SOC-Design Lab3 FIR

github link *(https://github.com/Charlee0207/SOC-Design/tree/main/Lab3)*

report link *(https://hackmd.io/ZRIFy4t8Rai-nfZyGJMa9A?both)*

NTHU EECS24 109020014 李承澔

Brief introduction

Resource Constraints

This lab focuses on implementing a Finite Impulse Response (FIR) engine in Verilog. The FIR design is constrained to have 11 taps, and it must use only one adder and one multiplier.

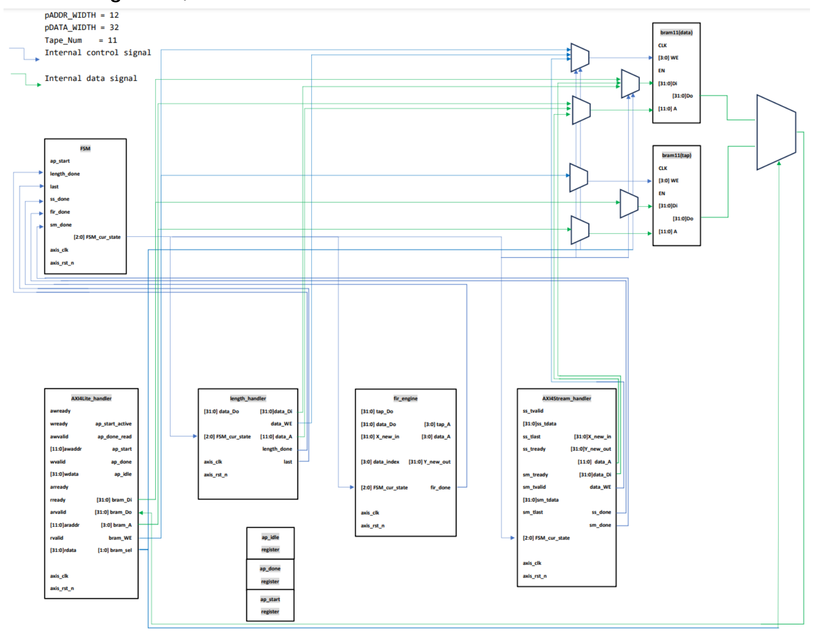
Communacation Protocal

To control and monitor the FIR system, the host or testbench initiates the system by signaling the `ap_start`. The system's operational status can be gauged through `ap_idle` and `ap_done` signals, which the host/testbench continuously polls. This communication is established using the AXI-Lite protocol.
Subsequent to the initial communication, the input data `X[n]` is streamed into the system . In response, the output `Y[n]` is streamed back to the host. This data transfer is using the AXI-Stream protocol.
Furthermore, both the coefficient `h[n]` and the input data `X[n]` need to be stored in the BRAM.

Block diagram

Datapath

6 modules: `FSM`, `AXI4Lite_handler`, `length_handler`, `fir_engine`, `AXI4Stream_handler`



Control signals

In `FSM`, the `[2:0] FSM_cur_state` stores the main control signal, including `IDLE`, `CHECK_LEN`, `STREAM_IN`, `FIR_COMP`, `DONE`, `STREAM_OUT`.
And the `ap_start`, `length_done`, `last`, `ss_done`, `fir_done` and `sm_done` are assertted by host, `length_handler`, `length_handler`, `AXI4Stream_handler`, `fir_engine` and `AXI4Stream_handler`, respectively.

Operation description

FSM phases

1. **IDLE Phase**
   - The FIR system remains in an idle state.
   - The host have to program the coefficients, represented by `h[n]`, and also set the data length, `N`, during this phase.
2. **CHECK_LEN Phase**
   - The `length_handler` is invoked during this phase.

