A HIGH-PERFORMANCE HARDWARE-BASED HIGH-ASSURANCE TRUSTED WINDOWING SYSTEM

Jeremy Epstein
Cordant, Inc.
11400 Commerce Park Drive
Reston VA 22091
jepstein@cordant.com

Abstract1

TRW's Trusted X Window System prototype established that it is possible to build a high assurance windowing system, given a trusted operating system as a base. This paper describes an extension of that architecture that uses custom designed hardware to provide a high-performance, low-cost windowing system while retaining the high-assurance character of the original design.

1. Introduction

The TRW Trusted X Window System (henceforth TX) prototype showed that high assurance multi-level secure windowing is not an oxymoron. [TXArch93] describes the TX architecture. However, TX has a fundamental performance limitation: all screen drawing is performed by updating a "virtual frame buffer", which is then merged by the software TCB into the physical screen. As a result, screen updates are slow. In addition, software is unable to take advantage of any graphics hardware, because hardware access is limited to the TCB.

In this paper we describe the design for a hardware board, which coupled with the TX design can yield a high-performance, high-assurance, low-cost workstation.

Section 2 gives a brief introduction to the TX architecture. Section 3 describes the design for the hardware, and contrasts it with both the software-based solution in TX, and with other hardware-based solutions. Section 4 describes some particular considerations in building the proposed board for IBM PC hardware. Section 5 compares this architecture to related work, while section 6 summarizes our results. Section 7 is a summary of acronyms used in the paper.

2. TX Architecture

The TX architecture, as described in [TXArch93], relies on an underlying high-assurance (e.g., TCSEC B3 or A1 [TCSEC85]) operating system that supports both single-level and multi-level subjects over a range of Mandatory Access Control levels with high-bandwidth inter-subject communication. TX uses the time-tested concept of replication (or polyinstantiation) of subjects to allow untrusted software to provide most of the system's functionality. Figure 1 shows a simplified version of the TX architecture.2 Items shown above and to the left of the double line are non-TCB, while items shown below and to the right of the double line are TCB elements.

Keyboard and mouse input is received by TX/IM, and passed to the X single level server3 (TX/SLS) corresponding to the currently selected MAC label. That is, if the currently selected label is Secret/B/, then all input would be passed to the X server on the left in Figure 1, and the X server on the right would be unable to detect the presence of any input. The only processing performed by TX/IM is to search for the Secure Attention Key (SAK) sequence, which is used to invoke the trusted path facility.

An unmodified X client makes requests to the TX/SLS running at its same MAC label, which in turn draws into a single level *virtual frame buffer* (VFB). Each VFB represents what a single TX/SLS views as the contents of the screen. When a TX/SLS updates its VFB, it notifies TX/DM, which merges the VFBs for all TX/SLSs and updates the *physical frame buffer*

2The following items are omitted: TX/PE and TX/SEs (used for cut and paste support); TX/CIT, TX/SIT, and TX/M (used for initialization).

3In X, the term *server* refers to the software that manages the graphics hardware, while a *client* is an application that uses graphics hardware. Thus, an X server is always local to a user, while a client may be local or remote. While consistent, this nomenclature frequently confuses new users of X, who are used to clients being local and servers being remote!

(PFB), which is used by the graphics hardware to update the screen. Access to the PFB is limited to TX/DM, along with access to all other graphics hardware (because the graphics hardware is not trusted to provide separation of requests from multiple MAC labels).

Users can manipulate only those windows at the current selected MAC label. That is, to move, resize, or provide keyboard or mouse input to a window, the current MAC label must equal the window's MAC label. This is enforced by TX/IM, which sends all input to the TX/SLS corresponding to the currently

