INTRODUCTION TO FPGA

SYFALA WORKSHOP - PAW 23

Pierre Cochard, Maxime Popoff, Romain Michon,
Tanguy Risset, Yann Orlarey, Matthieu Imbert
and The Emeraude Team

December 1, 2023
INTRODUCTION TO FPGA

Context

HIGH LEVEL PROGRAMMING
Make programming of audio DSP easy

FPGA
High Performance Embedded Platform

Command Line Compiler

./syfala
<table>
<thead>
<tr>
<th></th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What’s an FPGA?</td>
</tr>
<tr>
<td>2</td>
<td>Hardware Description Language (HDL)</td>
</tr>
<tr>
<td>3</td>
<td>High Level Synthesis (HLS)</td>
</tr>
<tr>
<td>4</td>
<td>SoC Architecture</td>
</tr>
<tr>
<td>5</td>
<td>Hardware/Software Co-Design</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>
What’s an FPGA?
Field-Programmable Gate Array

**Field-Programmable:** the internal components of the device and the connections between them are programmable after deployment.

**Gate:** refers to logic gates, the basic building blocks for all the hardware on the chip.

**Array:** there are many (billions) of them on the chip.
What's an FPGA?

FPGAs contain an array of programmable blocks.
FPGAs contain an array of **programmable blocks**.

Programming them means **choosing and configuring** the blocks,
FPGAs contain an array of *programmable blocks*.

Programming them means *choosing and configuring* the blocks

and connecting them with *reconfigurable interconnects*. 
INTRODUCTION TO FPGA

What’s an FPGA?

FPGAs contain an array of programmable blocks.

Programming them means choosing and configuring the blocks and connecting them with reconfigurable interconnects.

This architecture defines the function implemented in the FPGA.
Configurable Logic Block (CLB)

Made of a Look Up Table (LUT) and a Register: Can do any logic function N to 1.
(e.g., XOR, Custom State Machine, MUX)
Configurable Logic Block (CLB)

What's an FPGA?

Made of a **Look Up Table (LUT)** and a Register: Can do any logic function N to 1. (e.g., XOR, Custom State Machine, MUX)

Other Blocks

**Memory Blocks**: On chip **RAM** blocks.

**DSP Blocks**: for custom **math functions**.

**Independent Clock Networks** with their **PLLs**.
**What’s an FPGA?**

**(Re)Configurable Hardware**

“Programmed” using a **Hardware Description Language** (HDL).

You don’t program an FPGA, you **configure** it (as a circuit).

The programming model of FPGAs is the digital circuit.
**INTRODUCTION TO FPGA**

**What's an FPGA?**

**CPU**

Traditional computer architectures use a CPU. Programmable software.

Throughput and computational power limited by the number of core and frequency.

**FPGA**

Throughput and computational power limited by the number of resources available (and the clock freq.).

[Diagram of CPU and FPGA processes]
Parallelization: Using resources in parallel. FPGA embed a lot of physical I/O, allowing for parallel data processing (e.g., multichannel).
What's an FPGA?

Pipelining: Using resources sequentially
Increase throughput but doesn't reduce latency.
FPGAs are a better fit than CPUs for **real-time audio processing**.

- High **throughput**.
- Very **low latency**.
- High **sampling rate** (>20MHz).
- Very large number of **inputs/outputs**.
- **Parallel** data processing.

They have been increasingly used in recent years for real-time audio Digital Signal Processing (DSP) applications. But they are very hard to program!
FPGA Workflow (with Syfala)

Really High-level synthesis flow

High-level synthesis flow

Classic HDL flow

1. High-level DSP lang (e.g. Faust)
2. Faust Compiler
3. High-level language (e.g. C++)
4. HLS Tool
5. Synthesis Tool

INTRODUCTION TO FPGA

What's an FPGA?

High-level language (e.g. Faust)

Compiler

HLS Tool

Synthesis Tool
Hardware Description Language (HDL)
Hardware Description Language

Describe the **structure and behavior** of electronic circuits.

It allows for the **synthesis** of an HDL description into a **netlist** (a specification of physical electronic components and how they are connected together), which can then be **placed** and **routed** to produce the set of masks used to create an integrated circuit (very long process).

**Two** major hardware description languages: **VHDL** and **Verilog**.

HDL text books are usually distinguishing three different abstraction levels: **structural**, **dataflow** (RTL: functional), **behavioral** (Sequential).
High Level Synthesis (HLS)
High Level Synthesis

**High-Level Synthesis (HLS)** is an automated design process that takes an abstract behavioral specification of a digital system and generates a **register-transfer level (RTL)** structure that realizes the given behavior.

