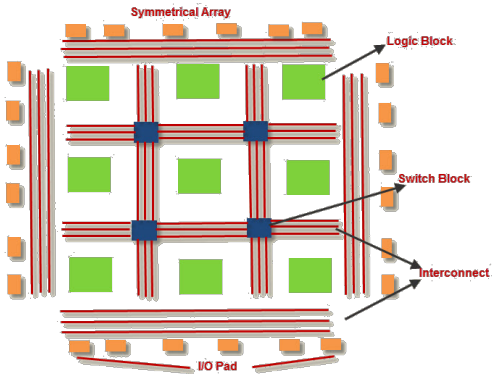
Lecture 2: MPSoC intro, ZYNQ chip, basic design, and debugging

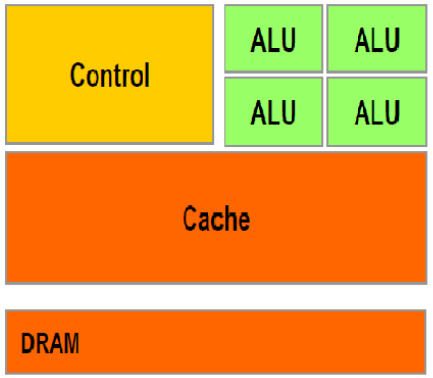
(UART with PWM)

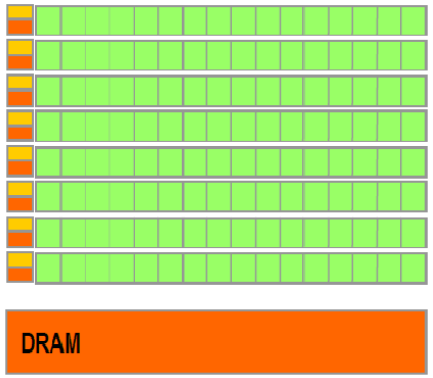
Emad Samuel Malki Ebeid,

University of Southern Denmark

Is it FPGA, CPU, or GPU to us in autonomous robots?







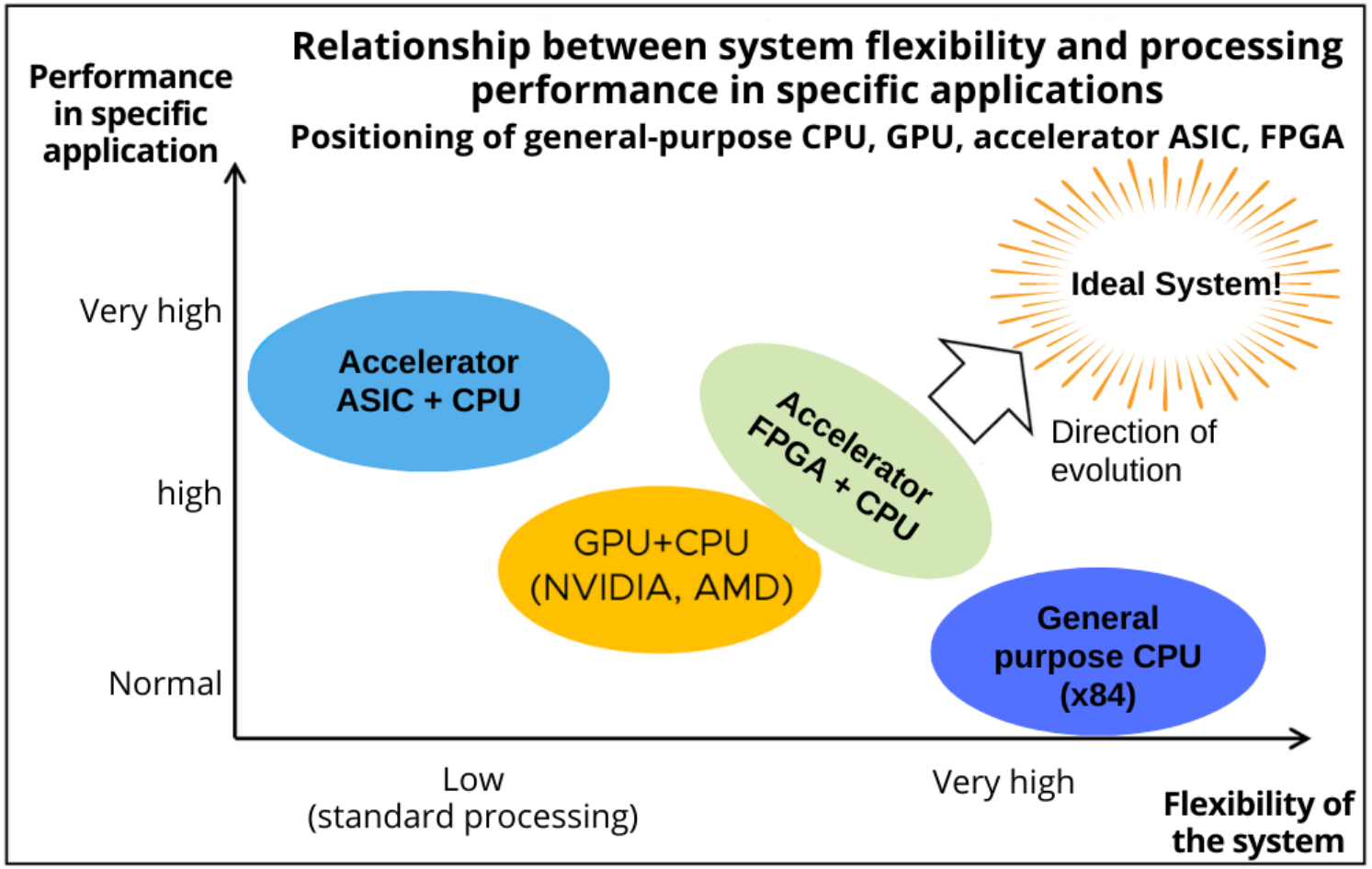
Introduction

- Autonomous operations of drones have significantly increased in the last few years, spanning from autonomous infrastructure inspections to interactions and manipulation of hard-to-reach objects.
- Robots and Drones integrate several technologies and algorithms for accomplishing their tasks (e.g., path planning and scene reconstruction, 3D sensing, localization, exploration and navigation). That poses many challenges due to the enormous amount of data that needs to be processed onboard of the drone in real time. Commonly, CPUs and GPUs are used to develop novel algorithms and handle a wide range of tasks quickly.

Real-time system requirements?

- Algorithms are typically captured in C/C++ or some other high-level language, which abstracts the details of the computing platform.
- Improving the runtime of software was based on two central concepts:
  - increasing processor clock frequency and
  - using specialized processors
- Incremental speedup through processor clock frequency is not enough to deliver a viable product to market.
- Speedup by adding more processing cores per chip.
- Multicore processors to boost software performance.

Performance



https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf
https://hacarus.com/tech/20201207-amd-acquires-xilinx/

Performance

- **Clock Frequency:** a high clock frequency translates into a higher performance execution rate of an algorithm**?**

- Instruction fetch (IF)
- Instruction decode (ID)
- Execute (EXE)
- Memory operations (MEM)
- Write back (WB)

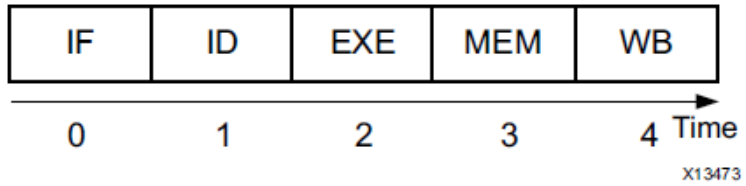


Figure 3-1: **Processor Instruction Execution Stages**

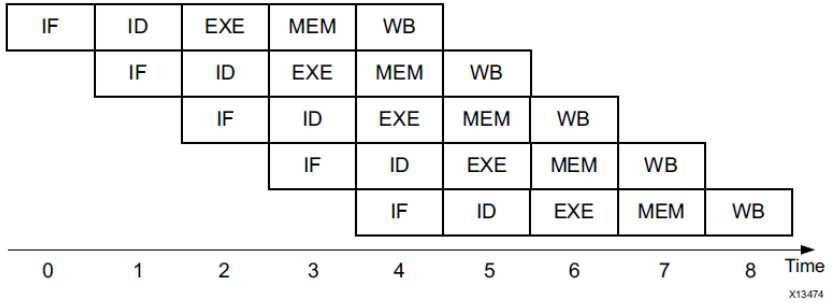


Figure 3-2: **Processor with Multiple Instruction Execution Units**

Performance

- FPGA executes a single program at a time on a custom circuit for that program.
- Therefore, changing the user application changes the circuit in the FPGA.

