Trick Simulation Environment:
Advanced Topics

John Penn (L-3Com/ER7)
Danny Strauss (L-3Com/ER7)
Alex Lin (NASA/ER7)

also authored by
Scott Killingsworth (NASA/ER7)
Eddie Paddock (NASA/ER7)
Les Quiocho (NASA/ER7)
Keith Vetter (Spoonbill Services/ER7)

Agenda/Schedule

- **Realtime and Distributed Topics**
  1. **Trickcomm and the Variable Server**
  2. **Multiprocessing and Realtime**
  3. **Master/Slave Import/Export**
  4. **Real World Realtime/Multiprocessing/Master/Slave example**
  5. **Monte Carlo**
  6. **Generic Malfunction Insertion**
  7. **Units Upgrade**
  8. **Wide Character Support**

- **Additional Material**
  1. **External Clocks and Timers**
  2. **External Libraries and Trick Math Library**

Trickcomm and the Variable Server

- **Objective**
  - **Describe the Trickcomm package**
  - **Describe the Variable server**

- **Prerequisites**
  - **Knowledge of sockets**

Trickcomm

- **Trickcomm is a C library built on top of the system TCP/IP socket library.**
- **Originally, Trickcomm was provided as a consistent “stream” interface over sockets, reflective memory, and shared memory.**
- **Over the past few years, Sockets have proved fast enough so Trickcomm only supports sockets now.**
- **Trickcomm can be used as a standalone package.**
  - **Usable as a library to non-Trick sims**
  - **Usable under Windows**

Trickcomm

- **Provided functions – Connecting (Server)**
  - **tc_init(TCDevice * listen_device) ;**
    - Initializes a listen socket and begins listening for a client connection.
    - **`listen_device.port`** must be specified. Uses unix *listen().*
  - **tc_listen(TCDevice * listen_device) ;**
    - Returns true if a client is trying to connect on the listen socket.
    - Uses unix *poll().*
  - **tc_accept(TCDevice *listen_device, TCDevice *device) ;**
    - Accepts the client connection request from **`listen_device`** onto **`device`**.
    - Will block until client connects. Endianness of client is recorded.
    - Uses unix *accept().*
- **Provided function – Connecting (Client)**
  - **tc_connect(TCDevice *device);**
    - Connects to a listening socket. Endianness of server is recorded.
    - **`listen_device.port`** & **`hostname`** must be specified. Uses unix *connect().*

- **Provided functions – Connecting (Server and Client)**
  - **tc_multiconnect(TCdevice *device, char *connection_tag, char *my_tag, TrickErroHndlr *error_handler) ;**
    - Both the server and client call tc_multiconnect
    - Each side provides **`connection_tag`** which must be equal
    - Each side provides **`my_tag`** which must be different
    - tc_multiconnect will use multicasting sockets to find other connections that have the same **`connection_tag`** and different **`my_tag`**
    - tc_multiconnect will determine who is the server and client and call tc_accept and tc_connect with appropriate port numbers to establish a connection (user does not specify port)
    - tc_multiconnect returns when the connection is made

```
process 1:
tc_multiconnect(dev , “important_comm!” , “side a” , err_hndlr) ;

process 2:
tc_multiconnect(dev , “important_comm!” , “side b” , err_hndlr) ;
```

- **Provided functions – Read/Write**
  - **tc_read(TCDevice * device, char *buffer, int size) ;**
    - Reads **`size`** number of bytes
  - **tc_write(TCDevice * device, char *buffer, int size) ;**
    - Writes **`size`** number of bytes
  - **tc_read_byteswap(TCDevice * device, char *buffer, int size, ATTRIBUTES *attr) ;**
    - Calls tc_read. If other side of connection is opposite endianness, takes structure information of **`buffer`** from **`attr`** and byteswaps **`buffer`** .
    - **`attr`** is generated for each structure by ICG (in S_source.c)
  - **tc_write_byteswap(TCDevice * device, char *buffer, int size, ATTRIBUTES *attr) ;**
    - If other side of connection is opposite endianness, takes structure information of **`buffer`** from **`attr`** and byteswaps **`buffer`** . Calls tc_write.
    - **`attr`** is generated for each structure by ICG (in S_source.c)

Trickcomm - Blocking

- **Provided Functions - Blocking**
  - **tc_blockio(TCDevice * device, TCCommBlocking blockflag);**
    - Sets the socket blocking type
    - Blocking
      - A connection that "blocks" on a read/write will wait until it has read/written all the data over its connection before proceeding. Blocking will force a system call and put itself to sleep and wait on the OS to wake it up.
    - No blocking
      - "Non-blocking" is asynchronous in nature and will read/write whatever it can offering no guarantee that it has finished.
    - Timed blocking
      - The timed block will block for a specified period of time, and give up if time expires. The "timed block" will consume CPU time as the read waits for data.
    - “All or nothing" blocking
      - The "all or nothing" block will not block until there is something to read. Once something is on the pipe, it will block indefinitely until it receives all data it expects. The "all or nothing" approach will consume CPU time as it waits.
  - **tc_set_blockio_timeout_limit(TCDevice *device, double limit) ;**
    - Sets the time a Timed blocking socket will wait for data

- **Provided Functions**
  - **tc_pending(TCDevice * device) ;**
    - Checks to see if any data is available for reading
  - **tc_isValid(TCDevice * device) ;**
    - Checks to see if the socket is still connected
  - **tc_error(TCDevice *device, int on_off) ;**
    - Turns on/off error messages from trickcomm activity
  - **tc_disconnect(TCDevice * device) ;**
    - Disconnects a socket

Server/Client Communication Sequence

Server

Initialize Listen Device
tc_init(TCDevice * listen_device) ;
Server sets up a socket called a "listen device" to listen for client connections. This listen is done on a predetermined port agreed upon by server and client. Returns when a connection is ready to made on the listen device

Accept Client
tc_accept(TCDevice *listen_dev , TCDevice *dev)
Accept client on listen device socket. Returns dev

Setup Blocking
tc_blockio(TCDevice *dev, TCCommBlocking block);
Set the socket to blocking/timed/non blocking/all or nothing

Read and Write
tc_read(TCDevice *dev, char *buffer, int size);
tc_write(TCDevice *dev, char *buffer, int size) ;
Send and receive data

Close Conection
tc_disconnect(TCDevice *dev);
Shutdown the socket

Client

Connect To Server
tc_connect(TCDevice *)
Client connects to server over socket. Connection is done over predetermined port.

Setup Blocking
tc_blockio(TCDevice *dev, TCCommBlocking block);
Set the socket to blocking/timed/non blocking/all or nothing

Read and Write
tc_read(TCDevice *dev, char *buffer, int size);
tc_write(TCDevice *dev, char *buffer, int size) ;
Send and receive data

Close Conection
tc_disconnect(TCDevice *dev);
Shutdown the socket

Trick Communications

Communication Sequence with tc_multiconnect

Server	Client
Connect to other side **tc_multiconnect(TCDevice *, char *t1, char *t2…) ;** Finds other machine/port number**.** Determines who is server/client. Connects and returns device	**Connect to other side** **tc_multiconnect(TCDevice *, char *t1, char *t2…) ;** Finds other machine/port number**.** Determines who is server/client. Connects and returns device
Setup Blocking **tc_blockio(TCDevice *dev, TCCommBlocking block);** Set the socket to blocking/timed/non blocking/all or nothing	**Setup Blocking** **tc_blockio(TCDevice *dev, TCCommBlocking block);** Set the socket to blocking/timed/non blocking/all or nothing
Read and Write **tc_read(TCDevice *dev, char *buffer, int size);** **tc_write(TCDevice *dev, char *buffer, int size) ;** Send and receive data	**Read and Write** **tc_read(TCDevice *dev, char *buffer, int size);** **tc_write(TCDevice *dev, char *buffer, int size) ;** Send and receive data
Close Conection **tc_disconnect(TCDevice *dev);** Shutdown the socket	**Close Conection** **tc_disconnect(TCDevice *dev);** Shutdown the socket

Variable Server

- **What is the variable server?**
  - **The variable server is a TCP/IP server (using Trickcomm) which runs in an asynchronous simulation thread. Clients may connect to the server and set/get simulation parameters.**

- **Why use it?**
  - **Nice for interactive GUIs**
  - **Simple asynchronous way to drive the simulation**
  - **Simple interface for probing states for graphics displays or stripcharts**
  - **Useful for crew training when an instructor needs to introduce a specific scenario on-the-fly**
  - **Useful for debugging**

- **Where to connect?**
  - **Clients connect to the port sys.exec.work.var_serve_listen_dev.port which is usually 7000.**
  - **Clients must be Trickcomm clients or emulate the handshaking.**

Variable Server

- **Variable server sends all commands through the input processor**
  - **All valid input processor commands available**
    - Setting variables with assignment statements (e.g. **`mystruct.x = 5.0;`**)
    - Command sim to run/freeze/dump checkpoint (sim_control panel)
    - Call jobs

Variable Server

- **In addition to handling all valid input processor commands, the variable server has specific commands to handle sending back data to the client at a semi-regular frequency**
  - **var_add <var_name> ;**
    - Add a variable name to the list to send back to the client
  - **get <var_name> ;**
    - Synonym for var_add
  - **var_remove**
    - Remove a variable name from the list
  - **var_send ;**
    - Send data back once instead of cyclically (typically used when “var_pause” is in effect)
  - **var_clear ;**
    - Clear list of variables
  - **var_cycle = <freq> ;**
    - Set frequency of data being sent
  - **var_pause ;**
    - Pauses variable server from returning data
  - **var_unpause**
    - Unpauses variable server
  - **var_exit**
    - Exits the variable server for that client

Variable Server

- **Returned values from variable server**
  - **Whitespace delimited ASCII**
  - **Asynchronously snapshotted from simulation**
  - **No guarantee of a regular return frequency**
  - **No guarantee of data homogeneity. Values can be from different frames of execution**

- **Variable server updated in Trick 07**
  - **Return values can be binary**
  - **Whole data structures can be returned**
  - **Synchronous option added where data is colleted and sent to the variable server clients with the main simulation loop**
    - Synchronous data is as regular as the sim keeps time
    - Guaranteed to be homogeneous

Variable Server

- **Other variable server commands**
  - **var_units <param_name> <units> ;**
    - Send back param_name with the specified units
  - **var_debug = <0,1,2,3> ;**
    - Print out increasing amounts of debugging information

- **New in Trick 07**
  - **var_ascii**
    - Send back data in ascii (default)
  - **var_binary**
    - Send back data in binary
  - **var_sync = (Yes|No) ;**
    - Send data back synchronously

Variable Server

- **All of Trick's runtime GUIs use the variable server**
  - **Sim control panel**
  - **Trick View (TV)**
  - **Malfunction Trick View (MTV)**
  - **Stripcharts**

Variable Server

- **Trick provides a Tcl/Tk package to connect and use the variable server**

```
# Tcl/Tk stub to include Simcom, the variable server comm package

# Add Trick's path the the search path for packages
global auto_path
set auto_path [linsert $auto_path 0 $env(TRICK_HOME)/bin/tcl]

# Trick's Simcom connect package
package require Simcom
```

- **Tcl/Tk variable server demonstration program**

```
#!/usr/bin/wish

# This is a small demonstration program to show how to connect to a
# simulation using the variable server.

# The GUI is a single slider bar.  The slider sets a relatively unused
# variable sys.exec.in.sync_port_offset when it changes in value
# and reads the value back from the sim at ~10Hz

# Add Trick's path to the search path for packages
global auto_path
set auto_path [linsert $auto_path 0 $env(TRICK_HOME)/bin/tcl]

# Trick's Simcom connect package
package require Simcom

# write out the return from the sim
proc get_sim_state { } {
        global my_sock

        gets $my_sock result
        puts "result = $result"
}
<continued on next page>
```

Variable Server

```
# callback to send the new value of the slider to the sim
proc update_slider { y } {
        global my_sock

        # y contains the value of the slider, send it to the sim
        Simcom::send_cmd $my_sock "sys.exec.in.sync_port_offset = $y ;"
        puts "sending sys.exec.in.sync_port_offset = $y"
}

######## start of main #########
# connect to sim on localhost
set my_sock [Simcom::connect localhost 7000]

# set up callback for reading results
fileevent $my_sock readable [list get_sim_state]

# uncomment to show variable_server debug messages
#Simcom::var_server_debug $my_sock 2

# set variable server to send back sync_port_offset.  Use default rate of ~10Hz
Simcom::send_cmd $my_sock ”var_add sys.exec.in.sync_port_offset ;"
Simcom::send_cmd $my_sock ”var_send ;"

# make a simple slider bar to set the sync_port_offset
scale .yslider -from 0 -to 200 -orient horizontal -variable y -command update_slider
pack .yslider -side top -pady 2m
```

Trick Real-Time

Real-Time Clocks

- **Trick's definition of a real-time simulation:**
  - **A simulation that can consistently and repetitively execute its scheduled math models to completion within some predetermined interval time frame for an indefinite period of time. This predetermined interval time frame is referred to as the real-time frame, and it is typically determined by real-world SW/HW or some output data requirement such as 30 Hz graphics or a hardware control frequency.**

- **By default, Trick uses the Linux system call clock_gettime() for its real-time clock reference - (legacy call was to gettimeofday())**

- **By default, during an under run the Trick executive spins on calls to clock_gettime() at the end of each real-time frame (rt_software_frame) to wait for real-time to catch up to sim**

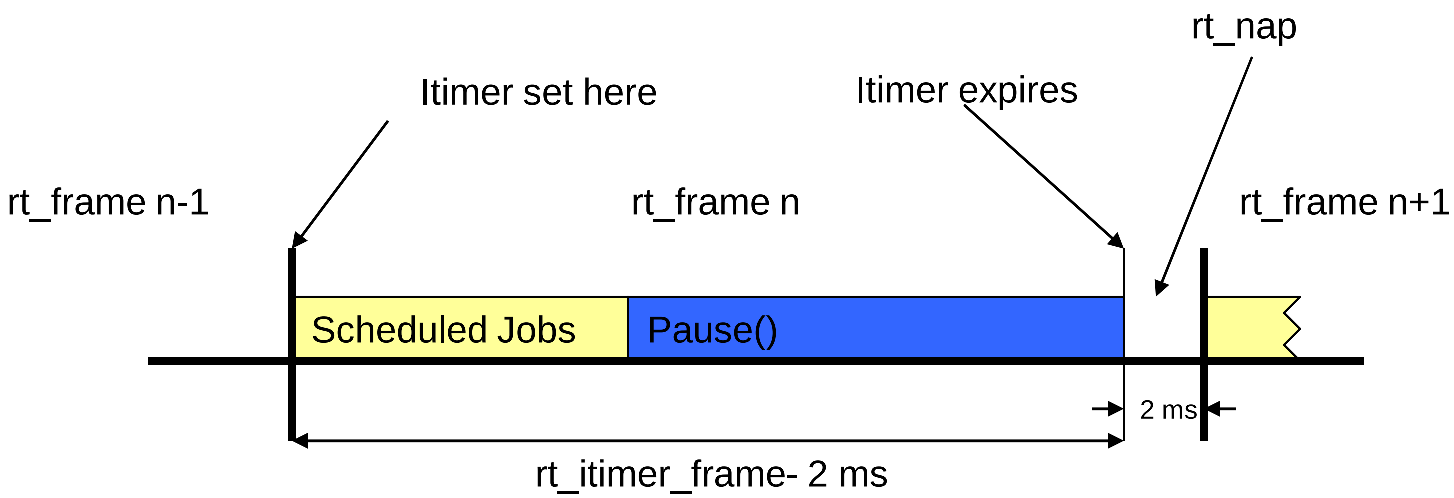
- **Let's look at Trick real-time control parameters**

```
sys.exec.in.rt_software_frame = <double> ;
    •defines real-time frame

sys.exec.in.rt_clock = (Gettimeofday|EXTERNAL);
    •Exec_Clock enumerated type
        •Gettimeofday – default system clock (uses
        clock_gettime())
        •EXTERNAL for external clock function
            •We will talk about this later
```

- **Let's look at Trick real-time control parameters -- itimers**

```
sys.exec.in.rt_itimer = (Yes|No) ;
    •Flag to enable itimers

sys.exec.in.rt_itimer_frame = <double> ;
    •defines interval for signal

sys.exec.in.rt_itimer_pause = (Yes|No) ;
    •enables pause() to release process control at end
    of RT scheduling frame before itimer signal (if not
    set, the executive spins on clock)

sys.exec.in.rt_nap = (Yes|No) ;
    •releases process control from executive while
    executive is waiting for resource after itimer
    signal
```

Realtime



rt_nap
Itimer set here
Itimer expires
rt_frame n-1
rt_frame n
rt_frame n+1
Scheduled Jobs
Pause()
2 ms
rt_itimer_frame- 2 ms

Real-Time

- **Variable for simulation RT performance analysis**

sys.exec.in.frame_log = (Yes|No) ;

- **Performance intrusive (increases executive overhead) flag that will log real-time analysis data for simulation**
  - **Note that job performance data will be very accurate**

- **Real-time process, processor and memory control variables**

sys.exec.in.rt_cpu_number[#] = <int> ;

- **assign CPU number to process/thread ID #, CPU numbers begin at 0**
- **# refers to the simulation process or thread ID; 0 is reserved for clock (SGI platform allows control of clock process), 1 is the main thread, 2 is the first child thread, n ... child threads**

sys.exec.in.rt_lock_to_cpu[#] = (Yes|No) ;

- **Yes to lock thread ID # to the CPU no. defined in rt_cpu_number above**

sys.exec.in.rt_lock_memory[#] = (Yes|No) ;

- **Yes to lock thread ID # into memory**

sys.exec.in.rt_semaphores[#] = (Yes|No) ;

- **Yes to use pthread mutexes for thread ID # synchronization (semaphore legacy syntax)**
- **No will use default of spinlocks (CPU hog)**

- **More real-time process/processor control variables**

sys.exec.in.rt_nond_pri[#] = (Yes|No) ;

- **Yes to set the non-degrading real-time priority (defined in rt_priority below) for thread ID #**

sys.exec.in.rt_priority[#] = <int> ;

- **Real-time priority integer for thread ID # (1 is the highest, 2 is the second highest, ..., n)**
  - » **Note that priority setting requires root privilege**

- **Several of the realtime features require the simulation to run as root**
- **Ways to give root privilege to sim**
  - **Run as root, or**
  - **Change owner of executable to root and set user id bit**
    - su to root or use sudo command (see man page) to give root privileges using chown and chmod commands

```
> chown root S_main_Linux_3.4_234.exe
> chmod 4775 S_main_Linux_3.4_234.exe
```

Interrupt Topics

- **Recommended services to turn off to protect against system interrupts and process context switching**
  - **Turn off everything but:**
    - ***acpid, anacron, atd, autofs, crond, cups, gpm, kudzu, lm_sensors, messagebus, netfs, nfslock, portmap, rawdevices, sshd, syslog, xinet.d***
    - ***IMPORTANT service to turn off!***
      - If left on, the **irqbalance** service can change processor interrupt assignments during simulation execution
- **Cat /proc/interrupts to see interrupt to processor mapping**
  - **This may help in your Trick process to processor real-time assignments for muti-processor platforms**
    - ***Assign Trick real-time process to processors that do not have interrupts assigned to them***

- **Set “isolcpus” option in “/boot/grub/menu.lst” to isolate CPU from UNIX scheduler**

```
title Fedora Core ISOLATE_CPU_1
 root (hd0,2)
 kernel /vmlinuz-2.6.12-1.1376_FC3 ro
 root=/dev/VolGroup00/LogVol00 rhgb quiet isolcpus=1
 initrd /initrd-2.6.12-1.1376_FC3.img
```

Interrupts cont.

