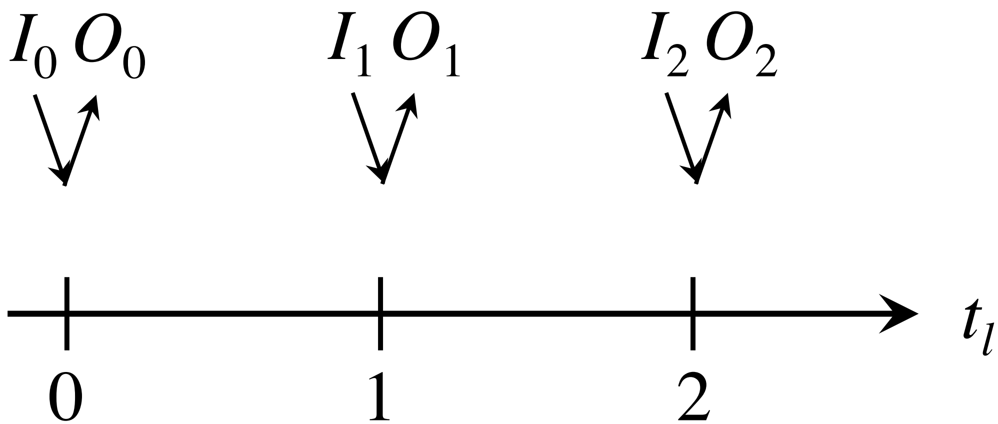
Real-Time Ticks for Synchronous Programming

Reinhard von Hanxleden (U Kiel)
Timothy Bourke (INRIA and ENS, Paris)
Alain Girault (INRIA and U Grenoble)

19 Sep 2017, FDL '17, Verona

Discrete (Logical) Time in Synchronous Programming



- Synchrony Hypothesis:
  Outputs are synchronous with inputs
- Computation "does not take time"
- Actual computation time does not influence result
- Sequence of outputs **determined** by inputs

Synchronous Execution

```
Initialize Memory
for each input event do
    Compute Outputs
    Update Memory
end
```

```
Initialize Memory
for each clock tick do
    Read Inputs
    Compute Outputs
    Update Memory
end
```

Fig. 1 Two common synchronous execution schemes: event driven (left) and sample driven (right).

[Benveniste et al., *The Synchronous Languages Twelve Years Later*, Proc. IEEE, 2003]

Multiform Notion of Time

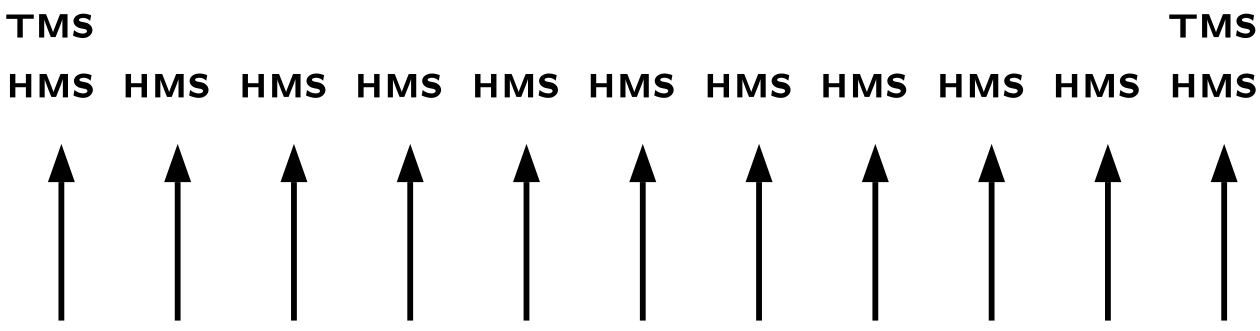
Only the simultaneity and precedence of events are considered.

This means that the physical time does not play any special role.

This is called multiform notion of time.

[https://en.wikipedia.org/wiki/Esterel]

Packaging Physical Time as Events



[Timothy Bourke, SYNCHRON 2009]

Event "HMS": 100 μsec have passed since last HMS

Event "TMS": 1000 μsec have passed since last TMS

A Problem With That ...

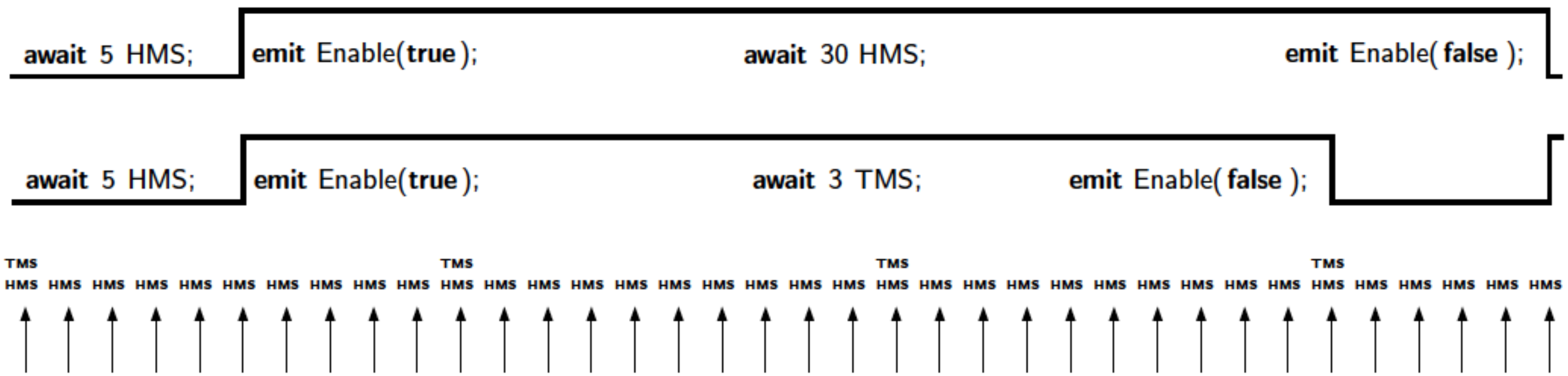
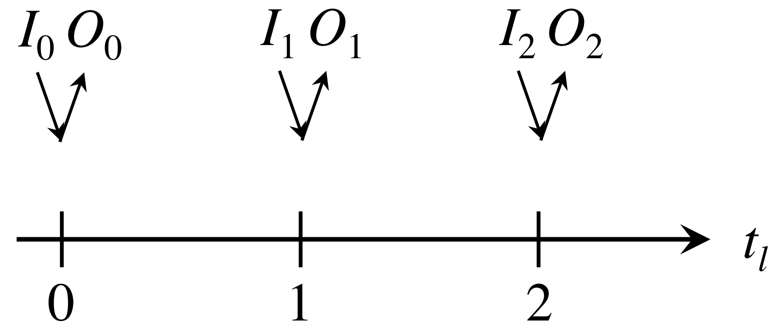


Fig. 4: Granularity of timing inputs

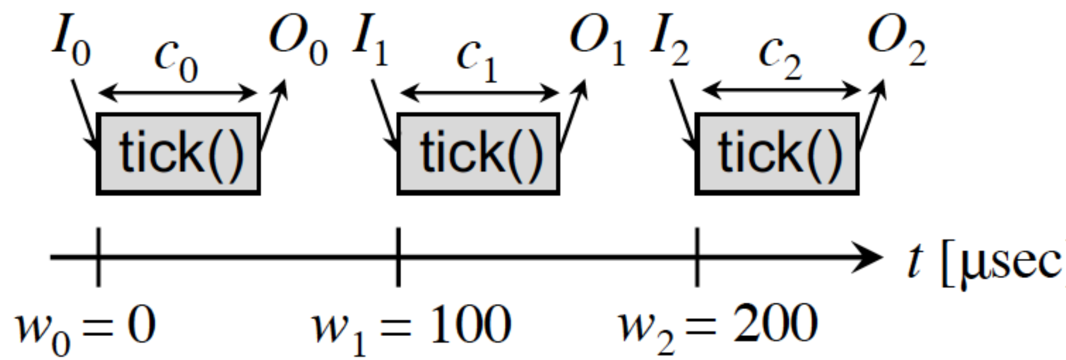
[Timothy Bourke, SYNCHRON 2009]

Dynamic Ticks

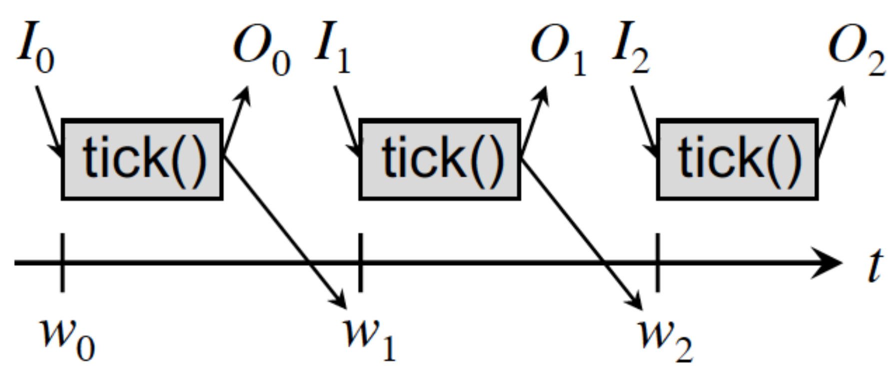
- Recall logical time:



- Physical time, time-triggered:



- Physical time, dynamic ticks:



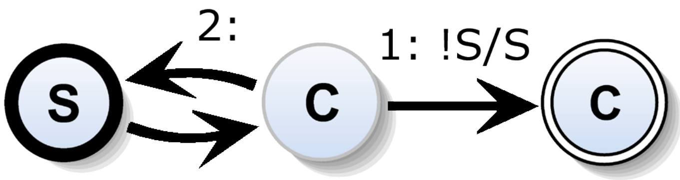
Esterel

```
module PAUSE_USEC:

input current_usec : integer;                     % Simulated time
input wait_usec : integer;                        % Time of delay
function min(integer, integer) : integer;
output wake_usec : combine integer with min;      % Time of next wake up
var my_wake_usec : integer in                     % Local copy of wake_usec

% Compute physical time when PAUSE_USEC should terminate
my_wake_usec := ?current_usec + ?wait_usec;

% Loop until current_usec = my_wake_usec
trap done in
  loop
    emit wake_usec(my_wake_usec);
    pause;
    if ?current_usec = my_wake_usec then
      exit done;
    end if;
  end loop
end trap
end var
end module
```

Host Code

```
int main()
{
  int notDone, prev_tick_end_usec = 0;

  RACE_reset();     // Reset automaton
  time_reset();     // Initialize time

  // Loop until tick function terminates
  do {
    // Set inputs
    RACE_I_current_wall_usec(get_current_wall_usec());
    RACE_I_prev_tick_end_usec(prev_tick_end_usec);

    notDone = RACE();   // Call tick function
    prev_tick_end_usec = get_current_wall_usec();

    // Wait until wake_usec
    microsleep(wake_usec - prev_tick_end_usec);
  } while (notDone);

  return 0;
}
```

