AGH

Mechatroniczne systemy wykonawcze,
sensoryczne i sterujące

Architektury hybrydowe SoC

dr inż. Grzegorz Góra

D1-Lab 20

ggora@agh.edu.pl

http://home.agh.edu.pl/~ggora/

Katedra Robotyki i Mechatroniki

Wydział Inżynierii Mechanicznej i Robotyki

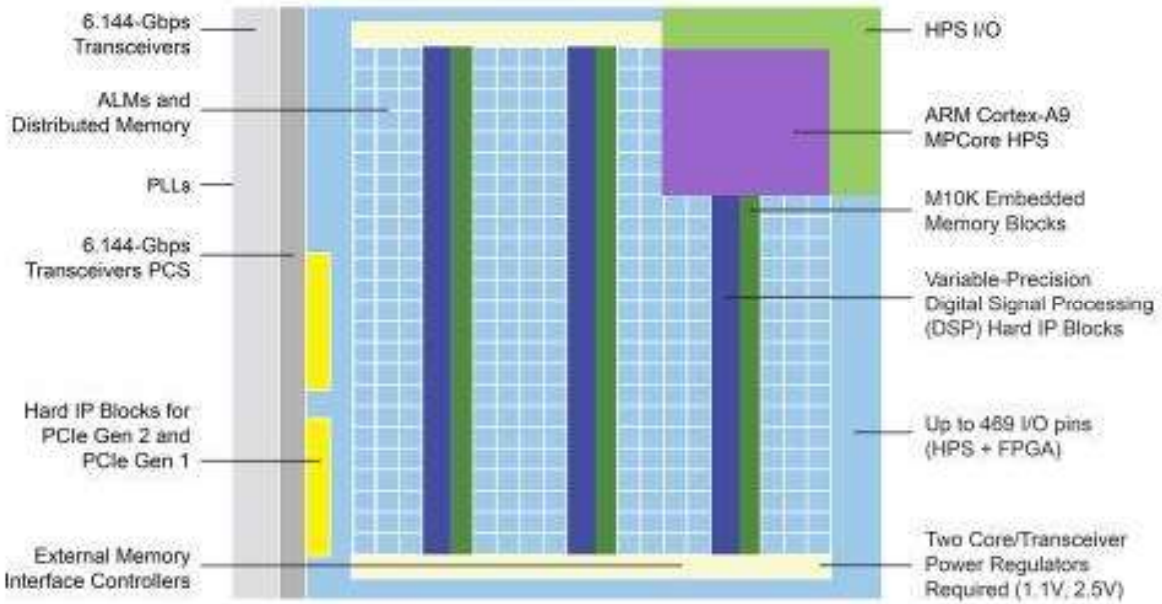
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

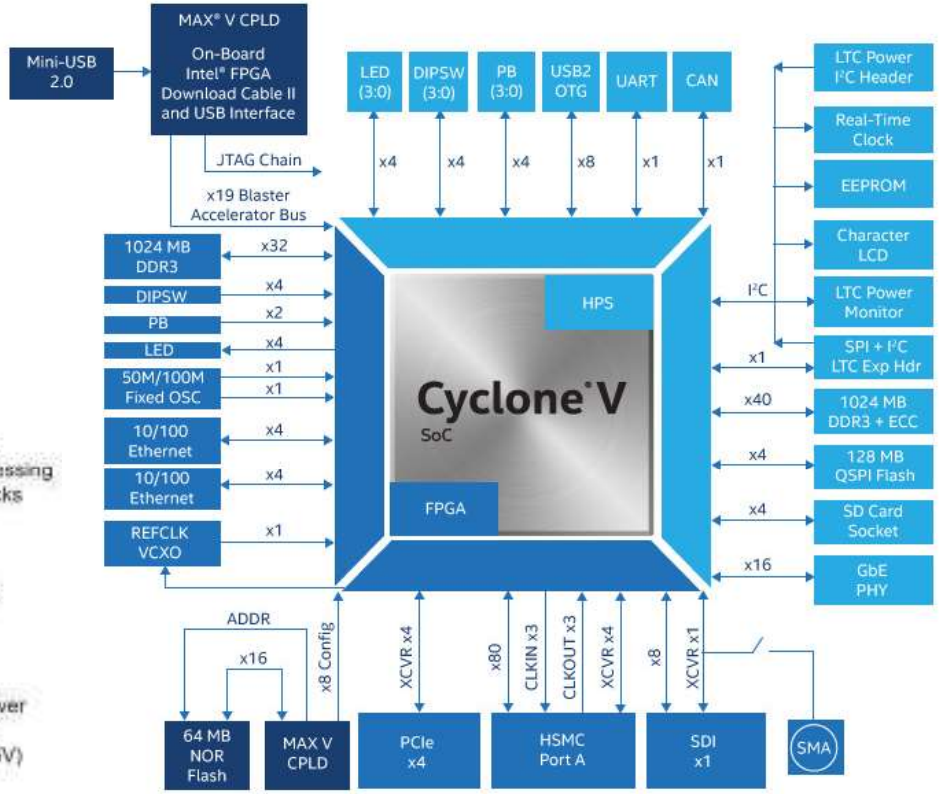
PLAN WYKŁADU

1. Układy hybrydowe SoC = FPGA + HPS (MCU)
2. Jakie możliwości daje hybrydowa platforma sprzętowa?
3. Możliwe konfiguracje zasilania i wykorzystania części FPGA i HPS
4. Wymiana danych pomiędzy częściami FPGA i HPS
5. Mapowanie pamięci
6. Obraz kardy SD
7. Układy peryferyjne HSP
8. Proces uruchamiana HPS
9. DE10-Nano
10. Cyclone V SoC – Jak zacząć?
11. HPS – Program w języku C/C++
12. FPGA – Modulator PWM
13. FPGA – Moduł obsługi Enkodera inkrementalnego
14. FPGA + HPS – Wymiana danych

Cyclone V SoC
FPGA + HPS

Układy Cyclone V SoC to dwurdzeniowy procesor ARM Cortex-A9 z wbudowanym zestawem układów peryferyjnych oraz kontrolerem pamięci. Część FPGA jest połączona z systemem procesorowym (HPS) za pomocą szybkiej magistrali o przepustowości ponad 100 Gb/s.





Jakie możliwości daje hybrydowa platforma sprzętowa?

1. Możliwość dodania układów peryferyjnych do istniejącej architektury HPS – zasoby części FPGA układu mogą zostać wykorzystane do implementacji standardowych układów peryferyjnych (np. UART, SPI, I2C) w przypadku niewystarczającej ich liczby w części HPS.

2. Możliwość implementacji własnych układów peryferyjnych – zasoby części FPGA układu mogą zostać wykorzystane do implementacji dedykowanych (niestandardowych) układów peryferyjnych (zaimplementowanych przez użytkownika z wykorzystaniem języka opisu sprzętu VHDL/Verilog).

3. Możliwość podzielenia zadań na sprzętowe i programowe –
zadania łatwo realizowane sprzętowo (FPGA): obsługa czujników/przetworników, komunikacja niskopoziomowa, obliczenia na liczbach stałoprzecinkowych, filtracja sygnałów, generowanie sygnałów;
zadania łatwo realizowane programowo (HPS): obliczenia na liczbach zmiennoprzecinkowych (zwłaszcza podwójnej precyzji), obliczenia złożonych modeli matematycznych, komunikacja poprzez wysoko-poziomowe interfejsy (np. ethernet), algorytmy wymagające wykorzystania dużej ilości pamięci RAM do przechowywania danych.

Jakie możliwości daje hybrydowa platforma sprzętowa?

4. Możliwość podzielenia etapów pewnych zadań na sprzętowe i programowe – pewne etapy danego zadania mogą być realizowane sprzętowo (np. filtracja przychodzącego sygnału), a kolejne realizowane programowo (np. obliczenia i wysyłanie danych poprzez sieć ethernet).

5. Możliwość rekonfiguracji części FPGA układu „w locie" przez HPS – istnieje możliwość rekonfiguracji części FPGA układu SoC przez część HPS podczas działania układu; może to być wykorzystane do zmiany lub modyfikacji zadań realizowanych w części FPGA układu.

Cyclone V SoC
Możliwe konfiguracje zasilania i wykorzystania części HPS i FPGA

Części HPS i FPGA urządzenia mają oddzielne zewnętrzne źródła zasilania i mogą być włączane niezależnie. Można włączyć część HPS bez włączania części FPGA urządzenia. Jednak aby włączyć część FPGA, HPS musi być już włączony lub włączony jednocześnie z częścią FPGA.

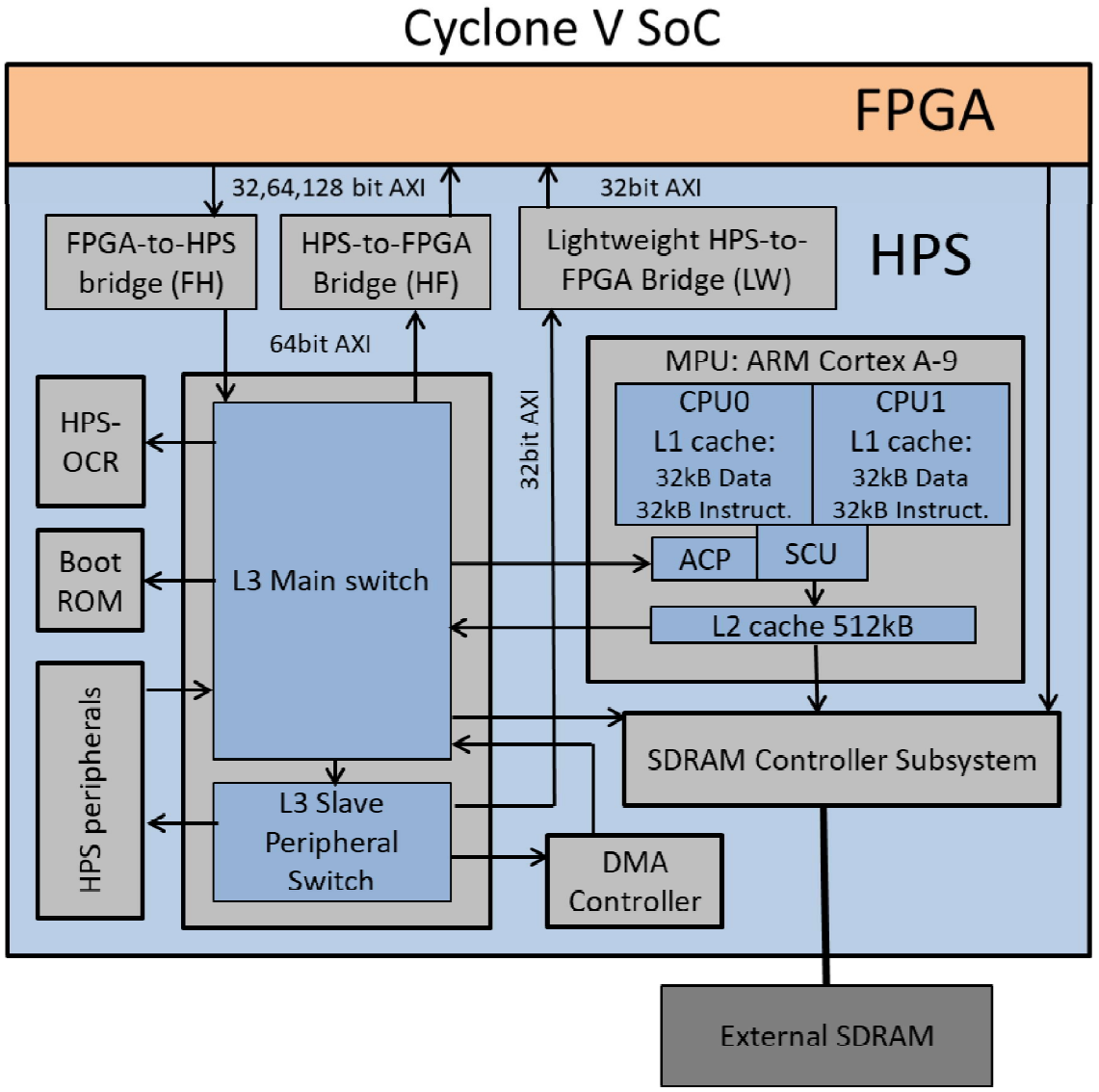
HPS Power	FPGA Power
On	On
On	Off
Off	Off

W związku z tym, możliwe jest użycie układu Cyclone V SoC w 4 różnych konfiguracjach:
- HPS & FPGA (części działają niezależnie od siebie – realizują różne zadania),
- HPS & FPGA (części współpracują ze sobą w celu realizacji zadań),
- tylko FPGA (część HPS musi być zasilana),
- tylko HPS.