- **Interrupts can be mapped to specific processors to optimize processor and hardware I/O performance**
  – **/proc/interrupts**
    - Dynamically updated file showing current interrupts and CPU mapping
  – **/proc/irq/***
    - Directories where * is the interrupt number that is shown in /proc/interrupts
    - Each irq directory contains a “special file” named smp_affinity which contains the bit wise number for the processor designation
    - smp_affinity files get reset every time you reboot so you need an initialization script to configure them
    - Interrupt balancing service (irqbalance) may need to be turned off
  – **In general, interrupts not relevant to your real-time process should be redirected away from your real-time processor**

MultiProcess (threaded) Simulations

- **Bring up trick_ui panel and select SIM_cannon_multi**
  - **CP SIM_cannon_multi**

- **Real-time features typically controlled from Modified_data/realtime.d**
    - Add to RUN_grav/input file just for this lesson

```
sys.exec.in.frame_log = Yes ;
sys.exec.in.rt_software_frame {s} = 0.01 ;
sys.exec.in.rt_itimer = Yes ;
sys.exec.in.rt_itimer_pause = Yes ;
sys.exec.in.rt_itimer_frame {s} = 0.01 ;
```

  – **10 millisecond frame (rt_software_frame) with itimers enabled**
    - Itimers and itimer pause keep sim from spinning on clock during underrun; prevents thrashing on single CPU machines
  – **frame_log parameter turns on real-time analysis logging**

- **Select RUN_grav and “Run” sim**

Multi-Processing

- **Bring up Data Products from trick_ui (DP button)**
- **Expand SIM_cannon_multi and select RUN_grav from sim pane and select DP_rt_frame from Data Product pane**
- **Launch single plot (square) from upper left menu bar**
  - **Real-Time Scheduling Frame**
    - ***Gives overrun/underrun in terms of overrun***
      - Overrun value is negative mirror of underrun
      - Add plotted overrun value to RT frame to get used processing time
        - » e.g.: if overrun value is -0.00995 for a 0.01 second frame, then only 0.00005 or 50 microseconds of the 10 ms frame was used
  - **Try zooming plot with middle mouse drag**
  - **Changing axis scale with left mouse drag**
  - **Right mouse resets plot**

Multi-Processing

- **Other Real-Time Scheduling Frame Plots**
  - **Frame Scheduling Time**
    - ***Executive overhead plot***
  - **Asynchronous Must Finish (AMF), Child Start and Complete, Depends On, and Master/Slave Sync Wait Time Plots**
    - Measures wait times
  - **Data Recording**
    - Measures time spent in data recording

- **Exit Real-Time Scheduling plot set from “Exit Plots” pop up dialog**

- **Deselect (dbl click) DP_rt_frame from Data Product pane**
- **Select (dbl click) DP_rt_itimer, DP_rt_jobs & DP_rt_timeline from Data Product pane**
- **Launch single plot (square) from upper left menu bar**
  - **Job Execution Times**
    - Each job plot point contains an accumulative sum of time the job has executed in each RT frame
    - Also gives executive overhead plot
  - **Execution Timelines**
    - Shows Job ID with respect to Real-Time (bar chart)
    - set sys.exec.in.frame_log_max_samples to increase logging time
- **Exit plots**

- **Since the sim still runs real-time, let's add a sleep(1) system call to the cannon_print_position2() function to induce an overrun**
  - **Use the trick_ui to edit cannon_print_position2.c**
- **Turn off itimers in RUN_grav/input to prevent interval timer signal from interrupting sleep**

```
sys.exec.in.rt_itimer = No ;
```

- **Make SIM_cannon_multi (No need to re-CP since S_define did not change)**

- **Select and execute SIM_cannon_multi with RUN_grav again**
  - **Notice constant overrun state**
  - **Freeze and shut down the simulation**

- **Bring up real-time frame plot (DP_rt_frame) again to show plot of continuous overrun (first deselect other plots)**

- **Now bring up DP_rt_jobs plot to show each job's execution performance for each RT frame**
  - **Notice 1 second overruns in cannon_print_position2() job plot**

- **Exit plots**

Multi-Processing

- **Let's use Trick pthreads multi-processing capability to solve overrun problem**
  - **Trick Multi-threaded simulations are called process groups**
  - **With trick_ui, edit S_define and add a C1 (Child thread 1) to the front of the *cannon_print_position2()* job call**
    - ***This will cause this job to run in parallel for the 0.1 second job frame***
  - **Re-CP SIM_cannon_multi**
  - **Turn itimers back on (threaded child processes do not receive itimer signals)**
    - ***Edit RUN_grav/input***
  - **Rerun RUN_grav**
  - **Note that the overruns did not go away**
    - What can we change about the parallel job to make the overruns go away?

Multi-Processing

- **Why didn't the overruns go away?**
  - **The Trick simulation executive still waits for each "synchronous" job at the end of each job frame before moving to the next job frame**

- **Edit the S_define file and change the cannon_print_position2 job class to asynchronous**
  - **This will cause the executive scheduler to not wait for job to complete and only reschedule after completion**
    - ***asynchonous_must_finish job class will cause the executive to wait for the job to finish if it is time to reschedule it***

- **After saving the S_define file, Re-CP, and rerun to see that overruns go away**
  - **Look at DP_rt_frame plots again**

- **Real-time parameters for cannon multi-process simulation**

```
#define CLOCK  0
#define PARENT  1
#define CHILD_THR  2
sys.exec.in.rt_nap = Yes;
sys.exec.in.rt_semaphores[CHILD_THR] = Yes;
sys.exec.in.rt_cpu_number[PARENT] = 0;
sys.exec.in.rt_cpu_number[CHILD_THR] = 0;
sys.exec.in.rt_lock_to_cpu[PARENT] = Yes;
sys.exec.in.rt_lock_to_cpu[CHILD_THR] = Yes;
sys.exec.in.rt_nond_pri[PARENT] = Yes;
sys.exec.in.rt_priority[PARENT] = 1;
```

- **Uncomment these real-time control parameters in RUN_grav/input**

- **Change owner of executable to root and set user id bit**
  - **su to root or use sudo command (see man page) to give root privileges using chown and chmod commands**

```
> cd ~/trick_sims/SIM_cannon_multi
> sudo su
> chown root S_main_${TRICK_HOST_CPU}.exe
> chmod 4775 S_main_${TRICK_HOST_CPU}.exe
```

  - **Configure simulation with root access**
  - **Rerun simulation**
  - **Look at RT plots and notice tight repeatable performance (should be no abnormal spikes)**

Master/Slave Import/Export

Distributed Processing

- **Trick uses a master/slave start up and sync design**
  - **Multiple S_defines, one master process and up to 16 slave processes**
    - ***Master launches slaves with remote (rsh) or secure (ssh) shell***
    - ***Slave process clocks are synced to the master over sockets***
      - Frequency of sync defined with sys.exec.in.rt_sync_frame
      - rt_sync_frame may be equal to or larger than rt_software_frame, but it must be a multiple of it
      - Itimer pause capability on slaved processes is deactivated because of master sync
      - External timers with pause capability can be used on slaves
    - ***By default, the master/slave processes will go to sleep (socket select call) waiting for handshake synchronization***
  - **Each master or slave process can also use Trick pthread or child multi-process capability (called a process group)**
  - **There is S_define syntax for importing and exporting data between master/slave process groups**
    - ***Trickcomm code is auto generated (uses tc_multiconnect)***

Distributed Processing

- **Let's look at a master/slave distributed sim across two S_defines, and import/export data between them**
  - **SIM_master_imp_exp and SIM_slave_imp_exp**
  - **Bring up the trick_ui and let's look at the two simulations**
    - ***First, look at the S_define for both sims***
      - Simple single sim_object with initialization and scheduled jobs
        - » Notice P# syntax for phased initialization job sequencing
        - » Notice import/export syntax - exported messages are attached to the job before its syntax and imported messages are attached to the job that immediately follows its syntax
    - ***Use trick_ui to view simple src and include files for simulations***
    - ***Finally, let's look at the input file setup for both simulations***
      - Notice the setup for master/slave configuration

- CP both SIM_master_imp_exp and SIM_slave_imp_exp sims with the trick_ui
  - Check for and resolve any errors

- Configure sim & shell to use secure shell for slave startup
  - Under SIM_master_imp_exp, edit RUN_Master/input
    - Add the line

```
sys.exec.in.remote_shell = TRICK_SSH ;
```

    - You may also need to add "./" to S_main_name to resolve the executable path if "." is not in your environment shell path

- **Continue secure shell configuration**
  - **From terminal and home directory:**

```
% ssh-keygen -t dsa
      (3 questions will be asked, hit return for all 3)
% cd ~/.ssh
% cp id_dsa identity
% cp id_dsa.pub authorized_keys
```

  - **Run ssh command twice to create known_hosts and test that ssh command works with no password prompt:**

```
% ssh localhost ls
<asks a question> Answer "yes"
<listing>
% ssh localhost ls
<listing given without questions>
```

- Add "sys.exec.in.echo_job = Yes;" to the input file for both Sim_master_imp_exp/RUN_Master and Sim_slave_imp_exp/RUN_Slave
  - This will show verbose print outs of job execution time
- Also, comment out the stop time in RUN_Master/input
  - We can use the sim control panel to freeze and shutdown
- Run Sim_master_imp_exp with RUN_Master from the trick_ui

- Notice that since sys.exec.in.enable_init_stepping is turned on in the master's input file, the control panel is waiting to step through the phased initialization class jobs
  - Step through phased init jobs to get to freeze, then run
  - Freeze and shutdown the sim after about 20 seconds

Distributed Processing

- View Status Messages pane in sim control panel for execution history of Master/Slave run

- Exit sim control and bring up DP from the trick_ui
- Select SIM_master_imp_exp/RUN_master from the Sims/Runs pane and DP_test from the Data Product pane

- Then click on the single plot icon in the upper left to bring up plot of import/export test data
  – Do these values make sense?

- Also look at the DP_rt plots

- A few words about dynamic connections between sims
  – Users can build their own connection managers with Trick by using child threads and asynchronous class jobs

Phasing & Job Times

- **Job Phasing (of regularly scheduled jobs)**
  - **S_define Syntax (P#) is identical to initialization job phasing**
  - **Initialization class job phases are synchronized across distributed simulations, but regularly scheduled phased jobs only perform a reorder sync within a single S_define**
  - **When regularly scheduled jobs use phasing (P#), the sort order becomes class, phase, and top down order in S_define**
  - **Phasing gives users another mechanism for scheduling order of like class jobs that is independent of top down order in S_define**

- **Start (or offset) and stop times can also be defined for each job entry in the S_define file**

- **See Trick user's guide for more info on job syntax in the S_define**

Trick Simulation Environment: Real World Real Time HIL Application

Alex Lin (NASA/ER7)

February 14th-16th, 2006

MRMDF Introduction

- The Multi-use Remote Manipulator Development Facility (MRMDF) is an International Space Station ground facility supporting
  - Astronaut training
  - Procedures development
  - Engineering evaluations associated with the operations of the Space Station Remote Manipulator System (SSRMS)

- Full scale functional replica of the SSRMS designed to operate in the 1-g environment
  - Hydraulically actuated arm
  - Arm control and safety electronics
  - Functional End Effector
  - Test Director Console
  - SSRMS simulation

MRMDF Introduction

- **Part 1**
  - **Machine hardware and software configuration**
  - **MRMDF real-time parameter settings**

- **Part 2**
  - **Real time performance analysis using Data Products**

MRMDF Introduction

- **MRMDF is a distributed system using 5 computers**
  - **Test Directors Console (TDC)**
    - Desktop Dell 2GHz Xeon workstation
    - Test Director interface
  - **Facility Control System Electronics (FCSE)**
    - VMIC 7750 VME Single Board computer 1.26Ghz Pentium III
  - **2 Manipulator Control System Electronics (MCSEA) and (MCSEB)**
    - VMIC 7750 VME Single Board computer 1.26Ghz Pentium III
  - **SSRMS simulation or Basic Operational Robotics Instructional System (BORIS) simulation**
    - Desktop Dell Dual 2GHz Xeon workstation
    - Astronaut/Trainer interface

- **Facility is on an isolated 1Gbs network**
  - **All network cards are 100Mbs**

MRMDF Machine Configuration

- **All computers running RedHat Enterprise Linux 4 (RHEL4)**
  - **Non-updated straight from the RHEL4 CD installation**
  - **Embedded computers set to boot to run level 3 (multi-user text only)**
  - **TDC and BORIS computers set to run level 5**

- **Some services turned off because they are non-essential or may hurt real-time performance**
  - **irqbalance**
  - **cpuspeed**
  - **sendmail**

MRMDF Machine Configuration

- **Output of services still running on embedded computers**

```
> chkconfig --list | grep "3:on" | sort
acpid           0:off   1:off   2:off   3:on    4:on    5:on    6:off
anacron         0:off   1:off   2:on    3:on    4:on    5:on    6:off
atd             0:off   1:off   2:off   3:on    4:on    5:on    6:off
autofs          0:off   1:off   2:off   3:on    4:on    5:on    6:off
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
cups            0:off   1:off   2:on    3:on    4:on    5:on    6:off
gpm             0:off   1:off   2:on    3:on    4:on    5:on    6:off
kudzu           0:off   1:off   2:off   3:on    4:on    5:on    6:off
lm_sensors      0:off   1:off   2:on    3:on    4:on    5:on    6:off
messagebus      0:off   1:off   2:off   3:on    4:on    5:on    6:off
netfs           0:off   1:off   2:off   3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
nfslock         0:off   1:off   2:off   3:on    4:on    5:on    6:off
portmap         0:off   1:off   2:off   3:on    4:on    5:on    6:off
rawdevices      0:off   1:off   2:off   3:on    4:on    5:on    6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
syslog          0:off   1:off   2:on    3:on    4:on    5:on    6:off
xfs             0:off   1:off   2:on    3:on    4:on    5:on    6:off
xinetd          0:off   1:off   2:off   3:on    4:on    5:on    6:off
```

- **Output of "lsmod" command**

```
> lsmod
Module                  Size  Used by
nfs                   218309  0
lockd                  63977  1 nfs
sunrpc                157093  3 nfs,lockd
vme_vmitmrf            11752  0
vme_universe          108708  0
md5                     4033  1
ipv6                  232705  12
parport_pc             24705  0
lp                     12077  0
parport                37129  2 parport_pc,lp
autofs4                23237  1
i2c_dev                11329  0
i2c_core               22081  1 i2c_dev
dm_mod                 54741  0
uhci_hcd               31065  0
hw_random               5845  0
e100                   39493  0
mii                     4673  1 e100
floppy                 58481  0
ext3                  116809  1
jbd                    71257  1 ext3
```

- **irqbalance service turned off the BORIS computer**
  - **irqbalance is a service that tries to spread interrupt handling to all available processors**
    - Reassigns interrupts to specific processors
    - If a simulation is “burning” all CPU cycles, any interrupt assigned to that CPU will not be serviced. The machine will appear “locked up”

MRMDF Machine Configuration

- **Output of "cat /proc/interrupts" with irqbalance on**

```
> cat /proc/interrupts
           CPU0       CPU1
  0:  686551361  690211955    IO-APIC-edge  timer
  1:      65200      64966    IO-APIC-edge  i8042
  8:          1          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 12:          0          0    IO-APIC-edge  i8042
 14:     273884     283455    IO-APIC-edge  ide0
 15:   10380479   10961469    IO-APIC-edge  ide1
169:    8677476    8633408   IO-APIC-level  uhci_hcd
177:       7127       7074   IO-APIC-level  Intel 82801BA-ICH2
185:   32970665          0   IO-APIC-level  uhci_hcd, eth0
193:   55050079   55390831   IO-APIC-level  ohci1394, nvidia
201:         30          0   IO-APIC-level  aic7xxx
209:    1678484    1679598   IO-APIC-level  aic7xxx
NMI:          0          0
LOC: 1376788791 1376785050
ERR:         10
MIS:          0
```

- **Output of “cat /proc/interrupts” with irqbalance off on the BORIS computer**

```
> cat /proc/interrupts
            CPU0       CPU1
  0:   451274705          0    IO-APIC-edge  timer
  1:         552          0    IO-APIC-edge  i8042
  8:           1          0    IO-APIC-edge  rtc
  9:           0          0   IO-APIC-level  acpi
 12:      506838          0    IO-APIC-edge  i8042
 14:       79358          0    IO-APIC-edge  ide0
 15:     4058658          0    IO-APIC-edge  ide1
169:           0          0   IO-APIC-level  uhci_hcd
177:        1278          0   IO-APIC-level  Intel 82801BA-ICH2
185:     9368789          0   IO-APIC-level  eth0, uhci_hcd
193:    27195681          0   IO-APIC-level  nvidia
NMI:           0          0
LOC:   451282615  451282536
ERR:           2
MIS:           0
```

MRMDF Trick Real-time Features

- **All machines in the MRMDF use a modified Trick 05.6.1**
  - **All MRMDF specific changes merged back into Trick 05.7.0**

- **MRMDF uses many of the real-time features in Trick**
  - **Scheduling**
    - 30Hz sync frame (0.033330 sec)
    - 300Hz real time loop (0.003333 sec)
  - **Distributed application across 5 computers**
  - **Master/Slave synchronization**
  - **Clock/timers - External timers**
  - **Real-time controls and parameters**
  - **Multi-threaded applications**

