

Układy FPGA w zastosowaniach kosmicznych

Dawid Linowski
Tomasz Rybak

#3 Spotkanie Gdańsk Embedded Meetup



Agenda

- Syderal Polska
- FPGA - wprowadzenie
- FPGA - design flow
- FPGA - narzędzia
- FPGA - przykład projektu w VHDL
- Elektronika w kosmosie
- FPGA w kosmosie
- Fazy rozwoju FPGA w projekcie lotnym



SYDERAL POLSKA

ELECTRONICS & SOFTWARE

- Firma założona w czerwcu 2016 r.
- Obecnie 15 pracowników
- Kompetencje w projektowaniu elektroniki i oprogramowania
- Współpraca z **Syderal Swiss** - transfer technologii
- Główne obszary działań:
 - Kontrolery mechanizmów i instrumentów
 - Moduły pamięci oparte o FLASH
 - Technologie związane ze splątaniem kwantowym



Czym jest FPGA?

Definicja

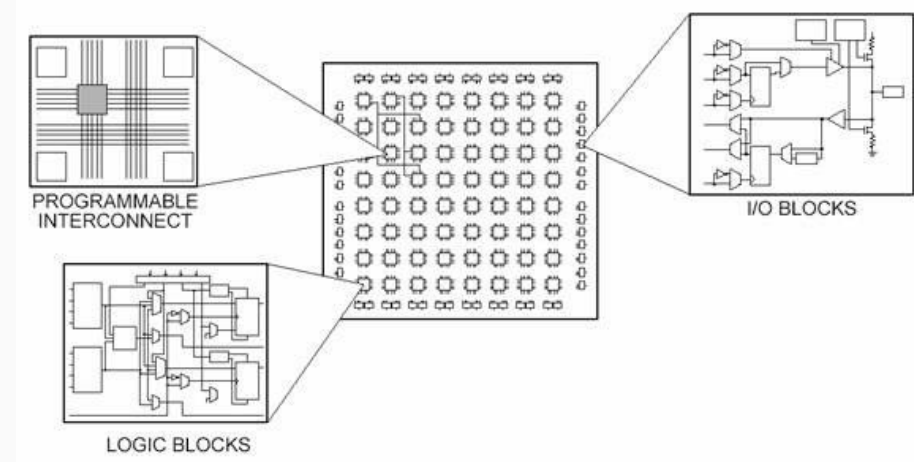


FPGA (Field-Programmable Gate Array) - bezpośrednio programowalna macierz bramek, rodzaj **programowalnego układu logicznego**.

Układ FPGA nie wykonuje żadnej funkcji, dopóki nie zostanie odpowiednio zaprogramowany przez projektanta. Projektowanie polega na tworzeniu połączeń między elementami logicznymi wewnątrz układu FPGA, który najczęściej jest wielokrotnie programowalny.

Architektura FPGA

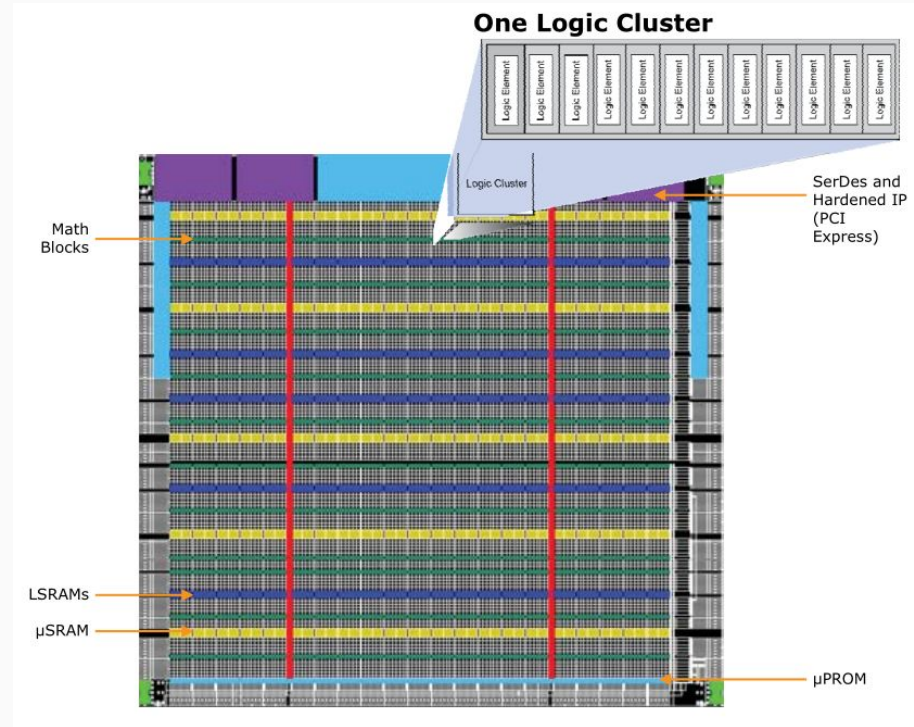
- Konfigurowalne bloki wejścia/wyjścia
- Programowalna macierz połączeń
- Bloki logiczne
- Dedykowane bloki o określonych, specjalistycznych funkcjach:
 - RAM
 - CPU (ARM core, IBM PowerPC...)
 - Kontrolery pamięci DDR
 - Kontrolery PCIe, Ethernet MAC,
 - Gigabitowe transceivery,
 - Komponenty do serializacji/deserializacji danych
 - bloki MAC (Multiply Accumulate), przydatne dla wszelkich operacji DSP



<https://www.ni.com/pl-pl/innovations/white-papers/08/fpga-fundamentals.html>

Architektura FPGA

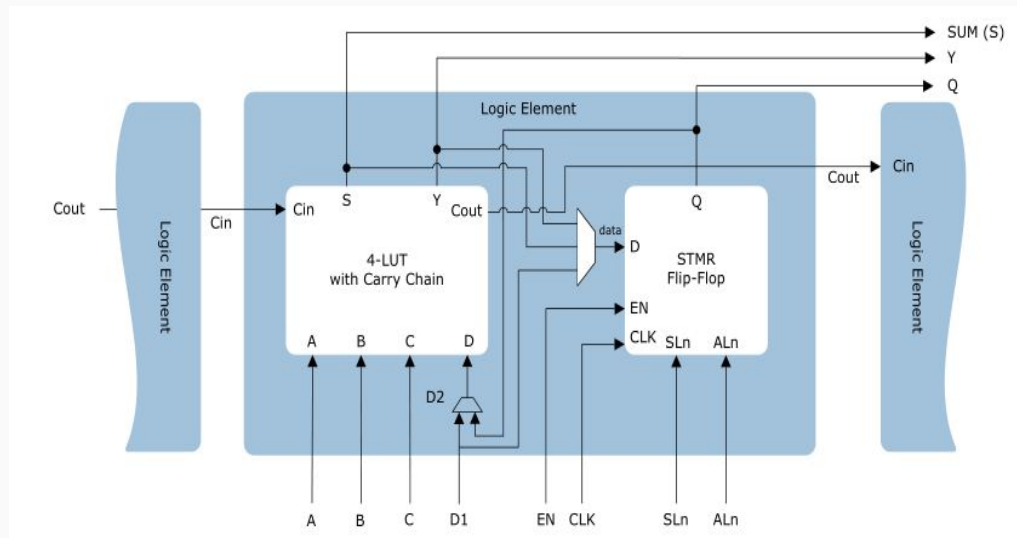
- Elementy logiczne (Logic Cluster = 12 x Logic Element)
- Pamięci wewnętrzne:
 - ulotne (SRAM)
 - nieulotne (PROM)
- Bloki matematyczne
- Hard IP-s (PCIe,...)



Microsemi "UG0574 User Guide RTG4 FPGA Fabric"

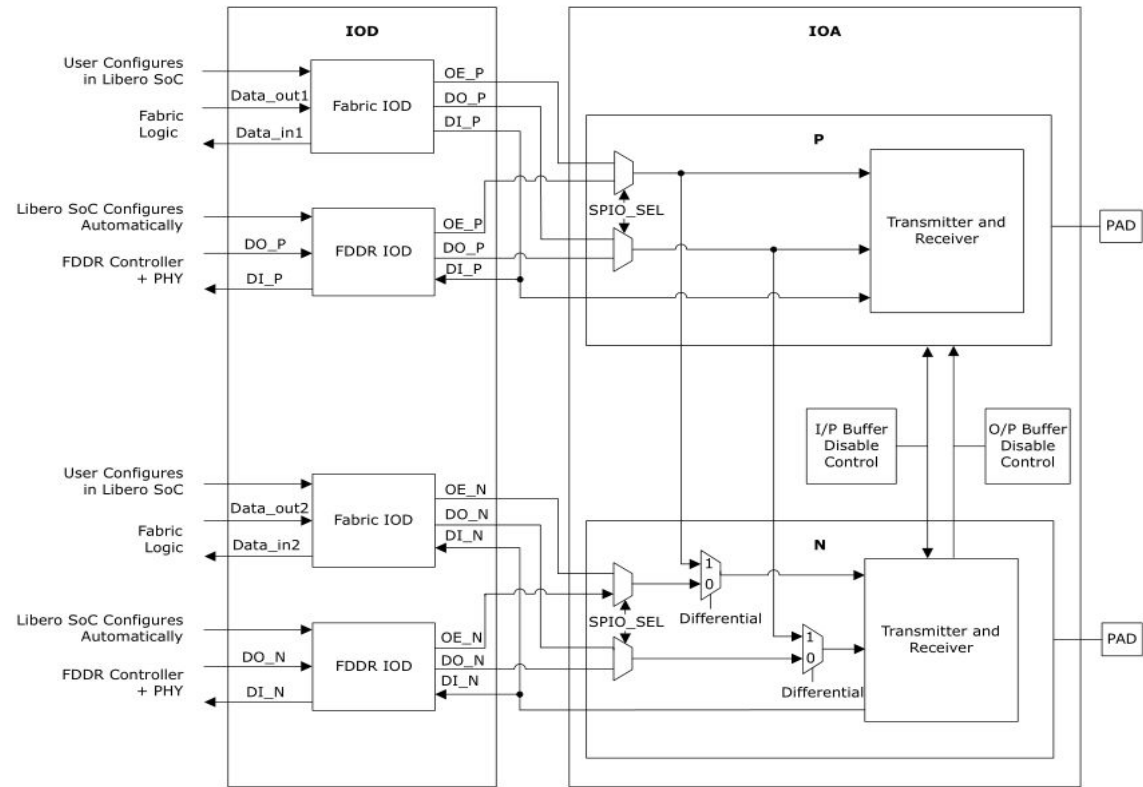
Podstawowy element logiczny (LE) na przykładzie RTG4:

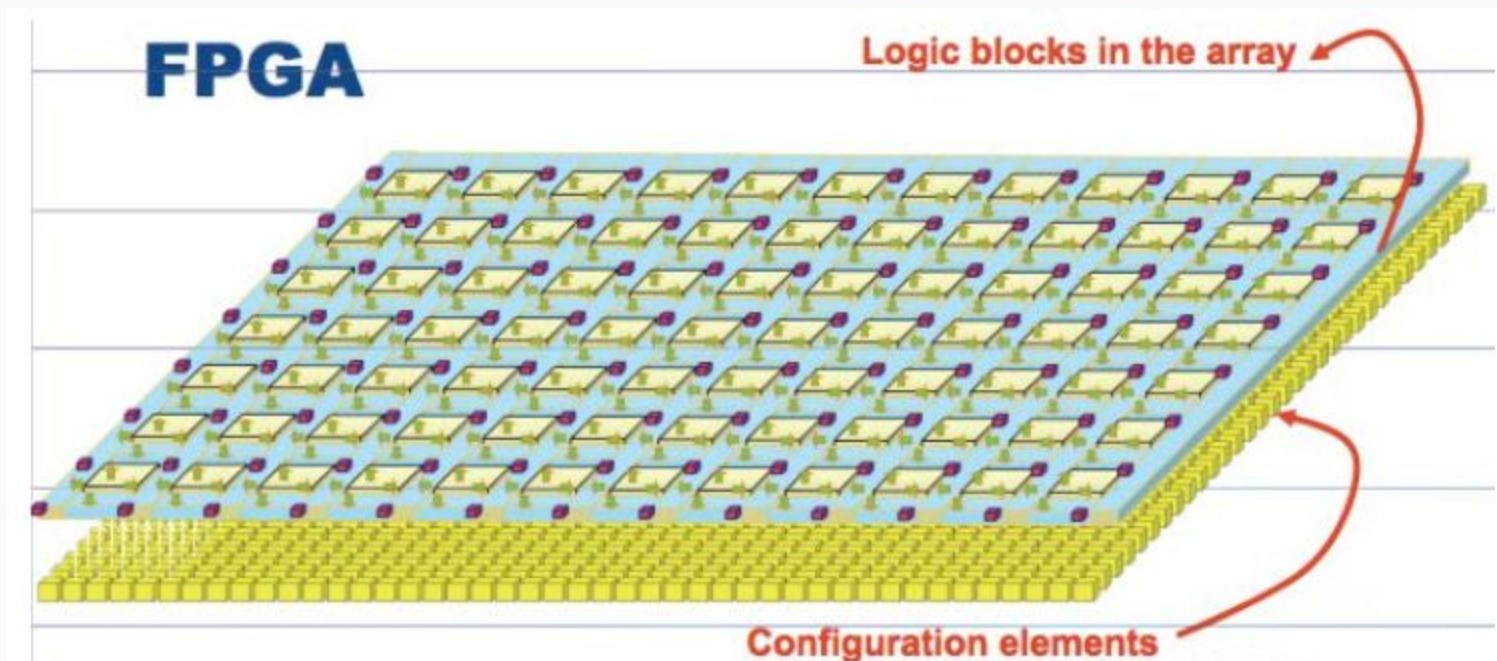
- 4-wejściowy LUT + przerzutnik typu D
- Można wykorzystać:
 - sam LUT
 - sam przerzutnik
 - oba elementy jednocześnie
- Wybór LE w FPGA, realizacja funkcji logicznej lub arytmetycznej w LUT, połączenia między LE - automatyczne podczas syntezy i “place and route”



Microsemi “UG0574 User Guide RTG4 FPGA Fabric”

- Część cyfrowa
 - bufor wejściowy
 - bufor wyjściowy
 - bufor trójstanowy
 - przerzutnik w padzie
- Część analogowa
 - transceivery (umożliwiają PCIe, SATA, ...)





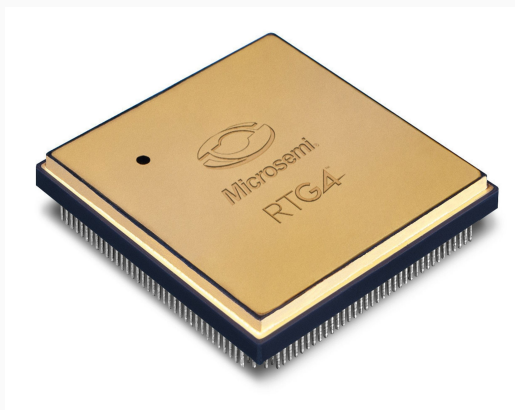
FPGA a pamięć konfiguracyjna

Typ pamięci konfiguracyjnej	Antifuse	Flash	SRAM
Natura programowania	Elektrycznie programowalne, połączenia tworzą się na zasadzie odwrotnej do bezpieczników	Elektrycznie programowalne tranzystory	Pamięć statyczna (macierz latchy)
Pamięć konfiguracyjna	Nieulotna	Nieulotna	Ulotna (potrzebna wewnętrzna lub zewnętrzna pamięć Flash)
Liczba możliwych programowań	Jednokrotne	Wielokrotne	Wielokrotne
Producent	Microsemi, Cobham	Microsemi, BRAVE	Xilinx, Atmel

FPGA a pamięć konfiguracyjna - przykłady



Microsemi RTAX2000S



Microsemi RTG4

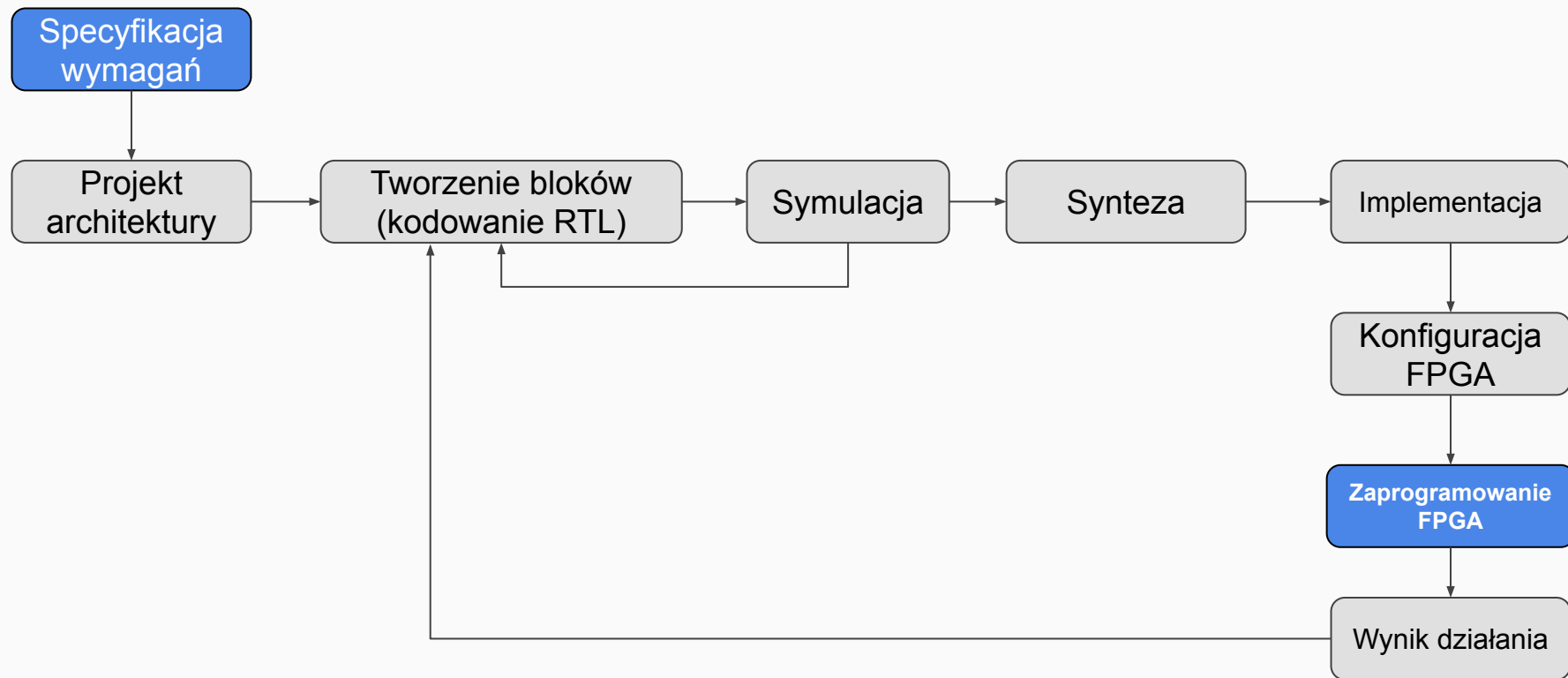


Xilinx Virtex-5QV

FPGA - co wyróżnia?

- Możliwość równoległego jak i sekwencyjnego przetwarzania.
- Pełen determinizm - precyzyjna kontrola nad czasem przetwarzania (z wyłączeniem przejść między różnymi domenami zegarowymi).
- Duża elastyczność - projektant sam tworzy “bloki sprzętowe”.
- Projektowanie bardzo niskopoziomowe - na poziomie bramek logicznych i połączeń między nimi.
- Często jako zamiennik układu ASIC przy produkcji małoseryjnej.

