University of British Columbia
Electrical and Computer Engineering
ELEC291/ELEC292

Lab 5 - Measuring Phasors

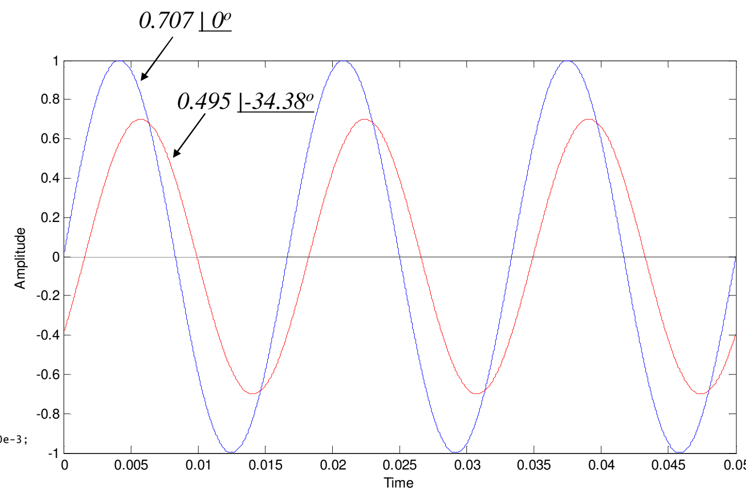
Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

March 4, 2022

Objectives

- Understand what is a phasor.
- Measure sine wave amplitude.
- Measure sine wave phase.
- Work with Makefiles (time permitting).

What is a Phasor?




```
t=0:0.01e-3:50e-3;
f=60;
w=2*pi*f;
y=sin(w*t);
z=0.7*sin(w*t-0.6);
plot(t,y,'b',t,z,'r',t,0*y,'k-');
xlabel('Time')
ylabel('Amplitude')
```

Matlab script

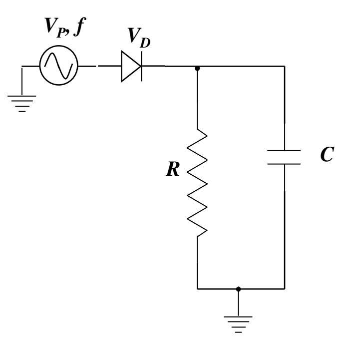
What do we need?

- Two sinusoidal signals. Without any DC, preferably! The function generators in the lab can do that.
- A circuit or method to measure the peak value of the signals. Notice that phasors use RMS voltages/currents. We may need to convert V_{peak} to V_{RMS}.
- A circuit or method to measure the phase difference. We will be using degrees.

Measuring Peak AC or RMS

- We can use different methods:
  - Peak detector. For good accuracy, a precision peak detector may be used. (**Good for project 2, but not for Lab #5**)
  - One shot measurement of the peak value by precisely timing the input waveform using a zero cross detector.
  - Fast ADC sampling:
    - look for the max value of the wave.
    - computation of the RMS using floating point arithmetic.
  - Direct RMS measurement using a RMS to DC converter such as the **AD536A** IC. Usually very expensive: 21$ to 112$!

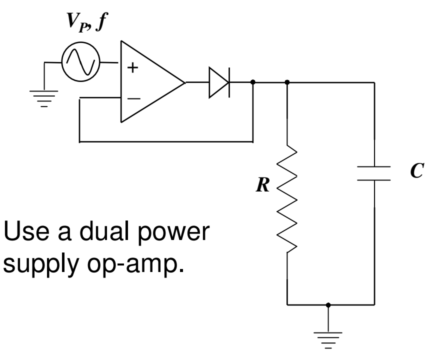
Simple Peak Detector



$$V_r = \frac{V_P - V_D}{f \times R \times C}$$

Something like this will be needed for project 2.

Precision Peak Detector



$$V_r = \frac{V_P}{f \times R \times C}$$

Use a dual power supply op-amp.

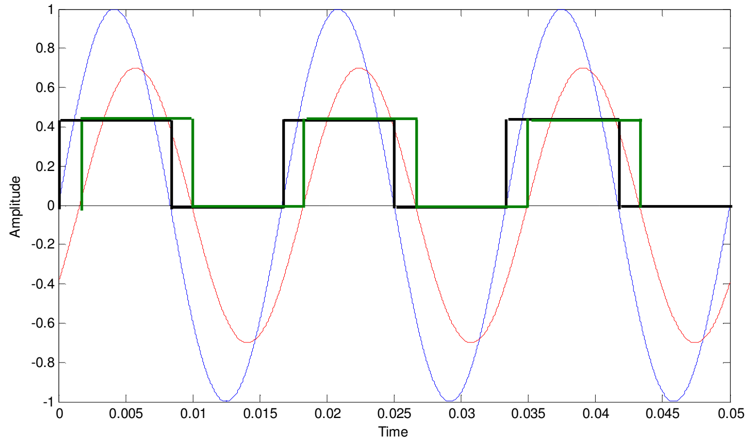
Good for measuring the magnitude of a phasor!

Warning: it works really well, but it is a lot of work to assemble two of these circuits for lab #5!

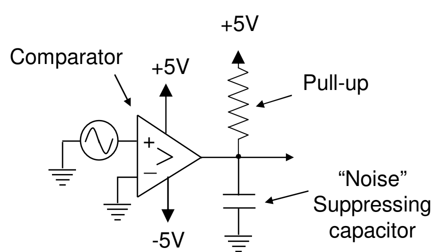
Zero Cross Detection

- If we know were the two waves cross zero we can:
  - Find the frequency of the waves.
  - Determine where the peak value is ’located’.
  - Measure the phase difference between the two waves.
- Since we need to know when the waves cross zero in order to determine their phase difference, we may as well use it to determine where the peak voltage is and measure it!

