Interactive Video Game

Group Report

An ECE532 (Digital Hardware) project of 2005

Leo Hwang (990832060)
Timothy Li (990964505)

March 28, 2005

Table of Contents

1. Overview

1.1. Goals

The goal of our project was to create a video game where the user interacts by making gestures into a camera. Video game elements are superimposed onto the person's outline and then displayed on a monitor.

1.2. System Block Diagram

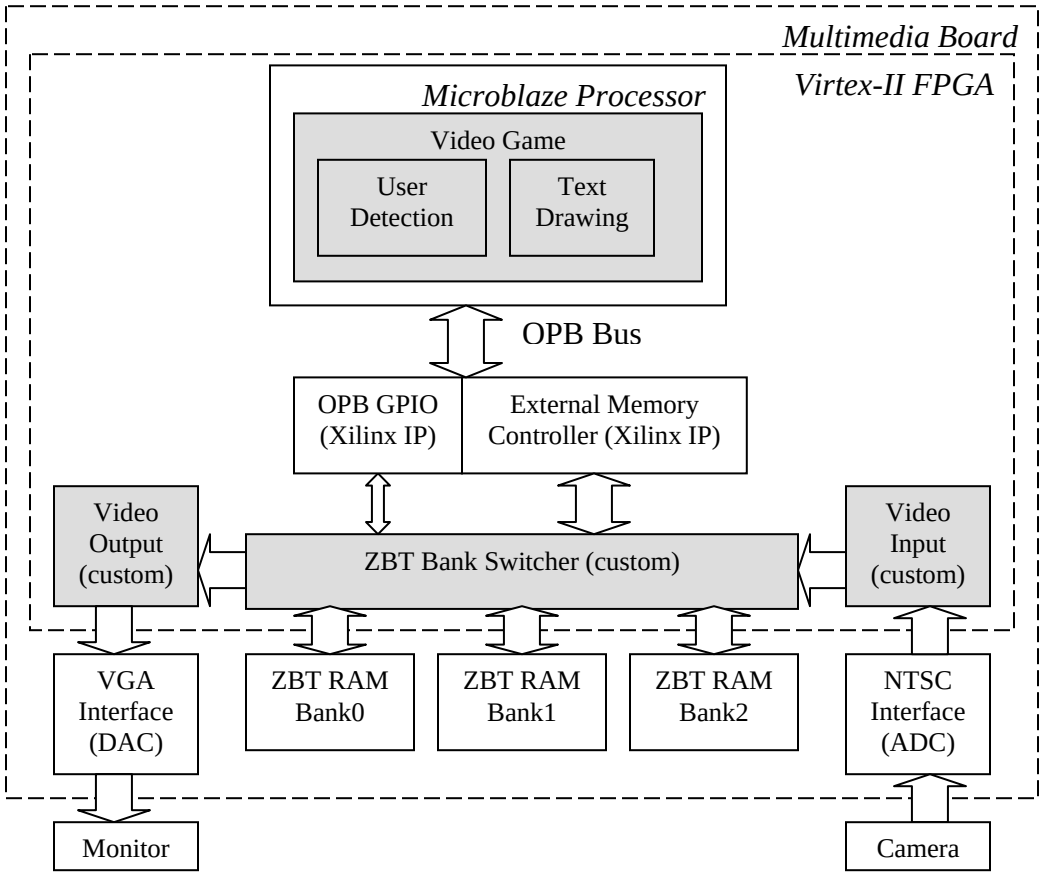

Figure 1 - Block diagram of project
(shaded blocks denotes those we created or modified)

1.3. Brief Description of IP

IP Name	What it does	Where it came from
External Memory Controller	Translates memory accesses on OPB bus into memory read/write commands.	Xilinx IP – with minimal changes
OPB GPIO	A set of software addressable input/output registers that can be connected to other IP.	Xilinx IP
Video Input	Reads video from a camera into a ZBT memory bank.	Custom
Video Output	Writes data from a ZBT bank to the monitor.	Xilinx example – modified to work in XPS
ZBT Bank Switcher (a.k.a. ZBT mux)	Connects three sets of memory controllers to three ZBT memory banks, and rotates the connections upon a particular signal.	Custom

1.4. Brief Description of Software

Software Block	What it does
Text Drawing	Converts a single character (number) into a bitmapped representation comprised of small squares.
User Detection	Finds the user's "flag" by partition the screen into 8x6 regions, and determining which region is most red.
Video Game	Blocks fall vertically down the screen, and the user must "catch" them with the flag before they reach the bottom.

