Computer Engineering

Introduction to Field Programmable Gate Arrays

Robert Trausmuth
Wiener Neustadt University of Applied Sciences
Summer 2006

Computer Engineering

Robert Trausmuth

- email: *robert.trausmuth@fhwn.ac.at*
- phone: +43.2622.89084.240
- fax: +43.2622.89084.99

- http://www.fhwn.ac.at/

Computer Engineering

Outline

- Introduction to FPGA
- FPGA history
- basic FPGA structure
- how to program FPGAs (VHDL)
- platform FPGAs
- on-chip bus system
- hardware/software codesign
- techniques for parallel processing

Computer Engineering

(traditional) design flow of logic circuits

- Determine the logic behavior (truth table)
- Simplify (Karnaugh maps)
- Select proper Integrated Circuits
- Develop Printed Ciruit Board
- Build prototype and test it

Technical limits through Large Scale Integration (LSI) and VLSI technique

Computer Engineering

Why are they called logic circuits?

- Logic: The study of the principles of reasoning.
- The 19th Century Mathematician, George Boole, developed a mathematical system (algebra) involving logic, Boolean Algebra.
- His variables took on TRUE, FALSE
- Later Claude Shannon (father of information theory) showed how to map Boolean Algebra to digital circuits:

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR
0	0	0
0	1	1
1	0	1
1	1	1

a	NOT
0	0
1	0

Computer Engineering

FPGA

- Field Programmable Gate Array
  - Control Logic Blocks (CLBs)
  - Can be combined as needed
  - Logic circuits are implemented as „program"

„Programmable Hardware"

Computer Engineering

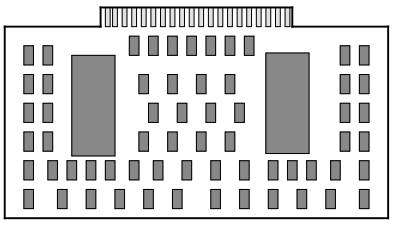
A brief history in time

- Logic circuits have been implemented in discrete elements (gates)
- Around 1960 a new idea was born:
  - Set up the gates and build controllable interconnections in between
- 1985 first FPGA-chip (XILINX)
- alternative: ASIC
  (Application Specific Integrated Circuit)

Computer Engineering

Why programmable logic?

- By the early 1980's most of the logic circuits in typical systems where absorbed by a handful of standard large scale integrated circuits (LSI).
  - Microprocessors, bus/IO controllers, system timers, ...
- Every system still had the need for random "glue logic" to help connect the large ICs:
  - generating global control signals (for resets etc.)
  - data formatting (serial to parallel, multiplexing, etc.)
- Systems had a few LSI components and lots of small low density SSI (small scale IC) and MSI (medium scale IC) components.



Computer Engineering

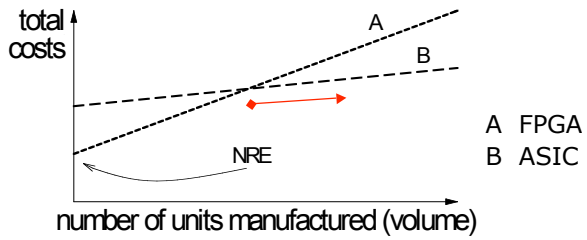
Why programmable logic?

- Custom ICs (ASICs) sometimes designed to replace the large amount of glue logic:
  - reduced system complexity and manufacturing cost, improved performance.
  - However, custom ICs are relatively very expensive to develop, and delay introduction of product to market (time to market) because of increased design time.
- Note: need to worry about two kinds of costs:
  1. cost of development, sometimes called non-recurring engineering (NRE)
  2. cost of manufacturing
  - A tradeoff usually exists between NRE cost and manufacturing costs

Computer Engineering

Why programmable logic?

- Custom IC approach only viable for products
  - very high volume (where NRE could be amortized),
  - not time to market sensitive.



Computer Engineering

Why programmable logic?

- FPGAs introduced as an alternative to custom ICs for implementing glue logic:
  - improved density relative to discrete SSI/MSI components (within around 10x of custom ICs)
  - with the aid of computer aided design (CAD) tools circuits could be implemented in a short amount of time (no physical layout process, no mask making, no IC manufacturing), relative to ASICs.
    - lowers NREs
    - shortens TTM
- Because of Moore's law the density (gates/area) of FPGAs continued to grow through the 80's and 90's to the point where major data processing functions can be implemented on a single FPGA.

Computer Engineering

Programmable Logic Devices

- PLDs
  - AND/OR matrix
  - Additional inverters and registers (flip flops)

  - PLA (AND free, OR free)
  - PAL (AND free, OR fixed)
  - PROM (AND fixed, OR free)

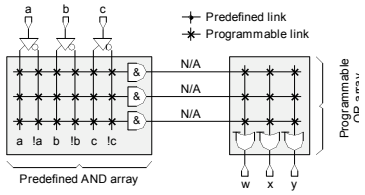
Computer Engineering

Programmable logic arrays (PLAs)

- Pre-fabricated building block of many AND/OR gates
  - Actually NOR or NAND
  - ”Personalized" by making or breaking connections among gates
  - Programmable array block diagram for sum of products form

Computer Engineering

Programmable logic arrays (PLAs)



Computer Engineering

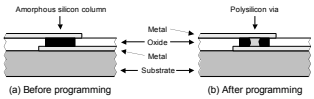
Programmable Logic

- Program (permanent) connections by
  - Breaking connections (blowing fuses)
  - Making connections (antifuses) *radiation hard!*
  - Writing EEPROM cells (rewritable)
  - Using RAM cells (volatile)
    - "booting" the chip
    - easy reconfigurable (even during operation!)

