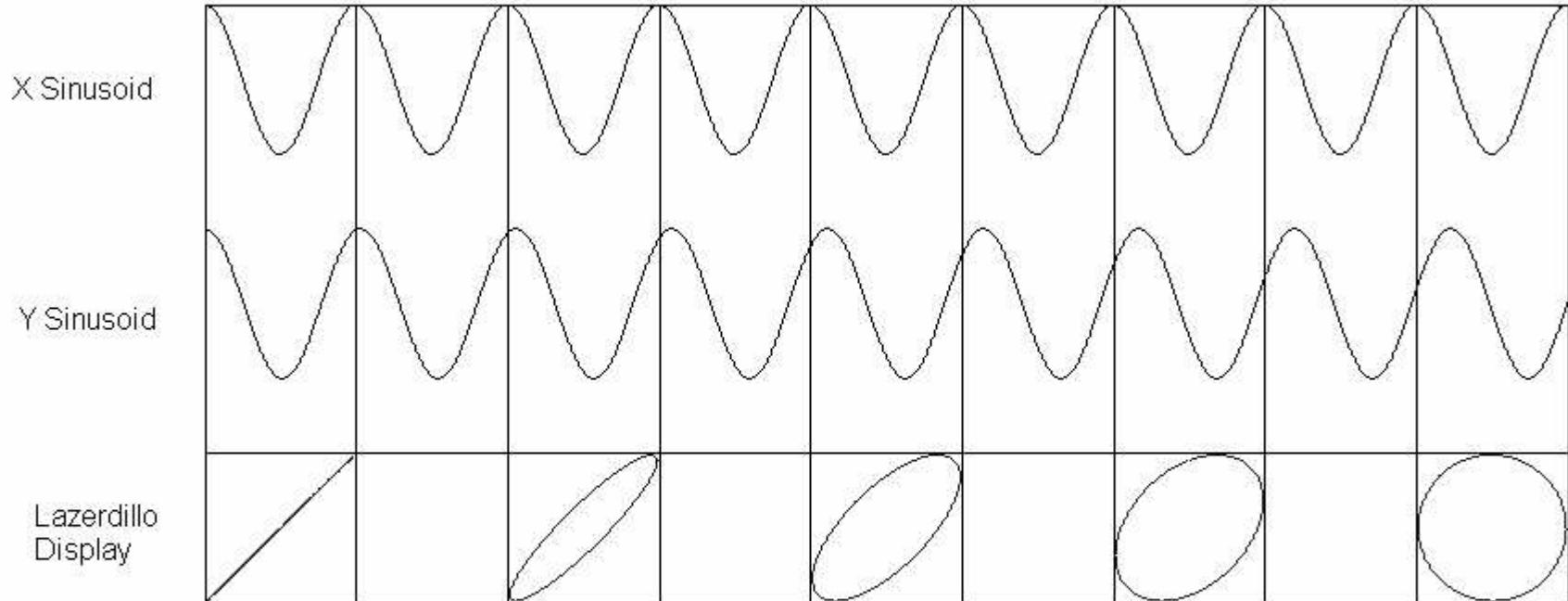
$$y = \sin 2\pi(79t + t)$$

$$x = \sin 2\pi a t$$

$$y = \sin 2\pi(b t + \phi(t))$$

- Small frequency offset becomes time-varying phase!

Rotating Figures



- Sinusoids have a slightly different frequency
- Frame by frame they appear to have identical frequencies with different instantaneous phase offsets

Rotation

a	80	80	80	80
b	80	79.9	79	75
α/β	1/1	1/1	1/1	1/1
$\phi(t)$	0	-0.1*t	-t	-5*t
Rotation Frequency	0 Hz	0.1 Hz	1 Hz	5 Hz
Rotation Period	0 sec	10 sec	1 sec	.2 sec

- Rotation frequencies above 1 Hz are too fast for the eye and the image blurs

Modes of Rotation

- Lets try a different example with a different base frequency ratio, say 2/3

$$x = \sin 2\pi 54t \qquad \qquad y = \sin 2\pi 80t$$

$$x = \sin 2\pi 54t \qquad \qquad y = \sin 2\pi(81t - t)$$

$$\phi_y(t) = -t \qquad \qquad frequency = 1\text{Hz}$$

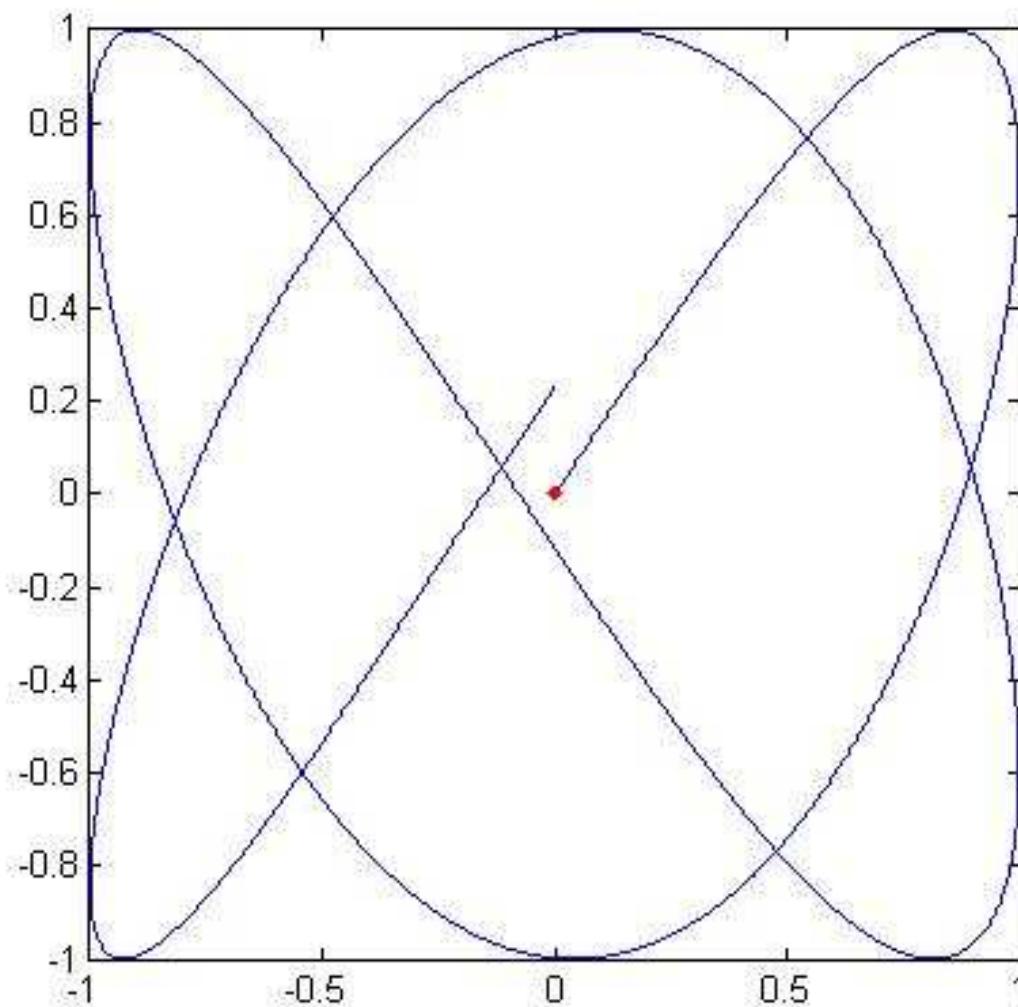
$$x = \sin 2\pi 54t \qquad \qquad y = \sin 2\pi 80t$$

$$x = \sin 2\pi(\frac{160}{3}t + \frac{2}{3}t) \qquad y = \sin 2\pi 80t$$

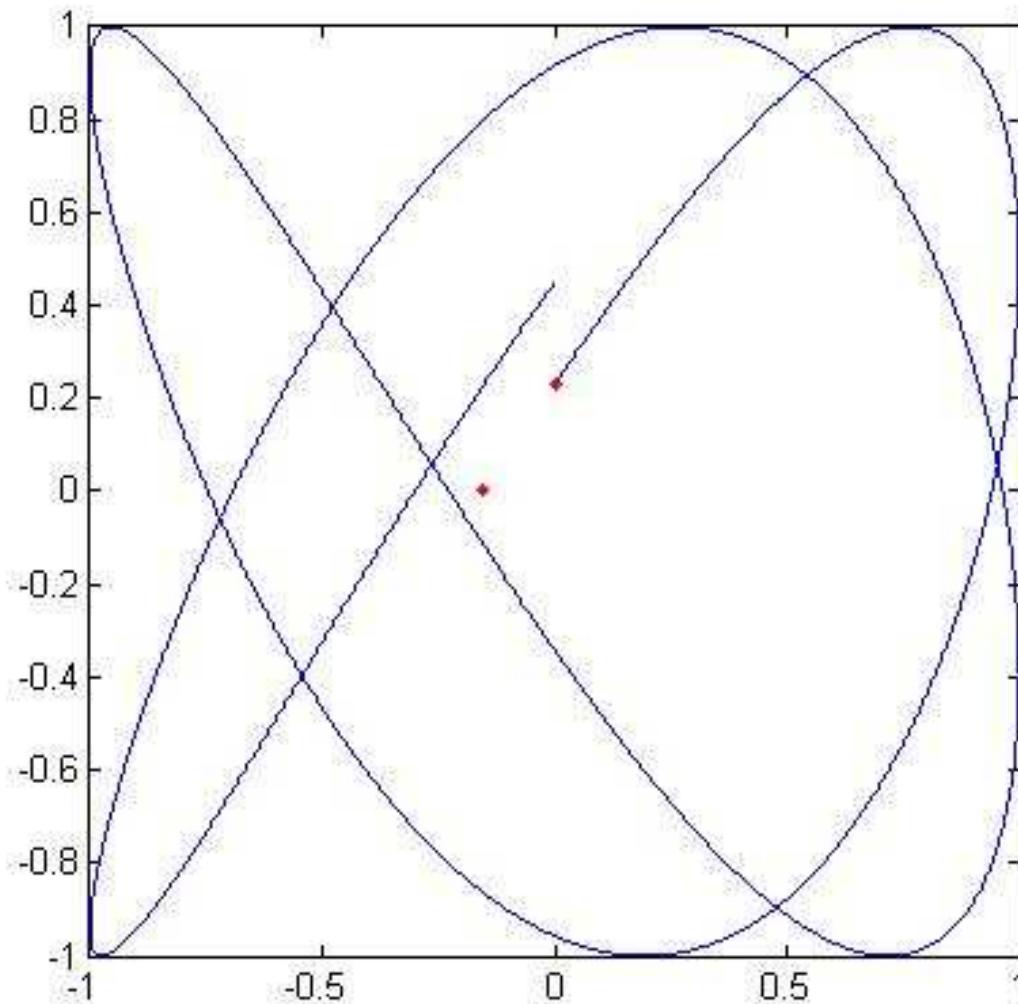
$$\phi_x(t) = \frac{2}{3}t \qquad \qquad frequency = 2/3\text{Hz}$$

