Actuating, Sensing and Control
Mechatronic Systems

Hybrid Architectures (SoC)

Grzegorz Góra PhD

D1-Lab 20

ggora@agh.edu.pl

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

Department of Robotics and Mechatronics

Faculty of Mechanical Engineering and Robotics

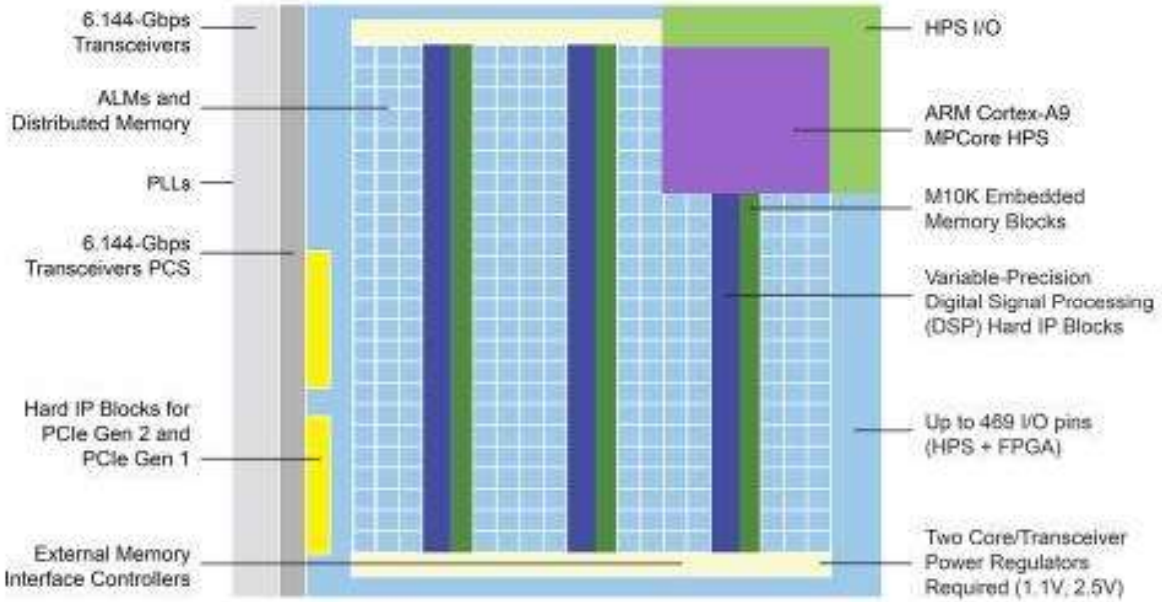
AGH University of Science and Technology

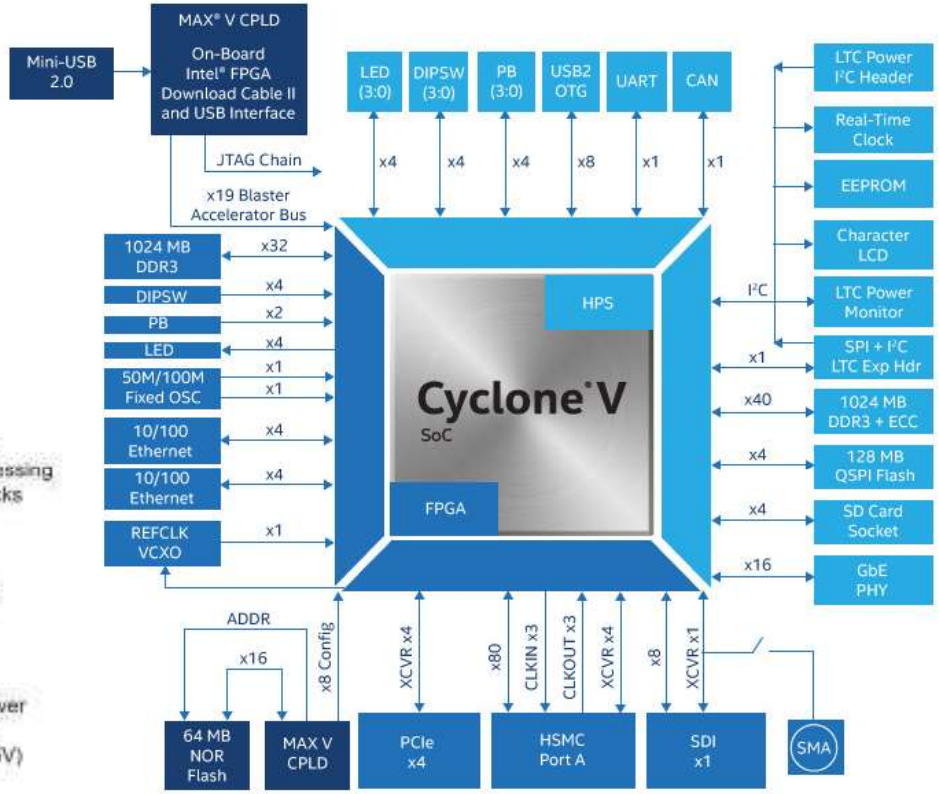
AGENDA

1. Hybrid devices SoC = FPGA + HPS (MCU)
2. What possibilities does a hybrid hardware platform offer?
3. Power Configurations
4. FPGA and HPS Communications
5. Memory mapping
6. SD Card Image
7. HSP Peripherals
8. HPS Booting Process
9. DE10-Nano
10. Cyclone V SoC – How to Get Started?
11. HPS – C/C++ Program
12. FPGA – PWM Modulator
13. FPGA – Incremental Encoder Module
14. FPGA + HPS – Data Exchange

Cyclone V SoC
FPGA + HPS

Cyclone V SoC FPGA devices offer a dual-core ARM Cortex-A9 MPCore processor surrounded by peripherals and a hardened memory controller. The FPGA fabric is connected to the hard processor system (HPS) through a high-speed internal bridges.





What possibilities does a hybrid hardware platform offer?

1. Ability to add peripheral modules to the existing HPS architecture – the resources of the FPGA part of the system can be used to implement standard peripheral modules (e.g., UART, SPI, I2C) in case there are insufficient modules in the HPS part.

2. Ability to implement custom peripheral modules – the resources of the FPGA part of the system can be used to implement dedicated (non-standard) peripheral modules (implemented by the user using hardware description languages such as VHDL/Verilog).

3. Ability to divide tasks between hardware and software –
tasks easily performed in hardware (FPGA): sensor/ADC communication, low-level communication, fixed-point number computations, signal filtering, signal generation;
tasks easily performed in software (HPS): floating-point number computations, calculations of complex mathematical models, communication through high-level interfaces (e.g., Ethernet), algorithms requiring the use of large amounts of RAM for data storage.

What possibilities does a hybrid hardware platform offer?

4. Ability to divide task stages between hardware and software – certain stages of a given task can be performed in hardware (e.g., filtering the incoming signal), while others can be performed in software (e.g., data computation and transmission via Ethernet).

5. Ability to reconfigure the FPGA part of the system "on the fly" by the HPS – it is possible to reconfigure the FPGA part of the SoC system by the HPS during operation; this can be used to change or modify the tasks performed in the FPGA part of the system.

Cyclone V SoC
Possible HPS and FPGA Power Configurations

The HPS and FPGA portions of the device have separate external power supplies and are power on independently. You can power on the HPS without powering on the FPGA side of the device. However, to power on the FPGA portion, the HPS must already be on or powered on at the same time as the FPGA portion.

HPS Power	FPGA Power
On	On
On	Off
Off	Off

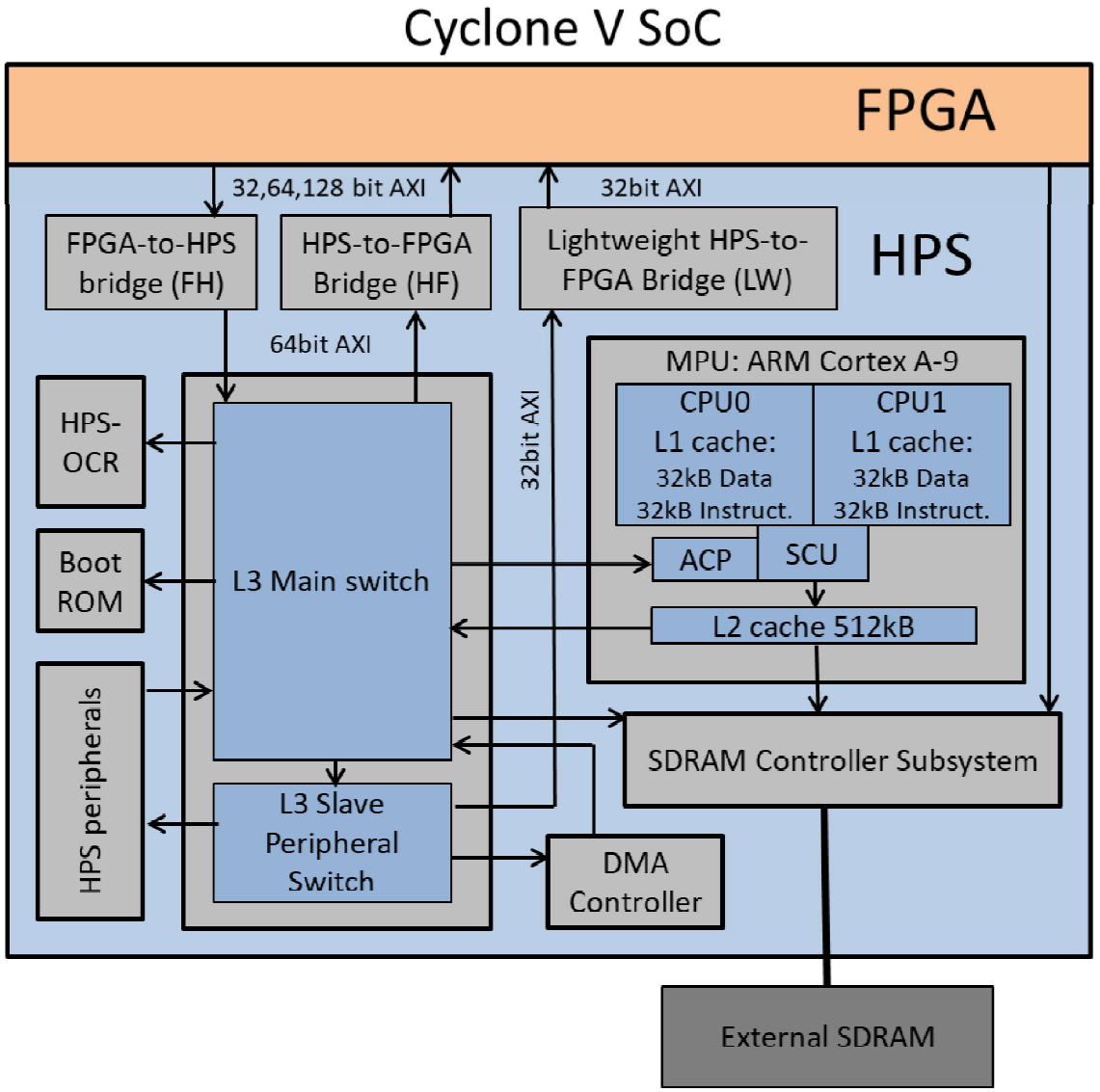
Therefore, it is possible to use the Cyclone V SoC in 4 different configurations:

- HPS & FPGA (parts operate independently of each other, perform independent tasks),
- HPS & FPGA (parts working together to accomplish tasks),
- FPGA-only (HPS supply is required),
- HPS-only.

