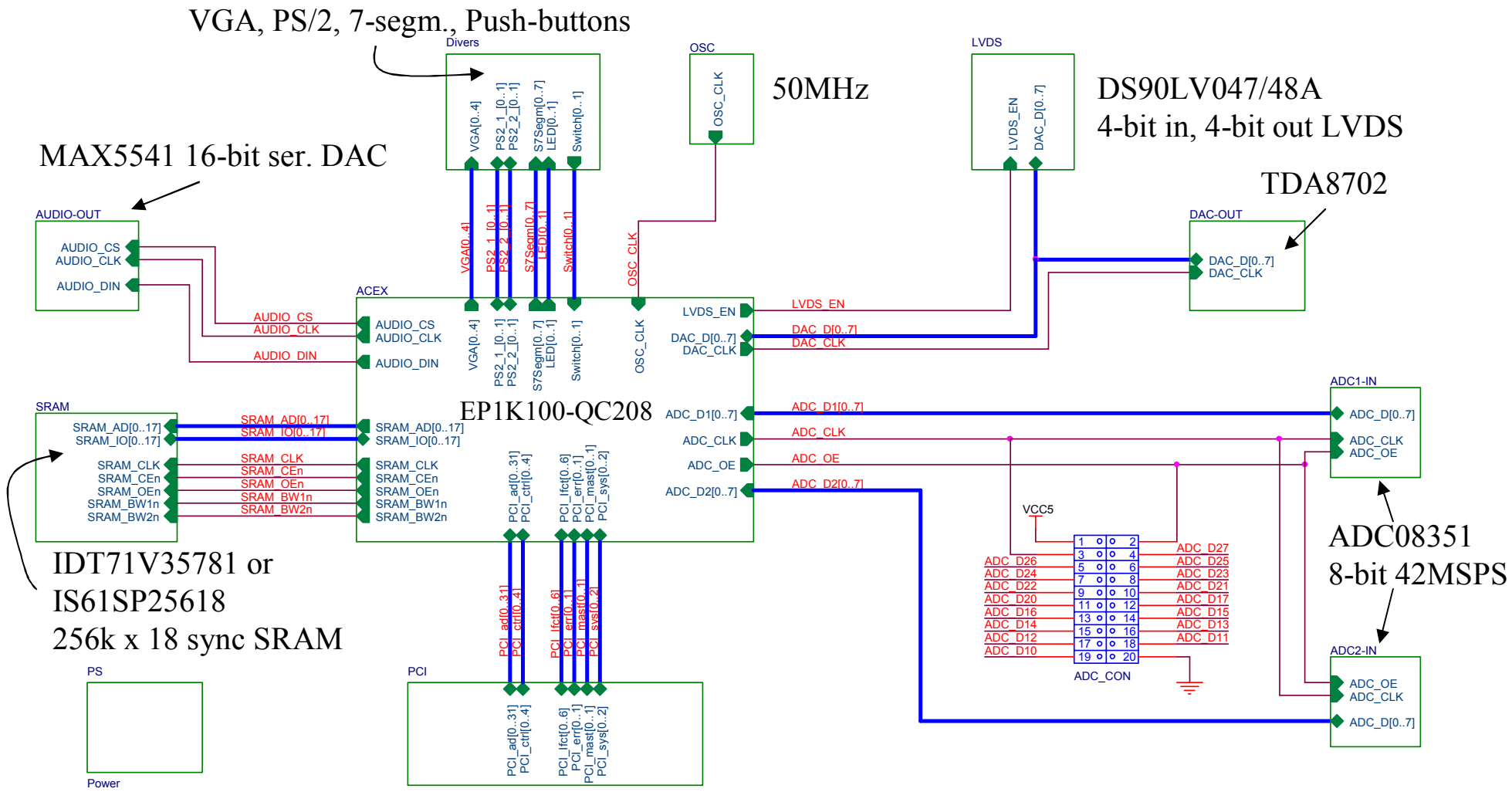
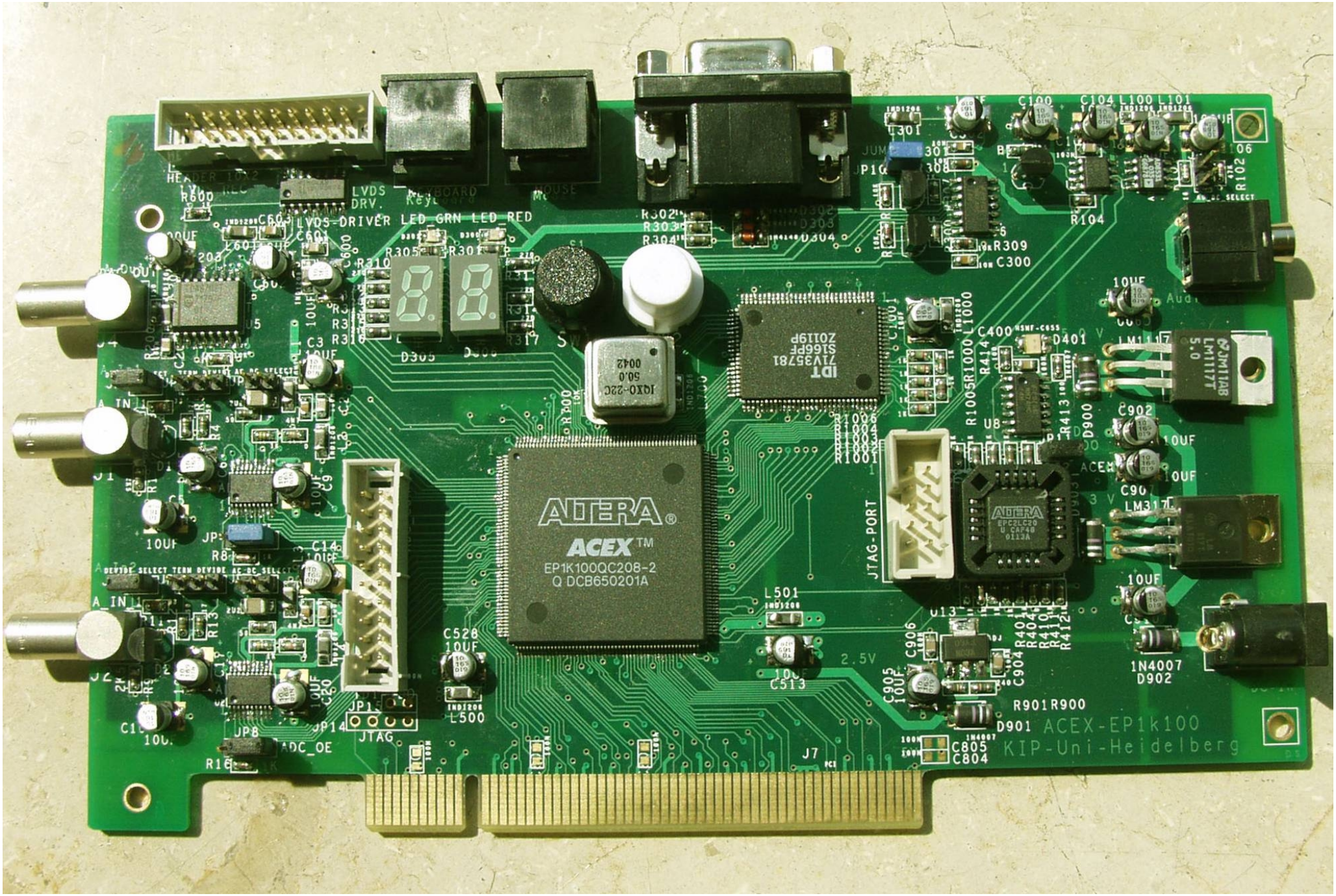
ACEX-KIP Board block diagram



2001-2005

angelov@kip.uni-heidelberg.de



2001-2005

angelov@kip.uni-heidelberg.de

Some ready building blocks

The following building blocks for MAX+PLUS II could speed up your designs for the ACEX-KIP Board. For more information you have to read the data sheets of the components on board (especially this is recommended for the ACEX chip, Sync-SRAM, Serial-DAC and LVDS).

1) Install all modules (.edf or .vhd files WITH the .sym files) in a separate directory. In MAX+PLUS II in \Options\User libraries add the directory to the list. So you can access this modules from your new design(s).

2) Copy the pin & chip assignments (from all_pins.acf) to the .acf file of your project.