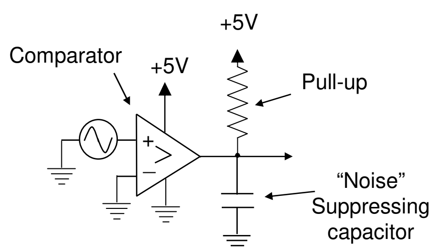
Zero Cross Detector Signals and Sine Waves



Zero Cross Detection

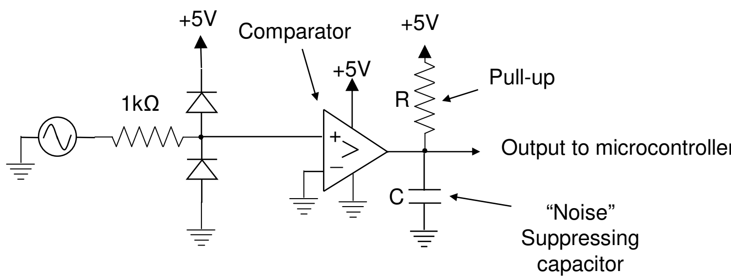


Works for any input voltage from -5V to +5V, but the output voltage of the comparator is either -5V or +5V



"Works" only for positive voltages from 0V to 5V, but the output voltage of the comparator is now either 0V or +5V.

Zero Cross Detection

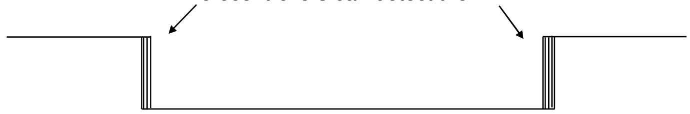


Works for "any" AC input voltage.
Comparator output is either 0V or +5V.

The 1kΩ resistor is used to limit the current throughout the diodes in case of over/under voltage.

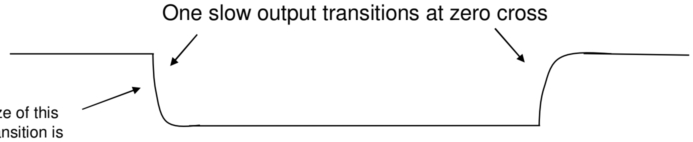
Noise suppressing capacitor

Multiple fast output transitions at zero cross. Modern fast microcontrollers can detect them!



Output signal without noise suppressing capacitor

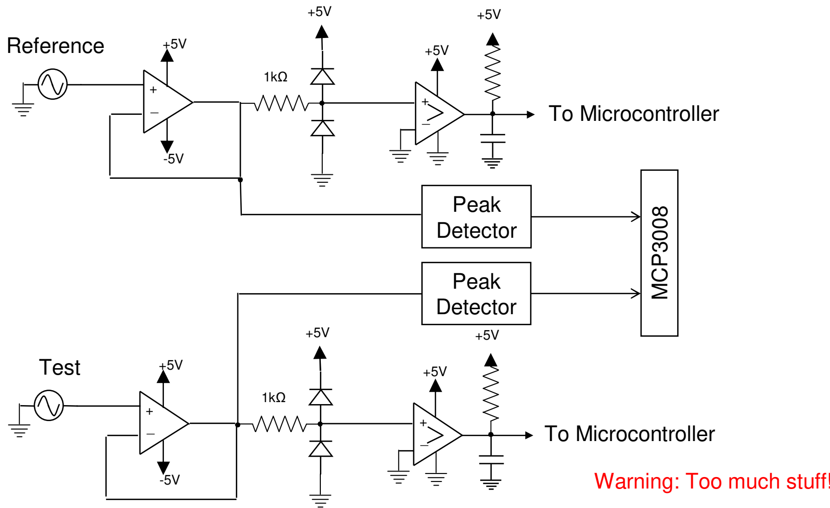
One slow output transitions at zero cross



The size of this slow transition is determined by tau=RC

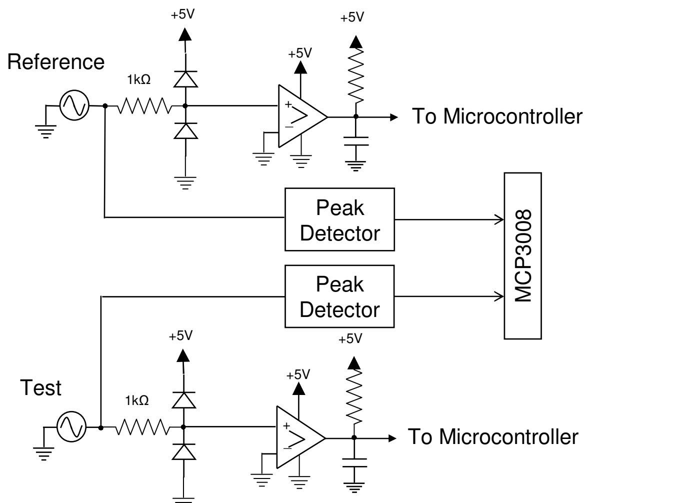
Output signal WITH noise suppressing capacitor

Zero Cross Detection & Peak Detector to ADC with high impedance inputs.

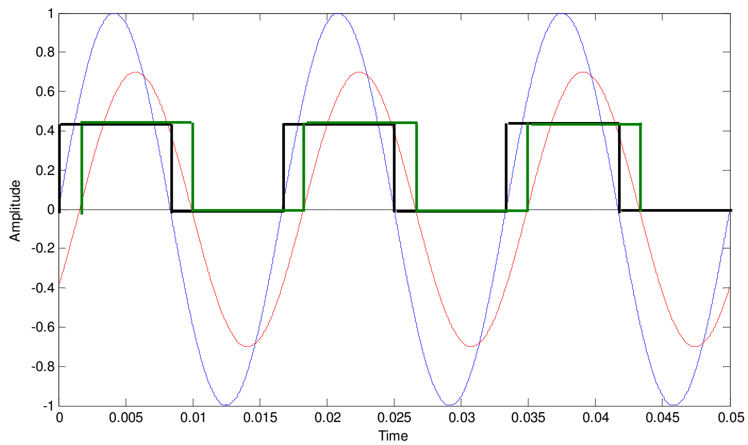


Warning: Too much stuff!

