Today's Agenda

- HPS Project Design Review
  - **General Design Issues with Linux**
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Realtime Design, 101

- How many tasks in the system?
- How do the tasks share data?
- Without a UI, how do we test the system?
- How do our design decisions impact the system’s time budget?

Our Hero's Project Design

- Use of Linux as a runtime environment
  - Our scorecard indicates there are many options
    - Threads or processes?
    - Pipes or messages?
  - Do I have to write a device driver?
  - Will I have to modify the kernel?

Threads

- Less expensive in terms of system resources:
  - Shares memory space of parent process
  - Depending upon implementation, many thread services are handled in user-space
- Can synchronize/communicate with global memory variables (a double-edged sword)
- Newer and less well supported, in general

Processes

- More expensive in terms of system resources:
  - Memory (replicates parent process memory)
  - Time (kernel call to create/switch)
- Provide protection through unique address space
- Can be a unique program having its own ‘main’
- Are visible to many system tools (PID)

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - **Inter-process Communication**
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Inter-process Communication

- Pipes
  - Uni-directional
  - Uses file descriptors
  - Processes must be related
- Named Pipes (FIFOs)
  - Uni-directional
  - Special file in filesystem
  - Unrelated processes may read/write FIFOs
- System-wide size limit on any Pipe

SYS V IPC

- Introduced with AT&T System V.2 release of Unix
  - Semaphores (SEM)
  - Shared Memory (SHM)
  - Message Queues (MSG)
- Allows any two unrelated processes to communicate and/or synchronize

HPS Implementation

- HPS Design Decisions
  - Heavyweight process
  - SYSV IPC MSG
  - SYSV IPC SHM
- Why?
  - Leave room for performance improvement
  - Exposure to more interesting process-level programming entities
  - Push you out of your comfort level (pthreads have many similarities to traditional RTOS)

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - **Process List**
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

