

execution moves to another virtual machine **240**, **244**. The access trap is reasserted when execution returns to the Dos VM **242** in the windowed mode or upon a switch from the full screen mode to the windowed mode.

Whenever the access trap detects an access attempt, the VDD **50'** is provided the access characterizing information obtained by the trap. Specifically, the board library **74'**, as shown in FIG. **9**, is provided with the access attempt information by the operating system **54** logically through the connection **246**. Preferably, the components of the VDD **50'** implement a single context version of the display driver **50**. However, multiple contexts can be readily supported as appropriate to support multiple controllers **19** under the Dos VM **242**.

Within a single context, the hardware interface objects **130'**, **132'**, **134'**, and **138'** operate to store the access characterizing data so as to maintain a representation of the intended state of the display based on the successive attempts to directly access the hardware. Preferably, the RegClassMap structures associated with the hardware interface objects are augmented with pointers to storage space for the state information associated with each class of hardware sub-elements. The hardware interface objects also provide emulation routines that generate operating system calls with the assistance of the O/S object **128'** to cause the display of a suitable representation of the intended screen appearance. These calls are applied to the operating system layer **54** and, in turn, suitably routed to the display driver **50**. Upon a switch to a full screen mode, the display state maintained by the hardware interface objects can be used directly to establish the intended display state by applying the state data to the register interface **30**.

The parser routine of the device driver **50'** is used effectively in reverse to analyze the access characterizing information. Preferably, the access traps serve to characterize access attempts to device classes implicitly by the assignment of addresses to class trap handlers. Thus, the class, address and value provided with the access are collected by the trap handlers and provided to the reverse parser routine. Thus, with each trapped access attempt, the access related data is stored in the corresponding RegClassMap identified storage. The reverse parser also performs an analysis against the register definitions ultimately determined from the class register instructions stored in the modes.dmm file. The result of the analysis is a logical determination of whether the register intended to be written is an index register, or other management function register, or a data register. Where, the intended register is an index register or other management function register, the resulting change in state is recorded. Where the intended register is a data register, the new state of the register is recorded and then a determination is made as to whether some emulation is required. Depending on the particular register being written, no emulation may be required or the full index and data access operation may then be performed.

Consequently, substantially the same hardware and operating system interface object definitions are preferably used and, further, the same methods of selecting between multiple functions that support differing display characteristics can be used to select among display characteristic emulation routines implemented by the encapsulated hardware interface modules. Where the parser routine detects an identifiable mode set, the shell object **126'** may be called via the O/S object **128'** to perform a mode set operation as previously described. The substantially linear call sequences implemented by the operating system objects **120'**, **126'** are directly enabled. The function call relation between the

operating system interface objects and the remainder of the VDD **50'** is therefore the same as in the case of the device driver **50**.

VIII. Conclusion:

Thus, a highly optimal device driver architecture suitable for supporting a complex and multi-function peripheral controller as well as operating as a virtual device driver has been described. The architecture of the described device driver directly supports dynamic configuration of the device driver at load time to specifically match the hardware configuration of the peripheral controller as preferably determined directly from the hardware on a per-sub-element detailed basis, that employs a modular architecture specifically supporting functional isolation of module changes in correspondence with specific sub-element designs, that provides for an efficient mechanism for performing mode switches of the operating state of the controller, that provides an efficient mechanism for maintenance and management of persistent data independent of mode switches through the support of independent context selectively with the performance of mode switches, and that provides for the efficient management of color depth transformation in video display controller applications. Furthermore, notwithstanding the modular complexity of the architecture, the supported inter-operative relationship between the modules enables substantially linearized call sequences to virtualize and implement the operating system API calls.

In view of the above description of the preferred embodiments of the present invention, many modifications and variations of the disclosed embodiments will be readily appreciated by those of skill in the art. It is therefore to be understood that, within the scope of the appended claims, the invention may be practiced otherwise than as specifically described above.

What is claimed is:

1. A device driver supporting the virtualized execution of applications within a computer system, said device driver comprising:

- a) an interface to an operating system that supports an access trap mechanism permitting said device driver to selectively establish an access trap for accesses by a predetermined application to a predetermined controller coupled to said computer system;
- b) a predetermined set of mode set programs, each including a sequence of instructions defining programming accesses to said predetermined controller to establish a respective one of a like set of operating modes of said predetermined controller; and
- c) a parser routine for analyzing successive trapped access attempts by said predetermined application to said predetermined controller, said parser routine analyzing said successive trapped access attempts against said predetermined set of mode set programs to identify an intended operating mode of said predetermined controller, wherein said predetermined controller includes a plurality of functional sub-elements providing for the implementation of a like plurality of operational aspects of said controller, and wherein said device driver includes a like plurality of interface modules, each of said plurality of interface modules providing for the emulation of a corresponding one of said plurality of operational aspects defining said intended operating mode of said predetermined controller.

2. The device driver of claim **1** wherein said plurality of interface modules further selectively provide for the programming of respective ones of said plurality of functional sub-elements of said predetermined controller.