DEMO
Dynamic Ticks in SCCharts



SCCharts

http://www.sccharts.com/

KIELER

The Key to Efficient Modeling

http://www.rtsys.informatik.uni-kiel.de/en/research/kieler

Eclipse Layout Kernel

https://www.eclipse.org/elk/

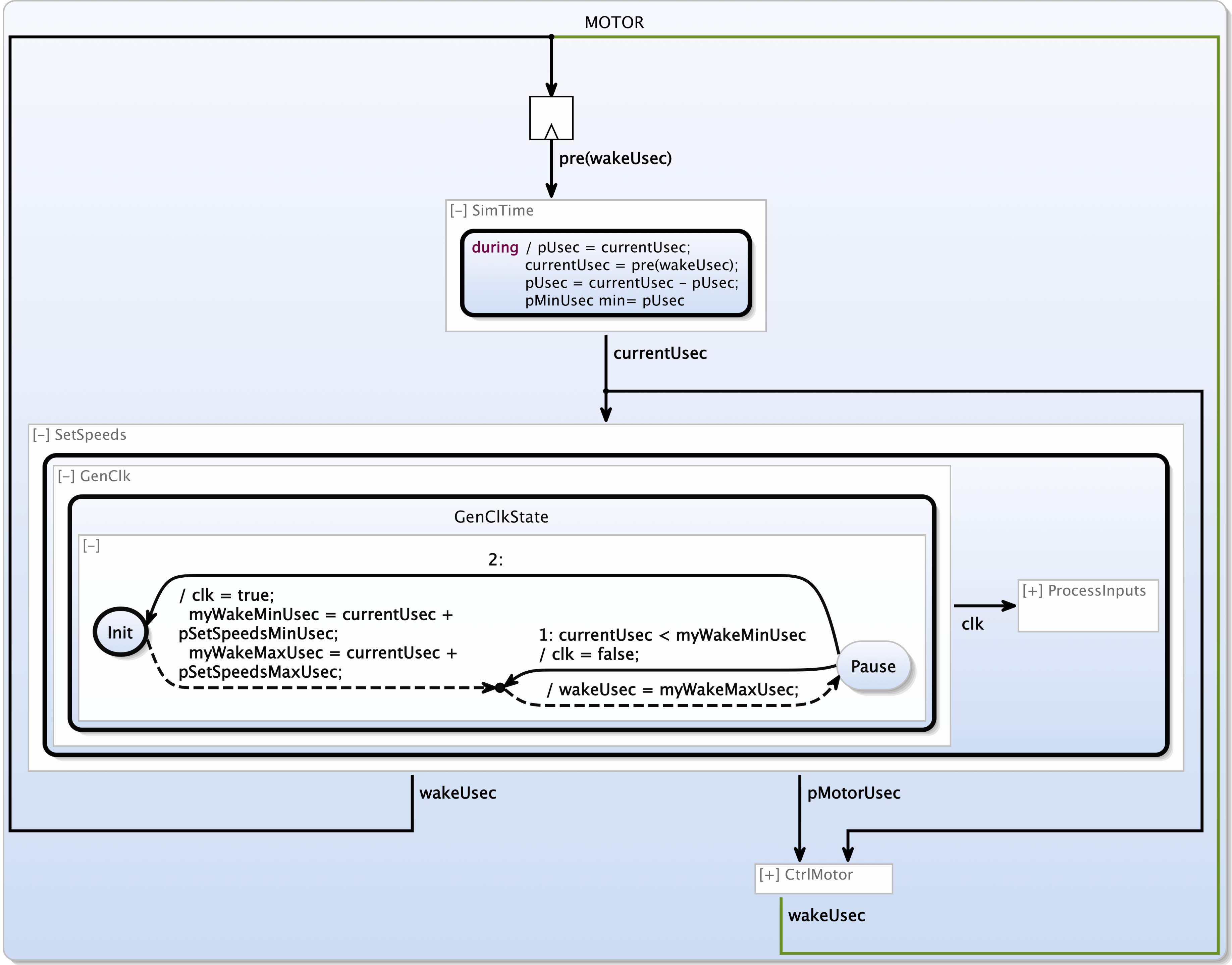
All available as open source under EPL

SCCharts Textual Representation

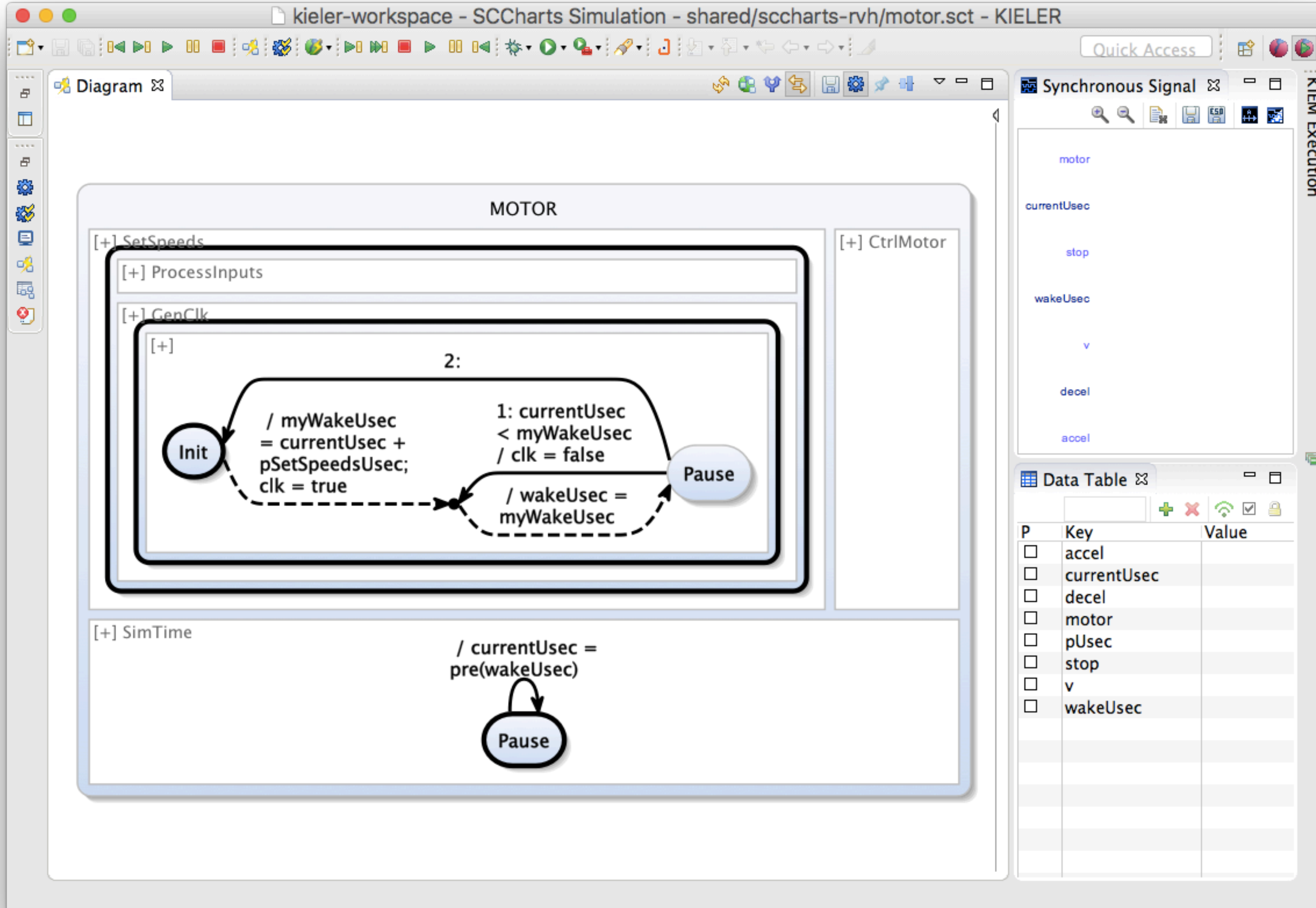
```
// Controller for stepper motor

scchart MOTOR {
  output int currentUsec = 0;  // Current simulated time; when deployed,
                                   this should be input
  output int wakeUsec;         // Time for next wake-up


  input bool accel, decel;     // Increase/decrease speed
  input bool stop;             // Emergency stop - sets (angular) speeds to 0


  output bool motor = false;   // Motor pulse
  output float v;              // [cm/sec] Robot speed
  output int pUsec;            // [usec] Half period for motor

  int pSetSpeedsUsec = 500000; // [usec] Period of speed control loop
  float dV = 2;                // [cm/sec] Delta v applied during one
                                   pSetSpeedsUsec
  float vMax = 20;             // [cm/sec] Max speed of left/right motor
  float cmPerHalfPeriod = 1;   // [cm] Distance traveled by motor per half
                                  period (duration of true or false)
```

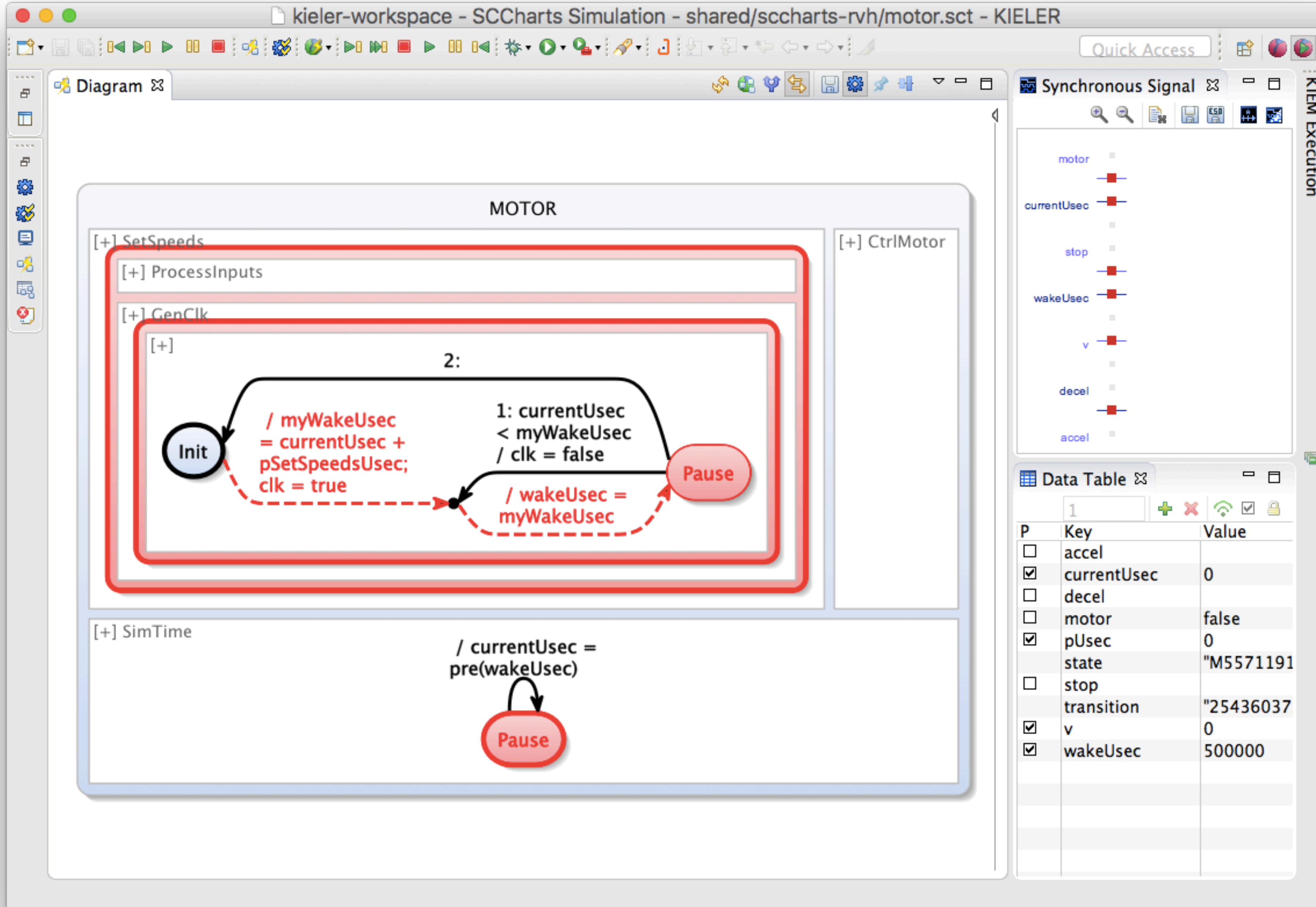
Now use KIELER to synthesize graphical SCChart with ELK and simulate …

```
// Controller for stepper motor

scchart MOTOR {
  output int currentUsec = 0;  // Current simulated time; when deployed, this should be input
  output int wakeUsec;         // Time for next wake-up

  input bool accel, decel;     // Increase/decrease speed
  input bool stop;             // Emergency stop - sets (angular) speeds to 0

  output bool motor = false;   // Motor pulse
  output float v;              // [cm/sec] Robot speed
  output int pUsec;            // [usec] Half period for motor

  int pSetSpeedsUsec = 500000; // [usec] Period of speed control loop
  float dV = 2;                // [cm/sec] Delta v applied during one pSetSpeedsUsec
  float vMax = 20;             // [cm/sec] Max speed of left/right motor
  float cmPerHalfPeriod = 1;   // [cm] Distance traveled by motor per half
                                  period (duration of true or false)

  // ==================================================================================
  region SetSpeeds:

  initial state SetSpeeds "" {
    bool clk; // Local clock

    // ==================================================================================
    region ProcessInputs:

    initial state Init
    --> Running immediate;

    state Running {
      entry / v = 0;

      // ==================================================================================
      region CalcV:

      initial state Pause
      --> Accel with clk & accel & !decel
      --> Decel with clk & decel & !accel;

      state Accel
      --> CheckMax immediate with / v += dV;

      state Decel
      --> CheckMin immediate with / v -= dV;

      state CheckMax
      --> SetPeriod immediate with v <= vMax
      --> SetPeriod immediate with / v = vMax;

      state CheckMin
      --> SetPeriod immediate with v >= - vMax
      --> SetPeriod immediate with / v = - vMax;

      state SetPeriod
      --> Pause immediate with v == 0 / pUsec = 0
      --> Pause immediate with / pUsec =
      '(int) (1000000 * cmPerHalfPeriod / v)';
    }
    o-> Running with clk & stop / pUsec = 0;

  // ==================================================================================
    region GenClk:

    initial state GenClkState "" {
      int myWakeUsec; //entry / clk = true;

      initial state Init
      --> AssertWakeTime immediate with / myWakeUsec = currentUsec +
      pSetSpeedsUsec; clk = true;

      connector state AssertWakeTime
      --> Pause immediate with / wakeUsec = myWakeUsec; // Initialize wakeUsec

  @layout[layerConstraint] LAST
      state Pause
      --> AssertWakeTime with currentUsec < myWakeUsec / clk = false
      --> Init;
    };
  };

  // ==================================================================================
  region CtrlMotor:

  initial state CtrlMotor "" {
    bool clk; // Local clock

    // ==================================================================================
    region GenClk:

    initial state GenClkState "" {
      int myWakeUsec;

      initial state Stopped
      --> AssertWakeTime immediate with pUsec > 0 / myWakeUsec = currentUsec +
      pUsec; clk = true;

      connector state AssertWakeTime
      --> Running immediate with / wakeUsec min= myWakeUsec; // Initialize wakeUsec

  @layout[layerConstraint] LAST
      state Running
      --> ResetClock with / clk = false;

      connector state ResetClock
      --> AssertWakeTime with pUsec > 0 & currentUsec < myWakeUsec
      --> Stopped;
    };

    // ==================================================================================
    region Motor:

    initial state Low
    --> High with clk / motor = true;

    state High
    --> Low with clk / motor = false;
  };

  // ==================================================================================
  region SimTime:

  initial state Pause
  --> Pause with / currentUsec = pre(wakeUsec);
}
```