Computer Engineering

Programmable logic

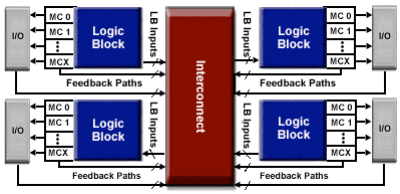
Technology	Symbol	Predominantly associated with ...
Fusible-link		SPLDs
Antifuse		FPGAs
EPROM		SPLDs and CPLDs
E²PROM/ FLASH		SPLDs and CPLDs (some FPGAs)
SRAM		FPGAs (some CPLDs)



Computer Engineering

Complex Programmable Logic Devices

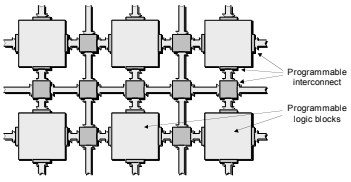
- CPLDs
  - Combination of a few PLAs and programmable interconnection lines
  - Good for fast logic implementation
  - Only few registers



Computer Engineering

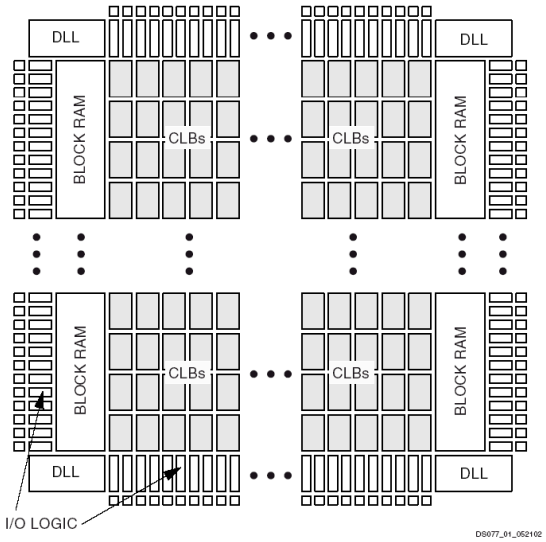
Field Programmable Gate Arrays

- FPGAs
  - Fine grained logic blocks
  - Special I/O blocks
  - Programmable switching matrix and lots of connection lines