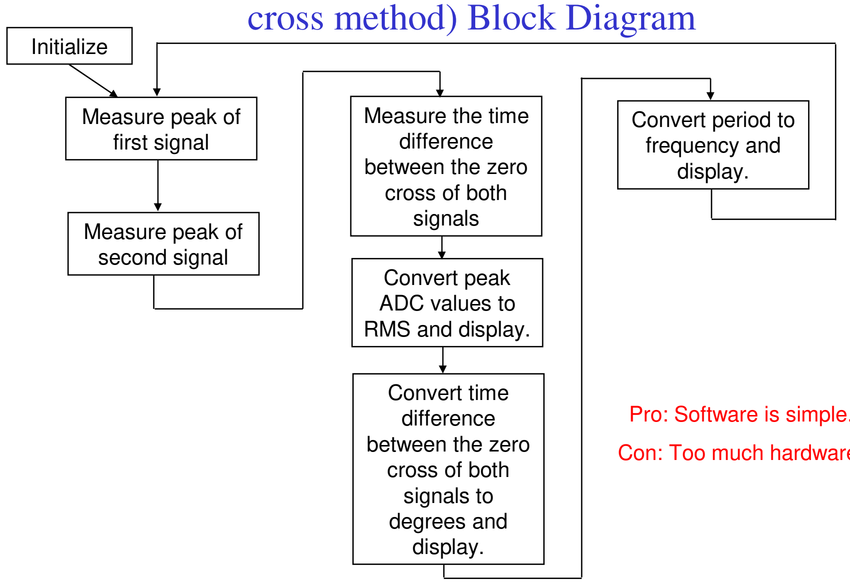
Zero Cross Detection & Peak Detector to ADC with low impedance inputs.



Zero Cross Detector Signals and Sine Waves



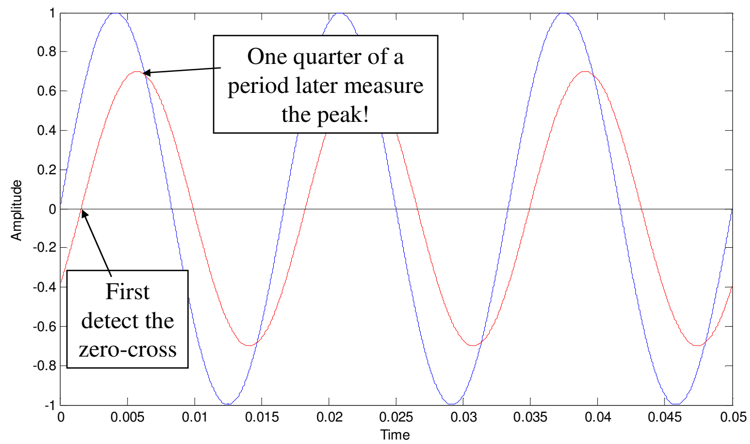
Phasor Measurement (using peak detector and zero cross method) Block Diagram



Pro: Software is simple.

Con: Too much hardware.

One Shot Measurement of the Peak Value with the ADC



MCP3008 Analog Input Model

Taken from the MCP3008/MCP3004 Data Sheet

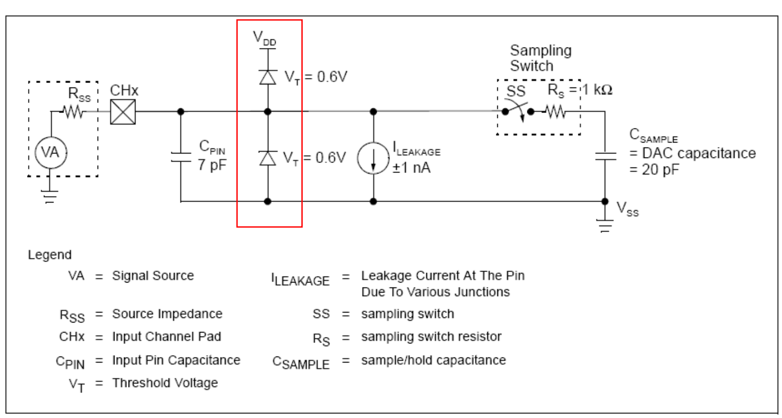
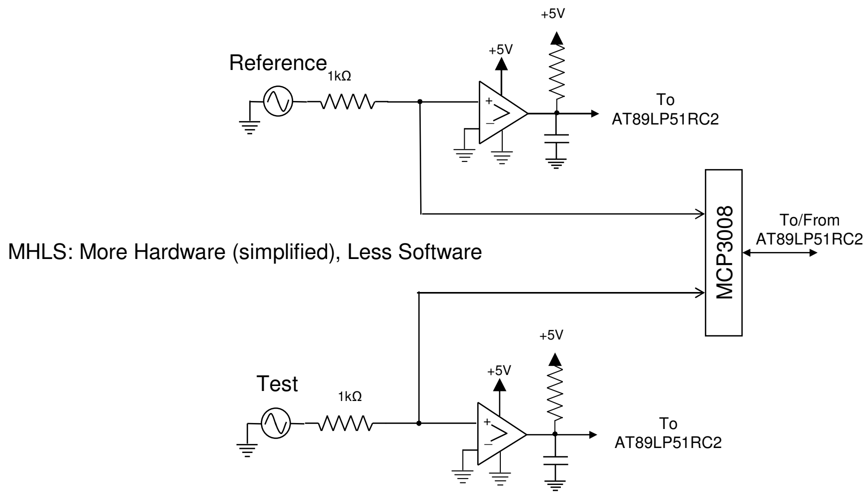