Cyclone V SoC
Communication between HPS and FPGA

Generally communication between HPS and FPGA in Cyclone V SoC devices can be accomplished in the following ways:

1) **HPS-to-FPGA bridge**
   Unidirectional high performance interface from HPS to FPGA. Transactions are usually conducted by the processor or Direct Memory Access (DMA) controllers present in HPS. Bridge is used for accessing FPGA logic, peripherals and memory.
2) **HPS-to-FPGA Lightweight bridge**
   Bidirectional low performance interface to the FPGA fabric. Usually used by the processor to access control and status registers of the components implemented into FPGA.



Cyclone V SoC
Communication between HPS and FPGA

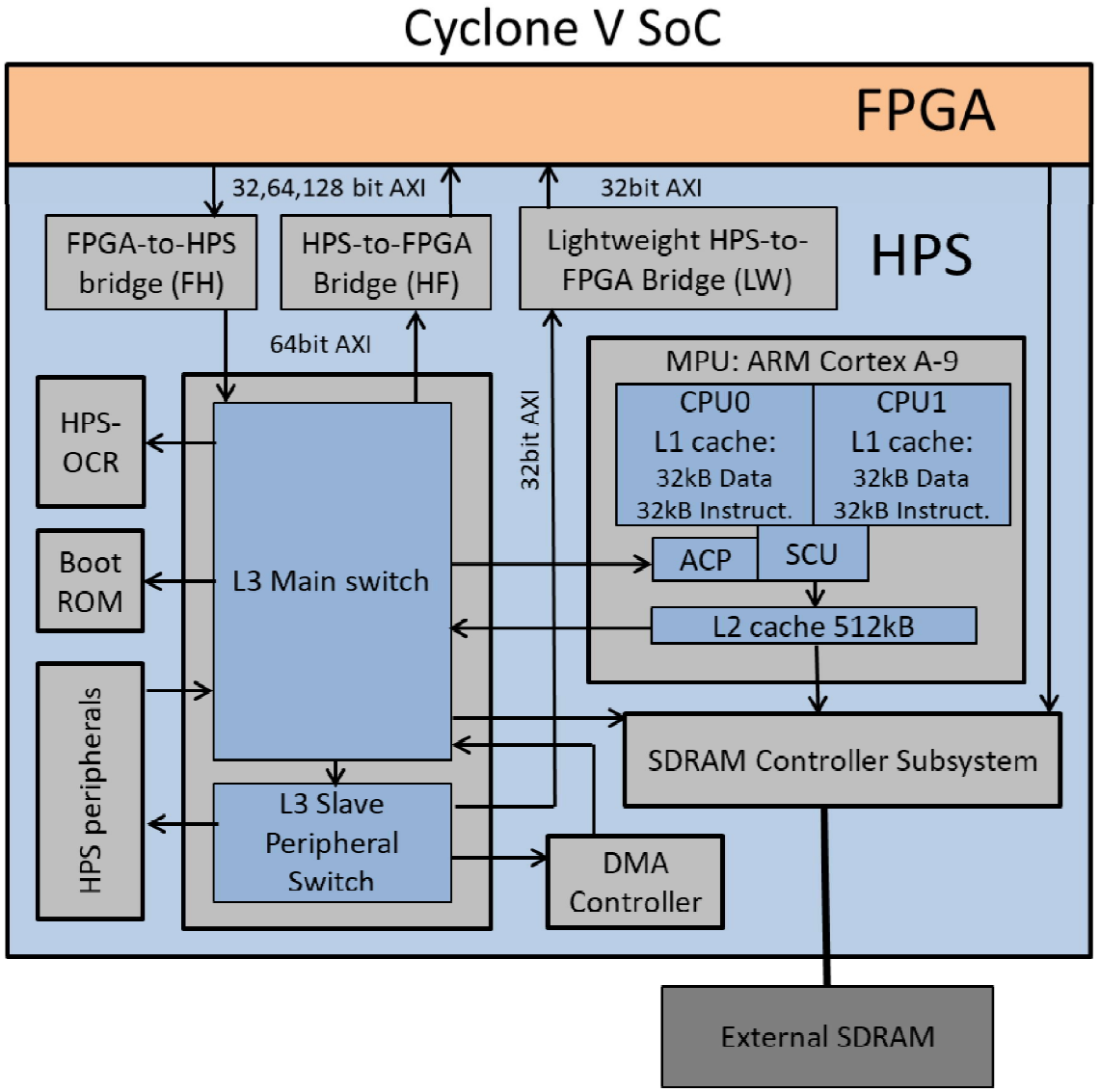
Generally communication between HPS and FPGA in Cyclone V SoC devices can be accomplished in the following ways:

3) **FPGA-to-HPS bridge**

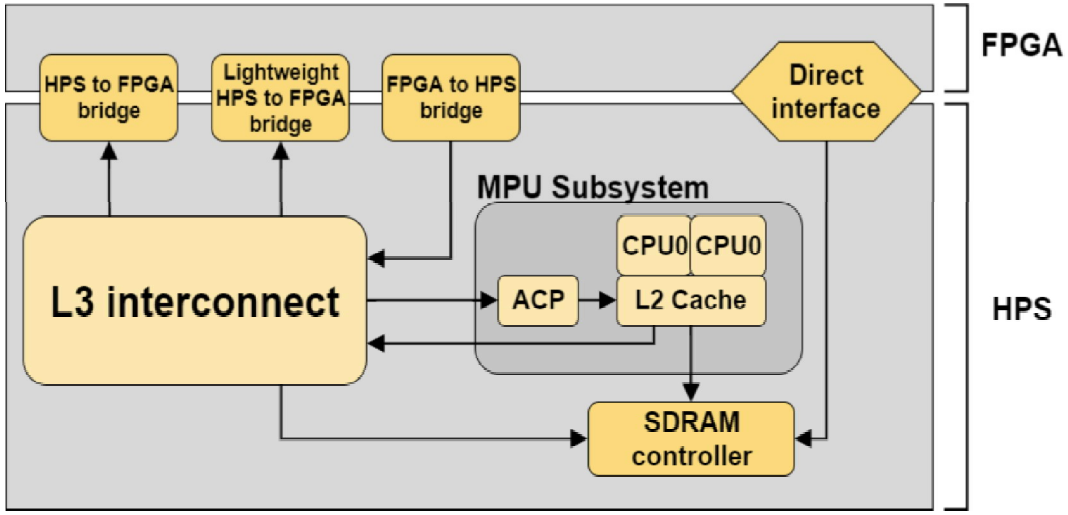
   Unidirectional high performance interface from FPGA to HPS peripherals and memory. Cached memory transactions are supported by adopting ARMs Accelerator Coherency Port (ACP).

4) **FPGA-to-HPS SDRAM interface**

   Bidirectional high performance interface from FPGA to HPS SDRAM controller. FPGA master has access to the processor's RAM. Synchronization of data read/write between the FPGA and HPS must be implemented in software.



