Pico Block Diagram Programming

This Block Diagram Compiler is intended for educational purposes. It should not be used for control or applications where safety and/or reliability would be of issue.

The Pico Block circuit board is limited at 0 to 3.3V for digital I/O and ADC channel 2. Analog DAC outputs and ADC inputs have a range of -10V to +10V. +5V, -15V and +15V taps are available but care should be taken to prevent shorts to ground. Limit supply drain current to 10mA.

It is not necessary to build the Pico Block circuit board. A programmed Pico will operate in the text mode with digital I/O. Analog inputs will be limited at 0 to 3.3V but DAC outputs require a low pass filter.

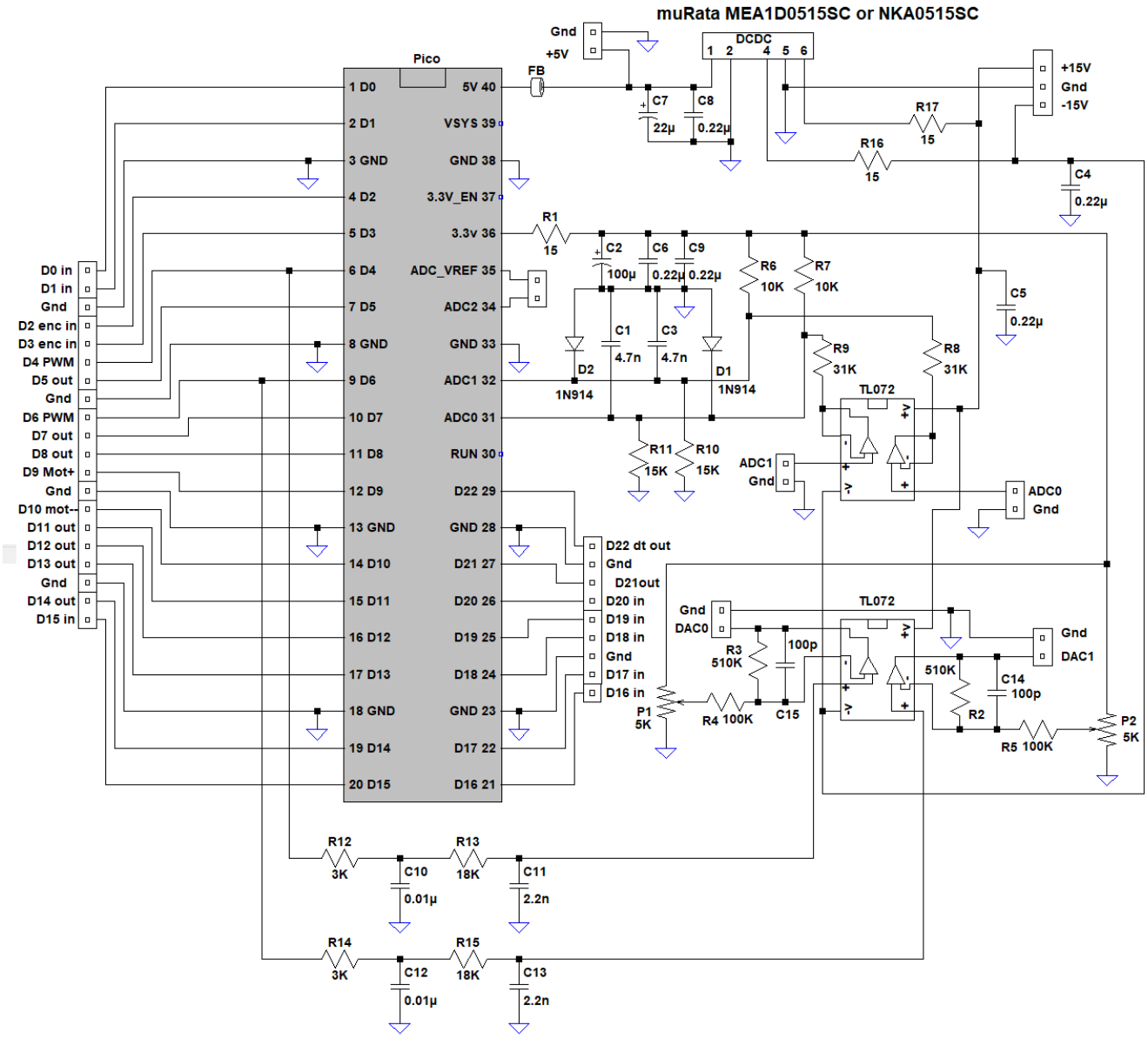
The program is written in C++ using the Arduino IDE and Earle Philhower's arduino-pico. The Pico is overclocked at 150MHz. If you have problems, consider using a standard clock rate. Want to get started fast, drop the **Pico_block_compiler_h.uf2** file into the Pico drive.

The Block program requires a serial terminal interface. Communication is through the USB cable at 460800 baud. A Windows compatible terminal, simple_crt.exe, written in Lazarus is supplied. It can send text program files, capture text data and plot up to five variables.

The Block compiler was designed as a minimal system:

- All I/O functions are preset on the device.
    - PWM limits the DAC outputs to about 1000Hz sinewave and 10-bits
- It is a text-based system with minimal line editing
- Text output slows the loop calculation time (about 1ms per value)
    - Loops are timed to the dt step value (validate timing on the D21 digital output)
- Block sorting is required for block outputs to update before using as inputs
- All inputs and outputs are variables.
    - Variables can be assigned a value using the set command
    - 100 is the maximum number of variables, including system variables
- Programs are limited to 50 block functions

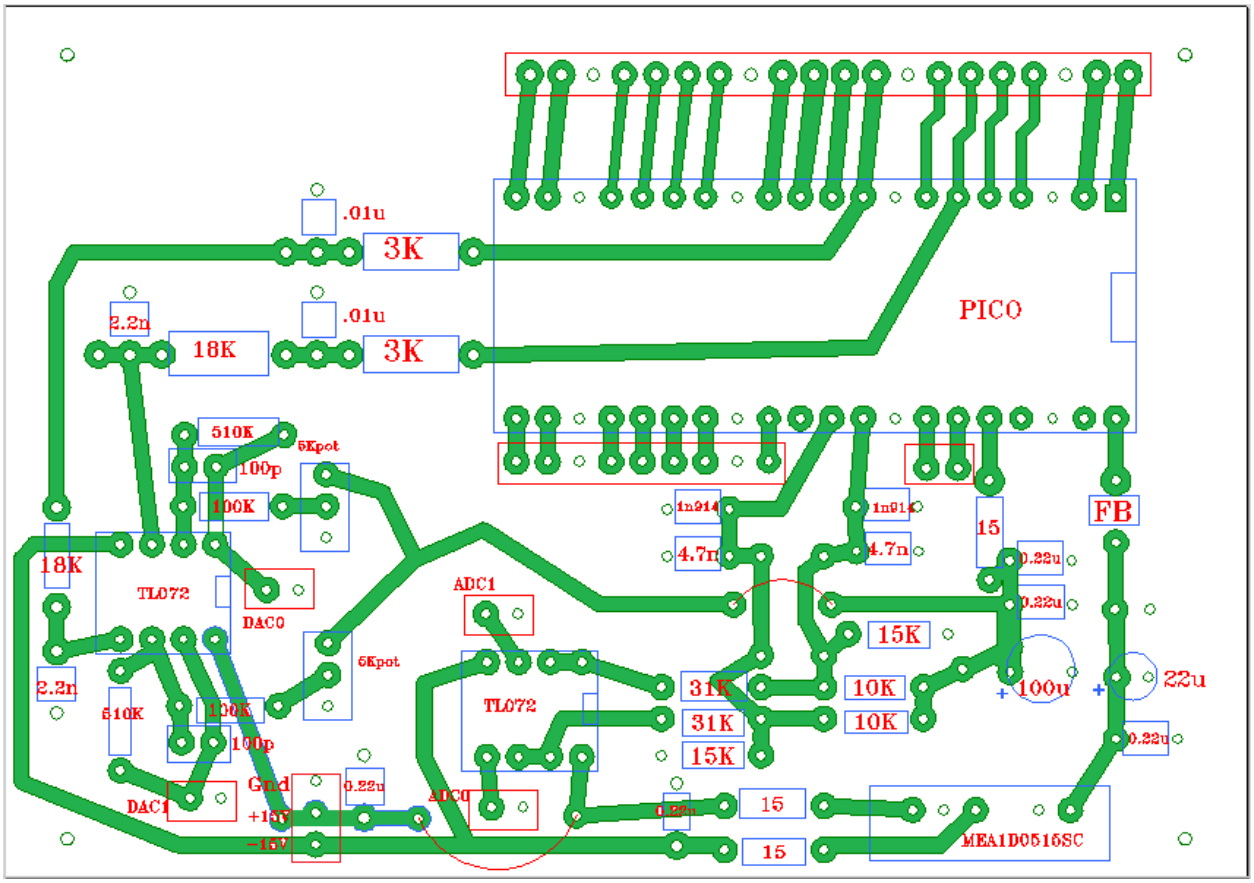
Pico Block Circuit Board



Notes:

- Use 1% metal film 1/4W Resistors
  - 1/4W carbon film Resistor Ok for R12-R15, R1, R16 and R17
- P1 and P2 5K potentiometer are 20 turn trimmers, vertical board mount
  - Adjust for DAC zero output
- C2, C7 100uF, 22uF low ESR capacitors
- C10, C11, C12, C13, C1, C3 are mylar film capacitors
  - Remaining are ceramic chip capacitors
- Shorting +15V or-15V to ground can damage the DC to DC converter
  - Limit external current draw to 10mA
- Applying less than zero volts or more than 3.3V to the Pico I/O pins can result in damage to the device
  - The voltage from DAC outputs, which can range from -14V to +14V, can damage I/O pins
- ADC0 and ADC1 are rated for -10V to +10V inputs
  - Ground when not in use

Circuit Board Component Placement:



An X-ray top view of the milled circuit board. The board is 100mm by 70 mm.

Use CNC files: **Drill_block.gcode** and **Profile_block.gcode**

The drill size is 0.7mm. The four holes at the corners can be hand drilled to 0.110” for mounting.

The red outlines indicated female 0.1” spaced socket strips. The Pico processor has male socket strip pins soldered to the card and the board uses female socket strips.

Machine tooled 8-pin dip sockets are recommended for the TL072 op amps.

Block Programming Text Entry Format:

Command<CR>
Block<SP> Variable<SP> Variable<SP> Variable<CR>

- All commands and block functions are three letter lower case names.
- Commands and block functions start on the first character of the line.
- The number of Input and Output variables varies for each block type.
- Variables are one to three characters separated by a space.
- A space <SP> is used as the delimiter.
- Variables names are assigned when a block function is entered or when a variable value is set.

Commands:

clr	Clear the program code and all variables
cal	Set the zero calibration forADC0 and ADC1, inputs must be grounded
zro	Zero the incremental encoder count
run	Run the block program, use <Esc> to break execution
lst	List the block program
var	List the block variables and their current values
hlp	List information on block functions, system variables and commands
srt	Sort the block program so all outputs are calculated before they are needed as inputs
lin line#	Set the program edit line pointer to line#. 0 <= line# <= 49
set variable value	Set variable = value
<Esc> or !	Stop program loop execution and return to the text editor. No <CR> needed.

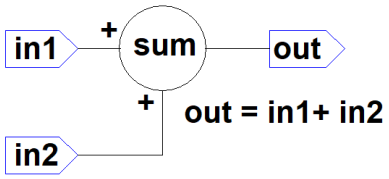
Built in Variables **t, dt, max, dec, ‘ ‘**:

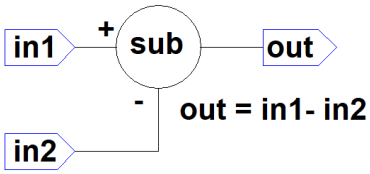
- **t** is time.
- **dt** is the step time.
- **max** is the maximum time.
- **dec** is the decimation number for printing.
  - Printing occurs at intervals of dec * dt.
- **fmt** formats number of places after decimal point
- **avg** is the number of ADC samples to average each reading
- **‘ ‘** is the blank variable (three spaces) used by the system.