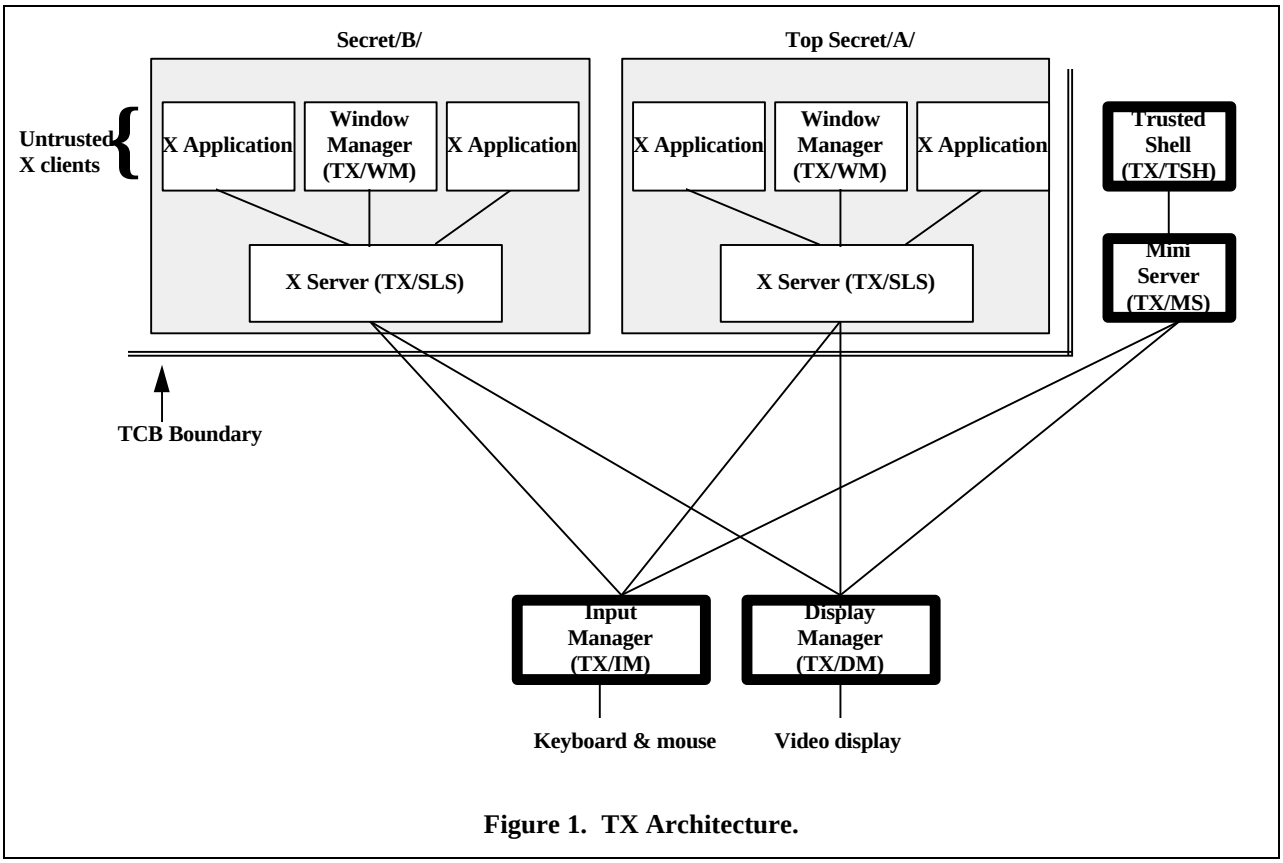

Figure 1. TX Architecture.

When an application action causes mapping or unmapping of a window4 on the screen (typically associated with starting a new application), the corresponding TX/SLS sends a message to TX/DM, which in turn recalculates the screen layout based on the new set of windows on the screen. When mapping windows, TX/DM also draws visible labels representing the MAC label on all four sides of the window.

Each TX/SLS has a matching TX/WM that performs (untrusted) window management of windows at that MAC label.

4Strictly speaking, this process only occurs with top level windows, as windows are defined in a hierarchical fashion in X (i.e., each push-button is a window within a pane of push-buttons, which is a window within a dialog box, which is a window within a top-level window).

selected label.

TX/MS performs an analogous function to the TX/SLS for the trusted shell application (TX/TSH). That is, it allows TX/TSH to draw on the screen using a small set of graphics primitives. TX/MS draws in a VFB, just as the TX/SLSs do. TX/DM merges the VFB belonging to TX/MS with the TX/SLS's VFBs, although TX/MS always takes precedence. Functions provided through TX/TSH include changing the current MAC label and starting instances of TX/SLS at new MAC labels. TX/MS and TX/TSH are inactive, except when invoked by the user through the SAK (as described above under TX/IM).

Figure 2 shows a sample window display, with shading to indicate which portion of the TX system provides the information on the screen. The background portion of the screen contains a fill pattern that is selected by the lowest TX/SLS (i.e., the least highly classified) associated with the login session.

Note that there is a complete set of processing software (i.e., TX/SLS, TX/WM, and applications) with a corresponding VFB for each unique MAC label in use. Thus, if a user is concurrently working with data at four different classification levels, there would be four X servers, four window managers, and four sets of applications running. Each unique combination of non-hierarchical categories is considered a different MAC label for purposes of TX software replication.