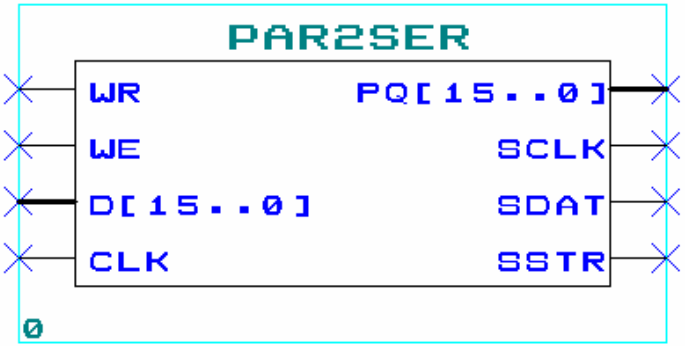
angelov@kip.uni-heidelberg.de

Parallel to serial converter (SPI)

(use it for the serial 16-bit DAC MAX5541)

Inputs:

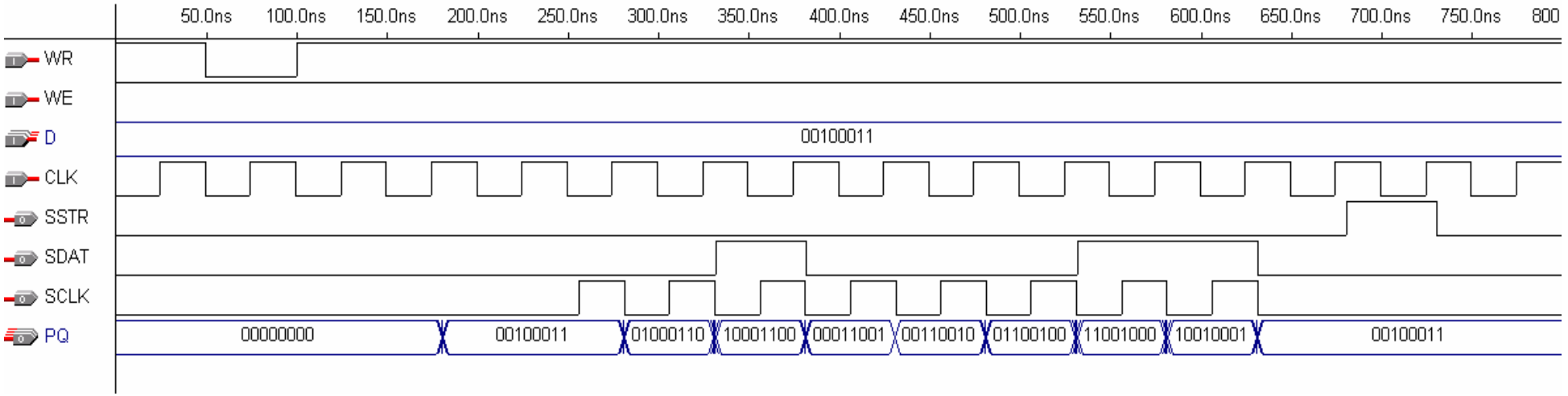
WR - write
WE - write enable
D[15..0] - data inputs
CLK - clock for the serial interface (<10MHz)



Outputs:

PQ[15..0] - par. outputs
SCLK - serial clock
SDAT - serial data
SSTR - serial strobe or CSn

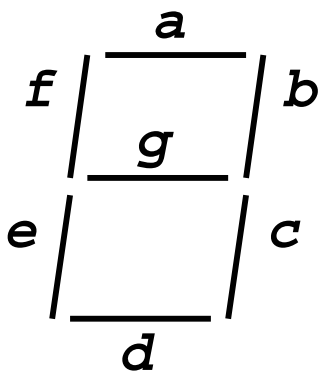
|AUDIO_CSn : OUTPUT_PIN = 120;
|AUDIO_CLK : OUTPUT_PIN = 119;
|AUDIO_DAT : OUTPUT_PIN = 116;



2001-2005

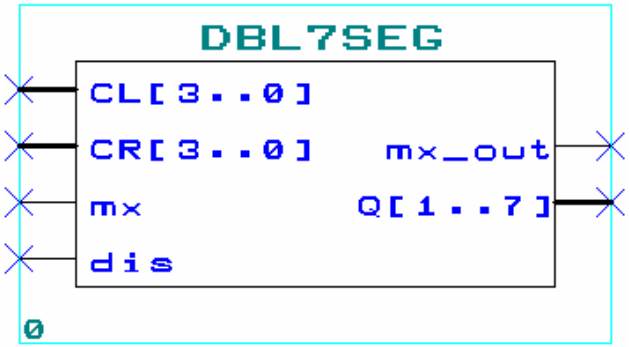
angelov@kip.uni-heidelberg.de

Dual 7 segment indicator



Inputs:

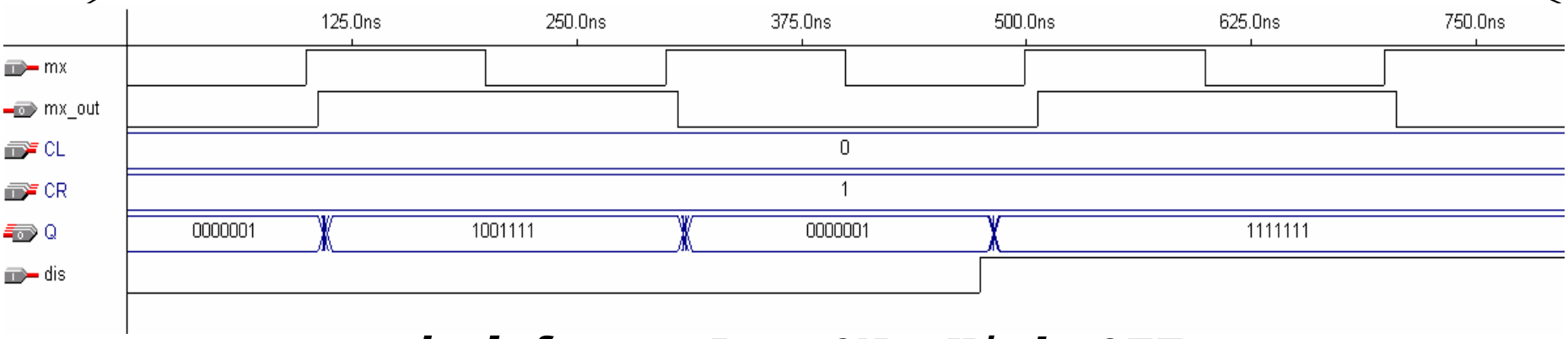
CL - Hex code (left)
CR - Hex code (right)
Mx - some clock for time mux (100-1000Hz)
Dis - disable outputs



Outputs:

Mx_out - mux control
Q1→a, Q2→b,.., Q7→g

|R7S_mux : OUTPUT_PIN = 179;
|R7S1 : OUTPUT_PIN = 192;
|R7S2 : OUTPUT_PIN = 180;
|R7S3 : OUTPUT_PIN = 186;
|R7S4 : OUTPUT_PIN = 187;
|R7S5 : OUTPUT_PIN = 189;
|R7S6 : OUTPUT_PIN = 191;
|R7S7 : OUTPUT_PIN = 190;



abcdefg *Low=ON, High=OFF*

angelov@kip.uni-heidelberg.de

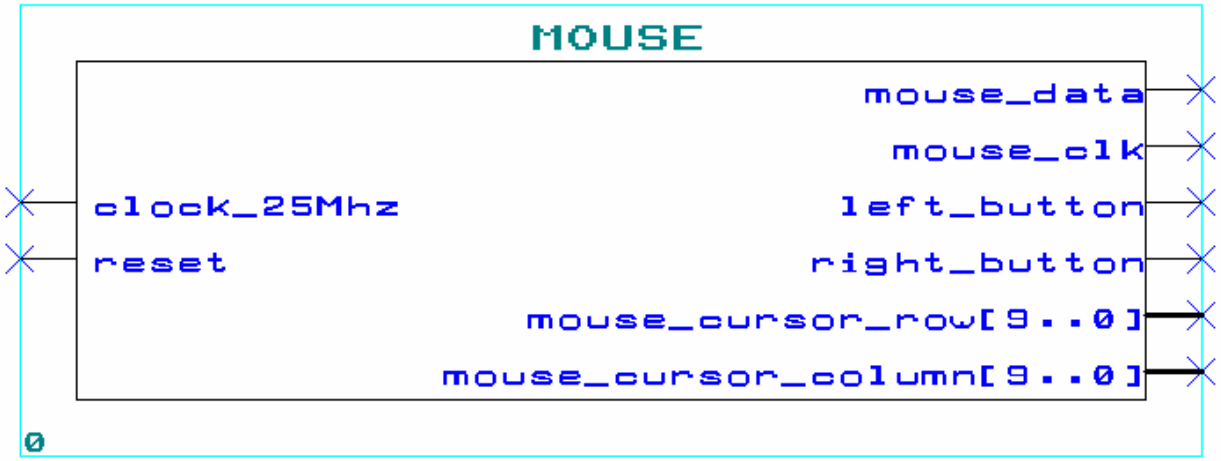
PS/2 Mouse*

Inputs:

Clock

Reset

(active high)



Outputs:

to the PS/2 mouse \ to your design /

|MDAT : BIDIR_PIN = 202;

|MCLK : BIDIR_PIN = 203;