Cyclone V SoC
FPGA to SDRAM data transfer

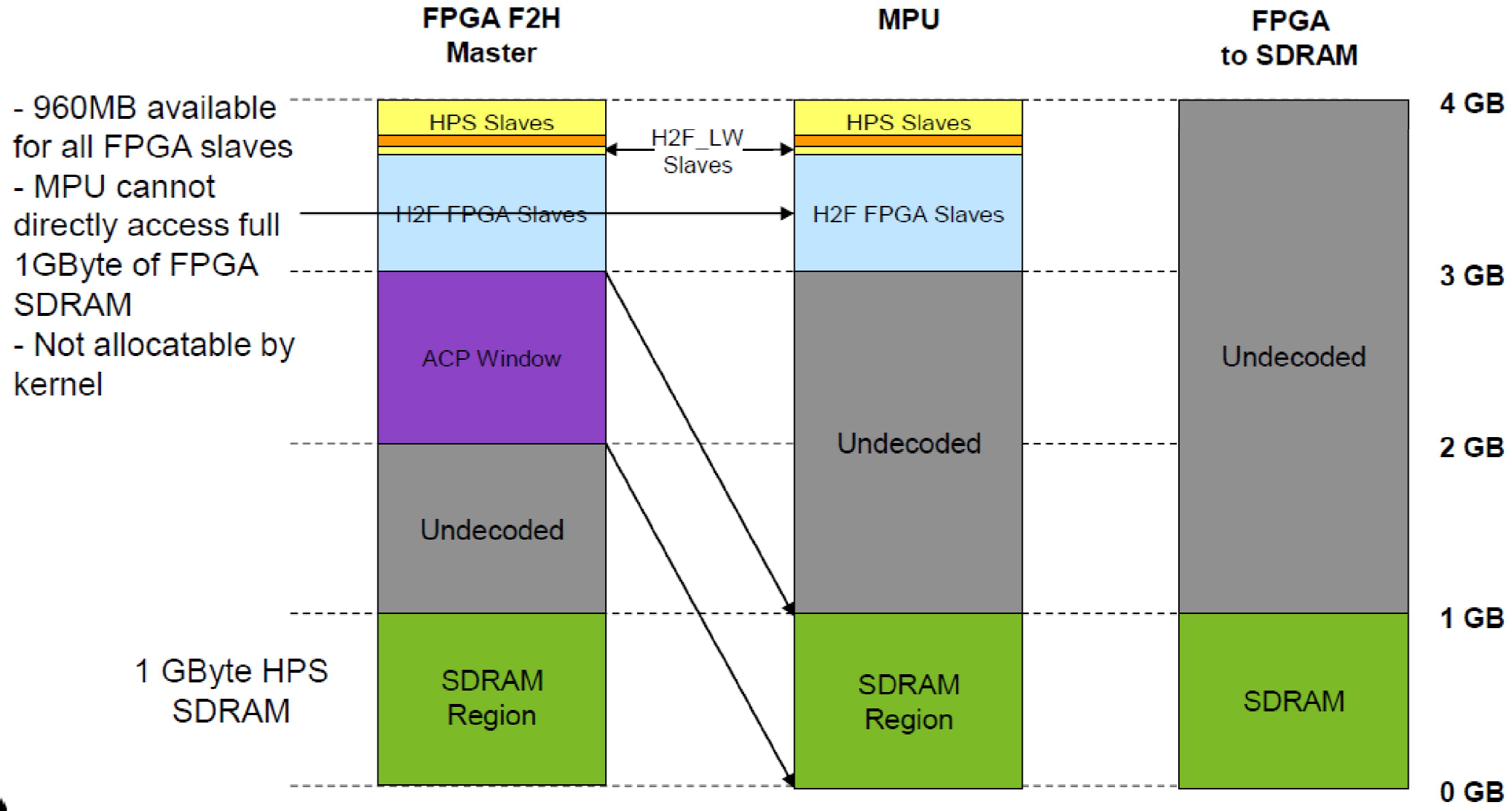


Data path	Bus width	Maximum throughput	Saturation frequency
FPGA-L3-SDRAM	32 bits	5.05 Gbps	120 MHz
	64 bits	10.10 Gbps	120 MHz
	128 bits	10.52 Gbps	65 MHz
FPGA-L3-ACP-SDRAM	32 bits	6.90 Gbps	-
	64 bits	8.64 Gbps	120 MHz
	128 bits	11.26 Gbps	90 MHz
FPGA-SDRAM	32 bits	7.52 Gbps	-
	64 bits	14.64 Gbps	-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

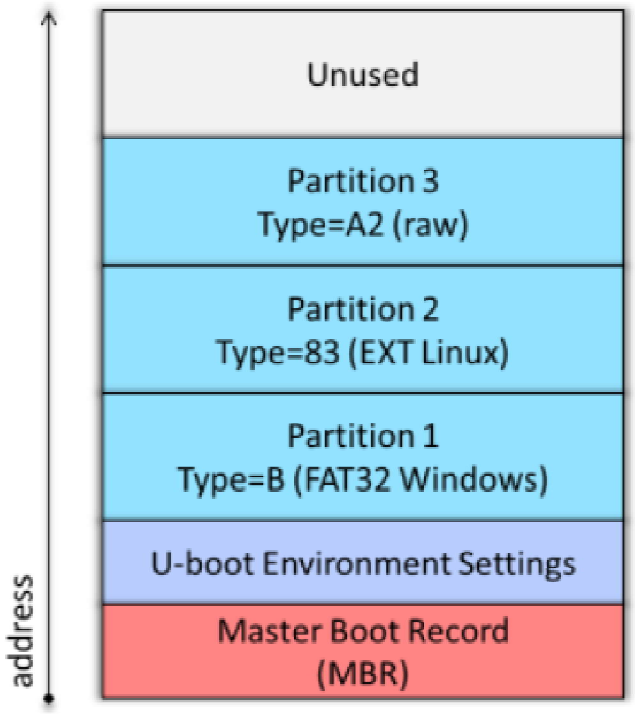
FPGA to SDRAM data transfer:

- **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
Memory Map



Cyclone V SoC
SD Card Image



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 Peripherals

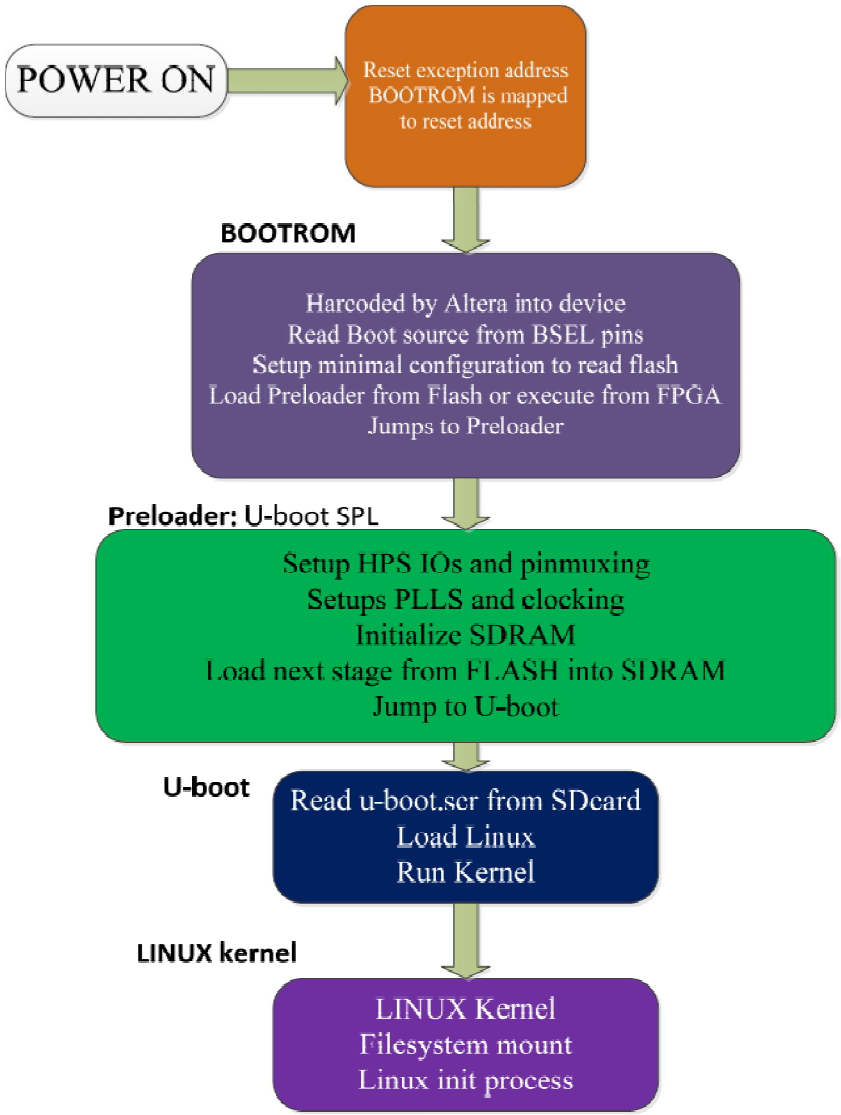
Possibilities of using HPS peripheral systems :

- HSP Peripherals (e.g.: UART, SPI) + dedicated HPS pins;
- HPS Peripherals + any FPGA pins;
- HPS pins as GPIO;
- FPGA pins controlled by HPS (LOANIO).