“Design Flow”

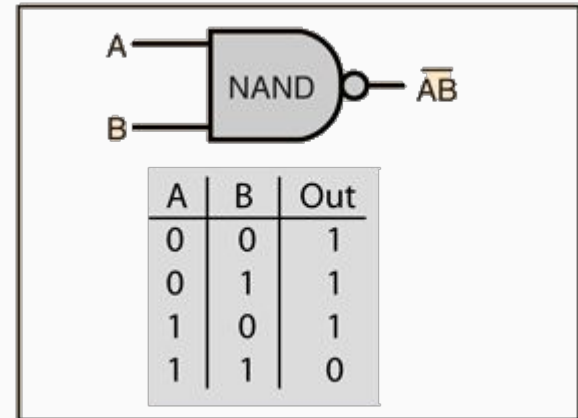


Jakie narzędzia są nam potrzebne? - Przykłady

- Kodowanie - język opisu sprzętu: **VHDL**, Verilog, SystemC, SystemVerilog, MyHDL...
- Symulacja: **Modelsim**, **Riviera-PRO**, Active-HDL, VCS, ISE, Vivado, Quartus, GHDL, Icarus Verilog, EDA playground...
- Synteza i implementacja: **Synplify**, Vivado, Quartus Prime...
- Programowanie: płytka deweloperska z układem FPGA lub programator dla samego układu
- Debugowanie: Chipscope Pro (Xilinx), Identify RTL Debugger (Synopsys)

VHDL - Prosty przykład

```
Ln# |
1   | LIBRARY ieee;
2   | USE ieee.std_logic_1164.all;
3   | USE ieee.numeric_std.all;
4   |
5   | entity weird_nand is
6   | port(
7   |     Clk      : in    std_logic; -- main clock
8   |     Rst_n    : in    std_logic; -- reset, active low
9   |     A        : in    std_logic; -- A input
10  |     B        : in    std_logic; -- B input
11  |     Output   : out   std_logic -- output
12  | );
13  | end weird_nand;
14  |
15  | architecture rtl of weird_nand is
16  |     signal and_output : std_logic;
17  |
18  | begin
19  |
20  |     Output <= not and_output;
21  |
22  |     and_process : process(Rst_n, Clk)
23  |     begin
24  |         if (Rst_n = '0') then
25  |             and_output <= '1';
26  |         elsif rising_edge(Clk) then
27  |             and_output <= A and B;
28  |         end if;
29  |     end process;
30  | end rtl;
```



VHDL - Prosty przykład

```
Ln# |
1  | LIBRARY ieee;
2  | USE ieee.std_logic_1164.all;
3  | USE ieee.numeric_std.all;
4  |
5  | entity weird_nand is
6  | port(
7  |     Clk      : in    std_logic; -- main clock
8  |     Rst_n    : in    std_logic; -- reset, active low
9  |     A        : in    std_logic; -- A input
10 |     B        : in    std_logic; -- B input
11 |     Output   : out   std_logic -- output
12 | );
13 | end weird_nand;
14 |
15 | architecture rtl of weird_nand is
16 |
17 |     signal and_output : std_logic;
18 |
19 | begin
20 |
21 |     Output <= not and_output;
22 |
23 |     and_process : process(Rst_n, Clk)
24 |     begin
25 |         if (Rst_n = '0') then
26 |             and_output <= '1';
27 |         elsif rising_edge(Clk) then
28 |             and_output <= A and B;
29 |         end if;
30 |     end process;
31 | end rtl;
```

Entity - czyli definicja elementu projektu.

VHDL - Prosty przykład

```
Ln# |
1  | LIBRARY ieee;
2  | USE ieee.std_logic_1164.all;
3  | USE ieee.numeric_std.all;
4  |
5  | entity weird_nand is
6  | port(
7  |     Clk      : in    std_logic; -- main clock
8  |     Rst_n    : in    std_logic; -- reset, active low
9  |     A        : in    std_logic; -- A input
10 |     B        : in    std_logic; -- B input
11 |     Output   : out   std_logic -- output
12 | );
13 | end weird_nand;
14 |
15 | architecture rtl of weird_nand is
16 |
17 |     signal and_output : std_logic;
18 |
19 | begin
20 |
21 |     Output <= not and_output;
22 |
23 |     and_process : process(Rst_n, Clk)
24 |     begin
25 |         if (Rst_n = '0') then
26 |             and_output <= '1';
27 |         elsif rising_edge(Clk) then
28 |             and_output <= A and B;
29 |         end if;
30 |     end process;
31 | end rtl;
```

Process - wewnątrz tego bloku instrukcje wykonywane są szeregowo.

Blok jest wywoływany przy zmianie wartości jednego z jego “parametrów” (tutaj Rst_n i Clk, które są na tzw. “liście czułości”).

Zmiana wartości sygnałów użytych wewnątrz procesu następuje dopiero po jego zakończeniu.

VHDL - Prosty przykład

```
Ln# |
1  | LIBRARY ieee;
2  | USE ieee.std_logic_1164.all;
3  | USE ieee.numeric_std.all;
4  |
5  | entity weird_nand is
6  | port(
7  |     Clk      : in    std_logic; -- main clock
8  |     Rst_n    : in    std_logic; -- reset, active low
9  |     A        : in    std_logic; -- A input
10 |     B        : in    std_logic; -- B input
11 |     Output   : out   std_logic -- output
12 | );
13 | end weird_nand;
14 |
15 | architecture rtl of weird_nand is
16 |
17 |     signal and_output : std_logic;
18 |
19 | begin
20 |
21 |     Output <= not and_output;
22 |
23 |     and_process : process(Rst_n, Clk)
24 |     begin
25 |         if (Rst_n = '0') then
26 |             and_output <= '1';
27 |         elsif rising_edge(Clk) then
28 |             and_output <= A and B;
29 |         end if;
30 |     end process;
31 | end rtl;
```

Część kombinacyjna - niezależna od resetu czy zegara. Funkcja logiczna jest natychmiast wykonywana, a jej wynik przypisywany do sygnału.

VHDL - Prosty przykład

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity testbench is
5  end testbench;
6
7  architecture tb of testbench is
8  component weird_nand is
9  port(
10     Clk      : in    std_logic; -- main clock
11     Rst_n    : in    std_logic; -- reset, active low
12     A        : in    std_logic; -- A input
13     B        : in    std_logic; -- B input
14     Output   : out   std_logic; -- output
15 );
16 end component;
17
18 constant C_RST_ON_TIME : time := 1 us;
19 constant C_CLK_PERIOD  : time := 50 ns;
20
21 signal Clk_test      : std_logic;
22 signal Rst_n_test    : std_logic;
23 signal A_test        : std_logic := '0';
24 signal B_test        : std_logic := '0';
25 signal Out_test      : std_logic;
26
27 begin
28     i_dut : weird_nand
29     port map(
30         Clk      => Clk_test,
31         Rst_n    => Rst_n_test,
32         A        => A_test,
33         B        => B_test,
34         Output   => Out_test
35     );
36
37     Rst_n_test <= '0', '1' after C_RST_ON_TIME;
38     --generate main 20 MHz clock
39     P_Clk_test : process
40
41     P_main_test : process
42
43
44
45 end tb;
```

Testbench dla przykładu.

P_Clk_test odpowiada za generację zegara 20MHz.

P_main_test generuje pobudzenia testowe i weryfikuje sygnał wyjściowy.

VHDL - Prosty przykład

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity testbench is
5  end testbench;
6
7  architecture tb of testbench is
8  component weird_nand is
9  port(
10     Clk      : in    std_logic; -- main clock
11     Rst_n    : in    std_logic; -- reset, active low
12     A        : in    std_logic; -- A input
13     B        : in    std_logic; -- B input
14     Output   : out   std_logic; -- output
15 );
16 end component;
17
18 constant C_RST_ON_TIME : time := 1 us;
19 constant C_CLK_PERIOD  : time := 50 ns;
20
21 signal Clk_test      : std_logic;
22 signal Rst_n_test    : std_logic;
23 signal A_test        : std_logic := '0';
24 signal B_test        : std_logic := '0';
25 signal Out_test      : std_logic;
26
27 begin
28     i_dut : weird_nand
29     port map(
30         Clk      => Clk_test,
31         Rst_n    => Rst_n_test,
32         A        => A_test,
33         B        => B_test,
34         Output   => Out_test
35     );
36
37     Rst_n_test <= '0', '1' after C_RST_ON_TIME;
38     --generate main 20 MHz clock
39     P_Clk_test : process
40     P_main_test : process
41
42
43
44
45 end tb;
```

Deklaracja komponentu - zgodnie z wcześniej przedstawioną definicją entity.

