Qspice - How Time Step Works

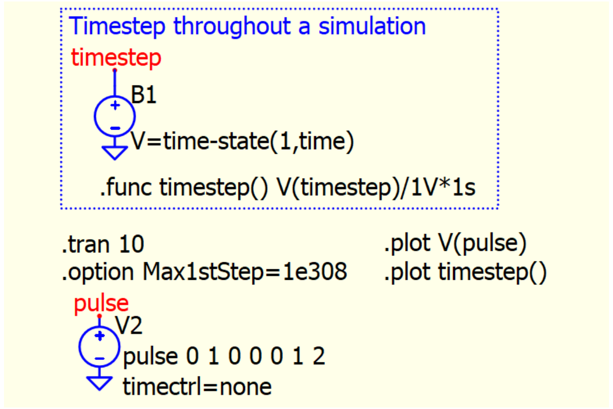
KSKelvin Kelvin Leung

Created on : 5-23-2024
Last Update : 11-29-2025

How Time Step Works in Qspice – TimeStep

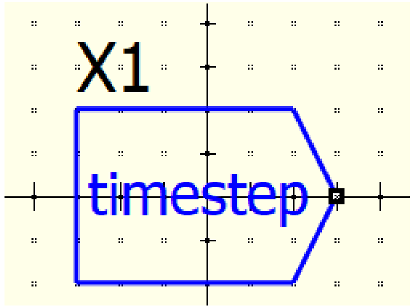
Qspice : timestep.qsch

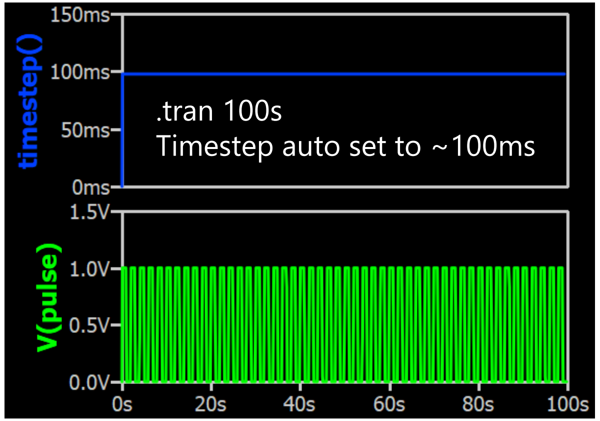
- Timestep
  - Qspice Simulation
    - time : returns the current time value
  - Simulation timestep can be determined using the state(n,x) function
  - B-source with formula **time – state(1,time)**
    - Calculates the difference between the current time and the time value from one time step ago
  - KSKelvin's Symbol library
    - a symbol named Timestep.qsym has been created to provide the timestep value

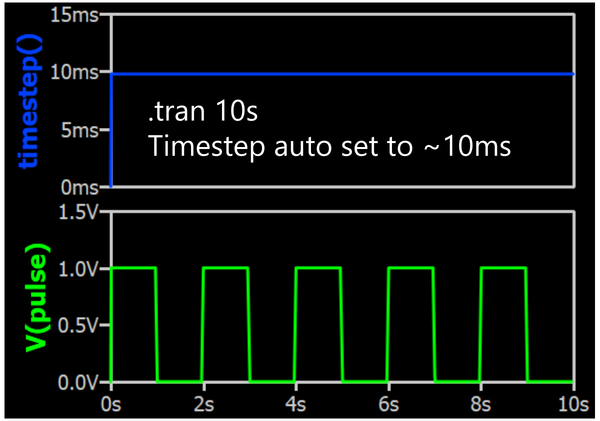


```
Timestep throughout a simulation
timestep
B1
V=time-state(1,time)
.func timestep() V(timestep)/1V*1s

.tran 10
.option Max1stStep=1e308
.plot V(pulse)
.plot timestep()

pulse
V2
pulse 0 1 0 0 0 1 2
timectrl=none
```

Timestep.qsym



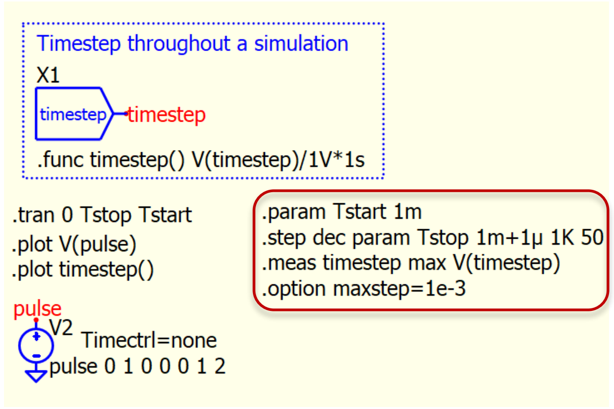




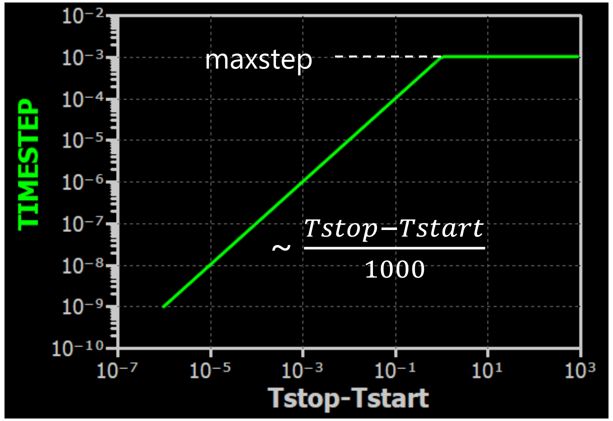
How timestep works?

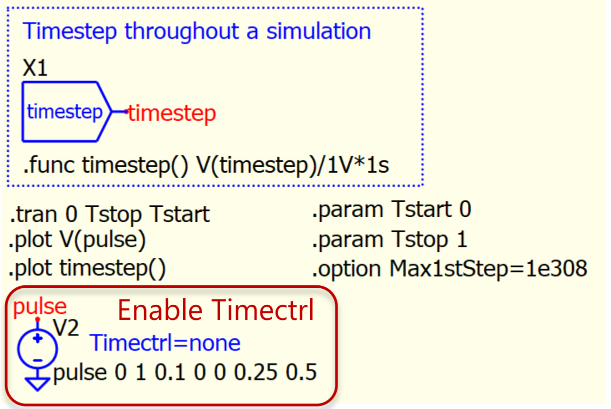
Qspice : timestep- MaxStep.qsch | timestep - Pulse Timectrl.qsch

- #1a .option maxstep
  - Maximum timestep
- #1b .tran Tstart to Tstop
  - Without timestep modification devices, Qspice set a constant timestep
  - Timestep= $\min\left(\sim\frac{Tstop-Tstart}{1000}, maxstep\right)$
- #2a Timectrl Devices
  - Device (Voltage Source, Switch, ¥-Device etc…) can affect timestep
  - A voltage source with instance parameter Timectrl can reduce the timestep at signal break point

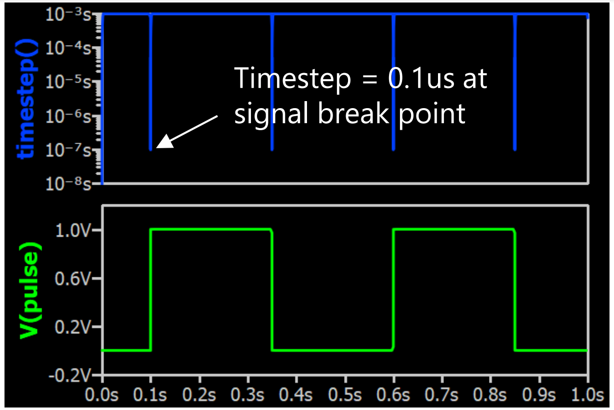


```
.func timestep() V(timestep)/1V*1s

.tran 0 Tstop Tstart
.plot V(pulse)
.plot timestep()

.param Tstart 1m
.step dec param Tstop 1m+1μ 1K 50
.meas timestep max V(timestep)
.option maxstep=1e-3

Timectrl=none
pulse 0 1 0 0 0 1 2
```





```
.func timestep() V(timestep)/1V*1s

.tran 0 Tstop Tstart
.plot V(pulse)
.plot timestep()

.param Tstart 0
.param Tstop 1
.option Max1stStep=1e308

Timectrl=none
pulse 0 1 0.1 0 0 0.25 0.5
```

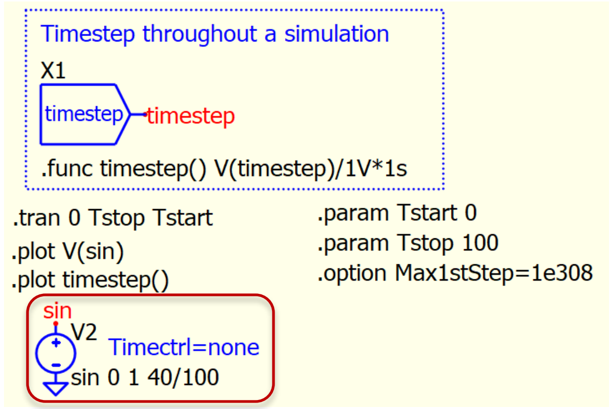


How timestep works?

Qspice : timestep - Sin Timectrl.qsch | timestep - Max1stStep.qsch

- #2b Timectrl Devices
  - V/I sources have different Timectrl strategies
    - For example, sine source reduce timestep when $\frac{dv}{dt}$ change direction
  - Setting the Instance parameter Timectrl=none for source will disable the timestep control strategy

- #3 .option Max1stStep
  - **.option Max1stStep** controls the maximum timestep size for the first timestep in a .tran
    - Default Max1stStep=100ns
  - To disable Max1stStep, set .option Max1stStep=1e308


Timestep throughout a simulation

X1

timestep timestep

.func timestep() V(timestep)/1V*1s

.tran 0 Tstop Tstart

.plot V(sin)

.plot timestep()

.param Tstart 0

.param Tstop 100

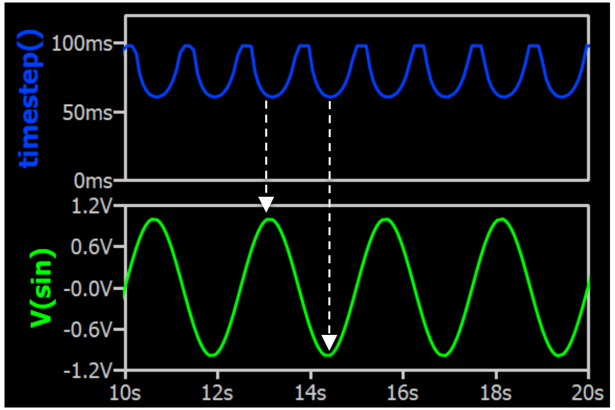
.option Max1stStep=1e308

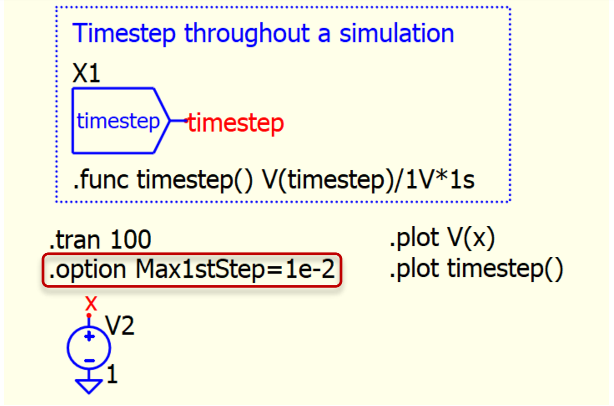
sin

V2

Timectrl=none

sin 0 1 40/100




Timestep throughout a simulation

X1

timestep timestep

.func timestep() V(timestep)/1V*1s

.tran 100

.option Max1stStep=1e-2

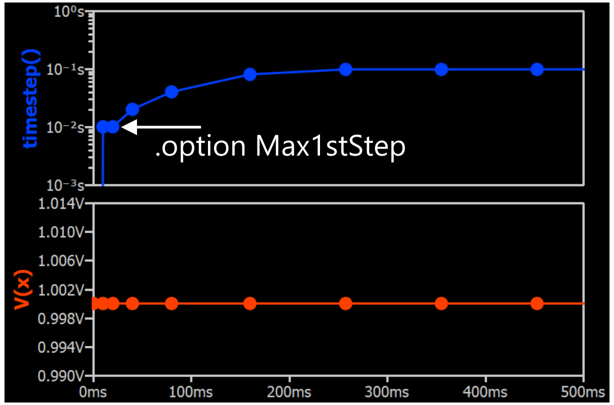
.plot V(x)

.plot timestep()

x

V2

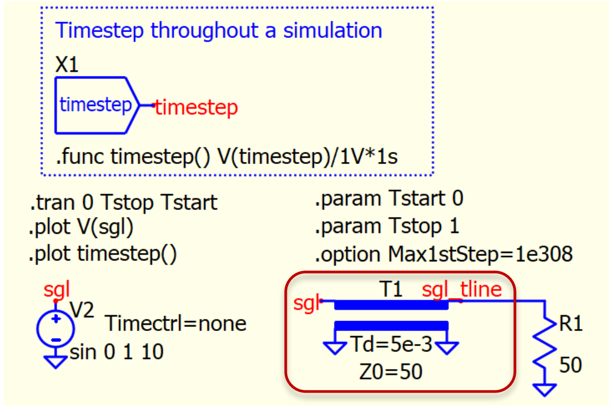
1



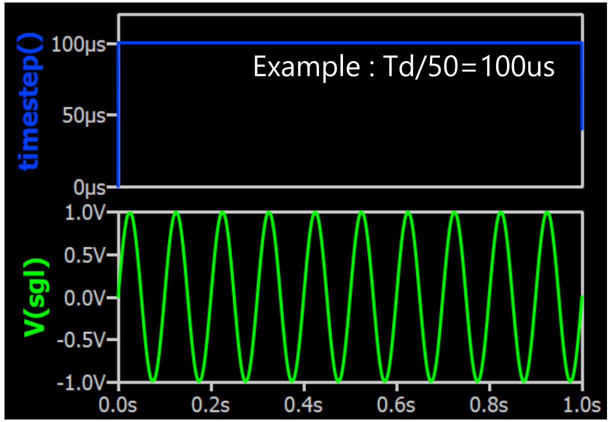
How timestep works?