- In this example the rotation frequency is different about the x and y axes

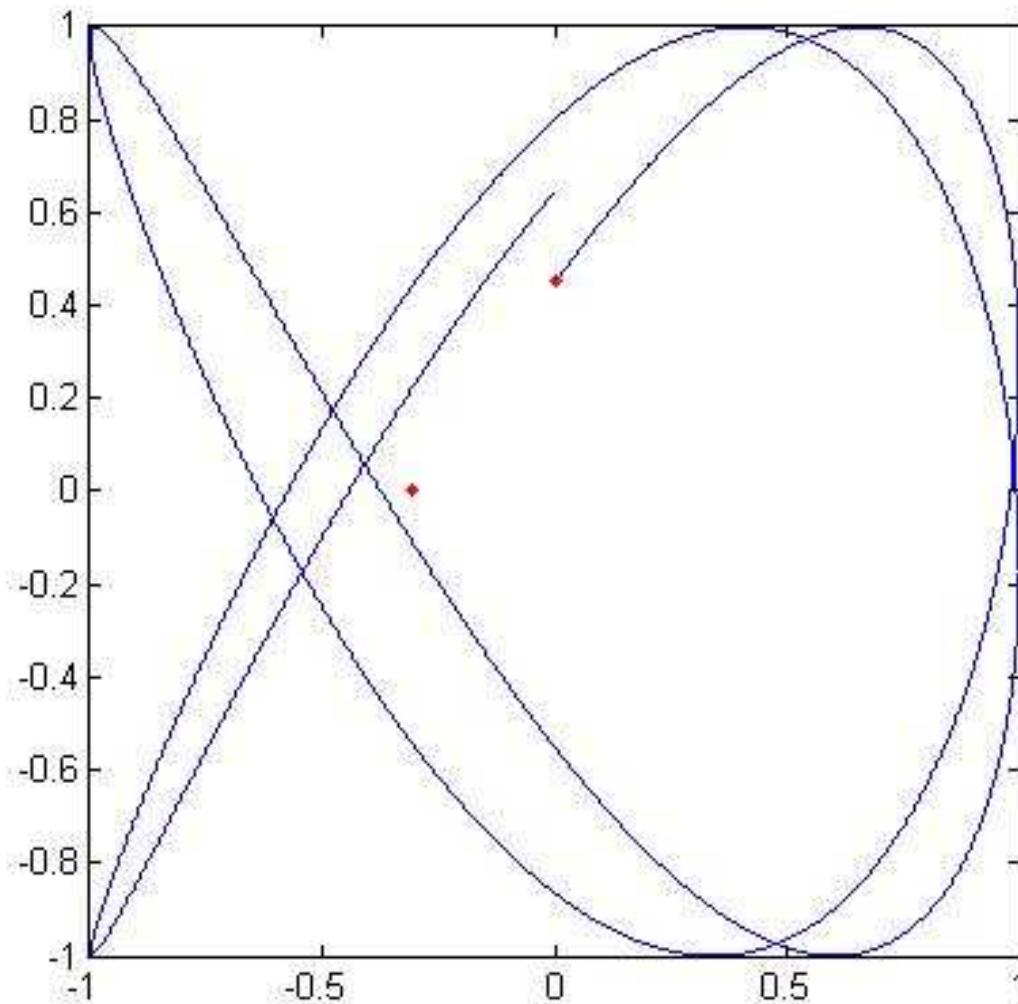
Rotation Speeds



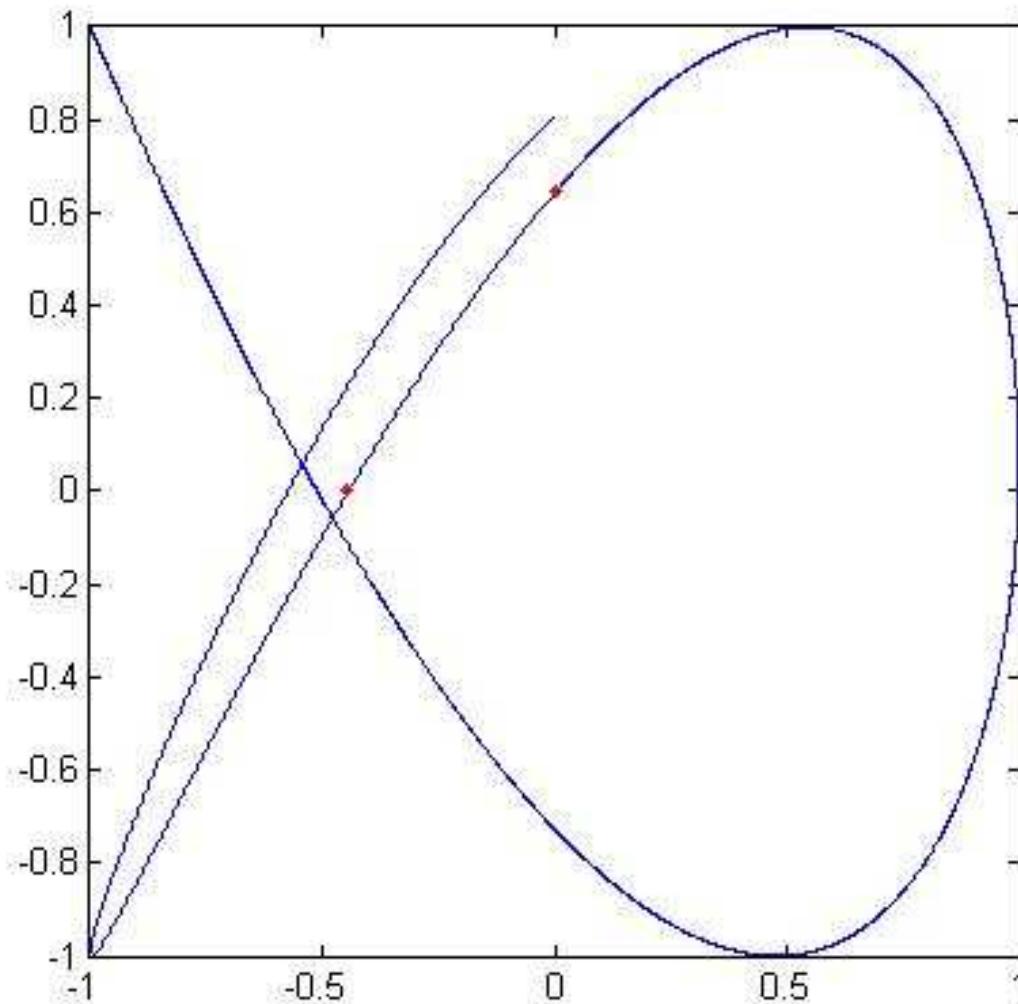
Rotation Speeds



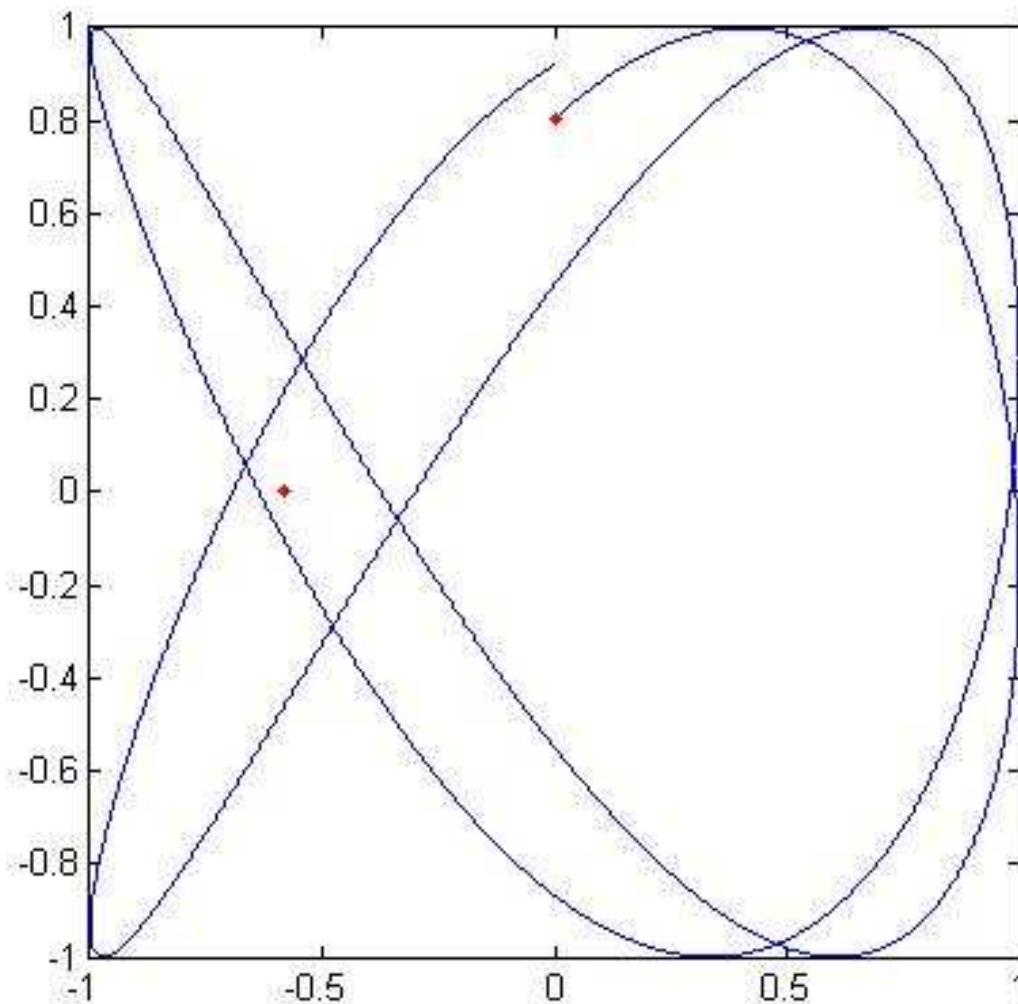
Rotation Speeds



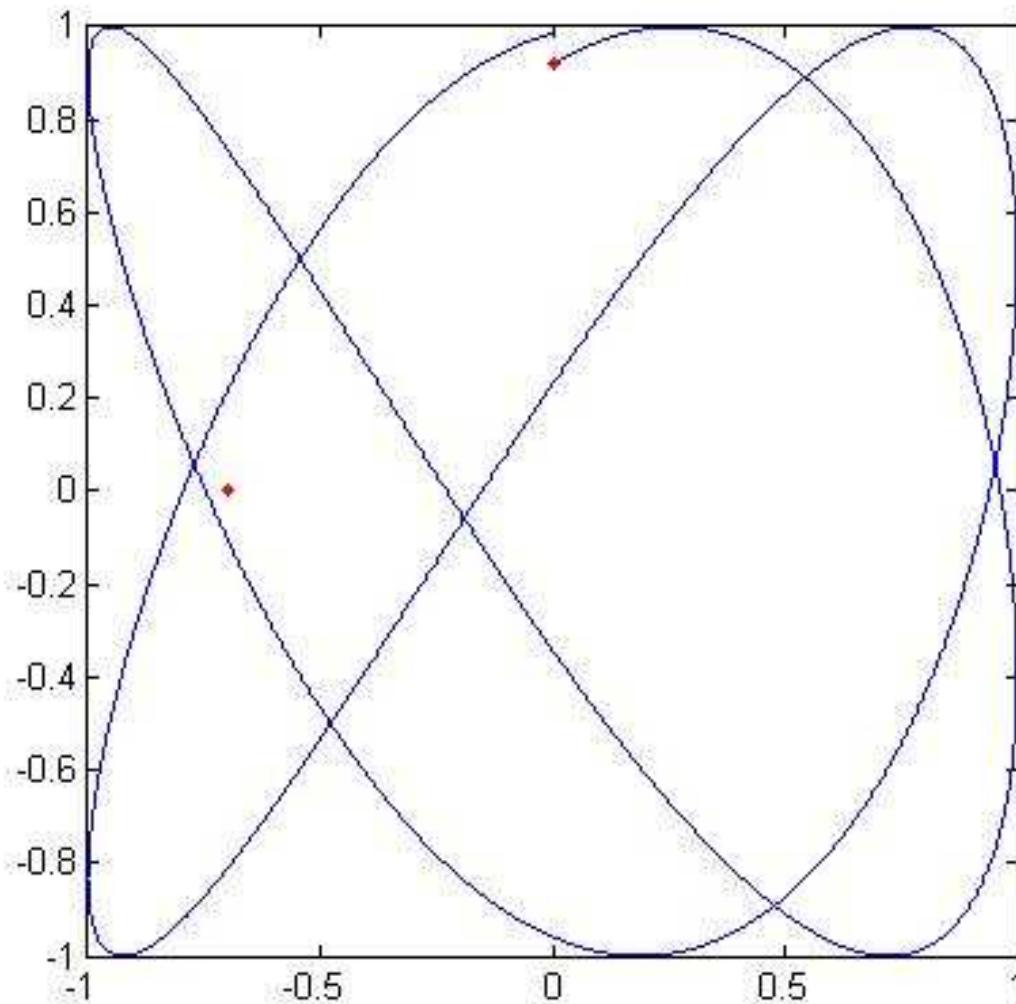
Rotation Speeds



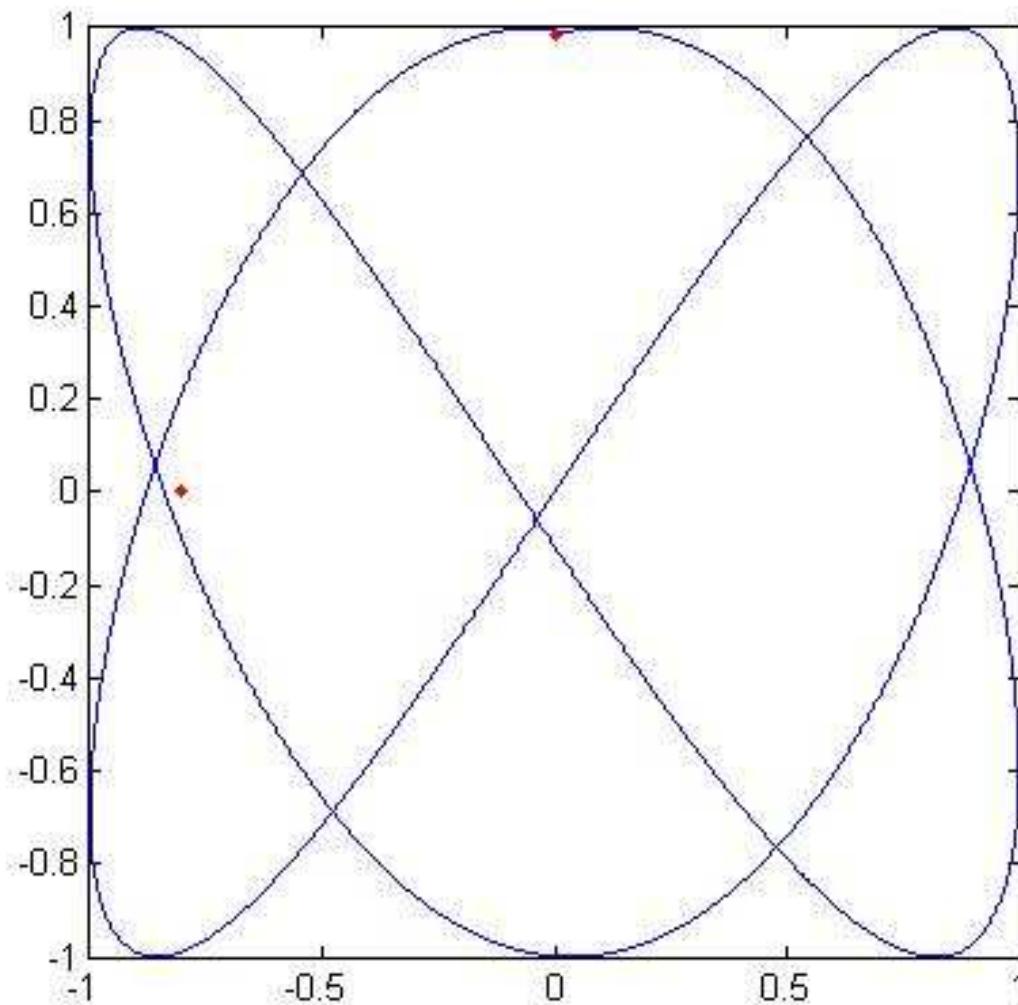
Rotation Speeds



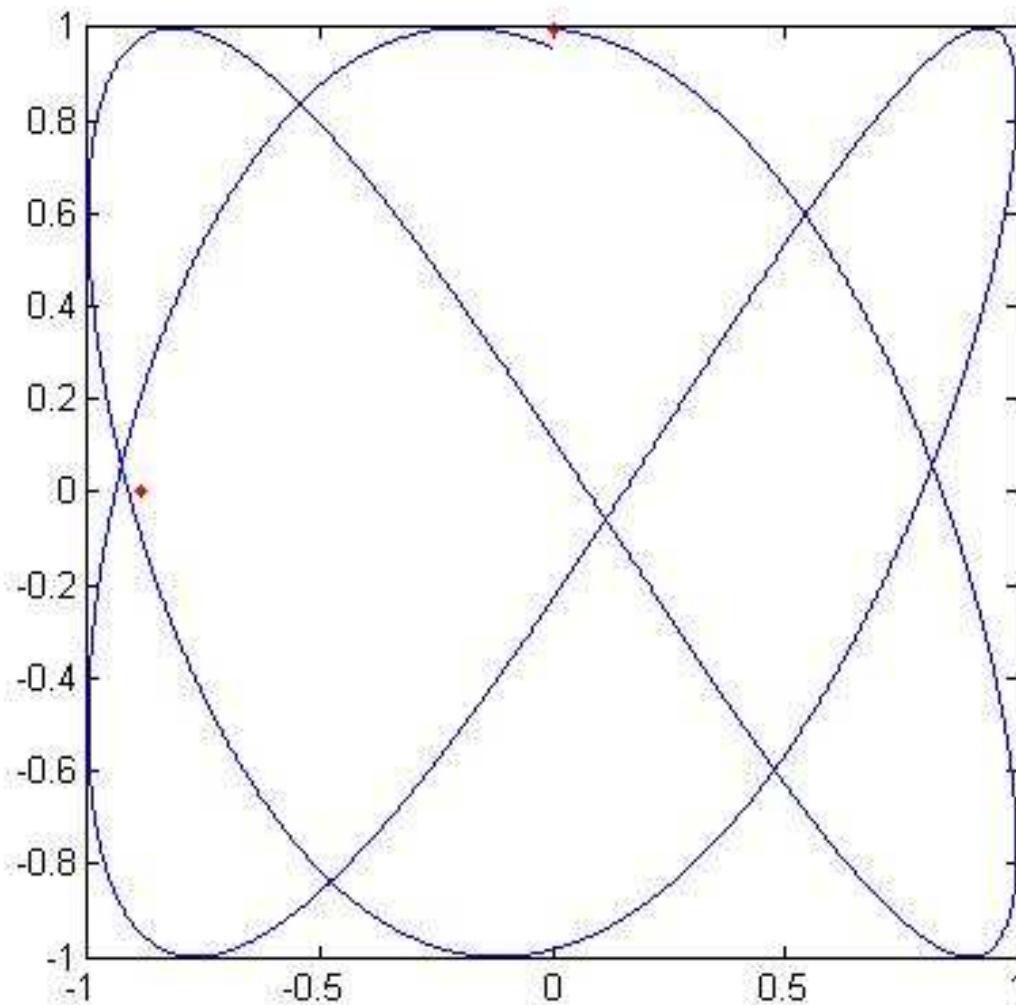
Rotation Speeds



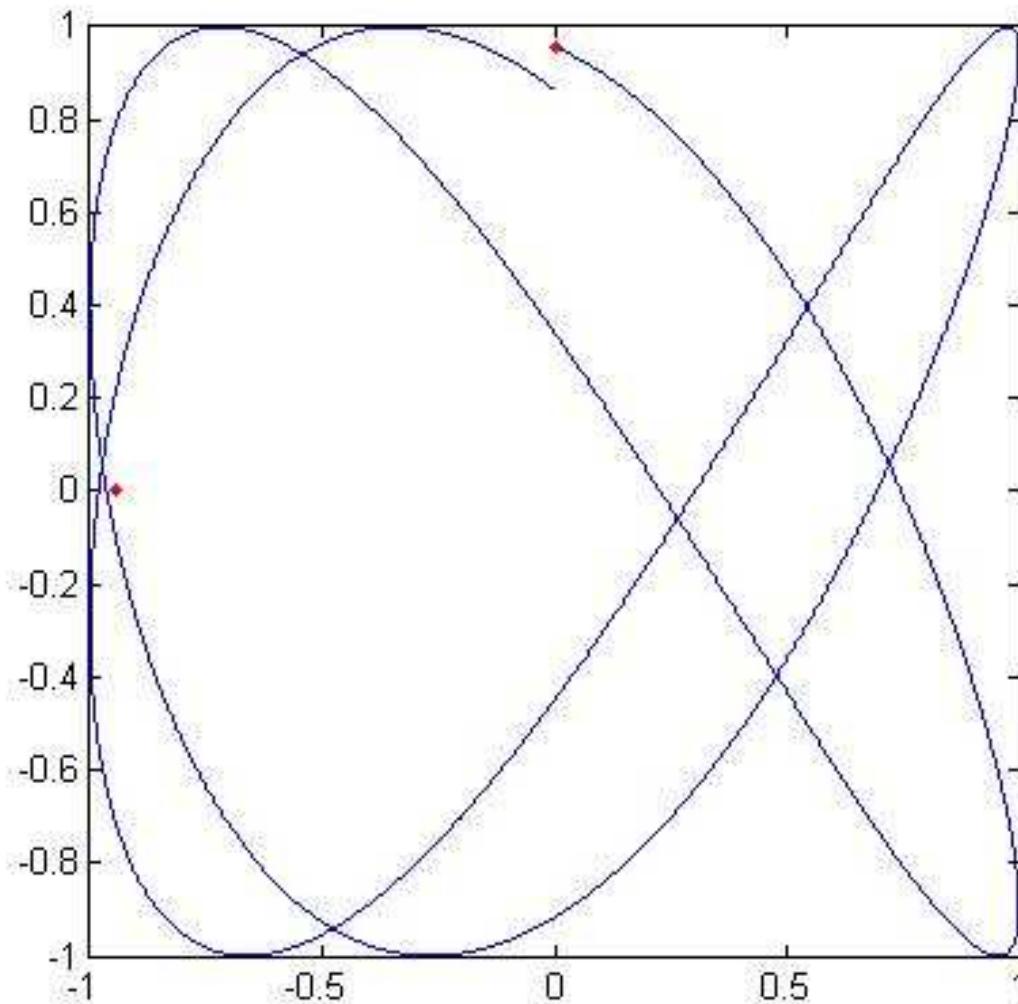
Rotation Speeds



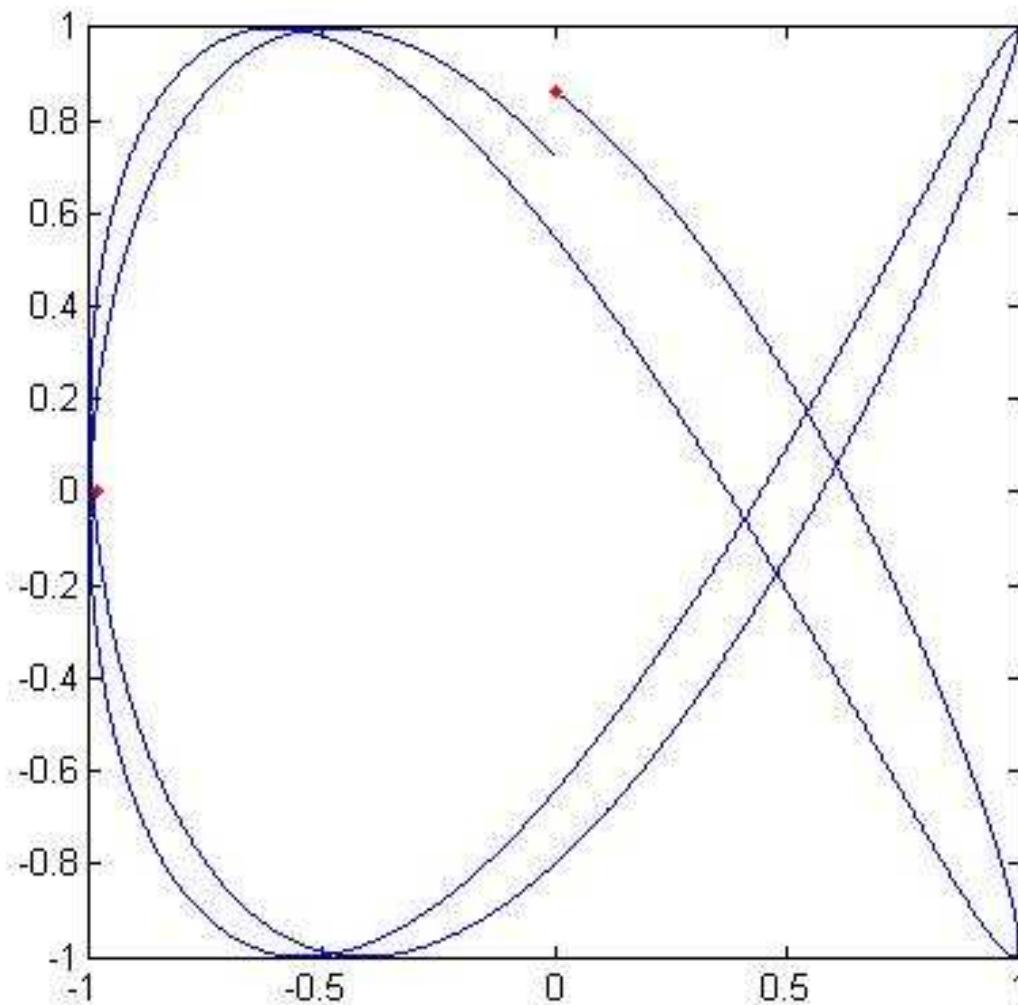
Rotation Speeds



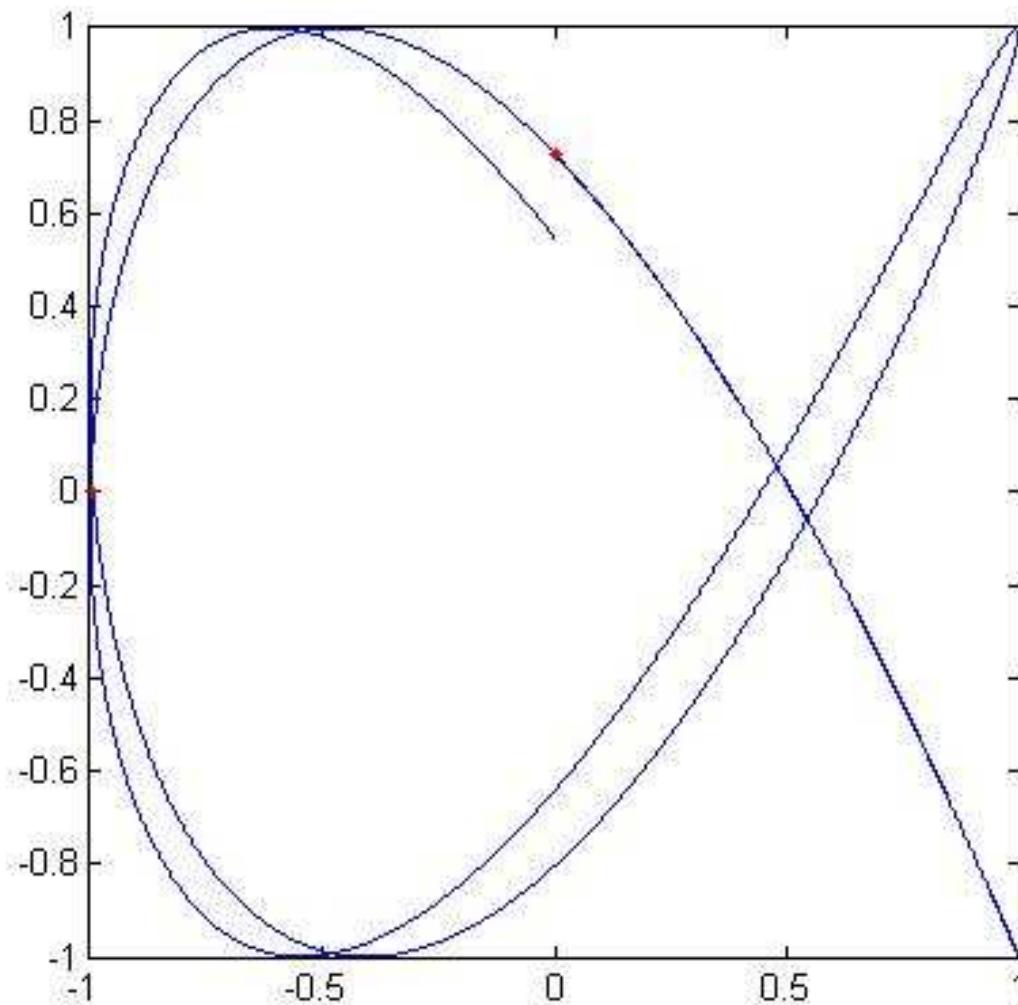
Rotation Speeds



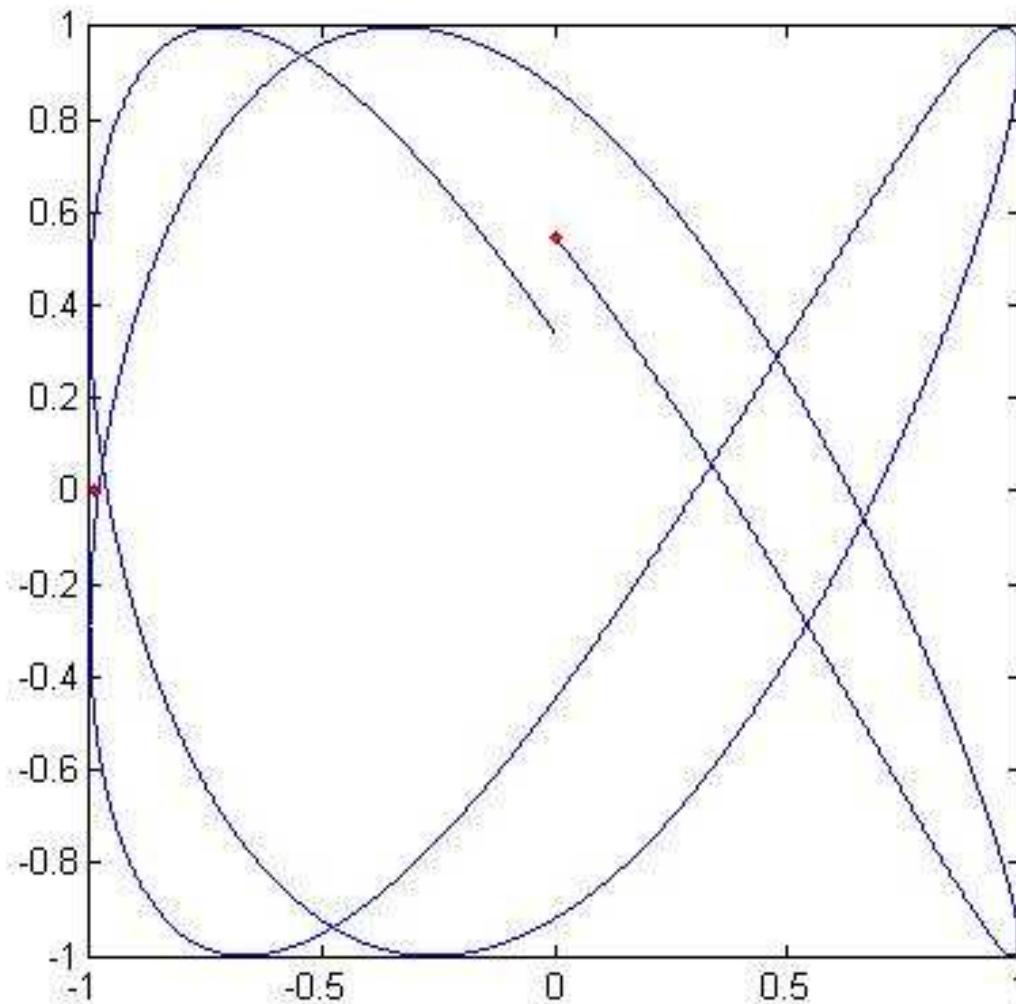
Rotation Speeds



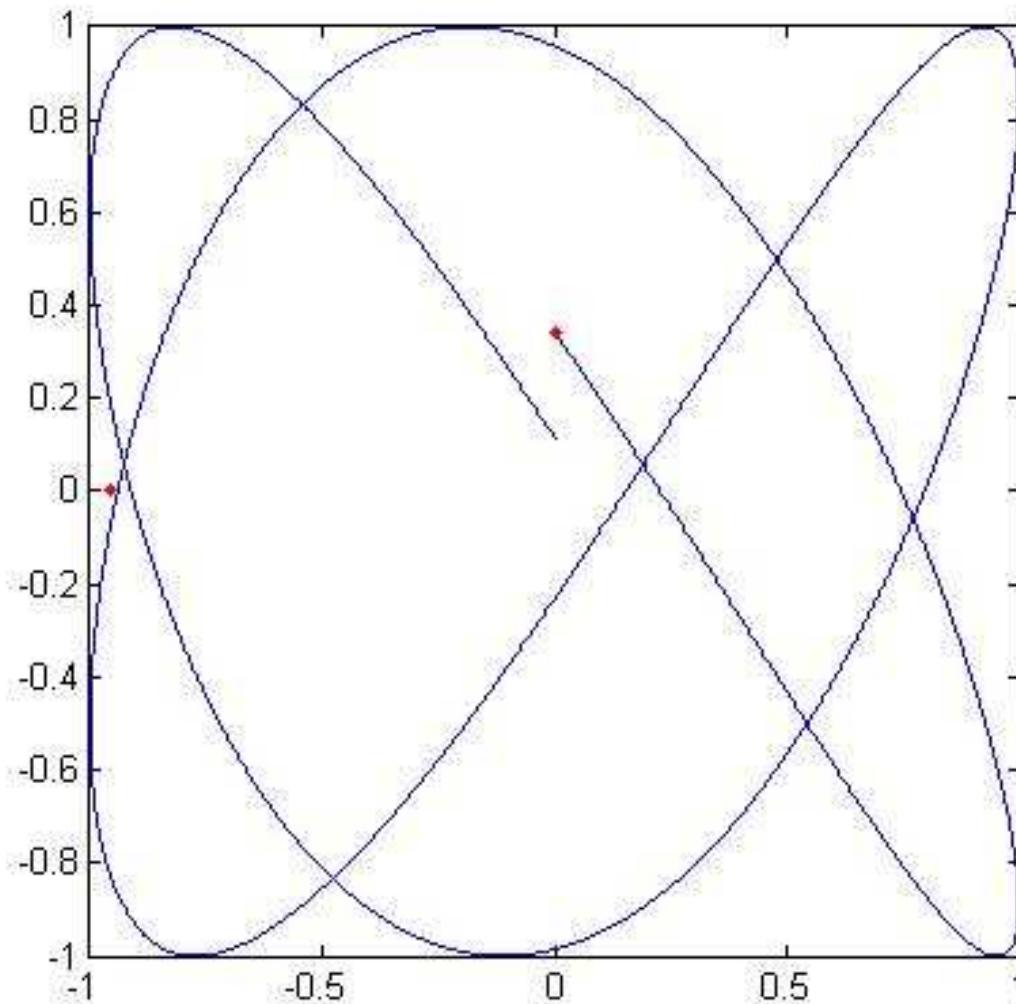
Rotation Speeds



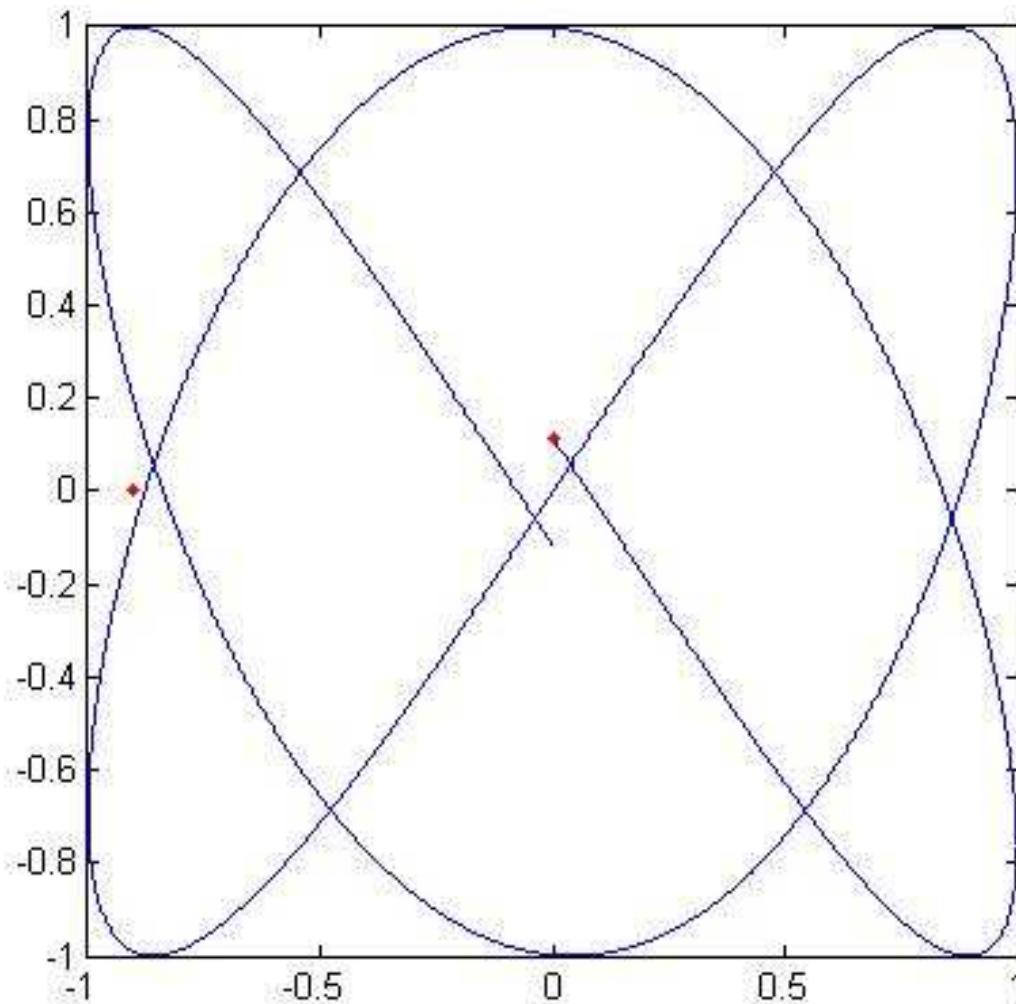
Rotation Speeds



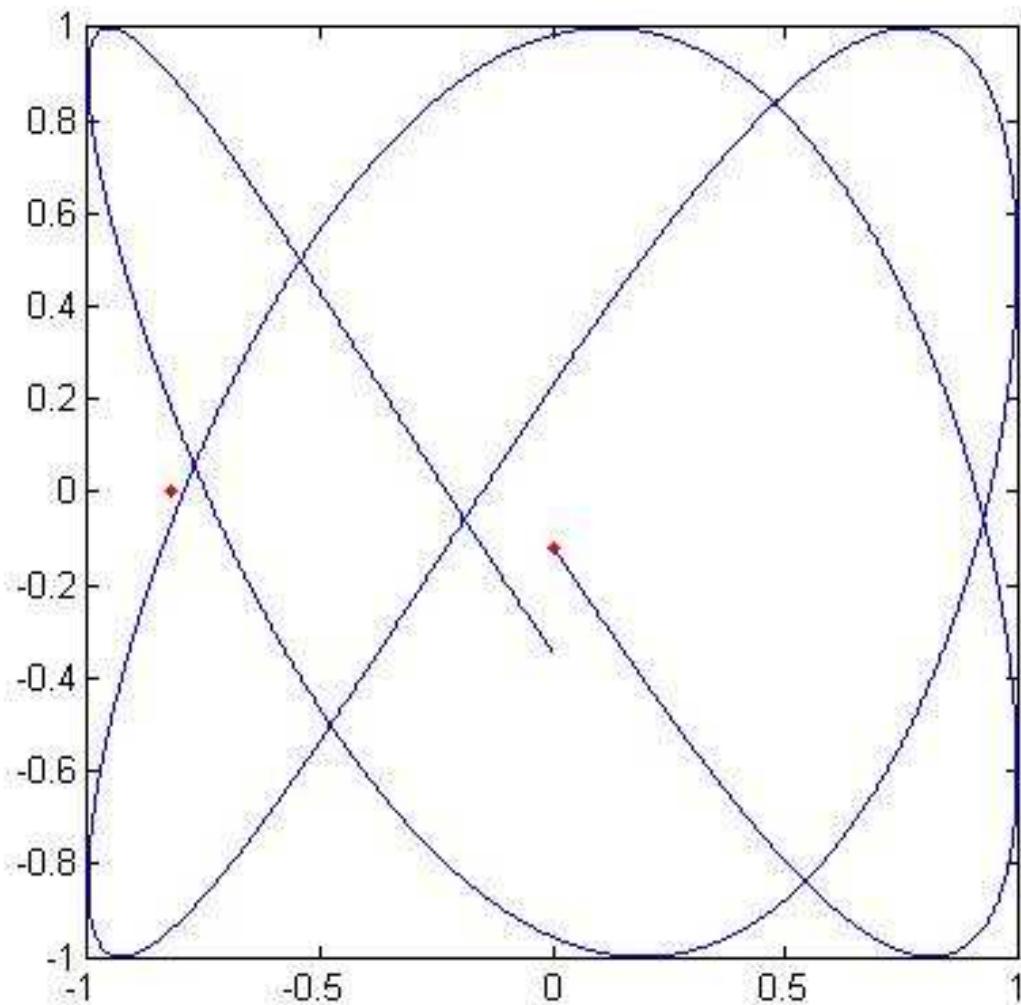
Rotation Speeds



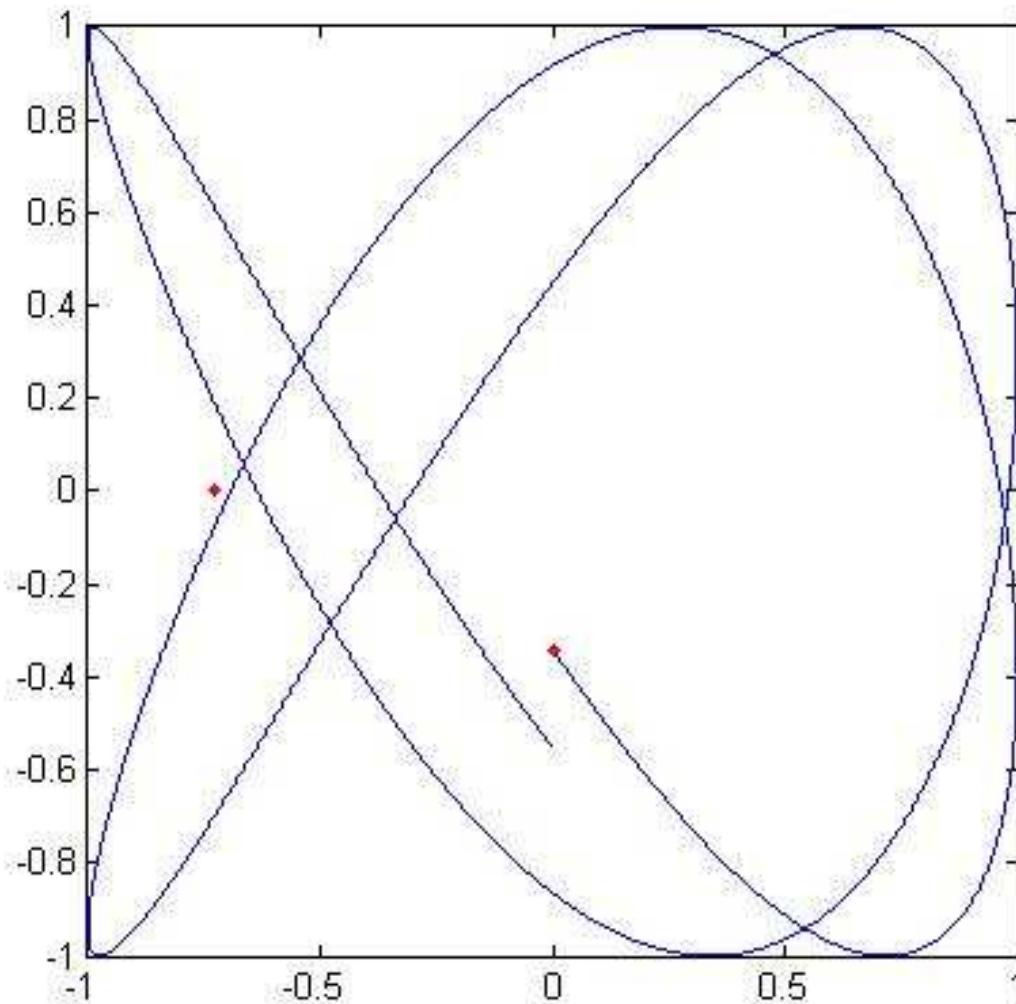
Rotation Speeds



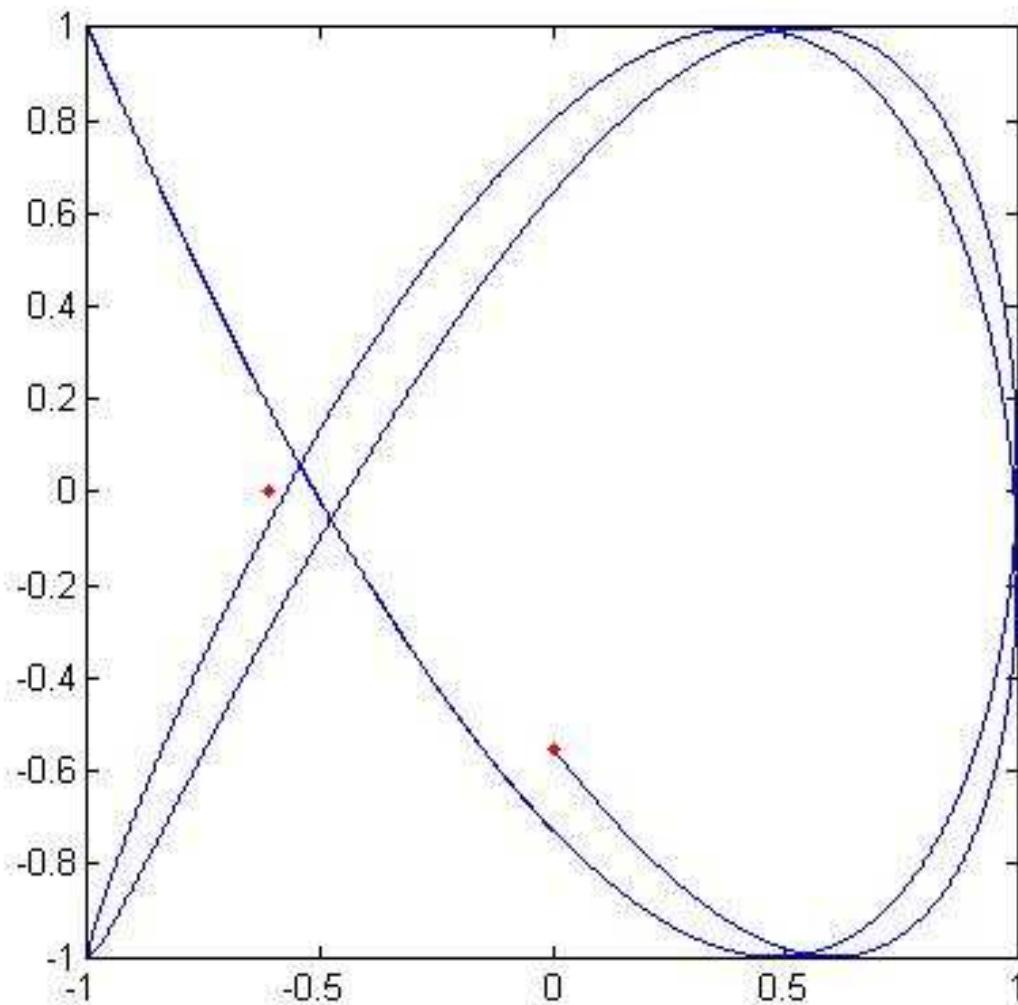
Rotation Speeds



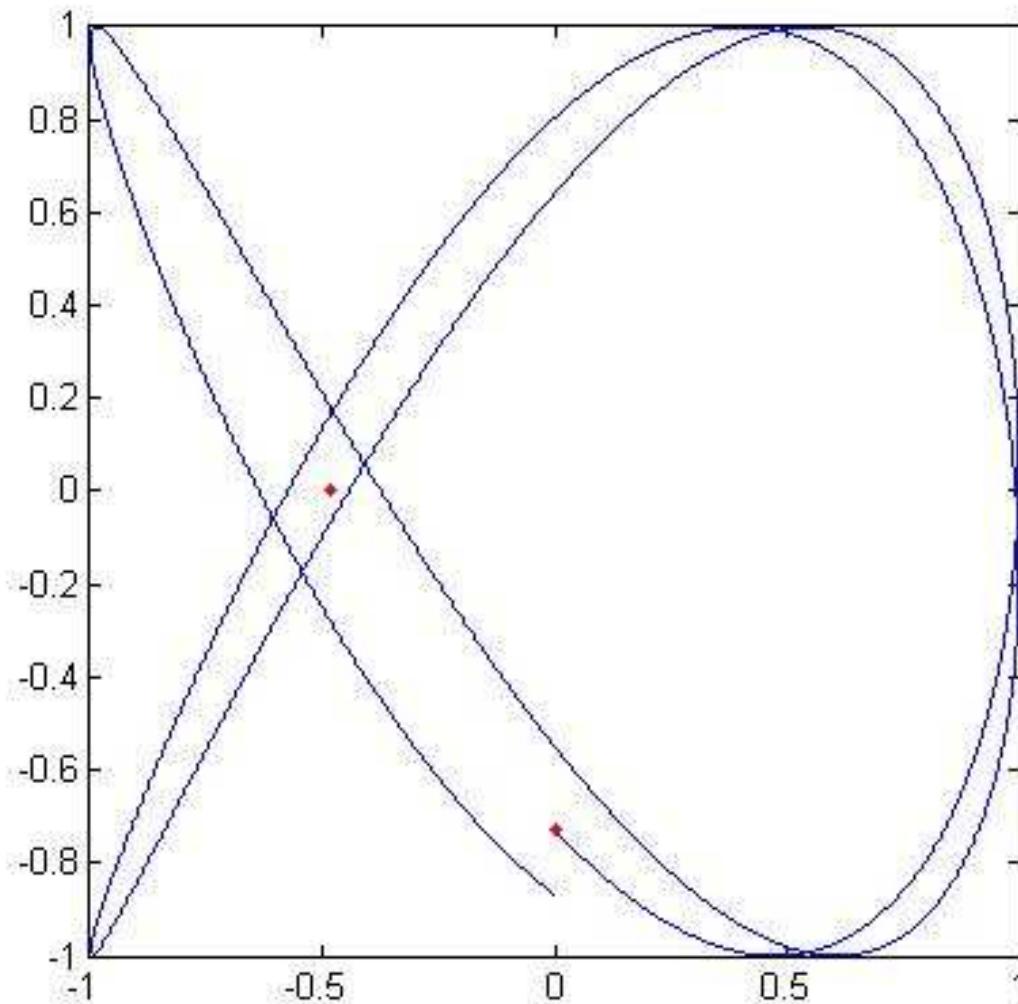
Rotation Speeds



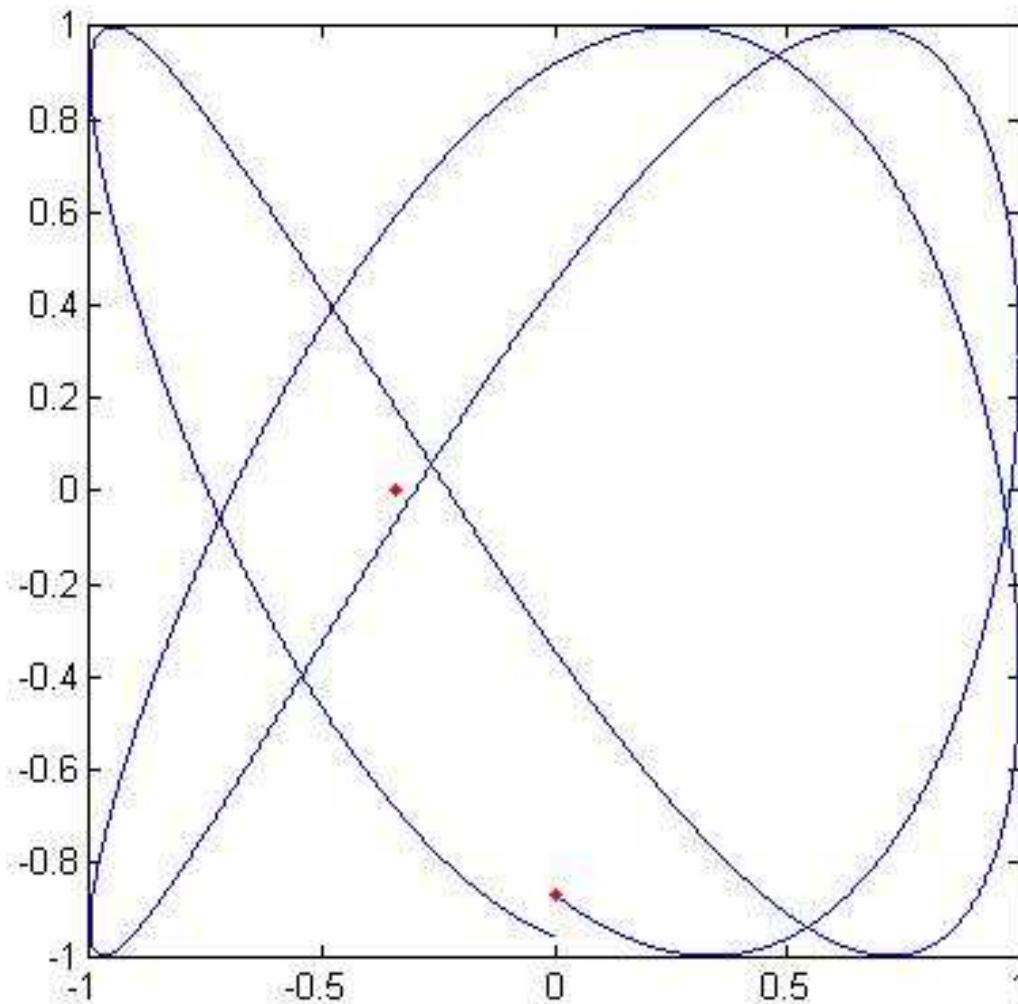
Rotation Speeds



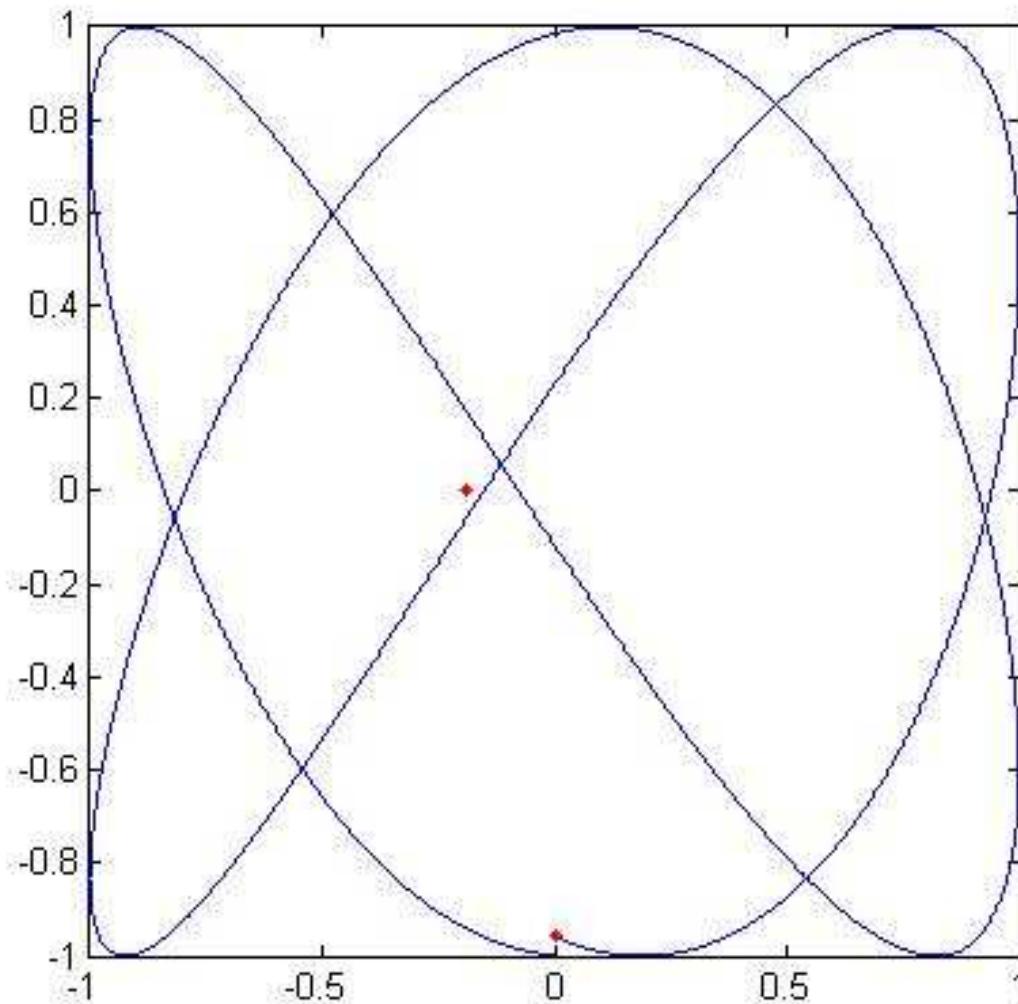
Rotation Speeds



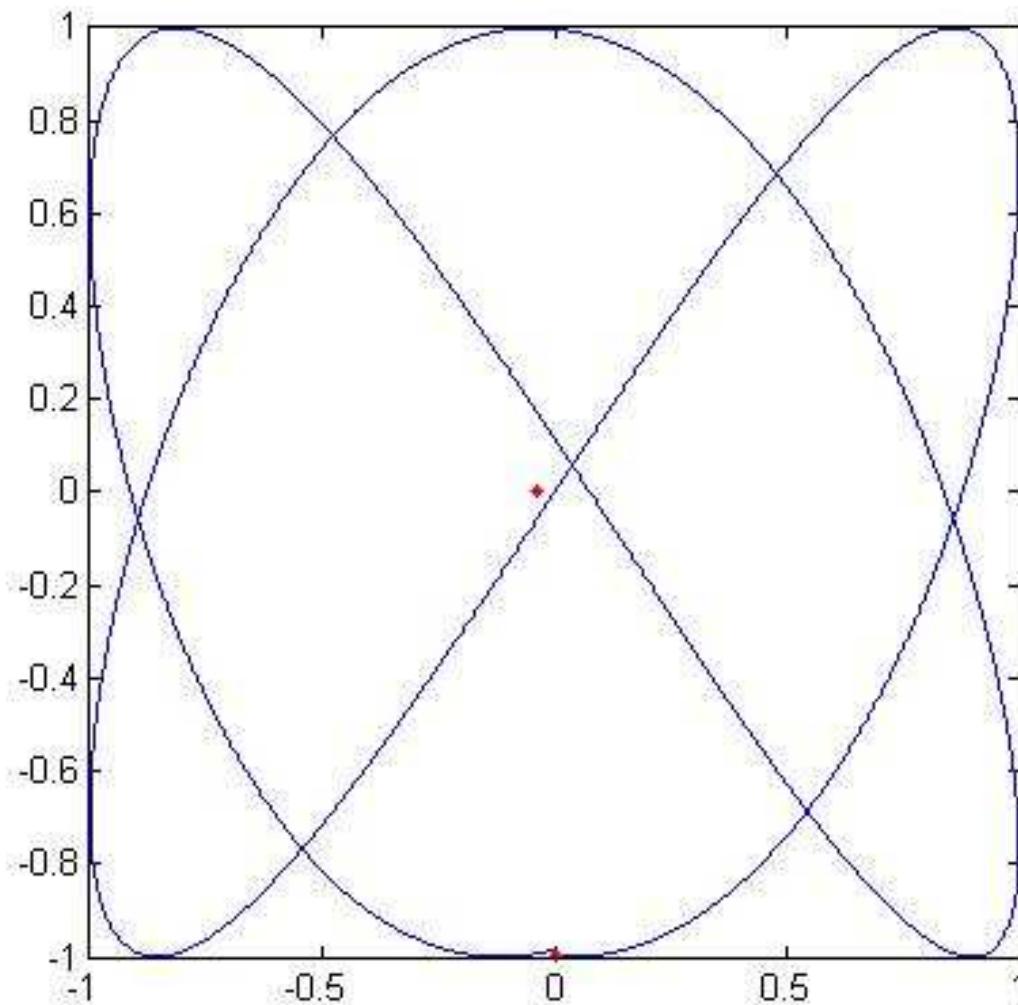
Rotation Speeds



Rotation Speeds



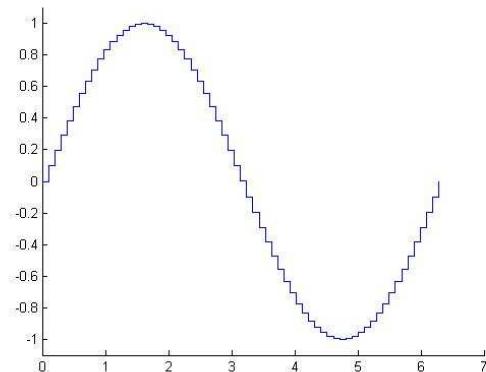
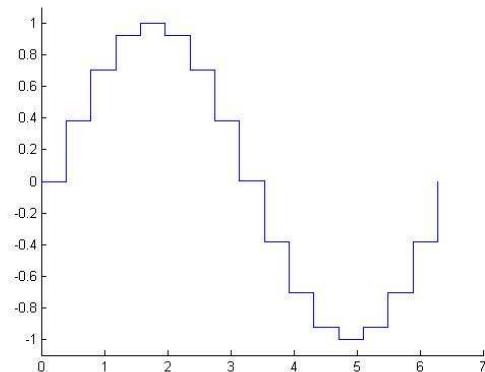
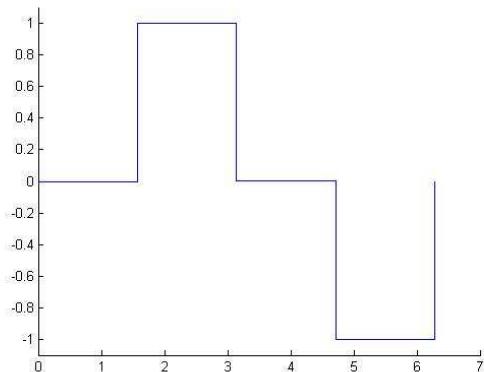
Rotation Speeds



Sine waves

- The microcontroller must make sine waves
- Generate sine waves with a lookup table
- Frequency control must be precise

Sine waves with 4, 16, 64 data points



Timing

- “Clean” sine wave requires more data points per period
- ... which requires a higher interrupt frequency
- Higher interrupt frequencies give less time to execute code between interrupts

Interrupts per Second

- Suppose 180 machine cycles is adequate to service a point calculation.
- Works with 8 bit auto-reload counter.
- The R31JP runs off a 11.0592 MHz clock
- 1 machine cycle equals 12 clock cycles
- Interrupt frequency:

$$11.0592M \frac{\text{clock cycles}}{\text{second}} \div 12 \frac{\text{clock cycles}}{\text{machine cycles}} \div 180 \frac{\text{machine cycles}}{\text{interrupt}} = 5120 \frac{\text{interrupts}}{\text{second}}$$

Table Advance

- Table advance for an 80 Hz wave:

$$256 \frac{\text{table elements}}{\text{period}} \div 5120 \frac{\text{interrupts}}{\text{second}} * 80 \frac{\text{periods}}{\text{second}} = 4 \frac{\text{table elements}}{\text{interrupt}}$$

- At Lazerdillo's maximum frequency the table skip will be 4 elements per interrupt
- Corresponds to 64 data points per period
- Sine waves will be reasonably smooth

Precision

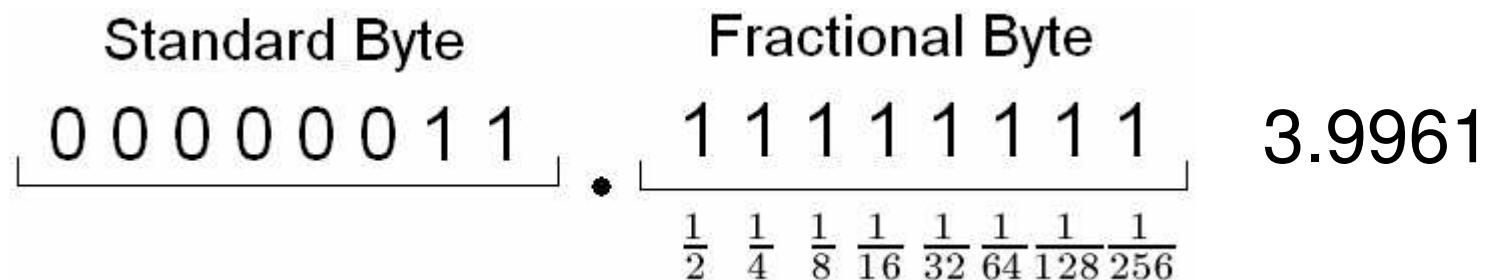
- Precise control of the frequency is needed to make rotating figures
- Previous example: a frequency ratio of 79.9/80 for 10 second rotation period.
- Requires frequency precision to 0.1 Hz
- That's a range of $80/0.1=800$

Table Counter Bytes

- A single byte has a factor of only 256 between the largest and smallest numbers
- Use a second byte to track the table advance
- Second byte provides fractional control over the table advance

Fractional Byte

- What is a fractional byte?



- Decimal positions provide fine control over the table advance
- Double byte table advance above converts to about 79.92 Hz
- Enables slow rotation

Poor 1-byte Table Addressing

- Suppose at each iteration we add the standard “advance” byte of the table advance to the table address and truncate the rest

Table Advance:
3.4 elements per
interrupt

Interrupt	1 Byte Address
0	0
1	3
2	6
3	9

Better 2-byte Table Addressing

- Use two bytes to track our position in the table.
- Do a 16 bit add to keep track of table position.
- Use one “rounded” byte to actually address → the 256-byte table .

Example Table
Advance:

3.4 elements per
interrupt

After 3 interrupts:

10.2

Interrupt	1 Byte Address	2 Byte Address
0	0	0→0
1	3	3.4→3
2	6	6.8→6
3	9	10.2→10

Rotation Revisited

- Lissajous Figures complete a rotation when $\phi(t)$ passes through an entire period
- With our example we have:

$$\frac{a}{b} = \frac{\#03.FFh}{\#04.00h} \approx \frac{79.92Hz}{80Hz}, \phi(t) = \#00.01h * t$$

- Converting to time:

$$256 \frac{\text{table elements}}{\text{rotation}} * 256 \frac{\text{fractional table elements}}{\text{table element}} \div$$

$$5120 \frac{\text{interrupts}}{\text{second}} \div 1 \frac{\text{fractional table element}}{\text{interrupt}} = 12.8 \frac{\text{seconds}}{\text{rotation}}$$

Details of the Rotation Period

$$256 \frac{\text{table elements}}{\text{rotation}} * 256 \frac{\text{fractional table elements}}{\text{table element}} \div$$

$$5120 \frac{\text{interrupts}}{\text{second}} \div 1 \frac{\text{fractional table element}}{\text{interrupt}} = 12.8 \frac{\text{seconds}}{\text{rotation}}$$

$$256 \frac{\text{table elements}}{\text{rotation}}$$

- $\phi(t)$ must pass through all 256 elements of the sine table

$$256 \frac{\text{fractional table elements}}{\text{table element}}$$

- The fractional byte yields 256 fractional table elements per full table element

$$5120 \frac{\text{interrupts}}{\text{second}}$$

- Calculated previously

$$1 \frac{\text{fractional table element}}{\text{interrupt}}$$

- Corresponds to: $\phi(t) = \#00.01h * t$

$$12.8 \frac{\text{seconds}}{\text{rotation}}$$

- Rotation period!

Consider the problem of making a digital voltmeter using the micro and an ADC:

X : Our 8 bit binary input
Y : Our scaled integer input

Last time:

$$Y = \frac{500}{256} X \quad 0 \leq X \leq 255$$

This operation involved careful multiplication (8 bit X times 16 bit 500) and a shift to divide by 256.

Suppose we want to average 100 measurements to computer the output Y. Now, X must hold a two-byte sum of 100 one-byte samples:

$$Y = \frac{500}{256 * 100} X \quad 0 \leq X \leq 25500$$

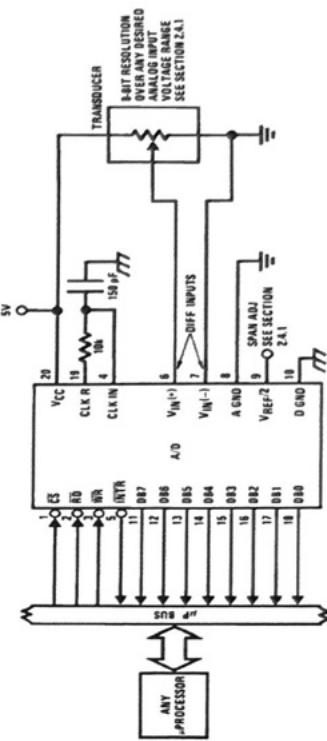
or

$$Y = \frac{1280}{65536} X \quad 0 \leq X \leq 25500$$

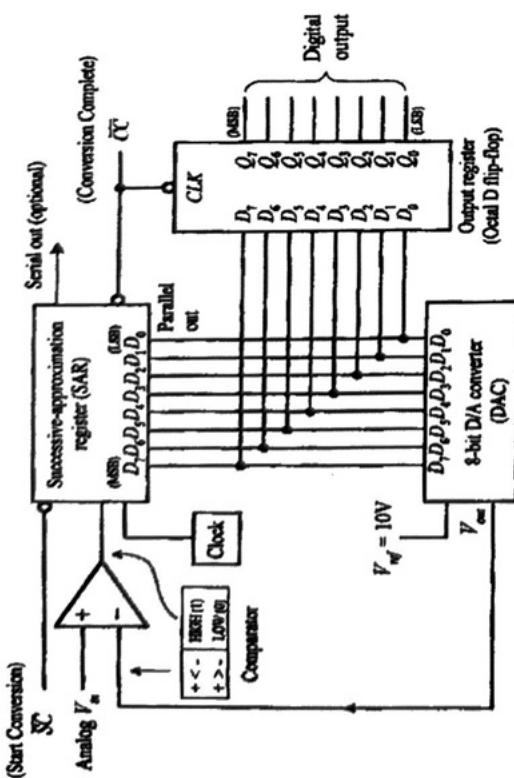
1

DSN00671-1

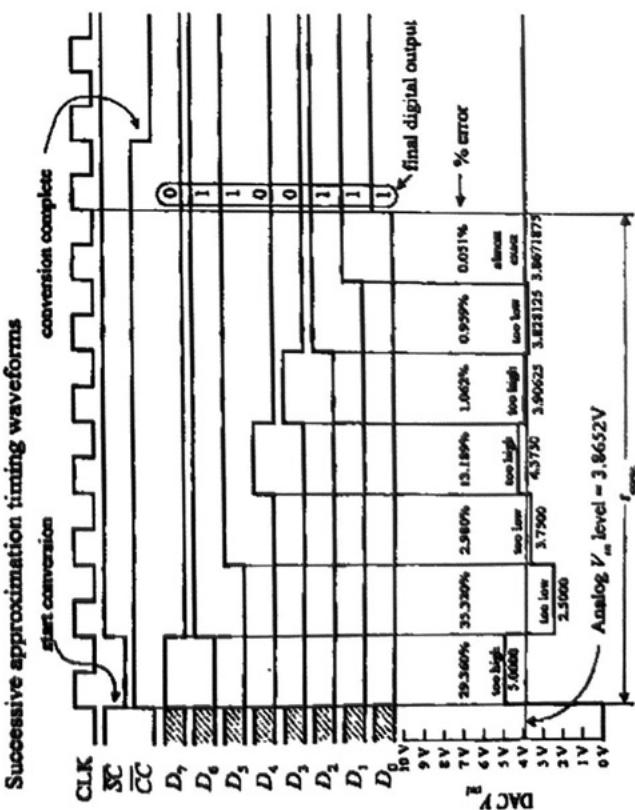
ADC0801/ADC0802/ADC0803/ADC0804/ADC0805 8-Bit μP Compatible A/D Converters



2



3



4

Binary	Unsigned	2's comp	offset	sign/mag	BCD	Gray
0000	0	0	-7	0	0	0000
0001	1	1	-6	1	1	0001
0010	2	2	-5	2	2	0011
0011	3	3	-4	3	3	0010
0100	4	4	-3	4	4	0110
0101	5	5	-2	5	5	0111
0110	6	6	-1	6	6	0101
0111	7	7	0	7	7	0100
1000	8	-8	1	-0	8	1100
1001	9	-7	2	-1	9	1101
1010	10	-6	3	-2		1111
1011	11	-5	4	-3		1110
1100	12	-4	5	-4		1010
1101	13	-3	6	-5		1011
1110	14	-2	7	-6		1001
1111	15	-1	8	-7		1000

- For non-negative numbers, unsigned binary is fine for all operations. What for overflow. Popular for address arithmetic.
- For addition/subtraction of signed numbers, 2's-complement is helpful.
- For multiplication/division of signed numbers, sign/magnitude format is often convenient.
- Other formats pop up in different applications. Some ADC and DAC chips use offset notation. Position encoders and other noise sensitive sensors sometimes use Gray coding.
- As we examine code today, note that registers names like X, XH, XL, Y, and XS stand for registers and bits YOU pick in the memory space!

5

```

TC2MS8
MOV A, XT      ;Register XT hold signed 2com byte
JB acc.7, Xneg ;if bit 7 is 1, x is negative
MOV XA, XT
CLR XS
RET

XNEG:
CPL A          ;x is negative, so find the
INC A          ;positive version
MOV XA, A       ;save the magnitude
SETB XS        ;set the sign bit
RET

How do we multiply an 8 bit number times a 16 bit number?

MUL8_16      ; byte in X, word in YH:YL. You pick these registers,
              ; e.g., R1, R2, R3, etc
MOV a, X       ; load X in accumulator
MOV b, YL     ; load YL in B register
MUL A, B
MOV Z0, A      ; Z0 is register of your choice. Put low byte in Z0
PUSH B        ; push the result high byte. We'll add this to X*YH
MOV A, X       ; load x into accumulator again
MOV B, YH     ; put YH in B
MUL A, B      ; now compute X*YH
POP 0          ; get high byte of X*YL into R0
ADD A, R0      ; add low byte of X*YH to high byte of X*YL
                ; NOTE! Carry may have been set!
MOV Z1, A      ; Put the sum in Z1
CLRA          ; Remember, B has high byte of X*YH. Clear A
ADDC A,B      ; Add C to high byte of X*YH. Result to A
MOV Z2, A      ; Save final result in Z2.
Ret           ; "Solution" (3 bytes) in Z0, Z1, Z2. Z2 is MSB

```

6

How do we multiply two 16 bit numbers (4 byte result)?

X = XH:XL, Y = YH:YL

Algorithm:

- First, multiply XL times Y. Yields ZL2, ZL1, ZL0 (msb to lsb).
- Second, multiply XH times Y. Yields ZH2, ZH1, XH0
- Compute four result bytes: Z3, Z2, Z1, Z0:

```

Z0 = ZL0
Z1 = ZL1 + ZH0 ; carry forward
Z2 = ZL2 + ZH1 ; carry forward
Z3 = ZH2

```

Decimal Adjust: A trick for performing decimal additions directly:

Example: Let 56d be represented by 56h!

Also, let 08d be represented by 08h. Try this:

```

MOV A, #56h
MOV R0, #08h      This program leaves 5Eh in the
ADD A, R0          accumulator.

```

Now, if you add the command: DA A, the accumulator will contain 64h. The result of adding 56d and 8d if we interpret the result as two decimal digits! Neat! Essentially adds 5 to 59, or more accurate, 06h to the low nibble. See the INTEL spec book for more details.

How do we multiply two 16 bit numbers (4 bit result)?

```

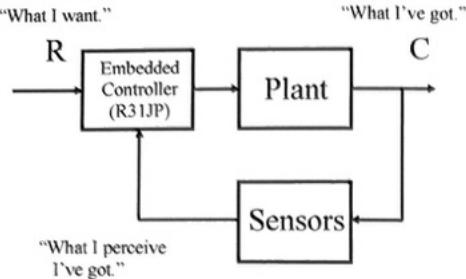
MUL16:          ;word in XH XL, word in YH YL. (XL=X) MOV Z2, A; store 3rd res byte in Z2
LCALL MUL8_16
MOV A, Z2        CLR A
PUSH ACC        ADDC A,B ;ZH2+C > A
MOV A, Z1        MOV Z3, A; store 3rd res byte in Z2
PUSH ACC        ret      ;done!
MOV A, Z0        ;four result bytes
PUSH ACC        ;in Z0, Z1, Z2, Z3
MOV XL, XH        ;(lsb to msb)
LCALL MUL 8_16 ;mul XH times Y
MOV B, Z0        ;save ZH0 in B
POP Z0          ;recall ZL0
POP ACC          ;recover ZL1 in A
ADD A,B          ;add ZH0 and ZL1
MOV B, Z1        ;save ZH1 in B
MOV Z1, A        ;ZH0+ZL1 > Z1
POP ACC          ;recover ZL2 in A
ADDC A, B        ;ZL2+ZH1+C > A
MOV B, Z2        ;save ZH2 in B

```

7

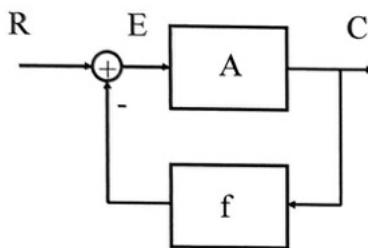
8

Often, we may find an embedded controller in an over all system structure that looks like this:



This is a feedback or “error-driven” structure. Let’s try to understand how it might behave! We’ll surely depend on the characteristics of all 3 blocks.

Before we get to the digital controller, let’s look at an analog example:



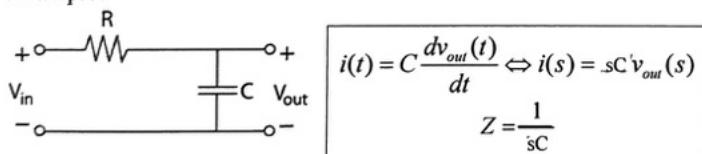
1

Aha! But what do we do if one of the blocks (or more) is not simply a gain, but rather a “dynamic” block described by a differential equation?

Answer: Find a way to “pretend” that the differential equation is an algebraic equation!

$$\text{Laplace Operator Calculus: } \frac{d}{dt} \Leftrightarrow s$$

Example:



Differential Equation:

$$v_{in} = iR + v_{out} = RC \frac{dv_{out}}{dt} + v_{out}$$

Laplace: “impedance” divider:

$$\frac{v_{out}(s)}{v_{in}}(s) = \frac{\frac{1}{sC}}{R + \frac{1}{sC}} = \frac{1}{sRC + 1}$$

$$\text{or } v_{out} \cdot [sRC + 1] = v_{in} \Leftrightarrow RC \frac{dv_{out}}{dt} + v_{out} = v_{in}$$

A little arithmetic:

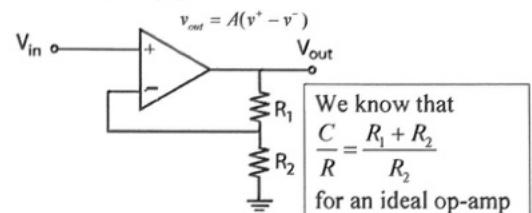
$$E = R - fC$$

$$C = AE = A(R - fC)$$

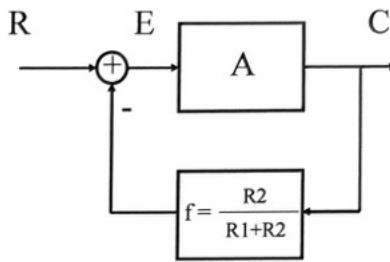
Collecting terms:

$$C + AfC = AR \text{ or } \frac{C}{R} = \frac{A}{1 + Af} \leftarrow \text{BLACK'S FORMULA}$$

Example: Non-inverting op-amp gain block:



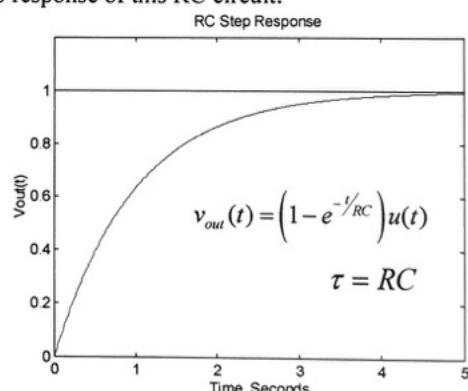
In block diagram form:



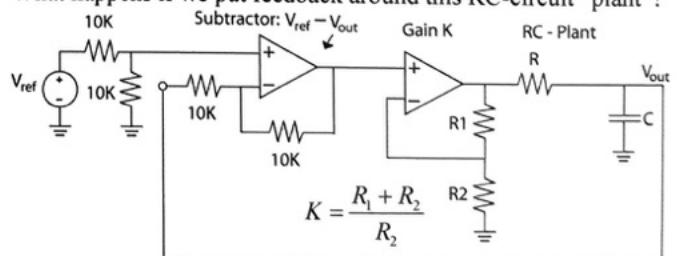
$$\text{According to Black's formula: } \frac{C}{R} = \frac{A}{1 + A \frac{R_2}{R_1 + R_2}} \rightarrow \frac{R_1 + R_2}{R_2} \text{ as } A \rightarrow \infty$$

2

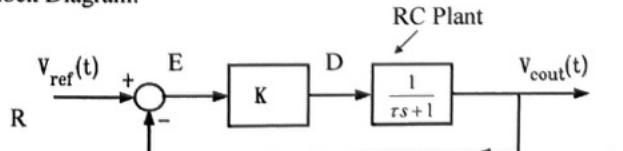
Unit step response of this RC circuit:



What happens if we put feedback around this RC-circuit “plant”?



Block Diagram:



4

Apply Black's formula: $A = \frac{K}{\tau s + 1}$ $f = 1$

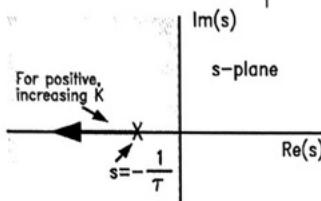
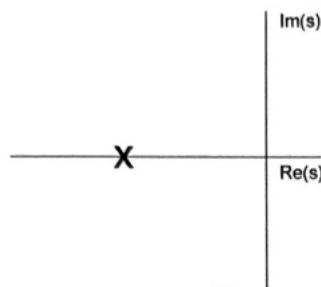
$$\frac{C}{R} = \frac{\frac{K}{\tau s + 1}}{1 + \frac{K}{\tau s + 1}} = \frac{K}{\tau s + 1 + K} = \frac{K}{\frac{\tau}{1+K}s + 1}$$

Pole-zero plot:

ORIGINAL RC CIRCUIT:
 Recall Solution:
 $v_{out}(t) = C = \left[1 - e^{-\frac{t}{RC}} \right] u(t)$
 "natural frequency"
 $s = -\frac{1}{RC}$

$$v_{out} = \frac{K}{1+K} \cdot \left[1 - e^{-\frac{t}{\tau(1+K)}} \right] u(t)$$

starts here near small $K \sim 0$



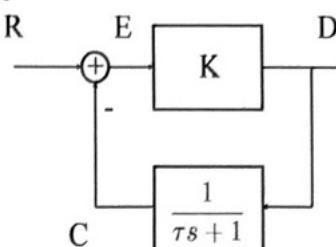
Always stable: steady state error, transient response improve as K gets large.

5

Why does the overall system appear to get "faster" if we use feedback with high gain?

Let's see how the loop "kicks" the plant.

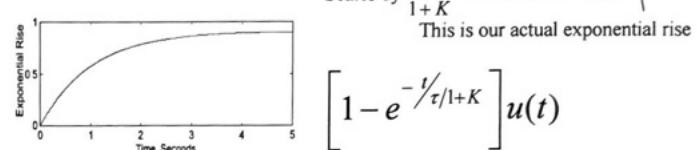
That is what's $\frac{D}{R}(s)$?



$$\frac{D}{R}(s) = \frac{K}{1 + \frac{K}{\tau s + 1}} = \frac{K}{\frac{s\tau}{1+K} + 1} = \frac{K}{1+K} \cdot \frac{(\tau s + 1)}{1+K} \cdot \frac{1}{\frac{\tau}{1+K}s + 1}$$

Takes a derivative and scales by tau

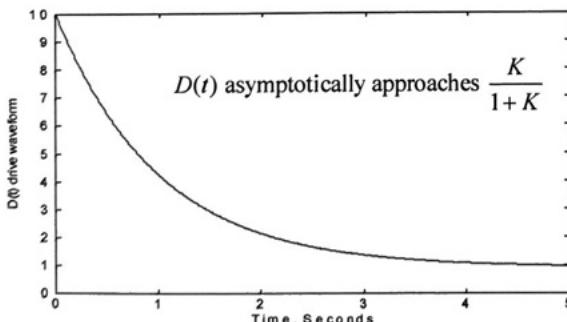
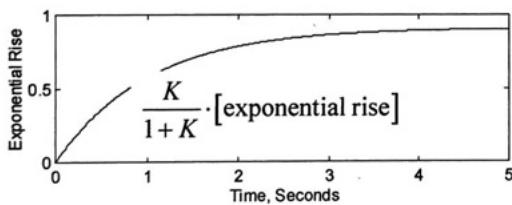
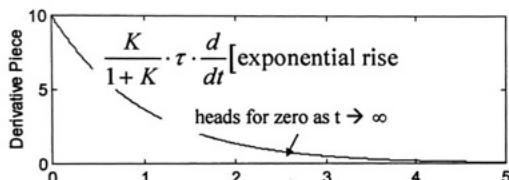
Here is the exponential rise:



$$\left[1 - e^{-\frac{t}{\tau(1+K)}} \right] u(t)$$

So $D(t)$ is the sum of two pieces:

6



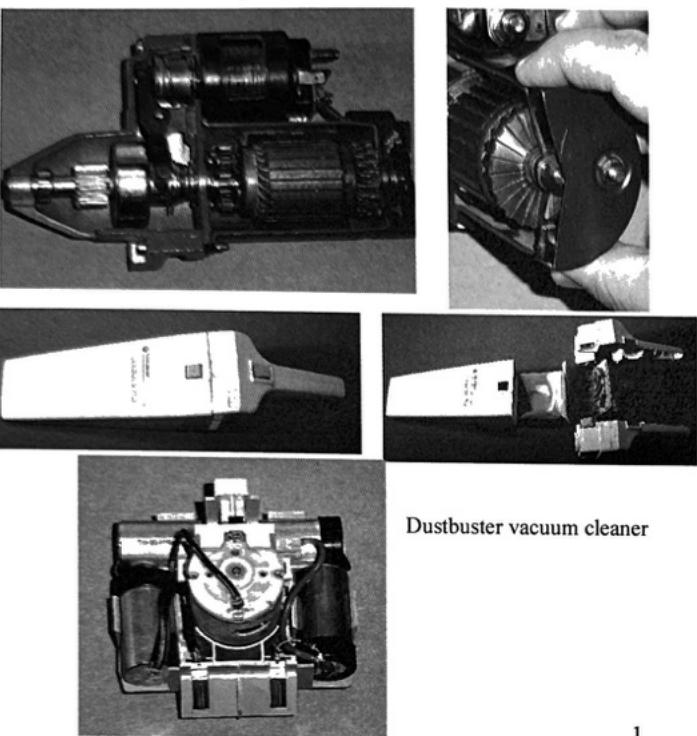
Initially, it "kicks" the RC plant very hard (relative to steady state) to get it "moving"...

7

One "plant" we'll work with: The DC motor.

DC Motors show up in lots of important commercial and industrial systems. Some examples:

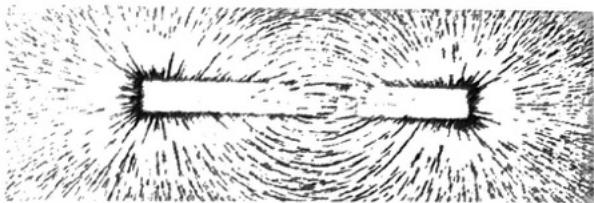
Starter motor in your car: (cut to reveal internals)



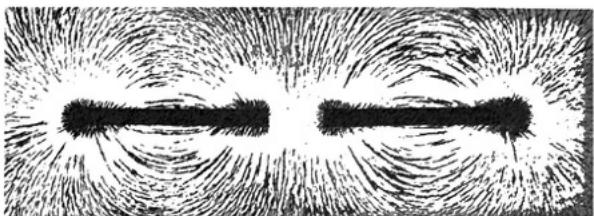
1

Magnetic fields exposed by iron-filings around bar magnets:

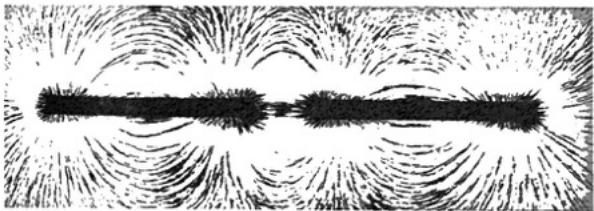
A single bar magnet:



Identical poles facing:

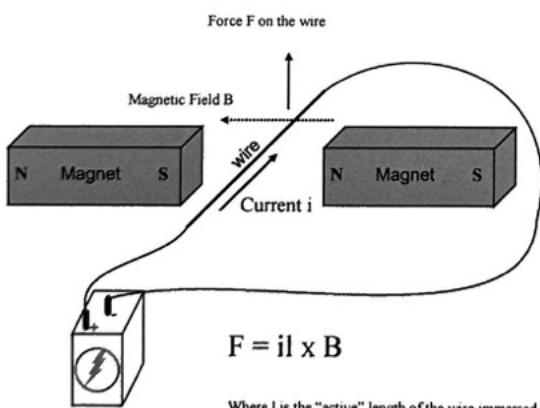


Opposite poles facing:



2

Force on a current-carrying wire immersed in a magnetic field:
(Lorentz Force)



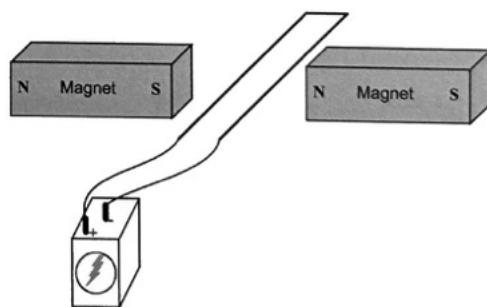
What is the direction of the force acting on this wire, which is carrying current "into" the page (indicated by the X on the wire)?



3

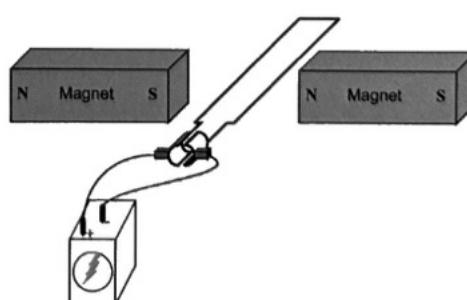
In the experiment shown below, which way will the wire rotate?

Assuming the battery never runs out, will the coil rotate forever?



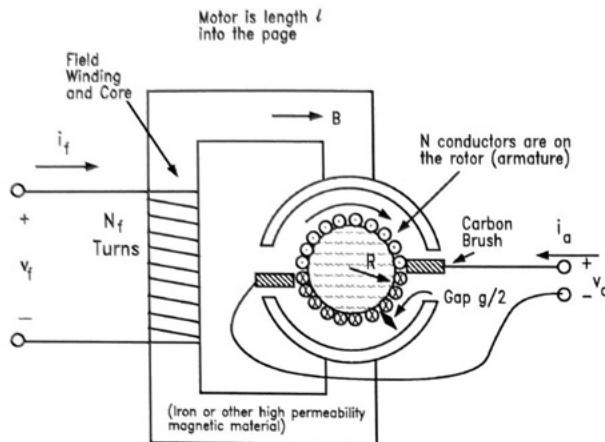
In the experiment shown below, which way will the wire rotate?

Assuming the battery never runs out, will the coil rotate forever?



4

Cartoon of a real DC motor with a “wound” field winding. The field winding could also be replaced with permanent magnets.



If the electromagnetic produces a constant magnetic field B , or if we are using permanent magnets, it's convenient to define a “motor constant” K :

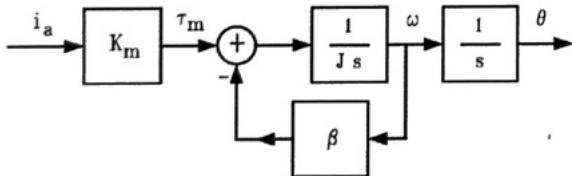
$$K = RNLB$$

Torque produced by the motor is equal to K times the armature current.

5

6

Let's practice making and manipulating block diagrams while we learn how the DC motor works: Start by examining a current source electrical drive (in these block diagrams, $K_m = K$):



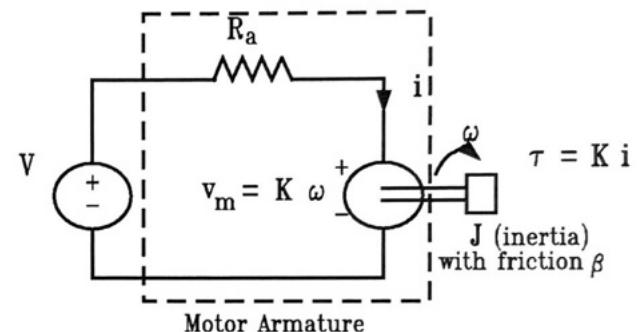
Remember that “friction” is represented by β . If we drive this system with constant current, how well will it maintain a constant speed in the face of disturbance, e.g., a sudden change in the friction?

This drive has no ability to reject disturbances or changes in the “plant” (motor), e.g., a change in friction!