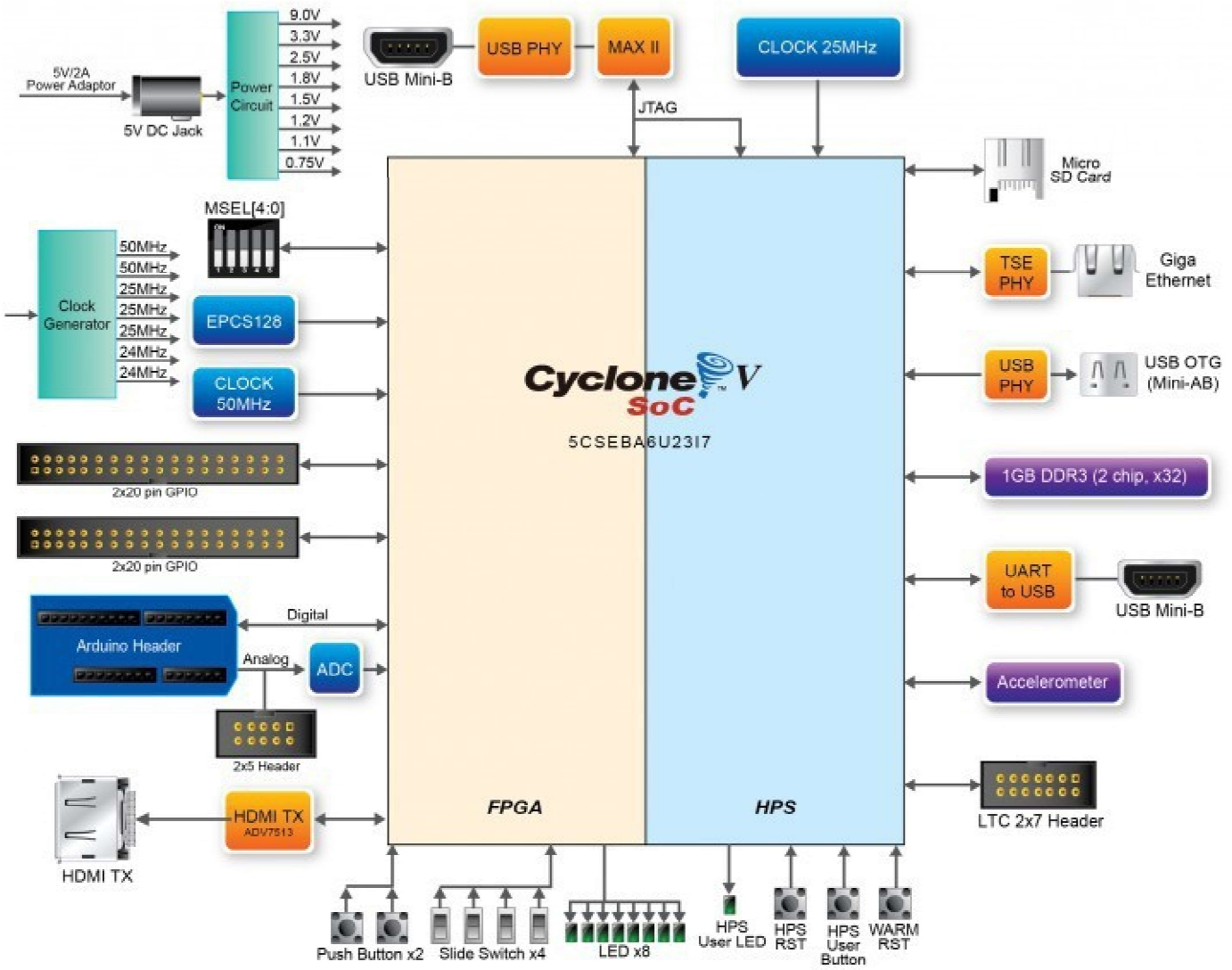
Cyclone V SoC
HSP Peripherals Mux Table

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

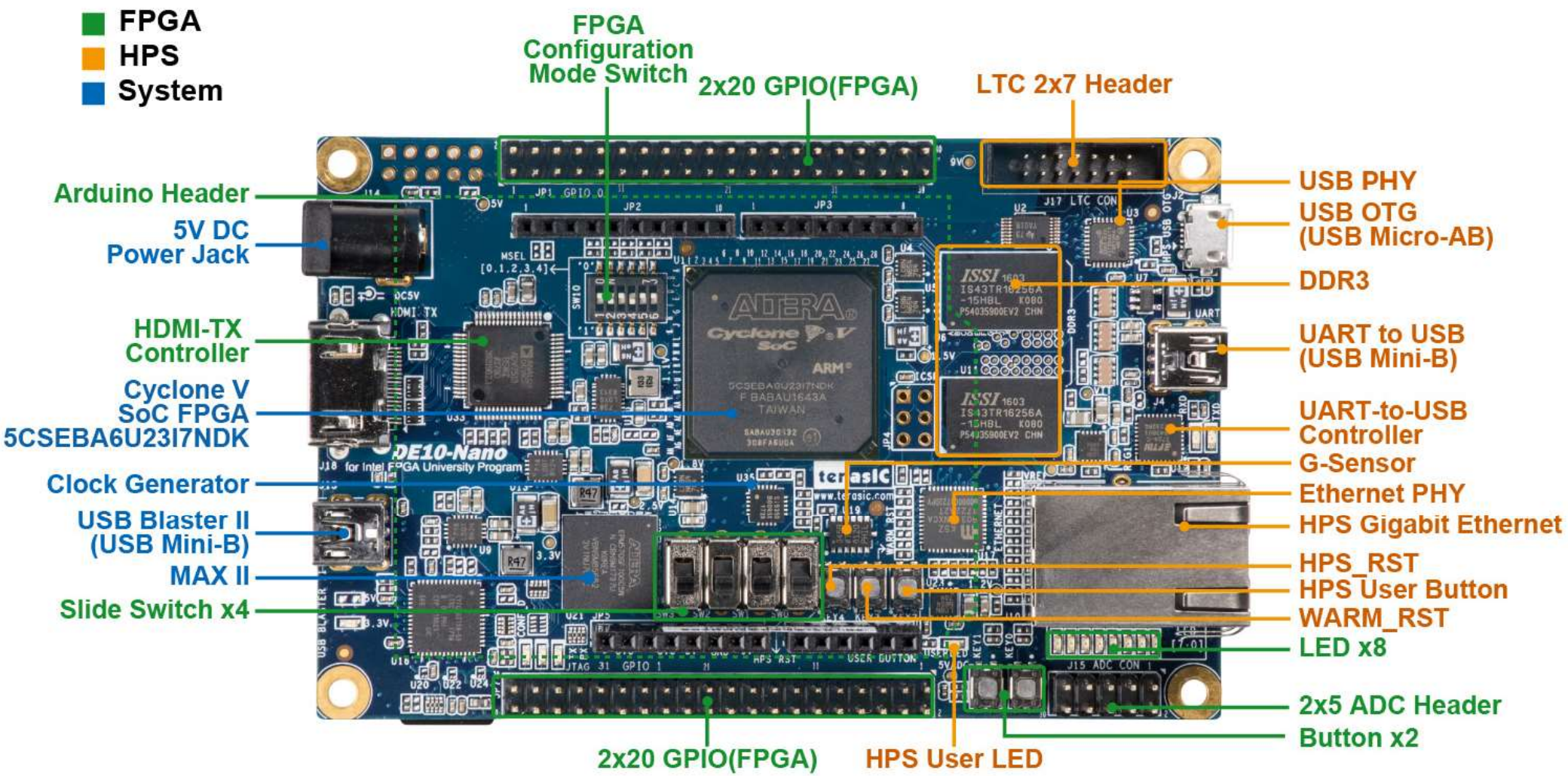
Cyclone V SoC
HPS Booting Process



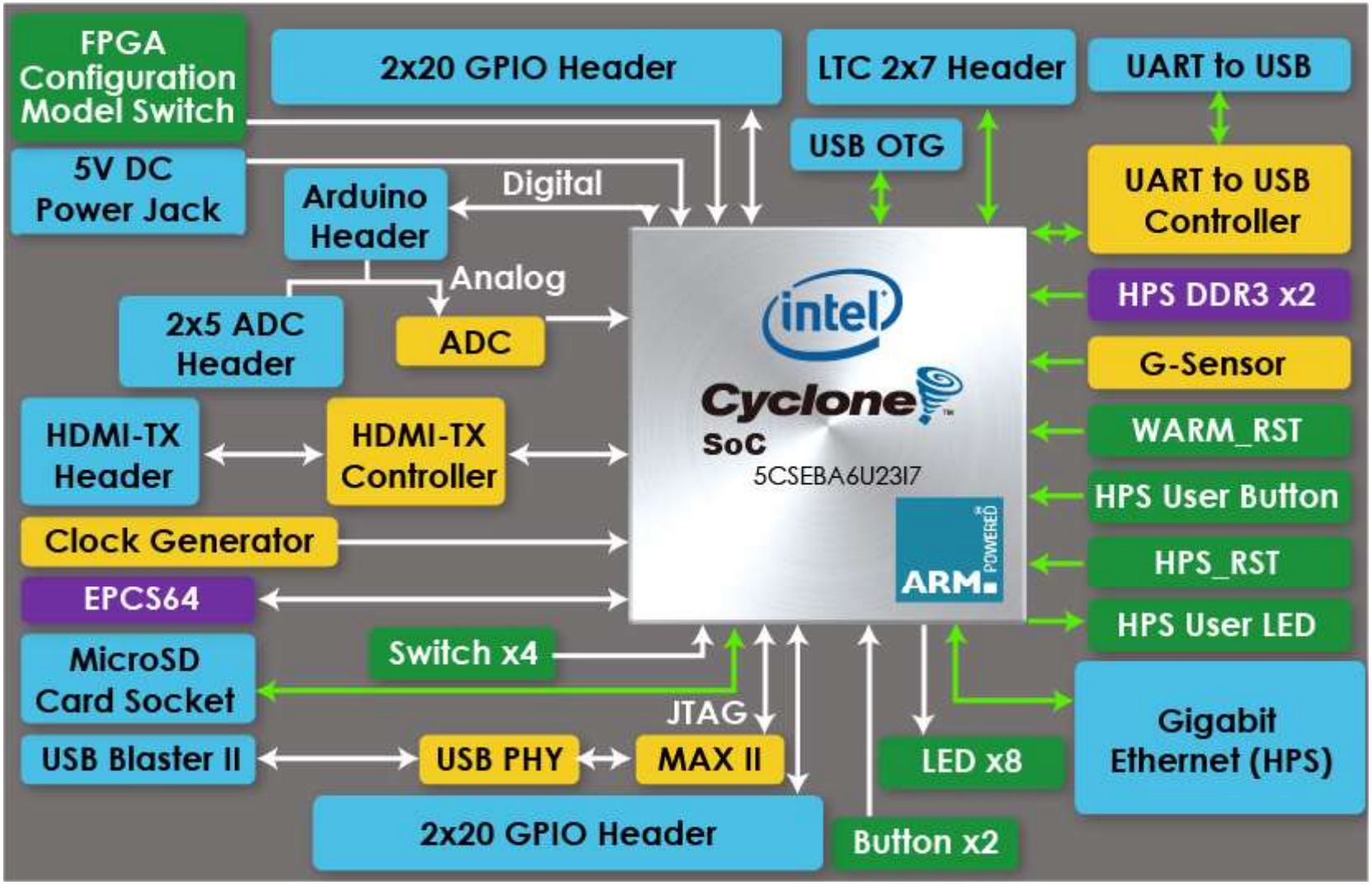
DE10-Nano


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
intel
Cyclone
SoC
5CSEBA6U23I7
HPS DDR3 x2
G-Sensor
HDMI-TX Header
HDMI-TX Controller
WARM_RST
HPS User Button
Clock Generator
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
ARM
POWERED

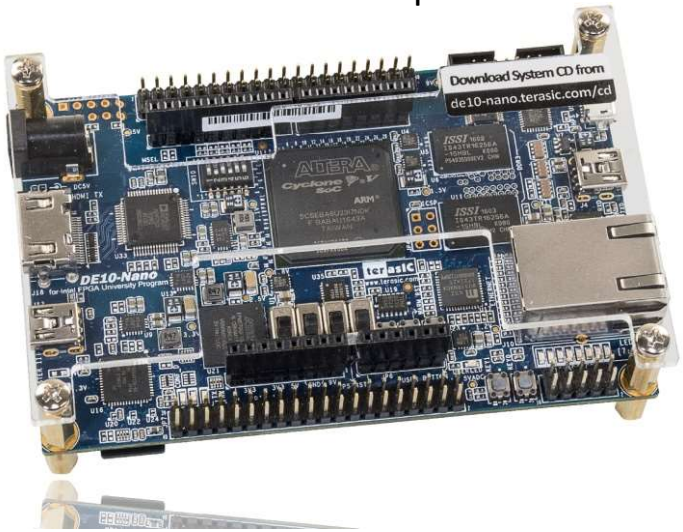
DE10 - Nano

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 – How to Get Started?