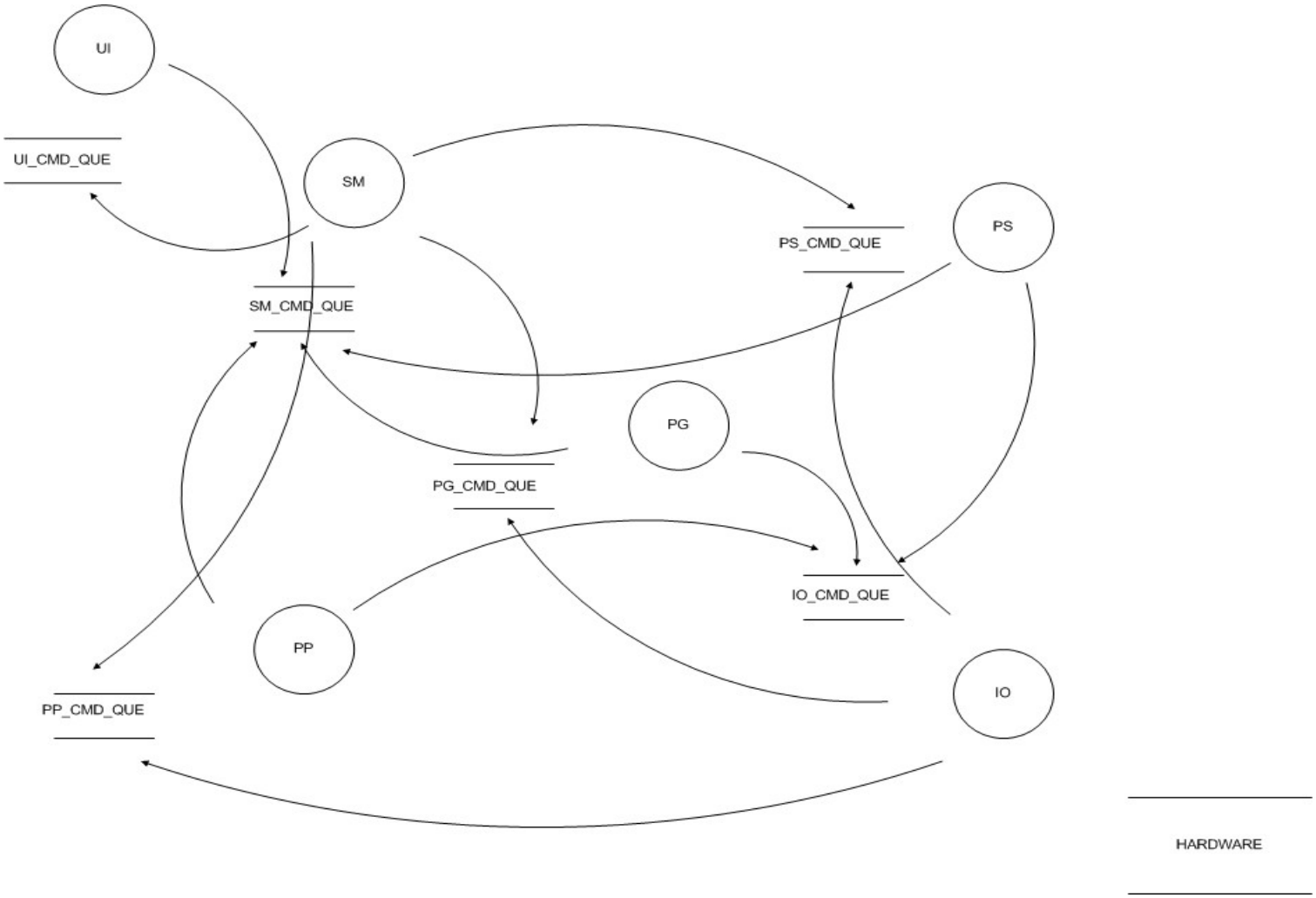
Design Aids

- Data Flow Diagram
- Process Diagram

Generic Data Flow Diagram

The Practical Guide to Structured Systems Design
Meilir Page-Jones, Yourdon Press
Page 60

HPS Data Flow Diagram



Process Diagram

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - **Message Protocol**
  - State Machine
  - I/O : Bits and Simulator
  - Tcl Test Harness

Part Stamper Messaging

- From the start, our hero's design has revolved around command messages for communication between the system tasks
- This is a methodology that is supported by all commercial embedded RTOS products
- We will uncover the advantages and disadvantages of this approach as we dig into our hero's design and implement it

Messaging in Linux

- One of the benefits of choosing Linux as a development environment is the multitude of ways to solve a problem
- Our hero chose to stick with the SYSV IPC mechanisms because these more closely map to traditional RTOS features
- The following is an excerpt from the msgop man page…

msgop man page

- To send or receive a message, the calling process allocates a structure that looks like the following
  - struct msgbuf {
    - long mtype; /* message type, must be > 0 */
    - char mtext[1]; /* message data */
  - };
  - but with an array mtext of size msgsz, a non-negative integer value. The structure member mtype must have a strictly positive integer value that can be used by the receiving process for message selection