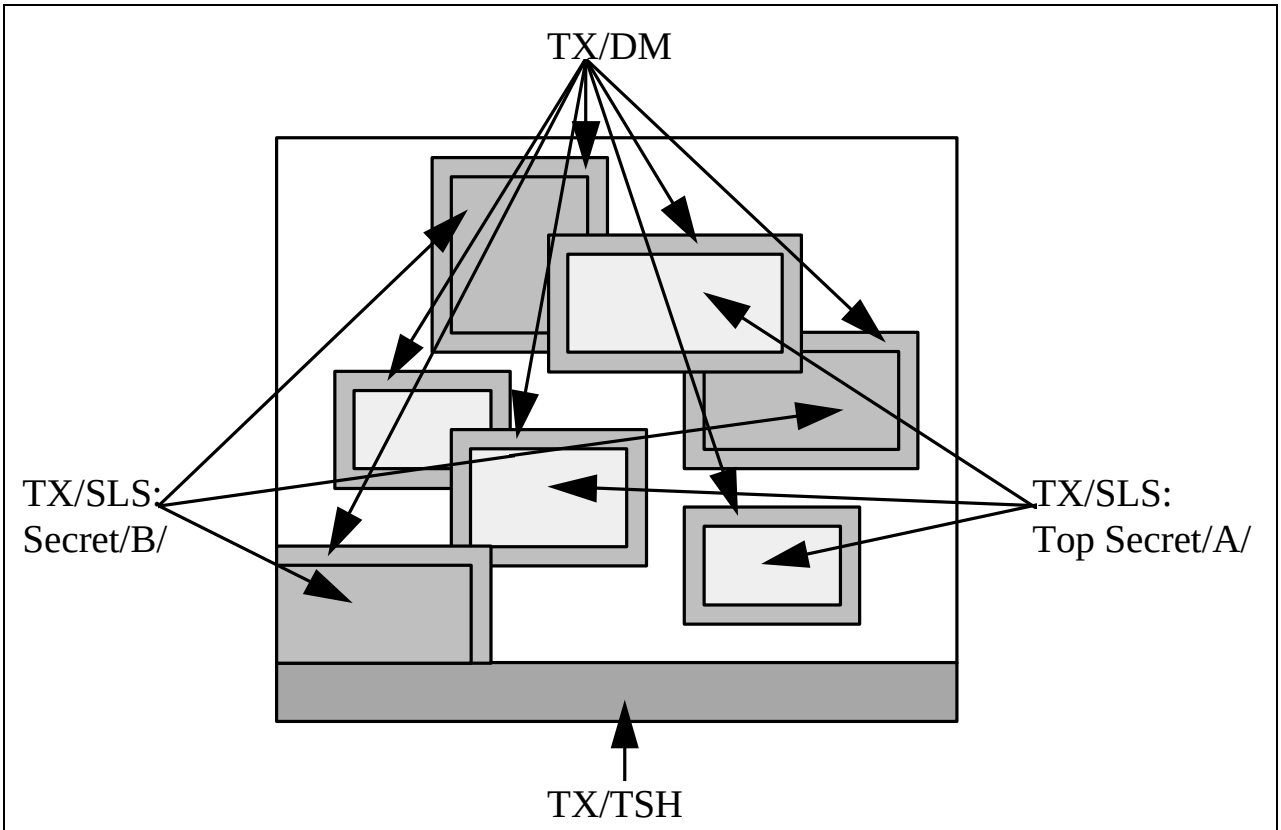

Figure 2. Sample TX Screen.

3. Hardware Design

There are several performance problems with the TX architecture relating to output processing due to the extra level of processing involved with each change to the display. While input processing theoretically has the same performance problem (due to TX/IM having to examine each keystroke for the SAK), human input rates are low enough that the additional overhead is not noticeable[5]. The remainder of this section describes the processing problems, provides a brief introduction to graphics hardware, and proposes a solution based on hardware polyinstantiation.

3.1. The Problem

Consider how a character typed by a user is echoed to the screen in an ordinary X system as compared to TX.[6] In ordinary X, the X server receives the character and routes it to the appropriate X client. That client responds by instructing the X server to draw the character in the window. The X server verifies that the portion of the window being drawn is not obscured by another window (or off the screen, or

---

[5] While the input processing speed was not noticable, the resources required to echo characters to the screen was quite noticable in the prototype implementation.

[6] Character processing in X is full-duplex: the X server does not provide echo, but rather relies on the X client to perform the echo. This is necessary because only the X client knows where to draw the character, what font to use, etc.

otherwise unavailable), and renders the character into the PFB, possibly using a hardware assist (i.e., hardware capable of rendering characters from a font stored memory). By contrast in TX, TX/IM receives the character, routes it to the X server (i.e., TX/SLS) at the currently select MAC label, which routes the character to the appropriate X client. Again, the client responds to the X server (i.e., TX/SLS), which performs the same processing, but draws the character into its VFB *without* any hardware assist. TX/DM then verifies that the portion of the VFB being updated is not obscured by a window at a different MAC label (a given TX/SLS is unable to determine this, since it only knows about windows at its own MAC label), and if no such limitation exists, copies the character from the VFB of the TX/SLS to the PFB, thus causing its display on the screen.

In a more extreme case, consider where an action would cause an X client to perform a three-dimensional rotation of an image in a screen, or where video is being shown in a window. In ordinary X, the client can request that the X server perform a 3D rotation (providing that the server provides that facility), which can be done in hardware. Similarly, a client can request that the server show a video in a window (again, providing that the server provides the facility), which may be possible in hardware. In TX, these features cannot use hardware, because the TX/SLS has no direct hardware access. Hence, for such real-time and high-performance graphics, TX is too slow to be usable.

Thus, we note that a system using TX as its multi-level secure windowing system will be unable to take advantage of high performance graphics hardware, and will hence be limited in its graphics performance. In addition, the context switching and message passing time in the underlying operating system becomes critical to the performance of the user interface.

3.2. A Brief Introduction to Graphics Hardware

To understand the proposed solution, it is necessary to have a high-level view of how graphics hardware works. As previously noted, ordinary X servers draw into a PFB. A PFB is a one-dimensional array, where each element of the array represents a single pixel on the screen. Each element of the PFB is one or more bits, depending on the type of video being drawn and the cost. For example, a black and white monitor would be driven by a PFB with one bit per pixel (where the definition of whether 0 represents black or white is dependent on the hardware designer). For color or grayscale graphics controllers, common values are four bits per pixel (16 simultaneous color or grayscale values possible), eight bits per pixel (256 simultaneous color or grayscale values possible), and 24 bits per pixel (16M simultaneous color or grayscale values possible).

It is thus possible to calculate the memory requirements for a PFB by multiplying the resolution to be provided (e.g., 1024 x 768) by the number of bits per pixel. Graphics controllers for current model IBM PCs typically have 1MB, which allows for 1024 x 768 with eight bits per pixel.

For four or eight bit controllers, the pixel value is not usually a color definition *per se*, but rather an index into a *colormap* which selects the red, green, and blue values associated with pixels having that value. Thus, it is possible to recolor all pixels of a single value without modifying the pixels, but rather by modifying the colormap entry. Depending on the graphics hardware, the colormap may be managed directly by the X server or by the operating system. If the operating system manages the colormap, then the X server uses system calls to request changes to the map. For 24 bit controllers, the pixel value typically contains eight bit red, green, and blue values, and hence no colormap is needed.

