# ReconOS v3 Workshop

February 2-3, 2012
University of Paderborn

## System set-up

### Prerequisites

- Xilinx 13.3 tools
- MicroBlaze Linux kernel

### *Log in*

Log in as user='workshop', password='workshop'

### Check out repository:

```
> rm -rf $HOME/reconos_v3
> git clone ssh://workshop@pc-techinf-
25.cs.upb.de:22222/var/git/reconos_v3 $HOME/reconos_v3
Note: password='workshop'
```

### Set ReconOS environmental variable:

```
> export RECONOS=$HOME/reconos_v3
```

### Build ReconOS  libraries:

```
> cd $RECONOS/linux/fsl_driver
> make
> cd $RECONOS/linux/getpgd
> make
> cd $RECONOS/linux/libreconos
> make
> cd $RECONOS/linux/readpvr
> make
```

### Copy ReconOS libraries and scripts to the root folder of the Linux kernel:

```
> cp $RECONOS/linux/fsl_driver/fsl.ko  /exports/rootfs_mb/.
> cp $RECONOS/linux/getpgd/getpgd.ko  /exports/rootfs_mb/.
> cp $RECONOS/linux/readpvr/readpvr  /exports/rootfs_mb/.
> cp $RECONOS/linux/scripts/load_fsl.sh  /exports/rootfs_mb/.
> cp $RECONOS/linux/scripts/load_getpgd.sh  /exports/rootfs_mb/.
> cp $RECONOS/linux/scripts/rcS  /exports/rootfs_mb/.
```

## Sort-Demo

In this tutorial, we will use threads that sort data blocks of 8 kbytes. The threads are implemented in software and/or hardware. The sorting thread waits for a message from an incoming message queue containing the address of the data chunk and sends a message to an outgoing queue when sorting is done. For this demo, we generate a number of 8 kbyte blocks of unsorted data. Each block can then be sorted by either a software or a hardware thread. As the hardware threads run in parallel to the CPU, they can achieve a performance boost.

### 1. Create software design

```
> export RECONOS=$HOME/reconos_v3
> cd $RECONOS/demos/sort_demo/linux
> make clean all
> cp $RECONOS/demos/sort_demo/linux/sort_demo /exports/rootfs_mb/.
```

### 2. Create hardware design

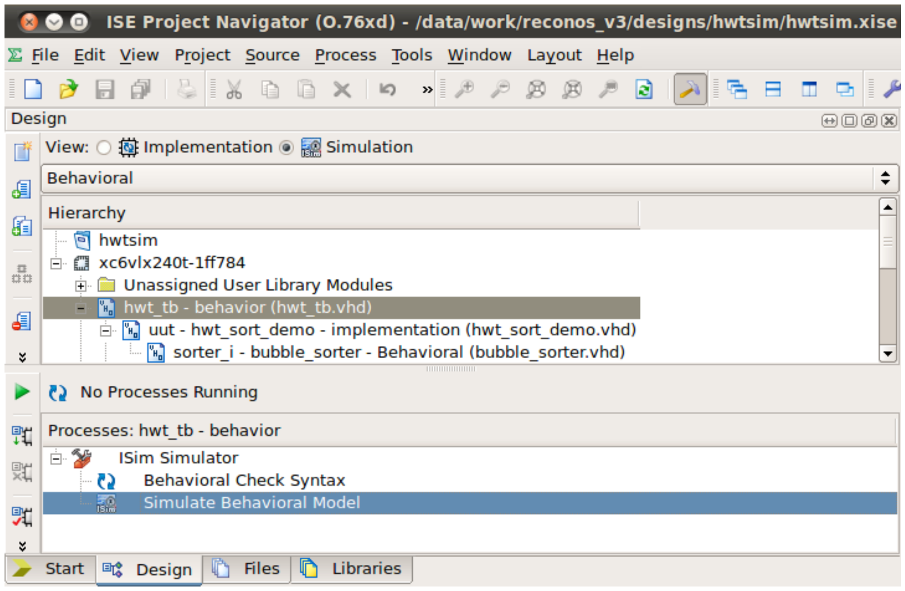
```
> cd $RECONOS/demos/sort_demo/hw
> ./setup.sh
> cd edk
> make clean all
```

Note: Do not wait for the hardware design to finish, but continue with the next step!

