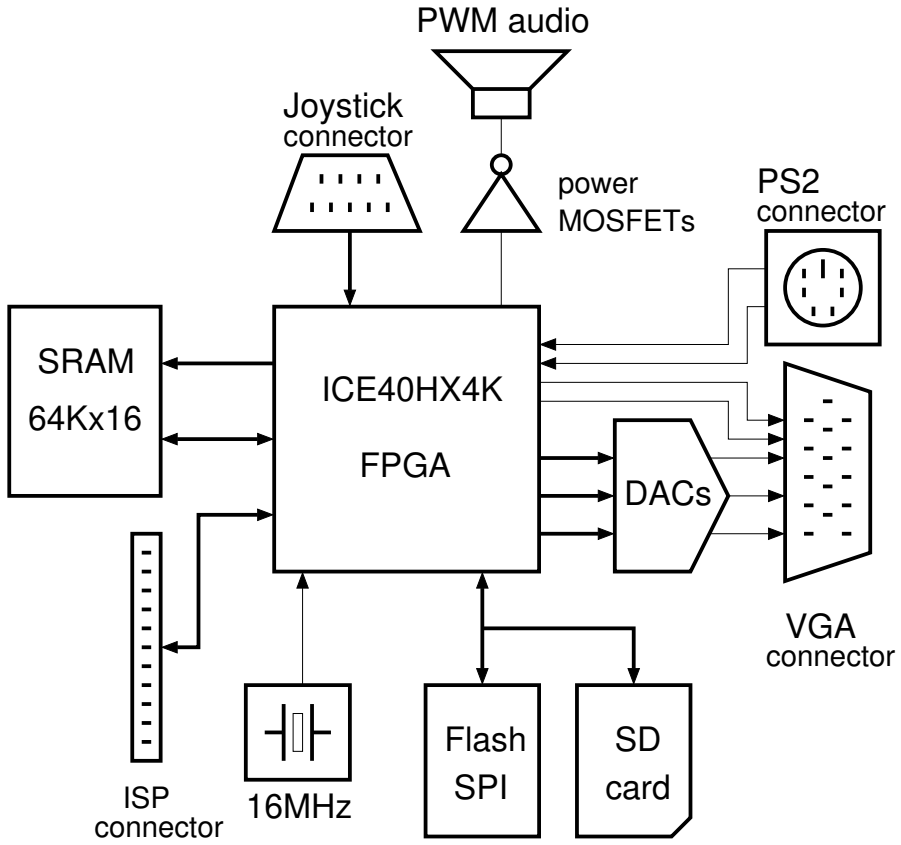
DRM65, a 6502 system with video and MMU in a FPGA

Jesús Arias

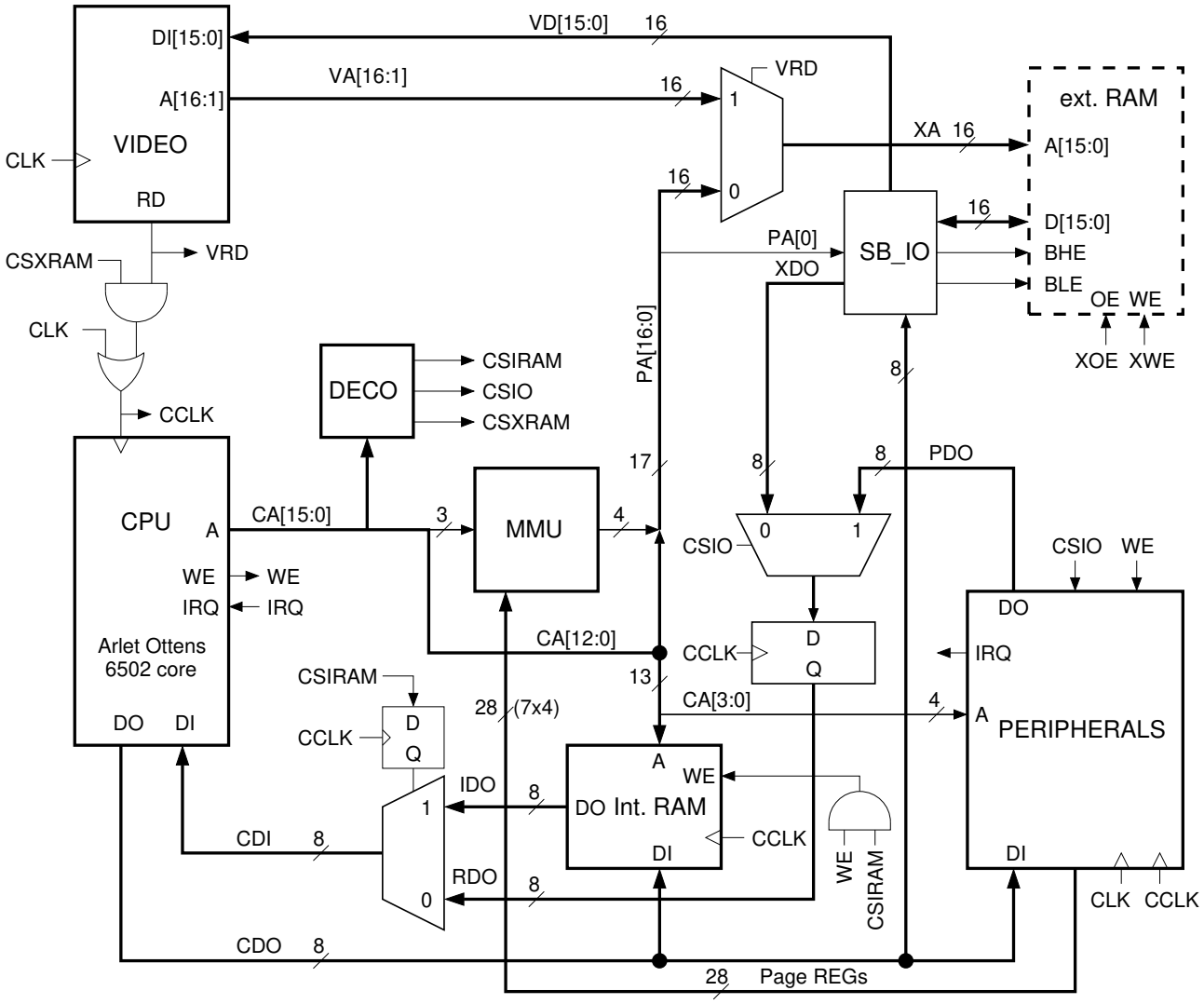
This 6502 prototype is built around an FPGA board designed for the emulation of 8-bit systems. Its block diagram is shown in the following figure:



It includes a Lattice ICE40HX4K FPGA along with a SPI flash for its configuration, a 16MHz clock generator that drives its internal PLL, an external SRAM memory with a 64K×16 capacity, and the interfaces to a VGA monitor, a PS2 keyboard, audio, and joystick. This particular FPGA was chosen because of its affordable price, relatively easy to solder package, and the availability of open source synthesis tools.

The VGA compatible video signal is generated by means of three 4-bits DACs, one for each color component, and two digital signals for the horizontal and vertical sync pulses. This allows us to display up to 2^{12}, or 4096 different colors on the screen. Yet, in order to reduce the amount of video RAM required the color depth is only 1 or 4 bits per pixel, so not more than 2 or 16 colors can be displayed on the screen simultaneously. The DACs are nothing more than arrays of resistors.

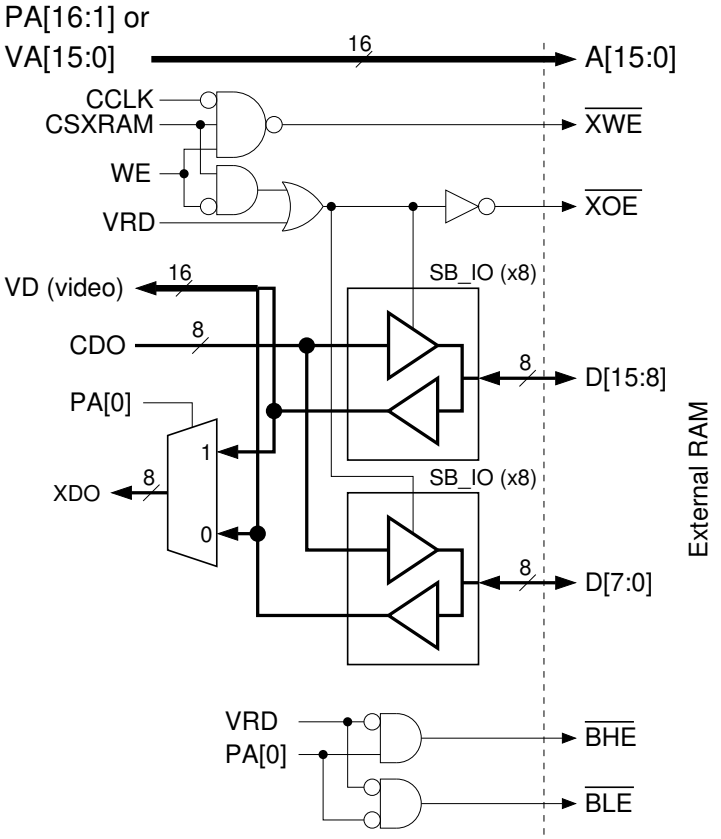
The logic built into the FPGA includes the 6502 core of Arlet Ottens, along with 8KB of boot memory, a video generator, the interface for the external memory, a minimum memory management unit, MMU, and several peripherals: Serial port, interrupt logic, PWM audio, and a PS2 keyboard interface. The block diagram of the synthesized logic is shown next:



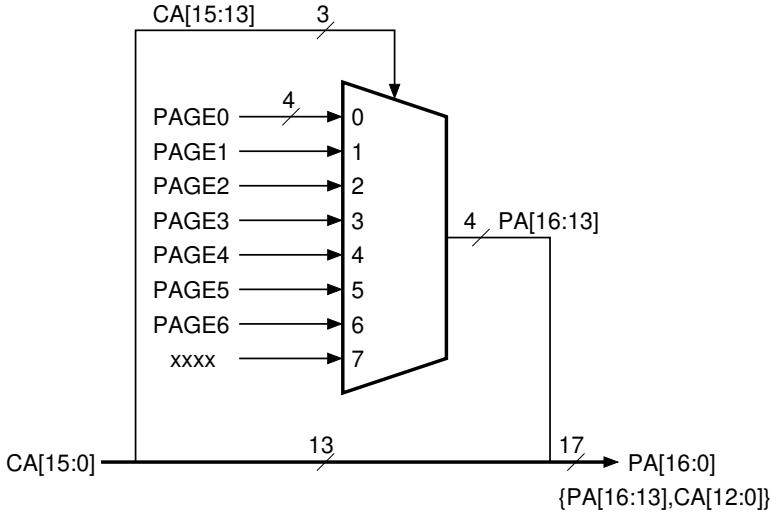
In this design the internal clock runs at 25MHz and the video timing matches the 640×480 VESA standard, but the resolution is only 512×400 pixels, mainly because having an horizontal resolution that is a power of two eases the design of the video address generation. The video generator steals clock cycles from the CPU when both blocks are addressing the external memory. In the monochrome mode one every sixteen cycles during the visible part of a video line goes to video instead of the CPU, while for the color mode this happens one every height cycles. So, the effective clock rate of the 6502 is 24.2MHz for the monochrome video mode or 23.4MHz for the color video mode. But these figures are averages. During the horizontal or vertical borders of the image the CPU runs at its full speed of 25MHz, and the same happens if the CPU is executing code from the internal memory. The video logic implements its cycle steal by forcing the CPU clock high when reading video data. The RDY input of the 6502 core is not used.