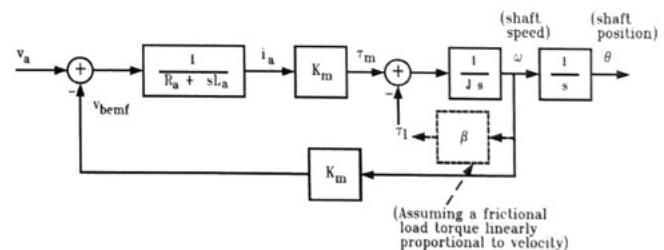
Lenz's Law: Moving a wire in a magnetic field induces a voltage across a wire.

This means that a DC motor can be used as a generator if we manually turn the shaft!

A circuit model for the DC motor driven by a voltage source:



What happens if we try driving the motor with a voltage source instead?



Now, if we drive with a constant input voltage V_a , and we have small armature resistance R_a , then, in steady state, the motor speed is relatively insensitive to changes in friction!

Current injected into the DC motor controls shaft TORQUE.

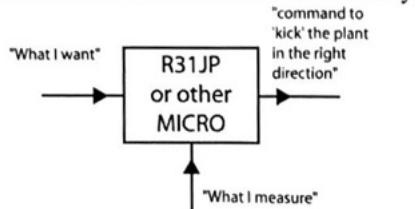
Voltage applied to the DC motor roughly controls shaft SPEED.

7

8

Remember our goal:

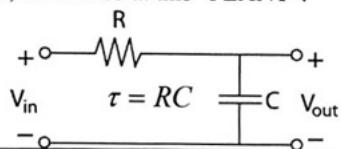
Configure the R31JP as the means of a feedback system



In a feedback system, we are typically concerned with 3 system properties:

- STABILITY
- TRANSIENT RESPONSE
- STEADY-STATE ERROR

Last time, recall, we looked at this "PLANT":

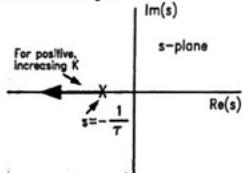


Using Laplace $\frac{d}{dt} \leftrightarrow s$	Differential Equation	Step Response
$\frac{V_{out}}{V_{in}}(s) = \frac{1}{\tau s + 1}$ transfer function	$\tau \frac{dv_{out}}{dt} + v_{out} = v_{in}$	$v_{out}(t) = (1 - e^{-\frac{t}{\tau}}) u(t)$

relate these two on a pole-zero map

1

Pole-zero map:



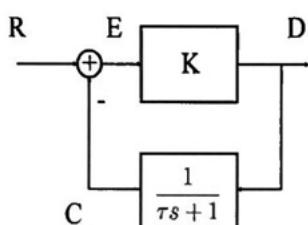
So, under feedback:

- Apparently always stable
- Improved transient response with increasing K
- Some steady state error, but decreases as K increases

Why does the overall system appear to get "faster" if we use feedback with high gain?

Let's see how the loop "kicks" the plant.

That is what's $\frac{D}{R}(s)$?



$$\frac{1}{\tau s + 1}$$

$$\tau_1 = \frac{\tau_2}{1+K}$$

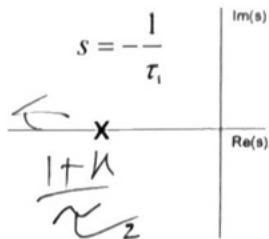
3

s = "natural frequency"

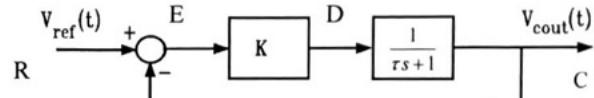
Pole-zero map:

$$s = -\frac{1}{\tau_1}$$

- τ must be positive, s must be negative for stability.
- So this plot tells us about stability and response speed.



Feedback controller for the RC-plant:



$$\frac{C}{R} = \frac{\frac{K}{\tau s + 1}}{1 + \frac{K}{\tau s + 1}} = \frac{K}{\tau s + 1 + K} = \frac{K}{\frac{\tau}{1+K}s + 1}$$

Step response:

$$v_{out} = \frac{K}{1+K} \cdot \left[1 - e^{-\frac{t}{\tau/1+K}} \right] u(t)$$

2

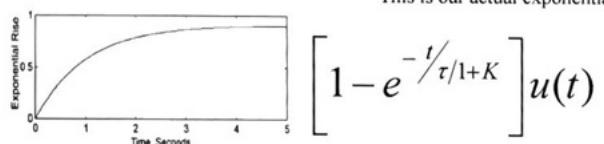
$$\frac{D}{R}(s) = \frac{K}{1 + \frac{K}{\tau s + 1}} = \frac{K}{\frac{s\tau}{1+K} + 1} = \frac{K}{1+K} \cdot \frac{1}{(\tau s + 1)} \cdot \frac{1}{\frac{\tau}{1+K}s + 1}$$

Takes a derivative and scales by tau

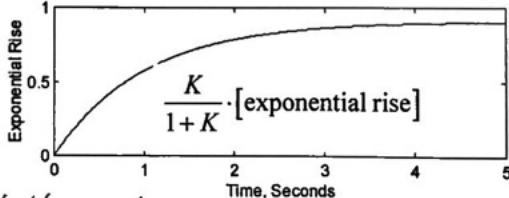
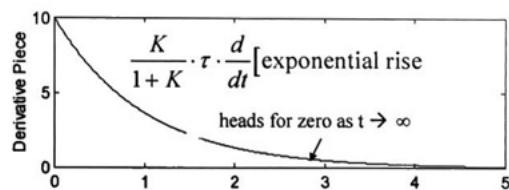
Scales by $\frac{K}{1+K}$; Takes the rise "as is"

This is our actual exponential rise

Here is the exponential rise:

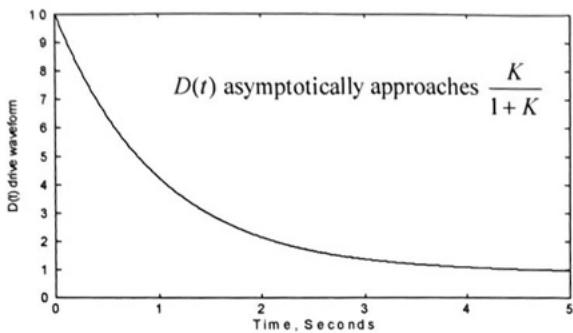


So D(t) is the sum of two pieces:



τ_1 = placeholder value
 τ_2 = motor property

4



D(t) asymptotically approaches K/(1+K).

But initially, it “kicks” the RC plant very hard (relative to steady state) to get it “moving”...

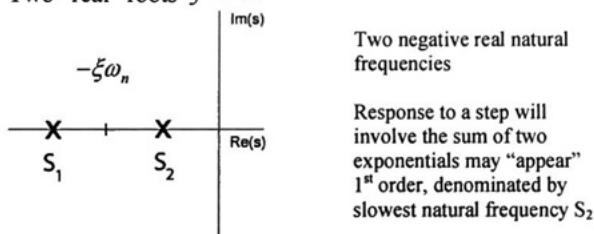
- Why are we focusing on step responses? Good input to examine transient response and steady error!
- Why are we focusing on 1st order systems? Well, they’re easy to understand. But they are obviously not the only system type in the world.

Second order systems:

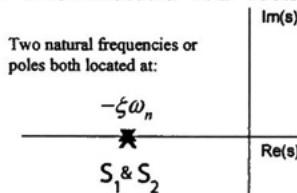
5

6

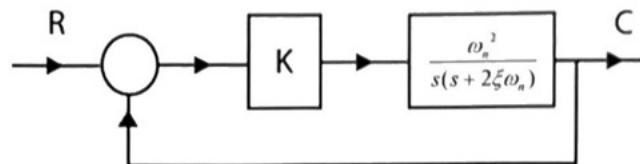
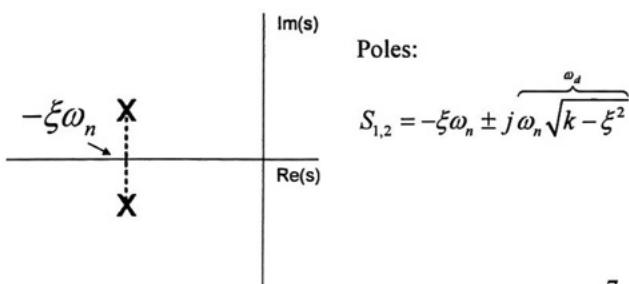
- Two “real” roots $\xi^2 > K$



- Two collocated “real” roots: $\xi^2 = K$



- Two complex roots: $\xi^2 < K$



$$\frac{C}{R} = \frac{\frac{K\omega_n^2}{s^2 + 2\xi\omega_n s}}{1 + \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + K\omega_n^2}} = \frac{K\omega_n^2}{s^2 + 2\xi\omega_n s + K\omega_n^2}$$

Why this 2nd order system? There are many slight variations. But this one is fine for exploring the behavior of a 2nd order denominator.

What are the natural frequencies? Roots of the denominator polynomial.

Let’s find roots of the denominator polynomial using quadratic formula:

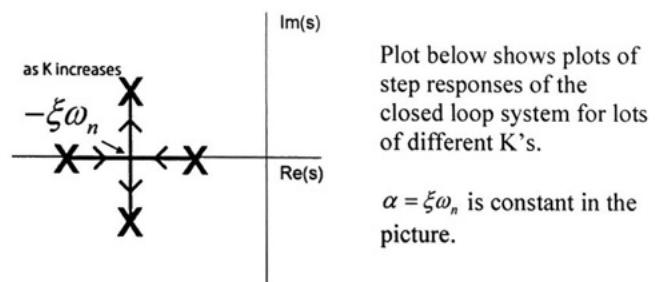
$$S_{1,2} = -\xi\omega_n \pm \sqrt{(\xi\omega_n)^2 - 4K\omega_n^2/4} = -\xi\omega_n \pm \omega_n\sqrt{(\xi^2 - K)}$$

3 interesting cases:

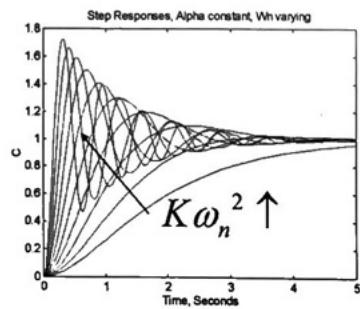
Step response for two complex roots with K = 1 :

$$C(t) = 1 - \frac{e^{-\xi\omega_n t}}{\sqrt{1-\xi^2}} \sin\left(\omega_d t + \tan^{-1}\frac{\sqrt{1-\xi^2}}{\xi}\right)$$

Root Locus (Pole-zero map) as we vary K: $0 \leq K \leq \infty$



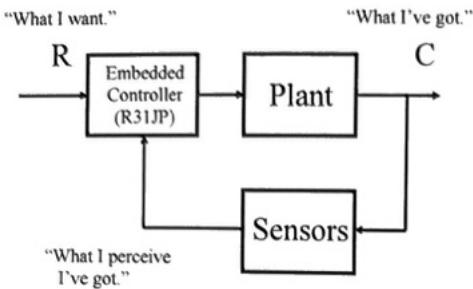
Finally: Why all the fuss about 1st and 2nd order systems? Many systems have a dominant 1st order pole or a complex conjugate pair of poles, and appear approximately as 1st and 2nd order.



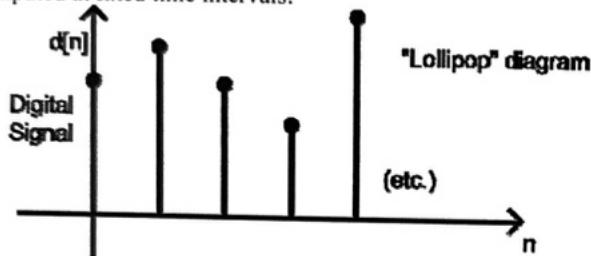
7

8

Today: Implement a digital, discrete-time feedback loop using the R31JP:



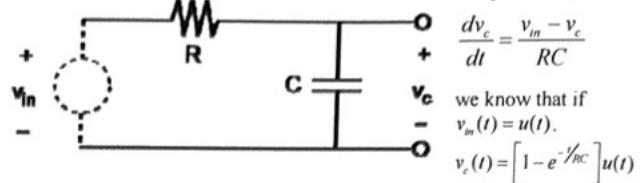
Here's the problem: The plant is often described by CT (continuous time) differential equations. The R31JP lives in a DT (discrete time) world with signal values that are measured or computed at fixed time intervals:



How do we connect the DT and CT worlds, and how do we make a model for the "whole" system?

1

Example: "Simulate" a CT circuit with DT computer equations



But suppose we didn't know the answer and wanted the computer to find it for us. One (poor ☹) approach is a DT approximation with Forward Euler.

$$\frac{dv_c}{dt} \approx \frac{v_c(t + \Delta t) - v_c(t)}{\Delta t} = \frac{v_c[n+1] - v_c[n]}{\Delta t}$$

where $v_c[n] = v_c(n\Delta t)$

$$\text{so } \frac{v_c[n+1] - v_c[n]}{\Delta t} = \frac{v_{in}[n] - v_c[n]}{RC}$$

$$\text{or } v_c[n+1] = \left[1 - \frac{\Delta t}{RC}\right]v_c[n] + \frac{\Delta t}{RC}v_{in}[n]$$

$$\text{or } v_c[n+1] + \left[\frac{\Delta t}{RC} - 1\right]v_c[n] = \frac{\Delta t}{RC}v_{in}[n]$$

Solve this difference equation by finding homogeneous and particular solutions, as for differential equations.

2

Aside:

Notice that transfer fns. Still work in DT.

Before, $s \Leftrightarrow \frac{d}{dt}$
Now, $z \Leftrightarrow$ a shift!

$$\text{Example: } v_c[n+1] + \left[\frac{\Delta t}{RC} - 1\right]v_c[n] = \frac{\Delta t}{RC}v_{in}[n]$$

$$\text{z-shift! } \left[z + \left(\frac{\Delta t}{RC} - 1\right)\right]v_c(z) = \frac{\Delta t}{RC}v_{in}(z)$$

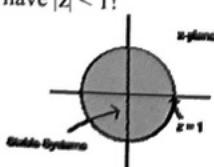
$$\text{Transfer fn: } \frac{v_c(z)}{v_{in}(z)} = \frac{\frac{\Delta t}{RC}}{z + \left(\frac{\Delta t}{RC} - 1\right)} = H(z)$$

$$\text{and: } z = \left(1 - \frac{\Delta t}{RC}\right)$$

The z "pole" is the solution of the homogeneous response. Must be inside the unit circle for stability.

(In CT, the actual capacitor voltage is: $v_c(t) = [1 - e^{-t/RC}]u(t)$)

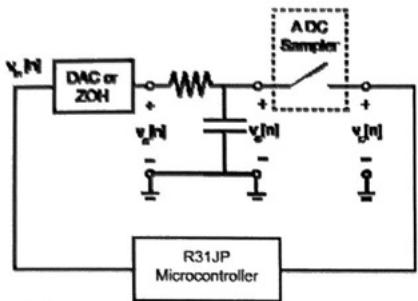
For stability we must have $|z| < 1$!



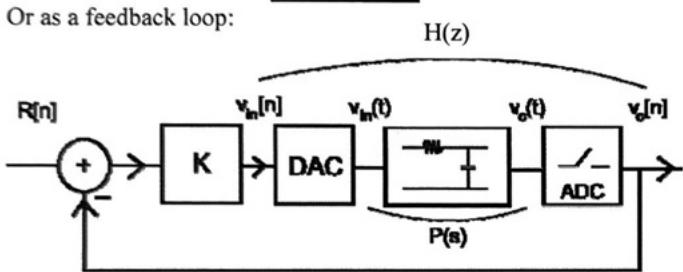
3

4

Back to the R31JP: Problem → feedback loop is mixed DT & CT!
Example:



Or as a feedback loop:



Strange!

$v_c(t)$ & $v_m(t)$ could be transferred and related by a "Laplace" style s-type transfer fxn.
 $v_c[n]$ & $v_m[n]$ could be related by z-transform transfer fxn.

Neat! 1 common representation, all in DT or all in CT, to describe feedback loop.

5

How is $H(z)$ related to $P(s)$?
Step invariant transformation! Sample the CT step response!

A unit step (DT) in $v_m[n]$ gives a unit step (CT) in $v_m(t)$.
Find $H(z)$:

1. First, compute CT step response:

$$v_m(t) = u(t) \Rightarrow v_c(t) = (1 - e^{-\frac{t}{\tau}})u(t)$$

2. Sample the CT step response:

$$v_c(nT) = (1 - e^{-\frac{nT}{\tau}})u(nT)$$

$$\text{or } v_c[n] = [1 - \lambda^n]u[n] \text{ where } \lambda = e^{-\frac{T}{\tau}}$$

3. Find a difference equation whose unit step response is the same as in step 2.

If you know 6.003, take z-transform:

$$v_c(z) = \frac{1}{1 - z^{-1}} - \frac{1}{1 - \lambda z^{-1}} = \frac{(1 - \lambda)z^{-1}}{(1 - z^{-1})(1 - \lambda z^{-1})}$$

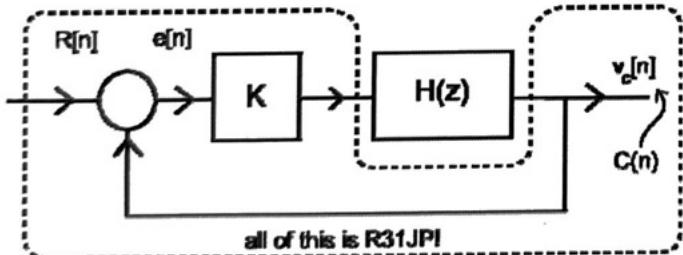
$$\text{so } \frac{v_c(z)}{u(z)} = \frac{v_c(z)}{v_m(z)} = \frac{z^{-1}(1 - \lambda)}{(1 - \lambda z^{-1})} = H(z)$$

$$\text{or } v_c[n] - \lambda v_c[n-1] = (1 - \lambda)v_m[n-1]$$

4. If you haven't already done so, take z-transform!

6

Now, the whole loop is in "DT"



People have tabulated how $P(s) \leftrightarrow H(z)$!
Makes life easy!

Now:

$$\begin{aligned} \frac{C}{R}(z) &= \frac{\frac{Kz^{-1}(1-\lambda)}{1-\lambda z^{-1}}}{1 + \frac{Kz^{-1}(1-\lambda)}{1-\lambda z^{-1}}} = \frac{Kz^{-1}(1-\lambda)}{1-\lambda z^{-1} + K(1-\lambda)z^{-1}} \\ &= \frac{Kz^{-1}(1-\lambda)}{1 + [K(1-\lambda) - \lambda]z^{-1}} \end{aligned}$$

Design K to give desired performance!

TABLE 3.1 Sampling of a continuous time system, $G(s)$.

The table gives the zero-order-hold equivalent of the continuous time system, $G(s)$, preceded by a zero-order hold. The sampled system is described by its pulse transfer operator. For second-order systems the pulse-transfer operator is given in terms of the coefficients of

$$H(q) = \frac{b_1 q + b_2}{q^2 + a_1 q + a_2}$$

$$q \Leftrightarrow z$$

$$a \leftrightarrow \frac{1}{T}$$

$$h \Leftrightarrow \Delta t$$

$$(T)$$

Sample
Interval

$G(s)$	$H(q)$ or the coefficients in $H(q)$
$\frac{1}{s}$	$\frac{h}{q - 1}$
$\frac{1}{s^2}$	$\frac{h^2(q + 1)}{2(q - 1)^2}$
e^{-sh}	q^{-1}
$\frac{a}{s + a}$	$\frac{1 - \exp(-ah)}{q - \exp(-ah)}$
$\frac{a}{s(s + a)}$	$b_1 = \frac{1}{a}(ah - 1 + e^{-ah}) \quad b_2 = \frac{1}{a}(1 - e^{-ah} - ah e^{-ah})$ $a_1 = -(1 + e^{-ah}) \quad a_2 = e^{-ah}$
$\frac{a^2}{(s + a)^2}$	$b_1 = 1 - e^{-ah}(1 + ah) \quad b_2 = e^{-ah}(e^{-ah} + ah - 1)$ $a_1 = -2e^{-ah} \quad a_2 = e^{-2ah}$
$\frac{ab}{(s + a)(s + b)}$	$b_1 = \frac{b(1 - e^{-ah}) - a(1 - e^{-bh})}{b - a}$ $b_2 = \frac{a(1 - e^{-bh})e^{-ah} - b(1 - e^{-ah})e^{-bh}}{b - a}$ $a_1 = -(e^{-ah} + e^{-bh})$ $a_2 = e^{-(a+b)h}$
$\frac{(s + c)}{(s + a)(s + b)}$	$b_1 = \frac{e^{-bh} - e^{-ah} + (1 - e^{-bh})c/b - (1 - e^{-ah})c/a}{b - a}$ $b_2 = \frac{c}{ab}e^{-(a+b)h} + \frac{b - c}{b(a - b)}e^{-ah} + \frac{c - a}{a(a - b)}e^{-bh}$ $a_1 = -e^{-ah} - e^{-bh} \quad a_2 = e^{-(a+b)h}$
$\frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$	$b_1 = 1 - \alpha(\beta + \frac{\zeta\omega_0}{\omega}\gamma) \quad \omega = \omega_0\sqrt{1 - \zeta^2} \quad \zeta < 1$ $b_2 = \alpha^2 + \alpha\left(\frac{\zeta\omega_0}{\omega}\gamma - \beta\right) \quad \alpha = e^{-\zeta\omega_0 h}$ $a_1 = -2\alpha\beta \quad \beta = \cos(\omega h)$ $a_2 = \alpha^2 \quad \gamma = \sin(\omega h)$
$\frac{s}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$	$b_1 = \frac{1}{\omega}e^{-\zeta\omega_0 h} \sin(\omega h) \quad b_2 = -b_1 \quad \omega = \omega_0\sqrt{1 - \zeta^2}$ $a_1 = -2e^{-\zeta\omega_0 h} \cos(\omega h) \quad a_2 = e^{-2\zeta\omega_0 h}$

Regulated Power Supply: given an unregulated DC input, produce a regulated DC output voltage, e.g. 5V. Today consider two possibilities.

Linear Regulator

For sake of comparison
And analysis, let's compute
Efficiency.

(ignore i_{small})

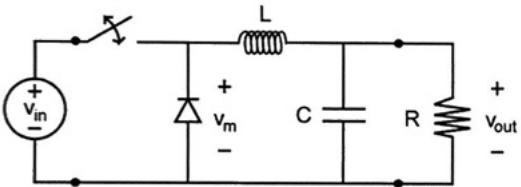
$$P_{in} = v_{in} \cdot I$$

$$P_{out} = v_{out} \cdot I$$

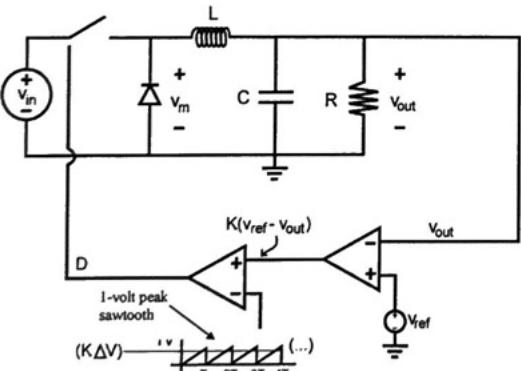
$$\text{so, } \eta = \text{efficiency} = \frac{P_{out}}{P_{in}} = \frac{v_{out}}{v_{in}}$$

For a typical $v_{in} = 10V$, $\eta = 0.5$ (only 50% of input power goes to load)
Where does the rest of the input power go? Dissipated in the transistor as waste heat.

Switching Regulator



1



Used in many early PC power supplies!

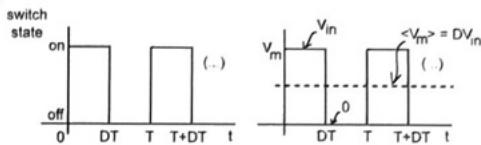
RLC "filter": find transfer fxn $\frac{v_{out}}{v_m}(s)$

$$\frac{v_{out}}{v_m}(s) = \frac{R \parallel \frac{1}{sC}}{sL + R \parallel \frac{1}{sC}} ; R \parallel \frac{1}{sC} = \frac{R}{sC} = \frac{R}{sRC + 1}$$

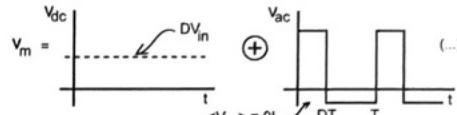
$$\text{so } \frac{v_{out}}{v_m}(s) = \frac{\frac{R}{sRC + 1}}{sL + \frac{R}{sRC + 1}} = \frac{R}{s^2 RLC + sL + R}$$

$$\text{or } \frac{v_{out}}{v_m}(s) = \frac{\frac{R}{sRC + 1}}{sL + \frac{R}{sRC + 1}} = \frac{\frac{1}{LC}}{s^2 + s \frac{1}{RC} + \frac{1}{LC}} = P(s)$$

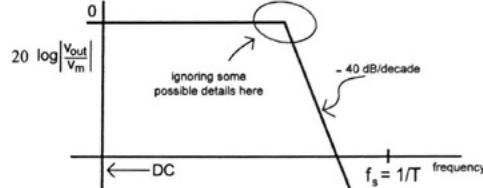
Controllable switch: operated periodically, with period T and "duty cycle" D:



NOTICE:

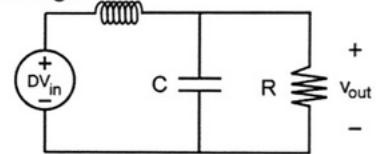


Design LCR "filter" so that



With this arrangement, v_{ac} will be severely attenuated.

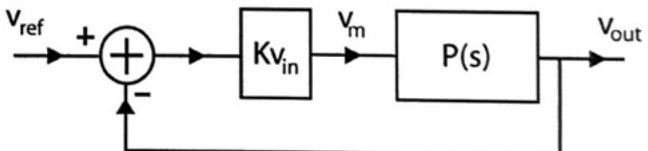
$Dv_{in} = v_{dc}$ will pass right through.



100% efficient! All input power goes to load (for ideal L, C).

2

Block Diagram:



Over all transfer function (Black's formula):

$$\frac{v_{out}}{v_m}(s) = \frac{Kv_m P(s)}{1 + Kv_m P(s)} = \frac{\frac{Kv_m}{LC}}{s^2 + s \frac{1}{RC} + \frac{1}{LC}} = \frac{\frac{Kv_m}{LC}}{1 + \frac{1}{s^2 + s \frac{1}{RC} + \frac{1}{LC}}}$$

$$\text{or } \frac{v_{out}}{v_m}(s) = \frac{\frac{Kv_m}{LC}}{s^2 + \frac{s}{RC} + \left(\frac{1}{LC} + \frac{Kv_m}{LC} \right)}$$

Is it stable? Depends!

- If R is present, finite, then denominator is quadratic with all positive coefficients. STABLE
- No load, poles on $j\omega$ axis. NOT STABLE

3

4

With R present, how does the system perform?

Aside: What are we worried about here? ORIGINAL TTL Spec permitted $5V \pm 300mV$. So, over or undershooting the spec could cause serious problems; possibly even destroy the load!

DC, tracking performance: suppose $v_{ref} = u(t)$ (turn on). After a long time, what is v_{out} ?

Final value theorem: $h(t), t \rightarrow \infty = \lim_{s \rightarrow 0} s \cdot H(s)$

$$\text{So, } v_{out}(s) = \frac{\frac{Kv_{in}}{LC}}{s^2 + \frac{s}{RC} + \left(\frac{1}{LC} + \frac{Kv_{in}}{LC}\right)} \cdot \frac{1}{s} \cdot \underset{\text{input}}{s}$$

Limit as $s \rightarrow 0$:

$$v_{out}(s) \underset{\lim(s \rightarrow 0)}{=} \frac{\frac{Kv_{in}}{LC}}{\frac{1}{LC} + \frac{Kv_{in}}{LC}} \approx 1 \quad \text{if } K \text{ is large}$$

So, for good steady state "tracking" performance, i.e., minimum steady state error, we would seem to want $K \rightarrow \text{large}$.

What happens to the closed loop pole (CLP) locations as K gets big?

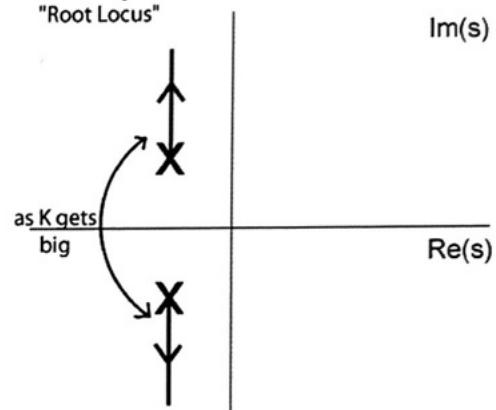
$$P_{1,2} = -\frac{1}{RC} \pm \sqrt{\frac{1}{(RC)^2} - \frac{4(1+Kv_{in})}{LC}}$$

As K gets huge, the term inside the square root gives us complex poles which are increasingly higher in frequency.

Real part stay fixed.

CLP locations in the s-plane:

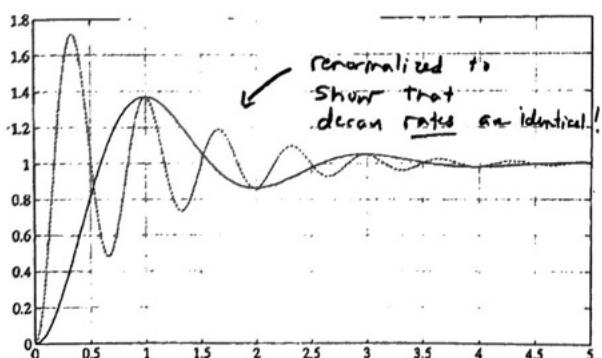
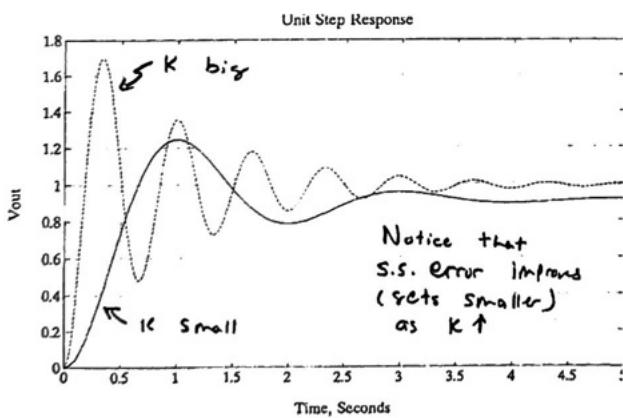
"Root Locus"



5

6

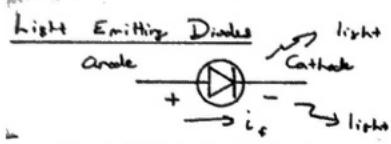
As K increases, decay rate remains constant, but oscillation frequency increases.



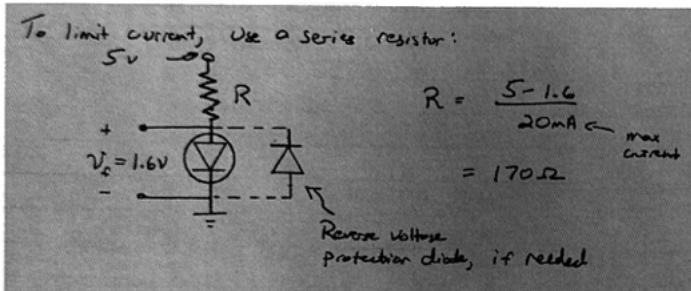
I. The Blinking Light:

Incandescent Lamp: A filament heated until it glows. Works with AC or DC, may be controlled and dimmed with switches, semiconductors, etc.

Light Emitting Diodes:

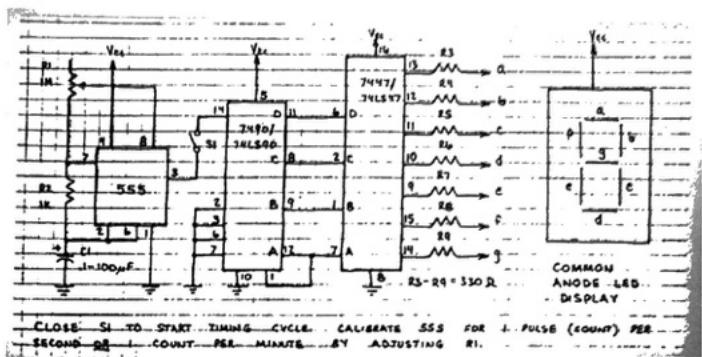


- Forward current typically limited to 10-20 mA.
- Maximum reverse voltage may be small, e.g., 5 V
- Forward voltage drop may range from 0.6 V to 2.2 V or more, depending on the LED technology.
- Limited spectral (color) output



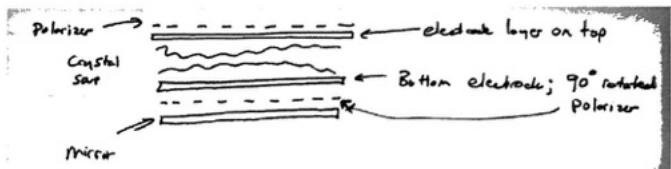
1

Multi-segment LED displays - useful for digits:



II. Liquid Crystal Displays:

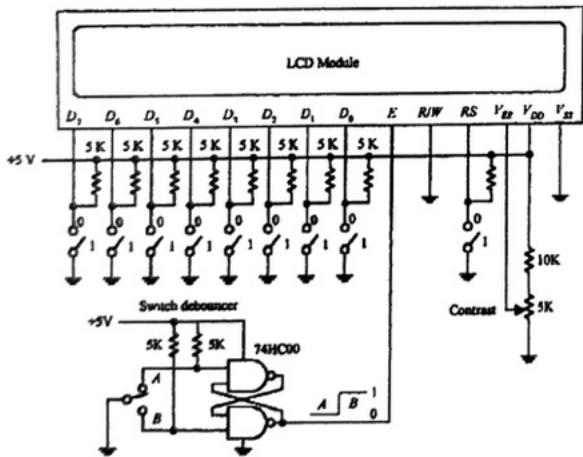
LCD displays work by controlling the polarization of light traveling through an optical "sandwich":



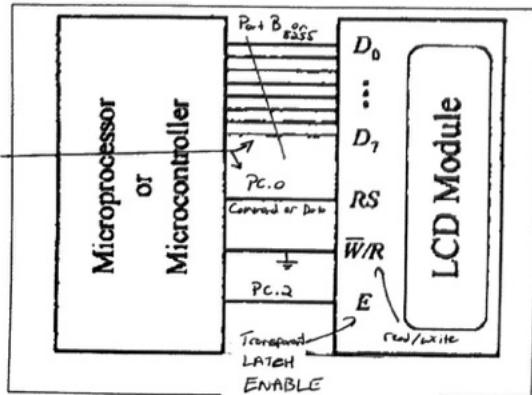
(This is discussed in Scherz.)

2

Test hookup described in Scherz:



The panel might connect to a microcontroller this way:



Sample programming plan:

On our staff nerd kit, "W" is the MINMON write command. Port A = FE08, Port B = FE09, Port C = FE0A, control register is at FE0B.

The Hitachi 44780 controller used in our display has created a ubiquitous standard. Typical programming plan:

- Activate the 8255: (all 3 ports output)
WFE0B = 80
- Set display for 8 bit communication, 5x7 character set:
WFE0A = 00 (lower the "E" line)
WFE09 = 38
WFE0A = 04 (raise the "E" line)
WFE0A = 00 (lower the E line, latching command)
- Turn display on, hide cursor:
WFE09 = 0C
WFE0A = 04 (raise the "E" line)
WFE0A = 00 (lower the E line, latching command)
- Clear display:
WFE09 = 01
WFE0A = 04 (raise the "E" line)
WFE0A = 00 (lower the E line, latching command)
- Set RAM address to zero:
WFE0A = 00 (lower the "E" line)
WFE09 = 80
WFE0A = 04 (raise the "E" line)
WFE0A = 00 (lower the E line, latching command)
- "Wash, rinse, repeat" to display characters ☺:
WFE0A = 01 (lower the "E" line)
WFE09 = 39 (displays a "9")
WFE0A = 05 (raise the "E" line)
WFE0A = 01 (lower the E line, latching command)

4

LCD Instruction Set

INSTRUCTION	R/S	R/W	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Clear Display	0	0	0	0	0	0	0	0	0	1
Display & Cursor Home	0	0	0	0	0	0	0	0	1	X
Character Entry Mode	0	0	0	0	0	0	0	1	I/D	S
Display & Cursor On/Off	0	0	0	0	0	0	0	1	D	C
Display/Cursor Shift	0	0	0	0	0	1	D/C	R/L	X	X
Function Set	0	0	0	0	1	DL	N	F	X	X
Set CGRAM Address	0	0	0	1	A	A	A	A	A	A
Set Display Address	0	0	1	A	A	A	A	A	A	A
Poll the "Busy Flag"	0	0	BF	X	X	X	X	X	X	X
Write Character to Display ^a	1	0	D	D	D	D	D	D	D	D
Read Character on Display ^b	1	1	D	D	D	D	D	D	D	D

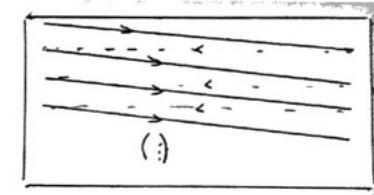
- I/D = Increment (I/D = 1)/Decrement (I/D = 0) each byte written to display
 S = Display shift on (S = 1), Display shift off (S = 0)*
 D = Turn display on (D = 1), Turn display off (D = 0)*
 C = Show cursor (C = 1), Hide cursor (C = 0)
 B = Underline cursor (B = 0, C = 1), Blink cursor (B = 1, C = 1)
 D/C = Move display (D/C = 1), Move cursor (D/C = 0)
 R/L = Direction of shift: Shift right (R/L = 1), Shift left (R/L = 0)
 DL = Set data interface length: 8-bit interface (DL = 1)*, 4-bit interface (DL = 0)
 N = Number of display lines: 2 line mode (N = 1), 1 line mode (N = 0)*
 F = Character font format: 5 x 10 dot (F = 1), 5 x 7 dot (F = 0)*
 BF = Poll the Busy Flag: controller not busy (BF = 0), controller busy (BF = 1)
 A = CGRAM or display address bit
 D = Character data bit
 a = Write character to display at the current cursor position
 b = Read character on display at the current cursor position
 X = Don't care
 * = Initialization settings

5

LCD Character Codes

Char. code	0000 0000 1111 1111	0001 1111 0000 1111	0110 1111 1100 1111	0010 1010 0101 0101
xxxx0000	0JP`P - パ	1AQa9. ハク	2BRbr' ハク	3CScs ハク
xxxx0001	!1AQa9. ハク	"2BRbr' ハク	#3CScs ハク	\$4DTdt ハク
xxxx0010	%5EUeu オナ	&6FVfv ハク	*7GW9w ハク	(8HXhx イク
xxxx0011	;Kk オサヒロ	,L¥1 ハシフワ	-=M]m ユスヘン)N^n セホ
xxxx0100	>:JZJzz エコハレ	/?O_o ツツラ		
xxxx0101				
xxxx0110				
xxxx0111				
xxxx1000				
xxxx1001				
xxxx1010				
xxxx1011				
xxxx1100				
xxxx1101				
xxxx1110				
xxxx1111				

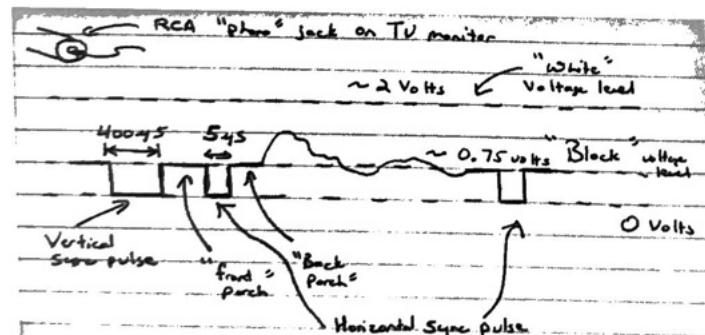
III. Composite video monitors:
"Scans" the screen:



Scans from the top to the bottom 60 times per second.
Scans a (slightly diagonal) horizontal line ~ 15750 times per second. Can vary between 15250 and 16250 Hz. Generally approximately 256 horizontal lines for "NTSC" TV standard.

6

How do we "take control" of a display with a composite input? Feed it an analog signal ("composite"):



- Vertical sync pulse returns the "gun" to the upper left corner of the screen.
- Horizontal sync pulse returns the "gun" to the left side of the screen to start a new "line".
- The shape and width of the sync pulses must be carefully controlled. Incorrect syncs can lead to a wavy or unreadable display.

Use sync generator chips to simplify raster timing.

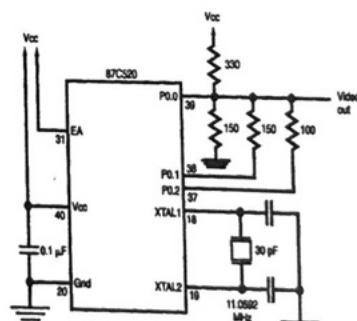
MM5321 and LM 1882 are classic "sync generators"

8275 and 6847 are classic "video display generators" or "controllers"

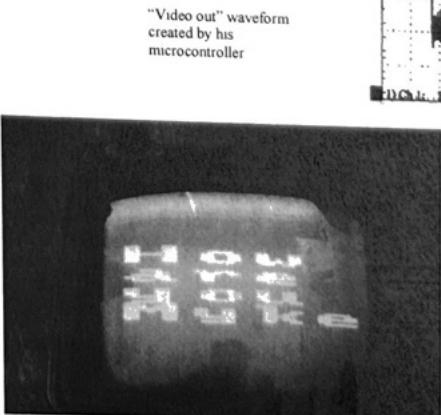
There are many other modern examples, with variants for fancier displays, e.g., VGA, component video, HDMI, etc.

Or, you can actually generate the correct timing pulses directly with your microcontroller!

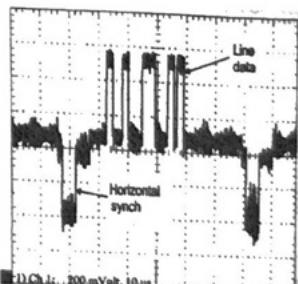
From Predko's "8051 Microcontroller": (I haven't tried this particular implementation, but the idea is generally sound...)



He's using Port 0 to create an informal video DAC (recall our discussion of the R2R ladder circuit).



As seen on the video monitor

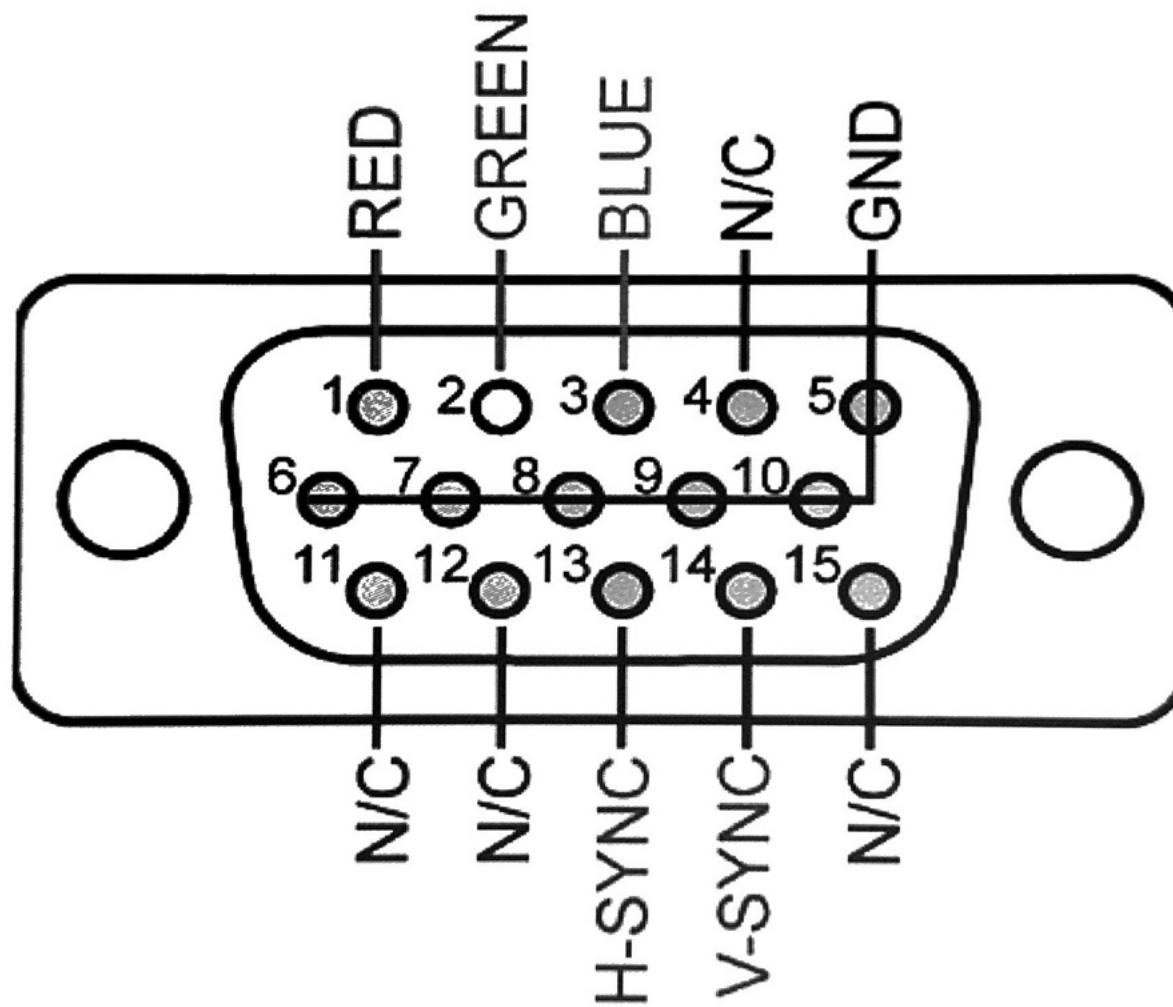


7

8

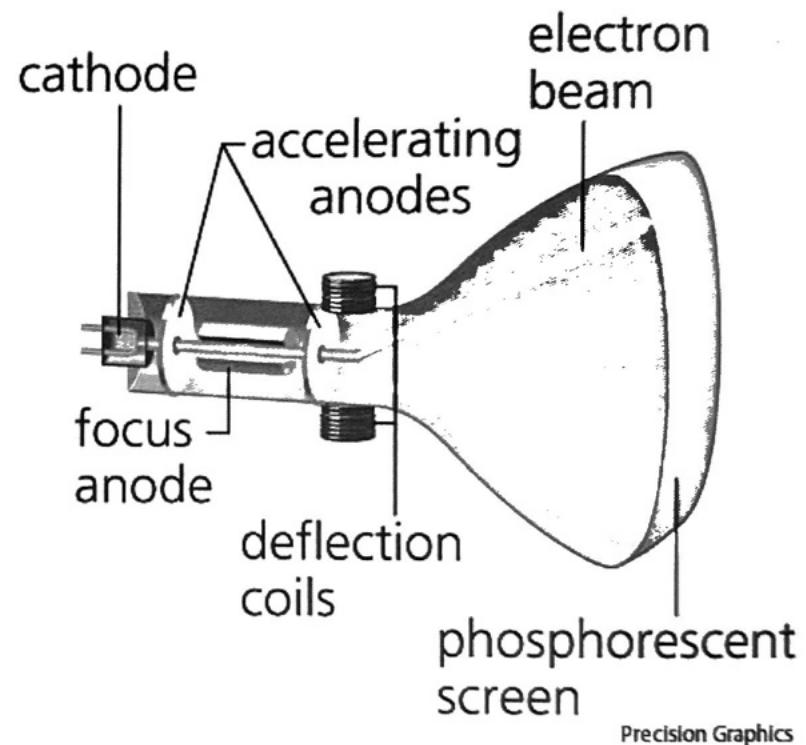
VGA Connection: D15 Sub-Miniature

VGA port, view from Wire Side



VGA: How does it work?