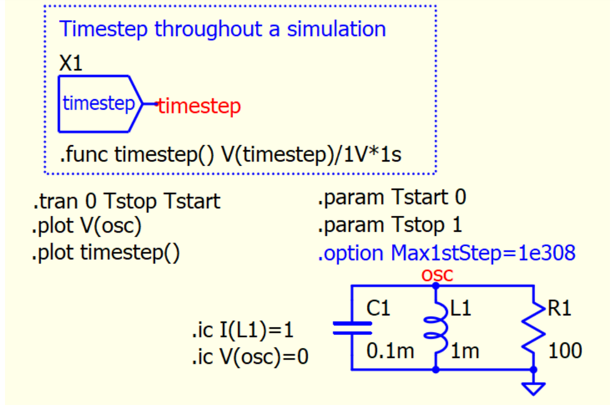
Qspice : timestep - Tline.qsch | timestep - LC.qsch

- #4 Transmission Line
  - Td of an ideal transmission line will force the target timestep to $\text{Timestep} = \frac{\text{Td}}{50}$
- #5 LC oscillation
  - Qspice can changes its timestep if circuit consist of resonant elements and before oscillation is damped

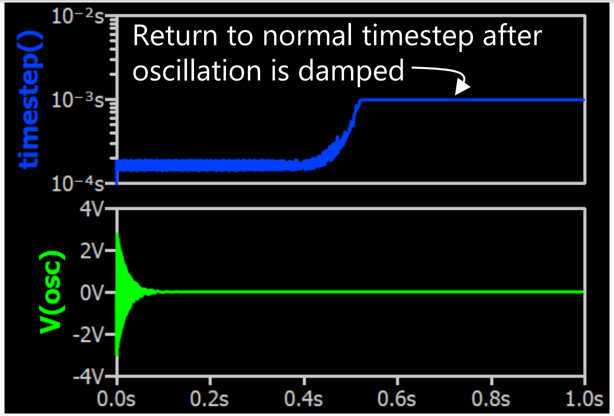


```
.func timestep() V(timestep)/1V*1s
.tran 0 Tstop Tstart
.plot V(sgl)
.plot timestep()
.param Tstart 0
.param Tstop 1
.option Max1stStep=1e308
```



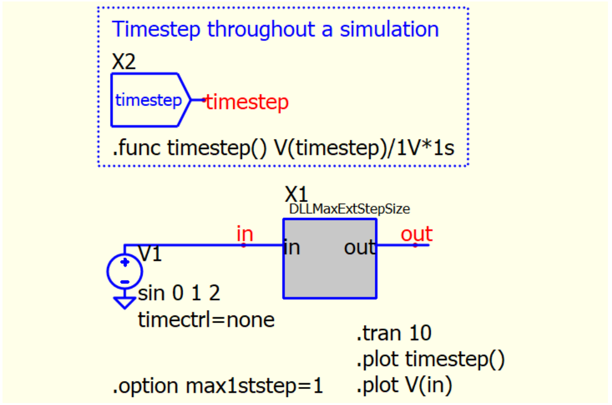


```
.func timestep() V(timestep)/1V*1s
.tran 0 Tstop Tstart
.plot V(osc)
.plot timestep()
.param Tstart 0
.param Tstop 1
.option Max1stStep=1e308
.ic I(L1)=1
.ic V(osc)=0
```

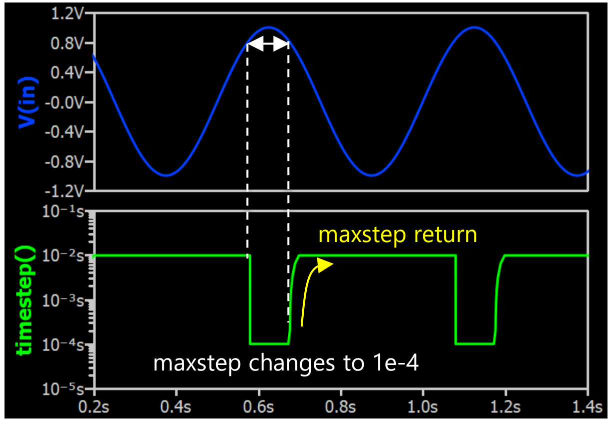


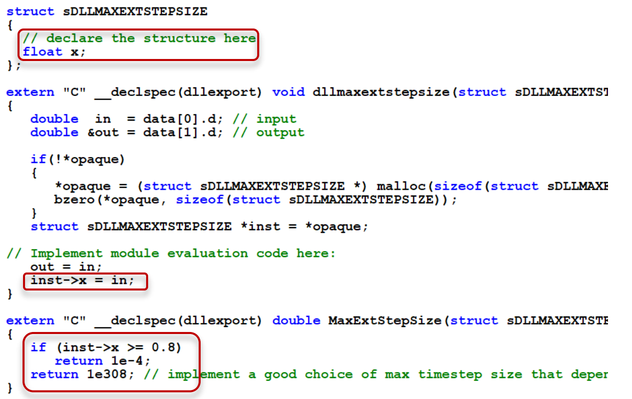
How timestep works?
Qspice : timestep - MaxExtStepSize.qsch

- #6 MaxExtStepSize (DLL)
  - MaxExtStepSize() is a function in DLL device
  - It allows a structure variable to be passed in order to control the maximum timestep
  - The return value of MaxExtStepSize() will determine the maxstep value
  - In this example
    - Target maximum step is determined by condition explained in #1b, which is 10s/1000=1e-2=10^{-2}s
    - In the DLL, MaxExtStepSize() reduces maxstep to 1e-4=10^{-4}s when V(in) > 0.8



```
.func timestep() V(timestep)/1V*1s
.tran 10
.plot timestep()
.plot V(in)
.option max1ststep=1
```



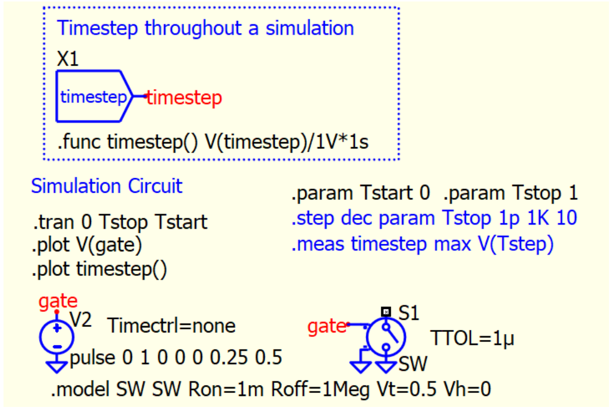


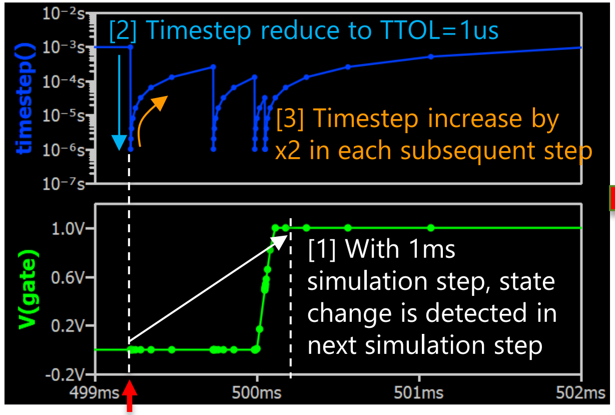
```
struct sDLLMAXEXTSTEPSIZE
{
 // declare the structure here
 float x;
};

extern "C" __declspec(dllexport) void dllmaxextstepsize(struct sDLLMAXEXTST
{
   double  in  = data[0].d; // input
   double &out = data[1].d; // output

   if(!*opaque)
   {
      *opaque = (struct sDLLMAXEXTSTEPSIZE *) malloc(sizeof(struct sDLLMAXE
      bzero(*opaque, sizeof(struct sDLLMAXEXTSTEPSIZE));
   }
   struct sDLLMAXEXTSTEPSIZE *inst = *opaque;

// Implement module evaluation code here:
   out = in;
   inst->x = in;
}

extern "C" __declspec(dllexport) double MaxExtStepSize(struct sDLLMAXEXTSTE
{
   if (inst->x >= 0.8)
      return 1e-4;
   return 1e308; // implement a good choice of max timestep size that deper
}
```

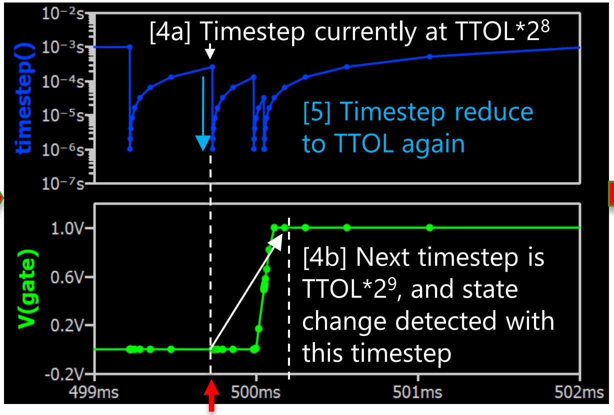
How timestep works? (TTOL Devices) – Switch as example

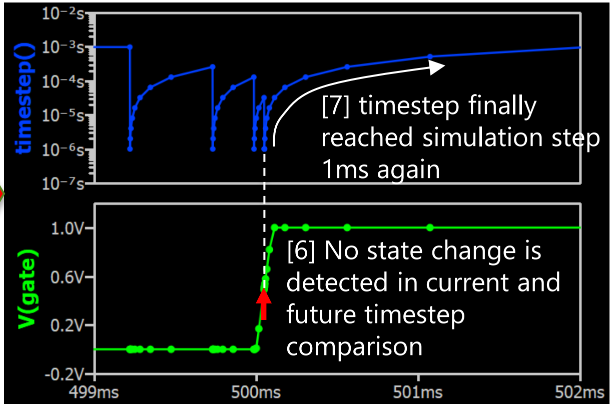
Qspice : timestep - SW TTOL.qsch

- TTOL Temporal Tolerance
  - TTOL is used in Switch, ¥-Device, Ø-Device etc…
  - In Ø-Device, user can control when to trigger *timestep=ttol in the Trunc() function
  - The Trunc() function in the TTOL device is implemented in a meaningful way to detect if the state has changed at the future simulation step (current simulation time + hypothetical next timestep)
  - If the future state, when compared to the current state, is found to have changed in the TTOL device, the *timestep=TTOL is assigned, forcing the next step to only increase by the value of TTOL
  - Following simulation will increase each step by the active timestep multiplied by 2
    - If a state change is detected again, the timestep will be reset to TTOL once more
    - If no state change is detected, the timestep will continue to increase by the active timestep multiplied by 2 until it reaches the simulation step determined by Qspice based on the simulation setup


Timestep throughout a simulation
X1
timestep
timestep
.func timestep() V(timestep)/1V*1s
Simulation Circuit
.param Tstart 0 .param Tstop 1
.tran 0 Tstop Tstart
.step dec param Tstop 1p 1K 10
.plot V(gate)
.meas timestep max V(Tstep)
.plot timestep()
gate
V2
Timectrl=none
pulse 0 1 0 0 0 0.25 0.5
gate
S1
TTOL=1µ
SW
.model SW SW Ron=1m Roff=1Meg Vt=0.5 Vh=0



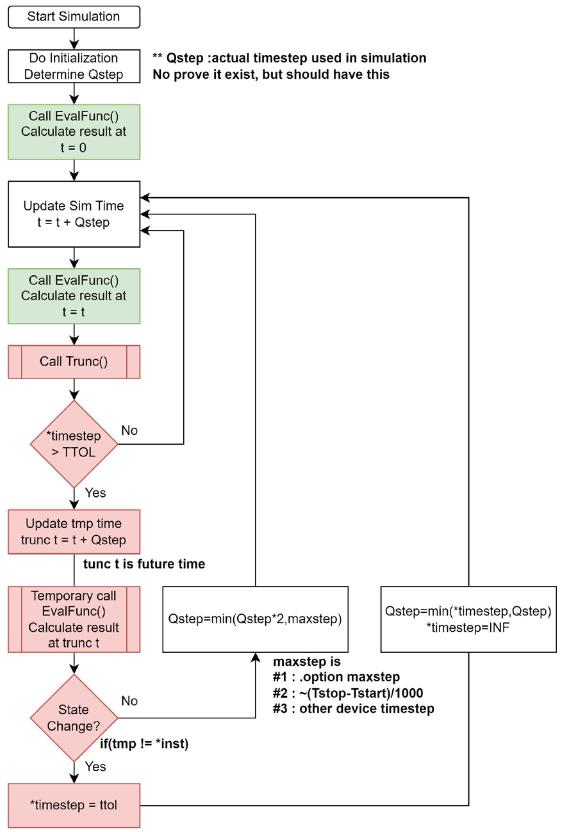


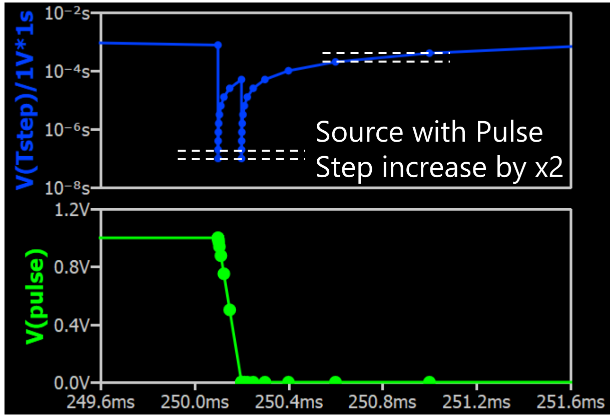


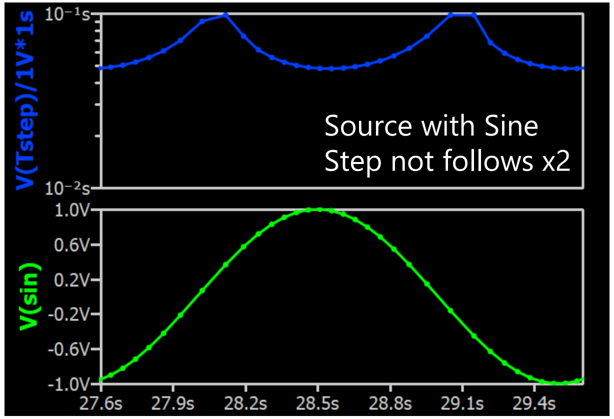
How timestep works? (TTOL Devices) – Switch as example

Qspice : timestep - Pulse Timectrl.qsch | timestep - Sin Timectrl.qsch

- TTOL Temporal Tolerance
  - Qspice employs different timestep control scheme. For example, source with sine doesn't follow x2 timestep relationship as TTOL does (shown in plot)
  - Qspice does not go back in time during simulation but instead looks at the future step (hypothetical) to determine whether it needs to reduce the next simulation timestep
  - If the future step causes a state change, Qspice recognizes that the current timestep is not suitable and reduces its timestep to TTOL
  - Once TTOL is triggered, every subsequent timestep is multiplied by 2 until it reaches the maximum step condition