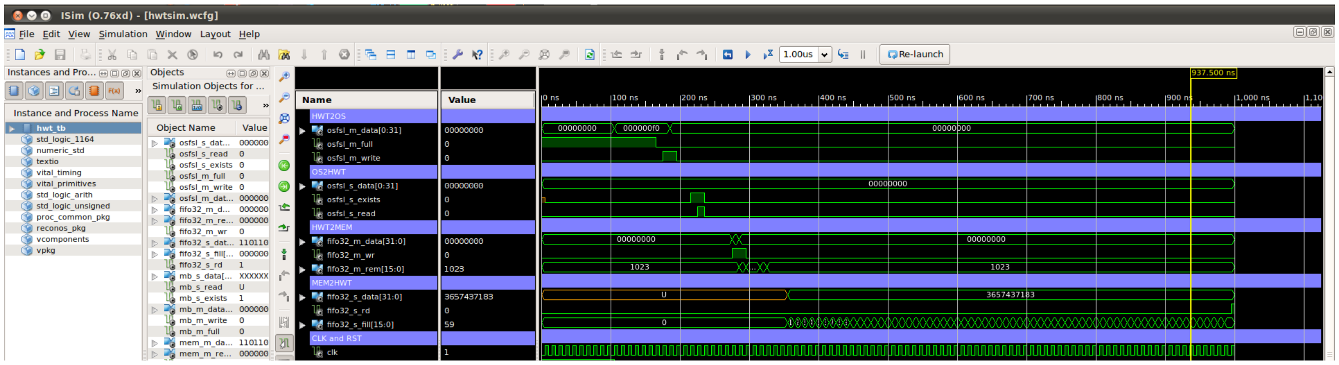
### *3. Simulate hardware thread (new terminal)*

```
> export RECONOS=$HOME/reconos_v3
> ise $RECONOS/designs/hwtsim/hwtsim.xise
```

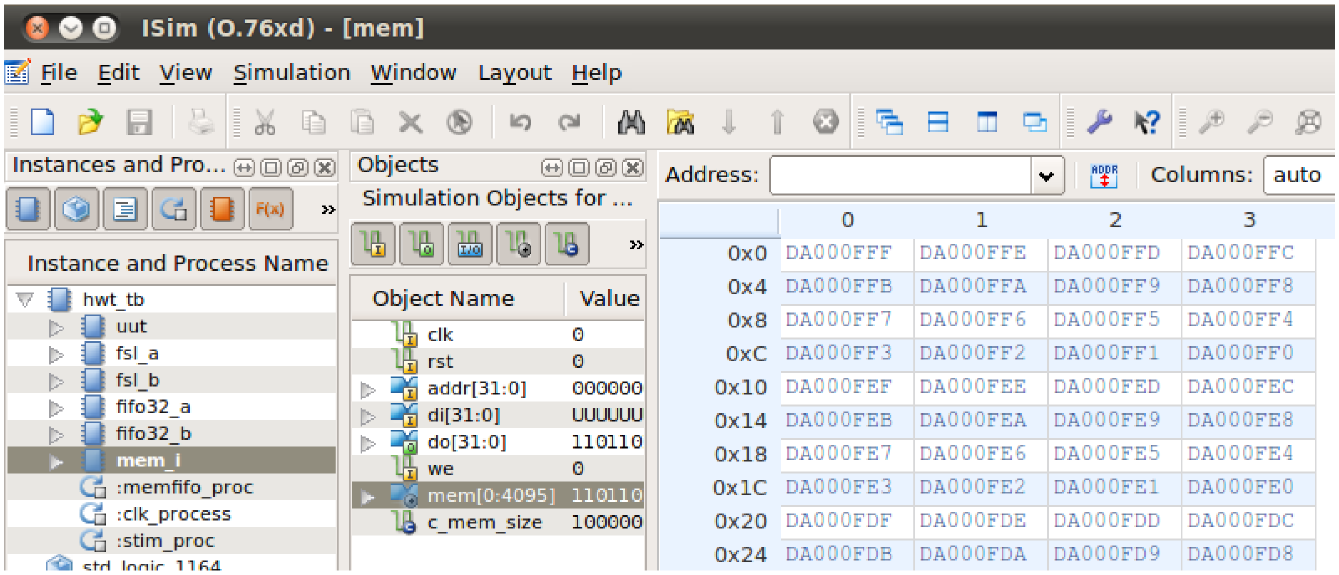
Note: hwt_memaccess.vhd is in '$RECONOS/demos/memtest/hw/hwt_memaccess_v1_00_b/hdl/vhdl'
Change to Simulation View, select test bench 'hwt_tb',
run 'Simulate Behavioral Model' to start ISim.