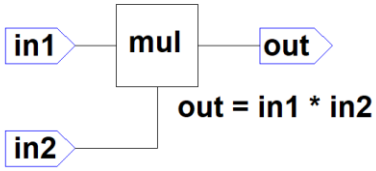
Summary of Block Functions:

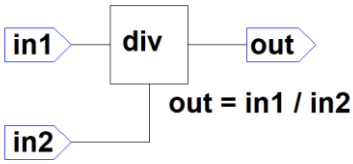
Block Code Input:
sum in1 in2 out
sub in1 in2 out
mul in1 in2 out
div in1 in2 out

Block Diagrams:

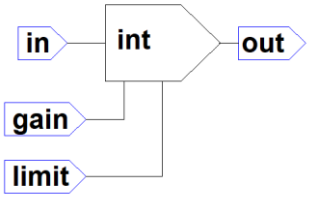


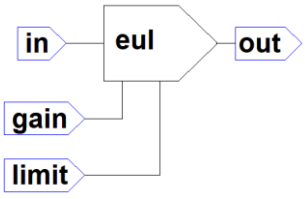




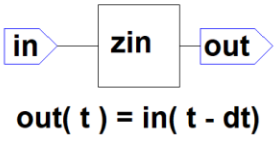


int in gain limit out 2nd order integration $out = gain * \int_0^t in\ dt$
eul in gain limit out 1st order integration limit == 0, no output limit
limit>0, -limit<=out<=limit
limit<0, 0.0<=out<=abs(limit)





zin in out delay one time step Z^{-1}



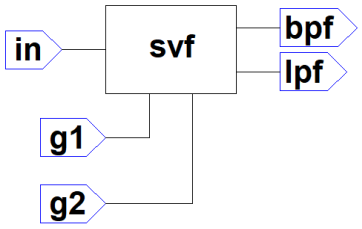
out(t) = in(t - dt)

svf in g1 g2 bpf lpf State Variable Filter

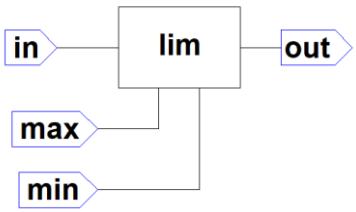
lpf = $\frac{g1g2}{S^2+ g1S+g1g2}$ bpf = $\frac{g1S}{S^2+ g1S+g1g2}$

lpf is a low pass filter and bpf is a band pass filter

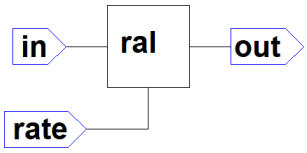
$\omega_n = \sqrt{g1g2}$ $\zeta = \frac{g1}{2\omega_n}$ $\omega_d = \omega_n (\sqrt{1 - \zeta^2})$



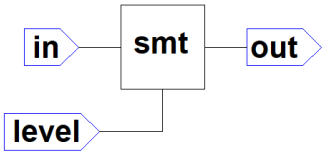
lim in max min out Limiter if in>max then out = max
if in<min then out = min
else out = in



ral in rate out **out** tracks input **in** until abs(in(t)-in(t-dt))/dt > **rate**
then the output change is limited to rate (e.g. volts per second)



smt in level out Schmitt trigger hysteresis
at t = 0, **out** must be set to an initial non-zero positive value
if **out**>0 then when **in**>**level**, +out --> -out
if **out**<0 then when **in**<-**level** -out --> +out



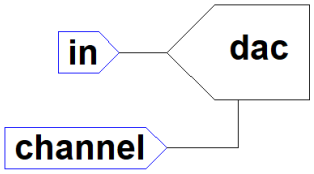
adc port out

0 <= **port** <= 2

out = -10.0 to 10.0V; **port** = 0,1 **out** = 0.0 to 3.3V; **port** = 2

dac in channel

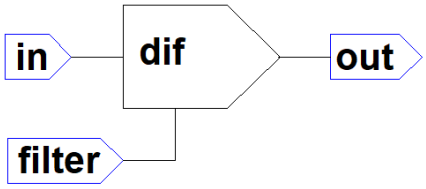
in = -10.0 to 10.0V; **channel** = 0, 1 for filtered PWM out on pins 4, 6



dif in filter out

Differentiator with a low pass filter cutoff

$out = \frac{\partial in}{\partial t} \cdot \frac{a}{S+a}$ where 0.0 < **filter** < 1.0



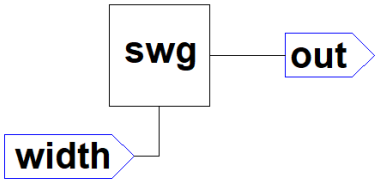
swg width out

Square Wave Generator

$frequency = \frac{1}{2*\boldsymbol{width}} Hz$, Duty cycle = 50%

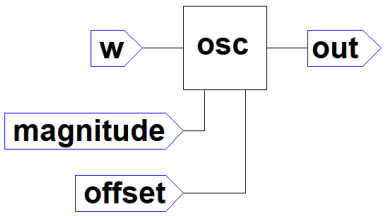
At t = 0, set **out** to positive peak value

out = ±peak value



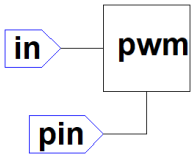
osc ω offset magnitude out

Sine wave oscillator
ω = 2*π*f in radians per second
out = **magnitude** * sine(**ω***t) + **offset**



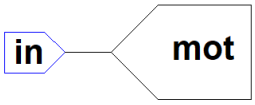
pwm in pin

Pulse width modulation
pin = 4 or **pin** = 6: 10-bit resolution at 146,484Hz
pin= 9 or **pin** = 10: 12-bit resolution at 2,289Hz
0.0 <= **in** <= 1.0



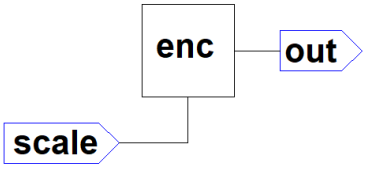
mot in

H-Bridge Motor Control with pulse width modulation 12-bit resolution
pin-9 is forward and pin-10 is reverse
-1.0 <= **in** <= 1.0, PWM duty cycle is 0 to 4095 at 2,289Hz



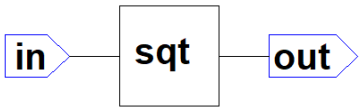
enc scale out

Incremental Encoder (inputs are pins 2 and 3)
out = (count) * **scale**
if **scale** = 0, count is reset to zero



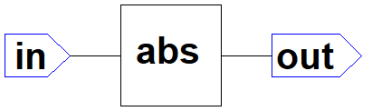
sqt in out

Square Root
for in >= 0, out = sqrt(in)
for in < 0, out = -sqrt(abs(in))



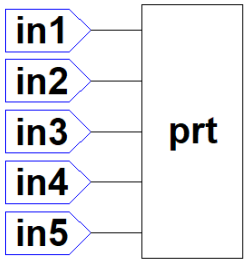
abs in out

Absolute value
out = abs(in)



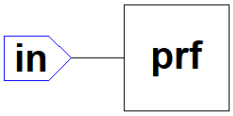
prt in1 in2 … in5

Print the value of the input variable
One to five variables are allowed
The output format is fixed to N-places after the decimal per the format variable **fmt**
Each variable prints with a delay of 1ms. So, five variables take 5ms.



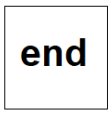
prf in

Print fast the value of the input variable
The output format is fixed to N-places after the decimal per the format variable **fmt**
The data is sent to the serial output without delay
Use with decimation to print without delays in high speed loops



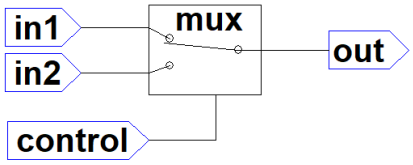
end

If **max** > 0, t=t+dt with a dt loop time
If **max** < 0, t=t+dt running with the minimal loop time (calculation time)
If **t** <= abs(**max**) loop back to the first block else end block execution



mux in1 in2 con out

if **control** <=0 **out** = **in1** else **out** = **in2**

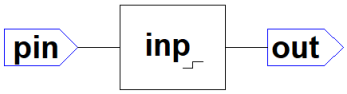


rst val var

Resets variable **var** to value **val**

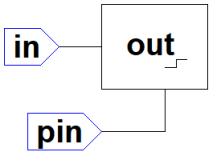
inp pin out

Read the digital input on I/O **pin**
out = 0 or 1



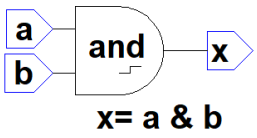
out in pin

If **in** <= 0 set I/O **pin** to a logic low else set I/O **pin** to a logic high



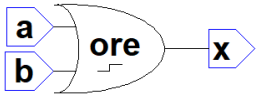
and a b x

Logical AND function
x is 0 or 1
if a <=0 it is a logic low else a logic high
if b <=0 it is a logic low else a logic high



ore a b x

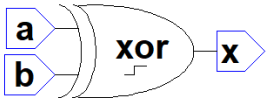
Logical OR function
x is 0 or 1
if a <=0 it is a logic low else a logic high
if b <=0 it is a logic low else a logic high



x= a | b

xor a b x

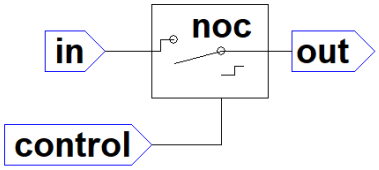
Logical Exclusive OR function
x is 0 or 1
if a <=0 it is a logic low else a logic high
if b <=0 it is a logic low else a logic high



x = a ^ b

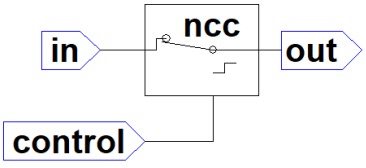
noc in con out

Normally open contact
If **con** <= 0 then **out** = 0
If **con** > 0 and **in** <= 0 then **out** = 0
If **con** > 0 and **in** > 0 then **out** = 1



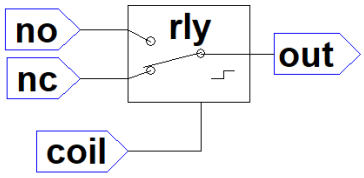
ncc in con out

Normally closed contact
If **con** > 0 then **out** = 0
If **con** <= 0 and **in** <= 0 then **out** = 0
If **con** <= 0 and **in** > 0 then **out** = 1



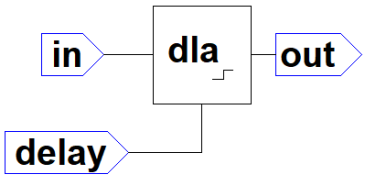
rly no nc coi out

Relay contacts
If **coi** <= 0 then **out** = **no** else out = **nc**
When the input (**no** or **nc**) <= 0 then the input value = 0
When the input (**no** or **nc**) > 0 then the input value = 1



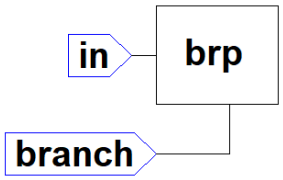
dla in del out

Logic Delay
If **in** <= 0 then **out** = **0**
If **in** > 0 then **out** = **1** after **del** seconds



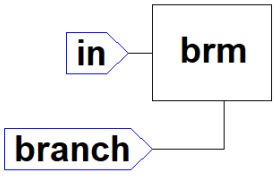
brp in bra

Branch if **in** >= 0, program counter = program counter + **bra**
else proceed to next block (program counter + 1)

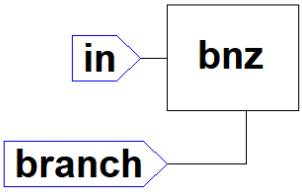


brm in bra

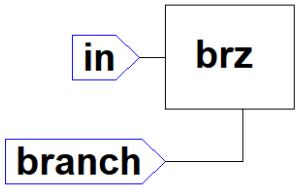
Branch if **in** < 0, program counter = program counter + **bra**
else proceed to next block (program counter + 1)



bnz in bra Branch if **in** != 0, program counter = program counter + **bra**
else proceed to next block (program counter + 1)



brz in bra Branch if **in** == 0, program counter = program counter + **bra**
else proceed to next block (program counter + 1)



Block Execution Order:

Each block function is executed in sequence until the end block is reached. Outputs for some block functions are updated as the block is executed. Others, the output is updated when the end block is reached.

