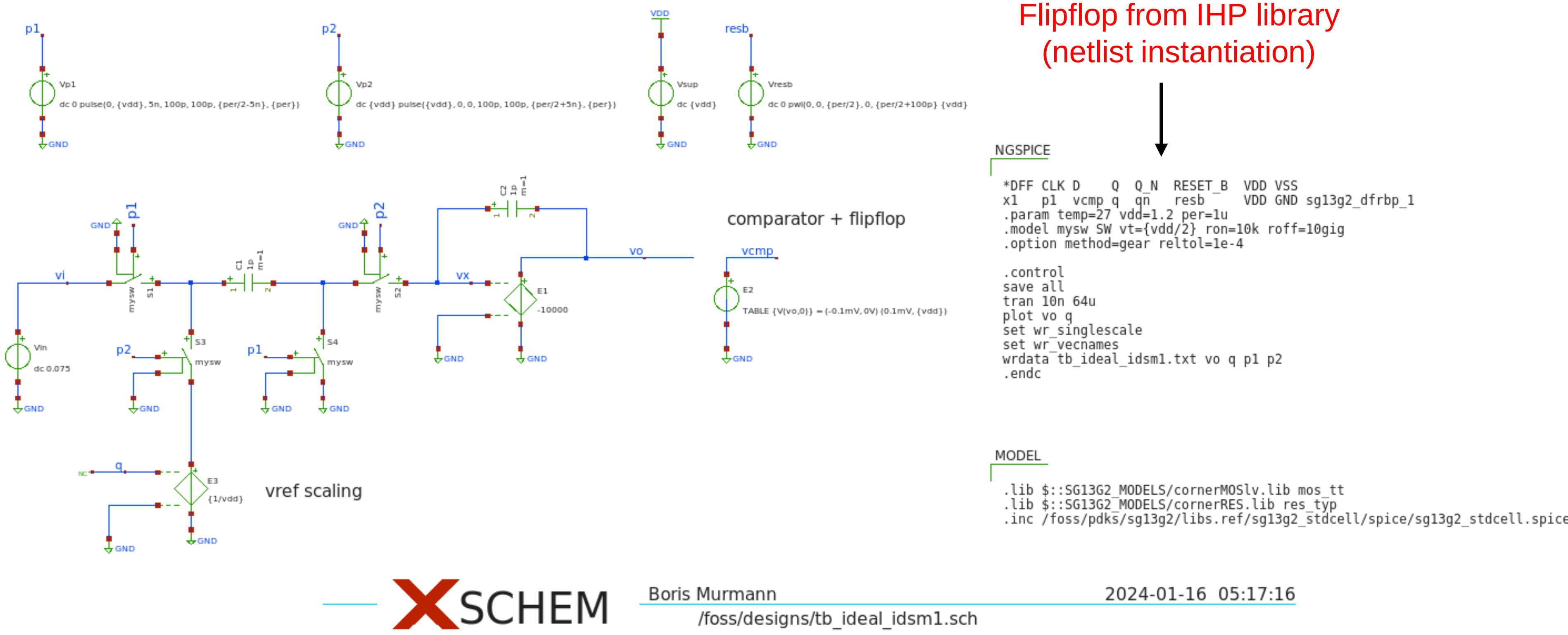
IDSM2 Circuit Model

Boris Murmann

bmurmann@hawaii.edu

IDSM1 Circuit Model

Flipflop from IHP library
(netlist instantiation)



p1
Vp1
dc 0 pulse(0, {vdd}, 5n, 100p, 100p, {per/2-5n}, {per})
p2
Vp2
dc {vdd} pulse({vdd}, 0, 0, 100p, 100p, {per/2+5n}, {per})
VDD
Vsup
dc {vdd}
resb
Vresb
dc 0 pwl(0, 0, {per/2}, 0, {per/2+100p} {vdd})
GND
vi
S1
mysw
C1
1p
m=1
S2
C2
vx
E1
-10000
vo
comparator + flipflop
vcmp
E2
TABLE {V(vo,0)} = (-0.1mV, 0V) (0.1mV, {vdd})
Vin
dc 0.075
S3
S4
q
NC
E3
{1/vdd}
vref scaling
XSCHEM
Boris Murmann
2024-01-16 05:17:16
/foss/designs/tb_ideal_idsm1.sch

NGSPICE

```
*DFF CLK D    Q  Q_N  RESET_B  VDD VSS
x1   p1  vcmp q  qn   resb     VDD GND sg13g2_dfrbp_1
.param temp=27 vdd=1.2 per=1u
.model mysw SW vt={vdd/2} ron=10k roff=10gig
.option method=gear reltol=1e-4

.control
save all
tran 10n 64u
plot vo q
set wr_singlescale
set wr_vecnames
wrdata tb_ideal_idsm1.txt vo q p1 p2
.endc
```

MODEL

```
.lib $::SG13G2_MODELS/cornerMOSlv.lib mos_tt
.lib $::SG13G2_MODELS/cornerRES.lib res_typ
.inc /foss/pdks/sg13g2/libs.ref/sg13g2_stdcell/spice/sg13g2_stdcell.spice
```

Fixing the IHP Standard Cell Symbols

- Start the container as root

```
docker exec -u root -t -i iic-osic-tools_xserver_uid_1000 /bin/bash
```

- Go to directory containing the IHP standard cell symbols

```
cd /foss/pdks/sg13g2/libs.tech/xschem/sg13g2_stdcells
```

