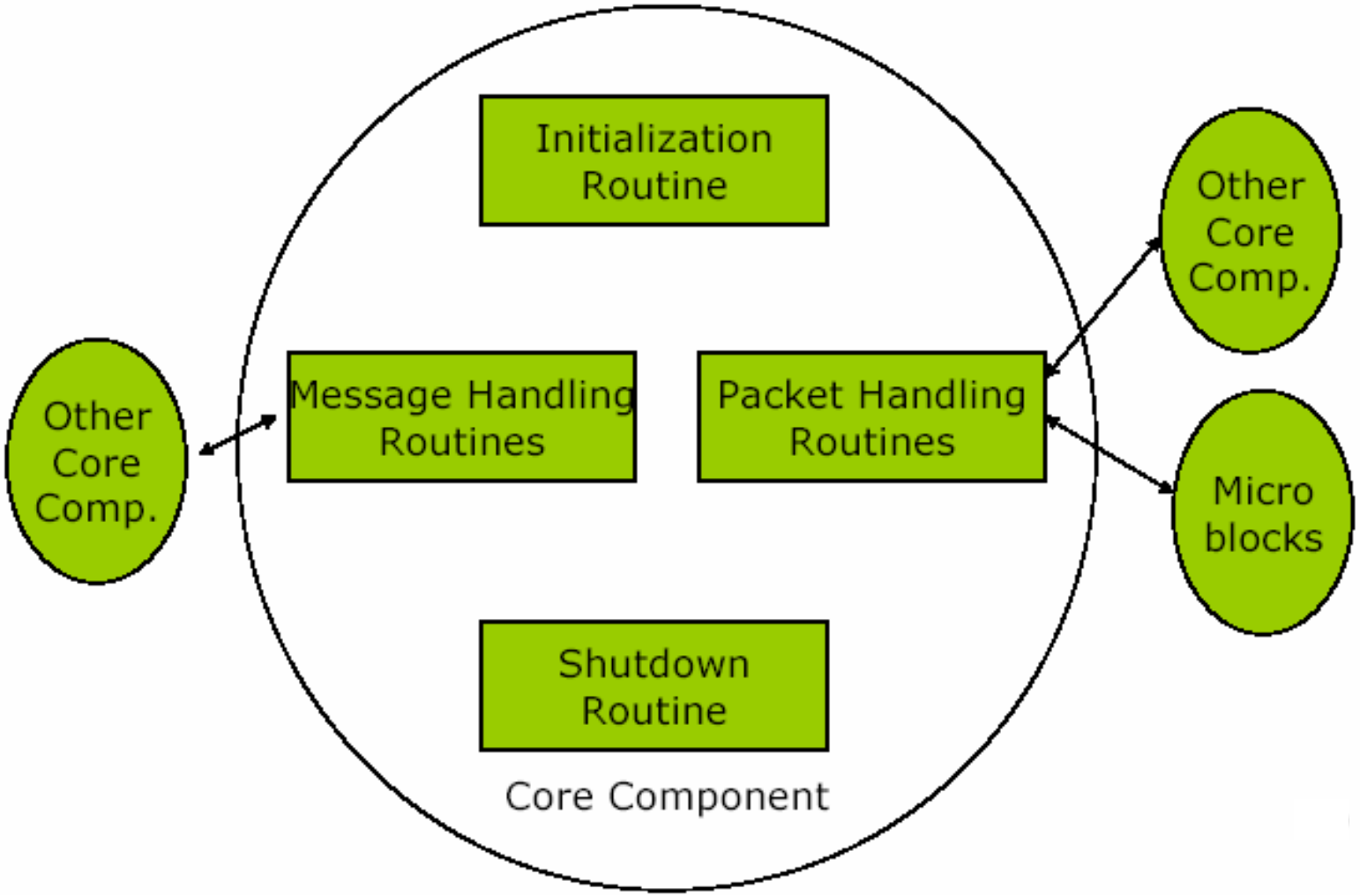
18544: Network Design and Evaluation

Core Components Review
and Using Hardware
Sept 19, 2006

Agenda

- Core Component Programming
  - Memory management
  - Patching symbols
  - Exchanging Packets
- Using Hardware
  - Building the core
  - Booting up the application

Structure of a Core Component



Memory Management [1]