For maximum performance, X servers map the PFB into their address space and manipulate the PFB directly using ordinary memory load and store instructions. That is, the PFB is not managed by the operating system's kernel, as the overhead in making operating system requests to modify each pixel would render the hardware unusable. Low-end graphics hardware converts the values in the PFB together with the colormap into electrical signals to the monitor. In this case, the X server translates high-level requests (e.g., draw a three-pixel wide line from pixel (X1,Y1) to (X2, Y2)) into modifications to pixels within the PFB. More sophisticated graphics hardware may include facilities to offload such drawing, so the host processor can spend more cycles running the application itself. High-end graphics hardware can include a buffer of related commands, and can then perform tasks such as rotation without any involvement by the X server.

In X, clients do not directly manipulate either the PFB or the colormap, but rather rely on the X server to perform those tasks.

3.3. Proposed Solution

Our goal is to reduce the performance bottleneck caused by TX/DM having to mediate all access to the PFB. To do so, we propose a hardware solution with

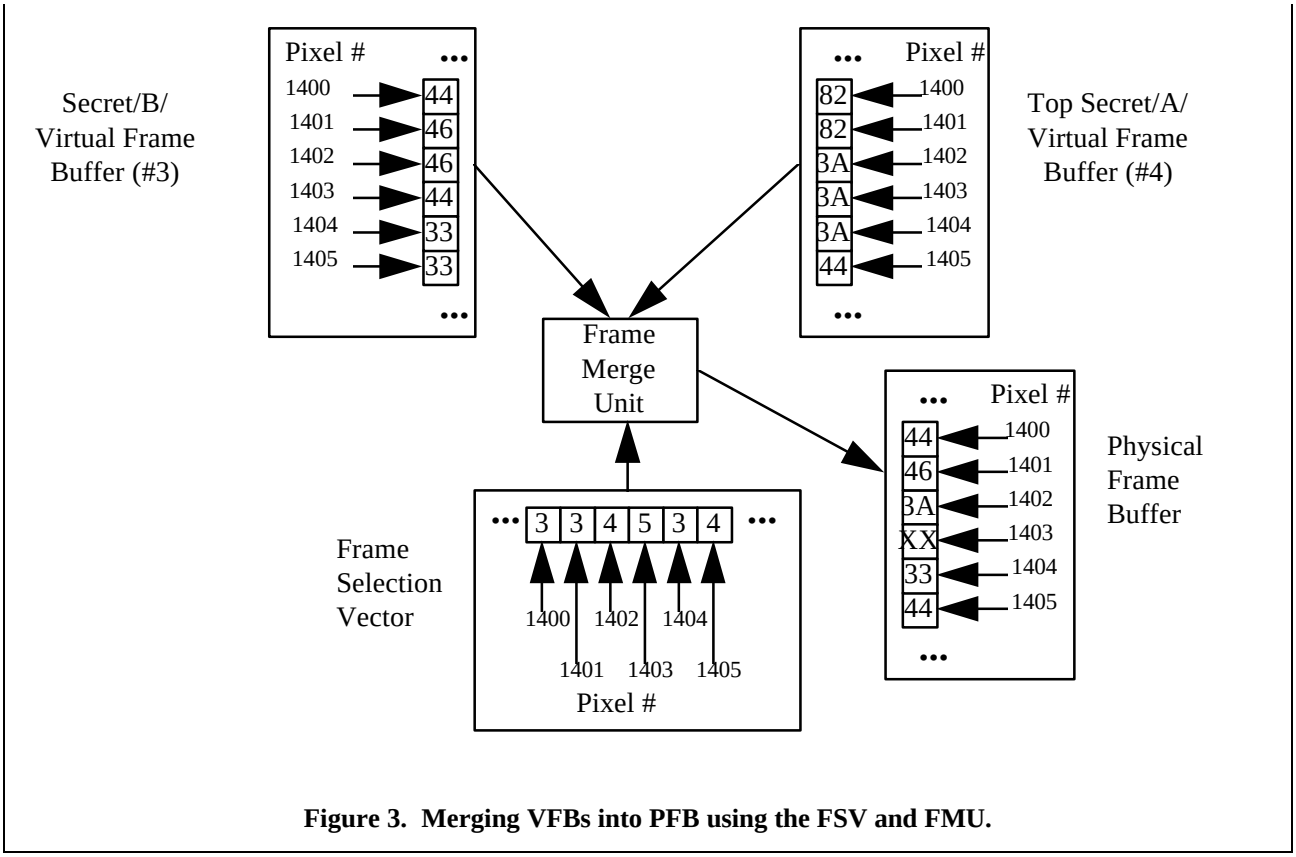

Figure 3. Merging VFBs into PFB using the FSV and FMU.

multiple *Virtual Graphics Subsystems* (VGS),7 each with its own VFB. Each TX/SLS would then have direct access to a VGS, and hence would not need to request that TX/DM perform screen updates. A *Frame Selection Vector* (FSV) determines, for each pixel on the screen, which of the VFBs "owns" that pixel. The *Frame Merge Unit* (FMU) constantly scans the FSV, selecting the pixel from the corresponding VFB and copying it to the PFB. Existing hardware can then translate the PFB into signals to the monitor, just as this operation occurs in current graphics controllers. Thus, if an SLS updates its VFB, the contents of the frame buffer would be updated at the next scan of that pixel (which typically happens 60 or 70 times per second).