FIGURE 4-1: *Analog Input Model.*

One Shot Measurement of the Peak Value with the ADC and Zero Cross



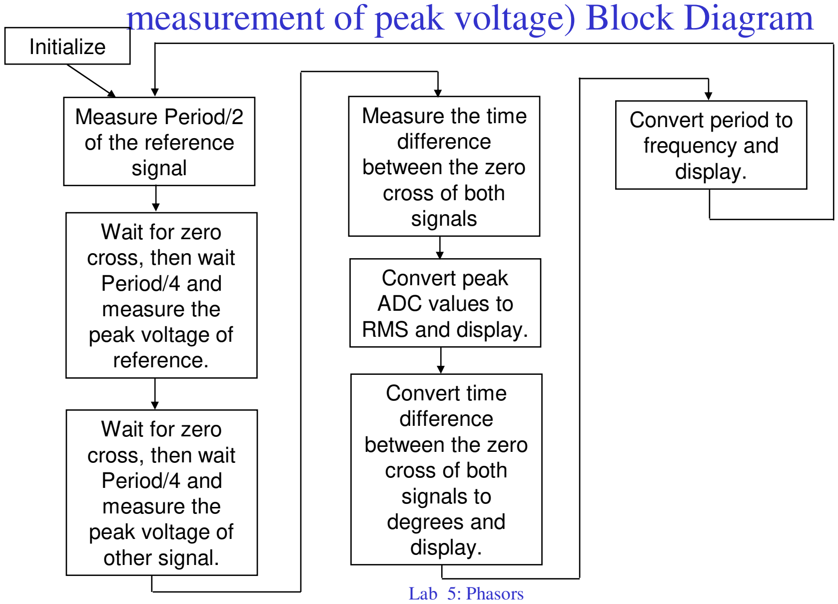
Measuring Half Period Using Timer 0

```
// Measure half period at pin P1.0 using timer 0
TR0=0; // Stop timer 0
TMOD&=0B_1111_0000; // Set timer 0 as 16-bit timer (step 1)
TMOD|=0B_0000_0001; // Set timer 0 as 16-bit timer (step 2)
TH0=0; TL0=0; myof=0; // Reset the timer and overflow counter
TF0=0; // Clear overflow flag
while (P1_0==1); // Wait for the signal to be zero
while (P1_0==0); // Wait for the signal to be one
TR0=1; // Start timing
while (P1_0==1) // Wait for the signal to be zero
{
    if (TF0) {TF0=0; myof++; }
}
TR0=0; // Stop timer 0
// [myof,TH0,TL0] is half the period in multiples of 1/CLK, so:
// (Assume Period is float)
Period=(myof*65536.0+TH0*256.0+TL0)*2.0;
```

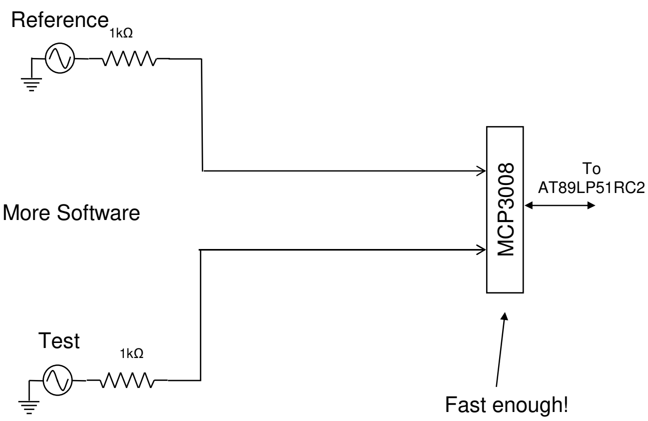
Half Period of the Reference Signal.

- After the code in the previous slide is executed we have half the period in [myof, TH0, TL0] in units of 1/CLK. For the AT89LP51RC2 the CLK is 22.1184MHz.
- We can do two things with half the period:
  - Multiply it by two to obtain the period, then get its inverse to obtain the frequency.
  - Divide it by two to find where the peak of the sine wave occurs.

Phasor Measurement (using zero cross and one shot measurement of peak voltage) Block Diagram

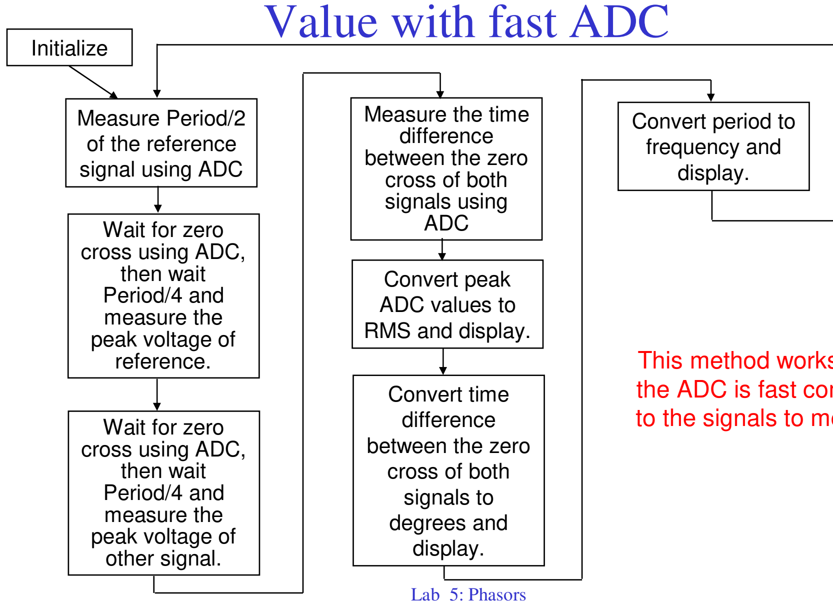


One Shot Measurement of the Peak Value with **FAST** ADC



LHMS: Less Hardware, More Software

One Shot Measurement of the Peak Value with fast ADC



This method works only if the ADC is fast compared to the signals to measure.

