



# ***Développez vos systèmes embarqués sur SoC FPGA***

***Comment embarquer Linux et développer en VHDL vos applicatifs dédiés au traitement d'image sans pénaliser le CPU.***

**Jean-Marie CODOL**  
Développeur

Submarine Open Technologies  
Montpellier

Avril 2022



# ***Développez vos systèmes embarqués sur SoC FPGA***

***Comment embarquer Linux et développer en VHDL vos applicatifs dédiés au traitement d'image sans pénaliser le CPU.***

***Partie 6***

# Plan

## Plan

JOUR 1

§001 Faire communiquer Linux avec un FPGA sur le SoC intel DE10-nano

- distribution fournie par Altera
- créer une image SD (en partie sur un serveur distant)
- compilation croisée (eclipse CDT embedded)
- activer les bridges HPS <-> FPGA par un device tree
- utiliser le bridge HPS2FPGA

J 2

§002 Partager une zone de RAM entre Linux et un FPGA sur le SoC intel DE10-nano

- **projet de lecture/écriture en RAM sur FPGA**
- projet de lecture/écriture en RAM sur CPU

J 3

§003 Cas pratique avec seuillage d'image

- projet openCV
- échange sur mémoire RAM



# Décharger le CPU d'une opération en un port classique

Limitations du bus de communication parallèle,  
avantages d'utiliser la mémoire RAM