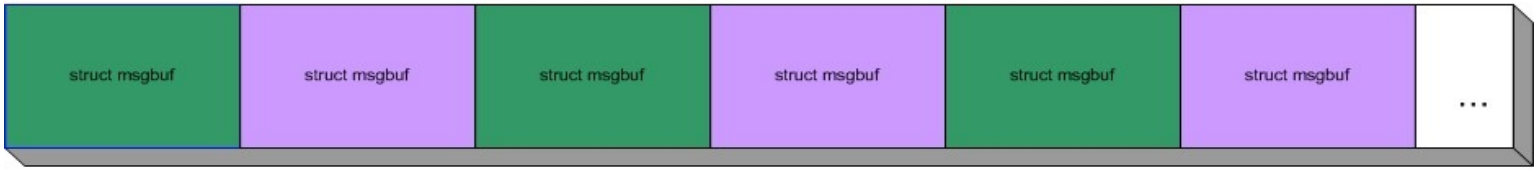
message.h

- #define MSGBUF_SIZE 256
- #define MSGBUF_PAYLOAD_SIZE (MSGBUF_SIZE-sizeof(long))
- Typedef struct {
  - MSG_LIST message;
  - PROC_LIST sender;
  - PROC_LIST receiver;
  - MP_ERROR error;
- } MESSAGE_PACKET;
- typedef struct {
  - MESSAGE_PACKET mp;
  - char data[MSGBUF_PAYLOAD_SIZE-sizeof(MESSAGE_PACKET)];
- } GENERIC_MSG;
- typedef struct {
  - long mtype;
  - GENERIC_MSG gm;

  } msgbuf;

SYSV IPC MESSAGE QUEUE

Msgget returns a handle to a kernel resource, a message queue




```
typedef struct {
   mtype   (long)
   gm      (GENERIC_MSG)
} msgbuf;
```

```
typedef struct {
  mtype          (long)
  gm.mp          (MESSAGE_PACKET)
  gm.data[236]   (char)
} msgbuf
```

```
typedef struct {
  mtype              (long)              4
  gm.mp.message      (MSG_LIST)          4
  gm.mp.sender;      (PROC_LIST)         4
  gm.mp.receiver;    (PROC_LIST)         4
  gm.mp.error;       (MP_ERROR)          4
  gm.data[2366];     (char)            236
} msgbuf                               (256)
```

Linux :	1.6 % data (memory) overhead	(4 / 256 = 0.0157)
PS Message Protocol :	6.3 % data (memory) overhead	(16/256 = 0.0625)
Total :	7.9 % data (memory) overhead	(20 / 256 = 0.078125)

Semantics of the PS Message Protocol

- Each task in the system has an input command message queue
- All message traffic to a task is routed to the input command message queue
- Every outbound command message generates an inbound reply packet, with fields from original message copied/filled in, indicating the command has completed, either successfully or in error

The Message Library

- void generic_msg_free(GENERIC_MSG *);
- GENERIC_MSG * generic_msg_init(MSG_LIST);

- GENERIC_MSG * msg_receive(MSG_LIST);
- void msg_send(GENERIC_MSG *);
- void msg_reply(GENERIC_MSG *);