Measure half period using the Fast ADC (version 1)

```
// Configure timer 0, leave time 1 alone!
TMOD&=0B_1111_0000; // Set timer 0 as 16-bit timer (step 1)
TMOD|=0B_0000_0001; // Set timer 0 as 16-bit timer (step 2)
// Reset the timer and overflow counters
TL0=0; TH0=0; myof=0;
while (GetADC(0)!=0); // Wait for the signal to be zero
while (GetADC(0)==0); // Wait for the sig. to be positive
TF0=0; // Clear overflow flag
TR0=1; // Start timer 0
while (GetADC(0)!=0) // Wait for the sig. to be zero again
{
    if (TF0) { TF0=0; myof++; }
}
TR0=0; // Stop timer 0. [myof-TH0-TL0] is the period in units of 1/CLK
// half_period below is a float variable
half_period=myof*65536.0+TH0*256.0+TL0; // The 24-bit number [myof-TH0-TL0]
```


Measure half period using the "Fast" ADC MCP3008 (version 2, with hysteresis)

```
TMOD&=0xf0; // Clear the configuration bits for timer 0
TMOD|=0x01; // Mode 1: 16-bit timer

// Measure half period at ADC CH0 using timer 0
TF0=0; // Clear overflow flag
TL0=0; // Reset the timer
TH0=0;
OVcnt=0;
while (GetADC(0)>2); // Wait for the signal to be zero
while (GetADC(0)<4); // Wait for the signal to be one
TR0=1; // Start timing
while (GetADC(0)>2) // Wait for the signal to be zero
{
    if (TF0)
    {
        TF0=0;
        OVcnt++;
    }
}
TR0=0; // Stop timer 0
half_period=OVcnt*65536.0+TH0*256.0+TL0; // half_period is "float"
```

MHLS vs. LHMS

- LHMS
  - Cheaper to make
  - More flexible
  - May not work with fast signals
  - Good for Lab #5
- MHLS
  - Works with slow processors
  - Works with slow ADCs
  - Works with fast signals
  - Less flexible
  - Good for Project #2

Incorrect (or noisy) measurements?

- Measure twice, display once!

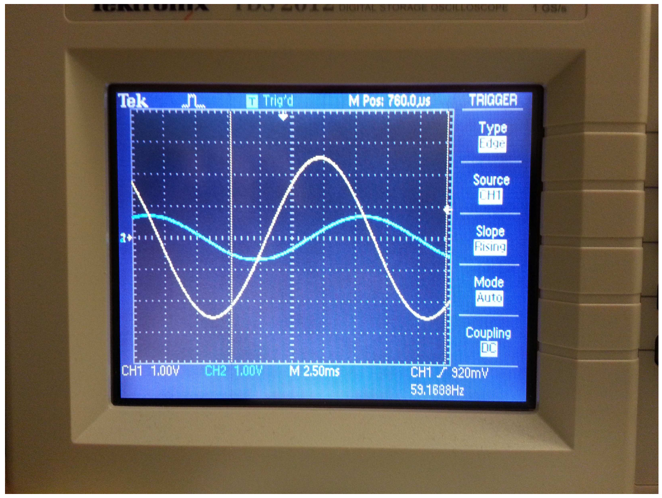
```
if (fabsf(f-p_f)>0.5) goto donotdisplay;
if (fabsf(v0-p_v0)>0.05) goto donotdisplay;
if (fabsf(v1-p_v1)>0.05) goto donotdisplay;
if (fabsf(phase-p_phase)>1.0) goto donotdisplay;
.
.
.
donotdisplay:
p_f=f;
p_v0=v0;
p_v1=v1;
p_phase=phase;
```

fabsf() is defined in math.h.

At the top of your code add:

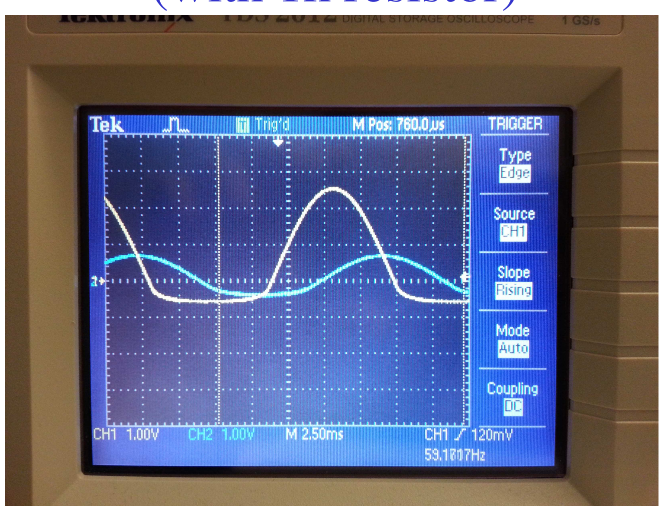
```
#include <math.h>
```

The Reference and Test Signals



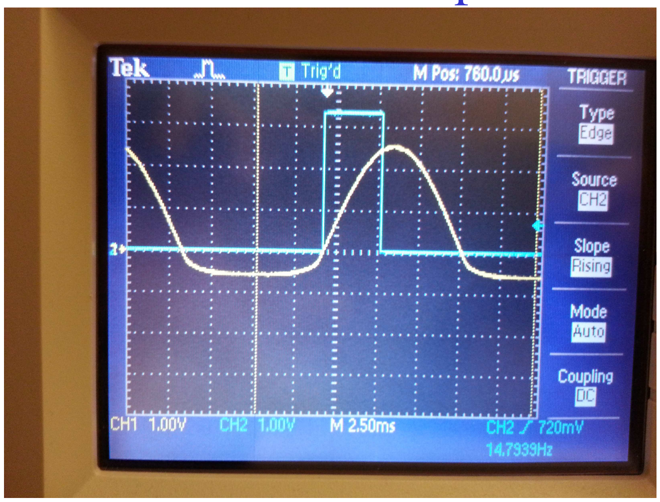
Reference: CH1 (yellow)
Test: CH2 (blue)