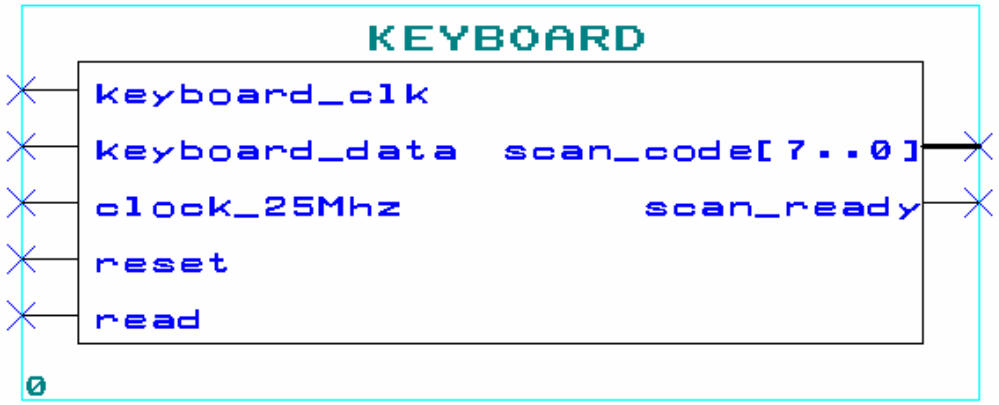
* taken from "Rapid prototyping of digital systems" by J.Hamblen and M.Furman.

angelov@kip.uni-heidelberg.de

PS/2 Keyboard*

Inputs:

From PS/2 Keyboard
Clock
Reset (act.H)
Read (clears the scan ready)



Outputs:

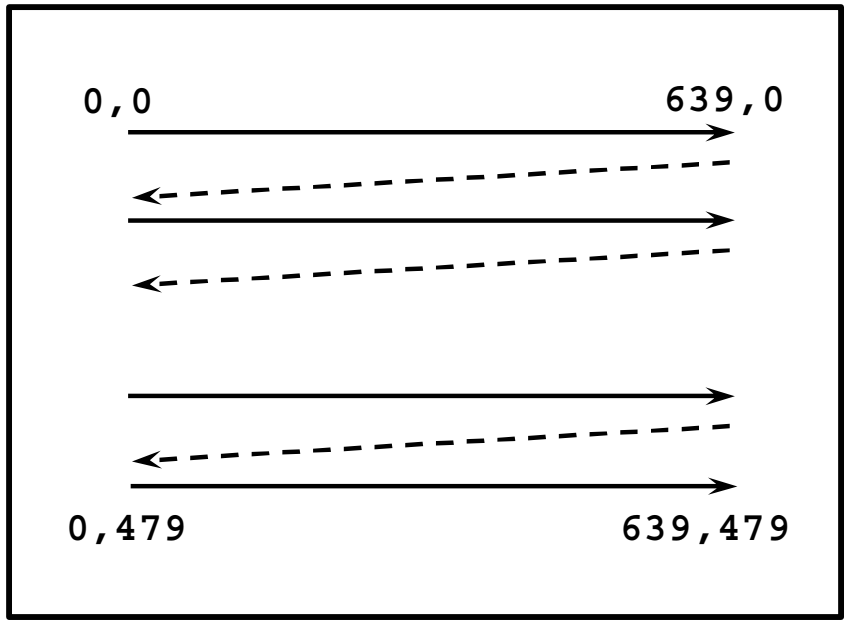
ASCII code
Scan ready will be set to '1' if a new code appears at the output

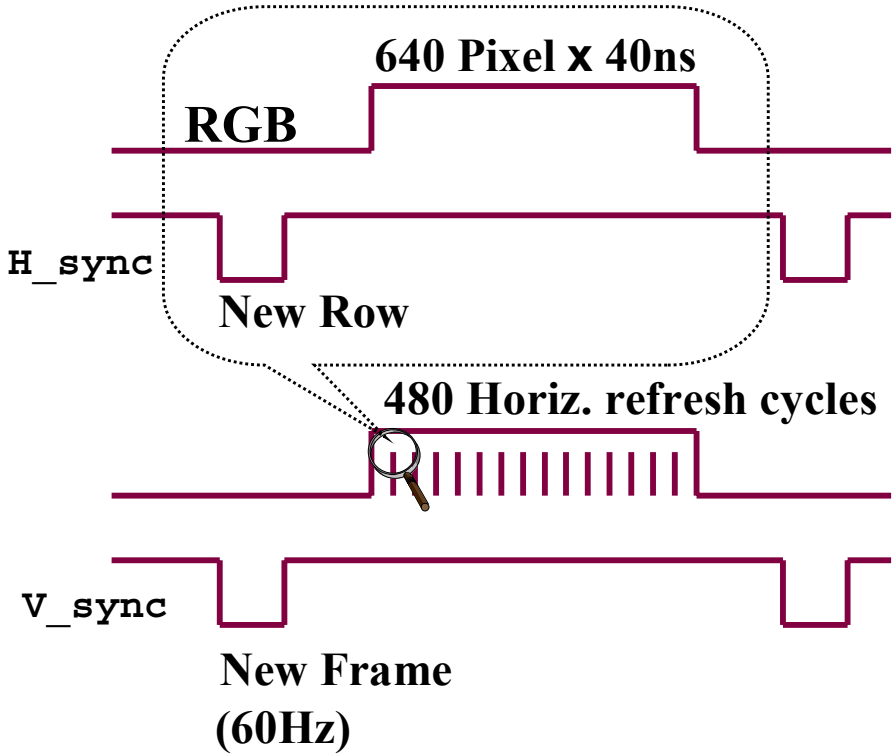
```
|KDAT : INPUT_PIN = 204;
|KCLK : INPUT_PIN = 205;
```

* taken from "Rapid prototyping of digital systems" by J.Hamblen and M.Furman.

angelov@kip.uni-heidelberg.de

VGA: Image and Sync Signals



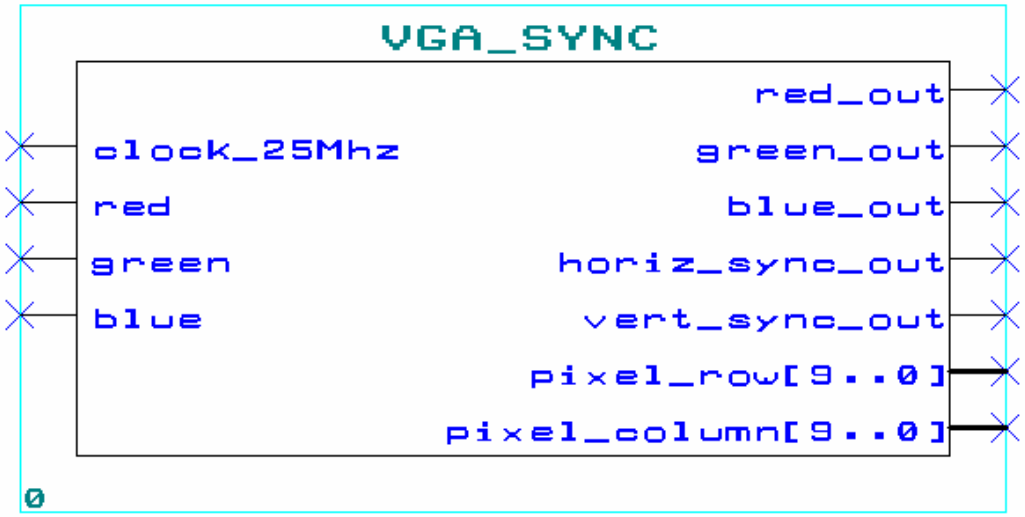


angelov@kip.uni-heidelberg.de

VGA Interface*

Inputs:

Clock
Red, green, blue from your design



Outputs:
**Red **
Green
Blue to VGA
H_sync
V_sync /

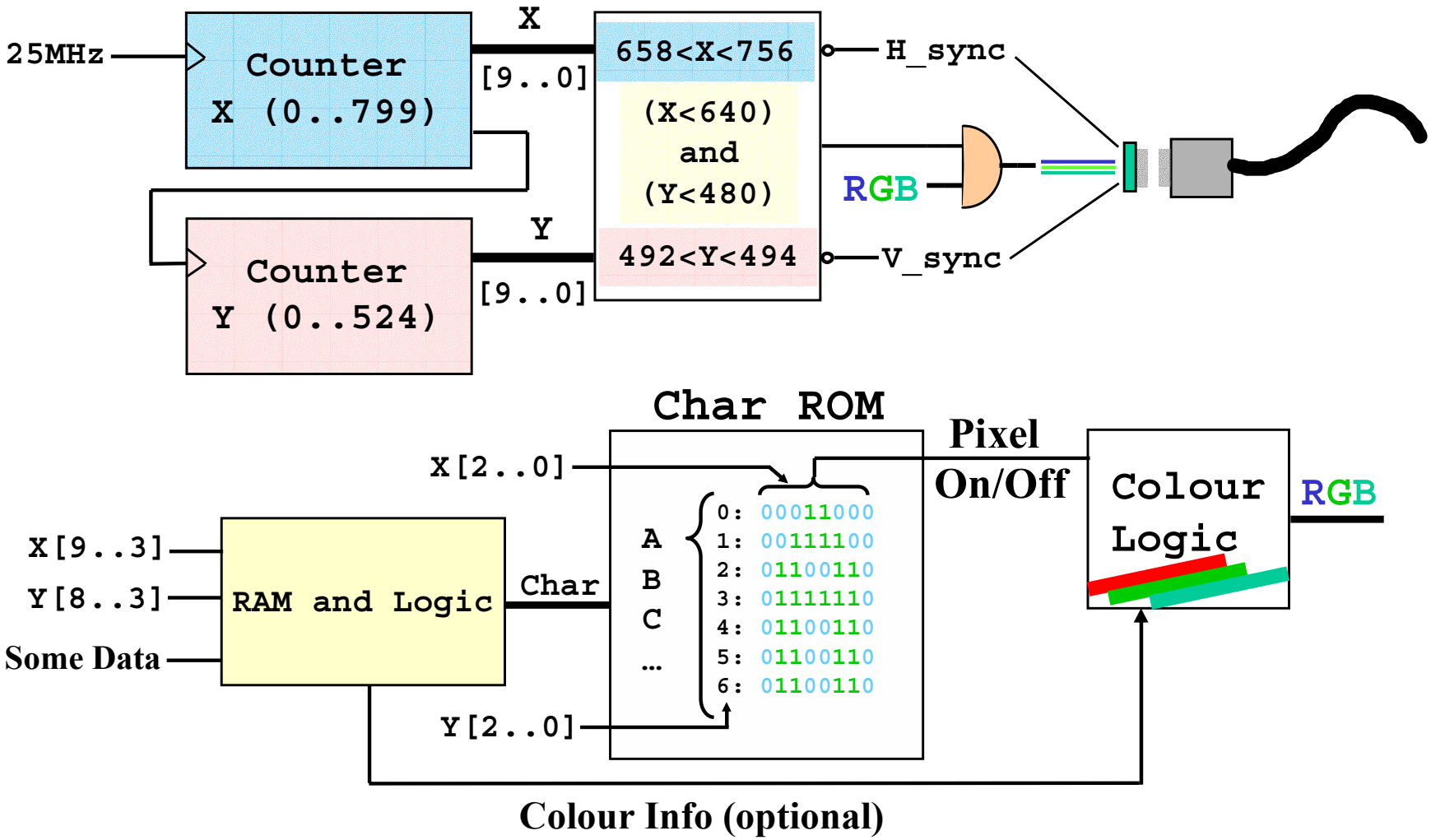
To your design

```
|BLUE   : OUTPUT_PIN = 200;
|GREEN : OUTPUT_PIN = 199;
|RED    : OUTPUT_PIN = 198;
|V_SYNC : OUTPUT_PIN = 197;
|H_SYNC : OUTPUT_PIN = 196;
```

* taken from "Rapid prototyping of digital systems" by J.Hamblen and M.Furman.

angelov@kip.uni-heidelberg.de

Block Diagram for simple Text Output

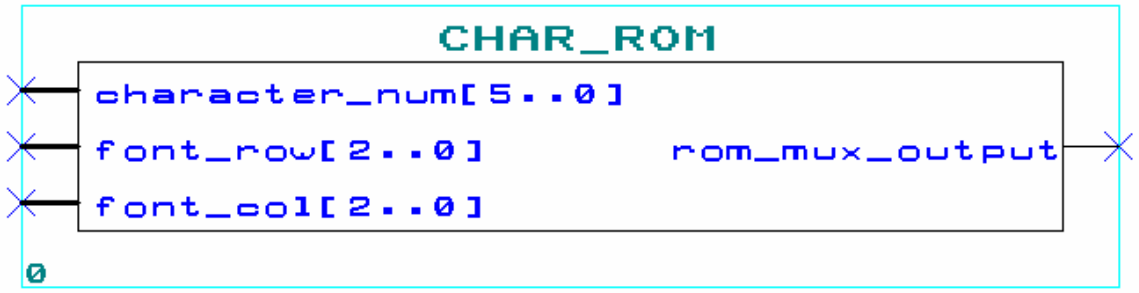


angelov@kip.uni-heidelberg.de

Character generator*

Inputs:

Char code (not ASCII) Normally you connect this inputs to X[2..0] and Y[2..0] (pixel coordinates)



Note: the file 'tcgrom.mif' contains the font definition

Outputs:

Pixel ON/OFF (Normally connected to some of the colour inputs of the VGA module)

The table was compressed in order to save RAM space

Code	0	1	2	3	4	...	9	10	11	...	15	16	17	...	34	35	36 ... 65
Char	0	1	2	3	4	...	9	A	B	...	F	G	H	...	Y	Z	<space>

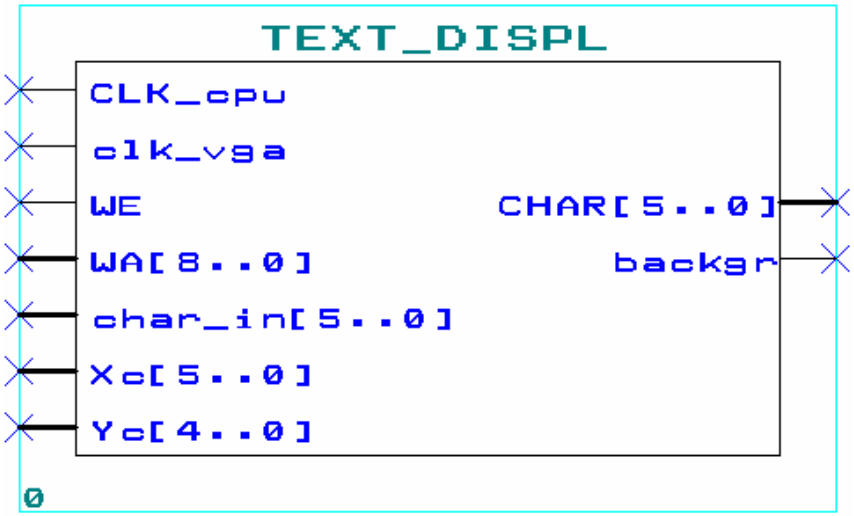
* taken from "Rapid prototyping of digital systems" by J.Hamblen and M.Furman.

angelov@kip.uni-heidelberg.de

Video RAM for text output

Inputs:

CPU clock
VGA clock
Write Enable
Write Address
Char # in
Column, row
(connect to
X[9..4] and
Y[8..4] pixel
coordinates)



Outputs:

Char at pos. Xc Yc
='1' if out of text
window

Clk_cpu and clk_vga could be different, WE is synchronous to clk_cpu. WA = row[3..0] & column[4..0] – the text window has 16 lines with 32 characters each. The module uses dual port RAM and is intended for use with ACEX family.

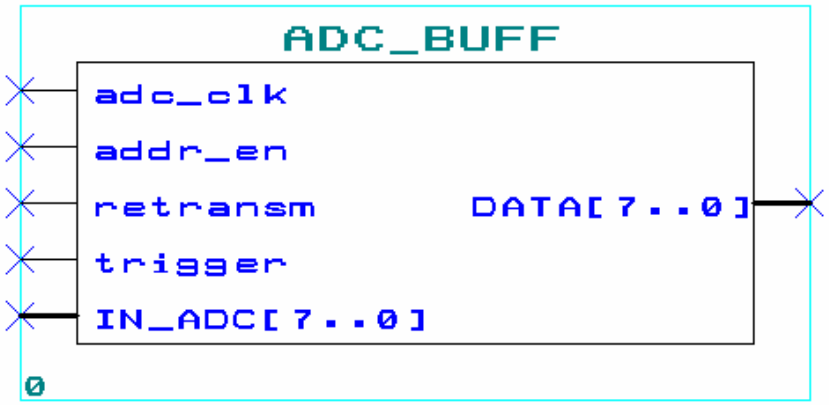
angelov@kip.uni-heidelberg.de

Buffer for ADC data

(use it for the fast 8-bit ADCs NSC ADC08351)

Inputs:

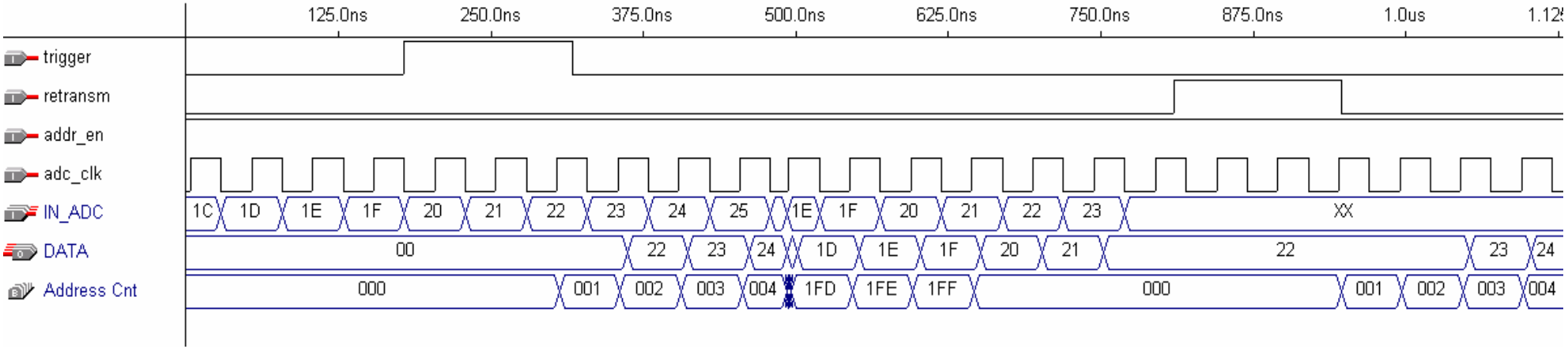
ADC clock
Address enable by read
Retransmit data
Trigger acquisition
Data inputs from ADC