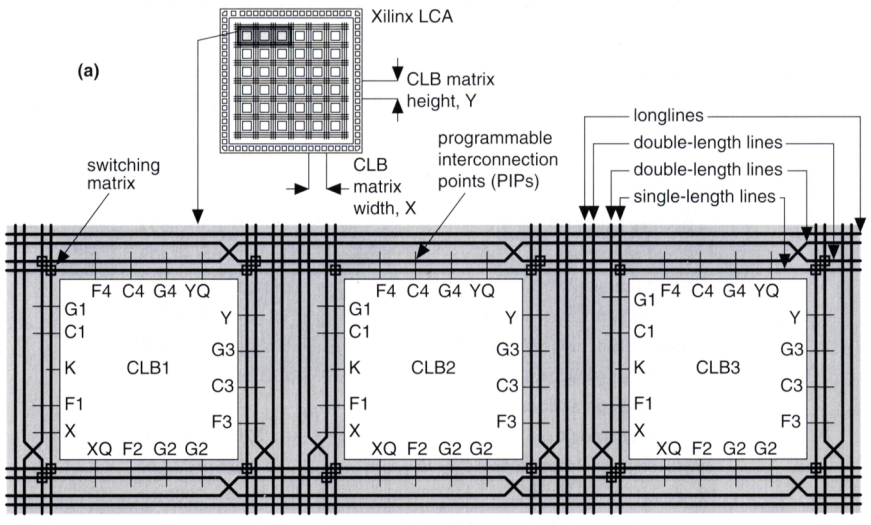
Computer Engineering

Example: Spartan II architecture

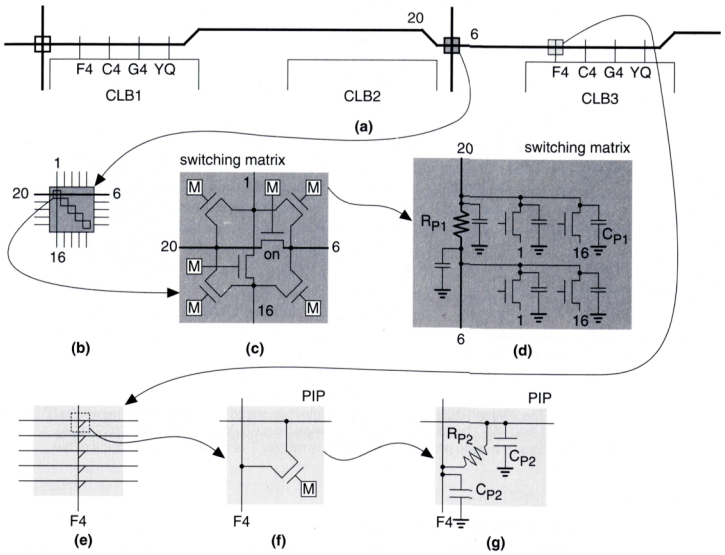


Computer Engineering

Switch matrix detail



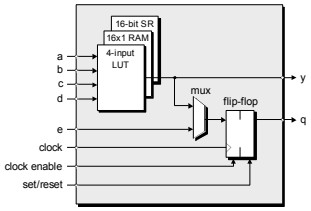
Switch matrix detail



Basic logic cell

- 16 bit shift register or
- 16 bit RAM or
- 4 to 1 lookup table

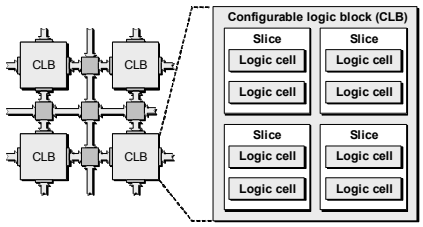
- multiplexer
- 1 bit register



Computer Engineering

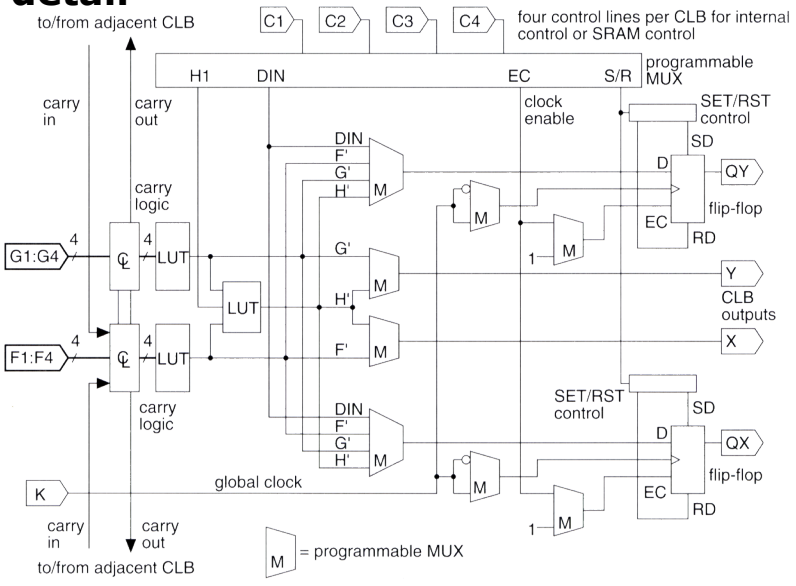
Control Logic Block

- Consists of several logic cells



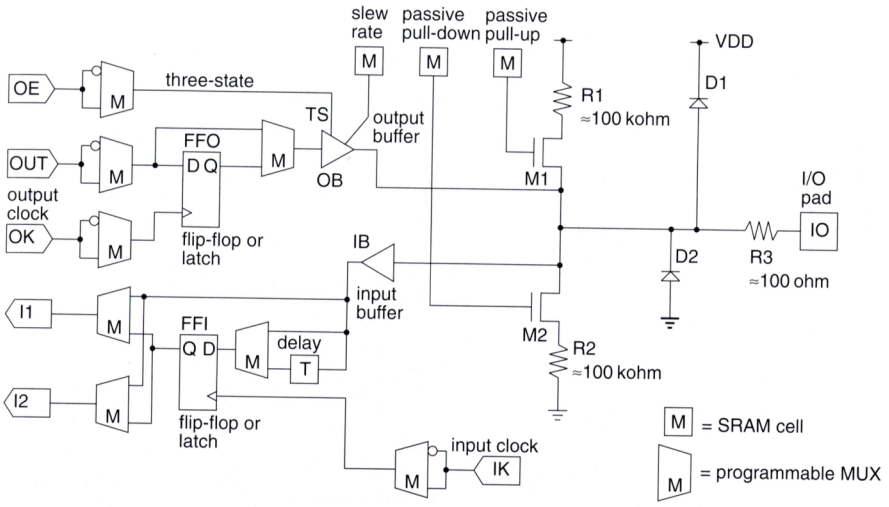
Computer Engineering

CLB detail



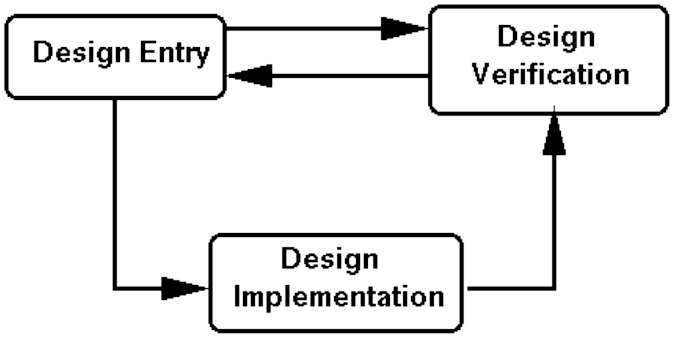
Computer Engineering

IOB detail



Computer Engineering

How to program FPGAs



Computer Engineering

FPGA design flow

- Design Entry:
  - Create your design files using:
    - schematic editor or
    - hardware description language (Verilog, VHDL)
- Design "implementation" on FPGA:
  - *Partition, place, and route* to create bit-stream file
- Design verification:
  - Use high language executable functional model (C/C++)
  - Use Simulator to check VHDL function
  - other software determines max clock frequency
  - Load onto FPGA device (cable connects PC to development board)
    - check operation at full speed in real environment.

Computer Engineering

The VHDL language

- Hardware Description Language (HDL)
  - High-level language for to model, simulate, and synthesize digital circuits and systems.
  - Not all VHDL statements are synthesizable!
- History
  - 1980: US Department of Defense starts Very High Speed Integrated Circuit program (VHSIC)
  - 1987: Institute of Electrical and Electronics Engineers ratifies IEEE Standard 1076 (VHDL'87)
  - 1993: VHDL language was revised and updated

Computer Engineering

Terminology

- Behavioral modeling
  - Describes the functionality of a component/system
  - For the purpose of simulation and synthesis
- Structural modeling
  - A component is described by the interconnection of lower level components/primitives
  - For the purpose of synthesis and simulation

Computer Engineering

VHDL concepts

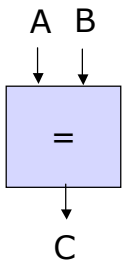
- Hierarchy
  - Self-contained components
- Abstraction
  - Behavioral description
  - Structural description
  - Data flow description
- Code Reuse
  - libraries

Computer Engineering

VHDL entity

- Black box from outside
- Only the interface is described

```
entity COMPARE is
    port (
      A, B: in std_logic;
      C: out std_logic
    );
end COMPARE;
```

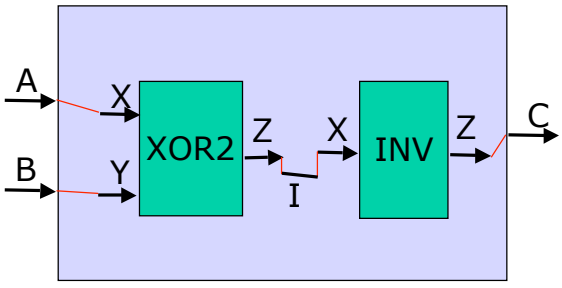


Computer Engineering

VHDL architecture

- Description of the implementation
- Local variables (signals) are allowed

- Structural
- Dataflow
- Behavioral



VHDL architecture – structural description

```
architecture ARCH_STRUCT of COMPARE is
  signal I: std_logic;

  component   -- previously defined somwhere else
    XOR2 port (X,Y: in std_logic; Z: out std_logic);
  end component;

  component   -- previously defined somwhere else
    INV port (X: in std_logic; Z: out std_logic);
  end component;

begin
  U0: XOR2 port map (X => A, Y => B, Z => I);
  U1: INV port map (X => I, Z => C);
end ARCH_STRUCT;
```


VHDL architecture – structural description

- Assemble your architecture out of already existing blocks
- You can use your own blocks for building large designs

- Many blocks are available
  - IP cores (intellectual property)

Computer Engineering

VHDL architecture – dataflow description

```
architecture ARCH_DFL of COMPARE is
begin
  C <= not (A xor B);
end ARCH_DFL;
```

- Description of the flow of data
  - Logic functions
  - assignments

Computer Engineering

VHDL architecture – behavioral description

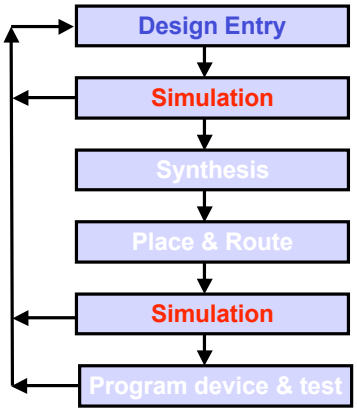
- Implementation is done with processes
  - Sequential execution of statements
  - All processes are run in parallel
  - Process is triggered by signal change

```
architecture ARCH_BHV of COMPARE is
begin
  mp: process(A, B)
  begin
    if (A = B) then
      C <= '1';
    else
      C <= '0';
    end if;
  end process;
end ARCH_BHV;
```

