Hardware Implementation of Automatic Control Systems using FPGAs

Lecturer PhD Eng. Ionel BOSTAN
Lecturer PhD Eng. Florin-Marian BÎRLEANU
University of Pitesti
Romania

Disclaimer:
This presentation tries to show the current trends in the hardware implementation of the PID controller using FPGA circuits and does not contain the authors' research work.

Overview

The aims of this group of educational activities are focused on three directions:

- Presentation of FPGAs as a support for hardware implementation of digital systems, such as high speed and high performance control systems that can be useful in renewable energy systems;
- FPGA project implementation – all steps we need in order to design, implement and test a complete application in FPGA;
- Applications: examples of hardware implementation of PID controllers using different types of implementation techniques:
    - HDL implementation;
    - Implementation using System Generator
    - Implementation using High Level Synthesis Tools

I. Introduction to Automatic Control Systems

I.1. Purpose of ACS;

I.2. Typical structure of ACS;

I.3. PID Controller ;

Software Implementation

Hardware Implementation in FPGA

FPGA Implementation Difficulties

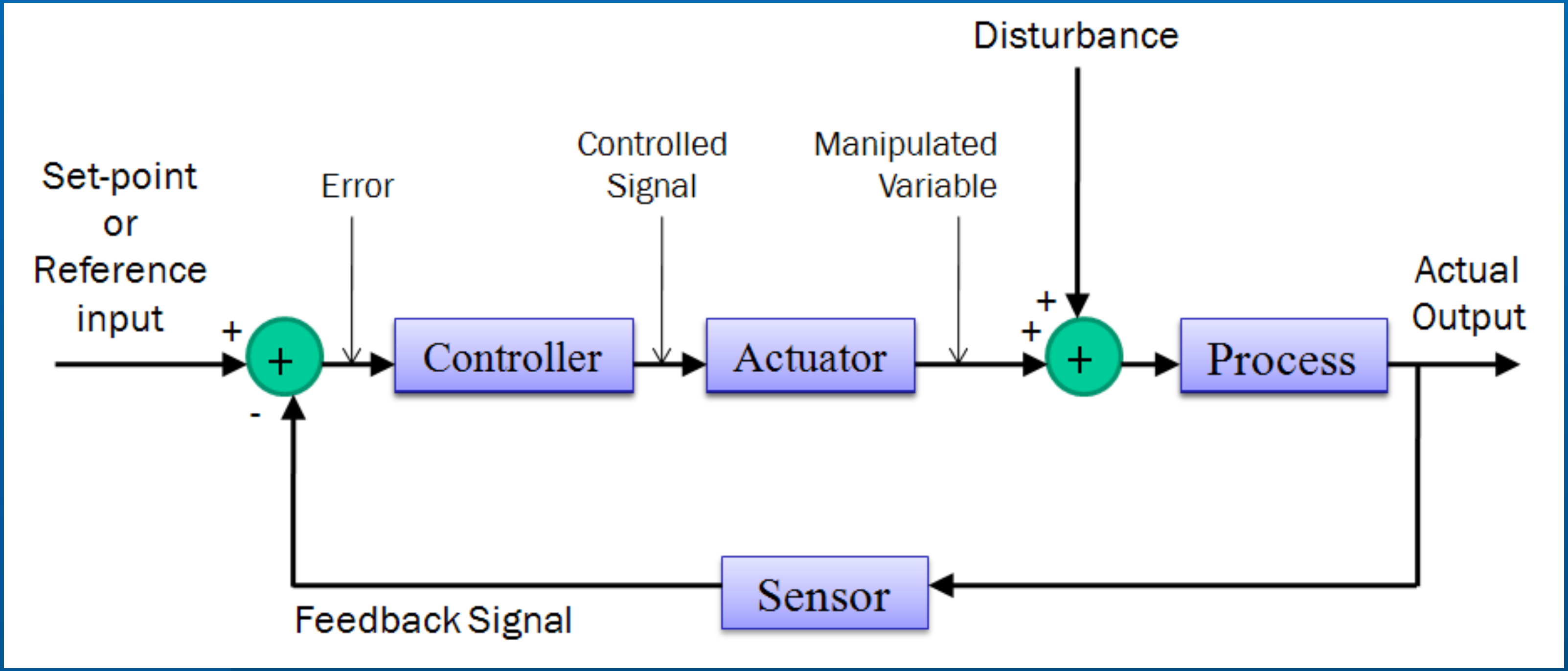
I.1. Purpose of Automatic Control Systems (ACS)

Purpose of Automatic Control Systems [1]:

- **Power amplification (gain)**
  - Positioning a large radar antenna by low-power rotation of a knob;
  - Opening and closing valves;
- **Remote control**
  - Robot arm used to pick up radioactive material;
  - Unmanned Aerial Vehicles;
  - Remote Terminal Unit in oil production;
- **Convenience of input form**
  - Changing room temperature by thermostat position;
  - Quality Control using limit switch;
- **Compensation for disturbances**
  - Controlling antenna position in the presence of large wind disturbance torque;
  - Control Inventory under variable demand;

Part I. Introduction to Automatic Control Systems

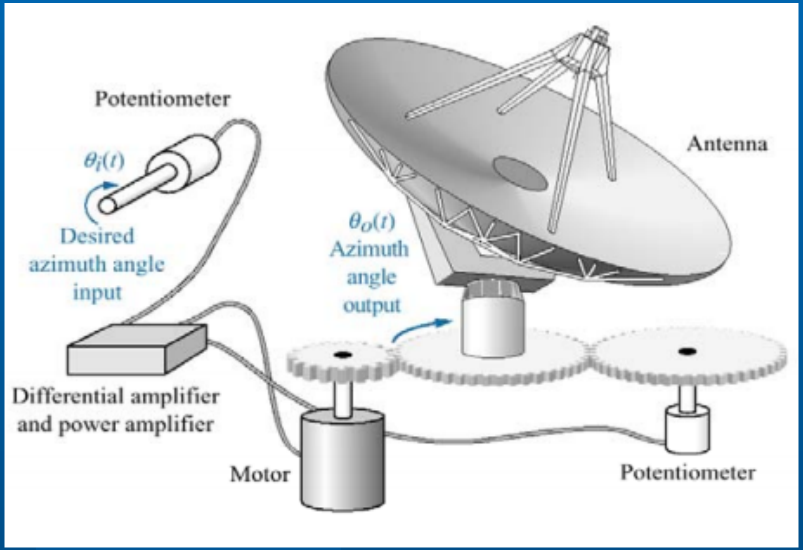
● *I.2. Typical Structure of Automatic Control Systems*

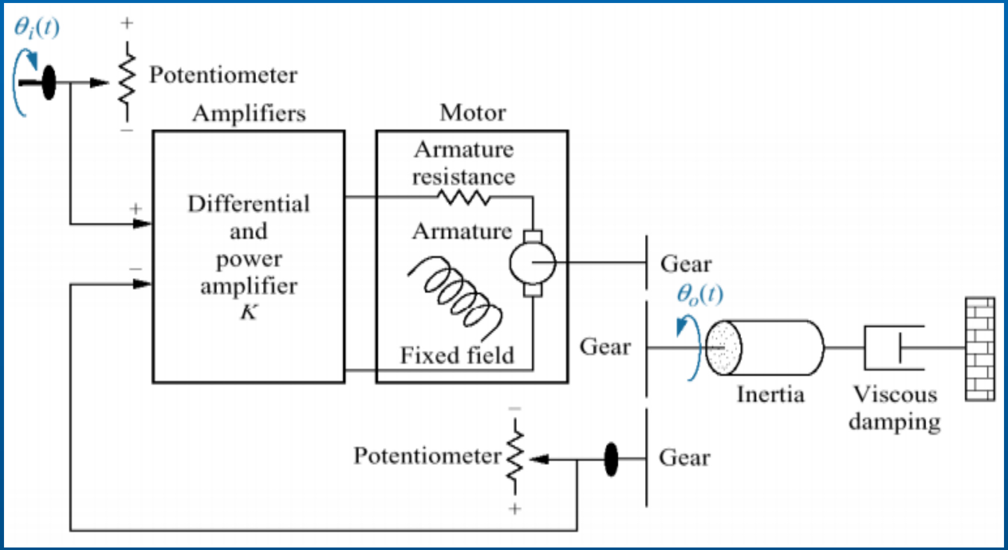


Part I. Introduction to Automatic Control Systems

• *I.2. Typical Structure of Automatic Control Systems*

Case Study – Antenna Azimuth Position Control System [1]

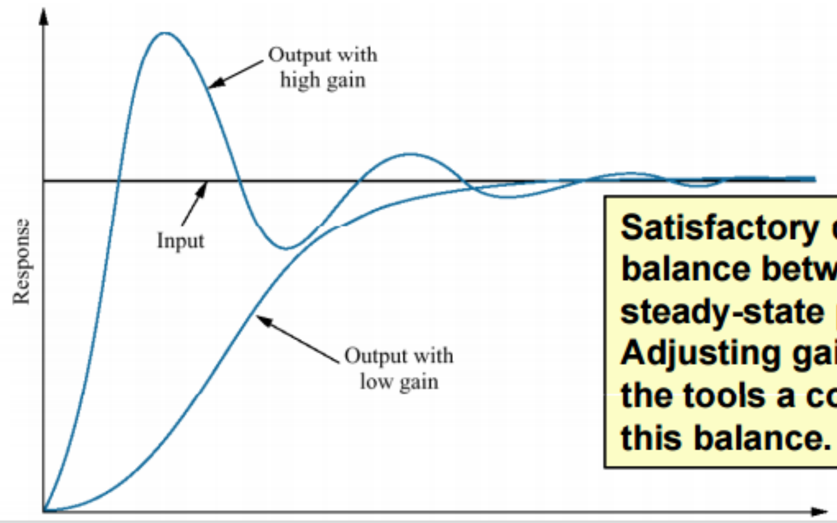




● *I.2. Typical Structure of Automatic Control Systems*

Case Study – Antenna Azimuth Position Control System [1]

- System normally operates to drive pointing error to zero.
- Motor is driven only when there is a pointing error.
- The larger the error the faster the motor turns.
- Too large a signal amplifier gain could cause overshoot/instability.



Satisfactory design revolves around a balance between transient performance, steady-state performance, and stability. Adjusting gain & adding compensators are the tools a control engineer has to achieve this balance.

• *I.3. PID Controller*

Proportional -*I*ntegral - *D*erivative Controller (*PID Controller*)

- most used algorithm in industrial control;
- can be software or hardware implemented;
- has tuning difficulties;

Role of each controller term:

- Proportional term – used to drive the controller output according to the size of the error;
- Integral term - used to eliminate the steady-state offset;
- Derivative term – used to evaluate the trend to correct the output and improve the overall stability by limiting the overshoots;

In many industrial application one control loop is not enough, for example in motor control:

- Torque is controlled by current loop PID;
- Speed is managed by the velocity PID cascaded with the current PID;
- Position is managed by the space PID cascaded with the velocity PID;

● *I.3. PID Controller – Software Implementation*

Complex control systems:

- More PID control loops are needed;
- Sequential execution of each PID loop implemented in software => increased time delay between input and output;
- In some cases, software implementation of ACS cannot meet the performance specifications;
- In some cases, microprocessor/microcontroller can be replaced with DSP;

In recent years Network Control Systems are very often used. In these cases many computational resources need to be allocated for communications purposes, which can reduce the performance.

● *I.3. PID Controller – Hardware implementation in FPGA*

Hardware vs. Software

- First PID controllers were implemented in hardware, using operational amplifiers;
- After microprocessor/microcontroller boom, PID controllers begun to be implemented in software;
- Software implementations => *sequential execution* => time delay => low performance in cases of complex systems, with many control loops. ***Software become unattractive!***

FPGA technology has some attractive advantages for hardware implementation of PID loops:

- FPGA allows multiple instances of PID controllers to operate concurrently, due to massive parallel resource available on chip ;
- Adding new PID controllers can be done without affecting the performance of existing controllers;
- FPGA are programmable (reconfigurable) and can be updated as easy as any microcontroller;
- In modern circuits such as Xilinx Zynq-7000 All Programmable SoC, additional advantages appears: ***Hardware become attractive again!***
  - Typical advantages of FPGA;
  - Typical advantages of ARM Cortex A9 CPU;
  - Costs & Power advantage over microcontrollers implementations;

I.3. PID Controller – FPGA implementation difficulties

– FPGAs offer great advantage for hardware implementation of the control systems;

– In order to use an FPGA, any controller must be designed and implemented using a Register Transfer Level (RTL) language such as VHDL or Verilog;

– RTL implementation is difficult for engineers with no background in hardware design;

– In present, using High-Level Synthesis tools and System Generator for DSP, any control engineer can design any controller by only needing to understand:
   – basic resources on an FPGA;
   – standard hardware I/O protocols;

– Using HLS tools and System Generator for DSP we can: implement, optimize, analyze and verify the design on an FPGA;

– Transformation from C/C++ specification to RTL requires no more adaptations than it was needed from C/C++ to DSP implementation.

Hardware implementation become equal to software implementation !

II. FPGA

II.1. FPGA architecture;

II.2. Hardware Description Language (HDL) Design Flow;

II.3. Hardware implementation using System Generator;

II.4. Hardware implementation using High Level Synthesis tools (Vivado)

II.5. Design Verification

● *Why FPGA?*

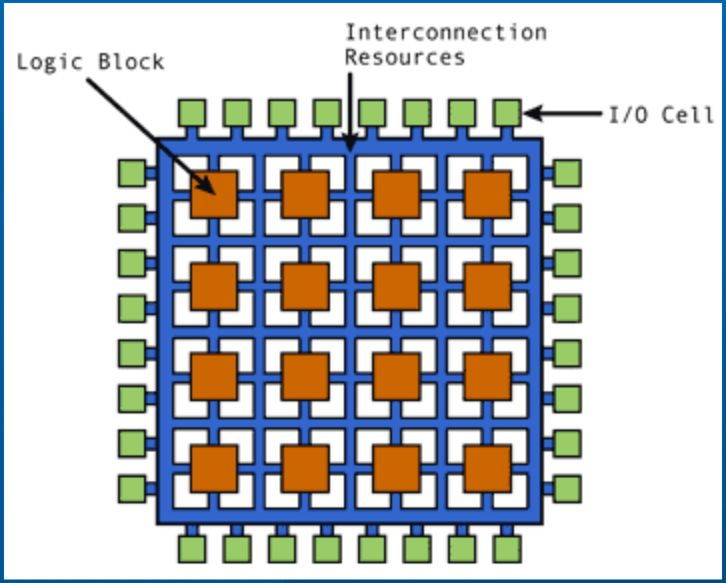
In present a great number of numerical systems are implemented with FPGA circuits due to their advantages over the general purpose logic IC:

- **completely reconfigurable** – different projects can be implemented on the same circuit at different times;
- **low cost** due to the mass production;
- **easy to use** – mature high level design tools are available;
- **an immense number of digital circuits** which are connected by the end user;
- **can be used for high speed systems** due to their parallel architecture;

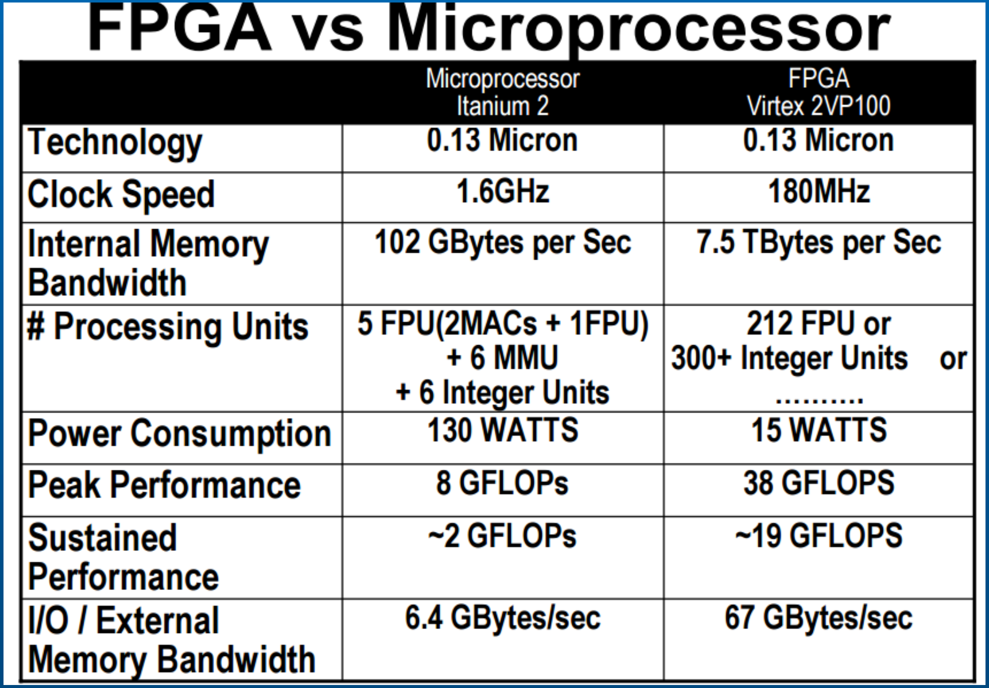
For any future specialist in electronics it is very important to be able to use these circuits in different designs.