Outputs:

Read data

The depth of the buffer is 512. *Trigger* starts the acquisition at *adc_clk* rate, *retransm* clears the address counter and starts reading



angelov@kip.uni-heidelberg.de

Digital oscilloscope

Inputs:

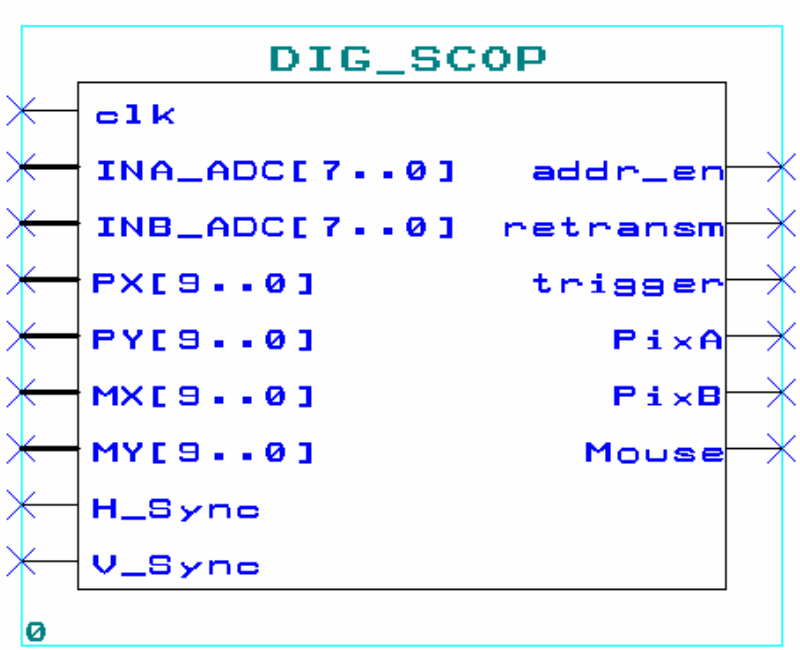
ADC & VGA clock

Data from ADC buffers

Pixel coordinates

Mouse coordinates

VGA sync signals



Outputs:

**Address enable **

Retransmit - ADC buffers

Trigger /

Pixel ON/OFF for ch.A

Pixel ON/OFF for ch.B

Mouse ON/OFF

There are some restrictions: the ADC and VGA clocks are the same; trigger appears during the dead time of the VGA.

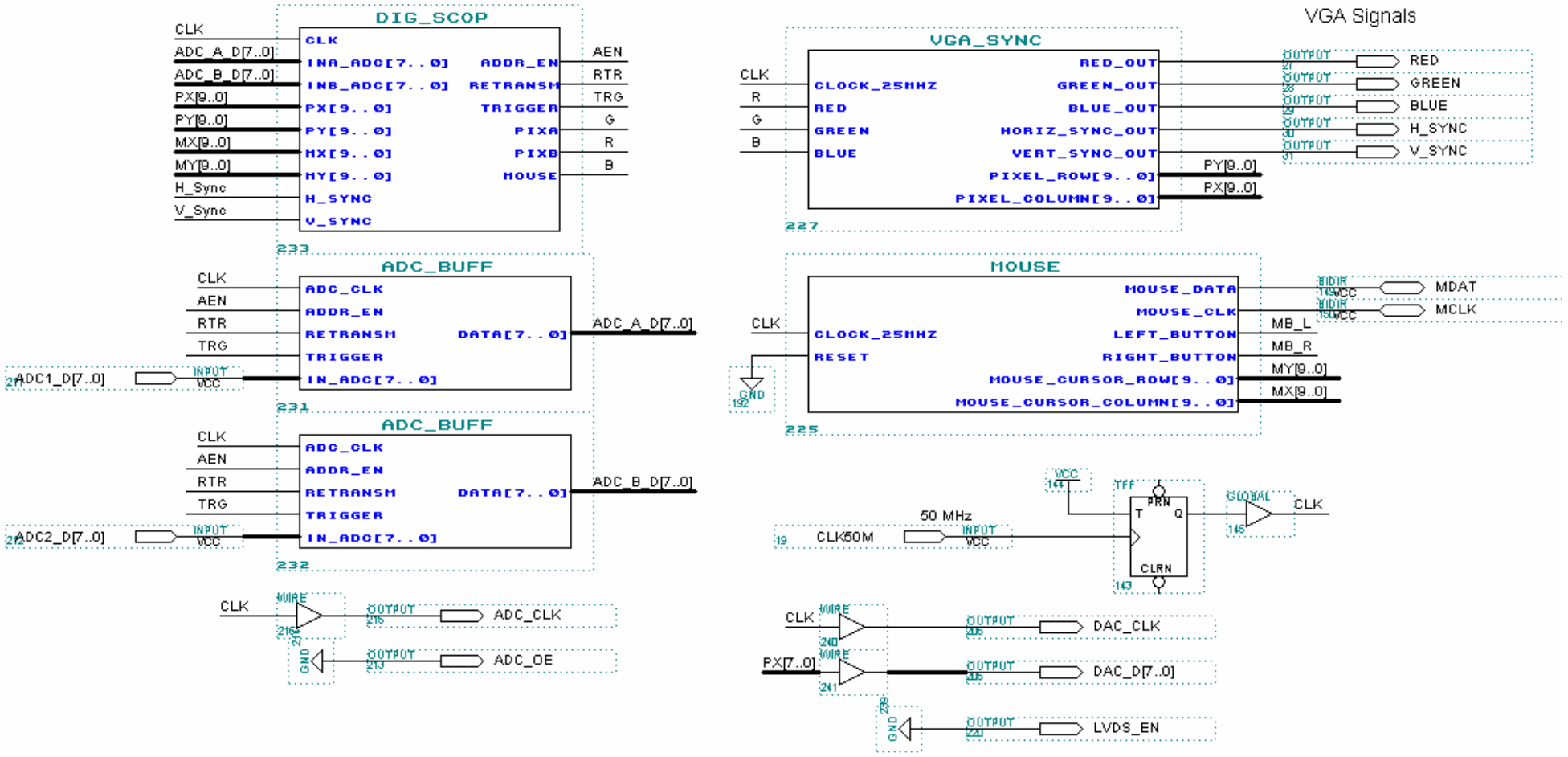
```
|ADC1_D0 : INPUT_PIN = 39;
|ADC1_D1 : INPUT_PIN = 38;
|ADC1_D2 : INPUT_PIN = 37;
|ADC1_D3 : INPUT_PIN = 36;
|ADC1_D4 : INPUT_PIN = 31;
|ADC1_D5 : INPUT_PIN = 30;
|ADC1_D6 : INPUT_PIN = 29;
|ADC1_D7 : INPUT_PIN = 28;
```

```
|ADC2_D0 : INPUT_PIN = 25;
|ADC2_D1 : INPUT_PIN = 24;
|ADC2_D2 : INPUT_PIN = 19;
|ADC2_D3 : INPUT_PIN = 18;
|ADC2_D4 : INPUT_PIN = 17;
|ADC2_D5 : INPUT_PIN = 16;
|ADC2_D6 : INPUT_PIN = 15;
|ADC2_D7 : INPUT_PIN = 14;
```

```
|ADC_OE  : OUTPUT_PIN = 27; -- put to GND
|ADC_CLK : OUTPUT_PIN = 26;
```