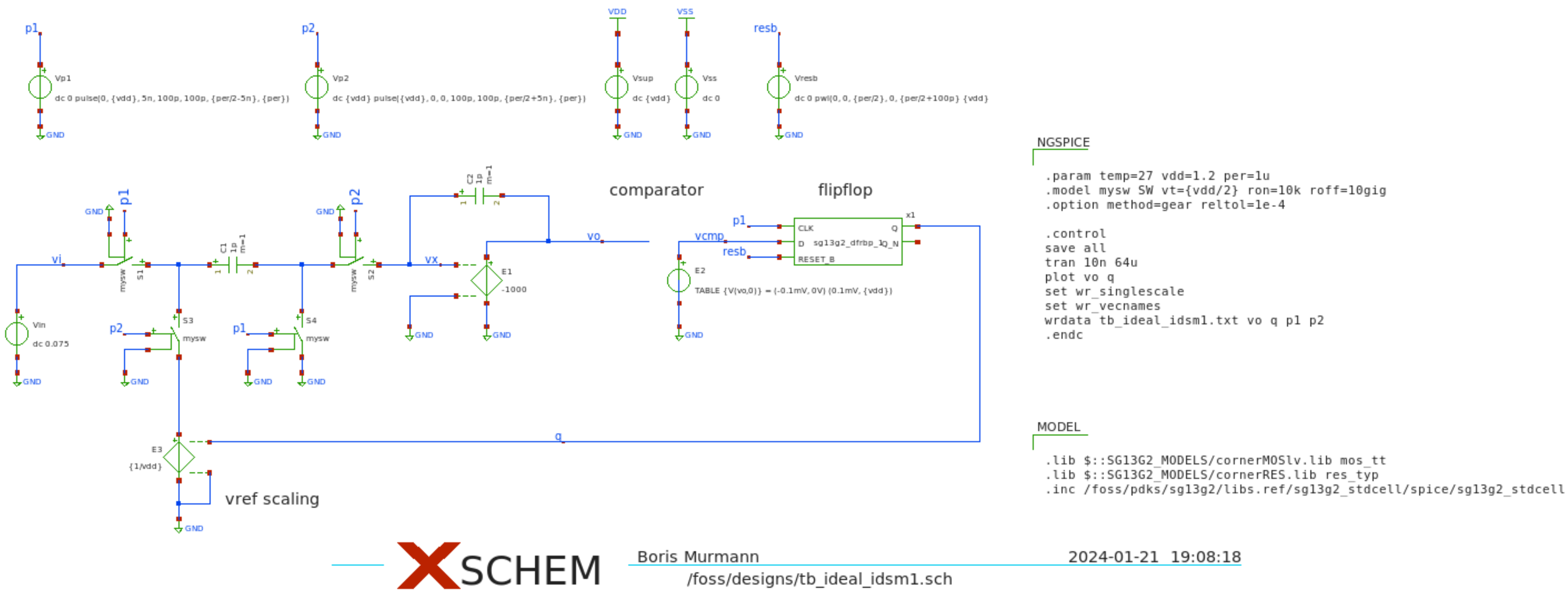
- Run this command to fix the symbols’ pin order (courtesy Mitch Bailey)

```
sed -i \
-e 's/@VGND @VNB @VPB @VPWR/@VDD @VSS/' \
-e 's/VGND=VGND VNB=VNB VPB=VPB VPWR=VPWR/VDD=VDD VSS=VSS/' \
-e 's/VGND VNB VPB VPWR/VDD VSS/' \
-e 's/@VDD @VSS @@Q @@Q_N/@@Q @@Q_N @VDD @VSS/’ *
```

- This still doesn’t completely fix the flipflop cell. Using a text editor, move “RESET_B” to the location shown below in in sg13g2_dfrbp_1.sym

```
format="@name @@CLK @@D @@Q @@Q_N @@RESET_B @VDD @VSS @prefix\\\\dfrbp_1"
```

First-Order Modulator with DFF Symbol


p1
Vp1
dc 0 pulse(0, {vdd}, 5n, 100p, 100p, {per/2-5n}, {per})
p2
Vp2
dc {vdd} pulse({vdd}, 0, 0, 100p, 100p, {per/2+5n}, {per})
VDD
Vsup
dc {vdd}
VSS
Vss
dc 0
resb
Vresb
dc 0 pwl(0, 0, {per/2}, 0, {per/2+100p} {vdd}
GND
comparator
flipflop
vi
Vin
dc 0.075
S1
mysw
C1
1p
m=1
S2
C2
vx
vo
E1
-1000
S3
S4
vcmp
E2
TABLE {V(vo,0)} = (-0.1mV, 0V) (0.1mV, {vdd})
x1
CLK
D
RESET_B
Q
Q_N
sg13g2_dfrbp_1
q
E3
{1/vdd}
vref scaling

NGSPICE

```
.param temp=27 vdd=1.2 per=1u
.model mysw SW vt={vdd/2} ron=10k roff=10gig
.option method=gear reltol=1e-4

.control
save all
tran 10n 64u
plot vo q
set wr_singlescale
set wr_vecnames
wrdata tb_ideal_idsm1.txt vo q p1 p2
.endc
```