- > the image is drawn line by line, frame by frame
- > the monitor draws the image using 5 signals: **Hsync**, **Vsync**, **Red**, **Green**, and **Blue**
- > each end of **line** is signaled by a **sync pulse** from **Hsync**
- > each end of **frame** is signaled by a different sync pulse from **Vsync**



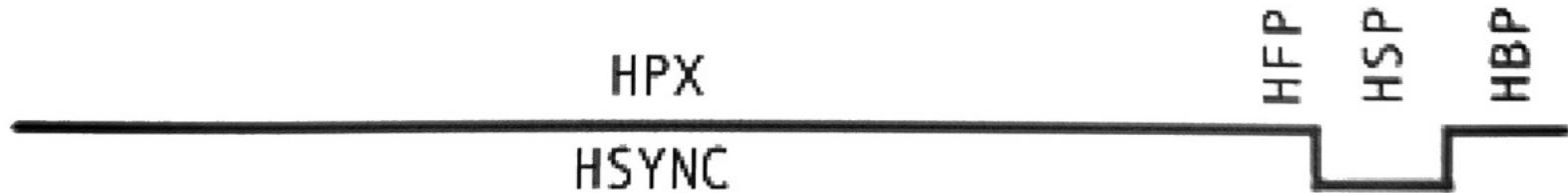
Precision Graphics

Syncing: Horizontal Sync

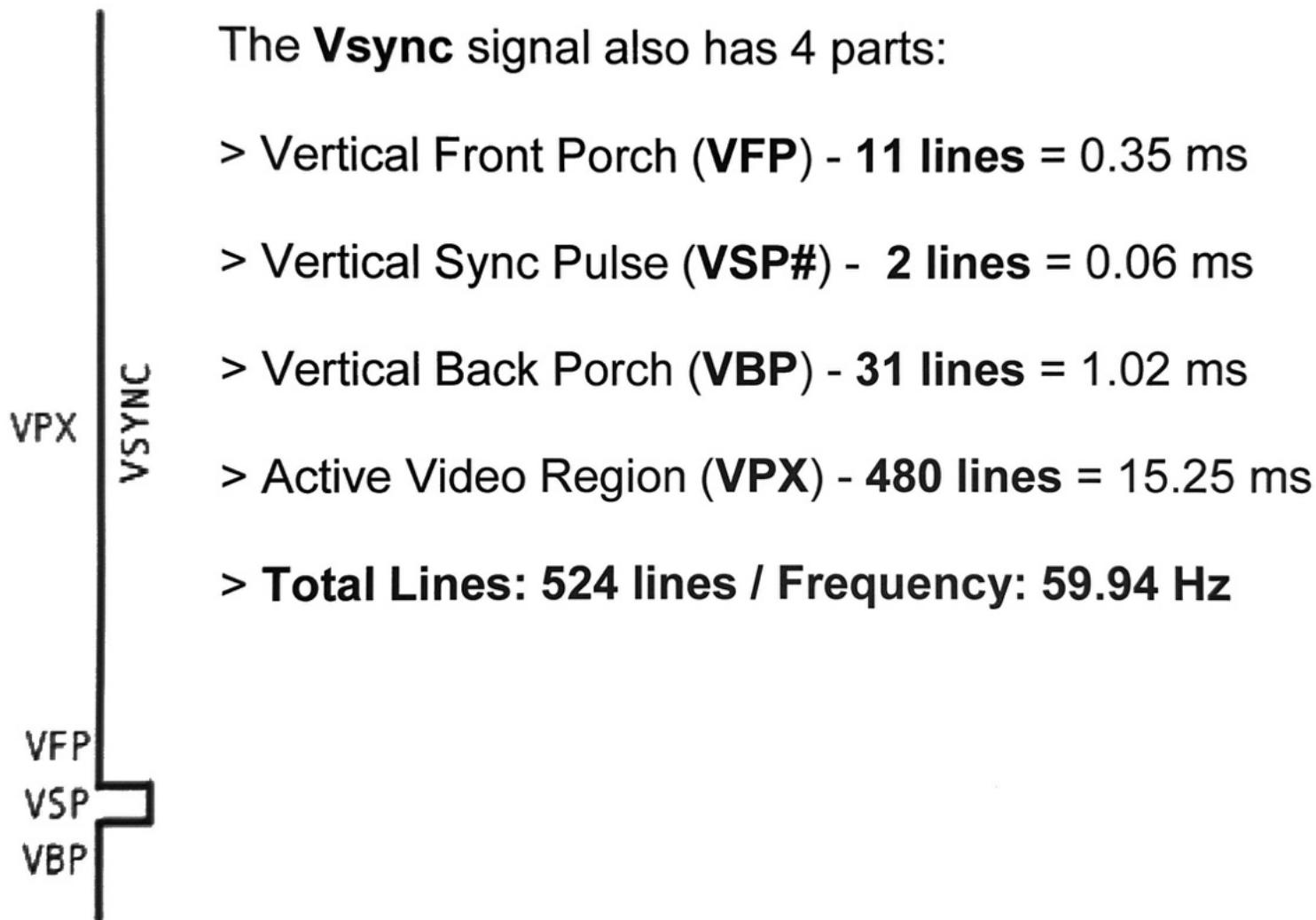
The **Hsync** signal has 4 parts:

- > Horizontal Front Porch (**HFP**) - **0.94 μs**
- > Horizontal Sync Pulse (**HSP#**) - **3.77 μs**
- > Horizontal Back Porch (**HBP**) - **1.89 μs**
- > Active Video Region (**HPX**) - **25.17 μs**
- > **Total Time: 31.77 μs / Frequency: 31.4686 KHz**

indicates signal is active low



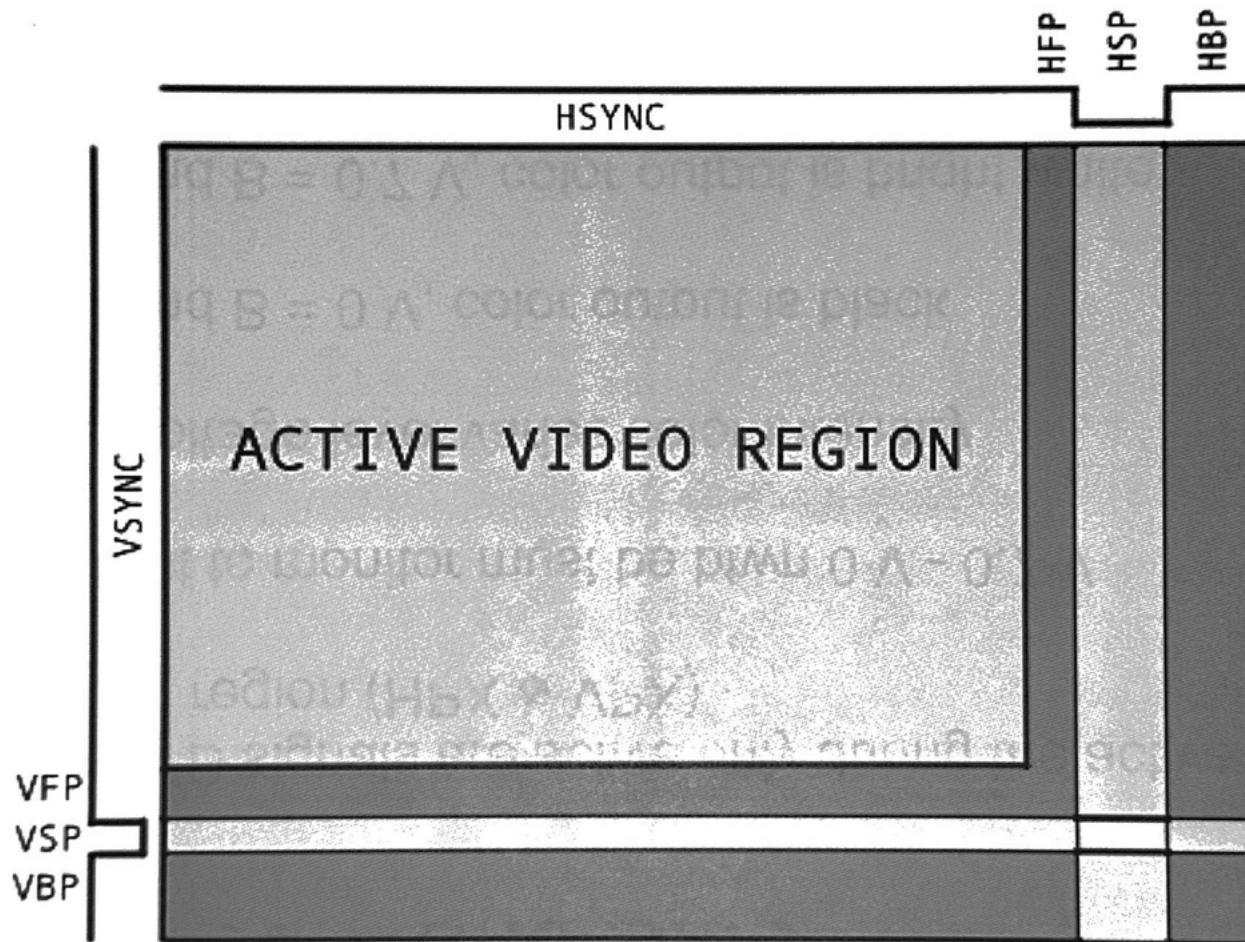
Syncing: Vertical Sync



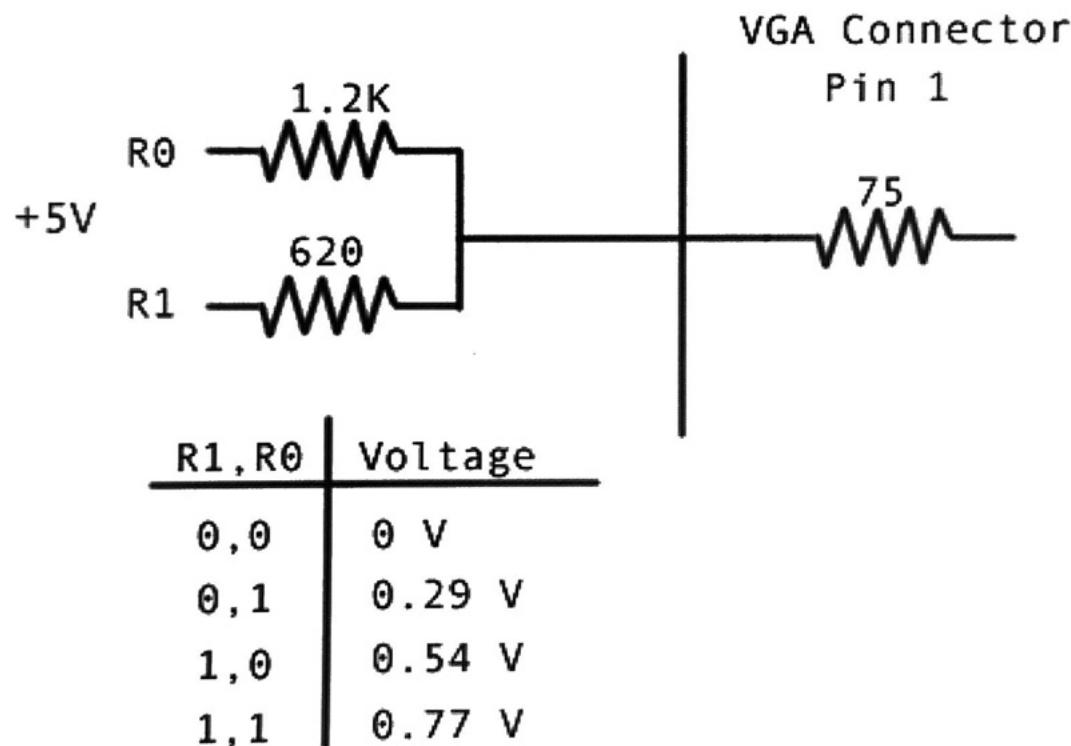
Red, Green, and Blue

- > R, G, and B signals are active only during the active video region (HPX & VPX)
- > RGB input to monitor must be btwn 0 V - 0.7 V
- > varying voltage level varies color intensity
- > if R, G, and B = 0 V, color output is black
- > if R, G, and B = 0.7 V, color output is bright white

Active Video vs. Sync

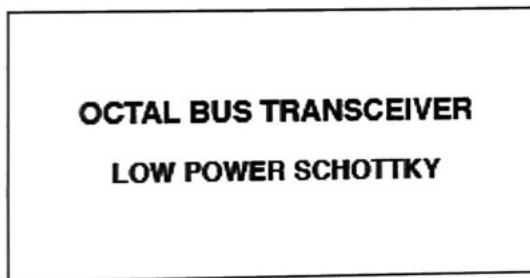


Creating Color! (6-bit resistor DAC)

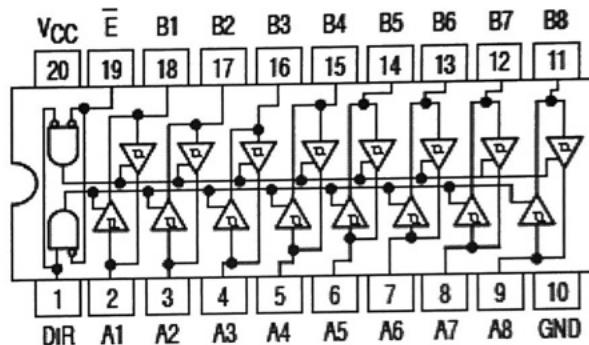


The 74LS245

Port 1's internal 30K resistors and the VGA connector's 75 Ohm impedance requires a buffer between the 89C430 and the VGA connector/DAC.



LOGIC AND CONNECTION DIAGRAMS DIP (TOP VIEW)



TRUTH TABLE

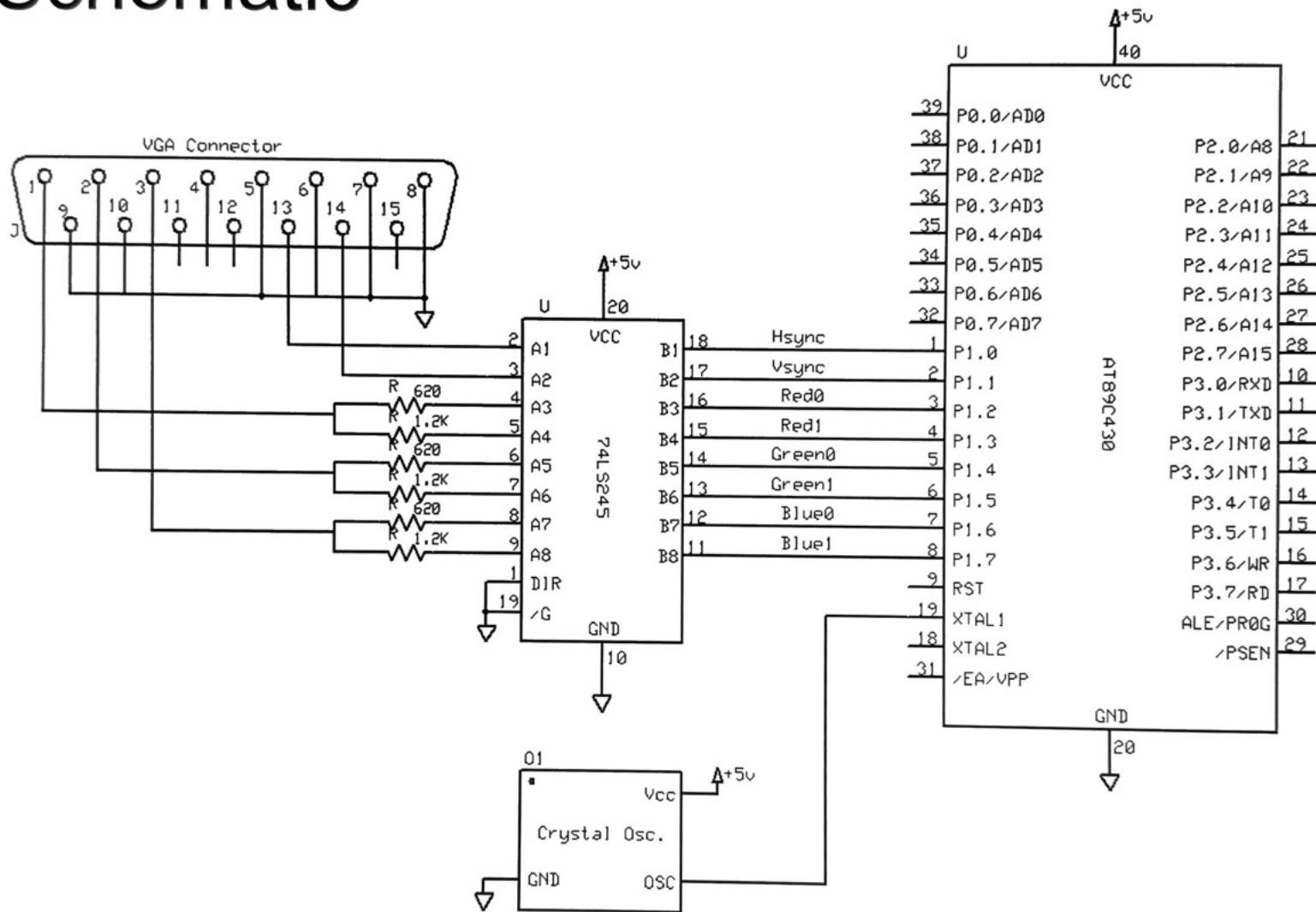
INPUTS		OUTPUT
E	DIR	
L	L	Bus B Data to Bus A
L	H	Bus A Data to Bus B
H	X	Isolation

H = HIGH Voltage Level

L = LOW Voltage Level

X = Immaterial

Schematic



Uncle Steve's Guide To Digital Signal Processing

With essential help from Ali Shoeb and Dan Lovell, former
6.115 Students!

Digital Signal Processing

- A satellite has just captured the following image of the moon and is ready to send it to an earth station.

Original Image



Digital Signal Processing

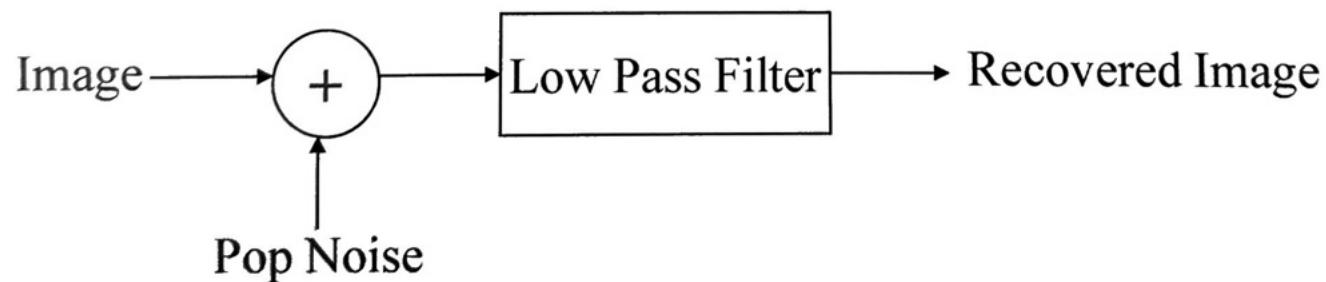
- Pop-Noise corrupts the the image of the moon as it is transmitted to earth.

Received Image



Digital Signal Processing

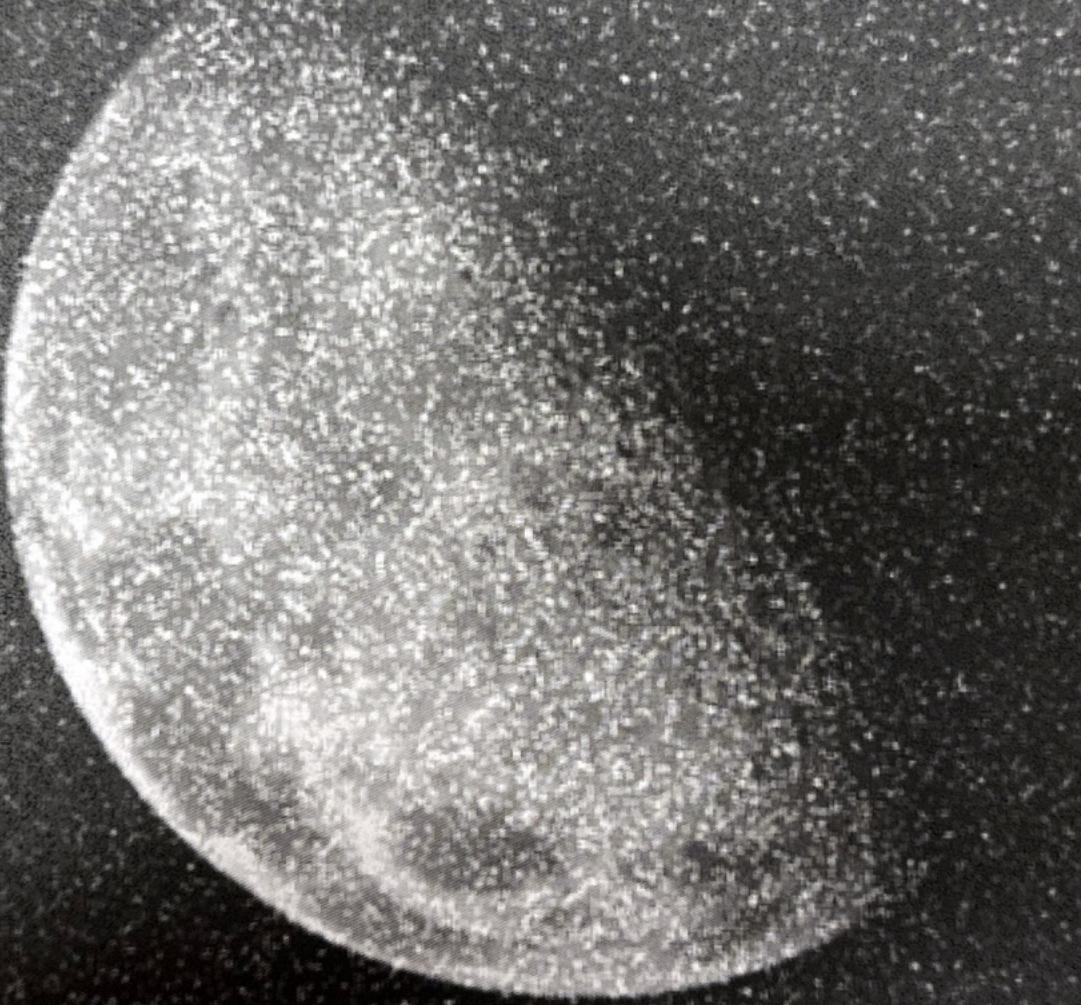
- High frequencies in images are fast transitions in pixel intensities: black → white or white → black in gray scale images.
- Pop-Noise introduces high frequency artifacts into an image.
- We can reduce high frequency by using a low pass filter.



Received Image



Low Pass Filter



Digital Signal Processing

- The low pass filter blurred the image of the moon and did not remove all of the Pop-Noise.
- Pop Noise corrupts a pixel in a neighborhood of pixels by changing its intensity to an extreme (black or white). This makes it easy to identify a pixel corrupted by Pop Noise from among a group of pixels.
- A Median Filter assigns each pixel in an image the median intensity of pixels in its neighborhood. The size of a pixel's neighborhood is specified by the user.

8-Pixel neighborhood surrounding
pixel corrupted by Pop Noise.

0	0	0
0	1	0
0	0	0

Pop Noise changed this value
from a $0 \rightarrow 1$

Median Filter will assign this
pixel a value of 0 since the
median pixel intensity in
neighborhood is zero.

Received Image



2X2 Median Filter

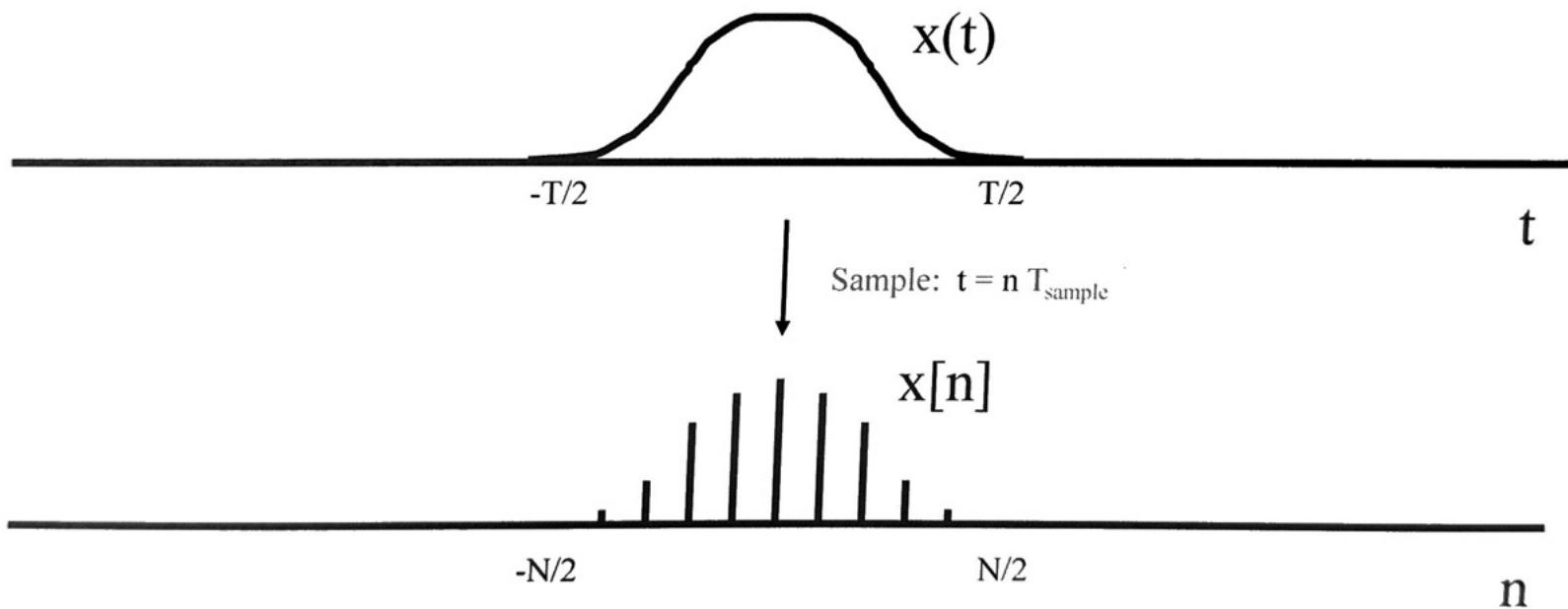


5X5 Median Filter



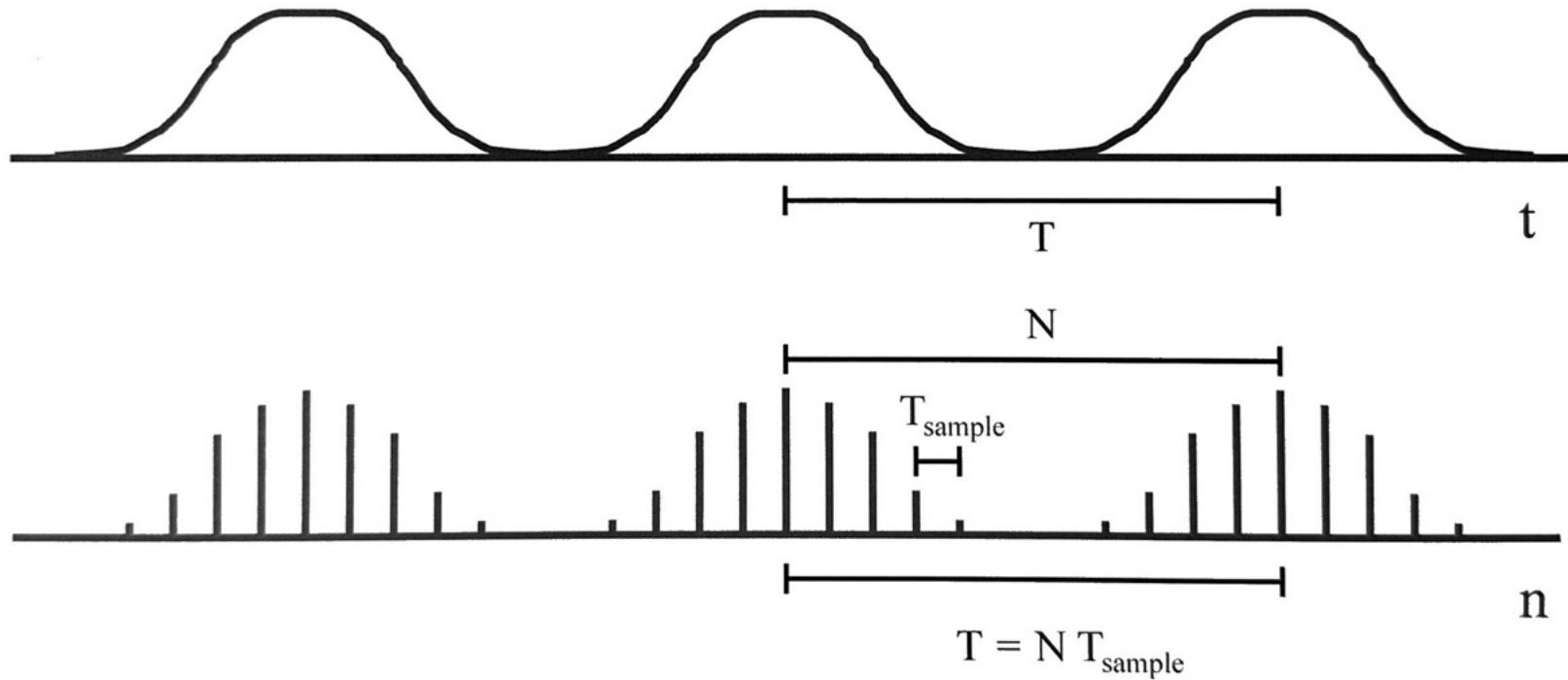
DT Signals are Samples of CT Signals

- We know how to construct a finite length discrete signal $x[n]$ by sampling a continuous signal $x(t)$ at the Nyquist Rate.



Periodic DT Signals Are Samples of Periodic CT Signals

- We can replicate the discrete signal $x[n]$ and think of it as samples of replicas of the continuous periodic signal $x(t)$.



Representing Continuous Periodic Signals

- Jean Fourier showed that any continuous periodic signal can be represented as infinite sum of harmonically related sinusoids. This representation tells about the frequency content of the signal.

$$x(t) = a_0 + \sum_{k=1}^{\infty} \underbrace{\{ B_k \cos(k\Omega_0 t) + C_k \sin(k\Omega_0 t) \}}_{\text{Weighted Sinusoids}} \quad -\infty < t < \infty$$

$$\Omega_0 = 2\pi / T$$

Periodic Square Wave

Let us see if Fourier is right!

- Suppose $x(t)$ is a square wave with a period T and Amplitude 1.
- Fourier claims

$$x(t) = (4/\pi) \sum_{n=1}^{\infty} \frac{\sin((2n-1)\Omega_0 t)}{(2n-1)}$$

$$\Omega_0 = 2\pi/T$$

- $x(t) = (4/\pi) [\sin(\Omega_0 t) + 1/3 \sin(3\Omega_0 t) + 1/5 \sin(5\Omega_0 t) \dots]$

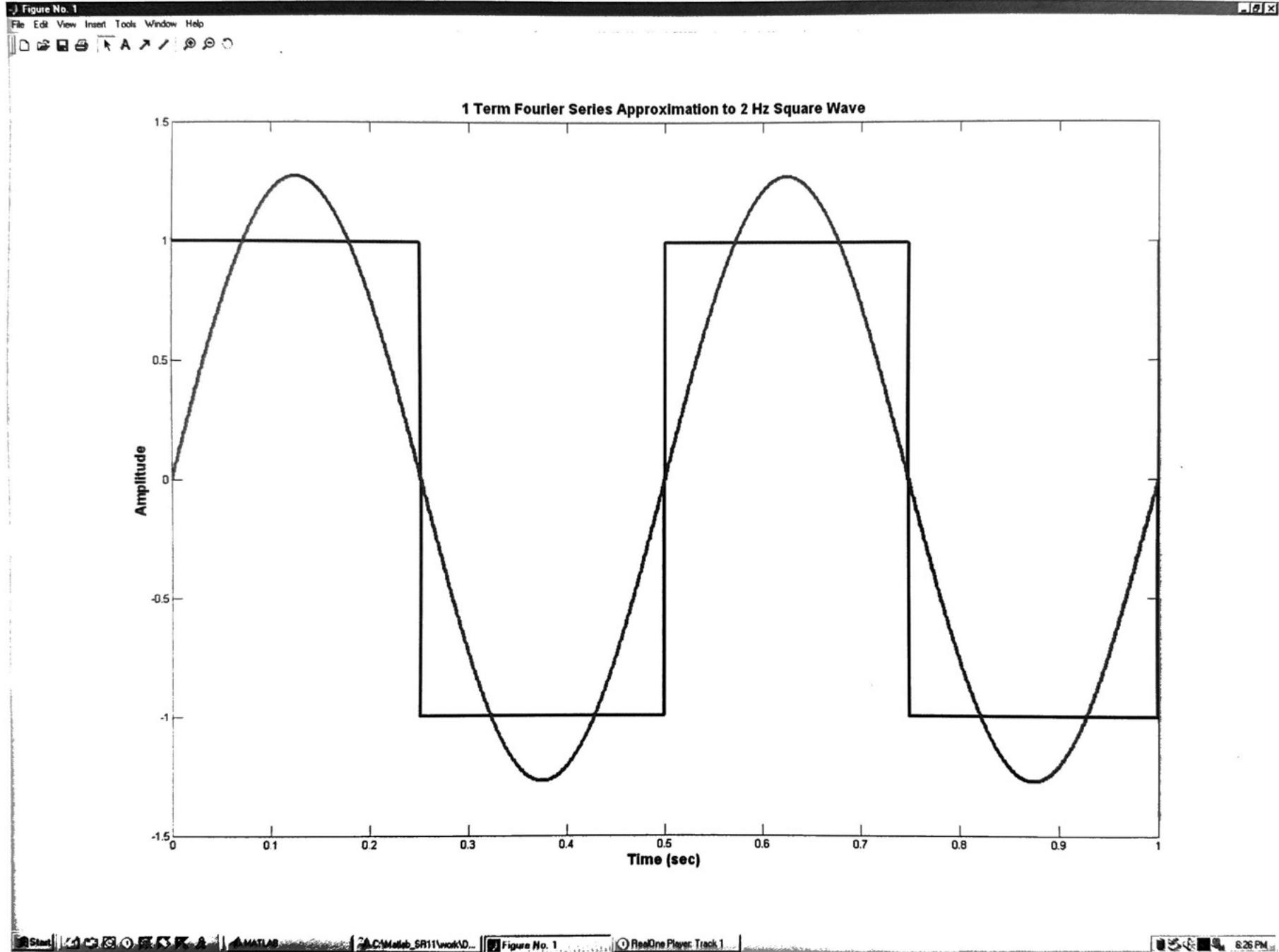
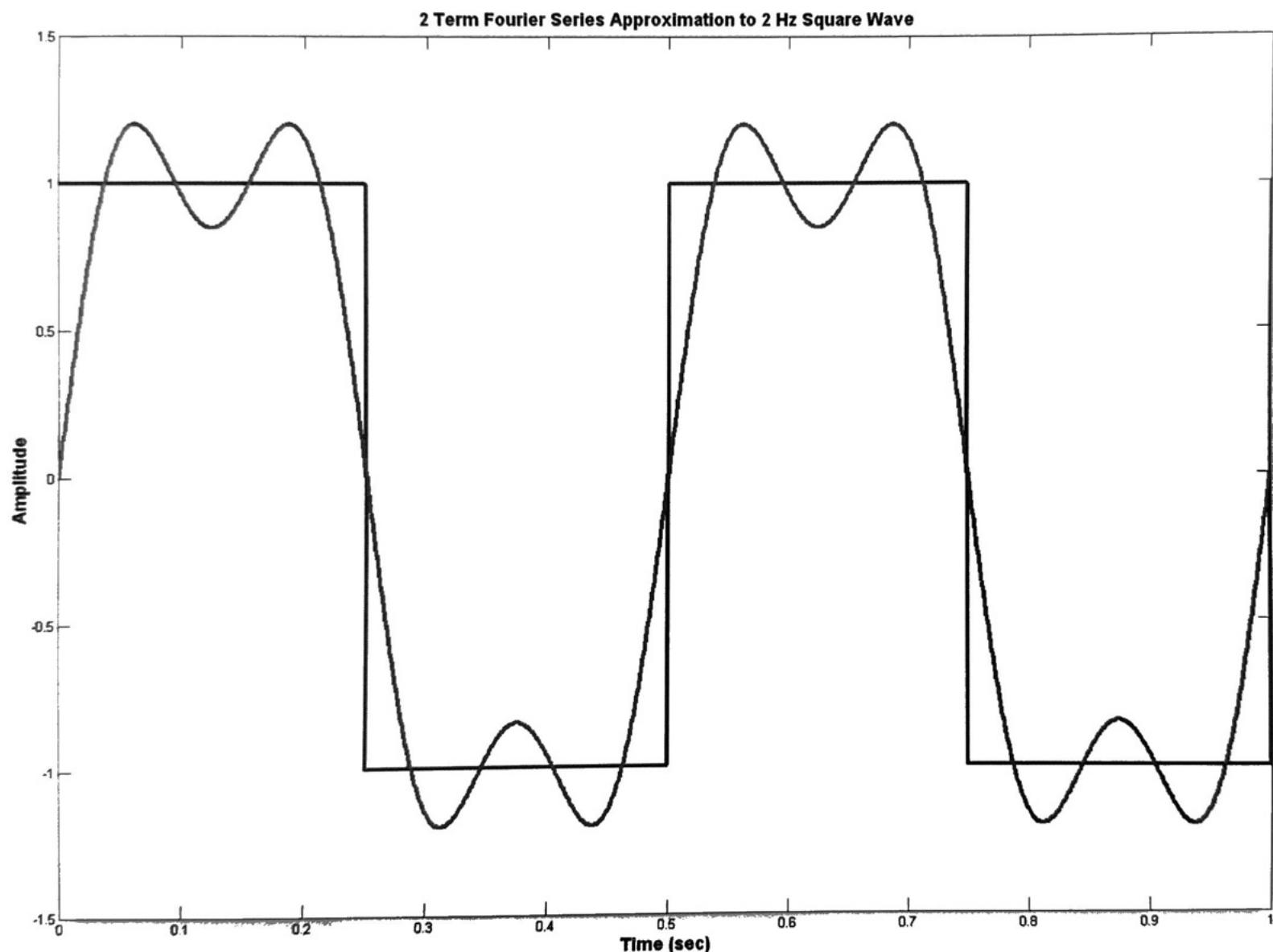
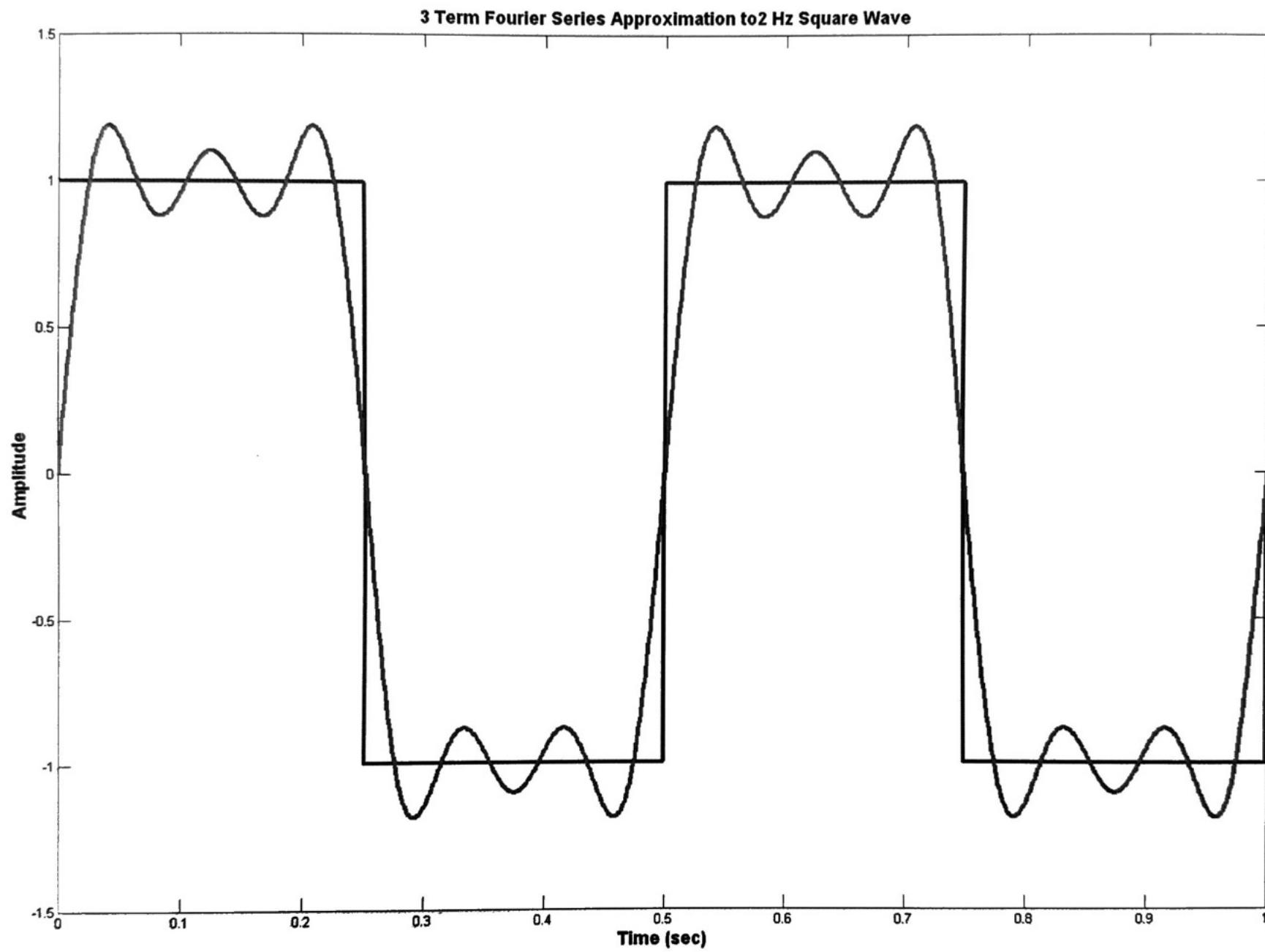


Figure No. 1

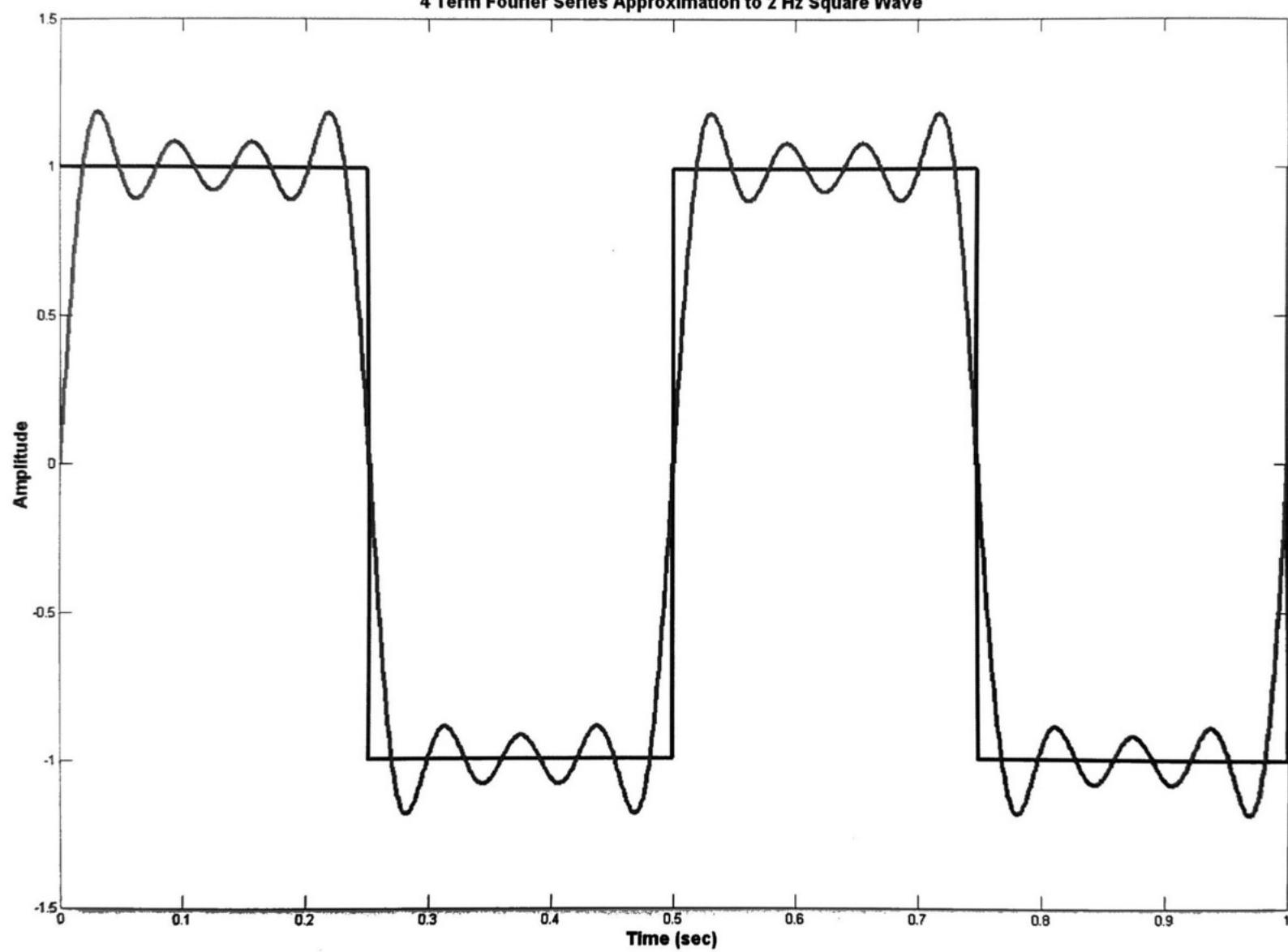
File Edit View Insert Tools Window Help

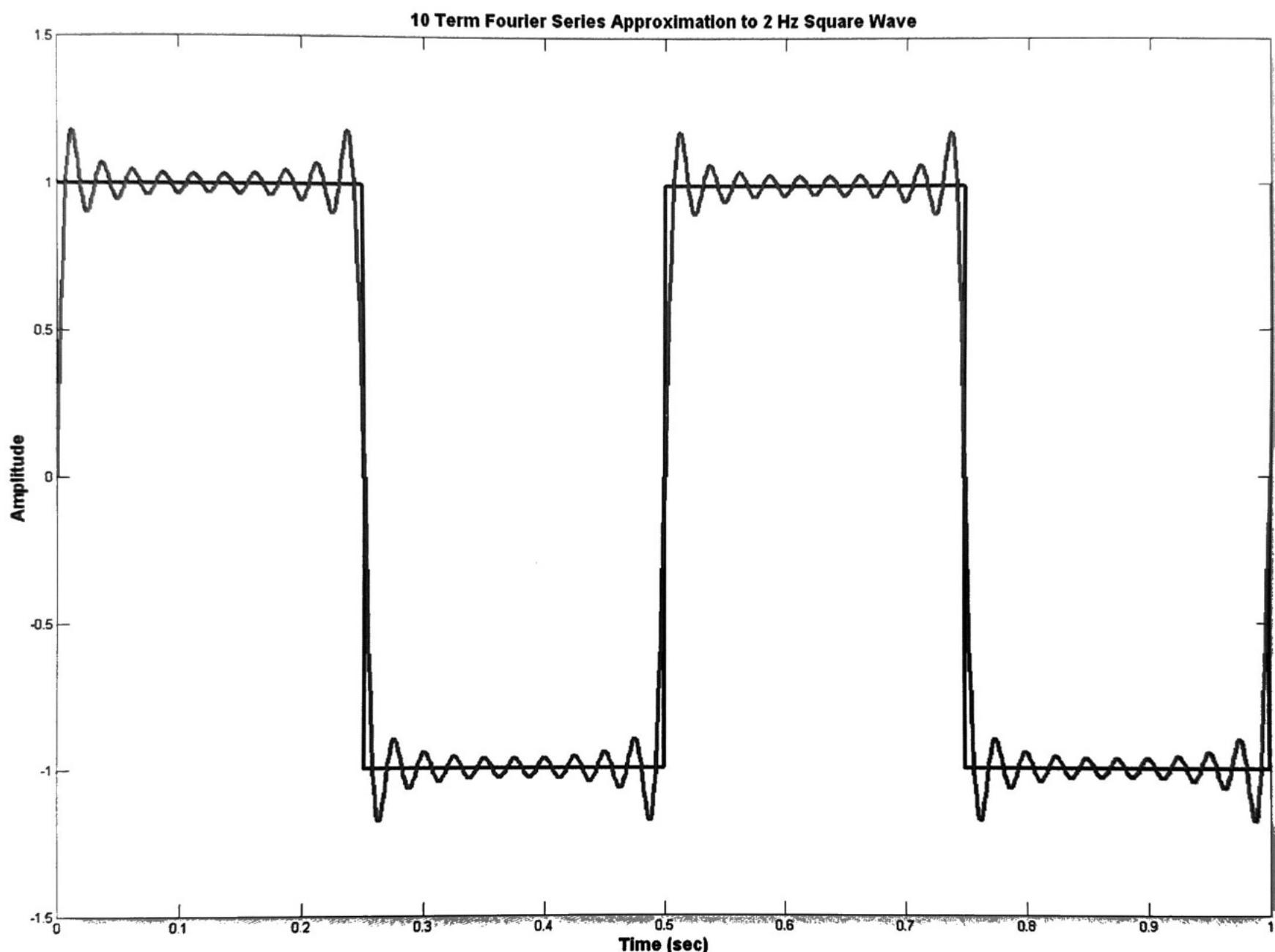






4 Term Fourier Series Approximation to 2 Hz Square Wave





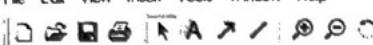
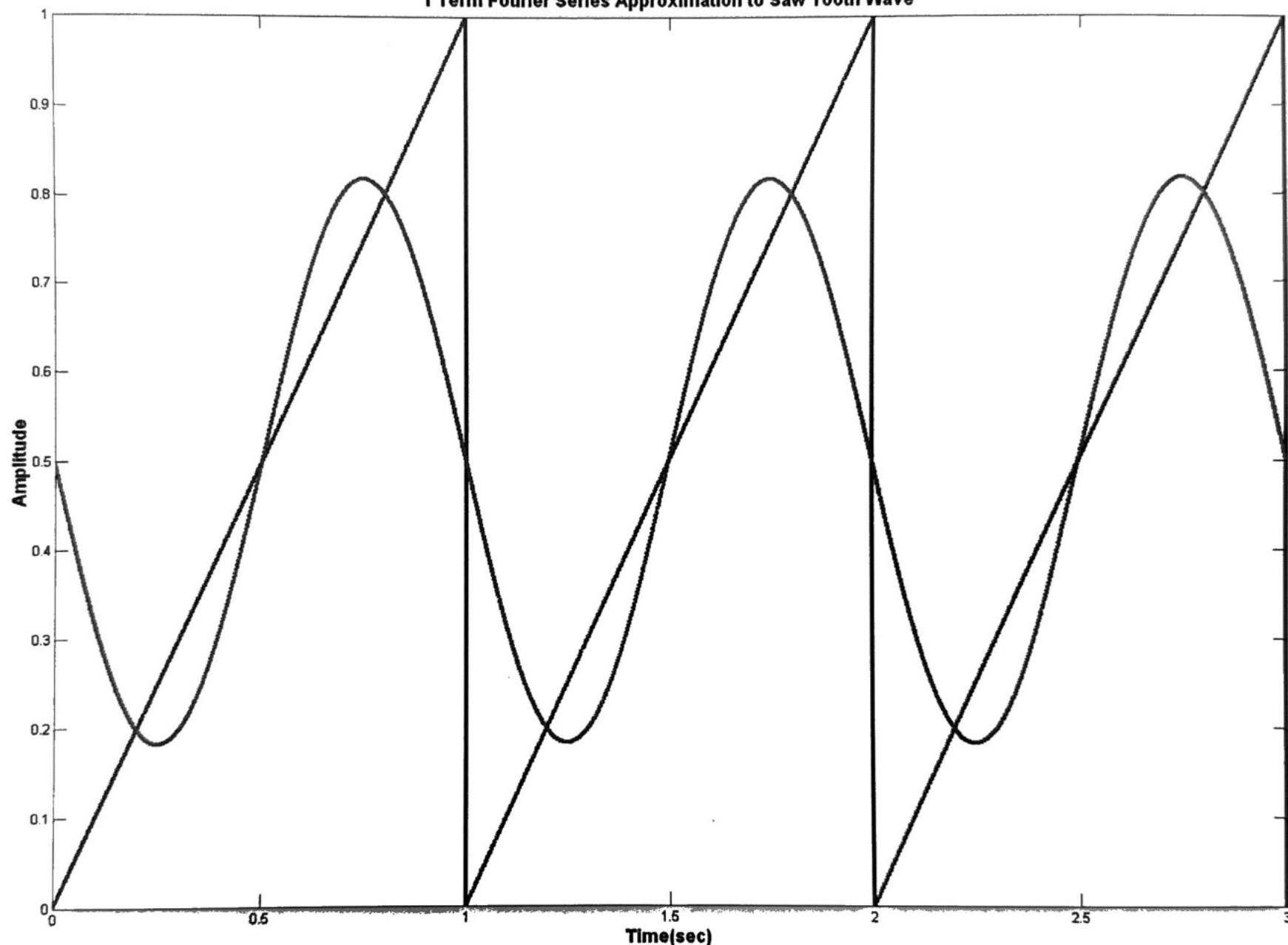
Periodic Saw Tooth Wave

