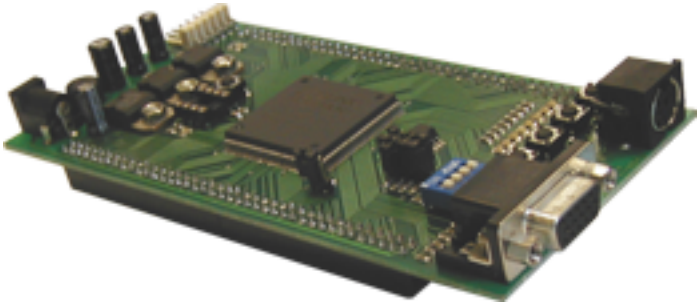
FPGA-evb-S2

The Xilinx Spartan-II Evaluation Board

Jan Pech

Board revision 1.0-A; October 18, 2001
Manual version 1.1; December 9, 2001



Original location of this document is on the FPGA-evb-S2 web site. The URL of this web site is http://fpga.f2g.net. You can find more of interesting thigs on this site; for example PCB layout files, schematics, application examples etc.

Contents

1 Overview

The FPGA-evb-S2 is an open source evaluation board based on the Xilinx Spartan-II family of FPGAs. As production data are freely available, everyone can build his own board. But, if you are not able to solder 208-pin PQFP package, you can order completed board from the CESYS GmbH. Their web site is http://www.cesys.com.

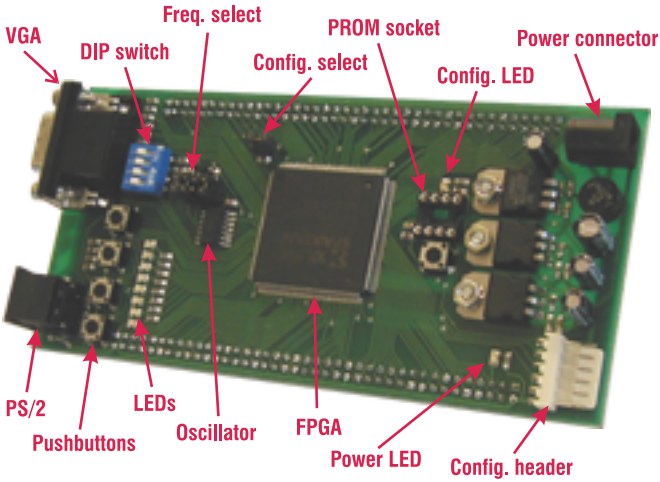
The board is intended for the 200,000 gate Xilinx's XC2S200-6PQ208 FPGA but it can be used for any Spartan-II part in 208-pin PQFP package at any speed grade. The main board contains only basic peripherals like a header programmable crystal oscillator, VGA monitor interface, LEDs, pushbuttons, DIP switches etc. Most of the FPGA's I/O signals are brought out to two 80-pin connectors for simple peripheral extensibility. The FPGA-evb-S2's peripherals can be easily expanded by an expansion board. Spartan-II FPGA can be configured through enclosed download cable or from the XC17S200A configuration PROM. The download cable is software compatible with Xilinx Parallel Cable III, so that may be used any Xilinx's software for FPGA configuration.

1.1 Brief Specification

Because of very low cost, the FPGA-evb-S2 is based on cheap two layer PCB. The board can be powered by AC or DC voltage ranging from 7 to 15 Volts. The board contains three linear voltage regulators for 2.5, 3.3 and 5 V. The board can operate standalone but it's easily expandable. The FPGA-evb-S2 board contains:

- Xilinx Spartan-II XC2S200-6PQ208C FPGA
- Programmable crystal oscillator 20–120 MHz
- Socket for the XC17S200A configuration PROM
- Two 80-pin expansion connectors
- PS/2 interface for PC keyboard or mouse
- VGA monitor output
- DIP×4 switch

- Four pushbuttons
- Eight LEDs



1.2 Applications

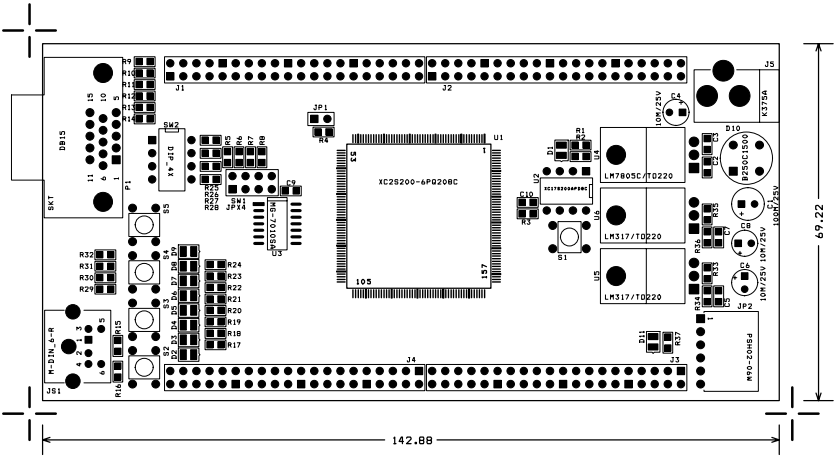
Thanks to low cost, easy extensibility, and high Spartan-II FPGA flexibility is the FPGA-evb-S2 ideal platform for next areas of utilization:

- Learning of programmable logic design
- ASIC replacement (especially in development stage)
- System on Chip (SoC) design
- Digital signal processing
- Microprocessor development

The FPGA-evb-S2 is good choice for digital design learning. The Spartan-II FPGA is supported by the Xilinx ISE WebPACK free development system so you can't spend money for development tools. This system can be freely downloaded from the Xilinx web site.

2 Circuitry Description

Complete schematics of the FPGA-evb-S2 board are in the appendix A on page 35 but schematics are not needed for common use of the board. It is fully sufficient following characterization only. For easier orientation there is depicted the FPGA-evb-S2 board assembly in following figure.



Following sections of this chapter describe–besides other things–juper settings. Symbols describing states of jumpers have following meaning:

Symbol	Meaning
○	opened jumper
●	closed jumper

2.1 Programmable Oscillator

The EPSON's MG-7010SA selectable-output PLL crystal oscillator produces the main clock signal for the FPGA. This device includes two PLLs and can output one frequency among 15 selections ranging from 20 to 120 MHz. Output of the oscillator is connected to pin global clock input GCK0 of the FPGA located at pin 80. Output frequency of the oscillator is selectable by combination of four jumpers SW1.

SW1				f
4	3	2	1	[MHz]
○	○	○	○	40
○	○	○	●	90
○	○	●	○	45
○	○	●	●	50
○	●	○	○	60
○	●	○	●	60
○	●	●	○	66.66
○	●	●	●	75
●	○	○	○	80
●	○	○	●	70
●	○	●	○	20
●	○	●	●	25
●	●	○	○	120
●	●	○	●	30
●	●	●	○	33.33
●	●	●	●	100

2.2 FPGA Configuration

There are two ways how to configure the Spartan-II FPGA on the FPGA-evb-S2 board. The first one is downloading of a configuration bitstream through the JTAG interface with the aid of the configuration cable. For this way of configuration it is possible to use any software which works with original Xilinx Parallel Cable III. The second way of configuration is downloading of a bitstream from the XC17S200A configuration PROM device. The FPGA-evb-S2 board contains the DIL socket for this device. Selection of kind of configuration is done by jumper JP1.

JP1	Configuration
○	JTAG
●	PROM

When the board is switched on the FPGA automatically passes to configuration mode. This state is indicated by the yellow LED. After successful configuration the FPGA will start normal operation and the LED will turn off. Whenever during operation it is possible to pass the

FPGA to the configuration mode by pressing the S1 pushbutton. It is not needed for configuration via the JTAG interface.

2.3 Download Cable

The JTAG download cable is fully compatible with the *Xilinx Parallel Cable III* except the JTAG connector. For configuration of an FPGA via our cable you can use any software which is able to operate with original Xilinx cable. Schematic of the download cable is in the appendix A on page 39. Documentation for the original download cable can be found at http://www.xilinx.com/support/programr/cables.htm.

Important note: If you want to configure the FPGA via the JTAG cable you have to set the *Start-Up Clock* option of your programming file to *JTAG Clock*. Without this option the FPGA wouldn't configure properly.

2.4 Power Supply

The FPGA-evb-S2 board can be powered by AC or DC voltage ranging from 7 to 15 Volts. You can use any common power supply adapter satisfying voltage requirements. The adapter have to be capable supply at least 500 mA.

The board contains three linear voltage regulators providing voltages 2.5 V, 3.3 V and 5 V. The core of the FPGA is powered by 2.5 V. All other devices are powered by 3.3 V. The only one exception is the PS/2 interface which must provide 5 V for PC keyboard or mouse. All voltages except the core voltage are brought out to expansion connectors. They can be used for powering of an expansion board.

2.5 Peripherals

This section describes complete peripheral wiring on the FPGA-evb-S2 board. This description is fully sufficient for applications development. Complete circuit schematics of whole board can be found in appendix A.

2.5.1 PS/2 Interface

The PS/2 interface is intended for connection of the PC keyboard or mouse to the FPGA-evb-S2 board. This port uses 5 V power supply so that is necessary to use appropriate FPGA I/O pins in LVTTL mode which is compatible with 5 V logic. This mode is implicit in every design system. Interface to the FPGA is described by following table.