MODEL

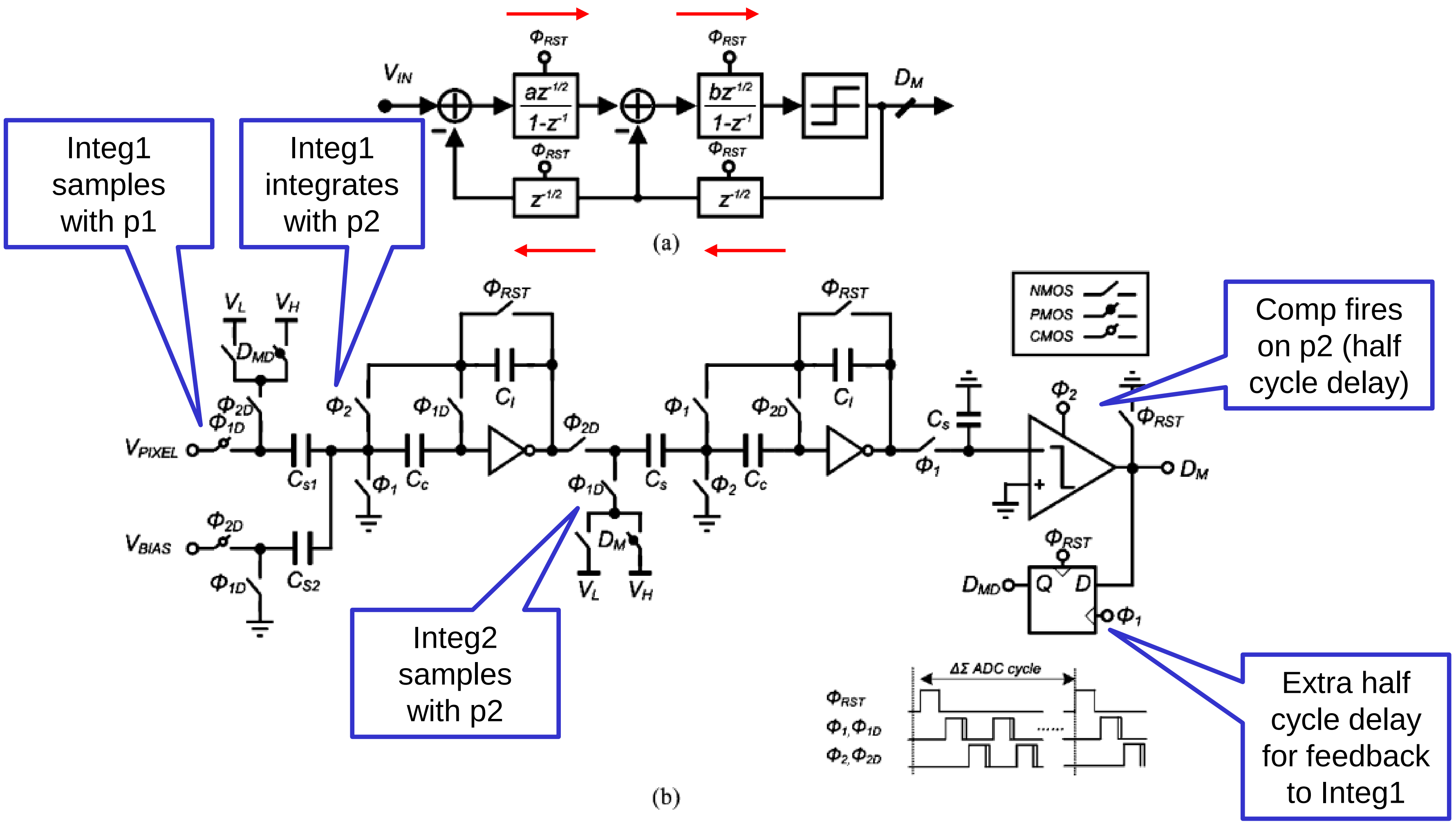
```
.lib $::SG13G2_MODELS/cornerMOSlv.lib mos_tt
.lib $::SG13G2_MODELS/cornerRES.lib res_typ
.inc /foss/pdks/sg13g2/libs.ref/sg13g2_stdcell/spice/sg13g2_stdcell
```

XSCHEM Boris Murmann 2024-01-21 19:08:18
/foss/designs/tb_ideal_idsm1.sch

Next Steps

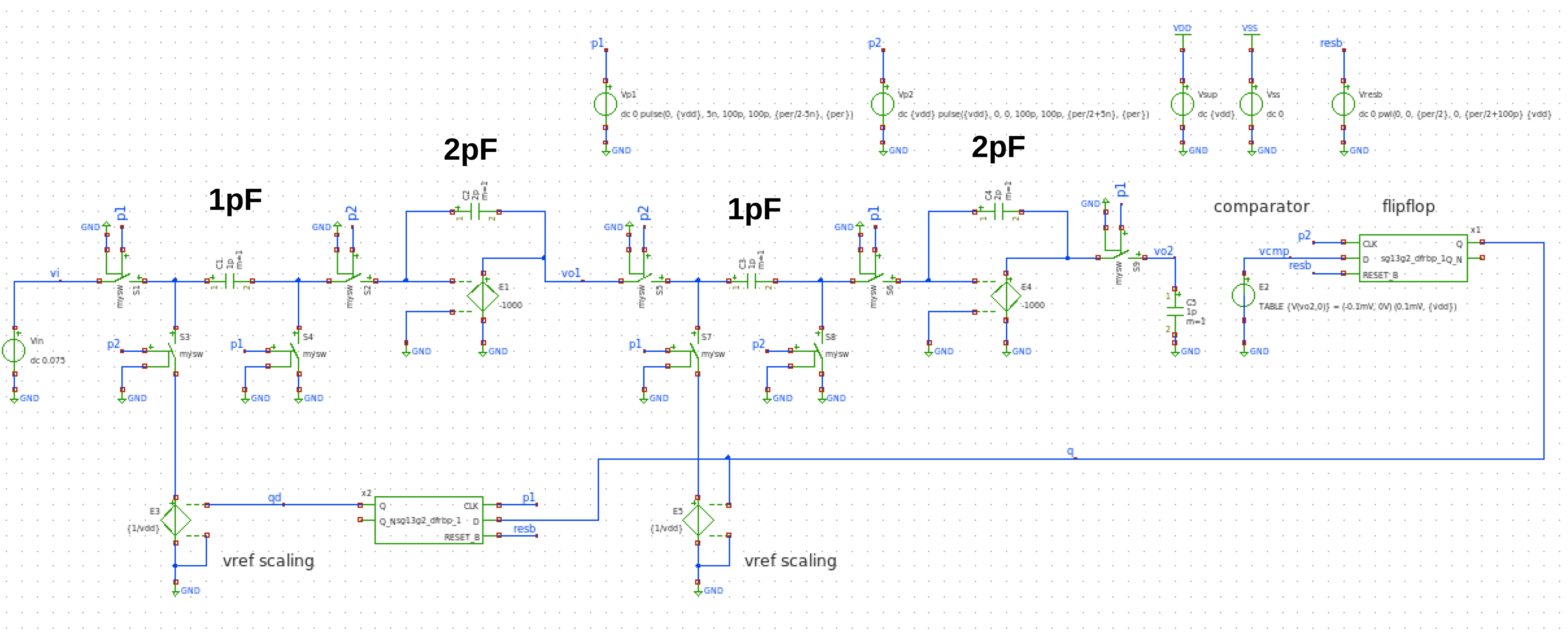
- Add second integrator (second-order modulator)
- Write some post-processing scripts
  - Detailed measurements
  - Emulate counters to create final digital output
- Increasingly “transistorize” the implementation
  - Shift voltages to practical levels (can’t have negative voltages)
  - Comparator
  - Clock generator
  - MOSFET switches
  - Inverter-based amplifier
    - With correlated double-sampling to achieve high gain