Figure 3 shows that for pixel 1400, the FMU selects the value 44, because the FSV entry for pixel 1400 is 3, indicating the Secret/B/ VFB, while for pixel 1402 the FMU selects the value 3A, because the FSV entry for that pixel is 4. The value XX is shown for pixel 1403 to indicate that the value of VFB #5 is not shown in the figure.

In this architecture, VGSs and their corresponding VFBs are dynamically allocated when users request creation of new X servers using the TX/TSH. To avoid improper object reuse, there must be a mechanism in the hardware to cause resetting a particular VGS and clearing the associated VFB. The zeroeth VGS is reserved for use by TX/DM and TX/MS for displaying TCB data, such as the trusted path menus and visual window labels.

Figure 4 shows the mapping of VGSs, VFBs, the FSV, and FMU. TX/DM still plays an important role in display: it is responsible for (a) drawing labels in windows by rendering the labels into VFB #0 and (b) updating the FSV whenever a window is mapped or unmapped to assign the necessary pixels to the corresponding VGS and VFB. However, these are both events that occur relatively infrequently compared to screen updates. Hence, the performance-critical portion of TX/DM is moved into hardware.

Note that the size of an entry in the FSV need not be the same as the size of a pixel. The FSV entries would typically be four bits, thus allowing up to 15 VGSs plus the reserved VGS for use with TCB data. A VFB entry would typically be eight or 24 bits, as noted above. Note that all VFBs (and the PFB) must be identical in size for this scheme to work, as the FMU does not map different size VFB entries into the PFB.

7Each graphics subsystem would typically consist of a single graphics chip. More sophisticated graphics subsystems might use several chips, but that has no impact on our architecture.

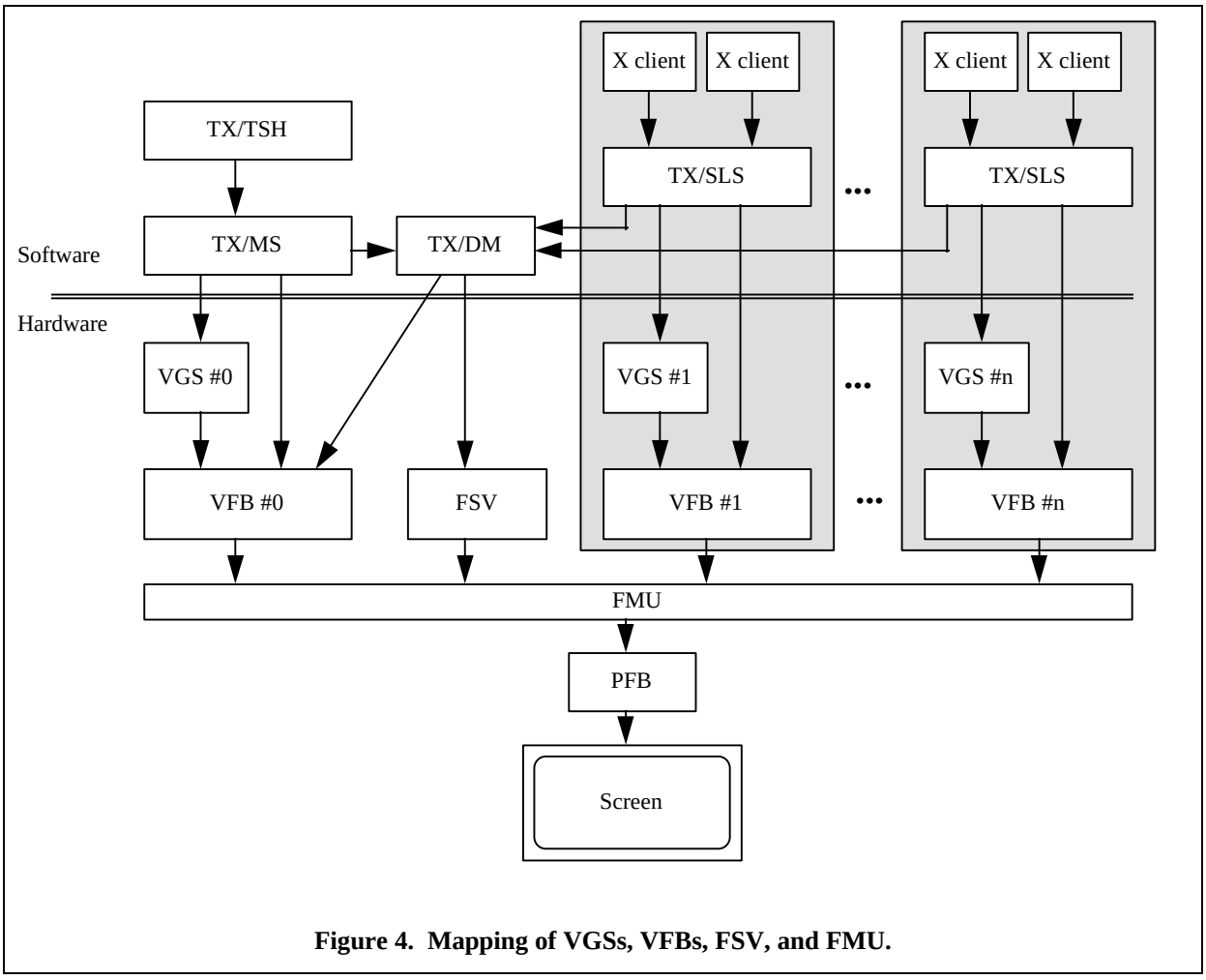

Figure 4. Mapping of VGSs, VFBs, FSV, and FMU.