● *II.1. FPGA Architecture*

Typical Architecture



FPGA advantage over a microprocessor



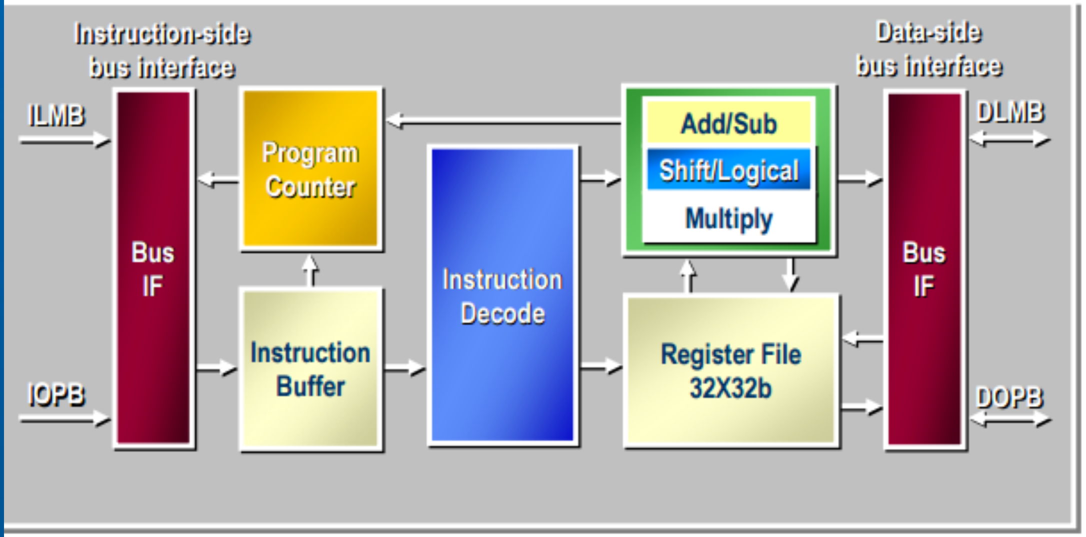
FPGA vs Microprocessor

	Microprocessor Itanium 2	FPGA Virtex 2VP100
Technology	0.13 Micron	0.13 Micron
Clock Speed	1.6GHz	180MHz
Internal Memory Bandwidth	102 GBytes per Sec	7.5 TBytes per Sec
# Processing Units	5 FPU(2MACs + 1FPU) + 6 MMU + 6 Integer Units	212 FPU or 300+ Integer Units or ....
Power Consumption	130 WATTS	15 WATTS
Peak Performance	8 GFLOPs	38 GFLOPS
Sustained Performance	~2 GFLOPs	~19 GFLOPS
I/O / External Memory Bandwidth	6.4 GBytes/sec	67 GBytes/sec

II.1. FPGA Architecture

Software microprocessor from Xilinx [2]

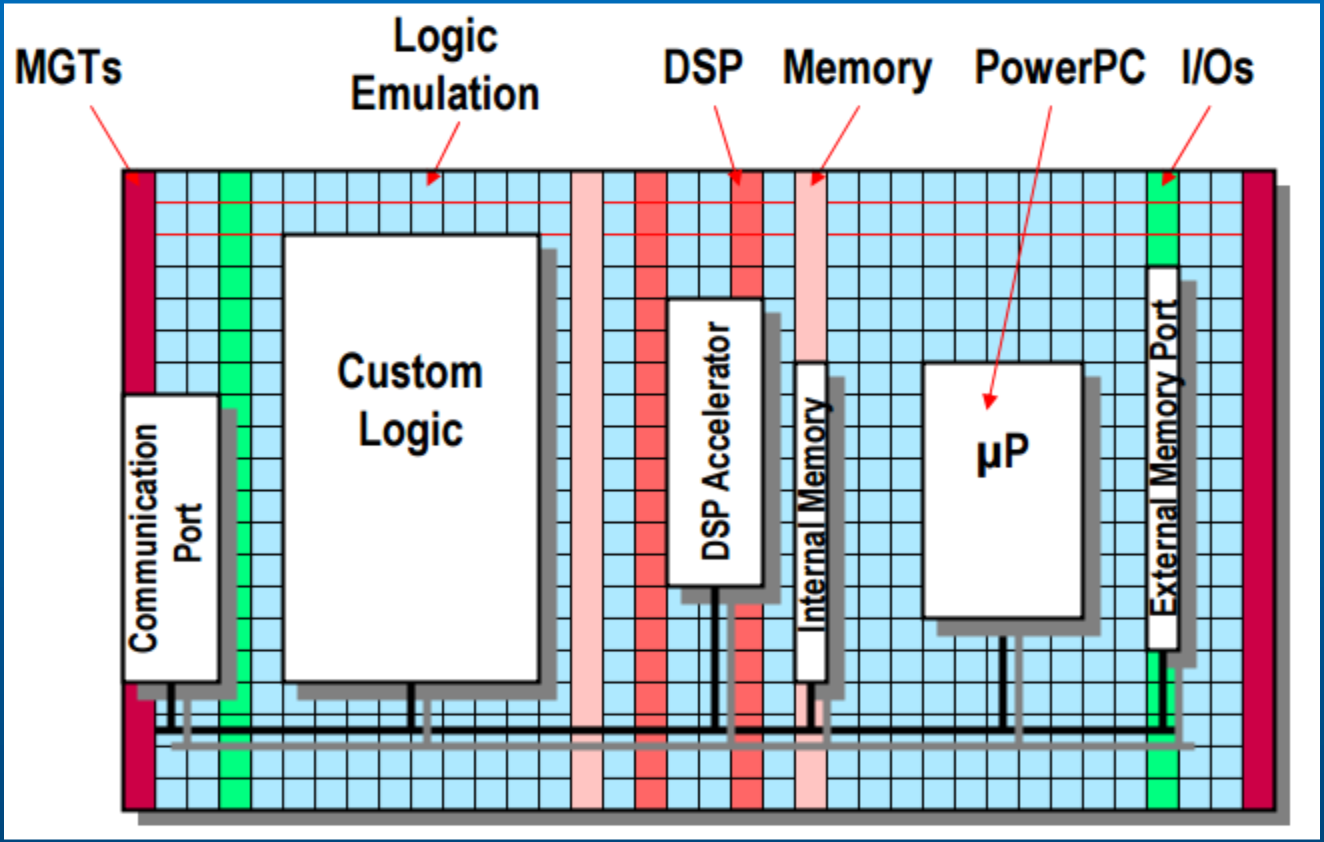
MicroBlaze 5.0



- 1400 LUT6
- 230 Dhrystone Mips
- > 200 fit in V5

● *II.1. FPGA Architecture*

System FPGA



● *II.1. FPGA Architecture*

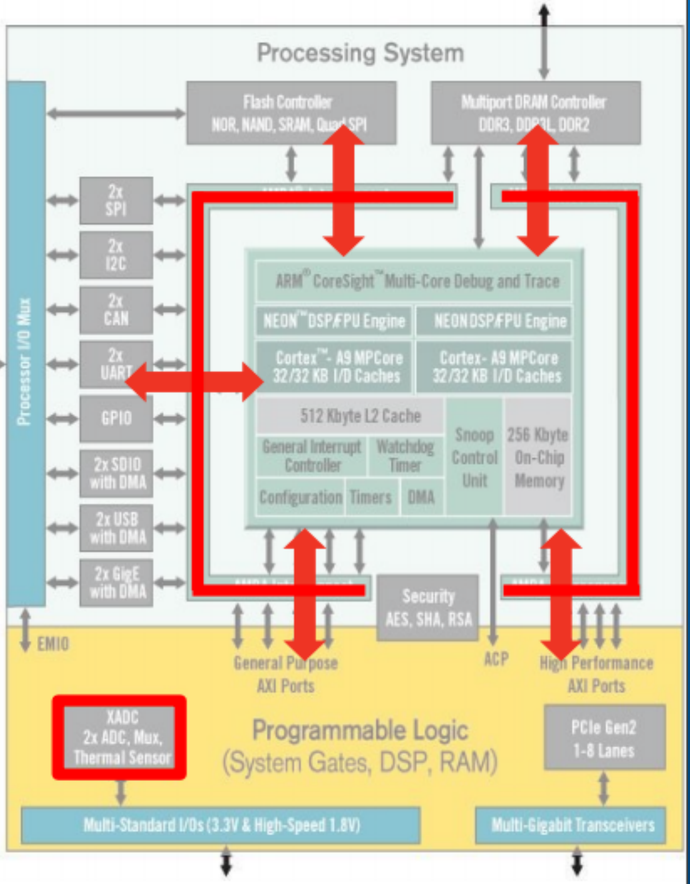
System on Chip

(SoC)

[3]

Zynq All-Programmable SoC

- **Processor System (PS)**
  - 2x ARM9 866MHz-1GHz 32K/32K I/D Caches
  - 512KB shared L2 Cache
  - 256KB On-chip memory
  - Memory controller
  - Bus interfaces, timers
  - Libraries, OSs, middleware
- **Programmable Logic (PL)**
  - 28K – 440K LCs
  - 240K – 3MB RAM
  - 80 – 2020 DSP blocks
  - I/O, Transceivers, PCIe, Ethernet…
- **Programmable ADC**
  - Inputs from Voltage, Temp sensors
- **AMBA AXI bus fabric**


Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
Security AES, SHA, RSA
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

● *II.2. HDL Design Flow [4]*

Hardware

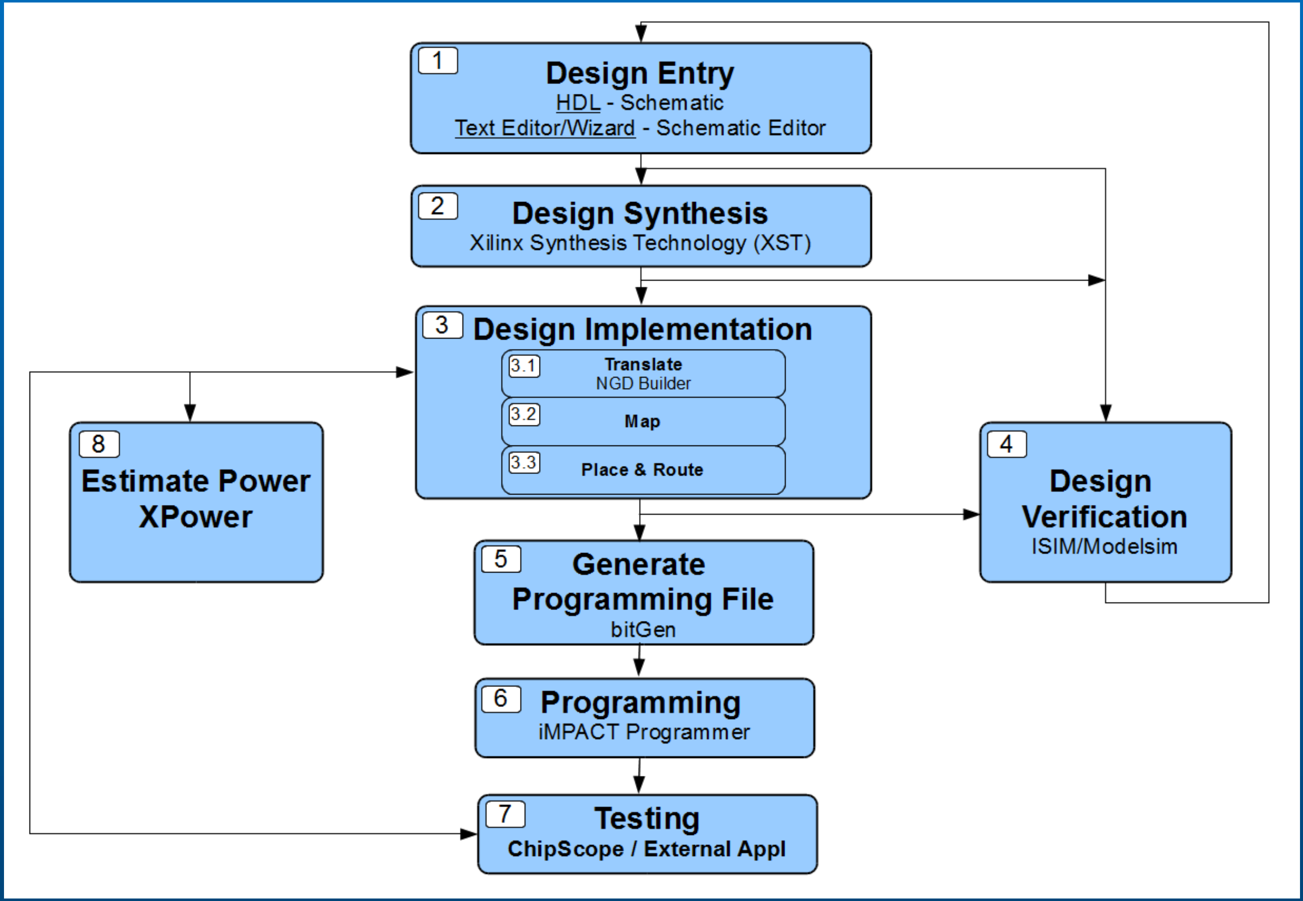
Description

Language

(HDL)

