A Fancy Digital Clock: Project Design

Ridwan Sami Alex Bell Geoff Young

April 5, 2010

Abstract

This document presents the planned design of a fancy digital clock with some hardware-accelerated graphics effects. The clock outputs scrolling images and smooth fonts to a computer monitor via VGA. First, we present a broad overview of the device. Then we provide details about the individual components in the device.

1 Introduction

The project will comprise a single Altera DE2 board connected to a computer monitor through the VGA connector. The monitor will be split into three regions as in Fig. 1. The upper left region is a digital clock which will display the current time. The lower left region will display the current date and day of week. The right half of the screen will display a vertically scrolling series of preset static images and captions. The data in the three regions will be overlayed on top of the background image, which will be a hardware-generated pattern.

2 Implementation plans

Fig. 2 is an overview of the major components of the device. The images will be 16-bit bitmaps hard-coded into the C code as arrays, which will ultimately be stored in SDRAM. The C arrays are generated from Windows BMP files via Perl script. One or more images from SDRAM will be composited into a vertically long final framebuffer stored on SRAM for quick access by the VGA controller. Only a region of the vertical framebuffer will be visible on-screen. The vertical offset of the visible region will be incremented to simulate sliding or scrolling.

2.1 VGA controller

The SRAM will be directly integrated into the VGA controller—not attached to the Avalon bus—so the VGA controller has quick access to the framebuffer for feeding the VGA DAC. Since we can only read 16 bits per 50 MHz clock cycle and write 16 bits per 2 clock cycles, we will use the 16-bit R5G6B5[1] pixel format as opposed to the more popular 24-bit true color pixel format (R8G8B8). This will simplify hardware since the VGA controller will read/write an entire pixel per SRAM access.

The processor, after fetching data from its SDRAM image memory and doing computations, will tell the VGA the color of each pixel in the buffer. The data from the processor will be written into a queue in block RAM before being committed to SRAM as a way of scheduling SRAM reads and writes. We are only reading from SRAM when we are painting the right half of the screen so the queue will be emptied and the data will be written to the SRAM while we are painting of the left half of the canvas.

2.1.1 SRAM layout

The SRAM will hold a single 256×1024 buffer of 16 bit color information. Only a 160×240 region will be visible at a given time and the pixels will be doubled to fit the 320×480 right half of the screen. The rightmost pixels in the image buffer will never be visible but exist only so that the width of the rows is a power of 2, which simplifies the logic in seeking to the start of a row.

2.1.2 Queue

There are two pointers in the SRAM which point to the front of the queue and the back queue. When a command is sent to the VGA peripheral, it is put on the front of the queue and increments the front pointer.

[1] The extra bit is given to the green channel because human eyes are most sensitive to minor differences in green light intensity. As a side effect of biasing the green channel, our grays may not look completely gray.