ISE Project Navigator (O.76xd) - /data/work/reconos_v3/designs/hwtsim/hwtsim.xise
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Behavioral
Hierarchy
hwtsim
xc6vlx240t-1ff784
Unassigned User Library Modules
hwt_tb - behavior (hwt_tb.vhd)
uut - hwt_sort_demo - implementation (hwt_sort_demo.vhd)
sorter_i - bubble_sorter - Behavioral (bubble_sorter.vhd)
No Processes Running
Processes: hwt_tb - behavior
ISim Simulator
Behavioral Check Syntax
Simulate Behavioral Model
Start
Design
Files
Libraries

**In ISim:**
Run 'Simulation>restart'
Run 'File>Open' and open 'hwtsim.wcfg'. In the opening dialog select 'Connect to Existing'.
Enter '1.0 us' as run time and select 'Simulation>Run'



In 'Instance and Process name' window select 'hwt_tb>mem_i'.
Right-click 'mem[0:4095]' object and start the 'Memory Editor'. Select 'Hexadecimal' as value radix.
Confirm that the memory is initialized with the data X“DA000FFF” -> X“DA000000”



| Object Name | Value |
|---|---|
| clk | 0 |
| rst | 0 |
| addr[31:0] | 000000 |
| di[31:0] | UUUUUU |
| do[31:0] | 110110 |
| we | 0 |
| mem[0:4095] | 110110 |
| c_mem_size | 100000 |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0x0 | DA000FFF | DA000FFE | DA000FFD | DA000FFC |
| 0x4 | DA000FFB | DA000FFA | DA000FF9 | DA000FF8 |
| 0x8 | DA000FF7 | DA000FF6 | DA000FF5 | DA000FF4 |
| 0xC | DA000FF3 | DA000FF2 | DA000FF1 | DA000FF0 |
| 0x10 | DA000FEF | DA000FEE | DA000FED | DA000FEC |
| 0x14 | DA000FEB | DA000FEA | DA000FE9 | DA000FE8 |
| 0x18 | DA000FE7 | DA000FE6 | DA000FE5 | DA000FE4 |
| 0x1C | DA000FE3 | DA000FE2 | DA000FE1 | DA000FE0 |
| 0x20 | DA000FDF | DA000FDE | DA000FDD | DA000FDC |
| 0x24 | DA000FDB | DA000FDA | DA000FD9 | DA000FD8 |