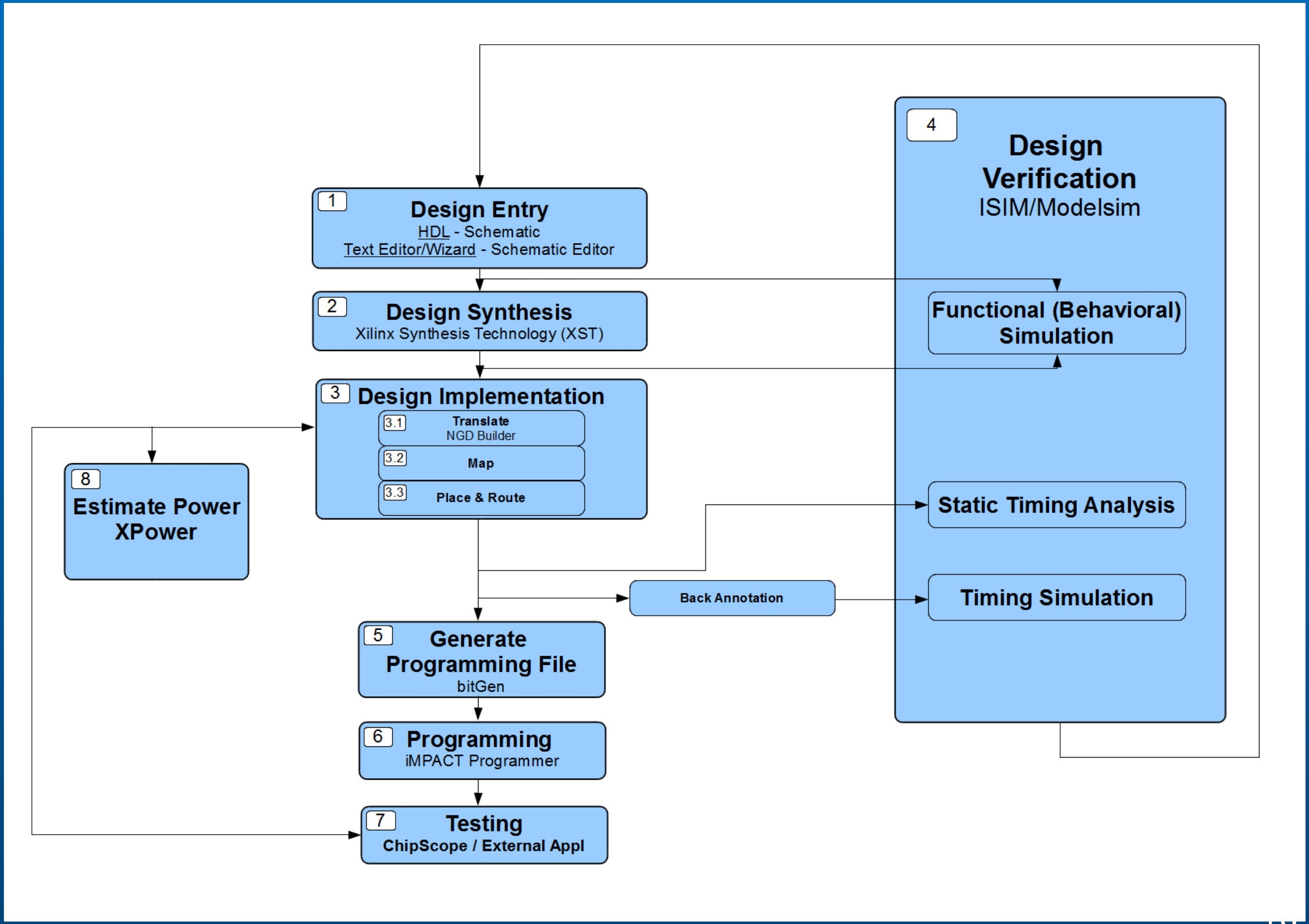
Example:

- ***VHDL;***
- ***Verilog;***



II.2. HDL Design Flow [4]

Verification



● *II.3. Hardware implementation using System Generator*

System Generator (SysGen) is [5], [6]**:**

- a system-level modeling tool that facilitates hardware design on FPGA;
- it extends Simulink in many ways to provide a modeling environment that is well suited to hardware design;
- the tool provides high-level abstractions that are automatically compiled into an FPGA at the push of a button;
- System Generator does not replace hardware description language (HDL)-based design, but does makes it possible to focus your attention only on the critical parts;
- Critical parts of the design can be made in HDL and less critical parts can be made using SysGen, and then the HDL and SysGen parts can be connected;

By analogy, most DSP programmers do not program exclusively in assembler; they start in a higher level language like C, and write assembly code only where it is required in order to meet performance requirements.

II.3. Hardware implementation using System Generator

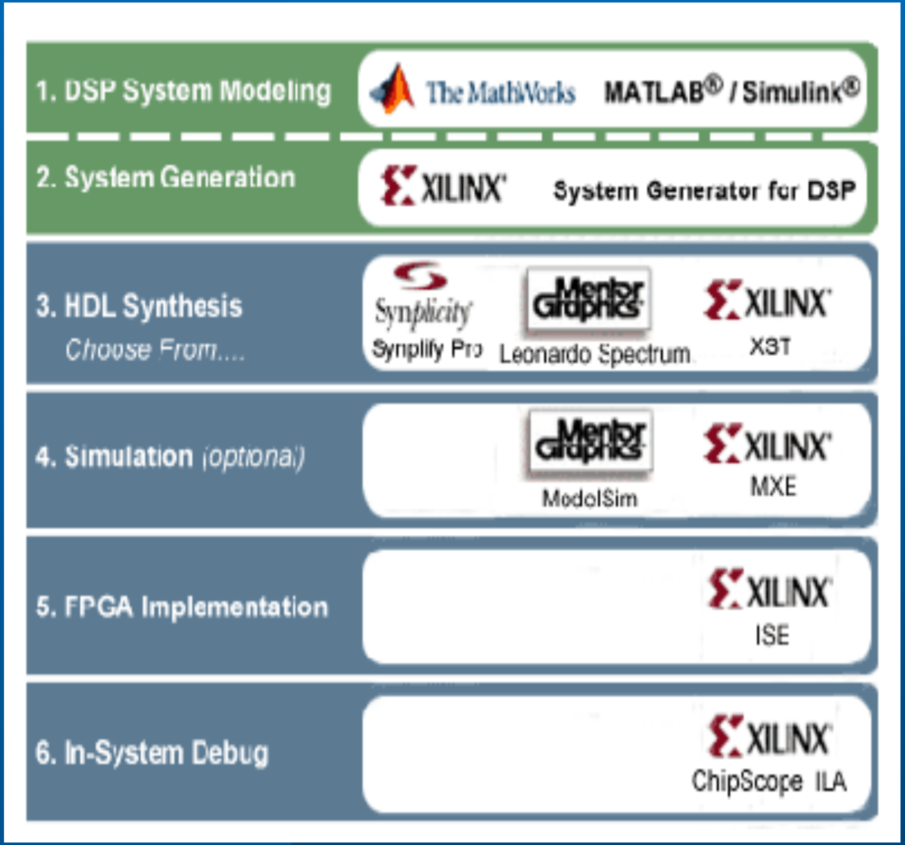
Where to use System Generator (SysGen) [5], [6]:

- Algorithm Exploration: System Generator is particularly useful for algorithm exploration, design prototyping, and model analysis. When these are the goals, you can use the tool to flesh out an algorithm in order to get a feel for the design problems that are likely to be faced, and perhaps to estimate the cost and performance of an implementation in hardware. The work is preparatory, and there is little need to translate the design into hardware. Simulink blocks and MATLAB M-code provide stimuli for simulations, and for analyzing results. Resource estimation gives a rough idea of the cost of the design in hardware.
- Implement a Part of a Large Design: Often System Generator is used to implement a portion of a larger design. For example, System Generator is a good setting in which to implement data paths and control, but is less well suited for sophisticated external interfaces that have strict timing requirements. In this case, it may be useful to implement parts of the design using System Generator, implement other parts outside, and then combine the parts into a working whole. A typical approach to this flow is to create an HDL wrapper that represents the entire design, and to use the System Generator portion as a component. The non-System Generator portions of the design can also be components in the wrapper, or can be instantiated directly in the wrapper.
- Implement a Complete Design: Many times, everything needed for a design is available inside System Generator. For such a design, pressing the Generate button instructs System Generator to translate the design into HDL, and to write the files needed to process the HDL using downstream tools.

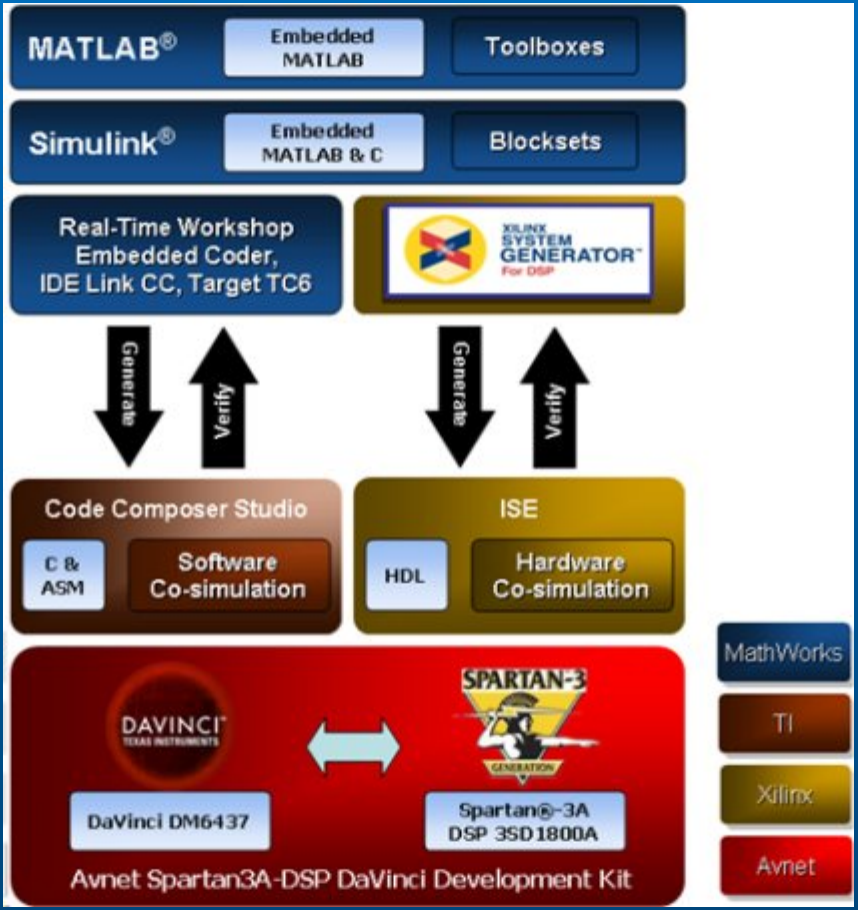
Part II. FPGA

- *II.3. Hardware implementation using System Generator*

Design Flow using System Generator

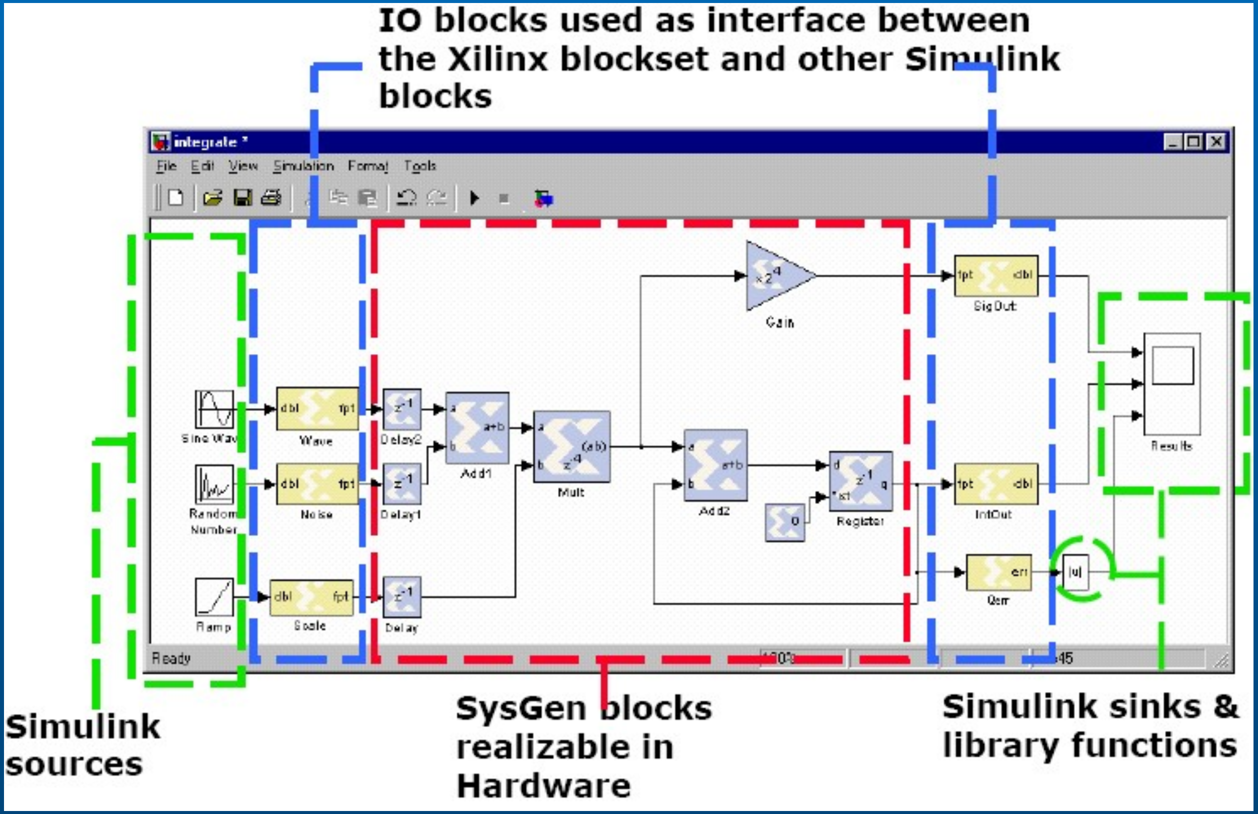


1. DSP System Modeling	The MathWorks MATLAB® / Simulink®
2. System Generation	XILINX System Generator for DSP
3. HDL Synthesis Choose From....	Synplicity Synplify Pro; Mentor Graphics Leonardo Spectrum; XILINX XST
4. Simulation (optional)	Mentor Graphics ModelSim; XILINX MXE
5. FPGA Implementation	XILINX ISE
6. In-System Debug	XILINX ChipScope ILA



- *II.3. Hardware implementation using System Generator*

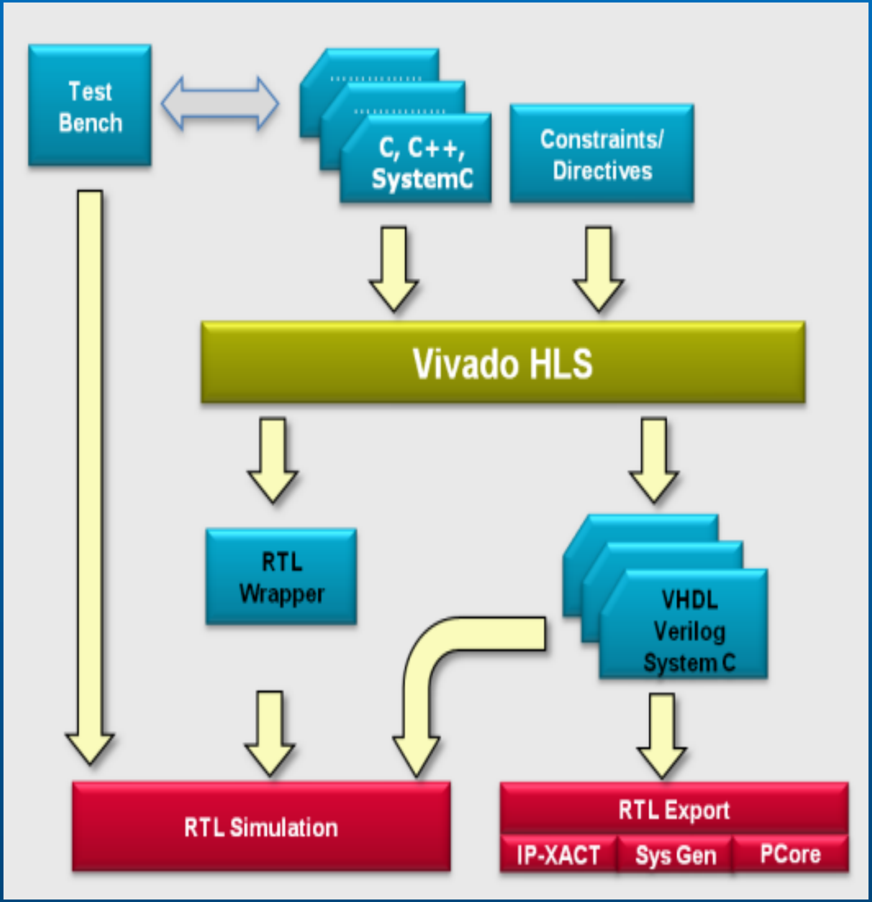
Design Flow using System Generator


IO blocks used as interface between the Xilinx blockset and other Simulink blocks
integrate *
File Edit View Simulation Format Tools
Sine Wave
Random Number
Ramp
Wave
Noise
Scale
Delay2
Delay1
Delay
Add1
Mult
Gain
Add2
Register
SigOut
IntOut
Qerr
Results
Ready
Simulink sources
SysGen blocks realizable in Hardware
Simulink sinks & library functions

● *II.4. Hardware implementation using HLS*

Design Flow [7], [8]

- **Inputs**: C/C++ based input design specification, constrains and directives;
- **Outputs**: RTL design files in VHDL, Verilog, SystemC;
- **In addition to that,** verification and implementation scripts, used to automated the RTL verification and RTL synthesis steps, are also created.