The block diagram is more complicated than expected because the 6502 core is designed to be be connected to a synchronous memory. The internal memory of the FPGA is of this type, but the external RAM, and also the peripherals, are asynchronous. Because of this a register for the input data bus has to be included, and the signal for the multiplexer that selects between internal RAM data or that coming from the registered external memory or peripherals has to be delayed one CPU clock cycle.

The decoder block is simply a combinatory-logic circuit that drives the corresponding selection signal depending on the value at the address bus of the CPU. Two other blocks that deserve a more detailed description are the bidirectional data bus interface for the external memory, SB_IO, and the MMU. The schematic of the former is:



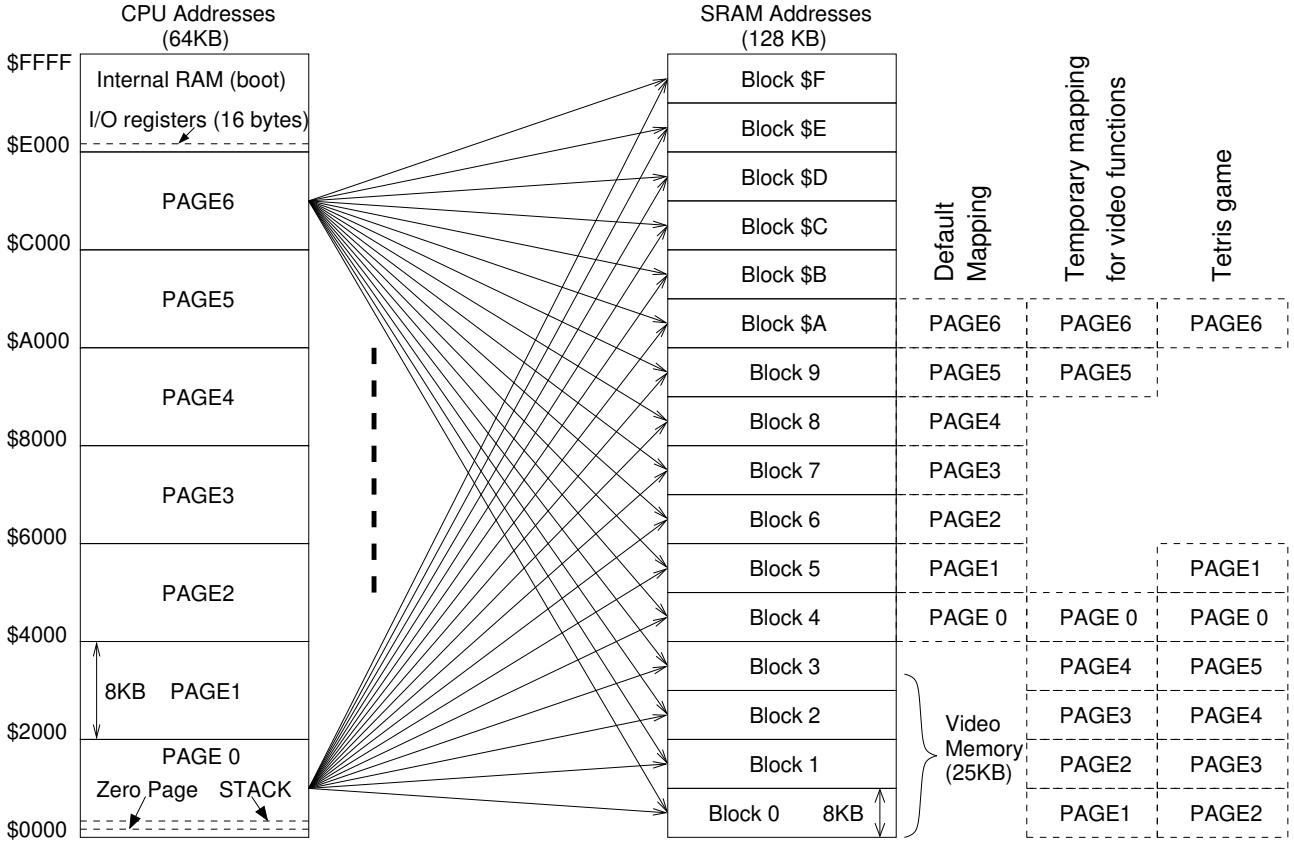
In addition with coping with the bidirectional data bus of the external RAM, the SB_IO block also manages the reading and writing of 8-bit values into a 16-bit memory. But, while all the CPU reads and writes are 8-bit wide, the video controller reads 16 bits every time it does a memory access. The bus-high-enable, BHE, and bus-low-enable, BLE, signals are generated accordingly, also taking into account that for 8-bit data even addresses are stored in the lower 8-bits of the memory, and odd addresses in the upper 8-bits.