Signal	FPGA Pin	Function
PS2-CLK	P87	PS/2 synchronization
PS2-DTA	P86	Bidirectional data

2.5.2 VGA Output

It is possible to connect a VGA monitor to the FPGA-evb-S2 board through standard 15-pin Canon connector. The monitor have to use analogue color signals and input impedance have to be 75 Ω. All the color signals are two bits wide so that it is possible to display most 64 colors.

Signal	FPGA Pin	Function
V-SYNC	P75	Vertical synchronization
H-SYNC	P74	Horizontal synchronization
BLUE0	P67	Blue color lsb
BLUE1	P68	Blue color msb
GREEN0	P69	Green clor lsb
GREEN1	P70	Green color msb
RED0	P71	Red color lsb
RED1	P73	Red color msb

2.5.3 LEDs

The board contains eight green LEDs for general use. Diodes are connected between FPGA outputs and power supply voltage. For light up a LED is needed low level on the FPGA output pin.

LED	FPGA Pin
0	P102
1	P101
2	P100
3	P99
4	P98
5	P97
6	P96
7	P95

2.5.4 DIP Switch

Quadruple DIP switch is connected between ground and FPGA pins with pull-up resistors. When a switch is switched-on then there is low level on FPGA input and vice-versa.

Switch	Signal	FPGA Pin
1	DSW0	P84
2	DSW1	P83
3	DSW2	P82
4	DSW3	P81

2.5.5 Pushbuttons

Four pushbuttons are wired alike the DIP switch. When the button is pushed the output level is low otherwise it is high. Pushbuttons are not debounced so one press can produce several glitches on output.

Button	Signal	FPGA Pin
S2	KEY0	P94
S3	KEY1	P90
S4	KEY2	P89
S5	KEY3	P88

2.5.6 Expansion Connectors

Most of the FPGA's I/O signals are brought out to two 80-pin expansion connectors for simple peripheral extensibility. Both expansion connectors are formed of two 40-pin connectors. Connection between the FPGA

and these connectors is described by following tables. The FP symbol in these tables stands for the FPGA pin number.

Connector J1					
FP	**Signal**	**Pin**		**Signal**	**FP**
	+5 V	1	2	+3,3 V	
	GND	3	4	GND	
P77	GCK1	5	6	IO63	P63
P62	IO62	7	8	IO61	P61
P60	IO60	9	10	IO59	P59
P58	IO58	11	12	IO57	P57
	+3,3 V	13	14	GND	
P49	IO49	15	16	IO48	P48
P47	IO47	17	18	IO46	P46
P45	IO45	19	20	IO44	P44
P43	IO43	21	22	IO42	P42
P41	IO41	23	24	GND	
	GND	25	26	+3,3 V	
P37	IO37	27	28	IO36	P36
P35	IO35	29	30	IO34	P34
P33	IO33	31	32	GND	
P31	IO31	33	34	IO30	P30
P29	IO29	35	36	+3,3 V	
	GND	37	38	IO27	P27
P24	IO24	39	40	IO23	P23

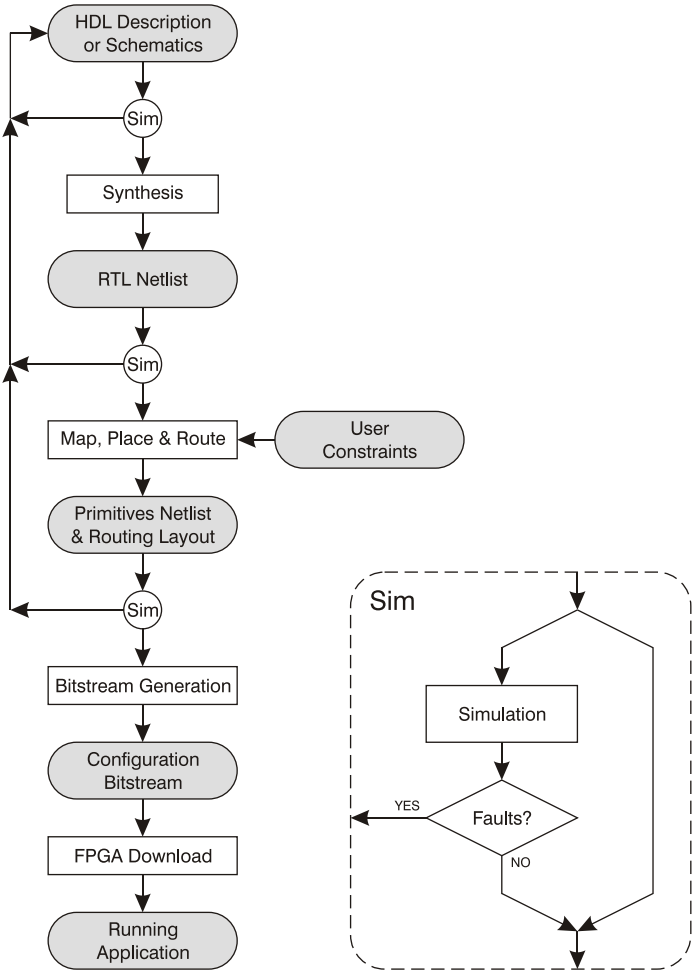
Connector J2					
FP	**Signal**	**Pin**		**Signal**	**FP**
P22	IO22	1	2	IO21	P21
P20	IO20	3	4	GND	
P18	IO18	5	6	IO17	P17
P16	IO16	7	8	IO15	P15
P14	IO14	9	10	GND	
	GND	11	12	+3,3 V	
P10	IO10	13	14	IO9	P9
P8	IO8	15	16	IO7	P7
P6	IO6	17	18	IO5	P5
P4	IO4	19	20	IO3	P3
	+3,3 V	21	22	GND	
P206	IO206	23	24	IO205	P205
P204	IO204	25	26	IO203	P203
P202	IO202	27	28	IO201	P201
P200	IO200	29	30	IO199	P199
	GND	31	32	IO195	P195
P194	IO194	33	34	IO193	P193
P192	IO192	35	36	IO191	P191
	GND	37	38	IO189	P189
P188	IO188	39	40	IO187	P187

Connector J3					
FP	**Signal**	**Pin**		**Signal**	**FP**
	GND	1	2	+3,3 V	
	GND	3	4	GCK3	P185
	GND	5	6	+3,3 V	
P182	GCK2	7	8	IO181	P181
P180	IO180	9	10	IO179	P179
P178	IO178	11	12	GND	
P176	IO176	13	14	IO175	P175
P174	IO174	15	16	IO173	P173
P172	IO172	17	18	+3,3 V	
	GND	19	20	GND	
P168	IO168	21	22	IO167	P167
P166	IO166	23	24	IO165	P165
P164	IO164	25	26	IO163	P163
P162	IO162	27	28	IO161	P161
P160	IO160	29	30	GND	
P154	IO154	31	32	+3,3 V	
P152	IO152	33	34	IO151	P151
P150	IO150	35	36	IO149	P149
P148	IO148	37	38	IO147	P147
P146	IO146	39	40	GND	

Connector J4					
FP	**Signal**	**Pin**		**Signal**	**FP**
	+3,3 V	1	2	IO142	P142
P141	IO141	3	4	IO140	P140
P139	IO139	5	6	IO138	P138
	GND	7	8	IO136	P136
P135	IO135	9	10	IO134	P134
P133	IO133	11	12	IO132	P132
	GND	13	14	+3,3 V	
P129	IO129	15	16	GND	
P127	IO127	17	18	IO126	P126
P125	IO125	19	20	GND	
P123	IO123	21	22	IO122	P122
P121	IO121	23	24	IO120	P120
P119	IO119	25	26	+3,3 V	
	GND	27	28	GND	
P115	IO115	29	30	IO114	P114
P113	IO113	31	32	IO112	P112
P111	IO111	33	34	IO110	P110
P109	IO109	35	36	IO108	P108
	GND	37	38	GND	
	+3,3 V	39	40	+5 V	

3 Applications Development

Digital design for an FPGA consists of several steps. The design flow is very similar to digital IC design. The design flow includes simulations and synthesis operations like a synthesis, map and place & route.



As the figure illustrates, it's possible to perform a logic simulation on every design level. Correct simulation can save a lot of time. Good practice is to simulate at least the top level design files. The top-level simulation is called functional. For larger design is suitable to simulate post-place & route netlist. This simulation is called timing.

When you design some FPGA application you will usually do following several steps:

1. You enter your logic description using an HDL (hardware description language) such as VHDL or Verilog. You can also use schematic description but it's complicated for complex designs.

2. Now you can use logic synthesis tool for translating your description to a netlist. The netlist is description of generic logic primitives like multiplexers, registers, gates, etc. The netlist also contains information about connection of the primitives.

3. After the synthesis is completed you can use implementation tools. These tools consist of a map tool and a place & route tool. The map tool converts an RTL netlist from synthesis to another netlist. This netlist contains FPGA primitives only. The place & route tool interconnects these primitives using FPGA routing resources.

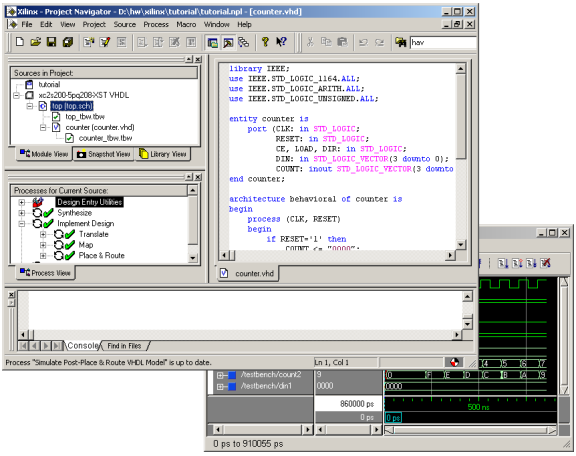
4. To allow configuration of an FPGA it is necessary to create a bitstream. The bitstream is a file that describes especially states of electronic switches of an FPGA.