1. Prepare an SD card with a Linux image

1.1 Download Linux from https://www.intel.com/content/www/us/en/developer/topic-technology/fpga-academic/materials-sd-card-images.html

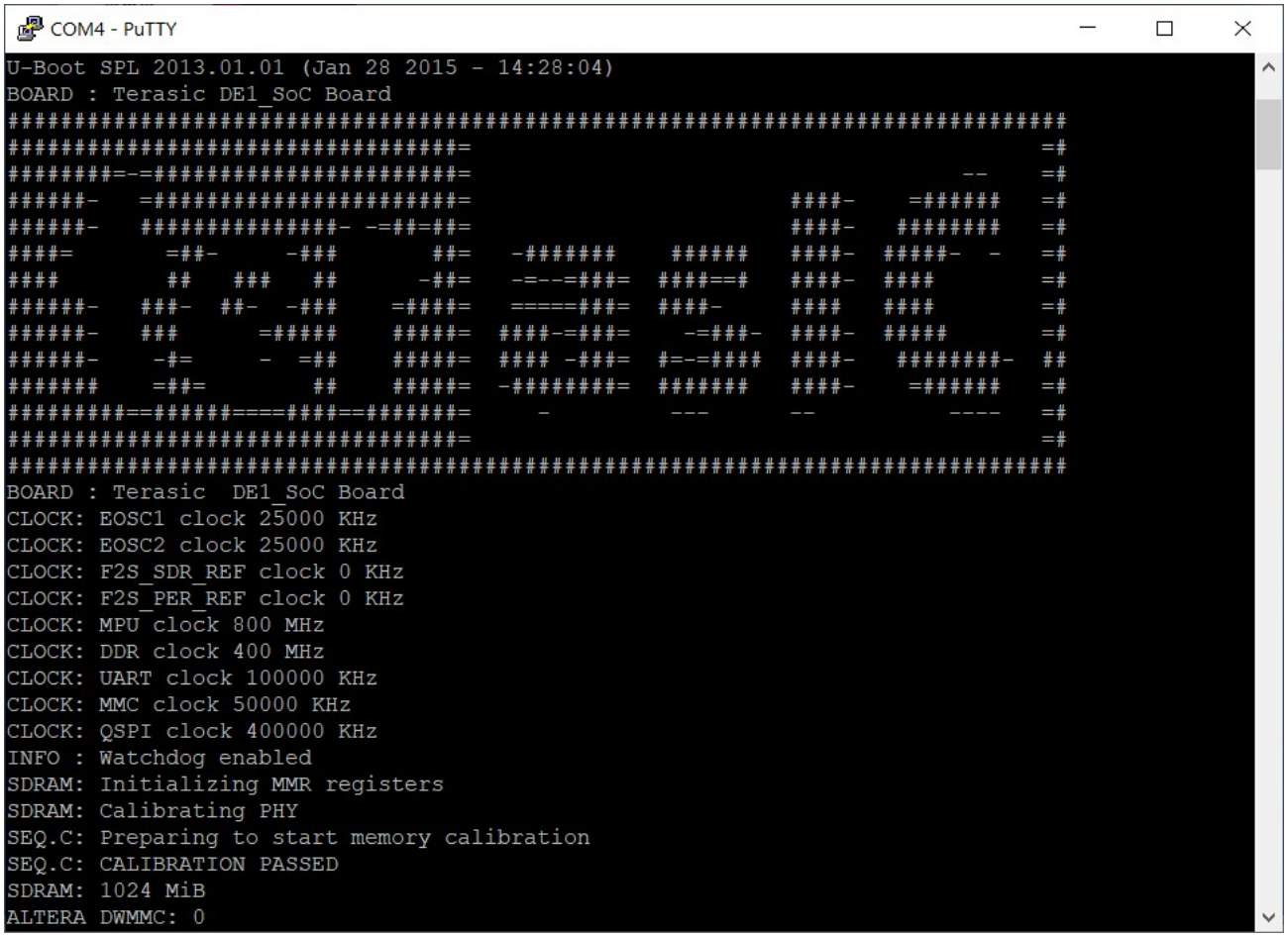
1.2a Write the image to the SD card (Windows):

1.2b Write the image to the SD card (Linux):

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

Cyclone V SoC – How to Get Started?

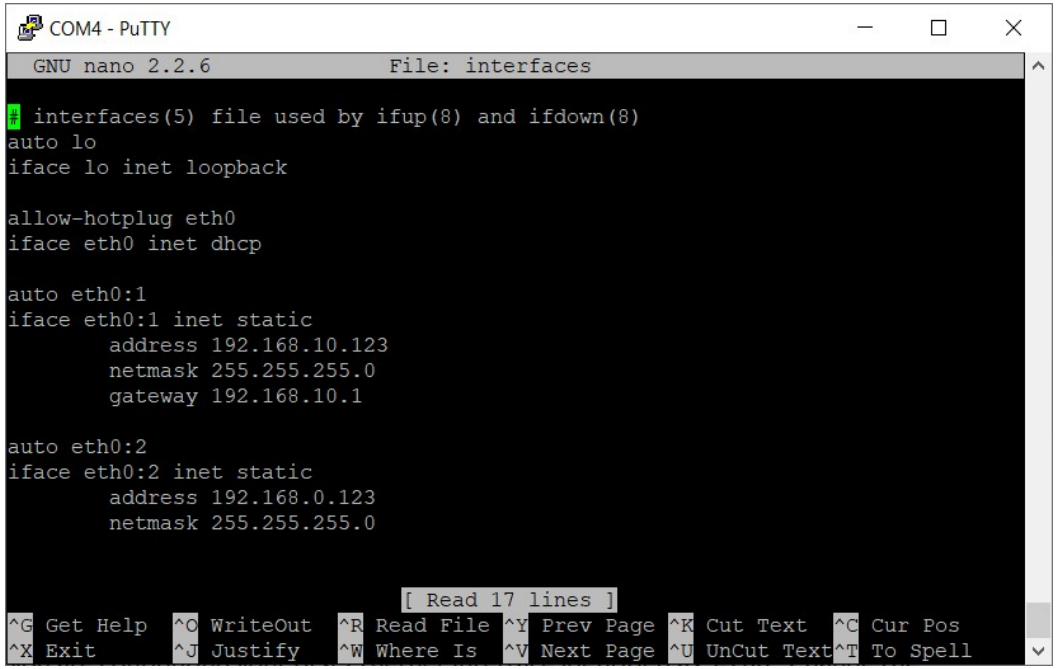
2. Use a terminal (e.g., PuTTY) to communicate with the HPS via the UART serial interface (Virtual COM port)



Cyclone V SoC – How to Get Started?

3. Set a static (or dynamic) IP address

> sudo nano /etc/network/interfaces



4. Set a password in the terminal

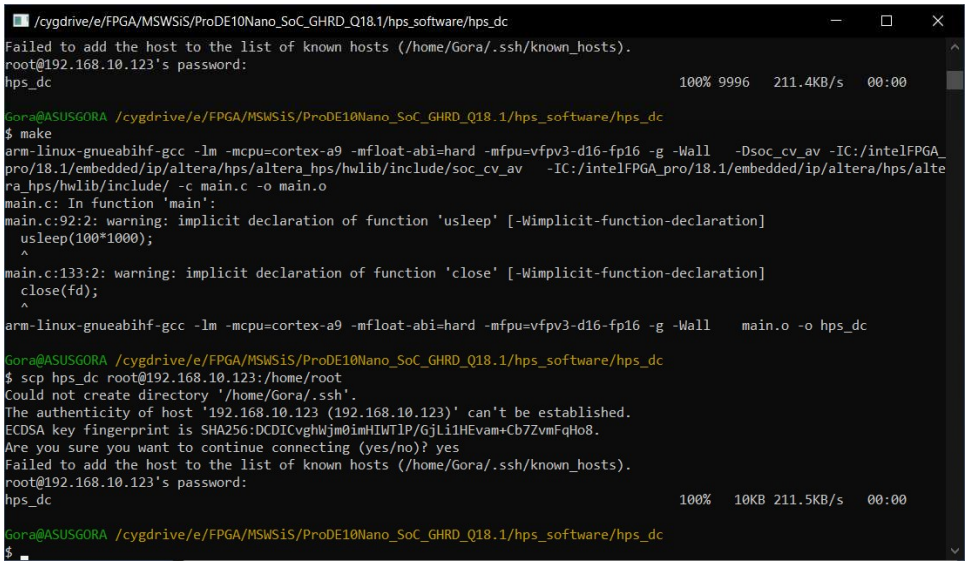
> passwd

5. Connect to the HPS via SSH using the Ethernet interface

HPS Part – A program in C/C++

(the HPS only or the HPS independently of the FPGA)

1. Launch SoC EDS Command Shell as a terminal on the PC



2. Write and compile the program using Makefile

> make

3. Transfer the program to the HPS using SCP

> scp *PROG_NAME* USER_NAME@IP_ADDRESS:/PATH_TO_COPY

eg.:

> scp hps_soft root@192.168.10.123:/home/root

HPS Part – Peripheral Systems

(the HPS only or the HPS independently of the FPGA)