Cyclone V SoC
Wymiana danych pomiędzy HPS a FPGA

Komunikacja między HPS a FPGA w układach Cyclone V SoC może być realizowana w następujący sposób:

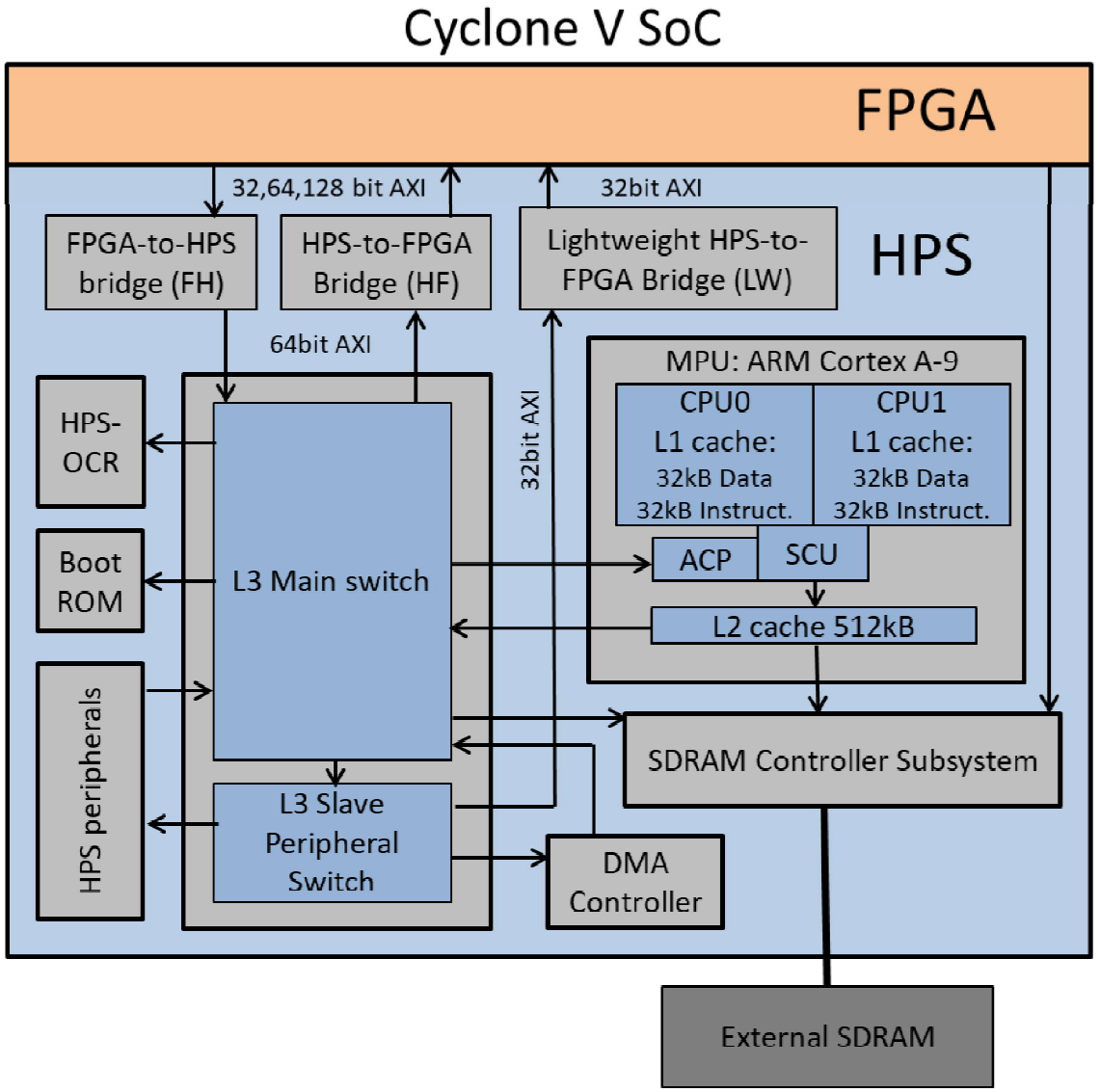
1) **HPS-to-FPGA bridge**
   Jednokierunkowy wysokowydajny interfejs z HPS do FPGA. Transakcje są zwykle realizowane przez procesor lub kontrolery DMA (Direct Memory Access) obecne w HPS. Mostek jest używany do dostępu do części FPGA, układów peryferyjnych i pamięci.
2) **HPS-to-FPGA Lightweight bridge**
   Dwukierunkowy interfejs o niskiej przepustowości danych do części FPGA. Zwykle wykorzystywany przez procesor do dostępu do rejestrów sterujących i statusowych komponentów zaimplementowanych w FPGA.



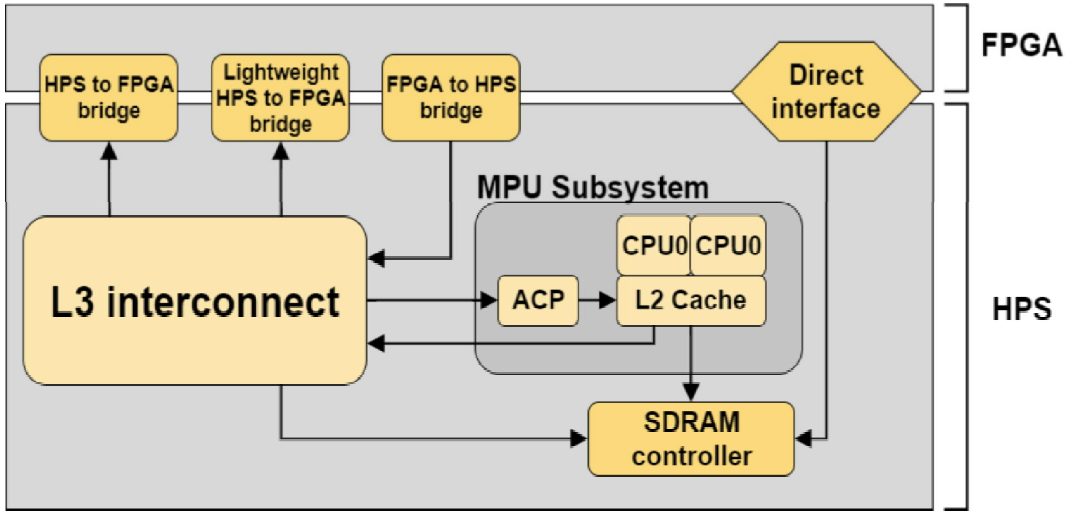
Cyclone V SoC
Wymiana danych pomiędzy HPS a FPGA

Komunikacja między HPS a FPGA w układach Cyclone V SoC może być realizowana w następujący sposób :

3) **FPGA-to-HPS bridge**
   Jednokierunkowy wysokowydajny interfejs z FPGA do układów peryferyjnych HPS i pamięci.
4) **FPGA-to-HPS SDRAM interface**
   Dwukierunkowy wysokowydajny interfejs z FPGA do kontrolera SDRAM. Część FPGA ma dostęp do pamięci RAM (DDRx) procesora. Dane znajdujące się w pamięci podręcznej (RAM) procesora są współdzielone przez części FPGA i HPS. Synchronizacja zapisu/odczytu danych pomiędzy FPGA i HPS musi być realizowana programowo.



Cyclone V SoC
Transfer danych z FPGA do SDRAM

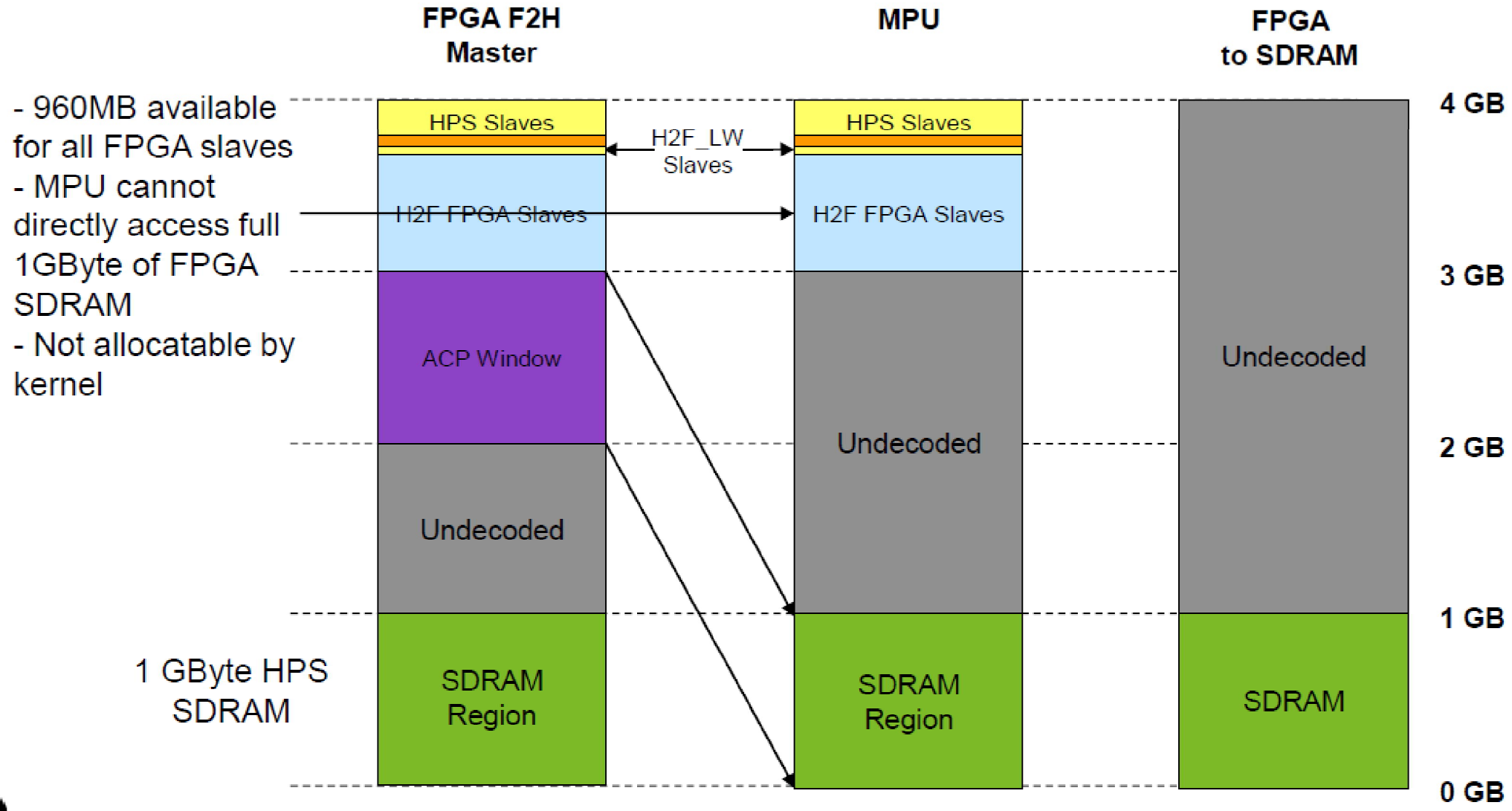


Data path	Bus width	Maximum throughput	Saturation frequency
FPGA-L3-SDRAM	32 bits	5.05 Gbps	120 MHz
	64 bits	10.10 Gbps	120 MHz
	128 bits	10.52 Gbps	65 MHz
FPGA-L3-ACP-SDRAM	32 bits	6.90 Gbps	-
	64 bits	8.64 Gbps	120 MHz
	128 bits	11.26 Gbps	90 MHz
FPGA-SDRAM	32 bits	7.52 Gbps	-
	64 bits	14.64 Gbps	-
	128 bits	17.68 Gbps	80 MHz
	256 bits	20.08 Gbps	45 MHz