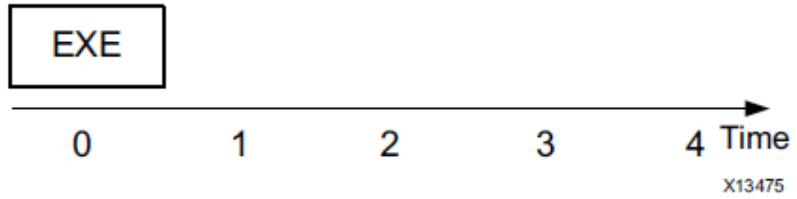


Figure 3-3: **FPGA Instruction Execution Stages**

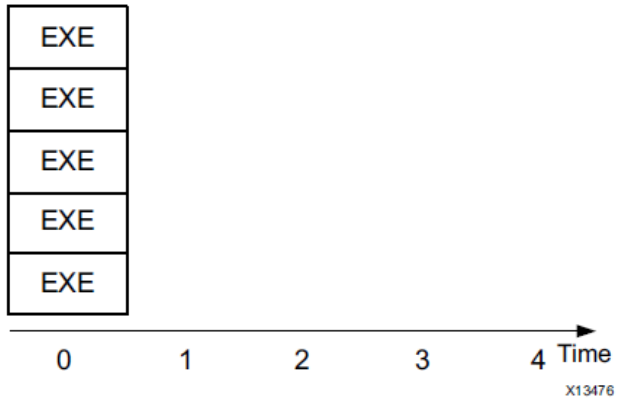


Figure 3-4: **FPGA with Multiple Instruction Execution Units**

- the FPGA has a nominal performance advantage of 9x compared to the processor

Performance matrics

- **Power consumption**
  - $P = 0.5 * C * V^2 * f$
    - Where P is power, C is capacitance, V is the voltage across the gate, f is the clock frequency.

- By creating a custom circuit per software program, an FPGA is able to run at a lower clock frequency with maximum parallelism between operations and without the instruction interpretation overhead found in a processor.

Latency

- **Latency** is the number of clock cycles it takes to complete an instruction or set of instructions to generate an application result value
  - five clock cycles/instraction.

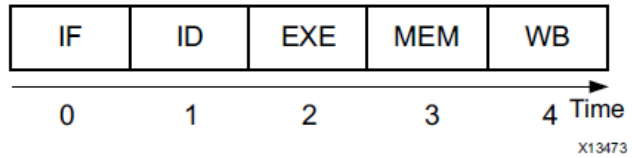


Figure 3-1: **Processor Instruction Execution Stages**

- Pipelining to lower the latency: the next instruction can be launched into execution before the current instruction is complete

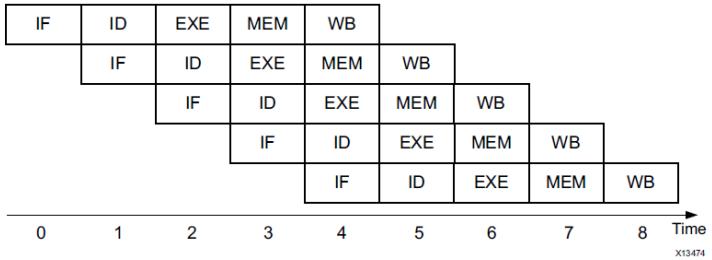


Figure 3-2: **Processor with Multiple Instruction Execution Units**

Throughput

- **Throughput** is the number of clock cycles it takes for the processing logic to accept the next input data sample.
- Figure 3-5 requires 10 ns between input samples,
- Figure 3-6 only requires 2 ns between input data samples.
- Figure 3-6 circuit implementation has higher performance, because it can accept a higher input data rate.