5. The bitstream can be downloaded into an FPGA chip. After the downloading the FPGA will perform the operation specified by description from first point.

3.1 Development Software

Since the FPGA-evb-S2 is based on Spartan-II FPGA, it is possible to use free Xilinx *ISE WebPACK* software for your designs. Naturally, you can use any other system of the ISE family like a Foundation, Alliance or BaseX but their cost is very high. You can use any third-party development system but you have to use some of ISE products for at least place & route of synthesized designs.

The WebPACK development system allows you to enter your designs using VHDL, Verilog or schematics. You can include schematics into your HDL designs but you can't mix Verilog and VHDL in one design. As an addition to the WebPACK there is also available limited edition of ModelTech *ModelSim Xilinx Edition* HDL simulator. This version is fully functional but for large designs it's slower than the full version.



The ISE WebPACK development environment and the ModelSim XE HDL simulator can be downloaded from http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack. You have to register before you can download the software. Since the WebPACK is compatible with the rest of Xilinx ISE development tools you can use most of the ISE documentation. The documentation for version 4.1 can be found at http://toolbox.xilinx.com/docsan/xilinx4/index.htm.

3.2 Design Flow

Typical steps of FPGA design flow were listed above. Following subsections closely describe the design flow steps. This description is referenced to the FPGA-evb-S2 evaluation board and the Xilinx WebPACK development system.

3.2.1 Design Entry

This document can't supply an HDL textbook. You can find some VHDL examples in chapter 4. For correct understanding it is necessary to know at least primer on VHDL. You can find a list of many books on VHDL at the http://www.vhdl.org/comp.lang.vhdl/ page. There is also a lot of additional useful information relative to VHDL at this page.

3.2.2 Synthesis, Map, Place & Route

This part of design implementation is described by WebPACK tutorials and ISE documentation. You can find the WebPACK tutorial at http://www.cesys.com web site. Another tutorial and complete ISE documentation can be found at http://toolbox.xilinx.com/docsan/xilinx4/ web page. This tutorial is intended for whole ISE family.

Implementation tools use not only HDL or schematic design description. Important part of the design is *user constraints*. They are usually listed in UCF file. These constraints assign FPGA pins to design I/O signals. You can also specify timing requirements, memory initialization values, etc. in this file. The complete user constraints file for the FPGA-evb-S2 is listed in appendix B on page 40. The file contains whole set of used FPGA I/O pins except expansion connectors pins. For your designs copy to your *.ucf file only a part of the file containig pins you use.

3.2.3 Simulation

The ModelSim HDL simulator comes with a tutorial. Basics of simulation can be also found in WebPACK tutorials. In the WebPACK you can use the waveform editor for generation of simulation stimuli. This way is suitable for simulation of smaller designs. Of course, you can write conventional HDL testbenches. Good web site about writing VHDL testbenches is http://www.i2.i-2000.com/ stefan/vcourse/html/.

3.2.4 Bitstream Generation

The generation of a bitstream is business of a development system. However, it is necessary to properly set up the *Start-Up Clock* property of *Startup Options*. Setup of this property depends on an FPGA configuration method:

- For configuration via a JTAG download cable you have to set this property to *JTAG Clock*.
- For configuration from a serial PROM you have to set this property to *CCLK*.

When you want to configure the FPGA using serial PROM, you have to create a PROM file after the bitstream generation. Created PROM file may be burnt into an XC17S200A serial PROM.

3.2.5 FPGA Programming

Configuration of the FPGA via the JTAG cable is easy. Before configuration of the device you have to have the generated bitstream. Then you have to plug the JTAG download cable into PC parallel port. Next you can connect the other connector of the cable into the FPGA-evb-S2 JTAG port. Now you can switch on the power supply and connect it to the FPGA-evb-S2. The FPGA configuration method depends on state of the JP1 jumper. It is described on page 4.

Now you can run the programming tool *iMPACT* simply by double clicking on the *Configure Device* item in processes window of the ISE development system. If you have correctly interconnected the PC, download cable and the FPGA-evb-S2, the iMPACT tool automaticaly recognizes the XC2S200 device. The programming tool also assigns your bitstream file to the device.

By now you can click by right mouse button on the XC2S200 device symbol. From a pull-down menu select the *Program...* item. New dialog box appears on the screen. You may select whether you want to verify the bitstream after programming or not. After pressing the *OK* button the iMPACT tool uploads your bitstream into the FPGA.

4 Sample Designs

These sample designs can be found on the support web site or on the CD-ROM that comes with the FPGA-evb-S2 board. Designs were created using Xilinx ISE WebPACK 4.1. If you use another design system you can use VHDL and UCF files only.

4.1 Running Light

This example shows how to use LEDs, pushbuttons and DIP-switch. The example implements a "running light." Only one LED lights at the same time. When you press the S5 button, the light runs over all LEDs. The direction of the light movement is selectable by the first DIP-switch. The button S2 acts as asynchronous reset.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity leds is
  port (rstn: in  std_logic ;                              -- active low reset
        clk : in  std_logic ;                              -- system clock
        u_d : in  std_logic ;                              -- direction select
        step: in  std_logic ;                              -- single step button
        led : out std_logic_vector (7 downto 0));          -- LED outputs
end leds;


architecture behavioral of leds is

  signal cnt: std_logic_vector (23 downto 0);
  signal ena: std_logic ;
  signal dir: std_logic ;

begin

  -- Switch and pushbutton synchronization to clock
  SYNC: process(clk, rstn)
  begin
    if (rstn = '0') then
      dir <= '0';
      ena <= '0';
    elsif (clk'event and clk='1') then
```

```
        dir <= u_d;
        ena <= not step;
      end if;
    end process;

    -- Synchronous up/down counter with enable and asynchronous reset
    COUNTER: process(clk, rstn, ena, dir)
    begin
      if (rstn = '0') then
        cnt <= (others => '0');
      elsif (clk'event and clk='1') then
        if (ena = '1') then
          if (dir = '1') then
            cnt <= cnt + 1;
          else
            cnt <= cnt - 1;
          end if;
        end if;
      end if;
    end process;

    -- Binary to 1 of 8 decoder with asynchronous reset, active low output
    DECODER: process(clk, rstn, cnt)
    begin
      if (rstn = '0') then
        led <= (others => '1');
      elsif (clk'event and clk='1') then
        case cnt(23 downto 21) is
          when "000" => led <= "11111110";
          when "001" => led <= "11111101";
          when "010" => led <= "11111011";
          when "011" => led <= "11110111";
          when "100" => led <= "11101111";
          when "101" => led <= "11011111";
          when "110" => led <= "10111111";
          when "111" => led <= "01111111";
          when others => NULL;
        end case;
      end if;
    end process;