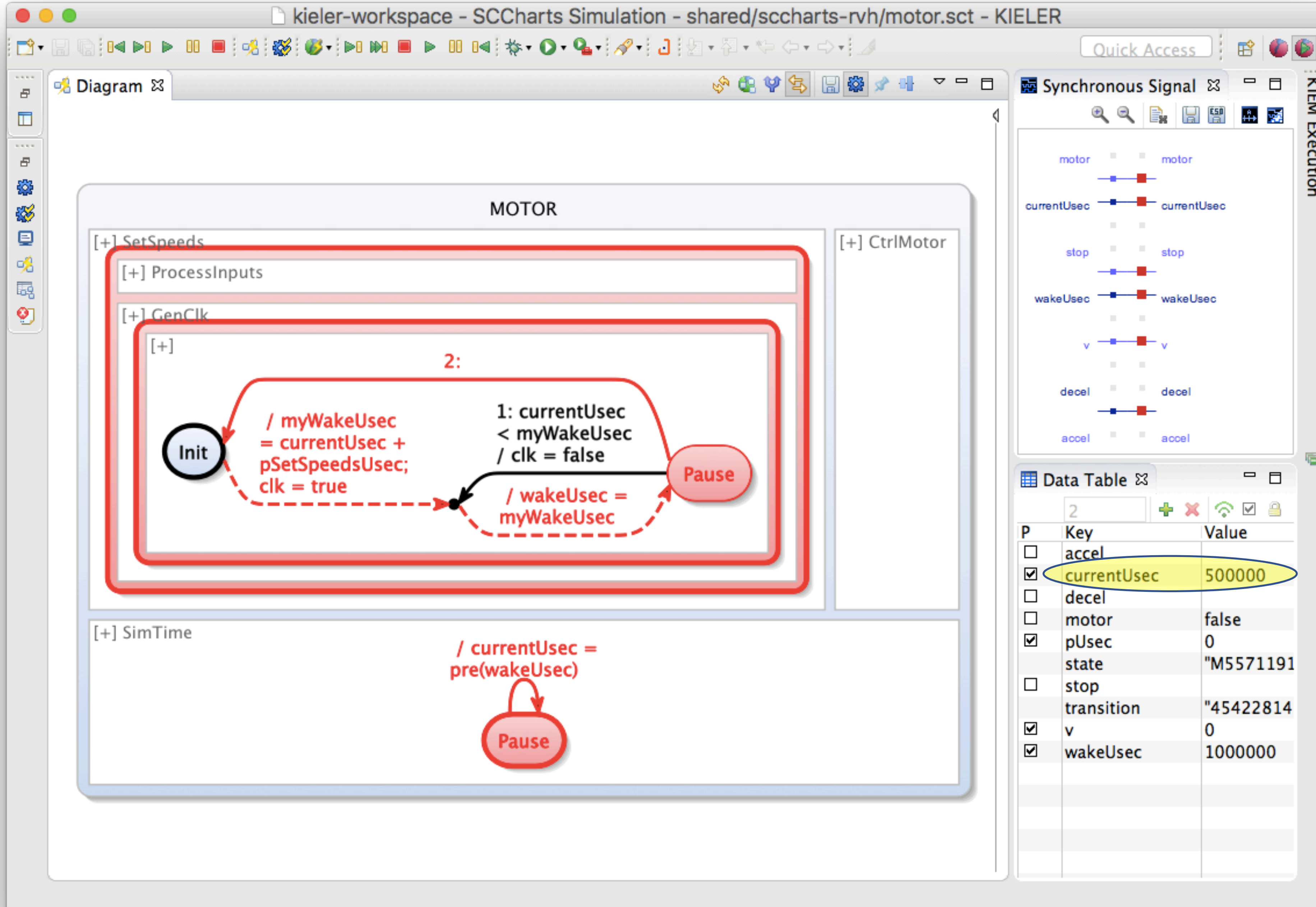


MOTOR
pre(wakeUsec)
[–] SimTime
during / pUsec = currentUsec;
currentUsec = pre(wakeUsec);
pUsec = currentUsec – pUsec;
pMinUsec min= pUsec
currentUsec
[–] SetSpeeds
[–] GenClk
GenClkState
[–]
2:
/ clk = true;
myWakeMinUsec = currentUsec +
pSetSpeedsMinUsec;
myWakeMaxUsec = currentUsec +
pSetSpeedsMaxUsec;
Init
1: currentUsec < myWakeMinUsec
/ clk = false;
Pause
/ wakeUsec = myWakeMaxUsec;
clk
[+] ProcessInputs
wakeUsec
pMotorUsec
[+] CtrlMotor
wakeUsec


kieler-workspace - SCCharts Simulation - shared/sccharts-rvh/motor.sct - KIELER
Quick Access
Diagram
MOTOR
[+] SetSpeeds
[+] ProcessInputs
[+] GenClk
[+]
2:
Init
/ myWakeUsec
= currentUsec +
pSetSpeedsUsec;
clk = true
1: currentUsec
< myWakeUsec
/ clk = false
Pause
/ wakeUsec =
myWakeUsec
[+] CtrlMotor
[+] SimTime
/ currentUsec =
pre(wakeUsec)
Pause
Synchronous Signal
motor
currentUsec
stop
wakeUsec
v
decel
accel
Data Table
P
Key
Value
accel
currentUsec
decel
motor
pUsec
stop
v
wakeUsec
KIEM Execution