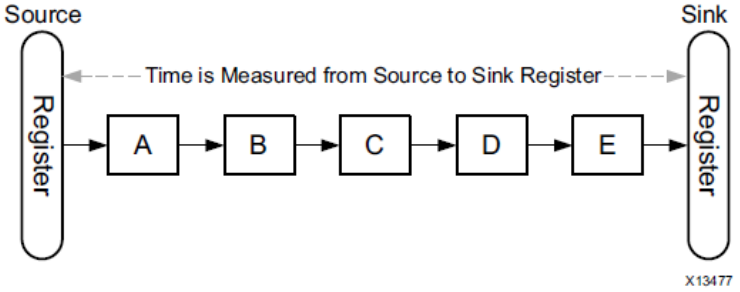


Figure 3-5: **FPGA Implementation without Pipelining**

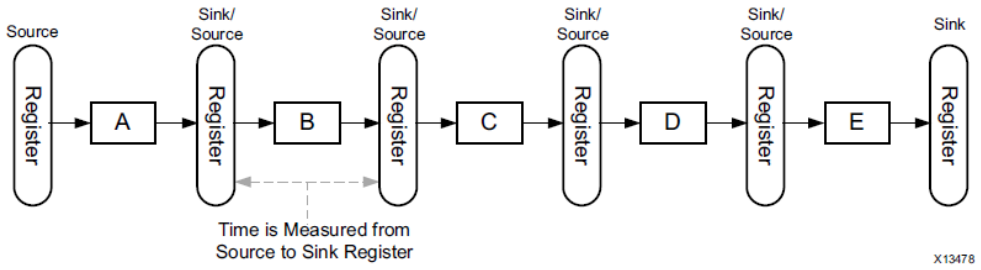


Figure 3-6: **FPGA Implementation with Pipelining**

FPGA

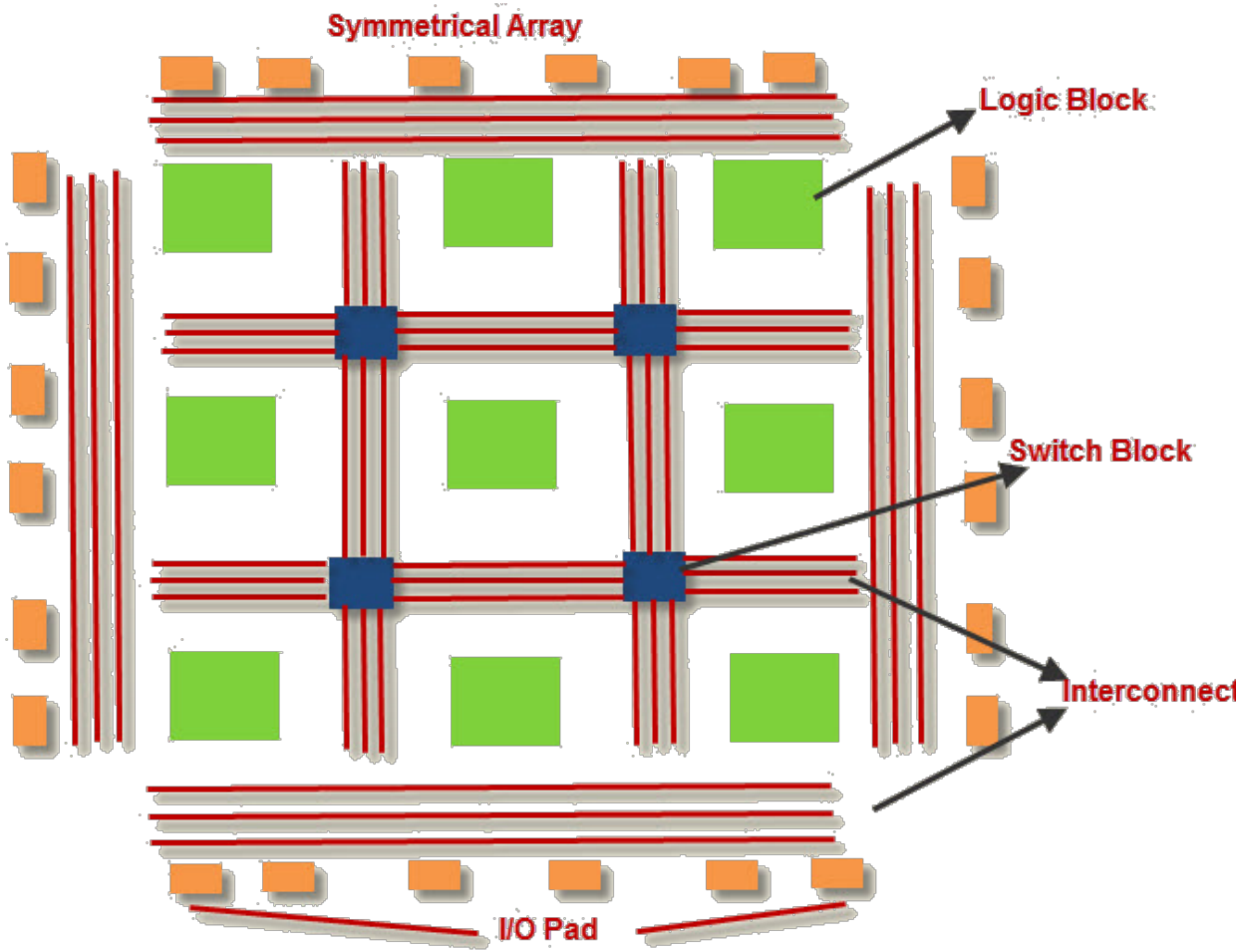
Background knowledge

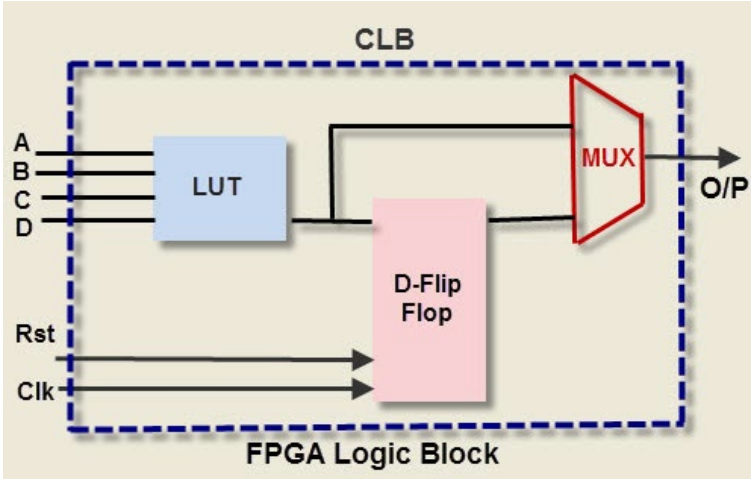
FPGA

- Field Programmable Gate Array (FPGA) is an integrated circuit containing gate matrix which can be programmed by the user "in the field" without using expensive equipment.
- An FPGA contains a set of programmable logic gates and rich interconnect resources, making it possible to implement complex digital circuits.
- FPGA devices are produced by a number of semiconductor companies:
  - Xilinx, Altera, Actel, Lattice, QuickLogic and Atmel.
- Configuration bitstream can be stored in FPGA using various technologies.
  - The majority of FPGAs is based on SRAM (Static RAM).

FPGA

- The general FPGA architecture consists of three types of modules. They are I/O blocks or Pads, Switch Matrix/ Interconnection Wires and Configurable Logic Blocks (CLB).



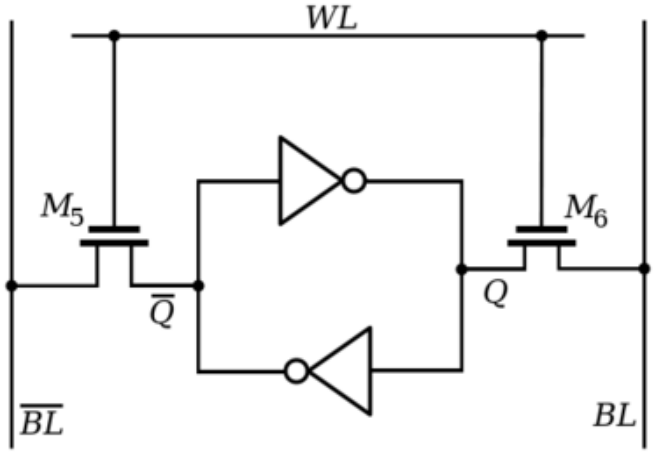


Memory cells: RAM

RAM is a **volatile** memory that the data is eventually lost when the memory is not powered

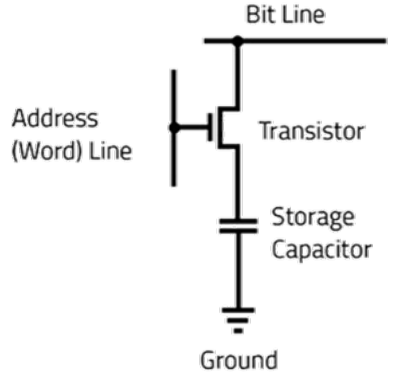
SRAM: Static random-access memory

- Speed ↑ size (bigger (6 MOSFETs))
- Cache memory
- Low power consumption
- 3 Operation States:
  - hold (WL=0), write & read (WL=1)



DRAM: Dynamic random-access memory

- Speed ↓ size (smaller (one MOSFET))
- Main memory
- Charge leakage
- High power consumption
- Refreshed frequently (few milliseconds)

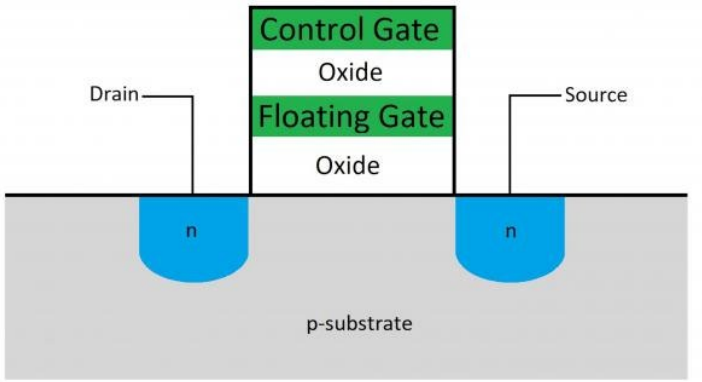


Single Memory Cell

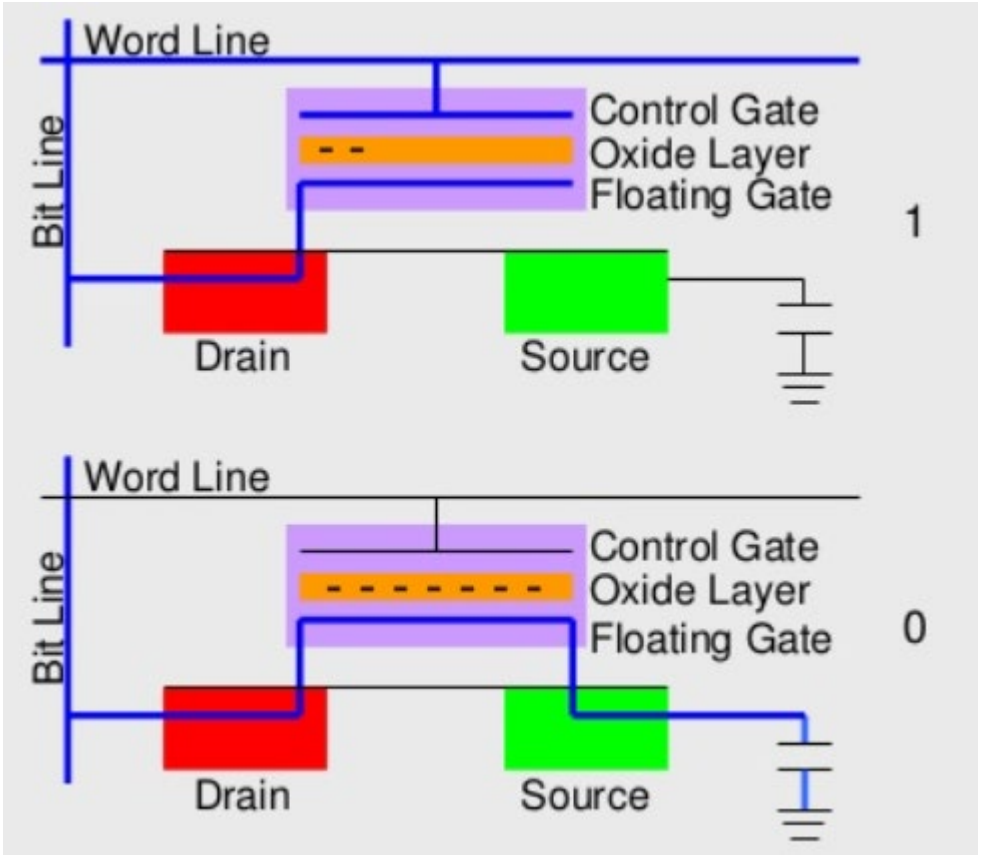
SRAM-based FPGAs

- It stores logic cells configuration data in the static memory
- Since SRAM is volatile, the FPGAs must be programmed (configured) upon start
- There are two basic modes of programming:
  - **Master mode**, when FPGA reads configuration data from an external source, such as an external Flash memory chip.
  - **Slave mode**, when FPGA is configured by an external master device, such as a processor (via a dedicated configuration interface or via a boundary-scan (JTAG) interface).
- SRAM-based FPGAs with an internal flash memory:
  - it contains internal flash memory blocks, thus eliminating the need to have an external non-volatile memory. It uses flash only during startup to load data to the SRAM configuration cells.