Computer Engineering

FPGA design flow

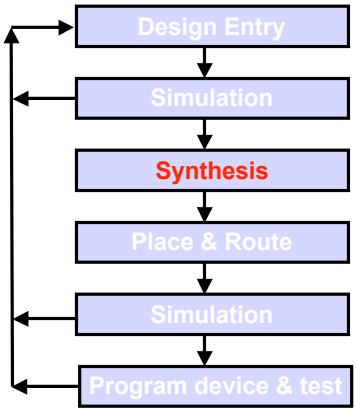
- Design entry
  - VHDL text
  - Schematic tool
  - State machine editor
  - Block diagram
- Simulation
  - Behavioral model (no timings, but basic functionality)



Computer Engineering

FPGA design flow

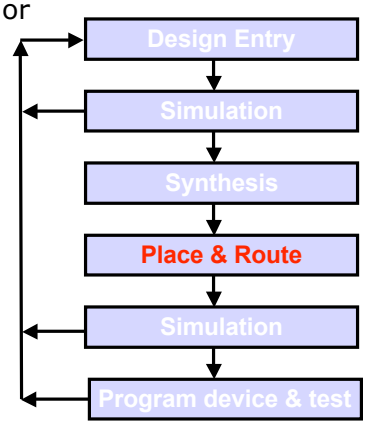
- Compilation & translation
  - Generates technology independent netlist
  - RTL schematic (HDL code analysis)
- Technology mapping
  - Mapping to technology specific structures:
    - **Look-up tables (LUT)**
    - **Registers**
    - **RAM/ROM**
    - **DSP blocks**
    - **Other device specific components/features**
- Logic optimization
  - Implementation analysis (technology view)



Computer Engineering

FPGA design flow

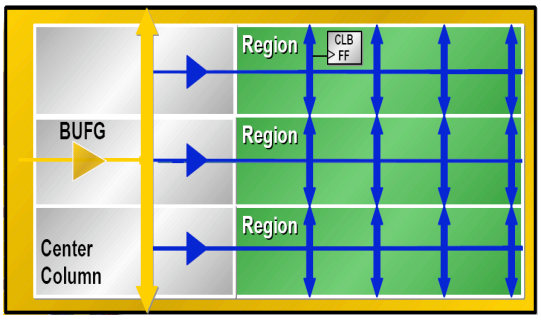
- FPGA fitter
  - Tools supplied by the FPGA vendor
  - Specific for each FPGA device architecture
- Functions
  - Place-and-route
  - Constraints editor
  - Backannotated netlist for timing simulation
  - Configuration bitstream



Computer Engineering

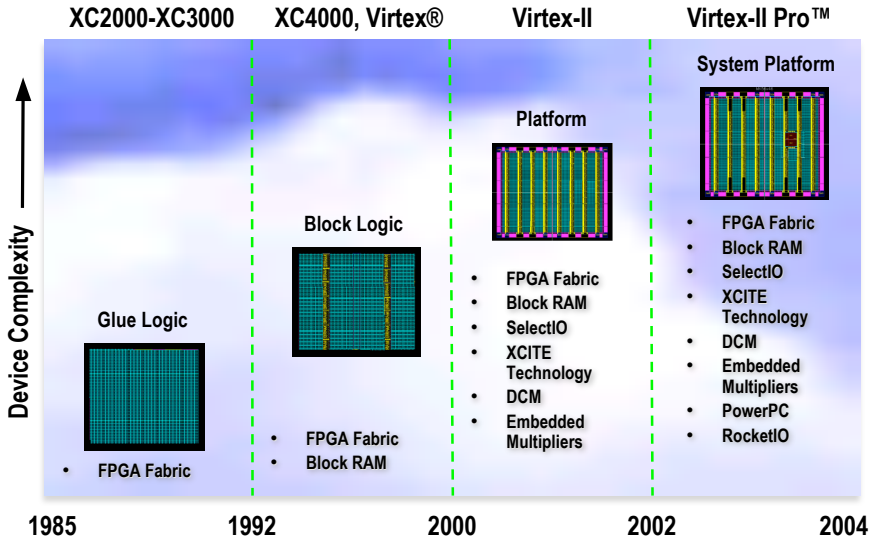
XILINX FPGA technology – clock network

- Signal is distributed via special lines and arrives at all CLBs at the same time
- Origin of clock signal is a BUFG
  - Driven by input pin
  - Driven by DCM
- 8 to 24 clocks



Computer Engineering

XILINX FPGA technology



Computer Engineering

Platform FPGAs

- additional building blocks in silicon
  - embedded multipliers
  - MAC-blocks for DSP functionality
  - digital clock managers (DCM)
  - embedded processor cores
  - high speed serial communication channels
  - co-processor access to hard core
  - Ethernet MAC network support

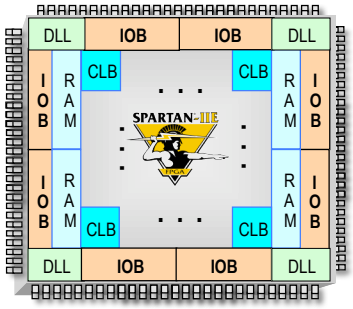
Computer Engineering

XILINX FPGA technology

- Spartan II e

I/O Connectivity
SelectIO™ Technology
Support major
I/O standards

Memory Resources
SRL16 registers
Distributed Memory
Block Memory
External Memory

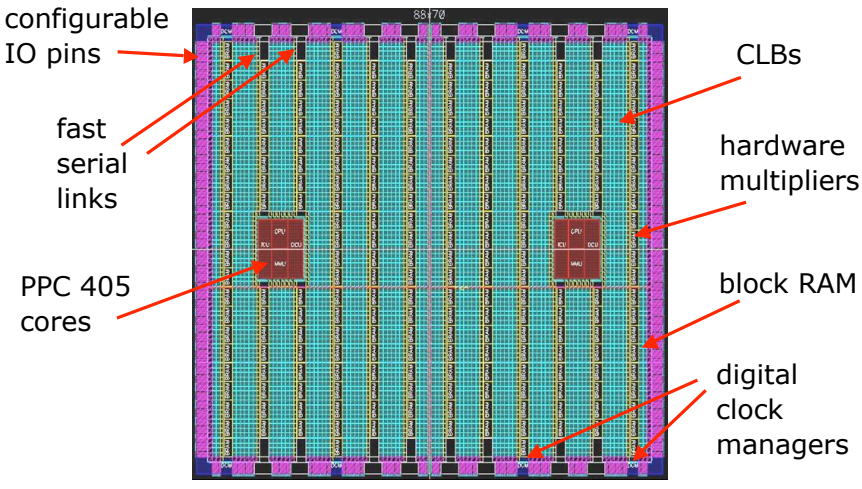


Logic & Routing
Flexible logic implementation
Vector Based Routing
Internal 3-State bussing