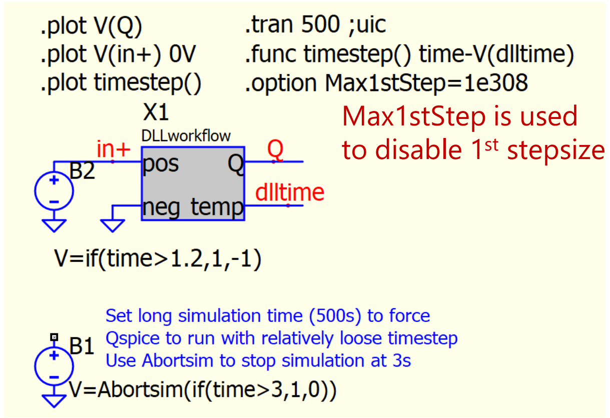




How timestep works? (DLL Ø-Device)

Qspice : \DLL\dll_workflow\DLLworkflow.qsch

- DLL workflow (analysis code)
  - C++ code with multiple display to return t, *timestep at moment includes
    - main : standard main call
    - trunc-main : main called from Trunc()
    - trunc-enter : just enter Trunc()
    - trunc-1st if : just after if(*timestep>ttol) is TRUE
    - trunc-2nd if : just after if(tmp!=*inst) is TRUE
    - trunc-leave : before leaving Trunc()
  - Major variable
    - inst->tmain : dll time (t)
  - Special setup in schematic
    - Setup .tran to 500s but abortsim at 3s, to force Qspice to default maxstep ~ 500ms
    - Timestep is calculated with analog time – DLL time with .func timestep()



```
.plot V(Q)              .tran 500 ;uic
.plot V(in+) 0V         .func timestep() time-V(dlltime)
.plot timestep()        .option Max1stStep=1e308
```

Max1stStep is used to disable 1st stepsize

V=if(time>1.2,1,-1)

Set long simulation time (500s) to force Qspice to run with relatively loose timestep
Use Abortsim to stop simulation at 3s

V=Abortsim(if(time>3,1,0))

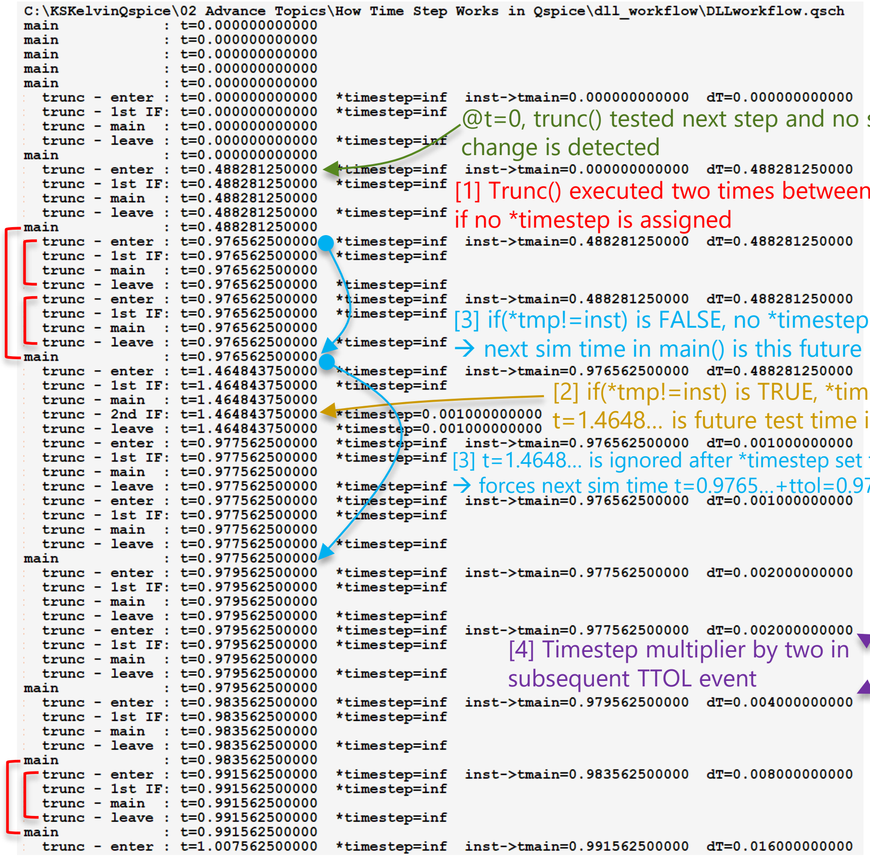
```
// Implement module evaluation code here:
   Q = pos > neg;
   temp = t;
   inst->lastQ = Q;
   inst->tmain = t;
   if (inst->inTrunc == 0)
      display("main              : t=%.12f\r\n",t);
   else
      display("  trunc - main    : t=%.12f\r\n",t);
}
```

```
extern "C" __declspec(dllexport) void Trunc(struc
{ // limit the timestep to a tolerance if the cir
   const double ttol = 1e-3;
   //const double ttol = 1;
   display("  trunc - enter : t=%.12f  *timestep=
   if(*timestep > ttol)
   {
      display("  trunc - 1st IF: t=%.12f  *timest
      bool   &Q    = data[2].b; // output
      double &temp = data[3].d; // output

      // Save output vector
      const bool   _Q    = Q   ;
      const double _temp = temp;

      inst->inTrunc=1;
      struct sDLLWORKFLOW tmp = *inst;
      dllworkflow(&(&tmp), t, data);
      inst->inTrunc=0;
   // if(tmp != *inst) // implement a meaningful 
   //     *timestep = ttol;
      if(tmp.lastQ != inst->lastQ){
          *timestep = ttol;
          display("  trunc - 2nd IF: t=%.12f  *tim
      }

      // Restore output vector
      Q    = _Q   ;
      temp = _temp;
   }
   display("  trunc - leave : t=%.12f  *timestep=
}
```

Inst->inTrunc = 1 if main is called from Trunc()

How timestep works? (DLL Ø-Device) : TTOL=1e-3 in Trunc()

Qspice : \DLL\dll_workflow\DLLworkflow.qsch



```
C:\KSKelvinQspice\02 Advance Topics\How Time Step Works in Qspice\dll_workflow\DLLworkflow.qsch
main             : t=0.000000000000
main             : t=0.000000000000
main             : t=0.000000000000
main             : t=0.000000000000
main             : t=0.000000000000
  trunc - enter : t=0.000000000000  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000000000
  trunc - 1st IF: t=0.000000000000  *timestep=inf
  trunc - main  : t=0.000000000000
  trunc - leave : t=0.000000000000  *timestep=inf
main             : t=0.000000000000
  trunc - enter : t=0.488281250000  *timestep=inf  inst->tmain=0.000000000000  dT=0.488281250000
  trunc - 1st IF: t=0.488281250000  *timestep=inf
  trunc - main  : t=0.488281250000
  trunc - leave : t=0.488281250000  *timestep=inf
main             : t=0.488281250000
  trunc - enter : t=0.976562500000  *timestep=inf  inst->tmain=0.488281250000  dT=0.488281250000
  trunc - 1st IF: t=0.976562500000  *timestep=inf
  trunc - main  : t=0.976562500000
  trunc - leave : t=0.976562500000  *timestep=inf
  trunc - enter : t=0.976562500000  *timestep=inf  inst->tmain=0.488281250000  dT=0.488281250000
  trunc - 1st IF: t=0.976562500000  *timestep=inf
  trunc - main  : t=0.976562500000
  trunc - leave : t=0.976562500000  *timestep=inf
main             : t=0.976562500000
  trunc - enter : t=1.464843750000  *timestep=inf  inst->tmain=0.976562500000  dT=0.488281250000
  trunc - 1st IF: t=1.464843750000  *timestep=inf
  trunc - main  : t=1.464843750000
  trunc - 2nd IF: t=1.464843750000  *timestep=0.001000000000
  trunc - leave : t=1.464843750000  *timestep=0.001000000000
  trunc - enter : t=0.977562500000  *timestep=inf  inst->tmain=0.976562500000  dT=0.001000000000
  trunc - 1st IF: t=0.977562500000  *timestep=inf
  trunc - main  : t=0.977562500000
  trunc - leave : t=0.977562500000  *timestep=inf
  trunc - enter : t=0.977562500000  *timestep=inf  inst->tmain=0.976562500000  dT=0.001000000000
  trunc - 1st IF: t=0.977562500000  *timestep=inf
  trunc - main  : t=0.977562500000
  trunc - leave : t=0.977562500000  *timestep=inf
main             : t=0.977562500000
  trunc - enter : t=0.979562500000  *timestep=inf  inst->tmain=0.977562500000  dT=0.002000000000
  trunc - 1st IF: t=0.979562500000  *timestep=inf
  trunc - main  : t=0.979562500000
  trunc - leave : t=0.979562500000  *timestep=inf
  trunc - enter : t=0.979562500000  *timestep=inf  inst->tmain=0.977562500000  dT=0.002000000000
  trunc - 1st IF: t=0.979562500000  *timestep=inf
  trunc - main  : t=0.979562500000
  trunc - leave : t=0.979562500000  *timestep=inf
main             : t=0.979562500000
  trunc - enter : t=0.983562500000  *timestep=inf  inst->tmain=0.979562500000  dT=0.004000000000
  trunc - 1st IF: t=0.983562500000  *timestep=inf
  trunc - main  : t=0.983562500000
  trunc - leave : t=0.983562500000  *timestep=inf
main             : t=0.983562500000
  trunc - enter : t=0.991562500000  *timestep=inf  inst->tmain=0.983562500000  dT=0.008000000000
  trunc - 1st IF: t=0.991562500000  *timestep=inf
  trunc - main  : t=0.991562500000
  trunc - leave : t=0.991562500000  *timestep=inf
main             : t=0.991562500000
  trunc - enter : t=1.007562500000  *timestep=inf  inst->tmain=0.991562500000  dT=0.016000000000
```

@t=0, trunc() tested next step and no state change is detected

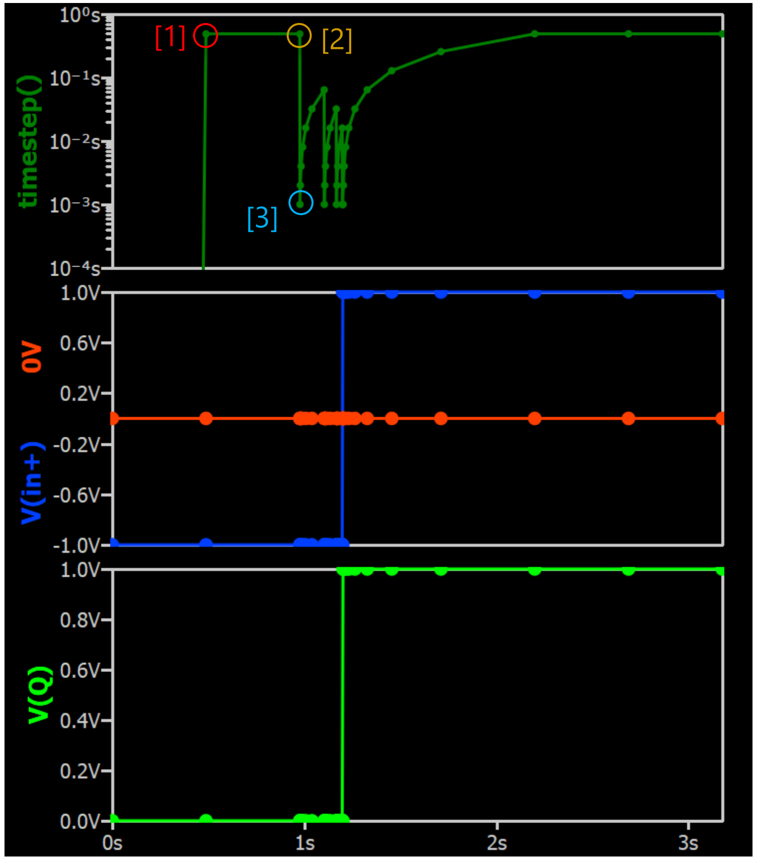
[1] Trunc() executed two times between each main() if no *timestep is assigned

[3] if(*tmp!=inst) is FALSE, no *timestep is assigned → next sim time in main() is this future test time

[2] if(*tmp!=inst) is TRUE, *timestep=TTOL t=1.4648... is future test time in Trunc()

[3] t=1.4648... is ignored after *timestep set to TTOL → forces next sim time t=0.9765...+ttol=0.9775...

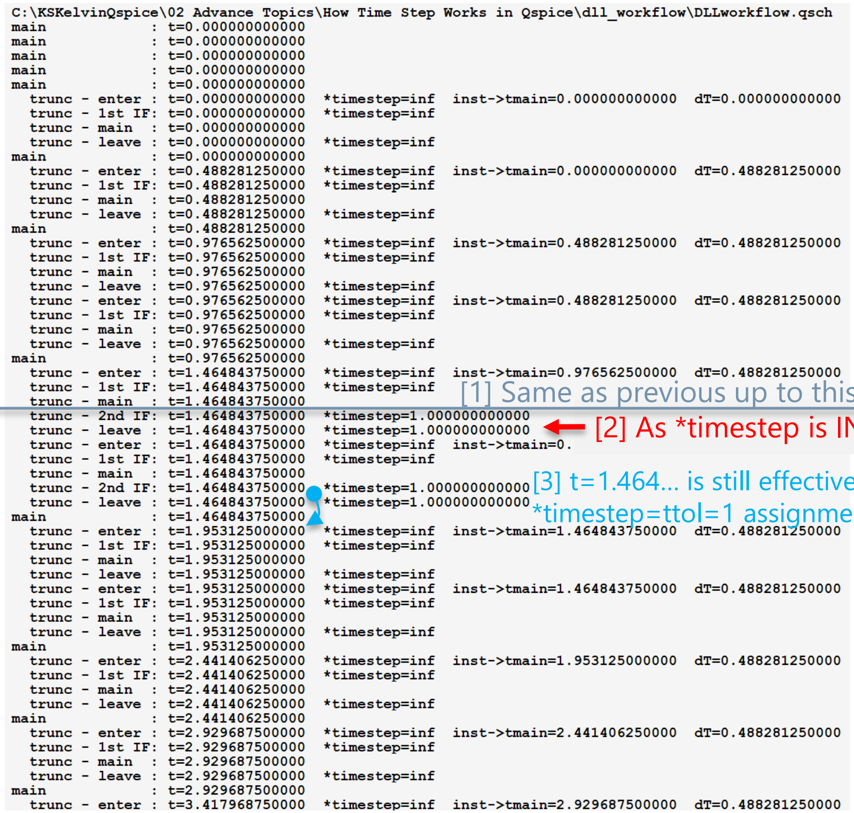
[4] Timestep multiplier by two in subsequent TTOL event



How timestep works? (DLL Ø-Device) : TTOL=1 in Trunc()