  end behavioral;
```

The design consists of three processes. The firts one is the *SYNC* process. This process synchronizes external asynchronous signals to the system clock. The second process is the *COUNTER*. This process imple-

ments a synchronous counter that can count up and down. The counter has an asynchronous reset and enable inputs. The third process *DECODER* forms a binary to "1 of 8" decoder.

When you want to make a LED light, you have to set corresponding FPGA output pin to low logic level. That's the reason why the decoder has negative (active low) outputs.

Quiescent, outputs of the DIP-switch and pushbuttons are on high logic level. When you press the button or switch the DIP to on state, outputs pass to low level. This means you have to use negative input logic for buttons and switch.

Good design practice shown by this design is to synchronize all asynchronous external signals to the system clock. If you do this you prevent from undesirable behavior of your design. Asynchronous external signals may cause for example state machines to jump to undefined states etc.

4.2 Keyboard

This examples shows how to handle a PC keyboard. You have to know at least primer how the keyboard works and how it communicates with a host system.

The keyboard simply sends *scan codes* to your computer or another host system. The scan code tells your system what key you have pressed or released. The keyboard communicates with your system via bidirectional synchronous serial interface. The PS/2 interface is described on page 6.

4.2.1 Keyboard to Host Communication

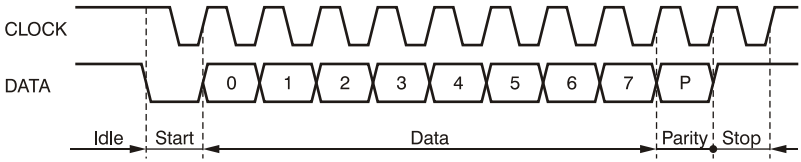
When you press some key, the keyboard will send scan code of the key. If you are still holding the key down for longer time than a typematic delay, additional scan code will be sent. This will be repeated until you release the key. When the key is released, the keyboard will send *F0* code followed by the scan code of released key.

Some keys produce more than one scan code only. Most of extended scan codes is formed of *E0* code followed by the scan code of the key. But some keys (for example the Pause/Break key) sends longer sequences. When an extended key is released, the keyboard will usually sends *E0* code followed by *F0* code and extended code(s) of the key.

Scan codes of a generic AT keyboard are listed in appendix C on page 42. The appendix contains scan codes for both key press and key release.

Protocol for the keyboard to host communication is depicted on following figure. Both signals are driven by a keyboard in this mode. Frequency of the clock signal is typically 20 to 30 kHz. The protocol has following properties:

- Start bit (logic 0)
- 8 data bits representing the scan code (LSB first)
- Parity bit (odd parity – data bits plus the parity bit are an odd number of ones)
- Stop bit (logic 1)



Besides scan codes, the keyboard can also send another messages. These messages are mostly replies to host commands (see following subsection). Keyboard to host commands can be following:

00: Key detection error / keyboard buffer overflow

83,AB: Keyboard ID

AA: Self-test passed

EE: Echo – sent to host after receiveing "echo" command

FA: Acknowledge (ACK)

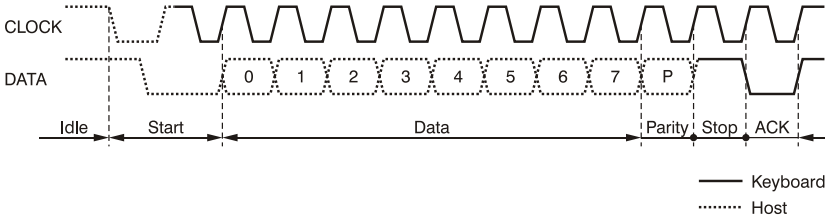
FC: Self-test failed

FE: Resend – host responds by re-transmitting the last byte

FF: Key detection error / keyboard buffer overflow

4.2.2 Host to Keyboard Communication

This direction of communication is used for setting status LEDs, typematic delay etc. The protocol for the host to keyboard communication is depicted on following figure. The clock signal is driven mostly by the keyboard. The data signal is driven mostly by the host system. So that it's needed to use I/O pins as open collectors.



Host to keyboard commands can be following:

ED: Turns on/off LEDs – keyboard replies with ACK (FA) and waits for another byte to be sent. Next byte sent determines the state of the LEDs (Bits 0-2 correspond to LEDs 1,2,3. Bits 3-7 should always be 0.)

EE: Echo – Keyboard should respond with Echo (EE)

F0: Set Scan Code Set – Responds with ACK (FA) and waits for another byte to be sent. Next byte sent will be either 01, 02, or 03 (corresponding to scan code sets 1, 2, and 3.) If 00 is sent (instead of 01, 02, or 03) keyboard will respond with ACK (FA) followed by the current scan code set (again, 01, 02, or 03).

F2: Get ID – Responds with ACK (FA) followed by an ID (A3, AB). This also enables scanning.

F3: Set repeat rate – Keyboard replies with ACK (FA) and waits for another byte to be sent. Next byte sent will determine the typematic repeat rate for the keybaord. *SEE NOTE BELOW* After this byte is sent, keyboard responds with another ACK (FA)

F4: Enable keyboard – Clear the buffer and start scanning for data; Replies with ACK (FA)

F5: Disable keyboard – Disables scanning and replies with ACK (FA). Does not affect indicator LEDs.

F6: Restore default values. Does not affect indicator LEDs.

F7: Set all keys typematic. Responds with ACK (FA)

F8: Set all keys make/break. Responds with ACK (FA)

F9: Set all keys make. Responds with ACK (FA)

FA: Set all keys typematic/make/break. Responds with ACK (FA)

FB: Set key type typematic.

FC: Set key type make/break.

FD: Set key type make.

FE: Resend – Keyboard responds by re-transmitting the last command it sent.

FF: Reset – Resets the keyboard.

4.2.3 Further Information

Entire description of an AT keyboard exceeds scope of this document. Especially, if you want use host to keyboard communication, you have to learn more. You can find more detailed information about AT keyboards at following web pages:

- http://govschl.ndsu.nodak.edu/~achapwes/PICmicro/keyboard/atkeyboard.html
- http://www.beyondlogic.org/keyboard/keybrd.htm

4.2.4 The Design

This example shows the keyboard to host communication only. The design receives a scan code from a keyboard and displays it on LEDs.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ps2 is
  port (resetn: in  std_logic ;                          -- active low reset
        clock : in  std_logic ;                          -- system clock
        ps2_clk: in  std_logic ;                         -- PS/2 clock line
        ps2_dta: in  std_logic ;                         -- PS/2 data line
        leds  : out std_logic_vector (7 downto 0));      -- LED outputs
end ps2;


architecture behavioral of ps2 is

  type state_type is (IDLE, START, DATA, PARITY); -- FSM states

  signal ps2_dv: std_logic ;                             -- PS/2 data valid
  signal prv_ps2_clk, act_ps2_clk : std_logic ;          -- auxiliary signals
  signal recdata: std_logic_vector (7 downto 0);         -- read data
  signal shift : std_logic ;                             -- enable for shift reg.
  signal n_shift : std_logic ;                           -- auxiliary signal
  signal latch: std_logic ;                              -- latch read data
  signal n_latch: std_logic ;                            -- auxiliary signal
  signal err : std_logic ;                               -- parity or stop error
  signal n_err : std_logic ;                             -- auxiliary signal
  signal parset: std_logic ;                             -- preset for parity check
  signal n_parset: std_logic ;                           -- auxiliary signal
  signal c_state , n_state : state_type ;                -- current & next states
  signal cntval: std_logic_vector (2 downto 0);          -- counter of data bits
  signal zero: std_logic ;                               -- counter is zero
  signal parbit: std_logic ;                             -- odd parity of data

begin

  -- Provides a one-shot pulse after every falling edge of PS/2 clock
  PS_CLK_SYNC: process(clock, resetn)
  begin
    if (resetn = '0') then
      prv_ps2_clk <= '1';
```

```
      act_ps2_clk <= '1';
    elsif (clock'event and clock = '1') then
      act_ps2_clk <= ps2_clk;
      prv_ps2_clk <= act_ps2_clk;
    end if;
  end process;

  ps2_dv <= (not act_ps2_clk) and prv_ps2_clk;

  -- Serial input, parallel output shift register
  SIPO: process(clock, resetn)
  begin
    if (resetn = '0') then
      recdata <= (others => '0');
    elsif (clock'event and clock = '1') then
      if (shift = '1') then
        recdata <= ps2_dta & recdata(7 downto 1);
      end if;
    end if;
  end process;

  -- Counter of data bits
  COUNT8: process(resetn, clock)
  begin
    if (resetn = '0') then
      cntval <= (others => '0');
    elsif (clock'event and clock = '1') then
      if (shift = '1') then
        cntval <= cntval + 1;
      end if;
    end if;
  end process;

  zero <= not (cntval(0) or cntval(1) or cntval(2));

  -- Parity check of received data
  PARITY_CHECK: process(clock, parset)
  begin
    if (parset = '1') then
      parbit <= '1';
    elsif (clock'event and clock = '1') then
      if (shift = '1' and ps2_dta = '1') then
        parbit <= not parbit;
      end if;
    end if;
  end process;
```

```
-- Synchronous process of control state machine
FSM_SYNC: process(clock, resetn)
begin
  if (resetn = '0') then
    c_state <= IDLE;
    shift <= '0';
    latch <= '0';
    err <= '0';
    parset <= '1';
  elsif (clock'event and clock = '1') then
    c_state <= n_state;
    shift <= n_shift;
    latch <= n_latch;
    err <= n_err;
    parset <= n_parset;
  end if;
end process;

-- Combinatorial process of control state machine
FSM_COMB: process(c_state, ps2_dv, ps2_dta, zero)
begin
  -- default values
  n_shift <= '0';
  n_latch <= '0';
  n_err <= '0';
  n_parset <= '0';
  case c_state is
    -- wait to receive data
    when IDLE => if ((ps2_dv and (not ps2_dta)) = '1') then
                   n_state <= START;
                   n_parset <= '1';
                 else
                   n_state <= IDLE;
                 end if;
    -- receive first data bit
    when START => if (ps2_dv = '0') then
                   n_state <= START;
                 else
                   n_state <= DATA;
                   n_shift <= '1';
                 end if;
    -- receive remaining data bits and parity
    when DATA => if (ps2_dv = '0') then
                   n_state <= DATA;
                 elsif (zero = '0') then
                   n_state <= DATA;
                   n_shift <= '1';
```

```
                    else
                      n_state <= PARITY;
                      if (parbit /= ps2_dta) then
                        n_err <= '1';
                      end if;
                    end if;
        -- receive stop bit
        when PARITY => if (ps2_dv = '0') then
                      n_state <= PARITY;
                    else
                      n_state <= IDLE;
                      n_latch <= '1';
                      n_err <= not ps2_dta;
                    end if;
      end case;
    end process;

    -- Output latch
    LED_OUTPUTS: process(resetn, clock)
    begin
      if (resetn = '0') then
        leds <= (others => '1');
      elsif (clock'event and clock = '1') then
        if (err = '1') then
          leds <= (others => '1');
        elsif (latch = '1') then
          leds <= not recdata;
        end if;
      end if;
    end process;

  end behavioral;
```

This design looks quite complicated but it's only appearance. The design illustrates using of regular digital design blocks like a state machine, latches, shift register etc.

The design consists of seven processes:

1. The *PS_CLK_SYNC* process synchronizes the PS/2 clock signal to global system clock. It provides "one-shot" pulse after every falling edge of the PS/2 clock. Length of the pulse is equal to one system clock period.

2. The *SIPO* process forms a shift register. The register has serial input, parallel output and it's eigth bits wide. The shift register is

used for receiving serial data from the keyboard.

3. The *COUNT8* process is three bit synchronous counter. The counter counts the number of data bits in every keyboard code.