MRMDF Real-time Scheduling

- **Scheduling**
  - **30Hz sync frame and 300Hz real time loop**
  - **Paritial S_define listing from an MCSE**

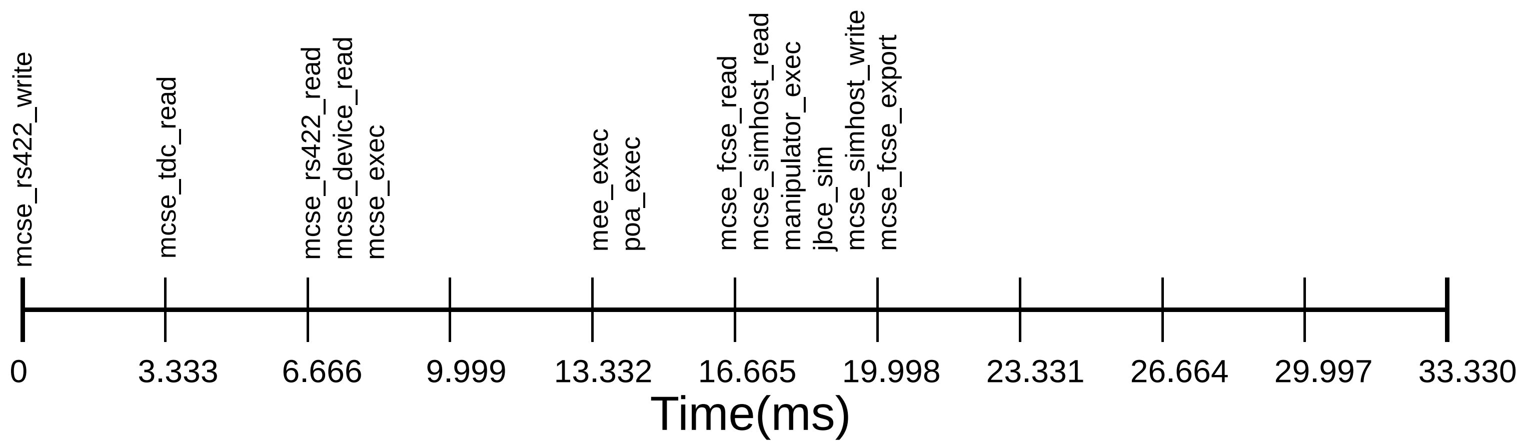
```
#define HZ30   0.033330   /*--- MRM SW FREQUENCY CYCLE TIME    ---*/

sim_object {

<data structures and init jobs>

(HZ30)           mcse/device: mcse_rs422_write(…) ;
(HZ30,0.003333) mcse/comm: mcse_tdc_read(…) ;
(HZ30,0.006666) mcse/device: mcse_rs422_read(…);
(HZ30,0.006666) mcse/device: mcse_device_read(…) ;
(HZ30,0.006666) mcse/mrm: mcse_exec(…) ;
(HZ30,0.013332) mcse/mee: mee_exec(…);
(HZ30,0.013332) mcse/poa: poa_exec(…);
(HZ30,0.016665) mcse/comm: mcse_fcse_read(…) ;
(HZ30,0.016665) mcse/comm: mcse_simhost_read(…) ;
(HZ30,0.016665) mcse/manipulator: manipulator_exec(…);
(HZ30,0.016665) mcse/jbce_sim: jbce_sim(…);
(HZ30,0.016665) mcse/comm: mcse_simhost_write(…) ;
(HZ30,0.016665) mcse/comm: mcse_fcse_export(…) ;

} mcse ;
```

MRMDF Real-time Scheduling

- **To set the frequencies in the S_define file is not enough to run real-time. Real time performance is greatly shaped by one parameter in the input file.**

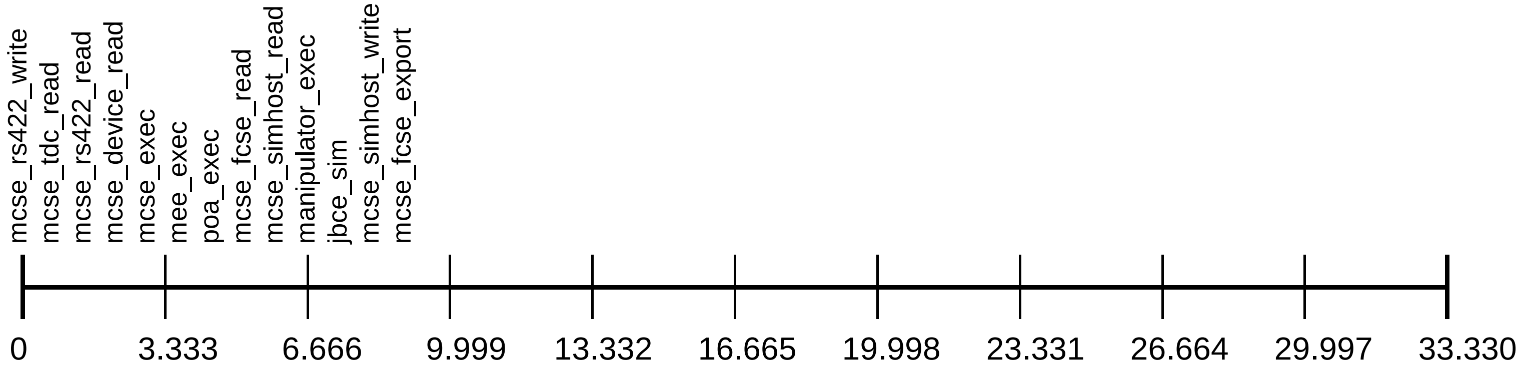
```
sys.exec.in.rt_software_frame {s} = 0.0033330 ;
```

  - **Setting rt_software_frame to 3.333ms tells Trick to schedule jobs for the next 3.333ms to run and wait (if underrunning) for the next frame**



MRMDF Real-time Scheduling

– **If we had set rt_software_frame to 33.33ms, all jobs within each 33.33ms would be scheduled to run in succession.**

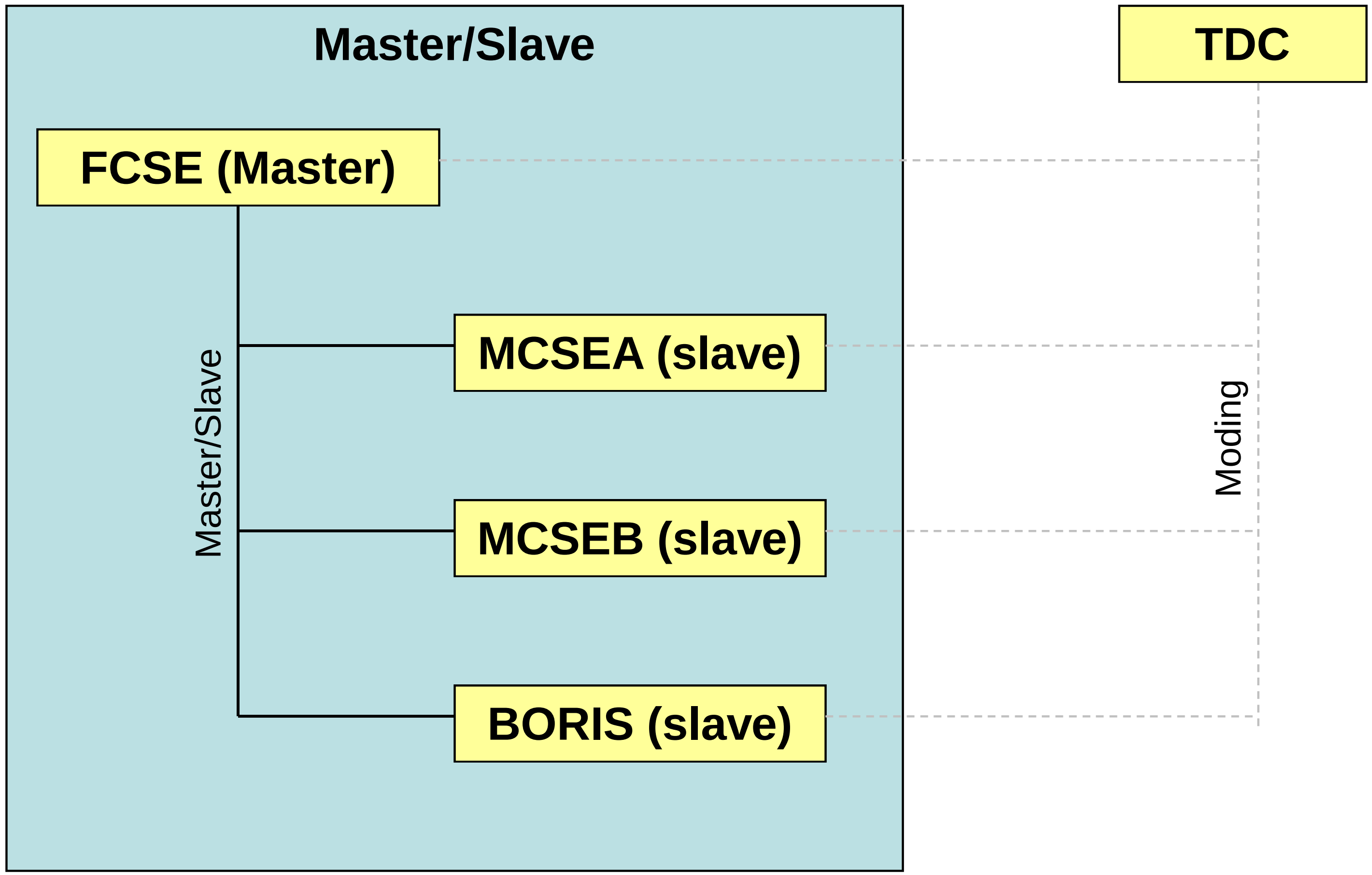


MRMDF Master/Slave Configuration

- **Master/Slave synchronization**
  - **MRMDF requires 4 computers to communicate synchronously**
  - **To ensure that all simulations are working with to the same clock we use Trick's Master/Slave synchronization capabilities**
  - **4 of the 5 computers participate in the Master/Slave setup**
    - FCSE Master
    - MCSEA Slave
    - MCSEB Slave
    - SSRMS/BORIS Slave

- **The 5th computer, the TDC is asynchronously connected to all systems**
  - **Controls simulation mode of the system**

MRMDF Master/Slave configuration



Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]         = Yes ;
sys.exec.in.slaves[num_slaves].machine_name    = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path        = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]  = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]   = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Turns on Master/Slave sync. Choices are**
    - Master_sync
    - Slave_Sync
    - No_sync

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]           = Yes ;
sys.exec.in.slaves[num_slaves].machine_name      = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path          = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]    = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Which remote shell to use to start slave simulations. Choices are**
    - TRICK_SSH
    - TRICK_RSH

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]         = Yes ;
sys.exec.in.slaves[num_slaves].machine_name    = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path        = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]  = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Activate this slave at initialization?**
    - Yes = Master will use a remote shell to automatically start slave simulation
    - No = Master will not start the slave, but will synchronize with the slave when connected

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]   = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Machine name to start the slave on.**
    - In this case ${MCSEA_HOST} is an environment variable

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]  = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Full path to slave simulation directory on remote machine**
    - In this case ${MCSEA_SIM_DIR} is an environment variable

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync   = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]  = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Relative path from slave simulation directory to run directory**
    - In this case ${MCSEA_RUN_DIR} is an environment variable

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]   = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Number of slaves the master will synchronize**
    - In MRMDF num_slaves = 3

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]          = Yes ;
sys.exec.in.slaves[num_slaves].machine_name     = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path         = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]   = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Will the Master wait at time=0 for all slaves to start before proceeding**

Master/Slave Configuration

- **Partial input file detailing Master/Slave activation and synchronization (FCSE)**

```
/* Activation */
sys.exec.in.ms_sync  = Master_sync ;
sys.exec.in.remote_shell = TRICK_SSH ;

int num_slaves = 0 ;

sys.exec.in.activate_slave[num_slaves]         = Yes ;
sys.exec.in.slaves[num_slaves].machine_name    = "${MCSEA_HOST}" ;
sys.exec.in.slaves[num_slaves].sim_path        = "${MCSEA_SIM_DIR}" ;
sys.exec.in.slaves[num_slaves].S_main_args[0]  = "${MCSEA_RUN_DIR}/input" ;
num_slaves++ ;

<similar code for mcse_b and boris>

sys.exec.in.slave_cnt = num_slaves ;

/* Synchronization */
sys.exec.work.slave_sync_at_init = Yes ;
sys.exec.in.sync_error_terminate = No ;
```

  - **Will the simulation terminate if loss of sync connection detected**
    - No = continue to run, but as a standalone sim.
    - In MRMDF, Loss of sync will send the facility to “safe” mode through a user model

Real-time Input Parameters

- MRMDF runs with 33.33ms major frame with 3.333ms minor frame
- When running we want executables to run at maximum non-degrading real-time priority

Real-time Input Parameters

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

  - **Allows the sim to sleep when we need to wait for events**
    - We want the simulation to “burn” for instant response

Real-time Input Parameters

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

  – **Don’t use an itimer to check for frame overruns**
    - Before Linux 2.6 kernels and Trick 5.6, using itimers with frames under 10ms was not possible.

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

  - **Allows the sim to sleep during underrunning frames**
    - rt_itimer_pause not applicable if rt_itimer is off

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

  – **Setting rt_software_frame to 3.333ms tells Trick to schedule jobs for the next 3.333ms to run and wait (if underrunning) for the next frame**

Real-time Input Parameters

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_exttimer = Yes ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

- **How often we check the realtime status with the external timer**

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

  - **Maximum time to wait for synchronization response**
    - If no response after 50ms, then the master breaks the sync connection and sends the facility to “safe”

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

– **Lock all memory in RAM. Cannot be paged to swap file.**
  - Must be root to use this parameter

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

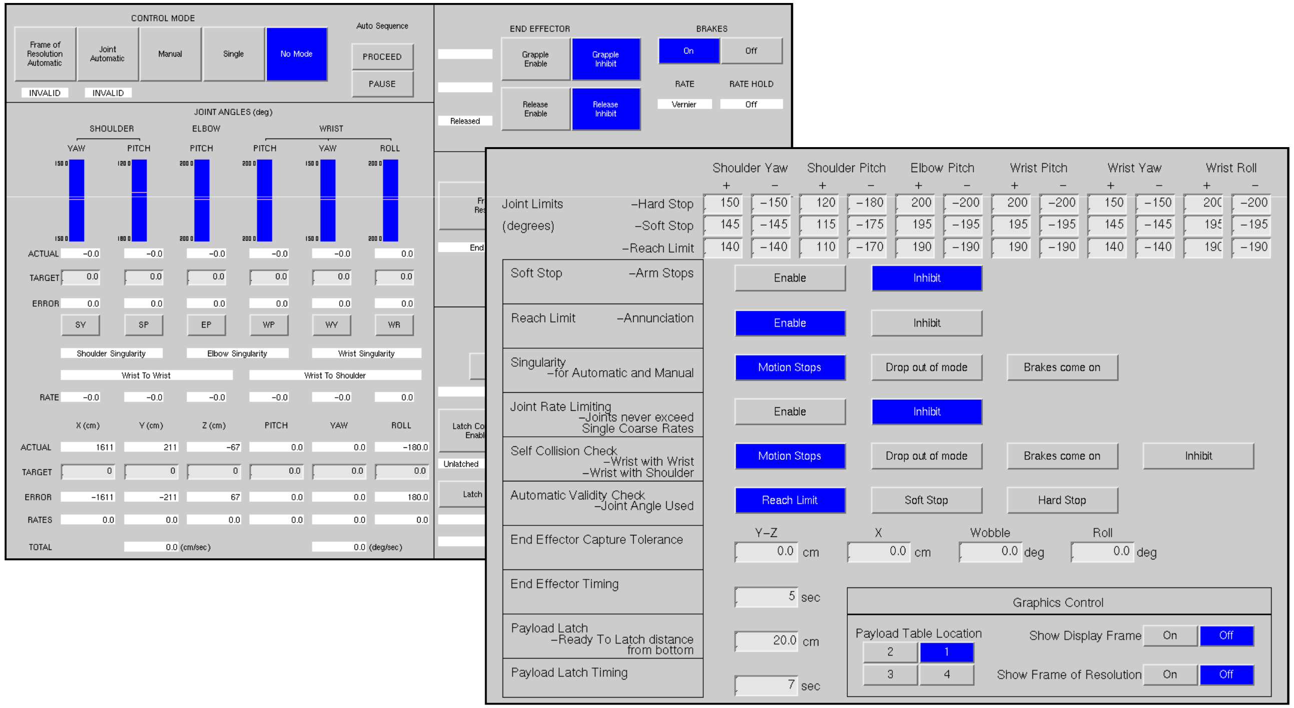
  – **Set the simulation to run with a non-degrading real-time priority.**
    - Must be root to use this parameter

- **Partial input file for real-time input parameters (FCSE)**

```
#define SW_FRAME 0.0333300
#define RT_FRAME 0.0033330

    /* SIM-TO-WALL-CLOCK SYNCHRONIZATION */
sys.exec.in.rt_nap = No ;
sys.exec.in.rt_itimer = No ;
sys.exec.in.rt_itimer_pause = No ;
sys.exec.in.rt_software_frame {s} = RT_FRAME ;
sys.exec.in.rt_sync_frame     {s} = SW_FRAME ;
sys.exec.in.sync_wait_limit  {s} = 0.050 ;

    /* MAXIMUM NON-DEGRADING PRIORITY */
sys.exec.in.rt_lock_memory[1] = Yes ;
sys.exec.in.rt_nond_pri[1]    = Yes ;
sys.exec.in.rt_priority[1]    = 1 ;
```

- **Sets the priority.**
  - Setting rt_priority = 1 will set the process to run at maximum priority on any platform
  - Warning: Setting rt_priority = 1 on a single processor computer with no “napping” will completely shut out all shells and input devices, i.e. mouse, keyboard

- **BORIS is an all-in-one generic robotics trainer**
  - **Arm simulation**
  - **Operator GUIs built into the simulation**



- **The BORIS simulation has 2 requirements forcing a multi-threaded design**
  - **Communication deadlines to meet with the facility**
  - **Handling updates to the GUIs for operator**

- **X-Windows event handling**
  - **Button presses**
  - **Text field updates**
  - **Window scrolling**
  - **Window resizing…**
  - **Some actions can generate thousands of events requiring seconds to complete**

- **BORIS requires a dual processor computer**
  - **Use one processor for keeping the simulation synchronized with the rest of the facility**
  - **Use the second to handle X-window updates through a "child" process.**
    - Child terminology held over from when Trick used parent/child forking and execing to run multiprocess applications

- **Assigning job to child thread in S_define:**

```
C1 (0.01, asynchronous) xgrt: grt_event_loop(…);
```

  - **C1 = Assign the job to run in the first child process**
  - **asynchronous = do not worry about job completion time**
  - **0.1 sec. cycle time = when job is finished reschedule the job to run the next 0.1 second boundary**
    - This particular job never returns so the cycle time is irrelevant

- **Data locks or mutexes are not provided by Trick**
  - **BORIS' main thread and child thread both make X function calls**
  - **The application will core dump if more than one thread tries to make X calls simultaneously**
  - **Added a pthread mutex that locks out other threads when making X calls**
  - **For this child job we also added code to ensure that no X updates occur once we are past 80% finished in the frame**
    - This is to make sure that we finish all X updates in time so the main thread will only have to wait a minimal time to acquire the mutex.