The MMU is basically a multiplexer that exchanges the 3 upper bits of the CPU address with 4 bits coming from 7 4-bit registers of the peripherals block, and by doing this it converts the CPU addresses to physical addresses. Notice that there is no register for the upper address range because in that case the internal memory or I/O registers are selected instead of the external memory and, therefore, the generated address doesn't matter.

Memory MAP

The prototype has the following memory map:



8KB of the internal FPGA RAM is used as a boot memory because its initial content can be programmed via the configuration bitstream of the FPGA. But it is a read/write memory, not a ROM, it was made writable in order to be able to place breakpoints into its code. It is mapped to the upper 8KB of the address space because the reset vector is located there (addresses $FFFC and $FFFD). The available internal memory is slightly less than 8KB because the first 16 bytes starting from address $E000 are reserved for peripherals.

The rest of the CPU address space is mapped to the external RAM via 7 page registers which select what 8KB block of the RAM is assigned to each 8KB page of the CPU. The 7 page registers are located in the I/O space starting at address $E008 and ending in address $E00E. Only the 4 lower bits of each register are implemented. The mappings actually used in the boot code and a video game are also detailed, but other mappings are possible. For instance, a multitasking scheme would use a different PAGE 0 for each task, allowing to have a different zeropage and stack for each task.

I/O Space

Addresses $E000 to $E00F are reserved for I/O registers. Their location and description follows:

addr	reg	WRITE 7	6	5	4	3	2	1	0	reg	READ 7	6	5	4	3	2	1	0
$E000	UTXD	TXDATA								URXD	RXDATA							
$E001	BORDER	---	---	---	---	Palette Index				USTAT	THRE	DV	FTXI	FRXI	---	TEND	OV	FE
$E002	CTRL1	CHSI	CVSI	CDLI	SDLI	EHSI	EVSI	ETXI	ERXI	STAT1	FHSI	FVSI	HSYN	VSYN	EHSI	EVSI	ETXI	ERXI
$E003	CTRL2	---	---	---	---	---	---	EKBI	VMOD	STAT2	FKBI	KBDV	---	---	---	---	EKBI	VMOD
$E004	PINOUT	MOSI	SCK	/SS0	/SS1	res	res	res	res	PINOUT	MOSI	SCK	/SS0	/SS1	res	res	res	res
$E005	PAL0	Green				Red				PININ	MISO	J6	J5	J4	J3	J2	J1	J0
$E006	PAL1	Index				Blue				---	---	---	---	---	---	---	---	---
$E007	PWM	PWM level (audio)								KBD	KEYBOARD scan code							
$E008	PAGE0	---	---	---	---	ADDR(MSB) for PAGE 0				PAGE0	---	---	---	---	ADDR(MSB) for PAGE 0			
---	---					---				---					---			
$E00E	PAGE6	---	---	---	---	ADDR(MSB) for PAGE 6				PAGE6	---	---	---	---	ADDR(MSB) for PAGE 6			

- UART data registers, UTXD and URXD ($E000). A write starts transmission. A read recovers the last received data. Both transmitter and receiver include holding registers (like a 1-byte FIFO), so, a character can be being transmitted or received while another valid data is stored in the corresponding holding register. The UART data format and speed are fixed as 8-bit, no parity, 1 stop bit, and 115200 bps. Speed can be changed by selecting a different 'DIVISOR' parameter for the UART module (now its value is 217: 25MHz/217=115207) and running the FPGA synthesis again.
- UART status, USTAT ($E001), read only with the following flags:
  - Bit 0: FE, Frame Error. Set to 1 when a received stop bit was low.
  - Bit 1: OV, Overrun. Set to 1 when a character is received while the holding register is already full.
  - Bit 2: TEND, Transmission End. Set to 1 when the holding register and the shift register of the transmitter are both empty.
  - Bit 4: FRXI, Flag for the RX interrupt. Set to 1 when DV is active and the RX interrupt is enabled (bit 0 of CTRL1).
  - Bit 5: FTXI, Flag for the TX interrupt. Set to 1 when THRE is active and the TX interrupt is enabled (bit 1 of CTRL1).
  - Bit 6: DV, Data Valid. Set to 1 when a character is received.
  - Bit 7: THRE, Transmitter Holding Register Empty. Set to 1 when there is space available for transmitting a new character. A logic 1 do not means the transmission is complete (see the TEND flag), it means that another character can be queued for transmission.
- Border color, BORDER ($E001), write only. Its 4 lower order bits selects the color palette entry to be assigned to the border of the screen.
- Control register, CTRL1 ($E002), with the following bits:
  - Bit 0: ERXI, Enable the UART receiver interrupt if 1. Interrupts are requested when DV is 1.
  - Bit 1: ETXI, Enable the UART transmitter interrupt if 1. Interrupts are requested when THRE is 1.
  - Bit 2: EVSI, Enable the video VSYN interrupt. Interrupts are requested on the falling edge of the vertical sync. pulse.
  - Bit 3: EHSI, Enable the video HSYN interrupt. Interrupts are requested on the falling edge of the horizontal sync. pulse.