# Décharger le CPU en utilisant la RAM

## Echanges entre HPS et FPGA

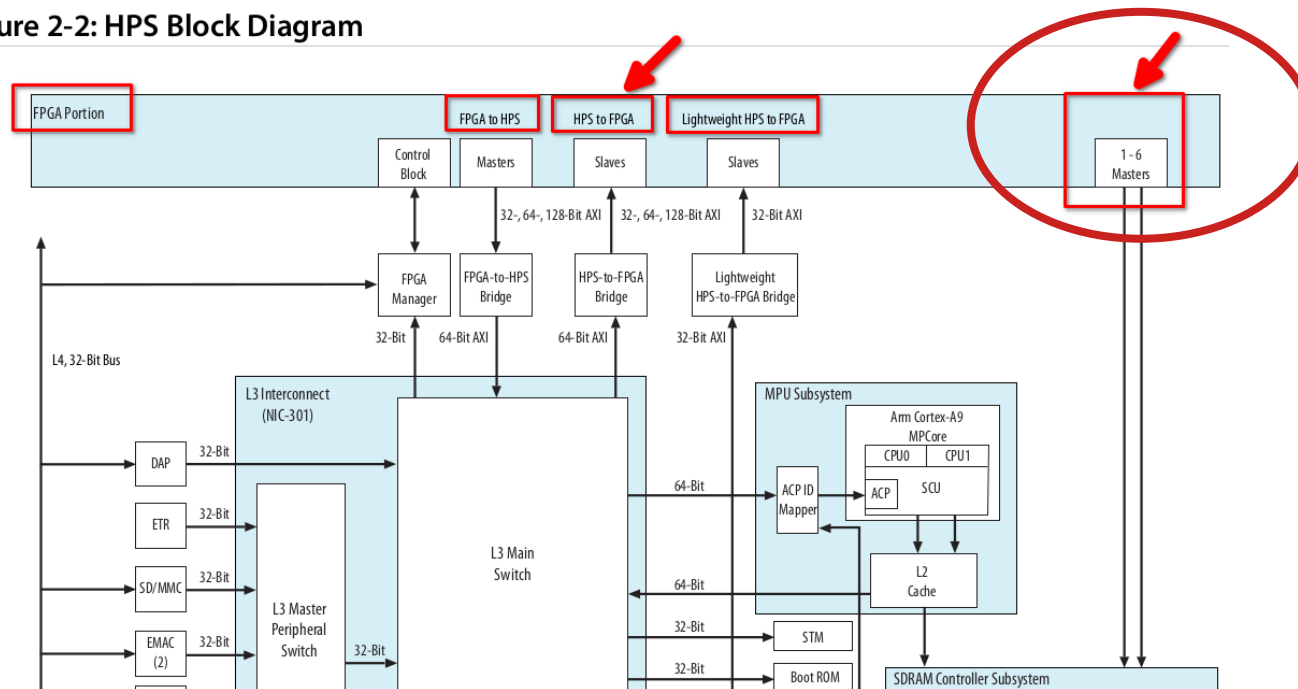
Rappel

2-4 HPS Block Diagram and System Integration

### HPS Block Diagram and System Integration

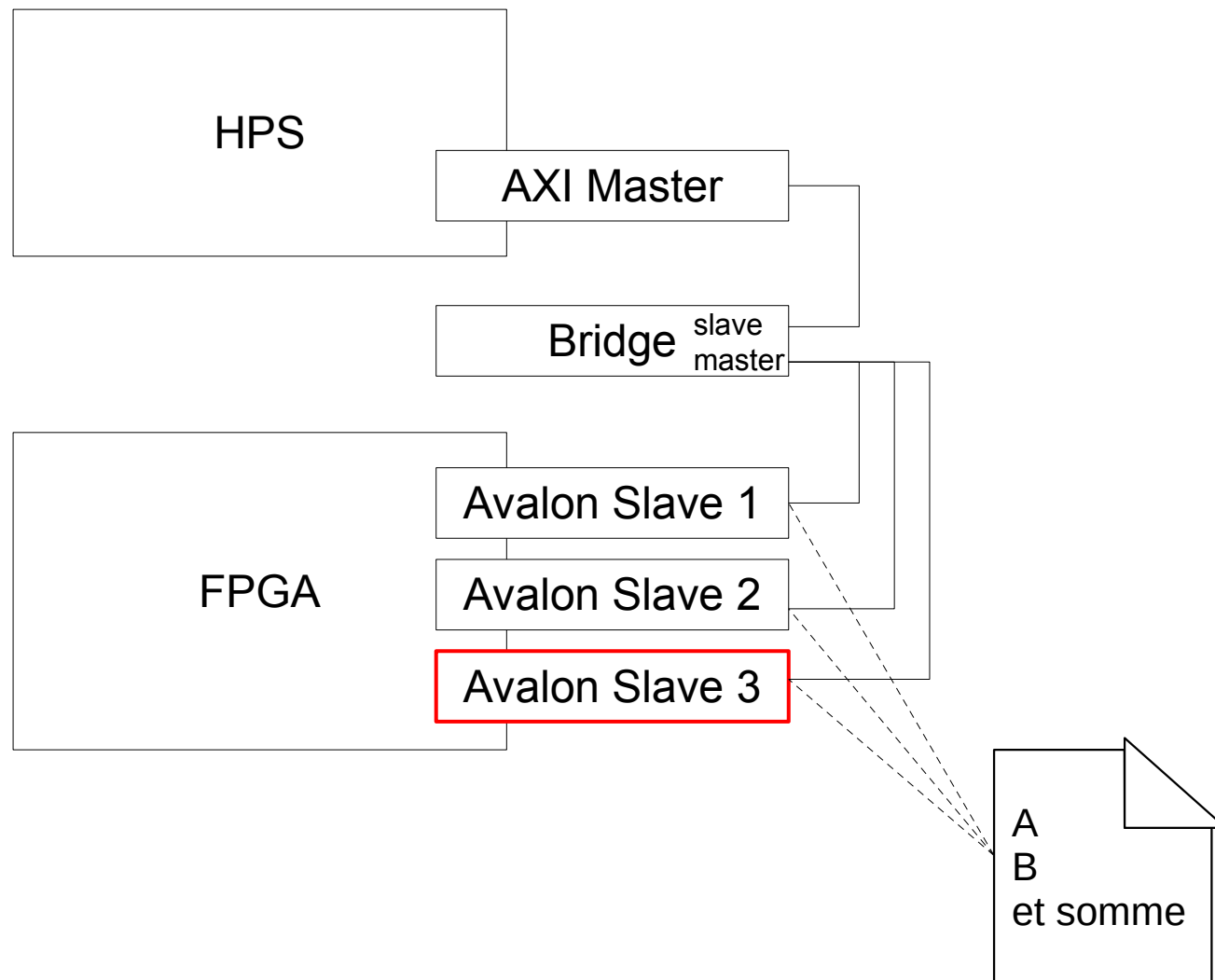
#### HPS Block Diagram

Figure 2-2: HPS Block Diagram



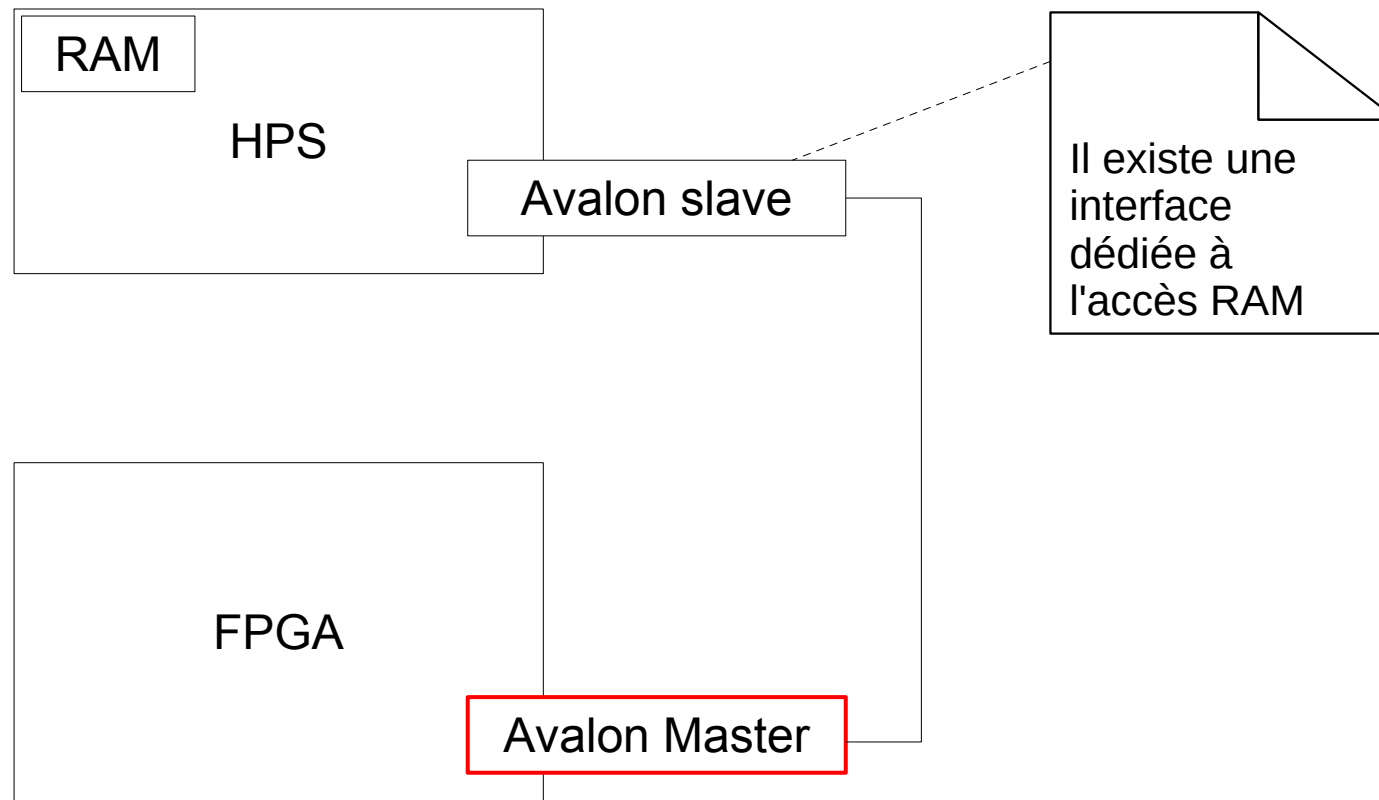
# Décharger le CPU en utilisant la RAM

## Echanges entre HPS et FPGA



# Décharger le CPU en utilisant la RAM

## Echanges entre HPS et FPGA



# Décharger le CPU en utilisant la RAM

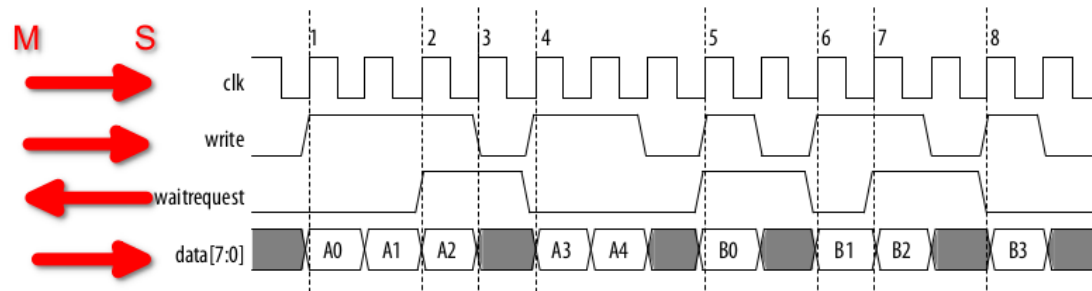
## Echanges entre HPS et FPGA

Rappel

### 3.5.2.2. waitrequestAllowance Equals One

The following timing diagram illustrates timing for an Avalon-MM host that has one clock cycle to start and stop sending transfers after the Avalon-MM agent deasserts or asserts `waitrequest`, respectively:

Figure 9. Host Write: `waitrequestAllowance` Equals One Clock Cycle



The numbers in this figure mark the following events:

1. The Avalon-MM host drives `write` and `data`.
2. The Avalon-MM agent asserts `waitrequest`. Because the `waitrequestAllowance` is 1, the host can complete the write.
3. The host deasserts `write` because the agent is asserting `waitrequest` for a second cycle.
4. The Avalon-MM host drives `write` and `data`. The agent is not asserting `waitrequest`. The writes complete.
5. The agent asserts `waitrequest`. Because the `waitrequestAllowance` is 1 cycle, the write completes.



# Décharger le CPU en utilisant la RAM

## Réserver une zone de RAM aux échanges entre HPS et FPGA

Dans la partition du noyau Linux, modifier extlinux.conf pour que Linux n'utilise que 512 Mo de RAM  
Sur le SoC

```
mkdir fat
mount /dev/mmcblk0p1 fat
vim fat/extlinux/extlinux.conf
```

## ajouter sur la dernière ligne net.ifnames et mem pour obtenir

```
LABEL Linux Default
    KERNEL ../zImage
    FDT ../socfpga_cyclone5_de0_nano_soc.dtb
    APPEND root=/dev/mmcblk0p2 rw rootwait earlyprintk console=ttyS0,115200n8 net.ifnames=0 mem=512M
```

```
umount fat
```

# Décharger le CPU en utilisant la RAM

## Configurer Platform Designer

Démarrer du projet Quartus "ghrd\_barebones\_template"

```
export DEWD=$HOME/de10nano-wd  
cd $DEWD/captronic_formation_fpga_img_proc/quartus_proj  
cp -r ghrd_barebones_template ghrd_sdram
```

Ouvrir le projet dans Quartus

```
~/intelFPGA_lite/20.1/quartus/bin/quartus&
```

# Décharger le CPU en utilisant la RAM

## Le composant Trigger

Récupérer et adapter du projet SimpleAdder le fichier VHDL hps\_out\_32bit.vhd qui servira de déclencheur.

