

Atelier 4

Picoblaze & Assembleur

Projet S4 info – H19
GIF402 – Conception d'un système ordonné

Larissa Njeimana

14 – Mars – 2019

Travail proposé

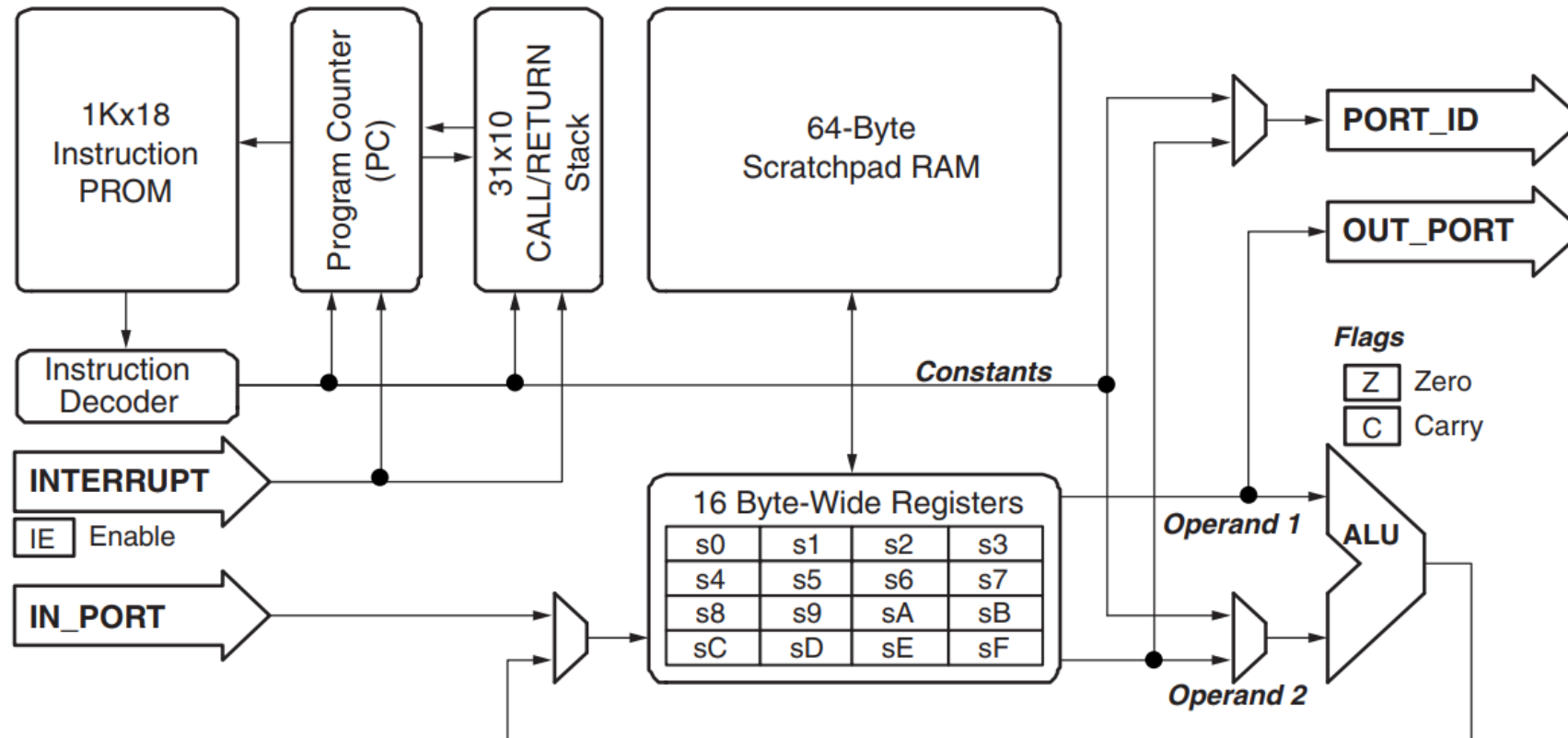
- Introduction à l'utilisation du langage **assembleur** dans l'environnement Vivado
 - Microcontrôleur embarqué **Picoblaze**
 - Jeu d'instructions **KCPSM6**

Langage assembleur

- Pourquoi utiliser le langage assembleur dans un FPGA
 - **Avantages :**
 - Simplicité des instructions
 - Implémentation facile de la logique séquentielle :
 - ex: Machine à états finis (MEF)
 - **Inconvénients :**
 - Manque de rapidité : une instruction prend plus d'un cycle d'horloge

Picoblaze

- Le Picoblaze est un microcontrôleur très simple à 8-bit :
 - Architecture RISC
 - Adressage sur 12-bit
 - Instructions sur 18-bit
 - **L'exécution d'une instruction prend 2 cycles d'horloge**