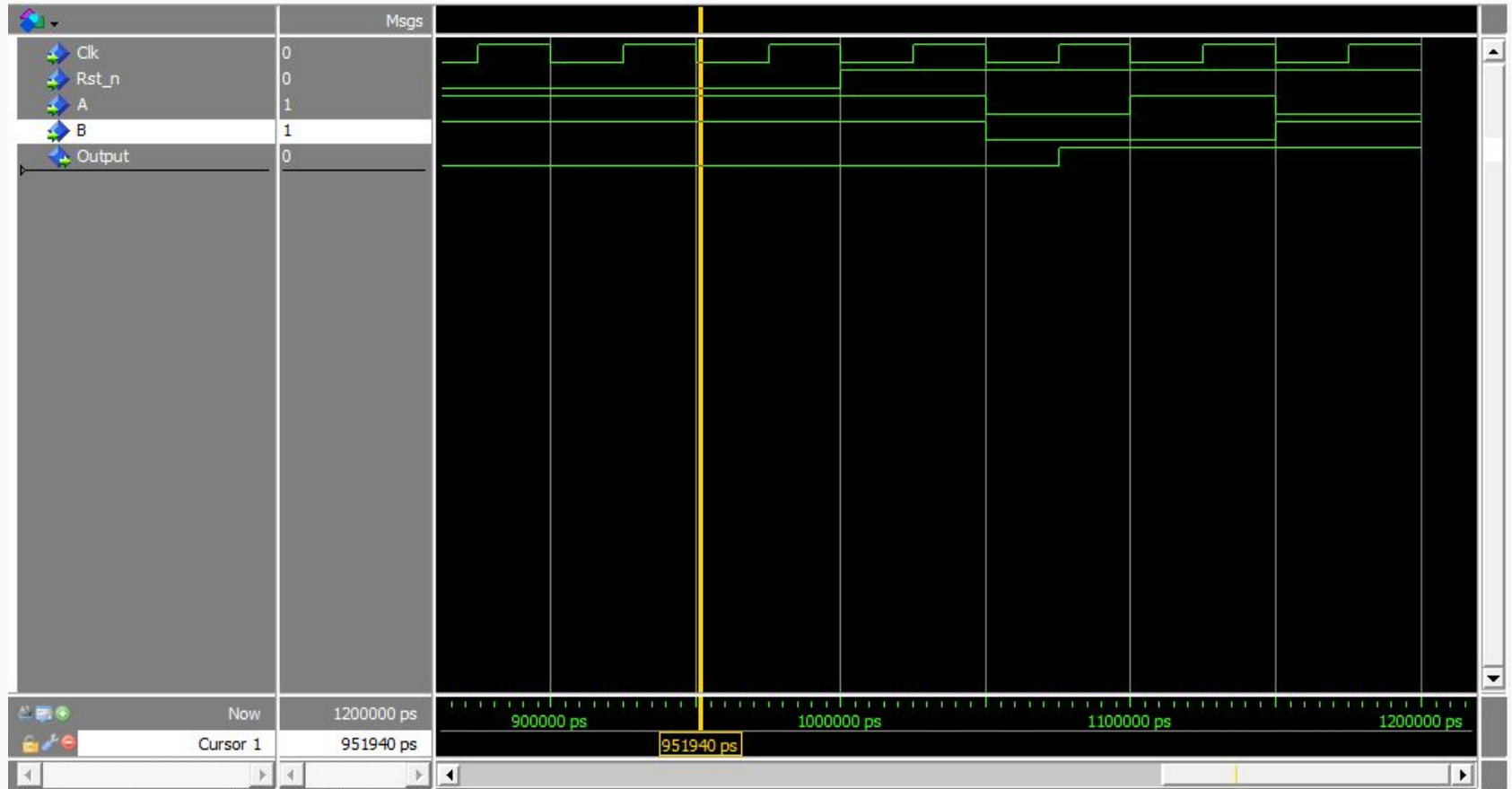
VHDL - Prosty przykład

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity testbench is
5  end testbench;
6
7  architecture tb of testbench is
8  component weird_nand is
9  port(
10     Clk      : in    std_logic; -- main clock
11     Rst_n    : in    std_logic; -- reset, active low
12     A        : in    std_logic; -- A input
13     B        : in    std_logic; -- B input
14     Output   : out   std_logic; -- output
15 );
16 end component;
17
18 constant C_RST_ON_TIME : time := 1 us;
19 constant C_CLK_PERIOD  : time := 50 ns;
20
21 signal Clk_test      : std_logic;
22 signal Rst_n_test    : std_logic;
23 signal A_test        : std_logic := '0';
24 signal B_test        : std_logic := '0';
25 signal Out_test      : std_logic;
26
27 begin
28     i_dut : weird_nand
29     port map(
30         Clk      => Clk_test,
31         Rst_n    => Rst_n_test,
32         A        => A_test,
33         B        => B_test,
34         Output   => Out_test
35     );
36
37     Rst_n_test <= '0', '1' after C_RST_ON_TIME;
38     --generate main 20 MHz clock
39     P Clk test : process
40
41     P main test : process
42
43
44
45 end tb;
```

Instancja - “wywołanie” komponentu w architekturze.

Można tak osadzać wiele instancji jednego komponentu.

VHDL - Prosty przykład



VHDL - Prosty przykład

Synplify Pro®

Done: 0 errors, 0 warnings, 12 notes [Search SolvNet](#)

Project Files | Design Hierarchy

Microsemi Axcelerator : RTAX2000S : CQFP352 : -1

[synth_proj] - N:\weird_nand_lib\target\rtax2000s_weird

- VHDL
- Logic Constraints (SDC)
- results

Run

Open Project...
Close Project
Add File...
Change File...
Add Implementation...
Implementation Options...
Add P&R Implementation
View Log

Frequency (MHz):
1 | Auto Const.

Continue On Error ☐
FSM Compiler ☒
FSM Explorer ☐
Resource Sharing ☒
Retiming ☐

Project Status | Implementation Directory | Process View | Report

Project Settings

Project Name	synth_proj	Implementation Name	results
Top Module	weird_nand_lib.weird_nand	Retiming	0
Resource Sharing	1	Fanout Guide	24
Disable I/O Insertion	0	FSM Compiler	1

Run Status


Job Name	Status	Icon	Icon	CPU Time	Real Time	Memory	Date/Time
Compile Input Detailed report	Complete	5	0	0	0m:00s	-	02/12/2019 13:36:25
Premap Detailed report	Complete	3	0	0	0m:00s	106MB	02/12/2019 13:36:27
Map & Optimize Detailed report	Complete	7	0	0	0m:00s	107MB	02/12/2019 13:36:27

Area Summary

Combinational Cells	2	Sequential Cells	1
DSP Blocks	0	Clock Buffers	1
IO Cells	5	Block RAMs	0

[Detailed report](#)

EDA playground

 **playground**

Run

Copy

Share

Collaborate

?

Log In

Languages & Libraries

Testbench + Design

SystemVerilog/Verilog

UVM / OVM

UVM 1.2

Other Libraries

None

OV 2.8.1

SVUnit 2.11

Tools & Simulators

Riviera-PRO EDU 2014.10

Compile & Run Options

timescale 1ns/1ns-sv2k9

+UVM_VERBOSITY=UVM_MEDIUM

Run Time: 10 ms

☐ Use run.do Tcl file

☐ Open EPWave after run

☐ Download files after run

Details

UVM Hello World

The Fastest Way to Get Started with UVM

☒ Public

Examples

Created by Victor EDA

testbench.sv | my_testbench_pkg.svh | my_sequence.svh | my_driver.svh |

SV/Verilog Testbench

```
1 class my_driver extends uvm_driver #(my_transaction);
2
3   `uvm_component_utils(my_driver)
4
5   virtual dut_if dut_vif;
6
7   function new(string name, uvm_component parent);
8     super.new(name, parent);
9   endfunction
10
11   function void build_phase(uvm_phase phase);
12     // Get interface reference from config database
13     if(!uvm_config_db #(virtual dut_if)::get(this, "", "dut_vif", dut_vif)) begin
14       `uvm_error("", "uvm_config_db::get failed")
15     end
16   endfunction
17
18   task run_phase(uvm_phase phase);
19     // First toggle reset
20     dut_vif.reset = 1;
21     @(posedge dut_vif.clk);
22     #1;
23     dut_vif.reset = 0;
24
25     // Now drive normal traffic
26     forever begin
27       seq_item_port.get_next_item(req);
```

design.sv

