November 2021

DUAL POST CODE DESIGN

HP INC

DUAL POST CODE DESIGN

BACKGROUND

Modern PC will disable any debug IO interface when production to avoid hacker attacks. The existing way used for BIOS debugging is using two-byte non-volatile memory to record POST (Power On Self-Test) code for identifying high level BIOS driver error. However, this debug method can't help first triage the issue when bringing up from customer side.

For helping to shorten the debugging period and save the service cost, we introduce the dual post code design idea.

PROBLEM DEFINITION

Under the framework of UEFI Firmware Report Status Code that used for identifying system firmware Process Code/Error Code/Debug Code, we set one variable of BIOS POST CODE for issue analyzing and debugging. But it can only spot which driver was launched, started, and recorded in non-volatile memory. If failed to capture the needed logs at the first place, it's wasting time for reproducing issue.

ABSTRACT

[CONCEPTS]

- Individual BIOS driver owns two variables:
    1. **BIOS Driver POST CODE** –
       unique code to identify driver type (PEI/DXE/SMM) and driver ID.
    2. **Driver Behavior Bit-Map POST CODE** –
       64-bit for recording driver function execution behavior status. This variable length is 64-bit. It can be extended to 128 bits for future.

BIOS Driver POST CODE and Driver Behavior Bit-Map POST CODE will be recorded in non-volatile memory, like CMOS...

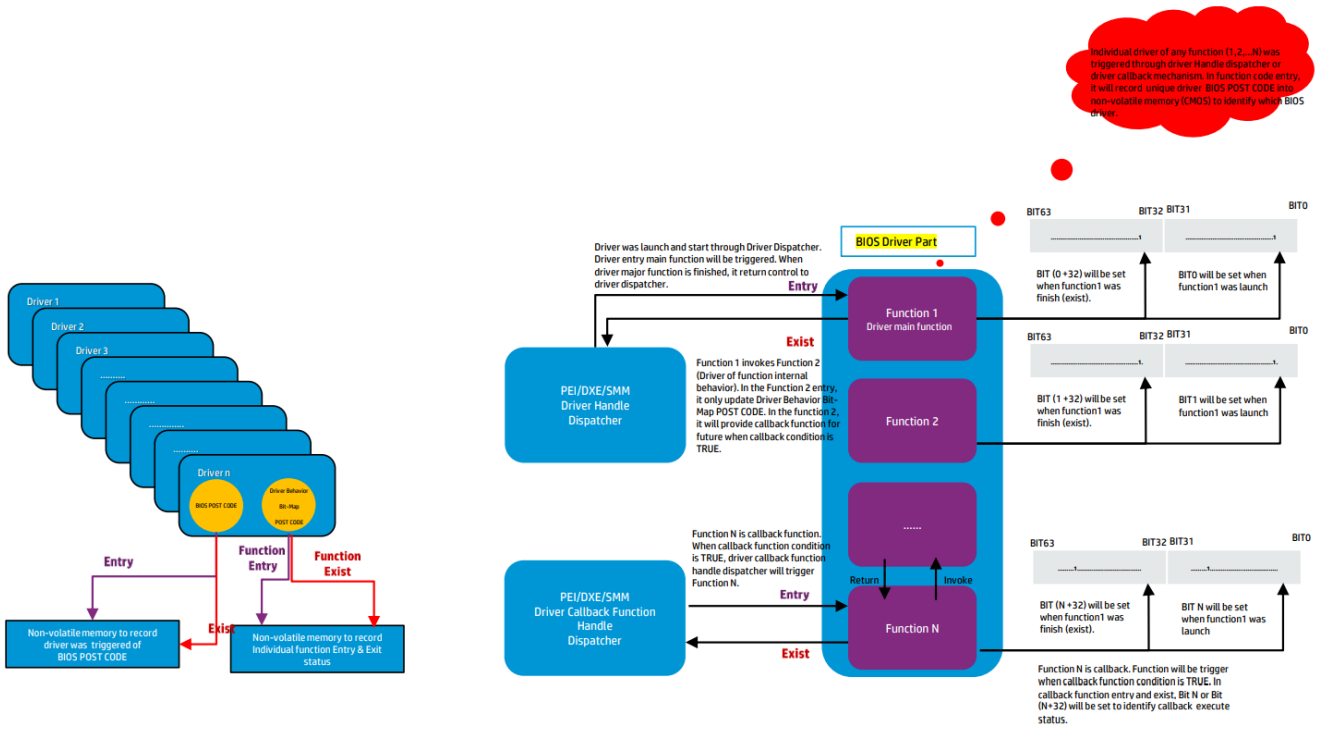
- One new PEI driver, one new DXE driver, and one new SMM driver will provide new interface. This interface can collect the sequence drivers POST CODE behavior and record in volatile memory.
- System BIOS can output drivers POST CODE serial log by specific hotkey and process when system hanging up, and trigger or output drivers POST CODE serial log by control policy when the system happens exception.