    - Its primary function is to decrement the data length value by 1.
    - When the data length reach zero, the `length_handler` will raise the `last` signal, signifying the end of data input.
3. **STREAM_IN Phase**
    - The phase signifies the activation of the `AXI4Stream_handler`, initiating the reception of input data denoted by `X`.
    - Once the reception process is complete, the `ss_done` signal is asserted to indicate the successful receipt of data.
4. **FIR_COMP Phase**
    - This is the computation phase, where the `fir_engine` takes over.
    - It processes the newly received data, the existing data in the shiftRAM, and the tapRAM to generate the desired output.
    - Upon successful computation, the `fir_done` signal is asserted.
5. **STREAM_OUT Phase**
    - The `AXI4Stream_handler` is re-invoked, but this time to dispatch the newly computed result `Y`, back to the host.
    - After successfully transmitting the result to the host, the `sm_done` signal is raised to signify the completion of the transmission.
6. **DONE Phase**
    - This phase signifies the completion of the current iteration of computation. The `FSM`, at this juncture, checks for the assertion of the `last` signal.
    - Based on the status of this signal, the FSM decides its next state - either returning to the `IDLE` phase or proceeding to the `CHECK_LEN` phase.

Receiving coefficient `h[n]` and `ap_start`

During the `IDLE` phase, the system is configured to accept the coefficient `h[n]` and the signal `ap_start`.
The process for this reception and transmission is managed by the `AXI4Lite_handler`.
In scenarios where the handler identifies simultaneous read and write requests, priority is given to the write request.
Its workflow can be summarized as follows:

```
//***Handling write address channel***
// Check awvalid and write status, and assert awready
if(awvalid && !write_flag) begin
    awready <= 1; wready <= 0; write_flag <= 1; //...other operation
end
// Deassert awready and assert wready
else if(awvalid && awready && write_flag) begin
    awready <= 0; wready <= 1; write_flag <= 1; //...other operation
end
//***Handling write data channel***
// master hasn't assertted wvalid yet
else if(wready && !wvalid && write_flag) begin
    awready <= 0; wready <= 1; write_flag <= 1; //...other operation
end
// master assert wvalid, transmit data and deassert wready and write flag
else if(wready && wvalid && write_flag) begin
    awready <= 0; wready <= 0; write_flag <= 0; //...other operation
end
// Write transaction failed, reset all bit
else begin
    awready <= 0; wready <= 0; write_flag <= 0;
end

//========================================================
//***Handling read address channel in the smae way***
//***Handling read data channel in the smae way***
```

Upon successful reception of data, the next step involves determining where to write this data based on the memory-mapped address.

- **The programming target is `ap_start`**
  Since `ap_start` is W/R, `ap_start` cannot be multi-driven by `AXI4Lite_handler` and top module.
  Hence, it assert `ap_start_active` signal try to

program target. And later when detecting a posedge clk, the `ap_start` controller will update it using value of `ap_start_active` .

- **The programming target is `coefficient`**
  It will subtract the `awaddr` by `'h20` , to map to the bram address.
  And setup the `bram_sel` to tapRAM, and `bram_WE` to 1.
- **The programming target is `length`**
  The length variable is store at `0x00` at dataRAM.
  It will setup `bram_A` to `0x00` , `bram_sel` to dataRAM, and `bram_WE` to 1.

Checking current tap data

As mentioned above, the `AXI4Lite_handler` read request is less prior to write one.
After finishing write request, if `arvalid` is asserted, the handler will connect `bram_A` to corresponding memory position according to `araddr` .
The tap data should be accessable 2 cycles later from bram. So it connects them to rdata and ends this transaction.

Checking current data number

After `ap_start_active` asserted and `ap_start` updated at next posedge clock, the FSM logic changes state to `CHECK_LEN` .
In this stage, the `length_handler` would be invoked to subtract the data length by 1, and assert `length_done` when finishing.
If the data length is 0, which means this is the last iteration, it asserts `last` signal to the system.

Receiving data `X[n]`

After `length_done` asserted, the FSM logic changes state to `STREAM_IN` .
The `AXI4Stream_handler` receive only 1 data of `X` by asserting `ss_ready` once when `ss_valid` once.
And after that, it asserts `ss_done` to tell FSM change state.
Rather than stored in FF, the new stream-in `X` would be stored to shiftRAM immediately. And it will connect new stream-in `X` outside to `X_new_in` , for `fir_engine` computing the first accumulation.

