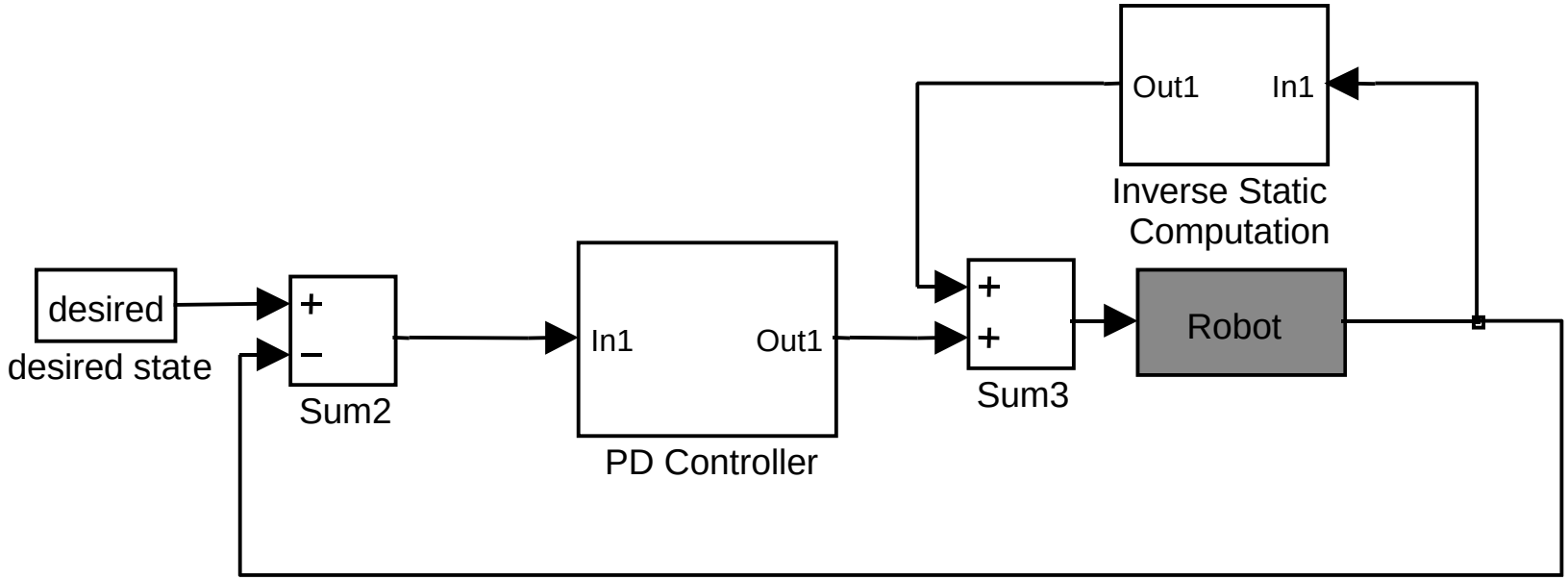
CS545—Contents XXI

- Case Study: Gravity Compensation with the Sarcos Dexterous Master Arm
  - + A Gravity Compensation Control Circuit
    - Primary goals and subgoals
    - Math and Algorithms
    - Automatic C-code generation with mathematica
  - + How to embed the controller in the VxWorks environment
    - Spinal-Cord: the low level I/O and negative feedback processor
    - Interprocessor communication (semaphore, shared semaphores, shared objects)
    - Motor-Cortex: the task level control processor
    - Creating a task program
- Reading Assignment for Next Class
    - See http://www-slab.usc.edu/courses/CS545