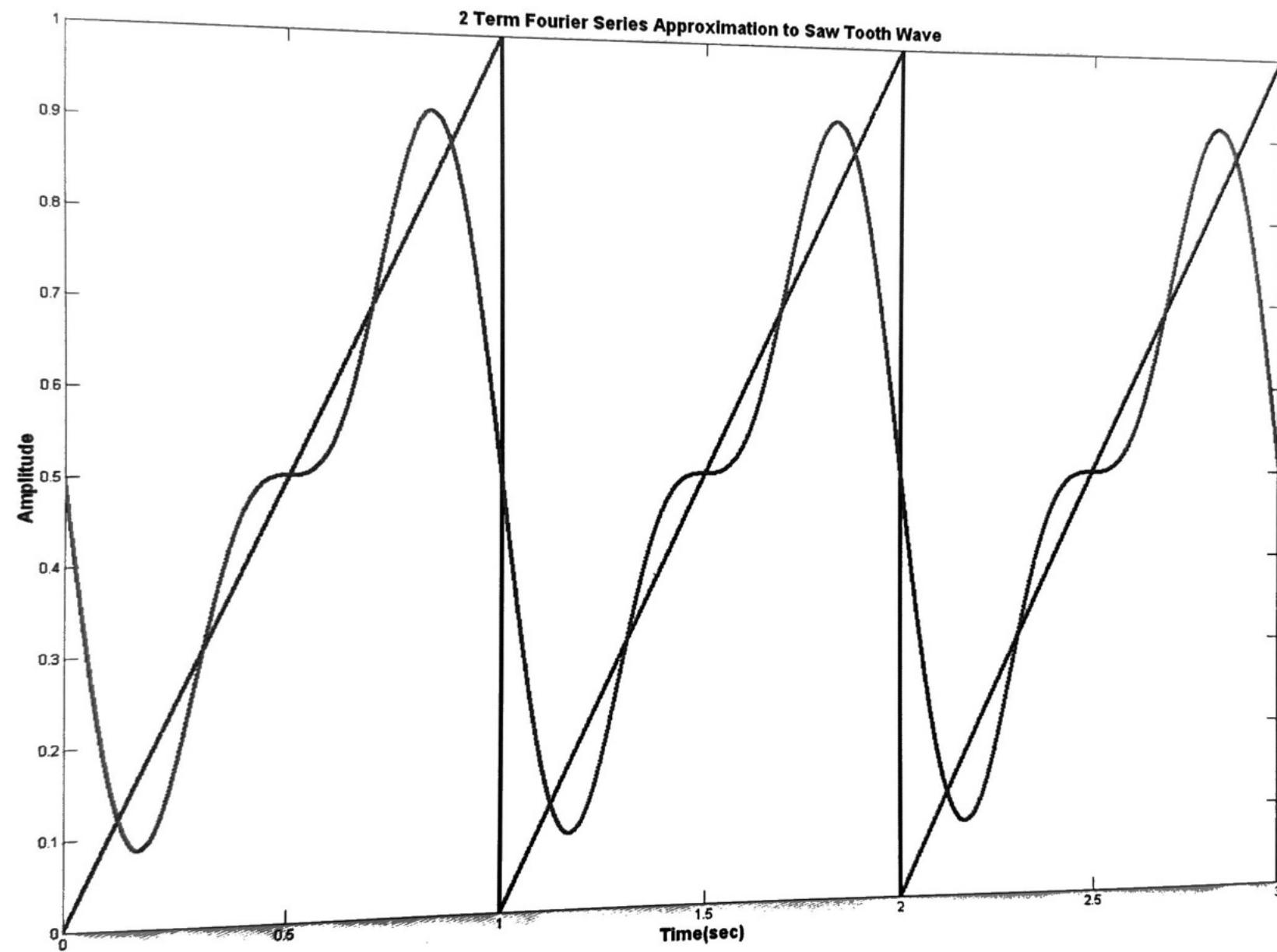
- Suppose $x(t)$ is a saw tooth wave with a period T and Amplitude 1.
- Fourier claims

$$x(t) = (T/2) - \sum_{n=1}^{\infty} \frac{T}{\pi n} \sin(n\omega_0 t)$$

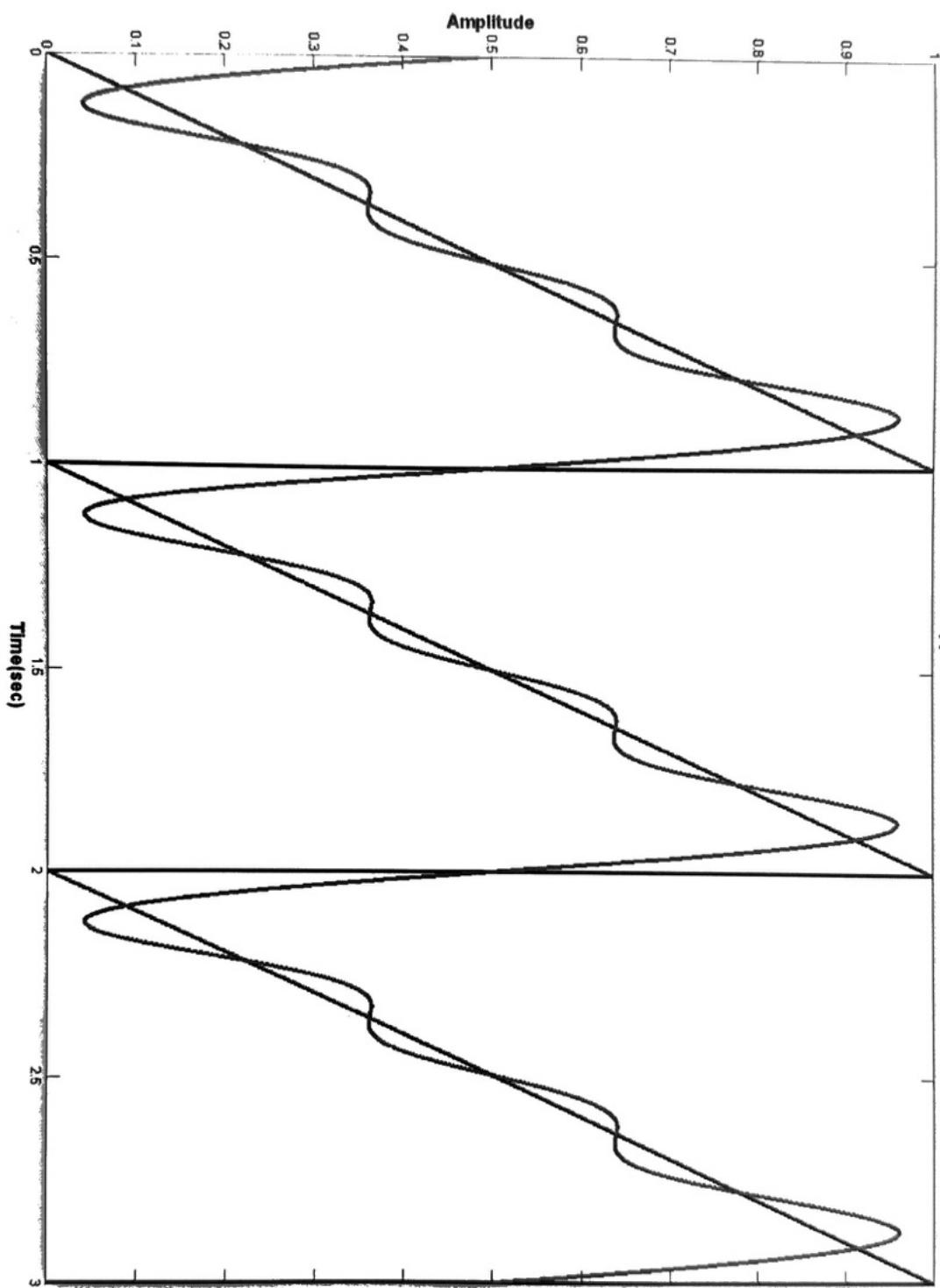
$$\Omega_0 = 2\pi/T$$

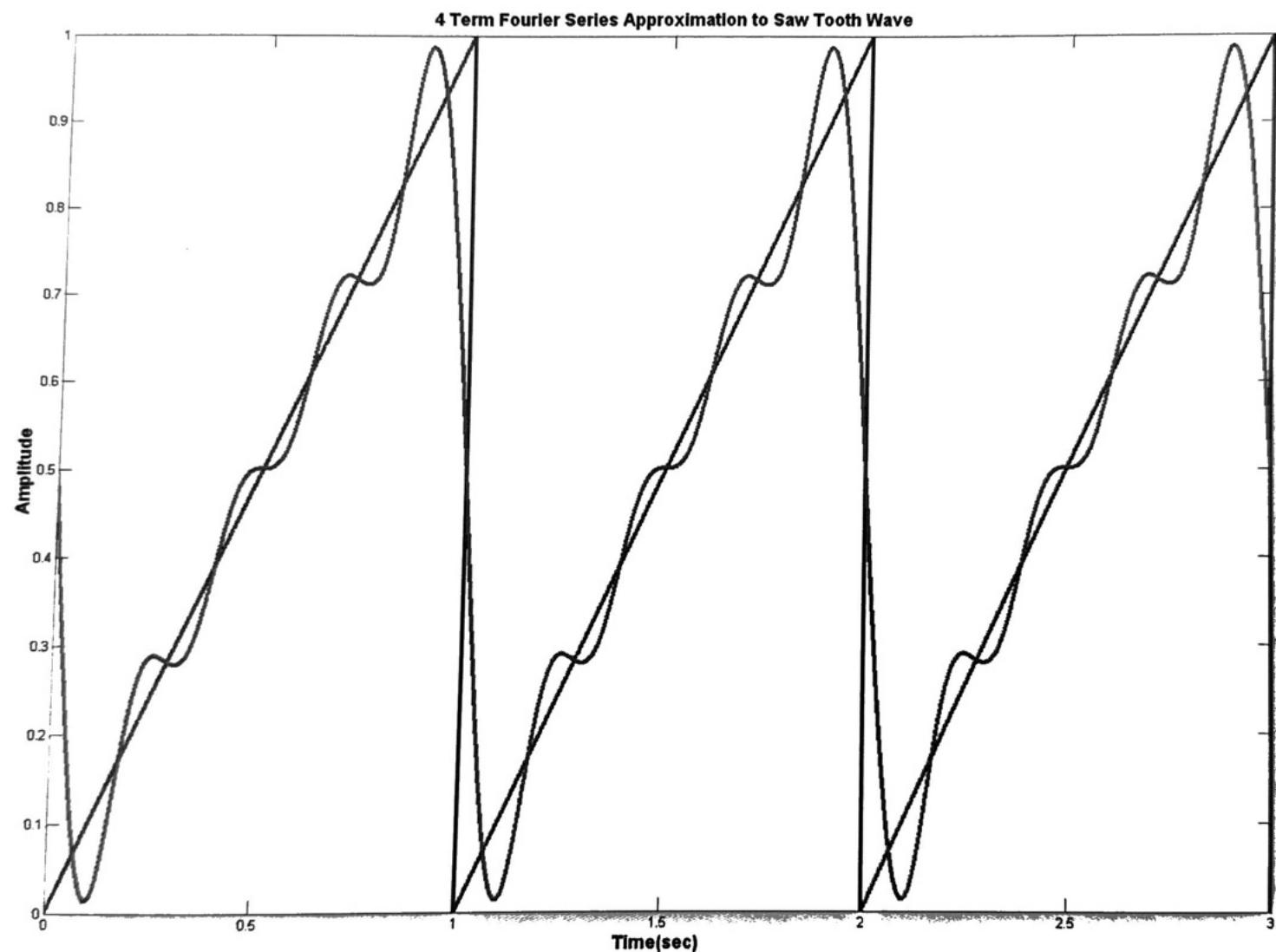
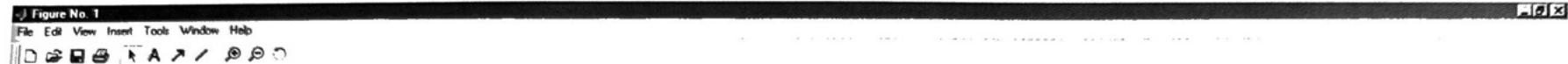
- $x(t) = (1/2) - [(T/\pi) \sin(\omega_0 t) + (T/2\pi) \sin(2\omega_0 t) \dots]$

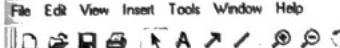
**1 Term Fourier Series Approximation to Saw Tooth Wave**



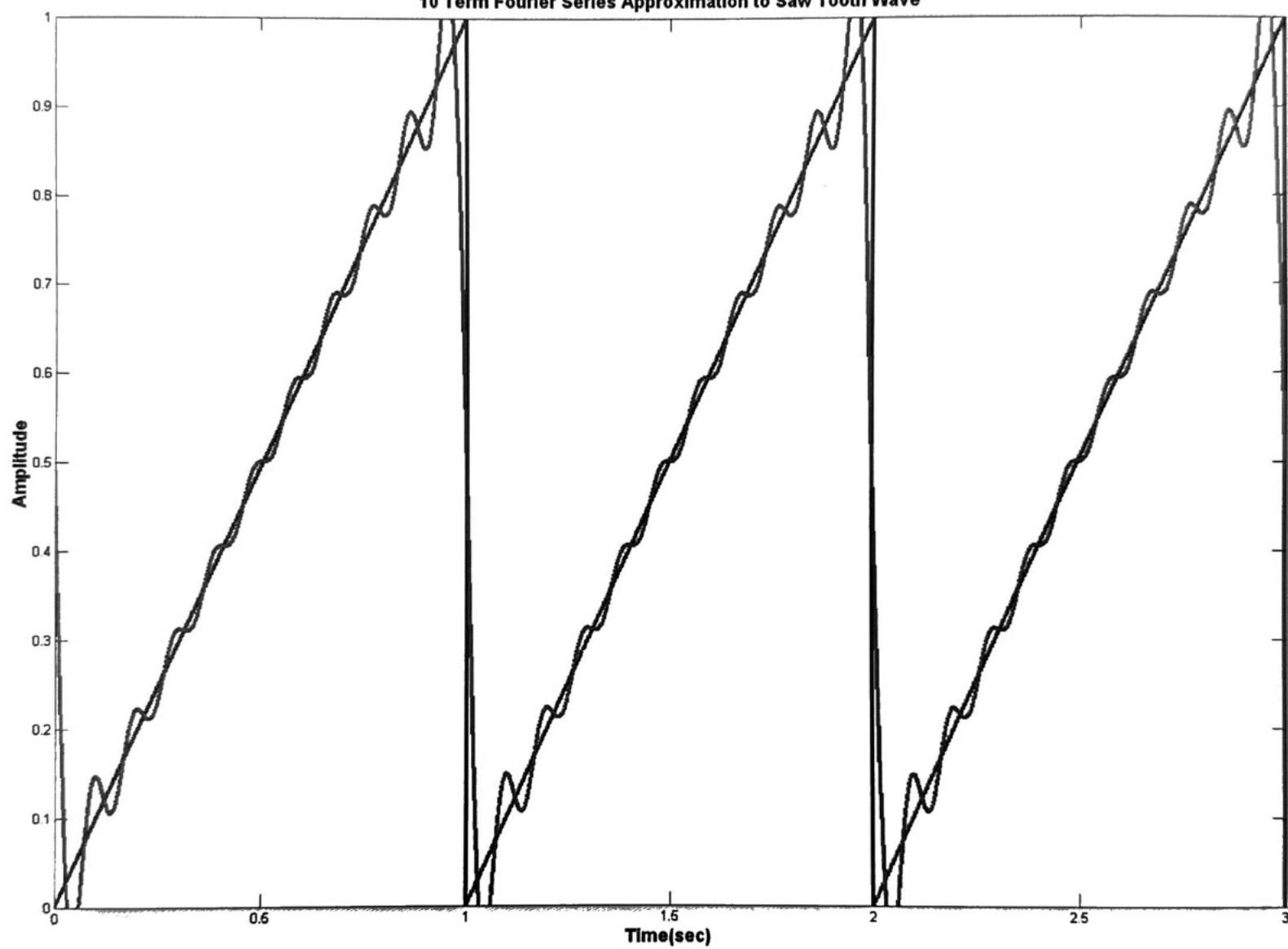
3 Term Fourier Series Approximation to Saw Tooth Wave







10 Term Fourier Series Approximation to Saw Tooth Wave



Periodic Triangle Wave

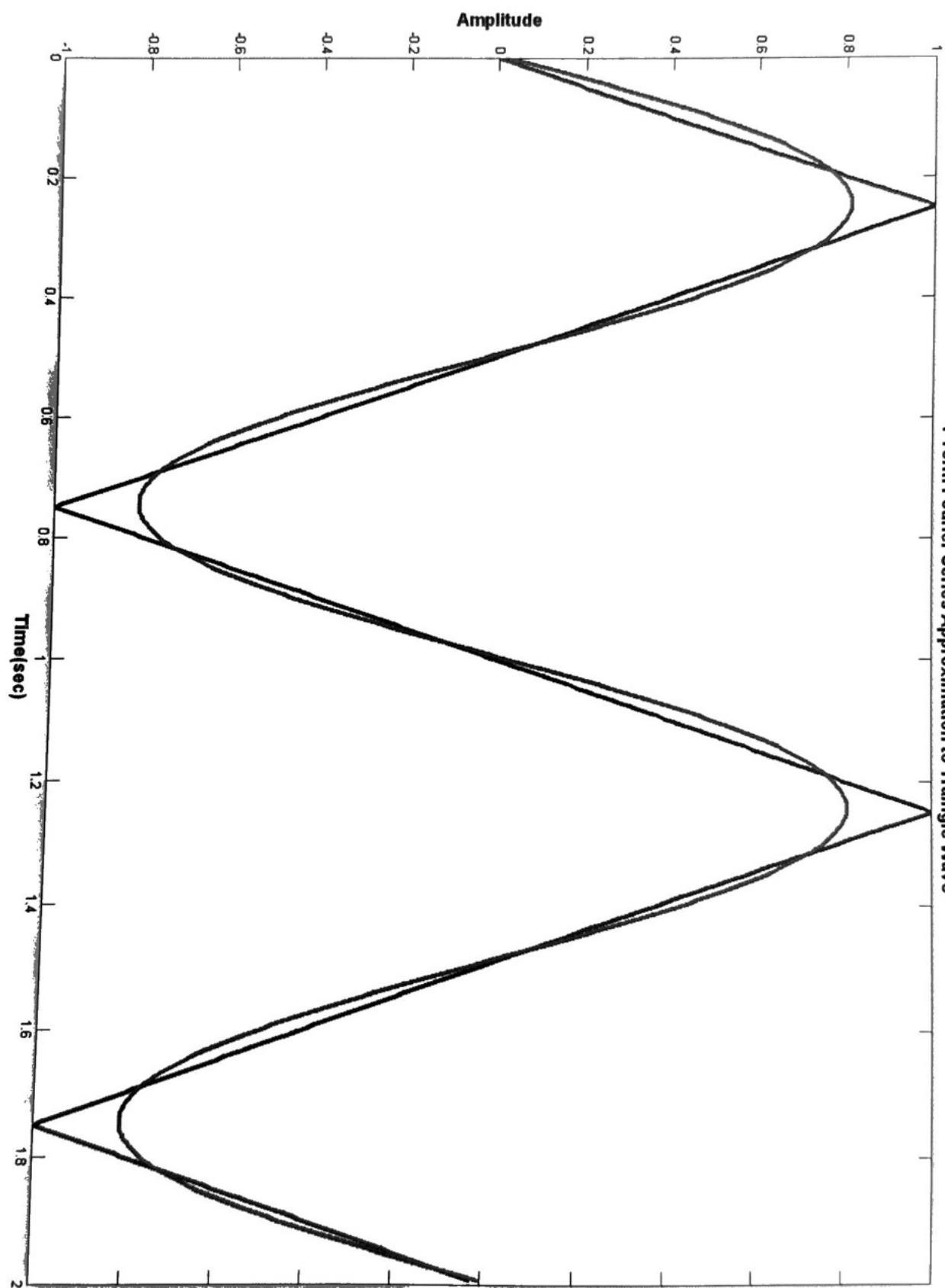
- Suppose $x(t)$ is a triangle wave with a period T and Amplitude 1.
- Fourier claims

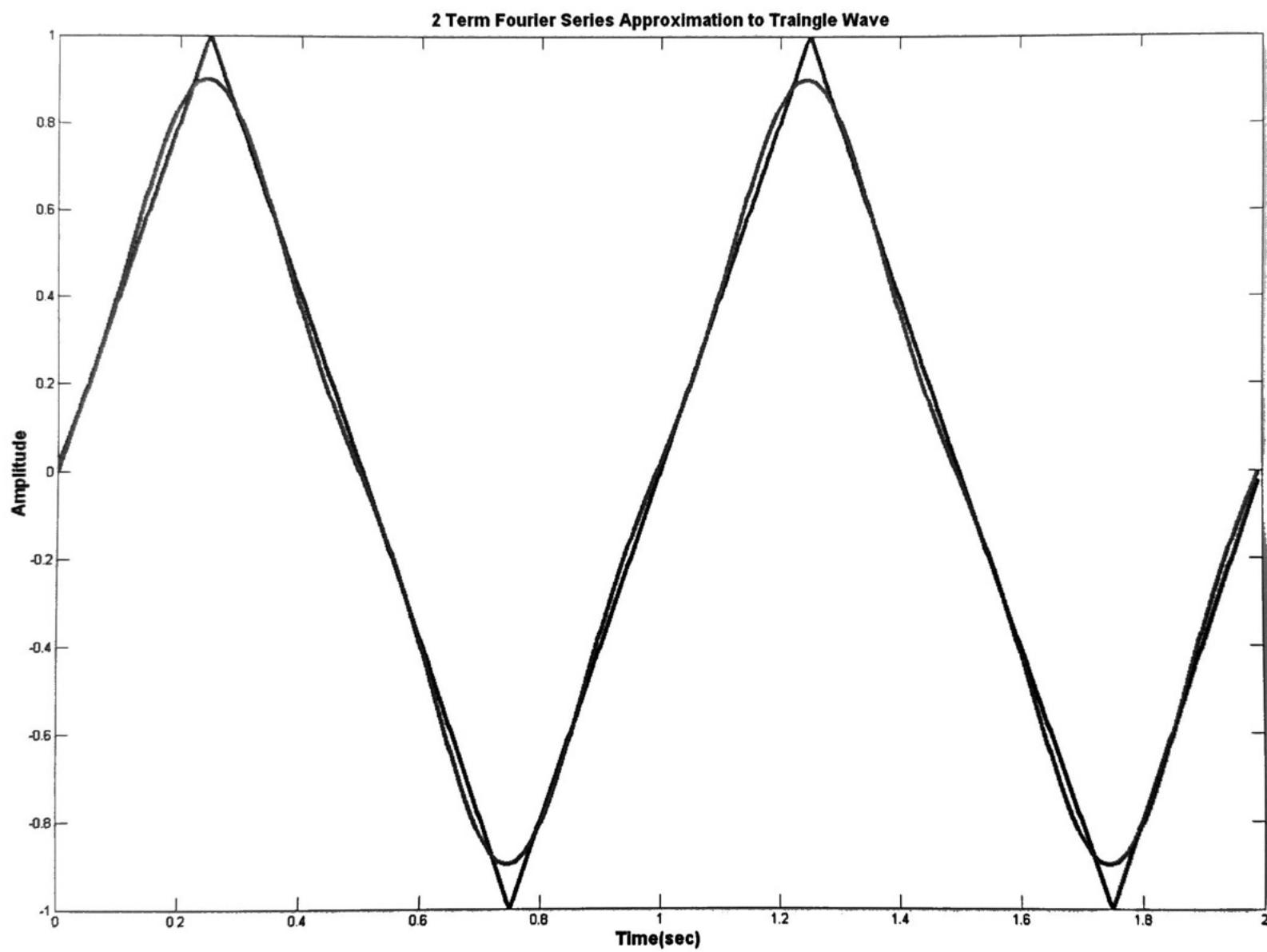
$$x(t) = \left(8/\pi^2\right) \sum_{n=1,3,5,7\dots} \frac{(-1)^{(n-1)/2}}{n^2} \sin(n\pi t/T)$$

$$\Omega_0 = 2\pi/T$$

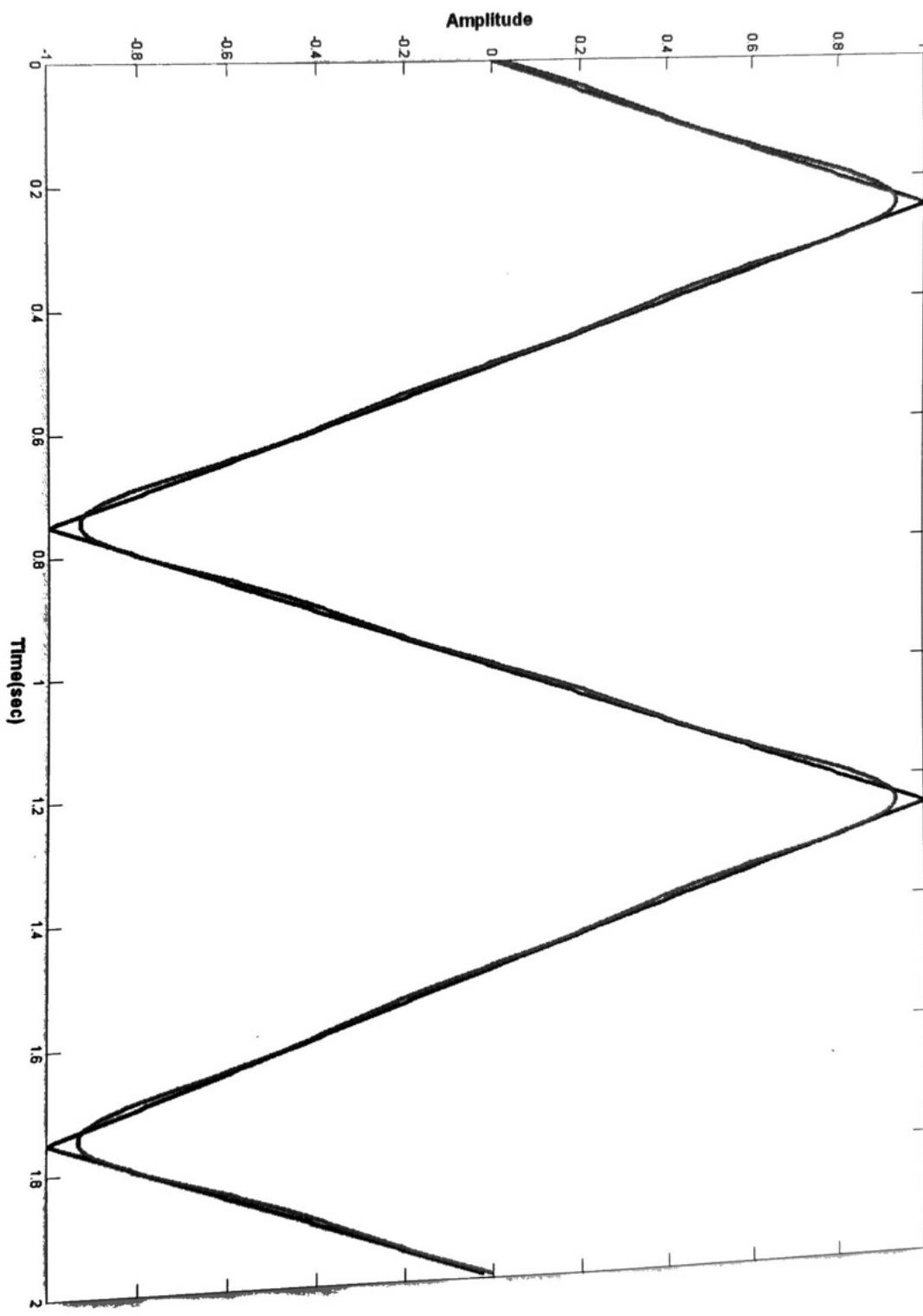
- $x(t) = (8/\pi^2) [\sin(\pi t/T) - (1/9) \sin(3\pi t/T) \dots]$

1 Term Fourier Series Approximation to Triangle Wave

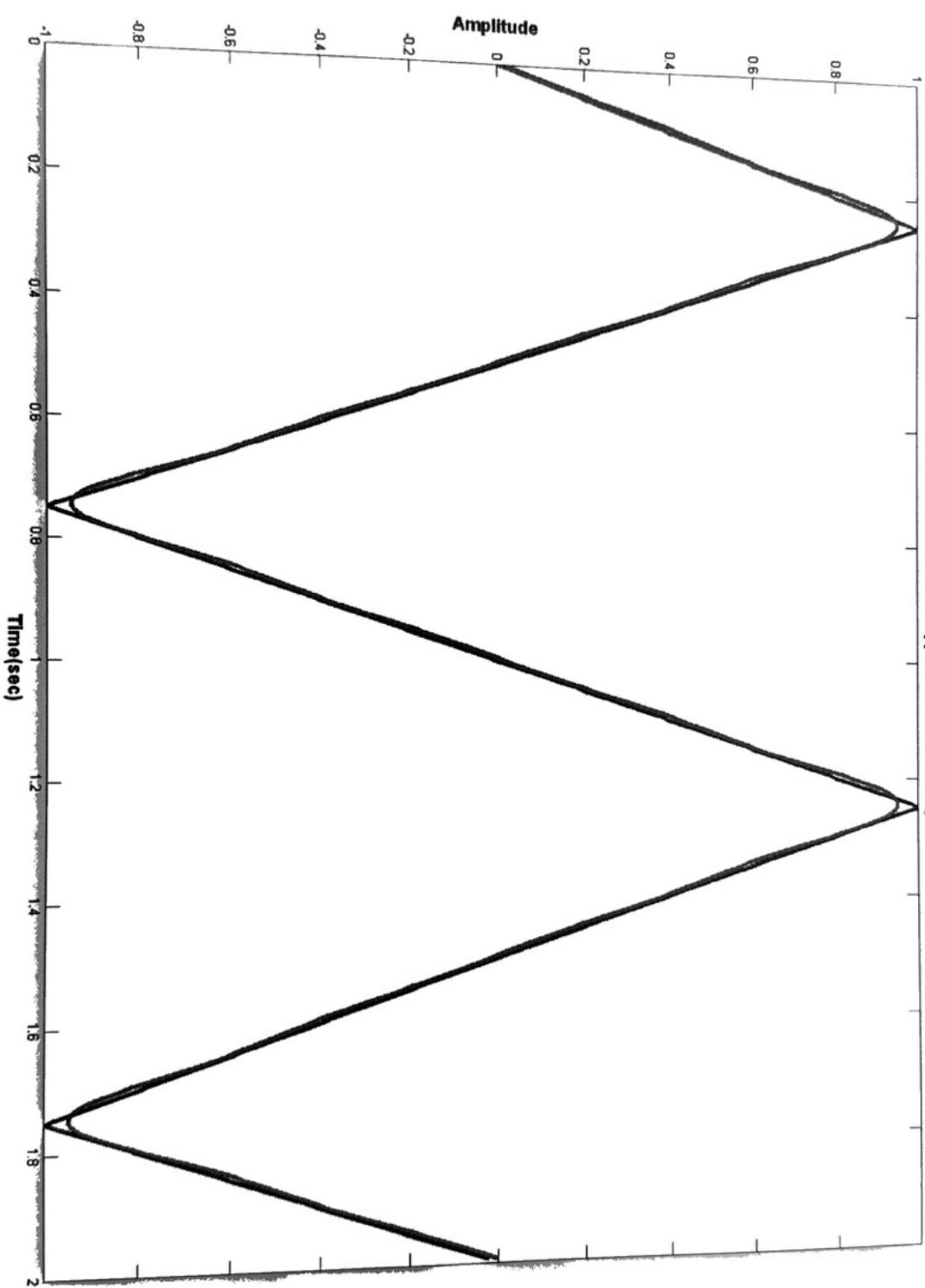


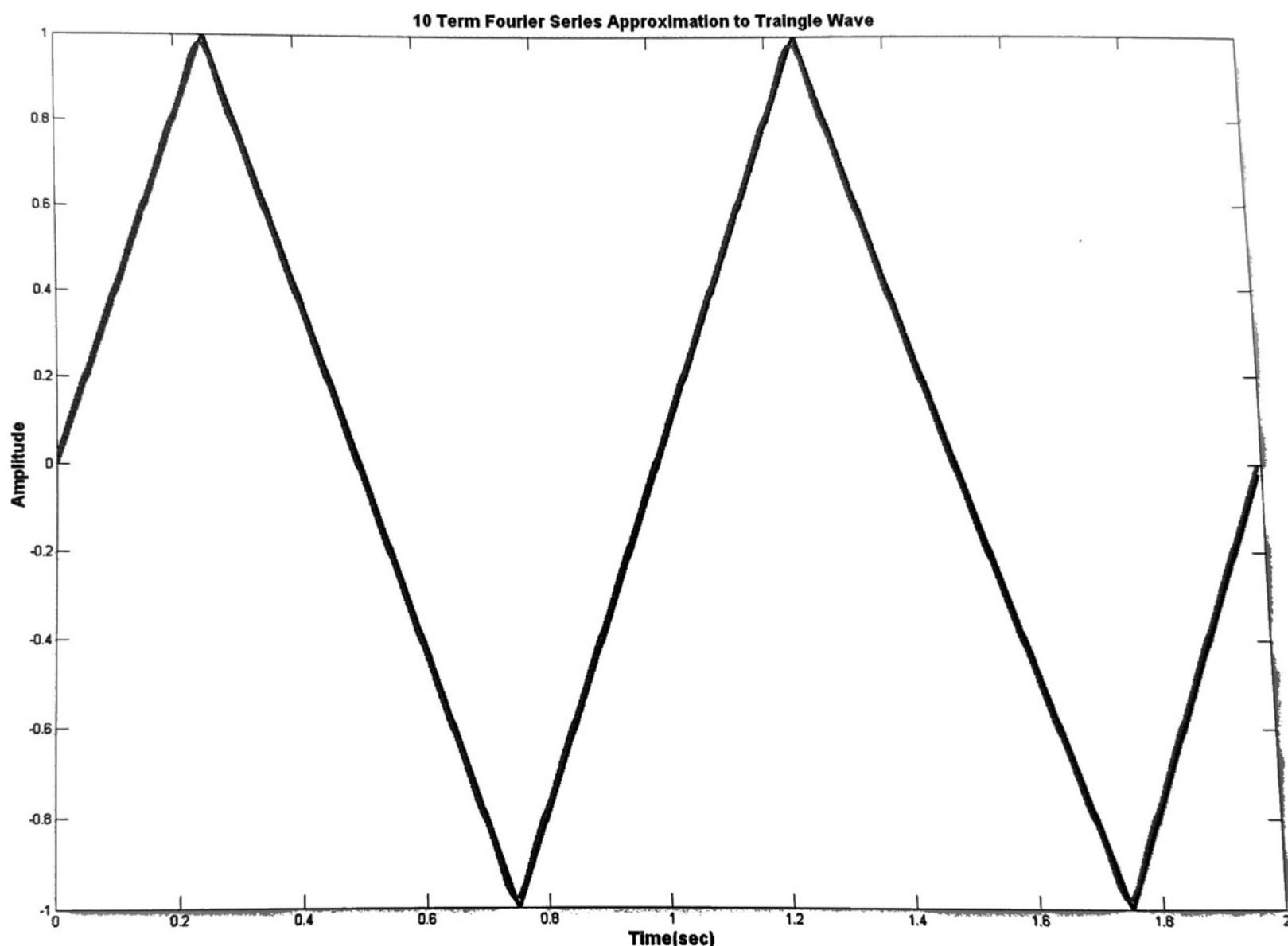


3 Term Fourier Series Approximation to Triangle Wave



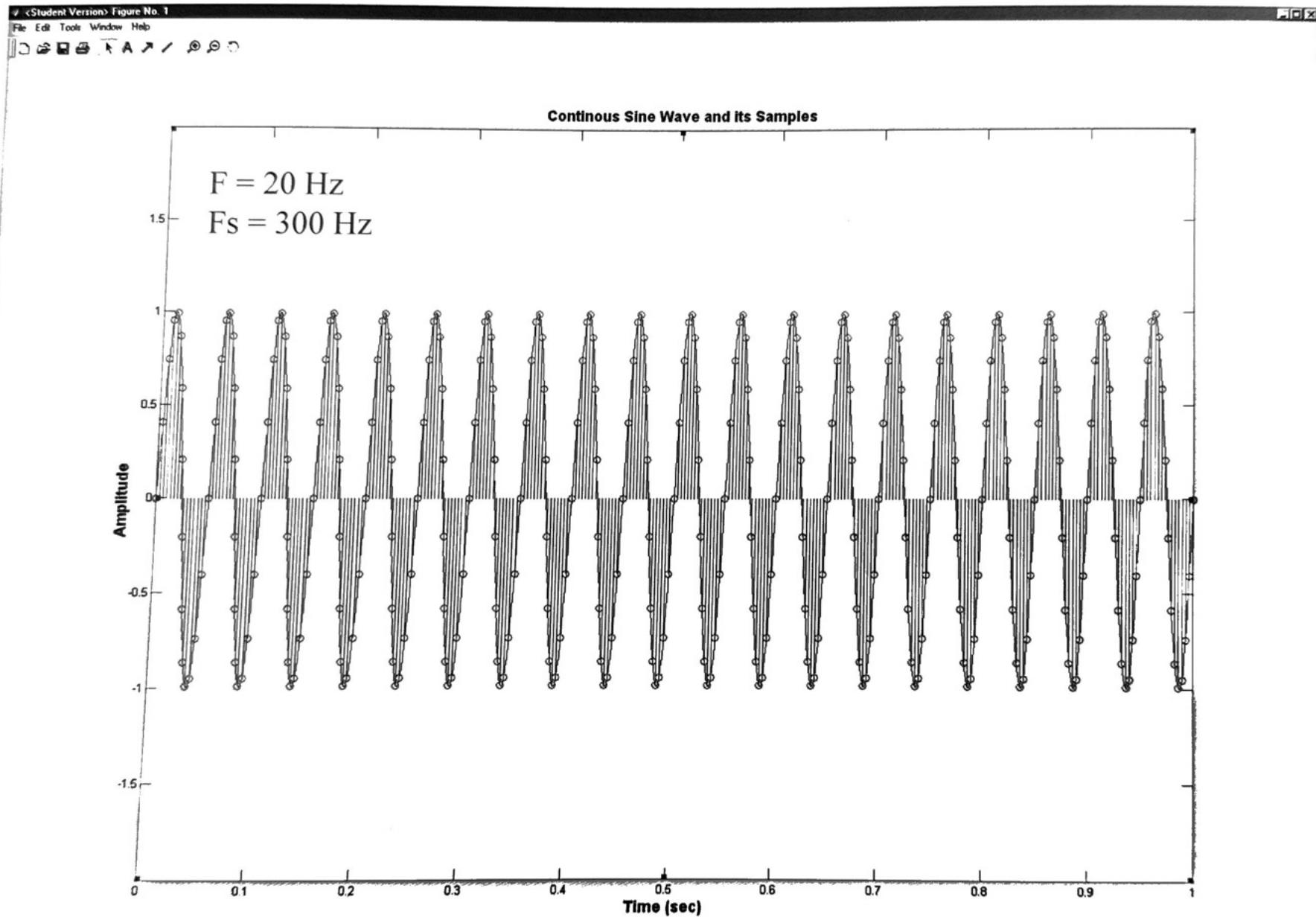
4 Term Fourier Series Approximation to Triangle Wave

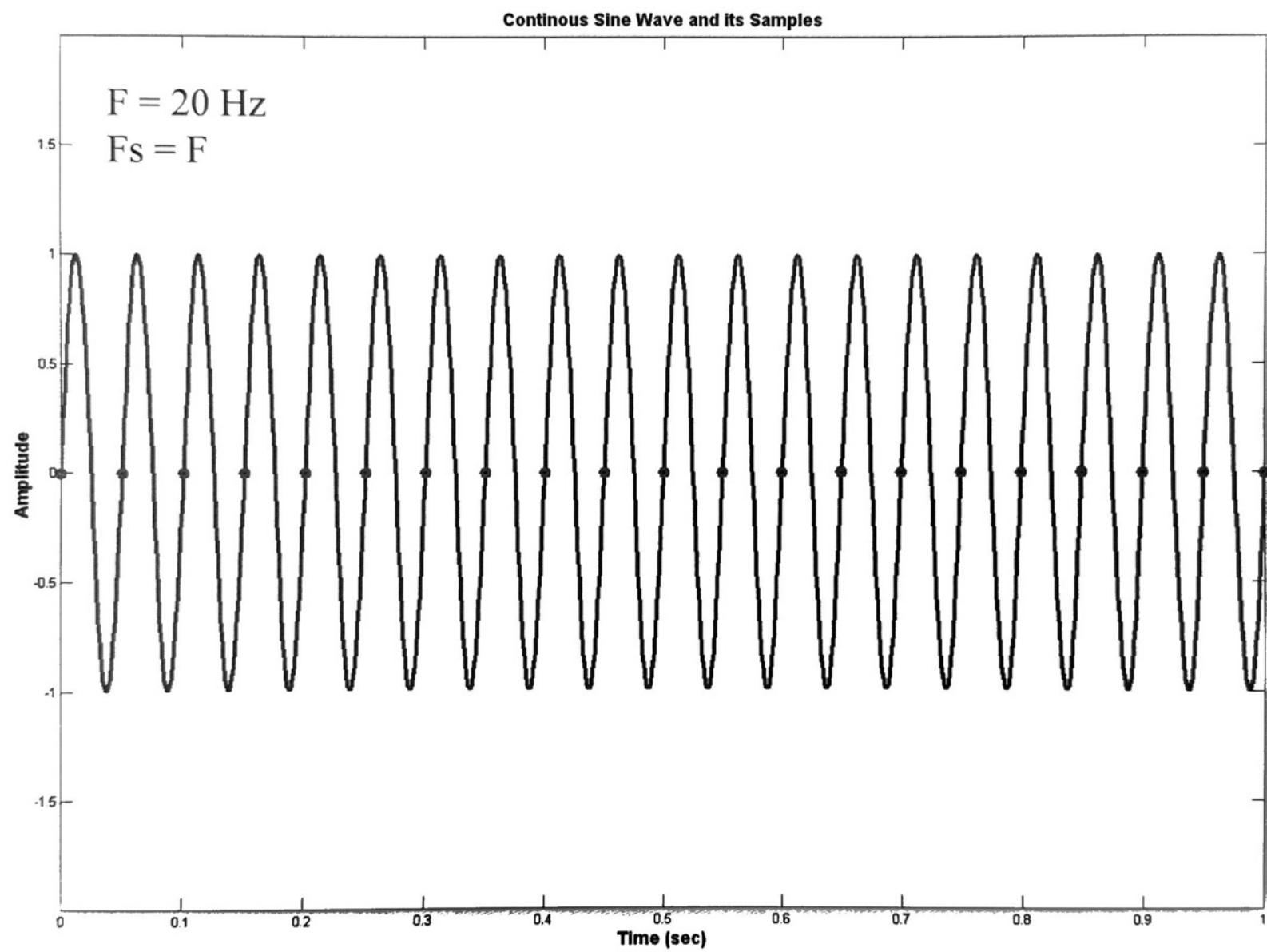


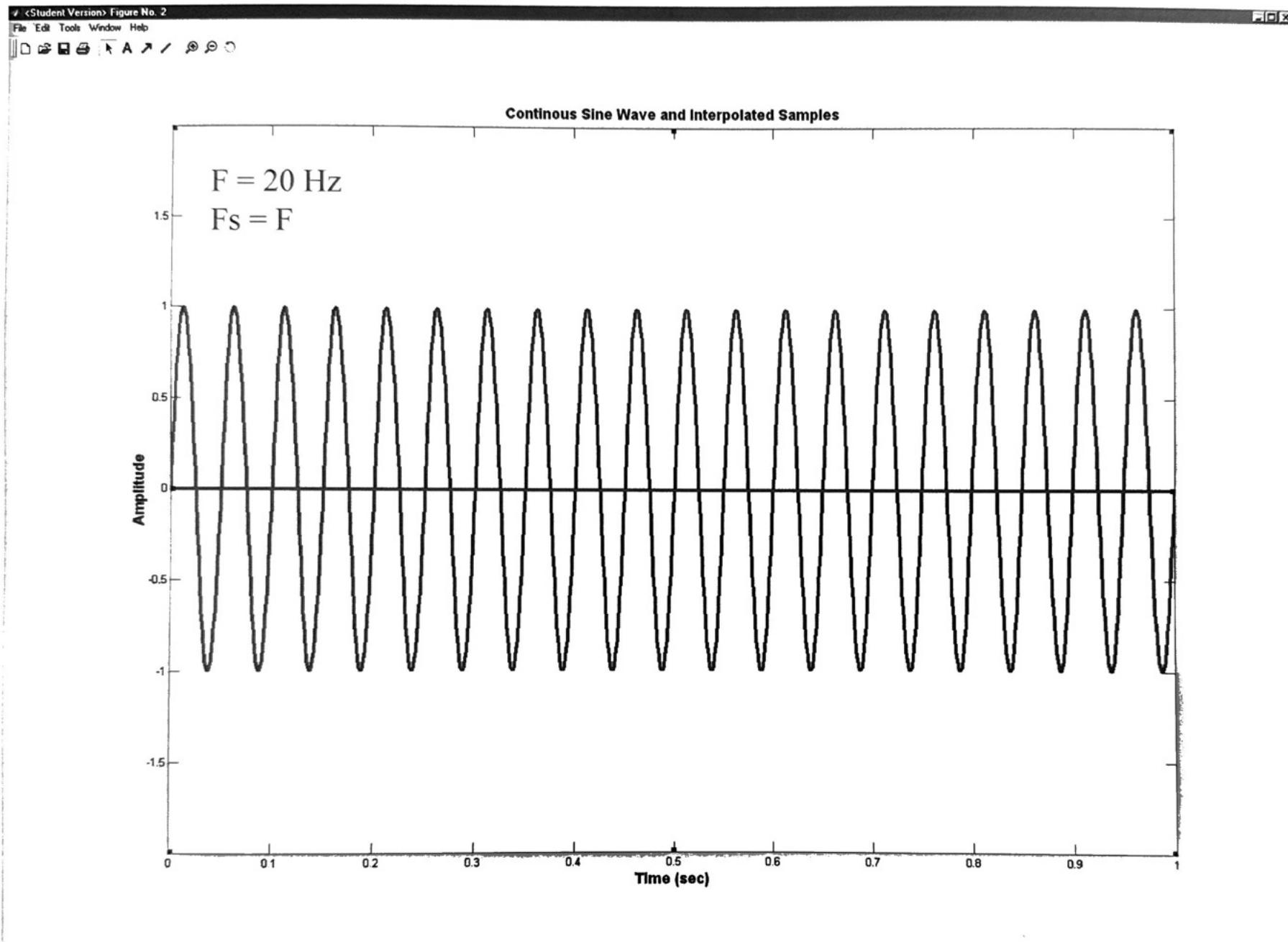


Sampling and Aliasing

- Sampling is the process by which a continuous-time (CT) signal is converted to a discrete-time (DT) signal.
- Nyquist indicates that we must sample a signal at a rate greater than twice its maximum bandwidth.
- Aliasing occurs when signal is sampled at a rate that does not meet the Nyquist criterion.