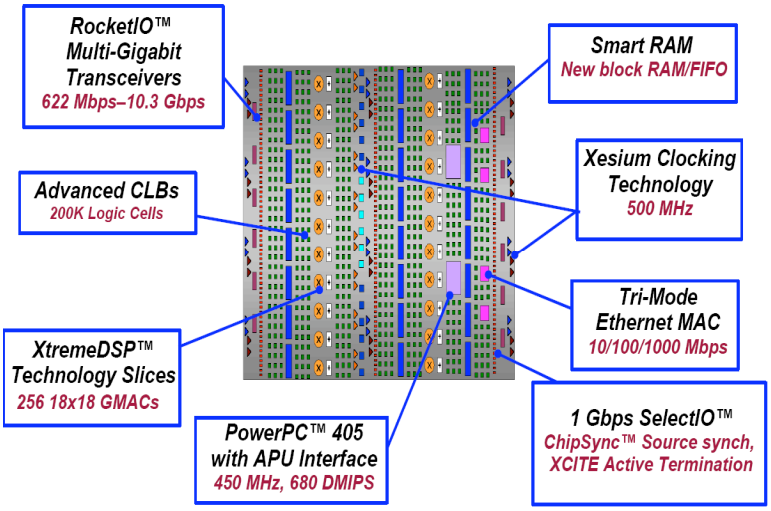
System Clock Management
Digital Delay Lock
Loops (DLLs)

Computer Engineering

XILINX FPGA technology – Virtex 2 Pro

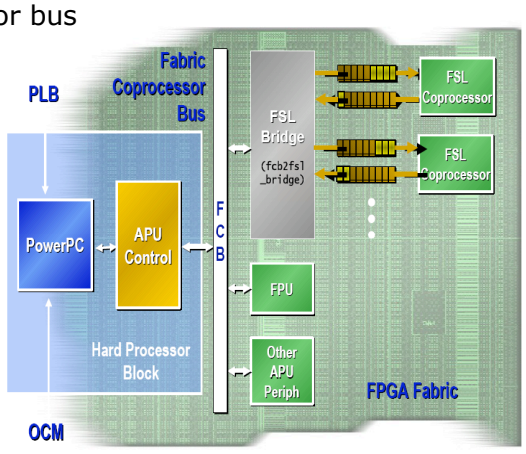


XILINX FPGA technology – Virtex 4



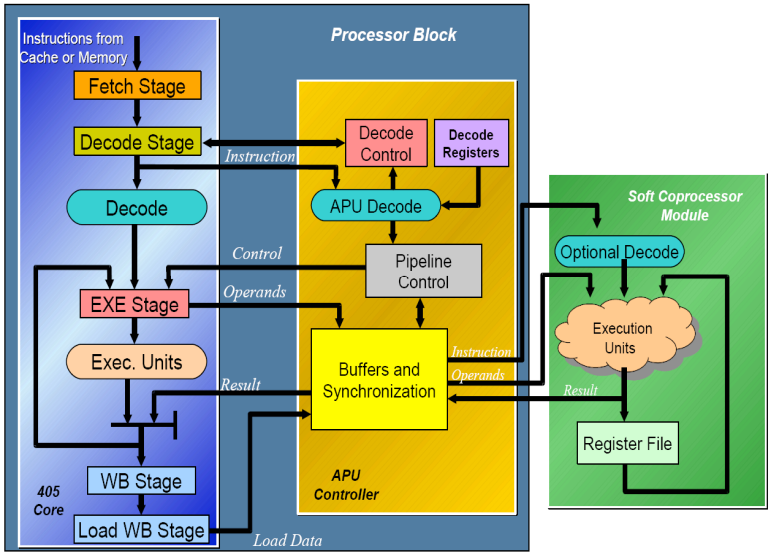
XILINX FPGA technology - Virtex 4

- Auxiliary Processor Unit
  - extends PPC instruction set
  - Fabric co-processor bus (FCB)
  - Can connect to multiple cores
  - Each decodes other instructions
  - Fast simplex link (FSL) connections are easy to do
  - But slower than direct FCB conn



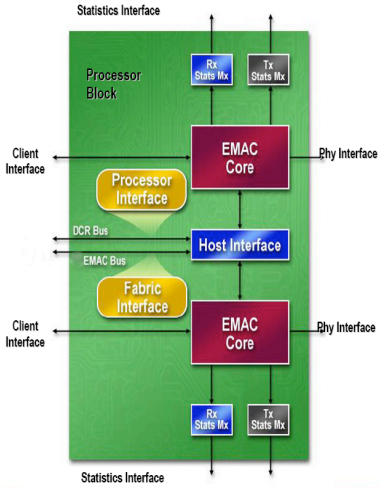
Computer Engineering

XILINX FPGA technology - Virtex 4



Computer Engineering

XILINX FPGA technology - Virtex 4



- Ethernet MAC
- Supports 10 Mb, 100 Mb and 1 Gb data per sec
- Phy interface off chip
  - electrical
  - optical

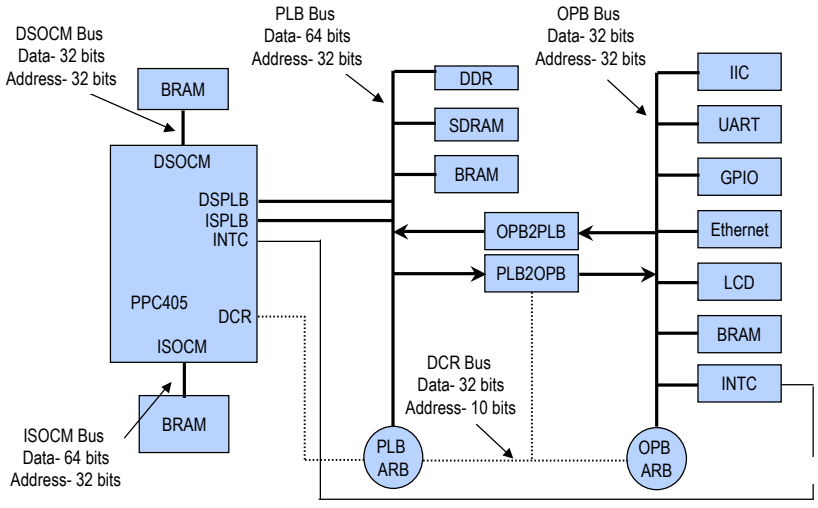
Computer Engineering

XILINX FPGA technology

- On-chip bus system
  - IBM CoreConnect Bus
    - PLB processor local bus
    - OPB on-chip peripheral bus
    - DCR device control register bus
  - further busses
    - OCM on-chip memory bus (instruction and data side)