- Memory used by core components managed using malloc and free
- Resource Manager manages memory accessed by the microengines
- To allocate memory from the Intel XScale core using Resource Manager

```
IX_EXPORT_FUNCTION ix_error
   ix_rm_mem_alloc(
        ix_memory_type arg_MemType,
        ix_uint32 arg_MemChannel,
        ix_uint32 arg_Size,
        void** arg_pMemoryAddr);
```

Memory Management [2]

- Address microengines use may be different than Intel XScale core
- For example, if you want to give an address returned by `ix_rm_mem_alloc` to the microengines, convert it to an address which can be used by microengines using `ix_rm_get_phys_offset` function
- Also tells the memory type and channel if required

Trace code

Patching Load-time Constants [1]

- A way of communication between the XScale core and microengines
- Used in patching memory locations determined when the core component is initialized
  - The values of these constants are not known at compile time, but rather are determined at the code loading time
  - The values cannot be changed once they are set (can be only changed by stopping and reloading the microengines)

Patching Load-time Constants [2]

- Signature of load-time constants:

```
IX_EXPORT_FUNCTION ix_error
    ix_rm_ueng_patch_symbols(
          ix_uint32 arg_MENumber,
          ix_uint32 arg_SymbolsNumber,
          const ix_imported_symbol
          arg_aSymbols[]);
```

- In microengine assembly code:

```
.import_var ETHERNET_SYMBOL_NAME
```

or in microengine C:

```
int ETHERNET_DATA =
    LoadTimeConstant ("ETHERNET_SYMBOL_NAME");
```


Trace code

Example

```
//Allocate shared memory for the control block
err = ix_rm_mem_alloc(IX_MEMORY_TYPE_SRAM,
  channel, sizeof(ethernet_control_block),
  (void**)eth_context->control_block);

if (err != IX_SUCCESS)
// error handling

err = ix_rm_get_phys_offset( eth_context-
  >control_block, NULL, NULL, NULL,
  &control_block_phys );

if (err != IX_SUCCESS)
// error handling
```

Example (Cont.)

```
//Now patch the control block symbol for all
//of the microengines on which Ethernet
//microblocks will run
for (i = 0; i < sizeof(me_numbers); i++) {
  ix_imported_symbol symbol;
  symbol.m_value = (ix_uint32)
  control_block_phys;
  symbol.m_name = ETHERNET_SYMBOL_NAME;
  err = ix_rm_ueng_patch_symbols(i, 1, &symbol);
  if (err != IX_SUCCESS)
  //error handling
}
```

Packet Handling

- The XScale Core handles infrequently - arriving packet types that require more complicated processing
  - Control and configuration packets
    - Routing update packets
  - Packets requiring lengthy processing
    - IP packets with options
  - Correcting erroneous conditions
    - Sending ICMP packets
  - Other packets not handled by microblocks

Packet Handling: MB⇨CC [1]

- The core component needs to register the packet handler routines
- The signature of the handler function:

```
ix_error (* ix_pkt_handler) (
  ix_buffer_handle arg_hDataToken,
  ix_uint32 arg_ExceptionCode,
  void* arg_pComponentContext)
```

Packet Handling: MB⇨CC [2]

- Use `ix_cci_cc_add_packet_handler` to register the handler in the core:

```
ix_error
  ix_cci_cc_add_packet_handler(
  ix_cc_handle arg_hComponent,
  ix_uint32 arg_InputID,
  ix_pkt_handler arg_Handler,
  ix_input_type arg_SourceType );
```


Trace code

Packet Handling: MB⇨CC [3]

- When microblock wants to send packets to its core component:
  - it sets the next block value to the unique packet handler identifier (i.e. Input ID)
  - The microblock can also set a 32-bit exception code

Trace code

Packet Handling: MB⇨CC [4]

- The following microengine C code sends packet to the Ethernet core component:

```
// If the entry is not valid, drop this
packet
if(!table_entry.valid) {
    dl_set_exception(ETHERNET_EXCEPTION_ID,0);
    dlNextBlock = IX_EXCEPTION;
    return;
}
```

Exception code

ID of the calling microblock (mapping to arg_InputID in ix_cci_cc_add_packet_handler)