Since the 2010, HLS allows for **fully sequential approach FPGA programming in C/C++**.
High Level Synthesis

High-Level Synthesis (HLS) is an automated design process that takes an abstract behavioral specification of a digital system and generates a register-transfer level (RTL) structure that realizes the given behavior.

Since the 2010, HLS allows for fully sequential approach FPGA programming in C/C++.

You still have to handle the design constraints.

Vitis High-Level Synthesis User Guide (UG1399)
Example: how to implement a multiply–accumulate (MAC) operation.

```c
for (int i = 0; i < 10; i++) {
    out += i*i;
}
```
Example: how to implement a multiply–accumulate (MAC) operation.

```cpp
for (int i = 0; i < 10; i++) {
    out += i*i;
}
```

Fully sequential approach:

![Diagram of fully sequential approach]

Reusing the same block: **high latency**, but **resource-efficient**. 

$$T_{exec} = (T_{Mult} + T_{Add}) \times 10$$
Example: how to implement a multiply–accumulate (MAC) operation.

```c
for (int i = 0; i < 10; i++) {
    out += i*i;
}
```

**Fully sequential approach:**

1. **Reg**
2. **Mult**
3. **Adder**

10 times

Reusing the same bloc: **high latency**, but **resource-efficient**. $T_{exec} = (T_{Mult} + T_{Add}) \times 10$

**Fully parallelized approach:**

Duplicate the function: very **expensive in resources** but allows a **low latency**. $T_{exec} = (T_{Mult} + 4 \times T_{Add})$
Example: how to implement a multiply–accumulate (MAC) operation.

Using **pragmas and directives** allows us to fine tune the **design constraints**...

**Fully sequential approach:**

```c
for (int i = 0; i < 10; i++) {
    out += i*i;
}
```

**Fully parallelized approach:**

```c
for (int i = 0; i < 10; i++) {
    //Fully unroll the loop
    out += i*i;
}
/*Equivalent of:
out+=0*0;
out+=1*1;
out+=2*2;
...
out+=9*9; */
```
But there are many parameters to take into account.
But there are many parameters to take into account.

**Fully sequential approach:**

![Fully sequential approach diagram]

We can pipeline this architecture by using the multiplier with $i+1$ while the result of $i$ is being computed by the adder.

$$T_{exec} = (T_{Mult} + T_{Add}) \times 10$$

$$T_{exec} = (T_{Mult} + T_{Add}) \times 5$$
But there are many parameters to take into account.

**Fully sequential approach:**

We can **pipeline** this architecture by using the **multiplier** with \( i+1 \) while the result of \( i \) is being computed by the **adder**.

\[
T_{exec} = (T_{Mult} + T_{Add}) \times 10
\]

\[
T_{exec} = (T_{Mult} + T_{Add}) \times 5
\]

**Fully parallelized approach:**

If \( i \) is stocked in the BRAM, we can only perform **2 accesses** per cycles.

\[
T_{exec} = (T_{Mult} + 4 \times T_{Add})
\]

\[
T_{exec} = (T_{Mult} + 4 \times T_{Add} + 5 \times T_{Mem})
\]
Syfala: Really High Level Synthesis

**Syfala** Implement the design constraints.
SoC Architecture
We will use a **SoC (System On Chip)**: FPGA + CPU (ARM).
Usually, we don’t want to implement the entire program on the hardware (FPGA).

Example with a FAUST program:

```faust
import("stdfaust.lib");
f=hslider("freq[knob:1]",400,50,2000,0.01);
sine0sc=os.oscrs(f);
echo=+-@(ma.SR*0.5)*0.5;
process=sine0sc:echo:*0.5;
```

`os.oscrs` is a sinusoidal oscillator based on a resonant filter.

To compute the coefficients of the filter from the frequency parameter, the `sin` and `cos` functions are needed.

The long delay in the echo implies the use of a lot of memory.
We can dispatch the computations and the memory use to save FPGA resources.

