CENTER OF EXCELLENCE FOR ADVANCED AND INTELLIGENT CONTROL

HUSRB/1203/221/020
This document has been produced with the financial assistance of the European Union. The content of the document is the sole responsibility of Faculty of technical sciences, University of Novi Sad and can under no circumstances be regarded as reflecting the position of the European Union and/or the Managing Authority.
FPGA based digital control

Zoltan Kincses

2014.01.10.
Overview

1. The FPGA architecture in general
2. The Xilinx FPGA family
3. The Digilent Atlys prototyping board
4. The Xilinx Design Flow
5. System Generator for DSP
6. Implementing LMS adaptive filter using System Generator
1. The FPGA architecture in general
The FPGA architecture

- **LB**: The Logic Block contains LUTs (Look-Up-Table) which can be used to realize for example arbitrary multiple-input (4 or 6) single-output logic functions. The output of the LUTs can be connected to D-type flip-flops. The Logic Block can contains multiplexers, simple logic gates and interconnects.

- **IOB**: The Input/Output Block is the interface between the inner programmable logic and the output world. The Input/Output Block supports approximately 30 industrial standards (e.g. LVDS, LVCMOS, LVTTL, SSTL ...).

- **PI**: The inner components of the FPGA are connected to each other using the Programmable Interconnect.

- **DCM/CMT**: The Digital Clock Manager circuit is capable to modify the frequency and the phase of the input clock.
Logic Block

Input

Programmable logic network

Carry logic

Carry out

Flip-Flop

Clock

Memory cell

Output

Carry in
Programmable logic network

Memory cells

8-1 Multiplexer

Output

A B C

S2 S1 S0
Logic cluster

Cluster input

Carry logic

Carry in

Clock

Programmable logic network

Flip-Flop

Memory cell

Cluster output

Carry out
Programmable Interconnect

• Types of interconnects
  – **Local** interconnect for the connection of the elements of the cluster
  – **Global** interconnect for the connection of the clusters
    • **Island** (Xilinx)
    • **Cellular**
    • **Long-line** (Altera, Actel)
    • **Row** (Actel antifuse)

• Programmable interconnect implementation methods
  – **SRAM** (Xilinx, Altera)
  – **EEPROM/Flash**
  – **Antifuse** (Actel)
2. The Xilinx FPGA family
### Xilinx FPGA family

#### High performance
- **Virtex (1998)**
  - 50K-1M gate, 0.22μm
- **Virtex-E/EM (1999)**
  - 50K-4M gate, 0.18μm
- **Virtex-II (1999)**
  - 40K-8M gate, 0.15μm
- **Virtex-II Pro/X (2002)**
  - 50K-10M gate, 0.13μm
- **Virtex-4 (2004) [LX, FX, SX]**
  - 50K-10M gate, 90nm
- **Virtex-5 (2006) [LX, LXT, SXT]**
  - 65nm
- **Virtex-5 FXT, TXT (2008)**
  - 65nm
- **Virtex-6 LXT, SXT (2009)**
  - 40nm
- **Virtex-7 (2011)**

#### Low cost
- **Spartan-II (2000)**
  - 15K-200K gate, 0.22μm
- **Spartan-IIIE (2001)**
  - 50K-600K gate, 0.18μm
- **Spartan-3 (2003)**
  - 50K-5M gate, 90nm
- **Spartan-3E (2005)**
  - 100K-1.6M gate, 90nm
- **Spartan-3AN (2006)**
  - 50K-1.4M gate, 90nm
- **Spartan-3A - DSP (2006)**
  - 1.8M-3.4M gate, 90nm
- **Spartan-6 LX, LXT (2009)**
  - 45nm
- **Spartan-6 LX, LXT (2009)**
  - 45nm
- **Spartan-6 LX, LXT (2009)**
  - 45nm
- **Artix-7 (2011)**
  - 28nm
- **Kintex-7 (2011)**
  - 28nm

---

The project is co-financed by the European Union

---

IPA Cross-border Co-operation Programme

Good neighbours creating common future
High-performance Xilinx Virtex FPGA family resources (1998-2012)

BRAM memory (Kb): Virtex; (27 648) Virtex-II; (46 592) Virtex-II Pro; (99 216) Virtex-4; (9936K) Virtex-5; (18567K) Virtex-6; (38304K) Virtex-7; (67680K)

Logic Cells: Virtex; (73 008) Virtex-II; (46 592) Virtex-II Pro; (99 216) Virtex-4; (200 448) Virtex-5; (331 776) Virtex-6; (758 784) Virtex-7; (1,954,560)