1.5. Clock Domains

This project, due to its video input and output components, necessitated the use of different clock domains, as shown in Figure 2. Thankfully, the domains were naturally self-containing, and they only interacted with each other indirectly through the rotation of memory banks (more on this in section 3.5).

Each ZBT bank is driven by the clock that its controlling module is running at. There may be an abrupt moment when switching the banks where it is suddenly connected to another clock. However, switching the banks is only allowed during the vertical blanking interval of the video output, and when the software has finished its drawing routines. Hence, this combination gives a "safe-period", where no data is being transferred, to switch the memory banks across modules, with their respective clock domains.

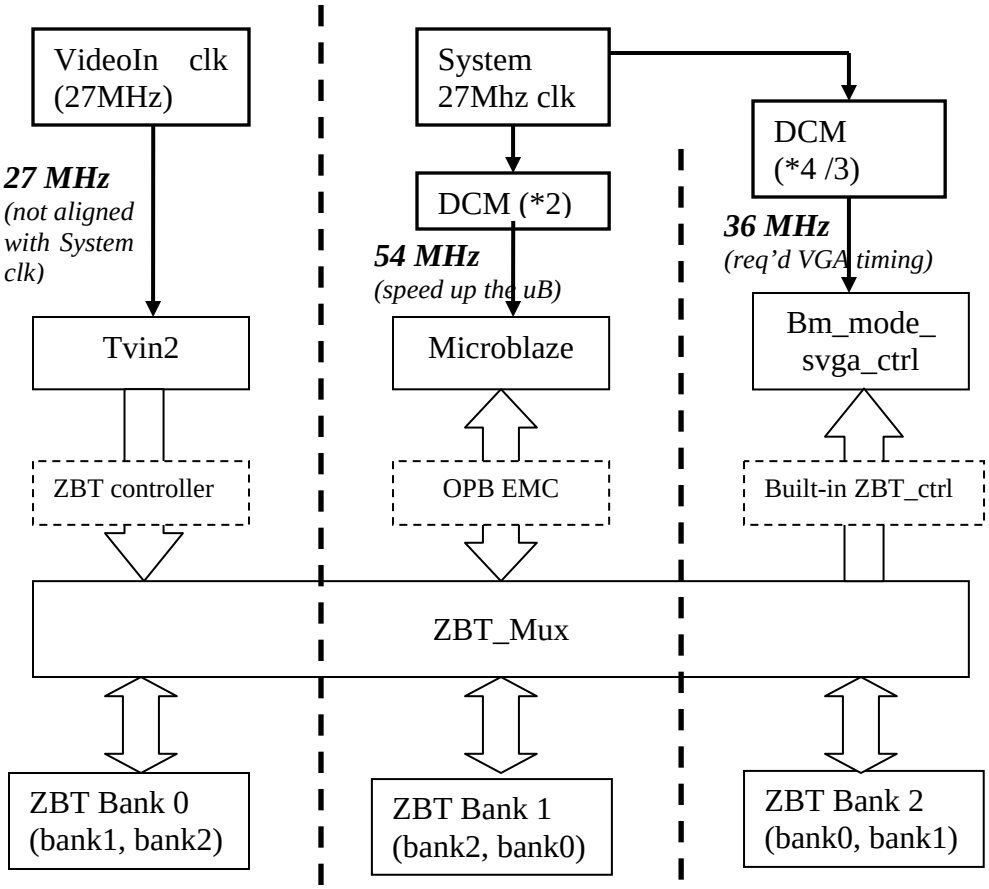

Figure 2 – Clock Domains found in project

2. Project Outcome

2.1. Review of Original Plan

The original proposal for the project stated that we would create an interactive video game, by using video input to superimpose the user right into the game. Of course, after the game elements were drawn, the combined display needs to be output. Tracking the user (to allow for the interaction) would have used some sort of motion or gesture detection.

2.2. Results

Overall, the outcome of the project was very close to the original plan. The final product was able to obtain user input through a camera, draw the game, and output to a monitor.

However, small compromises were made in three areas of the project. These compromises are non-detrimental; they slightly affect the quality of the result. The decision to not pursue their resolution stems from the short timeline for the project.