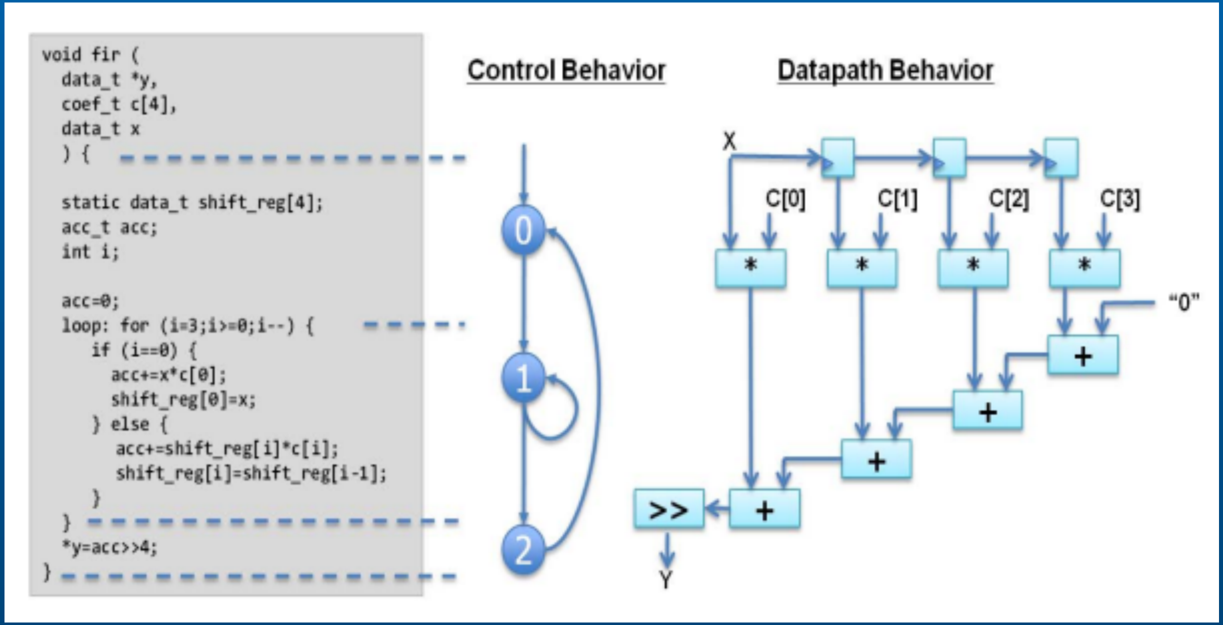


● *II.4. Hardware implementation using HLS*

Step 1: Control and Data paths Extraction [7], [8]

The first thing which is performed during HLS is to extract the control and data paths inferred by the code.

Example:



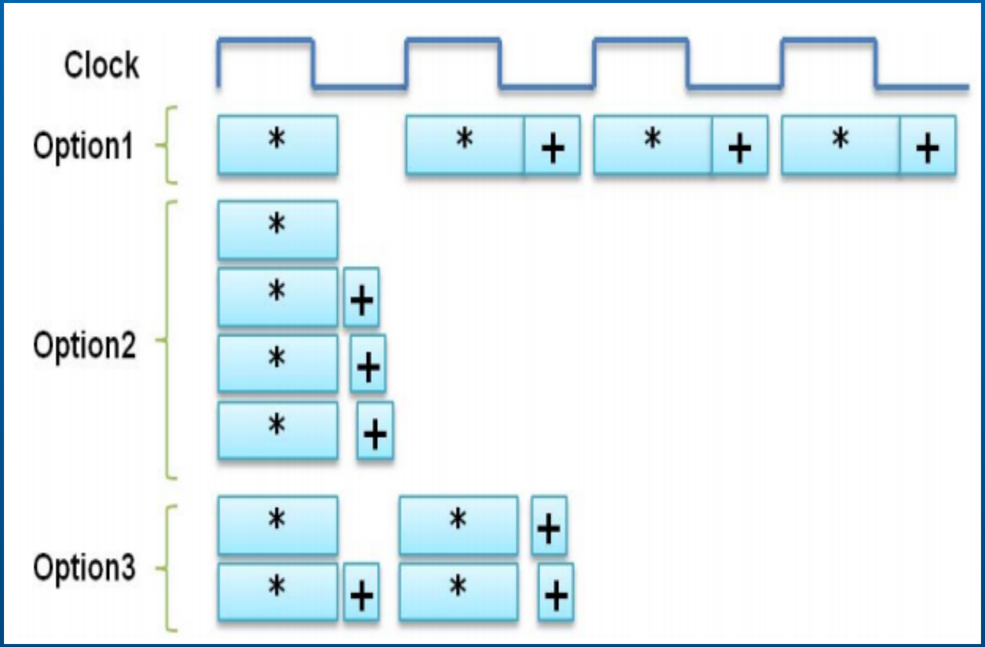
```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {

  static data_t shift_reg[4];
  acc_t acc;
  int i;

  acc=0;
  loop: for (i=3;i>=0;i--) {
    if (i==0) {
      acc+=x*c[0];
      shift_reg[0]=x;
    } else {
      acc+=shift_reg[i]*c[i];
      shift_reg[i]=shift_reg[i-1];
    }
  }
  *y=acc>>4;
}
```

● *II.4. Hardware implementation using HLS*

Step 2: Scheduling & Binding [7], [8]

For the same example code shown in the previous slide, multiple RTL implementations are possible.

1. Using 4 clock cycles means a single adder and multiplier can be used, as High-Level Synthesis can share the adder and multiplier across clock cycles: 1 adder, 1 multiplier and 4 clock cycles to complete.
2. If analysis of the target technology timing indicates the adder chain can complete in 1 clock cycle, a design which uses 3 adders and 4 multipliers but which finish in 1 clock cycle can be realized (faster but larger than option 1).
3. Take 2 clock cycles to finish but use only 2 adders and 2 multipliers (smaller than option 2 but faster than option 1).

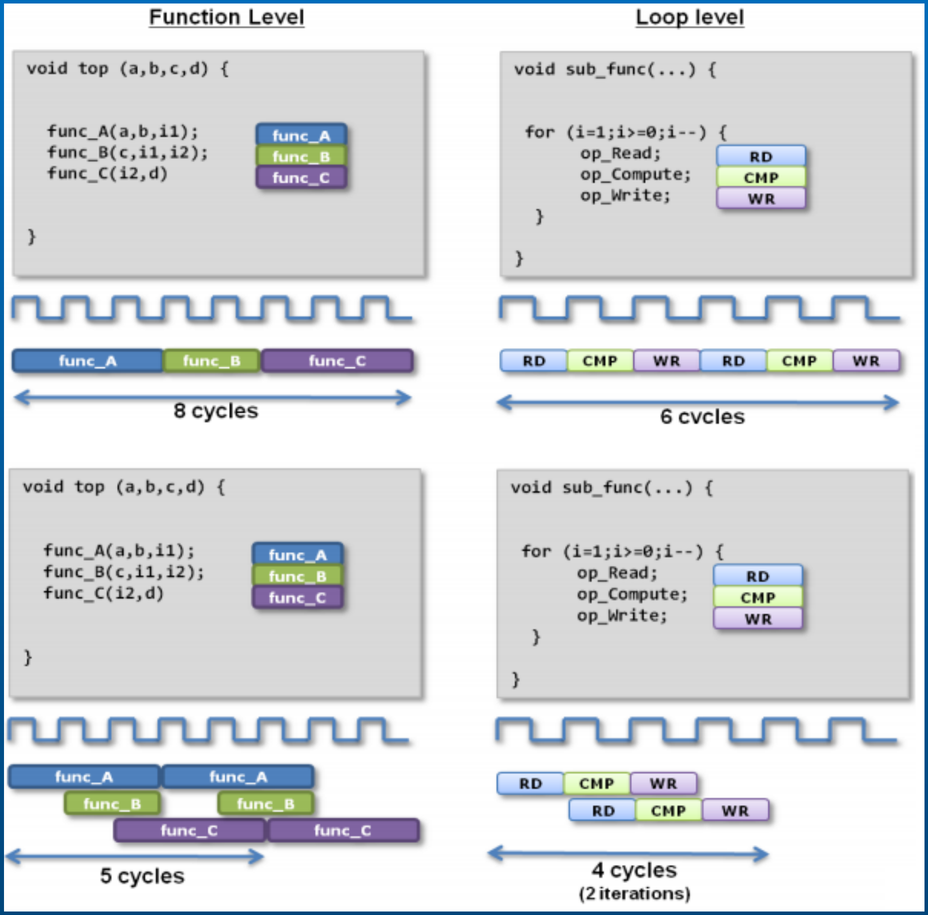


• *II.4. Hardware implementation using HLS*

Step 3: Optimization [7], [8]

High-Level Synthesis can perform a number of optimizations on the design to produce high quality RTL satisfying the performance and area goals.

Pipelining is an optimization which allows one of the major performance advantages of hardware over software, concurrent or parallel operation, to be automatically implemented in the RTL design.



```
void top (a,b,c,d) {

  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

}
```

```
void sub_func(...) {

 for (i=1;i>=0;i--) {
      op_Read;
      op_Compute;
      op_Write;
  }
}
```

```
void top (a,b,c,d) {

  func_A(a,b,i1);
  func_B(c,i1,i2);
  func_C(i2,d)

}
```

```
void sub_func(...) {

 for (i=1;i>=0;i--) {
      op_Read;
      op_Compute;
      op_Write;
  }
}
```

• *II.4. Hardware implementation using HLS [hls1]*

Step 4: Constrains [7], [8]

Finally, in addition to the clock period and clock uncertainty, High-Level Synthesis offers a number of design constraints including the ability to:

- Specify a specific latency across functions, loops and regions.
- Specify a limit on the number of resources used.
- Override the inherent or implied dependencies in the code and permit operations (for example, a memory read before write)

These constraints can be applied using High-Level Synthesis directives to create a design with the desired attributes.

- *II.5. Design Verification*

Best Practice [9]

1. Use modeling and simulation to optimize at the system level.
2. Automatically generate readable, traceable HDL code for FPGA prototyping.
3. Reuse system-level test benches with cosimulation for HDL verification.
4. Enable regression testing with FPGA-in-the-loop simulation.

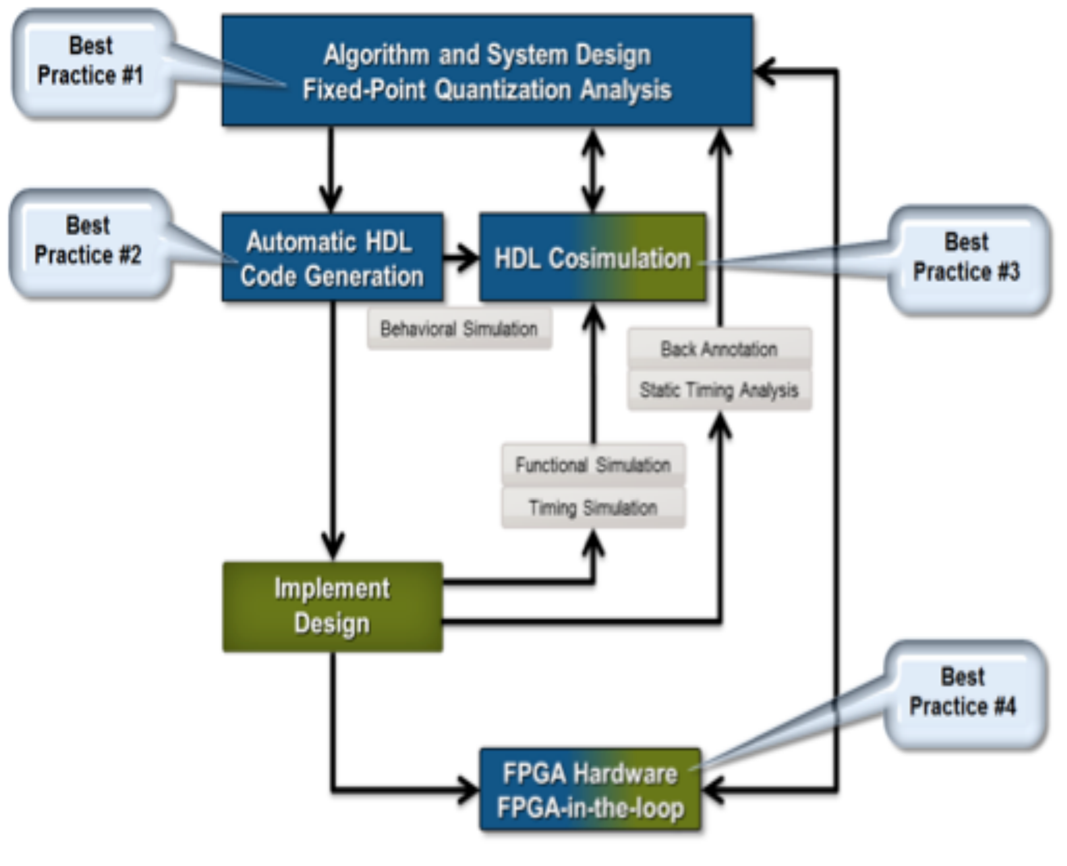
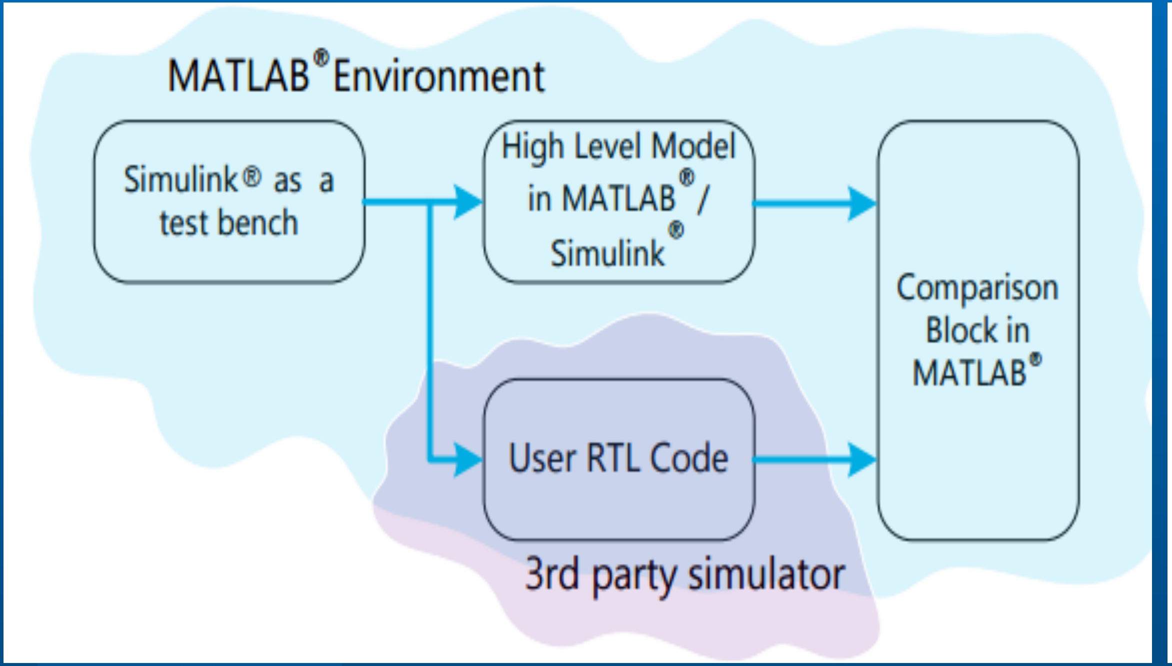
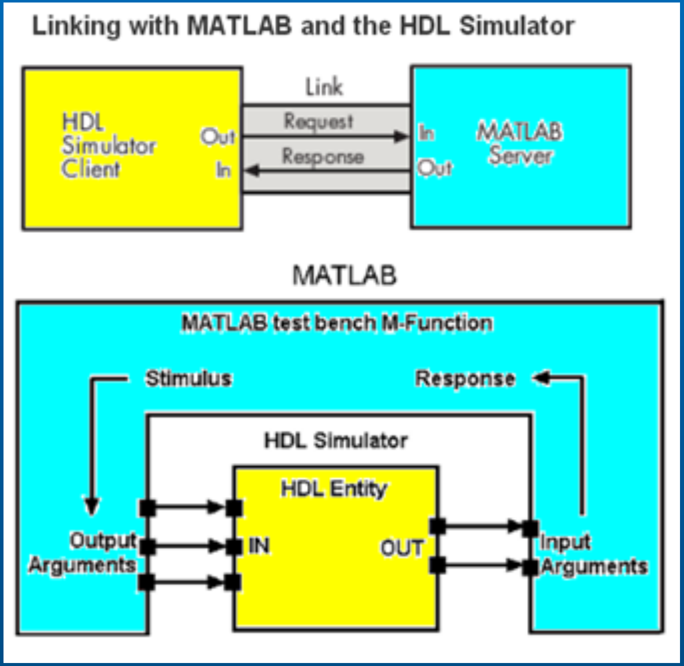