Programmable switch FPGAs



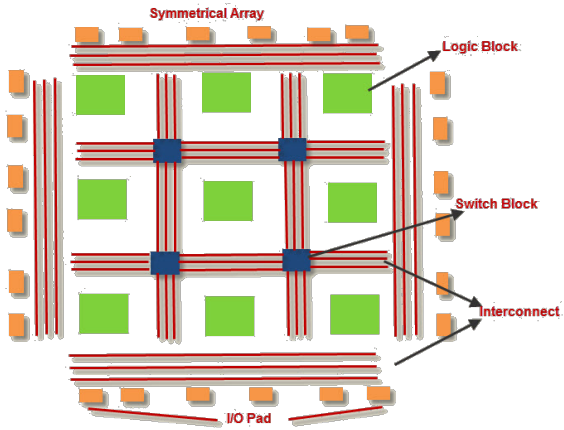
- It uses flash as a primary resource for configuration storage, and doesn't require SRAM
- Switch is a floating-gate transistor that can be turned off by injecting charge onto its floating gate
- It has a limit number of reprogramming
- More secure than SRAM-FPGA



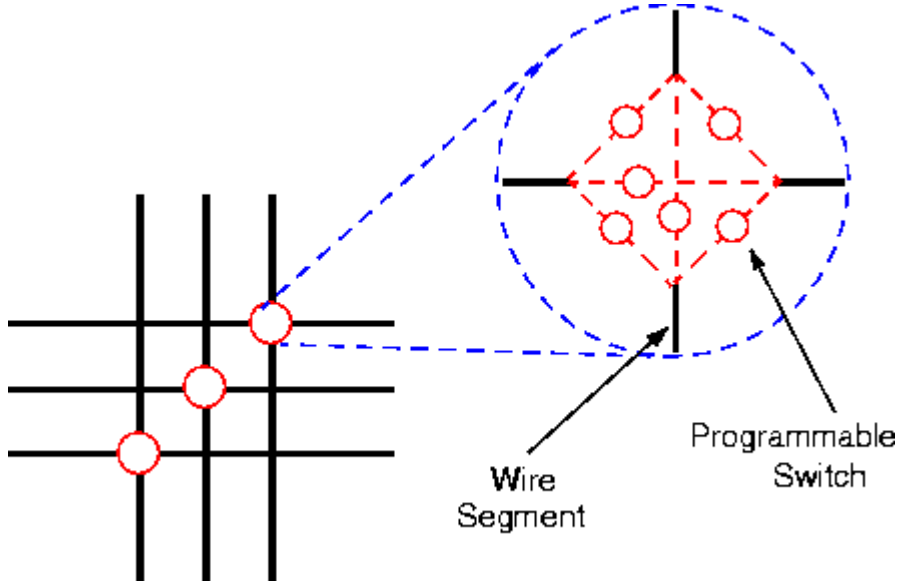
Fuse-based FPGA

- One-time program (OTP) FPGA
- The fuse makes or breaks the link between two wires
  - The amorphous silicon (nonconductive amterial) turns into a polycrystalline silicon-metal alloy (conductive) when exposed to high voltage.
- More secure than SRAM (no load from an external device)
- In high radiation environments (space or nuclear reactors), radiation events can cause SRAM, which contains the program, to change state but not in fused FPGA.

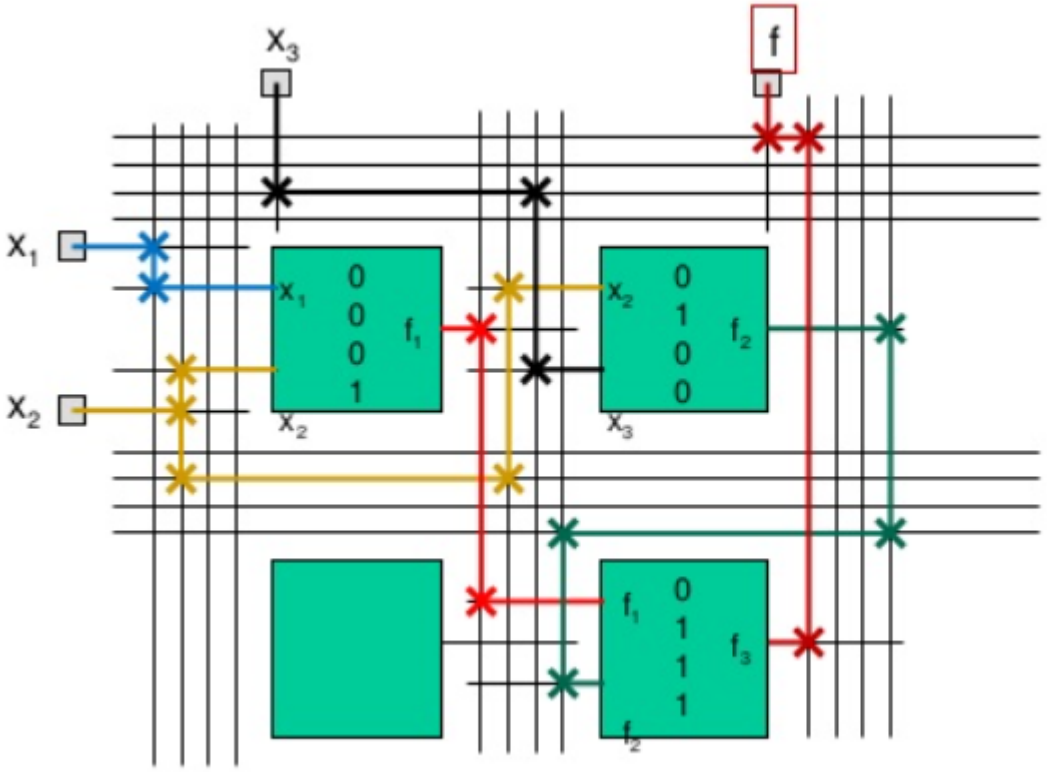
Programmable switch matrix



- All internal connections are composed of metal segments with programmable switching points and switching matrices to implement the desired routing.



Example: FPGA programming



$$f_1 = x_1 x_2$$

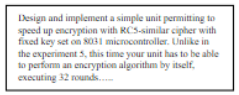
$$f_2 = \overline{x_2} x_3$$

$$f = x_1 x_2 + \overline{x_2} x_3$$

Other FPGA Building Blocks

- Clock distribution: element clock skew
- Embedded memory blocks
- Special purpose blocks:
  - DSP blocks: Hardware multipliers, adders and registers
  - Embedded microprocessors/microcontrollers
  - High-speed serial transceivers

Design flow

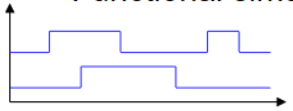


Specification / Pseudocode

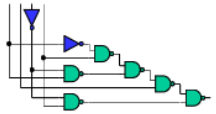
On-paper hardware design

VHDL description (Your Source Files)

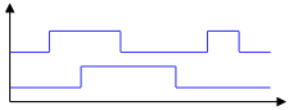
Functional simulation



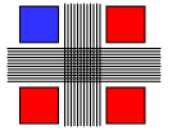
Synthesis



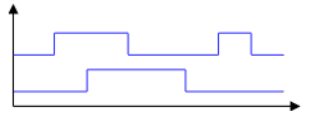
Post-synthesis simulation



Implementation

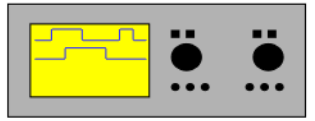


Timing simulation



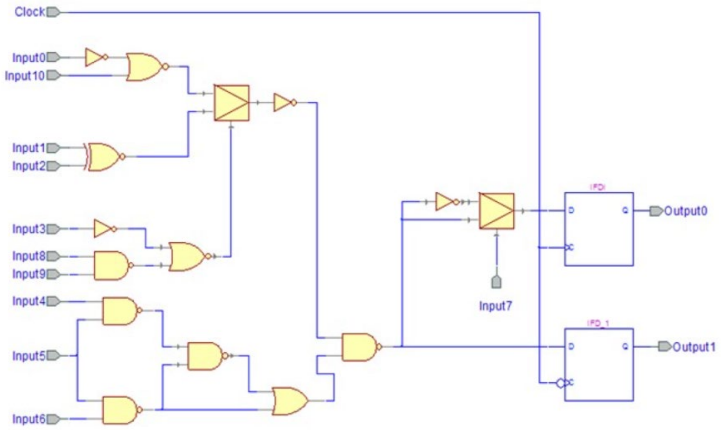
Configuration

On chip testing

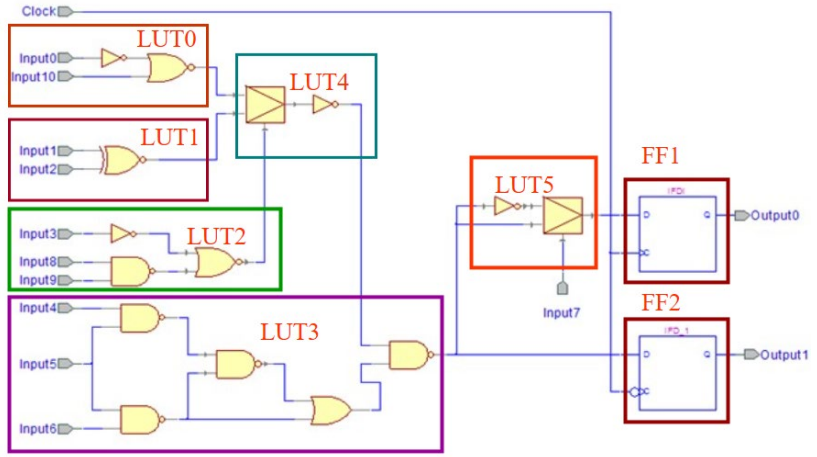


Synthesis

```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
                A1<=A when (NEG_A='0') else
                                not A;
                B1<=B when (NEG_B='0') else
                                not B;
                Y<=Y1 when (NEG_Y='0') else
                                not Y1;

                MUX_0<=A1 and B1;
                MUX_1<=A1 or B1;
                MUX_2<=A1 xor B1;
                MUX_3<=A1 xnor B1;

                with (L1 & L0) select
                                Y1<=MUX_0 when "00",
                                                MUX_1 when "01",
                                                MUX_2 when "10",
                                                MUX_3 when others;

end MLU_DATAFLOW;
```