  - Bit 4: SDLI, Set a Delayed Interrupt. Writing this bit with 1 triggers an interrupt 23 clock cycles later. This is done for single instruction stepping in debuggers. This bit is not stored.
  - Bit 5: CDLI, Clear the Delayed Interrupt. Writing this bit with 1 clears the delayed interrupt request immediately. This bit is not stored.
  - Bit 6: CVSI, Clear the VSYN Interrupt. Writing this bit with 1 clears the VSYN interrupt. This bit is not stored.
  - Bit 7: CHSI, Clear the HSYN Interrupt. Writing this bit with 1 clears the HSYN interrupt. This bit is not stored.

- Status register, STAT1 ($E002), with the following bits:

  - Bits 0 to 3: ERXI to EHSI: The same Interrupt enable bits of CRTL1
  - Bit 4: VSYN: The actual Vertical Sync. signal.
  - Bit 5: HSYN: The actual Horizontal Sync. signal.
  - Bit 6: FVSI: Flag of the Vertical Sync. Interrupt. Set to 1 when this interrupt is pending.
  - Bit 7: FHSI: Flag of the Horizontal Sync. Interrupt. Set to 1 when this interrupt is pending.

- Control register, CTRL2 ($E003). With the following bits:

  - Bit 0: VMOD: Video Mode. A value of 0 selects a monochrome video mode with 1 bit per pixel and a 512×400 pixel resolution. In this mode each byte of the video memory contains 8 pixels, with the MSB corresponding to the pixel displayed on the left. The pixel value selects between the first and the last entries of the color palette table (index #0 or index #15). A value of 1 selects a color video mode with 4 bits per pixel and a 256×200 pixel resolution. In this mode each byte of the video memory contains 2 pixels, with the high nibble corresponding to the pixel displayed on the left. Each nibble selects one on the 16 possible color palette entries.
  - Bit 1: EKBI: Enable Keyboard interrupt if 1. Interrupts are requested when an scan-code is received.

- Status register, STAT2 ($E003), with the following bits:

  - Bit 0: VMOD: Video mode. The same bit 0 of CTRL2.
  - Bit 1: EKBI: Enable Keyboard interrupt. The same bit 1 of CTRL2.
  - Bit 6: KBDV: Keyboard data valid. An scancode is available for read from the KBD register if 1. This bit is cleared reading the KBD register.
  - Bit 7: FKBI: Flag for the keyboard interrupt. A value of 1 indicates a pending keyboard interrupt. This flags is cleared reading the KBD register.

- Output pins, PINOUT ($E004). Its bits controls the logic levels applied to some external pins of the FPGA, like those connected to the SPI flash memory of the board.

- Input pins, PININ ($E005), read only. Its bits reflects the state of some input pins of the FPGA, like that of the serial data coming from the SPI flash or the switches of the joystick interface.

- Palette registers, PAL0 and PAL1 ($E005 and $E006), write only. When writing to PAL0 the data value with the red and green levels is stored into an 8-bit temporary register. This register is concatenated to the blue level, coming in the 4 lower bits of the data written to PAL1, and stored at the palette address selected by the higher 4 bits of the data. Thus, in order to write a palette entry first write to PAL0 and then write to PAL1.

- PWM register, PWM ($E007), write only. A PWM signal is generated using the horizontal counter of the video generator. The PWM output goes high after the falling edge of HSYN and returns to low when the PWM register, multiplied by two, matches the horizontal counter. The duty cycle is thus restricted to be between 26% and 90%. The value stored in the PWM register must be changed during the 208 clock cycles after the HSYN edge (before the displayed video) or some glitches could appear at the output. The PWM frequency is the same of HSYN, that is 25MHz/800 = 31250 Hz. This PWM signal can be used to reproduce sampled sounds.

- Keyboard input register, KBD ($E007), read only. The PS2 keyboard scan codes are stored here. This register has to be read only once because a read sets its value to $FF, and also clears the keyboard interrupt.

- Page Registers, PAGE0 to PAGE6 ($E008 to $E00E). These registers controls the mapping between CPU addresses and RAM addresses. Only the 4 lower bits are implemented.