Figure 1 - Model-Based Design best practices for FPGA prototype development

- *II.5. Design Verification*

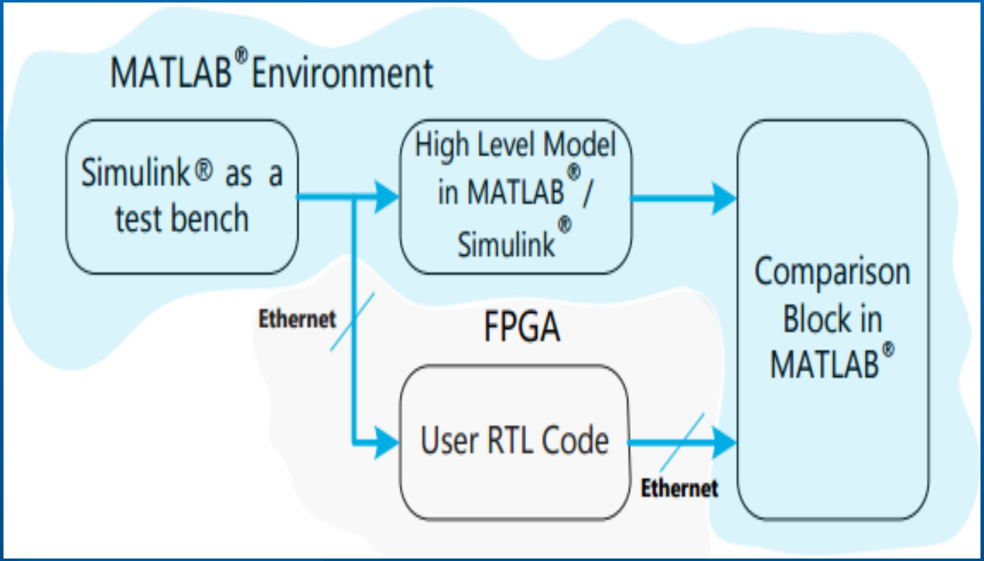
Cosimulation [10]

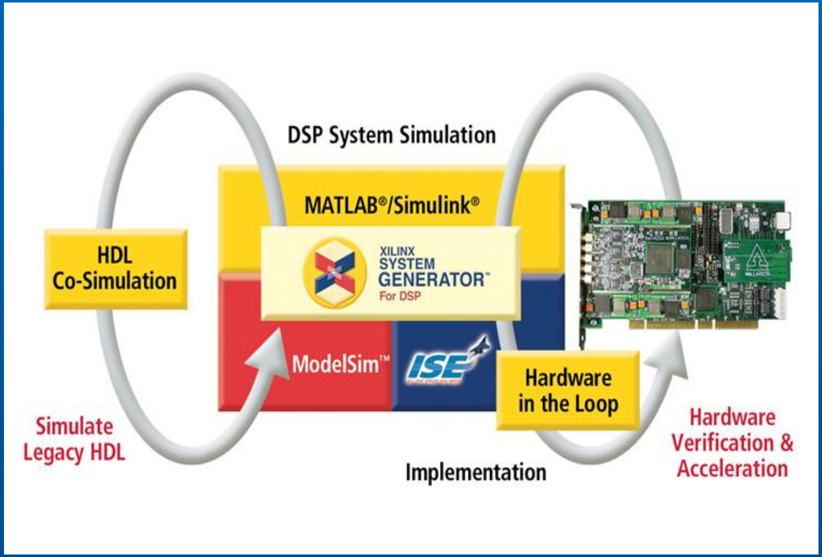

MATLAB® Environment
Simulink® as a test bench
High Level Model in MATLAB® / Simulink®
Comparison Block in MATLAB®
User RTL Code
3rd party simulator



- *II.5. Design Verification*

FPGA in the Loop (FIL) [10]


MATLAB® Environment
Simulink® as a test bench
High Level Model in MATLAB® / Simulink®
Comparison Block in MATLAB®
Ethernet
FPGA
User RTL Code
Ethernet



● *II.5. Design Verification – FIL*

FPGA-in-the-loop (FIL) enables you to run a Simulink or MATLAB simulation that is synchronized with an HDL design running on an Altera® or Xilinx® FPGA board.

This link between the simulator and the board enables you to:

- Verify HDL implementations directly against algorithms in Simulink or MATLAB.
- Apply data and test scenarios from Simulink or MATLAB to the HDL design on the FPGA.
- Integrate existing HDL code with models under development in Simulink or MATLAB.

● *II.5. Design Verification – FIL*

Requirements: [11]

- **MATLAB, Simulink, Fixed-Point Designer, HDL Verifier;**
- **FPGA design software (Xilinx® ISE® design suite, or Xilinx® Vivado® design suite, or Altera® Quartus® II design software);**
- **One of the supported FPGA development boards and accessories**

Steps to follow: [11]

Step 1: Set Up FPGA Development Board
Step 2: Set Up Host Computer-Board Connection
Step 3: Prepare Example Resources
Step 4: Launch FPGA-in-the-Loop (FIL) Wizard
Step 5: Specify Hardware Options in FIL Wizard

• *II.5. Design Verification – FIL*

Steps to follow (continued): [11]

Step 6: Specify HDL Files in the FIL Wizard

Step 7: Review I/O Ports in FIL Wizard

Step 8: Set Output Data Types in FIL Wizard

Step 9: Review Build Options in FIL Wizard

Step 10: Set Up Model

Step 11: Program FPGA

Step 12: Review Parameters of FIL Block

Step 13: Run FIL

III. Examples of PID controllers implemented in FPGA

Example 1: RTL synthesis of a PID controller

Example 2: HLS of a PID controller

Example 3: HLS of a fuzzy PID controller

Part III. Examples of PID controllers implemented in FPGA

● *Ex. 1: RTL Synthesis of a PID Controller*

PID Equations (time/Laplace) [12]

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \quad (1)$$

$$G(s) = K\left(1 + \frac{1}{sT_i} + sT_d\right) \quad (2)$$

$$u(t) = K_p e(t) + K_i\int_0^t e(\tau)d\tau + K_d\frac{d}{dt}e(t) \quad (3)$$

$$G(s) = K_p + \frac{K_i}{s} + sK_d \quad (4)$$

$$\Rightarrow$$

$$K_p = K, \quad K_i = \frac{K}{T_i}, \quad K_d = KT_d \quad (5)$$

Part III. Examples of PID controllers implemented in FPGA

● *Ex. 1: RTL Synthesis of an PID controller*

PID Equations (Discrete Time) [12]

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \qquad (3)$$

$$U(z) = \left[K_p + \frac{K_i}{1-z^{-1}} + K_d(1-z^{-1}) \right] E(z) \qquad (6)$$

$$U(z) = \left[\frac{(K_p + K_i + K_d) + (-K_p - 2K_d)z^{-1} + K_d z^{-2}}{1-z^{-1}} \right] E(z) \qquad (7)$$

$$U(z) - z^{-1}U(z) = \left[K_1 + K_2 z^{-1} + K_3 z^{-2} \right] E(z) \qquad (8)$$

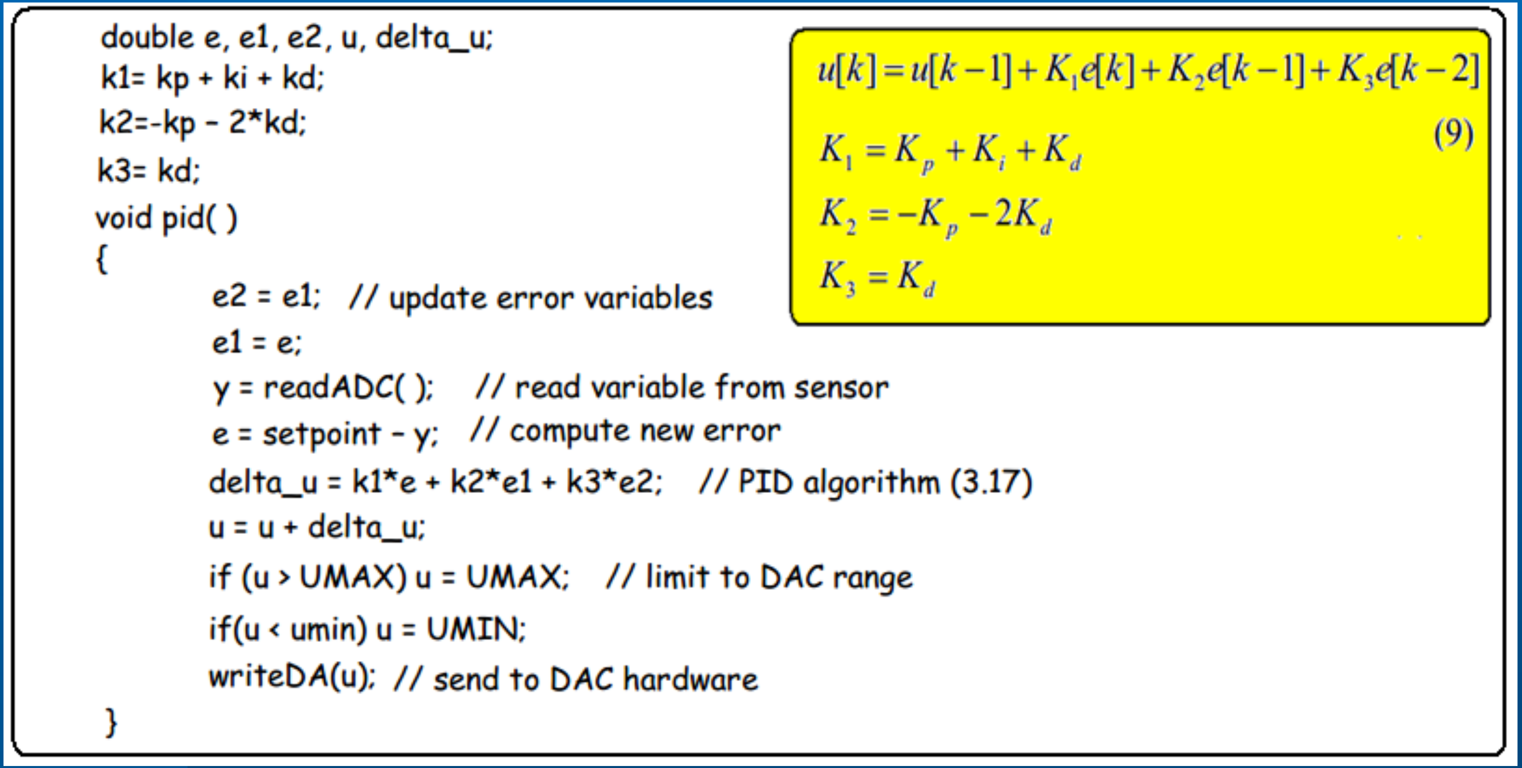
$$\Rightarrow$$

$$\boxed{\begin{aligned} & u[k] = u[k-1] + K_1 e[k] + K_2 e[k-1] + K_3 e[k-2] \\ & K_1 = K_p + K_i + K_d \qquad (9) \\ & K_2 = -K_p - 2K_d \\ & K_3 = K_d \end{aligned}}$$