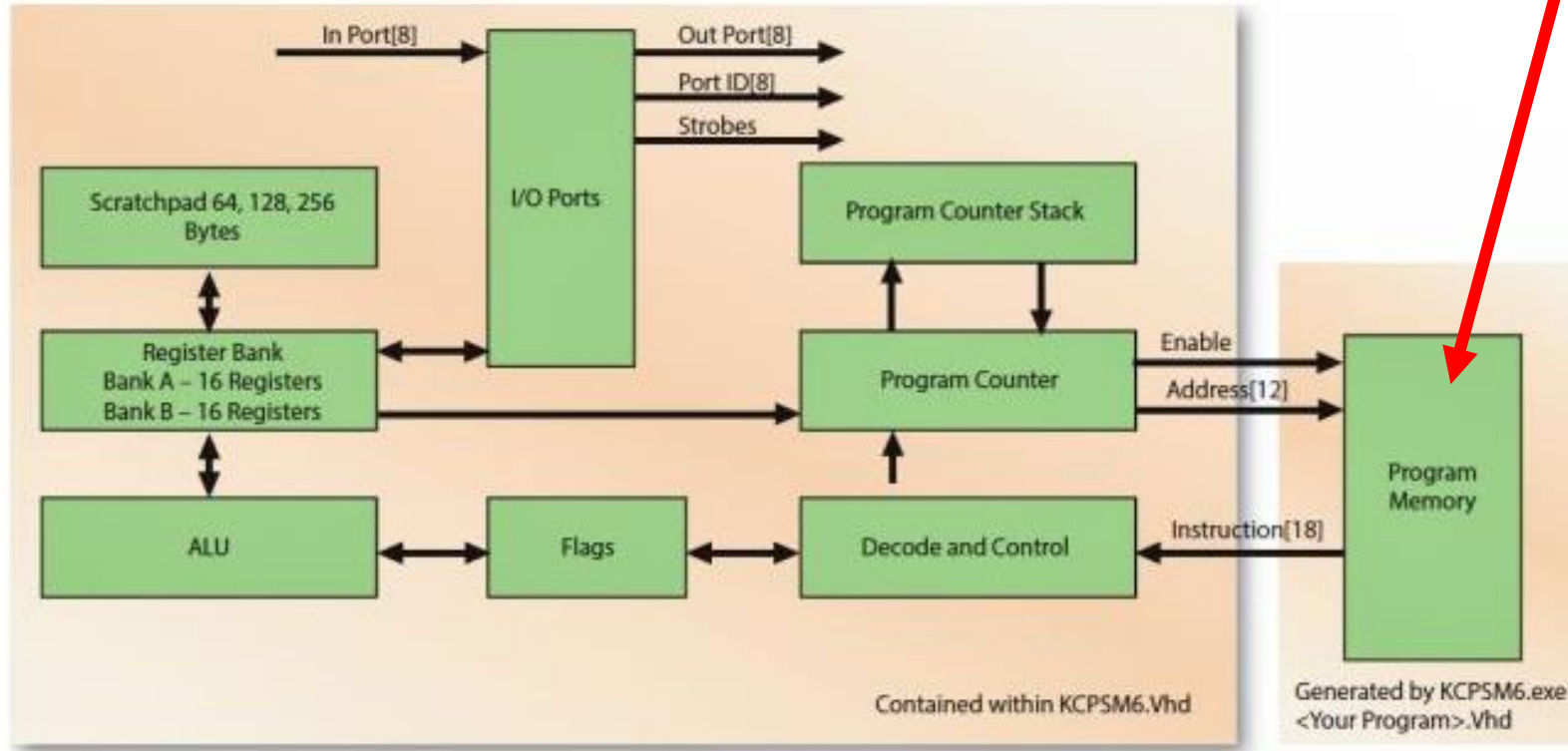


Picoblaze

- Le Picoblaze occupe au plus 2 Block RAM dans un FPGA
 - Possibilité de mettre plusieurs Picoblaze au sein d'un même design
- Ses performances dépassent beaucoup de microcontrôleurs 8-bit sur le marché
- Exemples d'application :
 - Contrôle de moteur
 - Calculatrice
 - Interface UART
 - Master SPI
 - Master I2C
 - Contrôle PID
 - etc