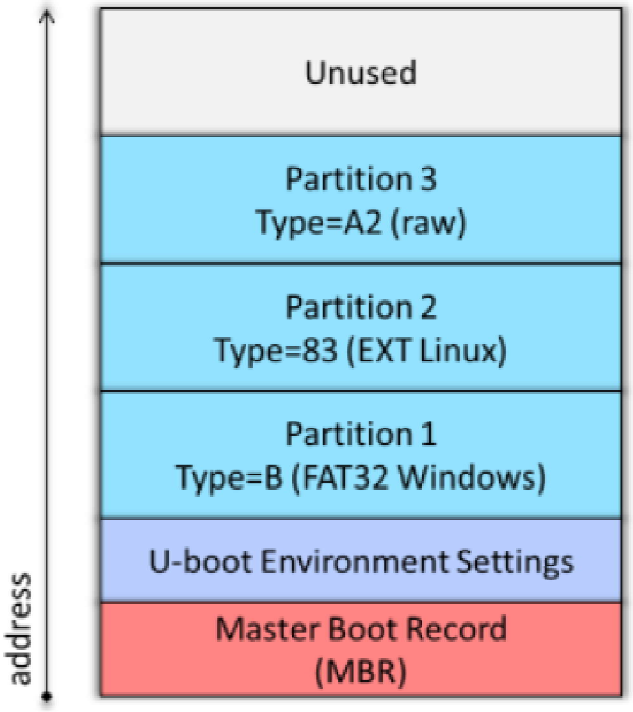
FPGA to SDRAM data transfer:

- **FPGA-SDRAM** – FPGA Master directly interacts with SDRAM controller.
- **FPGA-L3-SDRAM** – FPGA Master interacts with SDRAM controller via L3 interconnect.
- **FPGA-L3-ACP-SDRAM** – FPGA Master interacts with SDRAM controller via L3 interconnect and ACP.

Cyclone V SoC
Mapowanie pamięci



Cyclone V SoC
Obraz Karty SD



Location	File Name	Desciption
Partition 1 (FAT32)	socfpga.dtb	Device Tree Blob
	soc_system.rbf	FPGA configuration file
	u-boot.scr	U-Boot script: configures FPGA and loads kernel
	zImage	Compressed Linux kernel image file
Partition 2 (EXT3)	Various	Linux root file system
Partition 3 (A2 raw)	n/a	Preloader image(s)
	n/a	U-Boot image

Cyclone V SoC
HSP – Układy peryferyjne

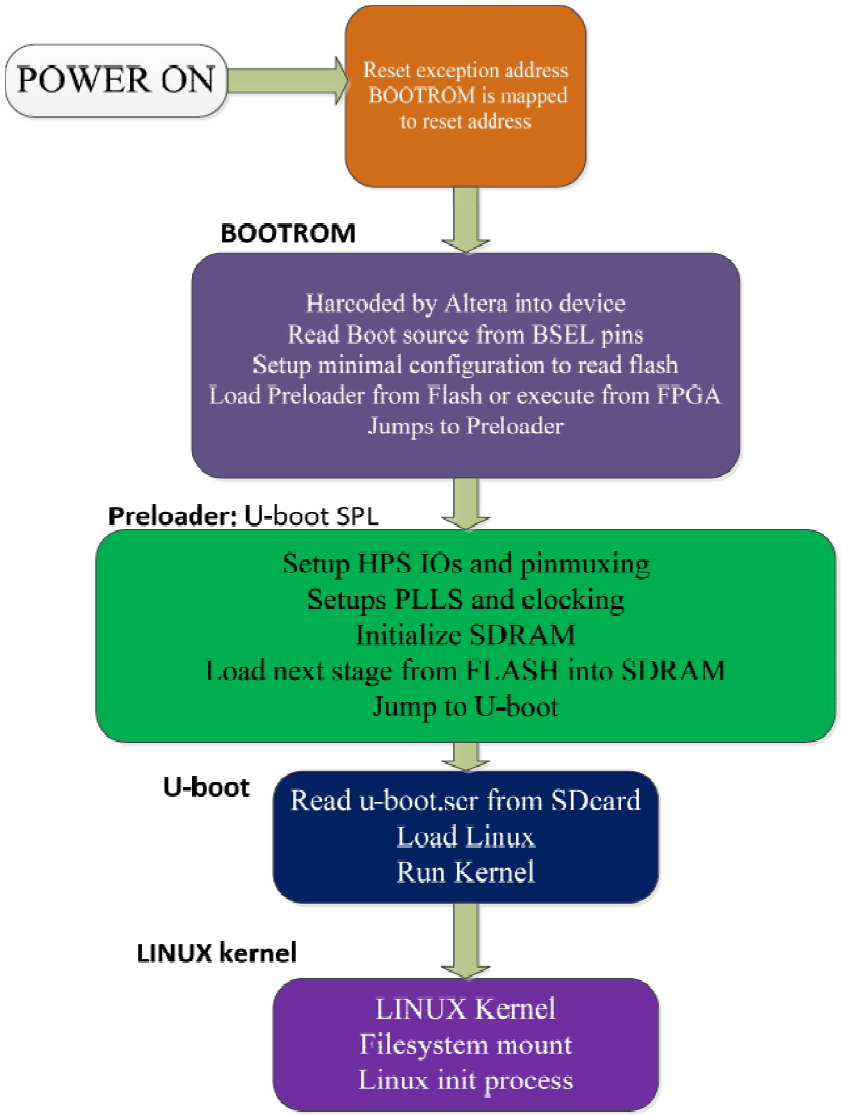
Możliwości wykorzystania układów peryferyjnych:

- Układ peryferyjny HSP (np. UART, SPI) + dedykowane piny po stronie HPS;
- Układ peryferyjny HPS + dowolne piny pod stronie FPGA;
- Piny ogólnego przeznaczenia GPIO po stronie HPS;
- Piny ogólnego przeznaczenia GPIO po stronie FPGA kontrolowane przez HPS (LOANIO).

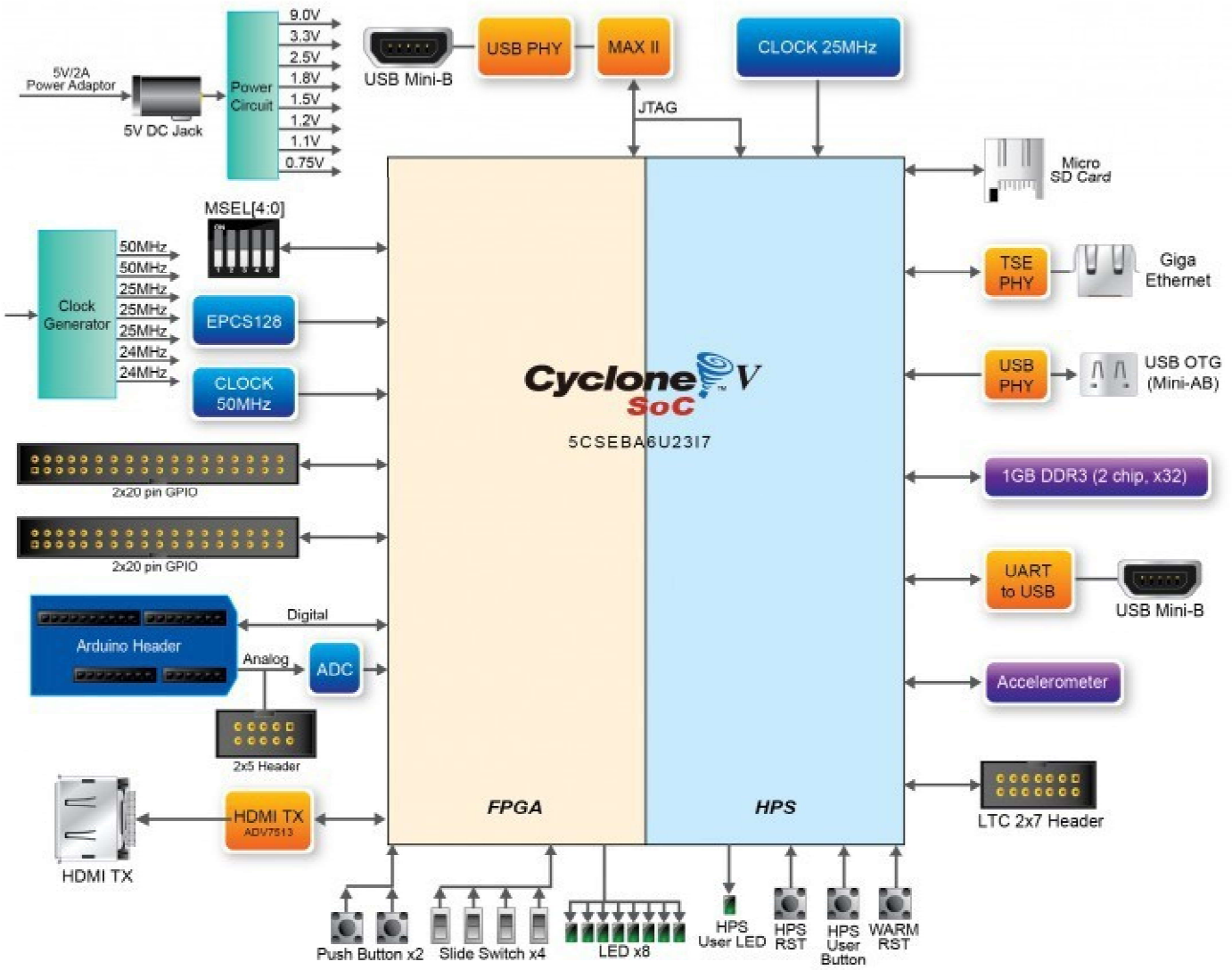
Cyclone V SoC
Tablica konfiguracji pinów dla układów peryferyjnych HPS