1. Instead of detecting the user's gestures, we detect a red object which the user holds.
2. Despite getting recognizable images from the video input, colours are not being decoded sharply or correctly.
3. The quality of the picture output is not as clean as we would like.

2.3. Compromises

2.3.1. Skin Detection -> Flag Detection

To detect the user's movements, we would need a skin detection algorithm. We decided to use something simpler and guaranteed to work: a red flag detection algorithm.

2.3.2. Saturation of YCbCr -> RGB Conversion

The YCbCr to RGB conversion was producing many false coloured pixels. We did not fix the problem, but focused on completing the remaining pieces of the project.

2.3.3. Lines in Video Output

After adding the Video Input block to the 3xZBT mux, we observed vertical lines in the display. This was assumed to be a timing issue. The vertical lines did not impede the user detection algorithm, so fixing this problem was put on low priority.

2.4. Suggestions for the Future

Obviously, the first place for improvement would be to the aforementioned compromises. Any or all of skin detection, fixing the YCbCr -> RGB conversion, or cleaning up the video output are worthwhile projects. Skin detection would require an implementation of various hue-based algorithms which can be found on the internet. The latter two would require a deep understanding of NTSC [color_spaces] [analog_decoder] or VGA [vga_dac] workings, above what we have learned in the course of the project. (To increase the picture quality, one needs to improve the YcbCr to RGB conversion algorithm found in ycrcb2rgb_v2mult_532.v.)

A hardware blitter – which we did not pursue, but would not be difficult now – could be inserted into the pipeline (via a fourth set of connections to the ZBT Mux and a fourth ZBT memory bank) to allow operations like clear screen or drawing large rectangles much faster than can be done with software. Commands could be sent to this module by software via GPIO or FSL connection.

As explained above, allowing hardware to access to video memory is straight-forward. Therefore, any hardware based video or image processing algorithm would be a project possibility.

Beyond this, more complicated projects could be built on top of this system. Within the speed constraints of this system, such projects are limited only by the imagination.

3. Description of Hardware Blocks

3.1. External Memory Controller

The OPB EMC module is a piece of Xilinx IP that maps access to a ZBT memory bank to a linear address segment on the OPB bus. It also generates the appropriate signals to control the ZBT memory, including accommodation for ZBT's pipeline delay.

In this project, it is used by the Microblaze processor to enable our software modules to read and write to the video buffer(s). The User Detection module reads the video memory looking for particular features, while the Text Drawing and Video Game modules draw the score and game elements, respectively, to the screen.

A small change was made to the module: its inout data-bus line was split into its constituent _I, _O, _T wires to conform with the input connections to the ZBT Bank Switcher. Otherwise it is identical to the Xilinx provided version.

3.2. OPB GPIO

The OPB GPIO module is a piece of Xilinx IP which allows software to write/read a register as data, whose bits can then drive/receive signals in hardware. In this project, one bit of output and input are used to form a trigger and notification to control buffer switching in the ZBT Bank Switcher. (See the ZBT Bank Switcher section for more details.)

3.3. Video Input (Tvin2)

The Video Input block is made up of three parts.

1. Lesley's video_input block [video_input]
2. Standalone ZBT memory controller [zbtcontrol]
3. A Verilog file to decipher video signals and send data to the ZBT controller (tvin2.v)

The video_input and ZBT controller blocks were not modified from their originals.

3.3.1. tvin2.v

This is the Verilog piece that transfers video data from the video_input block to the ZBT memory controller. In order to correctly place the incoming video data, an

understanding of the H (horizontal), V (vertical), and F (field) signals was required. The [linefeed] document contained the necessary information. We also needed to know the resolution of the NTSC format, so we referenced [coherence].

3.3.2. Video Input Testing

In order to test the correctness of the Video Input block, we needed some way to access the ZBT memory. By this time, we had a project which would display the ZBT memory contents on the screen. So the development of the Video Input block cycled between modifications, downloading the Video Input core, and then downloading the Video Output Core.

3.4. Video Output (bm_mode_svga_ctrl)

The Video Output block is based on the Xilinx BM_MODE_SVGA example [bmsvga]. The key modifications that allowed this module to work as a pcore were the removal of OBUFs and IOBUFs. OBUFs were directly replaced with `assign` statements, but IOBUFs had to have their bus lines separated into their _I, _O, _T components, and these wires propagated to the top of (and outside) this module. XPS would then resynthesize these (I)OBUFs at the system wrapper level.