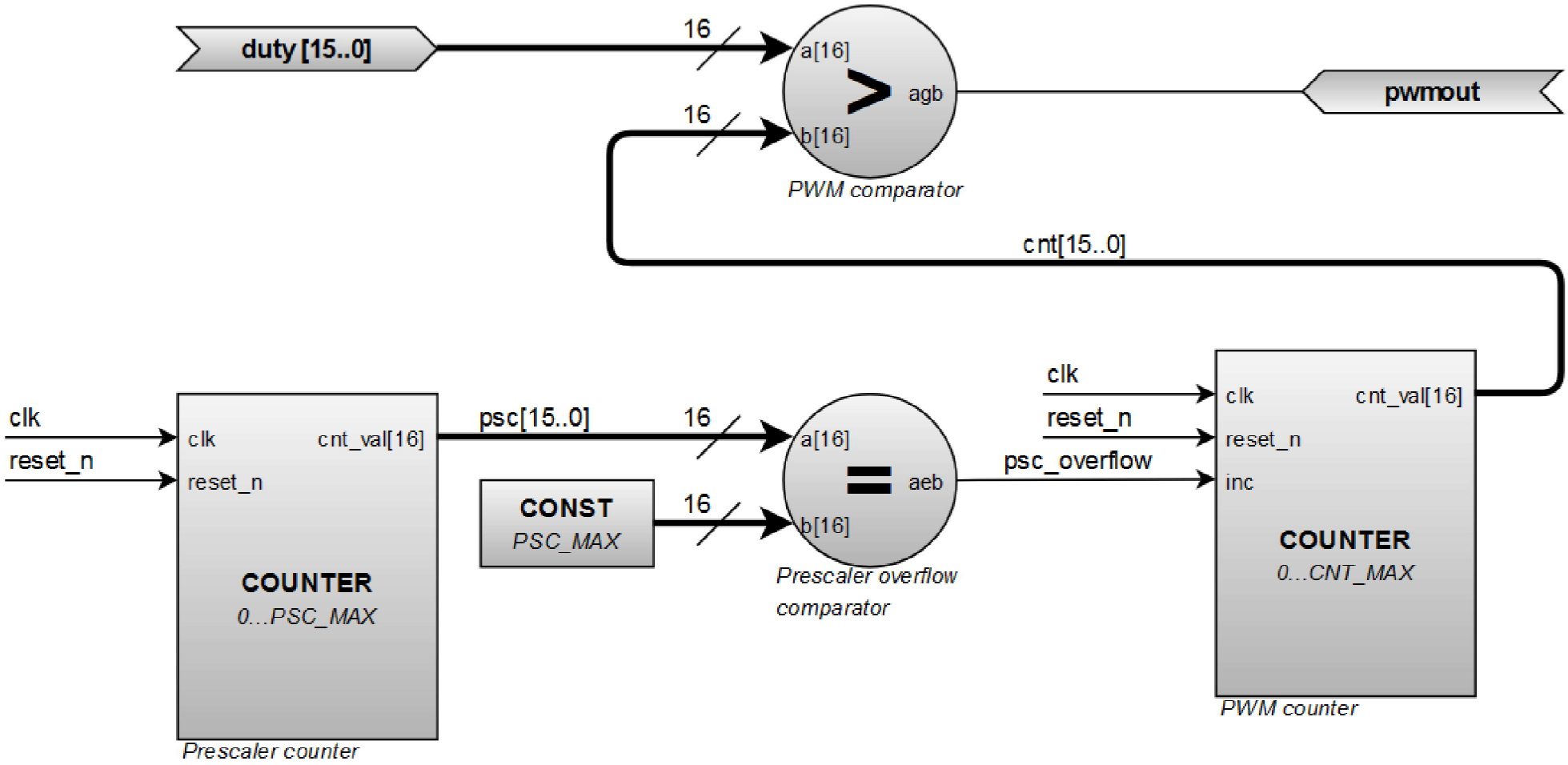
The HPS section can be treated as an independent microprocessor system (microcontroller). It can operate under the control of an operating system (e.g., Linux) in either console or graphical (GUI) mode. The peripheral systems present in the HPS section. Peripheral systems present in the HPS section:

- 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.

FPGA Part – Hardware module: PWM Modulator

(FPGA only or FPGA independently of HPS)

PWM Modulator Hardware Architecture:



FPGA Part – Hardware module declaration

(FPGA only or FPGA independently of HPS)

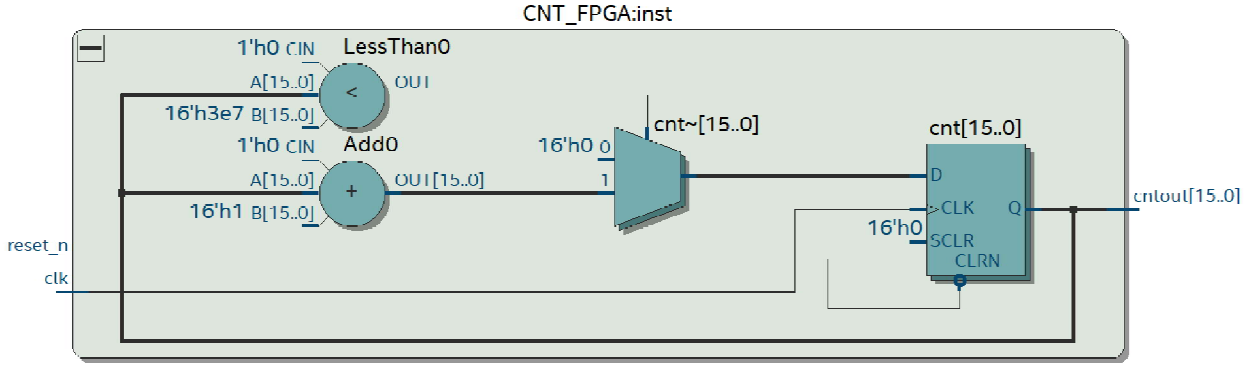
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4
5   entity FullAdderVhdl is
6      port(
7            A, B, Cin : in std_logic;
8            Sum, Cout : out std_logic
9      );
10  end entity;
11
12
13  architecture FullAdderVhdl of FullAdderVhdl is
14
15     -- Signals declaration
16     signal s1, s2, s3 : std_logic := '0';
17
18  begin
19
20     s1 <= A xor B;
21     s2 <= A and B;
22     s3 <= s1 and Cin;
23     Sum <= Cin xor s1;
24     Cout <= s2 or s3;
25
26     -- Example of AND4
27     -- s100 <= s101 and s102 and s103 and s104;
28
29     -- Example of OR4
30     -- s100 <= s101 or s102 or s103 or s104;
31
32  end architecture;
33
```

FPGA Part – „process" i „when" in VHDL language

(FPGA only or FPGA independently of HPS)

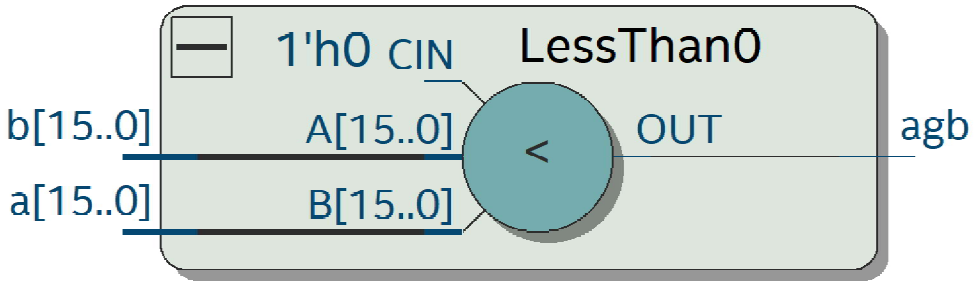
„process" construction used to implement the counter (sequential circuit):