Peripherals Mux Table					
RGMII0_TX_CLK			EMAC0.TX_CLK (Set0)	GPIO00	LOANIO00
RGMII0_TXD0		USB1.D0 (Set0)	EMAC0.TXD0 (Set0)	GPIO01	LOANIO01
RGMII0_TXD1		USB1.D1 (Set0)	EMAC0.TXD1 (Set0)	GPIO02	LOANIO02
RGMII0_TXD2		USB1.D2 (Set0)	EMAC0.TXD2 (Set0)	GPIO03	LOANIO03
RGMII0_TXD3		USB1.D3 (Set0)	EMAC0.TXD3 (Set0)	GPIO04	LOANIO04
RGMII0_RXD0		USB1.D4 (Set0)	EMAC0.RXD0 (Set0)	GPIO05	LOANIO05
RGMII0_MDIO	I2C2.SDA (Set0)	USB1.D5 (Set0)	EMAC0.MDIO (Set0)	GPIO06	LOANIO06
RGMII0_MDC	I2C2.SCL (Set0)	USB1.D6 (Set0)	EMAC0.MDC (Set0)	GPIO07	LOANIO07
RGMII0_RX_CTL		USB1.D7 (Set0)	EMAC0.RX_CTL (Set0)	GPIO08	LOANIO08
RGMII0_TX_CTL			EMAC0.TX_CTL (Set0)	GPIO09	LOANIO09
RGMII0_RX_CLK		USB1.CLK (Set0)	EMAC0.RX_CLK (Set0)	GPIO10	LOANIO10
RGMII0_RXD1		USB1.STP (Set0)	EMAC0.RXD1 (Set0)	GPIO11	LOANIO11
RGMII0_RXD2		USB1.DIR (Set0)	EMAC0.RXD2 (Set0)	GPIO12	LOANIO12
RGMII0_RXD3		USB1.NXT (Set0)	EMAC0.RXD3 (Set0)	GPIO13	LOANIO13
NAND_ALE	QSPI.SS3 (Set1) (Set0)	EMAC1.TX_CLK (Set0)	NAND.ALE (Set0)	GPIO14	LOANIO14
NAND_CE	USB1.D0 (Set1)	EMAC1.TXD0 (Set0)	NAND.CE (Set0)	GPIO15	LOANIO15
NAND_CLE	USB1.D1 (Set1)	EMAC1.TXD1 (Set0)	NAND.CLE (Set0)	GPIO16	LOANIO16
NAND_RE	USB1.D2 (Set1)	EMAC1.TXD2 (Set0)	NAND.RE (Set0)	GPIO17	LOANIO17
NAND_RB	USB1.D3 (Set1)	EMAC1.TXD3 (Set0)	NAND.RB (Set0)	GPIO18	LOANIO18
NAND_DQ0		EMAC1.RXD0 (Set0)	NAND.DQ0 (Set0)	GPIO19	LOANIO19
NAND_DQ1	I2C3.SDA (Set0)	EMAC1.MDIO (Set0)	NAND.DQ1 (Set0)	GPIO20	LOANIO20
NAND_DQ2	I2C3.SCL (Set0)	EMAC1.MDC (Set0)	NAND.DQ2 (Set0)	GPIO21	LOANIO21
NAND_DQ3	USB1.D4 (Set1)	EMAC1.RX_CTL (Set0)	NAND.DQ3 (Set0)	GPIO22	LOANIO22
NAND_DQ4	USB1.D5 (Set1)	EMAC1.TX_CTL (Set0)	NAND.DQ4 (Set0)	GPIO23	LOANIO23
NAND_DQ5	USB1.D6 (Set1)	EMAC1.RX_CLK (Set0)	NAND.DQ5 (Set0)	GPIO24	LOANIO24
NAND_DQ6	USB1.D7 (Set1)	EMAC1.RXD1 (Set0)	NAND.DQ6 (Set0)	GPIO25	LOANIO25
NAND_DQ7		EMAC1.RXD2 (Set0)	NAND.DQ7 (Set0)	GPIO26	LOANIO26
NAND_WP	QSPI.SS2 (Set1) (Set0)	EMAC1.RXD3 (Set0)	NAND.WP (Set0)	GPIO27	LOANIO27
NAND_WE		QSPI.SS1 (Set0)	NAND.WE (Set0)	GPIO28	LOANIO28
QSPI_IO0	USB1.CLK (Set1)		QSPI.IO0 (Set1) (Set0)	GPIO29	LOANIO29
QSPI_IO1	USB1.STP (Set1)		QSPI.IO1 (Set1) (Set0)	GPIO30	LOANIO30
QSPI_IO2	USB1.DIR (Set1)		QSPI.IO2 (Set1) (Set0)	GPIO31	LOANIO31
QSPI_IO3	USB1.NXT (Set1)		QSPI.IO3 (Set1) (Set0)	GPIO32	LOANIO32
QSPI_SS0			QSPI.SS0 (Set1) (Set0)	GPIO33	LOANIO33
QSPI_CLK			QSPI.CLK (Set1) (Set0)	GPIO34	LOANIO34
QSPI_SS1			QSPI.SS1 (Set1)	GPIO35	LOANIO35

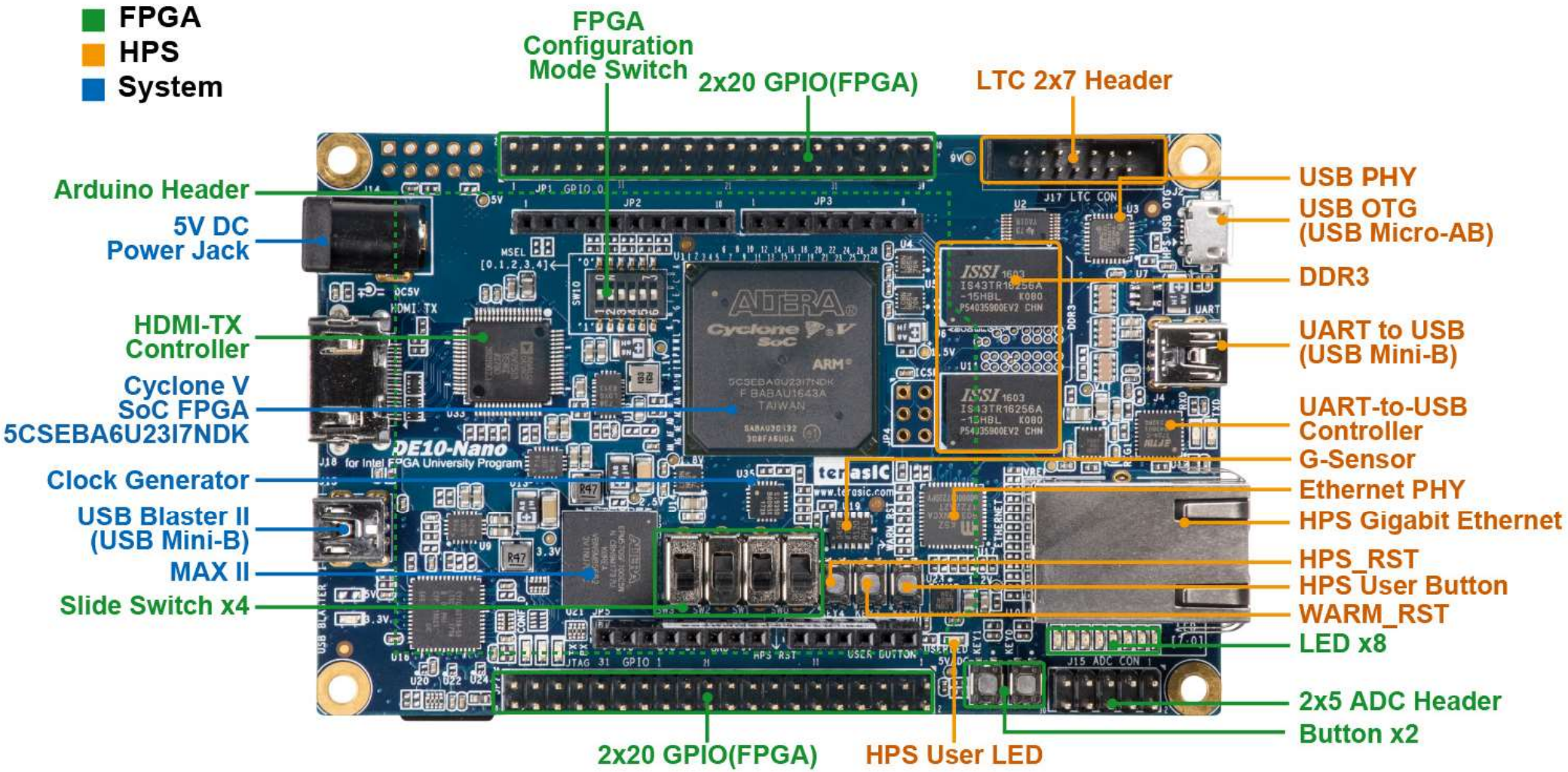
Cyclone V SoC
Proces uruchamiania HPS



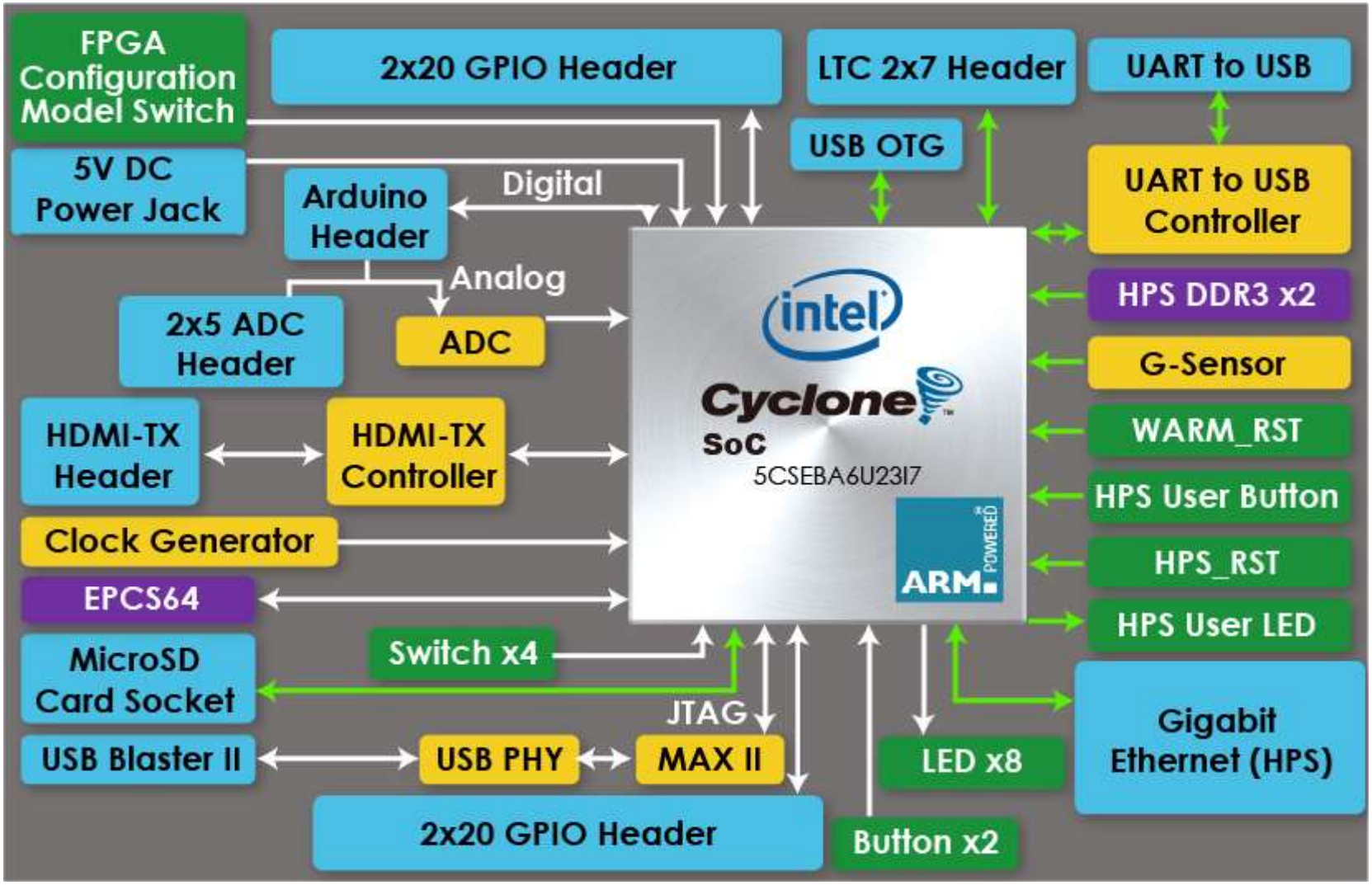
DE10 -Nano


5V/2A
Power Adaptor
5V DC Jack
Power
Circuit
9.0V
3.3V
2.5V
1.8V
1.5V
1.2V
1.1V
0.75V
USB Mini-B
USB PHY
MAX II
CLOCK 25MHz
JTAG
Micro
SD Card
MSEL[4:0]
Clock
Generator
50MHz
50MHz
25MHz
25MHz
25MHz
24MHz
24MHz
EPCS128
CLOCK
50MHz
TSE
PHY
Giga
Ethernet
USB
PHY
USB OTG
(Mini-AB)
Cyclone V
SoC
5CSEBA6U23I7
2x20 pin GPIO
2x20 pin GPIO
1GB DDR3 (2 chip, x32)
UART
to USB
USB Mini-B
Arduino Header
Digital
Analog
ADC
Accelerometer
2x5 Header
HDMI TX
ADV7513
HDMI TX
FPGA
HPS
LTC 2x7 Header
Push Button x2
Slide Switch x4
LED x8
HPS
User LED
HPS
RST
HPS
User
Button
WARM
RST