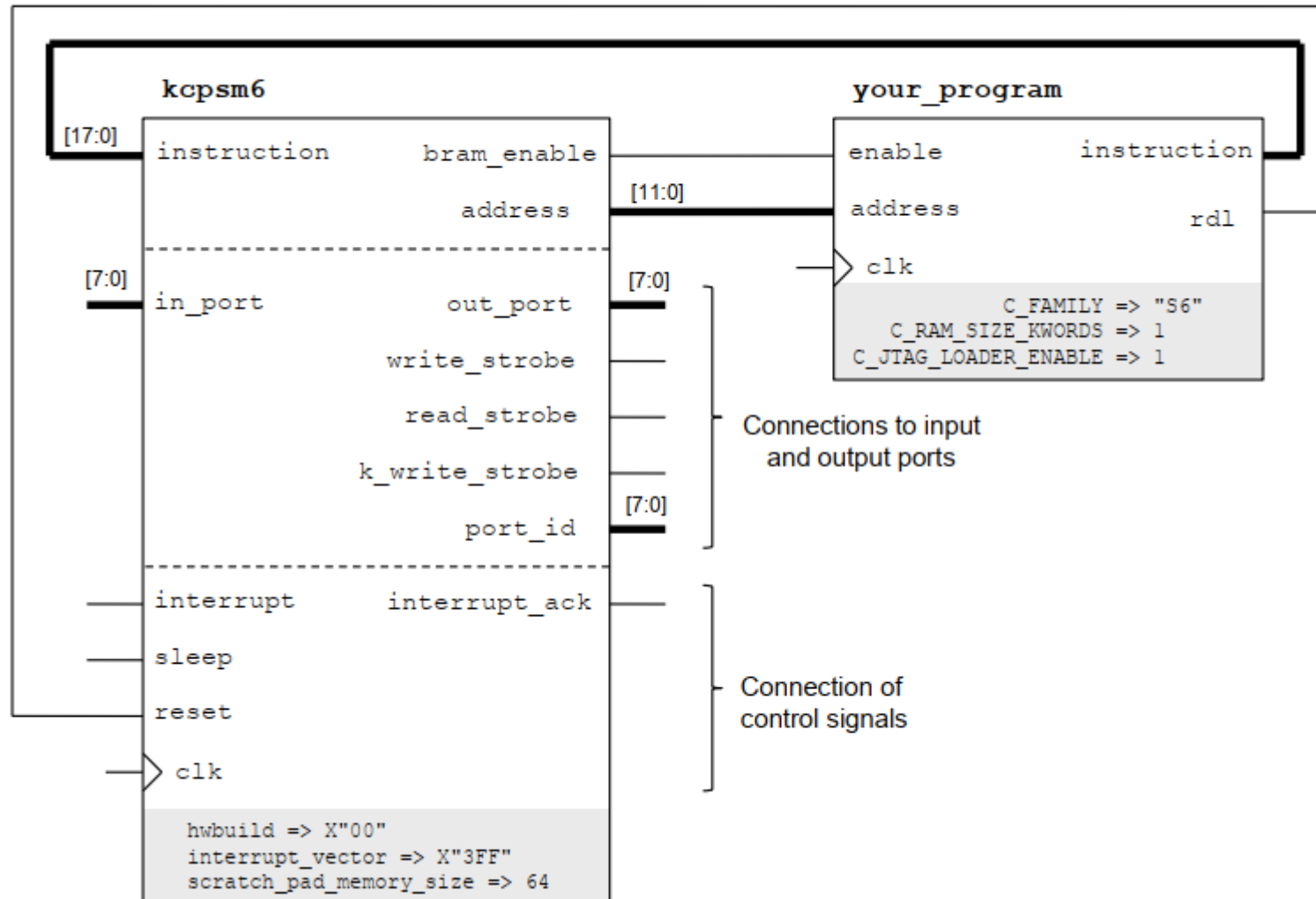
Picoblaze

- Le code du Picoblaze est offert par Xilinx.
- Il suffit d'écrire le programme qui sera placé dans la mémoire du Picoblaze.



Picoblaze

- Du côté **VHDL**, il s'agit d'un **component** comme un autre qui est rajouté.



Picoblaze

- Le Picoblaze utilise le jeu d'instructions **KCPSM6**.
- On écrit notre programme en assembleur et un compilateur va générer le fichier **VHD** correspondant

KCPSM6 Instruction Set

aaa : 12-bit address 000 to FFF
 kk : 8-bit constant 00 to FF
 pp : 8-bit port ID 00 to FF
 p : 4-bit port ID 0 to F
 ss : 8-bit scratch pad location 00 to FF
 x : Register within bank s0 to sF
 y : Register within bank s0 to sF

Page	Opcode	Instruction
------	--------	-------------

Register loading

55	00xy0	LOAD sX, sY
55	01xkk	LOAD sX, kk
71	16xy0	STAR sX, sY
71	17xkk	STAR sX, kk

Logical

56	02xy0	AND sX, sY
56	03xkk	AND sX, kk
57	04xy0	OR sX, sY
57	05xkk	OR sX, kk
58	06xy0	XOR sX, sY
58	07xkk	XOR sX, kk

Arithmetic

59	10xy0	ADD sX, sY
59	11xkk	ADD sX, kk
60	12xy0	ADDCY sX, sY
60	13xkk	ADDCY sX, kk
61	18xy0	SUB sX, sY
61	19xkk	SUB sX, kk
62	1Axy0	SUBCY sX, sY
62	1Bxkk	SUBCY sX, kk

Test and Compare

63	0Cxy0	TEST sX, sY
63	0Dxkk	TEST sX, kk
64	0Exy0	TESTCY sX, sY
64	0Fxkk	TESTCY sX, kk
65	1Cxy0	COMPARE sX, sY
65	1Dxkk	COMPARE sX, kk
66	1Exy0	COMPARECY sX, sY
66	1Fxkk	COMPARECY sX, kk

Page 54

Page	Opcode	Instruction
------	--------	-------------

Shift and Rotate

67	14x06	SL0 sX
67	14x07	SL1 sX
67	14x04	SLX sX
67	14x00	SLA sX
67	14x02	RL sX
68	14x0E	SR0 sX
68	14x0F	SR1 sX
68	14x0A	SRX sX
68	14x08	SRA sX
68	14x0C	RR sX