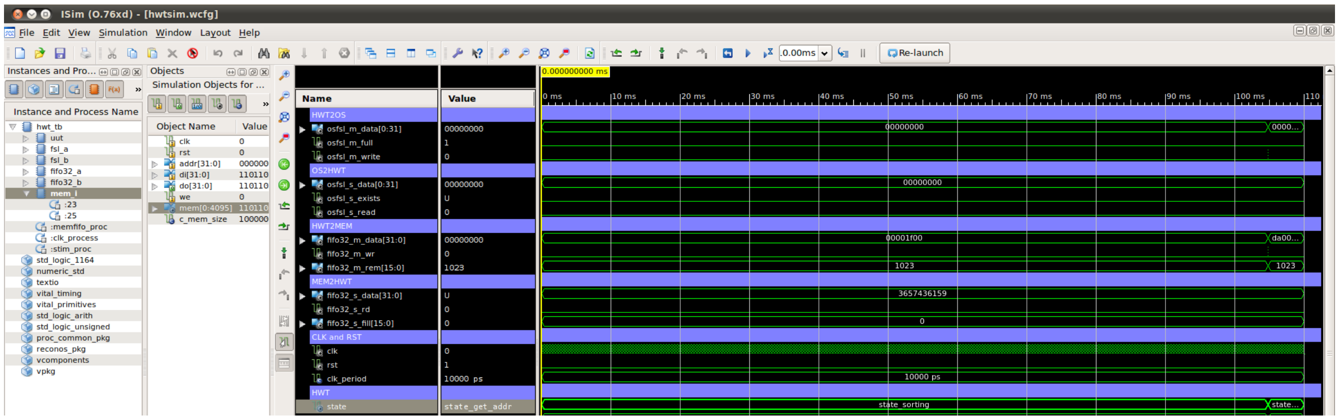
Close the internal 'Memory Editor' window (not ISim).
Enter '110.0 ms' as run-time and select 'Simulation>Run' (this takes about 9 minutes).
Note: Do not wait for the simulation to finish, but continue with the next step!

## 4 Review source code of software application and hardware thread

Software application:
```
> export RECONOS=$HOME/reconos_v3
> gedit $RECONOS/demos/sort_demo/linux/sort_demo.c &
```
*Hardware thread:*
```
> gedit
$RECONOS/demos/sort_demo/hw/hwt_sort_demo_v1_00_b/hdl/vhdl/hwt_sort_d
emo.vhd &
```

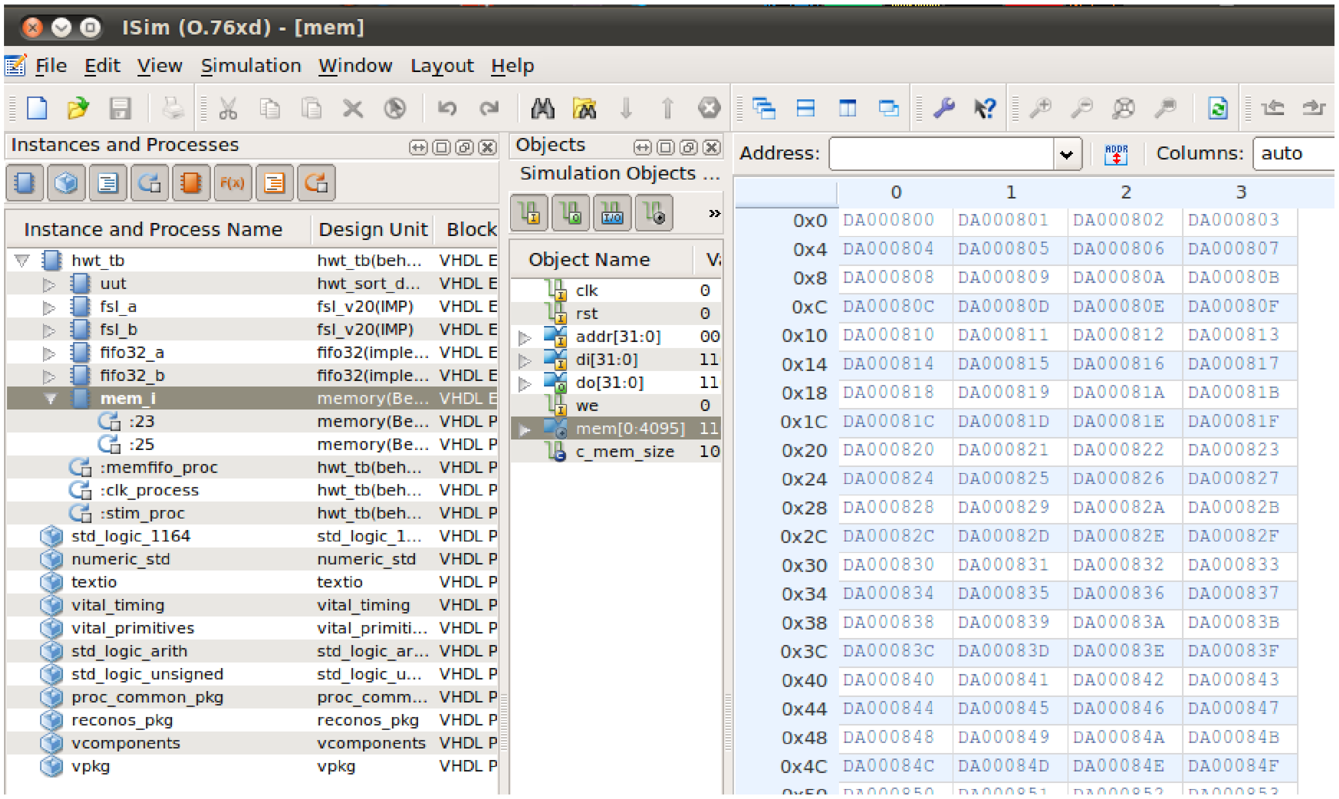
## 5. Review simulation of hardware thread (ISim)