Theory: Gravity Compensation

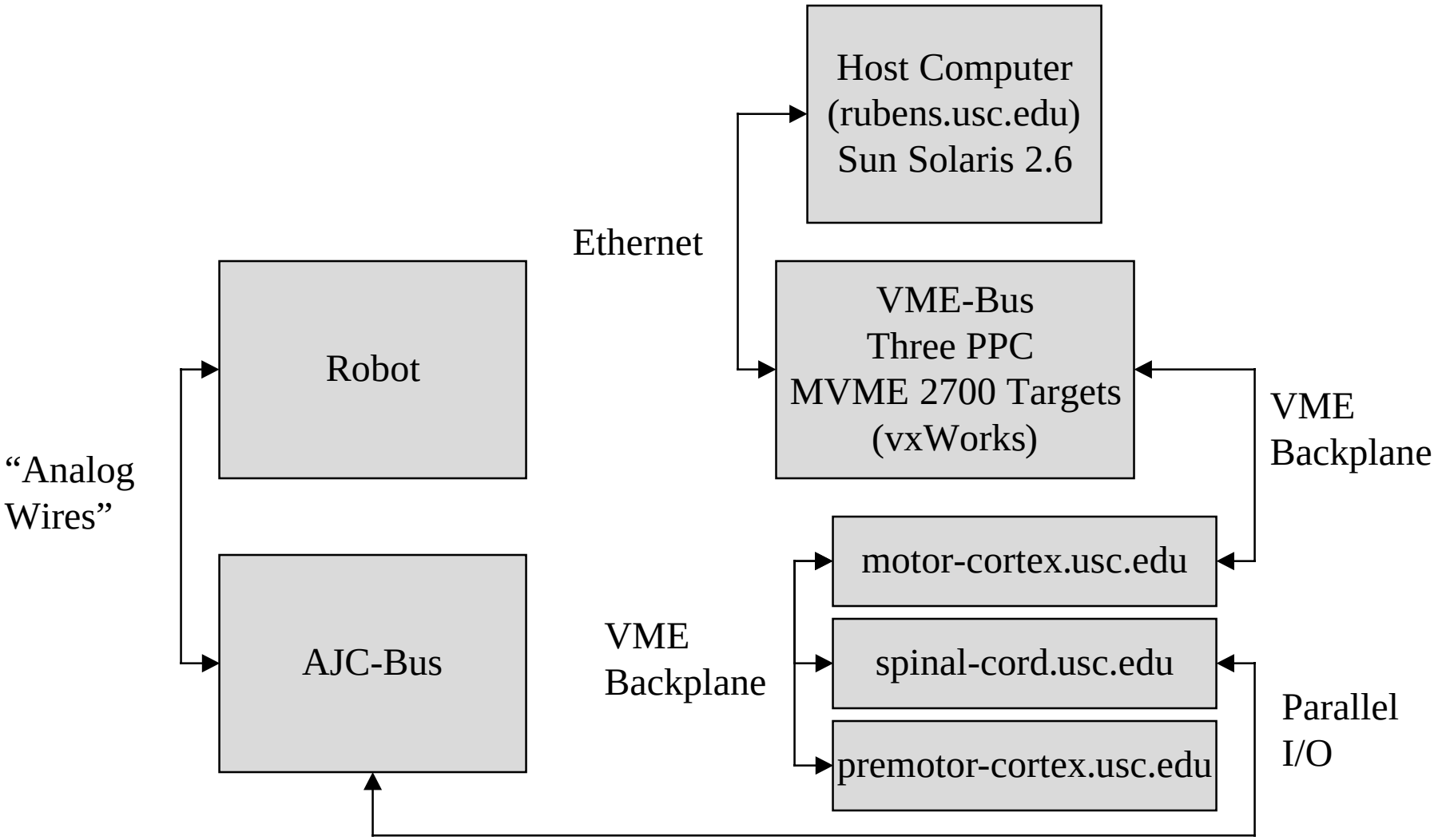
- At every timestep:
  - Read current positions from sensors
  - Calculate inverse static feedforward torque



How To Program The “Honey Sphere”?

- In Joint Coordinates:
  - Within a certain joint angle range of each DOF, add a negative component to the feedforward command proportional to the current DOF velocity
- In Cartesian Coordinates:
  - Check whether the endeffector is in the sphere
  - If yes, calculate viscous friction force according to endeffector velocity
  - Convert viscous force into joint torques with Jacobian Transpose
  - A “cheap version”: turn on viscous force in joint space if the endeffector is in the Cartesian sphere

Reminder: Setup of the Robotic System



What happens on Spinal-Cord?