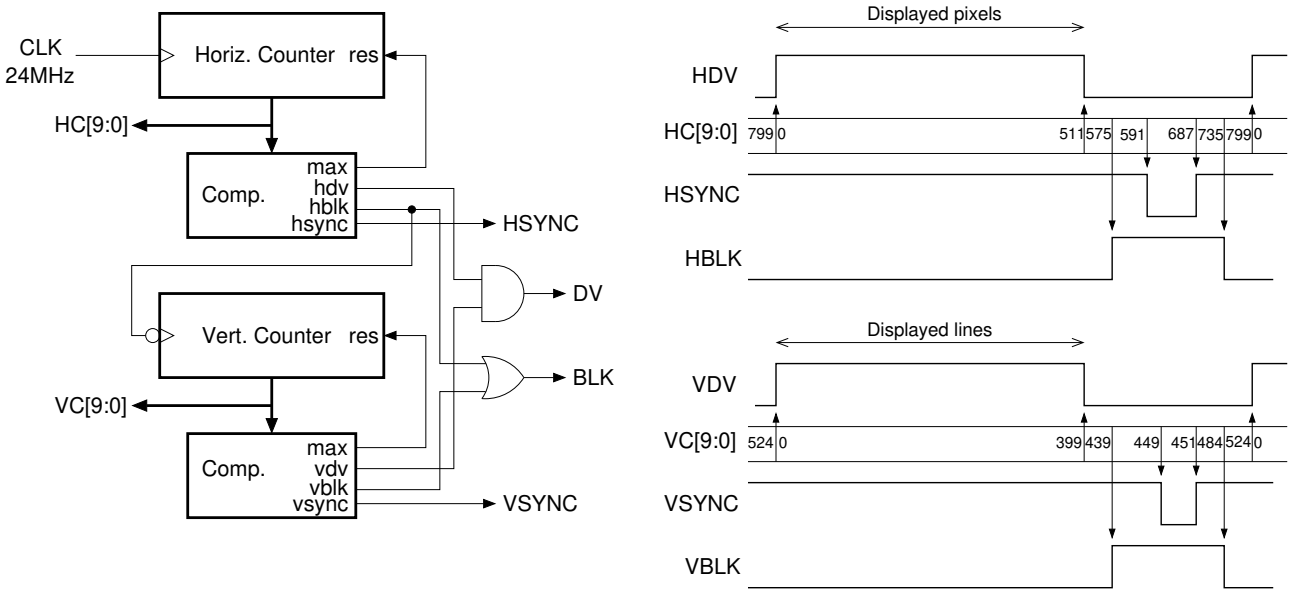
Video generation

Timing

The video generation block deserves some detailed explanation. It was designed to generate a VGA compatible signal and, thus, the timing of the standard 640×480×60Hz video mode was selected. but I wanted a power of two for the horizontal resolution, so, a data valid, DV, signal is active during the displayed part of the line that comprises only 512 pixels. The blanking signal, BLK, is active during the retrace time of the line (front porch, plus sync pulse, plus back porch). There are some times when DV and BLK are both low. In these cases a plain color is displayed for the borders of the line.

The vertical timing is the same as the standard VGA mode, but the displayed part is only 400 lines instead of 480. The same considerations for blanking and borders also apply to vertical timing.

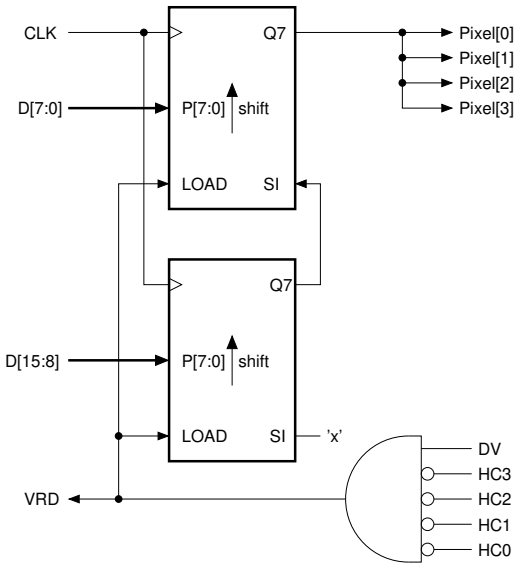
The block diagram of the timing logic is shown in the next figure, along with the corresponding timings for its horizontal and vertical blocks.



This logic generates the Horizontal and Vertical sync signals for the VGA monitor, and also a data valid, DV, signal that is active when refreshing the visible part of the video frame, and a blanking signal, BLK, that is active during retraces. It consist of two 10-bit counters and some comparator logic to select the appropriate modulus for each counter and to activate and deactivate the data valid, sync, and blanking signals.

Video shifter, 1bpp

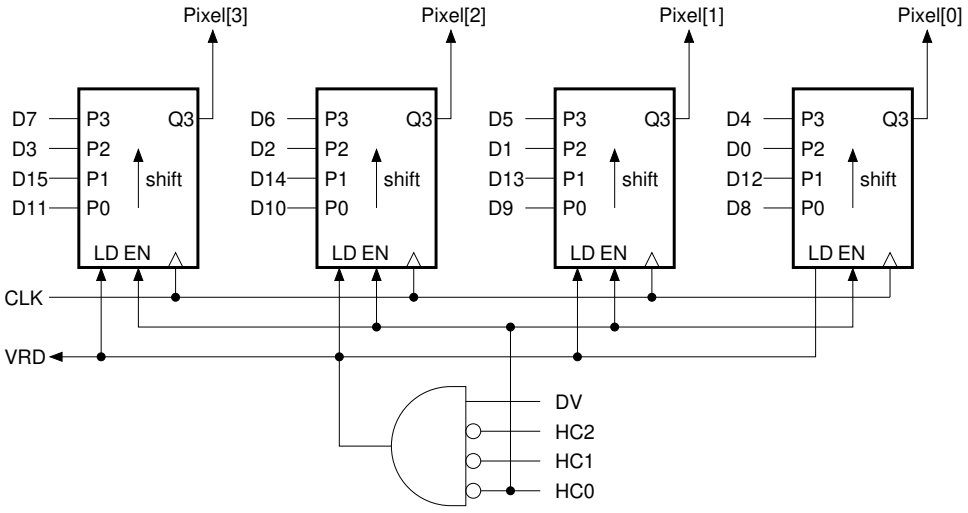
The video generator supports two modes, one of them has a 512×400 pixel resolution and 1 bit per pixel. The equivalent diagram of the video shifter for this mode is shown next:



Video data is loaded 16 bits at a time into a shift register using a mixed endian configuration. In this way the most significant bit of the lower byte of a 16-bit word is shifted out first and then comes the high byte. Each clock cycle there is a new single-bit pixel out of the shift register. The shift register is reloaded every 16 clock cycles when the signal DV is active. If DV is low no memory reads are performed and the shift register becomes empty, but its output doesn't matter because in that case we are displaying the border or in a retrace.