Template Circuit



Y. Chae et al., "A 2.1 M Pixels, 120 Frame/s CMOS Image Sensor With Column-Parallel ADC Architecture," in IEEE Journal of Solid-State Circuits, Jan. 2011. https://ieeexplore.ieee.org/document/5641589

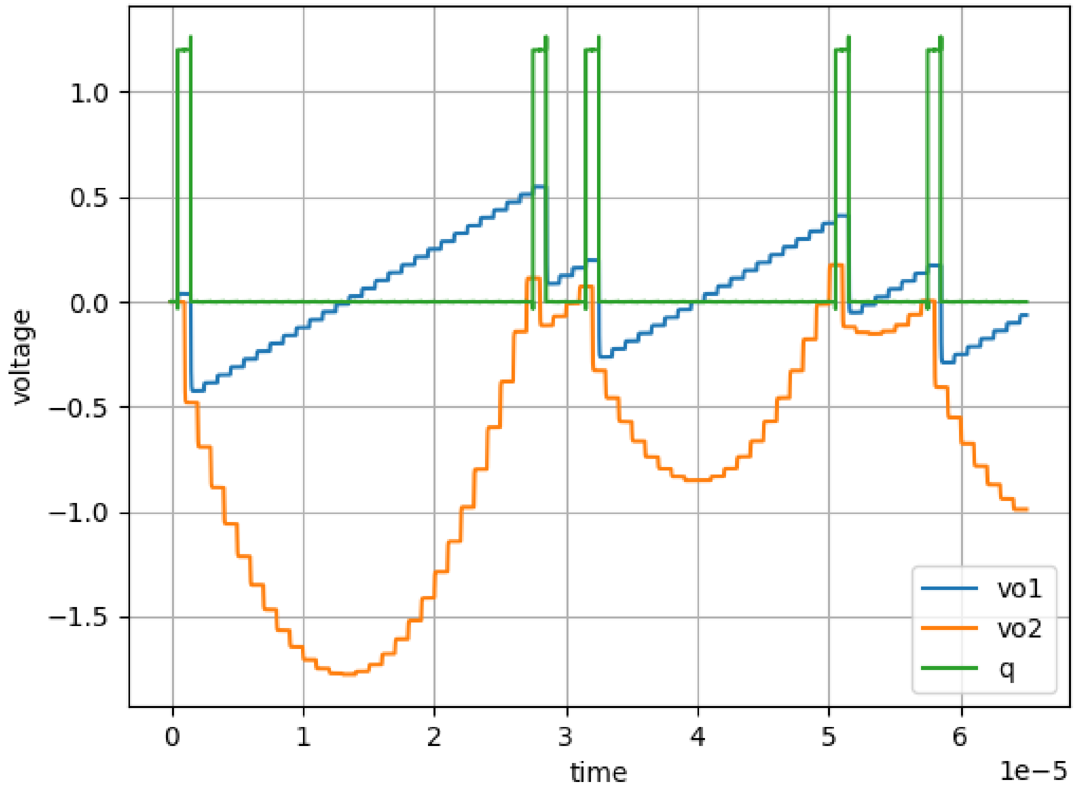
Second-Order Modulator



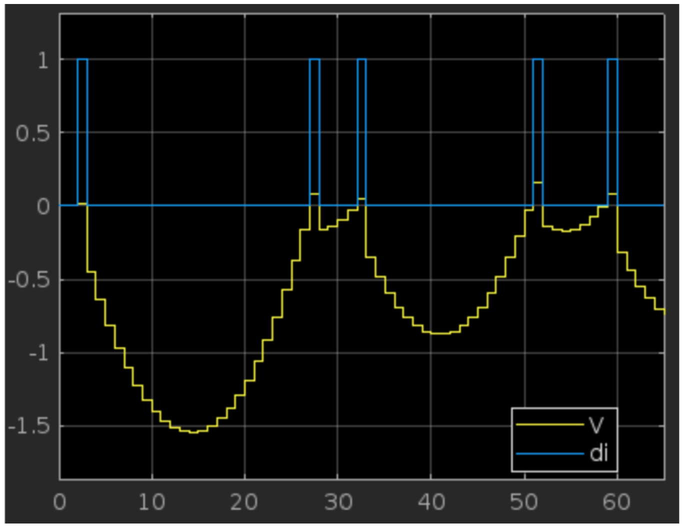
1pF
2pF
1pF
2pF
comparator
flipflop
vref scaling
vref scaling
vi
vo1
vo2
vcmp
qd
q
p1
p2
resb
Vin dc 0.075
Vp1 dc 0 pulse(0, {vdd}, 5n, 100p, 100p, {per/2-5n}, {per})
Vp2 dc {vdd} pulse({vdd}, 0, 0, 100p, 100p, {per/2+5n}, {per})
Vsup dc {vdd}
Vss dc 0
Vresb dc 0 pwl(0, 0, {per/2}, 0, {per/2+100p} {vdd})
E1 -1000
E4 -1000
E2 TABLE {V(vo2,0)} = (-0.1mV, 0V) (0.1mV, {vdd})
E3 {1/vdd}
E5 {1/vdd}