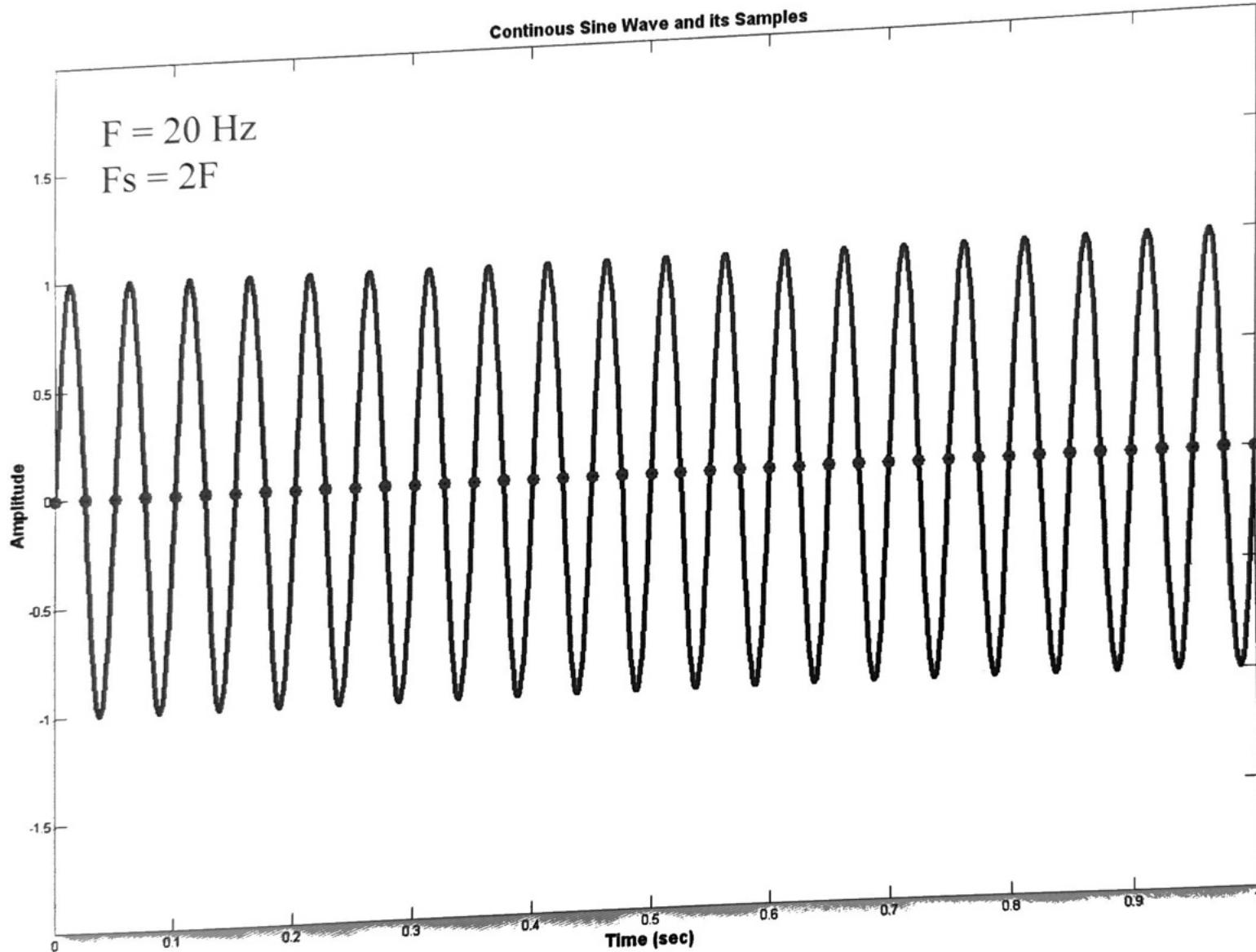


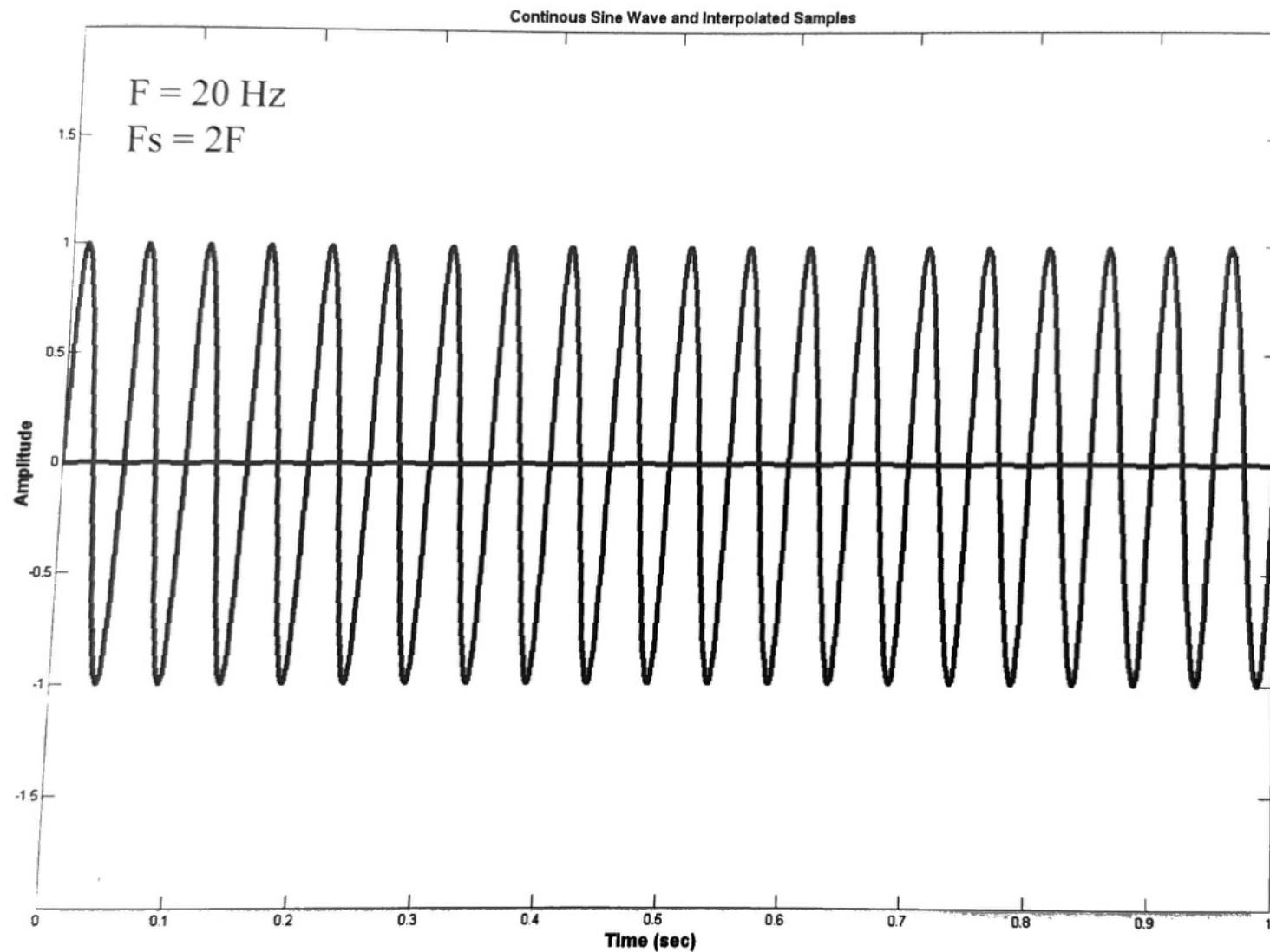


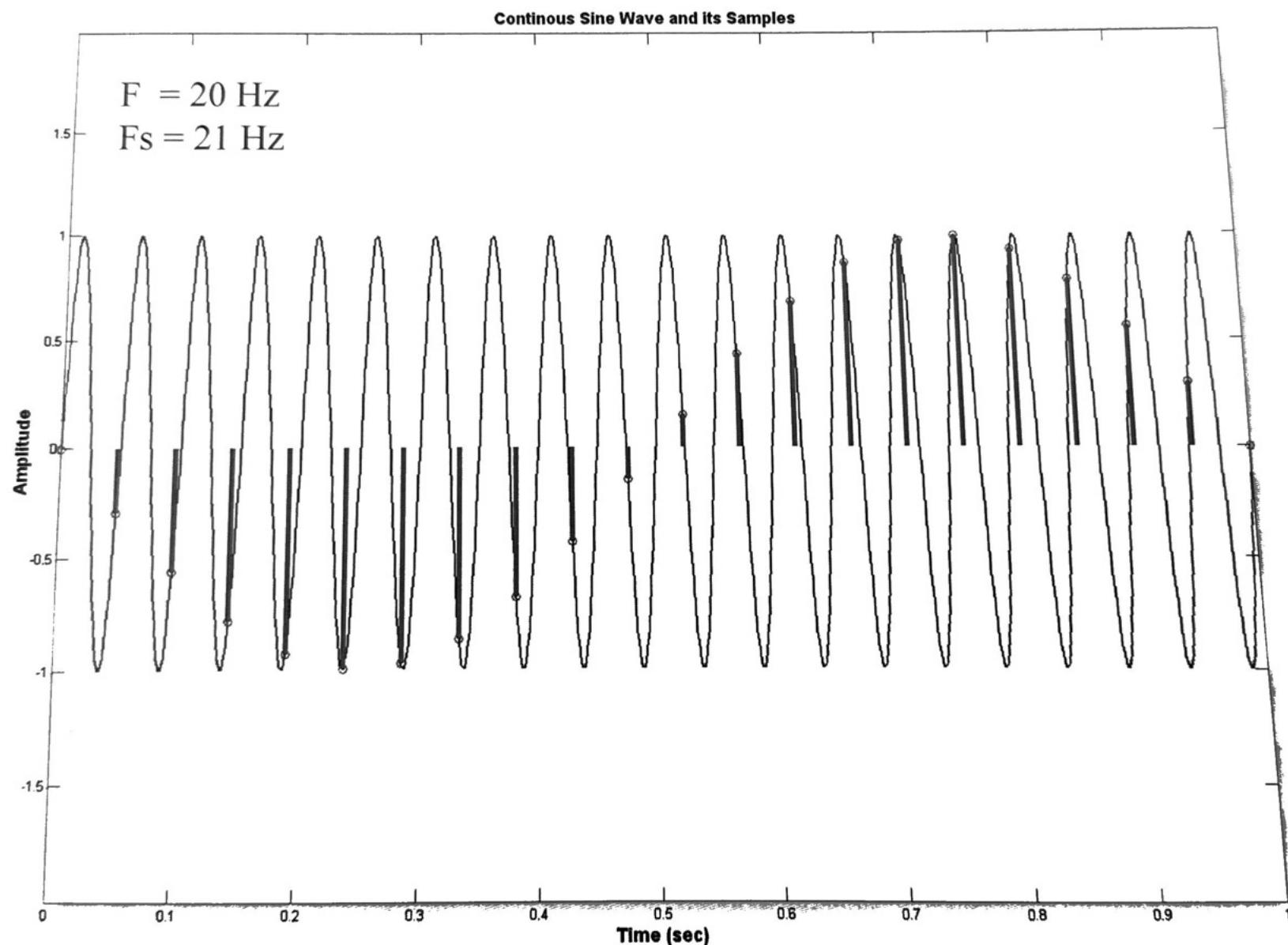


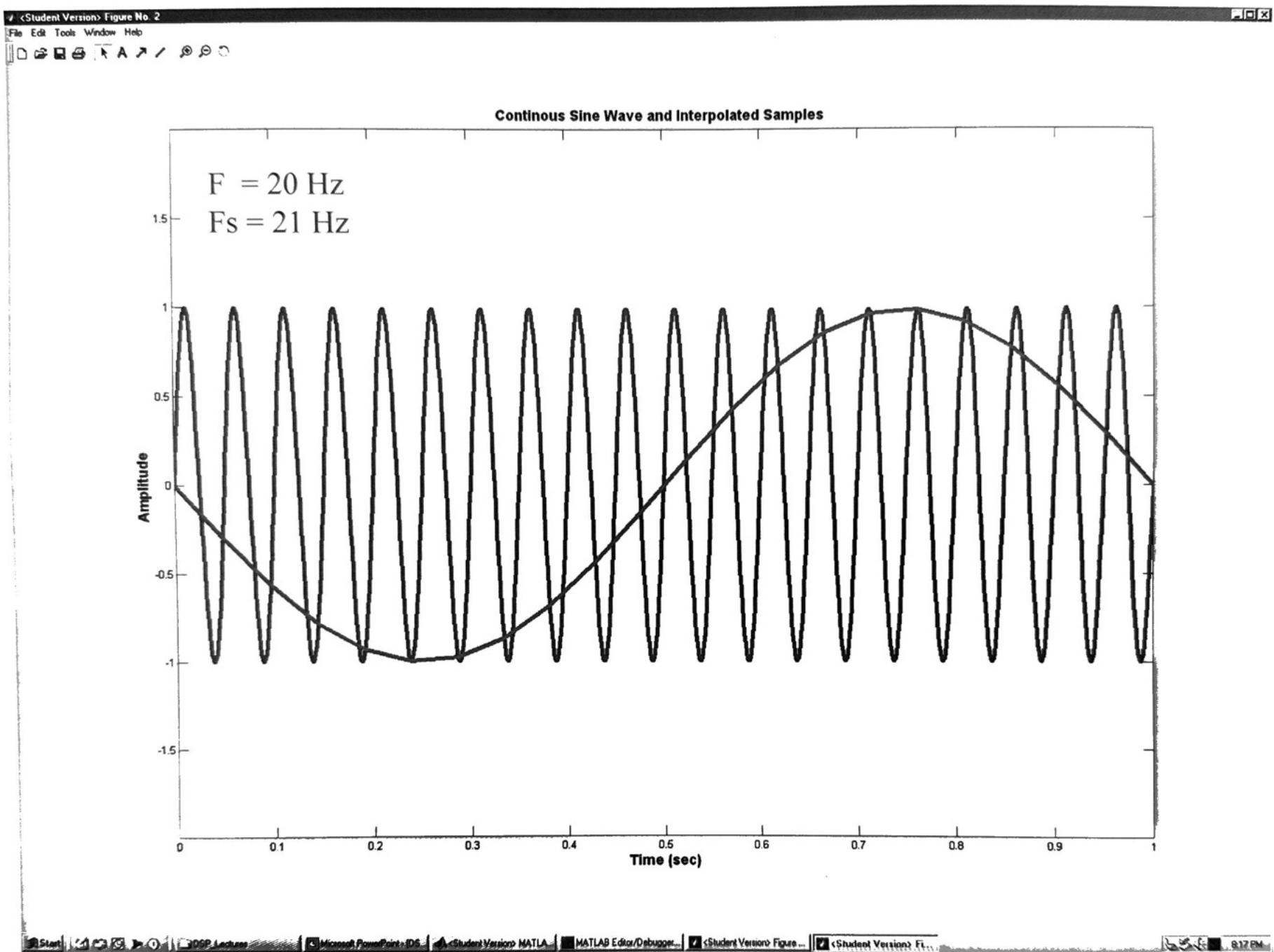


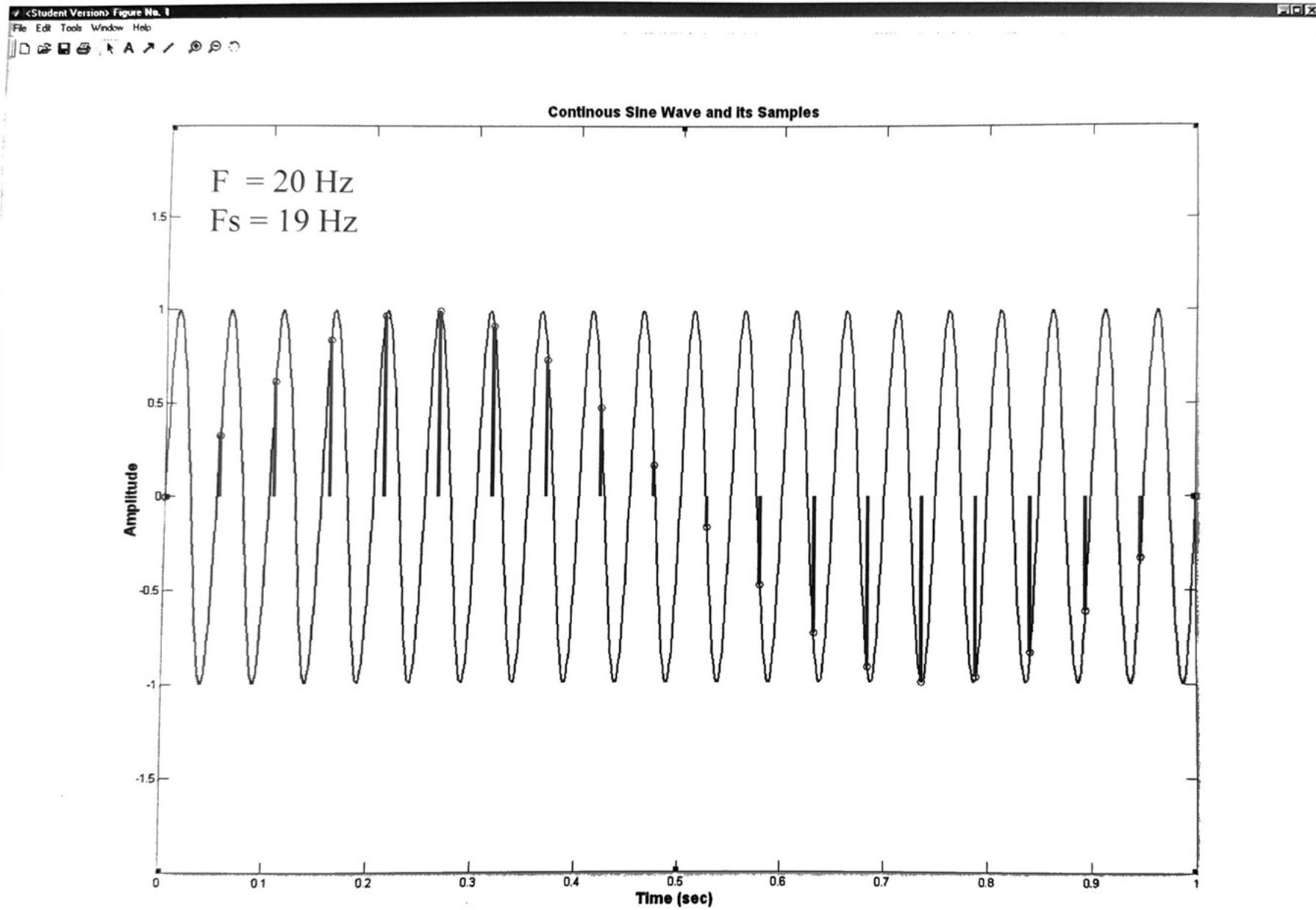
Continuous Sine Wave and its Samples

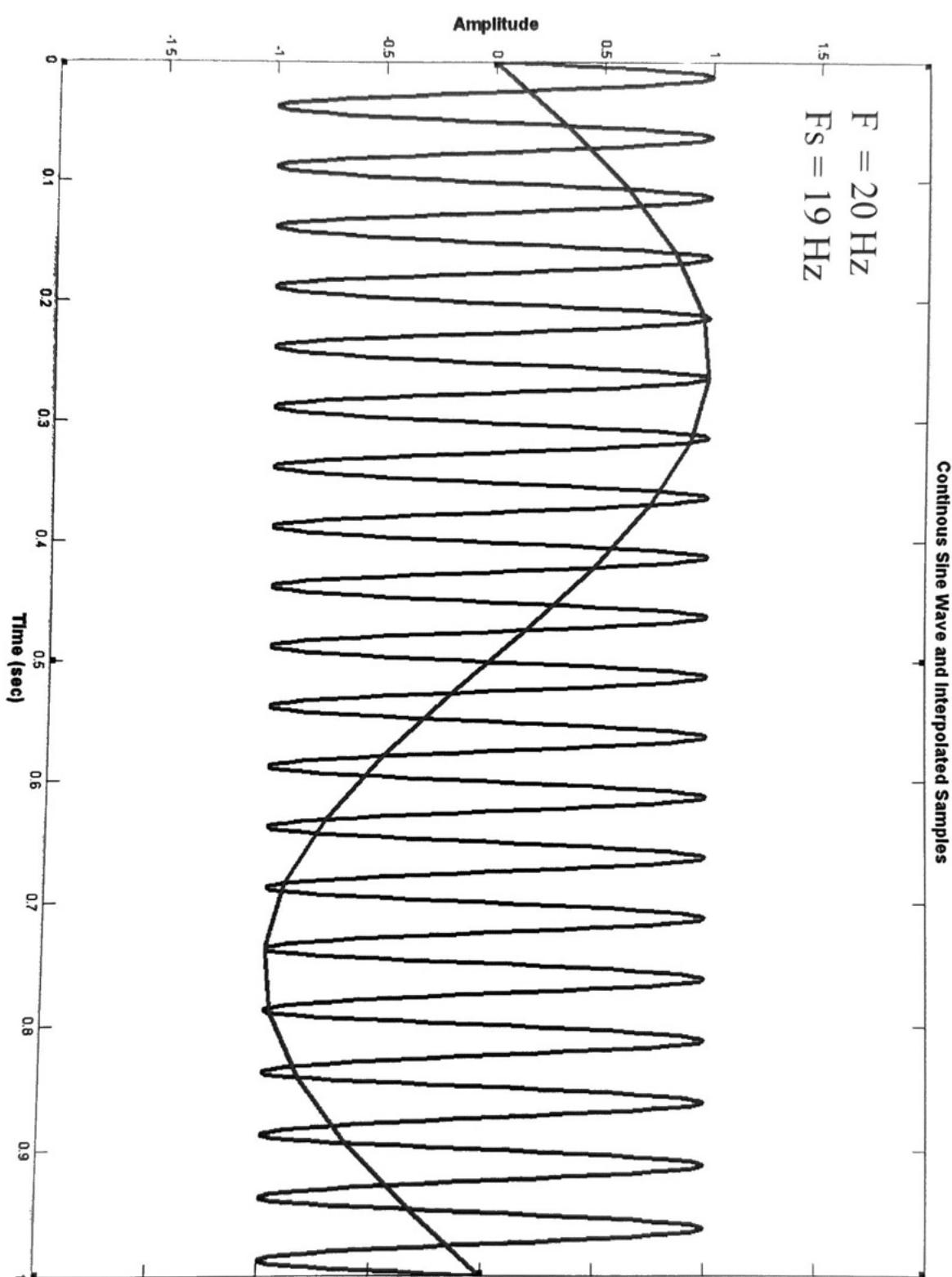






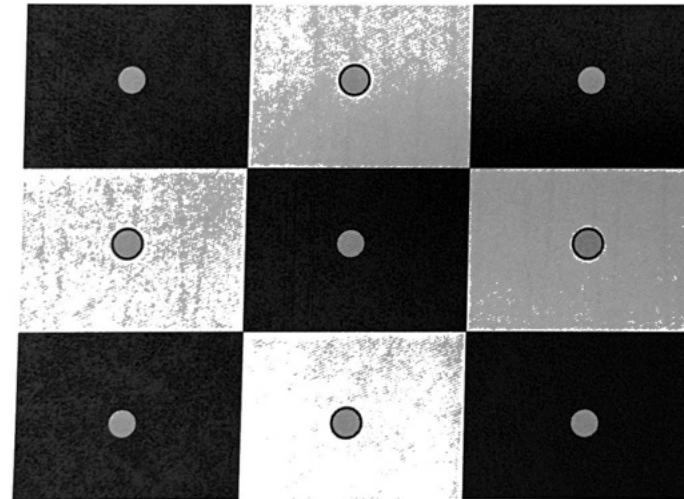






Sampling and Aliasing

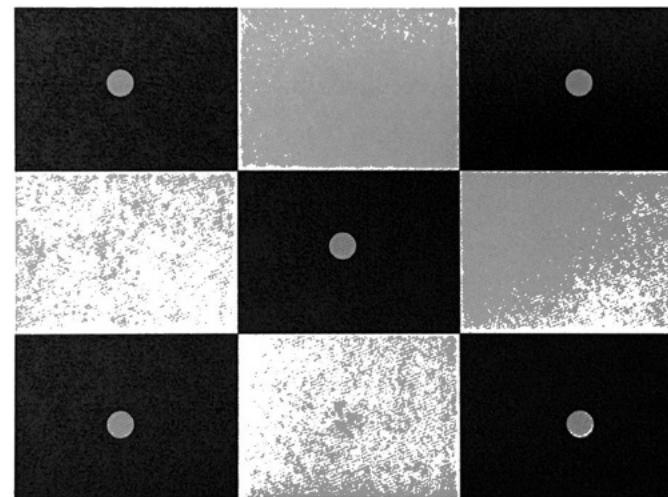
2D Sampling at Nyquist Rate



- Samples

Sampling and Aliasing

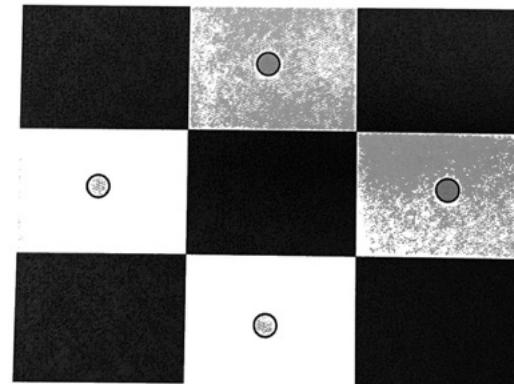
Aliasing in 2D Sampling Below Nyquist Rate



- Samples

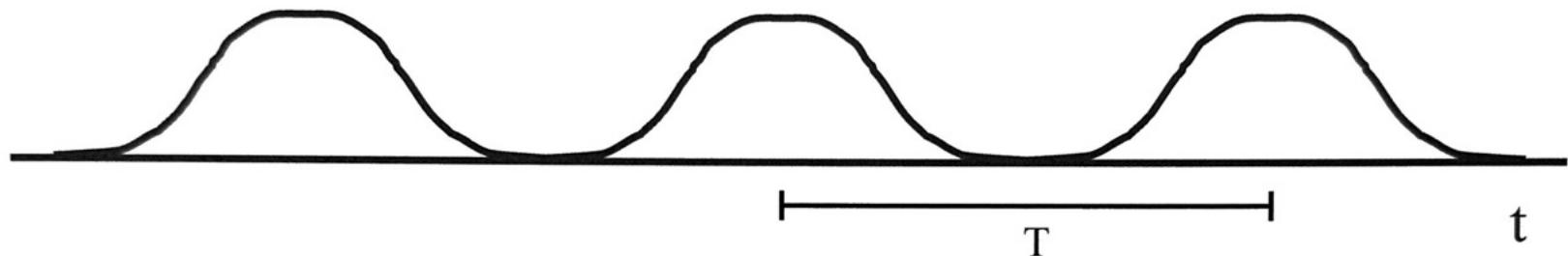
Sampling and Aliasing

Aliasing in 2D Sampling Below Nyquist Rate



- Samples

Periodic CT Signals



$$x(t) = a_0 + \sum_{k=1}^{\infty} \{ B_k \cos(k\Omega_0 t) + C_k \sin(k\Omega_0 t) \} \quad -\infty < t < \infty$$

$$\Omega_0 = 2\pi / T$$

Representing Finite Length Discrete Signals

- Since a discrete periodic signal $x[n]$ is just samples of a continuous periodic signal $x(t)$, we can figure out its frequency content by sampling the Continuous Fourier Series of the continuous periodic signal at the Nyquist Rate

$$x(t) = a_0 + \sum_{k=1}^{\infty} \{ B_k \cos(k\Omega_0 t) + C_k \sin(k\Omega_0 t) \} \quad \Omega_0 = 2\pi / T \quad -\infty < t < \infty$$

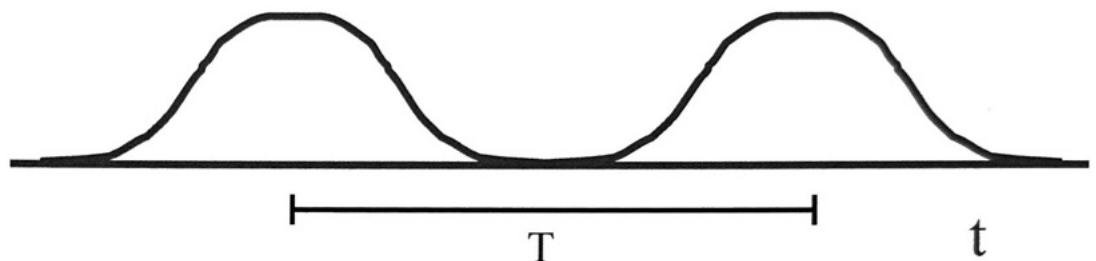
*This can't be true
because of the
Nyquist Sampling
Criterion*

↓
Sample:
 $t = n T_{\text{sample}}$
 $T = N T_{\text{sample}}$

$$x[n] = a_0 + \sum_{k=1}^{\infty} \{ B_k \cos[k\omega_0 n] + C_k \sin[k\omega_0 n] \} \quad \omega_0 = 2\pi / N \quad -\infty < n < \infty$$

Implications of Nyquist Sampling Criterion

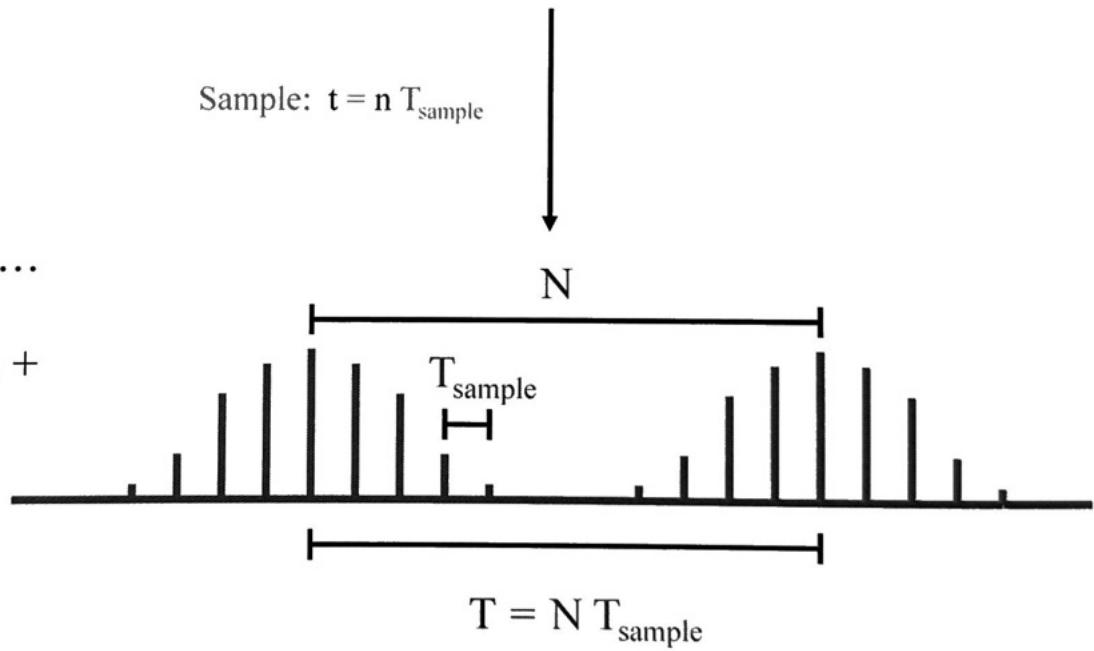
- Nyquist Sampling Criterion requires highest frequency in continuous signal to be $\leq F_{\text{sample}}/2$



- Continuous Fourier Series can express continuous signal as

$$a_0 + B_1 \cos \Omega_0 t + B_2 \cos 2\Omega_0 t + \dots + B_{\text{Nyquist}} \cos \Omega_{\text{Nyquist}} t + C_1 \sin \Omega_0 t + C_2 \sin 2\Omega_0 t + \dots + C_{\text{Nyquist}} \sin \Omega_{\text{Nyquist}} t$$

Sample: $t = n T_{\text{sample}}$



$$\Omega_{\text{Nyquist}} = 2\pi (F_{\text{sample}}/2)$$

Representing Finite Length Discrete Signals

- The Fourier Series for the replicated discrete signal $x[n]$ is

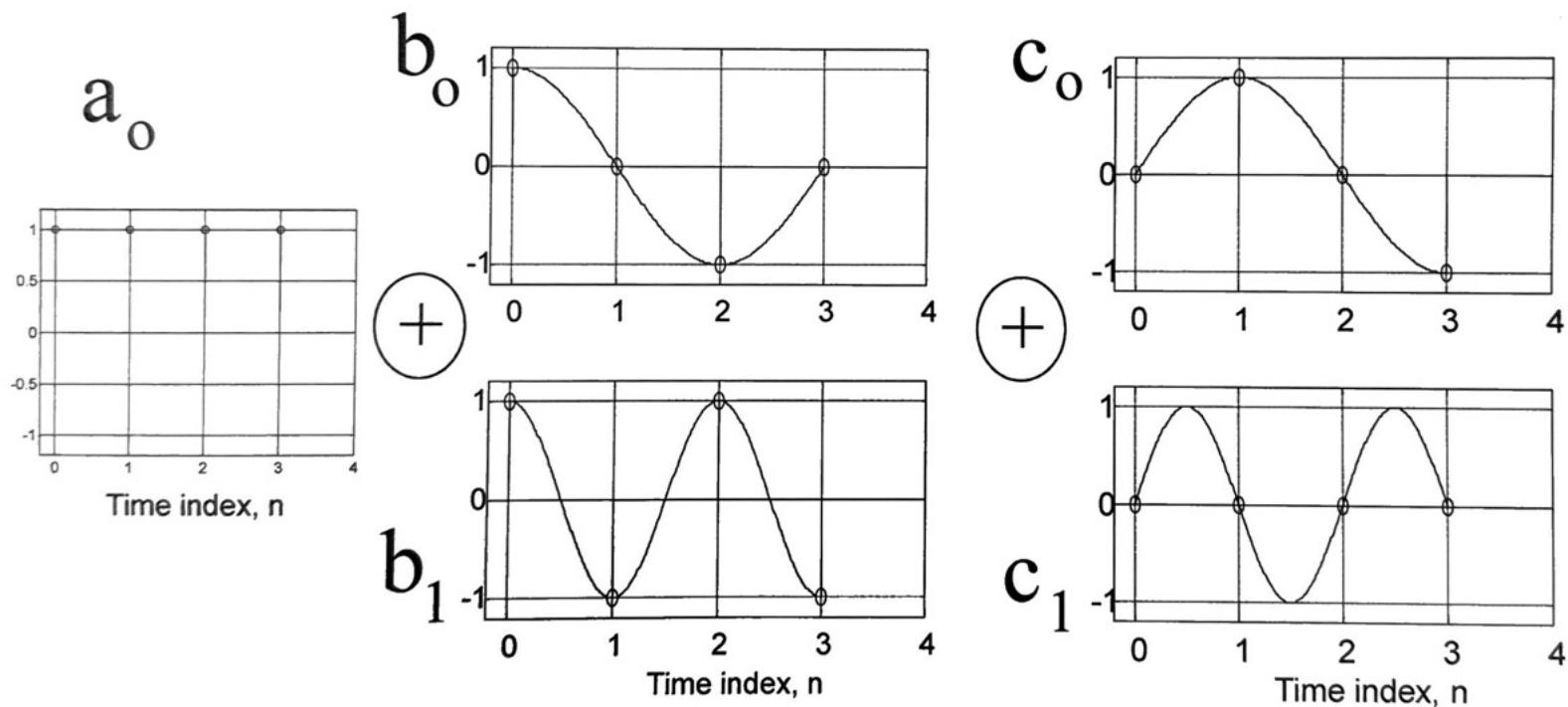
$$x[n] = a_0 + \sum_{k=1}^{N/2} \{ B_k \cos[k\omega_0 n] + C_k \sin[k\omega_0 n] \} \quad \omega_0 = 2\pi / N$$
$$-\infty < n < \infty$$

- The Fourier Series for the finite length discrete signal $x[n]$ is

$$x[n] = a_0 + \sum_{k=1}^{N/2} \{ B_k \cos[k\omega_0 n] + C_k \sin[k\omega_0 n] \} \quad \omega_0 = 2\pi / N$$
$$-N/2 \leq n \leq N/2$$

What does this all mean?

It means that, for example, any 4 data points $[d_0 \ d_1 \ d_2 \ d_3]$ can be represented by the sum of the following DT sequences (CT sines and cosines included as a guide for the eye):



Alternate Ways to Represent Data

So, the information in the four points

$$\begin{matrix} [d_0 \ d_1 \ d_2 \ d_3] \\ \text{are equally well} & \downarrow & \text{represented by:} \\ [a_0 \ b_0 \ c_0 \ b_1 \ c_1] \end{matrix}$$

These “spectral coefficients” are easy to find!

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} a_0 \\ b_0 \\ c_0 \\ b_1 \end{bmatrix}$$

Invert this equation to solve for spectral coefficients.

The Professional Way: FFT

Matlab offers a powerful tool for finding spectral coefficients, the FFT. Suppose that, in Matlab, we define:

$$D = [d_0 \ d_1 \ d_2 \ d_3]$$

Then `fft(D)` returns the answer:

$$P^* [a_0 \ (b_0 - jc_0)/2 \ b_1 \ (b_0 + jc_0)/2]$$

where P is the number of points in D (4 in this case).

More FFT Examples from Matlab

The FFT of a $P = 8$ point sequence would return:

$$P^* [a_0 \quad (b_0 - jc_0)/2 \quad (b_1 - jc_1)/2 \quad (b_2 - jc_2)/2 \quad b_3 \quad \dots \\ \quad (b_2 + jc_2)/2 \quad (b_1 + jc_1)/2 \quad (b_0 + jc_0)/2]$$

Some 4 point examples: $i = [0 \ 0.25 \ 0.5 \ 0.75]$

$\text{FFT}([1 \ 1 \ 1 \ 1])$ returns $[4 \ 0 \ 0 \ 0]$

$\text{FFT}(\sin(2\pi i))$ returns $[0 \ -2j \ 0 \ 2j]$

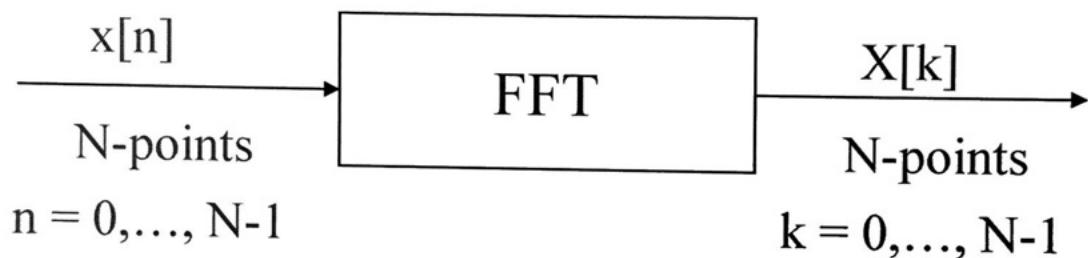
$\text{FFT}(\sin(4\pi i))$ returns $[0 \ 0 \ 0 \ 0]$

$\text{FFT}(\cos(2\pi i))$ returns $[0 \ 2 \ 0 \ 2]$

$\text{FFT}(\cos(4\pi i))$ returns $[0 \ 0 \ 4 \ 0]$

FFT

Fast Fourier Transform



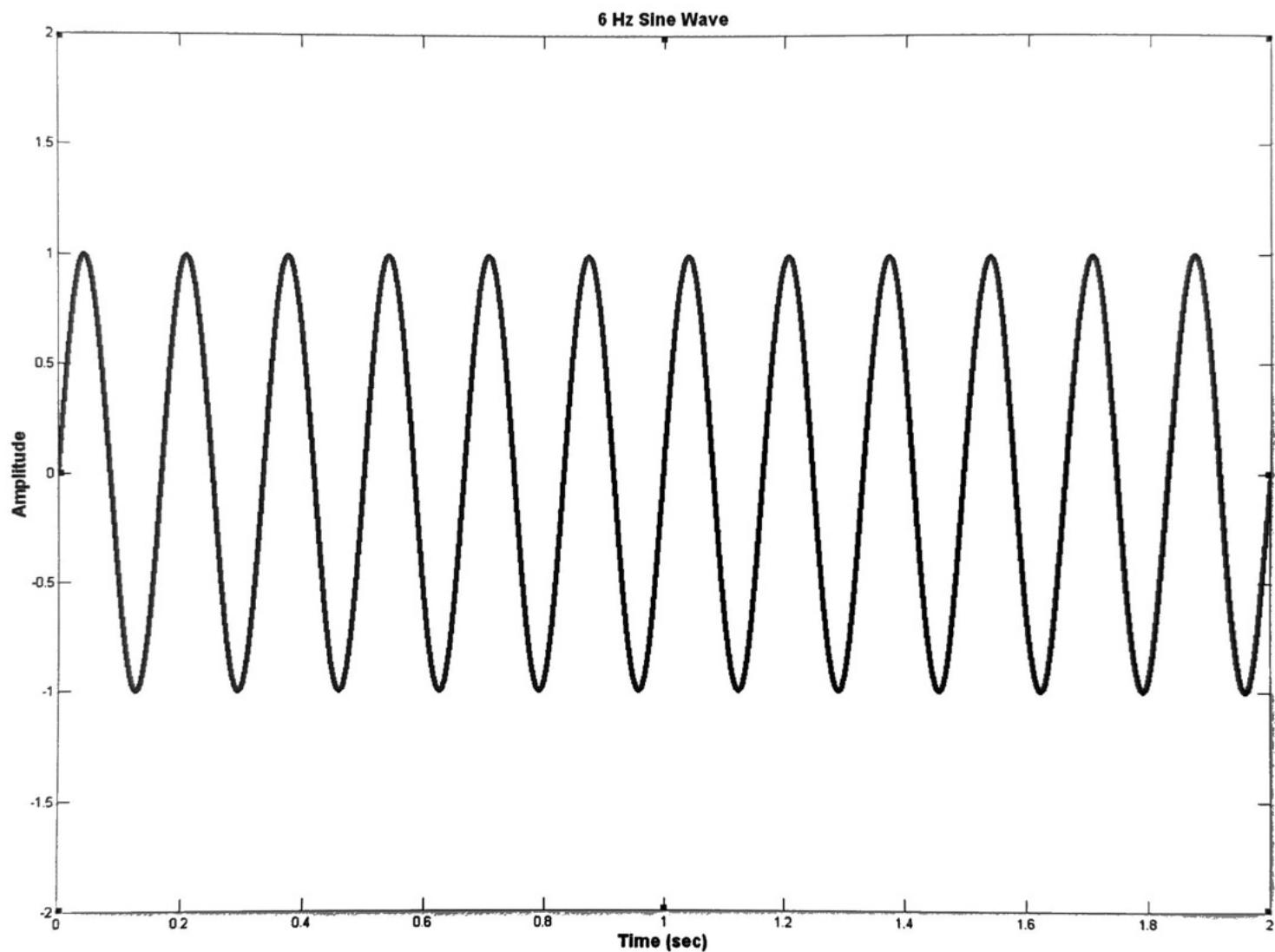
$$a_0 = X[0]/N$$

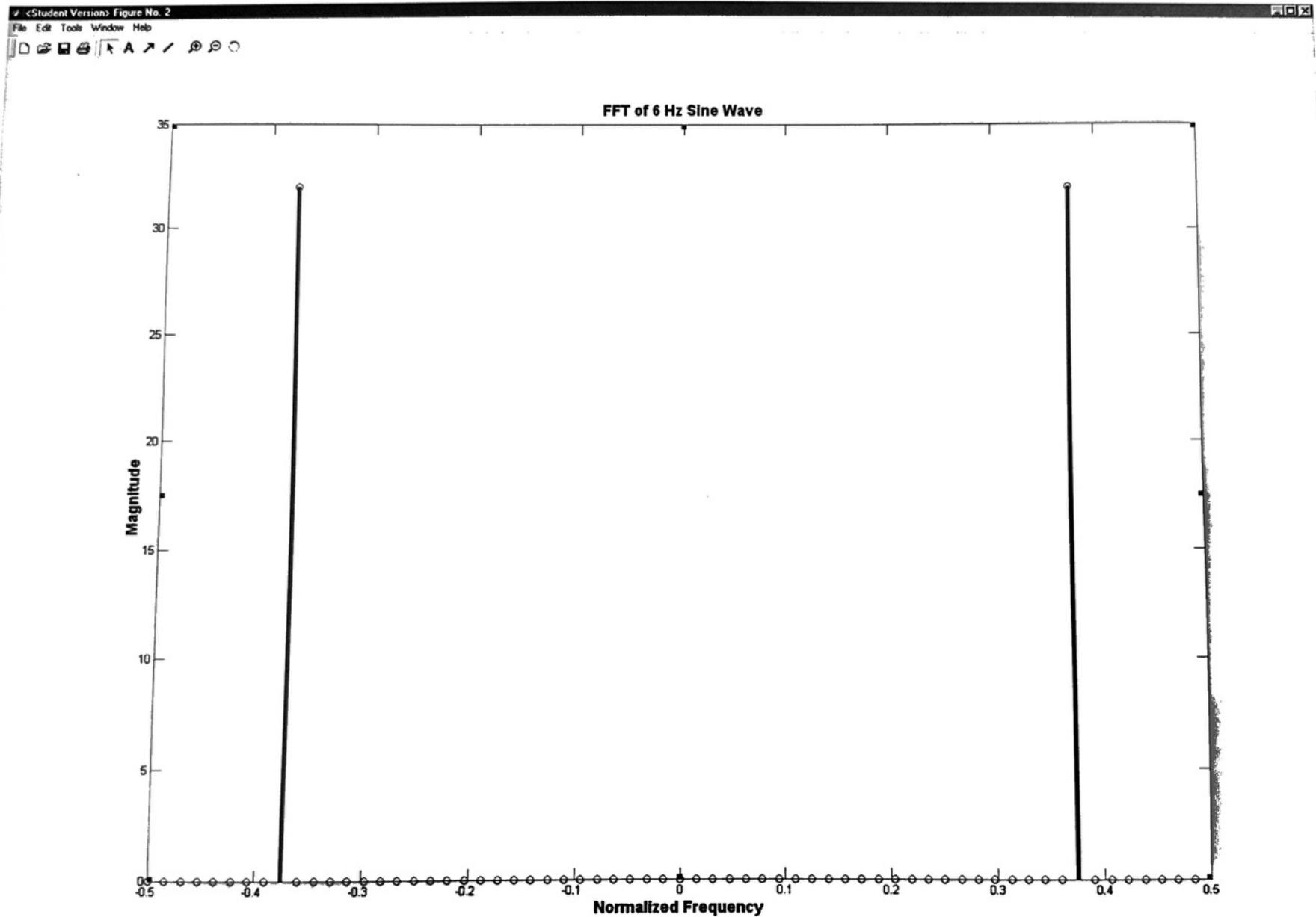
$$B_k = 2 * \text{Re}\{X[k]/N\} \quad C_k = 2 * \text{Im}\{X[k]/N\} \quad k = 1, \dots, N/2-1$$

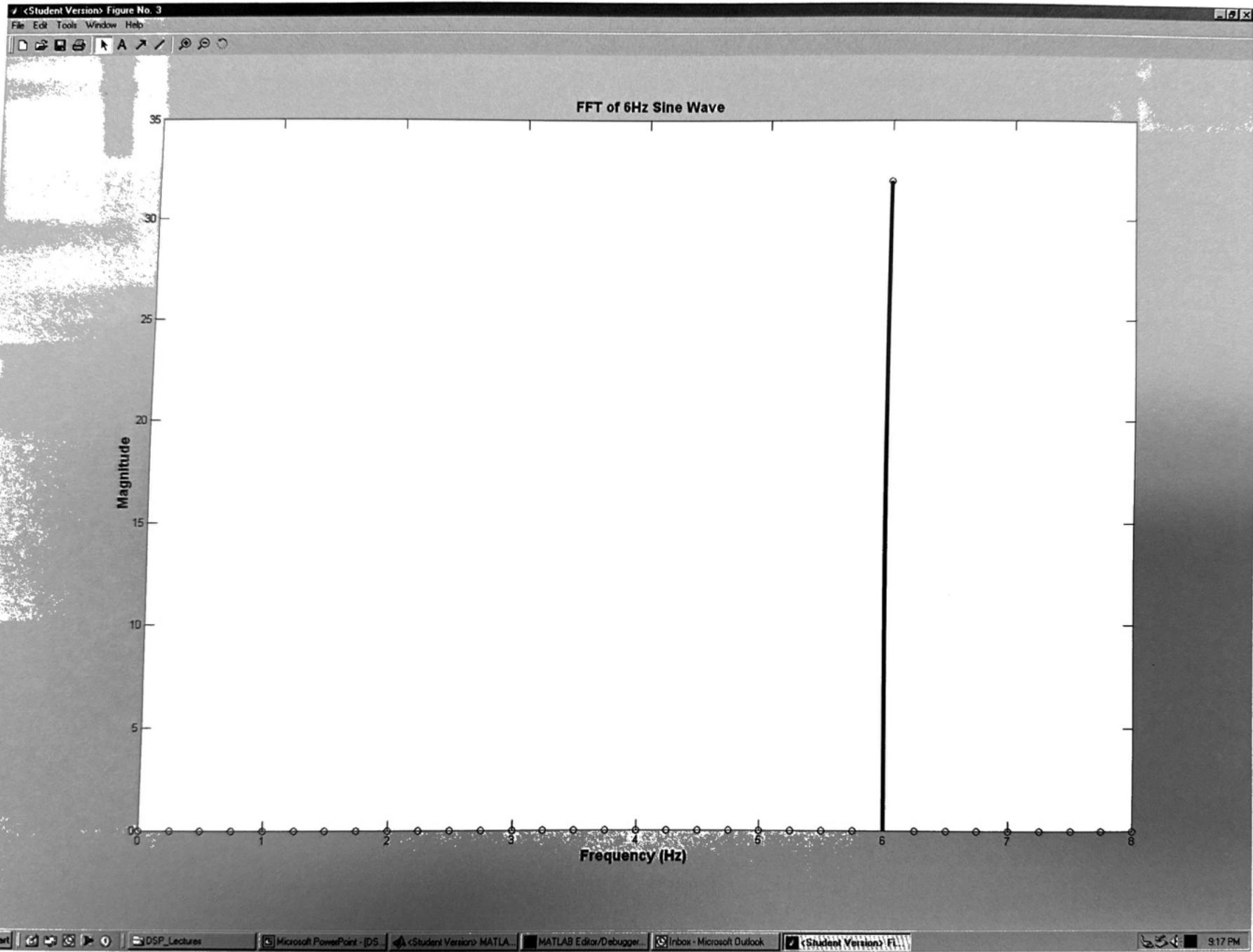
$$B_{N/2} = \text{Re}\{X[N/2]/N\} \quad C_{N/2} = \text{Im}\{X[N/2]/N\} = 0 \quad k = N/2$$

Sine Wave Examples

- FFT of 6 Hz Sine Wave will have:
 - 1 nonzero component at 6 Hz.

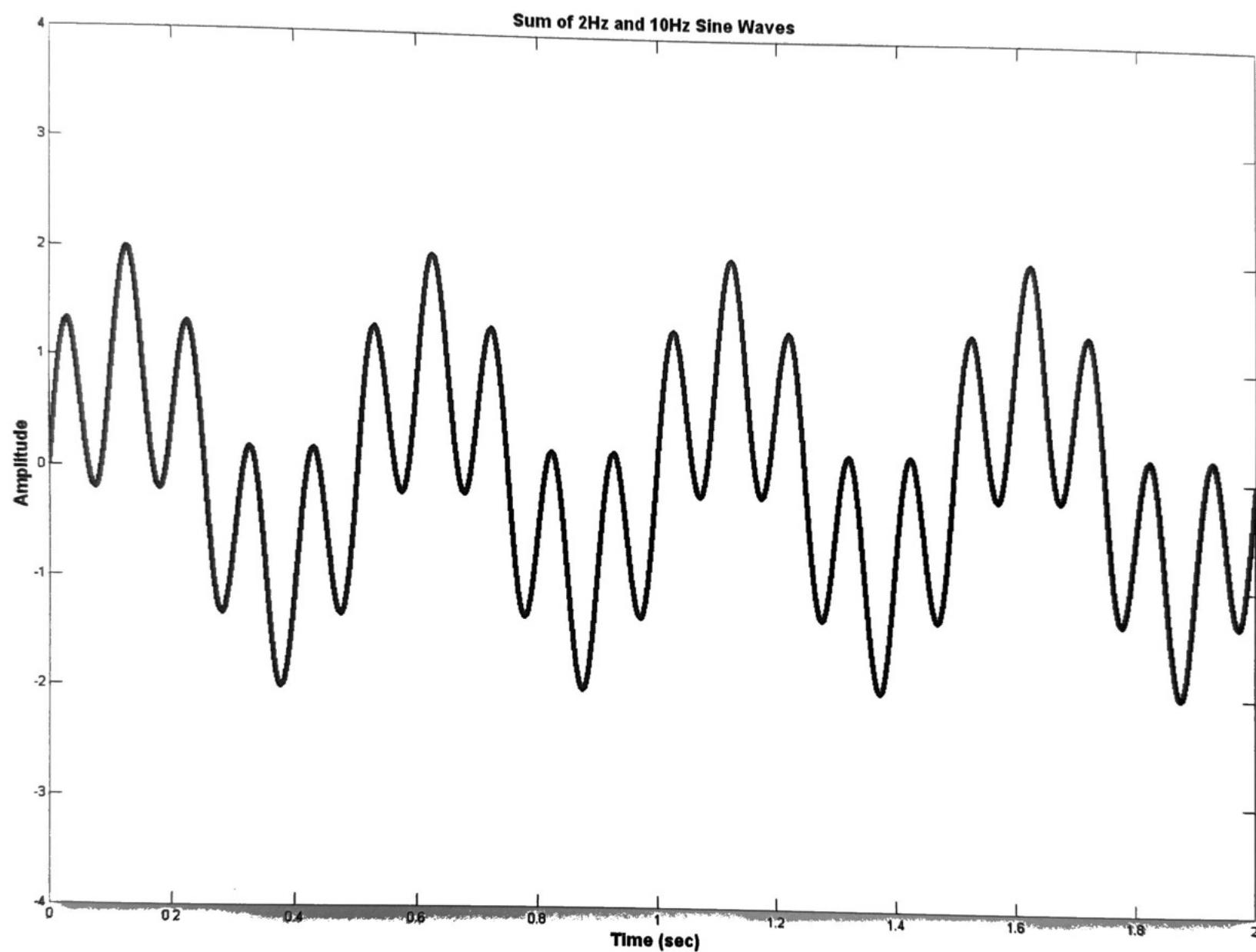
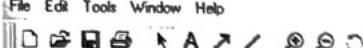


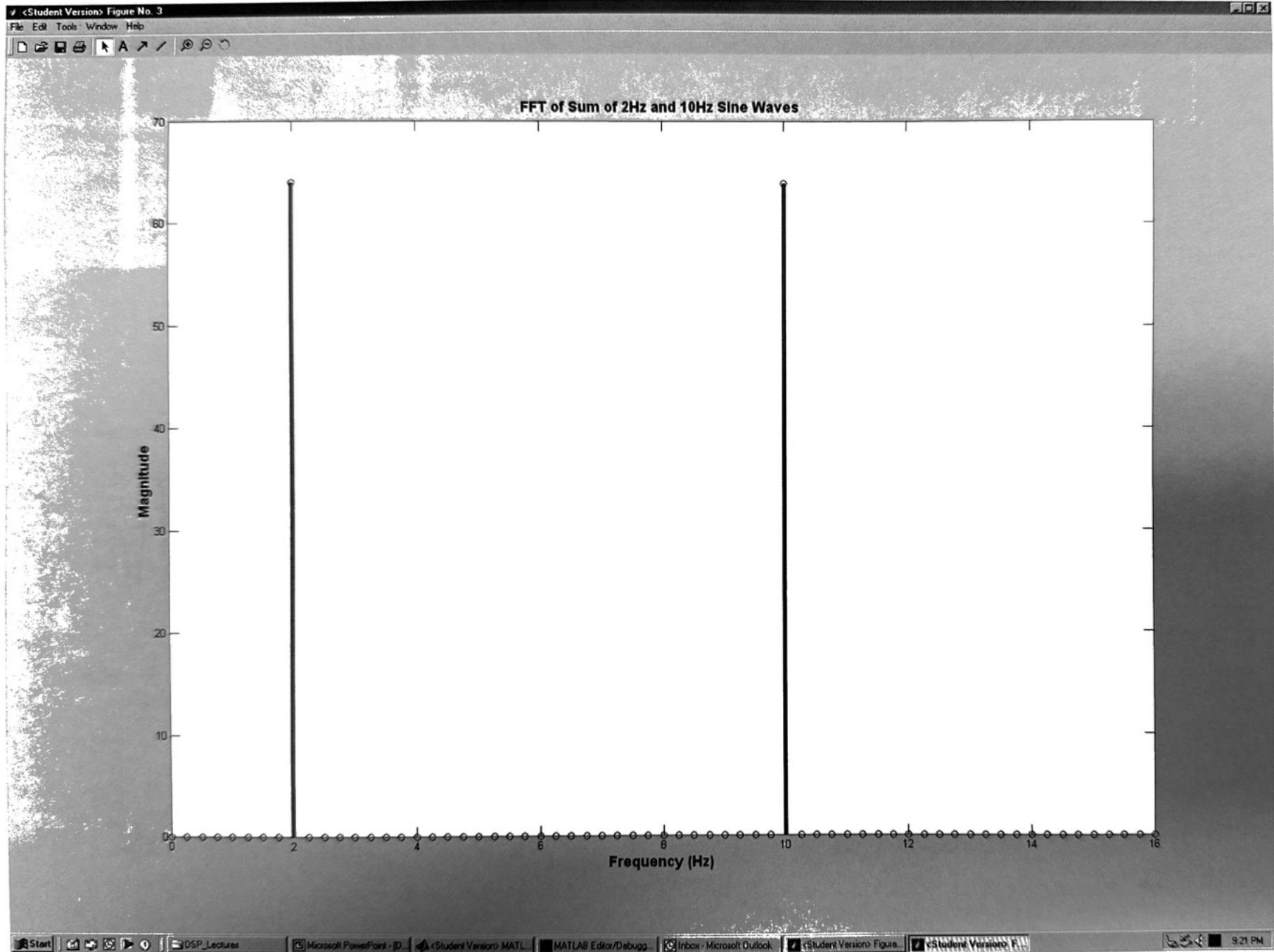




Sine Wave Examples

- FFT of sum of sine waves at 2 Hz and 10 Hz
 - 1 nonzero component at 2 Hz.
 - 1 nonzero component at 10 Hz





An Application Of Fourier Analysis

- The frequency composition of a signal given by Fourier Analysis is very useful for noise reduction.
- What are some examples of noise?
 - Additive Noise: Line Noise

$$\text{Observed Signal} = \text{Desired Signal} + \text{Noise Process}$$

- Convolutional Distortion: Communication Channel Noise

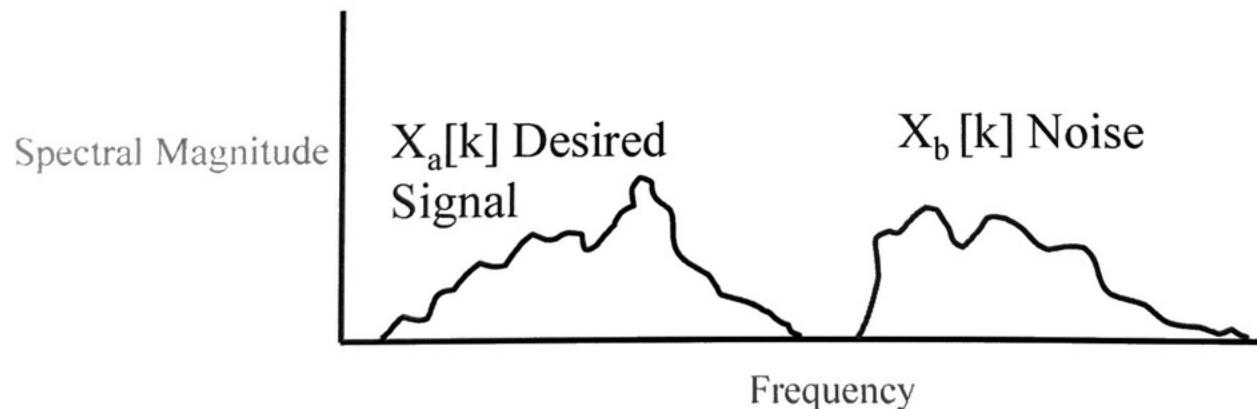
$$\text{Observed Signal} = \text{Desired Signal} * \text{Channel Impulse Response}$$

How Fourier Analysis Helps With Additive Noise

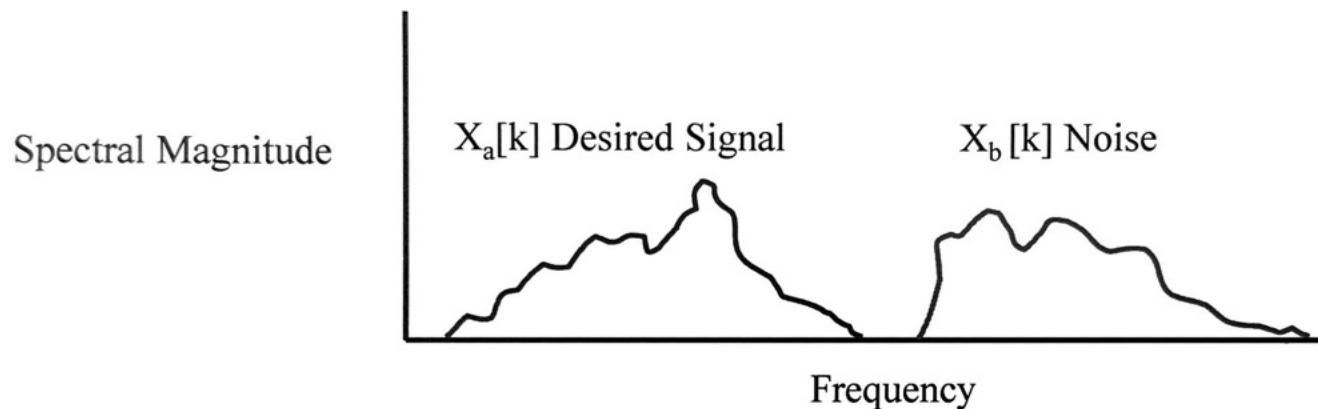
- DFT of the sum of two signals equals the sum of the individual DFTs:

$$x_a[n] + x_b[n] \rightarrow X_a[k] + X_b[k]$$

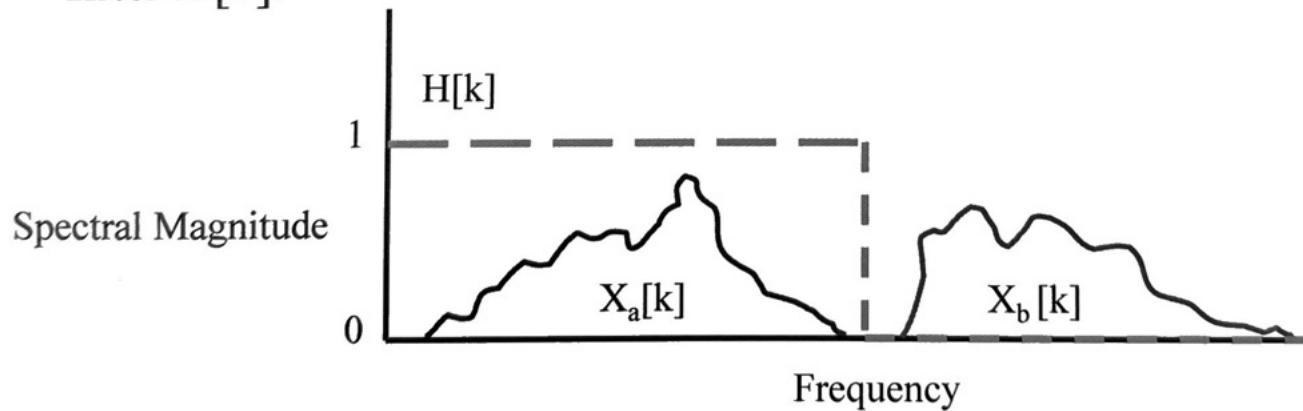
- We get lucky if $x_a[n]$ (desired signal) and $x_b[n]$ (noise) are composed of different frequencies!