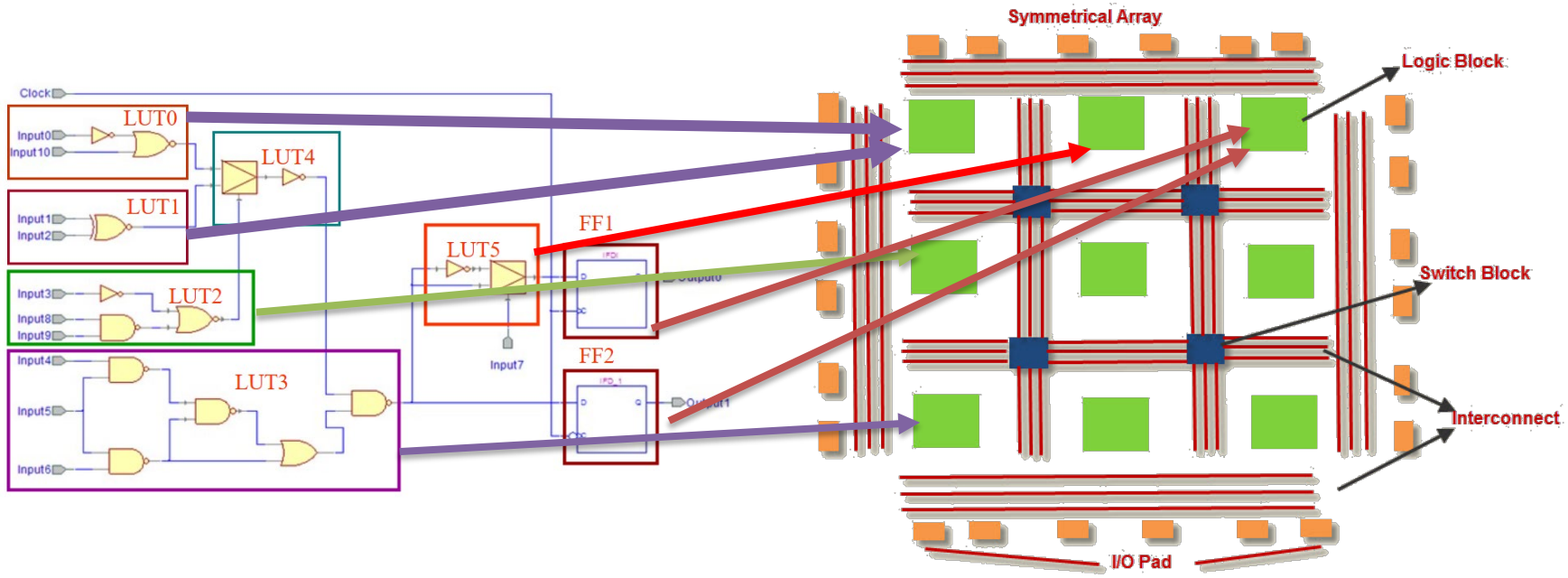


Circuit netlist

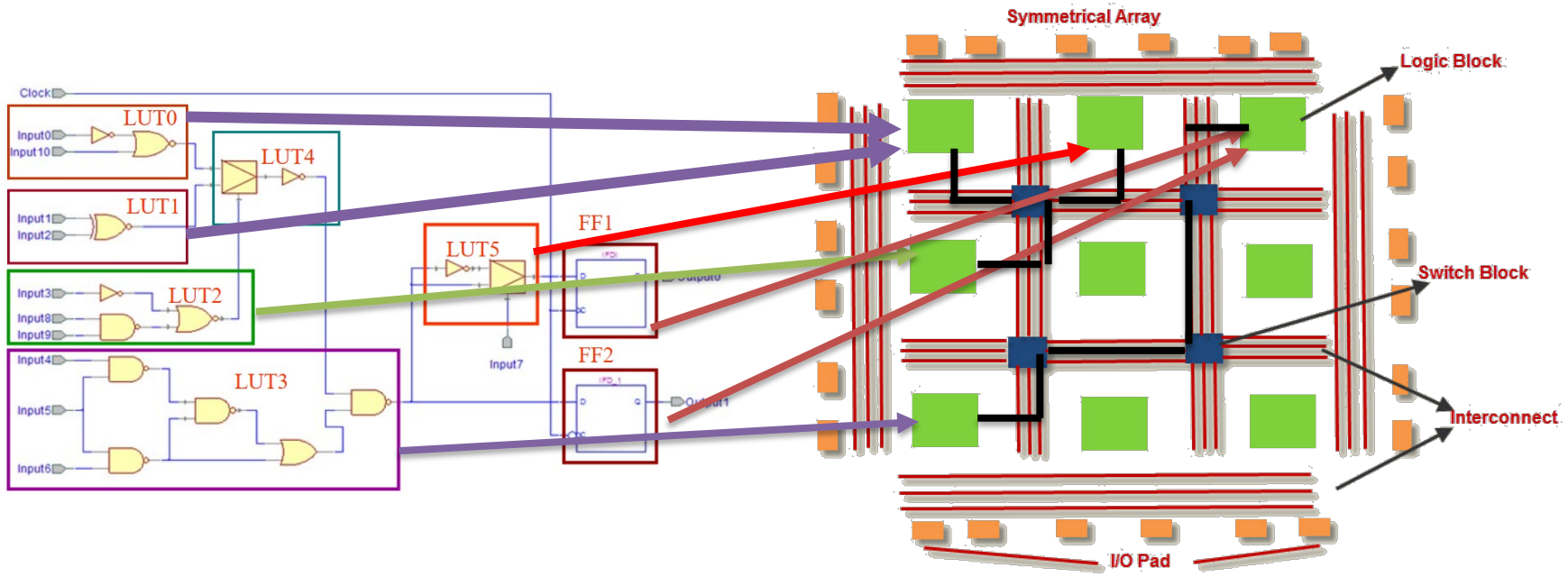


Mapping

Placing


Clock
LUT0
Input0
Input10
LUT4
LUT1
Input1
Input2
LUT5
FF1
Input3
LUT2
Input8
Input9
Input7
FF2
Input4
LUT3
Input5
Input6
Symmetrical Array
Logic Block
Switch Block
Interconnect
I/O Pad

Routing


Clock
LUT0
Input0
Input10
LUT4
LUT1
Input1
Input2
LUT5
FF1
Input3
LUT2
Input8
Input9
Input7
FF2
Input4
LUT3
Input5
Input6
Symmetrical Array
Logic Block
Switch Block
Interconnect
I/O Pad

Configuration

- Once a design is implemented, the synthesis tool generates a file that the FPGA can understand
- This file is called a bit stream: a BIT file (.bit extension)
- The BIT file can be downloaded directly to the FPGA, or can be converted into a PROM file which stores the programming information

ZYNQ architecture
CPU+FPGA

ZYNQ Architecture

- **Zynq** architecture created by Xilinx.
- It combines an FPGA with ARM cores and I/O into one product.
- The ARM part is called **Processing System** (**PS**).
  - application processor unit (APU)
  - fully integrated memory controllers
  - I/O peripherals
- The FPGA is called **Programmable Logic** (**PL**).