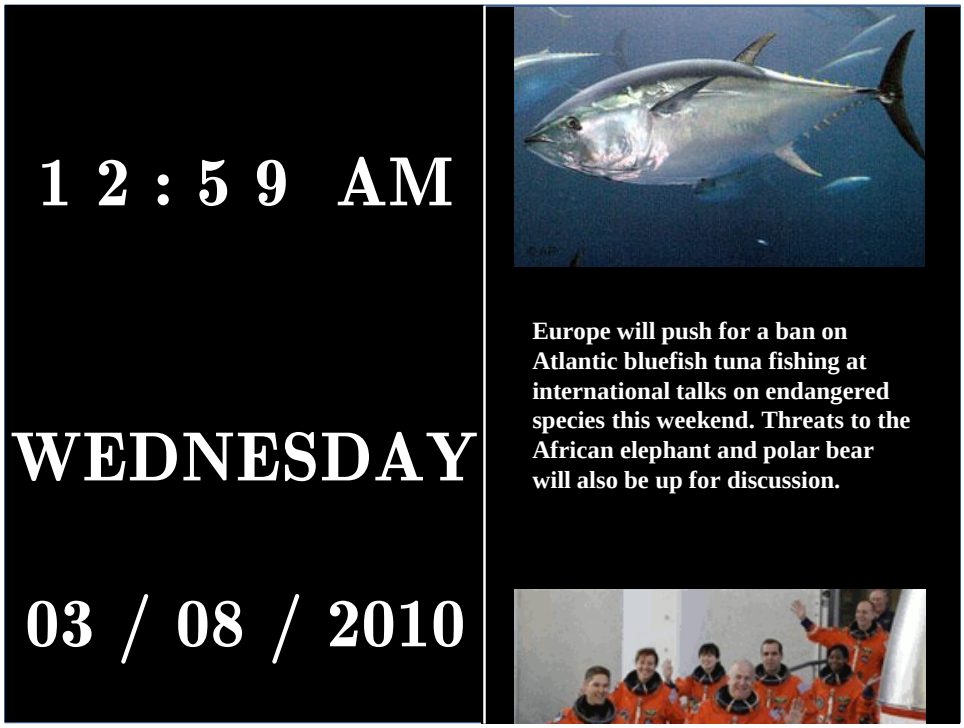

Figure 1: Our currently planned screen layout

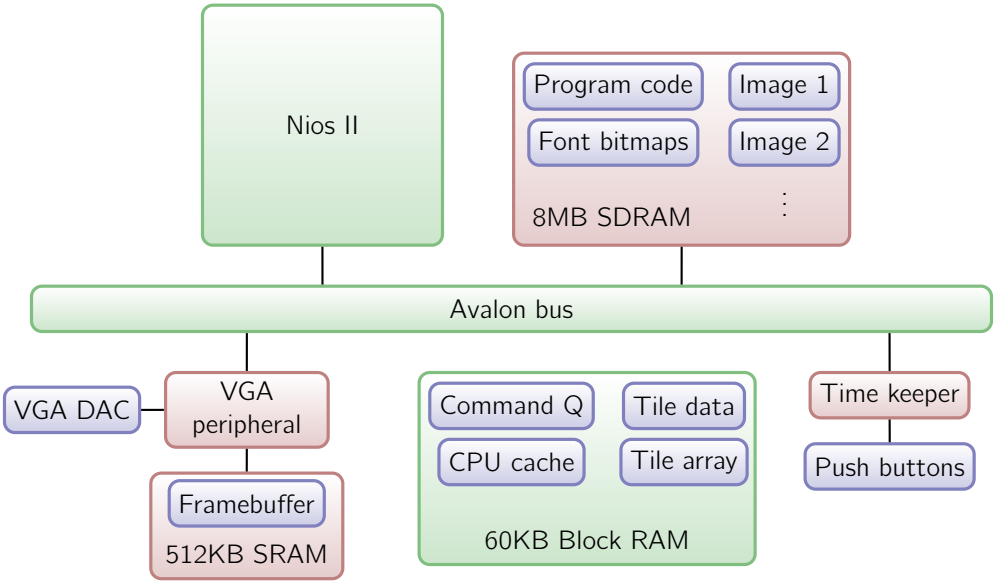

Figure 2: Overview of the components of the device

When a command is committed from the queue, the back pointer is incremented. When the front pointer and back pointer point to the same element, there are no commands in the queue waiting to be committed. The queue can store 63 commands although we may reduce the size if we find that we are never queuing more a certain number of commands. The queue pointers wrap around so it acts like a ring buffer.

Commands that do not deal with the SRAM could theoretically bypass the queue but we will put them on the queue for consistency and to simplify hardware logic. We may incur an insignificant delay during certain operations that do not touch SRAM.

2.1.3 Layers

The screen will be composed of two layers. The bottom layer is the background layer, which will be algorithmically generated in hardware on-the-fly as we scan across or it will be composed of small repeated tiles.