Multiplier: Virtex; (168) Virtex-II; (444) Virtex-II Pro; (512) Virtex-4; (1056) Virtex-5; (2016) Virtex-6; (3360)

Xilinx Spartan-6 LX FPGA

General structure

The project is co-financed by the European Union
Xilinx Spartan-6 LX FPGA

CLB

• CLB – Configurable Logic Block
  – 2 Slice
Xilinx Spartan-6 LX Slice

- Three different types
  - SLICEL, SLICEM, SLICEX
- **SliceL** (25%) = as *logic*: 6-LUT, 8 D-FF, wide MUX, Carry Logic
- **SliceM** (25%) = as *memory*: SliceL + SRL-32x1, RAM-64x1 memory
- **SliceX** (50%) = as *basic slice* (only logic): 6-LUT, 8 D-FF
Xilinx Spartan-6 LX BRAM

• Configurable BRAM
  – Contains 2 independent 9Kbit BRAM
  – Configurable as
    • FIFO
    • RAM
    • ROM
  – Configurable as
    • Single port
    • Dual port
    • Quad port
Xilinx Spartan-6 DSP Slice

- DSP48A1 block (~250MHz)
  - 18x18bit signed 2’s complement multiplier
  - 18bit pre-adder
  - 48-bit dedicated MUX
  - 48-bit post-adder/subtractor

\[ P = C \pm (A \times (D \pm B) + CIN) \]
Xilinx Spartan-6 IOB

- Single-ended signals:
  - 3.3V low-voltage TTL (LVTTL),
  - Low-voltage CMOS (LVCMOS) 3.3V, 2.5V, 1.8V, 1.5V, 1.2V
  - 3V PCI @ 33 MHz / 66 MHz
  - HSTL I - III @ 1.8V (memory)
  - SSTL I @ 1.8V, 2.5V (memory)
- Differential signals:
  - LVDS
  - Bus LVDS
  - mini-LVDS
  - Differential HSTL (1.8V, Types I and III)
  - Differential SSTL (2.5V, 1.8V, Type I)
  - DDR, DDR2, DDR3, LPDDR support
Xilinx Spartan-6 CMT – Clock Management Tile

DCM – Digital Clock management

1 CMT = 2 DCM + 1 PLL

Number of CMTs: 4 – LX45

DLL: Delayed Locked Loop
- Phase shift: 0°, 90°, 180°, 270°
- Clock multiplication (M)/division (D) 1.5, 2, 2.5, 3, 4, 5, ..., 16
- 5 MHz – x100 MHz

DFS: Digital Frequency Synthesis
- Clock signal duplexing / halving
- Input/Output clock signal buffering
Embedded processors on Xilinx FPGAs

• „Embedded” soft-processor cores:
  – Xilinx *PicoBlaze*: 8-bit (VHDL, Verilog HDL sources)
  – Xilinx *MicroBlaze*: 32-bit (EDK support)
  – 3rd Party processor cores (HDL forrás)

• „Embedded” hard-processor cores:
  – IBM *PowerPC* 405/450 processor (dedicated): 32-bit
  – Only Virtex II Pro, Virtex-4 FX, Virtex-5/6 FXT FPGAs
3. The Digilent Atlys prototyping board
Atlys™ Spartan-6 FPGA prototyping board

- Xilinx Spartan-6 LX45 FPGA
- 128Mbyte DDR2 16-bit
- 10/100/1000 Ethernet PHY
- USB2 port (programming and data transfer)
- USB-UART and USB-HID port (mouse/keyboard)
- 2 HDMI video input and 2 HDMI output
- AC-97 Audio Codec
- Real-time power monitor
- 16MByte x4 SPI Flash (configuration and data storage)
- 100MHz CMOS oscillator
- 48 I/O (external connection)
- GPIO: 8 LED, 6 pushbutton, 8 switch
- 1 PMOD, 1 VMOD connector
PMOD – Peripheral modules

- **PMOD connector (12 pin): 2 VCC + 2 GND + 8 data**
PMOD modules

• PMODs for expansion
  – Character LCD, OLED, 7segLED
  – GPS transceiver, WiFi, Bluetooth,
  – Ethernet IF, USB-UART, RS232
  – Joystick, Rotary Enc., Switches,
  – SD Card, Serial Flash,
  – A/D, D/A converters, H-bridge
  – Accelerometer, Gyroscope,
  – Thermometer,
  – ...