Register Bank Selection

70	37000	REGBANK A
70	37001	REGBANK B

Input and Output

73	08xy0	INPUT sX, (sY)
73	09xpp	INPUT sX, pp
74	2Cxy0	OUTPUT sX, (sY)
74	2Dxpp	OUTPUT sX, pp
78	2Bkcp	OUTPUTK kk, p

Scratch Pad Memory

(64, 128 or 256 bytes)

81	2Exy0	STORE sX, (sY)
81	2Fsss	STORE sX, ss
82	0Axy0	FETCH sX, (sY)
82	0Bsss	FETCH sX, ss

Page	Opcode	Instruction
------	--------	-------------

Interrupt Handling

83	28000	DISABLE INTERRUPT
83	28001	ENABLE INTERRUPT
84	29000	RETURNI DISABLE
84	29001	RETURNI ENABLE

Jump

87	22aaa	JUMP aaa
88	32aaa	JUMP Z, aaa
88	36aaa	JUMP NZ, aaa
88	3Aaaa	JUMP C, aaa
88	3Eaaa	JUMP NC, aaa
89	26xy0	JUMP@ (sX, sY)

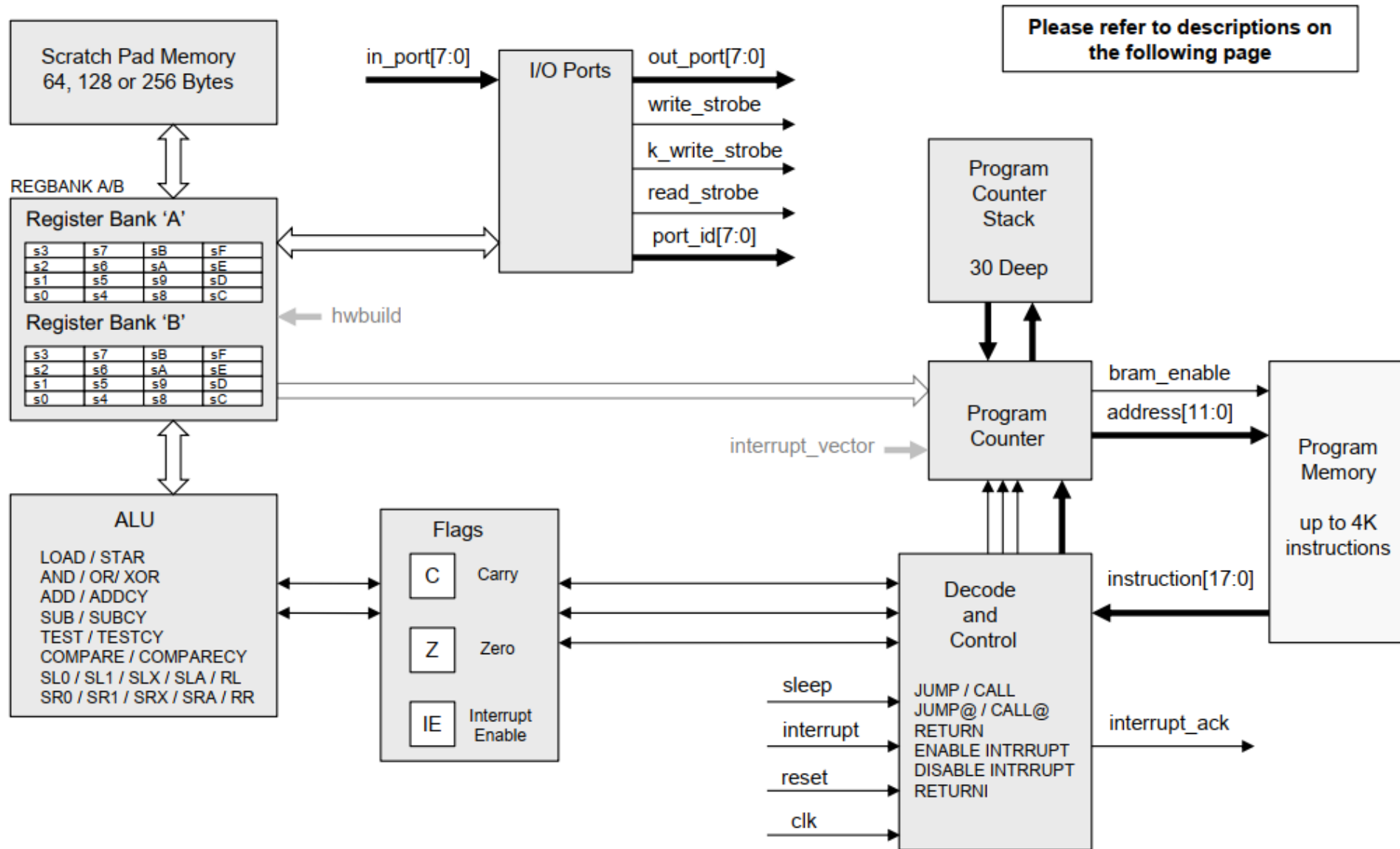
Subroutines

92	20aaa	CALL aaa
93	30aaa	CALL Z, aaa
93	34aaa	CALL NZ, aaa
93	38aaa	CALL C, aaa
93	3Caaa	CALL NC, aaa
94	24xy0	CALL@ (sX, sY)
96	25000	RETURN
97	31000	RETURN Z
97	35000	RETURN NZ
97	39000	RETURN C
97	3D000	RETURN NC
98	21xkk	LOAD&RETURN sX, kk