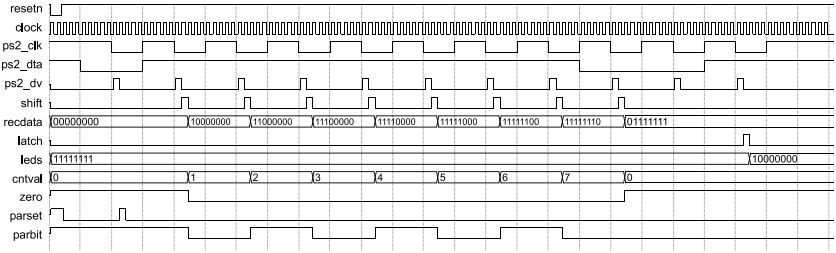
4. The *PARITY_CHECK* process computes the odd parity of received data. It has to be initialized before operation.

5. The *FSM_SYNC* process forms a synchronous part of a finite state machine. This process provides some initialisatin and synchronous transitions of the FSM into next states.

6. The *FSM_COMB* process is the second part of FSM. It's the combinatorial process which provides right state transitions and output signals setting.

7. The *LED_OUTPUTS* process provides latching of received data into output register. The data are inverted before latching.

Main part of the design is the state machine. It controls behavior of the rest of system. You can see that there are not defined values for particular states in the design. It's leaved in a synthesis tool to use optimum states encoding. You can see the function of the design on following figure.



The figure was obtained from simulation of the design in ModelSim. You can see waveforms of all important signals here.

4.3 VGA Signal Generator

You have to know something about VGA signals, before you can use the VGA output. However, these informations are poorly available. So, following sections describe essentials of VGA signals timing, levels etc.

4.3.1 VGA Signals

The VGA interface is formed by five signals – horizontal and vertical synchronization and three color signals (red, green, blue).

Synchronization signals are TTL compatible. Synchronization pulses may have positive or negative polarity. Negative synchronization signal is in logic 1 state and only pulses are logic 0. Positive synchronization signal is in logic 0 and pulses are logic 1. The right synchronization polarity depends on monitor mode.

Color signals are analogue; intensity of a color is relative to the voltage of the signal. Individual color signals are generated from two logic signals by simple converter. The converter is formed of two resistors. Input impedance of a monitor has to be 75 Ω for proper function of the converter. The converter is dual voltage divider only. Dependence of converter output voltage on status of input logic signals is described by following table.

Color		Output
bit 1	**bit 0**	**[V]**
0	0	0.000
0	1	0.272
1	0	0.561
1	1	0.833

All common monitors operates with input color voltages from 0 to 0.7 Volts. The converter generates a litle bit greater voltage for 1,1 inputs. The difference is very small; the converter can't destroy your monitor. The only one effect is that the difference in color between states 1,0 and 1,1 is less than other ones.

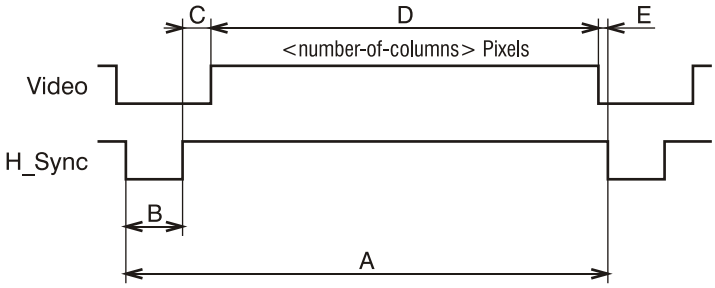
4.3.2 Signal Timing

The most important part of generation of any video signal is the signal timing. There is described timing of standard IBM VGA mode in this

section. You can find timing informations about some other modes in the appendix D on page 45.

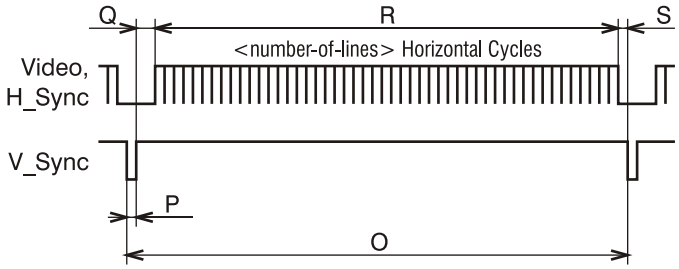
The standard IBM VGA has resolution of 640×480 pixels. The frame refresh rate is 59.940 Hz and the pixel rate is 25.175 MHz. Both synchronization signals have negative polarity. Timing for this mode is described below every following waveform.

The waveform of a single line of the VGA signal is depicted on following figure. Synchronization pulses on the figure have negative polarity.



Symbol	A	B	C	D	E
Time [μs]	31.778	3.817	1.907	25.422	0.636

Vertical synchronization timing is described by following figure and table. Synchronization pulses on the figure have negative polarity again.



Symbol	O	P	Q	R	S
Time [ms]	16.683	0.064	1.048	15.253	0.318

4.3.3 The Design

This example displays horizontal or vertical color stripes on a VGA monitor. The orientation of stripes is selectable by the first DIP switch. The example operates with resolution 640×480 pixels.

The standard IBM 640×480 video mode uses pixel rate 25.175 MHz. The oscillator on the FPGA-evb-S2 makes possible to set 25 MHz only. The example operates at this pixel rate so that the frame refresh rate will be 59.524 Hz. It's minimal difference compared to the original IBM video mode.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity vga is
  port (resetn: in  std_logic ;                          -- active low reset
        clock : in  std_logic ;                          -- system clock (25 MHz)
        orient : in  std_logic ;                         -- orientation of stripes
        hsync: out std_logic ;                           -- horizontal sync
        vsync: out std_logic ;                           -- vertical sync
        pixel : out std_logic_vector (5 downto 0));      -- pixel data
end vga;


architecture behavioral of vga is

  -- Horizontal timing constants
  constant H_PIXELS:  integer := 640;     -- number of pixels per line
  constant H_FRONTPORCH: integer := 16; -- gap before sync pulse
  constant H_SYNCTIME: integer := 96;   -- width of sync pulse
  constant H_BACKPORCH: integer := 48; -- gap after sync pulse
  constant H_SYNCSTART: integer := H_PIXELS + H_FRONTPORCH;
  constant H_SYNCEND: integer := H_SYNCSTART + H_SYNCTIME;
  constant H_PERIOD: integer := H_SYNCEND + H_BACKPORCH;
  -- Vertical timing constants
  constant V_LINES:   integer := 480;     -- number of lines per frame
  constant V_FRONTPORCH: integer := 10; -- gap before sync pulse
  constant V_SYNCTIME: integer := 2;      -- width of sync pulse
  constant V_BACKPORCH: integer := 33; -- gap after sync pulse
  constant V_SYNCSTART: integer := V_LINES + V_FRONTPORCH;
  constant V_SYNCEND: integer := V_SYNCSTART + V_SYNCTIME;
  constant V_PERIOD: integer := V_SYNCEND + V_BACKPORCH;
```

```
  signal hcnt: std_logic_vector (9 downto 0);   -- horizontal counter of pixels
  signal vcnt: std_logic_vector (9 downto 0);   -- vertical counter of lines
  signal hsyncint: std_logic ;                  -- internal horizontal sync
  signal enable: std_logic ;                    -- output enable for pixel data

begin

  -- Horizontal counter of pixels
  HORIZONTAL_COUNTER: process(clock, resetn)
  begin
    if (resetn = '0') then
      hcnt <= (others => '0');
    elsif (clock'event and clock = '1') then
      if hcnt < H_PERIOD then
        hcnt <= hcnt + 1;
      else
        hcnt <= (others => '0');
      end if;
    end if;
  end process;

  -- Internal horizontal synchronization pulse generation (negative polarity)
  HORIZONTAL_SYNC: process(clock, resetn)
  begin
    if (resetn = '0') then
      hsyncint <= '1';
    elsif (clock'event and clock = '1') then
      if (hcnt >= H_SYNCSTART and hcnt < H_SYNCEND) then
        hsyncint <= '0';
      else
        hsyncint <= '1';
      end if;
    end if;
  end process;

  -- Horizontal synchronization output
  hsync <= hsyncint;

  -- Vertical counter of lines
  VERTICAL_COUNTER: process(hsyncint, resetn)
  begin
    if (resetn = '0') then
      vcnt <= (others => '0');
    elsif (hsyncint'event and hsyncint = '1') then
      if vcnt < V_PERIOD then
        vcnt <= vcnt + 1;
      else
```

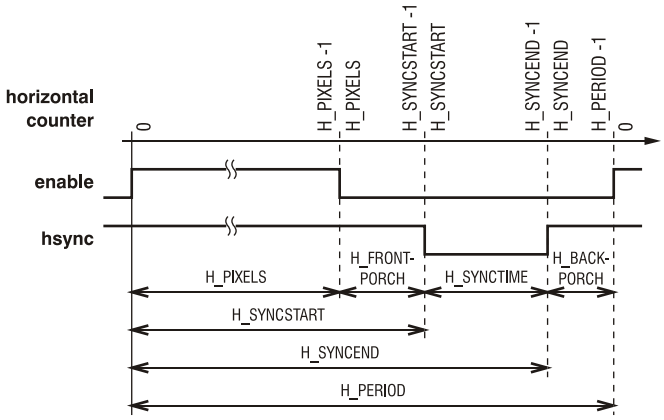
```
        vcnt <= (others => '0');
      end if;
    end if;
  end process;