Accessing shiftRAM and tapRAM to do computation

In `fir_engine` , we use a `cycle_counter` with case selection to indicate what operation should be done at this point as following.
At `cycle_counter = 0` , we feed the initial bram address.
The bram data is available at `cycle_counter = 2` , and do the computation. After computation, it will assert `fir_done` when `cycle_counter = 12` and deassert at next cycle.
In this implementaion, we use a 11DW bram, the `0x00` is used to store `length` . To maintain the shift data order, we use `data_index` to indicate the position and update at `CHECK_LEN` state.

```
// ... other operation
else if(FSM_cur_state==FIR_COMP)begin
    case(cycle_counter)
    'd0: begin
        tap_A <= 0; data_A <= data_index;
        // ... other operation
    end
    'd1: begin
        tap_A <= tap_A + 1;
        data_A <= (data_A=='d1) ? 'd10 : data_A-1;
        // ... other operation
    end
    'd2: begin
        tap_A <= tap_A + 1;
        data_A <= (data_A=='d1) ? 'd10 : data_A-1;
        accumulation <= tap_Do * X_new_in;
        // ... other operation
    end
    'd3, 'd4, 'd5, 'd6, 'd7,
    'd8, 'd9, 'd10,'d11: begin
        tap_A <= tap_A + 1;
        data_A <= (data_A=='d1) ? 'd10 : data_A-1;
        accumulation <= (bram_valid)
             ? accumulation + tap_Do*data_Do : accumulation;
        // ... other operation
    end
    'd12: begin
        tap_A <= 0; data_A <= 0;
        accumulation <= accumulation;
        // ... other operation
    end
    'd13: begin
        tap_A <= 0; data_A <= 0;
        accumulation <= accumulation;
        // ... other operation
    end
    default: begin
        tap_A <= 0; data_A <= 0;
        accumulation <= accumulation;
        // ... other operation
    end
    endcase
end
```

Transmitting the result `Y[n]`

After `fir_done` asserted, the FSM logic changes state to `STREAM_OUT`.
The `AXI4Stream_handler` will connect `ss_tdata` to the `accumulation` FF in `fir_engine` directly.
And just like `STREAM_IN` stage, doing the stream out use similar way.

FSM `next_state` decision

The `next_state` depends on such `<stageName>_done` signals.

```
always@(*)begin
case(FSM_cur_state)
IDLE: begin
    if(ap_start) FSM_next_state = CHECK_LEN;
    else FSM_next_state = FSM_cur_state;
end
CHECK_LEN: begin // update the data length
    if(length_done) FSM_next_state = STREAM_IN;
    else FSM_next_state = FSM_cur_state;
end
STREAM_IN: begin // axis slave receive X data, until ss_done asserted
    if(ss_done) FSM_next_state = FIR_COMP;
    else FSM_next_state = FSM_cur_state;
end
FIR_COMP: begin // fir engine do computation, until fir_done asserted
    if(fir_done) FSM_next_state = STREAM_OUT;
    else FSM_next_state = FSM_cur_state;
end
STREAM_OUT: begin //axis master transmit new result, until sm_done assereted
    if(sm_done) FSM_next_state = DONE;
    else FSM_next_state = FSM_cur_state;
end
DONE: begin // check continue do fir or goto idle
    if(last) FSM_next_state = IDLE;
    else FSM_next_state = CHECK_LEN;
end
default: begin FSM_next_state = FSM_cur_state; end
```

Generation of `ap_done` and `ap_idle`

- `ap_done`
  If the system finishs the last computation and stream-out, assert `ap_done`.

```
always@(posedge axis_clk) begin
    if(!axis_rst_n)
        ap_done <= 0;
    // deassert ap_done when ap_done was read
    else if(ap_done_read)
        ap_done <= 0;
    // assert ap_done when finished process and transection
    else if(FSM_next_state==DONE && last)
        ap_done <= 1;
    else
        ap_done <= ap_done;
end
```