sg13g2_dfrbp_1

Looks a little complicated; better to stick integrators into a subcircuit + symbol

Simulation Result



Simulink

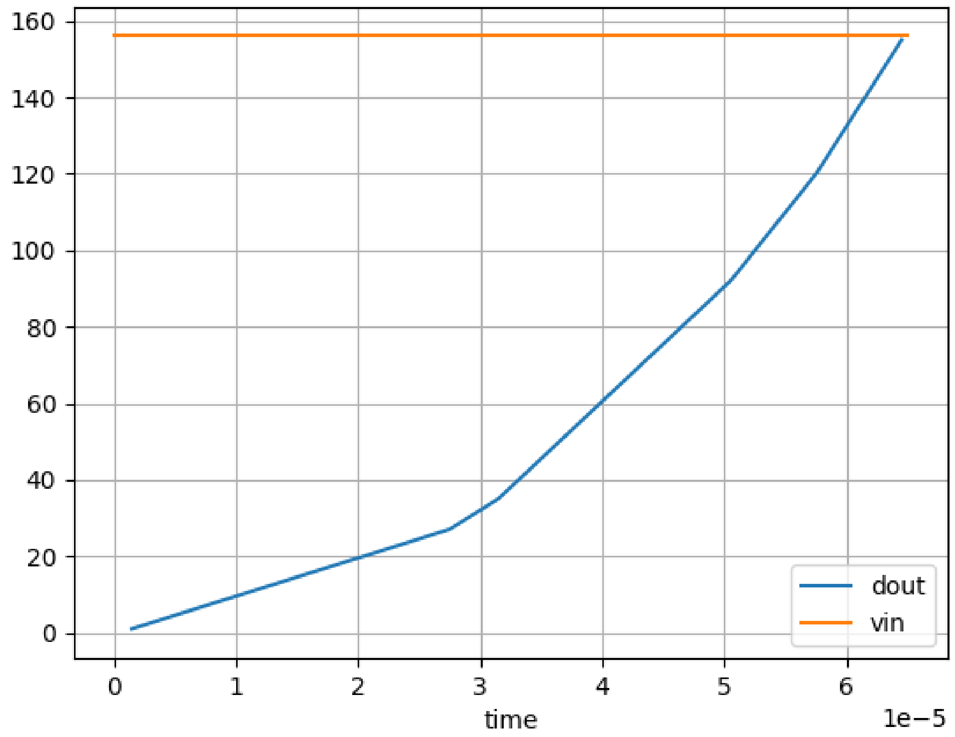


Close, but not the expected trajectory for int2 output. Why?

Postprocessing

```
# sample q and count number of ones
t = df['time']
ts = np.arange(1.5e-6, 1.5e-6+64*1e-6, 1e-6)
q = df['q']
interp_func = interp1d(t, q)
qsamp = interp_func(ts)
qsamp[qsamp > 0.5] = 1
qsamp[qsamp < 0.5] = 0
csum = np.cumsum(qsamp)
dout = np.cumsum(csum)
vin = 0.075*64.0*65.0/2.0
```

```
plt.figure(2)
plt.clf()
plt.plot(ts, dout, label="dout")
plt.plot([0, 65e-6], [vin, vin], label="vin")
plt.xlabel("time")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