As previously noted, the pixel value in the VFBs and PFB is (typically) an index into a colormap. For example, in Figure 3, the value 44 appears as a pixel value in both VFB #3 (labeled Secret/B/) and VFB #4 (labeled Top Secret /A/). In the TX prototype each TX/SLS configures its colormap independently to avoid the covert channels which might be present if they shared a common colormap. As a result, whenever the user selects a new current MAC label, the colormap from that TX/SLS is installed as the current colormap. The result of this design is that windows change colors in a distracting fashion when the user changes MAC labels. An alternate solution which could have been implemented in TX is to divide up the colormap, allocating certain entries to each TX/SLS in a static fashion. However, this reduces the maximum number of entries allocated to any individual TX/SLS.

In the proposed hardware architecture, if the FMU directly drives the screen, instead of generating a PFB, then it could take the colormaps corresponding to each VFB and directly generate the necessary signals to the monitor. Alternately, there could be a system colormap for use by the FMU. TX/DM would set the system colormap from the VGS's colormap whenever a MAC label is selected. This latter approach is the equivalent of the prototype TX implementation.

An important premise is that each TX/SLS is capable of communicating with the VGS and VFB at its MAC label, and that there is no mixing of information. The specifics of how this can be implemented depend on both the operating system and specific hardware architecture, and as such are discussed in the following section.

4. A PC Realization

The IBM-compatible personal computer has become the *de facto* standard for workstation hardware. Unfortunately, it has several significant flaws which make an implementation of the above architecture difficult. In this section we outline how the hardware described in section 3 could be implemented in a PC hardware environment. The PC

is probably the most complicated environment to design for; other hardware architectures would be easier than that described here.

shows a block diagram of a PC graphics card designed to use off-the-shelf VGA chips and memory. A *Bus Decode Unit* (BDU) decodes a 16 bit I/O address,

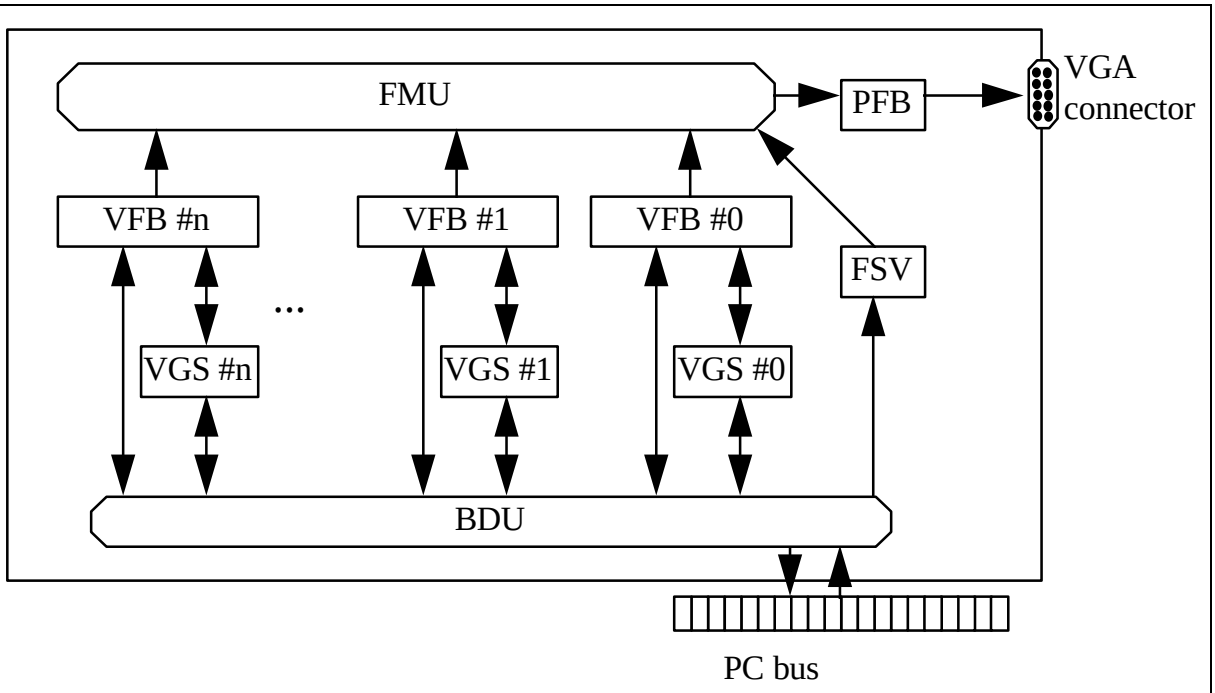

Figure 5. A PC Card Implementing the Proposed Design.

Intel x86 processors (including the 80486 and Pentium) access I/O devices both by issuing IN and OUT instructions and by accessing memory-mapped devices. For example, typical VGA8 controllers decode I/O addresses 3C0h through 3CFh for use in a wide variety of operations including setting up colormaps, configuring screen resolution, etc. The PFB for a VGA controller is accessed through a 64K sliding window mapped into main memory.9

In order to achieve reasonable performance and minimize cost of the resulting board, it is necessary to have hardware that operates within these limits (i.e., it behaves as a conventional VGA controller). Figure 5 examines the high order six bits, and routes it to the appropriate VGS. The BDU also decodes memory addresses to allow access to the VFBs and the FSV. The BDU relies on the host processor to prevent any invalid access, such as by a TX/SLS to the FSV.