- `ap_idle`
  If the `fir_engine` finishs the last computation but not streaming out yet, assert `ap_idle`.

```
always@(posedge axis_clk) begin
    if(!axis_rst_n)
        ap_idle <= 1;
    else if(ap_start)
        ap_idle <= 0;
    // assert ap_idle when fir finished last processing
    else if(FSM_next_state==STREAM_OUT && last)
        ap_idle <= 1;
    else
        ap_idle <= ap_idle;
end
```

Resource usage (FF, LUT, BRAM)

FF	LUT	TAP BRAM(behavior)	DATA BRAM(behavior)	BRAM(physical)
148 (0.14%)	398 (0.75%)	11	11	0(0%)

Timing report

Minimum clock period

Minimum clock period
10.842

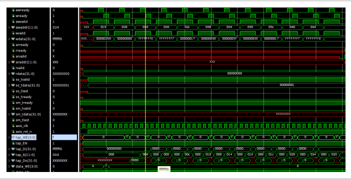
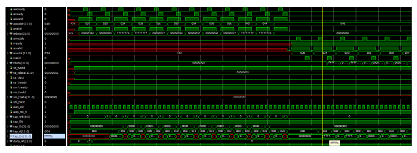
Timing on longest path, slack

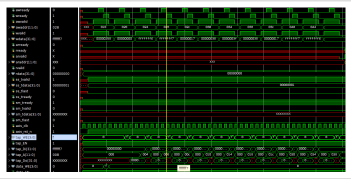
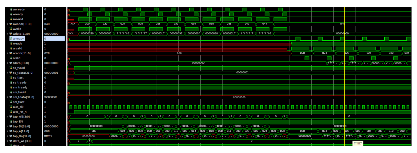
Timing on longest path	Slack
10.704	0.001

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.001ns  (required time - arrival time)
  Source:                 FSM_onehot_genblk1.FSM_cur_state_reg[4]/C
                            (rising edge-triggered cell FDRE clocked by axis_clk
  Destination:            genblk1.u_FIR/genblk1.shift_reg[31]/D
                            (rising edge-triggered cell FDRE clocked by axis_clk
  Path Group:             axis_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.842ns  (axis_clk rise@10.842ns - axis_clk rise@0.000
  Data Path Delay:        10.704ns  (logic 7.856ns (73.390%)  route 2.848ns (26.6
  Logic Levels:           9  (CARRY4=4 DSP48E1=2 LUT2=2 LUT4=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.128ns = ( 12.970 - 10.842 )
    Source Clock Delay      (SCD):    2.456ns
    Clock Pessimism Removal (CPR):    0.184ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Simulation Waveform

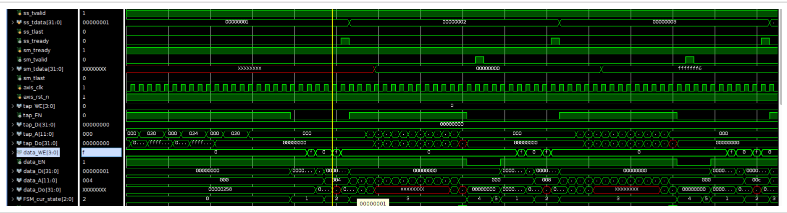
Programming coefficient, and checking

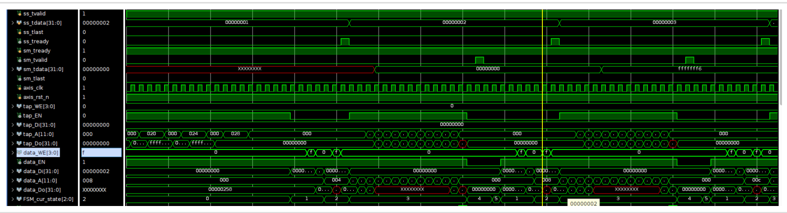
Programming 1st coef.	Checking 1st coef.
	

Programming 2nd coef.	Checking 2nd coef.
	

When `awready` and `awvalid` are sampled, the combinational logic assign the `tap_WE` and `tap_Di` according to the corresponding `awaddr`.
Later, `tap_WE` and `tap_Di` will be sampled at the rising clock edge, and write data into bram.
And the `tap_Do` shows the written data after a clock.