DE10 -Nano


FPGA
HPS
System
FPGA Configuration Mode Switch
2x20 GPIO(FPGA)
LTC 2x7 Header
Arduino Header
5V DC Power Jack
HDMI-TX Controller
Cyclone V SoC FPGA 5CSEBA6U23I7NDK
Clock Generator
USB Blaster II (USB Mini-B)
MAX II
Slide Switch x4
USB PHY
USB OTG (USB Micro-AB)
DDR3
UART to USB (USB Mini-B)
UART-to-USB Controller
G-Sensor
Ethernet PHY
HPS Gigabit Ethernet
HPS_RST
HPS User Button
WARM_RST
LED x8
2x5 ADC Header
Button x2
2x20 GPIO(FPGA)
HPS User LED

DE10 - Nano


FPGA Configuration Model Switch
2x20 GPIO Header
LTC 2x7 Header
UART to USB
5V DC Power Jack
Digital
Arduino Header
USB OTG
UART to USB Controller
Analog
2x5 ADC Header
ADC
HPS DDR3 x2
G-Sensor
HDMI-TX Header
HDMI-TX Controller
intel
Cyclone
SoC
5CSEBA6U23I7
WARM_RST
HPS User Button
Clock Generator
ARM
HPS_RST
EPCS64
HPS User LED
MicroSD Card Socket
Switch x4
JTAG
Gigabit Ethernet (HPS)
USB Blaster II
USB PHY
MAX II
LED x8
2x20 GPIO Header
Button x2

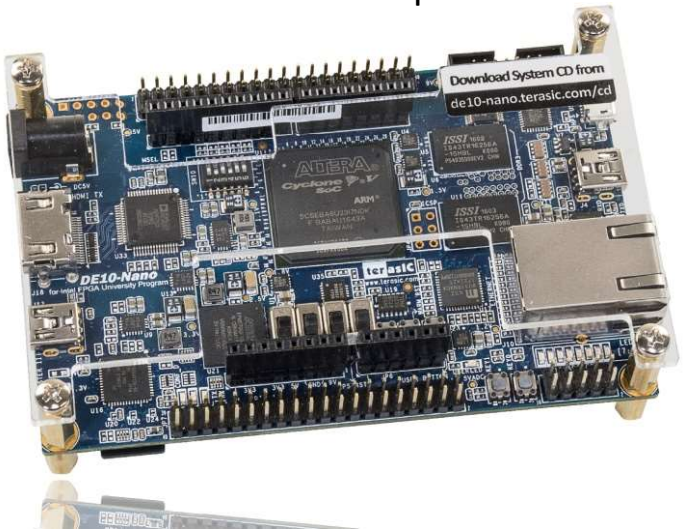
DE10 - Nano

FPGA Device

- Intel Cyclone® V SE 5CSEBA6U23I7 device (110K LEs)
- Serial configuration device – EPCS64 (revision B2 or later)
- USB-Blaster II onboard for programming; JTAG Mode
- HDMI TX, compatible with DVI 1.0 and HDCP v1.4
- 2 push-buttons
- 4 slide switches
- 8 green user LEDs
- Three 50MHz clock sources from the clock generator
- Two 40-pin expansion headers
- One Arduino expansion header (Uno R3 compatibility), can be connected with Arduino shields
- One 10-pin Analog input expansion header (shared with Arduino Analog input)
- A/D converter, 4-pin SPI interface with FPGA

HPS (Hard Processor System)

- 800MHz Dual-core ARM Cortex-A9 processor
- 1GB DDR3 SDRAM (32-bit data bus)
- 1 Gigabit Ethernet PHY with RJ45 connector
- USB OTG Port, USB Micro-AB connector
- Micro SD card socket
- Accelerometer (I2C interface + interrupt)
- UART to USB, USB Mini-B connector
- Warm reset button and cold reset button
- One user button and one user LED
- LTC 2x7 expansion header



Cyclone V SoC
Jak zacząć?

1. Przygotować kartę SD z obrazem Linuxa

1.1 Ściągnąć Linuxa z https://www.intel.com/content/www/us/en/developer/topic-technology/fpga-academic/materials-sd-card-images.html

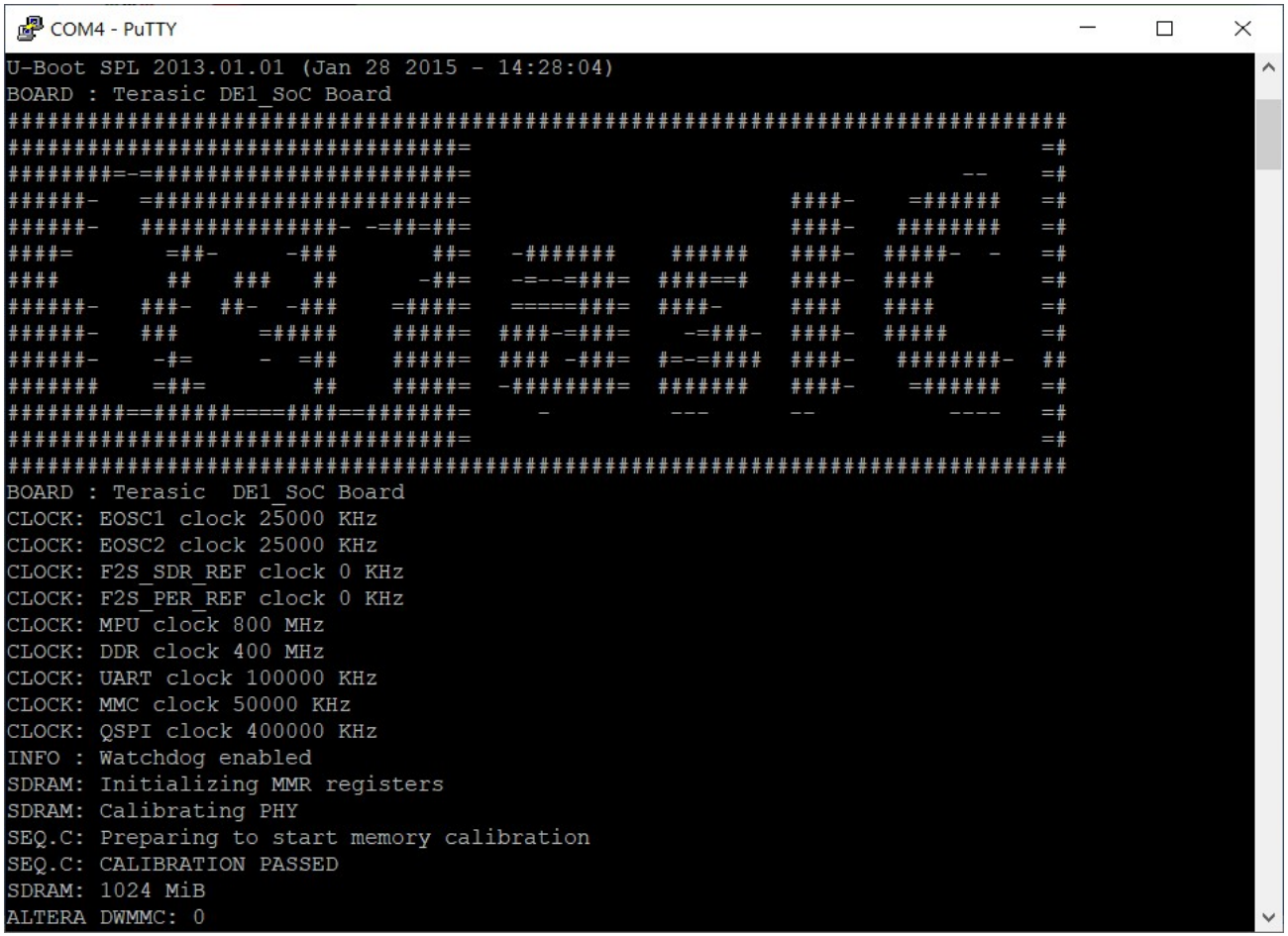
1.2a Wgrać obraz na kartę SD (Windows):

1.2b Wgrać obraz na kartę SD (Linux):

> sudo dd if=*PATH_TO_IMAGE_FILE* of=*PATH_TO_SDCARD_DEVICE*

Cyclone V SoC
Jak zacząć?

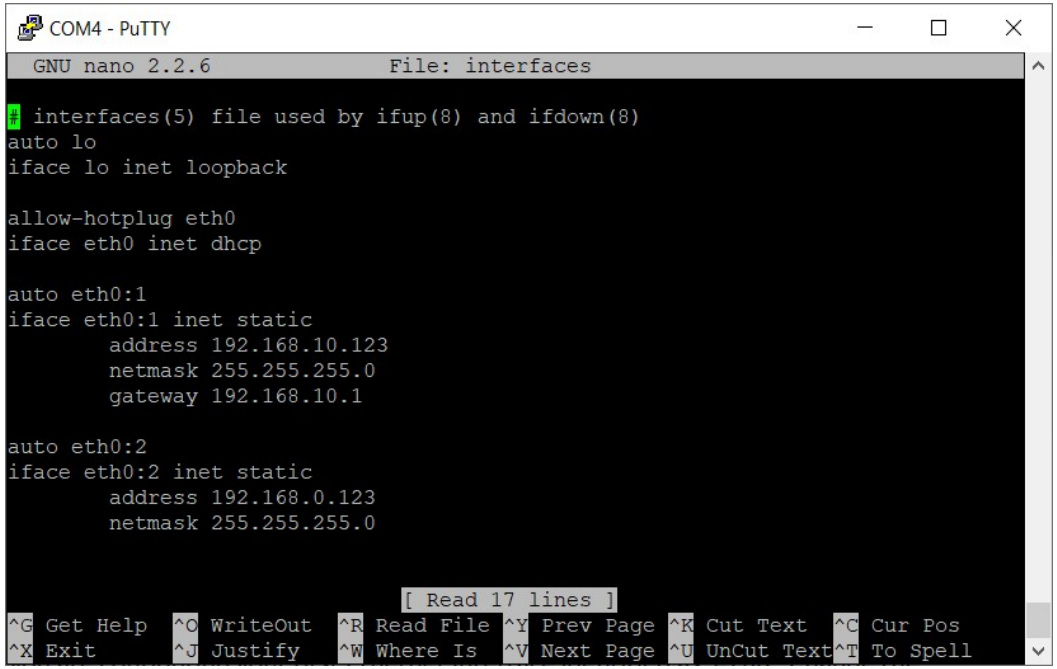
2. Wykorzystać terminal (np. Putty) do komunikacji z HPS poprzez interfejs szeregowy UART (Wirtualny port COM)



Cyclone V SoC
Jak zacząć?