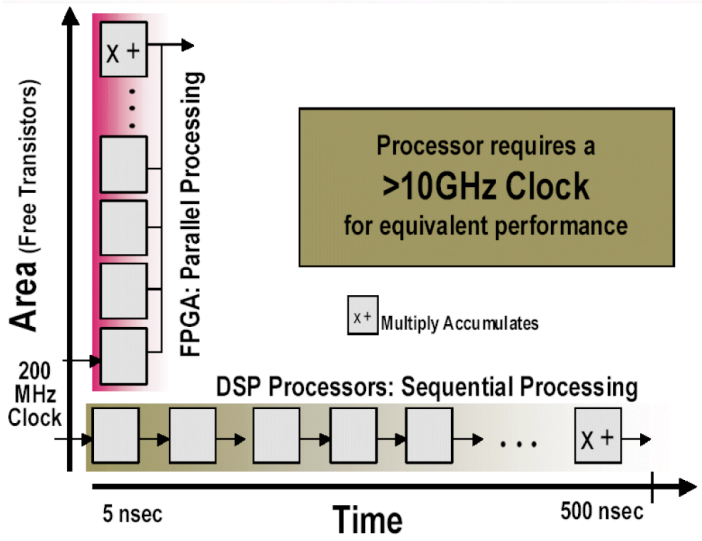
Computer Engineering

typical on-chip bus setup



Computer Engineering

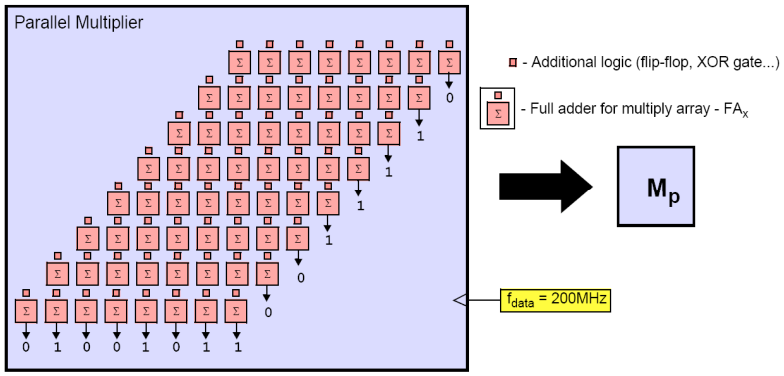
going parallel



Computer Engineering

parallel vs. serial

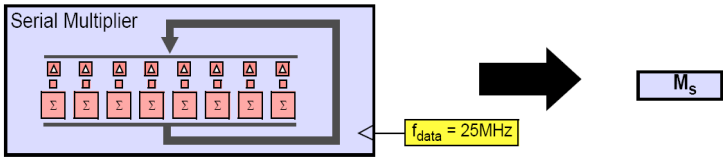
- 8x8 multiplication, 1 cycle => 200.000.000/sec with 64 full adders



Computer Engineering

parallel vs serial

- 8x8 multiplier, 8 cycles =>
- 25.000.000/sec with 8 full adders

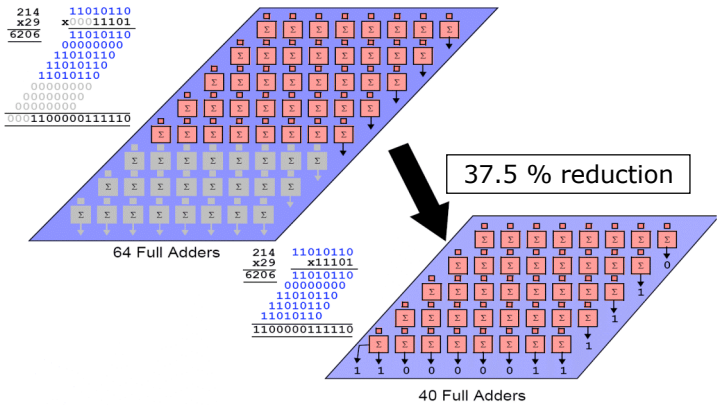


area x speed = const

Computer Engineering

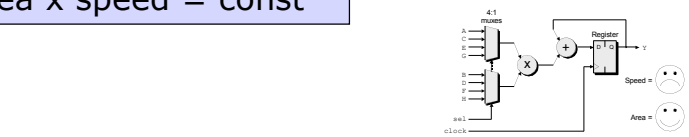
parallel vs. serial

- If we consider 5x8 sufficient (instead of 8x8 multiplication), we can save space

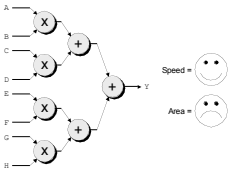


Example: digital filter processing

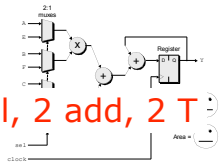
area x speed = const



1 mul, 1 add, 4 T



4 mul, 3 add, 1 T

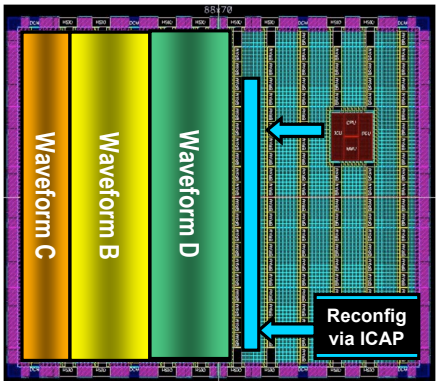


2 mul, 2 add, 2 T

Computer Engineering

Partially reconfigurability

- Newer devices support partially reconfigurability
  - "overlay technique" in hardware



Computer Engineering

Computer Engineering

System-on-chip design

- Fast functionality (ns to µs) is implemented in hardware
- Slow functionality (ms and slower) is usually done by software
  - generic
  - operating system
- System engineer has to decide which part of the solution is to be put into the fabric