Une copie se trouve dans

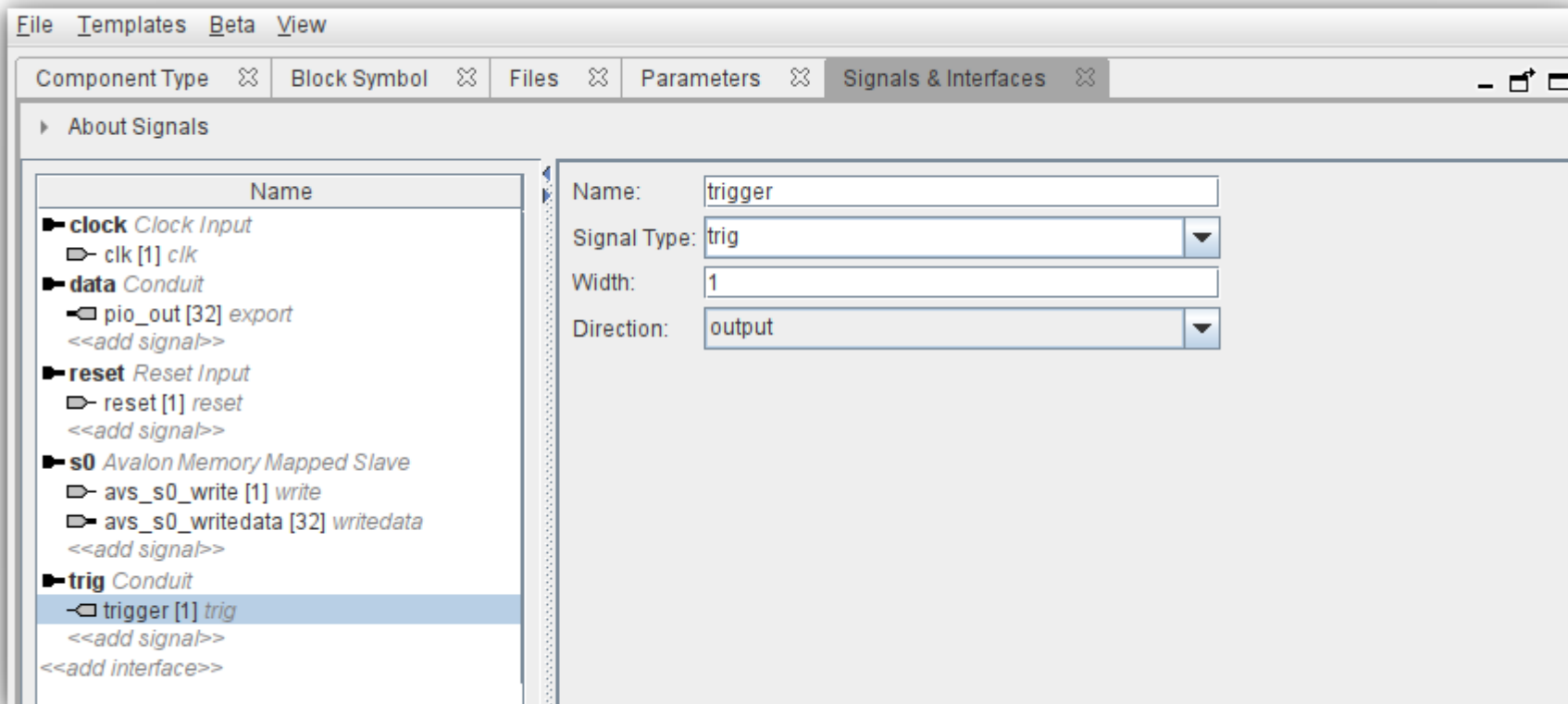
<captronic\_formation\_fpga\_img\_proc>/quartus\_proj/fichiers\_exemples/3-ram/VHDL hps\_out\_32bit.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity hps_out_32bit is
5  port
6  (
7      clk          : in std_logic;
8      reset        : in std_logic;
9      avs_s0_write : in std_logic;
10     avs_s0_writedata : in std_logic_vector(31 downto 0) ;
11     pio_out       : out std_logic_vector(31 downto 0) ;
12     trigger       : out std_logic
13 );
14
15 end entity;
16
17 architecture rtl of hps_out_32bit is
18 begin
19
20     process (clk)
21     begin
22         if (rising_edge(clk)) then
23             if (reset = '1') then
24                 pio_out <= "00000000000000000000000000000000" ;
25             elsif (avs_s0_write = '1') then
26                 pio_out <= avs_s0_writedata ;
27             --else
28             -- pio_out <= "00000000000000000000000000000000" ;
29             end if;
30             trigger <= avs_s0_write ;
31         end if;
32     end process;
33 end rtl;
```

# Décharger le CPU en utilisant la RAM

## Le composant Trigger

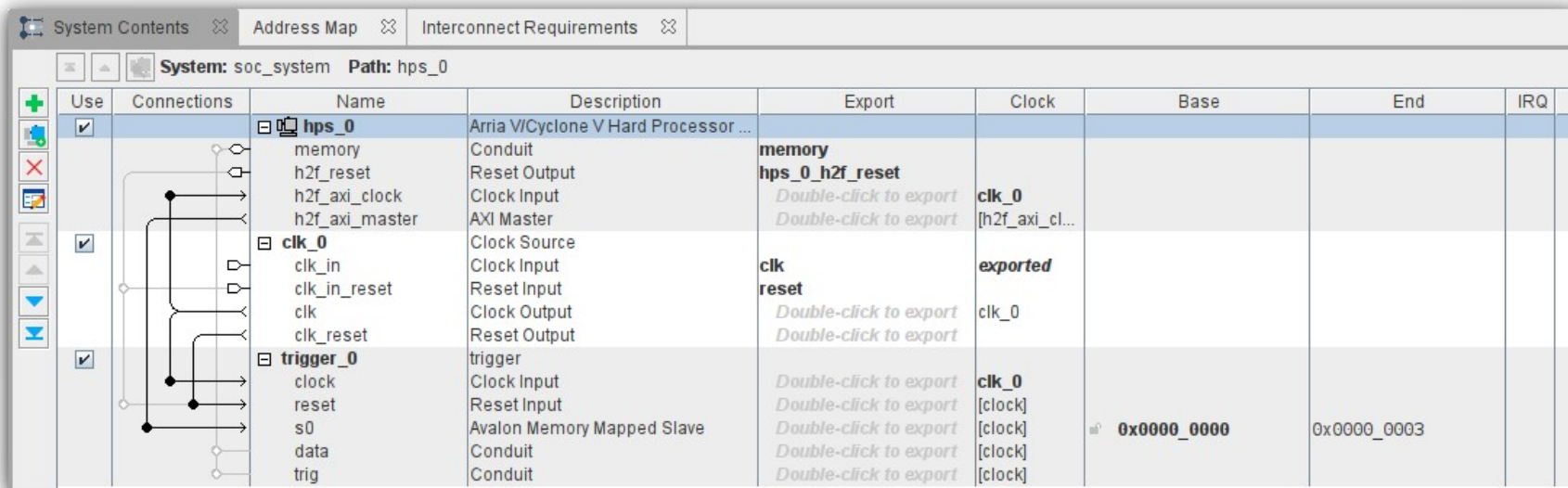
Paramétrer Platform Designer pour obtenir un composant coté FPGA qui peut recevoir une donnée du HPS.



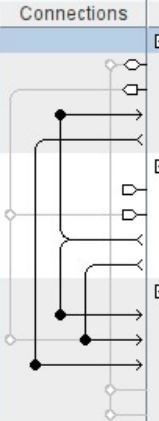
# Décharger le CPU en utilisant la RAM

## Le composant Trigger

Paramétrer Platform Designer pour obtenir un composant coté FPGA qui peut recevoir une donnée du HPS.  
À la génération HDL, les ports de soc\_system.v ne changent pas.



The screenshot shows the 'System Contents' window in Platform Designer. The 'System' is 'soc\_system' and the 'Path' is 'hps\_0'. The table lists the components and their connections:

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		<b>hps_0</b> memory h2f_reset h2f_axi_clock h2f_axi_master	Arria V/Cyclone V Hard Processor ... Conduit Reset Output Clock Input AXI Master	<b>memory</b> <b>hps_0_h2f_reset</b> <i>Double-click to export</i> <i>Double-click to export</i>	<b>clk_0</b> [h2f_axi_cl...			
<input checked="" type="checkbox"/>		<b>clk_0</b> clk_in clk_in_reset clk clk_reset	Clock Source Clock Input Reset Input Clock Output Reset Output	<b>clk</b> <b>reset</b> <i>Double-click to export</i> <i>Double-click to export</i>	<b>exported</b> clk_0			
<input checked="" type="checkbox"/>		<b>trigger_0</b> clock reset s0 data trig	Trigger Clock Input Reset Input Avalon Memory Mapped Slave Conduit Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	<b>clk_0</b> [clock] [clock] [clock]	<b>0x0000_0000</b>	0x0000_0003	