3.4.1. Video Output Testing

Not much testing was needed for this module, as it was already shown to work virtually without modification from the Xilinx example by Lesley's ISE video output sample found in [video_input]. The difficulty was to reroute the wires correctly in replacing the IOBUFs. The module was complete when the appropriate modifications synthesized without error in XPS.

3.5. ZBT Bank Switcher (zbt_mux)

3.5.1. Overview

The ZBT Bank Switcher is the key component to the system. It switches the all the ZBT memory access signals emanating from each of the Video Input, EMC and

Video Output modules. It contains an internal counter to cycle the state of buffer rotation it is currently at. The buffer rotation is clocked on the vertical blanking interval from the Video Output module, and is triggered when an external signal changes value.

By cycling the memory connections from the modules between the three memory banks, a triple-buffering scheme is set up; this is reminiscent of those found in graphics engines. The key difference is that since this project is done in hardware, work is done in parallel, as each memory bank is worked one of the modules at any time.

A pipeline of Video Input -> EMC -> Video Out is constructed such that a buffer is filled with video capture data, then passed to the Microblaze for software processing, and finally passed to the video display for output.

3.5.2. Diagram

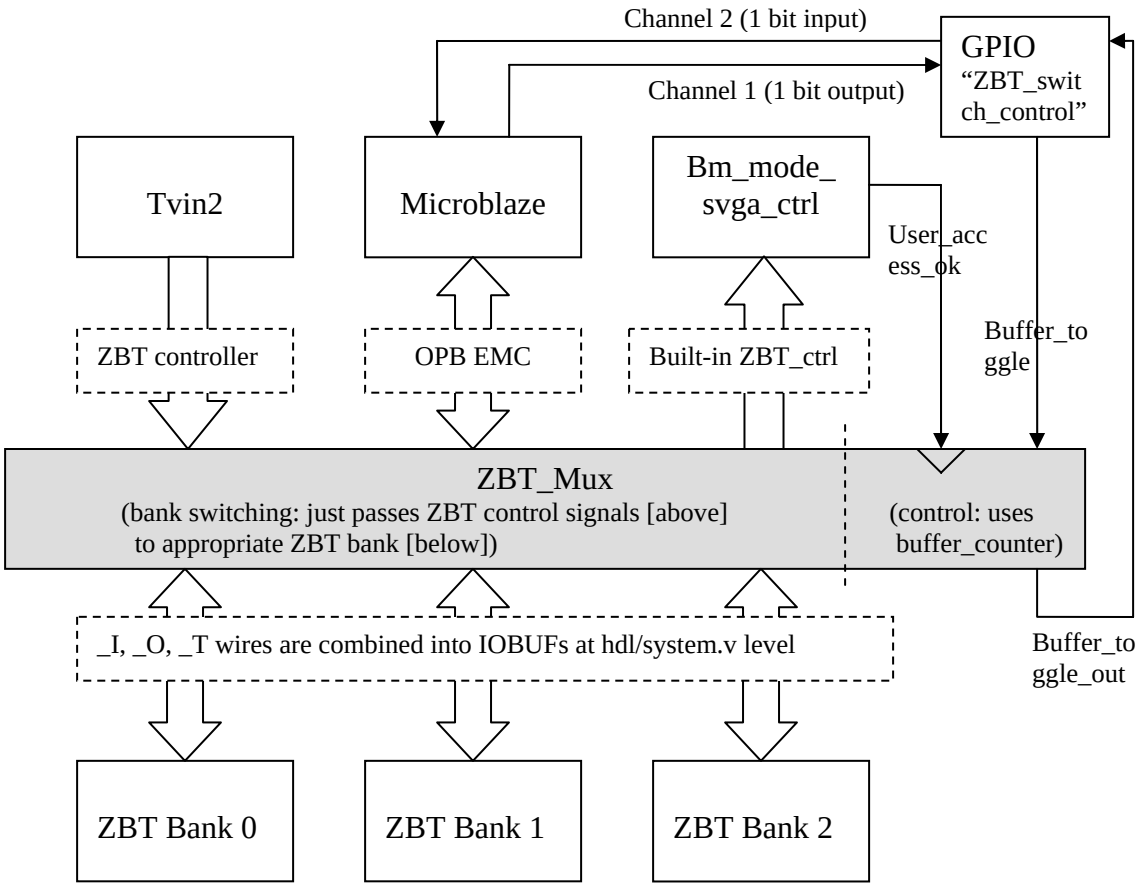

Figure 3 – Connections to/from the ZBT Mux and everything else it's attached to

3.5.3. Controlling Buffer Rotation

To control the bank rotation, `buffer_counter` sequences 0, 1, 2, 0, 1, 2, etc. `Buffer_counter` is incremented only on the positive edge of `User_access_ok` when `Buffer_toggle` (an input) is not equal to `Buffer_toggle_out` (a register). Synchronizing the rotation to the video output's vertical blanking interval ensures there is no tearing of the image.

`Buffer_toggle` is controlled by the user program. But the buffer rotation is not guaranteed to happen immediately since it is clocked to the vertical blanking interval (VGA currently displays at 85Hz). So the program should wait (poll) until buffer rotation is confirmed by reading a `buffer_toggle_out` value that is equal to the `buffer_toggle` value that was set.

Due to these timing issues, it was also a conscious decision to make `buffer_toggle` a toggling switch, instead of an on/off switch. The key idea is to cause a distinct singular event, and toggling prevents the need for interrupts to disable the switch later if an on/off switch were used.

3.5.4. Proper Cycling Order of Ports

Buffer switching is accomplished simply in Verilog with statements such as:

```
assign OUT1_CE = (buffer_counter == 0) ? INA_CE : (buffer_counter