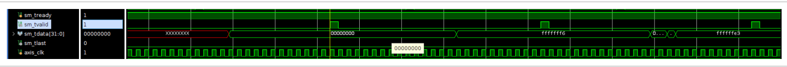
Data-in (stream-in)

Stream in 1st data


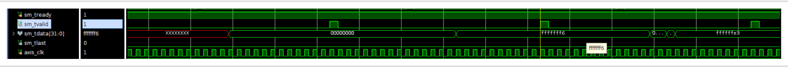
Stream in 2nd data


It first assert `data_WE` until the bram transaction finishing.
And 1 clock later, it assert `ss_tready` to tell the host to transfer next data.

Data-out (stream-out)

Stream out 1st data


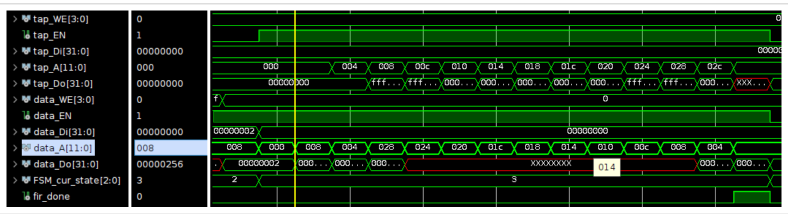
Stream out 2nd data



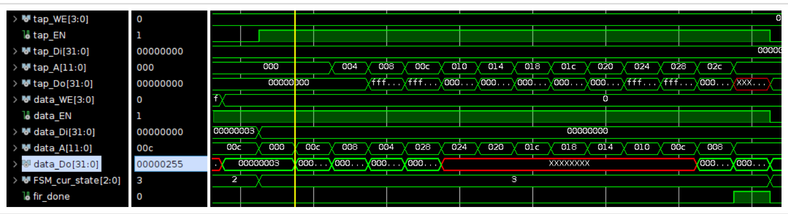
The correct result `sm_tdata` will be sampled by host only when `sm_tvalid` .

Controlling RAM access

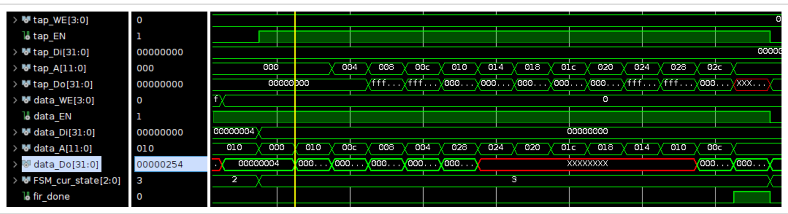
RAM access of 1st fir



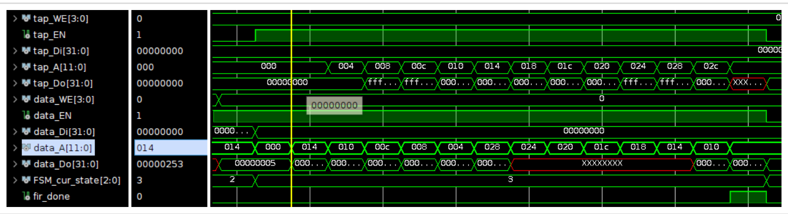
RAM access of 2nd fir



RAM access of 3rd fir



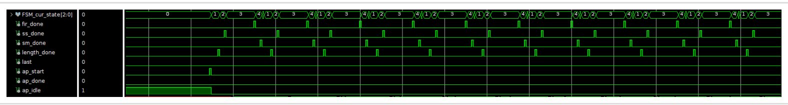
RAM access of 4st fir



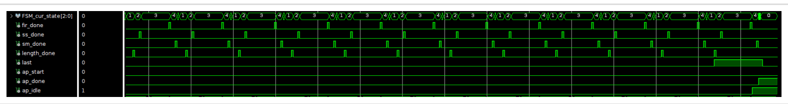
Since we use `bram11` for storing length and shift data, we must maintain the data pointer to read the correct shift data. Which result the first `data_A` address would increase by `0x04` , where the first `tap_A` address is `0x00` , in each iteration of fir computation.