```
1 // This is the SystemVerilog interface that we will use in our design
2 // our design to our UVM testbench.
3 interface dut_if;
4   logic clock, reset;
5   logic cmd;
6   logic [7:0] addr;
7   logic [7:0] data;
8 endinterface
9
10 `include "uvm_macros.svh"
11
12 // This is our design module.
13 //
14 // It is an empty design that simply prints a message whenever
15 // the clock toggles.
16 module dut(dut_if dif);
17   import uvm_pkg::*;
18   always @(posedge dif.clk)
19     if (dif.reset != 1) begin
20       `uvm_info("DUT",
21         $sformatf("Received cmd=%b, addr=0x%2h, data=0x%2h",
22           dif.cmd, dif.addr, dif.data), UVM_MEDIUM)
23     end
24 endmodule
```

KERNEL: Time: 0 ns, Iteration: 44, Instance: /package uvm_1.2.uvm_pkg/uvm_task_phase, Process: @INTERNAL#143_06.

KERNEL: UVM_WARNING /home/runner/my_testbench_pkg.svh(76) @ 10: uvm_test_top [] Hello World!

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 15: reporter [DUT] Received cmd=1, addr=0xbc, data=0x43

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 25: reporter [DUT] Received cmd=0, addr=0xf3, data=0x78

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 35: reporter [DUT] Received cmd=0, addr=0x60, data=0xe1

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 45: reporter [DUT] Received cmd=1, addr=0x03, data=0x7e

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 55: reporter [DUT] Received cmd=1, addr=0xdd, data=0x50

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 65: reporter [DUT] Received cmd=0, addr=0xee, data=0x57

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 75: reporter [DUT] Received cmd=0, addr=0x35, data=0x92

KERNEL: UVM_INFO /home/runner/design.sv(22) @ 85: reporter [DUT] Received cmd=1, addr=0xb2, data=0x01

KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_objection.svh(1271) @ 85: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' p

KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_report_server.svh(855) @ 85: reporter [UVM/REPORT/SERVER]

KERNEL: --- UVM Report Summary ---

KERNEL:

KERNEL: "" Report counts by severity

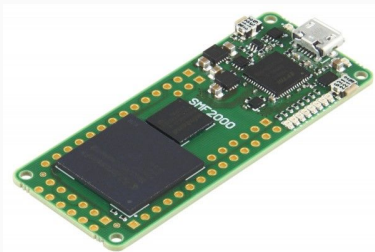
KERNEL: UVM_INFO : 11

KERNEL: UVM_WARNING : 1

Płyty deweloperskie



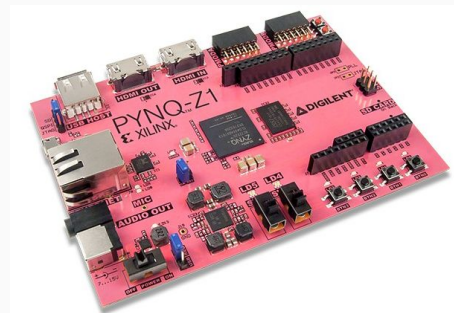
Digilent BASYS3 z Xilinx Artix-7



Trenz electronic SMF2000 z Microsemi Smartfusion2

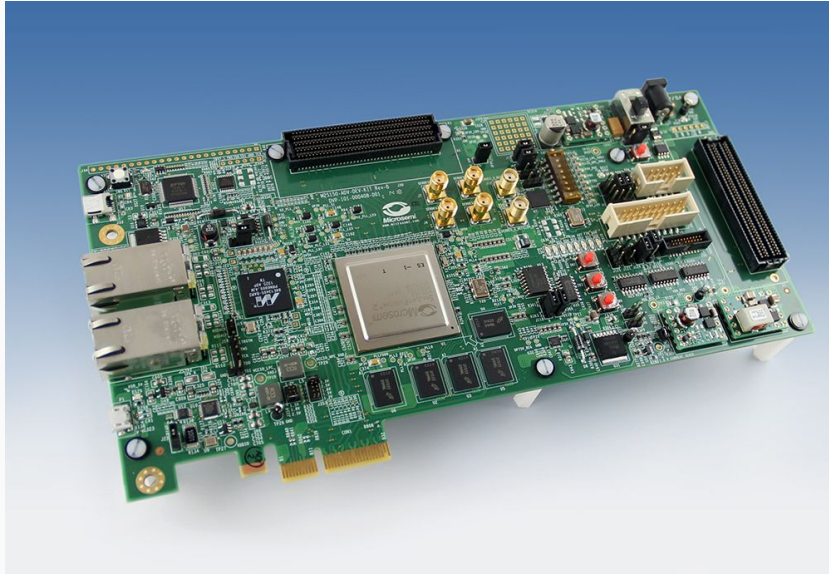


FUTUREM2SF-EVB z Microsemi SmartFusion2

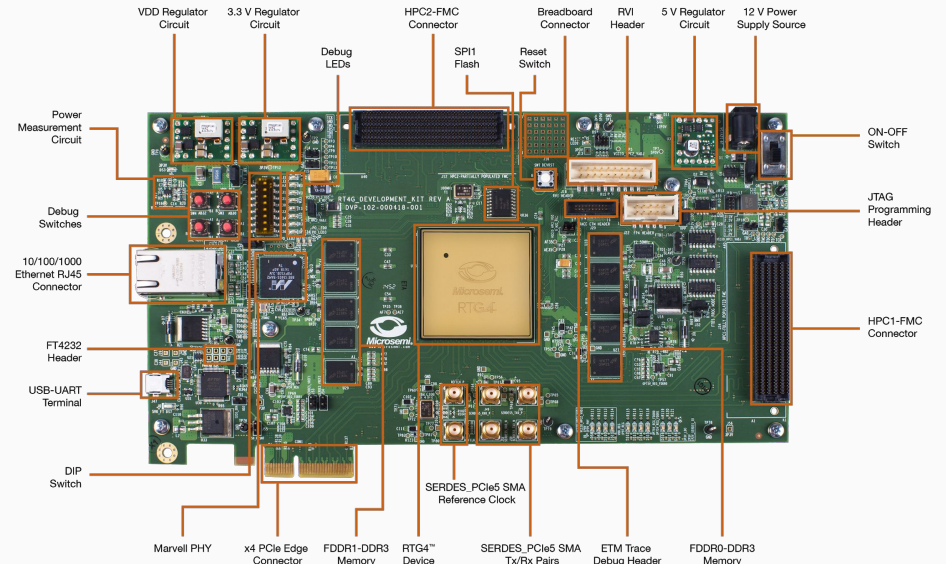


PYNQ-Z1 Python z Xilinx Zynq

Płytki deweloperskie



Microsemi SmartFusion2 Advanced Development Kit

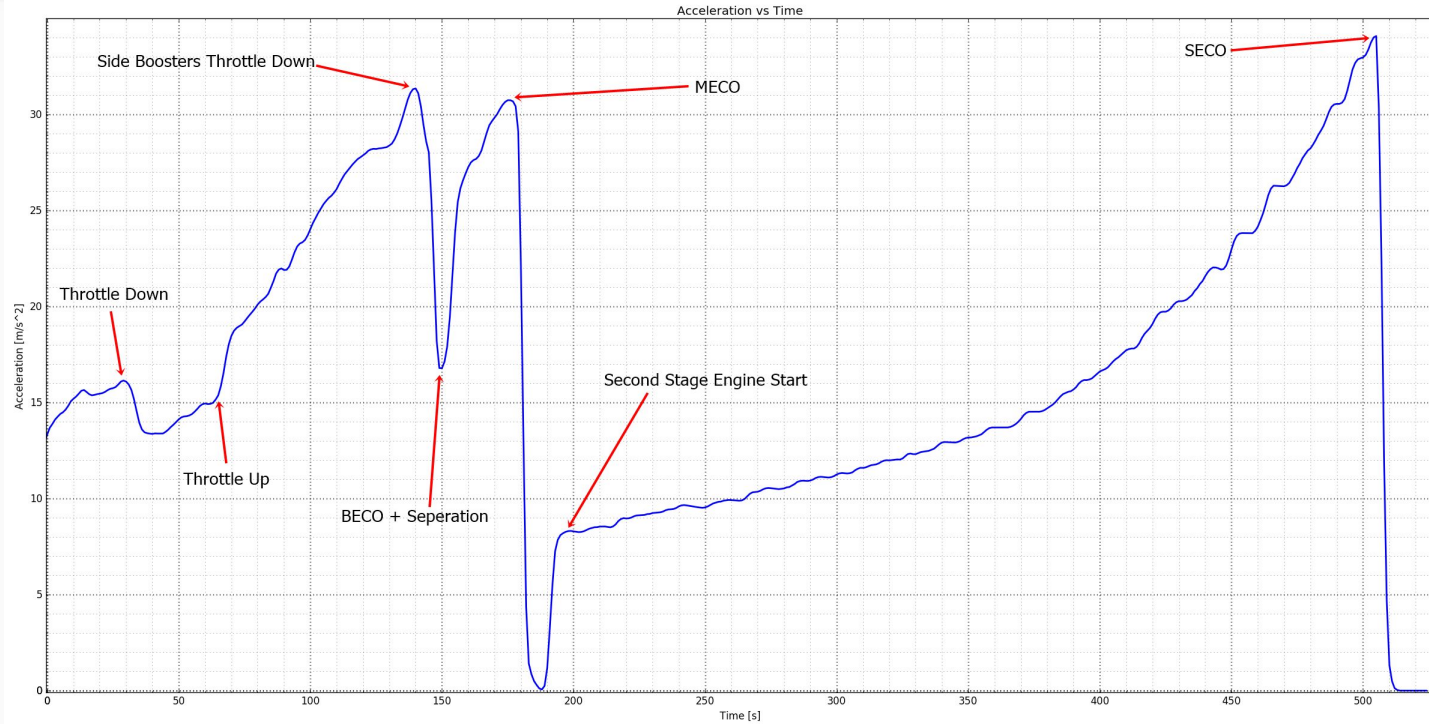


RTG4 Development Board with one RT4G150 FPGA

Elektronika w kosmosie - na co jest narażona?

W drodze na orbitę...

- Wibracje
- Przeciążenia



https://www.reddit.com/r/spacex/comments/7vtap9/falcon_heavy_test_flight_telemetry/

Na orbicie

- “Outgassing” - odgazowanie
- Różnice temperatur
 - Wpływ na parametry elektryczne komponentów
 - Naprężenia materiałów
- Odprowadzanie ciepła
- Promieniowanie
- Interferencje radiowe
- Ładowanie się (elektryczne) powierzchni statku
- Spadek wydajności paneli słonecznych
- Przebicie
- Inną elektronikę...



<https://www.epectec.com/pcb/wave-soldering-defects/outgassing.html>