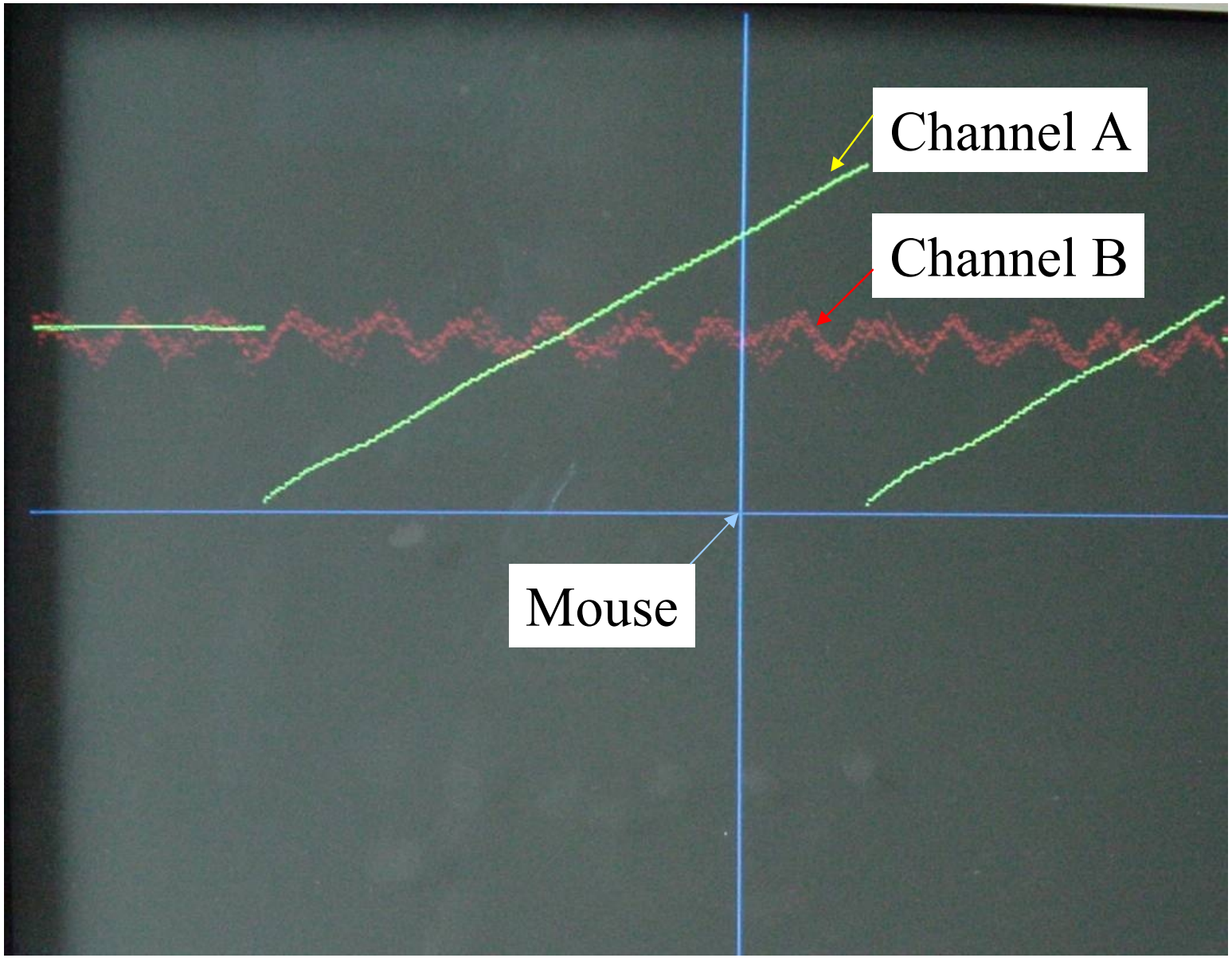

angelov@kip.uni-heidelberg.de

Digital Oscilloscope Design



2001-2005

angelov@kip.uni-heidelberg.de



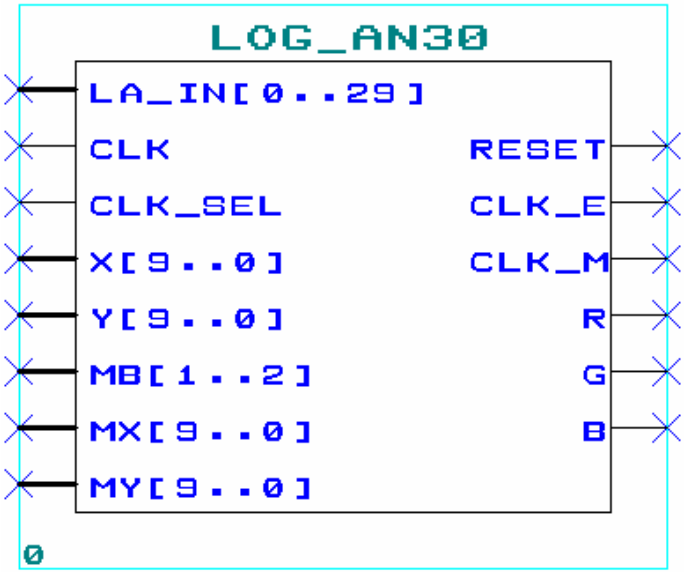
2001-2005

angelov@kip.uni-heidelberg.de

Logic Analyzer

Inputs:

30 signals from DUT
VGA clock
Connect to '1'
Pixel coordinates
from the VGA block
Mouse buttons
Mouse pointer
coordinates



Outputs:

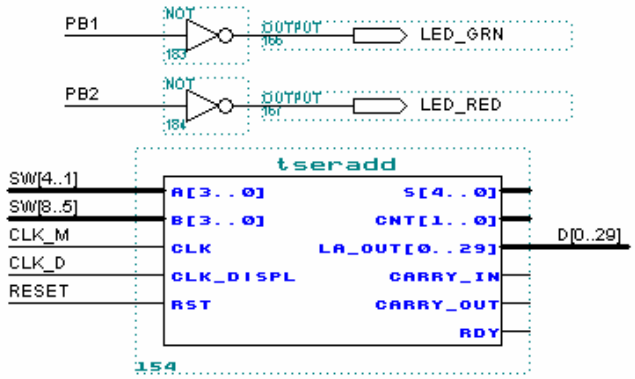
Reset to the DUT
Clock for display
Clock for the DUT
Red \
Green - to VGA
Blue /

The Design Under Test must be synchronous and must have a RESET signal. The logic analyzer block resets the DUT at the beginning of every VGA row, sends the clocks and plots up to 30 or 60 signals directly on the VGA display. You can label the signals with up to 4 characters stored in *signames.mif*. You can create this file from a normal text file by the utitility *dis_la.exe*.

angelov@kip.uni-heidelberg.de

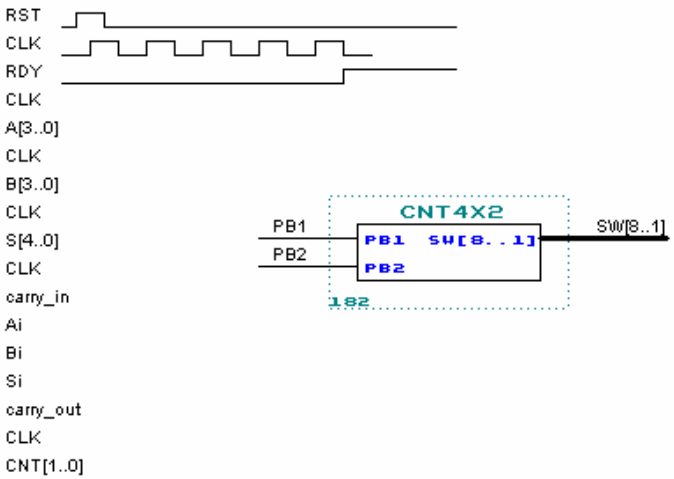
Logic analyzer design with Serial Adder

USER DESIGN



(4-bit Serial adder)

The wavediagram on VGA display looks like:



REMARKS:

The design will be clocked by CLK_M !

CLK_D comes 1 pixel before CLK_M to make the waveform casual.