Qspice : \DLL\dll_workflow\DLLworkflow.qsch

** TTOL is larger than timestep using in this simulation

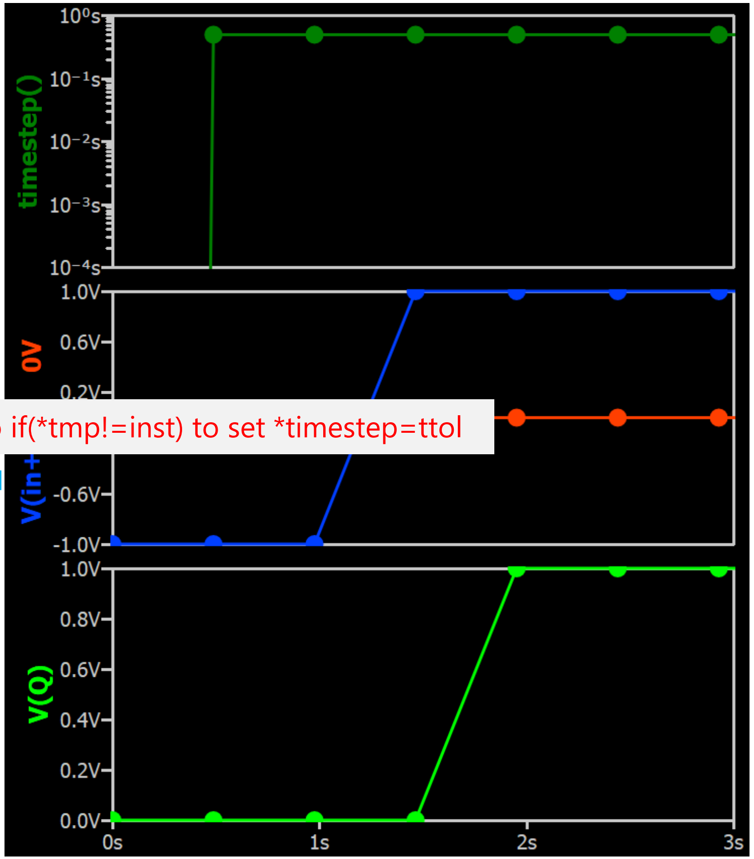


```
C:\KSKelvinQspice\02 Advance Topics\How Time Step Works in Qspice\dll_workflow\DLLworkflow.qsch
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
  trunc - enter : t=0.000000000000  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000000000
  trunc - 1st IF: t=0.000000000000  *timestep=inf
  trunc - main  : t=0.000000000000
  trunc - leave : t=0.000000000000  *timestep=inf
main            : t=0.000000000000
  trunc - enter : t=0.488281250000  *timestep=inf  inst->tmain=0.000000000000  dT=0.488281250000
  trunc - 1st IF: t=0.488281250000  *timestep=inf
  trunc - main  : t=0.488281250000
  trunc - leave : t=0.488281250000  *timestep=inf
main            : t=0.488281250000
  trunc - enter : t=0.976562500000  *timestep=inf  inst->tmain=0.488281250000  dT=0.488281250000
  trunc - 1st IF: t=0.976562500000  *timestep=inf
  trunc - main  : t=0.976562500000
  trunc - leave : t=0.976562500000  *timestep=inf
  trunc - enter : t=0.976562500000  *timestep=inf  inst->tmain=0.488281250000  dT=0.488281250000
  trunc - 1st IF: t=0.976562500000  *timestep=inf
  trunc - main  : t=0.976562500000
  trunc - leave : t=0.976562500000  *timestep=inf
main            : t=0.976562500000
  trunc - enter : t=1.464843750000  *timestep=inf  inst->tmain=0.976562500000  dT=0.488281250000
  trunc - 1st IF: t=1.464843750000  *timestep=inf
  trunc - main  : t=1.464843750000
  trunc - 2nd IF: t=1.464843750000  *timestep=1.000000000000
  trunc - leave : t=1.464843750000  *timestep=1.000000000000
  trunc - enter : t=1.464843750000  *timestep=inf  inst->tmain=0.
  trunc - 1st IF: t=1.464843750000  *timestep=inf
  trunc - main  : t=1.464843750000
  trunc - 2nd IF: t=1.464843750000  *timestep=1.000000000000
  trunc - leave : t=1.464843750000  *timestep=1.000000000000
main            : t=1.464843750000
  trunc - enter : t=1.953125000000  *timestep=inf  inst->tmain=1.464843750000  dT=0.488281250000
  trunc - 1st IF: t=1.953125000000  *timestep=inf
  trunc - main  : t=1.953125000000
  trunc - leave : t=1.953125000000  *timestep=inf
  trunc - enter : t=1.953125000000  *timestep=inf  inst->tmain=1.464843750000  dT=0.488281250000
  trunc - 1st IF: t=1.953125000000  *timestep=inf
  trunc - main  : t=1.953125000000
  trunc - leave : t=1.953125000000  *timestep=inf
main            : t=1.953125000000
  trunc - enter : t=2.441406250000  *timestep=inf  inst->tmain=1.953125000000  dT=0.488281250000
  trunc - 1st IF: t=2.441406250000  *timestep=inf
  trunc - main  : t=2.441406250000
  trunc - leave : t=2.441406250000  *timestep=inf
main            : t=2.441406250000
  trunc - enter : t=2.929687500000  *timestep=inf  inst->tmain=2.441406250000  dT=0.488281250000
  trunc - 1st IF: t=2.929687500000  *timestep=inf
  trunc - main  : t=2.929687500000
  trunc - leave : t=2.929687500000  *timestep=inf
main            : t=2.929687500000
  trunc - enter : t=3.417968750000  *timestep=inf  inst->tmain=2.929687500000  dT=0.488281250000
```

[1] Same as previous up to this line

[2] As *timestep is INF, still goto if(*tmp!=inst) to set *timestep=ttol

[3] t=1.464... is still effective and ignored *timestep=ttol=1 assignment

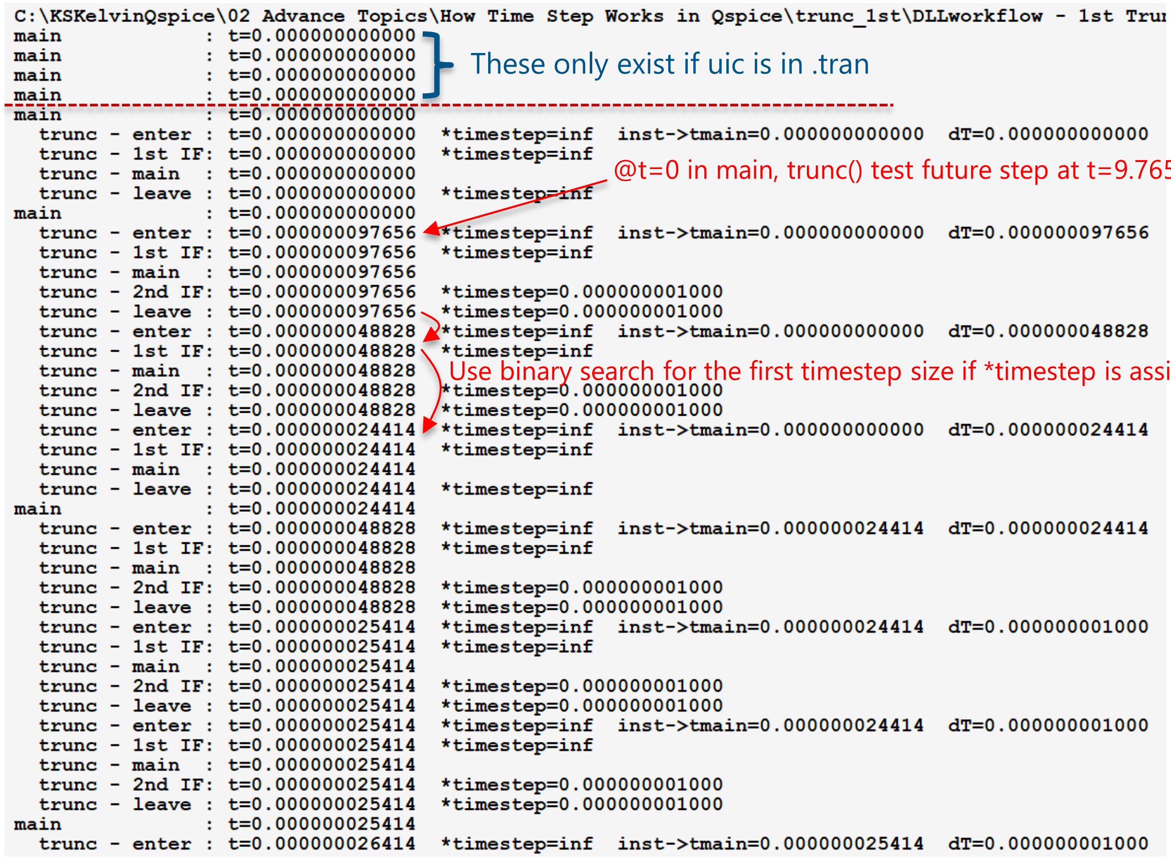


Conclusion

- Conclusion from this Study
  - Timestep in Qspice is adaptive, which determine by
    - .option maxstep
    - .option max1ststep : the first timestep in .tran
    - Tstart and Tstop in .tran
    - Devices with timestep control ability (e.g. Voltage source, Switch, ¥-Device)
    - Return of MaxExtStepSize() or *timestep in Trunc() in DLL block (Ø-Device)
  - Hypothetical timestep : Qspice never goes back in time during simulation, but it examines the future steps to determine if the timestep needs to reduce
  - Qspice devices can utilize output state changes with if(*tmp!=inst) OR whatever you do to force *timestep to change within Trunc().
  - *timestep in Trunc() is always equal +INF when just enter Trunc(). It seems if condition (*timestep>TTOL) is always TRUE in Trunc()
    - *timestep in Trunc() not actual timestep itself but a determination factor for actual timestep
  - If trunc() exit with *timestep change, next simulation time will force to increase by this amount of change (but with exception that *timestep > ~(Tstop-Tstart)/1000)
  - The actual timestep will be increased by a factor of 2 in each subsequent step, until
    - Re-trigger of if(*tmp!=inst) OR
    - Reach ~(Tstop-Tstart)/1000 OR
    - Timestep limit from other devices

Appendix : How timestep works? (DLL Ø-Device) – First Step in DLL

Qspice : \DLL\trunc_1stTrunc\DLLworkflow - 1st Trunc.qsch



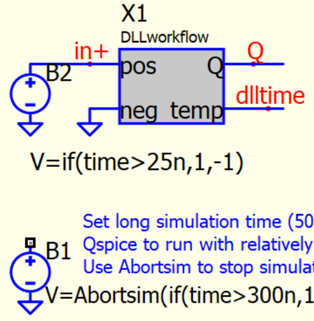
```
C:\KSKelvinQspice\02 Advance Topics\How Time Step Works in Qspice\trunc_1st\DLLworkflow - 1st Tru
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
main            : t=0.000000000000
  trunc - enter : t=0.000000000000  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000000000
  trunc - 1st IF: t=0.000000000000  *timestep=inf
  trunc - main  : t=0.000000000000
  trunc - leave : t=0.000000000000  *timestep=inf
main            : t=0.000000000000
  trunc - enter : t=0.000000097656  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000097656
  trunc - 1st IF: t=0.000000097656  *timestep=inf
  trunc - main  : t=0.000000097656
  trunc - 2nd IF: t=0.000000097656  *timestep=0.000000001000
  trunc - leave : t=0.000000097656  *timestep=0.000000001000
  trunc - enter : t=0.000000048828  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000048828
  trunc - 1st IF: t=0.000000048828  *timestep=inf
  trunc - main  : t=0.000000048828
  trunc - 2nd IF: t=0.000000048828  *timestep=0.000000001000
  trunc - leave : t=0.000000048828  *timestep=0.000000001000
  trunc - enter : t=0.000000024414  *timestep=inf  inst->tmain=0.000000000000  dT=0.000000024414
  trunc - 1st IF: t=0.000000024414  *timestep=inf
  trunc - main  : t=0.000000024414
  trunc - leave : t=0.000000024414  *timestep=inf
main            : t=0.000000024414
  trunc - enter : t=0.000000048828  *timestep=inf  inst->tmain=0.000000024414  dT=0.000000024414
  trunc - 1st IF: t=0.000000048828  *timestep=inf
  trunc - main  : t=0.000000048828
  trunc - 2nd IF: t=0.000000048828  *timestep=0.000000001000
  trunc - leave : t=0.000000048828  *timestep=0.000000001000
  trunc - enter : t=0.000000025414  *timestep=inf  inst->tmain=0.000000024414  dT=0.000000001000
  trunc - 1st IF: t=0.000000025414  *timestep=inf
  trunc - main  : t=0.000000025414
  trunc - 2nd IF: t=0.000000025414  *timestep=0.000000001000
  trunc - leave : t=0.000000025414  *timestep=0.000000001000
  trunc - enter : t=0.000000025414  *timestep=inf  inst->tmain=0.000000024414  dT=0.000000001000
  trunc - 1st IF: t=0.000000025414  *timestep=inf
  trunc - main  : t=0.000000025414
  trunc - 2nd IF: t=0.000000025414  *timestep=0.000000001000
  trunc - leave : t=0.000000025414  *timestep=0.000000001000
main            : t=0.000000025414
  trunc - enter : t=0.000000026414  *timestep=inf  inst->tmain=0.000000025414  dT=0.000000001000
```

These only exist if uic is in .tran

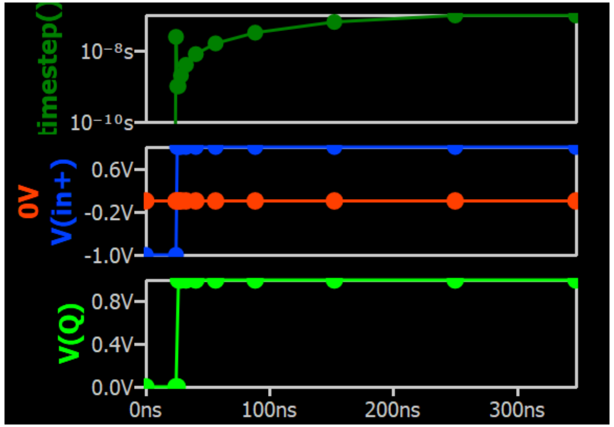
@t=0 in main, trunc() test future step at t=9.7656e-8

Use binary search for the first timestep size if *timestep is assigned

```
.plot V(Q)            .tran 100n*1000 ;uic
.plot V(in+) 0V       .func timestep() time-V(dlltime)
.plot timestep()      .option Max1stStep=10n
```



Set long simulation time (500s) to force Qspice to run with relatively loose timestep
Use Abortsim to stop simulation at 3s



Appendix A

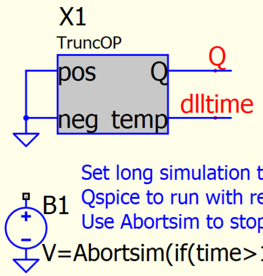
*timestep in TRUNC

*timestep=TTOL in Trunc()