- **Excerpt from child process**

```
while (1) {
   /* we do not want to update the GUI if we are nearing time to call
     grt_update_values,  don't update if we are over 80% through the frame */
   rem = fmod ( exec_get_sim_time() , job_call_time ) ;
   if ( rem != 0.0 && rem < (job_call_time * 0.80)) {
       /* lock the mutex if we can */
       if ( pthread_mutex_trylock(&grt_info->x_lock) == 0 ) {
           /* dispatch 5 events */
           for ( ii = 0 ; ii < 5 ; ii++ ) {
               if ( XtAppPending( grt_if->app_context ) & XtIMXEvent ){
                  XtAppNextEvent( grt_if->app_context , &next_event ) ;
                  XtDispatchEvent( &next_event ) ;
               }
           }
           /* unlock the mutex so grt_update_values can run */
           pthread_mutex_unlock(&grt_info->x_lock);
       }
   }
   else {
       /* we are nearing or on the cycle to run grt_update_values,
          sleep a little to allow grt_update_values to run */
       usleep(100) ;
   }
}
```

- **Partial input file parameters associated with real-time (BORIS)**

```
#define SW_FRAME 0.0333300

sys.exec.in.rt_itimer       = No ;
sys.exec.in.rt_itimer_pause = Yes ;
sys.exec.in.rt_nap          = No ;
sys.exec.in.rt_itimer_frame   {s} = SW_FRAME ;
sys.exec.in.rt_software_frame {s} = SW_FRAME ;
sys.exec.in.rt_enable_clock_reset = Yes ;
sys.exec.in.sync_error_terminate = 0 ;

sys.exec.in.rt_lock_to_cpu[1] = Yes ;
sys.exec.in.rt_cpu_number[1] = 1 ;
sys.exec.in.rt_lock_memory[1]  = Yes ;
sys.exec.in.rt_nond_pri[1] = Yes ;
sys.exec.in.rt_priority[1] = 1 ;
sys.exec.in.rt_lock_to_cpu[2] = Yes ;
sys.exec.in.rt_cpu_number[2] = 0 ;
```

- **Partial input file parameters associated with real-time (BORIS)**

```
sys.exec.in.rt_lock_to_cpu[1] = Yes ;
sys.exec.in.rt_cpu_number[1] = 1 ;
sys.exec.in.rt_lock_memory[1]  = Yes ;
sys.exec.in.rt_nond_pri[1] = Yes ;
sys.exec.in.rt_priority[1] = 1 ;
sys.exec.in.rt_lock_to_cpu[2] = Yes ;
sys.exec.in.rt_cpu_number[2] = 0 ;
```

  - **rt_cpu_number[1] = Master thread process. We assign this to processor 1 where there are no interrupts being handled. We want to make sure that the Child thread is assigned to the other processor**

```
> cat /proc/interrupts
           CPU0       CPU1
  0:  451274705          0    IO-APIC-edge  timer
  1:        552          0    IO-APIC-edge  i8042
  8:          1          0    IO-APIC-edge  rtc
  9:          0          0   IO-APIC-level  acpi
 12:     506838          0    IO-APIC-edge  i8042
 14:      79358          0    IO-APIC-edge  ide0
 …
```

MRMDF Multithreaded Applications

- **Using ps to confirm we are locked down on the correct processor**
  - **Use ps arguments to show extra information**

```
> ps -eLo pid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm

  PID  CLS RTPRIO  NI PRI PSR %CPU STAT WCHAN          COMMAND
 7789  FF      99   - 139   1  1.7 SLl  pause          S_main_Linux_3.
 7789  TS       -   0  20   0  0.0 SLl  -              S_main_Linux_3.
```

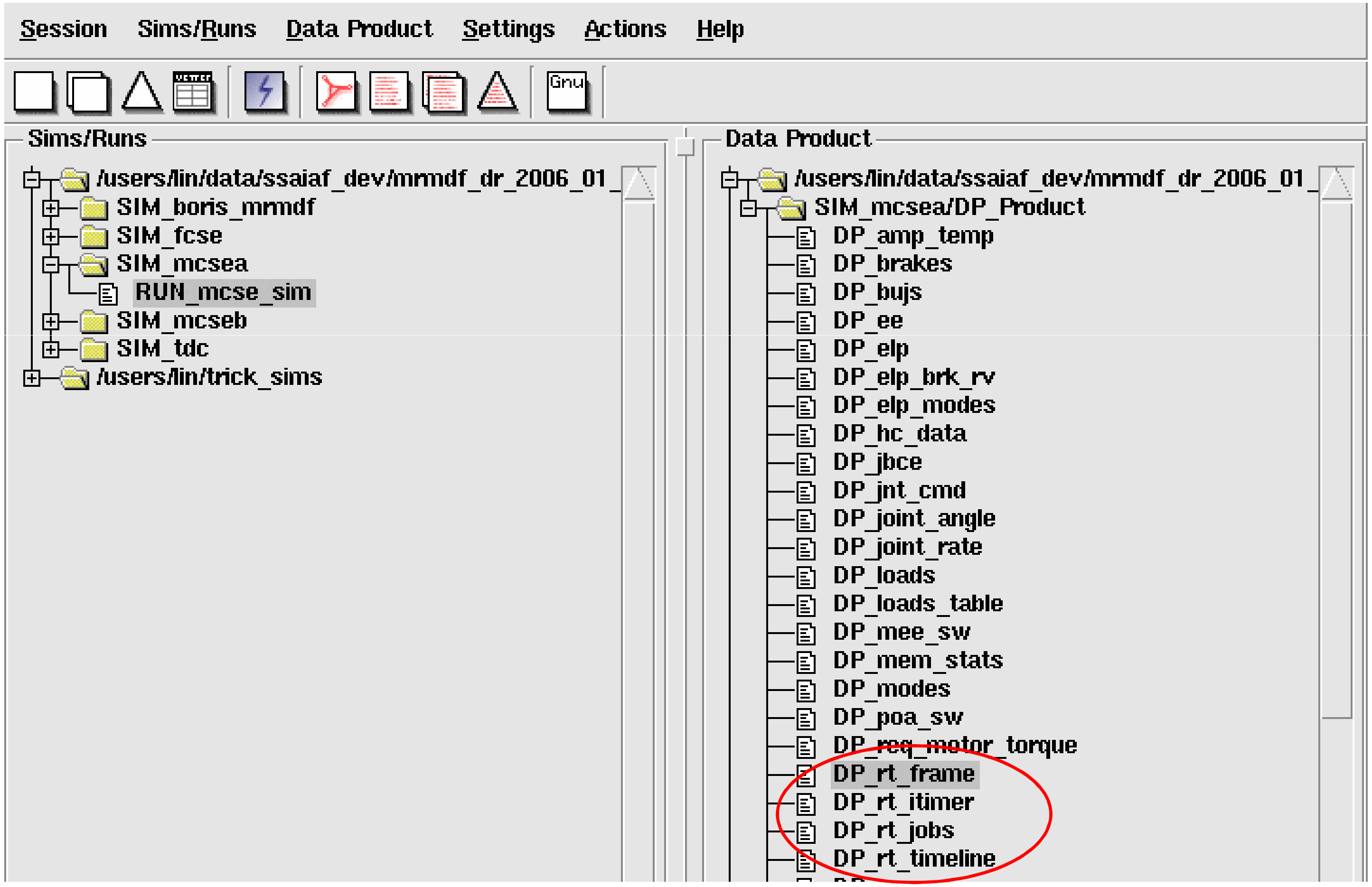
- **PID**
  - **Process ID**
  - **All threads of same sim will show same PID**
- **CLS**
  - **Scheduling class of the process**
    - TS = SCHED_OTHER
    - FF = SCHED_FIFO (realtime scheduler)
- **RTPRIO**
  - **Realtime priority**
    - For Linux 99 = highest priority
- **PSR**
  - **Processor that process is currently assigned to.**
- **%CPU**
  - **Percentage of that individual CPU used**

- **Use data products to**
  - Interpret and analyze real-time performance graphs
  - Pinpoint functions causing overruns

- **When things go right:**
  - **The printout at sim termination shows zero overruns:**

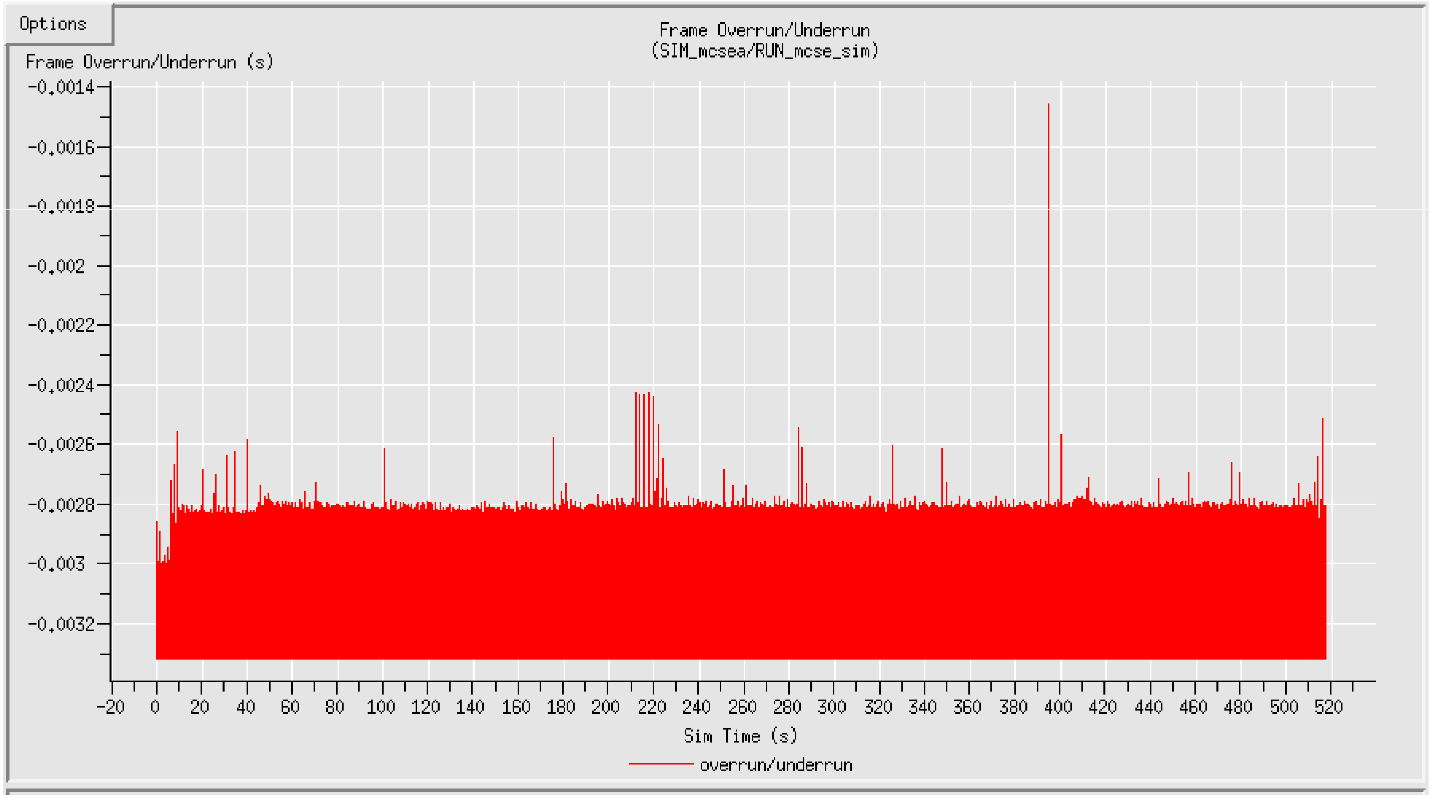
```
SIMULATION TERMINATED IN
  PROCESS: 1
  JOB/ROUTINE: 1/master.c
DIAGNOSTIC:
Sim Control Shutdown.

LAST JOB CALLED: mcse.mcse_tdc_read(&mcse.mcse)
              TOTAL OVERRUNS:            0
PERCENTAGE REALTIME OVERRUNS:        0.000%


       SIMULATION START TIME:        0.000
        SIMULATION STOP TIME:      517.755
     SIMULATION ELAPSED TIME:      517.755
         ACTUAL ELAPSED TIME:      517.755
        ACTUAL CPU TIME USED:       57.649
    SIMULATION / ACTUAL TIME:        1.000
       SIMULATION / CPU TIME:        8.981
  ACTUAL INITIALIZATION TIME:        0.000
     INITIALIZATION CPU TIME:        0.128
```

- **Each Trick sim automatically generates 4 DP files dedicated for real-time analysis: DP_rt_frame, DP_rt_itimer, DP_rt_jobs, and DP_rt_timeline**

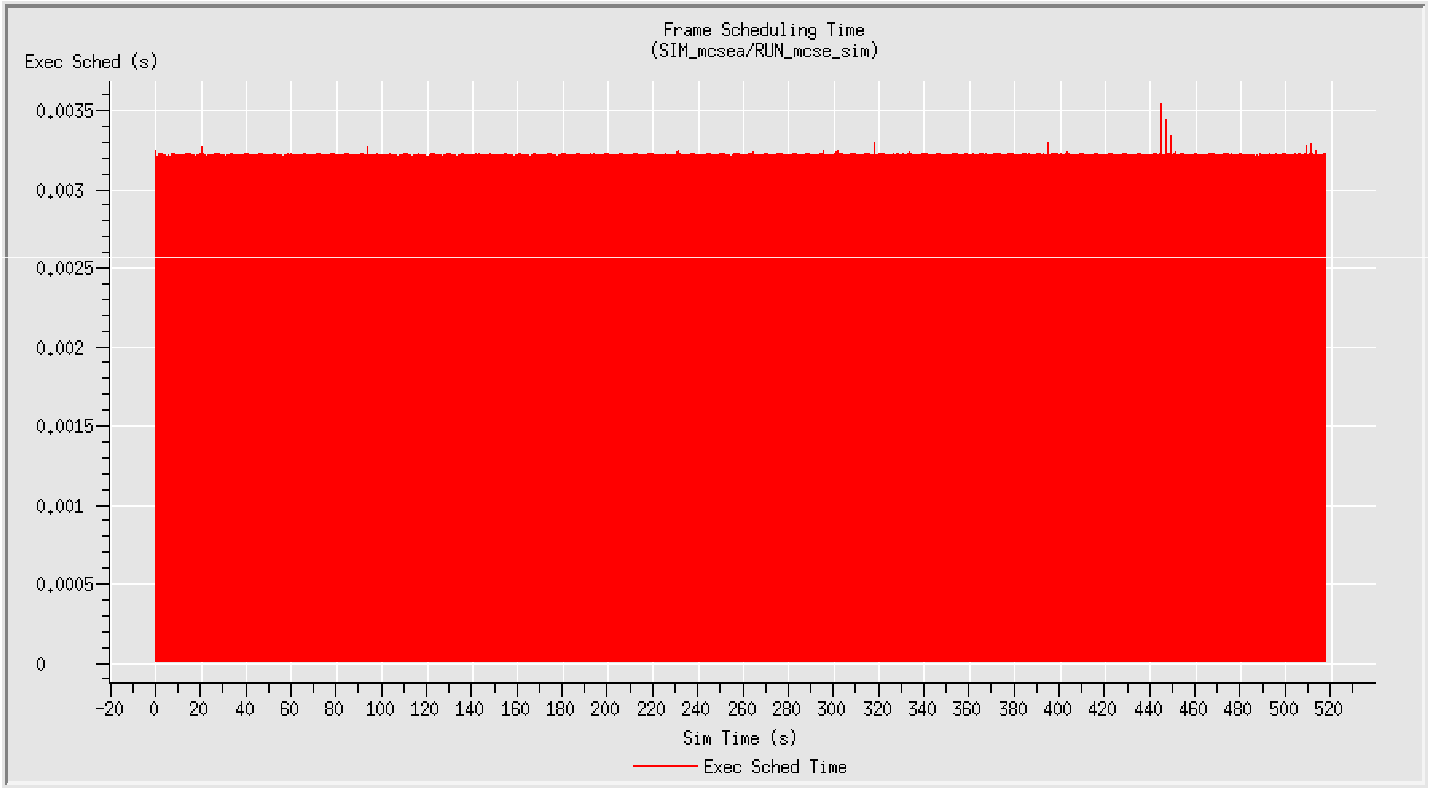


MRMDF Real-Time Performance Analysis

- **Selecting the DP_rt_frame graph brings up 4 pages**
  - **First graph on the first page shows Frame Overrun/Underrun times**
  - **Negative points are underruns, minimum value = -sys.exec.in.rt_software_frame (-3.333ms)**
  - **Positive points are overruns**
  - **Spike at time ~= 395, but not an overrun**