The project is co-financed by the European Union
3. The Xilinx Design Flow (XDF)
FPGAs programing language:

- **I.** Traditional HDL languages:
  - a.) VHDL,
  - b.) Verilog

- **II.** C-based languages (C → FPGA synthesis):
  - a.) Impulse-C,
  - b.) Catapult-C,
  - c.) Handel-C,
  - System-C, Mitrion-C, ... (and ~10 other)

- **III.** Modell based languages:
  - a.) Matlab Simulink based System Generator,
  - b.) NI LabView (FPGA Module)
Design entry:
- HDL (.vhd)
- Schrmntic (.sch)
- State diagramm

Testbench
RTL simulation

Synthesis

Implementation
- Translate
- Map
- Place & Route

Bitstream generation
- .ngc / .edf
- .pcf
- .ncd
- .bit

FPGA

Static Timing Analysis

Timing simulation
Main steps of the XDF (I.)

• **1.) Modular or component based system design**
  – Design the HDL description, schematic, or state-diagram = design entry
  – Defining user-design constraints

• **2.) Simulation:**
  – every level of the system design
  – HDL testbench
Main steps of the XDF (II.)

• **3.) Synthesis and implementation:**
  
  – **Synthesis:** The HDL description transformed general gate—level components during the „logic synthesis” (e.g. logic gates, FFs)
  
  – **Implementation:** 3 main steps:
    
    • **TRANSLATE:** Merging more design files (maybe in different HDL language) into one netlist (EDF) file. The netlist contains the standard textual description of the components and their connections.
    
    • **MAP:** Technology mapping of the created „logic” design using the EDIF file created in the previous step. This process transforms the „logic” design into CLBs and IOBs.
    
    • **Placer & Route (PAR):** The previously created CLB and IOB design placed into real FPGA cells, and the connections between these cells are also created. The output of these process is an .NGC file.
Main steps of the XDF (III.)

• 4.) **Static timing analysis:** Determining the timing parameters (max. clock frequency, gate delay time, signal propagation delay...)

• 5.) **Bit-stream:** Generate FPGA configuration file (.BIT) and download it to the FPGA (the setup of the CLBs, and programmable interconnects is required in every startup, thanks to the SRAM technology used in the Xilinx FPGAs).
4. System Generator for DSP
Overview of System Generator for DSP

• The industry’s system-level design environment (IDE) for FPGA
  – Integrated design flow from the Simulink software to the BIT file
  – Leverages existing technologies
  – MATLAB, Simulink
  – HDL synthesis
  – IP Core libraries
  – FPGA implementation tools

• Simulink library of arithmetic, logic operators, and DSP functions
  – BIT and cycle-true to FPGA implementation

• Arithmetic abstraction
  – Arbitrary precision fixed-point, including quantization and overflow
  – Simulation of double precision as well as fixed point
Overview of System Generator for DSP

• VHDL and Verilog code generation for many Xilinx FPGA devices
  – Hardware expansion and mapping
  – Synthesizable VHDL and Verilog with model hierarchy preservation
  – Mixed-language support for VHDL/Verilog
  – Automatic invocation of the CORE Generator software to utilize IP cores
  – ISE project generation to simplify the design flow
  – HDL testbench and test vector generation
  – Constraint file (XCF), simulation DO file generation
  – HDL co-simulation via HDL C-simulation

• Verification acceleration by using hardware-in-the-loop through Parallel Cable IV,

• Platform Cable USB, and Network-based as well as Point-to-Point Ethernet connections
Model Based Design using System Generator

- Develop an executable spec using Simulink

- Refine the hardware algorithm using System generator
  - Verify hardware against executable spec
System Generator for DSP platform designs

- Simulink software verification
- HDL co-simulation verification
- Hardware Co-Simulation verification
System Generator based design flow

- Simulink software verification

The project is co-financed by the European Union
System Generator design-flow

- HDL Co-simulation verification
System Generator design-flow

- Hardware Co-simulation verification

The project is co-financed by the European Union
Interfacing with SysGen Design

- The Simulink environment uses a 64-bit 2’s complement “double” to represent numbers in a simulation.
  - Max/min: +/- 9.223 x 10^{18}
  - Resolution: 1.08 x 10^{-19}
  - Wide desirable range, but not efficient or realistic for FPGAs
- The Xilinx blockset uses n-bit fixed point numbers (2’s complement is optional)
- Thus, a conversion is required when Xilinx blocks communicate with Simulink blocks

The project is co-financed by the European Union
Gateway In

• The Gateway In block supports parameters to control the conversion from double precision to n-bit Boolean, signed (2’s complement), or unsigned fixed-point precision.
• During conversion the block provides options to handle extra bits.
• Defines top-level input ports in the HDL design generated by System Generator.
• Defines testbench stimuli when the Create Testbench box is checked in the System Generator block.
• Names the corresponding port in the top level HDL entity.
Gateway Out

- The Gateway Out block converts data from System Generator fixed point type to Simulink double
- Defines I/O ports for the top level of the HDL design generated by System Generator
- Names the corresponding output port on the top level HDL entity provided the option is selected
Data types

- **FIX** data type produces a signed 2’s complement number
- **UFIX** data type produces unsigned number
- When the output of a block is user defined, the number is further conditioned according to the selected Quantization and Overflow options.