Signals connected to the MCP3008 (with 1k resistor)



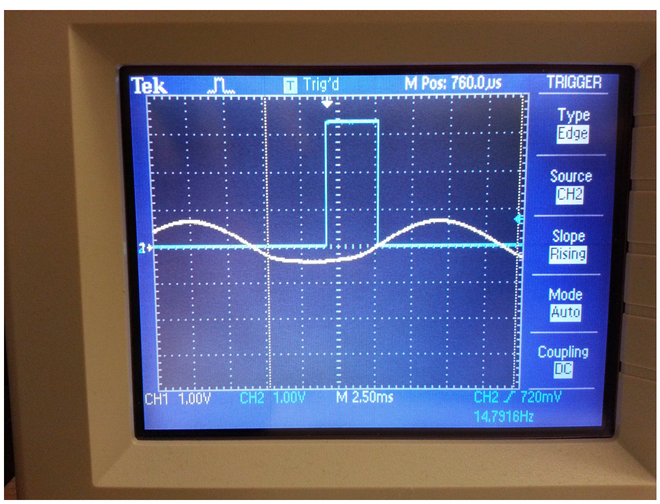
Only the positive part of the sine waves are correct!

Using P0.0 to debug ‘times’ using the oscilloscope



The debug pin (CH2, P0.0) is set to one when the ‘Reference’ wave crosses zero and clear to zero when the ‘Test’ signal crosses zero

Using P0.0 to debug using the scope

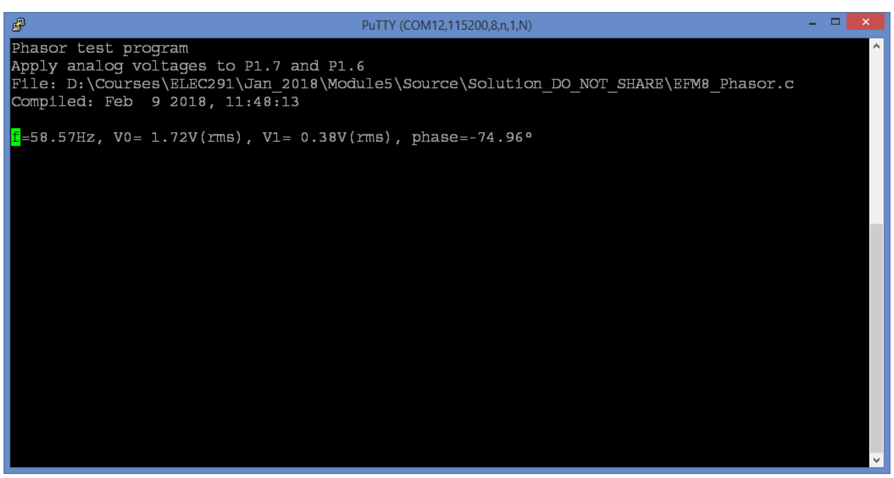


The debug pin (CH2, P0.0) is set to one when the ‘Reference’ wave crosses zero and clear to zero when the ‘Test’ signal crosses zero

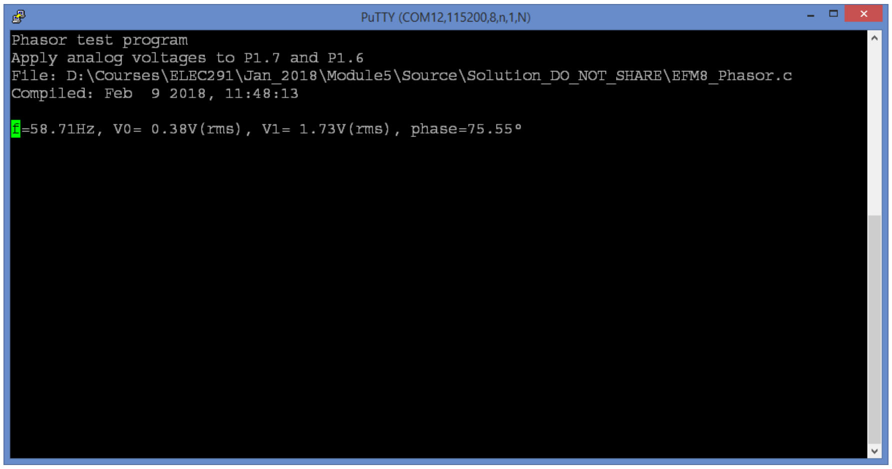
How to compute the phase difference in degrees

- One complete period of the wave is equivalent to 360°.
- The width of the pulse from the previous slide is about 3.5ms, and the period of the wave is about 16.9ms.
- Therefore the phase difference is 3.5*(360°/16.9) =74.56°.