<https://www.aascworld.com/wp-content/uploads/2017/10/Space-Thermal2.png>

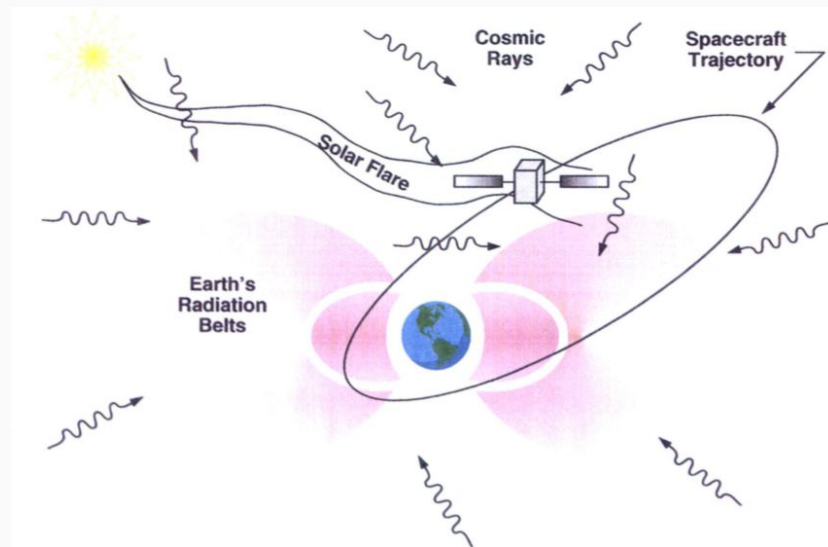
Promieniowanie w kosmosie

Pochodzenie:

- ze Słońca...
- spoza Układu Słonecznego.

Czym jest?

- Cząstki elementarne: głównie elektrony, protony i jony ciężkie.
- Promieniowanie wtórne:
 - Rentgenowskie powstające przez interakcje elektronów z ekranowaniem.
 - Cząstki powstałe w wyniku promieniowania jonizującego
 - Środowisko/otoczenie planet, np. dookoła Marsa, Ziemi
 - ...



<https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19990116210.pdf>

Promieniowanie jest specyficzne dla **każdej misji**.

Trzeba uwzględniać:

- Charakterystykę **orbity** i podróży w przestrzeni kosmicznej,
- **Czas** trwania misji,
- Możliwe do zastosowania **ekranowanie**,
- Dużą **dynamikę zmian promieniowania**, zależną od aktywności słonecznej
- dla każdej misji nadal pozostaje wiele niepewnych wartości.

Wszystkie informacje wymienione wyżej muszą być rozważone aby przygotować zakresy promieniowania dla projektu oraz wybrać i przetestować odpowiednie komponenty EEE.

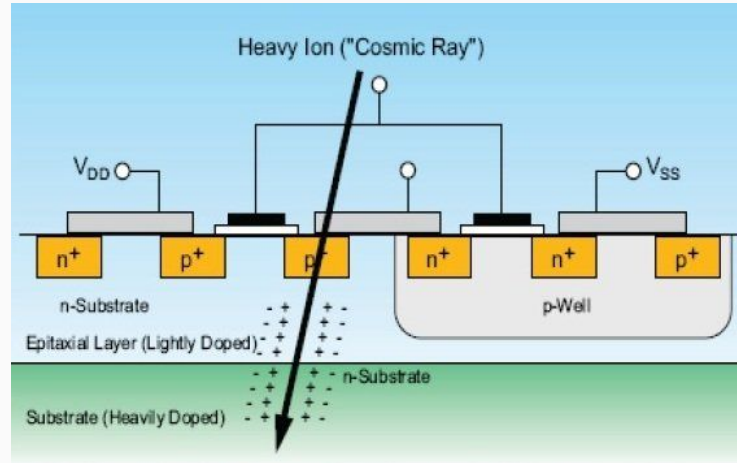
- Cumulative Effects
- Single Event Effects (SEE)
 - SEE **niedestrukcyjne** (*soft errors*; tymczasowe i można z nich wyjść):
 - Single Event Transient (SET)
 - Single Event Upset (SEU)
 - Single Event Failure Interrupt (SEFI)
 - SEE **destrukcyjne** (*hard errors*; mogą prowadzić do uszkodzenia):
 - Single Event Latchup (SEL)
 - Single Event Burnout (SEB)
 - Single Event Gate Rupture (SEGR)

Narażenie na pierwotne i wtórne promieniowanie powoduje stosunkowo **stałe, długotrwałe** zmiany w układach scalonych, np.:

- charakterystykę,
- degradację parametrów,
- ostatecznie prowadzi do utraty funkcjonalności komponentu.

Skutki promieniowania w FPGA - Single Event Effects (SEE)

- przyczyna: pojedyncza cząstka jonizująca
- może wywołać bardzo różne efekty.

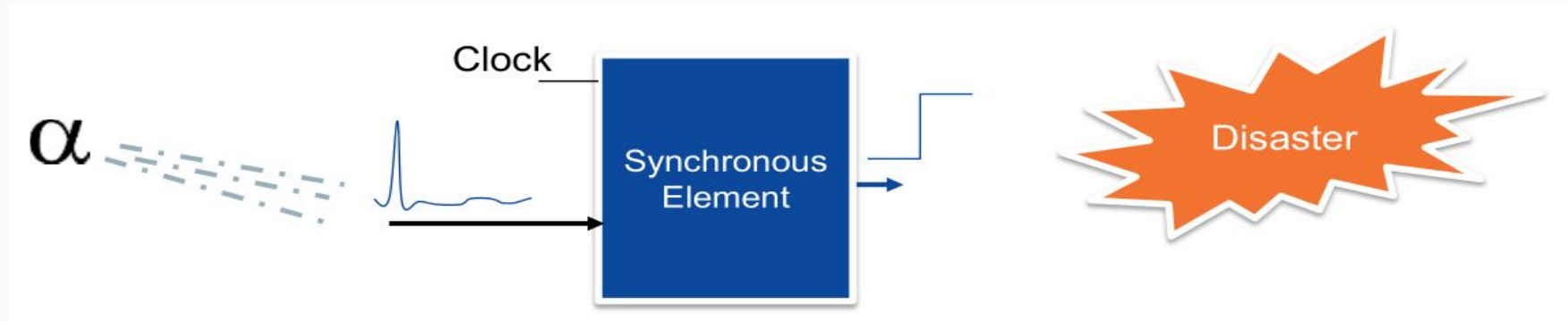


http://www.esa.int/ESA_Multimedia/Images/2012/12/Radiation-driven_Single_Event_Effect

Skutki promieniowania w FPGA - Single Event Transient (SET)

- Nie jest destrukcyjne,
- chwilowy “glitch” napięciowy wewnątrz układu scalonego,
- może nie być w ogóle “zauważony”,
- może prowadzić do SEU (jeśli trafi na element pamiętający), w najgorszym przypadku do SEFI,
- wrażliwość na SET rośnie z częstotliwością.

Skutki promieniowania w FPGA - Single Event Upset (SEU)



https://www.microsemi.com/document-portal/doc_download/132934-design-techniques-for-implementing-high-reliable-designs-using-microsemi-space-fpgas-russia-2013

- przekłamanie pojedynczego bitu w układzie pamiętającym, np. zmiana stanu przerzutnika, latching, komórki pamięci SRAM,
- rozwiązanie: nadpisanie błędnej wartości prawidłową.

Skutki promieniowania w FPGA - Single Event Failure Interrupt (SEFI)

- powoduje reset, zablokowanie lub inne problemy w komponentach
- zwykle pojawia się w złożonych układach z wbudowaną logiką kontroli, np. w pamięciach (SDRAM, DRAM, NOR/NAND Flash), procesorach,
- wyjście: reset lub power cycling,
- przykład z Syderal Polska: obsługa SEFI w NAND Flash.

Skutki promieniowania w FPGA - Single Event Latchup (SEL)

- Przyczyna: niezamierzone wyzwolenie obecnego w układach CMOS pasożytniczego tyrystora (PNPN lub NPNP).