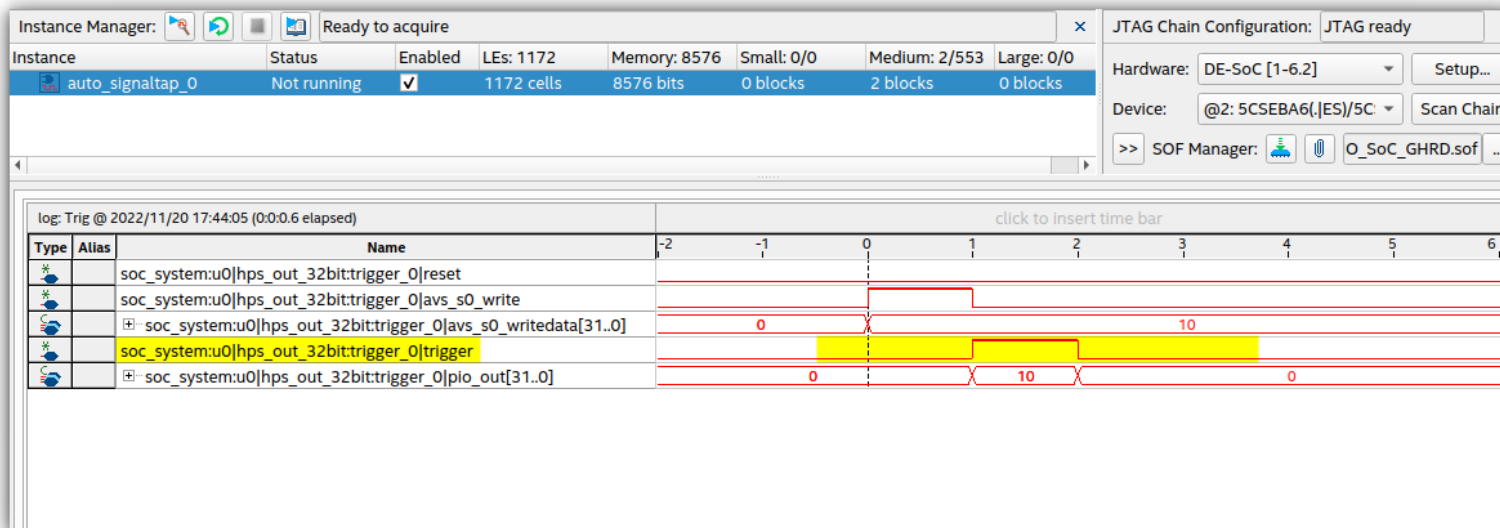


# Décharger le CPU en utilisant la RAM

## Le composant Trigger

Débogage des signaux avec Signal Tap

Avant la correction du bug de maintien de pio\_out



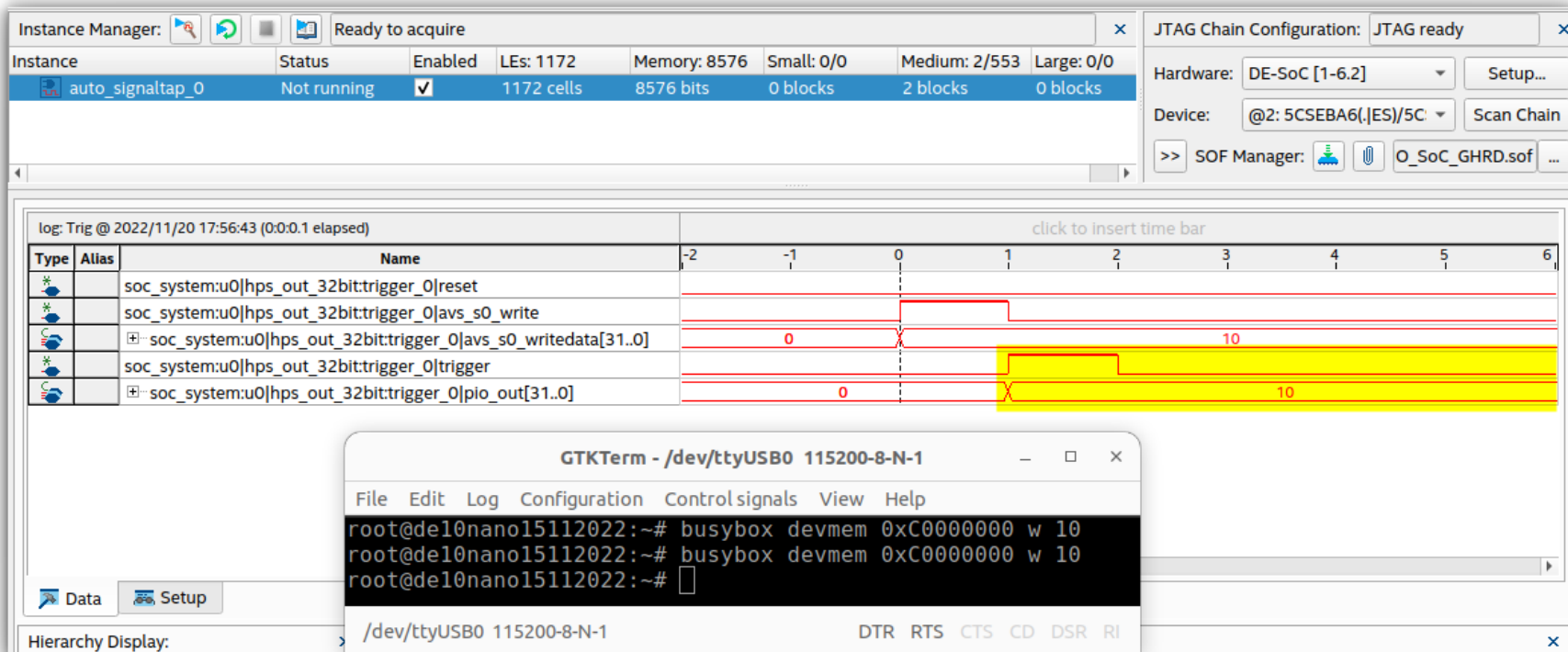
```
GTKTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@de10nano15112022:~#
root@de10nano15112022:~# busybox devmem 0xC0000000 w 10
root@de10nano15112022:~# 
/dev/ttyUSB0 115200-8-N-1 DTR RTS CTS CD DSR RI
```

# Décharger le CPU en utilisant la RAM

## Le composant Trigger

Débogage des signaux avec Signal Tap

Après la correction du bug de maintien.



# Décharger le CPU en utilisant la RAM

## Le composant d'écriture dans la ram

Le composant s'écrit en VHDL

Une copie du fichier se trouve dans

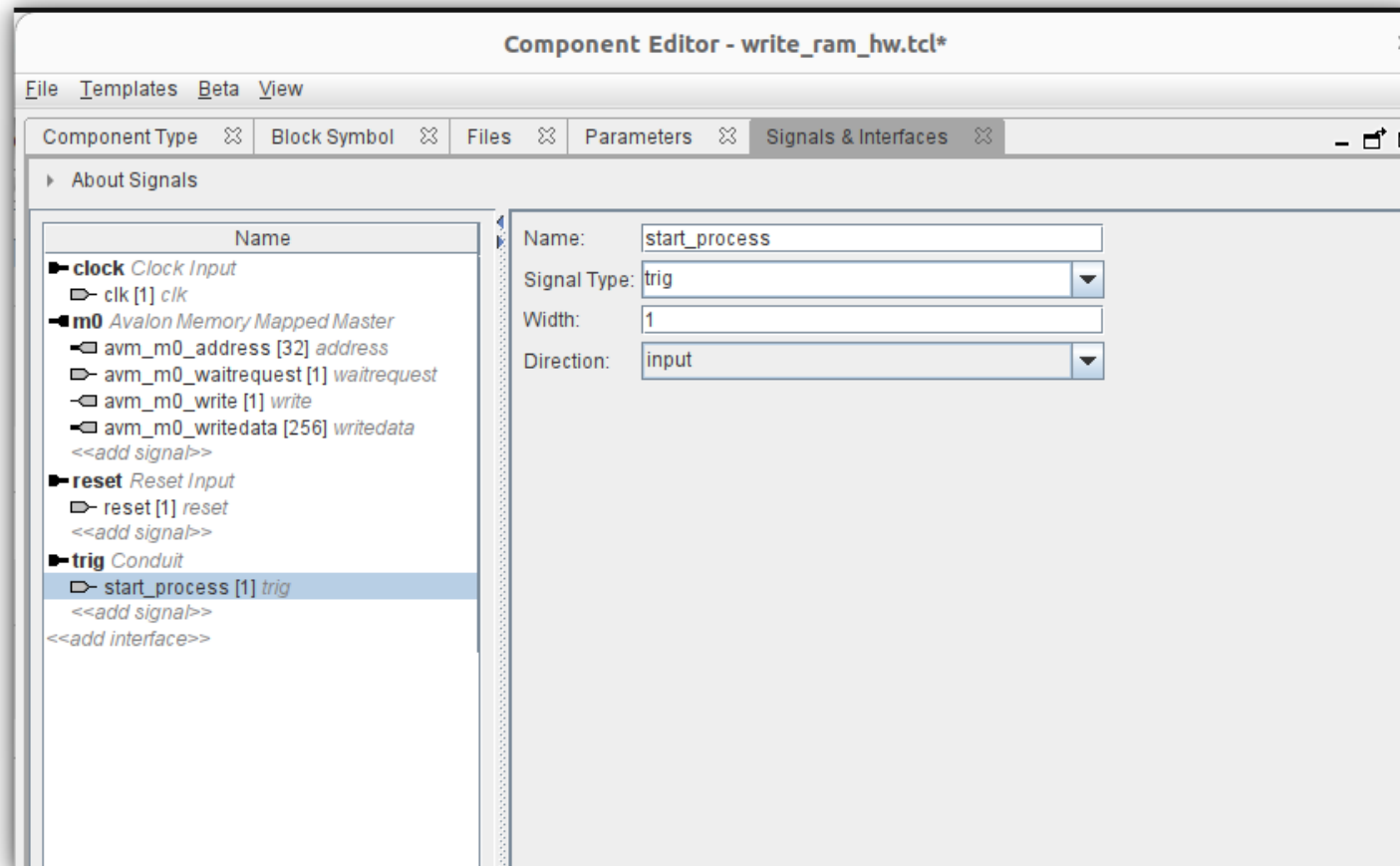
<captronic\_formation\_fpga\_img\_proc>/quartus\_proj/fichiers\_exemples/3-ram/write\_ram.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ENTITY write_ram IS
6  PORT (
7      reset : IN STD_LOGIC := '0';
8      clk : IN STD_LOGIC;
9      start_process : IN STD_LOGIC := '0';
10     avm_m0_write : OUT STD_LOGIC;
11     avm_m0_address : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
12     avm_m0_writedata : OUT STD_LOGIC_VECTOR(255 DOWNTO 0);
13     avm_m0_waitrequest : IN STD_LOGIC := '0'
14     --avm_m0_byteenable : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
15     --avm_m0_burstcount : OUT STD_LOGIC_VECTOR(10 DOWNTO 0);
16     --out_data : OUT STD_LOGIC_VECTOR(255 DOWNTO 0);
17 );
18 END write_ram;
19
20 ARCHITECTURE BEHAVIOR OF write_ram IS
21     signal avm_m0_write_sig : std_logic ;
22     signal avm_m0_address_sig : std_logic_vector(31 DOWNTO 0) ;
23     signal avm_m0_writedata_sig : std_logic_vector(255 DOWNTO 0) ;
24 BEGIN
25     process (clk)
26     begin
27         if (rising_edge(clk)) then
28             if (reset = '1') then
29                 avm_m0_write_sig <= '0' ;
30                 avm_m0_address_sig <= std_logic_vector(to_unsigned(0, 32)) ;
31                 avm_m0_writedata_sig <= std_logic_vector(to_unsigned(0, 256)) ;
32             elsif (start_process = '1') then
33                 avm_m0_write_sig <= '1' ;
34                 avm_m0_address_sig <= std_logic_vector(to_unsigned(536870912, 32)) ;
35                 avm_m0_writedata_sig <= std_logic_vector(to_unsigned(55, 256)) ;
36             end if;
```