Qspice : \Appendix\trunc_operation\TruncOP.qsch

```
.tran 500 ;uic
.func timestep() time-V(dlltime)
.plot timestep()
```

```
; disable first-step limit
.option MAX1STSTEP=1e308
```



Set long simulation time (500s) to force Qspice to run with relatively loose timestep
Use Abortsim to stop simulation at 3s

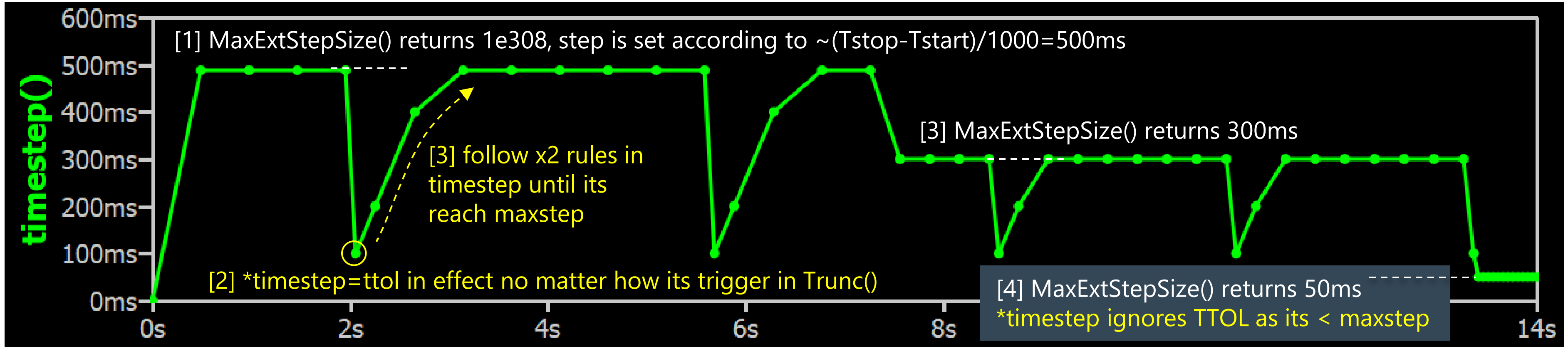
```
// Implement module evaluation code here:
   temp = t;
   inst->tmain = t;
   inst->counter++;
   if (inst->inTrunc == 0)
   {
      display("main             : t=%.12f\r\n",t);
   }
   else
      display("  trunc - main   : t=%.12f\r\n",t);
}

extern "C" __declspec(dllexport) double MaxExtStepSize
{
   if(inst->counter > 50)
      return 5e-2;
   else if (inst->counter > 25)
      return 3e-1;
   else
      return 1e308; // implement a good choice of max
}
```

Change maxstep at different count

Test concept

- A counter in module evaluation code
- Whatever counter%10==0, trigger *timestep=ttol in Trunc()

```
extern "C" __declspec(dllexport) void Trunc(
{ // limit the timestep to a tolerance if th
   const double ttol = 1e-1;

// if(tmp != *inst) // implement
//    *timestep = ttol;
   if(inst->counter%10==0)
   {
      *timestep = ttol;
      display("  trunc - 2nd IF:
      inst->counter++;
   }
```



[1] MaxExtStepSize() returns 1e308, step is set according to ~(Tstop-Tstart)/1000=500ms

[2] *timestep=ttol in effect no matter how its trigger in Trunc()

[3] follow x2 rules in timestep until its reach maxstep

[3] MaxExtStepSize() returns 300ms

[4] MaxExtStepSize() returns 50ms
*timestep ignores TTOL as its < maxstep

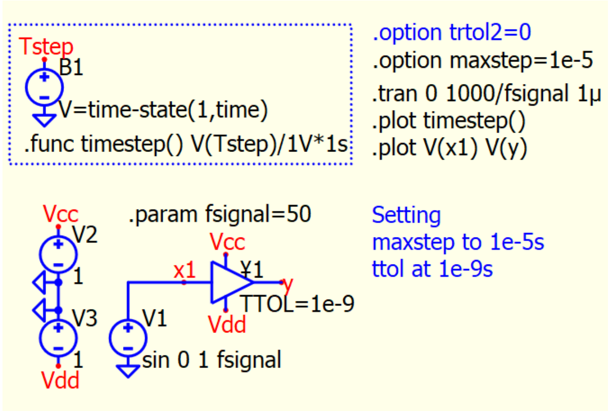
Appendix B

Simulation with Long Run

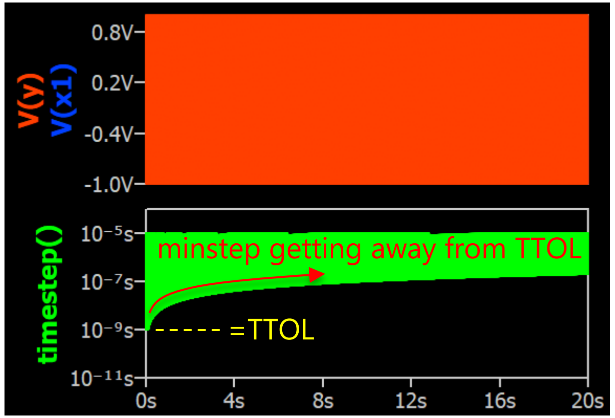
timestep÷time limitation

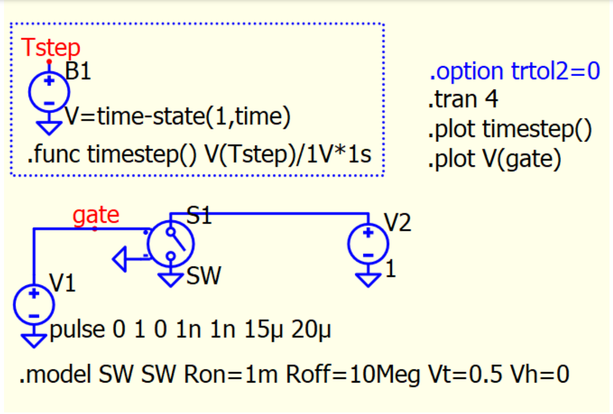
Qspice : \Appendix\long sim\timestep-ttol.qsch | timestep-maxstep.qsch

- timestep÷time limitation
  - By Mike Engerhardt, Qspice cannot go to tiny time steps are latetimes because it might have not enough resolution (i.e. timestep÷time has to be several orders of magnitude larger than 1e-15 so that it can do the math)
  - First example, a device with TTOL is used in a long simulation run, and the minimum timestep gradually increases over time
  - Second example, a V-source and switch with default timectrl TTOL in a long simulation run, and the maxstep keeps changing throughout the simulation

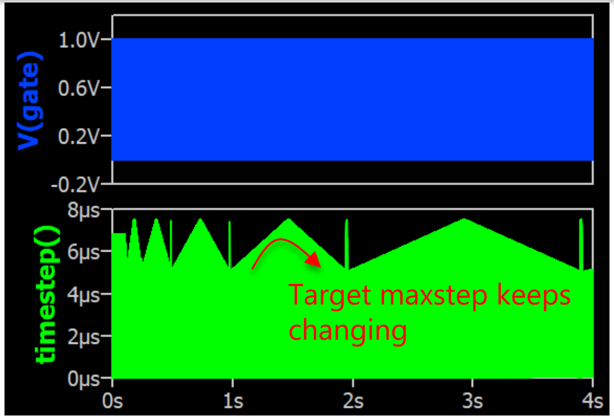


```
.func timestep() V(Tstep)/1V*1s
.option trtol2=0
.option maxstep=1e-5
.tran 0 1000/fsignal 1µ
.plot timestep()
.plot V(x1) V(y)
.param fsignal=50
```





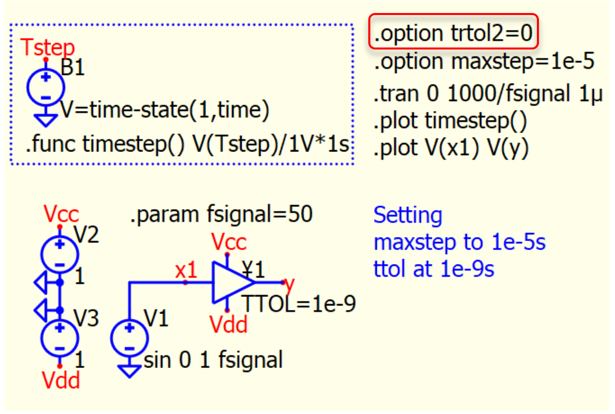
```
.func timestep() V(Tstep)/1V*1s
.option trtol2=0
.tran 4
.plot timestep()
.plot V(gate)
.model SW SW Ron=1m Roff=10Meg Vt=0.5 Vh=0
```



TRTOL2 in timestep÷time limited situation

Qspice : \Appendix\long sim\timestep-ttol.qsch | timestep-maxstep.qsch

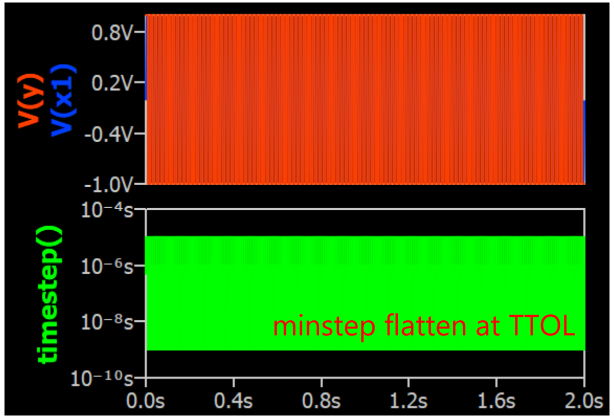
- TRTOL2
  - Trtol2 : Another dimensionless truncation error guidance
  - **Default TRTOL2=1e-8**
  - It can observe that by focusing TRTOL2=0 can flatten maxstep and minstep along simulation
  - Quote from Mike Engerhardt, TRTOL2 is Qspice option to prevents the simulation from crashing by going to a smaller timestep that is actually required

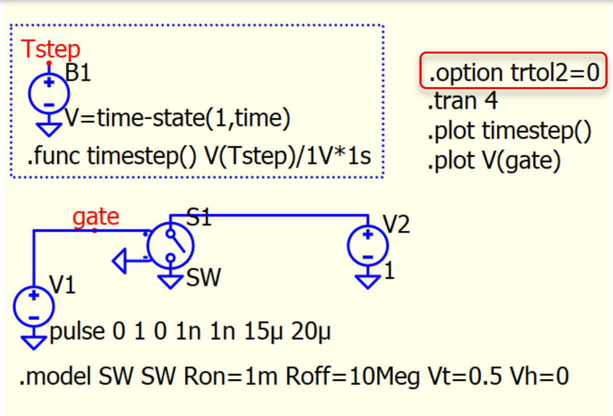


```
.option trtol2=0
.option maxstep=1e-5
.tran 0 1000/fsignal 1µ
.plot timestep()
.plot V(x1) V(y)
```

```
V=time-state(1,time)
.func timestep() V(Tstep)/1V*1s
.param fsignal=50
TTOL=1e-9
sin 0 1 fsignal
```

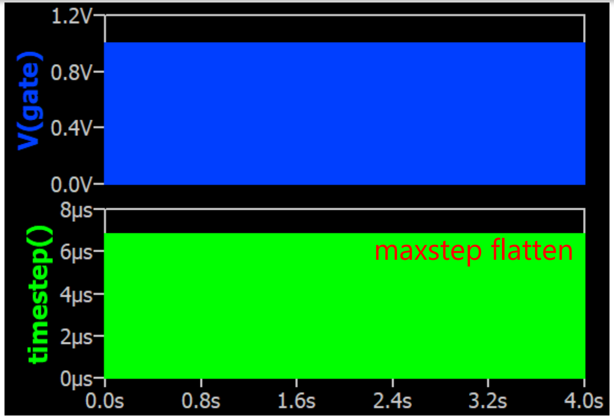
Setting maxstep to 1e-5s ttol at 1e-9s





```
.option trtol2=0
.tran 4
.plot timestep()
.plot V(gate)
```

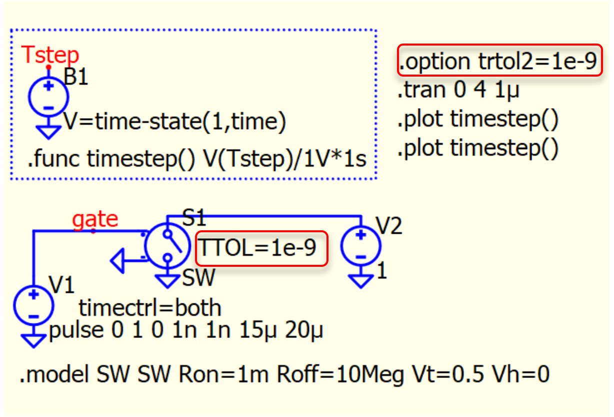
```
V=time-state(1,time)
.func timestep() V(Tstep)/1V*1s
pulse 0 1 0 1n 1n 15µ 20µ
.model SW SW Ron=1m Roff=10Meg Vt=0.5 Vh=0
```



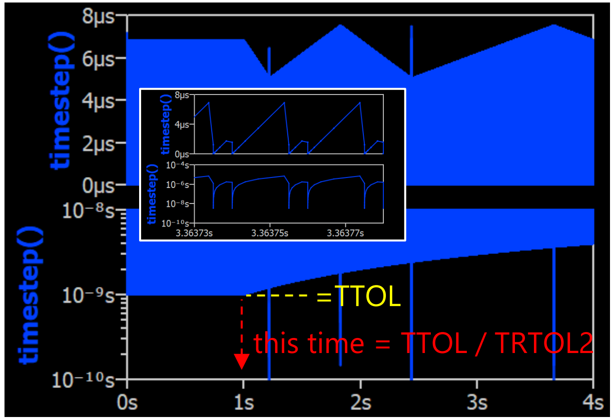
Study of TRTOL2

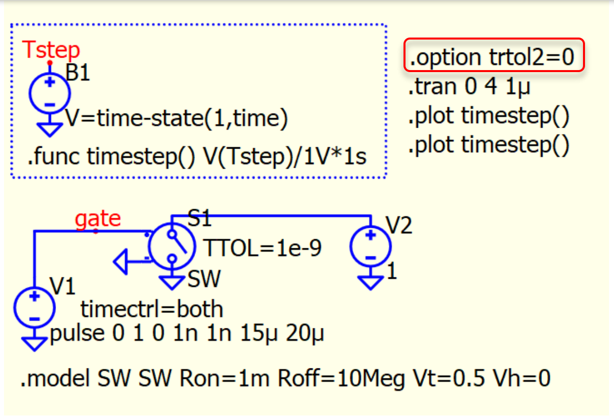
Qspice : \Appendix\long sim\timestep-trtol2.qsch