[BENEFITS]

- Support directly debugs in production config, no need to rebuild the debug BIOS bootleg.
- Provide serial logging of individual driver POST CODE for analyzing and debugging issue in first triage on customer's end.
- Software solution. No HW cost adder.
- Crossly support on OS ecosystem.

BLOCK DIAGRAM

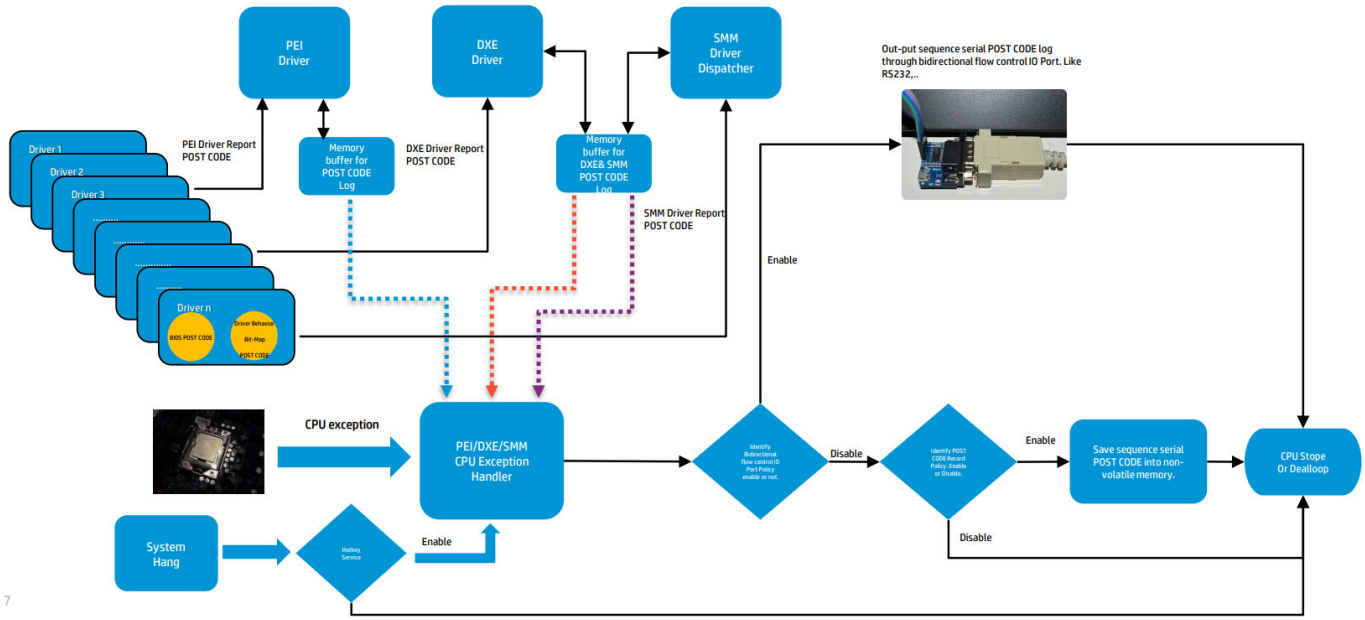
- Individual BIOS driver owns two variables
  - **BIOS Driver POST CODE** (32-bit length) is a fixed value and pre-defined unique value to identify which BIOS driver was triggered. The BIOS POST CODE follows UEFI Status Code architecture to defines BIOS driver of function code entry. The BIOS Driver POST CODE will be set to zero when system firmware triggering exist-boot service callback event and hand over control to OS.
  - New variable of **Driver Behavior Bit-Map POST CODE** (64-bit length) is a dynamic variable to record driver execution behavior (which BIOS driver of function entry and exist status). This variable will be separated to two partitions. The Bit 0 ~ Bit 31 is the individual function entry status. When individual function(0,1,2,…N) entry launching, the Bit x will be set. The rest Bit 32 ~ Bit 63 is individual function existing status. When individual function(0,1,2,…N) completed the existence, the Bit (32+ x) will be set.