3. Ustawić statyczny (lub dynamiczny) adres IP

> sudo nano /etc/network/interfaces



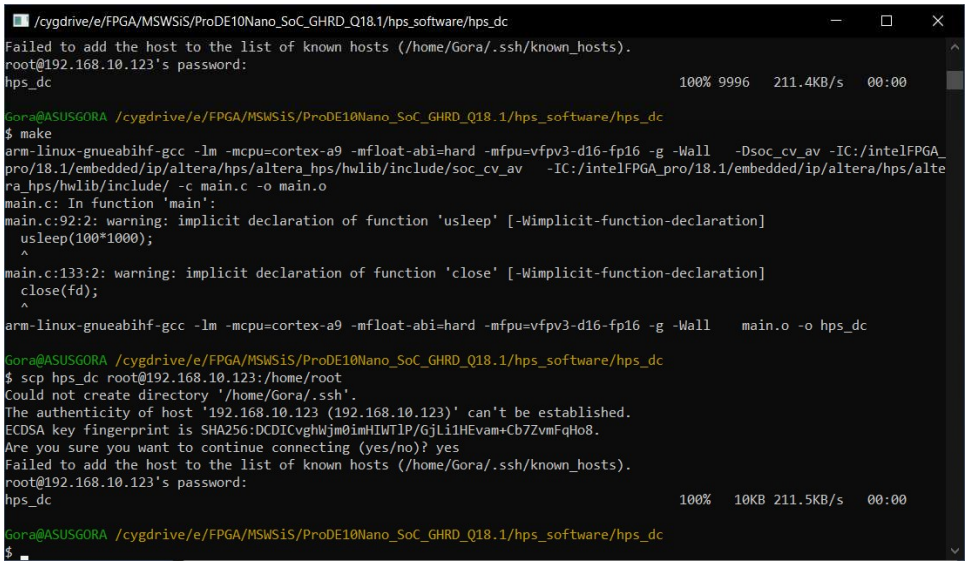
4. Ustawić hasło w terminalu

> passwd

5. Połączyć się z HPS poprzez SSH wykorzystując interfejs Ethernet

Część HPS – Program w języku C/C++
(wykorzystanie tylko HPS lub HPS niezależnie od FPGA)

1. Uruchomienie *SoC EDS Command Shell* jako terminala na komputerze PC



2. Napisanie i skompilowanie programu wykorzystując Makefile

> make

3. Przesłanie programu do HPS wykorzystując np. SCP

> scp *PROG_NAME* USER_NAME@IP_ADDRESS:/PATH_TO_COPY

np.:

> scp hps_soft root@192.168.10.123:/home/root

Część HPS – Układy peryferyjne

(wykorzystanie tylko HPS lub HPS niezależnie od FPGA)

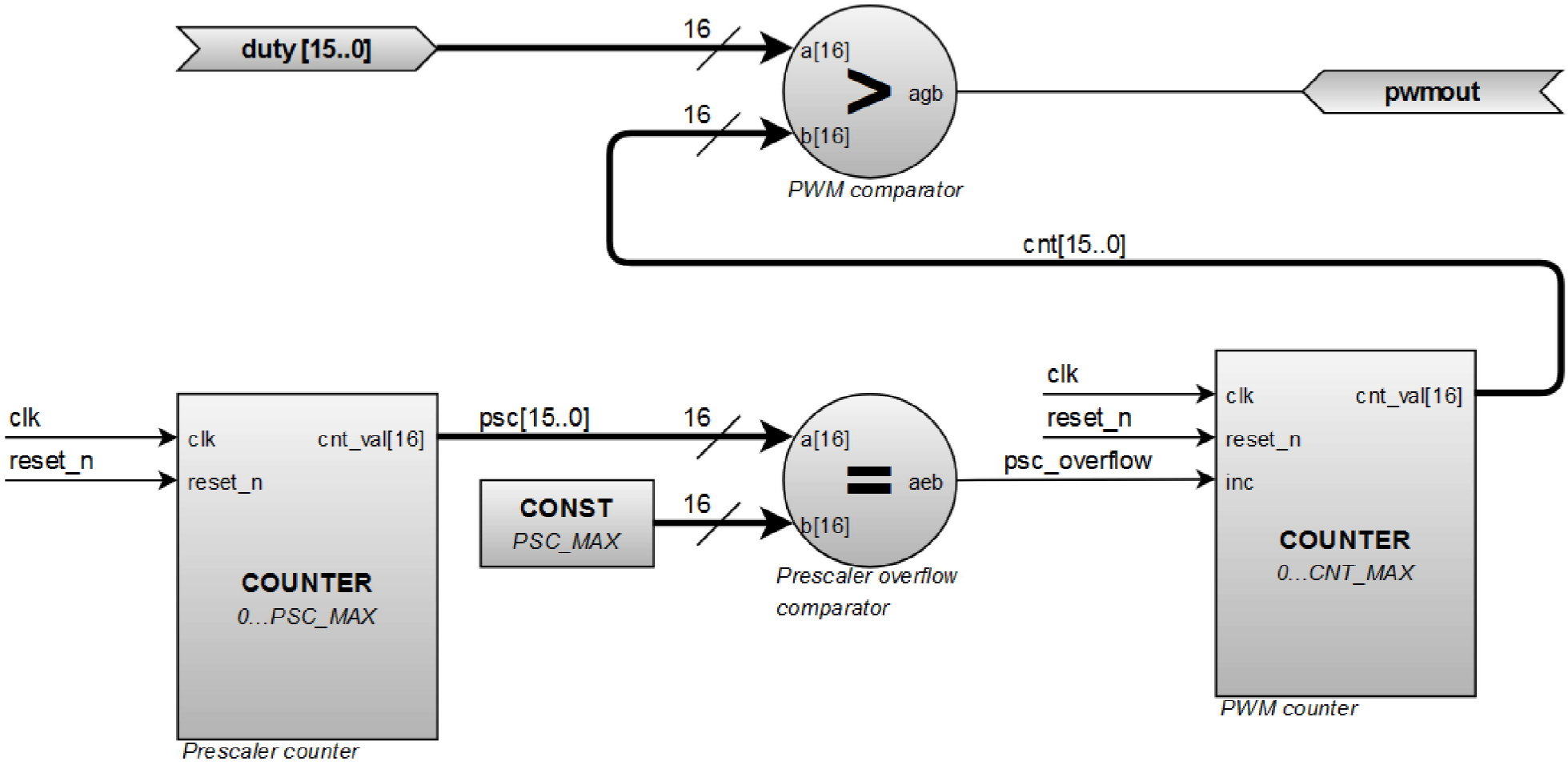
Część HPS może być traktowana jako niezależny układ mikroprocesorowy (mikrokontroler). Może pracować pod kontrolą systemu operacyjnego (np. Linux) w wersji konsolowej lub graficznej (GUI). Układy peryferyjne występujące w części HPS:

- Ethernet Media Access Controller,
- NAND Flash Controller,
- Quad SPI Flash Controller,
- SD/MMC Controller,
- USB Controller (x2),
- SPI Controller (x4),
- UART Controller (x2),
- I2C Controller (x4),
- CAN Controller (x2),
- GPIO Ports.

Część FPGA – Moduł sprzętowy: Modulator PWM

(wykorzystanie tylko FPGA lub FPGA niezależnie od HPS)

Architektura sprzętowa Modulatora PWM:

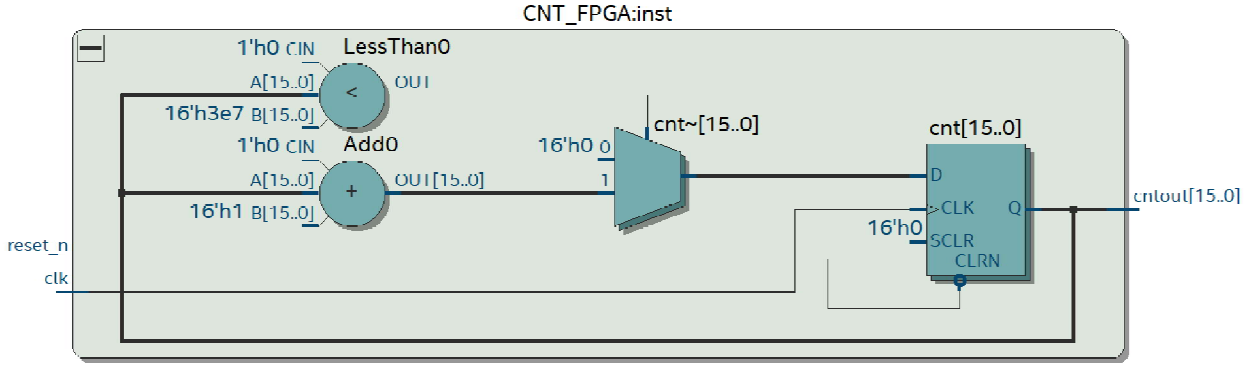


Część FPGA – konstrukcja „process” i „when” w języku VHDL

(wykorzystanie tylko FPGA lub FPGA niezależnie od HPS)

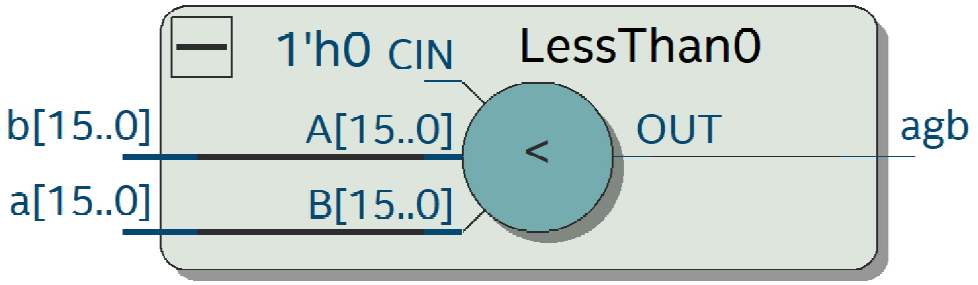
Konstrukcja „process” wykorzystana do zaimplementowania licznika (układu sekwencyjnego):

```
process(clk, reset_n)
begin
   if(reset_n = '0')then
      cnt <= 16d"0";
   elsif rising_edge(clk) then
      if(cnt < CNT_MAX)then
         cnt <= cnt + 16d"1";
      else
         cnt <= 16d"0";
      end if;
   end if;
end process;
```



Konstrukcja „when” wykorzystana do zaimplementowania komparatora (układu kombinacyjnego):

```
agb <= '1' when(a > b) else '0';
```



Część FPGA – Moduł sprzętowy: Enkoder inkrementalny

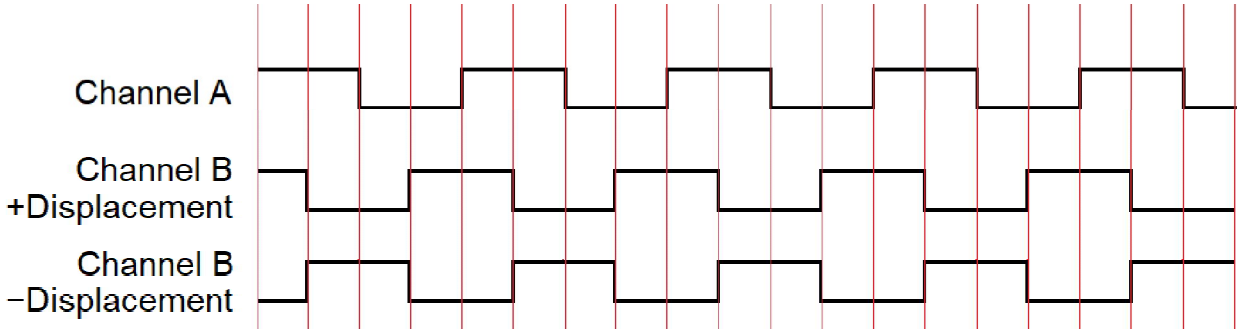
(wykorzystanie tylko FPGA lub FPGA niezależnie od HPS)

Enkoder inkrementalny (enkoder przyrostowy, przetwornik obrotowo-impulsowy) – pozwala na wyznaczenie pozycji względnej (względem pozycji początkowej po włączeniu). Najczęściej występuje z interfejsem kwadraturowym 2 lub 3 kanałowym.

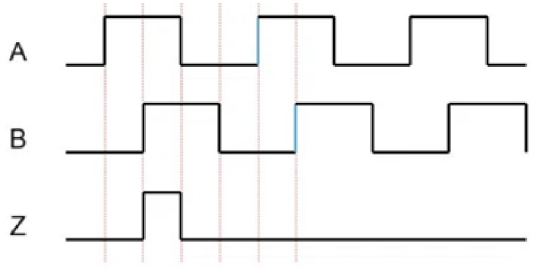
Możliwe typy wyjść:

- **Push-Pull** – sygnał cyfrowy w standardzie (najczęściej) TTL (5V) lub LVTTL (3V3);
- **Open Collector** – wymaga zewnętrznego rezystory typu Pull-up;
- **różnicowy** – wymaga zewnętrznego konwertera standardów napięciowych RS422 na TTL/LVTTL.