- Study of TRTOL2
  - This simulation setup with V-source timectrl and TTOL both works together
  - It is observed that the timestep starts failing from TTOL from **simulation time = TTOL / TRTOL2**
  - Therefore, forcing TRTOL2=0 will extend this time to infinite and maxstep and minstep both flatten across entire simulation

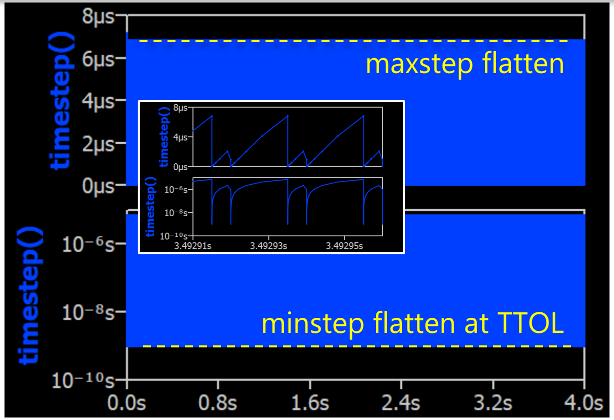


```
V=time-state(1,time)
.func timestep() V(Tstep)/1V*1s
.option trtol2=1e-9
.tran 0 4 1µ
.plot timestep()
.plot timestep()
TTOL=1e-9
timectrl=both
pulse 0 1 0 1n 1n 15µ 20µ
.model SW SW Ron=1m Roff=10Meg Vt=0.5 Vh=0
```





```
V=time-state(1,time)
.func timestep() V(Tstep)/1V*1s
.option trtol2=0
.tran 0 4 1µ
.plot timestep()
.plot timestep()
TTOL=1e-9
timectrl=both
pulse 0 1 0 1n 1n 15µ 20µ
.model SW SW Ron=1m Roff=10Meg Vt=0.5 Vh=0
```



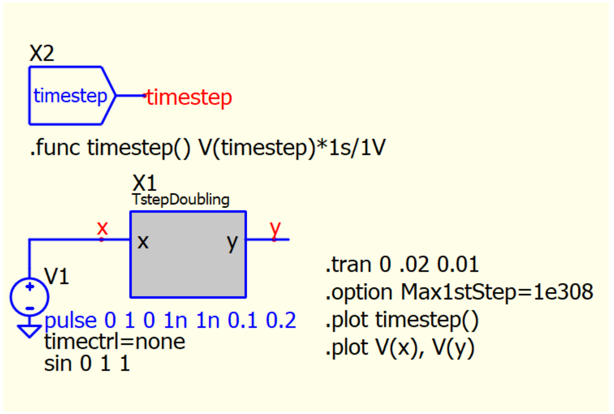
Appendix C

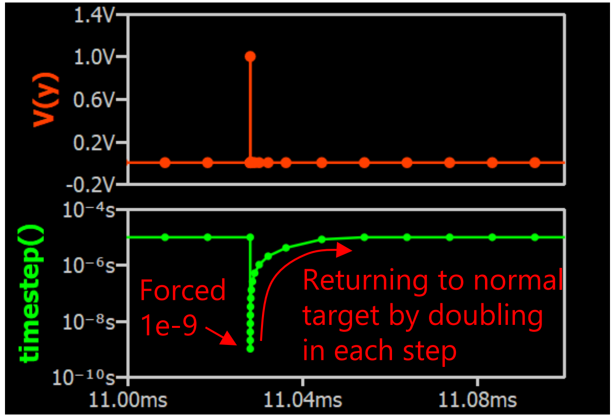
Timestep Doubling in Qspice

Timestep Doubling in MaxExtStepSize() function

Qspice : \Appendix\timestep_doubling\TstepDoubling.qsch

- Timestep Doubling
  - Whether you use Trunc() or MaxExtStepSize() to set the timestep, it will trigger the timestep doubling algorithm to adjust the timestep back to the desired step size
  - This example forces MaxExtStepSize() to set a timestep of 1e-9 every 0.001s. It confirms that when the timestep is no longer forced to be 1e-9, it will return to the desired step size using a timestep doubling strategy





```
struct sTSTEPDOUBLING
{
  // declare the structure here
  float MaxStepTtol;
  float lastT;
  bool MaxStepTrig;
};

extern "C" __declspec(dllexport) void tstepdoubling(str
{
   double  x = data[0].d; // input
   double &y = data[1].d; // output

   if(!*opaque)
   {
      *opaque = (struct sTSTEPDOUBLING *) malloc(sizeof
      bzero(*opaque, sizeof(struct sTSTEPDOUBLING));
   }
   struct sTSTEPDOUBLING *inst = *opaque;

// Implement module evaluation code here:
   y = inst->MaxStepTrig;

   inst->MaxStepTtol = 1e-9;
   inst->MaxStepTrig = 0;
   if ( t - inst->lastT > 0.001 )
   {
      inst->MaxStepTrig = 1;
      inst->lastT = t;
   }
}

extern "C" __declspec(dllexport) double MaxExtStepSize(
{
   if (inst->MaxStepTrig)
      return inst->MaxStepTtol;
   else
      return 1e308; // implement a good choice of max t
}
```

About every 0.001s, set MaxStepTrig flag

Return timestep as MaxStepTtol (1e-9) from MaxExtStepSize() function

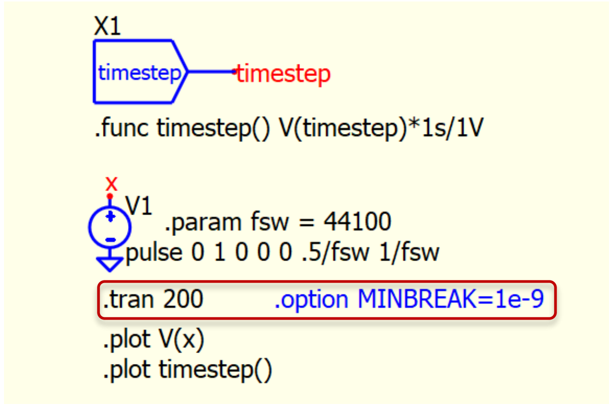
Appendix D

MinBreak in Timectrl

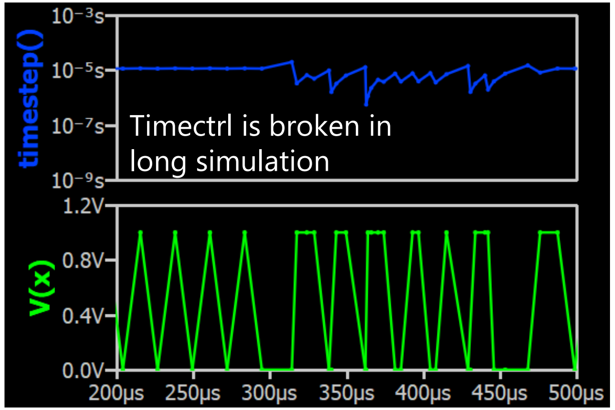
MinBreak in Timectrl of V-source

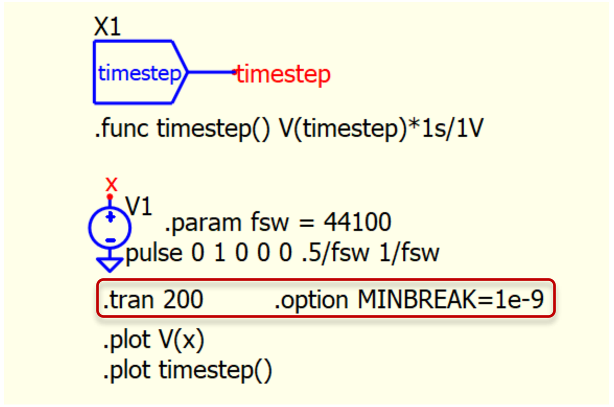
Qspice : \Appendix\timestep_doubling\Option - Minbreak (.tran 200).qsch

- MinBreak in Timectrl
  - In long simulation run, timestep control of V-source may be broken in long simulation run
  - In this situation, several approaches can be considered
    - Add a 1pF capacitor in parallel to V-source, to limit timestep in slew
    - Add maxstep to limit maximum step
    - Add TTOL-device for TTOL timestep scheme
    - Add .option minbreak for minimum timestep in breakpoints for V-source

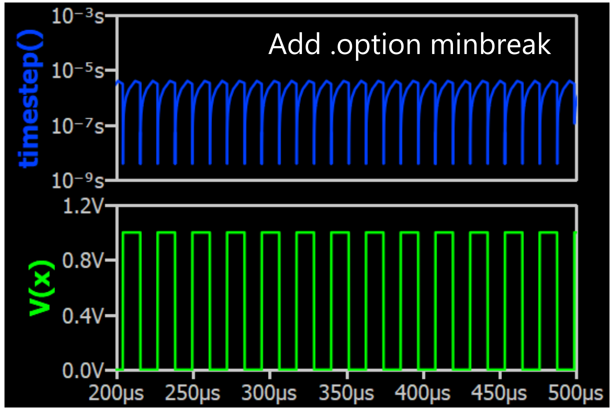


```
.func timestep() V(timestep)*1s/1V
.param fsw = 44100
pulse 0 1 0 0 0 .5/fsw 1/fsw
.tran 200          .option MINBREAK=1e-9
.plot V(x)
.plot timestep()
```





```
.func timestep() V(timestep)*1s/1V
.param fsw = 44100
pulse 0 1 0 0 0 .5/fsw 1/fsw
.tran 200          .option MINBREAK=1e-9
.plot V(x)
.plot timestep()
```



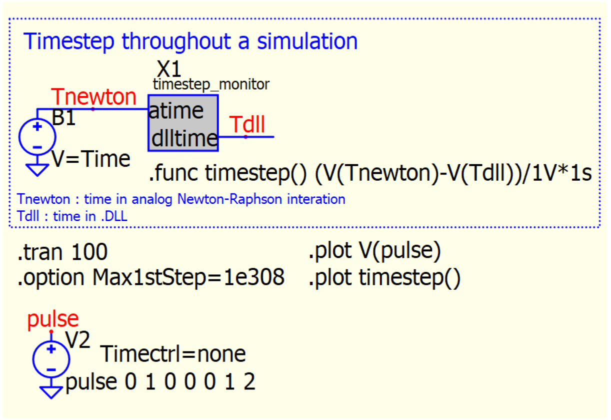
Appendix E

Timestep with DLL

Appendix : How Time Step Works in Qspice – TimeStep Monitor (DLL)

Qspice : \Appendix\timestep_monitor DLL\timestep_monitor.qsch

- Timestep Monitor
  - Simulation Time of Qspice can be found as
    - Time in analog Newton-Raphson interation: **Time**
    - DLL Time : **t** in DLL block
  - DLL Time always one step behind Analog Time
    - Therefore, different of analog time and DLL time is simulation timestep
  - Method to read timestep
    - Cpp block with dlltime=atime, where atime is analog time and dlltime is dll time delayed by one timestep
    - Calculate different between analog and DLL time for timestep



```
.func timestep() (V(Tnewton)-V(Tdll))/1V*1s
```

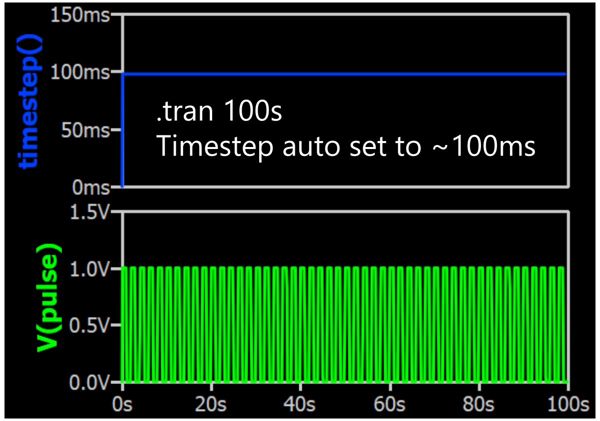
Tnewton : time in analog Newton-Raphson interation
Tdll : time in .DLL

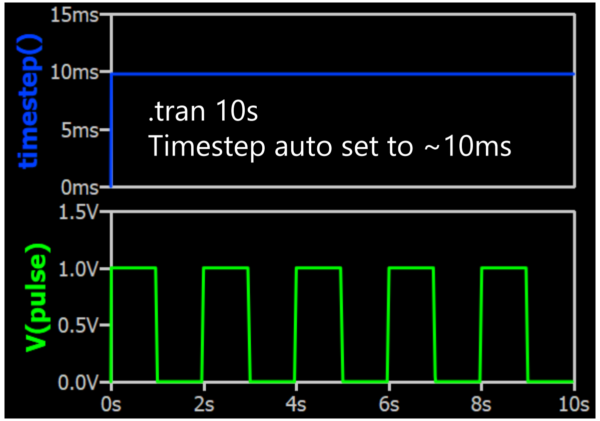
```
.tran 100
.option Max1stStep=1e308
.plot V(pulse)
.plot timestep()
```

- Code

```
// Implement module evaluation code here:
   dlltime = atime;
```

  - only 1 line of code to pass time from input to output
  - as dll time (output) is always one step delay of input, different between dlltime and atime is timestep





Appendix F

DLL Workflow and Timestep Study

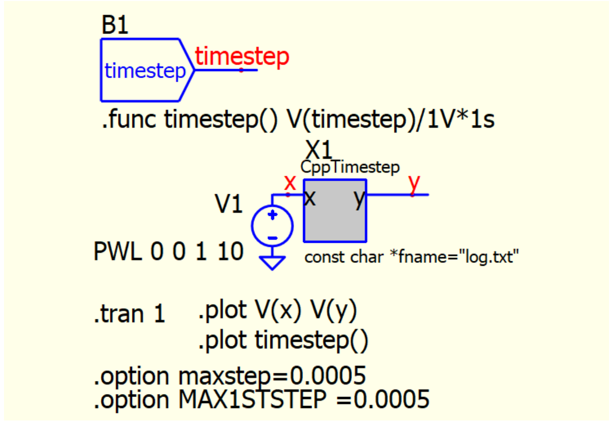
Explanation of DLL operation flow by Rdunn

- Explanation quote from Robert (Rdunn)
  - After QSpice does some initialization, the simulation cycle works like this:
    1. Call MaxExtStepSize(). QSpice selects a next timepoint/step not greater than the returned step size.
    2. [If Trunc() present] Call Trunc() with the next hypothetical timestep values. Keep calling Trunc() until the value returned in *timestep would no longer reduce the next timestep.
    3. Call the evaluation function with the final timepoint value. Commit the results to the simulation data.
    4. If not finished, goto 1.
  - So, if Trunc() is not present, calculations in (1) and (3) are executed only once per simulation data point. There is no speed advantage to doing a calculation in MaxExtStepSize() vs the evaluation function.
  - On the other hand, if Trunc() is present, things change. The canonical Trunc() function calls the evaluation function with a hypothetical timepoint/step. The eval code is executed at least twice (at least once in (2) and exactly once in (3)). So the calculations are done multiple times. If you are choosing between putting code in MaxExtStepSize() vs the eval function, the former guarantees the calculation is done only once per timestep whether or not Trunc() is implemented.
  - Alternatively (and more generally), if you have eval function code that you don’t want executed when called from Trunc(), you can test the ForKeeps flag. QSpice clears the flag before (2) and sets it before (3).