Video shifter, 4bpp

The video generator also supports a 16 color mode, but in this case the horizontal and vertical resolution were halved in order to keep the required memory low. In this mode each pixel is output twice using a shifter with the following block diagram:

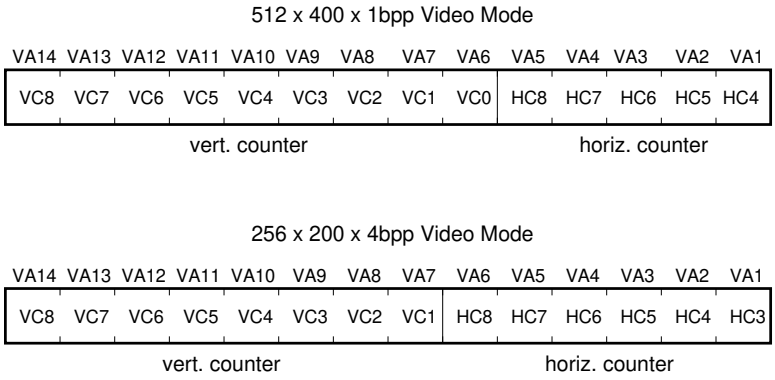


Notice that the 4 shift registers do not shift on odd pixels, so, this is equivalent to have half the clock frequency or half the horizontal resolution. Memory is read every 8 clock cycles if DV is active.

While this circuit looks different that the one for the monochrome mode its actual components are the same. The shift register is configured as a single 16-bit shifter or a quadruple 4-bit shifter depending on a mode signal. Data is routed differently on each case but the flip-flops are the same. The two equivalent circuits are shown here instead of the actual circuitry for the sake of clarity.

Video address generation

The video memory is always read as a 16-bit data, therefore no bit 0 is included in the video address. The actual memory address depends on the selected video mode, but in any case it is built by concatenating some bits of the horizontal and vertical counters, as it is shown in the following figure:

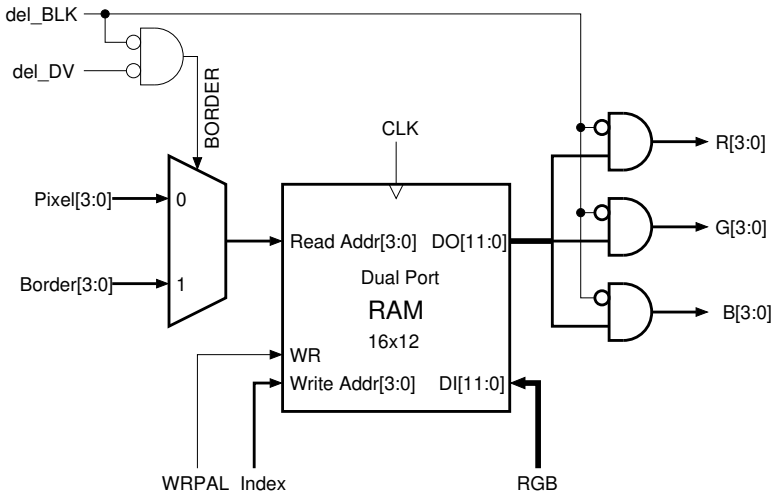


Notice that for the 4-bpp mode the least significant bit of the vertical counter is not included in the video address, resulting in lines being duplicated.

The total memory addressed by the video generator is 12.5KWords or 25KBytes (The vertical data valid signal becomes low when the vertical counter reaches the value 400 and no higher addresses are read, so, the highest video address is 0x63FF or 25599). The VA15 and VA16 lines are always zero.

Video palette

The output of the video shifter is a 4-bit data stream, Pixel[3:0], that is connected to the palette circuit of the following figure:

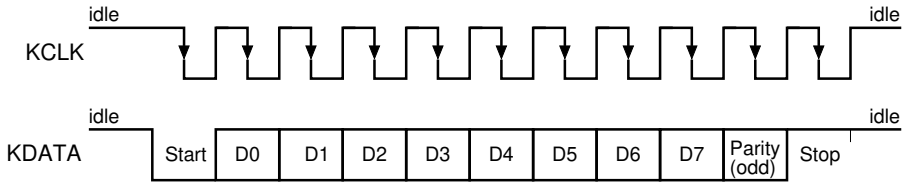


The palette consist mainly of a RAM memory with separated read and write buses that translates the 4-bit pixel value into a 12-bit RGB color output. The internal memory of the FPGA is of this type and the palette ends reserving a single memory block (256x16 bits). In addition to the RAM, there are two more blocks in the palette logic. One is for selecting an alternative entry when displaying the border region of the screen, and the other is for forcing a black color during retraces because VGA monitors can get crazy otherwise.