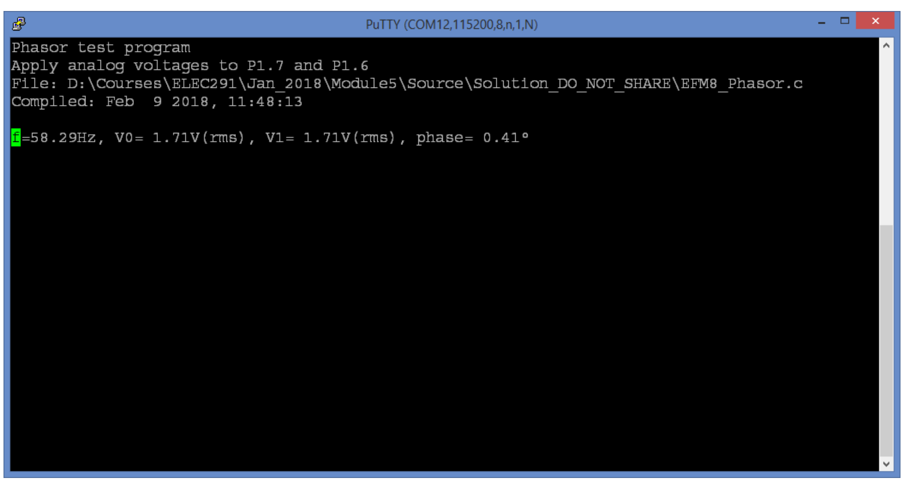
Output (Original Circuit)



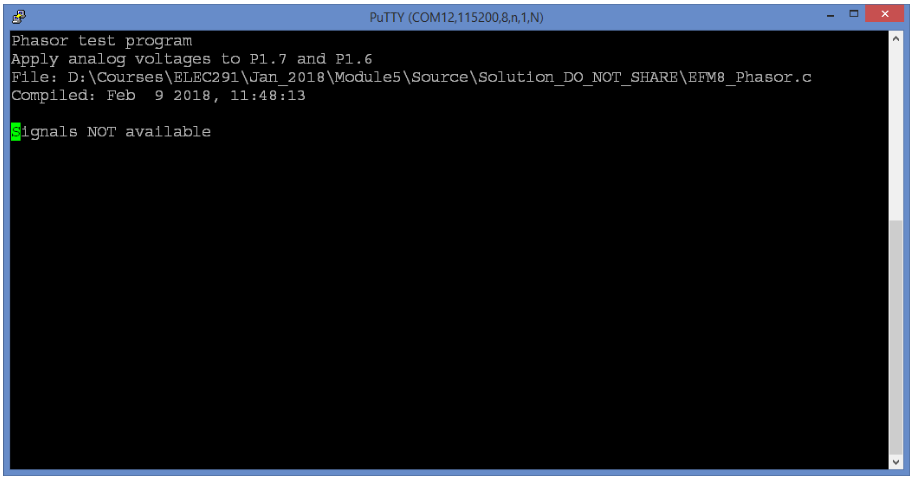
Output (Signals Swapped)



Output (Same Signal Both Inputs)



Output (no signals connected)



Working With Makefiles (time permitting)

- Industry standard.
- Easy compilation and linking of multiple files.
- Works for any processor, compiler, or operating system.
- Allows to automate tasks.
- Permits the execution of external programs, for example the flash loader.

Working With Makefiles in CrossIDE

To work with Makefiles for the EFM8 board we need these programs:

- CrossIDE V2.26 (or newer) & GNU Make V4.2 (or newer). In CrossIDE installation.
- CALL51 Toolchain for Windows. In CrossIDE installation.
- EFM8 Flash Loader: EFM8_prog. Available on Connect.
- PuTTy.

Available on Connect

- All the Makefile projects as a zip file.
- The "EFM8 Microcontroller System" with the instructions on how to use the Makefiles.
- Soon: Instructions and Makefiles for the ATmega328, PIC32MX130, SAMD20, LPC824, and MSP430 processors. (Needed for Lab #6 and project #2)

Setting the Path for the Programs

- The folder of these programs: GNU make, the compiler, the flash loader, and PuTTy must be available in the PATH environment variable of your computer.
- Also there is a little program called "wait.exe" that is used by some of the Makefiles. Its folder must be in the PATH as well.
- Instructions to set the PATH are available here:

  http://www.computerhope.com/issues/ch000549.htm

CrossIDE Workflow With Makefiles

- Creation and Maintenance of Makefiles.
- Using Makefiles with CrossIDE: Compiling, Linking, and Loading.
- Testing.

Example Makefile

```
# Specify the compiler to use
CC=c51
# Object files to link
OBJS=Blinky.obj

# The default 'target' (output) is Blinky.hex and 'depends' on
# the object files listed in the 'OBJS' assignment above.
# These object files are linked together to create Blinky.hex.
Blinky.hex: $(OBJS)
        $(CC) $(OBJS)
        @echo Done!

# The object file Blinky.o depends on Blinky.c. Blinky.c is compiled
# to create Blinky.o.
Blinky.obj: Blinky.c
        $(CC) -c Blinky.c

# Target 'clean' is used to remove all object files and executables
# associated wit this project
clean:
        @del $(OBJS) *.asm *.lkr *.lst *.map *.hex *.map 2> nul

# Target 'FlashLoad' is used to load the hex file to the microcontroller
# using the flash loader.  If the folder of the flash loader has been
# added to 'PATH' just 'EFM8_prog' is needed.  Otherwise, a valid path
# must be provided as shown below.
LoadFlash:
        EFM8_prog Blinky.hex

# Phony targets can be added to show useful files in the file list of
# CrossIDE or execute arbitrary programs:
Dummy: Blinky.hex Blinky.Map

explorer:
        explorer .
```

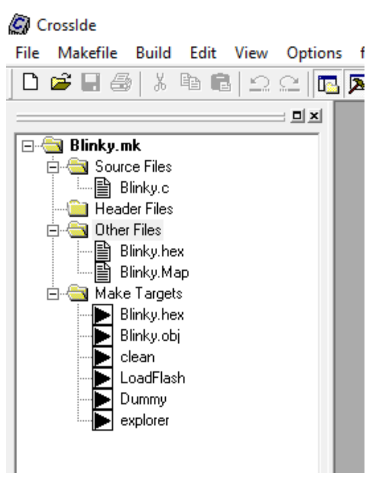
Target

Dependencies

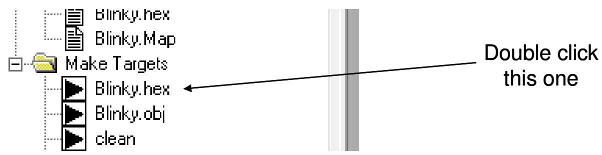
Actions

Makefile as a CrossIDE Project

To open a Makefile in CrossIDE, click “Makefile”→”Open” and select the Makefile to open.