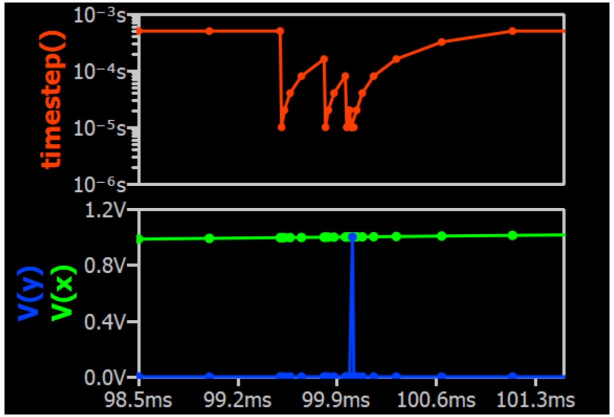
DLL Workflow

Qspice : \Appendix\DLL Workflow\CppTimestep.qsch

- Purpose
  - **CppTimestep.cpp** has been created to log the process flow into a file named log.txt. It includes variables t for time and dT for timestep to analyze the workflow of the DLL
  - The code consists of functions such as evaluation function (named as CppTimestep() in this example), MaxExtStepSize(), and Trunc()
  - This Cpp block is setup to trigger a state change at 100ms where input signal is crossing 1V, this is to demonstrate effect of *timestep forced to ttol in Trunc()



```
.func timestep() V(timestep)/1V*1s
.tran 1
.plot V(x) V(y)
.plot timestep()
.option maxstep=0.0005
.option MAX1STSTEP =0.0005
```



DLL Workflow

Qspice : \Appendix\DLL Workflow\Fixed TTOL Timestep\CppTimestep.qsch

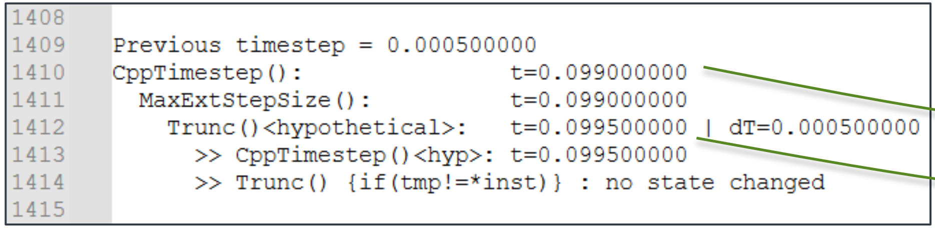
```
struct sCPPTIMESTEP
{
  // declare the structure here
  FILE *fptr;      // File pointer for logging timestep data
  bool init;       // Initialization flag
  bool inTrunc;    // Trunc() flag

  double lastX;    // Previous input value (for edge detection)
  double lastT;    // Previous simulation time
  bool Q;          // Current output state
};

extern "C" __declspec(dllexport) void cpptimestep(struct sCPPTIMESTEP **opaque, double t, ur
{
   double       x     = data[0].d   ; // input
   const char * fname = data[1].str; // input parameter
   double      &y     = data[2].d   ; // output

   if(!*opaque)
   {
      *opaque = (struct sCPPTIMESTEP *) malloc(sizeof(struct sCPPTIMESTEP));
      bzero(*opaque, sizeof(struct sCPPTIMESTEP));
   }
   struct sCPPTIMESTEP *inst = *opaque;

// Implement module evaluation code here:

   // First-time initialization - open log file
   if (!inst->init){
      inst->fptr = fopen(fname,"w");
      inst->init = true;
   }

   double dT = t - inst->lastT;  // Calculate time since last evaluation

   // Log current timestep information
   if (!inst->inTrunc){
      fprintf(inst->fptr,"\n");
      fprintf(inst->fptr,"Previous timestep = %.9f\n",t-inst->lastT);
      fprintf(inst->fptr,"CppTimestep():                    t=%.9f\n",t);
   }
   else
      fprintf(inst->fptr,"        >> CppTimestep()<hyp>:  t=%.9f\n",t);

   // Detect rising edge (0->1 transition) and generate pulse
   y = 0;
   if( inst->lastX < 1 & x >= 1)
      y = 1;

   // Store current state for next evaluation
   inst->lastT = t;
   inst->lastX = x;
   inst->Q = y;
}
```

```
extern "C" __declspec(dllexport) double MaxExtStepSize(struct sCPPTIMESTEP *inst, double t)
{
   fprintf(inst->fptr,"  MaxExtStepSize():          t=%.9f\n",t);
   return 1e308; // implement a good choice of max timestep size that depends on struct sCPI
}

extern "C" __declspec(dllexport) void Trunc(struct sCPPTIMESTEP *inst, double t, union uData
{ // limit the timestep to a tolerance if the circuit causes a change in struct sCPPTIMESTEI
   const double ttol = 1e-5; // 1ns default tolerance
   inst->inTrunc = true;
   fprintf(inst->fptr,"    Trunc()<hypothetical>:    t=%.9f | dT=%.9f\n",t,t-inst->lastT);
   if(*timestep > ttol)
   {
      struct sCPPTIMESTEP tmp = *inst;
      cpptimestep(&(&tmp), t, data);

      // Check if output state would change with this timestep
      if(tmp.Q != inst->Q){
         *timestep = ttol;    // Reduce timestep to tolerance if change detected
         fprintf(inst->fptr,"        >> Trunc() {if(tmp!=*inst)} >> state has changed\n");
         fprintf(inst->fptr,"        >>                         *timestep = ttol=%.9f\n",ttol);
      }else{
         fprintf(inst->fptr,"        >> Trunc() {if(tmp!=*inst)} : no state changed\n");
      }
   }

   inst->inTrunc = false;  // Reset Trunc() flag
}

extern "C" __declspec(dllexport) void Destroy(struct sCPPTIMESTEP *inst)
{
   free(inst);
}
```

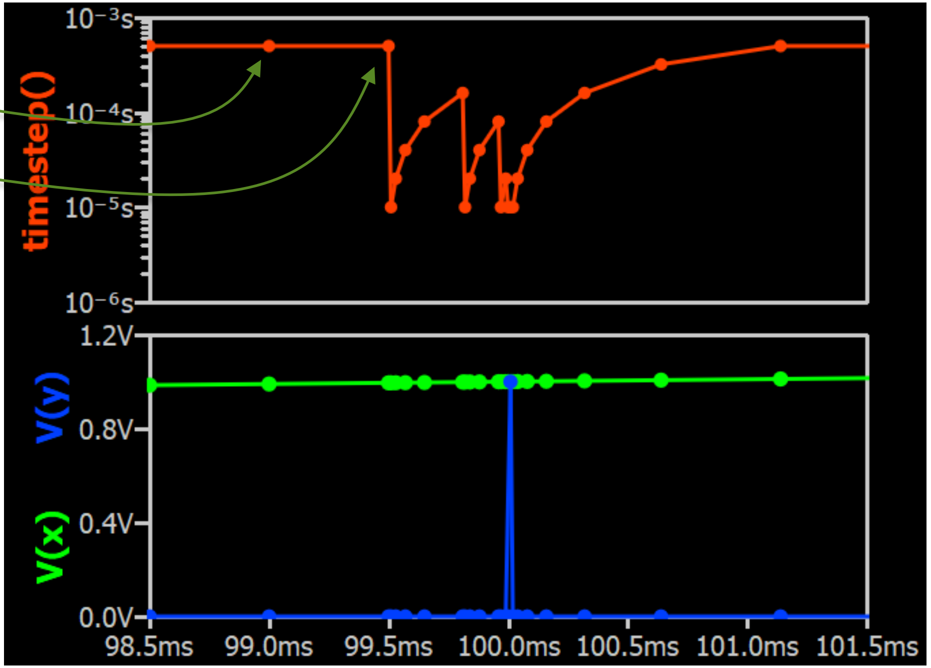
DLL Workflow

Analysis : Log.txt line#1408 to #1415

Flow : CppTimestep() >> MaxExtStepSize() >> Trunc() [may call multiple times]



```
1408
1409    Previous timestep = 0.000500000
1410    CppTimestep():               t=0.099000000
1411      MaxExtStepSize():          t=0.099000000
1412        Trunc()<hypothetical>:   t=0.099500000 | dT=0.000500000
1413          >> CppTimestep()<hyp>: t=0.099500000
1414          >> Trunc() {if(tmp!=*inst)} : no state changed
1415
```



#1410 : Execute the CppTimestep() function (evalution function).

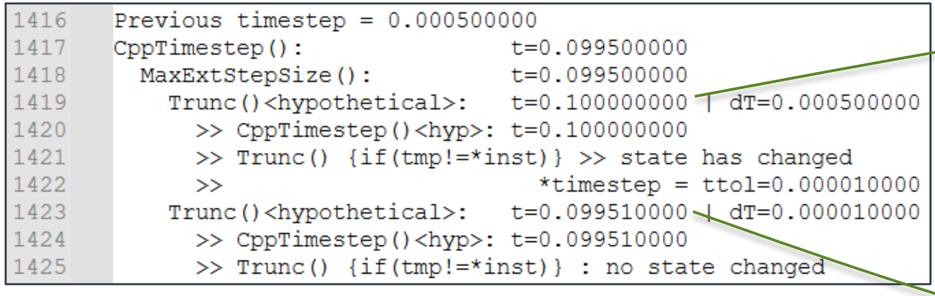
#1411 : Proceed to call MaxExtStepSize() to obtain the maxstep; the variable t in MaxExtStepSize() represents the current simulation time t

#1412 : Trunc() serves as a hypothetical time testing function, the variable t is hypothetical next step. It continues to run Trunc() until no smaller timestep (*timestep) is assigned, ultimately adopting the last assigned timestep for the next simulation time

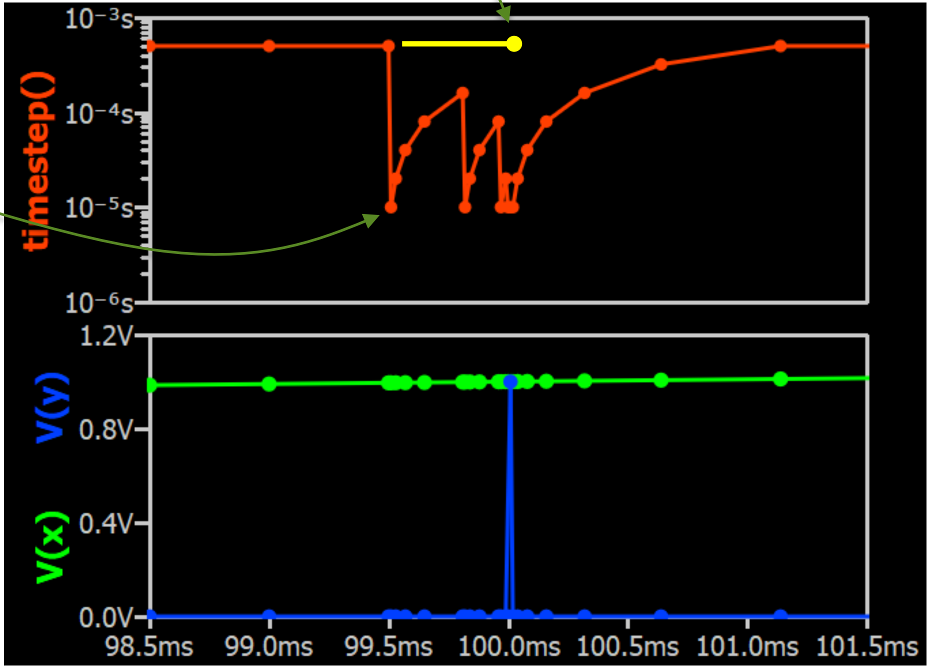
#1413 : Within Trunc(), it is essential to provide the hypothetical next step to the evaluation function (i.e., CppTimestep()); this necessitates hypothetical calls to CppTimestep()

DLL Workflow

Analysis : Log.txt line#1416 to #1425



```
1416    Previous timestep = 0.000500000
1417    CppTimestep():                t=0.099500000
1418      MaxExtStepSize():           t=0.099500000
1419        Trunc()<hypothetical>:    t=0.100000000 | dT=0.000500000
1420          >> CppTimestep()<hyp>: t=0.100000000
1421          >> Trunc() {if(tmp!=*inst)} >> state has changed
1422          >>                       *timestep = ttol=0.000010000
1423        Trunc()<hypothetical>:    t=0.099510000 | dT=0.000010000
1424          >> CppTimestep()<hyp>: t=0.099510000
1425          >> Trunc() {if(tmp!=*inst)} : no state changed
```

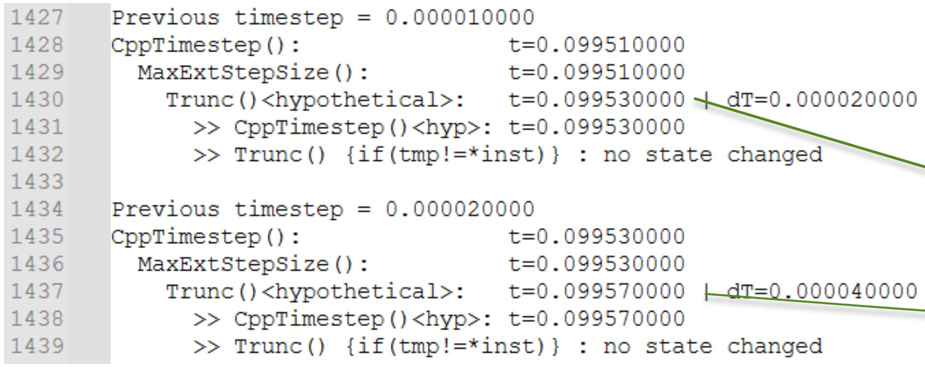


#1419-#1422 Within Trunc(), a hypothetical simulation time is evaluated. A state change is confirmed with the condition if(tmp != *inst), resulting in the assignment of *timestep = ttol. Consequently, the hypothetical timestep is reduced to ttol (1e-5 in this instance).