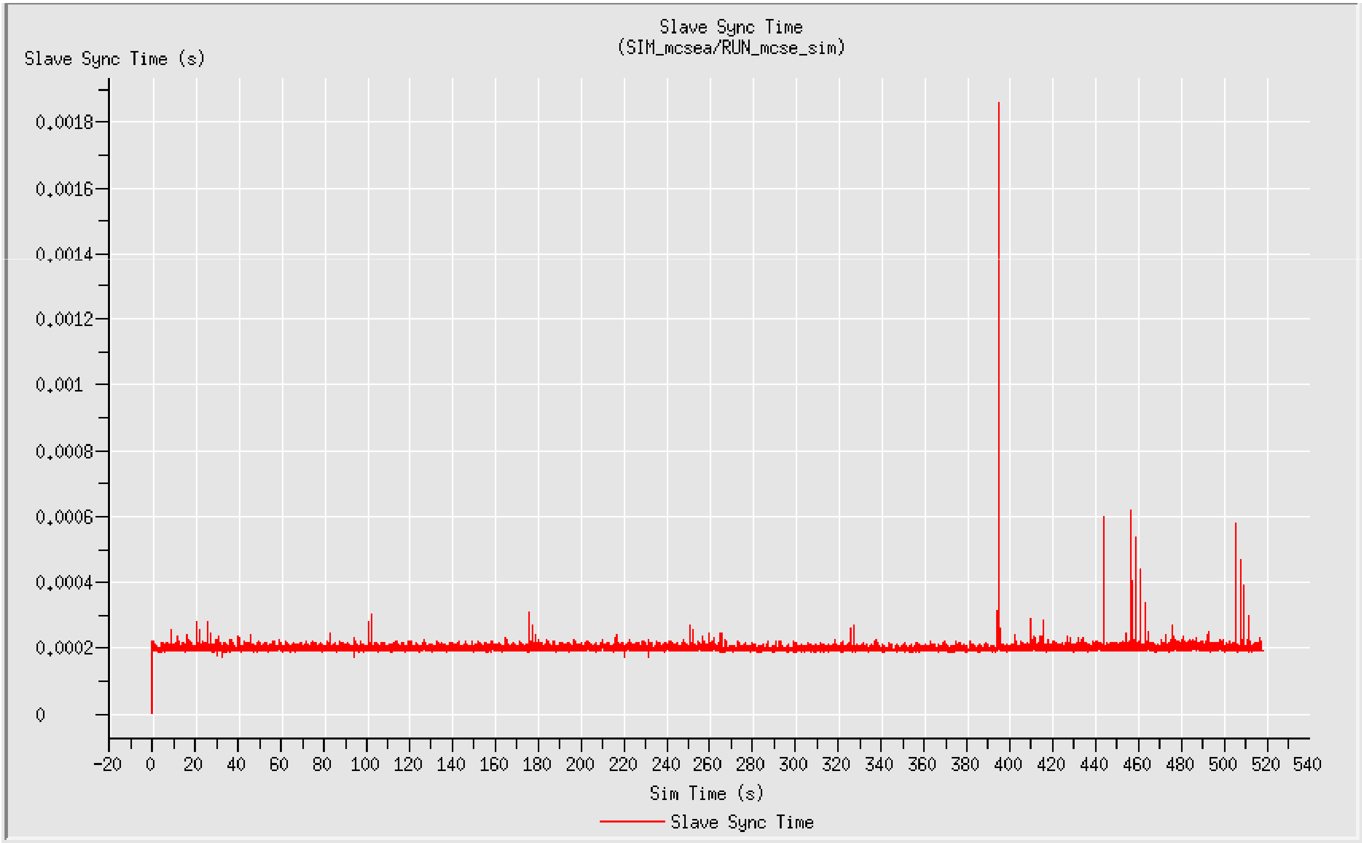


- **The bottom graph shows how long the Trick executive took to execute each frame**

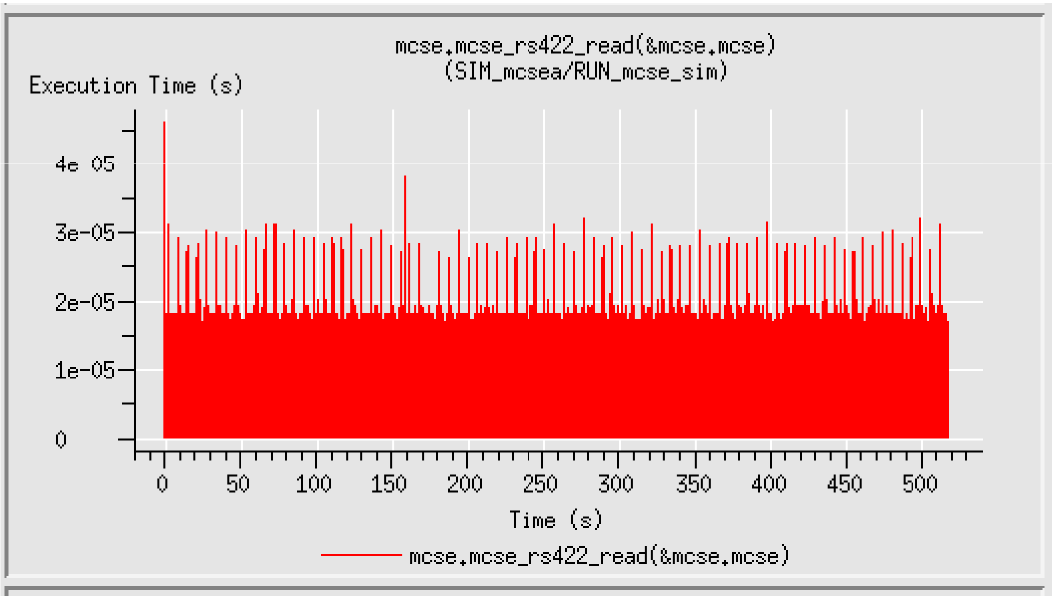


MRMDF Real-Time Performance Analysis

- **Another graph of interest in DP_rt_frame is the Master/Slave Sync Time graph**
  - **Normally takes 200us to sync this sim to the Master**
  - **Corresponding spike at time ~= 395**

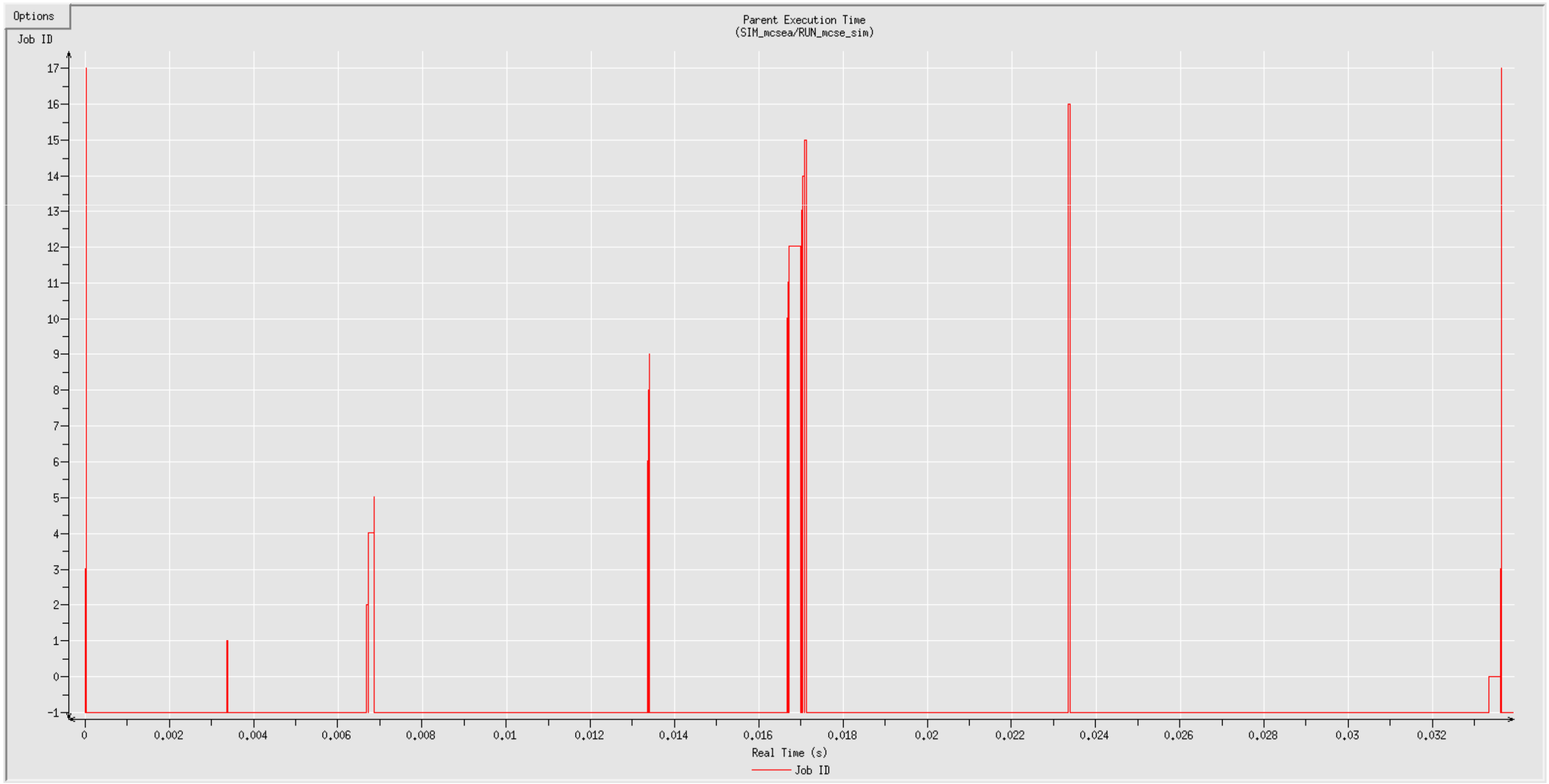


- **Selecting the DP_rt_jobs file will bring up multiple pages detailing how long each job took in each frame**
  - **On average mcse_rs422_read takes 30us or less**



MRMDF Real-Time Performance

- **Selecting the DP_rt_timeline file will bring up a single graph plotting job currently executing vs. real-time**
  - **Stretched to show first 33.33ms frame**

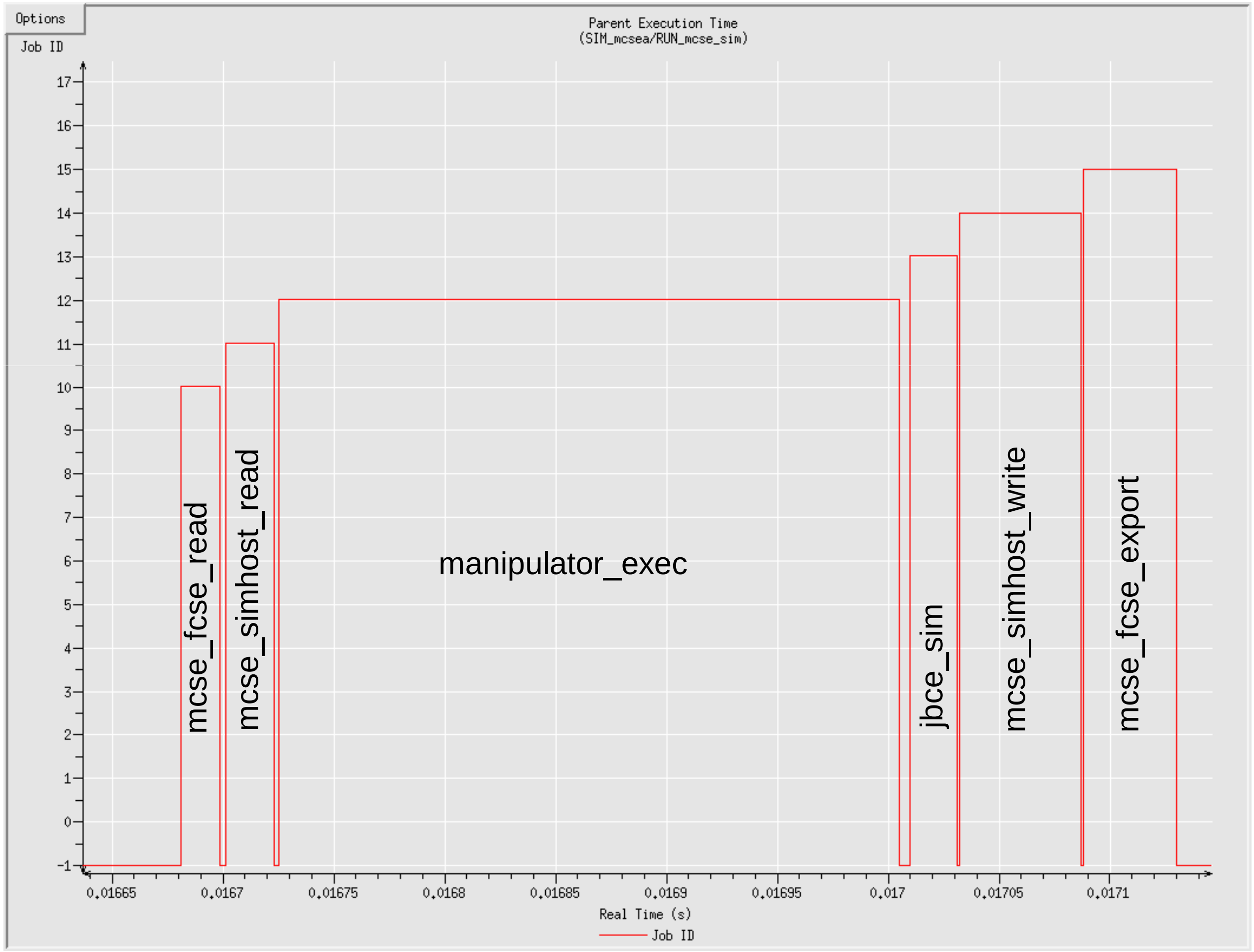


MRMDF Real-Time Performance

- **To translate Job IDs to function names see RUN_mcse_sim/S_job_execution**
  - **S_job execution lists out all jobs, their frequencies at init…**

```
Enabled | PID |  Start    |  Cycle    |    Stop    | ID  | Job Name
===============================================================================
automatic Jobs:
      1 |   1 | 0.000000 | 0.000000 |    1e+37 |   0 | sys.input_processor(&sys.exec)
scheduled Jobs:
      1 |   1 | 0.003333 | 0.033330 |    1e+37 |   1 | mcse.mcse_tdc_read(&mcse.mcse)
      1 |   1 | 0.006666 | 0.033330 |    1e+37 |   2 | mcse.mcse_rs422_read(&mcse.mcse)
      1 |   1 | 0.000000 | 0.033330 |    1e+37 |   3 | mcse.mcse_rs422_write(&mcse.mcse)
      1 |   1 | 0.006666 | 0.033330 |    1e+37 |   4 | mcse.mcse_device_read(&mcse.mcse)
      1 |   1 | 0.006666 | 0.033330 |    1e+37 |   5 | mcse.mcse_exec(&mcse.mcse)
      1 |   1 | 0.013332 | 0.033330 |    1e+37 |   6 | mcse.mee_exec(&mcse.mcse)
      1 |   1 | 0.013332 | 0.033330 |    1e+37 |   7 | mcse.meece_sim(&mcse.mcse)
      1 |   1 | 0.013332 | 0.033330 |    1e+37 |   8 | mcse.poa_exec(&mcse.mcse)
      1 |   1 | 0.013332 | 0.033330 |    1e+37 |   9 | mcse.poace_sim(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  10 | mcse.mcse_fcse_read(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  11 | mcse.mcse_simhost_read(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  12 | mcse.manipulator_exec(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  13 | mcse.jbce_sim(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  14 | mcse.mcse_simhost_write(&mcse.mcse)
      1 |   1 | 0.016665 | 0.033330 |    1e+37 |  15 | mcse.mcse_fcse_export(&mcse.mcse)
      1 |   1 | 0.023331 | 0.333300 |    1e+37 |  16 | mcse.mcse_tdc_export(&mcse.mcse)
logging Jobs:
      1 |   1 | 0.000000 | 0.033330 |    1e+37 |  17 | mcse.data_record_pack(&mcse.mcse)
```

- **Close-up of the timeline near 0.016665sec**



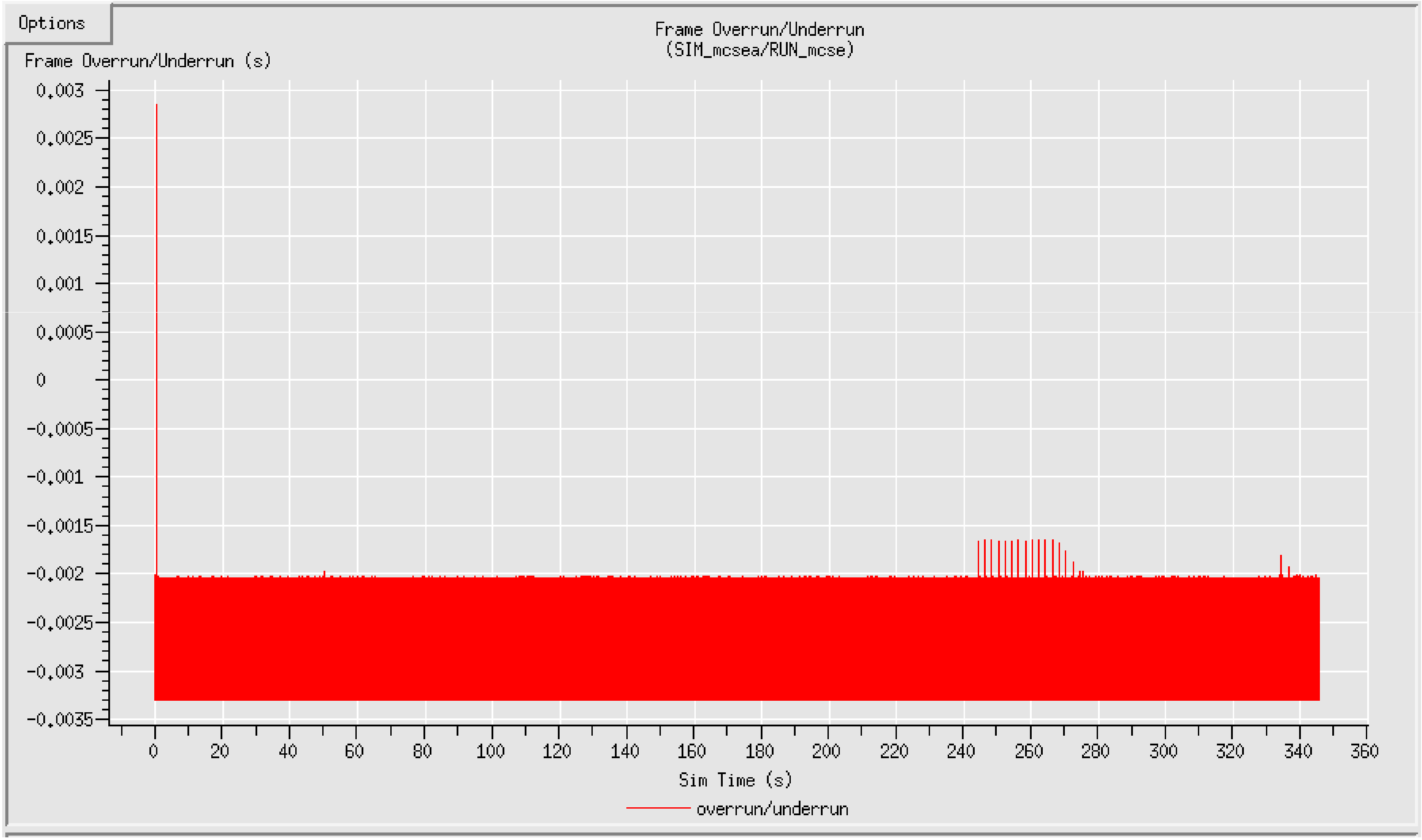
MRMDF Real-Time Performance

- **When there are overruns**
  - **The printout at sim termination shows the number of overruns:**

```
SIMULATION TERMINATED IN
  PROCESS: 1
  JOB/ROUTINE: 1/master.c
DIAGNOSTIC:
Sim Control Shutdown.

LAST JOB CALLED: mcse.mcse_tdc_read(&mcse.mcse)
              TOTAL OVERRUNS:             6
PERCENTAGE REALTIME OVERRUNS:        0.006%


       SIMULATION START TIME:         0.000
        SIMULATION STOP TIME:       345.672
     SIMULATION ELAPSED TIME:       345.672
         ACTUAL ELAPSED TIME:       345.672
        ACTUAL CPU TIME USED:        50.899
    SIMULATION / ACTUAL TIME:         1.000
       SIMULATION / CPU TIME:         6.791
  ACTUAL INITIALIZATION TIME:         0.000
     INITIALIZATION CPU TIME:         0.151
```