```
dout[-1]
```

```
155.0
```

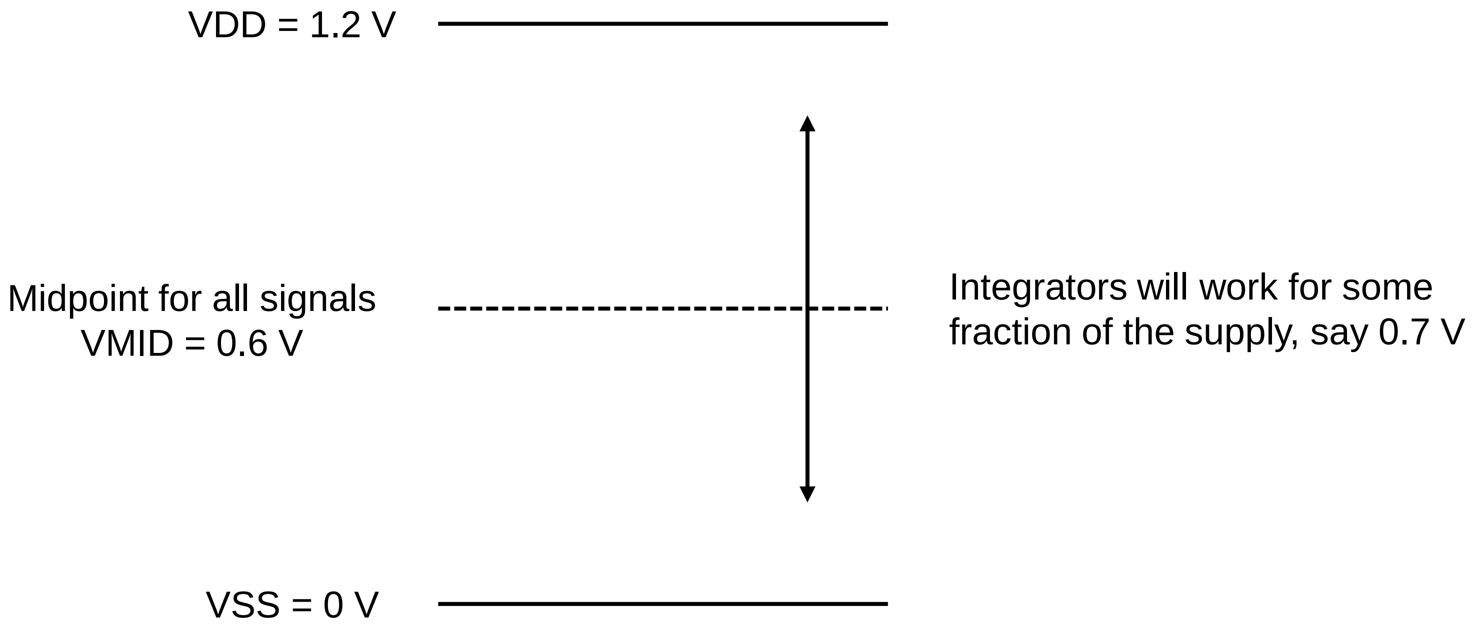
```
vin
```

```
156.0
```

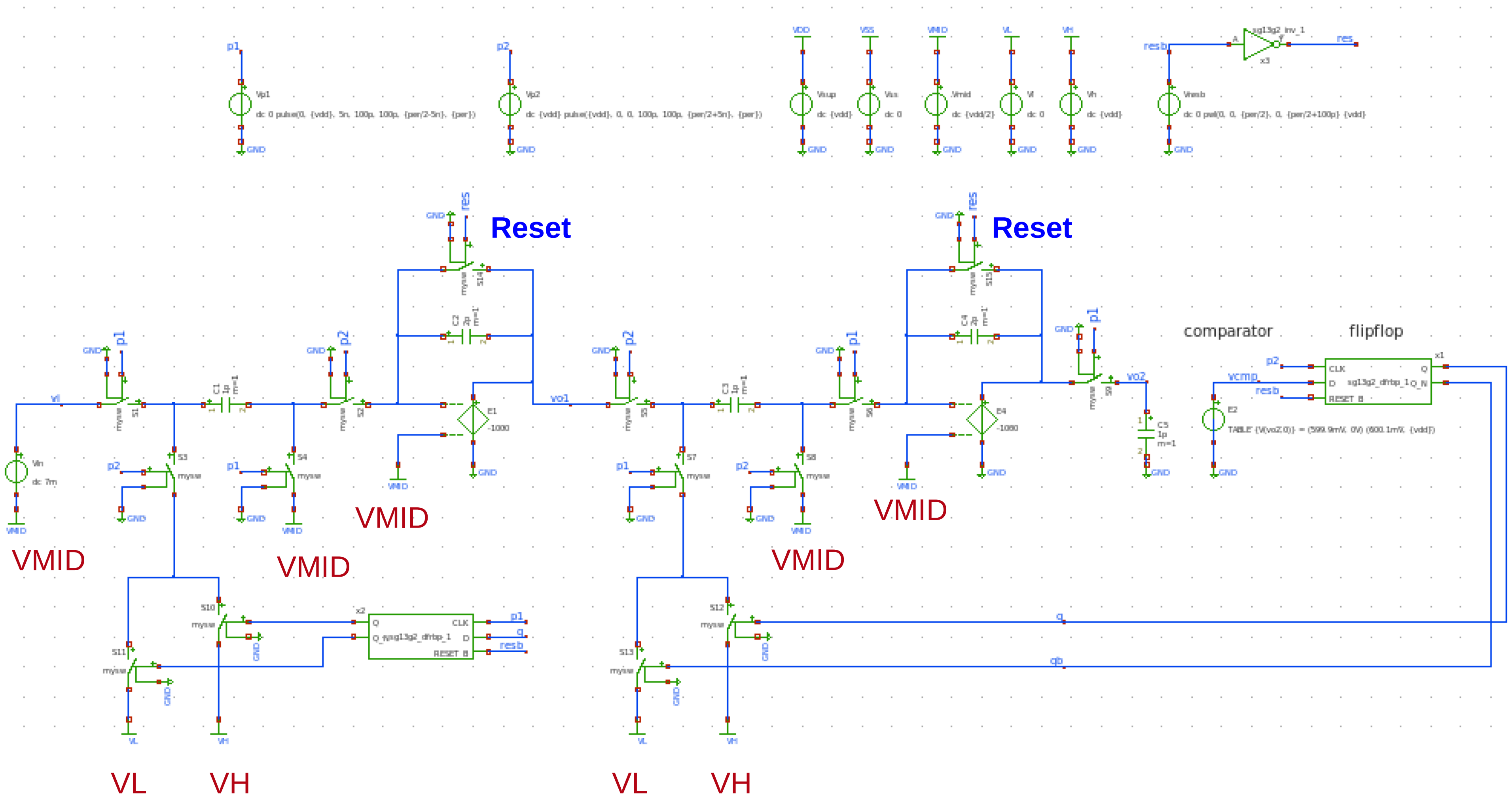
Off by one quantization step after (64+1 us). Need one more clock cycle because of initial reset phase.

Probably OK, but best to do an input sweep to assess quantization error bounds (like we did in Simulink)

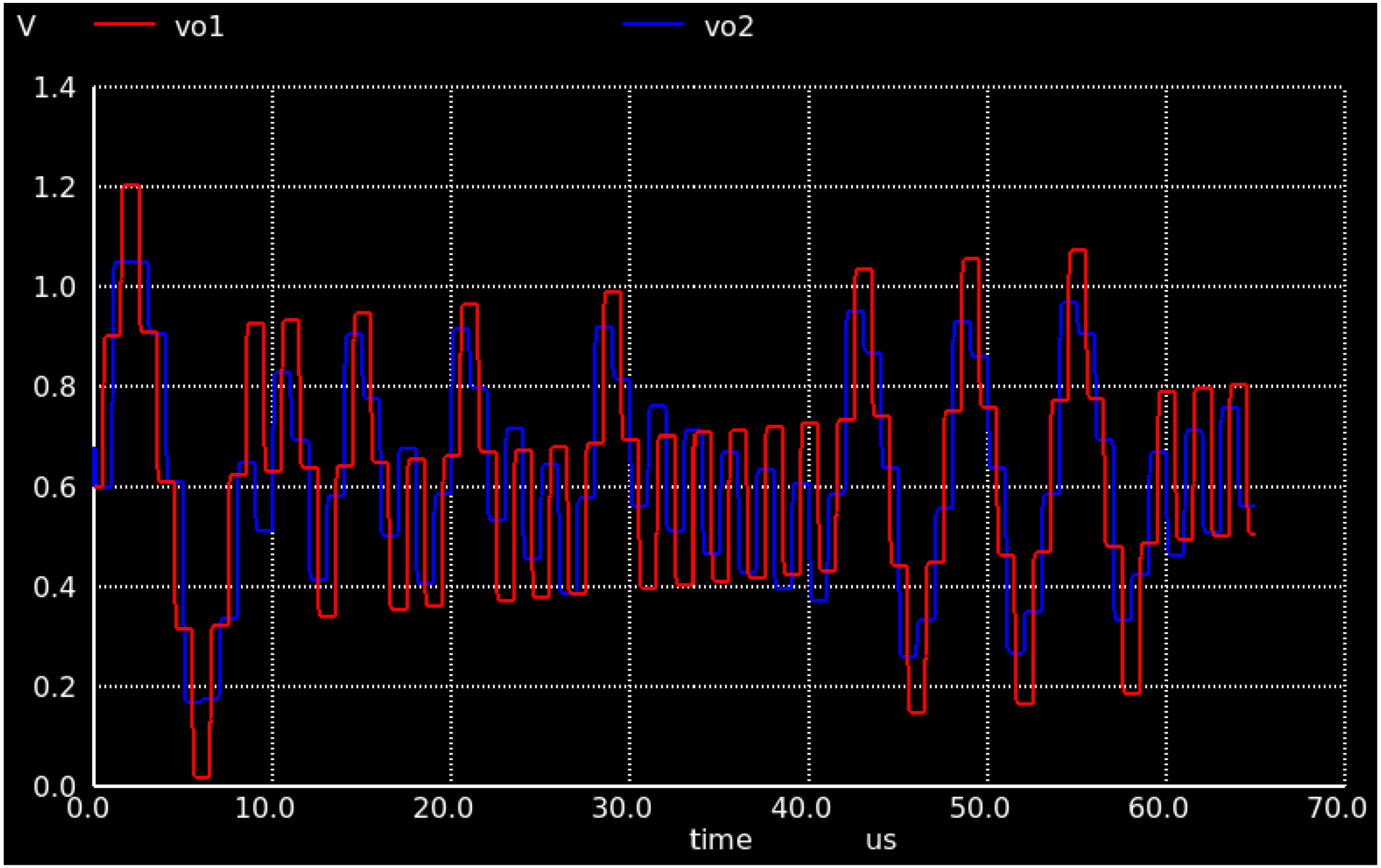
Voltage Range Considerations


VDD = 1.2 V
Midpoint for all signals
VMID = 0.6 V
VSS = 0 V
Integrators will work for some fraction of the supply, say 0.7 V

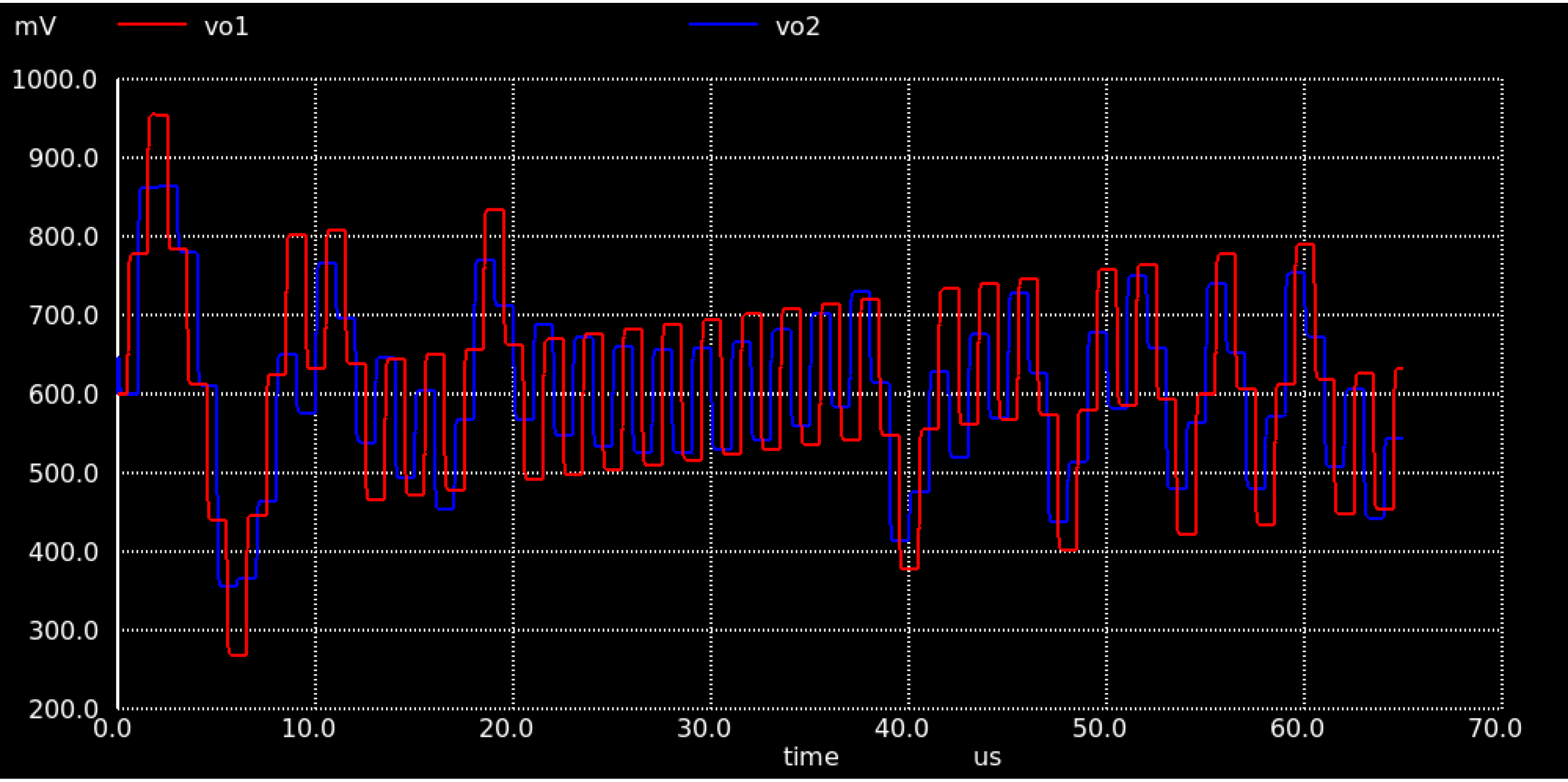