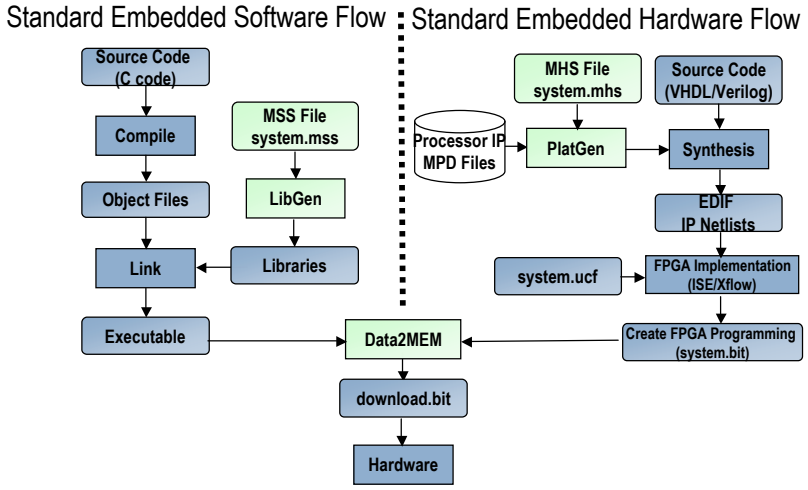
Computer Engineering

C to HDL tools

- Create C code
- Profile code
- Identify critical code segments
- Translate to VHDL
- Create stream interfaces to VHDL parts
- Create APU co-processor
- Replace critical code by using streams
- Co-processor uses CPU with own instruction sets

Computer Engineering

hardware/software codesign flow



Computer Engineering

hardware/software co-design

- Recent projects
  - Wavetable synthesizer
  - VHDL code generator + framework

Computer Engineering

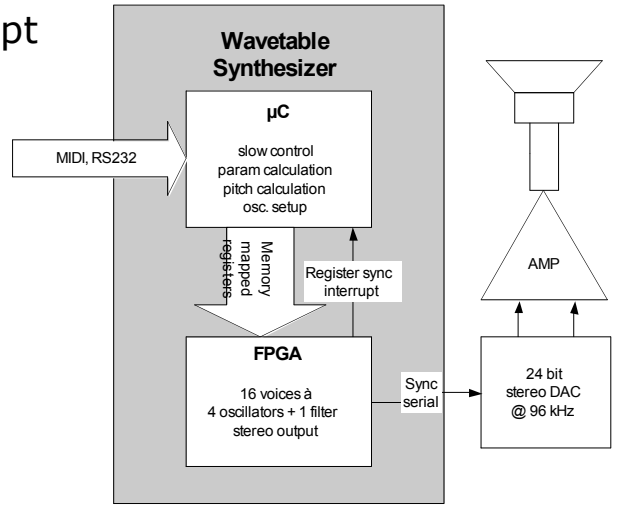
Wavetable Synthesizer

- Whole synthesizer in one chip
  - 16 voices with 4 generators each
  - One filter per voice
  - MIDI control
- Why SOC ?
  - Inherent parallelism of multiple voices and filters can be implemented in logic
  - Filter coefficient calculations and modulation parameters can be done by microcontroller
  - Communication to logic will be done via memory mapped registers and interrupts

Computer Engineering

Wavetable Synthesizer

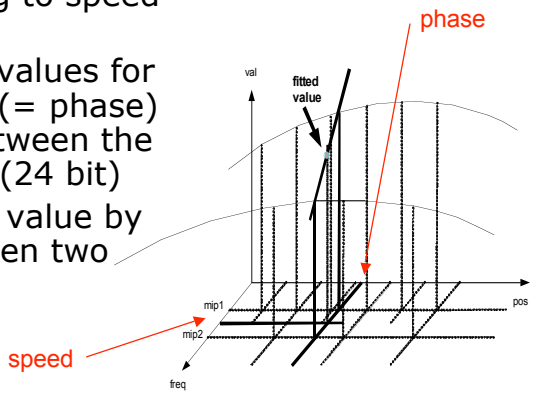
- The concept



Computer Engineering

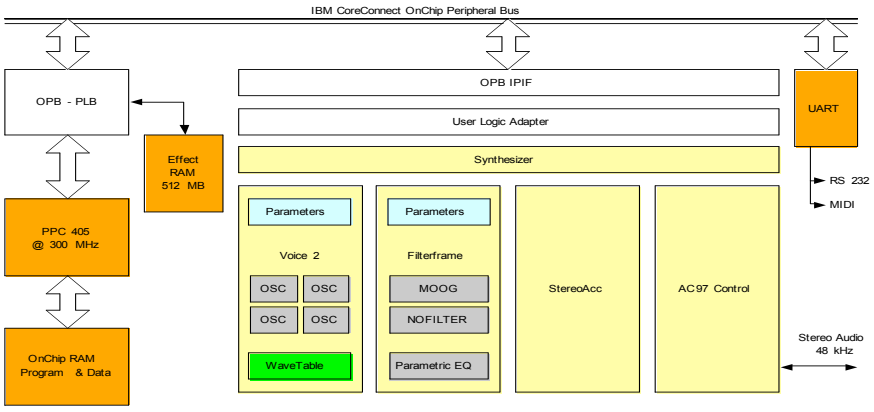
Wavetable oscillator

- Waveform data is 16 bit
- Determine two adjacent miptables according to speed (= frequency)
- Get two waveform values for the actual position (= phase) by interpolating between the two nearest points (24 bit)
- Get final waveform value by interpolating between two speeds (24 bit)



Computer Engineering

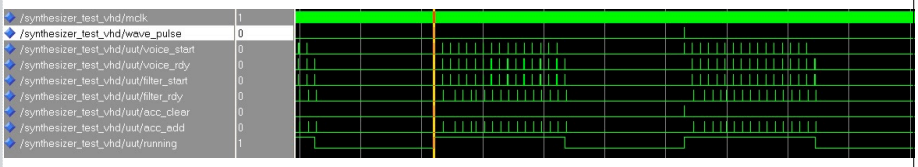
Wavetable Synthesizer Overview



Computer Engineering

Wavetable synthesizer pipelined

- One oscillator needs 21 cycles
  - 11 cycles for register access
  - 13 cycles for wavetable memory access
- One voice needs 66 cycles
- The filter needs 35 cycles (26 for Moog)
- One total run is done in 1083 cycles (2083 available @ 48 kHz)



Computer Engineering

Wavetable Synthesizer µC interface

- 8 direct registers for audio data send/return
- 2 memory regions for parameter data
  - Oscillator control (16 registers * 64)
  - Voice/filter control (32 registers * 16)
- Two interrupts
  - each frame (48000/sec)
  - each millisecond (1000/sec)

Wavetable Synthesizer sound demos

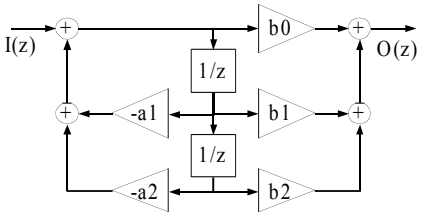
- LP-sweep (4 pole 24 dB/octave Moog)
- BP-sweep (4 pole, E3=-1,E4=2,E5=-1)
- Voice sample
- Strings sample

FAUST2HDL compiler

- FAUST is a programming language for real-time sound processing and synthesis.
- A process model is transferred into FAUST code. The compiler optimizes the calculations and produces C++ code.
- Instead of C++ code we produce VHDL code and utilize the possible parallel execution of calculations on the FPGA.
- We implement a full Harvard architecture with up to 8 bus systems and 8 ALU blocks.