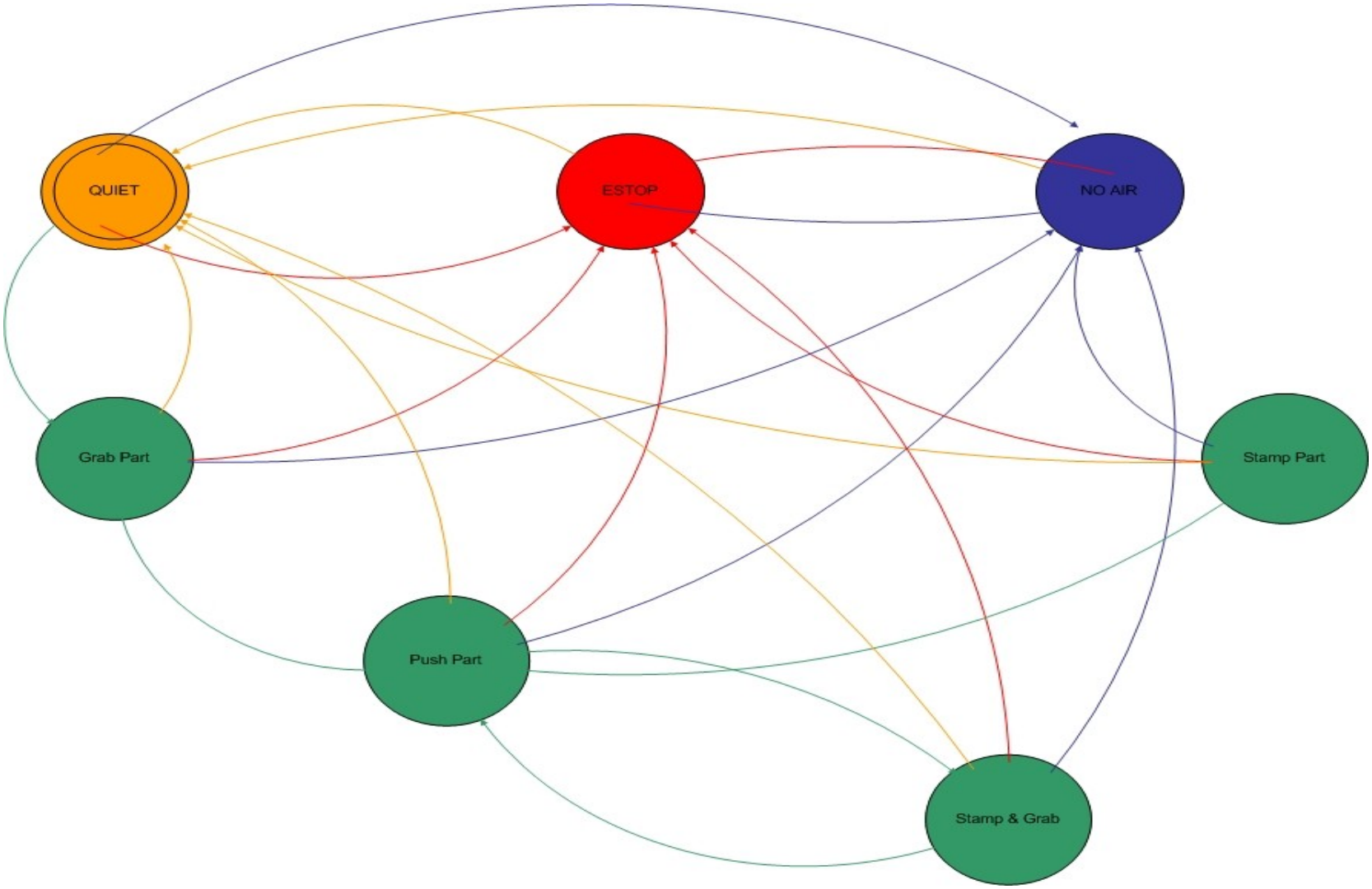
Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - **State Machine**
  - I/O : Bits and Simulator
  - Tcl Test Harness