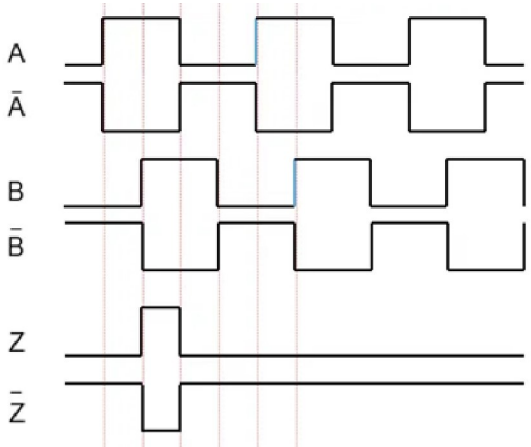
Enkoder z interfejsem 2-kanałowym AB:



Enkoder z interfejsem 3-kanałowym ABZ (wyjście PP/OC):



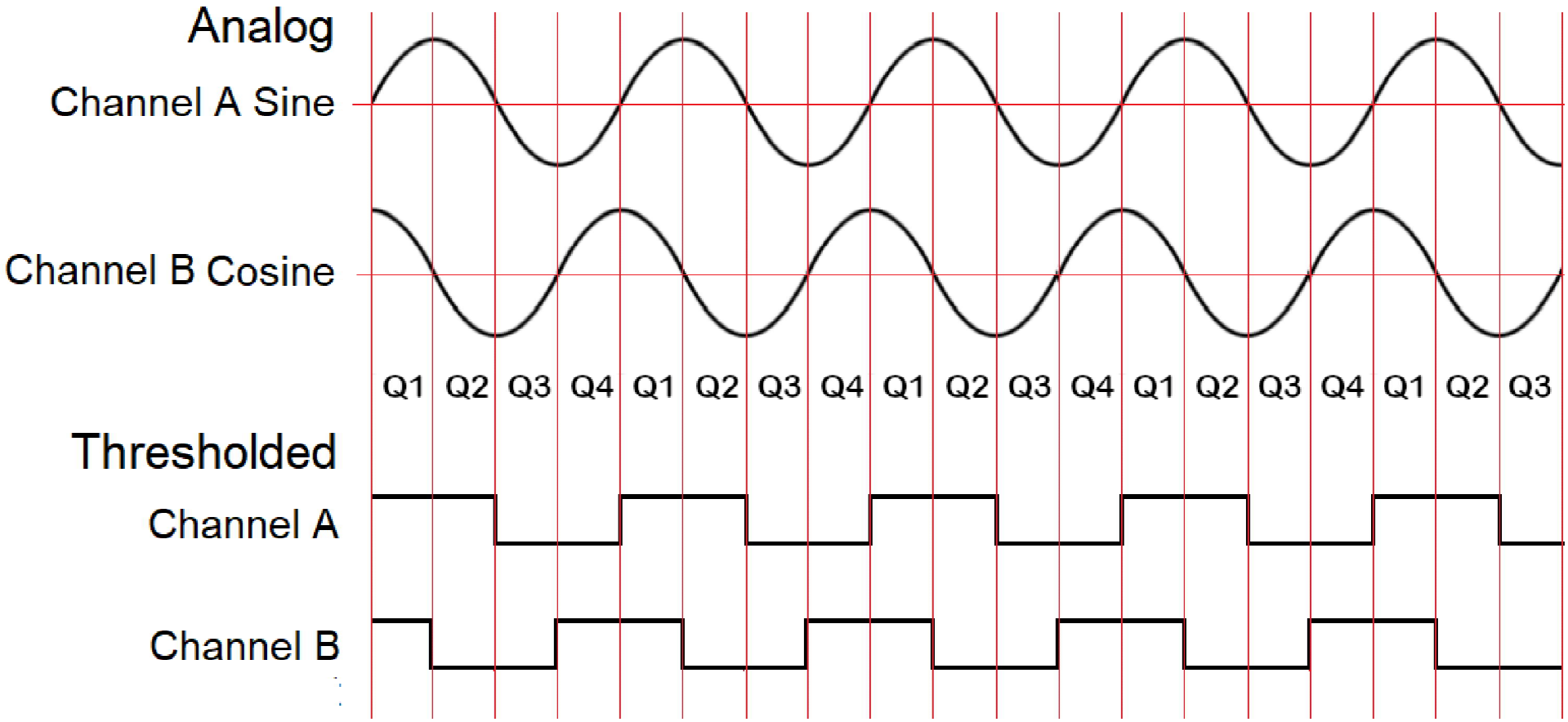
Enkoder z interfejsem 3-kanałowym ABZ (wyjście różnicowe):



Część FPGA – Moduł sprzętowy: Enkoder inkrementalny

(wykorzystanie tylko FPGA lub FPGA niezależnie od HPS)

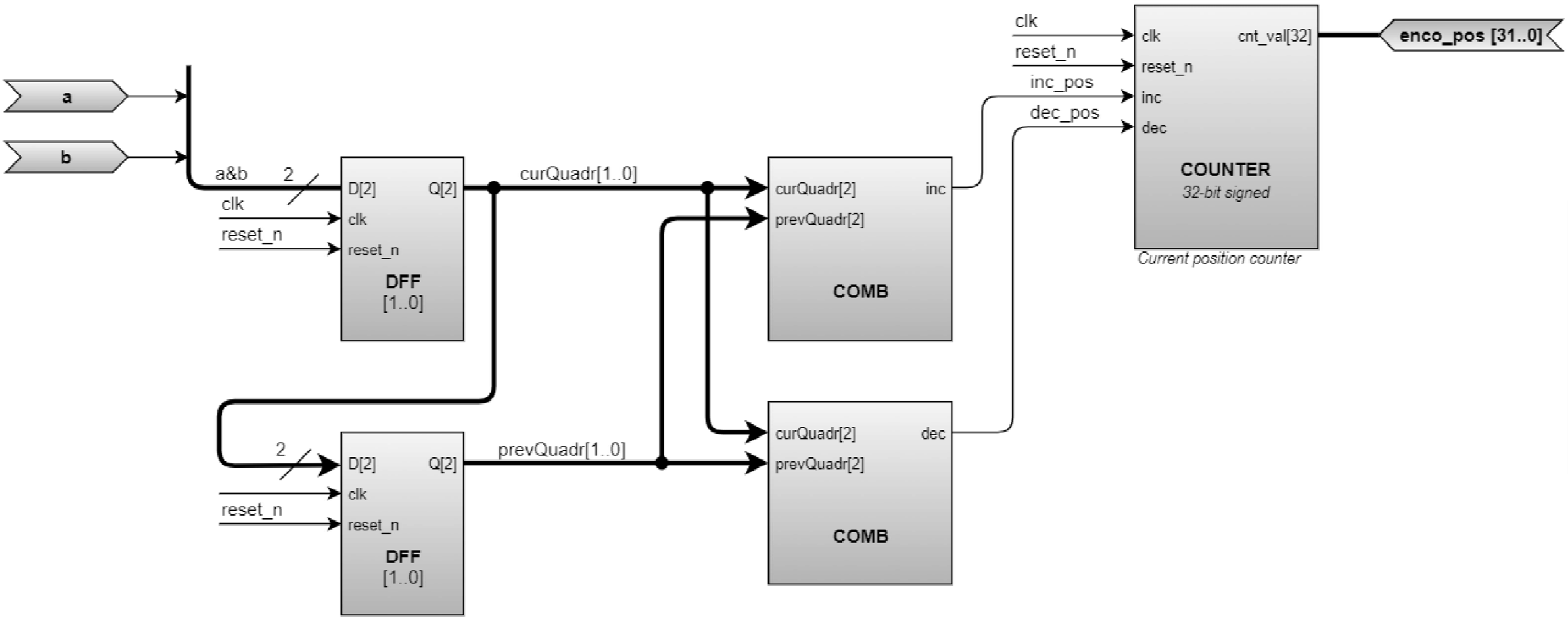
Enkoder z interfejsem 2-kanałowym AB:



Część FPGA – Moduł sprzętowy: Enkoder inkrementalny

(wykorzystanie tylko FPGA lub FPGA niezależnie od HPS)

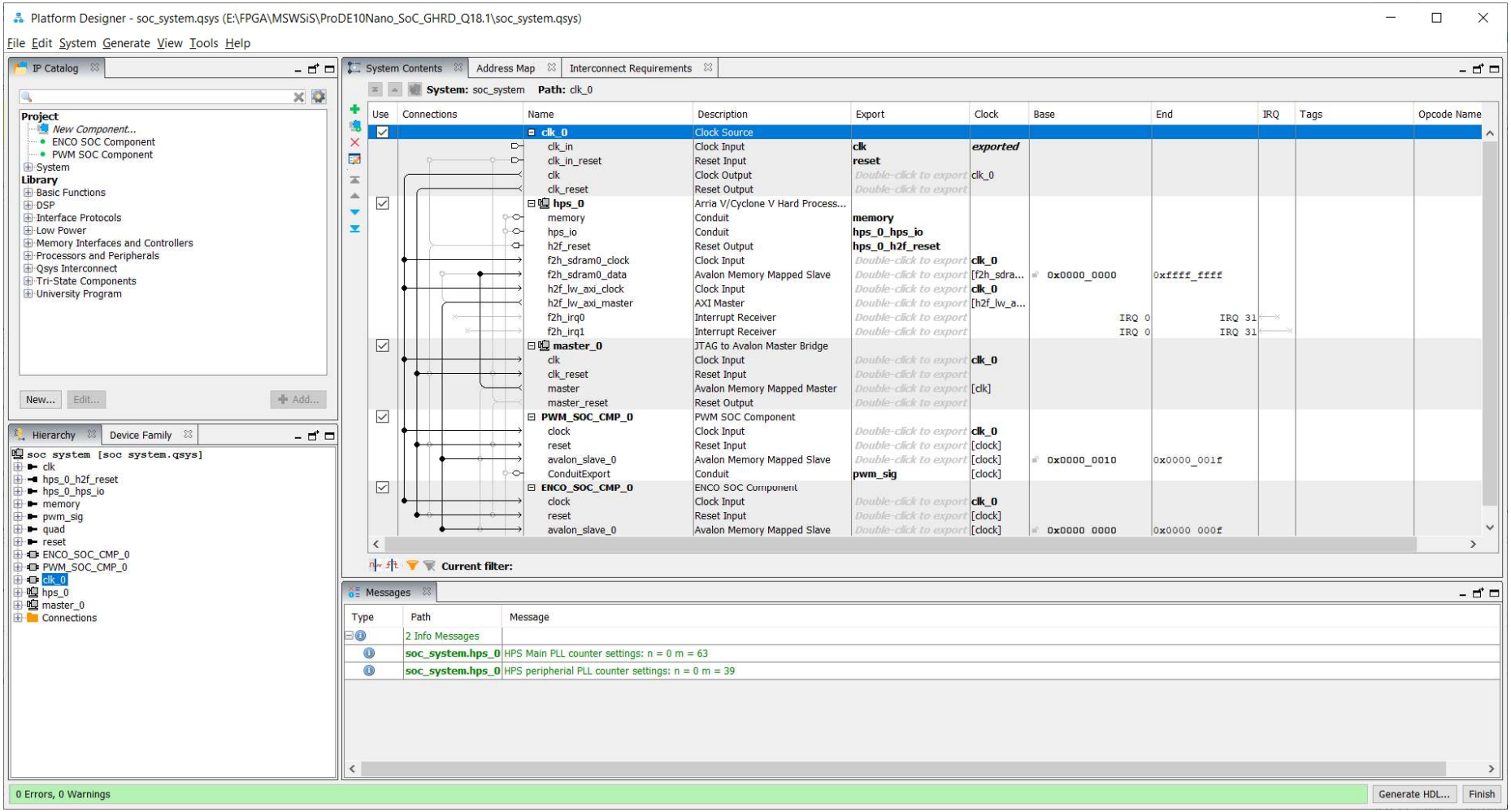
Architektura sprzętowa Modułu obsługi Enkodera inkrementalnego:



FPGA + HPS – Platform Designer

(wykorzystanie części FPGA i HPS do realizacji wspólnego zadania)