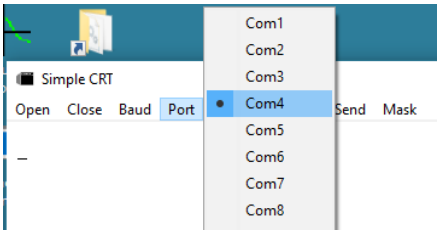
Simple numerical operations contain no time dependent data. These are updated when the block is executed. Blocks do require the that input variables are updated prior to execution. Blocks should be sequenced such that outputs used for inputs to other blocks are calculated first. If you are uncertain to the order, use the **srt** command.

Temporal calculations such as integration, differentiation, delays, switches and relays will update at the end of each loop. These outputs remain constant for any input through each loop. The method does have a slight drawback since the system is delayed by one cycle. This results in an added phase shift of the outputs.

Branching instructions require sections of code to remain as written. Thus, the **srt** command is disabled when these blocks are present.

Programming:

Plug the USB cable into the computer and into the Pico processor. Run Simple_CRT.exe to communicate with the Pico. In simple_crt, set the **Port** for your board and click open. Type in your block program or use menu **Send, File** to upload a block text file program. Type run<CR> to run the program.



Select your com port first

Then, open the com port

An example entry and edit session. The prompt shows the current program line number:

0: int one g lim out

1: prt out

2: end

3: set one 1

3: set g 1

3: set lim 0

3: lst

int one g   lim out
prt out
end

3: var

set dt 0.010000
set t   0.000000
set max 1.000000
set dec 1.000000
set fmt 6.000000
set avg 1.000000

```
set one 1.000000
set g   1.000000
set lim 0.000000
set out 0.000000

3: set dt 0.1

3: run
0.000000
0.050000
0.150000
0.250000
0.350000
0.450000
0.550000
0.650000
0.750000
0.850000
 ***** T = MAX *****
```

Reset time to zero, reset the integrator and edit line 1 to print t plus the output:

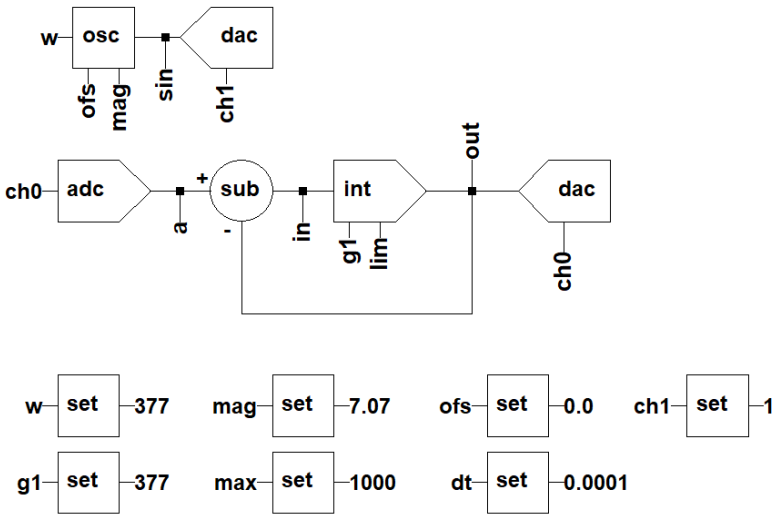
```
3: set t 0

3: lin 1
prt out

1: prt t out

2: lin 3
nop

3: lst

int one g   lim out
prt t   out
end

3: run
0.000000 0.950000
0.100000 1.050000
0.200000 1.150000
0.300000 1.250000
0.400000 1.350000
0.500000 1.450000
0.600000 1.550000
0.700000 1.650000
0.800000 1.750000
0.900000 1.850000
 ***** T = MAX *****

3:
```

Low Pass Filter Example:

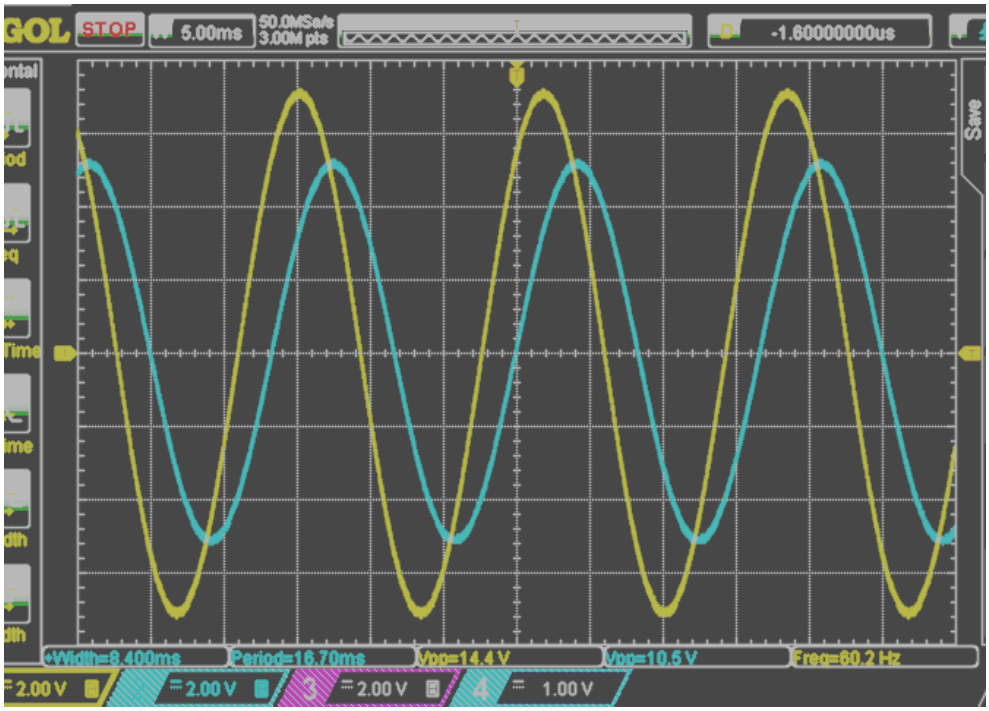
A single pole low pass filter with an ADC input and a DAC output. The ADC input is scaled for -10 to +10V and the DAC output is scaled for a -10 to +10V output. The integrator gain g1 sets the break frequency for ω. The frequency in Hz is 60. Connect a wire from the DAC1 output to the ADC0 input pin.

$$\text{Low Pass Filter } \frac{out}{in} = \frac{g1}{s+g1}$$

Sinewave out with input Low Pass Filter



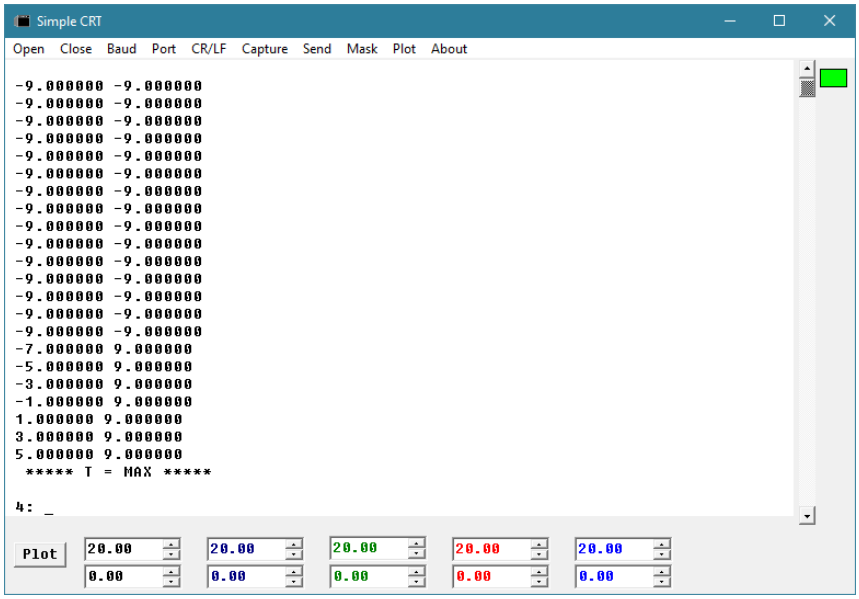

```
clr
osc w ofs mag sin
adc ch0 a
sub a out in
eul in g1 lim out
dac out ch0
dac sin ch1
end
set ch0 0
set ch1 1
set g1 377
set lim 10
set dt 0.0001
set max 1000
set w 377
set mag 7.07
set ofs 0
```



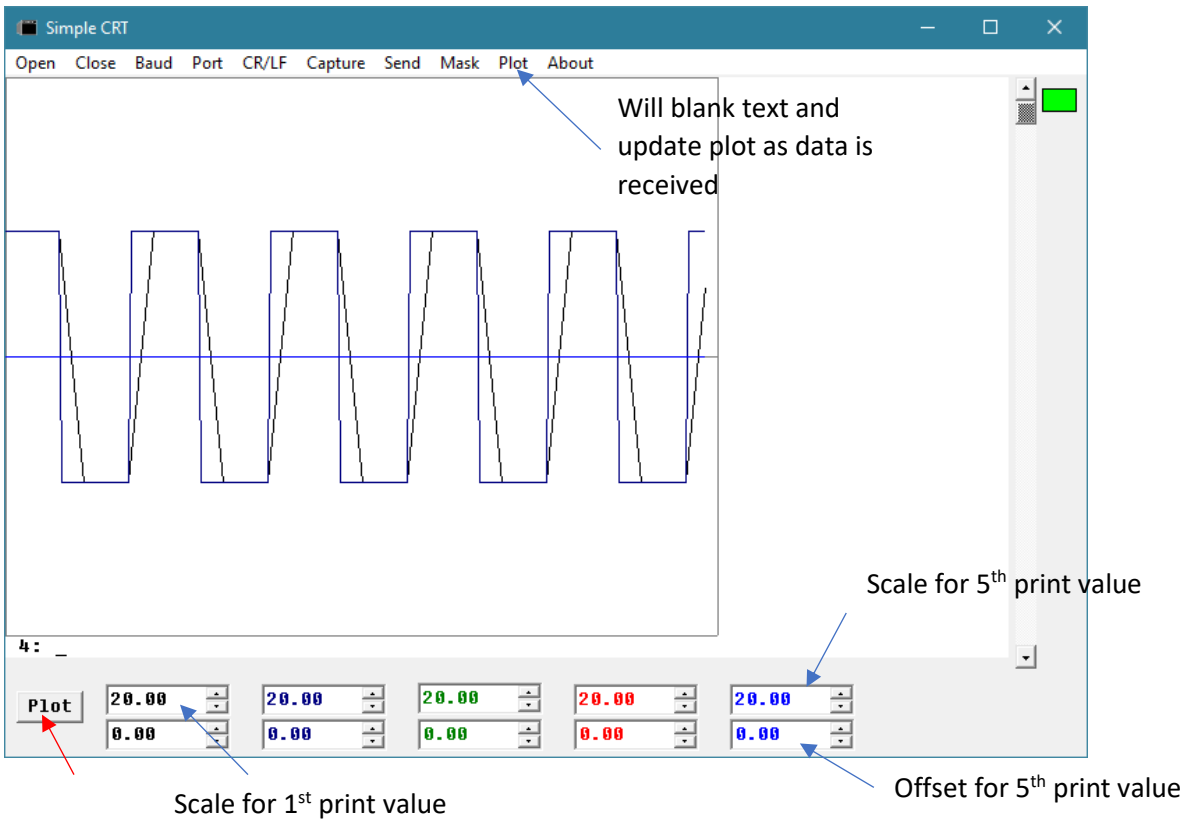
Oscilloscope capture of DAC0 and DAC1 outputs.

A square wave oscillator with output rate limiting:

```
clr
set lim 200
set out 9
set max 2.56
set dt 0.01
set tw 0.25
swg tw out
ral out lim a
prt a out
end
```



The time step dt and max are set to print 256 values. Clicking Plot scans the last 256 lines stored in Simple_crt and plots the values.