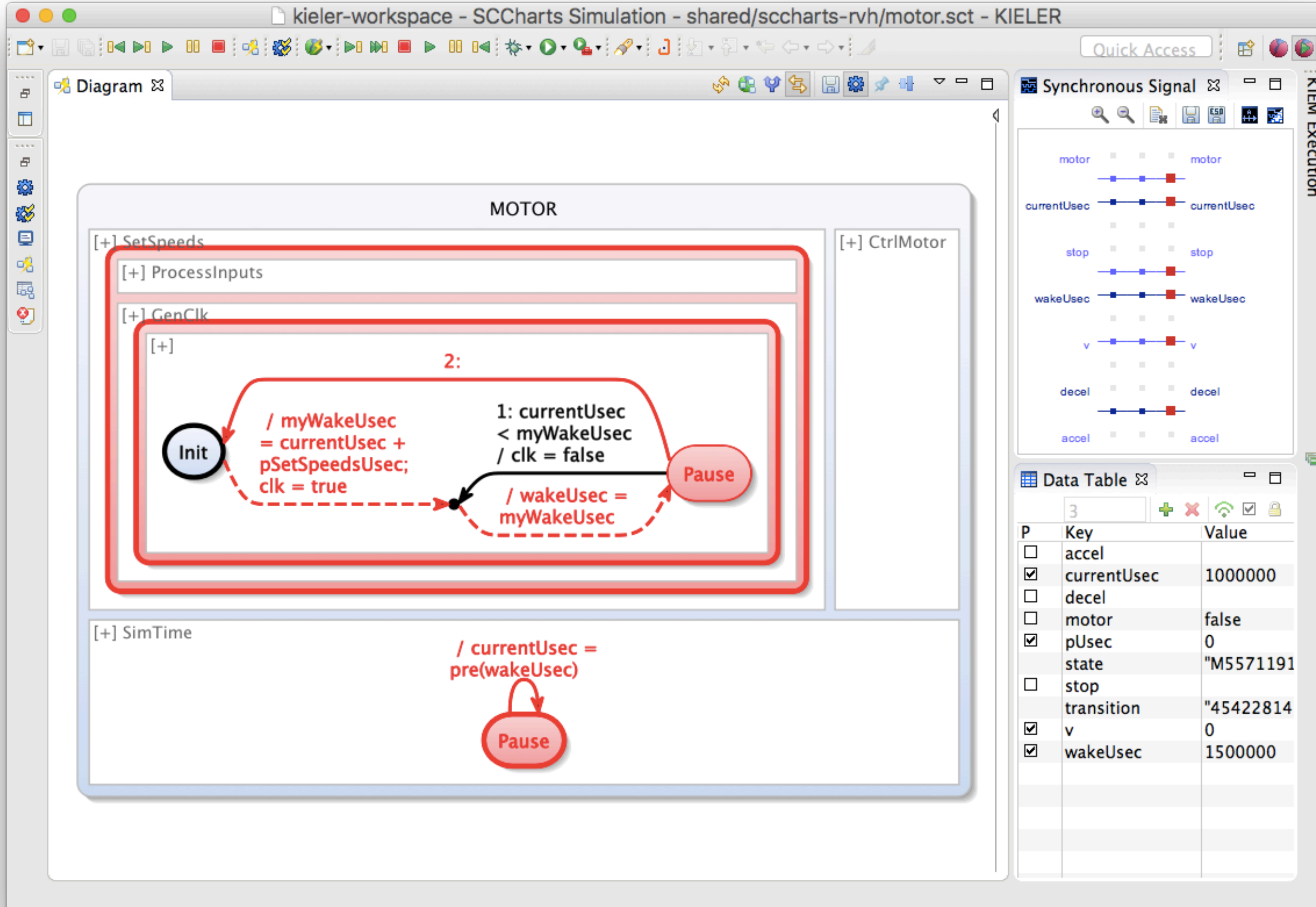


Logical time: 0 Physical time: 0 µsec

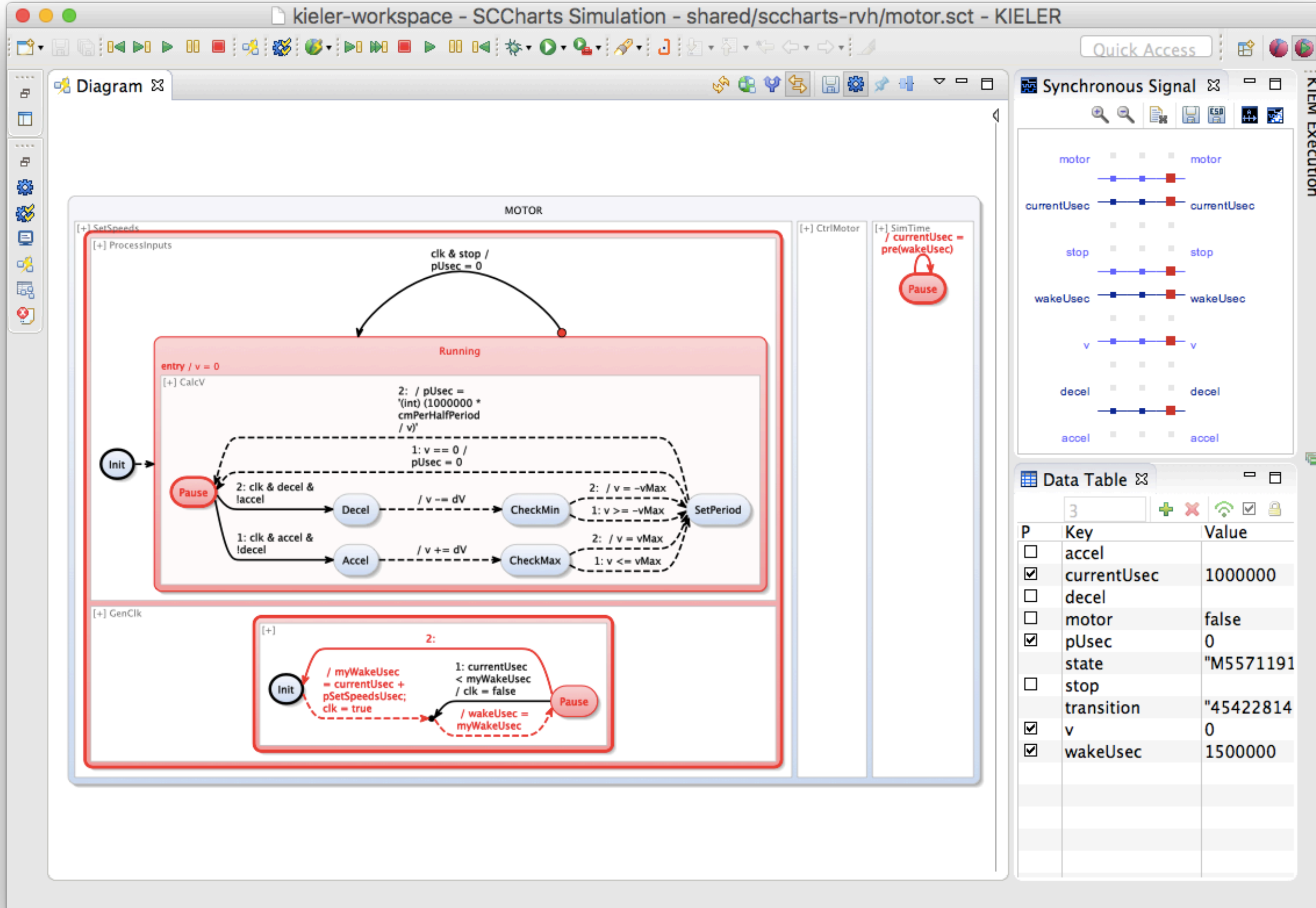


Logical time: 1

Physical time: 500,000 μsec

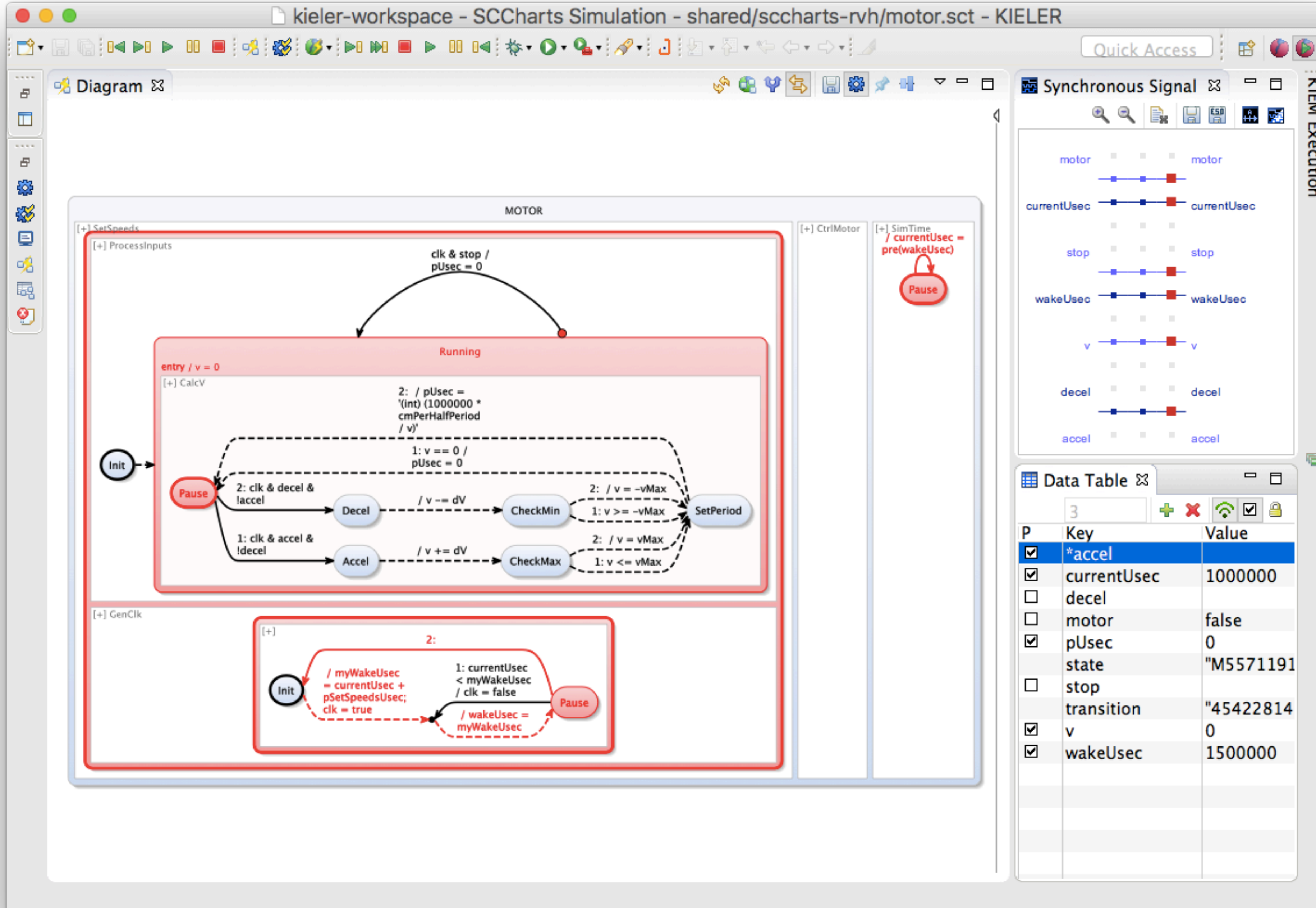


Logical time: 2 Physical time: 1,000,000 µsec

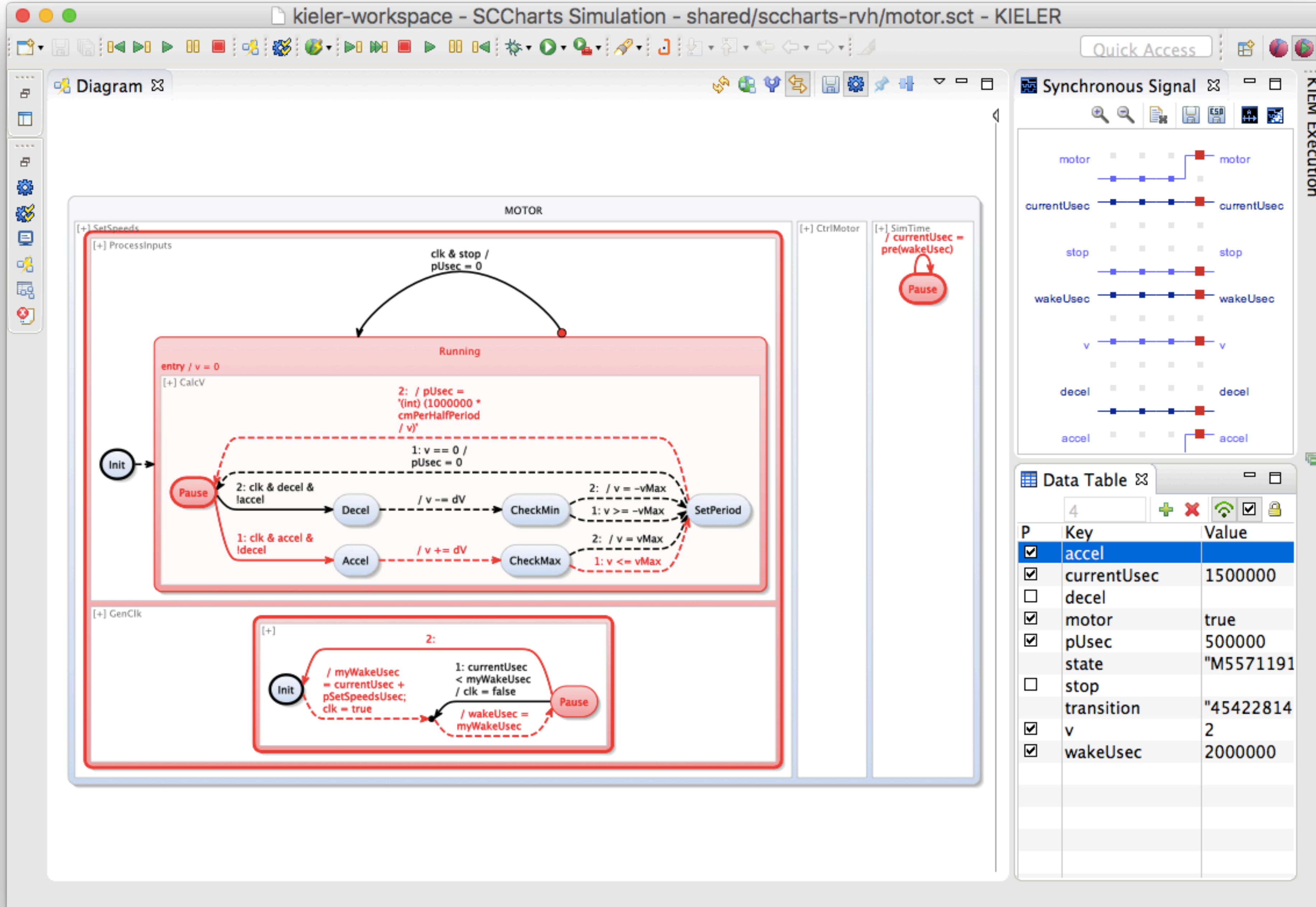


Logical time: 2

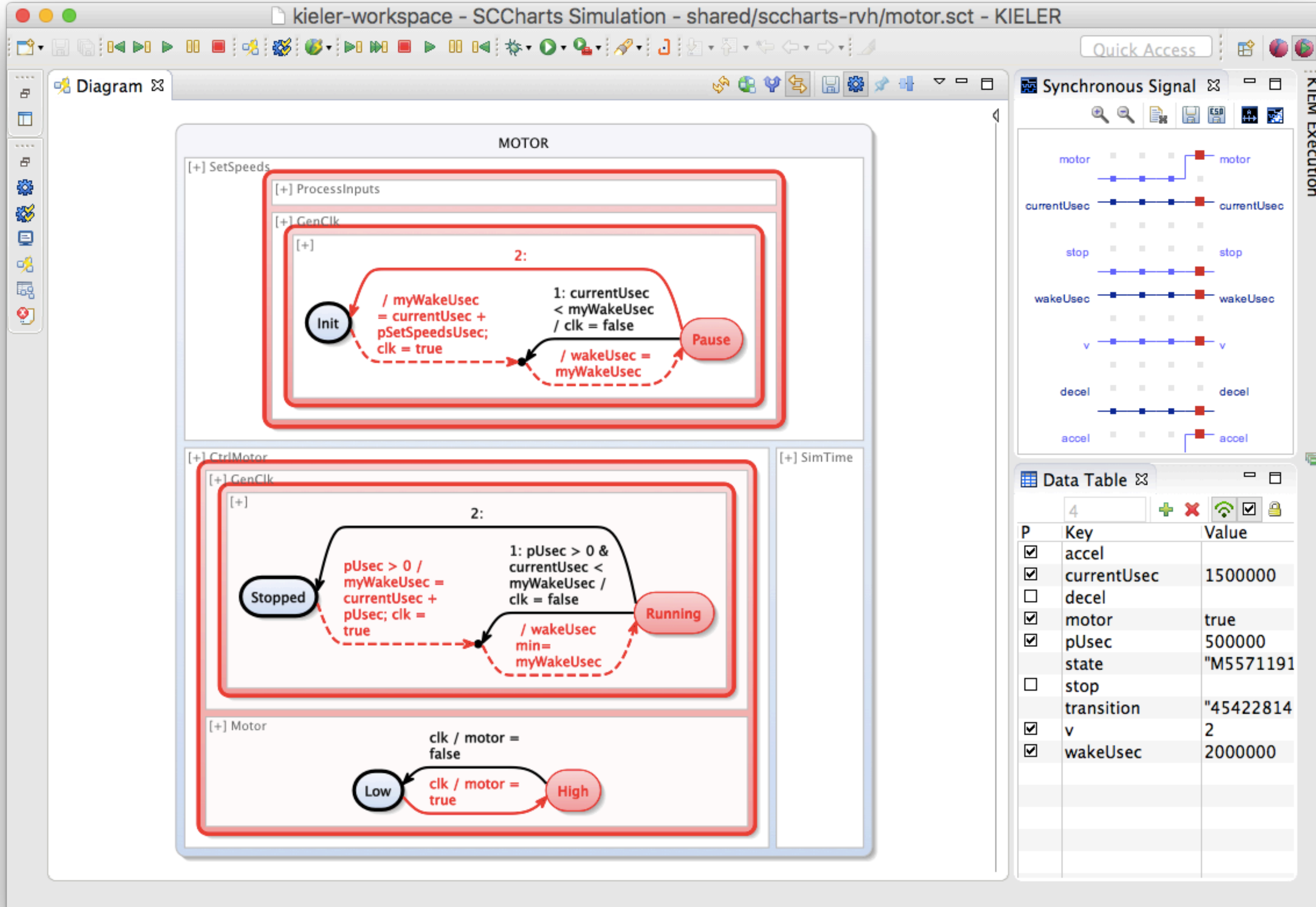