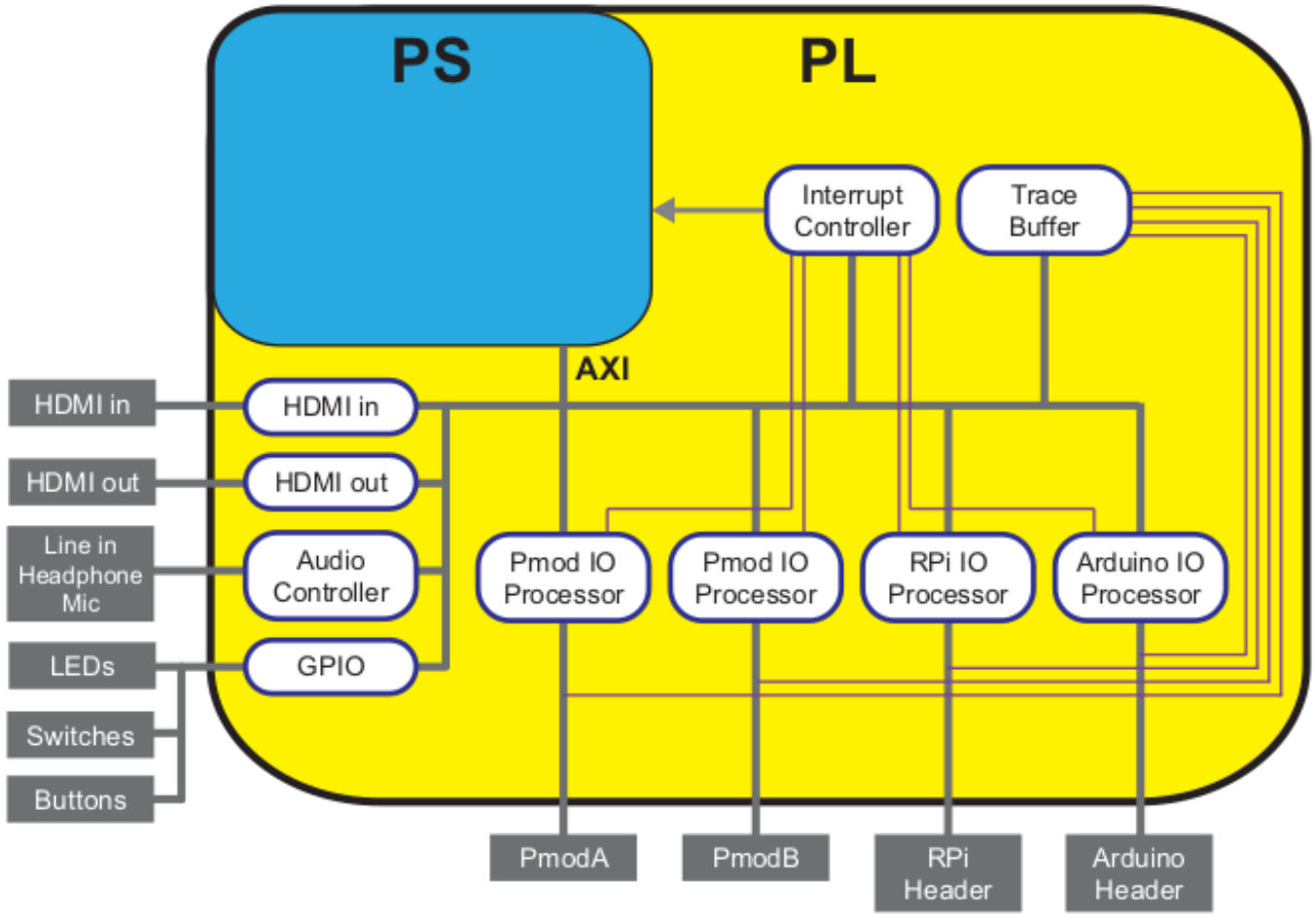
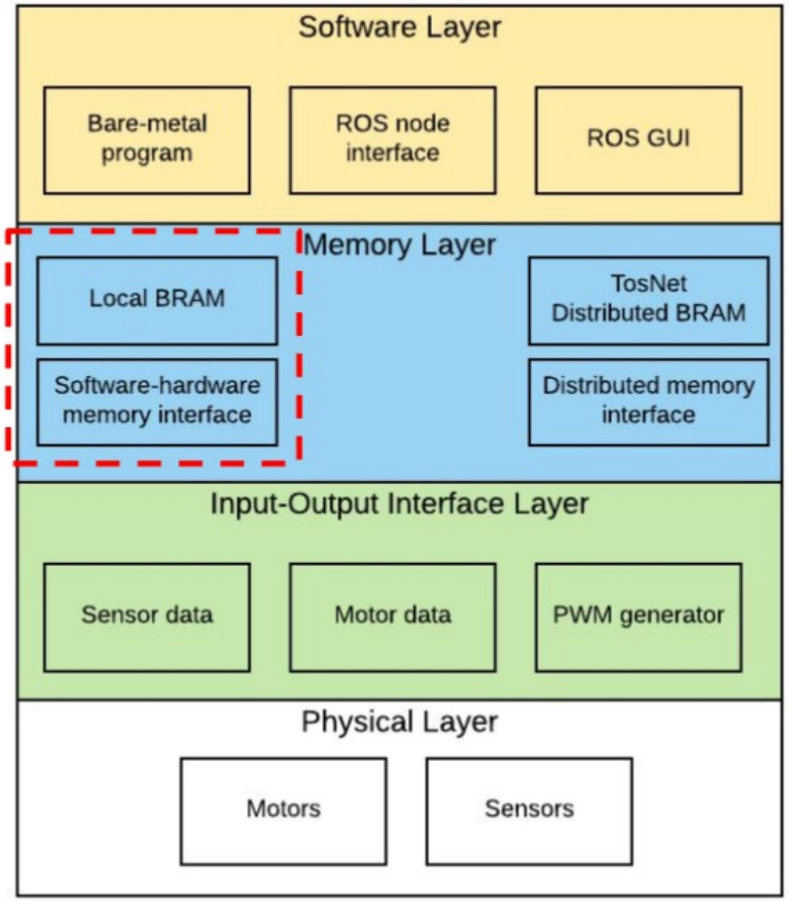


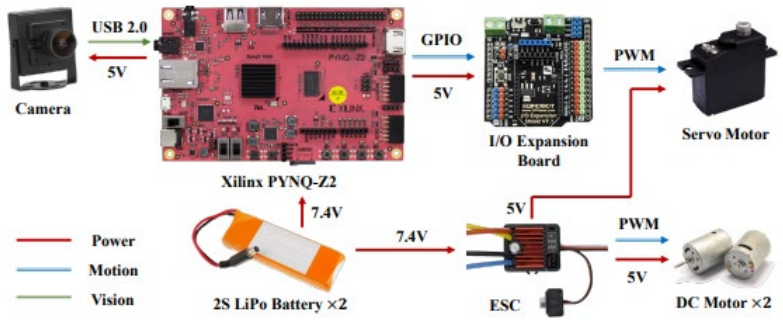
Figure 22.7: Interface functionality implemented by the base overlay (for PYNQ-Z2)

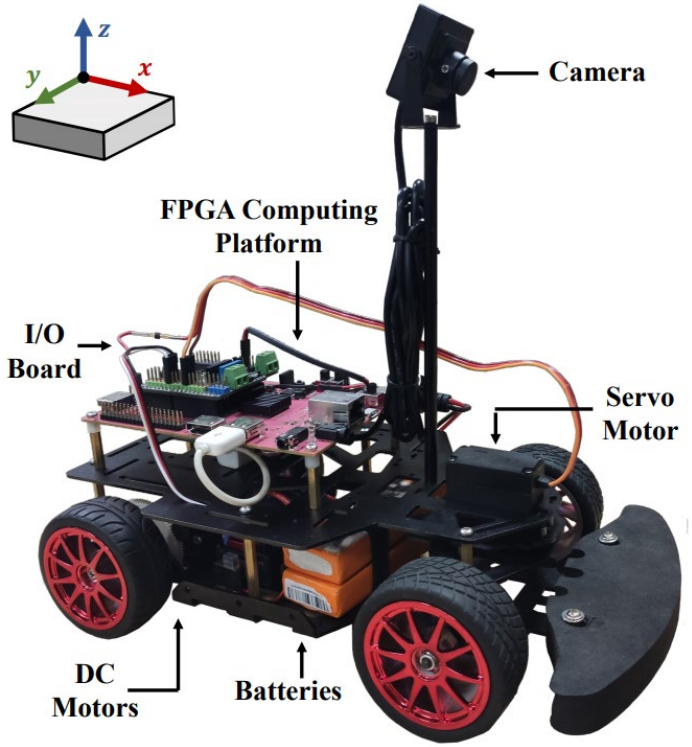
PS-PL specs

- PS: processing system
  - dual ARM Cortex-A9 processors, 866MHz to 1GHz frequency
  - multiple peripherals
  - hard silicon core
- PL: programmable logic
  - shares the same FPGA series 7 programmable logic
  - logic cells: 28k - 444k (430k to 6.6M gates)
  - flip-flops: 35k - 554k
  - DSP/MAC: 80 - 2020
  - AD converter: two 12bits