Computer Engineering

FAUST2HDL compiler

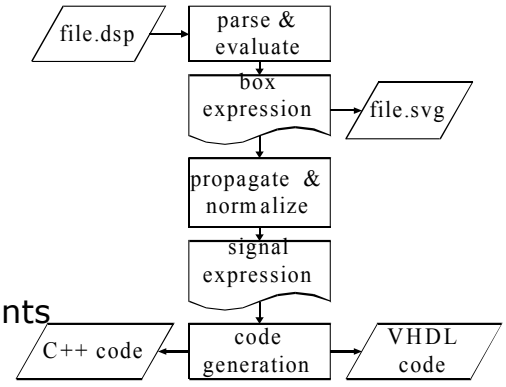



```
import("music.lib");

gain = vslider("gain", 0, 0, 1, 0.1) ;

filter(b0,b1,b2,a1,a2) = + ~ conv2 : conv3
   with
   {
        conv2(x)      = 0 - a1*x - a2*x';
        conv3(x)      = b0*x + b1*x' + b2*x'';
   };

process = filter(1.25,1.54,1,1.73,1.80) * gain;
```


Computer Engineering

FAUST2HDL compiler

- During the last stage, a pseudo assembly code is generated
- This code is optimized for parallel calculation and minimum resource requirements
- Then the VHDL file is generated



FAUST2HDL compiler

```
LOAD            LREG0       MREG0
LOAD            LREG1       MREG1
LOAD            LREG2       MREG2
LOADN           LREG3       CONSTF_0
LOAD            LREG4       MREG3
STORE           XREGR0_0    MREG3
ALU MUL         LREG3       LREG4       LREG5
LOADN           LREG6       CONSTF_1
ALU MUL         LREG6       XREGR0_0    LREG7
ALU ADD         LREG5       LREG7       LREG8
ALU SUB         INPUT0      LREG8       LREG9
MOVE            LREG9       XREGR0_0
STORE           XREGR0_0    MREG2
STORE           LREG2       MREG1
LOADN           LREG10      CONSTF_2
ALU MUL         LREG10      LREG2       LREG11
LOADN           LREG12      CONSTF_3
ALU MUL         LREG12      XREGR0_0    LREG13
ALU ADD         LREG11      LREG13      LREG14
ALU ADD         LREG1       LREG14      LREG15
ALU MUL         LREG0       LREG15      LREG16
STORE           LREG16      OUTPUT0
```

```
entity FAUST_process is
  port (
    -- input left / right
    INPUT0, INPUT1 : std_logic_vector(19 downto 0);
    -- output left / right
    OUTPUT0, OUTPUT1: std_logic_vector(19 downto 0);
    -- output left back, right back
    OUTPUT2, OUTPUT3: std_logic_vector(19 downto 0);
    -- output center / subwoofer
    OUTPUT4, OUTPUT5: std_logic_vector(19 downto 0);
  );
end FAUST_process;

architecture arch of FAUST_process is
  -- local registers
  -- memory block prototype
begin
  -- instantiation of dual port memory blocks

  -- FSM for local reg <-> memory interaction

  -- FSM for local reg <-> local reg interaction

  -- FSM for ALU op

end arch;
```

Computer Engineering

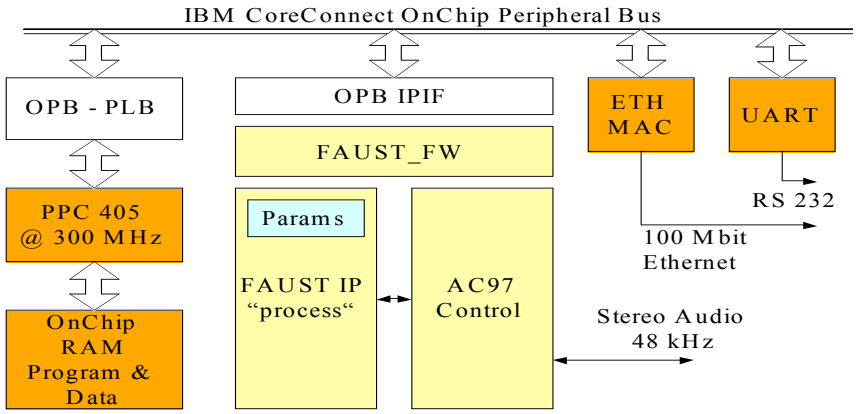
FAUST2HDL framework

- The VHDL file is part of a SOC framework
- The framework has been reused from the wavetable synthesizer
  - Interface to µC via memory mapped registers
  - Parameter calculations should be done in software
  - Sound processing is done in hardware
  - Stereo in, 5.1 out (AC97 protocol standard)

Computer Engineering

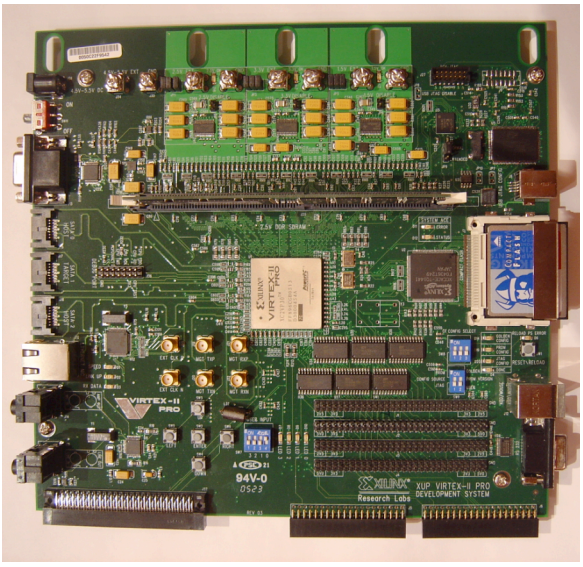
Computer Engineering

FAUST2VHDL code generator



Computer Engineering

XUP V2P board – the hardware



Computer Engineering