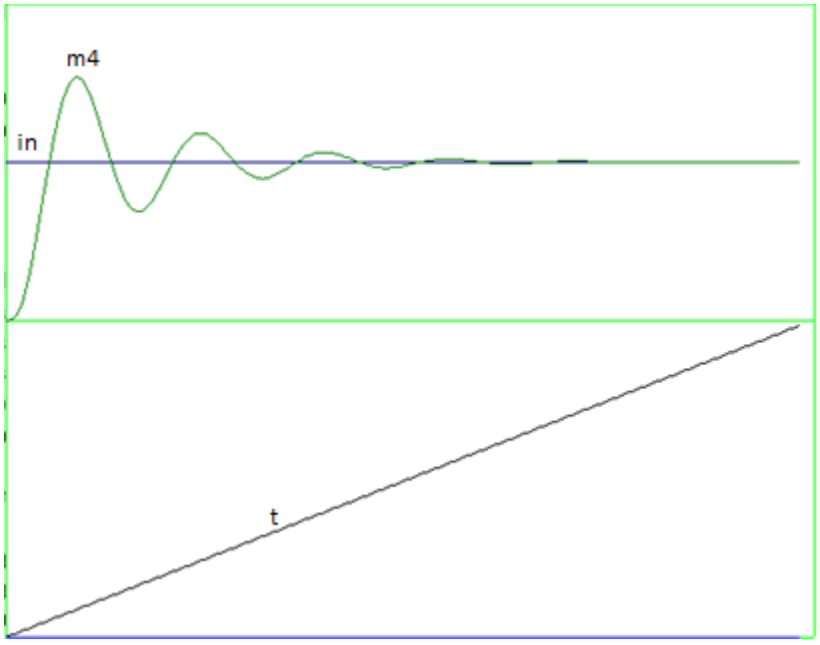


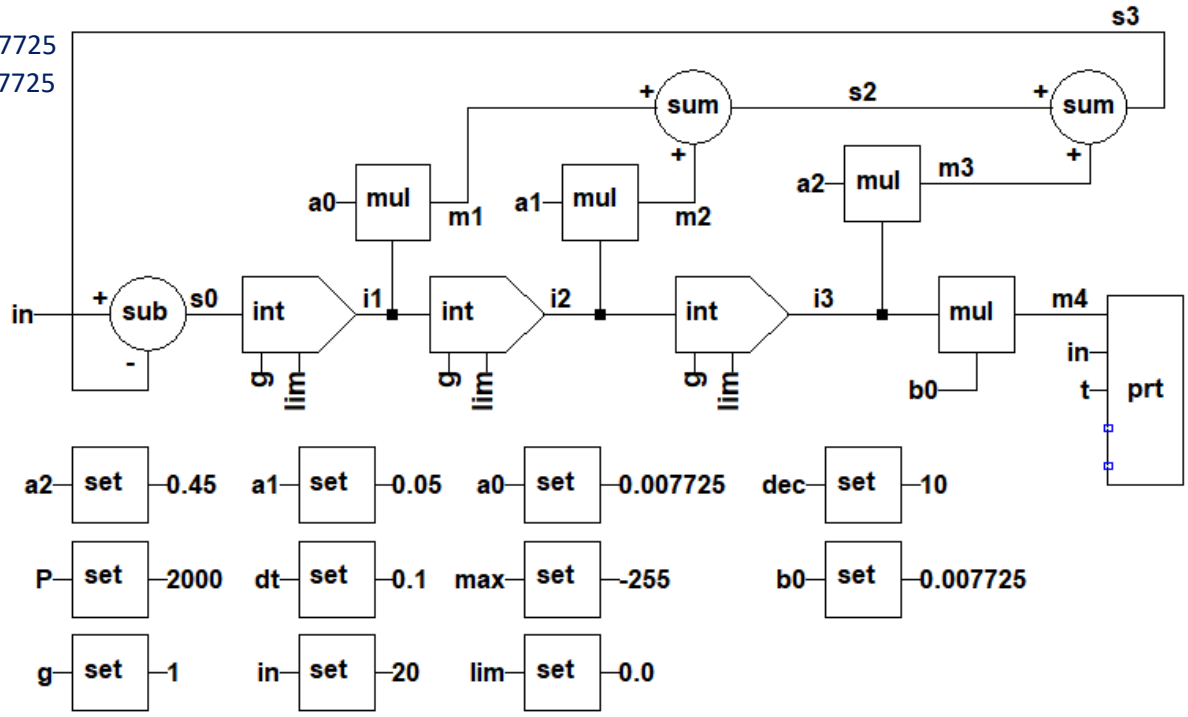
A simulation of a third order equation:

Setting **max** negative runs the simulation at maximum speed instead of real-time. Setting **dec** to 10 sets printing to 10*dt = every 1 second.

$$H(s) = \frac{0.007725}{s^3 + 0.45 \cdot s^2 + 0.05 \cdot s + 0.007725}$$

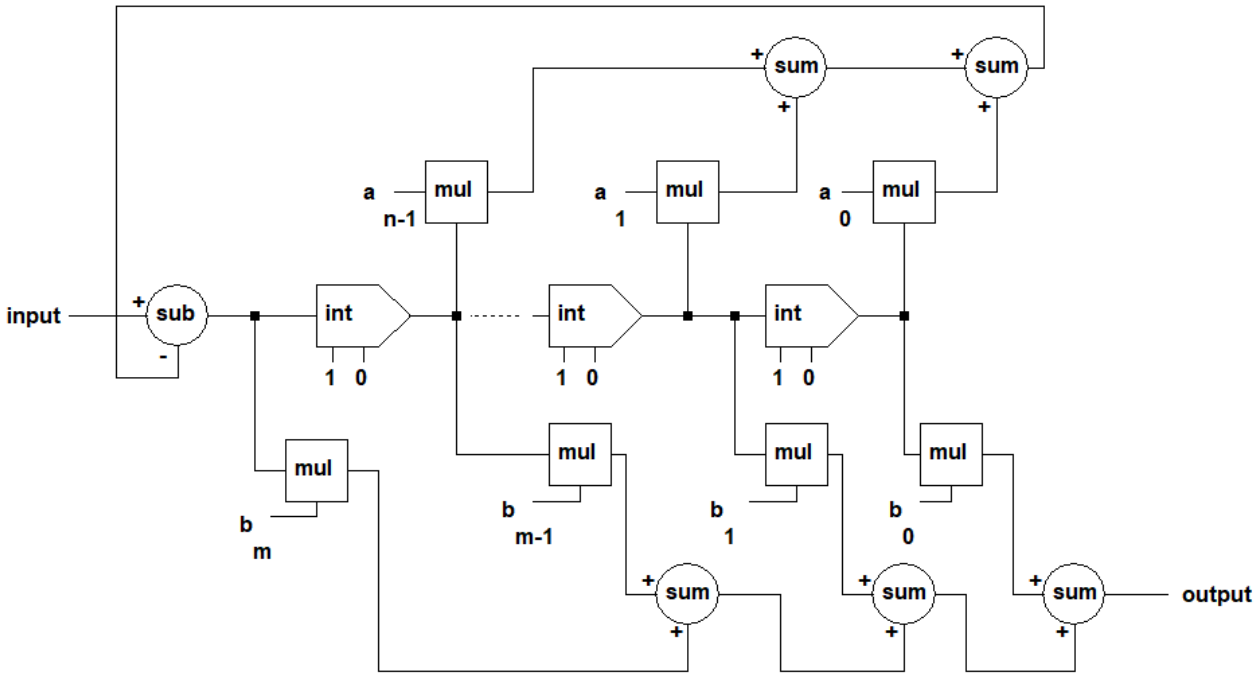
```
clr
mul a2 i1 m1
mul a1 i2 m2
mul a0 i3 m3
sum m1 m2 s2
sum s2 m3 s3
sub in s3 s0
mul i3 b0 m4
int s0 g lim i1
int i1 g lim i2
int i2 g lim i3
prt m4 in t
end
set in 10
set dec 10
set dt 0.1
set max -252
set g 1
set b0 0.007725
set a0 0.007725
set a1 0.05
set a2 0.45
```





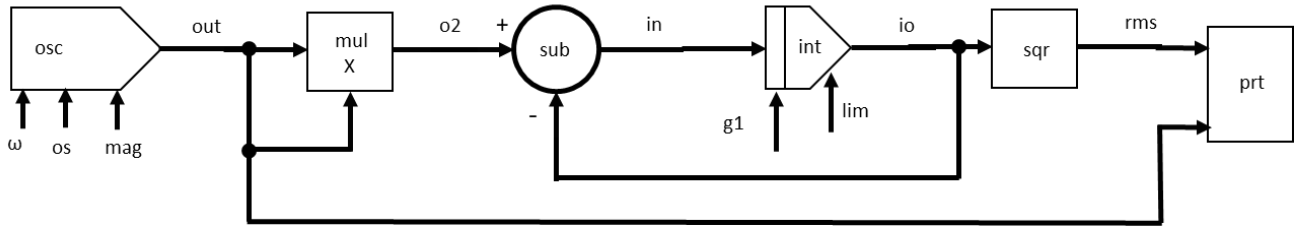
The general form for simulating state equations:

$$\frac{output}{input} = \frac{b_m S^m + b_{m-1} S^{m-1} + b_{m-2} S^{m-2} + \cdots + b_0}{S^n + a_{n-1} S^{n-1} + a_{n-2} S^{n-2} + \cdots + a_0}$$

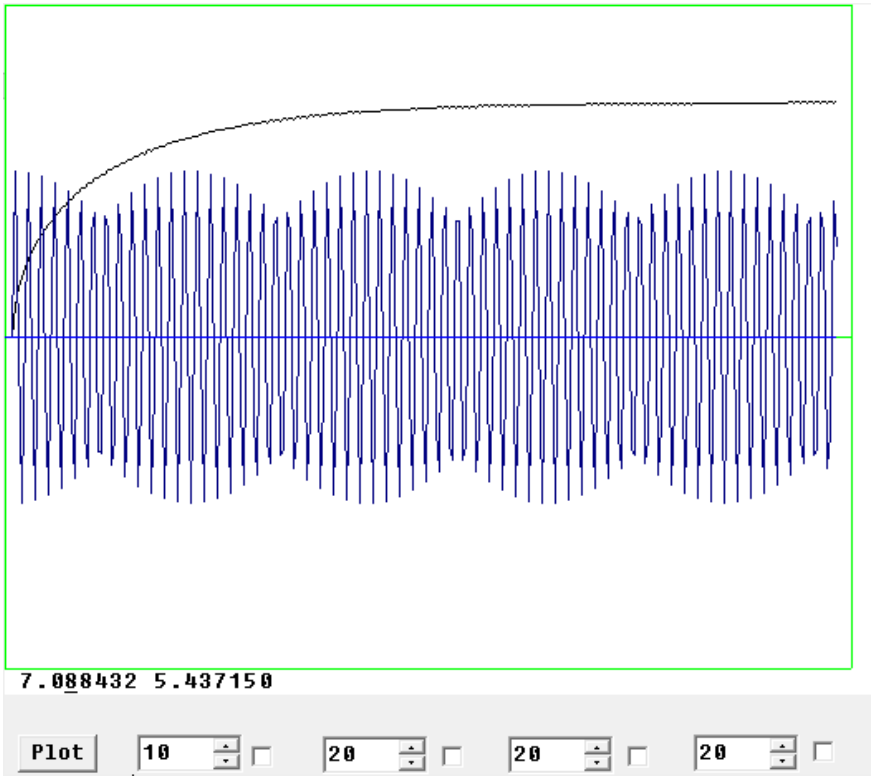


A simulation to calculate the RMS value of a sine wave with and amplitude of 10.0:

RMS Converter out $= \sqrt{\int in^2 dt}$




```
clr
set dt 0.001
set dec 8
set max 2
set w 200
set os 0
set mag 10
set g1 3
osc w os mag out
mul out out o2
sub o2 io in
int in g1 lim io
sqt io rms
prt rms out
end
```