Example Implementation

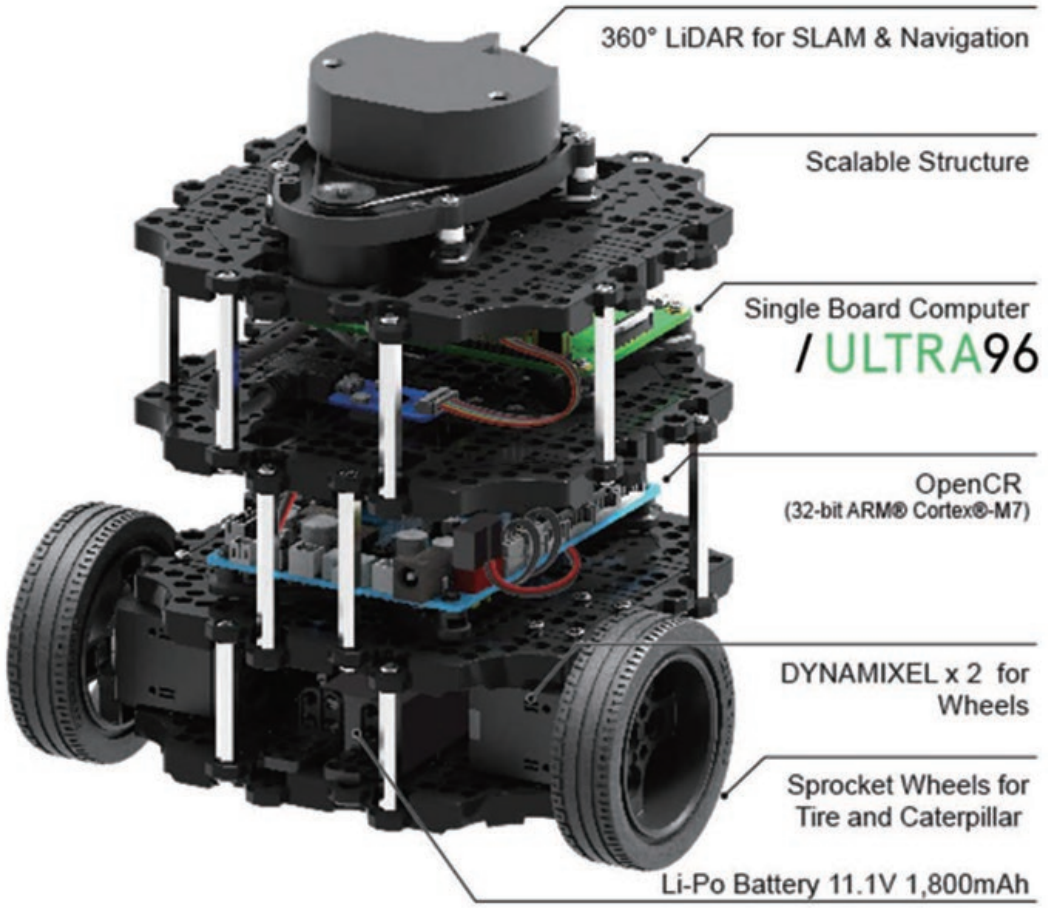






http://weisong.eng.wayne.edu/_resources/pdfs/wu20-HydraMini.pdf

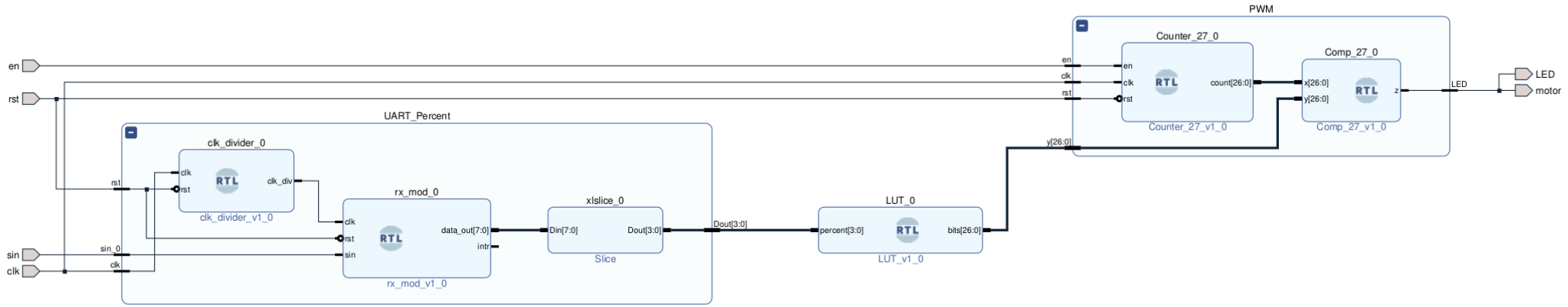
MPSoC


360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer
/ ULTRA96
OpenCR
(32-bit ARM® Cortex®-M7)
DYNAMIXEL x 2 for
Wheels
Sprocket Wheels for
Tire and Caterpillar
Li-Po Battery 11.1V 1,800mAh

Lab

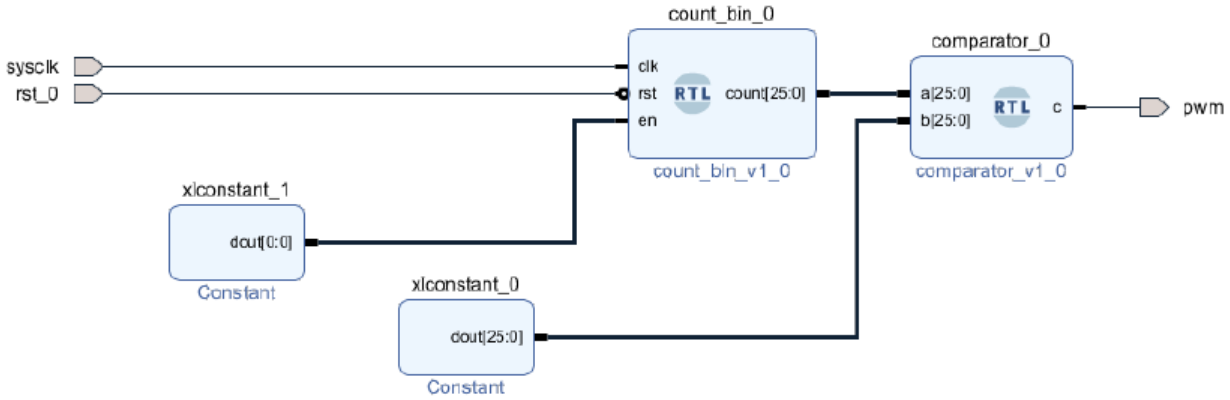
Lab exercies: UART to PWM

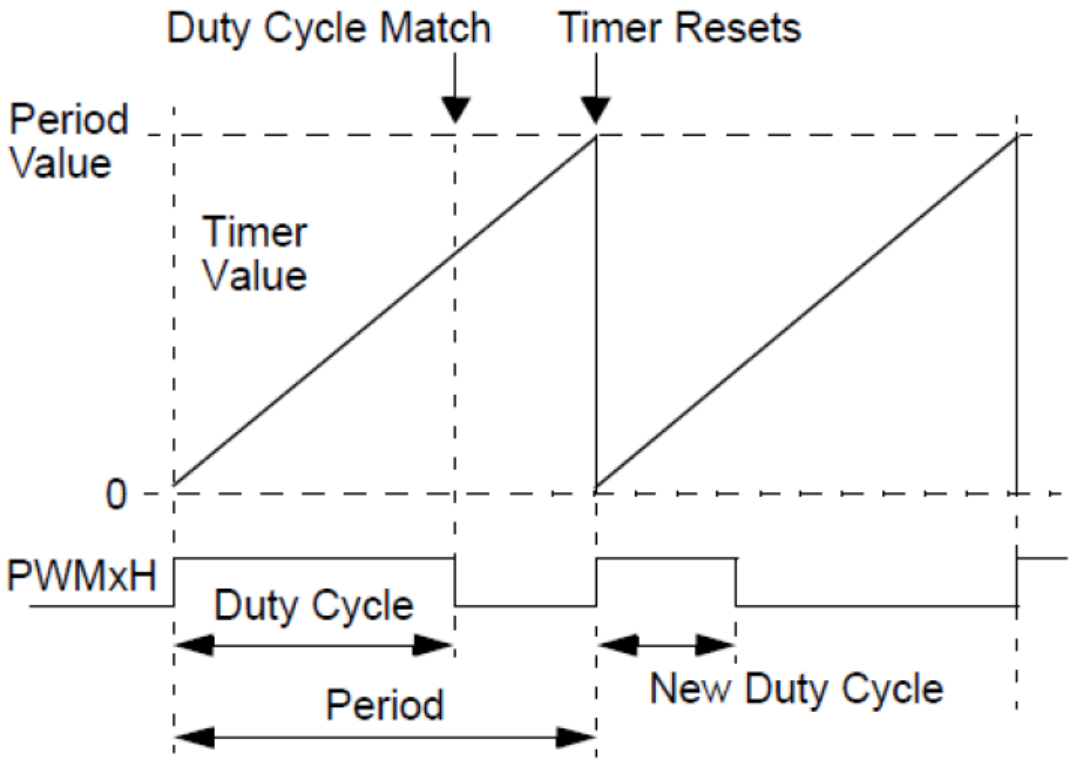
- The exercies aims to control the illumination of an LED using data coming from the UART, that represents the illumination value, and converts it into a PWM signal that will be sent to the LED.