Adjust the time window (zoom out)
Review the signals of the hardware thread.



| Name | Value |
|---|---|
| HWT2OS | |
| osfsl_m_data[0:31] | 00000000 |
| osfsl_m_full | 1 |
| osfsl_m_write | 0 |
| OS2HWT | |
| osfsl_s_data[0:31] | 00000000 |
| osfsl_s_exists | U |
| osfsl_s_read | 0 |
| HWT2MEM | |
| fifo32_m_data[31:0] | 00000000 |
| fifo32_m_wr | 0 |
| fifo32_m_rem[15:0] | 1023 |
| MEM2HWT | |
| fifo32_s_data[31:0] | U |
| fifo32_s_rd | 0 |
| fifo32_s_fill[15:0] | 0 |
| CLK and RST | |
| clk | 0 |
| rst | 1 |
| clk_period | 10000 ps |
| HWT | |
| state | state_get_addr |

Verify in the Memory Editor that the first half of the memory is sorted “DA000800” -> “DA000FFF”.



| Instance and Process Name | Design Unit | Block |
|---|---|---|
| hwt_tb | hwt_tb(beh... | VHDL E |
| uut | hwt_sort_d... | VHDL E |
| fsl_a | fsl_v20(IMP) | VHDL E |
| fsl_b | fsl_v20(IMP) | VHDL E |
| fifo32_a | fifo32(imple... | VHDL E |
| fifo32_b | fifo32(imple... | VHDL E |
| mem_i | memory(Be... | VHDL E |
| :23 | memory(Be... | VHDL P |
| :25 | memory(Be... | VHDL P |
| :memfifo_proc | hwt_tb(beh... | VHDL P |
| :clk_process | hwt_tb(beh... | VHDL P |
| :stim_proc | hwt_tb(beh... | VHDL P |
| std_logic_1164 | std_logic_1... | VHDL P |
| numeric_std | numeric_std | VHDL P |
| textio | textio | VHDL P |
| vital_timing | vital_timing | VHDL P |
| vital_primitives | vital_primiti... | VHDL P |
| std_logic_arith | std_logic_ar... | VHDL P |
| std_logic_unsigned | std_logic_u... | VHDL P |
| proc_common_pkg | proc_comm... | VHDL P |
| reconos_pkg | reconos_pkg | VHDL P |
| vcomponents | vcomponents | VHDL P |
| vpkg | vpkg | VHDL P |

| Object Name | V |
|---|---|
| clk | 0 |
| rst | 0 |
| addr[31:0] | 00 |
| di[31:0] | 11 |
| do[31:0] | 11 |
| we | 0 |
| mem[0:4095] | 11 |
| c_mem_size | 10 |