     == 1) ? INB_CE : INC_CE;
assign OUT2_CE = (buffer_counter == 0) ? INB_CE : (buffer_counter
     == 1) ? INC_CE : INA_CE;
assign OUT3_CE = (buffer_counter == 0) ? INC_CE : (buffer_counter
     == 1) ? INA_CE : INB_CE;
```

Note how each output is connected to the input in a different sequence to achieve the pipeline effect.

Inputs must cycle the outputs in **reverse** order to correctly match the pipeline:

```
assign INA_DATA_A = (buffer_counter == 0) ? OUT1_IN_D_A :
     (buffer_counter == 1) ? OUT3_IN_D_A : OUT2_IN_D_A;
```

The non-intuitive order above was a problem resolved during development, requiring some consideration to understand and debug. It is summarized by the table below, which arranges the pipeline by time and fixed outputs (based on the output assign statements) –

observe how the inputs cycle from right to left as time progresses. (E.g. follow INA from top to bottom…)

TIME \ Output	OUT1 (Bank 0)	OUT2 (Bank 1)	OUT3 (Bank 2)
Cycle 0	INA (Camera)	INB (Microblaze)	INC (Monitor)
Cycle 1	INB (Microblaze)	INC (Monitor)	INA (Camera)
Cycle 2	INC (Monitor)	INA (Camera)	INB (Microblaze)

Despite possible timing issues (i.e. each input module uses a different clock!) a sufficiently "safe" time to switch the buffers is during the combination of the vertical blanking interval and the end of Microblaze draw routine. Synchronization to the video input was ignored because a system assumption is that the camera is in a fixed location – therefore it will keep writing the same data to its assigned buffer, and can be disconnected anytime after about 1/30 seconds (the NTSC frame rate).

3.5.5. ZBT Bank Switcher Testing

Because the video output module was working before this module was, the ZBT Bank Switcher was tested simply by downloading it to the board, and seeing whether correct video output was obtained. Odd effects tended to indicate some timing or netlist error, whereas having only certain frames displaying properly hinted at possible problems with the cycling order (as described above).

3.6. System.mhs and System.ucf

Although technically not hardware blocks, these two files respectively indicate the netlists between each instantiated module, and external pin connections to the Multimedia board [mboard_ug]. Therefore, these files are paramount to the understanding of the hardware setup of this project.

4. Description of Software

4.1. Text Drawing

This part of the software synthesized character output (although only digits are currently implemented) by using a crude 5x3 bitmap of "dots" (squares). Thus numbers were represented as in Figure 4, and this file contained the function to draw such arrays of dots, and the dot layout ("bitmap") for each digit. This system could be easily extended to letters, but was not pursued.