# Décharger le CPU en utilisant la RAM

## Platform Designer

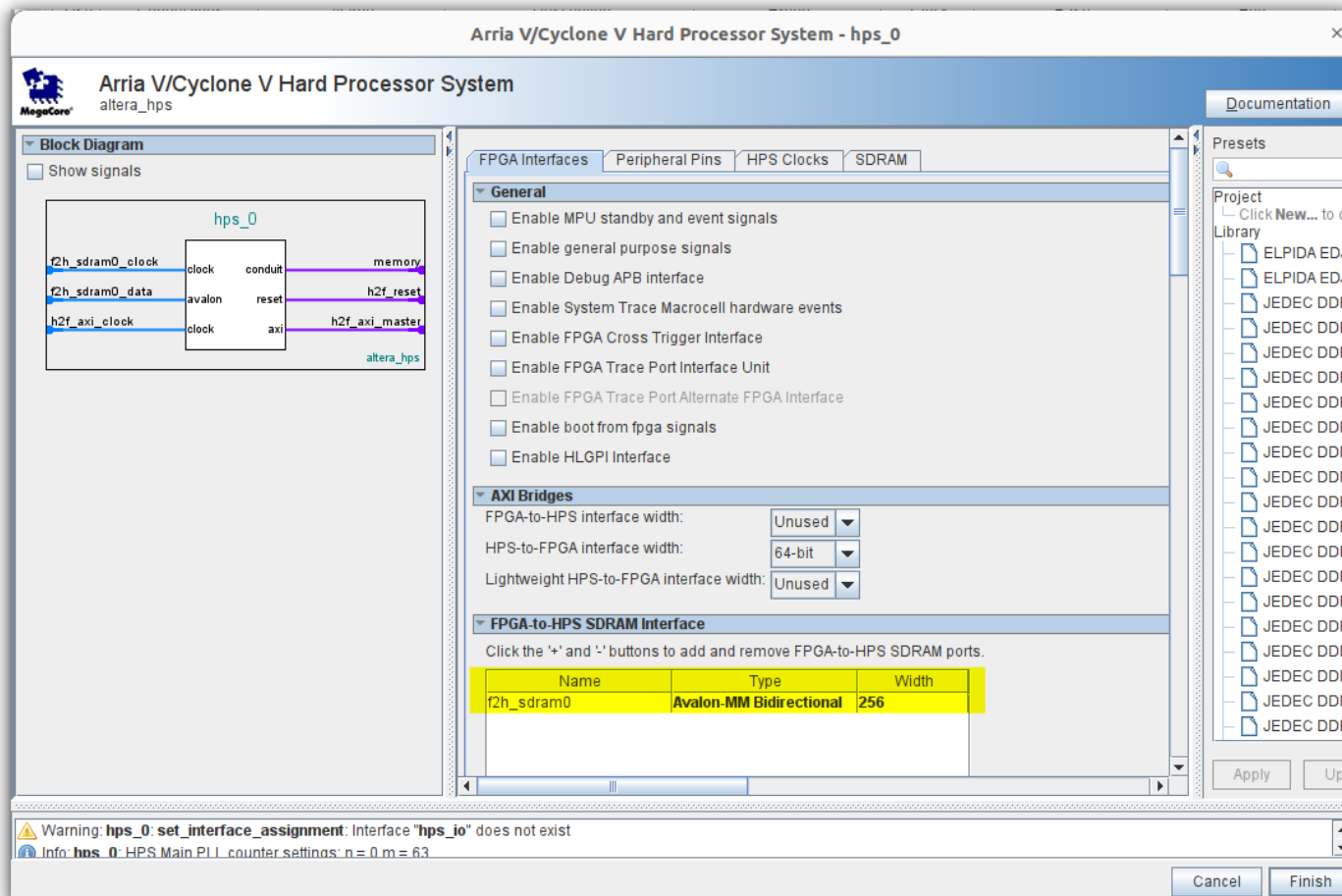
On crée un composant dans Platform Designer



# Décharger le CPU en utilisant la RAM

## Platform Designer

On active un bus d'accès au contrôleur de RAM dans hps\_0



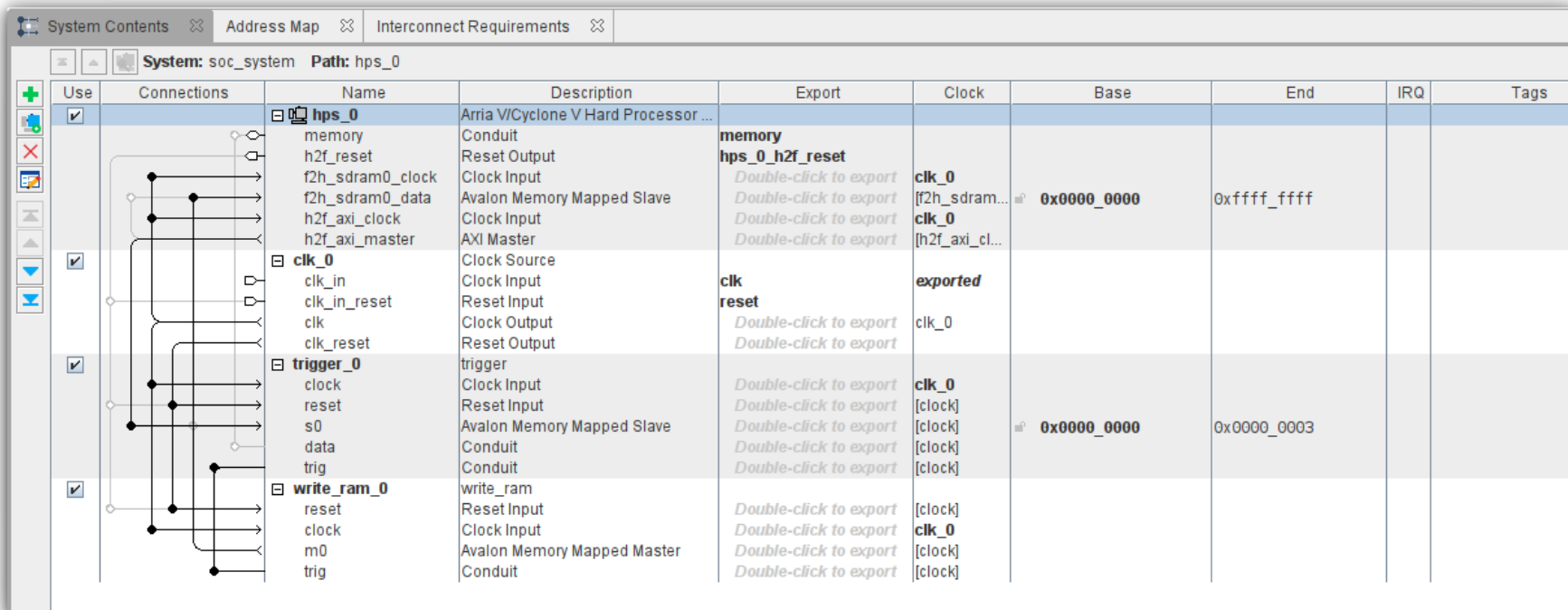


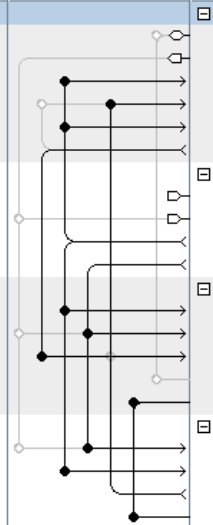
# Décharger le CPU en utilisant la RAM

## Projet Quartus

On instancie ce composant et on connecte les entrées/sorties nécessaires.

On génère les fichier HDL, puis on compile le projet Quartus.