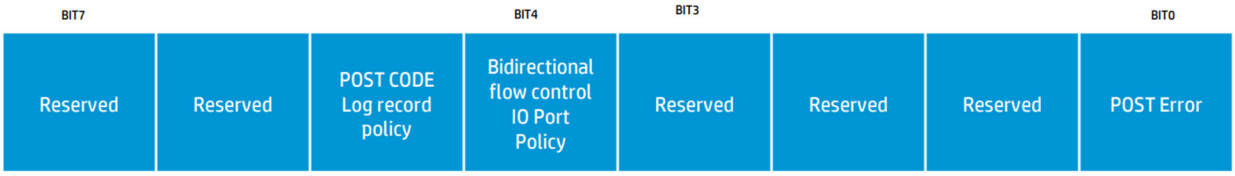
POST CODE LOGGING BLOCK DIAGRAM

WHEN SYSTEM HAPPENS EXCEPTIONS

- New BIOS PEI, DXE, and SMM drivers will provide an interface (function) and allocate memory. The interface is sequence-collected various BIOS Driver POST CODE and Driver Behavior Bit-Map POST CODE through individual BIOS driver actively invoke the interface and report POST CODE.
- In PEI, DXE, SMM phase entry, system BIOS will initiate CPU exception handler. When CPU happens exception, the CPU exception handler will get sequence serial POST CODE logs and output it through bidirectional flow control IO Port. Like RS232... CPU exception handler also save serial POST CODE log into the non-volatile memory.



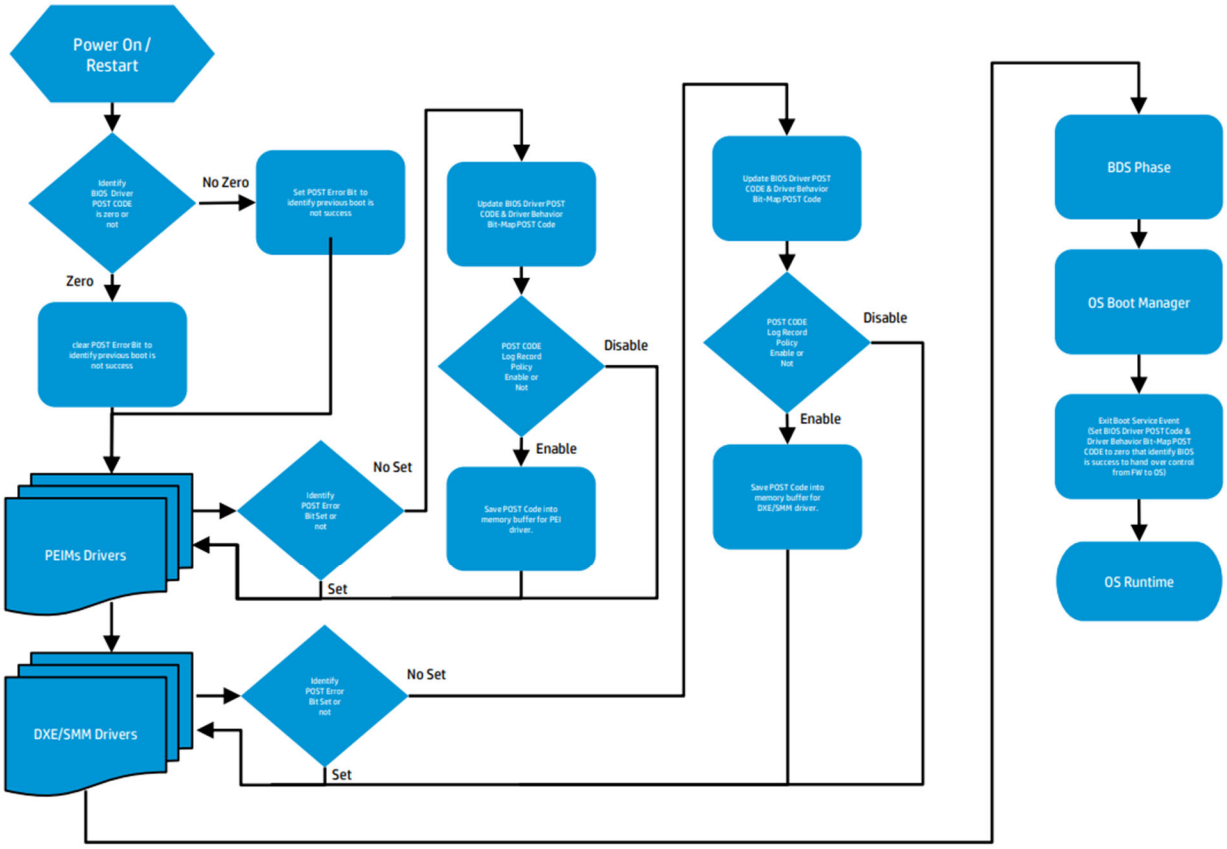
BYTE FORMAT FOR THE CONTROL POLICY & STATUS