```
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;
```



„when" construction used to implement comparator (combinational circuit):

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



FPGA Part – Hardware Module: Incremental Encoder

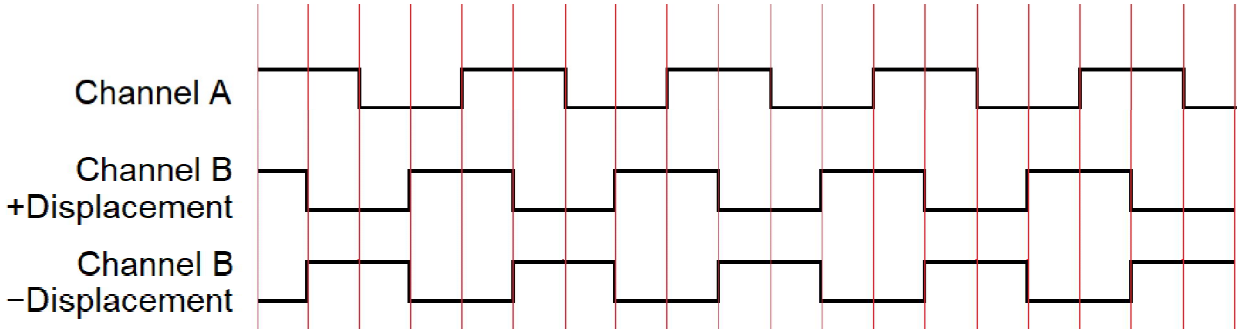
(FPGA only or FPGA independently of HPS)

Incremental Encoder – enables the determination of relative position (with respect to the initial position after power-up). It is most commonly used with a quadrature interface, either 2-channel or 3-channel.

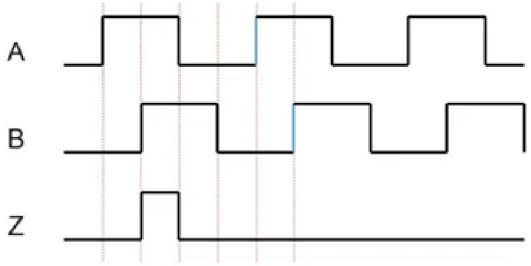
Possible output types:

- **Push-Pull** – a digital signal in the (most commonly) TTL standard (5V);
- **Open Collector** – requires an external Pull-up resistor;
- **Differential** – requires an external voltage standard converter from RS422 to TTL/LVTTL.

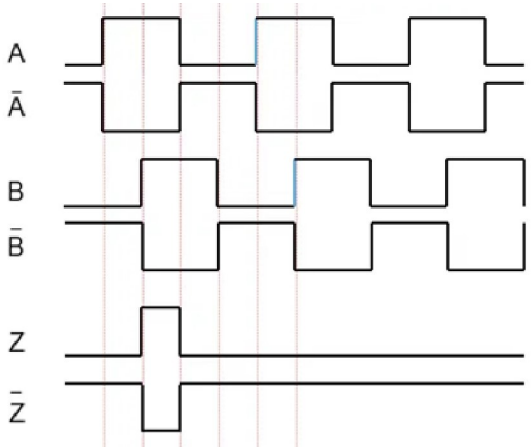
Encoder with 2-channel AB interface:



Encoder with 3-channel ABZ interface (Output types: PP/OC):



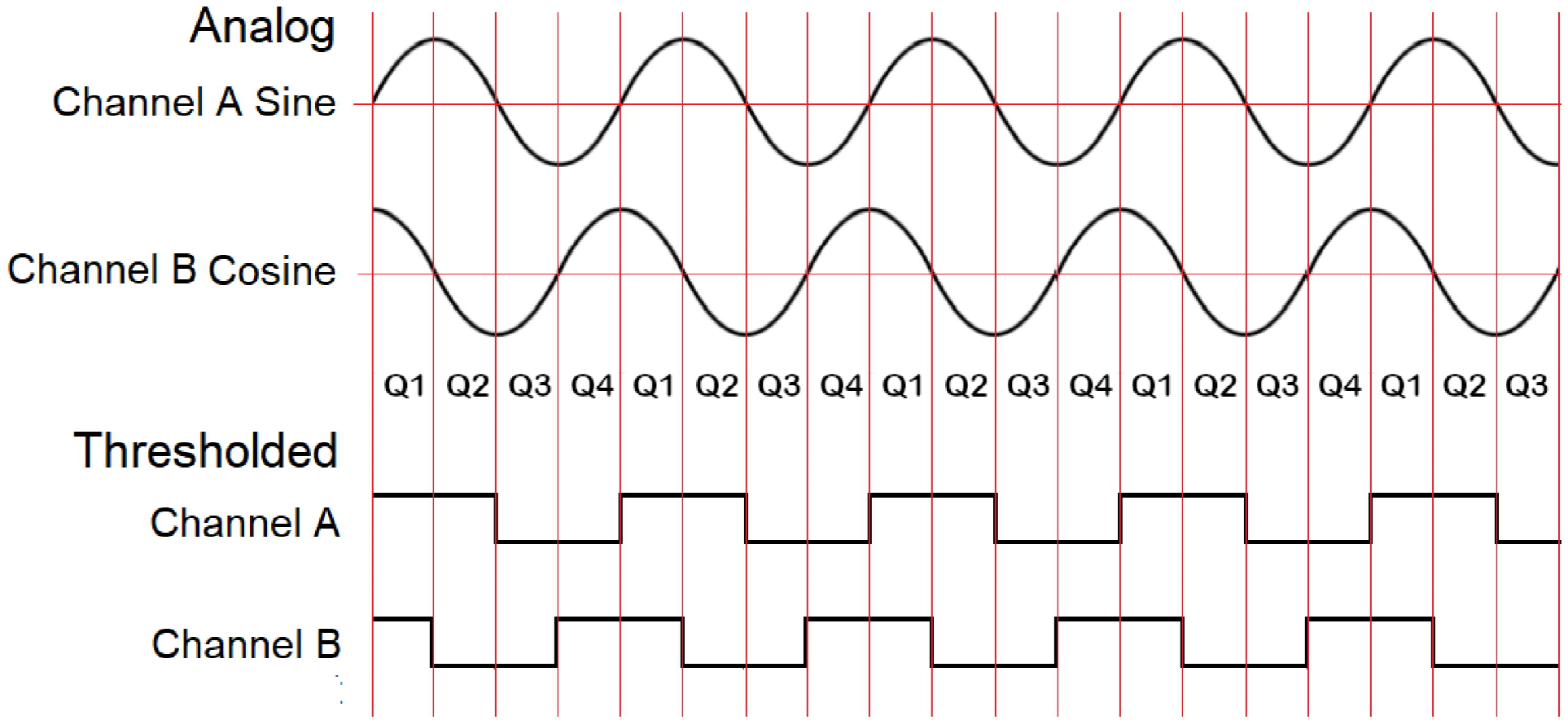
Encoder with 3-channel ABZ interface (Differential output):



FPGA Part – Hardware Module: Incremental Encoder

(FPGA only or FPGA independently of HPS)

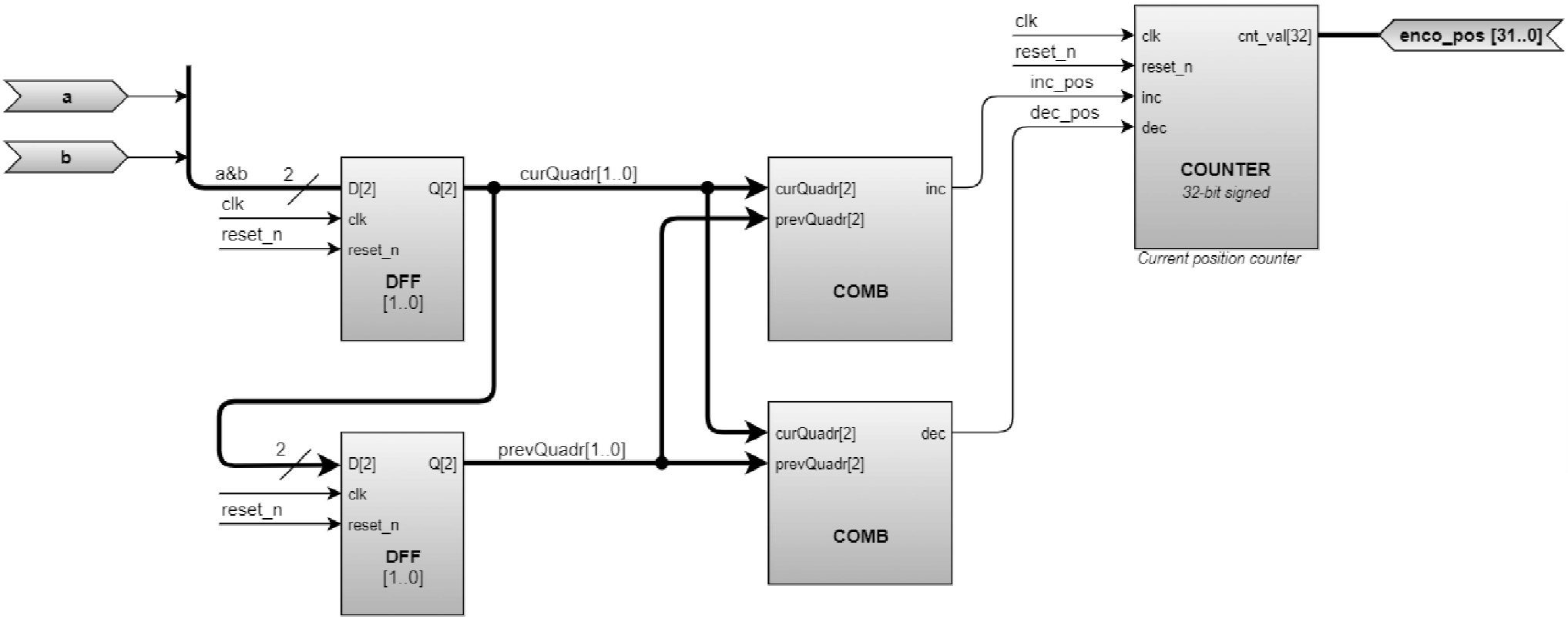
Encoder with 2-channel AB interface:



FPGA Part – Hardware Module: Incremental Encoder

(FPGA only or FPGA independently of HPS)

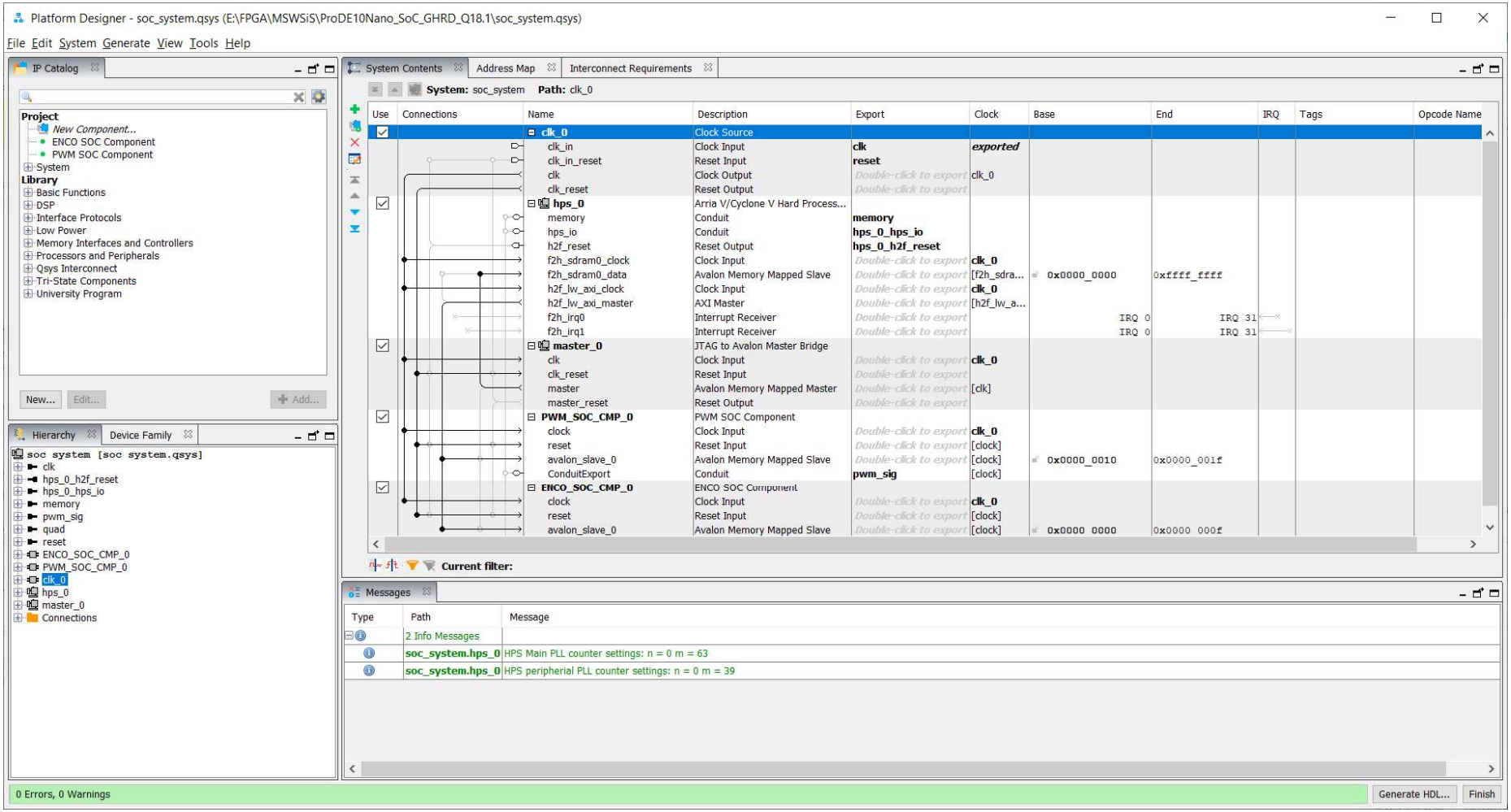