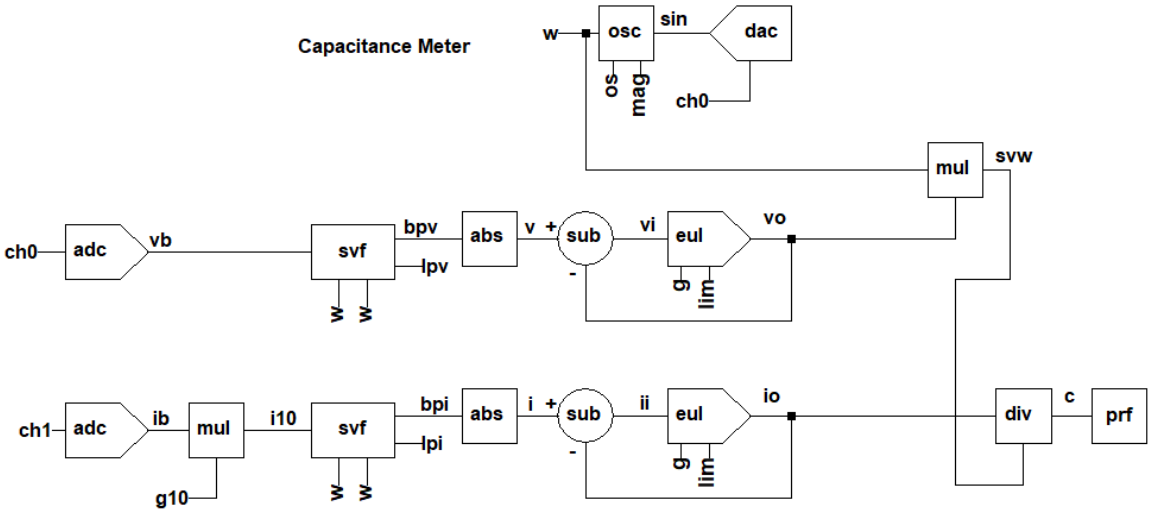


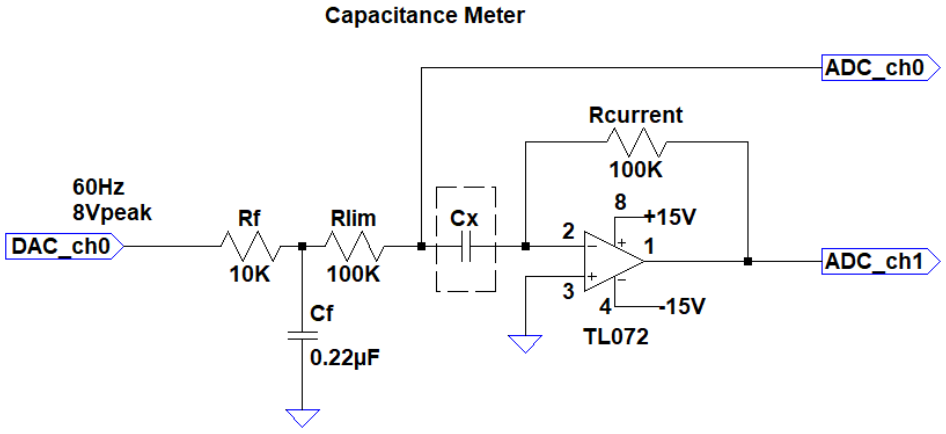
The last averaged value is 7.088432. The 31.8Hz sine wave is aliased in the plot.

Capacitance Meter:

Capacitance is measured by sensing voltage and current. C = I/(ω*V). Two state variable band pass filters are used to remove the DC component of each signal. The filter also reduces noise. The rectified signal (absolute value) is low pass filtered to yield a low noise magnitude of the capacitor voltage and the capacitor current. The measured capacitance value is printed every two seconds.

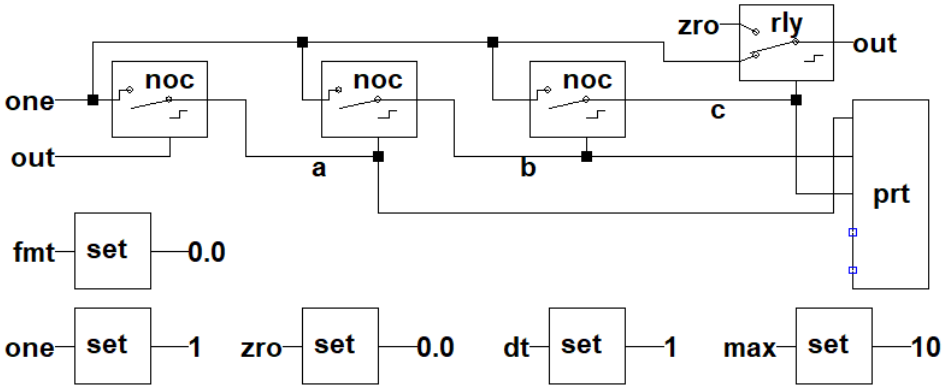
```
clr
osc w os mag sin
dac sin ch0
adc ch0 vb
adc ch1 ib
mul ib g10 i10
svf vb w w bpv lpv
svf i10 w w bpi lpi
abs bpv v
abs bpi i
sub v vo vi
sub i io ii
eul vi g lim vo
eul ii g lim io
mul w vo svw
div io svw c
prf c
end
set g10 10
set ch0 0
set ch1 1
set w 377
set os 0
set mag 8
set g 1
set dt 0.0002
set max 1000
set dec 10000
```





Note: The circuit is on a separate breadboard. Plus and minus 15V are available on the Pico block circuit card.

Switch operation showing circular loop timing:




```
clr
set one 1
set out 1
set dt 1
set max 16
set fmt 0
noc one out a
noc one a b
noc one b c
rly z one c out
prt a b c out
end
```

Simple CRT

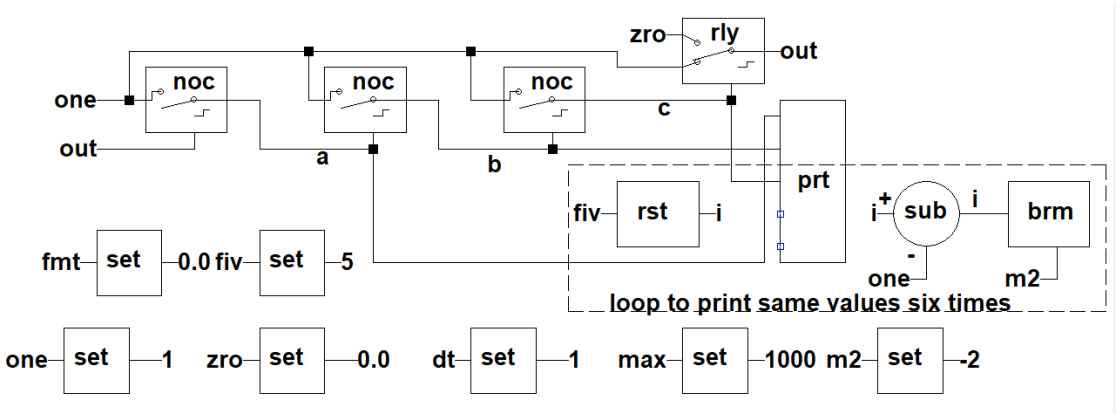
Open Close Baud Port CR/LF Capture Send Mask Plot About

```
4: prt a b c out

5: end

6: run
0  0  0  1
1  0  0  1
1  1  0  1
1  1  1  1
1  1  1  0
0  1  1  0
0  0  1  0
0  0  0  0
0  0  0  1
1  0  0  1
1  1  0  1
1  1  1  1
1  1  1  0
0  1  1  0
0  0  1  0
0  0  0  0
 ***** T = MAX *****

6: _
```

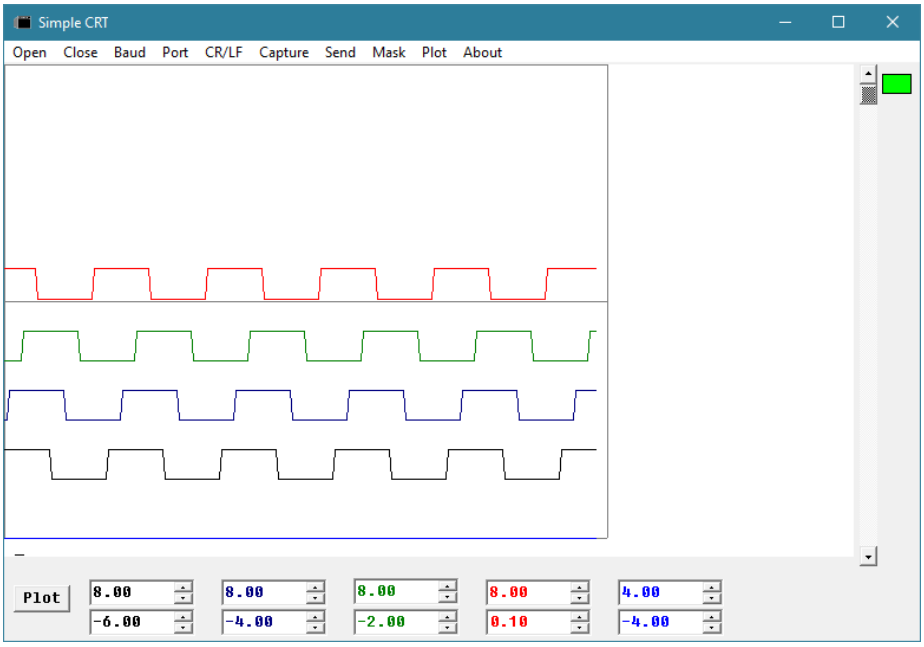
Plot

20.00	20.00	20.00	20.00	20.00
0.00	0.00	0.00	0.00	0.00

To plot values that change each cycle, a branch loop is added to repeat the printing of the values. This stretches the plot for better visibility.

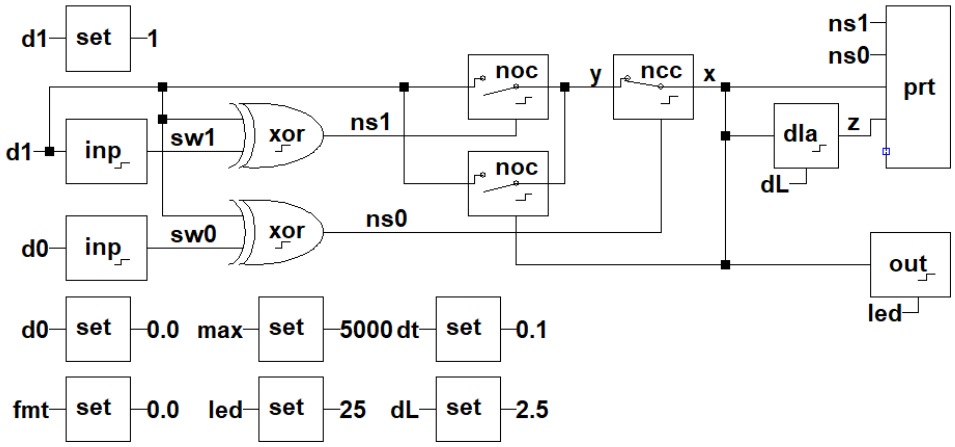



```
clr
set one 1
set m2 -2
set fiv 5
set out 1
set dt 1
set max 1000
set fmt 0
noc one out a
noc one a b
noc one b c
rly z one c out
rst fiv i
prt a b c out
sub i one i
brp i m2
end
```

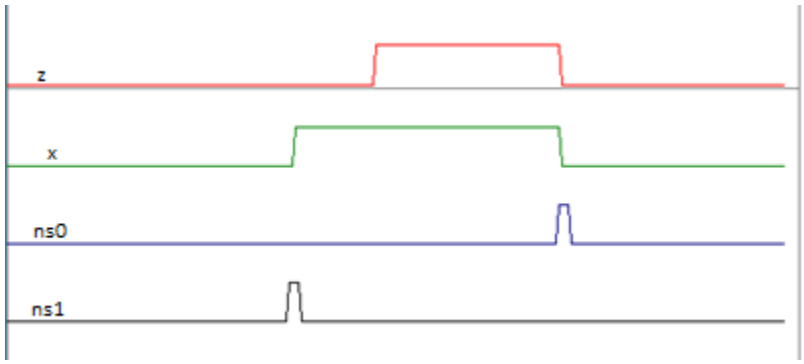


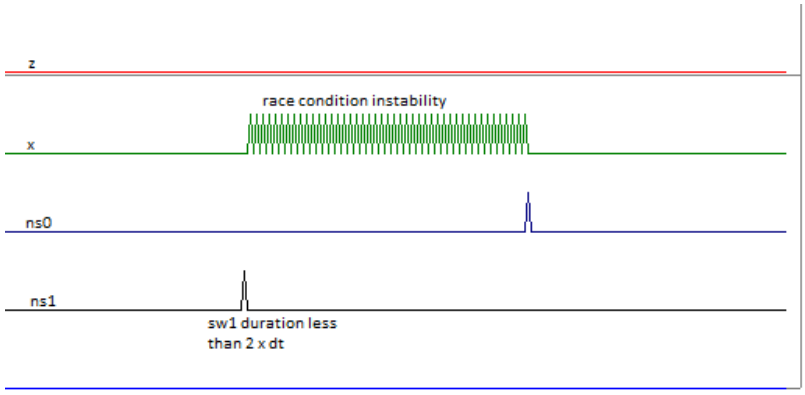
Normally open and normally closed contacts form a latching relay for run/stop operation:

The external push buttons on D0 and D1 ground the input pins when pressed. Use Exclusive OR blocks to invert the pulled up digital inputs. The delayed output z might be used to kick out a motor start capacitor relay.




```
clr
set d1 1
set d0 0
set fmt 0
set max 5000
set led 25
set dt 0.1
set dL 2.5
inp d1 sw1
inp d0 sw0
xor d1 sw1 ns1
xor d1 sw0 ns0
noc d1 ns1 y
noc d1 x y
ncc y ns0 x
dla x dL z
out x led
prt ns1 ns0 x z
end
```

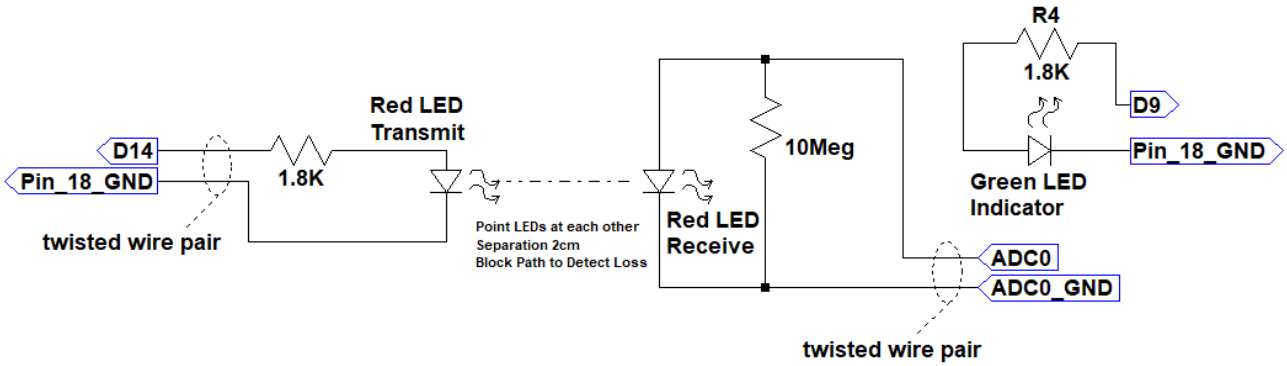




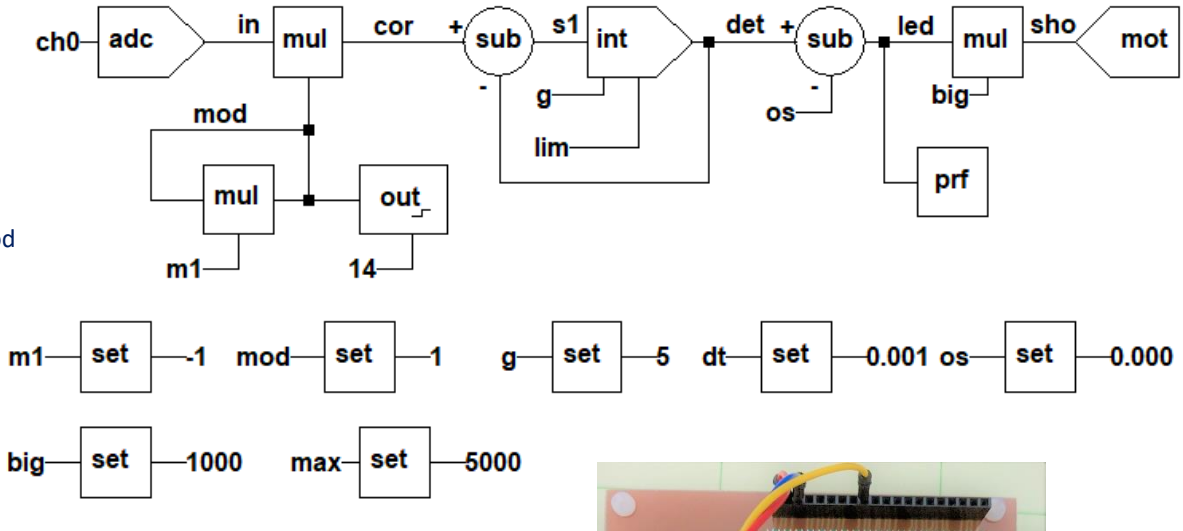
The start sw1 and stop sw0 signals must be pressed for at least two loops to prevent instability. Changing dt to 10ms or less prevents this condition since it is unlikely a button can be held for less than 10ms.

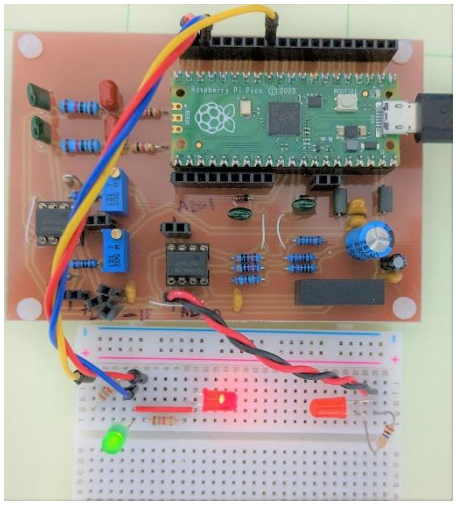
Correlated sampling below noise level signal detection:

An external LED is modulated at 500Hz. The receiver generates around 1mV of differential signal. The ADC0 input has a resolution of about 1mV when averaged 16 times but a noise level of 10mVpp. Averaging the modulated on/off difference yields a sub-mV detection level.

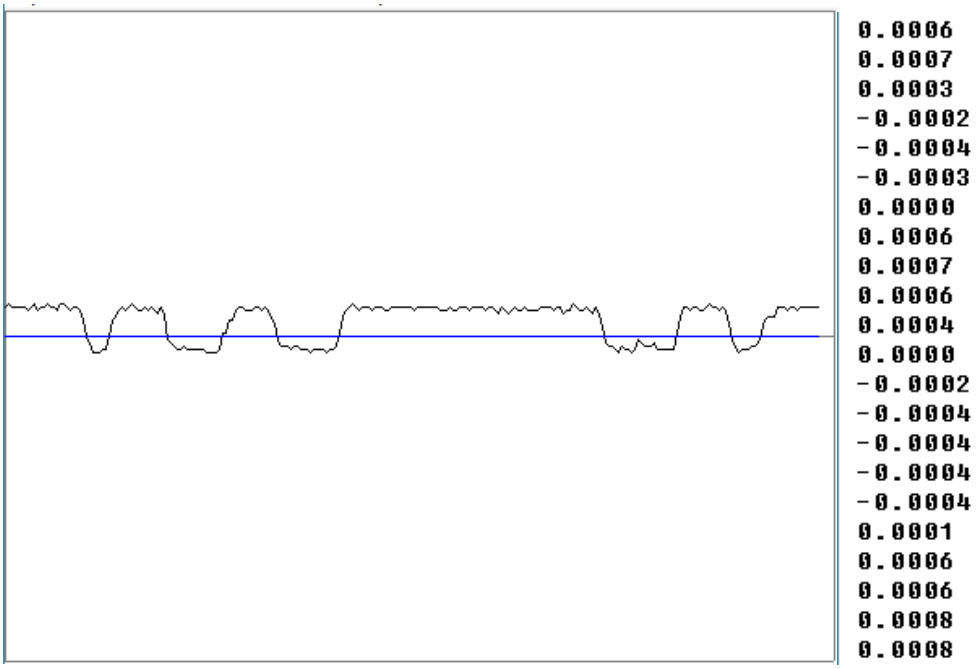



```
clr
adc ch0 in
mul in  mod cor
sub cor det s1
int s1  g   lim det
sub det os  led
mul led  big sho
mot sho
mul mod m1 mod
out mod 14
prf led
end
set dt  0.001
set t   0.0
set max 5000.0
set dec 100
set fmt 4
set avg 16
set ch0 0
set m1 -1
set mod 1.0
set g 5
set lim 0
set os  0.0005
set big 1000.0
set t1  0.001
set 14 14
```