Also notice that here the data valid and blanking signals are delayed two clock cycles in order to take into account the delays of the video pipeline (one cycle for the shifter and another cycle for the palette memory)

PS2 keyboard interface

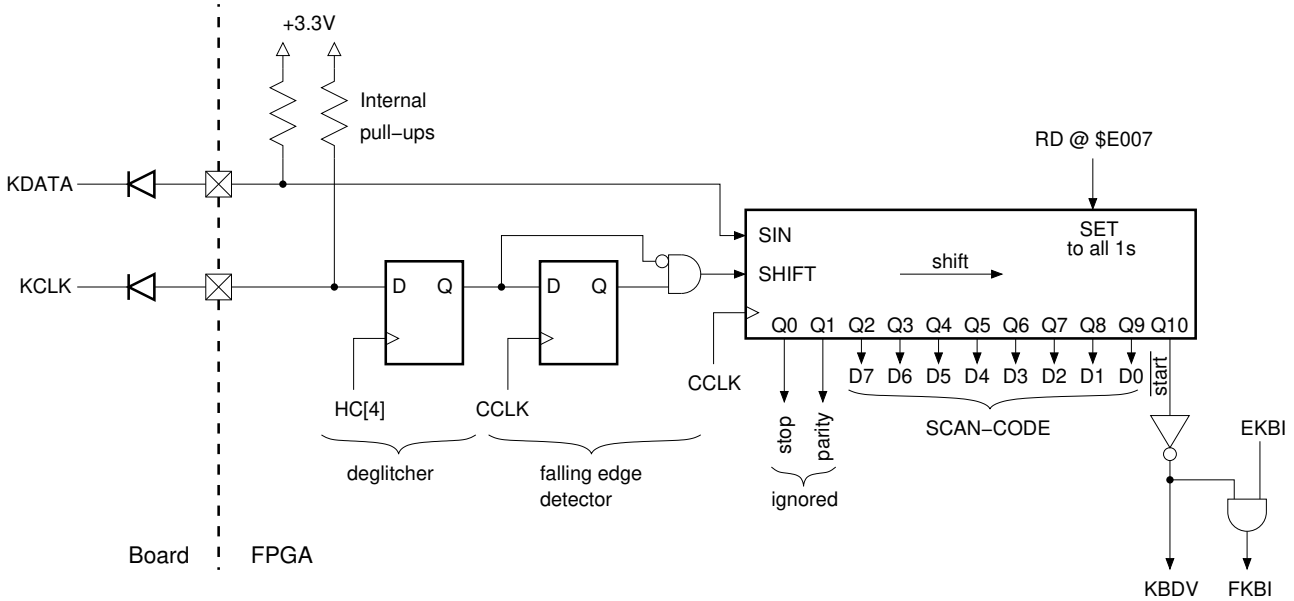
The interface for a PS2 keyboard is a perfect example of a quite simple circuit creating a lot of trouble. In the connector we get two lines, one for clock and other for data, which are driven by the keyboard. Only rarely these lines are driven by the computer, and this possibility wasn't considered here, but in order to allow for the required bidirectionality both lines are of the open collector type. When a key is pressed a data frame is sent to the computer following the chronograph of the figure:



The clock frequency can vary between 10kHz and 16kHz, and as it is already shown, 11 bits are transmitted, including the 8 bits of the key scan-code, one parity bit (odd), an start bit which is always low, and another stop bit which is always high. Data is valid on the falling edges of the clock.

The designed peripheral is little more than a plain 11-bit shift register with its shifting controlled by the detected falling edges of the keyboard clock. It starts with all its bits set to high, and, when the start bit arrives at the eleventh output of the register its low value signals an available scan-code to read. The reading of the scan-code also sets all the bits of the register to one, leaving the register ready to receive a new scan-code.

The logic of the PS2 peripheral is simple enough, but we also have to take into account the electrical interface. First, we can't connect the keyboard signals directly to the FPGA because these are 5-volt signals and the FPGA pins aren't 5-volt tolerant. This problem was solved by placing two diodes in series with the inputs and enabling two pull-up resistors to 3.3 volt inside the FPGA. In this way, when the data or clock lines are low the corresponding FPGA pin is also low, and for the other case the pull-ups will keep the input at 3.3V while in the keyboard side the same signal is at 5V. The block diagram of the interface, including diodes and pull-ups is shown next:



Well, the diodes actually degrade the low logic level of the signals, but +700mV should still be recognized as low. Just in case, a low forward voltage diode, like a Schottky, could be used instead of a regular small-signal diode. But these diodes weren't responsible for the receiving errors that plagued the first versions of the peripheral. In the diagram there is a flip-flop labeled "deglitcher" that was added in the path of the clock line as a remedy for the mentioned errors. The actual cause was the slow low-to-high transition in the clock line due

to the pull-up, and also the fact of being asynchronous with respect to the internal clock. At first I suspected the lack of Schmitt-trigger buffers for the input pins that resulted in "dirty" clock edges inside the FPGA, but the FPGA pins actually have more than 200mV of hysteresis, so this shouldn't be the problem. It could be related to metastability instead. But with the deglitcher clock connected to CCLK there are still some bad scancodes received (yet they are rare), and taking into account that these flip-flops can toggle at hundreds of megahertz the probability of a metastable cycle should be very low when running at 24MHz.

At the end I resorted to sample the keyboard clock using a slow clock signal coming from the horizontal counter of the video generator. HC[4] has a rising edge every 32 clock cycles or 1.28μs, and with this deglitcher in place the PS2 receiver finally worked perfectly. My guest is that the Schmitt-trigger hysteresis was not enough to filter the noise of the clock line, probably because the keyboard was connected using long wires into a breadboard (any ground wire has a inductance that can cause ringing on signal edges).

As a cautionary tale for any future design, we have to be extra careful about any edge-sensitive signal coming into the FPGA.