Modified Schematic



VL = 0 V, VH = VDD, V_{in} = 7 mV



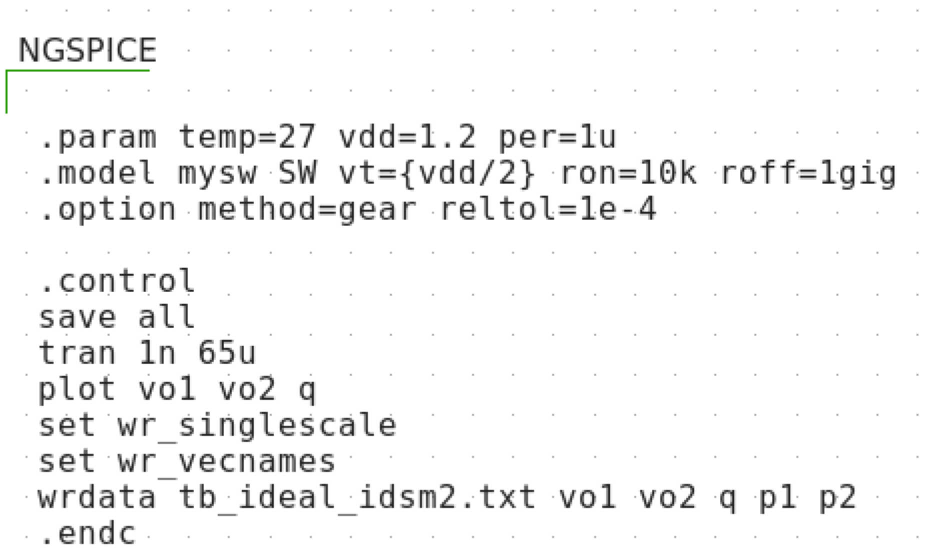
VL = 0.25V, VH = VDD-0.25V, V_{in} = 7 mV



Looks good!

Simulation Settings

- You may have already noticed that these idealized circuit simulations are relatively slow
- This is partly because the circuits contains very fast transients, high-gain elements, sharp nonlinearities, huge impedance changes (Ron/Roff)
- You may sometimes see convergence issues that can often be resolved making things slightly less ideal (e.g., roff = 10gig → 1 gig)



```
NGSPICE

 .param temp=27 vdd=1.2 per=1u
 .model mysw SW vt={vdd/2} ron=10k roff=1gig
 .option method=gear reltol=1e-4

 .control
 save all
 tran 1n 65u
 plot vo1 vo2 q
 set wr_singlescale
 set wr_vecnames
 wrdata tb_ideal_idsm2.txt vo1 vo2 q p1 p2
 .endc
```

Next Steps

- Add second integrator (second-order modulator)
- Write some post-processing scripts
  - Detailed measurements
  - Emulate counters to create final digital output

- Increasingly “transistorize” the implementation
  - Shift voltages to practical levels (can’t have negative voltages)
  - Comparator
  - Inverter-based amplifier with CDS
  - MOSFET switches & clock generator