Hardware Architecture of the Incremental Encoder Module:



FPGA + HPS – Platform Designer

(Utilization of FPGA and HPS Parts for performing a common task)

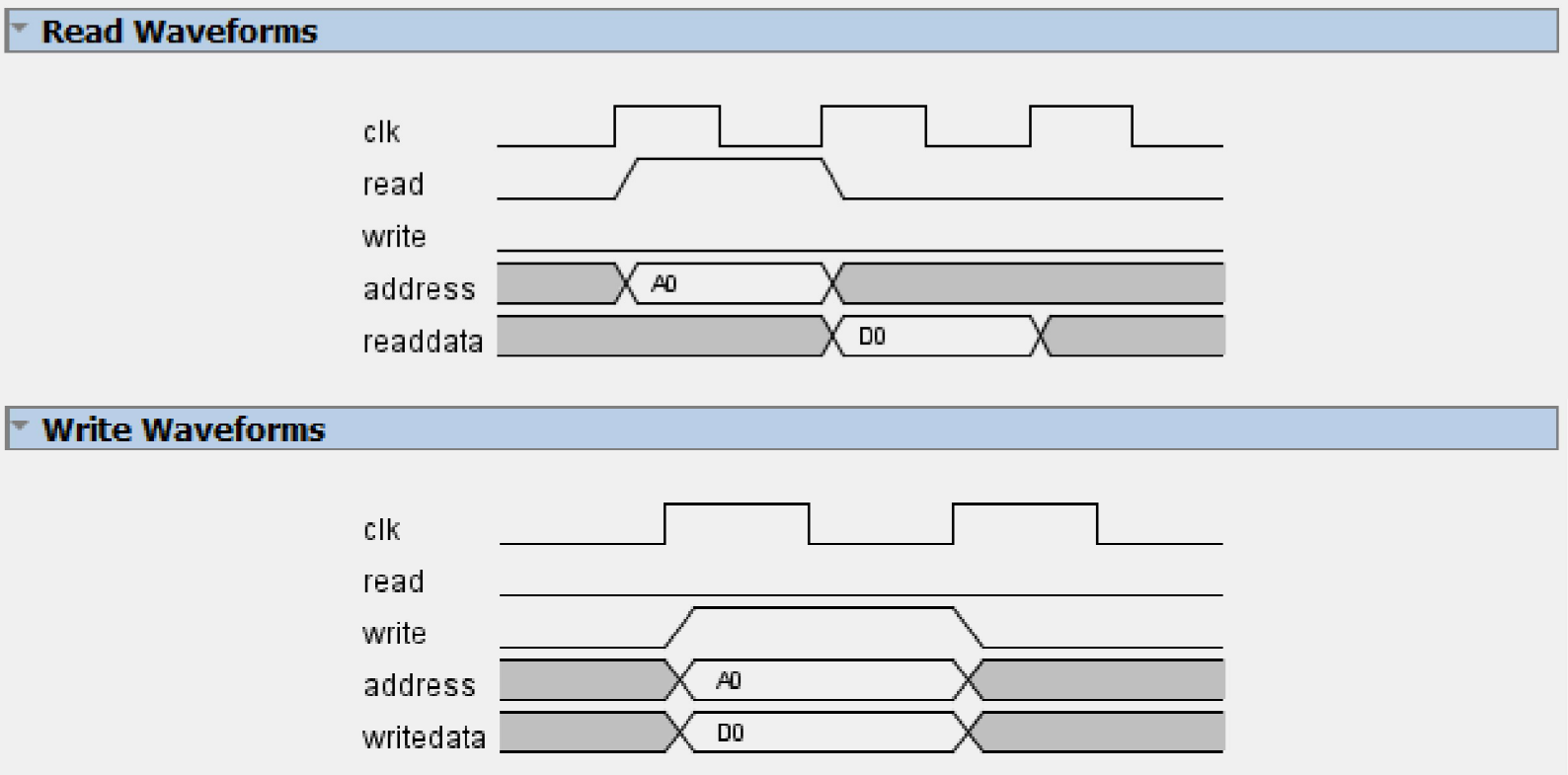
Platform Designer – a tool for configuring the architecture of the HPS part of the system.



FPGA + HPS – Avalon Memory Mapped Slave Interfaces

(Utilization of FPGA and HPS Parts for performing a common task)

Avalon-MMS – an interface that enables communication between the microprocessor core and peripheral devices (e.g., UART, I2C, SPI). From the microprocessor side, communication involves writing/reading data from a specific memory space (address space). From the hardware side (FPGA), communication involves responding to write/read requests for a parameter at a defined address using the Avalon bus.



FPGA + HPS – Communication via the Avalon bus (VHDL)

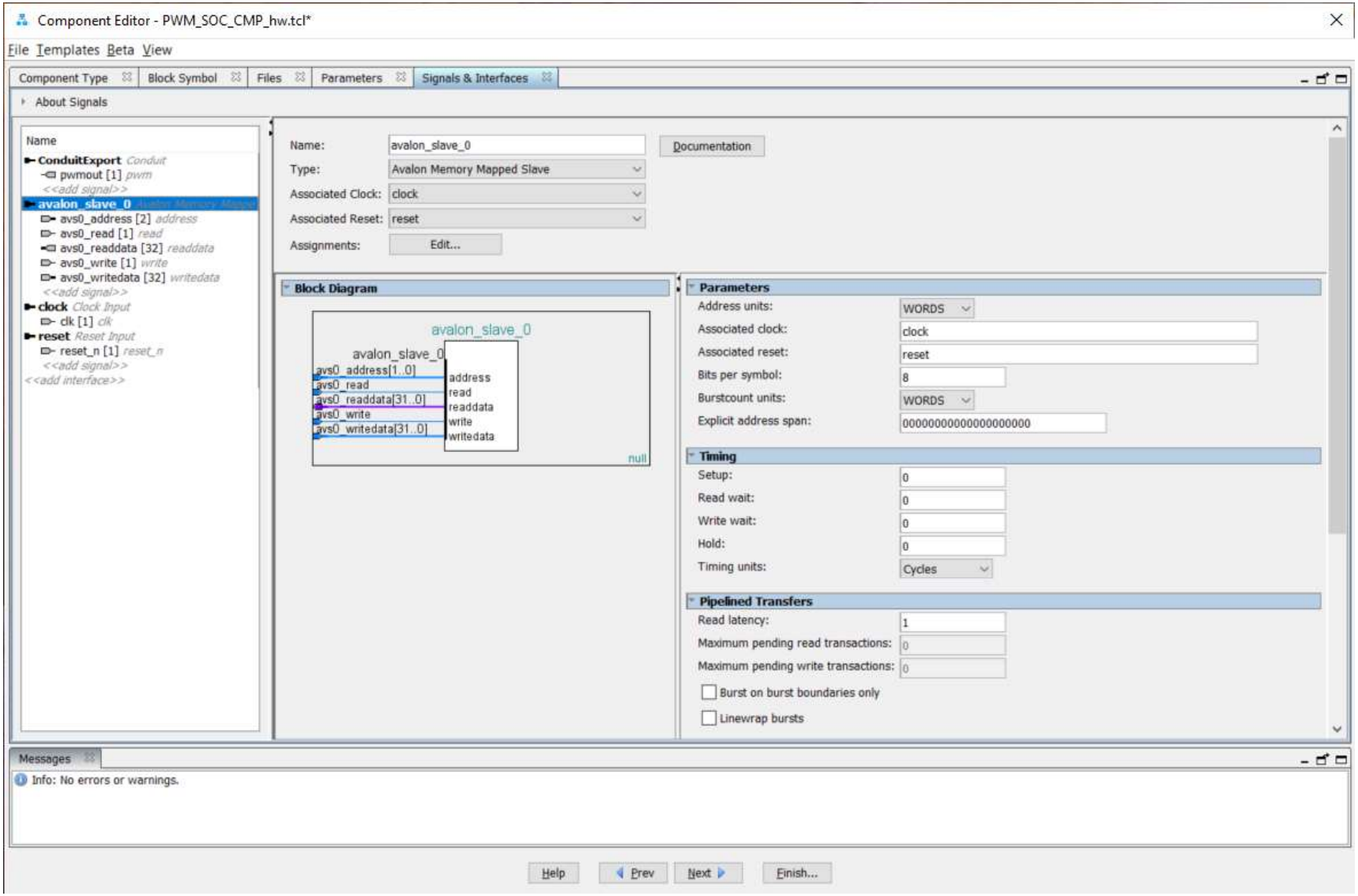
(Utilization of FPGA and HPS Parts for performing a common task)

```
-- ---------------------------------------------------------
-- 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 – Hardware module with an Avalon-MMS interface

(Utilization of FPGA and HPS Parts for performing a common task)



FPGA + HPS – Communication between the MCU system and the periph. sys. (C/C++)

(Utilization of FPGA and HPS Parts for performing a common task)

```
// 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.