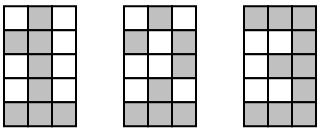

Figure 4 – Dot-based representations of the digits 1, 2, and 3

4.1.1. Text Drawing Testing

Testing this module required us to run the game and play diligently until we scored over 200 points, to ensure that all numbers were printed correctly, and three digits had the correct math to isolate that digit's value.

4.2. User Flag Detection

Each frame of video has a resolution of 640x480. This area is divided into 8 by 6 blocks of 80x80 pixels each. The algorithm locates the block containing the most "red" pixels. Of course, there is some lenient definition of "red," so that objects close to a red colour can be detected. This module was written in C (CheckRedFlag.h).

4.2.1. User Flag Detection Testing

To test the algorithm, we drew a box at the location returned by the algorithm. The "red" definitions and thresholds were modified and tested until the algorithm reliably detected a red object.

Testing the detection module was very quick, as software modifications only required a short recompile and download.

4.3. Video Game

The video game was written in C as well (Game.h). The game loop performs this sequence of actions:

- Detect the location of the user's flag
- Run game logic and write to video memory
- Switch buffers by toggling the ZBT Bank Switcher via GPIO

4.3.1. Video Game Testing

Testing the video game, like the detection algorithm, was done visually. There wasn't much testing to be done, as most of the game worked on the first run.

5. Description of Design Tree

README files have been placed in various locations to mirror this and provide additional information. In the interest of saving space, all synthesizable files have been cleaned from the project and snapshot directories.

5.1. Documentation

The “doc” directory contains this document and the demonstration presentation in PDF format.

5.2. NTSC_in_VGA_out_triple_buf

This folder contains the final version of the Interactive Video Game system. The HDL and code have been given additional commenting for the sake of this system’s future users. It has been checked that the comments have no adverse effects from the demonstration version.

Important files and directories to refer to are `system.mhs`, `data/system.ucf`, `pcores/*`, and `TestApp/src/*`.

5.3. Reference Designs

This folder contains compressed archives of designs we referenced while creating this project. Designs from Xilinx have been replaced with URL links to save space.

5.4. Snapshots

This folder contains snapshots of the project in various stages of development. Despite being partial designs, they are synthesizable and downloadable to the Multimedia Board, and will function to the stage of development they reflect.

6. References

6.1. Module References

[bmsvga] Bit Mapped Mode SVGA
http://www.xilinx.com/products/boards/multimedia/docs/examples/BM_MODE_SVGA.zip

[coherence] Video Scene Coherence, Frame Buffers, and Line Buffers
(xapp296_1_0.pdf)
from Video Input Processing Examples
http://www.xilinx.com/products/boards/multimedia/docs/examples/Video_Input_Processing_Examples.zip

[color_spaces] ycrcb2rgb (XAPP283.pdf) & 422 to 444 (XAPP294.pdf)
from Video Input Processing Examples
http://www.xilinx.com/products/boards/multimedia/docs/examples/Video_Input_Processing_Examples.zip

[linefeed] Line Feed Decoder (xapp286_04.pdf)
from Video Input Processing Examples
http://www.xilinx.com/products/boards/multimedia/docs/examples/Video_Input_Processing_Examples.zip

[video_input]
attached file reference_designs/Lesley_video_examples.tar.gz

[zbtcontrol] Standalone ZBT Memory Controller
http://www.eecg.toronto.edu/~pc/courses/edk/modules/6.3/zbt_no-opb.zip

6.2. Multimedia Board Chip References

[analog_decoder] Analog Devices ADV7185 - Advanced 12-Bit Video Decoder
http://www.eecg.toronto.edu/~pc/courses/edk/doc/video_decoder_ADV7185_0.pdf

[mboard_ug] Xilinx Multimedia Board – Users Guide
http://www.eecg.toronto.edu/~pc/courses/edk/doc/Multimedia_UserGuide.pdf

[vga_dac] FMS3810 – Fairchild Semiconductor Triple Video D/A Converters
http://www.fairchildsemi.com/ds/FM/FMS3810.pdf

[zbt_ram] Samsung Electronics - 512Kx36 & 1Mx18 Pipelined NtRAM
http://www.eecg.toronto.edu/~pc/courses/edk/doc/ZBT_k7n163601a.pdf