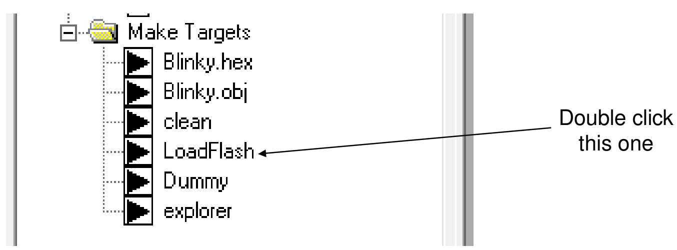


Compiling/Linking



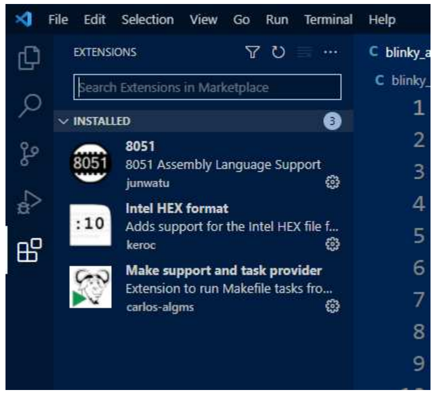

```
----------------- CrossIde - Running Make -----------------
c51 -c Blinky.c
c51 Blinky.obj
Done!
```


Load Flash Memory




```
----------------- CrossIde - Running Make -----------------
EFM8_prog.exe -ft230 -r Blinky.hex
Serial flash programmer for the EFM8LB1. (C) Jesus Calvino-Fraga (2012-2017)
Connected to COM12
Blinky.hex: loaded 208 bytes
.Found EFM8LB12F64E_QFP32.  Id: 0x3442
Sending 'setup' command... Done.
Erasing flash memory...
#############################################################
#############################################################
##### Done.
Writing flash memory...
## Done.
Verifying...  Done.
Running program...  Done.
Actions completed in 2.2 seconds.
```

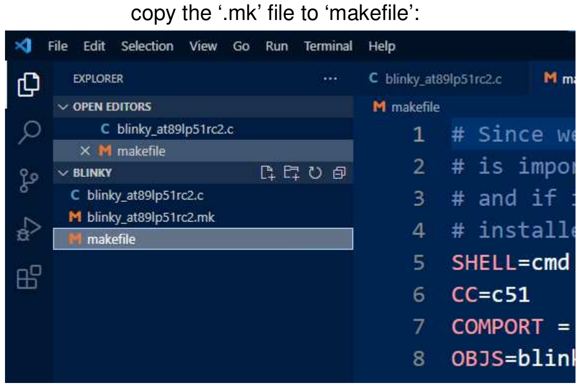
Makefiles with VS Code

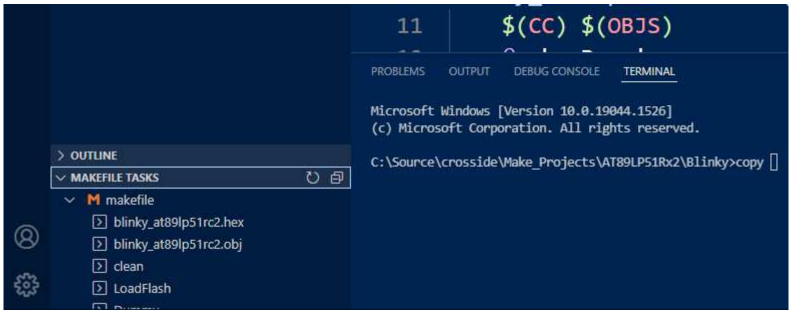


Install the extension “Make support and task provider”

Makefiles with VS Code

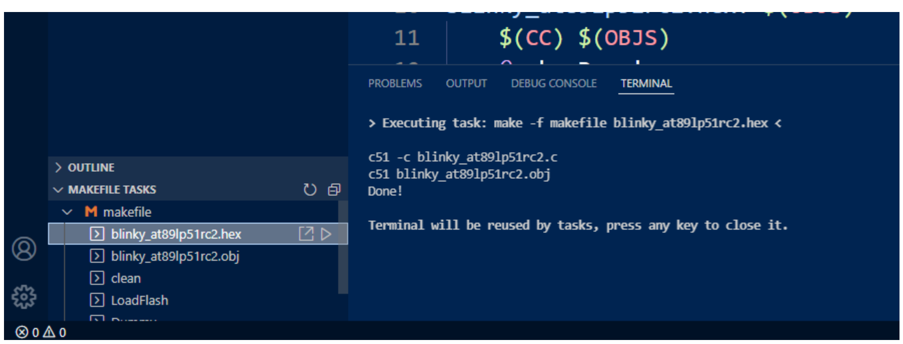
The makefile extension undertand only the file ‘makefile’. So copy the ‘.mk’ file to ‘makefile’:





The bottom left shows all the 'MAKEFILE TASKS' tab. If you expand, you should see the tasks available. To build (compile + link) click the .hex task:

Makefiles with VS Code



To load the microcontroller, click 'LoadFlash':