  -- Vertical synchronization pulse generation (negative polarity)
  VERTICAL_SYNC: process(hsyncint, resetn)
  begin
    if (resetn = '0') then
      vsync <= '1';
    elsif (hsyncint'event and hsyncint = '1') then
      if (vcnt >= V_SYNCSTART and vcnt < V_SYNCEND) then
        vsync <= '0';
      else
        vsync <= '1';
      end if;
    end if;
  end process;

  -- Enabling of color outputs
  OUTPUT_ENABLE: process(clock)
  begin
    if (clock'event and clock = '1') then
      if (hcnt >= H_PIXELS or vcnt >= V_LINES) then
        enable <= '0';
      else
        enable <= '1';
      end if;
    end if;
  end process;

  -- Output image generation (horizontal or vertical color stripes)
  IMAGE: process(enable, orient, hcnt, vcnt)
  begin
    if (enable = '0') then
      pixel <= (others => '0');
    elsif (orient = '1') then
      pixel <= hcnt(7 downto 2);
    else
      pixel <= vcnt(7 downto 2);
    end if;
  end process;

end behavioral;
```

The design consists of two counters, two comparators, output enable

generator and a simple color stripes generator. Every block of the design is formed by a single process.

1. The *HORIZONTAL_COUNTER* process counts right number of system clock periods per line.
2. The *HORIZONTAL_SYNC* process generates the negative polarity horizontal synchronization signal.
3. The *VERTICAL_COUNTER* process counts right number of horizontal synchronization pulses per frame. The process expects negative polarity sync signal.
4. The *VERTICAL_SYNC* process generates the negative polarity vertical synchronization signal.
5. The *OUTPUT_ENABLE* signal produces output enable signal for pixel data output. It's correct to output black signal when a cathode ray beam retraces.
6. The *IMAGE* process generates the pixel color data.

Following figure illustrates function of horizontal counter and synchronization generator. Behavior of vertical ones is very similar.



5 Support

The official FPGA-evb-S2 web site is http://fpga.f2g.net. You can find there documentation, application examples, links etc. If you won't be able to find a solution of your problem on this web site, try to contact me. I prefer contact by e-mail but you can try ICQ or phone too.

Not everyone is able to build the board himself. Not everyone has enough of free time to do it. So, you can order completed one from the CESYS GmbH (http://www.cesys.com).

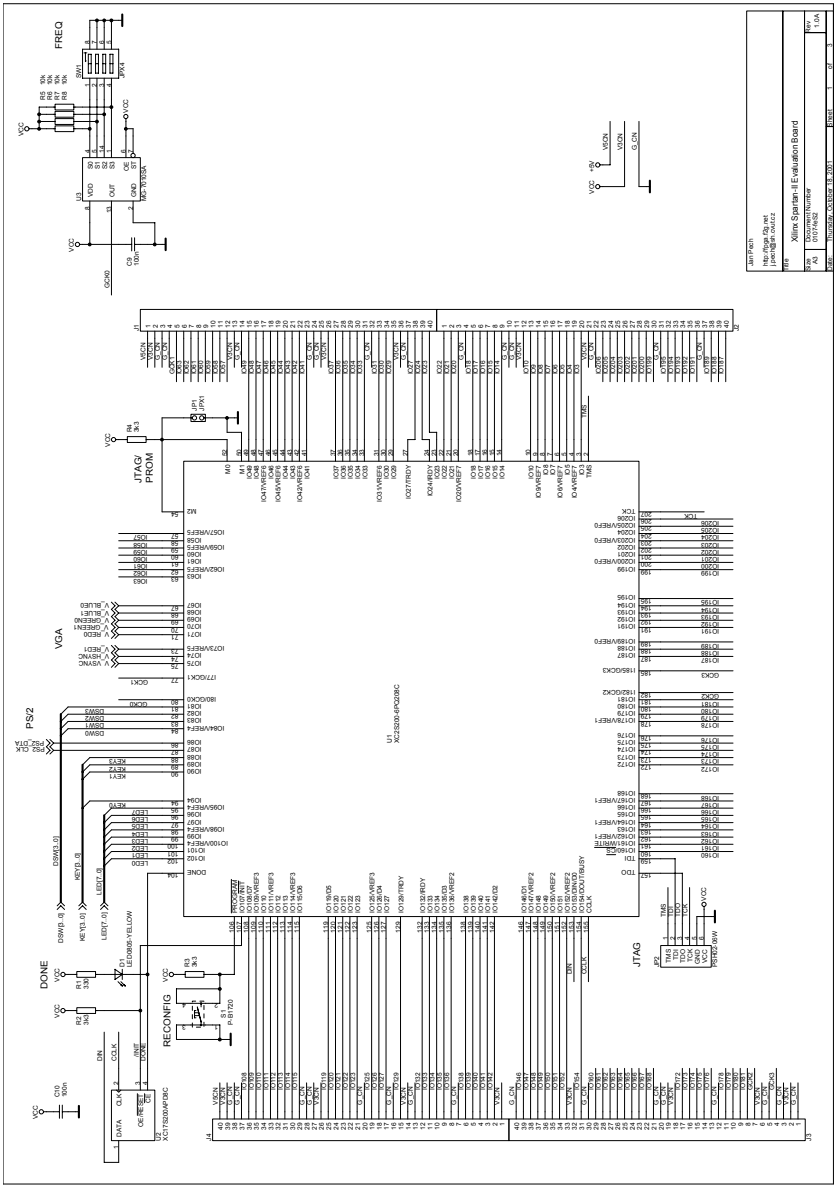
Jan Pech
j.pech@sh.cvut.cz
+420 723 760802
ICQ: 56 431 283

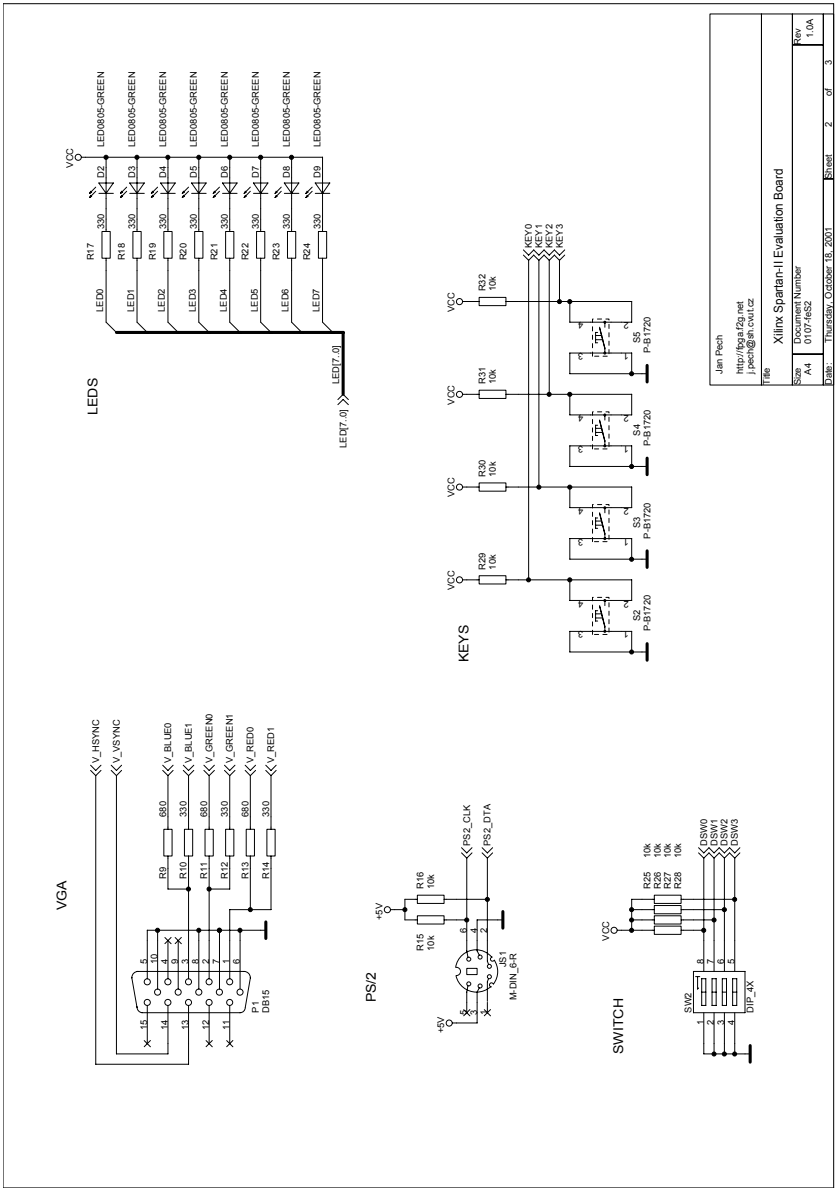
A Schematics

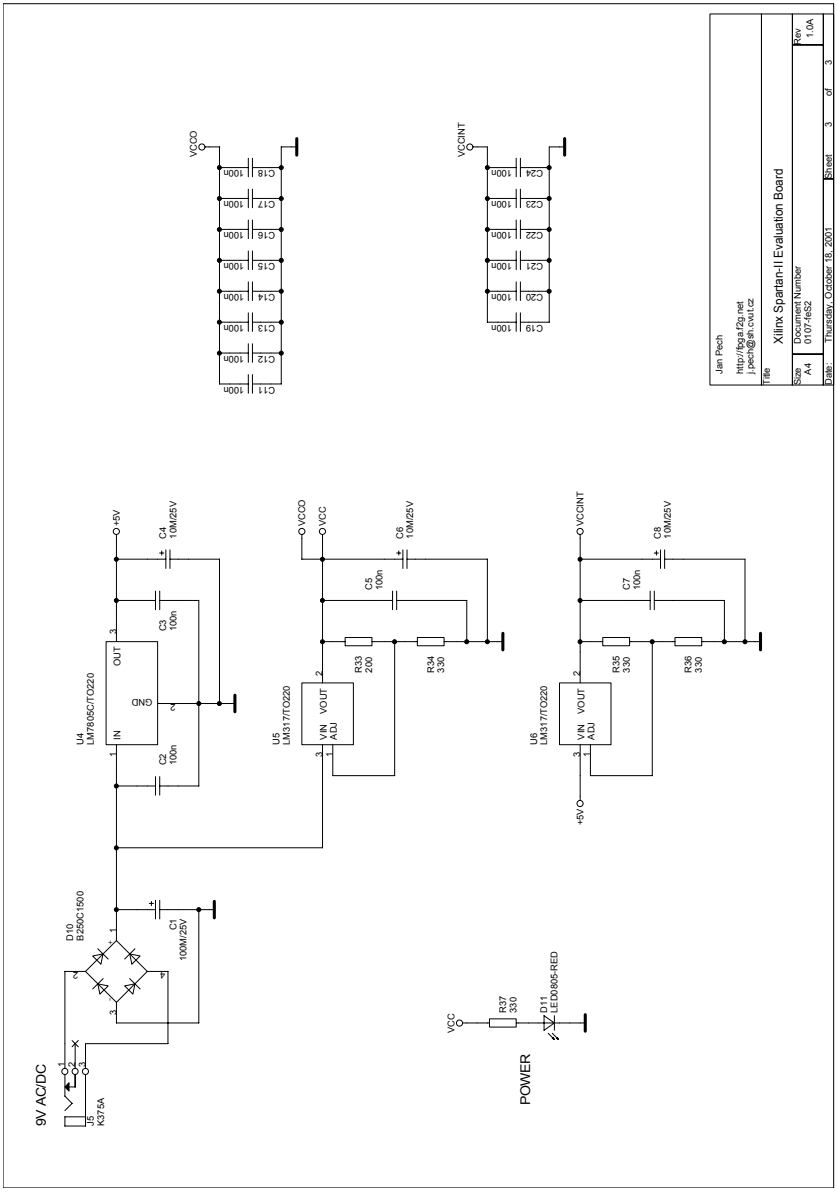
Following pages depict complete schematics of the FPGA-evb-S2 board and the JTAG download cable. Schematics of the FPGA-evb-S2 consist of three sheets:

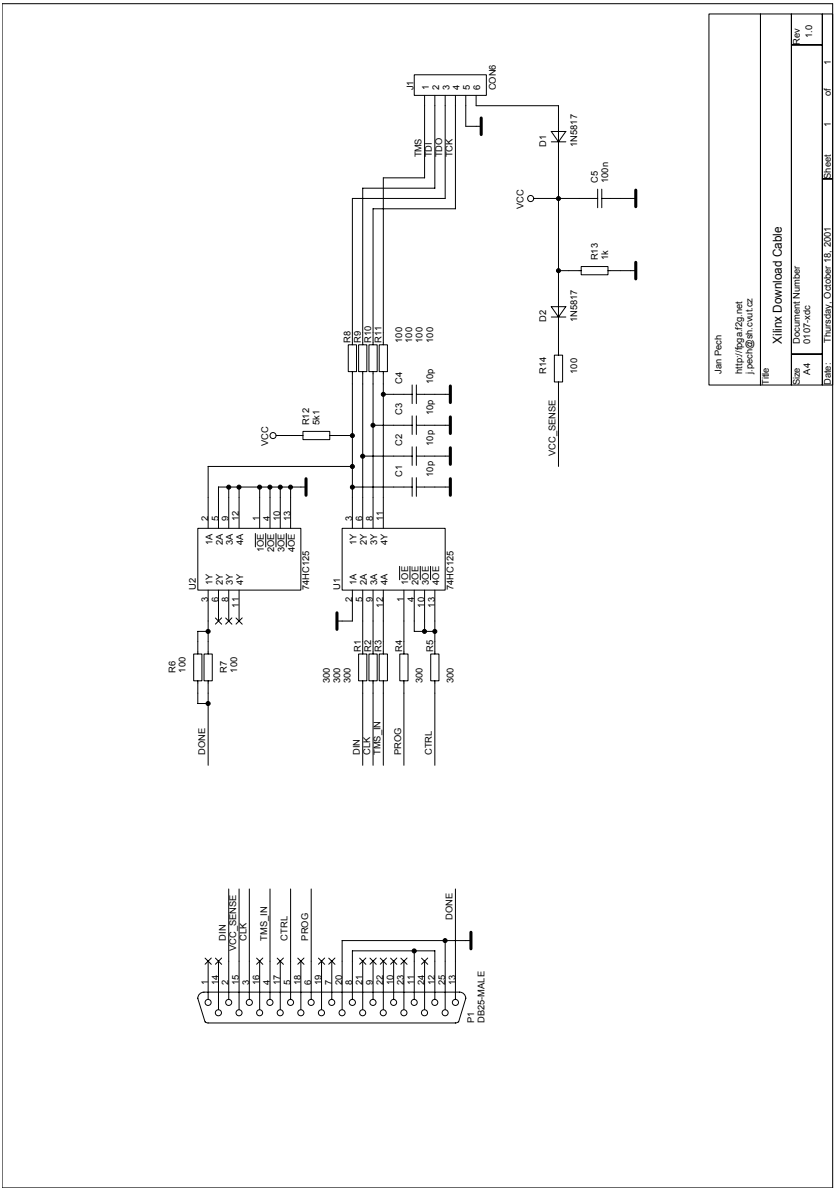
- FPGA with fundamental circuitry on page 36.
- Peripherals on page 37.
- Power supply on page 38.

The schematic of the JTAG download cable on page 39 is identical with the schematic of the Xilinx Parallel Cable III. The only exception is the JTAG connector header. This cable uses six-pin connector only.


Jan Pech
http://fpga.net
jan@fpga.net
Xilinx Spartan-II Evaluation Board
Document Number
Rev
Monday, October 15, 2001
Sheet
FREQ
JTAG/ PROM
VGA
PS/2
DONE
RECONFIG
JTAG


LEDS
VCC
LED0805-GREEN
R17 330
R18 330
R19 330
R20 330
R21 330
R22 330
R23 330
R24 330
D2
D3
D4
D5
D6
D7
D8
D9
LED0
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED[7..0]
LED[7..0]
KEYS
VCC
R32 10k
R31 10k
R30 10k
R29 10k
KEY0
KEY1
KEY2
KEY3
S5
S4
S3
S2
P-B1720
VGA
V_HSYNC
V_VSYNC
V_BLUE0
V_BLUE1
V_GREEN0
V_GREEN1
V_RED0
V_RED1
680
330
680
330
680
330
R9
R10
R11
R12
R13
R14
P1
DB15
PS/2
+5V
R16 10k
R15 10k
PS2_CLK
PS2_DTA
JS1
M-DIN_6-R
SWITCH
VCC
R25 10k
R26 10k
R27 10k
R28 10k
DSW0
DSW1
DSW2
DSW3
SW2
DIP_4X
Jan Pech
http://fpga42g.net
j.pech@sh.cvut.cz
Title
Xilinx Spartan-II Evaluation Board
Size
A4
Document Number
0107-feS2
Rev
1.0A
Date:
Thursday, October 18, 2001
Sheet
2
of
3


Jan Pech
http://fpga42g.net
j.pech@sh.cvut.cz
Title
Xilinx Spartan-II Evaluation Board
Size
A4
Document Number
0107-feS2
Rev
1.0A
Date:
Thursday, October 18, 2001
Sheet
3
of
3
VCCO
C18 100n
C17 100n
C16 100n
C15 100n
C14 100n
C13 100n
C12 100n
C11 100n
VCCINT
C24 100n
C23 100n
C22 100n
C21 100n
C20 100n
C19 100n
9V AC/DC
J5
K375A
D10
B250C1500
C1
100M/25V
+5V
C4
10M/25V
C3
100n
C2
100n
U4
LM7805C/TO220
IN
GND
OUT
VCCO
VCC
C6
10M/25V
C5
100n
R33
200
R34
330
U5
LM317/TO220
VIN
ADJ
VOUT
VCCINT
C8
10M/25V
C7
100n
R35
330
R36
330
U6
LM317/TO220
VIN
ADJ
VOUT
+5V
POWER
VCC
R37
330
D11
LED0805-RED


Jan Pech
http://fpga42g.net
j.pech@sh.cvut.cz
Title
Xilinx Download Cable
Size
A4
Document Number
0107.xdc
Rev
1.0
Date:
Thursday, October 18, 2001
Sheet
1
of
1
J1
CON6
TMS
TDI
TDO
TCK
D1
1N5817
D2
1N5817
C5
100n
VCC
R13
1k
R14
100
VCC_SENSE
R8
R9
R10
R11
100
100
100
100
C4
C3
C2
C1
10p
10p
10p
10p
R12
5k1
U1
74HC125
U2
74HC125
R6
100
R7
100
R1
R2
R3
R4
R5
300
300
300
300
300
DONE
DIN
CLK
TMS_IN
PROG
CTRL
P1
DB25-MALE
VCC_SENSE

B User Constraints File

This is listing of complete *.ucf file for the FPGA-evb-S2 evaluation board. Copy part of the file containig pins you use only to your design UCF.

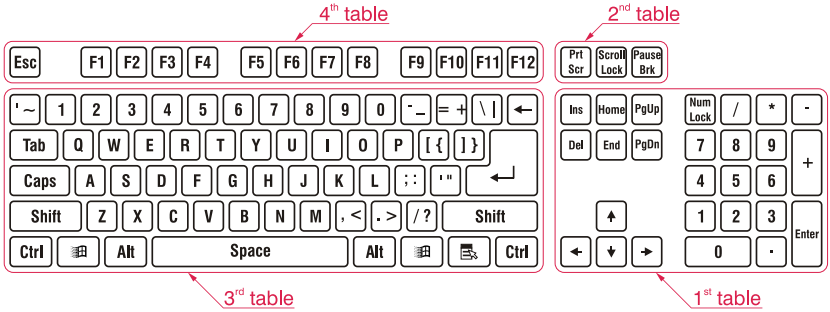
The file has easy syntax. Lines beginning with the # character are comments. Pins signed by prefix P are pins of a PQFP package. Signals which belong to a bus are denoted by postfix $<n>$, where the n is the index of the signal.