Physical time: 1,000,000 µsec



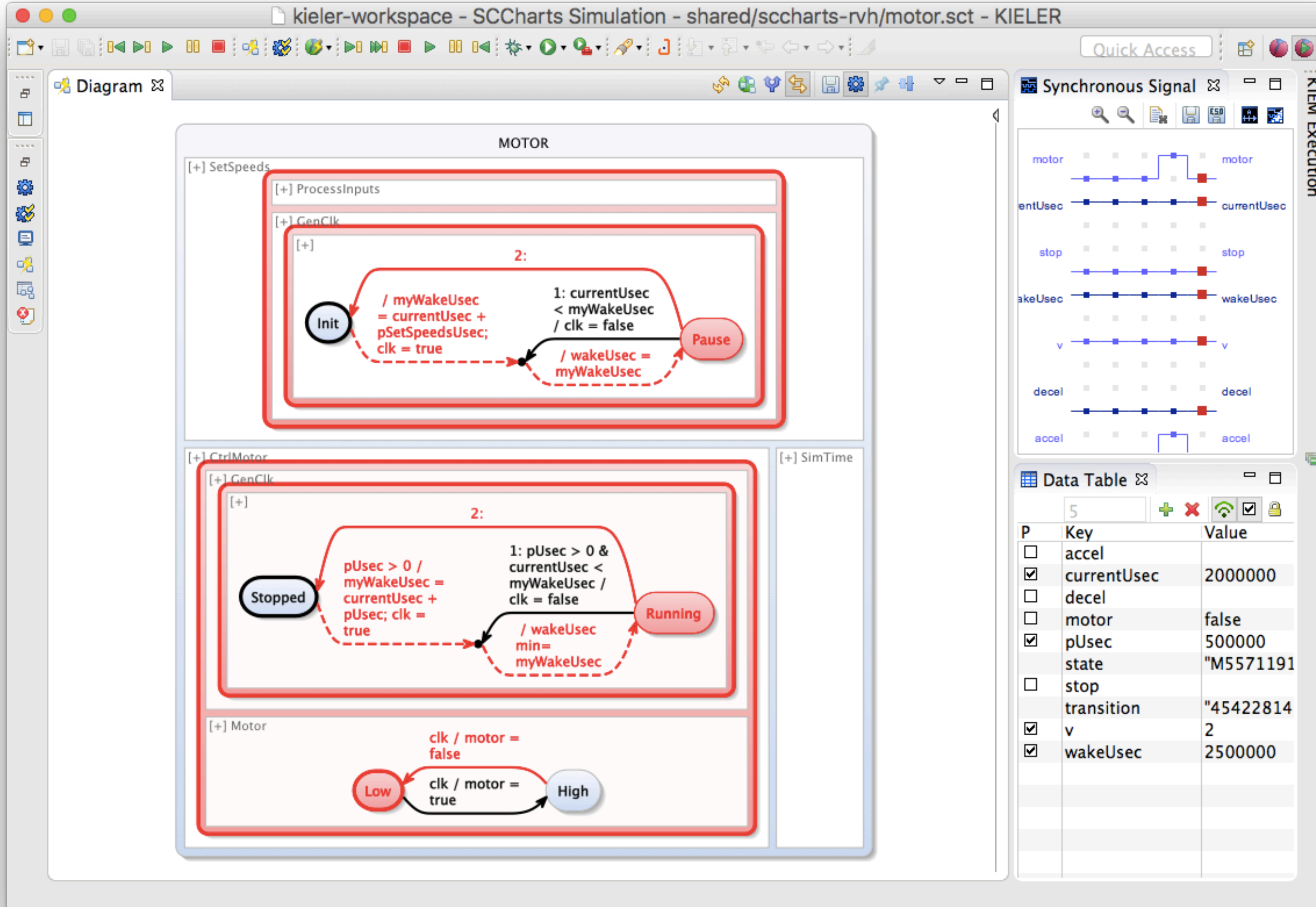
Logical time: 2 Physical time: 1,000,000 µsec



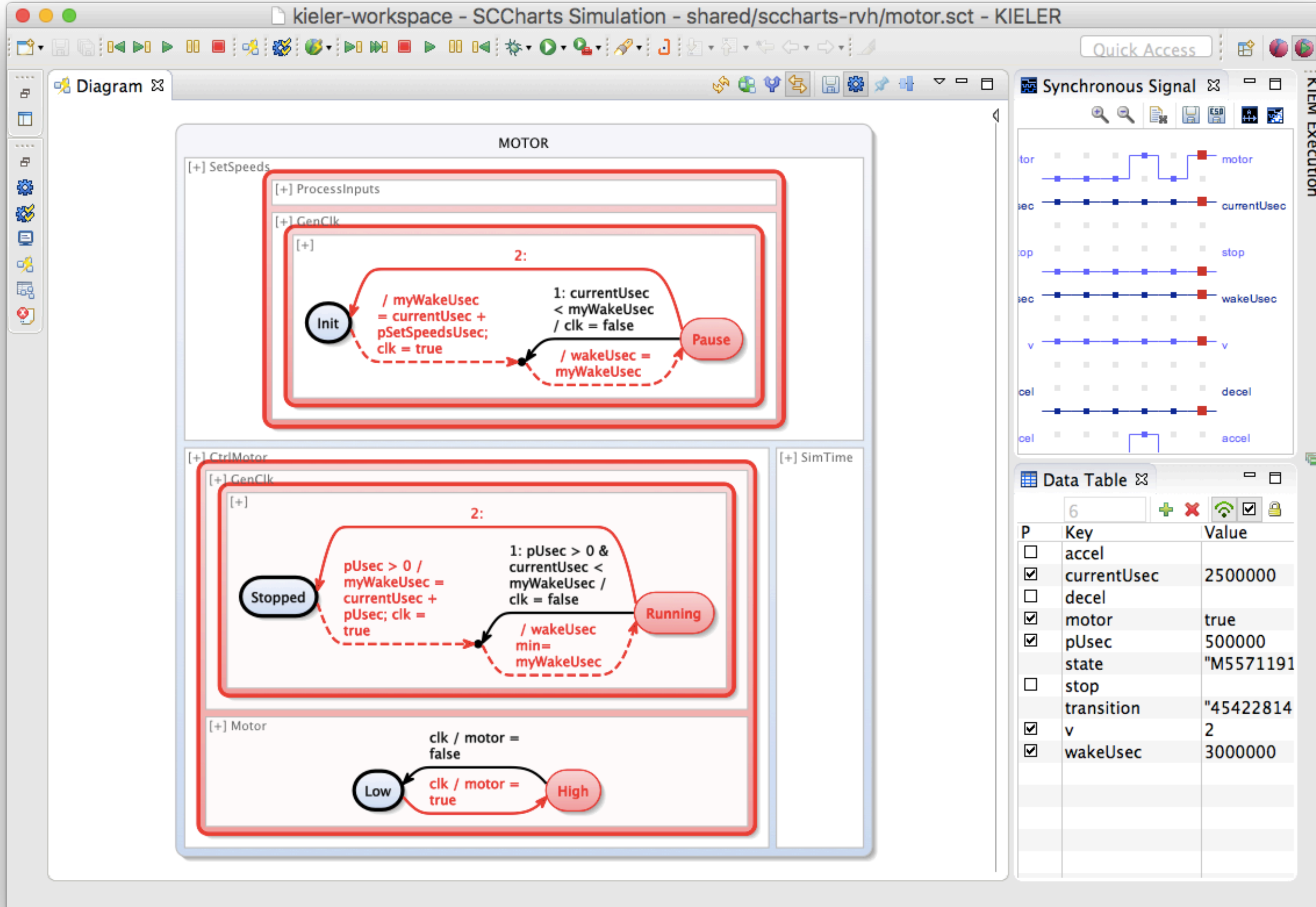
Logical time: 3 Physical time: 1,500,000 μsec



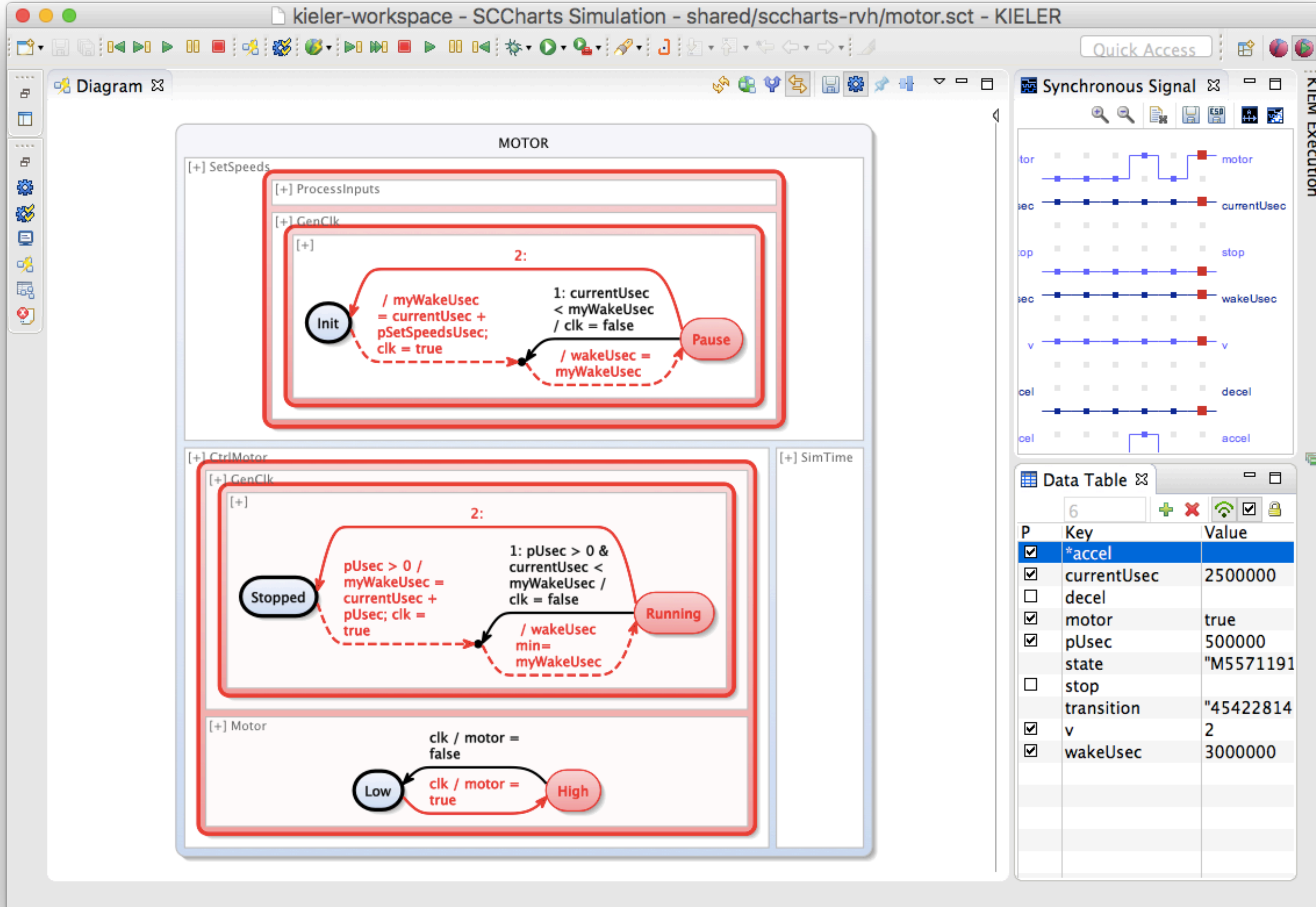
Logical time: 3 Physical time: 1,500,000 µsec



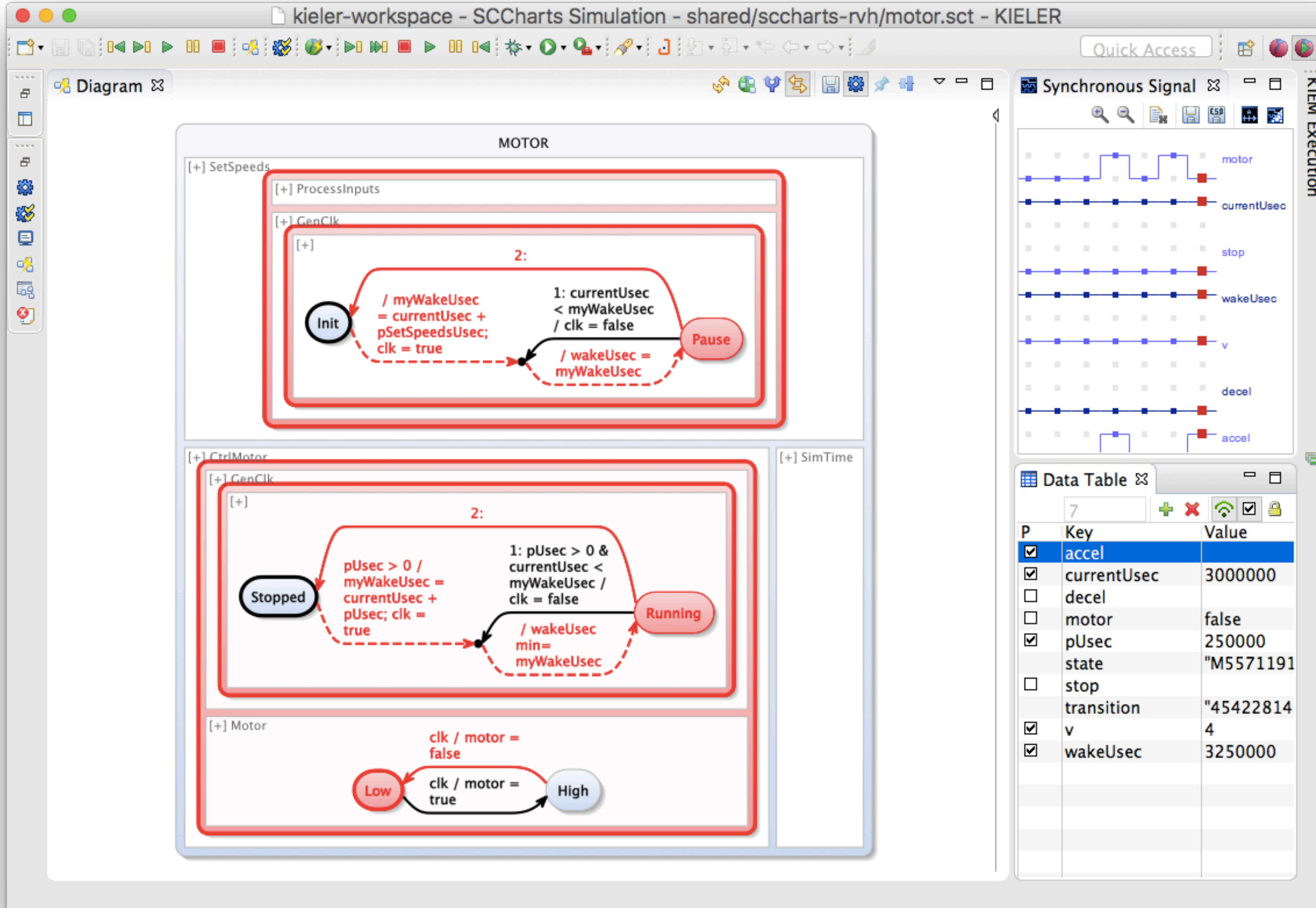
Logical time: 4 Physical time: 2,000,000 µsec