How Fourier Analysis Helps



- To recover $X_a[k]$ from $X_b[k]$, we multiply the above spectrum with a filter $H[k]$:

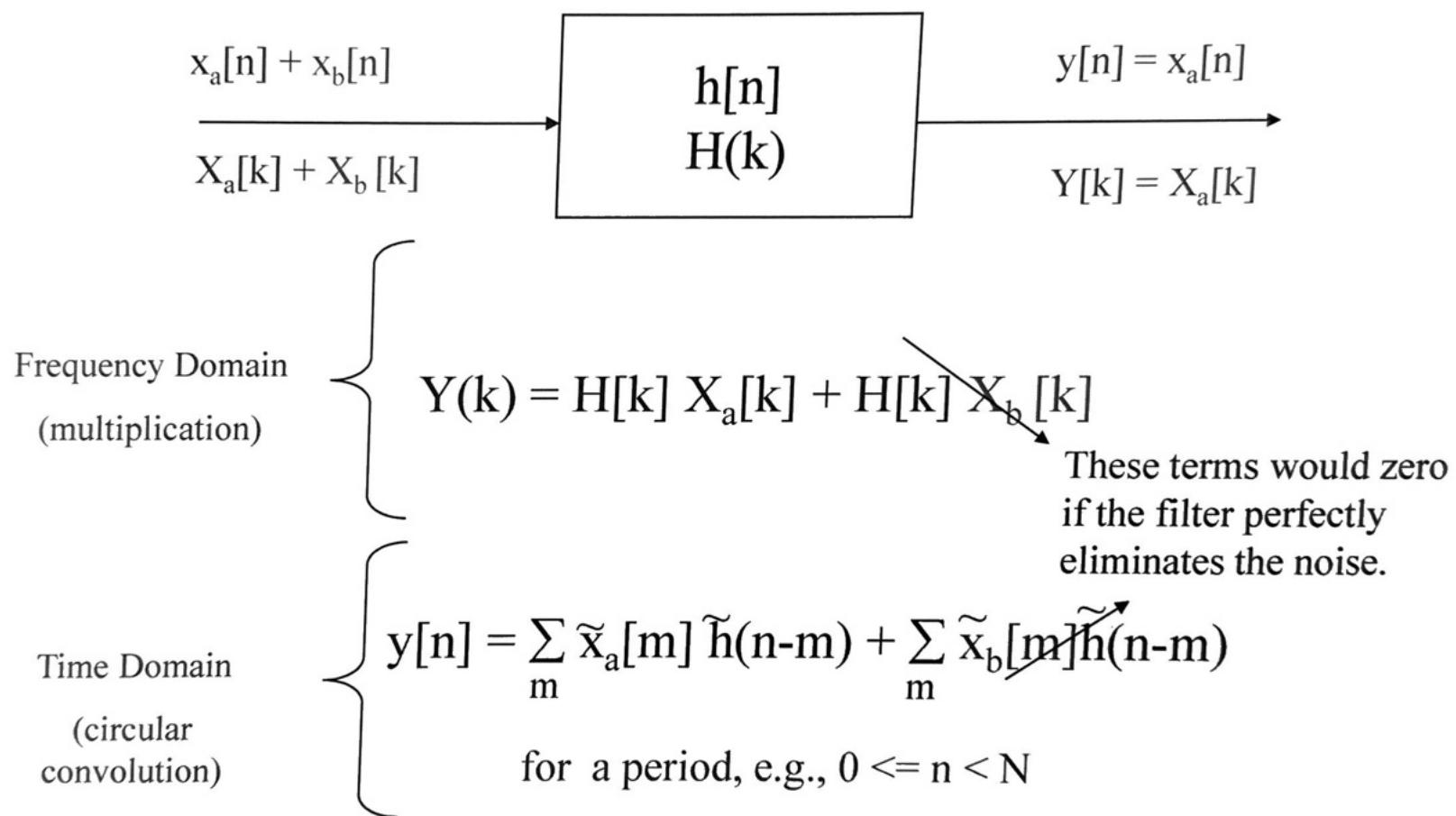


Multiplication in Frequency

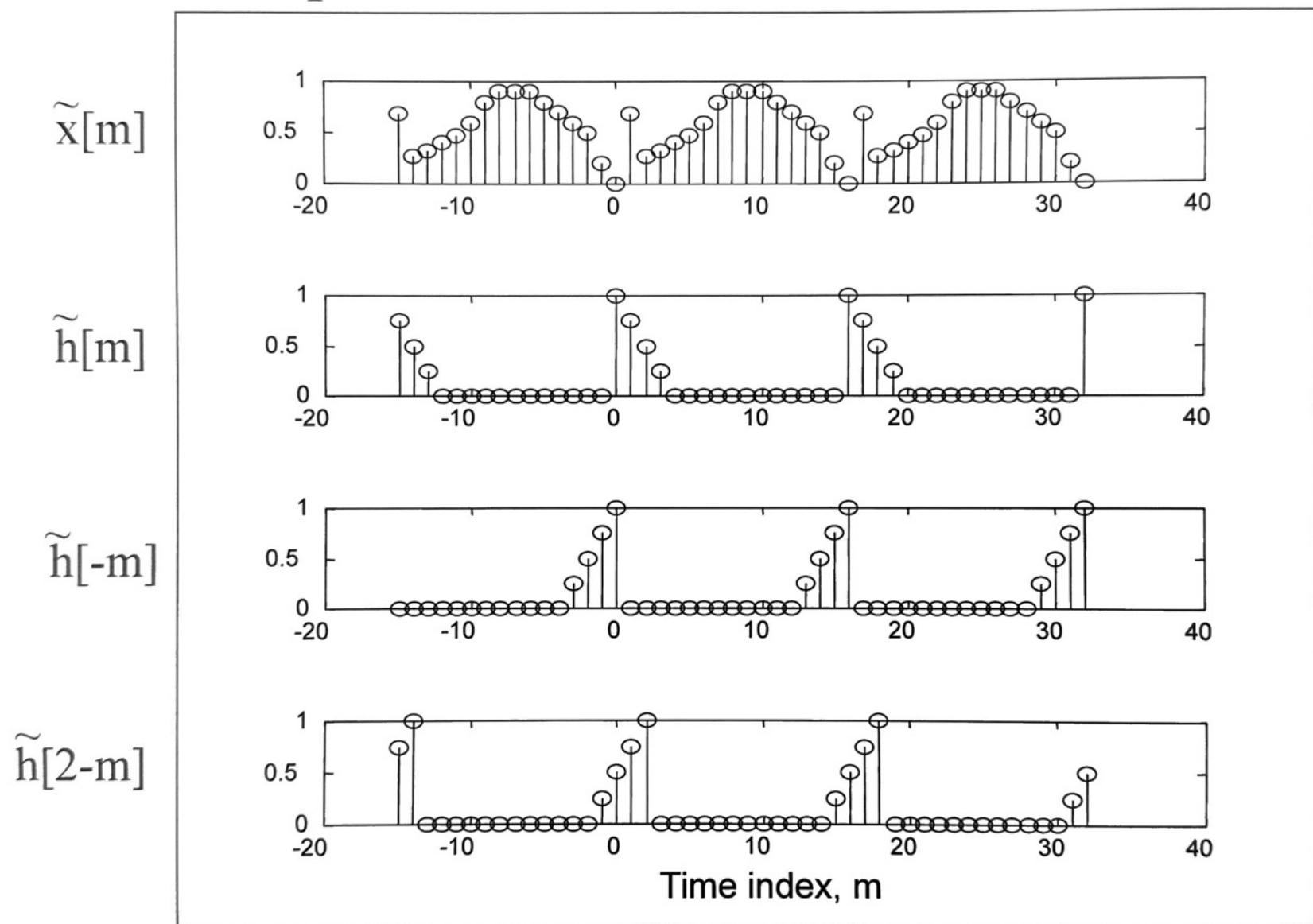
- We just saw that multiplication in the frequency domain allowed us to recover $X_a[k]$ from $X_b[k]$.
- Multiplication in Frequency = Convolution in Time
- Recall that our windows of data $x[n]$ came from an imagined periodic or repeated version $\tilde{x}[n]$. So, the convolution operation implied by multiplying DFT's is a “circular convolution”:

$$X[k] H[k] \rightarrow \sum_{m=0}^{N-1} \tilde{x}[m] \tilde{h}[n-m] \text{ (for } 0 \leq n < N\text{)}$$

Additive Noise Reduction Block Diagram

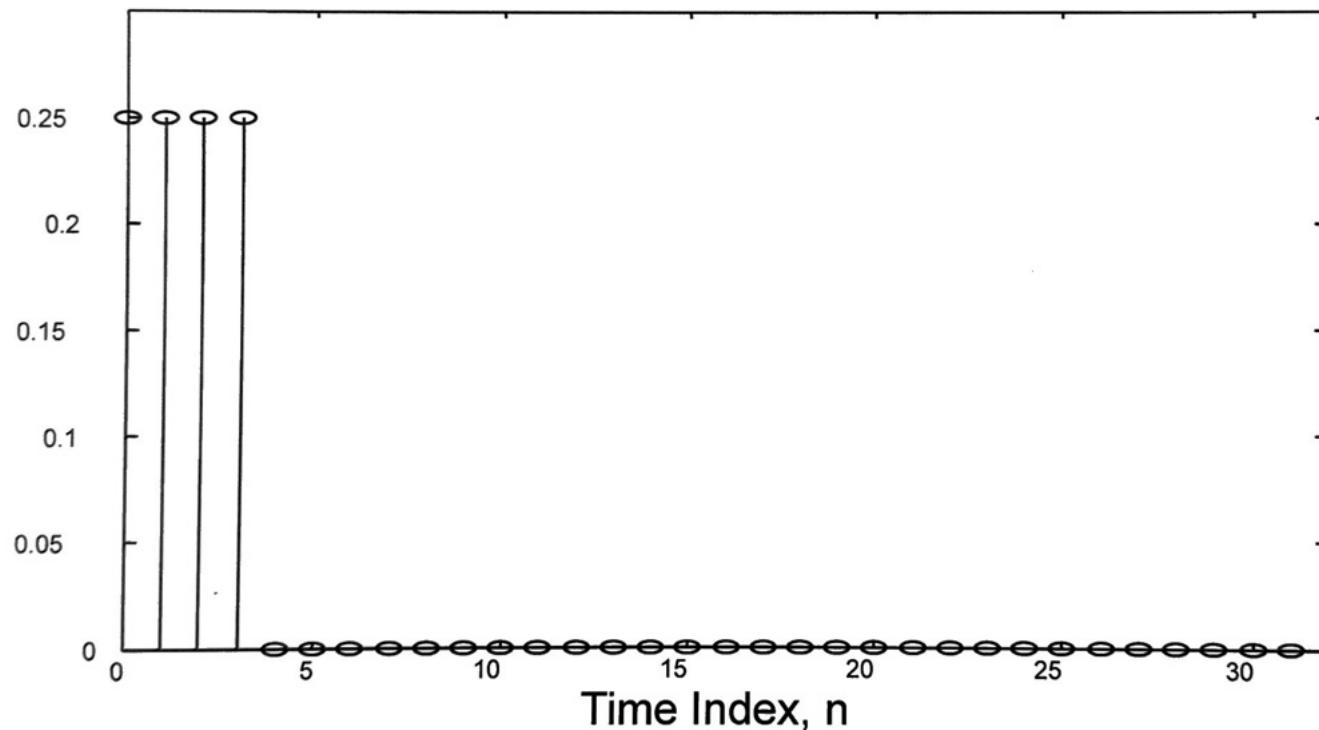


Components for Circular Convolution

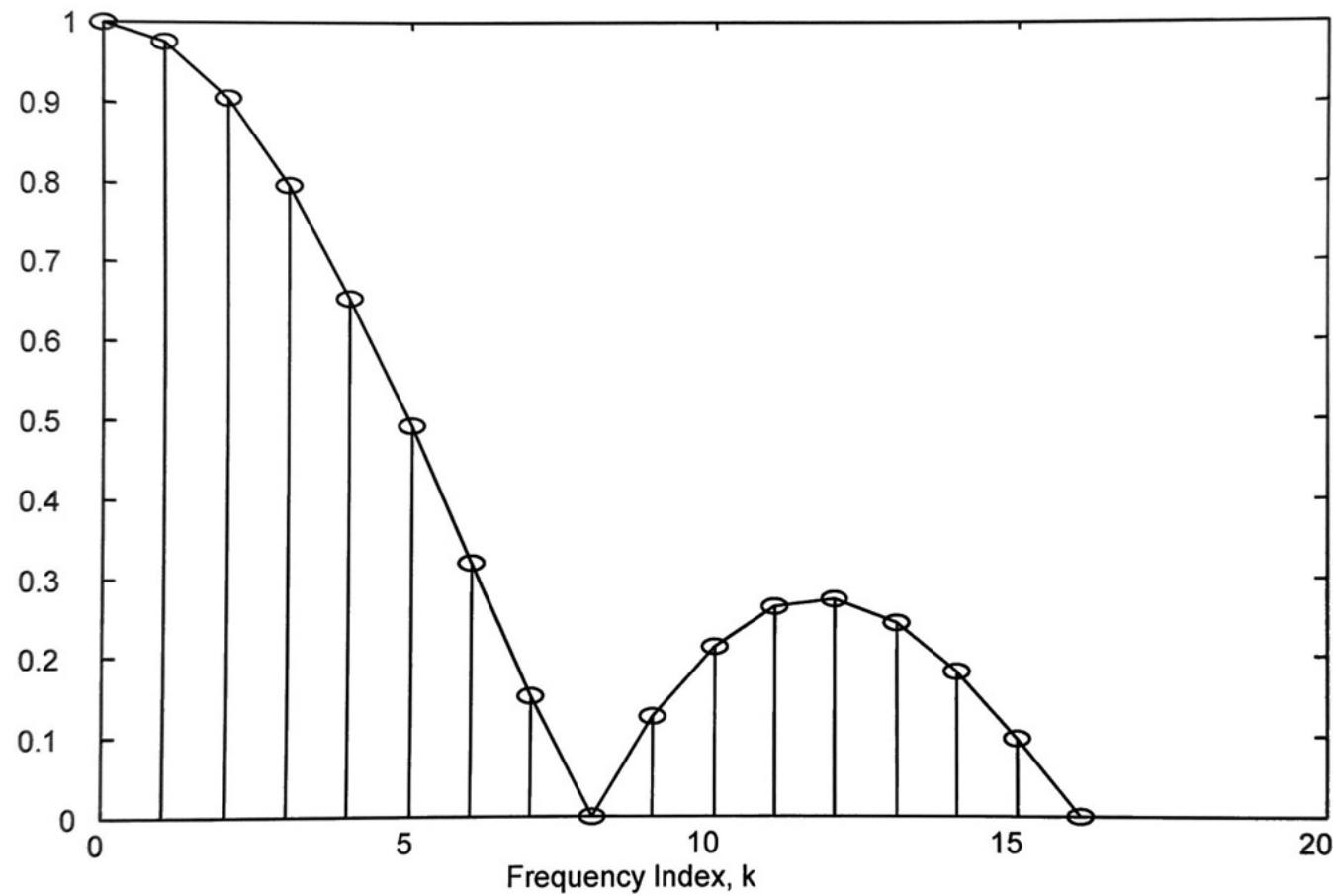


Easy Example: Low-Pass Box

Suppose we convolve a signal in time with this:



In frequency, that's like multiplying by this:



Box and Sinc are Duals!

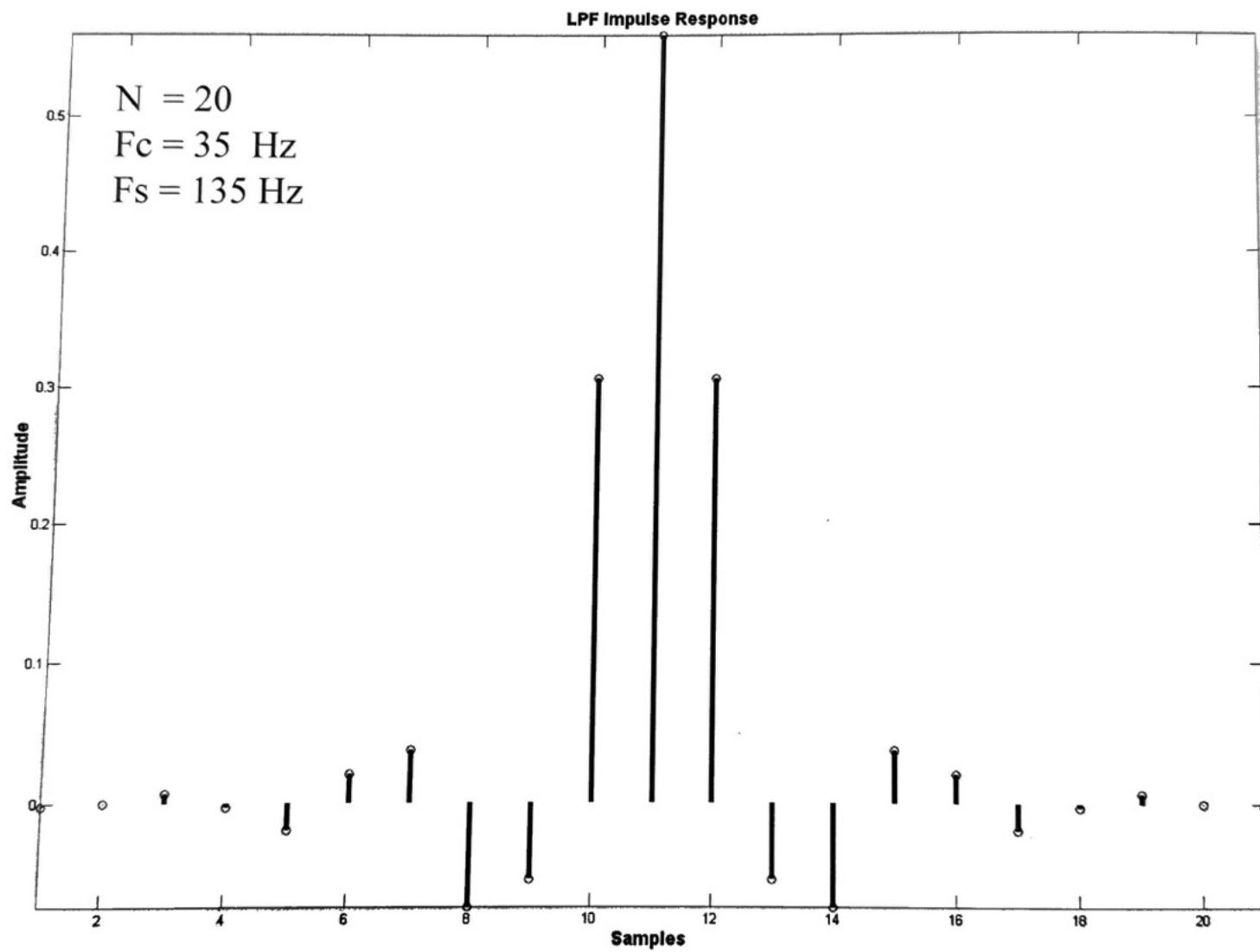
- Convoluting with a box gives a low-pass filter effect, but not a very good low pass filter effect!
- Convoluting with a Sinc function in time gives a perfect low-pass box in frequency, but a filter with a global impulse response.
- Usually, we pick a compromise – neither compactly supported in frequency or in time!

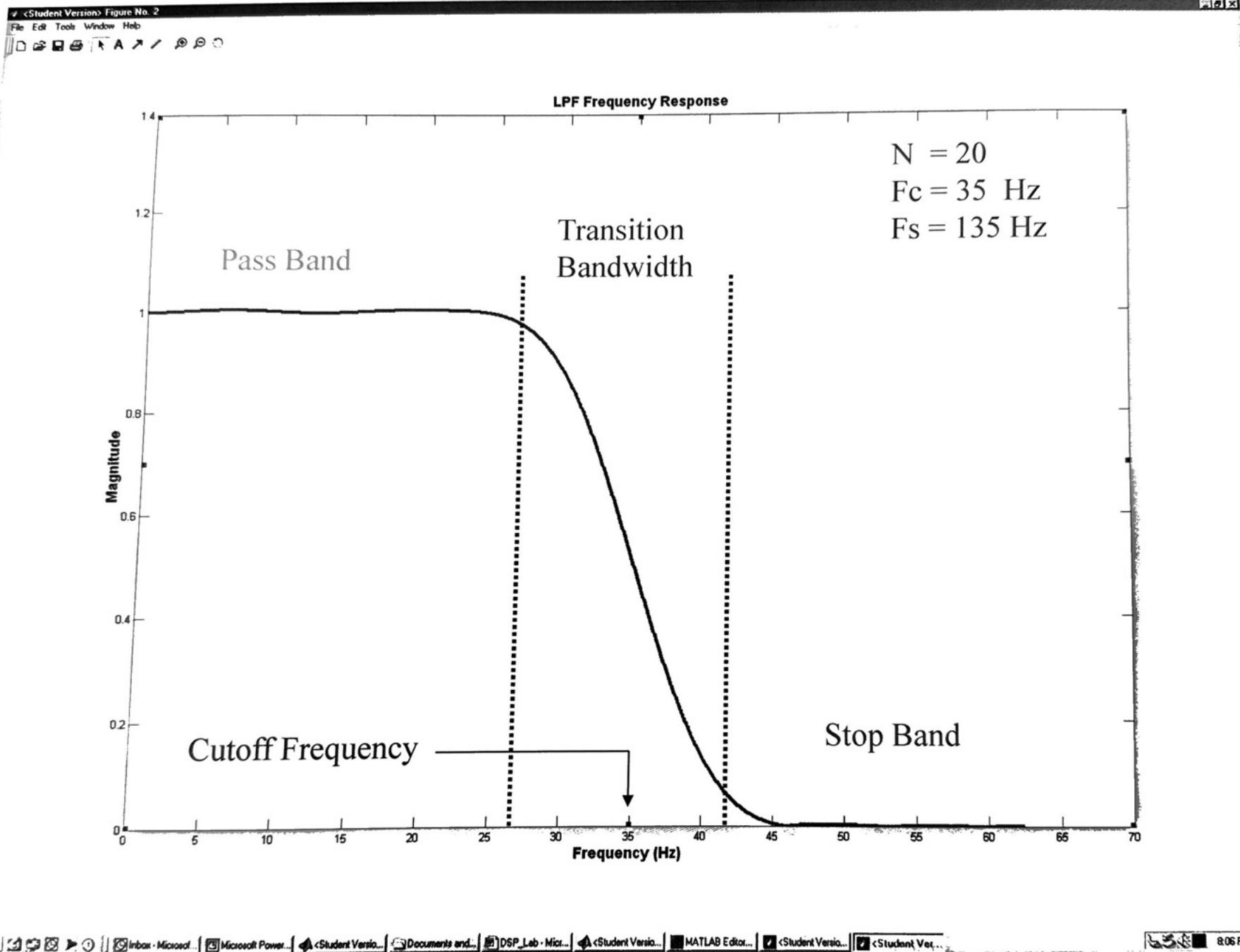
FIR Filters

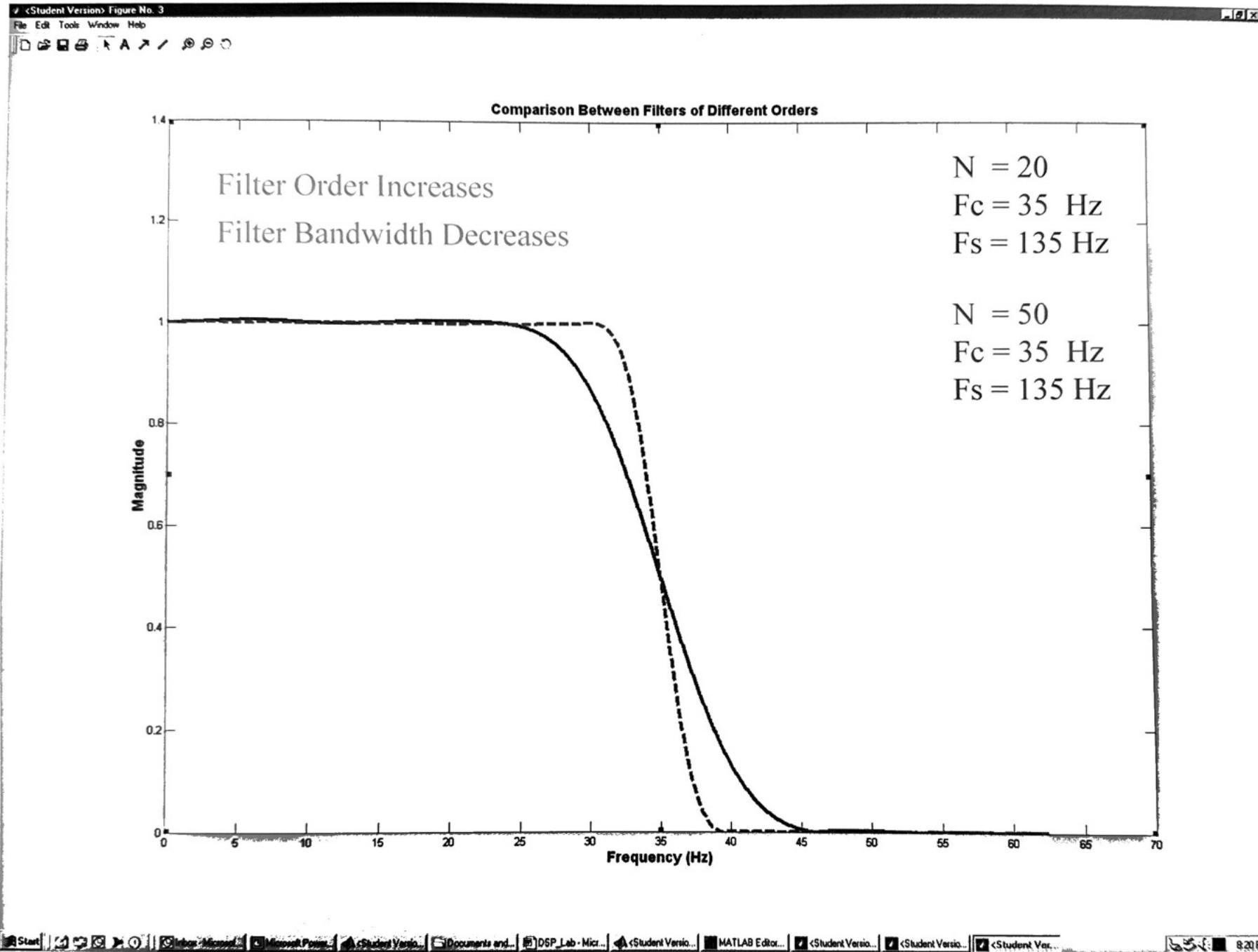
- Finite Impulse Response Filters are LTI systems that selectively modify the frequency spectrum of an input signal.
- Design an FIR Low Pass Filter in Matlab using fir1

$$h = \text{fir1}(N, 2*(F_c/F_s));$$

↑ ↑ ↑
Impulse Response Filter Order Cutoff Frequency



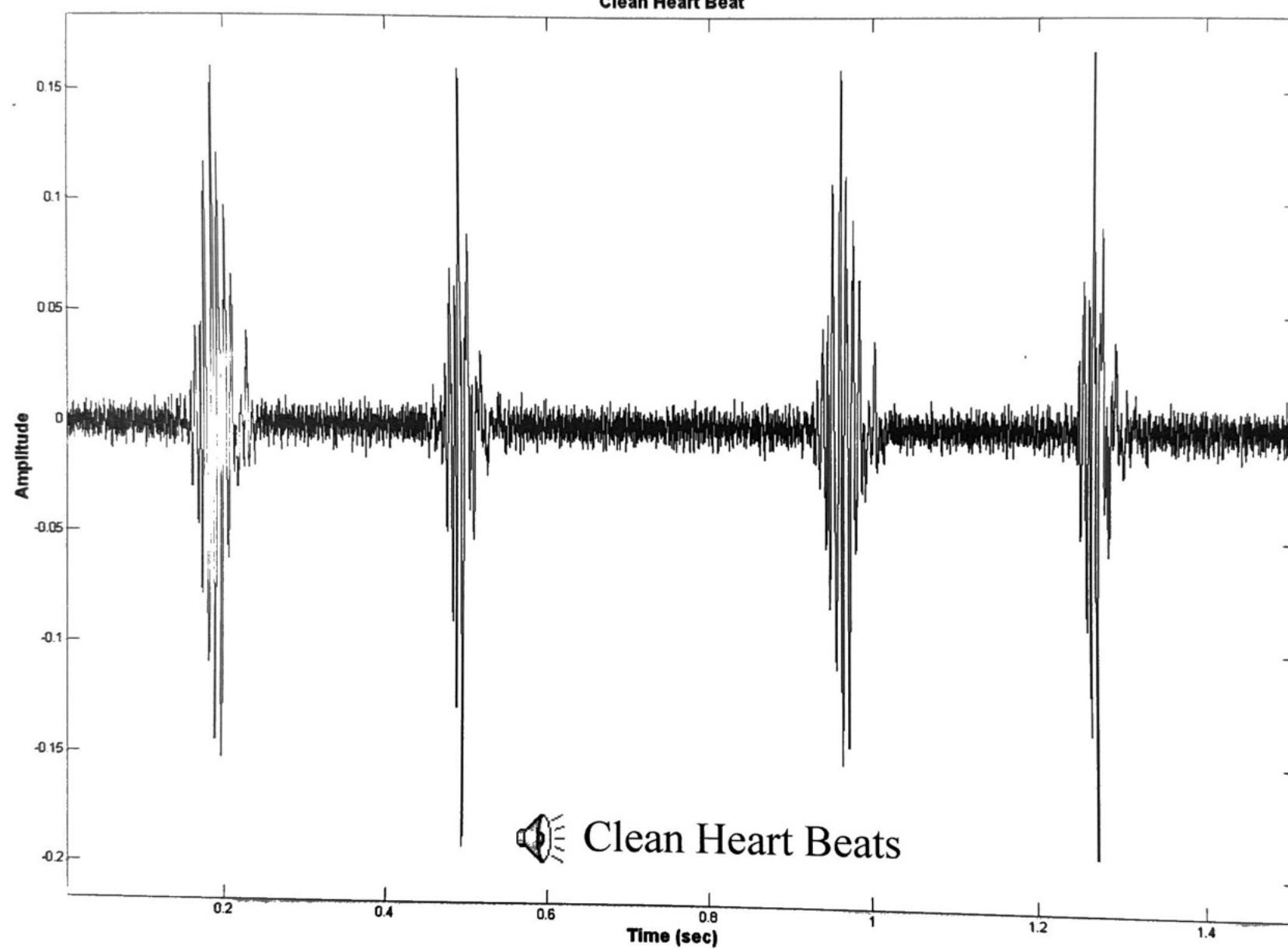


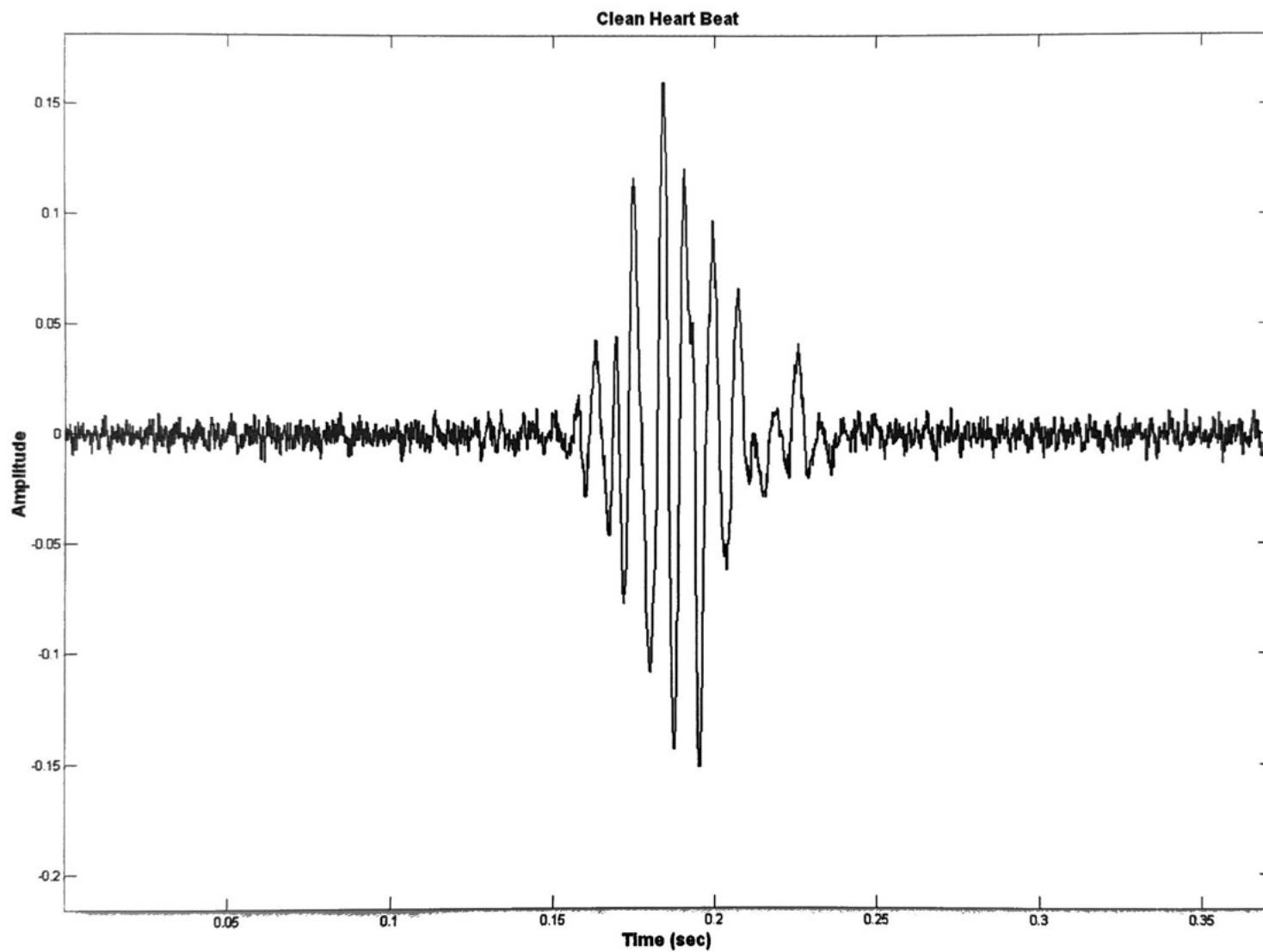


An Idealized Example

- Now we will try to remove noise composed of a single 500 Hz tone from a heart beat.

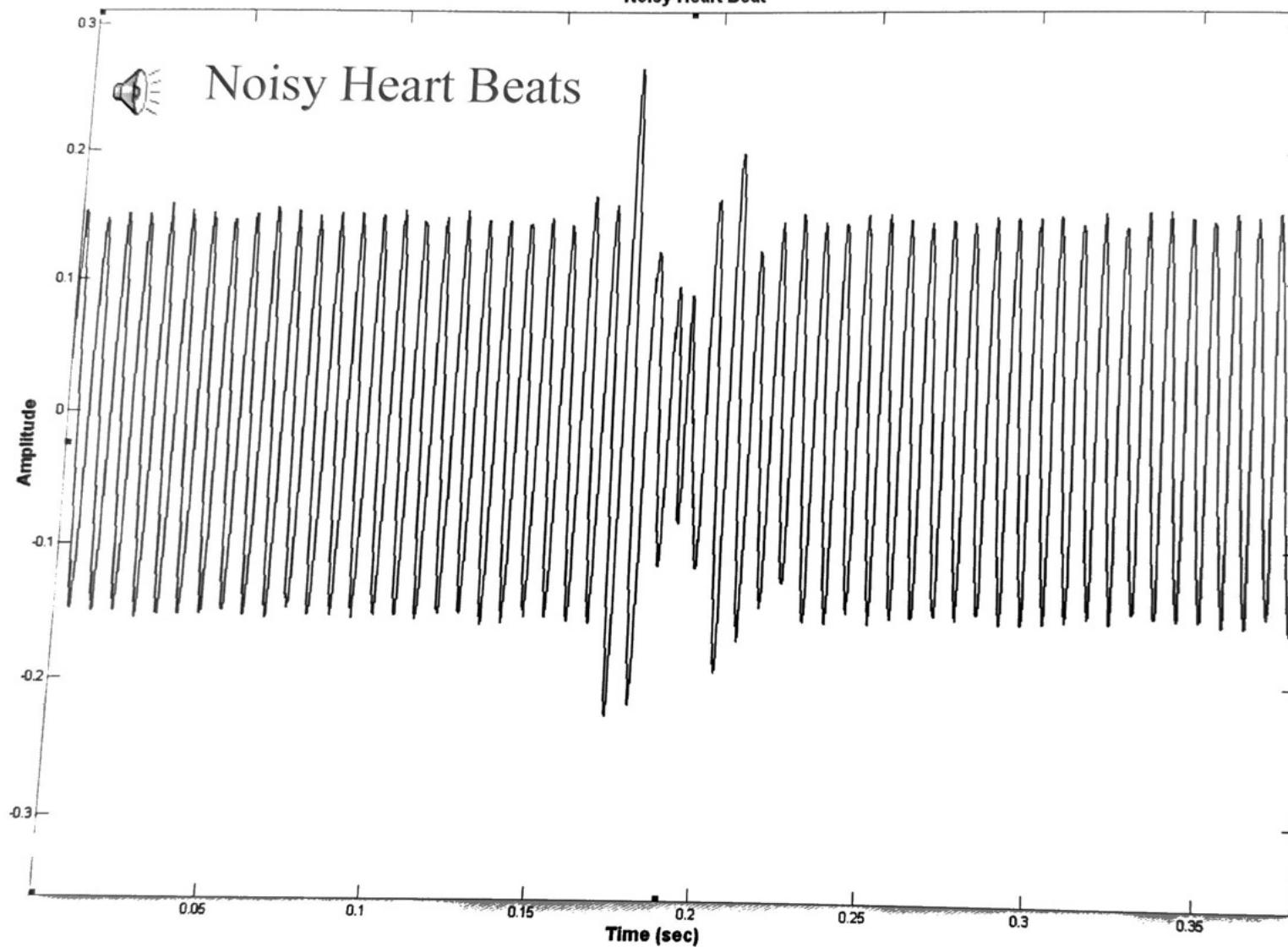
Clean Heart Beat





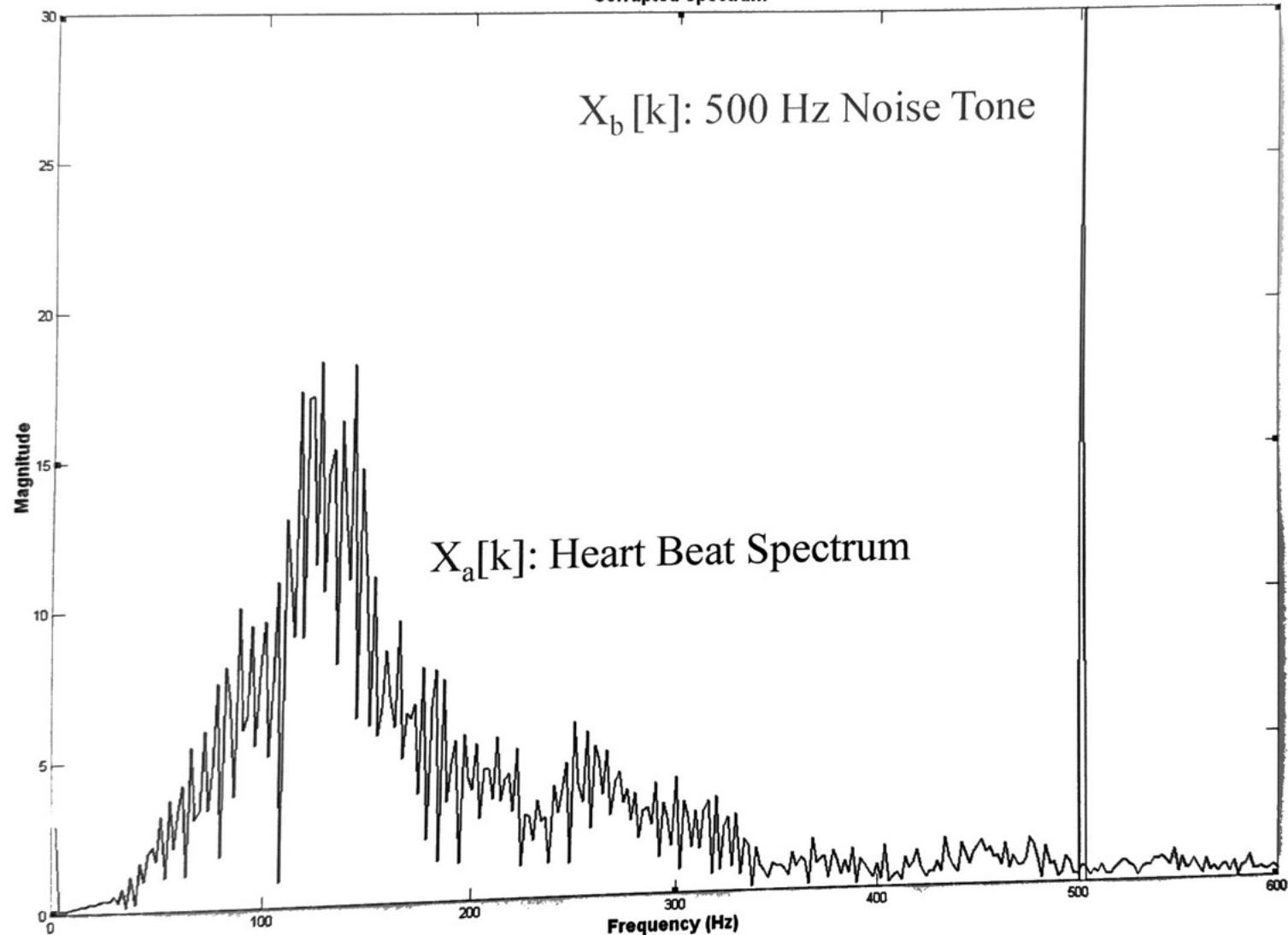


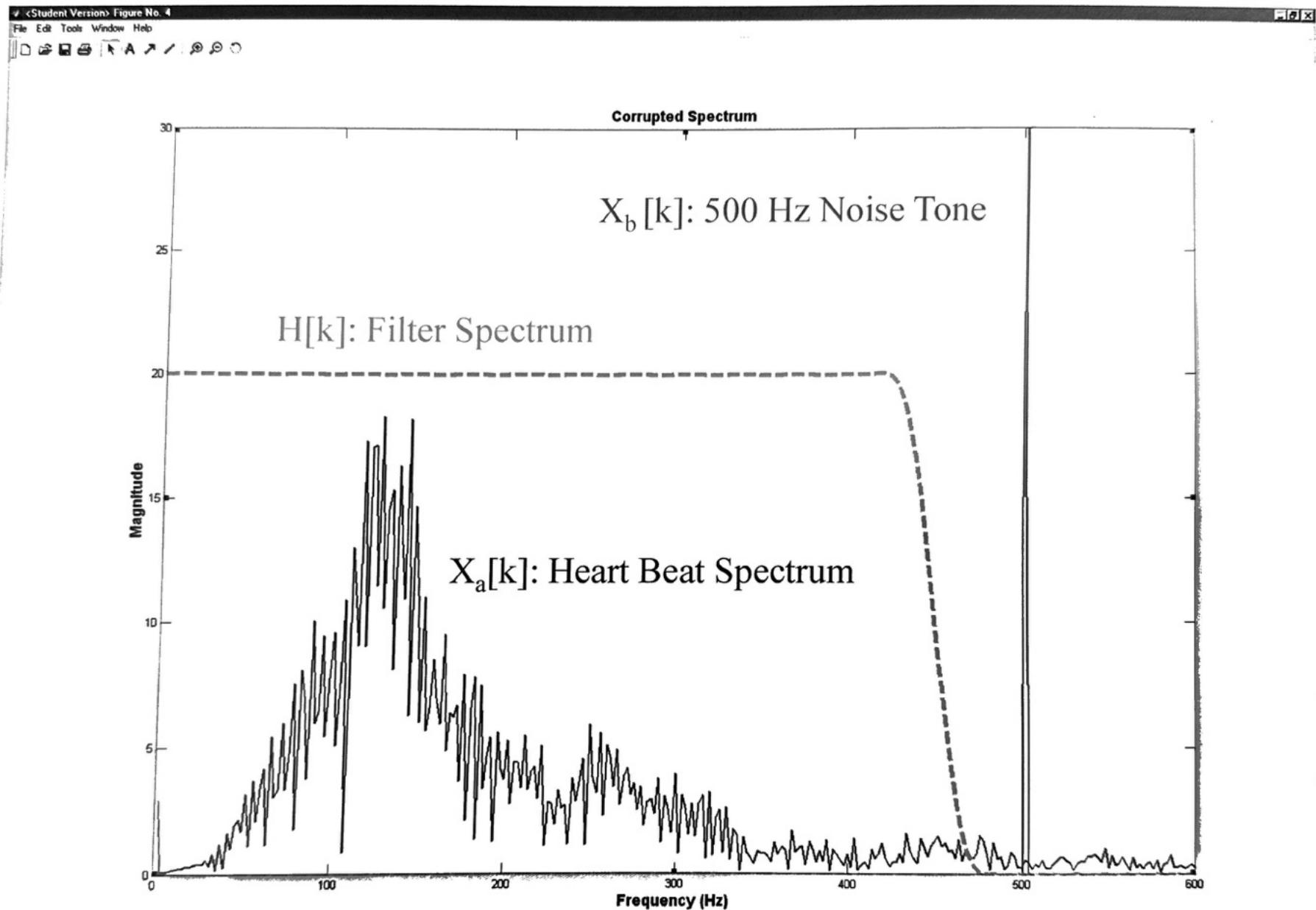
Noisy Heart Beat

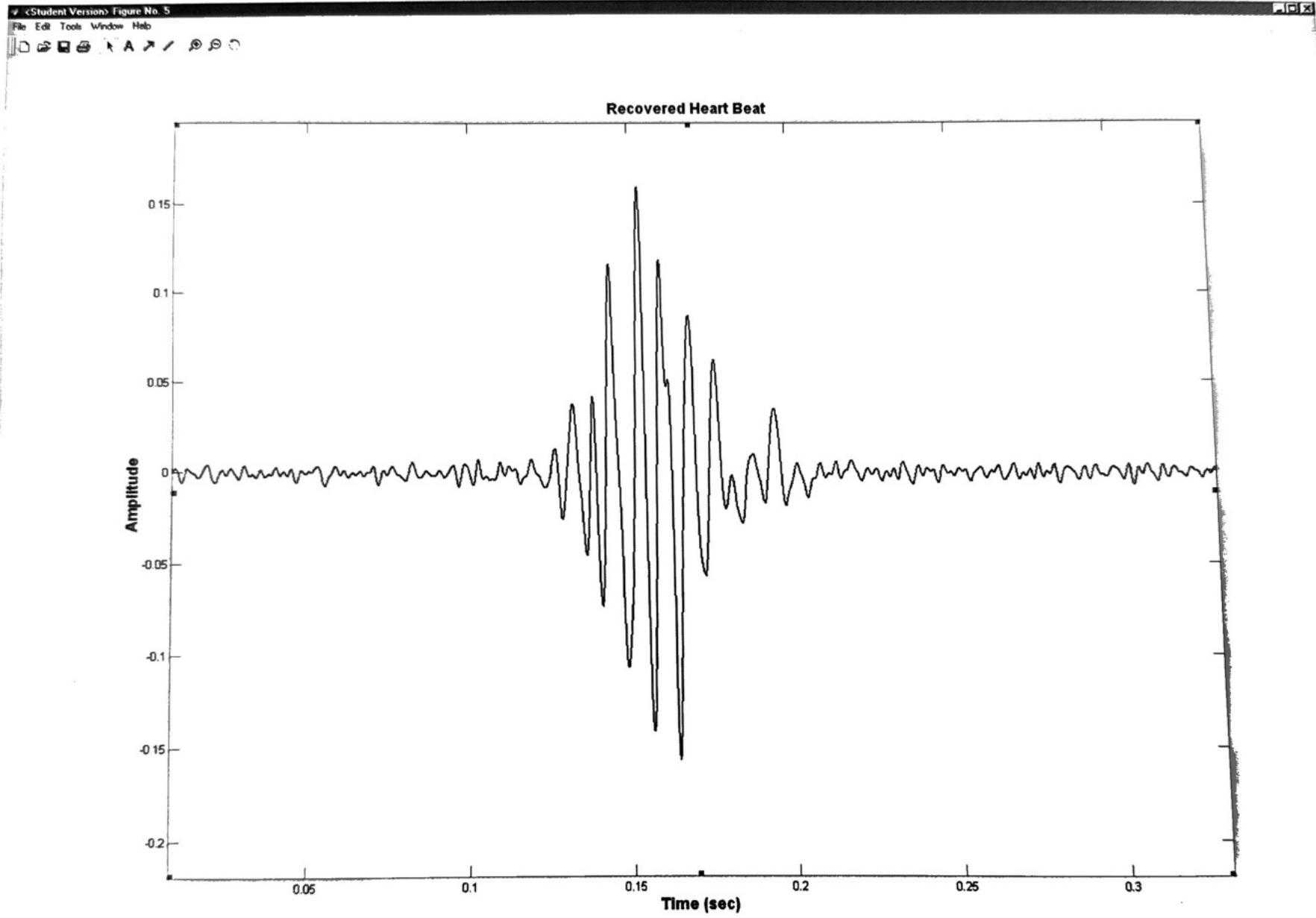


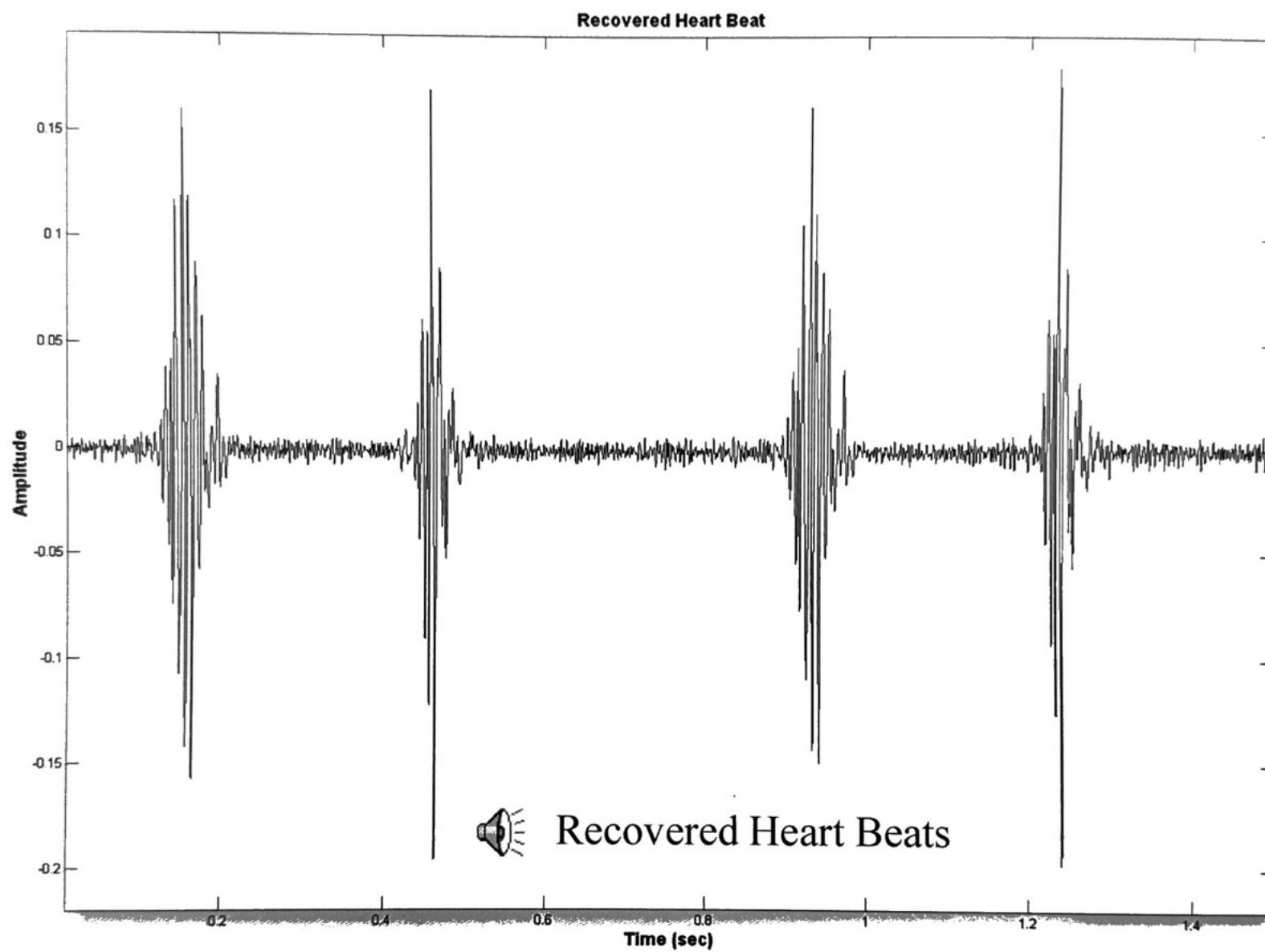


Corrupted Spectrum

 $X_b[k]$: 500 Hz Noise Tone $X_a[k]$: Heart Beat Spectrum

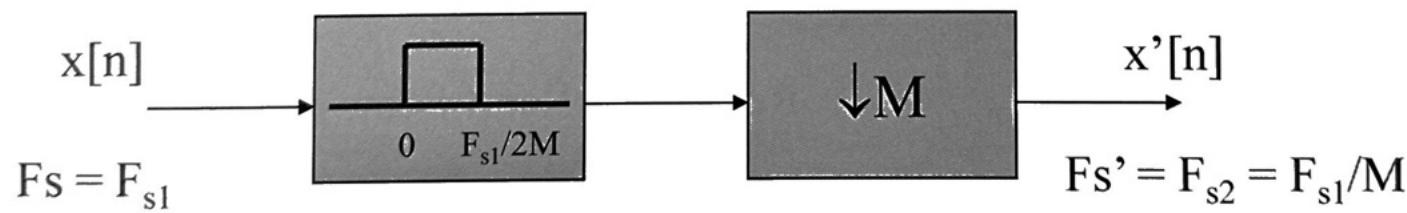






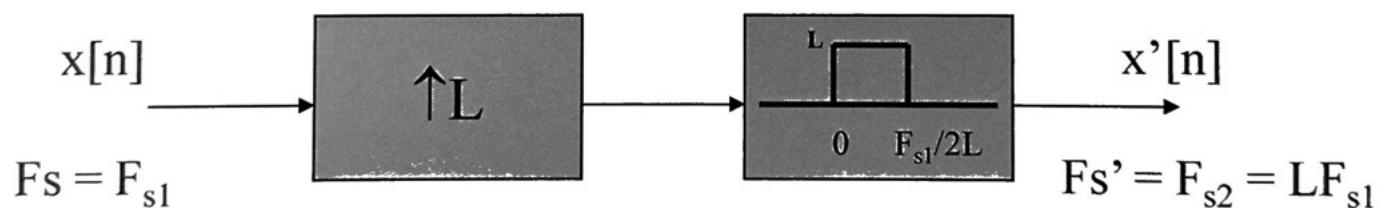
Downsampling by a factor of M

- If a signal has been sampled at a rate F_{s1} , it can be downsampled by a factor of M to a new sampling frequency F_{s2} using the following procedure
 - Band-limit the signal to half the new sampling rate: $F_{s2}/2$
 - Throw out $M-1$ samples for every sample.