FPGA
- Audio/Sample Rate
  - Typ., 32 kHz to 192 kHz

ARM
- Control Rate
  - Typ., 100 Hz to 1000 Hz

Memory
- BRAM
  - 270 KB
    - Small objects
- DDR
  - 512 MB
    - Large arrays
import("stdfaust.lib");
f=hslider("freq[knob:1]",400,50,2000,0.01);
sine0sc=os.oscrs(f);
echo=+~@(ma.SR*0.5)*0.5;
process=sine0sc:echo:*0.5;
**Introduction to FPGA**

SoC Architecture: Balancing Computations

```plaintext
import("stdfaust.lib");
f=hslider("freq[knob:1]",400,50,2000,0.01);
sineOsc=os.oscrs(f);
echo=+-@((ma.SR*0.5)*0.5);
process=sineOsc:echo:*0.5);
```

A part of the program is **software** on the **CPU** (C++ compiled with gcc)

A part of the program is **hardware** on the **FPGA** (C++ synthetised with HLS or directly HDL)

Syfala use a FAUST backend to automatically do the dispatch.
import("stdfaust.lib");
f=hslider("freq[knob:1]",400,50,2000,0.01);
sineOsc=os.oscrs(f);
echo=+-@*0.5*0.5;
process=sineOsc:echo:*0.5;

A part of the program is **software** on the **CPU** (C++ compiled with gcc)

A part of the program is **hardware** on the **FPGA** (C++ synthetised with HLS or directly HDL)

**Syfala use a FAUST backend to automatically do the dispatch.**

**How can we handle this co-design?**
Hardware/Software Co-Design
Different workflows:

- Using the **same high level language** for both **hardware and software** allows you to compile a single file with an acceleration function that will be hardware. (e.g. Intel oneAPI DPC++, Spatial Lang)

- **Generate IPs** from HLS tools (in C++) or directly write them in **HDL**. Then, **Interconnect** them with the software part using the **IP integrator block design (BD)**.

*An IP is a reusable unit of logic, cell, or integrated circuit layout design. It's the hardware implementation of a function.*
Different workflows:

- Using the **same high level language** for both **hardware** and **software** allows you to compile a single file with an acceleration function that will be hardware. (e.g. Intel oneAPI DPC++, Spatial Lang)

- **Generate IPs*** from HLS tools (in C++) or directly write them in **HDL**. Then, **Interconnect** them with the software part using the **IP integrator block design** (BD).

*An IP is a reusable unit of logic, cell, or integrated circuit layout design. It's the hardware implementation of a function.
Example: Syfala implementation

Workflow:
- Generate a **FAUST IP** with HLS tools from C++ (generated with FAUST).
Example: Syfala implementation

Workflow:
- Generate a **FAUST IP** with HLS tools from C++ (generated with FAUST).
- Configure the **CPU** (baremetal or Linux).
Example: Syfala implementation

**Workflow:**

- Generate a **FAUST IP** with HLS tools from C++ (generated with FAUST).
- Configure the **CPU** (baremetal or Linux).
- Import a custom **VHDL I2S IP**.
**Workflow:**

- Generate a **FAUST IP** with HLS tools from C++ (generated with FAUST).
- Configure the **CPU** (baremetal or Linux).
- Import a custom **VHDL I2S IP**.
- Connect the IPs and the **memory**.
Example: Syfala implementation

Workflow:
- Generate a **FAUST IP** with HLS tools from C++ (generated with FAUST).
- Configure the **CPU** (baremetal or Linux).
- Import a custom **VHDL I2S IP**.
- Connect the IPs and the **memory**.
- Connect the **external ports**.
INTRODUCTION TO FPGA

Hardware/Software Co-Design

Example: Syfala implementation

Vivado Bloc Design
Conclusion
FPGAs are very powerful embedded platforms that offer unique features in the context of real-time audio DSP.

Sample-per-sample computation, high sampling rate, extremely low latency, large number of GPIOs allowing for direct interfacing with audio codec chips, etc.
FPGAs are very powerful embedded platforms that offer unique features in the context of real-time audio DSP. Sample-per-sample computation, high sampling rate, extremely low latency, large number of GPIOs allowing for direct interfacing with audio codec chips, etc.

But programming them is extremely complex and out of reach to non-specialized engineers as well as to most people in the audio community.
FPGAs are very powerful embedded platforms that offer unique features in the context of real-time audio DSP. Sample-per-sample computation, high sampling rate, extremely low latency, large number of GPIOs allowing for direct interfacing with audio codec chips, etc.

But programming them is extremely complex and out of reach to non-specialized engineers as well as to most people in the audio community.

The use of HLS tools has enhanced accessibility, but it still necessitates substantial knowledge in the field of microelectronics.
FPGAs are very powerful embedded platforms that offer unique features in the context of real-time audio DSP. Sample-per-sample computation, high sampling rate, extremely low latency, large number of GPIOs allowing for direct interfacing with audio codec chips, etc.