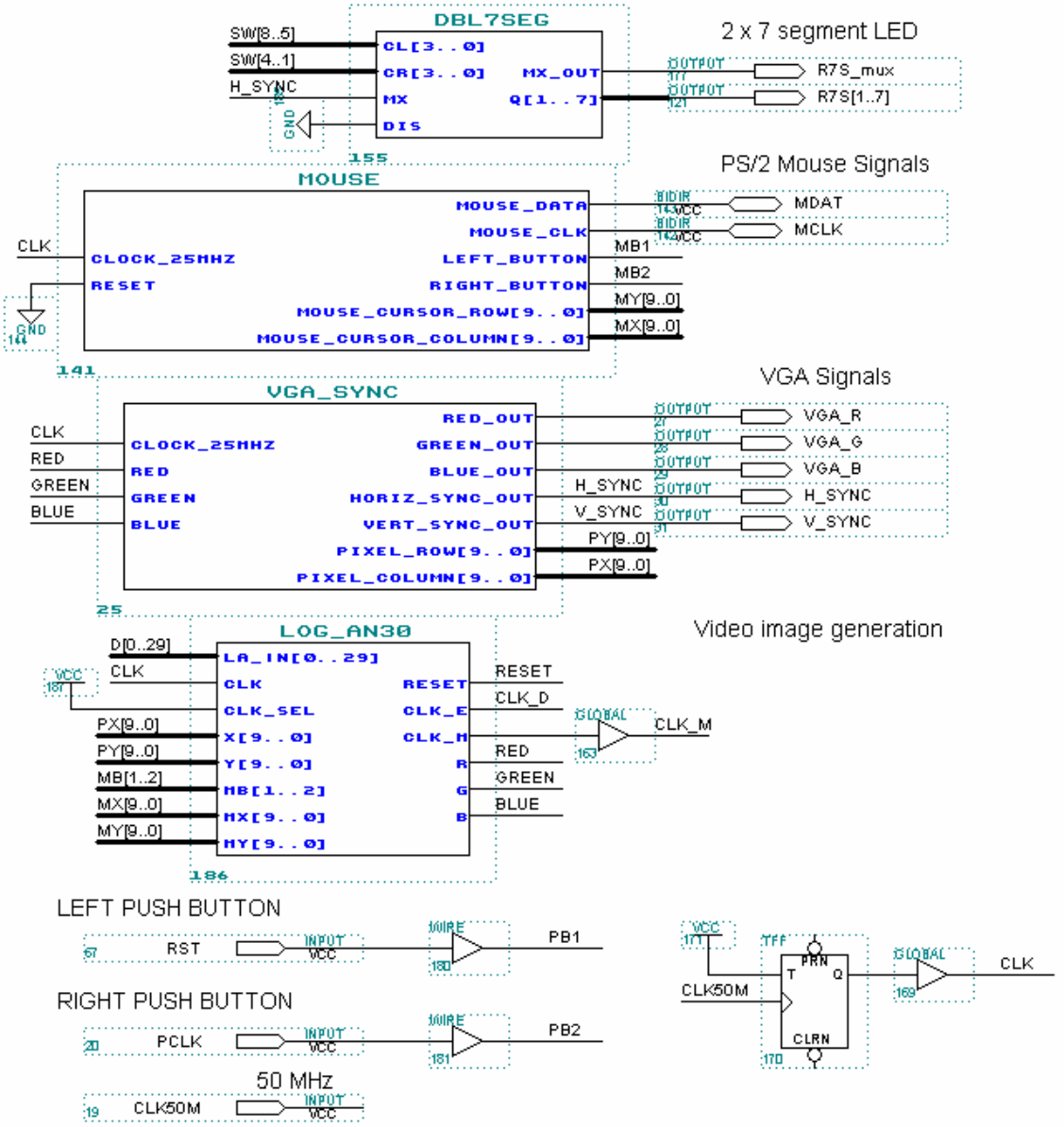
To use 60 signals, edit the generic Nsig in VMUX2.

To have more clock cycles in the waveform edit the generic CLK_PERIOD in VMUX2

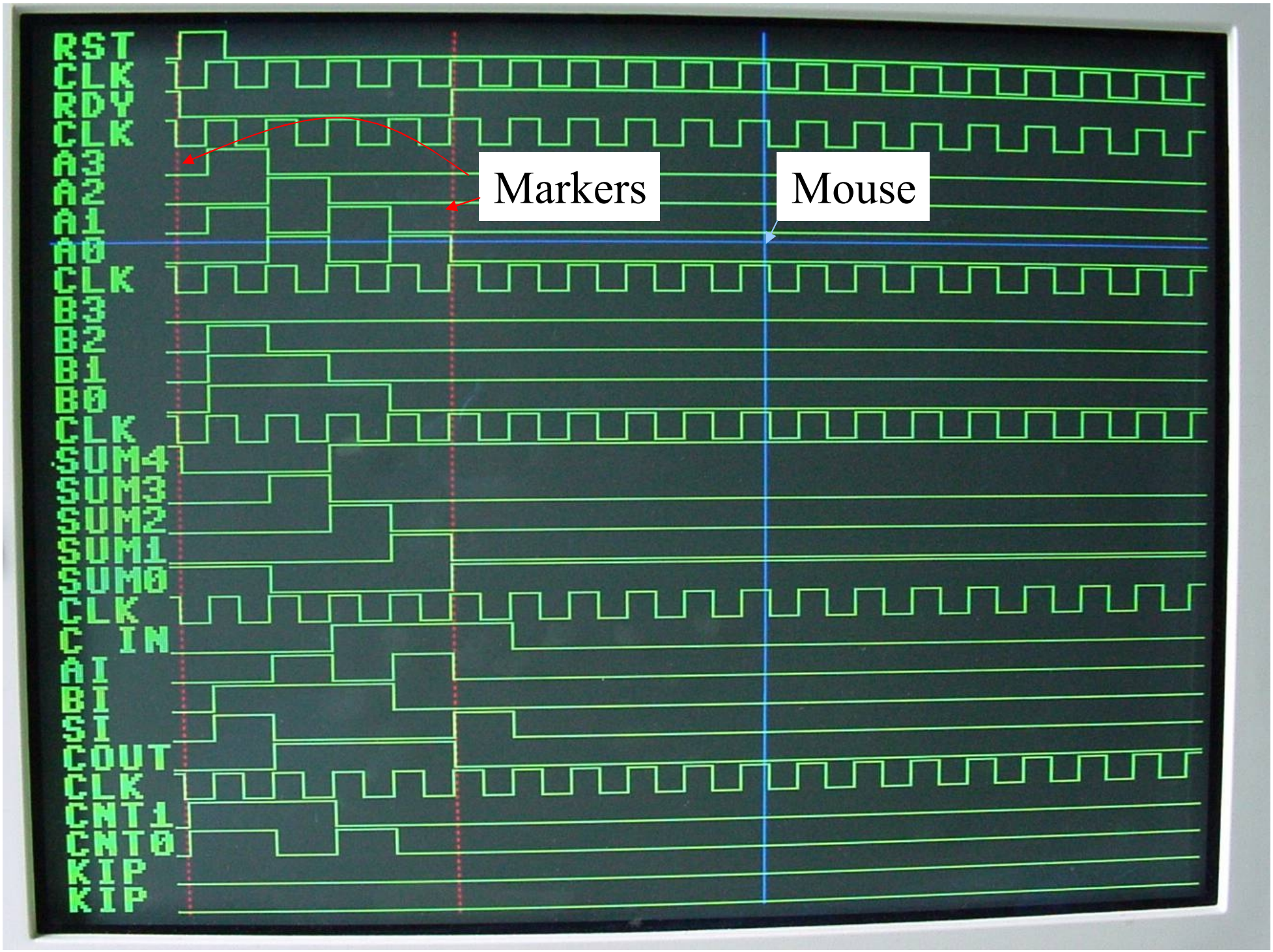
Any other user design should have at least the 30 | 60 outputs.

Any other user design should have a reset.

LOGIC ANALYZER DESIGN



angelov@kip.uni-heidelberg.de



2001-2005

angelov@kip.uni-heidelberg.de

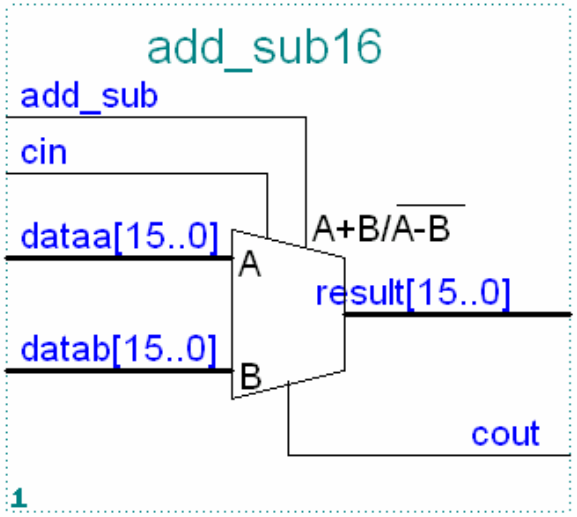
Some hints:

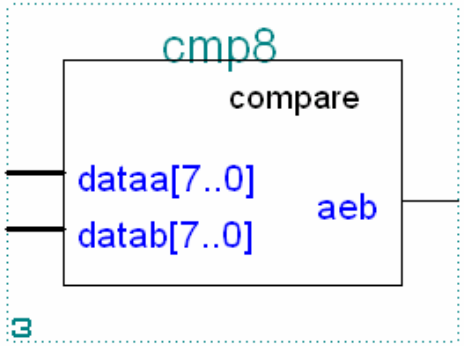
1) Use hierarchical design methodology.
2) Simulate your sub-designs before trying to use them in the higher levels of the hierarchy. To do that faster set the Compiler in \Processing to Functional Extractor. When your sub-design works satisfactory, compile it fully to see the area (or Logic Cells) needed. Keep track on warnings like 'compiler ignores something as the design does not depend on it', or 'some flip-flop is stuck to GND'.
3) Do not try to connect your blocks with wires and buses, instead use labels as much as possible, give clear names to the signals. If you need to connect two signals with different names, use the symbol 'wire'.
4) For more than 1-bit wide functions use the Mega Wizard plug-in to create exactly the block you need, some examples of such blocks are given later.

angelov@kip.uni-heidelberg.de

Mega-wizard Plug-In (arithmetic components)

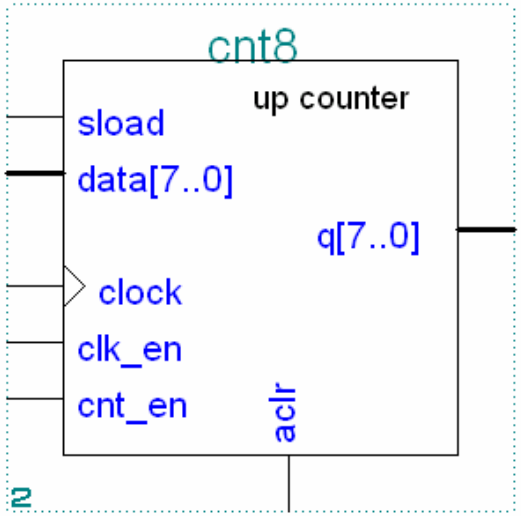
Add/subtract module with carry-in & carry-out. Check with the simulator the meaning of carry-signals by subtract!





