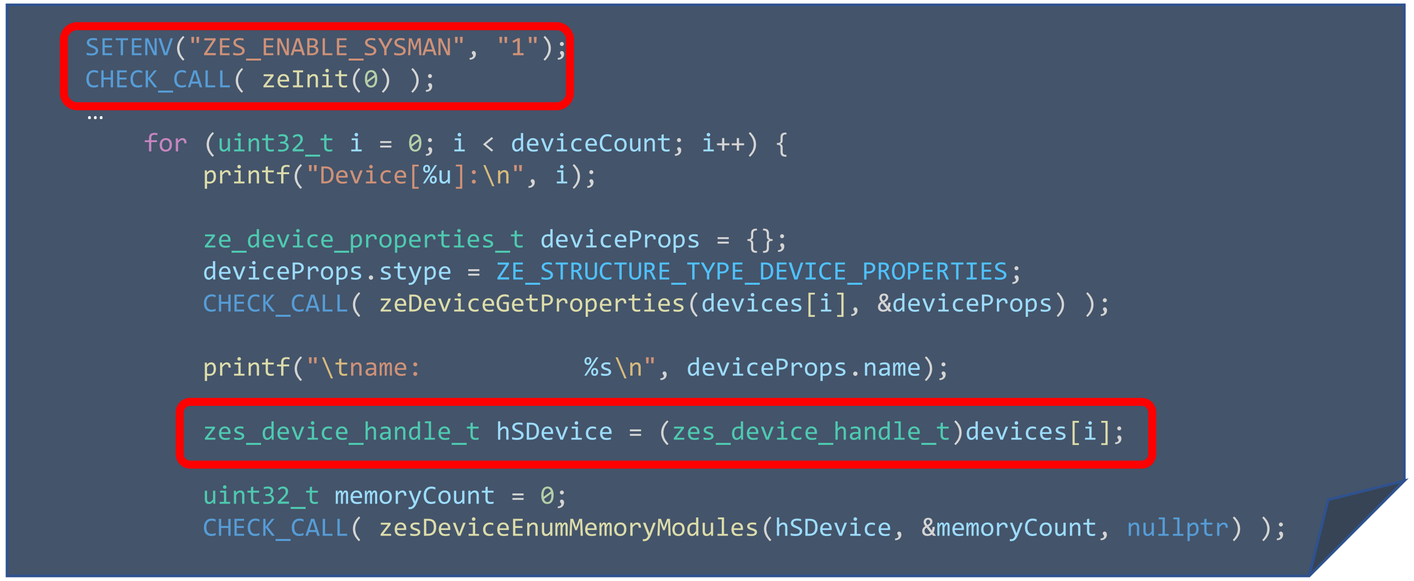
oneAPI™

Discussion Topic: Decoupling Level Zero Sysman from Core APIs

Background

- Currently, Level Zero Sysman is tightly coupled to Level Zero Core APIs
  - Initialized with the same zeInit(), enabled with an environment variable
  - Operate on the same device handles (with a cast)



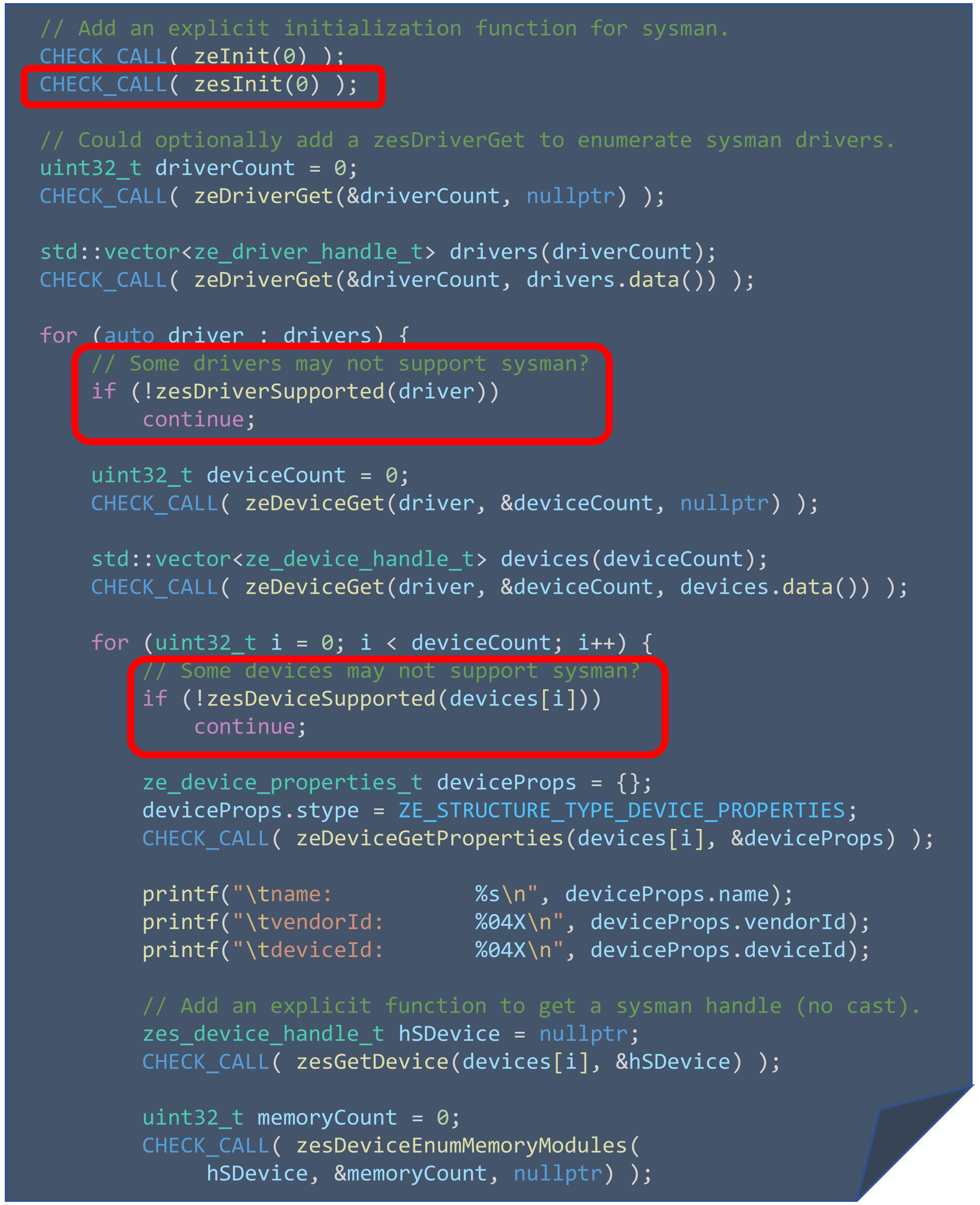
```
SETENV("ZES_ENABLE_SYSMAN", "1");
CHECK_CALL( zeInit(0) );
…
    for (uint32_t i = 0; i < deviceCount; i++) {
        printf("Device[%u]:\n", i);

        ze_device_properties_t deviceProps = {};
        deviceProps.stype = ZE_STRUCTURE_TYPE_DEVICE_PROPERTIES;
        CHECK_CALL( zeDeviceGetProperties(devices[i], &deviceProps) );

        printf("\tname:           %s\n", deviceProps.name);

        zes_device_handle_t hSDevice = (zes_device_handle_t)devices[i];

        uint32_t memoryCount = 0;
        CHECK_CALL( zesDeviceEnumMemoryModules(hSDevice, &memoryCount, nullptr) );
```