But programming them is extremely complex and out of reach to non-specialized engineers as well as to most people in the audio community.

The use of HLS tools has enhanced accessibility, but it still necessitates substantial knowledge in the field of microelectronics.

You will be able to program them at a very high level of abstraction using the Syfala Toolchain.
FPGAs are very powerful embedded platforms that offer unique features in the context of real-time audio DSP. Sample-per-sample computation, high sampling rate, extremely low latency, large number of GPIOs allowing for direct interfacing with audio codec chips, etc.

But programming them is extremely complex and out of reach to non-specialized engineers as well as to most people in the audio community.

The use of HLS tools has enhanced accessibility, but it still necessitates substantial knowledge in the field of microelectronics.

You will be able to program them at a very high level of abstraction using the Syfala Toolchain.

What’s next?
11h00: Introduction to Syfala Workflow (Pierre Cochard)
11h30: Hands-on: Connecting to Grid5000 and launching docker container (Syfala Team)
Appendix 1: Syfala Compilation Flow

- Faust.dsp
- Hardware.cpp (High-Level)
- Software.cpp (High-Level)

High Level Synthesis (HLS)

- RTL
- VHDL

FAUST Compiler

gcc

Baremetal/Linux

ARM

IP

FPGA

RTL synthesis/impl
**Dataflow** (assignments using logic expressions).
- How data flows through the system
- **Gate-level** implementation.
- **Concurrents** signals assignments statements.

```vhdl
entity half_adder is
port (a, b: in std_logic;
     sum, carry_out: out std_logic);
end half_adder;

architecture dataflow of half_adder is
begin
    sum <= a xor b;
    carry_out <= a and b;
end dataflow;
```
**Appendix 2: VHDL example with a half ladder**

**Dataflow** (assignments using logic expressions).
- How data flows through the system
- **Gate-level** implementation.
- **Concurrents** signals assignments statements.

```vhdl
entity half_adder is
port (a, b: in std_logic;
     sum, carry_out: out std_logic);
end half_adder;

architecture dataflow of half_adder is
begin
    sum <= a xor b;
    carry_out <= a and b;
end dataflow;
```

**Behavioral** (if then, case statements etc.).
- Most abstract style.
- One or more **process** statements.
- Each process is a single concurrent statement that contains **sequential** statements.

```vhdl
entity half_adder is
port (a, b: in std_logic;
     sum, carry_out: out std_logic);
end half_adder;

architecture behavior of half_adder is
begin
    ha: process (a, b)
    begin
        if a = 1 then
            sum <= not b;
            carry_out <= b;
        else
            sum <= b;
            carry_out <= 0;
        end if;
    end process ha;
end behavior;
```
Appendix 2: VHDL example with a half ladder

Dataflow (assignments using logic expressions).
- How data flows through the system
- **Gate-level** implementation.
- **Concurrents** signals assignments statements.

**Behavioral** (if then, case statements etc.).
- Most abstract style.
- One or more **process** statements.
- Each process is a single concurrent statement that contains **sequential** statements.

```vhdl
entity half_adder is
  port (a, b: in std_logic;
        sum, carry_out: out std_logic);
end half_adder;

architecture dataflow of half_adder is begin
  sum <= a xor b;
  carry_out <= a and b;
end dataflow;
```

**Structural** (instantiating primitive entities, e.g. logic gates gates and flip-flops).
Entity is described as a set of **interconnected components**.

```vhdl
entity half_adder is
  port (a, b: in std_logic;
        sum, carry_out: out std_logic);
end half_adder;

architecture structure of half_adder is begin
  component xor_gate
    port (i1, i2: in std_logic;
          o1: out std_logic);
  end component;

  component and_gate
    port (i1, i2: in std_logic;
          o1: out std_logic);
  end component;

  u1: xor_gate port map \
      (i1=>a, i2=>b, o1=>sum);
  u2: and_gate port map \
      (i1=>a, i2=>b, o1=>carry_out);
end structure;
```
void compute_fir(float* fTemp) {
#ifdef HLS
#pragma HLS array_partition variable=fTemp type=complete
//avoid dependence in the innerloop
   outer_loop: for (int i = 0; i < N; i++) {
#pragma HLS pipeline
//pipeline the outerloop
   inner_loop: for (int n = 0; n < BLOCK_NSAMPLES; n++) {
#pragma HLS UNROLL
//parallelize the innerloop ()
      fTemp[n] += samples[i+BLOCK_NSAMPLES-1-n] * coeffs[i];
   }
}
#endif
}