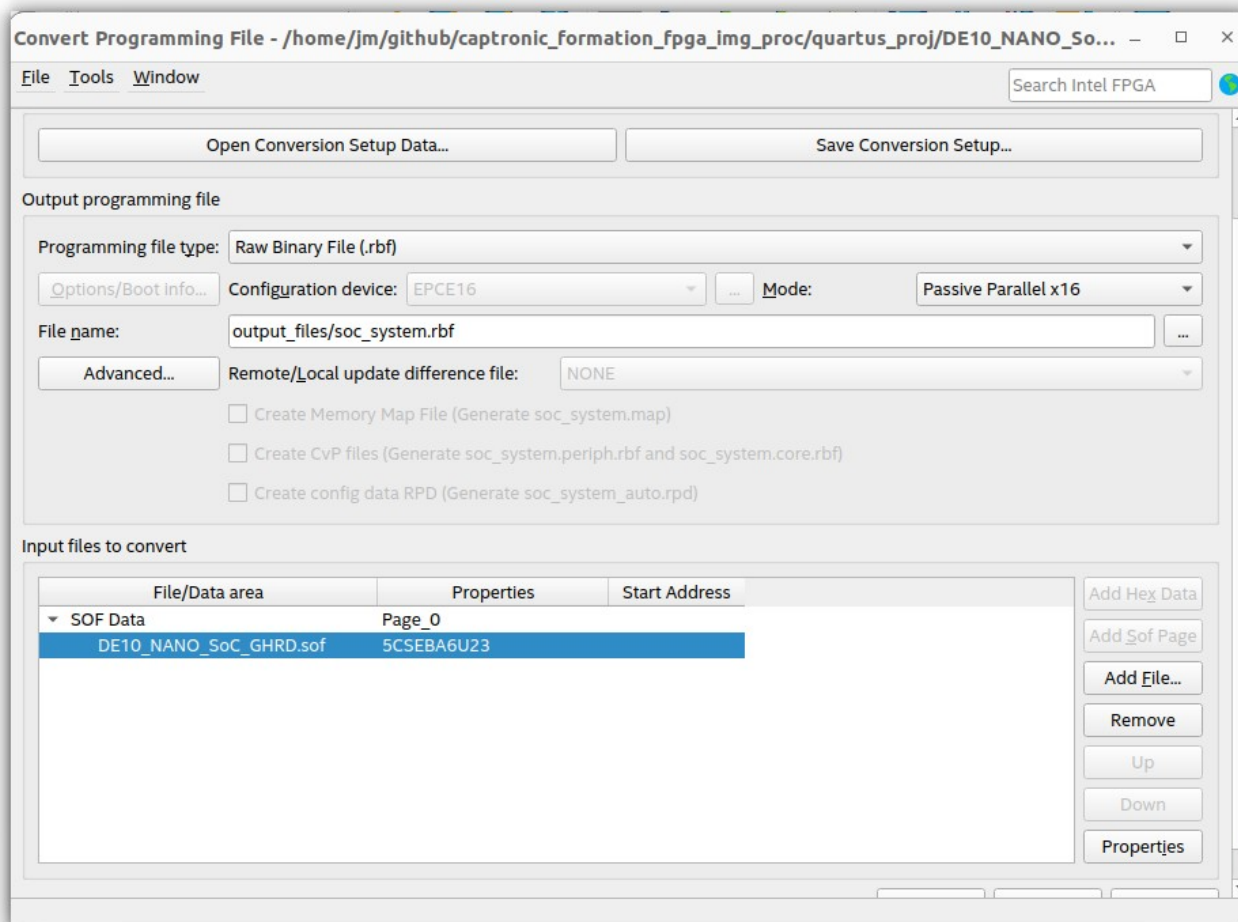
Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
<input checked="" type="checkbox"/>		<b>hps_0</b> memory h2f_reset f2h_sdram0_clock f2h_sdram0_data h2f_axi_clock h2f_axi_master	Arria V/Cyclone V Hard Processor ... Conduit Reset Output Clock Input Avalon Memory Mapped Slave Clock Input AXI Master	<b>memory</b> <b>hps_0_h2f_reset</b> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	<b>clk_0</b> [f2h_sdram... <b>clk_0</b> [h2f_axi_cl...	<b>0x0000_0000</b>	<b>0xffff_ffff</b>		
<input checked="" type="checkbox"/>		<b>clk_0</b> clk_in clk_in_reset clk clk_reset	Clock Source Clock Input Reset Input Clock Output Reset Output	<b>clk</b> <b>reset</b> <i>Double-click to export</i> <i>Double-click to export</i>	<b>exported</b> clk_0				
<input checked="" type="checkbox"/>		<b>trigger_0</b> clock reset s0 data trig	trigger Clock Input Reset Input Avalon Memory Mapped Slave Conduit Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	<b>clk_0</b> [clock] [clock] [clock] [clock]	<b>0x0000_0000</b>	<b>0x0000_0003</b>		
<input checked="" type="checkbox"/>		<b>write_ram_0</b> reset clock m0 trig	write_ram Reset Input Clock Input Avalon Memory Mapped Master Conduit	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i>	[clock] <b>clk_0</b> [clock] [clock]				

# Décharger le CPU en utilisant la RAM

## Projet Quartus

On génère le fichier rbf (raw binary file) avec Quartus (file -> convert programming file).

Ne pas oublier de compresser le rbf (case à cocher) pour que le rbf fasse à peu près 2 Mo.



# Décharger le CPU en utilisant la RAM

## Modifier u-boot

Ajouter à u-boot les commandes qui permettent au cyclone-v d'activer le controleur de RAM pour la partie FPGA.

```
ssh formateur@151.80.152.63
cd <chemin-vers-u-boot>
vi include/config_distro_bootcmd.h
ajouter les instructions suivantes à la variable distro_bootcmd :

mw 0xFFC25080 0x0
fatload mmc 0:1 0x20000000 soc_system.rbf
fpga load 0 0x20000000 0x7000000
mw 0xFFC2505C 0xA
mw 0xFFC25080 0xFFFF

Recompiler u-boot
export ARCH=arm
export CROSS_COMPILE=$HOME/intelFPGA/20.1/embedded/host_tools/linaro/gcc/bin/arm-eabi-
make socfpga_de10_nano_defconfig
make -j 9
# le resultat se trouve dans u-boot-with-spl.sfp

Sur le PC de développement, récupérer u-boot
scp formateur@151.80.152.63:/home/formateur/de10nano-wd/u-boot/u-boot-with-spl.sfp .
# Remplacer u-boot directement sur la carte SD
sudo dd if=u-boot-with-spl.sfp of=/dev/sdhXXXX bs=64k seek=0 oflag=sync
sync
sudo sync

rebooter avec la carte SD
```

# Décharger le CPU en utilisant la RAM

## Modifier u-boot

Explications ([lien](#) d'un tuto qui référence [cet](#) article)

1. `mw 0xFFC25080 0x0`

0xFFC25080 est l'adresse du registre de 32 bit **fpgaportrst** (qui fait partie du module "sdr" qui gère la sdram) Dans le document de référence, il est indiqué en section 2-67 que ce registre permet de faire un "reset" des ports sdram.

Quand on les met à zéro, le port correspondant entre en mode "reset" et n'en sortira qu'à l'écriture d'un 1.

Bits 3 :0 => read data ports 3,2,1, et 0.

Bits 7 :4 => write data ports 3,2,1, et 0.

Bits 13:8 => command ports 3,2,1, et 0.

Il n'y a que 14 bits utiles, le reste est réservé.

2. **fatload mmc 0:1 0x2000000 sdram.rbf**

Charge sdram.rbf en ram à l'adresse 0x2000000.

3. **fpga load 0 0x2000000 0x700000**

Charge le contenu de la ram (0x2000000) sur le fpga (taille 0x700000 = taille max).

`mw 0xFFC2505C 0xA`

0xFFC2505C : registre **staticcfg**. bit 3 : applycfg => "apply all the settings loaded in SDR".

md 0xFFC2505C lit la valeur actuelle du registre (oco 0x2), on ne change que le bit 3 ce qui donne 0xA.

`mw 0xFFC25080 0xFFFF`

On désactive le reset de l'étape 1 : **fpgaportrst**.

boot

démarre le noyau Linux en passant les arguments standards.

# Décharger le CPU en utilisant la RAM

## Modifier u-boot

Verifier le soft FPGA.

The screenshot shows a web browser window with the title "Important Note about FPGA/HPS SDRAM Bridge - MitySOM-5CSX Altera Cyclone V - Critical Link Support - Brave". The address bar shows the URL "support.criticallink.com/redmine/projects/mityarm-5cs/wiki/Important\_Note\_about\_FPGA\_HPS\_SDRAM\_Bridge". The page content is from a Redmine wiki and is titled "Important Note about FPGA/HPS SDRAM Bridge (2013-12-13)".

Accueil Projets Redmine shortcuts

Mity CPU Platforms -

MitySOM-5CSX Altera Cyclone V

Recherche: MitySOM-5CSX Altera Cycl...

Aperçu Activité **Wiki** Forums

Wiki »

### Important Note about FPGA/HPS SDRAM Bridge (2013-12-13)

Altera has recently disclosed an implementation requirement related to the use of the FPGA to HPS SDRAM bridges. Until more formal technical notes are published at Altera, this page will be maintained to outline the understood issue for the benefits of MitySOM-5CSX customers.

Configuration of the FPGA to HPS SDRAM (fpga2sdram) AXI bridges involves the following major steps:

1. First, the FPGA ports on the fpga2sdram peripheral must be placed in reset. This is accomplished by writing a zero to the FPGAPORTRST register in the SDRAM Controller control group.
2. Second, the FPGA must be configured with an image that includes the configuration of the fpga2sdram ports. The FPGA fabric asserts configuration input ports at the input to the fpga2sdram bridges. The configuration ports affect such things as the width of the port as well as the direction, etc. When the FPGA is not configured, these configuration inputs are not defined.
3. Third, once the configuration inputs are set, the configuration must be then latched / applied to the fpga2sdram bridge peripheral. This is accomplished by writing a one to the APPLYCFG bit in the STATICCFG register in the SDRAM Controller control group. **This bit can only be written to while the SDRAM DDR Interface is guaranteed to be completely IDLE (including transfers from the ARM cores, DMAs, etc.).**
4. Finally, the FPGA ports on the fpga2sdram peripheral can be taken out of reset based on your configuration. This is accomplished by writing ones to the appropriate bits in the FPGAPORTRST register in the SDRAM Controller control group.

If these steps are not followed, attempting to use an FPGA port on the HPS SDRAM bridge controller may result in critical failure -- the HPS subsystem may freeze and effectively lock up the processor.

As a consequence to the note bolded in step 3, the fpga2sdram controller cannot be practically configured while most high level operating systems are running (linux, windows, android, etc.). Altera has provided the capability to set the configuration bit in step three with a macro command in their (and Critical Link's) u-Boot port. This is accomplished by copying a small routine to on-chip RAM that disables caches and asserts the APPLYCFG bit and then returns operation to the typical DDR space.

Therefore, if you have an FPGA design that utilizes the fpga2sdram controller, **you must program the FPGA in u-Boot following an FPGA / power on reset situation.**

Once the SDRAM controller is properly configured, the FPGA ports may be reset and enabled (steps 1 and 4) as often as necessary in order to facilitate reloading of the FPGA -- as long as the new fpga2sdram port configuration matches the original configuration. This will allow for reconfiguration of an FPGA while running linux, if necessary.