In this design, the VGSs are off-the-shelf SVGA chips, and the VFBs are off-the-shelf memory chips. The PFB is a standard video RAM, and the FSV is an off-the-shelf memory chip. Off-the-shelf components can also be used to translate the PFB to video signals. The only custom hardware is the FMU and the BDU. From our experience in designing other PC hardware, we believe that the FMU and BDU could be combined into a single custom chip (most likely an ASIC, or Application Specific Integrated Circuit). The BDU is moderately complex, as it needs to forward requests from the bus to the appropriate VGS and send responses back, mapping I/O addresses in the process.

Operating system support necessary for this card is as follows: Each TX/SLS would need to have I/O access allowed to the I/O addresses of the corresponding VGS,10 and its memory map would

8VGA, or Video Graphics Array, and its superset Super VGA (or SVGA) is the standard for modern PC video controllers. There are a wide variety of largely incompatible SVGA controllers, each of which use a separate command set.

9The 64K window size is a holdover from the early PC architecture, where addresses above 640K were reserved. To move the sliding window, X servers issue OUT commands to the I/O ports, causing the VGA controller to adjust the video addresses mapped into main memory.

10It is undesirable to require that I/O access to the VGS go through the operating system, as the

need to include the corresponding VFB. Fortunately, the Intel 80x86 architecture includes a facility to allow non-ring 0 processes (i.e., those processes not running in the most privileged processor state) to access selected I/O addresses. Thus, the operating system would simply configure the appropriate set of VGS addresses for each TX/SLS, and the CPU hardware performs the necessary protection.

Changes to the X server would be minimal to use this hardware. TX/DM would need a mechanism to notify the TX/SLS of the I/O and memory addresses it should use, rather than the default values hardwired into the code.

4.1. Areas for Future Work

Because we have not performed a detailed hardware design, it is likely that there are flaws yet to be discovered. One item we do not yet have a solution for is how to reliably clear the VGS so it can be dynamically assigned to a new TX/SLS without fear of object reuse. A fallback position would be to either statically assign the VGSs (i.e., VGS #1 is always Top Secret/A/, even if a particular user is not using that MAC label), or to require that the workstation be power-cycled before reassigning VGSs to new labels. Another alternative would be to rely on a hardware feature of the VGSs to perform the clearing on command from the BDU. However, we are unsure whether off-the-shelf VGA chips that would be used for VGSs would have such a facility.

This design allows untrusted software to directly access the bus (to access the VGS and VFB). As a result, it clearly has opportunities for hardware level covert timing channels. We do not have any solution to this problem.

4.2. Estimated Manufacturing Cost

It is not obvious how many VGSs and VFBs a user would need, as different users will need to operate with varying numbers of simultaneous MAC labels. In addition, if VGSs are statically assigned (as noted above might be desirable to avoid covert channels), then more VGSs might be required.

As a result of this uncertainty, we believe that the board described above should be built with three VGSs, three VFBs, and sockets to insert additional VGSs and VFBs. That is, the base model would allow operation of two simultaneous labels, plus a VGS and VFB for use by the TCB for labeling and trusted path.

The actual cost of a board depends on the size of each VFB, the sophistication of the VGSs, and other factors to be determined during detailed hardware design. We have presumed a low-end VGS and 1MB VFBs. We have also assumed that the screen has no more than 1 million pixels, which requires a FSV of 512KB.

Given such assumptions, we believe that such a board could be manufactured today in large quantities (10,000 units) for about $300 each, as shown in Table 1. Fluctuations in memory cost will obviously affect the price significantly. Adding additional VGS/VFB pairs would cost about $65 each. Thus, a full board with 16 VGSs and VFBs could be manufactured for $300+($65×13)=$1145. Note that these figures do not include any allowance for hardware engineering, software development, or profit. While this is certainly not a low-cost board compared to a standard VGA card, it is truly inexpensive compared to having 15 computers on a user's desk!

Table 1. Estimated board manufacturing cost.

Item	Cost
Printed circuit board	$35
BDU/FMU ASIC	$20
Physical Frame Buffer (1MB video RAM)	$40
Virtual Graphics Subsystem (qty 3)	$75
Virtual Frame Buffer (1MB RAM, qty 3)	$120
Sockets for 13 more VGSs and VFBs	$5
Miscellaneous components	$5
Total (with 3 VGS/VFB pairs)	$300
Total (with 16 VGS/VFB pairs)	$1145

4.3. Performance

The design proposed here is such that graphics performance should be almost equal to that of X using a VGA card with the same graphics chip as is used in the VGS. The FMU should not introduce any noticeable overhead. The BDU should not introduce significant overhead either, except when multiple instances of TX/SLS are contending for bus access to access the BDU. Using a fast bus (e.g., a PCI bus) should minimize such contention.

Operating system overhead is a significant concern, as context switching and message passing times can cause software bottlenecks. However, the performance of this design should be significantly better than that of the TX prototype which sends the VFB from the TX/SLS to the TX/DM in a message,

---

aforementioned sliding window is manipulated by performing direct I/O operations to the VGS. Requiring each such operation to go through the operating system would seriously damage performance.

which is a significant strain on operating system message passing.

5. Related Work

The closest work to that described here is a patent granted to Loral [Loral91]. The notion of using polyinstantiated hardware is common to the approach presented there. However, Loral did not have any concept similar to the FSV or the FMU. Hence, the screen was divided into vertical bands, and windows of a given label were confined to a single band. By dividing the screen up, the screen and the system are much less usable than in the approach described here.