Version Control

101	14x80	HWBUILD sX
-----	-------	------------

KCPSM6 Architecture and Features



Picoblaze

- Plus de documentations se trouvent dans les fichiers se trouvant dans le répertoire de cet atelier, dans le sous-dossier \PicoblazeSources

 KCPSM6_Release9_30Sept14.zip

 KCPSM6_User_Guide_30Sept14.pdf

 PicoBlaze_Design_in_Vivado.pdf

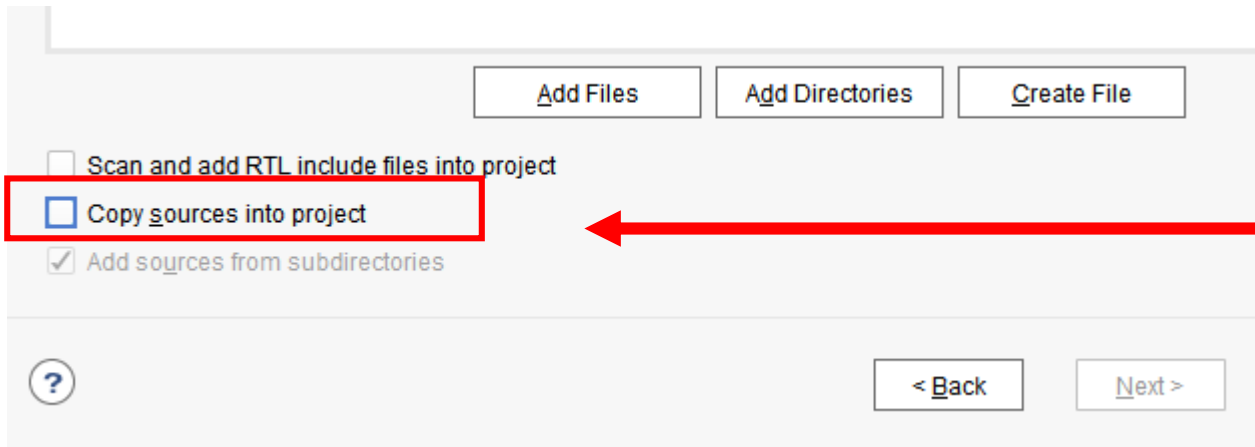
- Pour les détails sur les instructions assembleurs, le fichier **KCPSM6_User_Guide_30Sept14.pdf** définit en détails chaque instruction.

Exercice 1

- Pour mieux comprendre, faisons un premier exemple avec le Picoblaze.
- Créer un nouveau projet Vivado ciblant la carte ZYBO Z7-10 que vous appellerez **Exercice1**.
- Copiez dans le dossier du nouveau projet Vivado crée **Exercice1**, les fichiers suivants se trouvant dans le dossier de l'atelier sur le site web:
 - Demo1/Demo1_Top.vhd
 - Demo1/Demo1_constraints.xdc
 - PicoblazeSources/kcpsm6.exe
 - PicoblazeSources/kcpsm6.vhd
 - PicoblazeSources/ROM_form.vhd

Exercice 1

- En utilisant l'option **Add Sources** dans Vivado, rajouter les fichiers suivants qui viennent d'être copiés dans le répertoire du projet Vivado :
 - Demo1_Top.vhd
 - kcpsm6.vhd
 - Demo1_constraints.xdc



Avant de cliquer sur **Add Files**, décocher l'option « Copy sources into project »!!

Exercice 1

- Ouvrez le fichier **Demo1_Top.vhd**, il contient les composants :
 - **kcpsm6** pour le picoblaze
 - **myProgram** qui contiendra notre programme

```
--  
-- Declaration of the KCPSM6 component including default values for generics.  
--  
component kcpsm6  
generic(  
    hwbuild           : std_logic_vector(7 downto 0) := X"00";  
    interrupt_vector  : std_logic_vector(11 downto 0) := X"3FF";  
    scratch_pad_memory_size : integer := 64 -- other options are 128, 256  
);  
port (  
    address      : out std_logic_vector(11 downto 0);  
    instruction   : in std_logic_vector(17 downto 0);  
    bram_enable   : out std_logic;  
    in_port       : in std_logic_vector(7 downto 0);  
    out_port      : out std_logic_vector(7 downto 0);  
    port_id       : out std_logic_vector(7 downto 0);  
    write_strobe  : out std_logic;  
    k_write_strobe : out std_logic;  
    read_strobe   : out std_logic;  
    interrupt     : in std_logic;  
    interrupt_ack  : out std_logic;  
    sleep         : in std_logic;  
    reset         : in std_logic;  
    clk           : in std_logic  
);  
end component;  
  
--  
-- Declaration of the default Program Memory recommended for development.  
-- The name of this component should match the name of your PSM file.  
--  
component myProgram  
generic(  
    C_FAMILY : string := "S6";  
    C_RAM_SIZE_KWORDS : integer := 1;  
    C_JTAG_LOADER_ENABLE : integer := 0);  
Port (  
    address : in std_logic_vector(11 downto 0);  
    instruction : out std_logic_vector(17 downto 0);  
    enable : in std_logic;  
    rdl : out std_logic;  
    clk : in std_logic);  
end component;
```