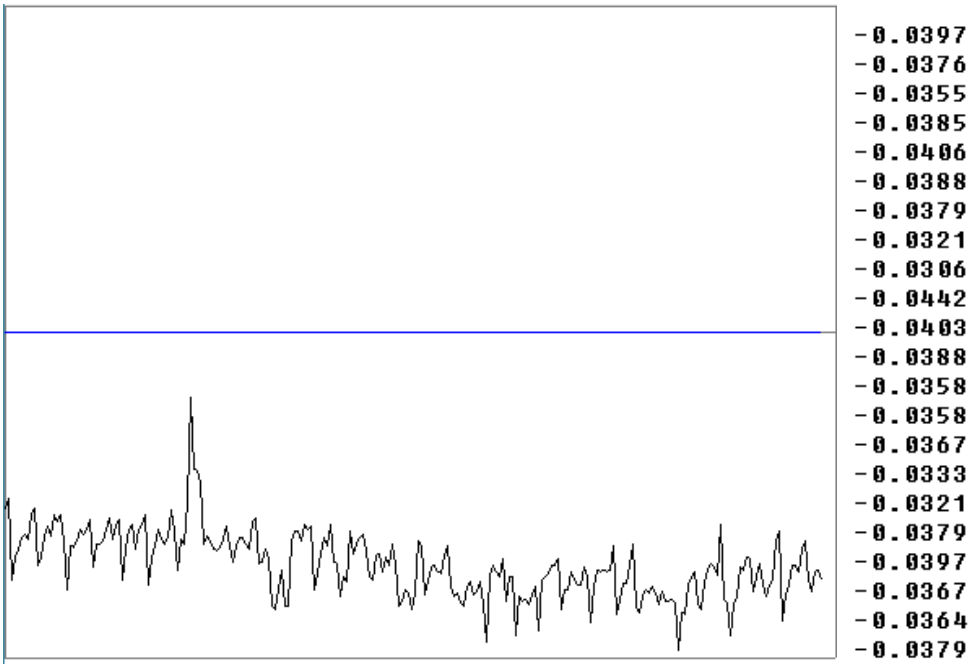




Note: set os for an unblocked signal of around 0.5mV



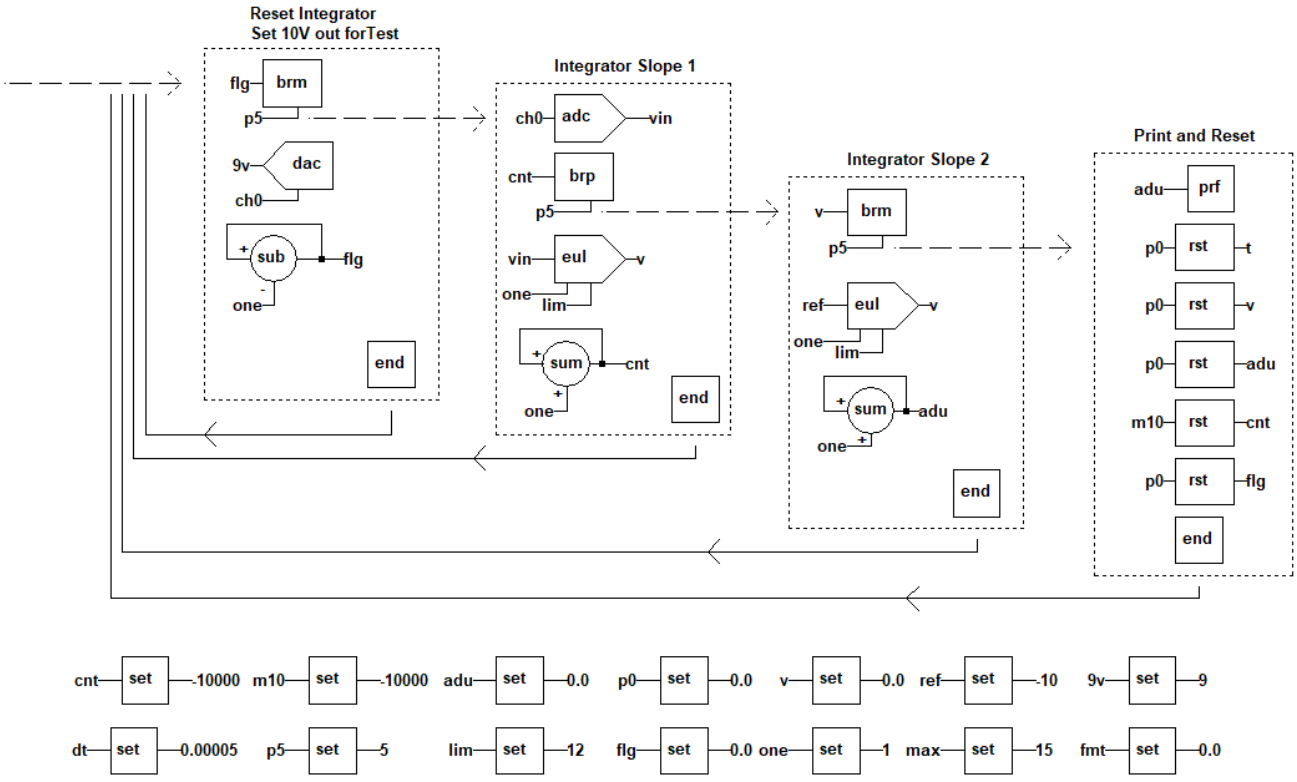
A plot of the blocked and unblocked light signal. Text at the right shows the magnitude of the signal.



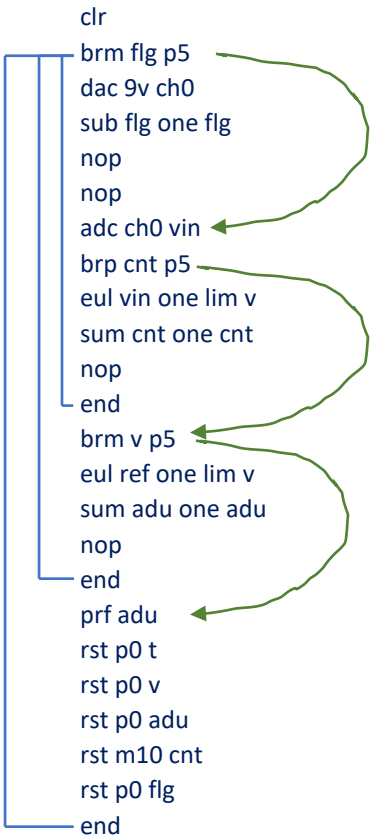
A plot of the blocked and unblocked ADC0 output signal. Text at the right shows the magnitude of the signal.

Simulation of a Dual Slope Integrating Analog to Digital Converter:

The input voltage is from ADC0 and is limited at 0 to 10V. Conditional branches are used for process loops that are skipped after each operation. Each sub-loop uses an end statement to update parameters at the end of each loop. The first loop is bypassed after counters and integrator reset. The second loop counts from -10000 to 0 as the integrator accumulates the average of the ADC0 voltage. Once a zero count is reached, this loop is bypassed, and the integrator is ramped with a negative slope from a fixed reference voltage. ADU counter is incremented each loop until the integrator reaches zero. The ADU count is the converted voltage. All loops are bypassed except the last loop, where ADU is printed, and a reset is performed.



Use DAC0 as the input voltage for ADC0. Set variable 9V to your test voltage or use a potentiometer to vary the fixed DAC0 voltage.

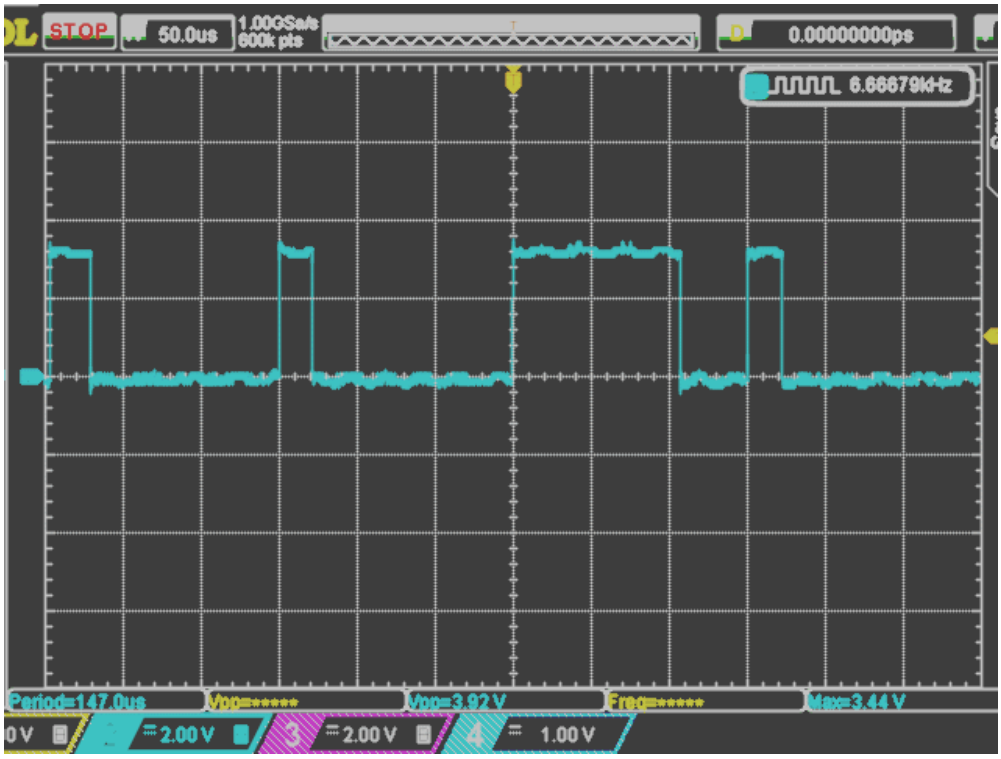



```
set cnt -10000
set m10 -10000
set adu 0
set p0 0
set v 0
set ref -10
set 9v 9
set p5 5
set one 1
set lim 12
set flg 0
set dt 0.00005
set fmt 0
set max 15
```

Checking Loop Timing on D22:

For correct loop timing, the block calculation time plus print time must be less than dt. This can be observed on an oscilloscope attached to the D22 digital output. The following program has dt set to 150us. This is adequate time for the two integration calculations (about 10us each) plus the fast print which takes 90us every five cycles (due to dec = 5). Not running at the correct real time rate does not affect the calculation. The step of dt is used. To run at full calculation speed, set max to a negative limit.

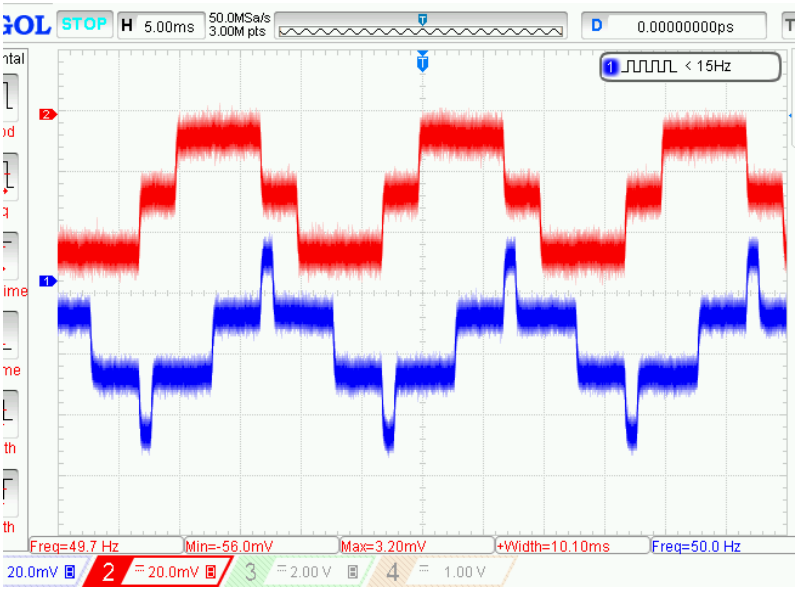
```
int a   w   lim b
int b   mw  lim a
prf a
end
set dt  0.00015
set t   0
set max 1000
set dec 5
set fmt 2
set avg 1
set a   -5.38
set w   50.0
set lim 10.0
set b   -8.51
set mw  -50.0
```



Analog Performance:

The PWM digital to analog converters are set for 1024 levels or 10-bits. The output range with ideal resistors is about ±10V. The step size is 19.3mV. The zero point is adjustable with the 5K potentiometer. Gain corrections can be made by pre-scaling the input value with a multiplier block.

The DAC PWM low pass filter has a -3dB cutoff around 2KHz. The DAC has a flat sinusoidal response (-1dB) out to 1KHz. The filter reduces PWM noise to under one bit (less than10mV). However, the output noise is limited by the 3.3V reference filter. Best results are achieved when the 100uF capacitor has a low ESR (0.2 ohms or better). The PWM filter equivalent step time constant is 128us with settling to 1% in 325us.



Low level signal outputs for both DAC channels. The noise is about 10mVpp which is half an LSB for the DAC range of 20V.

The ADC input range is -10V to +10V. There is about -58dB of cross talk between the ADC0 and ADC1 channels. This is due to the R/C filtering on the 3.3V reference.

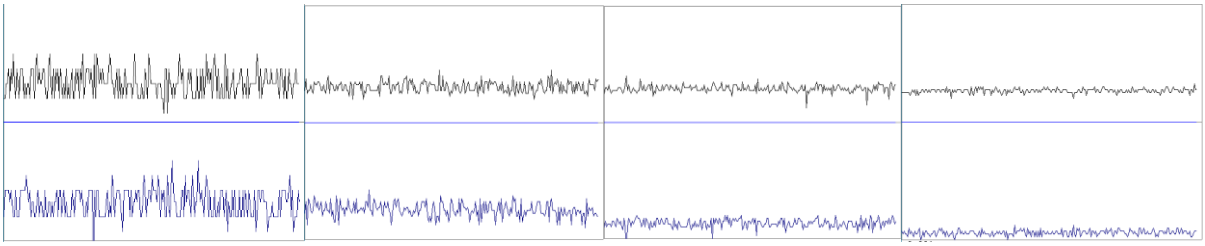
Keep unused inputs grounded. The op amp outputs can exceed ±10V. When this occurs, a small current (around 100uA) can be injected in substrate. The input will not be damaged but it can affect the ADC offset. You may note the opposing channel offset changes when one input is left floating. For negative saturation, it is around -100mV! If you drive the input to 14V, you could see as much as +800mV on the other channel.

The offset and gain will also vary with the accuracy of the 3.3V supply and the resistor tolerances. The ideal 12-bit ADU value is converted to voltage with this calculation:

V = (ADU * 20.35/4095) – 10.0 = (ADU * 0.004969) – 10.0 (for inputs ADC0 and ADC1)

Gain can be corrected with an additional multiply block function or by modifying the Arduino sketch constant ADC_GAIN. Use the **cal** command to zero channel 0 and 1 ADCs. Ground both inputs then execute the **cal** command.