Part III. Examples of PID controllers implemented in FPGA

- *Ex. 1: RTL Synthesis of a PID Controller*

PID Algorithm implementation in *C* [12]



```
double e, e1, e2, u, delta_u;
k1= kp + ki + kd;
k2=-kp - 2*kd;
k3= kd;
void pid( )
{
        e2 = e1;  // update error variables
        e1 = e;
        y = readADC( );    // read variable from sensor
        e = setpoint - y;  // compute new error
        delta_u = k1*e + k2*e1 + k3*e2;   // PID algorithm (3.17)
        u = u + delta_u;
        if (u > UMAX) u = UMAX;   // limit to DAC range
        if(u < umin) u = UMIN;
        writeDA(u);  // send to DAC hardware
}
```

$$u[k]=u[k-1]+K_1e[k]+K_2e[k-1]+K_3e[k-2]$$

$$K_1=K_p+K_i+K_d \quad (9)$$

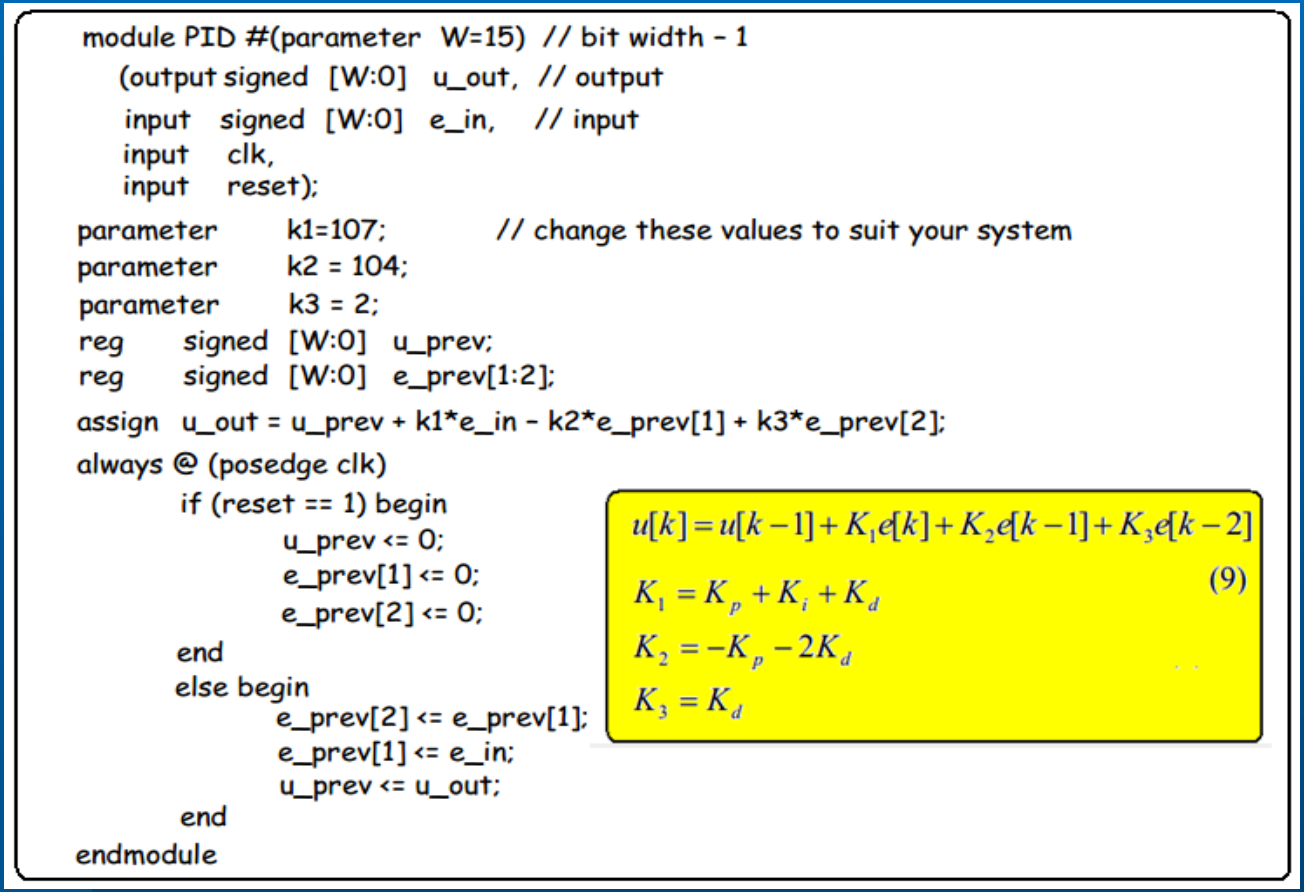
$$K_2=-K_p-2K_d$$

$$K_3=K_d$$

Part III. Examples of PID controllers implemented in FPGA

● *Ex. 1: RTL Synthesis of a PID Controller*

PID Algorithm Implementation in *Verilog* [12]



```
module PID #(parameter  W=15)  // bit width - 1
   (output signed  [W:0]  u_out,  // output
    input   signed  [W:0]  e_in,    // input
    input    clk,
    input    reset);
parameter       k1=107;          // change these values to suit your system
parameter       k2 = 104;
parameter       k3 = 2;
reg       signed  [W:0]  u_prev;
reg       signed  [W:0]  e_prev[1:2];
assign  u_out = u_prev + k1*e_in - k2*e_prev[1] + k3*e_prev[2];
always @ (posedge clk)
          if (reset == 1) begin
                    u_prev <= 0;
                    e_prev[1] <= 0;
                    e_prev[2] <= 0;
          end
          else begin
                    e_prev[2] <= e_prev[1];
                    e_prev[1] <= e_in;
                    u_prev <= u_out;
          end
endmodule
```

$$u[k] = u[k-1] + K_1 e[k] + K_2 e[k-1] + K_3 e[k-2]$$

$$K_1 = K_p + K_i + K_d \qquad (9)$$

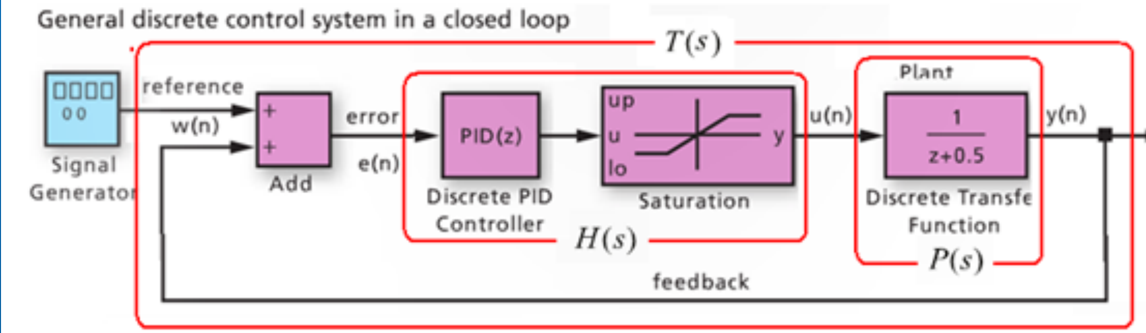
$$K_2 = -K_p - 2K_d$$

$$K_3 = K_d$$

Part III. Examples of PID controllers implemented in FPGA

Ex. 2: HLS of a PID Controller

Closed Loop Control System [13]



$$P(s) = \frac{Y(s)}{U(s)} = \frac{a}{s^2 b + c + d}$$

$$H(s) = \frac{U(s)}{E(s)} = K_P + \frac{1}{s} \cdot K_I + s \cdot K_D$$

$$T(z) = \frac{P(z) \cdot H(z)}{1 + P(z) \cdot H(z)}$$

$$P(z) = 10^{-5} \cdot \frac{2.38z^2 + 4.76z + 2.38}{z^2 - 1.903z + 0.9048}$$

$$H(z) = K_P + K_I \frac{T_S}{2} \frac{z+1}{z-1} + \frac{K_D}{T_F + \frac{T_S}{2} \frac{z+1}{z-1}}$$

$$P(z) = 10^{-5} \cdot \frac{2.38 + 4.76z^{-1} + 2.38z^{-2}}{1 - 1.903z^{-1} + 0.9048z^{-2}}$$

$$H(z) = \frac{U(z)}{E(z)} = G_P + G_I \frac{1+z^{-1}}{1-z^{-1}} + G_D \frac{1-z^{-1}}{1+Cz^{-1}}$$

$$G_P = K_P$$

$$G_I = K_I \frac{T_S}{2}$$

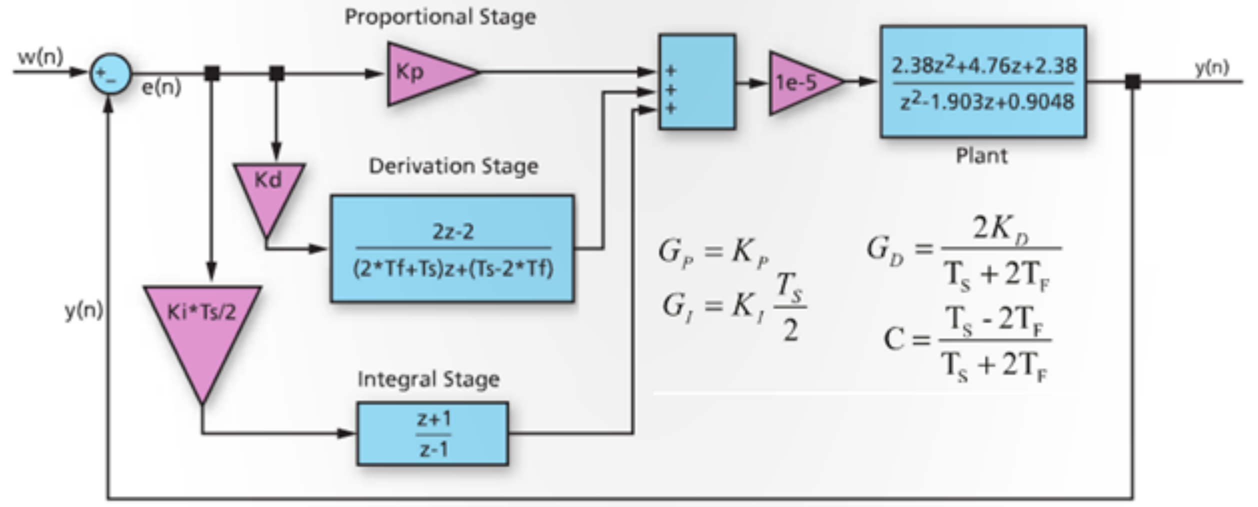
$$G_D = \frac{2K_D}{T_S + 2T_F}$$

$$C = \frac{T_S - 2T_F}{T_S + 2T_F}$$

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

Part III. Examples of PID controllers implemented in FPGA

- *Ex. 2: HLS of a PID Controller* [13]



$$G_P = K_P \qquad G_I = K_I \frac{T_S}{2}$$

$$G_D = \frac{2K_D}{T_S + 2T_F} \qquad C = \frac{T_S - 2T_F}{T_S + 2T_F}$$

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

$$P(z) = 10^{-5} \cdot \frac{2.38 + 4.76z^{-1} + 2.38z^{-2}}{1 - 1.903z^{-1} + 0.9048z^{-2}}$$

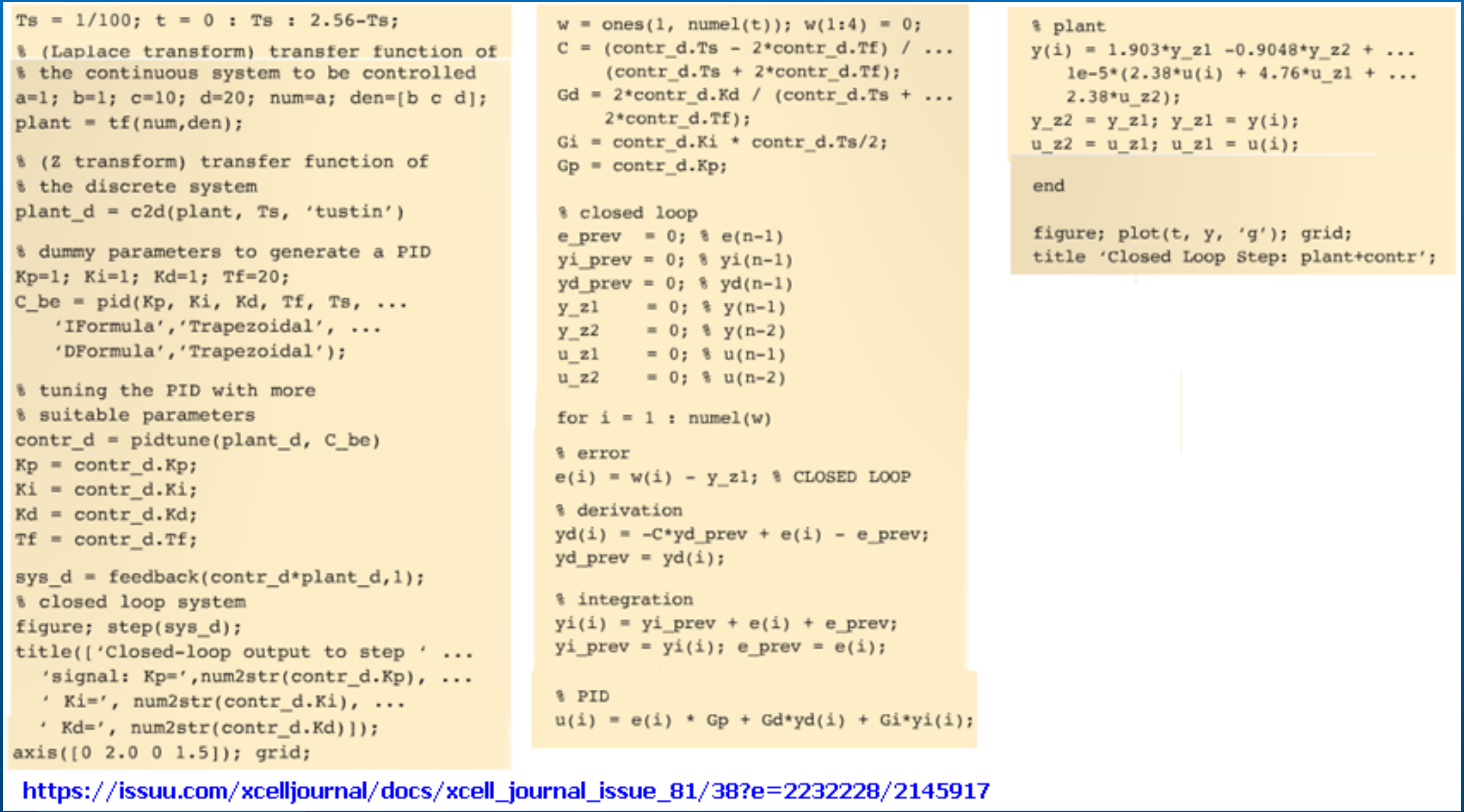
$$H(z) = \frac{U(z)}{E(z)} = G_P + G_I \frac{1+z^{-1}}{1-z^{-1}} + G_D \frac{1-z^{-1}}{1+Cz^{-1}}$$

$$\begin{aligned}
e(n) &= w(n) - y(n) \\
y_D(n) &= e(n) - e(n-1) - C \cdot y_D(n-1) \\
y_I(n) &= e(n) + e(n-1) + y_I(n-1) \\
y(n) &= 1.903 \cdot y(n-1) - 0.9048 \cdot y(n-2) + \\
&\quad + 10^{-5} \cdot (2.38 \cdot u(n) + 4.76 \cdot u(n-1) + 2.38 \cdot u(n-2))
\end{aligned}$$

Part III. Examples of PID controllers implemented in FPGA

- ## *Ex. 2: HLS of a PID Controller*

Matlab implementation [13]



```
Ts = 1/100; t = 0 : Ts : 2.56-Ts;
% (Laplace transform) transfer function of
% the continuous system to be controlled
a=1; b=1; c=10; d=20; num=a; den=[b c d];
plant = tf(num,den);

% (Z transform) transfer function of
% the discrete system
plant_d = c2d(plant, Ts, 'tustin')

% dummy parameters to generate a PID
Kp=1; Ki=1; Kd=1; Tf=20;
C_be = pid(Kp, Ki, Kd, Tf, Ts, ...
    'IFormula','Trapezoidal', ...
    'DFormula','Trapezoidal');

% tuning the PID with more
% suitable parameters
contr_d = pidtune(plant_d, C_be)
Kp = contr_d.Kp;
Ki = contr_d.Ki;
Kd = contr_d.Kd;
Tf = contr_d.Tf;

sys_d = feedback(contr_d*plant_d,1);
% closed loop system
figure; step(sys_d);
title(['Closed-loop output to step ' ...
   'signal: Kp=',num2str(contr_d.Kp), ...
   ' Ki=', num2str(contr_d.Ki), ...
   ' Kd=', num2str(contr_d.Kd)]);
axis([0 2.0 0 1.5]); grid;
```

```
w = ones(1, numel(t)); w(1:4) = 0;
C = (contr_d.Ts - 2*contr_d.Tf) / ...
    (contr_d.Ts + 2*contr_d.Tf);
Gd = 2*contr_d.Kd / (contr_d.Ts + ...
    2*contr_d.Tf);
Gi = contr_d.Ki * contr_d.Ts/2;
Gp = contr_d.Kp;

% closed loop
e_prev   = 0; % e(n-1)
yi_prev = 0; % yi(n-1)
yd_prev = 0; % yd(n-1)
y_z1     = 0; % y(n-1)
y_z2     = 0; % y(n-2)
u_z1     = 0; % u(n-1)
u_z2     = 0; % u(n-2)

for i = 1 : numel(w)

% error
e(i) = w(i) - y_z1; % CLOSED LOOP

% derivation
yd(i) = -C*yd_prev + e(i) - e_prev;
yd_prev = yd(i);

% integration
yi(i) = yi_prev + e(i) + e_prev;
yi_prev = yi(i); e_prev = e(i);

% PID
u(i) = e(i) * Gp + Gd*yd(i) + Gi*yi(i);
```