Exercice 1

- Toujours dans **Demo1_Top.vhd**, on définit 2 ports d'**entrées (input)** vers le Picoblaze :
 - Le port d'adresse 0 est relié aux **boutons** de la carte ZYBO
 - Le port d'adresse 1 est relié aux **switches** de la carte ZYBO

```
input_ports: process(sys_clock)
begin
    if sys_clock'event and sys_clock = '1' then

        case port_id(0) is -- we have to inputs so 1 bit in port id is enough

            -- Read input_port_a at port address 00 hex
            when '0' => in_port(3 downto 0) <= i_btn;

            -- Read input_port_b at port address 01 hex
            when '1' => in_port(3 downto 0) <= i_sw;

            -- To ensure minimum logic implementation when defining a multiplexer always
            -- use don't care for any of the unused cases (although there are none in this
            -- example).
            when others => in_port(3 downto 0) <= "XXXX";

        end case;

        in_port(7 downto 4) <= "0000";

    end if;
end process input_ports;
```

Exercice 1

- Toujours dans **Demo1_Top.vhd**, on définit 2 ports de **sorties (output)** vers le Picoblaze :
 - Le **port d'adresse 2** est relié aux **LEDs** de la carte ZYBO
 - Le **port d'adresse 4** est relié au **Pmod 8LD**
- Quand le nombre d'entrées/sorties le permet, l'encodage d'adresse sous forme **one-hot** est privilégié pour minimiser le nombre ressources logiques requises pour synthétiser le picoblaze (Nombre de Look-Up Table ou LUT)

```
output_ports: process(sys_clock)
begin
    if sys_clock'event and sys_clock = '1' then

        -- 'write_strobe' is used to qualify all writes to general output ports.
        if write_strobe = '1' then

            -- Write to output port w at port address 01 hex
            if port_id(1) = '1' then -- port 02
                q_leds <= out_port(3 downto 0);
            end if;

            -- Write to output port x at port address 02 hex
            if port_id(2) = '1' then -- port 04
                q_Pmod_8LD <= out_port;
            end if;

        end if;

    end if;
end process output_ports;

Pmod_8LD <= q_Pmod_8LD;
o_leds <= q_leds;
```

Exercice 1

- Nous venons de définir les entrées et les sorties vers le Picoblaze.
- Pour terminer notre design, il reste à écrire notre programme assembleur pour déterminer le comportement des sorties et le traitement fait avec les valeurs en entrée .
- Il faut que notre programme assembleur **ait le même nom** que le component dans **Demo1_Top.vhd**. Dans notre cas, c'est « myProgram ».
- Créer dans un fichier myProgram.psm avec un éditeur de texte. Ce fichier doit se trouver dans le même répertoire que **kcpsm6.exe**.

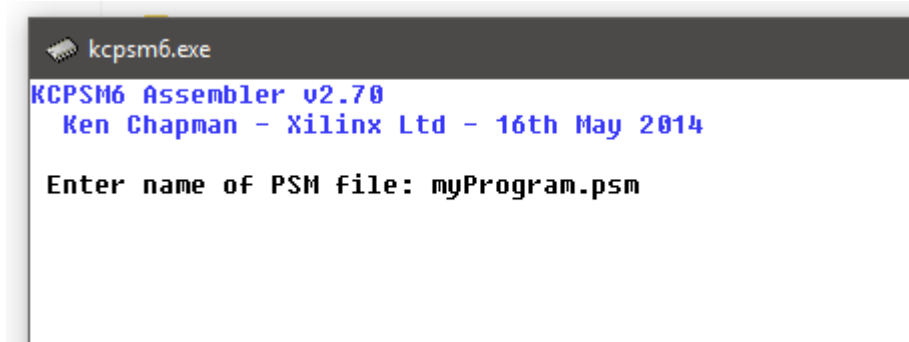
Exercice 1

- Dans myProgram.psm : copier le code se trouvant dans le dossier de l'atelier :
 - Demo1/**Exemple1.psm**
- On affiche la valeur des switch sur les LEDs de la carte ZYBO.
- Les valeurs des boutons sont affichées sur le Pmod 8LD.

```
; myProgram.psm
;
; This is the start of 'myProgram' for KCPSM6
;
; 4 Switches
CONSTANT Switches_port, 01
;
; 4 Buttons
CONSTANT Buttons_port, 00
;
; 4 LEDs
CONSTANT LED_port, 02
;
; 8 Pmod 8LD LEDs
CONSTANT Pmod_8LD_port, 04
;
; Bit assignments for each LED/button
CONSTANT bit0, 00000001'b ;
CONSTANT bit1, 00000010'b ;
CONSTANT bit2, 00000100'b ;
CONSTANT bit3, 00001000'b ;
CONSTANT bit4, 00010000'b ;
CONSTANT bit5, 00100000'b ;
CONSTANT bit6, 01000000'b ;
CONSTANT bit7, 10000000'b ;
;
;
start: INPUT s0, Switches_port
        OUTPUT s0, LED_port ; dis|
        ;
        INPUT s1, Buttons_port
        OUTPUT s1, Pmod_8LD_port ; dis|
        ;
        OR s1, bit4 ; set
        OUTPUT s1, Pmod_8LD_port ; turn
        JUMP start
```

Exercice 1

- Maintenant que notre programme assembleur est écrit, on peut le compiler.
- Cliquer sur **kcpsm6.exe** et entrez le nom du fichier que vous voulez compiler.