Dispatch loop will catch this packet and notify the core

Good Reference: IAX Framework Development Manual

Packet Handling: CC⇨MB [1]

- Core components can send packets to microblocks or other core components using the following API:

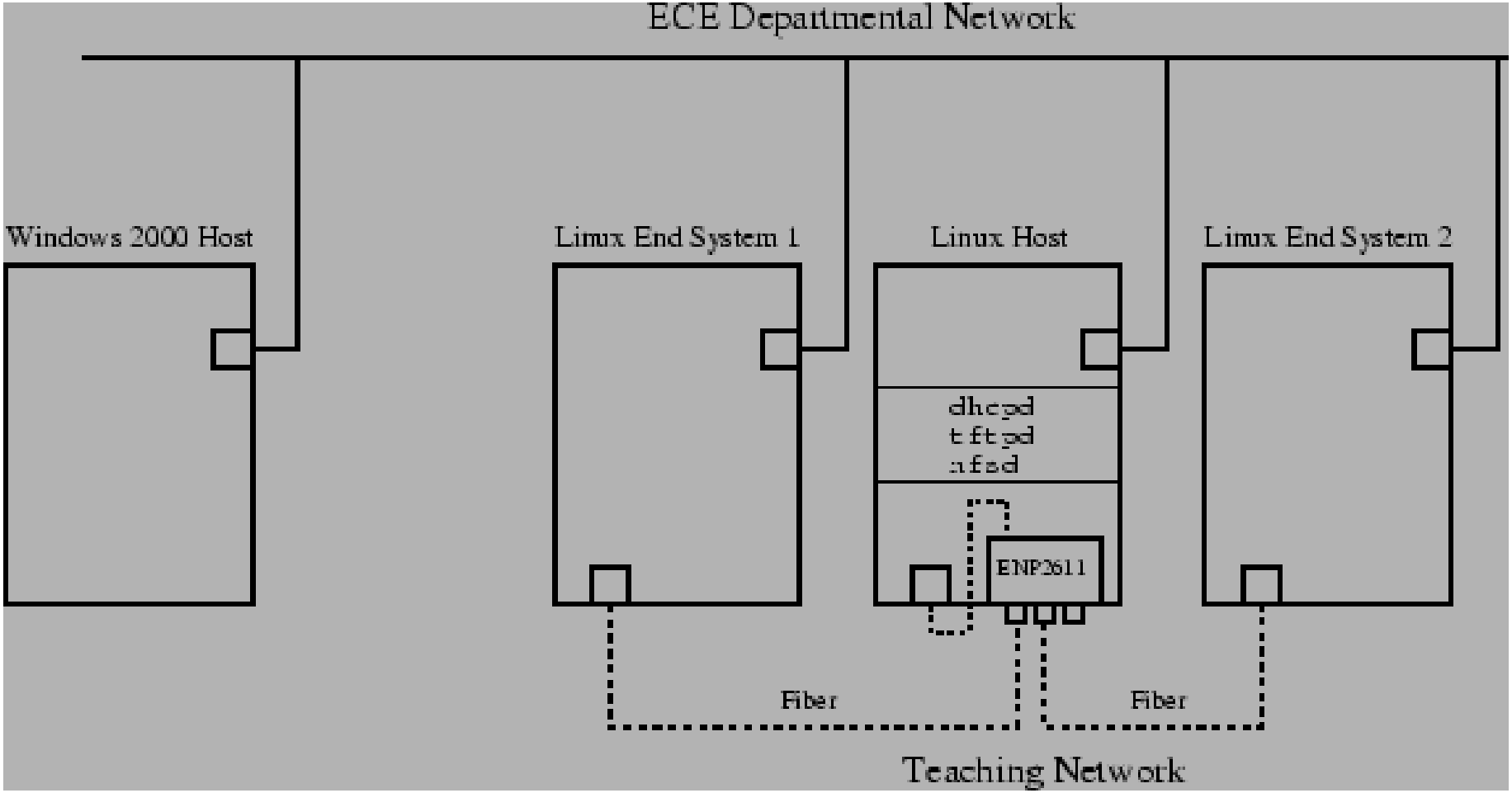
```
ix_error ix_rm_packet_send(
  ix_communication_id
       arg_CommunicationId,
  ix_buffer_handle arg_MessageBuffer,
  ix_uint32 arg_UserData );
```

Destination Microblock ID

- The dl_source macro polls the scratch ring to retrieve the packet.