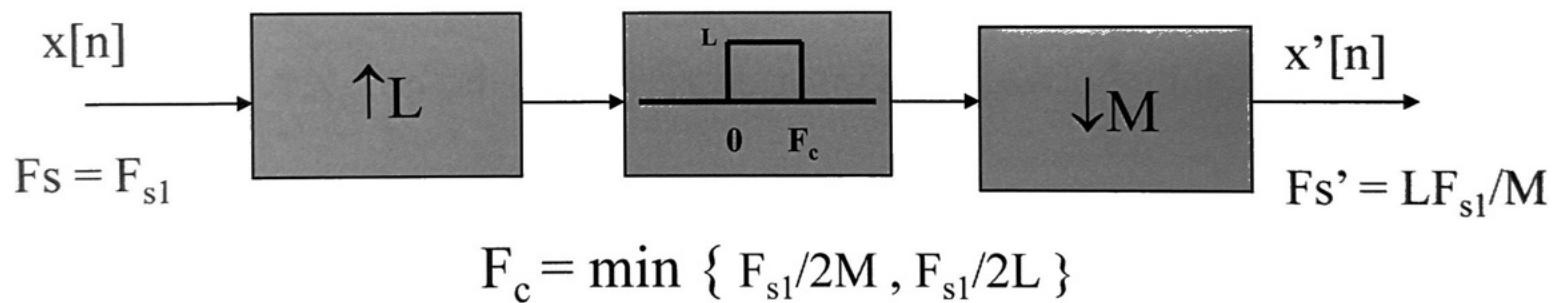
Upsampling by a factor of L

- If a signal has been sampled at a rate F_{s1} , it can be upsampled by a factor of L to a new sampling frequency F_{s2} using the following procedure
 - Insert $L-1$ zeros after every sample
 - Interpolate between the inserted zeros and the non-zero samples.



Changing the Sampling Frequency by Factor of L/M

- Upsample FIRST by a factor of L.
- Downsample SECOND by a factor of M.



Background

Class of discrete, nonlinear filters which incorporate a sorting element, e.g.

The Median Filter

Capable of suppressing impulse noise

Preserves underlying edges

Smooths large outliers even when signal and noise spectra overlap

Used extensively in image signal processing

Further Considerations

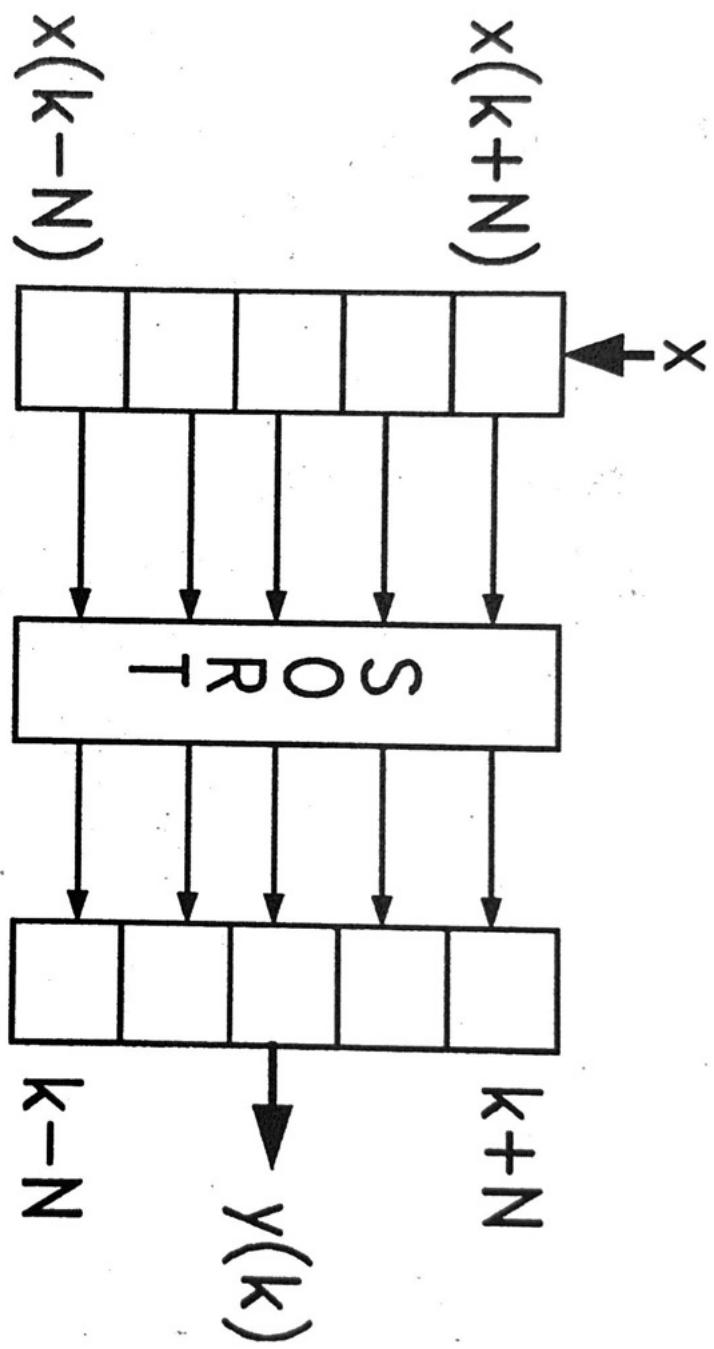
Median and Median-type filters are especially valuable when:

The probability distribution of the noise is unknown or heavily-tailed (e.g. Cauchy noise).

Signal and noise spectra overlap.

The underlying waveform has sharp edges.

The Median Filter

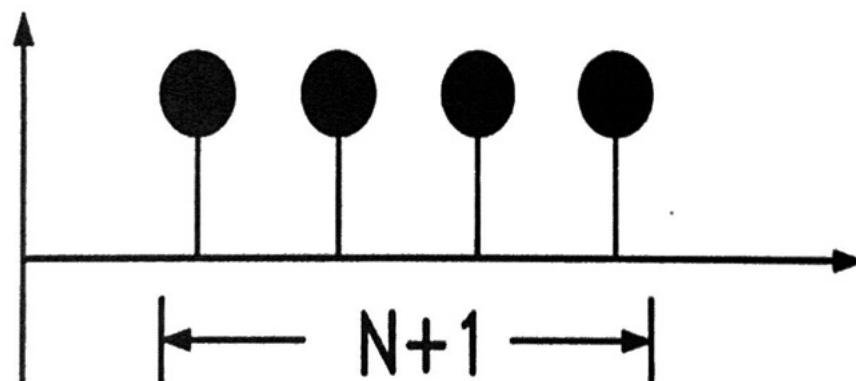


Filter Size = N (a positive integer)
Window Size = $2N+1$
Delay = N

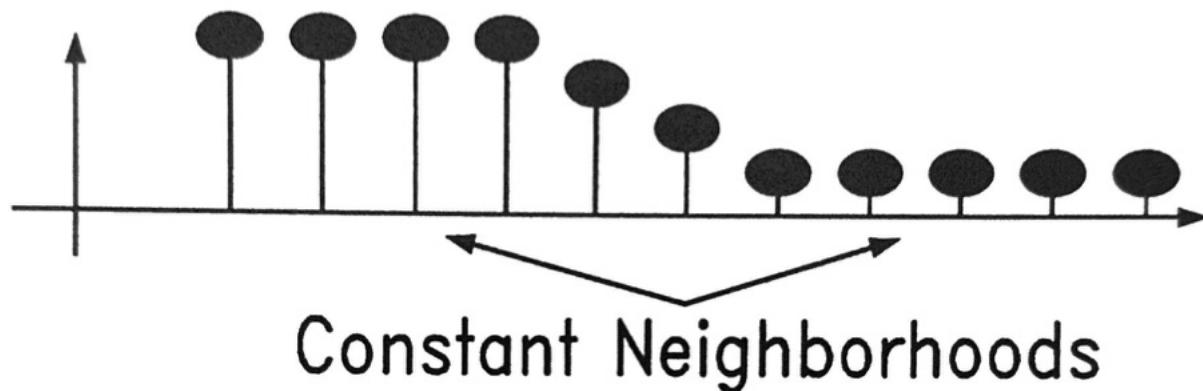
Geometric Analysis of the Median Filter

Definitions (for a filter of size N):

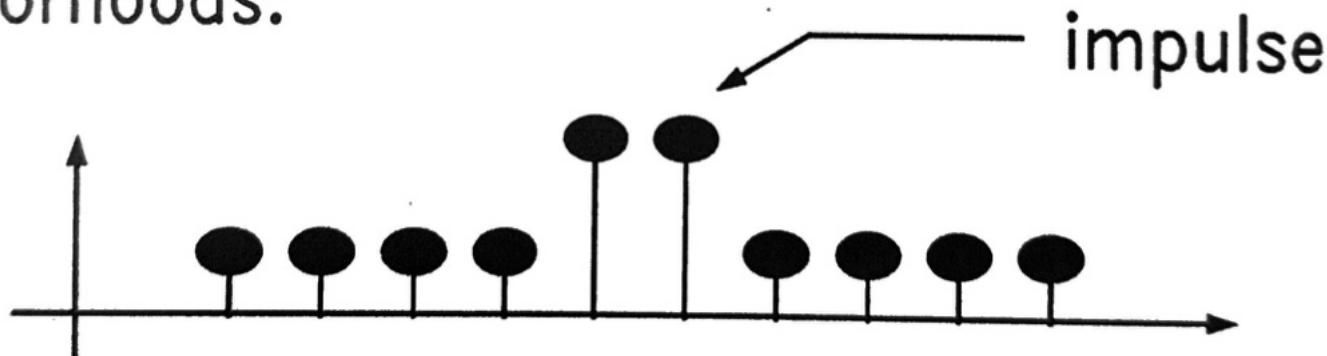
Constant Neighborhood- A section of at least $N+1$ consecutive, identically valued points:



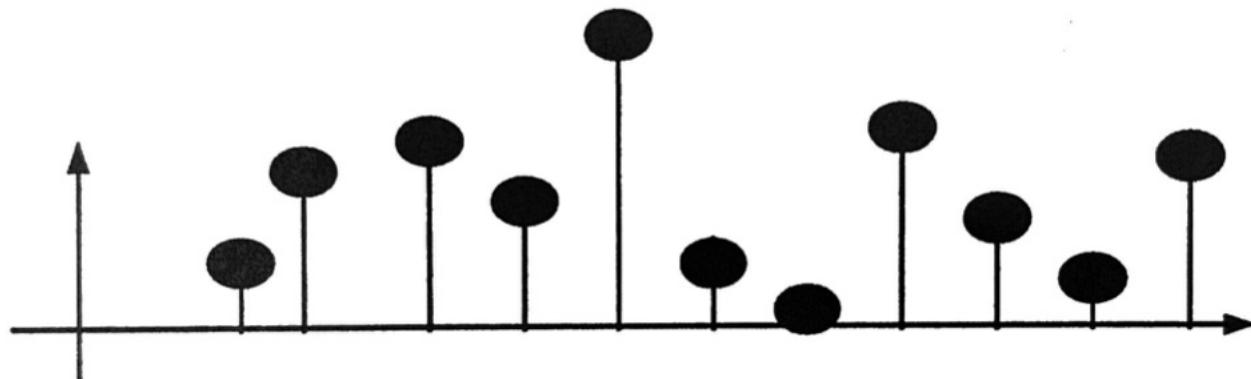
Edge— A monotonically rising or falling region between two constant neighborhoods:



Impulse— A section of one to N points surrounded by identically valued constant neighborhoods:



Oscillation – Any section that is not part of a constant neighborhood, an edge, or an impulse:



Root – A signal unaffected by a median filter of size N.

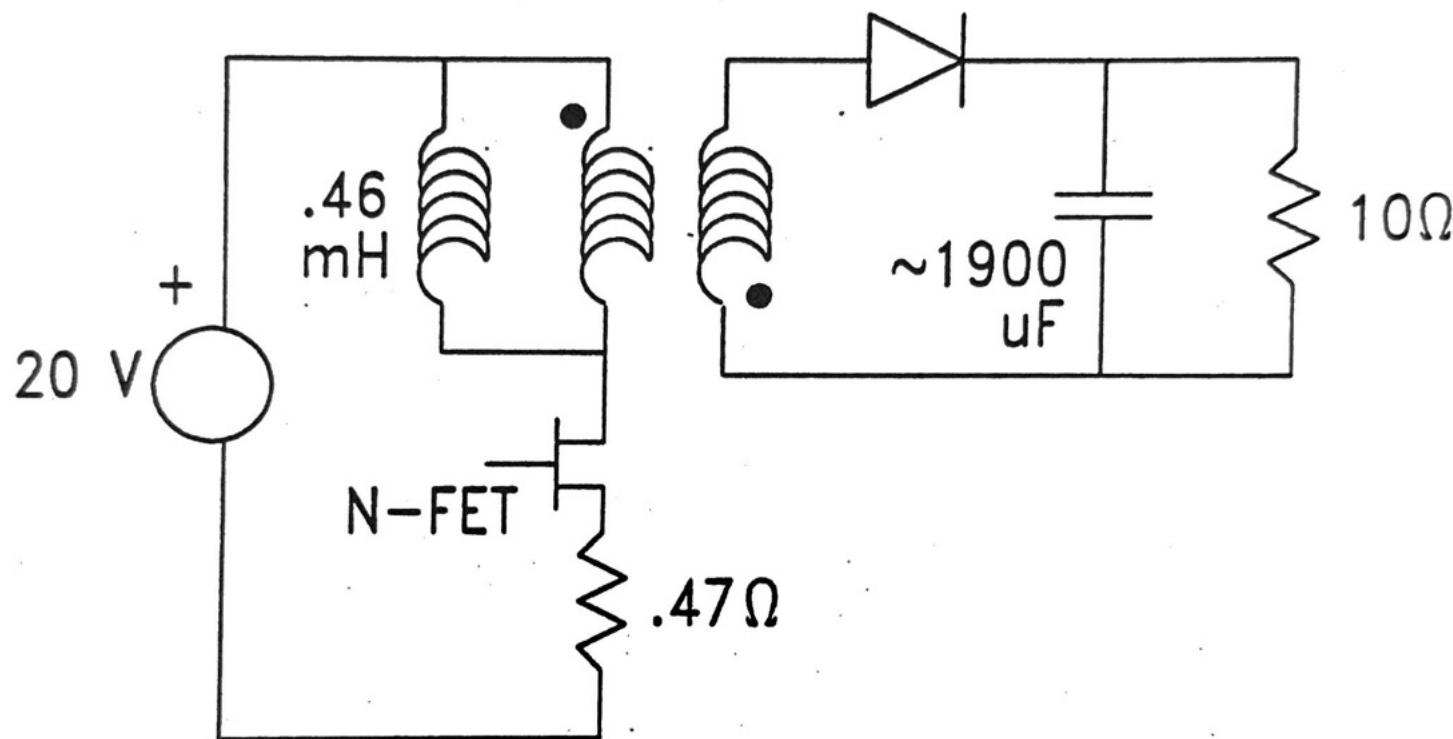
Some Properties of the Median Filter

Impulses are eliminated after a single pass of the median filter.

A root signal consists only of edges and constant neighborhoods.

In a root signal, increasing and decreasing regions must be separated by a constant neighborhood (at least $N+1$ constant points, *the geometric bandwidth*).

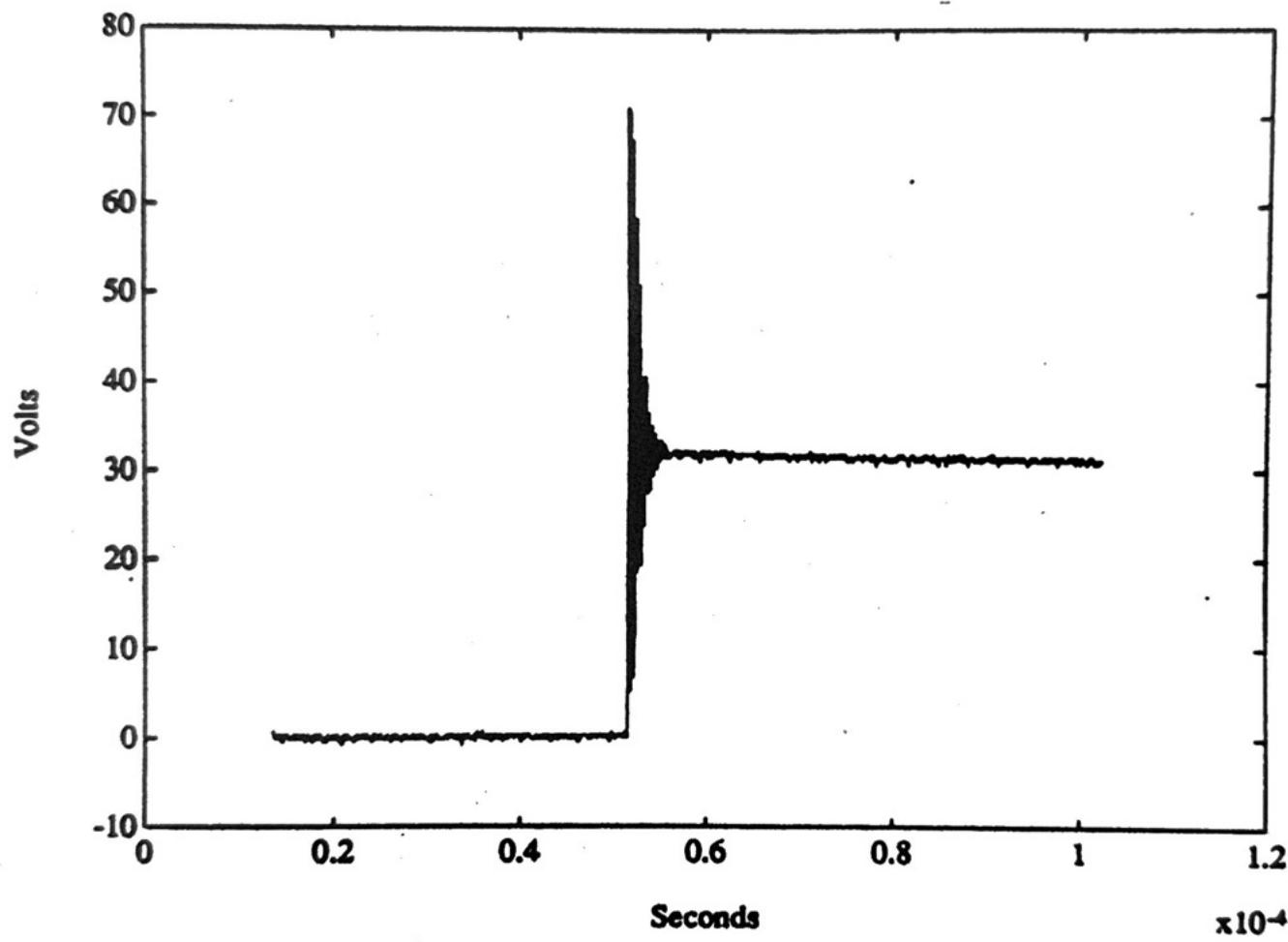
Flyback Converter



Switching Frequency = 5kHz

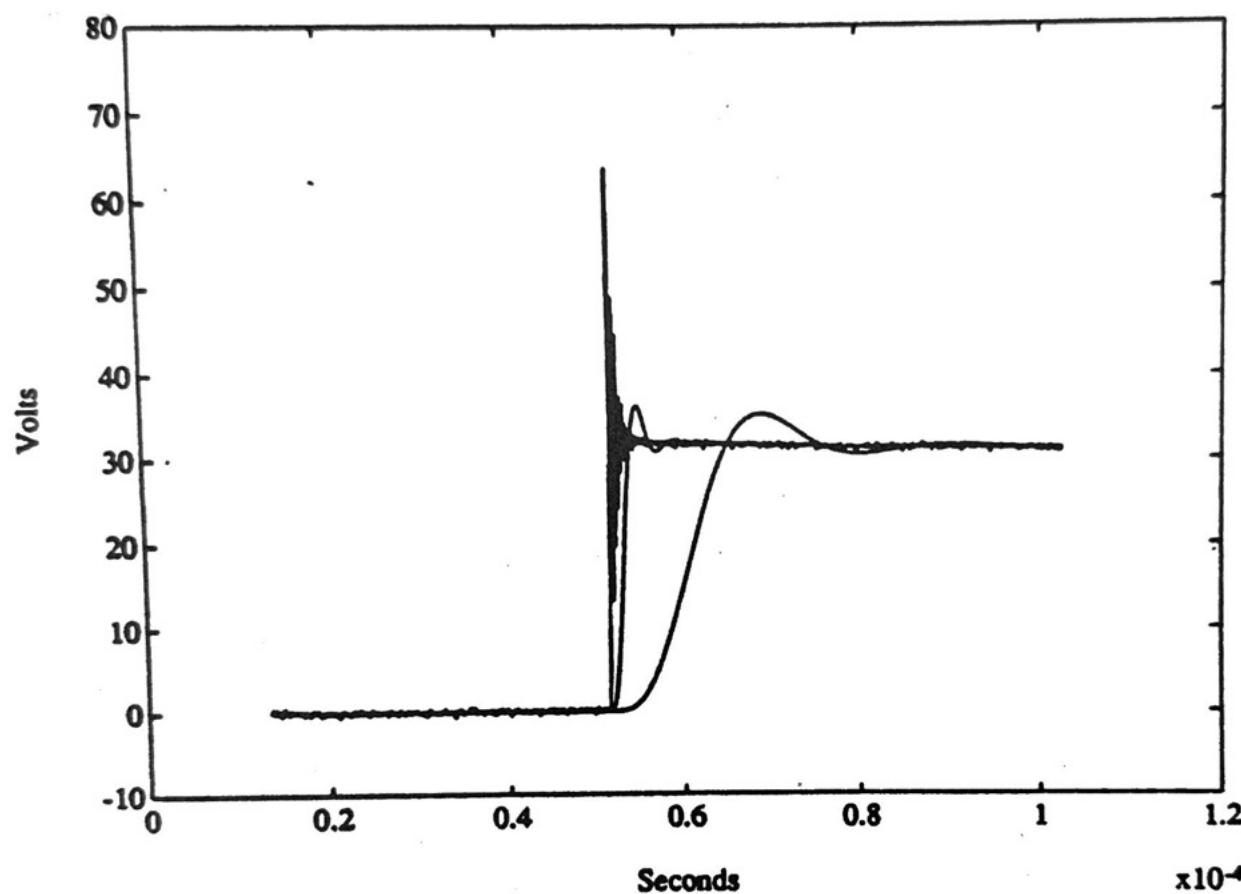
Output Power = 10 Watts

Switch Voltage During Turn-Off



Sample Rate = 10 MHz

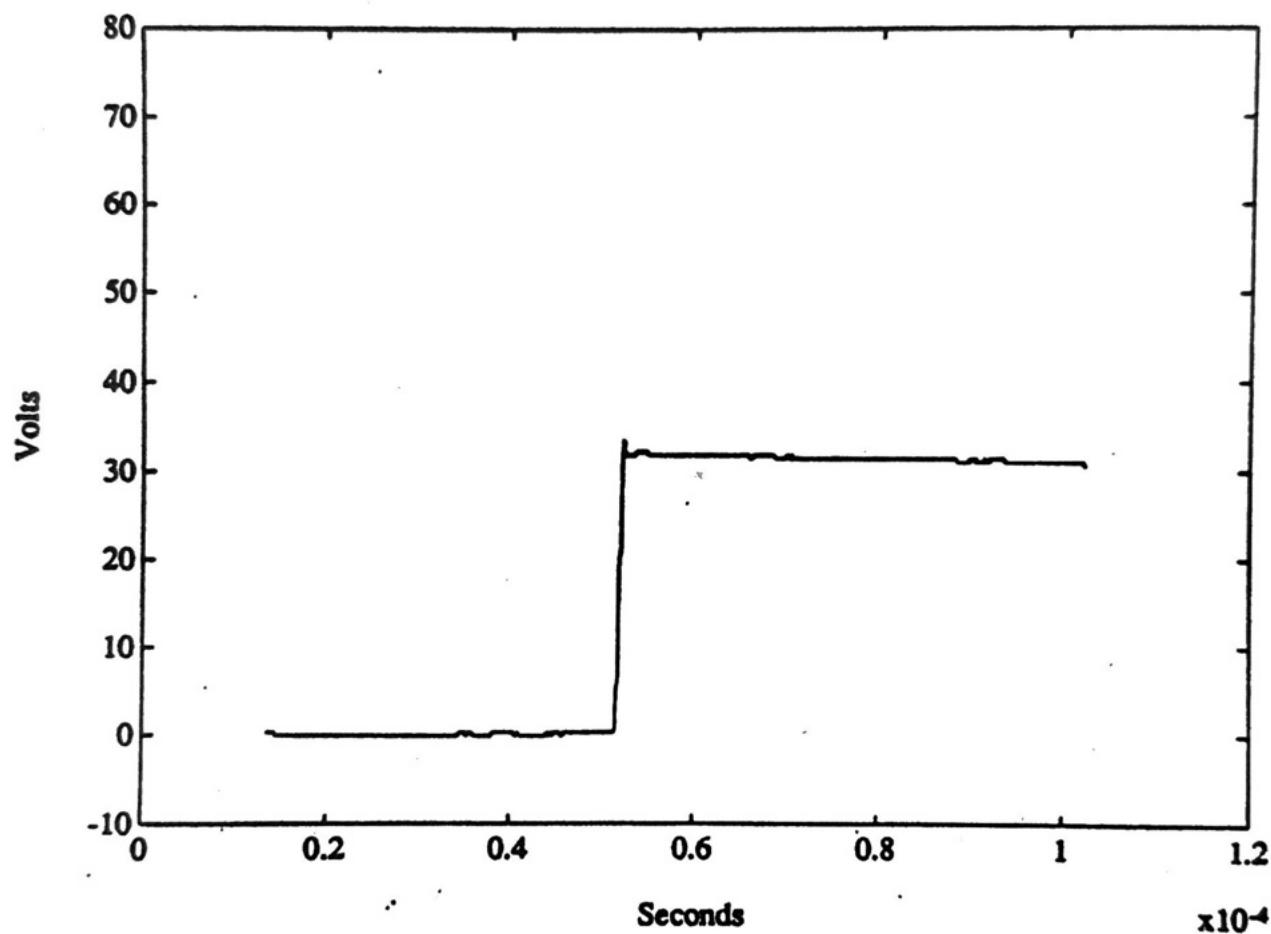
Lowpass Filtered Switch Voltage



Fourth-Order Butterworth Filters with cutoff frequencies at $0.5w$, $0.05w$, $0.01w$.

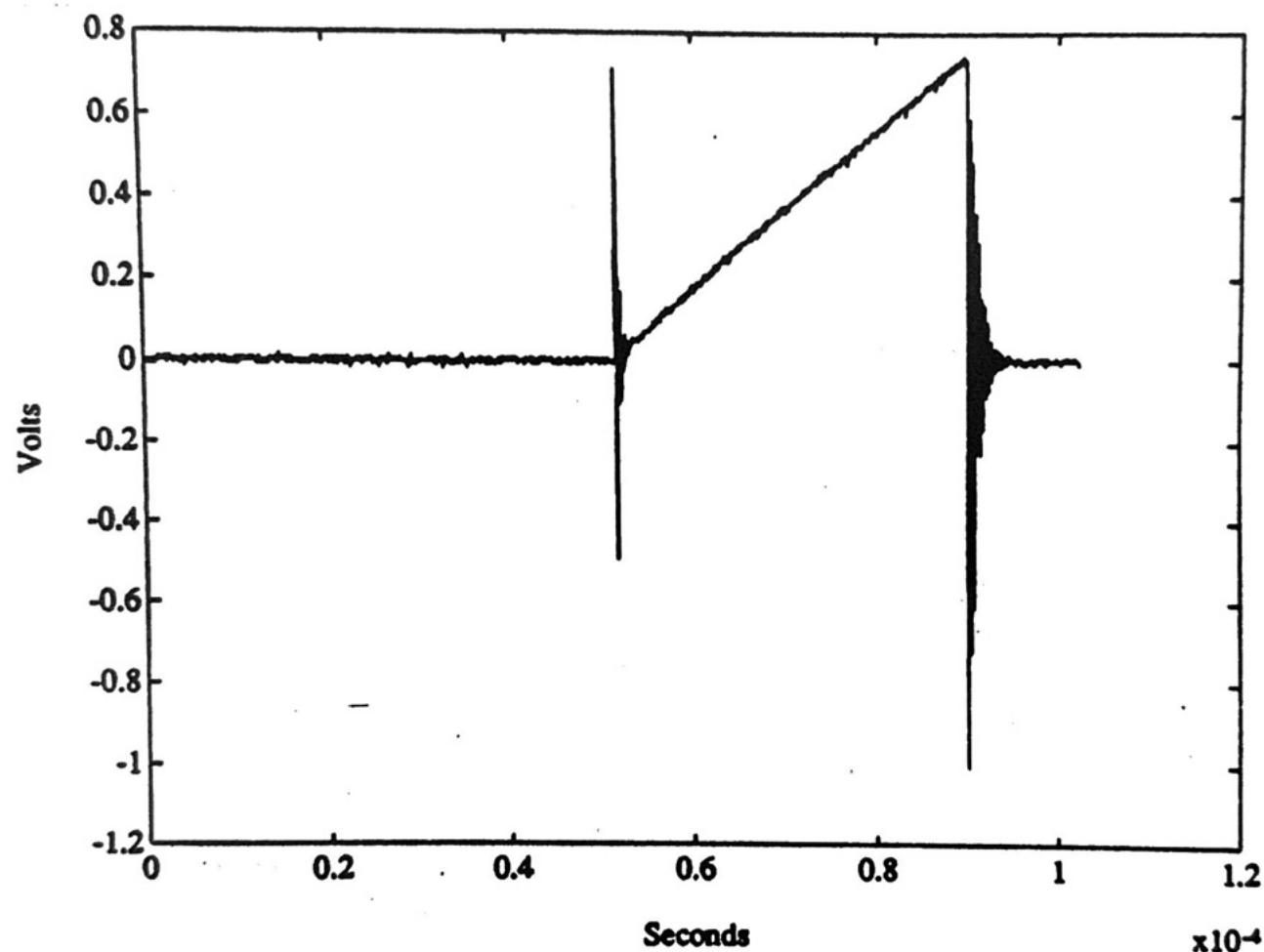
$$w = 2\pi \times 5 \times 10^7 \text{ rads/sec}$$

Median Filtered Switch Voltage



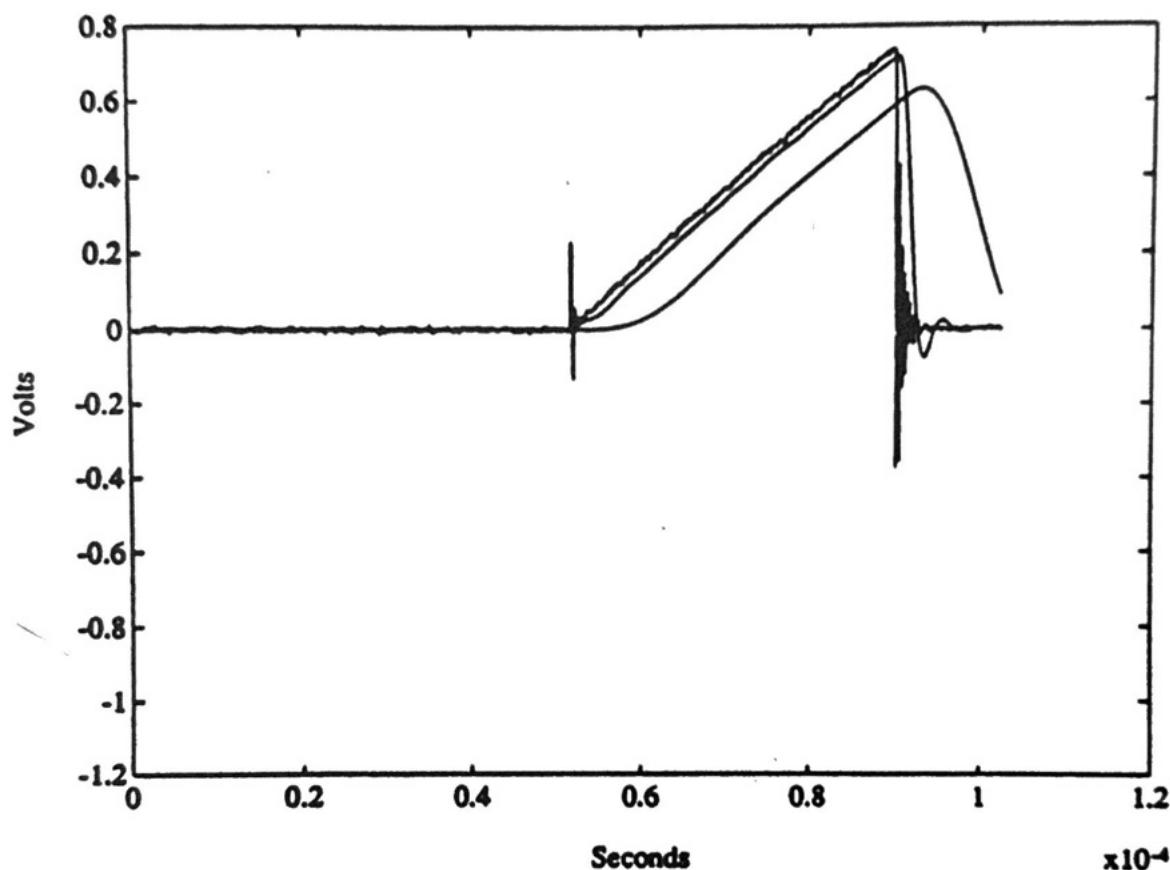
Filter Size N = 8

Voltage Across the Switch Current Sense Resistor



Sample Rate = 10 MHz

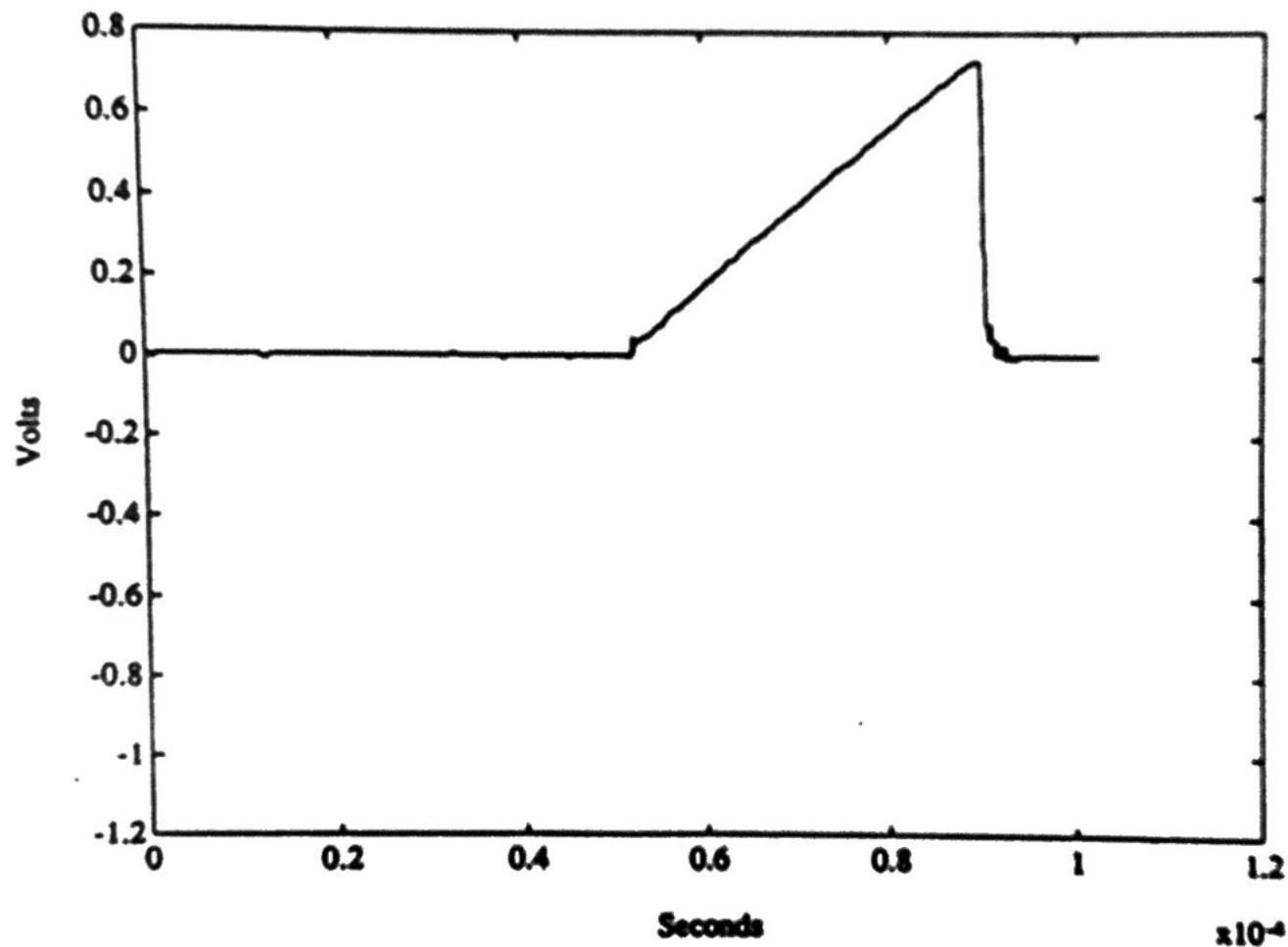
Lowpass Filtered Current Sense Voltage



Fourth-Order Butterworth Filters with cutoff frequencies at $0.5w$, $0.05w$, $0.01w$.

$$w = 2 \times \pi \times 5 \times 10^7 \text{ rads/sec}$$

Median Filtered Current Sense Voltage



Filter Size N = 5

Design Considerations

Median filters are nonlinear, shaping the local form or shape of signals. Examination of spectral behavior is of restricted value.

Trade-off: For greater smoothing, pick N large so that more signal structures appear as impulses and oscillations, which will be removed and reduced.

To preserve an underlying signal, the smallest structure of interest must be at least $N+1$ points long (the geometric bandwidth).

We have discussed his activities...

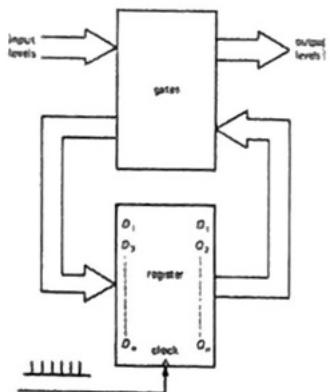


But who is he?

How is he constructed?

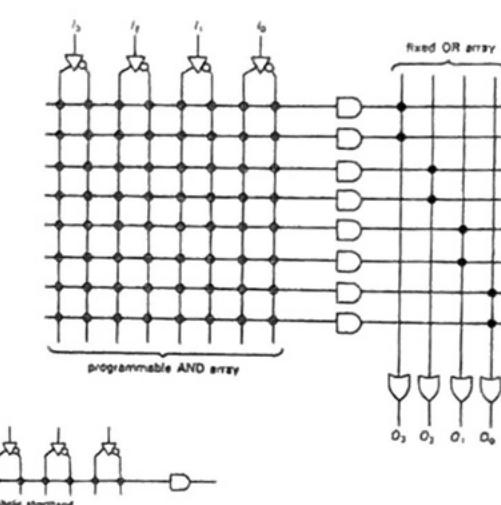
Can we make our own?

Finite State Machines:

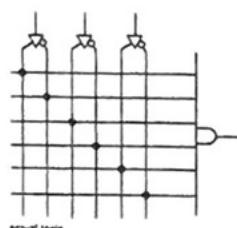


See Horowitz and Hill, 2nd edition, Chapter 8, for more details on these topics and descriptions of these included figures.

1

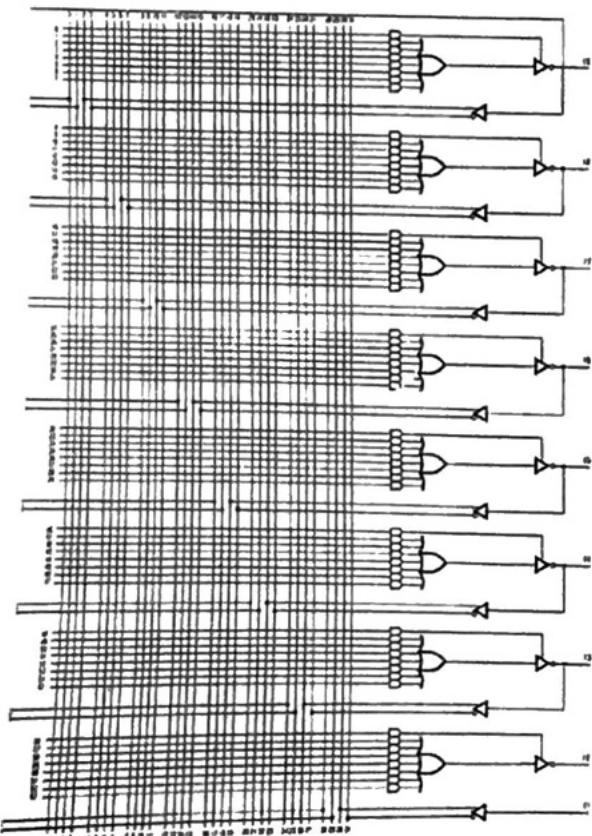


Note the schematic shorthand used to describe the programmable array



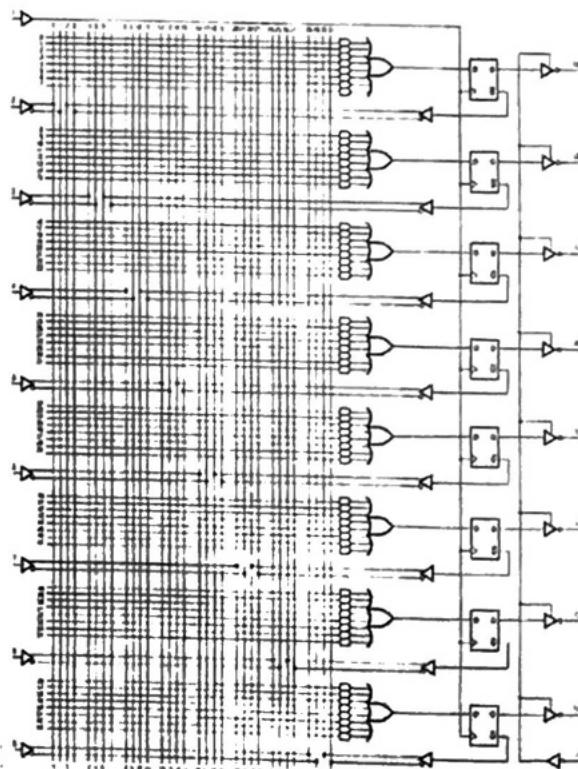
2

A combinational PAL (16L8, with 16 inputs max and 8 outputs max):



3

A PAL with registered outputs. The flip-flops can be used to make counters or as memory for "state". This is a 16R8.



4

A convenient way to make a bank of combinational logic, or a collection of gates programmable logic devices. We use one on our R31JP (Xilinx 9536 PLD) to create the XIO select line, and to switch memory positions for mon/run. Here's a simple model of part of a simple PAL.

DIP

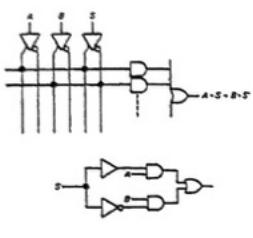
Here's the PAL in your chip kit:

It can be programmed with the CUPL language compiler, similar in many ways to RASM.

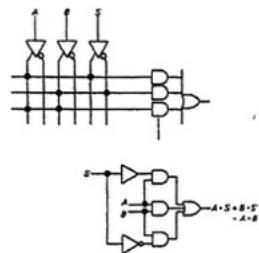
Be careful when programming to avoid poor logic, race conditions, designs that won't fit in the device, etc:

Two-Bit Selector:

With race condition:



With a fix for static race condition:



Display decoder, from Horowitz and Hill:



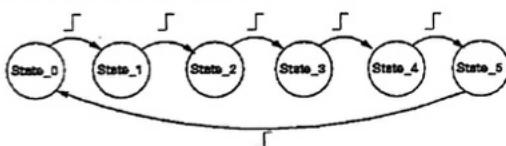
```
/** Inputs */
PIN 1 = a; /* segment a */
PIN 2 = b; /* segment b */
PIN 3 = c; /* segment c */
PIN 4 = d; /* segment d */
PIN 5 = e; /* segment e */
PIN 6 = f; /* segment f */
PIN 7 = g; /* segment g */

/** Outputs */
PIN 19 = ID3; /* msb of hex encode */
PIN 18 = ID2;
PIN 17 = ID1;
PIN 16 = ID0; /* lsb */

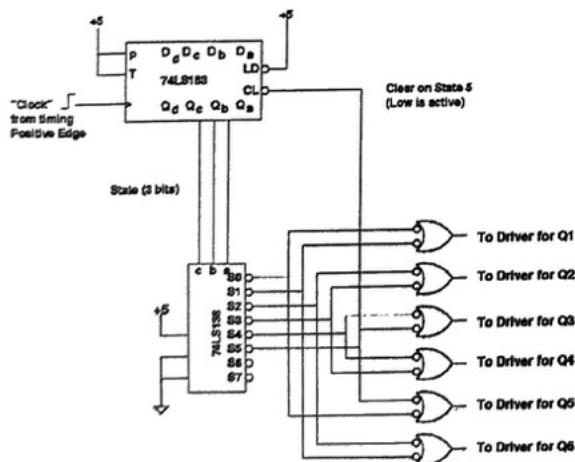
/** Declarations and Intermediate Variable Definitions */
zero = !a & b & c & d & e & f & !g;
one = !a & b & c & d & e & !f & !g;
two = a & b & c & d & e & !f & g;
three = a & b & c & d & !e & !f & g;
four = !a & b & c & d & !e & f & g;
five = a & !b & c & d & !e & f & g;
six = a & !b & c & d & e & f & g;
seven = a & b & c & d & e & !f & !g;
eight = a & b & c & d & e & f & g;
nine = a & b & c & d & !e & !f & g; /* two ways */
hex0 = a & b & c & d & !e & f & g;
hex1 = !a & b & c & d & e & f & g;
hex2 = !a & !b & c & d & e & f & g;
hex3 = !a & !b & !c & d & e & f & g; /* two ways */
hex4 = !a & !b & c & d & e & !f & g;
hex5 = !a & !b & !c & d & e & f & g;
hex6 = !a & !b & !c & d & !e & f & g;
hex7 = !a & !b & !c & d & !e & !f & g;

/** Logic Equations */
D3 = eight # nine # hexa # hexb # hexc # hexd # hexe # hexf;
D2 = four # five # six # seven # hexc # hexd # hexe # hexf;
D1 = two # three # six # seven # hexa # hexb # hexc # hexd;
D0 = one # three # five # nine # hexb # hexd # hexf;
```

Finite State Machine for a Motor Drive:



Hardware Implementation:



Implementation of a 138 selector in CUPL:

```
Name           demuxer;
Partno        ;
Revision      01;
Date          8/10/08;
Designer      PLD Expert;
Company       MIT;
Location      None;
Assembly      None;
Device        g22v10;
```

```
/* Simple combinatorial logic: 2-to-1 MUX */

pin 2 = a;
pin 3 = b;
pin 4 = c;
pin 5 = xiosel;

pin 14 = d;
pin 15 = e;
pin 16 = f;
pin 17 = g;
pin 18 = h;
pin 19 = i;
pin 20 = j;
pin 21 = k;

d = xiosel # !(a & !b & !c);
e = xiosel # !(a & !b & c);
f = xiosel # !(a & b & !c);
g = xiosel # !(a & b & c);
h = xiosel # !(a & !b & c);
i = xiosel # !(a & !b & c);
j = xiosel # !(a & b & !c);
k = xiosel # !(a & b & c);
```