Address: Columns: auto

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0x0 | DA000800 | DA000801 | DA000802 | DA000803 |
| 0x4 | DA000804 | DA000805 | DA000806 | DA000807 |
| 0x8 | DA000808 | DA000809 | DA00080A | DA00080B |
| 0xC | DA00080C | DA00080D | DA00080E | DA00080F |
| 0x10 | DA000810 | DA000811 | DA000812 | DA000813 |
| 0x14 | DA000814 | DA000815 | DA000816 | DA000817 |
| 0x18 | DA000818 | DA000819 | DA00081A | DA00081B |
| 0x1C | DA00081C | DA00081D | DA00081E | DA00081F |
| 0x20 | DA000820 | DA000821 | DA000822 | DA000823 |
| 0x24 | DA000824 | DA000825 | DA000826 | DA000827 |
| 0x28 | DA000828 | DA000829 | DA00082A | DA00082B |
| 0x2C | DA00082C | DA00082D | DA00082E | DA00082F |
| 0x30 | DA000830 | DA000831 | DA000832 | DA000833 |
| 0x34 | DA000834 | DA000835 | DA000836 | DA000837 |
| 0x38 | DA000838 | DA000839 | DA00083A | DA00083B |
| 0x3C | DA00083C | DA00083D | DA00083E | DA00083F |
| 0x40 | DA000840 | DA000841 | DA000842 | DA000843 |
| 0x44 | DA000844 | DA000845 | DA000846 | DA000847 |
| 0x48 | DA000848 | DA000849 | DA00084A | DA00084B |
| 0x4C | DA00084C | DA00084D | DA00084E | DA00084F |

## 6. Download bitstream to FPGA

Continue with this step when the bitfile (step 2) has been generated.
Switch the FPGA board on.

```
> export PATH=$PATH:$RECONOS/tools
> xilinx_firmware.sh
> dow $RECONOS/demos/sort_demo/hw/edk/implementation/system.bit
```

## 7. Start minicom on a different terminal

```
> minicom
```

Set serial port [STRG]+[A], [O], select 'serial port setup', [A], type: '/dev/ttyUSB0', 2x[ENTER],[ESC]
Set the correct baudrate: [STRG]+[A], [P], [C], [ENTER]

## 8. Compile Linux kernel and download it to FPGA

```
> cd $HOME/linux-2.6-xlnx
> make ARCH=microblaze CROSS_COMPILE=microblaze-unknown-linux-gnu-
simpleImage.ml605_epics
> dow $HOME/linux-2.6-
xlnx/arch/microblaze/boot/simpleImage.ml605_epics
```

## 9. Run application in minicom

```
> ./load_fsl.sh
> ./load_getpgd.sh
> ./sort_demo <#hw threads> <#sw threads> <#blocks>
```

Note: The number of blocks <*#blocks*> has to be a power of two.
Test different numbers of hw threads <*#hw threads*> and sw threads <*#sw threads*>.
Try: `./sort_demo 1 0 128`
Evaluate the speedup of different numbers of hw threads.

## Webcam-Demo

The webcam demo transfers the frame of a webcam (that is connected to a personal computer) to the FPGA using a TCP/IP connection. The frame can then be manipulated at the FPGA, before it is send back to the PC. The original frame and the manipulated frame are displayed together at the graphical user interface of the application.

### 1. Review source code of software application and hardware thread

Software application:
```
> gedit $RECONOS/demos/webcam/linux/webcam_demo.c &
> gedit $RECONOS/demos/webcam/linux/filter.c &
```

Hardware threads:
```
> gedit
$RECONOS/demos/webcam/hw/hwt_graphical_filter_v1_00_a/hdl/vhdl/hwt_gr
aphical_filter.vhd &
> gedit
$RECONOS/demos/webcam/hw/hwt_graphical_filter_v1_00_b/hdl/vhdl/hwt_gr
aphical_filter.vhd &
```

### *2. Create your own hardware and software threads to manipulate the frame*

Be creative and apply either graphical filters or transformations to the frame (i.e. mirror the frame). For hardware threads: Try different ReconOS API functions to transfer the frame into the local RAM. As the entire frame does not fit into the local RAM you might need to update it line by line. Each pixel is coded with 4 bytes. The 4-th byte is not used, the other three bytes represent rgb color values (red-green-blue).