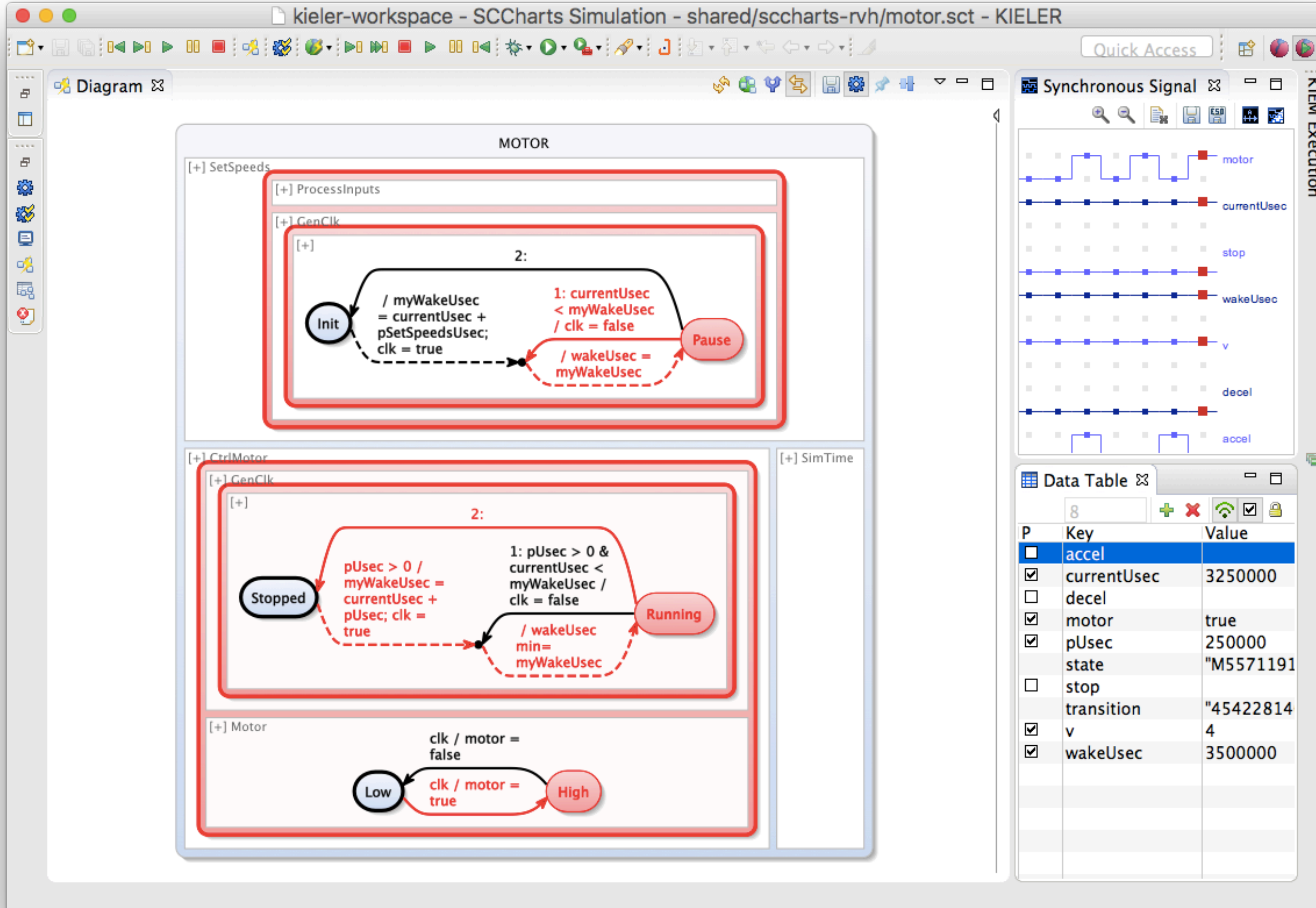
Logical time: 5 Physical time: 2,500,000 μsec



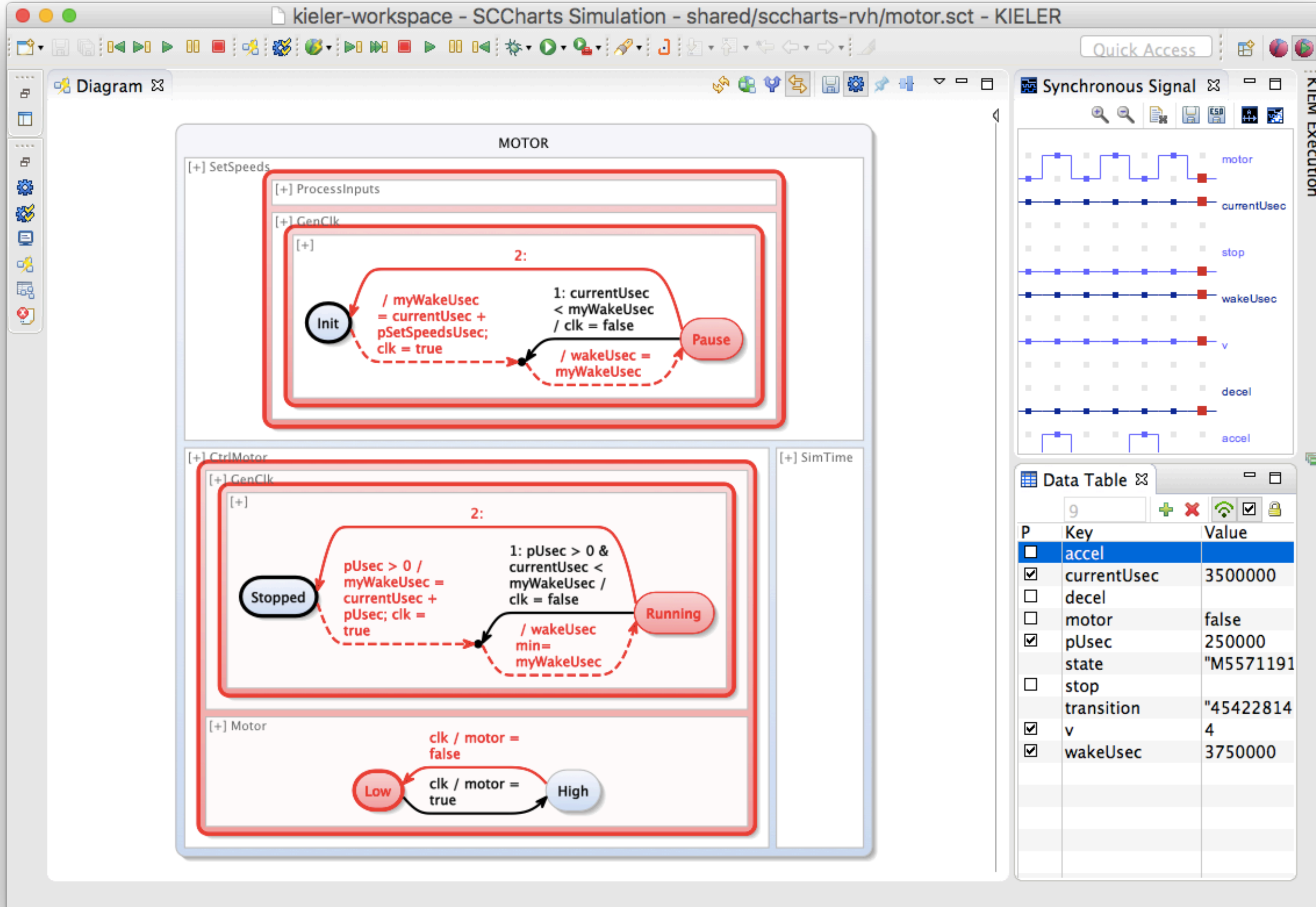
Logical time: 5 Physical time: 2,500,000 µsec



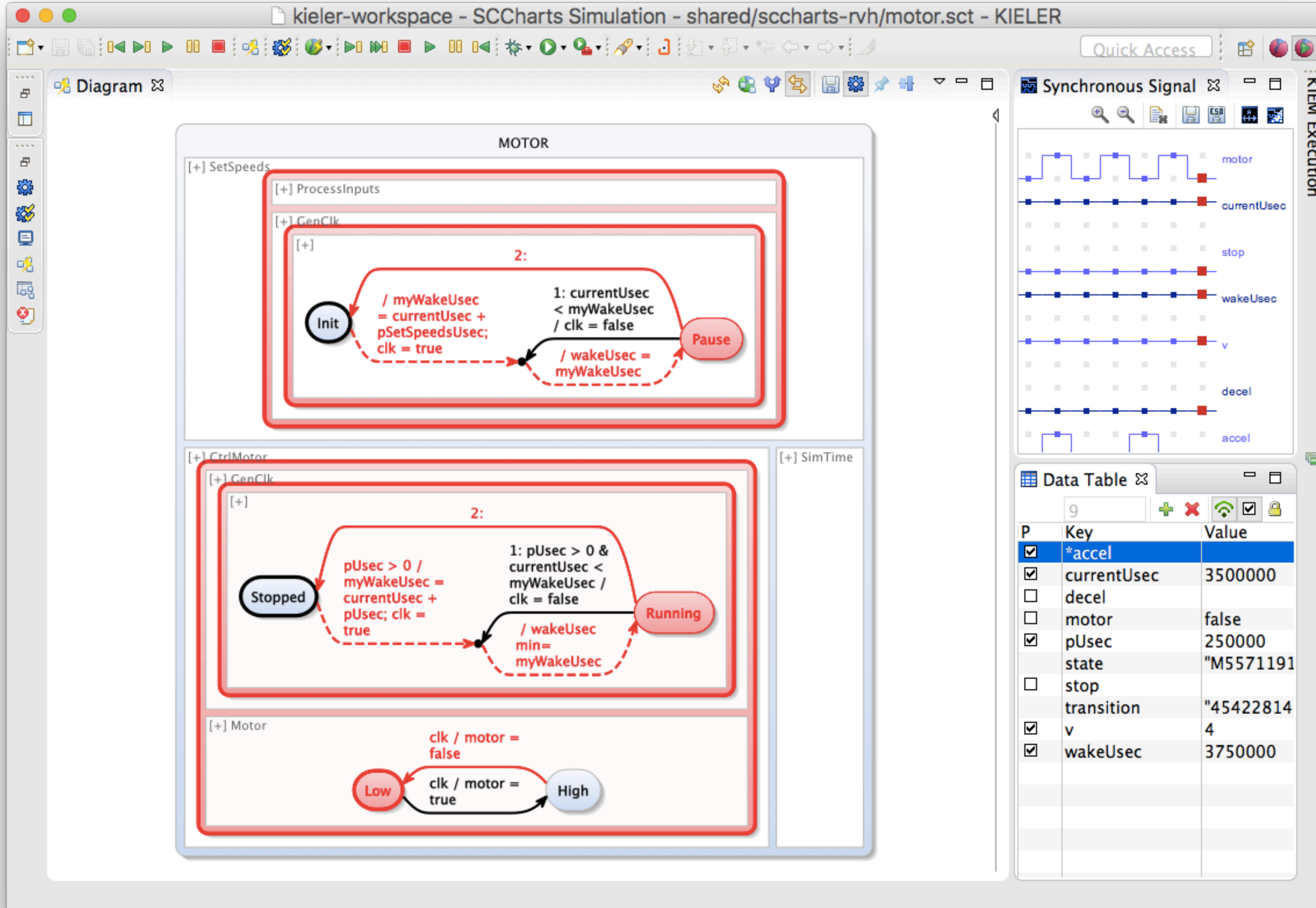
Logical time: 6 Physical time: 3,000,000 μsec