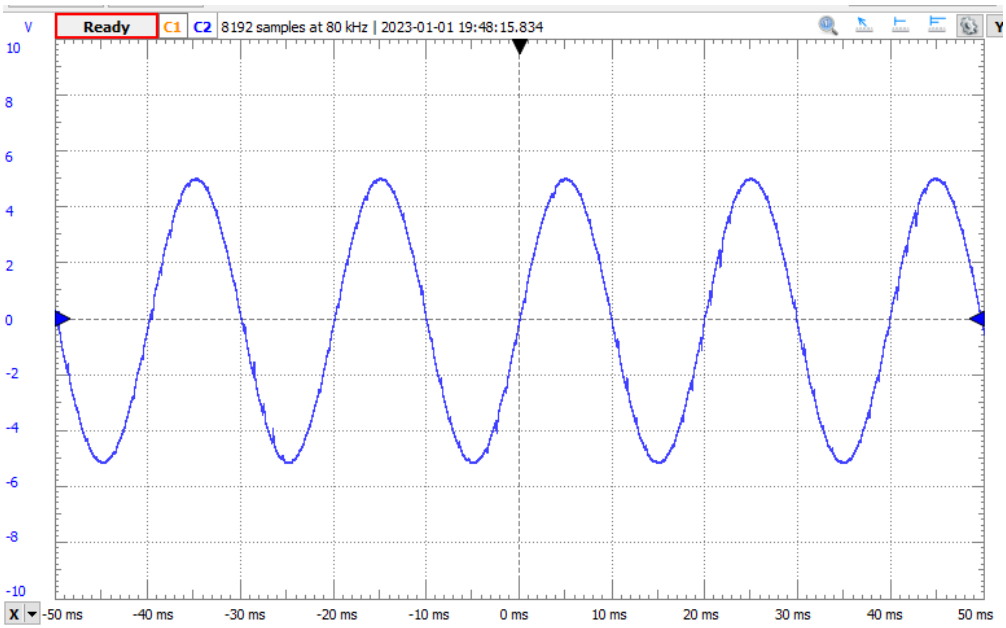
The **avg** variable sets the number of times the ADC is read then averaged. The ADC block will perform a single read in about 13.5uS (about 74KHz). Averaging increases the read time per sample: **avg** = 4 → 29us, **avg** = 16 → 92us, **avg** = 64 → 343us



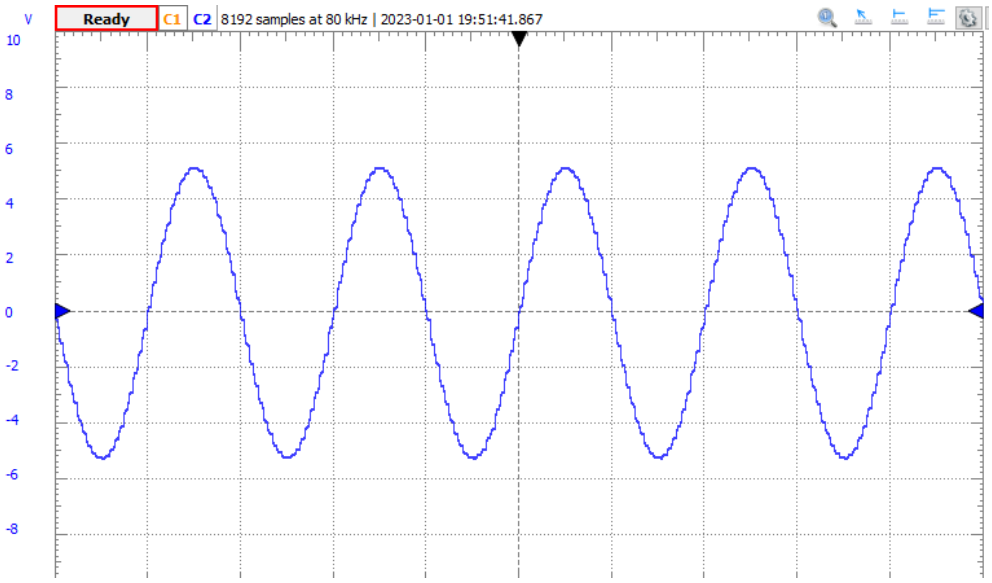
avg = 1 avg = 4 avg = 16 avg = 64

Averaging decreases the available sampling frequency but noise is reduced as shown above. The plot scale is 40mV top to bottom. Shown are ADC0 and ADC1 with grounded inputs.

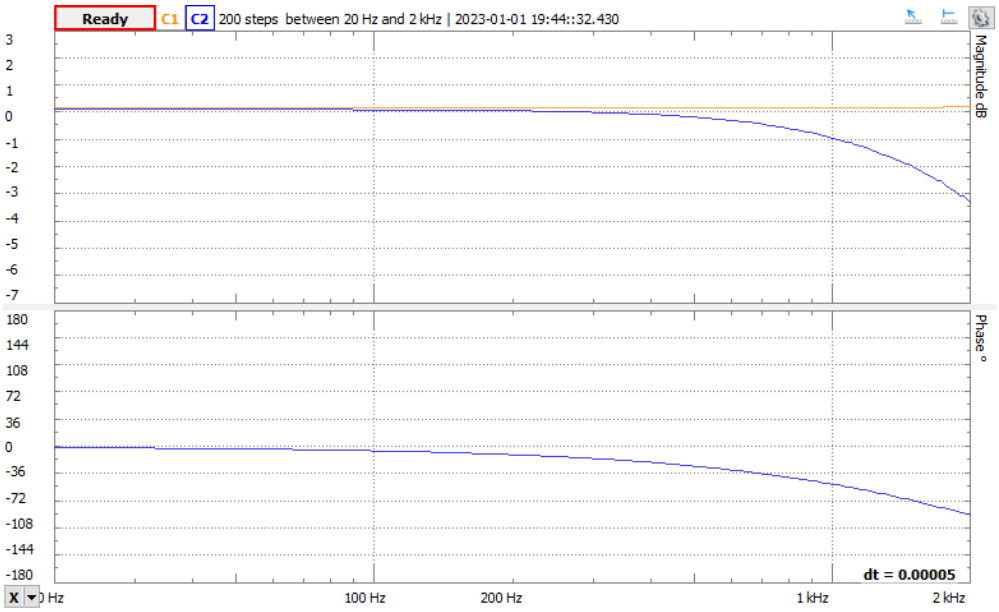
Timing jitter varies with the block functions used. The loop typically has about ±0.4uS of jitter for math only blocks. Printing and reading the ADC can introduce jumps of up to 5uS (perhaps due to interrupts). However, the average loop time is typically within 0.1uS of the set value. Shorter loop times will have greater sensitivity to the jitter.



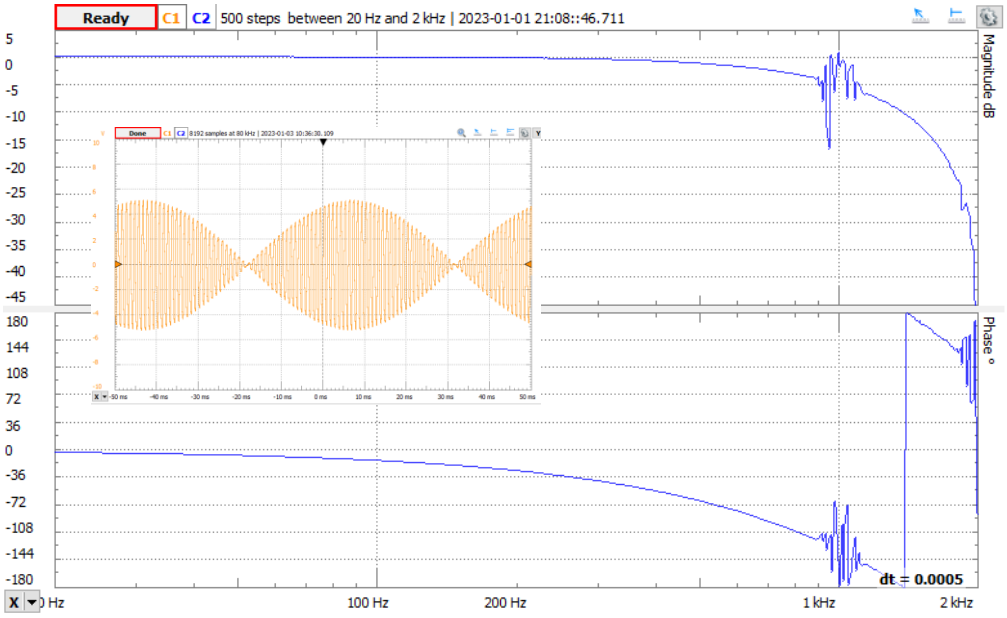
A 19.95KHz input sampled with dt = 0.00005 (20KHz). The 50Hz aliased signal shows signs of amplitude tearing caused by jitter.



A 1.95KHz input sampled with dt = 0.0005 (2KHz). The 50Hz aliased signal looks rippled because the DAC is following larger steps between each sample. There are 40 output steps for each sine cycle.



A Bode plot of the output/input response with a 20KHz sample loop. The DAC response is flat within 1dB up to 1KHz.



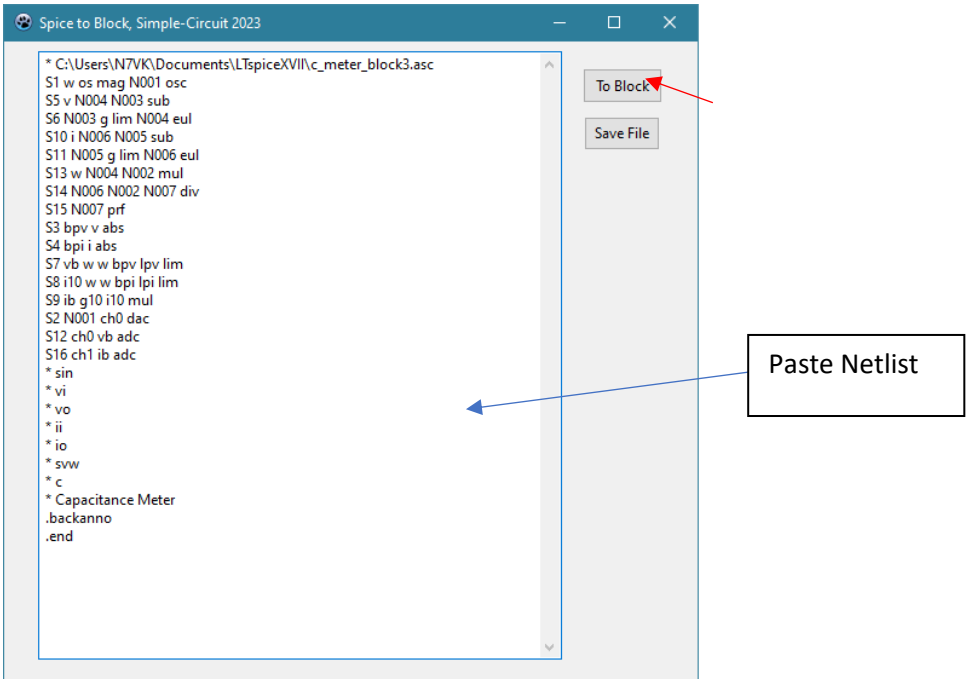
A Bode plot of the output/input response with a 2KHz sample loop. The faster phase drop off is caused by the slower sample rate. The output begins to break up around 1KHz which is the Nyquist frequency. The o'scope insert shows a sampled 990Hz waveform. The analyzer has difficulty determining magnitude and phase.

For best results, it is recommended to keep the sampling frequency five times higher than the highest useable frequency.

<u>Graphical Programming:</u>

The Block compiler is text only. However, the schematic capture in LTspice can be borrowed to layout a block program. The netlist generated is in the order the blocks are placed. Here is the process:

1. Copy the "block" symbol folder into the following directory: Documents\LTspice…\lib\sym
    a. Where LTspice… is the version you are running
2. Click Component menu buttonto select block and then chose your block from the list
3. Connect your blocks with wires
4. set blocks are used to initialize variables
    a. Use 0.0 for the value 0, otherwise it will become ground
5. Adding a Label Net to your wires will help with code reading but is not required
6. Save the schematic
7. Click on menu View then SPICE Netlist
8. Use Ctrl-A then Ctrl-C to copy the netlist text
9. Run spice_to_block.exe (Windows only, sorry)
10. Use Ctrl-V to paste the netlist into the text box
11. Click the To Block button
12. Click the Save button



Paste Netlist