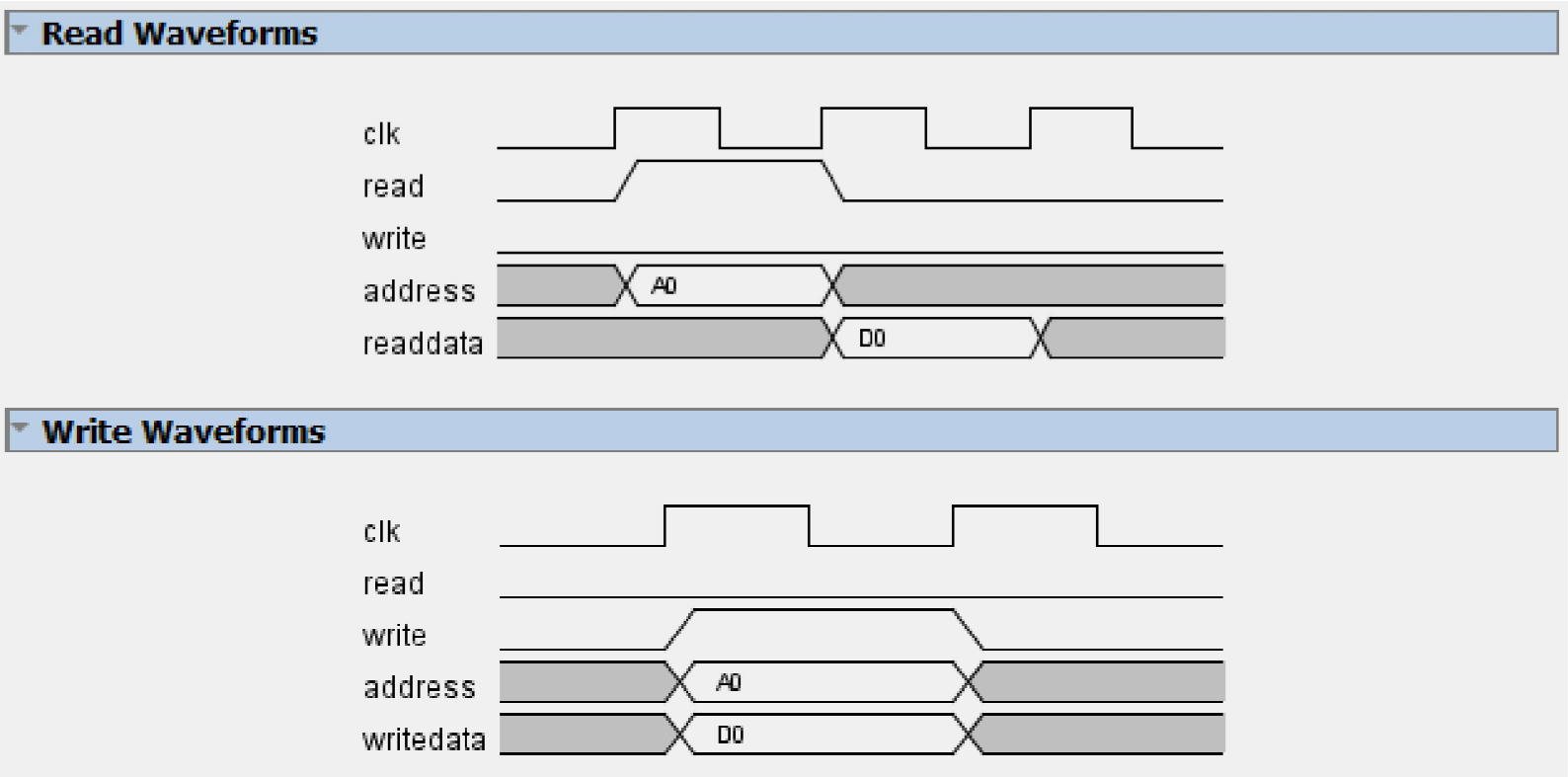
Platform Designer – to narzędzie do konfiguracji architektury części HPS systemu.



FPGA + HPS – Avalon Memory Mapped Slave Interfaces

(wykorzystanie części FPGA i HPS do realizacji wspólnego zadania)

Avalone-MMS – jest interfejsem pozwalającym na komunikację pomiędzy rdzeniem mikroprocesorowym a układami peryferyjnymi (np. UART, I2C, SPI). Od strony układu mikroporcesorowego komunikacja polega na zapisywaniu/odczytywaniu danych z pewnej przestrzeni pamięci (przestrzeni adresowej). Od strony sprzętowej (FPGA) komunikacja polega na reagowaniu na żądania zapisu/odczytu parametru o zdefiniowanym adresie wykorzystując magistralę Avalone.



FPGA + HPS – Komunikacja poprzez magistralę Avalone (VHDL)

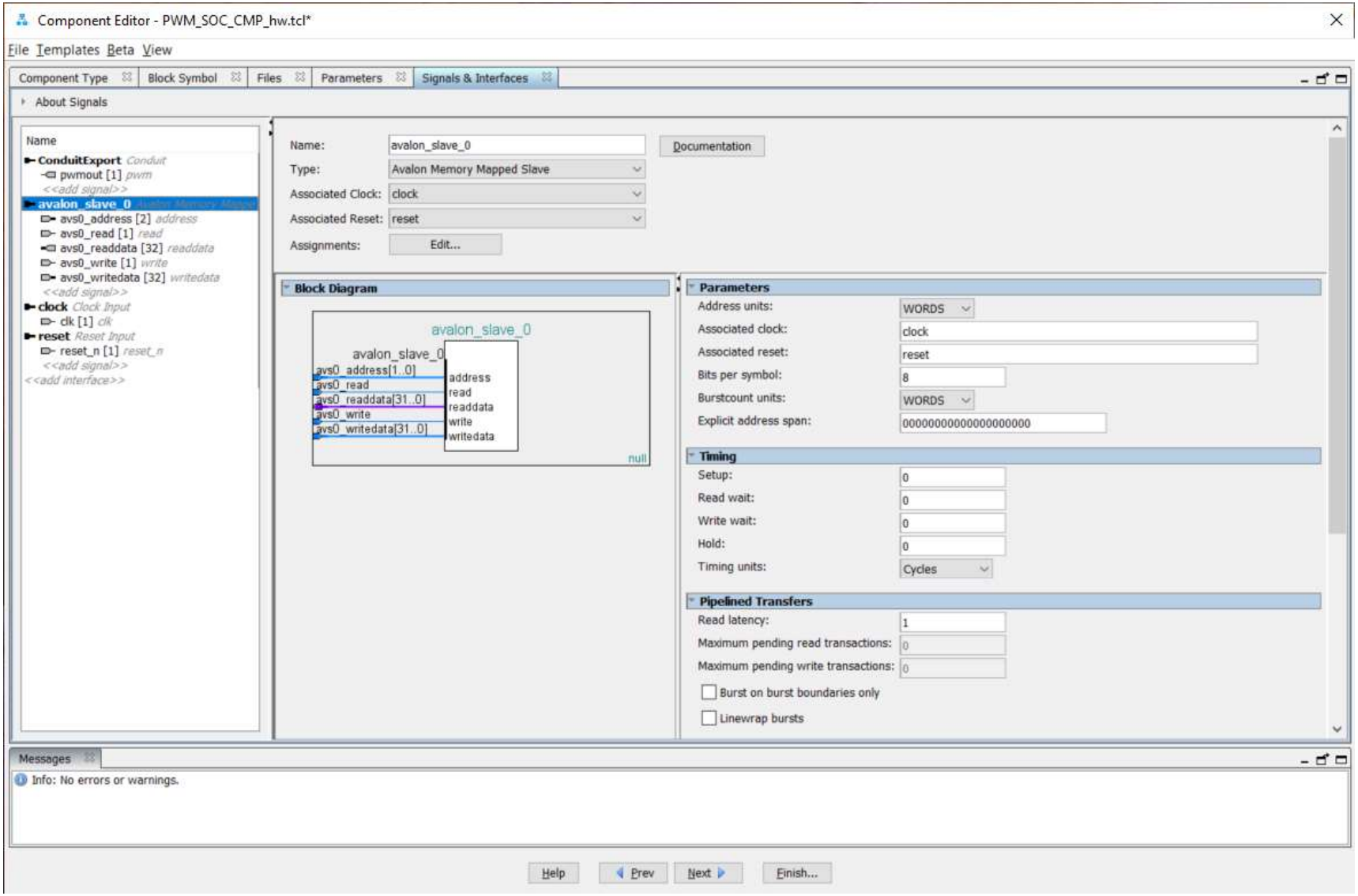
(wykorzystanie części FPGA i HPS do realizacji wspólnego zadania)

```
-- ----------------------------------------------------------
-- Avalone Slave S0 Interface - HPS to FPGA
-- ----------------------------------------------------------

process(clk, reset_n)
begin
   if(reset_n = '0')then
      avs0_readdata <= (others => '0');
   elsif rising_edge(clk) then
      if(avs0_write = '1')then
         case avs0_address is
            when AVS0_ADDR_REG0 => REG0 <= avs0_writedata;
            when AVS0_ADDR_REG1 => REG1 <= avs0_writedata;
            when AVS0_ADDR_REG2 => REG2 <= avs0_writedata;
            ...
            when others => null;
         end case;
      elsif(avs0_read = '1')then
         case avs0_address is
            when AVS0_ADDR_REG0 => avs0_readdata <= REG0;
            when AVS0_ADDR_REG1 => avs0_readdata <= REG1;
            when AVS0_ADDR_REG2 => avs0_readdata <= REG2;
            ...
            when others => avs0_readdata <= (others => '0');
         end case;
      end if;
   end if;
end process;
```

FPGA + HPS – Moduł sprzętowy z interfejsem Avalone-MMS

(wykorzystanie części FPGA i HPS do realizacji wspólnego zadania)



FPGA + HPS – Komunikacja ukł. mikroporcesorowego z ukł. peryferyjnym (C/C++)

(wykorzystanie części FPGA i HPS do realizacji wspólnego zadania)

```
// Lightweight HPS-to-FPGA Bridge
// ------------------------------------------------------------------------
#define LWH2F_REG_BASE          0xFF200000  // LW H2F Bridge Base Address
#define LWH2F_REG_SPAN          0x00200000  // LW H2F Bridge Span

// HPS COMPONENTS BASE ADDRESSES
#define ENCOSOC_BASE            0x00000000 // ENCO_SOC_CMP Address
#define PWMSOC_BASE             0x00000010 // PWM_SOC_CMP Address

// ENCO PARAMETERS ADDRESSES
#define ENCOSOC_CR_ADDR         0x1
#define ENCOSOC_CURPOS_ADDR     0x2

// PWM PARAMETERS ADDRESSES
#define PWMSOC_PSCMAX_ADDR      0x1
#define PWMSOC_CNTMAX_ADDR      0x2
#define PWMSOC_DUTY_ADDR        0x3

...

fd = open("/dev/mem", O_RDWR|O_SYNC);

...

lwh2f_base = mmap(NULL, LWH2F_REG_SPAN, PROT_READ|PROT_WRITE, MAP_SHARED, fd, LWH2F_REG_BASE);

...

// Component addresses
lwh2f_enco = lwh2f_base + ENCOSOC_BASE;
lwh2f_pwm = lwh2f_base + PWMSOC_BASE;

// Enco parameters
ENCO_CR = ((uint32_t *)lwh2f_enco) + ENCOSOC_CR_ADDR;
ENCO_CurrentPosition = ((int32_t *)lwh2f_enco) + ENCOSOC_CURPOS_ADDR;

// PWM parameters
PWM_PSCMax = ((uint32_t *)lwh2f_pwm) + PWMSOC_PSCMAX_ADDR;
PWM_CNTMax = ((uint32_t *)lwh2f_pwm) + PWMSOC_CNTMAX_ADDR;
PWM_Duty = ((uint32_t *)lwh2f_pwm) + PWMSOC_DUTY_ADDR;
```

BIBLIOGRAPHY

[1] Altera/Intel: Cyclone V Hard Processor System - Technical Reference Manual; 17.07.2018

[2] Altera/Intel: Altera SoC Linux Intro Workshop; 2016.

[3] Mariano Ruiz, Antonio Carpeño: Embedded Systems - Using Quartus and Buildroot for building Embedded Linux
[4] Systems (De1-SOC) V1.9; 2017.

[5] Terasic: DE10-Nano - User Manual; 2017.

[6] Rihards Novickis, Modris Greitans: FPGA Master based on chip communications architecture for Cyclone V SoC running Linux; 2018.

[7] Roberto Fernández Molanes, Juan J. Rodríguez-Andina, José Fariña: Performance Characterization and Design Guidelines for Efficient Processor-FPGA Communication in Cyclone V FPSoCs; 2018.

[8] Intel: Avalon® Interface Specifications; 2021.

[9] Intel: Platform Designer User Guide, 2018.