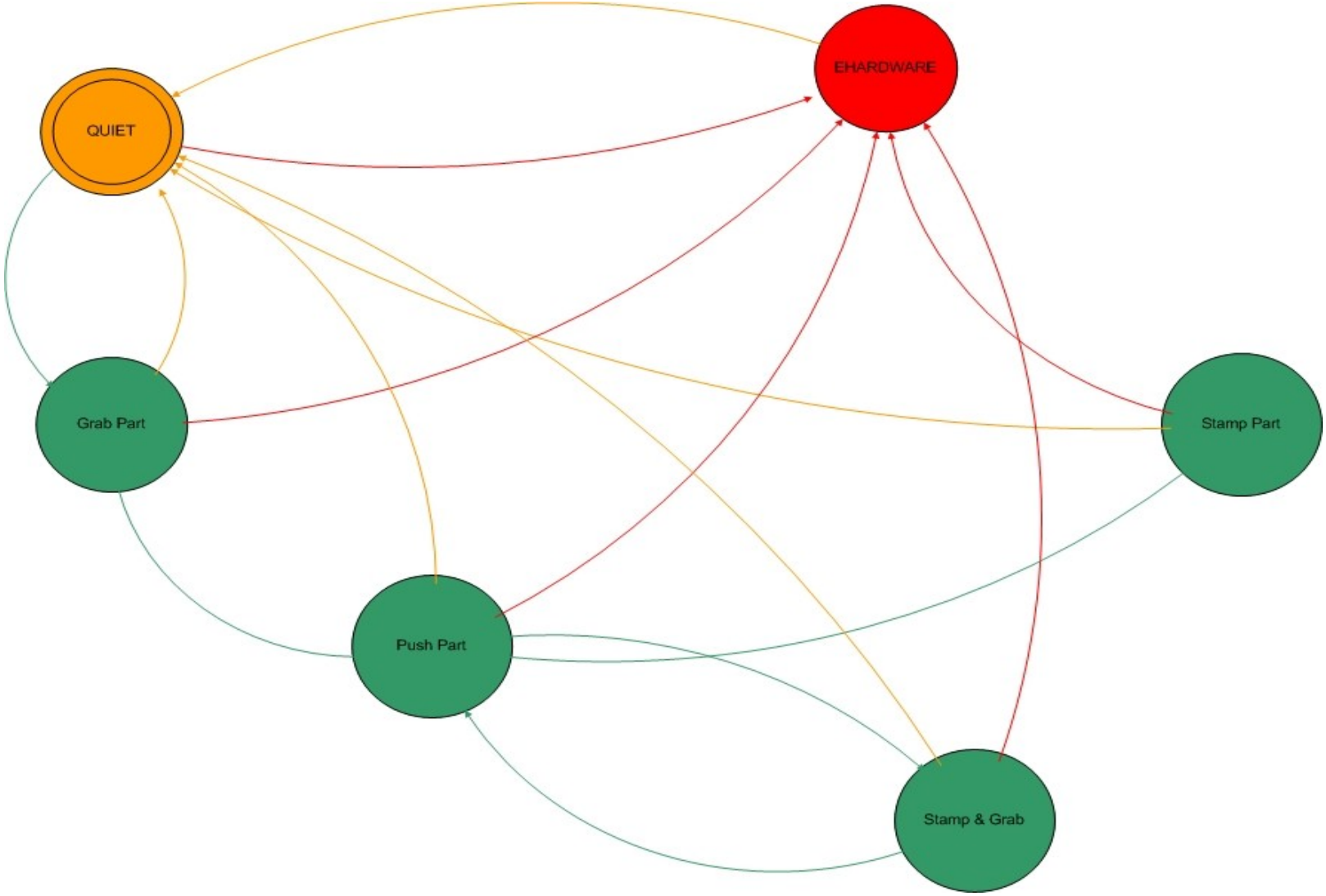
State Machine Basics

- A state machine consists of:
  - An initial state
  - A collection of states
  - A collection of inputs
  - A state transition table (the rules)
  - A terminal state
- An embedded system typically will not have a terminal state and will have a mechanism to return to the initial state (system reset)

HPS State Machine Diagram



HPS Reduced State Machine



HPS State Transition Table (TBD)

StateMachine Cheat Sheet

```
typedef enum {
        sm_start,                // 00
        sm_stop,                 // 01
        sm_return_state,         // 02
        sm_reset,                // 03
        sm_io_value_change,      // 04
        sm_cmd_last
} SM_CMD_LIST;

typedef struct {
        SM_CMD_LIST  command;
        int          continuous;
        int          state;
        IO_BIT_LIST  bit;
        int          value;
} SMCM;
```

```
typedef enum {
        state_quiet,             // 00
        state_ehardware,         // 01
        state_grab,              // 02
        state_push,              // 03
        state_stamp,             // 04
        state_stamp_grab,        // 05
        state_last,
        state_nostate = -1
} SM_STATE;
```

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - **I/O : Bits and Simulator**
  - Tcl Test Harness

The I/O Library

- int turn_off(IO_BIT_LIST);
- int turn_on(IO_BIT_LIST);
- int value_of(IO_BIT_LIST);
- int wait_until(IO_BIT_LIST);