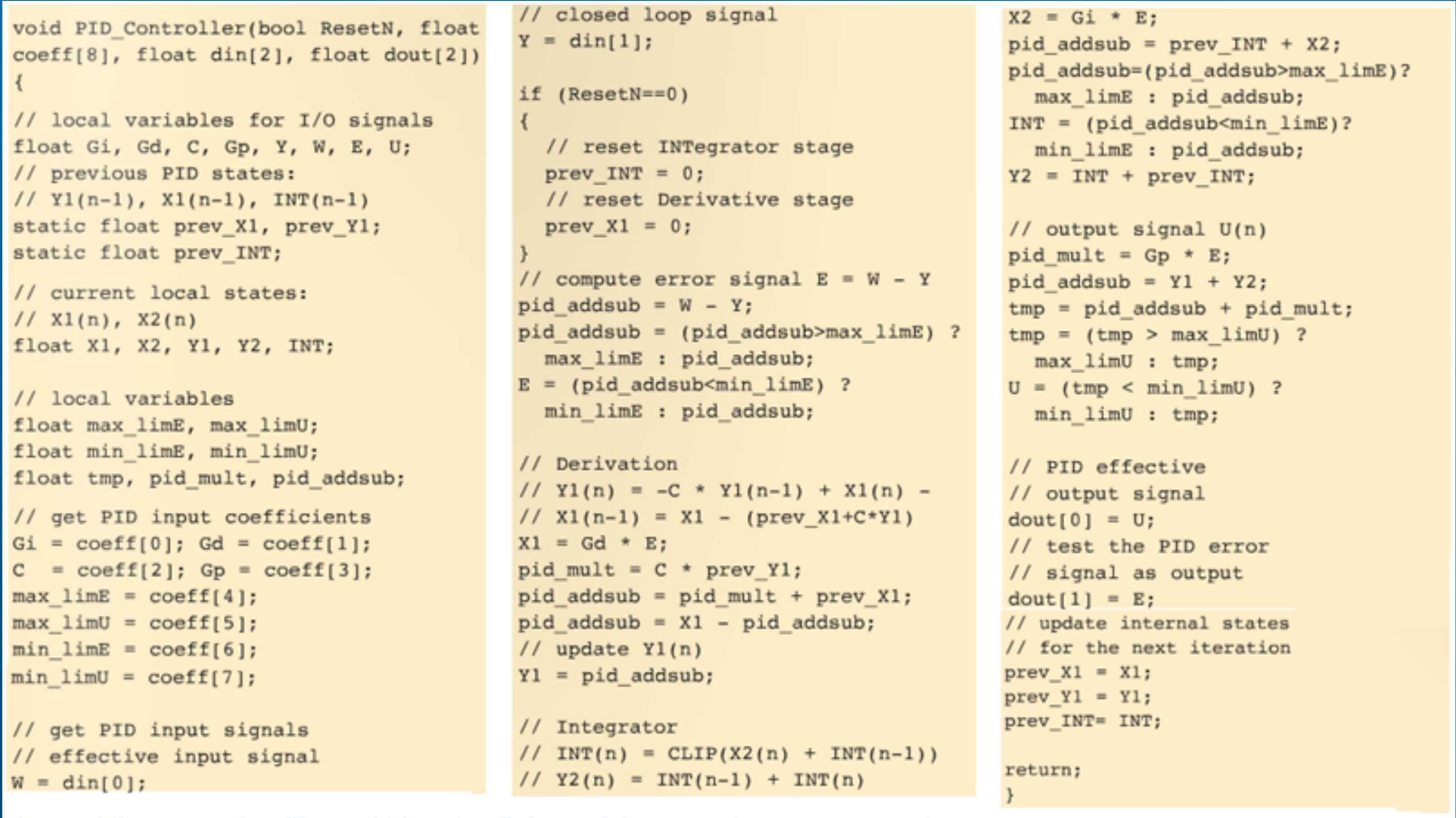
```
% plant
y(i) = 1.903*y_z1 -0.9048*y_z2 + ...
    1e-5*(2.38*u(i) + 4.76*u_z1 + ...
    2.38*u_z2);
y_z2 = y_z1; y_z1 = y(i);
u_z2 = u_z1; u_z1 = u(i);

end

figure; plot(t, y, 'g'); grid;
title 'Closed Loop Step: plant+contr';
```

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

Part III. Examples of PID controllers implemented in FPGA

- ## *Ex. 2: HLS of a PID Controller*

  C implementation [13]



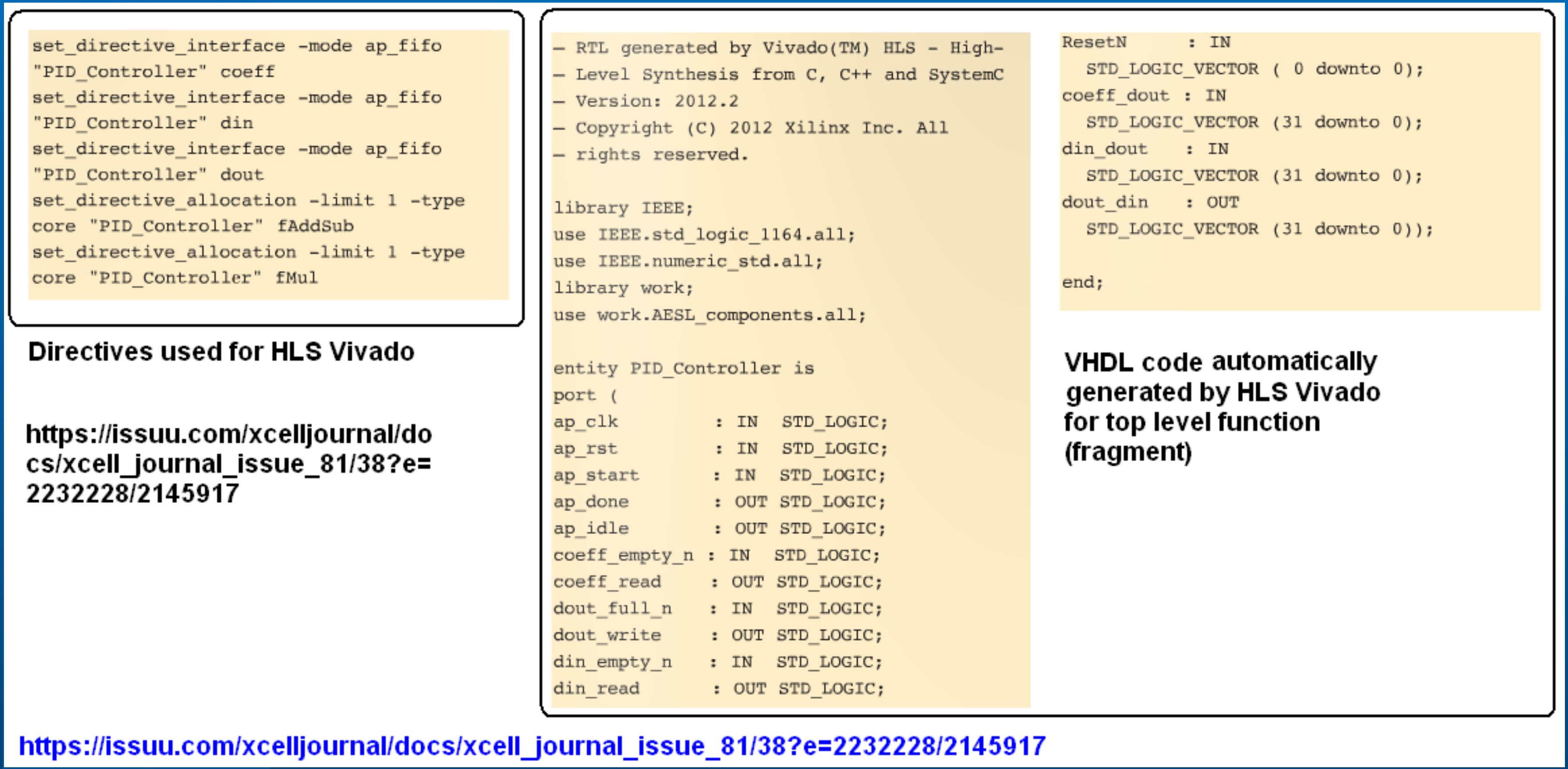
```
void PID_Controller(bool ResetN, float
coeff[8], float din[2], float dout[2])
{
// local variables for I/O signals
float Gi, Gd, C, Gp, Y, W, E, U;
// previous PID states:
// Y1(n-1), X1(n-1), INT(n-1)
static float prev_X1, prev_Y1;
static float prev_INT;

// current local states:
// X1(n), X2(n)
float X1, X2, Y1, Y2, INT;

// local variables
float max_limE, max_limU;
float min_limE, min_limU;
float tmp, pid_mult, pid_addsub;

// get PID input coefficients
Gi = coeff[0]; Gd = coeff[1];
C  = coeff[2]; Gp = coeff[3];
max_limE = coeff[4];
max_limU = coeff[5];
min_limE = coeff[6];
min_limU = coeff[7];

// get PID input signals
// effective input signal
W = din[0];
// closed loop signal
Y = din[1];

if (ResetN==0)
{
  // reset INTegrator stage
  prev_INT = 0;
  // reset Derivative stage
  prev_X1 = 0;
}
// compute error signal E = W - Y
pid_addsub = W - Y;
pid_addsub = (pid_addsub>max_limE) ?
  max_limE : pid_addsub;
E = (pid_addsub<min_limE) ?
  min_limE : pid_addsub;

// Derivation
// Y1(n) = -C * Y1(n-1) + X1(n) -
// X1(n-1) = X1 - (prev_X1+C*Y1)
X1 = Gd * E;
pid_mult = C * prev_Y1;
pid_addsub = pid_mult + prev_X1;
pid_addsub = X1 - pid_addsub;
// update Y1(n)
Y1 = pid_addsub;

// Integrator
// INT(n) = CLIP(X2(n) + INT(n-1))
// Y2(n) = INT(n-1) + INT(n)
X2 = Gi * E;
pid_addsub = prev_INT + X2;
pid_addsub=(pid_addsub>max_limE)?
  max_limE : pid_addsub;
INT = (pid_addsub<min_limE)?
  min_limE : pid_addsub;
Y2 = INT + prev_INT;

// output signal U(n)
pid_mult = Gp * E;
pid_addsub = Y1 + Y2;
tmp = pid_addsub + pid_mult;
tmp = (tmp > max_limU) ?
  max_limU : tmp;
U = (tmp < min_limU) ?
  min_limU : tmp;

// PID effective
// output signal
dout[0] = U;
// test the PID error
// signal as output
dout[1] = E;
// update internal states
// for the next iteration
prev_X1 = X1;
prev_Y1 = Y1;
prev_INT= INT;

return;
}
```

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

Part III. Examples of PID controllers implemented in FPGA

- *Ex. 2: HLS of a PID Controller*

VHDL code generation using HLS Vivado [13]



```
set_directive_interface -mode ap_fifo
"PID_Controller" coeff
set_directive_interface -mode ap_fifo
"PID_Controller" din
set_directive_interface -mode ap_fifo
"PID_Controller" dout
set_directive_allocation -limit 1 -type
core "PID_Controller" fAddSub
set_directive_allocation -limit 1 -type
core "PID_Controller" fMul
```

Directives used for HLS Vivado

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

```
— RTL generated by Vivado(TM) HLS - High-
— Level Synthesis from C, C++ and SystemC
— Version: 2012.2
— Copyright (C) 2012 Xilinx Inc. All
— rights reserved.

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library work;
use work.AESL_components.all;

entity PID_Controller is
port (
ap_clk          : IN   STD_LOGIC;
ap_rst          : IN   STD_LOGIC;
ap_start        : IN   STD_LOGIC;
ap_done         : OUT  STD_LOGIC;
ap_idle         : OUT  STD_LOGIC;
coeff_empty_n   : IN   STD_LOGIC;
coeff_read      : OUT  STD_LOGIC;
dout_full_n     : IN   STD_LOGIC;
dout_write      : OUT  STD_LOGIC;
din_empty_n     : IN   STD_LOGIC;
din_read        : OUT  STD_LOGIC;
```

```
ResetN      : IN
  STD_LOGIC_VECTOR ( 0 downto 0);
coeff_dout  : IN
  STD_LOGIC_VECTOR (31 downto 0);
din_dout    : IN
  STD_LOGIC_VECTOR (31 downto 0);
dout_din    : OUT
  STD_LOGIC_VECTOR (31 downto 0));

end;
```

VHDL code automatically generated by HLS Vivado for top level function (fragment)

https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

Part III. Examples of PID controllers implemented in FPGA

- *Ex. 3: High Level Synthesis of a Fuzzy PID Controller*

[14]

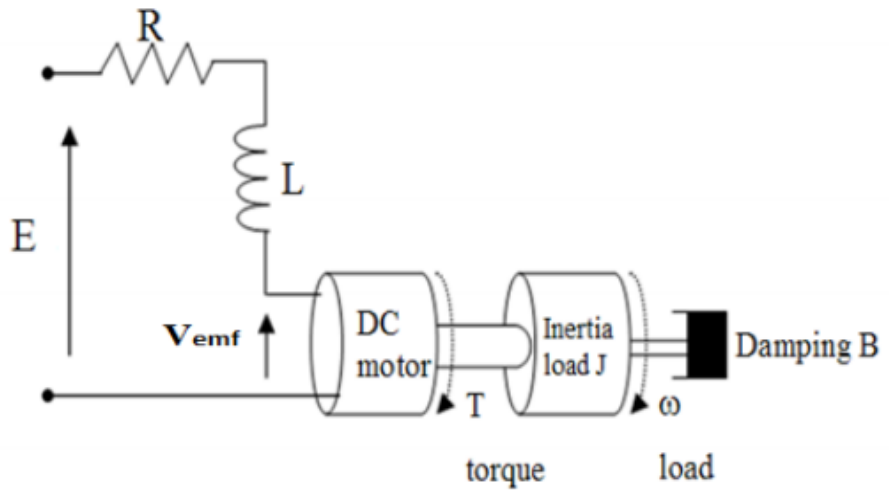


Fig.1. Model of DC motor

Table I DC Motor parameters

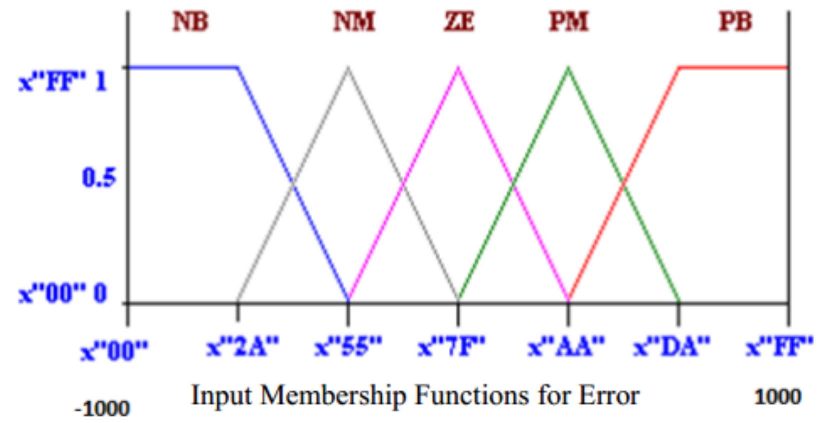
Parameter	Description	Value
R	Armature resistant	4.67 Ω
L	Armature inductance	170e-3 H
J	Moment of inertia	42.6e-6 Kg-m^2
f	Viscous-friction coefficient	47.3e-6 N-m/rad/sec
K_t	Torque constant	14.7e-3 N-m/A
K_b	Back-EMF constant	14.7e-3 V-sec/rad