```
### UCF file for the FPGA-evb-S2

# Clock input
NET "clk" LOC = "P80";

# PS/2 Interface
NET "ps2_clk" LOC = "P87";
NET "ps2_dta" LOC = "P86";

# VGA Output
NET "v_sync" LOC = "P75";
NET "h_sync" LOC = "P74";
NET "blue<0>" LOC = "P67";
NET "blue<1>" LOC = "P68";
NET "green<0>" LOC = "P69";
NET "green<1>" LOC = "P70";
NET "red<0>" LOC = "P71";
NET "red<1>" LOC = "P73";

# LEDS
NET "led<0>" LOC = "P102";
NET "led<1>" LOC = "P101";
NET "led<2>" LOC = "P100";
NET "led<3>" LOC = "P99";
NET "led<4>" LOC = "P98";
NET "led<5>" LOC = "P97";
NET "led<6>" LOC = "P96";
NET "led<7>" LOC = "P95";
```

```
# DIP Switch
NET "dsw<0>" LOC = "P84";
NET "dsw<1>" LOC = "P83";
NET "dsw<2>" LOC = "P82";
NET "dsw<3>" LOC = "P81";

# Pushbuttons
NET "key<0>" LOC = "P94";
NET "key<1>" LOC = "P90";
NET "key<2>" LOC = "P89";
NET "key<3>" LOC = "P88";

# Timing requirements
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 50 ns HIGH 50 %;
```

C Keyboard Scan Codes

This is listing of the scan code set 2 of an AT keyboard. This set is the default for all AT keyboards, and is the only set used by all modern PCs.



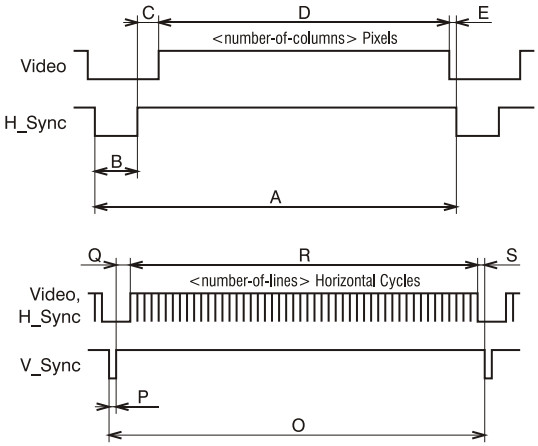
Key	Press	Release	Key	Press	Release
INS	E0,70	E0,F0,70	KP -	7B	F0,7B
DEL	E0,71	E0,F0,71	KP +	79	F0,79
HOME	E0,6C	E0,F0,6C	KP .	71	F0,71
END	E0,69	E0,F0,69	KP 0	70	F0,70
PG UP	E0,7D	E0,F0,7D	KP 1	69	F0,69
PG DN	E0,7A	E0,F0,7A	KP 2	72	F0,72
UP	E0,75	E0,F0,75	KP 3	7A	F0,7A
LEFT	E0,6B	E0,F0,6B	KP 4	6B	F0,6B
DOWN	E0,72	E0,F0,72	KP 5	73	F0,73
RIGHT	E0,74	E0,F0,74	KP 6	74	F0,74
NUM	77	F0,77	KP 7	6C	F0,6C
KP /	E0,4A	E0,F0,4A	KP 8	75	F0,75
KP *	7C	F0,7C	KP 9	7D	F0,7D
			KP ENT	E0,5A	E0,F0,5A

Key	Press	Release
Print Screen	E0,12,E0,7C	E0,F0,7C,E0,F0,12
Scroll Lock	7E	F0,7E
Pause/Break	E1,14,77,E1,F0,14,F0,77	none

Key	Press	Release	Key	Press	Release
A	1C	F0,1C	5	2E	F0,2E
B	32	F0,32	6	36	F0,36
C	21	F0,21	7	3D	F0,3D
D	23	F0,23	8	3E	F0,3E
E	24	F0,24	9	46	F0,46
F	2B	F0,2B	[	54	F0,54
G	34	F0,34	]	5B	F0,5B
H	33	F0,33	;	4C	F0,4C
I	43	F0,43	’	52	F0,52
J	3B	F0,3B	,	41	F0,41
K	42	F0,42	.	49	F0,49
L	4B	F0,4B	/	4A	F0,4A
M	3A	F0,3A	‘	0E	F0,0E
N	31	F0,31	-	4E	F0,4E
O	44	F0,44	=	55	F0,55
P	4D	F0,4D	\	5D	F0,5D
Q	15	F0,15	BCKSP	66	F0,66
R	2D	F0,2D	SPACE	29	F0,29
S	1B	F0,1B	TAB	0D	F0,0D
T	2C	F0,2C	CAPS	58	F0,58
U	3C	F0,3C	L SHFT	12	F0,12
V	2A	F0,2A	L CTRL	14	F0,14
W	1D	F0,1D	L WIN	E0,1F	E0,F0,1F
X	22	F0,22	L ALT	11	F0,11
Y	35	F0,35	R SHFT	59	F0,59
Z	1A	F0,1A	R CTRL	E0,14	E0,F0,14
0	45	F0,45	R WIN	E0,27	E0,F0,27
1	16	F0,16	R ALT	E0,11	E0,F0,11
2	1E	F0,1E	MENU	E0,2F	E0,F0,2F
3	26	F0,26	ENTER	5A	F0,5A
4	25	F0,25	ESC	76	F0,76

Key	Press	Release	Key	Press	Release
F1	05	F0,05	F7	83	F0,83
F2	06	F0,06	F8	0A	F0,0A
F3	04	F0,04	F9	01	F0,01
F4	0C	F0,0C	F10	09	F0,09
F5	03	F0,03	F11	78	F0,78
F6	0B	F0,0B	F12	07	F0,07
			ESC	76	F0,76

D VGA Signal Timing



Meaning of symbols in following tables: Res–resolution, VF–vertical frequency, HF–horizontal frequency, PR–pixel rate, HSP–horizontal synchronization polarity, VSP–vertical synchronization polarity.

Res	640×350	640×480	720×400	460×480
VF [Hz]	70.087	59.940	70.087	75.000
HF [kHz]	31.469	31.469	31.469	37.500
PR [MHz]	25.175	25.175	28.322	31.500
A [μs]	31.778	31.778	31.778	26.667
B [μs]	3.813	3.817	3.813	2.032
C [μs]	1.907	1.907	1.907	3.810
D [μs]	25.422	25.422	25.422	20.317
E [μs]	0.636	0.636	0.636	0.508
O [ms]	14.268	16.683	14.269	13.333
P [ms]	0.064	0.064	0.064	0.080
Q [ms]	1.907	1.048	1.112	0.427
R [ms]	11.122	15.253	12.711	12.800
S [ms]	1.176	0.318	0.381	0.027
HSP	+	-	-	-
VSP	-	-	+	-

Res	720×400	640×480	800×600	800×600
VF [Hz]	85.039	85.008	75.000	72.188
HF [kHz]	37.927	43.269	46.875	48.077
PR [MHz]	35.500	36.000	49.500	50.000
A [μs]	26.366	23.111	21.333	20.800
B [μs]	2.028	1.556	1.616	2.400
C [μs]	3.042	2.222	3.232	1.280
D [μs]	20.282	17.778	16.162	16.000
E [μs]	1.014	1.556	0.323	1.120
O [ms]	11.759	11.764	13.333	13.853
P [ms]	0.079	0.069	0.064	0.125
Q [ms]	1.107	0.578	0.448	0.478
R [ms]	10.546	11.093	12.800	12.480
S [ms]	0.026	0.023	0.021	0.770
HSP	-	-	+	+
VSP	+	-	+	+

Res	1024×768	800×600	1024×768	1024×768
VF [Hz]	60.004	85.061	75.029	84.997
HF [kHz]	48.363	53.674	60.023	68.677
PR [MHz]	65.000	56.250	78.750	94.500
A [μs]	20.677	18.631	16.660	14.561
B [μs]	2.092	1.138	1.129	1.016
C [μs]	2.462	2.702	2.235	2.201
D [μs]	15.754	14.222	13.003	10.836
E [μs]	0.369	0.569	0.203	0.508
O [ms]	16.666	11.756	13.328	11.765
P [ms]	0.124	0.056	0.050	0.044
Q [ms]	0.600	0.503	0.466	0.524
R [ms]	15.880	11.179	12.795	11.183
S [ms]	0.062	0.019	0.017	0.015
HSP	-	+	+	+
VSP	-	+	+	+