I/O Monitor Cheat Sheet

```
typedef enum {
        io_turn_off,    // 00
        io_turn_on,     // 01
        io_value_of,    // 02
        io_report,      // 03
        io_reset,       // 04
        io_cmd_last
} IO_CMD_LIST;
```

```
typedef enum {
        cylAretracted,    // 00
        cylAextended,     // 01
        cylAsolenoid,     // 02
        cylBretracted,    // 03
        cylBextended,     // 04
        cylBsolenoid,     // 05
        cylCretracted,    // 06
        cylCextended,     // 07
        cylCsolenoid,     // 08
        airpressure,      // 09
        Estop,            // 10
        IO_LAST
} IO_BIT_LIST;
```

I/O Simulator

- By using the same technique that all High School and College Physics problems are based on, a very simple and effective solution is found
- When applying all of the forces for an equation, the Physics student is always cautioned to assume the force from friction is neglible and set it to 0
- In the Part Stamper Simulator, I/O activation times are negible, as soon as a control line is toggled, the input values are adjusted.

Today's Agenda

- HPS Project Design Review
  - General Design Issues with Linux
  - Inter-process Communication
  - Process List
  - Message Protocol
  - State Machine
  - I/O : Bits and Simulator
  - **Tcl Test Harness**

Tcl : What is it?

- Tool Command Language
  - Similar to other shell languages, e.g.
    - Perl
    - Bash
    - C shell
  - With some important differences…
    - Tk widget (easy window programming)
    - Embeddable
    - Extensible

The C / Tcl Philosophy

- As espoused by John K.Ousterhout…
  - To make a Tcl application as flexible and powerful as possible, you should organize its C code as a set of new Tcl commands that provide a clean set of primitive operations…The purpose of the C code is to provide basic operations that make it easy to implement a wide variety of useful scripts.

Tcl and the Tk Toolkit
John K. Ousterhout, Addison-Wesley
Page 281

Tcl Meets Part Stamper

- pstcl adds following commands :
  - sendcmd : send ps command message
- pstcl adds following linked variables :
  - Log_trace
  - Log_message
  - Log_debug
  - Log_tcl
  - Log_simulate

pstcl : sendcmd

- New Tcl command : sendcmd
- Connects pstcl to rest of PartStamper application
- Allows any command message to be generated and sent in the system
- With a message passing software architecture, this single test command allows virtually all aspects of the system to be tested. With a script-capable test harness, regression tests can be created from the engineer's interactive test tool.

sendcmd Cheat Sheet

```
typedef enum {
        PROC_NONE = -1,
        PROC_pgrabber = 0,
        PROC_ppusher = 1,
        PROC_pstamper = 2,
        PROC_smachine = 3,
        PROC_ui = 4,
        PROC_tcl = 5,
        PROC_main = 6,
        LAST_PROC
} PROC_LIST;
```

```
typedef enum {
        MP_NOERROR,             // 00
        MP_NOMESSAGE,           // 01
        MP_NOSENDER,            // 02
        MP_NORECEIVER,          // 03
        MP_BADCOMMAND,          // 04
        MP_NULLRETURN,          // 05
        MP_UNHANDLEDMSG,        // 06
        MP_BADSTATE,            // 07
        MP_LAST
} MP_ERROR;
```

```
typedef enum {
        PS_COMMAND_MSG   // 00
        SM_COMMAND_MSG   // 01
        UI_COMMAND_MSG   // 02
        PS_GENERIC_MSG   // 03
        LAST_MSG
} MSG_LIST;
```

Part-Stamper Cheat Sheet

```
typedef enum {
        ps_grab_part,     // 00
        ps_push_part,     // 01
        ps_stamp_part,    // 02
        ps_return_state,  // 03
        ps_reset,         // 04
        ps_cmd_last
} PS_CMD_LIST;

 typedef struct {
        PS_CMD_LIST  command;
        int                  state;
 } PSCM;
```