**Quantization:**
- Truncate
- Round (unbiased: +/- Inf)

**Overflow:**
- Wrap
- Saturate
- Flag as error

---

Good neighbours creating common future

The project is co-financed by the European Union
**Boolean types**

- The Xilinx blockset also uses the type Boolean for control ports, such as CE and RESET.
- The Boolean type is a variant of the one-bit unsigned number in that it will always be defined (High or low).
  - A one-bit unsigned number can become invalid; a Boolean type cannot.
Floating-Point types

- **Floating-point Precision**
  - **Single**: Specifies single precision (32 bits)
  - **Double**: Specifies double precision (64 bits)
  - **Custom**: Activates the field below so you can specify the Exponent width and the Fraction width.
    - Exponent width: Specify the exponent width
    - Fraction width: Specify the fraction width
Creating a System Generator design

Create model and add new element

Start Simulink

The System Generator model in Simulink

The project is co-financed by the European Union
Creating a System Generator design

- Build the design by dragging and dropping blocks from the Xilinx blockset onto your new sheet.

Connect the blocks by pulling the arrows at the sides of each block.
Finding blocks

- The Xilinx blockset has eleven major sections
  - AXI4: FFT, VDMA
  - Basic elements: counters, delays
  - Communication: error correction blocks
  - Control Logic: MCode, black box
  - DSP: FDATool, FFT, FIR
  - Data Types: convert, slice
  - Index: all Xilinx blocks (a quick way to view all blocks)
  - Math: multiply, accumulate, inverter
  - Memory: dual port RAM, single port RAM
  - Shared memory: FIFO
  - Tools: ModelSim, resource estimator
Configuring your blocks

• Double-click or go to Block Parameters to view and change the configurable parameters of a block using multi-tabbed GUI

• Number of tabs and type of configurable parameters under each tab is block dependent

• Some common parameters are:
  – Precision: User defined or full precision
  – Arithmetic Type: Unsigned or twos complement
  – Number of Bits: total and fraction
  – Overflow and quantization: Saturate or wrap overflow, truncate or round quantization
  – Latency: Specify the delay through the block
Creating a System Generator design

SysGen Data Path and helper blocks

Gateway blocks used to interface between Simulink and SysGen blocks

Simulink Sources

Simulink Sinks

The project is co-financed by the European Union
System Generator Desing

Start simulation by pressing the play button

- All SysGen design must contain a System Generator block
  - Used to set global netlisting attributes

- Designs may have levels of hierarchy
  - Double click to “push” into a subsystem

The project is co-financed by the European Union
Sampling period

• Every System Generator signal must be “sampled”; transitions occur at equidistant discrete points in time, called *sample times*
• Each block in a Simulink design has a “sample period,” and it corresponds to how often the function of that block is calculated and the results outputted
• The sample period of a block *directly* relates to how that block will be clocked in the actual hardware
• This sample period must be set explicitly for:
  – Gateway In
  – Blocks without inputs
• The sample period can be “derived” from the input sample times for other blocks
System Generator Token
Setting the global sampling time

Sampling period = 1
System Generator token
Selecting compilation target

- Speed up simulation
  - Various varieties of hardware co-simulation
- Generate Hardware
  - HDL Netlist, NGC Netlist, Bitstream
- Analyze Performance
  - Timing and Power Analysis
System Generator token
Generating HDL code

- Specify the implementation Parameters
  - HDL Netlist as the compilation mode
  - Select the target part
  - Set HDL language
  - Set the FPGA Clock Period (in Clocking tab)
- Check Create Testbench
- Generate the HDL

The project is co-financed by the European Union
Hardware Co-simulation
Choosing compilation target

• Select the Co-simulation target hardware
Hardware Co-simulation

Design compilation

Press the generate button

Design automatically complied to produce bitstream

The project is co-financed by the European Union
Hardware Co-simulation

Run time co-simulation blocks
6. Implementing LMS adaptive filter using System Generator
LMS adaptive filters using System Generator

• Examples
  – How to implement LMS adaptive filter using System Generator
  – Determining the correct number of weights
  – Determining the correct step size
  – Dynamic channel characteristic
  – ECG adaptive filtering

• We would also like to thank for the Xilinx University Program