Problem Statements

- Unclear who should enable Sysman, and how, and when
  - Need to support enabling Sysman after the Core APIs
  - Need a solution that does not involve an environment variable
- More applications are wanting to access more Sysman properties
  - Don’t want to replicate all of Sysman in the Core APIs
  - Some Sysman applications may not need the Core APIs
- Various GitHub issues:
  - ZES_ENABLE_SYSMAN vs multiple layers using L0 #36
  - Support for building just L0 Sysman support without IGC dependency #529

Proposal

- Add a zesInit() for initializing sysman, independent of zeInit()
- Add query functions for sysman driver and device support
- Add function to get sysman handle from core API handle
- Consider enumerating sysman separately from core APIs?
  - Can use driver and device UUIDs to match across the two?



```
// Add an explicit initialization function for sysman.
CHECK_CALL( zeInit(0) );
CHECK_CALL( zesInit(0) );

// Could optionally add a zesDriverGet to enumerate sysman drivers.
uint32_t driverCount = 0;
CHECK_CALL( zeDriverGet(&driverCount, nullptr) );

std::vector<ze_driver_handle_t> drivers(driverCount);
CHECK_CALL( zeDriverGet(&driverCount, drivers.data()) );

for (auto driver : drivers) {
    // Some drivers may not support sysman?
    if (!zesDriverSupported(driver))
        continue;

    uint32_t deviceCount = 0;
    CHECK_CALL( zeDeviceGet(driver, &deviceCount, nullptr) );

    std::vector<ze_device_handle_t> devices(deviceCount);
    CHECK_CALL( zeDeviceGet(driver, &deviceCount, devices.data()) );

    for (uint32_t i = 0; i < deviceCount; i++) {
        // Some devices may not support sysman?
        if (!zesDeviceSupported(devices[i]))
            continue;

        ze_device_properties_t deviceProps = {};
        deviceProps.stype = ZE_STRUCTURE_TYPE_DEVICE_PROPERTIES;
        CHECK_CALL( zeDeviceGetProperties(devices[i], &deviceProps) );

        printf("\tname:          %s\n", deviceProps.name);
        printf("\tvendorId:      %04X\n", deviceProps.vendorId);
        printf("\tdeviceId:      %04X\n", deviceProps.deviceId);

        // Add an explicit function to get a sysman handle (no cast).
        zes_device_handle_t hSDevice = nullptr;
        CHECK_CALL( zesGetDevice(devices[i], &hSDevice) );

        uint32_t memoryCount = 0;
        CHECK_CALL( zesDeviceEnumMemoryModules(
            hSDevice, &memoryCount, nullptr) );
```

Discussion Questions

- Does the proposal move in the right direction?
  - Would this help your applications or libraries?
- How important is backward compatibility?
  - Can we retrofit into Level Zero 1.x without breaking backward compatibility?
  - What should we do in Level Zero 2.0 if we break backward compatibility?
- Do we need to separate privileged sysman from unprivileged sysman?
- How to handle ZES_ENABLE_SYSMAN_LOW_POWER?
  - Possible argument to zesInit()?
- Do we need to decouple other tools also (e.g., debugging)?

oneAPI™

Thank You!

http://oneapi.com