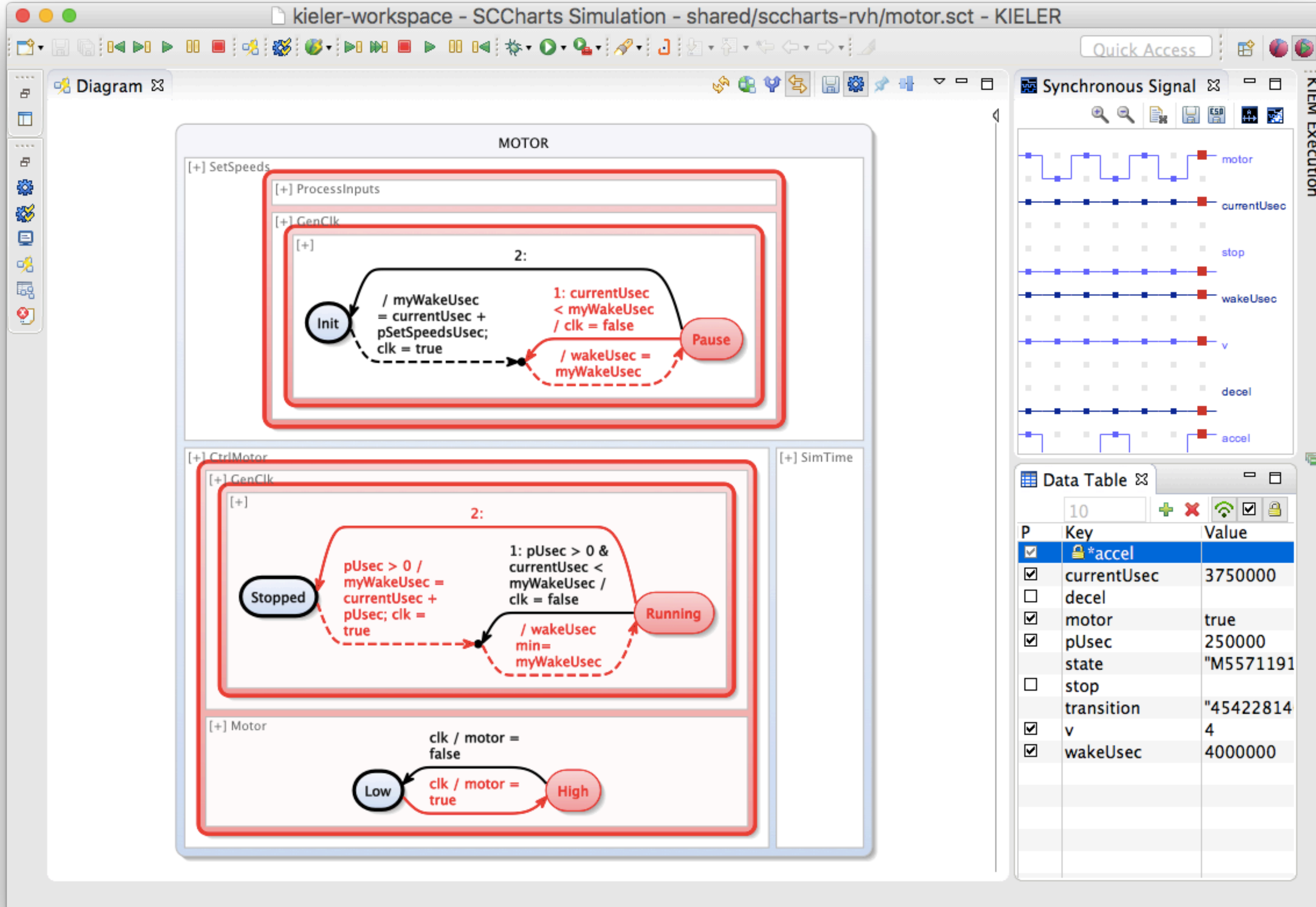
Logical time: 7 Physical time: 3,250,000 µsec



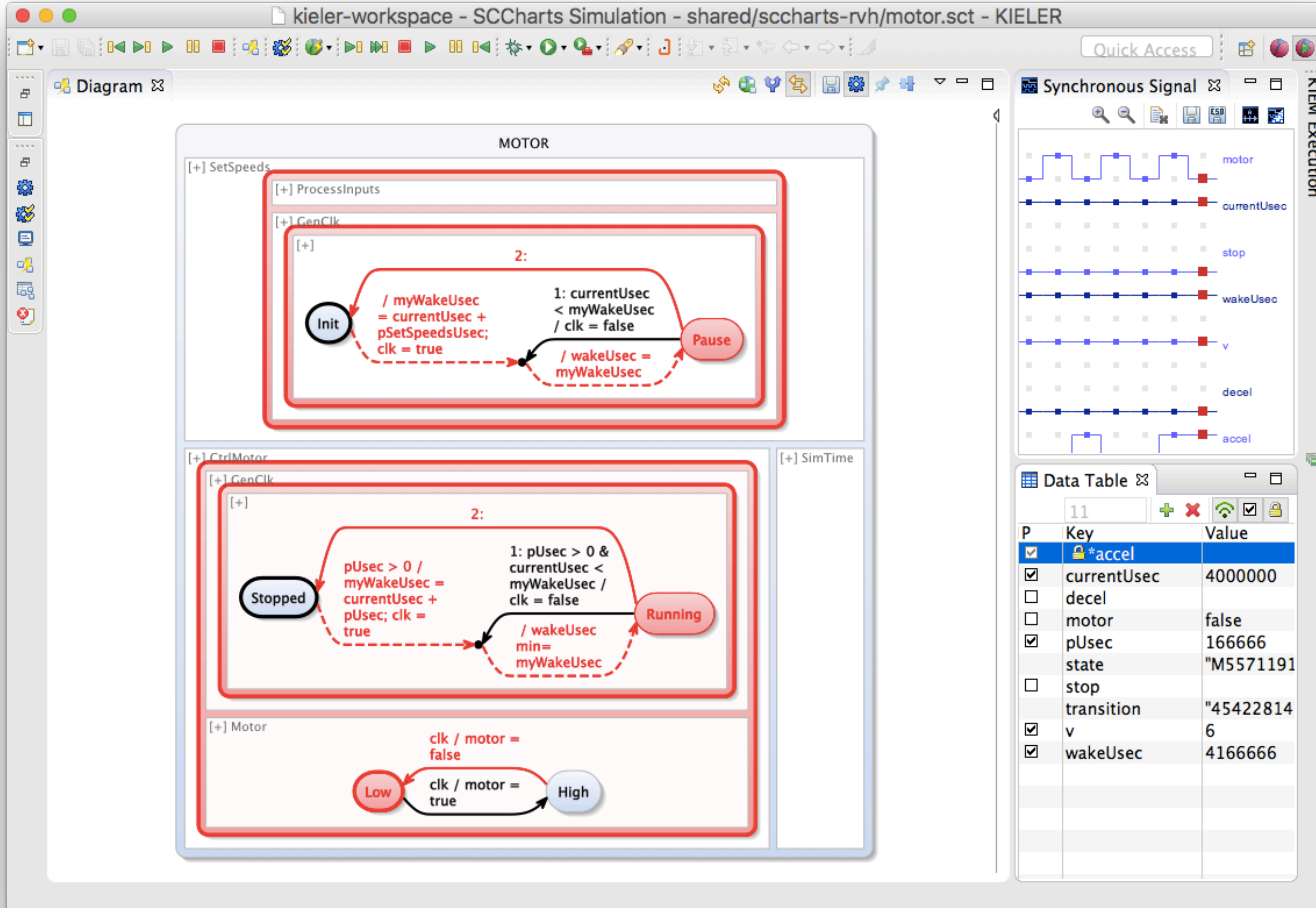
Logical time: 8 Physical time: 3,500,000 μsec



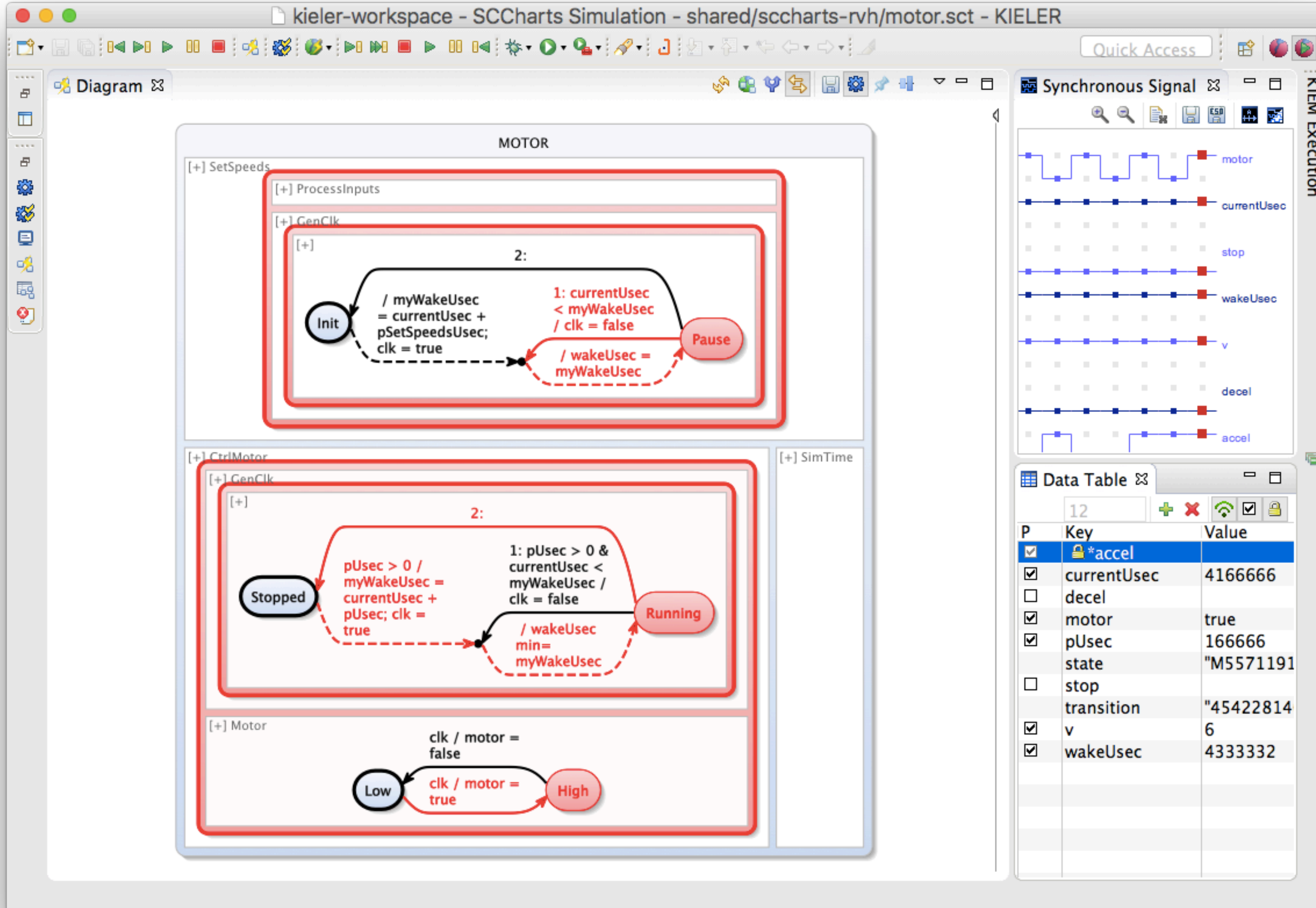
Logical time: 8 Physical time: 3,500,000 μsec



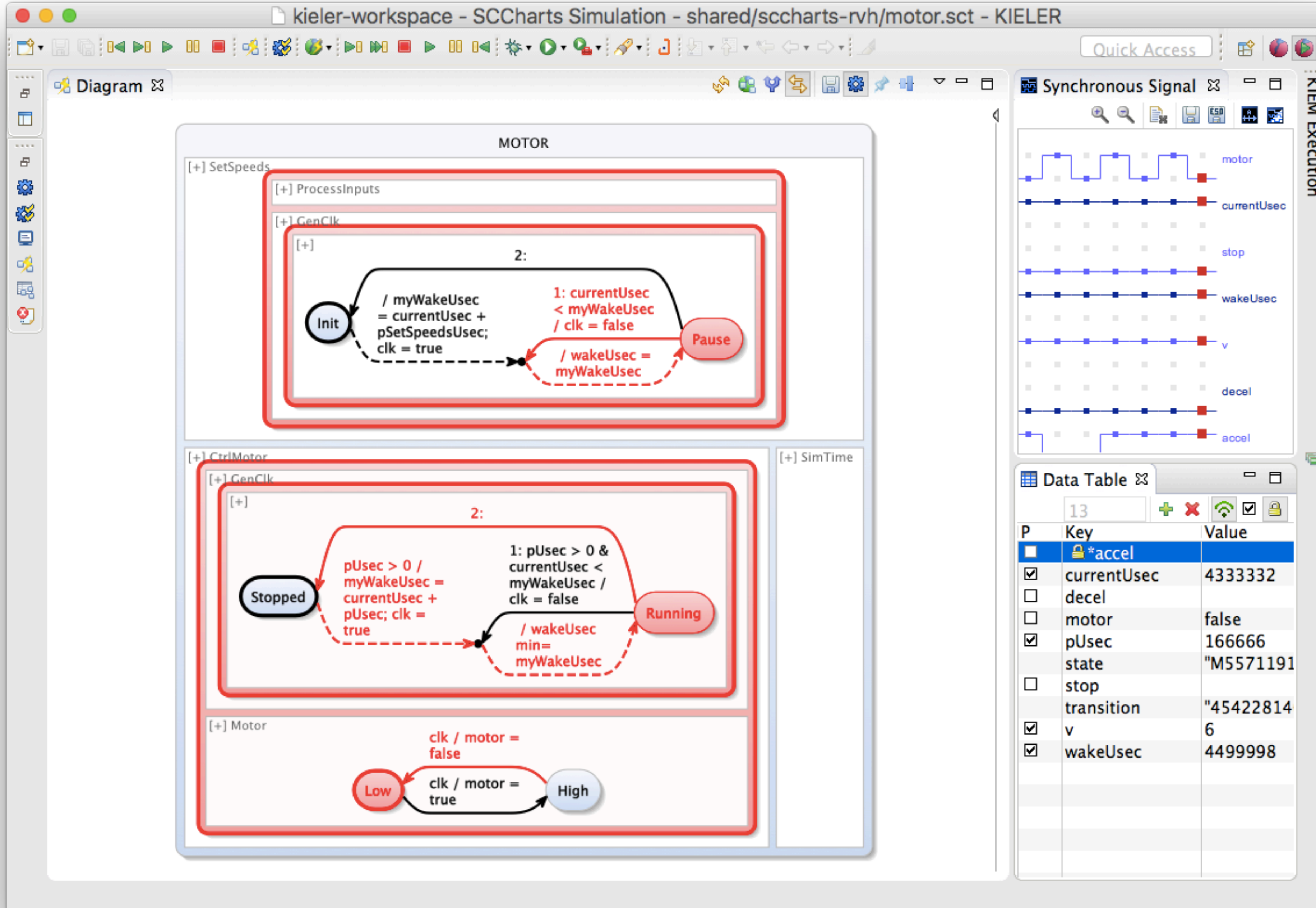
Logical time: 9 Physical time: 3,750,000 µsec