On top of the background, the upper layer will consist of elements such as the time, date, and images. Tiles will be used for text and numbers on the left half of the screen. The VGA controller will decide which pixels to display based on transparency bits that will be stored with the monochrome tile data. The characters used in the time, date, and captions will have 3 bits of transparency information so that edges appear smooth against the background. On the right half of the screen, a predetermined color—such as bright purple—will be designated a see-through pixel. This is similar to chroma keying used in video production and old computer video drivers. Since images are confined to the right half and text is confined to the left half, tiles will never intersect with the framebuffer.

2.1.4 Hardware/software interface

We are currently planning on using the `address` port as an opcode and 16-bit `writedata` port as data to request one of many operations:

- Load in pixel data into tile identifier x from CPU to block RAM. This is the command through which the processor sets up the tiles for the VGA hardware to use.
- Commit the current 16-bit pixel from `writedata` directly to the current pixel in SRAM. Also increment the pixel counter so subsequent writes will occur at the next pixel. Pixels will be sent from the processor to the VGA peripheral one at a time in the same order that the framebuffer is read out to the VGA DAC. Once we hit the 160th pixel on the row, the hardware should jump the pixel counter to the next row since we'll never need to write further than that; pixels beyond the 160th are always outside the screen area and only exist as padding.
- Seek to a particular row and start painting from there during subsequent write operations.
- Immediately change the offset dictating which portion of the vertically long image buffer is visible on screen. This offset is normally incremented during every vertical sync period to simulate smooth motion but in some cases, we may want to jump to a different part of the framebuffer.
- Update tile in position (x, y) in the tile array with a new tile identifier so that it displays a new number.

The only data we anticipate needing to read from the VGA peripheral is the vertical offset integer that determines what portion of the image buffer is visible. Using this information, we can make sure we only paint parts of the SRAM that are not currently being displayed.

2.2 SDRAM controller

The SDRAM will hold instructions and data for the processor. The SDRAM controller is mostly automatically generated by the SOPC builder. Although SDRAM is an order of magnitude slower than SRAM, we do not anticipate that it will be a bottleneck since we will allocate a generous cache to the processor so consecutive localized reads will not incur a large penalty. By hard-coding bitmap images and fonts into the C code, we avoid difficulties associated with interfacing with an additional image source peripheral such as an SD card reader, which is otherwise outside the scope of this project.

As the first actions upon starting the program code, the CPU will take the font tiles from SDRAM and write them to block RAM. By having the tiles set up in software, we always have the option of loading in tiles on the fly in case we need a new tile set. As the program runs, images will be taken from SDRAM and written row by row to the SRAM attached to the VGA controller. Interrupts from the time keeper will prompt the CPU to change the tiles that show the current time.

2.3 Time keeper

This peripheral will simply generate an interrupt every 50 million clock cycles of the 50 MHz clock to tell the processor to advance the displayed clock by one second. Upon startup, the clock will display 12:00 PM on January 1st, 2000. By holding the buttons on the board, the user will be able to change the time. Pressing the buttons will generate interrupts that will tell the CPU to display different tiles to display different numbers.

Accurate time keeping is actually an auxiliary task. Our primary focus is fancy graphics and the *digital clock* aspect simply gives us an opportunity to use tiles with alpha channels and makes the device useful.

3 Milestones

Milestone 1: March 29 Hard-code a bitmap into the C code; get the processor to transfer this bitmap from SDRAM to the VGA controller, which will write it to SRAM and display it. Also, have the hardware increment the vertical offset every vertical blanking period to scroll the framebuffer.

Milestone 2: April 12 Scrolling right-hand side should be working now. Start painting tiles with alpha channels (e.g. bitmap fonts).

Milestone 3: April 28 Introduce a special effect (scrolling pictures fade in at top and fade out at bottom? Digital clock numbers appear to flip when changing digits?).