Compartmented Mode Workstations (CMWs) [CMSREQS87] provide similar functionality to TX at a lower cost. Because of their lower assurance, the X server is included in the TCB, and hence is able to take advantage of existing graphics hardware. Thus, they have no need for special purpose hardware as is proposed here.

The Secure Computing Corporation is conducting research on a TCB that supports policies in which the VGS memory regions have non tranquil security attributes. With such a TCB, applications in which the required number of separate displays grows to exceed the hardware limited number of VGSs can still be supported with minimal impact on performance. The TCB makes it possible to have displays that are in a "hot backup" state which can be displayed very quickly as needed. The integrity of the separation between the different X servers that use a single VGS in sequential order is assured by the TCB's enforcement of the security labels on the VGS's memory region and the separation capabilities of the system's security policy. Control over the transition between one of the hot backups and a currently active X server is done by a very simple display controller subject that makes use of existing TCB control facilities to change the security label on a VGS. The operational view is similar to the existing multiple display X Window managers. Each display screen would have windows associated with subjects at different security levels, but different display screens could have different groups of applications. Cut and paste between windows and across screens is supported and controlled by the system security policy.

6. Conclusions

We believe that the design presented here is feasible for a low cost, yet high assurance windowing system. By leveraging the existing TX research performed by TRW, a board could be manufactured for as little as $300 that would allow a high assurance operating system to incorporate high performance windowing.

Such a board could be used for other purposes as well, unrelated to high assurance computing, as follows:

(1) If the screen can be reasonably partitioned, the board could be used to provide parallel video processing: multiple applications could simultaneously use the graphics hardware for high performance display. Thus, the board could be considered as a MIMD (Multiple Instruction Multiple Datastream) parallel processing graphics engine.

(2) The board could be used to provide fast switching between multiple desktops. A user might have a desktop for software development and a separate desktop for documentation writing, each of which runs different applications. Switching among the desktops does not require the applications to redraw their windows, but only requires updating the FSV.

(3) The board could be used as a development platform for new windowing systems: a user could run a windowing system for development using one VGS (and associated VFB) and use a separate VGS and VFB as a testbed without fear that a bug in the test windowing system would crash the development environment.

While (2) and (3) are feasible using the TRW TX prototype (as described in [VWS92]), (1) is only possible using the hardware solution proposed here.

7. Acronyms

BDU	Bus Decode Unit. The hardware logic for decoding bus operations and passing them to the appropriate VFB, FSV, or VGS, and for placing replies from the VFB, FSV, or VGS back on the bus.
FMU	Frame Merge Unit. The hardware logic to merge the VFBs into the PFB by selecting pixels based on corresponding values in the FSV.
FSV	Frame Selection Vector. A memory buffer used to select each pixel from the appropriate VFB.
PFB	Physical Frame Buffer. The binary image of the physical screen.
RAM	Random Access Memory.
TX/DM	Display Manager. The portion of TX responsible for managing the

	display and rendering window labels.
TX/IM	Input Manager. The portion of TX responsible for managing keyboard and mouse input.
TX/MS	Mini Server. The portion of TX responsible for rendering the trusted path display.
TX/SLS	Single Level Server. An untrusted X server running at a single MAC label. For every TX/SLS, there is exactly one VGS, one VFB, and one TX/WM.
TX/TSH	Trusted Shell. The portion of TX responsible for the trusted path user interface.
TX/WM	Window Manager. An untrusted X window manager running at a single MAC label. For every TX/WM, there is exactly one VGS, one VFB, and one TX/SLS.
VFB	Virtual Frame Buffer. The binary image of the screen associated with a TX/SLS or TX/MS. For every VFB, there is exactly one VGS, one TX/SLS, and one TX/WM.
VGA/SVGA	Video Graphics Array/Super Video Graphics Array. The standard for IBM PC graphics hardware.
VGS	Virtual Graphics Subsystem. A single-level graphics hardware subsytem. For every VGS, there is exactly one VFB, one TX/SLS, and one TX/WM.

8. Acknowledgments

The author appreciates the encouragement of his colleagues at Secure Computing Corporation for encouraging him to write this paper. Comments from the anonymous referees were also very helpful in improving the quality of the paper. Finally, we acknowledge the Trusted X project at TRW, which plowed the ground in which this idea grew. Key team members on that project (in addition to the author) were Hilarie Orman, John M^{c}Hugh, Rita Pascale, Marty Branstad, Ann Marmor-Squires, and Doug Rothnie.

9. References

[CMWREQS87] John P.L. Woodward, *Security Requirements for System High and Compartmented Mode Workstations*, DIA Document number DDS-2600-5502-87, November 1987.

[Loral91] Richard Sherman, George Dinolt, and Frank Hubbard, *Multilevel Secure Workstation*, U.S. Patent 5,075,884, December 24, 1991.

[TCSEC85] National Computer Security Center, *Trusted Computer Systems Evaluation Criteria*, DoD 5200.28-STD, Fort Meade, MD, December 1985.

[TXArch93] *A High Assurance Window System Prototype*, J. Epstein, *et al*, Journal of Computer Security, Vol. 2, No 2&3, 1993.

[VWS92] *Virtual Window Systems: A New Approach to Supporting Concurrent Heterogeneous Windowing Systems*, R. Pascale and J. Epstein, Proceedings of the 1992 USENIX Summer Conference, San Antonio TX, July 1992.