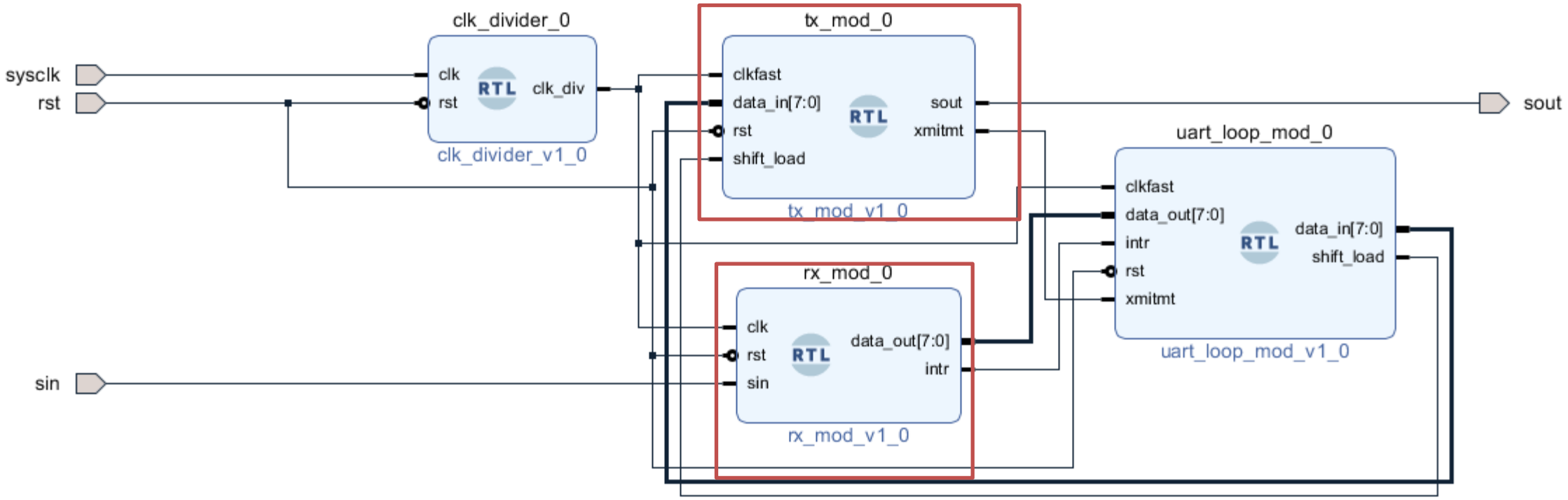
- Source: https://github.com/DIII-SDU-Group/EmbeddedSystems/tree/main/UART2PWM

Notes: PWM


count_bin_0
comparator_0
sysclk
rst_0
clk
rst
en
RTL
count[25:0]
a[25:0]
b[25:0]
RTL
c
pwm
count_bin_v1_0
comparator_v1_0
xlconstant_1
dout[0:0]
Constant
xlconstant_0
dout[25:0]
Constant


Duty Cycle Match
Timer Resets
Period
Value
Timer
Value
0
PWMxH
Duty Cycle
Period
New Duty Cycle

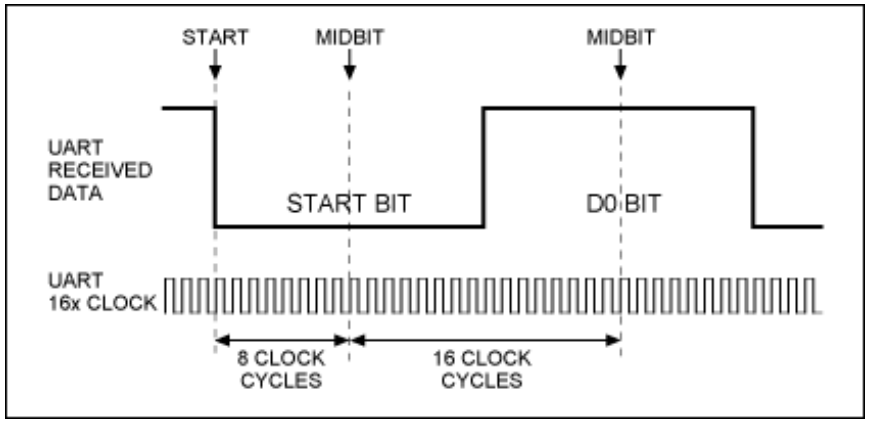
Notes: UART Overall system



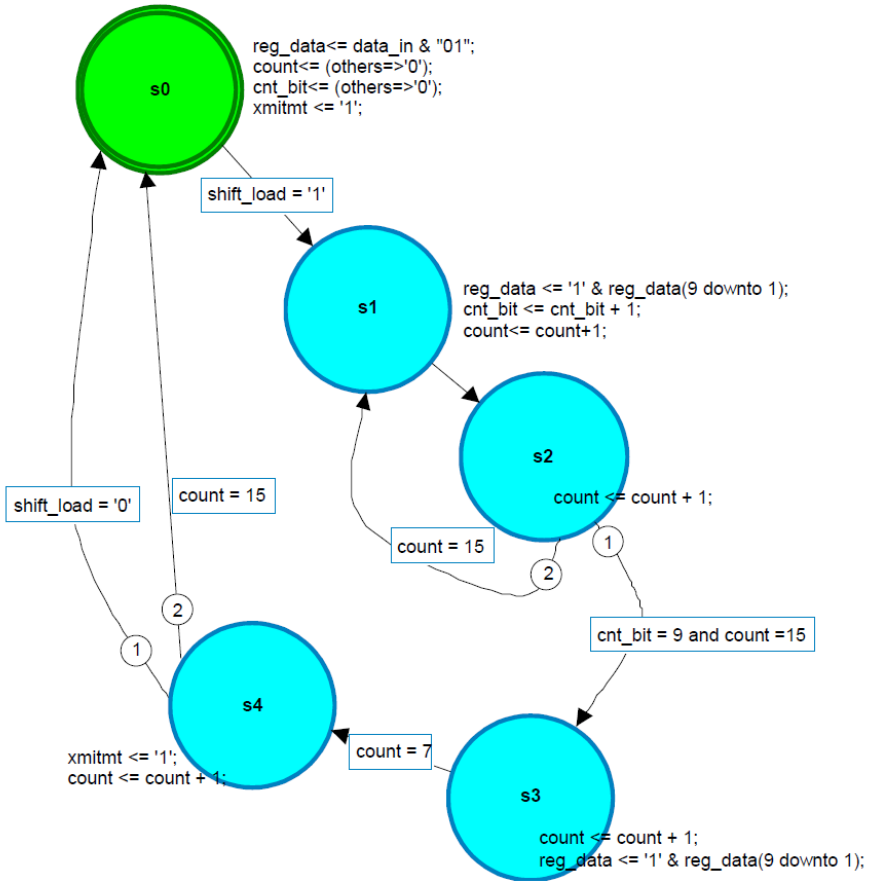
- The UART solo code is in itsLearning

Receiver

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;


ENTITY rx_mod IS
   PORT(
      clk      : IN     std_logic;
      rst      : IN     std_logic;
      sin      : IN     std_logic;
      data_out : OUT    std_logic_vector (7 DOWNTO 0);
      intr     : OUT    std_logic);

END rx_mod ;

-- hds interface_end
ARCHITECTURE rtl OF rx_mod IS
    signal rxreg: std_logic_vector(9 downto 0);
    signal count: unsigned (3 downto 0);
    signal rxmt: std_logic;
    signal rxin,start_flag: std_logic;
    begin
        process (clk, rst)
            begin
                if (rst = '0') then
                    count <= (others => '0');
                    rxmt <= '1';
                    rxreg <= (others => '1');
                    intr <= '0';
                    rxin <= '1';
                    start_flag<='0';
                elsif (rising_edge(clk)) then
                    rxin<=sin;
                    if (rxmt = '1' and rxin = '0') then
                        count <= (others => '0');
                        rxmt <= '0';
                        rxreg <= (others => '1');
                        start_flag<='0';
                    elsif (count = 7 and rxmt = '0' and rxin = '0' and start_flag='0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= (others => '0');
                        start_flag<='1';
                    elsif (count = 15 and rxmt = '0') then
                        rxreg <= rxin & rxreg(9 downto 1);
                        count <= count + 1;
                    else
                        count <= count + 1;
                    end if;
                    if (rxmt = '0' and rxreg(9) = '1' and rxreg(0) = '0') then
                        intr <= '1';
                        rxmt <= '1';
                    else
                        intr <= '0';
                    end if;
                end if;
        end process;
        data_out <= rxreg(8 downto 1);
END rtl;
```



Transmitter



- Default values are 0