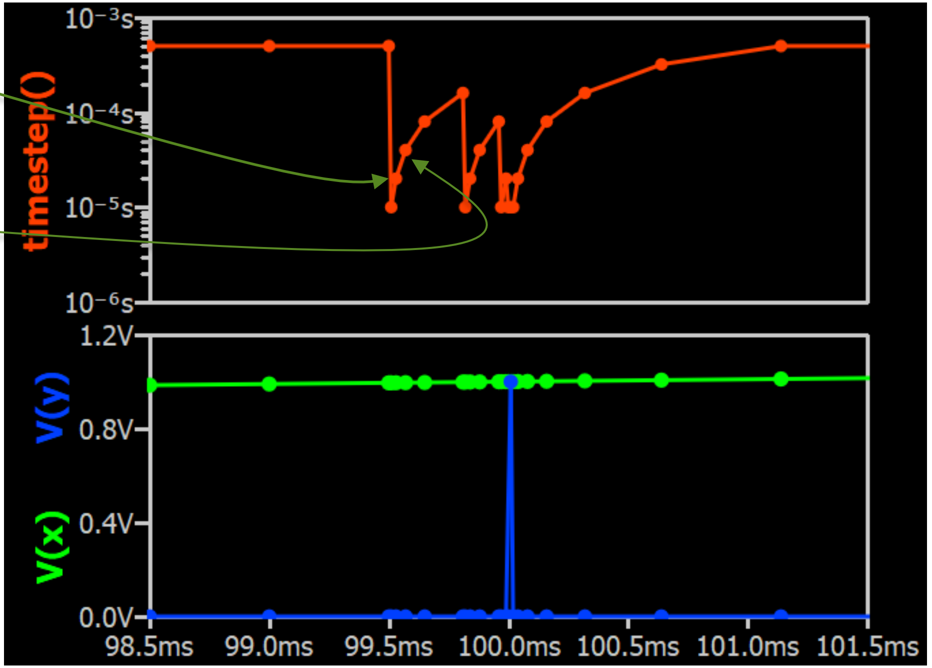
#1423-1425 Subsequent to reevaluating the hypothetical simulation time, the hypothetical time equates to the last simulation time plus *timestep (ttol in this case), yielding t = 0.995 + TTOL = 0.9951. Ultimately, no state change is identified, and the simulation proceeds with this as the next simulation step.

DLL Workflow

Analysis : Log.txt line#1416 to #1425



```
1427    Previous timestep = 0.000010000
1428    CppTimestep():                t=0.099510000
1429      MaxExtStepSize():           t=0.099510000
1430        Trunc()<hypothetical>:    t=0.099530000 | dT=0.000020000
1431          >> CppTimestep()<hyp>: t=0.099530000
1432          >> Trunc() {if(tmp!=*inst)} : no state changed
1433
1434    Previous timestep = 0.000020000
1435    CppTimestep():                t=0.099530000
1436      MaxExtStepSize():           t=0.099530000
1437        Trunc()<hypothetical>:    t=0.099570000 | dT=0.000040000
1438          >> CppTimestep()<hyp>: t=0.099570000
1439          >> Trunc() {if(tmp!=*inst)} : no state changed
```

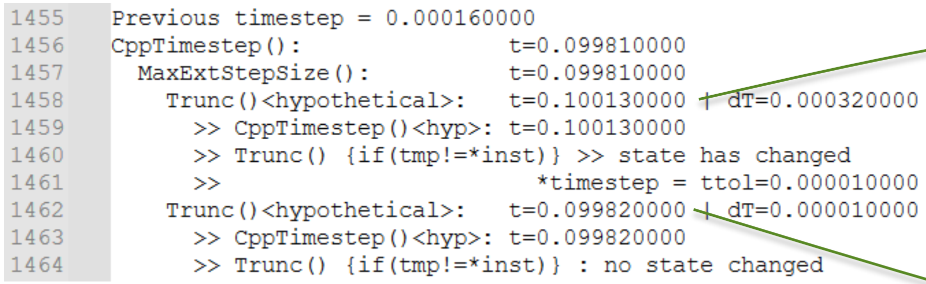


#1430-#1432 The current timestep is presently equal ttol (1e-5), which is smaller than the maxstep. In this scenario, the hypothetical timestep is established as double (x2) the previous timestep, resulting in 2e-5 in this illustration. During the hypothetical timestep examination, no state change is identified, thus there is no alteration in *timestep. This simulated time can successfully pass the test and advance to become the subsequent simulation step

#1437-#1439 Similarly as before, the process continues with doubling the timestep in the hypothetical test.

DLL Workflow

Analysis : Log.txt line#1458 to #1439

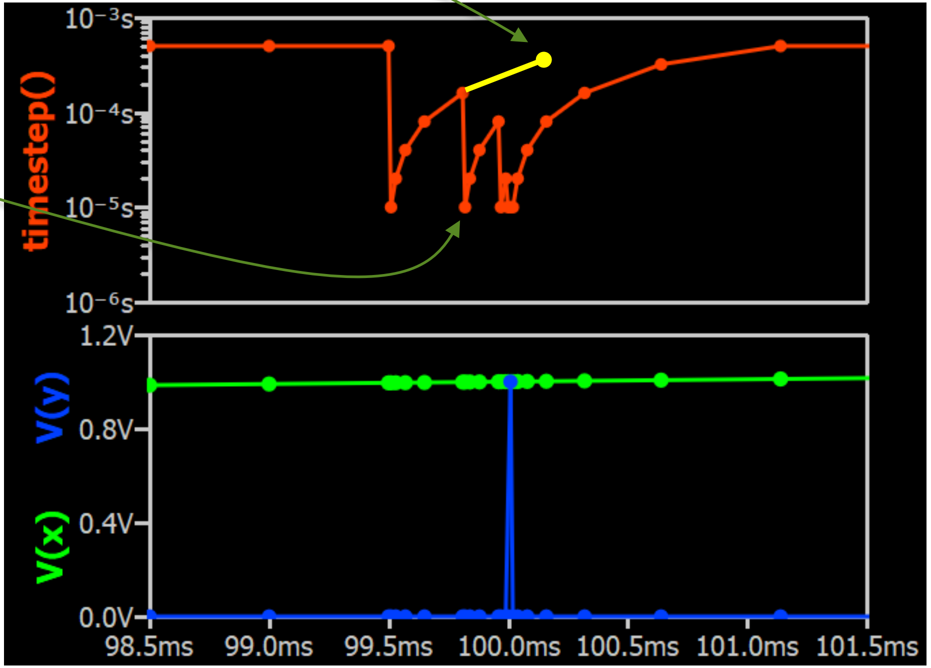


```
1455  Previous timestep = 0.000160000
1456  CppTimestep():               t=0.099810000
1457    MaxExtStepSize():          t=0.099810000
1458      Trunc()<hypothetical>:   t=0.100130000 | dT=0.000320000
1459        >> CppTimestep()<hyp>: t=0.100130000
1460        >> Trunc() {if(tmp!=*inst)} >> state has changed
1461        >>                      *timestep = ttol=0.000010000
1462      Trunc()<hypothetical>:   t=0.099820000 | dT=0.000010000
1463        >> CppTimestep()<hyp>: t=0.099820000
1464        >> Trunc() {if(tmp!=*inst)} : no state changed
```

#1458-#1461 The timestep is undergoing a doubling process, and during this hypothetical simulation time test, when it adopts a timestep of 0.0032 as the next step size, a state change is identified. Consequently, triggering *timestep=ttol once again.

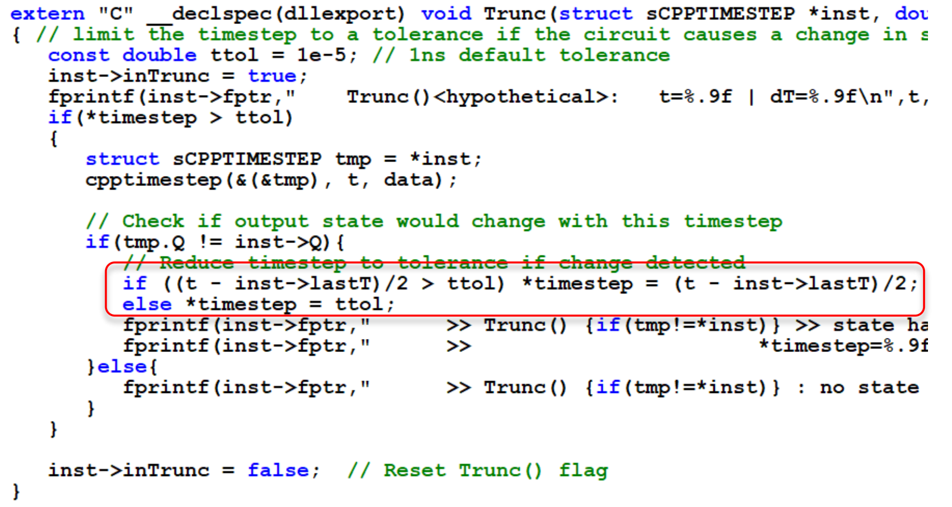
#1437-#1439 The timestep is reassigned to ttol once more and successfully passes the hypothetical test. Consequently, the next simulation step is set as the current simulation time (0.09981) plus ttol = 0.09982

This sequence persists until the timestep reaches the designated reference maxstep, and no further state changes are detected



DLL Workflow – Halving Hypothetical Timestep

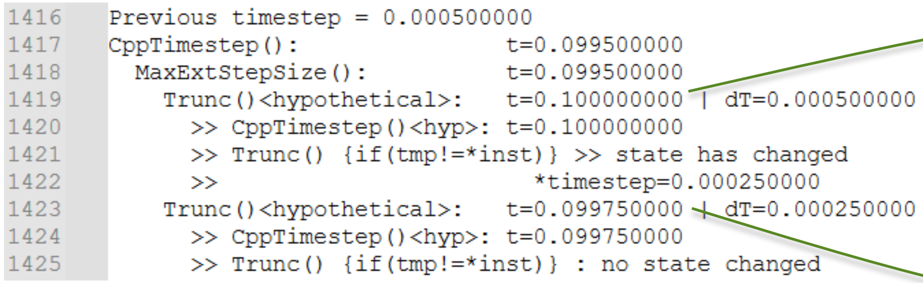
- *timestep Halving
  - **Trunc()** default implementation involves directly assigning ***timestep=ttol** in response to a state change event
  - It is feasible to employ a different timestep adjustment method, such as halving the timestep
  - In this section, the next hypothetical timestep is established as **(t – inst->lastT)/2**, which equates to the current hypothetical timestep divided by 2. The else condition restricts ***timestep** to **ttol** to avoid forcing the hypothetical timestep to an extremely small value



```
extern "C" __declspec(dllexport) void Trunc(struct sCPPTIMESTEP *inst, dou
{ // limit the timestep to a tolerance if the circuit causes a change in s
    const double ttol = 1e-5; // 1ns default tolerance
    inst->inTrunc = true;
    fprintf(inst->fptr,"    Trunc()<hypothetical>:   t=%.9f | dT=%.9f\n",t,
    if(*timestep > ttol)
    {
        struct sCPPTIMESTEP tmp = *inst;
        cpptimestep(&(&tmp), t, data);

        // Check if output state would change with this timestep
        if(tmp.Q != inst->Q){
            // Reduce timestep to tolerance if change detected
            if ((t - inst->lastT)/2 > ttol) *timestep = (t - inst->lastT)/2;
            else *timestep = ttol;
            fprintf(inst->fptr,"        >> Trunc() {if(tmp!=*inst)} >> state ha
            fprintf(inst->fptr,"        >>                          *timestep=%.9f
        }else{
            fprintf(inst->fptr,"        >> Trunc() {if(tmp!=*inst)} : no state
        }
    }

    inst->inTrunc = false;  // Reset Trunc() flag
}
```

DLL Workflow – Halving Hypothetical Timestep in Trunc()

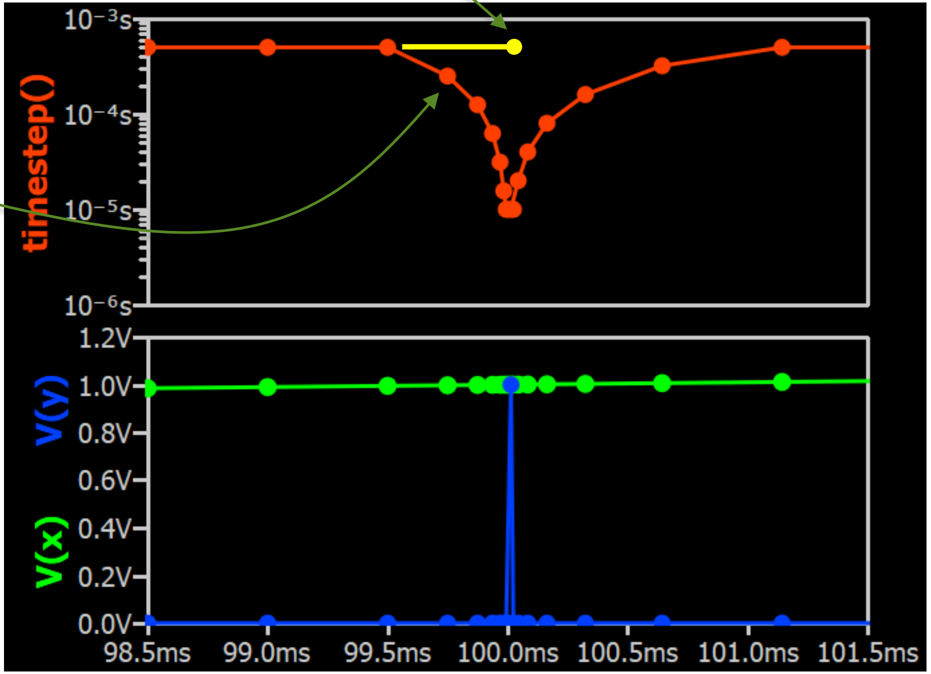
Analysis : Log.txt line#1416 to #1425



```
1416    Previous timestep = 0.000500000
1417    CppTimestep():               t=0.099500000
1418      MaxExtStepSize():          t=0.099500000
1419        Trunc()<hypothetical>:   t=0.100000000 | dT=0.000500000
1420          >> CppTimestep()<hyp>: t=0.100000000
1421          >> Trunc() {if(tmp!=*inst)} >> state has changed
1422          >>                        *timestep=0.000250000
1423        Trunc()<hypothetical>:   t=0.099750000 | dT=0.000250000
1424          >> CppTimestep()<hyp>: t=0.099750000
1425          >> Trunc() {if(tmp!=*inst)} : no state changed
```

#1419-#1422 During the testing of hypothetical simulation time in Trunc(), a state change is identified. The halving algorithm assigns *timestep as half of the current hypothetical timestep, leading to a reduction to 0.00025 in this scenario to proceed with the next Trunc() test

#1423-#1425 Rather than immediately decreasing the timestep to ttol, the halving algorithm only reduces the hypothetical timestep by half. Trunc() validates this timestep, and as no state change is detected, *timestep remains unchanged. This hypothetical timestep is then utilized to compute results from the main evaluation function


timestep()
10^{-3}s
10^{-4}s
10^{-5}s
10^{-6}s
V(y)
V(x)
1.2V
1.0V
0.8V
0.6V
0.4V
0.2V
0.0V
98.5ms
99.0ms
99.5ms
100.0ms
100.5ms
101.0ms
101.5ms