- Skutek: zatrzaśnięcie się układu w stanie zwarcia. Wysoki prąd może, ale nie musi prowadzić do uszkodzenia układu z powodu nadmiernej temperatury.
- Wyjście: power-reset układu.

- TID
- LET / LET_{th}
- cross-section

TID (Total Ionising Dose):

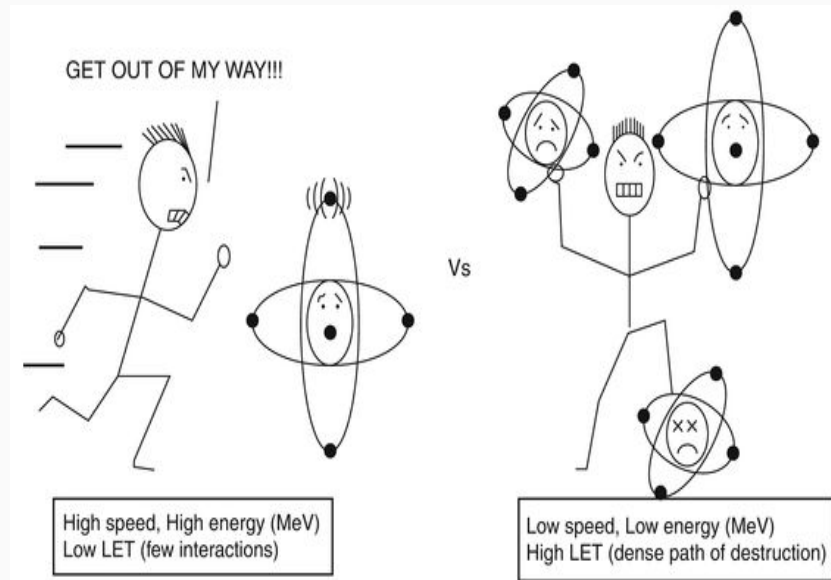
- Całkowita dawka promieniowania przyjęta przez układ w określonym czasie [rad - Radiation Absorbed Dose]
- W zależności od misji i orbity, typowy TID wynosi od kilku do kilkuset krad
- RTAX example: 300 krad

LET (Linear Energy Transfer)

- ilość energii promieniowania jonizującego absorbowaną na jednostkowej drodze [eV/cm]
- LET rośnie ze wzrostem ładunku
- LET maleje ze wzrostem prędkości cząstki (ma mniej czasu na interakcję z materiałem)

LETth (Linear Energy Transfer threshold)

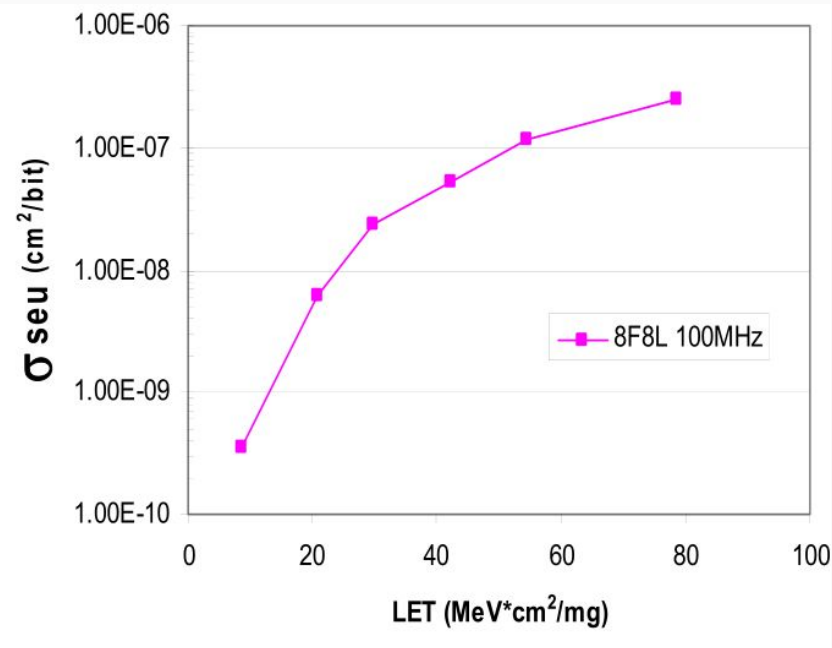
- LET znormalizowane do gęstości materiału docelowego [eV-cm²/mg]
- RTAX example: LETth (SEL) > 117 MeV-cm²/mg, LETth (SEU) > 37 MeV-cm²/mg



<https://radiologykey.com/interactions-of-particulate-radiation-with-matter/>

Cross section σ

- Fluencja cząstek: liczba cząstek przypadająca na powierzchnię [cm²]
- σ = liczba SEE / fluencja
- reprezentuje prawdopodobieństwo, że cząstka spowoduje wystąpienie SEE
- RTAX example: cross section (SEU) = 1E-9 cm².



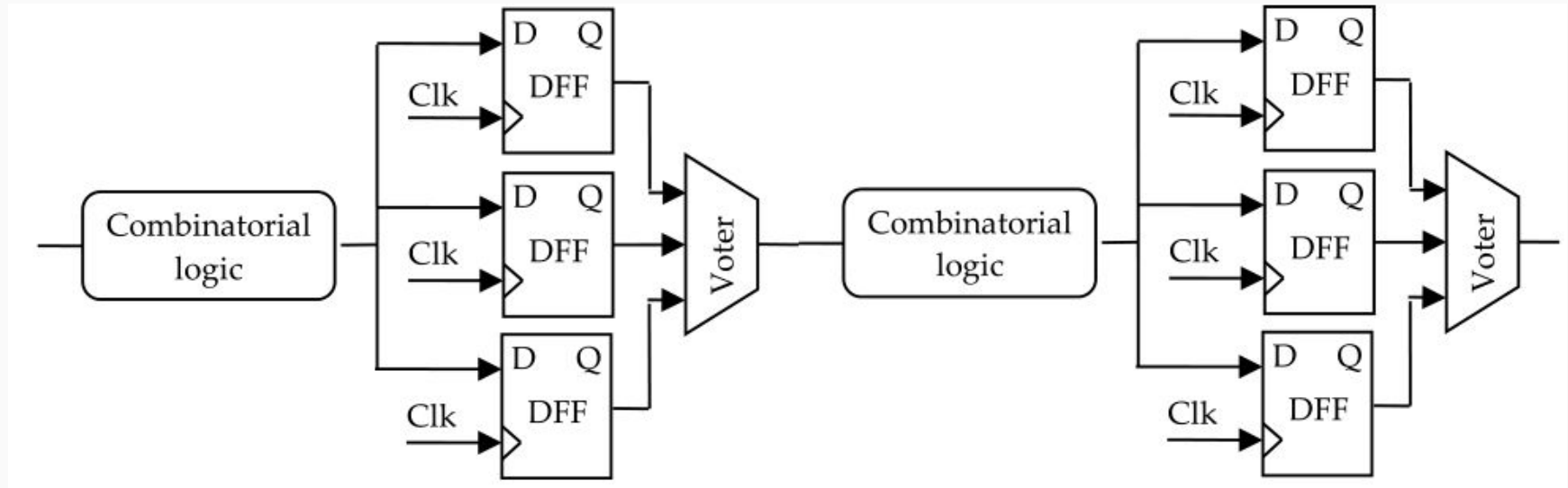
https://nepp.nasa.gov/mapld_2008/presentations/i/01%20-%20Berg_Melanie_mapld08_pres_1.pdf

Jak łagodzić skutki zakłóceń? - *mitigation techniques*

Mitigation techniques

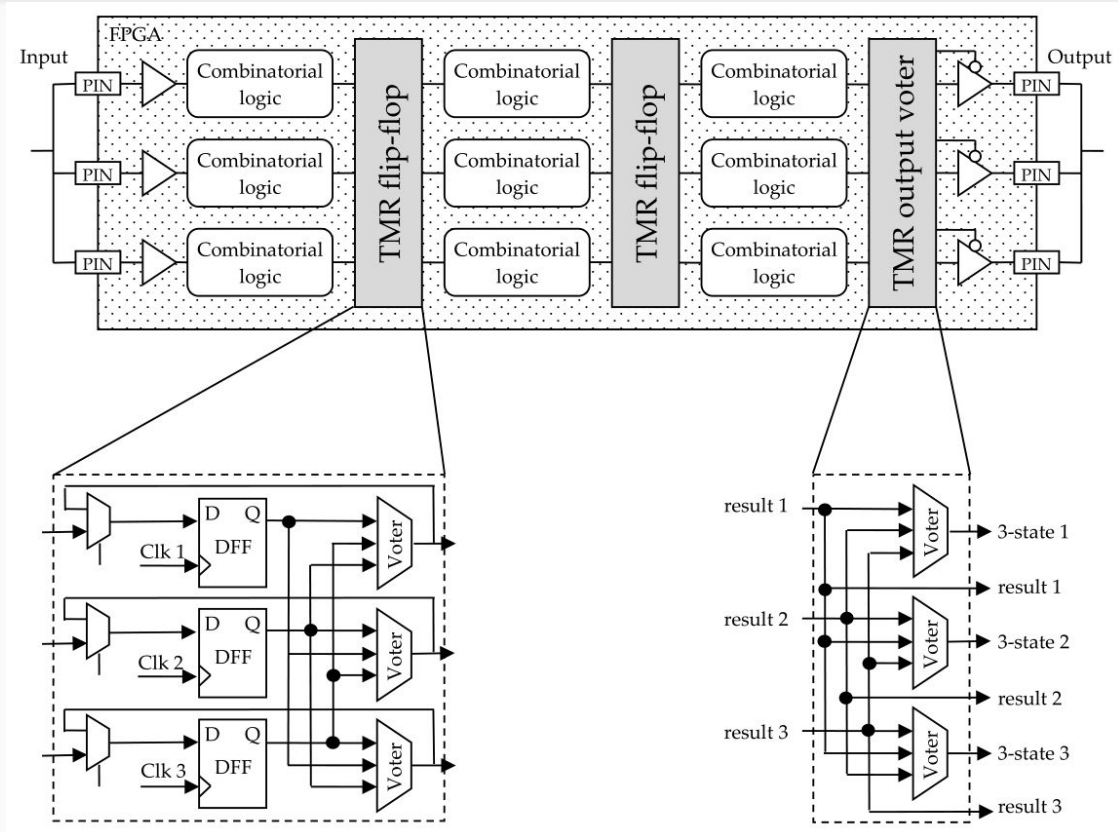
- Triple Modular Redundancy (TMR)
- Error Code Correction (ECC)
- Memory scrubbing
- FSM encoding
- ...

Local TMR: chroni przed SEU w rejestrach



Mitigation techniques - TMR

Global TMR:
chroni przed SET i SEU



- TMR sprzętowy (*native TMR*)
 - Microsemi: RTAX, RTG4; Xilinx: Virtex-4QV, Virtex-5QV
 - Wszystkie przerzutniki zawierają wbudowany TMR.
- TMR programowy
 - Odpowiedni atrybut dla syntezy tworzy dodatkową logikę automatycznie.
 - Więcej niż potrojenie potrzebnej logiki, pogorszenie timingów.
 - Przydaje się dla układów, które:
 - też zaliczają się do układów Radiation Tolerant
 - nie mają takiej odporności na promieniowanie, jak np. RTAX
 - nie mają sprzętowego TMR (RT ProASIC3).

Mitigation techniques - TMR

Device	Actel ProASIC3 RT3PE3000L	Actel Axcelerator RTAX 2000SL
Total Dose	>58.5kRad	>300kRad
SEL Immunity	> 68 MeVcm ² /mg	> 117 MeVcm ² /mg
Qualification Level	MIL-STD-883 Class E (Extended flow)	MIL-STD-883 class V QML Class V qualified
Native TMR (Triple module redundancy)	No Radiation mitigation required to meet specs	Yes. All instantiated flip-flops embed native TMR.

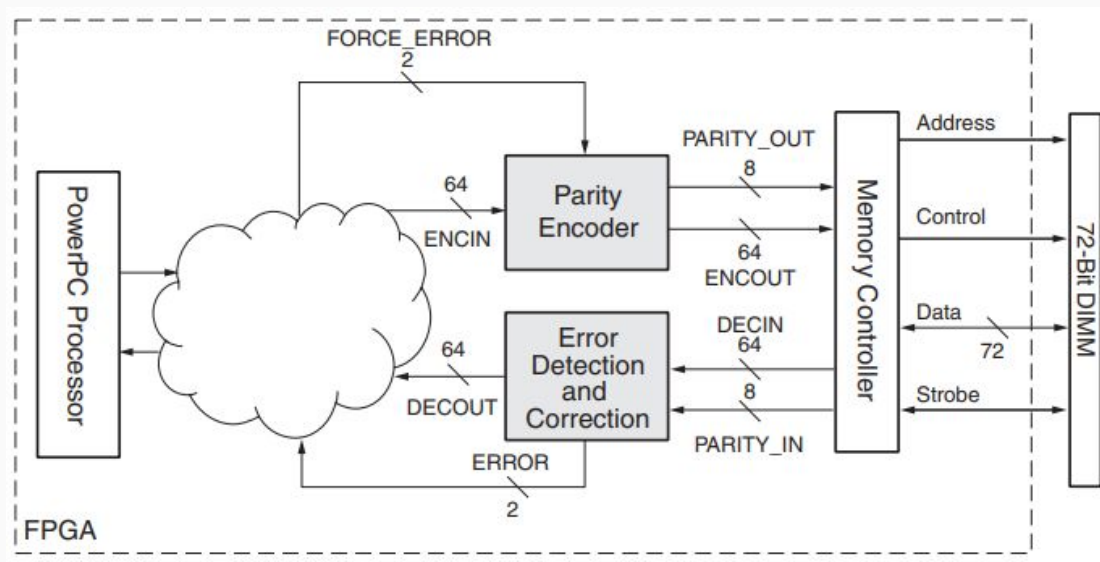
“Experience gained in Flash-based FPGA for InSight”, SYDERAL

Mitigation techniques - Error Correcting Codes

- Bit parzystości
- CRC (Cyclic Redundancy Check)
- EDAC (Error Detection and Correction)
 - Hamming codes
 - Reed-Solomon codes
 - BCH codes (Bose–Chaudhuri–Hocquenghem)

Mitigation techniques - Hamming code

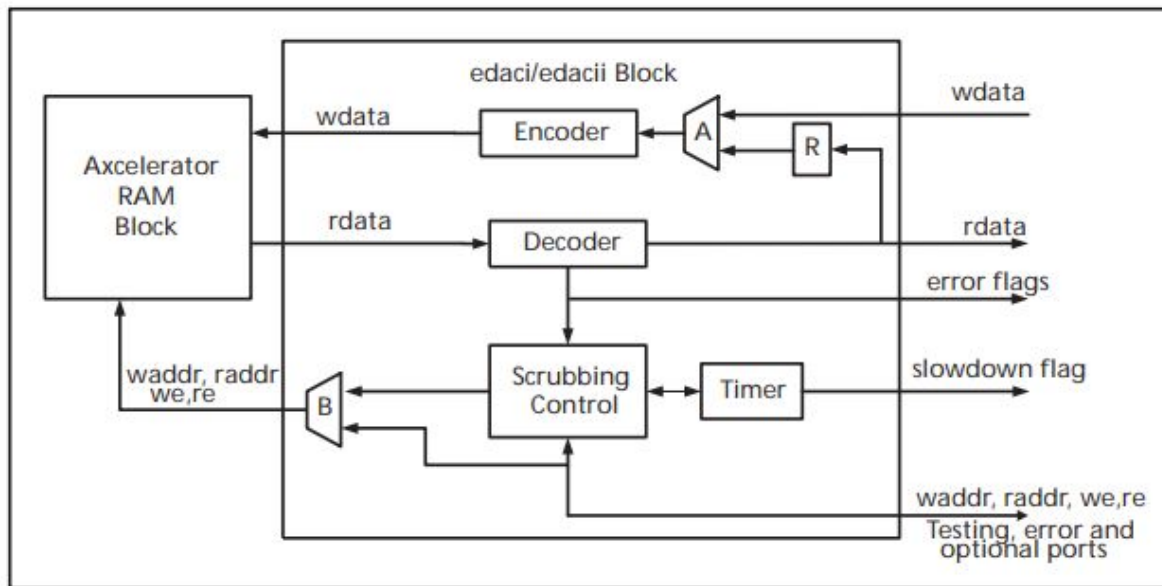
- Dokłada dodatkowe bity korekcji do bitów danych.
- Koryguje pojedynczy błąd, podwójny błąd jedynie wykrywa.
- Algorytm - nieskomplikowany (XOR-s).



https://www.xilinx.com/support/documentation/application_notes/xapp645.pdf