Using the Hardware

Network Setup: Physical Layout



Network Setup (Cont.)

- Linux hosts (the black boxes) connected to the ENP-2611 board to form a teaching Network
- Radisys ENP-2611 board
  - Hosted by one of the Linux machines
  - Carries the Intel IXP 2400 Network Processor
  - 3 fiber ports to connect to the hosts (see the back of the black box hosting the board)
  - Runs embedded Linux on XScale processor
- Windows machine connected to the ENP-2611 board via RS-232

Building the Core [1]

- Download proj_544fall04_core.tar.gz from the course site to `/opt/ixa_sdk_3.5/src` on the Linux host. Untar the file using the command

  ```
  tar xzvf proj_544fall04_core.tar.gz
  ```

  Note: the code is already copied to the Linux host by TAs so you can simply skip this step if no change is made on the code.
- Make sure that `/opt/hardhat/previewkit/arm/xscale_be/bin` is in your PATH. (Use `echo $PATH` in Bash Shell)

Build the Core [2]

- On the Linux host, cd to the directory

  `/opt/ixa_sdk_3.5/src/applications/proj_544fall04/ipv4_enp2611`

- Issue the command

  ```
  make -f Makefile.linux_kernel
  ```

  Note: to rebuild the code, issue the following command before the above one:

  ```
  make -f Makefile.linux_kernel clean
  ```

- cd to the directory

  `/opt/hardhat/previewkit/arm/xscale_be/target/opt/proj_544fall04/ipv4_enp2611`

- Issue the command

```
cp linux_kernel/xscale_be/ixp2400/debug/*
```

Linux Host Configuration

- Runs DHCP daemon
  - Allows auto configuration of the board via crossover cable
- Provides NFS service to give storage access – board have no onboard disk

Getting things to work!

- Run the script

  `/opt/ixp_scripts/host_startup.sh`

  - This will bring up the DHCP daemon, TFTP daemon, and the NFS service
  - Setup a HyperTerminal Session (Baud Rate: 57600, Flow Control: None) on the Windows machine
  - Reset the ENP-2611 board (press “red” button on the board)
  - Login as root through the HyperTerminal session (no password required)

Running ipv4_enp2611 on the hardware

- Compile the microengine code on the Windows machine (set the preprocessor symbol to USE_IMPORT_VAR and not SIMULATION)
  - Must use IXA SDK 3.5 for compilation
  - Can use IXA SDK 4.2 for simulation to utilize new features
- Copy the microengine image `ipv4_enp2611.uof` to the Linux host at

  `/opt/hardhat/previewkit/arm/xscale_be/target`
  `/opt/proj_544fall04/ipv4_enp2611`

Note: the ipv4_enp2611.uof file is already copied to the Linux host by TAs so you can simply skip this step if no change is made on the code.