- At high sampling rate (e.g., 500-1000Hz)
  - Read sensory data (positions, velocities, torques from load cells)
  - Process sensory data (filtering and numerical differentiation)
  - Receive desired trajectory and feedforward commands through inter-processor communication
  - Safety Check: Are the desired values in a permissible range
  - Generate total commands: u=PD+FF
  - Safety Check: Are commands in a permissible range
  - Send commands to the robot
  - Provide the state of the robot in shared memory

Interprocessor Communication in VxWorks: Shared Memory (VxMP)

- Initializing Shared Memory
  - + The following C-code creates a shared memory object “sm_joint_state” on the current processor

```
if (smNameFind("smJointState", (void**)&sm_joint_state, &mtype, NO_WAIT) == ERROR) {
   sm_joint_state = (SL_Jstate*)smMemCalloc(N_DOFS+1,sizeof(SL_Jstate*));
   if (sm_joint_state == NULL)
     return;
   error = smNameAdd("smJointState", (void*)smObjLocalToGlobal(sm_joint_state),
             sizeof(SL_Jstate)*(N_DOFS+1));
   if (error == ERROR)
     return;
   printf("Global shared memory for Joint States is set at 0x%x.\n",
          (char *)smObjLocalToGlobal((void*)sm_joint_state));
 }
```

Interprocessor Communication in VxWorks: Shared Memory (cont'd)

- Using the Shared Memory
  - The following C-code finds a shared memory object and stores its pointer in "sm_joint_state" on the current processor

```
 if (smNameFind("smJointState", (void**)&sm_joint_state, &mtype, NO_WAIT) == ERROR) {
    sm_joint_state = (SL_Jstate*)smMemCalloc(N_DOFS+1,sizeof(SL_Jstate*));
    return ERROR;
}

printf("Global shared memory for Joint States was found at 0x%x.\n",sm_joint_state);
```

Semaphores

- Binary Flags to prioritize and synchronize tasks on a processor or between processors
  - + Semaphores have two possible states:
    - Full (1)
    - Empty (0)
- Primarily two functions are used to handle semaphores
  - + SemGive
  - + SemTake

The Behavior of Semaphores

Figure 2-9 **Taking a Semaphore**

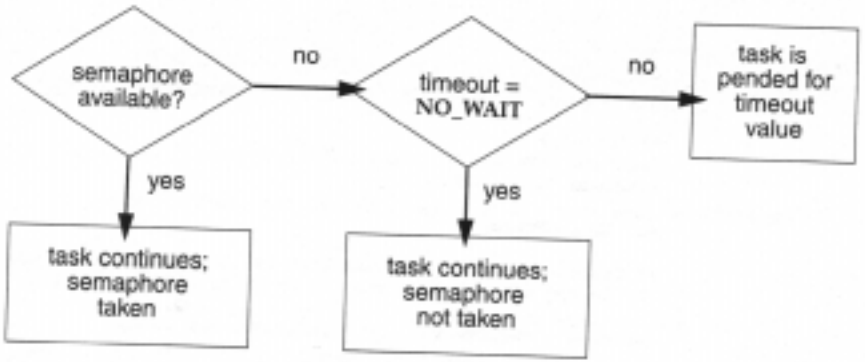
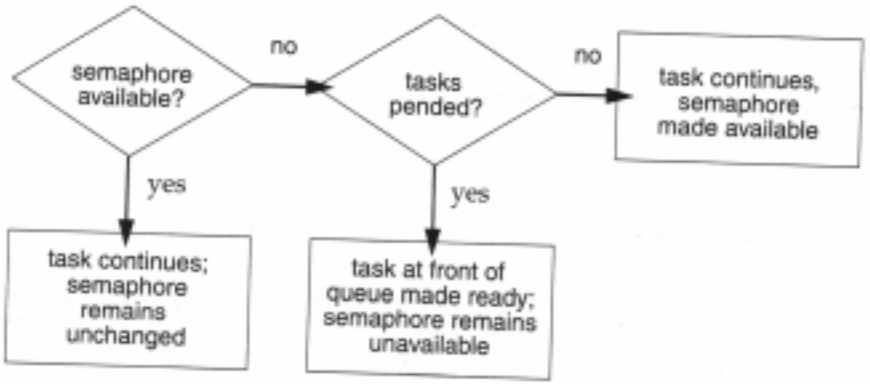