FSM state transition

FSM state after ap_start programmed

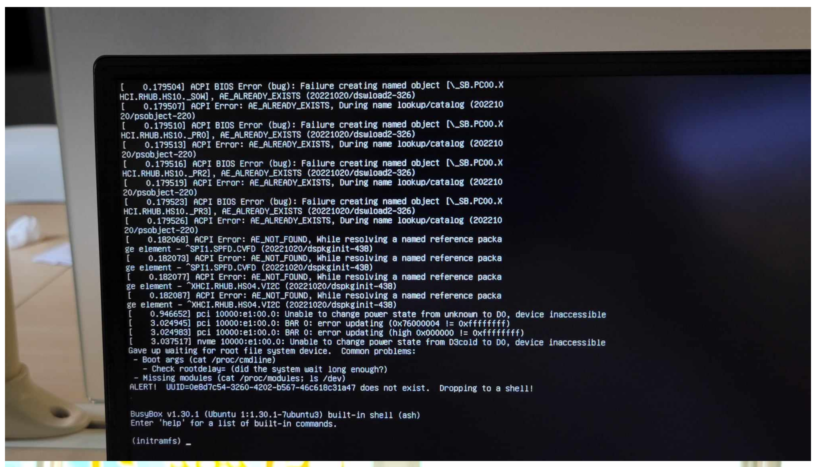


FSM state after ap_start programmed



FSM changes the phase correctly when the `done` signal asserted.