### 3. Create software design

```
> cd $RECONOS/demos/webcam/linux
> make clean all
> cp $RECONOS/demos/webcam/linux/webcam_demo /exports/rootfs_mb/.
```

### 4. Create graphical interface

```
> cd $RECONOS/demos/webcam/gui
> make clean all
```

### 5. Create hardware design

```
> cd $RECONOS/demos/webcam/hw
> ./setup.sh
> cd edk
> make clean all
```

## 6. Download bitstream to FPGA

```
> export PATH=$PATH:$RECONOS/tools
> xilinx_firmware.sh
> dow $RECONOS/demos/webcam/hw/edk/implementation/system.bit
```

## 7. Start minicom on a different terminal

```
> minicom
```

Set serial port [STRG]+[A], [O], select 'serial port setup', [A], type: '/dev/ttyUSB0', 2x[ENTER],[ESC]
Set the correct baudrate: [STRG]+[A], [P], [C], [ENTER]

## 8. Download Linux kernel to FPGA

```
> dow $HOME/simpleImage.ml605_epics
```

## 9. Run application in minicom

```
> ./load_fsl.sh
> ./load_getpgd.sh
> ./webcam_demo
```

## 10. Start graphical interface (other terminal)

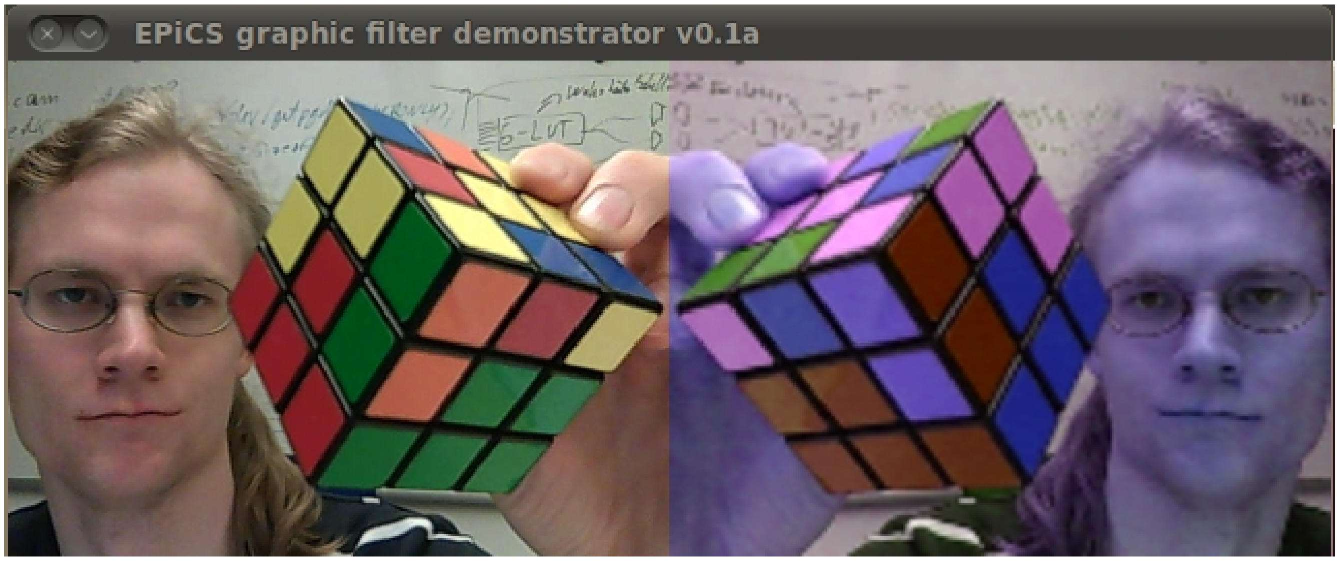
Connect the webcam to the PC. Then, start the graphical interface.

```
> cd $RECONOS/demos/webcam/gui/bin
> ./webcam 192.168.x.2
```

Note: x=50+id of your computer. (i.e.: x=50+08=58)

### *Example for frame manipulation*

*mirror frame and switch color components*


EPiCS graphic filter demonstrator v0.1a