Mitigation techniques - memory scrubbing

- Przeciwdziała kumulowaniu się przekłamanych bitów w pamięci
- Cykliczny: odczyt -> wykrycie przekłamania (EDAC) -> ponowne nadpisanie komórki pamięci właściwą wartością.



- Kodowanie **one-hot**: nie wystarcza (*others* clause optimized)
- Kodowanie **one-hot + safe**: dodatkowa logika do automatycznego resetu FSM, gdy z powodu zakłócenia pojawia się stan niezdefiniowany
- **Hamming-3 Error Correction/Detection**: automatyczna korekcja pojedynczego błędu w FSM, który pracuje dalej normalnie.

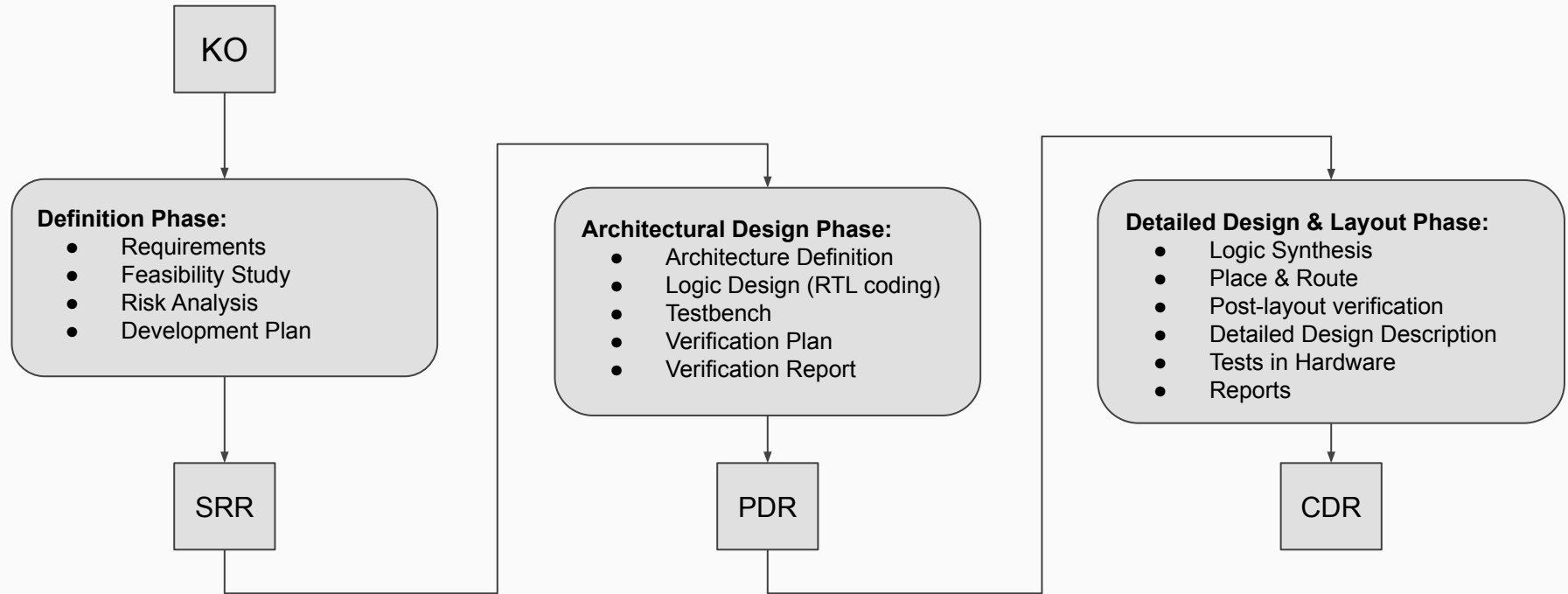
Najważniejsze parametry na przykładzie Microsemi RTG4

- Configuration memory upsets immunity to LET > 103 MeV.cm²/mg
- Single-event latch-up (SEL) immunity to LET > 103 MeV.cm²/mg
- SEU-hardened registers eliminate the need for triple-module redundancy (TMR)
 - Immune to single-event upsets (SEU) to LET > 37 MeV.cm²/mg
 - SEU rate < 10⁻¹² errors/bit-day (GEO Solar Min)
- SRAM has a built-in error detection and correction (EDAC)
 - Upset rate < 10⁻¹¹ errors/bit-day (GEO Solar Min)
 - Single error correction and double error detection (SECDED)
- Single-event transient (SET) upset rate < 10⁻⁸ errors/bit-day (GEO Solar Min) with optional SET filter
- Total ionizing dose (TID) > 100 krad

<https://www.microsemi.com/product-directory/rad-tolerant-fpgas/3576-rtg4>

- Zamiast komponentów *rad-hard* / *radiation tolerant* używamy komponentów COTS
- Implementujemy w nich *mitigation techniques*
- W przestrzeń kosmiczną wysyłanych jest więcej egzemplarzy takiej elektroniki - redundancja na poziomie całego urządzenia a nie modułu/instrumentu.

Space applications - FPGA Design Flow



BONUS

- **Instrument Control Unit for the FLORIS, FLEX mission**
 - PCB design for the Power Supply and Driver modules.
 - FPGA modules development and verification.
 - Development of Packet Utilisation Standard (PUS) handling software.
- **Motor Controller Demonstrator (MCD), PLIIS programme**
 - Design of a module controlling two stepper motors based on SpaceWire RMAP interface communication. Prime contractor for the project.
- **Reaction Wheel with Local Speed Control, ESA CTP**
 - Improvement of torque stability for the future ARIEL mission needs.
 - Subcontractor to Bradford for Electronics hardware and FPGA development



Credits: ESA

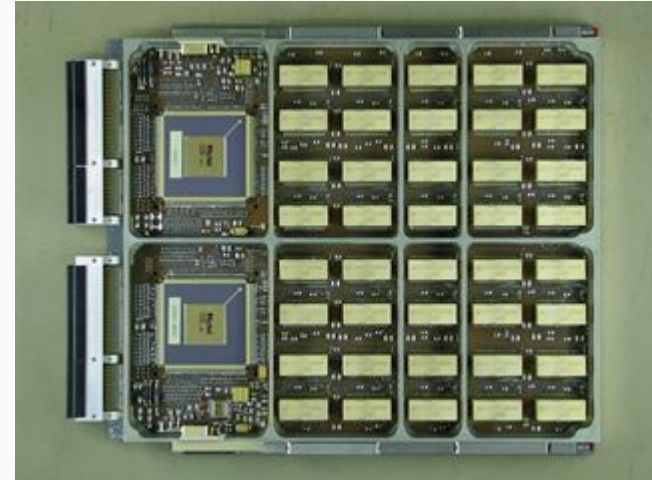


MCD Breadboard

Flash Memory Module (FMM)

Development of flash based non-volatile memory module. Project concentrates on the FPGA development and will follow the ECSS FPGA Design Flow up to the CDR phase.

- Development of an architecture which will be scalable in terms of memory size.
- Flash memory controller implementation and test.
- Data protection implementation and test.



GAIA SDRAM-based Mass Memory Module
Source: SYDERAL Swiss

Self-calibrating electronic controller for satellite quantum entanglement source

- The first step towards ensuring constant high-quality generation of entangled photon pairs for satellite QKD missions of the future.
- Funded by the National Centre for Research and Development
- Realisation period: September 2019 – August 2022.
- Partners: University of Gdansk, Nicolaus Copernicus University Torun



Example of satellite QKD concept.
Credits: <http://www.2physics.com>