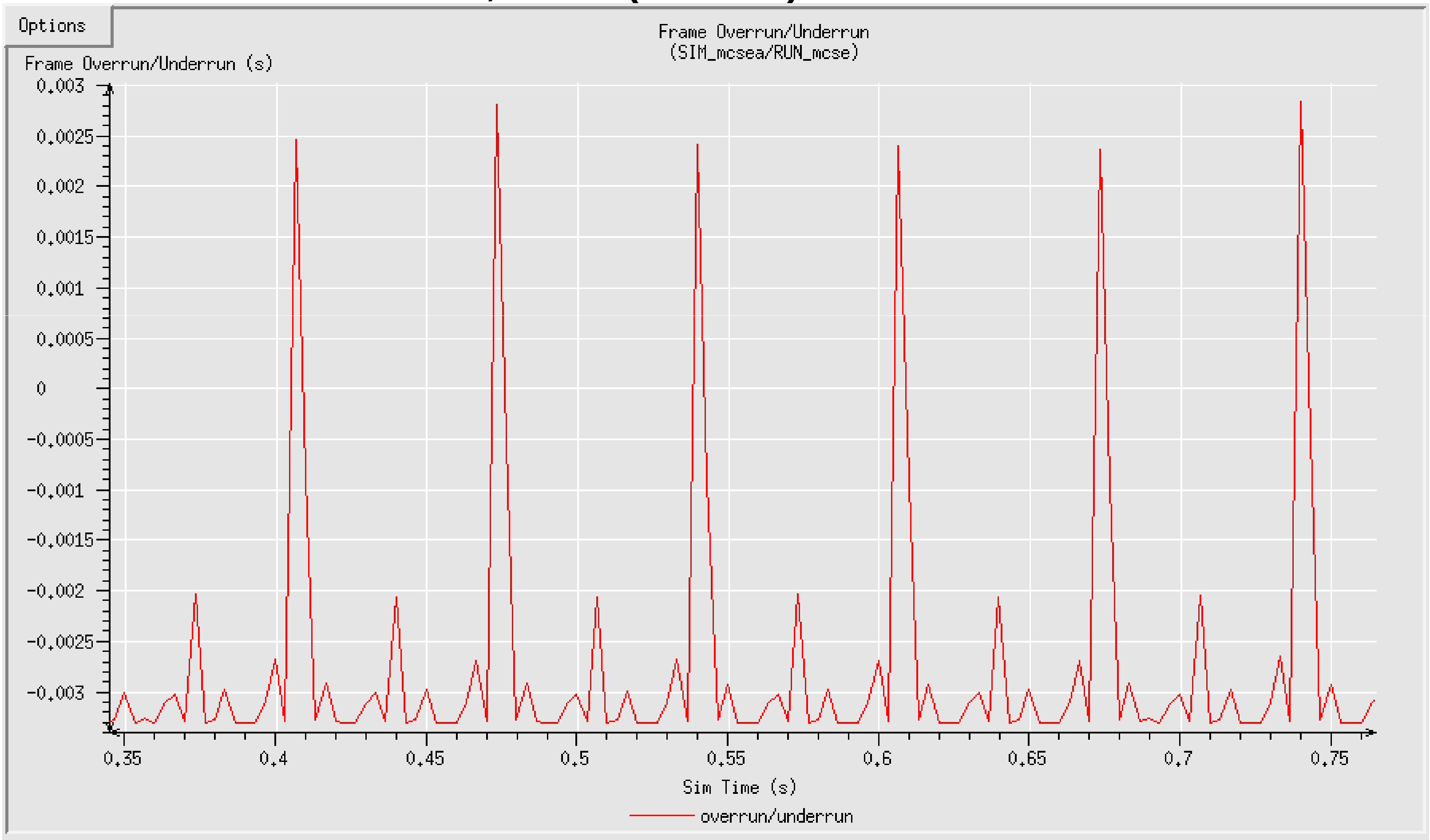
MRMDF Real-Time Performance

- **Selecting DP_rt_frame shows the overruns within the first second**



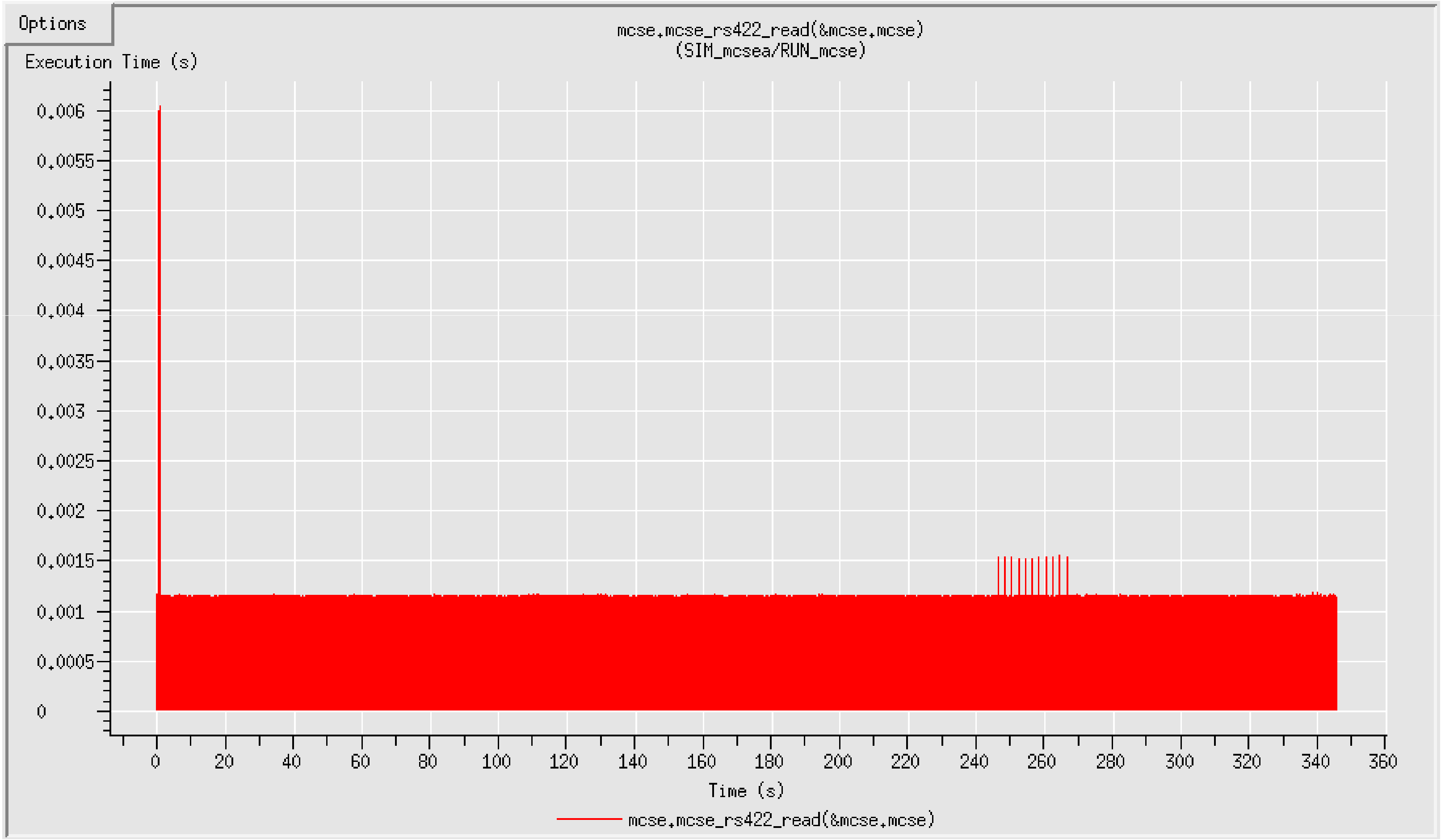
MRMDF Real-Time Performance

- **Close up of Frame Overrun/Underrun**
  - **About 2.5ms overruns, 5.8ms (2.5+3.33) total run time for these frames**



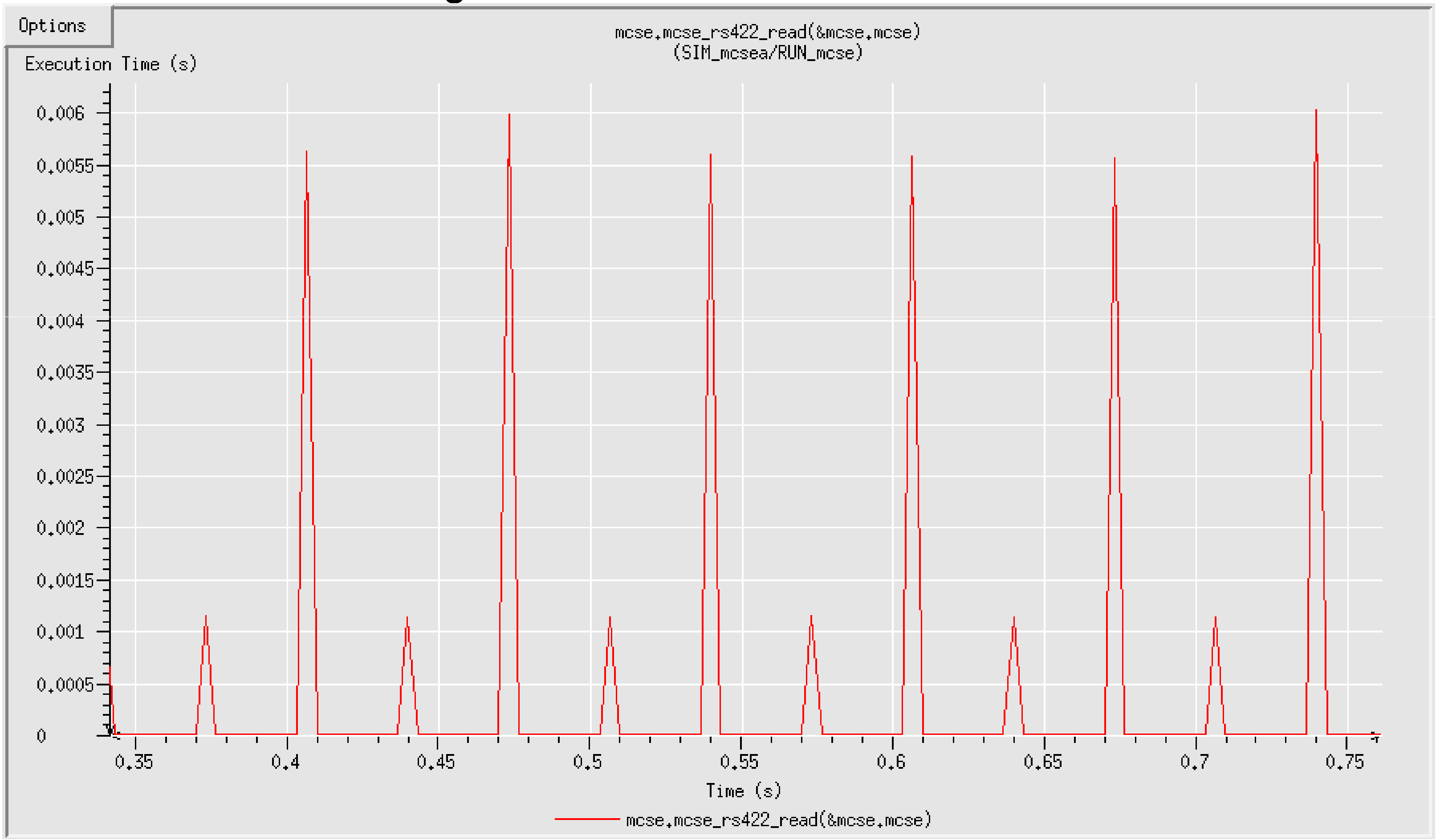
MRMDF Real-Time Performance

- **Looking at DP_rt_jobs at mcse_rs422_read we see the same spike**



MRMDF Real-Time Performance

- **Close up of mcse_rs422_read**
  - **About 5.7ms running time for the overrun frames**



MRMDF Real-Time Performance

- **In this case, it is known that in the first second, there are large initialization packets passed which can cause the job to take a couple of extra milliseconds to complete.**

- **All inter-simulation communications should be non-blocking or have a time out limit.**
  - **Trick provides both non blocking sockets or ones with timed blocking**
- **All simulations that include X-Windows GUIs should run on multi-CPU machines.**
  - **Simulation and X-event loop should be separated into 2 threads**
    - Simulation should run on processor by itself
    - X-Windows event loop should be assigned to different processor
- **Good to run non-specialized hardware/software configurations**

Monte Carlo

Overview

- **This tutorial briefly focuses on input file requirements to allow Trick to perform “Monte Carlo” simulation**

- **In Chapter 12 of the Trick User Training Guide, it was shown how to use Trick to vary jet firing sequences for the cannon jet control problem, both using ‘hard-coded’ inline data and Gaussian randomly generated data**

- **Here we look at our spring mass damper system simulation (`SIM_spring`, which has now been copied as `SIM_spring_mc`) and allow Trick to perform Monte Carlo for two specific examples**

Monte Carlo Input Review

- **Monte Carlo input files begin with `M_*` at the sim level**

- **Input file syntax:**
  - **NUM_RUNS: maximum number of runs**
  - **VARS: list of parameters (Trick variables) to vary**
  - **FILE_DATA: tells Trick to run with data from the DATA: list**
  - **DATA: values for the above Trick variables**

- **Once this file is created, Trick basically parses this input file and generates multiple runs under a new subdirectory called `MONTE_RUN_*` (where * corresponds to the name in the input file `M_<file_name>`)**

Example 1 – Varying Damping (Inline)

```
% vi M_smd_inline
```

NUM_RUNS: 100 **<= maximum number of runs**

```
VARS:
smd.spring.input.damping {N*s/m} FILE_DATA ;
```
<= damping coeff

DATA: **<= inline variation from undamped to overdamped system**

```
  0.0000
  2.0000
  4.0000
  8.0000
 16.0000
 32.0000
 64.0000
128.0000
256.0000
512.0000
```

Example 2 – Varying M, K, C (Gaussian)

```
% vi M_smd_gaussian
```

```
NUM_RUNS: 50

VARS:
smd.spring.input.mass {kg}
gaussian( seed = 1, sigma = 0.6862, mu = 8.0, rel_min = -2.0, rel_max = 2.0 ) ;

smd.spring.input.stiffness {N/m}
gaussian( seed = 1, sigma = 0.6862, mu = 128.0, rel_min = -64.0, rel_max = 64.0 );

smd.spring.input.damping {N*s/m}
gaussian( seed = 1, sigma = 0.6862, mu = 8.0, rel_min = -4.0, rel_max = 48.0 ) ;
```

`DATA:` **<= notice that data fields are empty; being randomly generated above**

- **Here we use syntax to set up a Gaussian distribution of mass, stiffness, and damping (notice seed (initializes random number generator), sigma (std dev), mu (mean), rel_min and rel_max)**

- **For this example, Trick randomly generates the run data through an interface to the GNU Scientific Library (`trick_gsl_rand.c`)**

Monte Carlo Execution

- **To execute either of these examples, two flags must be set in the input file (e.g., `RUN_monte.inline/input`):**
  - `sys.exec.monte.in.active = Yes ;`
  - `sys.exec.monte.input_files[0] = “M_smd_inline” ; (or “M_smd_gaussian”)`

- **Run the sim for the first example:**
  - `S_main_* RUN_monte.inline/input`

- **This capability was originally designed to be distributed across multiple machines on a network, ssh is used to run slave sims across the network. Ssh may ask for a password.**

- **Notice the new `RUN_MONTE_monte.inline` which contains the output data (can visualize multiple curves through trick_dp)**

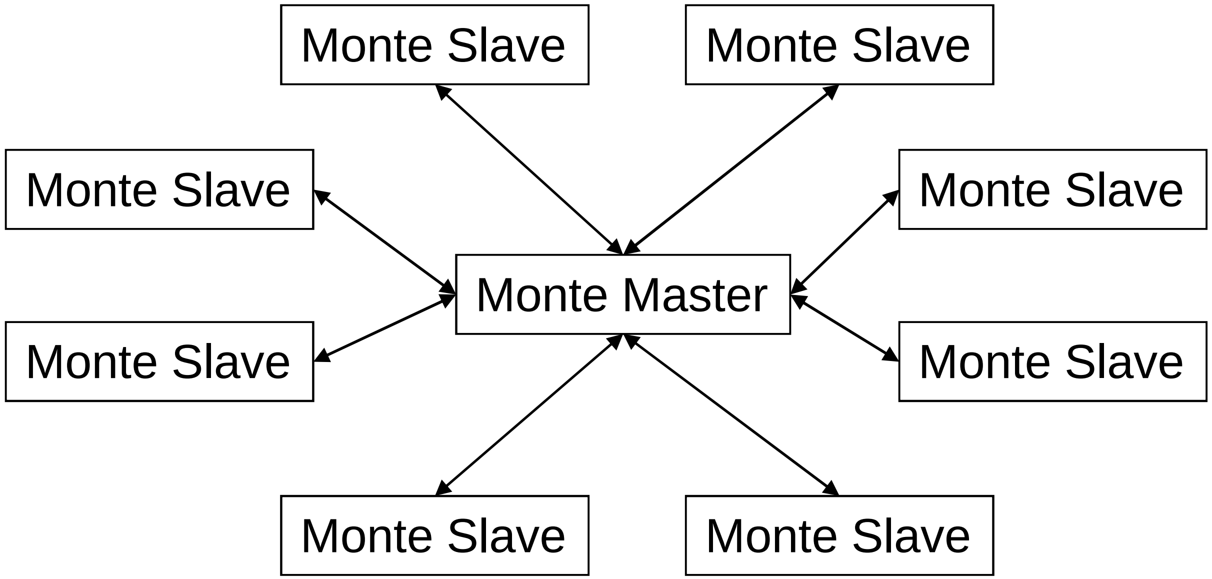
Monte Carlo Execution (cont.)