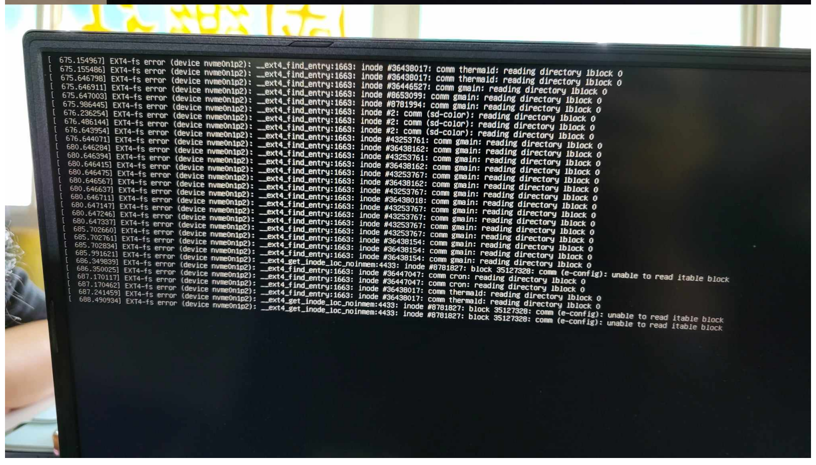
Reason for late submission



```
[    0.179504] ACPI BIOS Error (bug): Failure creating named object [\_SB.PC00.X
HCI.RHUB.HS10._SOW], AE_ALREADY_EXISTS (20221020/dswload2-326)
[    0.179507] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog (202210
20/psobject-220)
[    0.179510] ACPI BIOS Error (bug): Failure creating named object [\_SB.PC00.X
HCI.RHUB.HS10._PR0], AE_ALREADY_EXISTS (20221020/dswload2-326)
[    0.179513] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog (202210
20/psobject-220)
[    0.179516] ACPI BIOS Error (bug): Failure creating named object [\_SB.PC00.X
HCI.RHUB.HS10._PR2], AE_ALREADY_EXISTS (20221020/dswload2-326)
[    0.179519] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog (202210
20/psobject-220)
[    0.179523] ACPI BIOS Error (bug): Failure creating named object [\_SB.PC00.X
HCI.RHUB.HS10._PR3], AE_ALREADY_EXISTS (20221020/dswload2-326)
[    0.179526] ACPI Error: AE_ALREADY_EXISTS, During name lookup/catalog (202210
20/psobject-220)
[    0.182068] ACPI Error: AE_NOT_FOUND, While resolving a named reference packa
ge element - ^SPI1.SPFD.CVFD (20221020/dspkginit-438)
[    0.182073] ACPI Error: AE_NOT_FOUND, While resolving a named reference packa
ge element - ^SPI1.SPFD.CVFD (20221020/dspkginit-438)
[    0.182077] ACPI Error: AE_NOT_FOUND, While resolving a named reference packa
ge element - ^XHCI.RHUB.HS04.VI2C (20221020/dspkginit-438)
[    0.182087] ACPI Error: AE_NOT_FOUND, While resolving a named reference packa
ge element - ^XHCI.RHUB.HS04.VI2C (20221020/dspkginit-438)
[    0.946652] pci 10000:e1:00.0: Unable to change power state from unknown to D0, device inaccessible
[    3.024945] pci 10000:e1:00.0: BAR 0: error updating (0x76000004 != 0xffffffff)
[    3.024983] pci 10000:e1:00.0: BAR 0: error updating (high 0x000000 != 0xffffffff)
[    3.037517] nvme 10000:e1:00.0: Unable to change power state from D3cold to D0, device inaccessible
Gave up waiting for root file system device.  Common problems:
 - Boot args (cat /proc/cmdline)
   - Check rootdelay= (did the system wait long enough?)
 - Missing modules (cat /proc/modules; ls /dev)
ALERT!  UUID=0e8d7c54-3260-4202-b567-46c618c31a47 does not exist.  Dropping to a shell!


BusyBox v1.30.1 (Ubuntu 1:1.30.1-7ubuntu3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs) _
```



```
[  675.154967] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438017: comm thermald: reading directory lblock 0
[  675.155486] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438017: comm thermald: reading directory lblock 0
[  675.646798] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36446527: comm gmain: reading directory lblock 0
[  675.646911] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #8653099: comm gmain: reading directory lblock 0
[  675.647003] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #8781994: comm gmain: reading directory lblock 0
[  675.986445] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #2: comm (sd-color): reading directory lblock 0
[  676.236254] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #2: comm (sd-color): reading directory lblock 0
[  676.486144] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #2: comm (sd-color): reading directory lblock 0
[  676.643954] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253761: comm gmain: reading directory lblock 0
[  676.644071] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438162: comm gmain: reading directory lblock 0
[  680.646284] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253761: comm gmain: reading directory lblock 0
[  680.646394] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438162: comm gmain: reading directory lblock 0
[  680.646415] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  680.646475] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438162: comm gmain: reading directory lblock 0
[  680.646567] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  680.646637] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438018: comm gmain: reading directory lblock 0
[  680.646711] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  680.647147] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  680.647246] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  680.647337] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #43253767: comm gmain: reading directory lblock 0
[  685.702660] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438154: comm gmain: reading directory lblock 0
[  685.702761] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438154: comm gmain: reading directory lblock 0
[  685.702834] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438154: comm gmain: reading directory lblock 0
[  685.991621] EXT4-fs error (device nvme0n1p2): __ext4_get_inode_loc_noinmem:4433: inode #8781827: block 35127328: comm (e-config): unable to read itable block
[  686.349839] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36447047: comm cron: reading directory lblock 0
[  686.350025] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36447047: comm cron: reading directory lblock 0
[  687.170117] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438017: comm thermald: reading directory lblock 0
[  687.170462] EXT4-fs error (device nvme0n1p2): __ext4_find_entry:1663: inode #36438017: comm thermald: reading directory lblock 0
[  687.241459] EXT4-fs error (device nvme0n1p2): __ext4_get_inode_loc_noinmem:4433: inode #8781827: block 35127328: comm (e-config): unable to read itable block
[  688.490934] EXT4-fs error (device nvme0n1p2): __ext4_get_inode_loc_noinmem:4433: inode #8781827: block 35127328: comm (e-config): unable to read itable block
```

Due to the Notebook SSD failure, I have to use the workstation to do the Lab before the SSD got repaired.