Wiki

- Page de démarrage
- Index par titre
- Index par date

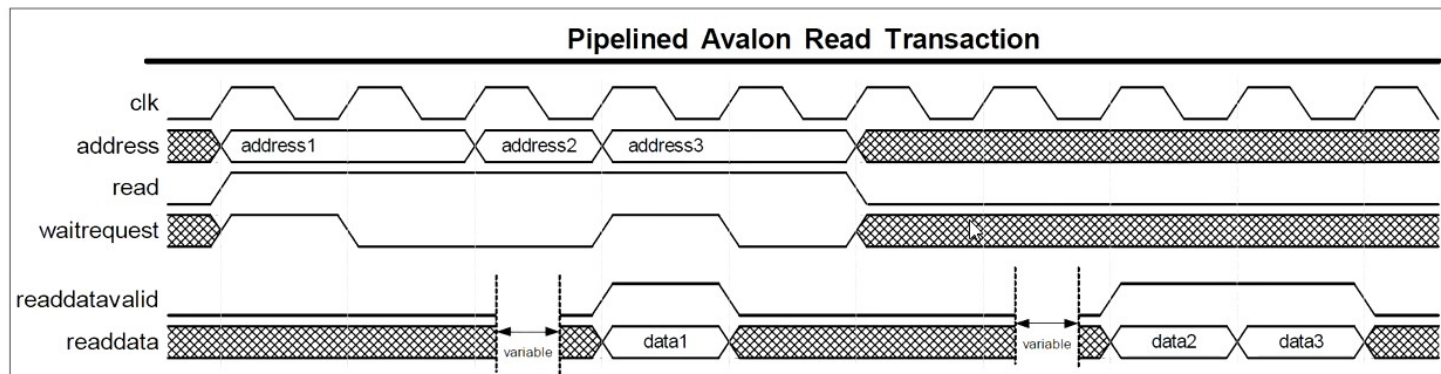


# Décharger le CPU en utilisant la RAM

## Utiliser avantageusement le contrôleur de RAM

Tirer parti de l'accès RAM

### The Three Types of Avalon MM Interfaces - Pipelined



# Décharger le CPU en utilisant la RAM

## Configurer le SoC dans Platform Designer

Exemple sur un projet d'envoi d'un burst de 10 valeurs sur 256 bits

The image shows a State Properties dialog box for a state machine configuration. The dialog has tabs for General, Incoming Transitions, Outgoing Transitions, Actions, and Format. The General tab is selected, showing a table of output ports and values. A red arrow points to the 'avm\_m0\_burstcount[10:0]' field, which is set to 10. The background shows a state machine diagram with states 'ask\_read' and 'wait\_read'.

Output Port	Output Value	Additional Conditions
avm_m0_address[31:0]	536870912	
avm_m0_read	1	
avm_m0_byteenable[31:0]	4294967295	
avm_m0_burstcount[10:0]	10	
out_data[255:0]	avm_m0_readdata[255:0]	
< New >		

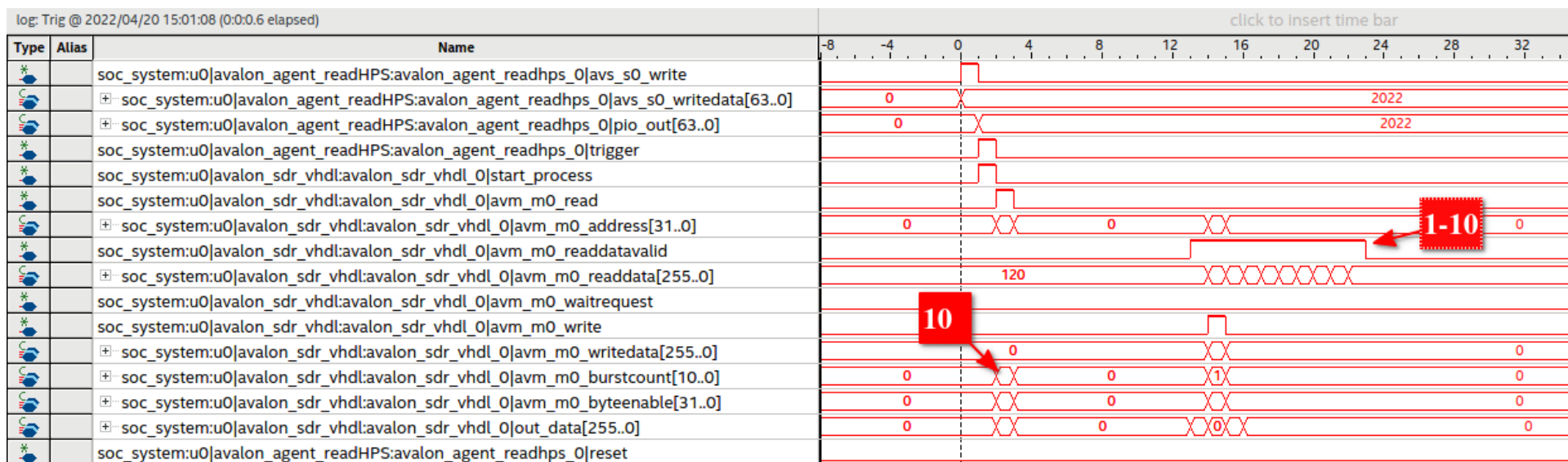
Buttons: OK, Close, Apply, Help

# Décharger le CPU en utilisant la RAM

## Configurer le SoC dans Platform Designer

Tirer parti de l'accès RAM

20 cycles @50MHz pour recevoir 10\*256 bit soit 6,4 Gbps



# Plan

## Plan

JOUR 1

§001 Faire communiquer Linux avec un FPGA sur le SoC intel DE10-nano

- distribution fournie par Altera
- créer une image SD (en partie sur un serveur distant)
- compilation croisée (eclipse CDT embedded)
- activer les bridges HPS <-> FPGA par un device tree
- utiliser le bridge HPS2FPGA

J 2

§002 Partager une zone de RAM entre Linux et un FPGA sur le SoC intel DE10-nano

- projet de lecture/écriture en RAM sur FPGA
- projet de lecture/écriture en RAM sur CPU

J 3

§003 Cas pratique avec seuillage d'image

- **projet openCV**
- échange sur mémoire RAM

# Décharger le CPU : seuillage d'image sur FPGA

## Préparer OpenCV

Installer opencv

Ou plus rapidement sur le SoC :

```
## Cross-compiler opencv
wget https://github.com/opencv/opencv/archive/4.x.zip
unzip 4.x.zip
cd opencv-4.x
mkdir build_hardfp
cd build_hardfp
~/intelFPGA/20.1/embedded/embedded_command_shell.sh
cmake -D OPENCV_GENERATE_PKGCONFIG=ON -DBUILD_SHARED_LIBS=OFF -DCMAKE_TOOLCHAIN_FILE=../platforms/linux/arm-
gnueabi.toolchain.cmake ../
make opencv_core opencv_imgcodecs opencv_imgproc gen-pkgconfig
## les librairies se trouvent dans ./lib et ./3rdparty/lib

## les headers sont repartis de partout ...
cd ..
find . -name "opencv2"
./include/opencv2
[...]
./modules/core/include/opencv2
./modules/imgproc/include/opencv2
./modules/imgcodecs/include/opencv2
./build_hardfp/opencv2
[...]
```



# Décharger le CPU : seuillage d'image sur FPGA

## Préparer OpenCV

Installer opencv

Ou plus rapidement sur le SoC :

```
## Il faut d'abord agrandir la partition de la distribution (avec GParted par exemple)
## Puis installer opencv (core + imgcodecs pour l'écriture dans un fichier .png)
apt install libopencv-core-dev
apt install libopencv-imgcodecs-dev

## modifier sur le SoC le fichier d'entete :
vi /usr/include/opencv2/opencv_modules.hpp
# commenter les lignes correspondants aux modules non-installes

## On pourra recuperer le fichier source dans $DEWD/captronic_formation_fpga_img_proc/patches/sdram/main.cpp
g++ main.cpp -lopencv_core -lopencv_imgcodecs
./a.out

## puis sur le PC de développement:
DE10NA_ADDR=192.168.1.16
export DEWD=$HOME/de10nano-wd
sshfs -o reconnect,ServerAliveInterval=15,ServerAliveCountMax=3 \
root@$DE10NA_ADDR:/ $DEWD/remote_de10_nano/
```