Running ipv4_enp2611 on the hardware

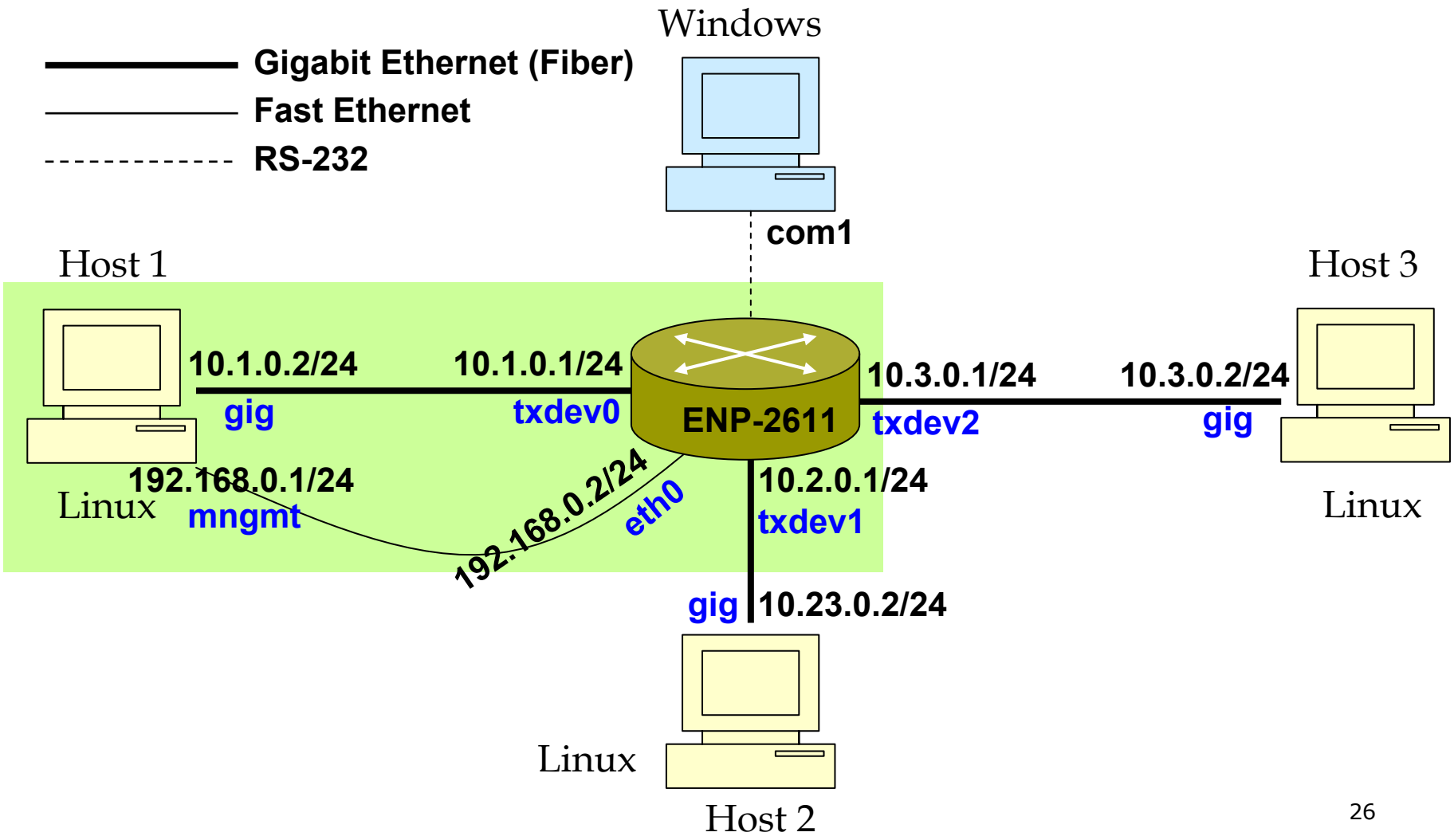
- On the ENP-2611 board, run the script `/root/init_script.sh`
- On each Linux machine, run `/opt/ixp_scripts/endsys_startup_X.sh` (X=1 if machine hosting ENP-2611 or X=2 otherwise)

PINGing the hosts

- Now we should be able to PING across the board

  e.g. ping 10.x.0.1 (x=1,2,3 are the fiber ports on the ENP-2611) ping 10.x.0.2 (x=1,2,3 are the host machines)

Lab Setup: Logical Layout



Quick Summary

Action (in sequence)	Description
Host: /opt/ixp_scripts/ host_startup.sh	Start DHCP/TFTP/NFS services on the host
Host & ENP-2611: Doing DHCP	ENP-2611 gets IP, image filename, and NFS mount directory
Host & ENP-2611: Doing TFTP	ENP-2611 downloads the kernel image. Start running the kernel.
ENP-2611: /root/init_script.sh	Install loadable kernel modules (CCs) Initialize IP address and ARP/route tables on the board
Host: /opt/ixp_scripts/ endsys_startup_X.sh	Set IP addresses for the gigabit Eth I/F. Initialize ARP/route tables on the hosts

Where to Obtain the Code Base…

- Microengine (WorkBench):

  http://www.cs.cmu.edu/afs/andrew.cmu.edu/course/18/544/dl/proj_544fall04.zip

- Core Code Base (Linux):

  http://www.cs.cmu.edu/afs/andrew.cmu.edu/course/18/544/dl/proj_544fall04_core.tar.gz

Questions?

It is recommended to use online instructions posted on the course website for setting up the hardware.