- **Now run the sim for the second example:**
  - **`S_main_* RUN_monte.gauss/input`**

- **Notice the new `RUN_MONTE_monte.gauss` which contains the output data (again, multiple curves can then be visualized through trick_dp)**

Monte Carlo Slaves

- **Previous examples used only a single worker**
- **Trick's Monte Carlo capability optimized for multiple workers**



Monte Carlo Slaves

- **To add slaves, allocate space for sys.exec.monte.in.slaves**
  - **Unlimited number of slaves can be specified**

```
sys.exec.monte.in.slaves = alloc(4) ;
sys.exec.monte.in.slaves[0].machine_name = “slave_1" ;
sys.exec.monte.in.slaves[1].machine_name = “slave_2" ;
sys.exec.monte.in.slaves[2].machine_name = “slave_3" ;
sys.exec.monte.in.slaves[3].machine_name = “slave_3" ;
```

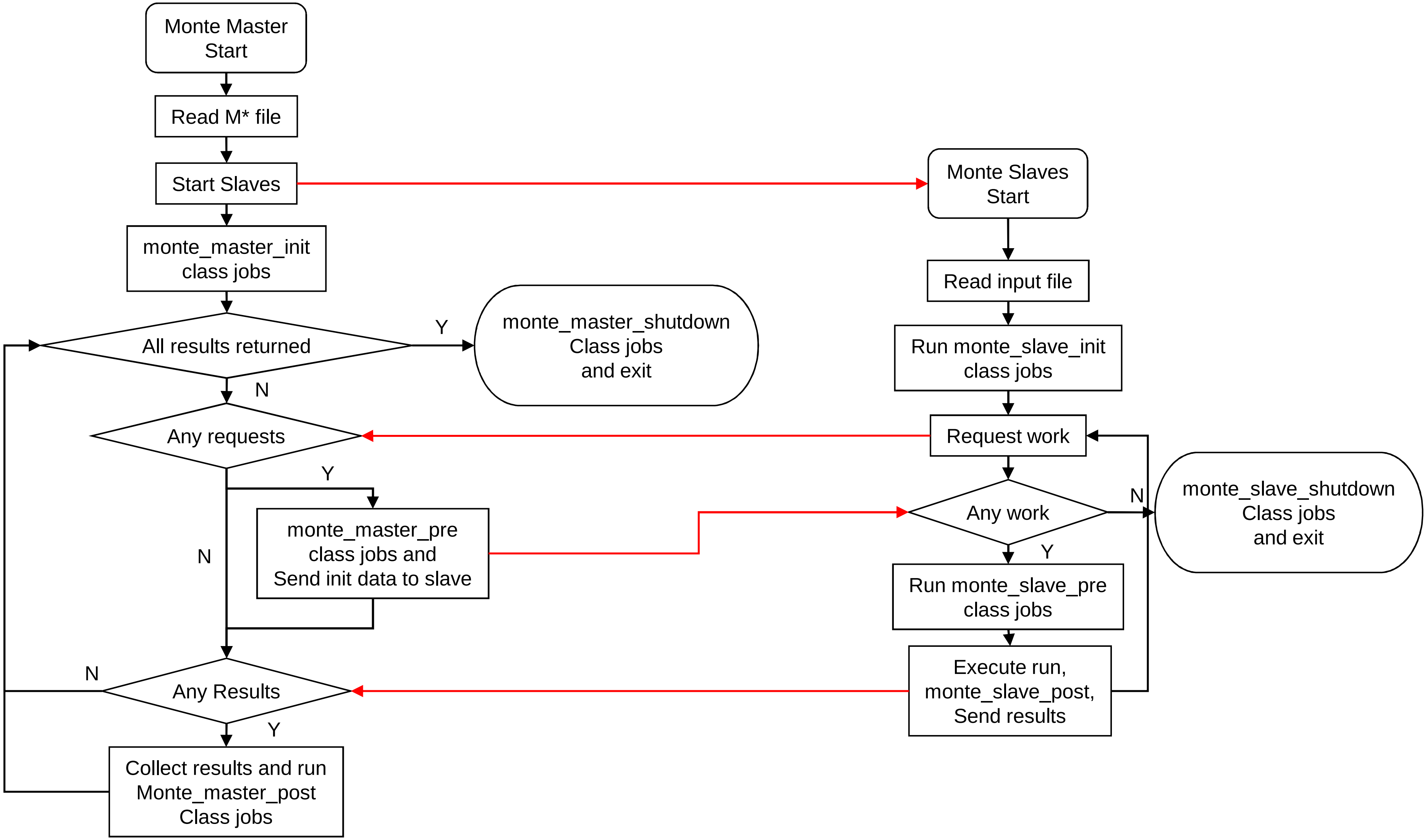
- **Trick will automatically start each slave simulation with ssh**
- **Slaves ask the master for work when they are ready for work**
  - **Faster slave machines will do more work**
- **You can start multiple slaves on the same machine**
  - **Useful for machines with multiple processors**
- **You can specify which remote shell to use and add additional arguments to the remote shell call for each slave**

```
sys.exec.monte.in.slaves[1].remote_shell = TRICK_RSH ;
sys.exec.monte.in.slaves[1].remote_shell_args = “-l user”;
```

Job Classes

- **Monte Carlo specific job classes to handle master/slave interations**
  - **Monte_Master_Init**
    - Runs when master sim is initialized
  - **Monte_Master_Pre**
    - Runs before new data is dispatched to slave sim
    - Useful for calculating/optimizing next run values if desired
  - **Monte_Master_Post**
    - Runs after result is returned from slave
    - Useful for calculating statistics for returning results
  - **Monte_Master_Shutdown**
    - Runs when master shuts down
  - **Monte_Slave_Init**
    - Runs when slave sim is initialized
  - **Monte_Slave_Pre**
    - Runs after new data is received from master
  - **Monte_Slave_Post**
    - Runs after slave sim is completed (sends result to master)
  - **Monte_Slave_Shutdown**
    - Runs when monte carlo master comm is lost and slave shuts down

Monte Carlo Master/Slave Interaction



Monte Slaves

- **The master sets a timeout value sys.exec.monte.in.timeout**
  - **Default timeout is 120 seconds**
- **Each slave must return a result within its individually timed timeout period**
  - **If no result is returned, the slave is assumed dead and the run's initial data is redispatched to the next available slave**
  - **Slaves can be "killed" and no results will be lost**

Monte Carlo Notes

- **A dry run flag was recently added: sys.exec.monte.in.dryrun**
  - **Useful for generating random distributions without actually doing the runs**

- **All data recording for all runs is saved.**
  - **Large data sets can generate enormous amounts of data.**
  - **Take care on what to data record**

- **Almost too easy to add slaves**
  - **Tendency to add machines which seem unused**
  - **Monte Carlo slaves tend to use 99.9% of CPU**
  - **Don’t use too many machines in your lab!**

Generic Malfunction Insertion

Generic Malfunction Insertion

- **The Generic Malfunction Insertion capability allows users to override the value of any simulation variable or call a malfunction job at any time during the simulation**

- **Two new job classes**
  - **malfunction**
  - **malfunction_trigger**

  - **Neither new class has a calling frequency**
  - **Associated with jobs within the S_define file within the malfunction definition in the input file**
  - **Called whenever associated jobs are called**

Malfunction Syntax

- **Input file malfunction syntax**

```
begin malfunction <malf_name> {

  trigger {
    condition: <condition> ;
      or
    job: “<malf_trigger name from S_default.dat>” ;
    insert_(before|after): “<job from S_default.dat>” ;
    hold: (Yes|No) ;
  }

  <param> {
    insert_(before|after): “<job from S_default.dat>” ;
    units: <units>
    scale_factor: <value> ;
    bias: <value> ;
  }

  call “<malf job from S_default>” (before|after) “<job from S_default>” ;
  job “<job from S_default>” = (On|Off) ;

}
```

Triggers

- **Triggers can be either a condition statement or a call to a new class of "malfunction_trigger" class job.**
  - **malfunction_trigger jobs are faster to execute but must be compiled and declared in the S_define file**
  - **Condition statements are easily modified but suffer a performance penalty when they are parsed each time the condition is evaluated**
- **Triggers can be associated with any job in the simulation**
- **Triggers are evaluated each time the associated job is run**
- **Triggers may be evaluated before or after its associated job**
- **Triggers can be held, meaning once triggered the malfunction is always on from that point on**

- **Trigger Examples**

```
trigger {
  condition: <condition> ;
    or
  job: “<malf_trigger name from S_default.dat>” ;
  insert_(before|after): “<job from S_default.dat>” ;
  hold: (Yes|No) ;
}
```

```
trigger {
  condition: sys.exec.out.time >= 30.0 ;
  insert_before: “dyn.cannon_integ(&dyn.cannon)” ;
  hold: Yes ;
}
```

```
trigger {
  job: “dyn.cannon_malfunction_trigger(&dyn.cannon)” ;
  insert_after: “dyn.cannon_integ(&dyn.cannon)” ;
  hold: No ;
}
```

- **Parameter example**

```
<param> {
  insert_(before|after): "<job from S_default.dat>" ;
  units: <units>
  scale_factor: <value> ;
  bias: <value> ;
}
```

```
dyn.cannon.vel[0] {
  insert_after: "dyn.cannon_integ(&dyn.cannon)" ;
  units: "m/s"
  scale_factor: 0.0 ;
  bias: 1.0 ;
}
```

Malfunction Syntax

- **Calling a malfunction job**

```
call “<malf job from S_default>” (before|after) “<job from S_default>” ;
```

```
call “dyn.cannon_malf(&dyn.cannon)” after “dyn.cannon_integ(&dyn.cannon)”;
```

- **Turning jobs on/off**

```
job “<malf job from S_default>” = (On|Off) ;
```

```
job “dyn.cannon_abort(&dyn.cannon) = On ;
```

- **Manually turn malfunctions on/off within the input processor or commanded from a variable server client**

```
malfunction_cmd my_malf manual_on ;

malfunction_cmd my_malf manual_off ;
```

- **Remove manual override and return to evaluating triggers**

```
malfunction_cmd my_malf remove_manual ;
```

Units Upgrade

Units upgrades

- **Larger set units accepted in 07**
  - **Many SI prefixes accepted for metric units**
    - Exception is kft
  - **Meters is now “m”**
  - **“M” still accepted for meters for backwards compatibility**
  - **Multiplier operators are strongly encouraged to avoid ambiguities**
    - e.g. Is “mm” millimeters or meters*meters?

ICG –u output

```
Trick Measurement Units Summary
-------------------------------
                Time:   s min  hr day
Angular Displacement:   r   d  as  am rev
             Voltage:   v
            Amperage: amp
          Resistance: ohm
               Sound:  dB
            Unitless:  -- cnt one

English System Units
--------------------
 Linear Displacement:  ft  in  yd  mi  n.m.
                Mass:  sl lbm
               Force:  oz lbf
         Temperature:   R   F

Metric System Units
-------------------
 Linear Displacement:   m
                Mass:   g  mt
               Force:   N
         Temperature:   C   K

Prefixes for Multiples and Submultiples
(Not valid for English system units)
---------------------------------------
10**-1    d          10      da
10**-2    c          10**2    h
10**-3    m          10**3    k
10**-6    u          10**6    M
10**-9    n          10**9    G
10**-12   p          10**12   T
```

Wide Character (Unicode) Support

Wide Character Support

- **For Internationalization (and/or many other Unicode characters) :**
  - **Trick 07 supports type wchar_t .**
    - Fully supported by Checkpoint / Reload.

Wide Character Support

Pre-requisite: Setup your Locale

A locale specifies the national / cultural / language conventions that you would like your (locale aware apps) to follow.

```
setenv LC_CTYPE en_US.utf8 ← Specifies character encoding.
setenv LC_COLLATE POSIX  ← Specifies string sorting order.
```

LC_CTYPE is required by Trick and must specify a UTF-8 locale (at least while the input processor is running).

There are additional locale environment variables. It's not required that you set them all. For more information: % man locale

Wide Character Support

- **Wide-character strings in a structure definition.**

```
File  Edit  View  Terminal  Tabs  Help
   double          JD_start;         /* (day)  Julian date at the begin
ning of the sim run.*/
   double          JD;               /* (day)  Current Julian date */
   double          right_ascension;  /* (d)   */
   double          declination;      /* (d)   */
   double          hour_angle;       /* (d)   */
   double          solar_azimuth;    /* (d)   */
   double          solar_elevation;  /* (d)   */
   wchar_t*        label_UTC;        /* (--) wide character pointer */
   wchar_t*        label_JD;         /* (--) wide character pointer */
   wchar_t*        label_Azimuth;    /* (--) wide character pointer */
   wchar_t*        label_Elevation;  /* (--) wide character pointer */
} SUN_PRED;
sun_pred.h [+]                                    25,36          92%
```

Wide Character Support

- **A .d file to initialize the wide-strings.**

```
File  Edit  View  Terminal  Tabs  Help
sun_predictor.sun.label_UTC        = "UTC";
sun_predictor.sun.label_JD         = "JD";
sun_predictor.sun.label_Azimuth    = "方位角";
sun_predictor.sun.label_Elevation  = "高度";
~
~
Japanese_labels.d                                    1,1            All
"Japanese_labels.d" 4L, 184C
```

- **An alternate .d file which does exactly the same thing.**

```
File  Edit  View  Terminal  Tabs  Help
sun_predictor.sun.label_UTC        = "UTC";
sun_predictor.sun.label_JD         = "JD";
sun_predictor.sun.label_Azimuth    = "\u65b9\u4f4d\u89d2";
sun_predictor.sun.label_Elevation  = "\u9ad8\u5ea6";
~
~
Japanese_labels_alt.d [+]                            1,1            All
:1
```

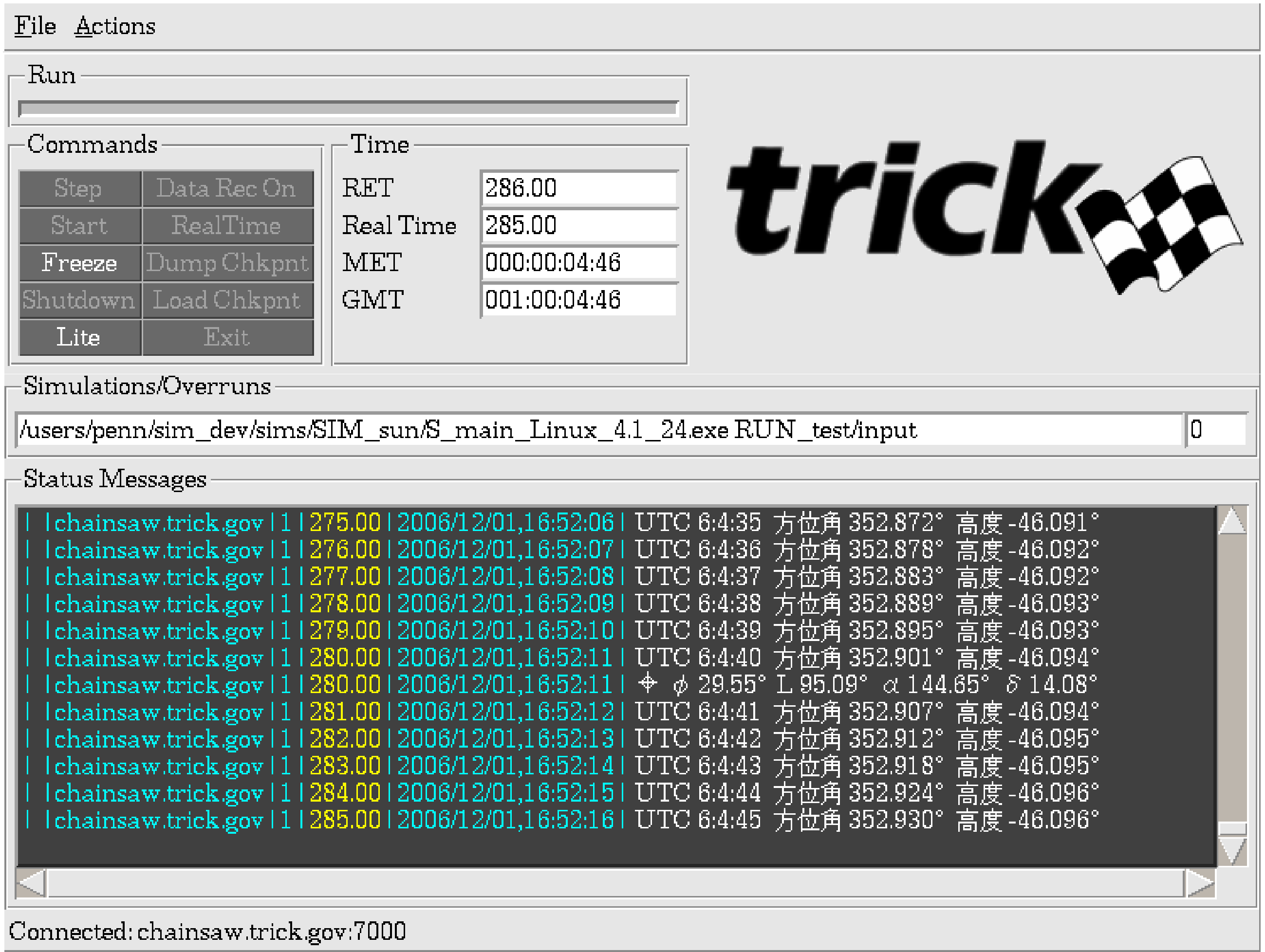
- **If you can’t remember how to enter the Unicode character in your editor.**