A screenshot of a Windows command prompt window titled "kcpsm6.exe". The window has a dark gray title bar with a small icon on the left. The text inside the window is as follows:

```
KCPSM6 Assembler v2.70  
Ken Chapman - Xilinx Ltd - 16th May 2014  
  
Enter name of PSM file: myProgram.psm
```

- Enter.

Exercice 1

- Le résultat de la compilation va s'afficher.
- S'il n'y a pas d'erreurs « **Assembly completed successfully** » apparaîtra. Sinon un message d'erreur montrera les lignes de code qui sont problématiques.
- Ne pas fermer la fenêtre ci-contre : avec la commande **R**, on peut recompiler sans avoir à rentrer le nom du programme à chaque fois.

```
kcpsm6.exe
KCPSM6 Assembler v2.70
Ken Chapman - Xilinx Ltd - 16th May 2014

Enter name of PSM file: myProgram.psm

Reading top level PSM file...
C:\ZYB0\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.psm

A total of 70 lines of PSM code have been read

Checking line labels
Checking CONSTANT directives
Checking STRING directives
Checking TABLE directives
Checking instructions

Writing formatted PSM file...
C:\ZYB0\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.fmt

Expanding text strings
Expanding tables
Resolving addresses and Assembling Instructions
Last occupied address: 015 hex
Nominal program memory size: 1K (1024)    address(9:0)
Occupied memory locations: 22
Assembly completed successfully

Writing LOG file...
C:\ZYB0\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.log
Writing HEX file...
C:\ZYB0\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.hex
Writing VHDL file...
C:\ZYB0\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.vhd

KCPSM6 Options.....
R - Repeat assembly with 'myProgram.psm'
N - Assemble new file.
Q - Quit
```

Exercice 1

- Un fichier **myProgram.vhd** est généré et c'est lui qu'il faut rajouter à notre projet vivado.
- Le fichier **myProgram.fmt** est une version de notre fichier .psm mais formater de façon plus élégante. On peut copier son contenu dans **myProgram.psm** pour avoir un code le plus lisible possible.

```
kcpsm6.exe
KCPSM6 Assembler v2.70
Ken Chapman - Xilinx Ltd - 16th May 2014

Enter name of PSM file: myProgram.psm

Reading top level PSM file...
C:\ZYBO\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.psm

A total of 70 lines of PSM code have been read

Checking line labels
Checking CONSTANT directives
Checking STRING directives
Checking TABLE directives
Checking instructions

Writing formatted PSM file...
C:\ZYBO\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.fmt

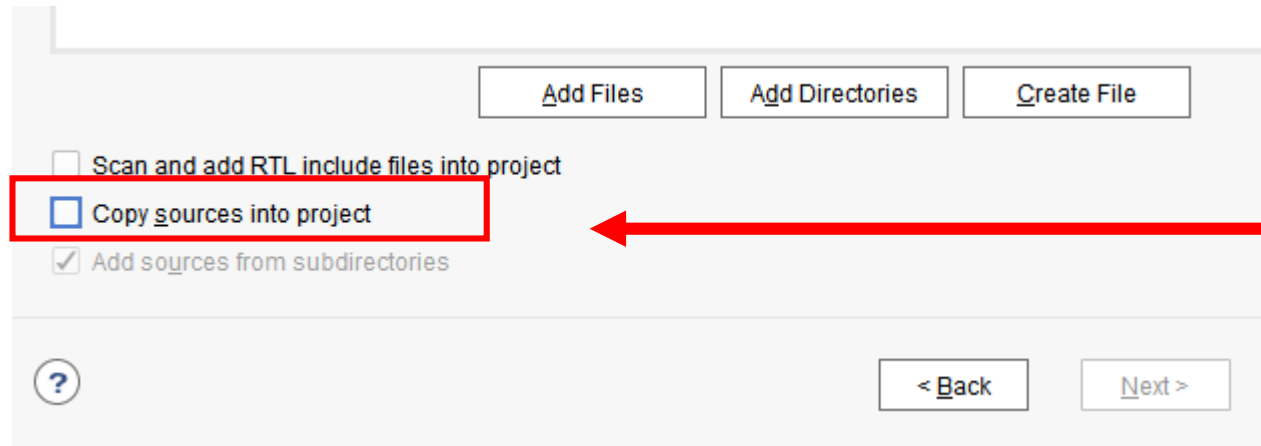
Expanding text strings
Expanding tables
Resolving addresses and Assembling Instructions
Last occupied address: 015 hex
Nominal program memory size: 1K (1024)    address(9:0)
Occupied memory locations: 22
Assembly completed successfully

Writing LOG file...
C:\ZYBO\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.log
Writing HEX file...
C:\ZYBO\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.hex
Writing VHDL file...
C:\ZYBO\Work-2018.2\Ateliers\Atelier4_Picoblaze\Demo1\myProgram.vhd

KCPSM6 Options.....
R - Repeat assembly with 'myProgram.psm'
N - Assemble new file.
Q - Quit
```

Exercice 1

- En utilisant l'option Add Sources de Vivado, ajouter le fichier **myProgram.vhd** à notre projet.