# Décharger le CPU : seuillage d'image sur FPGA

## Format d'image OpenCV

Manipuler une image sur OpenCV

```
9  #include <iostream>
10
11  #include "opencv2/opencv.hpp"
12  #include "opencv2/imgcodecs.hpp"
13
14
15  using namespace std;
16  using namespace cv;
17
18
19  #define BRIDGE_TRIGGER      (0xC0000000) // HPS_2_FPGA
20  #define BRIDGE_TRIGGER_SPAN (0x08) // 64 bits => 64/8 = 8 octets
21
22  #define BRIDGE_SDRAM        (0x20000000) // de debut de la ram reservee (@512Mo)
23  #define BRIDGE_SDRAM_SPAN  (0x20) // 256 bits => 256/8 = 32 octets => 0x20
24
25  #define HPS_2_FPGA_64bits (0x00)
```

trigger

sdram

# Décharger le CPU : seuillage d'image sur FPGA

## Format d'image OpenCV

Manipuler une image sur OpenCV

```
47  Mat img(50, 50, CV_8UC1, Scalar(100));
48  if (img.empty())
49  {
50      cout << "\n Image not created. You"
51              " have done something wrong. \n";
52      return -1;    // Unsuccessful.
53  }
54
67  fd = open("/dev/mem", O_RDWR | O_SYNC);
68
69  if (fd < 0) {
70      perror("Couldn't open /dev/mem\n");
71      return -2;
72  }
73
74  bridge_trigger_map = (uint8_t *)mmap(NULL, BRIDGE_TRIGGER_SPAN, PROT_READ | PROT_WRITE,
75      MAP_SHARED, fd, BRIDGE_TRIGGER);
76  bridge_ram_map = (uint8_t *)mmap(NULL, BRIDGE_SDRAM_SPAN, PROT_READ | PROT_WRITE,
77      MAP_SHARED, fd, BRIDGE_SDRAM);
78  if (bridge_trigger_map == MAP_FAILED) {
79      perror("mmap trigger failed.");
80      close(fd);
81      return -3;
82  }
83  if (bridge_ram_map == MAP_FAILED) {
84      perror("mmap ram failed.");
85      close(fd);
86      return -3;
87  }
88
89
90
91  trigger_map = bridge_trigger_map + HPS_2_FPGA_64bits ;
92  sdram_map   = bridge_ram_map   + 0 ;
93
94  img.data = sdram_map ;
```

# Décharger le CPU : seuillage d'image sur FPGA

## Format d'image OpenCV

Manipuler une image sur OpenCV

```
101 // trigger avec la config
102 printf("write config   = %" PRIu64 "\tat address %x\n", config, trigger_map);
103 *((uint64_t *)trigger_map) = config ;
104
121 // PNG
122 vector<int> compression_params;
123 compression_params.push_back(IMWRITE_PNG_COMPRESSION);
124 compression_params.push_back(9);
125
126 printf("write alpha.png") ;
127 bool rr = false ;
128 try
129 {
130     rr = imwrite("alpha.png", img, compression_params);
131 }
```

# Plan

## Plan

JOUR 1

§001 Faire communiquer Linux avec un FPGA sur le SoC intel DE10-nano

- distribution fournie par Altera
- créer une image SD (en partie sur un serveur distant)
- compilation croisée (eclipse CDT embedded)
- activer les bridges HPS <-> FPGA par un device tree
- utiliser le bridge HPS2FPGA

J 2

§002 Partager une zone de RAM entre Linux et un FPGA sur le SoC intel DE10-nano

- projet de lecture/écriture en RAM sur FPGA
- projet de lecture/écriture en RAM sur CPU

J 3

§003 Cas pratique avec seuillage d'image

- projet openCV
- **échange sur mémoire RAM**

## Décharger le CPU : seuillage d'image sur FPGA

# Modification de l'image OpenCV depuis le FPGA

## Changer le fichier VHDL de la machine d'état

```

40 ARCHITECTURE BEHAVIOR OF avalon_sdr_vhdl IS
41     TYPE type_fstate IS (init,wait_read,write_start,ask_read,write_end);
42     SIGNAL fstate : type_fstate;
43     SIGNAL reg_fstate : type_fstate;
44     constant ramp_v : std_logic_vector (255 downto 0) := X"00102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F0";
45
108         avm_m0_writedata <= ramp_v(255 downto 0);
109         --avm_m0_writedata <= "00000000000000000000000000000000";
110

```

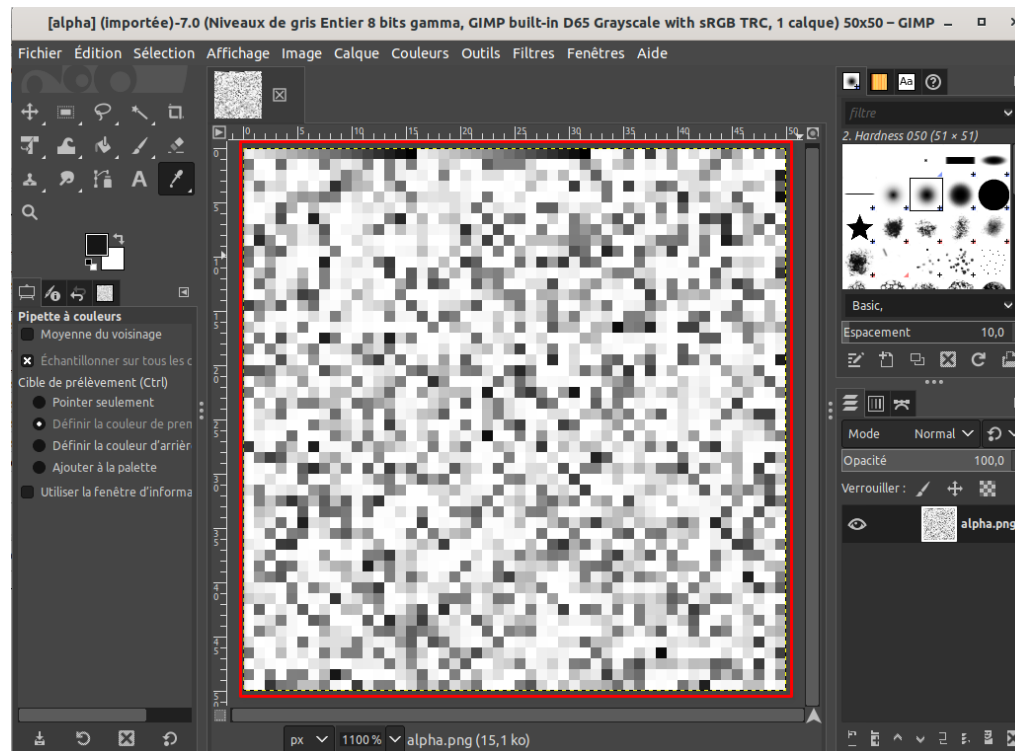
```
## Sur le SoC
g++ main.cpp -lopencv_core -lopencv_imgcodecs
./a.out
```

# Décharger le CPU : seuillage d'image sur FPGA

## L'image modifiée

Après récupération de l'image sur le SoC

Visualiser l'image générée



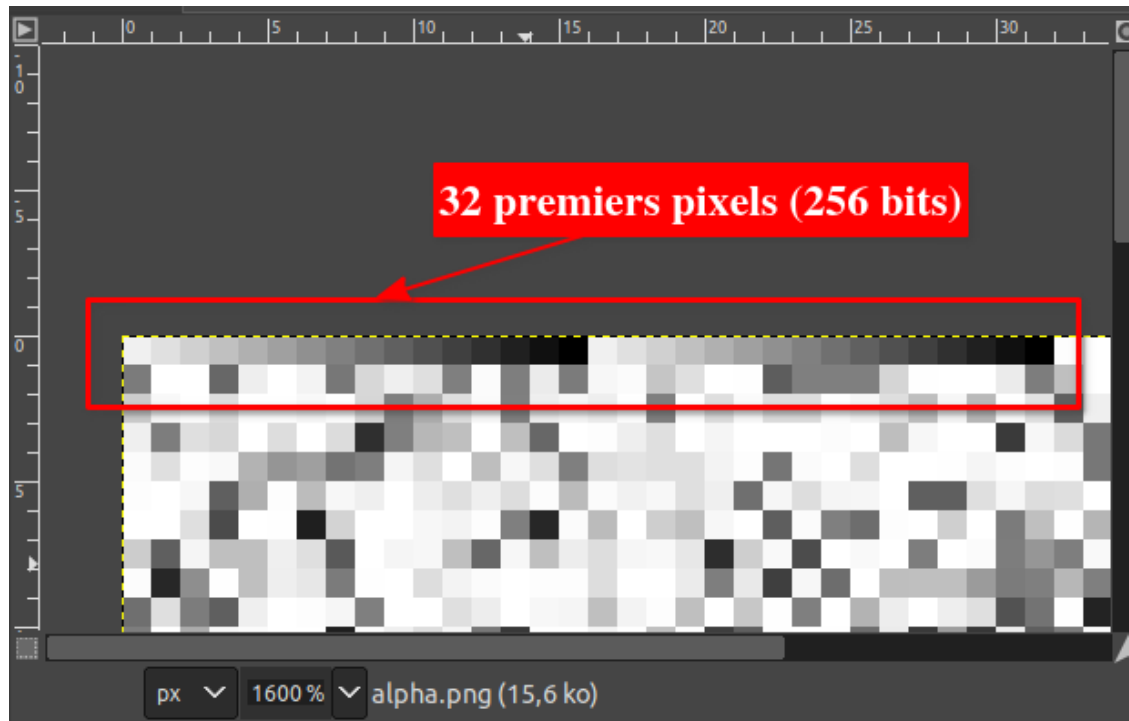


# Décharger le CPU : seuillage d'image sur FPGA

## L'image modifiée

Après récupération de l'image sur le SoC

Visualiser l'image générée



# Conclusion

## Avantages/difficultés du déchargement du CPU en utilisant un SoC FPGA

### Avantages

Puissance de calcul disponible

### Difficultés

Prise en main de l'outil de développement Quartus

Langages de programmation HDL

# Conclusion

A venir dans une formation complémentaire :

- Communication entre le HPS et le CPU via un module noyau (fichier caractère)
- Utiliser un écran tactile pour obtenir une tablette dédiée au traitement du signal embarqué.