Comparator, output is 1 if dataa=datab

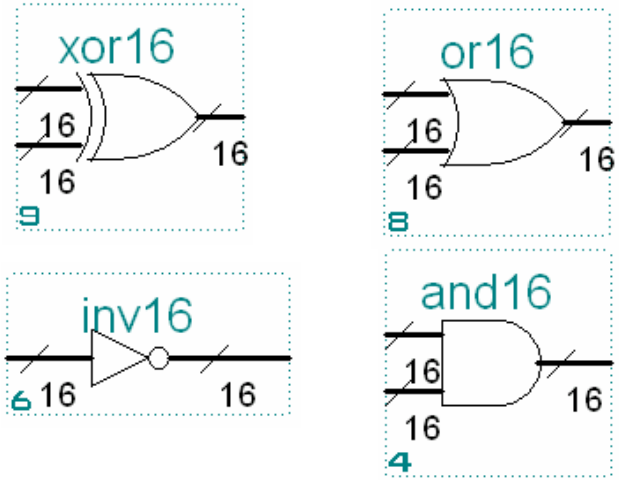
Synchronous counter with sync load, async clear. Clk_en is for counting & sload, while cnt_en is for counting only.



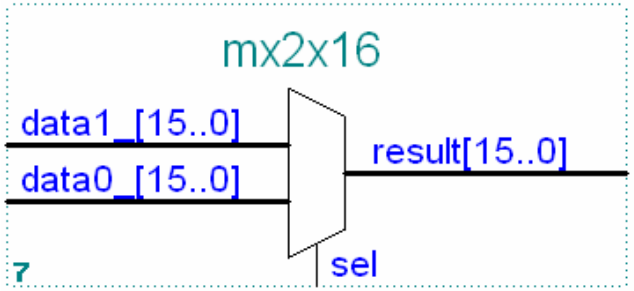
angelov@kip.uni-heidelberg.de

Mega-wizard Plug-In (gates)

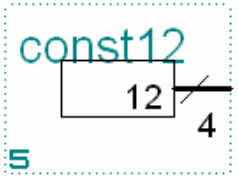
Bitwise logical operations:



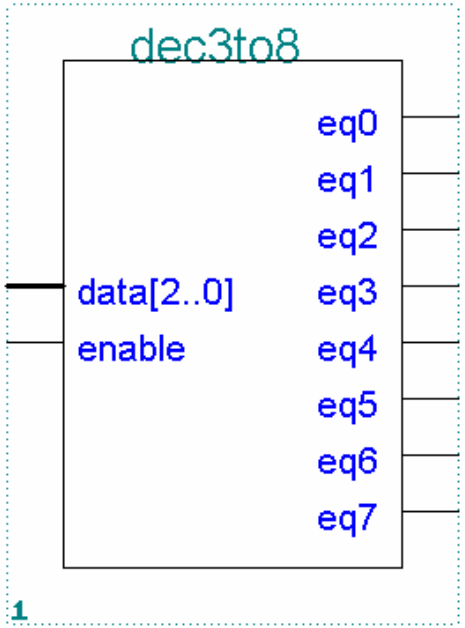
Multiplexer 2:1 for 16-bits



Constant with selectable width



Decoder 3 => 8 with enable

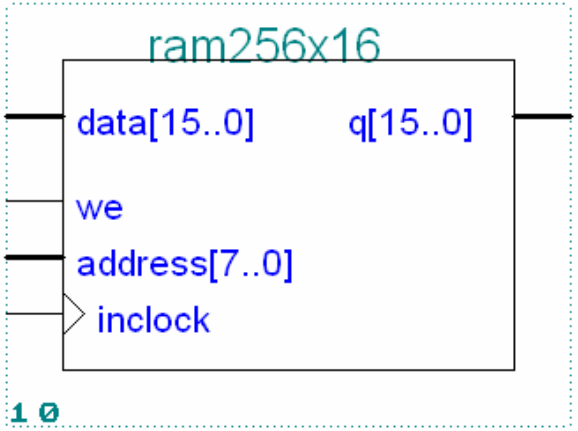


=1 if data=0 & enable='1'

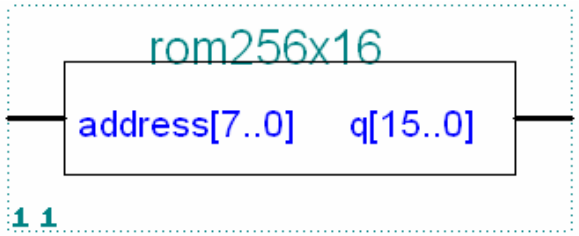
=1 if data=7 & enable='1'

angelov@kip.uni-heidelberg.de

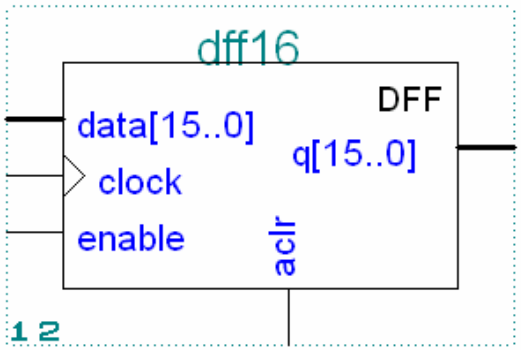
Mega-wizard Plug-In (storage components)



SRAM. It is strongly recommended to configure it as synchronous, with registered data inputs, address and write control, but unregistered output. When do you get valid data? Simulate if not clear!



ROM. Totally asynchronous. You need to set the name of the memory initialization file (.mif)

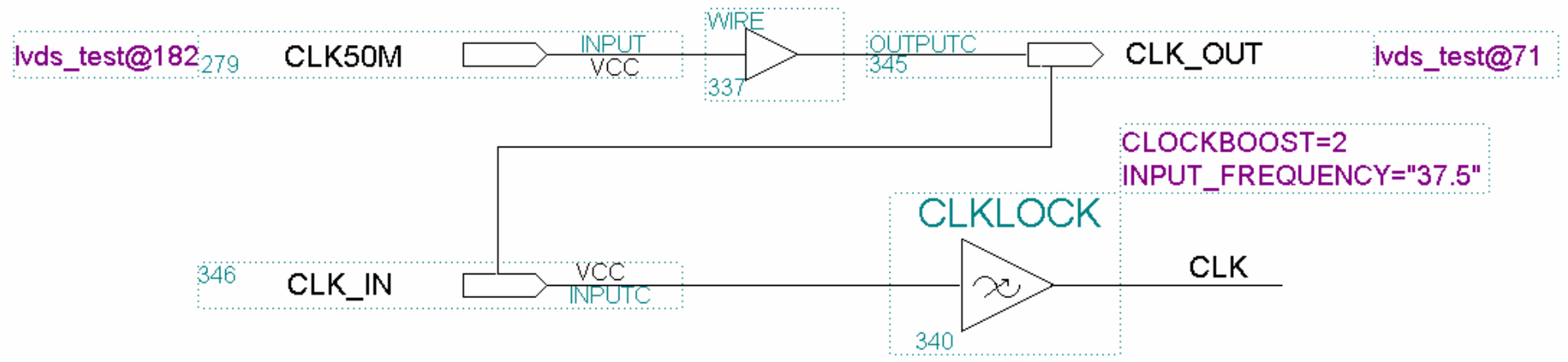


REGISTER (16 D-flip-flops) with clock enable and asynchronous clear.

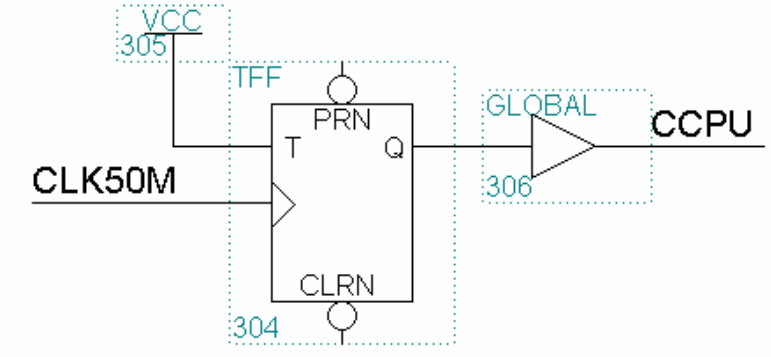
angelov@kip.uni-heidelberg.de

Clock

In order to multiply the input clock x2 use the following circuit:



If you need to divide the input clock by 2 use the following circuit:



Instead of using ‘Global’ you can set in Global Project Logic Synthesis the Auto Global to ON.

angelov@kip.uni-heidelberg.de