Wide Character Support

- **Terminal Output**

```
File  Edit  View  Terminal  Tabs  Help
|  |chainsaw.trick.gov|1|57.00|2006/12/01,16:54:49|  UTC 6:0:57   方位角 351.612°  高度 -45.984°
|  |chainsaw.trick.gov|1|58.00|2006/12/01,16:54:50|  UTC 6:0:58   方位角 351.618°  高度 -45.985°
|  |chainsaw.trick.gov|1|59.00|2006/12/01,16:54:51|  UTC 6:0:59   方位角 351.623°  高度 -45.985°
|  |chainsaw.trick.gov|1|60.00|2006/12/01,16:54:52|  UTC 6:1:00   方位角 351.629°  高度 -45.986°
|  |chainsaw.trick.gov|1|60.00|2006/12/01,16:54:52|  ♁  φ 29.55°  L 95.09°  α 144.65°  δ 14.08°
|  |chainsaw.trick.gov|1|61.00|2006/12/01,16:54:53|  UTC 6:1:01   方位角 351.635°  高度 -45.986°
|  |chainsaw.trick.gov|1|62.00|2006/12/01,16:54:54|  UTC 6:1:02   方位角 351.641°  高度 -45.987°
|  |chainsaw.trick.gov|1|63.00|2006/12/01,16:54:55|  UTC 6:1:03   方位角 351.647°  高度 -45.987°
|  |chainsaw.trick.gov|1|64.00|2006/12/01,16:54:56|  UTC 6:1:04   方位角 351.652°  高度 -45.988°
|  |chainsaw.trick.gov|1|65.00|2006/12/01,16:54:57|  UTC 6:1:05   方位角 351.658°  高度 -45.988°
|  |chainsaw.trick.gov|1|66.00|2006/12/01,16:54:58|  UTC 6:1:06   方位角 351.664°  高度 -45.989°
|  |chainsaw.trick.gov|1|67.00|2006/12/01,16:54:59|  UTC 6:1:07   方位角 351.670°  高度 -45.989°
|  |chainsaw.trick.gov|1|68.00|2006/12/01,16:55:00|  UTC 6:1:08   方位角 351.675°  高度 -45.990°
|  |chainsaw.trick.gov|1|69.00|2006/12/01,16:55:01|  UTC 6:1:09   方位角 351.681°  高度 -45.990°
|  |chainsaw.trick.gov|1|70.00|2006/12/01,16:55:02|  UTC 6:1:10   方位角 351.687°  高度 -45.991°
|  |chainsaw.trick.gov|1|70.00|2006/12/01,16:55:02|  ♁  φ 29.55°  L 95.09°  α 144.65°  δ 14.08°
|  |chainsaw.trick.gov|1|71.00|2006/12/01,16:55:03|  UTC 6:1:11   方位角 351.693°  高度 -45.991°
|  |chainsaw.trick.gov|1|72.00|2006/12/01,16:55:04|  UTC 6:1:12   方位角 351.698°  高度 -45.992°
|  |chainsaw.trick.gov|1|73.00|2006/12/01,16:55:05|  UTC 6:1:13   方位角 351.704°  高度 -45.992°
|  |chainsaw.trick.gov|1|74.00|2006/12/01,16:55:06|  UTC 6:1:14   方位角 351.710°  高度 -45.993°
|  |chainsaw.trick.gov|1|75.00|2006/12/01,16:55:07|  UTC 6:1:15   方位角 351.716°  高度 -45.994°
|  |chainsaw.trick.gov|1|76.00|2006/12/01,16:55:08|  UTC 6:1:16   方位角 351.722°  高度 -45.994°
```

- **Sim Control Panel Output**



Wide Character Support

- **Wide character strings in a checkpoint file**

File Edit View Terminal Tabs Help

```
sun_predictor.sun.JD_start = 2453962.75 ;
sun_predictor.sun.JD = 2453962.750671296 ;
sun_predictor.sun.right_ascension = 144.6524902594217 ;
sun_predictor.sun.declination = 14.08073747818402 ;
sun_predictor.sun.hour_angle = -185.9944119461217 ;
sun_predictor.sun.solar_azimuth = 351.6177096562982 ;
sun_predictor.sun.solar_elevation = -45.98453626450691 ;
sun_predictor.sun.label_UTC = "UTC" ;
sun_predictor.sun.label_JD = "JD" ;
sun_predictor.sun.label_Azimuth = "方位角" ;
sun_predictor.sun.label_Elevation = "高度" ;
sun_predictor.job[0].in.phase = 60000 ;
sun_predictor.job[0].in.job_depend = alloc(1) ;
chk_59.00                                                  254,1          83%
```

Wide Character Support

- **Convenience routines to convert to and from Wide-character to Narrow-character strings.**

File Edit View Terminal Tabs Help

```
#ifdef __cplusplus
#extern "C" {
#endif

#include <stddef.h>
#include <wchar.h>
size_t wcs_to_ncs_len (const wchar_t *wcs);
size_t ncs_to_wcs_len (const char *ncs);
size_t wcs_to_ncs (const wchar_t *wcs, char *ncs, size_t ncs_max_len );
size_t ncs_to_wcs(const char *ncs, wchar_t *wcs, size_t w_max_size );
#ifdef __cplusplus
}
#endif
wcs_ext.h [RO]                                    49,1          92%
```

Additional Material

External Clocks and Timers

Clocks & Itimers

- **A Trick simulation can be configured with interval timers (or itimers) that use setitimer(), pause() and a signal handler to manage a “go to sleep” and “wake up” at the end of each configured itimer frame**
  - **Itimers are suited to facilitate processor sharing**
  - **This itimer frame can be larger than the RT frame, but it must be a multiple of it**
  - **Since UNIX signals (SIGALRM) are used, this feature increases executive overhead, and itimer frames smaller than 10 milliseconds are not recommended**

- **By default, during an overrun the Trick executive logs the over run and keeps going to try to catch up to real-time.**
  - **Sim can be configured to terminate or freeze when a maximum number of over runs in a row occur, or when a single over run surpasses a specified time limit**

Executive Clock Calls

- **executive clock calls**
  - **double clock_time (GMT_STRUCT*)**
    - ***This function returns the simulation's current real-time clock value in total seconds from the simulations reference time mark***
  - **void clock_reset (double ref)**
    - ***This function resets the simulation clock reference mark by subtracting the passed reference time (total seconds) from the current time returned by a clock_time() call.***
  - **These function also works with external clocks.**

- **Trick supports the use of external clocks**
  - **To configure an external clock:**
    - ***Input file: sys.exec.in.rt_clock = EXTERNAL ;***
    - ***In an initialization job, set function pointers to user defined functions for external clock initialization and time acquisition:***
      - sys.exec.in.trick_external_clock_init
        - » Make necessary systems calls to initialize external clock device and load the passed argument with total seconds (gmt->y_secs), which will be used by the executive for a clock reference point. This function is of type void and the single passed argument is a pointer of type GMT_STRUCT.
        - » void trick_external_clock_init (GMT_STRUCT * gmt)
      - sys.exec.in.trick_external_clock_time
        - » Make necessary external clock calls to load the passed argument with the current time in total seconds (gmt->y_secs), which will be used by the executive for the elapsed clock time calculation (current time minus the reference point established in trick_external_clock_init()). This function is of type void and the single passed argument is a pointer of type GMT_STRUCT.
        - » void trick_external_clock_time (GMT_STRUCT * gmt)

External Clocks/Timers

- **Trick also supports the use of external timers**
  - **To configure an external timer:**
    - ***Input file: sys.exec.in.rt_exttimer = On ;***
    - ***In an initialization job, set function pointers to user defined functions for external timer start, reset, stop & pause:***
      - sys.exec.in.trick_external_timer_start
        - » This function should make necessary systems calls to start the user defined external interval timer. The Trick executive will call this user provided function one time in initialization at the beginning of the real-time frame. The interval frame time must be defined in the Trick sys.exec.in.rt_itimer_frame variable. Trick itimers can not be used in conjunction with external timers. The trick_external_timer can be set up as a one-time timer with the next interval being reset at the end of each frame with the external function trick_external_timer_reset(), or it could be set up with reoccurring intervals without using the reset function. The function trick_external_timer_pause() will be defined by the user to wait for the timer. This function is of type void and the single passed argument is a pointer of type void. This same voided pointer argument is passed to all of the external timer routines.
        - » void trick_external_timer_start (void *)

– sys.exec.in.trick_external_timer_reset
  » This function should make necessary systems calls to reset the user defined external interval timer (started by trick_external_timer_start()). The Trick executive will call this user provided function to set the next timing interval after real-time has caught up in the under run condition. If the sim is in an overrun condition, this function will not be called called, but the external_timer_stop function will be called. The interval frame time must be defined in the Trick sys.exec.in.rt_itimer_frame variable. Trick itimers can not be used in conjunction with external timers. The function trick_external_timer_pause() will be defined by the user to wait for the timer. This function is of type void and the single passed argument is a pointer of type void. This same voided pointer argument is passed to all of the external timer routines.
  » void trick_external_timer_reset (void *)
– sys.exec.in.trick_external_timer_stop
  » This function should make necessary systems calls to allow the external timers to recover from an overrun condition. The Trick executive will call this user provided function at the end of the simulation real-time frame when an overrun occurs. This function is of type void and the single passed argument is a pointer of type void. This same voided pointer argument is passed to all of the external timer routines.
  » void trick_external_timer_stop (void *)

External Clocks/Timers

– sys.exec.in.trick_external_timer_pause

» This function should make the necessary systems calls to wait for the user defined external interval timer (defined by trick_external_timer_start() and/or trick_external_timer_reset()). The Trick executive will call trick_external_timer_pause() at the end of each simulation itimer frame (defined by sys.exec.in.rt_itimer_frame) to wait for the real-time clock to catch up. When Trick is set up to use itimers, it simply uses the UNIX pause() to wait for the SIGALRM. If external timers are used, trick_external_timer_pause() is called in place of pause, which is used for Trick itimers. This wait can be implemented by what ever means are available to detect the timer from the timer device drivers. Again, external timers can not be used in conjunction with Trick itimers. This function is of type void and the single passed argument is a pointer of type void. This same voided pointer argument is passed to all of the external timer routines.

» void trick_external_timer_pause (void *)

External Clocks/Timers

- **Example of initialization job setting up external clock/timer functions in SIM_master_timer/RUN_master**
  - **Let's look at the src file init_ext_clock_timers.c**
    - ***View file with editor in trick_ui or terminal***
  - **Also, view S_define with trick_ui or terminal**
    - ***See addition of init_ext_clock_timers() initialization job call***

```
(initialization) extclocktimer: init_ext_clock_timers();
```

  - **CP SIM_master_timer**
    - ***Correct any errors***

External Clocks/Timers

- **Example of initialization job setting up external clock/timer functions in SIM_master_timer/RUN_master**
  - **Add to input file**

sys.exec.in.rt_clock = EXTERNAL ;

sys.exec.in.rt_exttimer = On ;

- **note that itimer frame that is required is already set to 0.05 in Modified_data/realtime.d**

sys.exec.in.rt_itimer = Off ;

- **external timers and itimers conflict with each other**

  - **No changes necessary in slaved simulation**
    - ***It will just keep syncing to master through sockets***
  - **Run SIM_master_timer /RUN_master to test external clock and external timer capability**

External Libraries and the Trick Math Library

Set up the Environment

- **Objective**
  - **Setup Trick Environment**

- **Prerequisites**
  - **Login credentials**

Adding External Libraries

- **There may be situations where we want Trick just to include a pre-built library or include an entire directory with one library dependency. To accomplish that, Trick has multiple ways of including external libraries**

- **Libraries that do not need to be compiled**
  - **Preferred: Add the library to the environment variable TRICK_USER_LINK_LIBS**

```
setenv TRICK_USER_LINK_LIBS = -L/path/to/my/lib –lmy_lib
```

  - **Non-preferred: Add the library to the LIBRARY_DEPENDENCY list of a module included in the simulation**

```
/* TRICK_HEADER
PURPOSE: (no purpose)
LIBRARY_DEPENDENCIES: ((lib_to_include.a))
*/
```

Adding External Libraries

- **Libraries that do need to be compiled**
  - **Preferred: Include a file called S_overrides.mk in your sim directory that adds a dependency to your library, compiles it and includes it for linking**
  - **Great flexibility using this method, allows custom makefiles in library directories**

```
# S_overrides.mk

TRICK_USER_LINK_LIBS += -L${HOME}/trick_models/ball/L1/object_${TRICK_HOST_CPU} -lball

${S_MAIN}:  ${HOME}/trick_models/ball/L1/object_${TRICK_HOST_CPU}/libball.a

clean: clean_build_ball

${HOME}/trick_models/ball/L1/Makefile:
       cd ${HOME}/trick_models/ball/L1 ; make_build lib libball.a

${HOME}/trick_models/ball/L1/object_${TRICK_HOST_CPU}/libball.a:
${HOME}/trick_models/ball/L1/Makefile
       cd ${HOME}/trick_models/ball/L1 ; ${MAKE}

clean_build_ball:
       cd ${HOME}/trick_models/ball/L1 ; ${MAKE} clean
```

Adding External Libraries

- **Libraries that do need to be compiled**
  - **Non-preferred: Add the library to the LIBRARY_DEPENDENCY list of a module included in the simulation. Include a "relative" path to the library so Trick can find it.**
  - **Trick will gather all source files in the library path and compile them to the lib name specified in the LIBRARY_DEPENDENCY**
  - **All source code in the directory must compile with standard TRICK_CFLAGS**

```
/* TRICK_HEADER
PURPOSE: (no purpose)
LIBRARY_DEPENDENCIES: ((rel/path/to/lib/lib_to_include.a))
*/
```

Trick Math Library

- **The Trick mathematical support library provides numerous built-in utility functions for a variety of modeling tasks**

- **This library can be found in the following directory:**

```
$TRICK_HOME/trick_source/trick_utils/math/src
```

- **List contents of the math library**

```
% ls $TRICK_HOME/trick_source/trick_utils/math/src

deuler_123.c       dmtxmt.c             dvxm.c               matxmat.c
deuler_132.c       dmtxv.c              dvxv_add.c           matxtrans.c
deuler_213.c       dmxm.c               dvxv_sub.c           matxvec.c
deuler_231.c       dmxmt.c              eigen_hh_red.c       quat_mult.c
deuler_312.c       dmxv.c               eigen_jacobi_4.c     quat_norm.c
deuler_321.c       drandom_gaussian.c   eigen_jacobi.c       quat_norm_integ.c
dLU_Choleski.c     dS_function.c        eigen_ql.c           quat_to_mat.c
dLU_solver.c       dsingle_axis_rot.c   euler_matrix.c       rand_num.c
dm_add.c           dv_add.c             gauss_rnd_bell.c     roundoff.c
dm_copy.c          dv_copy.c            gauss_rnd_pseudo.c   tm_print_error.c
dm_ident.c         dv_cross.c           LU_bksb.c            transxmat.c
dm_init.c          dv_dot.c             LU_dcmp.c            transxtrans.c
dm_invert.c        dv_init.c            LUD_inv.c            transxvec.c
dm_invert_symm.c   dv_mag.c             LUT_inv.c            trick_gsl_rand.c
dm_orthonormal.c   dv_norm.c            makefile             trns_fnct_1o.c
dm_print.c         dv_print.c           mat_copy.c           trns_fnct_2o.c
dm_scale.c         dv_scale.c           mat_permute.c        uniform_rnd_1.c
dm_sub.c           dv_skew.c            mat_print.c          uniform_rnd_triple.c
dm_trans.c         dv_store.c           mat_to_quat.c        vec_print.c
dmtxm.c            dv_sub.c             mat_trans.c          wave_form.c
```

Trick Math Library Summary

- **Vector/Matrix algebra (3x3) – dv_*.c, dvx*.c, dm*.c and dmt*.c**
- **Matrix algebra (nxn) – mat_*.c, matx*.c and transx*.c**
- **Linear equation solvers - dLU*.c, dm_invert*.c, and LU*.c**
- **Euler transformations – euler_matrix.c, deuler_*.c, and dsingle_axis_rot.c**
- **Quaternion transformations – quat_*.c**
- **Eigensolvers - eigen_*.c**
- **Random number generation – drandom_gaussian.c, gauss*.c, rand*.c, trick_gsl_rand.c, and uniform*.c**
- **Other odds and ends functions (e.g., wave form generator)**

Trick Math Library Summary (cont.)

- **Access to the Trick math library can be achieved by adding the following prototype definition to the your source code:**

```
#include “trick_utils/math/include/trick_math_proto.h”
```

- **Several other library type routines exist throughout the suite of Trick-based sim packages (e.g., robotics, GN&C, mechanisms) and are planned to be folded back into this library for upcoming Trick releases**

- **Additional capabilities/requests are always welcome!**

Trick Math Headers

- **Perhaps as important as the individual math subroutines is the functionality provided through the Trick math header files**

- **Macros were originally created to provide execution speed over their subroutine counterparts (using the C preprocessor for inline code expansion)**

```
% ls -1 $TRICK_HOME/trick_source/trick_utils/math/include

complex.h
matrix_macros.h
quat_macros.h
rand_generator.d
rand_generator.h
reference_frame.h
trick_math_error.h
trick_math.h
trick_math_proto.h
vector_macros.h
wave_form.h
```

Trick Math Headers (cont.)

- **In addition to early speed benefits, many Trick math models were implemented through macros to make code more readable as well as provide easy mapping back to formulation/equations (e.g., refer to the matrix_macros and vector_macros headers)**

- **Euler and Quaternion transformations rely on both the reference_frame.h and quat_macros.h header files, respectively**

- **Access to the Trick math headers can be achieved by adding the following to your source code:**

```
#include "trick_utils/math/include/trick_math.h”
```