Mani Shankar Anand, Barjeev Tyagi - "Design and Implementation of Fuzzy Controller on FPGA", I.J. Intelligent Systems and Applications, 2012, 10, 35-42

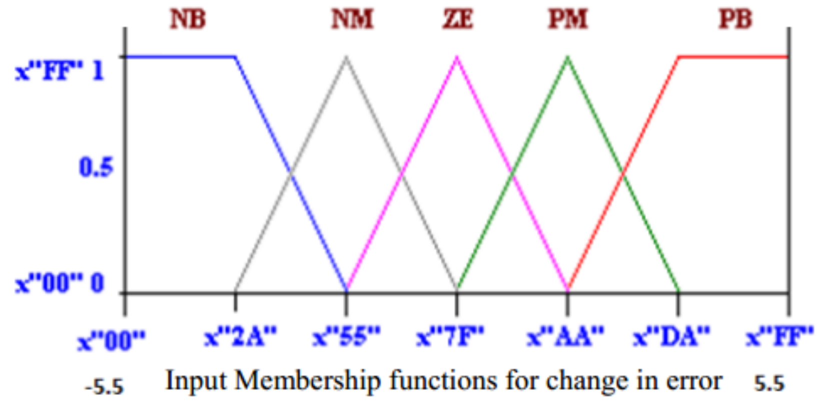
Part III. Examples of PID controllers implemented in FPGA

- *Ex. 3: High Level Synthesis of a Fuzzy PID Controller*

[14]



Input Membership Functions for Error



Input Membership functions for change in error

Table 2 Fuzzy final rules

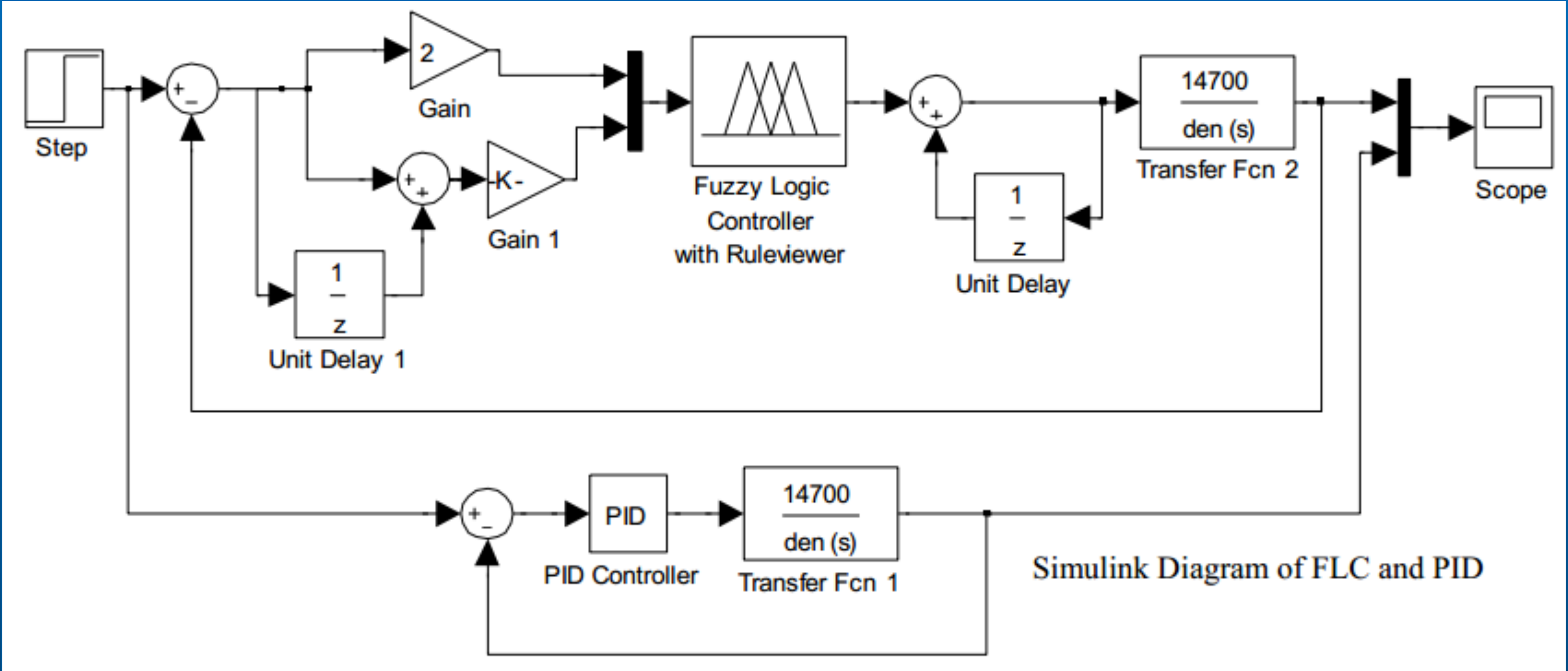
E CE	NB	NM	ZE	PM	PB
PB	NM	NS	NB	PB	PB
PM	NM	NM	NB	PB	PB
ZE	NB	NB	ZE	PB	PB
NM	NB	NB	NB	PM	PM
NB	NB	NB	NB	PS	PM

Mani Shankar Anand, Barjeev Tyagi -
"Design and Implementation of Fuzzy Controller on FPGA",
I.J. Intelligent Systems and Applications, 2012, 10, 35-42

Part III. Examples of PID controllers implemented in FPGA

- *Ex. 3: High Level Synthesis of a Fuzzy PID Controller*

[14]



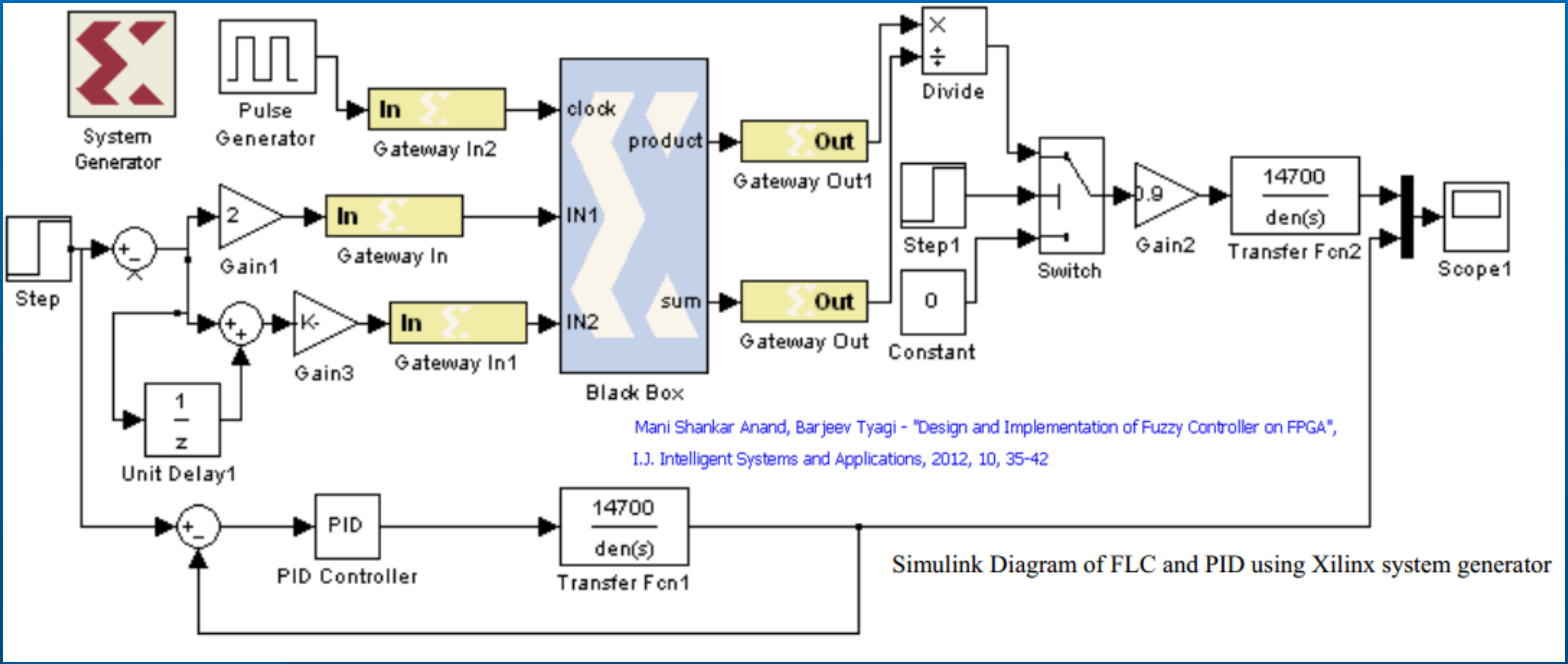
Simulink Diagram of FLC and PID

Mani Shankar Anand, Barjeev Tyagi - "Design and Implementation of Fuzzy Controller on FPGA", I.J. Intelligent Systems and Applications, 2012, 10, 35-42

Part III. Examples of PID controllers implemented in FPGA

• *Ex. 3: High Level Synthesis of a Fuzzy PID Controller*

[14]



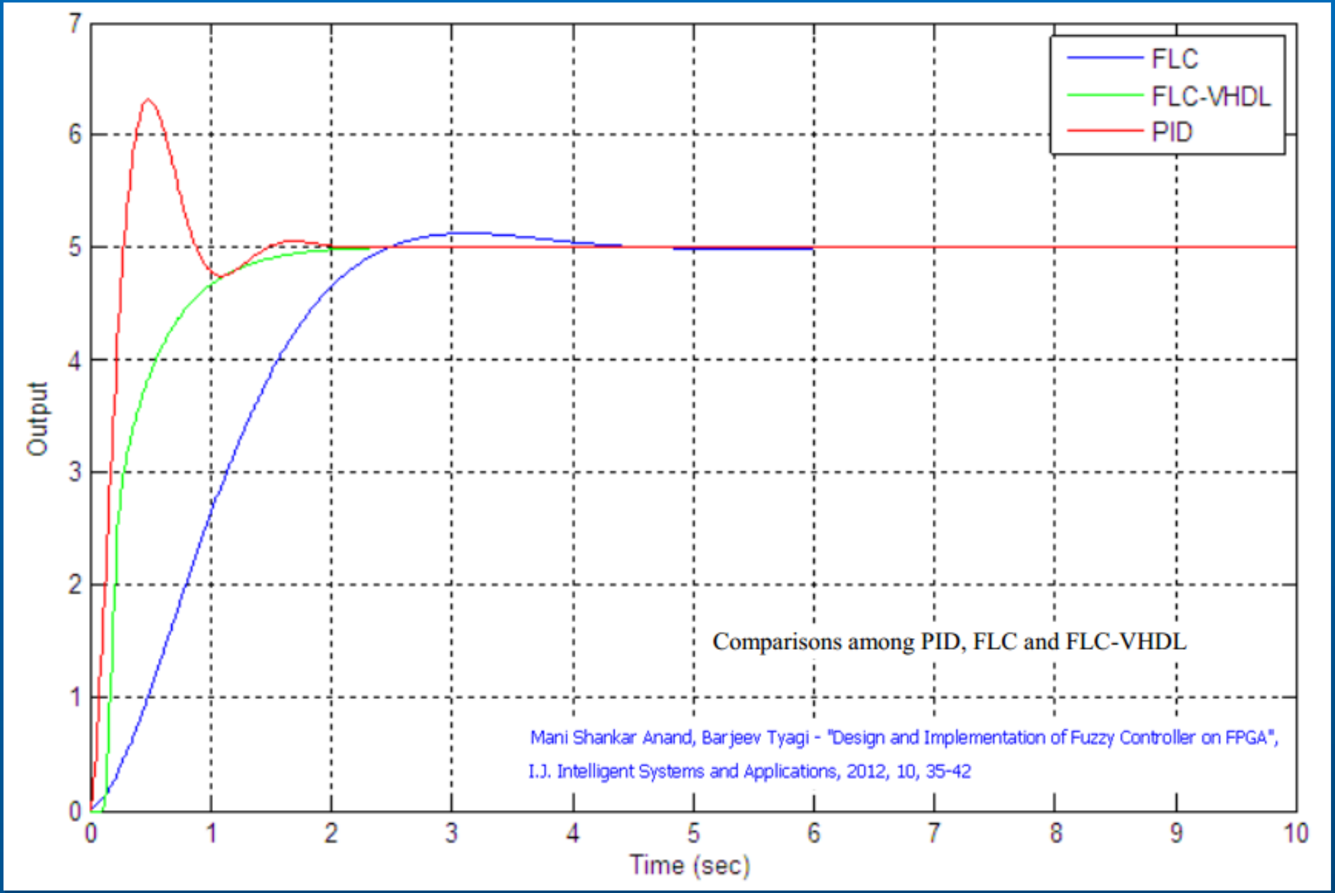
Mani Shankar Anand, Barjeev Tyagi - "Design and Implementation of Fuzzy Controller on FPGA", I.J. Intelligent Systems and Applications, 2012, 10, 35-42

Simulink Diagram of FLC and PID using Xilinx system generator

Part III. Examples of PID controllers implemented in FPGA

● *Ex. 3: High Level Synthesis of a Fuzzy PID Controller*

[14]



Comparisons among PID, FLC and FLC-VHDL

Mani Shankar Anand, Barjeev Tyagi - "Design and Implementation of Fuzzy Controller on FPGA", I.J. Intelligent Systems and Applications, 2012, 10, 35-42

● *References*

[1] Fouad M. AL-Sunni , “Introduction_To_Control_Systems”,
https://opencourseware.kfupm.edu.sa/colleges/ccse/se/se302/files%5C1._Introduction_To_Control_Systems_SE302_Topic_1_-_Introduction_to_Linear_systems.pdf

[2] http://www.xilinx.com/products/design-tools/microblaze.html

[3] http://www.xilinx.com/products/silicon-devices/soc.html

[4] https://www.so-logic.net/documents/knowledge/tutorial/Basic_FPGA_Tutorial_ISE/Basic_FPGA_Tutorial2.html#toc0

[5] http://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html

[6] http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/sysgen_user.pdf

[7] http://files.meetup.com/3753142/VHLS_NY_Final_Distributed.pdf

[8] Vivado Design Suite User Guide,
http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_4/ug902-vivado-high-level-synthesis.pdf

[9] http://www.eejournal.com/archives/articles/20110825-mathworks/

[10] Paweł Ządek, Arkadiusz Koczor, Michał Gołek, Łukasz Matoga, Piotr Penkala, “Improving Efficiency of FPGA-in-the-Loop Verification Environment”, IFAC-PapersOnLine 48-4 (2015) 180–185

[11] Verify HDL Implementation of PID Controller Using FPGA-in-the-Loop.
http://www.mathworks.com/help/hdlverifier/examples/verify-hdl-implementation-of-pid-controller-using-fpga-in-the-loop.html

[12] Varodom Toochinda, Digital PID Controllers, http://controlsystemslab.com/category/articles/control-engineering/pid/

[13] https://issuu.com/xcelljournal/docs/xcell_journal_issue_81/38?e=2232228/2145917

[14] Mani Shankar Anand, Barjeev Tyagi, Design and Implementation of Fuzzy Controller on FPGA, I.J. Intelligent Systems and Applications, 2012, 10, 35-42

Thank you for your attention!