Avant de cliquer sur **Add Files**, décocher l'option « Copy sources into project »!!

Exercice 1

- Maintenant que tous les fichiers dans l'hierarchie du projet Vivado sont présents, on peut générer le bitstream.
- Corriger s'il y a des erreurs.
- Une fois, le bitstream généré, ouvrez **Hardware Manager**. Assurez-vous de brancher le Pmod 8LD sur le **port JD**.
- Programmer la carte Zybo Z710 et observer le comportement. Il devrait refléter ce qui a été défini dans le code assembleur dans **myProgram.psm**.

Exercice 2

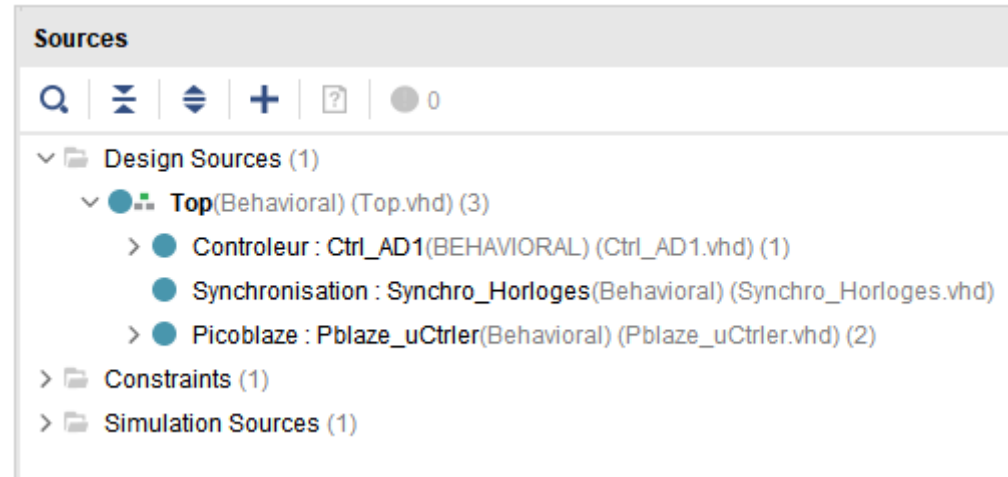
- Si on veut modifier le comportement du programme, il faut :
 - Modifier **myProgram.psm**
 - Compiler avec **kcpsm6.exe (si sa fenêtre est restée ouverte, faire R pour recompiler)**
 - Le fichier VHD **myProgram.vhd** va être régénéré automatiquement.
 - Relancer la génération du bitstream
 - Programmer le nouveau bitstream avec **Hardware Manager**
 - Observer les modifications

Exercice 2

- Remplaçons le code dans **myProgram.psm** avec ce qui se trouve dans le dossier de l'atelier :
 - Demo1/**Exemple2.psm**
- Suivre la démarche décrite dans l'acétate précédente.
- Quel est le comportement de ce nouveau code?
- Faites le lien avec le code assembleur dans **myProgram.psm**

Exercice 3

- Dans le dossier de l'atelier, le sous-dossier Demo2 comprend un projet Vivado avec un picoblaze intégré.
- Il reprend des morceaux du projet de l'atelier du 21 février avec l'ADC (Pmod AD1) mais cette fois-ci, sans processeur ZYNQ.



Exercice 3

- Un programme assembleur est déjà écrit dans un fichier qui s'appelle aussi **myProgram.psm**.
- Le fichier **Pblaze_uCtrler.vhd** prend les échantillons (12-bit) provenant de la MEF et les envoient vers le Picoblaze pour traitement.
- Il faut deux registres dans le Picoblaze 8-bit pour recueillir les échantillons.
- Le programme assembleur compte le nombre de fois qu'un échantillon est égal à 0x0FFF et affiche ce compteur sur les 4 LEDs de la carte ZYBO.
- On veut aussi afficher la valeur de l'échantillon sur le Pmod 8LD. Comme il n'y a que 8 Leds sur ce Pmod, les 4-LSB de l'échantillon sont tronqués.

Exercice 3

- Générer le bitstream.
- Brancher le Pmod AD1 sur le port JC et le Pmod 8LD sur le port JD.
- Faites le montage avec le potentiomètre (ou tout autre capteur) pour générer le signal d'entrée de l'ADC.
- Programmer le FPGA avec Hardware Manager pour observer le comportement.
- Faites le lien entre le code assembleur.

Exercice 3

- Maintenant à votre tour de modifier, le code assembleur dans [myProgram.psm](#) pour définir vos propres traitements du signal provenant de l'ADC.

Questions?