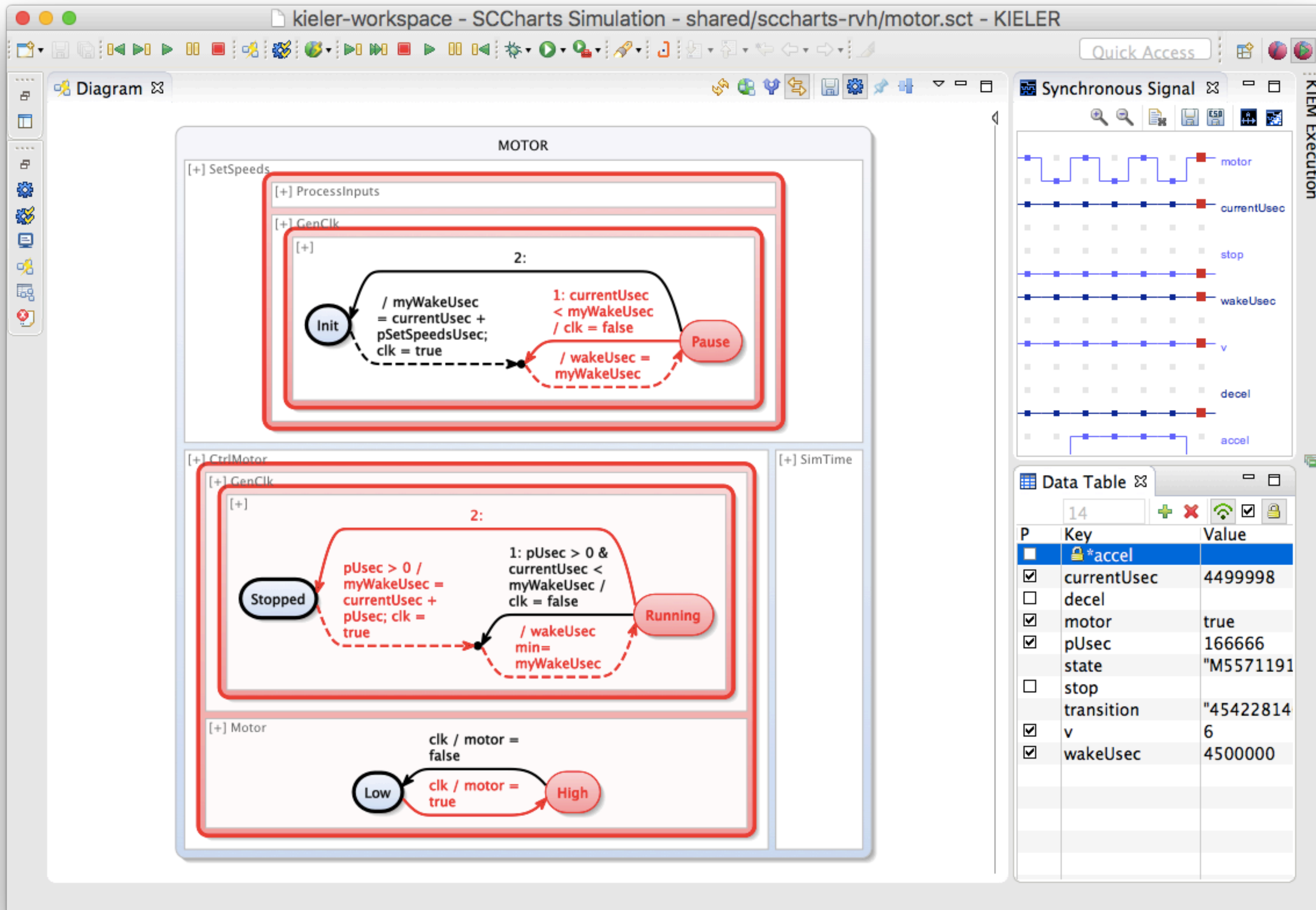
Logical time: 10 Physical time: 4,000,000 µsec



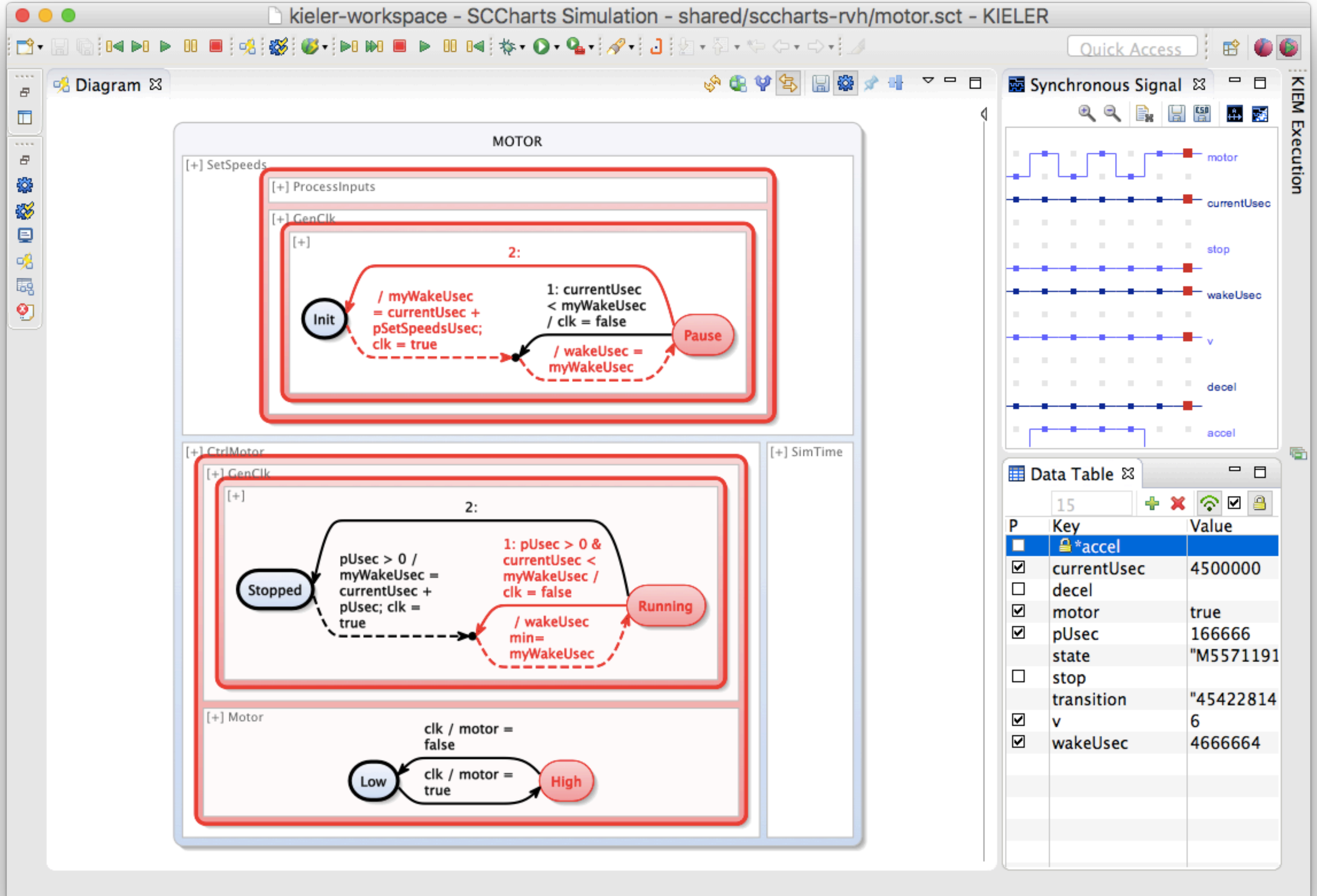
Logical time: 11 Physical time: 4,166,666 µsec



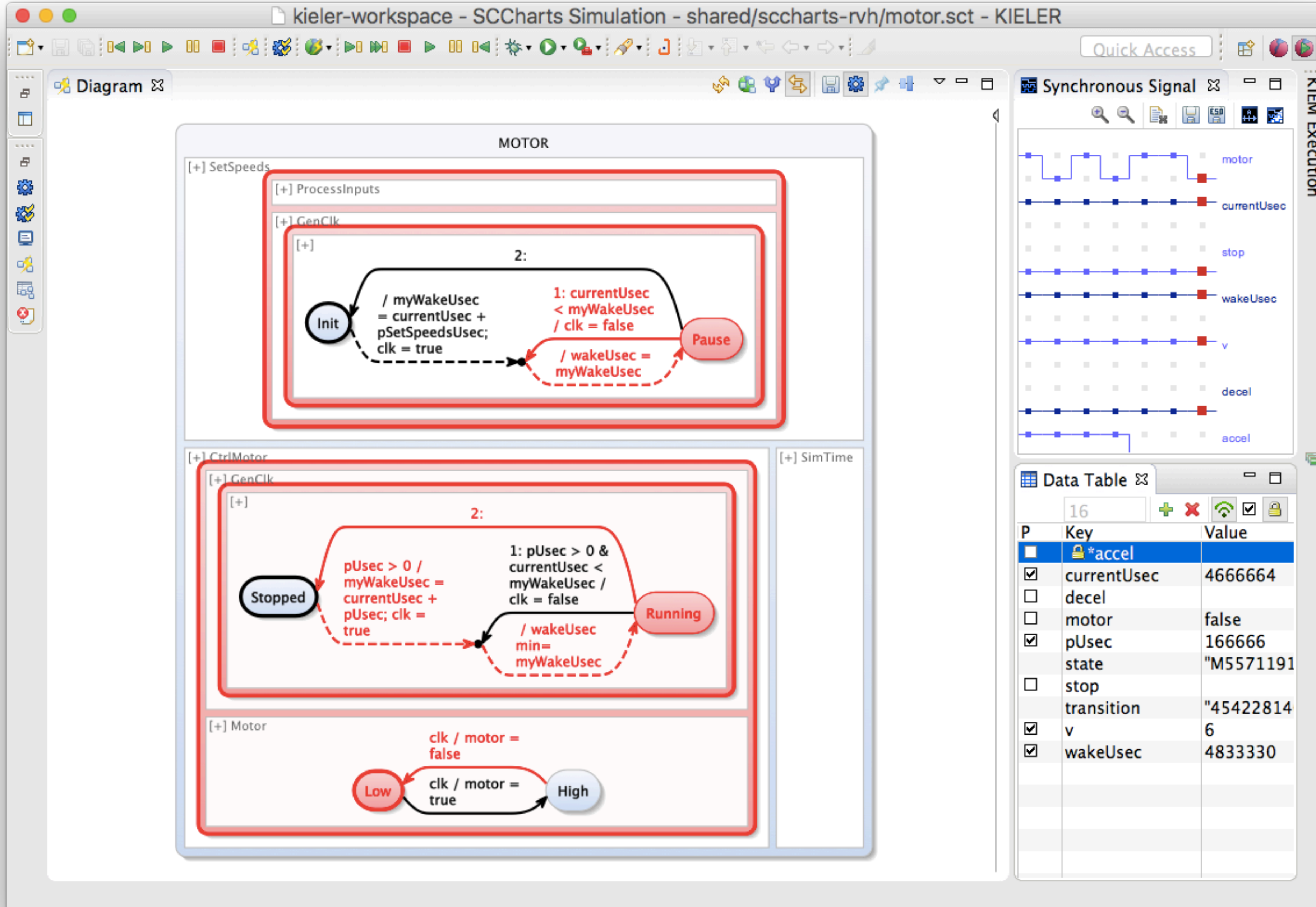
Logical time: 12 Physical time: 4,333,332 µsec



Logical time: 13 Physical time: 4,499,998 µsec



Logical time: 14   Physical time: 4,500,000 μsec



Logical time: 15 Physical time: 4,666,664 μsec

Wrap-Up

- Dynamic ticks seamlessly integrate physical time with logical time
- Existing synchronous languages (Esterel, SCCharts) already provide all necessary features

Future work

- Timing analysis
- Multiclocking
- Language extensions

Thanks!