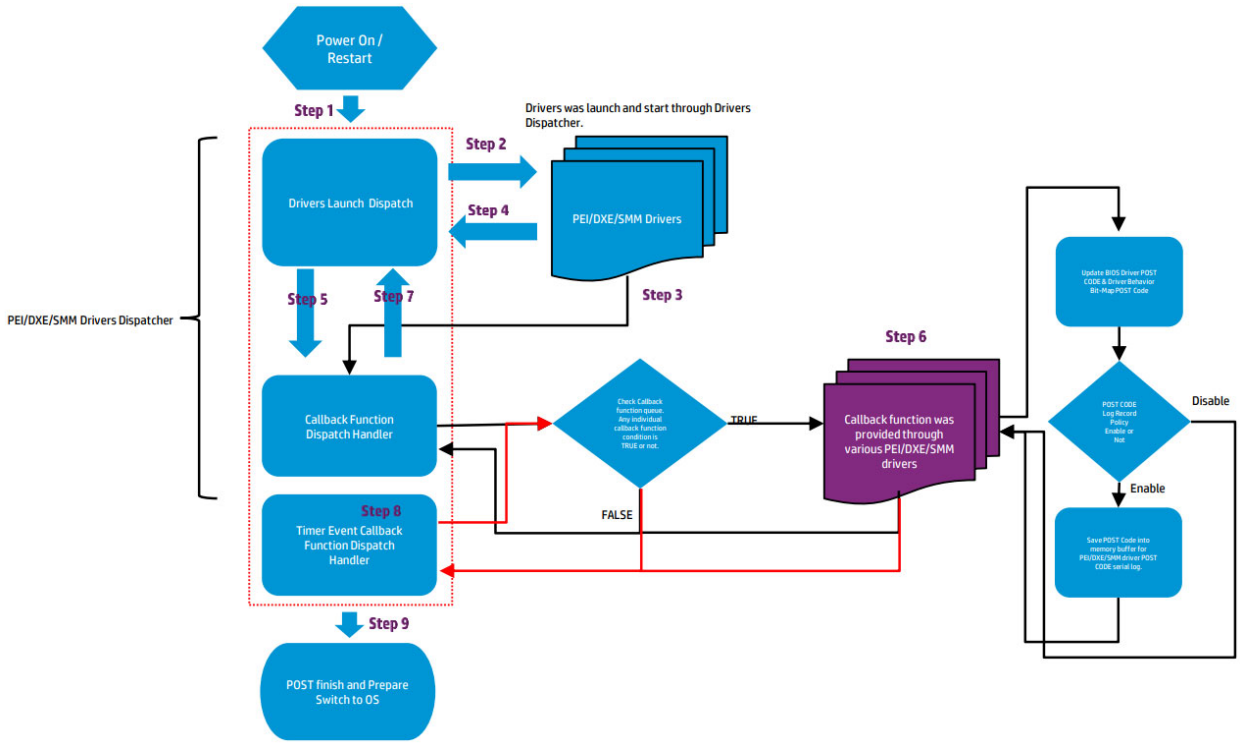
- One byte used for managing Dual POST CODE Control Policy and Status
  - BIT 0 ~ BIT 3: Status
    - BIT 0 - Identify previous POST (Boot) error that system firmware (BIOS) doesn't success to hand over control from BIOS to OS.
  - BIT 4 ~ BIT 8: Control Policy
    - BIT 4 - Bidirectional flow control IO Port policy control (Enable/Disable)
    - BIT 5 – POST CODE Log record policy (Enable/Disable)

NORMAL BOOT FLOW

BIOS Driver POST CODE and Driver Behavior Bit-Map POST CODE Update flow.



PEI/DXE/SMM CALLBACK MECHANISM FLOW



Step 1 - System power on and hand over the control to Drivers Dispatch

Step 2 - Individual drivers will be launched & started through Driver Dispatcher

Step 3 – Individual driver will trigger the callback of Dispatch Handler when PEI driver providing the interface through PeiInstallPpi, DXE driver providing interface through InstallProtocolInterface, DXE driver trigger the event callback function through Signal Event, or DXE driver register the timer event callback function through Set Timer & Create Event.

Step 4 – Individual driver entry function is completed and hand over the control to PEI/DXE/SMM Dispatcher.

Step 5 – Launch the callback function of Dispatch handler to check callback function queue. Any individual callback function conditions are TRUE or not.

Step 6 – When the condition is TRUE, callback function will be executed. And update BIOS driver POST CODE & Driver Behavior Bit-Map POST CODE.

Step 7 - Callback Dispatch handler finished and hand over the control to PEI/DXE/SMM drivers Launch Dispatch.

Step 2 ~ Step 7 is repeated until all PEI/DXE/SMM drivers launch & start.

Step 8 – Timer event call function dispatch to handle period timer event function.

Step 9 – When all PEI/DXE/SMM drivers launched and BDS is done, prepare it and switch to OS.

Disclosed by Tom Hung, Ethan Huang, Nicholas Feng, EK Chiang and Matilda Lin, HP Inc.