Figure 2-10 **Giving a Semaphore**



Shared Memory Semaphores

- Initializing a Shared Memory Semaphore

```
if (smNameFind("smJointStateSem", (void**)&sm_joint_state_sem, &mtype,NO_WAIT)==ERROR) {
   sm_joint_state_sem = semBSmCreate (SEM_Q_FIFO, SEM_FULL);
   if (sm_joint_state_sem == NULL)
     return;
   error = smNameAdd("smJointStateSem", (void*)sm_joint_state_sem, T_SM_SEM_B);
   if (error == ERROR)
     return;
   printf("Global shared semaphore for Joint State is set at 0x%x.\n",
         (char*)smObjLocalToGlobal((void*)sm_joint_state_sem));
 }
```

- Finding the Shared Memory Semaphore

```
if (smNameFind("smJointStateSem", (void**)&sm_joint_state_sem, &mtype,NO_WAIT)==ERROR) {
   return ERROR;
}
printf("Global shared semaphore for Joint State is set at 0x%x.\n",sm_joint_state_sem);
```

How to use Shared-Memory

- Create shared memory object
- Create shared memory semaphore
- For using the share memory:
  - Task semaphore
  - Read form or write to memory
  - Give semaphore

What is happening on Motor-Cortex?

- Motor-Cortex just executes Tasks
  - At high sampling rate (e.g., 500Hz)
    - Read sensory date from shared memory
    - Generate desired trajectory and feedforward commands
    - Write desired trajectory and feedforward commands to shared memory
- Tasks need to consist of (at least) 3 function
  - Initialization function of the task (not time critical)
  - Run function of the task (real-time)
  - Function to change the parameters of the task (not time critical)

Adding a New Task

- Write C-functions that contain the 3 required routines
  - + (templates: my_task.c will be provided)
- Compile the C-code
- Add to VxWorks:
  - – E.g., vxworks> ld < my_task.o
- Link the code into exisiting C-code
  - – E.g., vxworks> addTask(“cs545”,myinit,myrun,mychange)
    - + (this assumes you wrote the functions myinit, myrun,mychange)

What is happening in the INIT function?

- Bring the robot to an initial (safe) posture
- Initialize variables
- Trigger task execution

What happens in the RUN function?

- Assign appropriate values to feedforward commands and desired trajectory variables (“joint_state”, “joint_des_state”)
- Definition of these structures (see SL.h)

  SL_Jstate joint_state[N_DOF+1]

  SL_Dstate joint_des_state[N_DOF+1]
- Possible DOFs:
  - SFE (shoulder flex-extend)
  - SAA (shoulder aduction-abduction)
  - HR (humeral rotation)
  - EB (elbow)
  - WFE (wrist flex-extend)
  - WAA (wrist adduction-abduction)
  - Finger DOFS are not used

```
typedef struct { /* joint space state for each DOF */
  real   th;   /* theta */
  real   thd;  /* theta-dot */
  real   thdd; /* theta-dot-dot */
  real   u;    /* torque command */
  real   load; /* sensed torque */
} SL_Jstate;

typedef struct { /* desired values for controller */
  real   th;   /* desired theta */
  real   thd;  /* desired theta-dot */
  real   uff;  /* feedforward command */
} SL_DJstate;
```

What happens in the CHANGE function?

- Interactively change variable assignments, e.g., change some gains for the “honey sphere”